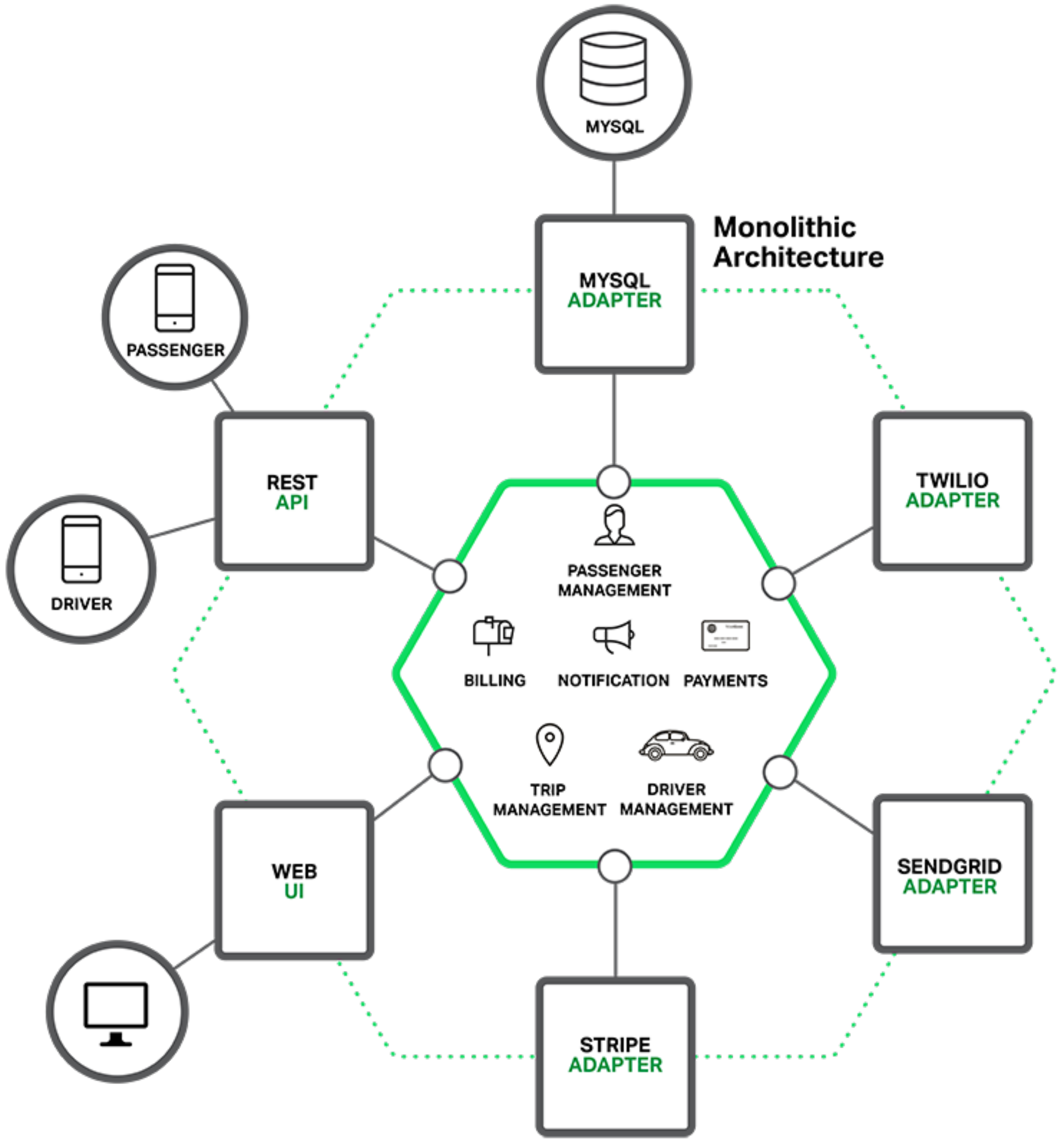
Microservices

Design of a monolithic application



For phones calls and send automated messages

For pub/sub services

For online payment

A Design of a Monolithic Application

- Despite having a logical modular architecture, the application is packaged and deployed as monolith (e.g. a WAR file on Tomcat or Glassfish server)

- Monolithic applications are easy to develop (inter process communication, shared objects, etc)

- Monolithic applications are easy to test: implement end-to-end testing by simply launching the application and testing the UI

- Monolithic applications are also simple to deploy: copy the packaged application to a server

- Scale the application by running multiple copies behind a load balancer

Problems with monolithic application

- Successful applications have a habit of growing over time and eventually becoming huge!!
- For a large and complex monolith application, your development organisation is probably in a world of pain:
    - Too large for any single developer to fully understand
    - Fixing bugs and implementing new features correctly becomes difficult and time consuming
    - The larger the application, the longer the start-up time is after making any changes
- Monolithic applications can also be difficult to scale when different modules have conflicting resource requirements (which hardware can be beneficial?)
- Reliability: a bug in any module, such as a memory leak, can potentially bring down the entire process
- Monolithic applications make it extremely difficult to adopt new frameworks, languages and technologies.

Challenges with monolithic software

Difficult to scale

Architecture is hard to maintain and evolve

Lack of agility

Long Build/Test/Release Cycles (who broke the build?)

New releases take months

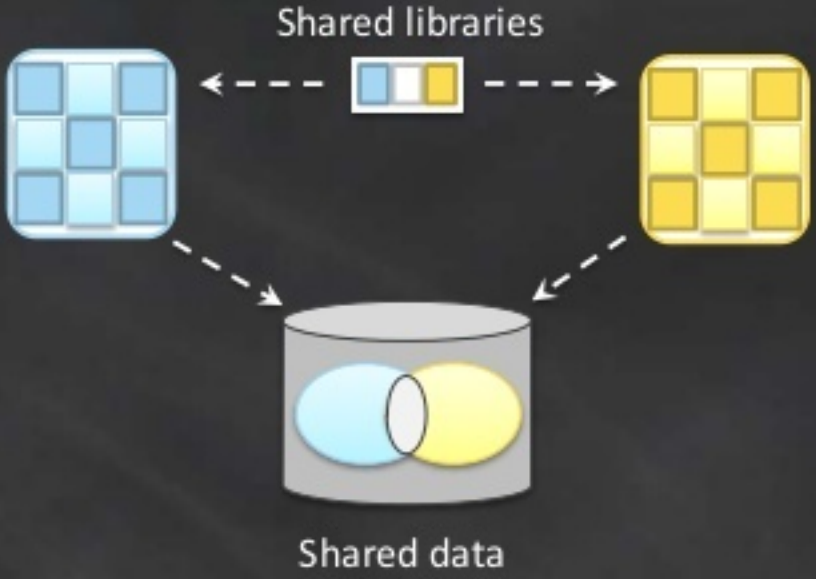
Lack of innovation

Operations is a nightmare (module X is failing, who's the owner?)

Long time to add new features

Frustrated customers

Too much software coupling

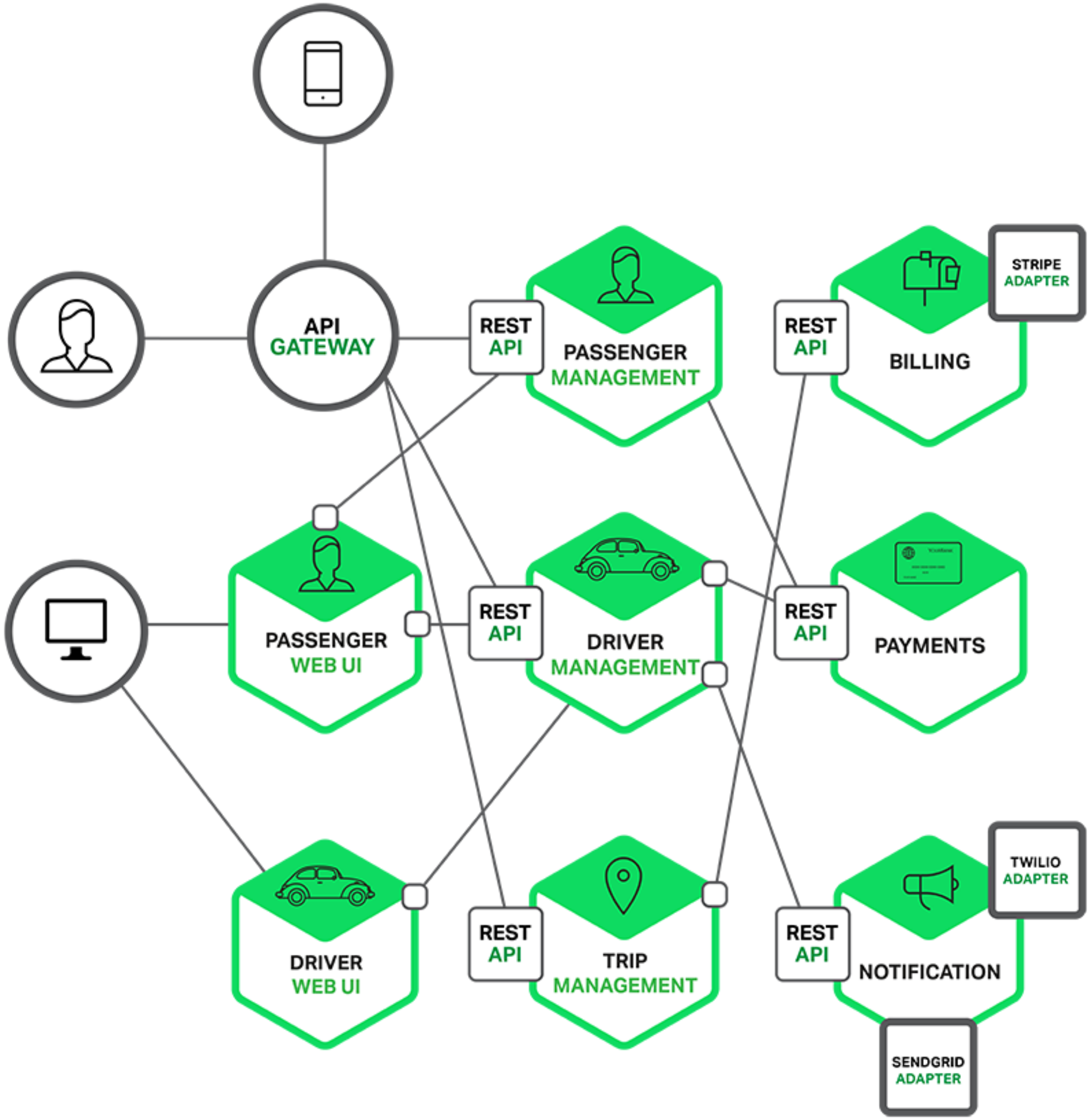


Microservices Architecture pattern

“Instead of building a single monstrous monolithic application, the idea is to split your application into set of smaller, interconnected services.”

