Quienes Somos

Universidad Tecnológica del Perú

SESION 1
DISEÑO DE PATRONES

FUNDAMENTOS DE PENSAMIENTO DE DISEÑO

Para diseñar la arquitectura de un sistema de software, exploramos soluciones al mismo tiempo que trabajamos para descubrir el problema a resolver.

Design thinking

Pone a los seres humanos en el centro de la atención. Centrarse en las personas afectadas por sus decisiones de diseño lo ayuda a concentrarse en los problemas exactos que deben resolverse. También fundamenta la exploración de su solución recordándole que su propósito es crear software que ayude a las personas

Los cuatro principios del pensamiento de diseño

Como aplicar el pensamiento de diseño a la Arquitectura de Software ?.

1. El gobierno humano. Todo diseño es social por naturaleza.
2. Regla de la ambigüedad. Preservar la ambigüedad.
3. Regla de rediseño. Todo el diseño es rediseño.
4. Regla de tangibilidad. Hacer ideas tangibles para facilitar la comunicación.

Trabajamos con humanos

La regla humana también nos recuerda que los arquitectos no están separados de nuestros equipos. Trabajamos directamente con ellos para diseñar la arquitectura juntos. Construir software es una actividad intensamente social. Los arquitectos de software son una parte integral de cada equipo.

Al separar al arquitecto del equipo, se corta la conexión humana que el arquitecto comparte con todos los tocados por la arquitectura.

Preservamos la ambiguedad

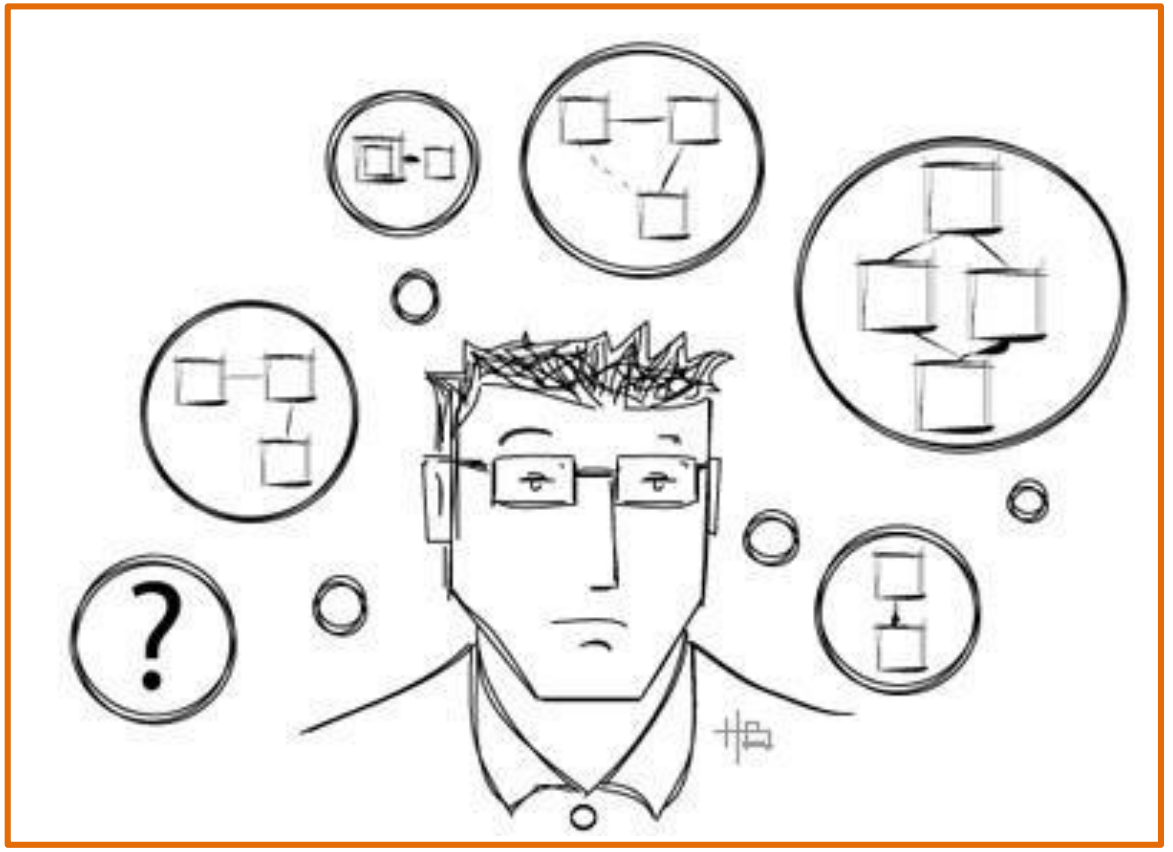
La ambigüedad en la ingeniería es peligrosa. Una vez que hayamos tomado una decisión de diseño, debemos compartirla con precisión y claridad. Permitir que los requisitos, las decisiones de diseño y los compromisos sigan siendo ambiguos es la mejor manera de destruir un proyecto. Antes de consolidar una decisión de diseño, podemos utilizar la ambigüedad para mantener las opciones abiertas.

Los arquitectos diseñan una arquitectura minimalista. Una arquitectura minimalista solo muestra cómo se logran los atributos de calidad de alta prioridad y reduce los riesgos para promover esos atributos de calidad. Todas las demás decisiones de diseño se dejan abiertas para que los diseñadores intermedios las determinen.

Diseño es rediseño

La regla de rediseño nos alienta a pensar en lo que ya sabemos al explorar patrones y diseños pasados. A medida que pasa el tiempo y a medida que construimos más software, mejora nuestro conocimiento institucional sobre cómo diseñar un gran software.

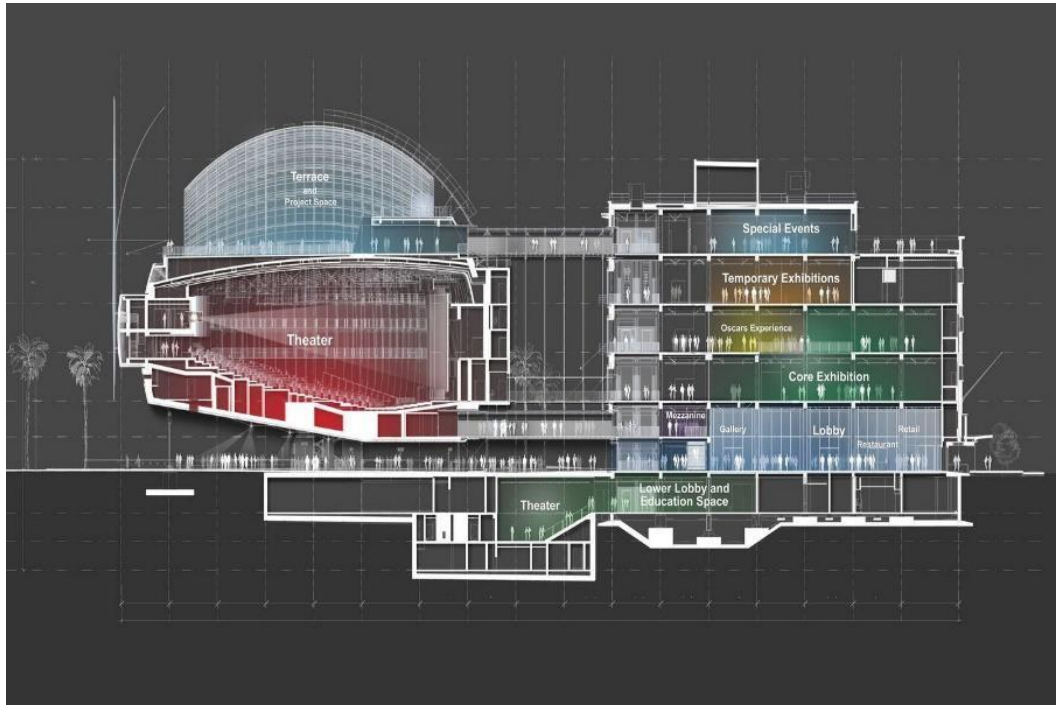
Al diseñar arquitecturas de software, dedicaremos más tiempo a refinar los diseños existentes que a los nuevos. Una de las formas menos efectivas de diseñar una arquitectura de software es ignorar los sistemas de software que nos presentaron.



Hacer la Arquitectura Tangible

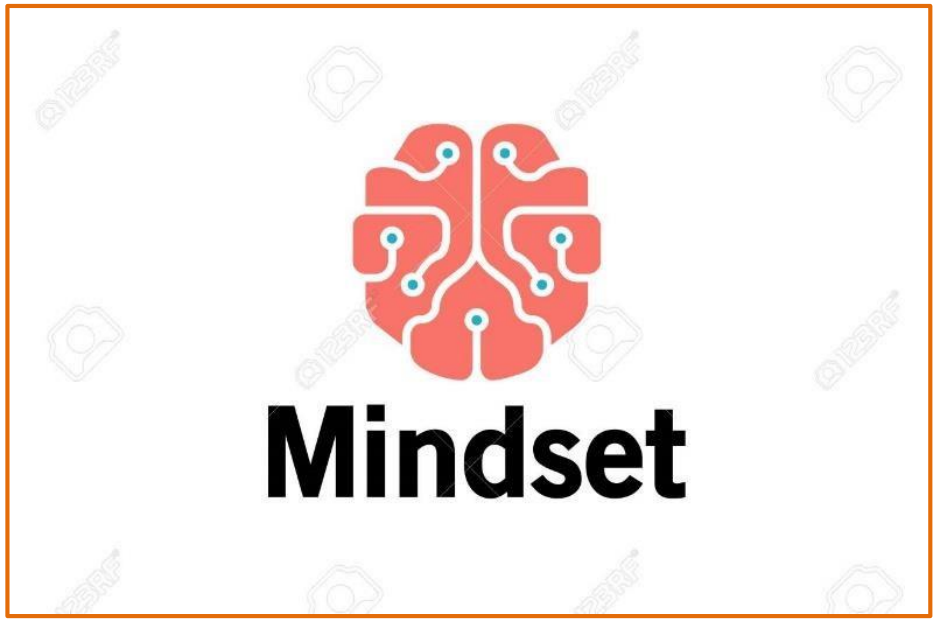
Si bien las estructuras en la arquitectura pueden existir en el código, esto no hace que la arquitectura sea más tangible. El código es difícil de leer y no lo hace

Dibújalo. Haz que cobre vida en el código que escribes. Construir prototipos que permitan a las personas experimentar estructuras y atributos de calidad. Cree modelos simples que muestren cómo funciona una parte de la arquitectura



Adoptar una mentalidad de diseño

El diseño de un sistema de software nos obliga a pensar en la arquitectura desde la perspectiva de diferentes mentalidades de diseño. Una mentalidad de diseño es una forma de pensar sobre el mundo, de modo que centramos nuestra atención en los detalles correctos en el momento adecuado



Explorar

Hacer

Entender

Evaluar

Adoptar una mentalidad de diseño

Para comprender el problema, debemos investigar:

- Los objetivos comerciales
- Los atributos de calidad que son importantes para nuestros accionistas.
- Aprender cómo funciona nuestro equipo
- Una idea más profunda de las prioridades y los compromisos entre las decisiones de diseño.

Hágalo Real: La forma más común en que hacemos que la arquitectura sea real es mediante la creación de modelos. La fabricación va más allá de los diagramas de caja y línea.
Construyendo prototipos, escribiendo documentos, haciendo cálculos de números y una variedad de otros enfoques.

CAVAR PARA REQUISITOS ARQUITECTONICAMENTE SIGNIFICATIVOS

Adoptar una mentalidad de diseño

Un requisito arquitectónico significativo, o ASR, es cualquier requisito que influye fuertemente en nuestra elección de estructuras para la arquitectura. Es responsabilidad del arquitecto de software identificar los requisitos con importancia arquitectónica.

- Restricciones Decisiones de diseño inmutables.
- Atributos de calidad Propiedades visibles externamente.
- Requisitos funcionales influyentes.
- Otros influyentes Tiempo

Limitar las opciones de diseño con restricciones

La mayoría de los sistemas de software tienen solo unas pocas restricciones:

Las restricciones comerciales limitan las decisiones sobre las personas, el proceso, los costos y el cronograma. Las restricciones técnicas limitan las decisiones sobre la tecnología que podemos usar en el sistema de software. Aquí hay algunos ejemplos de cada uno:

Technical Constraints	Business Constraints
Programming Language Choice	Team Composition and Makeup
Anything that runs on the JVM.	*Team X will build the XYZ component.*
Operating System or Platform	Schedule or Budget
It must run on Windows, Linux, and BeOS.	*It must be ready in time for the Big Trade Show and cost less than $800,000.*
Use of Components or Technology	Legal Restrictions
We own DB2 so that's your database.	*There is a 5 GB daily limit in our license.*

Capturar restricciones como declaraciones simples

Las restricciones, una vez decididas, son 100 por ciento no negociables. Sea conservador en aceptar restricciones. Hay una gran diferencia entre esto debe hacerse o usted fallará y esto debe hacerse a menos que tenga una buena razón para no hacerlo.

Constraint	Origin	Type	Context
Must be developed as open source software.	Mayor van Damme	Business	The City has an Open Data policy and citizens must have access to source code.
Must build a browser-based web application.	Mayor van Damme	Technical	Decreases concerns about software delivery and maintenance.
Must ship by the end of Q3.	Mayor van Damme	Business	Avoids end of fiscal year budget issues.
Must support latest Firefox web browser.	City IT	Technical	Officially supported browser.
Must be served from a Linux server.	City IT	Technical	City uses Linux and open source where possible.

Definir atributos de calidad

Los atributos de calidad describen las propiedades visibles externamente de un sistema de software y las expectativas para el funcionamiento de ese sistema. Los atributos de calidad definen qué tan bien un sistema debe realizar alguna acción.

Design Time Properties	Runtime Properties	Conceptual Properties
Modifiability	Availability	Manageability
Maintainability	Reliability	Supportability
Reusability	Performance	Simplicity
Testability	Scalability	Teachability
Buildability or Time-to-Market	Security	

Cada decisión de arquitectura promueve o inhibe al menos un atributo de calidad

Definir atributos de calidad

Los atributos de calidad describen las propiedades visibles externamente de un sistema de software y las expectativas para el funcionamiento de ese sistema. Los atributos de calidad definen qué tan bien un sistema debe realizar alguna acción.

¿Son los atributos de calidad requisitos no funcionales?

Cuando diseña una arquitectura de software, es útil distinguir entre **funcionalidad, restricciones y atributos de calidad** porque cada tipo de requisito implica que un conjunto diferente de fuerzas influye en el diseño. Por ejemplo, las restricciones no son negociables, mientras que los atributos de calidad pueden ser matizados e implicar concesiones significativas.

Capturamos atributos de calidad

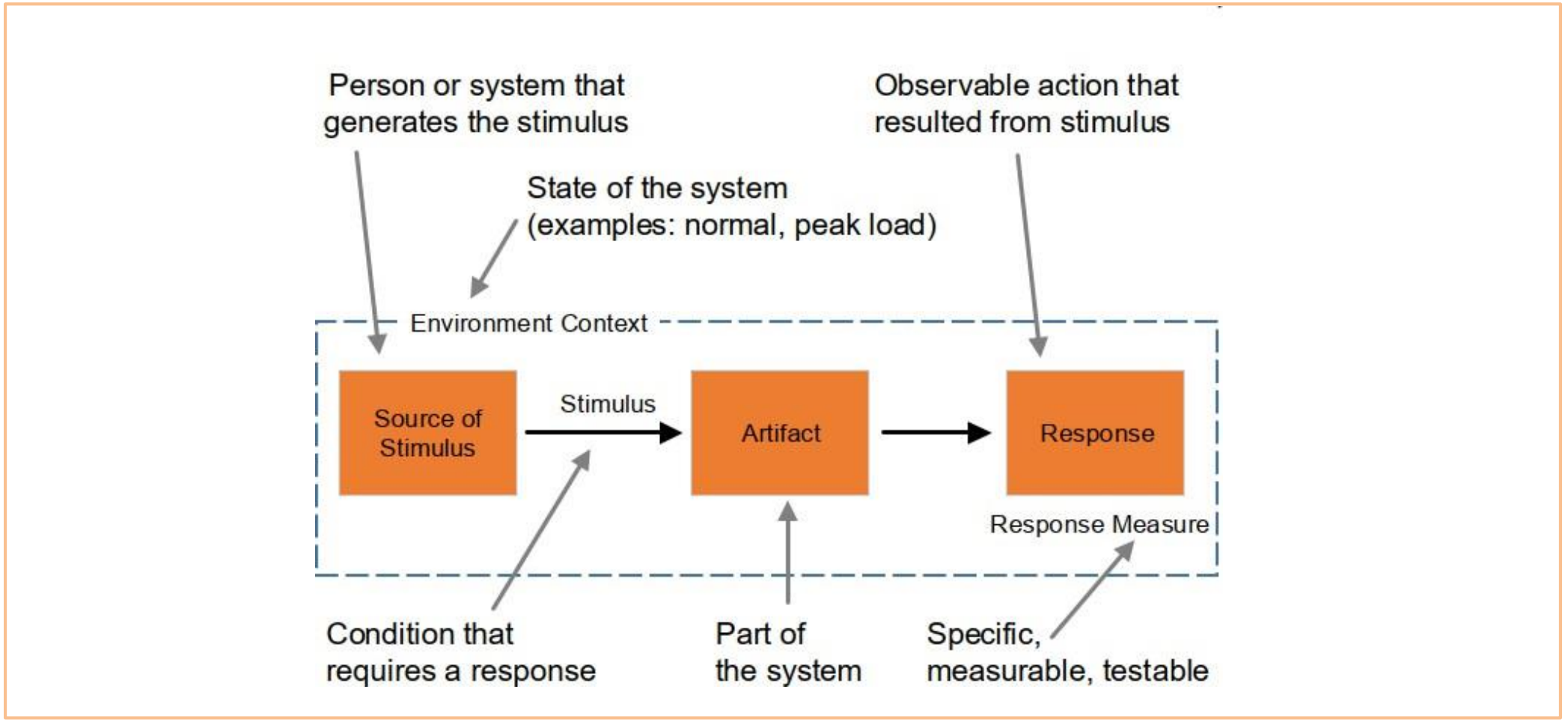
Un atributo de calidad es solo una palabra. La escalabilidad.

Necesitamos dar significado a estas palabras para entenderlas

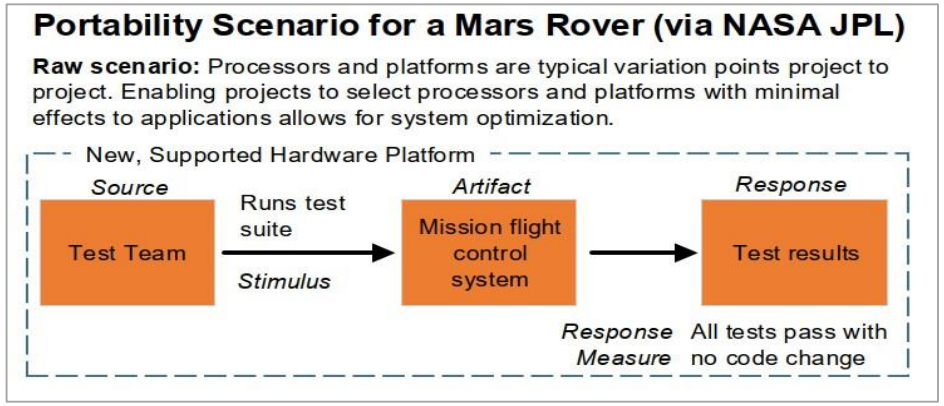
Escenario del Atributo de Calidad

describen cómo se espera que el sistema de software funcione dentro de un contexto ambiental determinado. Hay un componente funcional para cada escenario, estímulo y respuesta, al igual que cualquier característica. Los escenarios de atributos de calidad difieren de los requisitos funcionales, ya que califican la respuesta utilizando una medida de respuesta

Capturamos atributos de calidad



Ejemplos de escenarios de atributos de calidad



Un escenario comunica la intención del requisito para que cualquiera pueda entenderlo. Los grandes escenarios son precisos y medibles. Dos personas que leen el mismo escenario de atributo de calidad deberían comprender la escalabilidad o el rendimiento o la capacidad de mantenimiento del sistema.

Here are some quality attribute scenarios for the Project Lionheart case study:

Quality Attribute	Scenario	Priority
Availability	When the RFP database does not respond, Lionheart should log the fault and respond with stale data within 3 seconds.	High
Availability	A user's searches for open RFPs and receives a list of RFPs 99% of the time on average over the course of the year.	High
Scalability	New servers can be added during a planned maintenance window (less than 7 hours).	Low
Performance	A user sees search results within 5 seconds when the system is at an average load of 2 searches per second.	High
Reliability	Updates to RFPs should be reflected in the application within 24 hours of the change.	Low
Availability	A user-initiated update (for example, starring an RFP) is reflected in the system within 5 seconds.	Low
Availability	The system can handle a peak load of 100 searches per second with no more than a 10% dip in average response times.	Low
Scalability	Data growth is expected to expand at a rate of 5% annually. The system should be able to grow to handle this with *minimal* effort.	Low

Ensucia tus manos:

Refina estas notas en escenarios de atributos de calidad.

