

# **Microservices with Spring**

# What You Will Get From This Course?

- Introduction to Microservices architectural style
- Brief Introduction to Spring Boot
- Hands-On Experience with the Spring Cloud Components
- A Practical Guide / Excellent Starting Point

# Objectives

- By the end of this course you will be able to
  - Articulate the Microservices architectural style
    - Advantages / Disadvantages
  - Build simple Spring Boot Applications
    - Web interfaces, RESTful interfaces, Spring Data, Spring Data REST
  - Build Microservices applications using Spring Cloud
    - Utilize centralized configuration using Spring Cloud Config and Bus
    - Utilize service discovery using Eureka
    - Implement resilient service clients with Ribbon, Feign, and Hystrix

# Intended Audience

Experienced Java / Spring Developers

Not for Beginners

# Prerequisites

Knowledge of Java

Knowledge of Spring Framework / Spring Boot

ApplicationContext, Profiles, RestTemplate, @Value, @Autowired, @Component, Java Configuration, etc.

Maven – Basic concepts, dependencies, how to run mvn clean package

Git – Basic knowledge

# Caveats

Spring Cloud Changes Rapidly

Microservices are an Evolving Topic

Unlikely to cover all Topics

- Goal is to be reasonably comprehensive
- Enable you to fill in gaps yourself

# Lab Setup

Prepare your local environment to work on the exercises.

# What Are Microservices?

Understanding the Microservices architectural style  
and its impact



# Module Outline

## **Defining Microservices**

### Microservices Explanation

- Understanding the Monolith
- Understanding Microservices

### Practical Considerations

# What Are Microservices

Presently a lot of hype!

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 impli

# 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

# 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)

# Microservices – Working Definition..

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 silver bullet
- The microservices approach involves drawbacks and risks
- New! You may be using microservices now and not know it!

# Current Trends

- Twitter moved from Ruby/Rails monolith to Microservices.
- Facebook moved from PHP monolith to Microservices
- Netflix moved from Java monolith to Microservices

# Module Outline

Defining Microservices

## **Microservices Explanation**

- **Understanding the Monolith**
- Understanding Microservices

Practical Considerations



# Microservices Example

Consider a monolithic shopping cart application:

Web / mobile interfaces

Functions for:

- Searching for products

- Product catalog

- Inventory management

- Shopping cart

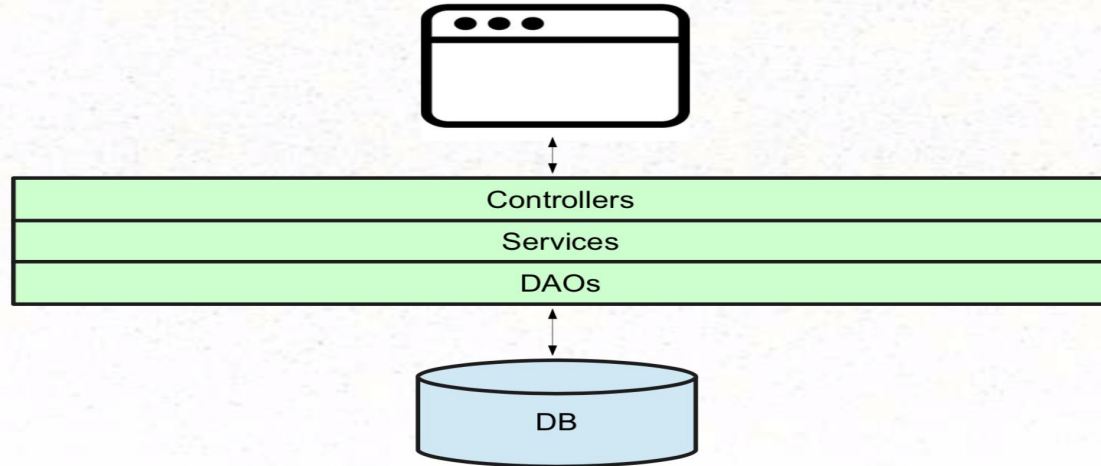
- Checkout

- Fulfillment

How would this look with microservices?

