Unit 7 - Epilogue

Network Information System Technologies

Contents

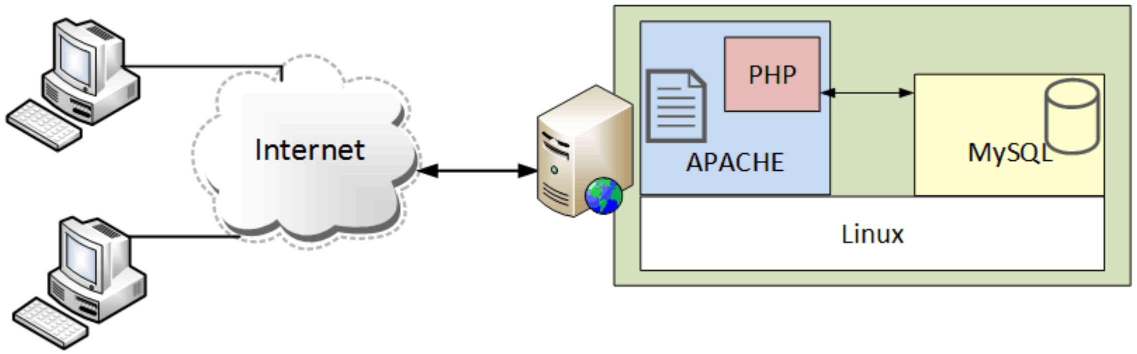
Goals

- Revise the contents developed in this subject
- Identify the pieces that may be used nowadays in order to build distributed applications, mainly in the scope of web applications
- Propose cloud platforms as the ideal support in the development and execution of distributed applications.

Contents

1. Revisiting Wikipedia

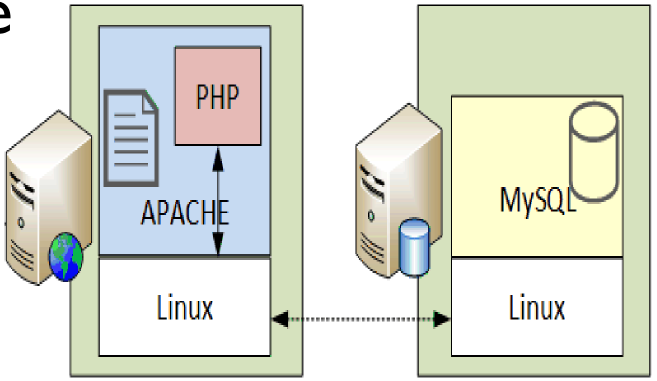
- Wikipedia is a LAMP system



  - Concretely, a web application
- It needs a bandwidth that cannot be achieved with a single connection: **multiple servers** are needed!
  - Multiple problems arise:
    - How to provide a single public endpoint
    - How to synchronise these servers
    - How to ensure availability in case of failures
    - How to increase throughput

1. Revisiting Wikipedia

- Its main internal services may be easily separated
  - PHP needs APACHE
  - The MySQL DBMS is an independent service



- This first modification is insufficient because it generates pieces that are too big: an application redesign[(*)] is needed, building components with…
  - High cohesion
  - Low coupling
- The system should be replicated in order to allow its geographical distribution

[(*)] The abilities needed to this end cannot be achieved in a single semester

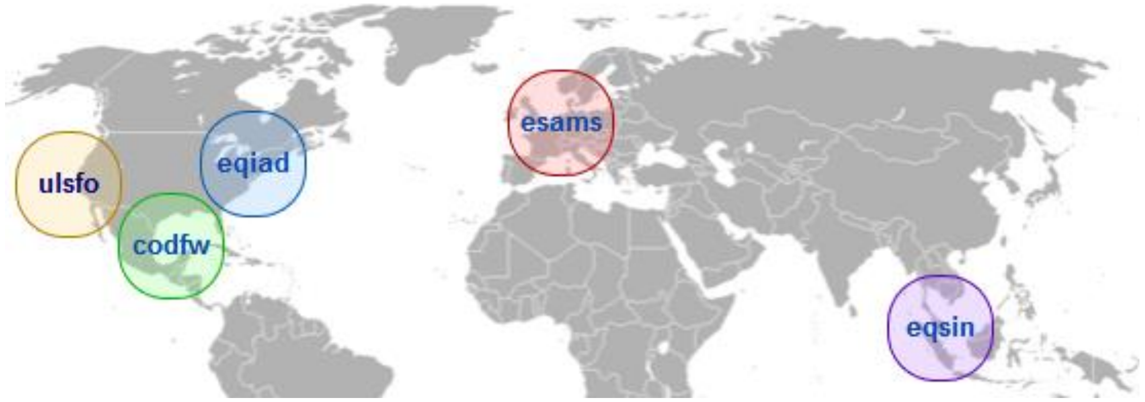
1.1 Geographical replication of Wikipedia

With replication, we achieve...

- *…an increase in throughput*
- *…an improvement in fault tolerance*

…and, complemented with geographical distribution...

