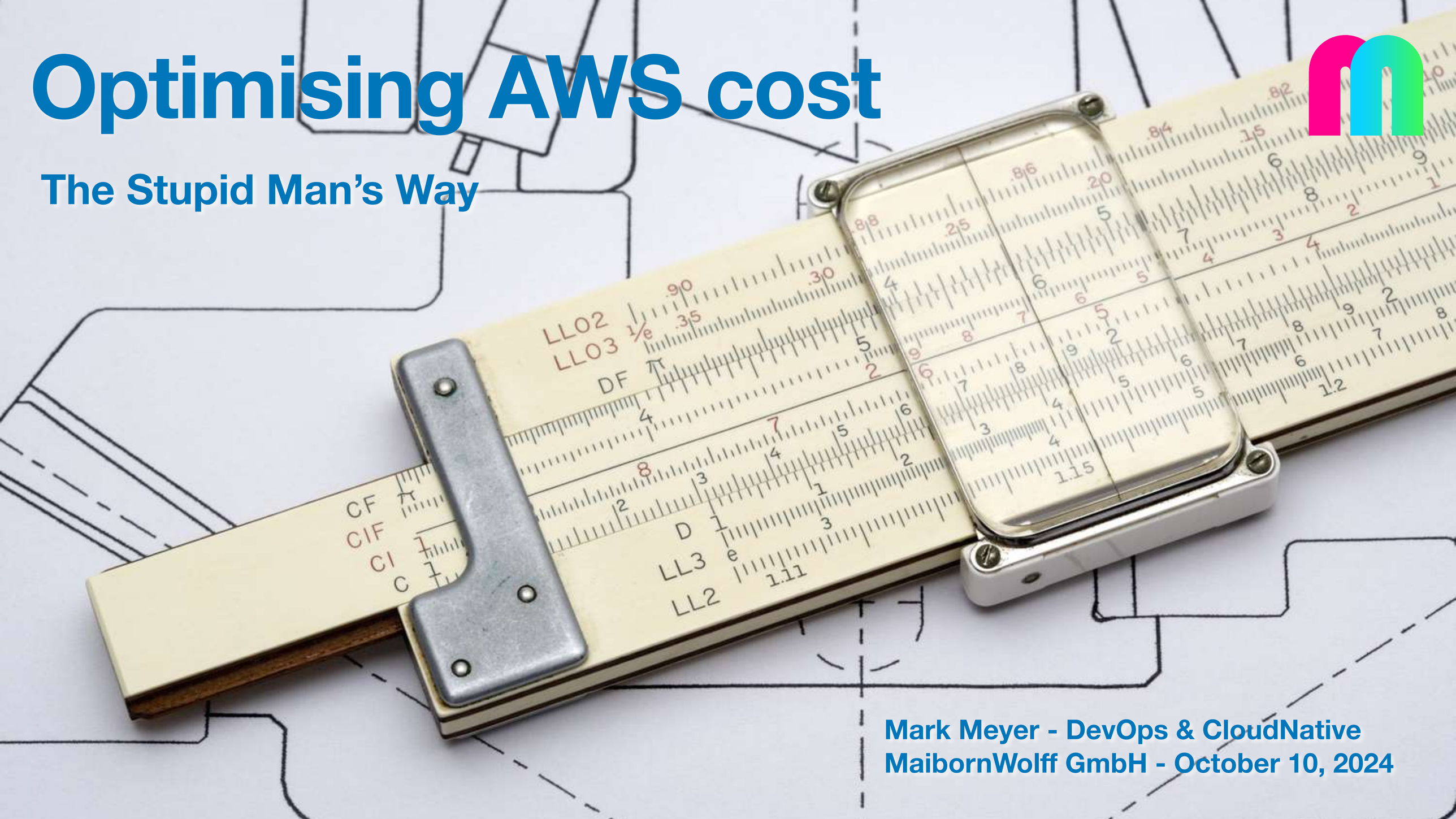
Optimising AWS cost

The Stupid Man's Way



Mark Meyer - DevOps & CloudNative
MaibornWolff GmbH - October 10, 2024

What to expect

Answers? More questions!

