ARCHITECTURE OF LARGE-SCALE SYSTEMS

By Dr. Taghinezhad

Mail:
a0taghinezhad@gmail.com

- Chapter 6:Organization: Scaling Your Organization for Modern Applications

Chapter 6:Organization: Scaling Your Organization for Modern Applications

Chapter Overview

•**Purpose**: To explore the concept of service ownership in a large-scale, service-based application and the necessity of structured ownership for effective system management.
•**Key Topics**: Single Team Owned Service Architecture (STOSA), service ownership principles, benefits of clear ownership.

Introduction to Service Ownership

Service Ownership: A Key to Team Accountability

- Defines team responsibility for a service lifecycle (design to maintenance).
- Enables division of complex applications across teams.
- Structured ownership reduces ambiguity and boosts accountability.

Single Team Owned Service Architecture (STOSA)

- **Definition**: STOSA is an organizational and architectural approach where a single, dedicated team manages each service within an application.
- **Objective**: To establish clear ownership and accountability, reduce dependencies, and promote efficient, independent development within a large organization.

Core Principles of STOSA

- **STOSA Compliance: Key Criteria**
  1. **Service-Based Architecture**
     1. Modular design with independent services.
  2. **Multiple Development Teams**
     1. 3–8 engineers per team for optimal management.
     2. Each service is assigned to a single team.
  3. **Unique Service Ownership**
     1. One team per service; no shared ownership.
     2. Ownership is clearly documented and accessible.
  4. **End-to-End Responsibility**
     1. Teams manage:
        1. Design & Architecture
        2. Development & Testing
        3. Deployment & Monitoring
        4. Incident Resolution

Core Principles of STOSA (Cont.)

- **STOSA Compliance: Additional Criteria**
  - 5. **Well-Defined Boundaries & APIs**
    - Services interact via documented APIs.
    - Minimizes cross-team dependencies, ensuring clear communication.
    - STOSA Systems:
      - **STOSA Application**: Uniform services meeting criteria.
      - **STOSA Organization**: Teams follow STOSA rules, enhancing accountability.
      - Example: 12 services (A-L) managed by 5 teams, each with clear ownership.

6. **Data Ownership**
   6. Services manage their own data via encapsulation.
   7. External data accessed only through APIs.

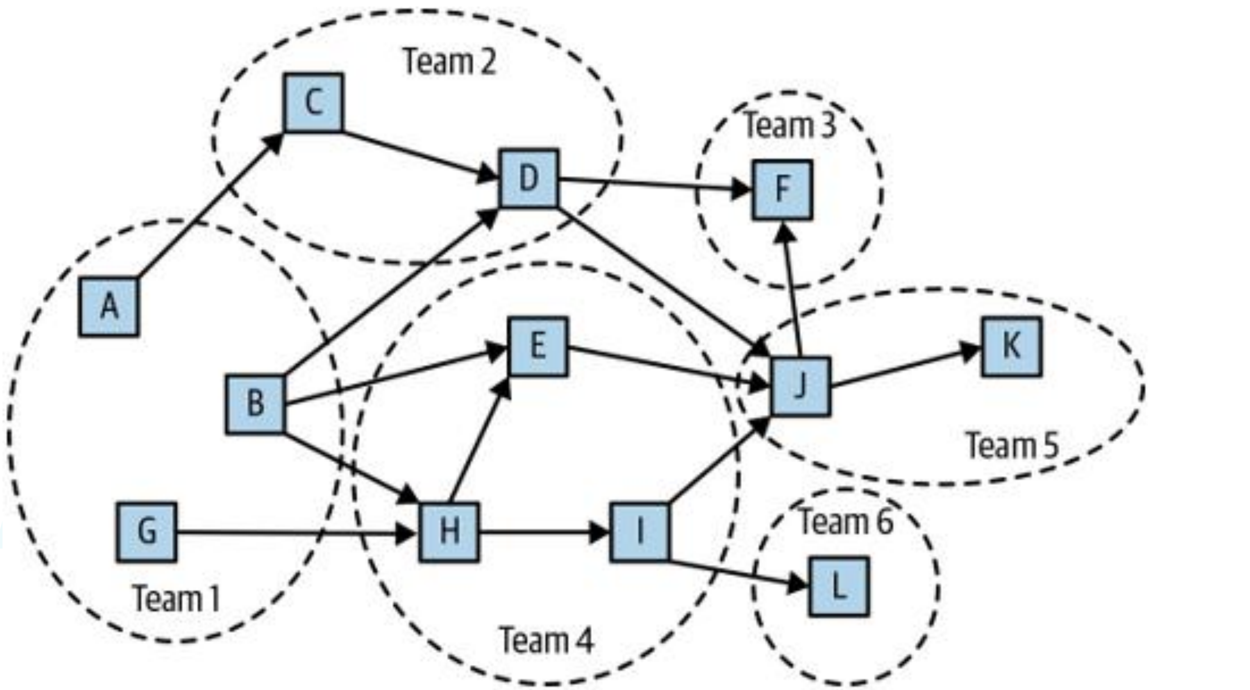
7. **Service-Level Agreements (SLAs)**
   6. Define service performance expectations.
   7. SLA violations monitored and addressed by responsible teams.

