Implementing SRE Best Practices

Karun Subramanian
IT Operations Expert

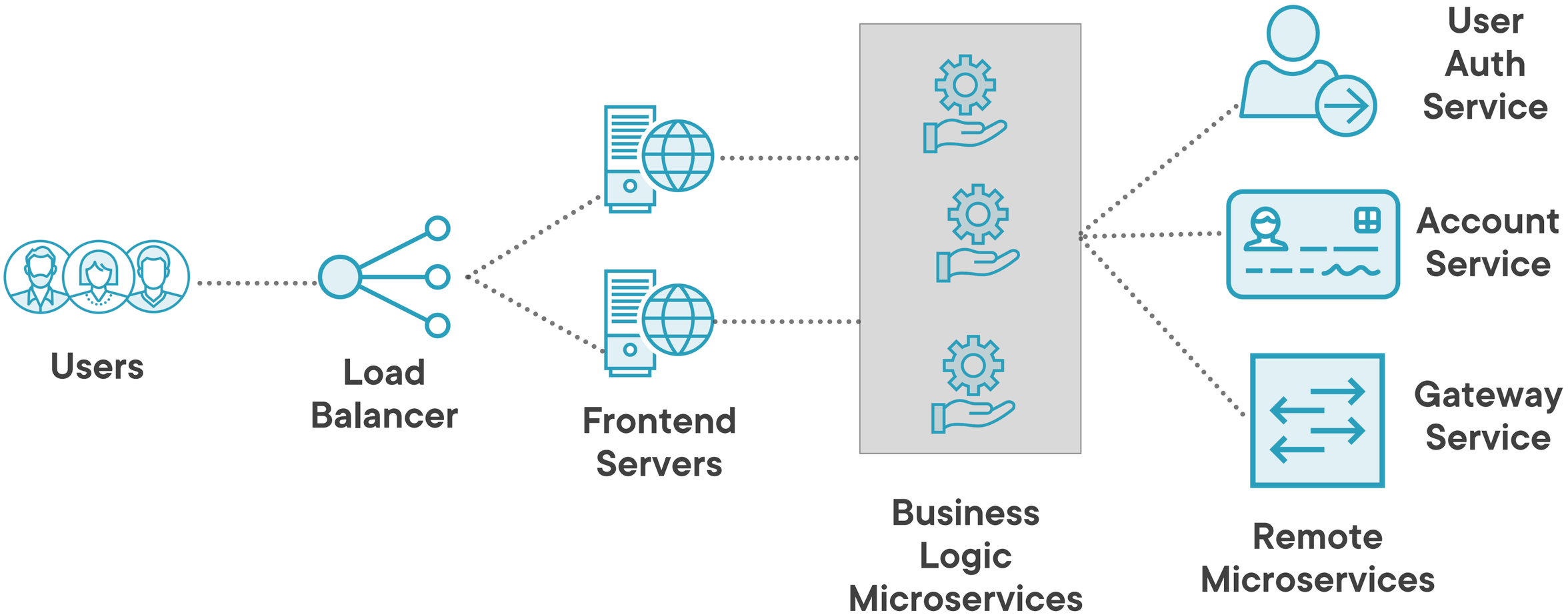
@karunops www.karunsubramanian.com

Overview

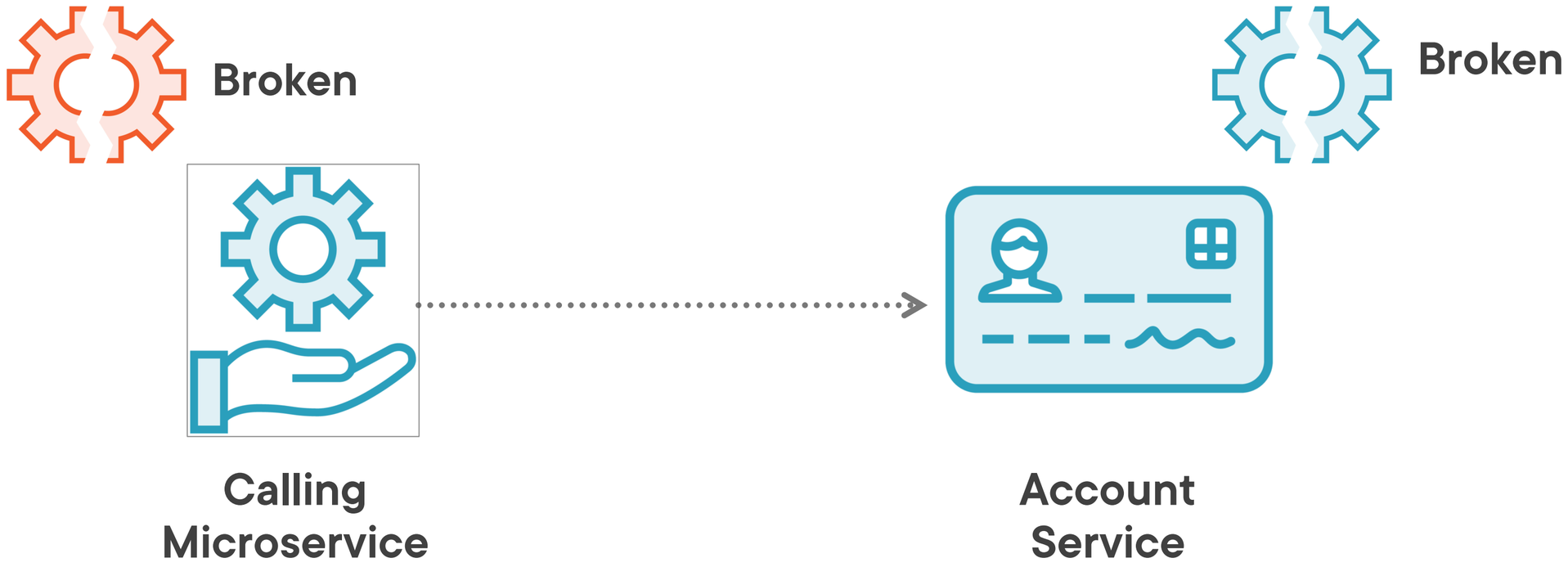
- Utilizing the circuit breaker pattern
- Comparing DNS load balancing vs. dedicated load balancer
- Transitioning to canary-based deployment
- Understanding distributed consensus
- Designing auto scaling based on load
- Implementing effective health checks

Utilizing the Circuit Breaker Pattern

Typical Distributed Architecture



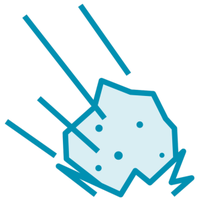
Deadly Pitfall



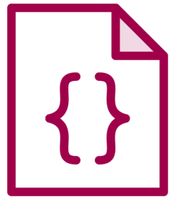
Poorly performing remote web services can cause the calling services to break

Circuit Breaker Pattern

When a remote service fails, fail fast



Helps to deal with failures to minimize user impact (failing gracefully)



Instead of bringing down the entire application, helps to run the application with reduced functionality

Avoids cascading failures

Circuit Breaker Logic

Track the number of failures encountered while calling a remote service

Fail (open the circuit) when a pre-defined count is reached

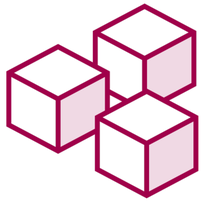
Wait for a pre-defined time and retry connecting to the remote service

