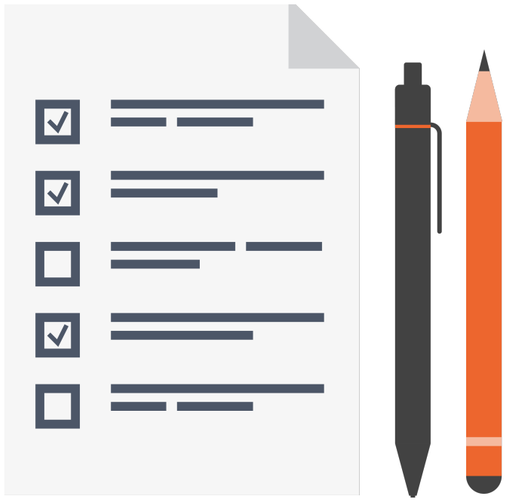
Microservices Design

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

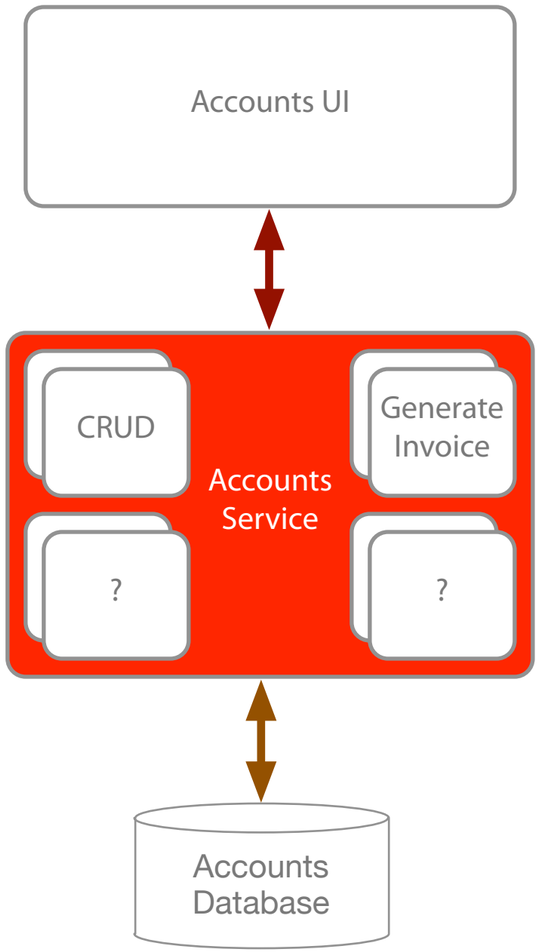
Tools for testing and feedback

Tools for deployment

Microservices Design

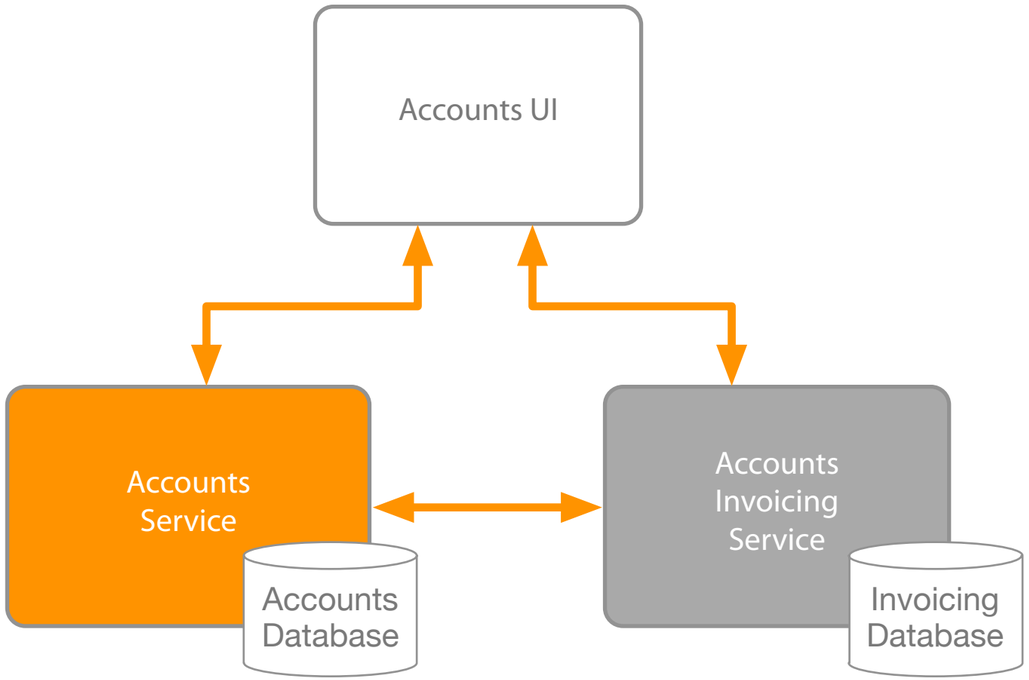
Principles | Approach

Approach: High Cohesion



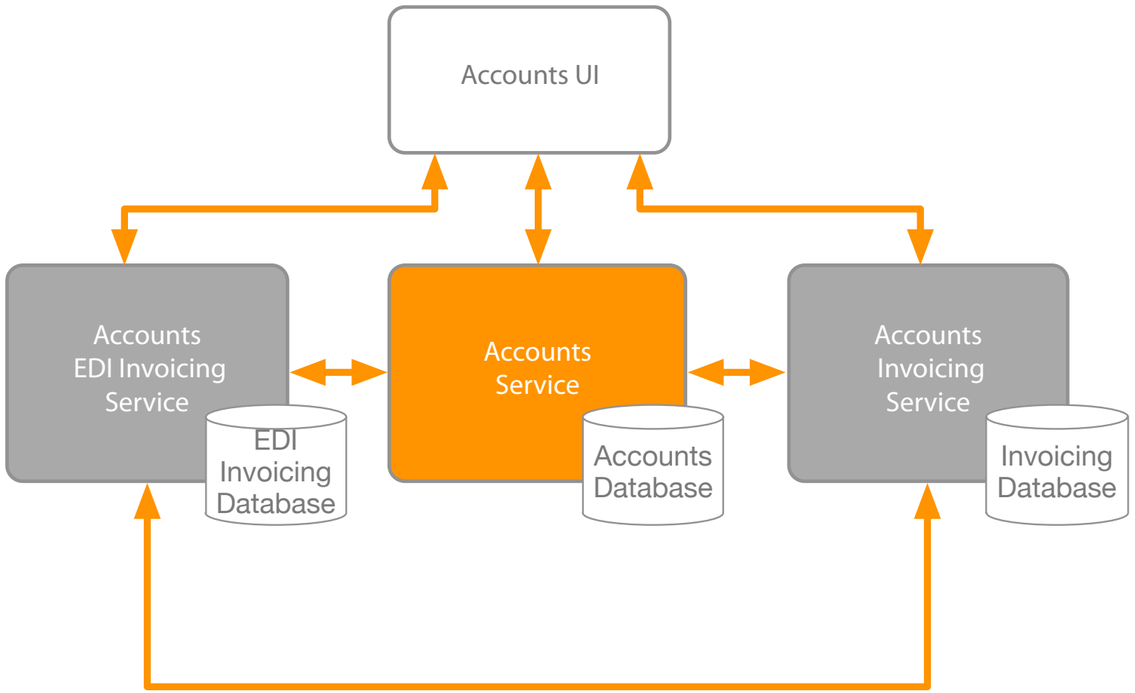
- Identify a single focus
  - Business function
  - Business domain
- Split into finer grained services
- Avoid “Is kind of the same”
- Don't get lazy!
- Don’t be afraid to create many services
- Question in code\peer reviews
  - Can this change for more than one reason

Approach: High Cohesion



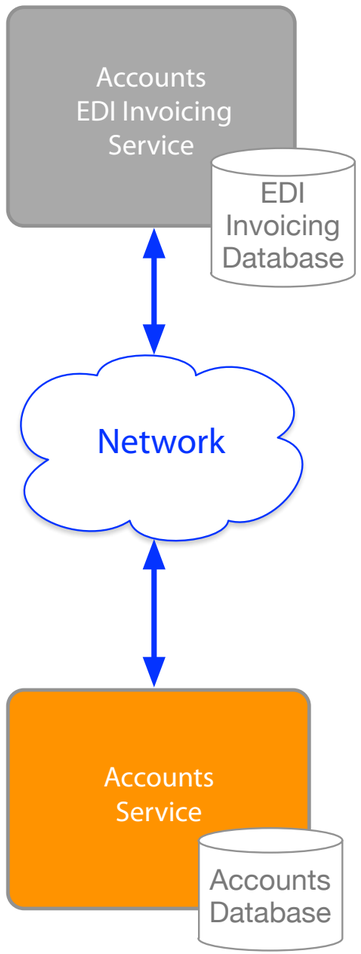
- Identify a single focus
  - Business function
  - Business domain
- Split into finer grained services
- Avoid “Is kind of the same”
- Don't get lazy!
- Don’t be afraid to create many services
- Question in code\peer reviews
  - Can this change for more than one reason

