Características arquitectónicas

IIC2173 - Arquitectura de sistemas de software

Departamento de Ciencia de la Computación

Escuela de Ingeniería – PUC

Nicolás Acosta – Gerardo Crot



Repaso

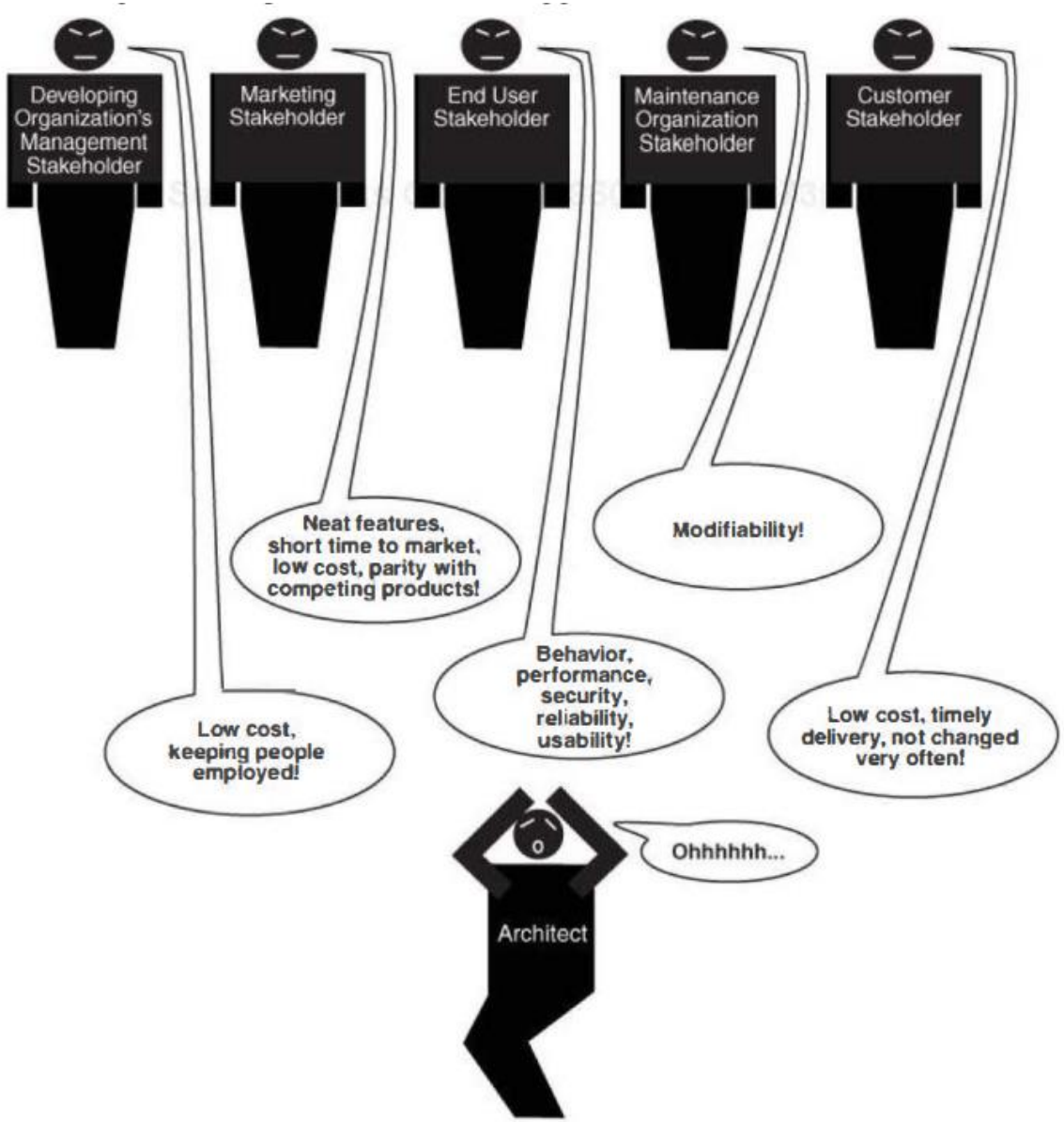
- Analogía constructiva
- Cohesión
- Acoplamiento



Métodos de identificación de componentes



- Partición dominio vs técnico:
  - Ejemplo de decisión: En capas vs Servicios
  - Usualmente el que conocemos solo es el técnico
- Modelo actor/acción (Desde RUP)
  - Usuario hace X sobre Y con rol Z
  - Ayuda a capturar requisitos
  - Muy cercano a historias de usuario
- *Event storming*
  - ¿Qué eventos gobiernan la app?
  - Ejemplo en proyecto
  - Buen alcance de complejidad
- Domain Driven Design: Veremos más adelante



Los requerimientos pueden llegar desde distintos involucrados…
¿qué atendemos primero?

Stakeholders

- Partes involucradas del proyecto que pueden verse afectadas por el resultado
- Peticiones ilimitadas
- Poca o nula consideración por otras áreas
- Tienen vistas parciales del negocio



¿Qué suelen pedir?

Table 3.1. Stakeholders for a System and Their Interests

Name	Description	Interest in Architecture
Analyst	Responsible for analyzing the architecture to make sure it meets certain critical quality attribute requirements. Analysts are often specialized; for instance, performance analysts, safety analysts, and security analysts may have well-defined positions in a project.	Analyzing satisfaction of quality attribute requirements of the system based on its architecture.
Architect	Responsible for the development of the architecture and its documentation. Focus and responsibility is on the system.	Negotiating and making tradeoffs among competing requirements and design approaches. A vessel for recording design decisions. Providing evidence that the architecture satisfies its requirements.
Business Manager	Responsible for the functioning of the business/organizational entity that owns the system. Includes managerial/executive responsibility, responsibility for defining business processes, etc.	Understanding the ability of the architecture to meet business goals.
Conformance Checker	Responsible for assuring conformance to standards and processes to provide confidence in a product's suitability.	Basis for conformance checking, for assurance that implementations have been faithful to the architectural prescriptions.
Customer	Pays for the system and ensures its delivery. The customer often speaks for or represents the end user, especially in a government acquisition context.	Assuring required functionality and quality will be delivered; gauging progress; estimating cost; and setting expectations for what will be delivered, when, and for how much.
Database Administrator	Involved in many aspects of the data stores, including database design, data analysis, data modeling and optimization, installation of database software, and monitoring and administration of database security.	Understanding how data is created, used, and updated by other architectural elements, and what properties the data and database must have for the overall system to meet its quality goals.
Deployer	Responsible for accepting the completed system from the development effort and deploying it, making it operational, and fulfilling its allocated business function.	Understanding the architectural elements that are delivered and to be installed at the customer or end user's site, and their overall responsibility toward system function.
Designer	Responsible for systems and/or software design downstream of the architecture, applying the architecture to meet specific requirements of the parts for which they are responsible.	Resolving resource contention and establishing performance and other kinds of runtime resource consumption budgets. Understanding how their part will communicate and interact with other parts of the system.

Bass, Kazman, Software architecture in practice

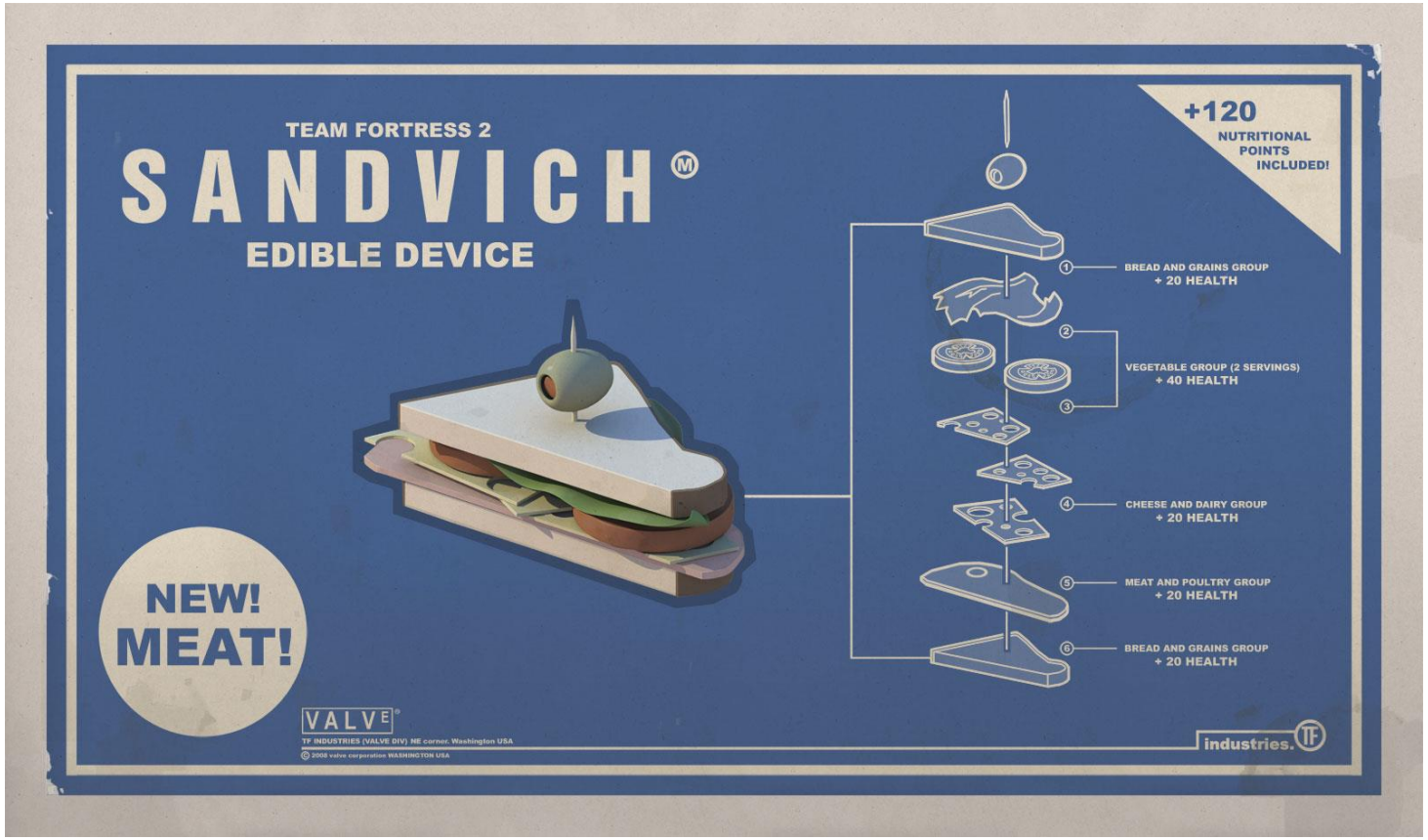
¿Qué suelen pedir?

