

Arquitectura en **Capas** **Monolito**

Luis Felipe Gutierrez
Daniel Perez
Maria Paula Rodriguez

GRUPO 1



Arquitectura de Software



Stack Designado

Capas Monolítica (1 App, 1 BD)

F&B: Java + Jakarta EE + JSF (Java)

DB: (GlassFish + Payara) + MariaDB

Arquitectura Monolítica (Capas Monolito)

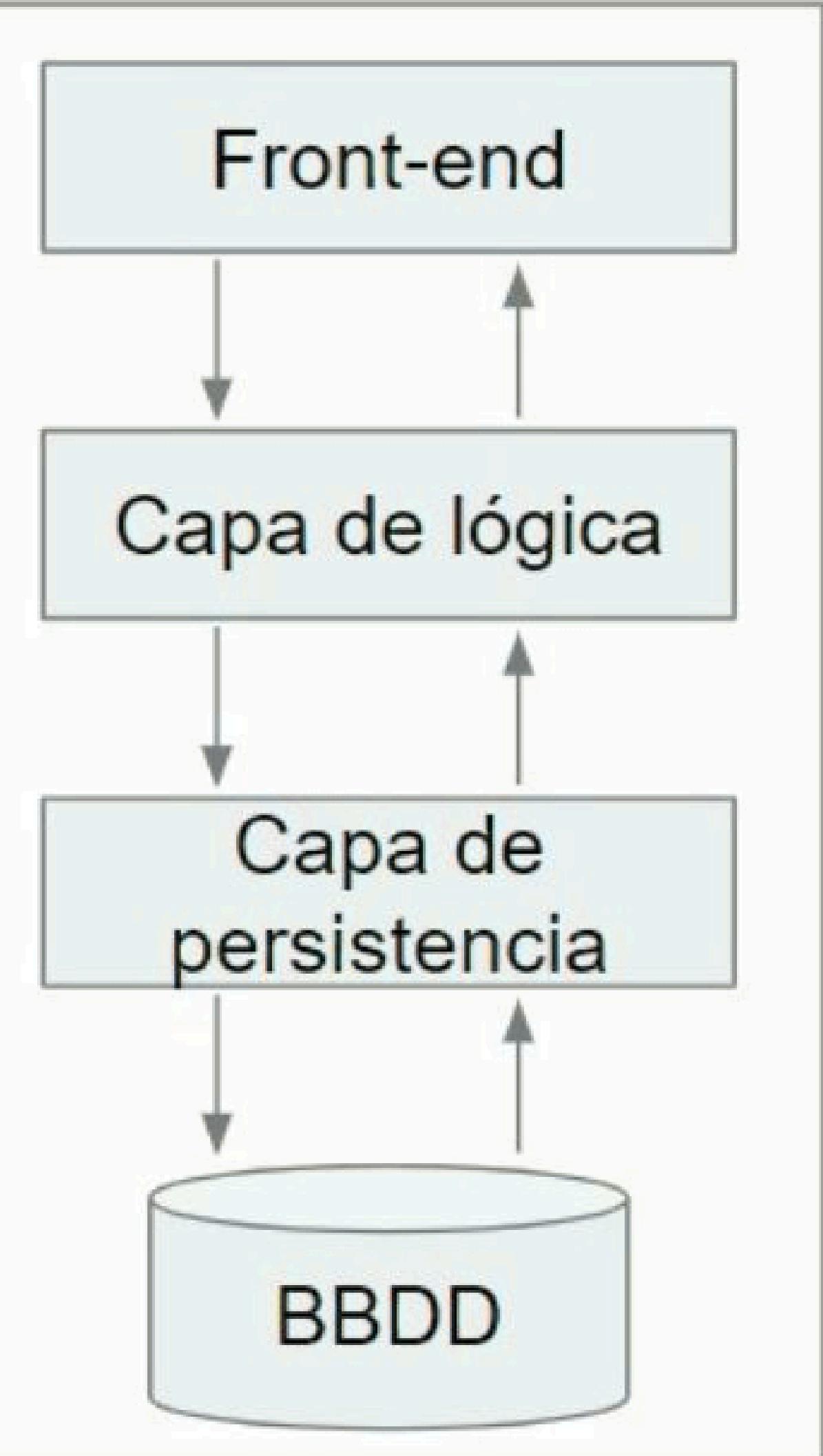
Definición

Una sola app que integra UI, lógica y datos en un mismo despliegue.

Presentación, negocio, persistencia en un solo artefacto.

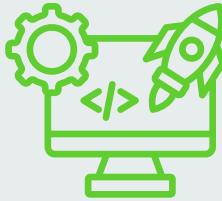
Historia

La arquitectura monolítica dominó desde los inicios de la informática por su simplicidad (**60's–2000**). Con el crecimiento de la complejidad de las aplicaciones, surgieron los microservicios en los 2000 como alternativa más escalable, aunque el enfoque monolítico sigue siendo útil en ciertos casos (**sistemas pequeños/medianos**).



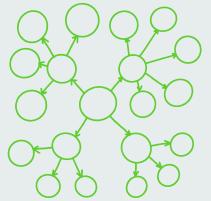
Características de la Arquitectura Monolítica

Despliegue único



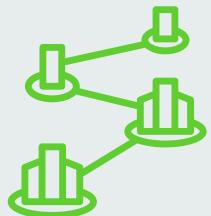
Toda la app se despliega como una sola unidad; fácil de administrar, pero con riesgo de tiempos de inactividad al actualizar.

Alta cohesión



Componentes fuertemente acoplados; facilita comunicación interna, pero dificulta los cambios aislados.

Escalabilidad limitada



Solo se puede escalar duplicando toda la app, incluso si solo una parte lo necesita.

Mantenimiento difícil



A mayor tamaño, más complejo el código; los cambios pueden afectar múltiples partes del sistema.



Ventajas

Desarrollo inicial rápido: Permite crear un producto funcional más rápido, ya que todo está contenido en una sola base de código sin necesidad de comunicación entre múltiples servicios.

Simplicidad: Su diseño y despliegue son menos complejos, ideal para proyectos pequeños o con recursos limitados. No requiere servicios de comunicación externos ni protocolos avanzados.

Facilidad en pruebas: Al ser una única unidad, las pruebas unitarias e integradas son más simples, ya que no hay dependencias externas ni servicios distribuidos.

Desventajas

Escalabilidad limitada: No permite escalar partes específicas del sistema; toda la aplicación debe replicarse, lo cual resulta ineficiente.

Dificultad para gestionar grandes sistemas: A medida que el sistema crece, el código puede volverse acoplado y difícil de mantener sin afectar otras partes del sistema.

Despliegue y mantenimiento complicados: Las actualizaciones deben realizarse sobre toda la aplicación, aumentando el riesgo de errores que afecten al sistema completo.

Casos de Uso (Cuándo aplicarla)



Aplicaciones pequeñas o medianas

Ideal para proyectos con una carga moderada y sin requerimientos altos de escalabilidad.