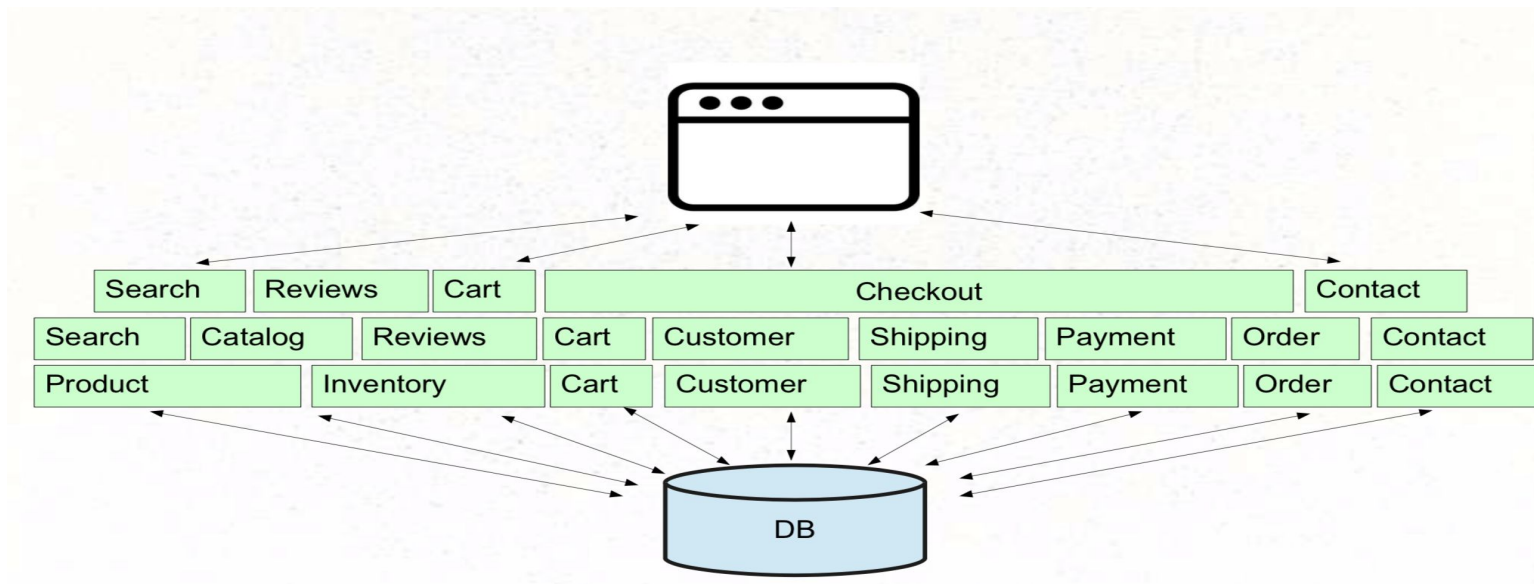
# Monolithic Application Example

Monolithic shopping cart application:



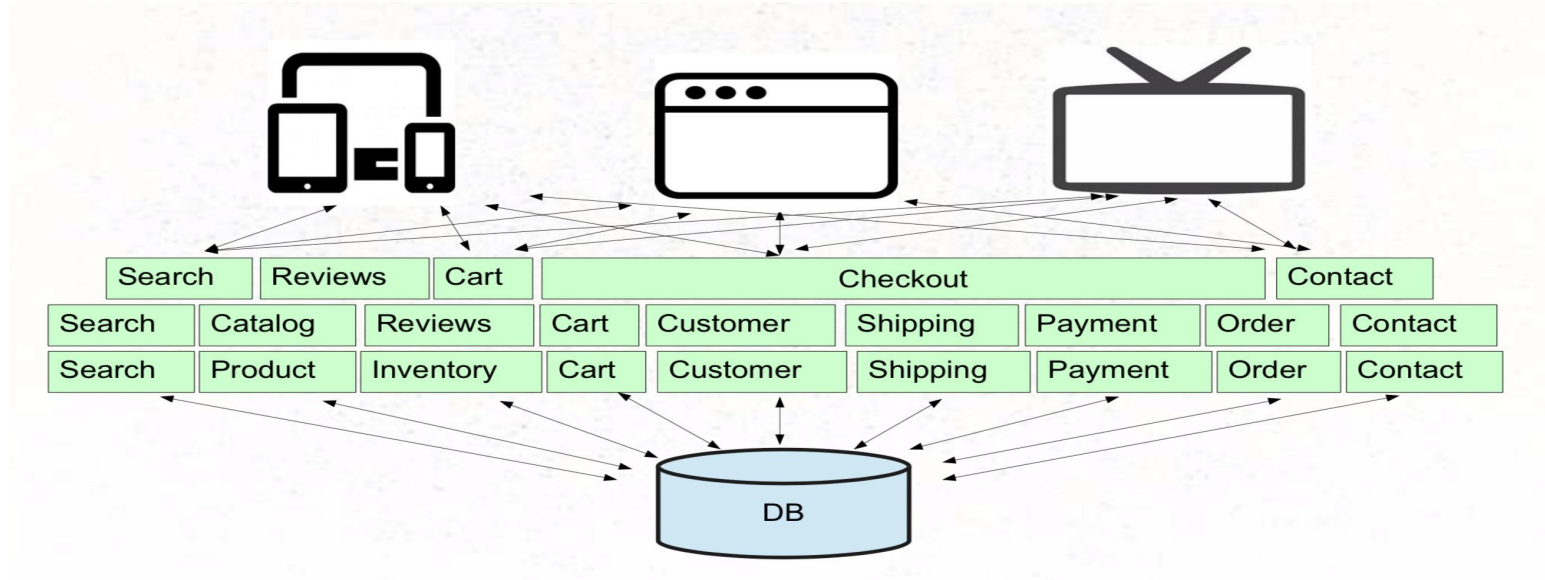
# Monolithic Application Example

## Understanding the Monolithic Architecture



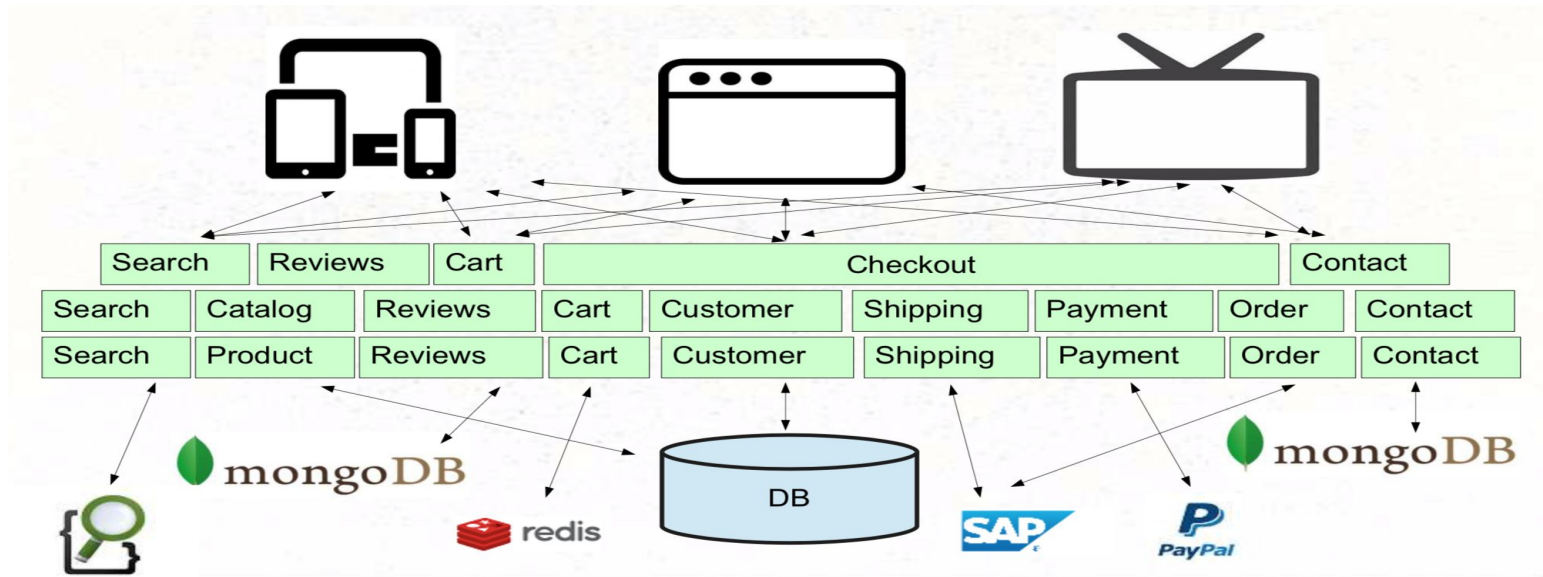
# Monolithic Challenges

New types of client applications



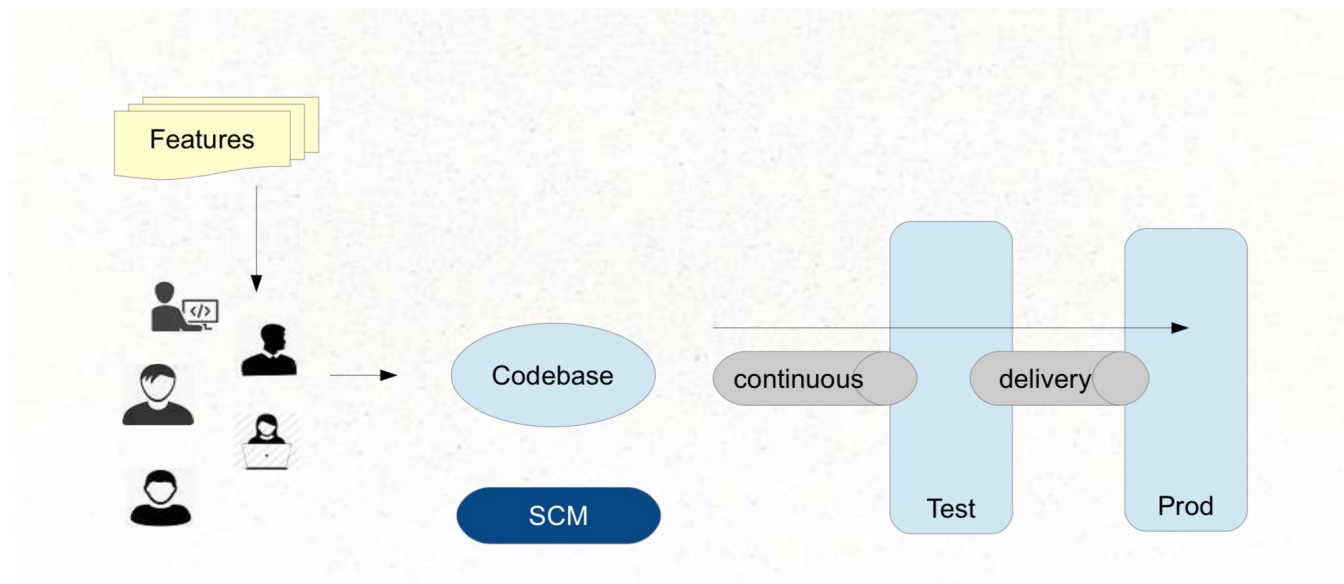
