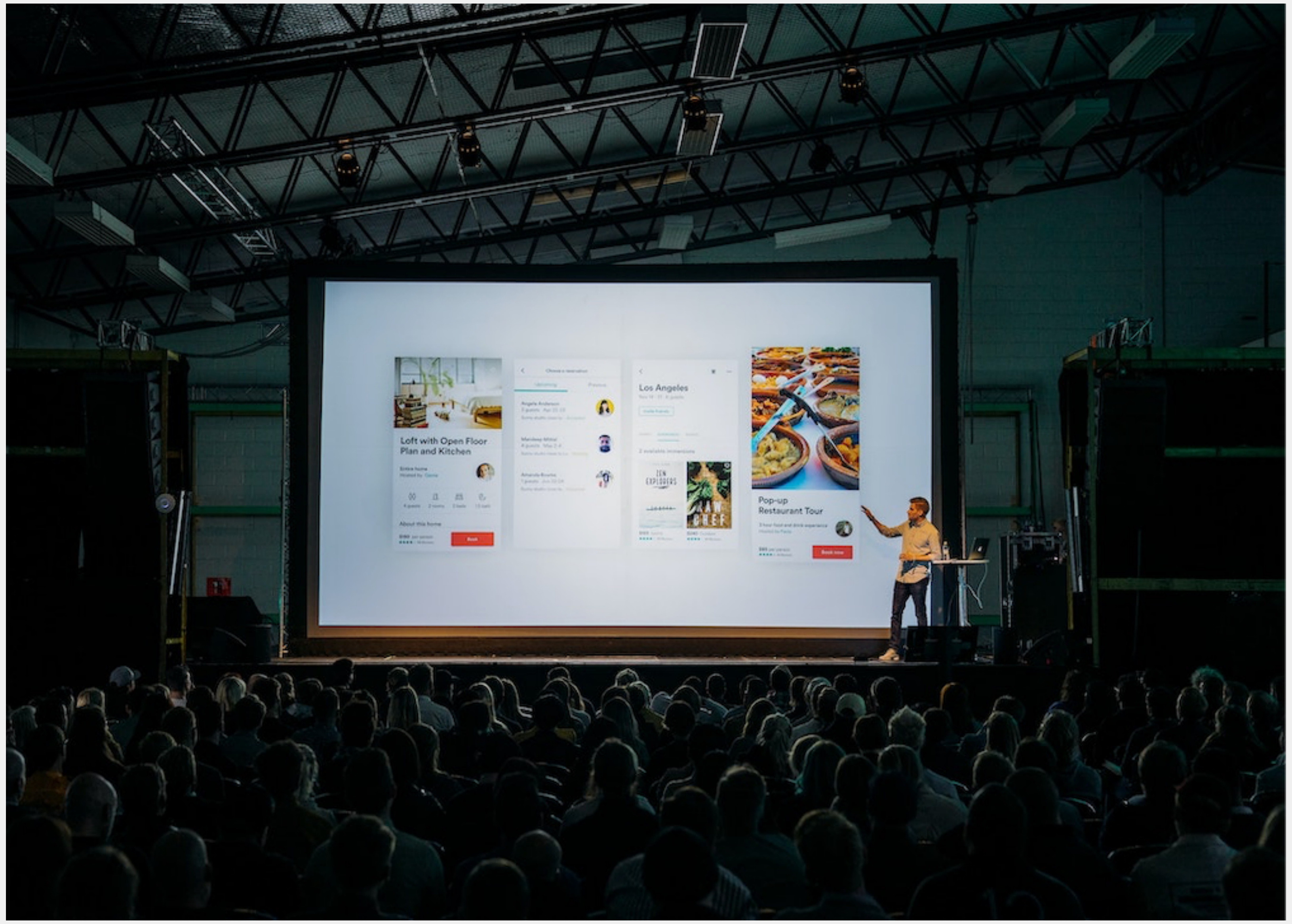
Central Wisconsin IT CONFERENCE

Tips and tricks to get started with microservices

Juan Peredo

https://www.linkedin.com/in/juanperedotech

One time at a conference



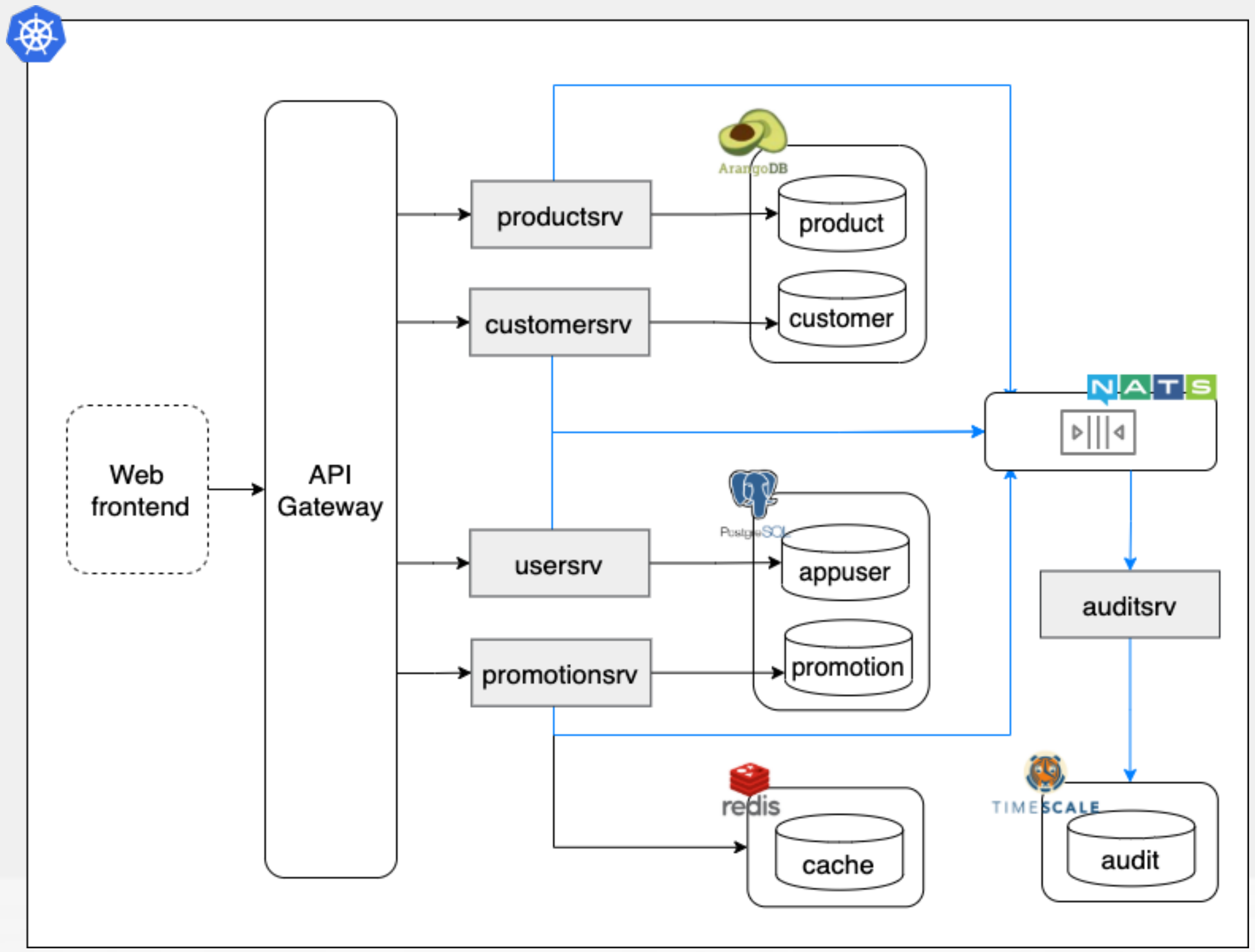
- I asked a speaker:

  Would you recommend microservices?

