Microservices

Just another buzzword?

@dotjson

Whose using them?

"Microservice architecture is not a silver bullet… However, we have found that there are a huge number of benefits which vastly outweigh any disadvantages."

– Matt Heath, Technical Lead @ Hailo

Why are they using them?

Scale

Agility

Availability

Flexibility

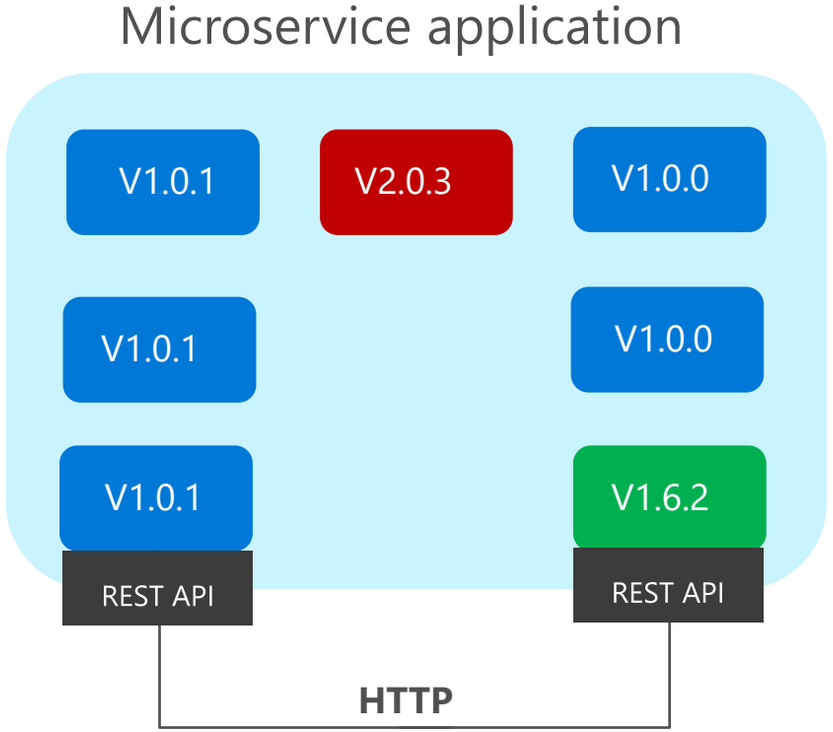
The Art of Scalability - Martin L. Abbott & Michael T. Fisher

What does “microservice” mean?

- **Distributed** architectural design style
- A microservice is a single component of a larger microservice system
- Encapsulation and modularization at the service level
- How small actually is micro?
- It is a refinement of SOA, DDD, and Component-based SE
- Complimented by cloud computing, DevOps and new workloads

Microservice Principles

A r
- i ed
- i able
- independently scalable



A microservice will:

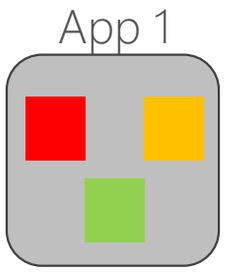
A microservice can:
- be developed by a small engineering team
- talk to other microservices via standard protocols
- contain both code and dedicated state
- use other microservices via standard interfaces
- encapsulate a business function
- update, fail and restart whilst the system is running
- support mixed languages, tools and platforms*