This talks topics:

- Why? Two principles we'll apply
- What? Three core strategies
- How? Nine practices to apply

I'll try to keep the focus on a conceptual level. I'll share ideas with some pointers on how it can be implemented (e.g. in tooling, AWS services).

The Principles

We don’t compete with AWS.

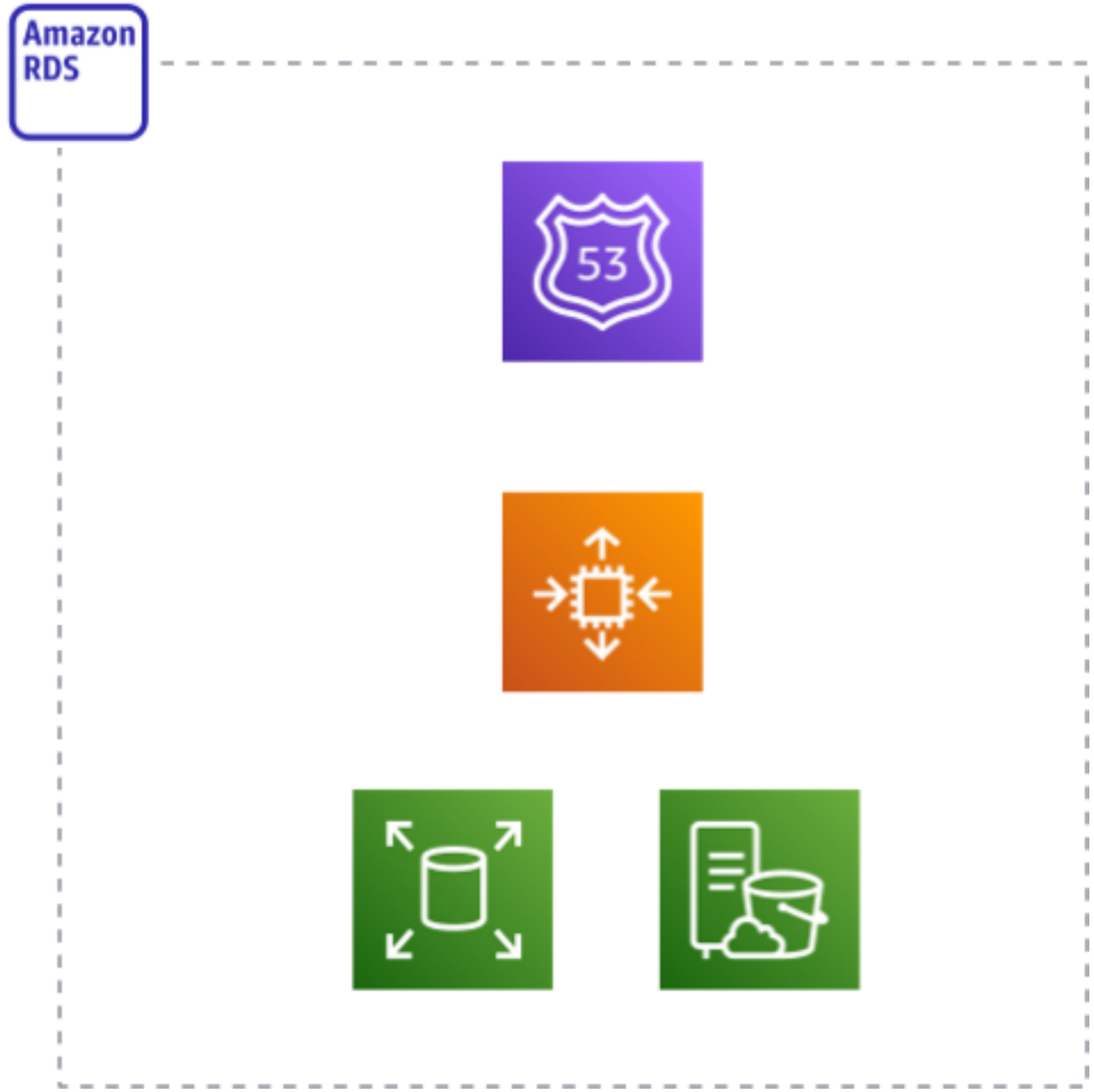
- Author’s personal opinion

AWS Service

Service “RDS”

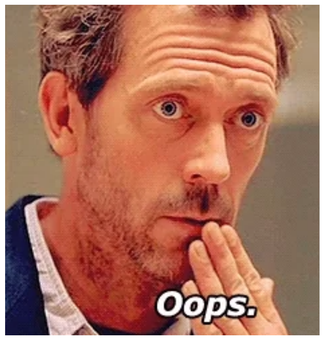
=> AWS builds services on AWS

- RDS is made of:
  - EC2
  - EBS
  - S3
  - Route53
- “I can do that!”