- His answer:

  Don't do it!

...So I decided to build some



Your guide on this journey

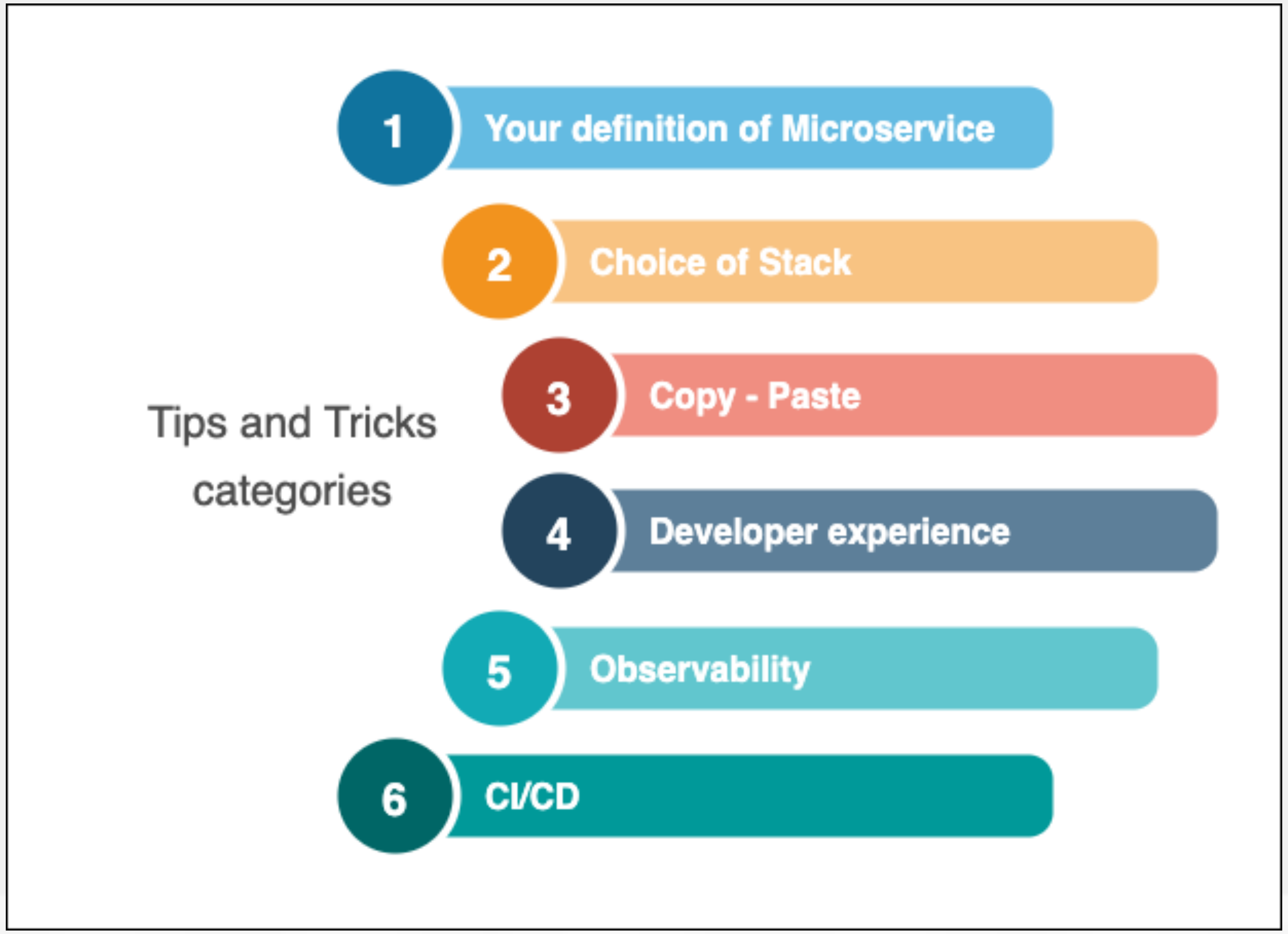
Cloud consultant / architect / developer and everything in between.