Upon successful connection, close the circuit. Upon failure, restart the timer

Implementing Circuit Breaker

Use a proxy layer such as istio

Use Hystrix, opensource library from Netflix

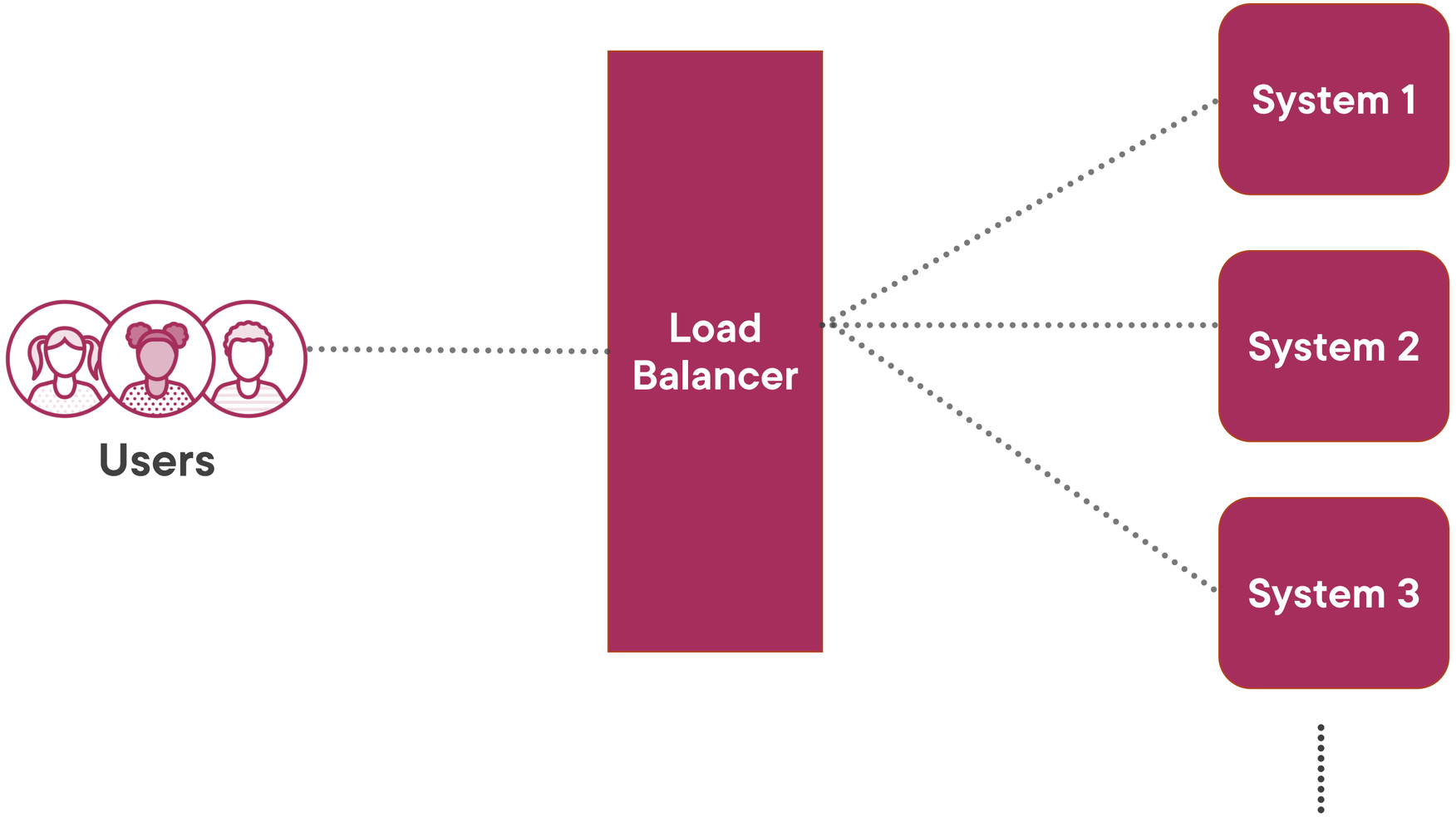


Istio is technology independent (Blackbox approach) and sits outside of your application

Hystrix is built with your code (Whitebox approach) but may provide additional functionalities

Implementing Effective Load Balancing

Load Balancers



Two Functions of Load Balancing

High Availability

Failures can be tolerated in a pool of servers

Performance

By adding more servers in a pool, effective performance is increased by horizontally spreading the load

Three Basic Ways to Load Balance

DNS

Closest to the client. Multiple A records enable client to choose one at random.

Hardware Load Balancer

Feature-rich. Highly performant. Typically used within datacenter.

Software Load Balancer

Relies on host hardware. Can be useful for dev environments.

DNS Load Balancing

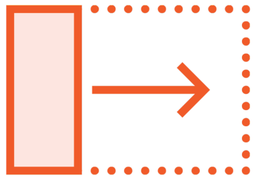
DNS server returns multiple IP addresses when a client looks up a hostname

DNS server can be configured to choose a datacenter closest to the client's geographical location

DNS server can also determine the targets based on load/capacity of the datacenter

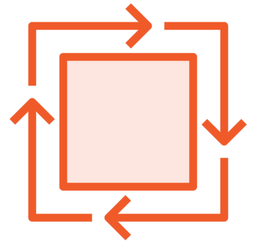
Among the IP addresses returned, client chooses an IP at random

Potential Problems with DNS Load Balancing



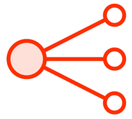
Not easy to control (client picks an IP randomly)

Requests can end up being sent to a server that is down



Clients cache DNS entries heavily and it is not easy to clear them

Dedicated Load Balancers

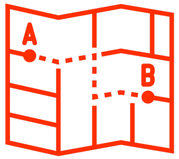


Layer3 (Network), Layer4 (Transport) and Layer7(Application) load balancers

L3 load balancers operate on source and destination IP addresses

L4 load balancers operate on IP + Port (TCP protocol)

L7 load balancers operate by processing application data such as HTTP URL, cookie, header etc.



