Tema 7 - Epílogo

Tecnologías de los Sistemas de Información en la Red

Índice

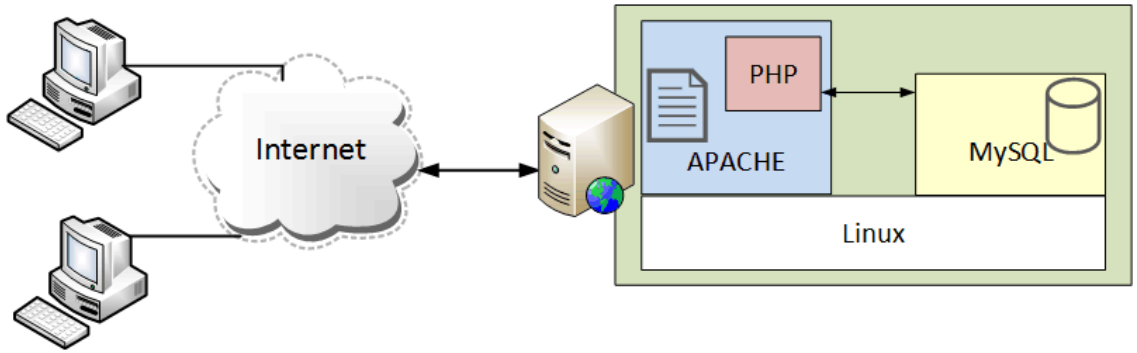
Objetivos

- Repasar y relacionar los contenidos desarrollados en el curso
- Concretar las piezas con las que se puede contar en la actualidad para crear aplicaciones distribuidas, especialmente en el ámbito de aplicaciones web
- Motivar las plataformas en la nube como mejora para sustentar el funcionamiento de aplicaciones distribuidas.

Índice

1. Revisitando la Wikipedia

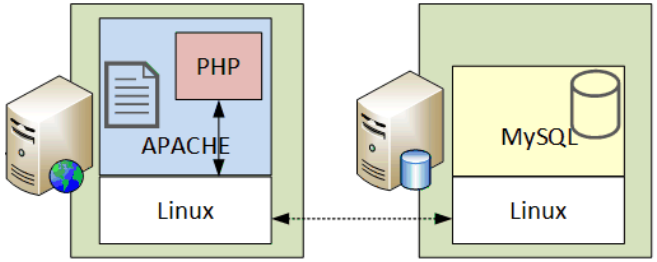
- Recordamos que la Wikipedia es un sistema LAMP



  - Concretamente, una aplicación web
- Necesidad de ancho de banda imposible para una única conexión: **múltiples** servidores
  - Problema para disponer de un mismo punto de acceso
  - Problema de sincronización entre ellos
  - Problema de disponibilidad ante fallos
  - Incremento del rendimiento

1. Revisitando la Wikipedia

- La separación de los grandes servicios internos en la Wikipedia es una tarea sencilla
  - PHP necesita APACHE
  - El SGBD MySQL es un servicio independiente



- Esta modificación es insuficiente porque produce piezas demasiado grandes: hay que rediseñar(*) la aplicación con componentes que posean…
  - Alta cohesión
  - Bajo acoplamiento
- También se debe replicar el sistema para facilitar la disponibilidad geográfica

(*) las habilidades para esto están fuera de nuestros objetivos

1. 1 Replicación geográfica de la Wikipedia

Mediante la replicación se persigue…

- *… incrementar el rendimiento*
- *… mejorar la tolerancia a fallos*

… y por ser geográfica…

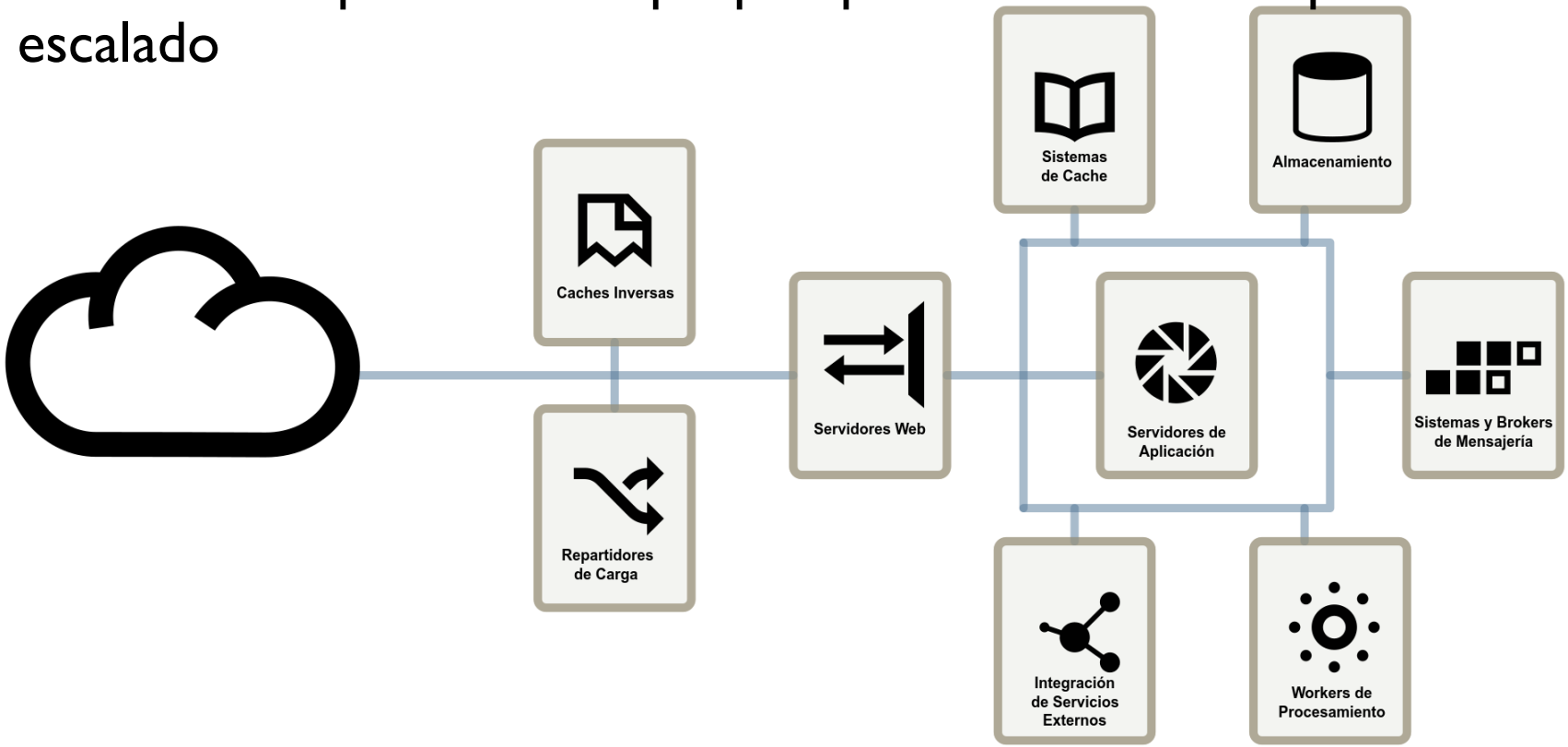
- *reducir la latencia respecto a los clientes*

