UNIVERSIDAD DE LA REPÚBLICA
FACULTAD DE INGENIERÍA

Instituto de Computación - InCo
PROYECTO DE GRADO

Extensibilidad de herramienta de gestión para VMware Cloud Director

Autores:

Aldo Díaz Betizagasti

Marcelo Sureda

Tutor: Gustavo Vázquez

Cliente: Pyxis - Ignacio Duarte

Resumen

El uso de servicios en la nube, en particular de *Infrastructure as a Service* (IaaS), está cada vez más extendido, lo que implica no sólo mayor cantidad de usuarios, sino también mayor cantidad de proveedores y ofertas. Los proveedores deben brindar mejor calidad en el servicio de base (virtualización, redes, disponibilidad), como también mejores servicios agregados que permitan a los usuarios mayor control, servicios anexos (respaldos, elasticidad de recursos, manejo de tareas), información (reportes de uso), entre otros aspectos, para sacar el mayor provecho de sus recursos de cloud. Los proveedores apuestan a un portal único de atención donde los usuarios puedan manejar todos los servicios habituales y anexos sin tener que acceder a varios sistemas alternadamente. Las plataformas de IaaS administradas por Pyxis y HG se basan en tecnologías VMware y su gestión para los clientes finales en el producto *VMware Cloud Director*. En las últimas versiones del producto se incorporaron APIs que permiten tanto el control y acceso a las acciones que se realizan a través de la página web como extender el producto con nuevas funcionalidades para que proveedores y usuarios puedan acceder a un único portal de servicios. VMware ha liberado diferentes APIs en cada una de las versiones del producto, generando que éstas se encuentren en diferente nivel de madurez, con una coherencia en interfaz muy baja y con una visión disjunta de los distintos accesos programáticos. Esto dificulta la implementación de extensiones y dificulta acompasar los tiempos de desarrollo con los del mercado. El plan del proyecto consiste en mitigar este problema desarrollando una nueva API que extienda las ya existentes, interactuando con ellas y ofreciendo un único punto de entrada a las funcionalidades. El sistema se debe integrar de forma natural y fluida con el producto estándar.

Se realizó un análisis del estado del arte enfocado en los productos mencionados anteriormente. A partir de ese punto, se acotaron las opciones a dos posibles alternativas de extensión de la plataforma. Una prueba de concepto, implementando un caso de uso simple con ambas alternativas y un análisis de las soluciones le permitió al equipo, en conformidad con el cliente, decantarse por una basándose en criterios establecidos. La alternativa preferida involucra un conjunto de tecnologías que se deben integrar entre sí y, a su vez, con la plataforma de VMware Cloud Director para conformar la solución deseada. Bajo este enfoque se define la arquitectura del sistema a implementar. Se relevaron requerimientos con administradores del sistema para identificar funcionalidades percibidas como necesarias pero de las que, actualmente, el sistema carece. Así se definen cinco nuevos casos de uso y se implementan, junto a un conjunto de funcionalidades base requeridas para los mismos (i.e., librerías de soporte). Este trabajo concreta el diseño en una implementación real, validando la arquitectura previamente delineada en la prueba de concepto. La implementación realizada materializa una plataforma con un conjunto de funcionalidades fundamentales para extender el producto de forma simple.

Palabras clave: Computación en la nube, IaaS, Proveedores de Servicio, VMware Cloud Director, Extensión APIs

Tabla de contenido

Introducción

Definición del trabajo

La utilización de servicios en la nube, en particular de *Infrastructure As A Service* (IaaS), está cada vez más extendido, lo que implica no sólo mayor cantidad de usuarios sino también mayor cantidad de proveedores y oferta. Por esta razón, para los proveedores, no sólo es importante brindar mejor calidad en el servicio de base (virtualización, redes, disponibilidad), sino también brindar mejores servicios agregados que permita a los usuarios control, servicios anexos (respaldos, elasticidad de recursos, manejo de tareas), información (reportes de uso) y otros aspectos que son relevantes a la hora de sacar el mayor provecho de los recursos de cloud contratados. Por esta razón, los proveedores apuestan a un portal único de atención en donde los usuarios no solo puedan realizar las tareas esperables en una plataforma *IaaS* (prender/apagar una VM, conectar una red, acceder a una consola), sino también manejar todos los servicios anexos sin tener que acceder a varios sistemas alternadamente.

Las plataformas de *IaaS* administradas por Pyxis (*Pyxis*, 2021) y HG (*HG*, 2022) se basan en tecnologías VMware, y su gestión, para los clientes finales, en el producto VMware Cloud Director (Fojta, 2022). Las últimas versiones del producto han tomado en cuenta lo dicho en el párrafo anterior y han desarrollando APIs que permiten no solo el manejo de las acciones que se realizan desde el producto a través de su portal de manera programática (principalmente el manejo del Datacenter Virtual), sino que también permiten ampliar el portal de autoservicio con el agregado de nuevas funcionalidades. De esa forma los proveedores pueden usar dicho producto como portal único de servicios. La estrategia de VMware se ha dado de manera parcial, desarrollando nuevas APIs a medida que se liberan versiones más recientes del producto. Esto hace que las distintas APIs se encuentren en diferente nivel de madurez, con una coherencia en interfaz muy baja y una visión disjunta de los distintos accesos programáticos. Por el mismo motivo, desarrollar es complejo y genera que los tiempos de liberación de las nuevas funcionalidades no se acompasan con los tiempos de mercado.

El proyecto consiste en desarrollar una API intermedia de más alto nivel que interactúe con las diferentes APIs del producto brindando un único punto de entrada programática a las diferentes capacidades del producto. Esta API tiene como finalidad facilitar la construcción de nuevas funcionalidades y la integración de otros sistemas con el portal de vCloud Director de manera de poder ampliar la oferta de los proveedores de Cloud con una visión unificada de uso a sus usuarios. Esta API no solo debe permitir cumplir con los requerimientos funcionales (permitir desarrollar nuevas funcionalidades), sino que debe también cumplir con requerimientos no funcionales como performance, concurrencia, seguridad y otros aspectos que son claves teniendo en cuenta el uso que tendrá el portal y la criticidad y confidencialidad de los datos que son manejados por el mismo.

Objetivos

Hay un conjunto de objetivos que se plantean para afrontar los desafíos descritos en la sección anterior:

- Investigar las diferentes APIs con las que cuenta el producto.
- Diseñar un nuevo conjunto de APIs que contemple las funcionalidades de las distintas APIs ya existentes pero pensando en el desarrollo de funcionalidades que deben ser ofrecidas al cliente final. Está API debe tener una visión unificada de las APIs que se encuentran por debajo de ella y debe permitir un desarrollo sencillo y ágil.
- Diseñar una arquitectura que permita implementar la capa intermedia de servicios cumpliendo tanto los requisitos funcionales como no funcionales del problema.
- Desarrollar una implementación de referencia de dicha API.
- Desarrollar al menos dos funcionalidades integradas al portal de vCloud Director utilizando esta API y así validar la arquitectura y la implementación de referencia realizada.

Los resultados esperados por el cliente dentro del alcance del proyecto son los siguientes:

- Relevamiento de APIs disponibles.
- Definición de una nueva API de mayor nivel de abstracción.
- Arquitectura de la capa intermedia que implemente las APIs definidas en el punto anterior.
- Implementación de referencia de la arquitectura definida: framework que dé soporte a las funcionalidades requeridas.
- Implementación de al menos dos casos de uso que validen la arquitectura e implementación de referencia utilizando el framework desarrollado.

Organización del documento

A continuación, un resumen de los temas tratados en cada uno de los capítulos que componen el documento.

En el siguiente capítulo, Contexto y Antecedentes, se presentan y definen conceptos generales sobre las tecnologías y la solución a desarrollar. En la sección sobre Investigación del Estado del Arte se presenta un detalle de la situación actual de las plataformas, productos y tecnologías utilizados por el cliente.

En el capítulo Prueba de Concepto, a partir del abanico de diferentes opciones disponibles presentadas en la sección de Investigación del Estado del Arte, se dan detalles de las pruebas de concepto realizadas con dos enfoques diferentes: *vRealize Orchestrator* y una extensión personalizada con lenguaje Python en el *backend* y el *framework Clarity* para el frontend. Hay una comparación entre ambas alternativas, incluyendo el análisis FODA[1] de cada una.

[1] Ver glosario. Análisis FODA: Fortalezas - Oportunidades - Debilidades - Amenazas.

En el capítulo Relevamiento de Casos de Uso, se detalla el procedimiento para relevar los requerimientos a implementar (entrevista con proveedores de servicio). Se da cuenta de los requerimientos relevados y luego se presenta, especificando el criterio de priorización, cuáles son seleccionados para implementar los casos de uso.

En el capítulo de Arquitectura se describe, a nivel general, el diseño de la arquitectura basado en la opción seleccionada a partir del análisis realizado en el capítulo Prueba de Concepto, y posteriormente refinada a partir de los requerimientos relevados en el capítulo Relevamiento de Casos de Uso. Para una descripción completa de la arquitectura de la solución desarrollada, se recomienda consultar el documento Documento de Arquitectura de Software (Díaz & Sureda, 2021a).

En el capítulo Implementación, se detalla el proceso de implementación del producto, la codificación y el testing. También se dan algunos detalles de las tecnologías utilizadas (información que es ampliada en uno de los anexos). Se incluye la documentación del código y unas guías de buenas prácticas para aquellos interesados en extender el core de la solución.

Finalmente, en el capítulo Conclusiones y Trabajo Futuro, se presentan las conclusiones sobre el trabajo realizado, qué fue lo que se consiguió de los objetivos originalmente planteados y se listan aquellos puntos en los que se podría mejorar el sistema, considerando también funcionalidades a agregar.

En los Anexos se hace enfoque en diferentes áreas para complementar la información disponible en el cuerpo principal del informe:

- Guía de referencia técnica para uso del framework
- Plan de testing
- Tecnologías utilizadas

Contexto y antecedentes

Contexto y marco conceptual

A continuación se describen algunas definiciones de conceptos básicos, herramientas y productos para contextualizar el marco de trabajo del proyecto.

Computación en la nube

(*IBM - ¿Qué Es Cloud Computing?*, 2021)
(*Microsoft - What Is Cloud Computing? A Beginner's Guide*, 2021)
Computación en la nube, o según el original en inglés *Cloud computing*, consiste en el suministro de recursos informáticos a demanda, desde aplicaciones hasta centros de datos, a través de Internet y con un modelo de pago según uso. Todo esto se logra mediante la virtualización de servicios computacionales. Sus principales características son:

- Flexibilidad o elasticidad en los recursos computacionales utilizados: se incrementan o reducen los recursos de forma rápida y sencilla para cubrir la demanda requerida por el usuario.
- Servicio medido de los recursos computacionales para que el usuario tenga la posibilidad de pagar de acuerdo al uso.
- Autoservicio (en inglés, self-provisioning) de los recursos computacionales que el usuario requiere.

La computación en la nube ofrece modelos de despliegue público, privado e híbrido.

Infrastructure as a Service (IaaS)

(*IBM - Infraestructura De IBM Cloud Como Servicio*, 2021)
(*Microsoft - What Is Cloud Computing? A Beginner's Guide*, 2021)
La infraestructura como servicio (*IaaS*) es el tipo de cloud computing más básico (las otras categorías, que no son abordadas en el alcance de este proyecto, son PaaS y SaaS). Le permite al usuario asignar los recursos de cómputo o procesamiento, red, almacenamiento, sistema operativo y seguridad bajo demanda. Permite escalar y reducir los recursos según sea necesario. El proveedor, habitualmente referido por su nombre en inglés (*provider*), es la compañía que ofrece una plataforma de servicios basada en infraestructura, aplicaciones y almacenamiento disponibles en la nube (Microsoft Cloud Service Provider, 2021).

VMware Cloud Director

(Fojta, 2021)
Anteriormente conocido como *vCloud Director for Service Providers*, *VMWare Cloud Director* es una plataforma cuya finalidad es operar y administrar la infraestructura y los recursos computacionales virtuales disponibles en la nube. Brinda a sus usuarios la sensación de acceder a centros de cómputos virtuales dedicados. Sus servicios están orientados a *providers* y a empresas como usuarios finales, llamados *tenants*. Al *tenant*, en la documentación y

literatura de VMware también se le denomina *org*. Es la organización cliente, o usuario directo del servicio de computación en la nube que es ofrecido por el *provider*.

Las siguientes entidades corresponden al entorno ofrecido a los *tenants* de VMware Cloud Director. Todas sus definiciones son tomadas de Bose (2021).

- ***Virtual Data Center***: En español, Centro de Procesamiento de Datos virtual, habitualmente abreviado como VDC. Ambiente que permite alojar máquinas virtuales, vApps, VMs, templates, etc.
- ***Virtual Machine***: En español, máquina virtual, generalmente abreviado como VM. Es la unidad básica del Virtual Data Center. Tiene asociadas una determinada capacidad de cómputo, memoria, almacenamiento y acceso a red. Se pueden crear VMs desde templates o nuevas desde cero así como también instalar un sistema operativo desde una imagen ISO.
- ***vApp***: Es un contenedor para almacenar VMs que operan juntas para correr una aplicación multicomponente. Los componentes de dicha aplicación se ejecutan en diferentes VMs. Las vApps de VMware suelen agrupar y administrar múltiples VMs que realizan tareas comunes. Se pueden crear templates de vApps para el rápido y sencillo despliegue de vApps y VMs. También es posible tener una vApp con una única VM.
- ***Catálogos***: Son usados para almacenar VMs, imágenes ISO de instalación, etc. Los usuarios los pueden utilizar para crear de forma sencilla ambientes con especificaciones e imágenes predefinidas para las nuevas vApps y VMs .
- ***Red de VDC - Org VDC network***: es la red de un Virtual Data Center de una organización (o *tenant)*, que está disponible para todas las vApps y VMs. La red del Virtual Data Center de la organización puede estar aislada, sin acceso a internet, o enrutada con acceso a internet.
- ***Red de vApp - vApp network***: Es una red que está disponible sólo dentro de una vApp para los componentes (VMs) de la misma. VMs de otras vApps no pueden acceder a la red de esta vApp. Este enfoque provee un nivel de separación adicional entre diferentes vApps. Cada red de vApp tiene su propia puerta de enlace para conectarse a la red del Centro de Procesamiento Virtual (VDC) de la organización.

Un grupo de servidores de VMware Cloud Director está conformado por uno o más servidores instalados en plataforma Linux. Cada servidor del grupo ejecuta un conjunto de servicios denominados la célula de VMware Cloud Director ("*cell*", según la documentación del inglés original). Todas las células comparten una única base de datos y almacenamiento del servidor de transferencia (*transfer server storage*). Cada célula de VMware Cloud Director se conecta a la instancia de vSphere y los recursos de red de los que depende para la parte de infraestructura. Para mayor información sobre la arquitectura, consultar la documentación disponible en la web de referencia.

vRealize Orchestrator

(*What Is vRealize Orchestrator (vRO)? | IT Automation Platform*, 2021)
vRealize Orchestrator es una moderna plataforma de automatización de flujos de trabajo que simplifica y automatiza complejas tareas de administración de infraestructuras de centros de

procesamiento de datos. Ofrece facilidades en cuanto a extensibilidad y agilidad en cuanto al despliegue de soluciones en la nube.

RabbitMQ

(Johansson & Vinka, 2020, pág. 17)
RabbitMQ es un software de cola de mensajes, usualmente denominado corredor de mensajes (*message broker*), manejador de colas (*queue manager*) o bus de mensajes (*message bus*). Es un software en el que se pueden definir colas y diferentes aplicaciones pueden conectarse para enviar y recibir mensajes a través de él. Esto permite la comunicación asincrónica para que las aplicaciones que producen y consumen mensajes interactúen con la cola en lugar de comunicarse directamente entre sí. Implementa el protocolo estándar AMQP (*AMQP Advanced Message Queuing Protocol*, 2021). Entre las opciones disponibles para extender la funcionalidad de VMware Cloud Director, RabbitMQ es utilizado como *middleware* de mensajería. El servidor de AMQP debe estar configurado para interactuar con VMware Cloud Director y éste, operando como productor de mensajes, los envía hacia RabbitMQ que, a su vez, los rutea a colas específicas. Los sistemas externos, operando como consumidores, se conectan al servidor de RabbitMQ escuchando de esas colas, recibiendo y procesando los mensajes enviados por VMware Cloud Director (Fojta, 2018, pág. 25). Algunos conceptos fundamentales para entender el funcionamiento interno de RabbitMQ son:

- ***Exchange***: Son agentes de RabbitMQ que aceptan el mensaje desde el proceso productor y lo rutea a una cola (*queue*) según la configuración de los *bindings* y los valores de *routing-keys*. Hay cuatro diferentes tipos de *exchange* en RabbitMQ. En el sistema de ruteo de extensión de VMware se utiliza el tipo más sencillo: *direct exchange*. El *direct exchange* rutea el mensaje recibido a la cola indicada en el *binding* que coincida con el *routing-key* del propio mensaje.
- ***Binding***: Es un enlace configurado para conectar el *exchange* con la *queue*.
- ***Routing-Key***: Es un atributo del mensaje, utilizado para identificar la cola de mensajes a la que se debe rutear el mensaje, según la configuración del *binding*.
- ***Queue***: La cola de mensajes es un buffer que recibe los mensajes desde el *exchange* y los disponibiliza para ser consumidos por el proceso consumidor.

Snapshot

(*VMware Inc. - Take a Snapshot of a Virtual Machine*, 2020)
En informática, una copia instantánea de volumen o *snapshot* (en inglés foto instantánea), es una instantánea del estado de un sistema en un momento determinado. Puede referirse a una copia real del estado de un sistema o de una capacidad que ofrecen los sistemas de copia de seguridad.

API

(*IBM - What Is an Application Programming Interface*, 2020)
API, del inglés *Application Programming Interface*, es una interfaz de software que brinda servicios a otros componentes de software. Un documento que describe cómo construir o usar

tal interfaz o conexión es una especificación de la API. Un sistema computacional que cumple con esa especificación se dice que implementa o expone esa API.

Angular

Angular es un entorno de trabajo enfocado en crear y mantener aplicaciones web de una sola página. Sigue un paradigma de diseño de tal forma que todo se construye a partir de entidades compuestas por una vista, estilos y un controlador, llamadas componentes. Su arquitectura base nativa separa completamente el nodo de backend con el del frontend (*What Is Angular?*, 2021). Es oficialmente mantenido por Google y es actualmente muy popular dentro de la comunidad, ofreciendo un sin fin de librerías de terceros que ofrecen herramientas útiles que agilizan y mejoran el desarrollo web.

Estado del arte

El proyecto inicia con la investigación de un conjunto de tecnologías y herramientas relacionadas con la extensión de VMware Cloud Director. Se busca, según el objetivo definido en el proyecto, analizar las opciones disponibles para extender la funcionalidad de las APIs del producto. El análisis comienza a partir de la información disponible en el sitio oficial de VMware Cloud Director (Fojta, 2021).

