# SRE Classroom

Presented by Fabian Geisberger & Jean Joswig

## Agenda

Basics of Non-Abstract Large System Design

Problem Statement

Group Work Session (with break)

Example Solution

## Before we start...

There will be a handout.

We will share a link with further reading at the end.

Relax. Have fun.

Keep that little paper with the number on your seat.

Slides will be shared.

## Basics

## Disclaimer: Ain’t Nobody Got Time for That!

Due to time constraints this is not a complete discussion of large system design abstractions/solutions/patterns.

There are other talks that discuss this in more depths.

## Basics - Topics

- Requirements & Scaling
- Dealing with Loss
- Keeping State
- Non-abstract Design

## Basics: Requirements & Scaling

## Requirements

Identify Service Level Indicators (SLI) and Objectives (SLO)

- Data freshness
- Availability
- Latency

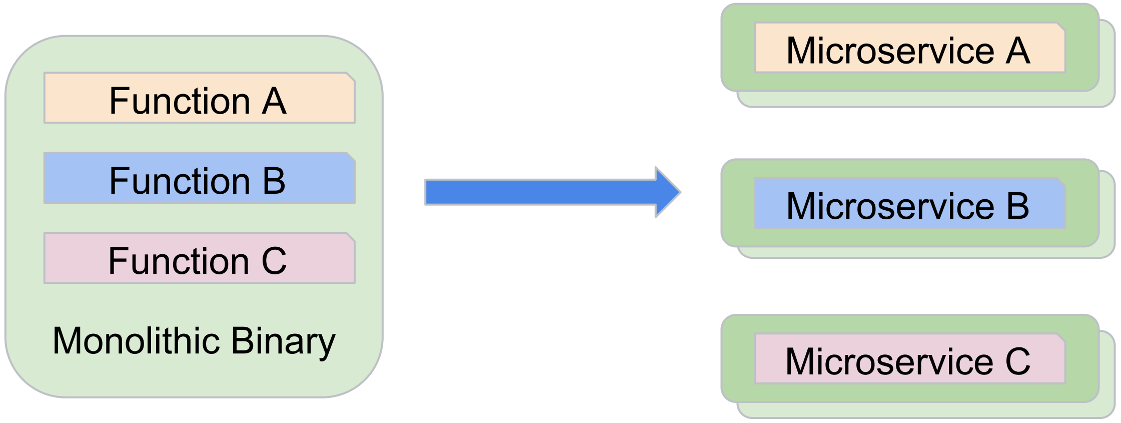
Sample SLO: 99%ile of queries returns valid result within 100ms

Service Level Agreement (SLA) is the contract containing all relevant SLOs and the punishment if it’s violated.

Design for current and consider future scale: What about 10x?

## Scaling via Microservices

Replace a monolithic binary with distinct microservices.