STOSA Application and Organization

- **STOSA-Based System Characteristics**
- **STOSA Application**:
  - All services meet STOSA criteria for uniformity and predictable interactions.
- **STOSA Organization**:
  - Teams adhere to STOSA rules, enabling streamlined management and accountability.

Example STOSA-Based vs. Non-STOSA Systems

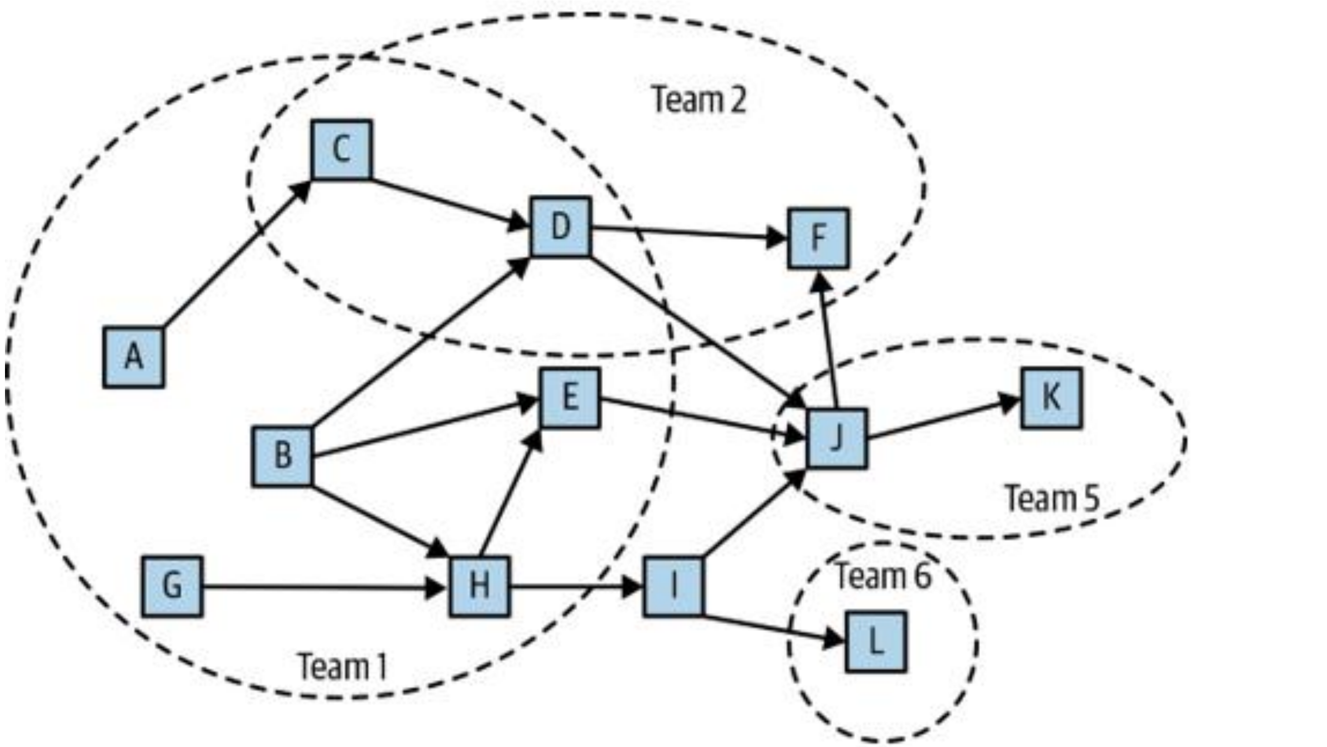
- **STOSA Example**:
  - An application with twelve services (A through L), managed by five teams.
  - **Every service** has **one owner**; no overlapping responsibilities.
  - **Clear** ownership allows **efficient management**, direct points of contact, and structured incident responses.



STOSA-based organization with a STOSA application

Example STOSA-Based vs. Non-STOSA Systems

- **Non-STOSA Example**:
  - Service I lacks ownership.
  - Services C and D are managed by multiple teams.
  - Result: Confusion, delays, and unstructured problem-solving.