Linkedin: http://linkedin.com/in/juanperedotech

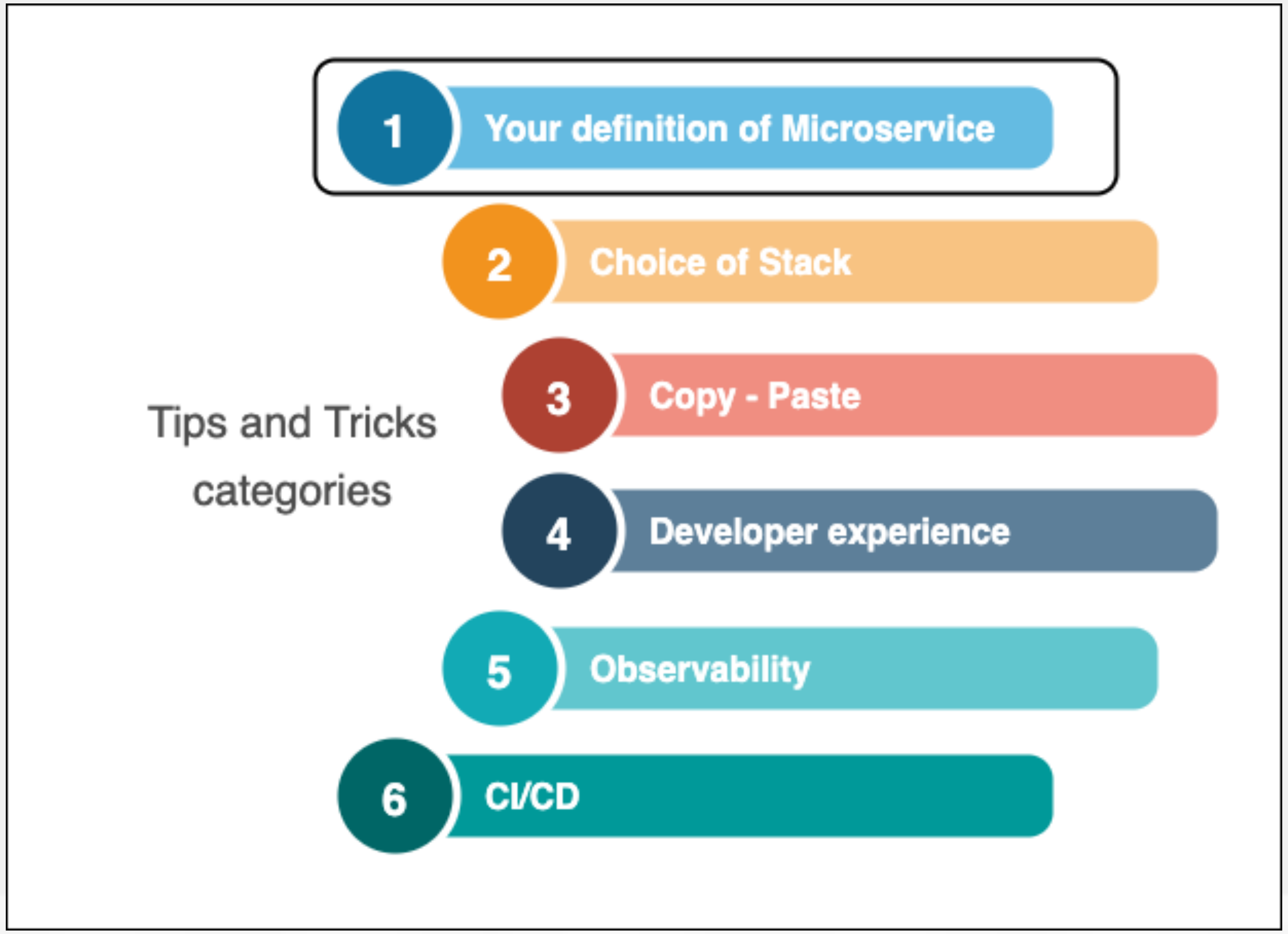
Twitter: @JuanPeredoTech

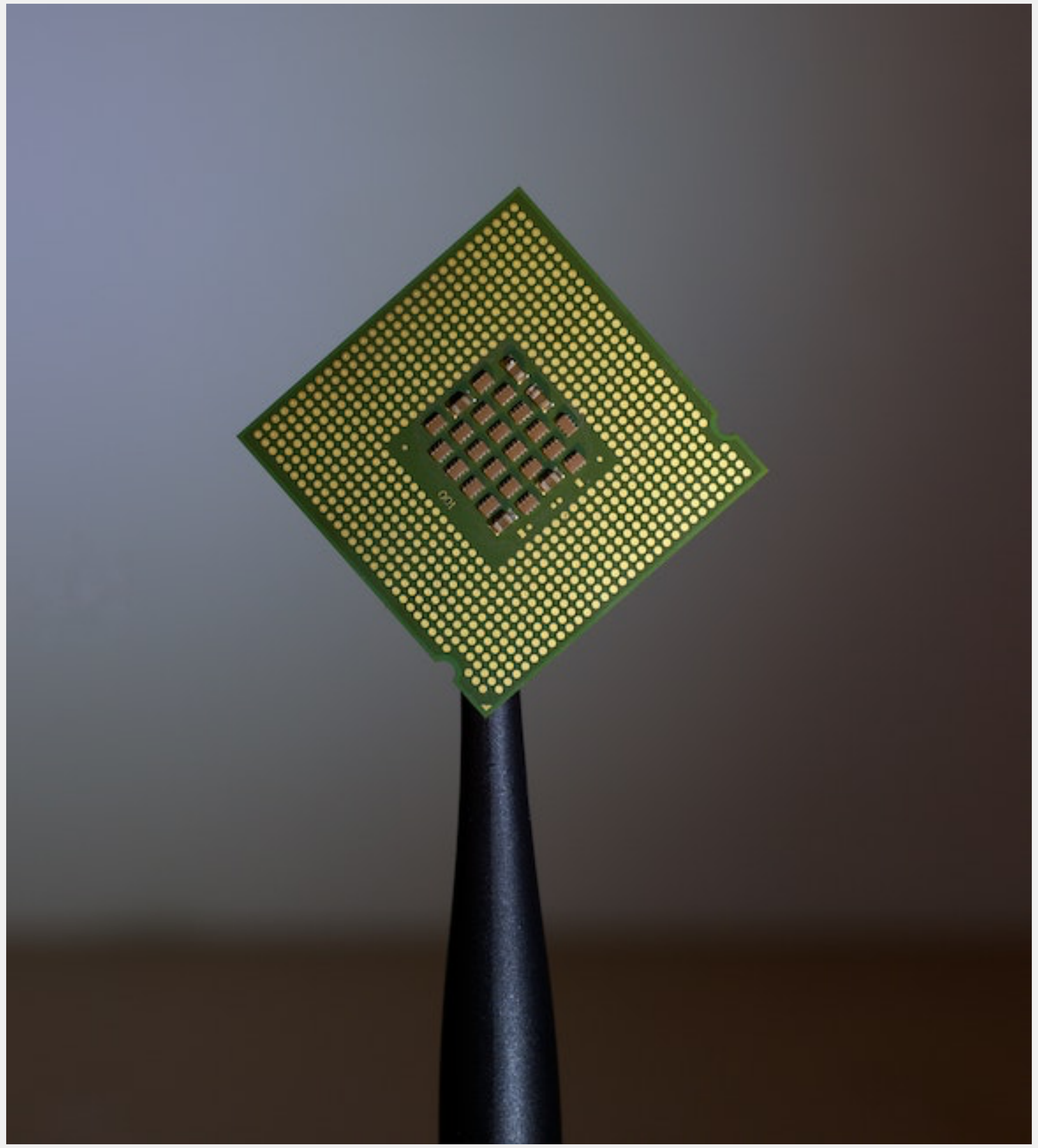
Github: https://github.com/camba1

Lessons learned



Lessons learned



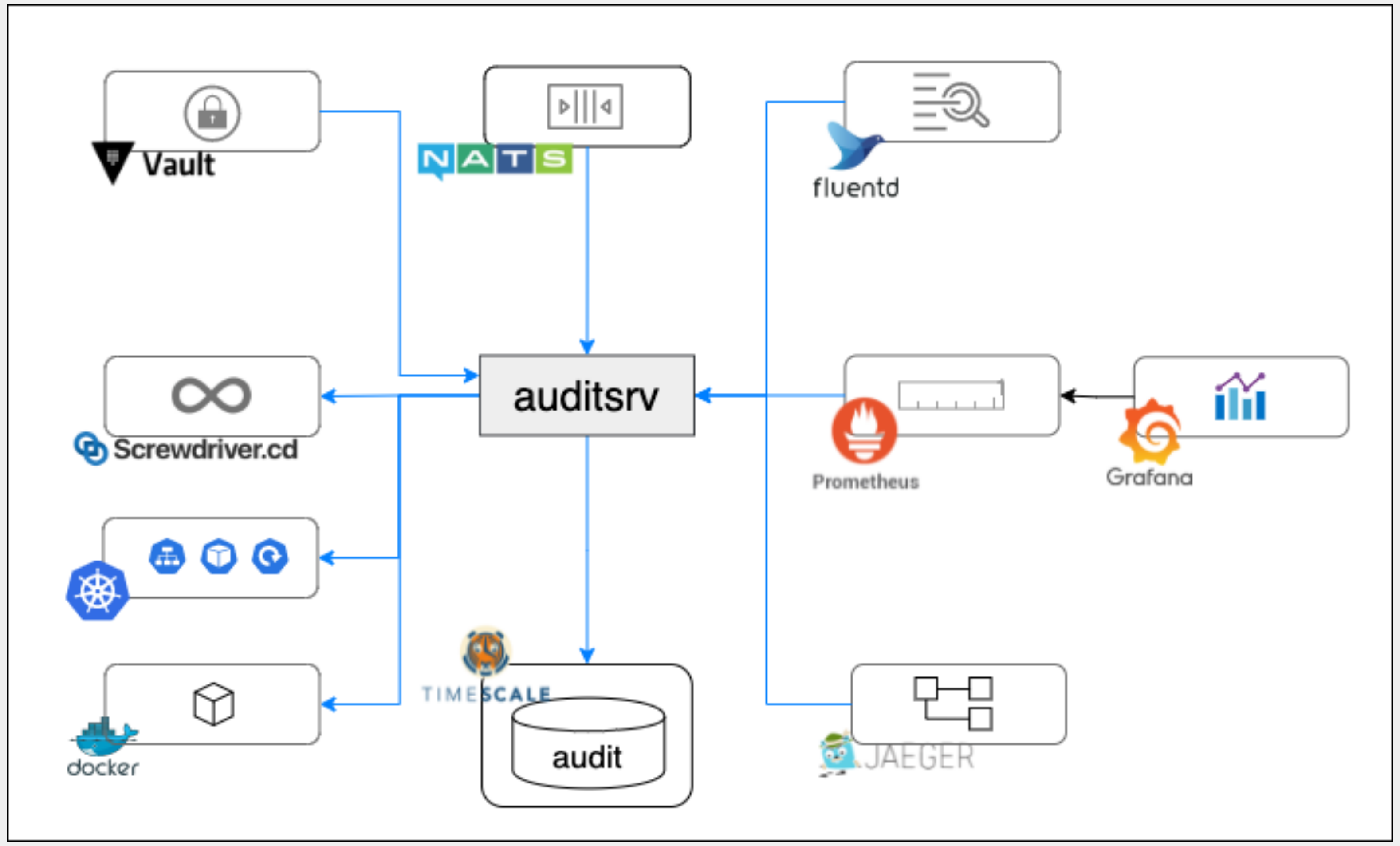


Small is nimble, but also complex

