Microservice Architecture

UNSW SYDNEY

These lecture slides are from the book "*Head First Software Architecture*",

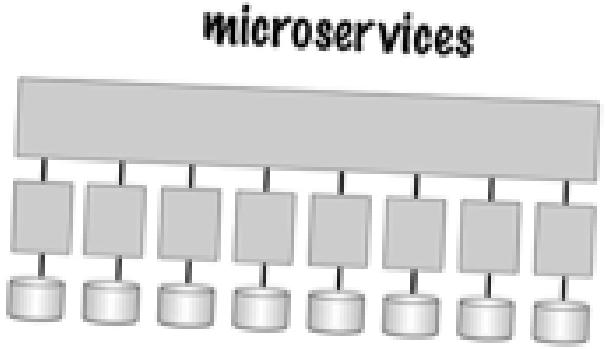
by Raju Gandhi, Mark Richards, Neal Ford, O'Reilly Media, Inc., March 2024

Introduction to Microservices

- ❖ Microservices are single-purpose, independently deployed units.
- ❖ Ideal for environments requiring frequent changes and scalability.

Examples:

- Netflix's streaming services
- Amazon's product catalogue.



Defining Microservices

- Performs one specific function exceptionally well.

Examples:

- Dedicated microservice like "*Monitor Heart Rate.*"
- "Authenticate User" service, "*Generate Invoice*" service.
- "User Profile Management" service.
- "Shopping Cart" service.
- "Notification and Alert" service.
- "Recommendation Engine" service (e.g., Netflix recommendations).

Exercise: Define Microservices

Identify single-purpose microservices below:

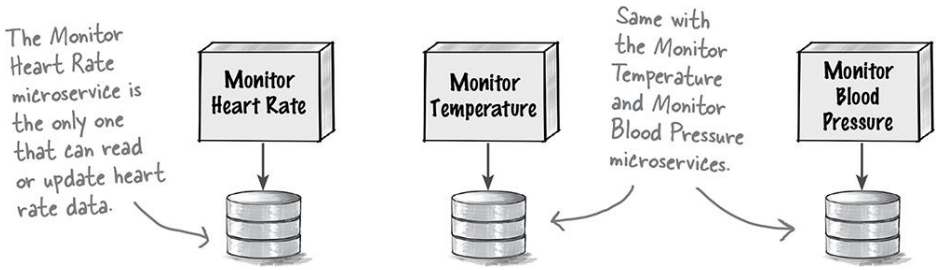
- ❑ Add a movie to your personal “to watch” list
- ❑ Pay for an order using your credit card
- ❑ Generate sales-forecasting and financial-performance reports
- ❑ Submit and process a loan application to get that new car you’ve always wanted
- ❑ Determining the shipping cost for an online order

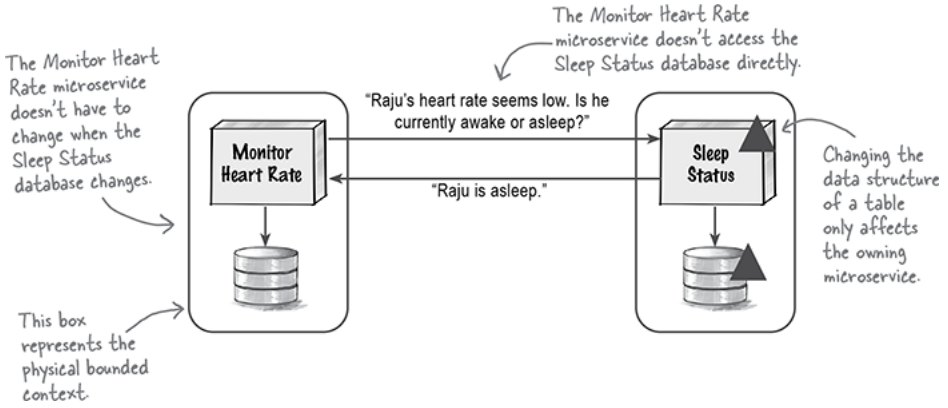
Key Characteristics of Microservices

- Own their own data (Physical bounded contexts).
- Direct data access restricted to owning microservice.