Algunas de las extensiones y herramientas disponibles para trabajar con VMware Cloud Director son las siguientes:

- ***Servicio de Kubernetes*** (*Container Service Extension (CSE)*, 2021)
  La extensión de servicio de contenedor (Container Service Extension, CSE) es una oferta de producto independiente de VMware que funciona junto con VMware Cloud Director. Mediante CSE, como proveedor de servicios, se puede ofrecer un servicio de Kubernetes (*Kubernetes Documentation*, 2021) a los *tenants*, lo que les permite implementar clústeres de Kubernetes totalmente funcionales en un modo seguro de varios *tenants* y de autoservicio. CSE extiende la interfaz de usuario y la API de VMware Cloud Director para la administración del ciclo de vida de los clústeres de Kubernetes. El administrador del sistema de VMware Cloud Director puede instalar y utilizar CSE. A partir de VMware Cloud Director 10.2, puede utilizar vSphere with VMware Tanzu en VMware Cloud Director para crear y administrar clústeres de Tanzu Kubernetes.

- ***pyvcloud*** (*Pyvcloud Documentation*, 2021)
  pyvcloud es el SDK oficial de Python de código abierto para VMware Cloud Director. Tiene soporte de VMware, Inc.

- ***vcd-cli*** (*Vcd-Cli*, 2021)
  vcd-cli es la interfaz de línea de comandos oficial para VMware Cloud Director. Permite a los administradores y *tenants* del sistema realizar operaciones desde la línea de

comandos permitiendo mayor comodidad y automatización. Es desarrollado y mantenido por VMware, Inc.

- ***Proveedor de VMware Cloud Director Terraform*** (*Terraform Registry*, 2021)
  Es una herramienta de lo que se conoce como Infrastructure as Code (*IBM - Infrastructure as Code*, 2019). Terraform es una infraestructura de código abierto como herramienta de software que proporciona un flujo de trabajo consistente, a través de línea de comandos, para administrar servicios en la nube. Terraform codifica las API de la nube en archivos de configuración declarativos. El proveedor de VMware Cloud Director se utiliza para interactuar desde Terraform con los recursos de VMware Cloud Director.

- ***Extensión de vRealize Operations Manager para VMware Cloud Director*** (*VRealize Operations Management Pack for vCloud Director*, 2020)
  VRealize Operations Management Pack para VMware Cloud Director es un módulo de administración adicional para vRealize Operations Manager (*VMware Inc. - What Is vRealize Operations?*, 2022). Está diseñado específicamente para VMware Cloud Director. Este módulo de administración monitorea el estado de las entidades compatibles de VMware Cloud Director y envía alertas inteligentes para los recursos monitoreados por el proveedor de servicios (*provider*). También ayuda en los casos de uso de planificación de la capacidad.
  vRealize Operations Manager Tenant App para vCloud Director es una aplicación orientada al *tenant* que proporciona a un administrador de la organización (org admin) visibilidad de su entorno de VMware Cloud Director. Los *providers* pueden permitir el acceso a la aplicación a cualquier *tenant*. Resuelve casos de uso de monitoreo, troubleshooting (resolución de problemas) y planificación de capacidad para un *tenant*. Esto está disponible sólo para *providers*.

- ***Complemento vRealize Orchestrator para VMware Cloud Director*** (*Using the vRealize Orchestrator Plug-In for vCloud Director*, 2021)
  El complemento *vRealize Orchestrator* se usa para ejecutar flujos de trabajo que automatizan los procesos de VMware Cloud Director. El complemento contiene un conjunto de flujos de trabajo estándar. También puede crear flujos de trabajo personalizados que implementen la API del complemento para automatizar tareas en el entorno de VMware Cloud Director.

Utilidad de extensiones relevadas

Se realiza una investigación de las diferentes herramientas y tecnologías listadas. Se llega a las siguientes conclusiones respecto a las posibilidades ofrecidas por cada una de ellas.

- **CSE:** Brinda posibilidades de extender VMware Cloud Director por medio de la integración con la tecnología de Kubernetes. Está completamente orientado a Kubernetes, y sólo permite extender las APIs en lo referente a la funcionalidad del

manejo de clústeres de contenedores administrados con Kubernetes. Tiene integración con vcd-cli.

- **pyvcloud:** Está en la base de otras extensiones y herramientas disponibles. Ofrece acceso directo a las APIs y posibilita su extensión de manera más flexible para quienes poseen conocimientos de programación en lenguaje Python. En el proceso de investigación de éste SDK, que sirve para programar lógicas complejas en directa interacción con VMware Cloud Director, se descubre otro SDK que permite extender el frontend, integrándose de forma natural con el portal existente del producto. Este SDK o framework, llamado vcd-ext-sdk (*vmware/vcd-ext-sdk*, 2021), está conformado por un conjunto de templates para crear extensiones de VMware vCloud Director, y que se pueden tomar como base para implementar nuevos servicios. Las plantillas aplican las mejores prácticas para el desarrollo de extensiones de API e UI. Se ofrecen en Java, Javascript, TypeScript, nodejs y Python. Son el punto de partida natural para crear una extensión al portal de VMware Cloud Director manteniendo el "look-and-feel" original. Para ello, se recomienda utilizar el mismo paquete de estilos del portal: Clarity (*Clarity Design System*, 2021). Conjugando estas dos tecnologías, pyvcloud en el backend y vcd-ext-sdk en el frontend, se abre un gran abanico de posibilidades para extender el producto.

- **vcd-cli:** Permite interactuar con VMware Cloud Director mediante línea de comandos en diferentes SO (Linux, OSx, Windows). Está basado en pyvcloud, y se puede extender programando en Python. Para ello se puede crear un branch del proyecto en Github, o generar contribuciones al proyecto existente. Para beneficiarse de vcd-cli, y sus posibles extensiones, el usuario queda limitado al uso de scripts con los comandos vcd-cli.

- **Proveedor de VMware Cloud Director Terraform:** Permite integración con una potente herramienta de IaC, como Terraform. Está fuertemente orientada al despliegue de nuevos entornos en VMware Cloud Director. No brinda posibilidades de extensión que ofrecen otras herramientas.

- **vRealize Operations:** ofrece potentes funcionalidades de monitoreo, control, y operaciones (*Operations Manager*). Le brinda significativas capacidades de control a los tenants sobre las entidades en sus data center virtuales.

- **vRealize Orchestrator:** está enfocado a la administración de flujos de trabajo y el complemento permite integración con VMware Cloud Director.

Un criterio aplicado para la selección es descartar aquellas herramientas enfocadas en una única tecnología. Es el caso de CSE que se enfoca en Kubernetes. Ésta es una herramienta de uso extendido pero que no está dentro del alcance de los despliegues con IaaS. También, por ese motivo, se descarta vcd-cli, que se encuadra en la creación y administración de la infraestructura mediante un lenguaje imperativo. El mismo caso aplica a Proveedor de VMware

Cloud Director Terraform. Encasillar el proyecto dentro de los márgenes de Terraform y una tecnología de *IaC* no cuadra con los objetivos iniciales.

Las dos últimas herramientas mencionadas, tanto vRealize Operations, como *vRealize Orchestrator* también están orientadas a fines muy específicos. De todas formas, se considera profundizar la investigación de *vRealize Orchestrator* por las posibilidades que ofrece para personalizar y extender la funcionalidad del producto estándar.

En una primera aproximación a las tecnologías y herramientas, la que parece brindar mayor nivel de flexibilidad para extender VMware Cloud Director es el SDK de Python pyvcloud para el backend, integrando con el framework vcd-ext-sdk para el frontend.

Habiendo arribado a estas conclusiones, se continúa el proyecto estudiando en mayor profundidad las posibilidades ofrecidas por estas dos alternativas:

- pyvcloud & vcd-ext-sdk
- *vRealize Orchestrator*

En el siguiente capítulo se presentarán los esfuerzos realizados en ese sentido, bajo el modelo de una Prueba de Concepto para cada una de esas herramientas.

Actualización sobre el estado del arte

En consideración a la duración del proyecto y que la investigación del estado del arte fue realizada al comienzo del mismo, se indagó nuevamente sobre las tecnologías utilizadas para saber sobre posibles cambios importantes sucedidos en este último tiempo. Nuevas herramientas o versiones de los productos pueden determinar nuevas posibilidades con respecto al diseño o implementación del producto actual.

El enfoque de la actualización está en los productos que fueron incluídos en la solución, a su vez de una pesquisa sobre posibles alternativas que hayan surgido.

Por un lado, una de las herramientas pilares de la solución, el SDK en Python *pyvcloud* para el VMware Cloud Director se encuentra en la misma versión desde que se comenzó el desarrollo de la solución de este proyecto, y en este último tiempo se incluyeron tres versiones menores que añaden principalmente arreglo de fallas y cambios en funcionalidades específicas no directamente relacionadas con el proyecto.

Es el caso análogo para vcd-cli, la interfaz de línea de comandos para VMware Cloud Director, sigue en la misma versión desde octubre del 2020 y han liberado solo una subversión.

El producto *vRealize Orchestrator* tuvo varias versiones nuevas liberadas, en las que se incluyen nuevas versiones de los paquetes que contiene el producto para aquellas funcionalidades principales que ofrece, nuevas funcionalidades de criptografía en base a un estándar en particular, nueva capacidad de integración con el producto vRealize Automation y

vRealize Automation Cloud, soporte de la funcionalidad de integración de vCenter Server con las últimas versiones de la API de vSphere y por supuesto una lista de reparación de fallas. En resumen, se concluye que los últimos cambios incluidos en este producto no afectan a la solución implementada.

En las versiones posteriores a la 9.7 inclusive de VMware Cloud Director, se incluye por defecto el complemento para incluir extensiones al portal de VMware Cloud Director. Esto facilita el proceso de instalación del front-end, puesto que en versiones anteriores primero se debía subir este complemento mediante un servicio de la API para luego poder subir el proyecto del front-end a través del mismo. Luego, a grosso modo, el resto de las actualizaciones realizadas en las últimas versiones de VMware Cloud Director son referentes a la integración con el suite de productos que ofrece VMware, actualizaciones de seguridad, mejoras en la interfaz de usuario entre otros cambios no directamente relacionados con la solución.

En conclusión, el software utilizado en el proyecto ha tenido pocos cambios menores en este tiempo y no particularmente relevantes para el uso que se le da en el proyecto. El único cambio relevante que se ha dado fue la inclusión nativa del paquete administrador de extensiones en el VMware Cloud Director, facilitando el proceso de liberación del frontend. Por ende, se mantiene el fundamento utilizado para la toma de decisiones en base a los factores recabados en el estado del arte.

Prueba de concepto

Una vez determinadas cuáles herramientas y tecnologías aplican al alcance del proyecto, se plantea realizar una prueba de concepto con cada una de ellas para poder comparar entre diferentes soluciones con el objetivo de seleccionar la que se tomará como base del framework a desarrollar.

Descripción del caso de uso

Para la prueba de concepto, y en acuerdo con el cliente, se determina implementar un caso de uso simple que consiste en agendar, para una hora determinada, el apagado de una máquina virtual (VM).

Para la prueba de concepto se presenta solamente el escenario de éxito para agendar el apagado de un VM.

Identificador	Agendar Apagado de una VM
Actor	System Admin // Tenant Admin (pyxisresearch) // Org/Tenant User (vappauth, usrvapp)
Descripción	El objetivo es fijar una hora para apagar una VM determinada.
Precondiciones	El usuario está autenticado en el sistema
Postcondiciones	Queda agendada la tarea de apagado para la VM seleccionada.

Secuencia de pasos

1. El usuario selecciona la VM
2. El usuario selecciona la opción de apagar la VM a una hora determinada
3. El usuario ingresa la hora en el sistema
4. El sistema registra la solicitud
5. El sistema inicia un timer para activarse a la hora ingresada
6. En la hora indicada por el timer el sistema se activa:
    a. Si la VM se encuentra encendida (powered on) entonces se realiza la operación de apagado (VM power off) y finaliza.
    b. Si la VM se encuentra en un estado diferente a encendida (por ej. apagada, suspendida, desplegada, etc.), entonces no se realiza ninguna acción y finaliza.

A continuación, en la figura 1, se presenta un diagrama del flujo que se debe seguir.

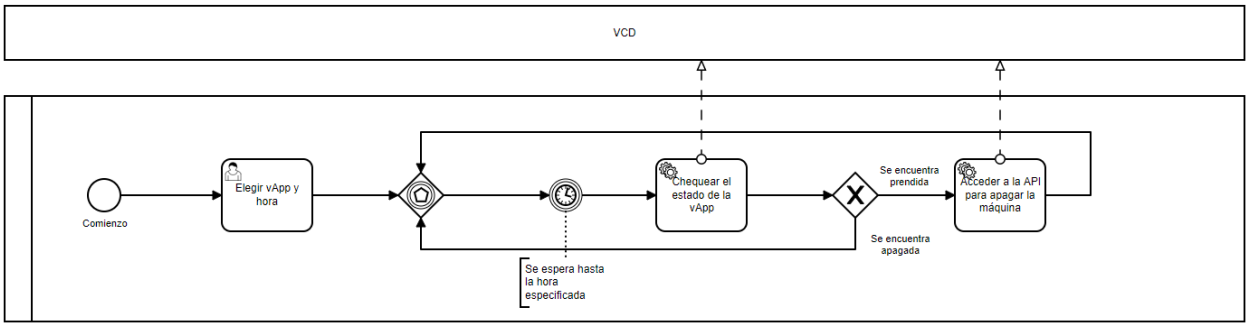

Fig. 1 - Flujo Caso de Uso Agendar Apagado de una VM

Enfoques

vRealize Orchestrator

El *vRealize Orchestrator* es un producto oficial de VMware que brinda un marco de desarrollo capaz de realizar un conjunto de tareas complejas en una secuencia arbitraria. Esta secuencia de tareas es denominada flujos de trabajos y puede ser empaquetada en instancias que luego son exportadas directamente al VMware Cloud Director. Permite a los usuarios administradores visualizar los flujos disponibles para su organización y, en base a un sistema de permisos, permitir que los usuarios directos puedan acceder y ejecutarlos.

Al ser un producto oficial, no solo tiene una integración resuelta con el VMware Cloud Director sino que también con otros productos de VMware como vRealize Suite y vCloud suite.

La idea inicial de utilizar este producto para la resolución del problema principal del proyecto se fundamenta en modelar las funcionalidades requeridas mediante el sistema de flujos de trabajo compuesto en tareas. Uno de los principales puntos requeridos para estas funcionalidades es la comunicación con el VMware Cloud Director, algo que se encuentra implementado e incluido en paquetes dentro del marco de trabajo que ofrece el vRealize Orchestrator.

Otro detalle interesante observado fueron las diversas herramientas que posee el producto para realizar la ejecución de procesos de manera asincrónica, tanto a nivel de paquetes como a nivel de tareas.

La prueba de concepto implementada en el *vRealize Orchestrator* consistió de un único flujo de trabajo que determinaba una lista de tareas que modelan el caso de uso de apagar una máquina virtual en un tiempo determinado. Para ello, se utilizaron herramientas de sincronización, como temporizadores, visualizadores temporales que retrasan el proceso hasta el momento adecuado, ejecución de un conjunto de instrucciones de código que implementan la lógica principal y finalmente componentes que, en base a los resultados obtenidos, retornan una respuesta acorde. Finalmente, al concluir y depurar el flujo, se liberó en el portal de VMWare correspondiente para el acceso a su ejecución.

pyvcloud & vcd-ext-sdk

Los conceptos profundizados en la solución del presente enfoque están detallados en el documento Architecting a VMware vCloud Director Solution for VMware Cloud Providers (Fojta, 2018), y los detalles técnicos de la implementación se encuentran en el documento Extensión VMware Cloud Director - Detalles de la Solución Técnica con pyvcloud & vcd-ext-sdk (Díaz & Sureda, 2021b).

Como punto de partida se toma de modelo un ejemplo disponible en la web (*Extending VMware vCloud Director Functionalities*, 2019) con su correspondiente repositorio en Github (*Groupe-Sii/lumext:*, 2020). Tomada del ejemplo disponible en la web, en la figura 2 se presenta la imagen de su diseño básico de arquitectura, incluyendo una representación del flujo de los mensajes en el sistema.

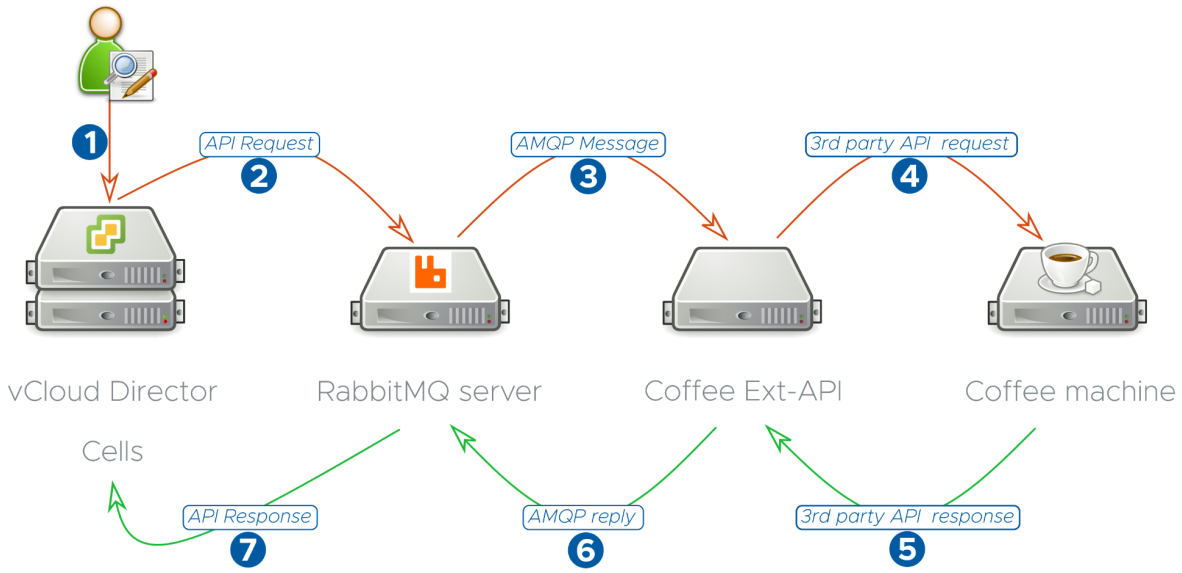

Fig. 2 - Flujo de mensajes en modelo de ejemplo
Tomado de Extending VMware vCloud Director Functionalities (2019)

En este diagrama aparecen, por primera vez a lo largo de nuestra investigación, algunas entidades fundamentales con las que se debe interactuar e incluir en el diseño de la arquitectura de la solución a implementar: las células de VMware Cloud Director y el administrador de cola de mensajes (RabbitMQ).

La descripción del flujo se describe a continuación:

1. El usuario inicia el requerimiento a través de la interfaz del portal (o directamente accediendo a la API expuesta por la extensión, por ejemplo, mediante cURL).
2. VMware Cloud Director formatea el requerimiento en un mensaje AMQP con una clave de ruteo específica y es enviado al servidor RabbitMQ.

