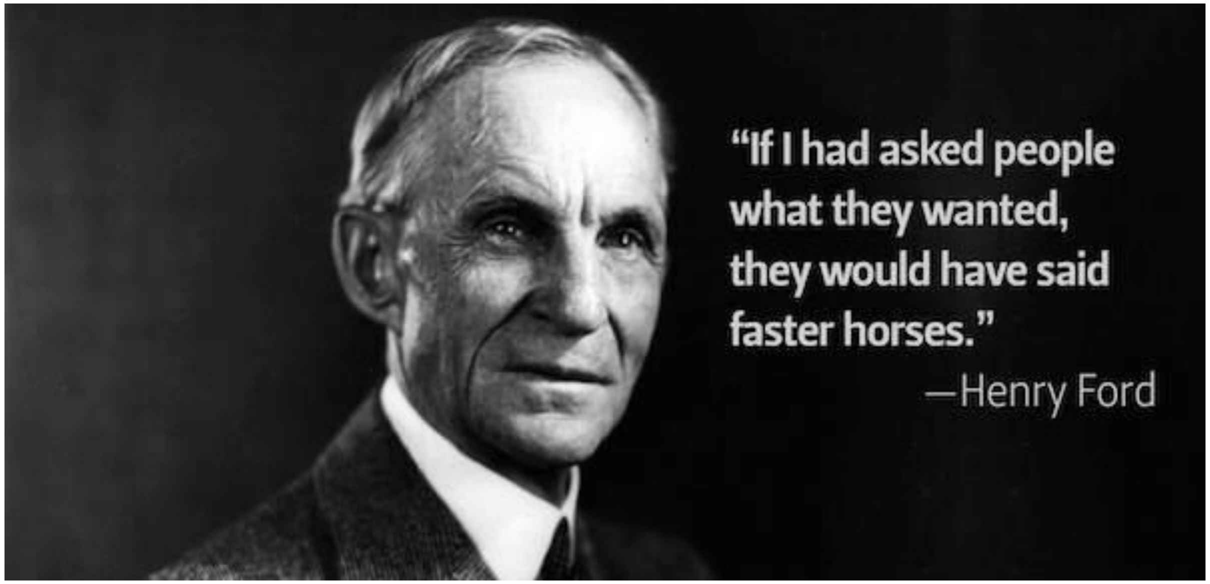
Software Architecture

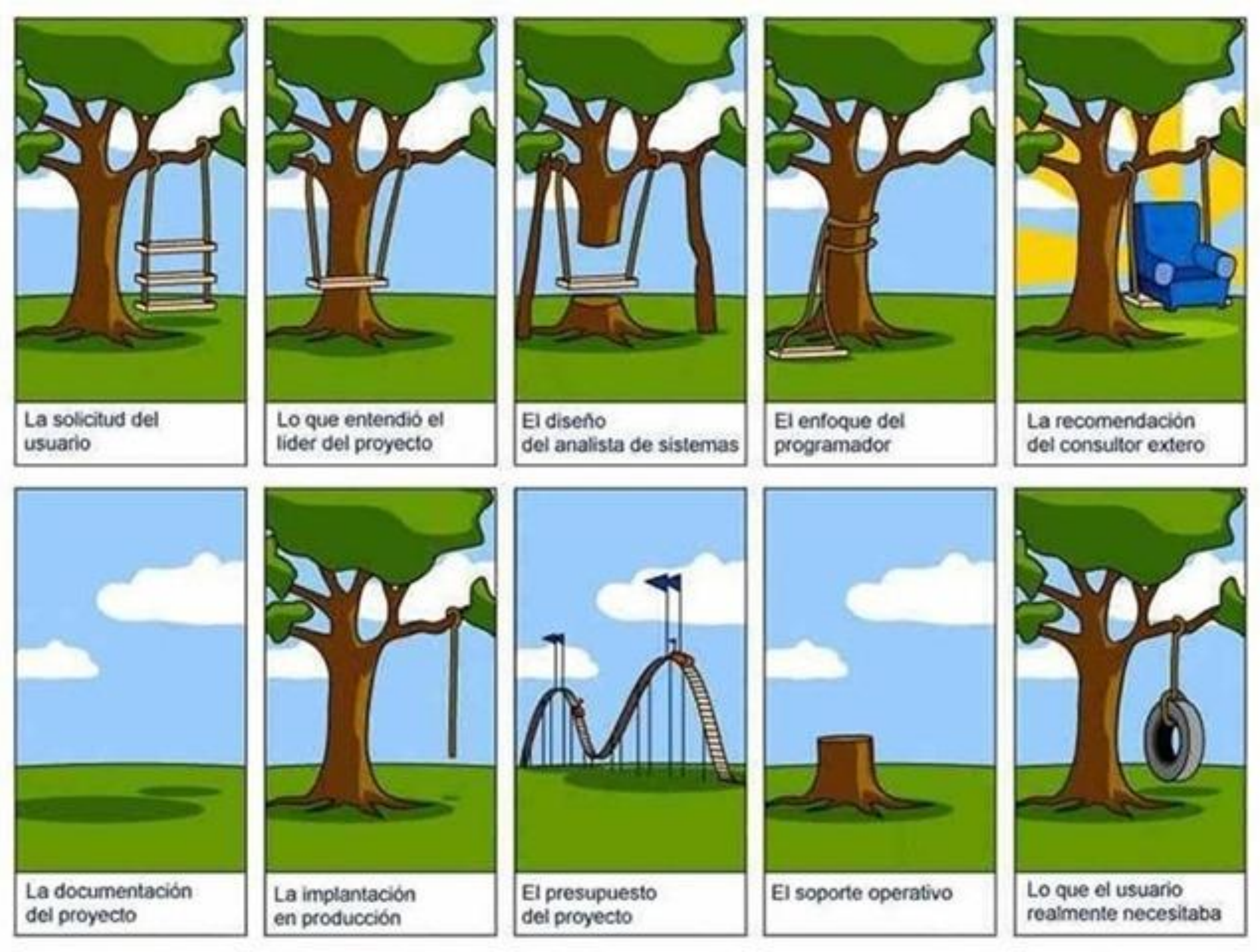
Software Architecture


"If I had asked people what they wanted, they would have said faster horses."
—Henry Ford

You've got to start with the customer experience and work back towards the technology.

—Steve Jobs

Los Requerimientos y su importancia en el desarrollo del Software.


La solicitud del usuario
Lo que entendió el líder del proyecto
El diseño del analista de sistemas
El enfoque del programador
La recomendación del consultor extero
La documentación del proyecto
La implantación en producción
El presupuesto del proyecto
El soporte operativo
Lo que el usuario realmente necesitaba

Fuente: http://kuainasi.ciens.ucv.ve/red_educativa/blogs/20

Servicios web - Interoperabilidad

Creando un Recurso con POST

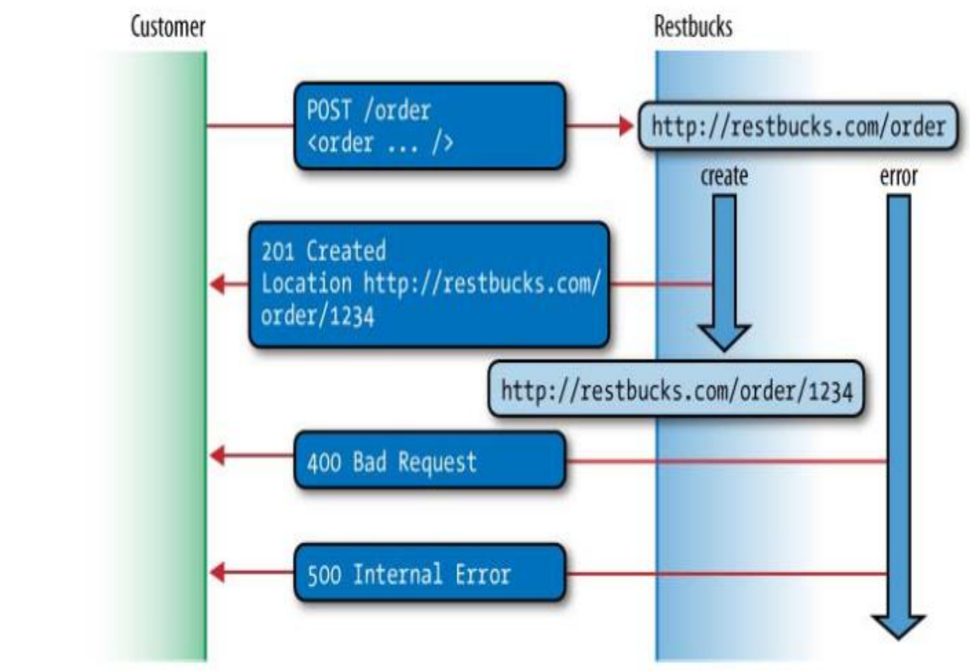

Customer
Restbucks
POST /order
<order ... />
http://restbucks.com/order
create
error
201 Created
Location http://restbucks.com/order/1234
http://restbucks.com/order/1234
400 Bad Request
500 Internal Error