# Monolithic Challenges

New types of persistence / services



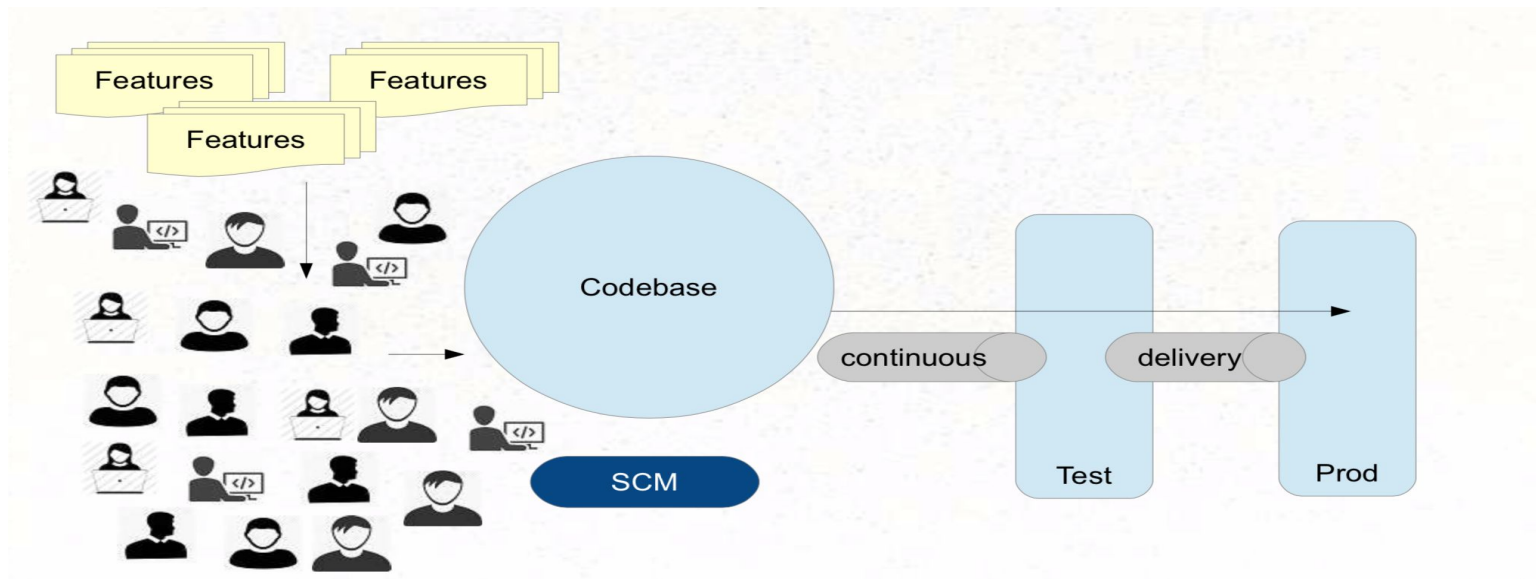
# Monolithic Challenges

Single Codebase, Deployment, Versioning, Team Size



# Monolithic Challenges

Using Teams / Language Constructs



# Understanding the 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

# Monolithic Drawbacks

- 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
    - Amazon's “2 Pizza” rule
- Deployment as single unit
  - Cannot independently deploy single change to single
  - component.
  - Changes are “held-hostage” by other changes