- *...a latency decrease in client-server interactions*

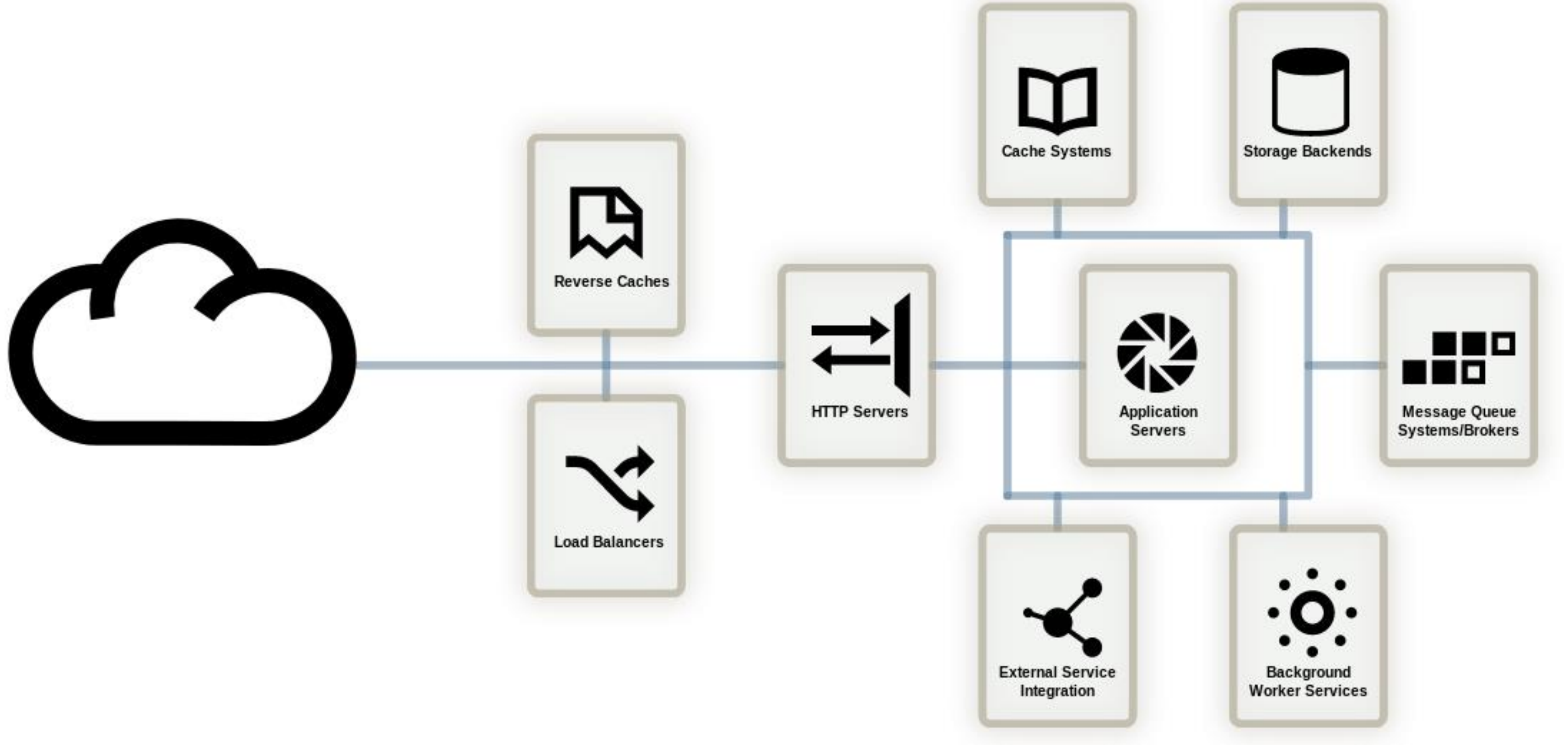


- Eqiad: (primary) application servers
- Codfw: (secondary) application servers
- Esams, Ulsfo and Eqsin are caching datacentres

Contents

2. Building blocks in a web application

Current web applications combine multiple components, each one with its own requirements on scalability.



Source: *Concurrent Programming for Scalable Web Architectures*

2.1 Web server

This element is specialised in HTTP request/response management including: connection handling, SSL/TLS cyphering, compression and static contents provisioning.

- Request processing is carried out by application servers.

Scalability strategy	The basic solution for web servers is cloning. If they do not hold any state, that is straightforward.
Real world examples	Apache HTTP Server is currently the most popular web server. There are alternatives that provide better scalability, such as nginx and lighttpd.
Cloud-based examples	Google App Engine (GAE) internally uses Jetty, a Java-based, high-performance web server.

2.2 Application servers

An application server processes the incoming requests, including parameter validation, database querying, communication with backend systems and template rendering.
With that, it builds the appropriate answer for each request.

Scalability strategy	Application servers do not share any platform resources directly in order to enhance their scalability: • If there is any shared state, scalability becomes very difficult and complex. • Solution: Coordination and communication between application servers should be outsourced to shared backend services.
Real world examples	Popular environments include dedicated scripting languages such as Ruby (on Rails), PHP or Python. In Java, application containers like RedHat's JBoss Application Server or Oracle's GlassFish are also very popular.
Cloud-based examples	GAE and Amazon's Elastic Beanstalk support Java Servlets. GAE can also host Python-based and Go-based applications.

2.3 Load balancing and reverse caches

In order to manage multiple HTTP servers, we need a **load balancer** that spreads the incoming requests among those server instances.

- It is called "reverse proxy" when it runs at the HTTP layer.

A **reverse cache** is a reverse proxy that keeps the dynamic content generated by the web application.

- It doesn't forward requests to the application server if the answer has already been generated in previous requests.

Scalability strategy	Load balancers may be cloned. Different strategies are required to balance their load again. • A popular approach to balance a web application is to provide multiple servers to a single hostname via DNS. Reverse caches provide an easy parallelizable service.
Real world examples	Load balancers: HAProxy, perlbal and nginx. Reverse proxies with caching: Varnish and nginx
Cloud-based examples	ELB (Amazon) is a load balancing and caching service.

2.4 Message queue system

For all those components that don't require any specific communication tool, a message-oriented middleware may provide their main integration mechanism.

Scalability strategy	A decentralised messaging system can provide better scalability. Message-oriented middleware systems with a message broker require partitioning of messaging participants and replication of message brokers.
Real world examples	AMQP is a messaging protocol with several mature implementations, such as RabbitMQ. A popular broker-free and decentralized messaging system is ØMQ.
Cloud-based examples	Amazon provides SQS, a message queueing solution. GAE provides a queue-based solution for the handling of background tasks and a dedicated XMPP messaging service. • Both services have high latencies for message delivery. It is not reasonable to use them as part of HTTP request handling. • Several EC2-based custom architectures have rolled out their own messaging infrastructure, based on products such as ØMQ.