Evaluator	Responsible for conducting a formal evaluation of the architecture (and its documentation) against some clearly defined criteria.	Evaluating the architecture's ability to deliver required behavior and quality attributes.
Implementer	Responsible for the development of specific elements according to designs, requirements, and the architecture.	Understanding inviolable constraints and exploitable freedoms on development activities.
Integrator	Responsible for taking individual components and integrating them, according to the architecture and system designs.	Producing integration plans and procedures, and locating the source of integration failures.
Maintainer	Responsible for fixing bugs and providing enhancements to the system throughout its life (including adaptation of the system for uses not originally envisioned).	Understanding the ramifications of a change.
Network Administrator	Responsible for the maintenance and oversight of computer hardware and software in a computer network. This may include the deployment, configuration, maintenance, and monitoring of network components.	Determining network loads during various use profiles, understanding uses of the network.
Product-Line Manager	Responsible for development of an entire family of products, all built using the same core assets (including the architecture).	Determining whether a potential new member of a product family is in or out of scope and, if out, by how much.
Project Manager	Responsible for planning, sequencing, scheduling, and allocating resources to develop software components and deliver components to integration and test activities.	Helping to set budget and schedule, gauging progress against established budget and schedule, identifying and resolving development-time resource contention.
Representative of External Systems	Responsible for managing a system with which this one must interoperate, and its interface with our system.	Defining the set of agreement between the systems.
System Engineer	Responsible for design and development of systems or system components in which software plays a role.	Assuring that the system environment provided for the software is sufficient.
Tester	Responsible for the (independent) test and verification of the system or its elements against the formal requirements and the architecture.	Creating tests based on the behavior and interaction of the software elements.
User	The actual end users of the system. There may be distinguished kinds of users, such as administrators, superusers, etc.	Users, in the role of reviewers, might use architecture documentation to check whether desired functionality is being delivered. Users might also use the documentation to understand what the major system elements are, which can aid them in emergency field maintenance.

Bass, Kazman, Software architecture in practice

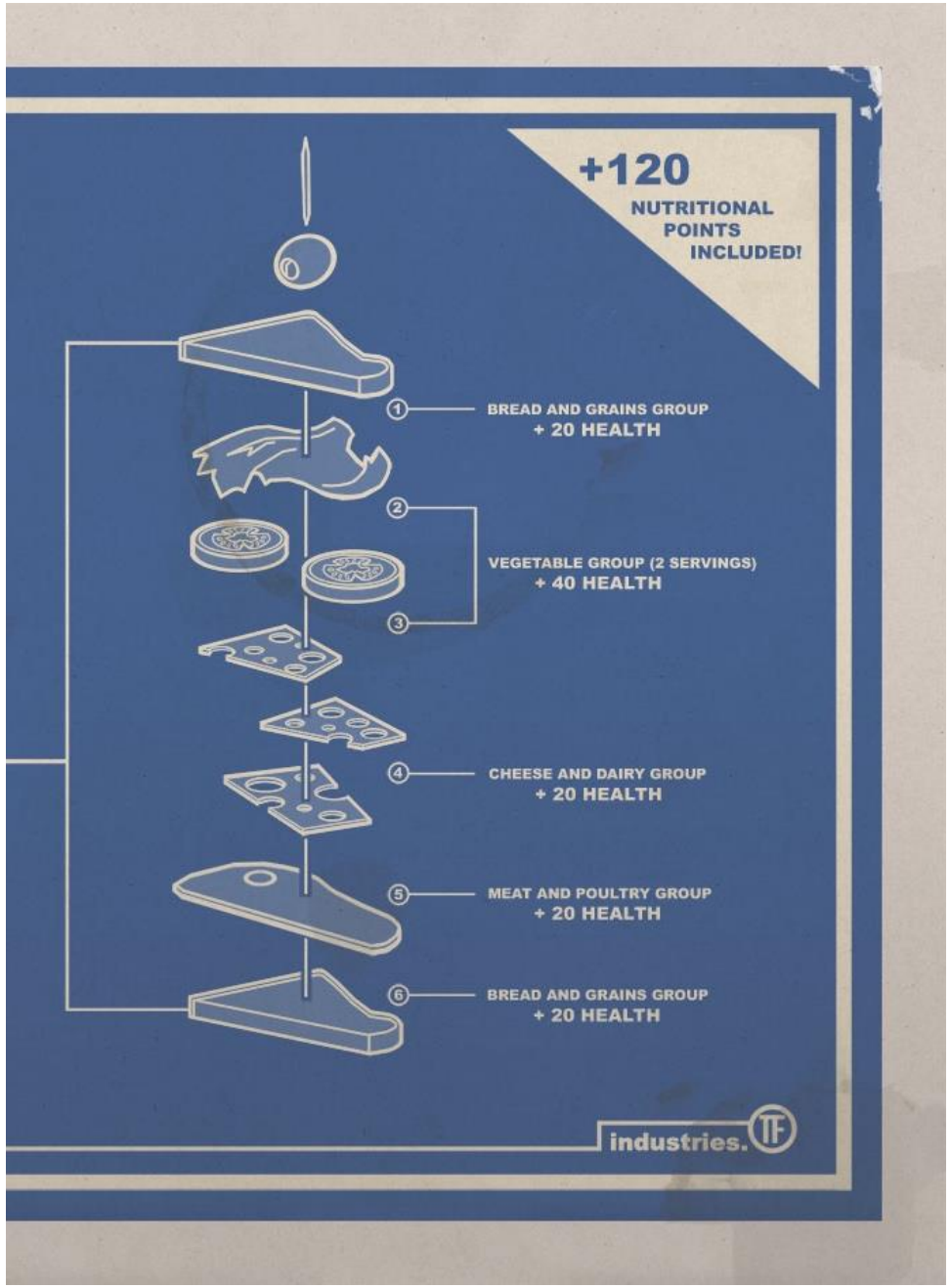
Atributos de calidad

- Conocidos también como:
  - NFR (*Non-Functional Requisites*)
  - Características arquitectónicas
  - Etc.
- Las propiedades no funcionales (atributos de calidad) son el resultado de las decisiones de arquitectura
- Los requisitos no funcionales, son los "valores" que se esperan de dichas propiedades



Atributos de calidad

- Pueden influir en la comprensión/tratamiento de la parte técnica
- Críticos para la operación y el éxito de la operación
- Su definición proviene de las metas de negocio y XP del equipo



Tipos

Tipo	Ejemplos de NFR
Estructural	Mantenibilidad, Flexibilidad/Extensibilidad, Interoperabilidad
Operacional	Performance, Escalabilidad, Confiabilidad, Disponibilidad
Transversal	Seguridad, Auth/auth, Usabilidad, Reusabilidad, Soporte, Time to market

¿Cómo decide el arquitecto cuáles atributos son clave?

