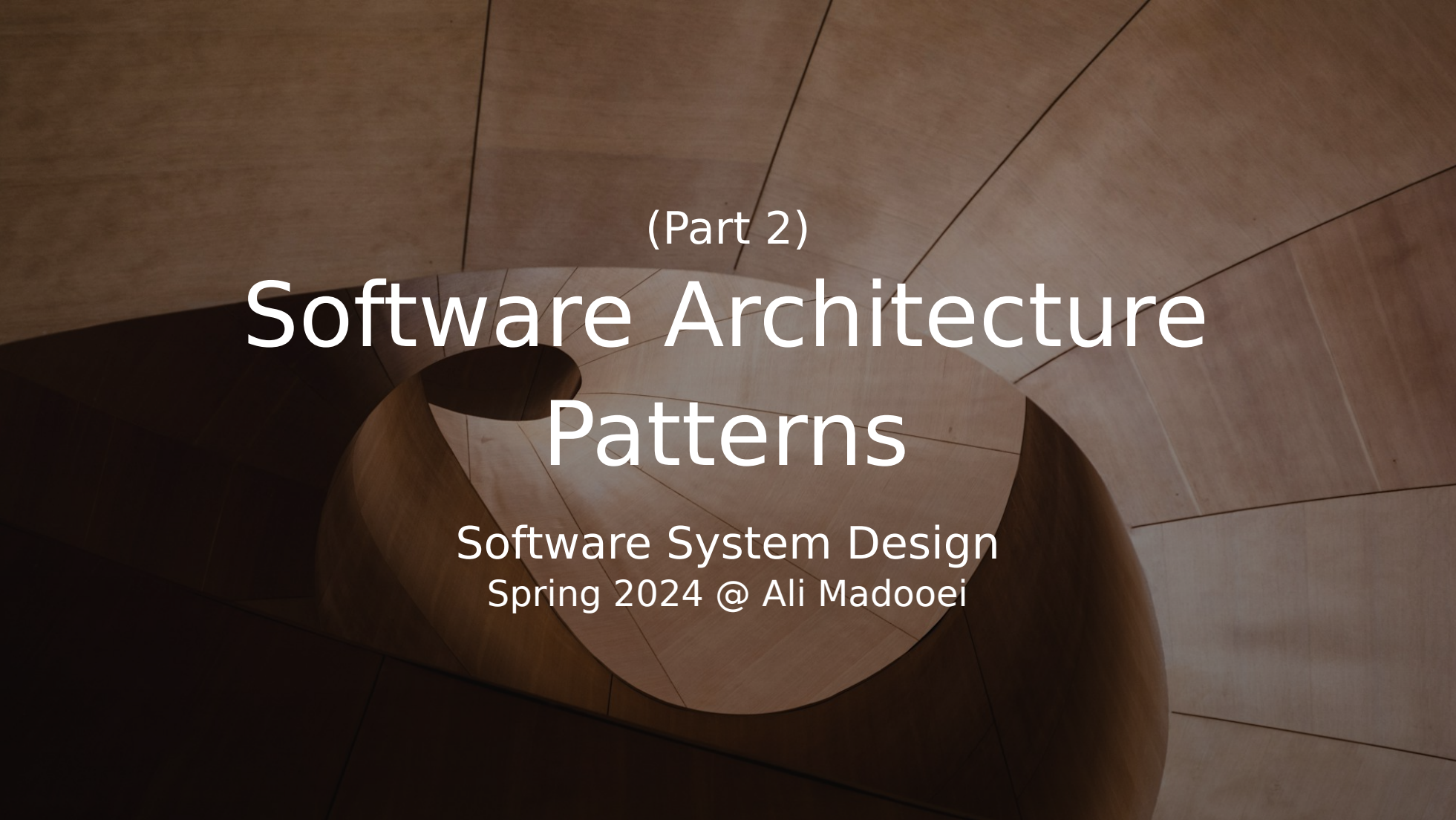

(Part 2)

Software Architecture Patterns

Software System Design

Spring 2024 @ Ali Madooei

Service-Oriented Architecture (SOA)

While client-server architecture focuses on the separation of the client and server components, Service-Oriented Architecture (SOA) takes this concept further by breaking down the server-side components into smaller, more modular services.

- In MERN stack, it is common to have separate services for database management and API routing.
- We can scale the database service independently of the API service.

Activity: SOA vs. Layered

- How does Service-Oriented Architecture (SOA) differ from or relate to Layered Architecture?

Solution: SOA vs. Layered

- **Layered Architecture**: Separate functionalities within a single application.
  - Presentation, Business Logic, Data Access.
- **Service-Oriented Architecture (SOA)**: Externalize these functionalities as distinct services.
  - Each service is reusable across different applications.
  - Communicate over network using APIs.