Figure 4-4. *Creating an order via POST*

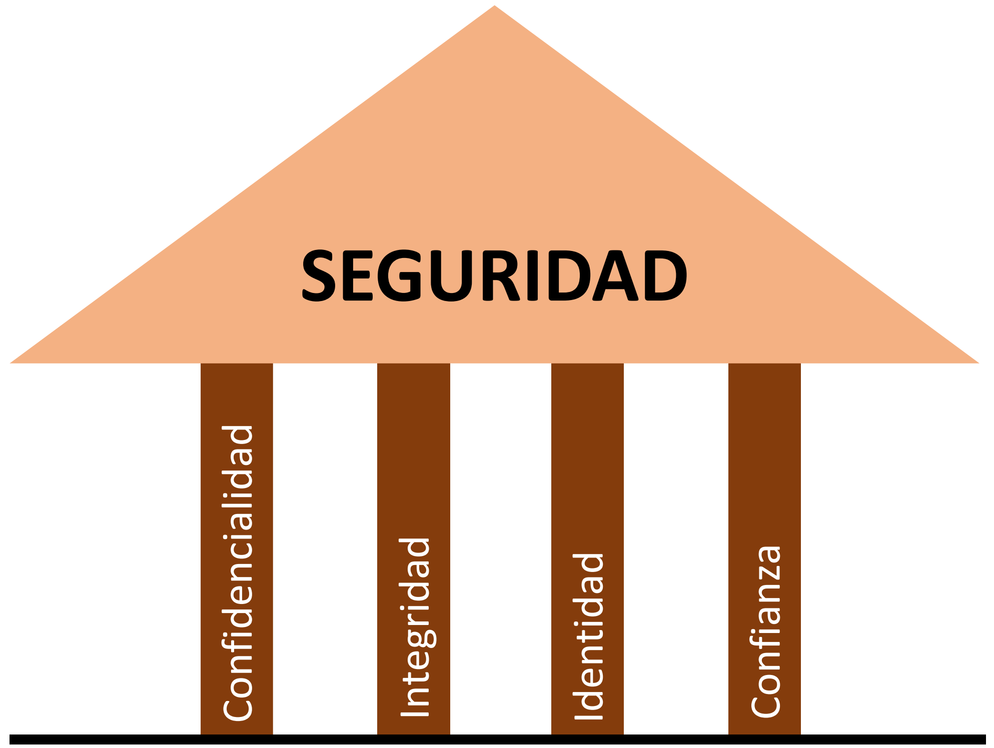
Fuente: (Webber, Parastatidis, & Ian, 2010)

Formatos:

- JSON
- XML

	ACCION	HTTP	SQL
C	Crear	POST	INSERT
R	Leer	GET	SELECT
U	Actualizar	PUT	UPDATE
D	Eliminar	DELETE	DELETE

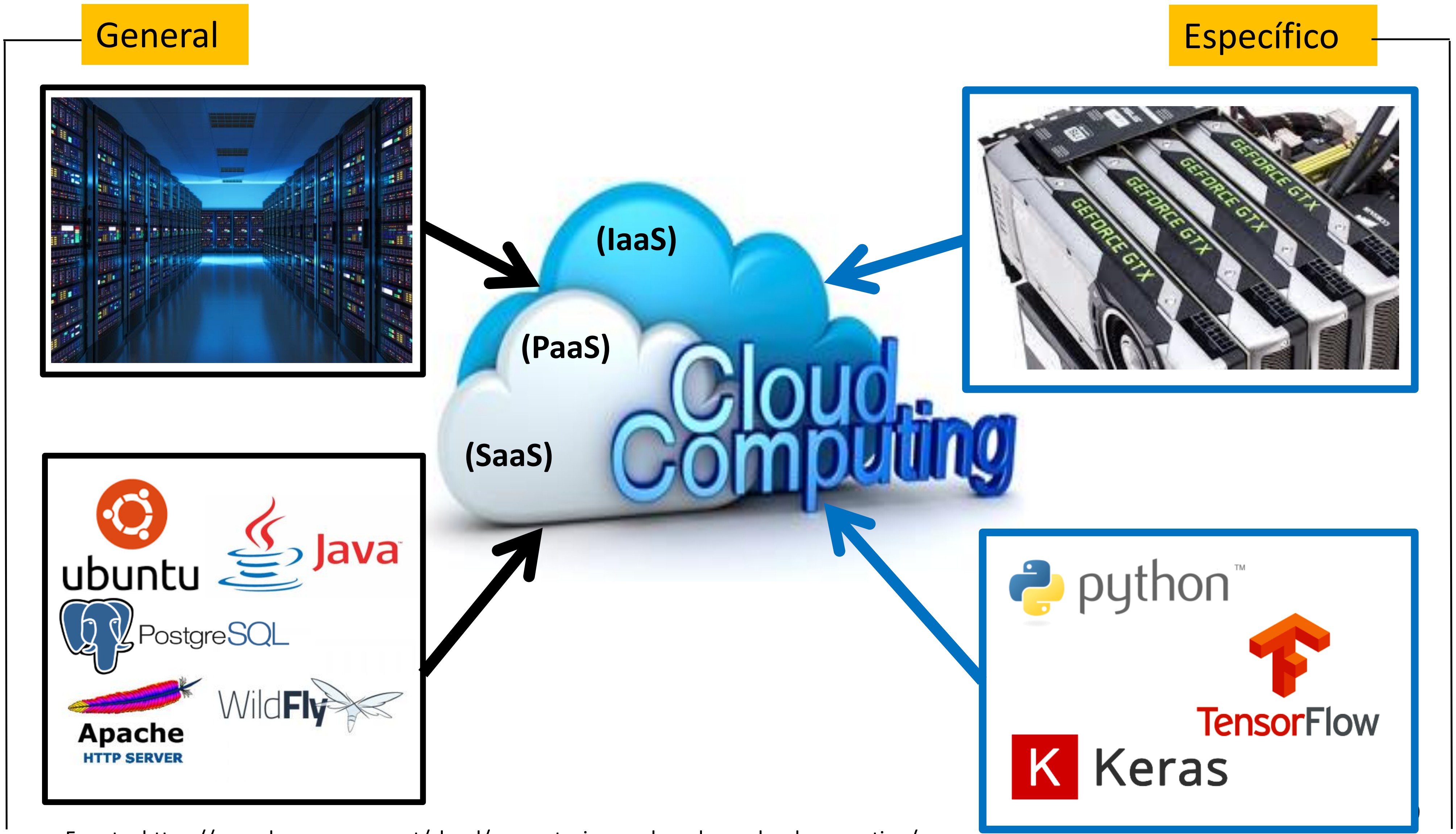
Seguridad



Webber et al. (2010)

- HTTPS, Tokens y Cifrado (Claves Encriptadas)
- Auditorias (Triggers)
- JAAS - Servicio de Autorización y Autenticación de Java. (OWASP – ESAPI)

Computación en la nube (Cloud Computing):



Fuente: https://www.lomasnuevo.net/cloud/computacion-en-la-nube-o-cloud-computing/

Software architecture?

“the highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces.”

Rational Unified Process definition, working off the IEEE definition

http://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf

Software architecture

The IEEE standard 1471 defines software architecture as the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution.

Software architecture serves as the blueprint for both the system and the project developing it, defining the work assignments that must be carried out by design and implementation teams.

Why do we need software architecture?

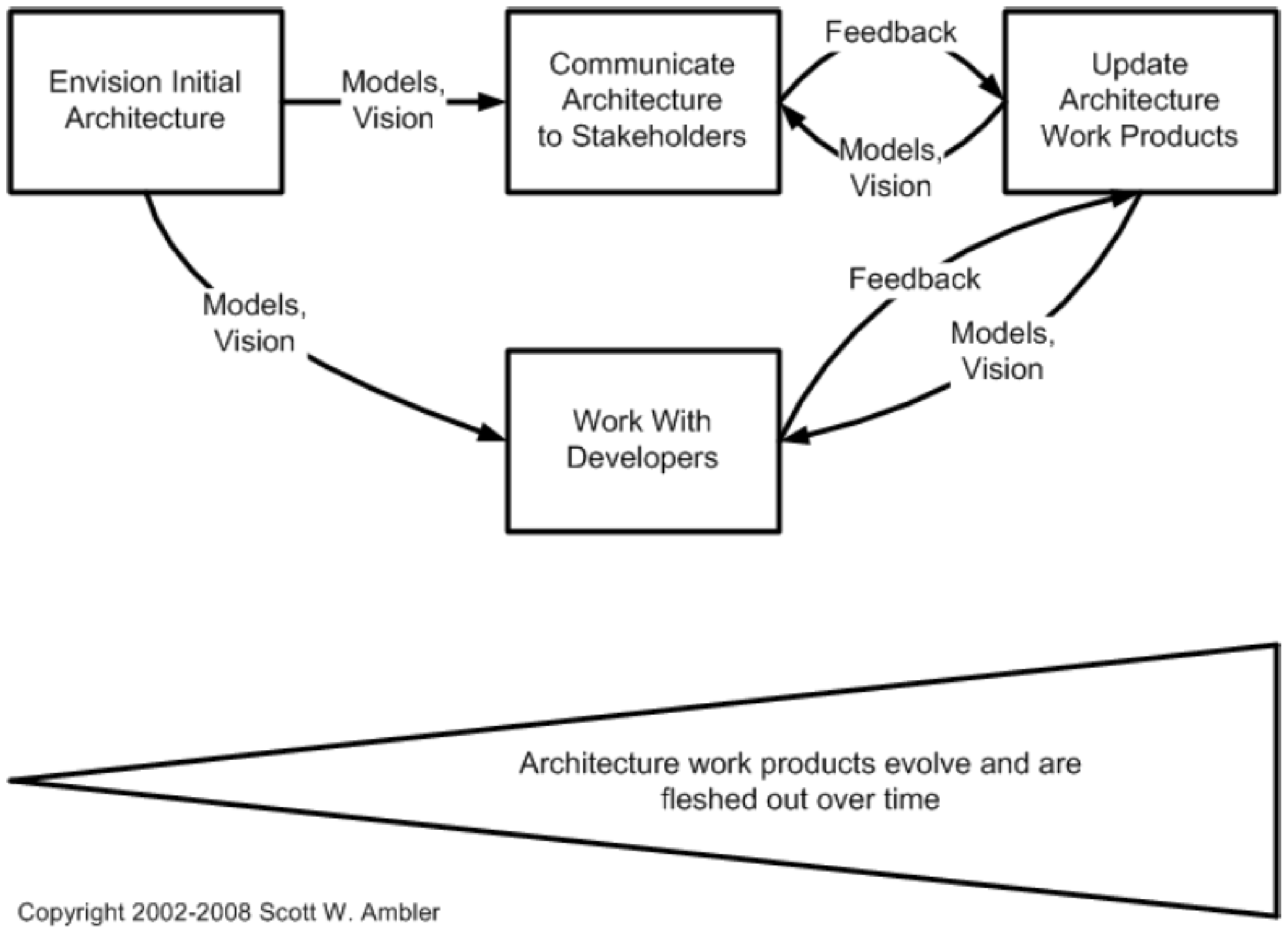
- Software architecture as a means for communication
- Software architecture as a representation of early design decisions
- Software architecture as a basis for work-breakdown structure
- Software architecture as a means to evaluate quality attributes
- Software architecture as a unit of reuse