3. El servidor backend de la extensión consume el mensaje de la cola definida en RabbitMQ, verifica que se cumplan prerrequisitos, realiza validaciones y contacta los sistemas externos.
4. Se envía requerimientos a través de APIs predefinidas al servidor externo (en el caso de este ejemplo particular, una máquina de café, aunque en la implementación real se trata de un servidor LDAP).
5. La máquina de café responde el requerimiento.
6. El servidor backend de la extensión procesa la respuesta enviada y es capaz de enviar la respuesta al requerimiento de VMware Cloud Director, a través de la cola indicada en el campo “reply-to” en el mensaje inicial de AMQP.
7. VMware Cloud Director puede consumir la respuesta del servidor de RabbitMQ y usar el contenido como respuesta de la llamada inicial del usuario a la API.

En la implementación del framework, y atendiendo a los requisitos del caso de uso descrito (ver sección Descripción del Caso de Uso), el servidor externo (la cafetera del ejemplo) y el servidor del backend de la extensión se aúnan en un único componente, ya que no hay otro sistema externo que intervenga en la solución. El único sistema con el que será necesario interactuar, además del servidor de RabbitMQ, es el servidor de VMware Cloud Director.

Considerando la necesidad de acceder al ambiente disponible de testing, conectándose a una VPN que aísla el acceso a internet, se configura una máquina virtual con SO Linux para poder conectarse a la VPN sin perder conectividad a internet en la máquina host. Esa misma máquina virtual se utiliza como plataforma de desarrollo y testing. Debido a la necesidad de desarrollar en diferentes lenguajes, se instalan un conjunto de herramientas que faciliten el proceso de desarrollo, como por ejemplo: Git, curl, Python con su “package manager” PIP, el software requerido para el frontend como NodeJS, NPM, Angular y Yarn. Los detalles de todas las herramientas y tecnologías utilizadas en esta prueba de concepto, con sus correspondientes versiones y una breve descripción de su instalación y uso se encuentran en el documento de referencia.

Como soporte de la configuración del backend, se utiliza un archivo en formato YAML que maneja un conjunto de pares clave - valor. La información que se almacena en el archivo de configuración y es leída por el proceso backend en el servidor al inicializarse es la siguiente:

- Información del servidor VMware Cloud Director: dirección IP o nombre del servidor, credenciales para autenticarse en el servidor, opciones de inicio de sesión y mínima versión de API aceptada.
- Opciones de la extensión: nombre de la extensión, namespace y patrón de la URI admitida por la extensión dentro del servidor.
- Información del servidor RabbitMQ: dirección IP o nombre del servidor, puerto TCP, credenciales para autenticación en RabbitMQ, *exchange* y *routing-key* o *queue* del servidor. Para una explicación completa sobre *exchange* y *routing-key* consultar el apartado de RabbitMQ en la sección de Marco Conceptual del presente documento, o la referencia en Architecting a VMware vCloud Director Solution (Fojta, 2018, pág. 25).

Como parte del despliegue de la solución, es necesario registrar la extensión con VMware Cloud Director. Las alternativas para realizar esta acción son las siguientes:

- Mediante un POST (por ejemplo, usando cURL o Postman) de un elemento “service” a la API Rest de vCloud Director. El método HTTP a usar es POST y el *endpoint* /api/admin/extension/service.
- Con funcionalidad de la librería pyvcloud. En este caso se puede desarrollar un script en Python para realizar el registro de la extensión. Está basada en la funcionalidad disponibilizada por el punto anterior.

Como trabajo adicional dentro de la prueba de concepto, se desarrolla un script en Python que realiza el registro de la extensión. La posibilidad de administrar el registro con un script facilita varias tareas como la integración con el servidor de RabbitMQ, desplegar la solución en el servidor y las futuras pruebas. El script, llamado *register_vcd_extension*, tiene su documentación técnica en la sección Módulo Register vCD Extension del documento *Extensión VMware Cloud Director* - Detalles de la Solución Técnica (Díaz & Sureda, 2021b, pág. 12). El script permite registrar una nueva extensión, eliminar la extensión previamente registrada y listar las extensiones disponibles en el servidor. Las diferentes funcionalidades son accedidas mediante opciones de línea de comando.

Para registrar la extensión se deben incluir en el requerimiento enviado al servidor las opciones de la extensión (nombre, namespace, y patrón de la URI), junto al *exchange* y *routing-key* del servidor de RabbitMQ. El proceso de registrar la extensión en el backend consiste, básicamente, en asociar el patrón de la URI de la extensión, expresado como una expresión regular, con un *exchange* y una *routing-key* en el servidor de RabbitMQ. En esta implementación la URI tiene la siguiente forma:

```
https://[vcd-host]/api/vApp/{vApp-id}/schedule/{hhmm}/shutdown
```

La expresión regular utilizada para representar el patrón del URI que extiende la API es:

```
/api/vApp/.*/schedule/(([01][0-9])|(2[0-3]))[0-5][0-9]/shutdown
```

El servicio de backend debe conectarse al servidor de RabbitMQ, escuchar en la cola correspondiente a esa routing-key y así podrá recibir y procesar los requerimientos que llegan desde el frontend.

Una vez definido Python como el lenguaje de programación a utilizar en el servidor, se debe optar por una librería que maneje mensajes AMQP y que actúe como cliente del servidor de RabbitMQ. Las principales alternativas disponibles son:

- Kombu: librería usada en el proyecto de Groupe SII, que se tomó como punto de partida para esta implementación (*Kombu Documentation*, 2019).
- PIKA: librería que implementa el protocolo AMQP y que se mantiene independiente del soporte elegido para la capa de red (*Pika Documentation*, 2017).

La opción elegida fue PIKA por las siguientes razones: es muy popular, es fácil encontrar ejemplos de su uso en internet, la documentación es muy completa y de buen nivel y, principalmente, es el cliente Python recomendado por el equipo de RabbitMQ (Johansson & Vinka, 2020, págs. 56 - 61).

El siguiente diagrama ilustra la interacción entre la extensión registrada en VMware Cloud Director, el *broker* RabbitMQ y el servicio de backend en Python, incluyendo el uso de la librería PIKA, con las operaciones realizadas en cada paso.

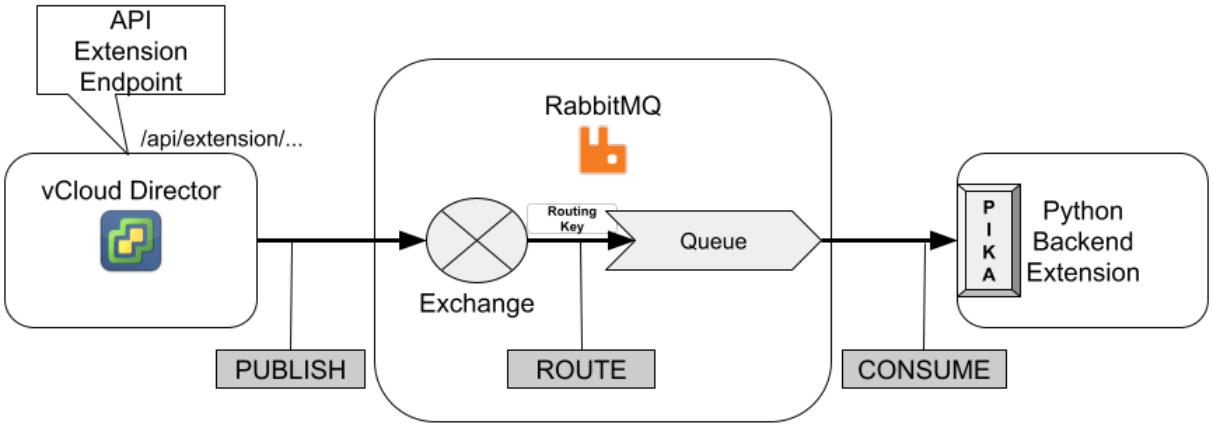

Fig. 3 - Arquitectura de Mensajes AMQP con RabbitMQ
Imagen basada en Fojta (2018, pág. 26, Fig. 10)

Una vez registrada la extensión para el backend, se debe subir al servidor el *plugin* del frontend. Dicho *plugin* contiene la aplicación frontend que correrá en la UI integrado al portal de VMware Cloud Director.

La funcionalidad del *upload* del *plugin* está disponible para los *providers* en la interfaz de usuario del portal de VMware Cloud Director a partir de la versión 9.7 inclusive, y también se mantiene la opción de hacerlo a través de las APIs del producto. Durante el proceso de subida, o más tarde, se puede publicar la extensión del frontend para *tenants* seleccionados, o si se prefiere, mantenerla disponible únicamente para el *provider*.

El componente del frontend es una sencilla aplicación web (es el *plugin* que debe subirse al servidor VMware Cloud Director) desarrollada en Angular, TypeScript, nodeJS y utilizando Clarity para los estilos, lo que permite integrarse visualmente con el portal del producto estándar (se mantiene el “look and feel”). Está conformada por un único módulo, dos componentes básicos y dos servicios que permiten cumplir con los cometidos simples del caso de uso en la interfaz del usuario. La interfaz permite las siguientes operaciones:

- Desplegar la lista de vApps disponibles en la organización del usuario.
- Seleccionar una de las vApps de la lista.
- Tomar el ingreso de la hora de apagado en formato hhmm.

- Enviar un requerimiento HTTP con el método POST al *endpoint* registrado previamente para la extensión de la API.

Partiendo del boceto de la arquitectura presentada para el proyecto de Groupe SII (*Groupe-Sii/lumext:*, 2020), que fue tomado de ejemplo, y tras el análisis realizado, se bosqueja un diseño preliminar de arquitectura, presentado a continuación en la figura 4.

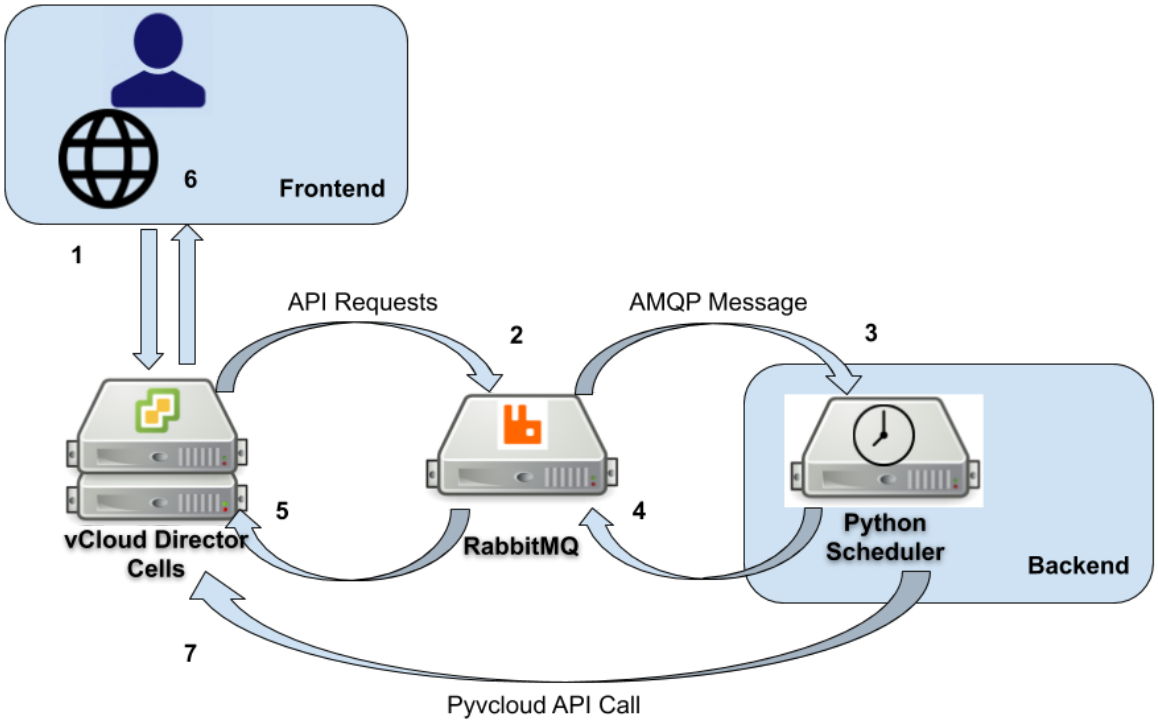

Fig. 4 - Boceto de la arquitectura para la prueba de concepto. Incluye flujo de mensajería.

El flujo representado se describe a continuación.

1. Desde el portal de la aplicación, el usuario accede a la página de la extensión y selecciona la operación para agendar el apagado de la VM seleccionada a la hora determinada. Se envía el mensaje con el requerimiento del frontend al *endpoint* correspondiente de la API de vCloud Director.
2. VMware Cloud Director realiza la conversión del mensaje al formato marcado por el estándar AMQP para ser enviado al servidor RabbitMQ. En este punto, VMware Cloud Director indica que el mensaje se rutea a través del *exchange* y *routing-key* asociados al *endpoint* que recibió el requerimiento y utilizados al momento de registrar la extensión.
3. El servidor de RabbitMQ disponibiliza el mensaje del requerimiento con el *routing-key* indicado y el servidor backend de Python lo captura a través de las primitivas ofrecidas por la librería Pika.
4. El servicio backend analiza y procesa el mensaje. Luego formatea y envía la respuesta al servidor RabbitMQ. Se debe indicar que la respuesta va al *exchange* y a la cola de

respuesta que fueron indicados en el mensaje recibido (se identifica a través de otra *routing-key*).
    a. Se agenda la actividad de apagado de la VM al horario indicado.
5. Se envía la respuesta del servidor de mensajería RabbitMQ al nodo de VMware Cloud Director.
6. Se envía la respuesta con el resultado del procesamiento del servicio backend, al frontend para que se presente la salida al usuario con el resultado de la operación.
7. Cuando llega la hora que fue agendada, se procesa la actividad de apagado de la VM invocando al servidor VMware Cloud Director mediante pyvcloud.
    a. Se llama a los métodos correspondientes de la librería.

Comparación entre los diferentes enfoques

Una vez finalizada la fase donde se implementó el mismo caso de uso con los dos enfoques, se realiza un análisis FODA (Fortalezas - Oportunidades - Debilidades - Amenazas) de cada uno de ellos.

Análisis FODA enfoque vRealize Orchestrator

Análisis interno

- ***Fortalezas***
    - Ofrece una suite completa de herramientas que permiten modelar una gran variedad de problemas
    - Es un producto oficial estable y mantenido por VMware.
    - Integra paquetes que permiten abstraer la integración con una amplia variedad de productos externos tanto de VMWare como de terceros. El caso más notable es el cliente que se integra con la instancia del VMware Cloud Director. También, clientes que permiten el acceso a otras plataformas en la nube, bases de datos SQL externas, entre otros.
    - Interfaz de usuario gráfica que facilita el uso de las herramientas mencionadas.
    - Funcionalidades básicas ya previstas y resueltas como sistema de usuarios, instalación “stand-alone” del producto, integración directa dentro del VCD, manejo de tareas programas, monitorización de flujos, entre varios otros.
- ***Debilidades***
    - Para llevar a cabo la explotación de todos los recursos que ofrece la plataforma se requiere pasar por una considerable curva de aprendizaje.
    - Presenta un riesgo al ser un producto relativamente nuevo que está en constante cambio.
    - Es un producto que requiere gestionar una licencia para la/las organización/es involucrada/s.
    - Implementación más árdua de requisitos que no sigan las pautas y estructura recomendadas de construcción del VRO.
    - Implementación de requerimientos que no se alineen con los principios básicos del framework implican un aumento de complejidad considerable que

posiblemente conlleve la implementación de “workarounds”, y requieran un mayor “know how” del producto. Por ejemplo, la necesidad de una funcionalidad particular que no se implemente en los paquetes nativos dedicados a ello. Los paquetes se utilizan como si fueran una interfaz y su implementación es abstraída del desarrollador.

Análisis externo

- ***Oportunidades***
  - Se alinea con el objetivo del proyecto de resolver problemas en VMware Cloud Director, siendo también un producto oficial hecho por VMware, pensado para trabajar con los productos que ofrece la empresa.
  - Ofrece una interfaz y sesión de usuario dentro del VMware Cloud Director para poder ejecutar los flujos de trabajo implementados en la plataforma.
  - Provee un sistema de permisos de usuarios manejados por el administrador de los tenants.
  - Al ser un producto establecido y de VMWare, ofrece seguridad al cliente a la hora de decidir su integración en la instancia de VMware Cloud Director de su organización.
- ***Amenazas***
  - Al no ser un producto popular o de código abierto, la cantidad de información por parte de la comunidad no abunda.
  - Siguiendo el punto anterior, la cantidad de información para aprender sobre el producto es en su mayoría por parte de una sola fuente: VMWare. El acceso a cursos completos y detallados presenta un costo.
  - El cliente puede no poseer y/o no querer contratar una licencia del producto para poder ejecutar las soluciones ya implementadas.

Análisis FODA enfoque pyvcloud & vcd-ext-sdk

Análisis interno

- ***Fortalezas***
  - El desarrollo para el backend en Python ofrece varias ventajas:
    - Es un lenguaje elegante, sencillo de aprender y comprender, con abundante documentación y ejemplos disponibles en línea.
    - Ofrece gran cantidad de herramientas, simplificando la solución de problemas, disminuyendo el tiempo de desarrollo e incrementando la productividad.
    - Es multiplataforma y portable. Se puede realizar el desarrollo sobre cualquier sistema operativo: Windows, MAC o Linux.
    - Ofrece muchas facilidades a la hora de integrarse con sistemas externos.
  - El SDK en el frontend se integra naturalmente con el entorno de VMware Cloud Director.
  - El enfoque descrito ofrece un gran nivel de flexibilidad.
- ***Debilidades***
  - El uso de Python también presenta desafíos:

        - Su popularidad genera que la gran cantidad de información disponible no siempre sea de calidad, esté actualizada o resulte útil.
        - La performance con múltiples hilos (“multi-threading”) no es buena en Python. Esto puede representar problemas a la hora de intentar escalar la solución.
    - Requiere interiorizarse de los pormenores de la librería Pika para interactuar con RabbitMQ.
    - Las APIs ofrecidas, tanto en backend como en frontend, por su extensión y complejidad, requieren un tiempo de análisis y estudio, para comprenderlas en profundidad y adquirir las destrezas y conocimientos necesarios para realizar el desarrollo.
    - Alto número de dependencias.

Análisis externo

- ***Oportunidades***
    - Ya se encuentran disponibles extensiones completamente funcionales que pueden ser extendidas, o tomarse como templates para desarrollar nuevas extensiones.
    - Este enfoque ofrece una gran flexibilidad para extender las APIs de vCloud, que es uno de los principales objetivos del proyecto.
    - Se considera que será sencillo encontrar programadores capacitados en la tecnología y lenguaje seleccionados.
- ***Amenazas***
    - A priori, se percibe que el desarrollo de extensiones más complejas y elaboradas pueden consumir mucho tiempo y resultar costosas en relación a otro tipo de productos que ya posean soluciones específicas para la nube (por ej. vRealize).