# Module Outline

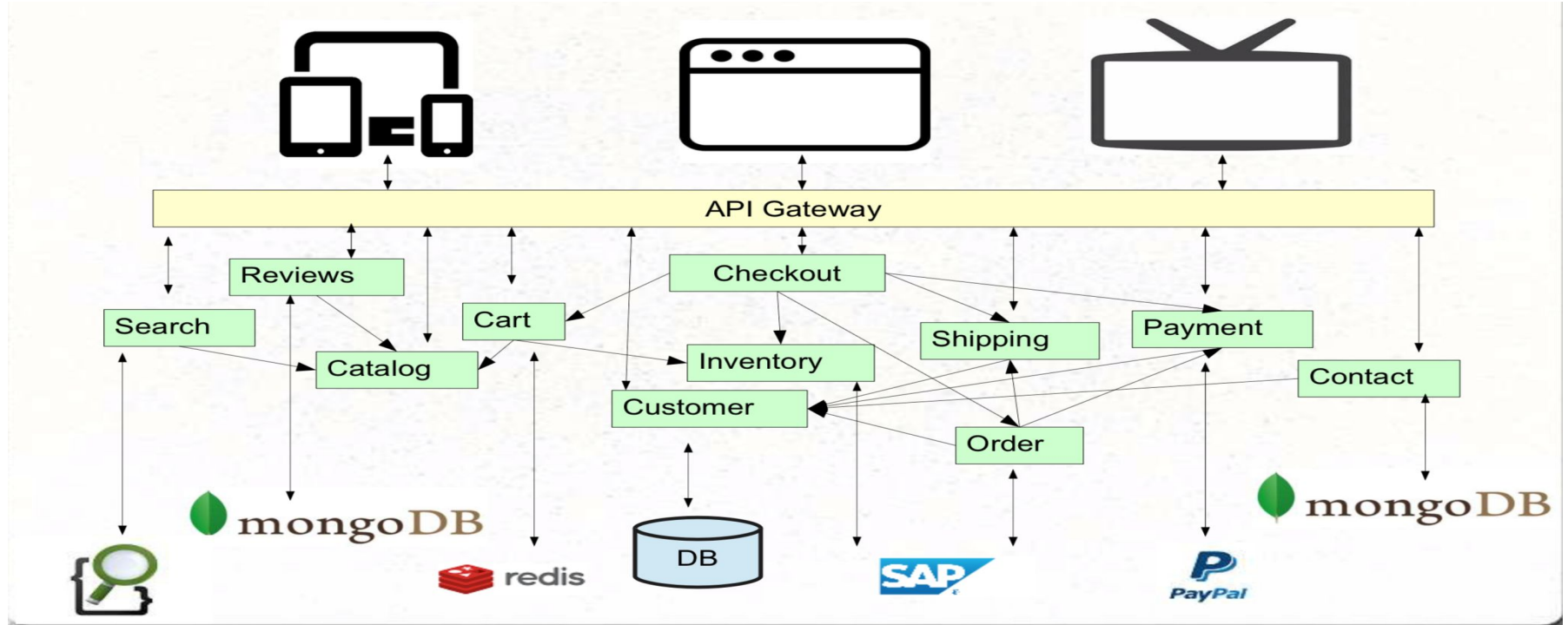
Defining Microservices

Microservices Explanation

- Understanding the Monolith
- **Understanding Microservices**

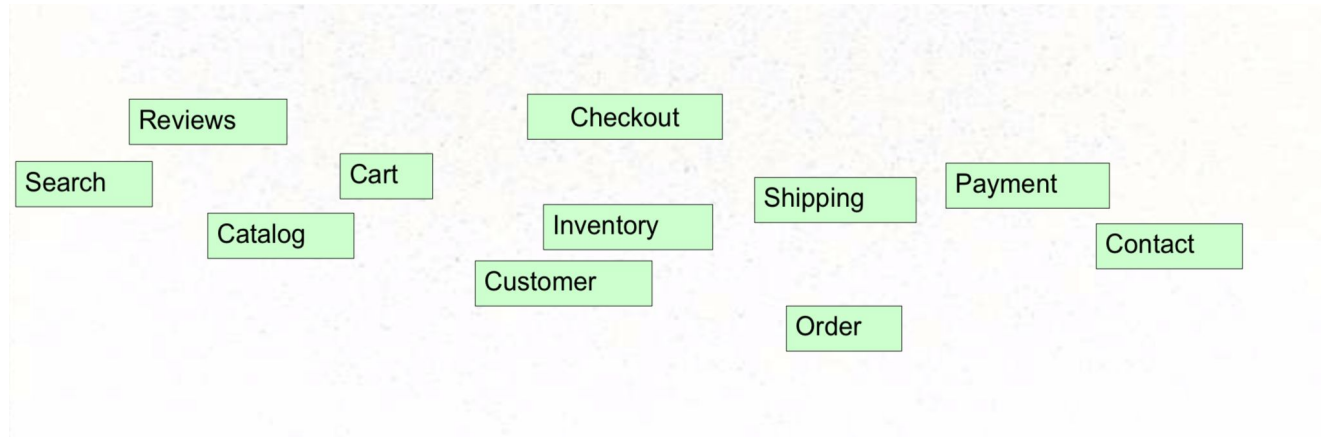
Practical Considerations

# Enter Microservices architecture



# Componentization via Services

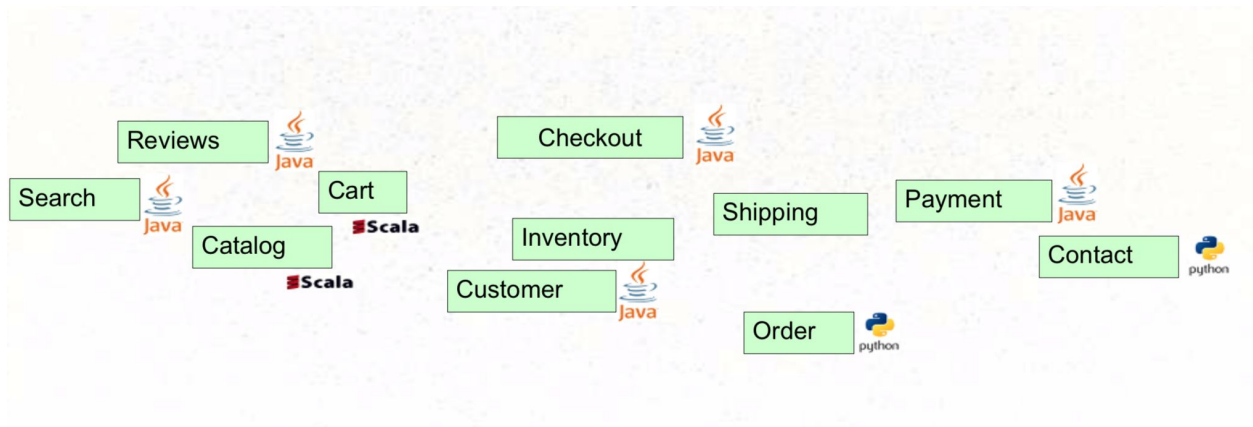
- NOT language constructs.
- Where services are small, independently deployable applications
- Forces the design of clear interfaces
- Changes scoped to their affected service