- Tiempo
  - Somos un startup: en 6 meses nos compran
- $$$
  - Solo tenemos 5 devs
- Mercado
  - Mañana sale el Hooli Box Gavin Belson Signature edition
- Estándares
  - El equipo solo maneja C y Assembler
- Pasado/Presente/Futuro



Categorización de decisiones relevantes

- *Allocation of Responsabilites*
  - ¿Qué debe hacer el sistema a todo nivel?
  - Quién se hace/puede hacer cargo y de qué
- *Coordination model*
  - ¿Qué debe coordinarse con qué?
  - ¿Qué recursos consumo?
  - ¿A qué concurrencia y costo?
- *Data model*
  - Tipos
  - Propiedades
  - Metadata
  - Almacenamiento y Organización



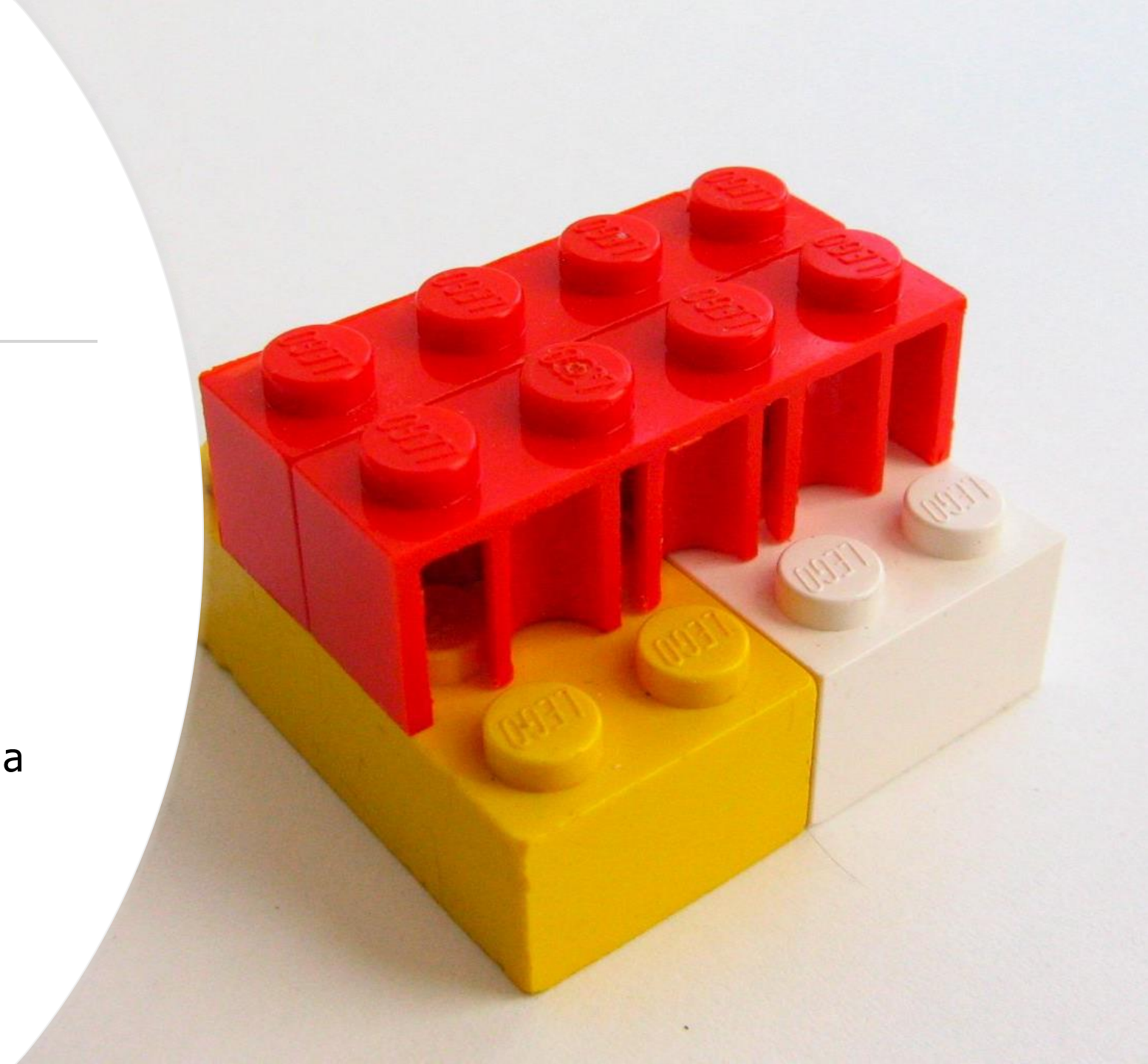
Categorización de decisiones relevantes

- *Mapping*
  - Entre elementos
  - Async/sync
  - Ley de Conway
- *Binding time decisions*
  - Mercado
  - Tiempo
  - Coordinación
  - Manejo de recursos
- Elección de tecnología
  - Stack
  - Soporte del equipo
  - Soporte operacional
  - Efectos secundarios



Requisitos estructurales

- Inquietudes sobre la estructura y el desarrollo del código
- Alto grado de recompensa, pero requieren un pago inicial
- Deuda técnica puede afectar mucho a futuro

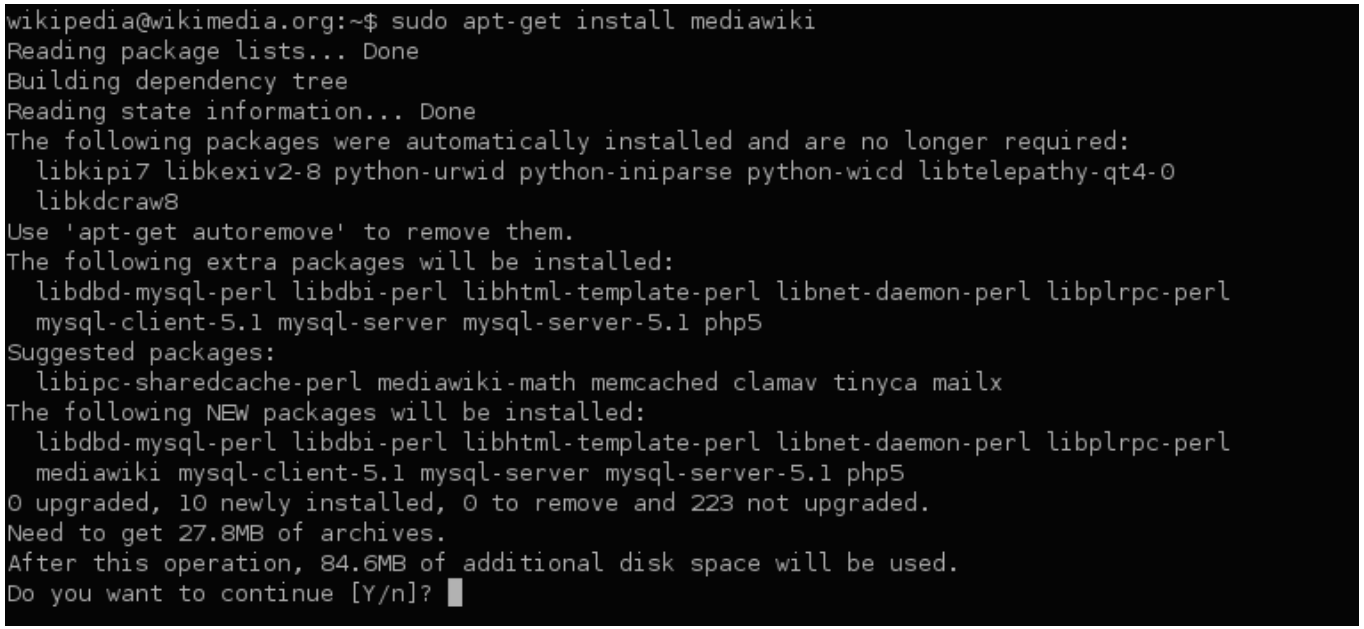


Mantenibilidad

Capacidad de poder solventar y reparar la operación de un sistema

- Operabilidad
- Simplicidad (En código y estructura)
- *Evolvability*

- Incluye *upgrades necesarias*, mejoras, *fixes*
- Tanto Técnico y práctico
- Ej.: Sistema APT en Linux

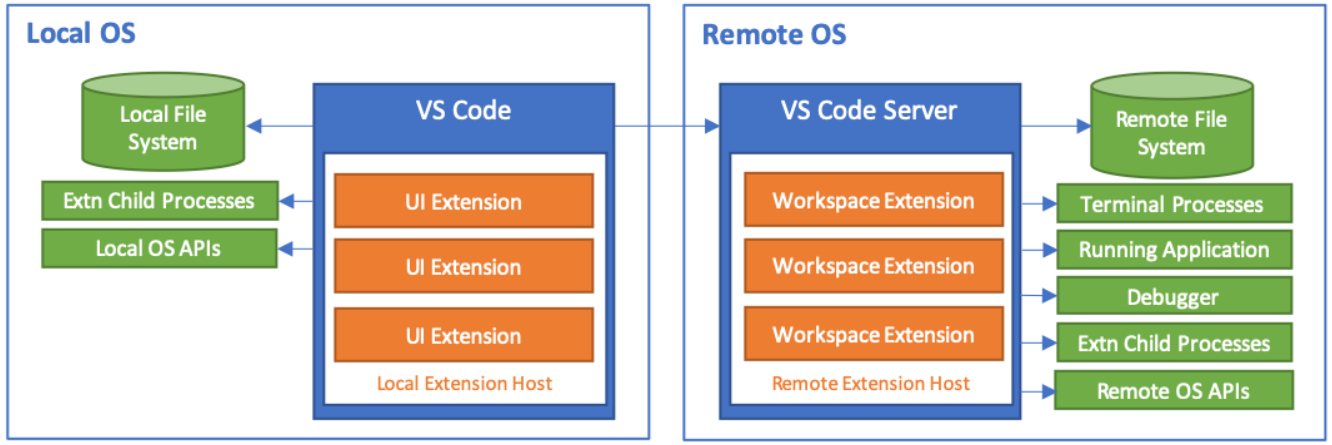


Flexibilidad / Extensibilidad

Capacidad de extender un sistema y agregar funcionalidad

- ¿Cuánto esfuerzo necesito?