Monolithic application approach

: Presentation

: Application

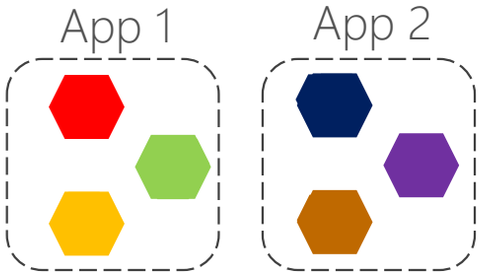
: Data



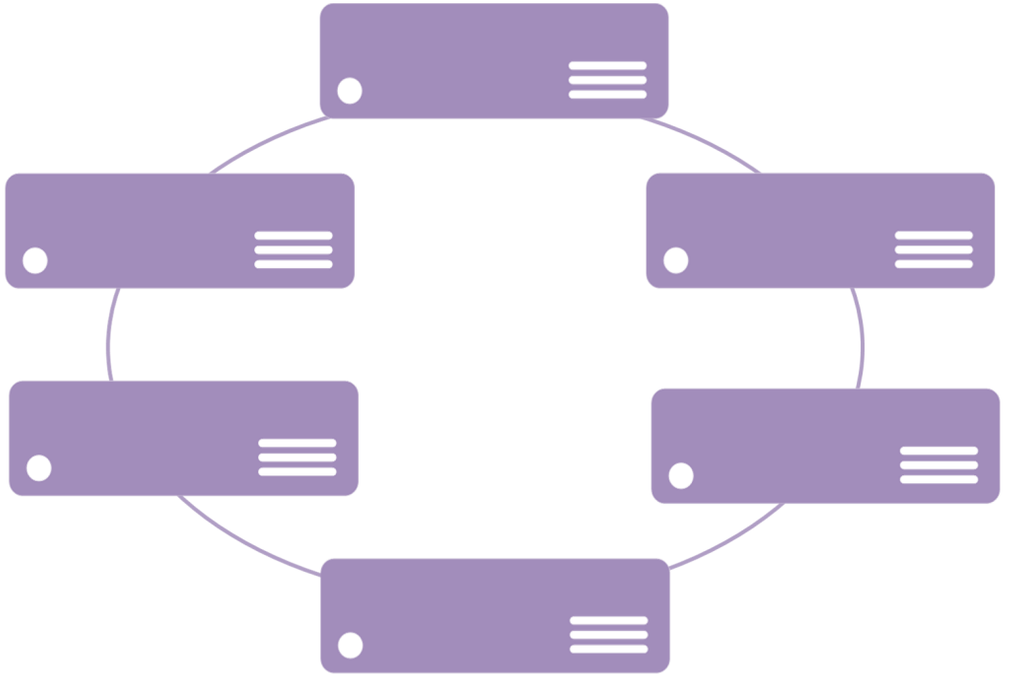
- Scales by cloning the app on multiple servers/VMs/Containers

Microservices application approach

- A microservice application separates functionality into separate smaller services.



- Scales out by deploying each service independently creating instances of these services across servers/VMs/containers

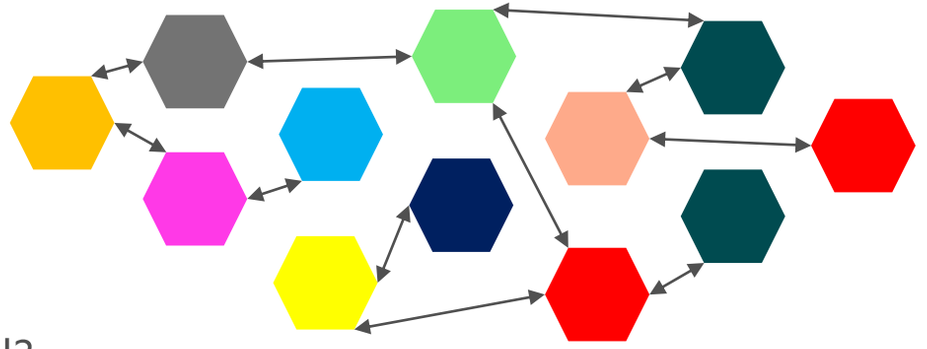


What's the catch?

What about network latency?

Who authorises who?

How do the microservices know about each other?



How do we monitor our system?

What language? Protocol? Interface? Should we use?

What happens if a service fail?

How do I perform complex queries?

CAP Theorem!?

"Turns out that selecting the runtime and language is just one step in building products in a microservices architecture. Another important aspect an organization has to think about is what *stack* to use for things like RPC, resilience, and concurrency."

