



# Микросервисная архитектура

# **Меня хорошо видно & слышно?**



# **Защита проекта**

## **Тема: Интернет-магазин**



**Леонтьева Елена**

Программист отдела развития интеграционных систем  
ООО «СКБ-онлайн»

# План защиты

Цели проекта

Что планировалось

Используемые  
технологии

Что получилось

Схемы/архитектура

Выводы



Что было в начале, что знали до курса, сколько времени заняло выполнение проекта

# Что планировалось

Работаю в отделе интеграции, писала интеграционные приложения из одного микросервиса, которые взаимодействуют через WMQ, AMQA. Поверхностно знала технологии docker и kubernetes.

1. Научиться писать многосервисные приложения, знать, какие есть инструменты для декомпозиции приложения на микросервисы.
2. Познакомиться ближе и лучше понимать работу Docker, Kubernetes.
3. Познакомиться с разработкой API.
4. Знать область применения тех или иных архитектурных решений, уметь их использовать.



# Цели проекта

Какие цели вы поставили и  
какие задачи решили своим  
проектом

1. Разработка учебного приложения с использованием паттернов микросервисной архитектуры
2. Получить практические навыки в развертывании приложения из манифестов в Kubernetes, настройке инфраструктуры.

# Используемые технологии



LoopBack



```
    requestBody.status=STATUS.NEW;
    await this.orderRepo.create(_requestBody);
```

# Что получилось

&gt; .vscode

&gt; dist

&gt; kuber

&gt; node\_modules

&gt; public

&gt; schemes

&gt; src

&gt; \_tests\_

&gt; connectors

&gt; controllers

| TS index.ts

| TS order.controller.ts

| TS ping.controller.ts

| README.md

&gt; datasources

&gt; docs

&gt; flow

&gt; models

&gt; repositories

&gt; store

| TS application.ts

&gt; OUTLINE

&gt; TIMELINE order.controller.ts

⌚ мелкие правки Le... 2 days

🕒 File Saved 3 days

```
let payment = new SvcConnector(CONFIG.payment.host, 60000, CONFIG.trace);
let payReq = new BalanceReserve();
payReq.order_id = orderID;
payReq.user_id = _requestBody.user_id;
payReq.price = _requestBody.price;
```

```
let payRes = await payment.postReq(payReq);
```

```
console.log(payRes);
```

```
let notify = new SvcConnector(CONFIG.notify.host, 60000, CONFIG.trace);
let message = new Message();
message.order_id = orderID;
message.user_id = _requestBody.user_id;
message.date=new Date().toString();
```

```
if (payRes.error) {
    //let payDel = await payment.delReq(payReq);
    await this.orderRepo.deleteById(orderID);
    console.log("Платеж не прошёл. Заказ отменён.");
    message.message = "Оплата не прошла. Недостаточно средств на счете. Заказ отменён."
    notify.postReq(message)
    return this.response.status(payRes.code ?? 500).send(payRes.message);
}
```

```
await this.orderRepo.updateById(_requestBody.order_id, {status: STATUS.PAID});
```

```
//РЕЗЕРВ ТОВАРА
```

```
let stock = new SvcConnector(CONFIG.stock.host, 60000, CONFIG.trace);
```

