Cloud Applications Architecture

Course 4 - Scalability

What is a Scalable System?

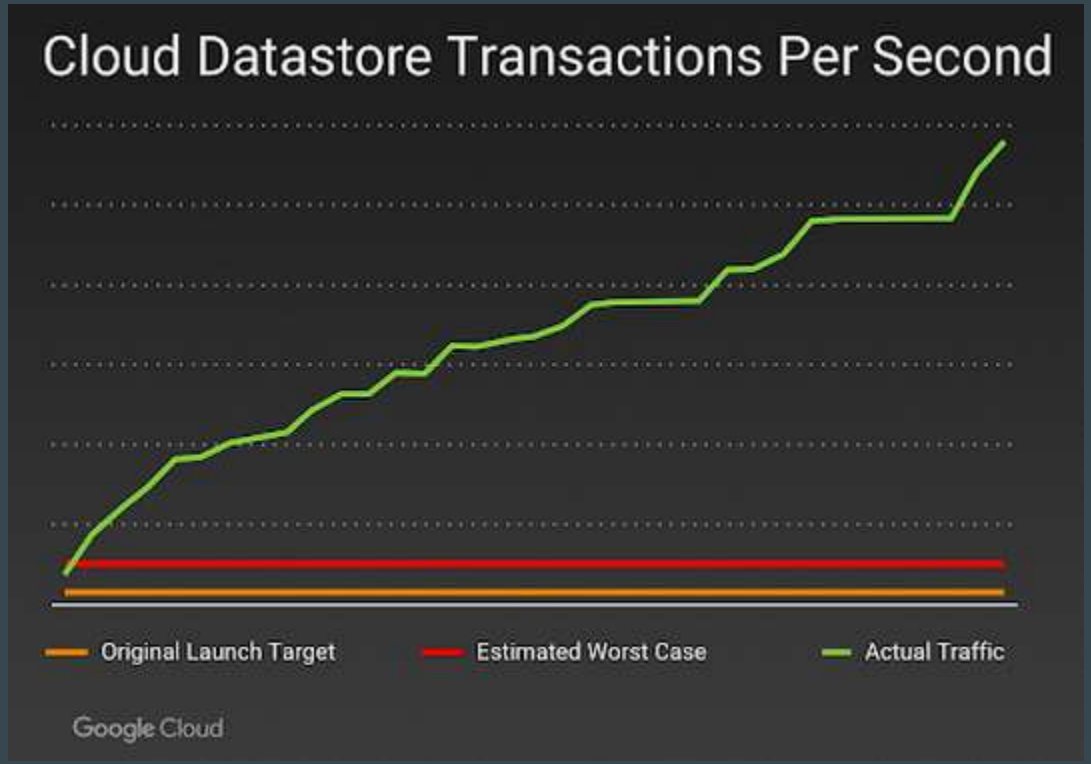
Match the Required Load

How do we handle traffic spikes? (e.g. Black Friday)

How do we handle gradual growth of the product?

How can we minimized cost if our system is used only during certain timeframes?

Load Testing

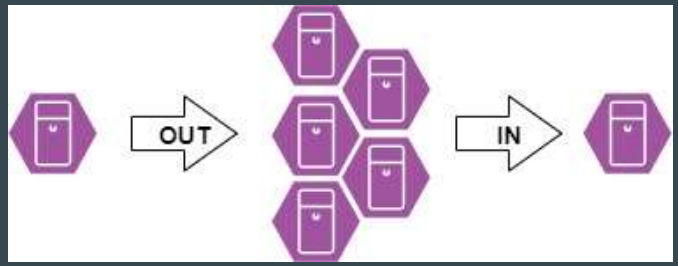


Pokemon Go launch day
source

Types of Scalability

Vertical
(Up/Down)

Horizontal
(In/Out)



Comparison

Vertical

- Usually easier/straightforward to implement (update the VM).
- Compatible with any app architecture (e.g. stateful apps).
- Doesn’t require additional components.
- Usually requires restarting the service.
- Might be the only approach for certain components (e.g. RDBMS)

Horizontal

- Infinite scale.
- Requires stateless apps (promotes better architectures).
- Requires load balancers (not always).
- Does not touch already existing VMs (i.e. no downtime) (unless when scaling in).

Context

We will focus on IaaS mostly (i.e. working with VMs).

While vertical scaling can definitely help, we will focus on horizontal scaling.