Examples:

- Order service maintains its own order history database.
- Inventory service owns and manages product availability data.
- Payment service manages transaction records independently.
- User Authentication service securely stores user credentials separately.



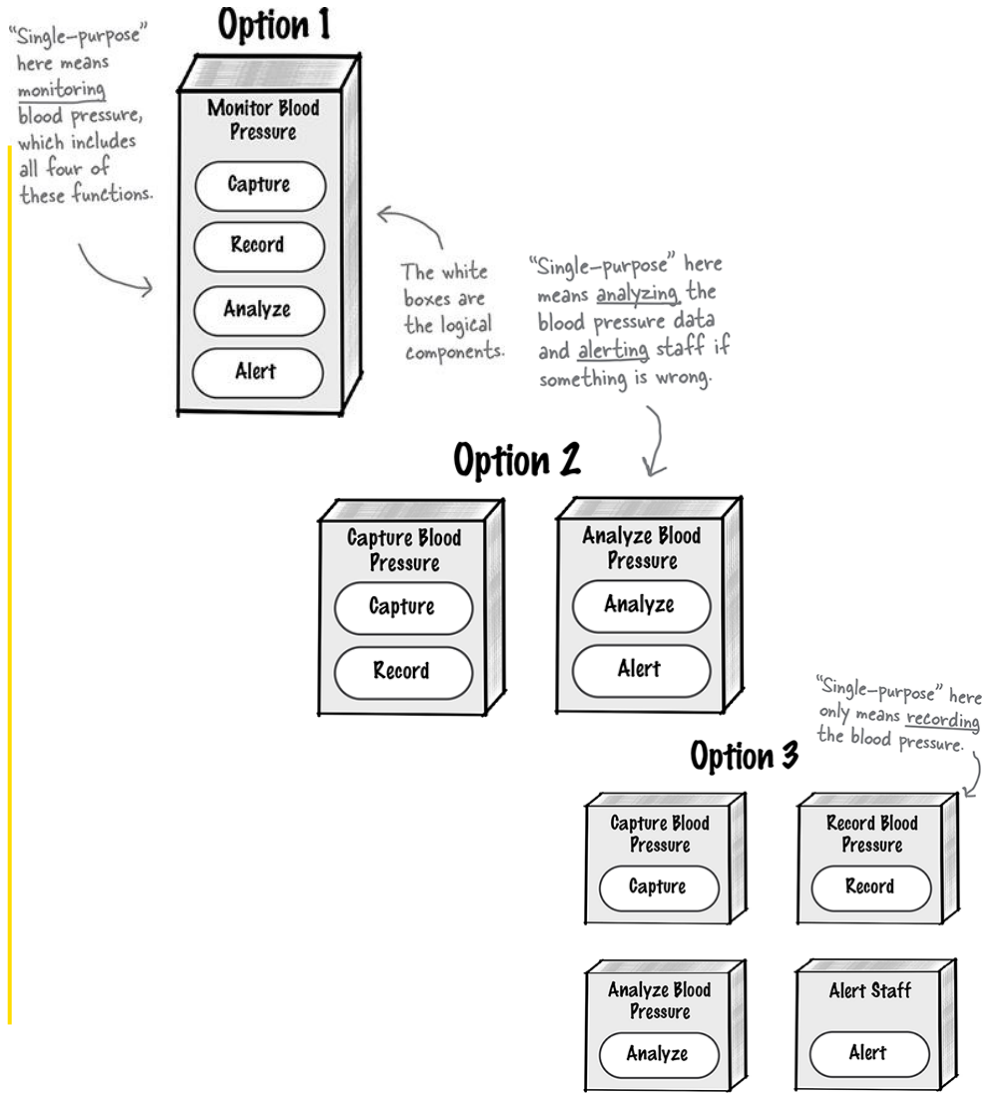


Determining Granularity

- Granularity: The scope of a microservice's responsibility.
- Avoid too fine-grained ("Grains of Sand" antipattern).

Examples:

- Single microservice handling payment transactions.
- A microservice dedicated to shipping and tracking orders.
- Product review and rating as a distinct service separate from product information.
- User notification service isolated from user profile management



Granularity Disintegrators
(Reasons to Make Services Smaller)

Cohesion: Functions within a service should be closely related.

