Sistemas Distribuidos I
(75.74)

Interfaces y Protocolos

Layers, Tiers, Interfaces, Protocolos

Docentes

- Pablo D. Roca
- Ezequiel Torres Feyuk
- Guido Albarello
- Ana Czarnitzki
- Cristian Raña

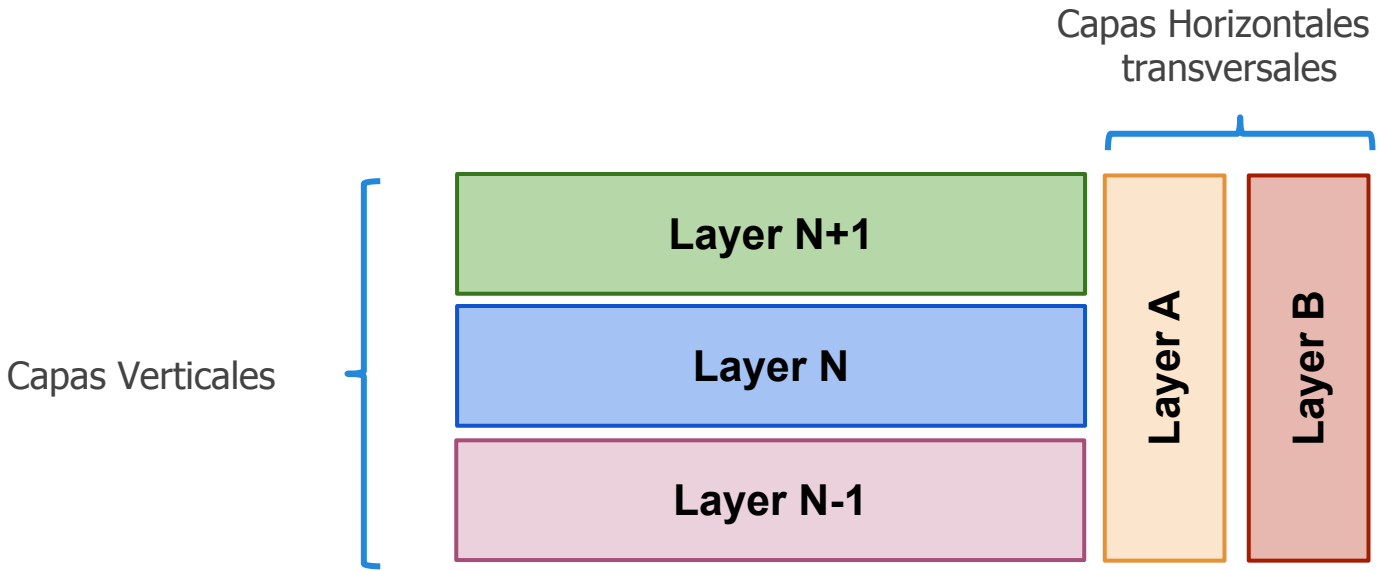
Agenda

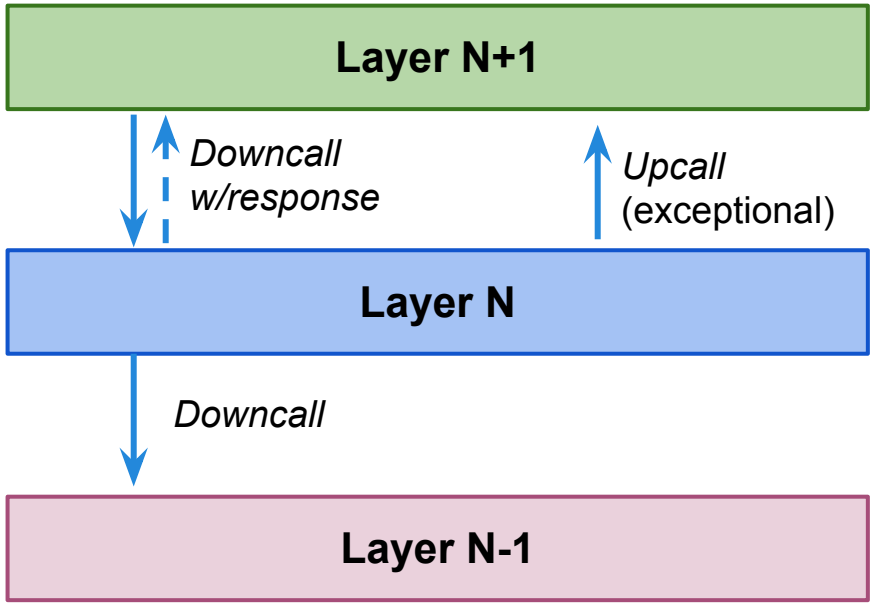
- **Modelos de Capas: Layers y Tiers**
- Interfaces
- Protocolos
- RESTful