- Smaller services are easier to maintain
- But each service adds complexity to the system

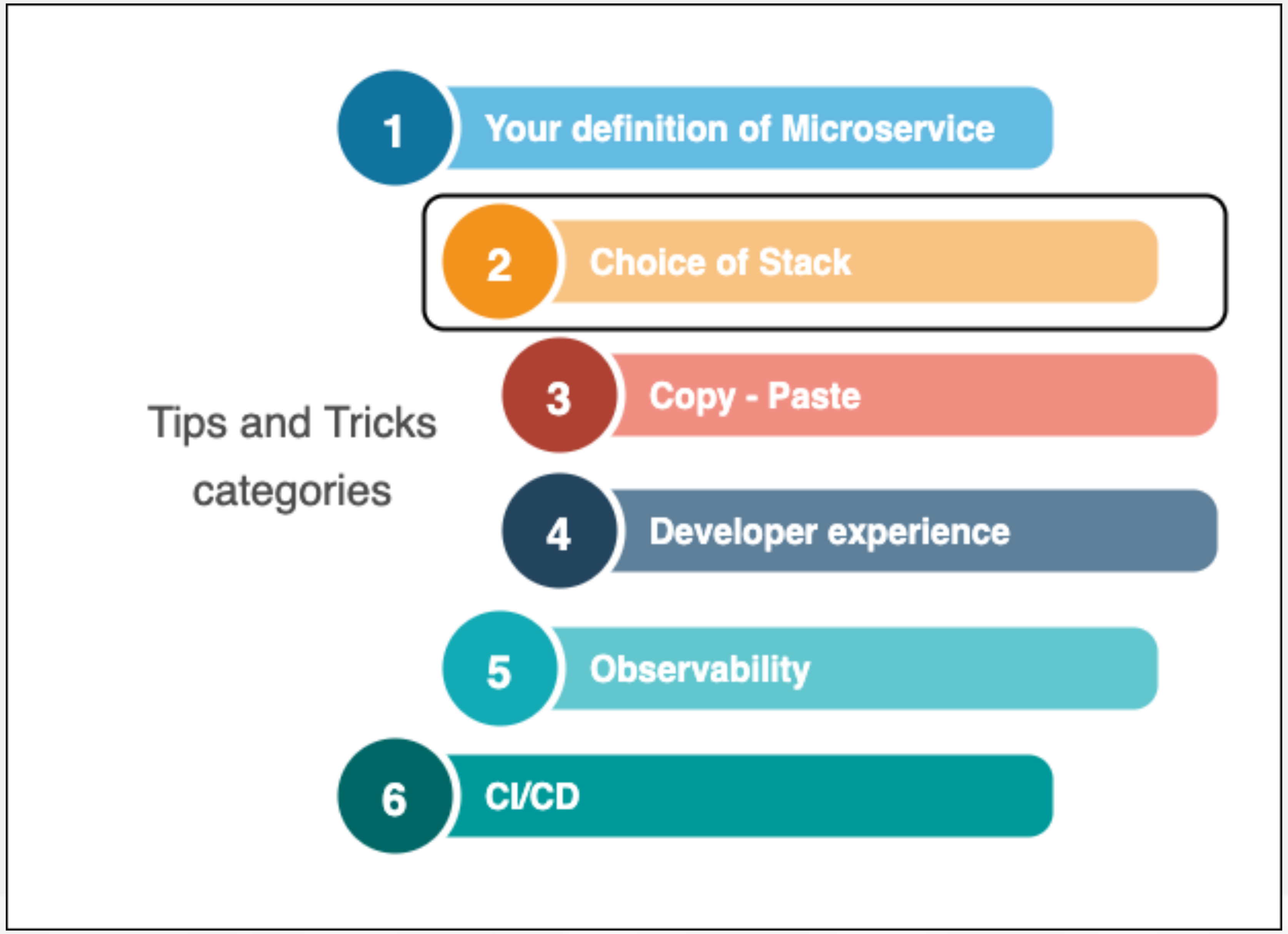
➡ Choose services size wisely

Audit Microservice Example



- This is a very small service
- However, it requires a whole ecosystem to be effective

