**Microservices Patterns:**

source : <https://microservices.io/patterns/index.html>

**[1] Application architecture patterns**

*Context: You are developing a server-side enterprise application. It must support a variety of different clients including desktop browsers, mobile browsers and native mobile applications. The application might also expose an API for 3rd parties to consume. It might also integrate with other applications via either web services or a message broker. The application handles requests (HTTP requests and messages) by executing business logic; accessing a database; exchanging messages with other systems; and returning a HTML/JSON/XML response. There are logical components corresponding to different functional areas of the application.*

*Problem: What’s the application’s deployment architecture?*

*Forces: (i) There is a team of developers working on the application (ii) New team members must quickly become productive (iii) The application must be easy to understand and modify (iv) You want to practice continuous deployment of the application (v) You must run multiple instances of the application on multiple machines in order to satisfy scalability and availability requirements (vi) You want to take advantage of emerging technologies (frameworks, programming languages, etc)*

Solution 1: **Monolithic Architecture**

1. a single Java WAR file. (ii) a single directory hierarchy of Rails or NodeJS code

Diagram

Description automatically generated

|  |  |
| --- | --- |
| Advantages | Drawbacks |
| Simple to develop - the goal of current development tools and IDEs is to support the development of monolithic applications | The large monolithic code base intimidates developers, especially ones who are new to the team. The application can be difficult to understand and modify. As a result, development typically slows down. |
| Simple to deploy - you simply need to deploy the WAR file (or directory hierarchy) on the appropriate runtime | Overloaded IDE - the larger the code base the slower the IDE and the less productive developers are.  Overloaded web container - the larger the application the longer it takes to start up. |
| Simple to scale - you can scale the application by running multiple copies of the application behind a load balancer | Continuous deployment is difficult.  Scaling the application can be difficult  Obstacle to scaling development. Obstacle to scaling development |

Solution 2: **Microservice Architecture**

Define an architecture that structures the application as a set of loosely coupled, collaborating services. Each service is:

1. Highly maintainable and testable - enables rapid and frequent development and deployment
2. Loosely coupled with other services - enables a team to work independently the majority of time on their service(s) without being impacted by changes to other services and without affecting other services
3. Independently deployable - enables a team to deploy their service without having to coordinate with other teams
4. Capable of being developed by a small team - essential for high productivity by avoiding the high communication head of large teams

Diagram

Description automatically generated

|  |  |
| --- | --- |
| Advantages | Drawbacks |
| Enables the continuous delivery and deployment of large, complex applications  (i) Improved maintainability (ii) Better testability (iii) Better deployability (iv) Better deployability | Eliminates any long-term commitment to a technology stack.  (i) Developers must implement the inter-service communication mechanism |
| Each microservice is relatively small | Deployment complexity. |
| Improved fault isolation.  Eliminates any long-term commitment to a technology stack. | Increased memory consumption. The microservice architecture replaces N monolithic application instances with NxM services instances. If each service runs in its own JVM (or equivalent) |

[2] Decomposition

[3] Refactoring to microservices

[4] Data management

Database per Service

Shared database

Saga

API Composition

CQRS

Domain event

Event sourcing

[5] Transactional messaging

[5] Testing

[6] Deployment patterns

[7] Cross cutting concerns

[8] Communication style

D[9] External API

9.1 API Gateway

Context : Online store (which use microservices architecture pattern) implement product detail page develop multiple version of product detail page:

e.g. 1. HTML/Java script based UI for desktop and mobile browsers

2. Native android/iphone clients (clients interact with server via REST APIs)

3. And, expose product details via REST API for the use by 3rd party applications.

Problem : How do the clients of a Microservices-based application access the individual services?

(1. microservices typically provide fine-grained APIs. (ii) different clients need different data (iii) n/w performance is different for different type of clients (iv) no. of services instance and their locations changes dynamically i.e. host and ports(v) partitioning into services can change over time and should be hidden from clients.

Solution : Implement an API gateway that is the single entry point for all the clients.

It can handle request in two ways : (1)some requests simply proxied/routed to the appropriate service. (ii) It can handles other requests by fanning out to multiple services.

Diagram, timeline

Description automatically generated

9.2 Backends for Frontends

A variation of the above pattern is "Backends of Frontends" pattern. It defines a separate API gateway for each kind of client.

Diagram

Description automatically generated

[10] Service discovery

Problem :

()

Solution:

[D][11] Reliability

Problem : How to prevent a network or service failure from cascading to other services?

