Microservices

Module Outline

- Defining Microservices
- Microservices Explanation
- Understanding the Monolith
- Understanding Microservices
- Practical Considerations

What are Microservices?

- Best described as:
  - An architectural style
  - An alternative to more traditional 'monolithic' applications
  - Decomposition of single system into a suite of small services, each running as independent processes and intercommunicating via open protocols
- With all the benefits / risks this implies.

Definitions from the Experts

- Developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

  -Martin Fowler

- Fine-grained SOA

  –Adrian Cockcroft - Netflix

- Gartner:

  "A microservice is a service-oriented application component that is tightly scoped, strongly encapsulated, loosely coupled, independently deployable and independently scalable."

Microservices – Working Definition:

- *Composing a single application using a suite of small services …..(rather than a single, monolithic application)*
- *… each running as independent processes (not merely modules / components within a single executable)*
- *… intercommunicating via open protocols (Like HTTP/REST, or messaging)*
- *…Separately written, deployed, scaled and maintained*

(potentially in different languages)

- *Services encapsulate business capability (rather than language constructs (classes, packages) as primary way to encapsulate.*
- *Services are independently replaceable and upgradable*

Microservices are not:

- The same as SOA

  SOA is about integrating various enterprise applications. Microservices are mainly about decomposing single applications

- A Solution for Everything!!
  - The microservices approach involves drawbacks and risks
  - It's New! You may be using microservices now and not know it!

Microservices Example

- Consider a monolithic shopping cart application:
  - Web / mobile interfaces
- Functions for:

  Searching for products

  Product catalog

  Inventory management

  Shopping cart

  Checkout

  Fufillment
- How would this look with microservices?