- Eqiad: servidores de aplicación (primarios)
- Codfw: servidores de aplicación (secundarios)
- Esams, Ulsfo y Eqsin son CPDs para *caching*

Índice

2. Bloques constructivos de una **aplicación web**

Las aplicaciones web actuales combinan componentes, cada uno de los cuales presenta sus propias particularidades respecto al escalado



Fuente: Concurrent Programming for Scalable Web Architectures

2.1 Servidor web

Estrategia de escalado	La solución básica para servidores web es la replicación. Si los servidores no mantienen estado se facilita el escalado
Ejemplos reales	El servidor APACHE es el más popular. nginx y lighttpd destacan por mejorar la escalabilidad
Ejemplos en la nube	Google App Engine (*GAE*) emplea internamente Jetty, programado en Java

2.2 Servidores de aplicación

Estrategia de escalado	Los servidores de aplicación no deben compartir recursos directamente para facilitar el escalado: • Compartiendo estado, la escalabilidad se volverá muy difícil y compleja • Solución: servicios externos para la coordinación y comunicación entre varios servidores de aplicación
Ejemplos reales	Los más populares incluyen lenguajes de scripting, como Ruby (on Rails), PHP o Python. Contenedores de aplicación populares para Java, como JBoss (RedHat) y GlassFish (Oracle)
Ejemplos en la nube	GAE y Elastic Beanstalk (Amazon) soportan servlets (Java). GAE puede soportar también aplicaciones basadas en Python y Go

2.3 Repartidores de carga y caches inversas

Estrategia de escalado	Es fácil clonar repartidores de carga, pero existen varias opciones para reequilibrar su carga tras la replicación. • Una forma popular consiste en disponer de múltiples servidores bajo un mismo nombre DNS. El servicio de caches inversas es fácilmente paralelizable
Ejemplos reales	Repartidores de carga: HAProxy, perlbal y nginx. Proxies con funcionalidades cache: Varnish y nginx
Ejemplos en la nube	ELB (Amazon) es un servicio dedicado para este reparto de carga.

2.4 Sistemas de mensajería

Estrategia de escalado	Una infraestructura descentralizada puede proporcionar mejor escalabilidad Los sistemas de mensajería con bróker necesitan alternativas de escalado elaboradas, incluyendo el particionamiento de los participantes y la replicación de los brokers
Ejemplos reales	AMQP es un protocolo de mensajería con varias implementaciones maduras, como RabbitMQ ØMQ destaca como sistema de mensajería sin broker y descentralizado
Ejemplos en la nube	Amazon ofrece una solución de colas de mensajes (SQS) GAE dispone de una solución basada en colas para el manejo de tareas en segundo plano, y un servicio de mensajería XMPP • Estos servicios incurren en grandes latencias para el envío de mensajes, y no pueden emplearse en el procesamiento de peticiones HTTP • Varias arquitecturas basadas en EC2 han desarrollado su infraestructura de mensajería sobre alguno de los anteriores productos, como ØMQ

2.5 Almacenamiento de datos en el *backend*

Estrategia de escalado	Es difícil lograr el escalado del almacenamiento persistente. • Las aproximaciones habituales son el particionado vertical de datos y el sharding (horizontal).
Ejemplos reales	MySQL es un SGBD con soporte para clustering. Entre los SGBDs escalables no relacionales encontramos Riak, Cassandra y Hbase. Los sistemas de ficheros distribuidos más empleados en arquitecturas web de gran escala son HDFS, GlusterFS y MogileFS.
Ejemplos en la nube	GAE ofrece almacenamiento de datos y almacenamiento *blob*. Amazon ofrece varias soluciones en el ámbito de datos en la nube (p.e. RDS, DynamoDB, SimpleDB) y almacenamiento de ficheros (p.ej S3).

