Microservices Architecture

Introduction

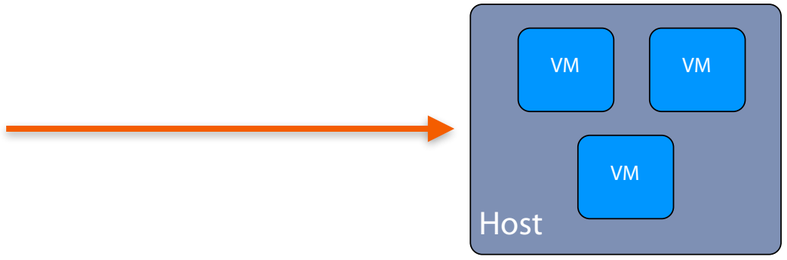
Rag Dhiman

ragcode.com | @RagDhiman

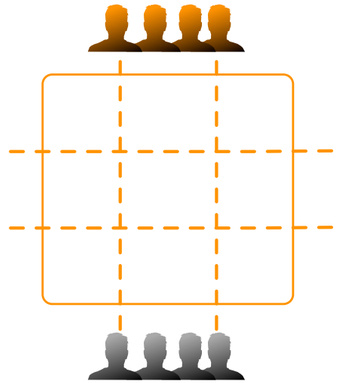
Course Overview

First Module
Introduction

Second Module
Design Approach

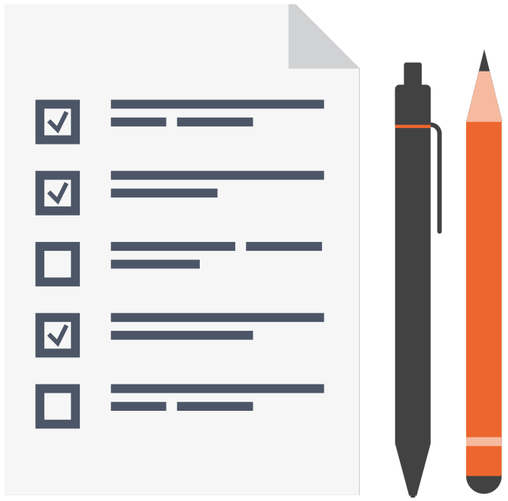


Third Module
Technology



Fourth Module
Moving Forwards

Module Overview



Microservices

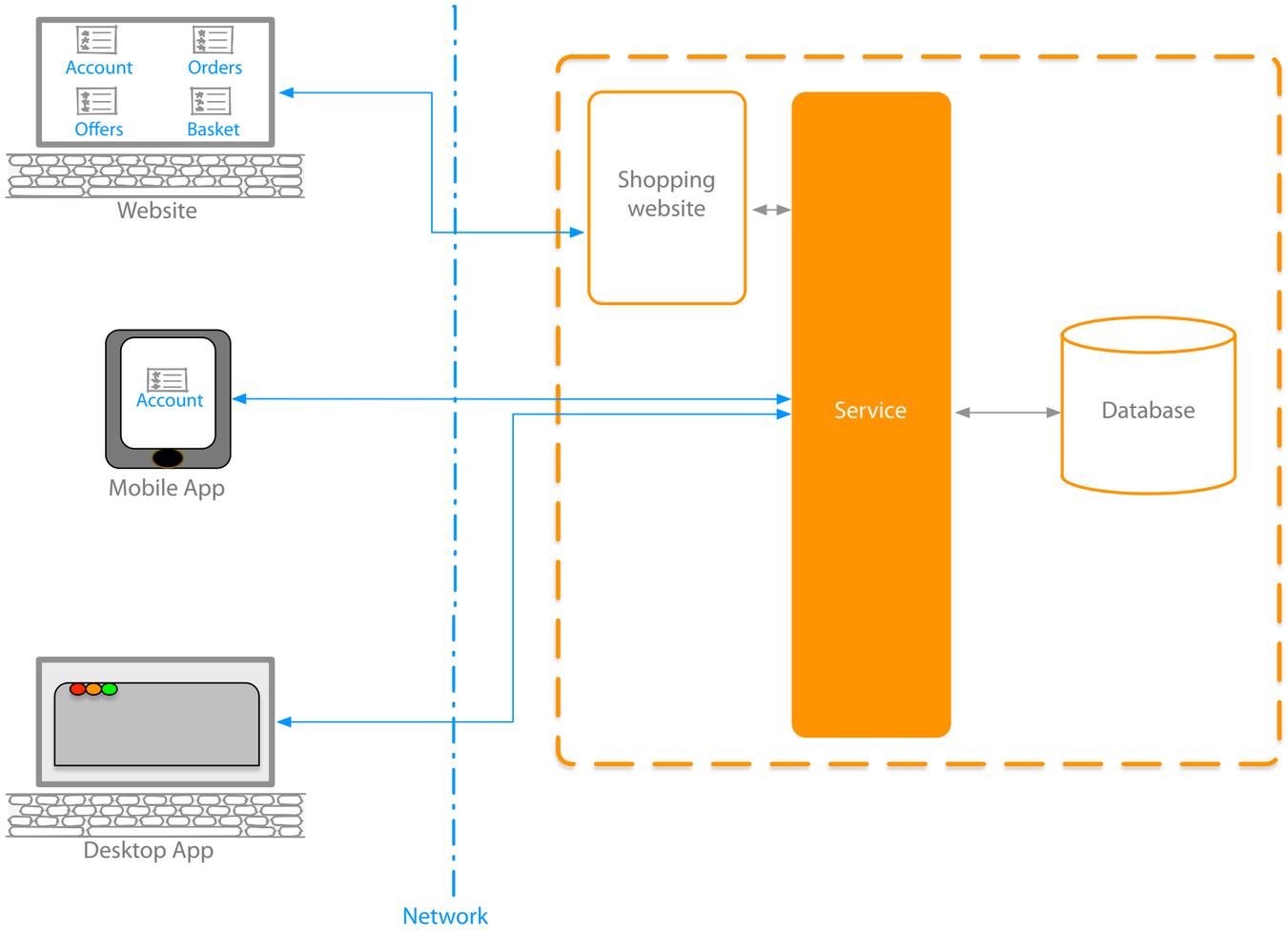
Emergence of Microservices

Microservices Design Principles

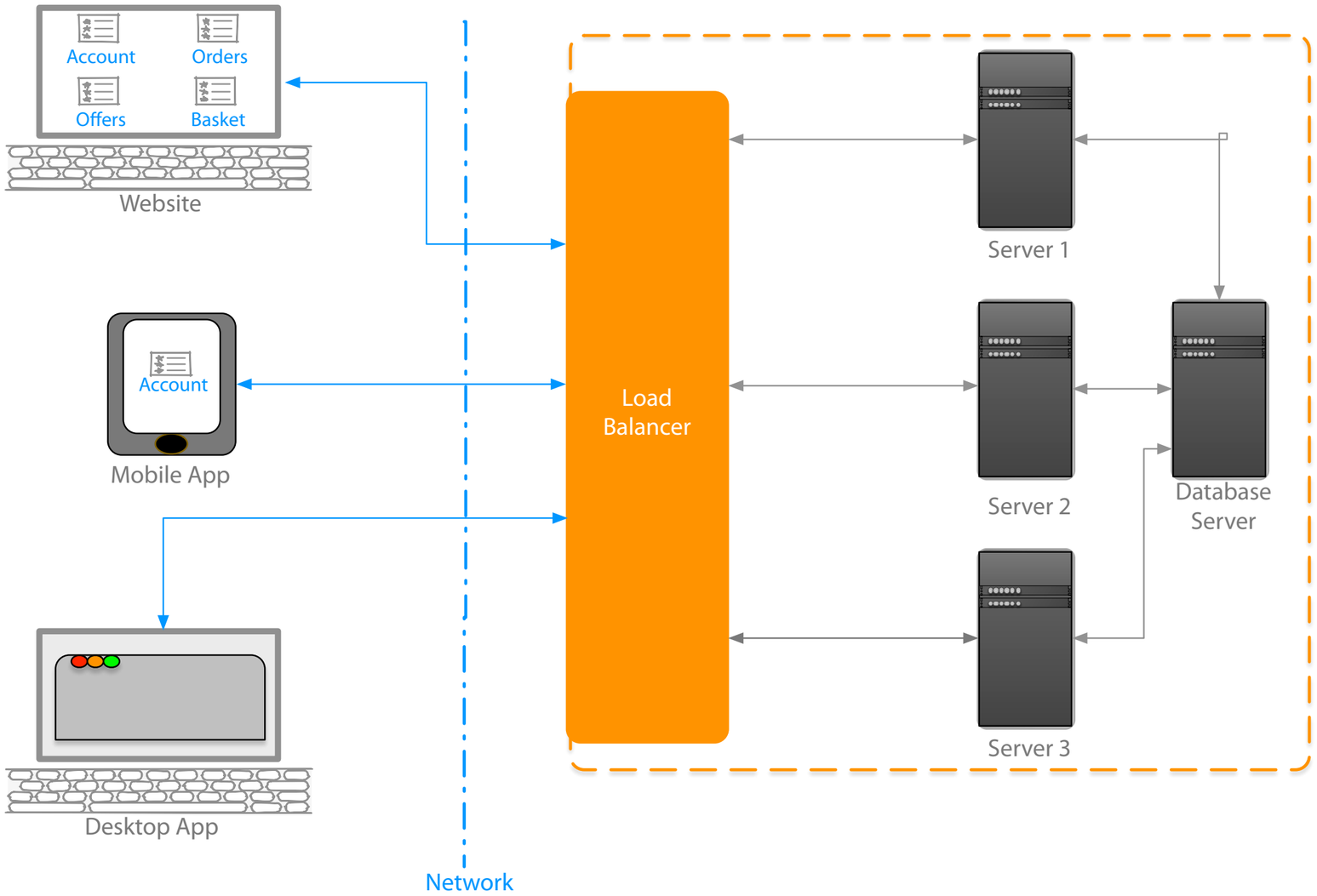
Microservices

What is a Service? | Introduction | The Monolithic

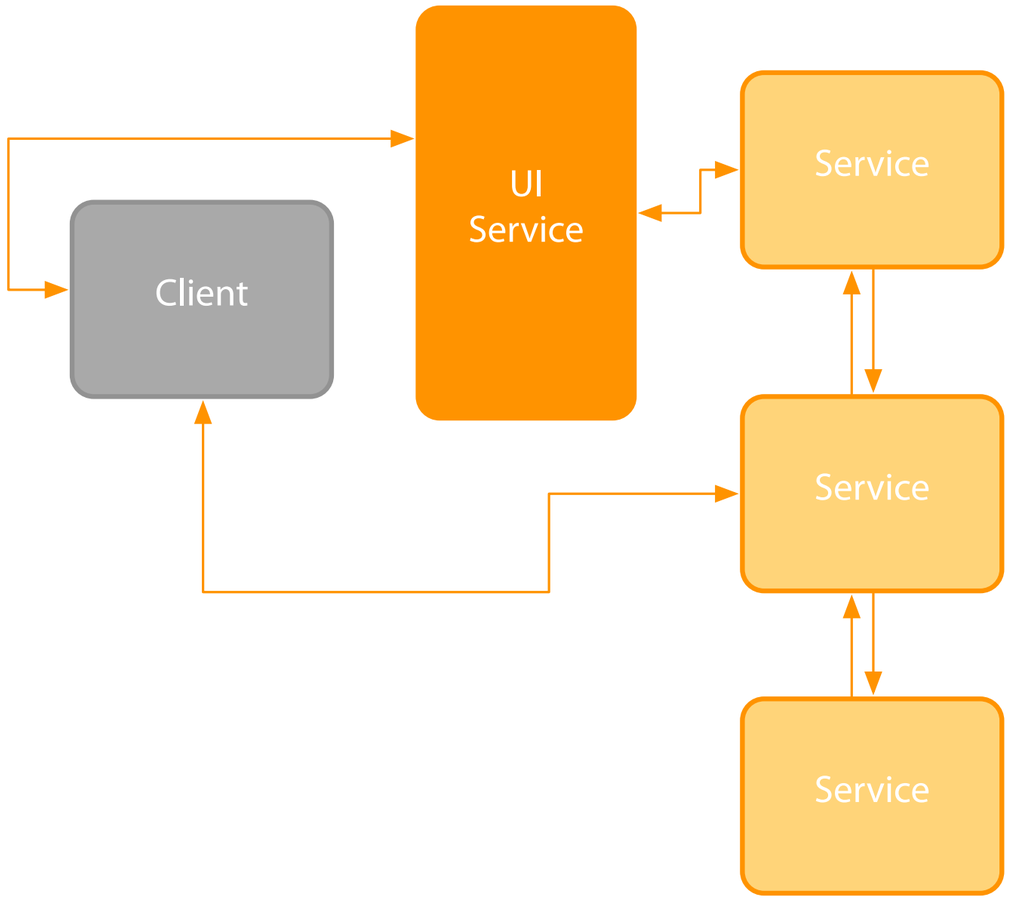
Microservices: What is a Service?



Microservices: What is a Service?

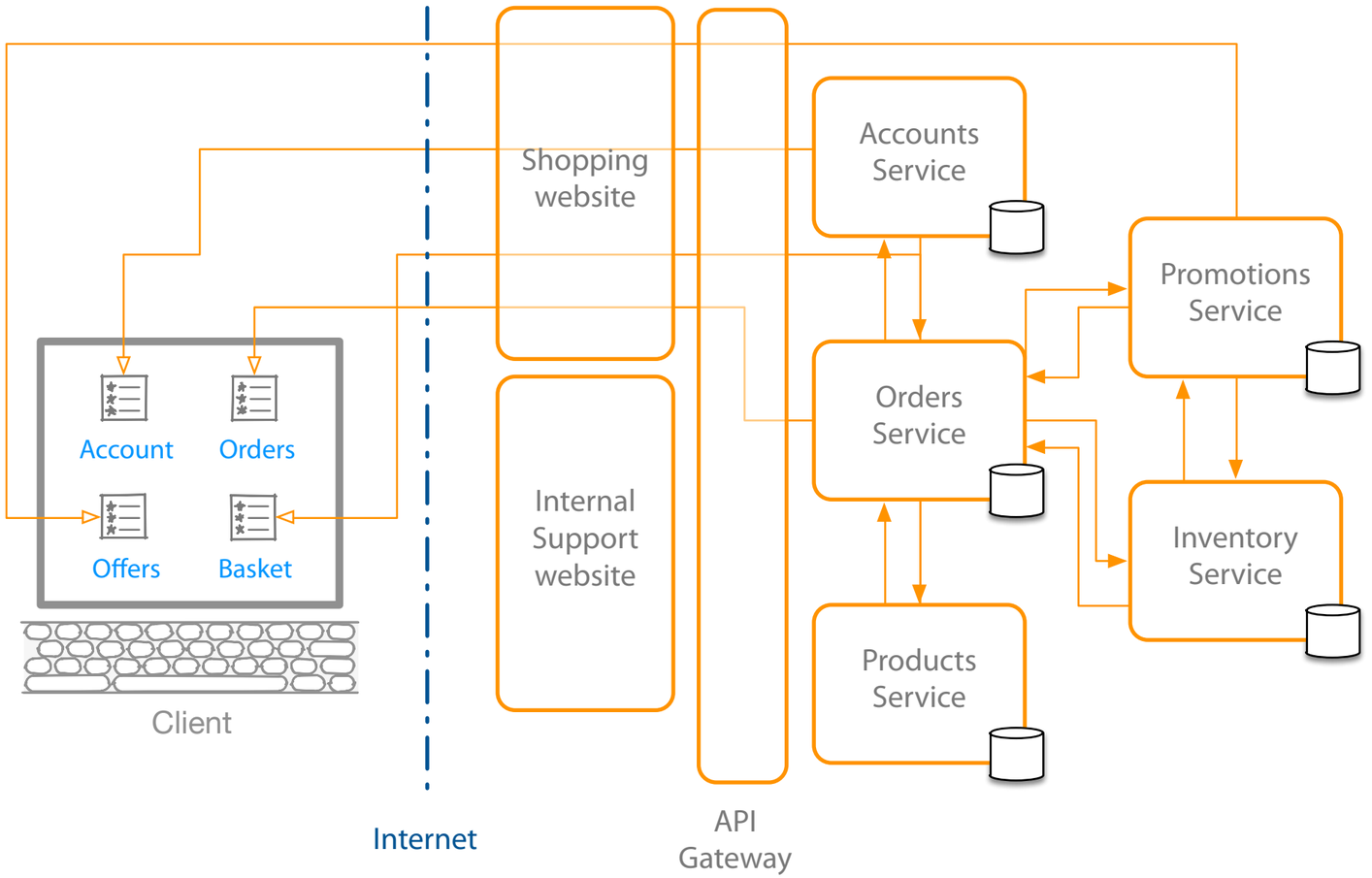


Microservices: Introduction



- SOA done well
  - Knowing how to size a service
  - Traditional SOA resulted in monolithic services
- Micro sized services provide
  - Efficiently scalable applications
  - Flexible applications
  - High performance applications
- Application(s) powered by multiple services
- Small service with a single focus
- Lightweight communication mechanism
  - Both client to service and service to service
- Technology agnostic API
- Independent data storage
- Independently changeable
- Independently deployable
- Distributed transactions
- Centralized tooling for management

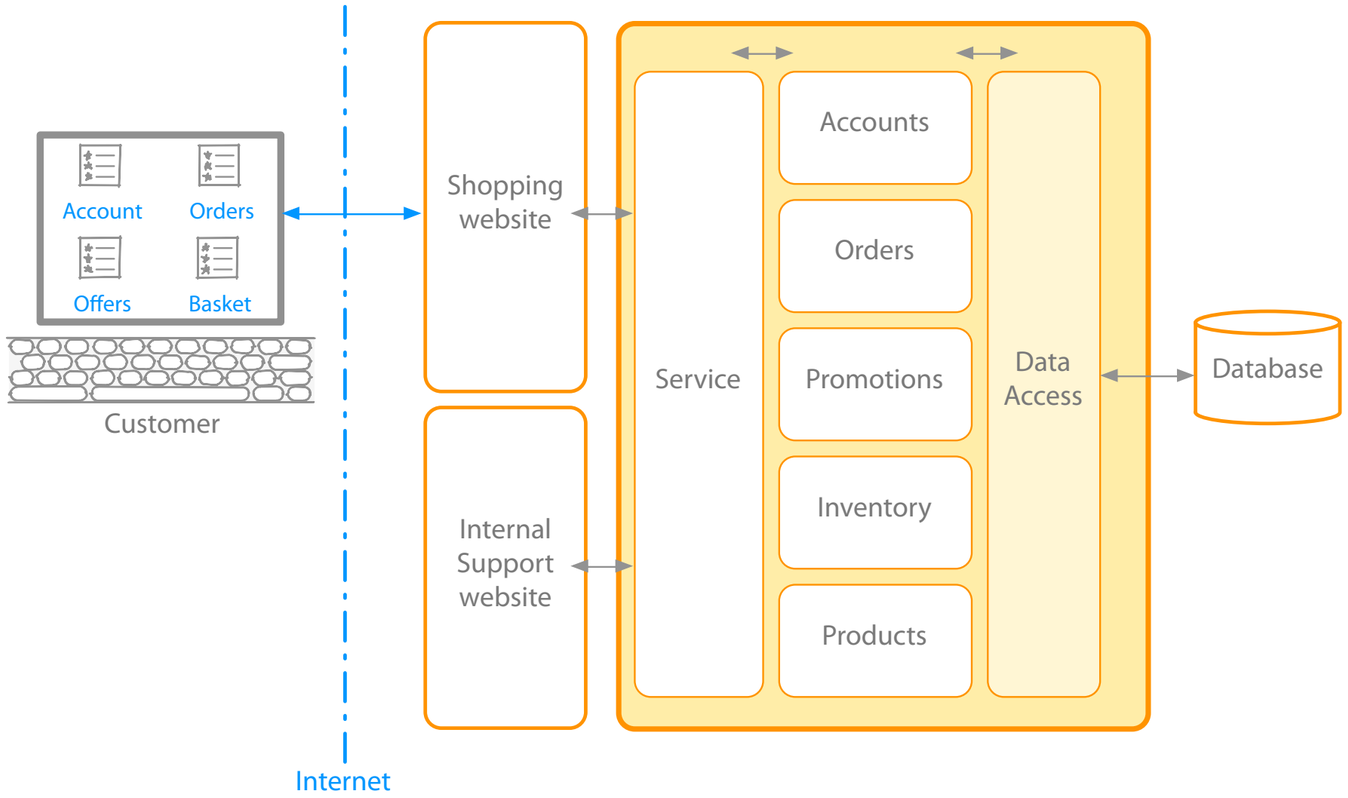
Microservices: Introduction



Microservices

What is a Service? | Introduction | The Monolithic

Microservices: The Monolithic



Typical enterprise application

No restriction on size

Large codebase

Longer development times

Challenging deployment

Inaccessible features

Fixed technology stack

High levels of coupling

- Between modules
- Between services

Failure could affect whole system

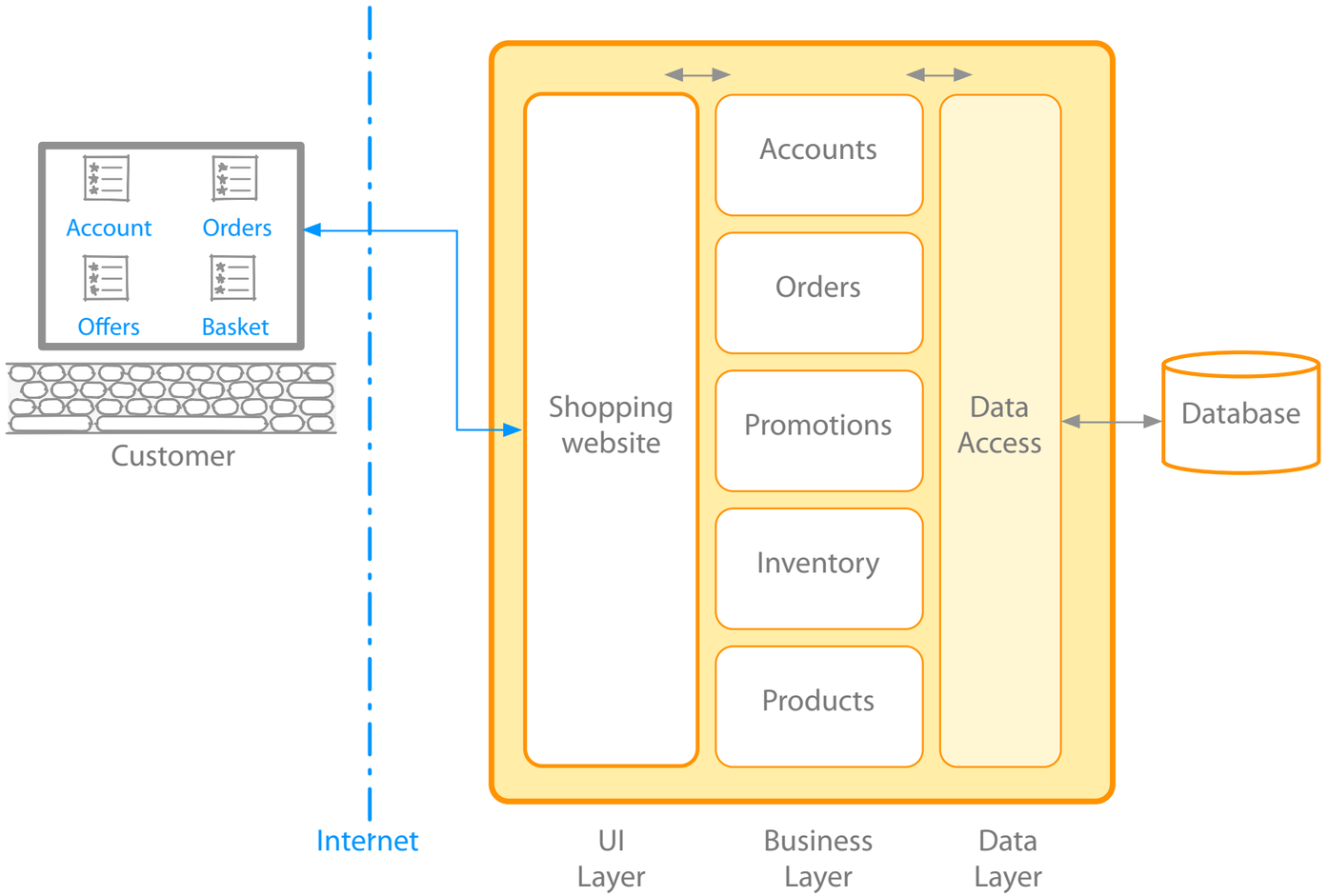
Scaling requires duplication of the whole

Single service on server

Minor change could result in complete rebuild

Easy to replicate environment

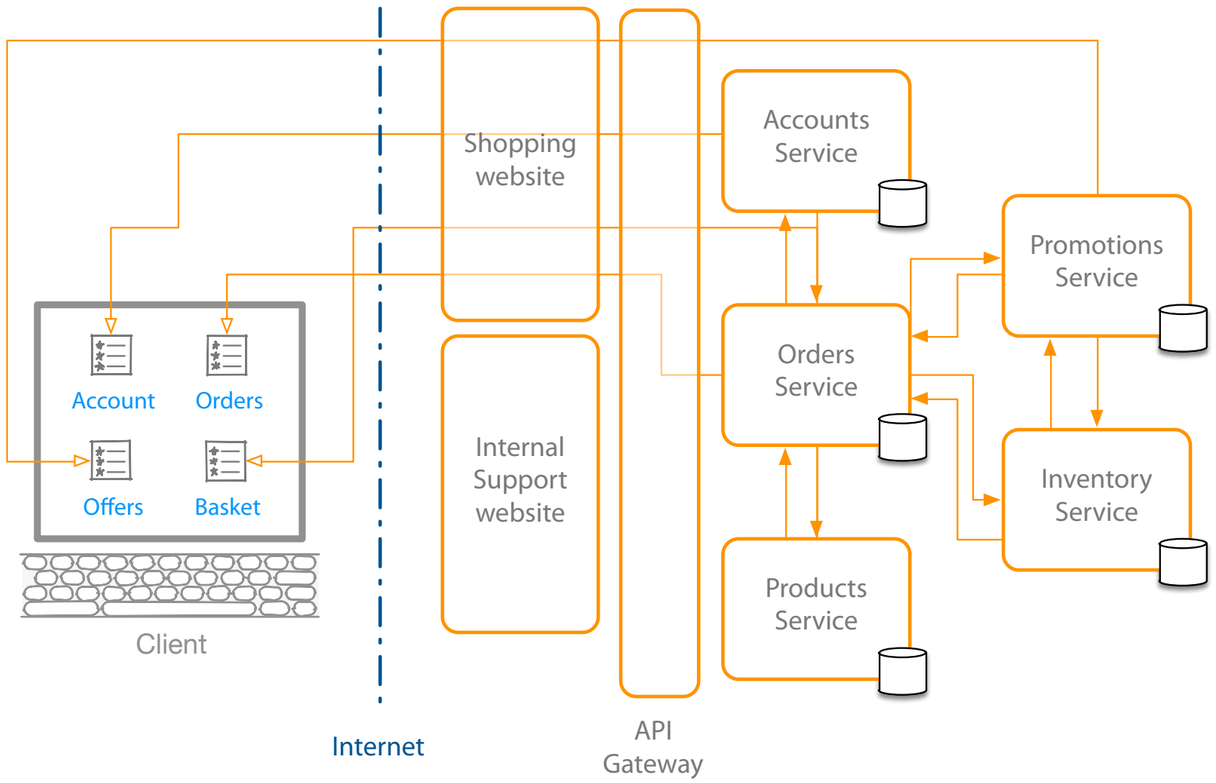
Microservices: The Monolithic



Emergence of Microservices

Why Now? | Benefits

Emergence of Microservices : Why Now?