Durante una reunión, las partes interesadas del Proyecto Lionheart compartieron los siguientes enunciados. Para cada declaración, identifique el atributo de calidad y cree un escenario formal de atributo de calidad de seis partes.

- Hay una pequeña cantidad de usuarios, pero cuando un usuario envía una pregunta o un problema, debemos poder responder rápidamente, dentro de un día hábil.
- Los lanzamientos ocurren al menos una vez al mes. Lo ideal es que enviemos el código cuando esté listo.
- Necesitamos verificar que el índice RFP está construido correctamente. La verificación debe ser automatizada.
- Necesitamos un nuevo equipo de desarrollo permanente para que se acelere rápidamente después de que el actual equipo de contratistas que contratamos se haya ido.

Ensucia tus manos:

Refina estas notas en escenarios de atributos de calidad.

Aquí hay algunas cosas para pensar:

- ¿Qué atributo de calidad sugiere cada declaración? Está bien inventar si es útil para describir la preocupación de manera efectiva.
- ¿Hay respuestas implícitas o medidas de respuesta?
- ¿Qué información faltante puede completar en función de sus propias experiencias?

Busque Clases de Requisitos Funcionales:

Requisitos funcionales influyentes

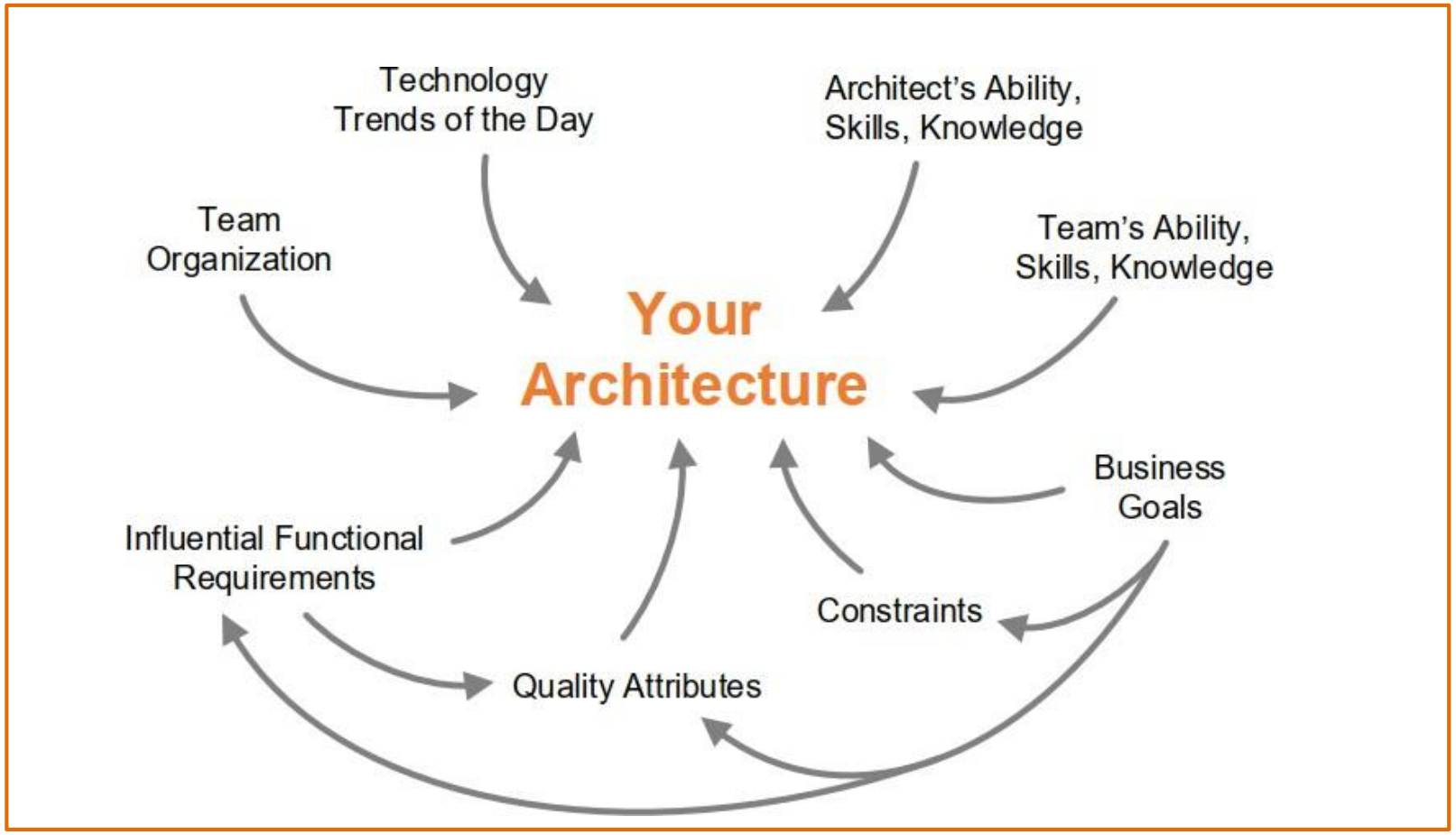
Cuando un requisito funcional impulsa la toma de decisiones arquitectónicas, lo llamamos un requisito funcional influyente. Los requisitos funcionales influyentes pueden denominarse destructores de arquitectura

1. Comience con un boceto de arquitectura nocional que resuma su pensamiento actual sobre la arquitectura.
2. Identifique las clases generales de requisitos que representan el mismo tipo de problema arquitectónico.
3. Para cada clase de problema identificada, recorra la arquitectura nocional y muestre cómo lograr cada grupo de requisitos.

Si no es inmediatamente obvio cómo implementaría la característica basada en los requisitos de grano grueso conocidos, podría tener un significado arquitectónico.

Averigüe qué más influye en la arquitectura

Además de los ASR, hay una serie de otros factores que afectarán la arquitectura tanto directa como indirectamente.



Aprende a vivir con la ley de Conway

La forma en que su equipo está organizado y prefiere colaborar influye en el diseño de la arquitectura.

Si tienes tres equipos, terminarás con tres componentes. Los límites de comunicación entre las personas se manifiestan como elementos límites en la arquitectura. La ley de Conway funciona en ambos sentidos. Las vías de comunicación diseñadas en la arquitectura también influirán en la forma en que organiza sus equipos.

Si desea diseñar el mejor software posible, debe estar preparado para reorganizar su equipo. Otros influyentes generalmente solo se registran como parte de la justificación de las decisiones de diseño. Hay tantas cosas que pueden influir en la arquitectura que es prácticamente imposible documentar todos los influencers potenciales antes de tomar decisiones de diseño.

Cava por la información que necesitas

Donde encuentro la información para iniciar mi Arqitectura ?

Encontrará ASR en las historias de usuario,

- Implícitas en la solicitud de un administrador, e insinuadas por partes interesadas que saben lo que quieren pero que no saben muy bien cómo explicarlo.
- La cartera de productos contiene un tesoro de ASR.
- Los atributos de calidad están implícitos o se asumen en casi todos los requisitos funcionales.
- A veces, una historia de usuario describirá claramente los tiempos de respuesta, las necesidades de escalabilidad o cómo manejar las fallas.
- Resalte estos detalles como escenarios de atributos de calidad para que no se pierdan en el registro de características.
- Hable con las partes interesadas. Averigua qué les preocupa. Pregunte a los interesados qué les emociona. Comparte los riesgos y abre las preguntas que veas. Aquí hay algunos métodos adicionales que puede usar para extraer ASR interesantes:.

Construya un ASR Workbook

El libro de trabajo de ASR es un documento vivo y cambia rápidamente. El libro de trabajo de ASR proporciona contexto e información para programadores, evaluadores y, por supuesto, arquitectos

Sample ASR Workbook Outline

- Purpose and Scope
- Intended Audience
- Business Context
  - Stakeholders
  - Business Goals
- Architecturally Significant Requirements
  - Technical Constraints
  - Business Constraints
  - Quality Attribute Requirements
    - Top Scenarios
  - Influential Functional Requirements
    - Top Users or User Personas
    - Use Cases or User Stories
- Appendix A: Glossary
- Appendix B: Quality Attributes Taxonomy

CAVAR PARA REQUISITOS ARQUITECTONICAMENTE SIGNIFICATIVOS

Patrones de Software

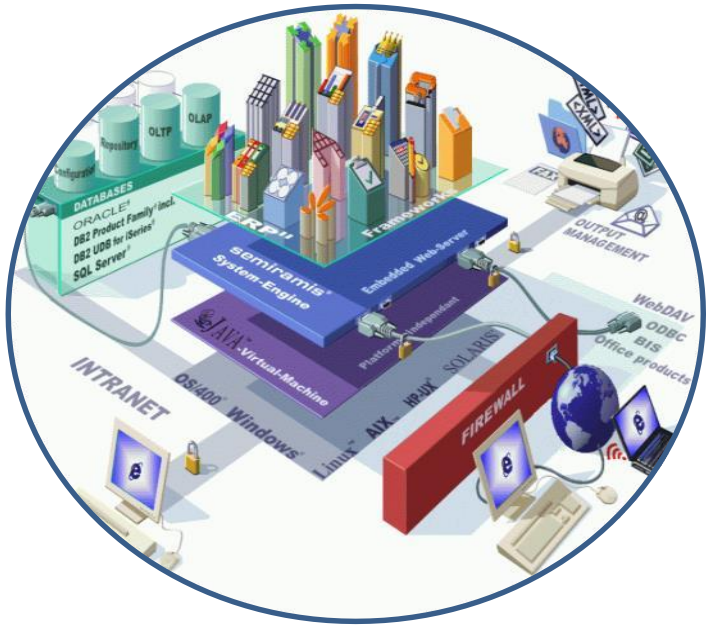
Por que preocuparnos en los patrones de software

La complejidad de las soluciones de software recientes aumenta continuamente debido a la continua evolución de las expectativas comerciales, con software complejo, no solo la actividad de desarrollo del software se vuelve muy difícil, sino que también las tareas de mantenimiento y mejora del software se vuelven tediosas y requieren mucho tiempo.

Los patrones de software son un factor tranquilizador para los arquitectos, desarrolladores y operadores de software. El campo de la arquitectura de software ayuda a suavizar y enderezar el camino hacia la producción de paquetes de software bien definidos y diseñados.

La Arquitectura – Aspectos cruciales

La arquitectura es esencial para los sistemas que se están volviendo cada vez más complejos debido a la adición continua de módulos nuevos.



Componentes participantes

Capacidades de cada componente

Conectividad entre componente

Se debe tener en cuenta una variedad de factores al decidir la arquitectura. Una serie de decisiones arquitectónicas deben considerarse meticulosamente para fortalecer la arquitectura final.
No solo los requisitos funcionales, sino también los requisitos no funcionales también deben inscribirse en la arquitectura.
Normalmente, el patrón de arquitectura es para diseñar una arquitectura genérica para un sistema, como una solución de software..

La Arquitectura – Trabajo con Patrones

Como describir o trabajar con los Patrones ?

Nombre: Una forma significativa y memorable de referirse al patrón, generalmente una sola palabra o frase corta.
Problema: Esta es una descripción concisa del problema en cuestión. Tiene que arrojar algo de luz sobre las metas y objetivos que se deben alcanzar dentro del contexto.
Contexto: el contexto típicamente ilustra las condiciones previas bajo las cuales se puede aplicar el patrón. Es decir, es una descripción del estado inicial antes de que se aplique el patrón.
Fuerzas: Esto es para describir las fuerzas y restricciones relevantes y cómo interactúan / entran en conflicto entre sí..

La Arquitectura – Estructura de Patrones

Solución: todo esto se trata de explicar claramente cómo alcanzar las metas y objetivos previstos. La descripción debe identificar tanto la estructura estática de la solución como su comportamiento dinámico.

Contexto resultante: Esto indica las condiciones posteriores a la aplicación del patrón. La implementación de la solución normalmente requiere compensaciones entre las fuerzas en competencia. Este elemento describe qué fuerzas se han resuelto y cómo, y cuáles siguen sin resolverse. También puede indicar otros patrones que pueden ser aplicables en el nuevo contexto.

Ejemplos: se trata de incorporar algunas aplicaciones de muestra del patrón para ilustrar cada uno de los elementos (un problema específico, el contexto, el conjunto de fuerzas, cómo se aplica el patrón y el contexto resultante).

Justificación: Es necesario dar una explicación / justificación convincente del patrón en su conjunto o de los componentes individuales dentro de él. La razón tiene que indicar cómo funciona realmente el patrón y cómo resuelve las fuerzas para lograr los objetivos deseados y

La Arquitectura – Patrones relacionados

Patrones relacionados: hay otros patrones similares y, por lo tanto, las relaciones entre este patrón y otros deben estar claramente articuladas. Estos pueden ser patrones predecesores, cuyos contextos resultantes corresponden al contexto inicial de éste. O, estos pueden ser patrones sucesores, cuyos contextos iniciales corresponden al contexto resultante de éste. También puede haber patrones alternativos, que describen una solución diferente para el mismo problema, pero bajo diferentes fuerzas. Finalmente, estos pueden ser patrones co-dependientes, que pueden / deben aplicarse junto con este patrón.

Usos conocidos: Esto tiene que detallar las aplicaciones conocidas del patrón dentro de los sistemas existentes. Esto es para verificar que el patrón sí describe una solución comprobada para un problema recurrente. Los usos conocidos también pueden servir como ejemplos de valor agregado.

Tipos de Patrones de Software

Patrones de Arquitectura: El patrón de arquitectura para un sistema de software ilustra la estructura de nivel macro para toda la solución de software, mejora la partición y promueve la reutilización del diseño.

Patrones de Diseño: Proporciona un esquema para refinar los subsistemas o componentes de un sistema, o las relaciones entre ellos, describe una estructura comúnmente recurrente de componentes de comunicación que resuelve un problema de diseño general dentro de un contexto particular, articula cómo los diversos componentes dentro del sistema colaboran entre sí para cumplir con la funcionalidad deseada.

Aplicaciones Monolíticas

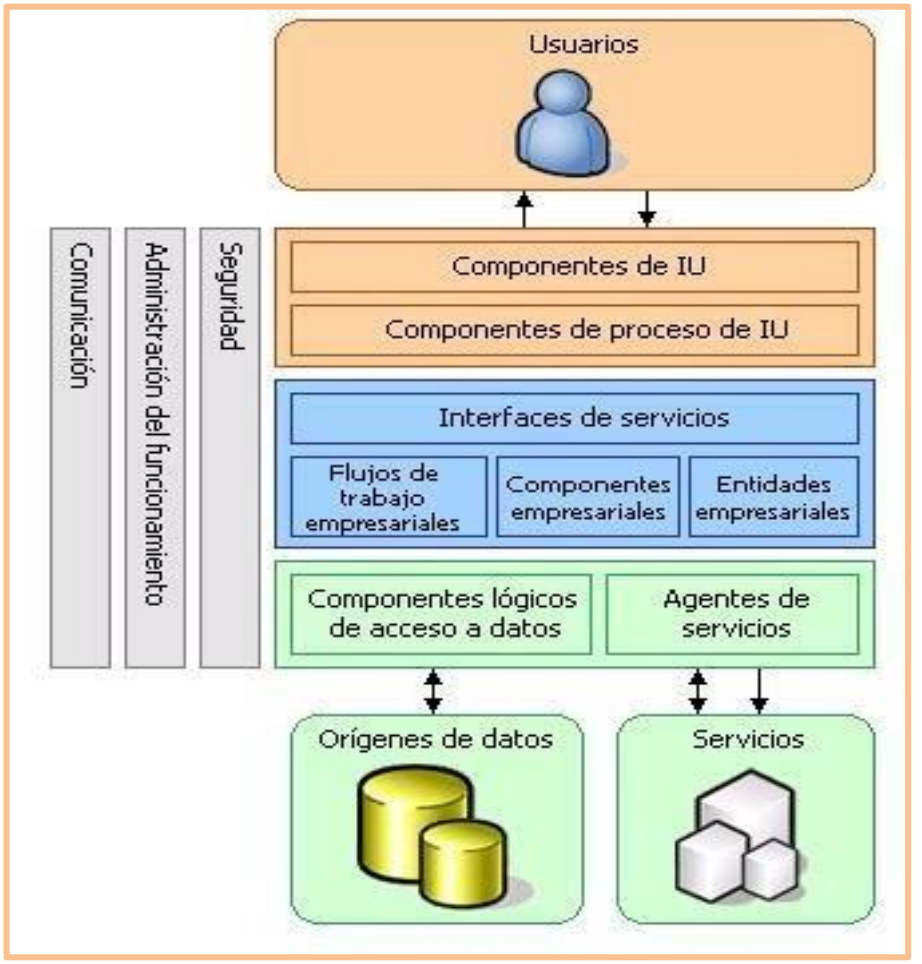
En ingeniería de software una **aplicación monolítica** hace referencia a una aplicación software en la que la capa de interfaz de usuario y la capa de acceso a datos están combinadas en un mismo programa y sobre una misma plataforma.

Una aplicación monolítica es autónoma, e independiente de otras aplicaciones. La filosofía de diseño es que la aplicación es responsable no sólo de una única tarea, si no que es capaz de realizar todos los pasos o tareas necesarias para completar una determinada función.

En ingeniería de software, **una aplicación monolítica describe una aplicación de software qué está diseñado sin modularidad**. En general, la modularidad es deseable, ya que permite reutilizar partes de lógica de una aplicación y además facilita el mantenimiento permitiendo refactorizar o sustituir partes de la misma sin necesidad de cambiar todo.

Aplicaciones Monolíticas

Diseña la aplicación monolítica de tu organización



Aspectos de discusión

Cuales son las desventajas de las Aplicaciones Monolíticas?

Cual es su experiencia con aplicaciones monolíticas?

Por que migraría a otro tipo de Arquitectura?

Arquitectura orientada a objetos

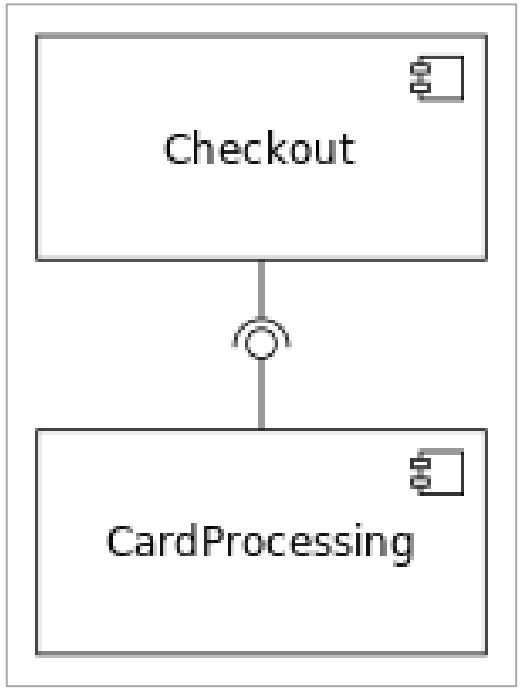
Los objetos son los bloques de construcción fundamentales y fundamentales para todo tipo de aplicaciones de software.

La estructura y el comportamiento de cualquier aplicación de software se pueden representar mediante el uso de objetos múltiples e interoperables.
Los objetos encapsulan elegantemente las diversas propiedades y las tareas de una manera optimizada y organizada.

Los objetos se conectan, se comunican y colaboran a través de interfaces bien definidas. Por lo tanto, el estilo arquitectónico orientado a objetos se ha convertido en el dominante para la producción de aplicaciones de software orientadas a objetos.

En última instancia, un sistema de software se ve como una colección dinámica de objetos que cooperan, en lugar de un conjunto de rutinas o instrucciones de procedimiento.

Arquitectura de ensamblaje basado en componentes (CDB)



Las aplicaciones monolíticas y masivas se pueden **dividir en múltiples componentes** interactivos y más pequeños.
Los componentes emergen como el bloque de construcción para diseñar y desarrollar aplicaciones de escala empresarial.
Los componentes **exponen interfaces bien definidas** para que otros componentes puedan encontrar y comunicarse. Esta configuración proporciona un mayor nivel de abstracción que los principios de diseño orientados a objetos.
CBA no se centra en temas como los protocolos de comunicación y los estados compartidos. **Los componentes son reutilizables, reemplazables, sustituibles, extensibles, independientes, etc.**

Uso del patrón Inyección de Dependencias

(CDB) Programación orientada a aspectos AOP

El estilo AOP apunta a aumentar la modularidad permitiendo la separación de las preocupaciones de corte transversal. **AOP incluye métodos de programación y herramientas que admiten la modularizarían** de inquietudes en el nivel del código fuente. Programación orientada a aspectos que descompone la lógica del programa en partes distintas (preocupaciones, las áreas cohesivas de funcionalidad).