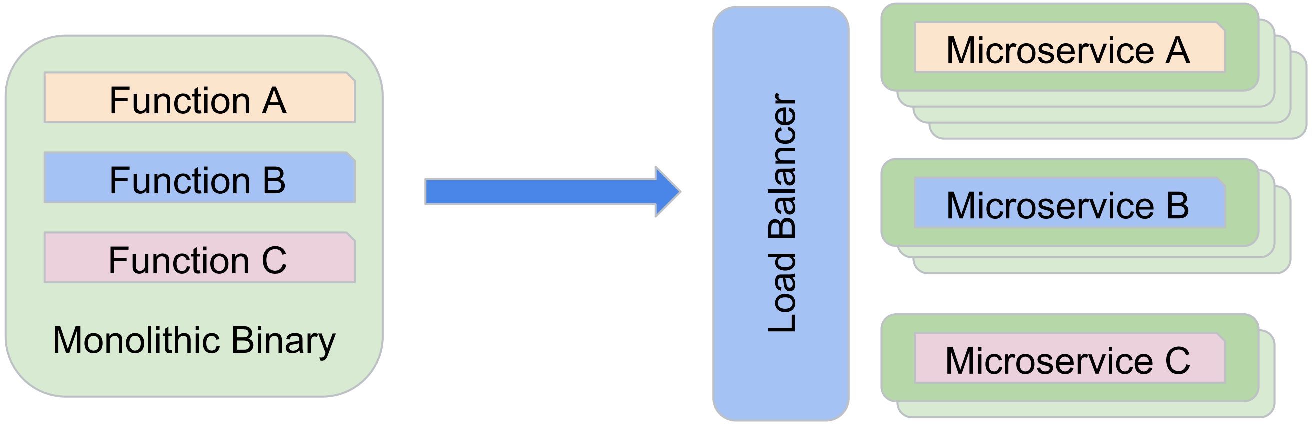


## Microservices Add Flexibility

Allows horizontal scaling. Makes functional scaling easier.

Each component can scale independently after deployment.

Add load balancers for work distribution.



Google

## Scaling in Distributed Systems

Typically refers to horizontal scaling.

Other relevant types of scaling:

- Geographical: More physical locations
- Functional: Different feature sets

## Dealing with Loss and Failure

## Failure Domains

Failure is not an option, it is certainty.

- Single process
- Machine
- Rack enclosure
- Datacenter
- Servers running the same binary
- Global configuration system

Comic by KC Green - gunshowcomic.com

## Dealing with Failure

Preferred: Make it somebody else’s problem.

Cloud providers offer various ready-made solutions.

Be aware of their limitations.

## Defending Against Failure

**Decouple**: Spread responsibilities across multiple processes.

**Avoid global changes**: Use a multi-tiered canary.

**Spread risk**: Don’t depend on one backend.

**Degrade gracefully**: Keep serving if configs are corrupt or fail to push.

## Achieving Reliability

Run N+2 stacks geographically distributed.

N = deployment large enough to deal with standard load

Why +2?

- Planned maintenance
- Unplanned maintenance

## Keeping State and Data

## Keeping State

Useful for consistency, performance and reliability.

Regardless of the amount of data involved, it all comes down to global consensus.

Always prefer stateless. It is easier.

## Speaking about Consistency

Global consistency critical for large systems.

Find authoritative instances of other services (leader election).

Quick discovery of sharded data.

Various algorithmic implementations for this: Paxos, Raft

## Consistency - CAP

CAP = consistency (C), availability (A), partition resilience (P)

Networks are not reliable, but partitions are rare.

Need to be able to converge after partition.

E.g. Spanner: CP plus high availability

## Hot Data and Hotspotting

Some data is accessed more frequently than others.

Frequent access to the same data can cause servers to overload.

It can also create bottlenecks via I/O.

One solution: Sharding.

## Caches

Move data access to faster I/O to handle hot data (e.g. to RAM).

Improves user experience.

Reduces load on backend.

Capacity vs performance cache. Capacity cache can be dangerous.

Should not form part of the request-per-second capacity guarantee for a service.

## Non-abstract Design

## Suggestions

For each microservice consider:

- Disk I/O
- QPS
- Network Bandwidth

Use back of the envelope numbers - easier to calculate in head.

- One day has 25 hours.
- Each month has 30 days.
- Each year has 300 days. Or 400.

## Problem Statement

## Problem Statement

Design a photo-serving system that will have an initial global user base of 1 million users, each user uploading 50 photos per day.

The UI provides two pages: a thumbnail view with 10 thumbnails per page, based on a search result (user, latest, topical, hashtag); and a detail view of a specific image.

An uploaded compressed picture is 4 MB in size. Each thumbnail is 256 KB. Provide a solution for the complete product.

## SLOs

Serve the thumbnail page (10 thumbnails) within 250 ms at the 99.9 pctl (HTTP 200).

Serve the image page (1 image, full-resolution) within 200ms at the 99.9 pctl (HTTP 200).

These SLOs do not apply to data older than 30 days.

## Available Infrastructure

Network (SLA 99.99 % availability).