- Payment processing separate from user authentication.

Fault Tolerance: Separating unstable functions for better reliability.

- Isolating an unstable email notification service.

Access Control: Easier management of sensitive data.

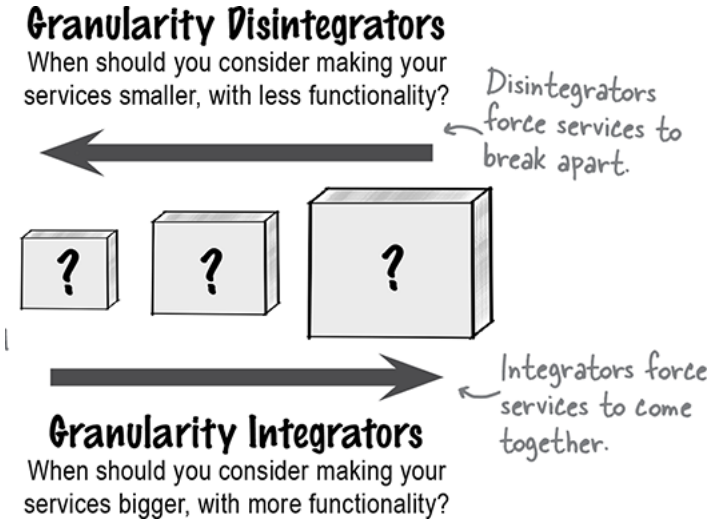
- Isolating financial data access.

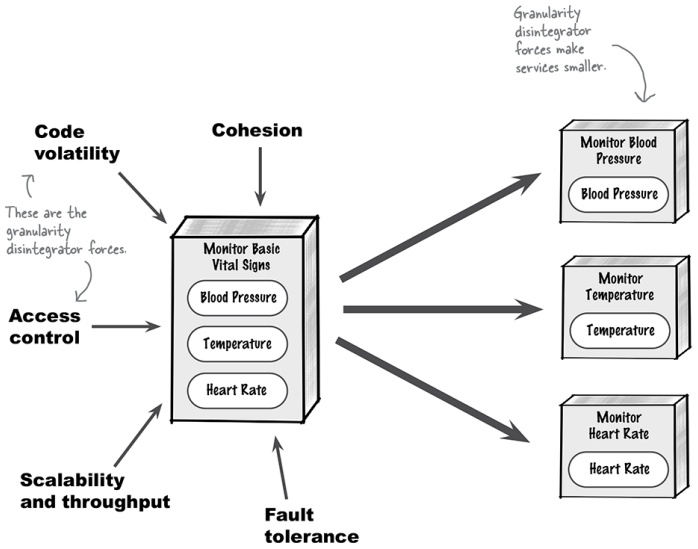
Code Volatility: Isolating frequently changing parts.

- User interface components separated from stable backend logic.

Scalability: Independent scaling for high-demand components.

- High-traffic "search" feature isolated for scaling.