Need to respond to change quickly

Need for reliability

Business domain-driven design

Automated test tools

Release and deployment tools

On-demand hosting technology

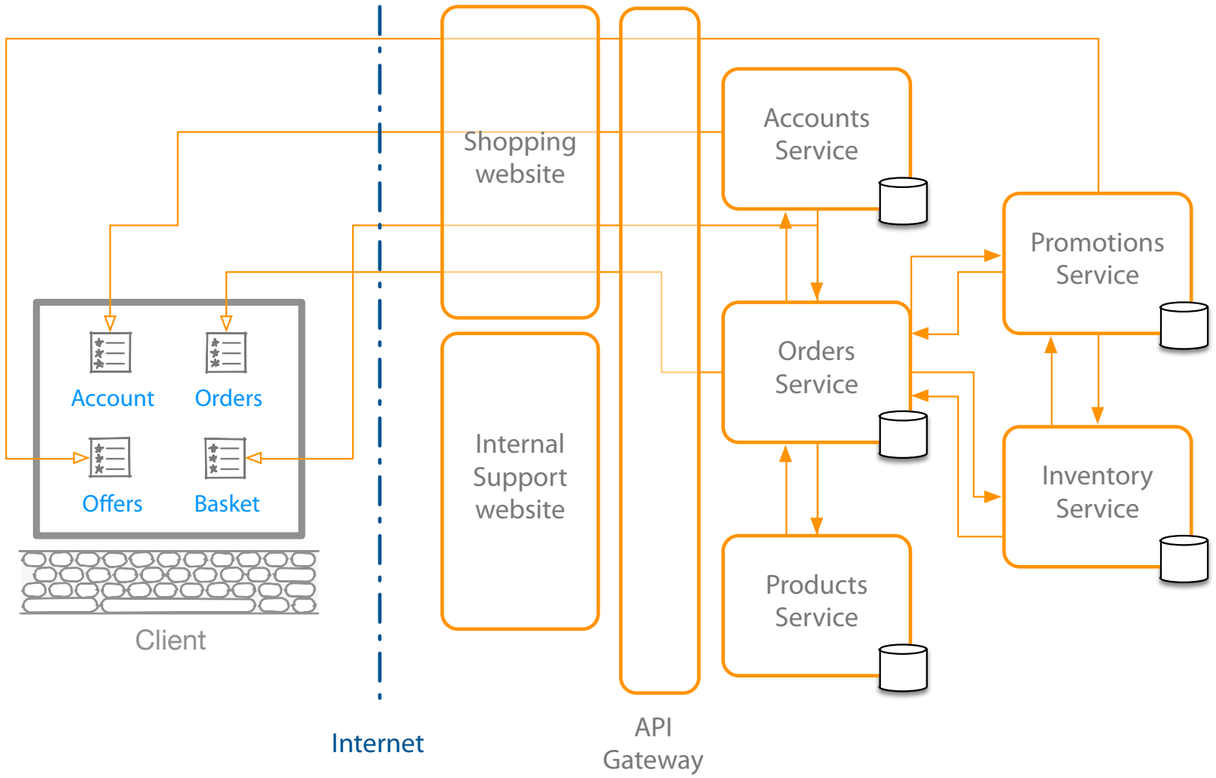
On-line cloud services

Need to embrace new technology

Asynchronous communication technology

Simpler server side and client side technology

Emergence of Microservices : Benefits



Shorter development times

Reliable and faster deployment

Enables frequent updates

Decouple the changeable parts

Security

Increased uptime

Fast issue resolution

Highly scalable and better performance

Better ownership and knowledge

Right technology

Enables distributed teams

Microservices Design Principles

Introduction | Principles | Summary

Microservices Design Principles: Introduction

High Cohesion

Autonomous

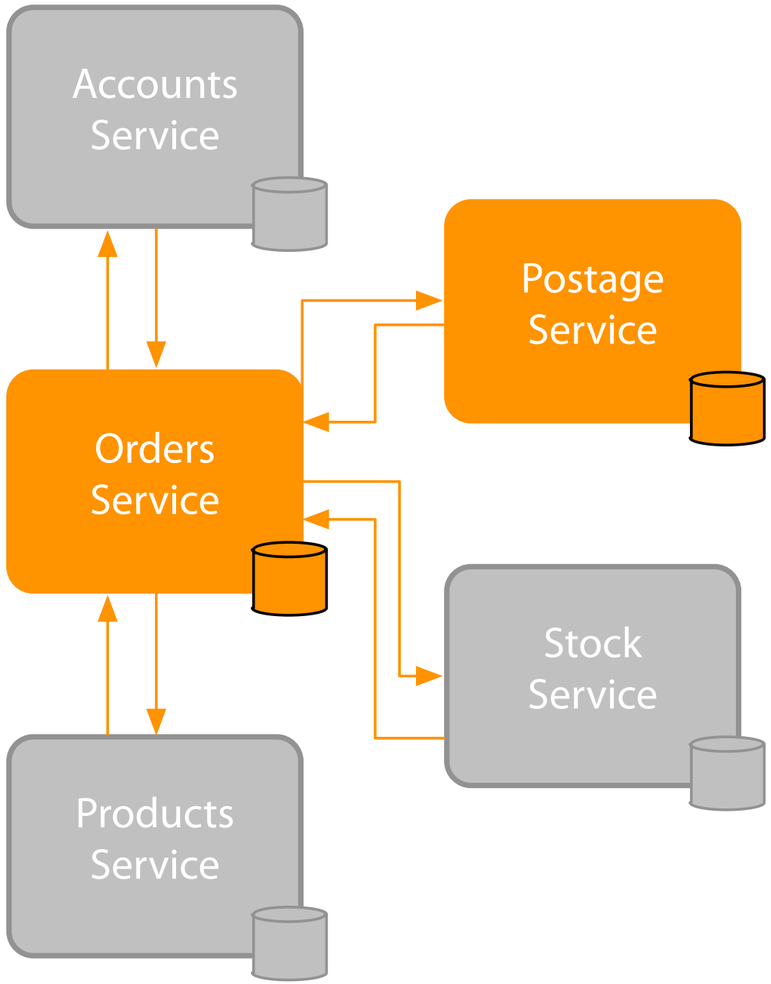
Business Domain Centric

Resilience

Observable

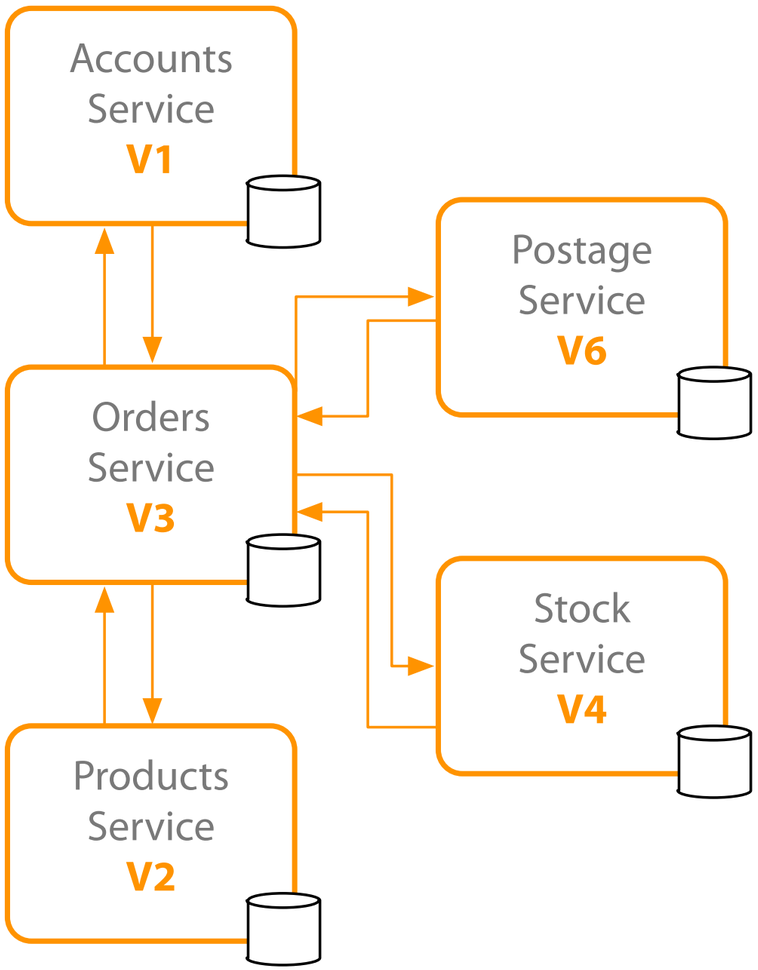
Automation

Microservices Design Principles: High Cohesion



- Single focus
- Single responsibility
  - SOLID principle
  - Only change for one reason
- Reason represents
  - A business function
  - A business domain
- Encapsulation principle
  - OOP principle
- Easily rewritable code
- Why
  - Scalability
  - Flexibility
  - Reliability

Microservices Design Principles: Autonomous



Loose coupling

Honor contracts and interfaces

Stateless

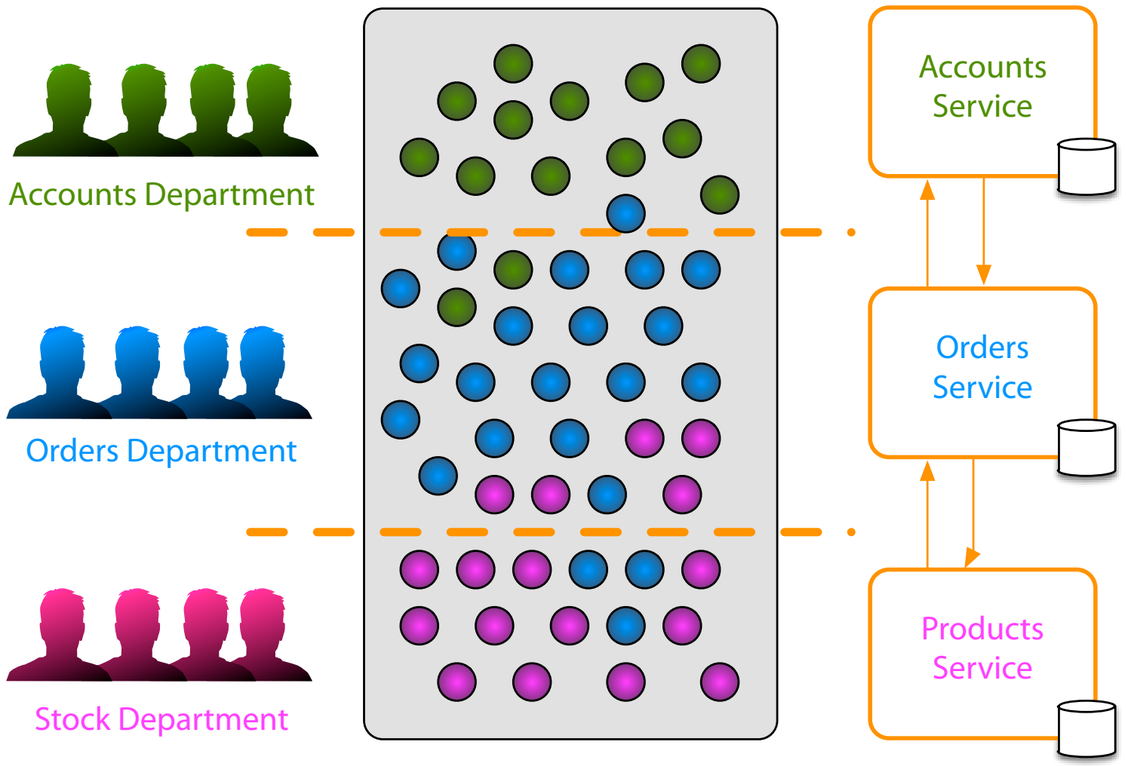
Independently changeable

Independently deployable

Backwards compatible

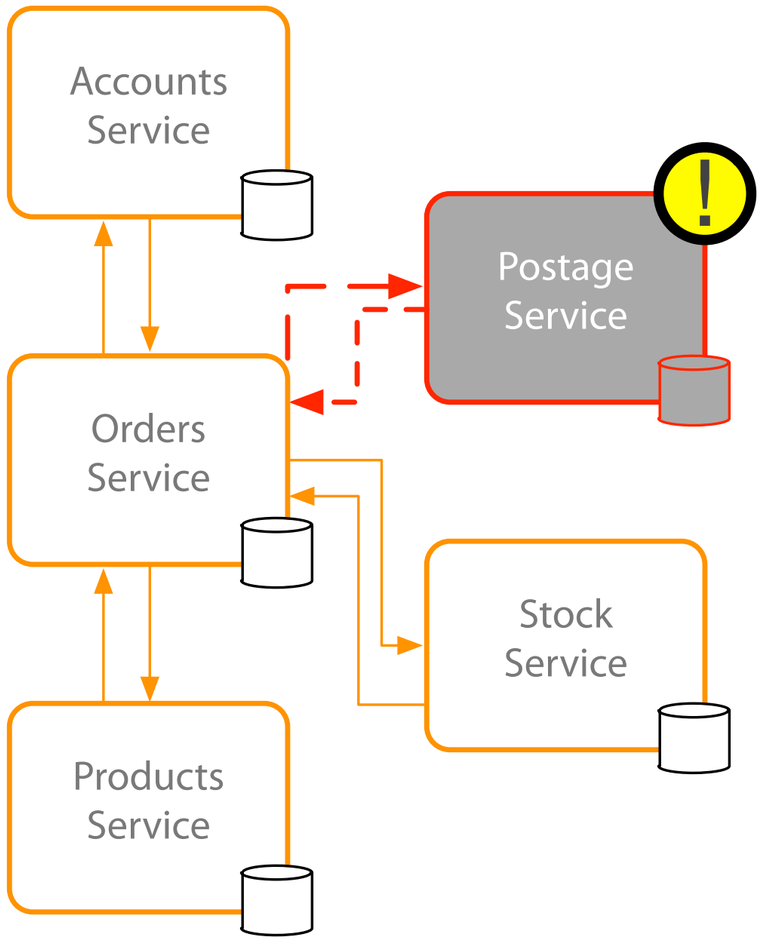
Concurrent development

Design Principles: Business Domain Centric



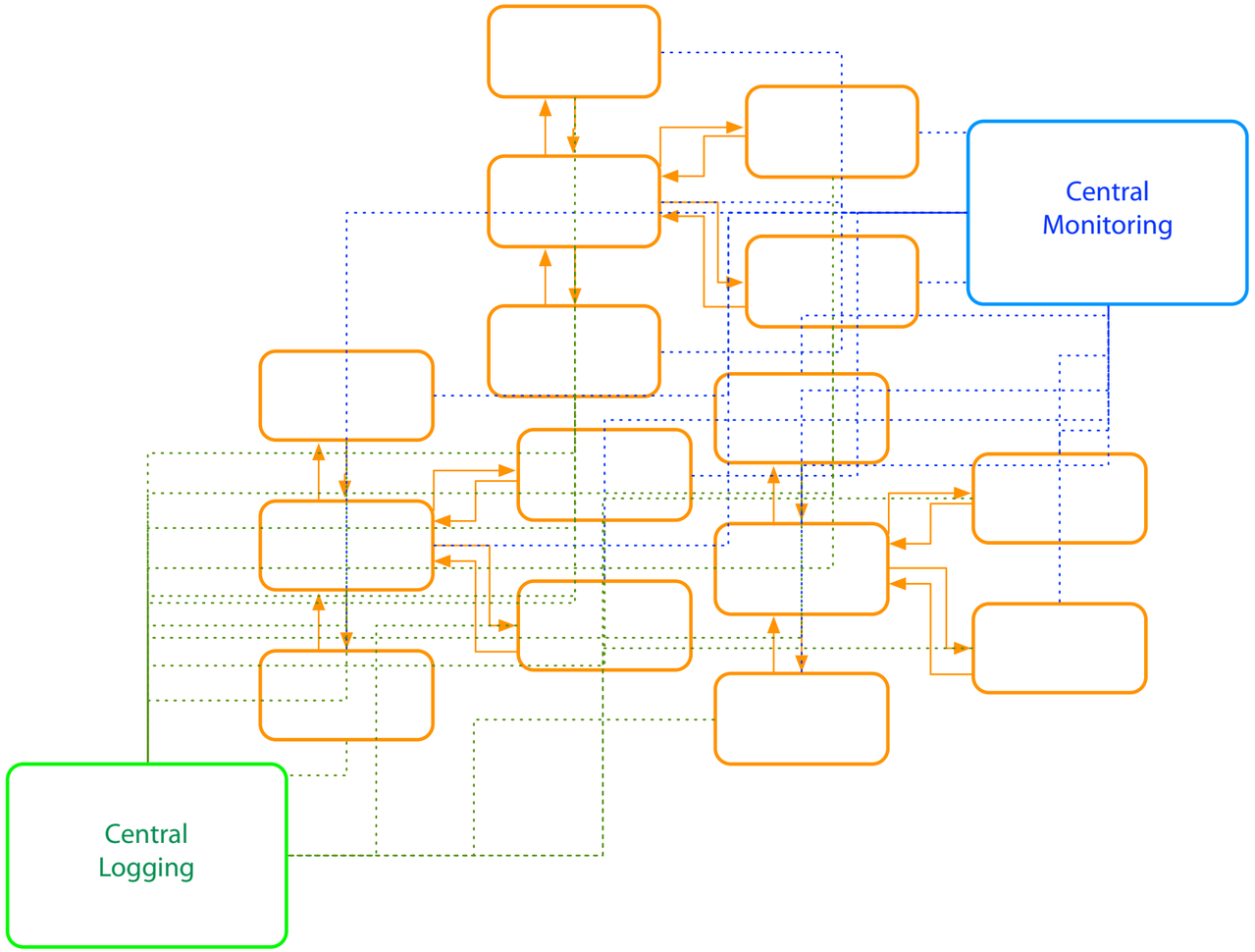
- Service represents business function
  - Accounts Department
  - Postage calculator
- Scope of service
- Bounded context from DDD
- Identify boundaries\seams
- Shuffle code if required
  - Group related code into a service
  - Aim for high cohesion
- Responsive to business change

Microservices Design Principles: Resilience



- Embrace failure
  - Another service
  - Specific connection
  - Third-party system
- Degrade functionality
- Default functionality
- Multiple instances
  - Register on startup
  - Deregister on failure
- Types of failure
  - Exceptions\Errors
  - Delays
  - Unavailability
- Network issues
  - Delay
  - Unavailability
- Validate input
  - Service to service
  - Client to service

Microservices Design Principles: Observable



System Health

- Status
- Logs
- Errors

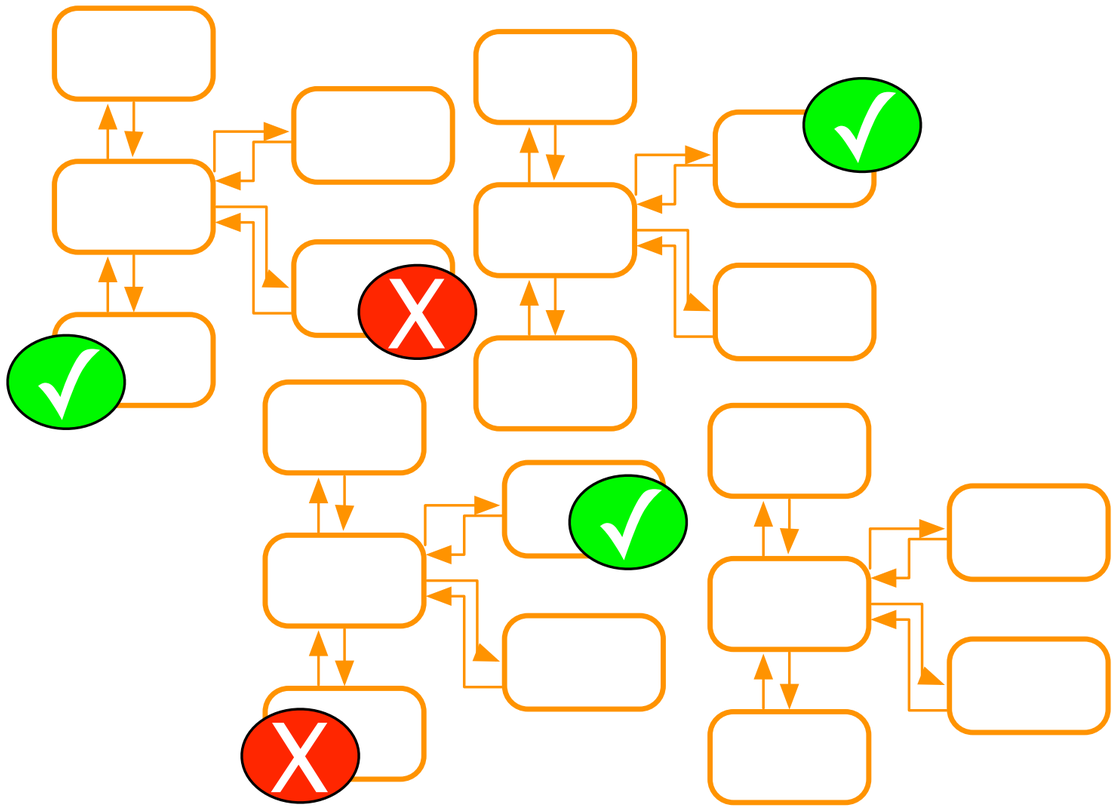
Centralized monitoring

Centralized logging

Why

- Distributed transactions
- Quick problem solving
- Quick deployment requires feedback
- Data used for capacity planning
- Data used for scaling
- Whats actually used
- Monitor business data

Microservices Design Principles: Automation



Tools to reduce testing

- Manual regression testing
- Time taken on testing integration
- Environment setup for testing

Tools to provide quick feedback

- Integration feedback on check in
- Continous Integration

Tools to provide quick deployment

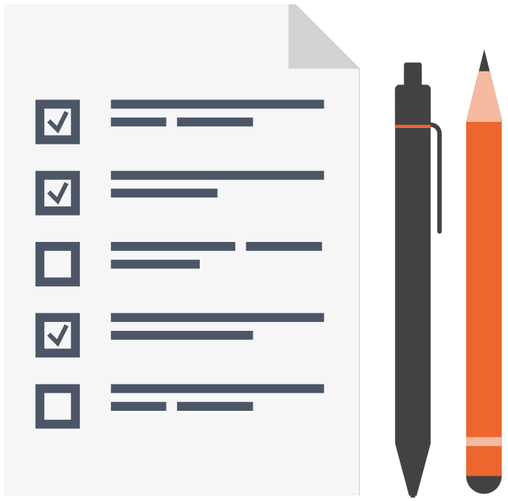
- Pipeline to deployment
- Deployment ready status
- Automated deployment

- Reliable deployment
- Continuous Deployment

Why

- Distributed system
- Multiple instances of services
- Manual integration testing too time consuming
- Manual deployment time consuming and unreliable

Module Summary



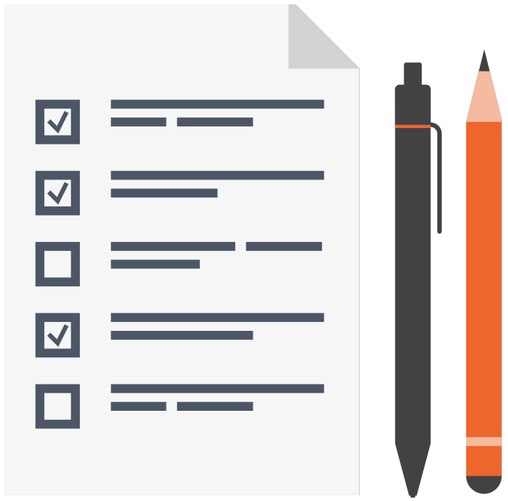
- Microservices
  - Service
  - Introduction
  - The Monolithic
- Emergence of Microservices
  - Why Now?
  - Benefits
- Microservices Design Principles
  - High Cohesion
  - Autonomous
  - Business Domain Centric
  - Resilience
  - Observable
  - Automation