- Adopted by Amazon, ebay, Netflix, and many other startups
- A service typically implements a set of distinct features or functionality, such as order management, customer management, etc

Microservices Architecture pattern



"Each microservice is a mini-application that has its own hexagonal architecture consisting of business logic along with various adapters"

"**service-oriented architecture** composed of **loosely coupled elements** that have **bounded contexts**"

Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)

“service-oriented architecture composed of loosely coupled elements that have bounded contexts”

Services communicate with each other over the network

Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)

“**service-oriented architecture** composed of **loosely coupled elements** that have **bounded contexts**”

You can update the services independently; updating one service doesn’t require changing any other services.

Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)

"service-oriented architecture composed of loosely coupled elements that have bounded contexts"

Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)

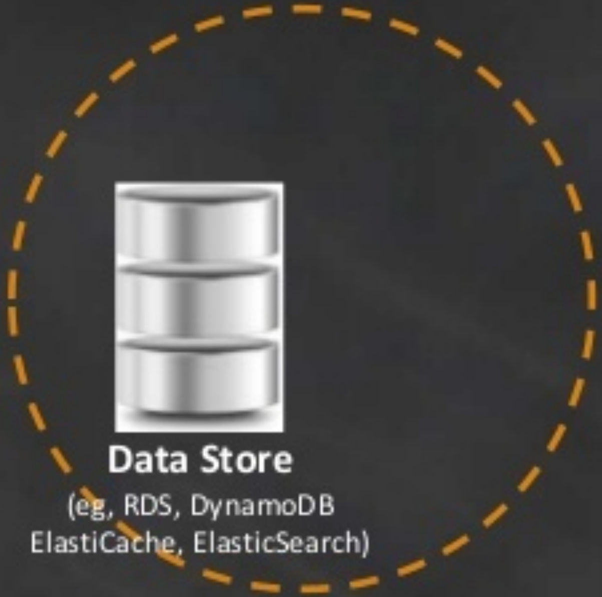
Self-contained; you can update the code without knowing anything about the internals of other microservices

“Do one thing, and do it well”

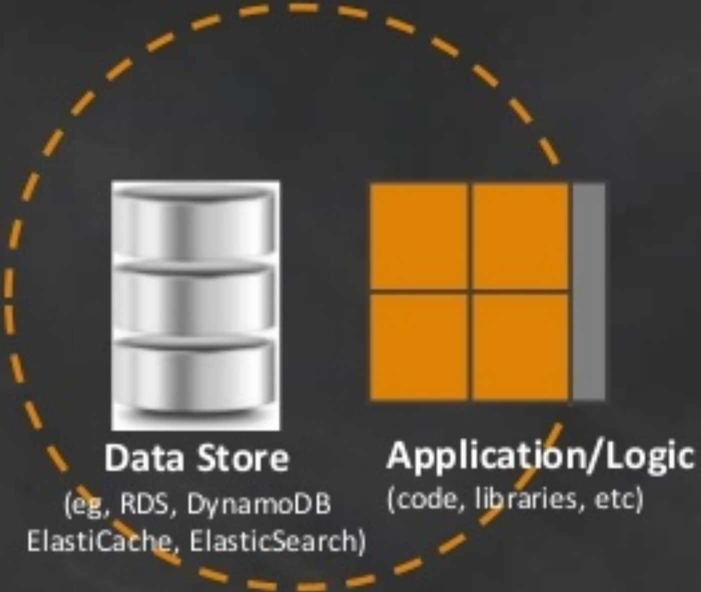
“Do one thing, and do it well”

