Universidad CENFOTEC

Maestría en Ingeniería del Software con énfasis en Inteligencia Artificial

Documento final de Proyecto de Investigación Aplicada I y II

Propuesta de una arquitectura de software que permita “acoplar” y “desacoplar” componentes que agregan nuevas características al sistema Builds 2.0 de la empresa Wind River Systems

Araya Ruiz, Gabriel

Lee Pan, Felix

Diciembre, 2023

TRIBUNAL EXAMINADOR

Este proyecto fue aprobado por el Tribunal Examinador de la carrera: **Maestría Profesional en Ingeniería del Software con énfasis en Inteligencia Artificial Aplicada,** requisito para optar por el título de grado de **Maestría,** para los estudiantes: **Lee Pan Felix y Araya Ruiz Gabriel.**

RODRIGO SEBASTIAN HERRERA GARRO (FIRMA)
Digitally signed by RODRIGO SEBASTIAN HERRERA GARRO (FIRMA)
Date: 2024.03.18 18:33:45 -06'00'

M.Sc. Rodrigo Herrera Garro
Tutor

LUIS GUILLERMO ALVARADO QUESADA (FIRMA)
Firmado digitalmente por LUIS GUILLERMO ALVARADO QUESADA (FIRMA)
Fecha: 2024.03.20 10:25:05 -06'00'

MAU. Luis Guillermo Alvarado Quesada
Lector 1

FIRMADO DIGITALMENTE

M.Sc. Alex André Solis Barrantes
Lector 2

San José, Costa Rica, 4 de marzo de 2024

Firmada digitalmente, de conformidad con la Ley de Certificados, Firmas Digitales y Documentos Electrónicos N° 8454, destacando el artículo 9°-

Declaración de Derechos de Autor

Se declara que el siguiente documento fue realizado por Gabriel Antonio Araya Ruiz y Felix Lee Pan con cédula de identidad 1-1713-0144 y 1-1652-0058 correspondientemente.

Debido a la naturaleza del trabajo, y que la seguridad de la empresa en cuestión se puede ver vulnerada, se solicita que el documento se mantenga confidencial por un periodo de 5 años, con el fin de que, una vez se concluya el trabajo, las sugerencias sean aplicadas en la organización y no exista una brecha de información en este periodo.

Tabla de Contenidos

Tablas

Ilustraciones

CAPÍTULO I

INTRODUCCIÓN

1.1 Generalidades

La arquitectura de software como tema tiene muchas aristas, ya que su definición, al momento de construir un sistema, puede ser influida por muchos aspectos, como puede ser inexperiencia, sesgos, necesidades de negocio, entre otros, que afectan su correcta aplicación a largo plazo dependiendo de las características del software a construir o actualizar.

Dado lo anterior, conocer sobre las diversas arquitecturas de software que existen y sus casos de éxito, en la industria, a lo largo de los años, deben ser pilares cuando se tienen que tomar decisiones arquitectónicas en un software, por lo que el presente documento recopila y analiza la documentación existente, con el fin de proponer una arquitectura que apoye a la centralización, escalabilidad y mantenibilidad de sistemas ya existentes, para su mejora y adición de nuevas características a lo largo del ciclo de vida que posea.

1.2 Antecedentes del problema

El desarrollo rápido, para salir al mercado con un producto, es un tema muy común en las empresas de software actualmente, con el fin de ganar mercado y apostar rápido por una solución innovadora que pueda generar beneficios, para luego crecer con ese producto. Sin embargo, esa estrategia a largo plazo puede traer problemas de diseño en el software, haciendo que sea difícil y costoso el desarrollo de nuevas funcionalidades o de mejorar características del software actual.

Además, se debe tomar en cuenta que muchas veces los clientes de un producto de software (dependiendo de su contexto), pueden pedir soluciones personalizadas para atender alguna necesidad específica, lo que hace que la implementación de esas características, dentro del producto, sea desgastante para los desarrolladores por la arquitectura en la que fue desarrollado.

Aplicar incorrectamente los principios de ingeniería de software, especialmente en lo que respecta a la arquitectura de software, genera diversos problemas. Estos inconvenientes incluyen demoras en la implementación de software, lo que a su vez puede resultar en un producto final poco atractivo para los clientes y tener repercusiones económicas adversas.

En relación a todo lo anterior, la empresa Wind River Systems, objeto de estudio del presente documento, desarrolló el producto llamado Wind River Studio el cual ha sido una plataforma con mucho potencial para la empresa y un producto que crece exponencialmente desde su concepción, pero como se describe anteriormente, en ocasiones las necesidades de negocio tienen mayor importancia que la forma en la que se desarrolla un producto, por lo que algunos componentes del producto en mención cuentan con brechas de diseño que complica su mantenimiento a largo plazo.

1.3 Definición y descripción del problema

Para el presente documento se estudiará el diseño de los componentes de construcción de sistemas operativos de Wind River Studio, componentes que se generalizaron con el nombre de Builds.

Builds es una plataforma de desarrollo de software y sistemas operativos para dispositivos integrados (*embedded systems*). Está diseñado para ayudar a los desarrolladores a crear y desplegar sistemas integrados de forma más eficiente y rápida.

Builds incluye herramientas de desarrollo, depuración y análisis de sistemas integrados, así como un conjunto de sistemas operativos en tiempo real (RTOS) y de sistemas operativos de propósito general, así como distribuciones de Linux y el sistema operativo propietario de la empresa, VxWorks. También cuenta con características de virtualización y contenedores (utilizando otros componentes de Wind River Studio), lo que permite a los desarrolladores ejecutar varios sistemas operativos y aplicaciones en un solo hardware.

Builds es utilizado en una variedad de aplicaciones, así como en industrias como la automotriz, aeroespacial, de defensa, energía, telecomunicaciones, entre otras.

El problema en cuestión que enfrenta Wind River Systems con el componente Builds, es que en su transformación a futuro necesita que esta pieza clave de Wind River Studio posea características claves de un producto de software, como lo pueden ser el fácil mantenimiento del mismo y principalmente que su diseño se adapte de manera sencilla a componentes externos (sistemas de terceros, plugins, etc.) e internos (nuevos sistemas operativos, componentes diversos de Wind River Studio, etc.) para la extensión de características.

Para hacer frente a la problemática planteada anteriormente la empresa se encuentra en el desarrollo de Builds 2.0 (la evolución natural de Builds) y se deben buscar soluciones para poder ser ágiles, rápidos y eficientes en la manera en la que se extiende el *core* del sistema, esto con el fin de poder ofrecer características y funcionalidades a

los clientes de acuerdo a sus necesidades, con una arquitectura de software que permita "acoplar" o "desacoplar" dichas características al *core* del producto. Por consiguiente, se formula la siguiente pregunta de investigación en relación con Builds 2.0:

> *¿Cómo lograr ser ágiles, rápidos y eficientes al extender el core del sistema con nuevas características y funcionalidades presentadas por los clientes y la misma empresa, creando una arquitectura de software que permita "acoplar" y "desacoplar" dichos atributos?*

1.4 Justificación

La recopilación y análisis de bibliografía, tecnologías y casos de éxito de arquitecturas de software que se apeguen a lo esperado por la empresa Wind River Systems, permitirá realizar un estudio para poder ofrecer una propuesta de implementación para la interconexión de sistemas y extensibilidad de características de manera centralizada y sencilla, que permita solucionar el problema actual y sea una hoja de ruta para el desarrollo del producto Builds 2.0 y que le permita a la empresa dejar una huella en el mundo dado el impacto del producto en la industria y sus clientes.

1.5 Viabilidad

1.5.1 Punto de vista técnico

En este punto se cuenta con la experiencia de los encargados de realizar la investigación, primeramente en el ámbito de diseño y desarrollo de software y además, experiencia en calidad y pruebas de software.

Además, se contará con el apoyo de los arquitectos de la empresa, líderes técnicos de diversos equipos y gerentes de ingeniería que trabajan directamente con el producto que será objeto de la propuesta arquitectónica.

1.5.2 Punto de vista operativo

Además del apoyo directo de miembros importantes del proyecto dentro de la empresa, se cuenta además con acceso a las arquitecturas de software actuales y al código, tanto del *core* del sistema como de otros sistemas que serán los que se espera conectar, por medio de la propuesta de diseño de arquitectura.

Por lo que la viabilidad en el aspecto operativo, está cubierta por el apoyo de la empresa para sacar adelante la presente investigación.

1.5.3 Punto de vista económico

Desde el punto de vista de la viabilidad económica, se espera consultar bibliografía e información gratuita, con el fin de que cualquier persona que utilice el presente documento, como guía para otra investigación similar, pueda acceder a todas las referencias consultadas durante la investigación.

Además, no se prevén gastos desde el punto de vista de herramientas de apoyo o en aspectos relacionados con la empresa objeto de estudio.

Sin embargo, se recalca que cualquier gasto inesperado que pueda salir durante el desarrollo del presente documento, será asumido por los investigadores.

1.6 Objetivo general

Presentar a la empresa Wind River Systems una propuesta de diseño de una arquitectura de software que permita “acoplar” y “desacoplar” componentes que agregan nuevas características y funcionalidades.

1.6.1 Objetivos específicos

- Buscar bibliografías, tecnologías y casos de éxito de arquitecturas de software existentes que se apeguen a la interconexión de sistemas y extensibilidad de nuevas características.

- Seleccionar arquitecturas de software existentes y/o buenas prácticas que ayuden a resolver el nuevo diseño de arquitectura, apropiada para la interconexión de sistemas y extensibilidad de nuevas características.
- Analizar la estructura de las arquitecturas seleccionadas, las cuales ayuden a diseñar una nueva arquitectura que se apegue a la interconexión de sistemas y extensibilidad de nuevas características.
- Diseñar una nueva arquitectura de software que permita “acoplar” o “desacoplar” componentes que agreguen nuevas características y funcionalidades, utilizando el análisis de bibliografías, tecnologías y casos de éxito de arquitecturas de software existentes.
- Presentar la propuesta del diseño de la arquitectura de software a la empresa Wind River.

1.7 Alcances y limitaciones

- La investigación se enfocará únicamente en el diseño y propuesta de una arquitectura de software para “acoplar” o “desacoplar” componentes que agregan nuevas características y funcionalidades para la empresa Wind River Systems.

- El presente estudio abarca únicamente las bibliografías, tecnologías y casos de éxito de arquitecturas de software existentes que se apeguen a la interconexión de sistemas y extensibilidad de nuevas características.

- La arquitectura propuesta se centrará exclusivamente en los aspectos conceptuales y lógicos del software, evitando abordar cualquier consideración vinculada a la implementación, ya sea a nivel de hardware, arquitectura en la nube u otros elementos similares.

1.8 Marco de Referencia: Organizacional y Socioeconómico

1.8.1 Historia

Según G. Booch (2018, 108), los términos software e ingeniería de software han avanzado mucho, desde aspectos en donde apenas se comprende la programación como una cosa en sí misma a tener *frameworks*, patrones y muchos aspectos más. Se observa la evolución en donde las computadoras eran seres humanos y luego se tienen máquinas, que al inicio eran mucho más caras que las personas que las programaban y actualmente es, al contrario, el programador o el software mucho más importante.

Nacen diversos lenguajes de programación, diferentes perspectivas de entender el software, algunos desde el punto de vista científico otros más ingenieril. Nacen cuerpos de conocimiento que se actualizan con el pasar de los tiempos, para finalmente visualizar tecnologías como la Inteligencia Artificial (IA) y el cloud, que son algunos ejemplos que apoyan como el software e ingeniería de software siguen avanzando, todo lo anterior se apega a la evolución que también han tenido las arquitecturas de software que actualmente apoyan a la mantenibilidad, escalabilidad y diseño de sistemas (Booch, 2018, 108).

Aunado a lo anterior, la empresa Wind River Systems, fundada en 1981, se ha destacado en estar en la vanguardia del desarrollo de software principalmente enfocado en los sistemas embebidos y sistemas operativos en tiempo real, virtualización entre otros.

En su historia ha tenido hitos como el desarrollo del sistema operativo VxWorks, sus diversas distribuciones de Linux, sus relaciones comerciales con entidades como la NASA, su compra por parte de Intel y su reciente adquisición por parte de la empresa del sector automovilístico Aptiv, consagrando sus tecnologías y además ampliando las oportunidades de negocio a futuro como parte de la empresa antes mencionada.

1.8.2 Tipo de negocio y mercado meta

La empresa se dedica principalmente a ofrecer software y servicios para sistemas operativos en tiempo real, sistemas embebidos y virtualización, teniendo en cuenta principalmente los sistemas operativos VxWorks y sus distribuciones de Linux. Servicios que han evolucionado al producto Wind River Studio, por lo que muchos de los aspectos mencionados se pueden aplicar a diversas industrias como puede ser la industria automotriz, médica, aeroespacial, redes de comunicación, defensa entre otros.

1.9 Revisión sistemática de la literatura y estado de la cuestión

Con el fin de abordar el tema de la interconexión de sistemas y la extensibilidad de nuevas características en la propuesta de este documento se debe de buscar investigaciones, tecnologías y casos de éxitos de arquitectura de software existentes. Por lo que se realizará una investigación exhaustiva para identificar fuentes primarias y relevantes que proporcionen información sobre este tipo de arquitecturas de software que tenga la capacidad de incorporar nuevas funcionalidades de manera flexible.

A continuación, se listan las siguientes investigaciones que se desarrollaron para el diseño de una arquitectura de *Plug-and-Play* en donde resuelven diferentes tipos de problemas:

1.9.1 Aplicación de la arquitectura orientada a servicios universal *Plug-and-Play*

En Valencia, España, se realizó una investigación en el Instituto Universitario de Automática e Informática Industrial de la Universidad Politécnica de Valencia, donde se aplica una arquitectura universal orientada a servicios, *Plug-and-Play,* para facilitar la integración de Robots Industriales en líneas de producción.

Esta investigación se basa en la integración de diferentes equipos y dispositivos en células robotizadas industriales con tecnología de interfaz de Ethernet y dispositivos de bajo coste que permite desarrollar soluciones potentes e inteligentes. Sin embargo,

programar eficientemente estos dispositivos requiere conocimientos específicos sobre sus arquitecturas de *hardware*, lenguajes de programación dedicados y protocolos de comunicación de bajo nivel. Por otra parte, esta investigación utiliza la arquitectura Universal *Plug-and-Play* (UPnP), que se considera una de las más adecuadas para las células robotizadas. Mediante el uso de esta arquitectura, se ha desarrollado un banco de pruebas basado en dos robots industriales. Los resultados obtenidos demuestran que el uso de esquemas integrados basados en arquitecturas orientadas a servicios reduce los tiempos de integración y se adapta bien a la integración de células robotizadas industriales (Valera et al., 2012, 24-31).

1.9.2 Diseño y verificación de la arquitectura de *Plug-and-Play*

En la Universidad de Massachusetts Amherst, Estados Unidos, un grupo de ingenieros realizan una investigación acerca del diseño arquitectural de "*Plug-and-Play*", en el que los componentes representan las unidades computacionales de un sistema y los conectores representan las interacciones entre esas unidades. Tomar decisiones sobre la semántica de estas interacciones fue una parte clave del proceso de diseño. Sin embargo, fue difícil elegir la semántica de integración adecuada debido a la amplia gama de alternativas y a la complejidad del comportamiento del sistema afectado por esas elecciones. Técnicas como la verificación de estados finitos se pueden utilizar para evaluar el impacto de las decisiones de diseño en el comportamiento general del sistema (Wang et al., s.f.).

Esta investigación presenta el enfoque "*Plug-and-Play*" que permite a los diseñadores experimentar con opciones de diseño alternativas de las interacciones entre componentes de manera fácil y rápida. Con este enfoque, los conectores que representan semánticas de integración específicas se componen a partir de una biblioteca de bloques de construcción predefinidos y reutilizables. Además, se definen interfaces estándar para los componentes que reducen el impacto de los cambios de interacción en los cálculos de los componentes. Este enfoque, según la investigación, facilita la verificación en tiempo de diseño al mejorar la reutilización de los modelos de componentes y al proporcionar modelos formales reutilizables para los bloques de

construcción de conectores, reduciendo así el tiempo de construcción del modelo para la verificación de estados finitos (Wang et al., s.f.).

1.9.3 El talón de Aquiles de las arquitecturas de software *Plug-and-Play*

En el Instituto de Rochester de Tecnologías, Nueva York, Estados Unidos se realiza un estudio empírico sobre las vulnerabilidades en las arquitecturas de *Plug-and-Play* (2019), que permiten agregar o eliminar funcionalidades adicionales de un sistema durante su tiempo de ejecución a través de interfaces bien definidas. Sin embargo, la adición de complementos puede aumentar los ataques de una aplicación o introducir comportamientos no confiables en el sistema.

En este estudio, utilizaron un enfoque basado en la teoría fundamentada para realizar un estudio empírico de las vulnerabilidades comunes en arquitecturas de software *Plug-and-Play*. Además, realizaron también una revisión sistemática de la literatura y evaluaron si los resultados del estudio están respaldados por la literatura existente.

En conclusión, este estudio analizó varias aplicaciones y sistemas que utilizan este tipo de arquitecturas y encontraron un total de 303 vulnerabilidades relacionadas con complementos, causadas por 16 tipos diferentes de vulnerabilidades en las decisiones de diseño de extensibilidad. Por otra parte, se identificaron 19 procedimientos de mitigación para corregirlas y la literatura respaldó 12 tipos de ellas. Además, se descubrieron 4 tipos de vulnerabilidades y 11 técnicas de mitigación que no se encontraron en la literatura revisada. Todos los resultados fueron confirmados por profesionales y resaltaron cómo estos resultados pueden tener un impacto en la práctica (Santos et al., 2019).

Este estudio, proporciona una visión general de los tipos de vulnerabilidades en este tipo de arquitectura, sin embargo, también explica diferentes tipos de técnicas de mitigación que se pueden poner en práctica durante el desarrollo de la propuesta de la presente investigación.

1.9.4 Arquitectura orientada a servicios de amplia área

En el Laboratorio Key de Beijing de Investigación y Evaluación de Sistema de Tecnología de Automatización de Despacho de Energía, Beijing, China (2018) se realizó una investigación de un diseño de arquitectura orientada a servicios de amplia área para la conexión y reproducción de equipos de la red eléctrica, que adopta el mecanismo de proxy de servicios para gestionar y controlar el dominio de servicios que se accede a través de la red de área amplia mediante el centro de gestión de dominios.

Por consiguiente, el centro de gestión controla el registro, el direccionamiento y la información de servicios, también proporciona funciones de acceso y comunicación sin mantenimiento, gestión de objetos de servicios, suscripción y distribución entre proveedores y consumidores en interacciones de amplia área.

Como resultado la arquitectura propuesta es capaz de lograr la interacción de servitización de aplicaciones de red eléctrica y mejorar el nivel de intercambio de información y la capacidad de colaboración empresarial entre la estación principal de despacho y la subestación eléctrica inteligente. También puede proporcionar una guía práctica para el acceso automático de equipos de subestación, la mejora de funciones de la estación de control principal y la transformación de servicios de sus aplicaciones (Li et al., 2018, 353-357).

En conclusión, se abordan diferentes investigaciones y desarrollos relacionados con arquitecturas que acoplan y desacoplan funcionalidades externas al sistema central, en donde se puede afirmar que teniendo este tipo de arquitectura, puede facilitar la integración de distintos complementos añadiendo nuevas funcionalidades al sistema, además de la reutilización de ellas y obteniendo una mejora en la eficiencia con respecto al agregar o añadir nuevos elementos con comportamientos distintos al sistema central.

1.9.5 Arquitectura basada en *plugins* para el desarrollo de software científico

El Centro de Aplicaciones de Tecnologías Avanzada de Cuba (2013), desarrolla una arquitectura basada en *plugins* para el desarrollo de software científico, de la cual

los especialistas dedican la mayor parte del tiempo analizando, diseñando e implementando elementos de lógica de negocios.

En este tipo de aplicaciones, muchos de estos elementos no cambian entre proyectos. Sería muy conveniente minimizar el tiempo requerido para desarrollar estos elementos y maximizar el tiempo necesario para la implementación de algoritmos, que son los objetivos principales de estas aplicaciones. En conclusión, este artículo propone una arquitectura basada en *plugins* para el desarrollo de software científico, además permite la reutilización de elementos de lógica empresarial y pone la implementación eficiente de algoritmos en el centro del proceso de desarrollo (Bustio Martínez et al., 2013).

CAPÍTULO II

MARCO CONCEPTUAL

2.1 Diseño de arquitectura de software

Según el ingeniero Cervantes, profesor investigador en la UAM-Iztapalapa, México (s.f.), el concepto de la arquitectura de software siempre ha sido malentendido con respecto a sus principios en la que frecuentemente repercute de manera negativa en la construcción de sistemas de software.