Database-as-a-Service (DBaaS)

DBaaS is a cloud computing service model that provides database services for web and mobile applications.

- In MERN stack, **MongoDB Atlas** is an example of a DBaaS platform that provides managed MongoDB databases.
- DBaaS platforms handle tasks such as database provisioning, scaling, backup, and security.

Backend-as-a-Service (BaaS)

BaaS is a cloud computing service model that provides backend services for web and mobile applications.

- It allows developers to focus on building the frontend (client) of the application while the backend services are managed by a third-party provider.
- Common services provided by BaaS platforms include user authentication, data storage, and push notifications.
- Google Firebase and AWS Amplify are examples of BaaS platforms.

Functions-as-a-Service (FaaS)

FaaS is a cloud computing service model that allows developers to run code in response to events without managing servers.

- BaaS models offerns common services and basic CRUD operations.
- However, if we need to run custom code in response to specific events, we can use FaaS platforms like AWS Lambda or Google Cloud Functions.
- This results in the **serverless architecture** where developers focus on writing functions that respond to events rather than managing servers.

Detour: Cloud Computing

Cloud computing refers to the delivery of computing resources (like servers, storage, databases, networking, software, analytics, intelligence, and more) over the internet, which is often referred to as "the cloud.”

- **Examples**: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP).

Cloud Computing Services

In addition to DBaaS, BaaS, and FaaS, cloud computing platforms offer a wide range of services to help developers build, deploy, and scale applications:

- **Infrastructure as a Service (IaaS)**: Provides virtual servers, storage, and networking resources.
- **Platform as a Service (PaaS)**: Provides a platform for developing, testing, and deploying applications.
- **Container as a Service (CaaS)**: Provides container orchestration and management.
- **Software as a Service (SaaS)**: Provides software applications over the internet.
- **Machine Learning as a Service (MLaaS)**: Provides ML models and tools.

Microservices Architecture

Microservices architecture is an architectural style that structures an application as a collection of small, autonomous services.

- Each service is self-contained and can be developed, deployed, and scaled independently.
- Services communicate with each other over a network using lightweight protocols like HTTP or messaging queues.
- Microservices architecture is a popular choice for building large, complex applications that require scalability, flexibility, and resilience.

"Posts" App in Microservices Architecture

- **Auth Service**: Handles user authentication and authorization.
- **User Service**: Manages user profiles and settings.
- **Posts Service**: Handles the creation, retrieval, and deletion of posts.
- **Comments Service**: Manages comments on posts, including creating, retrieving, and deleting them.
- **Notifications Service**: Sends notifications to users based on activity.

Each service is a separate entity with its own codebase, database, and deployment process. They communicate with each other over the network using APIs or messaging queues.

Microservices: Characteristics

- **Focused Services**: Each microservice handles a specific business capability.
- **Scalability**: Independently scale each microservice based on its demand.
- **Flexibility**: Use the best tech stack for each microservice’s needs.
- **Resilience**: If one microservice fails, it doesn’t bring down the entire system.

Activity: Microservices vs. SOA

- How does the microservices architecture differ from or relate to the Service-Oriented Architecture (SOA)?

Solution: Microservices vs. SOA

- **Granularity**:
  - **Microservices**: Typically smaller, more fine-grained services focused on a single responsibility.
  - **SOA**: Generally consists of larger, more coarse-grained services that may encompass multiple business functions.
- **Communication**:
  - **Microservices**: Often communicate via lightweight protocols such as HTTP/REST or messaging queues.
  - **SOA**: Typically uses more complex communication protocols like SOAP (Simple Object Access Protocol) and enterprise service buses (ESBs) for integration.

Solution: Microservices vs. SOA (Contd.)

- **Deployment**:
  - **Microservices**: Designed for independent deployment, often in containers, allowing for individual scaling and updates.
  - **SOA**: Services are usually deployed as part of larger monolithic applications or as separate but interconnected components.
- **Data Management**:
  - **Microservices**: Each service manages its own data, promoting data encapsulation and autonomy.
  - **SOA**: Data is often shared and managed centrally, leading to tighter coupling between services.

Solution: Microservices vs. SOA (Contd.)

- **Evolution**:
  - **Microservices**: Evolved as a response to the limitations of SOA, focusing on smaller services and more decentralized approaches.
  - **SOA**: Predates microservices and laid the groundwork for service-based architectures, but has been criticized for its complexity and heavy reliance on centralized governance.

In summary, while both architectures are service-oriented, microservices are a more modern, focused, and decentralized approach compared to the traditionally heavier and more centralized SOA.

Deploying Microservices

- **Containerization**: Each service is containerized using Docker for environment consistency.
- **Communication**: Services communicate over well-defined APIs (e.g., via HTTP/REST) or asynchronous messaging for events.
- **Database**: Each service has its own database for true data independence.
- **Orchestration**: Use a software solution like Kubernetes to handle the deployment, scaling, and management of service containers.