Monolithic Application Example

- Monolithic Shopping cart Application

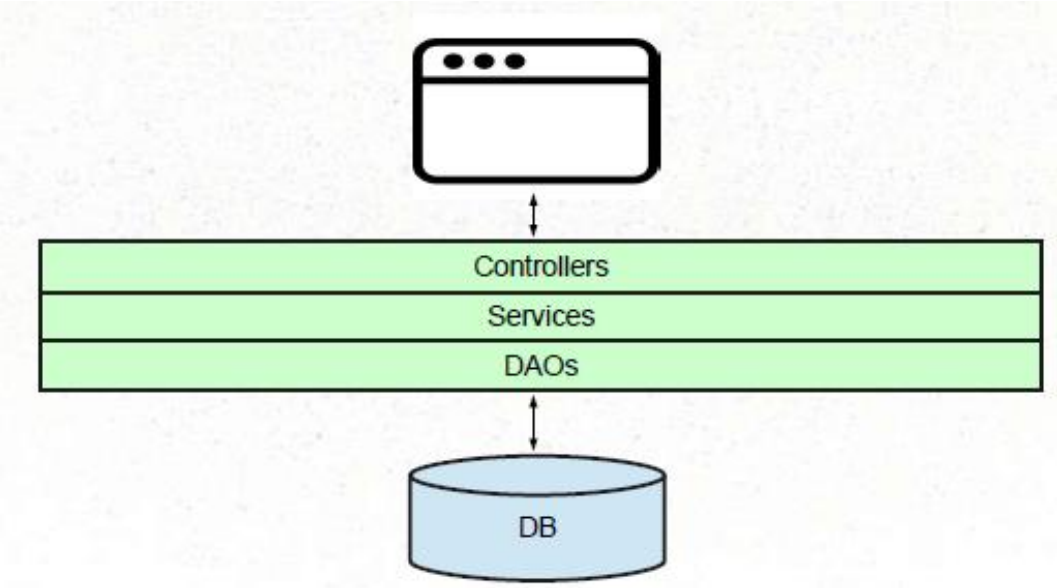


Image courtesy: Ken Krueger

Monolithic Application Example

- Understanding the Monolithic Architecture