El profesor, enfatiza que este concepto realmente se refiere a la estructuración del sistema creada en etapas tempranas del desarrollo, y además, este diseño representa un diseño de alto nivel del sistema que tiene 2 objetivos principales, las cuales son: "*satisfacer los atributos de calidad en la que incluye desempeño, seguridad y modificabilidad, y servir como guía de desarrollo*" (Cervantes, s.f.)

2.1.1 Importancia de la arquitectura de software

La arquitectura de software es crucial para determinar la capacidad de un sistema y visualizar si cumple con los atributos de calidad. La forma en que se estructura un sistema puede facilitar o dificultar el cumplimiento de estos.

Por otra parte, desempeña un papel fundamental en la guía del desarrollo en la que se puede visualizar los componentes del sistema de forma individual. De esta manera, se puede identificar riesgos de forma temprana.

Además, estos diseños arquitectónicos desarrollados en una organización pueden ser reutilizados para crear sistemas diferentes, lo que resulta en una reducción de costos y un aumento de la calidad, especialmente si estos diseños han sido exitosos en sistemas anteriores (Cervantes, s.f.).

2.2 Arquitectura de servicio universal *plug-and-play*

Este tipo de arquitectura es muy utilizado en la actualidad y se encuentran en la mayoría de los sistemas, sin embargo, muy pocas personas se dan cuenta de su existencia. No solo existe en aplicaciones basadas en productos, así como *Eclipse*, una plataforma de desarrollo creada para ser extendida indefinidamente por medio de *plug-*

ins (Calendamaia, 2014), o cualquier navegador en donde se puede personalizar agregando complementos y extensiones, sino que también existe en aplicaciones basadas en negocios donde las reglas comerciales y la lógica de procesamiento de datos varían.

La idea principal de la arquitectura es simple, ser capaz de conectar nuevas características a un componente existente sin que este componente tenga que conocer los detalles de implementación de estas características conectadas.

Según el Ing. Omar Elgabry (2019), la arquitectura de *plug-in* consta de dos componentes: un sistema centralizado y módulos de *plug-in*. Tal y como lo menciona el señor Elgabry, la clave del diseño es permitir la conexión de características adicionales como complementos a la aplicación principal, proporcionando extensibilidad, flexibilidad y aislamiento de las características de la aplicación y la lógica de procesamiento de datos.

Además, las reglas específicas y el procesamiento están separados del sistema central, por lo que en cualquier momento dado, se pueden añadir, remover y modificar componentes existentes con poco o ningún efecto en el resto del sistema central u otros módulos que los complementan (Elgabry, 2019).

2.2.1 Sistema central

Este se puede definir como la funcionalidad central del sistema y la base de la lógica de negocios. Es decir, un sistema totalmente abstracto en el que no existe ninguna implementación previa o específica.

Un ejemplo bastante claro es tener un flujo de trabajo genérico que define cómo la aplicación debe trabajar, pero la implementación y los pasos involucrados dentro de ese flujo son proporcionados por el complemento. Esto significa que cualquier funcionalidad que se interconecte con el sistema central, debe seguir ese flujo genérico que provee el mismo.

Por otra parte, también se deben considerar casos específicos que necesitan otro tipo de flujo, definición, o un proceso complejo que se requiere para interconectar con el sistema central.

Adicionalmente, el sistema central también debe proveer funcionalidades comunes que pueden ser reutilizadas por los complementos interconectados. Tales como el registro del historial de transacciones y fallos en el sistema. De esta manera, se evitarán códigos duplicados o repetitivos (en Inglés conocido como "*boilerplate code*") y tener una única fuente verídica.

2.2.2 Plugins

Los complementos (en inglés conocido como "*plug-ins*") son totalmente autónomos y funcionan de forma independiente en la que contienen procesamientos específicos, características adicionales y código personalizado que pretende mejorar o extender el sistema central con el fin de producir capacidades adicionales.

Generalmente, estos complementos son independientes de otros módulos, sin embargo, puede que exista alguno que necesite datos de otros o por lo menos asumir que existe un módulo en específico para funcionar de forma correcta. Por lo que es importante mantener la comunicación y conocer las dependencias entre los complementos lo mínimo posible.

2.2.3 Implementación de la arquitectura

El profesional Joseph Gefroh, ingeniero informático, explica paso a paso cómo diseñar un sistema que aplique una arquitectura de servicio universal *Plug-and-Play*. En donde explica, al inicio del artículo, que un "*plugin*" es un sistema que puede venir en diferentes formas, para ello, lo explica en la siguiente ilustración:

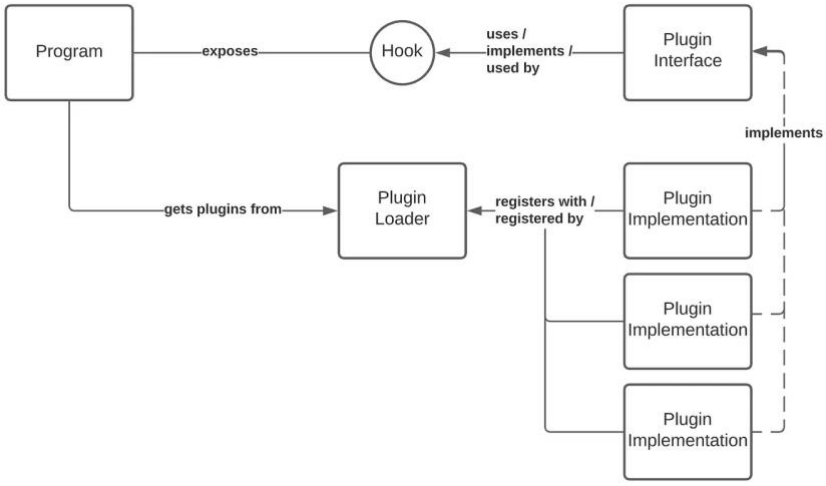

Ilustración 1 Componentes conceptuales de un sistema básico de "plugins"
Fuente: Gefroh, 2021

En el artículo, el señor Gefroh explica que la pieza principal es el "programa", tal y como se puede ver en la ilustración 1, el cual representa el sistema que se desea extender con nuevas funcionalidades que le darán otro tipo de comportamiento al sistema central. Sin embargo, este sistema central no tiene necesidad de saber quién, qué o cuáles funcionalidades contiene el "*plugin*" (Gefroh, 2021).

Seguidamente, explica la siguiente pieza que se llama en inglés como "*hook*" de la ilustración 1, que en términos de informática significa un conjunto de métodos predefinidos que no tiene un comportamiento como tal hasta que sean extendidos y definidos por un "*plugin*" (Gefroh, 2021).

Finalmente, el señor Gefroh menciona que el "*plugin*" no funciona si nadie llega a ejecutarla, por lo que se necesita un "*loader*" que es el encargado de cargar todas las funcionalidades para iniciar con la ejecución del mismo y existen 2 técnicas que realizan esta extensibilidad (Gefroh, 2021):

1. **"*Plugin-driven*":** técnica de la cual el "*plugin*" conoce cómo acceder el programa y registrarse por sí mismo. Es decir, el "programa" provee un método en el que el "*plugin*" puede extenderse por sí mismo.

2. **"*Program-driven*":** técnica de la cual el "programa" conoce dónde encontrar los "*plugins*" y cargar todo lo que se encuentra dentro de ellas. Es decir, el "programa" busca y encuentra el "*plugin*" y carga todo el contenido del mismo.

Utilizando estas técnicas, el sistema podrá extenderse de forma sencilla y rápida, manteniendo aisladas y segmentadas las características de cada uno de los "*plugins*" sin ser afectados por otras.

2.2.4 Ventajas y desventajas de la arquitectura *plug-and-play*

En este apartado, se explorará y se analizará las ventajas de la arquitectura como la flexibilidad, la reutilización de componentes y la facilidad de actualización. Sin embargo, también se abordarán las desventajas potenciales como la complejidad en la gestión de versiones, la dependencia de terceros y la posible falta de control sobre la calidad y seguridad de los componentes *plug-and-play* (Esterkin, 2020).

Ventajas	**Desventajas**
La arquitectura *plug-and-play* es la mejor manera de agregar funcionalidades particulares a un sistema que inicialmente no fue diseñado para admitirlas.	Los *plugins* pueden ser una fuente de virus y ataques por parte de actores externos.
Esta arquitectura elimina los límites en la cantidad de funcionalidades que una aplicación puede tener. Se pueden agregar número infinito de *plugins*	Tener muchos *plugins* en una aplicación puede afectar su rendimiento.

Tabla 1 Ventajas y desventajas de la arquitectura plug-and-play
Fuente: Esterkin, J. (2020)
Elaboración propia

Al comprender tanto los beneficios como las limitaciones de esta arquitectura, se pueden evaluar de manera más precisa sus usos en diferentes escenarios y contextos de desarrollo de software.

2.2.5 Usos prácticos de la arquitectura

Según Esterkin, algunos de los casos perfectos que se pueden utilizar este tipo de arquitectura *plug-and-play* son las siguientes:

1. Encriptación de software: un sistema que encripta un texto utilizando una llave secreta. El emisor encripta el mensaje con una llave secreta y el receptor lo desencripta asumiendo que él tenga la llave. Este tipo de comportamiento se puede observar en las aplicaciones “SendSafely” y “Mailvelope” en donde el proceso de encriptación funciona de forma automática, es decir oculta para el usuario. El usuario escribe el correo y éste es encriptado por un *plugin*.
2. *Plugin* de información local de un sitio web: este es un *plugin* que obtiene información de un sitio web y lo muestra en otra ventana del navegador. Como por ejemplo, *Rapportive* que conecta la red social *LinkedIn* en el navegador, o *Trello* que también se conecta directamente al navegador.

2.2.6 Archivo de manifiesto

En relación con la arquitectura *plug and play,* el archivo manifiesto es un concepto para tener en cuenta, el cuál según Microsoft, se define como un archivo de metadatos que determina un componente en específico. Por lo general, son archivos XML (en inglés conocido como *Extensible Markup Language*, traducido en español como Lenguaje de Marcas Extensible) que se pueden describir como un espacio para definir nombres, tipos de datos, configuraciones, propiedades, recursos, bibliotecas o librerías, entre otras en la que actúa como una interfaz que llega ser aplicado por un componente.

En el momento que un nuevo componente desea acoplarse a una aplicación central, este tipo de archivo debe de ser configurado y personalizado. Durante la ejecución de este, los datos del manifiesto filtran los otros componentes validos existentes en la aplicación central, con el fin de que estén disponibles para el contexto de la ejecución.

Estas propiedades definidas para el componente en el manifiesto se generan como espacios de configuraciones con el fin de que el usuario pueda especificar los valores respectivos que serán utilizados en tiempo de ejecución.

```
<?xml version="1.0" encoding="utf-8" ?>
<manifest>
   <control namespace="MyNameSpace"
      constructor="JSHelloWorldControl"
      version="1.0.0"
      display-name-key="JS_HelloWorldControl_Display_Key"
      description-key="JS_HelloWorldControl_Desc_Key"
      control-type="standard">
      <property name="myFirstProperty"
         display-name-key="myFirstProperty_Display_Key"
         description-key="myFirstProperty_Desc_Key"
         of-type="SingleLine.Text"
         usage="bound"
         required="true" />
      <resources>
         <code path="JS_HelloWorldControl.js"
            order="1" />
         <css path="css/JS_HelloWorldControl.css"
            order="1" />
      </resources>
   </control>
</manifest>
```

Ilustración 2 Ejemplo de un archivo de manifiesto
Fuente: Microsoft, 2023

2.3 Arquitecturas de referencia

Adicionalmente a la arquitectura que referencia directamente a conexiones *plug-and-play* se pueden encontrar más arquitecturas de software que tienen bases de construcción similares que pueden apoyar a construir sistemas escalables y mantenibles a largo plazo, entre ellas se encuentran la arquitectura de software hexagonal propuesta en el año 2005 por el Dr. Alistair Cockburn, co-autor del manifiesto ágil (Beck et al., 2001), la arquitectura de software de cebolla propuesta en el 2008 por Jeffry Palermo y finalmente la arquitectura limpia publicada en el 2017 por el también co-autor del manifiesto ágil, Robert C. Martin.

2.3.1 Arquitectura Hexagonal

La arquitectura hexagonal también llamada arquitectura de puertos y adaptadores, es una arquitectura que tiene como propósito “Permitir que una aplicación

sea ejecutada indistintamente por usuarios, programas, pruebas automatizadas, o archivos batch; y que sea desarrollada y probada por separado, sin los posibles dispositivos [*sic*] y bases de datos de los que dependa en tiempo de ejecución.” (Garrido de Paz, 2021).

Esta arquitectura busca desacoplar las diferentes responsabilidades de una aplicación usando un núcleo central que posea el dominio de la aplicación la cual no debe poseer dependencias, seguidamente se definen puntos de entrada o puertos y adaptadores para que otros módulos puedan extender al núcleo central como por ejemplo, bases de datos, testing automatizado entre otros.

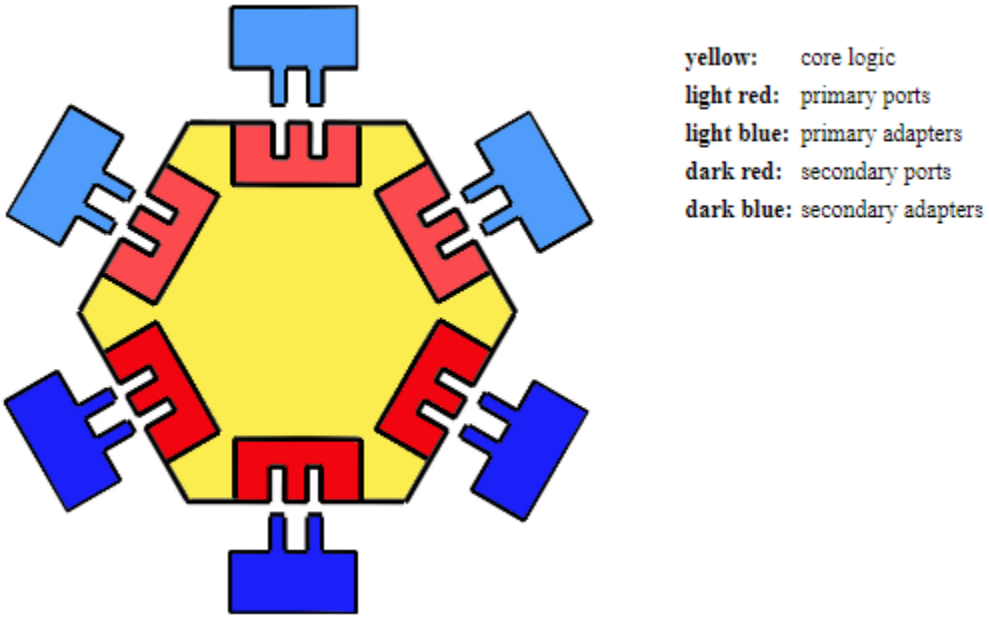

Ilustración 3 Visión general de la arquitectura hexagonal
Fuente: Andreas, s.f.

2.3.2 Arquitectura de cebolla

La arquitectura de cebolla o por capas, es una arquitectura inspirada en la arquitectura hexagonal, por lo que su propio autor expresa lo siguiente “Propongo un nuevo enfoque de arquitectura, aunque honestamente, no es completamente nueva” (Palermo, 2008).

Esta arquitectura, al igual que la propuesta por el Dr. Alistair Cockburn, sigue la premisa de “Externalizar la infraestructura y escribir el código del adaptador para que la infraestructura no se acople estrechamente” (Palermo, 2008) y además busca que todo acoplamiento sea hacia adentro y nunca a capas alejadas, por lo que la capa de dominio depende de sí misma.

En resumen, la arquitectura define que en el núcleo se encuentra el dominio de la aplicación, las siguientes capas serían adaptadores o interfaces y finalmente, en las capas exteriores se encontrarán los módulos o implementaciones menos genéricas que extienden las funcionalidades del núcleo como puede ser la base de datos, interfaces de usuario u otras implementaciones.

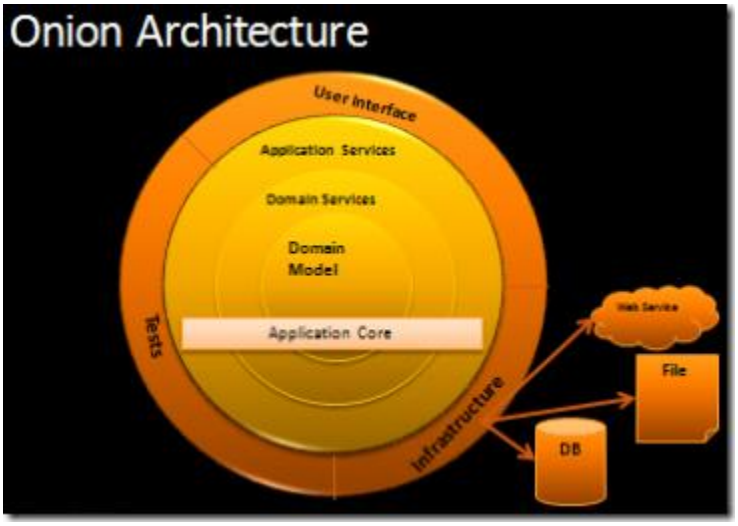

Ilustración 4 Visión general de la arquitectura de cebolla
Fuente: Palermo, 2008

2.3.3 Arquitectura limpia

La arquitectura limpia publicada en el libro “Clean architecture: A craftsman's guide to software structure and Design” se inspira igualmente en arquitecturas propuestas anteriormente incluida la arquitectura de puertos y adaptadores, en donde se busca una separación de responsabilidades y además la división de la aplicación en

diferentes capas, logrando que el software resultante presente las siguientes características:

- Independencia de frameworks o librerías
- Testabilidad
- Independencia de la interfaz de usuario
- Independencia de la base de datos
- Independencia de cualquier agente externo

La arquitectura presentada por Robert C. Martin, más que un patrón a seguir son recomendaciones hacia un modelo de construcción de software sostenible, en donde se hace hincapié en que las capas más centrales del software no conocen nada sobre las capas exteriores, contando con las capas de entidades, casos de uso, adaptadores y módulos externos, siguiendo con la línea de centralizar el dominio de la aplicación e ir construyendo adaptadores para la comunicación externa, dando como principal ventaja que si una pieza externa se vuelve obsoleta o se necesita renovar por alguna regla de negocio, como por ejemplo una base de datos o una interfaz de usuario, esta pueda ser reemplazada con una intervención mínima.

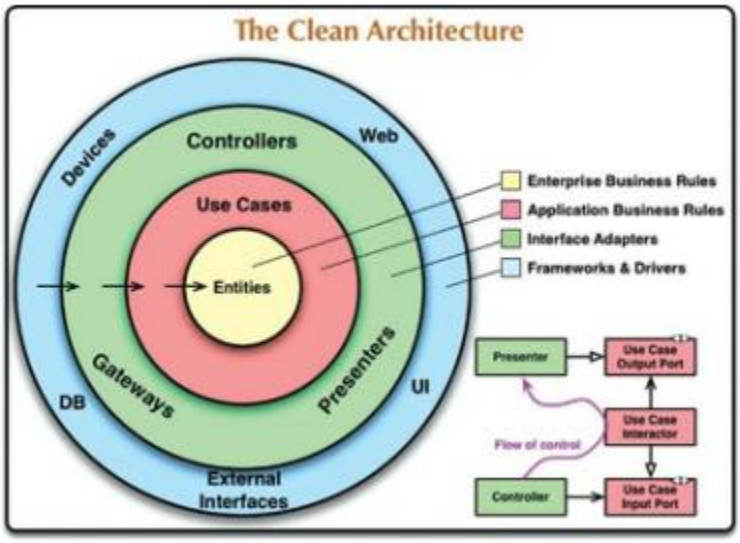

Ilustración 5 Visión general de la arquitectura limpia
Fuente: C. Martin, 2018

2.3.4 Arquitectura Microservicios

Esta arquitectura se compone de una colección de servicios pequeños y autónomos en donde cada uno es completamente independiente con su lógica de negocio implementado dentro de un contexto delimitado. (Microsoft, s.f.)

Los servicios son responsables de comunicar sus propios datos con otros. Esto diferencia del modelo tradicional, en donde una capa de datos independiente maneja la persistencia de los datos.

Para comunicarse entre sí, generalmente estos utilizan *APIs*, en la cual ocultan los detalles de la implementación interna de cada uno de los servicios.

Es importante mencionar que esta arquitectura permite la programación políglota. Es decir, los servicios no tienen que compartir la misma tecnología, librerías, entre otras.