L7 load balancers provide rich functionality for routing

Load Balancing Methods

Round Robin

Targets are rotated in a loop

Weighted Round Robin

Manually assigned weight influences the backend target

Least Loaded

Based on the reported load by the backend

Least Loaded with Slow Start

Prevents flooding the least loaded backend

Utilization Limit

Based on the reported utilization (QPS) by the backend

Hardware vs. Software Load Balancers

Hardware Based Load Balancers

Purpose built appliances

Feature rich

Can be expensive

Examples: F5, Imperva

Software Based Load Balancers

Software running on commodity hardware

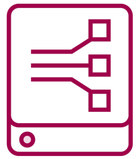
Comparable. Fast evolving

Relatively less expensive

Examples: HAProxy, NGINX

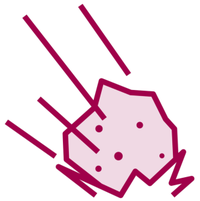
Transitioning to Canary-based Deployment

What is Canary-based Deployment?



Rolling out changes to small number of targets before releasing to all

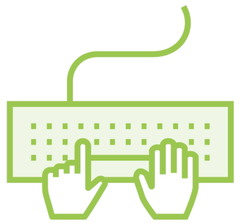
Like an insurance policy against bad releases



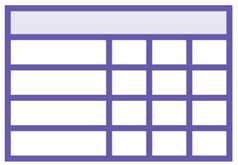
Can help reduce or eliminate user impact

Not a substitute for integration/load tests

How to Perform Canary-based Deployment?



Exact method will depend on your environment



Release up to 1% of the servers

Release on one server first

Proceed to release for early adaptors

Ensure that there are no catastrophic failures (server crash)

Release for all users

A failed canary is a serious issue. Analyze and implement robust testing process so that issues can be caught during testing.

Understanding Distributed Consensus

Inherent Problems with Distributed Systems

Hardware fails

Nodes in a distributed system can randomly fail

Network outages are inevitable

We cannot always guarantee 100% network connectivity

Need for a consistent view

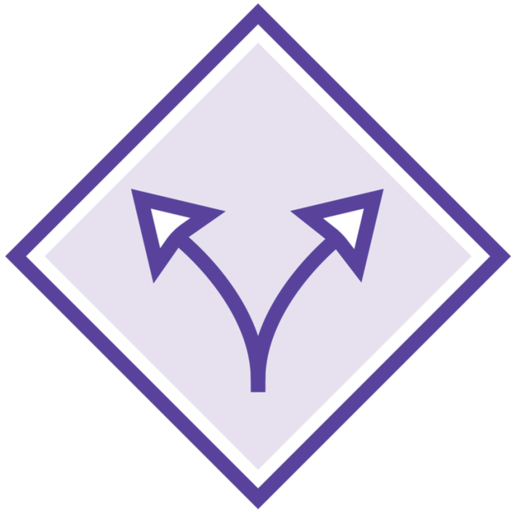
All nodes in a distributed system must achieve a consistent view of system state

CAP Theorem

A distributed system cannot simultaneously have these properties

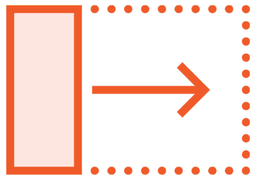
Consistency
Consistent views of data at each node

Availability
Availability of data at each node



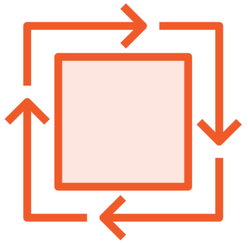
Network Partition
Tolerance to Network failures (which results in network partitions)

Distributed Consensus



Protocol to build reliable distributed systems

Ad-hoc methods such as heartbeats are not sufficient



Over the years, several protocols such as Paxos, Raft and Zab have been developed

Distributed Consensus Algorithm (Paxos)