Source: https://www.ou.nl/documents/40554/349790/IM0203_01.pdf

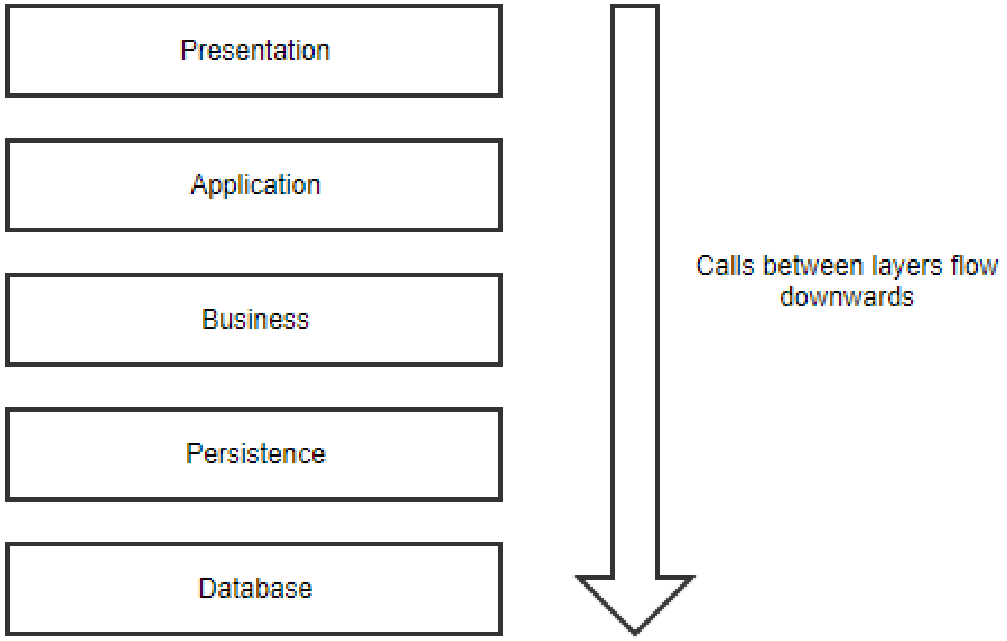
The Role of an Architec

Central activities:

- Design
- Document
- Assess
- Recover
- Maintain


Envision Initial Architecture
Models, Vision
Communicate Architecture to Stakeholders
Feedback
Models, Vision
Update Architecture Work Products
Models, Vision
Feedback
Models, Vision
Work With Developers
Architecture work products evolve and are fleshed out over time
Copyright 2002-2008 Scott W. Ambler

1. Layered Pattern



- One of the most well-known software architecture patterns.
- The idea is to split up your code into "layers"
- Each layer has a certain responsibility and provides a service to a higher layer.
- There isn't a predefined number of layers.
- Higher layers are dependent upon and make calls to the lower layers.
- You will see variations of this, depending on the complexity of the applications (delete, merge layers).

1. Layered Pattern

Advantages

- Most developers are familiar with this pattern.
- It provides an easy way of writing a well-organized and testable application.

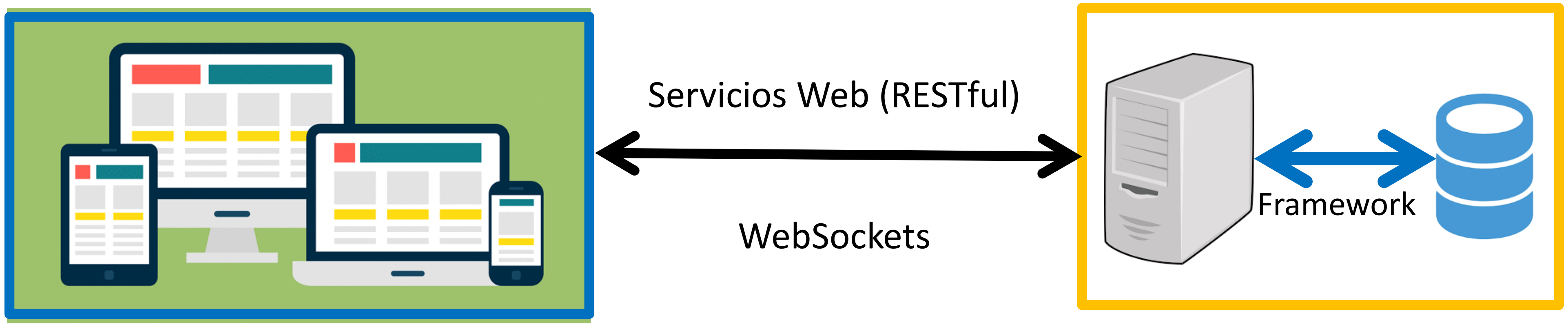
Disadvantages

- It tends to lead to monolithic applications that are hard to split up afterward.
- Developers often find themselves writing a lot of code to pass through the different layers, without adding any value in these layers. If all you are doing is writing a simple CRUD application, the layered pattern might be overkill for you.

Ideal for

- Standard line-of-business apps that do more than just CRUD operations

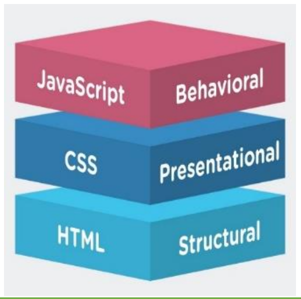
Propuesta Desarrollo Web



Fuente: https://www.caktusgroup.com/blog/2017/01/04/responsive-web-design/

Front-end:

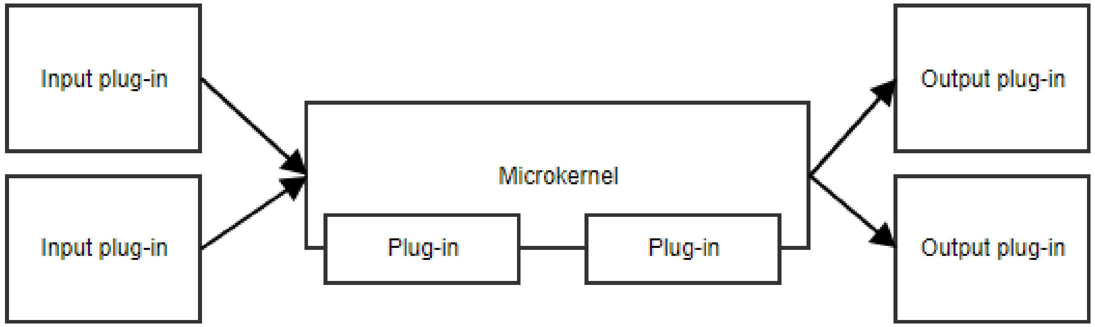
- Usar un Framework
  - Bootstrap (jQuery)
  - Angular Material (Angular)
  - React
- Servicios Web (RESTful)
- WebSockets
- Web Responsive
- No plugins (HTML5+CSS+JS)



Back-end:

- Servicios Web (RESTful)
- WebSockets
- Framework acceso a base de datos (JDBC)

2. Microkernel



- The microkernel pattern, or plug-in pattern, is useful when your application has a core set of responsibilities and a collection of interchangeable parts on the side.
- The microkernel will provide the entry point and the general flow of the application, without really knowing what the different plug-ins are doing.

- An example is a task scheduler.
- Another example is a workflow.

2. Microkernel

Advantages

- This pattern provides great flexibility and extensibility.
- Some implementations allow for adding plug-ins while the application is running.
- Microkernel and plug-ins can be developed by separate teams.

