

[Udemy Courses](#)[Architecture](#) ▾[Development](#) ▾[DevOps](#) ▾[Test Automation](#) ▾[Downloads](#)[About Me](#)[Topics](#)

Recent
Posts

Spring

WebFlux

Aggregation ^

MicroServices – How To Share DTO (Data Transfer Objects)

6 Comments / Architecture, Articles, Best Practices, Framework, Java, Maven, MicroService, Spring, Spring Boot, Spring WebFlux, Utility / By vlins / April 10, 2020

Overview:

Over the years, MicroServices have become very popular. They are smaller, modular, easy to deploy and scale etc. However MicroService architectures do have some challenges. MicroServices have specific responsibilities. In order to complete an application workflow / a task, multiple MicroServices might have to work together. For example, an user would like to buy a product in a web application providing his payment information and address for shipping. Here we could have 4 different MicroServices involved to complete the workflow.

- user-service
- order-service
- payment-service
- shipping-service

Choreograph
hy Saga
Pattern With
Spring Boot
Spring
WebFlux
WebSocket
gRPC Web
Example
Orchestratio
n Saga
Pattern With
Spring Boot

 Selenium

WebDriv

er - How

MicroServices communicate among themselves using contracts! That is, If the order-service needs to talk to the payment-service for payment processing, order-service needs to know how to send the request to the payment-processing like the required information for processing. Similarly it should also know what to expect from the payment-service. It is called contract in the MicroServices world! This contract should not break or other service developers should be informed about the breaking changes.


Note: If you are using [protocol-buffers/gRPC](#), you are good as you would not face this issue. So you can skip. Otherwise you should read 😊

Contract:

A payment-service might expect a request with below format to process an incoming request.

```
{
  "user":{
    "firstName":"vins",
    "lastName":"vins",
    "address":"123 non-main street"
  },
  "payment":{
```

To Test
REST API

 Introduci
ng
PDFUtil -
Compare
two PDF
files
textually
or
Visually



JMeter -
How To
Run
Multiple
Thread
Groups ii ^

```
"creditcard": "vins",  
"expiryMonth": "12",  
"expiryYear": "2030",  
"amount": "100",  
"currency": "USD"  
}  
}
```

So order-service must send the request in the above format for payment processing. Payment-service could process that successfully and send a response with status to the order-service. So that based on the status, order-service might take appropriate action. Order-service and Payment-service could have some models as shown here to send the request and receive the response.

```
public class User {  
  
    private String firstName;  
    private String lastName;  
    private String address;  
  
    // getters & setters  
  
}
```

Multiple
Test
Environm
ents



Selenium

WebDriv
er -

Design
Patterns
in Test

Automati
on -

Factory
Pattern



Kafka

Stream

With



If the payment is successful, order-service might contact the shipping-service to ship the product. Again the contract for the shipping-service could be different!

```
{
  "user":{
    "firstName":"vins",
    "lastName":"vins",
    "address":"123 non-main street",
    "phone" : "123-123-1231"
  },
  "product":{
    "id": "TV123",
    "quantity": 1
  },
  "shippingMethod": "ground"
}
```

The problem here is that, We need models (Java classes) to represent the request and response objects in each MicroService.

Common Mistake 1:

Often developers copy the DTOs from 1 microservice (**payment-service**) into other microservice (**order-service**)! Of course this is a bad practice and it leads redundant

Spring

Boot

 JMeter -

Real

Time

Results -

InfluxDB

&

Grafana -

Part 1 -

Basic

Setup

 JMeter -

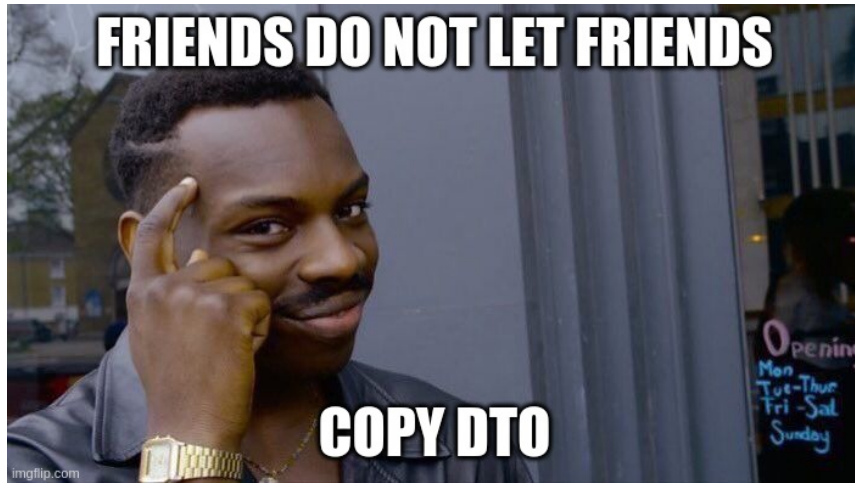
Distribut

ed Load

Testing

^

code maintenance.