- Requiere identificar escenarios de cambio probables
- Alta cohesión dentro los componentes y subsistemas débilmente acoplados facilitan el cambio
- Ej.: VScode y Extensiones, *Dependency injection*



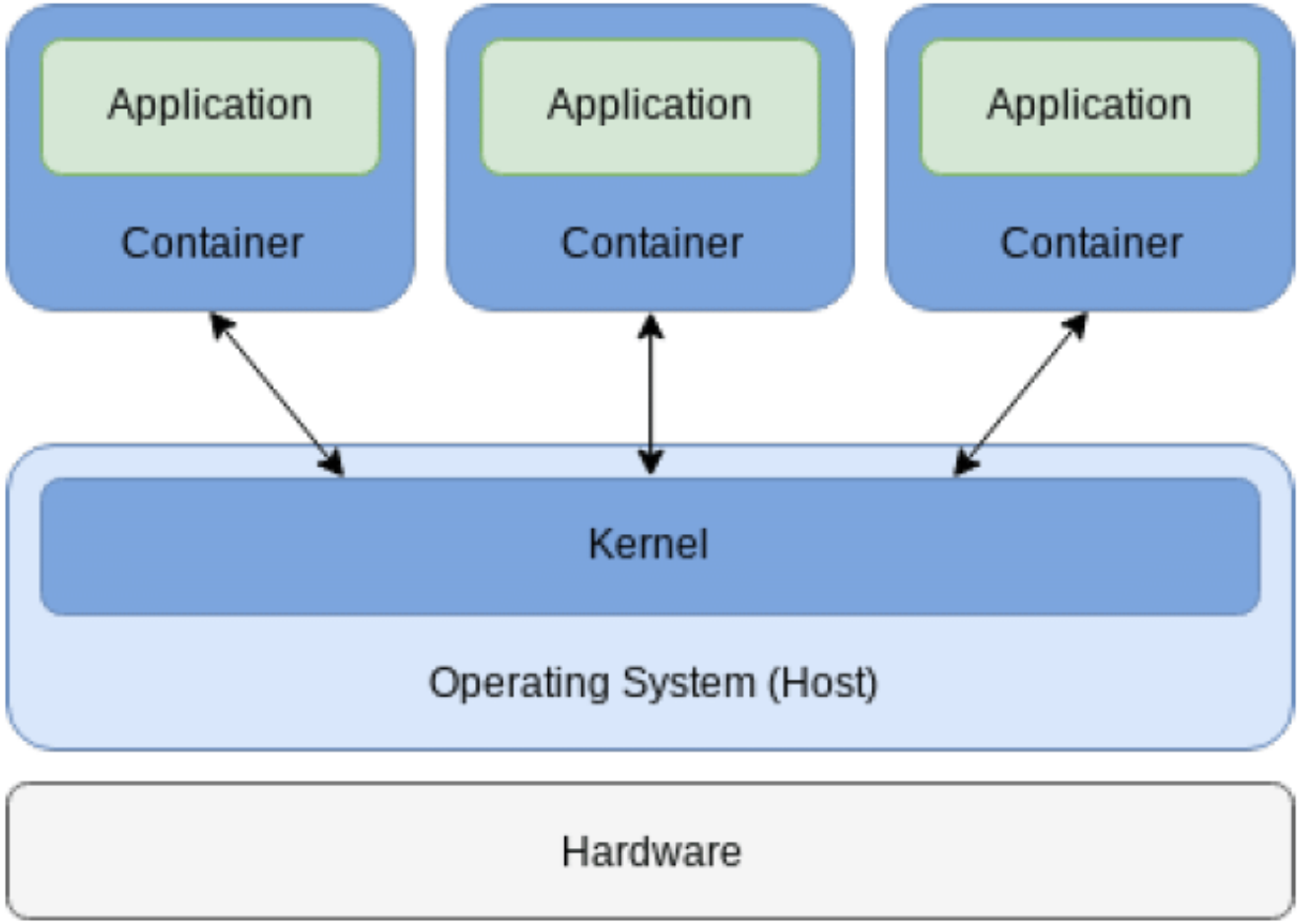
Portabilidad

Capacidad de funcionar en variedad de sistemas y plataformas con modificaciones mínimas.

- ¿Mi aplicación correrá siempre en un mismo tipo de servidor?

- La cohesión en los componentes

- Ej.: Flutter/React Native, *Containers*



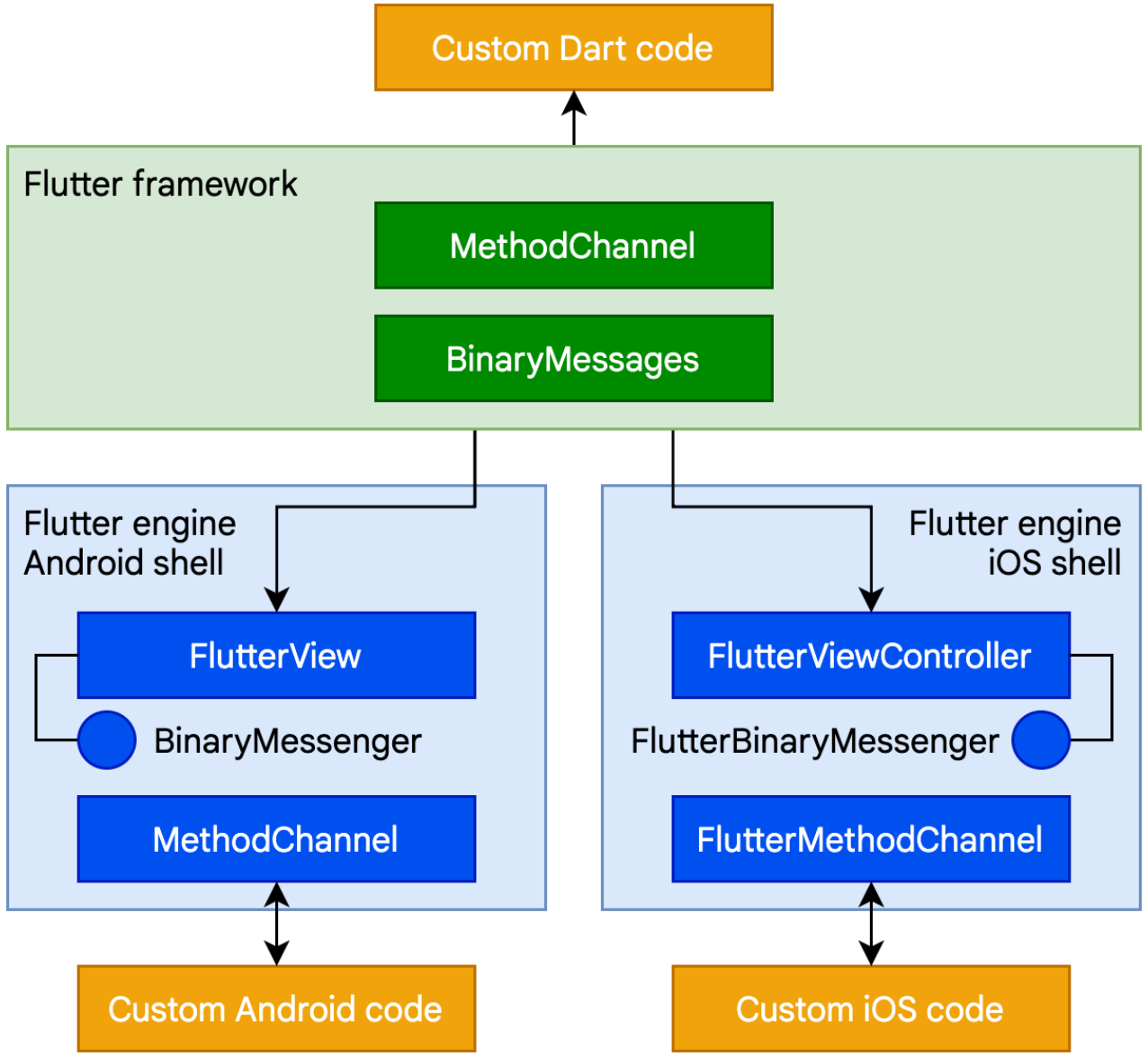
Portabilidad

Capacidad de funcionar en variedad de sistemas y plataformas con modificaciones mínimas.

