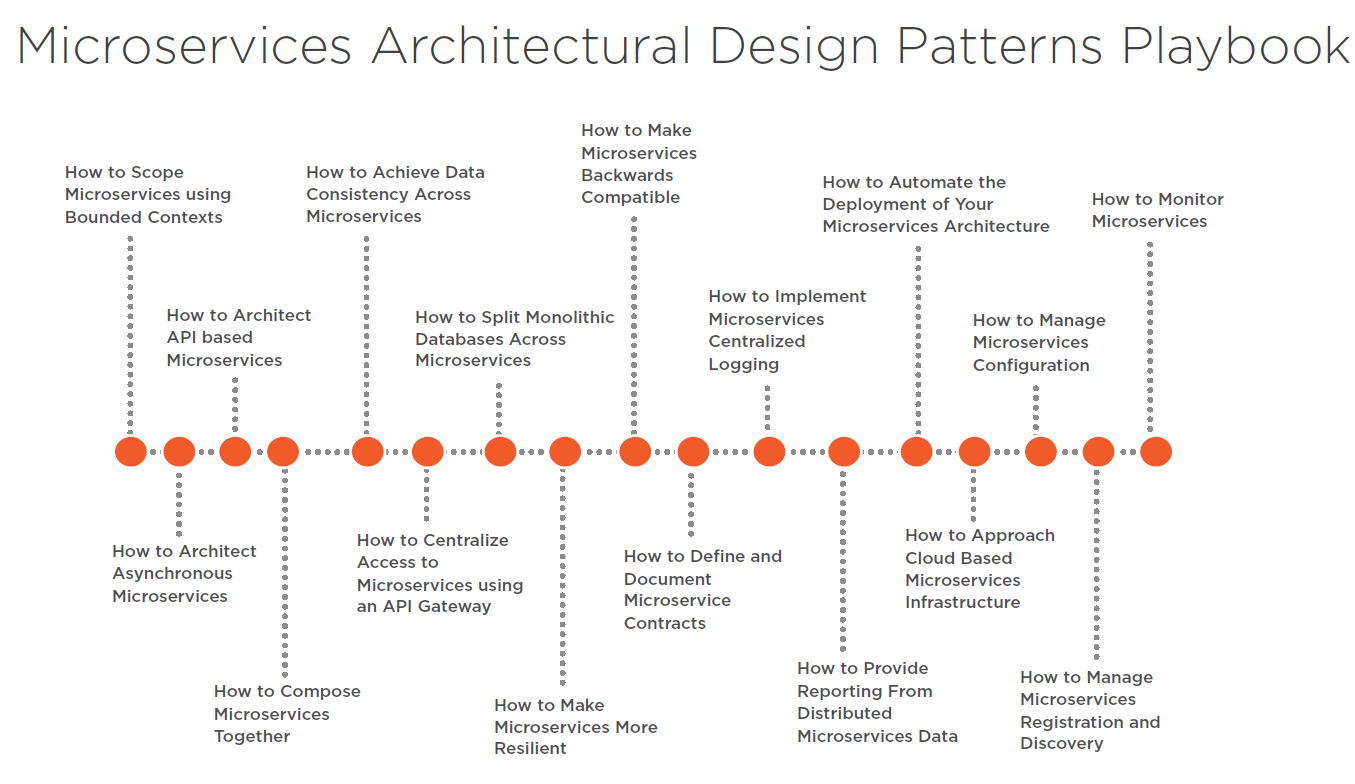
**Microservices**



|  |  |
| --- | --- |
| **PLDAE**  Problem -Learn -Design -Adapt -Evaluate | **Problem** : Sold products on the internet **Learn** : e-commerce domain, required specs, monolithic **Design** : monolithic, microservices **Adapt** : tech stack, java or .net ecosystem **Evaluate** : pros-cons , what’s wrong this design |
| Problem  Solution | 1. Problem: Increased Traffic, Handle More Request 2. Problem: Break Down Application into Microservices 3. Problem: Direct Client-to-Service Communication 4. Problem: Inter-service communication makes heavy load on network traffic 5. Problem: Chat with Support Agent 6. Problem: Service-to-Service Communications Chain Queries 7. Problem: Long Running Operations Can’t Handle with Sync Communication 8. Problem: Database Bottlenecks when Scaling, Different Data Requirements For Microservices 9. Problem: Cross-Service Queries and Write Commands on Distributed Scaled Databases 10. Problem: Manage Consistency Across Microservices in Distributed Transactions 11. Problem: Handle Millions of Events Across Microservices 12. Problem: Database operations are expensive, low performance 13. Problem: Deploy Microservices at Anytime with Zero-downtime and flexible scale |
| Micro  services Building Blocks | 1. **Microservices Decomposition — Breaking Down Microservices** 2. **Microservices Communications (Sync / Async Message-Based Communication, API Gws)** 3. **Microservices Data Management (Databases / Query / Commands)** 4. **Microservices Transaction Management** 5. **Microservices Distributed Caching** 6. **Microservices Deployments with Containers and Orchestrators** 7. **Microservices Resilience with Distributed Logging, Tracing and Health** 8. **Microservices Testing; Contract Testing, Integration Testing, E2E Test** 9. **Microservices Security; Authentication and Authorization** |
| Design Pattern Used in Micro-  Services | |  |  |  | | --- | --- | --- | | **Service Aggregator pattern** | **Materialized view pattern** |  | | **CQRS Design pattern** | **Saga pattern for distributed Transaction**  **Compensating transaction pattern** |  | | **Event Sourcing Pattern** |  |  | |
|  |  |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| GitHub - mehmetozkaya/Design-Microservices-Architecture-with-Patterns-Principles:  Design Microservices Architecture with Patterns & Principles. We're going  to learn how to Design Microservices Architecture with using Design Patterns,  Principles and the ... |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Microservices Checklist — Cover All Aspects of Microservices Architecture**