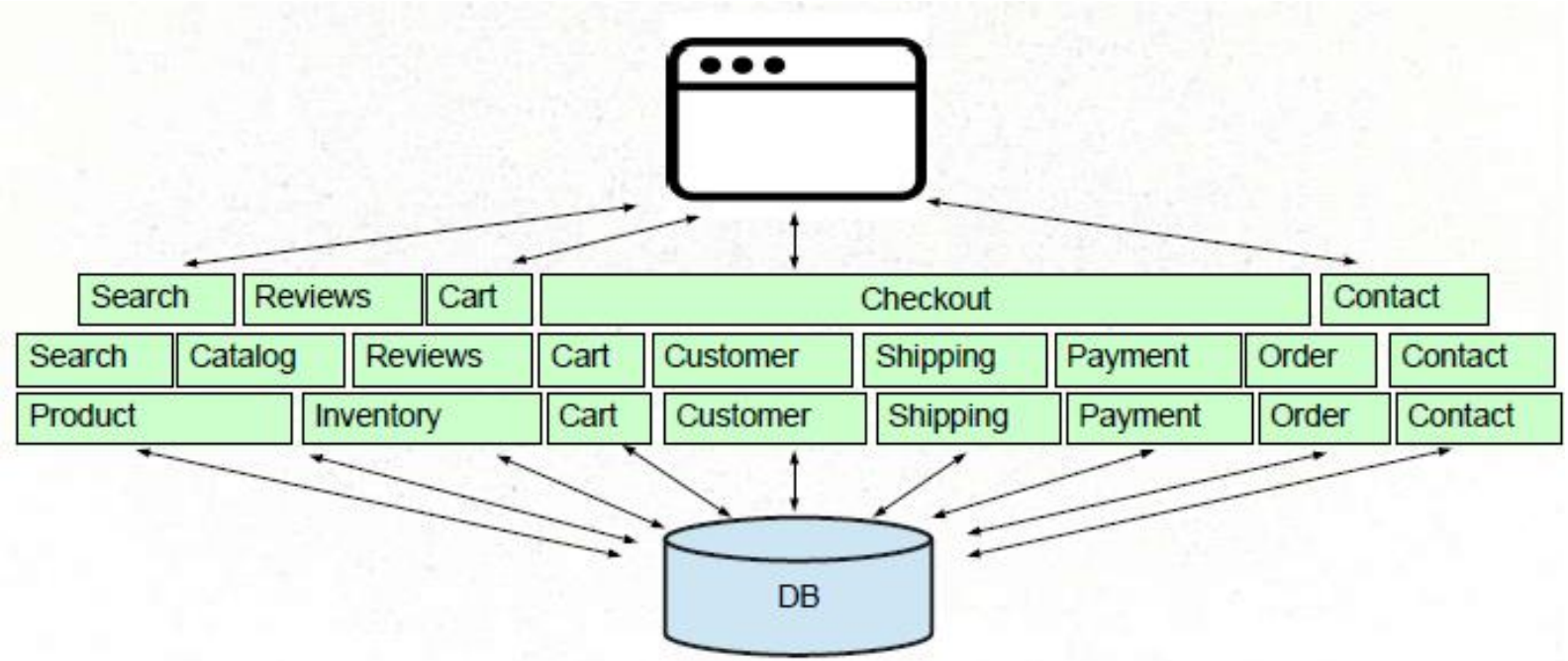


Image courtesy: Ken Krueger

Challenges in Monolithic Applications

- New Types of Client Applications

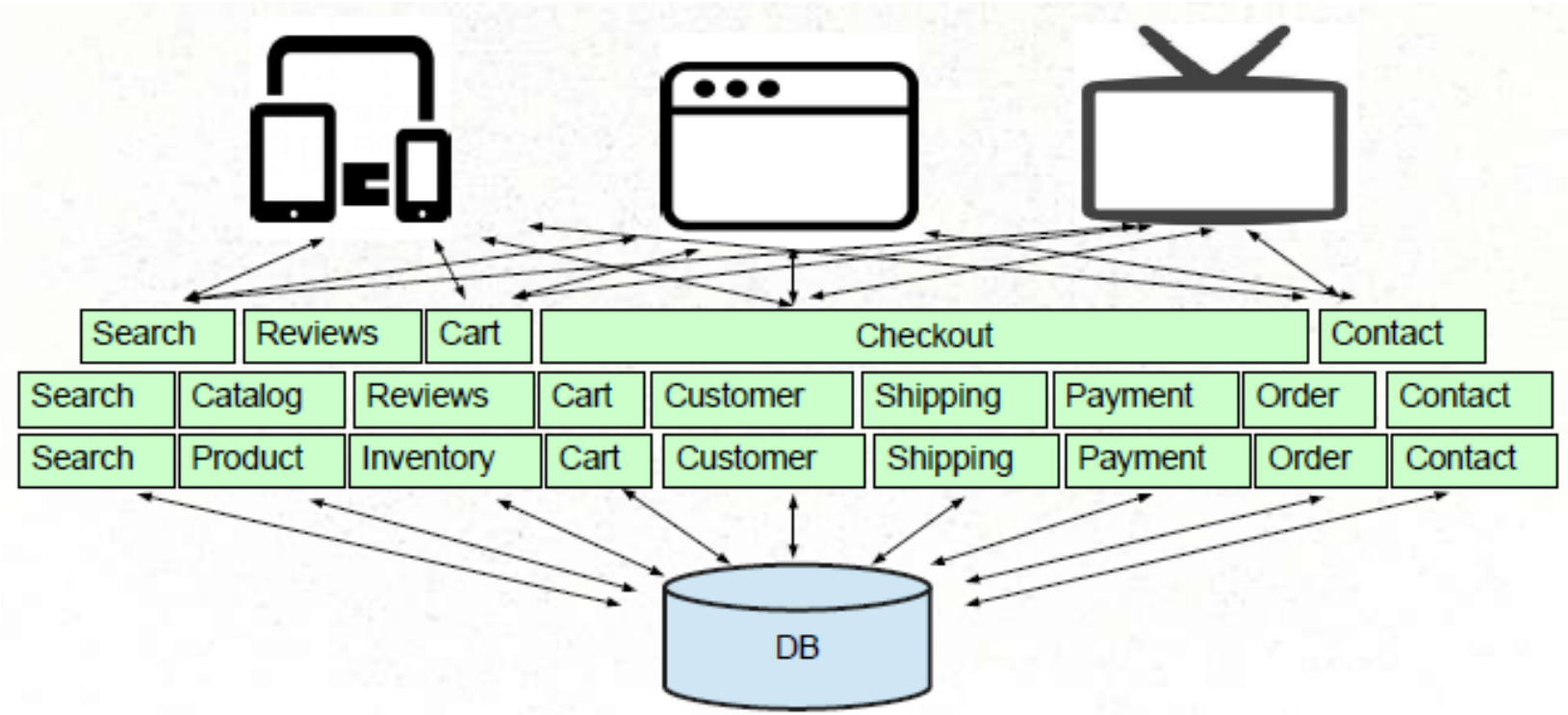
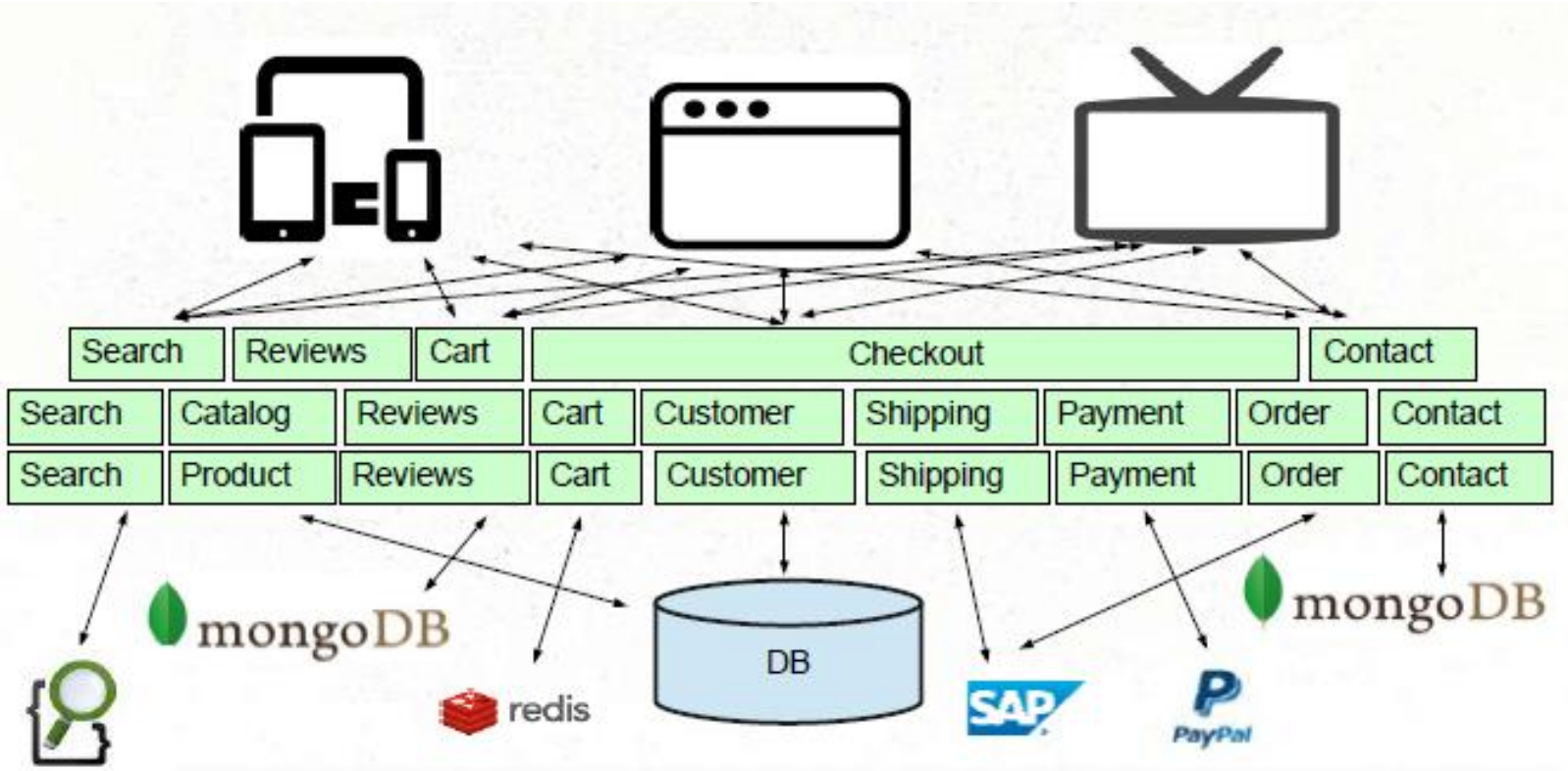


