Principles of Architecture

Event-Driven Architecture (EDA)

Definition/Description:
Event-Driven Architecture promotes asynchronous communication where components produce, detect, consume, and react to events. This architecture is commonly used in systems requiring real-time processing and scalability.

Diagram:

```
+------------------+
| Event Producer   |
+--------+---------+
         |
         v
+--------+---------+
| Event Broker     |
+--------+---------+
         |
         v
+--------+---------+
| Event Consumer   |
+------------------+
```

Domain-Driven Design (DDD)

Definition/Description:
Domain-Driven Design focuses on modeling complex business domains and creating a common language between developers and domain experts. It emphasizes the use of entities, value objects, aggregates, repositories, and services to encapsulate business logic.

Diagram:

```
+------------------+
| Domain Experts   |
+--------+---------+
         |
         v
+--------+---------+
| Ubiquitous Language
+--------+---------+
         |
         v
+------------------+
| Entities, Value Objects, Aggregates
+------------------+
```

Hexagonal Architecture (Ports and Adapters)

Definition/Description:
Hexagonal Architecture, or Ports and Adapters, aims to decouple core business logic from external systems using well-defined interfaces (ports) and adapters that implement these interfaces.

Diagram:

```
+------------------+
|     External     |
|     Systems      |
+--------+---------+
         |
         v
+--------+---------+
|     Adapters     |
+--------+---------+
         |
         v
+--------+---------+
|      Ports       |
+--------+---------+
         |
         v
+------------------+
|   Core Business  |
|     Logic        |
+------------------+
```

Clean Architecture

Definition/Description:
Clean Architecture separates concerns into layers, with the core business logic isolated from external dependencies. The layers include domain, use cases, interface adapters, and frameworks.

Diagram:

```
+------------------+
|   Frameworks/    |
|  Infrastructure  |
+--------+---------+
         |
         v
+--------+---------+
| Interface Adapters|
+--------+---------+
         |
         v
+--------+---------+
|    Use Cases     |
+--------+---------+
         |
         v
+------------------+
|    Domain        |
|    (Entities)    |
+------------------+
```

CQRS and Event Sourcing

Definition/Description:
CQRS separates read and write operations into different models, while Event Sourcing stores state changes as events, enabling the reconstruction of the current state by replaying events.

Diagram:

```
+------------------+
|  Command Model   |
+--------+---------+
         |
         v
+------------------+
| Event Store      |
+--------+---------+
         |
         v
+------------------+
|   Query Model    |
+------------------+
```

API Design and Governance

Definition/Description:
API Design and Governance involve creating and managing APIs to ensure they are consistent, secure, and maintainable. It includes defining specifications, handling versioning, and ensuring compliance with standards.

Diagram:

```
+------------------+
|   API Gateway    |
+--------+---------+
         |
         v
+------------------+
| API Documentation|
+--------+---------+
         |
         v
+------------------+
| Security &       |
|  Monitoring      |
+------------------+
```

Scalable Data Architectures

Definition/Description:
Scalable Data Architectures are designed to handle large volumes of data efficiently using distributed systems, data partitioning, and replication. They support both real-time and batch processing.

Diagram:

```
+------------------+
| Data Sources     |
+--------+---------+
         |
         v
+--------+---------+
| Distributed Data |
| Processing       |
+--------+---------+
         |
         v
+------------------+
| Data Storage     |
| (Data Lakes,     |
|  Warehouses)     |
+------------------+
```

Hybrid Cloud Architectures

Definition/Description:
Hybrid Cloud Architectures integrate on-premises infrastructure with public and private cloud services, providing flexibility, scalability, and cost efficiency.

Diagram:

```
+------------------+
|  On-Premises     |
| Infrastructure   |
+--------+---------+
         |
         v
+--------+---------+
| Private Cloud    |
+--------+---------+
         |
         v
+--------+---------+
| Public Cloud     |
+------------------+
```

Comparison:

Aspect	Event-Driven Architecture (EDA)	Domain-Driven Design (DDD)	Hexagonal Architecture	Clean Architecture	CQRS and Event Sourcing	API Design and Governance	Scalable Data Architectures	Hybrid Cloud Architectures
Primary Focus	Asynchronous communication and event handling	Modeling complex domains with rich business logic	Decoupling core logic from external dependencies	Separation of concerns and testability	Separation of read and write models, event storage	Standardization and control over APIs	Efficient data storage and processing	Integration across on-premises and cloud
Key Components	Event Producers, Event Consumers, Event Brokers	Entities, Aggregates, Repositories, Services	Domain layer, Application layer, Adapters	Layers: Domain, Use Cases, Interface Adapters, Frameworks	Command handlers, Query handlers, Event store	API Gateway, API Documentation, Security	Data Lakes, Data Warehouses, Distributed Databases	Cloud services, On-prem infrastructure, Connectivity
Communication Style	Asynchronous, event-based	Synchronous and asynchronous	Synchronous	Synchronous and asynchronous	Synchronous commands, asynchronous events	HTTP, REST, GraphQL, gRPC	Batch processing, real-time processing	Synchronous and asynchronous

Advantages	Loose coupling, scalability, flexibility	Better domain understanding, maintainability	High testability, flexibility	Testability, maintainability, flexibility	Scalability, performance, auditability	Consistency, security, scalability	Scalability, performance, reliability	Flexibility, scalability, cost efficiency
Challenges	Complexity, debugging, testing	Complexity, steep learning curve	Complexity, setup time	Complexity, requires discipline	Complexity, learning curve, potential data duplication	Complexity, governance overhead	Complexity, data consistency	Complexity, security concerns, data transfer costs
Use Cases	Microservices, real-time data processing	Complex domains, microservices	Microservices, maintainable applications	Enterprise applications, microservices	Financial applications, high-performance systems	Public APIs, enterprise integrations	Big data analytics, AI/ML workloads	Disaster recovery, global applications
Example Technologies	Kafka, RabbitMQ, AWS SNS/SQS	Java, C#, .NET, Entity Framework	Spring, Quarkus, NestJS	Angular, React, Spring Boot	Axon Framework, EventStore, Kafka	Swagger, Postman, Apigee	Hadoop, Spark, BigQuery	AWS, Azure, Google Cloud, VMware
Patterns	Event Sourcing, Pub/Sub	Aggregates, Repositories, Factories, Value Objects	Ports and Adapters	Layered Architecture, Dependency Rule	Event Sourcing, Command-Query Responsibility Segregation (CQRS)	API Gateway, Rate Limiting, Throttling	Lambda Architecture, Kappa Architecture	Multi-cloud, Hybrid deployments
Example Applications	Uber (real-time data processing)	Amazon (order management system)	Alura (online learning platform)	Google Ads (advertising platform)	Banking systems (transaction processing)	Stripe (payment processing API)	Netflix (data analytics platform)	Capital One (hybrid cloud infrastructure)