- Possibly unlimited scale
- Doesn’t require restart
- More interesting to talk about

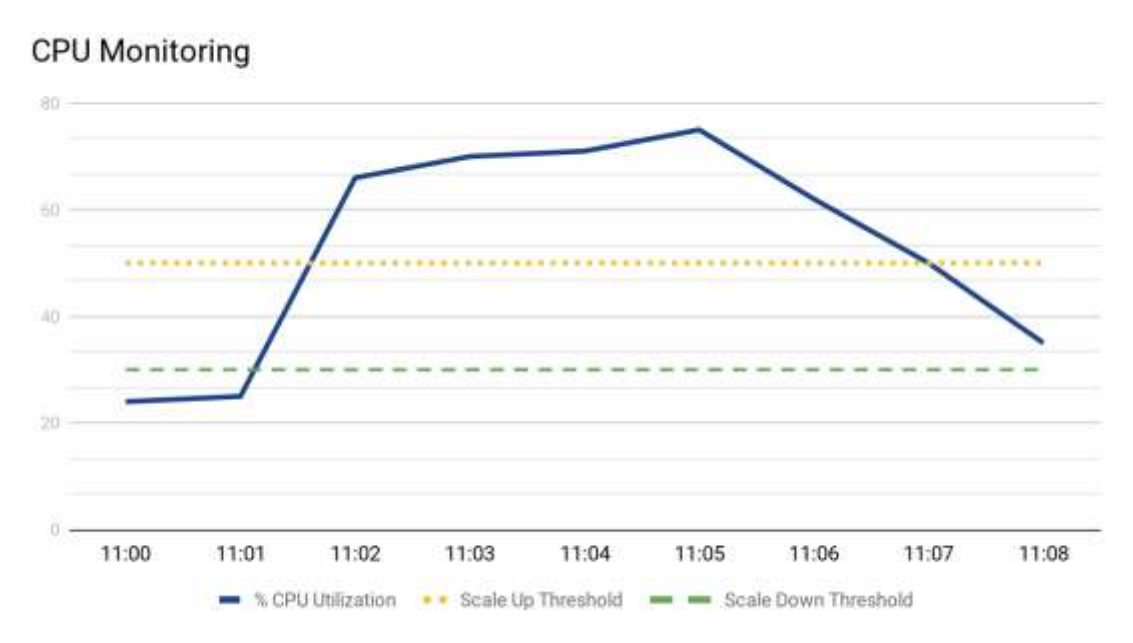
Scaling Strategies - Dynamic

Scale based on various **metrics** provided by the monitoring software:

- CPU, # of requests, memory, etc.

Choose the right metric for your system.

E.g. for a chat app, # of connections might be more relevant than CPU utilization.



Scaling Strategies - Dynamic

Aim to keep the load under 50% to have a buffer until additional VMs are ready

- Monitoring delay
- Scaling takes place only after sustained load.
  - E.g. scale up only if the load was above the threshold for 2 minutes
- Resource provisioning/initialization/setup
- The load balancer might take some time acknowledge the new VM
  - E.g. waits for the health check to pass at least 3 times

Scaling Strategies - Scheduled

Useful for predictable load patterns such as:

- Low usage at night
- High usage from Wednesday until Friday
- Events

Should be used when possible since it avoids scaling delays

- A.k.a. Resources are pre-warmed
- Providers can also pre-warm certain managed resources
    - E.g. managed load balancers

Traffic Routing

How do we route traffic when working with horizontal scaling?

Multiple approaches:

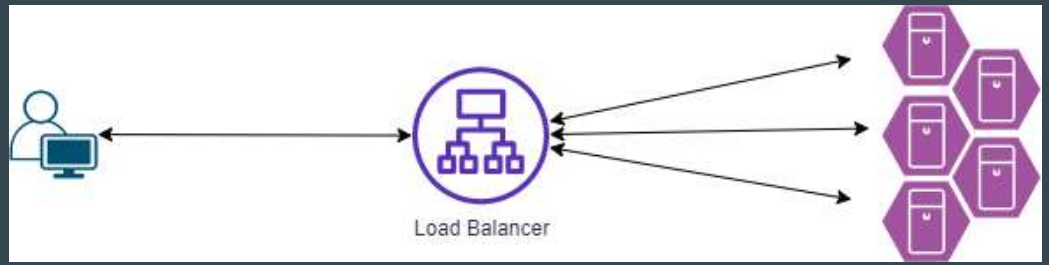
- Have a load balancer in front of the VMs
  - I.e. a single point of entry
- DNS load balancing
  - I.e. rely on the DNS service to redirect traffic to VMs
- Make the app aware of scaling
  - Rare, usually used for scaling databases

Load Balancers

Load Balancers

Serve multiple purposes:

1. Balance traffic across multiple targets
2. Monitor the health of target resources
3. Serve as a single point of entry
4. (Some) Caching
5. Others such as SSL offloading, authentication, A/B testing, etc.