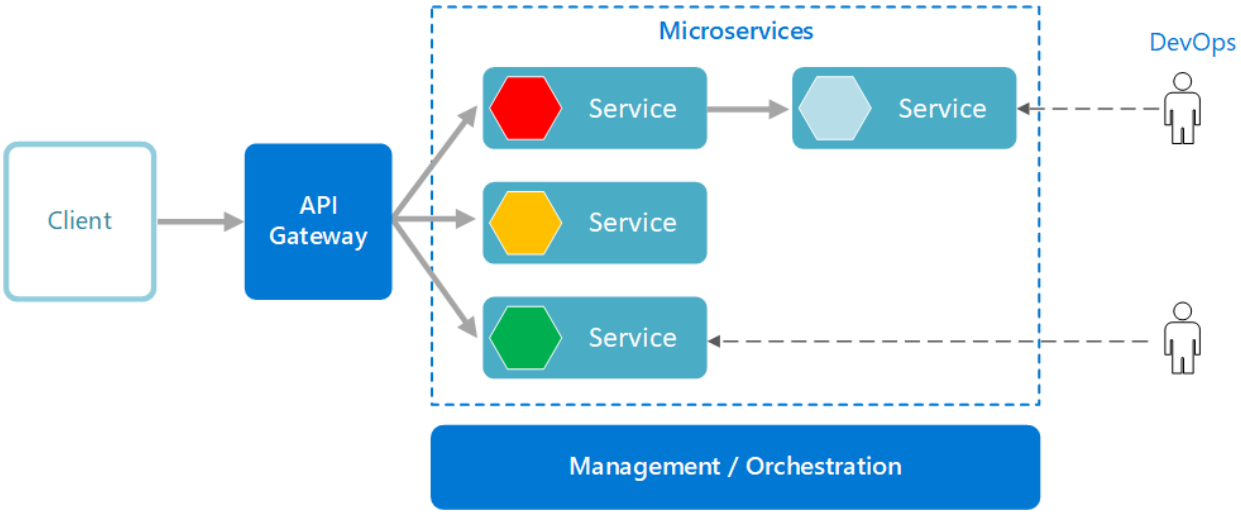

Ilustración 6 Estilo de Arquitectura de Microservicios
Fuente: Microsoft, s.f.

Según Microsoft, esta arquitectura se compone de 2 partes importantes, tal y como se muestra en la ilustración 6 identificada con el color azul, la administración, y la puerta de enlace API.

- Administración: este componente se encarga de implementar y colocar cada uno de los microservicios en nodos, detección de errores, escalabilidad, entre otras.

Por lo general, se utiliza una tecnología común, así como una plataforma de contenerización como *Docker* administrada por *Kubernetes* que los ejecuta y gestiona su tiempo de ejecución.

- Puerta de enlace API: este componente es el punto de entrada de comunicación entre los microservicios y clientes. Esto evita que la comunicación sea directa a los servicios.

Algunas ventajas de implementar este tipo de arquitectura son las siguientes:

- Gracias a que los microservicios se implementan de forma independiente, la administración de los errores y las versiones se vuelven aún más fácil, ya que se puede actualizar o revertir un solo servicio sin volver a implementar toda la aplicación.
- Como los microservicios no comparten código, minimizan las dependencias y resulta más fácil agregar nuevas características.
- Este tipo de arquitectura no interrumpe toda la aplicación si uno de los servicios no está disponible o tiene fallos.
- Permite la escalabilidad horizontal de los subsistemas que requieren más recursos, sin la necesidad de escalar toda la aplicación. Todo esto, mediante un orquestador.

Si se realiza una comparación de esta arquitectura con una monolítica, esta se compila como una unidad unificada y de forma autónoma e independiente de otras aplicaciones. Por otro lado, una arquitectura de microservicios es totalmente opuesto, este se basa en una variedad de servicios implementados de forma independiente.

Si bien se sabe que la arquitectura monolítica puede ser práctica al inicio de un proyecto, sin embargo, cuando este proyecto se vuelve grande y compleja, la escalabilidad se convierte en un desafío.

CAPÍTULO III

MARCO METODOLÓGICO

3.1 Tipo de Investigación

3.1.1 Investigación Aplicada

Según la docente de la Maestría en Orientación de la Universidad de Costa Rica, Zoila Vargas (2009), una investigación aplicada se entiende como la forma en cómo se utilizan los conocimientos adquiridos durante la práctica. Además, esta investigación también recibe el nombre de “investigación práctica o empírica” ya que busca la aplicación de lo investigado y genera resultados de una forma rigurosa, organizada y sistemática de conocer la realidad.

Por lo tanto, en una investigación aplicada, según la Sra. Vargas, es “aquel tipo de estudios científicos orientados a resolver problemas de la vida cotidiana o a controlar situaciones prácticas” (Vargas Cordero, 2009, p. 159). Para ello, se desarrolla en este estudio, la problemática para la investigación, la selección de una teoría basada en casos de éxito, examinar la situación, y finalmente ensayar y crear una propuesta para solución del problema.

3.2 Alcance Investigativo

Para el presente estudio, se utiliza el tipo de investigación práctica aplicada de tipo “estudios de casos”, la cual es uno de los tipos mencionados por el SEP (Sistema de Estudios de Posgrado de la Universidad de Costa Rica) relacionados con la Orientación, en la cual, según los lineamientos específicos para los trabajos finales de investigación aplicada de las maestrías profesionales, se indica lo siguiente:

> Los estudios de casos son un método de investigación que se emplea como práctica regular para estudiar rigurosamente, y paso a paso, los diversos factores que producen desarrollo, cambio o afectan una situación dada de un problema social determinado. El objeto de estudio puede ser un negocio, una familia, escuela, pandilla, grupo, u organización social, entre otras (Universidad de Costa Rica, Sistema de Estudios de Posgrado [SEP], 2007).

Además, durante la práctica de este estudio, se debe de analizar la resolución del problema o la mejora de una situación existente específica o particular según la Sra. Vargas, para comprobar un modelo o método por medio de la innovación y creatividad de una propuesta de intervención (Vargas Cordero, 2009, p. 162)

3.3 Enfoque

Para esta investigación, se utiliza el método cualitativo, que según Hernández Sampieri et al. (2006, p. 8) “se fundamentan más en un proceso más inductivo (explorar y describir, y luego generar perspectivas teóricas). Van de lo particular a lo general”.

En otras perspectivas, de acuerdo con Corbetta (2003) “el enfoque cualitativo evalúa el desarrollo natural de los sucesos, es decir, no hay manipulación ni estimulación con respecto a la realidad”. Por otra parte, según la descripción de Grinnell (1997), este enfoque de investigación “a veces referido como investigación naturalista, fenomenológica, interpretativa o etnográfica, es una especie de “paraguas” en el cual se incluye una variedad de concepciones, visiones, técnicas y estudios no cuantitativos”

De acuerdo con Hernández et al. (2006), una de las características de la investigación cualitativa es:

> El investigador plantea un problema, pero no sigue un proceso claramente definido. Sus planteamientos no son tan específicos como en el enfoque cuantitativo. Por otra parte, este tipo de investigación se utiliza primordialmente para descubrir y refinar preguntas de investigación (p. 8)

Con este enfoque, permite que esta investigación pueda analizar y evaluar diferentes escenarios o casos de éxito que han tenido algunas empresas al implementar una arquitectura que acople y desacople nuevas funcionalidades externas al sistema central.

3.4 Diseño

La presente investigación, se basa en un diseño cualitativo documental en donde se enfoca en un nivel de estudio exploratorio que busca estudiar casos de éxitos, conocer las propiedades y características de una situación e indagar las causas y las condiciones que generan la problemática.

Para ello, algunos de los métodos y técnicas ejecutadas en un diseño cualitativo documental son: entrevistas, análisis, observación empírica, estudios de casos, investigación bibliográfica, estudio etnográfico y/o estudio fenomenológico (Tesis y Másters, s.f.), por lo que es importante saber que para efectos de esta investigación, y por la naturaleza de ella, no se enfoca ni se trabaja con datos medibles o cuantificables.

3.5 Población y muestreo

A pesar de que esta investigación es de tipo cualitativa, el cual utiliza la técnica de muestreo no probabilístico (Hernández Sampieri et al., 2006), se delimitan de forma específica todos aquellos profesionales que puedan dar un aporte de información coherente para complementar dicho estudio. Por lo cual, para el presente estudio se le realizará entrevistas a las partes interesadas en la propuesta para Builds 2.0. Sin embargo, según Hernández et al (2006) es importante recalcar que en este tipo de estudios, la muestra planteada al inicio de la investigación puede ser diferente a la muestra final. Se pueden agregar casos que no se había complementado o excluir aquellos que sí estaban señalados en este apartado (Hernández Sampieri et al., 2006, p. 564). A continuación, se demuestra en una tabla la población seleccionada para esta investigación:

Profesional	**Nombre**
Jefe de equipo	Erick Arroyo
Líder de equipo	Jing Du
Líder técnico del proyecto Builds 2.0	Kendall González

Desarrollador Senior	Jeison Meléndez
Desarrollador	Juan Alvarado

Tabla 2 Población (muestreo no probabilístico)
Elaboración propia

CAPÍTULO IV

ANÁLISIS DE SITUACIÓN

En este capítulo, se detalla el análisis de la información de los instrumentos de recolección de datos, así como las entrevistas, búsquedas de información y la observación. Estos instrumentos fueron aplicados a las partes interesadas en la propuesta para Builds 2.0.

Con base a la información recolectada, se presenta información relevante que muestra el estado de la situación actual. Por lo tanto, se toman en consideración factores como el conocimiento y la experiencia de los expertos, desde el punto de vista de la profesión y la filosofía que tienen.

Además, el objetivo de este capítulo es profundizar en el estudio y análisis del negocio y arquitectura actual identificando algunas deficiencias que pueden surgir durante el proceso de desarrollo de la aplicación, y cómo estas pueden afectar de forma negativa su rendimiento, mantenibilidad, escalabilidad y la experiencia de usuario.

Tal y como se menciona en el capítulo 2 de esta investigación, la arquitectura de software es una disciplina crítica en el desarrollo de aplicaciones de software, ya que proporciona un marco sólido para la organización y estructuración del código. Por lo que en este análisis de situación, se pretende identificar y examinar los principales problemas que se encuentren en la arquitectura de Software-Software para poder diseñar una solución sólida y coherente a lo que se necesita.

4.1 Builds 2.0 y su objetivo

Según Erick Arroyo, Gerente encargado de la iniciativa Builds 2.0, la estrategia de Wind River es crear una solución única que permita la implementación de cada uno de los procesos de software para sistemas embebidos o inteligentes. Este proceso se conoce como la etapa de construcción (*build* en inglés) tanto en sistemas operativos como de las aplicaciones que se ejecutan en el sistema.

Esto permite ampliar el mercado que tiene la empresa así como operaciones militares, industrias médicas o también en la automotriz. Por ende, el señor Erick enfatiza

que la propuesta permitirá que el producto de Wind River sea flexible y extensible, en la que cubre múltiples casos de usos reales que tienen sus clientes.

Uno de los desafíos encontrados, según el señor Erick, es poder abarcar un mayor porcentaje del mercado de sistemas inteligentes, internet de las cosas y de sistemas embebidos, ya que hay una gran parte de clientes que no utilizan estos sistemas operativos y tienen ya versiones personalizadas que ellos mismos desarrollaron basado en Linux, o incluso sistemas operativos más enfocados a sistemas de entretenimiento (*infotainment* en inglés) que viven en vehículos que no tienen el caso de uso de Wind River.

Por otra parte, el señor Erick deja claro que la clave de la solución apunta a 3 criterios de arquitecturas muy importantes las cuales son modularidad, extensibilidad y flexibilidad.

Ajustar una solución a las necesidades específicas del cliente requiere de la flexibilidad, es decir no se entrega por fuerza una solución para que lo use el cliente, sino la empresa como tal se debe de adaptar al cliente. Esto impacta evidentemente en la proactividad del cliente y la eficiencia. Además, implementando diferentes casos de usos, soportando funciones básicas y funciones extras demuestran la extensibilidad de la propuesta.

El alcance de la propuesta ya desarrollada lo irá dictando mucho los requerimientos de usuario que se presenten en el resto del año de 2023. Sin embargo, se tiene planteado entregar un diseño de la propuesta al principio del próximo año 2024 e iniciar con su desarrollo después de la presentación del diseño.

Uno de los criterios de éxito mencionado por el señor Erick, es que se logre desarrollar una solución efectiva para los problemas y casos de usos reales de los clientes de Wind River. Ya que es una solución que va a ser utilizada por uno de los principales clientes de la empresa. En donde puedan adaptarse e integrar sistemas operativos de Wind River así como VxWorks y Wind River Linux en sus productos.

Su impacto se medirá a nivel de la aceptación del cliente con respecto a la calidad, defectos, rendimiento y seguridad de los datos. Esto aumenta la eficiencia, operatividad y adaptación de Wind River Studio.

Algunos de los requerimientos no funcionales mencionados por el señor Erick, son la definición de la ejecución de pruebas de rendimiento en la que evalúa tanto la escalabilidad, como la resiliencia que tengan los microservicios que se va a desarrollar y también los criterios de seguridad que exige la compañía, así como la implementación de control de accesos de los usuarios, identificación de posibles vulnerabilidades y la mitigación de las mismas, asegurar que los microservicios no tengan grandes cantidades de CVE (en inglés conocido como *Common Vulnerabilities and Exposures,* un sistema de catalogación pública que identifica vulnerabilidades de seguridad conocidas en el producto de software) críticos, seguir estándares de seguridad como OWASP, o CIS que pertenecen al gobierno de Estados Unidos, entre otras. Por lo que a nivel de requerimientos no funcionales está muy enfocado a nivel de "*performance*" y de seguridad.

4.1.1 Enfoque técnico

El líder técnico de equipo, Jing Du, recomienda que el diseño de la arquitectura de la propuesta sea de microservicios, ya que esto permite ventajas clave como el modularidad, escalabilidad y también la flexibilidad, tal y como lo mencionó el señor Erick, gerente encargado de la propuesta.

Con estos 3 criterios, según el señor Jing, los servicios pueden desarrollarse en diferentes lenguajes de programación y tecnologías, lo que permite a los equipos de desarrollo utilizar las herramientas que se adapten a sus necesidades. Adicionalmente, si un servicio necesita escalar se puede realizar de manera aislada sin afectar a los otros componentes.

El señor Jing y Jesion, desarrollador senior de Builds 2.0, recomiendan utilizar tecnologías como Kubernetes para la orquestación y gestión de contenedores, Docker para contenerizar las aplicaciones y sus dependencias, Kong como la capa que

proporciona seguridad, administración y monitorización de las APIs y gRPC para la comunicación entre los microservicios del proyecto. Además, para el desarrollo, utilizar Golang como lenguaje de programación principal ya que este lenguaje es conocido por su eficiencia y rendimiento y PostgreSQL como gestión de datos que ofrece robustez y escalabilidad.

Por otra parte, no se requieren requisitos de hardware ya que para ofrecer flexibilidad y escalabilidad la propuesta debe de ser diseñado tomando en cuenta el modelo de funcionamiento "*on demand*" (en español conocido como bajo demanda), lo que significa que se adapta a sus necesidades específicas en tiempo real sin requerir una infraestructura de hardware. En otras palabras, el 100% de la implementación de la propuesta será en la nube.

Uno de los desafíos de escalabilidad y rendimiento en el diseño es la implementación de un "*load balancer*" (balance de carga en español), que es una técnica que distribuye el tráfico de entrada entre múltiples servidores o recursos de manera equitativa, explica el señor Jing. Además del *load balancer*, el señor Jeison, enfatiza que es esencial la implementación del CI/CD para automatizar el proceso, realizar pruebas rigurosas y mantener un monitoreo continuo de los servicios utilizando un "*health check*" para cada uno de los microservicios con la finalidad de observar el comportamiento de los servicios. Y para la recolección de errores se puede utilizar Fluent bit que es una tecnología ya implementada por Wind River.

Adicionalmente, otro de los desafíos es con la seguridad del aplicativo. Para ello, se debe de diseñar una estrategia en donde combine las medidas respectivas para proteger los datos y garantizar la integración segura de la información. Estos son RBAC, conocido en inglés como Role-Based Access Control y el uso de la Red Privada Virtual o VPN.

4.1.2 Enfoque al proyecto

Según el señor Kendall González, líder técnico del equipo encargado de la propuesta, Builds 2.0 busca cambiar la filosofía actual, por una en la que pueda gestionar

las extensiones de una manera más ágil, tal y como es la filosofía de *plug-and-play.* Además, éste funcionará como un orquestador o un facilitador que ayude a agilizar el proceso de gestionar una nueva funcionalidad que necesita el usuario.

Durante la entrevista con los expertos, explican de forma detallada el funcionamiento principal de Builds 2.0 con el siguiente diagrama de flujo creada por el señor Kendall:

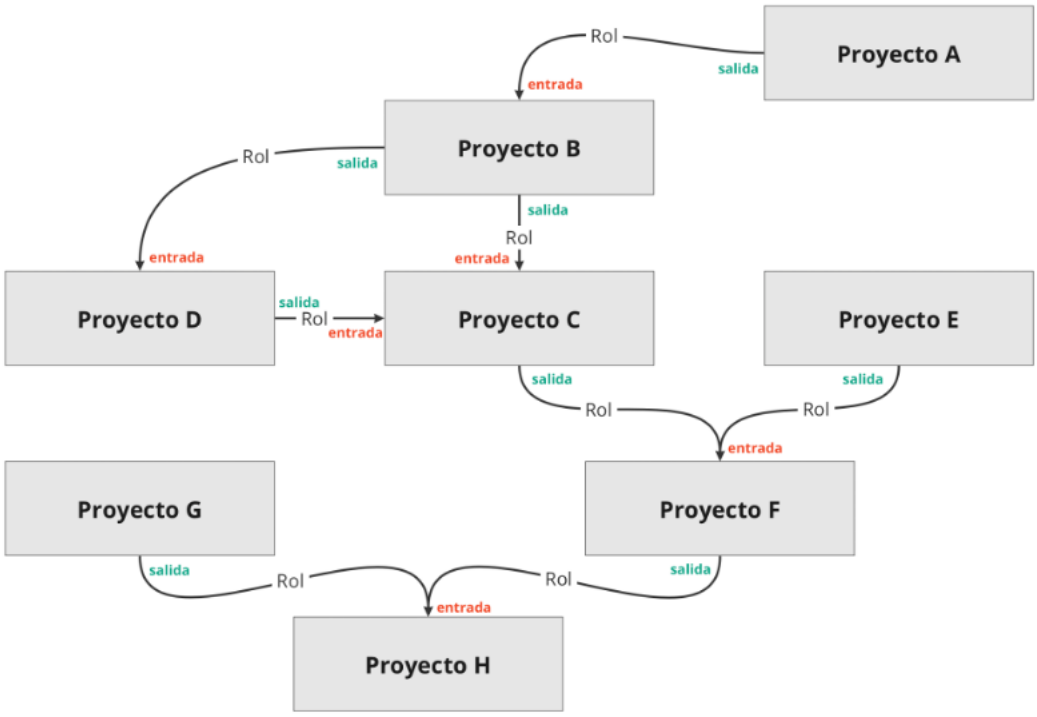

Ilustración 7 Diagrama de flujo de datos Builds 2.0
Fuente: Kendall González

En el diagrama anterior (ilustración 7), se muestran diferentes tipos de proyectos que se relacionan entre sí mediante un rol en específico que representa el tipo de salida que obtiene en una ejecución de un proyecto, y además también simboliza el tipo de entrada que requiere un proyecto para realizar su correspondiente ejecución. Significa que existen proyectos totalmente independientes de otros, pero también hay proyectos que dependen de los artefactos producidos por otros.

Estos proyectos se pueden representar como extensiones que contienen funcionalidades únicas y exclusivas del mismo que también pueden utilizar artefactos de otros proyectos para producir uno completamente nuevo.

Por lo que, según el señor Kendall, el objetivo de Builds 2.0 es construir proyectos que se puedan relacionar de forma dinámica sus dependencias y configuraciones.

4.2 Arquitectura actual de Builds 2.0

Según los señores Kendall y Juan, líder de proyecto y desarrollador de Builds 2.0, existen microservicios internos que se encargan de transcribir e interpretar un archivo de configuración que provee el usuario con indicaciones técnicas para la ejecución de un proyecto en específico. Además de otros microservicios que funcionan como la sincronización del estado de la ejecución a través de un componente tercero de la empresa.