Common Mistake 2:

Sometimes developers create a separate maven module for common DTOs. For example, their project structure would be as shown here.

```
shopping-cart
  common-dto
    - userDto
    - paymentDto
    - productDto
  user-service
  order-service
```

using

Docker

 JMeter -

How To

Test

REST API

/

MicroSer

vices

 JMeter -

Property

File

Reader -

A custom

config

element

^



payment-service
shipping-service

Each microservice would be adding the **common-dto** as maven dependency. Now it sounds like a good solution. There is no redundant code here. In future if we need to include, say an 'id' field for user, updating **UserDTO** in the **common-dto** module is enough.

Even though it sounds like a good solution it is actually not! When we dive deeper into the each microservice requirement, we could understand that better. One thing what we need to understand is user model in the payment-service is not same as the user model in the shipping-service or user model in the user-service!!

For example, each and every microservice could represent the user more or less as shown here.

- payment-service:

```
"user": {  
    "firstName": "vins",  
    "lastName": "vins",  
    "address": "123 non-main street"  
}
```

Selenium
WebDriv
er - How
To Run
Automat
ed Tests
Inside A
Docker
Containe
r - Part 1

Categori
es



- shipping-service:

```
"user": {  
  "firstName": "vins",  
  "lastName": "vins",  
  "address": "123 non-main street",  
  "phone" : "123-123-1231"  
}
```

- order-service:

```
"user": {  
  "id" : "43445",  
  "orders": [  
    {  
      "id" : "2232",  
      "itemId": "TV123",  
      "date": "01/01/2020",  
      "status": "complete"  
    }  
  ]  
}
```

Architecture

(62)

Arquillian (9)

Articles

(204)

AWS / Cloud

(17)

AWS (4)

Best

Practices

(75)

CI / CD /

DevOps (51)

Data Stream

/ Event

Stream (27)

Database (9)

^

So, we can not create a common user model which can be used across all the Microservices. We could also have a simple validation in the DTOs. For example, shipping-service must want to have a phone number of the user to send the shipping status. So we might throw an **IllegalStateException** before sending the user model to the shipping-service without the phone number. But payment-service might not need this information at all. So you might not want an exception when you call the payment-service with the common user model.

The real challenge here is – we need to reduce the code duplication. But at the same time we also can not generalize any model to use across all the services.

Then how to solve this problem!!?

Services With Client Library:

If you want to use a Database, say Postgres, you need a client library to connect to the DB. There is no common library! Each service provider provides their own library! In this case, Postgres provides a Java library. Similarly MongoDB provides their own library. If you take any application or service, like AWS, they provide their own java library to use their services. We are going to apply the same concept here!

Each MicroService is independent, has specific responsibilities, looks for specific information in specific format from other services to process any request (contract).

Design

Pattern (41)

Architectural

Design

Pattern (26)

Factory

Pattern (1)

Kubernetes

Design

Pattern (18)

Strategy

Pattern (1)

Distributed

Load Test

(9)

Docker (24)

ElasticSearch

h (2)



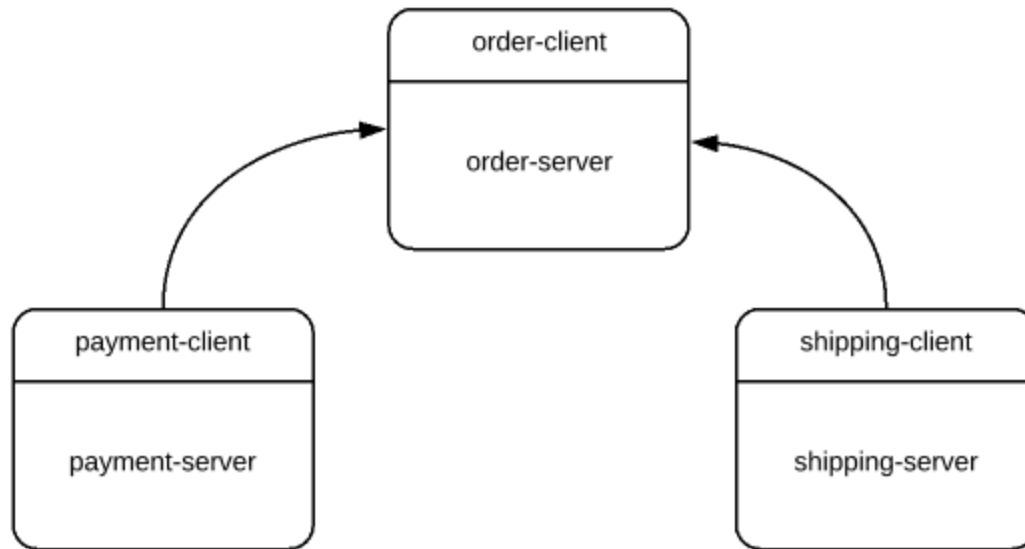
In this case, only the MicroService can share the models with other services. So we would be restructuring our project as shown here.

```
shopping-cart
  user-service
    user-client
    user-server
  order-service
    order-client
    order-server
  payment-service
    payment-client
    payment-server
  shipping-service
    shipping-client
    shipping-server
```

That is, each microservice would be a multi module maven project and would contain at-least 2 modules. **client** and **server**.