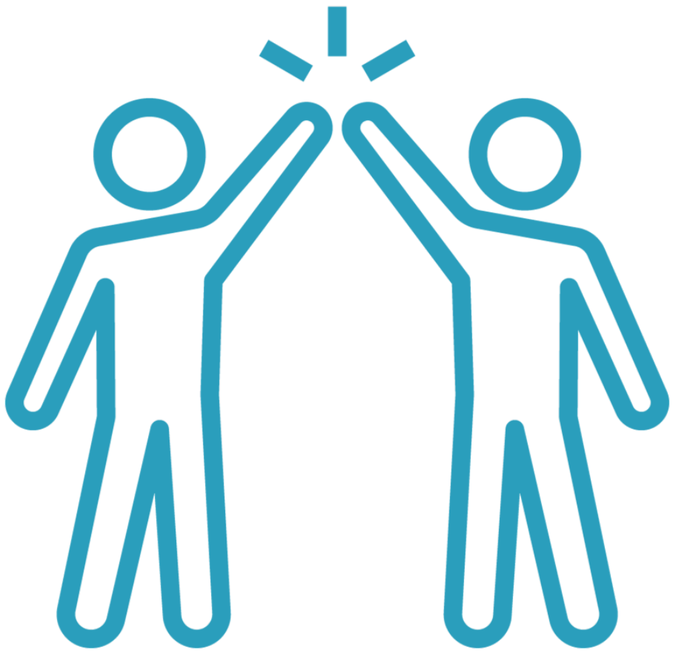
Paxos was the original solution to the distributed consensus problem

Systems in a distributed system send a series of proposals with a unique sequence number

When majority of the processes accept the proposal, the sender sends a commit message

The strict sequence of proposals solves the problem of ordering

Distributed Consensus Implementations



In addition to Paxos, other algorithms are available

- Raft
- Zab
- Mencius

Distributed consensus is generally implemented using commercial/opensource services

- Zookeeper
- Consul
- Etcd

Designing Auto Scaling

Autoscaling is a process by which the number of servers in a server farm is automatically increased or decreased based on the load

Four Benefits of Autoscaling

Reduces cost by running only the required servers

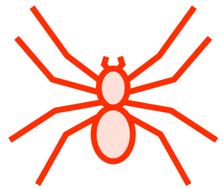
Flexibility to run less time-sensitive workload during low traffic

Automatically replace unhealthy servers (most commercial cloud vendors)

Increased reliability and uptime

Problems with Autoscaling

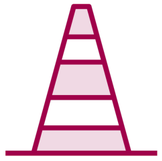
A dependent backend can get overwhelmed



Software bugs can trigger autoscaler to expand the server farm abruptly

Load balancing may not be intelligent enough to consider newly added servers (i.e. warm up period)

Autoscaling Best Practices



Scaling down is more sensitive/dangerous than scaling up. Fully test the scale-down scenarios

Ensure the backend systems (database, remote web service, etc.) can handle increased load