Sistemas heredados (legados)

Muchas aplicaciones tradicionales siguen siendo monolíticas por su estabilidad. Migrarlas a microservicios puede ser costoso o innecesario si cumplen bien su función.



Proyectos de inicio rápido

En startups o pruebas de concepto, permite desarrollar y desplegar un producto funcional en menos tiempo y con menor complejidad técnica.

Casos de Aplicación (Ejemplos reales)

Sistemas ERP tradicionales

Plataformas integradas como SAP ERP en su versión monolítica, donde todos los módulos (finanzas, ventas, inventarios) coexisten en una sola aplicación.

Aplicaciones bancarias

Sistemas internos de bancos que gestionan transacciones, cuentas y pagos dentro de una misma base de código, con bases de datos centralizadas y procesos interdependientes.



Jakarta EE + JSF

(JavaServer Faces)

Definición

- **Jakarta EE:** Conjunto de especificaciones para apps empresariales Java (ex Java EE).
- **JSF:** Framework de UI por componentes (Facelets .xhtml).

Historia

Java EE fue lanzado en 1999 por **Sun Microsystems** para estandarizar el desarrollo de aplicaciones empresariales. En 2017, **Oracle** transfirió el proyecto a la **Eclipse** Foundation, donde pasó a llamarse Jakarta EE, impulsando una evolución más abierta y comunitaria.



JAKARTA™ EE

Características de Jakarta EE y JavaServer Faces



Modularidad

Permite crear aplicaciones organizadas con componentes como **EJB**, **JPA**, **JMS** y **JSF**, facilitando el desarrollo modular y reutilizable.



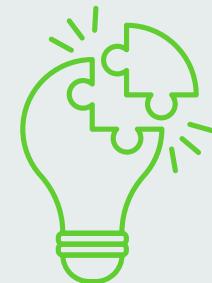
Integración

Se integra fácilmente con tecnologías empresariales (JPA para **persistencia**, JMS para **mensajería** y EJB para **lógica de negocio**) en un mismo entorno.



Desarrollo basado en componentes

JSF usa **componentes visuales reutilizables** para construir interfaces dinámicas y conectadas directamente con la lógica del backend.



Compatibilidad con servidores de aplicaciones

Funciona de forma nativa en **GlassFish** y **Payara**, que ofrecen servicios como seguridad, transacciones y escalabilidad.



Ventajas

Estándares abiertos: Basado en especificaciones oficiales, garantiza la interoperabilidad entre plataformas y servidores compatibles (GlassFish, Payara, WildFly, etc.).

Productividad del desarrollador: JSF ofrece componentes reutilizables (formularios, tablas, botones) que aceleran la creación de interfaces y facilitan la integración con JPA y EJB.

Escalabilidad: Diseñado para aplicaciones empresariales de gran escala, con gestión eficiente de transacciones, recursos y carga del sistema.

Desventajas

Curva de aprendizaje: Requiere dominar varias tecnologías (JPA, EJB, JMS, JSF), lo que puede resultar complejo para nuevos desarrolladores.

Rendimiento: Su naturaleza robusta puede generar sobrecarga en aplicaciones pequeñas o con requisitos de respuesta muy rápidos.

--



Casos de Uso (Dónde se aplica)

01

Aplicaciones empresariales

Ideal para sistemas grandes y complejos que requieren seguridad, escalabilidad y gestión de transacciones.

Ejemplos: sistemas de gestión empresarial, e-commerce y plataformas financieras.

02

Sistemas con múltiples capas

Perfecto para arquitecturas multicapa (presentación, negocio, datos), favoreciendo un desarrollo organizado y mantenable.

Ejemplos: sistemas ERP (Enterprise Resource Planning) y CRM (Customer Relationship Management).

Casos de Aplicación (Ejemplos reales)

Airbus

Utiliza Jakarta EE para sistemas de gestión interna y control de procesos.

Payara & GlassFish

Implementan las especificaciones de Jakarta EE, y son usados en proyectos empresariales de gran escala.



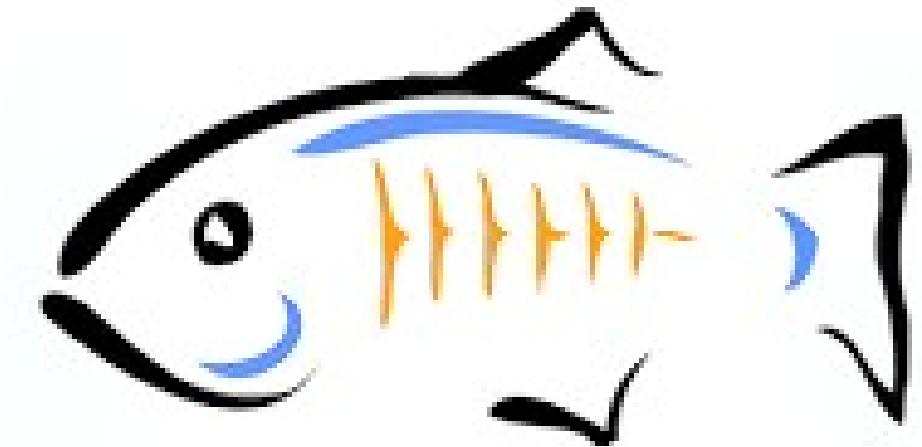
GlassFish y Payara

Definición

- **GlassFish:** Servidor de apps open-source que implementa Jakarta EE.
- **Payara:** Fork de GlassFish con foco en soporte comercial, estabilidad y parches.

Historia

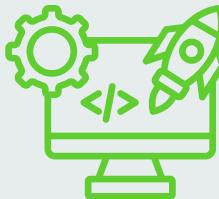
GlassFish nació en 2005 bajo **Sun Microsystems** como servidor de referencia para Java EE. Tras la adquisición por **Oracle** en 2010, el desarrollo continuó hasta que en 2015 fue adoptado por la comunidad de Payara, dando origen a **Payara Server**, una versión más robusta, segura y con soporte empresarial.



GlassFish



Características de GlassFish y Payara



Soporte Jakarta EE

Implementan todas las especificaciones de Jakarta EE, ideales para apps empresariales complejas.