2.6 Sistema de cache

Estrategia de escalado	Una cache distribuida es, esencialmente, una memoria tipo clave/valor. • Por ello se puede conseguir escalado vertical añadiendo más memoria central. Se obtiene mayor escalado uniendo clonado y replicación al particionado del espacio de claves.
Ejemplos reales	Memcached es una cache ditribuida de gran popularidad. Redis soporta tipos de datos estructurados y canales de publicación/subscripción.
Ejemplos en la nube	GAE soporta el API de memcache. Amazon ofrece una solución de cache denominada ElastiCache.

2.7 Sistema de procesamiento en *background*

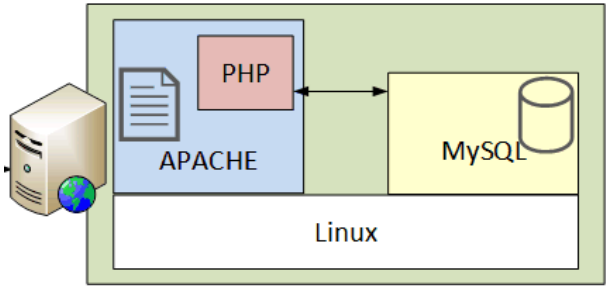
Estrategia de escalado	Añadir más recursos y nodos a la bolsa de trabajadores suele conseguir… • un incremento en la computación realizada • o un aumento en la capacidad para ejecutar más tareas concurrentes, gracias al paralelismo. Es más sencillo escalar bolsas de trabajadores cuando se trata de tareas reducidas, aisladas y sin dependencias.
Ejemplos reales	Hadoop es una implementación de la plataforma MapReduce para la ejecución paralela de ciertos algoritmos sobre grandes conjuntos de datos. Storm (Twitter) es un sistema de computación distribuida para datos en tiempo real orientado al procesamiento de streams, entre otros. Spark (APACHE) es un framework para el análisis de datos, diseñado para cluster en memoria.
Ejemplos en la nube	GAE dispone de un API para colas de tareas, útil para enviar tareas a un conjunto de trabajadores. Elastic MapReduce es un servicio de Amazon basado en MapReduce.

2.8 Integración de servicios externos

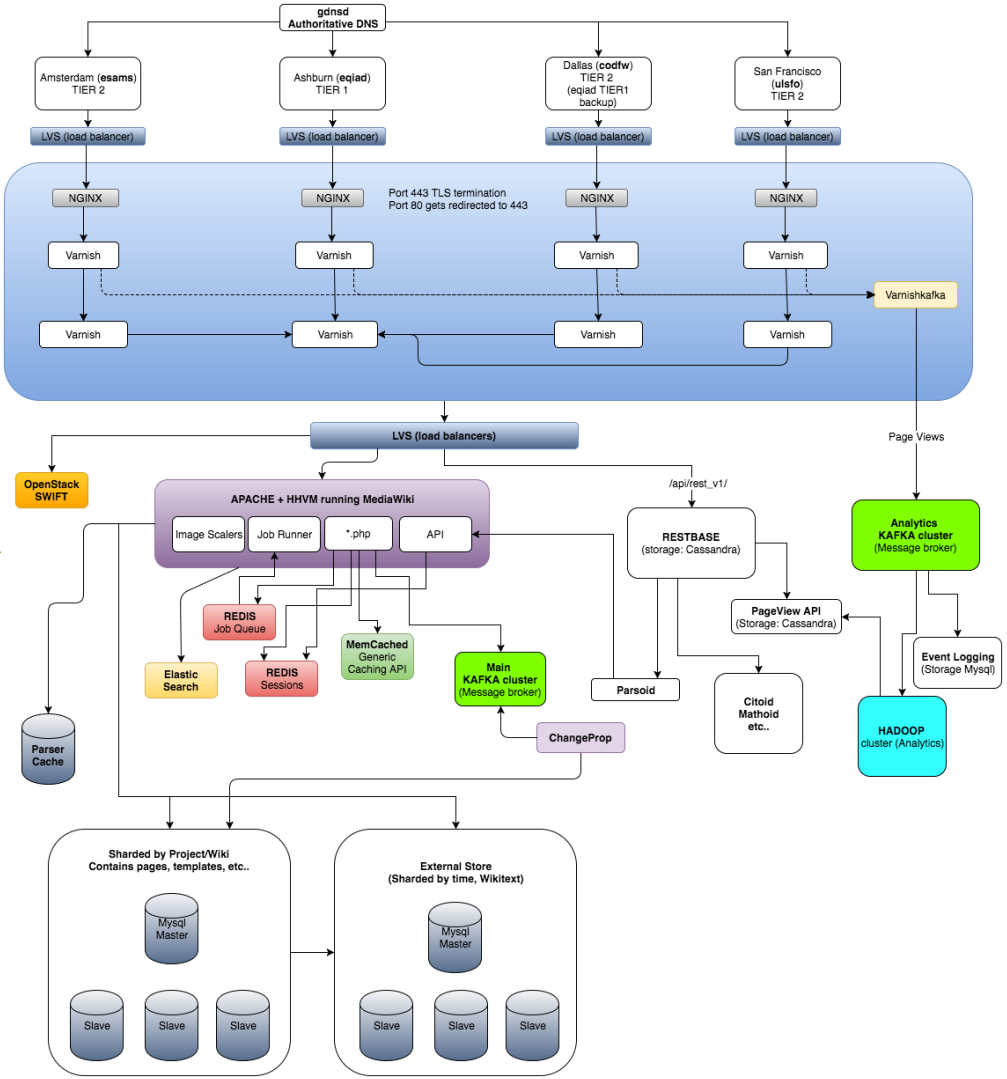
Estrategia de escalado	La escalabilidad de los servicios externos depende principalmente de su propio diseño e implementación. Para la integración, es útil centrarse en patrones de comunicación sin estado, escalables y de acoplamiento débil.
Ejemplos reales	Mule y Apache ServiceMix son dos productos de código libre que proporcionan *Enterprise Service Bus* y capacidades de integración.
Ejemplos en la nube	Los productos citados disponen de mecanismos de integración para servicios externos únicamente a bajo nivel. GAE permite acceder a recursos web externos mediante el API URLFetch. Como mejora se puede usar el API XMPP. Los servicios de mensajería de Amazon pueden usarse para la integración.