Storage system (SLA read 100 ms at 95 pctl, write 200 ms at 95 pctl).
Storage system handles downsampling, provides thumbnails on request.
Globally replicated, eventually consistent in the order of minutes.

Multiple datacenters in 3 regions: Europe, Asia, North America.

Each unavailable for one week each quarter for planned maintenance.

## Available Hardware

HDD Server: 24G RAM, 8 cores, 2x2TB hard drives, 10Gb ethernet

SSD Server: 24G RAM, 8 cores, 2x1TB SSD, 10Gb ethernet

As many as you need if you can make a case for it.

## Form up in Groups!

The number on your seating card is your group.

Each group has sticky notes, markers, and a facilitator.

Suggestion: Start designing a system diagram for one datacenter.
⇒ Does it meet the SLO?

Then expand the solution for multiple datacenters.
⇒ Does it still meet the SLO?
⇒ Run numbers and create a bill of materials

## Form up in Groups!

Perfect solution not required. Use back of the envelope style of reasoning.

Laptop not required.

You’ll work in groups, assisted by facilitators. Check your seat card for your group number.

Relax, this is not a test ;)

Have fun tackling a technical problem together.

## Group Work Time

Ask any facilitator if you have any questions.

We have a (optional :p) break from 15:30 to 16:00.

Meet back in here at 16:45 for sample solution.

## Example Solution

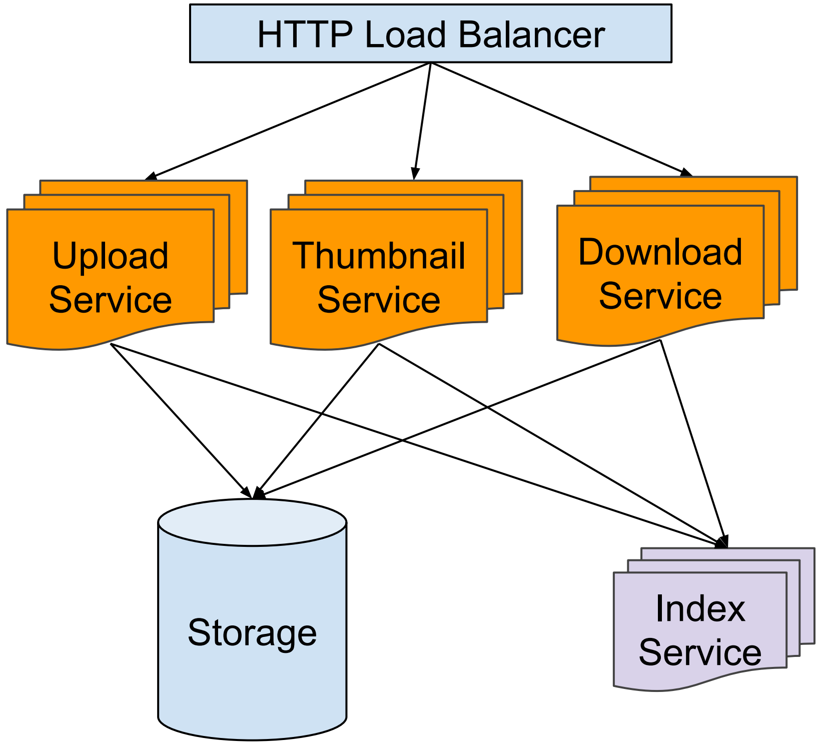
# Example Solution

A load balancer.

Dedicated web servers.

An index service.

A storage service.