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

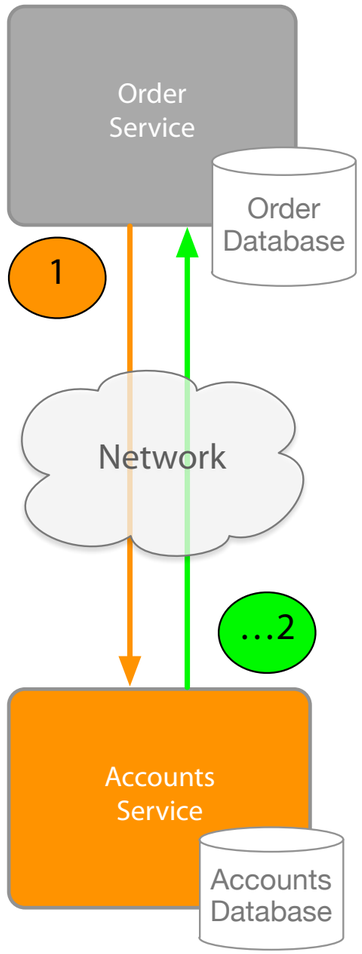
Approach: Autonomous



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

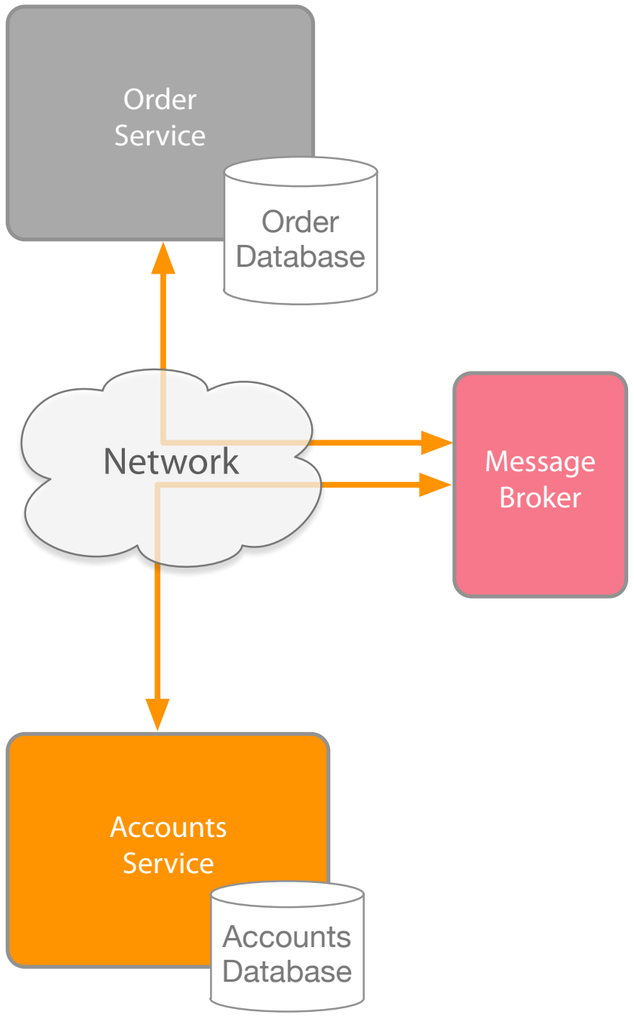
Approach: Autonomous



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

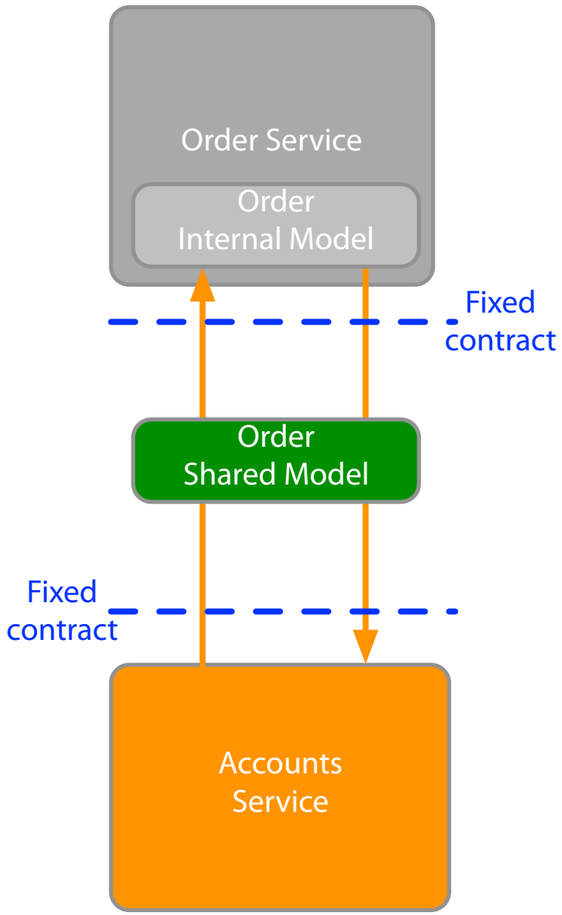
Approach: Autonomous



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

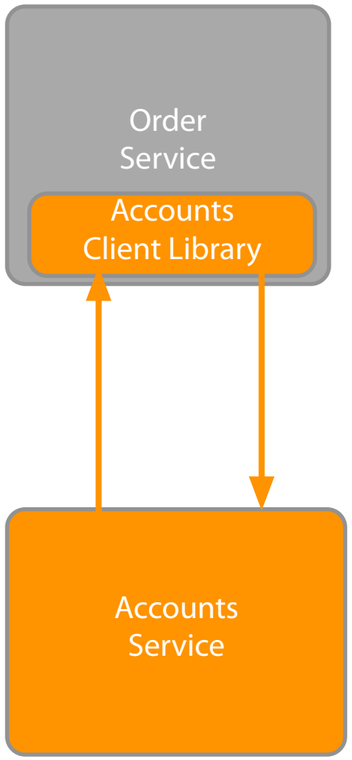
Approach: Autonomous



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