Image courtesy: Ken Krueger

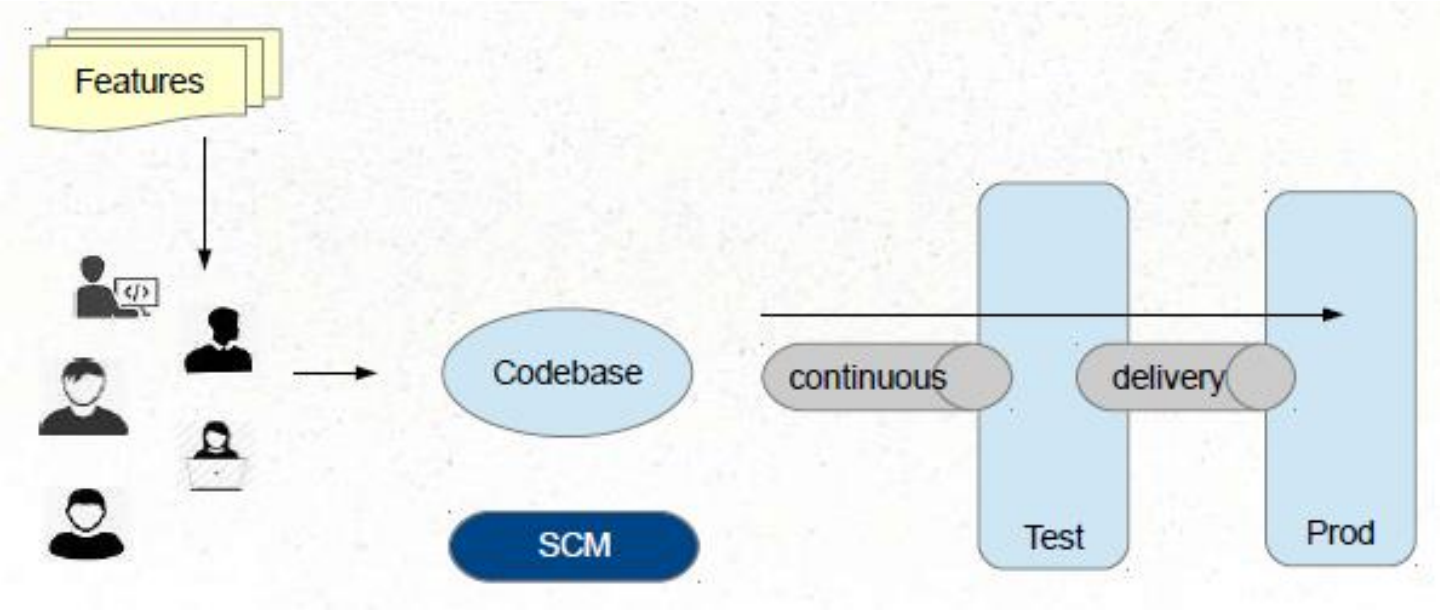
Challenges In Monolithic Application

- New types of persistence / services



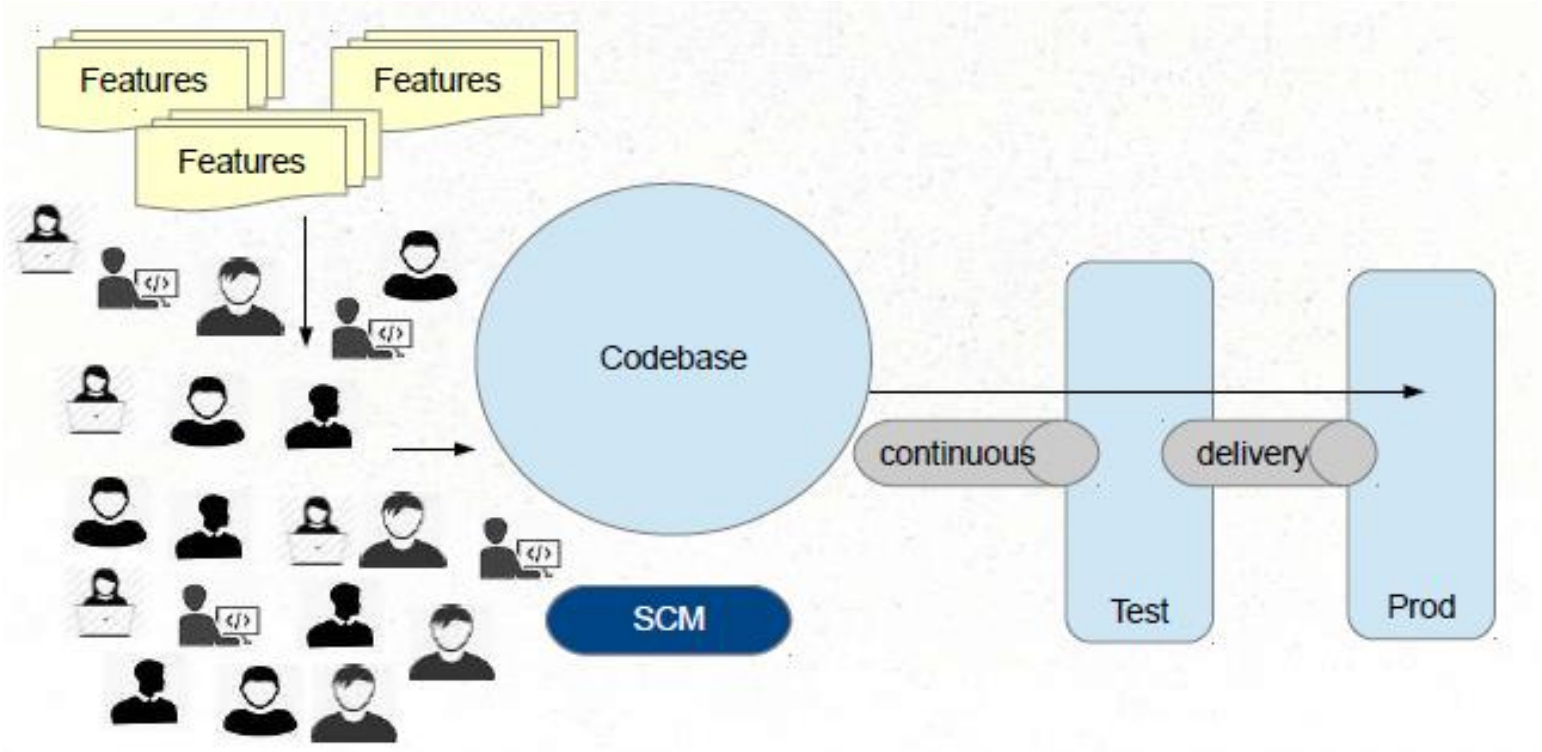
Challenges In Monolithic Application

- Single Codebase, Deployment, Versioning, Team Size



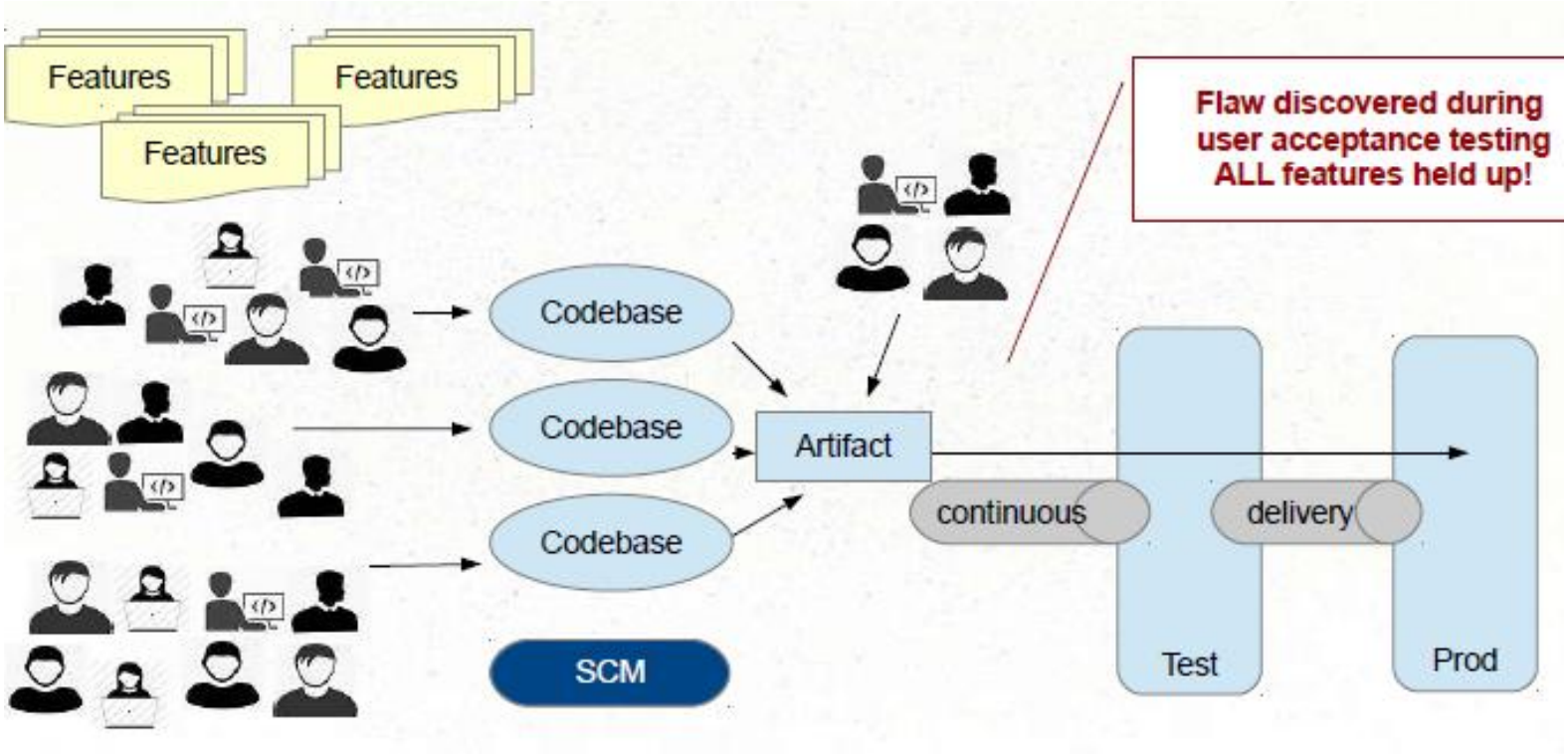
Challenges In Monolithic Application

- Single Codebase, Deployment, Versioning, Team Size



Challenges In Monolithic Application

- Single Codebase, Deployment, Versioning, Team Size



Monolithic Implementation

- Single application executable
  - Easy to comprehend, but not to digest. Must be written in a single language.