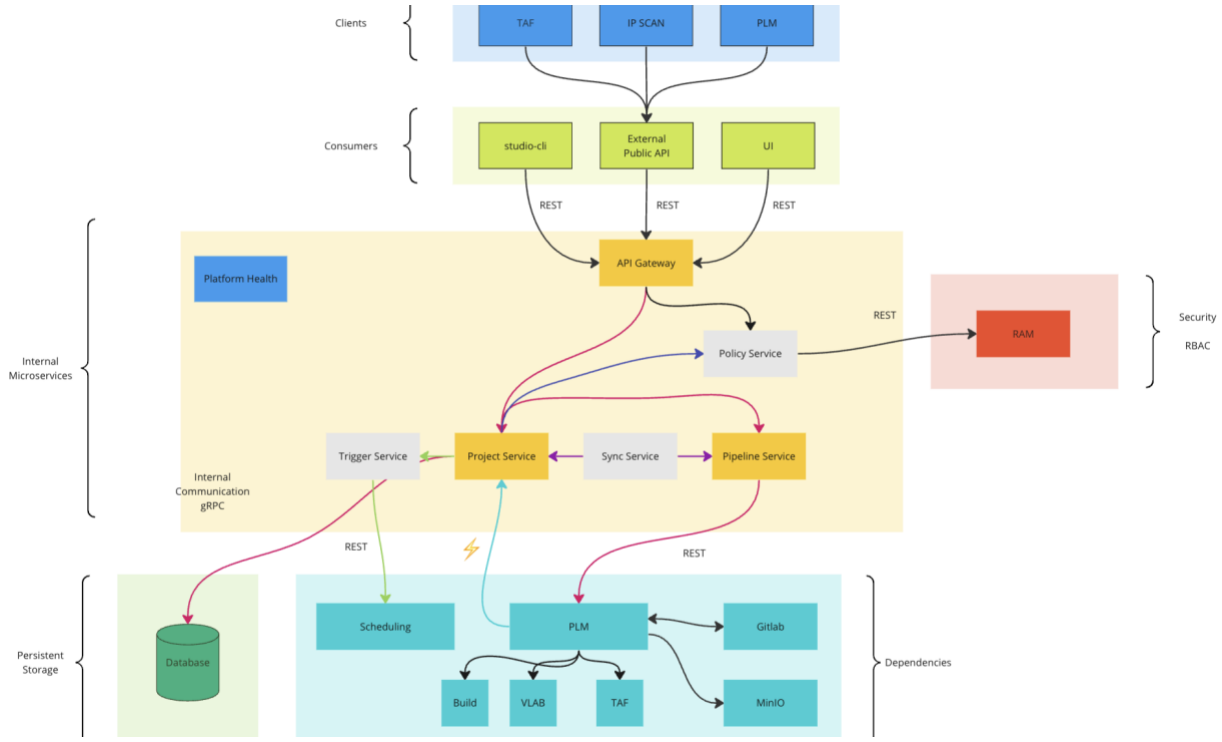

Ilustración 8 Arquitectura actual de Builds 2.0
Fuente: Kendall González

En la ilustración anterior (8), obtenida por el señor Kendall, se puede apreciar la fuerte dependencia que tiene cada uno de los microservicios, además de otras dependencias con servicios externos al Builds 2.0. A continuación, se explica cada uno de los bloques de forma detallada junto con sus dependencias según los señores Kendall y Juan:

- Consumidores (studio-cli, public APIS e Interfaz de Usuario): representa la forma en que el usuario se comunicará con Builds 2.0 para la ejecución de su proyecto. Puede ser por medio de líneas de comandos, interfaz de usuario o por medio de interfaz de programación de aplicaciones (conocido como "*API*").
- Puerta de enlace (API-Gateway): actúa como un único punto de entrada para los microservicios internos de Builds 2.0 en la que gestiona todas las tareas y procesamiento de llamadas simultáneas de las *APIs.*
- Servicios (Project, Sync, Pipeline, Trigger y Policy): son los servicios encargados de obtener las configuraciones de los usuarios, transcribirlas e interpretarlas para ser procesadas como tareas en el componente orquestador de la empresa. De esta manera ejecutarlas y obtener los artefactos correspondientes para cada uno de los proyectos.
- Servicios externos (RAM): son los servicios o librerías desarrolladas y compartidas a nivel interno de la empresa para ser consumidos por otros componentes como Builds 2.0.
- Componente orquestador (PLM, conocido como Pipeline Manager): representa un facilitador que espera recibir como entrada una tarea que pueda ser ejecutada para obtener artefactos como recursos de los proyectos.
- Componentes (Build, VLAB, TAF): estos son los responsables de compilar las tareas que le corresponden provenientes del componente orquestador.
- Base de datos: almacena las configuraciones técnicas dadas por el usuario para luego ser interpretadas, ejecutadas y compiladas por el componente orquestador.

4.3 Ventajas

Builds 2.0 viene a ser un cambio de filosofía a lo que se está implementado actualmente en la empresa para agilizar y mejorar el proceso de compilación de proyectos, según el señor Kendall. Y gracias a esta filosofía se puede visualizar algunas ventajas significativas que serán puntualizadas a continuación:

- ➢ Builds 2.0 proporciona una interfaz totalmente centralizada que gestiona y supervisa la ejecución de tareas múltiples. Esto mejora la experiencia de usuario y facilita el control y la administración del sistema.
- ➢ Builds 2.0 llega a automatizar procesos manuales que se realizaban en distintos componentes totalmente separados. Esto organiza todas las tareas por realizar en una sola aplicación.
- ➢ Builds 2.0 puede escalar los servicios horizontalmente según su demanda. Es decir, crea nuevas instancias de servicios para manejar cargas de trabajo adicionales y viceversa. Esto optimiza los recursos y mejora la eficiencia.
- ➢ Builds 2.0 monitorea y registra todas las ejecuciones y/o transacciones que realiza el usuario, para obtener información relevante sobre el rendimiento y comportamiento del sistema.
- ➢ Builds 2.0 coordina y planifica la ejecución o compilación de los proyectos; gracias a esto maximiza el aprovechamiento de los recursos evitando la sobrecarga y la inactividad de algunos componentes.
- ➢ Builds 2.0 al ser el único punto de entrada, gestiona y centraliza los accesos y autenticación, asegurándose que solo los usuarios y servicios autorizados puedan acceder a los recursos.

4.4 Problemáticas

Como la mayoría de las aplicaciones de software, nunca son diseñadas de una manera perfecta, ya que siempre existen aspectos que no se consideran al inicio de la planificación. Por lo que en este apartado, se identificarán algunas problemáticas que se

pueden observar en la arquitectura actual con base a las explicaciones dadas por los señores Kendall y Juan durante la recolección de datos.

4.4.1 Lógica de Negocios

Una lógica de negocios mal desarrollada puede impactar significativamente en las funcionalidades, seguridad y la eficiencia del software, por lo que los usuarios pueden experimentar una experiencia negativa. Por esta razón, es de suma importancia invertir parte del tiempo y esfuerzo en un diseño adecuado para los requerimientos del usuario.

Algunos de las problemáticas que se identificaron en la arquitectura actual son las siguientes:

- ➢ La dificultad en el mantenimiento y la evolución del software ya que la lógica de negocios es un poco confusa. Es decir, los mismos desarrolladores pueden tener problemas para entender y modificar el código existente. Esto puede aumentar los costos y el tiempo para realizar las mejoras pertinentes.
- ➢ Actualmente, algunas características pueden no funcionar de forma correcta o a como se espera, por lo que se puede identificar funcionalidades incoherentes o incompletas en cada uno de los microservicios.
- ➢ La vulnerabilidad de seguridad es bastante alta en la arquitectura actual, esto porque existen posibilidades de inyección de código o acceso no autorizado a recursos sensibles.

Por lo tanto, una lógica de negocios mal diseñada e implementada afecta múltiples aspectos de la aplicación desde su funcionalidad y el rendimiento hasta la escalabilidad y la integración con otros componentes. Es crucial dedicar esfuerzo en un buen diseño de lógica de negocios para evitar las problemáticas mencionadas anteriormente y lograr un software eficiente y confiable.

4.4.2 Dependencias

En el momento que los microservicios tienen dependencias unos a otros, surgen diferentes problemáticas que impactan su escalabilidad, fiabilidad y mantenibilidad del sistema. En Builds 2.0, se identifica este tipo de comportamiento, ya que su flujo de datos depende de otros servicios para ser ejecutados.

A continuación, se mencionan algunas de las principales problemáticas que se identificaron durante el proceso de recolección de datos con los señores Kendall y Juan:

- En Builds 2.0, los cambios de otros microservicios pueden llegar a afectar a otros servicios que dependen de él. Esto dificulta la evolución independiente de los microservicios.
- Si un microservicio llega a fallar, toda la aplicación dejará de funcionar. Por lo que se identifica riesgos de fallos en cascada en la que desencadena errores si uno de ellos colapsa.
- Actualmente, Builds 2.0 no tiene la capacidad de segmentar ni aislar ciertas funcionalidades para poder escalar, actualizar o reemplazar partes específicas del sistema sin afectar a otros por su alta dependencia entre microservicios.
- Existen problemas de consistencia y sincronización de datos ya que Builds 2.0 depende de cierta información de otros componentes internos de la empresa. Esto puede causar complejidad al sincronizar datos entre los sistemas.
- Debido a las dependencias, las pruebas automatizadas son mucho más complejas. Esto puede afectar los resultados de las mismas de otros servicios y también puede dificultar la detección de errores.
- Se observa latencia en la comunicación. Builds 2.0 necesita la respuesta de un componente externo para ejecutar ciertas funcionalidades, por lo que puede afectar el tiempo de respuesta en general del sistema y experiencias del usuario.
- Estas dependencias le agregan complejidad a los aspectos de seguridad, ya que cada uno de los servicios deben de establecer mecanismos de autenticación y

autorización para garantizar que los recursos que se están enviando estén protegidos y que valide si el usuario tiene acceso a ella.

4.4.3 Pruebas e integración

Según el señor Kendall, actualmente Builds 2.0 no cuenta con pruebas unitarias implementadas, por lo que pueden surgir problemáticas que afectan a la calidad, seguridad y estabilidad de la aplicación durante su desarrollo y mantenimiento.

La calidad y la confiabilidad del software son conceptos importantes que pueden determinar el éxito o fracaso de un proyecto, por lo que las pruebas juegan un papel fundamental que garantiza la robustez y la estabilidad del software como tal.

Algunas problemáticas identificadas en Builds 2.0 con respecto a las pruebas, son las siguientes:

- Al no tener pruebas unitarias, ni automatizadas, aumenta la probabilidad de que existan errores y fallos en el código base. Esto impacta de forma negativa el software una vez desplegado en producción.
- Dificulta la depuración e identificación temprana de errores antes de su despliegue a producción. Esto puede ser perjudicial y costoso para la reputación de la empresa.
- La falta de pruebas unitarias y automatizadas puede requerir procesos manuales que retrasa la entrega de nuevas funcionalidades, por lo que ralentiza el tiempo de comercialización y capacidad de respuesta a las necesidades del mercado.
- Maximiza el tiempo y recursos invertidos en pruebas manuales, ya que sin estas pruebas unitarias o automatizadas, el equipo desarrollador debe de dedicar más tiempo para realizar todo el proceso manual.
- Dificulta la trazabilidad e identificación de la causa original del problema y el desarrollador solo llegará a soluciones temporales para abordar el problema.
- Provoca cambios no previstos en el comportamiento del software y causa una inestabilidad generalizada en el código base del proyecto.

- Durante una integración de sistemas, se vuelve aún más complejo ya que no hay una garantía automatizada de que los cambios no afectan a otras partes del sistema.
- Dificulta la refactorización y optimización del código base. Esto limita la mejora continua del software.
- Deja pendiente muchas deudas técnicas sin identificar, por lo que aumenta los riesgos a fallos o errores no previstos.
- La falta de estas pruebas minimiza la confianza durante el proceso de desarrollo y no solo en los desarrolladores, sino también afecta a los usuarios directos, ya que pueden tener menos confianza al utilizar la aplicación.
- Minimiza la escalabilidad del equipo de desarrollo, ya que puede generar conflictos en el código y también errores difíciles de detectar.
- Sin pruebas unitarias ni automatizadas, es mucho más probable que algunas funcionalidades se pierdan con el tiempo debido a cambios en el código que no se validan, lo que impacta la experiencia del usuario.

Por lo tanto, la ausencia de estas pruebas en Builds 2.0, tiene un impacto importante en la calidad, estabilidad, confiabilidad y escalabilidad del software. Estas problemáticas destacan la importancia de incorporar prácticas sólidas de pruebas en el proceso de desarrollo, lo que permite mejorar la eficiencia y reducir los costos de mantenimiento, logrando un software de alta calidad. Es fundamental que los equipos de desarrollo tomen en consideración la importancia de estos puntos como parte integral del ciclo de software.

CAPÍTULO V

PROPUESTA DE SOLUCIÓN

En el presente capitulo se detalla la propuesta a presentar a la empresa Wind River Systems, para el proyecto Builds 2.0, tomando como base toda la información recolectada de las entrevistas y los diseños preliminares compartidos por el equipo de trabajo del mencionado proyecto.

Teniendo como objetivo principal mostrar una hoja de ruta para el desarrollo del proyecto en aspectos constructivos que ataquen las diversas problemáticas que poseen el proyecto Builds original y las versiones preliminares mencionadas en el análisis de resultados del proyecto Builds 2.0.

5.1 Enfoque y motivación

La propuesta se basa en las diversas arquitecturas de referencia presentadas en el capítulo 2 del presente documento, pues son arquitecturas ya establecidas en la industria y ampliamente conocidas, además de estar respaldadas por nombres de suma relevancia en la industria del software como lo pueden ser el Dr. Alistair Cockburn o el ingeniero Robert C. Martin.

Como se puede comprender del capítulo anterior, el proyecto Builds 2.0 tiene bien definidos los objetivos que el sistema debe cumplir, teniendo ya bocetos de arquitecturas que definen las comunicaciones de diversos servicios siguiendo la arquitectura de microservicios, lo cual ayuda a que la propuesta tenga una referencia de lo que se desea construir, sin embargo no se describe la arquitectura de software que se debe seguir para construir cada uno de los microservicios y aunque se mencionan algunas tecnologías a utilizar, estas no definen la forma en la que se deba implementar cada pequeño sistema dentro del proyecto completo.

Por lo que la presente propuesta se enfoca en definir una base inicial de arquitectura para la construcción de un microservicio, para tomar ese elemento como base constructiva de un sistema más grande y que además cumpla filosofías descritas como objetivos de la investigación como puede ser los conceptos de plug-and-play que asegura la escalabilidad de Builds 2.0, facilitando su conexión a sistemas externos y dentro del mismo sistema.

Además, la propuesta abarcará la arquitectura desde diversos puntos de vista como puede ser seguridad o pruebas de software para cubrir las diversas deficiencias que los entrevistados mencionan que posee el sistema actual, Builds; con el fin de dar mucho más valor y apoyar a que la construcción de Builds 2.0 pueda ser un éxito desde su diseño.

5.2 Estructura base de un microservicio

La investigación tiene como objetivo principal proponer una estrategia que permita expandir de manera sencilla Builds 2.0, siguiendo la filosofía de plug-and-play. Se busca facilitar su extensión dado que este sistema constituye el núcleo de una estructura más amplia con conexiones externas a otros sistemas y tecnologías. Por lo que inicialmente, se parte de la arquitectura de microservicios actualmente empleada en el proyecto, ya que presenta ventajas que se ajustan a las necesidades de la empresa Wind River Systems, siendo esta arquitectura la base estructural de todo el sistema.

Pero además de solo establecer la estructura de microservicios, la propuesta inicial consiste en establecer directrices para la construcción de cada microservicio, combinando dos arquitecturas de software: la arquitectura hexagonal del Dr. Alistair Cockburn y la filosofía de arquitecturas limpias de Robert C. Martin. El enfoque se centra en desarrollar unidades (microservicios) con características de calidad autónomas, que puedan integrarse y probarse de manera eficiente como entidades independientes. Buscando que estos beneficios se extiendan a todo el sistema al conectar estos microservicios para interactuar como una aplicación cohesionada.

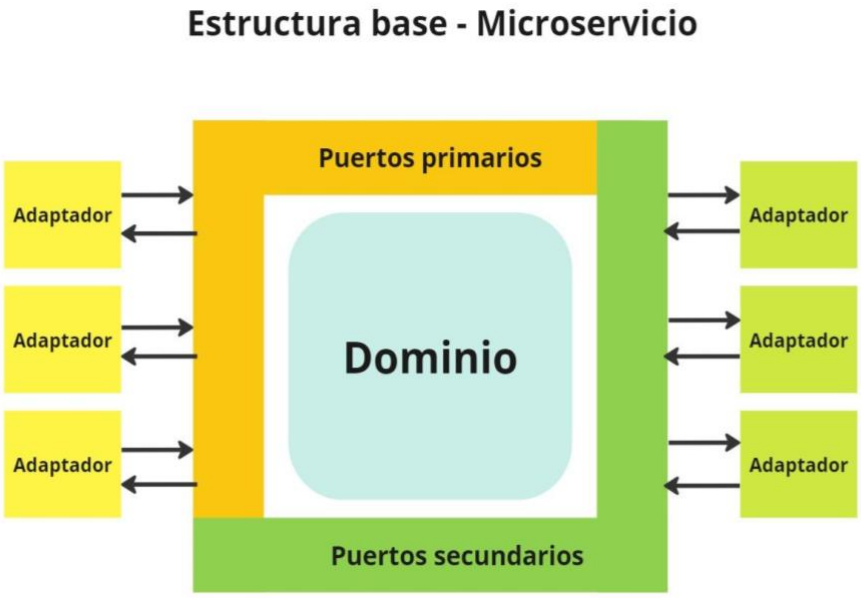

Ilustración 9 Estructura base de un microservicio - Arquitectura Hexagonal
Fuente: Elaboración propia

En la Ilustración 9 se observa una representación visual, con base en la arquitectura hexagonal y proporciona una guía para la construcción de microservicios como unidades y estructuras fundamentales para la propuesta actual. Esta representación visual presenta un centro que abarca todo lo relacionado con la lógica de negocio o en otras palabras, el dominio de la aplicación, específicamente el dominio del microservicio en cada caso.

En esta imagen, se plantea la existencia de puertos y adaptadores para facilitar la comunicación entre el dominio y el entorno externo al microservicio. La noción de puertos y adaptadores es equivalente a la arquitectura hexagonal y contribuye significativamente al cumplimiento del objetivo de la investigación, que busca establecer un sistema plug-and-play desde la arquitectura base hasta la construcción del sistema completo.

La concepción de puertos y adaptadores, mediante la aplicación de conceptos del paradigma de programación orientada a objetos, implica la definición de interfaces (puertos) que especifican con claridad las acciones que se deben seguir. Luego, se procede a la implementación de adaptadores específicos, detallando de manera más

precisa el método para llevar a cabo las acciones establecidas por dichas interfaces. Este enfoque se alinea con el principio de inversión de dependencias, facilitando un desacoplamiento más eficiente de las responsabilidades, elevando la cohesión del código y disminuyendo su nivel de acoplamiento.

A pesar de que la arquitectura hexagonal, en su concepción, define puertos primarios y secundarios con enfoques distintos en su aplicación, en este contexto se propone una simplificación de estos conceptos con el objetivo de facilitar la implementación de la arquitectura y reducir el peso cognitivo asociado a la construcción, abordando únicamente los puertos como contratos que los adaptadores deben seguir. En este sentido, el microservicio puede tener puertos primarios, secundarios, terciarios o la cantidad necesaria, según su enfoque.

Esta simplificación puede visualizarse también al considerar el concepto de un archivo manifiesto en la construcción de plugins en diversos sistemas, no obstante, se invierte el concepto, a diferencia de que el adaptador dicta al puerto cómo actuar, en este caso, se invierte el papel para que el puerto define cómo se debe actuar hacia el exterior de la aplicación. Este enfoque busca proporcionar una mayor claridad y simplicidad en la implementación de la arquitectura hexagonal dentro de la propuesta.

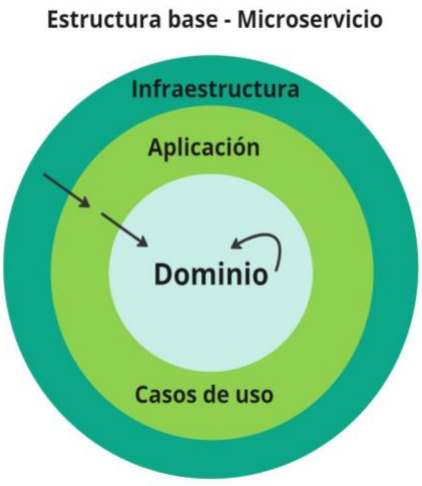

Ilustración 10 Estructura base de un microservicio - Arquitectura limpia
Fuente: Elaboración propia

La estructura fundamental del microservicio se puede conceptualizar también a través de las arquitecturas limpias. La Ilustración 10 ofrece una representación visual que consta de tres capas. En primer lugar, se encuentra el dominio de la aplicación, seguido por una capa que alberga los puertos, la cual engloba la aplicación en sí misma y/o los casos de uso asociados a dicha aplicación y por último, se presenta la capa de infraestructura, donde residen los adaptadores con implementaciones concretas de las interfaces definidas en la capa anterior.