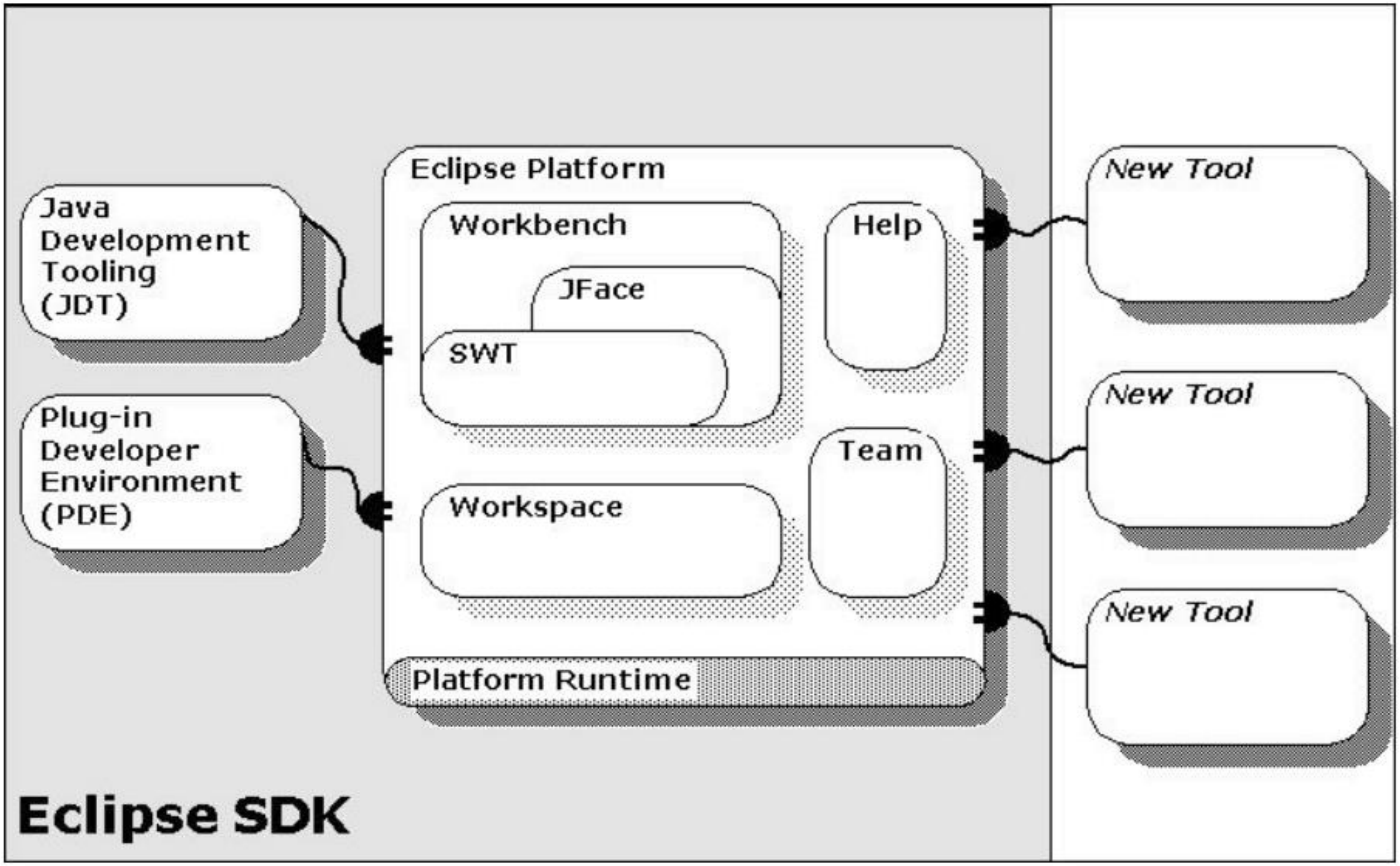
Disadvantages

- It can be difficult to decide what belongs in the microkernel and what doesn’t.
- The predefined API might not be a good fit for future plug-ins.

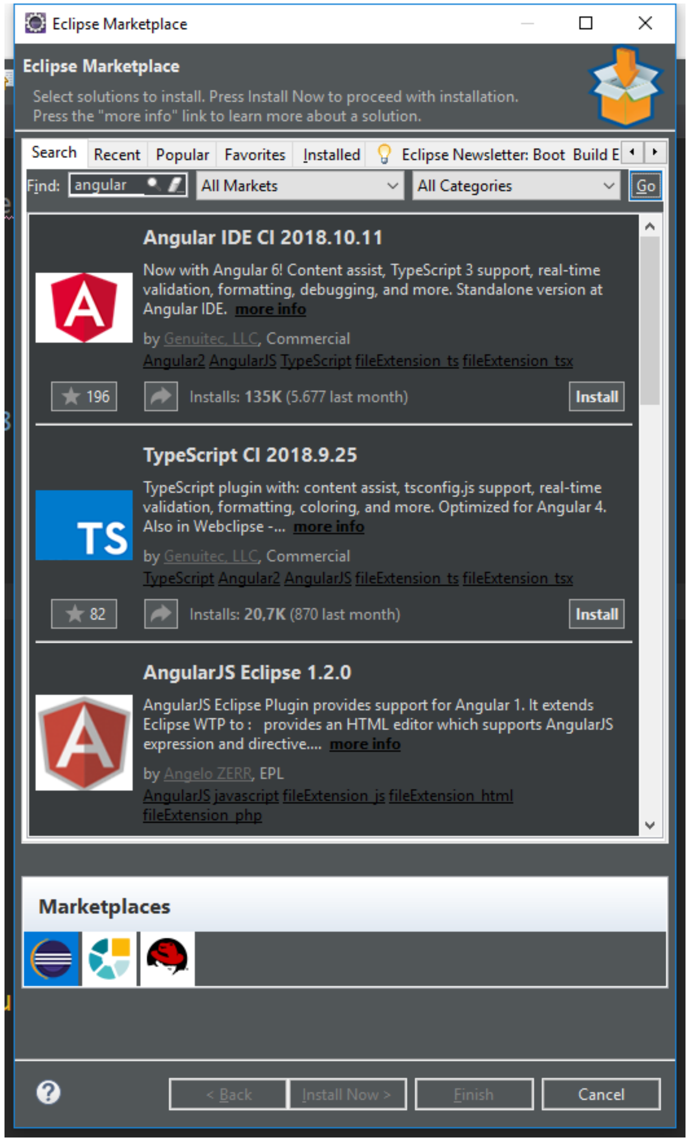
Ideal for

- Applications that take data from different sources, transform that data and writes it to different destinations
- Workflow applications
- Task and job scheduling applications

2. Microkernel: Eclipse Platform Architecture



2. Microkernel: Eclipse Platform Architecture



Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Search | Recent | Popular | Favorites | Installed | Eclipse Newsletter: Boot Build E

Find: angular | All Markets | All Categories | Go

Angular IDE CI 2018.10.11

Now with Angular 6! Content assist, TypeScript 3 support, real-time validation, formatting, debugging, and more. Standalone version at Angular IDE. more info

by Genuitec, LLC, Commercial

Angular2 AngularJS TypeScript fileExtension_ts fileExtension_tsx

196 | Installs: 135K (5,677 last month) | Install

TypeScript CI 2018.9.25

TypeScript plugin with: content assist, tsconfig.js support, real-time validation, formatting, coloring, and more. Optimized for Angular 4. Also in Webclipse -... more info

by Genuitec, LLC, Commercial

TypeScript Angular2 AngularJS fileExtension_ts fileExtension_tsx

82 | Installs: 20,7K (870 last month) | Install

AngularJS Eclipse 1.2.0

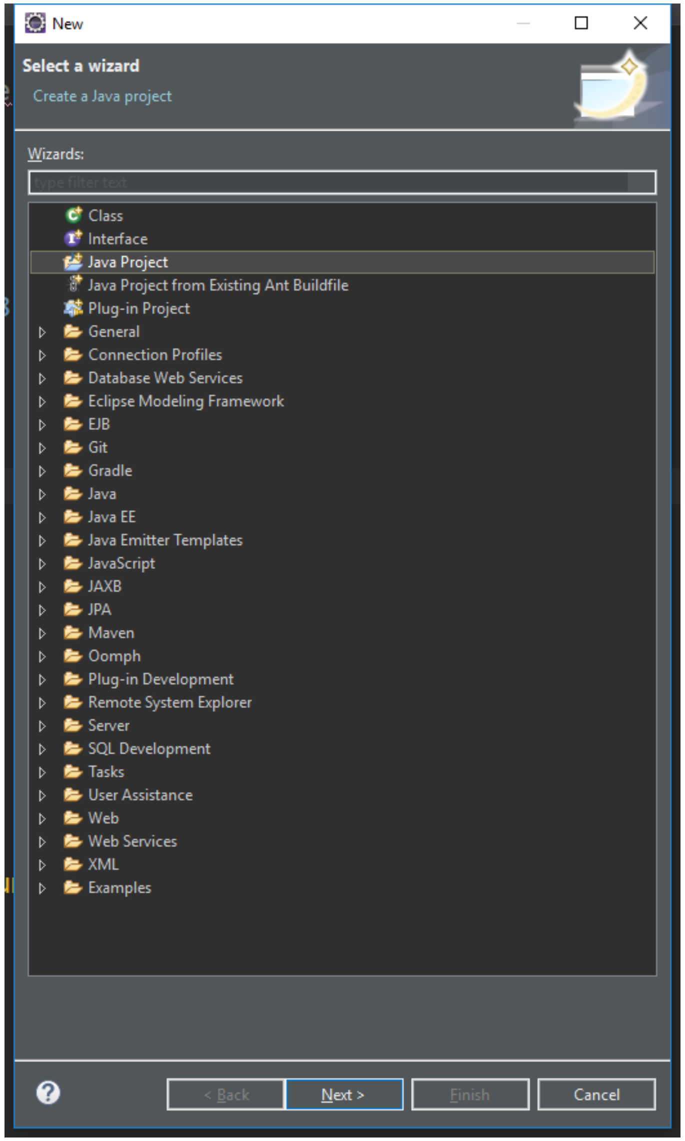
AngularJS Eclipse Plugin provides support for Angular 1. It extends Eclipse WTP to : provides an HTML editor which supports AngularJS expression and directive.... more info