Índice

3. La Wikipedia "a piezas"

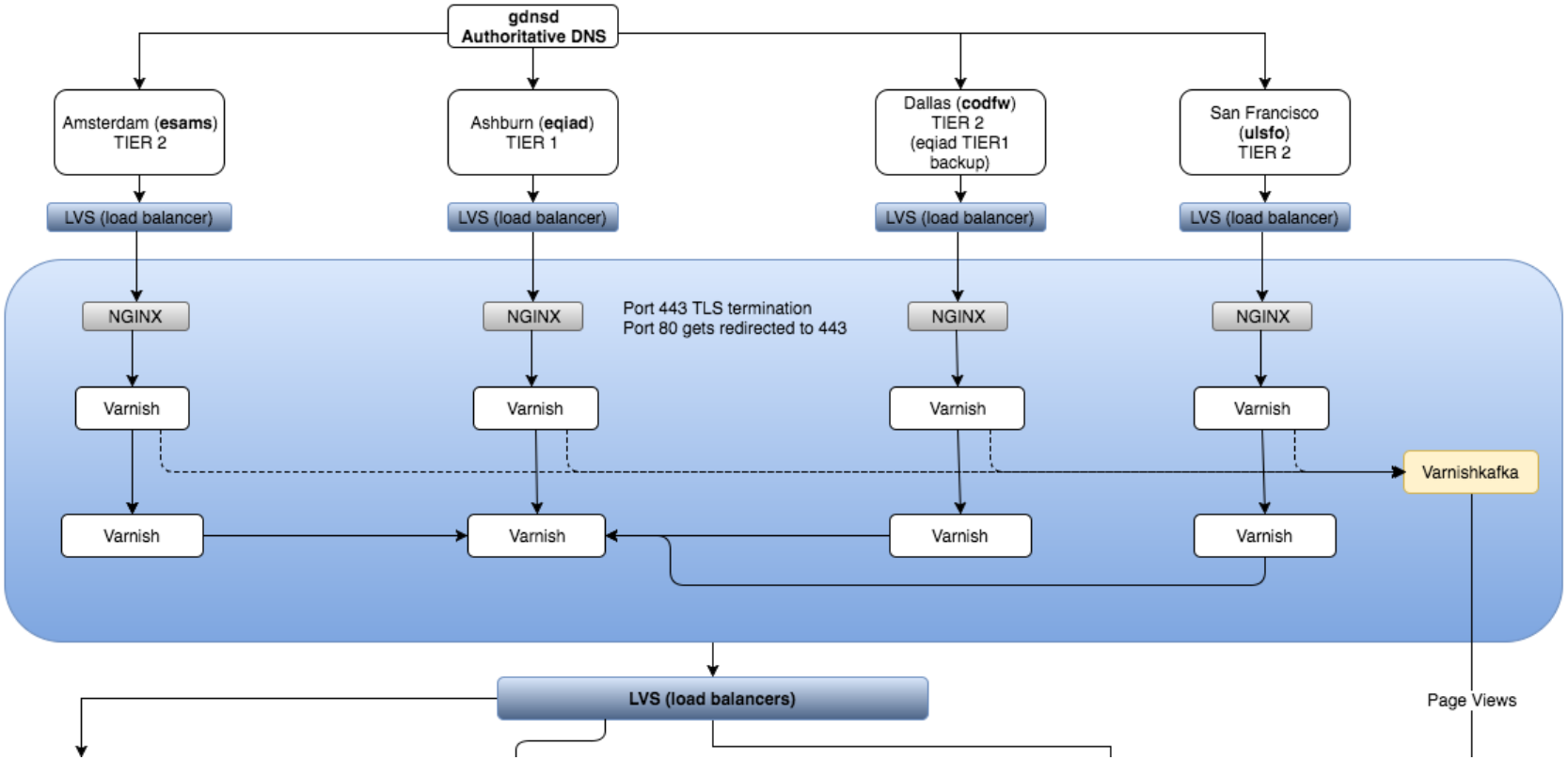


Todo ocurrió porque un sistema LAMP necesitaba rediseñarse para poder *crecer*



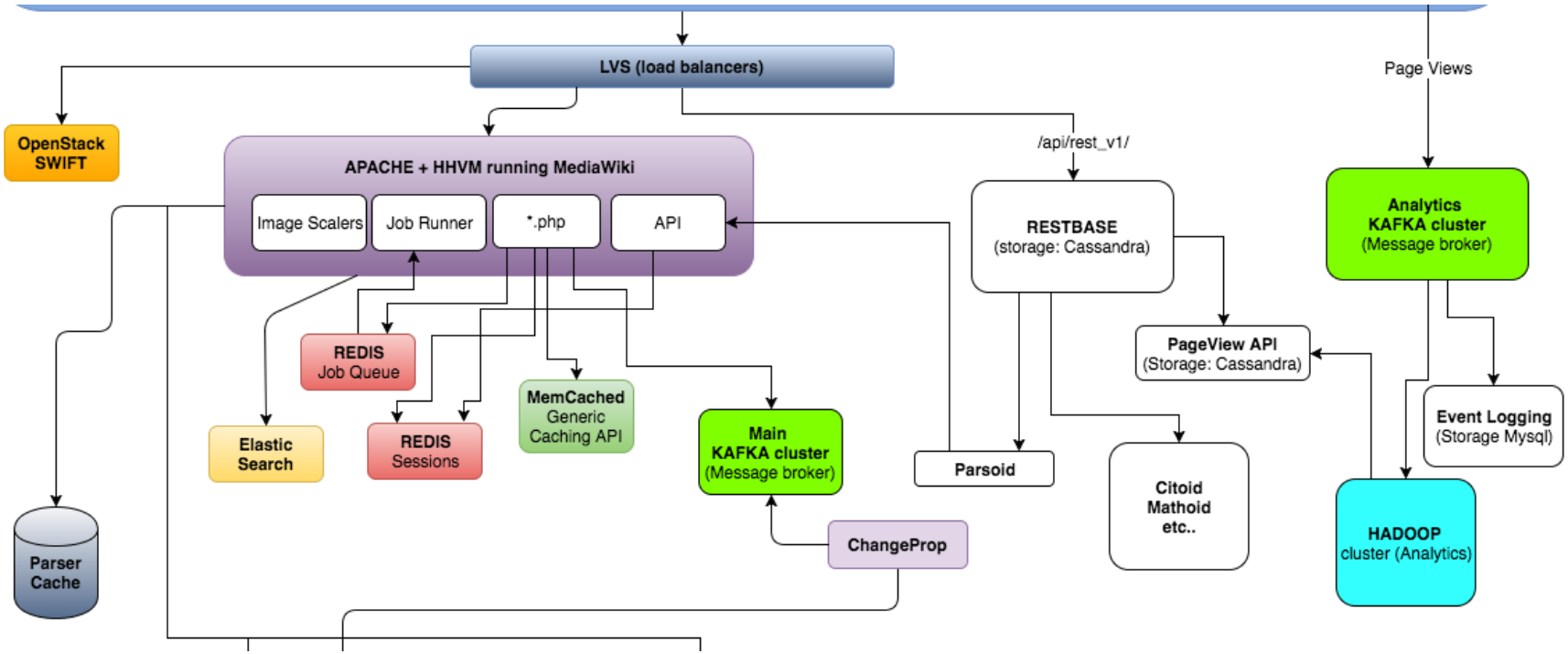