[**https://medium.com/design-microservices-architecture-with-patterns/microservices-checklist-cover-all-aspects-of-microservices-architecture-e3243fe1a53f**](https://medium.com/design-microservices-architecture-with-patterns/microservices-checklist-cover-all-aspects-of-microservices-architecture-e3243fe1a53f)

## Microservices Synchronous Communication

* Sync Communication Types: REST / gRPC / GraphQL / WebSocket
* API Gateway Requirements: Discovery, Registry, Route, Aggregation, Composition, Authentication
* Service-to-service Integration Considerations : Backend Microservices / 3rd party calls / B2B Gateways / IoT Devices
* Headless Architecture Decisions: Micro-frontend designs, Edge Functions

## Microservices Asynchronous Communication

* Async Com Types: 1–1 / 1-many (point-to-point / publish-subscribe)
* Async Com Way: Fire&Forget / Callback
* Async Tools: Message Broker / Event Bus / Queue / DLQ
* Async Patterns: Pub-Sub / Fanout / Topic Queue Chain / Event Filters / Rule Engine
* EDA Design Considerations: Event Backbone / Event Queue Topic Design / Real-time Event Streams / Event Filters, Routers, Deserializers / Event Stores

Lets continue with other sub components of Microservices Building Blocks.

## Microservices Data Management (Databases / Query / Commands)

* Database Chooses
* Queries
* Commands — Consistency
* Distributed Transactions
* Caching

So in this building block, we should also go one more deep level and identify consideration topics as bellowed:

## Database Chooses

* DB Types : Relational / NoSQL — Document, Key/Value, Wide Column, Graph, Time Series, DWH
* DB Scale Options : Master-Slave/ Masterless, Replication, Sharding, Read/Write
* DB Capabilities: CDC / Streaming / Sharding / K8s Management DBs / DWH Reporting / Big Data Analysis / AI/ML model processing

## Queries

* API Gateway / Composition / Aggregation / BFF
* CQRS / Materialized View / Event Sourcing / Read Database
* Analytics/AI-ML/Reporting/Real-time stream analysis and processing / Big Data Analysis

## Commands — Consistency

* Consistency: Strict / Eventual Consistency
* CQRS / Write Database / ACID / Event Sourcing / Repayable Snapshot / Single source of truth

## Microservices Distributed Transactions

* Transaction Management: 2PC / SAGA (Orchestrator / Choreography )
* Outbox / Dual Write / Change Data Capture / Message Broker Event Bus / read-after-write consistency
* MS Design : Idempotency / Stateless / Immutable events

## Microservices Distributed Caching

* Cache Type: Static Content (CDN) / In-memory / Distributed
* Cache Settings: Cache Expire / TTL / Hit — Miss — Invalidation

Lets continue with other sub components of Microservices Building Blocks.

## Microservices Deployments with Containers and Orchestrators

* IaC
* Devops CI/CD
* Network Management

So in this building block, we should also go one more deep level and identify consideration topics as bellowed:

## IaC

* Multi-Cloud Deployments — EKS, AKS, GKS
* PaaS, Managed K8s (Openshift), Databases (k8cassandra, ksql), EventBus (Confluence Kafka), Api Gw
* Serverless options, FAAS, Serverless and Edge Functions, CDN

## Devops CI/CD

* Deployments: Containers, Orchestrators, Serverless options, Frameworks: Openshift
* Pipeline Steps, Automation, Devops tools
* Package Management: Helm Charts, Artifactory Hubs, Re-usable container registry
* Deployment Strategy: Blue/Green, Canary, Recreate
* Configurations : Config Server, Encryption, Key Vault Management

## Network Management

* Load Balancing, Ingress, CDN, DNS Management, Cluster Management, Disaster recovery

Lets continue with other sub components of Microservices Building Blocks.

## Microservices Resilience

* Observability / Logging
* Monitoring
* Tracing
* Resilient / Fault Tolerance

So in this building block, we should also go one more deep level and identify consideration topics as bellowed:

## Observability / Logging

* Centralized Logging, Distributed Logging, Dashboards, ELK Stack
* Log Id per Request, Service App Logs, Integration Logs

## Monitoring

* System Monitoring: CPU, Ram
* Application Monitoring: Queue Count, unprocessed messages
* Health Checks : Microservices, Databases, Brokers Event Buses, 3rd party systems

## Tracing

* Distributed Transactions, Transaction Id, Correlation Id, E2E Request Id, Event Id
* OpenTelemetry Compliance: Jeager, Zipkin

## Resilient / Fault Tolerance

* API Call Management : Retry, Circuit Breaker, Bulkhead, Rate limiting, Fault Tolerant
* Kubernetes Patterns & Best Practices : Sidecar, Service Mesh, Service Proxy
* Async Fault Alarms: DLQ, Notifications, Alarms

Lets continue with other sub components of Microservices Building Blocks.

## Microservices Testing; Contract Testing, Integration Testing, E2E Test

* Pre-Deployment Test : Unit / Integration / Contract Test
* Post-Deployment Test : E2E SIT Test / Consumer-Driven Test / Chaos Test

## Microservices Security; Authentication and Authorization

* Authentication
* Identity Servers: Managed (IS4, Keycloak) and Serverless options (Cognito)
* Protocols: OAuth2.0 , OpenIDConnect flows
* Token Types and Claims
* Authorization
* API Gw Authentication flows, Claim based authorizations, RBAC
* Configurations
* Encryption, Key Vault Management
* SSL and TLS, Firewalls
* Audit Logs