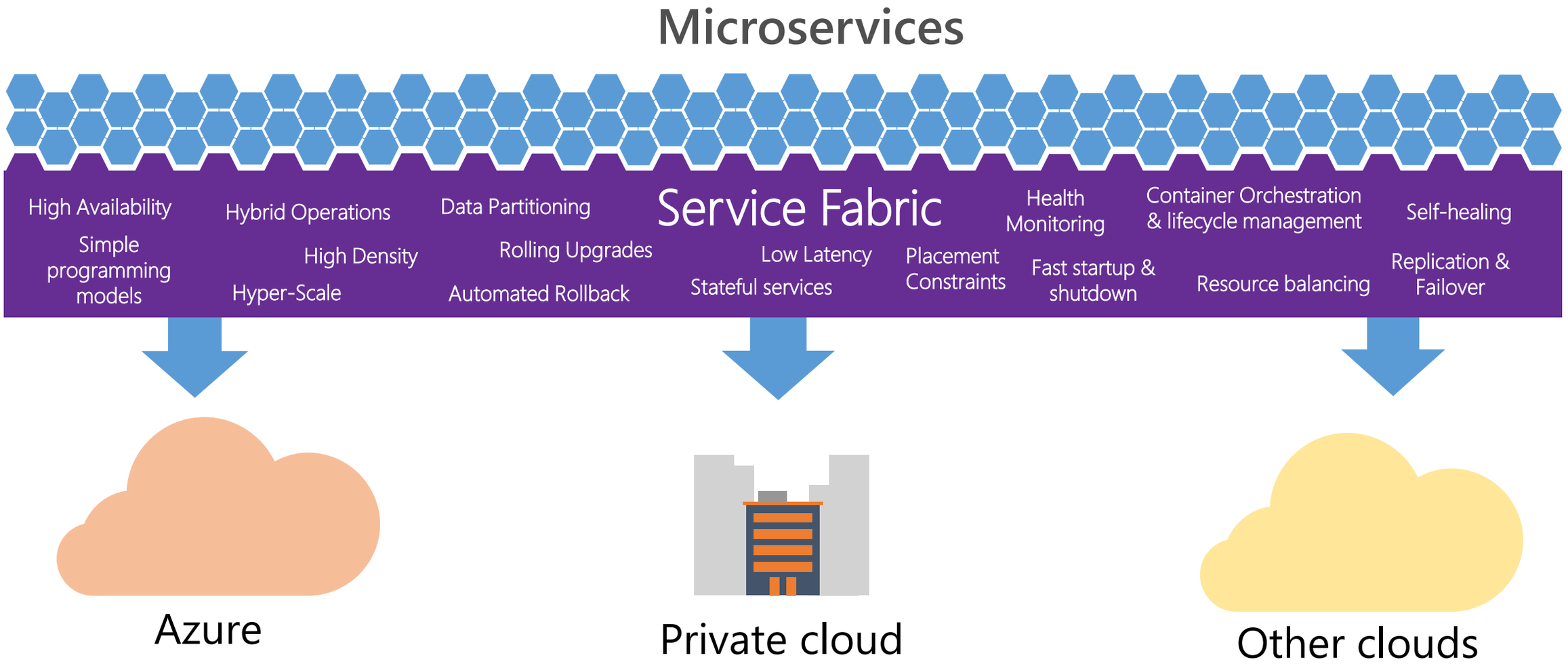
- Phil Calçado, Director @ DigitalOcean (formerly SoundCloud)