Lessons learned

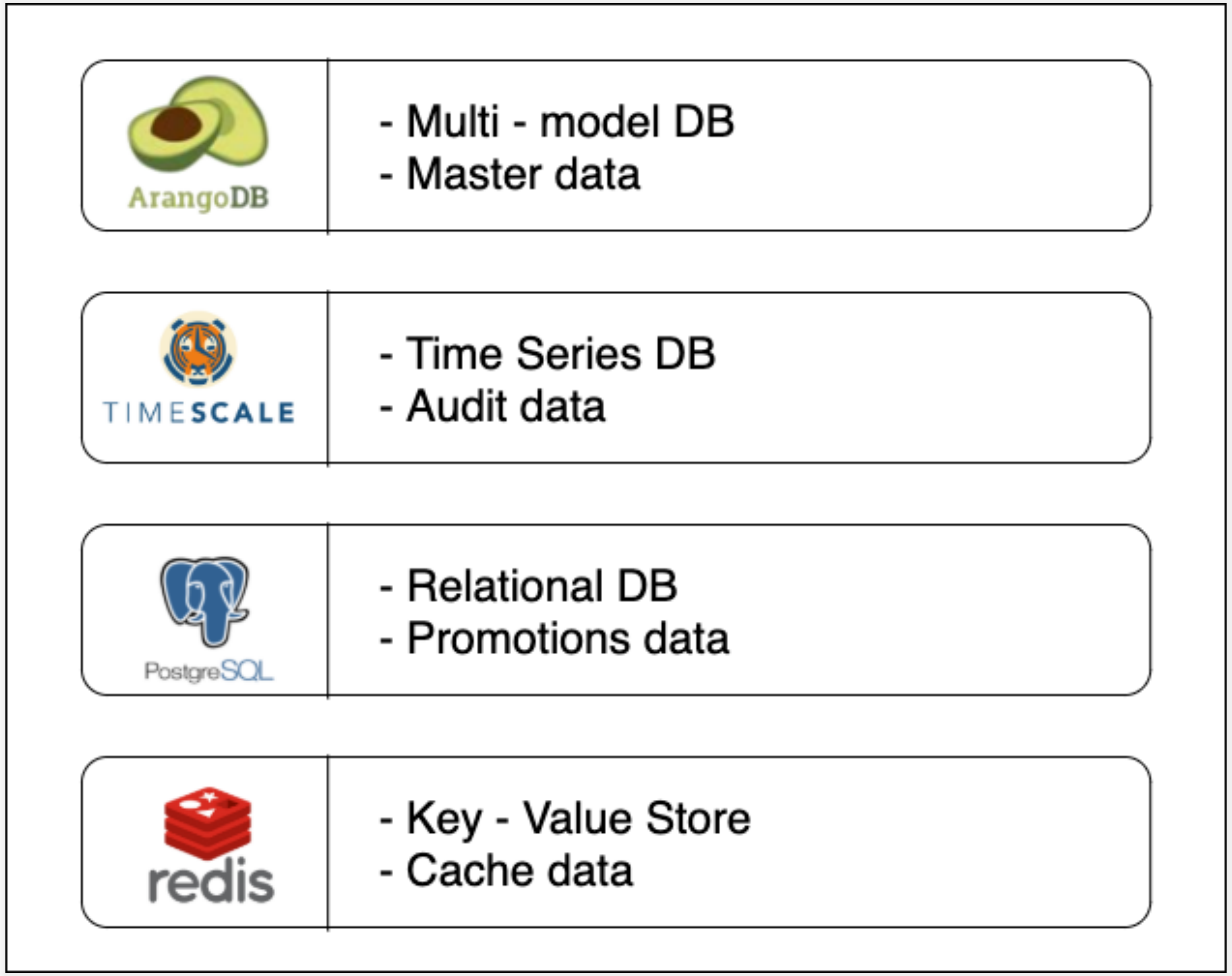


The polyglot promise

- Microservices enable use of multiple stacks
- However, requires:
  - Expertise on each stack
  - Future maintainability

➡ Use one stack unless absolutely necessary

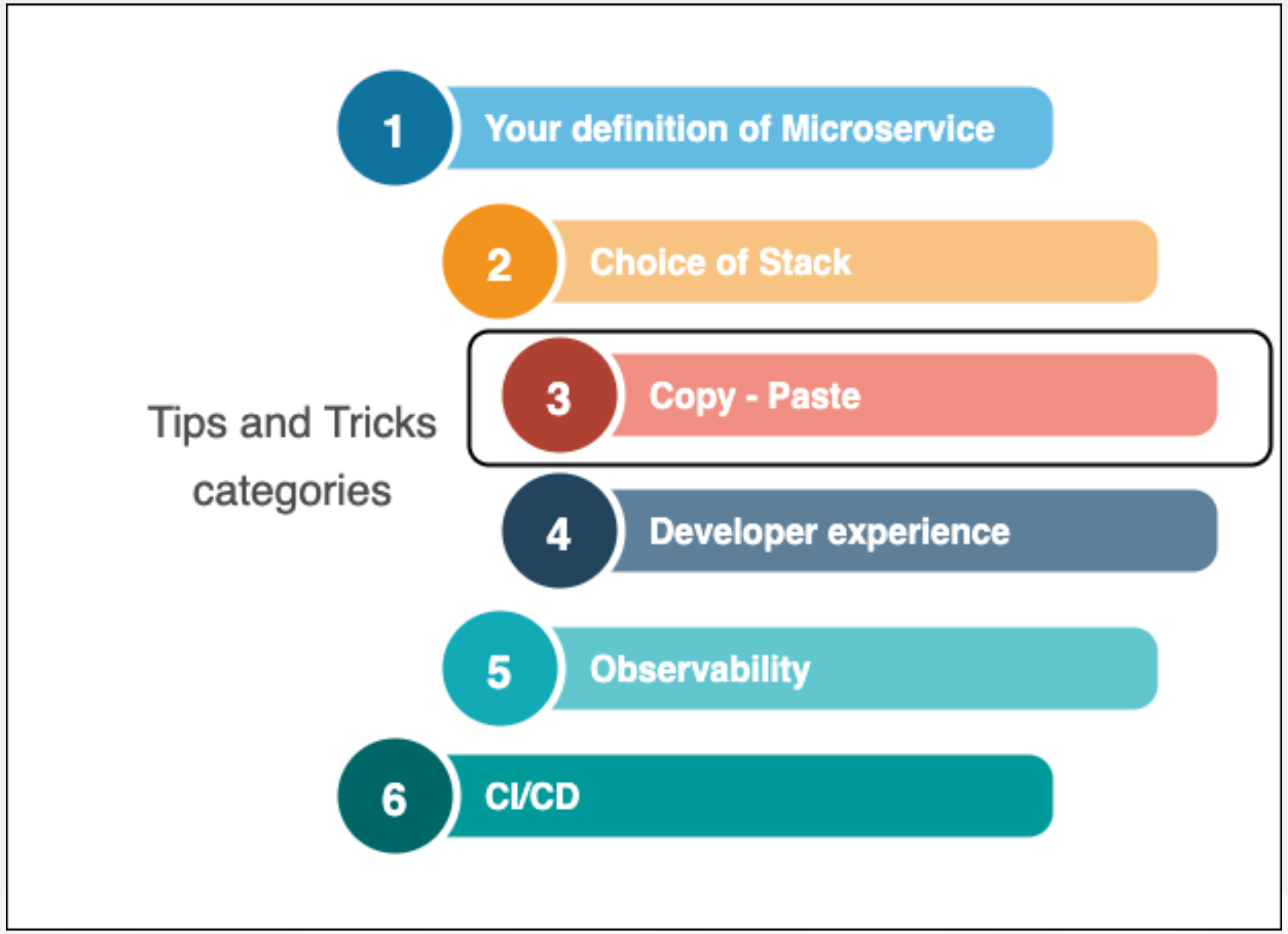
Our app uses multiple databases



ArangoDB	- Multi - model DB - Master data
TIMESCALE	- Time Series DB - Audit data
PostgreSQL	- Relational DB - Promotions data
redis	- Key - Value Store - Cache data

- Needed by the type of data to be stored
- Learned to use and maintain each one
- Glad rest of the stack is just Go and JavaScript