En los lenguajes orientados a objetos, la estructura del sistema se basa en la idea de *clases* y *jerarquías de clases*. La herencia permite modularizar el sistema, eliminando la necesidad de duplicar código. No obstante, siempre hay aspectos que son transversales a esta estructura

(CDB) Programación orientada a aspectos AOP

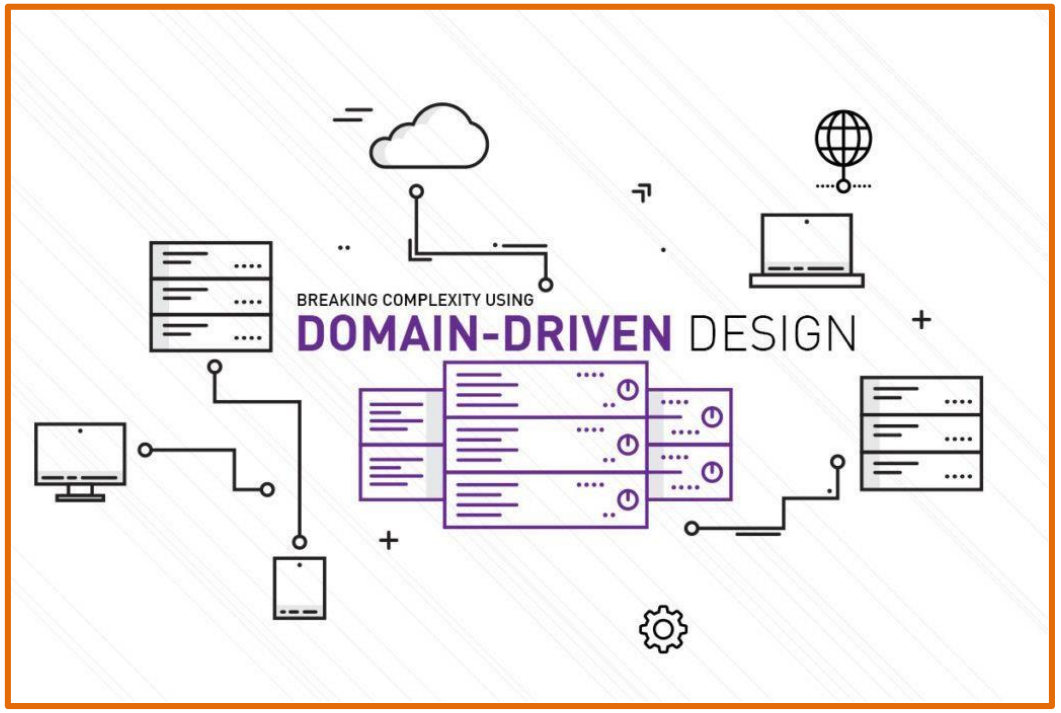
Formular conceptos y diseñar construcciones del lenguaje que permitan modelar estos aspectos transversales sin duplicación de código. En nuestro ejemplo, se necesitaría poder especificar de alguna manera concisa que antes de ejecutar ciertos métodos hay que llamar a cierto código.

```
1  public class MiObjetoDeNegocio {
2     public void metodoDeNegocio1() throws SinPermisoException {
3          chequeaPermisos();
4          //resto del código
5          ...
6     }
7
8     public void metodoDeNegocio2() throws SinPermisoException {
9          chequeaPermisos();
10         //resto del código
11         ...
12    }
13
14    protected void chequeaPermisos() throws SinPermisoException {
15         //chequear permisos de ejecucion
16       ...
17    }
18 }
19
```

Como controlo los permisos?

especificar de alguna manera concisa que *antes* de ejecutar *ciertos métodos* hay que llamar a *cierto código*.

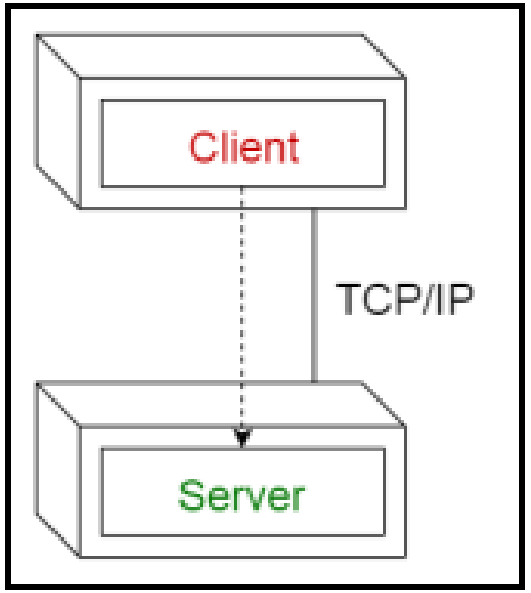
Arquitectura de Diseño Impulsado por Dominio (DDD)



El diseño impulsado por dominio es un enfoque orientado a objetos para diseñar software basado en el dominio empresarial, sus elementos y comportamientos, y las relaciones entre ellos. **Su objetivo es habilitar los sistemas de software que son una realización correcta del dominio empresarial subyacente mediante la definición de un modelo de dominio expresado en el lenguaje de los expertos en el dominio empresarial**. El modelo de dominio puede verse como un marco desde el cual las soluciones se pueden preparar y racionalizar.

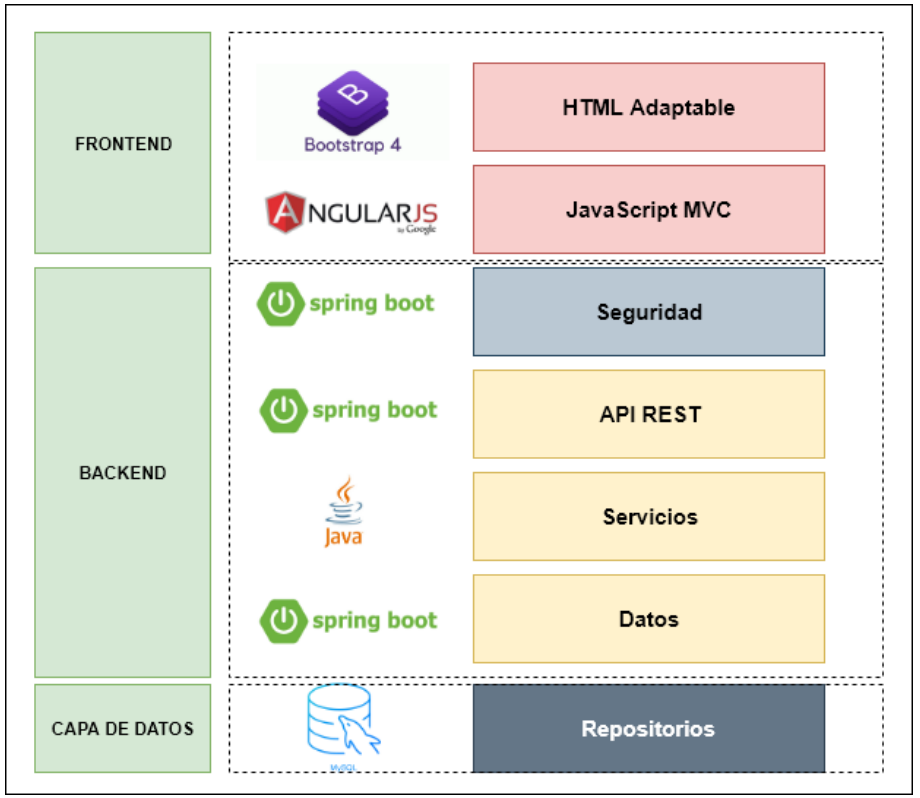
Arquitectura Cliente - Servidor

Este patrón segrega el sistema en dos aplicaciones principales, donde el cliente realiza solicitudes al servidor.
En muchos casos, el servidor es una base de datos con la lógica de la aplicación representada como procedimientos almacenados.
Este patrón ayuda a diseñar sistemas distribuidos que involucran un sistema cliente y un sistema servidor y una red de conexión. La forma más simple de la arquitectura cliente / servidor implica una aplicación de servidor a la que acceden directamente varios clientes.



Arquitectura en capas / en niveles

Normalmente, una aplicación de software empresarial comprende tres o más capas: capa de interfaz de presentación / usuario, capa de lógica de negocios y capa de persistencia de datos. Capas adicionales para permitir la integración con terceros



Arquitectura en capas / en niveles

La segregación en niveles

- Es útil para administrar y mantener cada capa en consecuencia.
- El poder del plug-in y el juego se realiza con este enfoque. Se pueden colocar capas adicionales según sea necesario. Hay marcos compatibles con el patrón del controlador de vista de modelo (MVC)
- La principal ventaja de la arquitectura en capas es la separación de
- preocupaciones Es decir, cada capa puede centrarse únicamente en su papel y responsabilidad.

Las capas y patrón de niveles hace que la aplicación:

- Mantenible
- Comprobable
- Fácil de asignar roles específicos y separados
- Fácil de actualizar y mejorar las capas por separado

Arquitectura de computación distribuida multinivel

- La arquitectura de dos niveles no es ni flexible ni extensible. Por lo tanto, la arquitectura de computación distribuida de múltiples niveles ha atraído mucha atención.
- Los componentes de la aplicación se pueden implementar en múltiples máquinas (estos pueden ubicarse conjuntamente y distribuirse geográficamente).
- Los componentes de la aplicación se pueden integrar a través de mensajes o llamadas a procedimientos remotos (RPC), invocaciones de métodos remotos (RMI), arquitectura de intermediario de solicitud de objetos comunes (CORBA), beans Java empresariales (EJB), etc.
- La implementación distribuida de los servicios de aplicaciones garantiza una alta disponibilidad, escalabilidad, capacidad de administración, etc. Las aplicaciones web, en la nube, móviles y otras aplicaciones orientadas al cliente se implementan utilizando esta arquitectura

Arquitectura SOA

Hemos estado jugando con los procesos de desarrollo de software orientados a objetos, basados en componentes, orientados a aspectos y basados en agentes.

Sin embargo, con la llegada de los paradigmas de servicio, los paquetes de software y las bibliotecas se están desarrollando como una colección de servicios.

Los servicios son capaces de ejecutarse independientemente de la tecnología subyacente. Además, los servicios pueden implementarse utilizando cualquier lenguaje de programación y script.

Los servicios son autodefinidos, autónomos e interoperables, públicamente detect evaluables, accesibles, reutilizables y compostables. Los servicios interactúan entr través de mensajes. Hay proveedores de servicios / desarrolladores y consumido clientes. Existen servicios de descubrimiento de servicios que aprovechan de f innata los registros y repositorios de servicios públicos y privados. Los servicio cliente pueden encontrar sus servicios de servicio de forma dinámica a través d servicios de descubrimiento de servicios.

Arquitectura SOA

Como son los servicios ?

- Son autodefinidos.
- Autónomos e interoperables
- Públicamente detectables,
- Evaluables,
- Accesibles,
- Reutilizables y compostables.

Los servicios interactúan entre sí a través de mensajes. Existen servicios de descubrimiento de servicios que aprovechan de forma innata los registros y repositorios de servicios públicos y privados.
Los servicios del cliente pueden encontrar sus servicios de servicio de forma dinámica a través de los servicios de descubrimiento de servicios.

Arquitectura SOA orientada a eventos

Como afecta las comunicaciones Asincronas a SOA?

SOA no es lo suficientemente bueno para manejar eventos en tiempo real de forma asíncrona

Es por eso que el nuevo patrón de SOA impulsada por eventos, que combina intrínsecamente la probada respuesta de SOA probada y los paradigmas de publicación-suscripción de eventos de EDA, está adquiriendo mucha atención y atracción en estos días

Los beneficios del patrón compuesto ED-SOA

Que mejorar?

Una aplicación monolítica pone toda su funcionalidad en un solo proceso. Para escalar, es obligatorio replicar toda la aplicación. Sin embargo, la partición de una aplicación en una colección de servicios de aplicación dinámicos facilita la elección y la replicación de uno o más componentes / servicios de aplicación para el escalado.

Integración efectiva de datos: en la arquitectura controlada por solicitud síncrona, el enfoque está en la reutilización de funciones retenidas remotamente e implementar la integración orientada a procesos, la integración de datos (entornos integrados, no es compatible) en forma innata en SOA

Oportunidad y confiabilidad: los eventos se propagan en tiempo real en todas las aplicaciones participantes para la captura, procesamiento, toma de decisiones y actuación de datos en tiempo real

Escalabilidad y sostenibilidad mejoradas: es un hecho que los sistemas asíncronos tienden a ser más escalables en comparación con los sistemas síncronos. Los procesos individuales bloquean menos y tienen menos dependencia de los procesos remotos / distribuidos

Arquitectura de Microservicios - MSA

El patrón arquitectónico orientado al servicio ha evolucionado durante décadas para expresar y exponer cualquier aplicación masiva, monolítica y masiva como una colección dinámica de servicios interdependientes. Los servicios están bendecidos con interfaces e implementaciones. Las interfaces son el mecanismo de contacto y contratación para cualquier aplicación habilitada para servicios.

El popular estilo SOA se basa principalmente en un modelo de datos compartido con múltiples jerarquías. El intercambio de bases de datos en SOA tiende a crear un tipo de acoplamiento de datos estrecho entre los servicios y otros componentes del sistema.

El otro desafío es que los servicios en el estilo SOA son típicamente de grano grueso y, por lo tanto, el aspecto de la reutilización es un asunto bastante difícil

Arquitectura de Microservicios - MSA

Que me debe permitir lograr el patrón de microservicios ?

Lograr de manera fácil y rápida los requisitos no funcionales, como la escalabilidad, disponibilidad y confiabilidad para cualquier aplicación de software..

El patrón MSA es para producir servicios de grano fino, acoplados de forma flexible, escalables horizontalmente, desplegables de forma independiente, interoperables, públicamente detectables, accesibles a la red, fácilmente manejables y compactables, que no solo es la unidad optimizada de construcción de software, sino que también permite la habilitación. Implementación y entrega de software más rápida

Orquestación asistida - MSA

El creciente ecosistema de herramientas, motores, plataformas y otras soluciones de infraestructura de software acelera el proceso de producción de aplicaciones centradas en microservicios. Con la orquestación asistida por herramientas, los microservicios se están orquestando hábilmente para presentar aplicaciones versátiles para la automatización, aceleración y aumento de negocios.

No hay tecnología o bloqueo de proveedores en lo que se refiere a las aplicaciones inspiradas en MSA. Cada microservicio está habilitado con su propia fuente de datos, que puede ser un sistema de archivos, SQL, NoSQL, NewSQL, caché en memoria, etc. Existen soluciones de puerta de enlace API para agilizar la administración de los microservicios de ciclo de vida de extremo a extremo

Patrón de Microservicios basados en Eventos

Las aplicaciones centradas en microservicios están siendo habilitadas para ser impulsadas por eventos. Hay algunos patrones arquitectónicos interesantes que emergen y evolucionan rápidamente. **Aparece el patrón de flujo de eventos**

Que ventajas nos proporciona este enfoque?

Método de mensajería políglota descentralizada: enfoque de microservicios ofrece más flexibilidad a los desarrolladores para elegir la solución de middleware de mensajería óptima.

Cada caso de uso tendrá sus propias necesidades específicas que obligan a diferentes tecnologías de mensajería como Apache Kafka, RabbitMQ o incluso cuadrículas de datos NoSQL controladas por eventos, como Apache Geode / Pivotal GemFire.

Patrones de arquitectura multinivel cliente / servidor

Patrones Arquitecturales

Necesidad de la evolución de los patrones cliente-servidor de dos niveles, condujeron a la evolución de los patrones cliente-servidor de tres niveles y, en consecuencia, de n niveles.

Las diferentes variantes de los patrones cliente-servidor, como el patrón maestro-esclavo, los patrones de igual a igual, etc., también se explican en profundidad con los casos de uso relevantes.

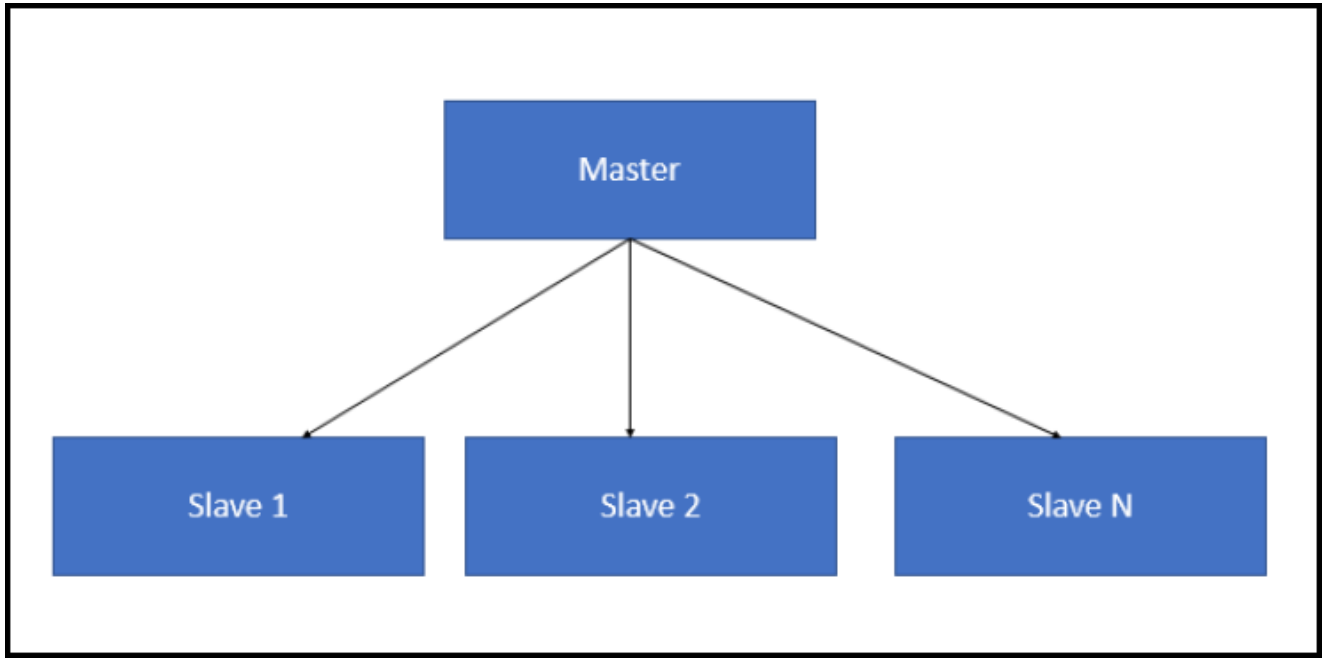
Los requisitos de las aplicaciones web son diferentes de los de las aplicaciones cliente-servidor, siendo el factor diferenciador clave las actualizaciones dinámicas de la interfaz de usuario basadas en los cambios en los datos subyacentes.