Non-STOSA-based organization

Advantages of a STOSA-based Application and Organization

- **Scalability**: STOSA-based applications can grow in both size and complexity, managed effectively by larger development teams.
- **Complexity Management**: STOSA distributes the complexity of large applications across multiple teams, with each team clearly owning a subset of services.
- **Clear Ownership and Responsibility**: Defined ownership across teams ensures accountability, facilitating efficient troubleshooting and development processes.
- **Supportable Interfaces**: Documented and supportable interfaces promote interoperability and maintainability as the application scales.

Service Ownership in a STOSA Organization

- **Ownership Definition**: A service-owning team in a STOSA structure is entirely accountable for all aspects of the service, regardless of dependencies on other teams (e.g., for infrastructure support).
- **Responsibilities**:
    1. **API Design**: Complete management of all APIs, internal and external, including design, implementation, testing, and version control.
    2. **Service Development**: Ownership of business logic, implementation, and testing specific to the service.
    3. **Data Management**: Complete responsibility for the service's data, including schemas, storage, and access patterns.
    4. **Deployment Management**: Planning and execution of service updates, ensuring stable deployment with rollback procedures if necessary.
    5. **Deployment Windows**: Determining safe deployment times, adhering to company-wide blackout periods and specific service windows.

Service Ownership in a STOSA Organization

- **Responsibilities (Cont.):**

6. **Infrastructure Changes**: Adjusting production infrastructure as needed for optimal performance (e.g., load balancing).

7. **Environment Management**: Overseeing production and non-production environments for testing, staging, and deployment.

8. **Service SLAs**: Setting, monitoring, and ensuring compliance with SLAs, with proactive responses to violations.

9. **Monitoring**: Establishing consistent monitoring, especially around SLA metrics and regular review of service health.

10. **Incident Response**: Implementing on-call rotation, managing notifications, and ensuring timely incident handling.

11. **Reporting**: Providing internal reports on operational health and status updates to other teams and management.

Role of Supporting Core Teams in a STOSA Organization

- **Shared Responsibilities**: In many cases, infrastructure elements like servers, tooling, and databases are managed by central core teams.
  - **Servers/Hardware**: Infrastructure typically managed by operations or cloud providers.
- **Tooling**: Deployment, monitoring, and incident management tools are often centralized for consistency.
- **Databases**: While the core database infrastructure may be managed centrally, data responsibility remains with the owning team.

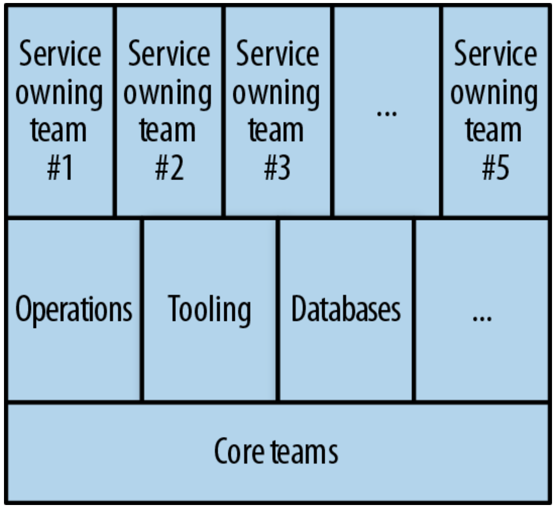
Organizational Structure in a STOSA-based System

- **Team Structure**
  - Service-owning teams are **peers** in the STOSA structure.
  - Supported by core teams (e.g., operations, databases, tooling) that provide:
    - Uniform infrastructure and tooling support.
    - No direct responsibility for service outcomes.
- **Culture of Accountability**
  - Service-owning teams retain full responsibility for their service, even if:
    - Failures result from dependencies (e.g., tools managed by another team).
  - Fosters ownership and proactive problem resolution.

Example:

•Deployment fails due to an external tool issue.

•Service-owning team remains accountable for restoring service health.



Decision-Making Autonomy for Service Teams

- **Flexibility & Advantages of Core Services**
  - **Flexibility in Core Services**
    - Teams can use alternative resources (e.g., non-standard databases/cloud providers) if they meet organizational standards.
    - Provides autonomy in service management.
  - **Advantages of Core Services**
    - Reduces operational burden for service teams.
    - Central teams focus on delivering high-quality, customer-centric tools to retain users.
  - **Encouraging Buy-In**
    - Perceived or actual choice in core services boosts engagement and satisfaction.
    - Standardized core services become essential in larger organizations but should remain team-focused.