by Angelo ZERR, EPL

AngularJS javascript fileExtension_js fileExtension_html fileExtension_php

Marketplaces

< Back | Install Now > | Finish | Cancel



New

Select a wizard

Create a Java project

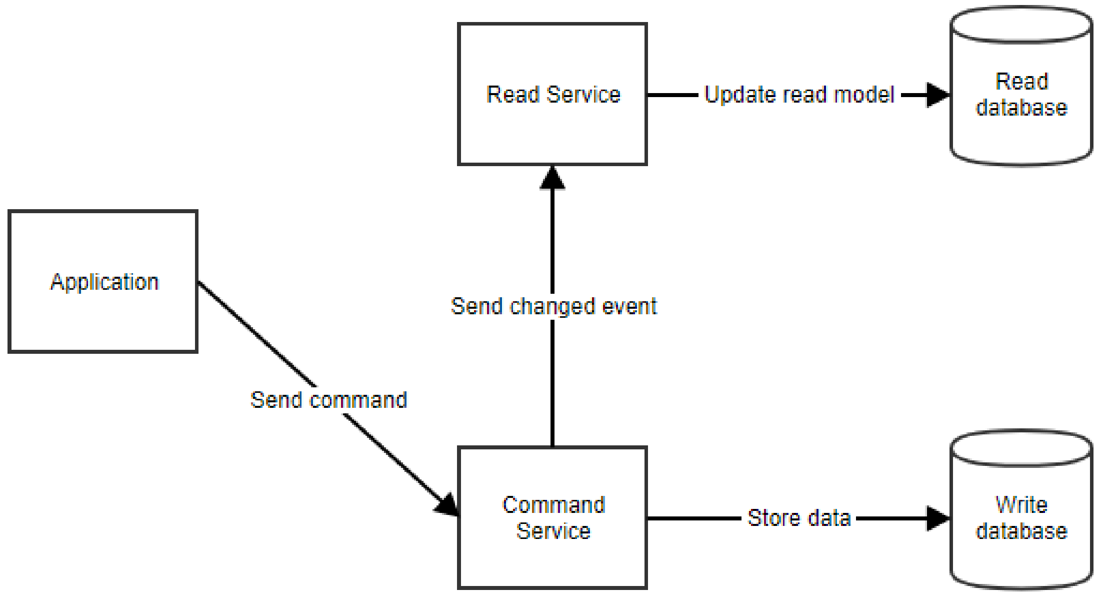
Wizards:

type filter text

- Class
- Interface
- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- General
- Connection Profiles
- Database Web Services
- Eclipse Modeling Framework
- EJB
- Git
- Gradle
- Java
- Java EE
- Java Emitter Templates
- JavaScript
- JAXB
- JPA
- Maven
- Oomph
- Plug-in Development
- Remote System Explorer
- Server
- SQL Development
- Tasks
- User Assistance
- Web
- Web Services
- XML
- Examples

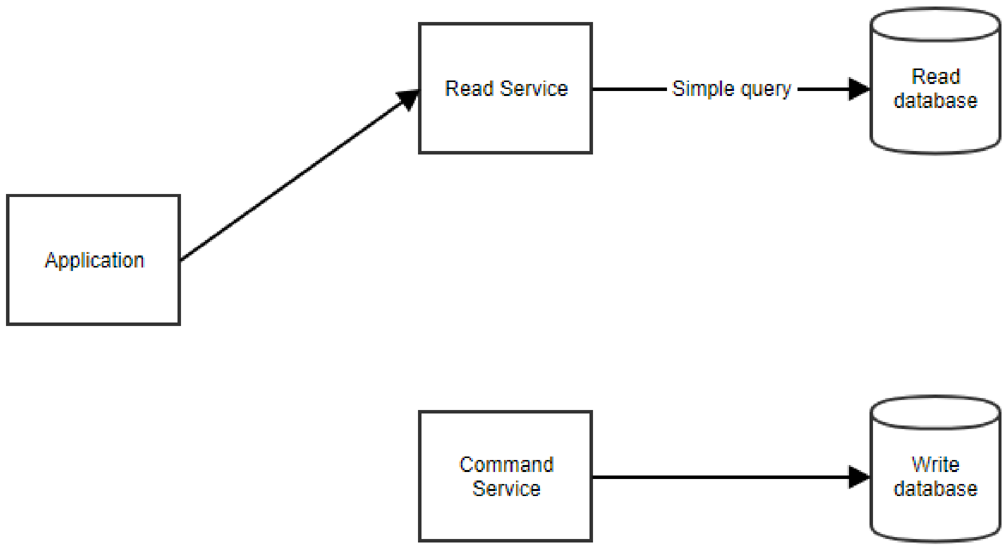
< Back | Next > | Finish | Cancel

3. **CQRS:** Command and Query Responsibility Segregation



- The central concept of this pattern is that an application has read operations and write operations that must be totally separated.
- This also means that the model used for write operations (commands) will differ from the read models (queries).
- Furthermore, the data will be stored in different locations. In a relational database, this means there will be tables for the command model and tables for the read model.

3. CQRS: Command and Query Responsibility Segregation



- Some implementations even store the different models in totally different databases, e.g. SQL Server for the command model and MongoDB for the read model.
- This pattern is often combined with event sourcing, which we'll cover below.

3. CQRS: Command and Query Responsibility Segregation

Advantages

- Command models can focus on business logic and validation while read models can be tailored to specific scenarios.
- You can avoid complex queries (e.g. joins in SQL) which makes the reads more performant.

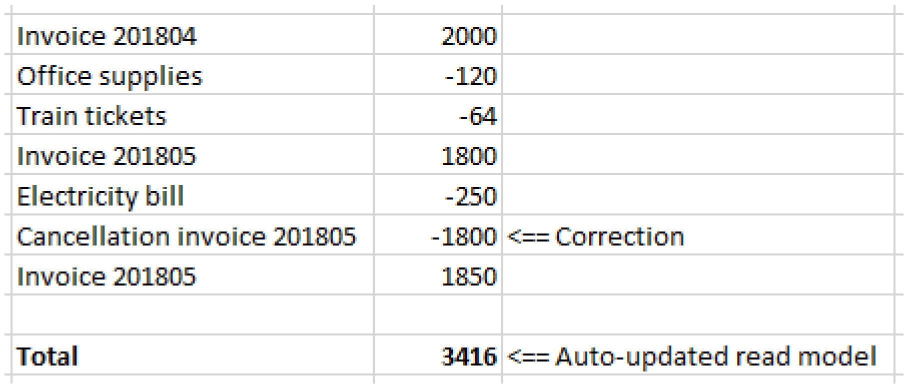
Disadvantages

- Keeping the command and the read models in sync can become complex.

Ideal for

- Applications that expect a high amount of reads
- Applications with complex domains

4. Event Sourcing



Invoice 201804	2000	
Office supplies	-120	
Train tickets	-64	
Invoice 201805	1800	
Electricity bill	-250	
Cancellation invoice 201805	-1800	<== Correction
Invoice 201805	1850	
Total	**3416**	<== Auto-updated read model

- As I mentioned above, CQRS often goes hand in hand with event sourcing.
- This is a pattern where you don't store the current state of your model in the database, but rather the events that happened to the model.
- So when the name of a customer changes, you won't store the value in a "Name" column. You will store a "NameChanged" event with the new value (and possibly the old one too).

- <u>A real-life analogy of event sourcing is accounting.</u>

- To make your life easier, you could calculate the total every time you add a line. This total can be regarded as the read model.

4. Event Sourcing

Advantages

- This software architecture pattern can provide an audit log out of the box. Each event represents a manipulation of the data at a certain point in time.

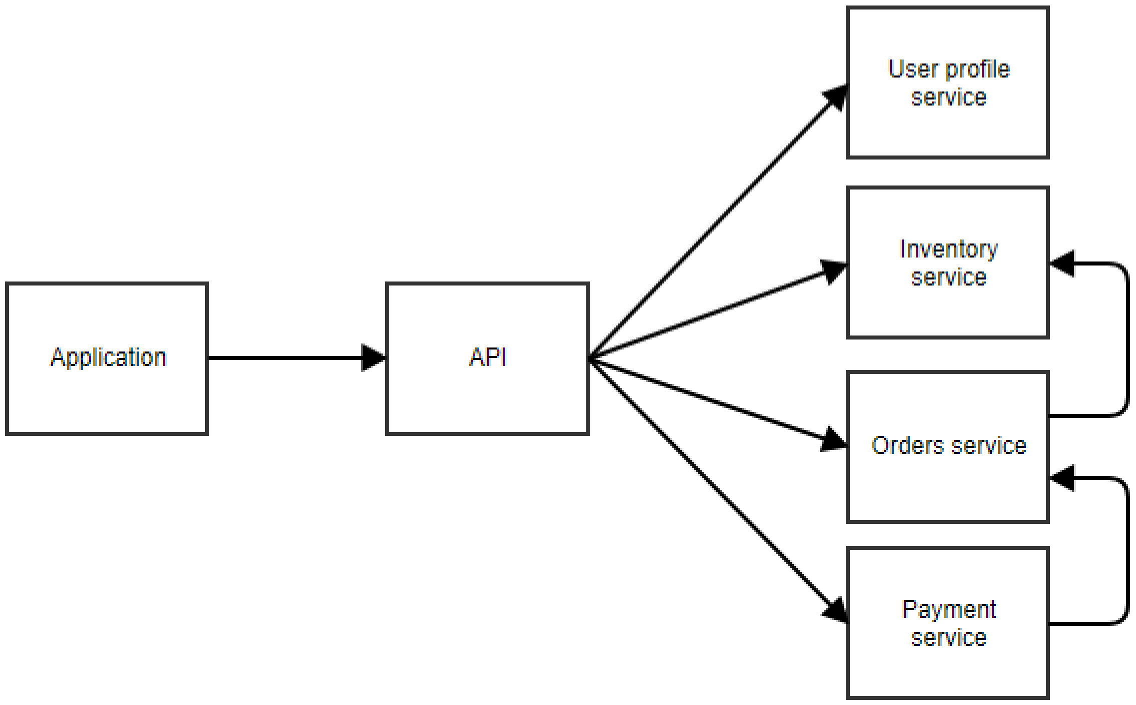
Disadvantages

- It requires some discipline because you can’t just fix wrong data with a simple edit in the database.
- It’s not a trivial task to change the structure of an event. For example, if you add a property, the database still contains events without that data. Your code will need to handle this missing data graciously.