Patrones Arquitecturales

- Patrones cliente-servidor de dos niveles, tres niveles y n niveles
- Two-tier, three-tier, and n-tier client-server patterns
- Patron master-slave
- Patron peer-to-peer
- Patron distributed client-server
- Patron model-view-controller
- Patron model-view-presenter
- Patron model-view-model
- Patron front controller

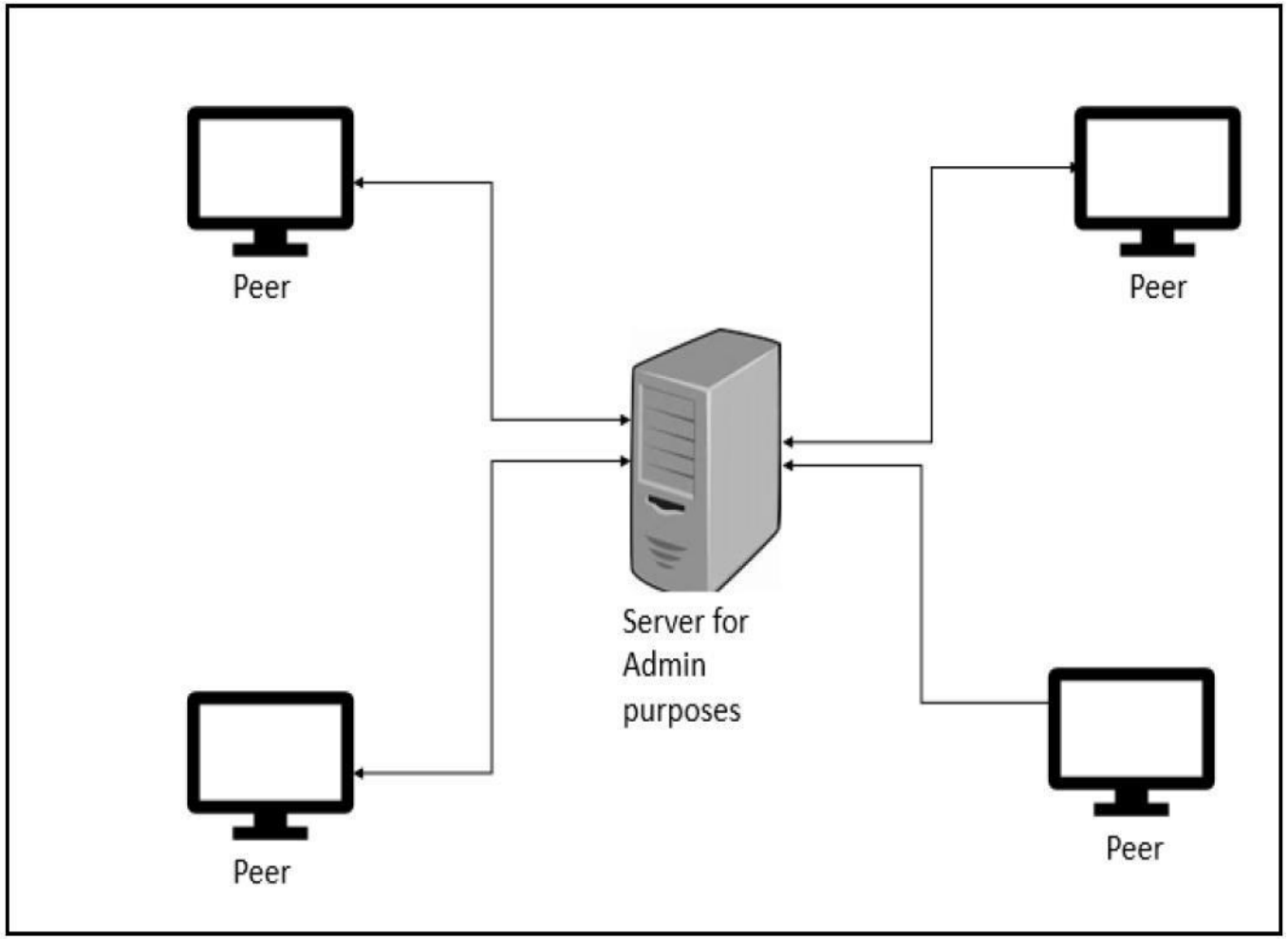
Patrón Master / Slave

El componente maestro distribuye el trabajo entre todos los componentes esclavos y calcula el resultado final al resumir los resultados que devuelve cada esclavo. El patrón maestro-esclavo se usa para la arquitectura de sistemas embebidos y se usa en el diseño de sistemas que realizan cálculos masivos en paralelo. El siguiente es un diagrama de secuencia del patrón maestro-esclavo:



Patrón Peer-to-Peer

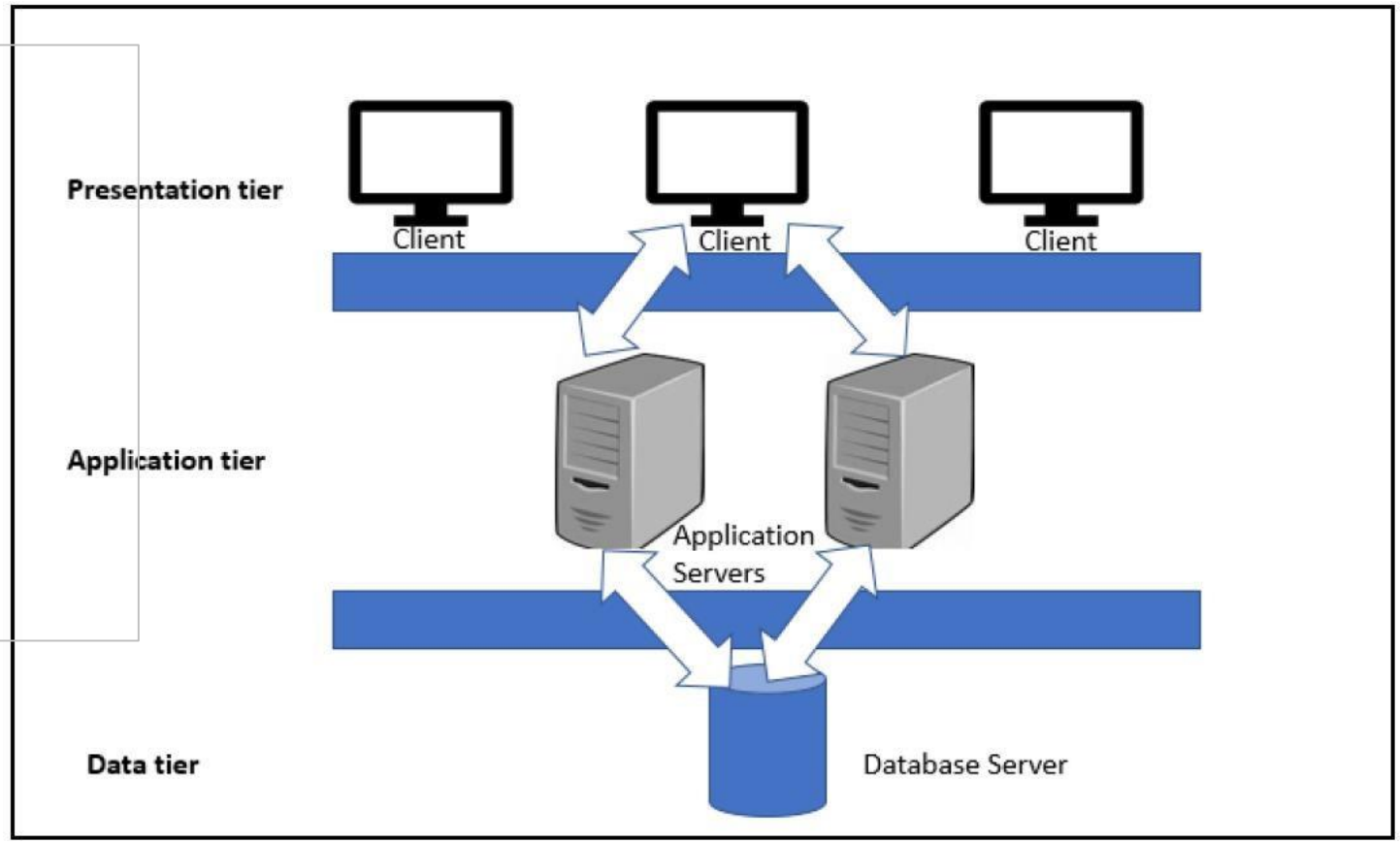
Los patrones arquitectónicos de igual a igual pertenecen a la categoría de **patrones cliente-servidor simétricos**. Simétrico en este contexto se refiere al hecho de que no hay necesidad de **una división estricta en términos de cliente, servidor**, etc. en la red de sistemas. En un patrón de igual a igual, un solo sistema actúa como cliente y como servidor. Cada sistema, también llamado un par, envía solicitudes a otros pares en la red y al mismo tiempo recibe y atiende solicitudes de otros pares, que son parte de la red



Arquitectura cliente-servidor de tres nivel

Los tres niveles que están presentes en esta arquitectura son los siguientes:

- El nivel de presentación
- El nivel de lógica de negocios o aplicación
- Nivel de datos



Los beneficios de la arquitectura de tres niveles:

Como nos beneficia?

Escalabilidad y flexibilidad: la principal ventaja de esta arquitectura es su escalabilidad y flexibilidad. Cada nivel de esta arquitectura es un componente modular, es decir, cualquier tipo de operaciones como cambios o actualizaciones realizadas a un nivel no afectan o causan tiempo de inactividad a los otros niveles.

Mayor seguridad: la división de tareas entre los distintos niveles brinda mayor seguridad a cada nivel.

todavía hay límites en la escalabilidad de la arquitectura cuando se trata de redes como Internet, que requieren una escalabilidad masiva.

Consideraciones de diseño para el uso de 3 capas:

- Si está desarrollando una aplicación con funcionalidad / configuración limitada para sistemas cliente. En este caso, otros componentes de la arquitectura, como la lógica de negocios y la lógica de datos, pueden distribuirse a otros niveles.

- Si está en el proceso de desarrollar una aplicación que se implementará dentro de una intranet donde todos los servidores están ubicados dentro de una red privada específica.

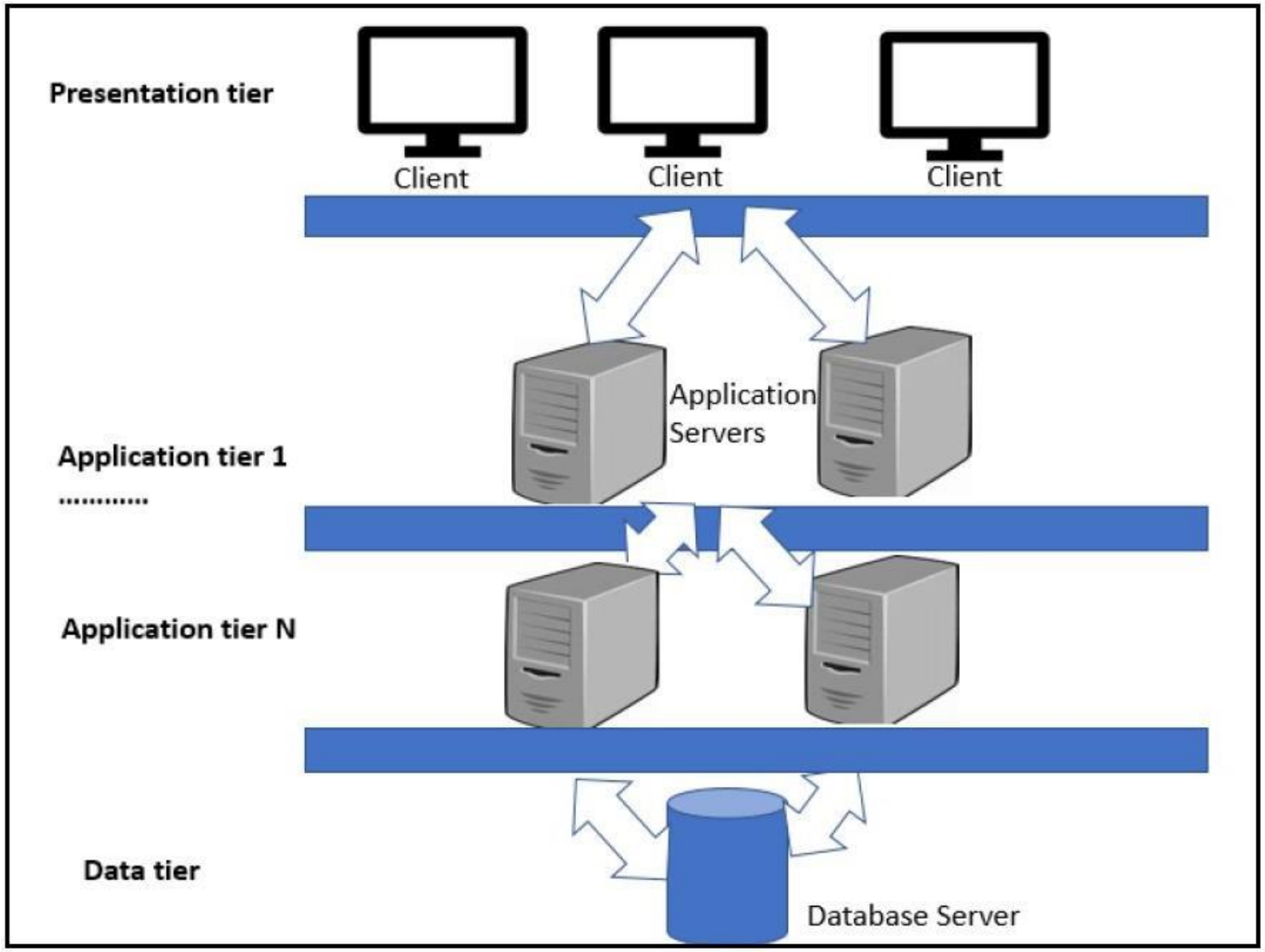
- Si está desarrollando una aplicación de Internet donde no hay restricciones de seguridad para implementar la lógica empresarial en las redes públicas de los servidores web o de aplicaciones.

Patrón de Arquitectura de n-capas:

- Una variante del patrón arquitectónico de tres niveles que ofrece una escalabilidad masiva.
- En un patrón arquitectónico de n niveles, el número total de niveles es n, donde n tiene un valor mayor que tres para diferenciarlo del patrón arquitectónico de tres niveles.
- En la arquitectura de n niveles, el nivel de aplicación (que es el nivel medio) se divide en muchos niveles.

- La distribución del código de aplicación y las funciones entre los distintos niveles varía de un diseño arquitectónico a otro. El diagrama del patrón arquitectónico de n niveles se muestra a continuación

Patrón de Arquitectura de n-capas:



Consideraciones de diseño para el uso de n-capas:

- Si está diseñando un sistema en el que es posible dividir la lógica de la aplicación en componentes más pequeños que podrían estar distribuidos en varios servidores. Esto podría llevar al diseño de múltiples niveles en el nivel de aplicación.

- Si el sistema en consideración requiere comunicaciones de red más rápidas, alta confiabilidad y gran rendimiento, n-tier tiene la capacidad de proporcionar eso, ya que este patrón arquitectónico está diseñado para reducir la sobrecarga causada por el tráfico de red

Ejemplo: aplicación web de carrito de compras

Como trabaja o se compone una aplicación de Carrito de compras?

Ejemplo: aplicación web de carrito de compras

El usuario del sitio de comercio electrónico utiliza la aplicación web del carrito de compras para completar la compra de artículos a través del sitio de comercio electrónico. La aplicación debe tener varias características que permitan al usuario realizar actividades como las siguientes:

- Adición de artículos seleccionados al carrito
- Cambio de cantidades de artículos en el carrito
- Realización de pagos

Presente con su equipo de trabajo el esquema de carrito de compras en capas.

Arquitectura Cliente /Servidor Distribuida

La arquitectura cliente-servidor de n niveles utilizada para la aplicación web del carrito de compras, que se describe en la sección anterior, es un ejemplo ideal de una arquitectura distribuida cliente-servidor.

Las arquitecturas distribuidas suelen tener algún tipo de componentes de servidor backend (como Mainframe, servidor de base de datos, etc.), un cliente inteligente en la interfaz y varios agentes en el medio, que se encargan de todas las actividades relacionadas con transacciones como el procesamiento de transacciones. seguridad, manejo de mensajes, etc., y una red para la comunicación..

Patrones de la Arquitectura de Microservicios

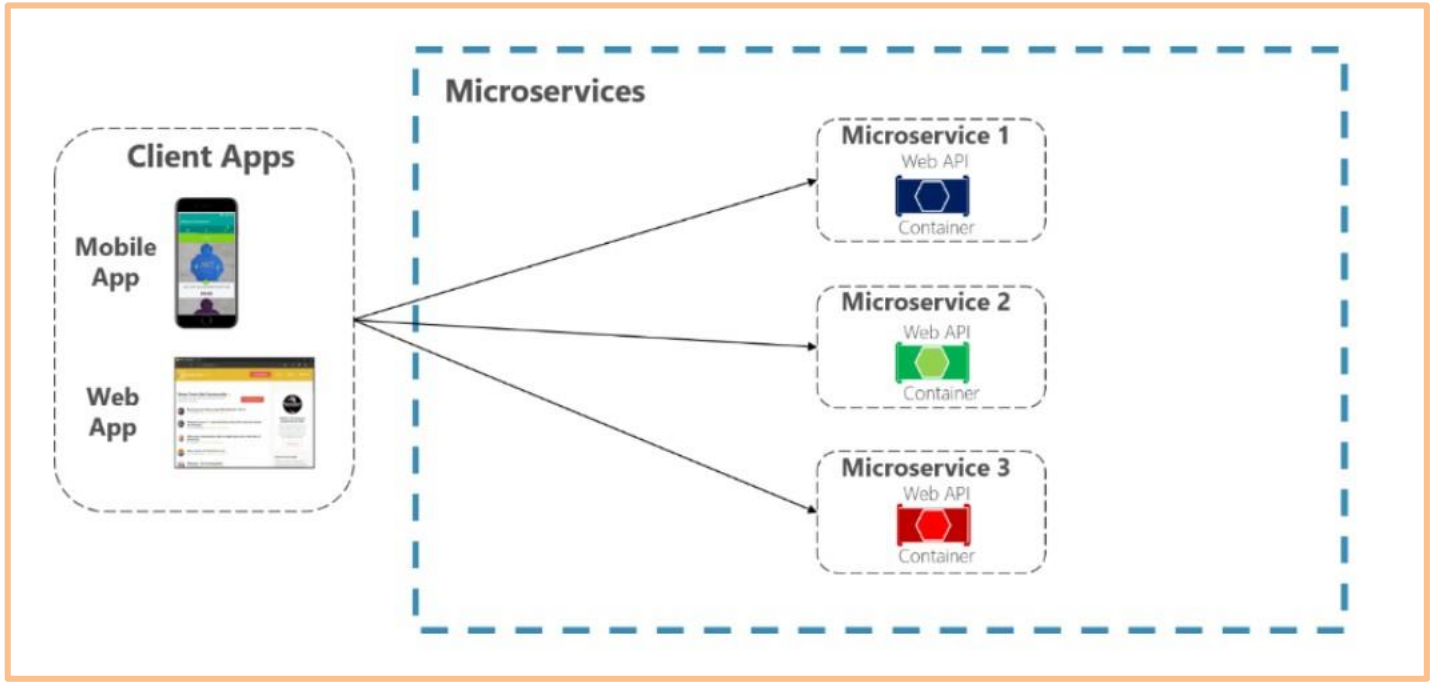
Patrones de diseño de Microservicios

El diseño de microservicios competentes que puedan funcionar con otros servicios de manera ininterrumpida y espontánea es esencial para el éxito previsto de la MSA.

El diseño de la arquitectura de las aplicaciones de nube, empresa, móvil, IoT, analítica, operativa y transaccional a través del poder de los microservicios se debe realizar de manera elegante y conveniente.

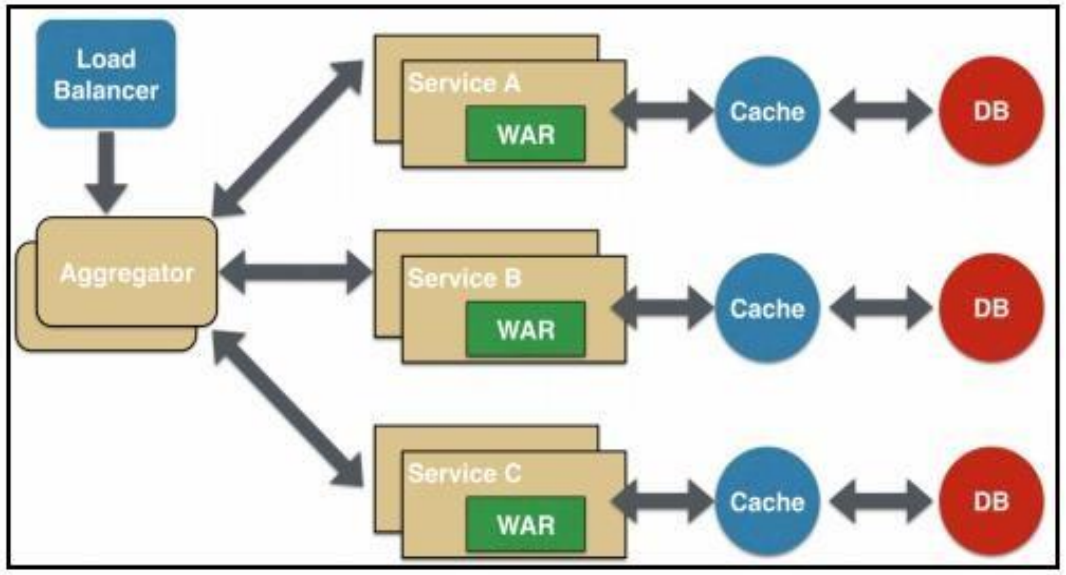
Además, el resplandeciente paradigma de MSA es futurista en el sentido de que cualquier nueva tecnología se puede usar fácilmente para producir microservicios de próxima generación que sean fácilmente accesibles, evaluables, maniobrables, reemplazables, sustituibles, etc.