2.5 Backend data storage

The backend data storage provides data persistency to multiple kinds of data (binary, structured, non-structured...).
There are several types of storage: relational DBMS, NoSQL data stores, distributed file systems...

Scalability strategy	Scaling data storages is a challenging task. Replication, vertical partitioning (denormalisation) and sharding (horizontal partitioning) are traditional approaches.
Real world examples	MySQL is a relational dBMS with clustering support. Riak, Cassandra and HBase are typical representatives of scalable NoSQL datastores. HDFS, GlusterFS and MogileFS are good scalability examples for distributed file systems.
Cloud-based examples	GAE provides both a data store and a blob store. Amazon provides several solutions for cloud-based data storage (e.g. RDS, DynamoDB, SimpleDB) and file storage (e.g. S3).

2.6 Cache system

These components provide volatile storage in order to reduce latency in the accesses to highly demanded elements.

- They usually are key/value memories that may be deployed (i.e. sharded) in multiple nodes.
- They may provide other advanced characteristics, as a PUB/SUB communication pattern.

Scalability strategy	Essentially, a distributed cache is a memory-based key/value store. • So, vertical scale can be achieved by provisioning more RAM to the machine. A higher scale is possible by cloning and replicating nodes and partitioning the key space.
Real world examples	Memcached is a popular distributed cache. Redis supports structured data types and publish/subscribe channels.
Cloud-based examples	GAE supports a Memcache API. Amazon provides a dedicated caching solution called ElastiCache.

2.7 Background worker service

Workers de Procesamiento

Heavy-weight tasks shouldn't be run by the application servers, specially when those servers use asynchronous programming.

Scalability strategy	Adding more resources and nodes to the background worker pool... • speeds up the computation • allows the execution of more concurrent tasks, thanks to parallelism. From a concurrency perspective, it is easier to scale worker pools when their jobs are small, isolated tasks with no dependencies.
Real world examples	Hadoop is an open-source implementation of the MapReduce platform, that allows the parallel execution of certain algorithms on large data sets. Twitter has released the Storm engine, a distributed real-time computation system targeting stream processing among others. Spark is an open source framework for data analytics designed for in-memory clusters.
Cloud-based examples	GAE provides a Task Queue API, that allows to submit tasks to a set of background workers. Amazon offers a MapReduce-based service, called Elastic MapReduce.

2.8 Integration of external services

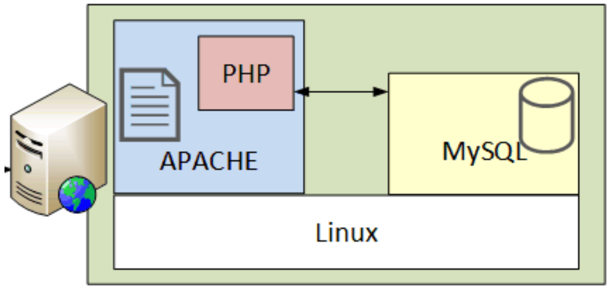
Integración de Servicios Externos

Enterprise environments often integrate additional backend systems, such as CRM/ERP systems or process engines. This is addressed by *enterprise service buses* (**ESB**) that replace the simpler messaging component for integration.
The backend enterprise architecture may be decoupled from the web architecture. In that case, **web services** are used for that integration instead.

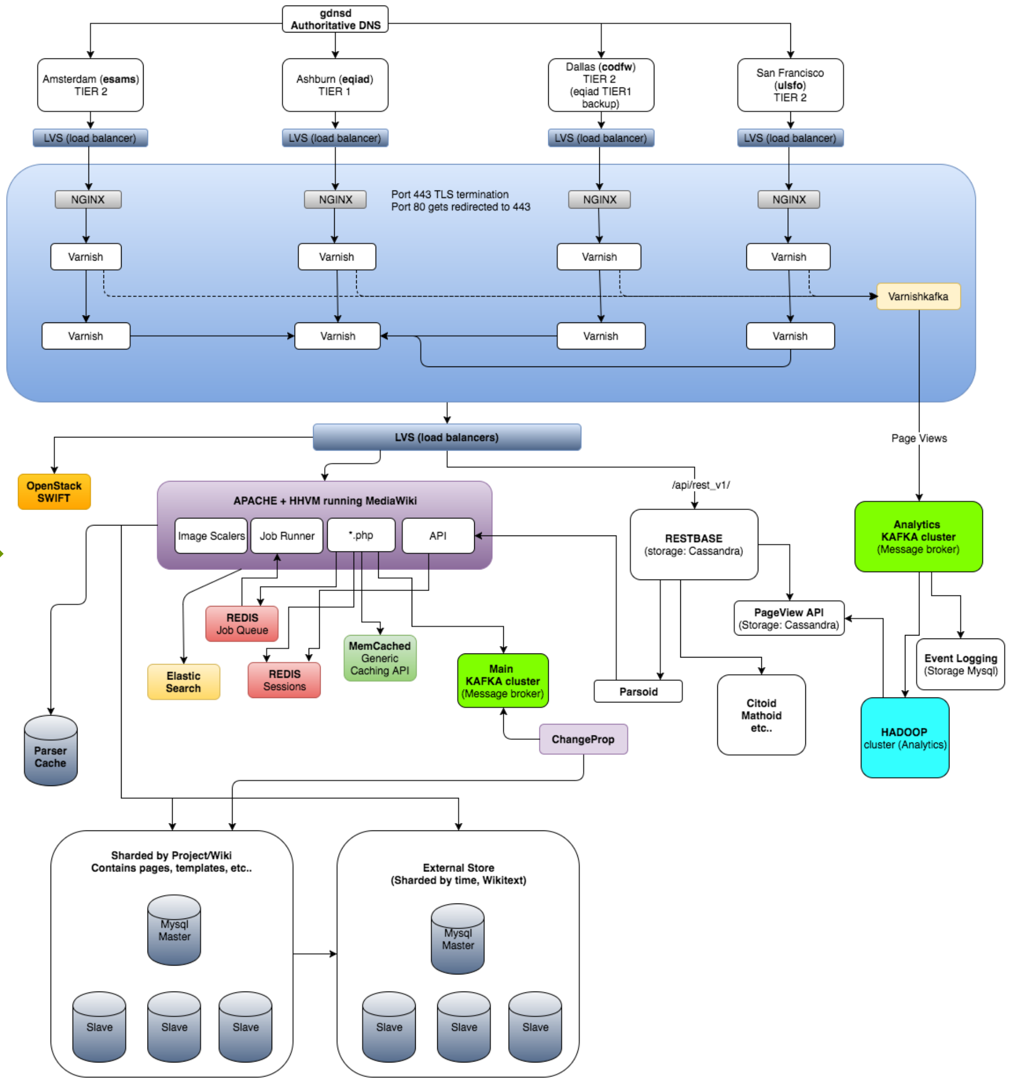
Scalability strategy	Scalability of external services depends mainly on their own design and implementation. Concerning the integration into a web architecture, it is helpful to focus on stateless and scalable communication patterns and loose coupling.
Real world examples	Mule and Apache ServiceMix are two open-source products providing integration features.
Cloud-based examples	GAE allows to access external web-based resources via URLFetch API. The GAE XMPP API may be used for message-based communication. The messaging services from Amazon may be used for integration.