Anatomy of a Micro-service

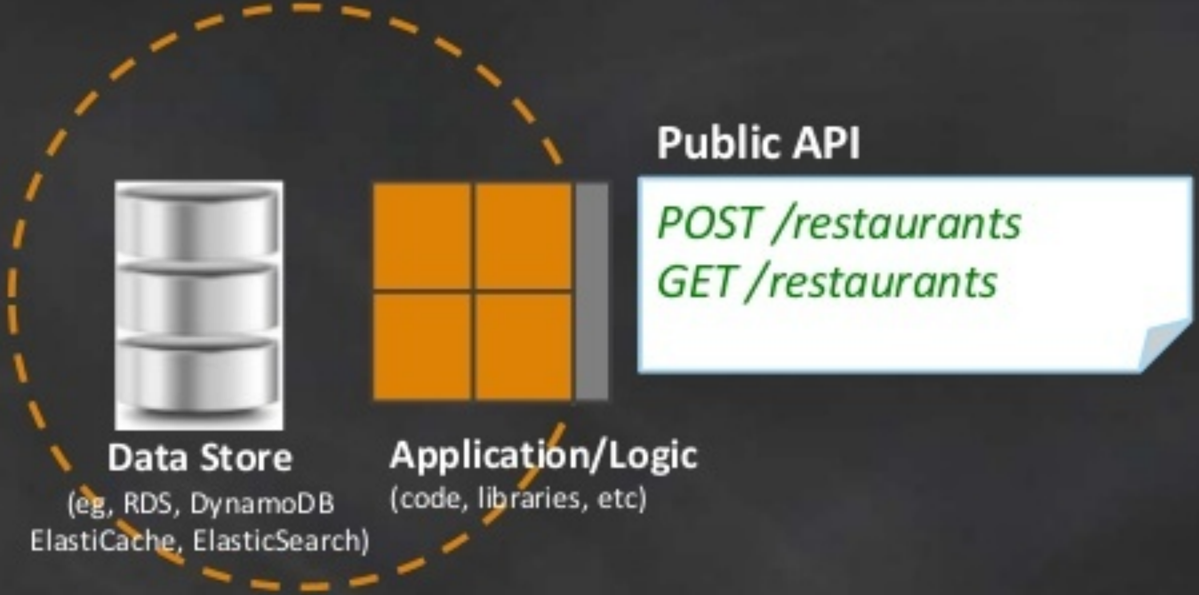
Anatomy of a Micro-service



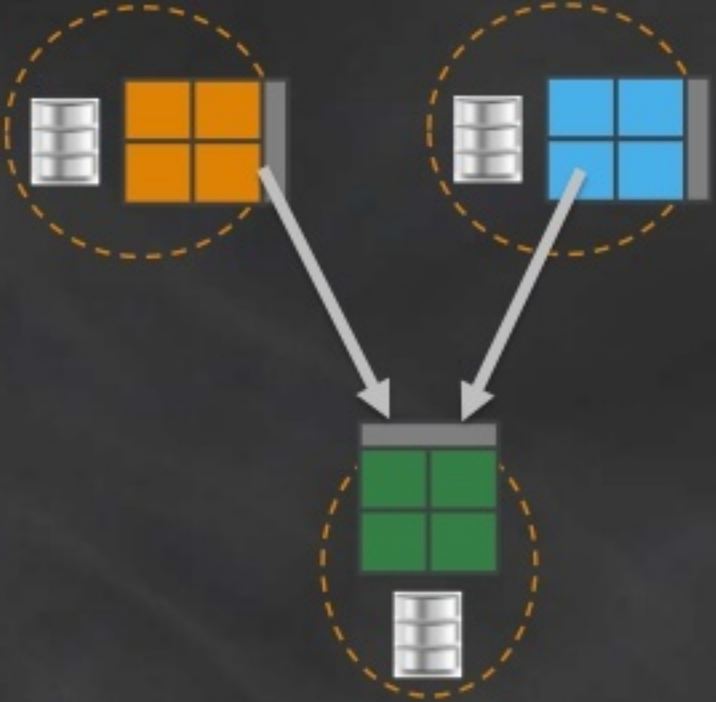
Anatomy of a Micro-service



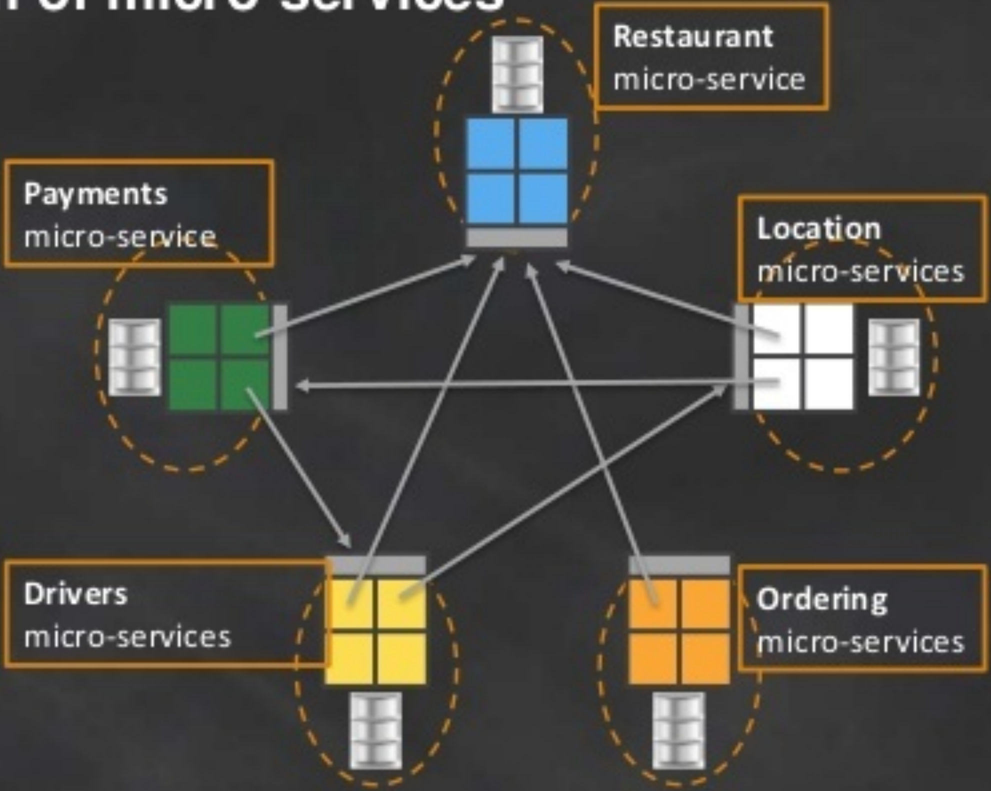
Anatomy of a Micro-service



Avoid Software Coupling



Ecosystem of micro-services

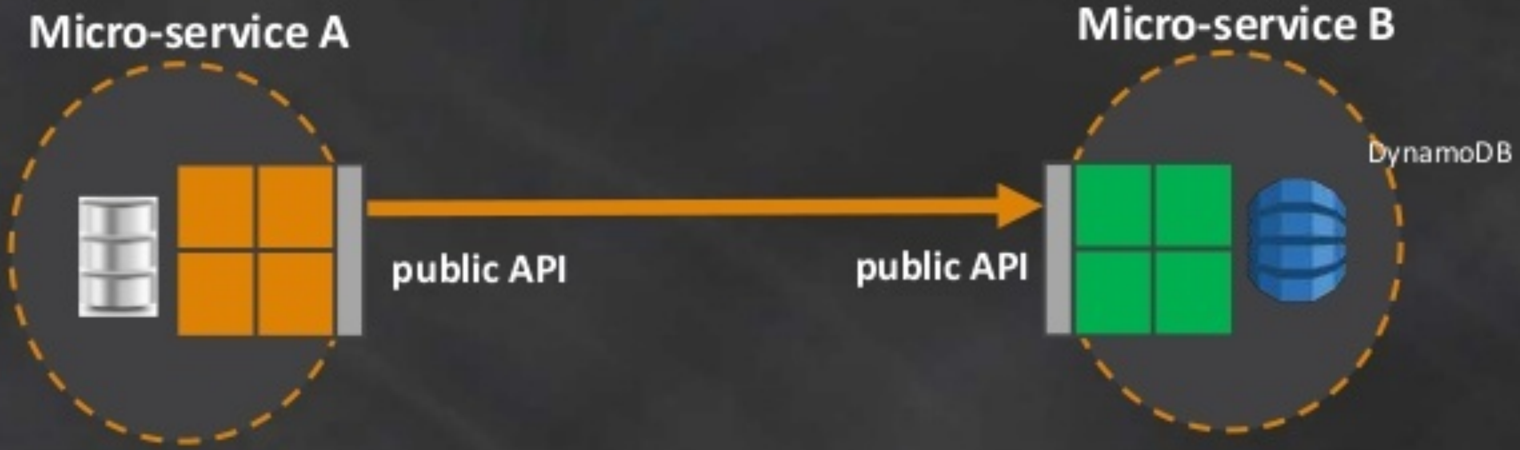


TENANCY AGREEMENT – ENGLA
RENTAL AGREEMENT
(For an Unfurnished House or Flat on an Assured
Terms and Conditions

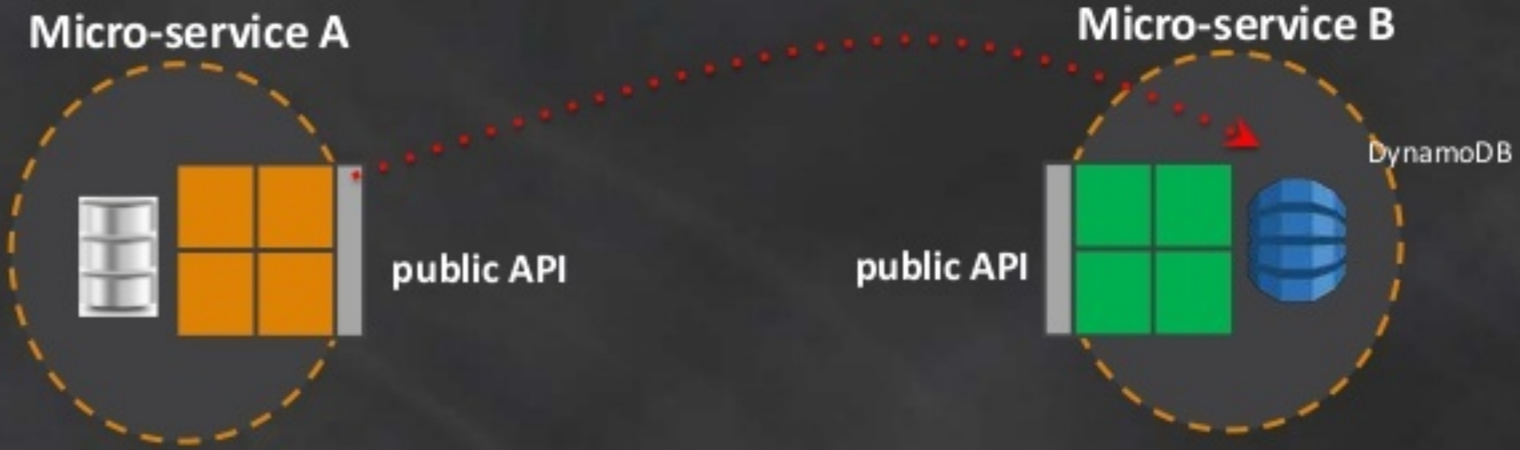
Principle 1

Micro-services only rely on each other's public API

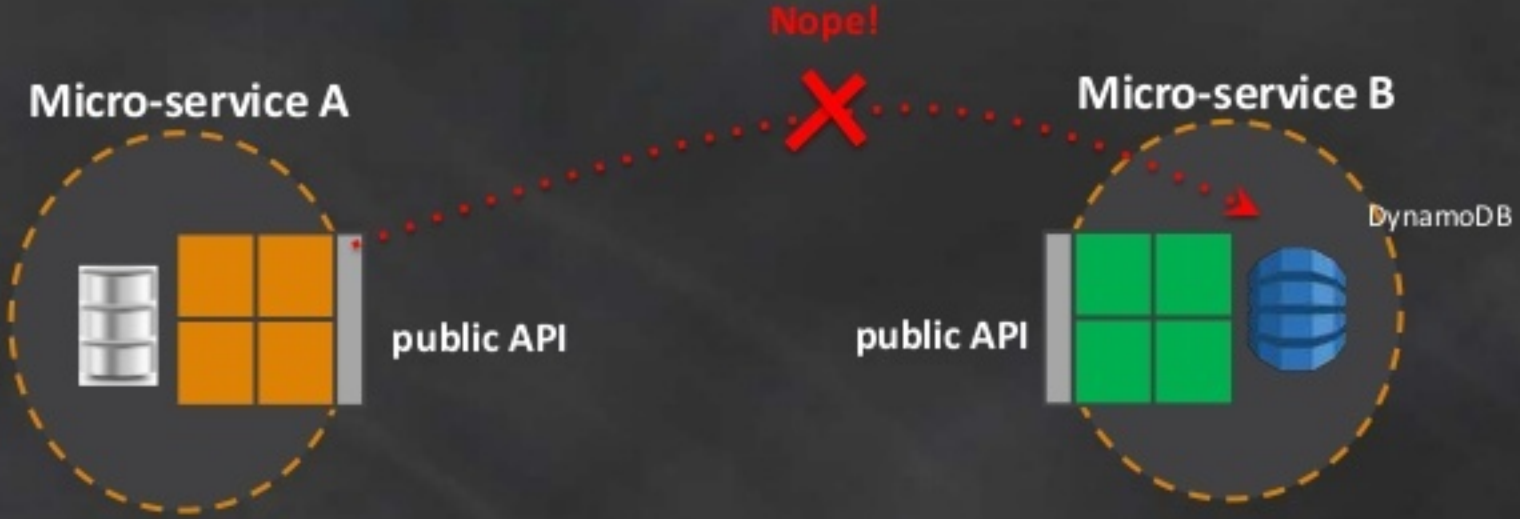
Principle 1: Microservices only rely on each other's public API



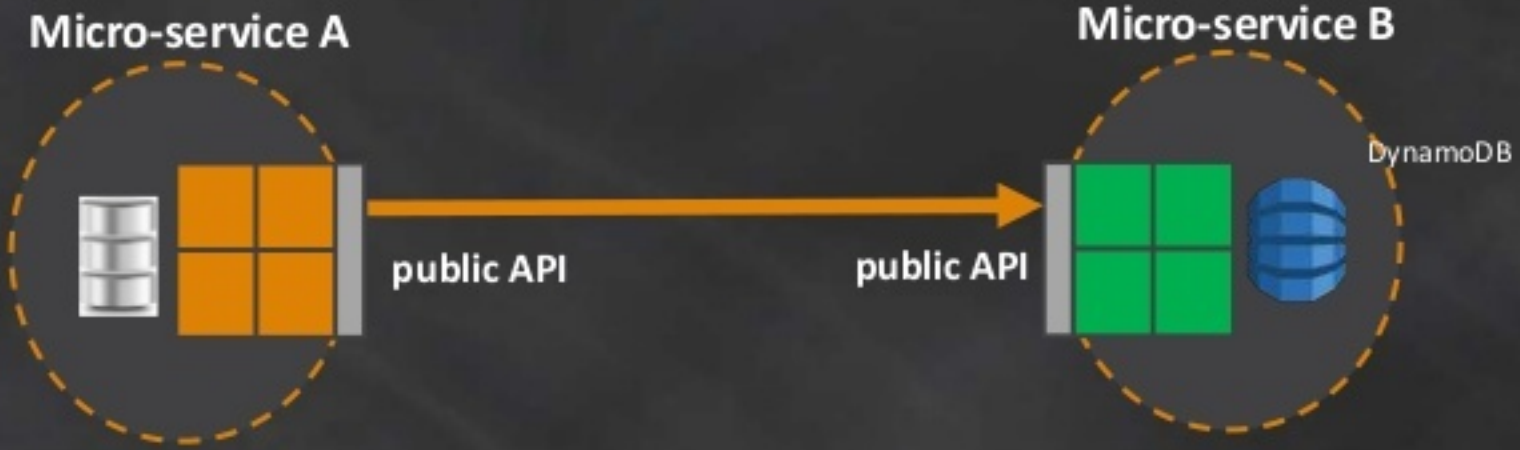
Principle 1: Microservices only rely on each other's public API (Hide Your Data)