- Modularity based on Program Language
  - Using the constructs available in that language (packages, classes, functions, namespaces, frameworks)
- Various storage / service technologies used
  - RDBMS, Messaging, eMail, etc.

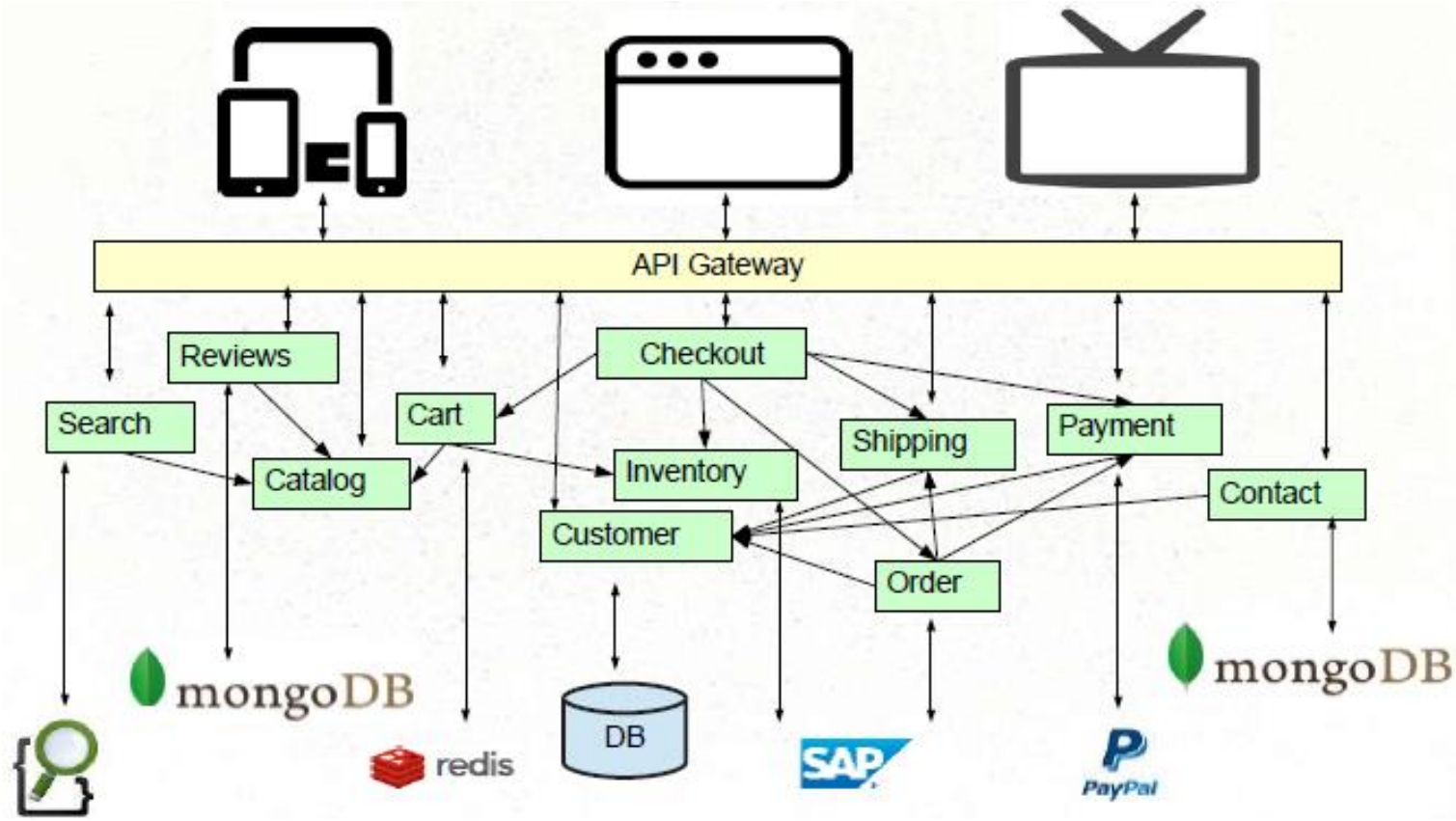
Monolithic Advantages

- Easy to comprehend (but not digest)
- Easy to test as a single unit (up to a size limit) Easy to deploy as a single unit.
- Easy to manage (up to a size limit)
- Easy to manage changes (up to a point) Easy to scale (when care is taken) Complexity managed by language constructs.