# Microservices:

Composed using suite of small services

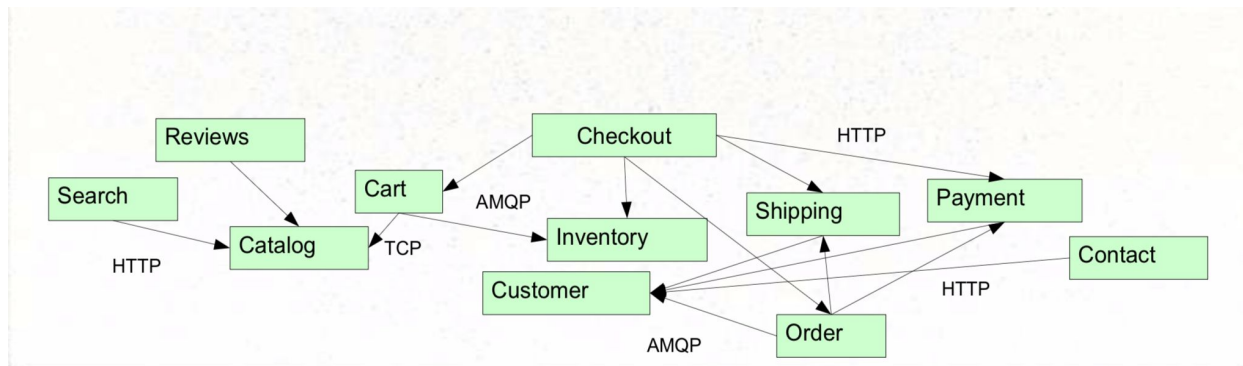
- Services are small, independently deployable applications
  - Not a single codebase
  - Not (necessarily) a single language / framework



# Microservices:

Communication based on lightweight protocols

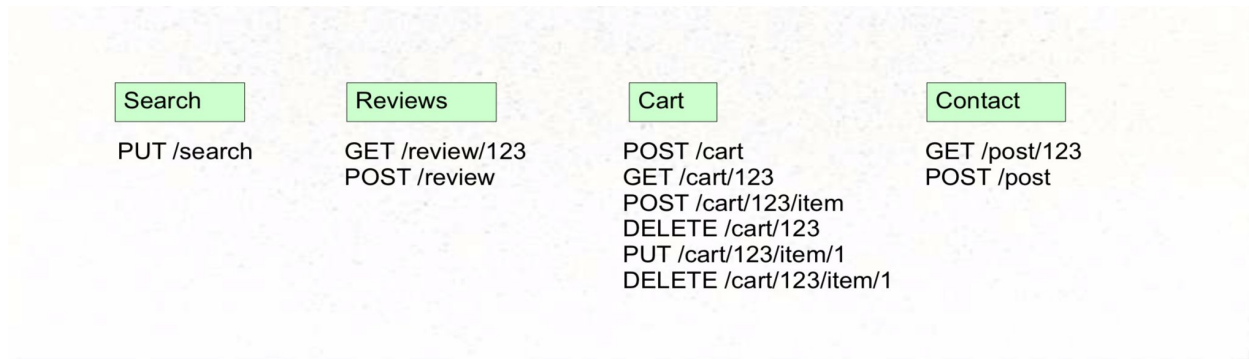
- HTTP, TCP, UDP, Messaging, etc.
  - Payloads: JSON, BSON, XML, Protocol Buffers, etc.
- Forces the design of clear interfaces
- Netflix's Cloud Native Architecture – Communicate via APIs
  - NOT Common Database