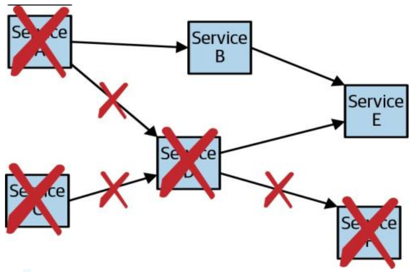
Chapter 7:Service Tiers

Overview

- In modern distributed systems with large, complex applications, maintaining availability is crucial.
  - A failure in a single service can trigger a **cascade failure**, leading to the failure of other dependent services.
    - especially problematic when the failure of a non-critical service results in the disruption of mission-critical services.
    - To manage this complexity and prioritize service availability, **service tiers** are introduced.

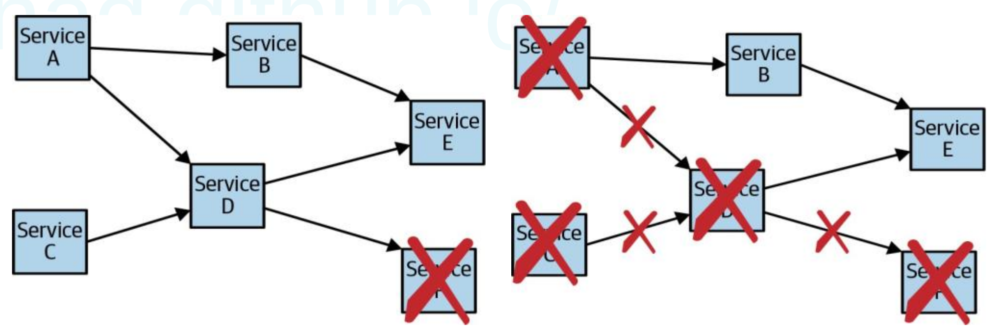
Application Complexity and Cascade Failures

- **Service Dependencies & Cascade Failures**
  - **Interconnected Services**
    - Large applications rely on multiple interdependent services.
    - A single service failure can cascade, affecting dependent systems.
    - **Example of Cascading Failure**
      - Failure of non-critical Service D disrupts mission-critical Service A, leading to widespread outages.
  - **Understanding & Mitigating Cascade Failures**
    - **Figures 7-1 & 7-2**: Illustrate minor failures causing large-scale outages.
    - Resiliency solutions:
      - Add safeguards between services.
      - Challenge: Increased complexity and cost.
    - Key Question:
      - How to distinguish critical service failures from non-critical ones?



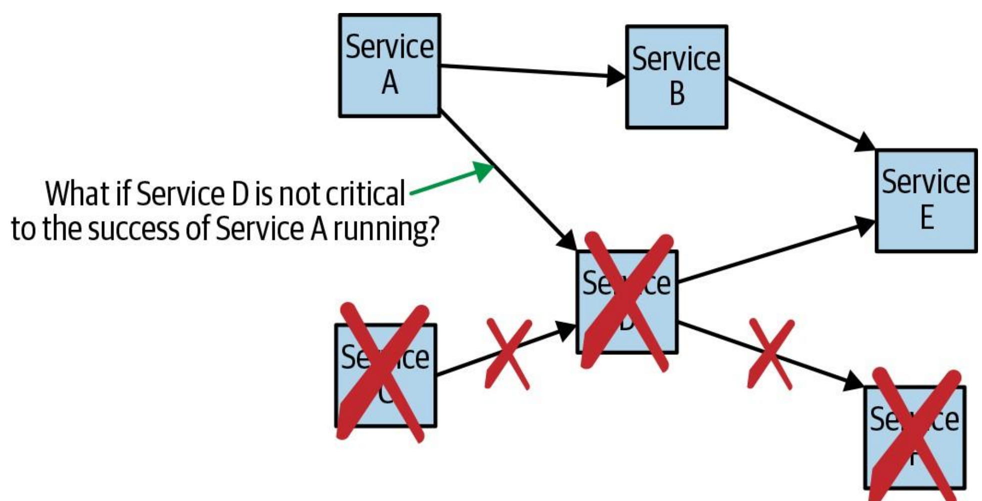
Application Complexity and Cascade Failures

- **Mitigating Cascade Failures**
  - **Safeguards Between Services**
  - To reduce the risk of cascading failures, safeguards can be implemented to isolate failures and prevent them from propagating.
- **Example Safeguard: Circuit Breaker Pattern**
  - **What it does:** The circuit breaker monitors a service's response and halts communication if the service shows signs of failure.
  - **How it works:**
    - If Service D fails or slows down, the circuit breaker opens, temporarily cutting off Service A’s reliance on it.
    - Service A can operate in a degraded mode, such as using cached or default data instead of waiting indefinitely for Service D.
    - The circuit breaker periodically tests Service D to see if it has recovered, then resumes normal operations.



What Are Service Tiers?