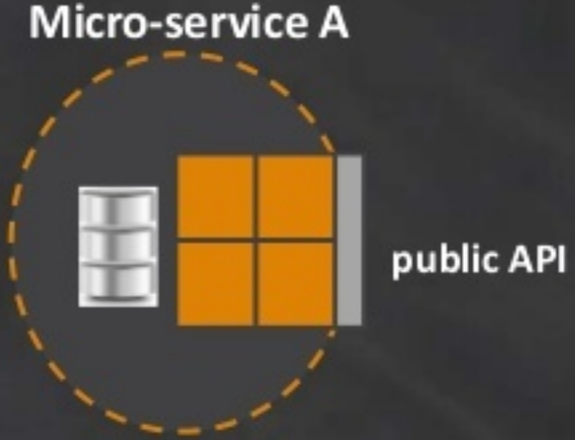
Principle 1: Microservices only rely on each other's public API (Hide Your Data)



Principle 1: Microservices only rely on each other's public API (Hide Your Data)



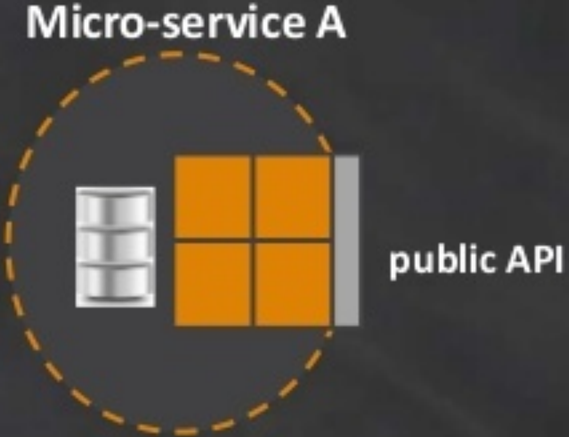
Principle 1: Microservices only rely on each other's public API (Evolve API in backward-compatible way…and document!)



Version 1.0.0

storeRestaurant (*id, name, cuisine*)

Principle 1: Microservices only rely on each other's public API (Evolve API in backward-compatible way…and document!)



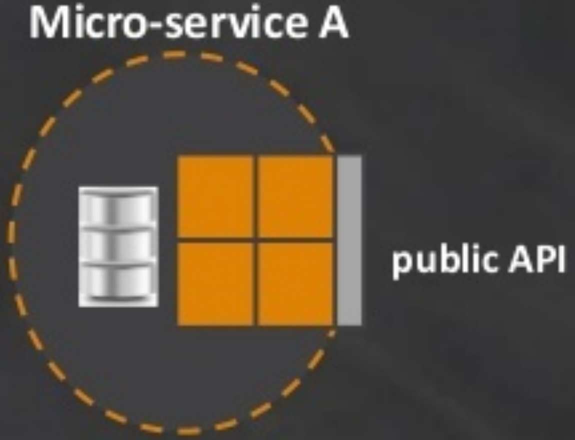
Version 1.0.0

storeRestaurant (id, name, cuisine)

Version 1.1.0

storeRestaurant (id, name, cuisine)
storeRestaurant (id, name, arbitrary_metadata)
addReview (restaurantId, rating, comments)

Principle 1: Microservices only rely on each other's public API (Evolve API in backward-compatible way…and document!)



Version 1.0.0

storeRestaurant (*id, name, cuisine*)

Version 1.1.0

storeRestaurant (id, name, cuisine)
storeRestaurant (id, name, arbitrary_metadata)
addReview (restaurantId, rating, comments)

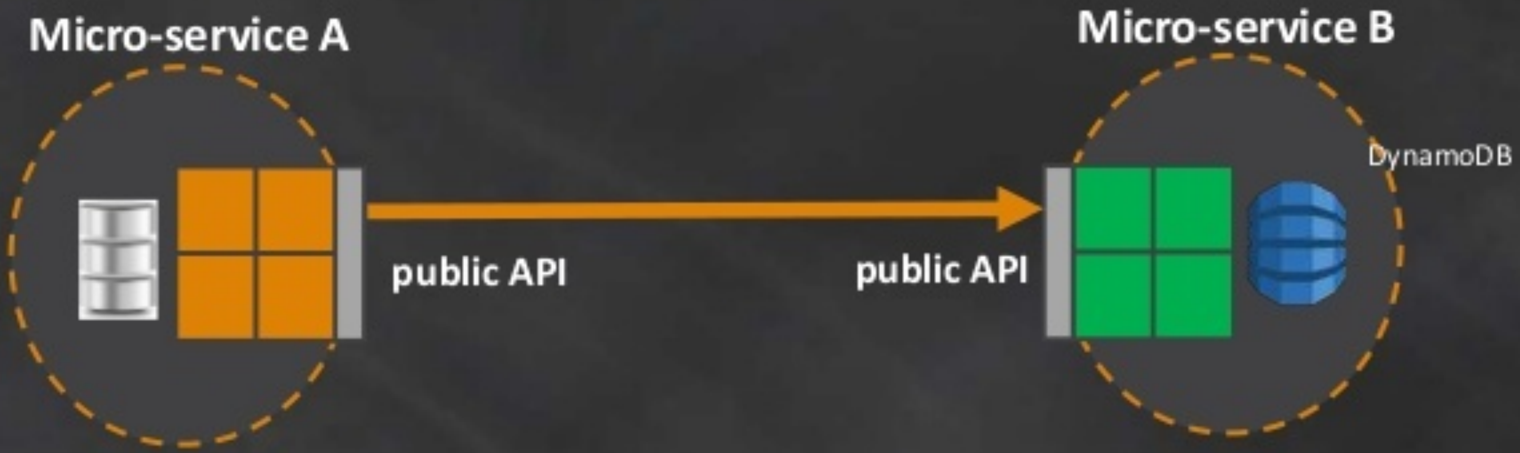
Version 2.0.0

storeRestaurant (id, name, arbitrary_metadata)
addReview (restaurantId, rating, comments)

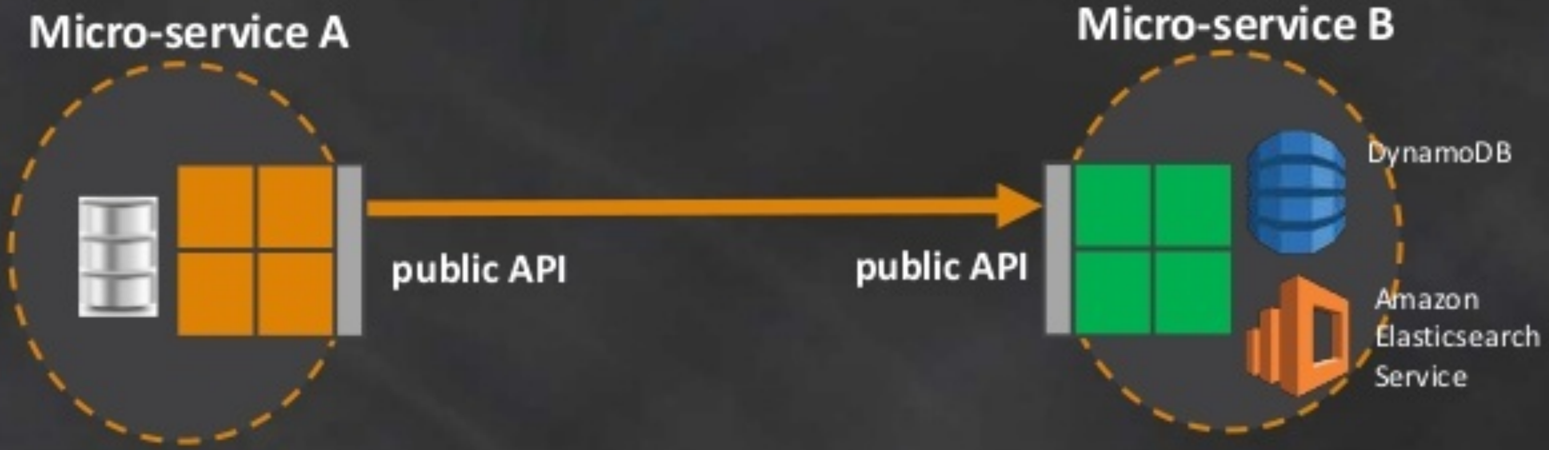
Principle 2

Use the right tool for the job

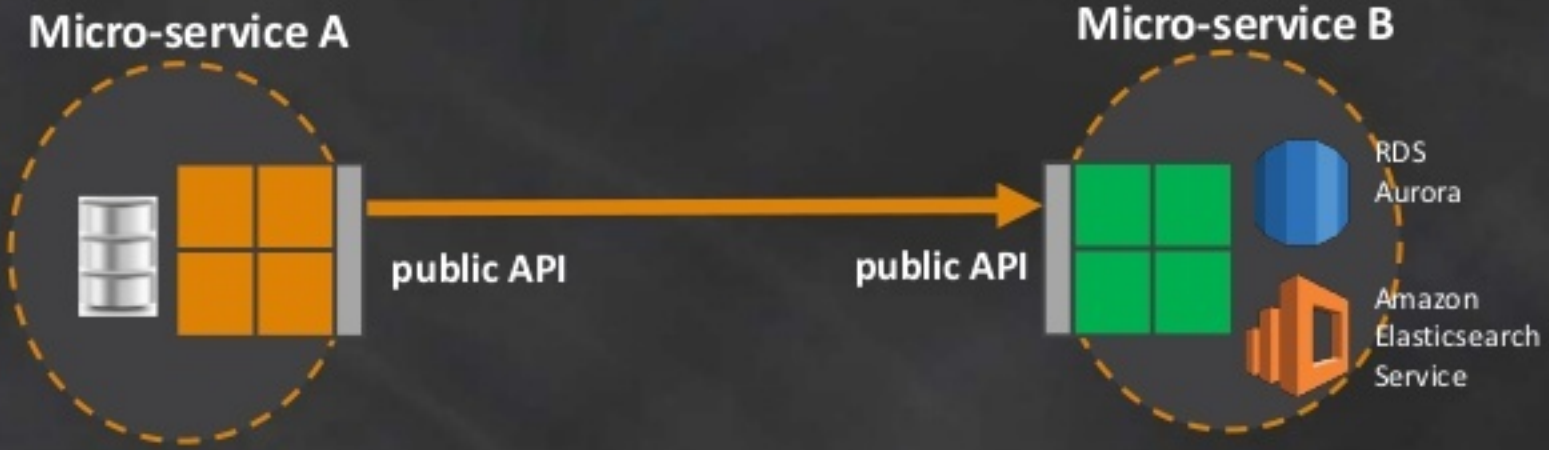
Principle 2: Use the right tool for the job
(Embrace polyglot persistence)