AWS Service

RDS as an example



- Rock solid service
- Largest cloud op on earth
- 38% EBIT margin
- 20+ years experience
- Backed by a small army

- “An operator”
- Random interwebs dude
- $120k yearly spend?
- High risk
- It’s you vs. All the bugs

Every $ we spend on AWS should have an outsized impact.

- Author’s personal opinion

AWS Economics

CW Logs vs. S3+Athena (1)

```
[3]: ret, daily, read = symbols("(retention) (daily) (read)")
```

```
[4]: period = 30. # number of days to account for, usually 1 month
```

```
[51]: cwlogs = ((daily*period*0.5) +          # $0.50 per gb ingest
                (daily*ret*0.03*0.2) +        # $0.03 per gb stored (Standard), at a compression factor of 0.2
                (read*ret*daily*0.005*0.2))   # $0.005 per gb scanned, at a compression factor of 0.2
```

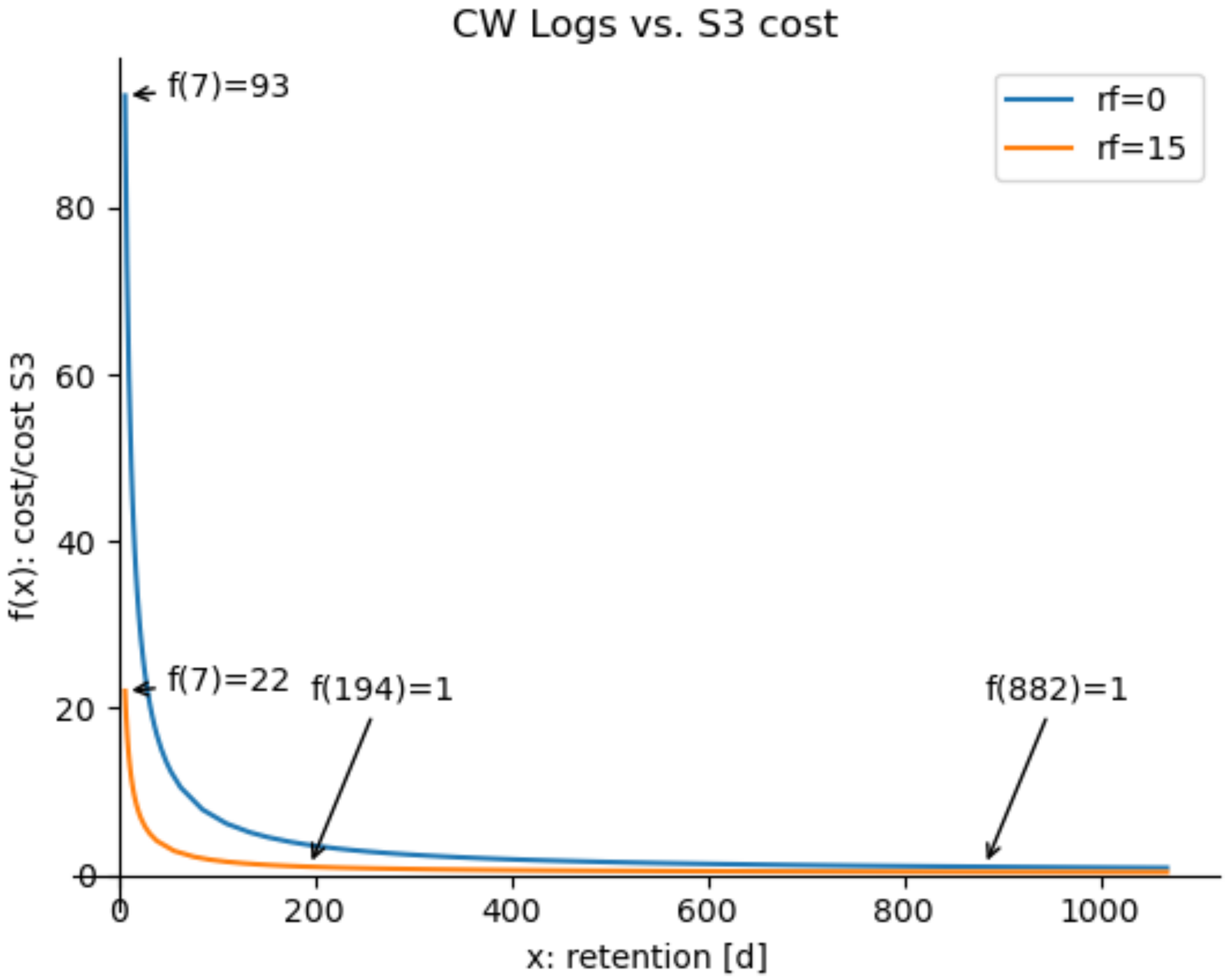
```
[8]: s3athena = ((daily*ret*0.023*s3compression) +        # $0.023 per gb stored
                 (read*ret*daily*0.005*s3compression))   # $0.005 per gb analyzed
```