Microservices Design

Rag Dhiman

ragcode.com | @RagDhiman

Module Overview



Microservices Design

- Principles
- Approach

Microservices Design: Principles

High Cohesion

Single thing done well

Single focus

Autonomous

Independently changeable

Independently deployable

Business Domain Centric

Represent business function

or represent a business domain

Resilience

Embrace Failure

Default or degrade functionality

Observable

See system health

Centralized logging and monitoring

Automation

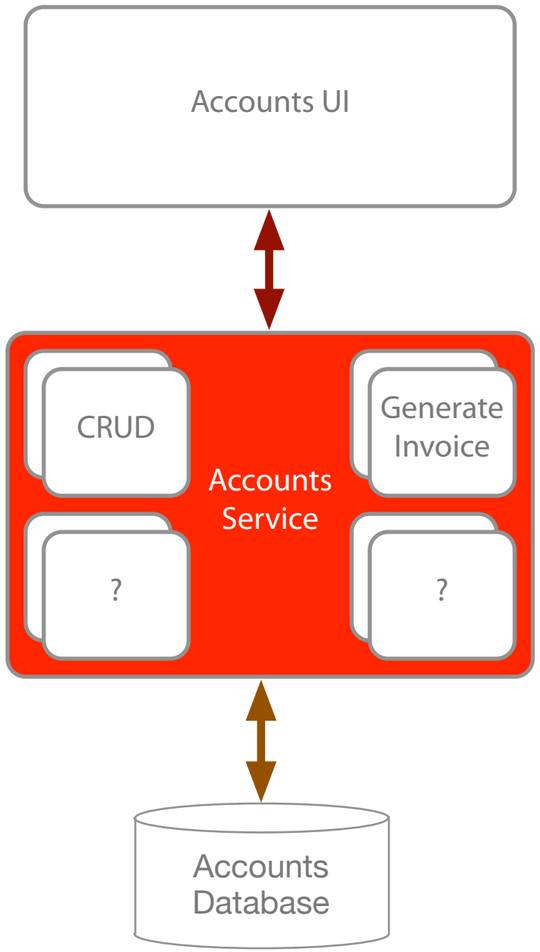
Tools for testing and feedback

Tools for deployment

Microservices Design

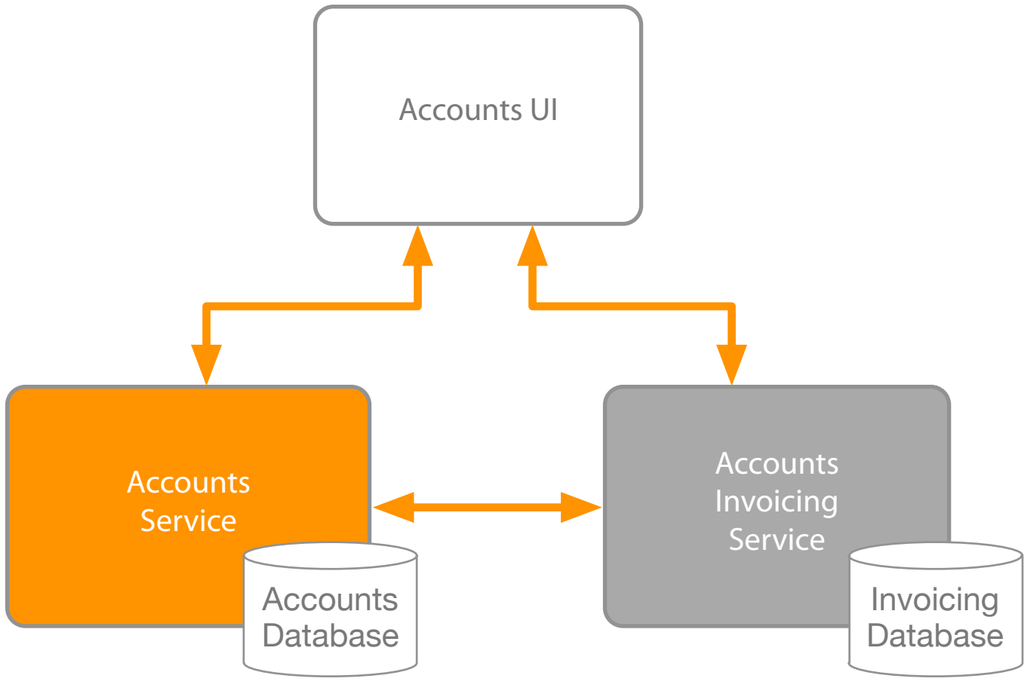
Principles | Approach

Approach: High Cohesion



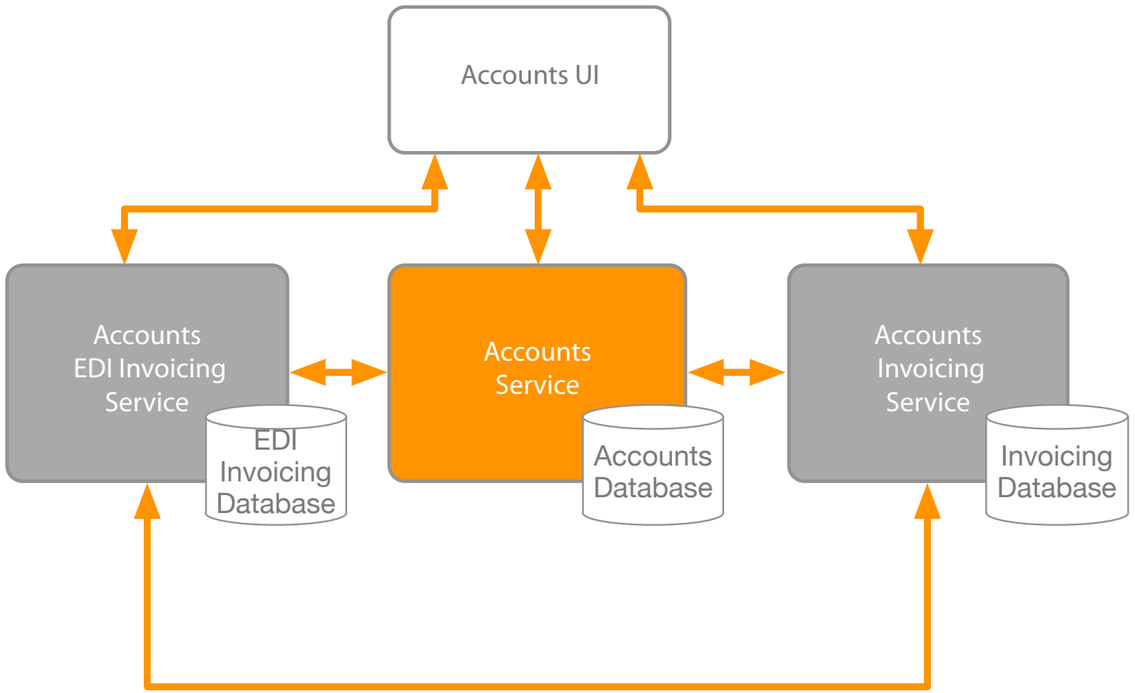
- Identify a single focus
  - Business function
  - Business domain
- Split into finer grained services
- Avoid “Is kind of the same”
- Don't get lazy!
- Don’t be afraid to create many services
- Question in code\peer reviews
  - Can this change for more than one reason

Approach: High Cohesion



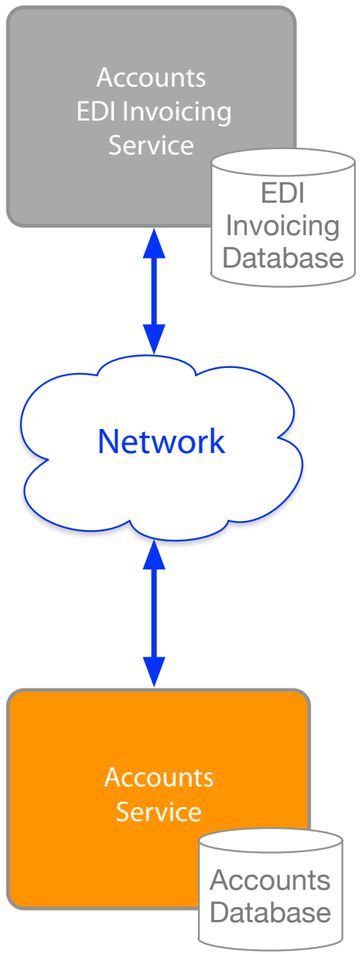
- Identify a single focus
  - Business function
  - Business domain
- Split into finer grained services
- Avoid “Is kind of the same”
- Don't get lazy!
- Don’t be afraid to create many services
- Question in code\peer reviews
  - Can this change for more than one reason

Approach: High Cohesion



- Identify a single focus
  - Business function
  - Business domain
- Split into finer grained services
- Avoid “Is kind of the same”
- Don't get lazy!
- Don’t be afraid to create many services
- Question in code\peer reviews
  - Can this change for more than one reason

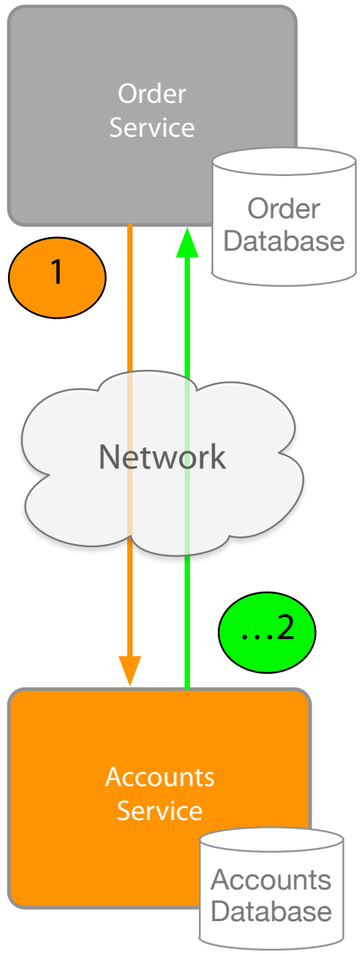
Approach: Autonomous



Loosely coupled

- Communication by network
  - Synchronous
  - Asynchronous
    - Publish events
    - Subscribe to events
- Technology agnostic API
- Avoid client libraries
- Contracts between services
  - Fixed and agreed interfaces
  - Shared models
  - Clear input and output
- Avoid chatty exchanges between services
- Avoid sharing between services
  - Databases
  - Shared libraries

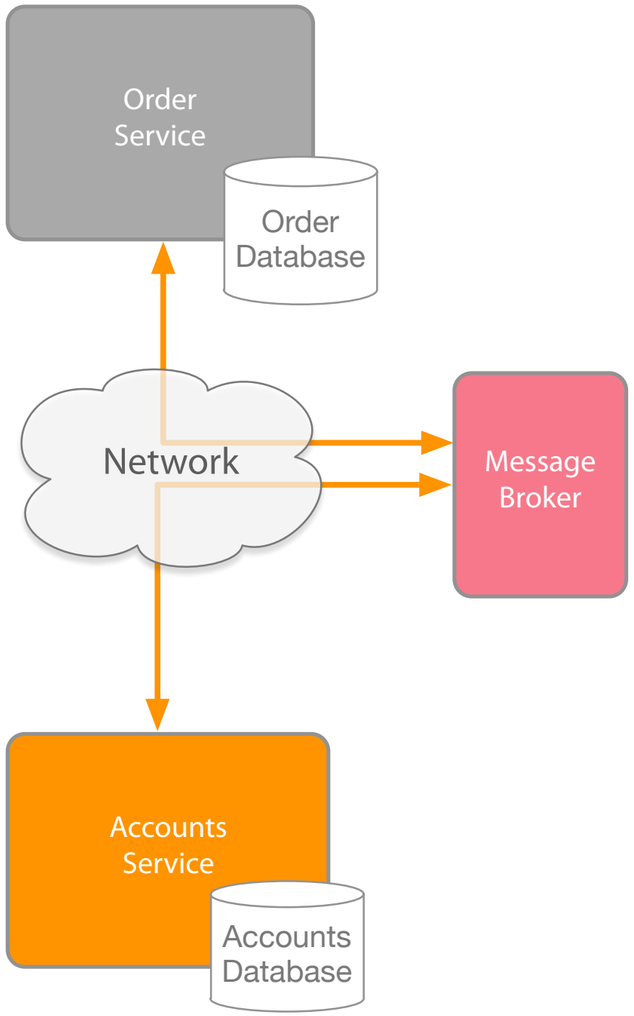
Approach: Autonomous



Loosely coupled

- Communication by network
  - Synchronous
  - Asynchronous
    - Publish events
    - Subscribe to events
- Technology agnostic API
- Avoid client libraries
- Contracts between services
  - Fixed and agreed interfaces
  - Shared models
  - Clear input and output
- Avoid chatty exchanges between services
- Avoid sharing between services
  - Databases
  - Shared libraries

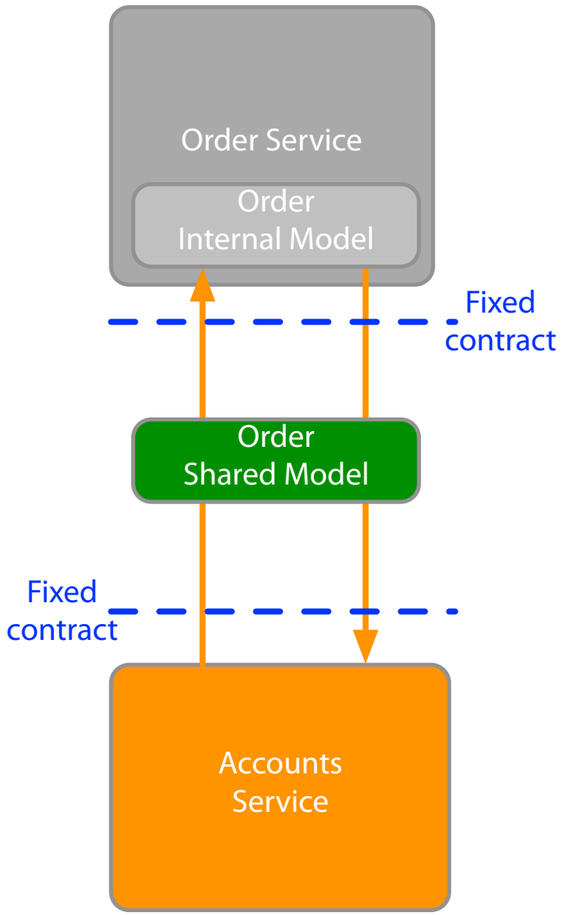
Approach: Autonomous



Loosely coupled

- Communication by network
  - Synchronous
  - Asynchronous
    - Publish events
    - Subscribe to events
- Technology agnostic API
- Avoid client libraries
- Contracts between services
  - Fixed and agreed interfaces
  - Shared models
  - Clear input and output
- Avoid chatty exchanges between services
- Avoid sharing between services
  - Databases
  - Shared libraries

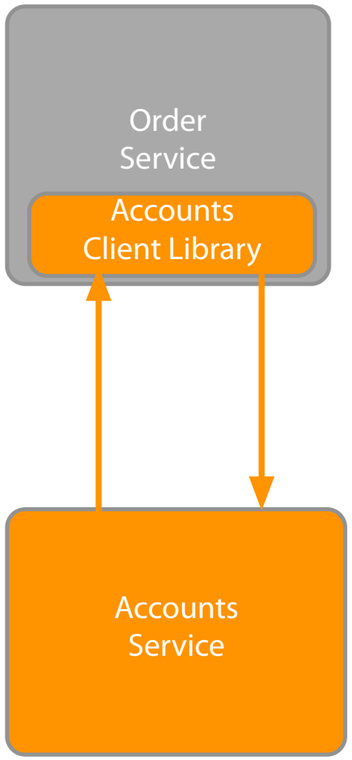
Approach: Autonomous



Loosely coupled

- Communication by network
  - Synchronous
  - Asynchronous
    - Publish events
    - Subscribe to events
- Technology agnostic API
- Avoid client libraries
- Contracts between services
  - Fixed and agreed interfaces
  - Shared models
  - Clear input and output
- Avoid chatty exchanges between services
- Avoid sharing between services
  - Databases
  - Shared libraries

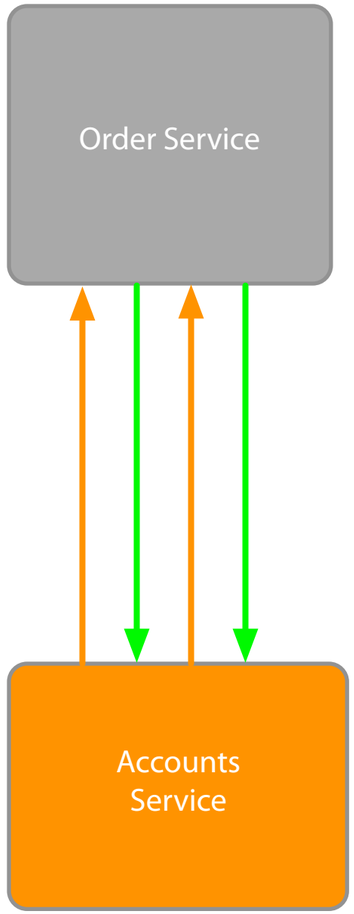
Approach: Autonomous



Loosely coupled

- Communication by network
  - Synchronous
  - Asynchronous
    - Publish events
    - Subscribe to events
- Technology agnostic API
- Avoid client libraries
- Contracts between services
  - Fixed and agreed interfaces
  - Shared models
  - Clear input and output
- Avoid chatty exchanges between services
- Avoid sharing between services
  - Databases
  - Shared libraries

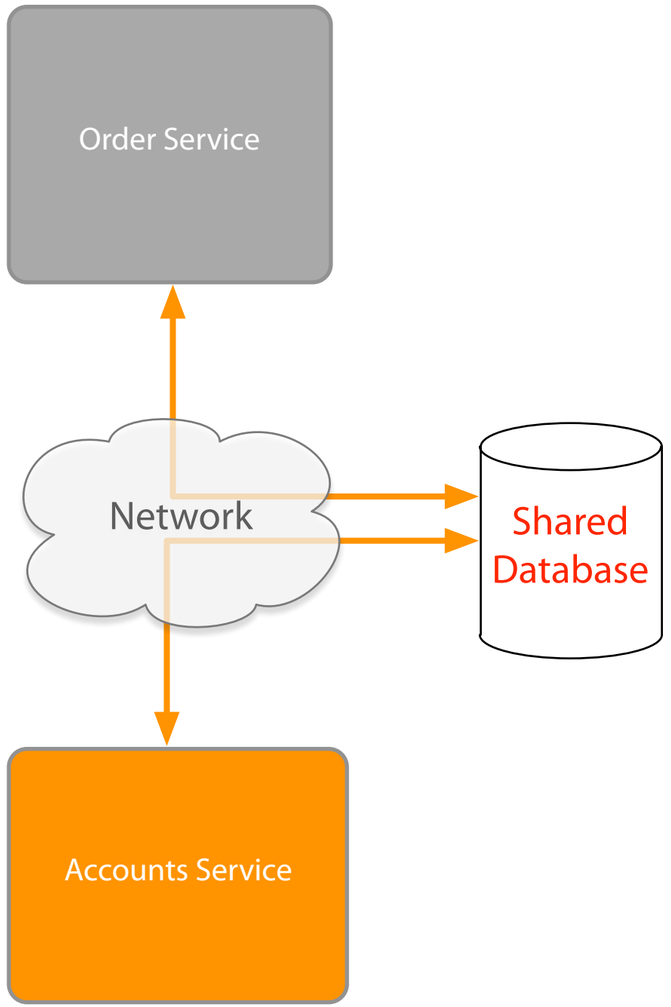
Approach: Autonomous



Loosely coupled

- Communication by network
  - Synchronous
  - Asynchronous
    - Publish events
    - Subscribe to events
- Technology agnostic API
- Avoid client libraries
- Contracts between services
  - Fixed and agreed interfaces
  - Shared models
  - Clear input and output
- Avoid chatty exchanges between services
- Avoid sharing between services
  - Databases
  - Shared libraries

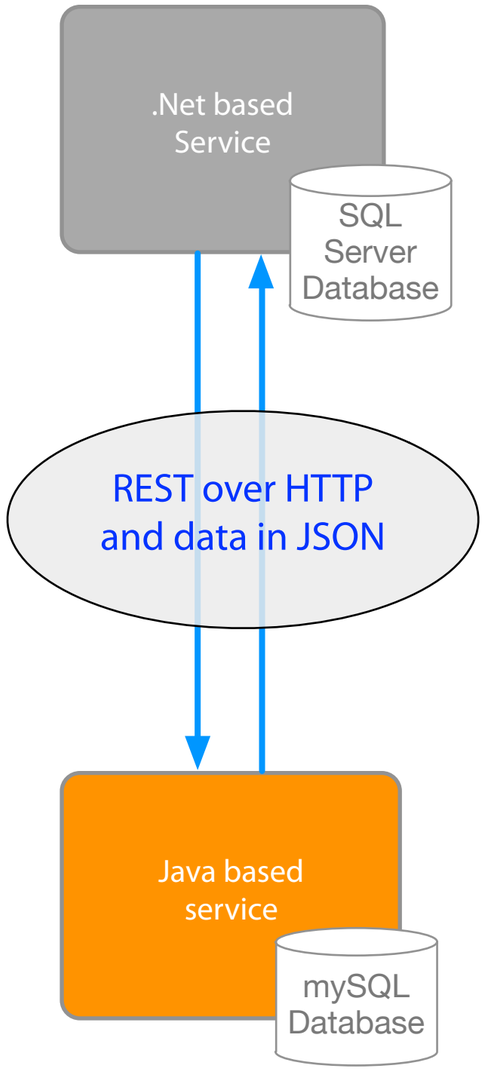
Approach: Autonomous



Loosely coupled

- Communication by network
  - Synchronous
  - Asynchronous
    - Publish events
    - Subscribe to events
- Technology agnostic API
- Avoid client libraries
- Contracts between services
  - Fixed and agreed interfaces
  - Shared models
  - Clear input and output
- Avoid chatty exchanges between services
- Avoid sharing between services
  - Databases
  - Shared libraries

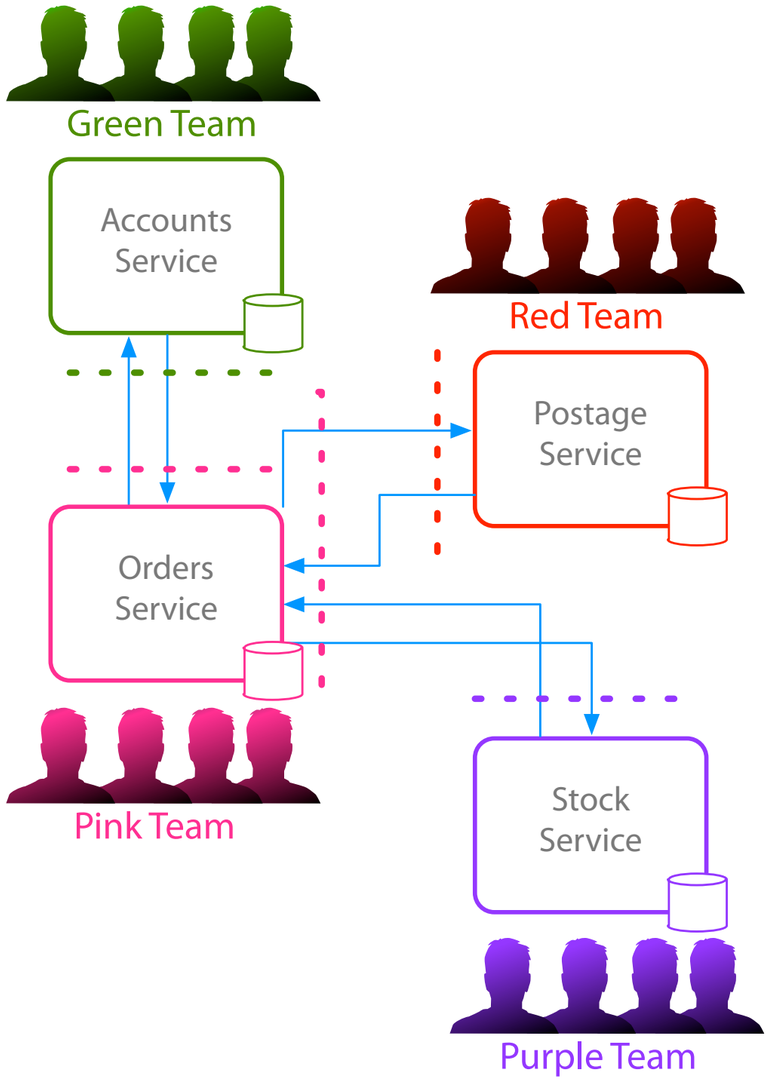
Approach: Autonomous



Loosely coupled

- Communication by network
  - Synchronous
  - Asynchronous
    - Publish events
    - Subscribe to events
- Technology agnostic API
- Avoid client libraries
- Contracts between services
  - Fixed and agreed interfaces
  - Shared models
  - Clear input and output
- Avoid chatty exchanges between services
- Avoid sharing between services
  - Databases
  - Shared libraries