- ¿Mi aplicación correrá siempre en un mismo tipo de servidor?

- La cohesión en los componentes

- Ej.: Flutter/React Native, *Containers*



Interoperabilidad / Integrabilidad

Capacidad de interactuar con otros sistemas de componentes e intercambiar información

- Usando interfaces/contratos
- APIs: Conjunto de funciones (parametrizadas) que controlan el acceso a los datos. Garantizan integridad de datos y reglas de negocio.

- Entendimiento a nivel sintáctico y semántico
  - ¿Cómo y qué decir?
- Se transfiere información relevante para la operación
  - *Systems of systems*, *Plugins*



Interoperabilidad / Integrabilidad

- Puede ofrecer y recibir data, o ambas
  - Ej.: Parte de una cadena de procesamiento
- Especial en la industria, en B2B, SaaS
- Otros ejemplos:
  - En salud: Conexión entre Fonasa e iMed
  - Gobiernos
  - Domótica
  - ERPs



Interoperabilidad: preguntas clave

- ¿Conocemos los sistemas que queremos que se integren?
  - ¿Nos interesa?
- ¿Podemos implementar una interfaz adecuada?
- ¿Podemos soportar la complejidad añadida?
- ¿Ofrecemos seguridad en nuestras interacciones?



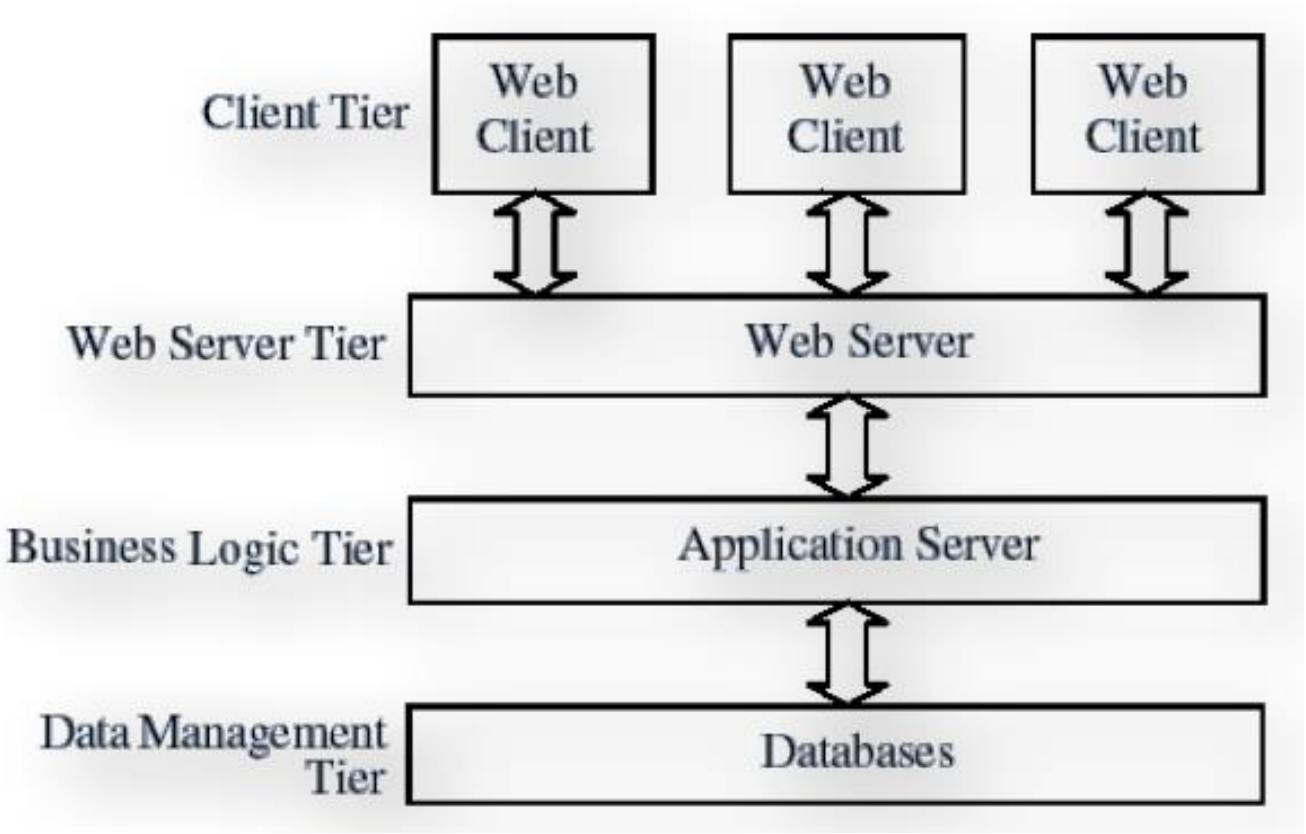
En qué fijarse en los NFR

- No son necesariamente excluyentes o no ligados, ni viceversa
- Algunos pueden ser vagos
  - Debe asignarse como responsabilidad del equipo
- Cómo lo busco:
  - En el curso: Relacionar con enunciado
  - En el trabajo: ¿qué priorizo?
- Recuerden que cada atributo es una promesa



Caso: Arquitectura de capas

- **Modularidad**: Separación de temas
- **Comunicación sincrónica** entre capas (request – wait – answer)
- **Implantación flexible**: capas independientes, distribuidas, heterogéneas, etc..
- Gran número de clientes, solicitudes concurrentes, requiere poco tiempo de proceso



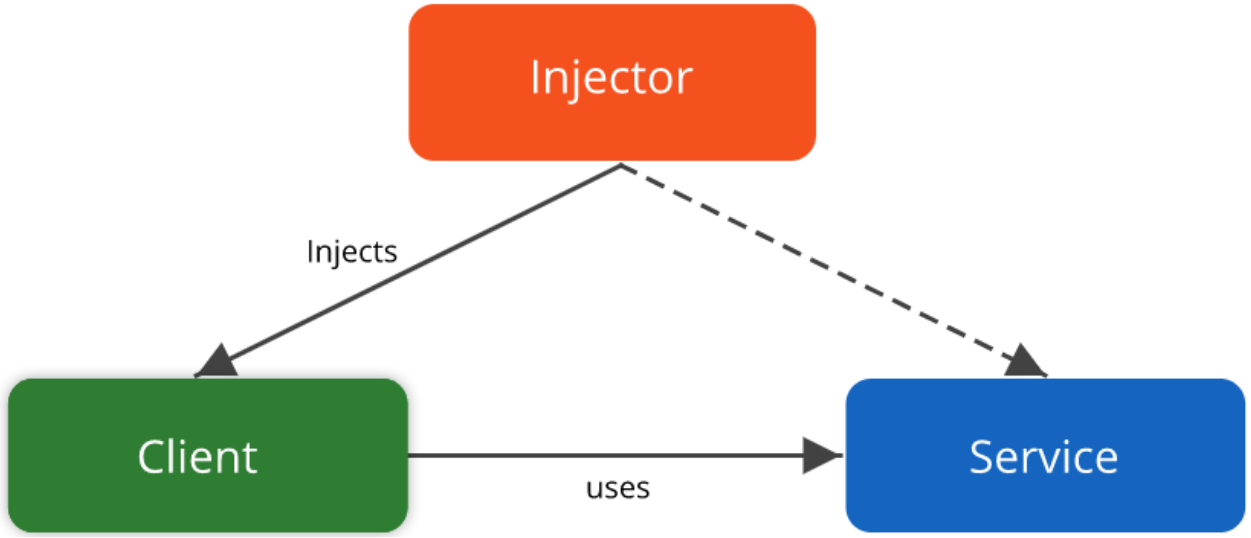
¿Qué atributos de calidad podrían priorizarse acá?