# Примененные паттерны

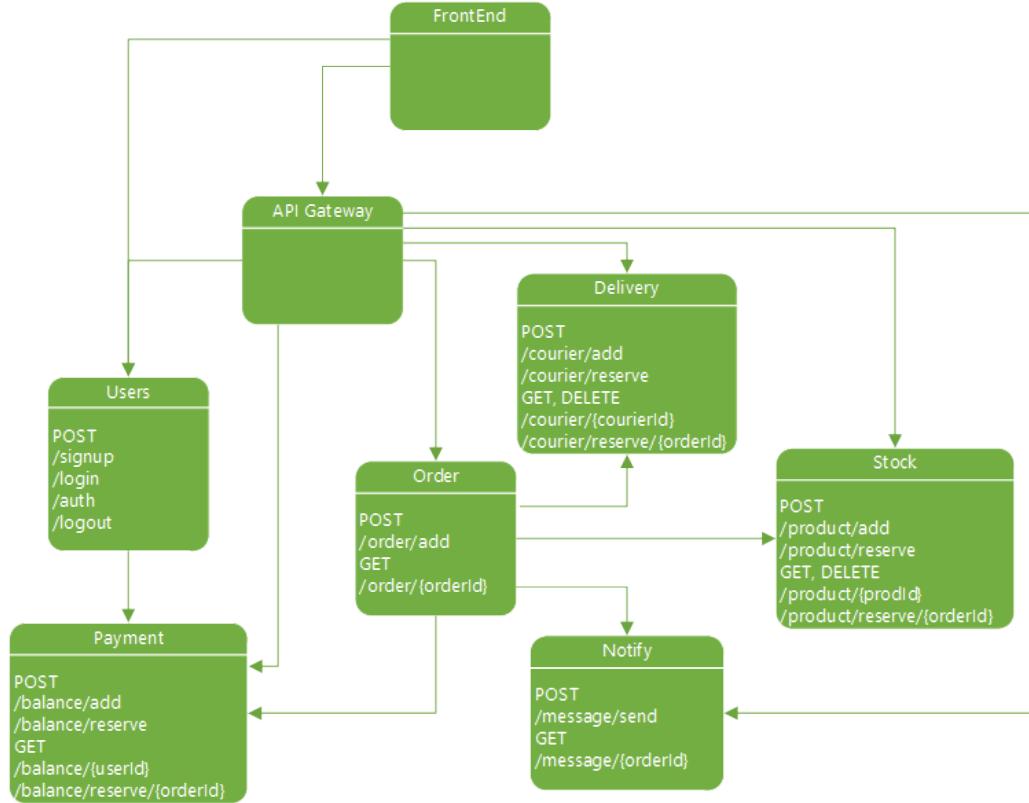
- 1.** Контейнеризация
- 2.** Server-Side Service Discovery
- 3.** Domain-Driven Design
- 4.** Database Per Service
- 5.** Сага (оркестрация)
- 6.** Health Check
- 7.** Идемпотентность