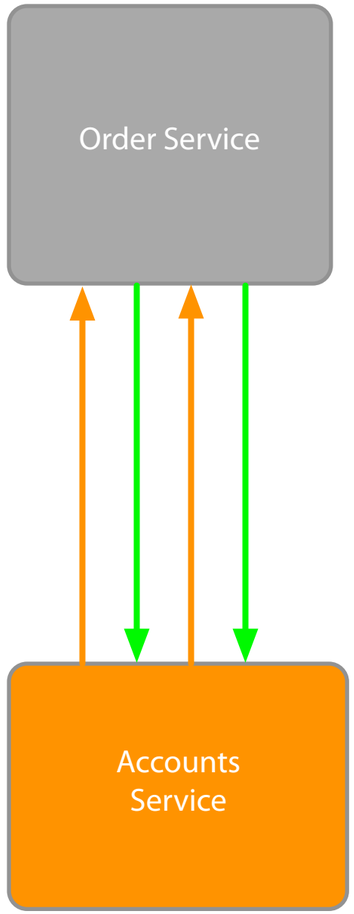
Approach: Autonomous



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

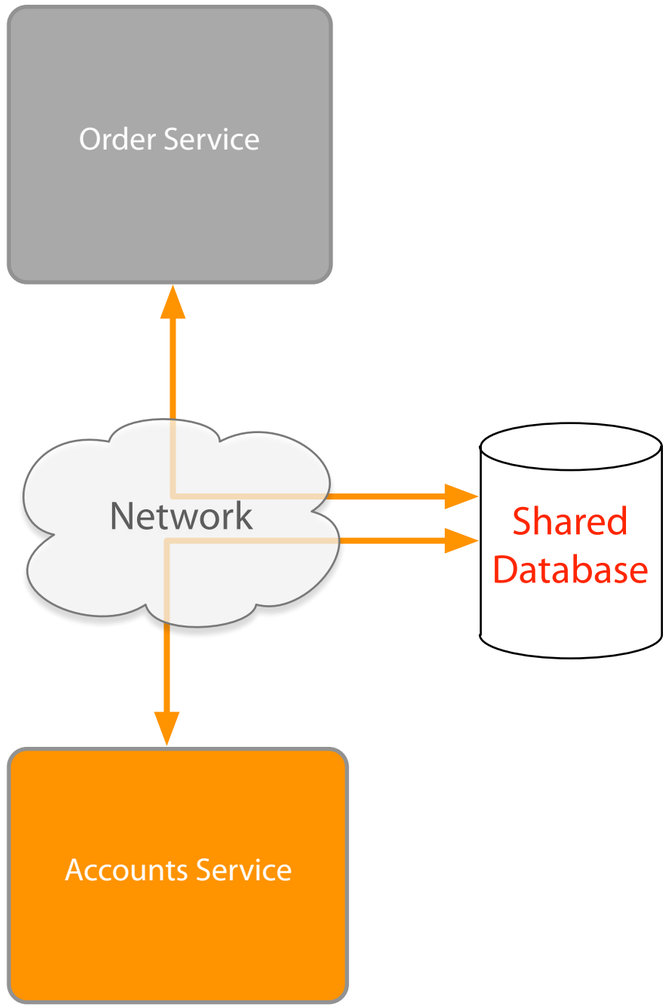
Approach: Autonomous



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

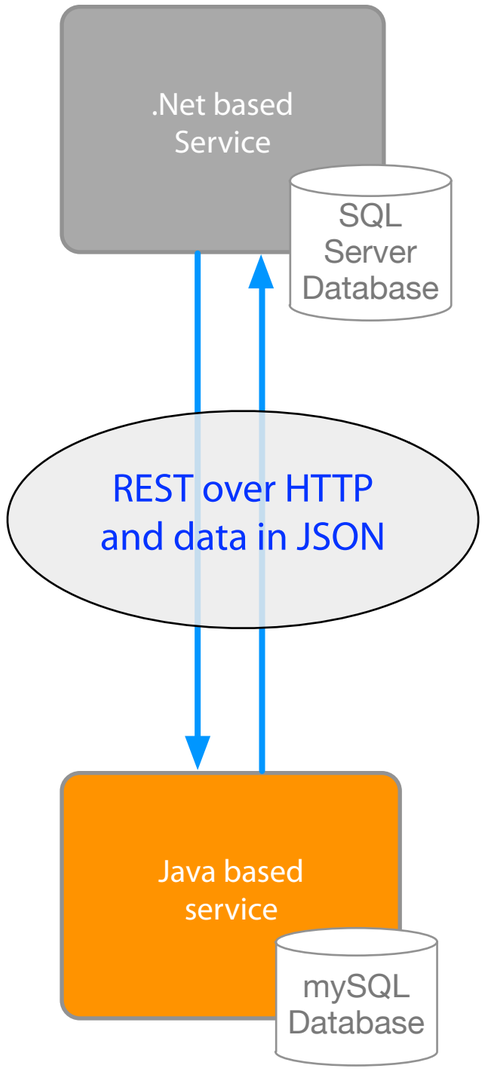
Approach: Autonomous



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

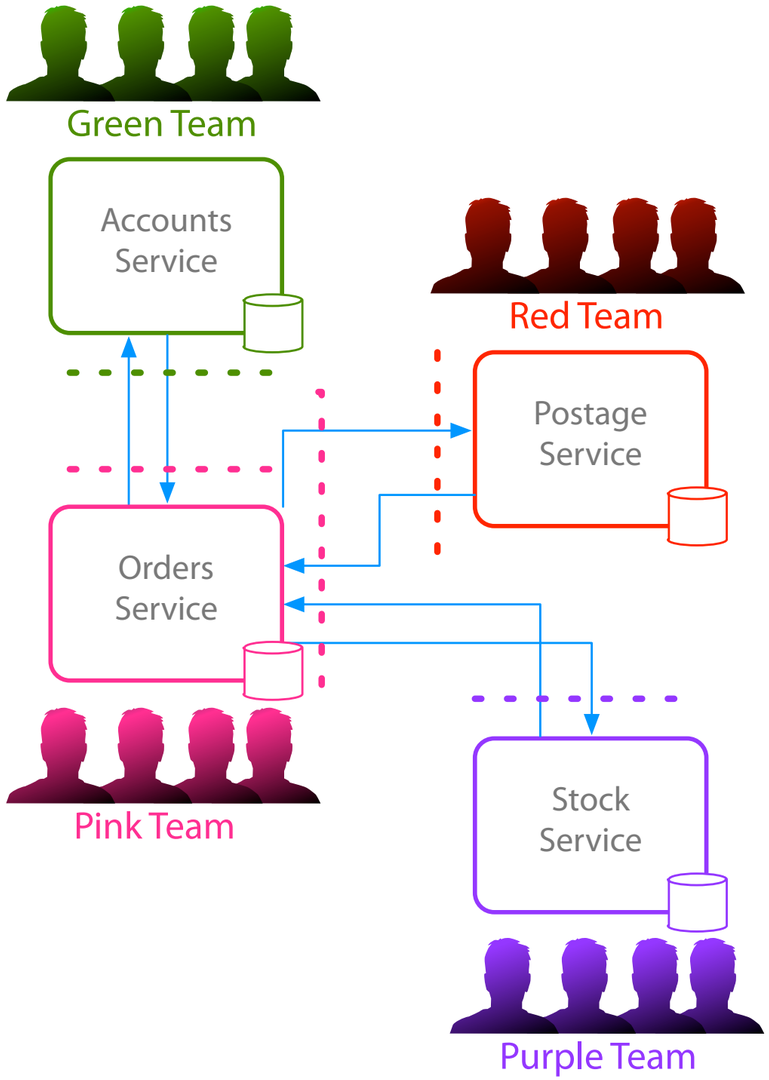
Approach: Autonomous



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



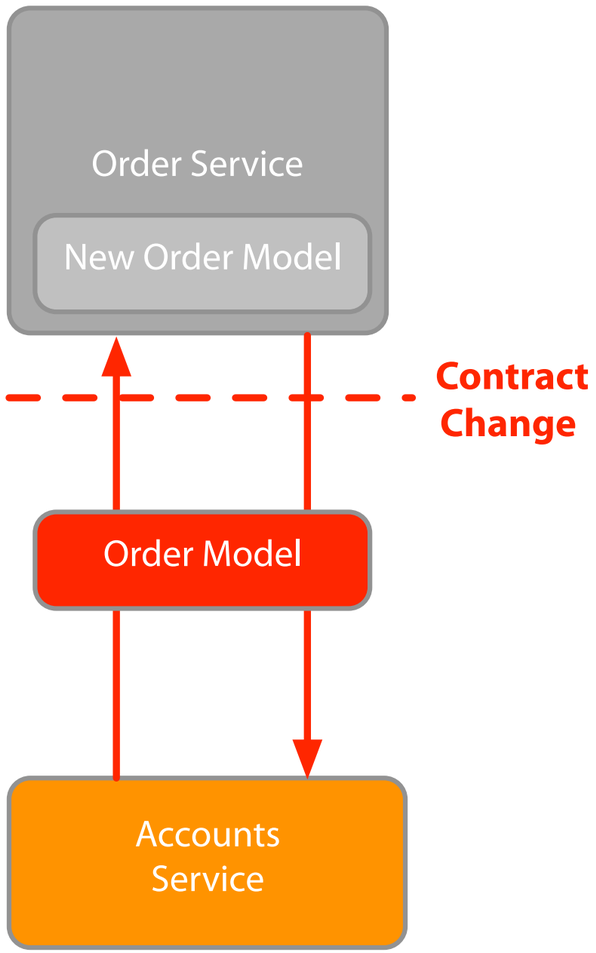