Conclusiones finales sobre la prueba de concepto

A continuación, se presenta una tabla comparando las características de cada uno de los enfoques adoptados.

	vRealize Orchestrator	**pyvcloud + vcd-ext-sdk**
Costo	Licencia (incluida con el producto)	Sin costo
Curva de aprendizaje	Pronunciada	Moderada
Estado inicial	Recursos ya disponibles	Desde cero
Flexibilidad	Depende del CU / Limitada	Alta
Información disponible	Limitada	Abundante

Integración con infraestructura	Muy buena	Depende del desarrollo / Requiere mayor esfuerzo

Los criterios que se decidieron priorizar para seleccionar una alternativa fueron los siguientes:

- Flexibilidad ofrecida por la tecnología
- Complejidad para desarrollar requisitos genéricos
- Dificultades a la hora de enfrentar la curva de aprendizaje

El análisis de ambas soluciones le permitió al equipo, en conformidad con el cliente, decantarse por pyvcloud + vcd-ext-sdk.

Relevamiento de casos de uso

Luego de definida la herramienta y realizada la prueba de concepto, se continuó con la etapa de análisis para lograr una descripción de los requisitos del cliente. A partir de varias reuniones con el mismo, se llegó a un requisito general de la herramienta que constaba de crear una solución que tenga la capacidad de poder resolver problemáticas varias asociadas con el uso del Cloud.

Para ello, se decidió proceder con el diseño de una solución que consta de un conjunto de herramientas básicas que sean comunes para futuras funcionalidades y que a su vez faciliten la implementación de las mismas. Por ejemplo, el desarrollo de un componente que sea capaz de navegar entre las relaciones de las entidades ya modeladas en el VMware Cloud Director, evitando así el uso directo de la API que ofrece.

Otro objetivo importante identificado para la etapa de diseño es tener en consideración el acceso y facilidad de incluir nuevas funcionalidades al sistema. Con el tiempo, la solución irá sufriendo iteraciones incrementales que incluirá la implementación de nuevos requisitos y es clave tener una estructura que considere la potencial degradación del software que acarrea cada incremento.

Entrevista con operadores de la plataforma

Para poder alimentar el *backlog* con las funcionalidades que se incorporarán en el framework a desarrollar, se coordinó una entrevista con profesionales dedicados y especializados en el uso del producto. El objetivo fue relevar requisitos con diferentes administradores de VMware Cloud Director para identificar aquellas funcionalidades percibidas como necesarias, pero de las que, actualmente el sistema carece. Luego, a partir de esas funcionalidades expuestas, se especificaron los requerimientos que les dan soporte y definen el framework. En resúmen, se debe identificar qué debe brindar el framework para poder implementar los casos de uso.

Antes de dar paso a la entrevista, se presenta brevemente la Prueba de Concepto desarrollada con la opción seleccionada: *pyvcloud* y *vcd-est-sdk*. De esa forma se brinda contexto a los entrevistados y se establecen las expectativas para que estén alineadas a los objetivos buscados.

Como preparación para la entrevista se elabora un cuestionario centrado en algunos tópicos clave:

- Ejecución y automatización de tareas
    - Planificación y enlazamiento de tareas simples.
    - Agenda de actividades. Programación de tareas para su ejecución en fecha(s) y hora(s) determinadas.
- Monitoreo y detección de eventos
    - Uso de recursos disponibles en el Data Center virtual.
    - Captura de métricas del sistema.

- Interfaz del sistema
    - Se apunta a identificar la preferencia por algún tipo determinado de interacción con el sistema.

Se detallan a continuación los principales puntos resaltados durante la entrevista:

- Relacionado a la automatización, se priorizan tareas diarias que apuntan al despliegue de una solución, con énfasis en la infraestructura. Por otro lado, los orgadmins, suelen estar más centrados en tareas de ajuste a partir de la capa del sistema operativo, por lo que aporta valor toda la información obtenida sobre métricas de la plataforma.
- Aportan valor las tareas programadas y automatizadas de apagado de VMs en un cierto orden. Por ejemplo, en aplicaciones multi-servidor, se considera buena práctica esperar a que se haya finalizado la aplicación antes de bajar el servidor de la Base de Datos.
- Se desea automatizar la extensión de VMs agregando capacidad de almacenamiento (discos).
- Se desea replicar la ejecución de tareas en varias organizaciones. Por ejemplo, apagar todas las VMs de los Data Centers virtuales de una lista de organizaciones.
- Las snapshots de VMs consumen una considerable cantidad de espacio. Se pretende detectar aquellas VMs con snapshots que no estén en uso, y en base a ciertos criterios, agendar su eliminación.
- Se desea contar con la posibilidad de realizar un reporte de recursos por organizaciones: Data Center virtuales, vApps, VMs, redes, IPs, etc.
- El auto-escalado de recursos es considerado de gran utilidad. Se debe monitorear el uso de recursos y reaccionar de forma automática para cubrir el incremento de demanda de trabajo. Se debe considerar escalado vertical (incrementar recursos de hardware de VMs) y horizontal (incrementar número de VMs que estén proveyendo el mismo tipo de servicio).

Otros puntos en los que se hizo menor énfasis, pero que fueron mencionados como características deseables son:

- Gestión de backups.
- Gestión de catálogos. VMware Cloud Director tiene templates, que son complejos de administrar, por lo que no se está utilizado en todo su potencial.
- Creación temporal de VMs para uso de única vez.
- Gestión de patches en VMs.
- La interacción con la plataforma se realiza a través del lenguaje de “Infrastructure as Code” (Terraform) o mediante el portal web, dependiendo del tipo de tarea.
- Gestión de eventos a través de un log de eventos, y que permite integrarse con sistemas existentes a través de notificaciones o alertas.

Respecto a requerimientos no funcionales se recomienda prestar atención a los siguientes puntos:

- Usabilidad
    - Integración natural con el portal, manteniendo el “look-and-feel” del producto estándar.

    - No generar la necesidad de aprender de cero un nuevo sistema: disminuir, dentro de lo posible, la complejidad existente en el acceso a las actuales herramientas de extensión.
- Disponibilidad
    - El servicio se debe mantener disponible de forma contínua e ininterrumpida.
- Seguridad
    - El framework no debe “romper” el nivel de seguridad existente ni generar vulnerabilidades en el esquema de acceso definido en la plataforma.
    - Se recomienda integración con el sistema existente de autenticación y autorización.
- Extensibilidad
    - El framework debe ser abierto para poder ser extendido.
- Rendimiento
    - Los tiempos de respuesta a los requerimientos enviados por el usuario deben ser aceptables desde el punto de vista operativo.

Casos de uso seleccionados

De la reunión surge una extensa lista de posibles requerimientos a cubrir. La lista fue depurada y los requerimientos priorizados con los siguientes criterios:

- Mantener aquellos requerimientos que, de acuerdo a quienes participaron en la reunión, son más críticos o agregan mayor valor a sus actividades.
- Mantener un número de requerimientos que fuera manejable dentro de los límites de tiempo del proyecto.
- Considerar su factibilidad en términos de nuestro conocimiento de la plataforma y del estado del arte.
- No superponer funcionalidades con otros productos de VMware en donde esos requerimientos ya estén cubiertos de forma nativa. Se considera además, la capacidad de integración con VMware Cloud Director, la sencillez de implementación, y costos asociados (licencias VMware, mantenimiento, curvas de aprendizaje, tiempos de desarrollo, etc.).

El proceso de selección continuó con una priorización de los requerimientos identificados basada en los criterios descritos. De esta forma, se definen cinco nuevos casos de uso a ser implementados, junto a un conjunto de funcionalidades base requeridas para los mismos (i.e. librerías de soporte). Los casos de uso seleccionados a partir del proceso realizado son los detallados a continuación.

1. Programación de secuencia de actividades

El requisito implica la programación de actividades con lógica agregada para capturar eventos, con la capacidad de interrumpir la ejecución de las actividades programadas si no se cumplen determinadas condiciones previas.
Un ejemplo, listar una secuencia de VMs que deben ser apagadas en determinado orden: primero el servidor Web, luego el servidor de aplicaciones y finalmente el servidor de base de

datos, para evitar dañar la BD por haber apagado el servidor durante el proceso de impactar una transacción en curso. En este caso, la condición sería ejecutar el apagado del servidor de la BD solo si el apagado del servidor de aplicaciones fue exitoso. El plan con el conjunto de tareas definidas puede ejecutarse de forma inmediata o programado para ser ejecutado a una hora y/o día determinados.
Este requisito implica varias funcionalidades que deben ser provistas por el framework:

- Agendar tareas
- Generar secuencia de tareas
- Capturar eventos
- Consultar estado de VMs
- Ejecutar acciones sobre las VMs (apagar, encender, reiniciar)

Este requisito debe enfocarse en el entendido de no pretender entrar en competencia con las funcionalidades nativas que ofrece *vRealize Orchestrator*.

2. Escalado de recursos

El escalado de recursos implica tomar una VM o un conjunto de ellas y a partir de los recursos disponibles y determinados parámetros, incrementar o disminuir la disponibilidad de los mismos. Los recursos considerados pueden ser CPU, memoria o disco.
Los parámetros pueden determinar la programación por horario o por día. Por ejemplo, ante el conocimiento de que en determinados días de la semana, como sábados o domingos, el consumo de determinado recurso es menor, se puede decrementar durante los fines de semana e incrementarlo de lunes a viernes.
Otra sugerencia consiste en basar el escalado en parámetros definidos según el consumo de un determinado recurso. En ese caso, el requerimiento implica la detección de determinado nivel de utilización de recursos en las VMs a través del resultado de sus métricas.
Los posibles sub-requerimientos que surgen de este planteo son:

- Extensión de discos en VMs
- Incremento/Decremento de capacidad/número de CPUs.
- Incremento/Decremento de memoria de VM.

Este requisito implica las siguientes áreas de funcionalidad a ser provistas por el framework:

- Agendar tareas
- Obtener métricas de consumo de recursos
- Ejecutar acciones sobre las VMs (apagar, encender, reiniciar)
- Acceder a recursos de VMs y modificar su configuración

3. Replicar acciones a varias organizaciones

El requisito implica tomar una acción y replicarla para varias organizaciones. Se deben definir acciones que sean replicables, ya que puede haber acciones que se ejecutan en un Data Center virtual que no sea posible ejecutar en otro debido a características particulares de cada uno de ellos, por ejemplo, apagar una VM con determinadas características, que no está disponible en otro ambiente.
Ejemplos posibles de acciones replicables son:

- Apagar todas las vApps/VMs de los Data Center virtuales de la organización.
- Encender todas las vApps/VMs de los Data Center virtuales de la organización.

- Agregar una vApp/VM con determinadas características a los Data Center virtuales de la organización.

Este requerimiento implica que el framework provea las siguientes áreas de funcionalidad:

- Acceder a las Organizaciones disponibles
- Acceder a los Data Center virtuales
- Ejecutar acciones sobre las VMs (apagar, encender, reiniciar)

4. Eliminación de Snapshots obsoletas

El requisito implica la eliminación, mediante el agendado de actividad, de snapshots que hayan permanecido en el sistema por más de un tiempo determinado. El proceso de eliminación puede agendarse para ser ejecutado de manera diaria o semanal. Tanto el período de ejecución como el tiempo de permanencia de las snapshots en el sistema se pueden ajustar paramétricamente.

Este requisito implica que el framework provea las siguientes áreas de funcionalidad:

- Agendar tareas
- Acceder a los snapshots de las VMs existentes
- Acceder a la fecha de creación de los snapshots
- Eliminar snapshots que cumplan las condiciones determinadas.

5. Reporte de recursos por organizaciones

Este requisito implica la generación de un reporte de la arquitectura actual tal como está definida en los Data Center virtuales de la organización. Se espera que el reporte incluya vApps, VMs, redes y otros recursos como IPs en uso, discos y características de CPU y memoria.

Este requisito implica que el framework provea las siguientes áreas de funcionalidad:

- Acceder a las Organizaciones disponibles y sus Data Center virtuales
- Acceder a los Data Center virtuales
- Acceder a vApps y sus recursos
- Acceder a recursos de VMs
    - CPU
    - Memoria
    - Interfaces de red
    - Discos

Arquitectura

Aspectos de arquitectura

En esta sección se presenta la arquitectura de software del framework como también el proceso con el cual, mediante sucesivas iteraciones, se llegó al diseño definitivo de la misma. En este documento no están presentados todos los diagramas y detalles de la arquitectura. Para acceder a un informe completo tomar como referencia el Documento de Arquitectura de Software [Arquitectura - Extensibilidad de herramienta de gestión para VMware Cloud Director] (Díaz & Sureda, 2021a). Los componentes y módulos dentro de cada subsistema, como así también la interacción entre ellos, se describen en detalle en dicho documento, tomando como base el modelo 4+1 (Kruchten, 1995).

En las siguientes secciones se presentan las vistas lógica y de distribución por el interés y detalle de información que presentan en cuanto al estilo arquitectónico y la infraestructura de la solución (deployment). Las otras vistas que componen el modelo 4+1 (las vistas de procesos, de implementación, y los casos de uso) se encuentran completamente detalladas en el documento antes mencionado (Díaz & Sureda, 2021a).

Luego de cumplida la fase donde se relevaron los requerimientos del sistema, se definió abordar el trabajo de definición de la arquitectura desde dos diferentes aspectos:

- Conceptual: Bajo este aspecto se definen qué tipo de entidades participan del modelo y cuáles son las relaciones entre ellas. En este primer aspecto, se modela una jerarquía de entidades presentes en VMware Cloud Director, agregando aquellas que se consideran necesarias para resolver los problemas planteados por los requerimientos descubiertos en la fase inicial. Algunas entidades identificadas en primera instancia, tras relevar requerimientos, son:
    - Tenant
    - Virtual Data Center
    - Virtual Machine
    - vApp (Virtual App)
    - Network
    - Snapshot

  Cada uno de los ítems de esta lista se convertirá en una clase dentro del componente Entidades formando parte fundamental del subsistema de backend. Ver figura 7 en la sección de Vista Lógica.

- Librerías de soporte: Este aspecto se enfoca en aquellos componentes que brindan una funcionalidad fundacional o de base al resto del framework. En este segundo aspecto de la arquitectura se definen las librerías que dan soporte al resto de las funcionalidades del framework. Estas librerías constituyen una base para el resto del sistema. Se identificaron grandes áreas de funcionalidades base que dan apoyo a la implementación de los diferentes requerimientos previamente identificados. La fase de prueba de concepto permitió identificar la necesidad de cubrir algunas áreas funcionales para que

el framework cumpla su objetivo. Dichas áreas, que pasarán a ser sub-componentes de las librerías de soporte, son:

- *TaskPlanner*: Administración de secuencias de tareas (Plan)
- *TaskScheduler*: Agendar tareas
- *RabbitMQClient*: Cliente RabbitMQ
- *vCloudSecurity*: Seguridad, autenticación y validación de permisos
- *Logger*: Registro de eventos

El diagrama puede verse en la figura 8 de la sección de Vista Lógica.

- Persistencia: Hay un último componente que se localiza en su propio módulo, separado de la librerías de soporte, y que brinda servicios de persistencia de datos a todo el framework como así también a las propias librerías. De esta manera, se estructura de mejor forma la arquitectura del subsistema de backend. Se deben persistir diferentes entidades para retomar el procesamiento de forma consistente ante un eventual reinicio del servidor. Los objetos a ser persistidos se serializan en formato de pares clave – valor para su almacenamiento

Como se ha visto, 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 tras sucesivas iteraciones. El detalle de cada uno de estos componentes se profundizan en el documento de arquitectura.

Vista lógica

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

Se confirma la participación en la solución de aquellos componentes que se habían incluido en el enfoque seleccionado de la prueba de concepto:

- Aplicación web de frontend integrada en la interfaz web de VMware Cloud Director
- Servidor o *broker* RabbitMQ como *middleware* de mensajería
- Servicio de backend que recibe y procesa los requerimientos que llegan desde el frontend a través del *broker*.

La implementación de la prueba de concepto permitió identificar y validar los detalles de la arquitectura. Ese modelo se describe en el capítulo Prueba de Concepto y en el documento de arquitectura (Díaz & Sureda, 2021a). El diagrama simplificado de distribución en ese mismo documento permite definir un estilo arquitectónico en capas (ver figura 5). Las capas de frontend y de backend, con sus respectivos componentes, aparecen claramente delimitadas.

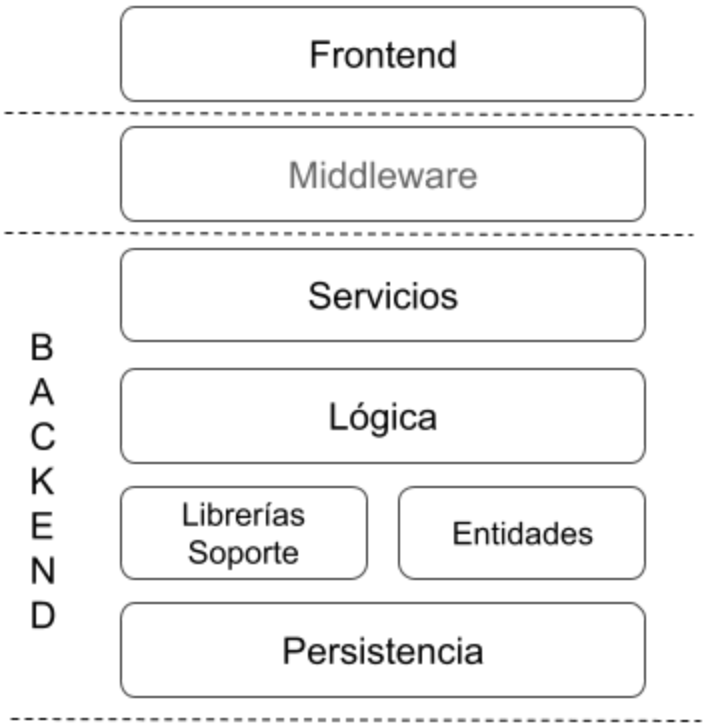

Figura 5 - Diagrama de estilo arquitectónico en capas: frontend y backend.

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, en la figura 6, el diagrama con sus capas o subsistemas, los componentes que lo integran y las relaciones macro entre ellos.

