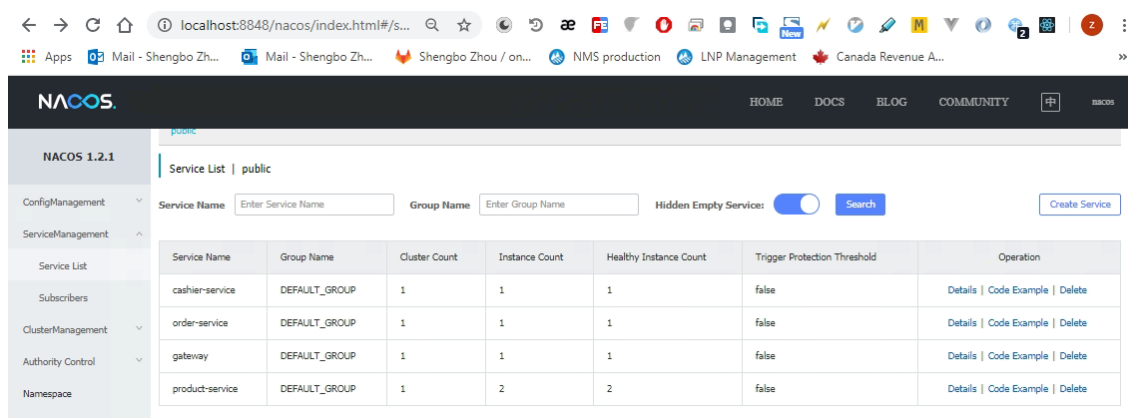
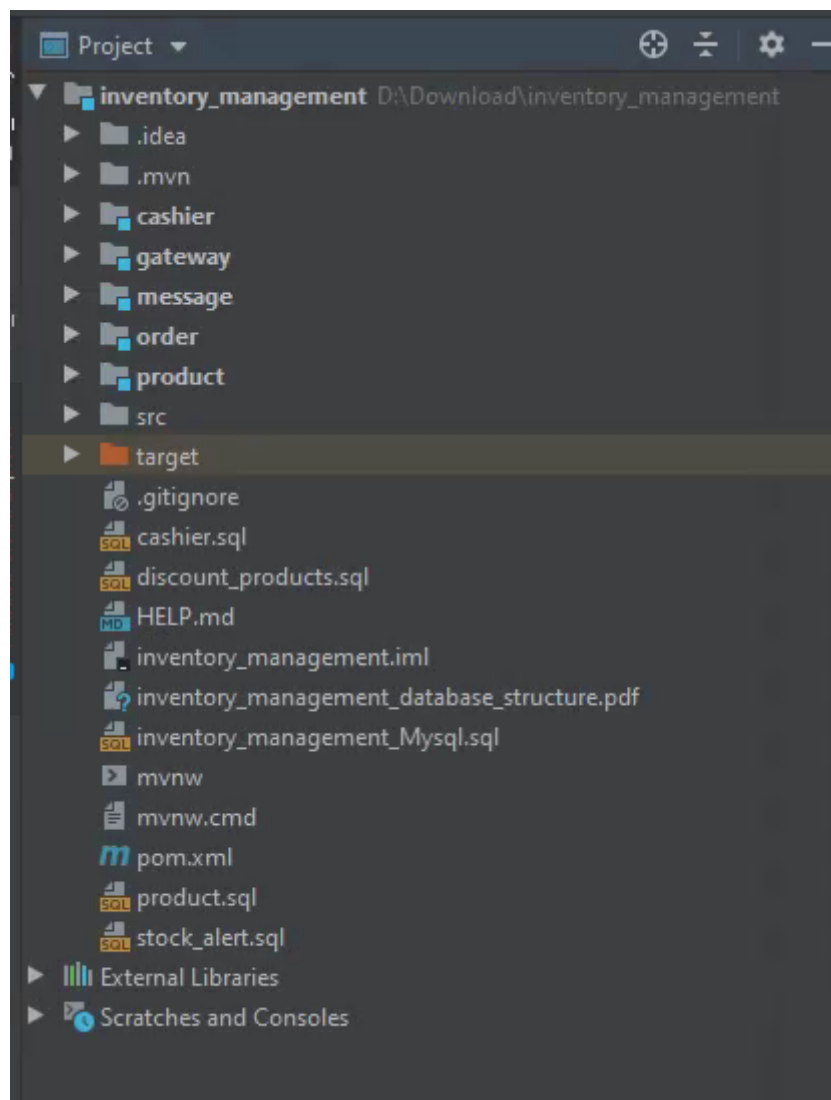