Microservice ownership by team

- 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



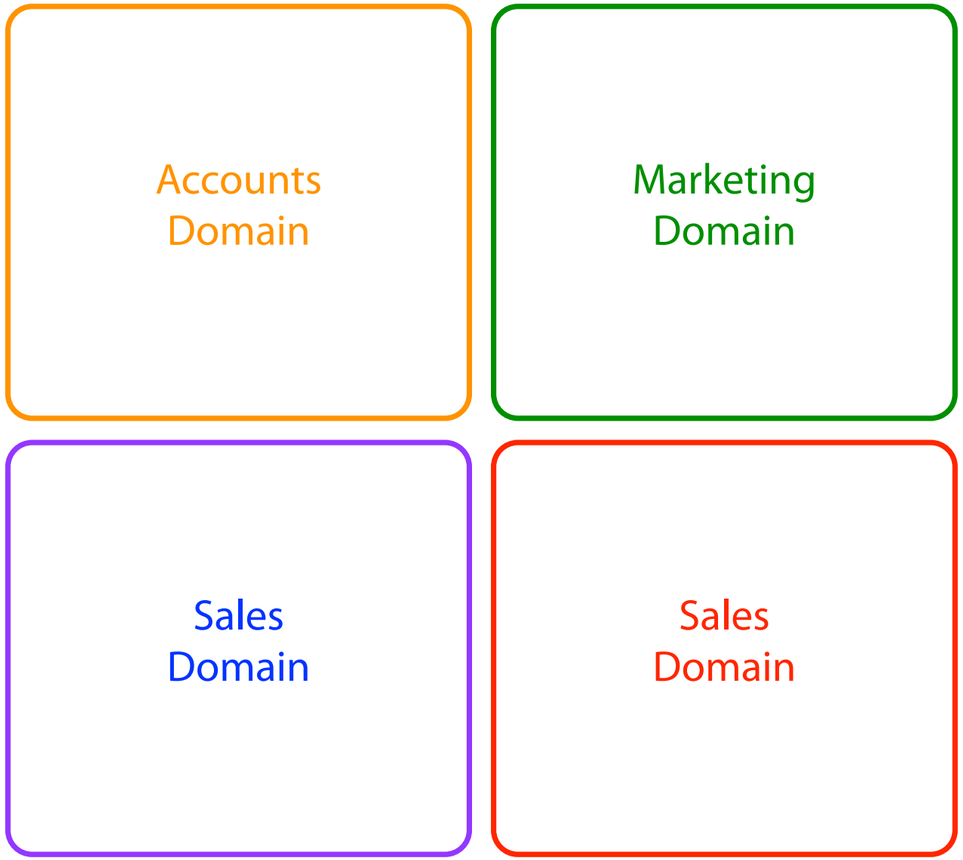
Microservice ownership by team

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

Versioning

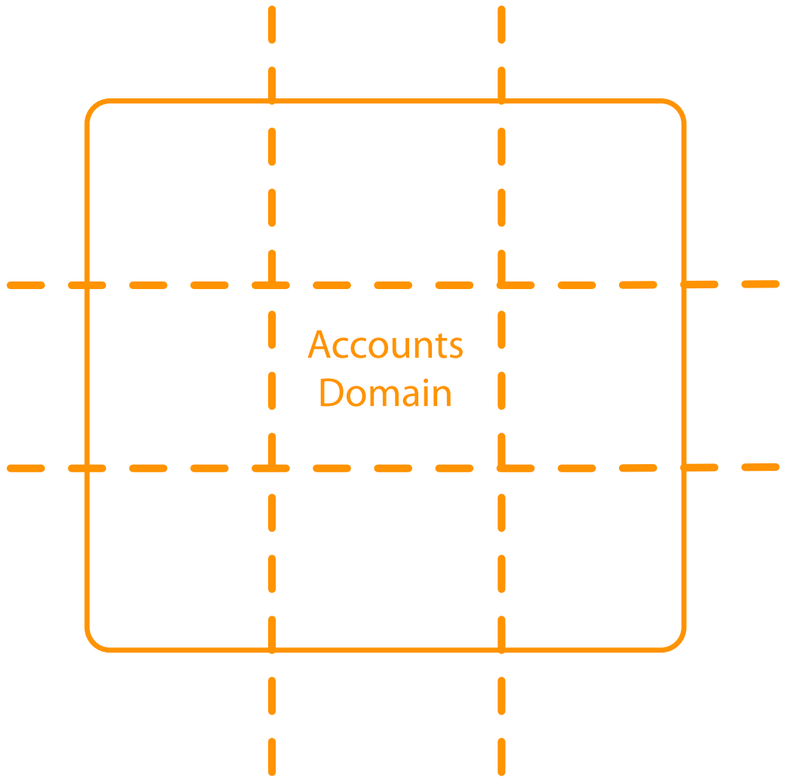
- Avoid breaking changes
- Backwards compatibility
- Integration tests
- Have a versioning strategy
  - Concurrent versions
    - Old and new
  - Semantic versioning
    - Major.Minor.Patch (e.g. 15.1.2)
  - Coexisting endpoints
    - /V2/customer/

