## 任务8：Identtityserver4登录中心

项目(IdentityServerSample)



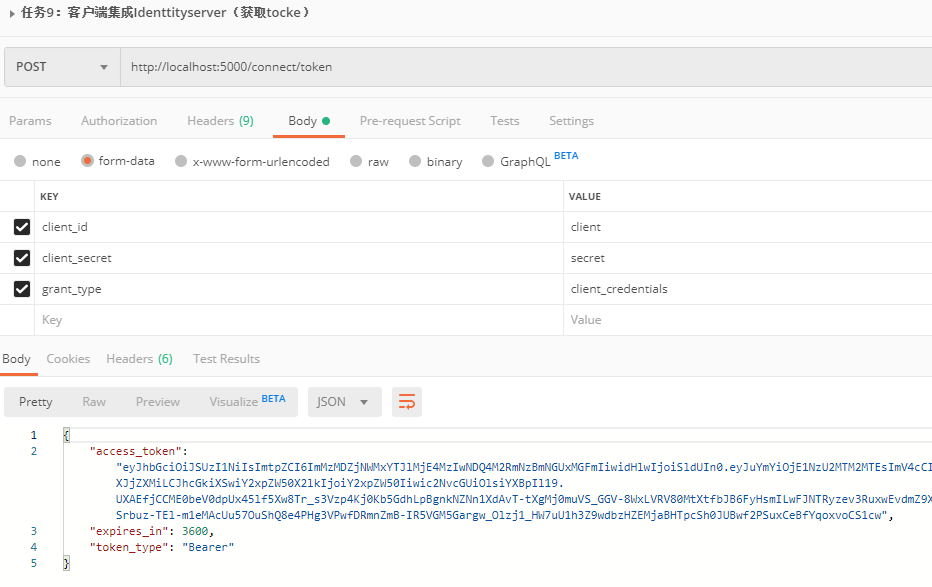
## 任务9：客户端集成Identtityserver

项目（ClientCredentialApi）

先在服务端拿到tocke，再去请求客户端

通过地址<http://localhost:5000/.well-known/openid-configuration>得到获取tocke的请求地址

需要开启服务端先拿到tocke



## 任务10：第三方ClientCredential模式调用

项目（ThirdPartyDemo）

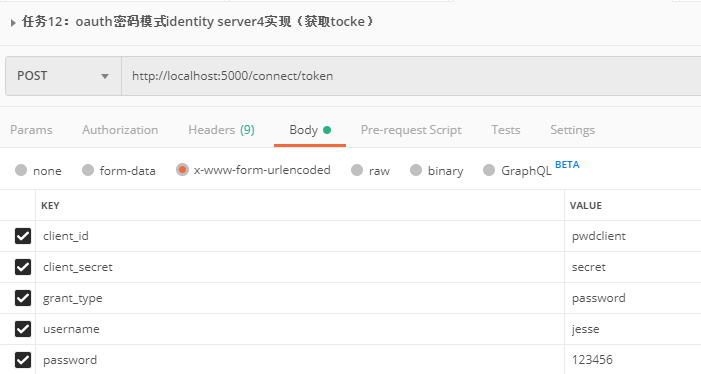
第三方应用去请求api，先要去服务端拿到tocke，再去请求客户端api，需要带上tocke