3. La Wikipedia "a piezas"

- Punto de entrada y reparto entre CPDs
  - DNS, repartidores y caches



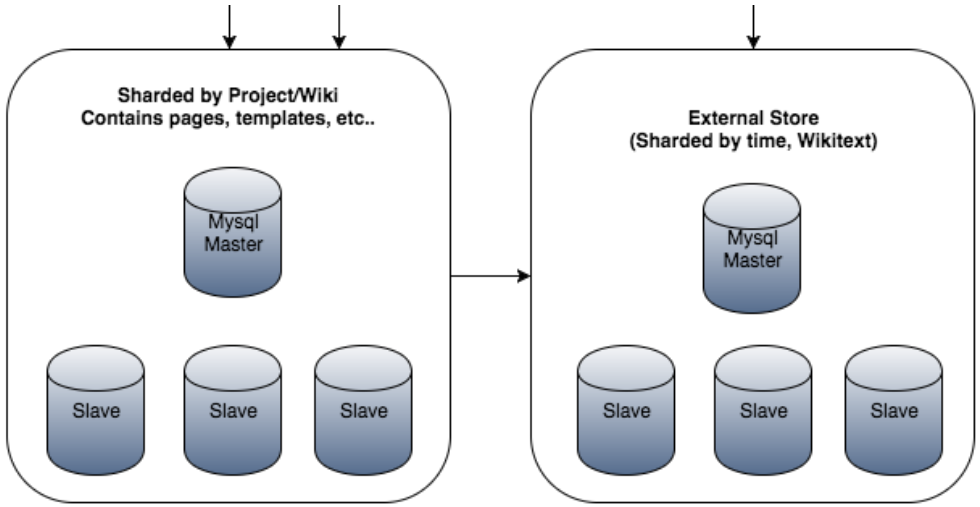
3. La Wikipedia "a piezas"