cheap to read!

No ingest fee!

AWS Economics

CW Logs vs. S3+Athena (2)



AWS Economics

CW Logs vs. S3+Athena (3)

- CWLogs/Athena are “Logs for the auditor”
- S3+Athena is cheaper at a minimum of around 30GB/d
- Both Athena and CWLogs have a bit of a learning curve
- If logs hamper dev productivity buy a product like DataDog, honeycomb.io
- In the end, all things being equal, the decision is an economical one

The Strategies

Make cost attributable

Strategy No.1: “Transparency”

- Make sure your least favourite service really is too costly
- Define dimensions on which to attribute, suggestion:
  - Two levels of AWS services (EC2, instance type,…)
  - Two levels of business/product criteria (customer, product area,…)
  - Two levels of organisational criteria (dept., team,…)
- You end up with a “cube” that splits the bill by these dimensions

Mop the floor

Strategy No.2: “Bottom up”

- Start with your AWS service dimensions
- Look at the consumption per service
- Eliminate things that don’t spark joy:
  - Simple: e.g. GP2 volumes (still a thing in 2024)
  - Medium: Rightsizing and orphaned resources
  - Harder: Disproportionate AWS Config/GuardDuty charges
- Rule of thumb: having 30% supporting services on your bill is not unusual

Re-Engineer for purpose

Strategy No.3: “Top Down”

- Start with your business dimensions
- Find which parts of the operation are too expensive (e.g. compared to the competition, or because users don’t value them)
- Drill down to AWS services until you find the cost driver(s)
- The solution very likely involves re-engineering both (AWS) Infra and the software that runs on top
- Purpose: make service more competitive

3 Strategies

Attribute
Mop the floor
Re-engineer

The Practices

The builders own the cost

Practice No.1

- If people are allowed to design systems without regard to cost, they will
- If people don’t know the cost of the systems they run, they can’t learn
- If you want to grow your AWS builders, you need to close the loop between design and operations
- As your business case changes, your AWS bill does too, hence cost optimisation needs to be a constant practice.
- You need some kind of implicit or explicit Deming cycle

Find usage excursions early

Practice No.2

- Tracking outliers in usage is a best practice
- At some, but not all, customers anomaly detection is in place
- “The bill” (aka the CUR) is a late tracking indicator, 48h delay
- If you want to avoid charges for usage excursions, track usage
- Service: “AWS Cost Anomaly Detection”, “Billing Alerts”, ok-ish