Approach: Business Domain Centric



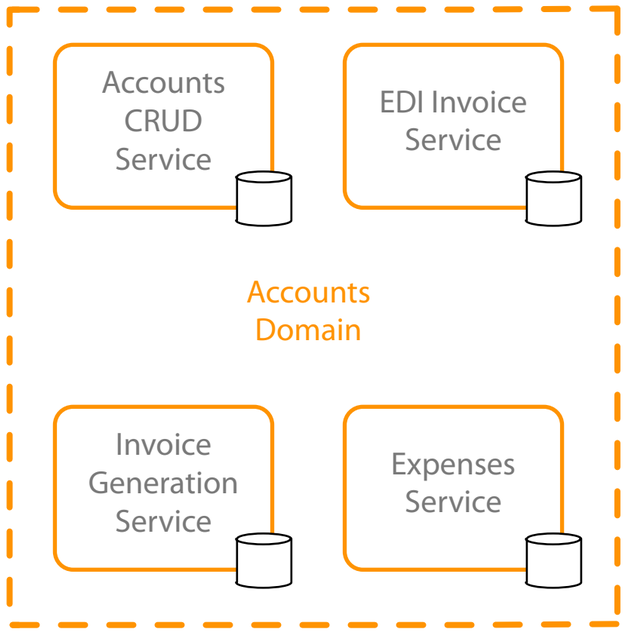
- Business function or business domain
- Approach
  - Identify business domains in a coarse manner
  - Review sub groups of business functions or areas
  - Review benefits of splitting further
  - Agree a common language
- Microservices for data (CRUD) or functions
- Fix incorrect boundaries
  - Merge 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