Esta representación arquitectónica del microservicio facilita una comprensión más clara de la implementación del principio de inversión de dependencias. En este contexto, ninguna capa puede depender de su capa superior, por lo tanto, la capa de infraestructura puede hacer uso de sí misma y de las capas de aplicación y dominio. La capa de aplicación puede hacer uso de sí misma y de la capa de dominio. Finalmente, la capa de dominio solo puede hacer uso de sí misma. Este enfoque asegura una estructura bien definida y un bajo acoplamiento entre las capas del microservicio.

Capa arquitectura limpia	**Definición de arquitectura hexagonal**
Dominio	Dominio
Aplicación o casos de uso	Puertos primarios y secundarios
Infraestructura	Adaptadores primarios y secundarios

Tabla 3 Relación entre la arquitectura hexagonal y el concepto de arquitecturas limpias
Elaboración propia

La Tabla 3 revela una relación directa 1:1 al combinar los conceptos presentados en las Ilustraciones 7 y 8. Se observa que cada uno de los conceptos explicados puede aplicarse de manera integral en la construcción de un microservicio como entidad autónoma. Esta integración posibilita la unificación de las ventajas inherentes a cada uno de los enfoques, permitiendo así aprovechar lo mejor de ambos mundos.

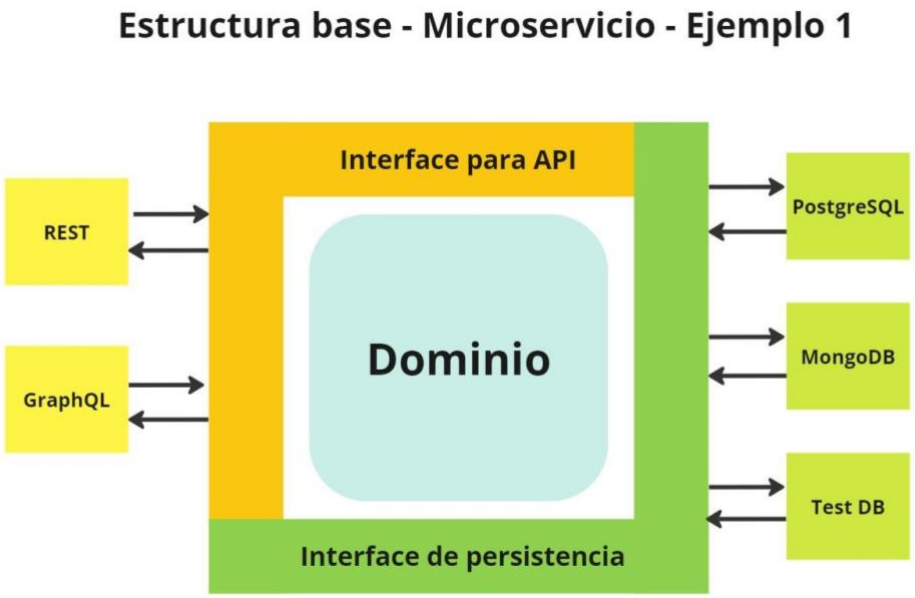

Ilustración 11 Ejemplo de un microservicio - Arquitectura Hexagonal
Fuente: Elaboración propia

La principal ventaja de la arquitectura propuesta es que el uso de puertos (interfaces) dentro de la aplicación permite desacoplar la lógica de negocio de implementaciones concretas de aspectos como una base de datos o una interfaz de comunicación API, lo que convierte a los puertos en contratos bien establecidos que permiten el intercambio de tecnologías o implementaciones en la aplicación.

Como se observa en la ilustración 11, una aplicación con unas interfaces bien definidas puede intercambiar entre diversas tecnologías para exponer unos APIs sin afectar la lógica de la aplicación e igualmente se observa que a nivel de persistencia se puede seguir el mismo principio, permitiendo además programar adaptadores a nivel de pruebas (como se observa del lado de persistencia de datos), para facilitar mucho ese aspecto dentro del microservicio, aislándolo de agentes externos de acuerdo a la necesidad del mismo y el proyecto en tema de pruebas.

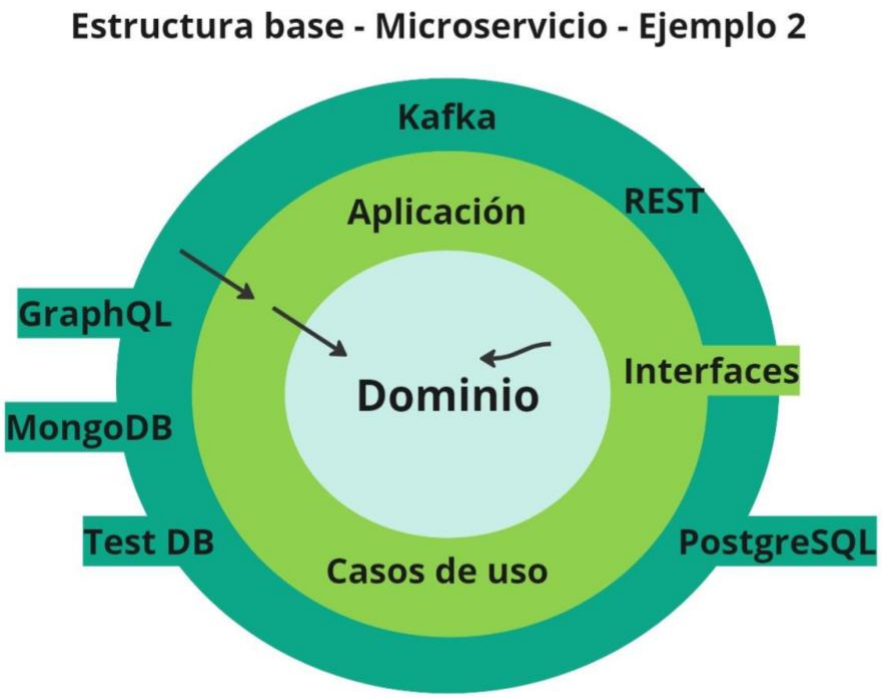

Ilustración 12 Ejemplo de un microservicio - Arquitectura limpia
Fuente: Elaboración propia

De igual manera, en la ilustración 12 se observa como igualmente se puede mapear lo explicado en la ilustración 11, pero en este caso a un nivel de capas, dando una responsabilidad muy concreta a cada componente de la arquitectura.

5.3 Estructura base de microservicios aplicada a la propuesta del proyecto Builds 2.0

Luego de definir la arquitectura que los microservicios deben de seguir en su construcción como unidad, se presenta un ejemplo enfocado en el proyecto en estudio que se basa en la estructura actual analizada en el capítulo anterior.

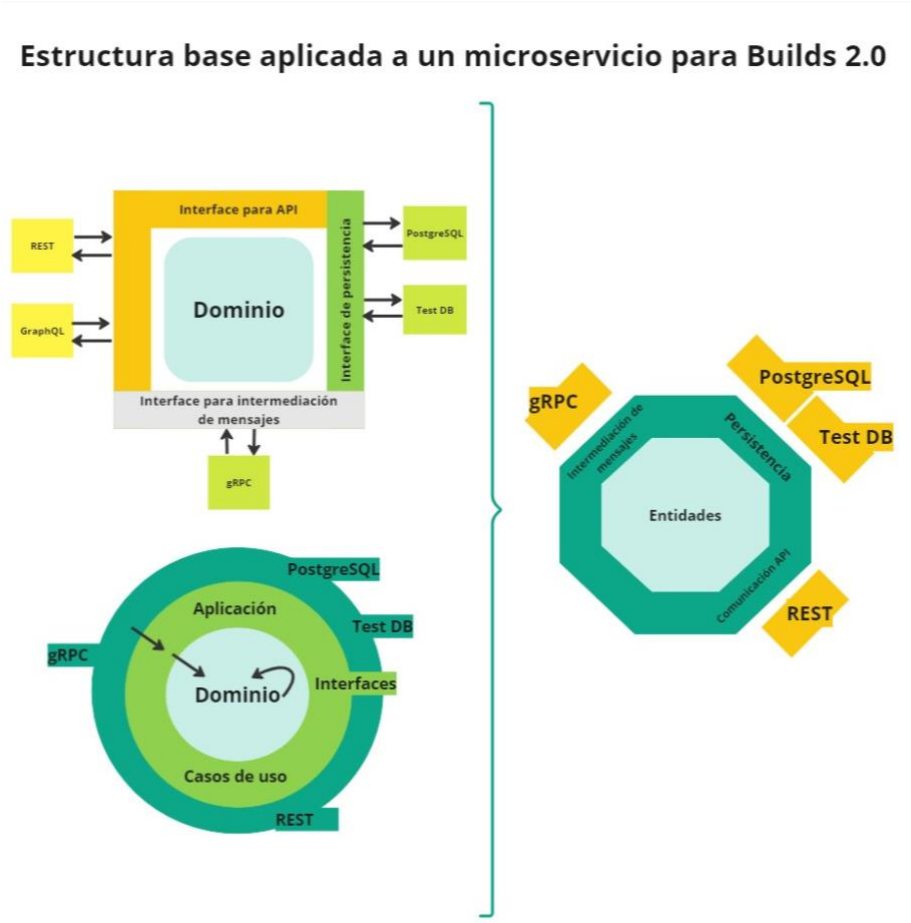

Ilustración 13 Estructura base aplicada a un microservicio para Builds 2.0
Fuente: Elaboración propia

En la ilustración 13, se presenta cómo todos los conceptos explicados en relación con la estructura base de un microservicio puede ser aplicado a un microservicio real para el proyecto Builds 2.0 en donde el dominio se traduce como las entidades y se puede observar como las diferentes capas están muy bien definidas.

El ejemplo anterior presenta un puerto para la comunicación entre los diferentes servicios, en este caso usando como ejemplo la tecnología gRPC, otro puerto para la persistencia de datos con un adaptador para el uso de una base de datos relacional con la tecnología PostgreSQL y luego otro adaptador para pruebas, que puede usarse para aislar la dependencia de la base de datos real para el desarrollo de pruebas automatizadas en diversos niveles, para luego terminar con un puerto para

comunicaciones API, en este caso usando tecnología REST, con el fin de poder hacer peticiones a servicios o sistemas externos de la arquitectura global de microservicios.

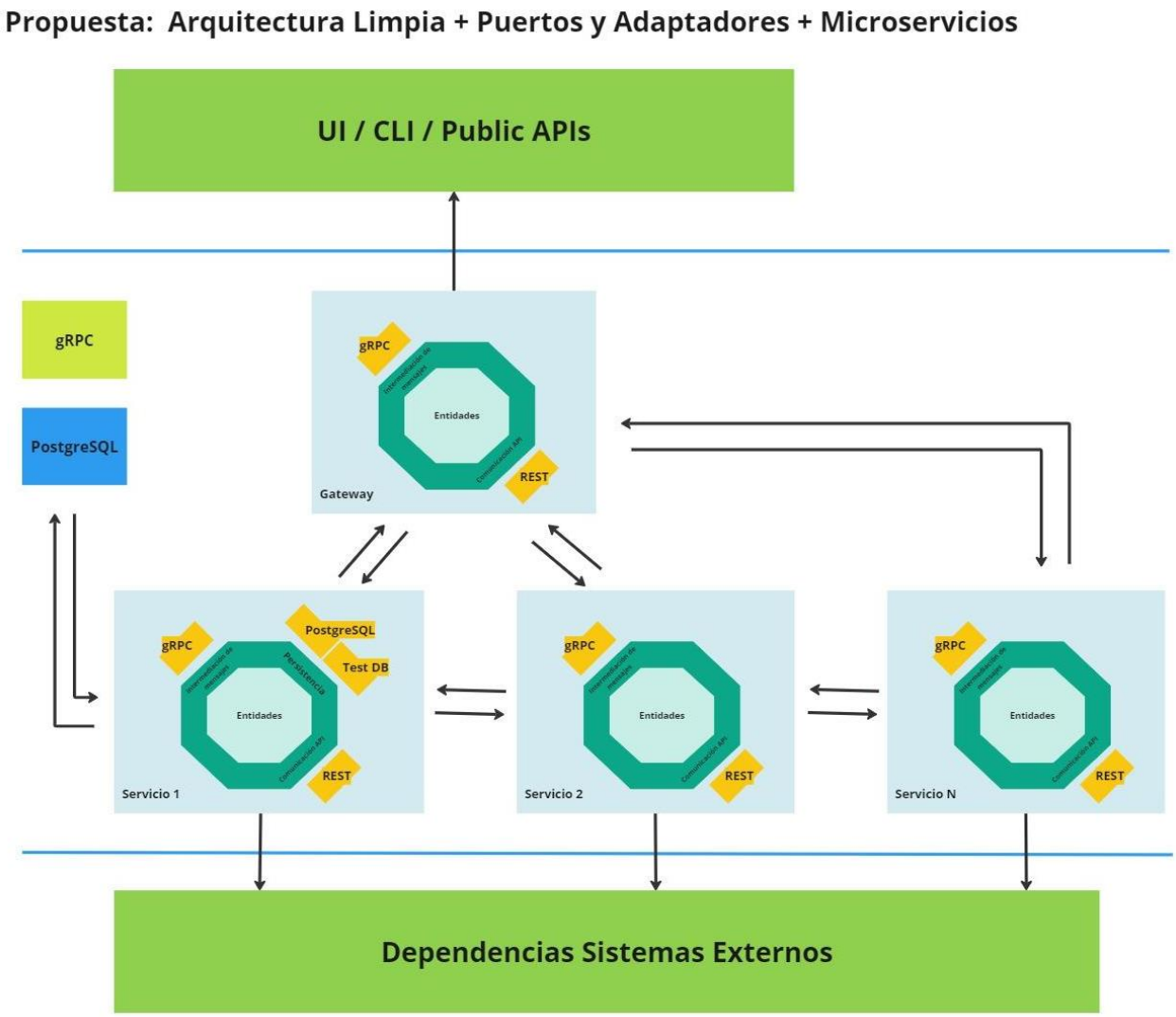

Ilustración 14 Propuesta global de microservicios para Builds 2.0
Fuente: Elaboración propia

En la ilustración 14, se puede observar una propuesta de arquitectura enfocada en Builds 2.0 que combina la arquitectura de microservicios, la arquitectura hexagonal y los conceptos de arquitecturas limpias, además utilizando las tecnologías de ejemplo presentadas en la estructura base del microservicio para el proyecto.

Este ejemplo se construye con base en la arquitectura actual, presentando 3 servicios diferentes (se etiqueta uno como “Servicio N” haciendo alusión de que se pueden construir los microservicios necesarios para el proyecto), en donde solo uno de ellos se encarga de la persistencia de datos, pero entre ellos se pueden comunicar gracias a la tecnología de intermediación de mensajes, contando con una puerta de enlace o *gateway* que funciona como punto de entrada de la aplicación y que en su caso, la implementación del adaptador para APIs tiene la responsabilidad de exponer APIs REST a interfaces de usuario, líneas de comando o simplemente APIs públicos para consumo de otras aplicaciones. En contraposición, la implementación del resto de servicios usa ese puerto para comunicarse con servicios o sistemas externos que usen la tecnología REST para su comunicación.

La principal ventaja de esta arquitectura, como se ha discutido a lo largo del capítulo, es que si por ejemplo una de las dependencias externas de alguno de los microservicios ya no utiliza la tecnología REST y cambia por ejemplo a utilizar GraphQL, solamente se debe intervenir el adaptador, ya que se cuenta con un contrato muy bien definido a nivel del puerto, por lo que la lógica de negocio es agnóstica de cómo se lleva a cabo esa comunicación.

Otra ventaja que tiene esta manera de construir los microservicios es que si, por ejemplo, alguna librería o paquete externo que utiliza la aplicación tiene un cambio de versión, que cambia la forma de hacer alguna acción, igualmente la intervención es más sencilla ya que solo se cambia el adaptador a esa nueva versión, evitando problemas durante la refactorización del código. Lo anterior es importante saberlo ya que la arquitectura no solo da la ventaja de cambiar tecnologías unas por otras, sino que también ayuda a mantener de mejor manera el código ya existente.

En todos los niveles se cumple el objetivo principal de una arquitectura *plug-and-play*, ya que para extender las funcionalidades del microservicio, dado el caso de una nueva funcionalidad para un cliente, solo se debe escribir el puerto (interfaz/contrato) con la lógica definida y luego se puede “conectar” cualquier tecnología que funcione

utilizando ese puerto, permitiendo la “conexión” y “desconexión” de adaptadores con diferentes implementaciones.

5.4 Beneficios de la propuesta de arquitectura desde la perspectiva de pruebas de software

Como se mencionaba en el objetivo del capítulo, la idea es que se piense desde el inicio cómo se pueden integrar y probar las diferentes piezas que engloban el proyecto Builds 2.0, con el fin de ofrecer a la empresa y sus desarrolladores todas las ventajas que provee realizar pruebas de software, ya que tener un buen aseguramiento de la calidad puede hacer que los costos del proyecto bajen o se mantengan en lo establecido, que la construcción del proyecto sea más eficiente, que los desarrolladores tengan seguridad del trabajo realizado, teniendo retroalimentación de manera rápida, y además que se cuente con datos para la toma de decisiones sobre el proyecto de acuerdo a su calidad, a lo largo del tiempo de vida del mismo.

5.4.1 Situación actual orientada a pruebas de software

De las diversas problemáticas que se encontraron durante el análisis de la arquitectura de software actual del proyecto, cabe destacar la dificultad de realizar pruebas automatizadas por la manera en la que está construido el proyecto, pero además cuando se revisan en detalle las problemáticas, con relación a la automatización, se puede llegar a la conclusión de que no se aplican correctamente principios de una arquitectura de pruebas bien definida.

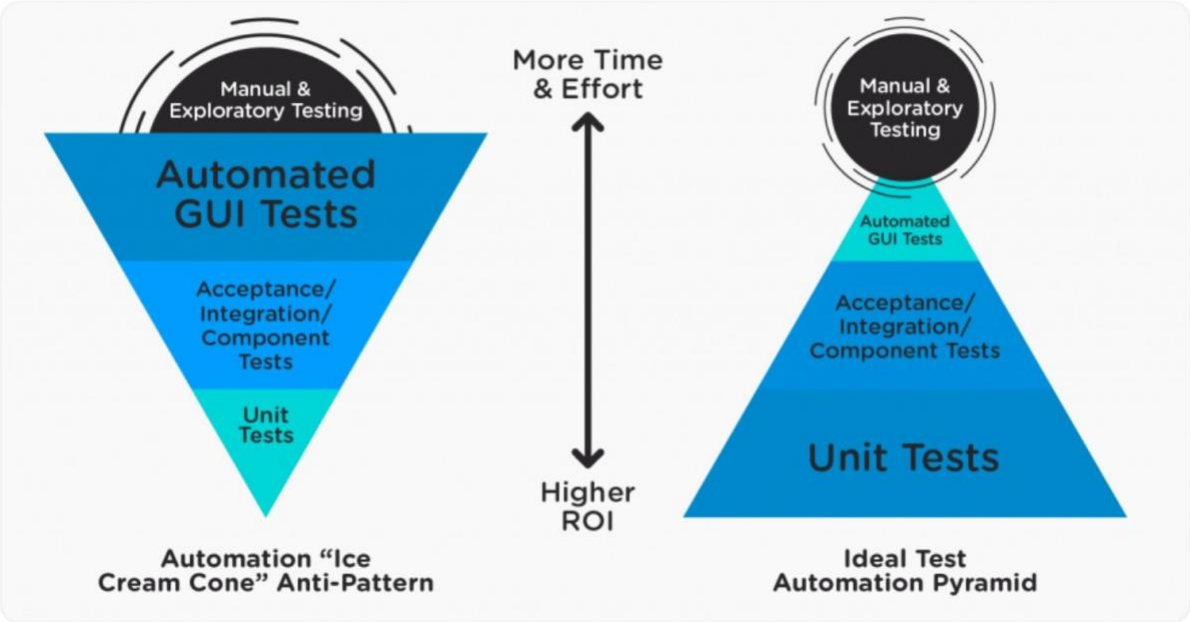

Ilustración 15 Automatización de pruebas en entornos ágiles
Fuente: Sofia Palamarchuk (2015)
Tomado de: https://abstracta.us/blog/test-automation/best-testing-practices-agile-teams-automation-pyramid/

Como se observa en la ilustración 15, el proyecto está enfrascado en el primer escenario, en el anti-patrón del "cono de helado", en donde las pruebas unitarias no tienen la importancia necesaria y se gasta mucho más tiempo automatizando otros niveles de prueba, intentando suplir una deficiencia de diseño en el proyecto que complica la implementación de pruebas.