Arquitecturas de Capas

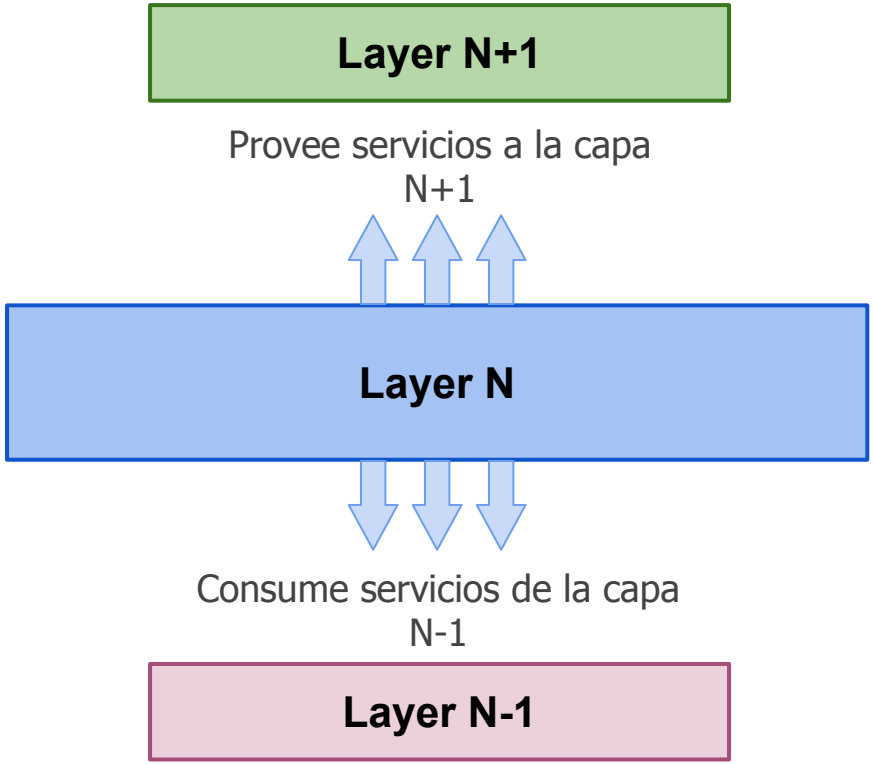
- Permiten dividir el problema en sub-problemas
- Fomentan el uso de interfaces
- Permiten intercambiar componentes reutilizando conectores y protocolos ya definidos
- Existen dos tipos de separación por capas:
  - *Layers,* o capas lógicas
  - *Tiers,* o capas físicas

Agrupación lógica de componentes y funcionalidades de un sistema.