Service Fabric

Managing the complexity

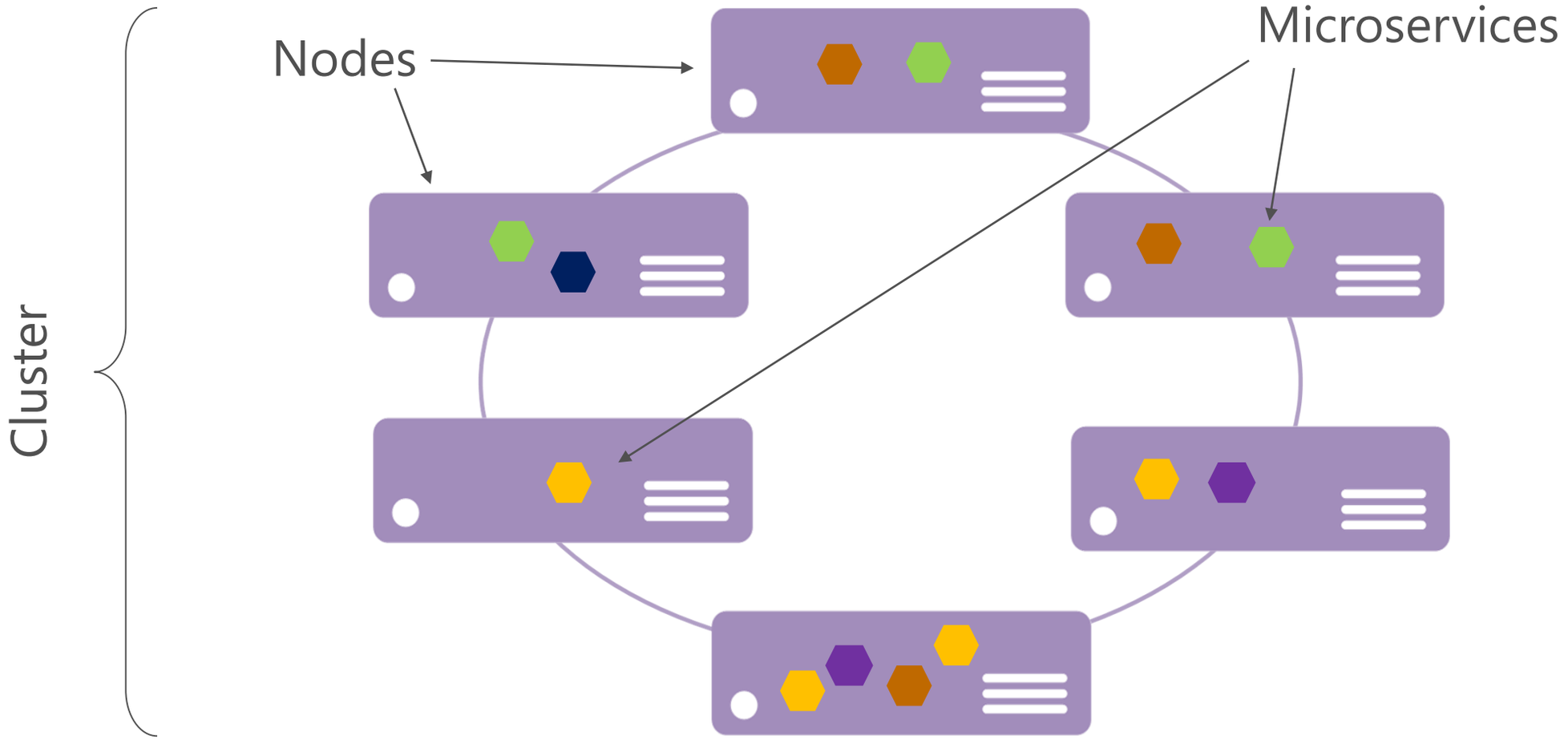
Microsoft Service Fabric

A platform for reliable, hyperscale, microservice-based applications



Service Fabric Cluster

The runtime environment your microservices run on top of



What happens if a microservice is too greedy? Or a node fails?

Demo

Programming Models

Reliable Actors

- Many small independent units of state (i.e. sensor, smart car, etc.)
- Single threaded objects
- Service Fabric manages concurrency and granularity of state
- Service Fabric to manages communication

Reliable Services

- You can use reliable collections (Dictionary and Queue) to store and manage state
- You control concurrency and granularity of state
- You manage communications between services

Antifragility

Summary

- Scalability
  Independently scalable services gives us higher utilization and reduces cost
- Agility
  Enables rapid delivery of new capabilities and shortens MTTR
- Availability
  Isolated errors allows service degradation and service replication enables fail over
- Flexibility
  Ability to use the right tool, languages and platform for the job
- Adaptability
  Update and deprecate microservices incrementally on the live system

Microsoft

@dotjson