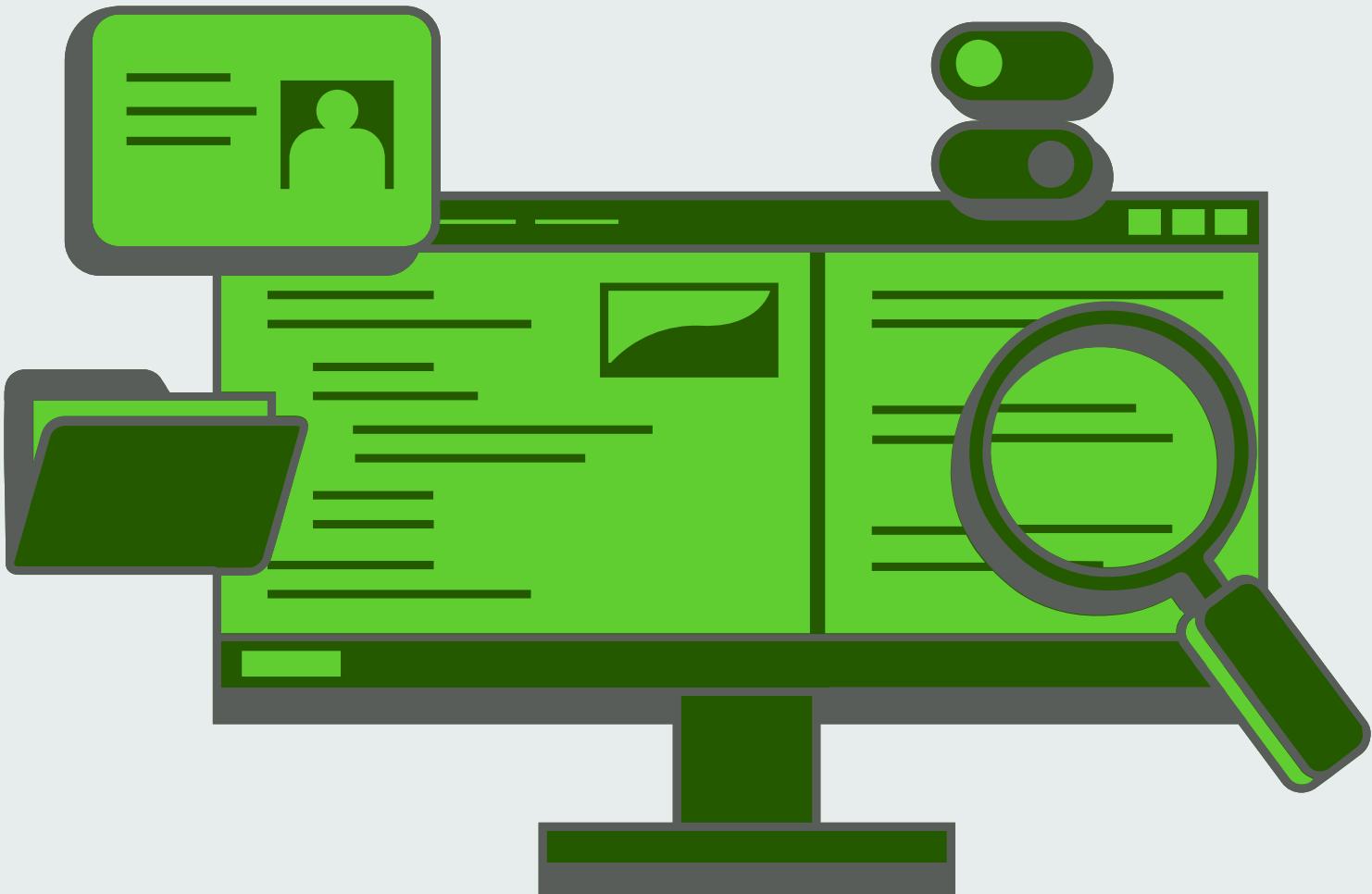
Escalabilidad

Soportan clústeres y manejan cargas altas, permitiendo crecer según las necesidades del negocio.



Comunidad activa

Payara tiene una comunidad fuerte y soporte empresarial, ideal para entornos críticos.



Ventajas

Código abierto: Ambos servidores son gratuitos y de código abierto, lo que permite su uso sin licencias costosas y brinda flexibilidad para personalizar el código según las necesidades del proyecto.

Soporte empresarial: Payara ofrece soporte profesional, actualizaciones frecuentes y parches de seguridad, ideal para organizaciones críticas (banca, telecomunicaciones, sector público).

Actualizaciones constantes: Payara se mantiene activo y actualizado, con mejoras continuas y nuevas características, superando en mantenimiento a GlassFish.

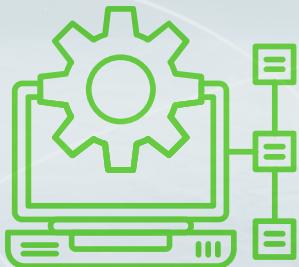
Desventajas

Complejidad: Requieren una configuración y mantenimiento cuidadoso, especialmente en entornos con clústeres, balanceo de carga o políticas de seguridad avanzadas.

Sobrecarga: Al implementar todas las especificaciones de Jakarta EE, pueden generar mayor consumo de recursos en aplicaciones pequeñas o de baja demanda.

--

Casos de Uso (Dónde se aplican)



Sistemas empresariales grandes

Ideales para aplicaciones de gran escala que exigen alta disponibilidad, seguridad y rendimiento.

Ejemplos: sistemas ERP, plataformas de banca electrónica y comercio electrónico con alto volumen transaccional.



Aplicaciones de misión crítica

Utilizados por **bancos, telecomunicaciones y servicios públicos**, donde no se permite el tiempo de inactividad.

Payara y GlassFish garantizan estabilidad, escalabilidad y tolerancia a fallos.

Casos de Aplicación (Ejemplos reales)

01

Telefónica

Usa Payara para gestionar servicios empresariales internos, gracias a su soporte extendido y capacidad para operar servicios críticos con alta disponibilidad.

02

Banco Santander

Implementa GlassFish en aplicaciones internas para gestionar procesos bancarios complejos.

La integración con Jakarta EE asegura transacciones seguras y escalabilidad en sus sistemas.



MariaDB

Definición

MariaDB es una base de datos relacional open source, compatible con MySQL y optimizada para alto rendimiento.

Historia

Nació en 2009 como fork de MySQL tras su compra por Oracle, y ha sido adoptada por grandes empresas por su fiabilidad y mejoras.



Características de MariaDB



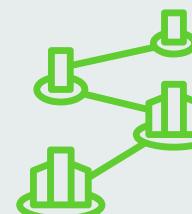
Compatible con MySQL

MariaDB mantiene compatibilidad total con MySQL, permitiendo migraciones sin necesidad de modificar el código o las estructuras existentes.



Rendimiento optimizado

Ofrece mejoras en la ejecución de consultas, índices y almacenamiento, con motores como Aria para manejar grandes volúmenes de datos de forma eficiente.



Escalable

Diseñada para crecer según las necesidades, soporta replicación y clustering, lo que permite su uso en entornos empresariales exigentes.



Alta disponibilidad

Incluye replicación maestro-esclavo, multimaster y clustering Galera, garantizando continuidad operativa ante fallos.

Ventajas

Código abierto: Totalmente gratuito y con una comunidad activa que mejora continuamente su rendimiento, seguridad y estabilidad. Ideal para empresas que buscan soluciones sin costos de licencia.

Alta disponibilidad: Soporta replicación y clustering, garantizando la continuidad del servicio incluso ante fallos del servidor.