Ideal for applications that

- Need to publish events to external systems
- Will be built with CQRS
- Have complex domains
- Need an audit log of changes to the data

5. Microservices



- When you write your application as a set of microservices, you're actually writing multiple applications that will work together.
- Each microservice has its own distinct responsibility and teams can develop them independently of other microservices.
- The only dependency between them is the communication.

5. Microservices

Advantages

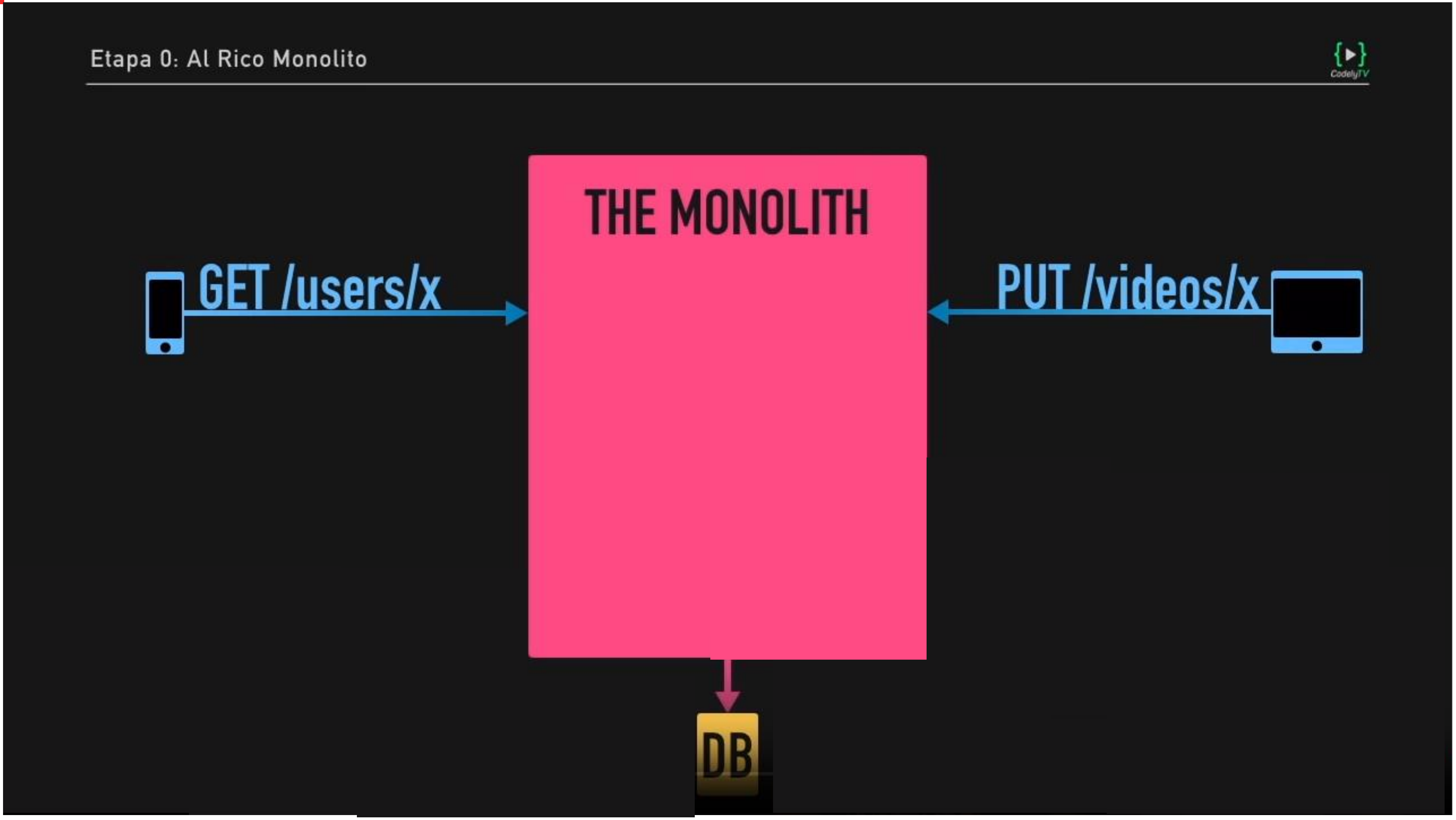
- You can write, maintain, and deploy each microservice separately.
- A microservices architecture should be easier to scale, as you can scale only the microservices that need to be scaled. There's no need to scale the less frequently used pieces of the application.
- It's easier to rewrite pieces of the application because they're smaller and less coupled to other parts.

Disadvantages

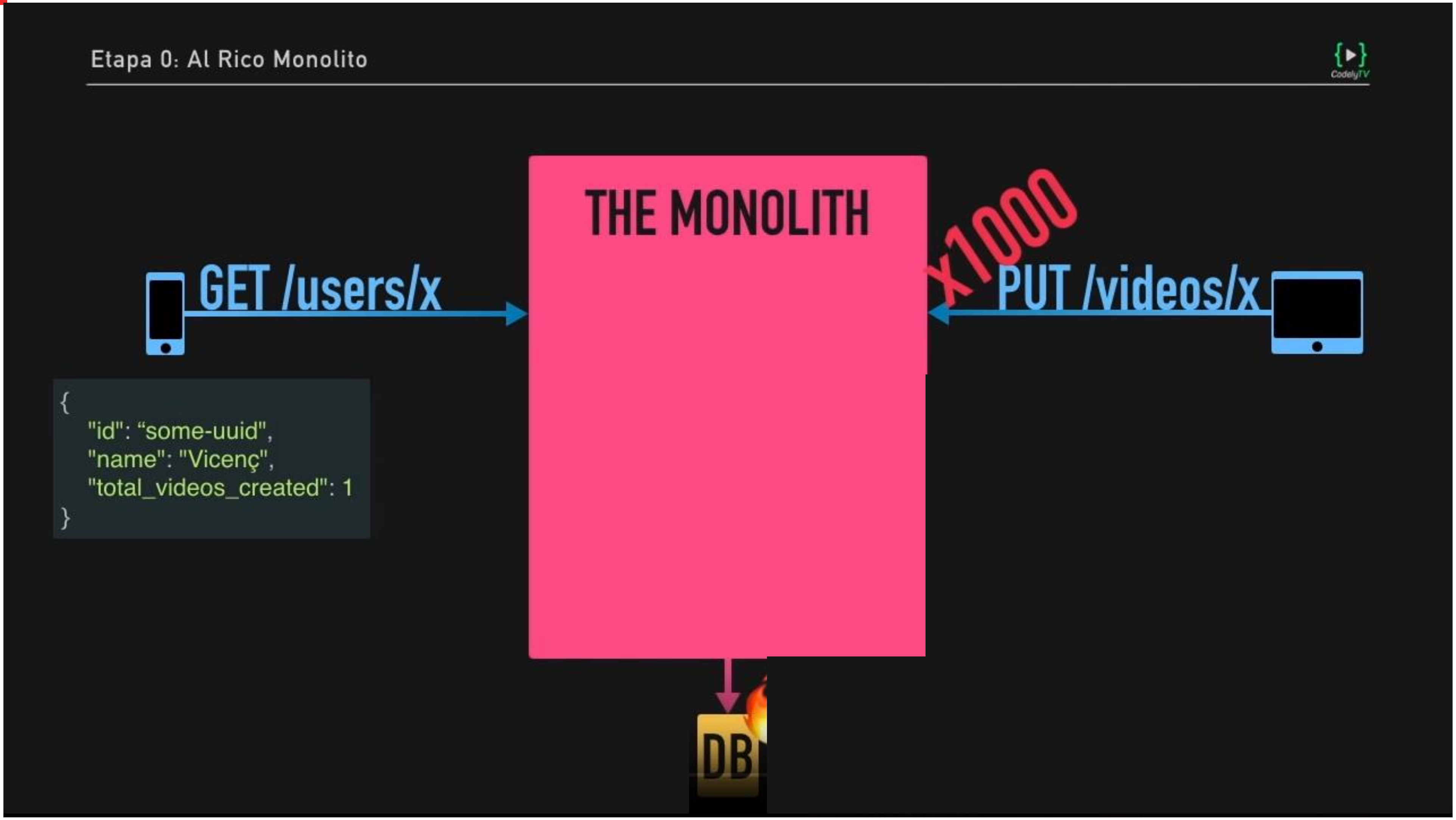
- Contrary to what you might expect, it's actually easier to write a well-structured monolith at first and split it up into microservices later. With microservices, a lot of extra concerns come into play: communication, coordination, backward compatibility, logging, etc. Teams that miss the necessary skill to write a well-structured monolith will probably have a hard time writing a good set of microservices.
- A single action of a user can pass through multiple microservices. There are more points of failure, and when something does go wrong, it can take more time to pinpoint the problem.