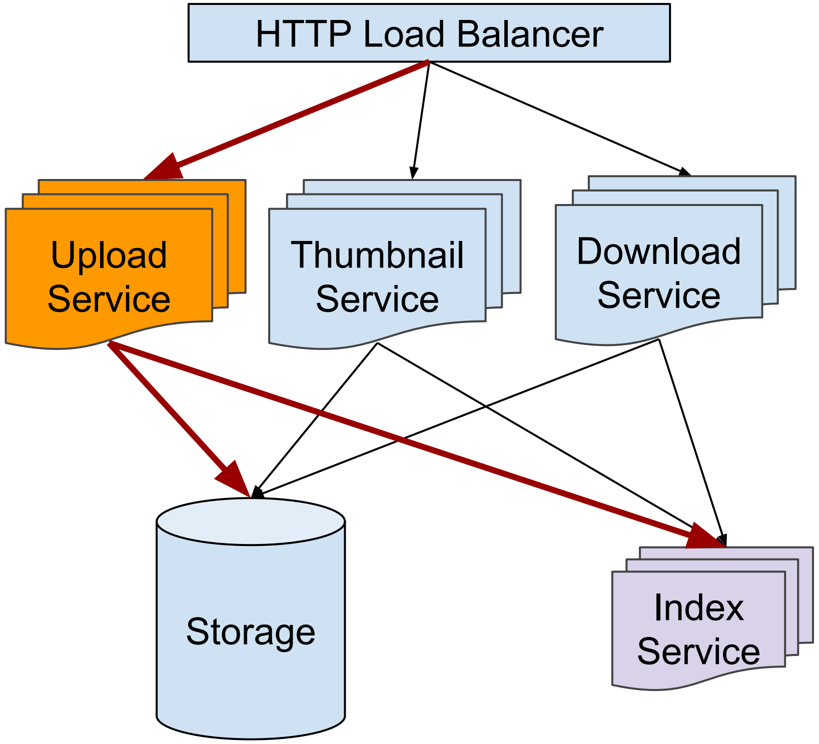


# Uploading Pictures

Send query to upload service.

Pipe data to storage.

Update index service.



# Searching

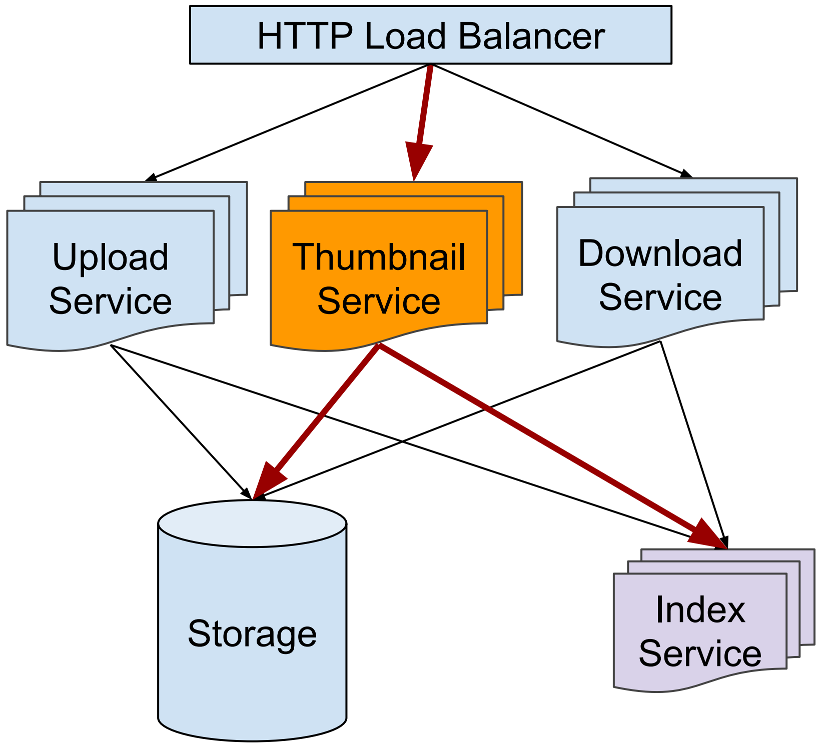
Send query to thumbnail service.

Query index service.

Retrieve thumbnails from storage.

Return thumbnail page.

Serve cached data if possible.