- **Service Tiers: Classifying Service Criticality**
  - **Definition**:
    - Labels used to categorize services based on their importance to business operations.
  - **Purpose**:
    - Distinguish **mission-critical** services from less essential ones.
    - Manage application complexity and maintain availability.
  - **Benefits**:
    - Clarifies the importance of individual services.
    - Identifies critical dependencies to prioritize resiliency efforts.



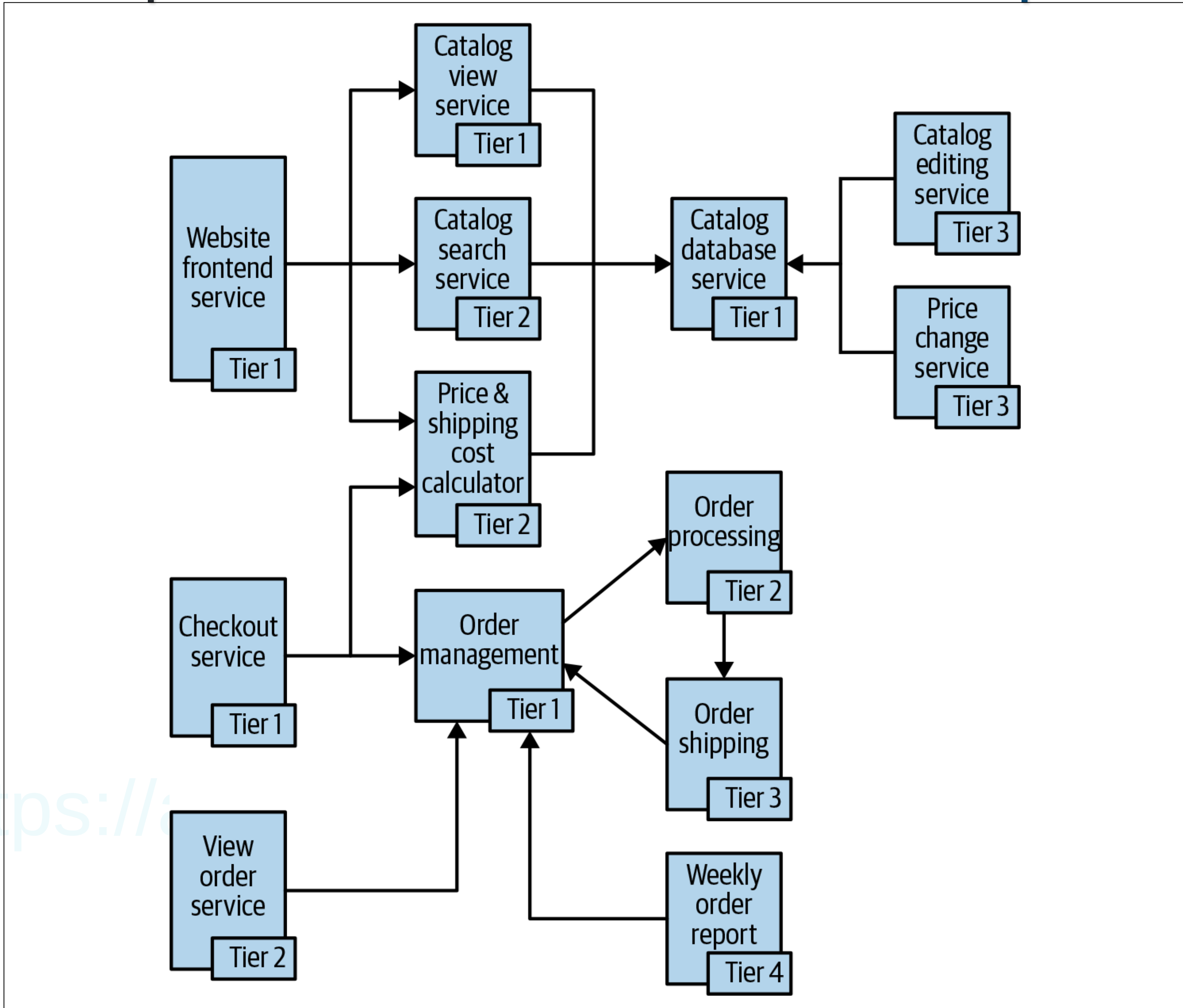
Assigning Service Tier Labels

- **Service Tier Classification: all services are assigned a tier.**
- **Tier 1: Mission-Critical Services**
  - **Definition**: Essential for application functionality; failure disrupts operations.
  - **Examples**:
    - Login Service
    - Credit Card Processor
    - Permission Service
    - Order Accepting Service
  - **Impact of Failure**: High; immediate resolution required.
- **Tier 2: Important Services**
  - **Definition**: Degrade user experience but do not halt system usage.
  - **Examples**:
    - Search Service
    - Order Fulfillment Service
  - **Impact of Failure**: Moderate; system remains functional but less effective.