Approach: Autonomous



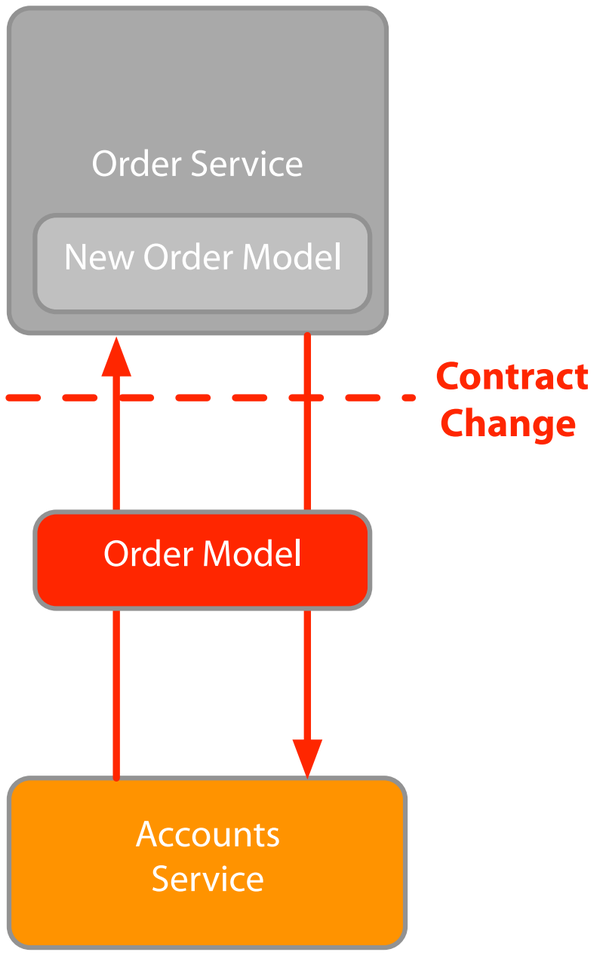
Microservice ownership by team

- Responsibility to make autonomous
- Agreeing contracts between teams
- Responsible for long-term maintenance
- Collaborative development
  - Communicate contract requirements
  - Communicate data requirements
- Concurrent development

Versioning

- Avoid breaking changes
- Backwards compatibility
- Integration tests
- Have a versioning strategy
  - Concurrent versions
    - Old and new
  - Semantic versioning
    - Major.Minor.Patch (e.g. 15.1.2)
  - Coexisting endpoints
    - /V2/customer/

Approach: Autonomous



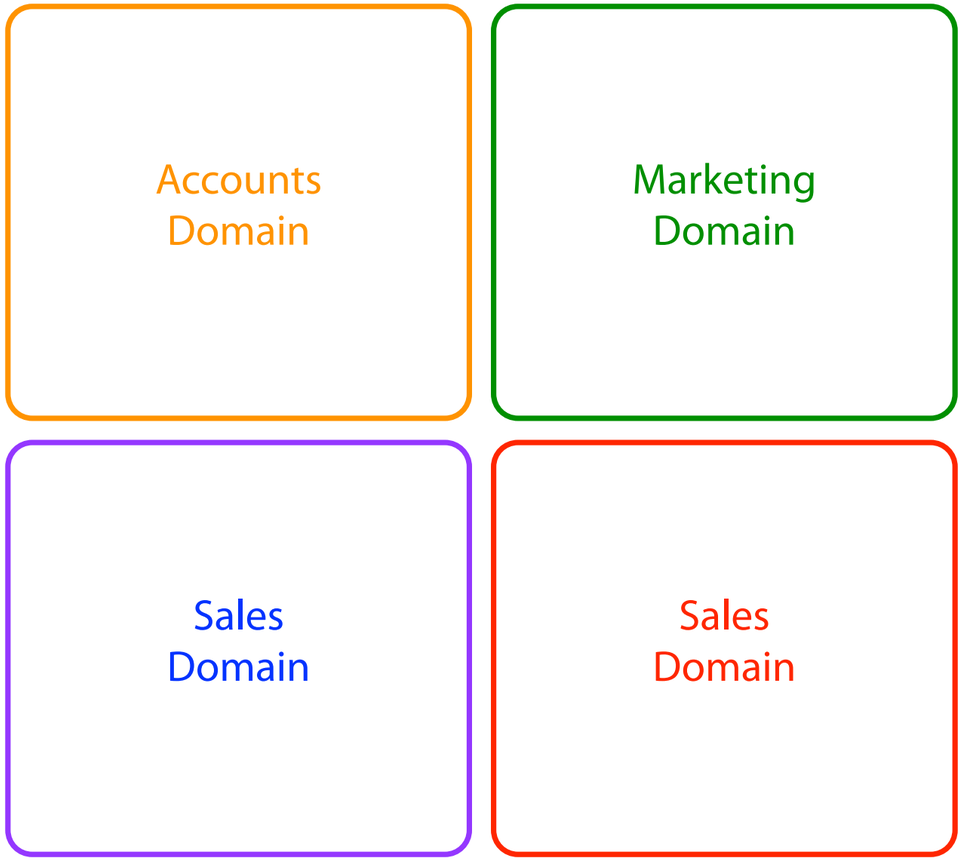
Microservice ownership by team

- Responsibility to make autonomous
- Agreeing contracts between teams
- Responsible for long-term maintenance
- Collaborative development
  - Communicate contract requirements
  - Communicate data requirements
- Concurrent development

Versioning

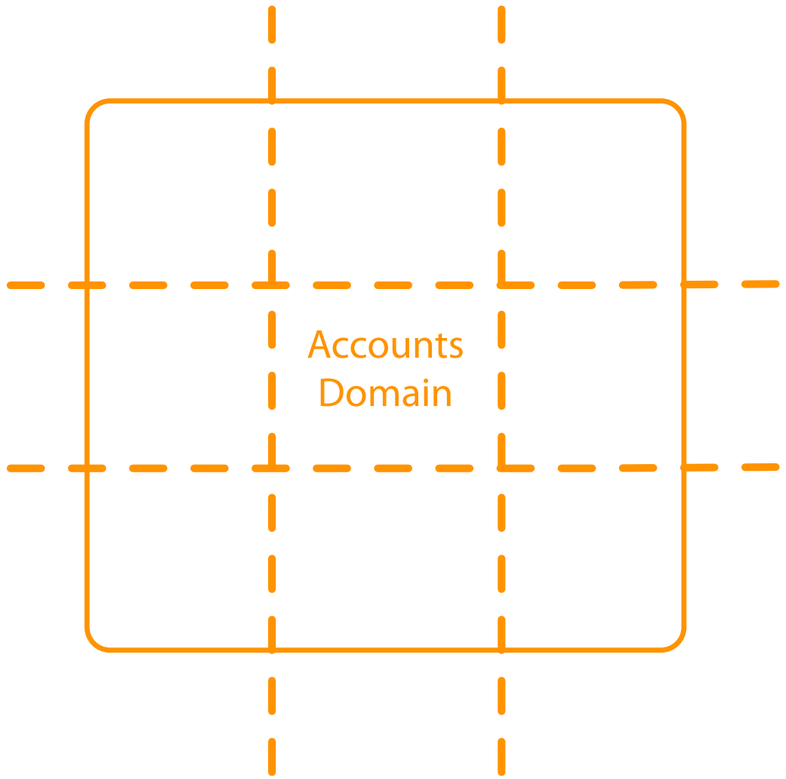
- Avoid breaking changes
- Backwards compatibility
- Integration tests
- Have a versioning strategy
  - Concurrent versions
    - Old and new
  - Semantic versioning
    - Major.Minor.Patch (e.g. 15.1.2)
  - Coexisting endpoints
    - /V2/customer/

Approach: Business Domain Centric



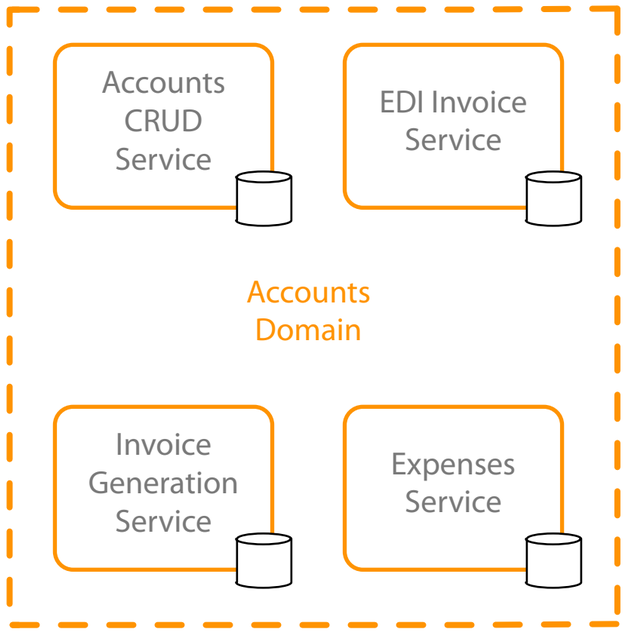
- Business function or business domain
- Approach
  - Identify business domains in a coarse manner
  - Review sub groups of business functions or areas
  - Review benefits of splitting further
  - Agree a common language
- Microservices for data (CRUD) or functions
- Fix incorrect boundaries
  - Merge or split
- Explicit interfaces for outside world
- Splitting using technical boundaries
  - Service to access archive data
  - For performance tuning

Approach: Business Domain Centric



- Business function or business domain
- Approach
  - Identify business domains in a coarse manner
  - Review sub groups of business functions or areas
  - Review benefits of splitting further
  - Agree a common language
- Microservices for data (CRUD) or functions
- Fix incorrect boundaries
  - Merge or split
- Explicit interfaces for outside world
- Splitting using technical boundaries
  - Service to access archive data
  - For performance tuning

Approach: Business Domain Centric



Business function or business domain

Approach

- Identify business domains in a coarse manner
- Review sub groups of business functions or areas
- Review benefits of splitting further
- Agree a common language

Microservices for data (CRUD) or functions

Fix incorrect boundaries

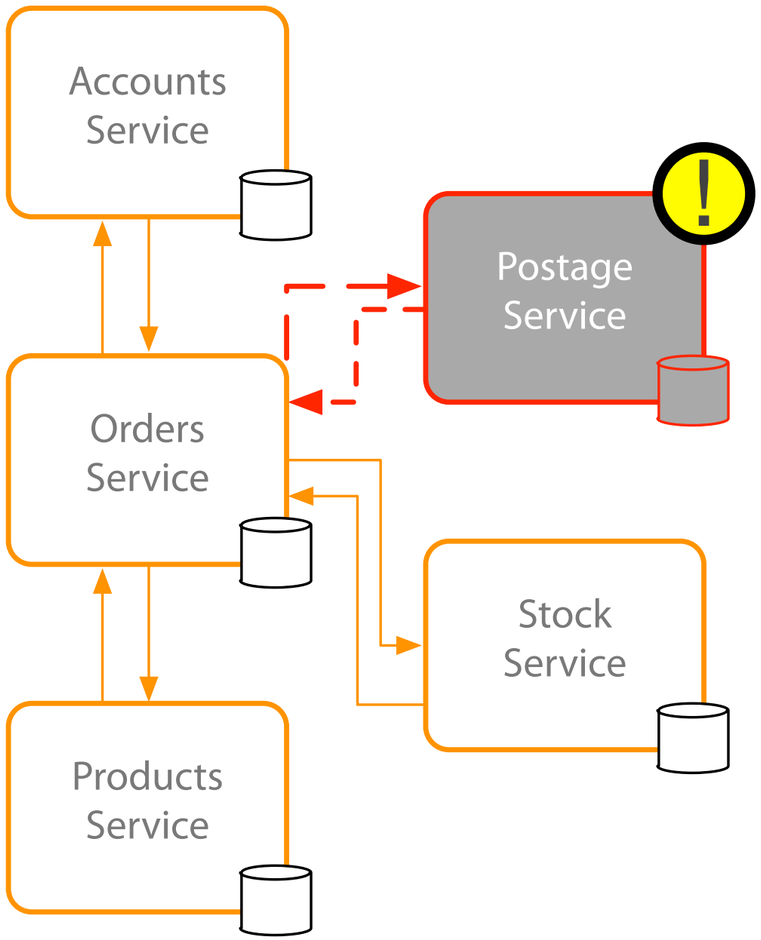
- Merge or split

Explicit interfaces for outside world

Splitting using technical boundaries

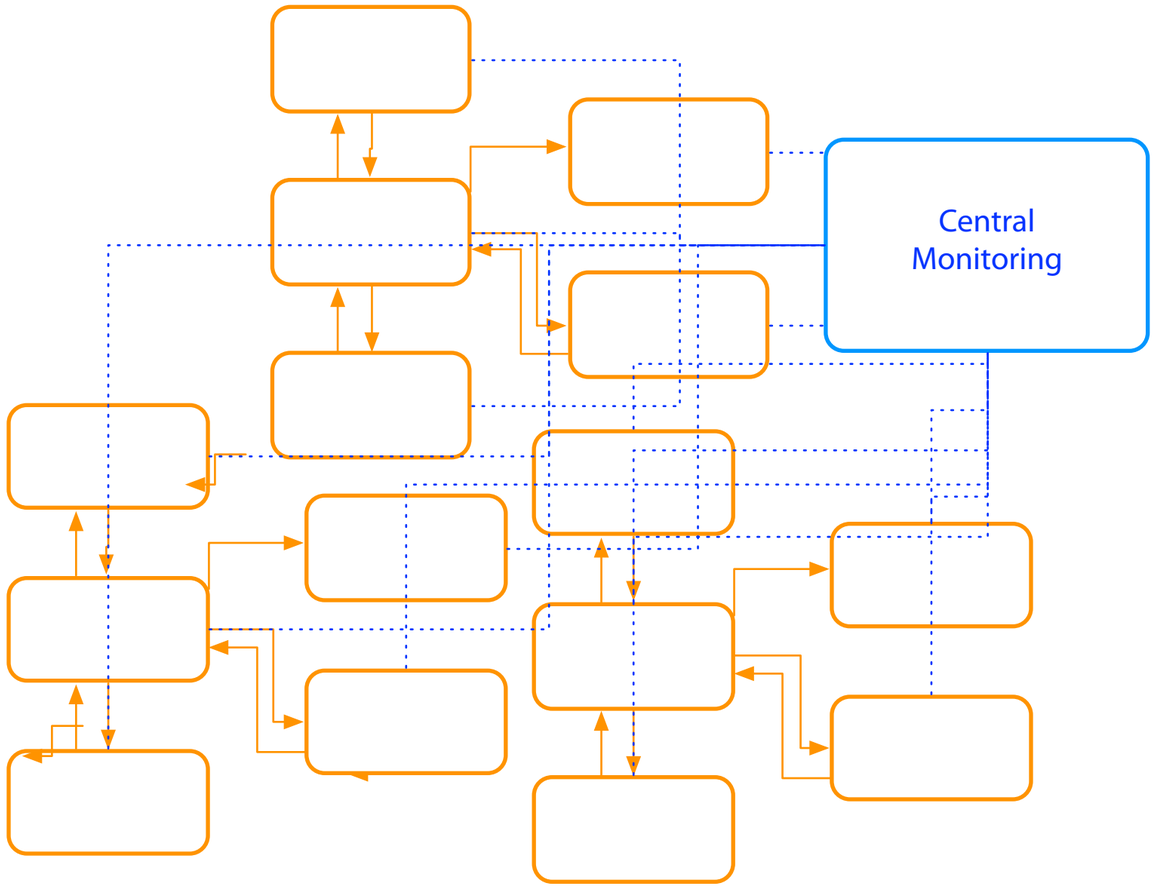
- Service to access archive data
- For performance tuning

Approach: Resilience



- Design for known failures
- Failure of downstream systems
  - Other services internal or external
- Degrade functionality on failure detection
- Default functionality on failure detection
- Design system to fail fast
- Use timeouts
  - Use for connected systems
  - Timeout our requests after a threshold
  - Service to service
  - Service to other systems
  - Standard timeout length
  - Adjust length on a case by case basis
- Network outages and latency
- Monitor timeouts
- Log timeouts

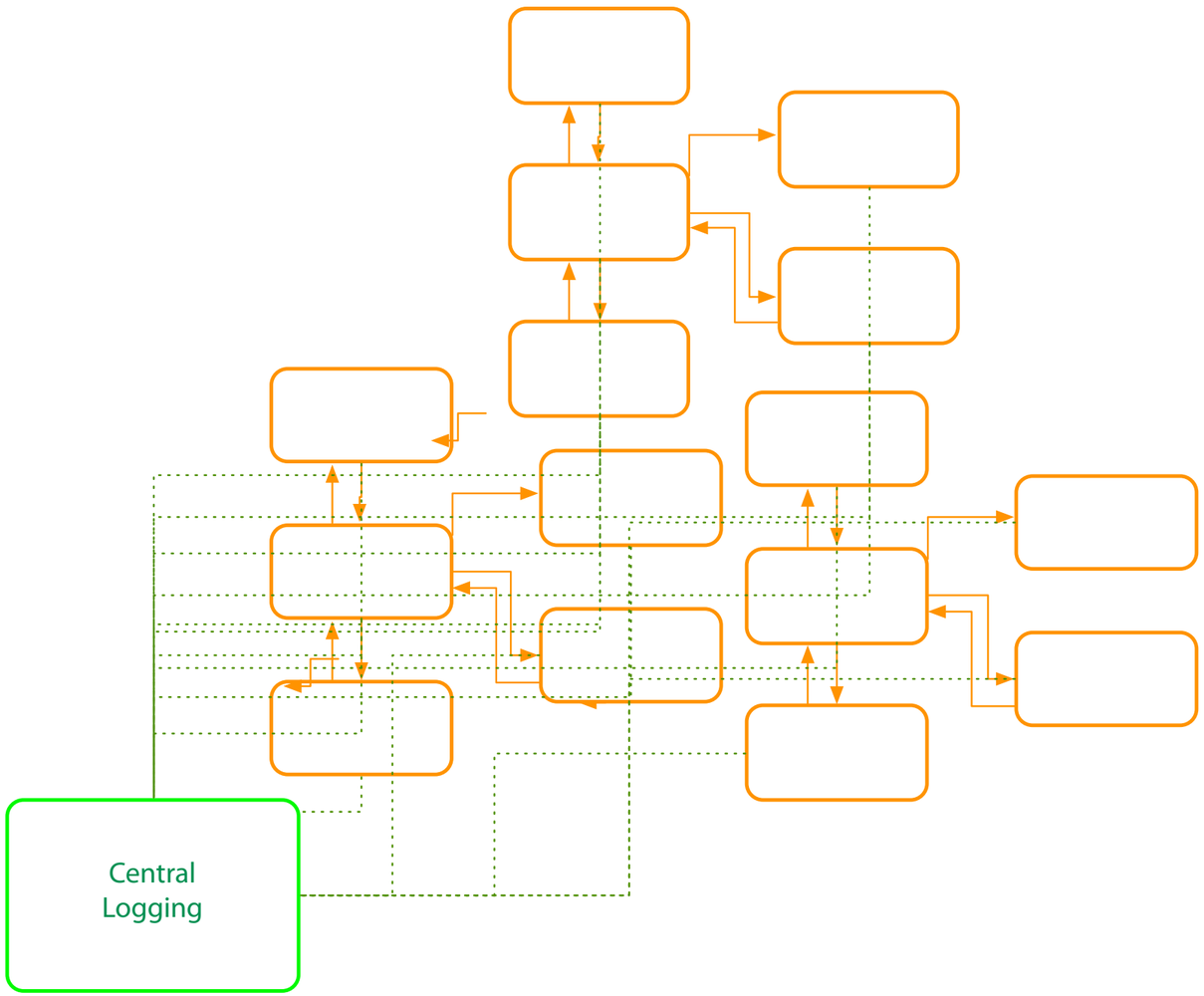
Approach: Observable



Centralized monitoring

- Real-time monitoring
- Monitor the host
  - CPU, memory, disk usage, etc.
- Expose metrics within the services
  - Response times
  - Timeouts
  - Exceptions and errors
- Business data related metrics
  - Number of orders
  - Average time from basket to checkout
- Collect and aggregate monitoring data
  - Monitoring tools that provide aggregation
  - Monitoring tools that provide drill down options
- Monitoring tool that can help visualise trends
- Monitoring tool that can compare data across servers
- Monitoring tool that can trigger alerts

Approach: Observable



Centralized Logging

When to log

- Startup or shutdown
- Code path milestones
  - Requests, responses and decisions
- Timeouts, exceptions and errors

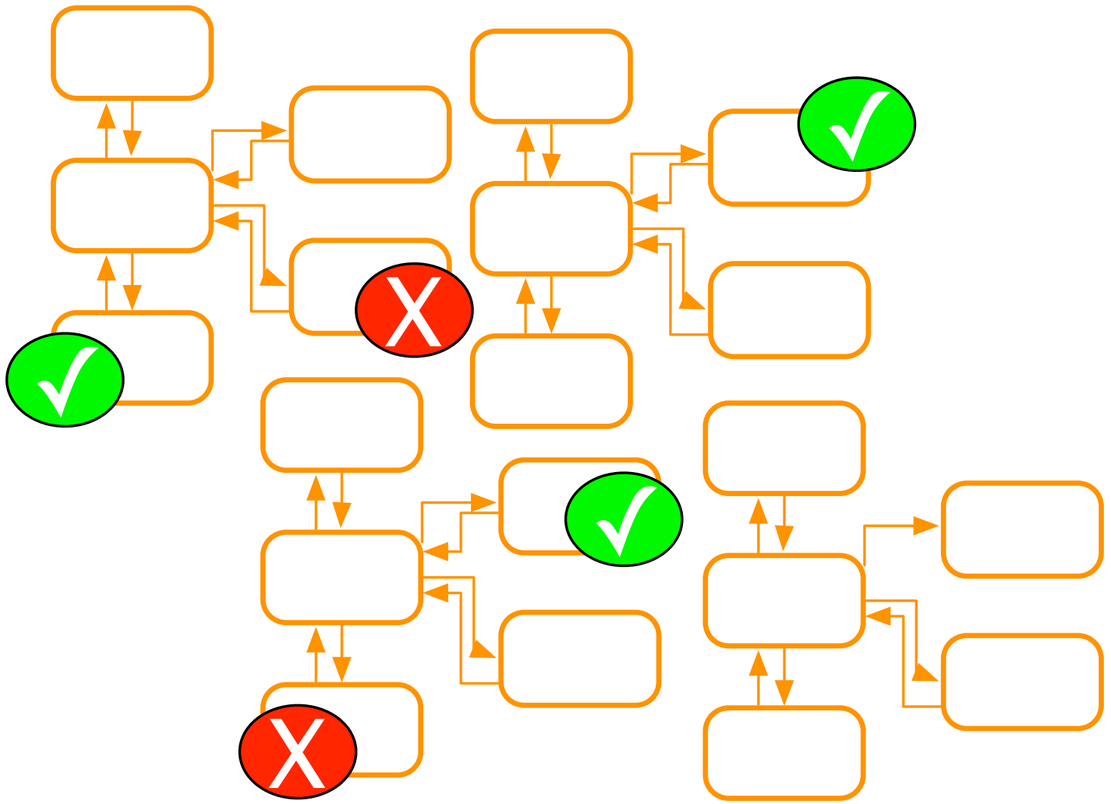
Structured logging

- Level
  - Information
  - Error
  - Debug
  - Statistic
- Date and time
- Correlation ID
- Host name
- Service name and service instance
- Message

Traceable distributed transactions

- Correlation ID
  - Passed service to service

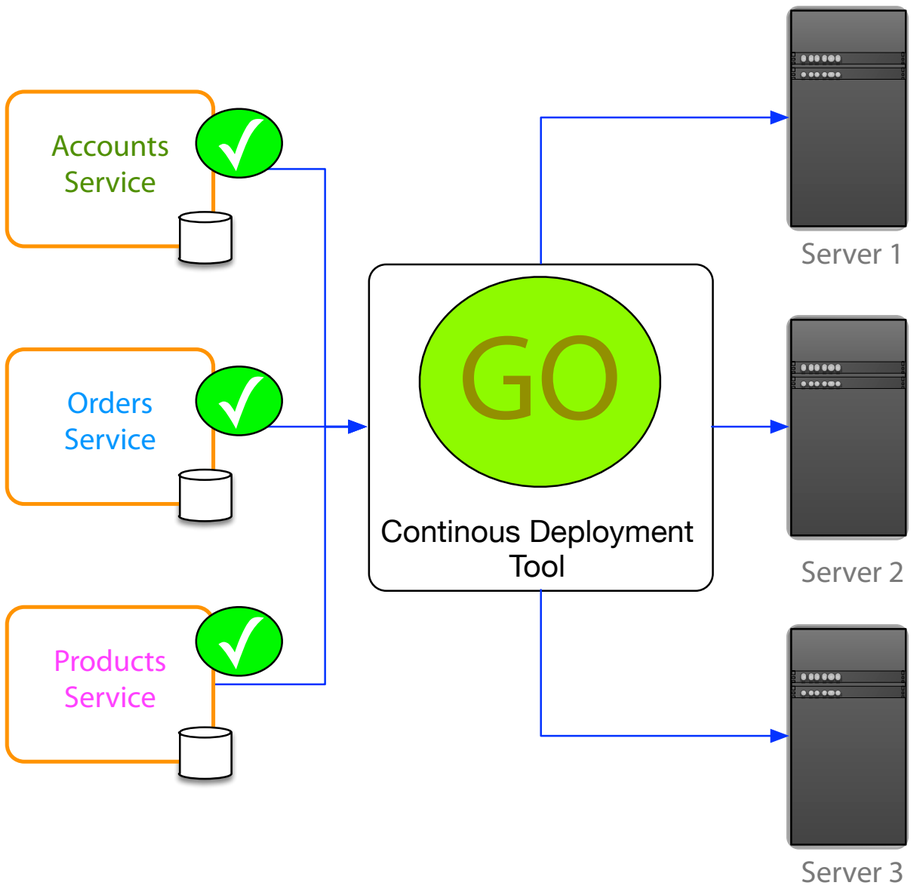
Approach: Automation



Continuous Integration Tools

- Work with source control systems
- Automatic after check-in
- Unit tests and integration tests required
- Ensure quality of check-in
  - Code compiles
  - Tests pass
  - Changes integrate
  - Quick feedback
- Urgency to fix quickly
- Creation of build
- Build ready for test team
- Build ready for deployment

Approach: Automation



Continuous Deployment Tools

Automate software deployment

- Configure once
- Works with CI tools
- Deployable after check in
- Reliably released at anytime

