Almacenamiento de objetos

Datos accesibles por HTTP a través de una API REST

📧 José Domingo Muñoz

🏫 IES Gonzalo Nazareno · Dos Hermanas

📚 SRI · Servicios de Red e Internet

01

¿Por qué otro tipo de almacenamiento?

El problema que resuelve

Los datos de hoy

Las aplicaciones actuales generan y sirven enormes cantidades de **datos no estructurados**:

- **Fotos y vídeos** que suben los usuarios de una red social o una tienda *online*
- **Copias de seguridad** de servidores y equipos
- **Logs**, métricas y ficheros de auditoría
- **Documentos**, adjuntos, paquetes de *software*, imágenes de contenedores…

Y además:

- Hay que acceder a ellos **desde cualquier sitio**: navegadores, móviles, otras aplicaciones
- El volumen **no cabe en un servidor** y no para de crecer
- No se pueden **perder**, aunque falle un disco o un servidor entero

¿Nos sirven NFS o iSCSI?

LO QUE OFRECEN

- Un sistema operativo **monta** el recurso
- Se trabaja con **ficheros** (NFS) o **bloques** (iSCSI)
- Pensados para una **red local** y unos pocos clientes
- Un servidor (o una cabina) con una **capacidad limitada**

LO QUE NECESITAMOS

- Que el cliente sea un **navegador** o una **aplicación**, no un sistema que monta discos
- Acceso **por Internet**, atravesando cortafuegos y *proxies*
- **Millones** de clientes a la vez
- Crecer **añadiendo servidores**, sin límite práctico

ℹ️ El almacenamiento de objetos no sustituye a NAS ni a SAN: resuelve **otro problema**.

La idea en una frase

- El almacenamiento de objetos es almacenamiento **al que se accede como a una web**: no se monta, se le hacen **peticiones HTTP** a través de una **API**
- Cada dato guardado tiene su propia **URL**
- Para guardarlo, leerlo o borrarlo se usan los **métodos HTTP** que ya conocemos
- Detrás de esa URL puede haber **cientos de servidores**: el cliente no lo sabe ni le importa

02

Conceptos

Objetos, contenedores y metadatos

¿Qué es un objeto?

Identificador

el nombre (o *clave*) del objeto

`fotos/playa.jpg`

Datos

el contenido: una imagen, un vídeo, un *backup*…

Metadatos

información sobre los datos

tipo, tamaño, fecha, *hash*, propios…

- Un **objeto** es una unidad de información: sus datos más sus metadatos, con un identificador único
- Los metadatos los pone el sistema (tamaño, fecha, `Content-Type`…) y también **el usuario** (`autor`, `proyecto`…)
- Un objeto **no se modifica por partes**: se sube entero y, si cambia, se **sustituye** entero

Contenedores

> *Los objetos se guardan en **contenedores** (en S3 se llaman **buckets**).*

- Un contenedor agrupa objetos, como un directorio… pero **no se pueden anidar**
- Dentro de un contenedor el espacio de nombres es **plano**: no hay carpetas de verdad
- Un nombre como `fotos/2026/playa.jpg` es **un único nombre** que contiene barras
- Las interfaces muestran «carpetas» agrupando por ese **prefijo**, pero es solo una forma de verlo
- Los **permisos** se suelen dar a nivel de contenedor

ℹ️ En OpenStack Swift la jerarquía es **cuenta → contenedor → objeto**. La cuenta corresponde a un **proyecto** de OpenStack.

Ficheros, bloques y objetos

	BLOQUES (SAN)	FICHEROS (NAS)	OBJETOS
El cliente ve…	Un disco	Carpetas y ficheros	Contenedores y objetos
¿Quién es el cliente?	Un sistema operativo	Un sistema operativo	Una aplicación, un navegador
Cómo se accede	iSCSI, FC	NFS, SMB	**HTTP** (API REST)
Organización	Bloques	Árbol de directorios	Espacio **plano** + metadatos
Modificar	Cualquier bloque	Cualquier parte del fichero	Objeto **entero**
Escala	Una cabina	Un servidor	Muchos servidores