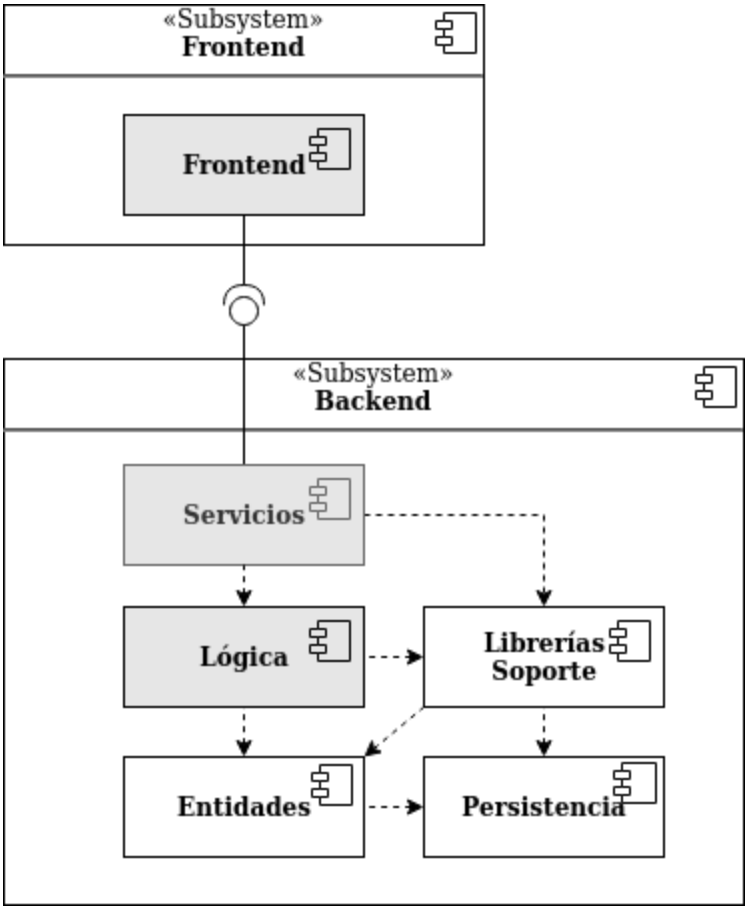

Figura 6 - Diagrama de subsistemas y componentes.

En el diagrama aparecen con fondo blanco aquellos componentes que comprenden el núcleo 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 el componente de Frontend en el subsistema de frontend.

Haciendo foco en los componentes fundamentales que componen el núcleo del framework, se presentan los diagramas de clases de los componentes de Entidades y las Librerías de Soporte.

En el componente *Entidades* se modela una jerarquía de entidades presentes en *VMware Cloud Director* necesarias para resolver los problemas planteados por los casos de uso. El diagrama de clases de la figura 7 presenta las interfaces de cada una de ellas y sus relaciones.

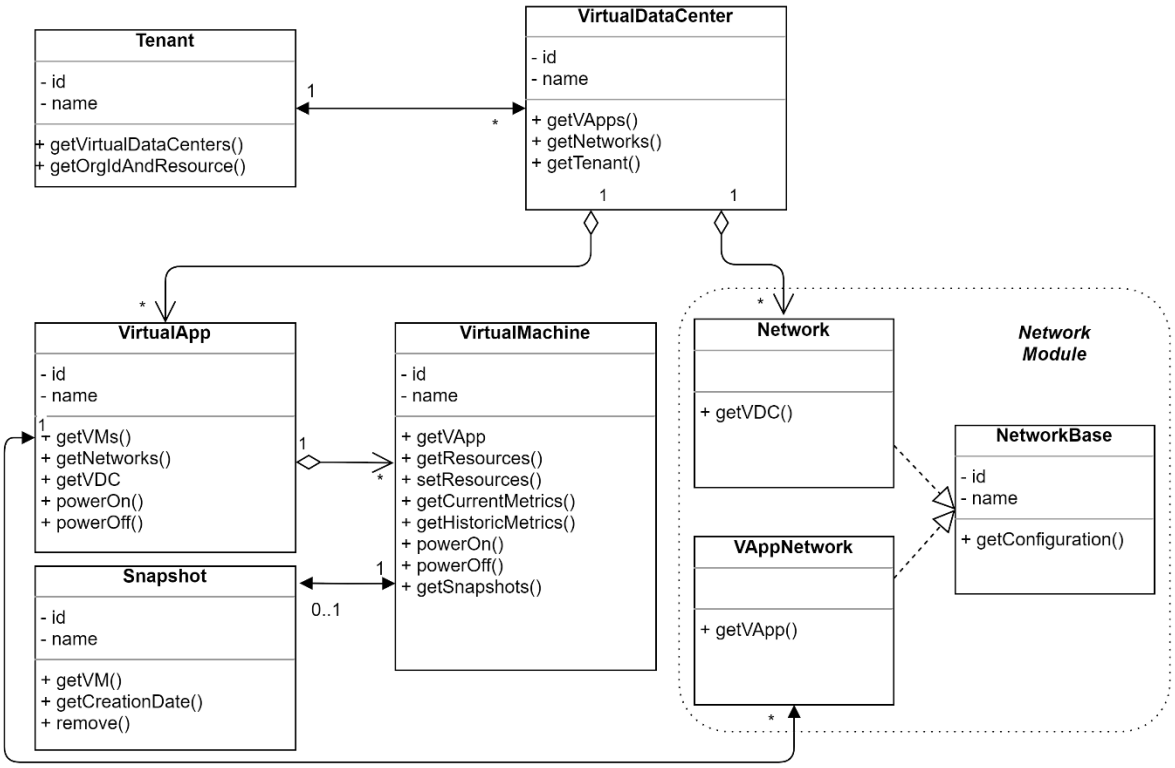

Figura 7 - Diagrama de clases. Enfoque conceptual de Entidades del sistema.

Como ya se mencionó, en el aspecto de las librerías se presentan aquellas clases que dan soporte al resto del framework. Constituyen un núcleo funcional disponible para el resto del sistema, que permite cumplir requerimientos de agenda de tareas (*TaskPlanner* y *TaskScheduler*), como también encargarse de algunos requerimientos no funcionales: interoperabilidad (*RabbitMQClient*), seguridad (*vCloudSecurity*) y auditabilidad (*Logger*), entre otros. A continuación, en la figura 8, se presenta el diagrama con las relaciones entre los diferentes subcomponentes.

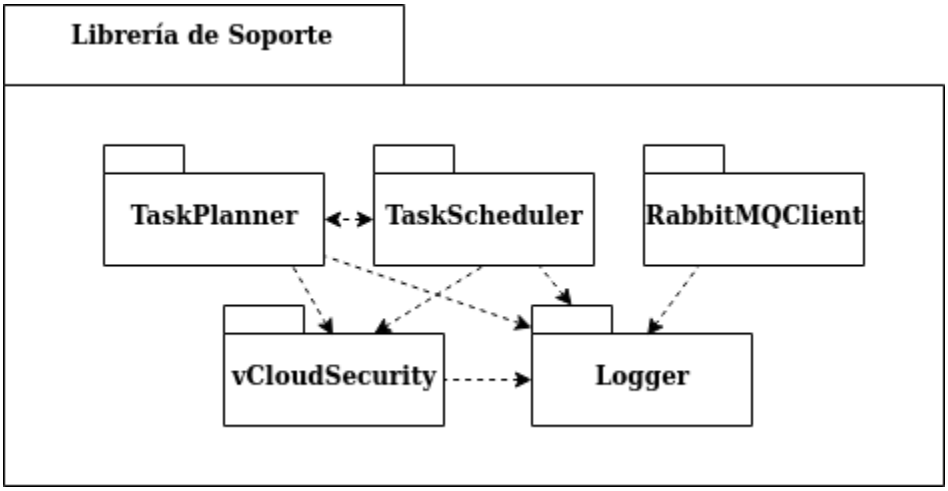

Figura 8 - Diagrama de componentes. Enfoque Librerías de Soporte.

Vista de distribución

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

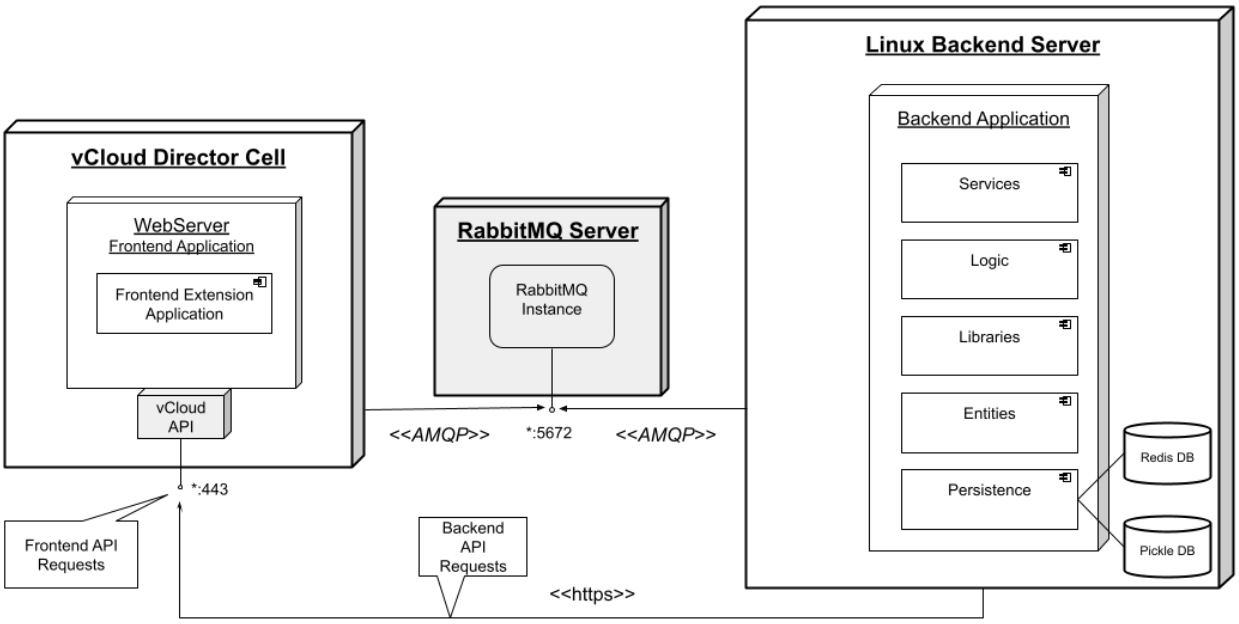

Figura 9 - Vista de distribución del framework.

Se presenta el servidor de *VMware Cloud 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 *VMware Cloud Director* dentro del servidor. El servidor Backend con SO Linux contiene todos los componentes y módulos descritos en la vista lógica.

Tal como se presentó en la prueba de concepto, se mantiene en el diseño de la arquitectura la integración con el servidor de *RabbitMQ*. El servidor de *RabbitMQ* es parte fundamental de la solución (ver figura 3). Dentro del mismo es necesario configurar un *exchange* y un *binding*, declarar una *queue* y definir una *routing-key*. Las definiciones de estos conceptos están presentes en la sección de Contexto y marco conceptual, bajo la descripción de RabbitMQ. La URL del *endpoint* de la extensión se asocian al *exchange* y al *routing-key* correspondiente para que *VMware Cloud Director* rutee el requerimiento al Backend. En el esquema se muestra 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 VMware Cloud Director, en el centro el servidor de RabbitMQ y a la derecha el servidor del Backend.

Implementación

Proceso de implementación

Al comienzo de la implementación se definió un plan de trabajo que permita hacer un seguimiento del nivel de avance. Teniendo una definición muy completa de la arquitectura y habiendo trabajado para documentar los casos de uso en detalle, se genera un extenso backlog para comenzar la implementación.

A cada caso de uso, y cada módulo implicado en su funcionalidad, se le asignan grupos de tareas básicas y se evalúa la asignación de cada una de ellas, según la dificultad y el conocimiento de los integrantes del grupo.

Se utiliza la herramienta *Azure Boards* de *Azure DevOps* (Microsoft, 2022), anteriormente conocida como Visual Studio Team System, para hacer un seguimiento del backlog, las historias y los sprints durante el proceso.

Los sprints se planifican con el objetivo de tener, en el menor tiempo posible, al menos un escenario básico de un caso de uso, operativo y funcional. Algunos requerimientos no funcionales (como la seguridad), son transversales a los casos de uso y demandan mayor esfuerzo, por lo que para algunos sprints es necesario enfocarse en finalizar algún módulo o desplegar mayor esfuerzo en tareas relacionadas a un componente en particular, antes que en finalizar la lógica de negocios del caso de uso específico. Esto motivó que algunos casos de uso, principalmente los que se trabajaron al principio, fueran necesarios hasta 4 o 5 sprints de dos semanas cada uno para tener el caso de uso funcional.

A continuación se presenta una lista con parte del backlog inicial generado para comenzar el desarrollo.

ID	Title	Assigned To	State
5	RabbitMQ Client	Marcelo Sureda	Done
17	Entities	Marcelo Sureda	Done
19	CU 5 / FE / Reporte de recursos por organización	Aldo Diaz Betizagasti	Done
20	CU 5 / BE / Reporte de recursos por organización	Marcelo Sureda	Done
21	Package de Frontend limpio	Aldo Diaz Betizagasti	Done
23	Interfaz para el módulo de vCloud Security	Marcelo Sureda	Done

La separación de la carga de trabajo en sprints ayudó a cumplir con los objetivos y hacer el seguimiento del avance del desarrollo, si bien no siempre fue posible cumplir con las tareas planificadas en cada sprint. El hecho de ser solamente dos integrantes en el equipo dificulta poder cubrir el trabajo planificado, cuando uno o ambos miembros del equipo no consiguen, por diferentes motivos (demanda de trabajo particular, otras materias de la carrera, imprevistos, etc.) dedicarle el tiempo requerido a las tareas asignadas.

Organización del código en backend

En el repositorio del código del backend la estructura de directorios es la siguiente:

```
\vcd-extension-backend
├── tests
├── usecases
│   ├── logic
│   └── services
└── vcdextension
    ├── entities
    ├── libraries
    ├── logic
    ├── persistence
    └── services
```

La organización del código refleja la arquitectura presentada en el capítulo correspondiente, con los packages, sub-packages y módulos que implementan los componentes definidos en la arquitectura.

- ***vcd-extension-backend***: Raíz del repositorio.
  - Archivos habituales: "*ReadMe*", licencia de software libre (Apache 2.0), etc.
  - pyproject.toml/setup.py/setup.cfg: Metadata del package de Python haciendo uso de la versión más reciente de setuptools. Dichos archivos son requeridos para distribuir y disponibilizar el framework a los usuarios finales.
- ***tests***: Contiene los tests de integración para el backend y tests unitarios.
  - test_curl.sh: *Script* Linux en *bash* que testea las APIs del servicio de backend.
  - Los tests unitarios se deben colocar en este directorio, pero al momento de redactar este documento, están pendientes de ser desarrollados.
- ***usecases***: La lógica y la capa de servicios de los casos de uso desarrollados están presentes en este directorio (ver capítulo Relevamiento de Casos de Uso).
  - config.yml: Ejemplo de archivo de configuración para arrancar el servidor.
  - startcustomserver.py/start_server.sh: Scripts para iniciar el servidor.
  - logic: Directorio que concentra la lógica de los casos de uso. Contiene cinco archivos con su módulo y clase correspondiente cada uno. Cada módulo corresponde a uno de los cinco casos de uso implementados.
  - services: Se compone de dos módulos: customservices.py y services.py. El primero hereda del componente ServiceRabbitMQ del framework para activarse cuando el servidor recibe un request personalizado para uno de los casos de uso (es decir, cuando es un *endpoint* que no es del estándar del framework). El segundo módulo (services.py) exponen los puntos de entrada a la lógica de los diferentes casos de uso.
- ***vcdextension***: Éste es el principal package de Python del framework. Es el núcleo que da base a toda la funcionalidad de los casos de uso. Todos los componentes y módulos están descritos en detalle en el documento de arquitectura (Díaz & Sureda, 2021a).
  - entities: Componente que modela las entidades presentes en VMware Cloud Director. Los módulos dentro del componente son: entity.py, network.py,

snapshot.py, virtualapp.py, virtualmachine.py, tenant.py, virtualdatacenter.py. Cada uno de ellos representa una clase dentro del componente.

- *libraries*: Componente que incluye clases que dan soporte al resto del framework. Como se presentó en la arquitectura estos módulos componen el núcleo funcional de soporte al resto del sistema. Está compuesto por funcionalidades auxiliares generales para el servidor en serverutils.py, constants.py, exceptions.py; en logger.py la clase Logger; vcloudsecurity.py implementa las funcionalidades de seguridad (clases VCloudSecurity, CheckPermissions y CheckUserAccess); el módulo taskplanner.py (clases Task y Plan) implementa la planificación de tareas; rabbitmqclient.py (clase RabbitMQClient) la interacción con RabbitMQ; taskscheduler.py (clase TaskScheduler) el agendado de actividades.
- *logic*: Este directorio agrupa la lógica básica del framework, dando soporte a los casos de uso personalizados. Está compuesto por los siguientes módulos: logicorgs.py (clase LogicOrgs), logictasks.py (clase LogicTasks), logicvapps.py (clase LogicVApps), logicvms.py (clase LogicVMs).
- persistence: Directorio que engloba la funcionalidad de persistencia del framework.
- *services*: Se compone de dos módulos: servicerabbitmq.py y services.py. El primero en la clase ServiceRabbitMQ, hereda la funcionalidad base de la clase RabbitMQClient de las librerías para evitar tener que lidiar con los detalles finos de la interacción con el *broker* de mensajería. Se define cuál servicio invocar y se llama al método correspondiente de la clase Services. En services.py se exponen los puntos de entrada a la capa de lógica del framework.

Organización del código en el frontend

En el repositorio del código del frontend la estructura de directorios es la siguiente:

```
\pyvcloud-frontend\src\main
├── menu.component.html
├── menu.component.ts
├── app.module.ts
├── app.routes.ts
├── business
    ├── auto-delete-snapshots.component
    ├── org-report.component
    ├── schedule-activities
    ├── task-on-multiple-orgs
    ├── auto-delete-snapshots
│   └── job-list
└── service
    └── http-backend.service.ts
```

En el directorio principal “main” se encuentran todos los archivos que implementan las funcionalidades de la solución. El mismo incluye, en un macro nivel, al componente del menú, el directorio con la implementación de cada caso de uso y el directorio con los servicios implementados.

- **Componente menú:** encargado de representar la navegación principal del sitio. Se compone de dos archivos distintos, la plantilla html y el archivo con la lógica. Se visualiza en el sitio web como una barra vertical siempre presente en la izquierda y contiene una lista con un hipervínculo a cada sección del sistema.
- **Archivos de configuración de Angular:** el entorno de trabajo Angular requiere una serie de archivos de configuración que permiten definir la estructuración de la solución. Entre ellos se encuentra el app.module.ts y app.routes.ts. El primero, tiene como objetivo registrar todos los componentes definidos y utilizados en el sistema. El segundo, contiene todas las rutas de navegación del sistema, por ejemplo, cada caso de uso tiene su propia sección en el sitio web de la que se puede acceder a través de una URL, para ello se define una dirección dentro del directorio, un nombre que identifica la sección y el componente que la implementa.
- **El directorio "business":** incluye todos los componentes que implementan los requisitos del proyecto. Para cada caso de uso, se creó un componente distinto que lo implementa. La estructura de estos componentes se compone de tres archivos distintos y encapsulan la implementación de la lógica, vista y estilos de manera independiente.
- **Componentes dentro del directorio "business":** cada uno de los componentes define en su vista un formulario que permite, al usuario, el ingreso de los parámetros requeridos referentes al caso de uso y por el lado de la lógica se obtiene la información ingresada, se procesa y se invoca el servicio correspondiente a través del componente de servicios.
    - "Auto-delete-snapshots": se encarga de implementar el caso de uso "Eliminación de Snapshots obsoletas".
    - "Org-report": se encarga de implementar el caso de uso "Reporte de recursos por organizaciones".
    - "Schedule-activities": se encarga de implementar el caso de uso "Programación de secuencia de actividades".
    - "Task-on-multiple-orgs": se encarga de implementar el caso de uso "Replicar acciones a varias organizaciones".
    - "Auto-scale": se encarga de implementar el caso de uso "Escalado de recursos".
    - "Job-list": se encarga de implementar el caso de uso "Listar planes agendados".
- **Directorio de servicios:** se encarga de organizar los servicios utilizados en el sistema. Contiene el servicio "http-backend" que tiene como principal objetivo centralizar las llamadas y la lógica utilizada para realizar a la interfaz de servicios del backend. Y luego, ofrece una interfaz a los componentes anteriormente listados con cada uno de los servicios.

