#ASLI ENGINEERING

Scaling Microservices

SWIPE

BY

ARPIT BHAYANI

Scaling Microservices

About me

I am Arpit Bhayani, Director of Engineering @ Unacademy

Something interesting

1. I run a YouTube channel named #Asli Engineering
   - 3 videos & essays engineering topics every week
     Microservices, Outage Dissections, CPython Internals
     Hash Tables, etc

2. I am building a programming language for kids
   Revine → syntax like python
   ↘ create small in-browser games

You can connect with me at arpitbhayani.me

@arpit_bhayani

Today, we talk about Scaling Microservices

What is scalability ?

ability to handle large number of
users concurrently without degrading the experience
without bloating up the cost

Scaling is pretty simple, if you have a lot of money

1. add more CPU, RAM, Disk, and Network
   ↳ Vertical scaling
2. add more machines
   ↳ horizontal scaling

But scaling is contextual

You need not design a system that handles
50,000 RPS on day 1

but maybe after 5 years ...

Depending on the org and product stage,
fence your scaling requirement

Scaling depends on SEO

1. Stage of the product
   ↳ You might not need microservices on Day 0
2. Engineering prowess
   ↳ Can existing engineers pull it off? ↗ Managed Services, Familiar Stack
3. Organization maturity
   ↳ Is org / product stable enough?

* Assess the situation well, before jumping the gun!! There is no going back!

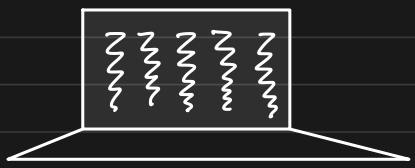
Most engineers suffer from "over-engineering" syndrome

We need someone who does not say YES all the time
↓
manager, architect, tech lead

Scaling microservices on low-level

* Unit Tech Economics

It is super important to know your numbers



How many users can each of your server handle?

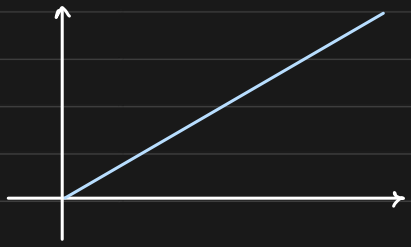
users → requests

1: Understand single-machine capacity

Why? because linear amplification

↓

predictability



if one machine handles 5000 RPS

to handle 50,000 RPS on a certain event, we need $\frac{50000}{5000}$ = 10 machines

No unreal expectations

Two fold impact → cost projections are predictable

→ capacity planning is simple for big days

eg: some Big Marketing Campaign

2: Go full throttle

Servers have multiple CPU cores ... leverage it

Make workloads concurrent wherever possible

- Multi-threading
- Multi-Processing

or concurrent IO

eg: AsyncIO & Event loops

But, beware about critical sections, concurrent updates

Hence use concurrent data structs to prevent data inconsistencies

* Keep your critical section as small as possible

Never put unnecessary statements in your critical section.

3: Batch wherever possible

Servers have enough RAM and Disk, use them to Batch

eg: do not make one DB write every REQ, see if you can batch update
do not make N-reads, see if you can batch

↳ Join v/s Multiple selects

4: Pick the right hardware

Understand the workload, usecase and access pattern and pick the right hardware.

- CPU intensive
- memory intensive
- network intensive
- GPU intensive
- Disk intensive

eg: video encryption
↳ CPU intensive

eg: training ML model
↳ CPU heavy

eg: EMR
↳ Disk heavy

eg: image upload service
↳ network heavy

In order to pick the right hardware,

Measure the right metrics
↳ APM Tools
↳ infra monitoring tools
↳ prometheus

} would give you visibility on the actual Bottleneck!!

Scaling microservices at high-level

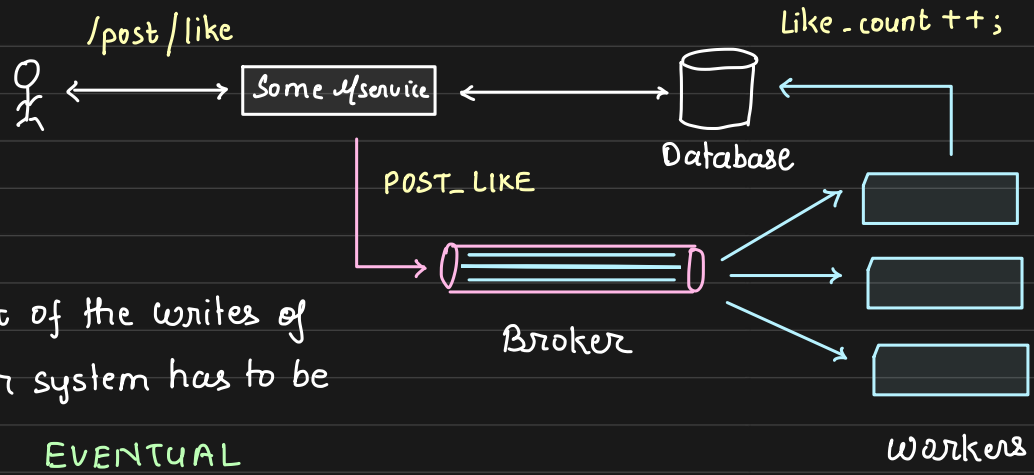
Simple systems scale

Guiding principle

Engineers love to over-complicate arch but it is totally unnecessary.

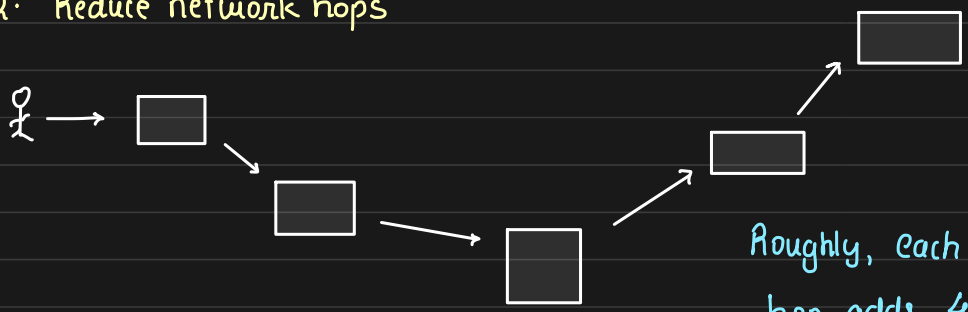
1: When in doubt, design it asynchronous

> What need not be done in realtime should not be done in realtime



Most of the writes of your system has to be EVENTUAL

2. Reduce network hops



With microservices, you can very easily end up doing ↑

Roughly, each network hop adds 4ms of latency!! + Sync Dep timeout / wait

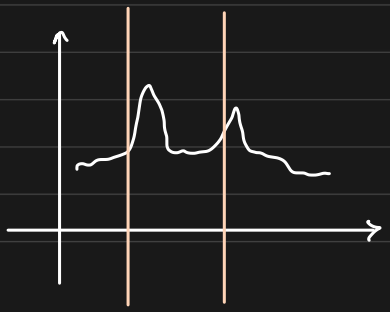
A great way to achieve this is share a DB

* counter intuitive but a great hack

3. Handling Spikes * Autoscaling groups might not be sufficient

Give your users a great Experience no matter what

Scaling is not instantaneous



If you know your users (traffic pattern) well, you can pro-actively scale up your (relevant) services to handle the load

Thank you for your time

Arpit Bhayani

@ arpit_bhayani