Rendimiento mejorado: Optimizado para operaciones de lectura y escritura a gran escala, con motores de almacenamiento eficientes y alto desempeño transaccional.

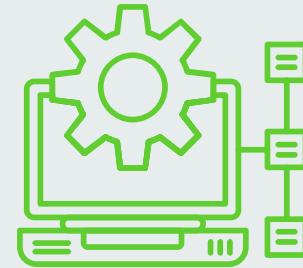
Desventajas

Requiere gestión: Su administración avanzada (clústeres, replicación, optimización) puede ser compleja en aplicaciones grandes o equipos sin experiencia en bases de datos.

Compatibilidad limitada con MySQL: Aunque muy similar, algunas características específicas de MySQL no se comportan igual, lo que puede generar dificultades en migraciones.

--

Casos de Uso (Dónde se aplican)



Aplicaciones web y móviles

Ideal para sitios web de **alto tráfico**, plataformas de e-commerce y apps móviles con grandes volúmenes de usuarios y datos.

Ofrece **alto rendimiento y baja latencia** en operaciones de lectura y escritura.



Sistemas empresariales

Ampliamente utilizado en sistemas ERP y CRM, donde se requiere una base de datos rápida, confiable y con alta disponibilidad para manejar transacciones complejas.

Casos de Aplicación (Ejemplos reales)

01

Wikipedia

Gestiona su enorme colección de artículos y metadatos con MariaDB, aprovechando su rendimiento optimizado en consultas y capacidad para manejar grandes volúmenes de datos.



02

Google

Implementa MariaDB en algunos servicios internos por su fiabilidad, compatibilidad con MySQL y escalabilidad en entornos distribuidos.



Relación entre los Temas Asignados

Qué tan común es el stack designado

Este stack sigue siendo **ampliamente usado en empresas tradicionales** que valoran estabilidad y tecnologías maduras.



SOLID

VS

Temas

Principio SOLID	Arquitectura Monolítica (Capas Monolito)	Jakarta EE + JSF	GlassFish/Payara	MariaDB
S: Single Responsibility	Cumple con la separación de responsabilidades entre capas.	JSF permite separar la lógica de presentación del backend.	Permite la separación de la lógica de servidor y cliente.	Gestiona exclusivamente la persistencia de datos.
O: Open/Closed	El monolito puede ser difícil de extender sin modificar el código existente.	Jakarta EE permite extensión mediante EJB, JPA, etc.	Payara facilita la adición de módulos sin modificar el núcleo.	Fácil extensión con nuevas tablas y procedimientos.
L: Liskov Substitution	Difícil de implementar debido al alto acoplamiento entre capas.	JSF facilita la implementación de componentes intercambiables.	Soporta implementación de servicios adicionales.	No aplica directamente, ya que es un sistema de bases de datos.
I: Interface Segregation	Puede tener interfaces complejas debido al enfoque monológico.	Jakarta EE permite la separación de interfaces y servicios.	Permite integrar distintas interfaces a través de módulos.	No aplicable directamente, es un sistema de gestión de bases de datos.
D: Dependency Inversion	La dependencia entre las capas puede ser difícil de manejar.	Jakarta EE facilita la inyección de dependencias a través de CDI.	GlassFish y Payara soportan la inyección de dependencias.	MariaDB gestiona dependencias de datos, pero no tiene un equivalente directo.

Atributos de Calidad vs Temas

Arquitectura	Monolítica (Capas Monolito)	Jakarta EE + JSF	GlassFish / Payara	MariaDB
Atributo de Calidad				
Compatibilidad	Limitada, ya que el sistema suele estar fuertemente acoplado. Difícil la integración con otros servicios o APIs externas.	Alta compatibilidad gracias a las especificaciones estándar de Jakarta EE. Permite interoperar con otros servicios y frameworks Java.	Totalmente compatibles con Jakarta EE, facilitando la interoperabilidad entre módulos y aplicaciones empresariales.	Compatible con MySQL y múltiples lenguajes de programación, lo que permite integrarse fácilmente en diferentes entornos.
Usabilidad	Depende de la interfaz implementada; no influye directamente en la arquitectura.	JSF mejora la usabilidad mediante componentes reutilizables y una interfaz web más consistente.	No aplica directamente; la usabilidad depende de la capa de presentación (JSF).	No aplica directamente; es una base de datos, su usabilidad depende de las herramientas de administración utilizadas.
Confiabilidad	Menor confiabilidad en sistemas grandes debido al riesgo de fallos globales. Un error afecta a toda la aplicación.	Proporciona confiabilidad al distribuir responsabilidades entre capas y manejar transacciones robustas.	Alta confiabilidad gracias al soporte de clustering, failover y monitoreo en Payara.	Elevada confiabilidad con replicación, clustering y recuperación ante fallos.
Seguridad	Difícil de mantener en sistemas grandes; requiere implementación manual de autenticación y control de acceso.	Jakarta EE incluye autenticación, autorización y manejo de sesiones seguras.	Ofrece características avanzadas de seguridad y cifrado configurable.	Proporciona cifrado de datos, autenticación robusta y control de acceso basado en roles.
Mantenibilidad	Baja mantenibilidad en aplicaciones extensas; el código tiende a acoplarse.	Alta mantenibilidad gracias a su modularidad (separación por capas).	Facilita el mantenimiento mediante herramientas de monitoreo y despliegue.	Mantenimiento sencillo en entornos pequeños, pero puede requerir experiencia técnica en clusters y replicación.
Portabilidad	Limitada, ya que depende del entorno del servidor monolítico.	Alta portabilidad dentro del ecosistema Java (corre en cualquier servidor compatible con Jakarta EE).	Portabilidad alta entre servidores Java compatibles (Payara, GlassFish, WildFly).	Portabilidad amplia entre sistemas operativos y entornos de desarrollo, compatible con MySQL.

Tácticas vs Temas

Táctica de Diseño	Arquitectura Monolítica (Capas Monolito)	Jakarta EE + JSF	GlassFish/Payara	MariaDB
Modularización	Limitada a las capas internas del monolito.	Jakarta EE promueve la modularización con EJB y JPA.	Payara soporta la modularización con microservicios.	MariaDB permite la modularización mediante bases de datos distribuidas.
Desacoplamiento	Alto acoplamiento entre capas y componentes.	JSF permite desacoplar la interfaz de usuario de la lógica de negocio.	Soporta desacoplamiento mediante inyección de dependencias.	No aplica directamente, es una base de datos.
Optimización	Requiere esfuerzos adicionales para optimizar el rendimiento.	Jakarta EE permite la optimización mediante configuración de recursos.	Optimización de procesos transaccionales.	Optimización en consultas y operaciones de base de datos.