Gestión de configuración del software (SCM)

Metodología

Como servidor de configuración y administración de código fuente se utiliza el servidor GitLab de la Facultad de Ingeniería. Se puede acceder al repositorio del código del backend en el siguiente enlace de Gitlab:
https://gitlab.fing.edu.uy/proyecto-grado-vcloud/vcd-extension-backend

La metodología de trabajo para ajustarse a los objetivos de seguimiento y orden propuestos se detallan a continuación:

1. Cada nueva funcionalidad o característica del software tiene asociada una tarea en Azure Boards (Visual Studio Team System)
2. A partir de la versión actual del código en Master se genera una nueva branch que referencia la nueva funcionalidad o característica.
3. Finalizado el desarrollo y testeada la modificación, se sube a GitLab el branch.
4. Se realiza un *Pull Request* a *Master*. Se revisa y autoriza el *Pull Request* realizando el *merge* a Master desde el portal de GitLab.
5. Para iniciar el siguiente ciclo de desarrollo, se debe cambiar a la branch local de *Master* y hacer un *pull* desde el servidor para sincronizar los cambios.

Si se presentan conflictos entre versiones en cualquier punto del proceso, se rechazan los cambios en el servidor. Se actualizan los branches locales y se resuelven dichos conflictos en el repositorio local antes de subirlos al servidor de GitLab. Durante el desarrollo, prácticamente no se encontraron conflictos, al realizarse la mayoría de los cambios de forma serializada.

Durante el primer sprint de implementación, todo el trabajo de creación de módulos y actualizaciones iniciales se realizó mediante sucesivos push directos a Master. Cuando el seguimiento se hizo más complejo se ajustó la forma de trabajo a la metodología descrita.

Versionado

Para administrar las versiones del framework de forma automatizada y de acuerdo al capítulo “Automated inclusion of version string from package” del libro Expert Python Programming (Ziadé & Jaworski, 2019), se incluye la siguiente línea en el setup.cfg:

```
version = attr: vcdextension.__version__
```

De esta forma, para la distribución del paquete de Python, se toma el valor desde el archivo vcdextension/__init__.py. En dicho archivo se agrega el siguiente contenido:

```
# version as tuple for simple comparisons
VERSION = (0, 0, 1)
# string created from tuple to avoid inconsistency
__version__ = ".".join([str(x) for x in VERSION])
```

La ventaja de este enfoque es que permite, sin duplicar información, sincronizar el acceso a la versión del software desde el paquete y el código fuente para, por ejemplo, usarlo con fines de auditoría en la clase Logger.

Mecanismos de programación aplicados

Hay un conjunto de mecanismos de la programación orientada a objetos (POO) y patrones de diseño generales, que fueron aplicados en la implementación que permiten cumplir algunos requerimientos tanto funcionales como no funcionales. A continuación se describen algunos de ellos, explicando qué función cumplen dentro del framework y en qué partes se aplican.

Herencia

La herencia, en POO, es un mecanismo que se utiliza para permitir la reusabilidad y extensibilidad. La herencia permite crear una nueva clase, también llamada clase derivada, hija o subclase, a partir de una clase ya existente, llamada clase base, padre o superclase. La subclase recibe (hereda) todo el comportamiento (métodos) y datos (atributos) de su superclase.

Con el objetivo de reutilizar código se aplica el mecanismo de herencia entre la clase base Entity y las clases derivadas Tenant, VirtualDataCenter, NetworkBase, VirtualMachine, Snapshot y VirtualApp. Se concentra en la clase Entity los atributos y métodos comunes a todas las entidades de VMware Cloud Director. También se aplica herencia entre la clase base NetworkBase y las clases derivadas Network (representa una red de un VDC) y VappNetwork.

Con el objetivo de extender la funcionalidad del sistema (extensibilidad) se utiliza el mecanismo de herencia entre la clase base abstracta (no permite ser instanciada) RabbitMQClient y la clase derivada ServiceRabbitMQ. En la clase base se encapsula la funcionalidad de interacción con el *broker* RabbitMQ a bajo nivel, y la clase derivada extiende su funcionalidad con el objetivo específico de procesar el requerimiento de entrada recibido del *broker*. En la clase derivada ServiceRabbitMQ se analiza el mensaje, se identifica el *endpoint* involucrado y se invoca el método correspondiente de la clase Services, que es el punto de entrada a la capa de lógica del sistema.

Otro punto en el que se utiliza la herencia es para la definición de excepciones con una clase base (VCDExtensionBase) que define el comportamiento básico para todas las excepciones y luego los casos de excepciones particulares como derivadas de ella.

El mismo sistema permite extender la funcionalidad para el usuario del framework, agregando nuevos *endpoints* en una clase personalizada que herede de la misma clase ServiceRabbitMQ. Este mecanismo puede observarse en el repositorio dentro del directorio `usecases`, donde hay cinco ejemplos de extensiones del framework (los cinco CU). La clase CustomServices deriva de la clase ServiceRabbitMQ que, como ya vimos, deriva a su vez de la clase RabbitMQClient. La clase CustomServices procesa los requerimientos desde el frontend para

los casos de uso personalizados extendiendo el framework, identificando aquellos *endpoints* que no son responsabilidad del propio framework. A partir de ahí se puede derivar la lógica a las clases de servicios y lógica personalizadas.

Singleton

El Singleton (Johnson et al., 1994, #144) es un patrón de diseño que restringe la cantidad de instancias de una clase a un único elemento. Se hace necesario cuando se necesita coordinar acciones de acceso entre todo el sistema con un único objeto.

Dentro del sistema, se definen como Singleton las clases VCloudSecurity, Logger, Persistence y TaskScheduler. Cada una de ellas ofrece un tipo de servicio al resto del framework y, por sus características, necesitan unificar dentro del sistema el acceso a sus funciones exportadas. Se realiza una implementación tradicional del lenguaje Python (Ziadé & Jaworski, 2019).

Decorator

El patrón Decorator (Johnson et al., 1994, #199) es un patrón de diseño que permite agregar determinado comportamiento a objetos de forma dinámica.

Dentro del framework es utilizado en forma intensiva para agregar validaciones de seguridad. En el módulo de seguridad se definen dos decorators:

- CheckPermissions: Valida que el usuario que inicia el requerimiento tenga permisos para realizar la acción y acceder a la entidad afectada.
- CheckUserAccess: Valida que el usuario que inicia el requerimiento tenga permiso para acceder a la vApp involucrada (hay permisos de acceso específicos a las vApps para los diferentes usuarios de la misma organización).

En ambos casos el usuario es obtenido de la información de contexto del mensaje recibido.

El decorator CheckValidMethods, definido en el módulo ServiceRabbitMQ, valida que los métodos HTTP empleados sean los definidos según el caso de uso.

En el módulo Exceptions se define el decorator LogicExceptionHandler para poder administrar en la capa de lógica aquellas excepciones lanzadas en capas inferiores.

Lazy initialization

Lazy initialization (Johnson et al., 1994, #128), inicialización perezosa o tardía en español, es un mecanismo para demorar la creación de un objeto o la realización de cualquier proceso costoso hasta el momento en que sea requerido por primera vez dentro del sistema. Todos los Singleton utilizan este mecanismo, pero se destaca en el caso de la clase VCloudSecurity. El objeto de la clase no es creado hasta que se requiere acceder al servidor de VMware, ya que la creación misma del objeto realiza la autenticación ante el servidor (invoca el método privado VCloudSecurity.__login).

Al inicializar el servidor, no es instanciada la clase VCloudSecurity. Cada vez que en cualquier punto del código se requiere acceder al servidor, se solicita la instancia singleton de la clase VCloudSecurity. Mediante este mecanismo se valida si existe la instancia. Si existe, se devuelve inmediatamente, y en caso que no exista, se la crea, se realiza la autenticación ante el servidor y se devuelve la instancia de la clase para que el módulo cliente pueda acceder al servidor.

Excepciones

El mecanismo de manejo de excepciones es utilizado para gestionar todo tipo de errores y excepciones de procesamiento. Permite que en la capa lógica se capturen todos los errores que se puedan disparar en otros puntos del código. De esta forma, el framework reacciona adecuadamente devolviendo al frontend un mensaje de error correspondiente al tipo de excepción siendo capturado.

Como se mencionó previamente, se combina con el mecanismo de herencia y el patrón de diseño decorator. A continuación se presenta cómo se define un método de la clase LogicOrgs, donde se obtienen las organizaciones disponibles en el sistema, agregando el decorator LogicExceptionHandler.

```
@LogicExceptionHandler
def get_orgs(cls, context_id):
```

De esta forma se puede extender el comportamiento de los métodos de la capa de Lógica del framework (y de los que a futuro pueda agregar el usuario), manteniendo un criterio uniforme en todo el sistema, controlando excepciones sin tener que chequear en cada caso todos los posibles tipos de error devueltos por las capas inferiores.

Desafíos técnicos

A lo largo del proceso de implementación hubo que enfrentar algunos obstáculos técnicos que significaron demoras en los tiempos de desarrollo previstos originalmente. Los más complejos de resolver fueron:

- Versión requerida de Angular obsoleta: Se dedicó un tiempo considerable, esfuerzo e investigación para el desarrollo de un sistema de frontend con tecnologías modernas y actualizadas. Sin embargo, el sistema de plugins de la versión concreta de VMware que se utilizó no admite las nuevas versiones de Angular. Al momento de escribir este documento, la última versión estable de Angular es la 12 y la versión con la que se logró integrar es la 4.
- Distribución Python: El estado del arte en lo relativo a las tecnologías utilizadas para la creación y distribución de paquetes de software de Python ha sido de difícil abordaje. Esto es debido a la multiplicidad de métodos disponibles para generar paquetes de distribución, a la documentación dispersa, caótica, y muchas veces, contradictoria y desactualizada. En los últimos años se ha establecido cierto orden gracias a la

organización PyPA (Python Packaging Authority) (*PyPA Documentation*, 2021), y en la actualidad se ha impuesto como estándar para crear distribuciones basadas en código fuente la herramienta setuptools (Ziadé & Jaworski, 2019, págs. 200-202). A pesar de la mejoría que eso significa, las mismas setuptools (*Setuptools*, 2022) han sufrido cambios durante el desarrollo del proyecto que representaron un desafío a la hora de empaquetar el software para su distribución. Teniendo en cuenta que el proceso de empaquetado y distribución del framework es fundamental para permitir su uso por parte de terceros, fue necesario adecuarse a estos cambios para tener un proceso de distribución consistente con los requerimientos más recientes.

Requisitos No Funcionales

En la presente sección se realizan consideraciones relativas al cumplimiento de requerimientos no funcionales.

Rendimiento

Se considera por fuera del alcance del proyecto realizar tests de carga o capturar métricas de rendimiento del sistema. Se analizará desde el punto de vista conceptual la capacidad del framework y se presentarán alternativas de despliegue que permitan mejorar su performance.

En un entorno multiusuario, en donde todos los requerimientos derivan a, y son atendidos por, un único servidor de backend, el cuello de botella apunta a ser el propio servidor. Se parte de las siguientes premisas:

- Se asume que tanto el servidor de VMware como de RabbitMQ están debidamente dimensionados y configurados para soportar la carga de recepción y ruteo de mensajes al servicio de backend.
- El conjunto de usuarios que utilizan el sistema son una cantidad limitada. Se estima que la cantidad total de usuarios que accedan esté en el orden de las decenas antes que de los millares. Los perfiles esperados son: administradores del provider, orgadmins, usuarios de la organización, operadores en general.

Además de procesar los requerimientos en línea de los diferentes usuarios, el mismo proceso de backend se encarga de lanzar la ejecución, en diferentes hilos de procesamiento, de todas aquellas actividades que hayan sido agendadas. Esto debe considerarse al momento de estimar la carga recibida.

En cuanto a rendimiento y concurrencia, el lenguaje Python tiene sus limitaciones. Si bien las tareas agendadas se ejecutan en hilos de procesamiento diferentes a los que atienden los requerimientos en línea de los usuarios, se debe notar que, debido a la implementación del intérprete Python, en particular al Global Interpreter Lock (GIL), sólo un hilo de procesamiento puede estar en ejecución a la vez (Python, 2022).

Una solución sería refactorizar el código haciendo uso de librerías de multiprocesamiento para lanzar subprocesos en lugar de hilos, pero implicaría cambios considerables en el código y la sustitución de una librería fundamental utilizada para agendar las tareas.

Sin necesidad de hacer cambios importantes en el código, se puede obtener una mejor distribución de recursos computacionales con un tipo de despliegue diferente. Esto consiste en dedicar un servidor backend por cada organización dentro del sistema. De esta forma se separa, de forma natural y en diferentes procesos, toda la carga de procesamiento del backend. Así se permite, además, simplificar los controles de seguridad (ver la sección Seguridad en este capítulo).

Disponibilidad y estabilidad

El servicio de backend debe permanecer activo y en ejecución de forma ininterrumpida para cumplir su doble cometido de atender requerimientos de usuarios conectados y ejecutar aquellas actividades agendadas en diferentes días y horarios.

Un error en la lógica que genere una terminación anormal del servicio de backend compromete este objetivo. Por ello, la gestión de excepciones a nivel de capa lógica capturando todos los errores posibles de capas inferiores con el decorator *LogicExceptionHandler* en los métodos correspondientes favorece este objetivo. Este manejo ayuda a devolver la respuesta correspondiente desde el backend al frontend, generando la información correspondiente en el log de eventos.

Hay eventos externos que pueden afectar la estabilidad del servicio de backend. El backend, para su funcionamiento, depende de la conectividad permanente con el servidor VWware Cloud Director y con el *broker* RabbitMQ. La caída del servidor remoto, o un problema de red, puede producir una pérdida de conexión. Dependiendo de cuánto tiempo esté interrumpida la conexión, el servicio de backend puede recuperarse o no. En caso que se supere el límite de intentos de conexión, se producirá una caída del servicio y deberá intervenir un operador para reiniciarlo.

Para detectar la terminación anormal del servicio se deben establecer procedimientos y métodos de monitoreo. Los mensajes en el log de eventos, antes de la terminación del servicio, dan detalles del error ocurrido para identificar la causa y permitir solucionar las posibles condiciones que provocaron la caída.

Seguridad

El servicio de backend, al tener necesidad de interactuar con VMware Cloud Director y el servidor de RabbitMQ, debe autenticarse ante dichos sistemas externos con las credenciales indicadas en la configuración (ver sección Configuración del servidor).

Credenciales RabbitMQ

Para operar con el servidor de RabbitMQ se debe definir un usuario con permisos para realizar las siguientes operaciones:

- Iniciar una conexión remota al servidor.
- Declarar un *exchange* y una *queue*.
- Realizar un enlace (operación *bind*) a una *queue* y a una *exchange*.
- Consumir mensajes de una *queue*.
- Publicar mensajes a un *exchange*.

Estas son las operaciones que debe realizar el backend para conectarse al *broker*, leer los requerimientos y enviar las respuestas.

Credenciales VMware Cloud Director

El usuario utilizado para conectarse al servidor de VMware Cloud Director determina las acciones que el servicio de backend podrá realizar. Las principales categorías (roles) a las que puede pertenecer un usuario son los siguientes:

- Administrador del sistema (*provider*): Puede acceder en forma ilimitada a todas las funciones, organización y entidades dentro del sistema.
- Administrador de organización (orgadmin): Tiene acceso a todas las funciones dentro de un *tenant*.
- Usuario de organización (catalog author, vApp author, vApp user, console access only): Tienen permiso de acceso a un conjunto limitado de funciones dentro de una organización específica.

Al implementar los casos de uso se debe considerar que todas las acciones a realizar para cumplir con sus requerimientos deben estar habilitadas para el usuario con el que el backend se autentica ante VMware Cloud Director. Por consiguiente, es deseable que el backend utilice un usuario con un nivel elevado de permisos, por ejemplo, administrador de sistema o de organización. Además, el servicio de backend podrá ejecutar todas aquellas acciones para las que tenga permisos el usuario autenticado contra el servidor VMware.

Por otro lado, el usuario autenticado desde el frontend puede ser, y generalmente es, diferente al usuario utilizado por el servicio de backend y es de esperar que tenga un rol con un nivel menor de privilegios, por ejemplo, vApp user. Por lo tanto, el backend será responsable de validar que no se le permita a los usuarios logueados en el frontend escalar privilegios de forma irregular. Es decir que el usuario del frontend no debería poder ejecutar una acción mediante el framework que no le esté permitida ejecutar mediante el portal estándar de VMware Cloud Director.

Para realizar este tipo de validaciones el framework utiliza el mismo esquema de seguridad que VMware Cloud Director sin implementar un sistema aparte. Todas las autorizaciones de seguridad se realizan considerando los permisos establecidos en el servidor.

Como se detalló en la sección de “Mecanismos de programación aplicados”, las validaciones de seguridad se realizan mediante los decorators *CheckPermissions* y *CheckUserAccess* en aquellos los métodos que lo requieran.

El decorator *CheckPermissions* es usado para validar los permisos del usuario. Los permisos del usuario que lanzó el requerimiento desde el frontend están incluidos en la información de contexto presente en el mismo mensaje original que llega desde el servidor de VMware. Se valida si la lista de permisos indicados en los parámetros del decorator están asignados al usuario en el servidor. A continuación, se presentan un par de ejemplos de uso del decorator CheckPermissions.