- Procesamiento
  - Repartidores, mensajería, API, eventos, caching interno, …



3. La Wikipedia "a piezas"

- Almacenamiento
  - Diferentes criterios de *sharding*



Índice

4. Aplicaciones distribuidas en la nube

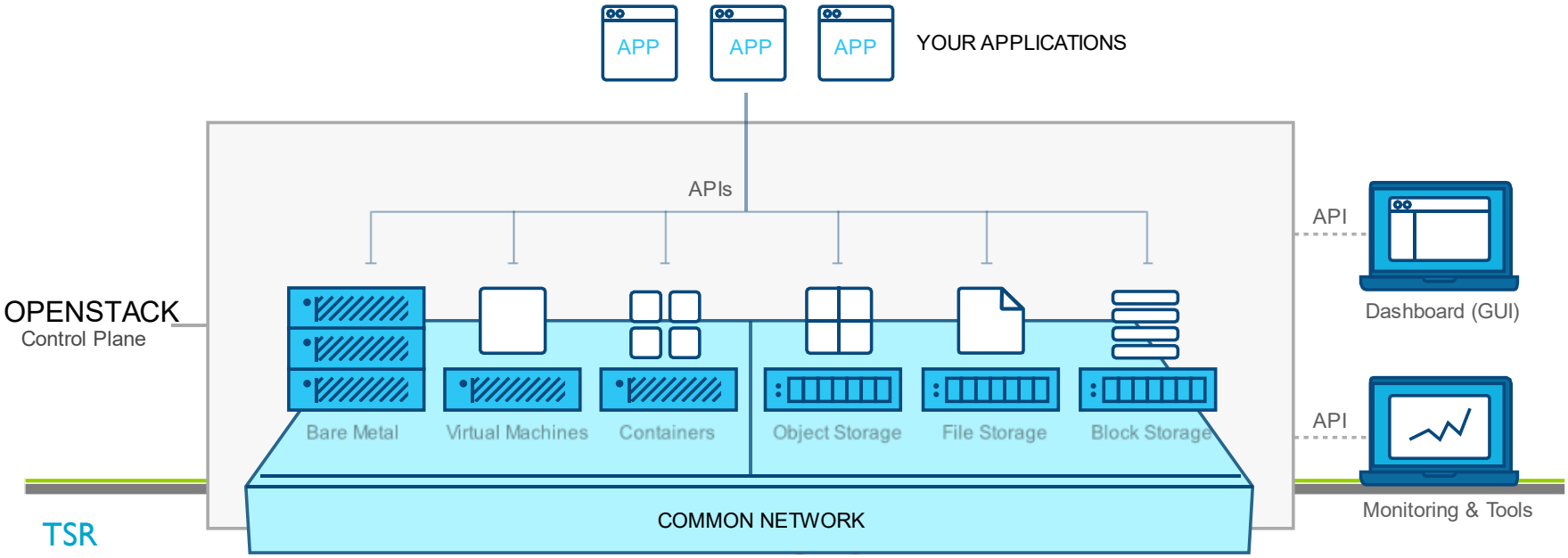
- Muchos de los requisitos de una aplicación a escala mundial ya se cumplen en la nube (CC)
  - Alcance mundial
  - Potencia computacional y de comunicación
  - Variedad de servicios que no necesitamos implementar
  - Propiedades garantizadas que podemos monitorizar
- El diseño, en la actualidad, de una aplicación con el alcance de la Wikipedia no se basaría en componentes discretos
  1. Determinar los servicios necesarios
  2. Concretar las propiedades a mantener
  3. Otros aspectos (coste, tecnología, …)
  4. … ¡y elegir proveedor de CC!

4.1 Intro Openstack

- Objetivo: plataforma de software libre para crear **infraestructuras de nube** privadas y públicas sobre equipamiento estándar
  - Promueve estándares libres
- Escalable y sin excesiva complejidad
  - Diseño modular basado en componentes
- Agnóstico en lo referente al hipervisor y al hardware
- Fundado por Rackspace y la NASA en 2010
  - El CERN es uno de sus más destacables usuarios

4.1 Intro Openstack

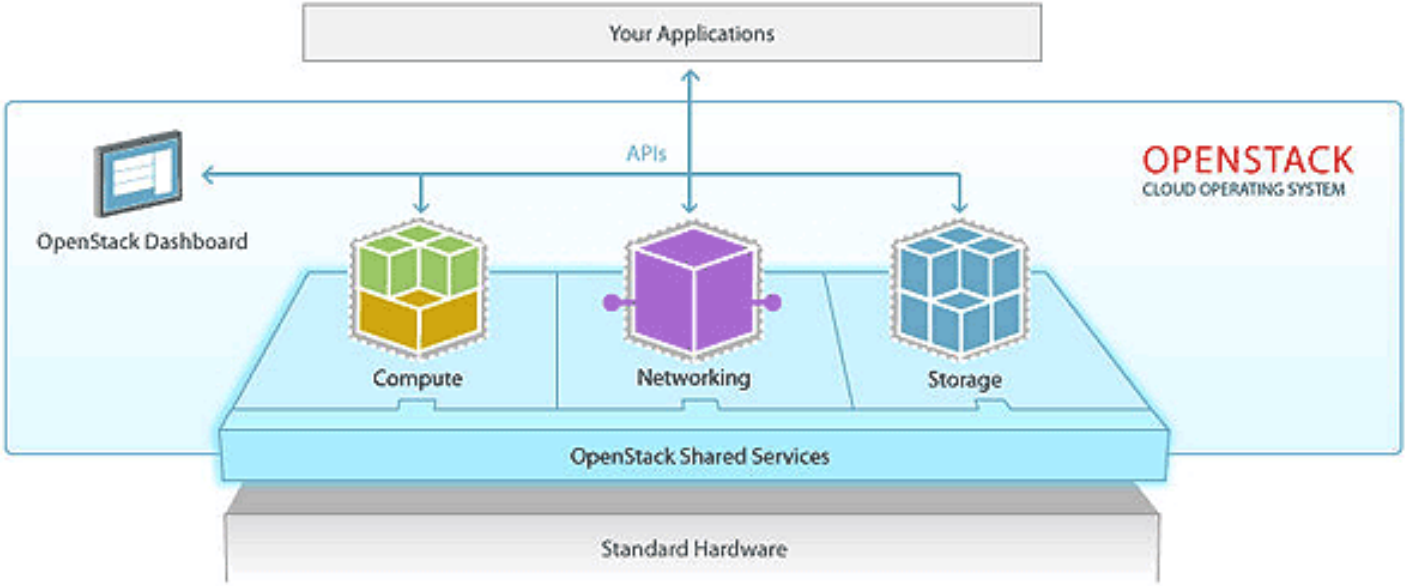
- En este nuevo escenario debemos determinar…
  - qué servicios se necesitan
  - cómo interactúa mi aplicación con ellos
  - qué garantías quiero mantener
  - los pasos necesarios para desplegar, monitorizar, etc…
  - qué proveedor puedo contratar