Assigning Service Tier Labels

- **Service Tier Classification (Tier 3 & Tier 4)**
- **Tier 3: Minimal Impact Services**
  - **Definition**: Failures have minor or unnoticed effects on users and operations.
  - **Examples**:
    - Customer Icon Service
    - Recommendations Service
    - Message of the Day Service
  - **Impact of Failure**: Low; minor disruption without significant consequences.
- **Tier 4: Non-Essential Backend Services**
  - **Definition**: Failures have no noticeable impact on customers or immediate operations.
  - **Examples**:
    - Sales Report Generator Service
    - Marketing Email Sending Service
  - **Impact of Failure**: Minimal; disruptions are negligible for users and business.

Example: Online Store Service Tiers services categorized by their importance to the business and customer experience.



Key Service Tier Examples

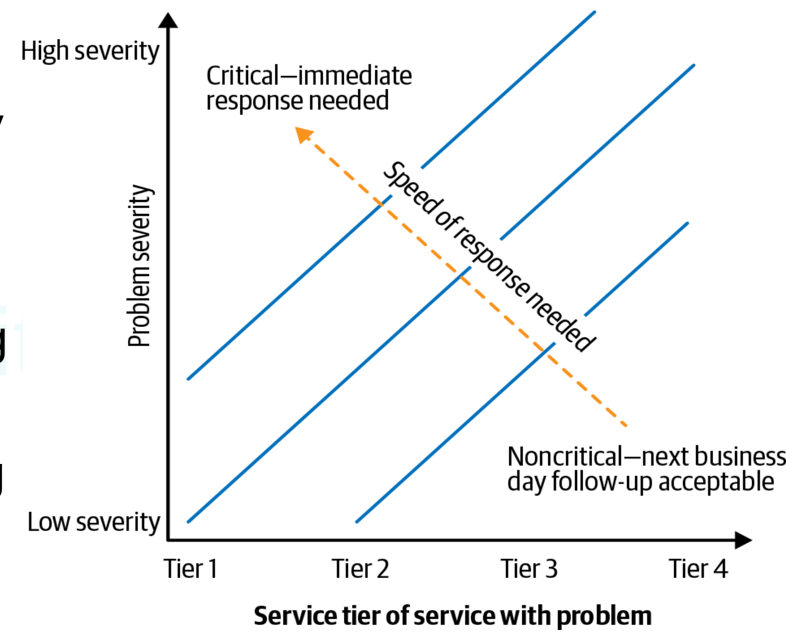
- **Tier 1: Mission-Critical Services**
  - **Website Frontend Service**: Displays the storefront; downtime makes the store inaccessible.
  - **Catalog View Service**: Supplies product details to the frontend; critical for usability.
  - **Catalog Database Service**: Stores product data; site unusable without it.
  - **Checkout Service**: Manages purchases; impacts revenue directly.
- **Tier 2: Important but Non-Critical**
  - **Catalog Search Service**: Supports product search; users can navigate manually if unavailable.
- **Tier 3: Minor Impact**
  - **Catalog Editing Service**: Allows staff to update entries; minor customer impact.
  - **Order Shipping Service**: Handles packaging; short outages have minimal effect.
- **Tier 4: Minimal Impact**
  - **Weekly Order Report**: Generates sales reports; delays have no customer impact.

Key Service Tier Examples

1. **Tier 1 Services** (Mission-Critical)
   1. **Website Frontend Service**:
      1. **Role**: Generates and displays the online storefront, handling the main interaction between the user and the site.
      2. **Reason for Tier 1**: If unavailable, the entire store is inaccessible to customers, significantly impacting their experience.
   2. **Catalog View Service**:
      1. **Role**: Reads from the catalog database to supply product details to the frontend service.
      2. **Reason for Tier 1**: Customers cannot view products without this service, heavily impacting usability.
   3. **Catalog Database Service**:
      1. **Role**: Stores all product information.
      2. **Reason for Tier 1**: Without access to the catalog data, no product can be displayed, making the site unusable.
   4. **Checkout Service**:
      1. **Role**: Manages the customer checkout process.
      2. **Reason for Tier 1**: Prevents customers from completing purchases, directly affecting revenue.