# Downloading a Full Size Picture

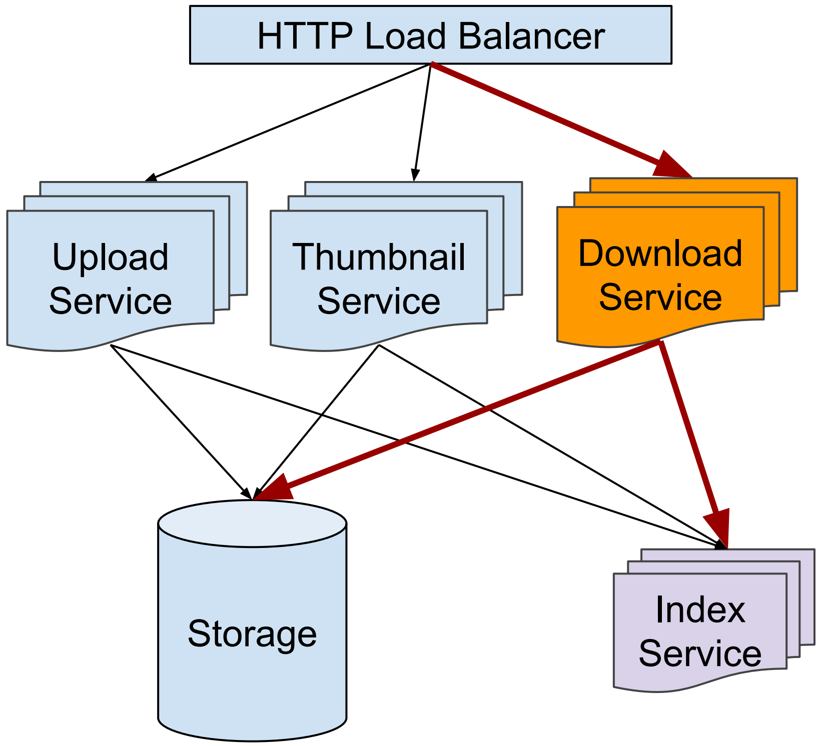
Send query to download service.

Query index service.

Retrieve picture blob from storage.

Serve picture data.

Serve cached data if possible.

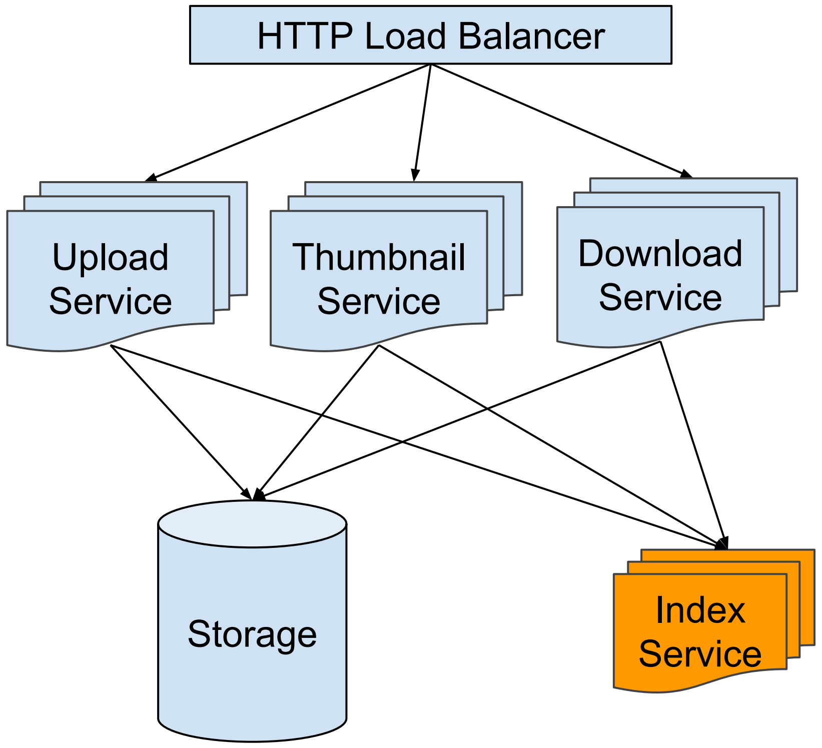


# Index Service

Stores metadata (tags, storage ID).