Benefits

- Quick to market
- Reliable deployment
- Better customer experience

Module Summary

High Cohesion

Single thing done well

Single focus

Approach

Keeps splitting service until it only has one reason to change

Autonomous

Independently changeable and deployable

Approach

Loosely coupled system

Versioning strategy

Microservice ownership by team

Business Domain Centric

Represent business function

or represent a business domain

Approach

Course grain business domains

Subgroup into functions and areas

Resilience

Embrace Failure

Default or degrade functionality

Approach

Design for known failures

Fail fast and recover fast

Observable

See system health

Centralized logging and monitoring

Approach

Tools for real-time centralized monitoring

Tools for centralized structured logging

Automation

Tools for testing and feedback

Tools for deployment

Approach

Continuous Integration Tools

Continuous Deployment Tools

Technology for Microservices

Rag Dhiman

ragcode.com | @RagDhiman

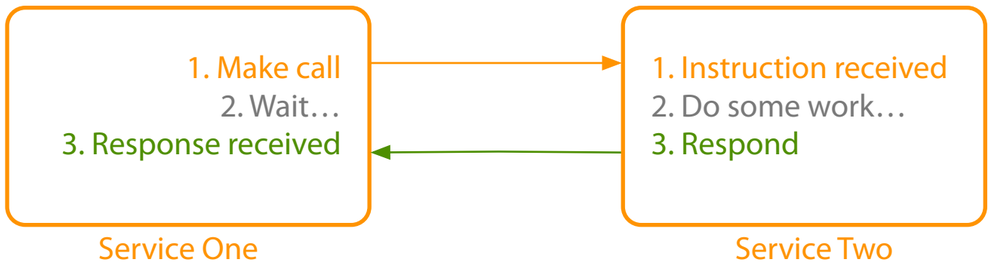
Module Overview

- Communication
- Hosting Platforms
- Observable Microservices
- Performance
- Automation Tools

Communication

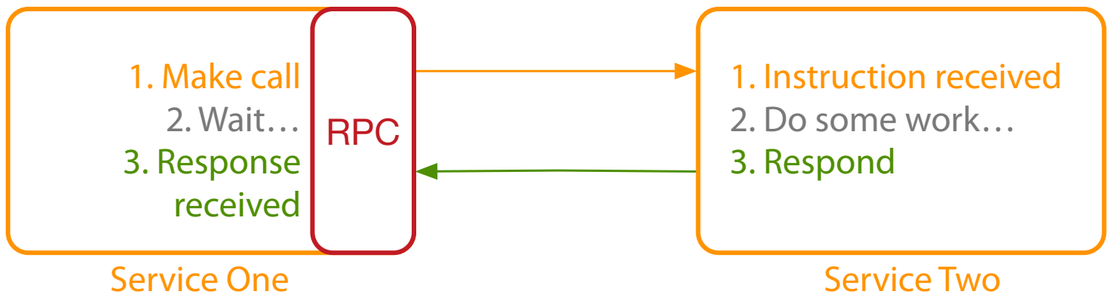
Synchronous | Asynchronous

Communication: Synchronous



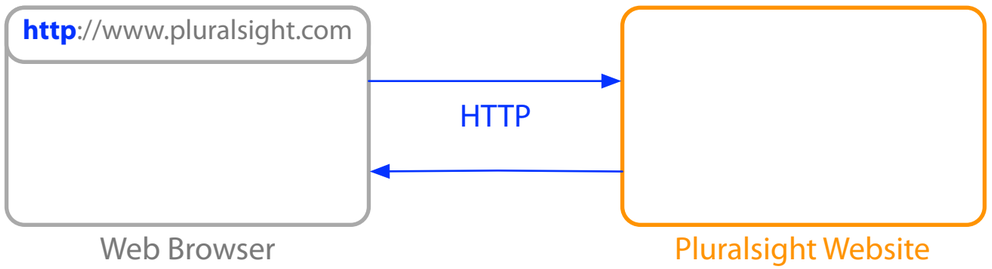
- Request response communication
  - Client to service
  - Service to service
  - Service to external
- Remote procedure call
  - Sensitive to change
- HTTP
  - Work across the internet
  - Firewall friendly
- REST
  - CRUD using HTTP verbs
  - Natural decoupling
  - Open communication protocol
  - REST with HATEOS
- Synchronous issues
  - Both parties have to be available
  - Performance subject to network quality
  - Clients must know location of service (host\port)

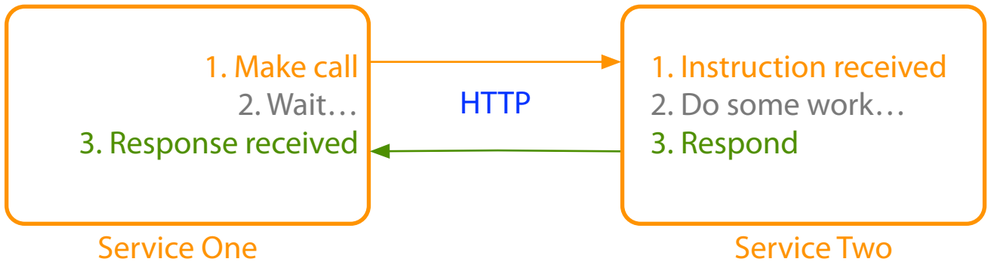
Communication: Synchronous



- Request response communication
  - Client to service
  - Service to service
  - Service to external
- Remote procedure call
  - Sensitive to change
- HTTP
  - Work across the internet
  - Firewall friendly
- REST
  - CRUD using HTTP verbs
  - Natural decoupling
  - Open communication protocol
  - REST with HATEOS
- Synchronous issues
  - Both parties have to be available
  - Performance subject to network quality
  - Clients must know location of service (host\port)

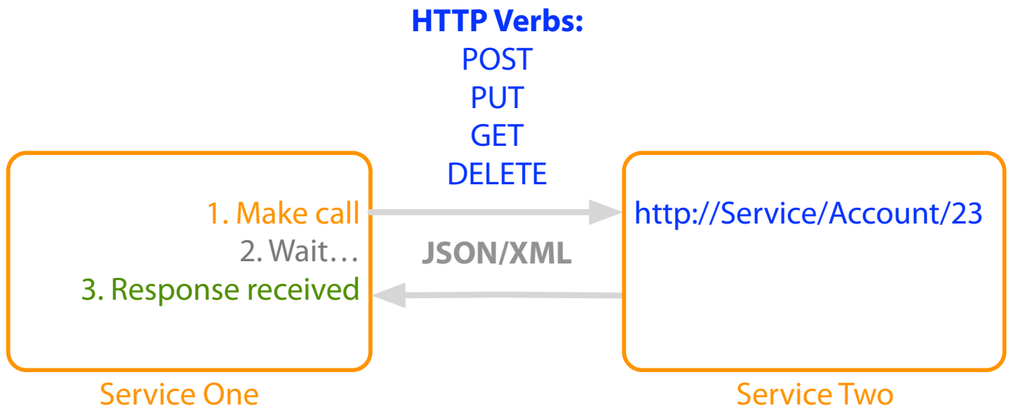
Communication: Synchronous





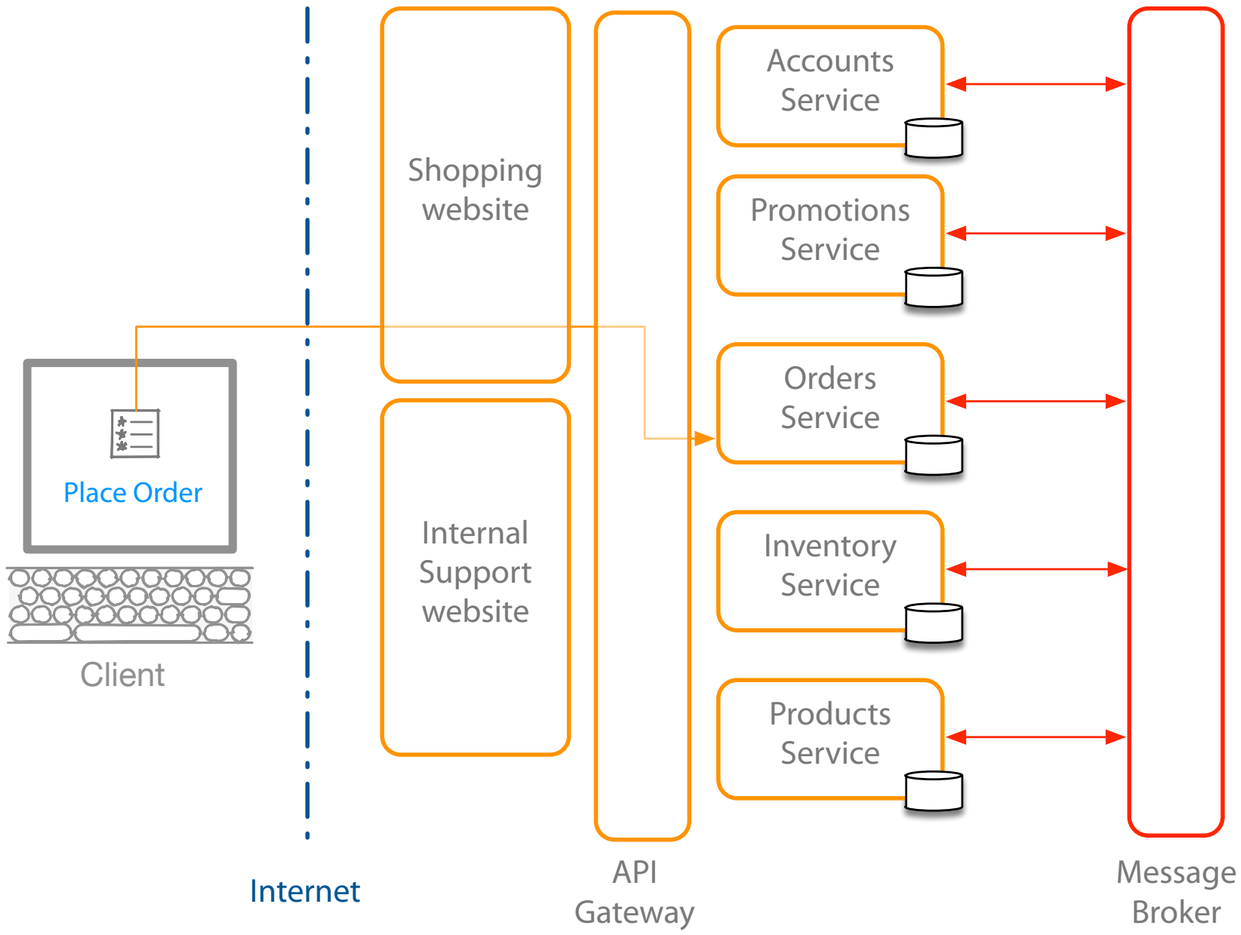
- Request response communication
  - Client to service
  - Service to service
  - Service to external
- Remote procedure call
  - Sensitive to change
- HTTP
  - Work across the internet
  - Firewall friendly
- REST
  - CRUD using HTTP verbs
  - Natural decoupling
  - Open communication protocol
  - REST with HATEOS
- Synchronous issues
  - Both parties have to be available
  - Performance subject to network quality
  - Clients must know location of service (host\port)

Communication: Synchronous



- Request response communication
  - Client to service
  - Service to service
  - Service to external
- Remote procedure call
  - Sensitive to change
- HTTP
  - Work across the internet
  - Firewall friendly
- REST
  - CRUD using HTTP verbs
  - Natural decoupling
  - Open communication protocol
  - REST with HATEOS
- Synchronous issues
  - Both parties have to be available
  - Performance subject to network quality
  - Clients must know location of service (host\port)

Communication: Asynchronous



Event based

- Mitigates the need of client and service availability
- Decouples client and service

Message queueing protocol

- Message Brokers
- Subscriber and publisher are decoupled
- Microsoft message queuing (MSMQ)
- RabbitMQ
- ATOM (HTTP to propagate events)

Asynchronous challenge

- Complicated
- Reliance on message broker
- Visibility of the transaction
- Managing the messaging queue

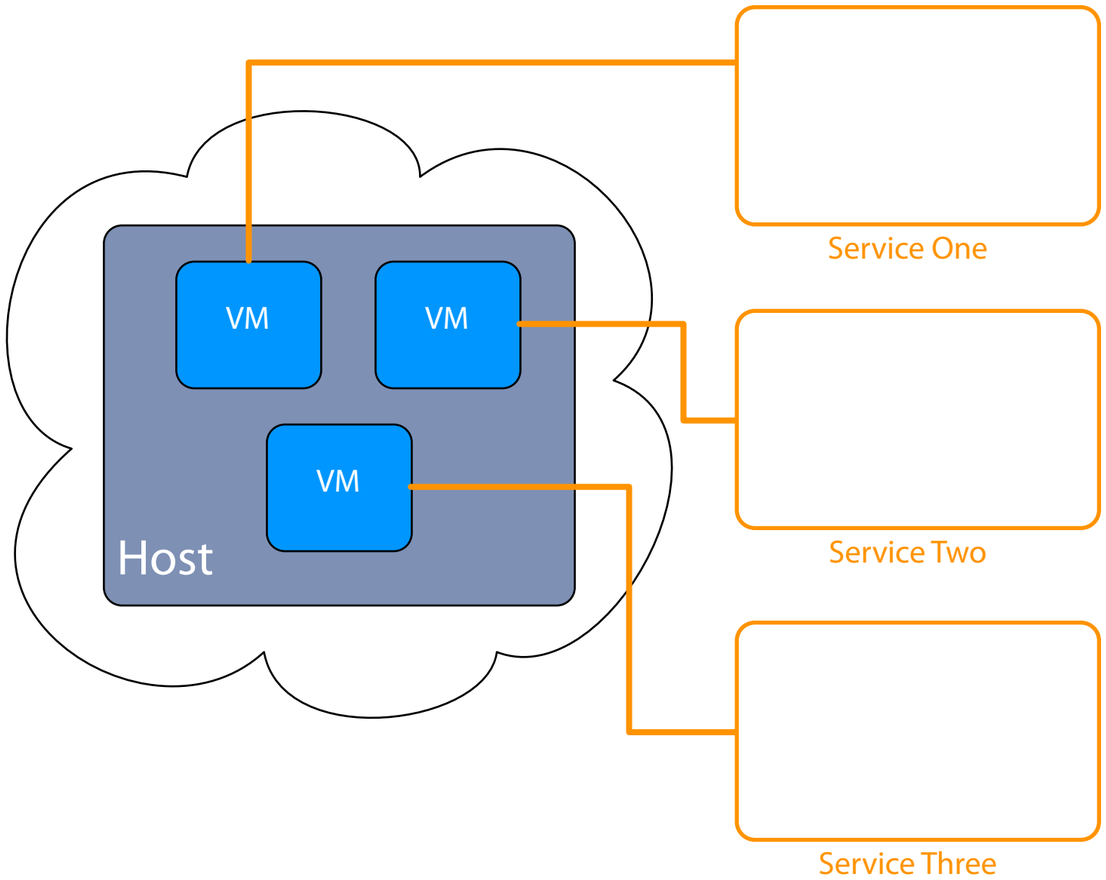
Real world systems

- Would use both synchronous and asynchronous

Hosting Platforms

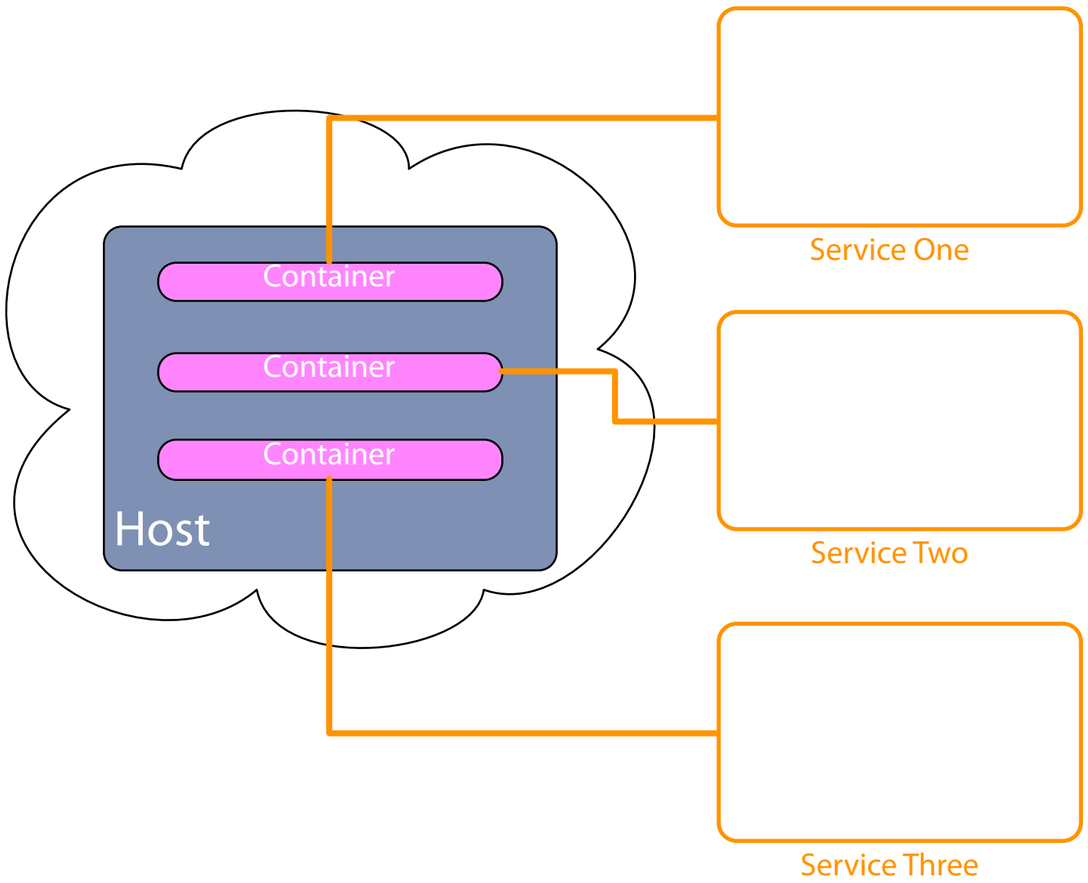
Virtualization | Containers | Self Hosting | Registry and Discovery

Hosting Platforms: Virtualization



- A virtual machine as a host
- Foundation of cloud platforms
  - Platform as a service (PAAS)
    - Microsoft Azure
    - Amazon web services
    - Your own cloud (for example vSphere)
- Could be more efficient
  - Takes time to setup
  - Takes time to load
  - Take quite a bit of resource
- Unique features
  - Take snapshot
  - Clone instances
- Standardised and mature

Hosting Platforms: Containers

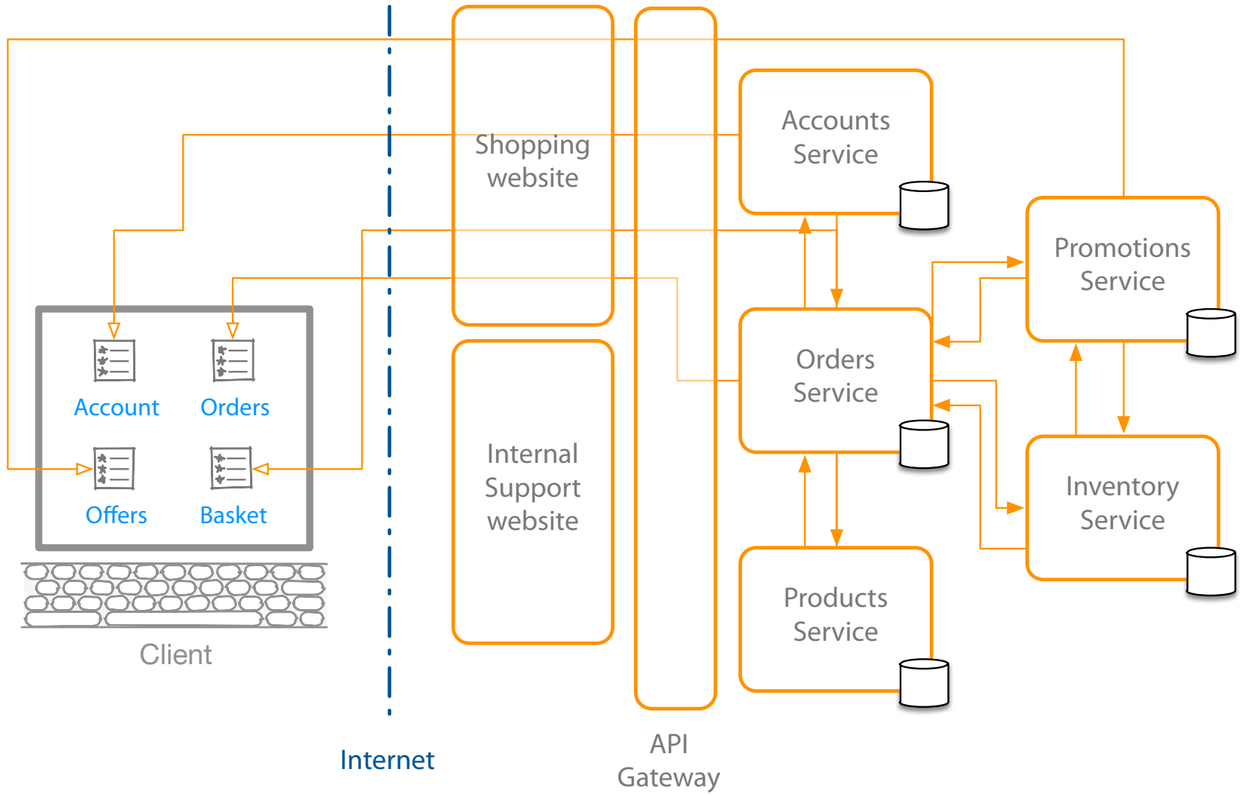


- Type of virtualization
- Isolate services from each other
- Single service per container
- Different to a virtual machine
  - Use less resource than VM
  - Faster than VM
  - Quicker to create new instances
- Future of hosted apps
- Cloud platform support growing
- Mainly Linux based
- Not as established as virtual machines
  - Not standardised
  - Limited features and tooling
  - Infrastructure support in its infancy
  - Complex to setup
- Examples
  - Docker
  - Rocker
  - Glassware

Hosting Platforms: Self Hosting

- Implement your own cloud
  - Virtualization platform
  - Implement containers
- Use of physical machines
  - Single service on a server
  - Multiple services on a server
- Challenges
  - Long-term maintenance
  - Need for technicians
  - Training
  - Need for space
  - Scaling is not as immediate

Hosting Platforms: Registration and Discovery



Where?