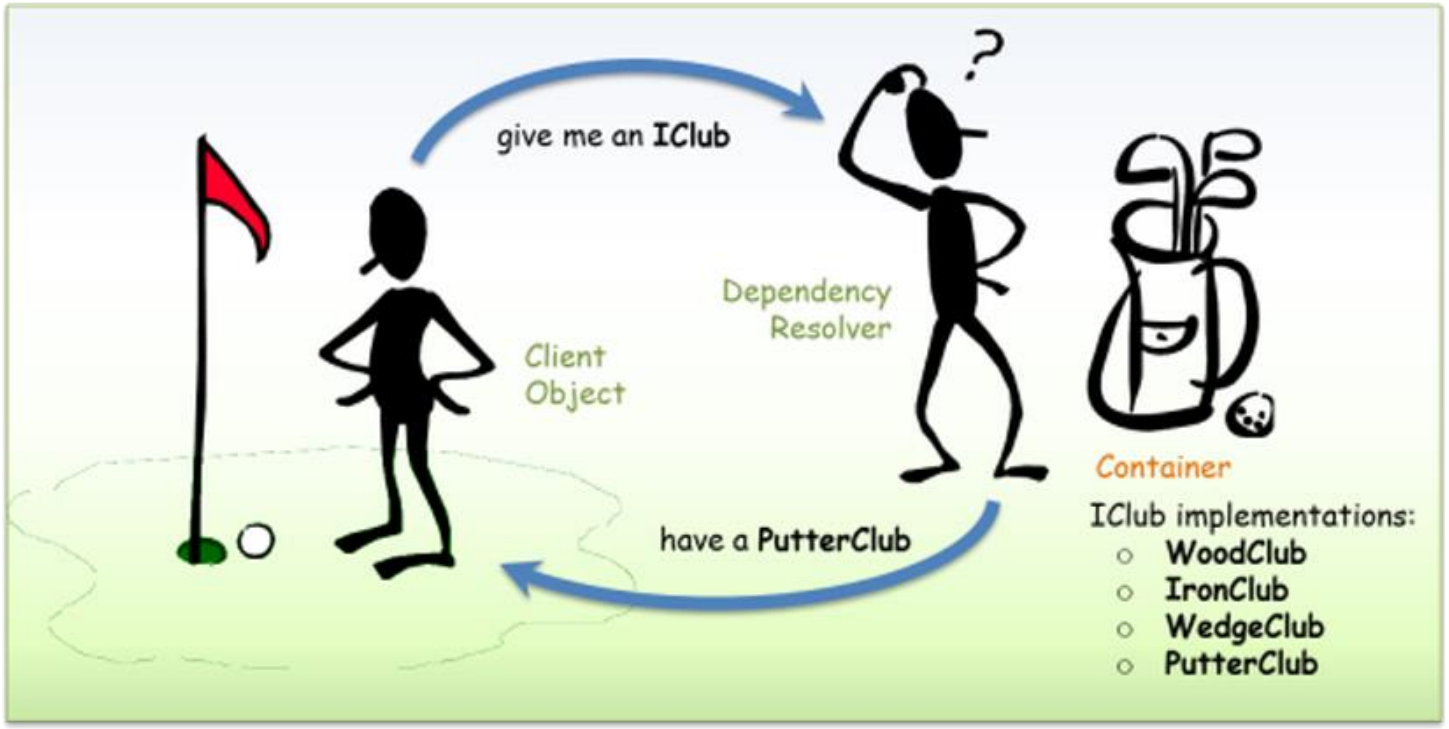
Caso: Arquitectura de capas

Atributo de Calidad	Ventajas
Disponibilidad	Réplica de servidores en cada capa, si uno falla, los otros responden. La QoS disminuye hasta restaurar el servidor fallado.
Manejo de fallas	La mayoría de los servidores web y de aplicación implementan una recuperación transparente (el cliente se redirecciona a un servidor replicado para responder a la solicitud).
Flexibilidad	La modularidad facilita el cambio, debido la encapsulación (presentacion, negocios y datos). El impacto del cambio al interior de una capa debería ser mínimo en las otras.
Desempeño	Alto desempeño mediante replicacción de software (threads concurrentes), velocidad de la conexión entre capas y la cantidad de datos a ser transferidos. Es recomendable minimizar las llamadas entre capas al procesar cada solicitud.
Escalabilidad	Como los servidores pueden replicarse, la arquitectura escala bien. En la práctica, la gestión de datos se convierte en el cuello de botella.

Caso: *Dependency Injector*

- Sistema para incorporar módulos de código y hacerlos reemplazables
- Inversión de control
- Composición sobre instanciamiento
- Promueve: Mantenibilidad / flexibilidad
- Degrada: Velocidad de desarrollo





https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/hands-on-labs/aspnet-mvc-4-dependency-injection

Atributos de calidad

IIC2173 - Arquitectura de sistemas de software

Departamento de Ciencia de la Computación

Escuela de Ingeniería – PUC

Nicolás Acosta – Gerardo Crot



Requisitos operacionales

- Influencian *runtime*
- Se manifiestan con la operación del negocio, basado en su comportamiento frente a situaciones
- Requieren varias iteraciones de ajuste



Performance / Desempeño

Carga de trabajo que una aplicación debe ejecutar en una unidad de tiempo, y/o *deadlines* que debe cumplir para una correcta operación.

- Carga de trabajo:
  - Transacciones
  - Mensajes
  - Lectura a la DB
  - Actualización de una DB distribuída en 2 fases



Performance: Métricas clave

- **Throughput – Salida:** Carga de trabajo en una unidad de tiempo (transacciones por segundo - tps)
  - Peak y promedio
- **Tiempo de respuesta:** Latencia que una aplicación exhibe al procesar una carga de trabajo.
  - Garantizado (límite) y promedio
- **Deadline – Plazo:** Ventana de tiempo para la compleción de una carga de trabajo
- Medida de capacidad



Performance: Suena demasiado bien…

- Siempre en consideración:
  - No siempre es crítico
  - Sobre priorizarlo puede perjudicar otros atributos
- Debe considerarse en contexto:
  - ¿Quizás 10 segundos si es aceptable?
- Sin métricas específicas, posee escaso valor
  - Casi idílico



Confiabilidad / Reliability

Una aplicación confiable protege su funcionamiento y datos y mantiene la operación

- Los sistemas son inherentemente inestables

- Tiene previsiones y estrategias para fallos
- Utiliza QoS en comunicaciones y otros
- Promete SLAs de diversos tipos (Durabilidad de datos, transmisión, resistencia a caídas)
- 99.99x%
- Cobra cada vez mas importancia con la AI

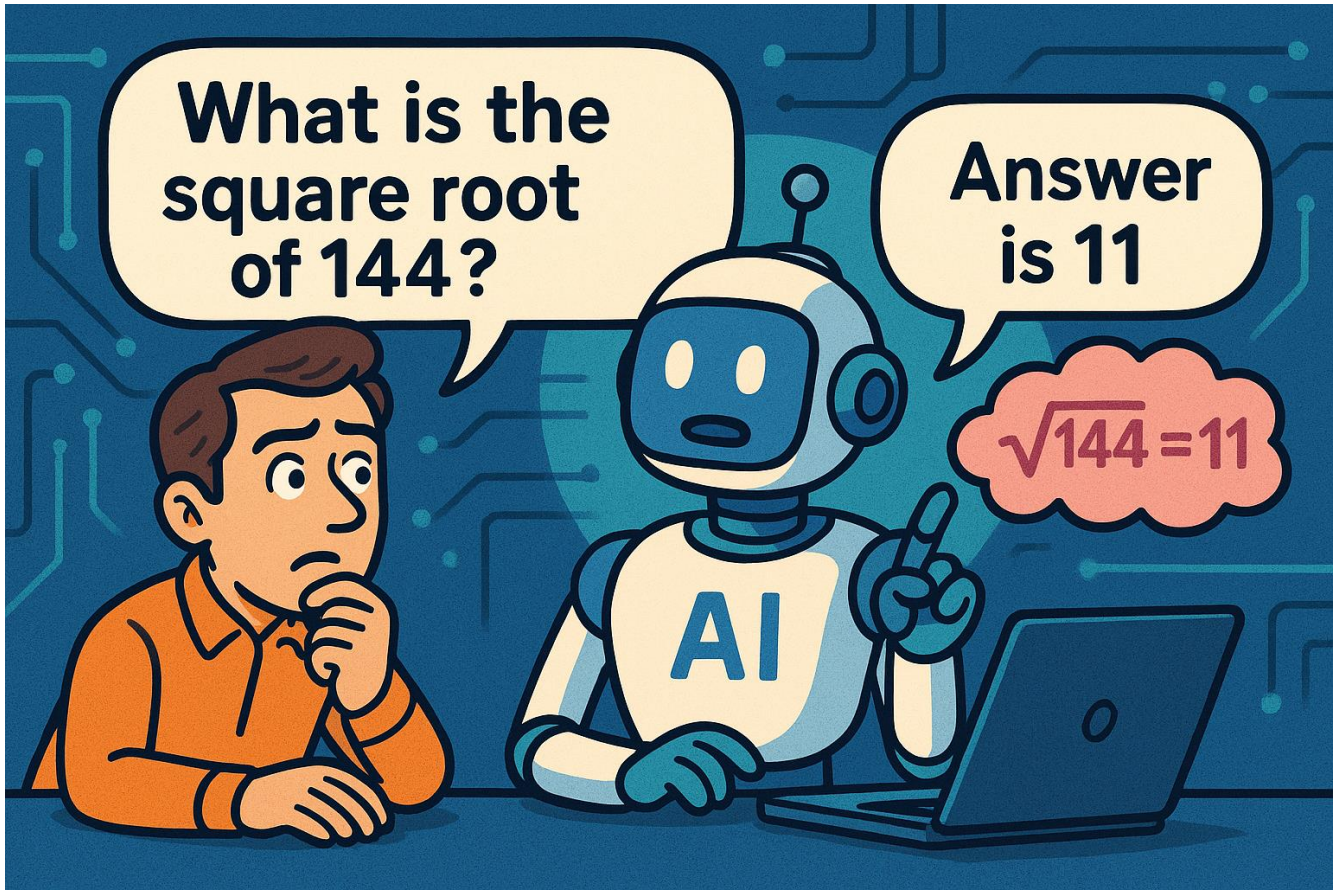


Factores en confiabilidad

Pros - Fomentan	Cons - Dificultan
Simplicidad + monitoreo Estándares y SLAs	Agilidad Visibilidad de valor Seguridad