Drawbacks Of Monolithic

- Language / Framework Lock
  - Entire app written with single technology stack. Cannot experiment / take advantage of emerging technologies
- Digestion
  - Single developer cannot digest a large codebase Single team cannot manage a single large application
- Deployment as single unit
  - Cannot independently deploy single change to single component.
- Changes are “held-hostage ” by other changes

Enter The Microservices

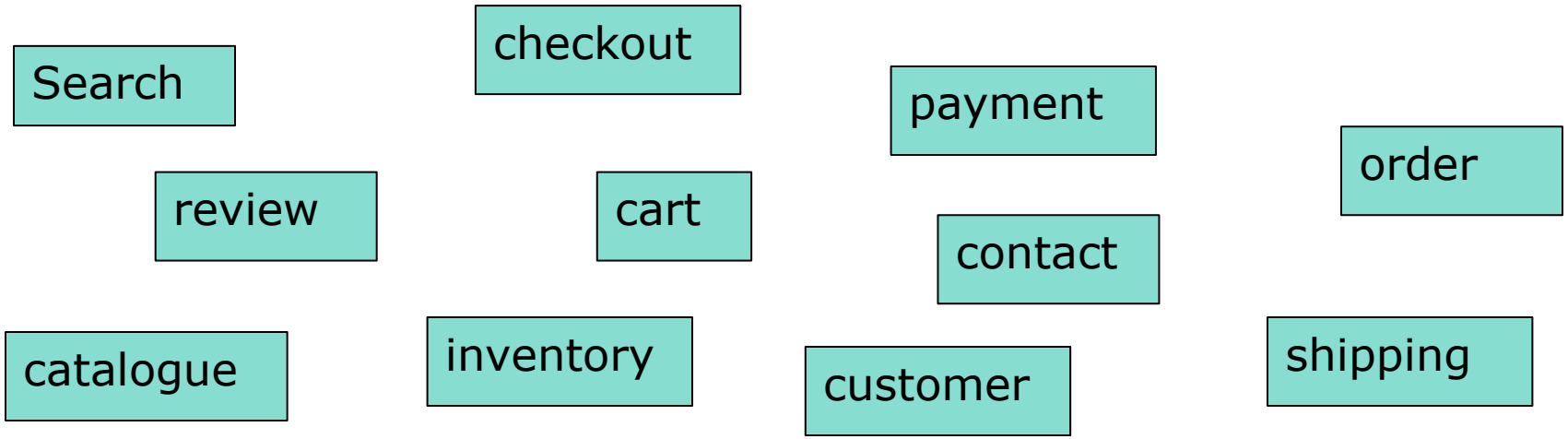


Characteristics Of Microservices

- Componentization via Services
- Composed using suite of small services
- Communication based on lightweight protocols
- Services encapsulate business capabilities
- Services easily managed
- Decentralized Governance
- Polyglot Persistence
- Polyglot Programming

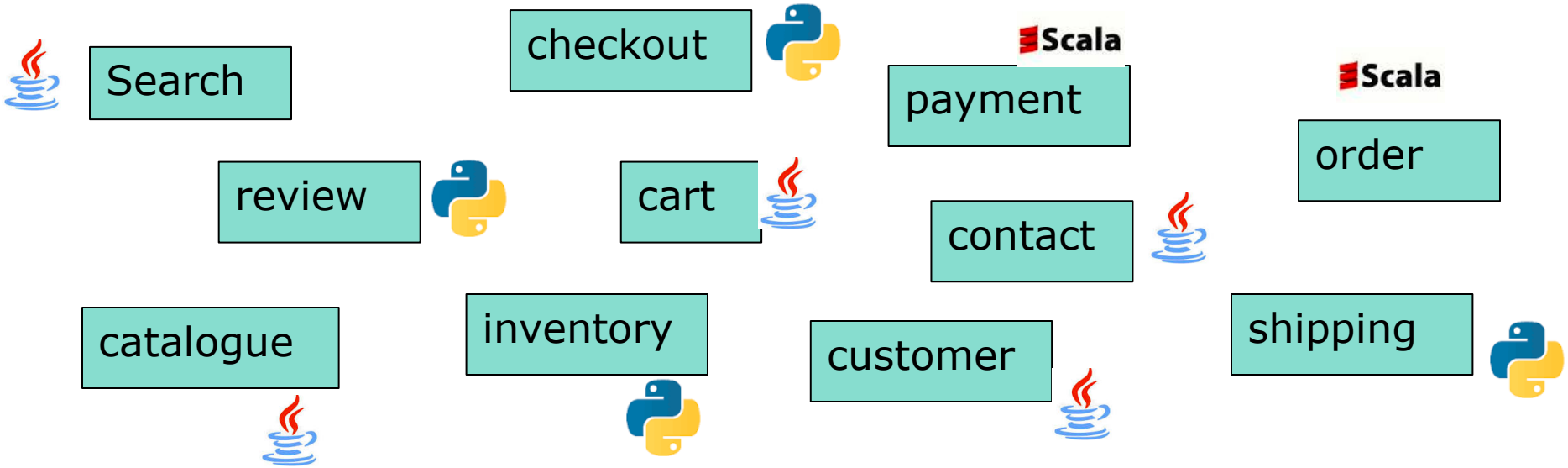
Componentization via Services

- NOT language constructs.
- Where services are small, independently deployable applications Forces the design of clear interfaces
- Changes scoped to their affected service