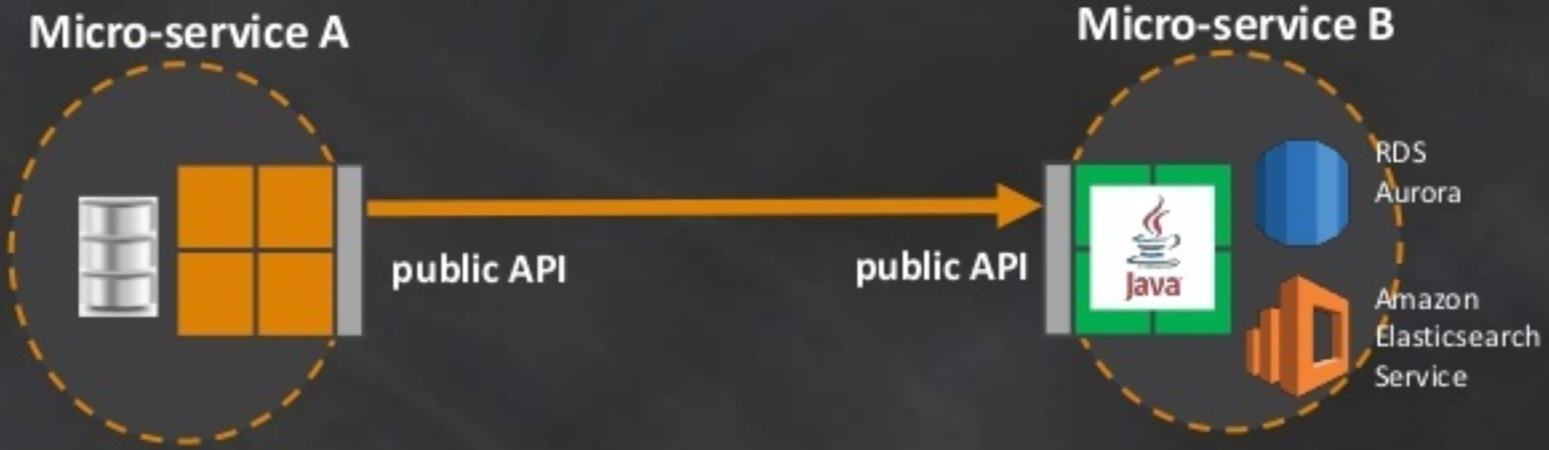
Principle 2: Use the right tool for the job (Embrace polyglot persistence)



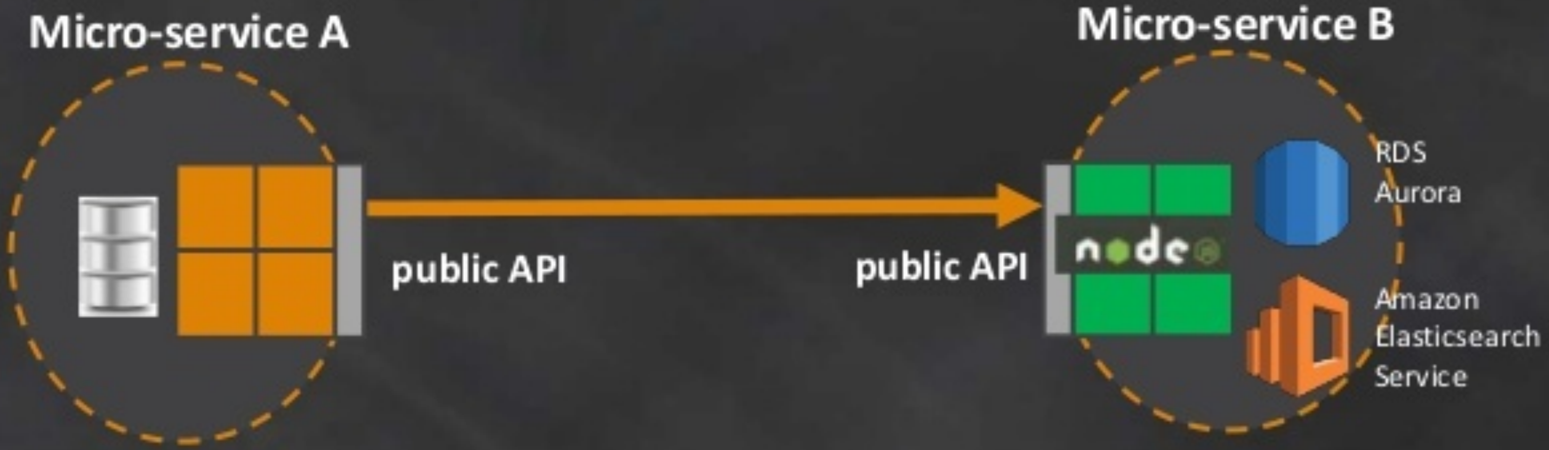
Principle 2: Use the right tool for the job
(Embrace polyglot persistence)



Principle 2: Use the right tool for the job
(Embrace polyglot programming frameworks)



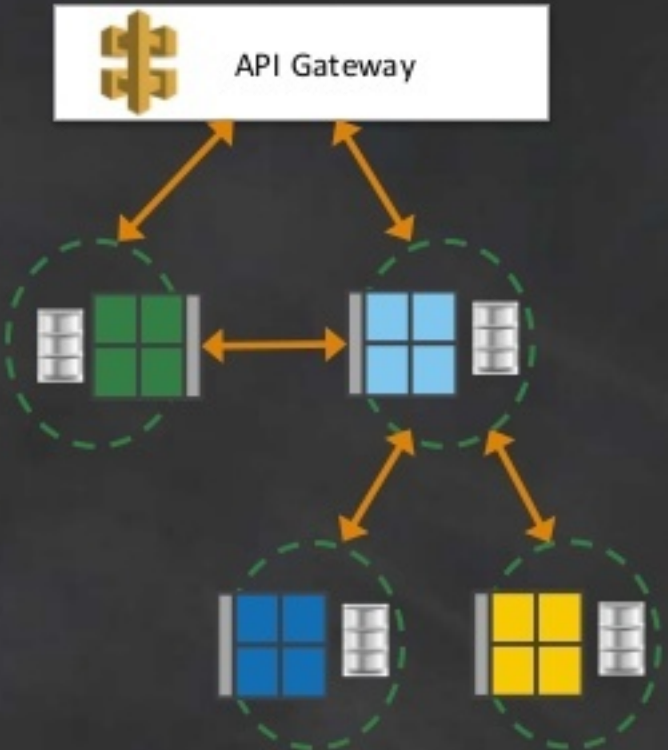
Principle 2: Use the right tool for the job
(Embrace polyglot programming frameworks)



Principle 3

Secure Your Services

Principle 3: Secure Your Services

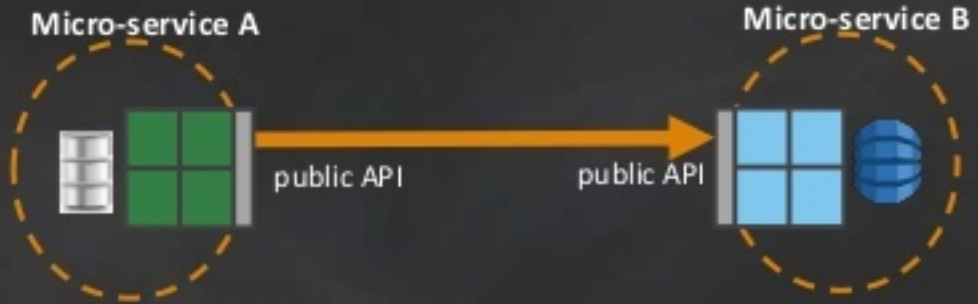


- **Defense-in-depth**
  - Network level (e.g. VPC, Security Groups, TLS)
  - Server/container-level
  - App-level
  - IAM policies
- **Gateway** ("Front door")
- **API Throttling**
- **Authentication & Authorization**
  - Client-to-service, as well as service-to-service
  - API Gateway: custom Lambda authorizers
  - IAM-based Authentication
  - Token-based auth (JWT tokens, OAuth 2.0)
- **Secrets management**
  - S3 bucket policies + KMS + IAM
  - Open-source tools (e.g. Vault, Keywhiz)

Principle 4

Be a good citizen within the ecosystem

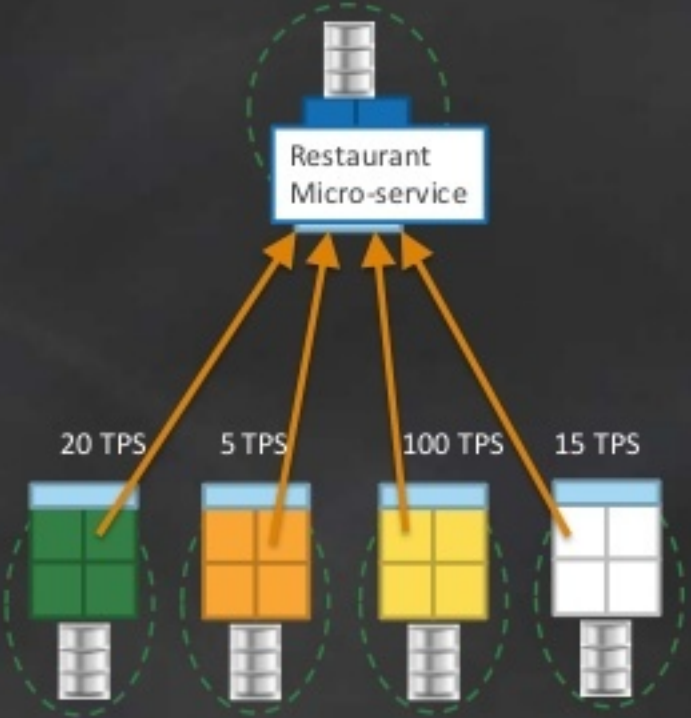
Principle 4: Be a good citizen within the ecosystem



Hey Sally, we need to call your micro-service to fetch restaurants details.
Sure Paul. Which APIs you need to call? Once I know better your use cases I'll give you permission to register your service as a client on our service's directory entry.