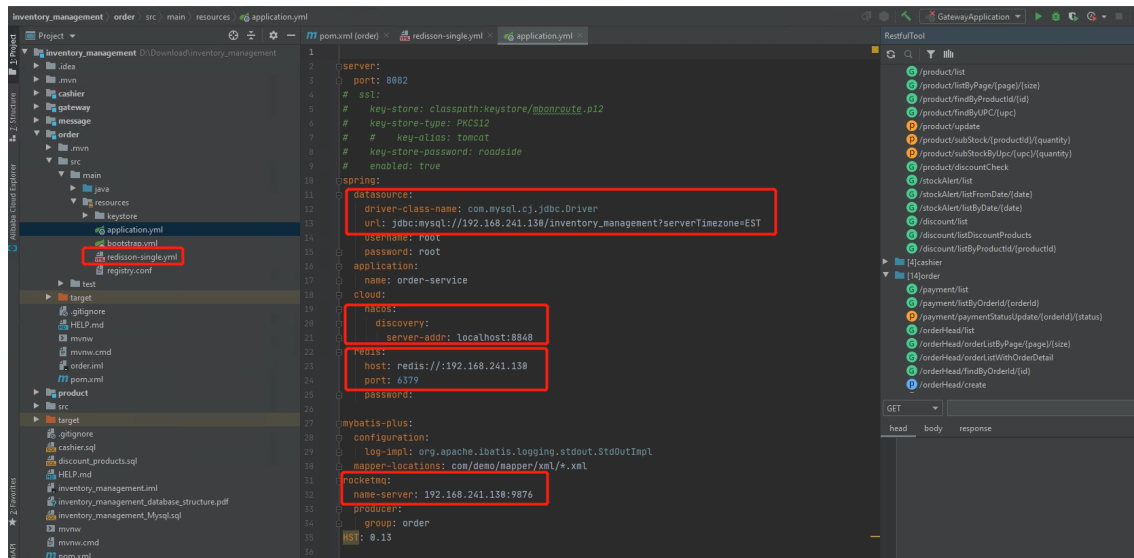
API Document

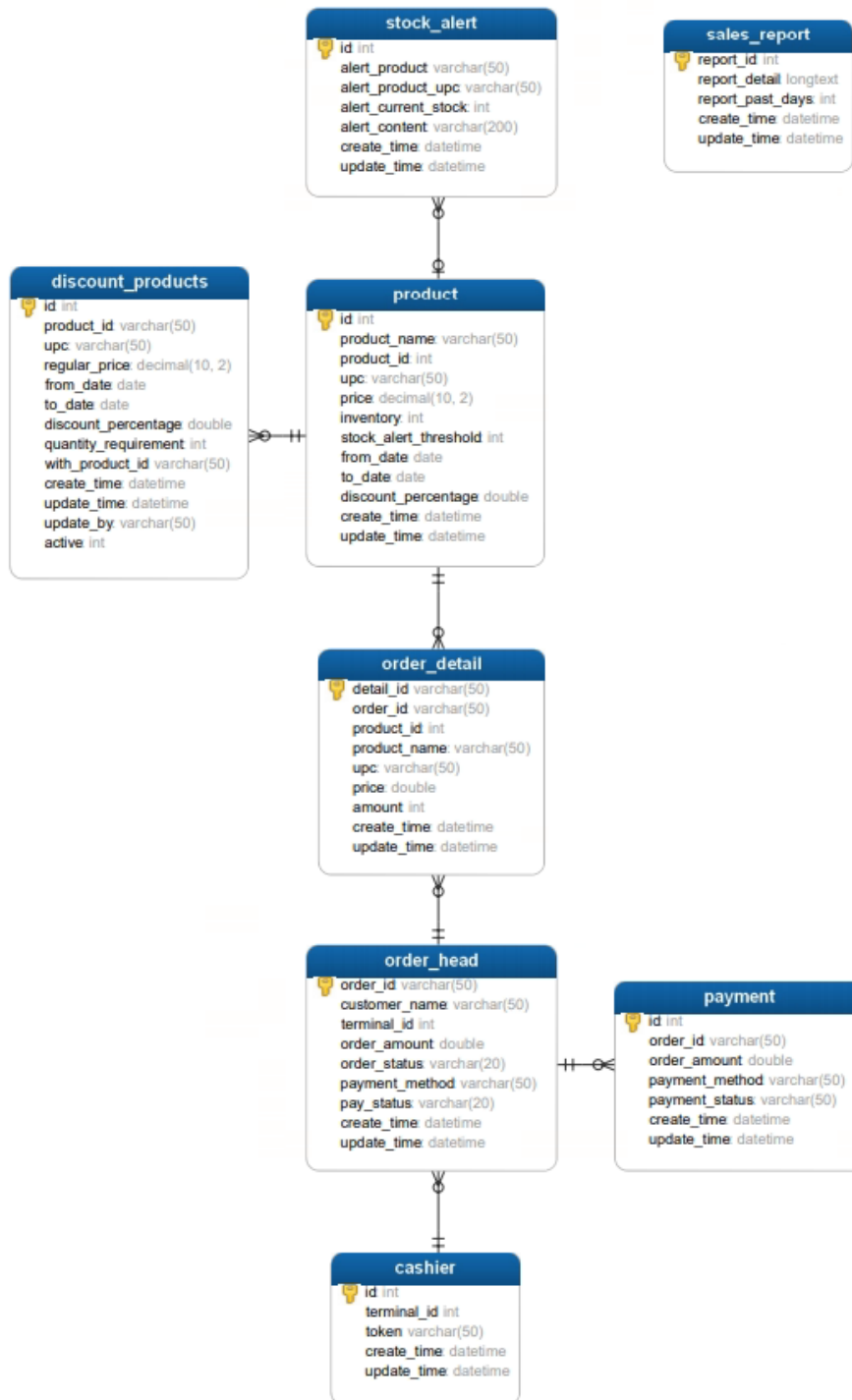
Brief introduction:

1. use Java, Spring Boot, Spring Cloud to create the microservices and RestFul APIs, Nacos to realize service registry and discovery, Gateway as Router, Feign & Ribbon as Load Balancer which can realize weighted rule, service can be scaled horizontally by deploying multiple instances



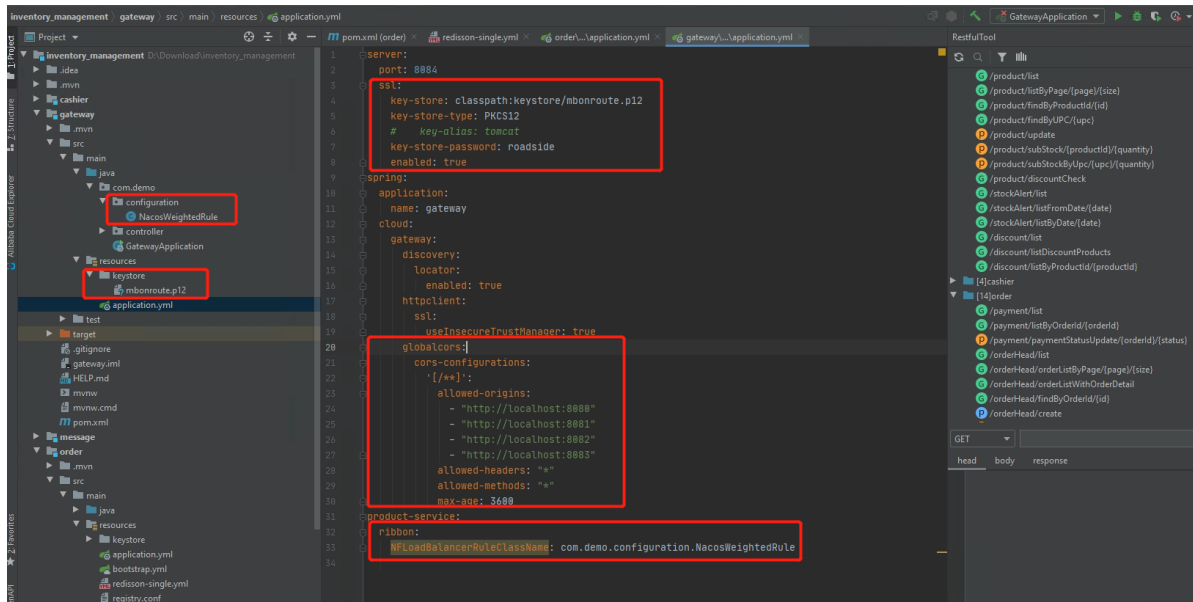
2. Mysql as the database, Redis for cache and distributed lock, Mybatis & Mybatis plus as ORM framework