Como se observa en el segundo escenario, lo ideal es tener un enfoque contrario, en donde las pruebas unitarias son las que tienen más retorno de inversión en su implementación, ya que su tiempo de ejecución es menor y proveen retroalimentación en menor tiempo que, por ejemplo, pruebas de interfaz de usuario que pueden tomar horas para dar un resultado, por lo que queda en evidencia la importancia que tiene que la arquitectura de software del proyecto apoye con igual fuerza una buena implementación de pruebas.

5.4.2 Soporte para pruebas desde la arquitectura propuesta

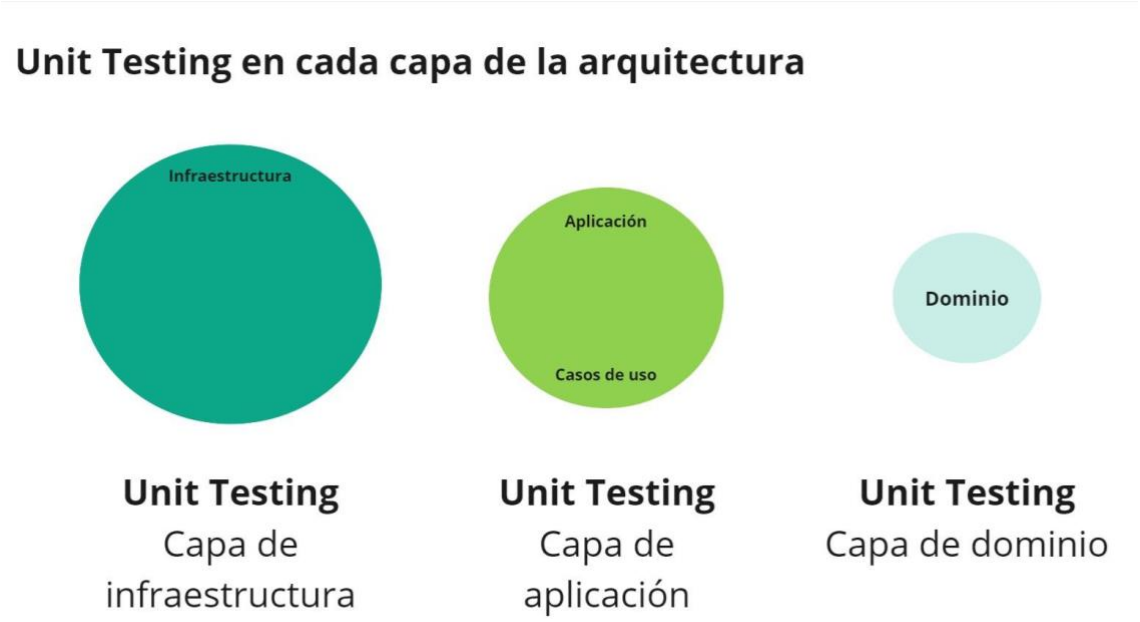

Ilustración 16 Pruebas unitarias en cada capa de la arquitectura de software
Fuente: Elaboración propia

Como se mencionó anteriormente, el escenario ideal para un buen aseguramiento de la calidad es que el producto que se va a construir se pueda probar y esta característica viene dada desde la concepción de la arquitectura que va a seguir el software.

Por lo anterior, en la ilustración 16, se puede observar que tener una buena distribución de responsabilidades, además de contar con capas bien definidas, permite que se puedan probar de manera independiente y de manera más sencilla, pero no solo a un nivel de pruebas unitarias, como la ilustración aduce, si no que en más niveles, como pueden ser los de integración, sistema o aceptación, incluso llegando a poder realizar, de mejor manera, pruebas a nivel no funcional, gracias a que la arquitectura le permite a los ingenieros de calidad de software diseñar su estrategia de pruebas de manera más eficiente y robusta, al no tener que descifrar cómo desestructurar una arquitectura que no se puede probar de manera sencilla.

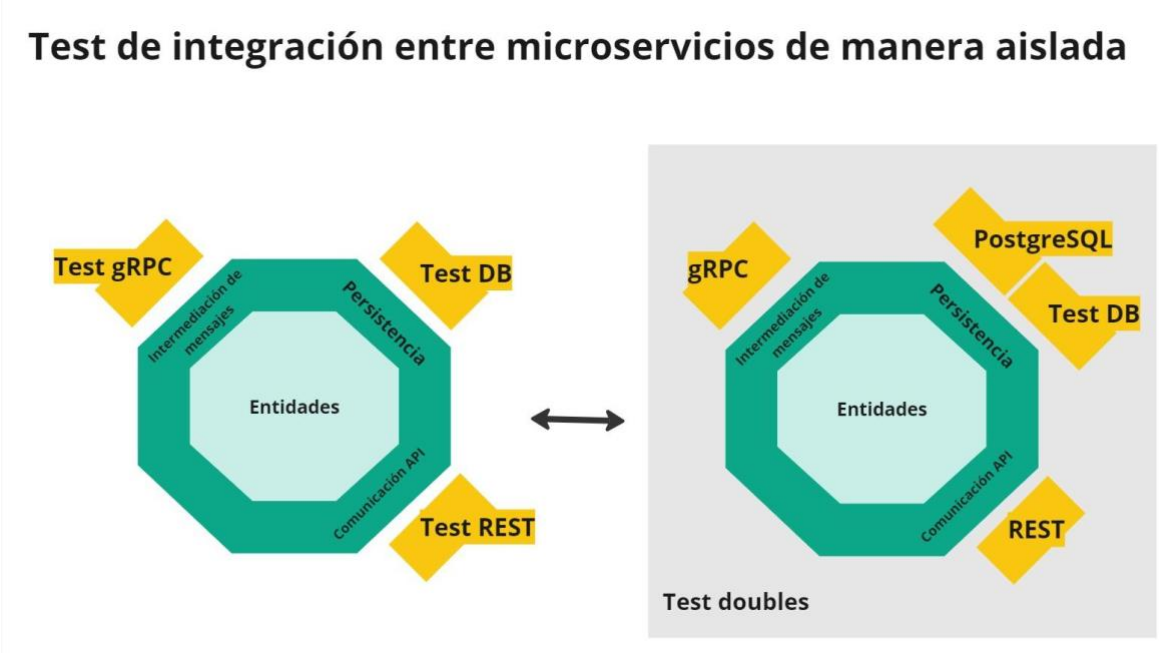

Ilustración 17 Pruebas de integración entre microservicios de manera aislada
Fuente: Elaboración propia

Finalmente, como se ha discutido durante la propuesta de la arquitectura, una de las ventajas más importantes, a nivel de pruebas, es la posibilidad de aislar, por medio de los adaptadores, la aplicación y el dominio de la aplicación, esto permitiendo tanto las pruebas unitarias como se muestra en la ilustración 16, pero también para realizar pruebas de integración como se puede observar en la ilustración 17.

Esta ilustración hace alusión a que se pueden escribir, por parte del equipo desarrollador o el equipo de calidad, adaptadores para pruebas para los diversos puertos del sistema, que permiten crear test dobles para probar de manera aislada diversas funcionalidades que el microservicio tenga, permitiendo además un conjunto de adaptadores que permite establecer una estrategia de pruebas robusta, como por ejemplo, probar la comunicación de API entre dos componentes, pero con un adaptador de base de datos de prueba, que ayuda a evitar el tiempo que se podría perder realizando una consulta para dar el resultado de una operación o empezar un enfoque completamente aislado con sólo adaptadores de pruebas, e ir agregando poco a poco los adaptadores “reales”, para encontrar defectos en algunos de ellos, logrando una predictibilidad muy alta al poder encontrar fácilmente cuál adaptador falló durante una

integración, las posibilidades a nivel de pruebas son muy altas con la arquitectura propuesta.

5.5 Generalización de la arquitectura para interfaces de usuario

Como se puede observar en la ilustración 12, que presenta la arquitectura de software propuesta para Builds 2.0, se hace énfasis en los microservicios que construyen las funcionalidades back-end, pero de igual manera la arquitectura puede ser utilizada, de la misma manera y con todos sus beneficios, en interfaces de usuario, los cuales se describen en la ilustración 12 como un componente que consume la puerta de enlace del proyecto.

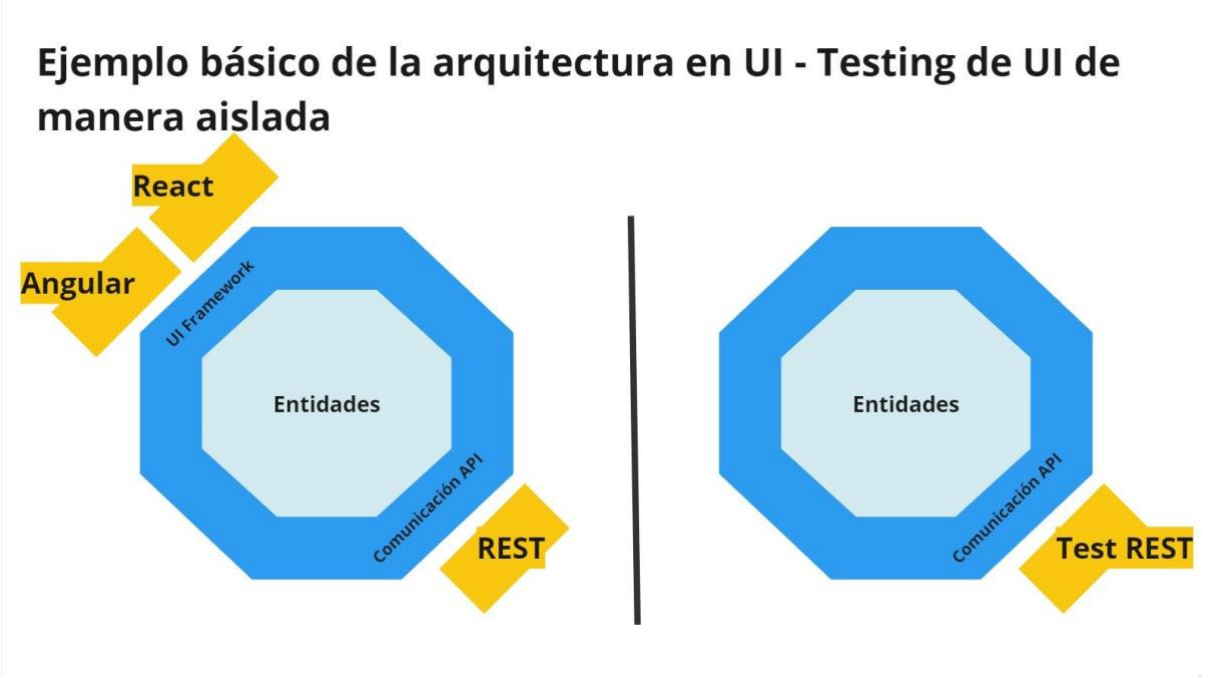

Ilustración 18 Ejemplo básico de la aplicación de la arquitectura en un UI
Fuente: Elaboración propia

Como se puede apreciar en la ilustración 18, la arquitectura propuesta se apega de igual manera a un proyecto que utiliza un framework como Angular o React y que cuenta con un puerto para la comunicación API, por lo que este ejemplo se podría acoplar sin problemas al diagrama de la ilustración 14 como la interfaz de usuario del proyecto,

teniendo las mismas ventajas presentadas durante el capítulo a nivel de mantenimiento y pruebas.

5.6 Otros aspectos no funcionales de la arquitectura

Además de los diversos enfoques aplicables a la arquitectura, que abarcan las distintas capas necesarias para una aplicación como la requerida por Wind River Systems en el proyecto Builds 2.0, la arquitectura propuesta no solo ofrece un robusto marco estructural, sino que también confiere múltiples beneficios en los ámbitos no funcionales, para cada microservicio en el cual se implementa. En los siguientes puntos, se detalla cómo la construcción del sistema conforme a la propuesta contribuye significativamente a los aspectos no funcionales de un sistema de software, algunos de los cuales fueron señalados por los interesados en el proyecto en las entrevistas realizadas, mientras que otros se han añadido debido a su importancia en el contexto de este.

5.6.1 Interoperabilidad

Con respecto a la interoperabilidad del sistema, la arquitectura al definir contratos (puertos) en cada microservicio aislando el dominio, pero extendiendo funcionalidad, permite la construcción ad-hoc de estos para poder interconectar diversas funcionalidades o sistemas externos del proyecto para permitir la interoperabilidad, esta ventaja también emana del diseño arquitectónico modular, proporcionado por los adaptadores, que facilita la creación personalizada de estos para lograr cumplir con el aspecto no funcional ya mencionado, aspecto buscado por el sistema Builds 2.0.

5.6.2 Seguridad

En lo referente al aspecto de la seguridad centrado en el proyecto Builds 2.0 y los resultados obtenidos de las entrevistas, el enfoque principal facilitado por la arquitectura propuesta es su capacidad para adoptar diversos enfoques en la definición de estrategias de pruebas de seguridad, tales como el uso de escáneres de CVS u otros similares. La arquitectura permite descomponer el sistema en capas claramente definidas, lo que

posibilita la aplicación del escáner a diferentes niveles, ya sea por capas o de manera integrada, contribuyendo a obtener resultados más precisos sobre la posible fuente de vulnerabilidades, permitiendo así una corrección más efectiva en el código.

5.6.2 Escalabilidad

En lo que respecta al aspecto de la escalabilidad, la continuidad del enfoque propuesto para Builds 2.0 en relación con los microservicios posibilita la replicación de los diversos microservicios en una arquitectura en la nube. Esta replicación permite incrementar la capacidad de carga del sistema, permitiendo así una respuesta eficaz ante cambios repentinos en el uso o consumo por parte de los usuarios, por lo cual, se pueden explorar distintos tipos de escalado, ya sea horizontal o vertical, de acuerdo con las necesidades específicas de la empresa.

5.7 Desventajas de la arquitectura propuesta

A pesar de las numerosas ventajas de la arquitectura propuesta, detalladas a lo largo de este capítulo, es imperativo reconocer que ninguna solución en software es perfecta, en este sentido es crucial destacar la principal desventaja inherente a la presente propuesta, la cual radica en su propensión a la sobre ingeniería dependiendo de los equipos de desarrollo que la implementen.

El anterior riesgo puede resultar en retrasos en los tiempos de desarrollo, deficiencias en aspectos no funcionales como el rendimiento y en última instancia, una implementación que no se alinee adecuadamente con los objetivos del proyecto, generando una complejidad que oscurece los diversos beneficios que podrían derivarse de su correcta ejecución.

Aunque es importante señalar que estas preocupaciones son inherentes a cualquier arquitectura de software, la presente propuesta es particularmente susceptible a ellas debido a su estrategia modular y filosofía plug-and-play, elementos poco comunes en sistemas de software. Además que la arquitectura propuesta se orienta hacia un enfoque de largo plazo, buscando cumplir con los objetivos del proyecto que se derivaron

de diversas entrevistas con los miembros del equipo, pero que a nivel de negocio podría no ser siempre la mejor opción y la filosofía de obtener un proyecto rápido podría desvanecer algunas ventajas que solo se obtienen manteniendo el desarrollo dentro de los estatutos provistos por la arquitectura.

5.8 Resumen de la arquitectura propuesta

En resumen, la propuesta se fundamenta en la aspiración de adoptar una arquitectura de software plug-and-play, permitiendo la extensión de funcionalidades, por lo que la combinación de la arquitectura hexagonal, la arquitectura de microservicios y los principios de las arquitecturas limpias, culmina en una estructura que se rige por la aplicación de la arquitectura desde la unidad, es decir, a nivel de servicios.

Esta arquitectura se sustenta en aspectos clave como el principio de inversión de dependencias y la inyección de dependencias, logrando una mayor cohesión, evitando el acoplamiento y ofreciendo ventajas como la implementación de contratos definidos específicamente según las necesidades del proyecto, esto se traduce en una mejora en la escalabilidad y la sostenibilidad a largo plazo, elementos cruciales para un proyecto como Builds 2.0.

Adicionalmente, la arquitectura facilita la aplicación de pruebas en paralelo, una característica vital en proyectos de software debido a las múltiples ventajas que las pruebas proporcionan, destacando la facilidad que la arquitectura brinda para el diseño e implementación de pruebas en diversos niveles.

Sin embargo, a pesar de todas las ventajas que la arquitectura propuesta aporta a Wind River Systems, es necesario considerar algunas desventajas, como la posibilidad de acumular una cantidad significativa de código o que una implementación deficiente podría conducir a una sobre ingeniería innecesaria en el proyecto, aunque este último punto depende más de los procesos internos que de la arquitectura en sí misma como fuente principal de errores.

Como conclusión, siguiendo principios de arquitectura de software, se logra dar una referencia desde diversos flancos para proveer una solución que ataque los puntos de dolor de la empresa y que permite construir el producto con todas las ventajas que puede tener un proyecto de software, tomando en cuenta aspectos como la calidad de software ya mencionada, pero también aspectos no funcionales que traen consigo muchas ventajas en ámbitos más allá de solo el desarrollo mismo del sistema, y que puede afectar positivamente ciclos de desarrollo, a los equipos mismos, efectividad a cambios, entre otros beneficios.

CAPÍTULO VI

CONCLUSIONES

6.1 Conclusiones

El principal objetivo de la presente investigación fue proponer un diseño para una arquitectura de software alineada a la filosofía *plug-and-play*, con la meta de ampliar las funcionalidades de un proyecto específico para la empresa Wind River Systems. Además, gran parte de la investigación se centró en destacar la relevancia que puede tener una arquitectura de software bien concebida en una empresa, así como su impacto en diversos aspectos, como la calidad del software.

La arquitectura de software es el cimiento desde donde cualquier proyecto se construye, su importancia es tan significativa que define la estructura que se debe seguir, además que proporciona normativas que pueden traer ventajas o desventajas consideradas durante el proceso de diseño en función de la realidad específica de una empresa o proyecto.

Una arquitectura sólida establece las bases de las diversas características que tendrá un sistema, teniendo esto en cuenta se definieron varios puntos de referencia, que se usaron para proponer lo mejor de la arquitectura hexagonal, la arquitectura de microservicios y la filosofía de arquitecturas limpias en la propuesta para la empresa en estudio.

En el ámbito académico y profesional de la ingeniería de software, resulta crucial emplear referencias bibliográficas, artículos y documentación tecnológica, así como estudiar casos de éxito empresarial, entre otros recursos. Estos materiales representan el conocimiento acumulado por individuos con experiencia tanto académica como práctica, proporcionando directrices valiosas para abordar diversas problemáticas.

En el contexto de esta investigación, adoptar esta estrategia, en lugar de depender exclusivamente del conocimiento propio, otorga mayor solidez a la propuesta. Además, su enfoque puede ser utilizado o adaptado por cualquier empresa o persona para aprovechar los beneficios de su implementación.

Por otra parte, destacar la relevancia de la calidad del software como un fundamento que se construye desde las etapas iniciales de un proyecto, particularmente desde la arquitectura de software, permite evidenciar cómo en ocasiones se subestima, dando lugar a diversos inconvenientes; entre estos se incluyen retrasos en el desarrollo, incumplimiento de las necesidades de los clientes y problemas en aspectos clave del software, como la mantenibilidad, la escalabilidad e interoperabilidad.

Finalmente, es importante recordar que el diseño de la arquitectura de software presentada en la investigación busca solucionar una problemática específica que posee un contexto dado en tiempo y espacio específicos, por lo que en el futuro muchos aspectos presentados en la investigación podrían cambiar dada la rapidez con la que avanza la tecnología, sin embargo, en el futuro mirar atrás una estrategia como la propuesta, podrá ayudar a inspirar nuevas soluciones desde los ojos de otros profesionales.

REFERENCIAS BIBLIOGRÁFICAS

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., C. Martin, R., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifesto for Agile Software Development. Retrieved July 5, 2023, from http://agilemanifesto.org/iso/en/manifesto.html

Booch, G. (2018). The History of Software Engineering. In On computing (p. 108). IEEE SOFTWARE.