Assume metadata size of 8 KB per picture.

Globally consistent view.



## Example Solution - Bill of Materials

## General QPS and Storage

1.000.000 daily users with 50 uploads each

1.000.000 * 50 / 24h / 3600s ~= **579 qps**

Data uploaded per day: 1.000.000 * 50 * 4MB = **191 TB**

For 30 days: 191 TB * 30 = **5.59 PB**

Thumbnail pages - assume 5 searches per user organically.

1.000.000 * 5 / 24h / 3600s ~= **58 qps**

Assume each search results in one full size download as well ~= **58 qps**

## Upload Service - Bandwidth

Incoming picture data per second:

579 QPS * 4 MB = 2314 MB/s -> 2314 / 1000 MB/s -> 2.314 NICs

Theoretical maximum for data streaming to storage including handling incoming:

1000 MB/s / 4MB = 250 QPS per NIC

**That means we need 3 servers worth of NIC.**

## Upload Service - Timing

RPCs to storage backend are streamed in parallel.

No user-facing SLO for storing.

Storage write starts within 200 ms. Network time per picture 4 ms.

Actual blocking time on NIC is 4 ms -> 250 qps

**Need 3 servers if we completely rely on remote storage.**

## Thumbnail Service - Bandwidth

58 qps incoming for thumbnail pages. 10 pics of 256 KB per result page.

58 qps * 10 pictures * 256 KB ~=  145 MB/s

**One NIC is sufficient.**

## Thumbnail Service - Timing

Storage reads start within 100 ms of RPC call.

Index service query ~0.5 ms

Network time per results page: 2.5 ms

SLO wait time per query 103 ms at 58 QPS.

Actual blocking time on NIC is 2.5 ms -> 400 qps

**Need 1 server if we completely rely on remote storage.**

## Download Service - Bandwidth

58 qps incoming for full size pics. Each pic is 4 MB.

58 qps * 4 MB ~= 232 MB/s

**One NIC is sufficient.**

## Download Service - Timing

Storage reads start within 100 ms of RPC call.

Network time per results page: 4 ms

Index service query ~ 0.5 ms

SLO wait time per query ~105 ms.

Actual blocking time on NIC is 4 ms -> 250 qps

**Need 1 server if we completely rely on remote storage.**

## Index Service - Bandwidth

Metadata per photo is 8 KB.

579 qps from upload service containing single metadata.
579 qps * 8 KB payload ~= 4.5 MB/s

Search queries return metadata for 10 entries:
58 qps * 10 * 8 KB reply ~= 4.5 MB/s

**One NIC can easily do this.**

## Index Service - Storage

1.000.000 daily users * 50 pics * 30 days = 1.500.000.000 entries

1.500.000.000 entries * 8 KB ~= 11.2 TB

**3 HDD servers or 6 SSD servers.**

SSD more future proof for I/O, but HDD are cheaper.

## Index Service - I/O Timing

~579 write qps result in ~579 IOPS

~58 search qps result in ~580 IOPS

HDD: Seek 2 ms + read/write 8K seq 0.04 ms ~= 2.04 ms

~490 IOPS per disk -> **2 servers needed**

SSD: (random 4K read) 0.02 ms + (4K seq r/w) 0.00003 ms ~= 0.02 ms

~ 50K IOPS per disk -> **1 server needed**

## Index Service - Write QPS

Remote storage is only eventually consistent.

Need a globally replicated search index.

Writes bounded by global consensus. 150 ms RTT =~ 6 qps for writes.

579 write QPS =~ 100 entries per batch

Since this is only metadata **one NIC can deal with this**.

## Index Service - Sizing Needs

| | HDD | SSD |
|---|---|---|
| Bandwidth | 1 | 1 |
| Storage | 3 | 6 |
| IOPS | 2 | 1 |
| **Total** | 3 | 6 |

## Load Balancer - Bandwidth

Load balancer is fully network bound and needs to be able to deal with all traffic.