Lessons learned

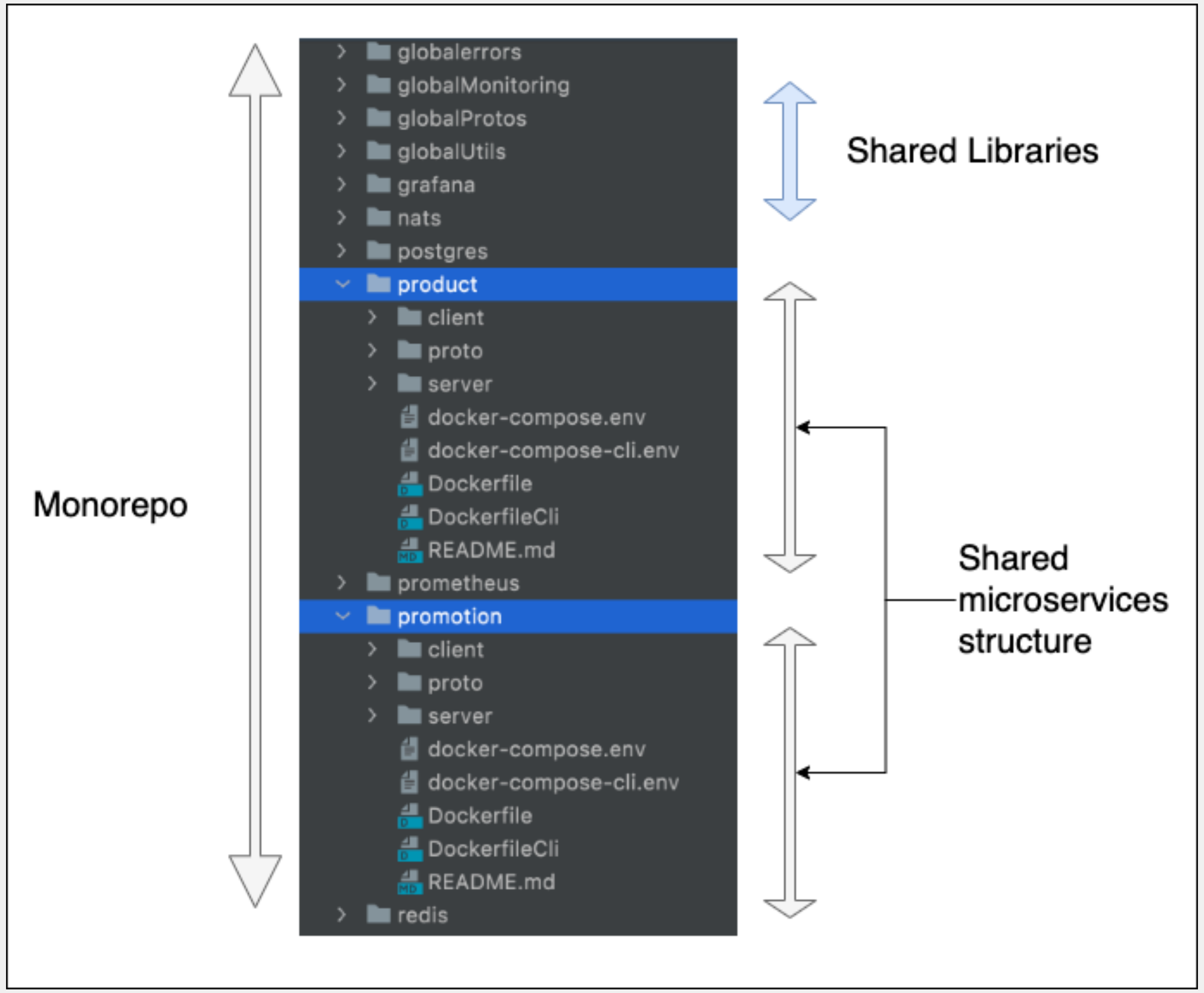


Copy - Paste Galore

- Theory : Microservices are independent
- Reality: A lot of the functionality is shared
  - A lot of copy-pasting
- Nightmare when making changes