03

HTTP y la API REST

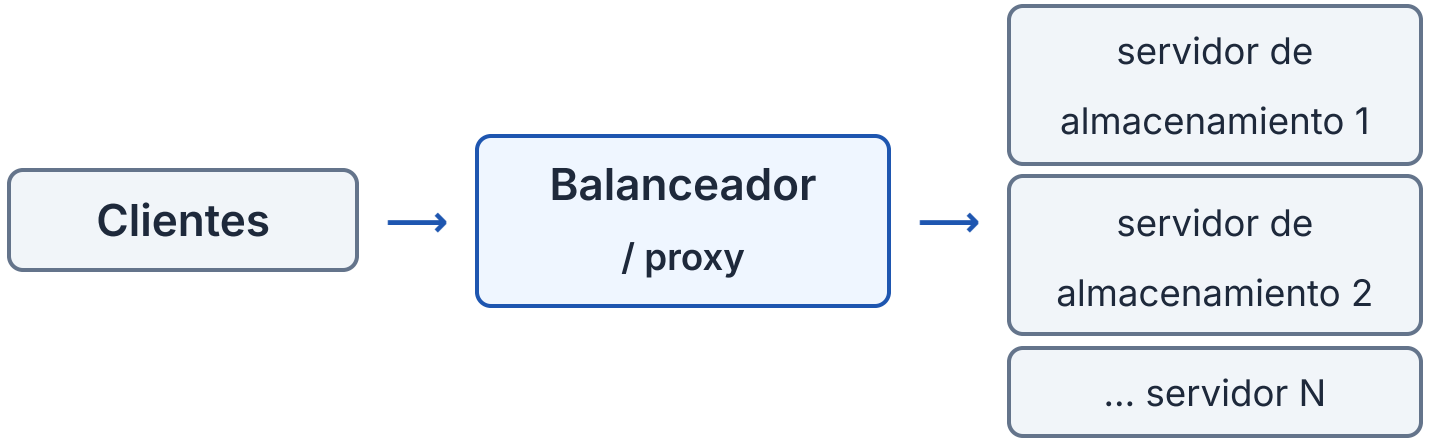
El corazón del almacenamiento de objetos

¿Por qué HTTP?

- **Todos lo hablan**: navegadores, `curl`, cualquier lenguaje de programación, cualquier móvil
- **Atraviesa la red**: pasa por cortafuegos, *proxies* y balanceadores sin configuración especial
- **Cada objeto es una URL**: se puede enlazar desde una página web o compartir
- Los **metadatos** viajan en las **cabeceras** (`Content-Type`, `Content-Length`, `ETag`...)
- Los **códigos de estado** indican el resultado de cada operación
- Con **HTTPS**, la comunicación va cifrada

✅ No hay que inventar un protocolo nuevo ni instalar nada especial en el cliente: **HTTP ya lo tiene todo**.

HTTP es *stateless*: por eso escala



- Cada petición lleva **todo lo necesario** (qué objeto, qué operación, quién la pide)
- Cualquier servidor puede atender cualquier petición: basta con **añadir servidores**
- Cada objeto se guarda **varias veces** en servidores distintos: si uno falla, no se pierde nada
- Es la misma idea del **balanceador de carga** que vimos con los servidores web

¿Qué es una API?

> *Una **API** (Application Programming Interface) es la forma que ofrece un servicio para que **otros programas** lo usen.*

- Define **qué operaciones** se pueden hacer y **cómo pedirlas**
- La aplicación no necesita saber cómo funciona el servicio por dentro
- En el almacenamiento de objetos, la API es **la única forma de entrar**: no hay otra

ℹ️ Una **API web** es una API a la que se accede con peticiones HTTP. Las respuestas no son páginas para personas, sino datos para programas (texto, JSON, cabeceras).

API REST

REST es una forma de diseñar APIs web aprovechando HTTP tal cual:

- Todo lo que se gestiona es un **recurso**: una cuenta, un contenedor, un objeto
- Cada recurso se identifica con una **URL**
- La operación la indica el **método HTTP**
- El resultado lo indica el **código de estado**
- Cada petición es **independiente**: no hay sesión en el servidor (*stateless*)

```
https://openstack.gonzalonazareno.org/swift/v1/AUTH_<proyecto>/fotos/playa.jpg
└──────────────────── servicio ──────────────────┘└─ cuenta ─┘└contenedor┘└─objeto─┘
```

Métodos HTTP = operaciones

MÉTODO	SOBRE UN CONTENEDOR	SOBRE UN OBJETO
`PUT`	Crearlo	**Subirlo** (o sustituirlo)
`GET`	**Listar** sus objetos	**Descargarlo**
`HEAD`	Ver sus metadatos	Ver sus **metadatos** sin descargarlo
`POST`	Cambiar metadatos y permisos	Cambiar sus metadatos
`DELETE`	Borrarlo (si está vacío)	**Borrarlo**

ℹ️ Son los métodos que vimos en el tema de HTTP. Aquí `PUT` y `DELETE`, que casi no se usan en las webs, son **imprescindibles**.