Contents

3. Dissecting Wikipedia

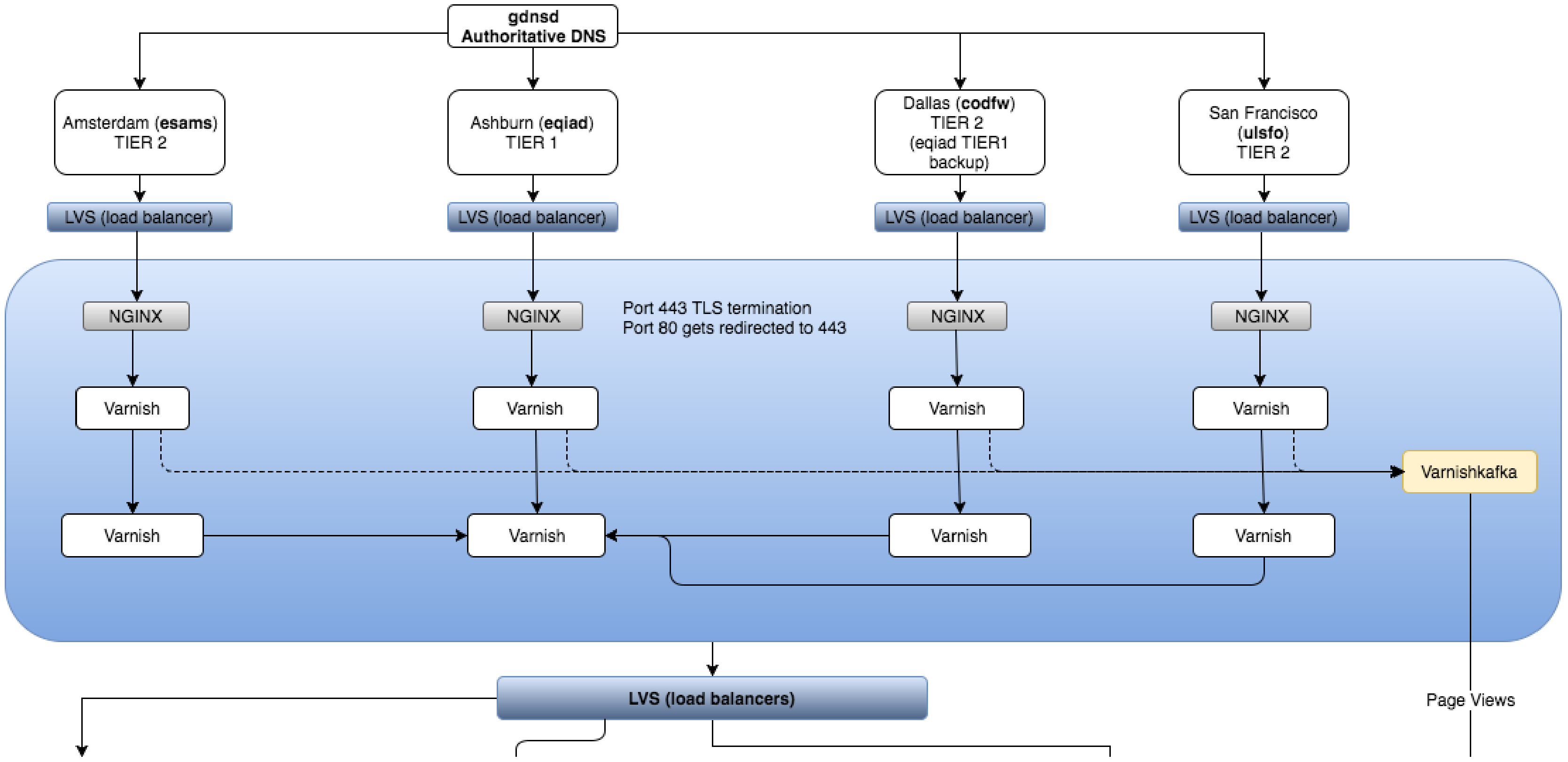


This evolution was started because LAMP systems need to be redesigned in order to be scalable.