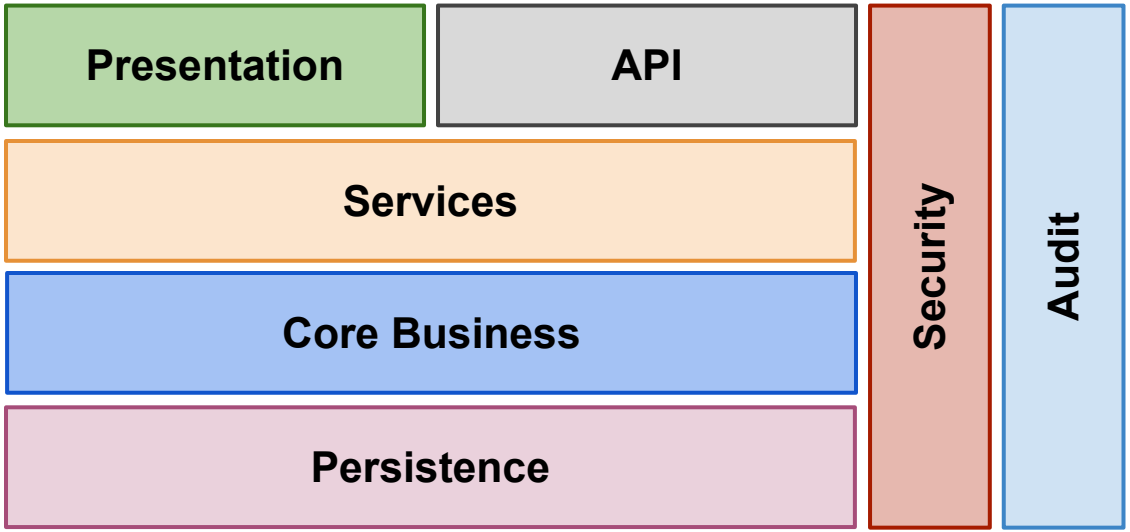

Layer N+1
Downcall
w/response
Upcall
(exceptional)
Layer N
Downcall
Layer N-1

Layers | Responsabilidades

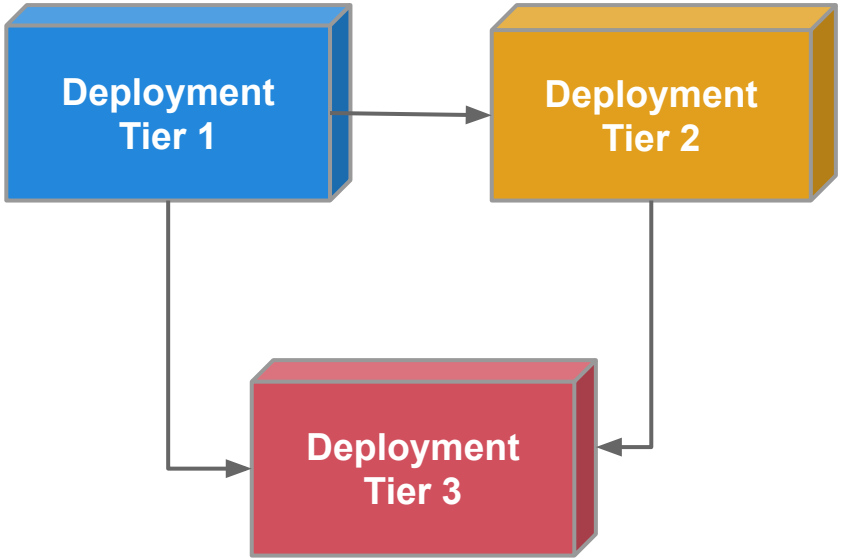


Módulo con responsabilidades limitadas, coherencia y cohesión

Layers | Ejemplo

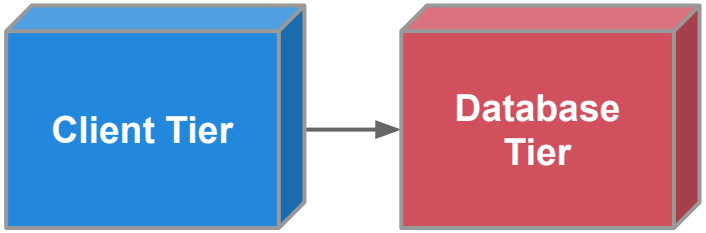


Describen la distribución física de componentes y funcionalidad de un sistema.