Using Service Tiers to Optimize Operations

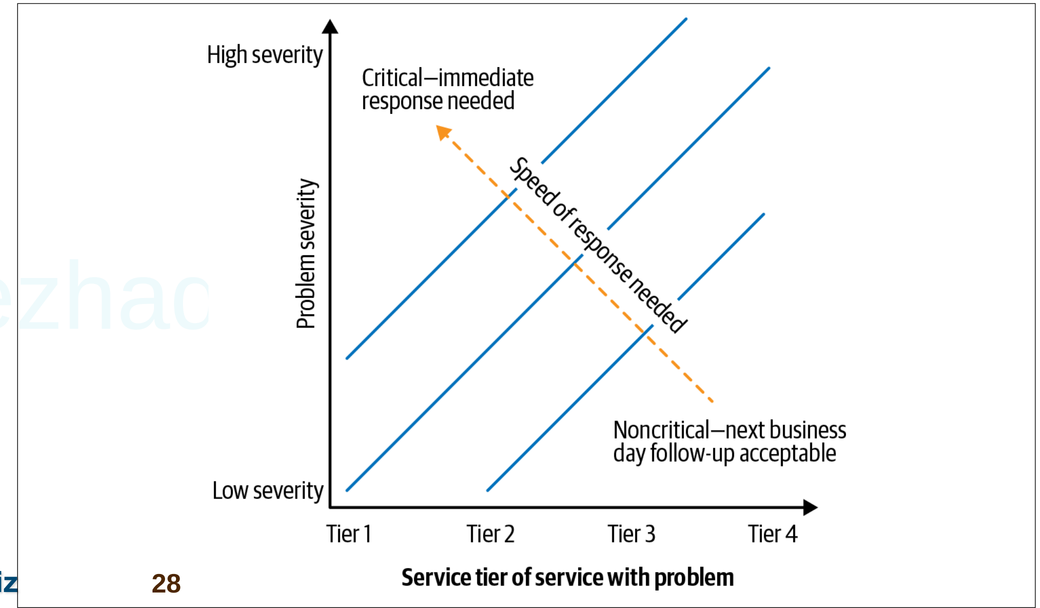
- **Benefits of Service Tiering**
- **Key Aspects**
  1. **Expectations**:
     1. Define SLAs by tier (e.g., highest uptime for Tier 1).
  2. **Responsiveness**:
     1. Align response based on severity and tier:
        1. Immediate action for Tier 1 high-severity issues.
        2. Tier 1 medium-severity takes precedence over Tier 3 high-severity (see Fig. 7-5).
  3. **Dependencies**:
     1. Evaluate tier levels to mitigate cascading risks.
     2. Avoid critical (Tier 1) services depending on non-critical (Tier 3) ones.



Practical Benefits of Service Tiering

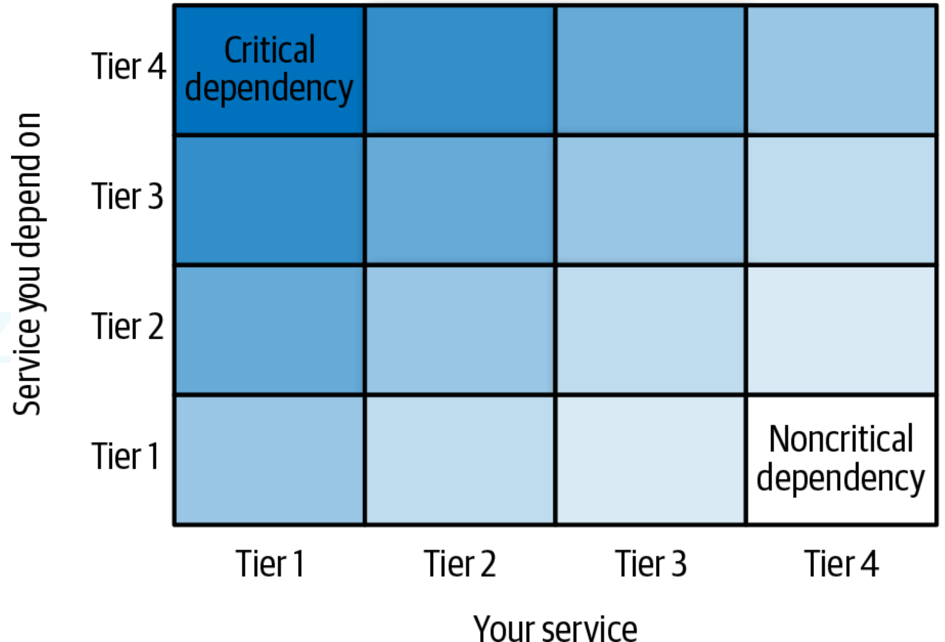
1. **Efficient Resource Allocation**: High-priority resources are focused on Tier 1 services, while less critical services receive proportionate attention.
2. **Improved Response Planning**: Tiers help prioritize alert notifications, set response schedules, and outline escalation paths.
3. **Informed SLA Development**: With tier-based SLAs, businesses can define clear expectations for availability and responsiveness.

- Service tiering ultimately supports system resilience, prioritizing critical operations while managing costs and complexity for less impactful services.



