
CLOUDFOUNDRY
SUMMIT
RUNNING AT SCALE
APRIL 18-20, 2018 | BOSTON

CLOUDFOUNDRY SUMMIT

The Java Ecosystem Collision: What is the future of Cloud Native?

Erin Schnabel

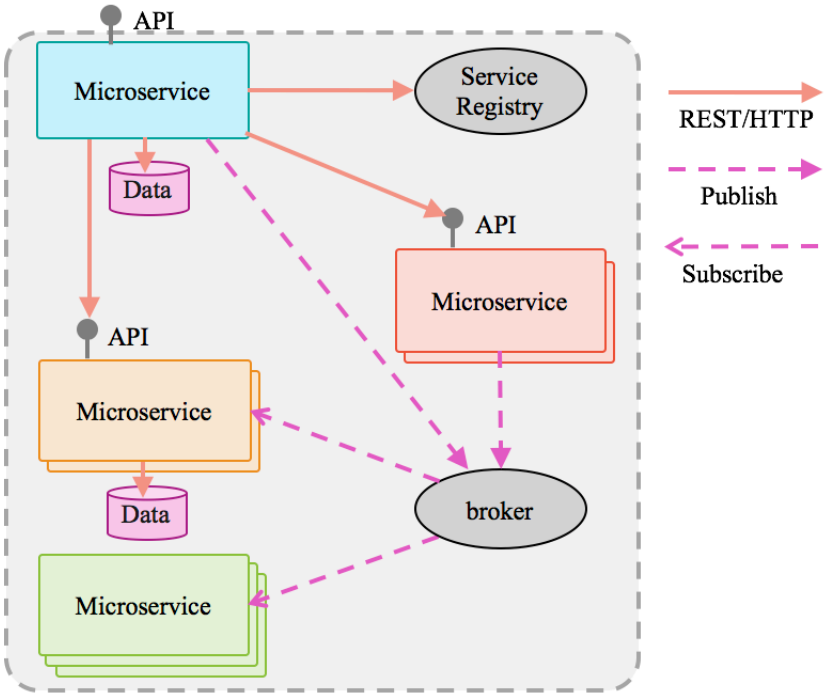
Senior Technical Staff Member, IBM

Cloud Native: A definition

- Microservice oriented
  - Loosely coupled
  - Declared external dependencies
- Container packaged
  - Resource isolation
  - Simplified operations
- Dynamically managed / orchestrated
  - Disposable instances → Elastic scaling

Microservices

- Encapsulated by API
  - Language-agnostic protocols
  - Replaceable
- Fault tolerant: No cascading failures
  - **Fail fast, gracefully**
  - Circuit breakers / bulkheads / timeouts
  - **Fallback**: retry vs. cached data
- Robust
  - **Expect rubbish**



Automation / Orchestration

- Provisioning/Deployment
  - Zero-downtime upgrades
- Load balancing / Scaling
- Health Management
  - **Cattle not Pets**
- Real-time monitoring
  - Logging & Metrics

Twelve Factor Applications

- Methodology for building SaaS applications

- Environment agnostic
  - any programming language
  - any backing services (or cloud provider..)

- http://12factor.net/

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing Services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