Ideal for:

- Applications where certain parts will be used intensively and need to be scaled
- Services that provide functionality to several other applications
- Applications that would become very complex if combined into one monolith

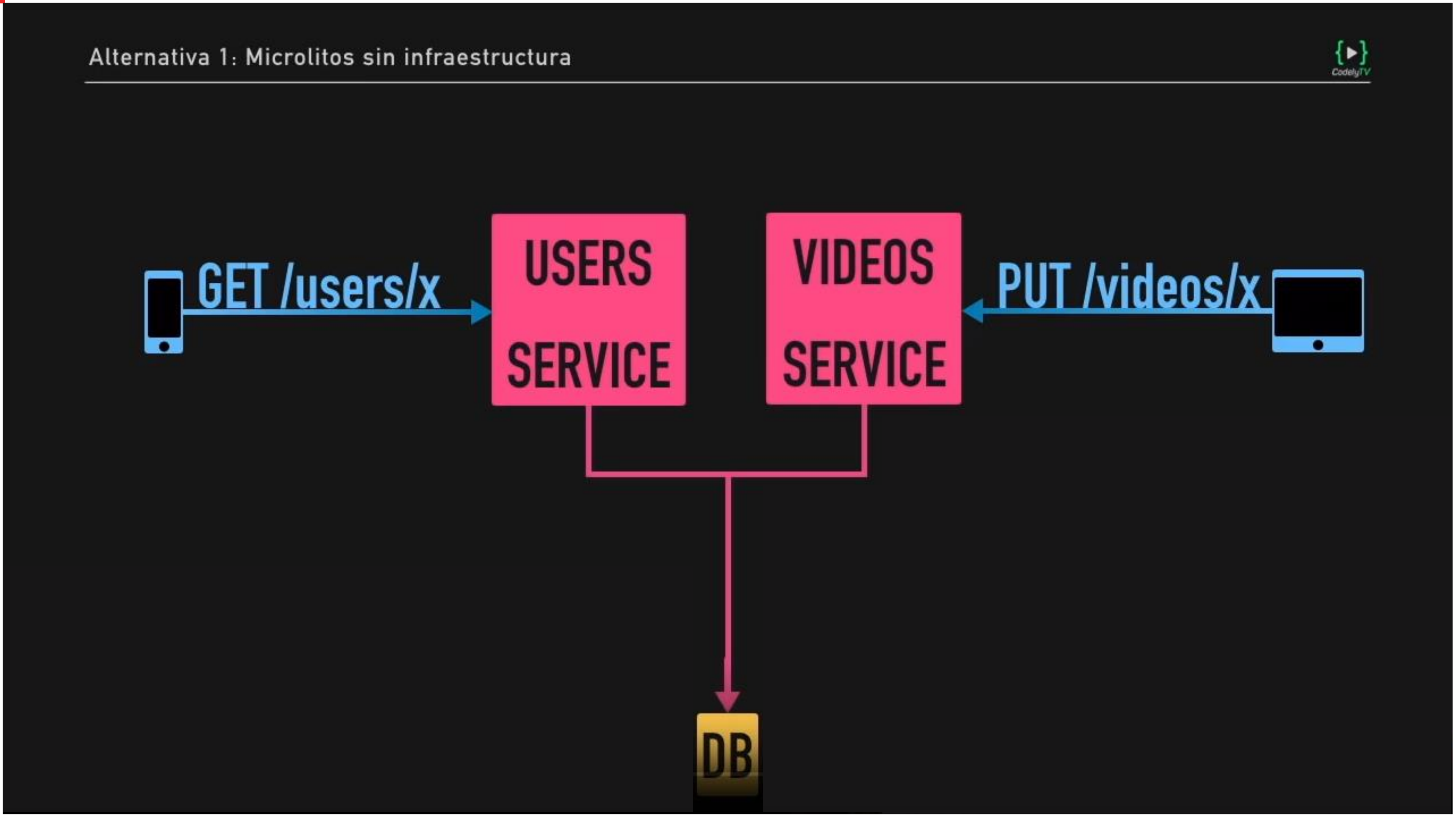


Etapa 0: Al Rico Monolito

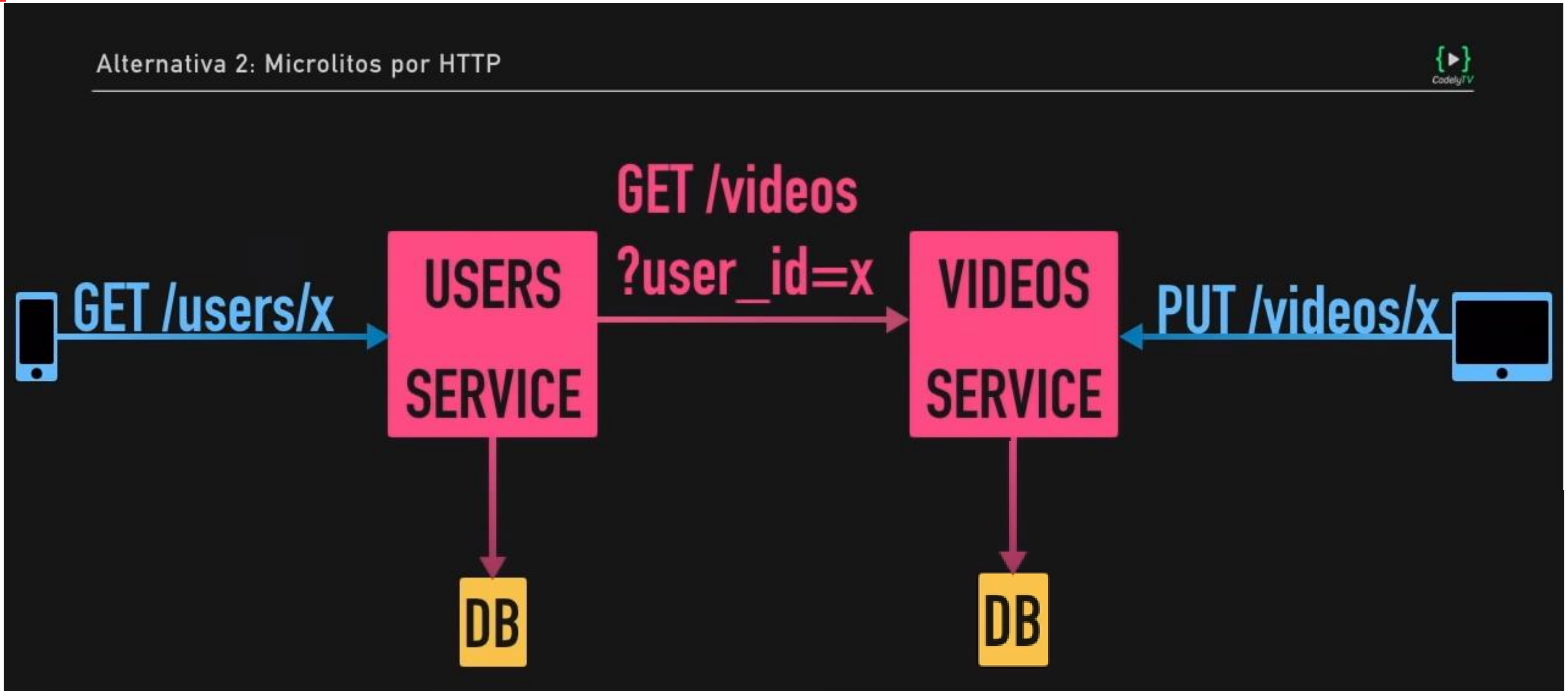


Etapa 0: Al Rico Monolito

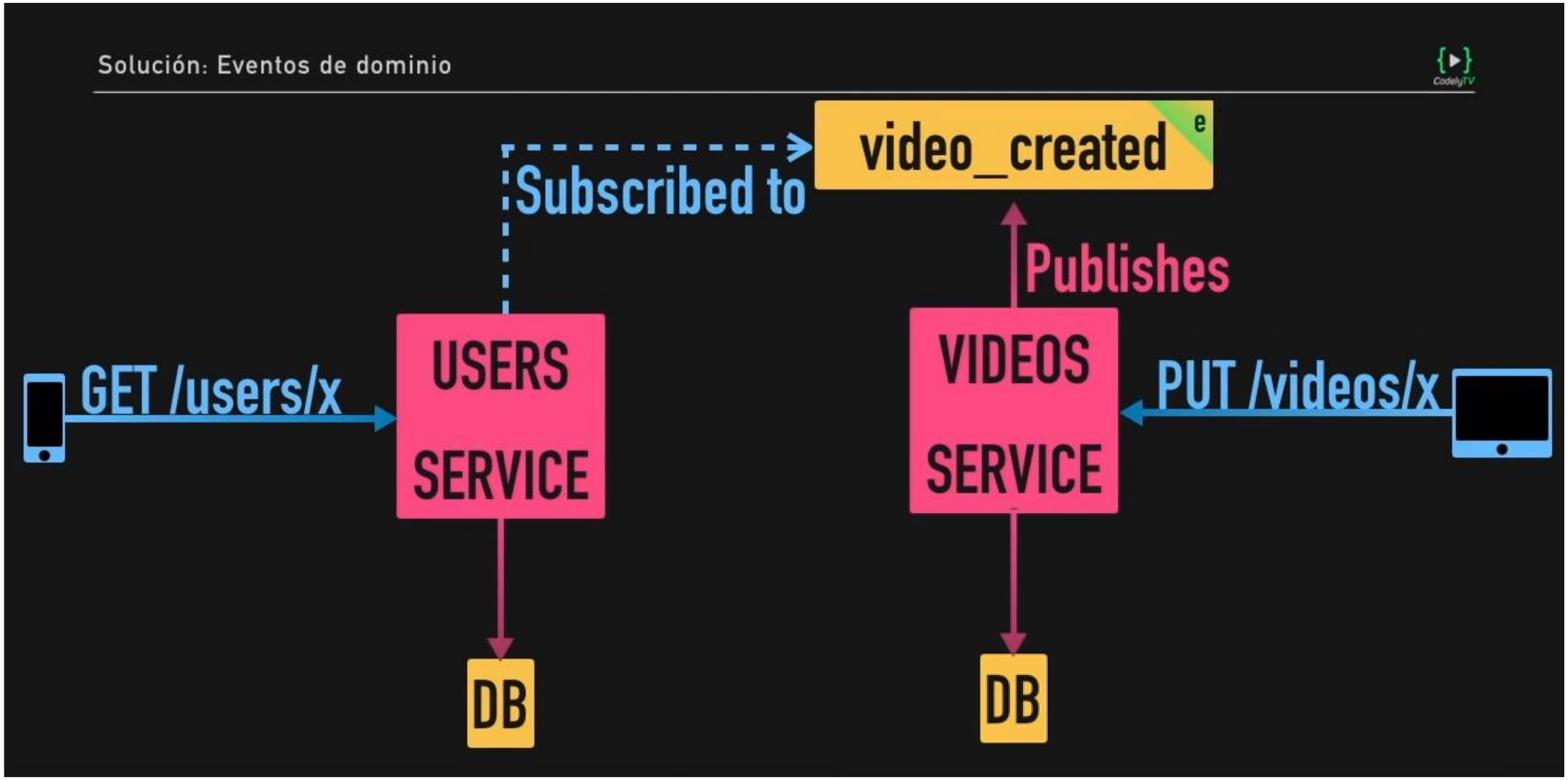
```
{
   "id": “some-uuid",
   "name": "Vicenç",
   "total_videos_created": 1
}
```