Managing Dependencies in Service Tiers

- **Dependency Criticality in Service Tiers**
  - **Understanding dependency criticality** is key when building a service. Figure 7-6 highlights the relationship between a service's tier level and that of its dependencies:
  - **Critical Dependency**: When your service tier (lower number) is more critical than the dependent service tier (higher number).
  - **Noncritical Dependency**: When your service tier is less critical (higher number) than the dependent service tier.

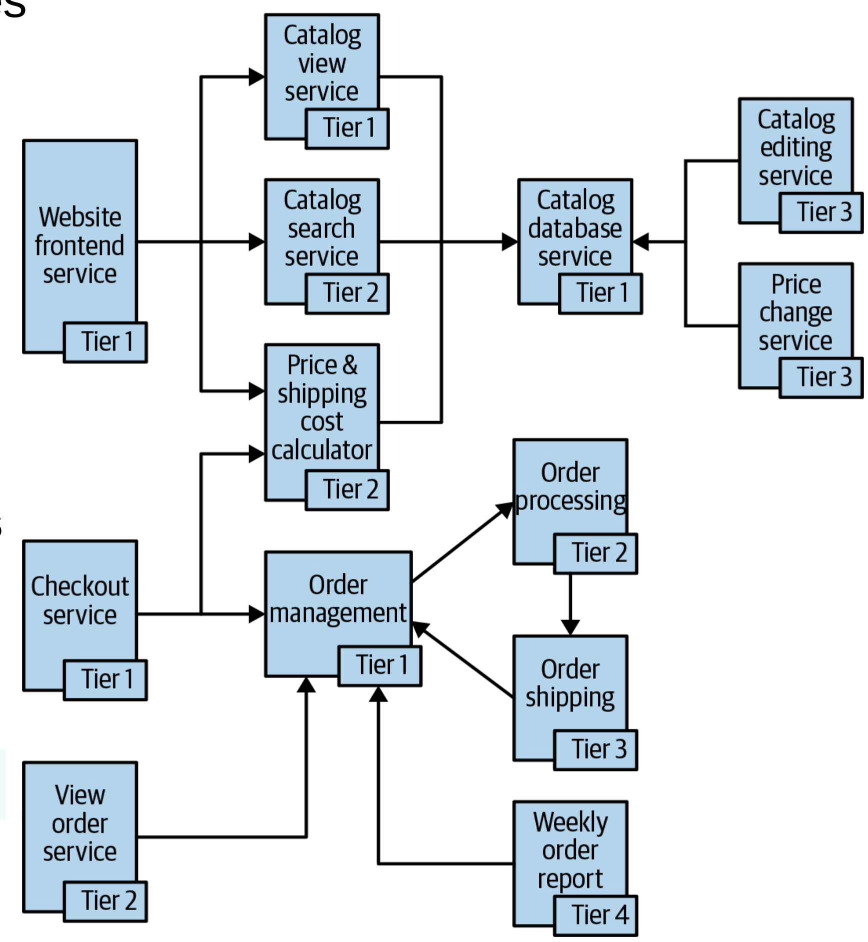


Types of Dependencies

- **Critical Dependency**
  - If a dependency is **critical**, the service must be designed to **handle dependency failures gracefully** to ensure minimal impact on users.
- **Example**:
  - Consider the **Website Frontend Service** (Tier 1) in an online store.
  - It depends on the **Price & Shipping Cost Calculator (PSCC)** service (Tier 2) to fetch current product prices.
  - **If the PriceShipingCostCalculator service is down**, the frontend service must still function and could use alternative strategies:
    - **Display a cached price** if available.
    - **Show the product page without a price**, with a message like “Price not currently available” or “Add to cart to see current price.”
  - This approach allows for **graceful degradation**—even if the experience is diminished, customers can still interact with the site.

Types of Dependencies (Cont.)

- **Noncritical Dependency**
  - If a dependency is **noncritical**, failures can be **tolerated with minimal handling**.
- **Example**:
  - The **Weekly Order Report Service** (Tier 4) depends on the **Order Management Service** (Tier 1) for data.
  - **If the Order Management Service is unavailable**, it is acceptable for the Weekly Order Report Service to **fail temporarily**.
  - Given that Order Management is a high-tier service, its issues will be resolved quickly, and the reporting service can resume operation without specific handling.



Benefits of Service Tiers for Dependency Management

- Service tiers provide a clear way to establish expectations for **availability, responsiveness, and reliability** across dependencies:
  - **Enhanced Clarity**: Service owners and developers understand criticality expectations and can manage dependencies appropriately.
  - **Simplified Communication**: Service tiers enable straightforward communication, reducing the risk of misunderstandings and service misconfigurations.
  - **Resilience Planning**: By knowing dependency tiers, developers can design appropriate fallback mechanisms or allow graceful degradation only where necessary.

THE END

https://ataghinezhad.github.io/