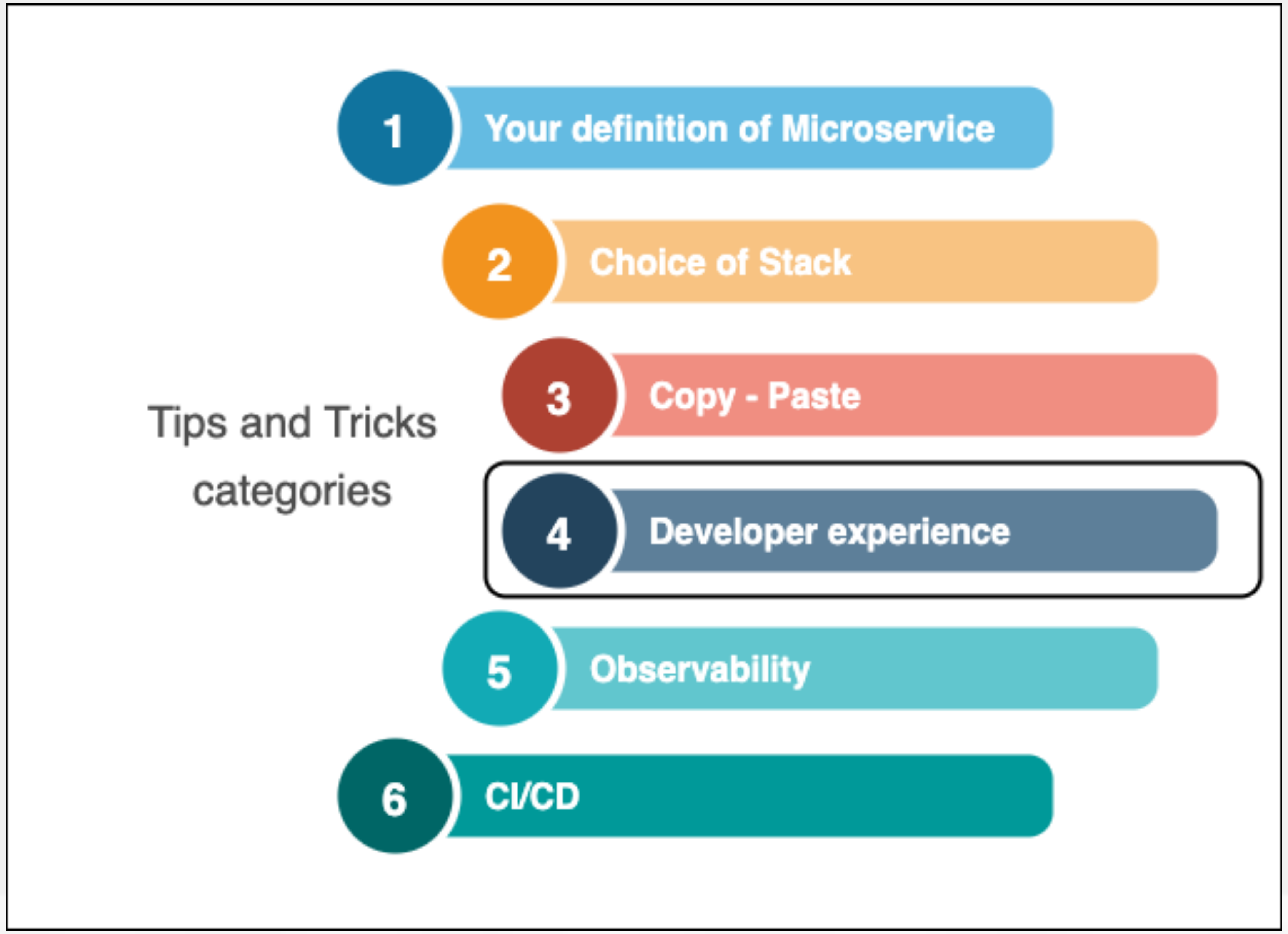
  ➡ Decide how & when to share code

Sharing code in our application



- Monorepo
- Shared libraries
- Shared structure

Lessons learned



Enable your team

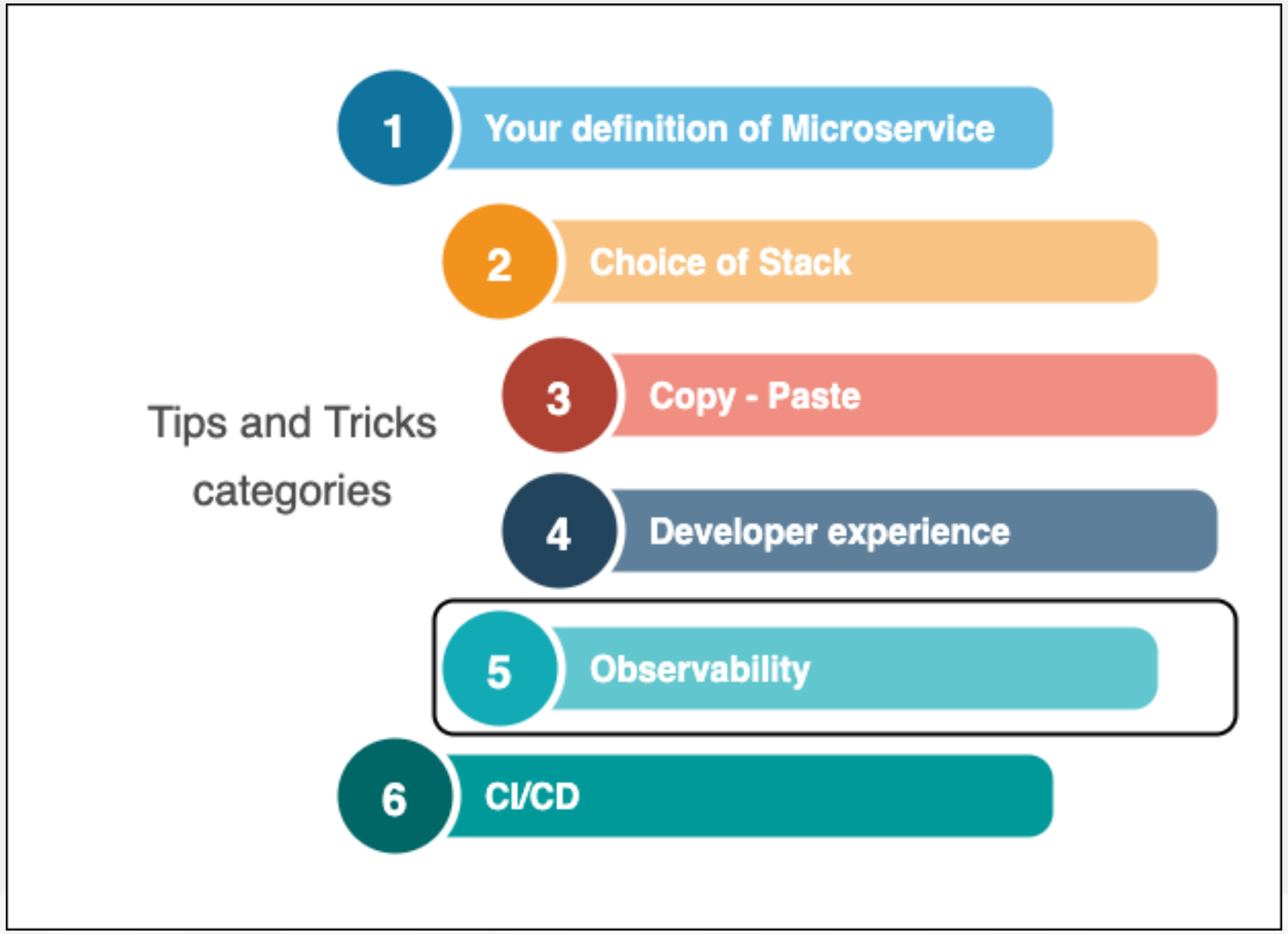
- Small services are easy to maintain
- Hundreds of them can destroy developer productivity

➡ Proactively ensure team has the right tools to be effective

Using Docker Compose

- Docker Compose allowed:
  - Bring up microservice & its dependencies
  - Hot reloading
  - Fast installs
  - Easy migration to K8s

Lessons learned

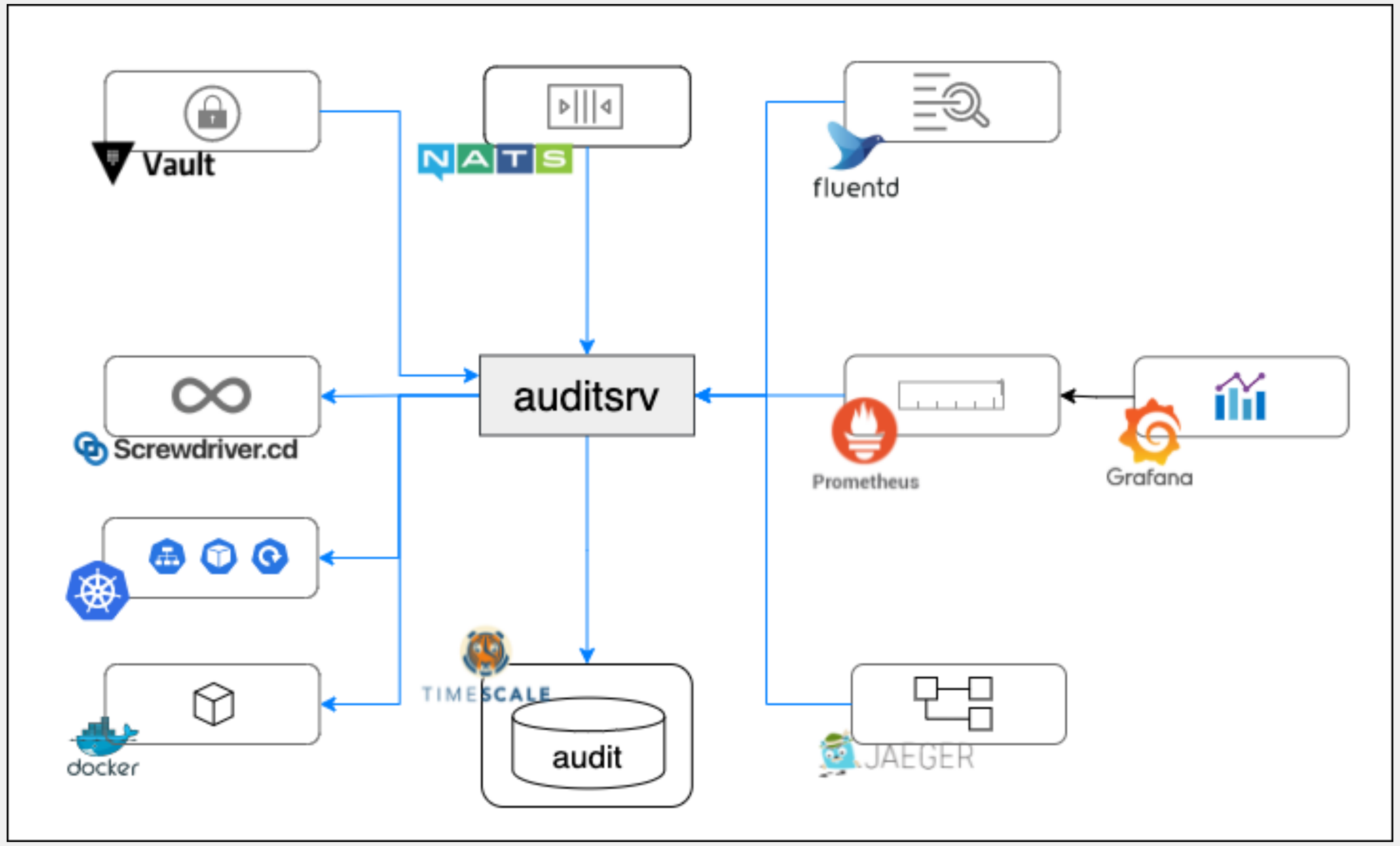


Observability is king

- Tracking issues in distributed systems is hard
- Measuring performance is complex
- Especially if transactions touch multiple services