gdnsd Authoritative DNS
Amsterdam (esams) TIER 2
Ashburn (eqiad) TIER 1
Dallas (codfw) TIER 2 (eqiad TIER1 backup)
San Francisco (ulsfo) TIER 2
LVS (load balancer)
NGINX
Port 443 TLS termination
Port 80 gets redirected to 443
Varnish
Varnishkafka
LVS (load balancers)
Page Views
OpenStack SWIFT
APACHE + HHVM running MediaWiki
Image Scalers
Job Runner
*.php
API
/api/rest_v1/
RESTBASE (storage: Cassandra)
Analytics KAFKA cluster (Message broker)
PageView API (Storage: Cassandra)
Event Logging (Storage Mysql)
REDIS Job Queue
Elastic Search
REDIS Sessions
MemCached Generic Caching API
Main KAFKA cluster (Message broker)
Parsoid
Citoid Mathoid etc..
HADOOP cluster (Analytics)
ChangeProp
Parser Cache
Sharded by Project/Wiki Contains pages, templates, etc..
Mysql Master
Slave
External Store (Sharded by time, Wikitext)

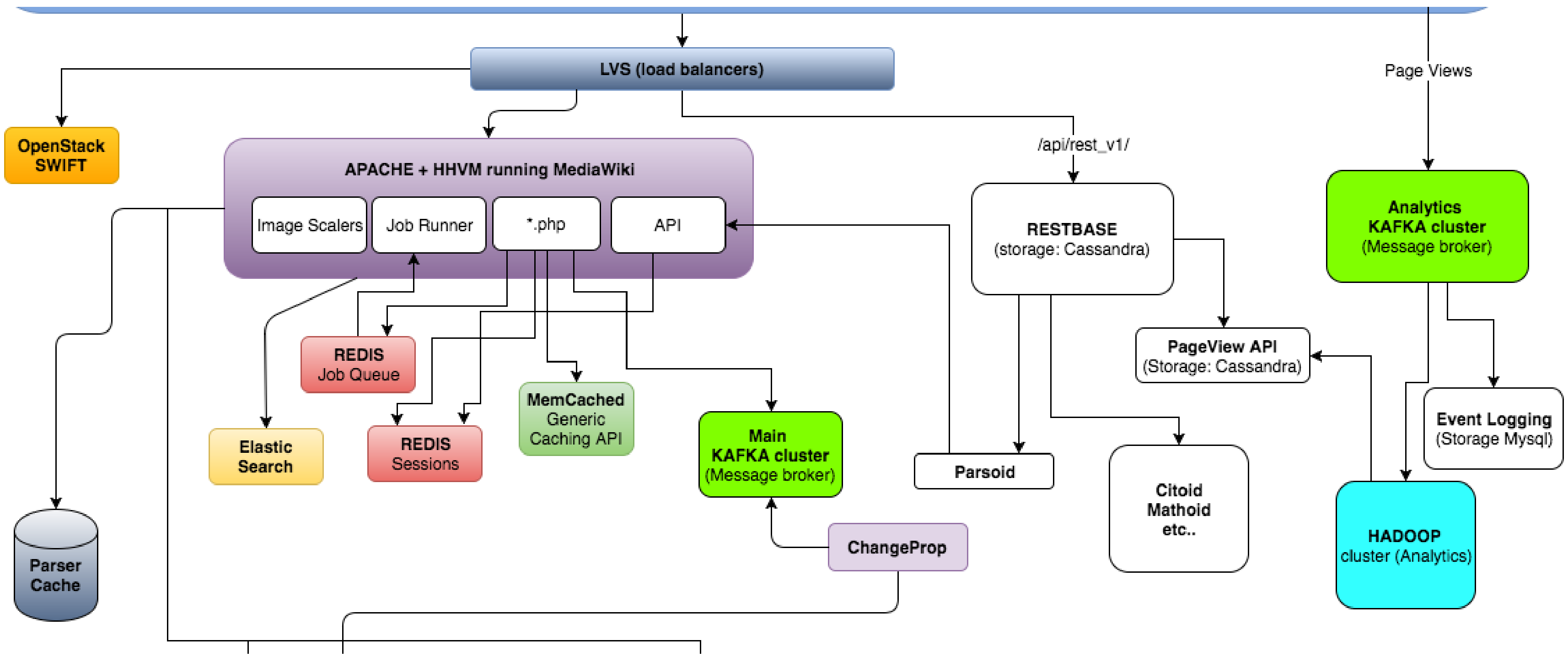
3. Dissecting Wikipedia

- Public endpoint and load balancing among datacentres.
  - DNS, load balancers and caches



3. Dissecting Wikipedia

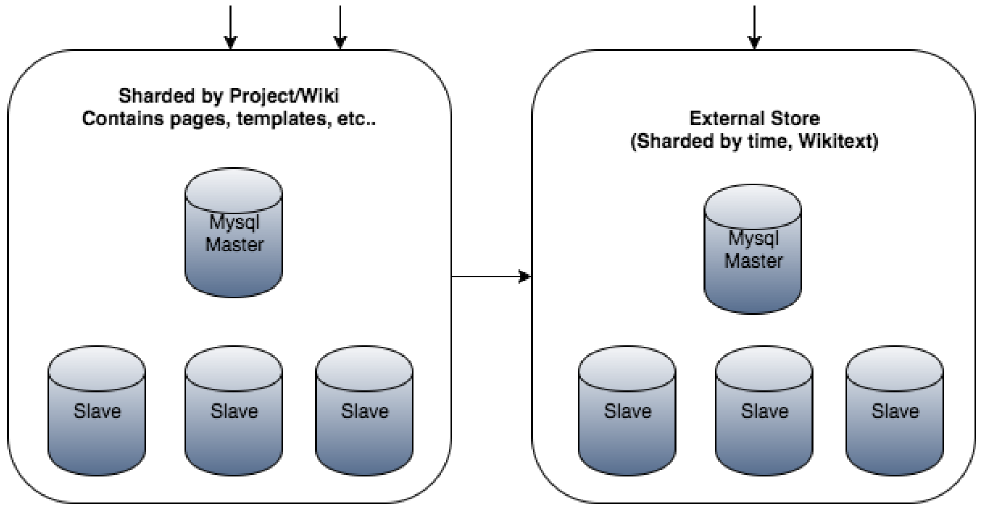
- Processing
  - Load balancers, messaging, API, events, internal caching, …