Patrones vs Temas

Patrón de Diseño	Arquitectura Monolítica (Capas Monolito)	Jakarta EE + JSF	GlassFish/Payara	MariaDB
MVC (Modelo-Vista-Controlador)	Muy adecuado para organizar las capas.	JSF implementa MVC para separar lógica de presentación y negocio.	Facilita la implementación de MVC.	No aplica directamente.
Singleton	Usado en servicios globales dentro de la aplicación.	Jakarta EE soporta el patrón Singleton a través de EJB.	GlassFish y Payara soportan singleton para servicios globales.	No aplica directamente.
Factory	Se utiliza para crear instancias de objetos dentro del monolito.	Jakarta EE facilita la creación de objetos mediante inyección de dependencias.	Payara soporta patrones como Factory a través de CDI.	No aplica directamente.

Mercado Laboral

vs

Temas

Tecnología	Demanda en el Mercado Laboral
Arquitectura Monolítica	Aunque las arquitecturas de microservicios están en auge, la arquitectura monolítica sigue siendo común, especialmente en empresas que ya utilizan sistemas grandes y tradicionales.
Jakarta EE + JSF	Alta demanda en empresas que requieren aplicaciones empresariales robustas. A menudo se busca experiencia en Jakarta EE para el desarrollo de aplicaciones de gran escala.
GlassFish/Payara	Demandados en entornos empresariales que requieren alta disponibilidad y transacciones seguras. Payara, en particular, está siendo adoptado por organizaciones que buscan soporte extendido.
MariaDB	Creciente demanda debido a su popularidad en aplicaciones de bases de datos web. Empresas que usan MySQL están migrando hacia MariaDB por su estabilidad y características avanzadas.

Ejemplo Práctico

Sistema de Gestión de Reservas de Hotel

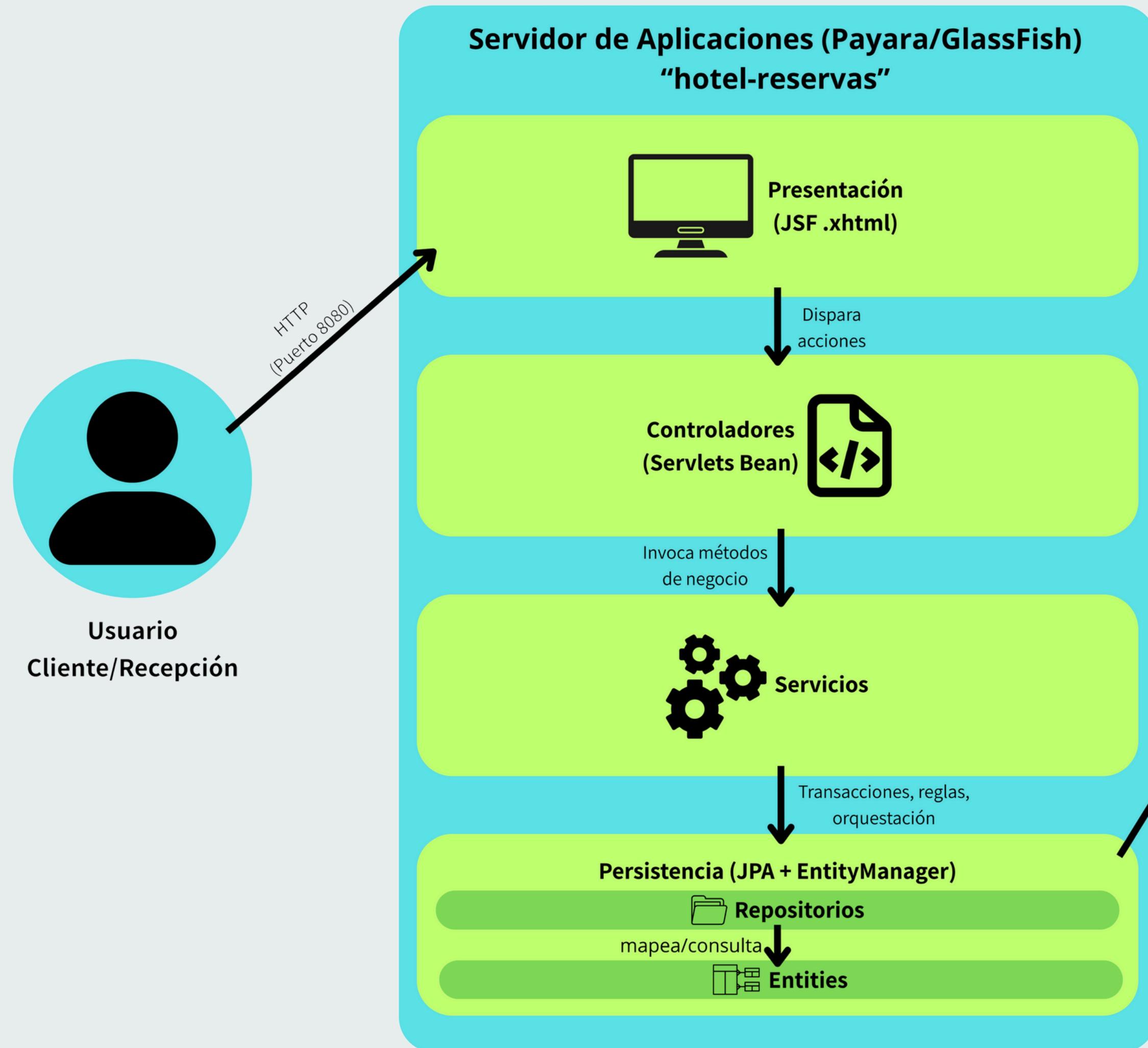
Entidades principales:

- **Cliente**: nombre, correo electrónico, teléfono.
- **Reserva**: fecha inicio, fecha fin, número de habitación, cliente.