()

Solution: A service client should invoke a remote service via a proxy that functions in a similar fashion to an electrical circuit breaker. When the number of consecutive failures crosses a threshold, the circuit breaker trips, and for the duration of a timeout period all attempts to invoke the remote service will fail immediately. After the timeout expires the circuit breaker allows a limited number of test requests to pass through. If those requests succeed the circuit breaker resumes normal operation. Otherwise, if there is a failure the timeout period begins again.

D[12] Security

12.1 Access token

Problem: How to communicate the identity of the requestor to the services that handle the request?

(Services often need to verify that a user is authorized to perform an operation)

Solution : The API Gateway authenticates the request and passes an access token (e.g. JSON Web Token) that securely identifies the requestor in each request to the services. A service can include the access token in requests it makes to other services.

[D][13] Observability

13.1 Log aggregation

Problem : How to understand the behaviour of an application and troubleshoot problems?

(Any solution should have minimal runtime overhead)

Solution: Use a centralized logging service that aggregates logs from each service instance. The users can search and analyse the logs. They can configure alerts that are triggered when certain messages appear in the logs.

13.2 Application metrics

Problem : How to understand the behaviour of an application and troubleshoot problems?

(Any solution should have minimal runtime overhead)

Solution:

Instrument a service to gather statistics about individual operations. Aggregate metrics in centralized metrics service, which provides reporting and alerting. There are two models for aggregating metrics:

push - the service pushes metrics to the metrics service

pull - the metrics services pulls metrics from the service

13.3 Audit logging

Problem : How to understand the behaviour of users and the application and troubleshoot problems?

(It is useful to know what actions a user has recently performed: customer support, compliance, security, etc.)

Solution: Record user activity in a database.

13.4 Distributed tracing

Problem : How to understand the behaviour of an application and troubleshoot problems?

(External monitoring only tells you the overall response time and number of invocations - no insight into the individual operations.

Log entries for a request are scattered across numerous logs)

Solution:

Instrument services with code that:

Assigns each external request a unique external request id

Passes the external request id to all services that are involved in handling the request

Includes the external request id in all log messages

Records information (e.g. start time, end time) about the requests and operations performed when handling a external request in a centralized service

13.5 Exception tracking

Problem :How to understand the behaviour of an application and troubleshoot problems?

(Exceptions must be de-duplicated, recorded, investigated by developers and the underlying issue resolved

Any solution should have minimal runtime overhead)

Solution: Report all exceptions to a centralized exception tracking service that aggregates and tracks exceptions and notifies developers.

13.6 Health check API

Problem : How to detect that a running service instance is unable to handle requests?

(An alert should be generated when a service instance fails

Requests should be routed to working service instances)

Solution:

A service has an health check API endpoint (e.g. HTTP /health) that returns the health of the service. The API endpoint handler performs various checks, such as

the status of the connections to the infrastructure services used by the service instance

the status of the host, e.g. disk space application specific logic

A health check client - a monitoring service, service registry or load balancer - periodically invokes the endpoint to check the health of the service instance.

13.7 Log deployments and changes

Problem : How to understand the behaviour of an application and troubleshoot problems?

(It useful to see when deployments and other changes occur since issues usually occur immediately after a change)

Solution: Log every deployment and every change to the (production) environment.

[14] UI patterns

You have applied the Microservice architecture pattern. Services are developed by business capability/subdomain-oriented teams that are also responsible for the user experience. Some UI screens/pages display data from multiple service. Consider, for example, an Amazon-style product detail page, which displays numerous data items including:

Basic information about the book such as title, author, price, etc.

Your purchase history for the book

...

...

14.1 Server-side page fragment composition

Problem : How to implement a UI screen or page that displays data from multiple services?

()

Solution: Each team developers a web application that generates the HTML fragment that implements the region of the page for their service. A UI team is responsible for developing the page templates that build pages by performing server-side aggregation (e.g. server-side include style mechanism) of the service-specific HTML fragments.

14.2 Client-side UI composition

Problem : How to implement a UI screen or page that displays data from multiple services?

()

Solution: Each team develops a client-side UI component, such an AngularJS directive, that implements the region of the page/screen for their service. A UI team is responsible implementing the page skeletons that build pages/screens by composing multiple, service-specific UI components

-----

API Gateway: (i) redirecting incoming requests (routing the request) (ii) security (iii) load balancing (server side) (check client side load balancing)

Eureka : naming service

configuration service: