**Design Patterns**

1. **MVC**
2. **Factory**
3. **Builder Pattern**
4. **Singleton Pattern**
5. **Saga Pattern**
6. **Circuit Breaker Pattern**
7. **API Gateway Pattern**
8. **Side Car**
9. **Adapter Pattern**
10. **Ambassador Pattern**

**Side Car (2 container in same pod – same ip, network) – local host**

**Example: Side container can process log and take action, shared logs**

**A diagram of a computer program

Description automatically generated**

**Adapter Pattern: (uses Side Car Pattern)**

**Side container can create unified data needed for outside processing**

**Example:**

**1) Splunk is good example where we want to send a request data in specific so each micro service can use this pattern and create common data format which can be consumed by splunk or any other app.**

**2) eCommerce integrate with 3rd party payment service, Payment service expect specific request format that’s different then the one e-commerce platform generates. This container can perform task of converting as per payment service needs.**

**3) Example of log4j , logback or java util logs can be unified using Slf4j so changing the underlining log implementation won’t hurt.**

**4) API Gateway is another good example where all the request will be handled by api gateway and underlining services won’t matter for end user.**

**A diagram of a computer

Description automatically generated**

**Ambassador Pattern:**

**Database connections can be managed by side car service and main service can access it as localhost. So all complexity like connection pool, retry policies, network communication will be manage by proxy service instead of main application.**

**A blue rectangle with text and arrows

Description automatically generated**

1. **API Gateway Pattern: An API gateway is the single entry point for all clients. It handles requests in one of two ways: routing them to the appropriate microservice, or by fanning out a request to multiple services.**
2. **Circuit Breaker Pattern: The circuit breaker pattern prevents a network or service failure from cascading to other services. When the circuit breaker detects a number of failed attempts above a threshold, it trips, and all further calls to the service return with an error, without the protected call being made. After a timeout period, it allows a limited number of test requests to pass through.**
3. **Saga Pattern: This pattern is used to manage data consistency across microservices in distributed transaction scenarios, often by using a series of local transactions and compensating transactions.**

**Good real time Example:**

1. **Created Advertiser, Campaign, Model, Start Model Generation**
2. **Data Asset, Hierarchy, Option, Meta data(advertiser,data partner)**

**All these happens independently and asynchronously using kafka.**

**Eventual consistently is achieved and compensating actions taken.**

**Long lived transactions with eventual consistency.**

1. **Event Driven Pattern:**

**Works on facilitating communication between micro services, Does not guarantee eventual consistency. Asynchronous communication and decoupling of services.**

**Difference with SAGA is possibly, SAGA needs to go step by step as it’s a long running single transaction, Where as event driven can call in any order based on requirement.**

**But both these pattern compliments each other very well.**

1. **Circuit Breaker Pattern:**

**This pattern is resilience of system and improve stability, in distributed environment.**

**This is often used with event driven or saga pattern. This is use to handle faults and handle failures gracefully.**

**Good real time Example:**

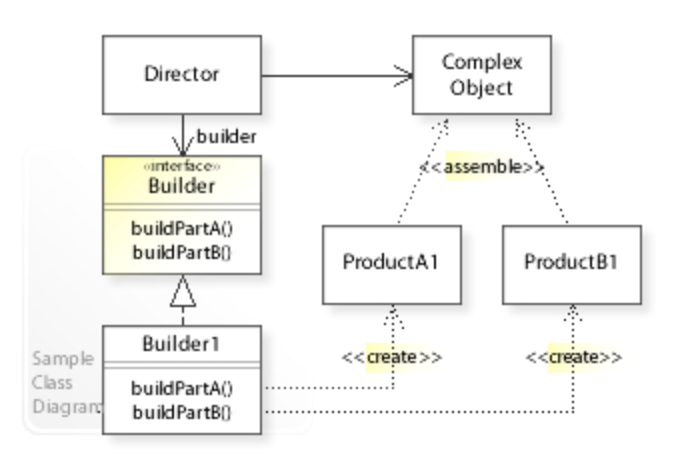
**Created Advertiser, Campaign, Model, Start Model Generation.**

**If for any reason Campaign service is down, Circuit breaker pattern helps in minimizing the request coming to campaign or other downward services and service down or failure can be handled gracefully.**

1. **Builder Pattern**

**In order to create a complex object with many optional configurations. This will be clean and more readable way of creating such objects. This type of pattern helps such use cases.**

**This provides flexible way to create complex objects.**

****