LVS (load balancers)
Page Views
OpenStack SWIFT
APACHE + HHVM running MediaWiki
Image Scalers
Job Runner
*.php
API
/api/rest_v1/
RESTBASE (storage: Cassandra)
Analytics KAFKA cluster (Message broker)
PageView API (Storage: Cassandra)
Event Logging (Storage Mysql)
REDIS Job Queue
MemCached Generic Caching API
Main KAFKA cluster (Message broker)
Elastic Search
REDIS Sessions
Parsoid
Citoid Mathoid etc..
HADOOP cluster (Analytics)
ChangeProp
Parser Cache

3. Dissecting Wikipedia

- Storage
  - Different sharding criteria



Contents

4. Distributed applications in the cloud

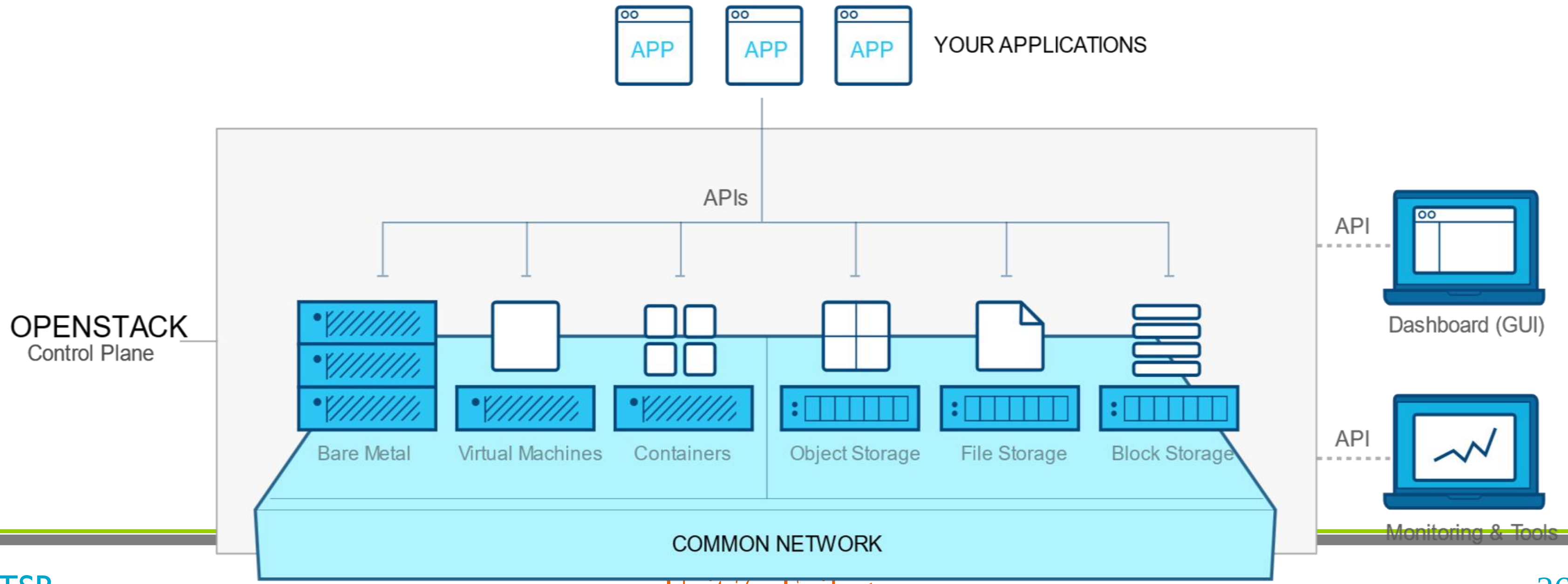
- Many of the requirements for building a highly scalable distributed service can be met in the cloud (CC)
  - Worldwide scope
  - Large computational and communicational capabilities
  - Provision of some base services
    - Thus, we do not need to implement them again
  - Guaranteed properties that we may monitor
- Currently, the design of a Wikipedia-like service need not rely on discrete components
  1. Determine the needed services
  2. Choose the properties to be ensured
  3. Other aspects (costs, technology...)
  4. …and choose a good cloud computing provider!

4.1 Intro to OpenStack

- Goal: Open source software platform to provide private (or public) **infrastructure as a service** systems
  - It promotes and uses open standards
- Scalable and easy to use
  - Modular design based on components
- Without hypervisor or hardware dependences
- Created by Rackspace and NASA in 2010
  - CERN is one of its users