Track the unit cost

Practice No.3

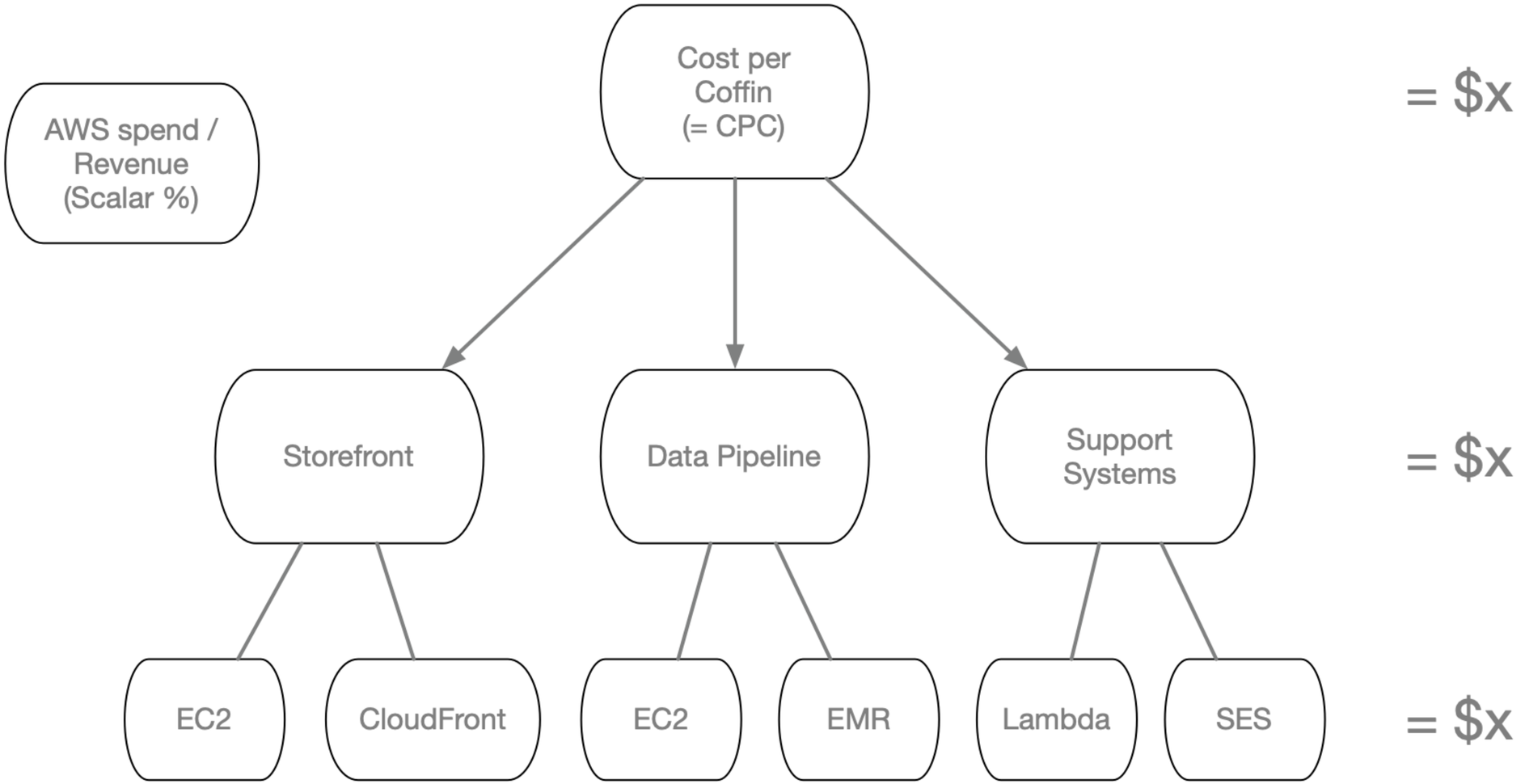
- Cost dimensions as basis
- Cost per quantum of service
- Value/revenue generating unit and drill down to AWS services
- Cost per…, Ex.: monthly active users, transactions, views, clicks, new customers, revenue generated, or just revenue
- A good cost model creates success stories

Track the unit cost

Practice No.3 - Sample

Stan's fomerly used coffins plc.

