ChatGPT

Architecture Design Patterns

I. Layering & Separation of Concerns

Pattern	Description	Use Case
Layered (N-tier)	Divide responsibilities into layers (UI, Business, Data)	Web apps, enterprise systems
Model-View-Controller (MVC)	Separates Model, View, and Controller logic	UI apps, web frameworks
Model-View-ViewModel (MVVM)	View binds to ViewModel exposing data and commands	Desktop/mobile apps (WPF, Android)
Model-View-Presenter (MVP)	Presenter handles interaction logic between view and model	Legacy GUI frameworks

II. Distributed & Scalable Systems

Pattern	Description	Use Case
Microservices	Independent, loosely coupled services	Scalable web platforms
Service-Oriented Architecture (SOA)	Services interact via ESB or orchestration	Enterprise systems, B2B
Client-Server	Clients make requests to a central server	Web apps, databases
Peer-to-Peer (P2P)	Each node acts as client and server	Torrent, blockchain
Broker	Middleware routes messages between components	Message-based systems
Message Bus	Centralized bus for service interaction via messages	Integration platforms
Event-Driven	Components react to events asynchronously	Real-time analytics, IoT
Serverless / FaaS	Functions triggered by events, managed by cloud	APIs, async processing
Space-Based Architecture	Memory-centric shared data grid	High throughput systems

III. Data Flow & Processing

Pattern	Description	Use Case
Pipe and Filter	Data flows through filters (stages)	Compilers, ETL pipelines
Batch Processing	Scheduled large data jobs	Reporting, data aggregation
Stream Processing	Continuous processing of data streams	Kafka Streams, Flink
Data-Centric / Blackboard	Shared data structure accessed by components	AI, scientific systems
Repository	Centralized data access logic	ORM, database logic
CQRS	Separate read and write models	Event sourcing, high read/write systems
ETL	Data ingestion pattern	Data warehouses, pipelines

IV. Control Flow & Logic

Pattern	Description	Use Case
Orchestration	Central controller manages workflow	BPM tools, workflow engines
Choreography	Services react based on events, no central control	Microservices, EDA
Rule-Based Architecture	Uses rules to drive decision-making	Fraud detection, compliance (Drools, OPA)
State Machine	Encapsulates transitions between states	Protocols, games
Interpreter	Defines language grammar and evaluation	Scripting, query engines
Workflow Engine	Declarative control of steps	BPMN, Airflow DAGs

V. Modularity & Reusability

Pattern	Description	Use Case
Component-Based	Application built from replaceable components	React, Angular
Plugin Architecture	Extend core functionality with external modules	IDEs, browsers
Microkernel (Plug-in)	Core system with extensible plugins	OS, compilers
Hexagonal	Core isolated via adapters	DDD, testable systems
Clean Architecture	Layers around domain, with DI	Maintainable apps

VI. Security & Policy

Pattern	Description	Use Case
Policy Decision Point	Separates policy evaluation from enforcement	OPA, Rego
Access Control Architecture	RBAC/ABAC-based permission checks	Enterprise systems
Zero Trust Architecture	Verifies identity on each access	Cloud-native security

VII. Reliability & Recovery

Pattern	Description	Use Case
Retry/Dead Letter Queue	Retry failed tasks and isolate failures	Task queues, email systems
Circuit Breaker	Prevents cascading failures	Netflix Hystrix, resilience
Bulkhead	Isolates components to contain failures	Microservices
Watchdog / Supervisor	Restarts failing components	Erlang OTP, Kubernetes

VIII. Deployment & Operations

Pattern	Description	Use Case
Immutable Infrastructure	Deploying artifacts without modification	Docker, CI/CD
Sidecar	Companion process provides support (e.g. logging)	Service mesh (Istio)
Ambassador	External interface to service	API Gateway
Gateway / API Gateway	Entry point for APIs with cross-cutting concerns	Kong, NGINX
Service Mesh	Network layer manages microservice communication	Istio, Linkerd

BONUS: Cross-Cutting Architectural Concepts

Concept	Explanation
Separation of Concerns	Break app into modules with specific responsibilities
Encapsulation	Hide internal details of modules
Scalability	Horizontal/vertical scaling support
Fault Tolerance	Graceful handling of errors/failures

Concept	Explanation
Asynchronous Messaging	Improves decoupling and performance