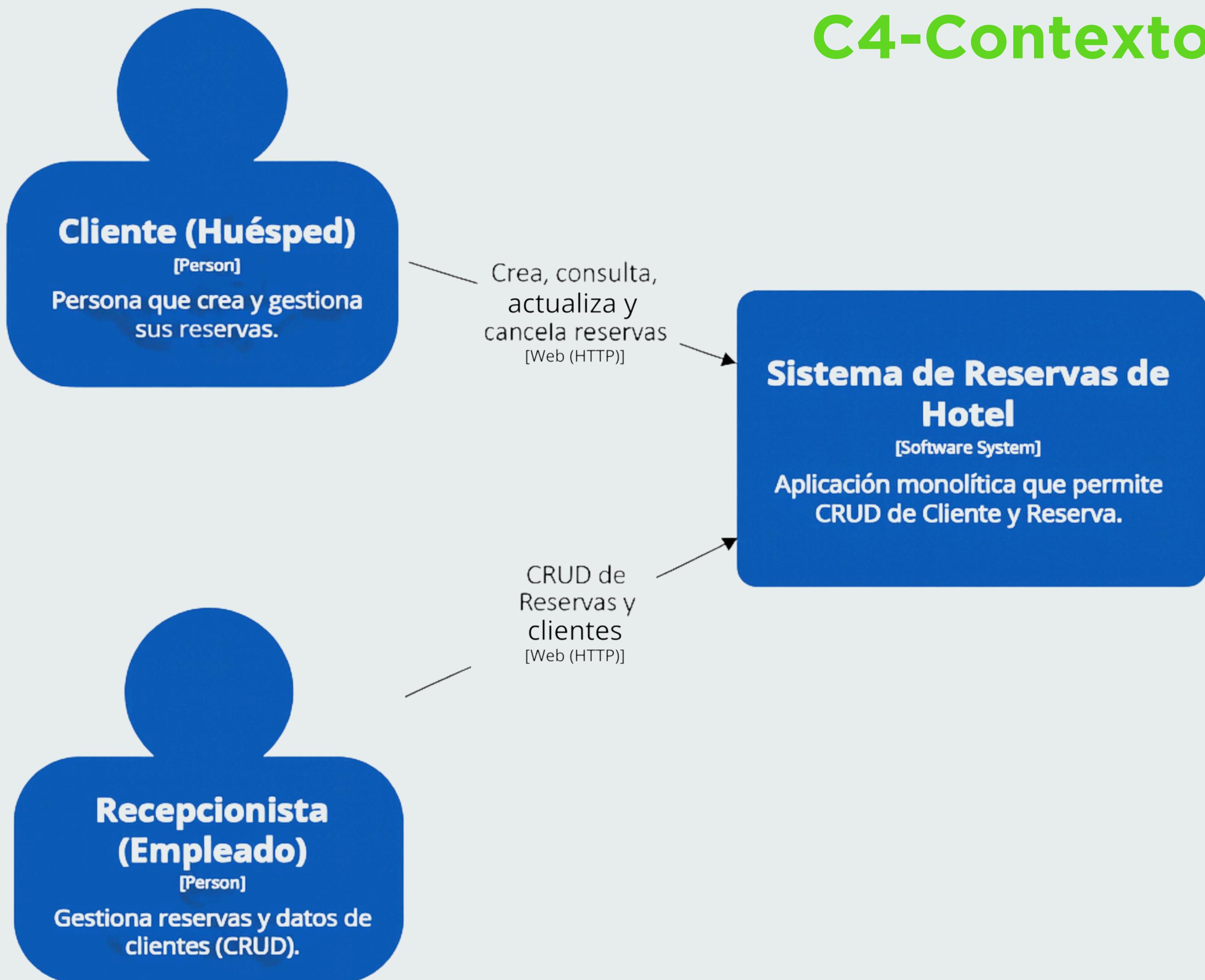
Operaciones CRUD:

- Crear, consultar, actualizar y eliminar reservas.
- Registrar y gestionar clientes.