Alternativa 1: Microlitos sin infraestructura



Alternativa 2: Microlitos por HTTP

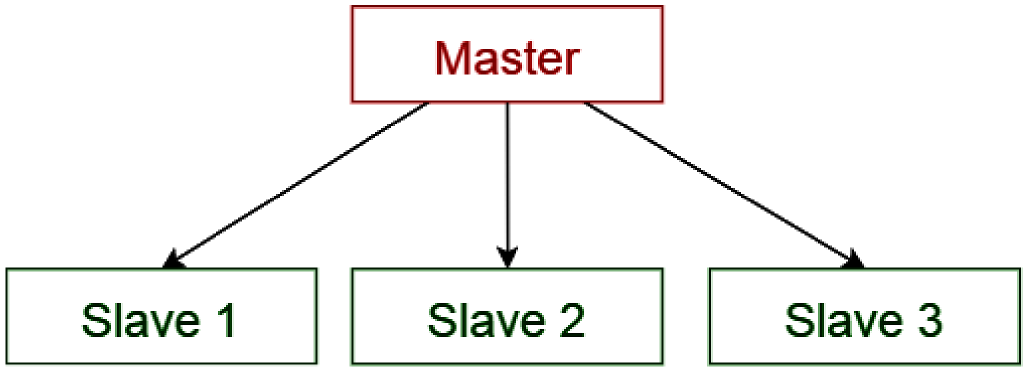

Solución: Eventos de dominio
CodelyTV
video_created
e
Subscribed to
Publishes
GET /users/x
USERS SERVICE
VIDEOS SERVICE
PUT /videos/x
DB
DB

6. Master / Slave

This pattern consists of two parties; **master** and **slaves**. The master component distributes the work among identical slave components, and computes a final result from the results which the slaves return.

Usage

- In database replication, the master database is regarded as the authoritative source, and the slave databases are synchronized to it.
- Peripherals connected to a bus in a computer system (master and slave drives).

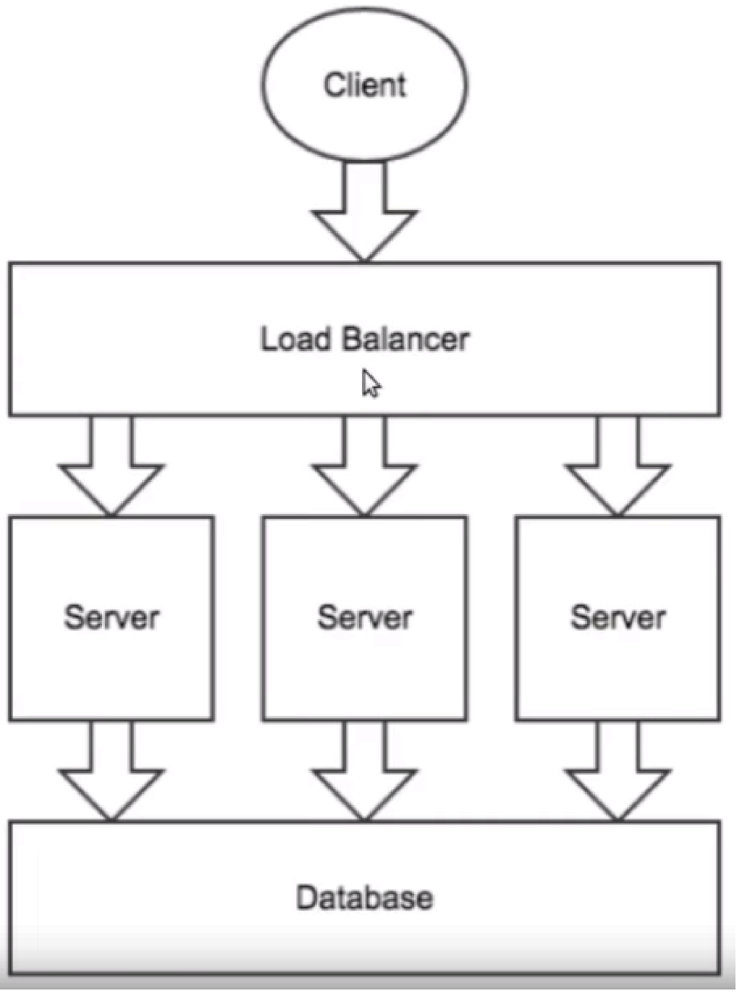


Master-slave pattern

6. Master / Slave

Issues of Monolithic

- Scaling the traffic is easy
- running the same executable in multiple machines and place the server behind a load balancer or use round robin DNS
- you have to scale all components (hardware, operative system) instead some of them.

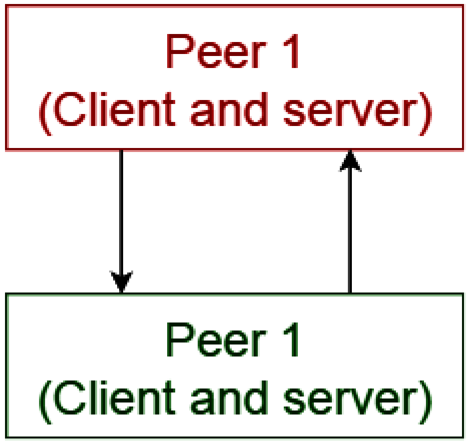


7. Peer- to - peer

In this pattern, individual components are known as **peers**. Peers may function both as a **client**, requesting services from other peers, and as a **server**, providing services to other peers. A peer may act as a client or as a server or as both, and it can change its role dynamically with time.

Usage

- File-sharing networks such as **Gnutella** and **G2**)
- Multimedia protocols such as **P2PTV** and **PDTP**.



Peer-to-peer pattern

References

[1] https://techbeacon.com/top-5-software-architecture-patterns-how-make-right-choice

[2] https://dzone.com/articles/software-architecture-the-5-patterns-you-need-to-k

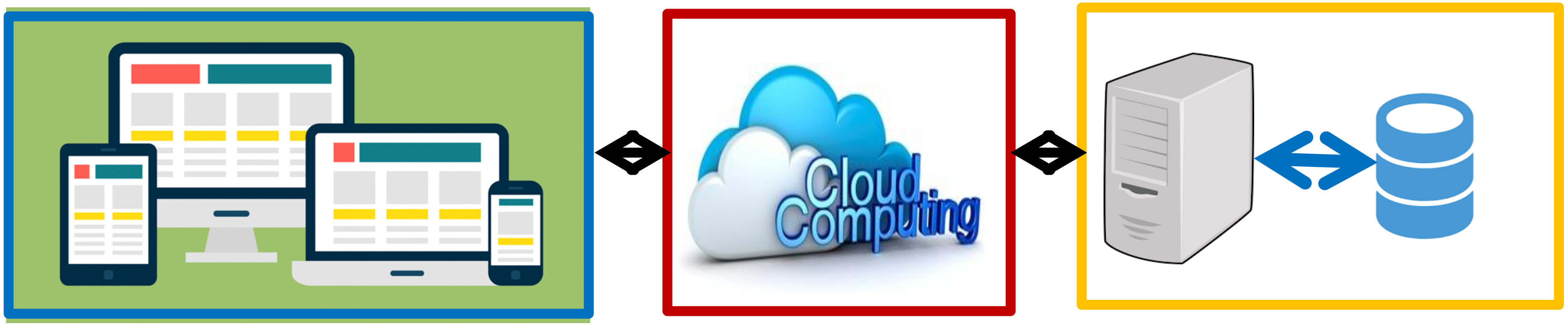
[3] https://www.youtube.com/watch?v=V4mjxJ5czog

[4] https://www.coursera.org/learn/software-architecture

[5] https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

gracias
go raibh maith agat
мерси
mochchakkeram
hvala
koszonom
malondo asante
dhanyavad
ngiyabonga
kia ora
dziękuję
dank je
grazie
sukriya
sagolun
taiku
teşekkür ederim
arigato
merci
obrigado
maururu
bedankt
kop khun krap
terima kasih
obrigada
djere dieuf
dakujem
спасибо
thank you
takk
danke
謝謝
tapadh leat
ありがとう
misaotra
blagodaram
kam sah hamnida
chokrane
rahmat
bayarlalaa
dankie
gratias ago
spas
mesi
diolch
grazzi
tack
merce
gracies
paldies
ευχαριστώ
tau

Proyecto Integrador



Fuente: https://www.caktusgroup.com/blog/2017/01/04/responsive-web-design/

Front-end:

- Usar un Framework
  - Bootstrap (jQuery)
  - (o) Angular Material (Angular)
- Validación
- Web Responsive
- No plugins (HTML5+CSS+JS)

Back-end:

- Servidor de aplicaciones (Java)
- Servicios Web (RESTful)
- Seguridad basada en roles
- Protocolo HTTPS
- Acceso a base de datos
- Validación
- Computación en la Nube