(Some) Examples

NGINX

HAProxy

Traefik

AWS Elastic Load Balancer (Application LB/Network LB)

Google Load Balancing

Azure’s several (and confusing) services

Types of Load Balancers

HTTP (AKA Layer 7 LB)

- Works at layer 7 of the OSI model
- Offers most features
- Adds significant latency (~400ms)
- Behaves as a **Proxy (Sends new requests)**
- Good default choice

TCP/UDP (AKA Layer 4 LB)

- Works at layer 4 of the OSI model - i.e. works with packets
- Behaves similar to a network router (the request is not decomposed)
- Ultra-high performance
- Not so many features

DNS

- Also works at layer 7
- Useful for global load balancing
- A domain might be pointing to several IP addresses. Clients usually connect to the first one - send IPs in desired order.
- **Issues with caching/TTL**

Benefits & Common Features

HTTP, HTTPS, WebSockets / UDP, TCP

SSL/TLS Termination

Monitoring

Single point of access

Enables auto-scaling

Benefits & Common Features

Routing based on path, query strings, subdomains

Health checks

Port mapping

Integration with other services (e.g. authentication)

Caching

Cross-region

(Some) Routing Strategies

Round-Robin

- Simplest, route traffic in a loop. Good enough for simple use cases

Weighted Round-Robin

- Prefer certain targets more than others (useful when target vary in sizes)

Least Connections

- Useful when requests vary in load

Application Layer Scalability

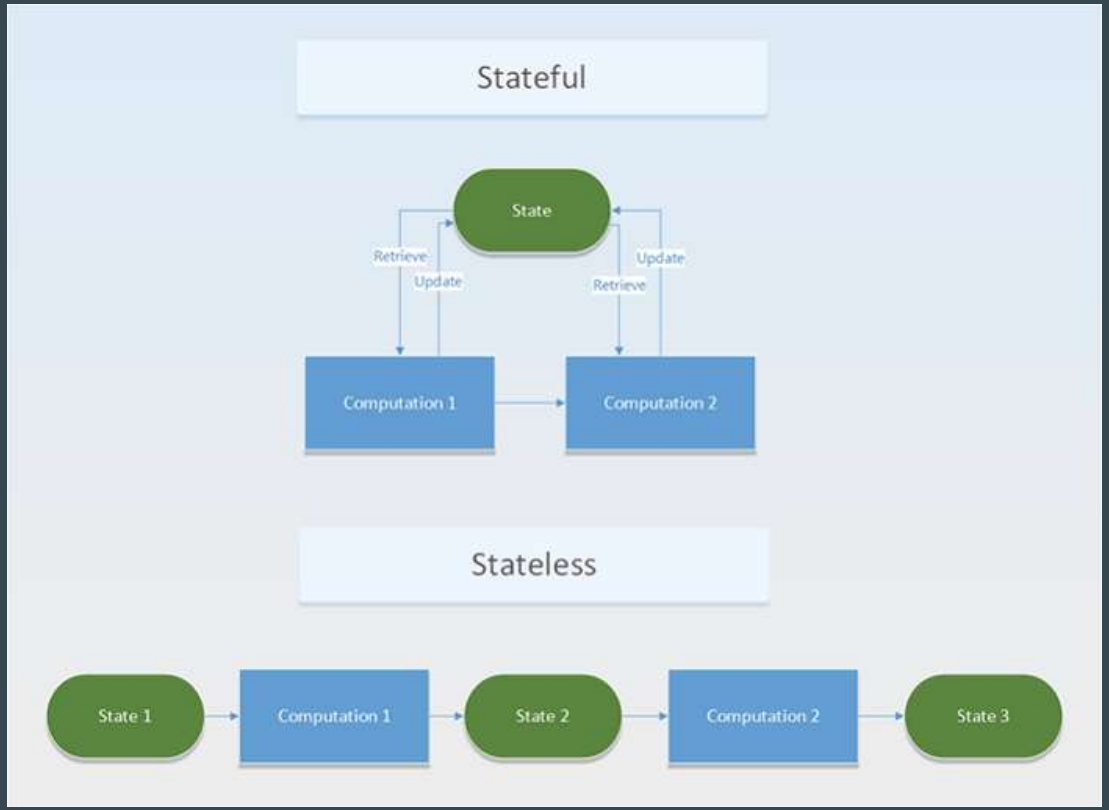
Stateless Applications

Applications were traditionally stateful (e.g. authentication sessions/shopping carts handled by the application layer).