4.1 Intro to OpenStack

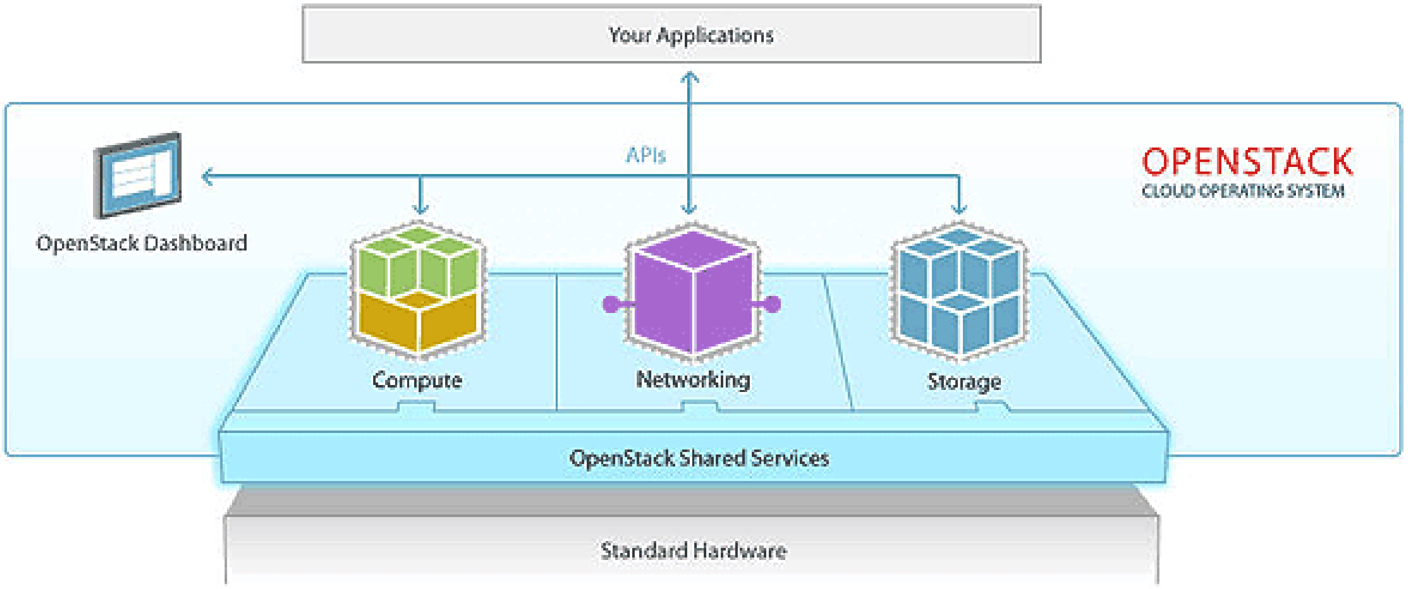
- In this new scenario, we should determine…
  - which services are needed
  - how our application interacts with them
  - which guarantees must be ensured
  - the steps needed in several stages: initial deployment, monitoring,…
  - what provider should be used



4.2 OpenStack services

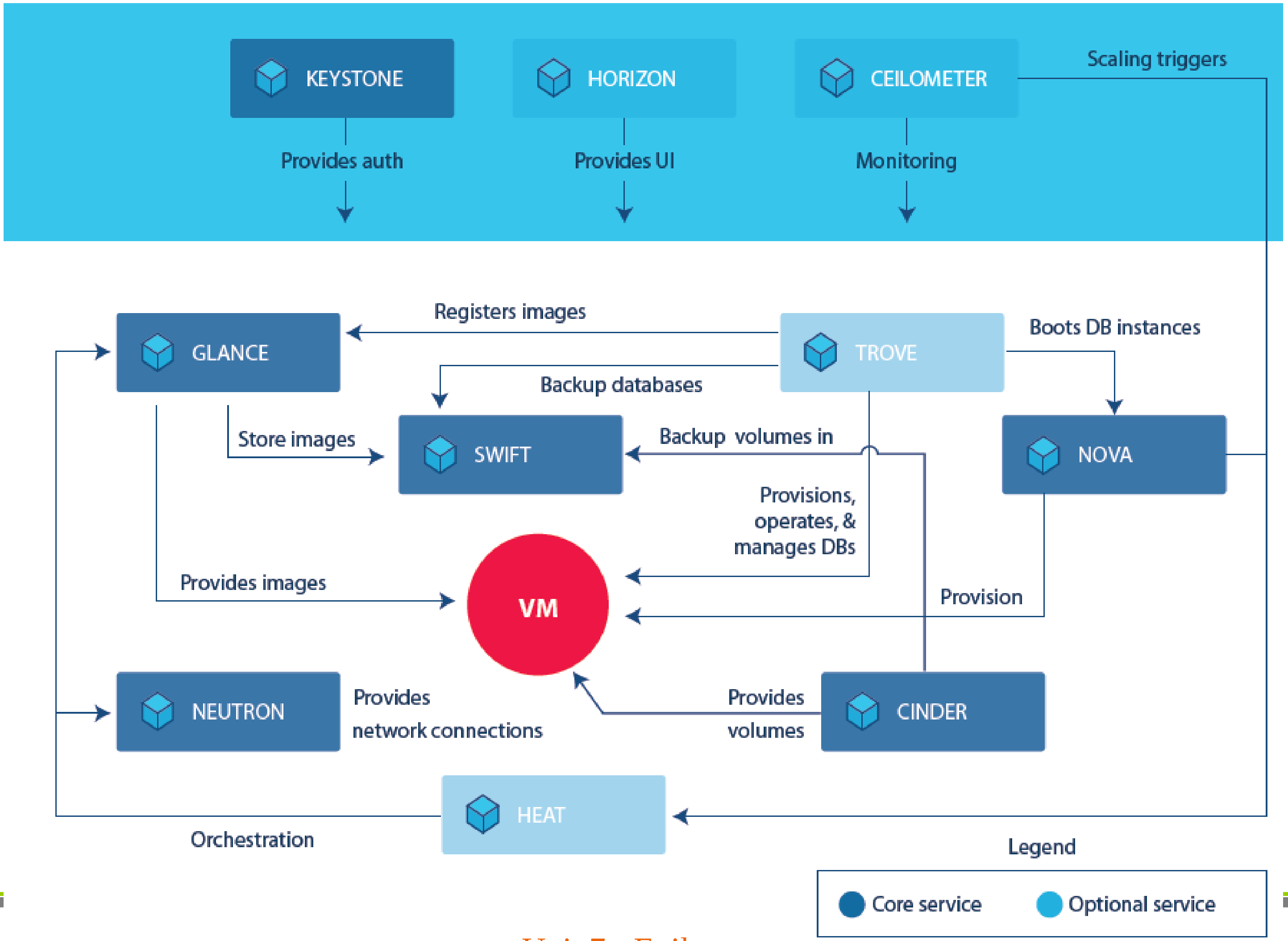
- Basic services are:
  - Compute
  - Networking
  - Storage