- Host, port and version

Service registry database

Register on startup

Deregister service on failure

Cloud platforms make it easy

Local platform registration options

- Self registration
- Third-party registration

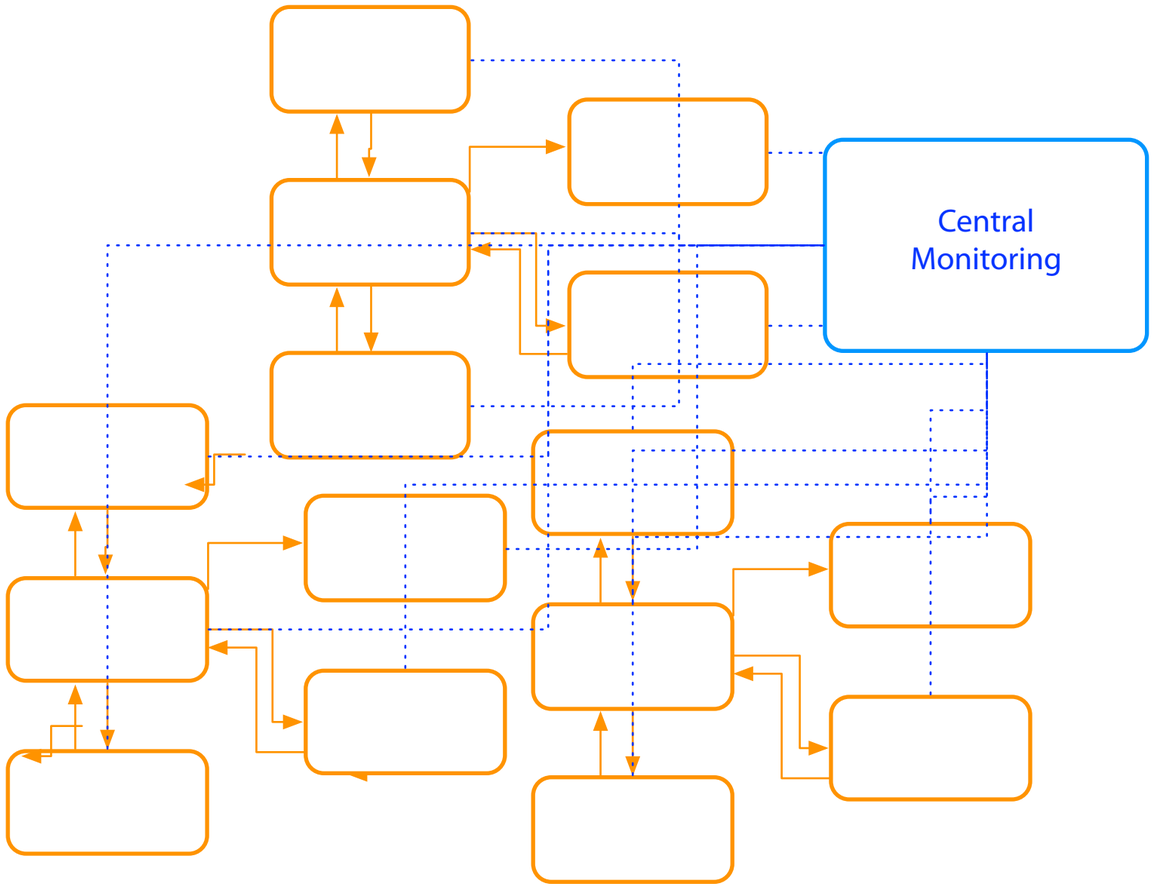
Local platform discovery options

- Client-side discovery
- Server-side discovery

Observable Microservices

Monitoring Tech | Logging Tech

Observable Microservices: Monitoring Tech

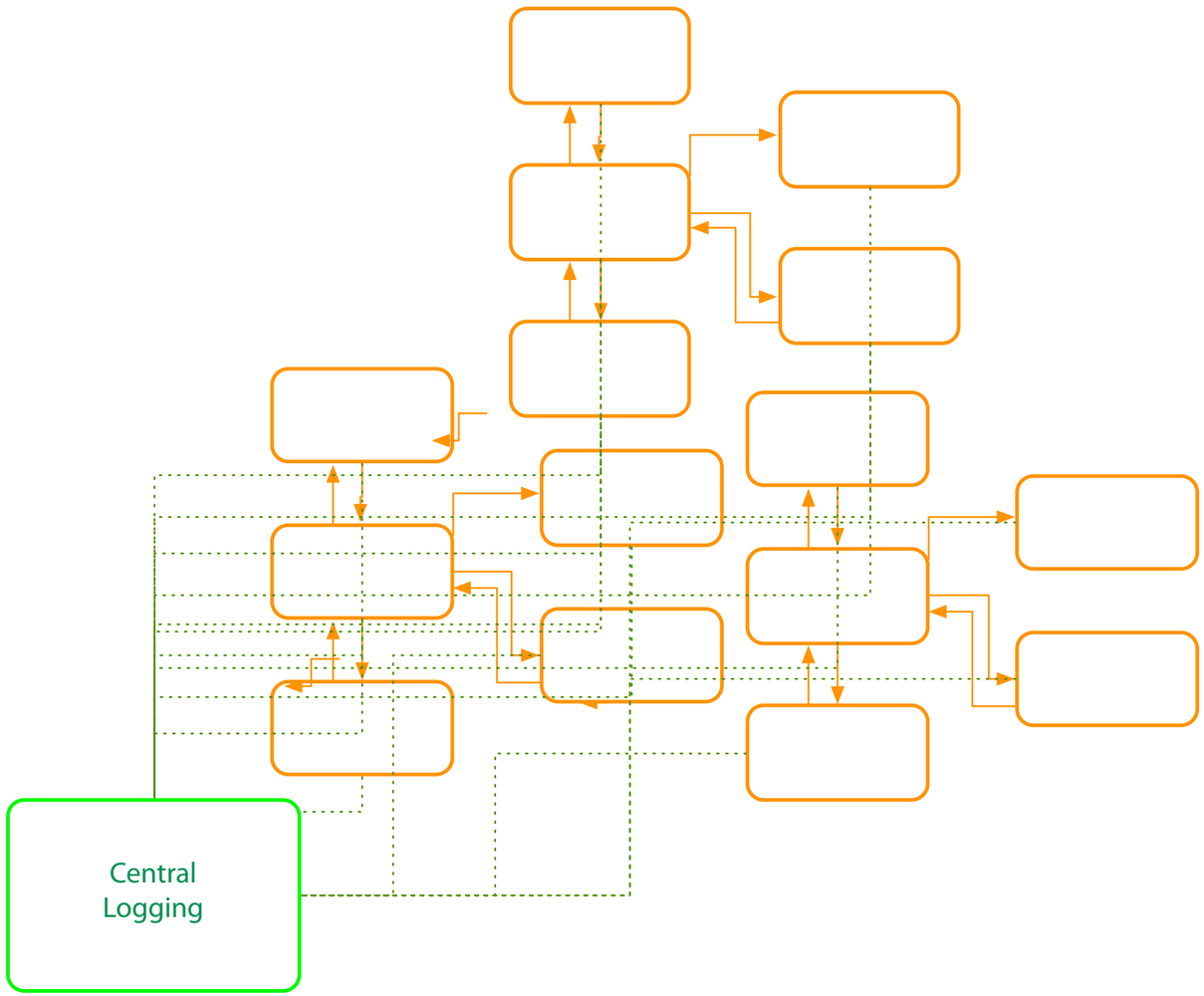


- Centralised tools
  - Nagios
  - PRTG
  - Load balancers
  - New Relic
- Desired features
  - Metrics across servers
  - Automatic or minimal configuration
  - Client libraries to send metrics
  - Test transactions support
  - Alerting
- Network monitoring
- Standardise monitoring
  - Central tool
  - Preconfigured virtual machines or containers
- Real-time monitoring

Observable Microservices

Monitoring Tech | Logging Tech

Observable Microservices: Logging Tech

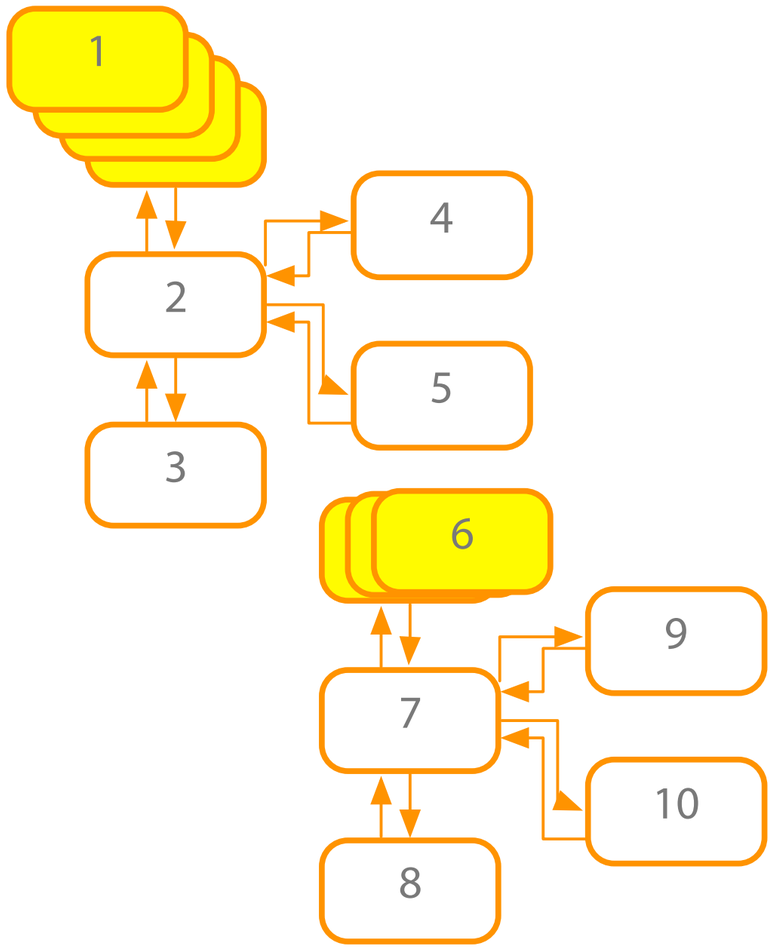


- Portal for centralised logging data
  - Elastic log
  - Log stash
  - Splunk
  - Kibana
  - Graphite
- Client logging libraries
  - Serilog
  - and many more…
- Desired features
  - Structured logging
  - Logging across servers
  - Automatic or minimal configuration
  - Correlation\Context ID for transactions
- Standardise logging
  - Central tool
  - Template for client library

Microservices Performance

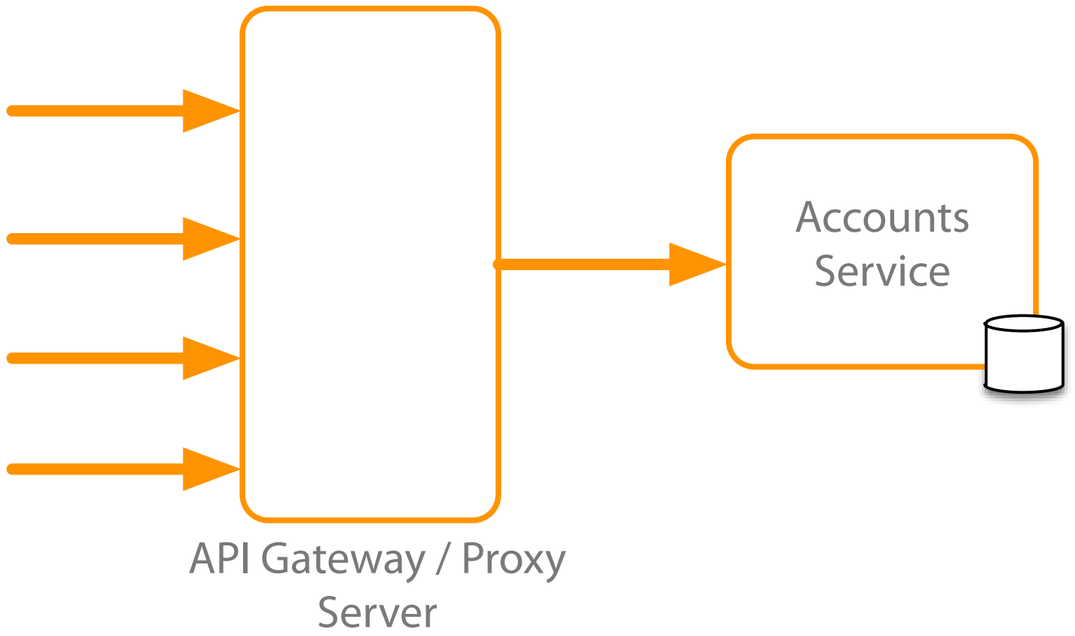
Scaling | Caching | API Gateway

Microservices Performance: Scaling



- How
  - Creating multiple instances of service
  - Adding resource to existing service
- Automated or on-demand
- PAAS auto scaling options
- Virtualization and containers
- Physical host servers
- Load balancers
  - API Gateway
- When to scale up
  - Performance issues
  - Monitoring data
  - Capacity planning

Microservices Performance: Caching



Caching to reduce

- Client calls to services
- Service calls to databases
- Service to service calls

API Gateway\Proxy level

Client side

Service level

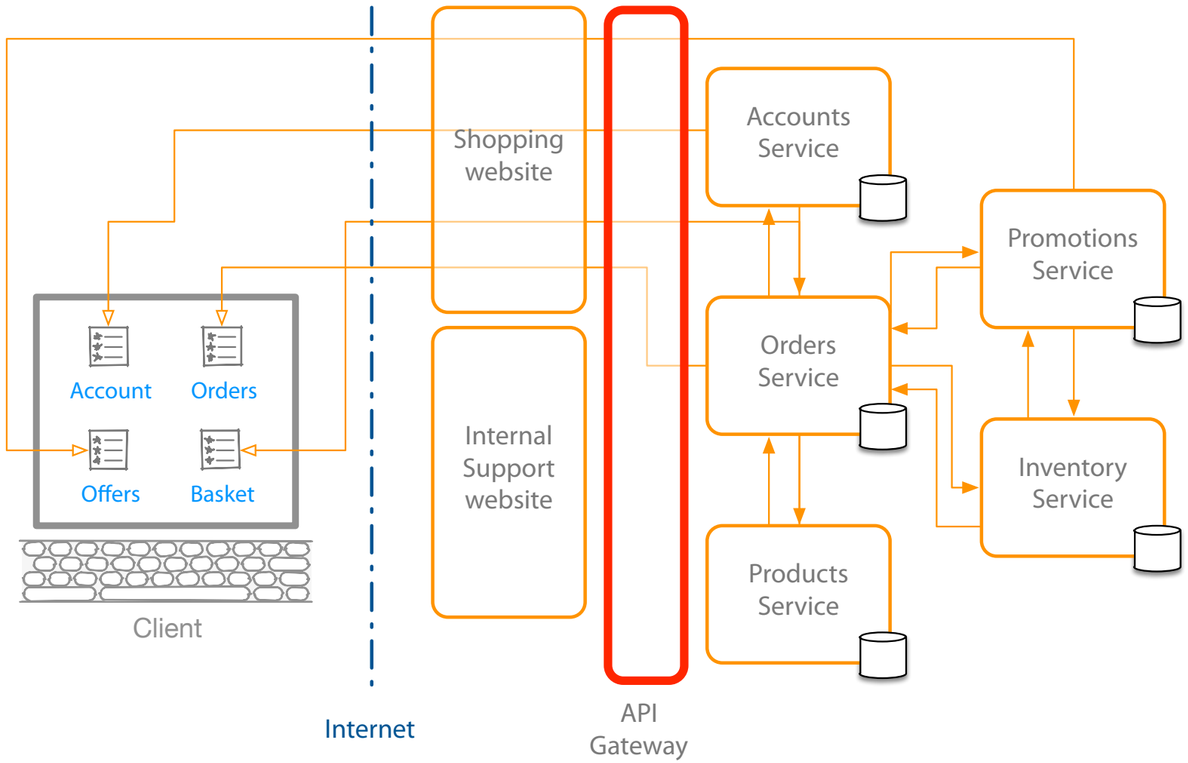
Considerations

- Simple to setup and manage
- Data leaks

Microservices Performance

Scaling | Caching | API Gateway

Microservices Performance: API Gateway



Help with performance

- Load balancing
- Caching

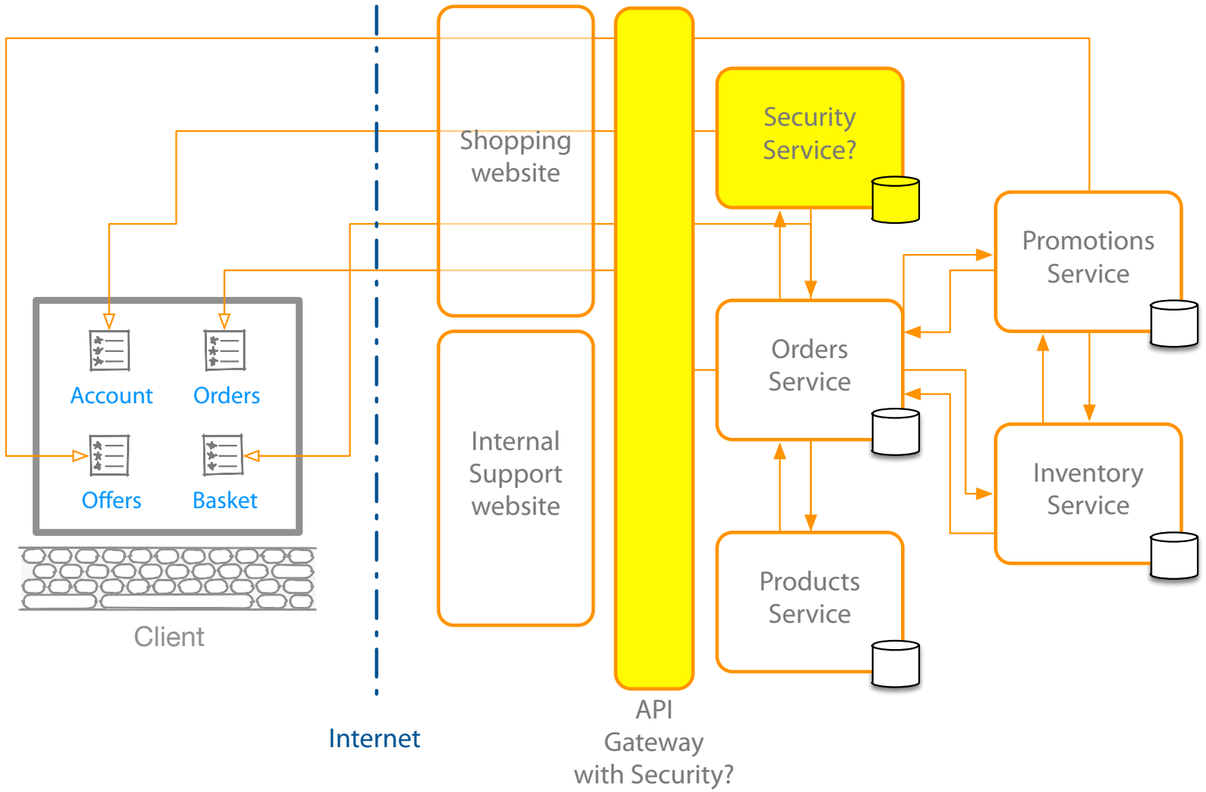
Help with

- Creating central entry point
- Exposing services to clients
- One interface to many services
- Dynamic location of services
- Routing to specific instance of service
- Service registry database

Security

- API Gateway
- Dedicated security service
- Central security vs service level

Microservices Performance: API Gateway



Help with performance

- Load balancing
- Caching

Help with

- Creating central entry point
- Exposing services to clients
- One interface to many services
- Dynamic location of services
- Routing to specific instance of service
- Service registry database

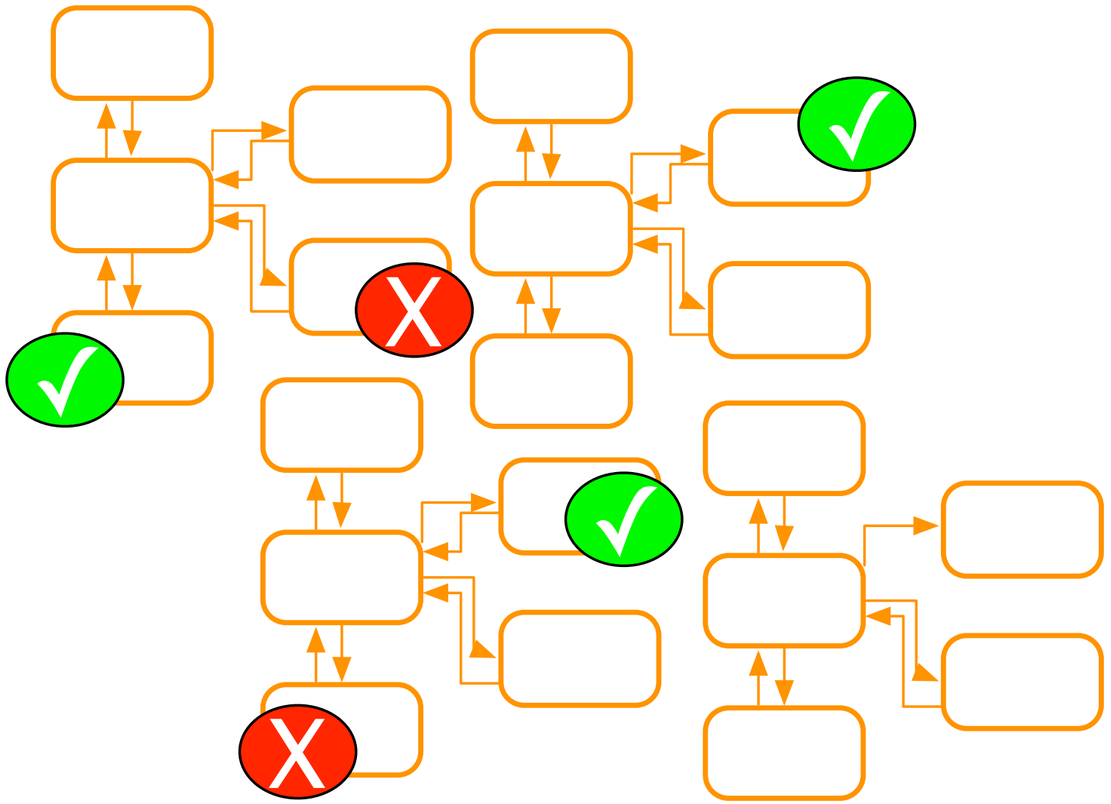
Security

- API Gateway
- Dedicated security service
- Central security vs service level

Automation Tools

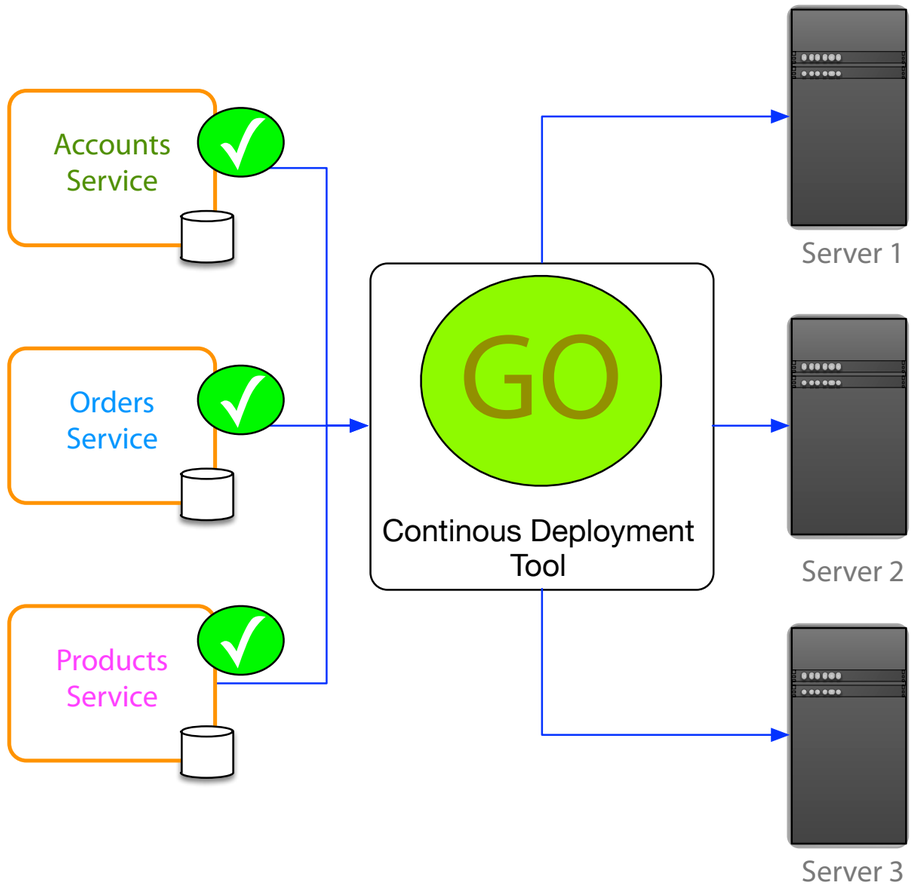
Continuous Integration | Continuous Deployment

Automation Tools: Continuous Integration



- Many CI tools
  - Team Foundation Server
  - TeamCity
  - Many more!
- Desired features
  - Cross platform
    - Windows builders, Java builders and others
  - Source control integration
  - Notifications
  - IDE Integration (optional)
- Map a microservice to a CI build
  - Code change triggers build of specific service
  - Feedback just received on that service
  - Builds and tests run quicker
  - Separate code repository for service
  - End product is in one place
  - CI builds to test database changes
  - Both microservice build and database upgrade are ready
- Avoid one CI build for all services

Automation Tools: Continuous Deployment



Many CD tools

- Aim for cross platform tools

Desired features

- Central control panel
- Simple to add deployment targets
- Support for scripting
- Support for build statuses
- Integration with CI tool
- Support for multiple environments
- Support for PAAS

Module Summary

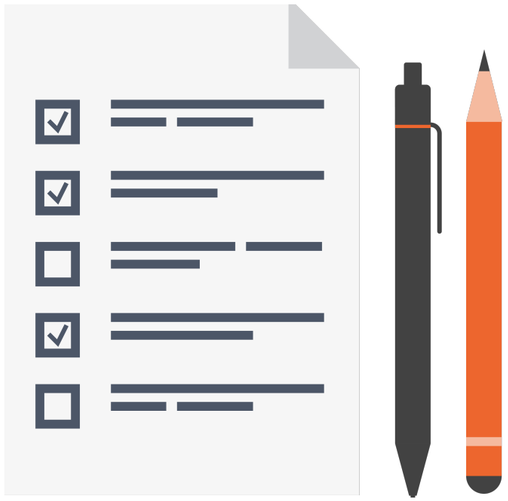
- Communication
  - Synchronous
  - Asynchronous
- Hosting Platforms
  - Virtualization
  - Containers
  - Self Hosting
  - Registry and Discovery
- Observable Microservices
  - Monitoring Tech
  - Logging Tech
- Performance
  - Scaling
  - Caching
  - API Gateway
- Automation Tools
  - Continuous Integration
  - Continuous Deployment