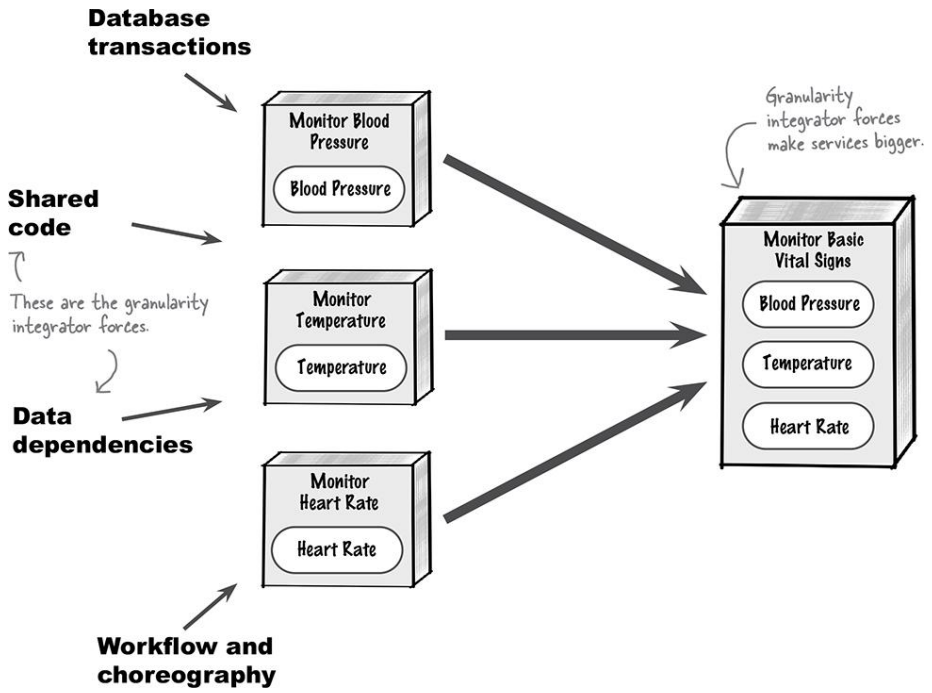




UNSW SYDNEY

Granularity Integrators
(Reasons to Make Services Larger)

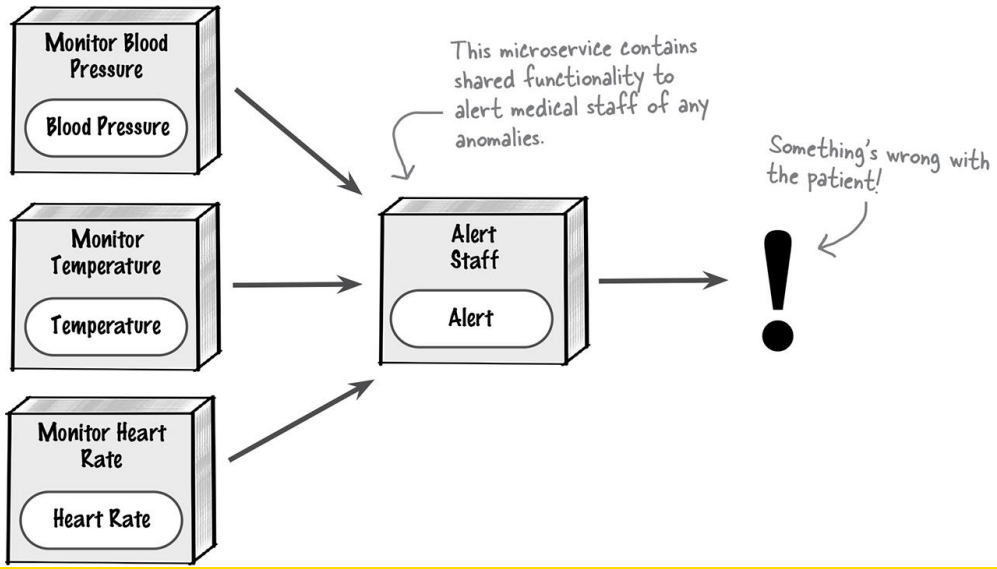
- Database Transactions: Easier to manage single commit/rollback operations.
  - Order creation and inventory deduction in one service.
- Data Dependencies: Maintain tightly coupled data together.
  - User profiles and preferences managed together.
- Workflow Efficiency: Reduce excessive inter-service communication.
  - Checkout service combining cart, pricing, and payment functionalities.



It’s about a right balance!

