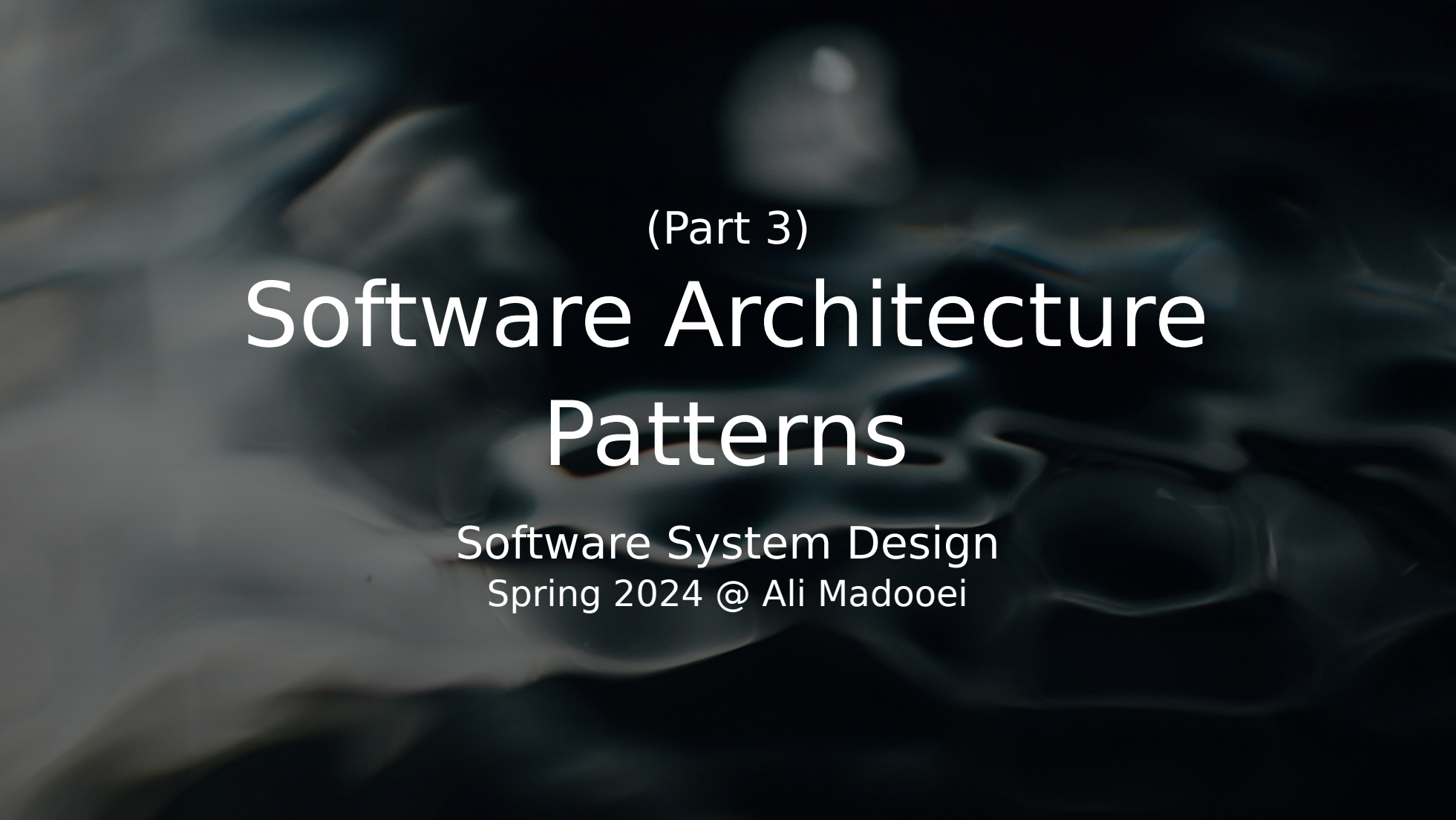

(Part 3)

Software Architecture Patterns

Software System Design

Spring 2024 @ Ali Madooei

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.

What is an Event?

- An event represents a change in state or an occurrence within a system.
- In programming, it’s often used to notify interested parts of the system that something has happened, allowing them to react accordingly.

In JavaScript, events are prevalent, especially in frontend frameworks and libraries where user interactions like clicks or form submissions trigger events.

```
document.querySelector('button').addEventListener('click', function(event) {
  console.log('Button was clicked!', event);
});
```

What is a Message?