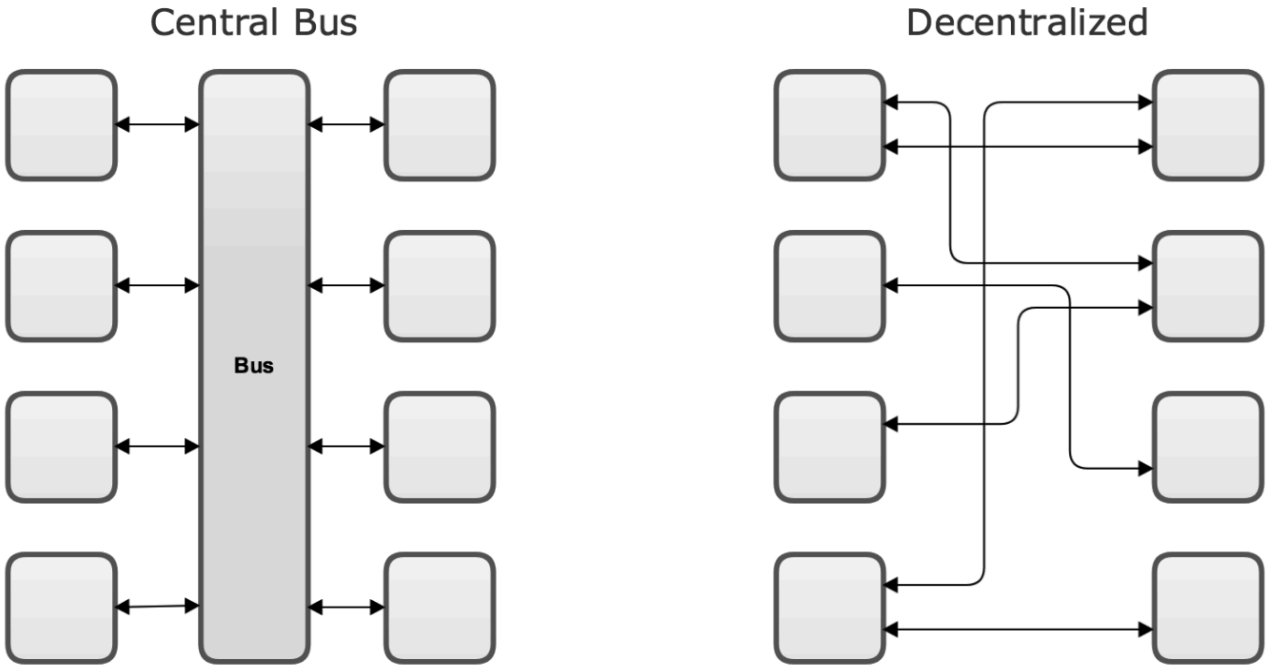
IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Smart Endpoints, Dumb pipes



https://medium.com/@nathankpeck/microservice-principles-smart-endpoints-and-dumb-pipes-5691d410700f

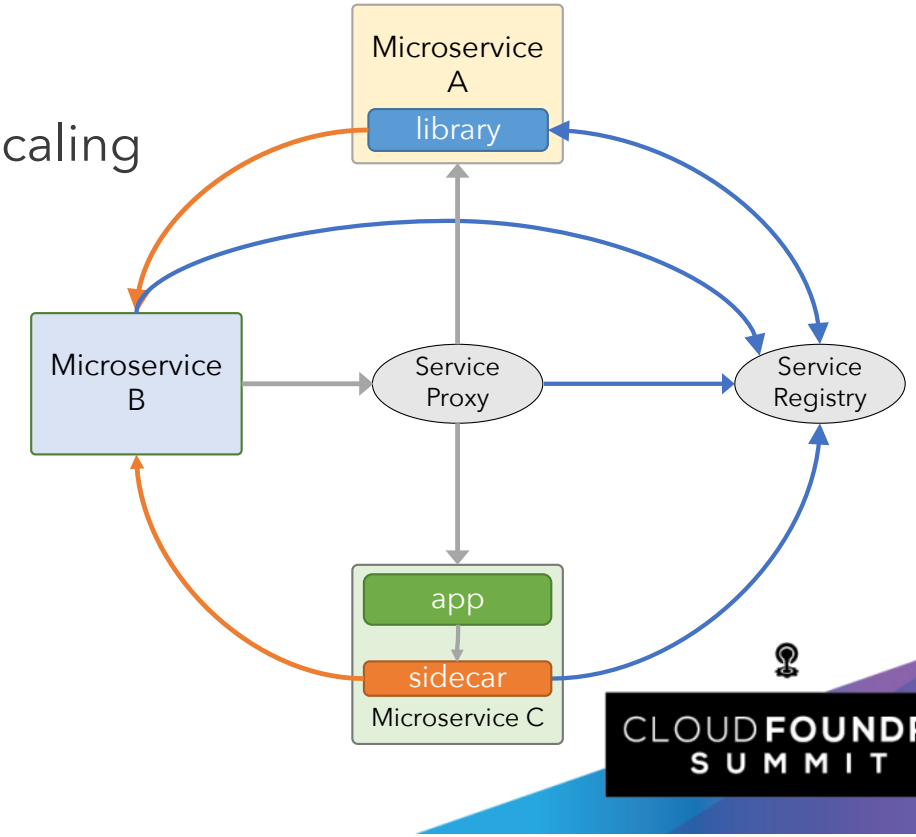
"Twitter microservices map looks just like the Netflix one. "We called this the 'Death Star' diagram"