But there are 61 services! (*OpenStack Rocky, 30/08/2018*)



4.2 OpenStack services

- This schema shows the most popular services



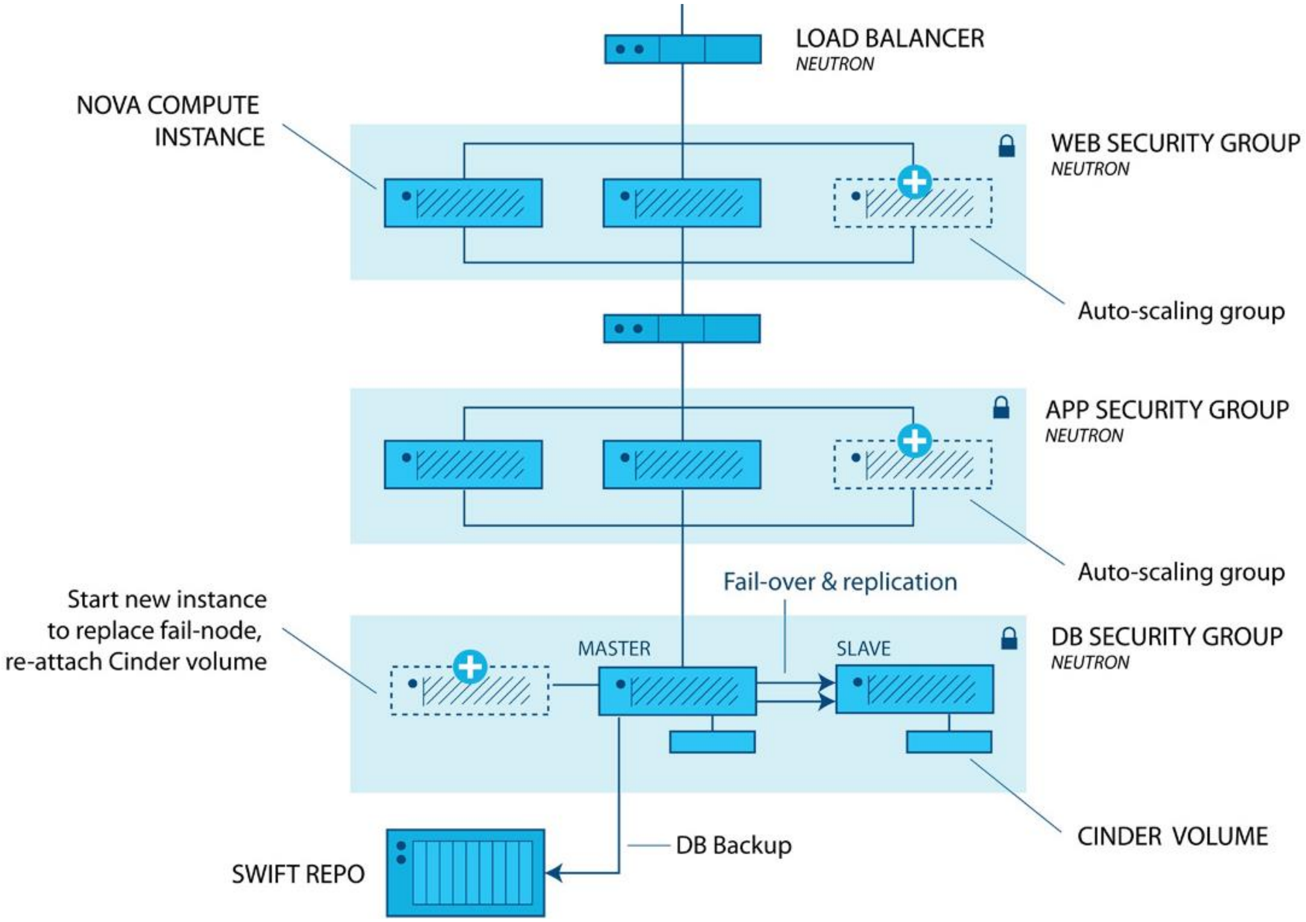
4.2 OpenStack services

- OpenStack proposes several combinations of services for specific application types

	keystone	neutron	nova	swift	glance	cinder
	Identity	Network	Compute	Object storage	Image service	Block storage
Video processing and broadcasting	yes	yes	yes	yes		
Web applications	yes	yes	yes	yes	yes	yes
Big Data	yes	yes	yes		yes	yes
eCommerce	yes	yes	yes		yes	yes
High performance computing	yes		yes		yes	yes

4.2 OpenStack services for web applications

- For web applications, there are 10 involved components



4.3 Programmer’s view

Libraries, like *libcloud*, transform the invocations into REST (*Representational State Transfer*) messages

- REST is used in order to automatically transfer the information, via HTTP, using a base URI in the server
  - Client sends an HTTP action (GET, POST, PUT o DELETE) request
  - Server replies with a (XML- or JSON-formatted) message

- The **keystone** configuration specifies a URL per API. E.g.,
  - identity (keystone) http://localhost:5000/v3
  - image (glance) http://localhost:9292/v1
  - compute (nova) http://localhost:8774/v2
  - volume (cinder) http://localhost:8776/v1

- Additional information in http://api.openstack.org

Contents

5. References

- 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
    - Proposed model for web applications.