Bustio Martínez, L., Coma Peña, Y., & Talavera Bustamante, I. (2013, Marzo 18). (PDF) Arquitectura basada en plugins para el desarrollo de software científico. ResearchGate. Retrieved Junio 16, 2023, from https://www.researchgate.net/publication/315891243_Arquitectura_basada_en_plugins_para_el_desarrollo_de_software_cientifico

Calendamaia. (2014, Enero 10). Eclipse IDE. Genbeta. Retrieved Junio 19, 2023, from https://www.genbeta.com/desarrollo/eclipse-ide

Cockburn, A. (2005). Hexagonal architecture – Alistair Cockburn. Alistair Cockburn. Retrieved July 5, 2023, from https://alistair.cockburn.us/hexagonal-architecture/

Corbetta, P. (2003). Metodología y técnicas de investigación social. Editorial McGraw-Hill.

Elgabry, O. (2019, Abril 30). Plug-in Architecture. and the story of the data pipeline… | by Omar Elgabry | OmarElgabry's Blog. Medium. Retrieved Junio 20, 2023, from https://medium.com/omarelgabrys-blog/plug-in-architecture-dec207291800

Esterkin, J. (2020, Junio 29). Plug-In Architecture - Design Your Software Architecture Using Industry-Standard Patterns. OpenClassrooms. Retrieved Julio 2, 2023, from https://openclassrooms.com/en/courses/6397806-design-your-software-architecture-using-industry-standard-patterns/6896171-plug-in-architecture

Garrido de Paz, J. M. (2021, December 3). Arquitectura Hexagonal - Patrón Puertos y Adaptadores. Hexagonal Me. Retrieved July 5, 2023, from https://jmgarridopaz.github.io/content/arquitecturahexagonal.html

Gefroh, J. (2021, Junio 30). How to Design Software — Plugin Systems | by Joseph Gefroh. Better Programming. Retrieved Julio 1, 2023, from https://betterprogramming.pub/how-to-design-software-plugins-d051ce1099b2

Grinnell, R. M. (1997). Social work research and evaluation. Quantitative and qualitative approaches (5ta Edición ed.). Peacock Publishers.

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2006). Metodología de la investigación (Cuarta ed.). MacGraw-Hill/Interamericana.

Li, D., Chen, Z., Deng, Z., Huang, W., Tang, J., Di, F., Gao, Z., Xie, Q., Liu, L., & Jiang, X. (2018, Marzo 8). A Wide Area Service Oriented Architecture Design for Plug and Play of Power Grid Equipment Play of Power Grid Equipment. ScienceDirect, 129(2018), 353-357. ISSN 1877-0509

Martin, R. C. (2018). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall.

Microsoft. (n.d.). Estilo de arquitectura de microservicios - Azure Architecture Center. Microsoft Learn. Retrieved Octubre 14, 2023, from https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices

Palermo, J. (2008, July 29). The Onion Architecture : part 1. Programming with Palermo. Retrieved July 5, 2023, from https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/

Ports-And-Adapters. (n.d.). Dossier Andreas. Retrieved July 5, 2023, from http://dossier-andreas.net/software_architecture/ports_and_adapters.html

Santos, J. C.S., Sejfia, A., Corrello, T., Gadenkanahalli, S., & Mirakhorli, M. (2019, Agosto 26-30). Achilles' Heel of Plug-and-Play Software Architectures:A Grounded Theory

Based Approach. Joanna C. S. Santos. Retrieved Julio 1, 2023, from https://joannacss.github.io/preprints/fse19-preprint.pdf

Tesis y Masters. (n.d.). Diseño metodológico: ¡aprende a hacerlo con estos ejemplos! Tesis y Másters. Retrieved Julio 5, 2023, from https://tesisymasters.com.co/diseno-metodologico/

Universidad de Costa Rica, Sistema de Estudios de Posgrado [SEP]. (2007). Lineamientos específicos para los tra- bajos finales de investigación aplicada de las maestrías profesionales (Manuscrito sin publicar ed.).

Valera, A., Juste, D., Sánchez, A., Ricolfe, C., Mellado, M., & Olmos, E. (2012, Enero). Aplicación de la Arquitectura Orientada a Servicios Universal Plug-and-Play para facilitar la Integración de Robots Industriales en Líneas de Producción. Revista Iberoamericana de Automática e Informática Industrial RIAI, 9(1), 24-31. 10.1016/j.riai.2011.11.003

Vargas Cordero, Z. R. (2009). LA INVESTIGACIÓN APLICADA: UNA FORMA DE CONOCER LAS REALIDADES CON EVIDENCIA CIENTÍFICA. Revista Educación, 33(1), 155-165. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwit0uKklvn_AhVvSzABHSs6CzIQFnoECDIQAQ&url=https%3A%2F%2Fwww.revistas.ucr.ac.cr%2Findex.php%2Feducacion%2Farticle%2FviewFile%2F538%2F589&usg=AOvVaw3Ivfpn7vepyb4CmwytF-ky&opi=89978449

Wang, S., Avrunin, G. S., & Clarke, L. A. (n.d.). Plug-and-Play Architectural Design and Verification. Plug-and-Play Architectural Design and Verification. Retrieved Junio 20, 2023, from http://ext.math.umass.edu/~avrunin/papers/wang08-plug_play_design_and_verif.pdf

ANEXO

Anexo 1. Entrevistas

Anexo 1.1 Enfoque al Negocio

Propuesta de una arquitectura de software que permita “acoplar” o “desacoplar” componentes que agregan nuevas características al sistema Builds 2.0 de la empresa Wind River Systems

Artefacto para entrevistas.

Objetivo: El propósito de esta entrevista es identificar y analizar el negocio del proyecto Builds 2.0 que se encuentra actualmente en la fase de planificación. Se identificarán casos de usos, necesidades, requerimientos y diferentes aspectos a considerar que ayuden con la propuesta de la investigación.

Mecánica: Entrevista personal a expertos acordes con el perfil de opinión que se busca conocer. El proceso se realizará por medio de Zoom Meetings. Durante la entrevista, se grabará dicha sesión y se transcribe al documento correspondiente.

Perfil de expertos a entrevistar:

Expertos ingenieros administrativos de Wind River que conozcan el negocio y los requerimientos solicitados por el cliente.

Información del experto entrevistado:

- ➢ Nombre
- ➢ Área experto
- ➢ Profesión
- ➢ Fecha

1. Años de experiencia en el puesto actual: ____

2. Nivel académico:

( ) Bachiller ( ) Licenciado ( ) Máster ( ) Doctorado

3. Rol que desempeña actualmente en la empresa: _____

4. ¿Cuál es el objetivo principal de Builds 2.0 y cómo se alinea con la estrategia y metas de Wind River?

5. ¿Cuáles son los principales desafíos o necesidades del negocio que motivaron la creación de Builds 2.0?

6. ¿Qué funcionalidades clave ofrece Builds 2.0 y cómo contribuyen a mejorar la eficiencia o productividad en el entorno empresarial?

7. ¿Cuál es el alcance de Builds 2.0 y cuáles son los hitos y plazos clave para su implementación?

8. ¿Qué tecnologías o plataformas específicas se utilizarán para desarrollar y ejecutar este aplicativo de software?

9. ¿Cuáles son los criterios de éxito para Builds 2.0 y cómo se medirá su impacto en el negocio?

10. ¿Qué otros requerimientos no funcionales consideras que se necesite implementar en Builds 2.0 y con qué enfoque?

Resultados de las entrevistas enfocados al proyecto (Anexo 1.1)

Entrevista 1: Erick Arroyo Blanco

Propuesta de una arquitectura de software que permita “acoplar” o “desacoplar” componentes que agregan nuevas características al sistema Builds 2.0 de la empresa Wind River Systems

Experto: Gerente en Ingeniería de Software

Profesión: Máster en Gerencia de Tecnología de la Información

Fecha: 1 de noviembre del 2023

Presentación:

El propósito de esta entrevista es identificar y analizar el negocio del proyecto Builds 2.0 que se encuentra actualmente en la fase de planificación. Se identificarán casos de usos, necesidades, requerimientos y diferentes aspectos a considerar que ayuden con la propuesta de la investigación.

La información presentada es y será estrictamente confidencial y se utilizará únicamente con fines académicos. Gracias por su comprensión.

1. Años de experiencia en el puesto actual: 1 año y 10 meses

2. Nivel académico:

( ) Bachiller ( ) Licenciado (X) Máster ( ) Doctorado

3. Rol que desempeña actualmente en la empresa: Gerente encargado de la iniciativa Builds 2.0

4. ¿Cuál es el objetivo principal de Builds 2.0 y cómo se alinea con la estrategia y metas de Wind River?

Una de las principales metas u objetivos estratégicos de Wind River es básicamente la solución de una plataforma cloud que permita implementar cada uno de los procesos de software para sistemas embebidos o sistemas inteligentes. Una de esas etapas es el proceso de buildeo tanto en sistemas operativos como de aplicaciones que se ejecuta en el sistema.

Entonces, esa es una de las principales goals a nivel de negocios, y nos permite ampliar tanto del mercado que tiene la empresa que ha sido muy enfocado incluso en el mercado militar, o en mercado de la industria médica. Aquí la idea es expandir para poder entrar a otros mercados como el de automotriz, el de telecomunicaciones, internet de las cosas en general y por ende se asocia una propuesta de una nueva arquitectura flexible y extensible que nos permita cubrir varios de los casos de usos reales que tienes nuestro clientes. Es decir, poder ampliar la cobertura y de casos de usos reales de clientes a partir de una arquitectura flexible y extensible.

5. ¿Cuáles son los principales desafíos o necesidades del negocio que motivaron la creación de Builds 2.0?

En este caso, los 2 principales sistemas operativos de Wind River son Vxworks y Wind River Linux, sin embargo, realmente para abarcar un mayor porcentaje del mercado de sistemas inteligentes, internet de las cosas y de sistemas embebidos, hay una gran parte de clientes que no utilizan estos sistemas operativos, sino que tiene versiones personalizadas que ellos mismo crearon basado en Linux que es Open Source o incluso hay industrias como el automotriz que tiene un sistema operativo más dedicado al área de “infotainment”, que son sistemas de entretenimiento que tienen los vehículos que no tiene el caso de uso de Wind River Linux o de VxWorks.

Entonces esta fue la principal necesidad de negocio que se ha venido identificando y por ende la necesidad de darle fuerza o impulso a la iniciativa Builds 2.0 que nos permite abarcar estas necesidades de los clientes.

6. ¿Qué funcionalidades clave ofrece Builds 2.0 y cómo contribuyen a mejorar la eficiencia o productividad en el entorno empresarial?

En la solución per-sé, anda apuntando a aplicar 3 criterios de arquitecturas muy importantes, la primera es modularidad, el otro es extensibilidad, y por último flexibilidad.

Entonces bajo una arquitectura que logre cumplir estos 3 parámetros generales, podemos ser flexibles para ajustar una solución a las necesidades específicas del cliente y no hacerlo al revés, que a la fuerza le estamos tratando de dar el cliente algo para que lo use, sino más bien nosotros adaptarnos a eso, lo cual impacta evidentemente la proactividad del cliente y tambien hay eficiencia.

Luego la extensibilidad, nos va a permitir tanto internamente, nosotros como Wind River, como a los clientes, implementar diferentes casos de usos, soportando ya funciones básicas y funciones extras que ellos necesitan.

Y por último con la modularidad, a nivel de productividad y eficiencia interno de Wind River, impacta directamente a los Scrum Teams que estén involucrados en esta área o funcionalidad de las plataformas Wind River Studio. Porque la idea es mantener una sola arquitectura que sea lo suficientemente flexible para soportar los diferentes sistemas operativos y aplicaciones.

7. ¿Cuál es el alcance de Builds 2.0 y cuáles son los hitos y plazos clave para su implementación?

El alcance de 2.0 lo va a ir dictando mucho los requerimientos de usuario que nos vengan en los próximos meses. Ya tenemos un diseño base que es con la que se va a implementar de manera que el alcance lo vamos a ir ajustando de forma incremental, es decir, se va a generar un MVP (producto mínimo viable) que respondan a los casos de usos de los clientes. Un ejemplo es el sistema operativo Android, que nos va a permitir lanzar esas primeras versiones de Builds 2.0 y que soporte esos casos de usos relacionados con Android y así con cualquier de los clientes o hasta incluso “stakeholders” internos de la empresa que lo quieren hacer.

Los hitos y plazos también lo va a ir dictando mucho la urgencia de negocio y la prioridad que tengan los clientes para que puedan soportar sus sistemas operativos y

sus aplicaciones. Se tienen en términos generales planteado que el año 2024 sea la mayor parte del alcance de la solución, de manera que ya quede establecida y que más bien lo que se tenga que realizar es mantenimiento y ajuste mínimo de la arquitectura para soportar nuevos casos de usos. Sin embargo, el plazo no es fijo, sino se va a ir implementando por partes y ajustándose las prioridades de negocio.

8. ¿Qué tecnologías o plataformas específicas se utilizarán para desarrollar y ejecutar este aplicativo de software?

En términos generales, ya se desarrolló un documento de arquitectura que fue previamente revisado con los arquitectos de la empresa de manera que se hizo un análisis de las tecnologías de manera que se respondieran al problema que se quiere resolver. A nivel de compatibilidad también se hizo un fuerte análisis en el documento de arquitectura de manera que lenguajes como GoLang, han sido decididos que se van a utilizar, también en comunicación entre microservicios se va a utilizar gRPC que mejora tanto la eficiencia como la operabilidad de los microservicios que forman la arquitectura, y también pues básicamente desde el punto de vista de seguridad aplicar las prácticas correctas a nivel de tecnologías para abordar la parte de vulnerabilidad del pleno desarrollo de la solución.

Desde el punto de vista de la infraestructura, se ha decidido seguir con las pautas que indica el equipo de la infraestructura de la plataforma de Wind River Studio de manera que se tengan los diferentes componentes, como los Helm charts, terraform, conductor que permitan instalar o desinstalar el componente Builds 2.0 sin mayor dependencia con otros componentes o bloques que conforman Wind River Studio. Entonces, básicamente las decisiones de arquitectura y de utilizar microservicios se basa también en responder las necesidades de los clientes y responder a los 3 principios que mencioné anteriormente que eran modularidad, extensibilidad y flexibilidad.

9. ¿Cuáles son los criterios de éxito para Builds 2.0 y cómo se medirá su impacto en el negocio?

Básicamente, el criterio de éxito para Builds 2.0, es que se logre ser una solución efectiva para los problemas y casos de usos reales de nuestros clientes. Es una solución que va a ser utilizada para uno de nuestros primeros clientes que ya están negociadas con Wind River Studio, esto es a nivel de éxito y a nivel de negocio.

A nivel operativo, pensaría que aumentar la eficiencia y la efectividad tanto los Scrum Teams que están alrededor de los mantenimiento de estas funcionalidades como en la mejora de eficiencia de calidad y también en la eficiencia de que otros grupos internamente de Wind River Studio, puedan adaptarse a esta solución, es decir, que la puedan integrar y que grupos como el de VxWorks o Linux puedan fácilmente adoptar esta solución más estándar y más genérica y dejar de lado los builds systems que tal vez hay muchos casos de usos que no puedan cubrir.

Su impacto se medirá mucho a nivel de la aceptación del cliente con respecto a la solución, a nivel de calidad, defectos, rendimiento y seguridad que tenga la solución, y tercer criterio que es cómo esta solución va a impactar de forma positiva la eficiencia, la operabilidad y adaptación dentro de Wind River Studio. Con esto se podrá medir el impacto tanto de interno como el externo.

10. ¿Qué otros requerimientos no funcionales consideras que se necesite implementar en Builds 2.0 y con qué enfoque?

Pensaría que con Builds 2.0, incluso la idea es cambiar mucho del paradigma de cómo se han desarrollado previamente otros componentes dentro de Wind River studio. Por ejemplo, en requerimientos no funcionales, la idea es que dentro de los “test plans” que se definan es poder ejecutar pruebas de rendimiento, donde se evalúa tanto la escalabilidad, como la resiliencia que tengan los microservicios que vamos a desarrollar, esto es una parte.

La otra parte que también se va a tomar en cuenta, evidentemente es la cuestion de seguridad, de manera que desde el diseño del desarrollo se cumpla los criterios de seguridad que exige la compañía para hacer “compliance”, llámese la debida implementación con el control de accesos que tiene los usuarios, también si nos

metemos un poquito en la parte de vulnerabilidad de ciberseguridad, asegurar que todos los microservicios no tengan una grande cantidad de CVEs críticos, sino que se mantengan dentro de los SLA o “Services Labels Agreements” que tenemos y por último tambien aplicar a nivel de seguridad, lo que indican otros estándares que estamos siguiendo, asi como OWASP, o CIS que también es del gobierno de Estados Unidos que indican las pautas necesarias que esperan los entes gubernamentales o clientes que requieren seguridad bastante elevadas.

Entonces a nivel de requerimiento no funcionales es muy enfocado a nivel de “performance” y muy enfocado a nivel de seguridad.

Anexo 1.2. Enfoque Técnico

Propuesta de una arquitectura de software que permita “acoplar” o “desacoplar” componentes que agregan nuevas características al sistema Builds 2.0 de la empresa Wind River Systems

Artefacto para entrevistas.

Objetivo: El propósito de esta entrevista es identificar y analizar la arquitectura de Builds 2.0, así como los aspectos técnicos. Se identificarán casos de usos, necesidades, requerimientos y diferentes aspectos a considerar que ayuden con la propuesta de la investigación.

Mecánica: Entrevista personal a expertos acordes con el perfil de opinión que se busca conocer. El proceso se realizará por medio de Zoom Meetings. Durante la entrevista, se grabará dicha sesión y se transcribe al documento correspondiente.

Perfil de expertos a entrevistar:

Expertos ingenieros técnicos de la empresa que conozcan la arquitectura actual y los requerimientos solicitados por el cliente.

Información del experto entrevistado:

- ➢ Nombre
- ➢ Área experto
- ➢ Profesión
- ➢ Fecha

1. Años de experiencia en el puesto actual: ____

2. Nivel académico:

( ) Bachiller ( ) Licenciado ( ) Máster ( ) Doctorado

3. Rol que desempeña actualmente en la empresa: _____

4. ¿Cuál es la arquitectura técnica de Builds 2.0 y qué tecnologías se utilizan en su desarrollo?

5. ¿Cuáles son los requisitos de hardware y software necesarios para ejecutar Builds 2.0 de manera eficiente?

6. ¿Cómo se abordan los desafíos de escalabilidad y rendimiento en el diseño y desarrollo de Builds 2.0?

7. ¿Qué enfoque se ha tomado para la gestión de base de datos y almacenamiento de datos en relación con Builds 2.0?

8. ¿Cuál es la estrategia de seguridad implementada en el aplicativo para proteger los datos y garantizar la integración de la información?

9. ¿Cómo se gestionan las actualizaciones de software para mantener Builds 2.0 seguro y actualizado?

10. ¿Qué herramientas de monitoreo y gestión se utilizan para garantizar el funcionamiento óptimo del aplicativo y para identificar y resolver problemas técnicos?

Resultados de las entrevistas enfocados al área técnica (Anexo 1.2)

Entrevista 1: Jing Du

Propuesta de una arquitectura de software que permita "acoplar" o "desacoplar" componentes que agregan nuevas características al sistema Builds 2.0 de la empresa Wind River Systems

Experto: Líder de Equipo

Profesión: Ingeniero de Software

Fecha: 24 de octubre del 2023

El propósito de esta entrevista es identificar y analizar la arquitectura de Builds 2.0, así como los aspectos técnicos. Se identificarán casos de usos, necesidades, requerimientos y diferentes aspectos a considerar que ayuden con la propuesta de la investigación.

La información presentada es y será estrictamente confidencial y se utilizará únicamente con fines académicos. Gracias por su comprensión.

1. Años de experiencia en el puesto actual: 15 años de experiencia

2. Nivel académico:

(X) Bachiller ( ) Licenciado ( ) Máster ( ) Doctorado

3. Rol que desempeña actualmente en la empresa:

Senior Member of Technical Staff - Team Lead / Tech Lead

4. ¿Cuál es la arquitectura técnica de Builds 2.0 y qué tecnologías se utilizan en su desarrollo?

La arquitectura técnica de Builds 2.0 se basa en la arquitectura de microservicios. En esta arquitectura, el proyecto se descompone en una serie de servicios independientes

que se comunican entre sí a través de gRPC. Cada uno de los servicios se encarga de una funcionalidad específica del proyecto.

Con esta arquitectura, permitimos una serie de ventajas clave, como la modularidad, la escalabilidad y la flexibilidad. Los servicios pueden desarrollarse en diferentes lenguajes de programación y tecnologías, lo que permite a los equipos de desarrollo utilizar las herramientas que mejor se adapten a sus necesidades. Además, si un servicio necesita escalar debido a una mayor demanda, se puede hacer de manera aislada sin afectar a otros componentes del proyecto.