# Схема архитектуры

## Исходники

1. ProjectHelm
2. User
3. Order
4. Payment
5. Stock
6. Delivery
7. Notify



# Схема Бд

Users

Users	
PK	id
	username
	firstName
	lastName
	email
	phone
	isAdmin

Payment

Balance	
PK	user_id
	account

BalanceReserve	
PK	id
	order_id
	user_id
	operation
	price
	completed

Order

Order	
PK	order_id
	user_id
	product_id
	number
	price
	date
	status

Stock

Product	
PK	product_id
	name

ProductReserv	
PK	order_id
	product_id
	number

CourierReserv	
PK	order_id
	courier_id
	date

Delivery

Courier	
PK	courier_id
	name

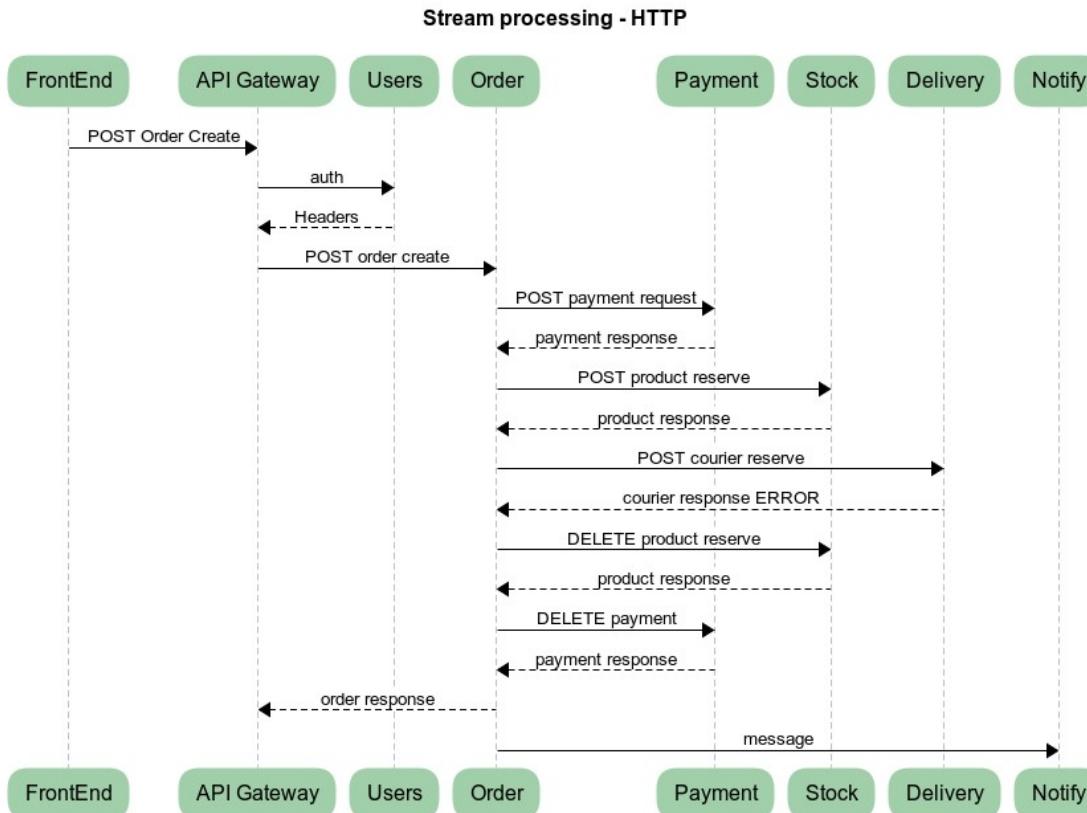
CourierReserv	
PK	order_id
	courier_id

Notify

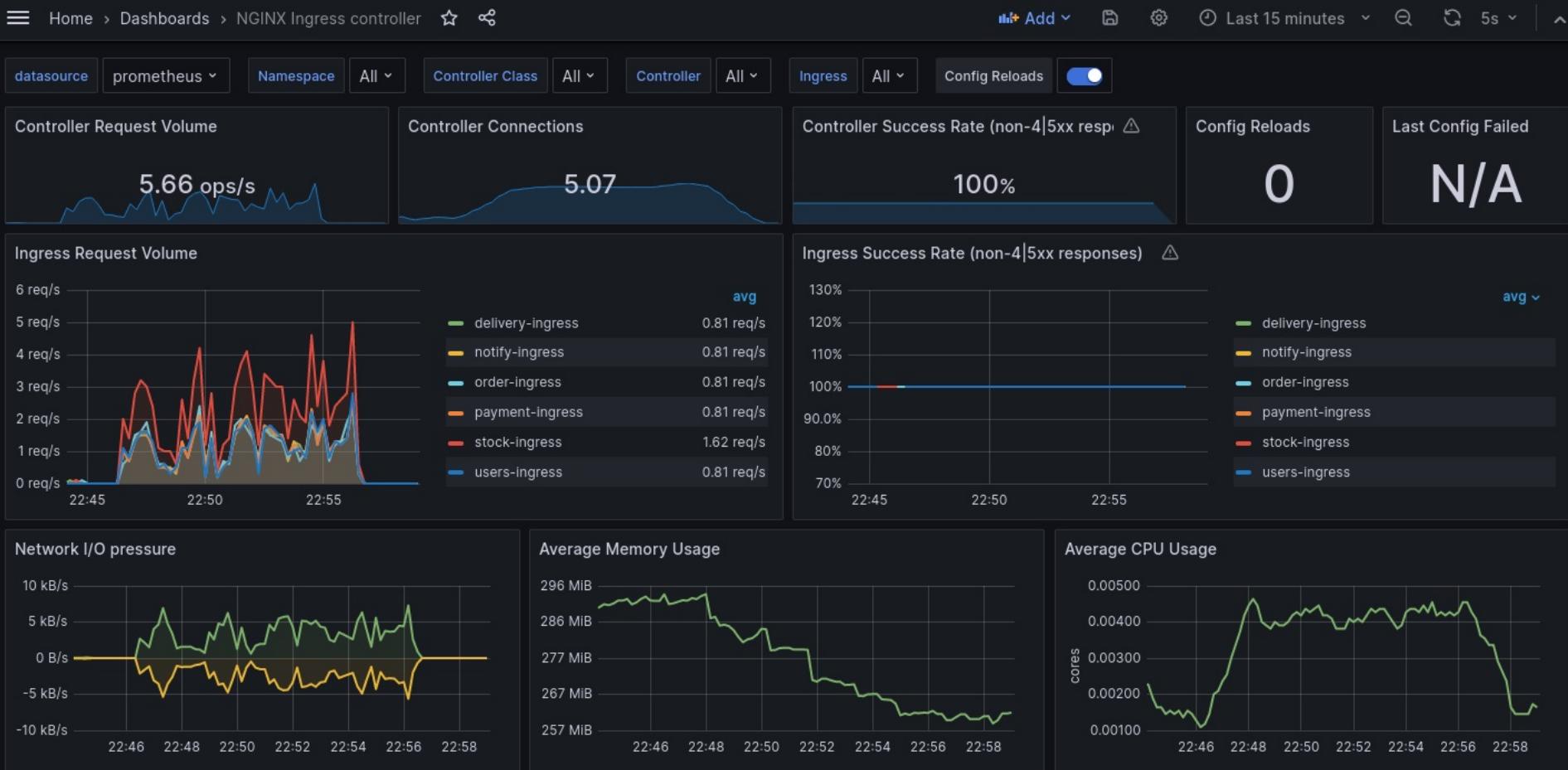
Message	
PK	id
	order_id
	user_id
	date
	message