– Adrian Cockcroft via twitter

https://twitter.com/adrianco/status/441883572618

Service Registration and Discovery

- Services need to find each other
- Direct / decentralized invocation
  - Required for **load balancing** and scaling
  - Environment changes constantly



- Client-side or server-side?
- Client library, sidecar, or proxy?

Fallacies of Distributed Computing

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero
- The network is homogenous

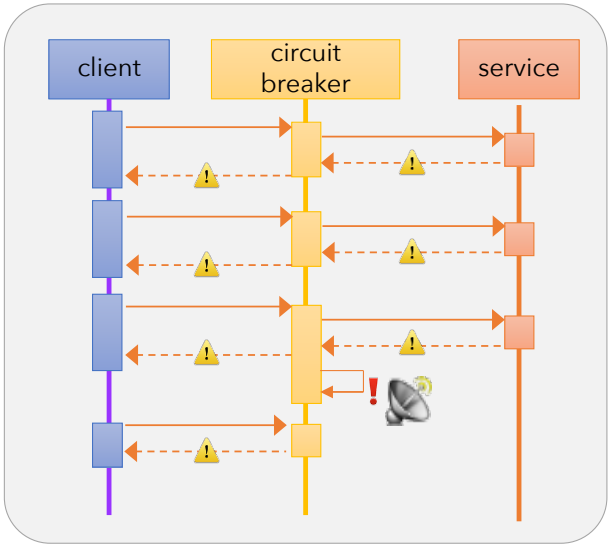
-- L Peter Deutsch, 1994

https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

Fault Tolerance

Circuit Breakers

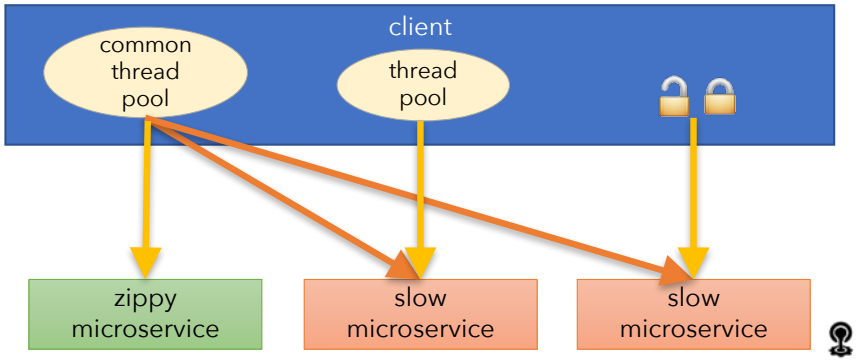
- Wrap remote calls, monitor for failures
- Notify when circuit is tripped
- Retry or Fallback?
- When is circuit reset?



Bulkheads

Ensure at most '*n*' threads waiting for a slow resource

- Thread isolation
  - Queue?
  - Timeout / fallback
- Semaphore isolation
  - Request sent if lock obtained



REST vs. Reactive

- REST/HTTP is inherently synchronous
  - Async libraries can mitigate
  - Does every message need an ack?

- Observer pattern:
  - One event publishes
  - 1..n subscribers/observers to act on events

- Events --> Reactive Streams

Synchronous calls considered harmful

Any time you have a number of synchronous calls between services you will encounter the multiplicative effect of downtime. Simply, this is when the downtime of your system becomes the product of the downtimes of the individual components. You face a choice, making your calls asynchronous or managing the downtime. At www.guardian.co.uk they have implemented a simple rule on the new platform - one synchronous call per user request while at Netflix, their platform API redesign has built asynchronicity into the API fabric.

https://martinfowler.com/articles/microservices.html

What is next for Cloud Native Java?

- Continuing with the new:
  - Serverless
  - Frameworkless

- But also returning to the old:
  - App Server & ESB vNext
    - Stateful interactions
    - Transactions