A message is a discrete unit of communication intended for transmission between services or components. It often contains information about a specific action to be taken or data to be processed.

A message can be structured as an object with properties detailing its intent and content.

```
const message = {
  type: 'ORDER_PLACED',
  data: {
    orderId: '12345',
    userId: '67890',
    item: 'Laptop',
    quantity: 1
  }
};
```

Message vs. Event

- **Message**: A generic term for a discrete unit of data that’s sent from one place to another. Messages can represent commands, queries, replies, or even events. A message is more about the act of sending and receiving data.

- **Event**: A special kind of message that signifies a change in state. An event typically follows the "publish-subscribe" model, where the sender (or publisher) doesn’t direct the event to a specific recipient. Instead, any number of recipients (subscribers) can react to the event. An event is more about signaling that something has happened.

Message Queues

Message queues are fundamental tools in asynchronous systems, serving as intermediaries to manage communication between services.

- **Role**: Temporarily store messages until they are retrieved and processed by the receiving service.
- **Decoupling**: Sender and receiver services operate independently, unaware of each other's state.
- **Benefits**:
  - **Load leveling**: Absorb variations in the rate of message arrivals.
  - **Load balancing**: Distribute processing work across multiple resources or services.

Event Streams

Event streams are a sequence of events that represent changes in a system over time.

- **Role**: Capture and store events for later processing or analysis.
- **Benefits**:
  - **Replayability**: Events can be replayed to rebuild system state.
  - **Real-time Processing**: Enables real-time analytics, monitoring, and decision-making.
  - **Scalability**: Supports high-throughput event processing.

Event Streams vs. Message Queues

- **Message Queues**:
  - Store and forward messages between services.
  - Messages are consumed once and then removed from the queue.
- **Event Streams**:
  - Store and persist events over time. (Events can be replayed, analyzed, and processed multiple times.)
  - Supports real-time processing and analytics.

Use message queues for immediate communication between services, while event streams are better suited for capturing and analyzing events over time or for real-time processing.

Activity: Designing with Message Queues

Imagine an e-commerce platform. When a user places an order, several processes need to be initiated:

1. Inventory check and update.
2. Payment processing.
3. Shipping details and tracking.
4. Notification to the user about order status.

Design a communication flow using message queues to handle these processes efficiently.

Solution: Designing with Message Queues

1. **Order Service** emits an "Order Placed" event. The event is sent to a message queue.
2. **Inventory Service** listens to this event, checks inventory, and emits "Inventory Updated" or "Inventory Out of Stock".
3. **Payment Service** starts processing after inventory check and emits "Payment Processed" or "Payment Failed".
4. **Shipping Service** awaits payment confirmation, then processes shipping and emits "Shipped with Tracking".
5. **Notification Service** listens to all events to update the user on each step of the process.

Activity: Designing with Event Streams

Consider a sports analysis platform. The platform captures data from various sensors placed on athletes and in the field to provide real-time statistics, player health metrics, and game insights to coaches and analysts.

- Describe how an event stream can be implemented to process this data in real-time. Consider factors like data volume, speed, and the need for real-time analytics.

Solution: Designing with Event Streams

1. Sensors continuously emit data as events into the event stream.
2. *Player Metrics Service* processes data specific to player health and performance in real-time.
3. *Game Insight Service* analyzes game patterns, player positions, and ball movement for strategic insights.
4. Analysts and coaches can tap into processed data streams for immediate insights during the game.
5. The system can also store historical data, allowing for post-game analysis and trend identification.

Popular Message Queue Providers

- **RabbitMQ**: Open-source message broker software that offers reliable, flexible queuing with a variety of plugins.
- **Apache Kafka**: Distributed streaming platform often used for real-time analytics and monitoring, can handle high volumes of data.
- **Amazon SQS (Simple Queue Service)**: Managed message queuing service by AWS, ensuring secure and scalable message communication.
- **Azure Service Bus**: Microsoft’s cloud-based message system for connecting apps across on-premises and cloud environments.
- **Google Cloud Pub/Sub**: Real-time messaging service from Google Cloud, designed for highly-dynamic, globally-distributed apps.