Ejemplo 1: En el constructor de la clase VirtualDataCenter se valida que el usuario tenga asociado el permiso ORGANIZATION_VDC_VIEW en el servidor.

```
class VirtualDataCenter(Entity):

        @CheckPermissions([RightName.ORGANIZATION_VDC_VIEW])
        def __init__(self, tenant=None, vdc_name=None, vdc_id=None, create=False,
                     context_id=None):
        """
        VirtualDataCenter initializer.
        …
        """
```

Ejemplo 2: En el método set_resources de la clase VirtualMachine se valida que el usuario tenga asociados los permisos VAPP_EDIT_VM_CPU y VAPP_EDIT_VM_MEMORY.

```
        @CheckPermissions([RightName.VAPP_EDIT_VM_CPU, RightName.VAPP_EDIT_VM_MEMORY])
        def set_resources(self, new_resources, context_id=None):
        """
        Set VM CPU and/or memory
        …
        """
```

El decorator CheckUserAccess valida que el usuario que inicia el requerimiento tenga permiso para acceder a la vApp involucrada. Se valida que el usuario del frontend pertenezca a la misma organización de la vApp que se quiere acceder, también se valida si la vApp puede ser accedida por todos los usuarios dentro de la organización, y si no es así (si hay una lista de acceso para la vApp), se valida si el usuario logueado desde el frontend puede acceder a esa vApp específica (si está en la lista de acceso). En el siguiente ejemplo se presenta un uso del decorator CheckUserAccess para validar el acceso en el constructor de la vApp.

```
class VirtualApp(Entity):
        """
        VirtualApp element inside VirtualDataCenter
        """

        @CheckUserAccess()
```

```
def __init__(self, vdc=None, vapp_name=None, vapp_id=None, create=False,
             context_id=None):
"""
VirtualApp initializer
…
"""
```

Con estas consideraciones, más las de la sección de Rendimiento, se puede comparar entre las implicaciones de hacer el despliegue ejecutando el servicio de backend con credenciales de administrador de sistema versus administrador de organización.

Características	**Credenciales de Administrador de Organización**	**Credenciales de Administrador de Sistema**
URLs (*)	Un prefijo de URL por organización y por operación.	Único prefijo de URLs para todo el sistema.
Ruteo (*)	Se debe registrar un servicio de backend por organización, definiendo un *exchange* y una *routing-key* dentro de RabbitMQ por organización.	Se registra y declara un *exchange* y una *routing-key* dentro de RabbitMQ para todo el sistema.
Validación y autorizaciones	Simplifica dentro del backend los controles de seguridad.	Mayor complejidad en los controles.
Rendimiento	Mejora el rendimiento al tener dedicado un servicio de backend por organización.	Se resiente el rendimiento al tener un único servicio de backend
Acceso por Tenants	Cada servicio accede sólo a su organización correspondiente.	El único servicio de backend accede a todos los *tenants*.

(*) La decisión también afecta el desarrollo en el frontend.

Nota: La implementación realizada fue haciendo uso de las credenciales del administrador de sistema. Un caso de uso como el que replica acciones en varias organizaciones no es posible implementar usando credenciales de administrador de organización.

Interoperabilidad

En referencia al modelo de calidad de McCall en “Ingeniería de Software Un enfoque práctico” (Pressman, 2003, #325) se define la interoperabilidad como el esfuerzo necesario para acoplar un sistema con otro. El framework de extensión debe comunicarse con VMware Cloud Director y con RabbitMQ.

Para la comunicación con el sistema de VMware el framework resuelve los detalles finos de comunicación con el servidor gracias a la librería pyvcloud. El usuario del framework no debe ocuparse de interactuar con el servidor durante la conexión y autenticación.

La interacción con el servidor de RabbitMQ se resuelve en las librerías de soporte con la clase abstracta RabbitMQClient, que inicializa la conexión y todos los objetos requeridos para interactuar con el *broker*. El usuario del framework al heredar la clase RabbitMQ puede implementar el método process_request() para analizar el mensaje recibido y enviarlo a la capa de lógica, para independizarse de los detalles de interacción con el broker.

Calidad de código

Guía de estilo

Para el desarrollo del backend se utiliza el IDE PyCharm (*PyCharm: The Python IDE for Professional Developers by JetBrains*, 2022). El IDE tiene integrada una funcionalidad que realiza inspecciones automáticas del código para validar errores y garantizar que se aplica un estilo predefinido. Seguir una guía estándar ayuda a uniformizar el aspecto, facilitando la lectura y comprensión del código. De esa forma se simplifica la detección de errores, incrementando la eficiencia del programador y logrando que el proyecto sea más simple de mantener.

En el proyecto, para el código de backend, se siguen las recomendaciones de PEP-8 (*PEP 8*, 2022). En PyCharm hay un conjunto predefinido de reglas que se validan. Se reportan diferentes niveles de severidad según el tipo de infracción: error, advertencia y advertencia leve. Los errores indican problemas de sintaxis en el código, lo que impide su ejecución. Las advertencias indican fragmentos de código que podrían introducir una falla funcional (*bug*). Las advertencias leves indican código que es correcto y funcional, pero que puede ser optimizado, como por ejemplo, código redundante.

Con una configuración por defecto del analizador de código del IDE, que incluye la validación de la guía PEP-8, se obtiene como resultado 0 errores, 1 advertencia y 9 advertencias leves.

La advertencia señalada por el analizador de PyCharm corresponde a un error de tipo de parámetro en la invocación al método list_resources() de la librería pyvcloud:

```
Expected type 'str', got 'EntityType' instead
```

La advertencia se debe a que en el *docstring* de la librería pyvcloud se indica incorrectamente *string* como el tipo del parámetro que, en realidad, corresponde a un enumerado *EntityType*. A continuación se comparte el *docstring* en el código de pyvcloud, donde se evidencia el problema:

```
    def list_resources(self, entity_type=None):
    """Fetch information about all resources in the current org vdc.
```

```
    :param str entity_type: filter to restrict type of resource we want to
        fetch. EntityType.VAPP.value and EntityType.VAPP_TEMPLATE.value
        both are acceptable values.
      """
```

No es un problema del código del framework ni de la lógica, ya que la inspección de código sólo valida el comentario del tipo de parámetro en el *docstring* y el funcionamiento del caso de uso es correcto, por lo tanto, dicha advertencia se puede desestimar.

Las advertencias leves corresponden a parámetros opcionales declarados en el encabezado de una función y no utilizados en el cuerpo de la misma. Del total de las 9 advertencias leves, en 7 casos corresponden a parámetros cuyo valor es utilizado por un *decorator* de la función aunque no en la propia función, y en otros 2 casos son parámetros de un método de la clase RabbitMQClient no requeridos por el framework, pero sí para interactuar con el servidor de RabbitMQ.

Medición de esfuerzo requerido - Cantidad de líneas de código

Entre los objetivos del desarrollo del framework figura extender la API del producto mientras se simplifica la interacción con la plataforma de VMware. Por ese motivo, es deseable que quien utilice el framework para desarrollar una extensión de VMware Cloud Director realice un esfuerzo menor que quien se vea obligado a desarrollar una extensión “desde cero”.

Una de las métricas más comunes para medir el esfuerzo requerido en un desarrollo es la cantidad de líneas de código. La opción disponible para realizar una comparación con el framework, es la prueba de concepto en la que se implementó el caso de uso Agendar Apagado de una VM (ver capítulo Prueba de concepto). La prueba de concepto se realizó sin contar con las librerías del framework, por lo que es adecuada para comparar con el caso de uso Programación de secuencia de actividades (ver capítulo Relevamiento de casos de uso). El caso de uso desarrollado con el framework es, incluso, más amplio que el desarrollo de la prueba de concepto, ya que mientras con el framework se puede agendar diferentes actividades, en la prueba de concepto sólo se puede agendar el apagado de una VM.

Los criterios adoptados para la comparación de cantidad de líneas de código son los siguientes:

- La comparación se realiza en el backend, que es donde se ubica el framework. El código del frontend puede ser exactamente el mismo en ambos casos.
- Se consideran líneas efectivas de código sin contar líneas con comentarios o en blanco (obligatorias por PEP-8).
- No se cuentan en la comparación los módulos del framework Entidades, Librerías de soporte y Persistencia ya que se consideran parte de lo proporcionado al usuario del framework, y es un núcleo común a todos los casos de uso.
- Se compara considerando sólo los módulos de las capas de Lógica y Servicios que corresponden a lo que el usuario del *framework* debe desarrollar para extender la API.

- Para los módulos de la capa de Servicios, donde se engloban funcionalidades de más de un caso de uso, se consideran aquellas líneas que corresponden específicamente al caso de uso Programación de secuencia de actividades (el que se quiere comparar).

Se presentan las líneas efectivas de código en los tres módulos desarrollados para la prueba de concepto:

Módulo	**# líneas**
utils.py	30
vapp_shutdown.py	36
vcd_scheduled_vm_shutdown.py	86
Total líneas de código	**152**

A continuación se presentan las líneas efectivas de código en los tres módulos desarrollados para el caso de uso Programación de secuencia de actividades haciendo uso del framework:

Módulo	**# líneas**
customservices.py	23
services.py	9
logicscheduleplan.py	16
Total líneas de código	**48**

Con la métrica seleccionada se observa que haciendo uso del framework se puede desarrollar una extensión equivalente al caso de uso de la prueba de concepto, con un 30% de las líneas de código requeridas para el desarrollo original.

Cubrimiento de tests

Se realizan diferentes tipos de tests en el ambiente presentados en el documento del Plan de Pruebas (Díaz & Sureda, 2022a). En dicho documento se presentan todos los tests realizados contra el framework, detallando por cada caso de uso los diferentes escenarios.

Se utiliza la herramienta Coverage.py (*Coverage.py*, 2022) considerando las operaciones básicas del framework y los cinco casos de uso desarrollados, cubriendo los escenarios habituales de éxito y falla.

Se presenta el resultado del cubrimiento obtenido con Coverage.py de una ejecución completa de los casos de prueba presentados. El cubrimiento total de los tests para el proyecto es un 87%.

```
Name                                                                    Stmts   Miss  Cover
-------------------------------------------------------------------------------------------
./vcdextension/__init__.py                                                 2      0   100%
./vcdextension/entities/__init__.py                                        0      0   100%
./vcdextension/entities/entity.py                                         39      0   100%
./vcdextension/entities/network.py                                        93     27    71%
./vcdextension/entities/snapshot.py                                       46      9    80%
./vcdextension/entities/tenant.py                                         43     11    74%
./vcdextension/entities/virtualapp.py                                     86     16    81%
./vcdextension/entities/virtualdatacenter.py                              65     24    63%
./vcdextension/entities/virtualmachine.py                                143     29    80%
./vcdextension/libraries/__init__.py                                       0      0   100%
./vcdextension/libraries/constants.py                                    108      0   100%
./vcdextension/libraries/exceptions.py                                    51      6    88%
./vcdextension/libraries/logger.py                                        22      1    95%
./vcdextension/libraries/rabbitmqclient.py                                74      6    92%
./vcdextension/libraries/serverutils.py                                   25      0   100%
./vcdextension/libraries/taskplanner.py                                   51      5    90%
./vcdextension/libraries/taskscheduler.py                                 51      6    88%
./vcdextension/libraries/vcloudsecurity.py                               229     55    76%
./vcdextension/logic/__init__.py                                           0      0   100%
./vcdextension/logic/logicorgs.py                                         11      0   100%
./vcdextension/logic/logictasks.py                                        14      0   100%
./vcdextension/logic/logicvapps.py                                        33      0   100%
./vcdextension/logic/logicvms.py                                          28      0   100%
./vcdextension/persistence/__init__.py                                     0      0   100%
./vcdextension/persistence/persistence.py                                 46      6    87%
./vcdextension/services/__init__.py                                        0      0   100%
./vcdextension/services/servicerabbitmq.py                                81      8    90%
./vcdextension/services/services.py                                       19      0   100%
./usecases/logic/logicautoscaling.py                                      50      2    96%
./usecases/logic/logicreplicatetasks.py                                   34      3    91%
./usecases/logic/logicresourcesreport.py                                  36      0   100%
./usecases/logic/logicscheduleplan.py                                     20      0   100%
./usecases/logic/logicsnapshotremoval.py                                  35      0   100%
./usecases/services/customservices.py                                     61      4    93%
./usecases/services/services.py                                           16      0   100%
./usecases/startcustomserver.py                                           10      0   100%
-------------------------------------------------------------------------------------------
TOTAL                                                                   1622    218    87%
```

Conclusiones y trabajo futuro

Conclusiones

Uno de los problemas que se enfrentó al comienzo del proyecto fue entender con precisión cuál era la propuesta y qué se pretendía obtener al final del mismo. La idea era crear un producto capaz de integrarse con el VMware Cloud Director (en su momento llamado vCloud Director) y poder resolver problemáticas que al día de hoy constaban de un proceso de varios pasos secuenciales y manuales. El objetivo final era obtener un software capaz de incluir soluciones para estos problemas, de tal modo que su resolución sea más eficiente para el usuario a cargo. A su vez, también se hizo un especial énfasis en la capacidad de extender el software para expandir sus capacidades a nuevos problemas contexto y tiempo dependientes.

A partir de la investigación inicial del estado del arte y posterior profundización en la materia, surgen dos posibles maneras distintas de enfocar la solución al problema. Se trabaja en una prueba de concepto para cada una, implementando una funcionalidad base y sencilla como ejemplo. También, de esta manera se logró validar la factibilidad de la solución al obtener como resultado una versión inicial operativa. Al final, con la experiencia de desarrollo de ambas maneras y versiones básicas e iniciales funcionando, se analizan las dos posibles vías y se elige implementar el producto a través de una solución desde cero, haciendo uso de los servicios que provee VMware. A grandes rasgos, esto permitiría tener un control más personalizado con la desventaja de tener que implementar funcionalidades que otros productos de VMware ya las incluía de manera nativa. Debido a la naturaleza del problema planteado, se presentaba un grado de innovación considerablemente alto y la ventaja de poder trabajar de manera más granular con los distintos componentes del sistema era fundamental.

Se comenzó con el desarrollo de la solución, optando por un modelo de desarrollo iterativo e incremental. Se dividió el proceso en sprints, en los que progresivamente se incluían más funcionalidades previamente planificadas. En reiteradas ocasiones, el producto se presentó al cliente a medida que avanzaba su desarrollo con el fin de obtener una retroalimentación del producto temprana y validar que sea acorde con la visión y necesidad del cliente.

Finalmente, la versión estable final del producto es una solución que se divide en dos partes principales. Por un lado, el frontend que se incluye como una extensión de la interfaz de VMware Cloud Director, y por otro lado el backend que es una aplicación que corre en un servidor independiente e implementa las funcionalidades del sistema y se comunica con los servicios que ofrece VMware Cloud Director. Se incluyen cinco casos de uso que fueron relevados a partir de profesionales que trabajan con VMware Director, con el propósito de implementar las herramientas bases necesarias que faciliten implementar e incluir nuevas funcionalidades en el futuro. Por ejemplo, la capacidad de comunicarse con la API de VMware Cloud Director a través de una cola de mensajería, poder agendar un conjunto de tareas a ser ejecutadas de manera programada en el futuro, persistir datos para la recuperación del sistema en caso de un fallo crítico, inclusión de una librería propia para el manejo de las entidades base

de VMware Cloud Director (como máquinas virtuales, vApps, etc.) que permite navegar entre sus relaciones, leer y escribir sus atributos, entre otros.

Se logró como resultado final una solución funcional que se integra con VMware Cloud Director a través de sus servicios expuestos para implementar las funcionalidades requeridas por los usuarios, a su vez permite el acceso de las mismas a través de una interfaz web integrada dentro del VMware Cloud Director. Como se comprobó en la sección Modificación de esfuerzo requerido, con el uso del framework se consigue desarrollar una extensión con aproximadamente un 70% menos de líneas de código. Sin embargo, todavía hay lugar para que el mismo siga evolucionando; en la siguiente sección se detallan varios puntos que serían de utilidad para el producto que por ciertas circunstancias y dado el marco de trabajo en el que se realizó, no se incluyeron en su última versión.

Trabajo futuro

La solución implementada en el proyecto cumple con los objetivos iniciales establecidos, sin embargo debido al tiempo y al marco de trabajo en el que se realizó el producto, se menciona una serie de puntos en los que la aplicación puede ser mejorada.

Para la verificación del sistema, es deseable tener a disposición un conjunto exhaustivo de pruebas unitarias que validen todas las funcionalidades de cada módulo en el backend. Por otro lado, para el frontend se pueden crear pruebas automatizadas que simulan la interacción del usuario en el sitio web; existen herramientas como Selenium que se dedican exclusivamente a ello.

Ciertas funcionalidades tienen el potencial de ser mejoradas, por ejemplo en el módulo de entidades en el backend se puede extender su funcionalidad y agregar la opción de crear entidades, se puede implementar utilizando el patrón de diseño factory. Luego, la función de autoescalado podría considerar trabajar también con el escalado horizontal de las compañías, por ejemplo creando más entidades de máquinas virtuales según corresponda en base a una configuración específica. El caso de uso de eliminación automática de snapshots se debe ajustar para que sea genérico a los Virtual Data Centers de una misma organización, y no específico por máquina virtual. Con identificar la organización del usuario será suficiente para que el sistema identifique y recorra todas las máquinas virtuales existentes de la organización, y para cada una de ellas deberá validar si tiene snapshots obsoletas para llevar a cabo su eliminación.

En el middleware, habilitar en la librería Pika la conexión SSL al servidor de RabbitMQ, en la actualidad se utilizan credenciales registradas en texto plano.

Otras funcionalidades misceláneas como reconexión automática del servidor para poder recuperarse ante una eventualidad de pérdida de línea serían deseables. Mostrar los mensajes de salida del backend en el idioma local ya que actualmente se muestran en inglés y podría ser

de importancia hacerlo acorde a la localidad del operador a cargo. Configurar el proceso del backend como un servicio del sistema operativo para iniciarlo automáticamente cuando el servidor es levantado.

Finalmente, sería deseable realizar algunos cambios en el frontend. Se debe actualizar la versión de Angular del proyecto. Debido a la versión del ambiente de VMware Cloud Director utilizado para el desarrollo, se tuvo que utilizar una versión de Angular que fuese compatible con el mismo. La diferencia es considerable entre la versión utilizada y la última estable existente de Angular, principalmente en términos de rendimiento, herramientas integradas, diseño de estructura base, seguridad, estabilidad, entre varios otros. Desde el lado de lo visual, se puede mejorar el diseño e implementación de la interfaz y experiencia de usuario. En particular, no fue uno de los focos principales en la etapa de implementación, y a su vez es una disciplina no dominada en el equipo.

Glosario

Docstring: En código Python se refiere al texto incluído en un comentario al principio de un módulo, de una clase o de un método. Están delimitados entre tres comillas dobles al principio y al final. Son utilizados para documentar el código. Utilidades de documentación como docutils (https://docutils.sourceforge.io/) reconocen las convenciones establecidas en el PEP 257 para automatizar la creación de documentación en línea.

Endpoint: Punto terminal de una comunicación entre dos extremos. En el presente documento se hace referencia al punto en donde el *backend* expone sus servicios y recibe los requerimientos desde el *frontend*. Dichos *endpoints* conforman la API, ya sea la del producto estándar o la de la extensión.

FODA: Su nombre proviene de las iniciales de fortalezas, oportunidades, debilidades, y amenazas. Es una herramienta de análisis para evaluar la situación general de una empresa, de una propuesta de negocio, un producto o de una solución específica para un problema. Considera ventajas y desventajas y analiza características internas (debilidades y fortalezas) y situación externa (amenazas y oportunidades).