# Схема саги



# Grafana

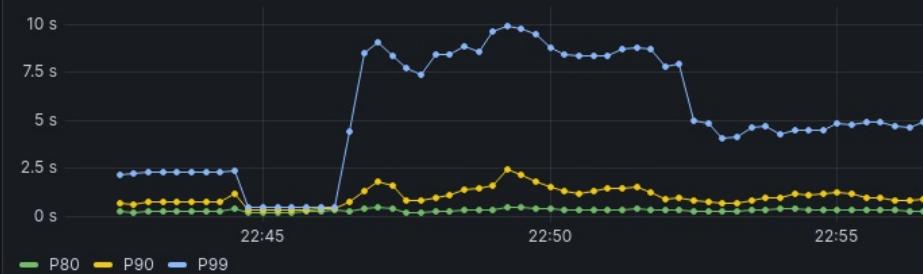


# Grafana

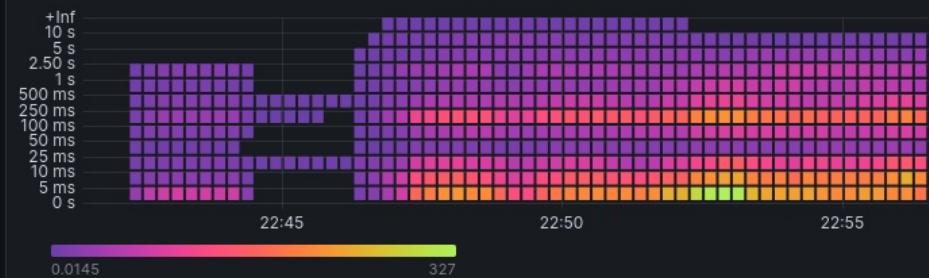
## Ingress Percentile Response Times and Transfer Rates ⓘ

Ingress ▾	P50 Latency	P90 Latency	P99 Latency	IN	OUT
users-ingress	156 milliseconds	312 milliseconds	775 milliseconds	120.30 B/s	96.30 B/s
stock-ingress	6 milliseconds	133 milliseconds	446 milliseconds	226.80 B/s	167.10 B/s
payment-ingress	232 milliseconds	1 second	7 seconds	135.60 B/s	84.00 B/s
order-ingress	907 milliseconds	4 seconds	8 seconds	191.10 B/s	104.10 B/s
notify-ingress	7 milliseconds	18 milliseconds	24 milliseconds	89.10 B/s	141.30 B/s
delivery-ingress	11 milliseconds	163 milliseconds	652 milliseconds	112.80 B/s	78.90 B/s

## Ingress Percentile Response Times



## Ingress Request Latency Heatmap



# Выводы и планы по развитию

1. Получила навыки в разработке API, которые уже применяю в работе
2. При увеличении нагрузки перейти на асинхронное взаимодействие (Kafka/MQ).
3. Логирование в ELK.
4. Кэширование товаров (сервис Product) и данных для входа (сервис Users).

Запланируйте пару минут на рефлексию в конце защиты проекта и расскажите о планах по развитию



# Спасибо за внимание!