Con respecto a las tecnologías, se utilizan kubernetes, docker, kong y gRPC para DevSecOps:

- Kubernetes: en donde la utilizaremos para la orquestación y gestión de contenedores, lo que permite una escalabilidad eficiente y una alta disponibilidad de los servicios de proyecto.
- Docker: en la cual ofrece la capacidad de contenerizar las aplicaciones y sus dependencias, lo que facilita el despliegue y la gestión consistente en diversos entornos.
- Kong: Se emplea como una puerta de enlace API, proporcionando una capa de seguridad, administración y monitorización para las APIs del proyecto.
- gRPC: Este protocolo de comunicación de alto rendimiento se usa para facilitar la comunicación entre los microservicios del proyecto, permitiendo un intercambio de datos eficiente y confiable.

Y en desarrollo, las tecnologías que se utilizarán son:

- Go (Golang): Se utiliza Golang como lenguaje de programación principal para el desarrollo de los microservicios de Builds 2.0. Este lenguaje es conocido por su eficiencia y rendimiento, lo que es especialmente beneficioso en una arquitectura de microservicios.

- PostgreSQL: Esta base de datos relacional se utiliza para el almacenamiento y gestión de datos. Ofrece robustez y escalabilidad, lo que es esencial para un proyecto de estas características.

5. ¿Cuáles son los requisitos de hardware necesarios para ejecutar Builds 2.0 de manera eficiente?

No existen requisitos de hardware específicos, ya que el enfoque principal de este proyecto es su completa flexibilidad y escalabilidad. El proyecto ha sido diseñado teniendo en cuenta un modelo de funcionamiento "on demand" o "bajo demanda", lo que significa que se adapta a las necesidades específicas en tiempo real sin requerir una infraestructura de hardware dedicada. Es decir, 100% cloud.

6. ¿Cómo se abordan los desafíos de escalabilidad y rendimiento en el diseño y desarrollo de Builds 2.0?

Una de las primeras consideraciones clave que abordamos es la implementación de un sistema de balanceo de carga robusto. Este tipo de “load balancer” es una técnica que distribuye el tráfico de entrada entre múltiples servidores o recursos de manera equitativa. Esto nos asegura que ningún servidor se sobrecargue, lo que es fundamental tanto para la escalabilidad vertical como horizontal.

7. ¿Qué enfoque se ha tomado para la gestión de base de datos y almacenamiento de datos en relación con Builds 2.0?

Hemos adoptado un enfoque meticuloso y altamente personalizado para la gestión de bases de datos y el almacenamiento de datos. Nuestra estrategia se centra principalmente en dos aspectos: la configuración del usuario y los resultados generados en cada ejecución (outputs).

8. ¿Cuál es la estrategia de seguridad implementada en el aplicativo para proteger los datos y garantizar la integración de la información?

Para lo que es la estrategia de seguridad de Builds 2.0 es una combinación de medidas diseñadas para proteger los datos y garantizar la integración segura de la información. Estas medidas incluyen un mecanismo de control de acceso, RBAC (Role-Based Access Control) y el uso de una VPN (Red Privada Virtual).

El mecanismo de control de acceso se encarga de decidir quién tiene acceso a qué partes de Builds 2.0. Esto se logra mediante la autenticación de usuarios y la autorización basada en roles, que sería Role-Based Access Control (RBAC), que define quién puede realizar qué acciones en función de su rol dentro de Wind River. Los roles se asignan de manera jerárquica, y se ajustan de acuerdo a las necesidades específicas de cada usuario o grupo, lo que garantiza que la información se comparte solo con quienes tienen la autorización adecuada.

9. ¿Cómo se gestionan las actualizaciones de software para mantener Builds 2.0 seguro y actualizado?

Para mantener el Builds 2.0 seguro y actualizado, es esencial implementar una estrategia de actualizaciones planificadas, utilizar CI/CD para automatizar el proceso, aplicar parches de seguridad de manera oportuna, realizar pruebas rigurosas y mantener una comunicación efectiva con el equipo y los clientes. El monitoreo continuo y la gestión de dependencias son fundamentales para garantizar un funcionamiento seguro y eficiente.

10. ¿Qué herramientas de monitoreo y gestión se utilizan para garantizar el funcionamiento óptimo del aplicativo y para identificar y resolver problemas técnicos?

Existen varios mecanismos de monitoreo como health check para cada microservicio, métricas implementadas para observar el comportamiento, recolección de los errores con Fluent bit y monitoreo de sus status como Prometheus.

Entrevista 2: Jeison Melendez

Propuesta de una arquitectura de software que permita “acoplar” o “desacoplar” componentes que agregan nuevas características al sistema Builds 2.0 de la empresa Wind River Systems

Experto: Desarrollador Senior del Equipo

Profesión: Ingeniero de Software

Fecha: 31 de octubre del 2023

El propósito de esta entrevista es identificar y analizar la arquitectura de Builds 2.0, así como los aspectos técnicos. Se identificarán casos de usos, necesidades, requerimientos y diferentes aspectos a considerar que ayuden con la propuesta de la investigación.

La información presentada es y será estrictamente confidencial y se utilizará únicamente con fines académicos. Gracias por su comprensión.

1. Años de experiencia en el puesto actual: 1 año de experiencia

2. Nivel académico:

( ) Bachiller (X) Licenciado ( ) Máster ( ) Doctorado

3. Rol que desempeña actualmente en la empresa:

Senior Desarrollo de Software

4. ¿Cuál es la arquitectura técnica de Builds 2.0 y qué tecnologías se utilizan en su desarrollo?

La arquitectura de Builds 2.0 se basa en una arquitectura de micro-servicios en cloud. En donde se pretende distribuir, con una serie de servicios, la construcción de

distintos sistemas operativos. Cada uno de estos microservicios van a estar desarrollados en Go y se van a comunicar entre ellos por medio de gRPC.

5. ¿Cuáles son los requisitos de hardware necesarios para ejecutar Builds 2.0 de manera eficiente?

Debido a que el aplicativo está enfocado en cloud, no existen requisitos de hardware como tal y si se requiere ejecutar el proyecto localmente, podría funcionar normalmente bien con una computadora con conexión al internet.

6. ¿Cómo se abordan los desafíos de escalabilidad y rendimiento en el diseño y desarrollo de Builds 2.0?

Como se mencionó anteriormente, esta arquitectura está enfocada en cloud, por lo que para poder escalar y monitorear el rendimiento de la arquitectura, se va a utilizar el servicio de Kubernetes. Esto nos permite por medio de HPA (horizontal pod autoscaling) manejar la escala de cada microservicio depende de la carga que tiene en el momento.

7. ¿Qué enfoque se ha tomado para la gestión de base de datos y almacenamiento de datos en relación con Builds 2.0?

Con respecto a la base de datos, vamos a utilizar el mismo mecanismo de base de datos que tenemos actualmente en el equipo, que es una base de datos postgres, en la que se va a encontrar en el servicio de Kubernetes y todos los microservicios se van a conectar a ella.

8. ¿Cuál es la estrategia de seguridad implementada en el aplicativo para proteger los datos y garantizar la integración de la información?

Se está utilizando una estrategia de autenticación por medio de JWT, utilizando el servicio Tozny implementado en la empresa. Por lo que cada vez que se requiere ingresar a la aplicación se debe de ingresar con el Token a cada microservicio.

9. ¿Cómo se gestionan las actualizaciones de software para mantener Builds 2.0 seguro y actualizado?

Esto se mantiene por medio de imágenes de contenedores en docker, estas se suben a Harbor y de ahí son consumidas a kubernetes donde se actualiza cada microservicio.

10. ¿Qué herramientas de monitoreo y gestión se utilizan para garantizar el funcionamiento óptimo del aplicativo y para identificar y resolver problemas técnicos?

En la empresa estamos utilizando un servicio que se llama health check, por lo que cada microservicios va a levantar un server en donde la aplicación se va a conectar y se encontrará en constante monitoreo de cada uno de los microservicios. Por lo que en caso de que ocurra un error, esto lo notifican al health check para que nosotros como los desarrolladores podamos realizar lo necesario para corregir el error.

Anexo 1.3. Enfoque al Proyecto

Propuesta de una arquitectura de software que permita “acoplar” o “desacoplar” componentes que agregan nuevas características al sistema Builds 2.0 de la empresa Wind River Systems

Artefacto para entrevistas.

Objetivo: El propósito de esta entrevista es identificar y analizar la arquitectura que se encuentra actualmente en la empresa seleccionada. Se identificarán casos de usos, necesidades, requerimientos y diferentes aspectos a considerar que ayuden con la propuesta de la investigación.

Mecánica: Entrevista personal a expertos acordes con el perfil de opinión que se busca conocer. El proceso se realizará por medio de Zoom Meetings. Durante la entrevista, se grabará dicha sesión y se transcribe al documento correspondiente.

Perfil de expertos a entrevistar:

Expertos ingenieros de la empresa que conozcan la arquitectura actual y los requerimientos solicitados por el cliente, así como el dueño del producto, líderes técnicos, y/o desarrolladores involucrados.

Información del experto entrevistado:

- ➢ Nombre
- ➢ Área experto
- ➢ Profesión
- ➢ Fecha

1. ¿Cuál es su profesión actual?

2. Nivel académico:

( ) Bachiller ( ) Licenciado ( ) Máster ( ) Doctorado

3. Rol que desempeña actualmente en la empresa: _____

4. ¿En qué se basa Builds 2.0 y cuál es su objetivo?

5. ¿Cuáles son los resultados obtenidos al ejecutar el proyecto?

6. Explique un ejemplo de un proyecto que pueda ser relacionado y ejecutado en Builds 2.0

7. Explique la arquitectura actual de Builds 2.0

8. ¿Qué problemáticas identifica con la arquitectura actual?

Resultados de las entrevistas enfocados al proyecto (Anexo 1.3)

Entrevista 1: Kendall González

Propuesta de una arquitectura de software que permita "acoplar" o "desacoplar" componentes que agregan nuevas características a al sistema Builds 2.0 de la empresa Wind River System

Experto: Líder técnico del proyecto Builds 2.0

Profesión: Lic. Ingeniería de Software

Fecha: 12 de julio del 2023

Presentación:

El propósito de esta entrevista es identificar y analizar la arquitectura que se encuentra actualmente en la empresa seleccionada. Se identificarán casos de usos, necesidades, requerimientos y diferentes aspectos a considerar que ayuden con la propuesta de la investigación.

La información presentada es y será estrictamente confidencial y se utilizará únicamente con fines académicos. Gracias por su comprensión.

1. ¿Cuál es su profesión actual?

Líder técnico y de equipo

2. Nivel académico:

( ) Bachiller (X) Licenciado ( ) Máster ( ) Doctorado

3. Rol que desempeña actualmente en la empresa: Líder técnico y de equipo.

4. ¿En qué se basa Builds 2.0 y cuál es su objetivo?

Para entrar a detalle de Builds 2.0, primero es necesario entender que esta aplicación es más como un cambio de filosofía de las funcionalidades actuales y se busca más a como una filosofía de *plug-and-play* que pueda gestionar las extensiones de una manera más ágil y también se puede considerar como orquestador o facilitador.

Actualmente, esta aplicación se puede resumir en el siguiente diagrama:

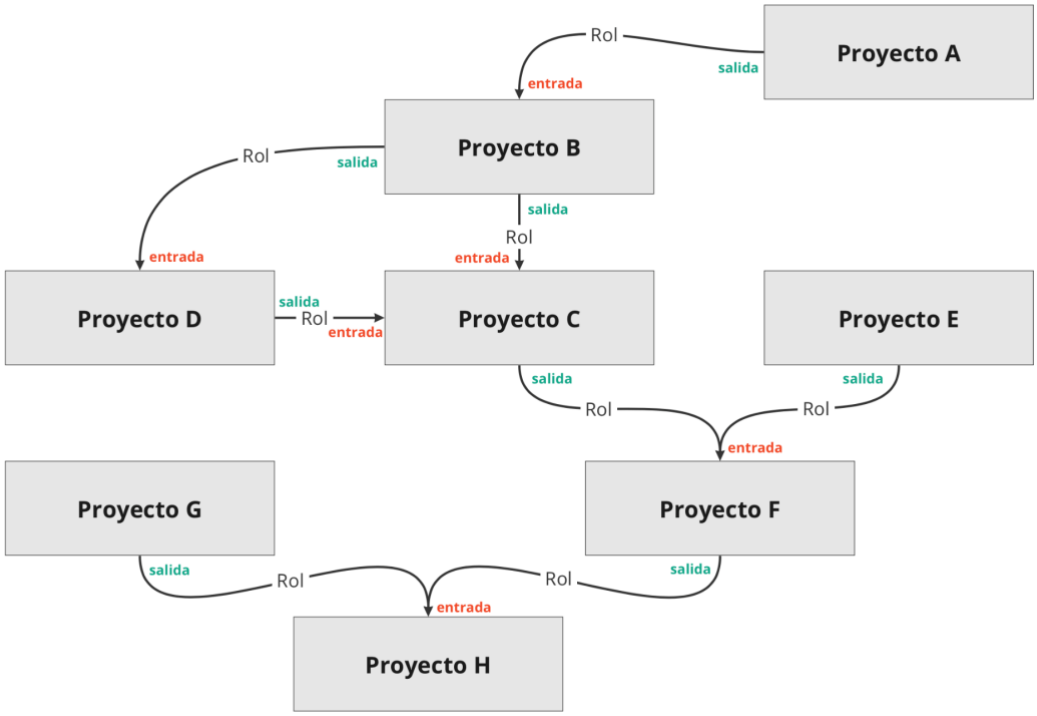

Diagrama presentado por el ingeniero

Cada caja mostrada en el diagrama se representa como “proyectos” y se pueden relacionar entre sí mediante un rol, y representa la responsabilidad que debe de cumplir para que el siguiente proyecto dependiente pueda funcionar, así como sus configuraciones. Por lo que el objetivo de este proyecto es construir proyectos que se puedan relacionar de forma dinámica las dependencias y sus configuraciones.

Cada uno de los proyectos, son representados como sistemas operativos

5. ¿Cuáles son los resultados obtenidos al ejecutar el proyecto?

Cada uno de estos proyectos pueden recibir como “inputs” artefactos o recursos producidos por otros proyectos (outputs). Estos “outputs” pueden ser probados, y enviados como adjunto para ser utilizados en otros proyectos. Además, estos mismos tienen sus propias versiones ya que pueden contener distintas respuestas.

6. Explique un ejemplo de un proyecto que pueda ser relacionado y ejecutado en Builds 2.0

Cada uno de estos proyectos son un sistema operativo diferente que pueden dar como resultado distintos artefactos que necesitan los proyectos dependientes a ellos.

7. Explique la arquitectura actual de Builds 2.0

Existen microservicios internos que llegan a comunicar con sus componentes dependientes que se encargan de realizar sus ejecuciones para la obtención de sus artefactos. Para ello, el usuario utiliza Builds 2.0 y solicita la construcción de un nuevo sistema operativo con las configuraciones seleccionadas por el mismo, éste se encarga de transcribir toda esta información y la transforma en una “tarea” para que pueda ser procesada por otro componente de la empresa. Esta “tarea” debe de ser capaz de procesar cualquier tipo de sistema operativo con sus respectivas configuraciones. Por lo que, existen “tareas” plantillas que representan el sistema operativo que el usuario está solicitando.

8. ¿Qué problemáticas identifica en la arquitectura actual?

- La lógica de negocios está distribuida en múltiples microservicios, por lo que puede generar problemas de coherencia y aumentar la complejidad.
    - La dificultad en el mantenimiento y la evolución del software ya que la lógica de negocios es un poco confusa.
    - Actualmente, algunas características pueden no funcionar de forma correcta o a como se espera
    - La vulnerabilidad de seguridad
- Dependencias

    - En Builds 2.0, los cambios de otros microservicios pueden llegar a afectar a otros servicios que dependen de él.
    - Si un microservicio llega a fallar, toda la aplicación dejará de funcionar.
    - Builds 2.0 no tiene la capacidad de segmentar ni aislar ciertas funcionalidades para poder escalar, actualizar o reemplazar partes específicas del sistema sin afectar a otros por su alta dependencia entre microservicios.
    - Existen problemas de consistencia y sincronización de datos.
    - Debido a las dependencias, las pruebas automatizadas son mucho más complejas.
    - Se observa latencia en la comunicación.
    - Estas dependencias le agregan complejidad a los aspectos de seguridad.
- No se están implementando las pruebas unitarias necesarias y de integración para cada uno de los microservicios.
    - Al no tener pruebas unitarias, ni automatizadas, aumenta la probabilidad de que existan errores y fallos en el código base.
    - Dificulta la depuración e identificación temprana de errores antes de su despliegue a producción.
    - La falta de pruebas unitarias y automatizadas puede requerir procesos manuales que retrasan la entrega de nuevas funcionalidades.
    - Maximiza el tiempo y recursos invertidos en pruebas manuales.
    - Dificulta la trazabilidad e identificación de la causa original del problema.
    - Provoca cambios no previstos en el comportamiento del software y causa una inestabilidad generalizada en el código base del proyecto.
    - Durante una integración de sistemas, se vuelve aún más complejo.
    - Dificulta la refactorización y optimización del código base.
    - Deja pendiente muchas deudas técnicas sin identificar.
    - La falta de estas pruebas, minimiza la confianza durante el proceso de desarrollo y no solo en los desarrolladores, sino también afecta a los usuarios directos.
    - Minimiza la escalabilidad del equipo de desarrollo.

    - Sin pruebas unitarias ni automatizadas, es mucho más probable que algunas funcionalidades se pierdan con el tiempo debido a cambios en el código que no se validan, lo que impacta la experiencia del usuario.
- Tampoco se realizan pruebas de carga y estrés por lo que no se sabe si existe un rendimiento deficiente en momento críticos y que afecte la experiencia del usuario.

Entrevista 2: Juan Alvarado

Propuesta de una arquitectura de software que permita “acoplar” o “desacoplar” componentes que agregan nuevas características al sistema Builds 2.0 de la empresa Wind River Systems

Experto: Ingeniero desarrollador de Builds 2.0

Profesión: Ingeniero de Software

Fecha: 12 de julio del 2023

Presentación:

El propósito de esta entrevista es identificar y analizar la arquitectura que se encuentra actualmente en la empresa seleccionada. Se identificarán casos de usos, necesidades, requerimientos y diferentes aspectos a considerar que ayuden con la propuesta de la investigación.

La información presentada es y será estrictamente confidencial y se utilizará únicamente con fines académicos. Gracias por su comprensión.

1. ¿Cuál es su profesión actual?

Ingeniero de Software

2. Nivel académico:

(X) Bachiller ( ) Licenciado ( ) Máster ( ) Doctorado

3. Rol que desempeña actualmente en la empresa: Desarrollador de Builds 2.0

4. ¿En qué se basa Builds 2.0 y cuál es su objetivo?

En lo que respecta al Builds 2.0, se enfoca más en adoptar una mentalidad de *plug and play*, lo que permite una gestión de extensiones de forma más ágil, de la cual se puede considerar como un facilitador de procesos.

5. ¿Cuáles son los resultados obtenidos al ejecutar el proyecto?

Los proyectos tienen la capacidad de recibir como entrada recursos o salidas de otros proyectos (output). Además, de que estos pueden llegar a ser sometidos a pruebas y luego se envían como salidas a otros proyectos.

6. Explique un ejemplo de un proyecto que pueda ser relacionado y ejecutado en Builds 2.0

Observa que estos proyectos se comportan de manera similar a sistemas operativos independientes y dependientes, lo que implica que pueden generar recursos que son necesitados por otros proyectos en forma de salida. En esencia, cada proyecto representa una plataforma única que proporciona resultados específicos y esenciales para aquellos proyectos que lo utilizan

7. Explique la arquitectura actual de Builds 2.0

Dentro de nuestro sistema, hemos implementado microservicios internos que se comunican con sus componentes dependientes para llevar a cabo sus ejecuciones y obtener los artefactos necesarios. Cuando un usuario desea crear un nuevo sistema operativo con configuraciones personalizadas, utiliza Builds 2.0 . Esta herramienta transcribe toda la información proporcionada por el usuario en una tarea, que luego es procesada por otro componente. La versatilidad es clave, ya que estas tareas deben ser capaces de manejar cualquier tipo de sistema operativo con sus configuraciones específicas. Para ello, hemos creado plantillas de tareas que representan los distintos tipos de sistemas operativos que el usuario puede solicitar.

8. ¿Qué problemáticas identifica en la arquitectura actual?

En este sistema, lamentablemente, no se ha dado la debida atención a la implementación de pruebas unitarias y de integración para cada uno de estos microservicios. Además, no se están llevando a cabo pruebas de estrés, lo que significa que desconocemos si hay problemas de rendimiento en momentos críticos que puedan afectar negativamente la experiencia de usuario.