Ejemplos de Confiabilidad:
Desert storm



Ejemplos de Confiabilidad: Alucinaciones AI

Disponibilidad

Puede mantener la operación el mayor tiempo posible (más 99.99x%)

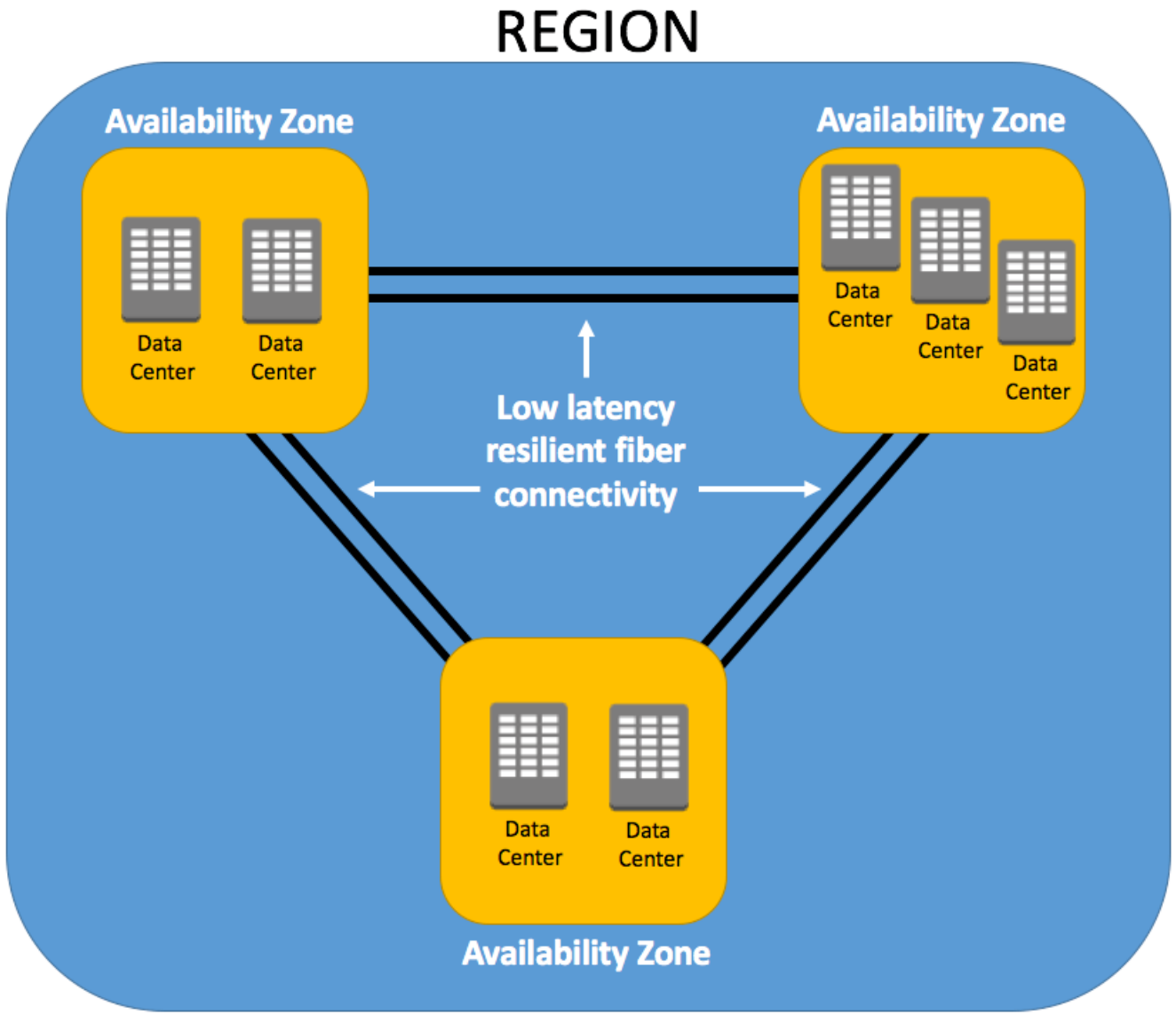
- Viene con un SLA de tiempo asociado
- Asociado a confiabilidad, derivado de otros requisitos
- Puede implicar un costo muy elevado
- Puede levantarse de caídas en sus componentes



¿Cómo lo logra AWS?

- 1 Región = Múltiples AZs (Min 3)
- 1 AZ = Clúster de Data Centers

- Tienen planes frente a desastres:
  - Lugares diferentes
  - Energía redundante
  - Conexiones redundantes



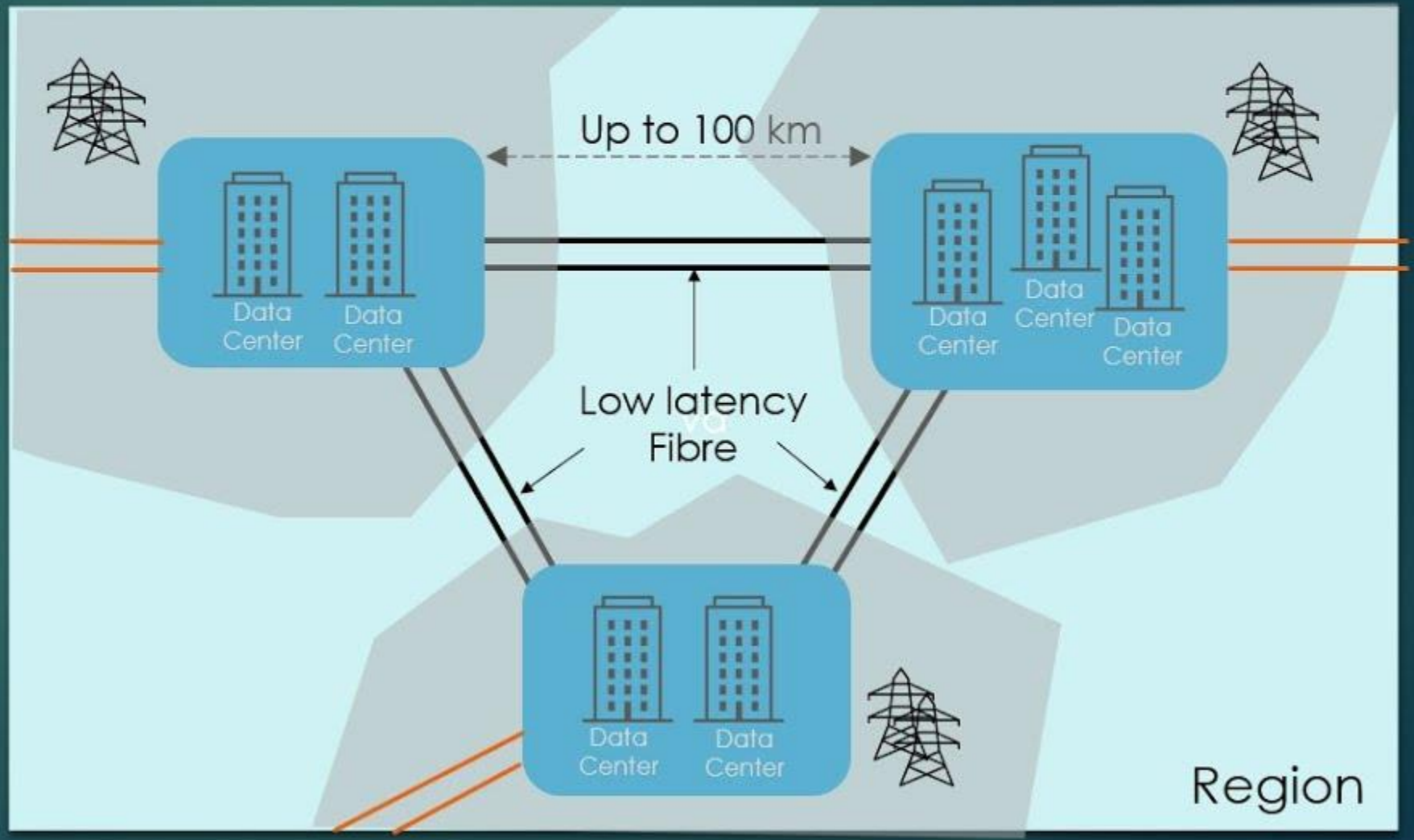
1 Region = Multiple AZs (Min 3)
1 AZ = Cluster of Data centres

EC2
Getting Started

Different floodplains (in most cases)

Redundant Power Supply

Redundant Network Connectivity



www.awstrainingcenter.com

bmc

The Nines of Availability

Availability percentages vs service downtime

Availability %	Downtime per year	Downtime per month	Downtime per week
90% (one nine)	36.5 days	72 hours	16.8 hours
99% (two nines)	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.9% (three nines)	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% (four nines)	52.56 minutes	4.32 minutes	1.01 minutes
99.999% (five nines)	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% (six nines)	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% (seven nines)	3.15 seconds	0.259 seconds	0.0605 seconds

https://www.bmc.com/blogs/service-availability-calculation-metrics/

bmc

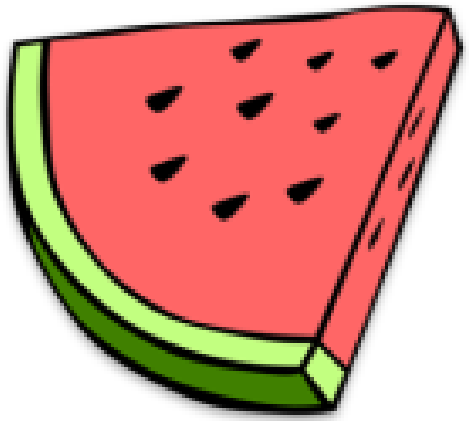
The Watermelon Effect

Your outputs (activities) meet defined targets, but the outcome (result) is not achieved.