4.2 Servicios Openstack

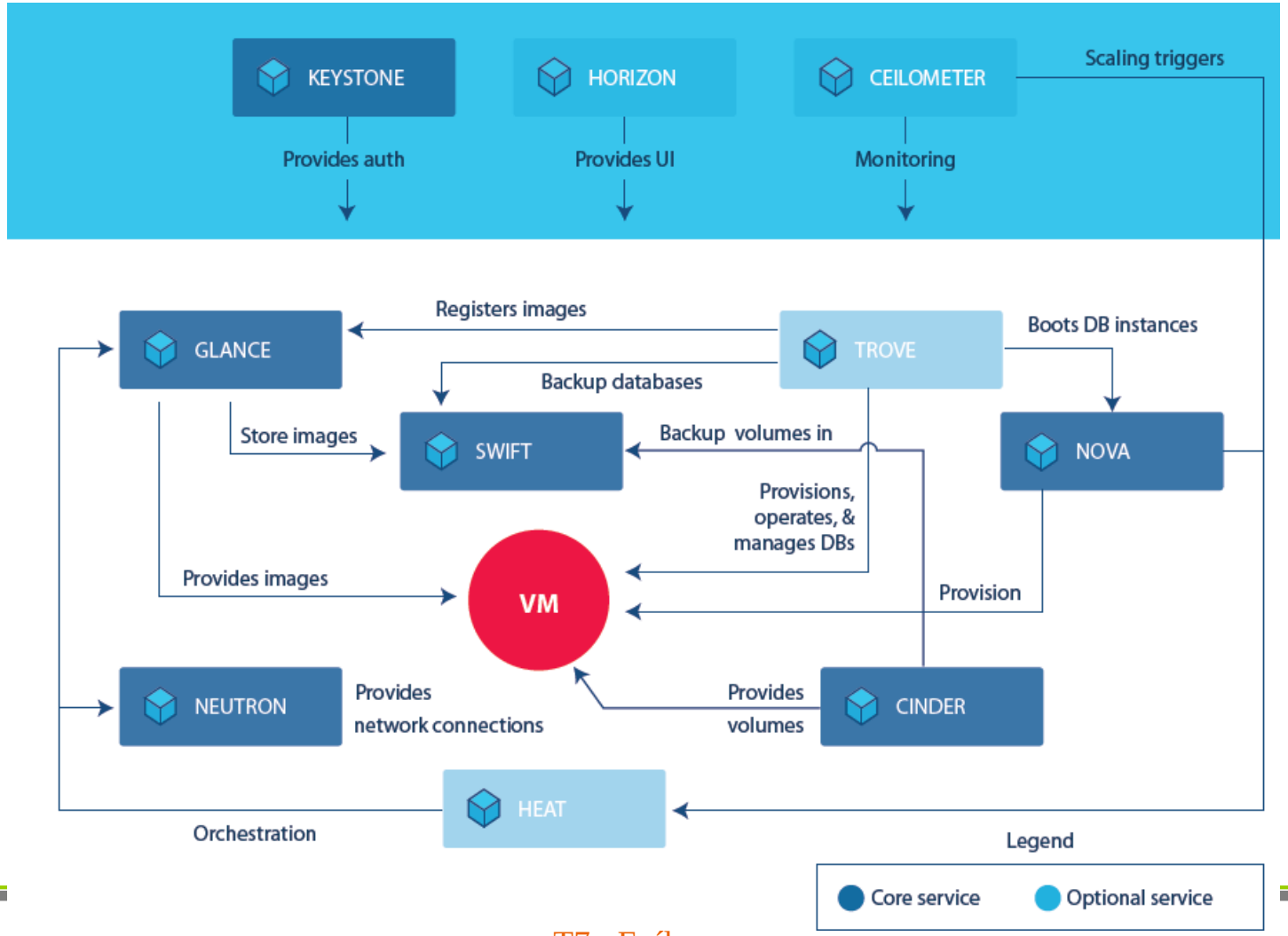
- Los servicios básicos se pueden resumir en
  - Computación
  - Comunicación (red)
  - Almacenamiento

¡Pero hay hasta 61 servicios! (*OpenStack Rocky, 30/08/2018*)