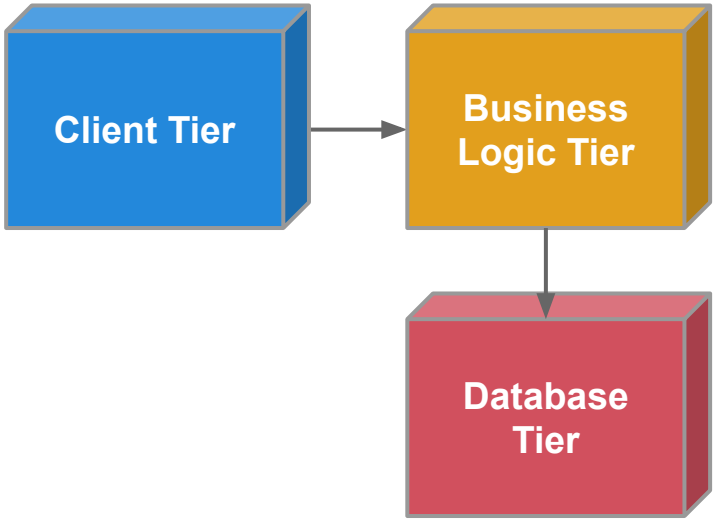


Tiers | Ejemplos

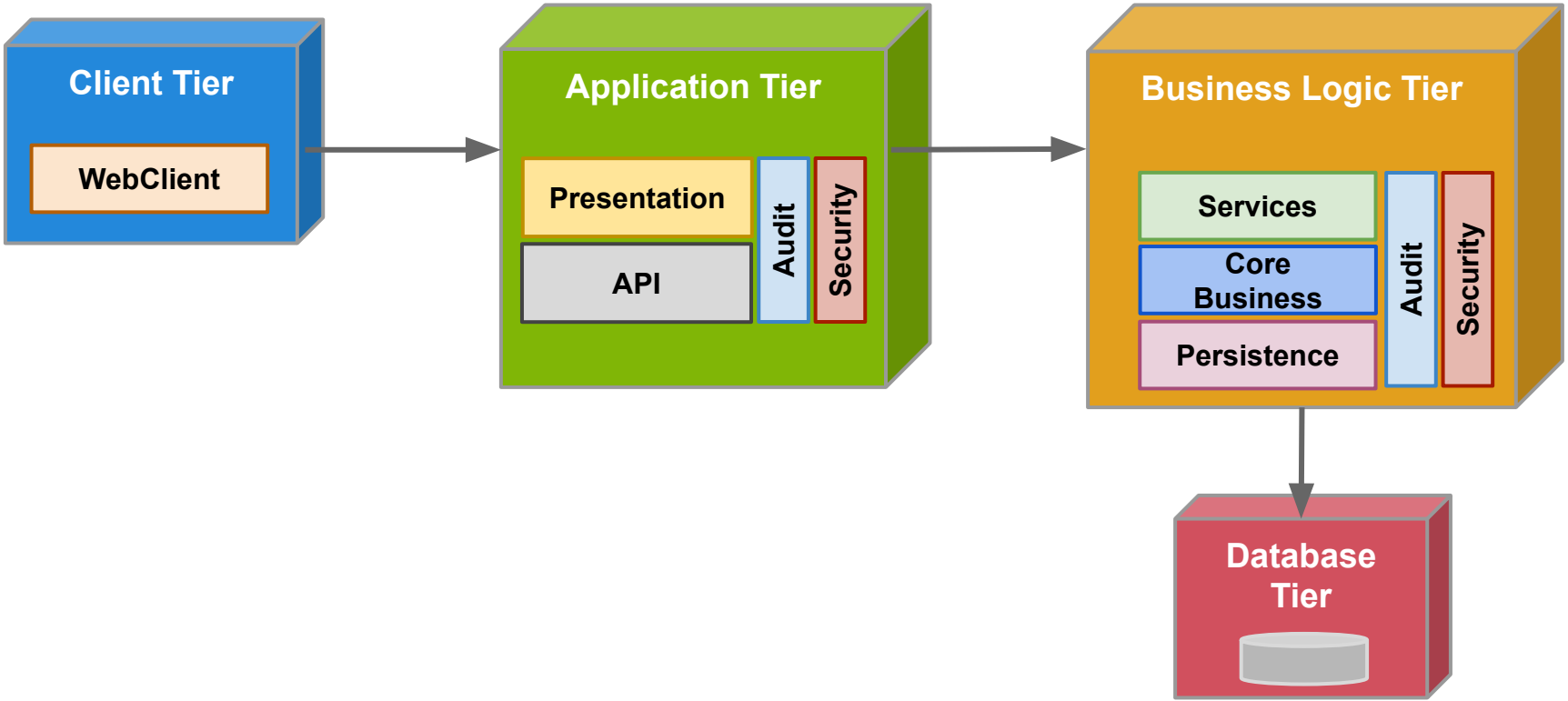
2-Tier Deployment



3-Tier Deployment



Tiers | Despliegue de *Layers*



Agenda

- Modelos de Capas: Layers y Tiers
- **Interfaces**
- Protocolos
- RESTful

Interfaces | Introducción

- Permiten la comunicación entre dos o más componentes/servicios/sistemas
- Diferentes contratos permiten diferentes clientes
- Sólo se expone **una parte** del sistema
- Esconden implementación
  - Implementación puede ser modificada sin cambiar contrato
  - Cambio de contrato implica un nueva versión

https://xkcd.com/1770/

Inter-Aplicaciones	Intra-Aplicaciones
Application Program Interfaces (APIs) Ej: • Cliente por consola consultando Web Server • JS de navegador consultando servidor • Servicio consultando Servicio	Facades - Mediators - Interfaces Ej: • *Layer* 2 consultando *Layer* a • Mensaje enviado a un objeto local • Mensaje enviado a un objeto remoto (?)

Interfaces | Problemas a resolver

- **Software es difícil de integrar**
  - No todos los componentes exponen interfaces útiles
  - Complejidad aumenta exponencialmente con la cantidad de componentes a integrar
- **Software es difícil de cambiar**
  - ¿Cómo cambiamos una API ya definida?
  - ¿Qué pasa un sistema está altamente acoplado a una API y esta cambia?

Orientados a entidades

- Desacoplamiento entre sistemas.
- Flexibilidad como objetivo.
- Admite extensiones a una funcionalidad no definida en su totalidad.

Orientados a procesos

- Componentes altamente acoplados.
- Alta performance como objetivo.
- Funcionalidad conocida y diversa.

Wikipedia/Wikimedia REST API

```
GET /page/title/{title}
POST /page/html/{title}
GET /page/media/{title}
GET /feed/featured/{yyyy}/{mm}/{dd}
```

```
POST /transform/html/to/wikitext/{title}
POST /media/math/check/tex
GET api.php?action=query&
    titles=Doesntexist%7CMain&format=json
```

Interfaces | Clasificación

Web APIs

- Web Services based APIs (HTTP+SOAP)
- REST based APIs

***Remote* APIs**

- Custom TCP/UDP services
- Object oriented: CORBA, JavaRMI
- Procedure oriented: RPC, gRPC

Library-based / Frameworks

- Java API
  - Ej.: OpenJDK vs Oracle JDK
- Android API

OS *related*

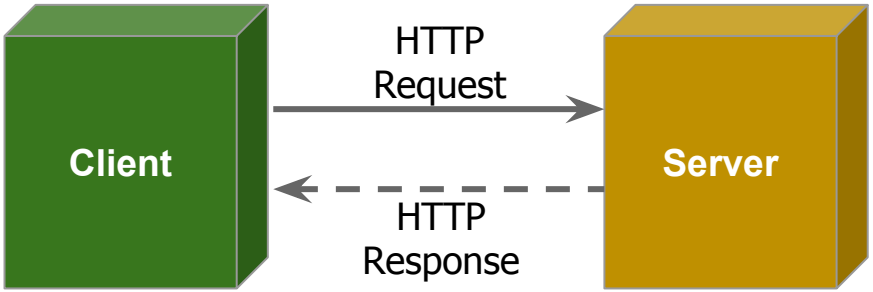
- POSIX
  - Ej.: Linux vs OpenBSD
- WinAPI

Agenda

- Modelos de Capas: Layers y Tiers
- Interfaces
- **Protocolos**
- RESTful

Características:

- Modelo *Client-Server*
- Modelo *Request-Reply*
- Sin estado



HTTP Request

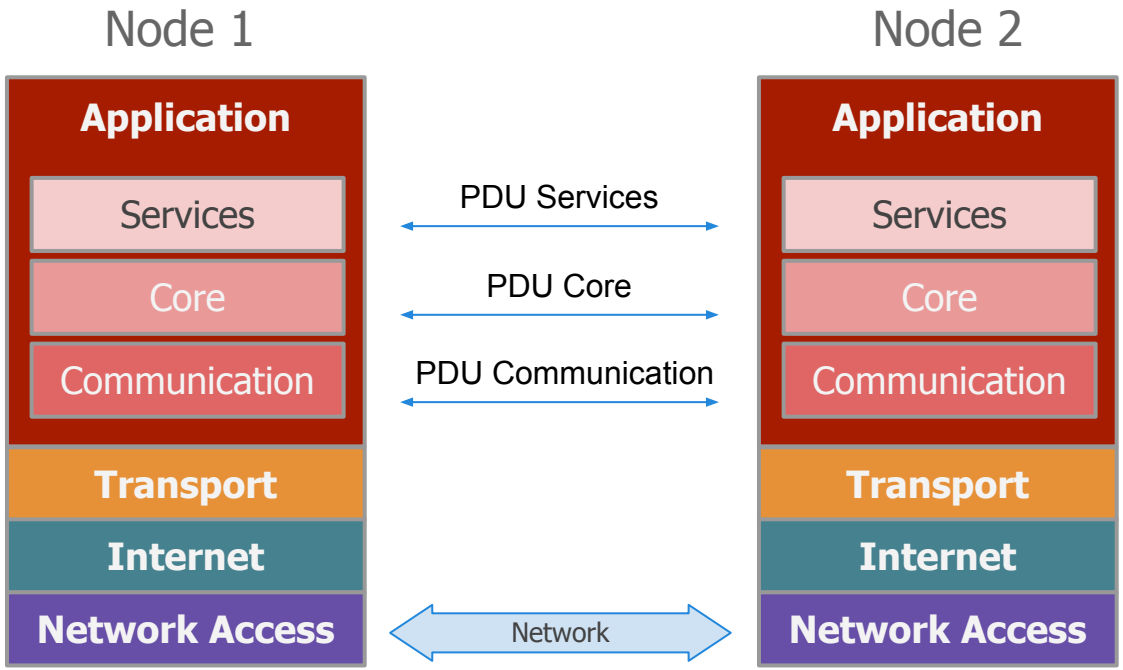
```
GET https://en.wikipedia.org/wiki HTTP/1.1
Host: www.example.com
Accept: text/html,application/xhtml+xml;
Accept-encoding: gzip, deflate, br
Accept-language: en-US,en;q=0.9,es;q=0.8
```

HTTP Response

```
HTTP/1.1 200 OK
Content-encoding: gzip
Content-language: en
Content-length: 18558
Content-type: text/html; charset=UTF-8


<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
...
</html>
```

Protocolos | Responsabilidades por capa

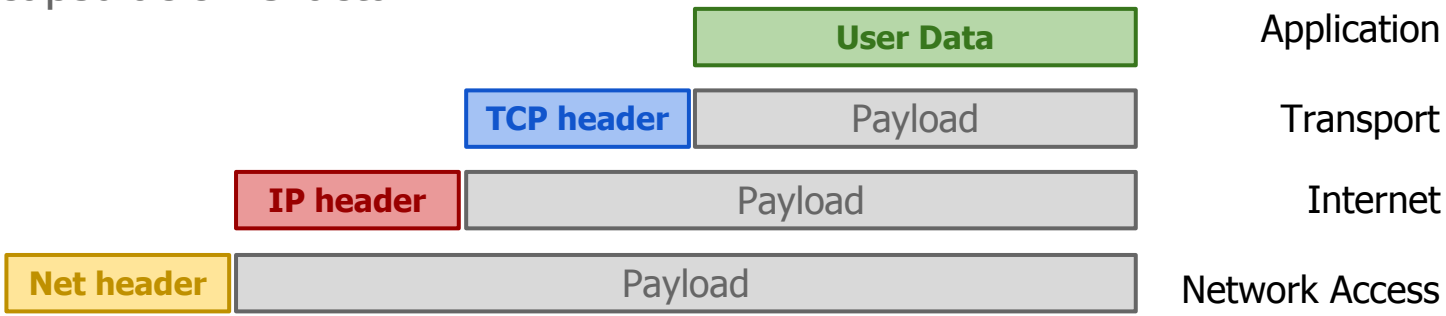


Protocolos | *Protocol Data Unit* (PDUs)

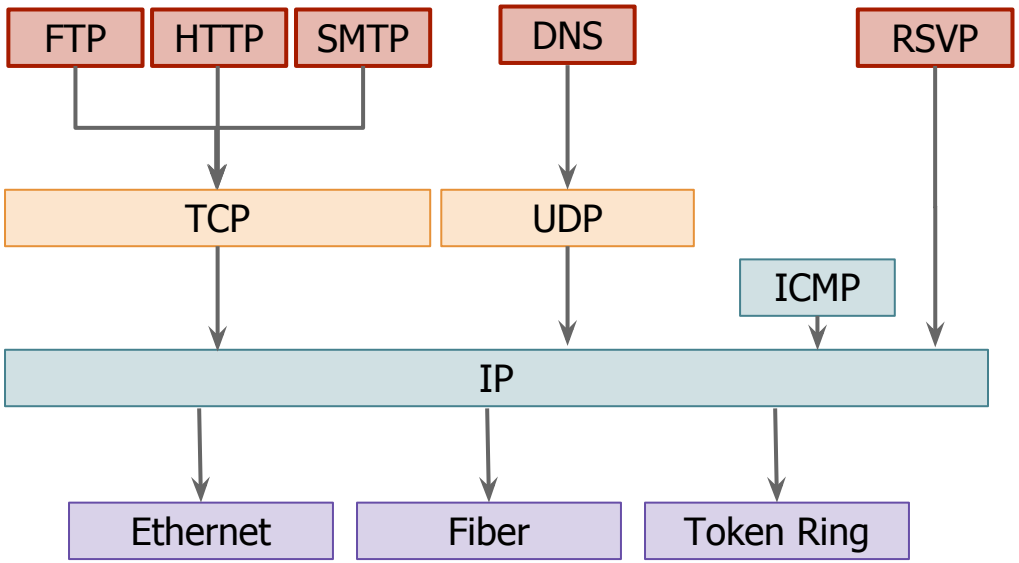
Encapsulación de PDUs entre capas:

1. Encapsulación exacta
2. Segmentación de paquetes
3. *Blocking* de paquetes

Ej. Encapsulación exacta:



Protocolos | Ejemplos



Application

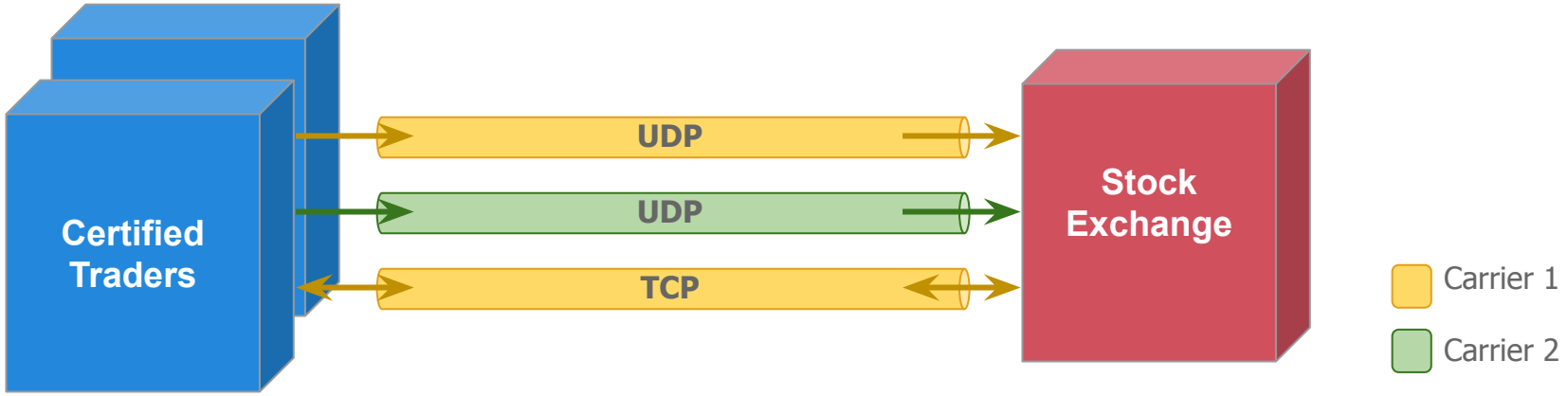
Transport

Internet

Network Access

Caso de Uso:

- Bolsa de comercio recibiendo información de muchos Operadores distribuidos
- Tráfico de paquetes con picos de transferencia
- Canales UDP para garantizar throughput (aún con repetidos)
- Canal TCP para pedir retransmisión



Agenda

- Modelos de Capas: Layers y Tiers
- Interfaces
- Protocolos
- **RESTful**

Representational State Transfer (aka REST/RESTful)

- Basado en entidades (*Web Resources*)
- Cada Web resource es representado por una **URL**
- **HTTP/HTTPS** utilizado como protocolo de comunicación
- **JSON/XML** utilizado como protocolo de serialización
- Cambio de estados a través de operaciones **CRUD**

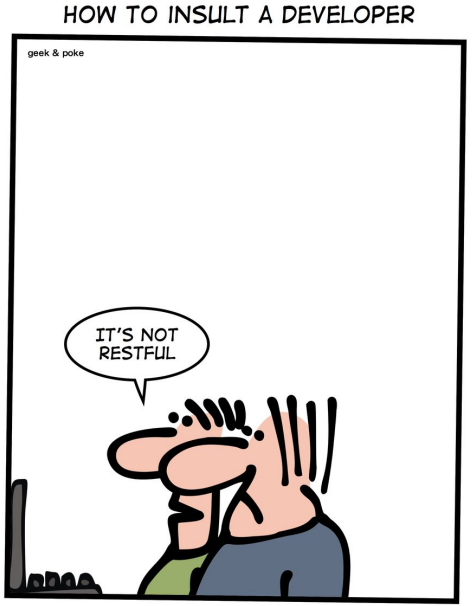
Operación	Verbo HTTP	Equivalente SQL
Create	POST	INSERT
Read	GET	SELECT
Update	PUT	UPDATE
Delete	DELETE	DELETE

Objetivos:

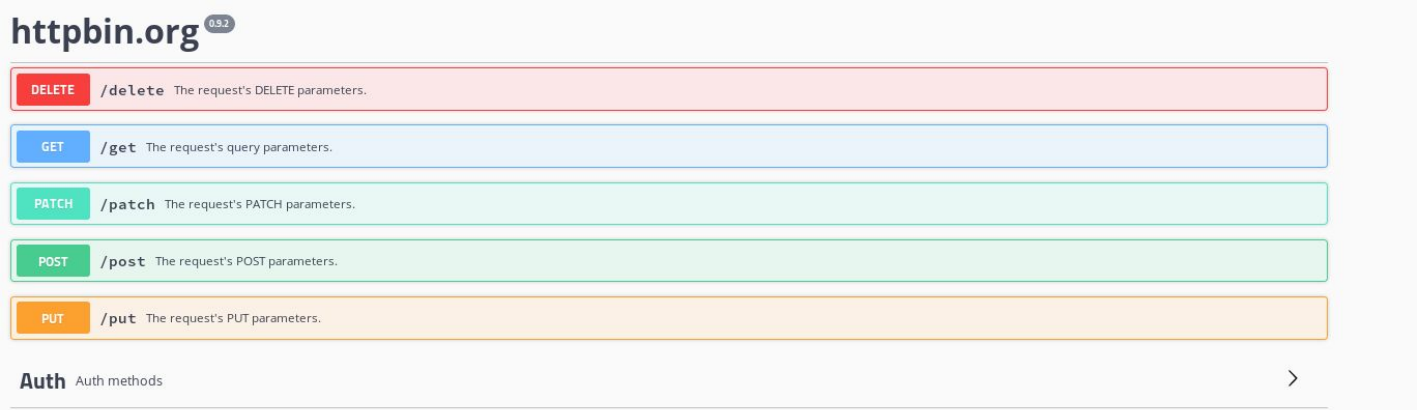
- Alta performance, escalabilidad, confiabilidad, etc.

Principios de Arquitectura:

- Arquitectura cliente/servidor
- *Cacheability*
- Interface Uniforme:
  - Hypermedia As The Engine Of Application State (HATEOAS)
- *Statelessness*
- *Layered system*



- *https://httpbin.org:*
  - Servicios de ejemplos RESTful online
- *https://github.com/flasgger/flasgger*
  - *Port* de Swagger para Python.
  - Genera interfaz de exploración y documentación de APIs.
  - Librería de documentación RESTful APIs utilizada por httpbin.org.



RESTful | Ejemplo | GitLab

- Plataforma *DevOps* incluyendo manejo de proyectos, control de versiones, *builds* y *deployments*.
- Ej. Entidades:
  - /projects
  - /groups
  - /users
  - /snippets
  - /runners
- Ej. Acciones:
  - /search
  - /lint

RESTful | Ejemplo | GitLab | Obtener Proyectos

```
$ curl --verbose -H Accept:application/json -X GET https://gitlab.com/api/v4/projects
> GET /api/v4/projects HTTP/1.1
> Host: gitlab.com
> Accept:application/json
>
...
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 2918
...
< [{ "id": 3472737,
     "description": "Inkscape vector image editor",
     "name": "inkscape",
     "web_url": "https://gitlab.com/inkscape/inkscape",
     "star_count": 1726, ...
     "namespace": { "id": 470642, "name": "Inkscape", "web_url": "https://gitlab.com/
groups/inkscape", ... }
   }, ... ]
```

RESTful | Ejemplo | GitLab | Obtener Referencias Anidadas

```
$ curl -X GET
https://gitlab.com/api/v4/projects/3472737/
users
...
< [ {
     "id": 43416, "name": "Felipe Sanches",
     "username": "fsanches",
     "state": "active", "avatar_url":
"https://secure.gravatar.com/avatar/3374412
c23252294142453a60b066300?s=80&d=identicon"
,
     "web_url":
"https://gitlab.com/fsanches"
    },
    ...
  ]
```

```
$ curl -X GET
https://gitlab.com/api/v4/users/43416
...
< { "id": 43416, "name": "Felipe Sanches",
    "username": "fsanches",
    "state": "active", "avatar_url":
"https://secure.gravatar.com/avatar/3374412
c23252294142453a60b066300?s=80&d=identicon"
,
  "web_url": "https://gitlab.com/fsanches",
  "created_at": "2014-07-21T19:32:00.713Z",
  "bio": null, "location": null,
  "public_email": "", "skype": "",
  "linkedin": "", "twitter": "",
  "website_url": "",
  "organization": null, "job_title": "",
  "work_information": null
}
```

RESTful | Ejemplo | GitLab | Crear Tags

```
$ curl -H Content-Type:application/json -X POST /projects/3472737/repository/tags
       --data '{"tag_name":"stable", "ref":"master"}'

{
"commit": {
    "id": "2695effb5807a22ff3d138d593fd856244e155e7",
    "short_id": "2695effb",
    "title": "Initial commit",
    "created_at": "2017-07-26T11:08:53.000+02:00",
    "parent_ids": [
      "2a4b78934375d7f53875269ffd4f45fd83a84ebe"
    ],
    "message": "Initial commit",
    "author_name": "John Smith",
  }, ...
  "name": "stable",
  "target": "2695effb5807a22ff3d138d593fd856244e155e7",
  ...
}
```

RESTful | Ejemplo | GitLab | Borrar un branch

```
$ curl --verbose -X DELETE
https://gitlab.com/api/v4/projects/3472737/repository/branches/issue-123
...
< HTTP/1.1 204 No Content
```

RESTful | Ejemplo | GitLab | *Status Codes*

200 OK	The GET, PUT or DELETE request was successful, the resource itself is returned as JSON.
204 No Content	The server has successfully fulfilled the request and that there is no additional content to send in the response payload body.
201 Created	The POST request was successful and the resource is returned as JSON.
304 Not Modified	Indicates that the resource has not been modified since the last request.
400 Bad Request	A required attribute of the API request is missing, e.g., the title of an issue is not given.
401 Unauthorized	The user is not authenticated, a valid user token is necessary.
403 Forbidden	The request is not allowed, e.g., the user is not allowed to delete a project.
404 Not Found	A resource could not be accessed, e.g., an ID for a resource could not be found.
405 Not Allowed	The request is not supported.
409 Conflict	A conflicting resource already exists, e.g.creating a project with a name that already exists
412	Indicates the request was denied. May happen if the If-Unmodified-Since header is provided when trying to delete a resource, which was modified in between.
422 Unprocessable	The entity could not be processed.
500 Server Error	While handling the request something went wrong server-side.