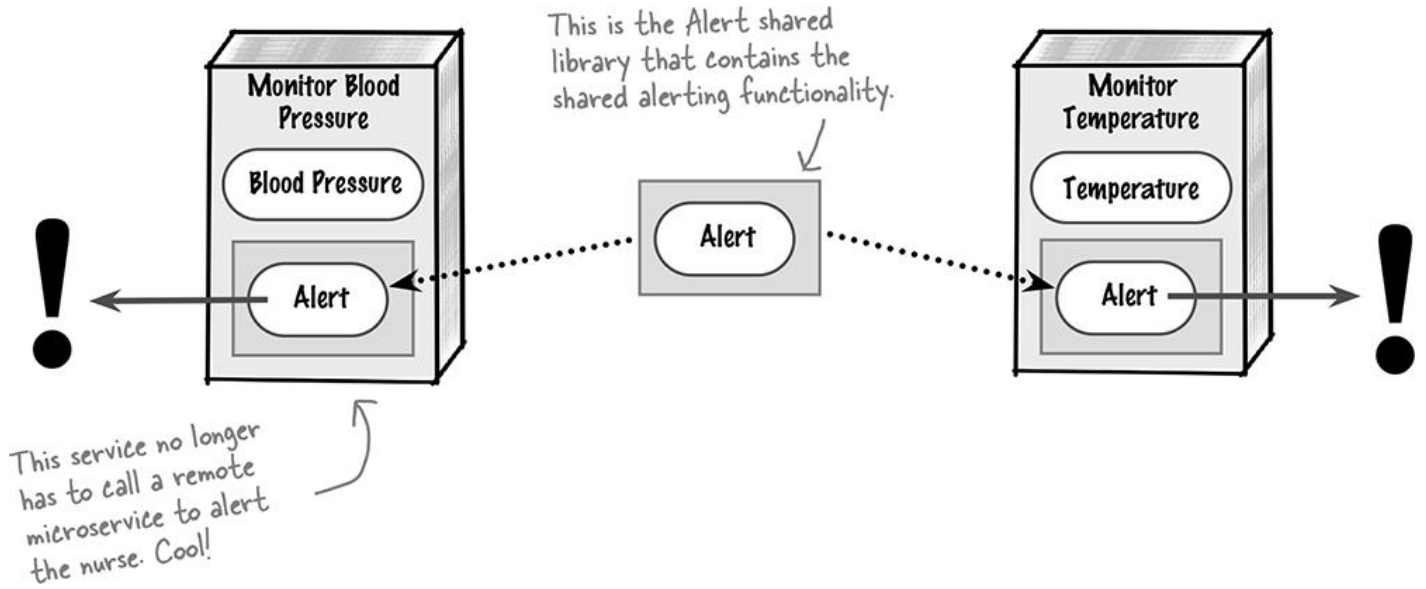
Sharing Functionality

- Shared Services: Standalone microservices accessed remotely.
  - Authentication service used by multiple microservices.
  - Shared alert functionality in *MonitorMe* medical alerts



Sharing Functionality

- Shared Libraries: Embedded at compile-time, deployed with each service.
  - Logging and error handling libraries.



Shared Services vs. Shared Libraries

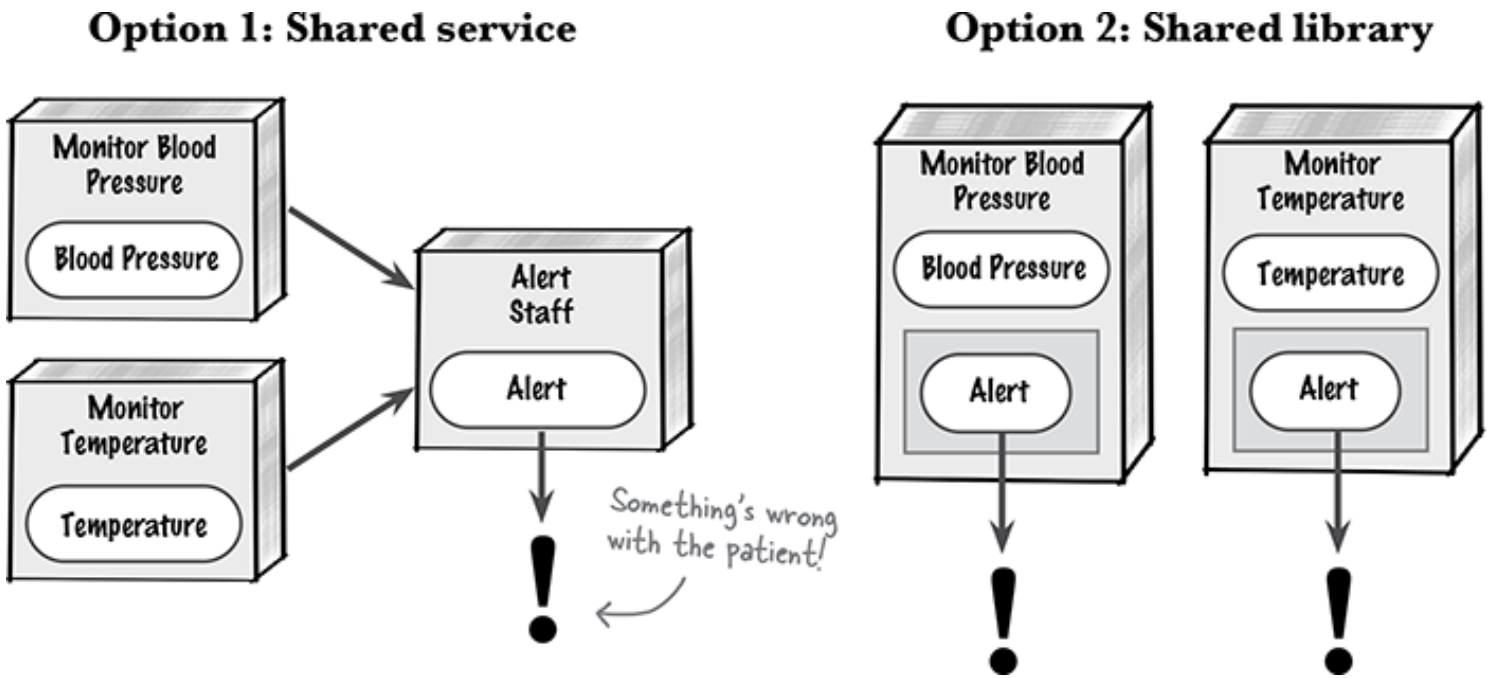
- Services: Agile, suitable for diverse environments, slower, less fault-tolerant.
  - Central user authentication service.