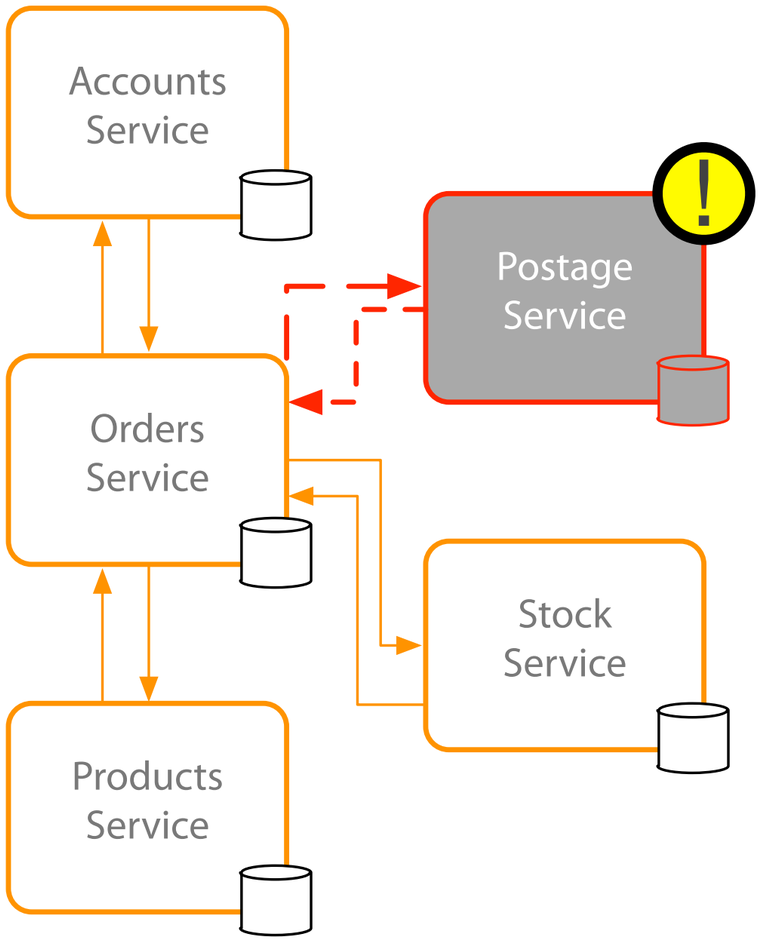
- Merge or split

Explicit interfaces for outside world

Splitting using technical boundaries

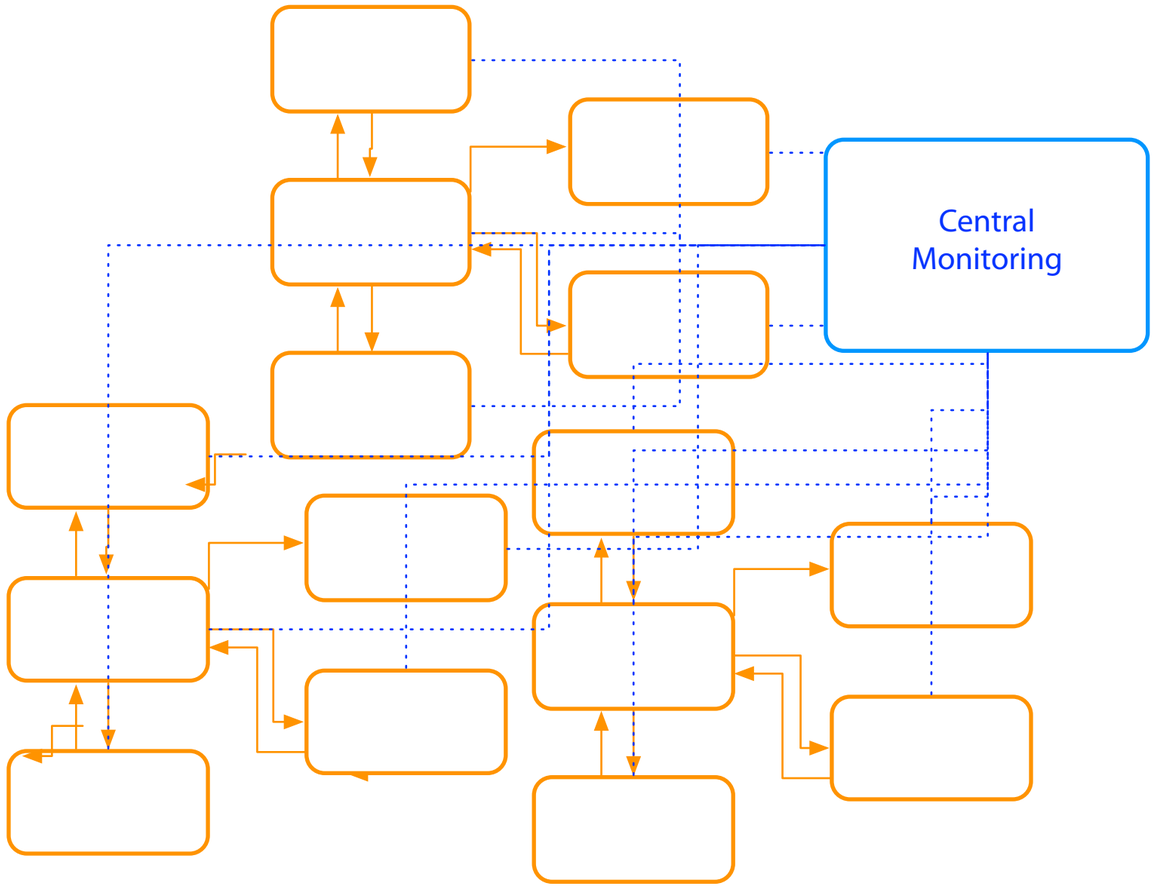
- Service to access archive data
- For performance tuning