3. Header in request for authentication and https certificate call are set for security

GET https://localhost:8081/product/list	
Params	Authorization
Headers (10)	
<input checked="" type="checkbox"/> Cache-Control ①	no-cache
<input checked="" type="checkbox"/> Postman-Token ①	<calculated when request is sent>
<input checked="" type="checkbox"/> Host ①	<calculated when request is sent>
<input checked="" type="checkbox"/> User-Agent ①	PostmanRuntime/7.28.4
<input checked="" type="checkbox"/> Accept ①	*/*
<input checked="" type="checkbox"/> Accept-Encoding ①	gzip, deflate, br
<input checked="" type="checkbox"/> Connection ①	keep-alive
<input checked="" type="checkbox"/> terminalId	101
<input checked="" type="checkbox"/> token	token101
Key	Value

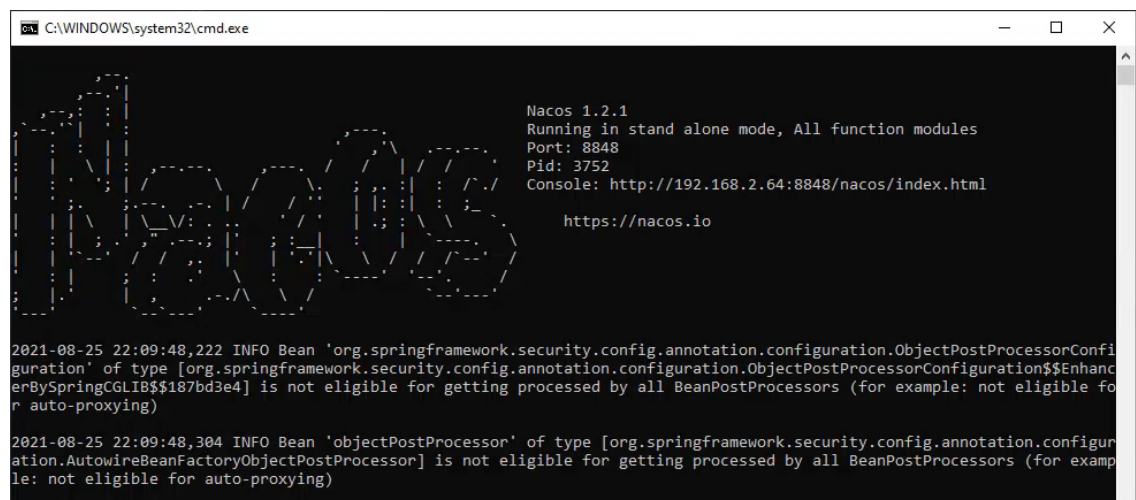


Project starting process:

1. First run the inventory_management_Mysql.sql to set the database for the application
2. Second need to start Nacos for the microservices registry and discovery