Comunicación directa de un cliente a un microservicio



Patrones de diseño de Microservicios

Agregator:



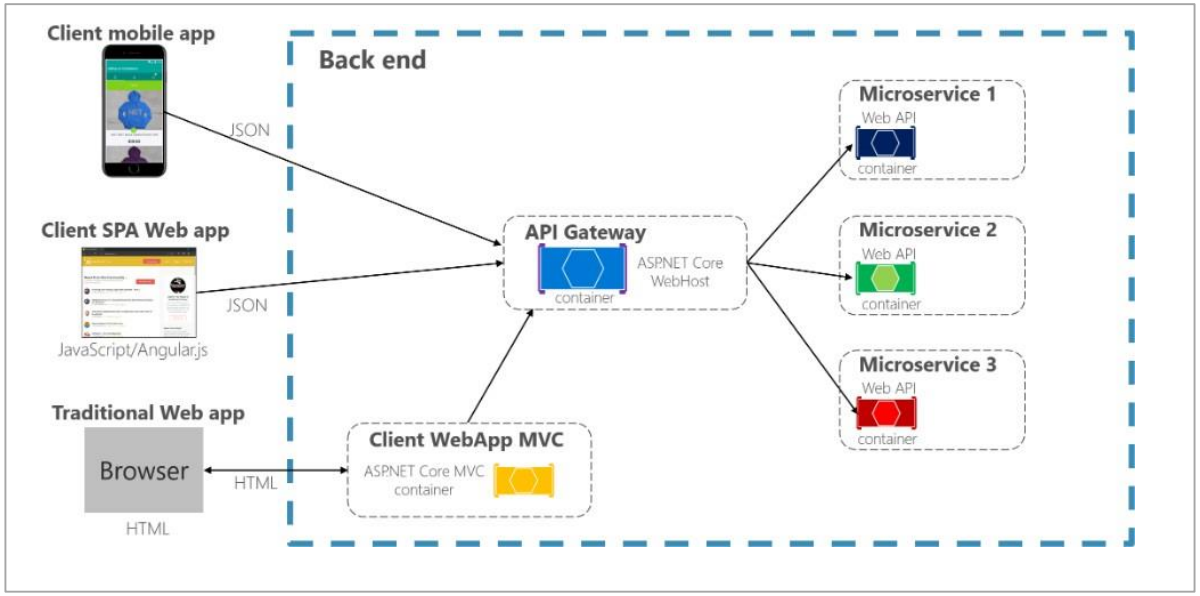
- Los servicios son relativamente pequeños en tamaño y, por lo general, un microservicio implementa una sola tarea

Se deben identificar y agregar múltiples servicios descentralizados y distribuidos para cumplir con una funcionalidad y funcionalidad empresarial completa.

El patrón agregador es, por lo tanto, esencial para la era MSA. Dado que cada servicio se expone mediante la interfaz RESTful de peso ligero, una aplicación, que comprende muchos microservicios, puede recuperar los datos de diferentes servicios y procesarlos / mostrarlos en consecuencia utilizando este patrón agregador

Usando un único servicio de API Gateway

Proxy: ligera variación de agregador



microservicios.

El patrón de proxy también puede independientemente, la idea es que

En este caso, el cliente no está involucrado en la actividad de agregación. Según las necesidades del negocio, se pueden invocar diferentes escalar cada

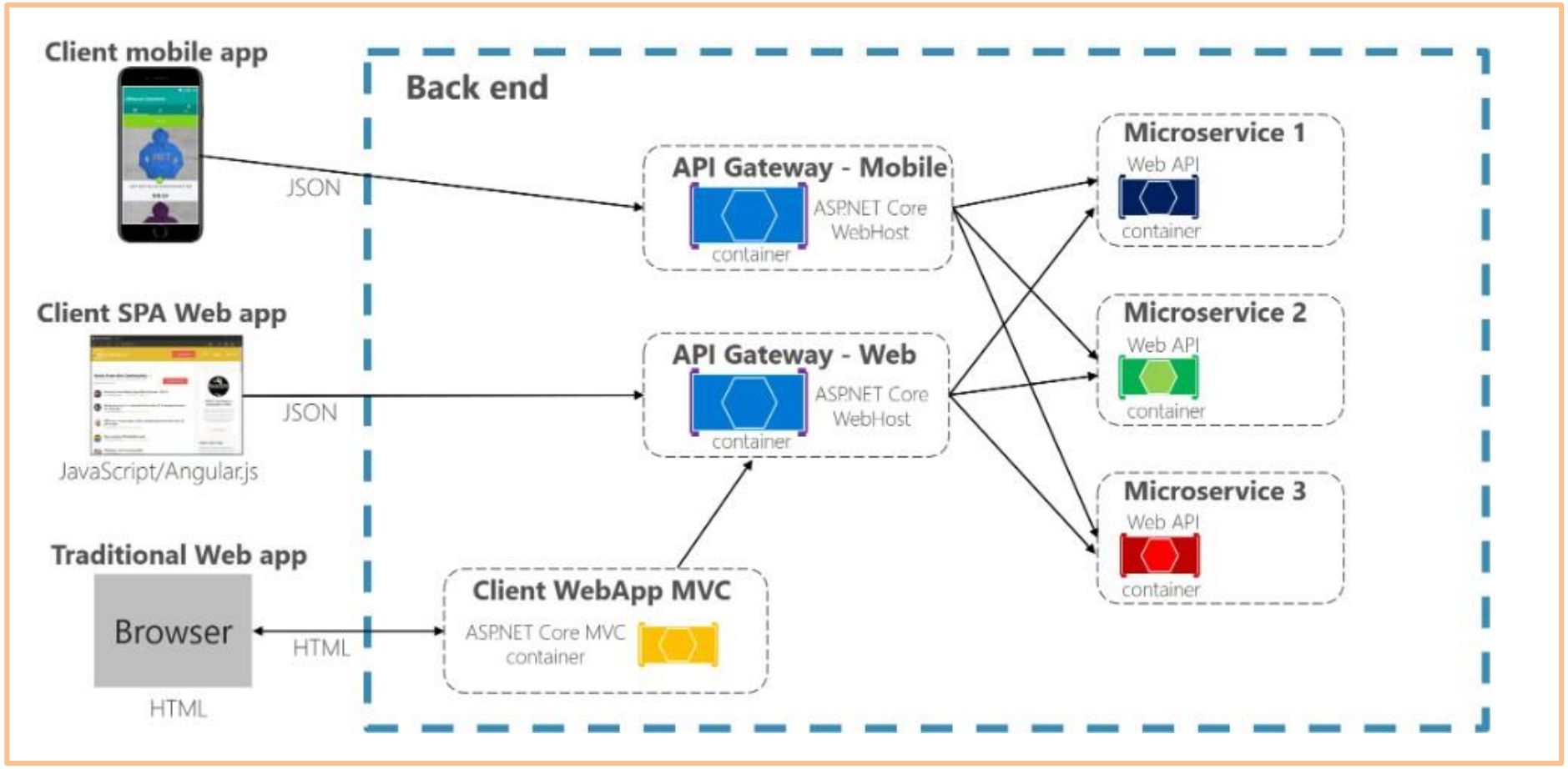
microservicio no necesita estar expuesto al consumidor.

El proxy puede ser un proxy tonto, en cuyo caso simplemente delega la solicitud a uno de los servicios. Alternativamente, puede ser un proxy inteligente donde se aplica alguna transformación de datos antes de que la respuesta se envíe al cliente.

Con la explosión de diferentes dispositivos de IoT y E / S, este patrón de proxy es beneficioso

Usando múltiples API Gateway / BFF

Proxy:



Patrones de diseño de Microservicios

Encadenado:

Esto es para producir una única respuesta consolidada a una solicitud. En este caso, la solicitud del cliente es recibida por el Servicio A, que luego se comunica con el Servicio B, que a su vez puede estar comunicándose con el Servicio C. Es probable que todos los servicios estén utilizando un mensaje de solicitud / respuesta HTTP síncrono. La principal preocupación aquí es que el cliente está bloqueado hasta que todos los servicios de la cadena finalicen el procesamiento. Es decir, la cadena del Servicio A al Servicio B y luego el Servicio B al Servicio C se completa. La cadena debe ser corta y pequeña; de lo contrario, la comunicación síncrona puede provocar un retraso.

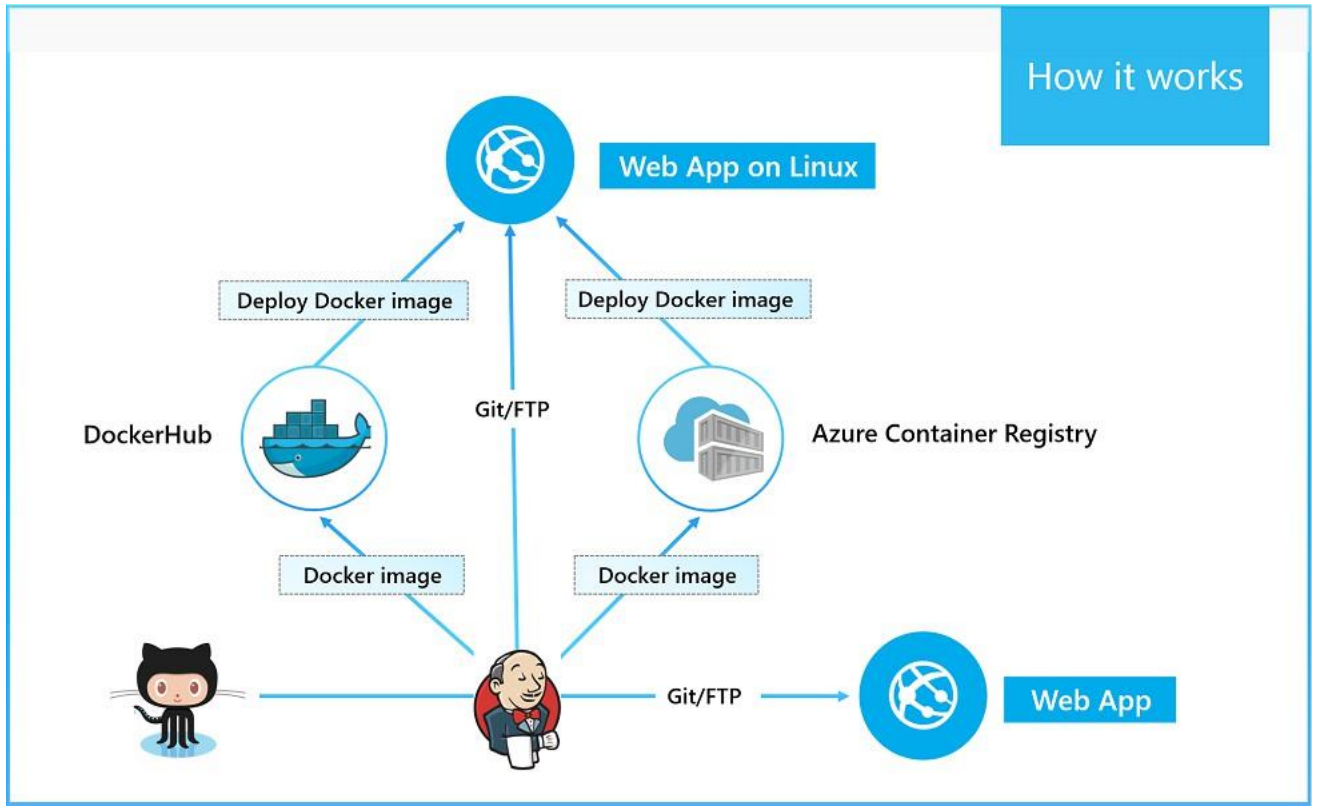
Base de datos:

Patrones de diseño de Microservicios

La persistencia de datos es un factor importante en cualquier microservicio. Recientemente se han creado nuevos sistemas de gestión de bases de datos para almacenar datos sin procesar y procesados. Hay grandes, rápidos, transmisión de datos, datos de IoT y varios tipos de procesamiento de datos, como procesamiento por lotes, en tiempo real, interactivo e iterativo. Las tecnologías y herramientas de captura de datos, ingesta, almacenamiento, procesamiento, minería, análisis y visualización de datos nuevos están surgiendo y evolucionando para respaldar la información basada en datos y las decisiones basadas en información.

Patrones para aplicaciones en contenedores y confiables

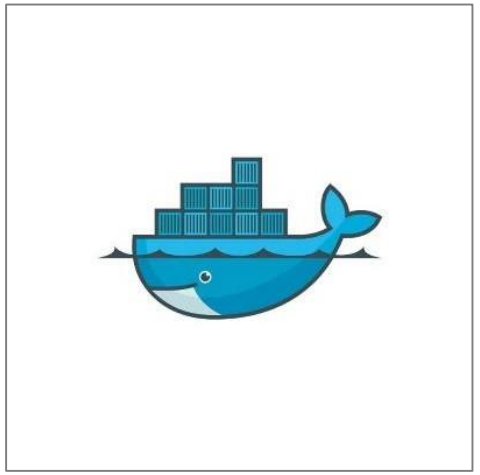
Contenerizacion con Docker



El paradigma de la contenedorización habilitada por Docker está en el camino correcto para convertirse en una tecnología impactante y perspicaz con una serie de avances cruciales introducidos por una creciente gama de productos de terceros y proveedores de herramientas.

Especialmente, el futuro pertenece a los entornos de nube en contenedores con la disponibilidad inmediata de tecnologías y herramientas de desarrollo, implementación, redes y composición de contenedores probados

Contenerizacion con Docker



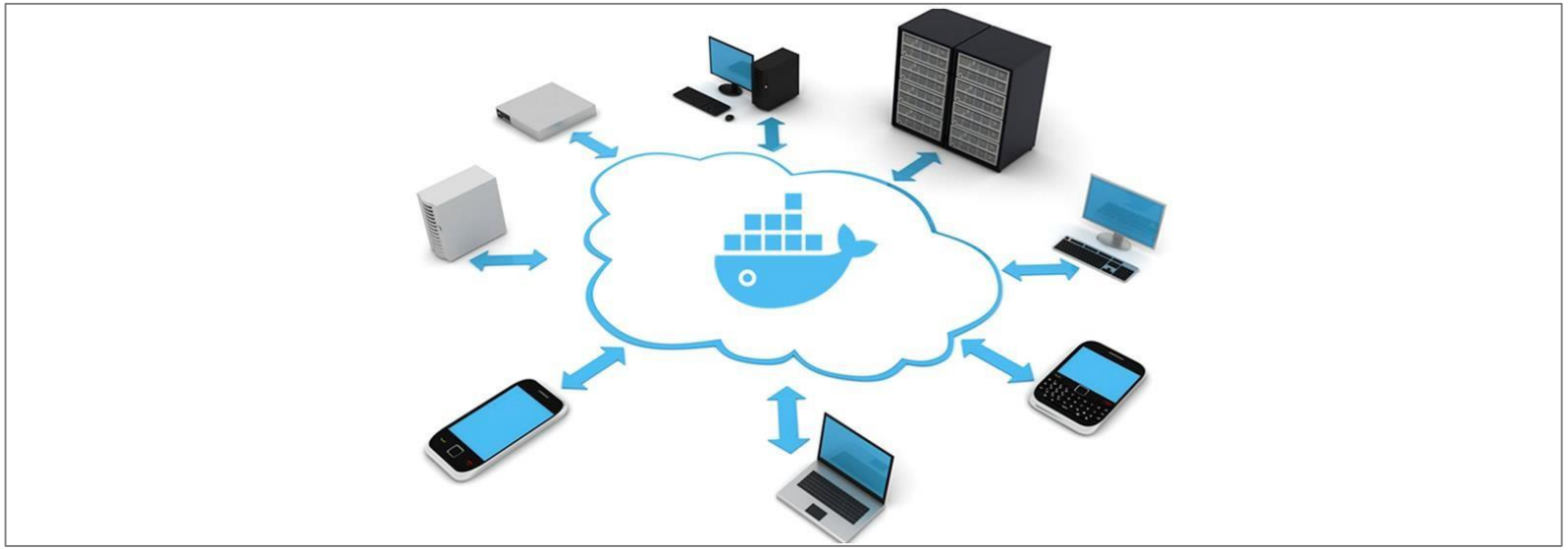
Los contenedores habilitados para Docker en asociación con las plataformas de orquestación, gobernanza, monitoreo, medición y administración, como Kubernetes, Mesos, etc., contribuirán inmensamente a la configuración y el mantenimiento de entornos de nube de contenedores de próxima generación que son muy famosos por su entrega.

- Aplicaciones de clase empresarial, basadas en microservicios, basadas en: eventos, orientadas a servicios,
- alojadas en la nube, llenas de conocimiento,
- integradas,
- habilitadas para AI,
- centradas en las personas, de nivel de operador,
- listas para la producción y
- sensibles a la infraestructura

Tendencias principales

Docker de código abierto se está equipando continuamente con más características y funciones adecuadas y relevantes para convertirla en la plataforma de TI más ejemplar.

Adopción sin precedentes de la tecnología de contenedores inspirada en Docker por parte de varios proveedores de servicios y soluciones de TI en todo el mundo para ofrecer una creciente gama de ofertas premium a sus venerables consumidores y clientes



Beneficios de Docker

La contenedorización es una forma poderosa de empaquetar e implementar herramientas. Docker , como su nombre lo indica, es una herramienta de implementación que empaqueta su código junto con todas las dependencias en un contenedor de aplicaciones de tal manera que no requiere una Máquina Virtual completa para ejecutarse.

Esto significa que si está ejecutando Linux OS en su PC y empaqueta toda esa aplicación en un contenedor, se ejecutarán igual de bien en la nube o en cualquier servidor estándar. Docker hizo popular a la virtualización basada en contenedores y casi todos los proveedores de la nube pública se están subiendo al carro, proporcionando su propio Container como servicio (CaaS)

Beneficios de Docker

Portabilidad

No se requiere una máquina virtual para cada aplicación.

Recursos compartidos

Docker usa Kernel compartido, lo que significa que son mucho más eficientes que los hipervisores en términos de recursos del sistema.

Docker es fácil de lanzar

Docker comenzó este nuevo proyecto llamado Libswarm que potencialmente facilitaría el uso de contenedores en la nube pública

Despliegue de tiempo de inactividad cero

Desplegar a menudo y rápidamente, ser completamente automático, lograr cero tiempos de inactividad, tener la capacidad de revertir, proporcionar una confiabilidad constante en todos los entornos, ser capaz de escalar sin esfuerzo y crear la autocuración. Sistemas capaces de recuperarse de fallos. Esto significa que puede tener DevOps Continuous Integration (CI), Continuous Delivery / Deployment (CD), lograr un tiempo de inactividad cero, tener la capacidad de revertir los productos de software

Los impulsores clave para la contenedorizacion.

La virtualización es un tipo de abstracción beneficiosa, que se logra mediante la incorporación de una capa adicional de direccionamiento indirecto entre los recursos de hardware y los componentes de software

A través de esta capa de abstracción recién introducida (monitor de máquina virtual o hipervisor (VMM)), cualquier tipo de aplicación de software puede ejecutarse en cualquier hardware subyacente sin ningún problema o dificultad. En resumen, la portabilidad del software se está logrando a través de esta capa de middleware. Sin embargo, el objetivo de portabilidad, muy publicado, no se cumple completamente con la técnica de virtualización. El software del hipervisor y los diferentes formatos de encapsulación de datos de diferentes proveedores impiden la portabilidad de la aplicación tan necesaria. Además, las diferencias de distribución, versión, edición y parches de los sistemas operativos y las cargas de trabajo de las aplicaciones dificultan la portabilidad sin problemas de las cargas de trabajo en los sistemas y ubicaciones.

Patrones de desarrollo Cloud Native

Nativo de la Nube.

Que es ser nativo en Nube ?

Cada empresa comienza a migrar sus cargas de trabajo a la nube.
Con esta migración se llega a la conclusión de que los sistemas deben ser rediseñados para desbloquear completamente el potencial de la nube.

- Es más que optimizar para la nube.
- Permite a las empresas ofrecer continuamente innovación con confianza.
- Capacita a los equipos de todos los días para crear sistemas a gran escala.
- Es una forma completamente diferente de pensar y razonar sobre la arquitectura de software.

Definiendo tu contexto para una definición.

¿Cuál es el contexto adecuado para nuestra definición de nube nativa? Bueno, por supuesto, **el contexto correcto es tu contexto**. Usted vive en el mundo real, con problemas del mundo real que está tratando de resolver. Si el nativo de la nube será de alguna utilidad para usted, debe ayudarlo a resolver sus problemas del mundo real. ¿Cómo definiremos tu contexto? Comenzaremos por definir lo que su contexto no es..

Su contexto.

Es su negocio y sus clientes y lo que es correcto para ambos. Usted tiene una gran experiencia, pero sabe que hay un cambio radical con un gran potencial para su empresa y desea saber más. **En esencia, su contexto es mayoritario**. No tiene capital ilimitado, ni un ejército de ingenieros, pero sí tiene la presión del mercado para ofrecer innovación ayer, no mañana, mucho menos el mes próximo o más allá. Necesita hacer más con menos, hacerlo rápido y seguro y estar listo para escalar

Por que es importante Cloud Native ?.

ejecutar sus aplicaciones en la nube es diferente de ejecutarlas en centros de datos tradicionales, es una dinámica diferente.

Un sistema que fue diseñado y optimizado para ejecutarse en un centro de datos tradicional no puede aprovechar todos los beneficios de la nube, **como la elasticidad**

La promesa de la nube nativa es la velocidad, la seguridad y la escalabilidad y ofrecer rápidamente innovación de mercado

Definiendo Cloud Native.

Nativo de la nube es una forma diferente de pensar y razonar acerca de los sistemas de software.

- Desarrollado por infraestructura desechable Compuesto por componentes limitados y aislados
- Escalas globalmente
- Abraza la arquitectura desechable
- Aprovecha los servicios en la nube de valor agregado
- Da la bienvenida a la nube políglota
- Permite a los equipos autosuficientes y de pila completa Impulsa el cambio cultural

Alimentado por infraestructura desechable..

Nativo de la nube es una forma diferente de pensar y razonar acerca de los sistemas de software.