What is a Container?

Containers are lightweight, stand-alone, and executable software packages that encapsulate a piece of software in a complete filesystem.

- Contain everything needed to run: code, runtime, system tools, libraries, settings.
- Consistent and reproducible environment across different stages.
- Eliminate the "it works on my machine" problem.

Why Did We Need Containers?

Historical Context: Software deployment faced challenges with consistency, scalability, and isolation.

- Differing environments led to deployment issues.
- Scaling applications required duplicating the entire VM or server.
- Isolating applications meant running multiple VMs, consuming more resources.

How Are Containers Different from VMs and Bare Metal?

- **Bare Metal:** Physical servers; direct access to hardware.
- **Virtual Machines (VMs):** Emulate physical hardware, run multiple OS instances.
- **Containers:** Share OS kernel, isolated processes; lightweight compared to VMs.

What is Docker?

Docker is a platform that automates container creation, deployment, and management.

- Provides a consistent environment for applications.
- Uses Dockerfile to define container specifications.
- Docker Hub: Repository for sharing container images.

Are Containers and Docker the Same?

No!

- **Containers:** A technology that encapsulates software in a consistent environment.
- **Docker:** A tool/platform to create, deploy, and manage containers.

Docker is a specific tool that popularized and streamlined the use of containers. It’s essential to distinguish between the methodology (containers) and the tool (Docker).

What are the Alternatives to Docker?

While Docker is popular, there are other containerization technologies:

- **Podman:** Similar to Docker but daemonless and rootless.
- **rkt (Rocket):** Emphasizes security and simplicity.

What is Kubernetes (K8s)?

Kubernetes is an open-source container orchestration platform.

- Manages the deployment, scaling, and operations of containerized applications.
- Ensures high availability, load balancing, and auto-scaling.
- Works with various container tools, not just Docker.

Why Do We Need Kubernetes?

Kubernetes is like an operating system for your cluster. It provides you with a platform to deploy, scale, and manage containerized applications across a cluster of machines.

- **Complexity:** Managing multiple containers manually is challenging.
- **Scalability:** Automatic scaling based on demand.
- **Resilience:** Ensures high availability and fault tolerance.

What are the Alternatives to Kubernetes?

Kubernetes dominates, but there are other orchestration solutions:

- **Docker Swarm:** Native clustering for Docker.
- **Apache Mesos:** A scalable cluster manager.

Who are the Common Providers for Docker and Kubernetes?

- **Docker Providers:** Docker Inc.
- **Kubernetes Providers:** Google (GKE), Amazon (EKS), Microsoft (AKS), and more.

Can We Use Containers Beyond Microservices?

Absolutely!

- Containers are versatile; not limited to microservices.
- Suitable for monoliths, traditional apps, batch jobs, and more.

Is it Possible to Implement Microservices Without Containers?

Yes!

- Containers simplify microservices but aren’t mandatory.
- Some organizations use VMs, serverless functions, or traditional servers.

Challenges in Microservices

A common challenge in microservices is managing the communication between services when we have many small, distributed services.

- Synchronous communication (like RESTful APIs) is when one service would make a direct call to another and wait for a response.
- While this worked, it had downsides:

1. **Tight Coupling**: Services needed to be aware of each other.
2. **Latency**: Waiting for responses increased.
3. **Reliability**: If one service failed, it could cascade and affect others.

Asynchronous Communication

To mitigate the challenges of synchronous direct communication, the industry started moving towards asynchronous communication.

- Instead of directly communicating, services would send **messages** (or emit *events*), often without expecting an immediate response.
- This reduced direct dependencies between services. A service emits an event when something notable happens, and any other service interested in that event reacts to it.
- **Message queues** and **event streams** became the backbone of this approach, leading to the rise of **event-driven architecture**.

Event-Driven Architecture (EDA)

Event-driven architecture is a software architecture pattern that promotes the production, detection, consumption, and reaction to events.

- Services communicate by emitting and reacting to events.
- Events are messages that represent a notable occurrence in the system.
- Event-driven systems are loosely coupled and highly scalable.

"Posts" App in Event-Driven Architecture

- **Event**: User creates a new post.
- **Message Queue**: Holds events like "Notify Followers", "Update Feed", "Check for Spam".
- **Services**:
  1. **Notification Service**: Retrieves the new post event, notifies the user’s followers.
  2. **Feed Service**: Updates the feeds of followers to include the new post.
  3. **Spam Detection Service**: Checks the post for potential spam or inappropriate content.
- **Outcome**: The new post is displayed in the feeds of followers, and the system ensures that the content is appropriate and relevant.