- Libraries: Faster, scalable, robust, but challenging dependency management.
  - JSON parsing libraries used across multiple microservices.

Exercise

Should the alert functionality in *MonitorMe* be a shared library or a shared service?

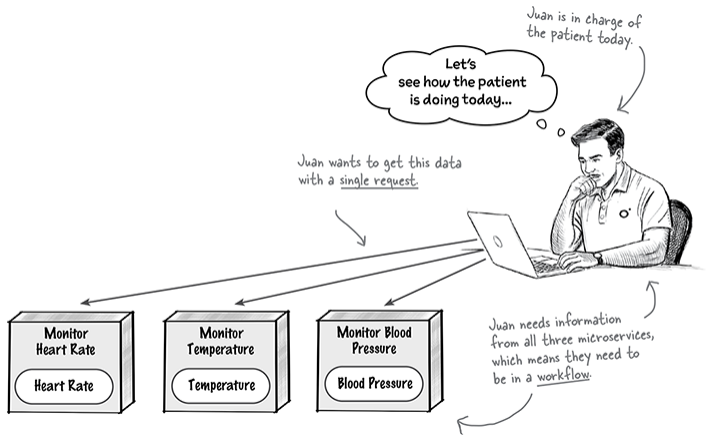
- Justify your decision.

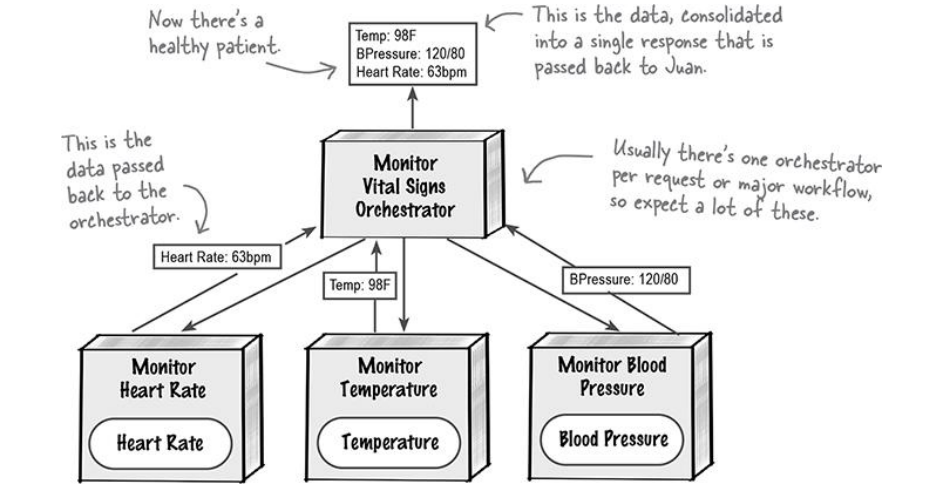


Workflow Management: Orchestration

- Central orchestration manages workflow, akin to a symphony conductor.
  - Pros: Centralized management, clear state/error handling.
  - Cons: Bottlenecks, high coupling, performance concerns.