- Desarrollado por infraestructura desechable Compuesto por componentes limitados y aislados
- Escalas globalmente
- Abraza la arquitectura desechable
- Aprovecha los servicios en la nube de valor agregado
- Da la bienvenida a la nube políglota
- Permite a los equipos autosuficientes y de pila completa Impulsa el cambio cultural

Compuesto por componentes limitados, aislados

Desplegar todo el monolito con la participación de cada equipo e, inevitablemente, se romperá algo completamente relacionado como resultado del despliegue

Un sistema monolítico en sí mismo puede convertirse en el cuello de botella para su propio avance, mientras que el segundo escenario muestra cómo el sistema puede ser su propio talón de Aquiles.
En los sistemas nativos de la nube, evitamos problemas como estos al descomponer el sistema en componentes aislados limitados. Los componentes limitados están enfocados

La automatización y la infraestructura desechable ayudan a minimizar el potencial de estos errores y nos permiten recuperarnos rápidamente de dichos errores, pero no pueden eliminarlos. Por lo tanto, los sistemas nativos de la nube deben ser resistentes al error humano

Escalas a nivel mundial

"Hay aspectos positivos y negativos de haber estado en nuestra industria durante mucho tiempo. En el lado bueno, has visto mucho, pero en el lado malo, tiendes a pensar que lo has visto todo"

Podría implementar el código de UI en AWS S3 y entregarlo a través de AWS CloudFront CDN. No necesitaría un equilibrador de carga elástico (ELB) delante de un mínimo de dos instancias de EC2 que ejecutan Apache, a su vez frente a otra (ELB) frente a un grupo de al menos dos instancias de EC2 que ejecutan un servidor de aplicaciones Java

Ejecutar el nivel de presentación en el borde de la nube como este fue un cambio de juego. Permitió una escala global prácticamente ilimitada, para ese nivel, prácticamente sin costo

Abraza la arquitectura desechable

Es esencial que aprovechemos la infraestructura desechable para implementar y escalar de manera independiente los componentes aislados limitados. Pero esto es solo el comienzo de las posibilidades. A su vez, la arquitectura desechable se basa en esta base, lleva la idea de la desechabilidad y el reemplazo al siguiente nivel, e impulsa aún más el valor comercial.

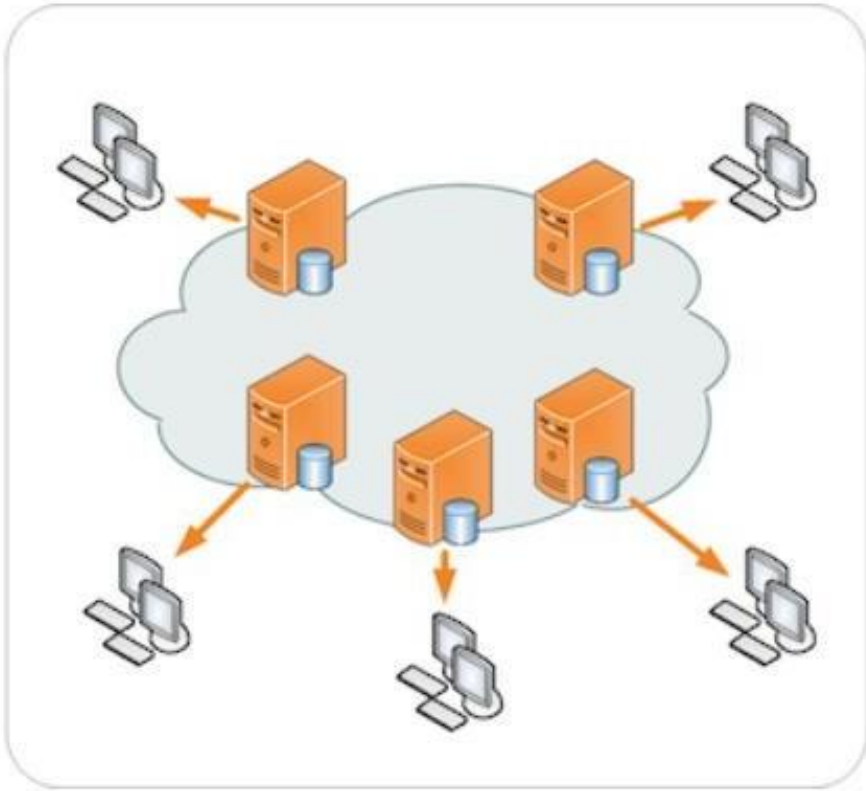
El monolito está grabado en nuestros cerebros e impregna nuestra forma de pensar. Nos lleva a decisiones de arquitectura y de negocios que pueden ser óptimas en el contexto del monolito, pero no en el contexto de la nube nativa

Abraza la arquitectura desechable

Que es una CDN y como se diferencia del Hosting ?

Es un tipo de infraestructura informática en la que se entrelazan varios ordenadores distribuidos geográficamente en varios data centers. Estos almacenan parte de la información y el contenido de los sitios web y los servirán al usuario final.

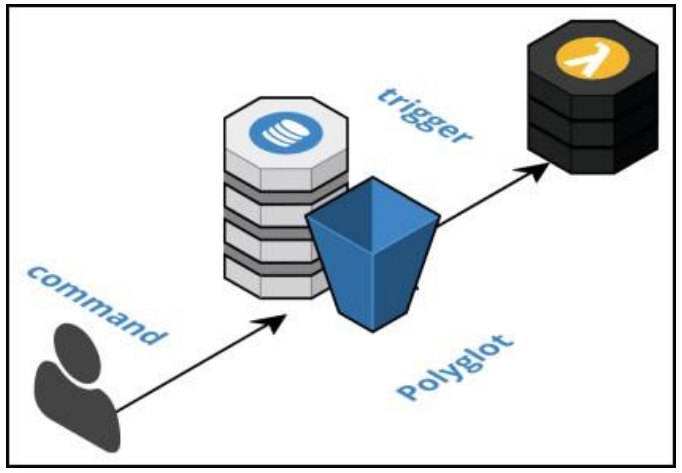
Sus ventajas son que mejoran la disponibilidad del servidor (*uptime*), alivian la carga de tráfico de este, son una barrera más de seguridad contra ataques informáticos, y además mejoran el rendimiento y los tiempos de carga.



Patrones de cimentación.

Bases de datos nativas en la nube por componente

Aproveche una o más bases de datos nativas en la nube totalmente administradas que no se comparten entre los componentes y reaccionan a los eventos emitidos para desencadenar la lógica de procesamiento intra-componente



Posibilitar la entrega de sistemas a escala global de manera rápida, continua y confiable. Las bases de datos modernas, de alto rendimiento, horizontalmente escalables y fragmentadas son fundamentales para lograr la escalabilidad global.

Estas bases de datos vienen en muchas variaciones que están especializadas para las características particulares de la carga de trabajo.

Cada componente tiene sus propias características de carga de trabajo, lo que requiere el uso de persistencia políglota, por lo que se emplean muchos tipos diferentes de bases de datos para respaldar estas características.

Bases de datos nativas en la nube por componente

Contexto, Problemas y Fuerzas

Posibilitar la entrega de sistemas a escala global de manera rápida, continua y confiable. Las bases de datos modernas, de alto rendimiento, horizontalmente escalables y fragmentadas son fundamentales para lograr la escalabilidad global.

Estas bases de datos vienen en muchas variaciones que están especializadas para las características particulares de la carga de trabajo.
Cada componente tiene sus propias características de carga de trabajo, lo que requiere el uso de persistencia políglota, por lo que se emplean muchos tipos diferentes de bases de datos para respaldar estas características.

Bases de datos nativas en la nube por componente

Solución

- Aproveche los servicios de base de datos nativos de la nube totalmente administrados de su proveedor de nube.
- Emplee múltiples tipos de bases de datos dentro de un componente, según sea necesario, para coincidir con las características de carga de trabajo del componente.
- Elija el tipo de base de datos, como el almacén de documentos, el almacenamiento de blob o la búsqueda tabla por tabla.
- Cada base de datos está dedicada a un componente específico y no se comparte entre los componentes.
- Utilice las funciones de gestión de captura de datos de cambios y de ciclo de vida y reaccione a los eventos emitidos para activar la lógica de procesamiento intra-componente.
- Utilice las funciones de replicación regional para crear implementaciones multirregionales, según sea necesario..

ases de datos nativas en la nube por compone

Existen dos modelos de entorno de base de datos en nube tradicional y base de datos como servicio (DBaaS)

En un acuerdo DBaaS clásico, el proveedor mantiene la infraestructura física y la base de datos, dejando al cliente administrar el contenido y la operación de la base de datos.

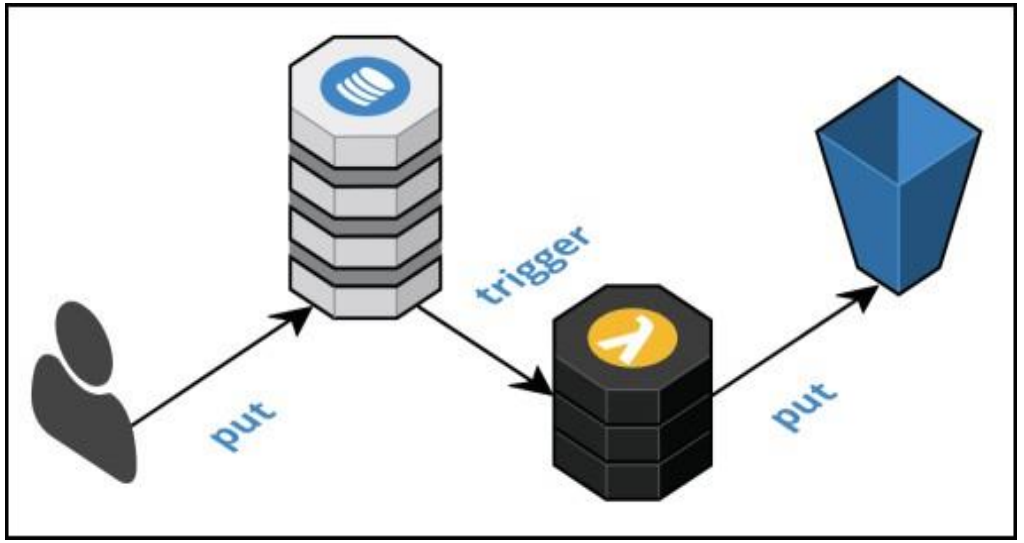
Que beneficios nos trae ?

- Escalabilidad instantánea
- Garantias de rendimiento
- Experiencia especializada

Ejemplo: desencadenante de base de datos nativo en la nube

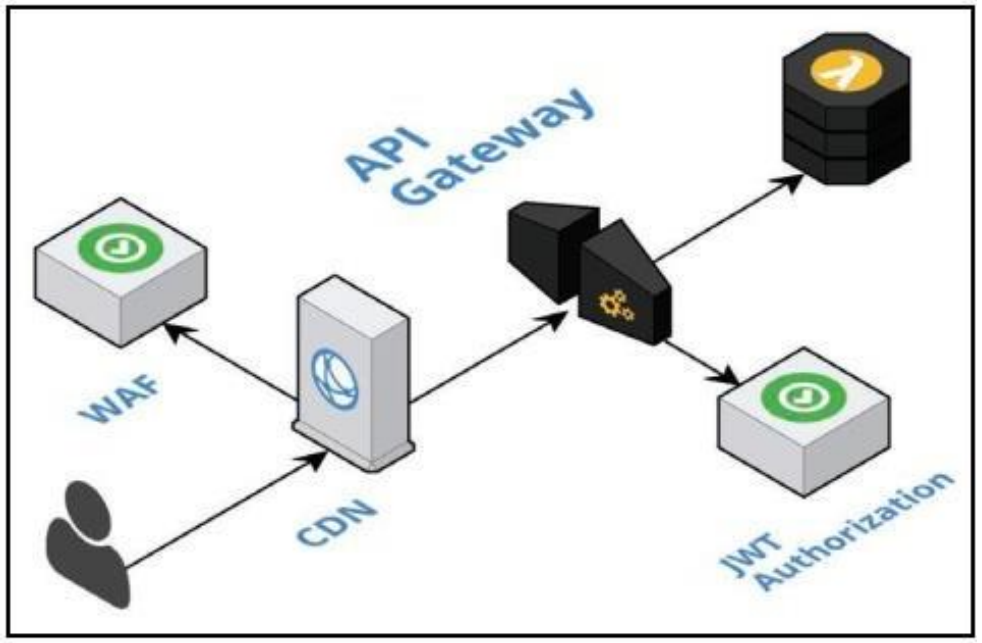
Este ejemplo típico muestra:

- Los bloques de construcción básicos que pueden permitir que múltiples bases de datos nativas de la nube dentro de un componente colaboren de forma asíncrona para crear una solución de persistencia cohesiva.
- Los datos **se colocan atómicamente en una tabla** de **DynamoDB** (almacén de documentos), esto, a su vez, activa una función que almacenará los datos de forma atómica en un depósito S3 (almacenamiento de blobs),
- Esto podría **desencadenar otra función.**
- Este patrón se puede repetir tantas veces como sea necesario hasta que los datos dentro del componente sean consistentes y,
- Fnalmente, **publique un evento en los componentes posteriores**, como veremos en el patrón de Abastecimiento de eventos.



API Gateway

Aproveche una puerta de enlace API totalmente gestionada para crear una barrera en los límites de un sistema nativo de la nube al impulsar las preocupaciones transversales, como la seguridad y el almacenamiento en caché, hasta el borde de la nube donde se absorbe algo de carga antes de ingresar al interior del sistema. .



Amazon WS WAF

Que es WAF y Cloud Front?

AWS WAF es un firewall de aplicaciones web que permite monitorizar las solicitudes HTTP y HTTPS que se reenvían a CloudFront y permite controlar quién obtiene acceso a su contenido.
En función de las condiciones que especifique, como las direcciones IP de las que provienen las solicitudes o los valores de las cadenas de consulta, CloudFront responde a las solicitudes con el contenido solicitado o con un código de estado HTTP 403 (Prohibido).
También puede configurar CloudFront para devolver una página de error personalizada cuando se bloquea una solicitud. Para obtener más información acerca de AWS WAF

Amazon CloudFront es un servicio web que acelera la distribución de contenido web estático y dinámico, como archivos .html, .css, .js y de imágenes, a los usuarios. CloudFront entrega el contenido a través de una red mundial de centros de datos denominados ubicaciones de borde

API Gateway

Contexto, Problema y Fuerzas

Los sistemas nativos de la nube están compuestos por componentes aislados limitados, que responden, son resistentes, elásticos y controlados por mensajes.

Toda la comunicación entre componentes se realiza de forma asíncrona a través de la transmisión de eventos.
Los componentes publican eventos en componentes secundarios y consumen eventos de componentes anteriores. **Nos esforzamos por limitar la comunicación sincrónica solo a las bases de datos nativas de la nube** dentro del componente y al servicio de transmisión de la nube nativa. Sin embargo, tarde o temprano, tenemos que implementar la comunicación sincrónica en los límites del sistema. En los límites, el sistema interactúa con los usuarios a través de aplicaciones móviles y aplicaciones web de una sola página y con sistemas externos a través de Open APIs.

Solución

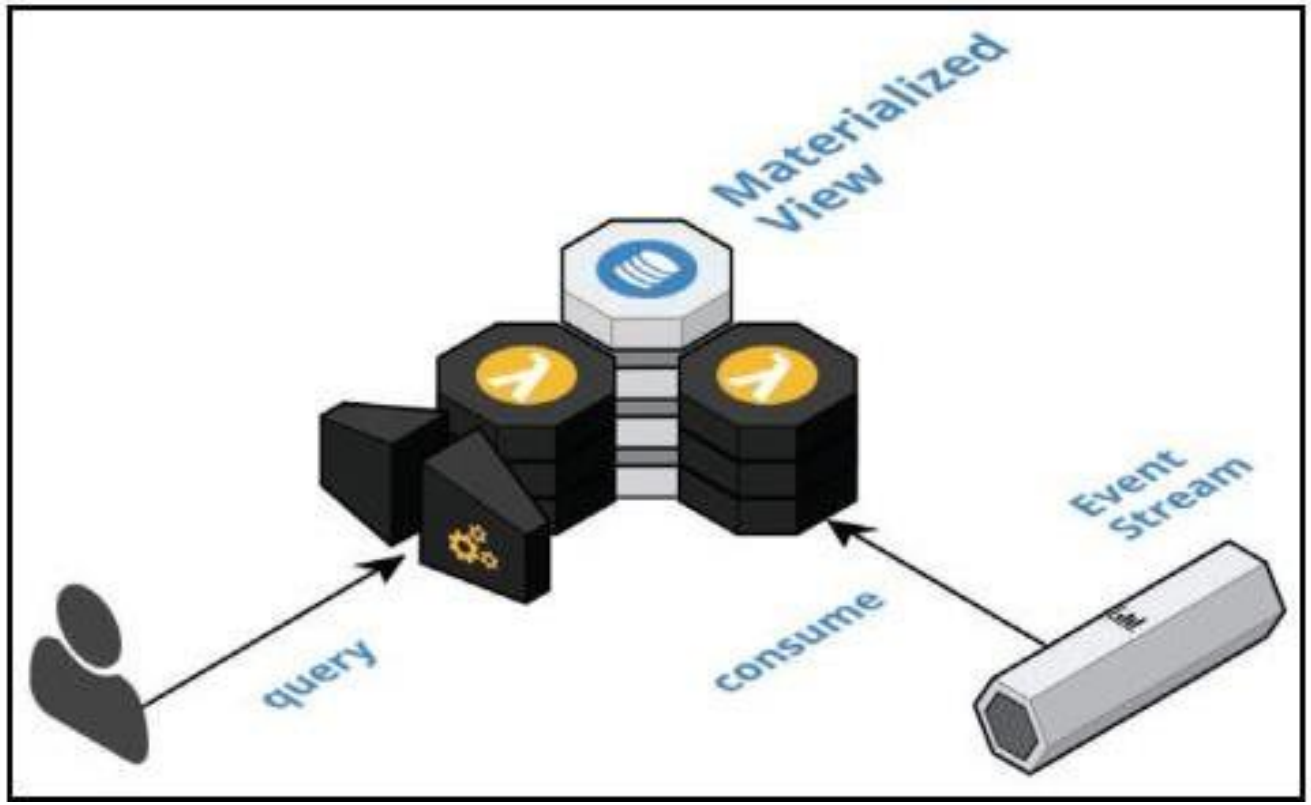
API Gateway

Aproveche el servicio de puerta de enlace API, para permitir que los equipos sean autosuficientes y controle su pila completa para que puedan centrarse en la propuesta de valor de sus componentes.

Estos servicios totalmente administrados brindan la capacidad de crear un perímetro seguro y escalable alrededor de los componentes de los límites**. La aceleración y la autorización de token de acceso son características esenciales.** Ofrezca sus puntos finales a través de la Red de entrega de contenido (CDN) de su proveedor de la nube para obtener beneficios adicionales de seguridad y rendimiento. **Agregue un Firewall de aplicaciones web (WAF)** para funciones de seguridad adicionales según sea necesario. Los equipos deben aprovechar la oferta de función como servicio de su proveedor de nube como el enfoque estándar de facto para implementar operaciones sincrónicas.