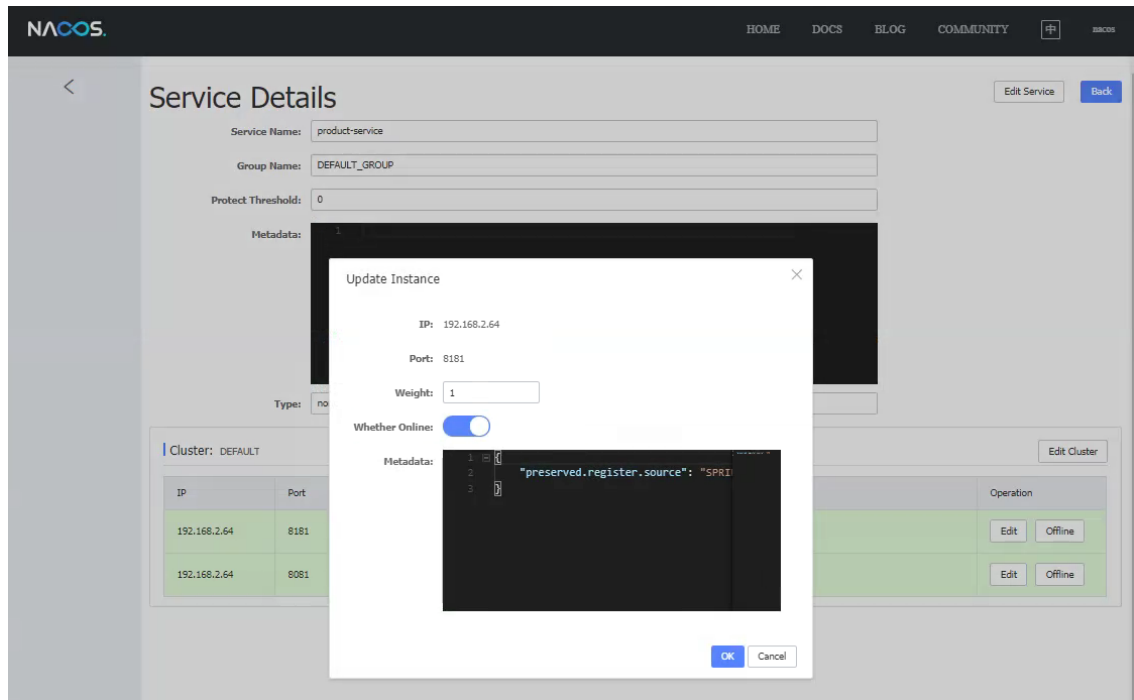
This PC > Jean (D:) > nacos > bin >

Name	Date modified	Type	Size
logs	2021/9/19 12:25 AM	File folder	
work	2021/9/19 12:25 AM	File folder	
derby.log	2021/8/25 10:09 PM	Text Document	1 KB
shutdown.cmd	2020/4/8 2:23 PM	Windows Command Sc...	1 KB
shutdown.sh	2020/4/8 2:23 PM	Shell Script	1 KB
startup.cmd	2020/4/8 2:23 PM	Windows Command Sc...	3 KB
startup.sh	2020/4/8 2:23 PM	Shell Script	5 KB



3. then can start gateway, product, order, cashier microservices, each service has to set the application name in application.yml so it can be discovered and registered into Nacos, the gateway can set Https certificate and the Cors configuration so no need for each service to set separately.

4. the same service can start several instances and set different weight in Nacos service configuration



5. go on testing whether all apis working fine or not

Summary of task completion from the assignment:

1. Considerations

- Payment must be made in full before an order can be closed.

before closing the order, application will first check payment status, once payment in full, will set the order status to completed, otherwise will send "payment not paid" response to the front end

- Cashier should not be able to add products to an order without enough inventory.

application will judge whether stock is enough or not for any product in the order, if not will throw RuntimeException and log stock not enough.

- Error handling
- Multiple instances of the service may be deployed to balance the load horizontally.

Using Nacos and Gateway, microservice can deploy many instances to scale horizontally and load balanced.

- The service maybe queried frequently during rush hours. It might not be a good idea to run all tasks synchronously on API calls.

Need to import MQ to lower the coupling and improve service efficiency

- Sales Report

```
sales statistics of past xx days
https://localhost:8082/orderDetail/salesStatistic/7
```

- Recall

```
Recall check by UPC
https://localhost:8082/orderHead/recallCheck/SP104
```

- Inventory Alert

when stock is lower than threshold, will create an alert record and save into database, in the future can use MQ and web socket to realize sending message and remind in the admin console of inventory management

- Security

use https and header authentication setting in request to realize simple authentication, and also need to encryp token for the front end when sending request

updates and to do list

Since time limitation, some functions are not fully realized and need to be improved.

Looking forward an opportunity to discussing with you for more details.