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

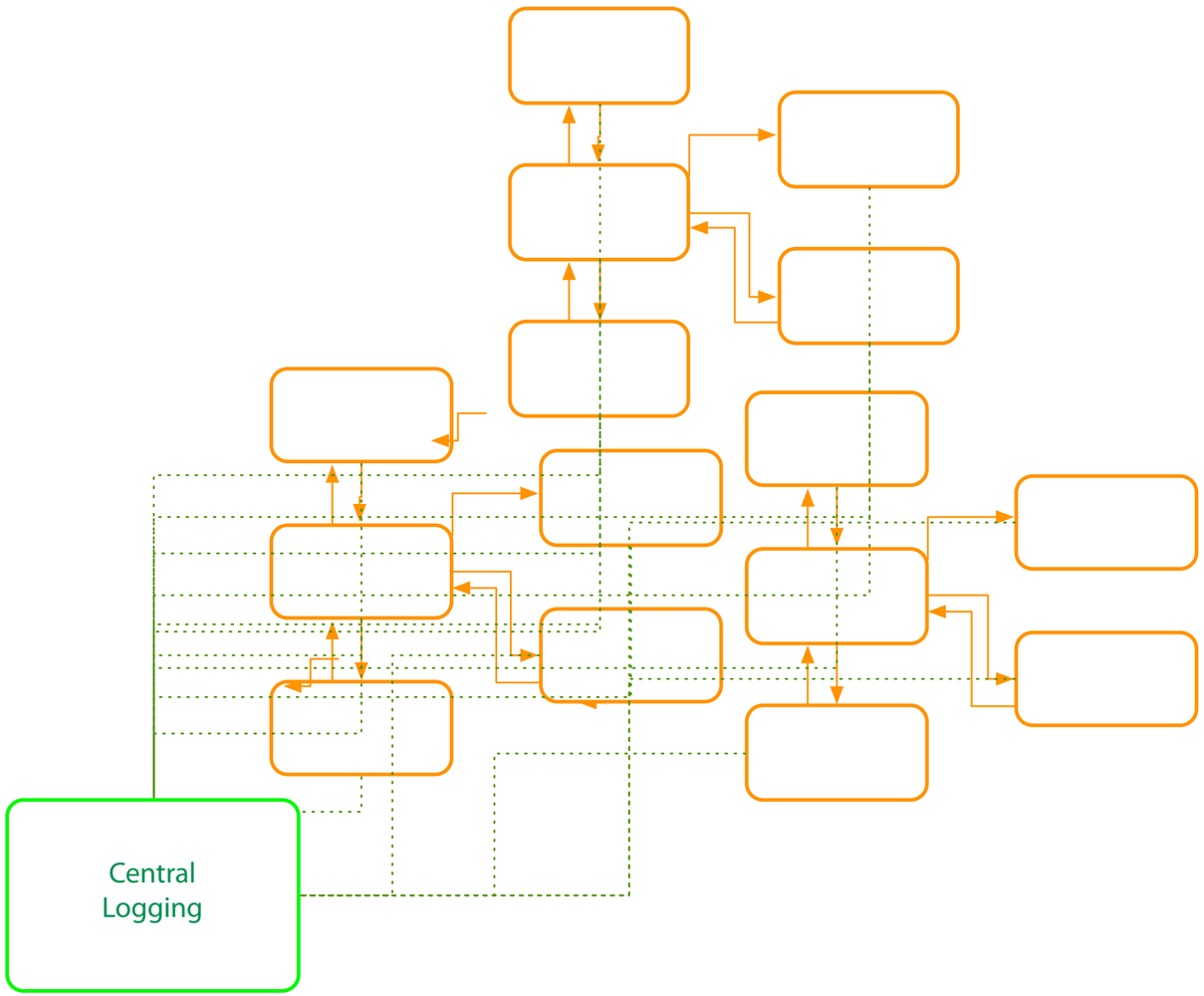
Approach: Observable



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

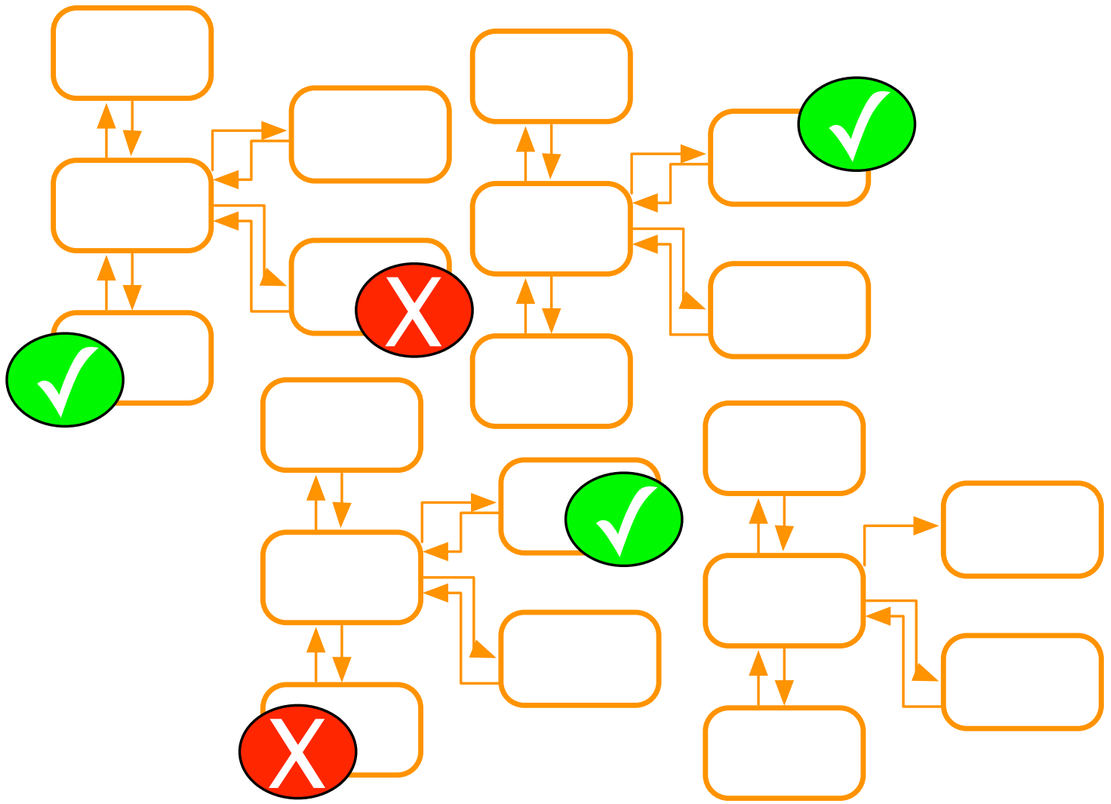