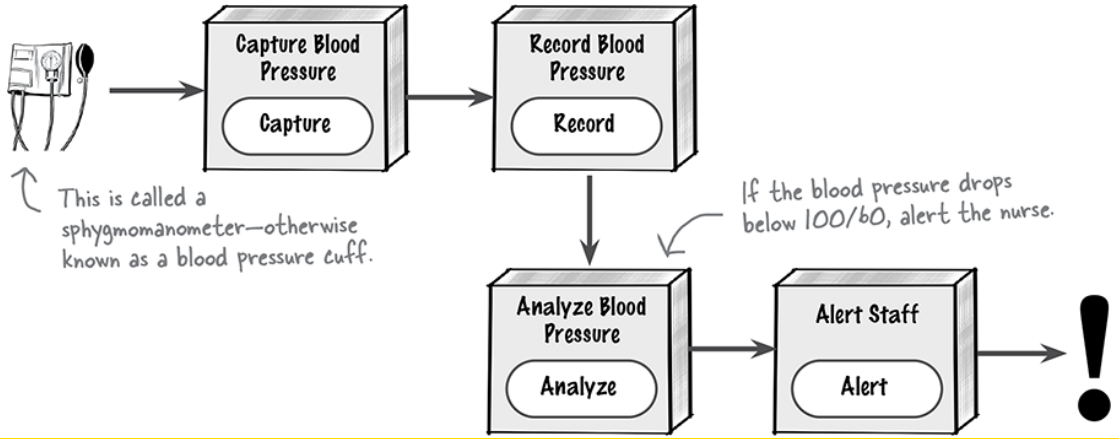
- Example:
  - Centralised order processing orchestrating payment, inventory, and shipment services.