Command Query Responsibility Segregation (CQR

Consuma eventos de cambio de estado de componentes ascendentes y mantenga vistas materializadas que admiten consultas utilizadas dentro de un componente



Command Query Responsibility Segregation (CQRS)

Contexto, Problema y fuerzas

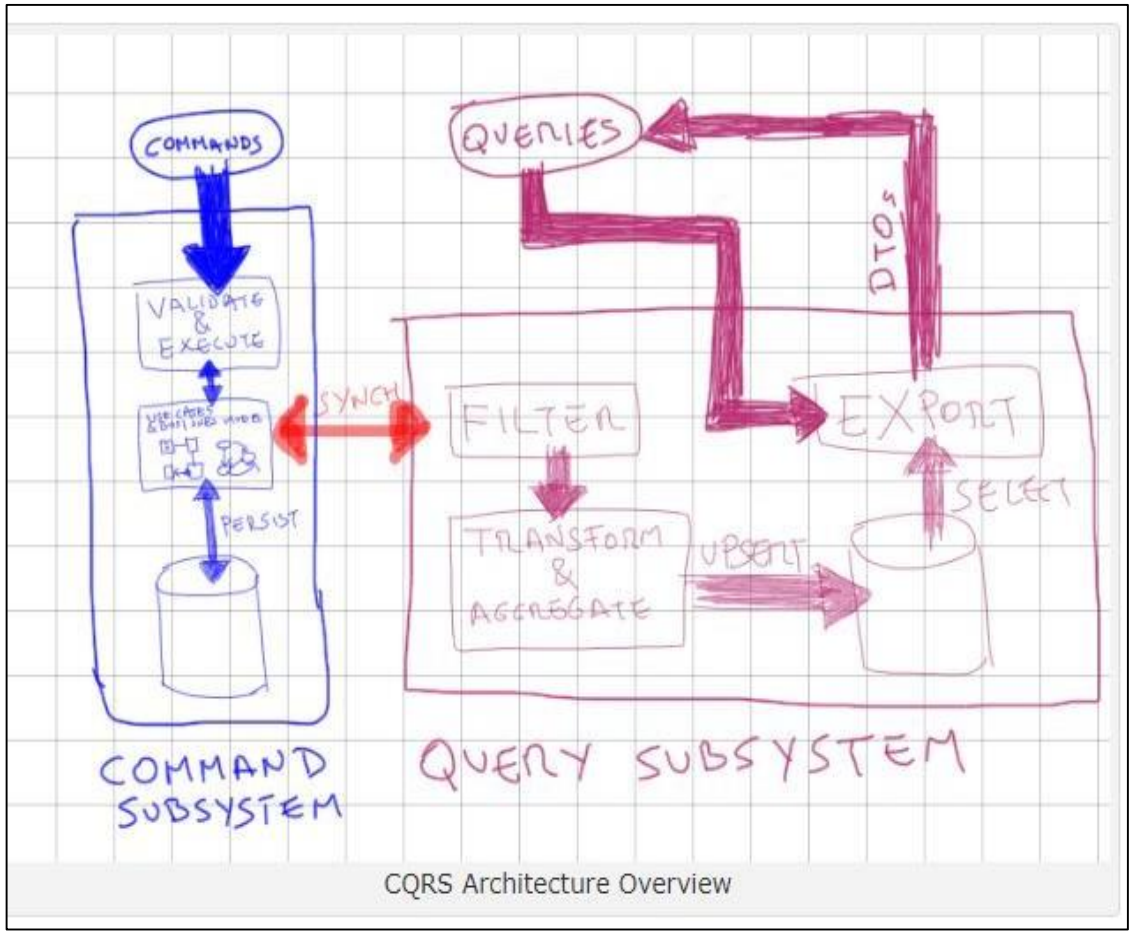
En una arquitectura tradicional, tenemos un único sistema que se encarga de realizar operaciones de negocio y nos permite consultar la información en la que se encuentra nuestro sistema.

Sin embargo, esto viola el **principio de única responsabilidad (SRP)**, ya que el mismo sistema se encarga de hacer dos cosas que en principio son muy distintas: **exponer operaciones que evolucionen el estado del sistema de forma consistente, y leer el estado del sistema.** Estas dos responsabilidades tienen requisitos muy diferentes desde distintos puntos de vista: funcional, escalabilidad, tiempo de respuesta, seguridad, criticidad, etc. Si nos tomamos en serio el principio de separación de responsabilidades, y somos partidarios de arquitecturas modulares en vez de sistemas monolíticos y centralizados, debemos buscar una arquitectura alternativa.

Command Query Responsibility Segregation (CQRS)

Contexto, Problema y fuerzas

(CQRS), es un estilo arquitectónico en el que tenemos dos subsistemas diferenciados, uno responsable de los comandos, y otro responsable de las consultas. **Por comando entendemos un petición por parte del usuario u otro sistema,** para realizar una operación de negocio, que evolucione el sistema de un estado a otro.
Cada uno de estos subsistemas tiene un diseño, modelo de información y mecanismo de persistencia diferente, optimizado para las tareas que deba afrontar. Normalmente el subsistema de consulta suele ser mucho más simple que el otro.



CQRS Architecture Overview

Command Query Responsibility Segregation (CQRS)

Solución

El subsistema de comandos, simplemente recibe peticiones de comandos, valida que éstos son consistentes con el estado actual del sistema, y si es así los ejecuta. Como resultado de la ejecución de un comando, el estado del sistema cambia, y ese cambio se comunica al subsistema de consultas mediante algún mecanismo de sincronización.

El subsistema de consulta recibe los cambios en el estado del sistema mediante el mecanismo de sincronización. **Durante la etapa de filtrado ignora los cambios en los que no está interesado.** En algunos mecanismo de sincronización esta etapa puede formar parte de la configuración del mismo, y no del código de aplicación. Después los cambios se pueden pasar por una etapa opcional donde se pueden transformar, añadirle información calculada y agregar información varios cambios.

Finalmente se actualiza de forma adecuada la base de datos. La ejecución de la consulta simplemente consiste en exportar los datos de la BBDD como DTOs y serializarlos en un formato adecuado.

.

Command Query Responsibility Segregation (CQRS) - Ejemplo

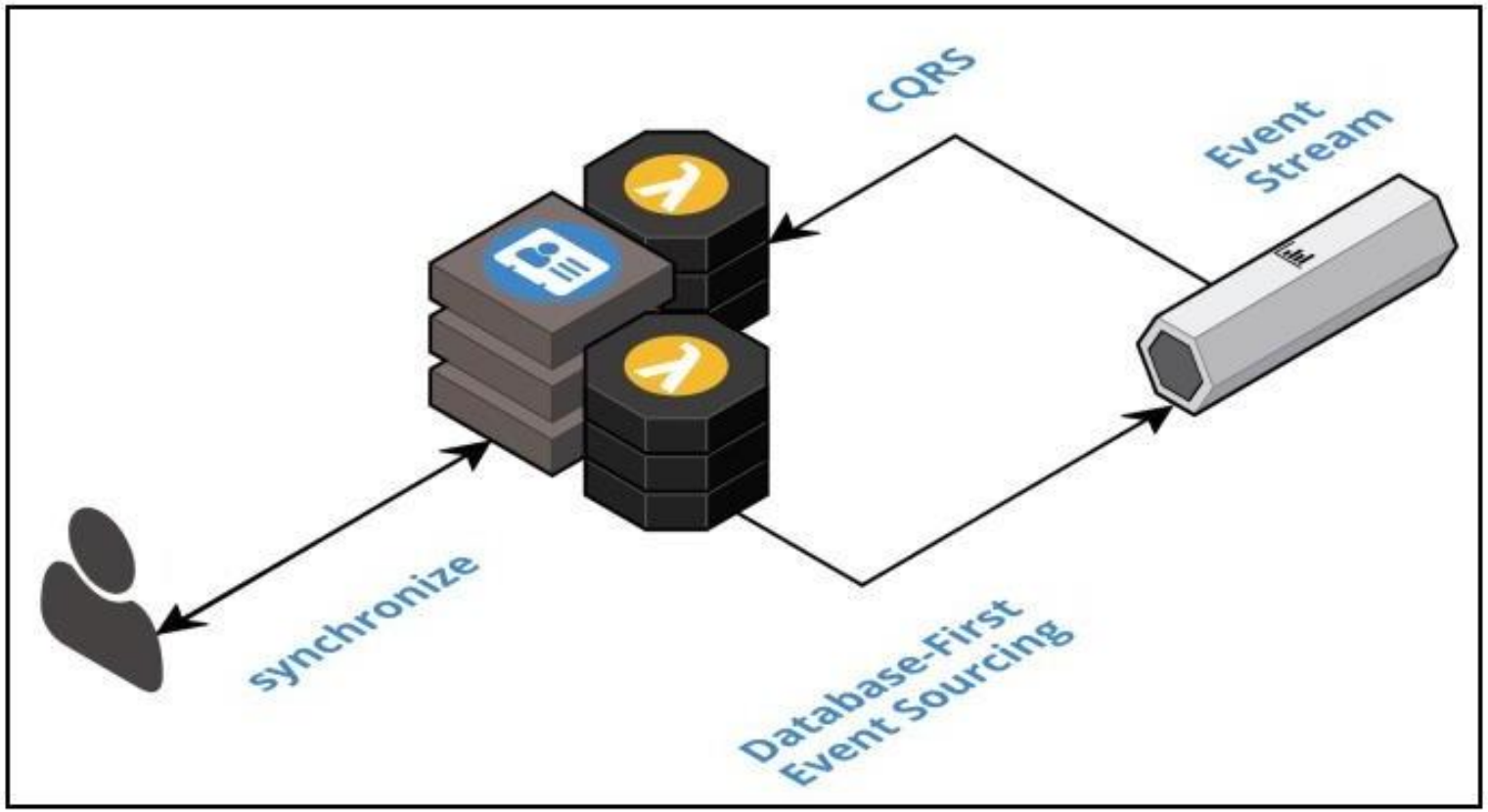
Ejemplo: inverse oplock

Necesitamos almacenar la última versión de un conjunto seleccionado de campos de una entidad de dominio específica. También tenemos que dar cuenta de los eventos que llegan fuera de orden. En el lado de creación de la ecuación, normalmente debemos tener en cuenta que varios usuarios actualizaron el mismo registro simultáneamente y una transacción sobrescribió a otra.

Desde el punto de vista de una regla empresarial, es importante que los eventos se procesen exactamente una vez; de lo contrario, pueden surgir problemas, como contar doble o no contar en absoluto. Sin embargo, nuestros sistemas nativos en la nube deben ser resistentes a las fallas y reintentar proactivamente para garantizar que no se eliminen mensajes. Desafortunadamente, esto significa que los mensajes pueden entregarse varias veces, por ejemplo, cuando un productor vuelve a publicar un evento o un procesador de flujo reintenta un lote que puede haber sido parcialmente procesado.

Offline-first database

Persista los datos del usuario en el almacenamiento local y sincronice con la nube cuando se conecte, de modo que los cambios del lado del cliente se publiquen como eventos y los cambios del lado de la nube se recuperen de las vistas materializadas



Offline-first database

Contexto, Problema y Fuerzas

En los sistemas nativos de la nube, la capa de presentación vive en el cliente y el dispositivo cliente es cada vez más móvil. Los usuarios tienen más de un dispositivo, incluidos teléfonos y tabletas, y, en menor medida, computadoras de escritorio tradicionales. **La capa de presentación debe comunicarse con el componente backend para enviar y recibir datos y realizar acciones**. Los usuarios móviles con frecuencia experimentan conectividad irregular, lo que aumenta la latencia y, a menudo, los deja completamente desconectados. El teorema de CAP establece que en presencia de una partición de red, uno tiene que elegir entre la consistencia y la disponibilidad. En el contexto de las aplicaciones modernas que enfrentan los consumidores, incluso un aumento temporal en la latencia se considera equivalente a una partición de red debido al costo de oportunidad de los clientes perdidos. Por lo tanto, se prefiere ampliamente elegir la disponibilidad en lugar de la coherencia y, por lo tanto, diseñar sistemas en torno a la consistencia eventual y la consistencia de la sesión. Teniendo en cuenta estas realidades, debemos diseñar una experiencia de usuario centrada en el cliente.

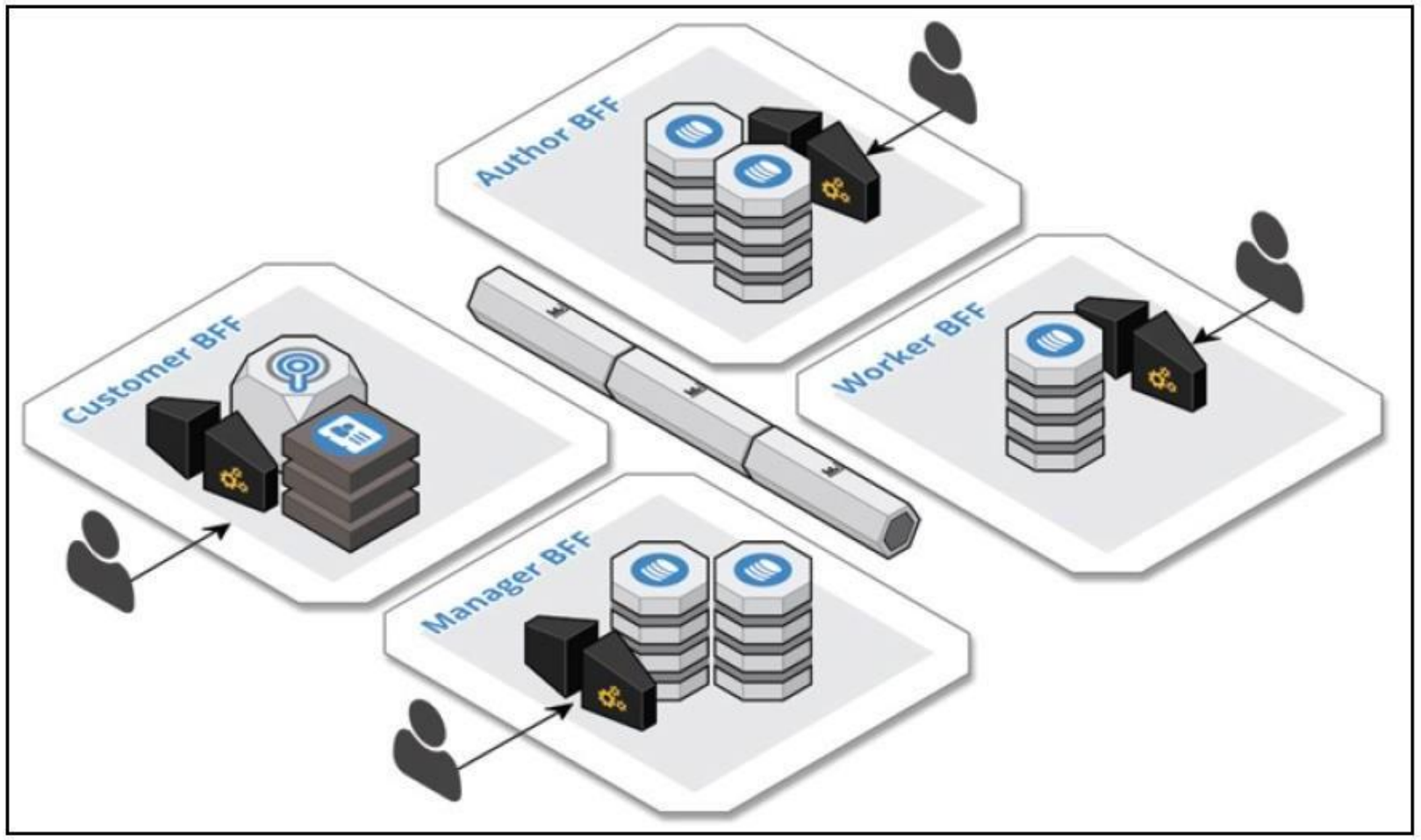
Offline-first database

Solución

Aproveche el primer servicio de base de datos sin conexión del proveedor de la nube para almacenar datos localmente y sincronizar con la nube y en todos los dispositivos. Calcule las vistas materializadas específicas del usuario, como se explica en el patrón CQRS, y sincronice las vistas con el dispositivo. Esto permite a los usuarios acceder siempre a la última información conocida. **Almacene acciones y entradas de usuario localmente y sincronice con la nube cuando esté conectado**. Esto permite a los usuarios continuar con sus intenciones incluso cuando están desconectados. La aplicación debe sincronizarse en acciones explícitas cuando sea posible, pero de lo contrario sincronizar en segundo plano cuando la conectividad es intermitente. La experiencia del usuario debe diseñarse de manera tal que el usuario entienda cuándo los datos pueden estar obsoletos o cuando las acciones estén pendientes debido a una conectividad limitada, e informarse cuando se resuelvan los problemas de conectividad. En el backend, el sistema aprovecha Database-First Event Sourcing cuando se produce la sincronización, para producir eventos basados en cambios de estado y acciones del usuario.

Backend For Frontend

Cree componentes de backend dedicados y autosuficientes para admitir las características de las aplicaciones de frontend centradas en el usuario.



Backend For Frontend

Contexto, problema, y fuerzas

Los sistemas reactivos, nativos de la nube, se componen de componentes aislados limitados, que proporcionan mamparos adecuados para permitir que los componentes sean sensibles, elásticos y resistentes.
Toda la comunicación entre componentes se realiza a través de la transmisión de eventos asíncronos y los componentes aprovechan las vistas materializadas para almacenar en caché los datos en sentido ascendente a nivel local.

Para aumentar la disponibilidad para una base de usuarios cada vez más móvil, aprovechamos una primera base de datos fuera de línea para almacenar datos en dispositivos y sincronizarnos con la nube.
La solución clásica es crear un backend genérico, que intente soportar todas las permutaciones. Sin embargo, esta solución tiende a implosionar, a medida que la lógica se enreda, ya que un solo equipo se esfuerza por cumplir con todos los requisitos de todos los demás equipos. **El componente único se convierte en el cuello de botella para todos los avances y un solo punto de falla, ya que no hay mamparos** funcionales entre todos los requisitos en competencia.

Backend For Frontend - Solucion

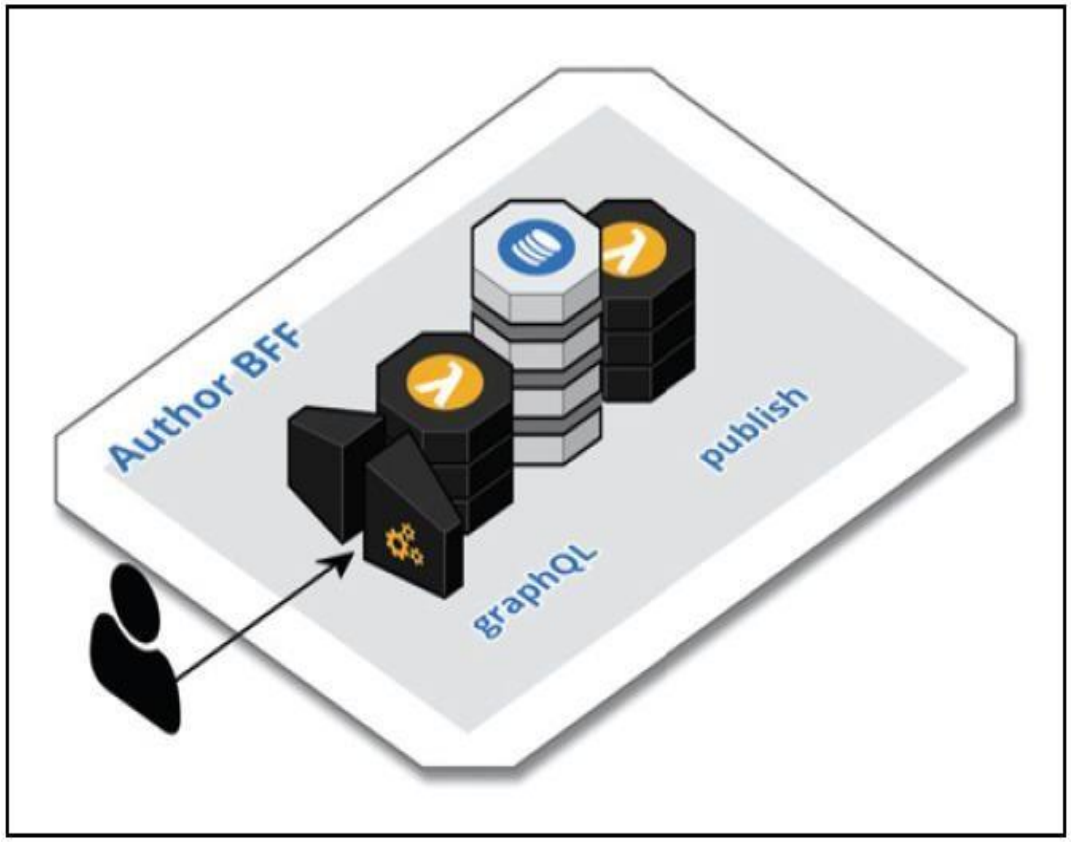
Si bien la aplicación de página única resulta efectiva para experiencias de usuario de canal único, este patrón proporciona resultados deficientes cuando se cuenta con varias experiencias de usuario en diferentes canales, lo cual a veces sobrecarga el navegador con la gestión de las interacciones con muchos servicios de respaldo basados en REST asincrónicos.