Structured logging

- 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

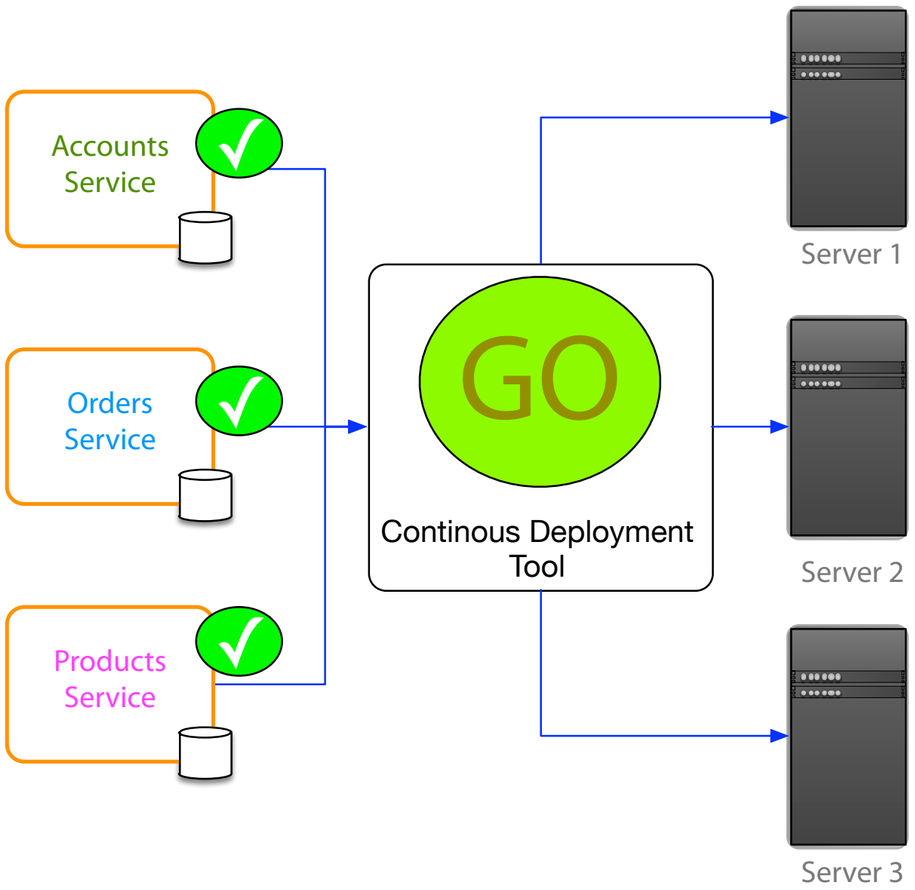
Approach: Automation



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