Framework: Un framework de software tiene como objetivo proveer una base o marco de desarrollo para construir aplicaciones de software o extender productos ya existentes. La idea principal de un framework es brindarle a su usuario una forma estándar de desarrollar soluciones a partir de funcionalidades básicas o genéricas ya provistas por el framework. De esta forma el usuario se pueda enfocar en resolver su problema específico sin preocuparse en detalles técnicos previamente resueltos. Un framework puede incluir programas de soporte, librerías de código, APIs y otros componentes.

IDE: Sigla del inglés Integrated Development Environment, en español: entorno de desarrollo integrado. Es una aplicación que incluye diferentes funcionalidades orientadas a facilitar la tarea de desarrollar software. Suelen incluir un editor, un analizador de código, un intérprete, y otras funcionalidades como integración con herramientas de versionado. Algunos IDEs populares son Eclipse, Visual Studio, NetBeans, Atom o PyCharm.

PEP: Sigla del inglés Python Enhancement Proposal, propuesta de mejora de Python. Es un mecanismo para proponer mejoras y nuevas funcionalidades al lenguaje Python. Cada propuesta se publica documentando sus decisiones de diseño, y se identifica con un número, por ejemplo, las guías de estilo de código se encuentran en el PEP 8.

POO: Programación Orientada a Objetos. Es un paradigma de programación basado en los conceptos de clases (abstracciones) y objetos (entes concretos) que corresponden a dichas clases. Se dice que cada objeto es una instancia de la clase. El encapsulamiento, el polimorfismo y la herencia son algunas de sus características principales.

Provider: El proveedor es la compañía que ofrece una plataforma de servicios basada en infraestructura, aplicaciones y almacenamiento en la nube.

SDK: Sigla del inglés Software Development Kit. Están compuestas por herramientas de software integradas y presentadas de forma conjunta. Tienen como objetivo la creación de aplicaciones y suelen enfocarse en una plataforma, en la creación de un tipo específico de aplicación, o en la incorporación de determinadas funcionalidades. Ejemplos típicos de SDK

son Java Development Kit o Microsoft Windows SDK (anteriormente .NET). Los SDK utilizados para extender VMware Cloud Director son pyvcloud y vcd-ext-sdk.

Tenant: Usuarios finales (empresas o particuales) de los servicios en la nube brindados por los *providers*.

Referencias

AMQP Advanced Message Queuing Protocol. (2021). AMQP: Home. Retrieved December 31, 2021, from https://www.amqp.org/

Bose, M. (2021, July 29). *VMware vCloud Director – A Short Overview*. NAKIVO. Retrieved December 13, 2021, from https://www.nakivo.com/blog/an-introduction-to-vmware-vcloud-director/

Clarity Design System. (2021). Clarity Design System. Retrieved December 21, 2021, from https://clarity.design/

Container Service Extension (CSE). (2021). VMware. Retrieved December 19, 2021, from https://vmware.github.io/container-service-extension

Coverage.py. (2022). Coverage.py 6.3.2 documentation. Retrieved March 1, 2022, from https://coverage.readthedocs.io/

Díaz, A., & Sureda, M. (2021a). *Documento de Arquitectura de Software* [Arquitectura - Extensibilidad de herramienta de gestión para VMware Cloud Director].

Díaz, A., & Sureda, M. (2021b). *Extensión VMware Cloud Director* [Detalles de la Solución Técnica con pyvcloud & vcd-ext-sdk].

Díaz, A., & Sureda, M. (2022a). *Plan de Pruebas* [Tests Framework Extensión de VMware Cloud Director].

Díaz, A., & Sureda, M. (2022b). *Referencia Técnica de Uso* [Referencia Técnica de Uso para Framework de Extensión de VMware Cloud Director].

Extending VMware vCloud Director functionalities. (2019, May 6). vUptime.io. Retrieved December 22, 2021, from https://vuptime.io/post/2019-05-06-extending-vcloud-director-functionalities/

Fojta, T. (2018). *Architecting a VMware vCloud Director Solution*. VMware, Inc. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/vcat/vmware-architecting-a-vcloud-director-solution.pdf

Fojta, T. (2021). *Documentación de VMware Cloud Director*. VMware Docs. Retrieved December 14, 2021, from https://docs.vmware.com/es/VMware-Cloud-Director/index.html#extensiones-y-herramientas-de-vmware-cloud-director-3

Fojta, T. (2022). *Documentación de VMware Cloud Director*. VMware Docs. Retrieved February 13, 2022, from https://docs.vmware.com/es/VMware-Cloud-Director/index.html

Grönholm, A. (2021). *Advanced Python Scheduler - apscheduler.triggers.cron*. apscheduler.triggers.cron — APScheduler 3.8.1.post1 documentation. Retrieved January 23, 2022, from https://apscheduler.readthedocs.io/en/stable/modules/triggers/cron.html#module-apscheduler.triggers.cron

groupe-sii/lumext:. (2020, September 7). GitHub - LUMext is a vCD UI & API extension to manage LDAP-based organisation's users and groups through VMware vCloud Director. Retrieved December 22, 2021, from https://github.com/groupe-sii/lumext

HG. (2022). Somos una empresa fundada por Antel en el año 2001, especializada en la generación de proyectos de integración tecnológica, expandiéndonos más allá de los tradicionales ciclos de diseño, desarrollo y administración. Retrieved April 10, 2022, from https://hg.com.uy/

IBM - Infraestructura de IBM Cloud como servicio. (2021). IBM Infraestructura. Retrieved December 13, 2021, from https://www.ibm.com/es-es/cloud/infrastructure

IBM - Infrastructure as Code. (2019, December 2). IBM. Retrieved February 14, 2022, from https://www.ibm.com/cloud/learn/infrastructure-as-code

IBM - ¿Qué es cloud computing? (2021). IBM Cloud computing: Guía completa. Retrieved December 13, 2021, from https://www.ibm.com/es-es/cloud/learn/cloud-computing-gbl

IBM - What is an Application Programming Interface. (2020, August 19). IBM. Retrieved February 14, 2022, from https://www.ibm.com/cloud/learn/api

Johansson, L., & Vinka, E. (2020). *The Optimal RabbitMQ Guide* (2nd ed.). 84codes. https://www.cloudamqp.com/rabbitmq_ebook.html

Johnson, R., Helm, R., Vlissides, J., & Gamma, E. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.

Kombu Documentation. (2019). Kombu 4.6.1 documentation. Retrieved February 19, 2022, from https://kombu.readthedocs.io/

Kruchten, P. (1995). *Architectural Blueprints—The "4+1" View Model of Software Architecture*. Rational Software Corp. https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

Kubernetes Documentation. (2021, July 20). Kubernetes. Retrieved December 20, 2021, from https://kubernetes.io/docs/home/

Microsoft. (2022, January 16). *Azure DevOps Services*. Microsoft Azure. Retrieved January 16, 2022, from https://azure.microsoft.com/en-us/services/devops/

Microsoft - What is a Cloud Service Provider - Definition. (2021). Microsoft Azure. Retrieved December 13, 2021, from https://azure.microsoft.com/en-us/overview/what-is-a-cloud-provider/

Microsoft - What Is Cloud Computing? A Beginner's Guide. (2021). Microsoft Azure. Retrieved December 13, 2021, from https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/

PEP 8. (2022). PEP 8: The Style Guide for Python Code. Retrieved February 25, 2022, from https://pep8.org/

Pika documentation. (2017). Introduction to Pika. Retrieved February 19, 2022, from https://pika.readthedocs.io/en/stable/

Pressman, R. S. (2003). *Ingeniería del software: un enfoque práctico* (R. Ojeda Martín, V. Yagüe Galaup, & I. Morales Jareño, Trans.). McGraw-Hill.

PyCharm: the Python IDE for Professional Developers by JetBrains. (2022). JetBrains. Retrieved February 26, 2022, from https://www.jetbrains.com/pycharm/

PyPA documentation. (2021, December 27). Python Packaging Authority. Retrieved May 15, 2022, from https://www.pypa.io/en/latest/

Python. (2022, 02 04). *threading — Thread-based parallelism — Python 3.10.2 documentation*. Python Docs. Retrieved February 4, 2022, from https://docs.python.org/3/library/threading.html

pyvcloud documentation. (2021). Python SDK for VMware vCloud Director — pyvcloud documentation. Retrieved December 19, 2021, from http://vmware.github.io/pyvcloud/

Pyxis. (2021). Pyxis | Somos ecosistema. Creamos soluciones en 360° para organizaciones en múltiples industrias, de mediano y gran porte. Acompañamos el ciclo de vida de cualquier proyecto, con [...]. Retrieved December 19, 2021, from https://pyxis.com.uy/

setuptools. (2022). setuptools documentation. Retrieved May 15, 2022, from https://setuptools.pypa.io/en/latest/index.html

Terraform Registry. (2021). Terraform Registry - Docs overview | vmware/vcd. Retrieved December 19, 2021, from https://www.terraform.io/docs/providers/vcd/index.html

Using the vRealize Orchestrator Plug-in for vCloud Director. (2021). Retrieved December 19, 2021, from https://docs.vmware.com/en/VMware-Cloud-Director/10.0/using-the-vrealize-orchestrator-plug-in-for-vcloud-director/

vcd-cli. (2021). VMware vcd-cli | Command Line Interface for VMware vCloud Director. Retrieved December 19, 2021, from https://vmware.github.io/vcd-cli/

VMware Inc. - Take a Snapshot of a Virtual Machine. (2020, February 12). VMware Docs. Retrieved February 14, 2022, from https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.vm_admin.doc/GUID-9720B104-9875-4C2C-A878-F1C351A4F3D8.html

VMware Inc. - What is vRealize Operations? (2022). VMware. Retrieved February 14, 2022, from https://www.vmware.com/products/vrealize-operations.html

vmware/vcd-ext-sdk. (2021). GitHub: vCloud Director Extension SDK. Retrieved December 21, 2021, from https://github.com/vmware/vcd-ext-sdk

vRealize Operations Management Pack for vCloud Director. (2020, October 19). VMware Marketplace. Retrieved December 19, 2021, from https://marketplace.vmware.com/vsx/solutions/management-pack-for-vcloud-director

What is Angular? (2021, October 28). Angular. Retrieved February 21, 2022, from https://angular.io/guide/what-is-angular

What is vRealize Orchestrator (vRO)? | IT Automation Platform. (2021). VMware. Retrieved December 31, 2021, from https://www.vmware.com/products/vrealize-orchestrator.html

Ziadé, T., & Jaworski, M. (2019). *Expert Python Programming: Become a Master in Python by Learning Coding Best Practices and Advanced Programming Concepts in Python 3.7*. Packt Publishing.

Anexo A - Referencia técnica del framework

En el presente anexo se da una breve referencia técnica y una guía para el usuario del framework que quiera desarrollar una extensión de VMware Cloud Director. Hay varios aspectos que atender al para desarrollar una extensión que se consideran a continuación a forma de panorama general. Para una referencia completa con detalles de las etapas de diseño, desarrollo, y despliegue de la extensión consultar el documento Referencia Técnica de Uso (Díaz & Sureda, 2022b).

El plan de implementación de la extensión se compone de las siguientes etapas:

- Relevar requerimientos y análisis
    - Definir requerimientos de la extensión a desarrollar
    - Analizar factibilidad de implementación técnica con la funcionalidad brindada por el framework.
- Resolver prerrequisitos técnicos
    - Prerrequisitos Frontend
        - Instalación ambiente de desarrollo Typescript
    - Prerrequisitos Backend
        - Instalación Framework
            - Definir modo de instalación:
                - Modo package Python vcd-extension
                vs.
                - Modo de desarrollo
            - Habilitar ambiente virtual de desarrollo
- Diseño
    - Integración Frontend - Backend.
    - Definiciones técnicas de la extensión
        - URL de extensión
        - Integración RabbitMQ
        - Operaciones expuestas
        - Método(s) HTTP
- Desarrollo
    - Frontend
    - Backend
        - Integración con Framework
        - Capa de servicios
        - Capa de lógica
- Despliegue
    - Registro de URL de extensión en el servidor
    - Integración del frontend en el servidor
    - Inicio del servidor de backend

Todas estas actividades están descritas en detalle en el documento de referencia.

Anexo B - Plan de testing

En el presente anexo se detalla el plan de pruebas usado para validar la funcionalidad del framework. Para una referencia completa del plan de pruebas, incluyendo descripciones de cada uno de los escenarios testeados, consultar el documento Plan de Pruebas (Díaz & Sureda, 2022a).

Durante el desarrollo del producto, se realizaron varios tipos de pruebas: pruebas unitarias, pruebas de integración y pruebas de sistema. Para las pruebas unitarias, se realizaron principalmente análisis estáticos de código y pruebas de caja negra basados en los requisitos y procesos de negocios. Por otro lado, las pruebas de integración fueron realizadas para asegurarse que la interacción y comunicación entre los componentes del sistema sea la esperada. Y por último, las pruebas de sistema que son las que tienen como objetivo probar el sistema como un todo, fueron definidas en base a los requisitos recabados en la etapa de análisis del proyecto.

Las pruebas de integración se dividen para los subsistemas de frontend y backend. Para el subsistema de backend se desarrolla un script en Bash Shell llamado *test_curl.sh*. Dicho script, al inicio solicita las credenciales para autenticarse en el servidor de VMware Cloud Director. Luego de autenticarse, presenta las opciones del menú. Para cada opción disponible en el menú, el script solicita al usuario, si corresponde, los parámetros requeridos, luego genera un requerimiento a la extensión del backend con la herramienta cURL, y finalmente, presenta en la consola la respuesta recibida desde el framework.

A continuación se listan todos los escenarios de prueba del plan:

- Caso de uso: Programación de secuencias de actividades.
    - Escenarios de éxito:
        - Se incluyen múltiples tareas distintas a ser planificadas, con un tipo de entidad diferente asociada a las tareas. Planificación para días específicos de la semana, en hora y minutos específicos.
        - Similar al escenario previo. Se ingresa un momento del día específico con hora, minuto y segundo, y para los días, un día específico: año, mes y día.
    - Escenarios de falla:
        - entity_type inválido.
        - entity_id inválido.
        - objeto ‘schedule’ con datos de agendado inválido.
        - acción inválida.
        - Formato inválido del objeto JSON.
- Caso de uso: Escalado automático de recursos
    - Escenarios de éxito:
        - Comprobar el flujo del funcionamiento normal eligiendo el recurso CPU. Escenario donde no se alcanza el umbral para actualizar recursos.

        - Comprobar el flujo del funcionamiento normal eligiendo el recurso Memoria. Escenario donde no se alcanza el umbral para actualizar recursos.
        - Comprobar el flujo del funcionamiento normal eligiendo el recurso CPU. Escenario donde se alcanza el umbral para actualizar recursos.
        - Comprobar el flujo del funcionamiento normal eligiendo el recurso Memoria. Escenario donde se alcanza el umbral para actualizar recursos.
    - Escenarios de falla:
        - Recurso memoria: valor inválido new_memory.
        - Recurso CPU: combinación inválida de valores new_num_cpus y new_num_cores_per_socket.
- Caso de uso: Acciones aplicadas a varias organizaciones
    - Escenarios de éxito:
        - Comprobar el flujo del funcionamiento normal del caso de uso, incluyendo más de una organización y seleccionando como tipo de entidad una vApp.
        - Comprobar el flujo del funcionamiento normal del caso de uso, incluyendo más de una organización y seleccionando como tipo de entidad una máquina virtual.
    - Escenarios de falla:
        - Incluir una organización inexistente en la lista de organizaciones.
        - Lista de organizaciones vacía.
- Caso de uso: Eliminación automática de snapshots
    - Escenarios de éxito:
        - Comprobar el flujo del funcionamiento normal del caso de uso. Escenario donde se supera el umbral de retención y se elimina el snapshot.
        - Comprobar el flujo del funcionamiento normal del caso de uso. Escenario donde no se supera el umbral de retención y no se elimina el snapshot.
    - Escenarios de falla:
        - Escenario donde no existe snapshot asociado a la VM.
        - Escenario con período de retención inválido: valor no numérico.
- Caso de uso: Reporte de recursos por organización
    - Escenarios de éxito:
        - Comprobar el flujo del funcionamiento normal del caso de uso
    - Escenarios de falla:
        - Se envía una organización inválida en la URL.
- Otros escenarios de falla:
    - Error de autenticación en servidor VMware
    - Error de autenticación en servidor RabbitMQ
    - Método HTTP inválido al invocar *endpoint* de extensión.
    - Operación eliminar Job agendado: Identificador de Job inválido.
    - URL inválido dentro del prefijo de URLs de la extensión.

Anexo C - Tecnologías

En este anexo se listan las tecnologías y herramientas (software, librerías, SDKs, frameworks, etc.) necesarias para el desarrollo de la extensión, con una breve descripción de cada una. Cuando corresponde se indica la versión utilizada.

- APScheduler (Advanced Python Scheduler) 3.7.0: Librería Python que permite agendar rutinas de código Python para ejecutarse con la periodicidad indicada.
- Angular 4.4.4: Lenguaje utilizado para el desarrollo en el frontend.
- Azure DevOps: Gestión de proyecto, historias, issues, tareas.
- Bash shell: Desarrollo de tests de integración para el servidor de backend.
- Clarity Design: Sistema de estilos de diseño para el frontend.
- Coverage: Librería Python utilizada para medir cubrimiento de tests.
- cURL 7.64.0: Tests de integración del servidor de backend.
- Gilab (servidor Fing) + Git 2.20.1 + SSH key-gen: Repositorio de código y gestión de configuración de software. Key-gen se utiliza para generar las llaves y acceder mediante SSH al servidor sin necesidad de ingresar usuario y contraseña.
- Google Drive + Google Docs: Repositorio de documentos.
- lxml 4.6.2: Librería Python para análisis y procesamiento de XML.
- NPM 6.14.6: Node Package Manager.
- NodeJS 10.22.0: Runtime de frontend.
- Oracle VirtualBox + VM Linux debianvm 4.19.0-10-amd64: Entorno local de testing.
- OpenVPN: VPN para acceder al ambiente de pruebas. Instalado dentro de la VM Linux con Debian.
- PIP 9.0.1: Package Manager de Python.
- PickleDB 0.9.2: Librería Python que permite gestionar una base de datos tipo clave-valor en un archivo local.
- Pika 1.2.0: Librería Python con implementación del protocolo AMQP. Utilizada para interactuar con RabbitMQ.
- PyCharm 2021.3.2: IDE de desarrollo para Python.
- Python 3.7.3: Lenguaje utilizado para el backend.
- python-dateutil 2.8.1 & tzlocal 2.1& pytz 2021.1: Librerías Python para manejo de fechas, horas y husos horarios.
- Python venv v3.6.7-1: Entorno virtual de desarrollo para Python.
- PyYAML 5.4.1: Librería Python para procesamiento de archivos YAML.
- Redis 6.2.3: Motor de base de datos de tipo clave-valor. Librería Redis 3.5.3: Binding con lenguaje Python.
- SetupTools: Módulo de Python utilizado para construir y distribuir paquetes en Python.
- Sphinx: Generador de documentación automático.
- Twine 3.4.2: Módulo Python utilizado para subir el paquete generado al portal Python Package Index (PyPI).
- Yarn 1.22.5: Manejador de dependencias en el frontend.