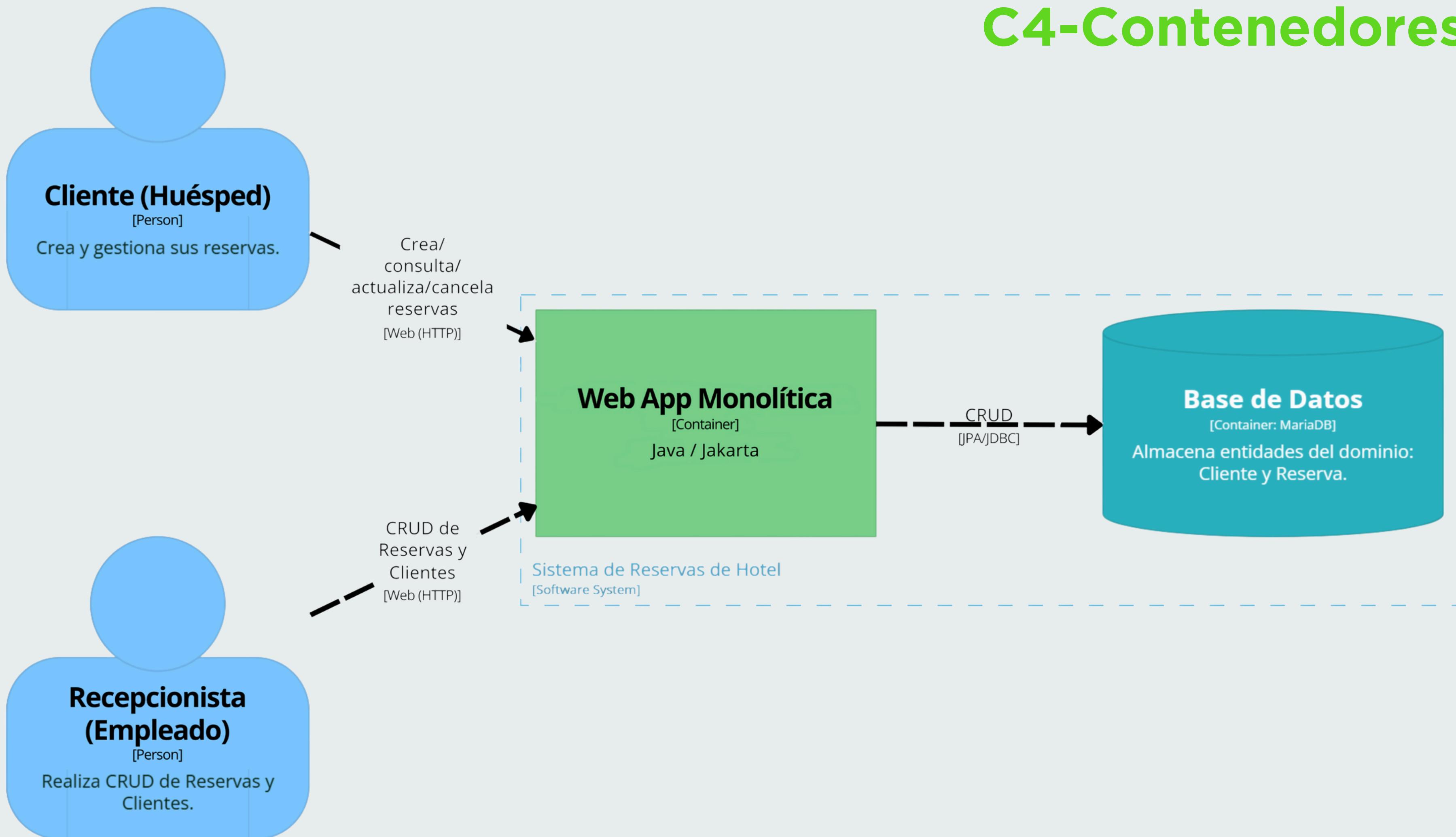
Alto nivel



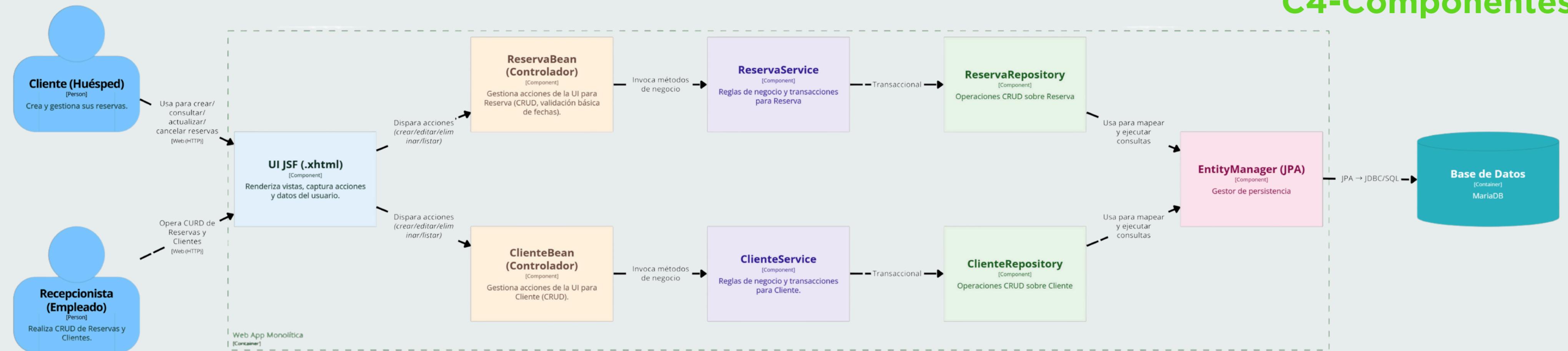
C4-Contexto



C4-Contenedores

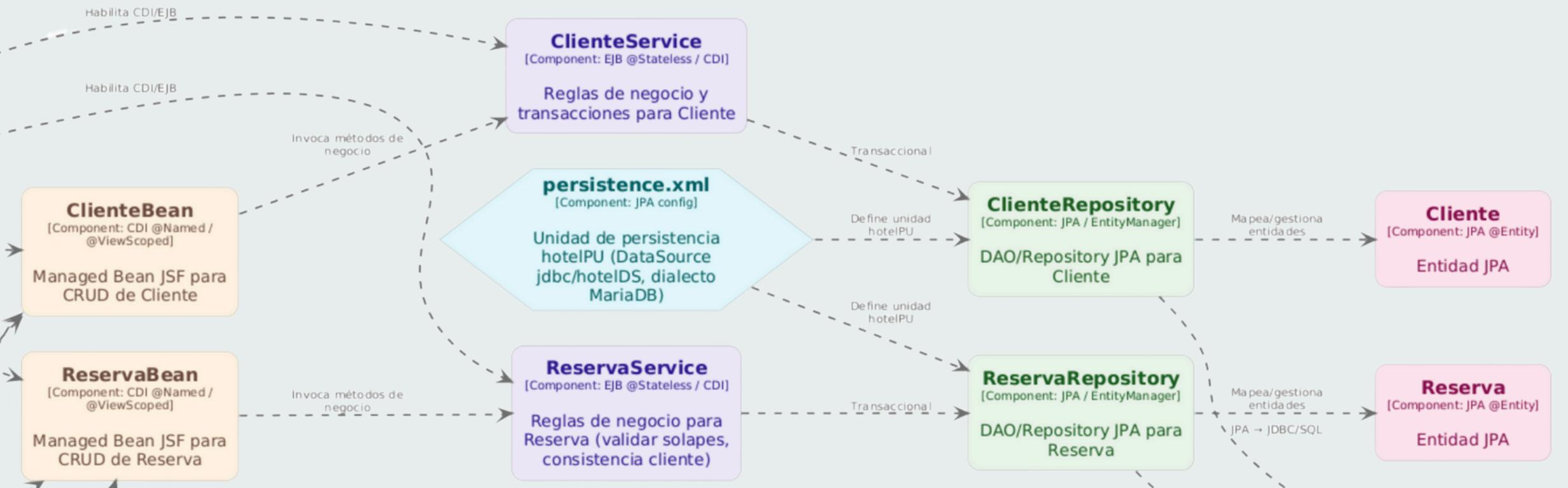
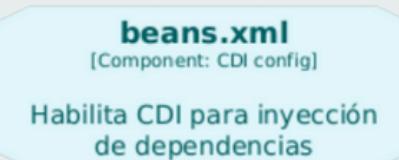
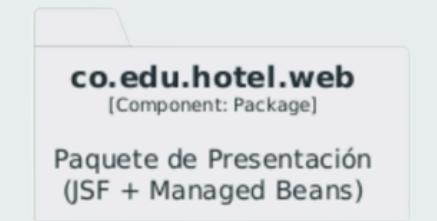
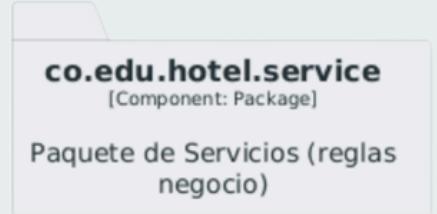
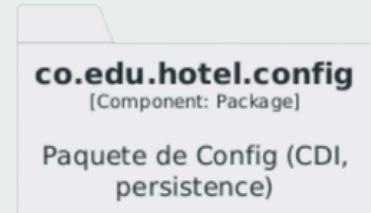


C4-Componentes



C4-Código

WebAppMonolitica
[Container: Java]



Cliente (Huésped)

[Person]

Crea una nueva reserva en la aplicación web.

1. 1. Ingresa datos [HTTP POST] 12. 12. Muestra mensaje [HTTP Response]

WebAppMonoplatica

[Container: Java]

UI JSF

[Component: JSF]

Vista donde el cliente ingresa datos de la reserva.

2. 2. Llama a crearReserva [Método Java] 11. 11. Actualiza vista [Mensaje]

ReservaBean

[Component: CDI]

Controlador JSF que gestiona la creación de reservas.

3. 3. Lógica de negocio [Método Java] 10. 10. Confirmación [Resultado]

ReservaService

[Component: EJB]

Aplica reglas de negocio y orquesta la transacción.

4. 4. Persiste reserva [Transacción JPA] 9. 9. Reserva creada [Reserva creada]

ReservaRepository

[Component: JPA]

Gestiona operaciones CRUD de la entidad Reserva.

5. 5. persist() [JPA] 8. 8. Entidad persistida [Reserva]

EntityManager

[Component: JPA API]

Ejecuta operaciones SQL contra la base de datos.