Principle 4: Be a good citizen within the ecosystem (Have clear SLAs)

Before we let you call our micro-service we need to understand your use case, expected load (TPS) and accepted latency



Principle 4: Be a good citizen within the ecosystem (Distributed monitoring, logging and tracing)

Distributed monitoring and tracing

- "Is the service meeting its SLA?"
- "Which services were involved in a request?"
- "How did downstream dependencies perform?"

Shared metrics

- e.g. request time, time to first byte

Distributed tracing

- e.g. Zipkin, OpenTracing

User-experience metrics

amazon webservices | Partner Network

awsmarketplace

splunk>

loggly

APPDYNAMICS

New Relic.

DATADOG

sumologic

sensu

...and many, many others!

Principle 5

More than just technology transformation

Conway's Law

"Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure."

Melvin E. Conway, 1967

Decentralize governance and data management

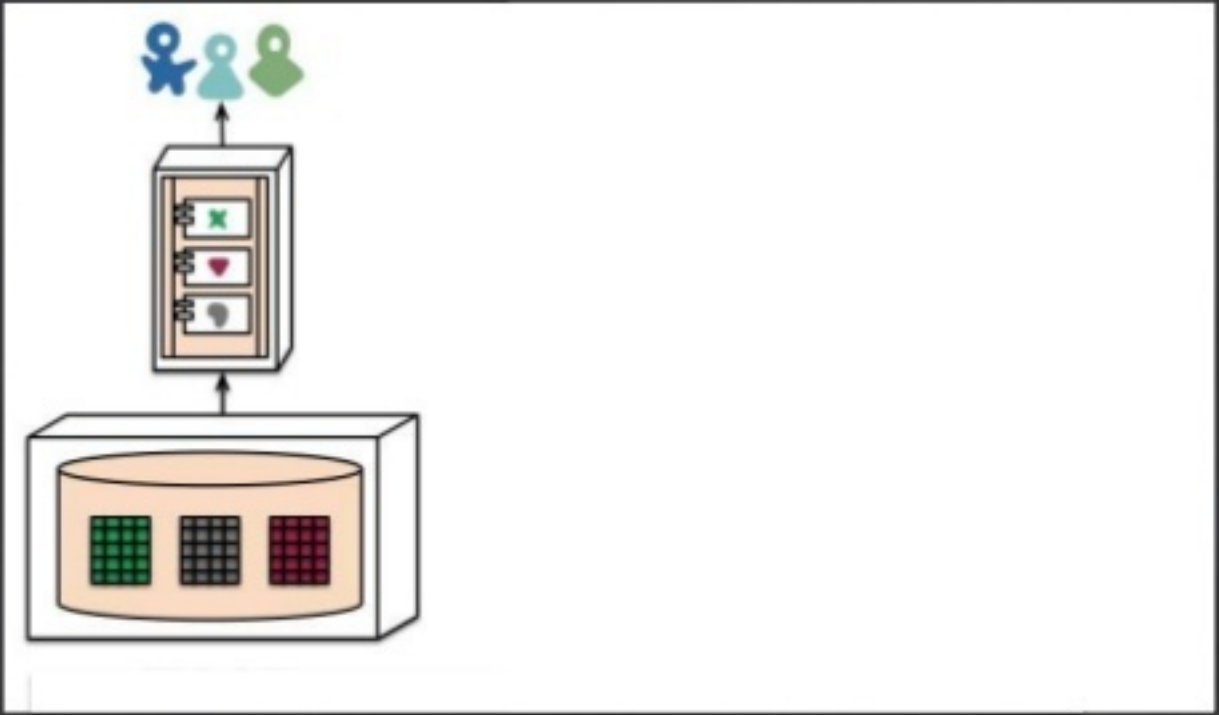
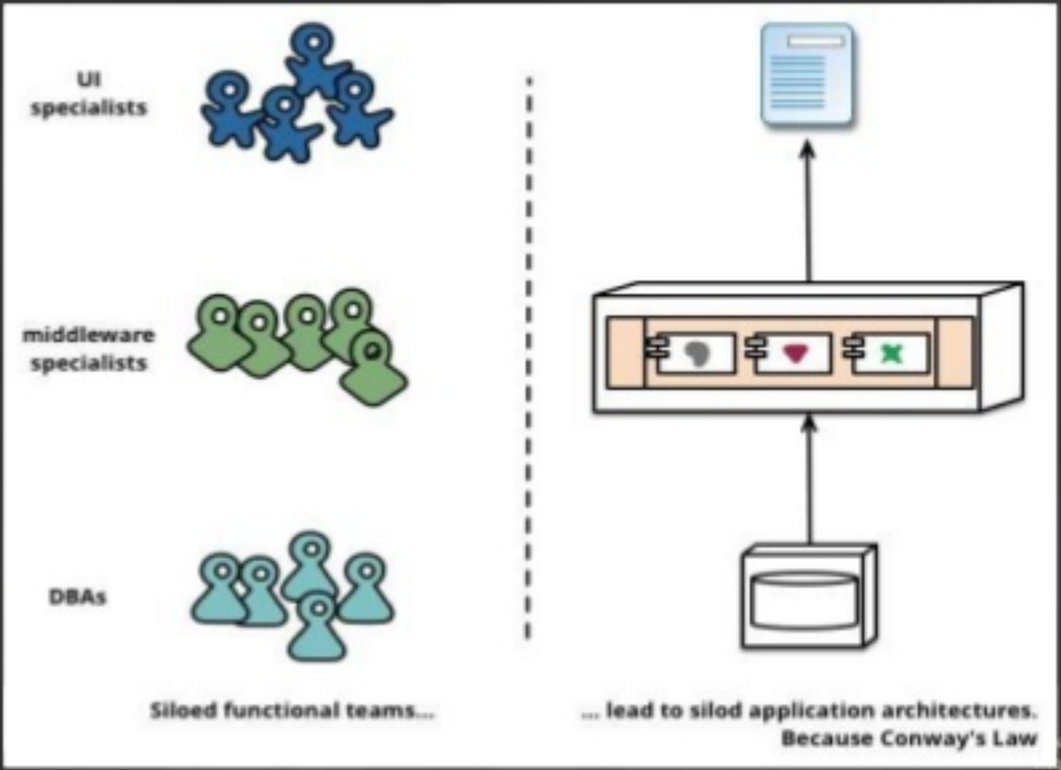


Image from Martin Fowler's article on microservices, at http://martinfowler.com/articles/microservices.html
No alterations other than cropping.
Permission to reproduce: http://martinfowler.com/faq.html

Silo’d functional teams → silo’d application architectures


UI specialists
middleware specialists
DBAs
Siloed functional teams...
... lead to silod application architectures.
Because Conway's Law