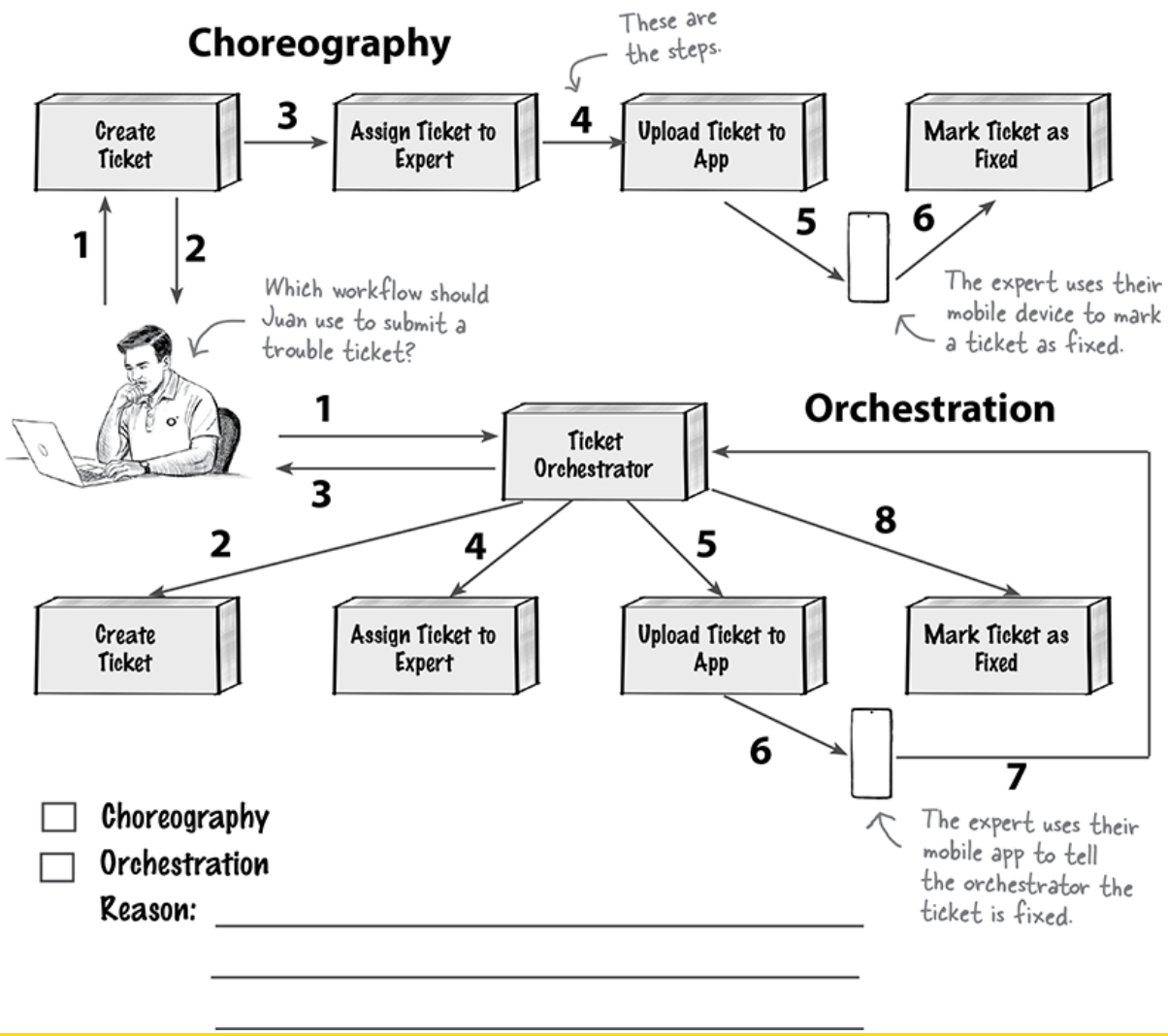


Workflow Management: Choreography

- Peer-to-peer service communication, like coordinated dance.
  - Pros: Scalable, loosely coupled, high responsiveness.
  - Cons: Complex error and state management.
- Example:
  - Event-driven updates between cart, inventory, and shipping services in an e-commerce site.



Exercise


Choreography
These are the steps.
Create Ticket
3
Assign Ticket to Expert
4
Upload Ticket to App
Mark Ticket as Fixed
1
2
5
6
Which workflow should Juan use to submit a trouble ticket?
The expert uses their mobile device to mark a ticket as fixed.
Orchestration
Ticket Orchestrator
1
3
2
4
5
8
Create Ticket
Assign Ticket to Expert
Upload Ticket to App
Mark Ticket as Fixed
6
7
The expert uses their mobile app to tell the orchestrator the ticket is fixed.
Choreography
Orchestration
Reason:

Advantages of Microservices

- Maintainability, Testability, Deployability, Evolvability.
- Exceptional scalability and fault tolerance.

- Examples:
  - Continuous deployment at Spotify
  - Scalable services at Netflix

Limitations of Microservices

- Complexity, especially in workflow management.
- Performance issues due to inter-service communications.

- Example:
  - Increased latency in highly interactive systems like gaming or real-time analytics platforms.

Balancing Microservices Architecture

- Decision criteria:
  - Business agility
  - Complexity handling
  - Team structure

- Optimal balance between granular control and practical maintainability.

- Example:
  - Amazon's product catalog services balancing granularity and maintainability.

Case Study - StayHealthy MonitorMe

- Successful real-world implementation of microservices.
- *Insights*: Balance granularity, effectively manage shared resources.
- Continuous focus on agility and operational stability.

- Example:
  - Reliable and scalable health monitoring system for critical patient data.

Microservices Star Ratings

Architectural Characteristic	Star Rating
Maintainability	★ ★ ★ ★ ★
Testability	★ ★ ★ ★ ★
Deployability	★ ★ ★ ★ ★
Simplicity	★
Evolvability	★ ★ ★ ★ ★
Performance	★ ★
Scalability	★ ★ ★ ★ ★
Elasticity	★ ★ ★ ★
Fault Tolerance	★ ★ ★ ★ ★
Overall Cost	$ $ $ $ $

Summary

- Microservices offer high flexibility but involve significant complexity.
- Requires crucial granularity and communication decisions.
- Evaluate and manage trade-offs carefully.

- Example:
  - Transitioning from monoliths to microservices at Uber.