Unit cost broken down into AWS services = cost model



Leverage Architecture

Practice No.4

- Well Architected + Prescriptive Guidance
- You’ll find: Design alternatives & best practices
- & boilerplate: FlowLogs, CloudTrail, Cost & Usage Reports (full detail)

Leverage Architecture

Practice No.4 - Sample (1) - VPC Arch

Private Addressing

NAT

VPC

Endpoint

Check!

Public Addressing

IGW

VPC

IPv6 Addressing

IGW

VPC

Egress only IGW

Leverage Architecture

Practice No.4 - Sample (2)
VPC Interconnect

Specialise to create value

Practice No.5

- AWS is a company with global reach
- Knowing your customer/use case very well can open up an opportunity to do custom engineering
- The lever from a highly specialised piece of infrastructure can be huge
- Things that AWS doesn’t do, ex. DataDog, ~~Kafka~~, ZoneHero
- Competing: features or different fundamentals!
- AWS, homegrown, 3rd-party services need a regular “competitiveness check”

Rightsize compute

Practice No.6

- “Rightsize” means, to find optimal compute resources
- Rightsizing does not (always) mean increasing utilisation
- Doubling instance/container size => higher efficiency, same average utilisation, e.g. because more comm’s happen locally, caching
- Doubling instance/container size => lower efficiency; same utilisation, e.g. because of resource contention/locking
- Large shared instances reduce overhead (e.g. DS overhead in EKS)

Optimise storage

Practice No.7

- Retain less data for a shorter amount of time
- Retain it in cold(er) storage tiers
- Service: “Amazon S3 Storage Lens”
- “Auto Tiering” - for people in a hurry
- Test Glacier restore, before you commit
- Pick good EBS types, find out if your services supports storage tiers

Design for data movement

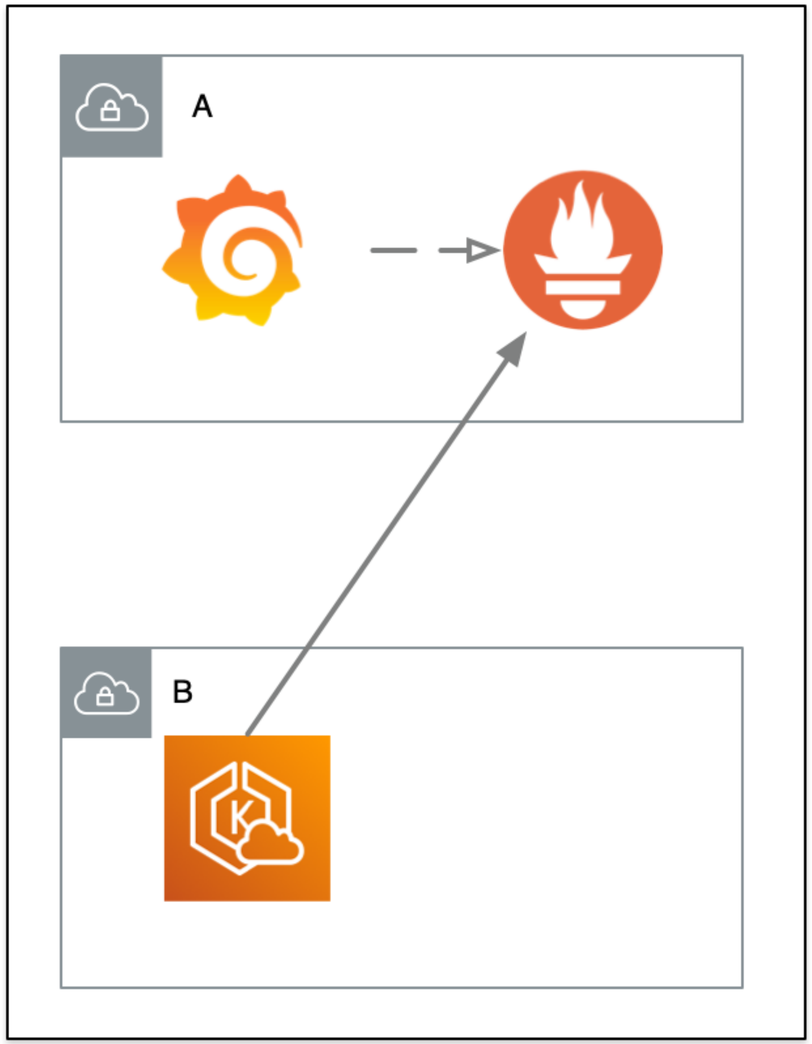
Practice No.8

- Ex: database, message broker, S3, SQS, Kafka, Redis, IP traffic
- In my naive mind, all of these tech’s can be a “message bus”
- Choice depends on: transfer volume, latency, message size, persistence, semantics, system context
- Data flow is paramount to both system performance and cost
- Keep data local (ex. X-AZ traffic, X-region traffic)