Decentralize governance and data management

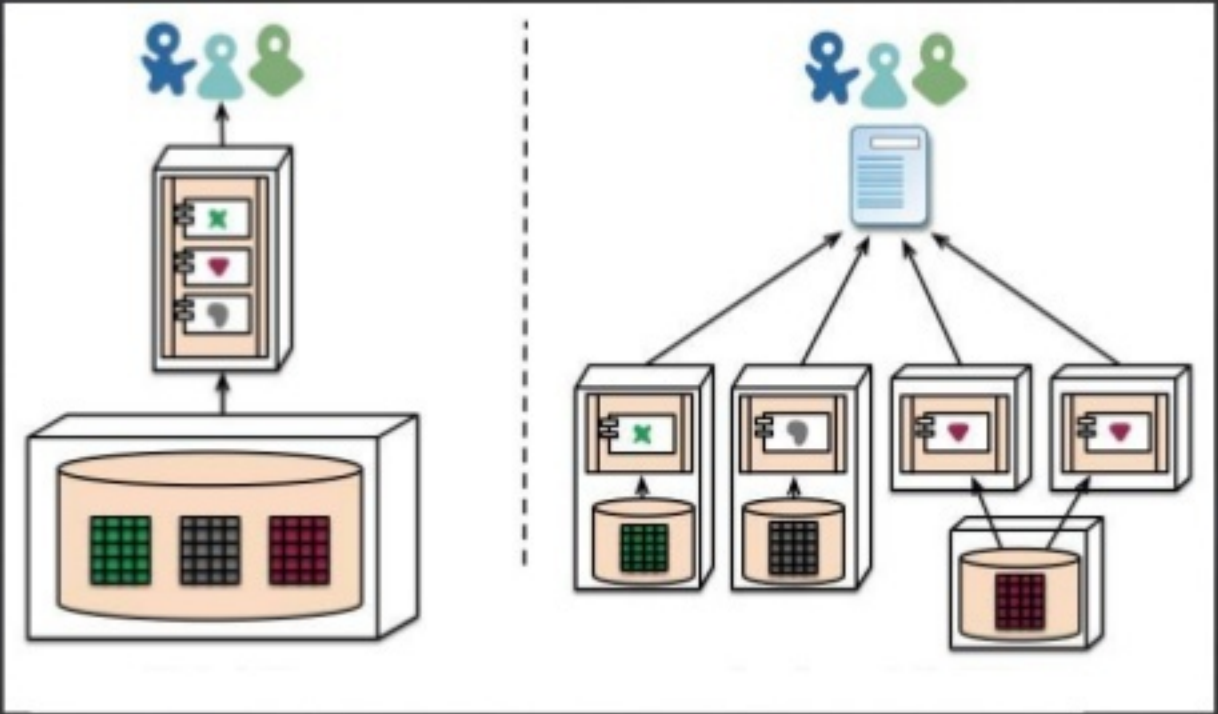
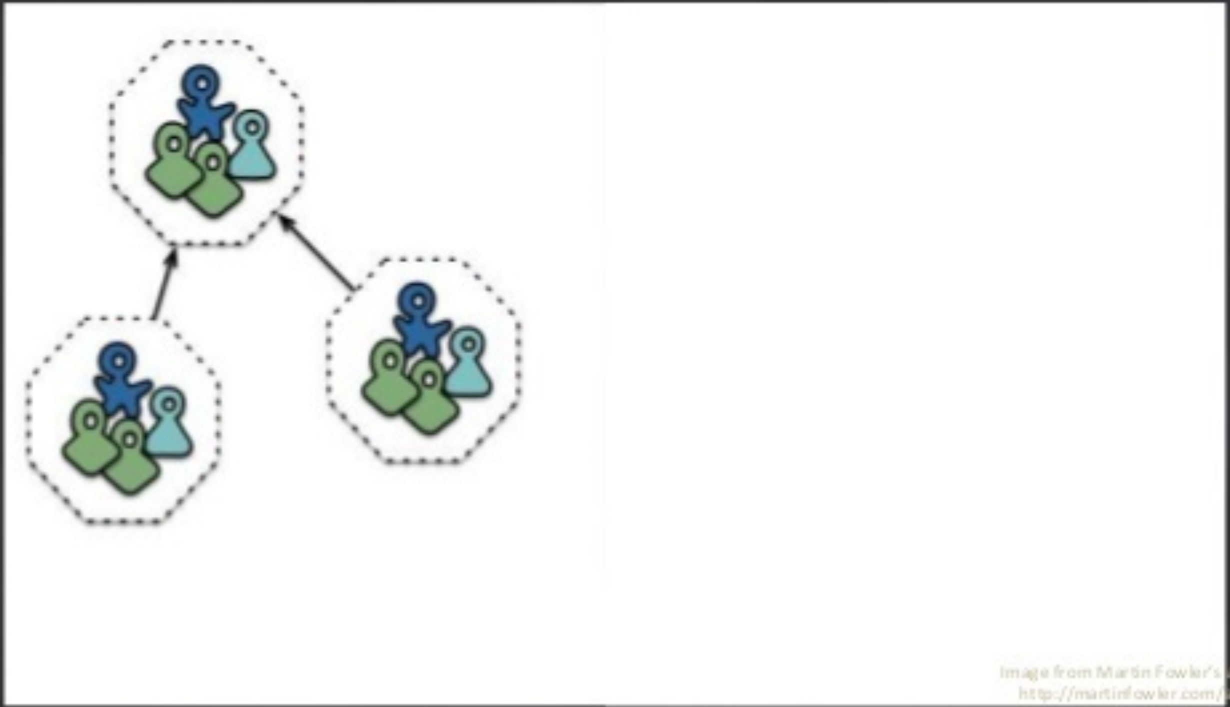
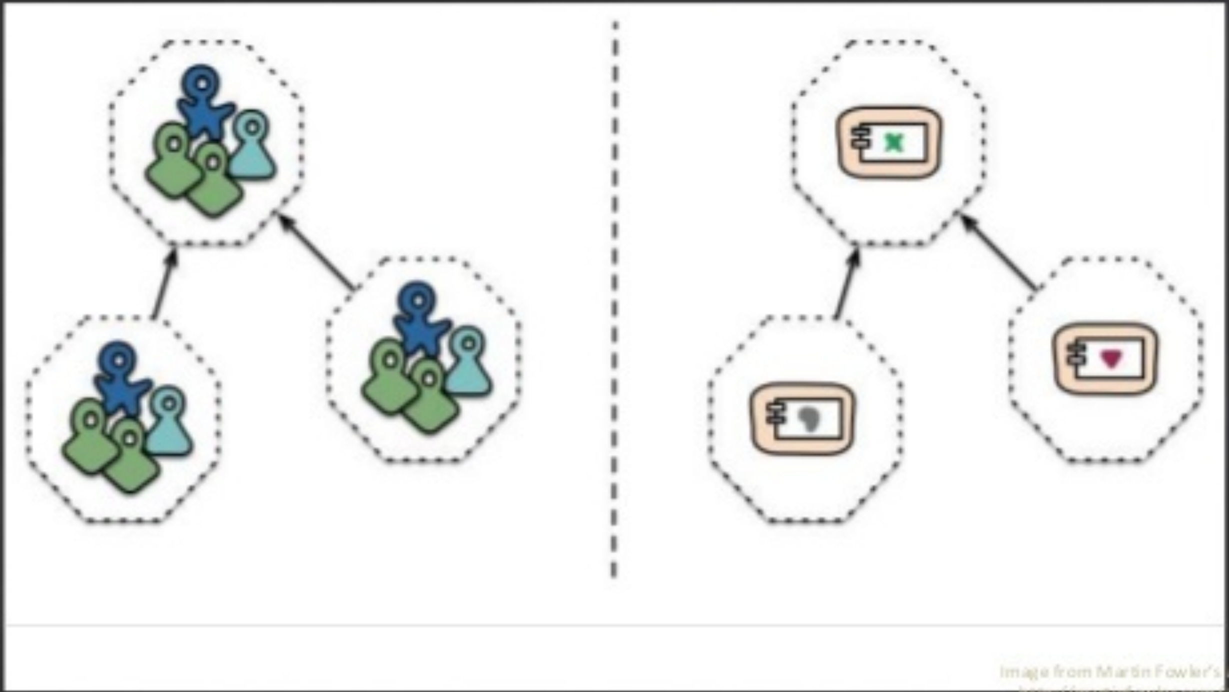


Image from Martin Fowler's article on microservices, at http://martinfowler.com/articles/microservices.html
No alterations other than cropping.
Permission to reproduce: http://martinfowler.com/faq.html

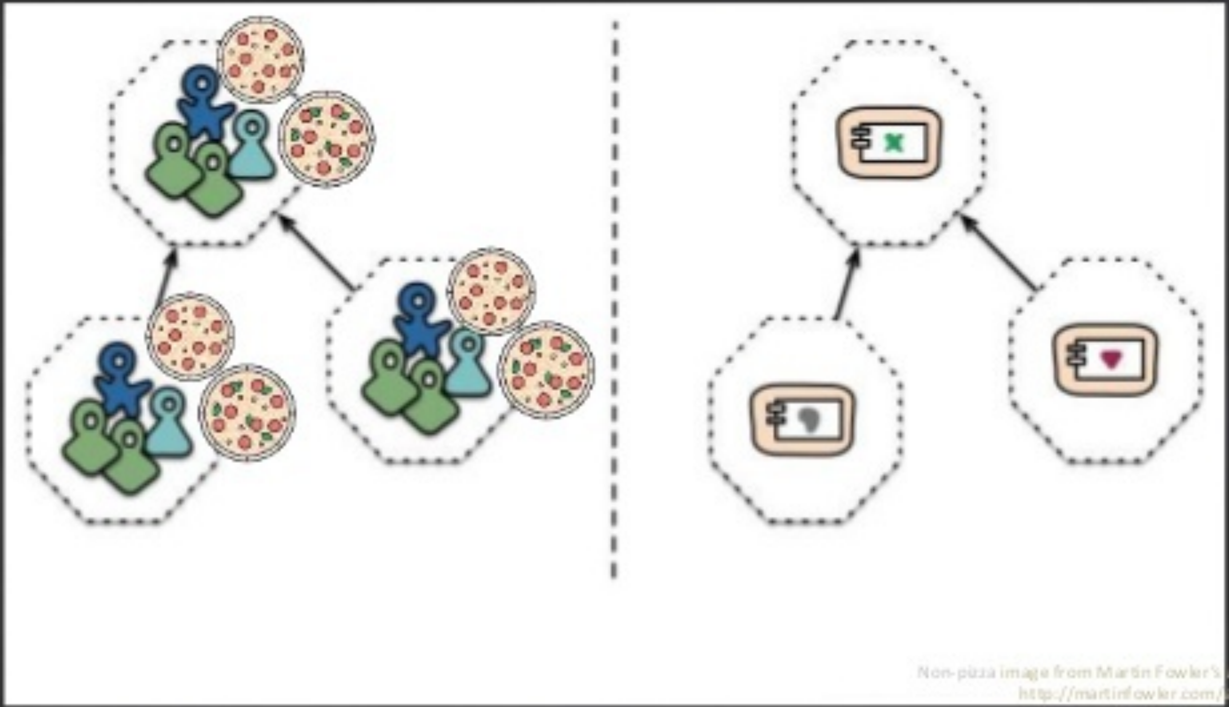
Cross functional teams → self-contained services



Cross functional teams → self-contained services



Cross functional teams → self-contained services
("Two-pizza teams" at Amazon)