Moving Forwards with Microservices

Rag Dhiman

ragcode.com | @RagDhiman

Module Overview



Brownfield Microservices

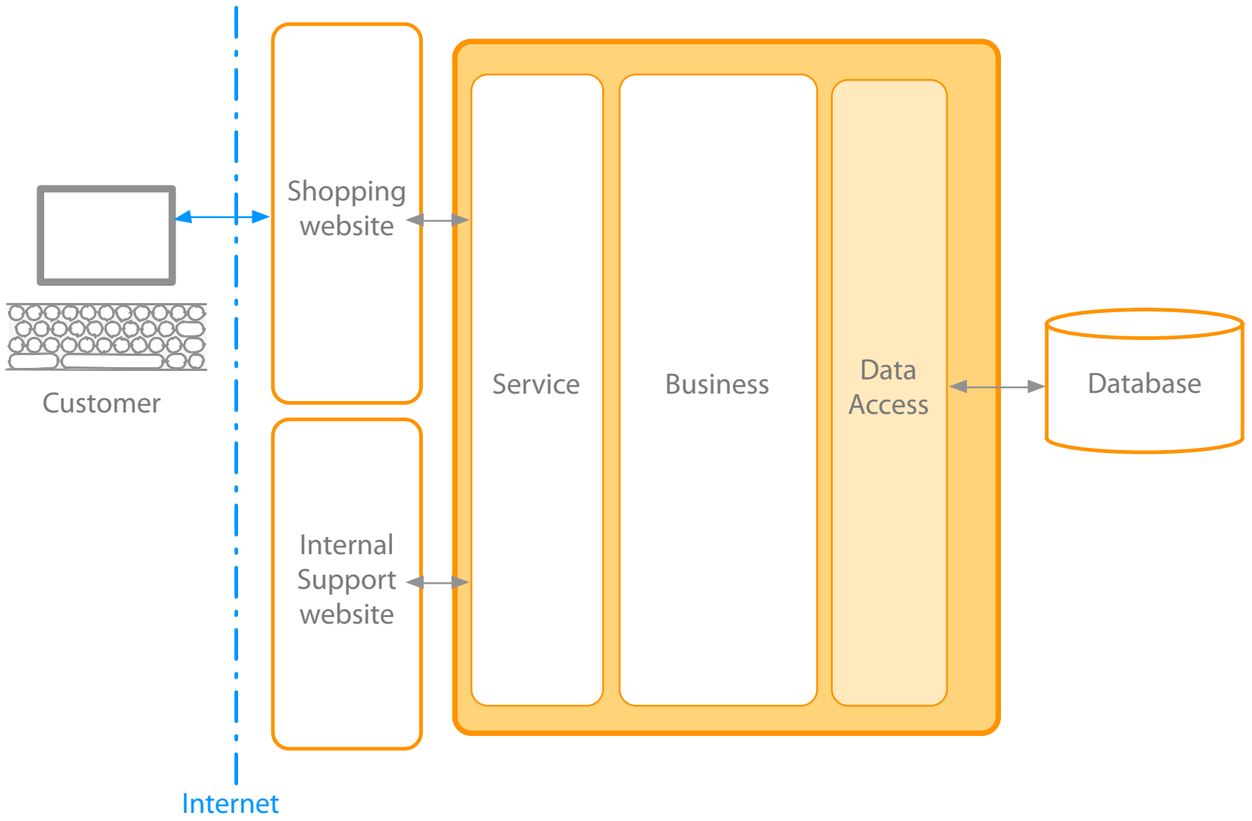
Greenfield Microservices

Microservices Provisos

Brownfield Microservices

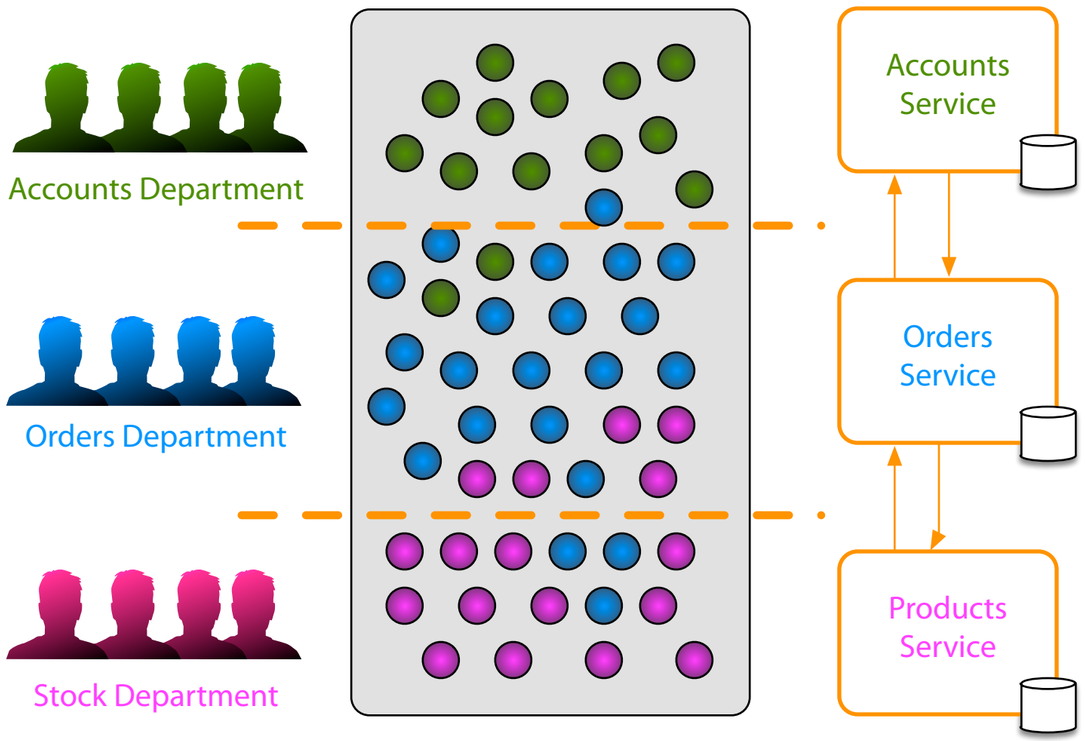
Approach | Migration | Database Migration | Transactions | Reporting

Brownfield Microservices: Approach



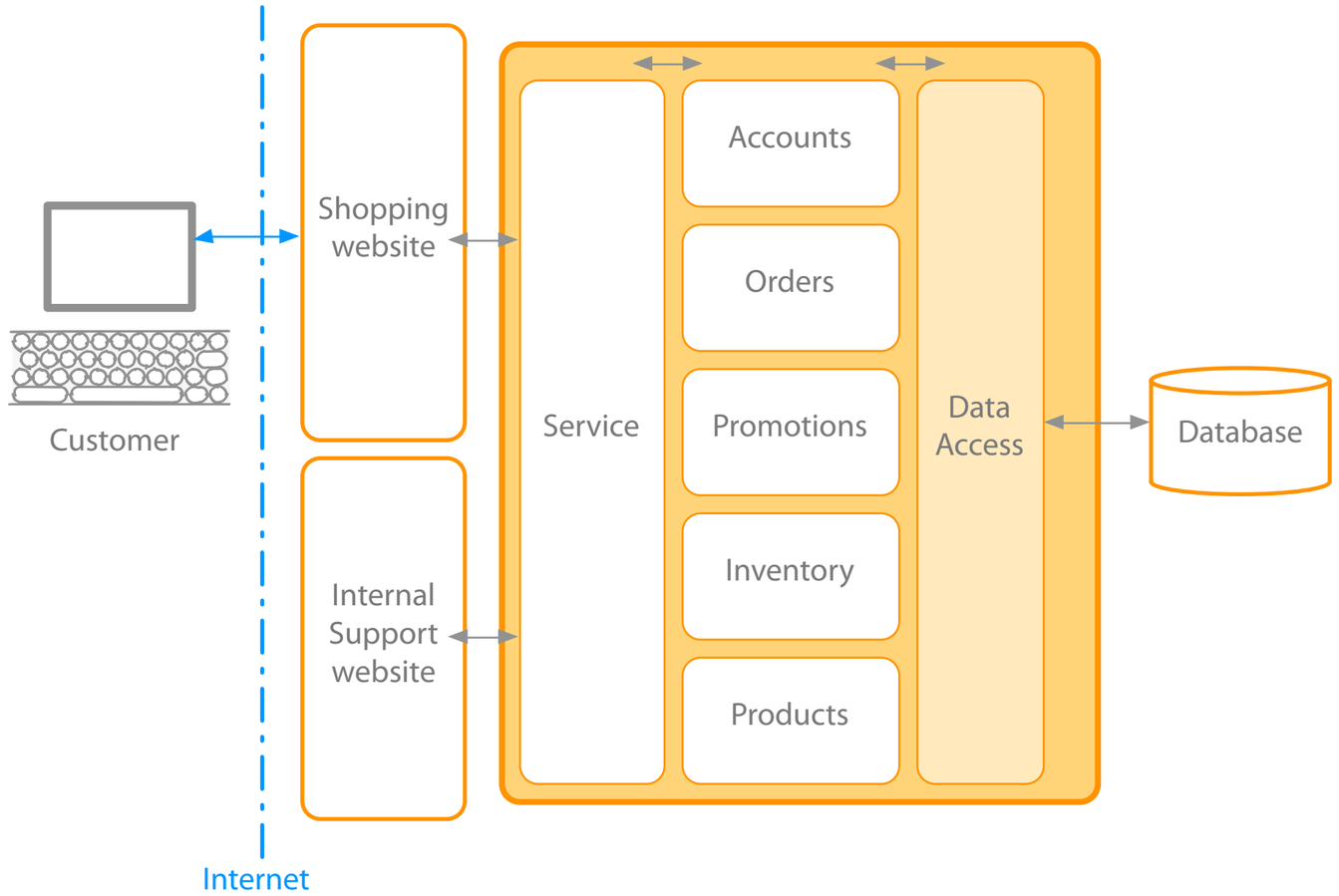
- Existing system
  - Monolithic system
  - Organically grown
  - Seems to large to split
- Lacks microservices design principles
- Identify seams
  - Separation that reflects domains
  - Identify bounded contexts
- Start modularising the bounded contexts
  - Move code incrementally
  - Tidy up a section per release
    - Take your time
    - Existing functionality needs to remain intact
  - Run unit tests and integration tests to validate change
  - Keep reviewing
- Seams are future microservice boundaries

Brownfield Microservices: Approach



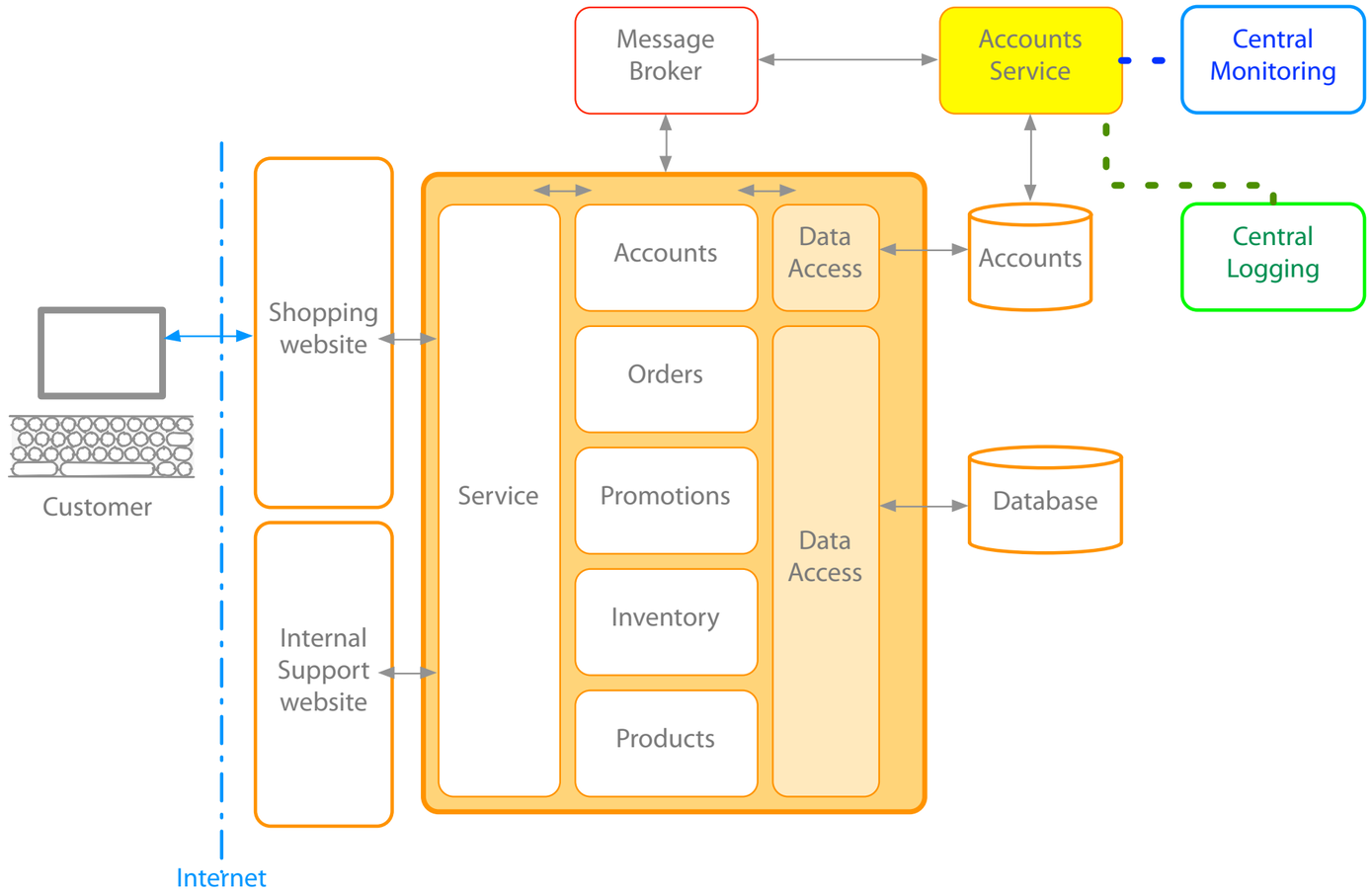
- Existing system
  - Monolithic system
  - Organically grown
  - Seems to large to split
- Lacks microservices design principles
- Identify seams
  - Separation that reflects domains
  - Identify bounded contexts
- Start modularising the bounded contexts
  - Move code incrementally
  - Tidy up a section per release
    - Take your time
    - Existing functionality needs to remain intact
  - Run unit tests and integration tests to validate change
  - Keep reviewing
- Seams are future microservice boundaries

Brownfield Microservices: Migration



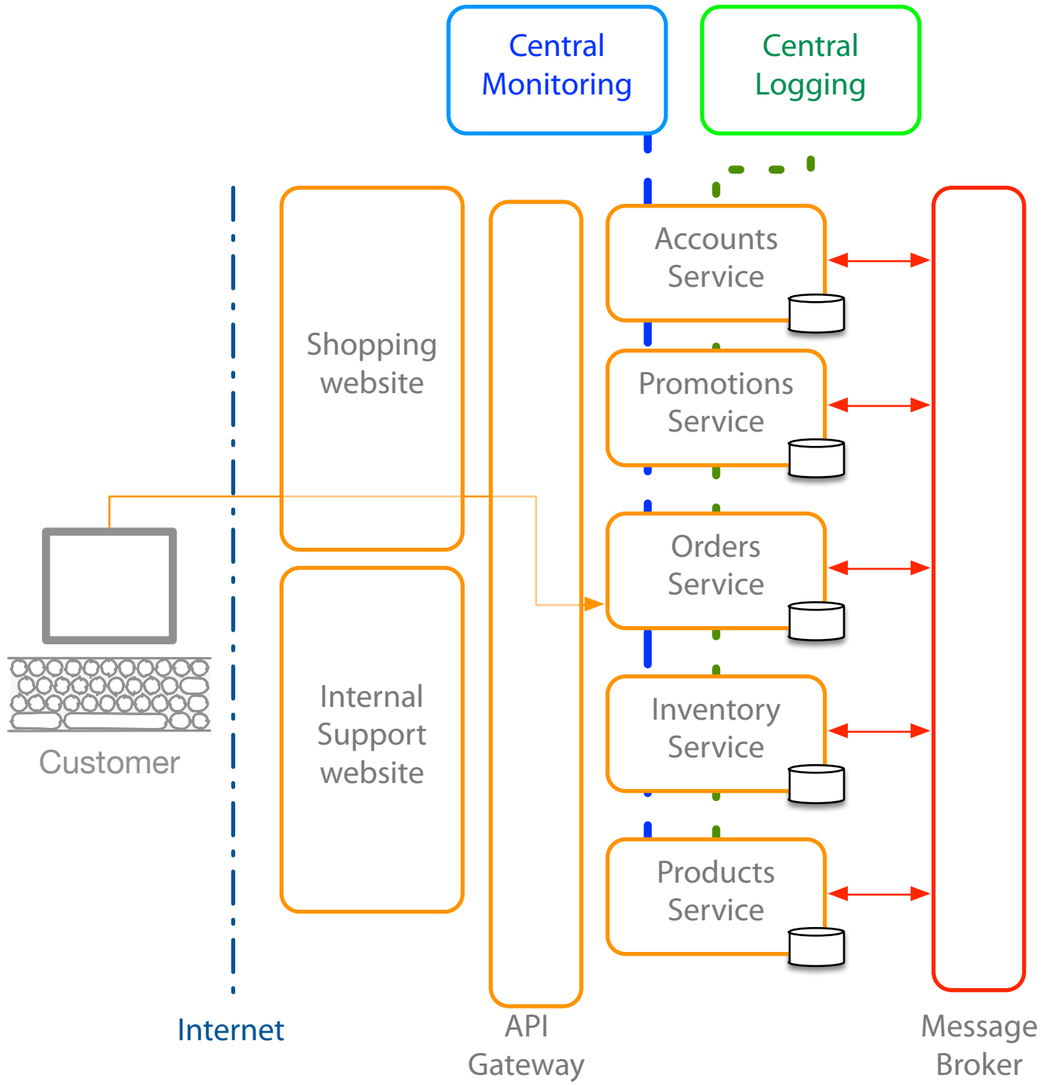
- Code is organised into bounded contexts
  - Code related to a business domain or function is in one place
  - Clear boundaries with clear interfaces between each
- Convert bounded contexts into microservices
  - Start off with one
    - Use to get comfortable
  - Make it switchable
    - Maintain two versions of the code
- How to prioritise what to split?
  - By risk
  - By technology
  - By dependencies
- Incremental approach
- Integrating with the monolithic
  - Monitor both for impact
  - Monitor operations that talk to microservices
  - Review and improve infrastructure
  - Incrementally the monolithic will be converted

Brownfield Microservices: Migration



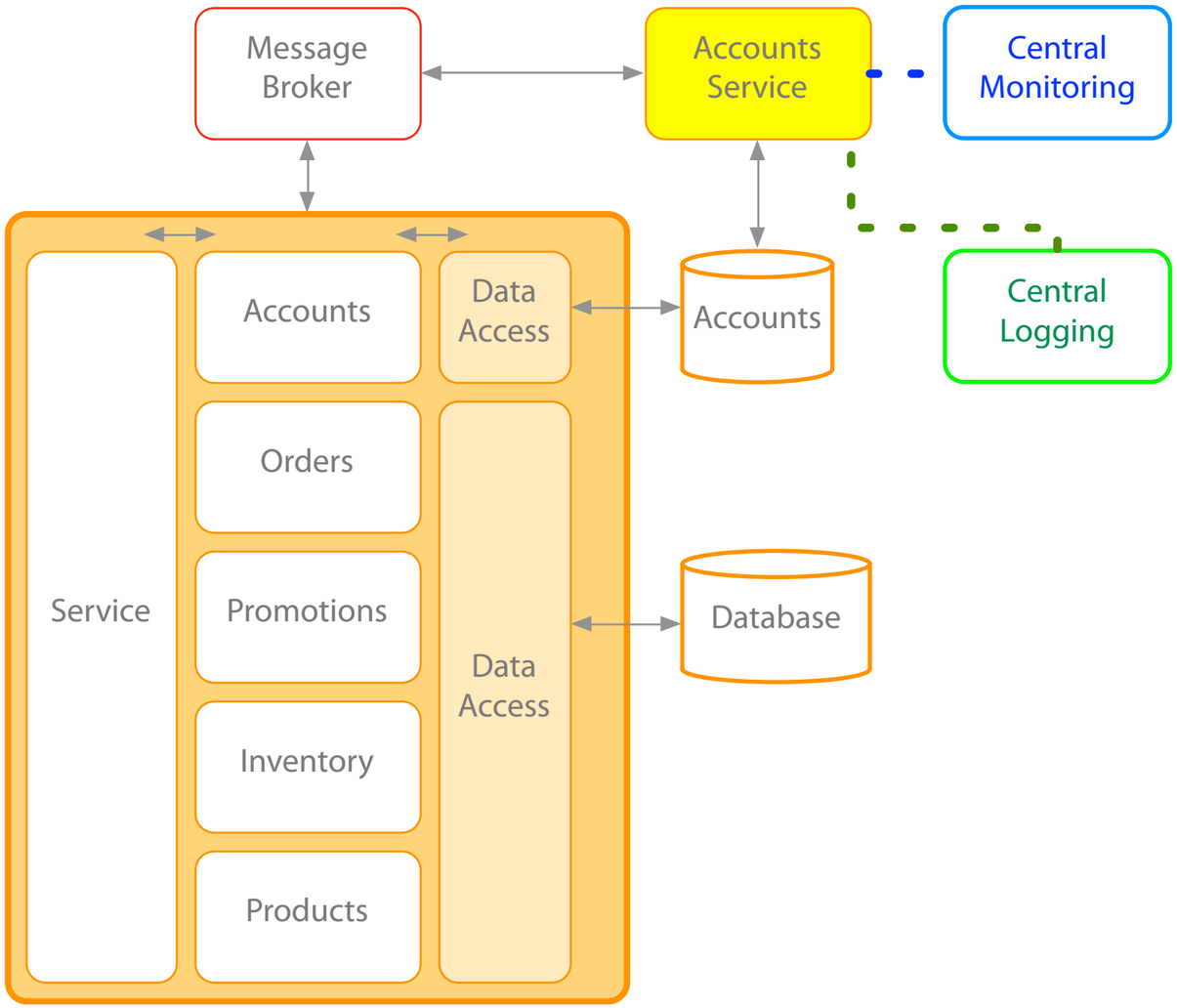
- Code is organised into bounded contexts
  - Code related to a business domain or function is in one place
  - Clear boundaries with clear interfaces between each
- Convert bounded contexts into microservices
  - Start of with one
    - Use to get comfortable
  - Make it switchable
    - Maintain two versions of the code
- How to prioritise what to split?
  - By risk
  - By technology
  - By dependencies
- Incremental approach
- Integrating with the monolithic
  - Monitor both for impact
  - Monitor operations that talk to microservices
  - Review and improve infrastructure
  - Incrementally the monolithic will be converted

Brownfield Microservices: Migration



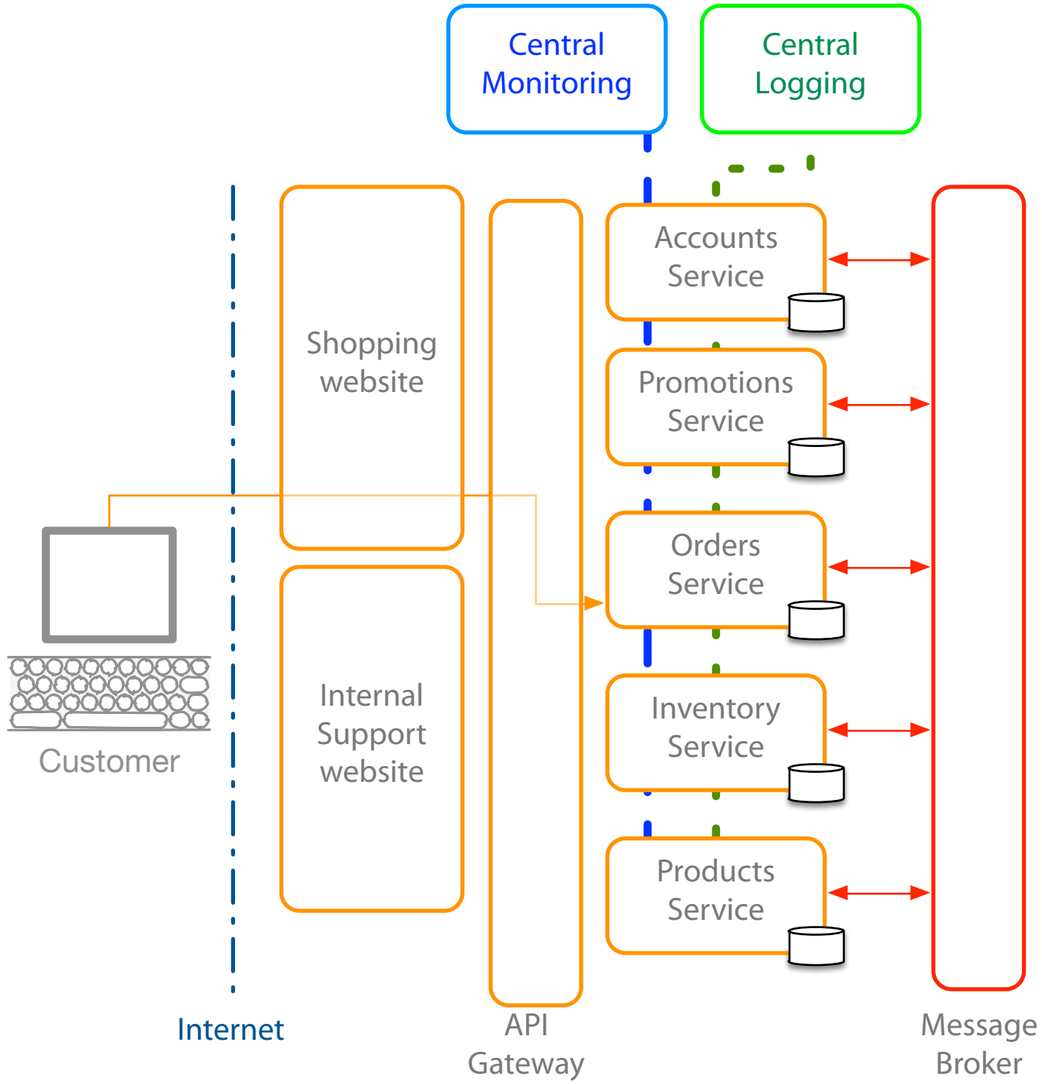
- Code is organised into bounded contexts
  - Code related to a business domain or function is in one place
  - Clear boundaries with clear interfaces between each
- Convert bounded contexts into microservices
  - Start of with one
    - Use to get comfortable
  - Make it switchable
    - Maintain two versions of the code
- How to prioritise what to split?
  - By risk
  - By technology
  - By dependencies
- Incremental approach
- Integrating with the monolithic
  - Monitor both for impact
  - Monitor operations that talk to microservices
  - Review and improve infrastructure
  - Incrementally the monolithic will be converted

Brownfield Microservices: Database Migration



- Avoid shared databases
- Split databases using seams
  - Relate tables to code seams
- Supporting the existing application
  - Data layer that connects to multiple database
- Tables that link across seams
  - API calls that can fetch that data for a relationship
- Refactor database into multiple databases
- Data referential integrity
- Static data tables
- Shared data