Upload traffic: 579 QPS, ~2314 MB/s
Thumbnail traffic: 58 QPS, ~145 MB/s
Full size traffic: 58 QPS, ~232 MB/s

Total: 695 QPS, ~2691 MB/s

**We need 3 servers.**

## Footprint - Summary

| | Bandwidth | Timing | Storage |
|---|---|---|---|
| Load Balancer | 3 | n/a | n/a |
| Upload Service | 3 | 3 | n/a |
| Thumbnail Service | 1 | 1 | n/a |
| Download Service | 1 | 1 | n/a |
| Index Service | 1 | 1 | 3 (HDD) |
| **TOTAL per DC** | 11 | | |

Google

## Footprint - Increased Robustness

Account for non-uniform load (multiply by 1.25).

Get rid of SPOF.

| | Bandwidth | Timing | Storage |
|---|---|---|---|
| Load Balancer | 4 | n/a | n/a |
| Upload Service | 3 | 3 | n/a |
| Thumbnail Service | 2 | 2 | n/a |
| Download Service | 2 | 2 | n/a |
| Index Service | 2 | 2 | 4 (HDD) |
| **TOTAL per DC** | 15 | | |

## Global Footprint

15 servers per cluster.

3 DCs * 15 servers = 45 servers global footprint

In reality and given our architecture you can start with less and use auto-scaling for each component.

## Improving with caches

We have unused resources on most servers. Use it for caches.

Performance of cache depends on access speed.

RAM: ~24 GB/s

SSD: ~1 GB/s, HDD: ~200 MB/s

Network: ~1 GB/s

Caches can be distributed but then have added bandwidth costs.

## So Many Unexplored Angles

Monitoring and alerting

Realistic storage backend behavior (degradation)

Long-term storage (hot and cold storage)

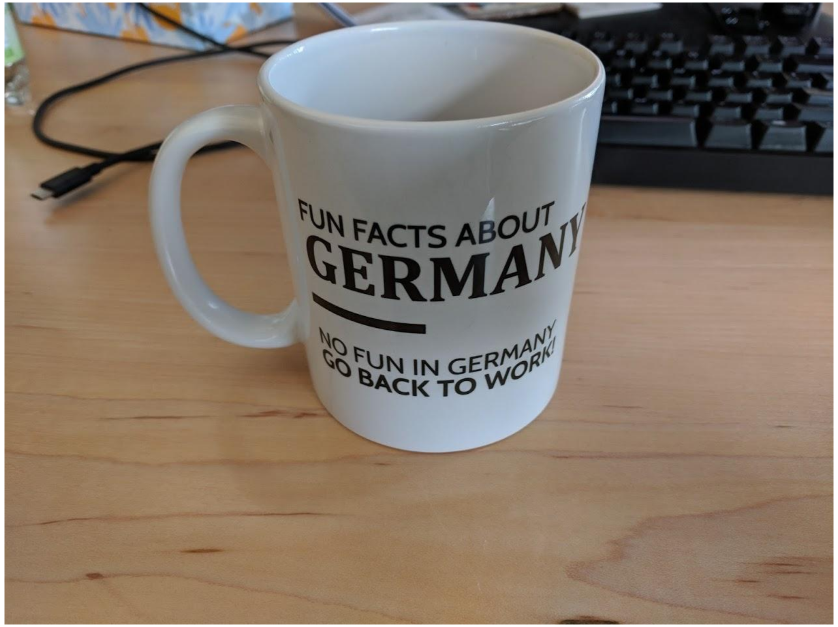
Capacity growth (per year, retention)

Privacy requirements (GDPR anyone?)

Toil (rollout, maintenance) - more a process thing though

## Wrap-up

## Questions?



Jean’s favorite mug at work :)

## Questions?

Thank you!

To our facilitators in the room!

Contact us

fabi@google.com

joswigjn@google.com

Further reading

https://goo.gl/LGea4R

Google