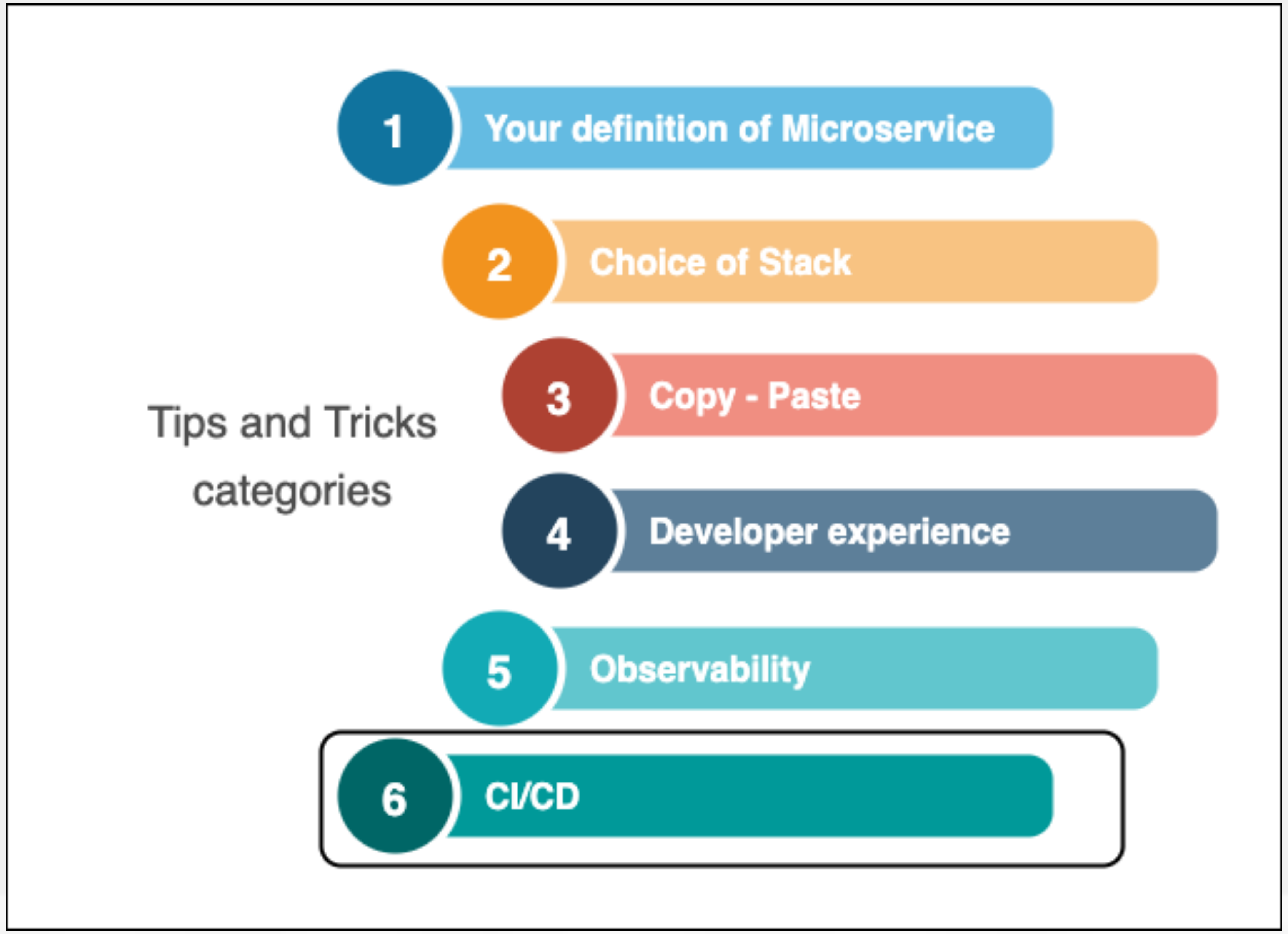
➡ Instrument observability in your services

Leveraging third party components

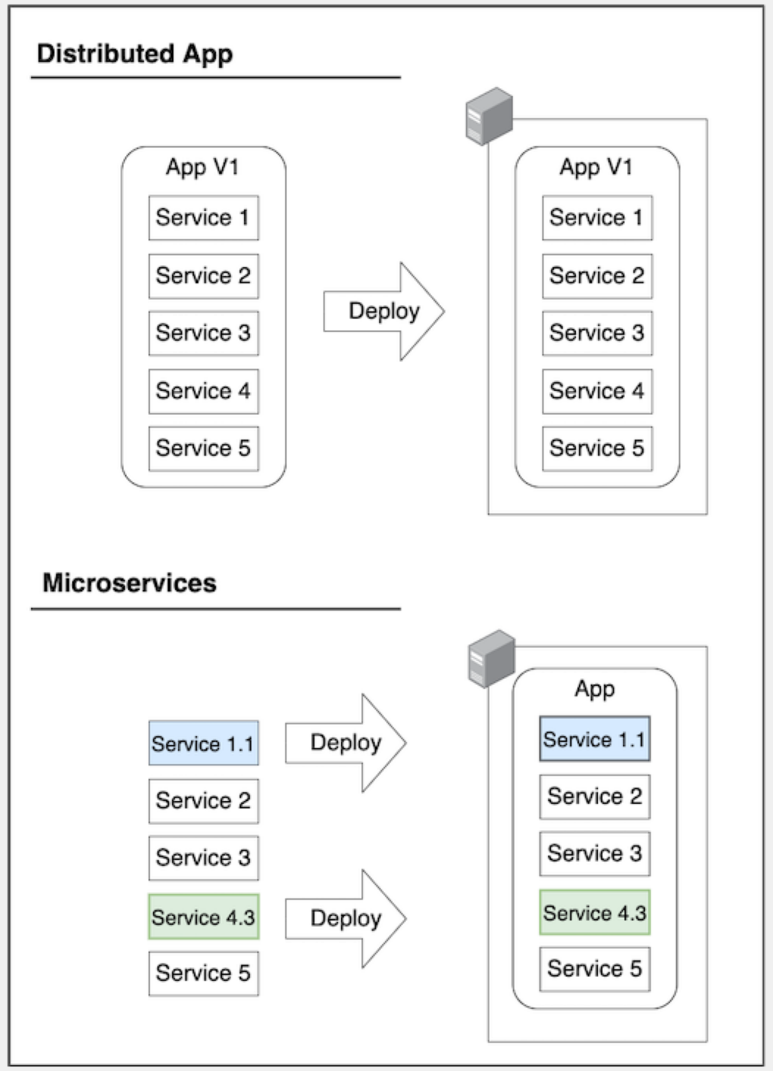


- Metrics: Prometheus
- Dashboards: Grafana
- Distributed tracing: Jaeger
- Log Processor: Fluentbit

Lessons learned



Automating deployments



- Manual deployments are not an option
- Distributed apps vs true microservices
- Even automation can get out of hand

  ➡ Automate, but keep it simple

Just KISS

- Deployments take just a couple of commands
- Choose the right tools

Questions?

Thanks to the CWIT Conference & all the sponsors

Appendix

Photos

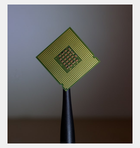
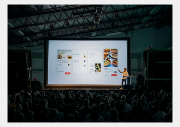
- Photo by Charles Deluvio on Unsplash
-  Photo by Brian Kostiuk on Unsplash
-  Photo by Teemu Paananen on Unsplash
-

Image by Gino Crescoli from Pixabay
- Photo by Hans-Peter Gauster on Unsplash

Photos

- Photo by Element5 Digital on Unsplash
- Photo by Leon on Unsplash