Email
Validation (1)
Framework
(104)
Functional
Test
Automation
(83)
Puppeteer
(1)
QTP (10)
Selenium
(76)
Extend
WebDriver
(11)
Ocular (2)





The client module would contain all the DTOs and the server module would contain the service, entity, controller classes. Since the controllers are ones which would be receiving the incoming requests, server module also need the DTOs. So in this case, payment-server would include payment-client as a maven dependency.

```

shopping-cart
  payment-service
    payment-client
    payment-server
  shipping-service
    shipping-client
    shipping-server
  
```

Page Object

Design (17)

Report (8)

Selenium

Grid (10)

TestNG (7)

gRPC (15)

Java (81)

Guice (2)

Reactor (41)

Jenkins (17)

Kafka (9)

Kubernetes

(8)

Linkerd (2)

Maven (7)

messaging

(11)



```
order-service
  order-client
  order-server
    mvn: order-client
    mvn: payment-client
    mvn: shipping-client
```

In the above structure, order-server module includes the below maven dependencies to get the required DTOs. So that it can get shipping-service user model from the shipping-client library and the payment-service user model from payment-client.

```
order-client
payment-client
shipping-client
```

Sample Code:

You can take a look at [this sample project](#) I have created as explained above. We have 2 services.

```
rating-service
```

MicroService
(76)
Mongo (4)
Monitoring
(13)
FileBeat (1)
Grafana (5)
InfluxDB (7)
Kibana (2)
Multi Factor
Authenticati
on (2)
nats (4)
Performance
Testing (44)
Extend
JMeter (5)
JMeter (43) ^

user-service

User wants to rate a product he has bought. So the user-service needs the DTOs for making the request to the rating-service. So it includes the rating-client library.

```
rating-service
```

```
    rating-client
```

```
    rating-server
```

```
        mvn: rating-client
```

```
user-service
```

```
    mvn: rating-client
```

Advantages:

- No code duplication.
- Specific models for specific services are used.
- Service specific validation can be included in the client library.
- Contract breaking changes can be communicated via library version easily.
- We can provide backward compatibility.

Workload

Model (2)

Little's Law

(1)

Web

Scraping (1)

Protocol

Buffers (15)

r2dbc (4)

Reactive

Programmin

g (40)

Redis (8)

rsocket (7)

Slack (3)

SMS (1)

Spring (73)

^

Happy coding 😊

Share This:

« Reactor Flux File Reading

Spring Boot CockroachDB Integration »

6 thoughts on “MicroServices – How To Share **DTO** (Data Transfer Objects)”

Spring Boot

(62)

Spring Data

(11)

Spring

WebFlux (62)

Udemy

Courses (5)

Utility (20)

WebSocket

(2)





Tegan Welton

February 26, 2022 at 10:10 AM

I am in fact pleased to read this website posts which contains lots of helpful information, thanks for providing such information.

Reply



Luis Hugo

April 7, 2022 at 2:05 PM

Awesome article, I have a question, what happen if we need our service be language agnostic? I mean, other team maybe like it implement their own project on GOLANG or RUST. So should I implement a client library for each of those languages? If that is right and we have to, so we are facing duplication of code and maintenance of it not just in one language but on many languages. I am right? What are your recommendations? Thank you

Reply



vlns



June 26, 2022 at 8:18 PM

It is a great question. I would suggest you to use protobuf for that.

<https://www.vinsguru.com/protobuf-a-simple-introduction/>

Reply



Mohan

May 1, 2023 at 12:44 PM

Fantastic explanation.. I have a doubt, What happen if we need to implement this client-library concept into our multiple working services? Will this break well functioning services if its need to tweak each of those applications?

Reply



vlns

May 2, 2023 at 12:13 AM

why do you want to change an already working service 😊 ?

Reply



Mohan

May 2, 2023 at 2:37 AM

There is a couple of services built few years ago, To the inter-microservices call they have implemented the copies of DTO's wherever required. And what happen if we need to implement this client-library concept into our multiple working services? Do we need to modify these existing services to implement client-library specification when we add a new service?

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *



Comment

Name *

Email *

Website

☐ Save my name, email, and website in this browser for the next time I comment.

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.

Post Comment