Main updates:

1. Use ResponseEntity to regulate the http response in Controller which would make the code more clear and regular.
2. Spilt the original one database to different databases, like Product, Order, Cashier, for different microservices, as a distributed project we should use different database for different microservice.
3. Import Spring Cloud Alibaba Seata solution tool to realize the distributed transactional business for the distributed project.
4. Import MQ and create another message service to send and receive the message, which could reduce the coupling and improve efficiency, the coming function I want to realize is using the webSocket to realize when placing order successfully, MQ will send the message actively and automatically to the front end from backend, which will tell the admin to handle the order in time.
5. Use @RestControllerAdvice and @ExceptionHandler to handle the exceptions globally, handle the self-defined exception like Product, Order exception, RuntimeException and the general Exception which will make it more friendly to the client.
6. create a simple front end admin console for this project, so from the website we can call our backend api and handle the real business

Inventory Management Console								admin
ProductModule	HomePage	OrderManagement						
ProductManagement								
OrderModule								
OrderManagement								
Order Id	Customer Name	Terminal Id	Total Price	Order Status	Payment Status	create Time	Operations	
8dce1b6f9b937427bc402124b19a288	Ben	101	65.53999999999999	pending	pending	2021-08-27T15:07:35	Cancel Finish	
7fce810b25ec2a0d9bcc98a5039cb0d9	Ben	101	65.53999999999999	pending	pending	2021-08-27T15:04:47	Cancel Finish	
ab3043b0b61b08be4f8826d9cb9abe5	Ben	101	65.53999999999999	pending	pending	2021-08-27T15:01:47	Cancel Finish	
c3aa237722864f845191b7f5c5b38	Ben	101	65.53999999999999	pending	pending	2021-08-27T14:59:31	Cancel Finish	
9e08b7bea7fa32d12c0e79e557744a92	Michael	101	243.515	completed	paid	2021-08-27T05:15:53	Cancel Finish	

7. use Docker-compose to deploy the microservices (not finished yet, since in the docker environment some mysql and Redis setting error, I still need to find the main cause)
8. Use @Scheduled(cron="0 0 12 * * ?") to realize some scheduled actions like sales statistic and stock alert

To do list:

1. Database structure design is not quite reasonable and still needs some optimization about the relations according to the real business logic which should discuss with BA and DBA for details.
2. Distributed lock and pressure test: As a distributed project for multi threads and high concurrency scenarios, the application should set Distributional Lock in the application, I put the Redisson Lock into application, start 100 threads and 3 application instances, and implemented the pressure test using Jmeter, the product subtract stock function seems working fine. Still needs code optimization and more pressure test.

HTTP Request.jmx (D:\Software\apache-jmeter-5.4.1\apache-jmeter-5.4.1\bin\HTTP Request.jmx) - Apache JMeter (5.4.1)													
Aggregate Report													
Name: Aggregate Report													
Comments:													
Write results to file / Read from file													
Filename: <input type="text"/> <input type="button" value="Browse..."/> <input type="checkbox"/> Log/Display Only <input type="checkbox"/> Errors <input type="checkbox"/> Successes <input type="button" value="Configure"/>													
Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec	
HTTP Request	500	12	8	29	30	42	6	102	0.00%	50.2/sec	14.05	13.28	
TOTAL	500	12	8	29	30	42	6	102	0.00%	50.2/sec	14.05	13.28	
<input type="checkbox"/> Include group name in label <input type="button" value="Save Table Data"/> <input checked="" type="checkbox"/> Save Table Header													

Inventory Management Console													
ProductModule	HomePage	OrderManagement											
ProductManagement													
OrderModule													
OrderManagement													
Order Id	Customer Name	Terminal Id	Total Price	Order Status	Payment Status	create Time	Operations						
8dce1b6f9b937427bc402124b19a288	Ben	101	65.53999999999999	pending	pending	2021-08-27T15:07:35	Cancel Finish						
7fce810b25ec2a0d9bcc98a5039cb0d9	Ben	101	65.53999999999999	pending	pending	2021-08-27T15:04:47	Cancel Finish						
ab3043b0b61b08be4f8826d9cb9abe5	Ben	101	65.53999999999999	pending	pending	2021-08-27T15:01:47	Cancel Finish						
c3aa237722864f845191b7f5c5b38	Ben	101	65.53999999999999	pending	pending	2021-08-27T14:59:31	Cancel Finish						
9e08b7bea7fa32d12c0e79e557744a92	Michael	101	243.515	completed	paid	2021-08-27T05:15:53	Cancel Finish						

3. CI/CD Automation: use Gitlab, Jenkins, Docker to realize CI/CD automation process, use Kubernetes for automating deployment, scaling, and management of applications.

Stay safe and take care.

Sincerely,

Tony Zhou