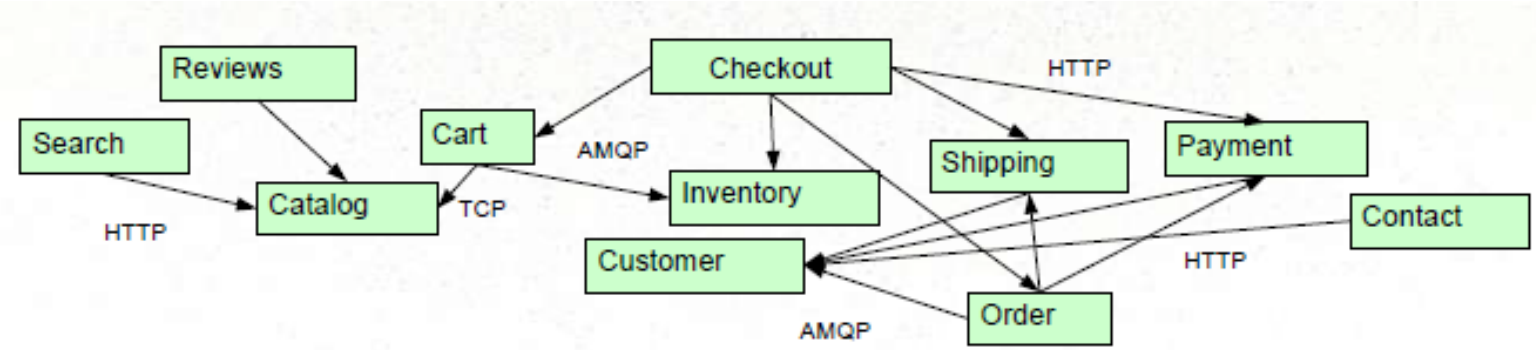
Composed using suite of small services

- Services are small, independently deployable applications
- Not a single codebase
- Not (necessarily) a single language / framework



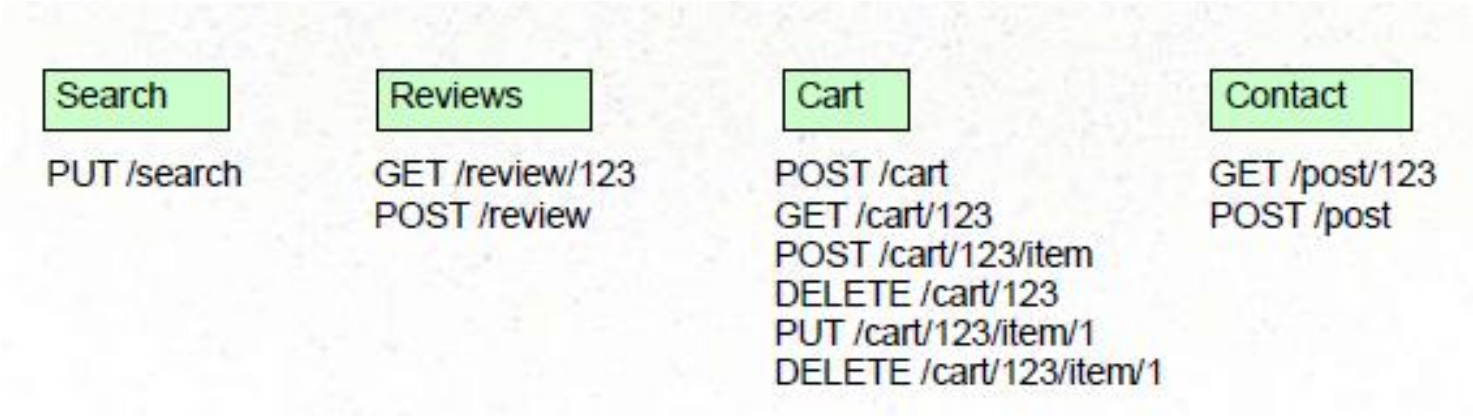
Communication based on lightweight protocols

- HTTP, TCP, UDP, Messaging, etc.
- Payloads: JSON, BSON, XML, Protocol Buffers, etc.
- Forces the design of clear interfaces
- Netflix's Cloud Native Architecture – Communicate via APIs – Not Common Database!



