Microservices Advantages

Contents

[1. Sample Artefact Application 1](#_Toc35527225)

[2. Maintainability 1](#_Toc35527226)

[3. API Accessibility 2](#_Toc35527227)

[4. Garbage Collection Tuning 2](#_Toc35527228)

[5. Code Refactoring 2](#_Toc35527229)

[6. Availability 2](#_Toc35527230)

[7. Troubleshooting 3](#_Toc35527231)

[8. Security Considerations 3](#_Toc35527232)

[9. Load Balancing 3](#_Toc35527233)

[10. Error Handling 3](#_Toc35527234)

[11. Secured and Unsecured API for same Microservice 3](#_Toc35527235)

# Sample Artefact Application

Consider an artefact application which has several modules. Customer Registration, Order, Invoice, Tracker, Administrator, Supplier Registration, Complaint Resolver, Artefact Maintainer, Artefact Selector, Forgot Password, Customer Login, Supplier Login, etc...

For Registration and Forgot Password the application sends OTP to Customer and Supplier. For Admin and Supplier Login application has MFA. MFA criteria are different for Admin and Supplier.

# Maintainability

Every module has some constants those are defined through properties file. Every microservice can contain its properties only in one file.

However, in monolithic application the properties may be consolidated in one file or different file per module.

If maintained in one file then file size becomes big.

If maintained in different files then number of files is big.

# API Accessibility

Microservices are generally exposed via REST APIs. The contract of APIs is same irrespective of client (Other Microservices/ External Clients). This helps in maintaining uniformity in API accessibility.

For example, say a microservice sends OTP for various activities. Say, this OTP is required for various activities such as

1. New User Registration
2. Existing User Forgot Password
3. End User Clicks on a link to resend OTP.

In every scenario the contract uniformity can be maintained.

# Garbage Collection Tuning

Consider Order module has more concurrent (10000) users than Administrator module (100). Memory requirement for both modules will be different. Similarly GC algorithm and its tuning may also differ and will be focused specific to the particular Microservice.

The above note can be considered for every module.

However, combining all modules in a monolithic application will accumulate the concurrent users and memory requirements. Also, GC tuning will have to consider each and every aspect of the application.

# Code Refactoring

Consider that code is refactored specific to Supplier Login microservice. Hence, only Supplier Login microservice has to be tested. Other microservices (specifically Customer and Admin Login) need not to be rested.

However, if code under the Jar that included in above three microservices is modified then all three microservices will have to be tested.

The same stands true for the Jar that is included in all microservices.

# Availability

Say, the application is available in various parts of the country. Hence, the application is deployed in various zones.

The availability requirement of Order microservice and Admin microservice may differ w.r.t. every zone. The requirements vary w.r.t. number of concurrent users. Hence, in different zones different modules may be scaled down and scaled up w.r.t. requirement.

In few zones the only one node of Administrator module may be sufficient in non-peak hours.

However, if all modules are combined in a monolithic application then scaling may vary. Reason is number of concurrent users for all modules.

# Troubleshooting

In case of any issues the troubleshooting process is focused on the particular microservice.

However, in monolithic application the testing may need to be carried on entire application.

# Security Considerations

Consider that every microservice is secured with JWT token. However, the Supplier and Admin Login modules have different MFA for login process.

Hence, the MFA functionality will be added only for those microservices and not for others. This way the configuration is restricted to only those microservices.

# Load Balancing

The load balancer algorithm may differ with respect to different microservice.

In monolithic application there is no chance to change load balancer algorithm with respect to every module.

# Error Handling

Say if Order microservice invokes invoice microservice. However, invoice microservice becomes unresponsive or does not return in stipulated time. In this case Circuit Breaker pattern can help to show user understandable message.

However, in a monolithic application because of synchronous calls the delay cannot be handled.

# Secured and Unsecured API for same Microservice

Consider **Artefact Selector** microservice. This can be available for guests and logged in users. For Guest users the API is unsecured and can be served via different nodes.