Códigos de estado = resultado

CÓDIGO	SIGNIFICADO EN EL ALMACENAMIENTO DE OBJETOS
`200 OK`	Descarga o listado correctos
`201 Created`	Objeto o contenedor creado
`202 Accepted`	Metadatos de un objeto actualizados (`POST`)
`204 No Content`	Operación correcta sin cuerpo: borrado, metadatos de un contenedor...
`401 Unauthorized`	Falta el token o no es válido
`403 Forbidden`	El token es válido, pero no hay permiso
`404 Not Found`	El contenedor o el objeto no existe
`409 Conflict`	Se intenta borrar un contenedor que no está vacío

Una petición real sin autenticar

```
$ curl -i https://openstack.gonzalonazareno.org/swift/v1/AUTH_prueba
HTTP/1.1 401 Unauthorized
Server: nginx/1.22.1
Content-Type: text/html; charset=UTF-8
Www-Authenticate: Swift realm="AUTH_prueba"
WWW-Authenticate: Keystone uri="http://172.22.123.100/identity"
X-Trans-Id: tx28d1aefad2d44697a661a-006aeae3aa
```

- Es una **petición HTTP normal**, hecha con `curl`
- El servidor responde con un **código de estado** (401) y unas **cabeceras**
- La cabecera `WWW-Authenticate` nos dice **dónde** conseguir credenciales: Keystone
- Delante de Swift hay un **proxy inverso** (nginx)

Autenticación con *token*

1. Usuario y contraseña al servicio de identidad (Keystone)
2. Keystone devuelve un **token** y la URL del almacenamiento
3. Cada petición lleva el token en una cabecera

```
GET /swift/v1/AUTH_<proyecto>/fotos HTTP/1.1
Host: openstack.gonzalonazareno.org
X-Auth-Token: gAAAAABp...
```

- El token **caduca** (por defecto, a la hora): después hay que pedir otro
- No hay cookies ni sesión: el token va **en cada petición** (*stateless*)
- Se usa HTTPS para que nadie pueda **capturar** el token

Privado o público

POR DEFECTO: PRIVADO

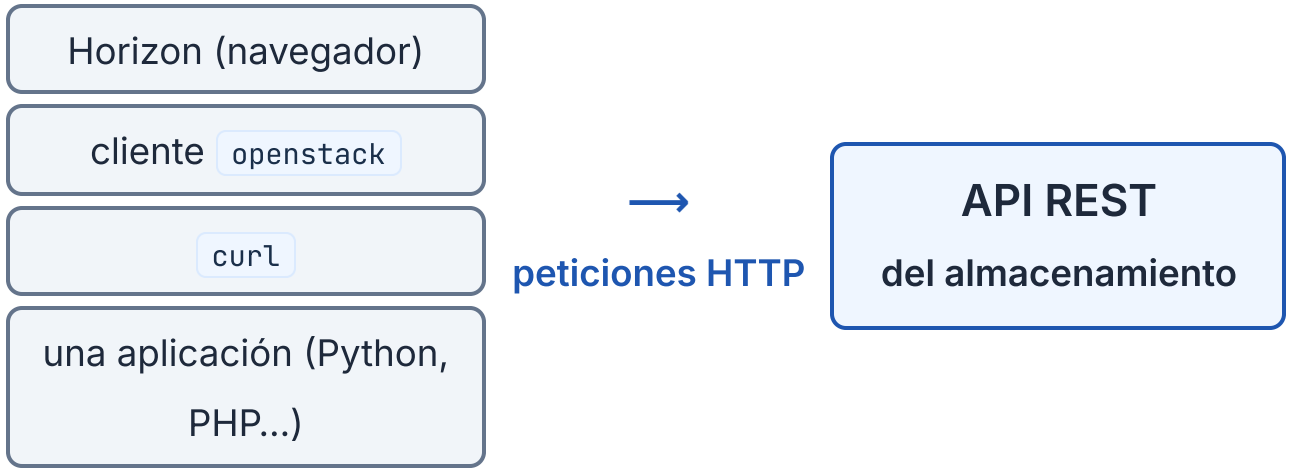
- Todo es **privado**: hace falta un token
- Solo los usuarios del **proyecto** pueden ver y modificar

CONTENEDOR PÚBLICO

- Se le da una **ACL de lectura pública** (`.r:*`)
- Cualquiera puede **descargar** sus objetos **sin token**
- Ya se puede poner la URL en un `<img src="…">` de una página web
- Sigue haciendo falta token para **subir o borrar**

⚠️ Muchas fugas de datos famosas han sido **buckets públicos por error**. Hazlo público solo cuando lo necesites.

Muchos clientes, una misma API



- La **interfaz web**, el **cliente de línea de comandos** y `curl` hacen **exactamente lo mismo**
- Todos acaban enviando las **mismas peticiones HTTP** a la API
- Lo que cambia es la **comodidad**: Horizon y `openstack` construyen las peticiones por nosotros

04

El almacenamiento de objetos hoy

Usos y servicios

¿Para qué se usa?