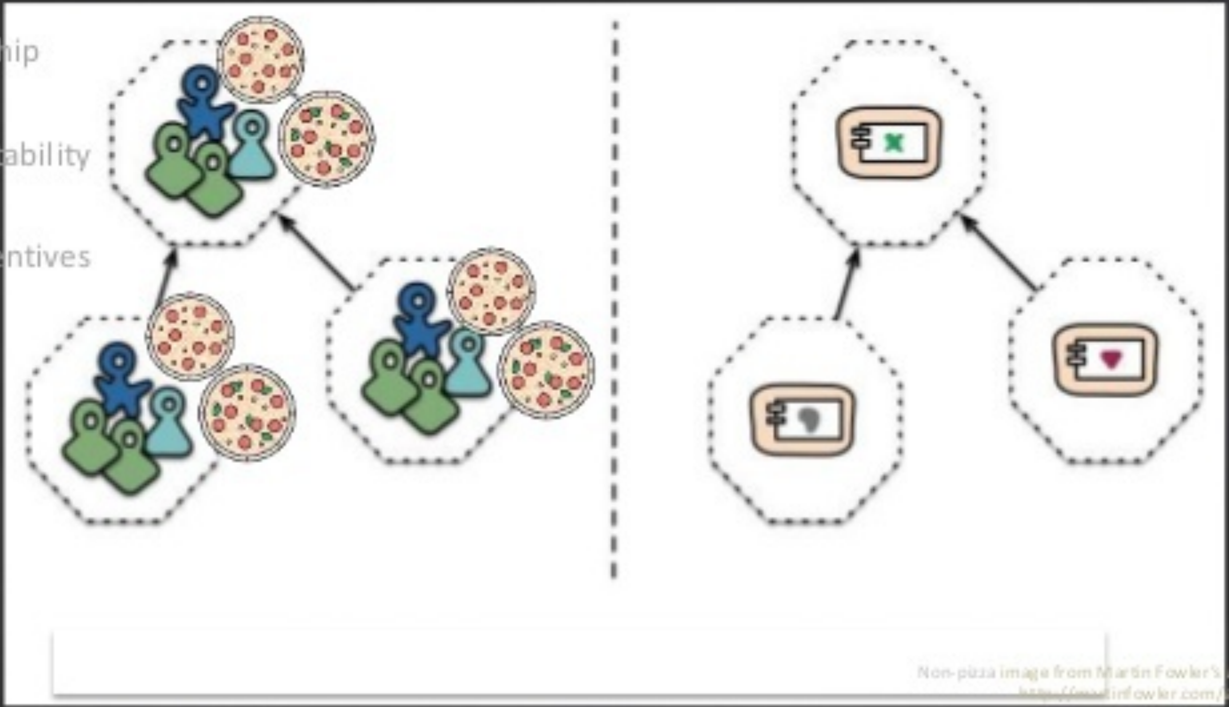
Cross functional teams → self-contained services
(“Two-pizza teams” at Amazon)

Full ownership

Full accountability

Aligned incentives

“DevOps”

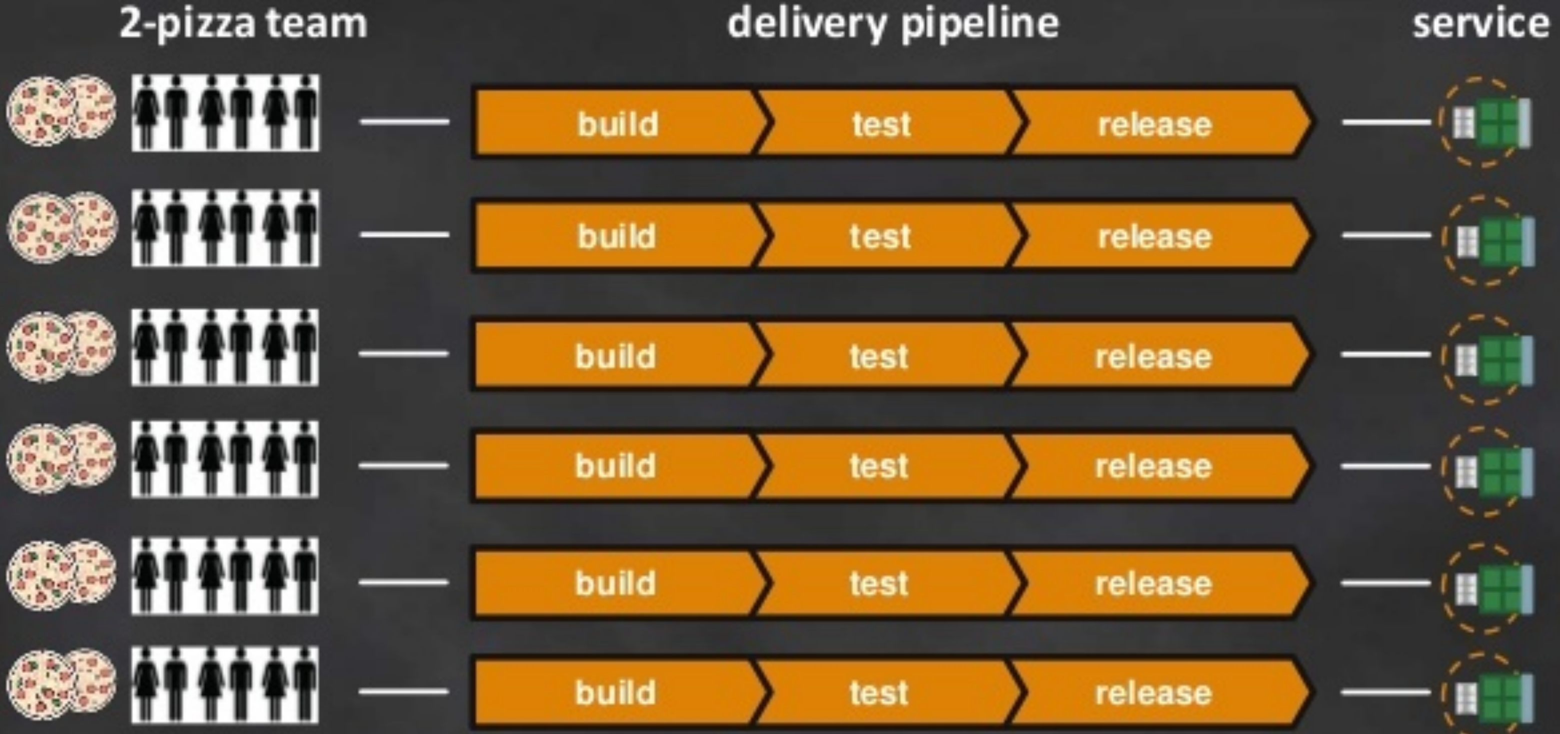


Non-pizza image from Martin Fowler's article on microservices, at http://martinfowler.com/articles/microservices.html
No alterations other than cropping.
Permission to reproduce: http://martinfowler.com/faq.html

Principle 6

Automate Everything

Focused agile teams



Principle 6: Automate everything

AWS CodeCommit

AWS CodePipeline

AWS CodeDeploy

RDS DynamoDB ElastiCache CloudWatch CloudTrail API Gateway SQS SWF

EC2 ECS Lambda Auto Scaling ELB Elastic Beanstalk SES SNS Kinesis

Principles of Microservices

1. Rely only on the public API
    - Hide your data
    - Document your APIs
    - Define a versioning strategy

2. Use the right tool for the job
    - Polygot persistence (data layer)
    - Polyglot frameworks (app layer)

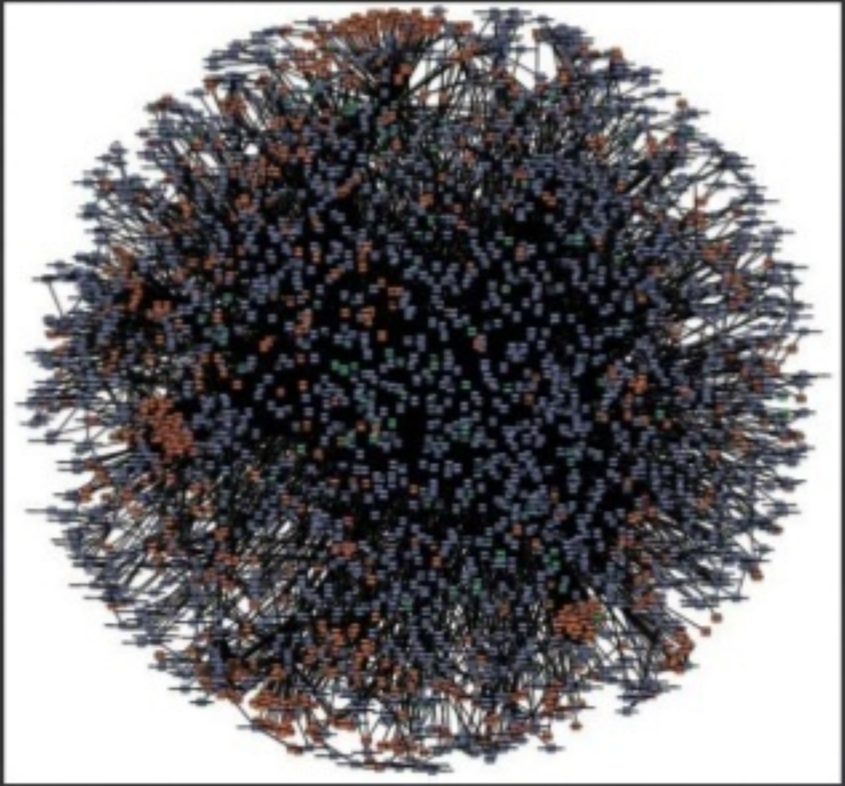
3. Secure your services
    - Defense-in-depth
    - Authentication/authorization

4. Be a good citizen within the ecosystem
    - Have SLAs
    - Distributed monitoring, logging, tracing

5. More than just technology transformation
    - Embrace organizational change
    - Favor small focused dev teams

6. Automate everything
    - Adopt DevOps

It’s a journey…



Expect challenges along the way…

- Understanding of business domains
- Coordinating txns across multiple services
- Eventual Consistency
- Service discovery
- Lots of moving parts requires increased coordination
- Complexity of testing / deploying / operating a distributed system
- Cultural transformation

Benefits of microservices

Easier to scale each individual micro-service

Rapid Build/Test/Release Cycles

Clear ownership and accountability

Benefits of microservices

Easier to scale each individual micro-service

Easier to maintain and evolve system

Rapid Build/Test/Release Cycles

New releases take minutes

Clear ownership and accountability

Short time to add new features

Benefits of microservices

Easier to scale each individual micro-service	Easier to maintain and evolve system	Increased agility
Rapid Build/Test/Release Cycles	New releases take minutes	Faster innovation
Clear ownership and accountability	Short time to add new features	Delighted customers