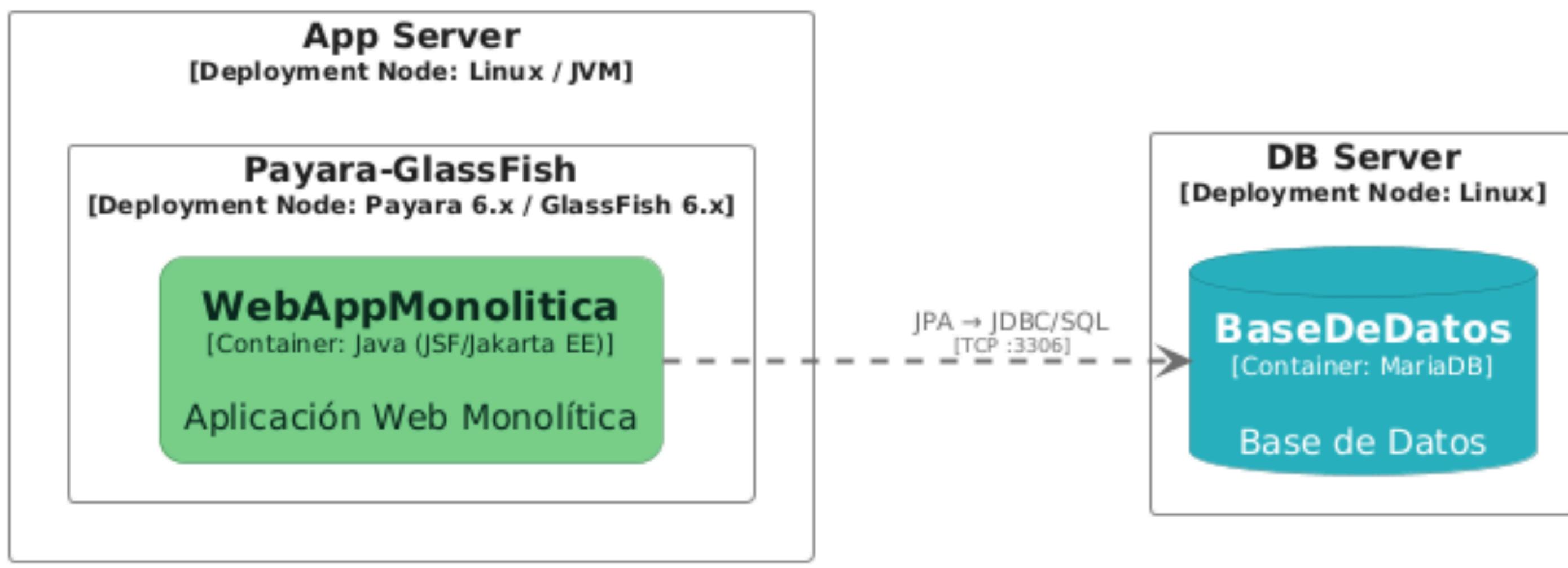
BaseDeDatos [Container: MariaDB]

Base de datos única con tablas Cliente y Reserva.

6. 6. INSERT SQL [SQL] 7. 7. Confirmación [Resultado]

Despliegue

Red de Producción VPC
[Deployment Node: Infraestructura]

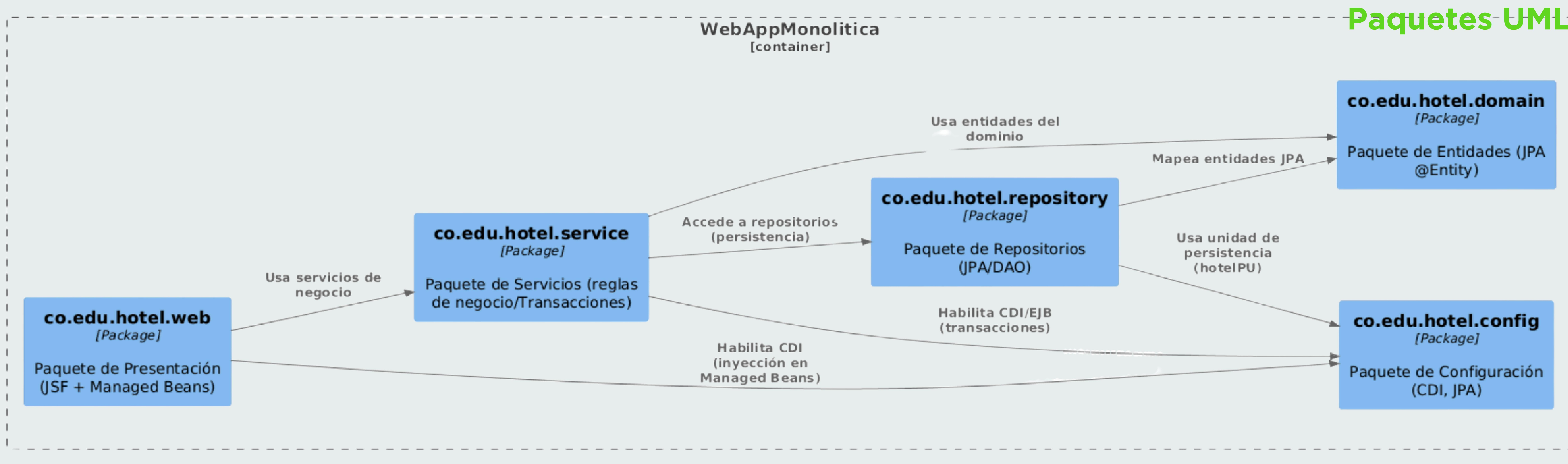


Cliente Navegador Web
[Deployment Node: Windows/macOS/Linux]

browser
[Infrastructure Node: Navegador
(Chrome/Firefox/Edge)]

Browser

Paquetes UML





Pontificia Universidad
JAVERIANA
Bogotá

Muchas Gracias

GRUPO 1



Arquitectura de Software

Thank
you! :)



Referencias

1. Bass, L., Clements, P., & Kazman, R. (2013). Software Architecture in Practice (3rd ed.). Addison-Wesley.
2. Garlan, D., & Shaw, M. (1994). An Introduction to Software Architecture. In Advances in Software Engineering and Knowledge Engineering (pp. 1-39). World Scientific Publishing.
3. Richards, M. (2015). Microservices vs Monolithic Architectures. O'Reilly Media.
4. Goncalves, A. (2020). Beginning Jakarta EE: Enterprise Edition for Java (2nd ed.). Apress.
5. Pau, L. (2016). JavaServer Faces (JSF) 2.0: The Complete Reference. McGraw-Hill Education.
6. Moris, R. (2018). Mastering Jakarta EE: Building Scalable and Secure Applications with Java. Packt Publishing.
7. Payara. (2021). What is Payara?. <https://www.payara.fish/>
8. Goncalves, A. (2020). Beginning Jakarta EE: Enterprise Edition for Java (2nd ed.). Apress.
9. Moris, R. (2018). Mastering Jakarta EE: Building Scalable and Secure Applications with Java. Packt Publishing.
10. Widenius, M. (2018). MariaDB: The Complete Reference. McGraw-Hill Education.
11. MariaDB Foundation. (2021). MariaDB Overview. <https://mariadb.org/>
12. Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.
13. Bass, L., Clements, P., & Kazman, R. (2013). Software Architecture in Practice (3rd ed.). Addison-Wesley.
14. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
15. Dahanayake, A., & Sol, H. (2012). The Importance of Java EE and Its Influence on Software Development Careers. Journal of Software Engineering.