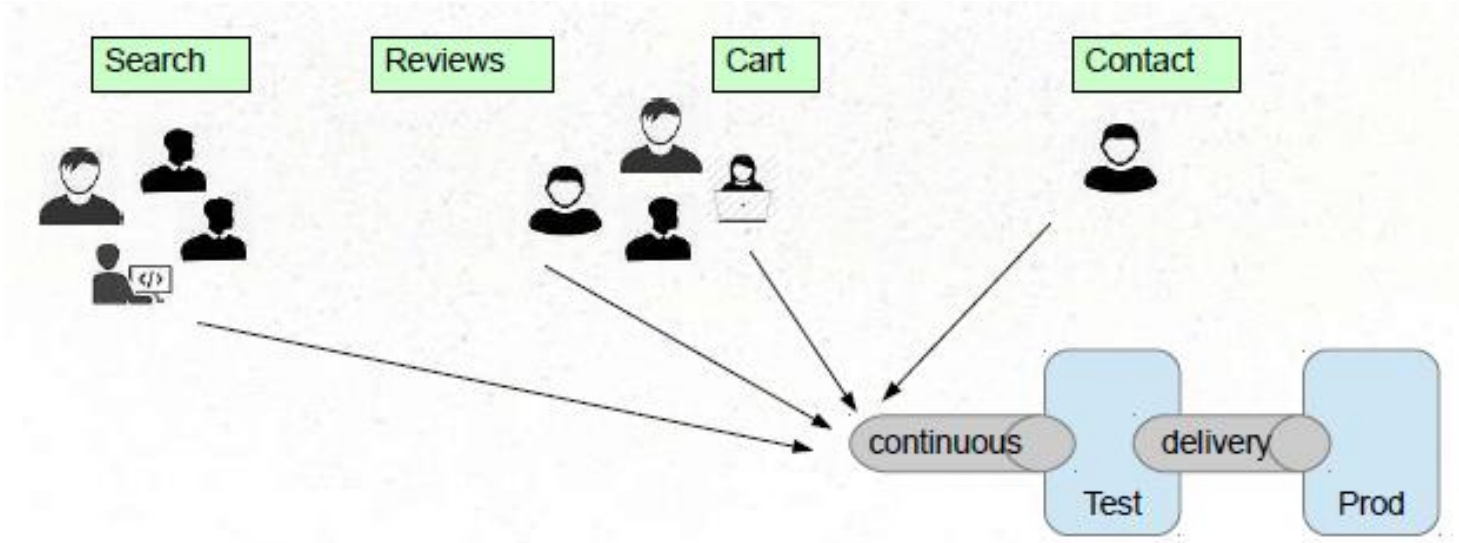
Services encapsulate business capabilities

- Not based on technology stack
- Vertical slices by business function (i.e. cart, catalog, checkout)
- ...Though technology chunk also practical (email service)
- Suitable for cross-functional teams



Services easily managed

- Easy to comprehend, alter, test, version, deploy, manage, overhaul, replace
- By small, cross-functional teams (or even individuals)

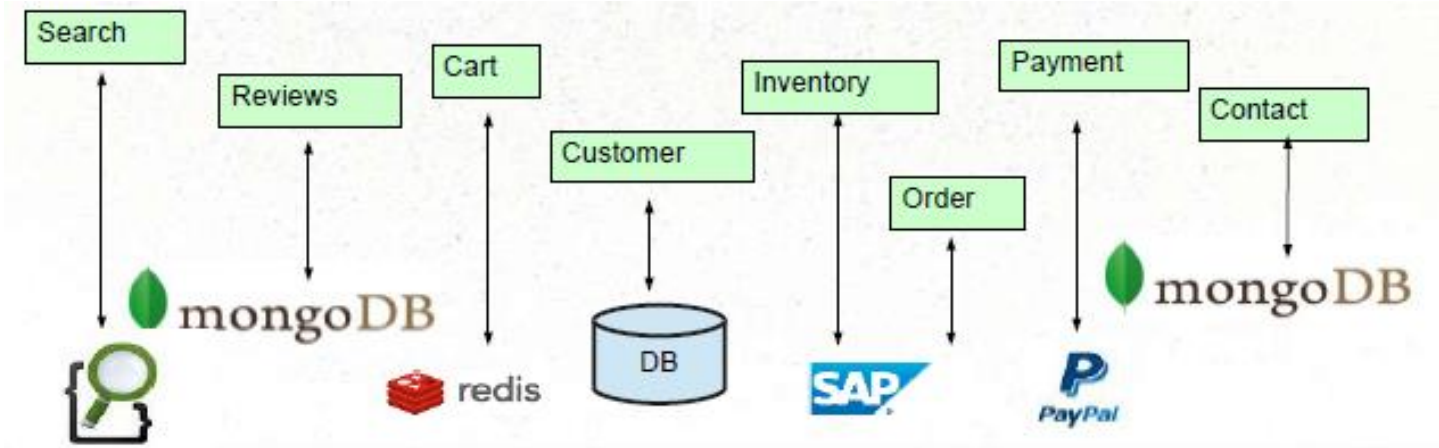


Decentralized Governance

- Use the right tool (language, framework) for the job. Services evolve at different speeds, deployed and managed according to different needs.
- Make services be “Tolerant Readers ”
- Consumer-Driven Contracts
- Antithesis of ESB

Polyglot Persistence

- Freedom to use the best technology for the job
- Don't assume single RDBMS is always best Very controversial.
  - No pan-enterprise data model!
  - No transactions!



Microservice Advantages

- Easy to digest each service (difficult to comprehend whole)
- VERY easy to test, deploy, manage, version, and scale single services
- Change cycle decoupled
- Easier to scale staff
- No Language / Framework lock.

Challenges with Microservices

- Complexity has moved out of the application, but into the operations layer
- Services may be unavailable
- Never needed to worry about this in a monolith! Design for failure, circuit breakers
- “Everything fails all the time ” - Werner Vogels, CTO Amazon
- Much more monitoring needed
- Remote calls more expensive than in-process calls

Challenges with Microservices (continued)

- Transactions: Must rely on eventual consistency over ACID
- Features span multiple services
- Change management becomes a different challenge
- Need to consider the interaction of services Dependency management / versions

Practical Considerations

How Do You Break a Monolith into Microservices?

- Primary consideration: business functionality:
  - Noun-based (catalog, cart, customer)
  - Verb-based (search, checkout, shipping)
  - Single Responsibility Principle
    - http://programmer.97things.oreilly.com/wiki/index.php/The_Single_Responsibility_Principle
  - Bounded Context
    - http://martinfowler.com/bliki/BoundedContext.html

How Micro is micro?

- Size is not the compelling factor
- Small enough for an individual developer to digest Small enough to be built and managed by small team
- Amazon's two pizza rule
- Documentation small enough to read and understand
- Dozens of secrets, not hundreds.
- Predictable.
- Easy to experiment with

Summary

- Microservices are an architectural style
- Decomposition of single system into independent running, intercommunicating services
- Alternative to Monolithic applications
- Microservices have advantages and disadvantages
- As do monoliths

Thank You

Image curtesy: Ken Krueger