# Microservices:

**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

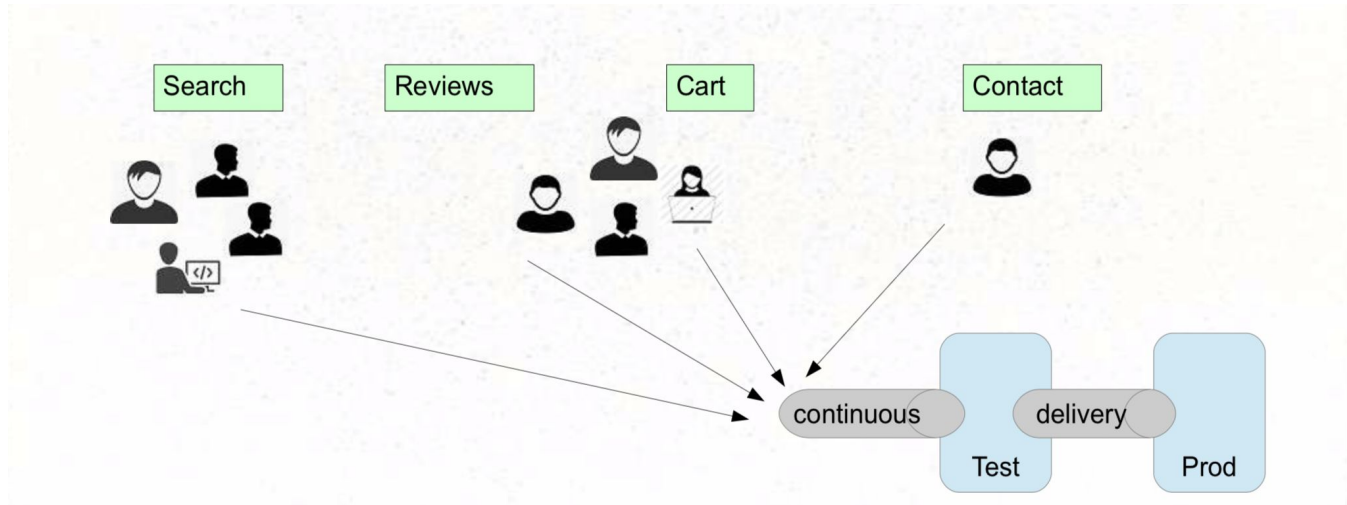




# Microservices:

## 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

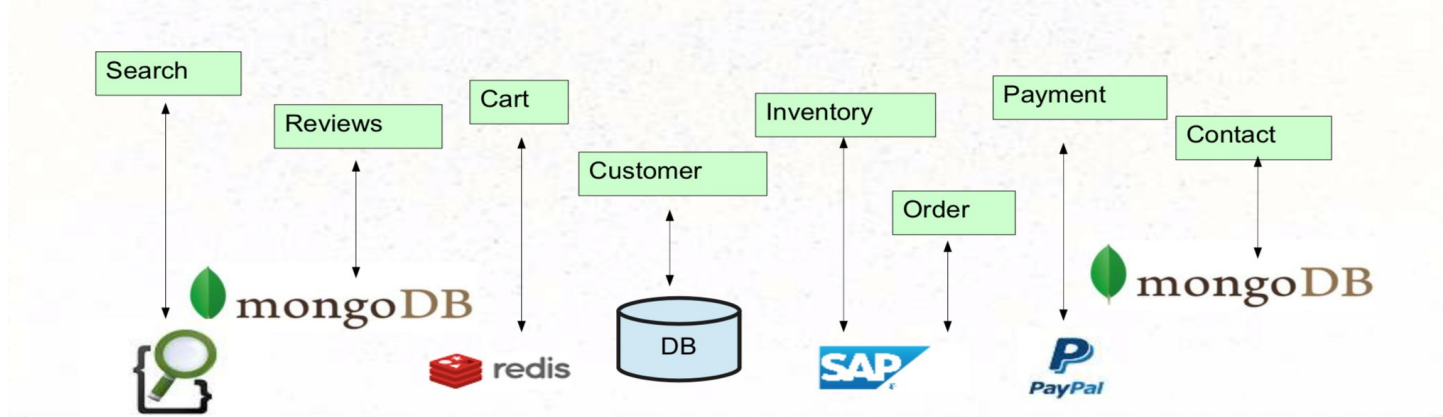
Services are not Orchestrated, but Choreographed

Services are not Orchestrated, but Choreographed..



# Polyglot Persistence

- Freedom to use the best technology for the job
  - Don't assume single RDBMS is always best
  - Very controversial! Many DBAs will not like this!
    - 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
  - Fallacies of Distributed Computing
- 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..

- 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
- Refactoring Module Boundaries

# 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 homogeneous.



# Module Outline

Defining Microservices

Microservices Explanation

- Understanding the Monolith
- Understanding Microservices

**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
  - 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
    - Social Security Act of 1935 – 63 pages
    - Affordable Care Act of 2010 – 906 pages
  - Dozens of secrets, not hundreds.
  - Predictable. Easy to experiment with

# Differences with SOA

- SOA addresses integration between systems.
  - Microservices address individual applications
- SOA relies on orchestration.
  - Microservices rely on choreography
- SOA relies on smart integration technology, dumb services
  - Microservices rely on smart services, dumb integration technology

# Are Monoliths Always Bad?

Consider etsy.com

As of February 2013: 1.49 billion page views, 4,215,169 items sold, \$94.7 million of goods sold, 22+ million members

150 developers deploy single WAR 60 times a day

Practices: CI; push button deployment; good monitoring; developers deploy to the site on the first day; VMs per developer; GitHub; Chef; IRC to control releases; dashboards; no source control branches.

# 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

<https://pivotal.io/microservices>

# The 12-Factor App

<https://12factor.net/>