4.2 Servicios Openstack

- En este esquema se muestran los servicios más habituales



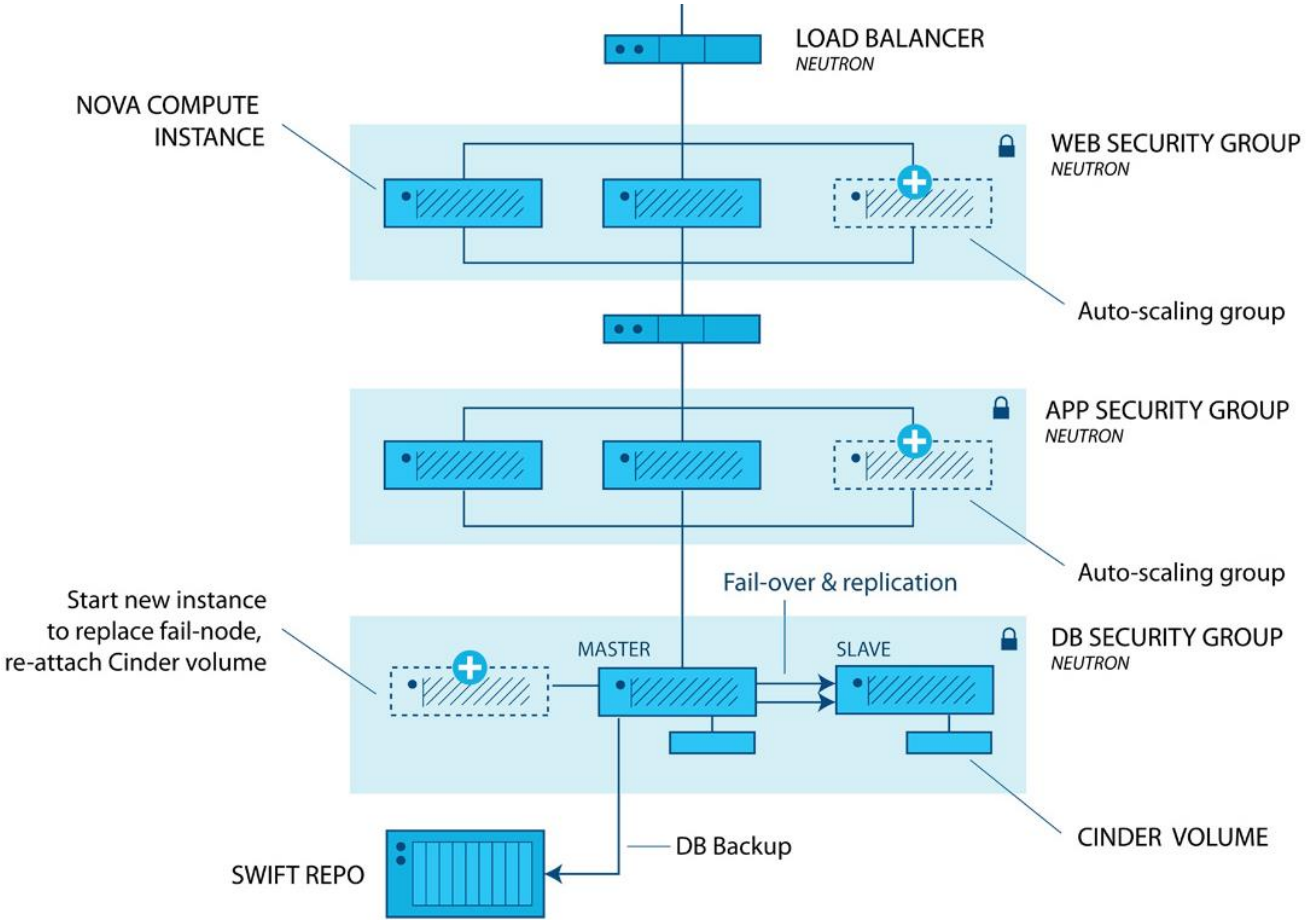
4.2 Servicios Openstack

- Openstack propone algunas combinaciones de servicios para tipos específicos de aplicación

	keystone	neutron	nova	swift	glance	cinder
	Identidad	Red	Computación	Almacenamiento de objetos	Servicio imágenes	Almacenamiento de bloques
Procesamiento y difusión de video	sí	sí	sí	sí		
Aplicaciones web	sí	sí	sí	sí	sí	sí
Big Data	sí	sí	sí		sí	sí
eCommerce	sí	sí	sí		sí	sí
Computación de alto rendimiento	sí		sí		sí	sí

4.2 Servicios Openstack para aplicaciones web

- En el caso de aplicaciones web hay un total de 10 componentes involucrados



4.3 La visión del programador

Las bibliotecas , como *libcloud*, transforman las invocaciones en mensajes REST (*Representational State Transfer*)

- REST se utiliza para la transferencia automática y controlada de información, mediante HTTP, empleando una URI base en el servidor
  - El cliente realiza una petición HTTP (GET, POST, PUT o DELETE)
  - El servidor contesta con un mensaje (formato XML o JSON)

- En la configuración de keystone se especifica un URL para cada API. P.ej.
  - identity (keystone) http://localhost:5000/v3
  - image (glance) http://localhost:9292/v1
  - compute (nova) http://localhost:8774/v2
  - volume (cinder) http://localhost:8776/v1

- Más información en http://api.openstack.org

Índice

1. Revisitando la Wikipedia
2. Bloques constructivos de una **aplicación web**
3. La Wikipedia "a piezas"
4. Aplicaciones distribuidas en la nube
5. Bibliografía

5. Bibliografía

- Wikipedia
  - https://meta.wikimedia.org/wiki/Wikimedia_servers
- Concurrent Programming for Scalable Web Architectures (Benjamin Erb)
  - http://berb.github.io/diploma-thesis/
- Openstack
  - https://www.openstack.org/software/project-navigator/openstack-components
  - https://www.openstack.org/assets/software/mitaka/OpenStack-WorkloadRefArchWebApps-v7.pdf
    - Modelo propuesto para aplicaciones web