Brownfield Microservices: Database Migration



Avoid shared databases

Split databases using seams

- Relate tables to code seams

Supporting the existing application

- Data layer that connects to multiple database

Tables that link across seams

- API calls that can fetch that data for a relationship

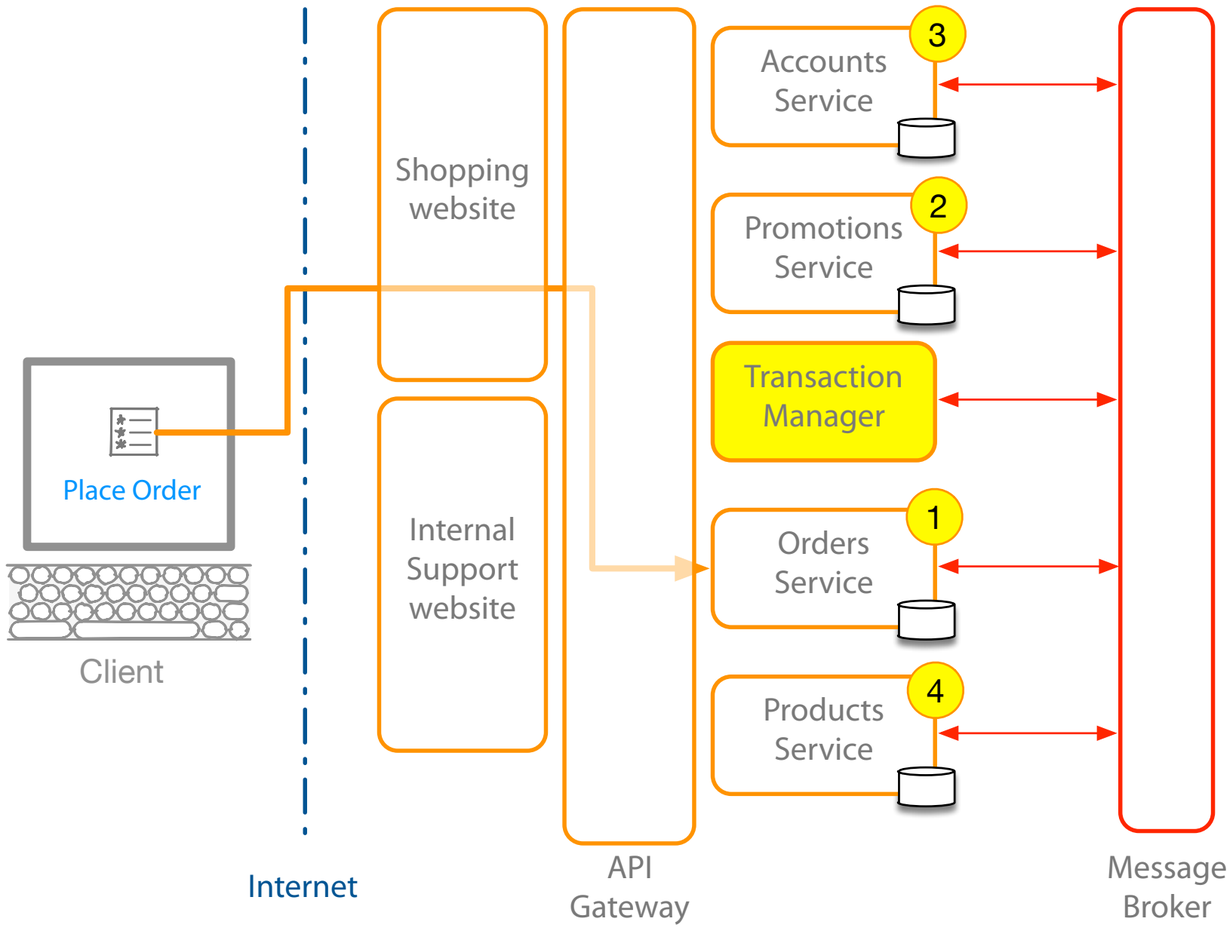
Refactor database into multiple databases

Data referential integrity

Static data tables

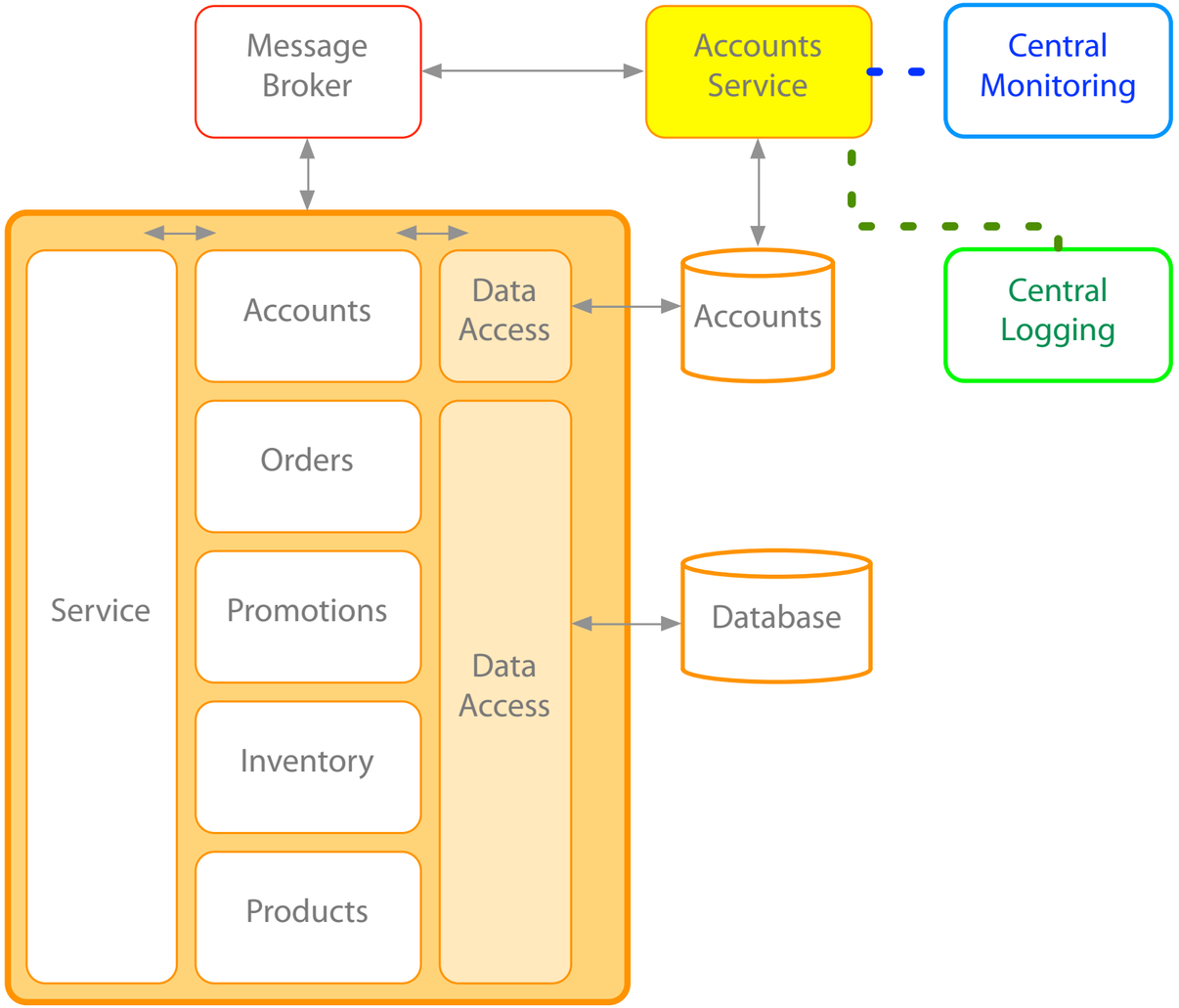
Shared data

Brownfield Microservices: Transactions



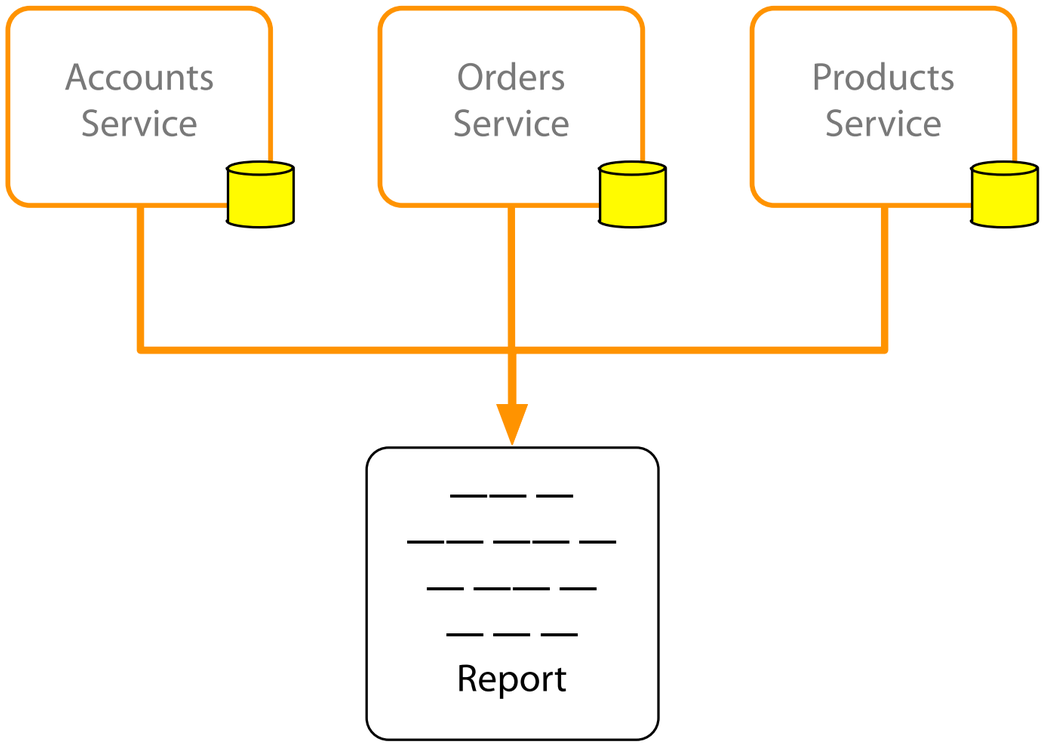
- Transactions ensure data integrity
- Transactions are simple in monolithic applications
- Transactions spanning microservices are complex
  - Complex to observe
  - Complex to problem solve
  - Complex to rollback
- Options for failed transactions
  - Try again later
  - Abort entire transaction
  - Use a transaction manager
    - Two phase commit
  - Disadvantage of transaction manager
    - Reliance on transaction manager
    - Delay in processing
    - Potential bottleneck
    - Complex to implement
- Distributed transaction compatibility
  - Completed message for the monolith

Brownfield Microservices: Transactions



- Transactions ensure data integrity
- Transactions are simple in monolithic applications
- Transactions spanning microservices are complex
  - Complex to observe
  - Complex to problem solve
  - Complex to rollback
- Options for failed transactions
  - Try again later
  - Abort entire transaction
  - Use a transaction manager
    - Two phase commit
  - Disadvantage of transaction manager
    - Reliance on transaction manager
    - Delay in processing
    - Potential bottleneck
    - Complex to implement
- Distributed transaction compatibility
  - Completed message for the monolith

Brownfield Microservices: Reporting



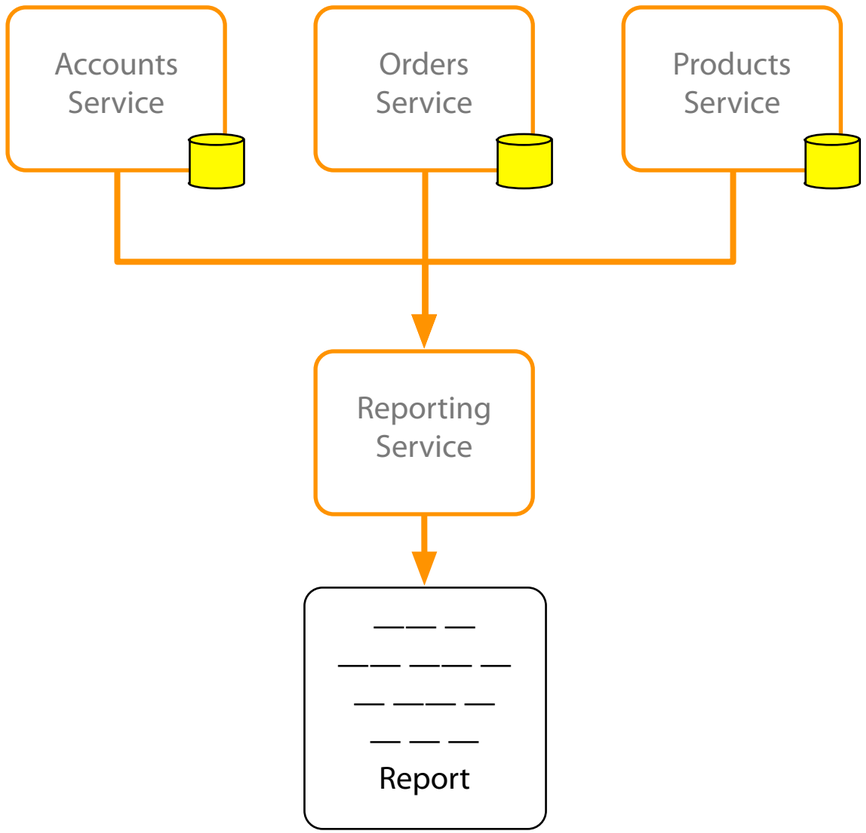
Microservices complicate reporting

- Data split across microservices
- No central database
- Joining data across databases
- Slower reporting
- Complicate report development

Possible solutions

- Service calls for report data
- Data dumps
- Consolidation environment

Brownfield Microservices: Reporting



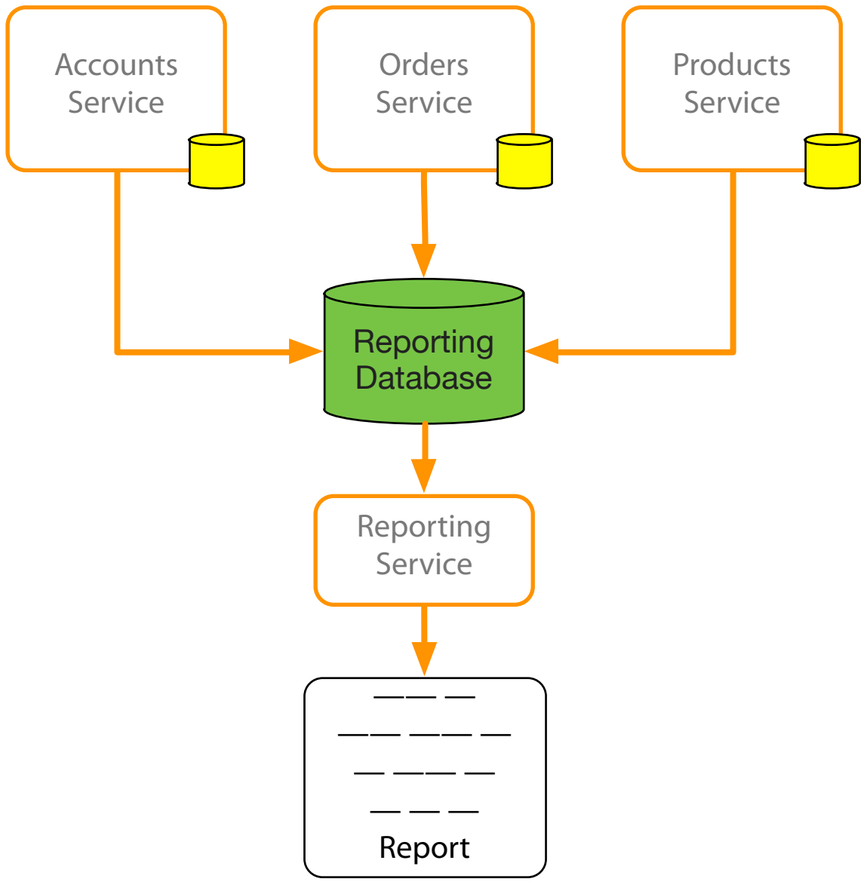
Microservices complicate reporting

- Data split across microservices
- No central database
- Joining data across databases
- Slower reporting
- Complicate report development

Possible solutions

- Service calls for report data
- Data dumps
- Consolidation environment

Brownfield Microservices: Reporting



Microservices complicate reporting

- Data split across microservices
- No central database
- Joining data across databases
- Slower reporting
- Complicate report development

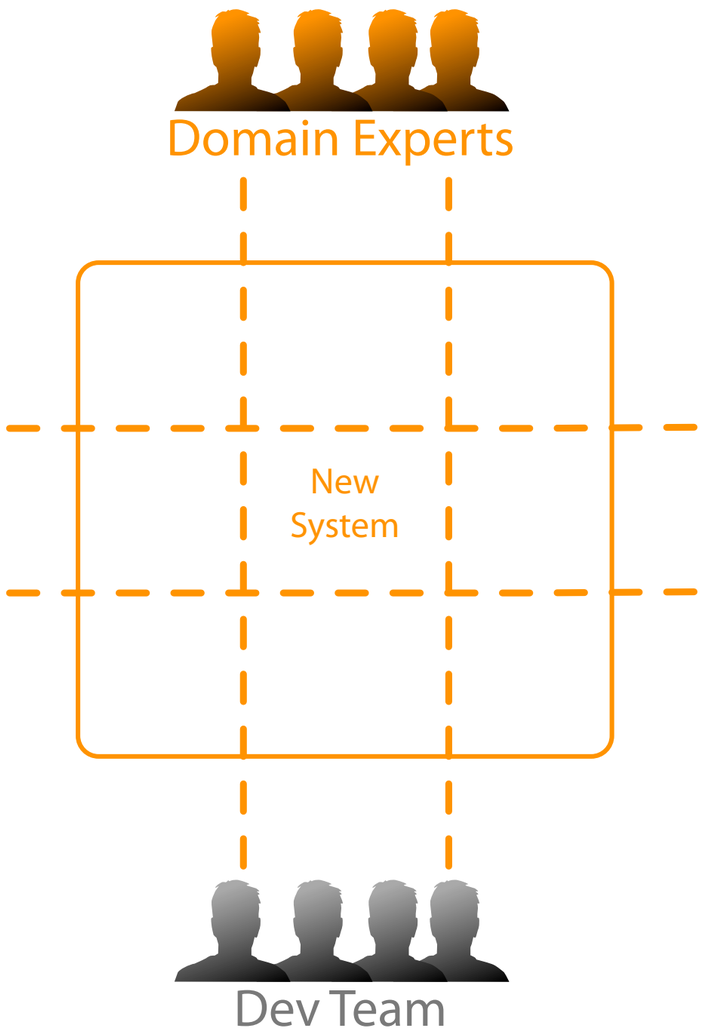
Possible solutions

- Service calls for report data
- Data dumps
- Consolidation environment

Greenfield Microservices

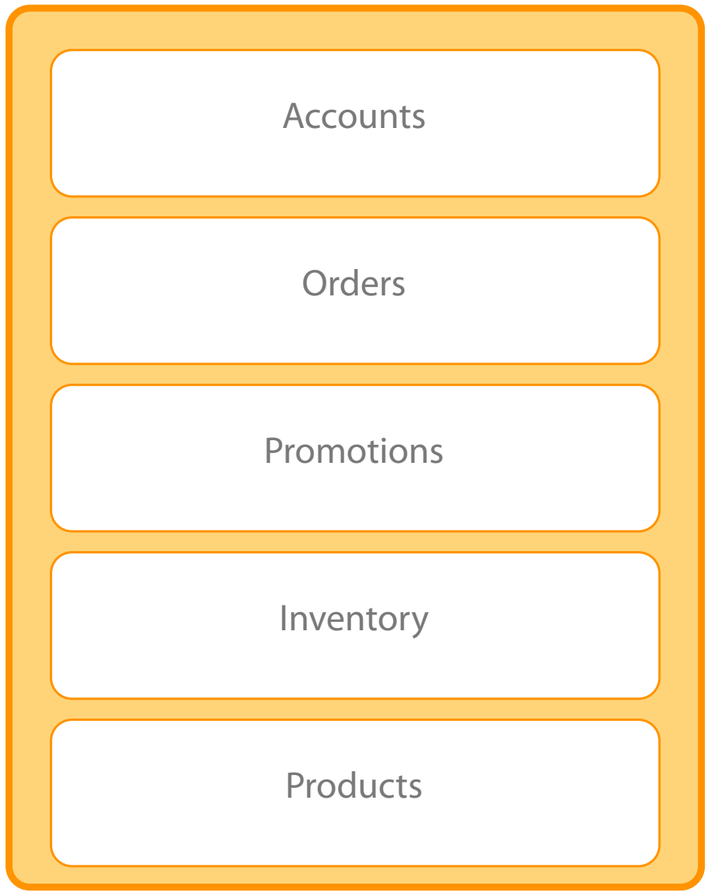
Introduction | Approach

Greenfield Microservices: Introduction



- New project
- Evolving requirements
- Business domain
  - Not fully understood
  - Getting domain experts involved
  - System boundaries will evolve
- Teams experience
  - First microservice
  - Experienced with microservices
- Existing system integration
  - Monolithic system
  - Established microservices architecture
- Push for change
  - Changes to apply microservice principles

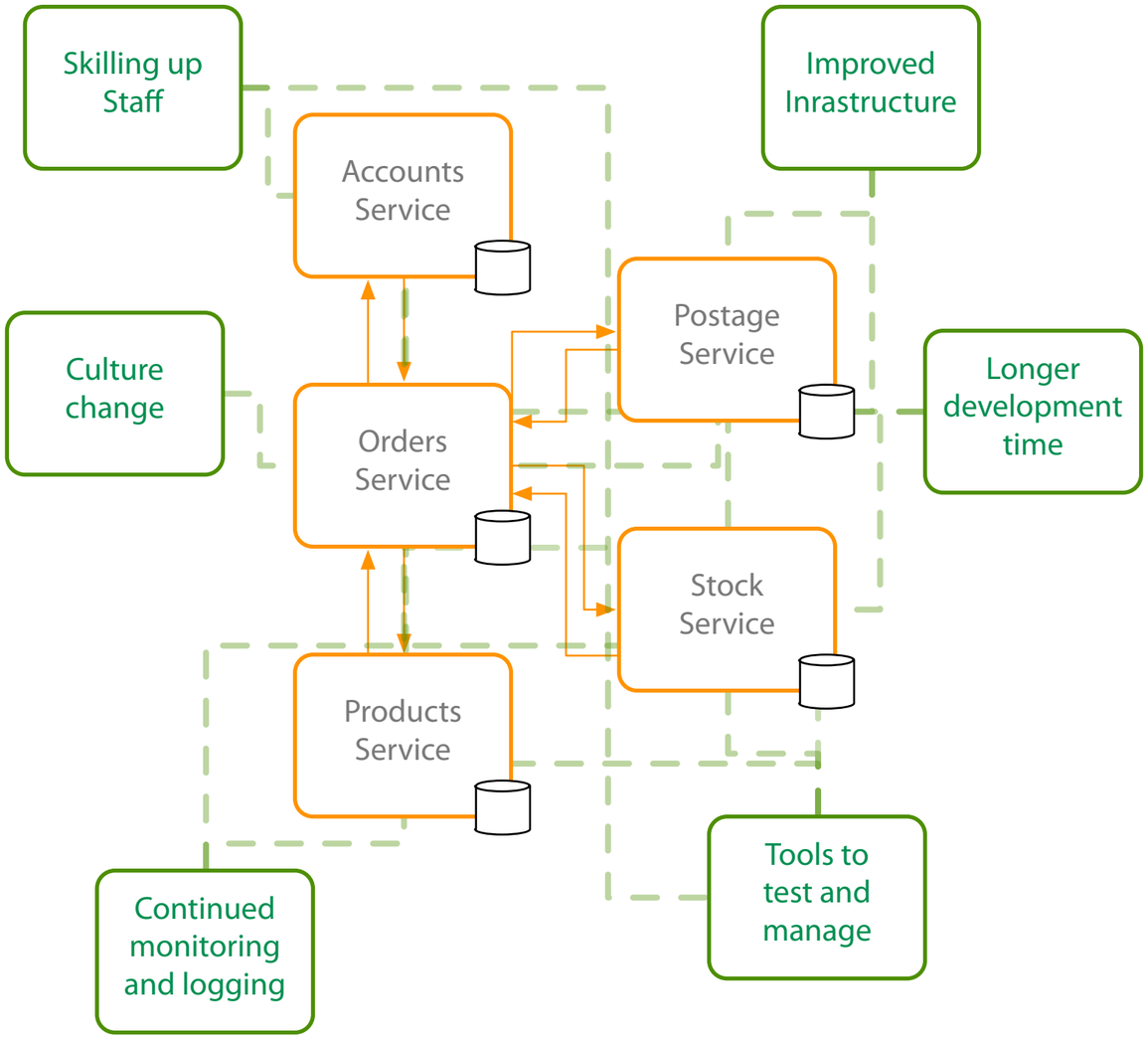
Greenfield Microservices: Approach



- Start off with monolithic design
  - High level
  - Evolving seams
  - Develop areas into modules
  - Boundaries start to become clearer
  - Refine and refactor design
  - Split further when required
- Modules become services
- Shareable code libraries promote to service
- Review microservice principles at each stage
- Prioritise by
  - Minimal viable product
  - Customer needs and demand

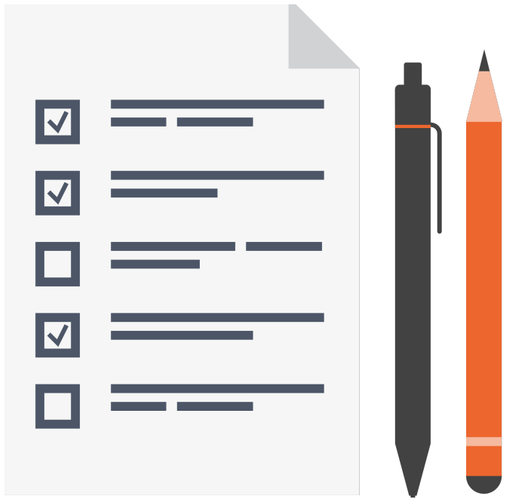
Microservices Provisos

Microservices Provisos



- Accepting initial expense
  - Longer development times
  - Cost and training for tools and new skills
- Skilling up for distributed systems
  - Handling distributed transactions
  - Handling reporting
- Additional testing resource
  - Latency and performance testing
  - Testing for resilience
- Improving infrastructure
  - Security
  - Performance
  - Reliance
- Overhead to mange microservices
- Cloud technologies
- Culture change

Module Summary



Brownfield Microservices

Greenfield Microservices

Microservices Provisos