Identificar una unívocamente una entidad entre diferentes sistemas es una prioridad

- Identificador debería tener información sobre la identidad que referencia

Identidad != Nombre

- Nombre puede cambiar, identidad no (Ej. https://tracker.com/pet?name=Lassie)
- URL define la identidad de la entidad, no el identificador (Ej. https://tracker.com/pet/12345)

```
// Pet entity. Bad ID
{
 "id": "12345",
 "type": "/dog",
 "name": "Lassie"
}

// Pet entity. Better ID
{
 "id": "/pet/12345",
 "type": "/dog",
 "name": "Lassie"
}
```

- Identificar correctamente una entidad ayuda a la integración de sistemas
- **Integración con otros sistemas**
  - Siempre usar URIs para identificar entidades
  - URIs pueden ser relativas si la entidad es del mismo sistema

```
// Bad relationship
{
 "id": "12345",
 "type": "/dog",
 "name": "Lassie",
 "owner": "98765",
}
```

```
// Better relationship
{
 "id": "/pet/12345",
 "type": "/dog",
 "name": "Lassie",
 "owner": "/owner/98765",
}
```

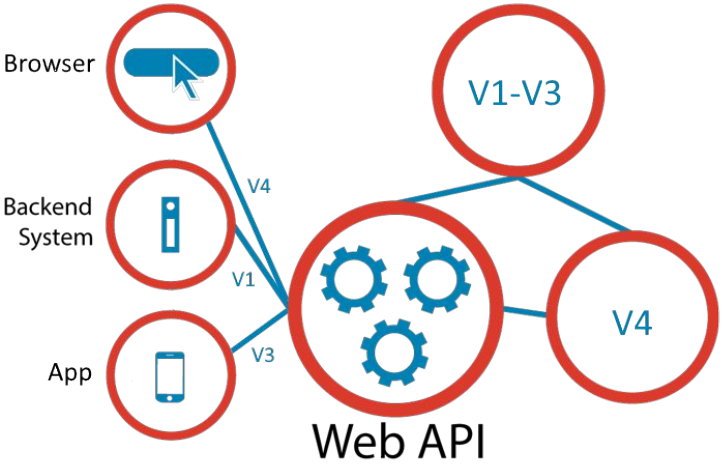
Semantic Versioning (semver):

- Estándar más utilizado para definir versiones de APIs y Librerías.
- Foco en brindar información de retrocompatibilidad de la interfaz.
- Ej.:

1.7.1

Major . Minor . Patch

- **Incremento de números de versión:**
  - **Major:** al introducir cambios incompatibles con la versión anterior.
  - **Minor:** al agregar funcionalidad pero mantener retrocompatibilidad
  - **Patch (aka build):** al introducir correcciones que no afectan la interfaz.



- **Tipos**
  - **Versionado explícito en URL**
    - Fácil de usar y testear
    - RESTful compliant (Interface uniforme)
  - **Versionado via HTTP Custom header**
    - Semánticamente incorrecto.
  - **Versionado via HTTP Accept header**
    - Semánticamente correcto. Indica cómo se quiere obtener el recurso
    - Díficil de testear

- **Versionado de objetos != Versionado de API**
- Tipos de versiones de objetos:
  - *Format Versioning*: la API pueden brindar distintas representaciones de **la misma entidad** y así cambiar su formato.
    - Ej.1 : Invocaciones desde distintas versiones de una misma API
    - Ej.2 : Al filtrar de datos del objeto resultante
  - *Historical Versioning*: la misma entidad tiene distintas versiones que fueron almacenadas a lo largo de su ciclo de vida.
    - Ej. : API de páginas en una Wiki

```
GET /api/pages/{id}/{rev}
GET /api/pages/{id}
GET /api/pages/{id}/latest
GET /api/pages/{id}?version={rev}
```

RESTful | Resumen

- Standard de facto de la industria
- Basado en entidades y cambios de estado
  - Ideal manejo de recursos
  - Complejo para ejecución remota de lógica de negocio
- La teoría es importante. También lo es ser pragmático: hay que encontrar un equilibrio
- ***Don't mess it up with the basics*!**
  - Utilizar HTTP/JSON directamente
  - Identificar unívocamente entidades con URLs (no con nombres)
  - Manejar correctamente versionado y queries

Bibliografía

- M. Van Steen, A. Tanenbaum: Distributed Systems. 3rd Edition. Pearson Education, 2017.
  - Capítulo 2.1 - Architectures, Architectural Styles
  - Capítulo 4 - Communication
- W. Stallings, Data and Computer Communication, 8th Edition, 2006.
  - Capítulo 2 - Protocol Architecture,TCP/IP, and Internet-Based Applications
- Microsoft Application Architecture Guide, 2nd Edition. 2009.
  - Capítulo 5: Layered application guidelines
  - Capítulo 19: Physical tiers and deployment
- Martin Nally. Designing Quality APIs. 2018.
  - https://www.youtube.com/watch?v=P0a7PwRNLVU