CONTENIDO DE WEBS Y APLICACIONES

- Fotos y vídeos que suben los usuarios
- Imágenes, CSS y JavaScript de una web
- **Webs estáticas** completas
- **Origen** de una CDN

COPIAS DE SEGURIDAD Y ARCHIVO

- Copias **fuera** de la organización
- Datos que se guardan años y casi no se leen

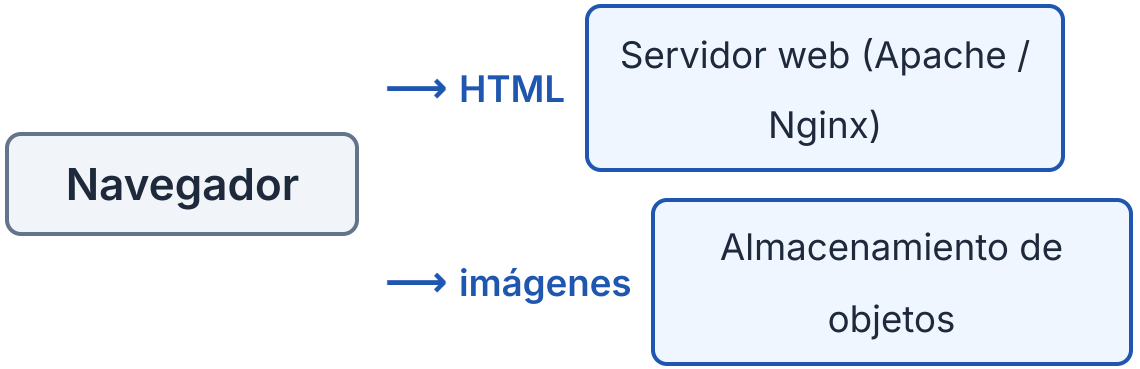
DATOS A GRAN ESCALA

- **Logs** y métricas
- *Data lakes*: datos para análisis
- Conjuntos de datos para entrenar modelos de **IA**

INFRAESTRUCTURA

- **Imágenes** de máquinas virtuales y de contenedores
- Distribución de **paquetes** de *software*
- Almacenamiento de aplicaciones como Nextcloud

Un ejemplo: una web con imágenes




```
<img src="https://openstack.gonzalonazareno.org/swift/v1/AUTH_<proyecto>/web/logo.png">
```

- El servidor web solo entrega el **HTML**: las imágenes las pide el **navegador** directamente al almacenamiento
- El servidor web no guarda ni sirve los ficheros pesados
- Si hay varios servidores web, todos usan las **mismas** imágenes

Servicios y *software*

	PRODUCTO	API
Nube pública	Amazon S3	S3
	Azure Blob Storage	Propia
	Google Cloud Storage	Propia (y compatible con S3)
Instalación propia	**OpenStack Swift**	Swift (opcional, compatible con S3)
	Ceph (RADOS Gateway)	S3 y Swift
	Garage, SeaweedFS…	S3

ℹ️ La API de **S3** se ha convertido en el estándar de facto. Las APIs cambian en los detalles, pero la idea es siempre la misma: **recursos, URLs y métodos HTTP**.

Nuestro escenario: OpenStack Swift

- Swift es el servicio de **almacenamiento de objetos** de OpenStack
- Lo tenemos en la nube del instituto: `openstack.gonzalonazareno.org`
- Cada **proyecto** de OpenStack tiene su **cuenta** de Swift
- Lo usaremos de tres formas:

HORIZON

La interfaz web de OpenStack

`OPENSTACK`

El cliente de línea de comandos

`CURL`

Peticiones HTTP a mano a la API

05

Resumen

Lo que tienes que saber

- El almacenamiento de objetos guarda **objetos**: datos + metadatos + identificador
- Los objetos se organizan en **contenedores** con un espacio de nombres **plano**
- No se monta: se accede por **HTTP** a través de una **API REST**
- En la API, cada recurso es una **URL**, cada operación un **método** y cada resultado un **código de estado**
- La autenticación va en un **token** que se envía en cada petición
- Por defecto es **privado**; un contenedor público se puede leer sin token
- Se usa para contenido web, *backups*, archivo, datos masivos e infraestructura
- **S3** es el estándar de facto; nosotros usaremos **OpenStack Swift**

Para profundizar

- Documentación de OpenStack Swift
- API de Object Storage de OpenStack
- Cliente `openstack`: comandos de *object store*
- Amazon S3: ¿qué es el almacenamiento de objetos?

SRI · UNIDAD 2 — ALMACENAMIENTO DE OBJETOS

¡Gracias!

Almacenamiento de objetos

📧 José Domingo Muñoz 🏫 IES Gonzalo Nazareno · Dos Hermanas 📚 SRI