BFF representa una evolución en la que un servicio agregador de backend reduce la cantidad total de llamadas desde el navegador y, a su vez, maneja la mayor parte de la comunicación dentro de servicios de respaldo externos, devolviendo finalmente una solicitud más fácil de gestionar al navegador. Los equipos de frontend desarrollen e implementen su propio servicio de agregador de backend (BFF) para manejar todas las llamadas de servicio externas necesarias para su experiencia del usuario particular, que con frecuencia se diseña para un navegador, una plataforma móvil o un dispositivo de IoT específicos. El mismo equipo crea tanto la experiencia del usuario como el BFF,

Backend For Frontend

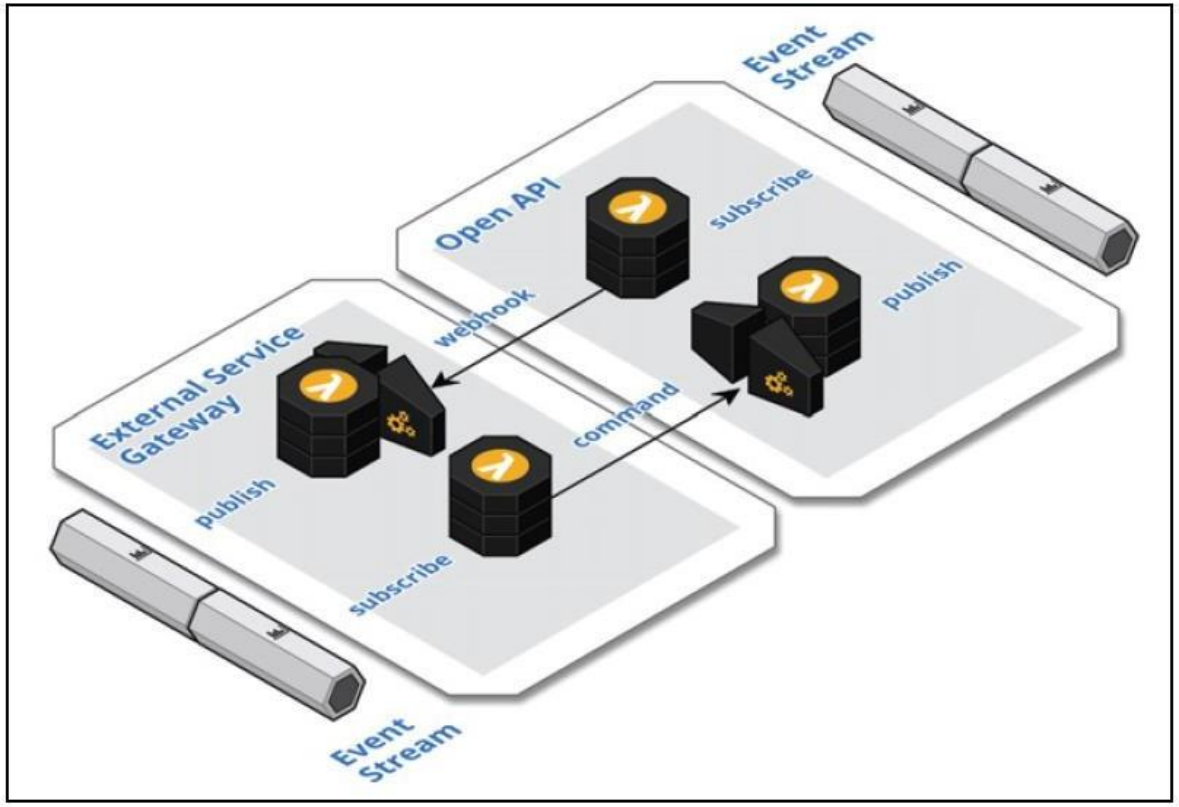
Ejemplo

Como se muestra en la siguiente figura, un componente BFF de Autorización generalmente consiste en dos de las tres API en el patrón de API Trilateral. Un BFF de creación tiene una API síncrona con comandos y consultas para crear y recuperar datos de entidad de dominio y una API asíncrona para publicar eventos a medida que cambia el estado de las entidades. El patrón de la puerta de enlace de la API se aprovecha para la interfaz síncrona, se sigue el patrón de base de datos nativa de la nube por componente para almacenar los datos en la persistencia políglota más apropiada, y la variante del patrón de base de datos de primer evento se utiliza para publicar eventos.



External Service Gateway

Intégrese con los sistemas externos mediante la encapsulación de la comunicación entre sistemas entrante y saliente dentro de un componente aislado limitado para proporcionar una capa anticorrupción que actúa como un puente para intercambiar eventos entre los sistemas.



External Service Gateway

Contexto, Problemas

Los sistemas a menudo necesitan integrarse con un sistema externo, como Stripe para el procesamiento de pagos, SendGrid para correos electrónicos transaccionales, AWS Cognito para la autenticación de usuarios o un sistema personalizado. Nuestros sistemas nativos de la nube están compuestos de componentes aislados limitados, que proporcionan mamparos adecuados para hacer que los componentes sean sensibles, elásticos y flexibles. El aislamiento se logra a través de la comunicación intercomponente asíncrona. Nuestro objetivo es eliminar todas las comunicaciones intercomponentes síncronas. Los sistemas externos proporcionan una API abierta para la comunicación entrante y saliente con otros sistemas..

External Service Gateway

Solucion

Tratar los sistemas externos como cualquier otro componente aislado acotado. Cree un componente para cada sistema externo que actúe como un puente bidireccional para transmitir eventos entre los sistemas

Outbound communication

El componente consume eventos internos e invoca los comandos apropiados en el sistema externo. Esta interacción sigue la variante Event-First del patrón de Event Sourcing y el sistema externo representa la base de datos que conservará los resultados de los comandos. Los eventos internos se transforman para cumplir con los requisitos del sistema externo. El componente puede emplear el Sourcing de eventos y los patrones de CQRS para capturar y almacenar en caché eventos adicionales según sea necesario para adornar la información que falta.

External Service Gateway

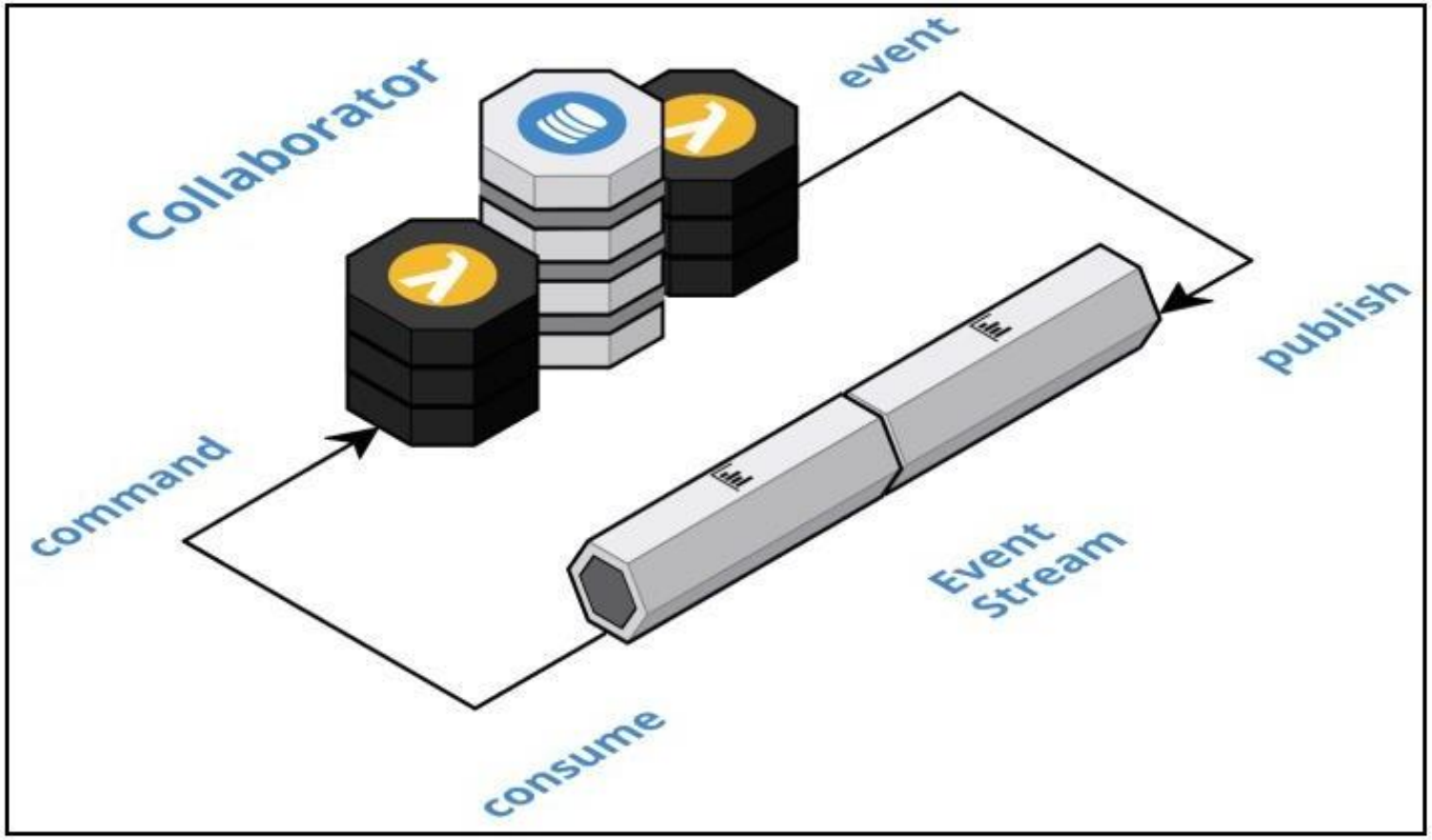
Comunicacion entrante

El componente consume eventos del sistema externo y los vuelve a publicar en la secuencia de eventos internos para delegar el procesamiento en los componentes posteriores. El evento interno contiene el contenido sin procesar del evento externo para auditoría y una versión transformada del contenido para consumo interno. Esta interacción sigue la variante Database-First del patrón de fuente de eventos y el sistema externo representa la base de datos que está emitiendo eventos de cambio de estado. El componente generalmente aprovecha el patrón de la puerta de enlace de la API para crear un servicio RESTful que se puede registrar con los webhooks del sistema externo. Alternativamente, cuando el proveedor de la nube ofrece el sistema externo, el componente registra funciones con el SPI del recurso de la nube.

PATRONES DE CONTROL

Event Collaboration

Publique eventos de dominio para desencadenar comandos en sentido descendente y crear una cadena reactiva de colaboración entre múltiples componentes.



Event Collaboration

Contexto, problemas, y fuerzas

Está construyendo un sistema reactivo, nativo de la nube, que se compone de componentes aislados limitados. El patrón de Segmentación de Responsabilidad de la Consulta de Comandos (CQRS) se aprovecha para replicar los datos de los componentes en sentido ascendente para evitar realizar una comunicación intercomponente síncrona no resistente para recuperar la información necesaria. Cada componente aprovecha los servicios en la nube de valor agregado para implementar una o más bases de datos nativas en la nube que son propiedad exclusiva de cada componente. Los datos de los componentes anteriores se sincronizan con estas bases de datos nativas de la nube para proporcionar acceso síncrono dentro de los componentes a los datos de alta disponibilidad. Este aislamiento permite que los componentes posteriores continúen funcionando incluso cuando sus dependencias ascendentes están experimentando una interrupción. Esto le permite a su equipo autosuficiente y de pila completa entregar innovación rápida y continuamente con la confianza de que un error honesto no afectará otros componentes.

Event Collaboration

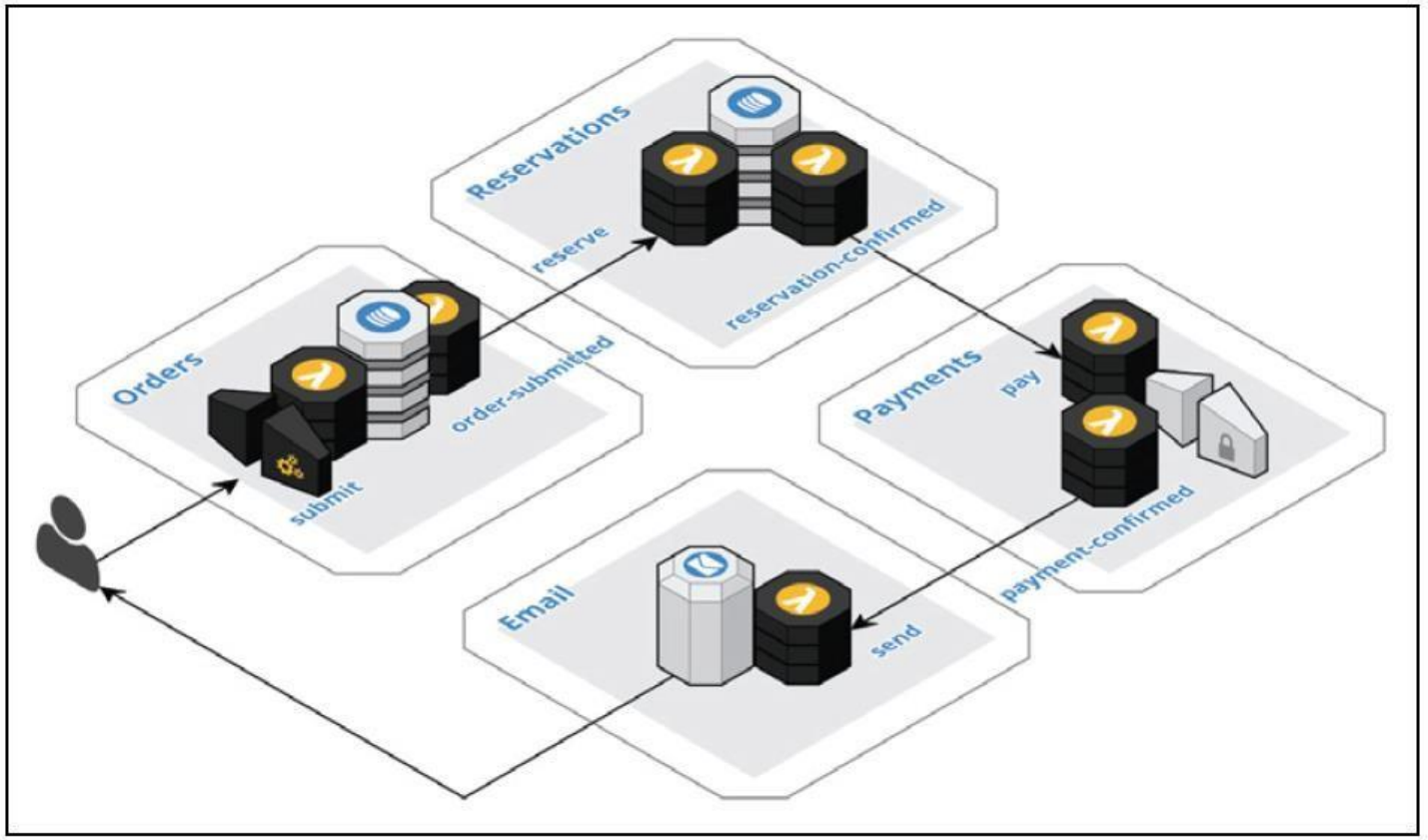
Solucion

Rediseñe la experiencia del usuario para que finalmente sea consistente. Reemplace la comunicación sincrónica entre componentes con la comunicación asíncrona entre componentes utilizando el flujo de eventos para publicar eventos que desencadenan comandos en sentido descendente. Los componentes ascendentes aplican la variante Database-First del patrón de fuente de eventos para publicar eventos de dominio que reflejen cambios en su estado. Los componentes en sentido descendente reaccionan a los eventos de dominio ejecutando un comando y publicando un evento de dominio para reflejar el resultado del comando

Event Collaboration

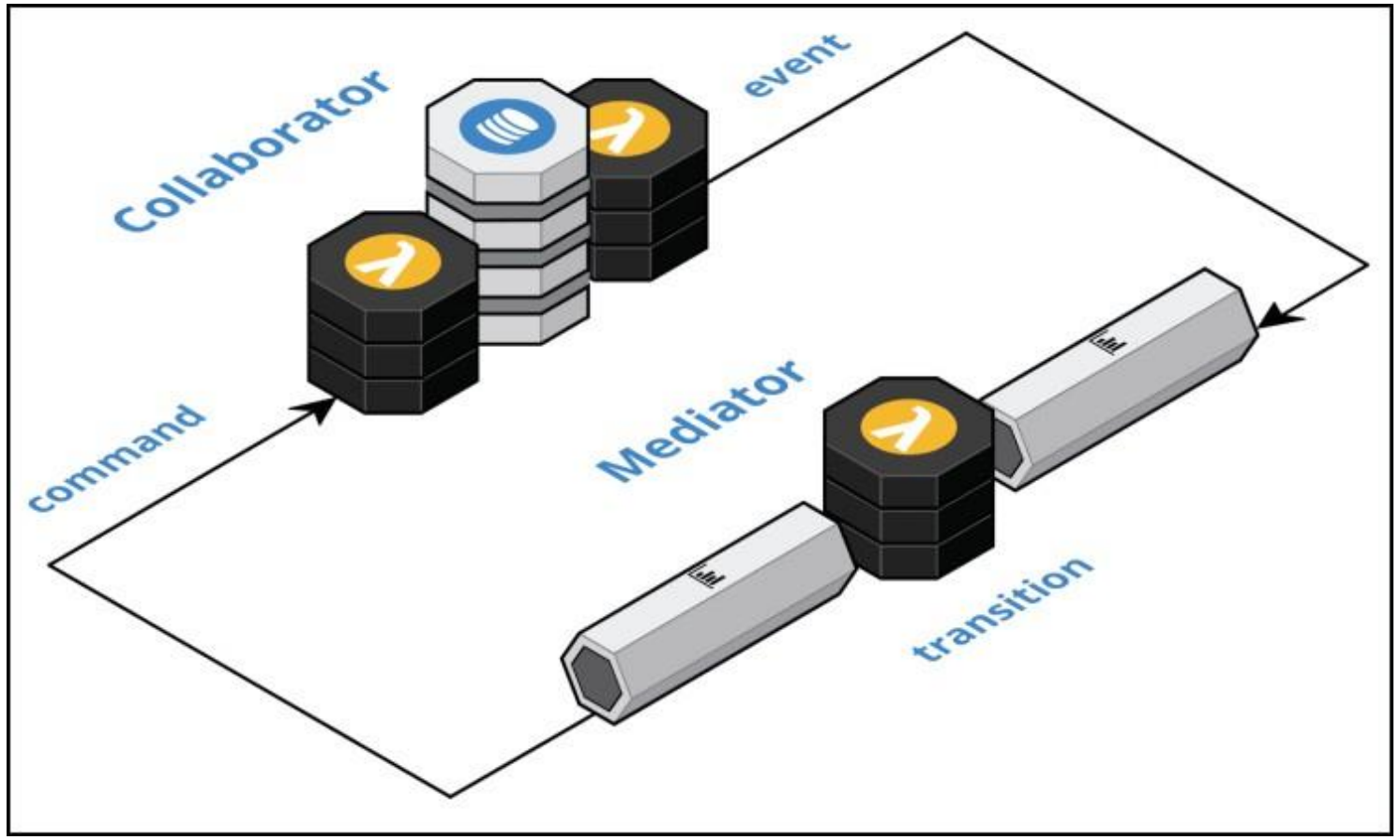
Ejemplo:

Este ejemplo, como se muestra en el siguiente diagrama, muestra la coreografía de las interacciones de múltiples componentes utilizando el patrón de colaboración de eventos. Los componentes están colaborando para implementar un proceso de negocios para ordenar un producto que debe reservarse con anticipación, como un boleto para un juego o una reserva de una aerolínea. El cliente completa y envía el pedido, y luego se debe confirmar la reserva, luego se debe cargar la tarjeta de crédito del cliente y, finalmente, enviar una confirmación por correo electrónico al cliente.



Orquestacion de Eventos

Aproveche un componente mediador para organizar la colaboración entre componentes sin acoplamiento de tipo de evento.



Orquestacion de Eventos

Contexto, problema y fuerzas

Está construyendo un sistema reactivo, nativo de la nube, que se compone de componentes aislados limitados. Está empleando los patrones de base de datos nativos en la nube por componente y transmisión de eventos, así como los patrones de abastecimiento de eventos y CQRS para garantizar que tenga los mamparos adecuados en su lugar para hacer que sus componentes sean sensibles, elásticos y flexibles. Esto ha aumentado la confianza de los equipos para ofrecer innovación y el sistema está creciendo rápidamente. Ha estado utilizando con éxito la colaboración de eventos sin formato para coreografiar los comportamientos de larga ejecución del sistema, pero con la complejidad cada vez mayor del sistema, está empezando a superar este enfoque. La arquitectura dirigida por eventos tiene la ventaja de separar a los productores de eventos específicos de los consumidores específicos. Los productores emiten eventos sin conocimiento de qué componentes consumirán los eventos y los consumidores capturarán eventos sin saber qué componente produjo el evento. Los eventos también resuelven los problemas de disponibilidad de la comunicación síncrona, porque los eventos son prod.

Orquestacion de Eventos

Solucion

Cree un componente para que cada proceso de negocio actúe como un mediador entre los componentes del colaborador y organice el flujo de colaboración entre esos componentes. Cada componente define los eventos que consumirá y publicará independientemente de cualquier proceso empresarial. El mediador asigna y traduce los eventos publicados de componentes ascendentes a los eventos consumidos de componentes descendentes. Estas asignaciones y traducciones se encapsulan en el mediador como un conjunto de reglas, que definen las transiciones en el flujo de colaboración

Orquestacion de Eventos

- Ejemplo: Orquestación de Ordenes

- Este ejemplo, como se muestra en el siguiente diagrama, demuestra la mediación de las interacciones de múltiples componentes utilizando el patrón de orquestación de eventos. Los componentes se ensamblan para implementar un proceso de negocios para ordenar un producto que debe reservarse con anticipación, como un boleto para un juego o una reserva de una aerolínea. El cliente completa y envía el pedido, luego se debe confirmar la reserva, luego se debe cargar la tarjeta de crédito del cliente y, finalmente, enviar una confirmación por correo electrónico al cliente. Este ejemplo se basa en el ejemplo presentado en el patrón de colaboración de eventos. Los componentes individuales se modifican para que sean completamente independientes y se agrega un componente de mediador para realizar la orquestación

Orquestacion de Eventos

Ejemplo: Orquestación de Ordenes