Configure upper limit on the number of servers

Have a kill switch that can be easily used

Three Systems to Act in Concert

Load Balancing

Minimize latency by routing location closest to the user

Nginx/HAProxy

Load Shedding

Serves requests that we can and drop excess traffic

Zuul/Envoy

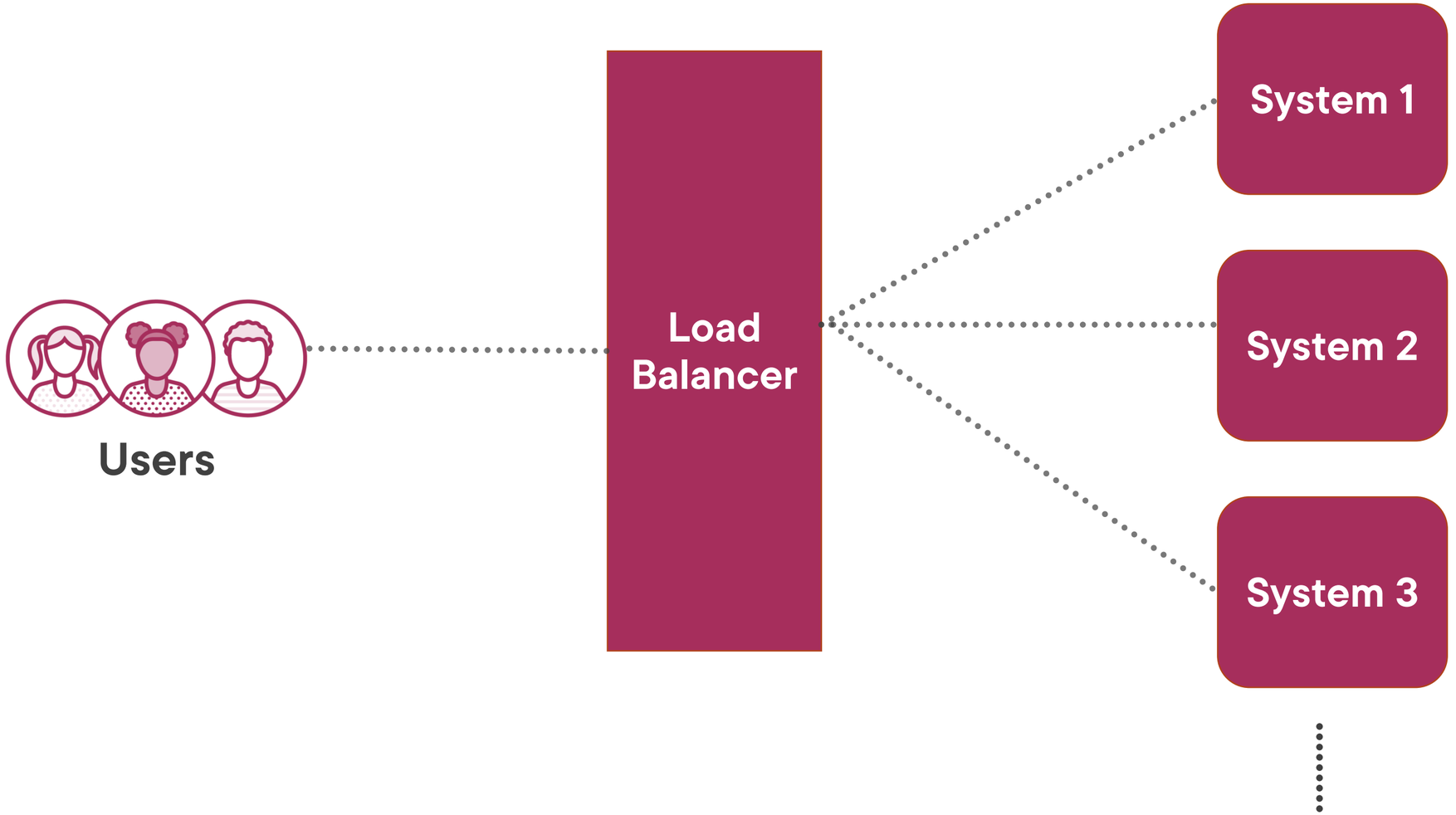
Autoscaling

Based on load, automatically scale up or down

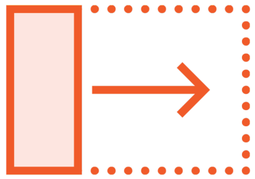
AWS/GCP/Azure

Implementing Effective Health Checks

Load Balancing Revisited



Load Balancers



Load balancers direct traffic to a set of backend servers

Load balancers need to know which servers are alive and healthy

Load balancers use health checks to determine the healthy servers

Effective Health Checks

Simple

Monitors for the availability of the backend server

Content verification

Sends a request and examines the response from the backend server

Reliable health checks

Health checks must cover situations such as ‘hung’ backend servers

Health checks with sophisticated content verification can increase network traffic. It can also be slow to mark a pool member as ‘down’

Summary

Utilize the circuit breaker pattern to fast-fail

- Hystrix (Whitebox approach)
- Istio (Blackbox approach)

Design load balancing with a mix of DNS and dedicated load balancers

- DNS load balancing may not be reliable

You must use canary releases

- Canary is a not a replacement for testing

CAP theorem states that you cannot simultaneously have

- Consistency
- Availability
- Tolerance to network partition

Summary

When designing auto scaling,

- Ensure backend systems can handle load
- Have a kill switch
- Develop accurate procedures for scale-down

Implement load balancing, load shedding and autoscaling to work together

Configure reliable load balancer health checks

- Simple ping is fast but may not be reliable
- Content-based is reliable but it may increase network bandwidth usage

Up Next:
Benefits of SRE