Popular Event Streaming Providers

- **Apache Kafka**: Originally developed at LinkedIn, Kafka is a distributed streaming platform known for high-throughput, built-in partitioning, and replication.
- **Amazon Kinesis**: AWS's managed service for processing real-time data streams. It offers capabilities like stream analytics, data lakes, and log and event data collection.
- **Azure Stream Analytics**: Microsoft's event streaming service, integrated with other Azure services. It provides real-time analytics and complex event processing.
- **Google Cloud Dataflow**: A fully managed stream and batch data processing service from Google Cloud, designed for fast, reliable, and scalable data analytics.
- **Apache Flink**: An open-source stream processing framework that offers high-throughput, low-latency, and exactly-once processing semantics.

Activity: Event-Driven vs. Microservices

- How does the event-driven architecture differ from or relate to the microservices architecture?

Solution: Event-Driven vs. Microservices

Microservices:

- Focus on breaking down applications into small, autonomous services.
- Each service handles a specific business capability.
- Services communicate via APIs or messaging queues.

Event-Driven Architecture:

- Focus on the production, detection, consumption, and reaction to events.
- Services communicate by emitting and reacting to events.
- Loosely coupled and highly scalable systems.

A microservices architecture can be event-driven, meaning services communicate by emitting and reacting to events. However, an EDA is not inherently microservices; it can involve services running as serverless functions or in a monolithic application.

Serverless Architecture

Serverless doesn’t mean there’s no server. Instead, it’s about abstracting away the server management and infrastructure aspects, allowing developers to focus solely on the code.

- Serverless is a cloud-native development model that allows developers to build and run applications without thinking about the servers.
- It automatically manages the infrastructure, scales up or down with the number of requests, and you only pay for what you use.
- It’s highly event-driven, meaning your code runs in response to events like HTTP requests, database modifications, etc.

Serverless Characteristics

- **Automatic Scaling**: Your functions scale automatically based on the number of incoming requests.
- **Event-Driven**: Functions are executed in response to specific events or triggers.
- **No Infrastructure Management**: No need to provision, maintain, or administer servers.
- **Pay-as-You-Go Pricing**: You only pay for the actual compute time your functions consume.

"Posts" App in Serverless Architecture

- **Post Creation Function**: A function that creates a new post in the database.
- **Like Function**: A function that increments the like count for a post.
- **Comment Function**: A function that adds a comment to a post.
- **Notification Function**: A function that sends notifications to users.
- **Image Processing**: Integrate a function that springs into action when a user uploads an image, optimizing the image for storage and ensuring quick load times on the frontend.

Activity: Microservices vs. Serverless

- How does the serverless architecture differ from or relate to the microservices architecture?

Solution: Microservices vs. Serverless

Microservices:

- Consist of small, independent services.
- Managed by developers or DevOps teams.
- Typically run on containers or VMs.

Serverless:

- Consist of functions that run in response to events.
- Managed by cloud providers.
- Automatically scale based on demand.
- Pay-as-you-go pricing model.

A microservices architecture can be implemented using serverless functions. Serverless functions can be used to handle specific functionalities within a microservices architecture. However, serverless is not limited to microservices and can be used independently for specific use cases.

Activity: Serverless vs. Event-Driven

- How does the serverless architecture differ from or relate to the event-driven architecture?

Solution: Serverless vs. Event-Driven

Serverless:

- Focuses on abstracting away server management.
- Functions run in response to events.
- Automatically scales based on demand and pay-as-you-go pricing model.

Event-Driven:

- Focuses on the production, detection, and consumption of events.
- Services communicate by emitting and reacting to events.
- Loosely coupled and highly scalable.

A serverless architecture can be event-driven, meaning functions are triggered by events. However, an event-driven architecture is not inherently serverless, as it can involve services running on servers.

Choosing the Right Architecture

Traditional Architectures

Monolithic, Layered, Client-Server, etc.

- **Simple/Small Scale Application**: Suitable for applications with a limited scope and functionality.
- **Vertical Scaling**: Easier to scale by adding more resources to a single server.
- **Prototyping or Rapid Development**: Faster initial development due to a single codebase and fewer moving parts.
- **Small/Inexperienced Development Team**: Easier to manage and understand for smaller or less experienced teams.
- **Limited Time/Budget**: Lower initial costs and quicker time to market.

Microservices Architecture

- **Complex/Large Scale Application**: Suitable for applications that need to be broken down into smaller, independent services.
- **Horizontal Scaling**: Each service can be scaled independently based on its specific load.
- **Independent Deployment**: Services can be deployed and updated independently, reducing the risk of system-wide failures.
- **Large/Experienced Development Team**: Can be managed more effectively by teams with specific domain knowledge.
- **Flexibility in Technology Stack**: Different services can use different technologies based on their requirements.