Stateless applications handle requests independently.

- They don’t rely on previous requests
- The entire context is deduced based on the current request).

Stateless Applications



source

Pets vs Cattle

Rather cruel, but good analogy.

Pets are hand-built servers that we cannot just get rid of.

Cattle, on the other hand, can easily be replaced.

Automation is the main point.

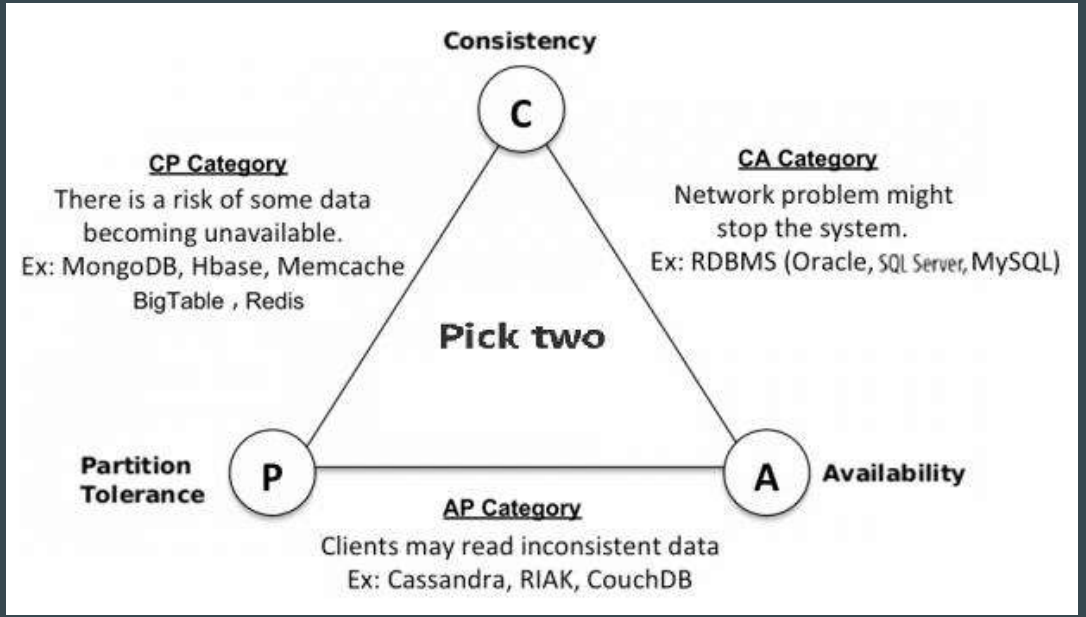
Database Layer Scalability

Context

NoSQL scale much easier than SQL databases (their main selling point).

We will cover different DB types in a future lecture.

The CAP Theorem



Consistency Types

Strong consistency

- When data is changed, every client sees the new data immediately.

Eventual consistency

- Some clients might see the new data with some delay
- ACID vs BASE

Certain DBs can be tuned to favor different consistency types.

Check out the Cosmos DB consistency levels

Scaling for Reads

Nodes dedicated just for reading.

Still a single node for writing.

Data replication is usually asynchronous (i.e. leading to eventual consistency).

Clients must be aware of it - i.e. read from nodes dedicated for reading.

Partitioning

Split the data into multiple subsets/groups/**partitions.**

Each item in a given partition has the same partition key.

Doesn’t imply multiple databases/nodes.

Still **improves performance as queries are usually executed within one partition**.

Queries should contain the partition key otherwise the DB engine has to look through all partitions.

Sharding

Similar to partitioning, but actually involves distributing the data across multiple nodes.

Bonus - Types of Traffic

North-South

- Refers to traffic from outside - i.e. from clients/external systems
- Requires more attention
  - Firewalls
  - NACLs
  - Authentication
  - Authorization
  - Rate Limiting
  - Schema verification

East-West

- Refers to traffic within the system's network - i.e. across servers/services
- Usually more relaxed checks

Summary