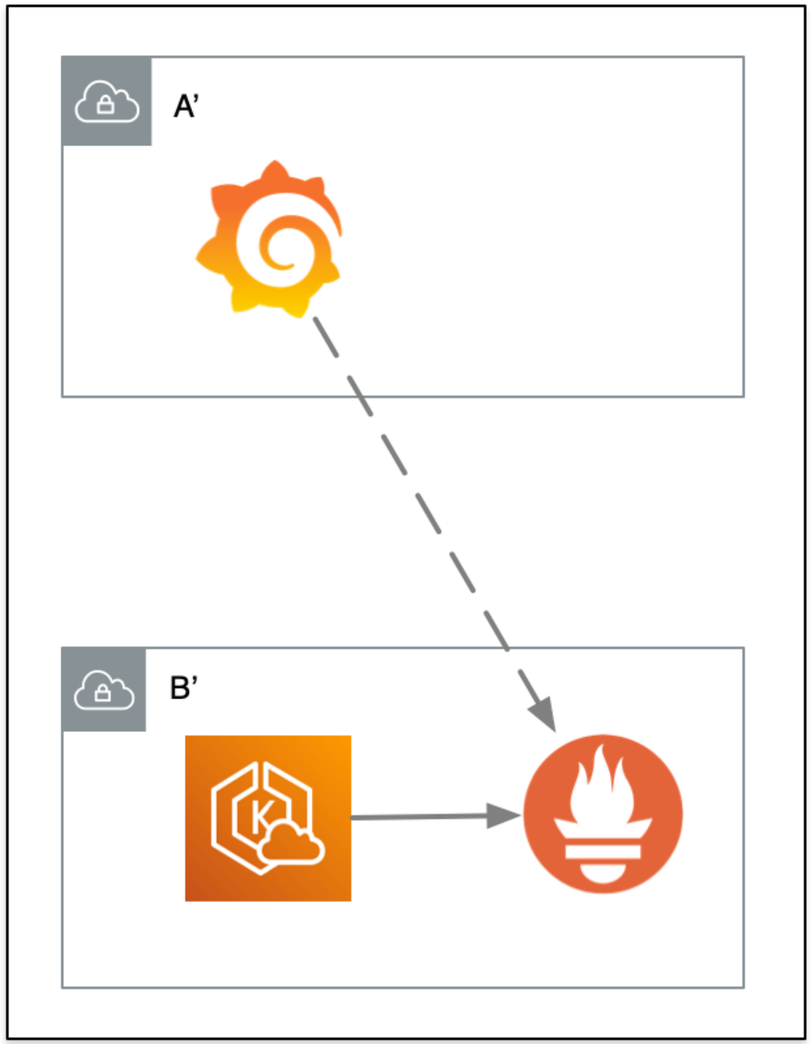
Design for data

Practice No.8 - Optimise Locality

Original



Changed



But check CAP theorem!

Haggle!

Practice No.9

- Engineering can lead:
  - Capacity plan like you were in a real data center!
  - Commit to usage: Reserve capacity (e.g. instances)
  - Use ephemeral resources: Storage & Spot
  - c.f. “Capacity Planning for Internet Services” - Cockcroft & Walker
- Things engineering can’t solve alone:
  - EDP aka “Cross-Service Discount”, Private Pricing Agreements
  - Dual vendor and announce your timeline to leave

Fin? Almost!

Cheesy self-promotion

Not really a practice

- Heartburn: You know you want it!
- Claim: “The tool for those who have no tools”
- The least I expect to come out of it is some discussion about sane defaults.
- https://codeberg.org/ofosos/heartburn

Thank you!