Event-Driven Architecture

- **Asynchronous Communication**: Suitable for systems where components need to communicate without waiting for a response.
- **Highly Scalable**: Can handle a large number of events and scale dynamically as the load changes.
- **Loosely Coupled Components**: Services are more independent, reducing the impact of changes in one service on others.
- **Real-Time Data Processing**: Ideal for applications that require immediate processing of data, such as analytics or monitoring systems.
- **Resilience**: Failure in one component doesn’t directly impact others, improving system reliability.

Serverless Architecture

- **Variable Workload**: Cost-effective for applications with fluctuating demand, as you only pay for what you use.
- **Minimal Infrastructure Management**: The cloud provider manages the infrastructure, reducing the operational overhead.
- **Fast Deployment**: Quick deployment and updates are possible, making it suitable for agile development.
- **Event-Driven Nature**: Functions are triggered by specific events, making it a good fit for event-driven architectures.
- **Microservices Compatibility**: Can be used in conjunction with microservices to handle specific functionalities.

Case Studies

For each of the following scenarios, discuss the most suitable architecture (Microservices, Event-Driven, Serverless, etc.) and justify your choice.

University Teaching Assistant Management System

A university department seeks to develop a software solution to streamline the application and hiring process for teaching assistants. The system should provide a platform for students to submit their applications, faculty to review and select candidates, and administrators to manage the overall process. Key features include a user-friendly interface for submitting and reviewing applications, a database to store applicant information and status, and notification capabilities to inform applicants of their application status. The system aims to simplify and expedite the hiring process, ensuring that teaching assistant positions are filled efficiently and effectively. The department would like to hire students to develop this app as a way of providing real-world experience and supporting student employment opportunities.

Solution: University Teaching Assistant Management System

- **Architecture Pattern**: Monolithic or Layered
- **Justification**:
  - The application is relatively simple and has a well-defined scope.
  - A monolithic architecture is easier to develop and manage, especially with a student development team.
  - Layered architecture can provide a clear separation of concerns, making it easier for students to collaborate on different parts of the system.

Online Course Affiliate Tracking Software

A group of experienced software developers has created a series of online courses on software development and published them on platforms like Udemy and Pluralsight. To boost sales, they want to implement an affiliate tracking system that rewards affiliates with a 50% revenue share and provides a 15% discount code for their audience. The system needs to track affiliate referrals, calculate commissions, and provide reporting tools for both the course creators and the affiliates. The solution should be scalable and cost-effective to accommodate fluctuations in affiliate activity and sales volume. The team is very cost-conscious as they are not confident that they can generate enough affiliate sales to justify a large investment in the tracking system.

Solution: Online Course Affiliate Tracking Software

- **Architecture Pattern**: Serverless
- **Justification**:
  - The pay-as-you-go model is cost-effective for a system with fluctuating usage.
  - Serverless architecture allows for easy scalability as affiliate activity varies.
  - Minimal infrastructure management is required, reducing operational overhead for the small team.

Canadian Cloud Storage Service

In response to regulations requiring data to be stored within Canada, a startup aims to create a cloud storage service similar to Dropbox, but with data centers located exclusively in Canada. The service should offer file storage, sharing, and synchronization across devices, with a focus on security and compliance with Canadian data protection laws. The architecture should be capable of handling high volumes of data and traffic, ensure rapid and reliable access to files, and allow for easy expansion and maintenance as the service grows. The startup has hired several experienced developers and cloud architects to design and implement the system, with a focus on scalability, security, and performance.

Solution: Canadian Cloud Storage Service

- **Architecture Pattern**: Microservices and Event-Driven
- **Justification**:
  - Microservices allow for independent scaling and updating of different components, essential for a growing service.
  - Event-driven architecture supports asynchronous processing, enhancing the system’s ability to handle high volumes of data and traffic.
  - This combination ensures rapid and reliable access to files while maintaining scalability and performance.

Conclusion

- **Evolution of Architectures**: Understanding the progression from monolithic to microservices and serverless architectures is crucial in modern software development.
- **Right Tool for the Right Job**: No one architecture fits all scenarios. The choice depends on the specific requirements and constraints of the project.
- **Microservices and Event-Driven**: These architectures provide scalability and flexibility, but come with their own set of challenges and complexities.
- **Serverless Advantages**: Serverless architecture offers a cost-effective solution with minimal infrastructure management, but may not suit every use case.