Outputs:

Your reports are green.

Success!



Outcome:

Your customers aren't happy.

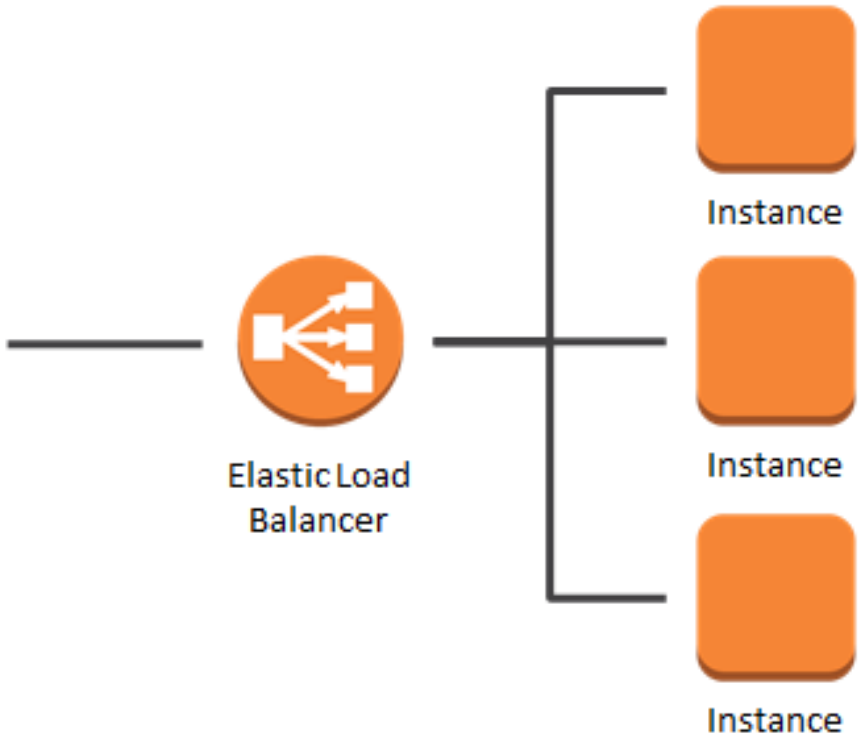
Fail.

https://www.bmc.com/blogs/service-availability-calculation-metrics/

Escalabilidad

Enfrentar aumento de demanda de forma funcional

- Pasar a 1/100/1M/1MM usuarios en el tiempo
- Posibles métodos:
  - *Load balancers*
  - *Brokers*
  - Asincronía
  - Estilos y patrones



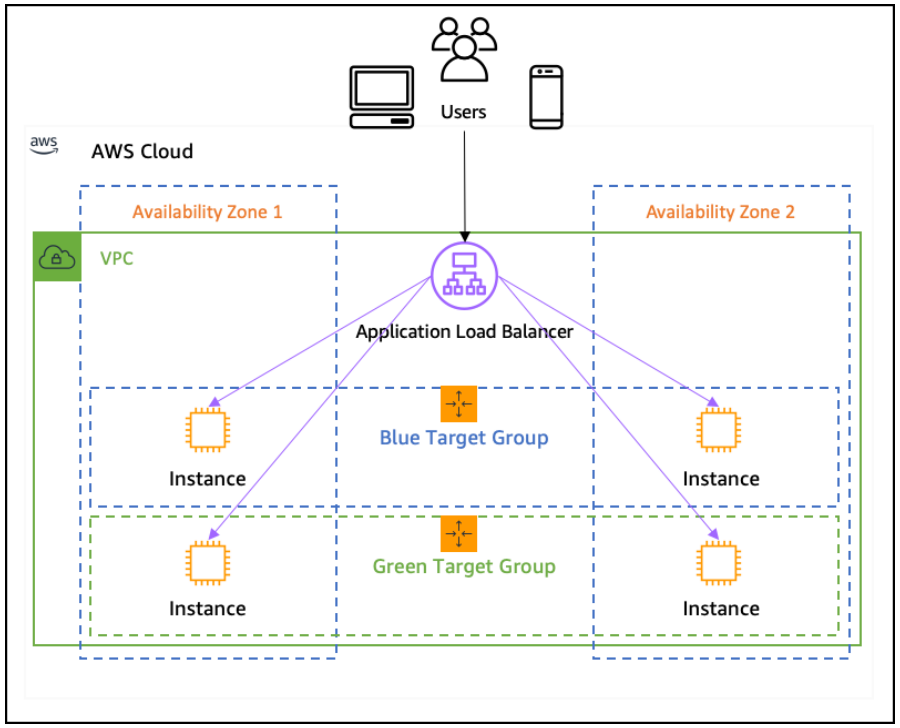
Escalabilidad: Métricas

- **Request – Cantidad de Solicitudes:** Multi-thread, balance de carga (repartir las solicitudes entre varios servidores/CPUs)
- **Cantidad de conexiones concurrentes**
- **Tamaño de datos:** Cantidad de datos que puede procesar la aplicación en cada transacción (incremento en el tamaño de los paquetes de datos)
- **Deployment – Implantación:** Cantidad de esfuerzo requerido en distribuir, actualizar, configurar una aplicación (incremento en cantidad de usuarios)



Escalabilidad: Costos

- Se eleva la complejidad
- Requiere esfuerzos de coordinación
- Puede requerir uso de asincronía



¿A quién le interesa qué?



Y a nivel de producto





Métricas de operación

Métrica	Definición	*Factor*
Accuracy	Grado en el cual un programa satisface su especificación y cumple los objetivos del **usuario**. El producto hace lo esperado?	**Trace-ability, completeness, consistency**
Reliability	Grado en el un programa cumpla sus funciones con precisión. **Cómo satisface los requisitos dado un periodo de tiempo?**	**Consistency, accuracy, error tolerance, simplicity**
Efficiency	Cantidad de recursos computacionales y código requerido para que el programa cumpla sus funciones. Utiliza los recursos de hardware bien?	**Execution efficiency, storage efficiency**
Integrity	Grado en el cual el acceso a datos o software por personas no autorizadas puede ser controlado. **Es seguro?**	**Access control, access unit**

Métrica	Definición	*Factor*
Usability	***Esfuerzo requerido para aprender, operar, ingresar datos, e interpretar la salida de un programa. Puedo manejarlo fácilmente?***	***User friendly, operability, training***
Maintainability	Esfuerzo requerido en ubicar y arreglar un error. **Puedo corregir una falla fácilmente?.**	**Readability and code structure, modularity, consistency, simplicity, documentation**
Testability	***Puedo probar el producto sin añadir costo adicional después de hacer cambios?***	***Simplicity and code structure, modularity, instrumentation, documentation***
Interoperability	Esfuerzo requerido para unir dos sistemas. **Puedo crear una interfaz con otros sistemas?**	**Modularity, compatibility**

Métrica	Definición	*Factor*
Flexibility	*Esfuerzo requerido para modificar un programa. Puedo ejecutar un cambio fácilmente?*	*Modularity, simplicity of design and code, generality, quality of documentation*
Portability	Esfuerzo para transferir un programa de una configuración de hardware y/o ambiente a otro. **Puedo usar el programa en otro hardware?**	**Modularity, quality of documentation, software system independence**
Reusability	*Grado en el cual un programa puede ser usado en otras aplicaciones. Puedo usar partes de la aplicación en otros programas?*	*Simplicity and code structure, modularity, quality and availability of documentation, software system independence, machine independence*

¿Son valiosos los atributos?

- Recuerden que cada atributo es una promesa que debe cumplirse
- Un atributo sin métrica pierde sustentabilidad
- Demasiados atributos limitan la capacidad de acción, pocos limitan utilidad
    - Debemos saber cuándo elegirlos