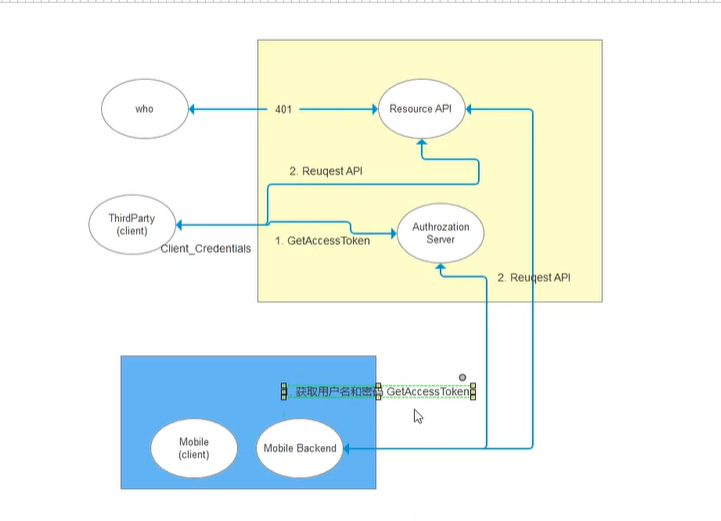
## 任务11：ClientCredential模式总结

去访问api，先要去Identtityserver4中心拿到tocke

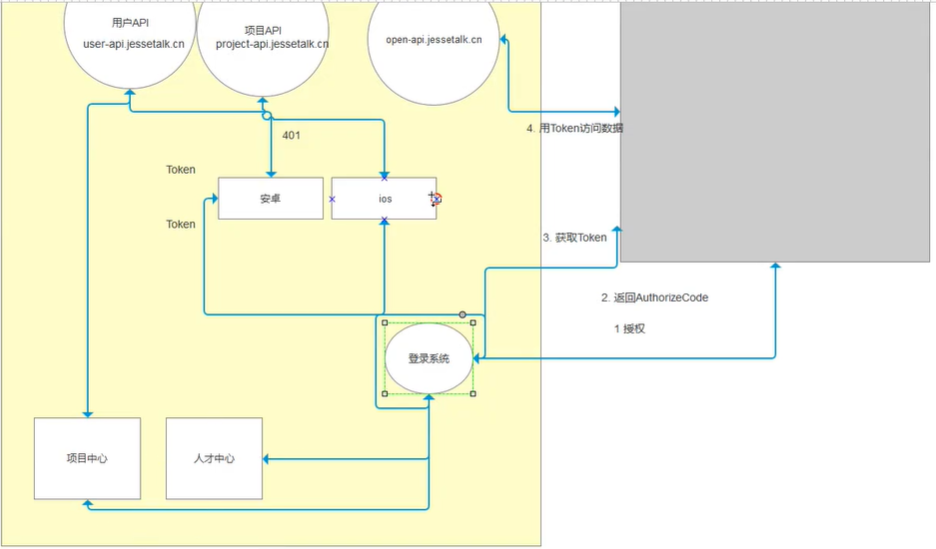
## 任务12：oauth密码模式identity server4实现

项目（IdentityServerSample2，ThirdPartyDemo2）

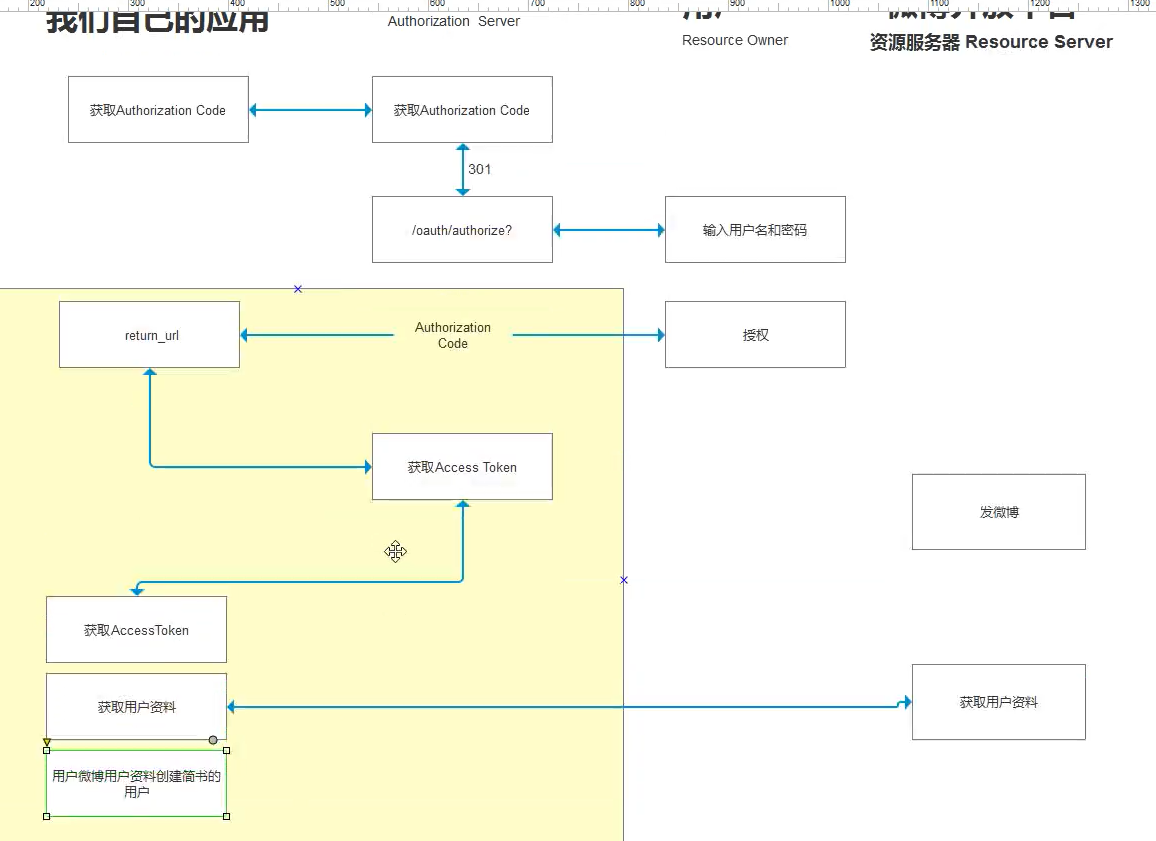


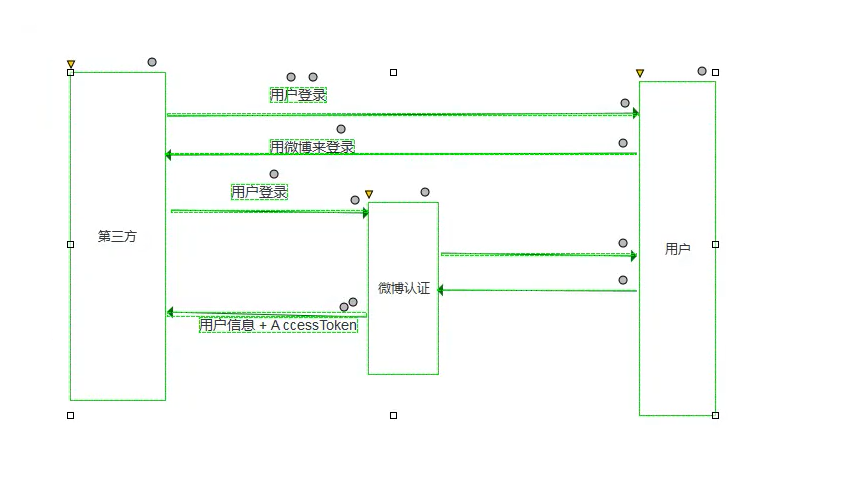


## 任务13：详解oauth2授权码流程



## 任务14：oauth2与open id connect 对比





## 任务15：oauth2 + oidc 实现 server部分

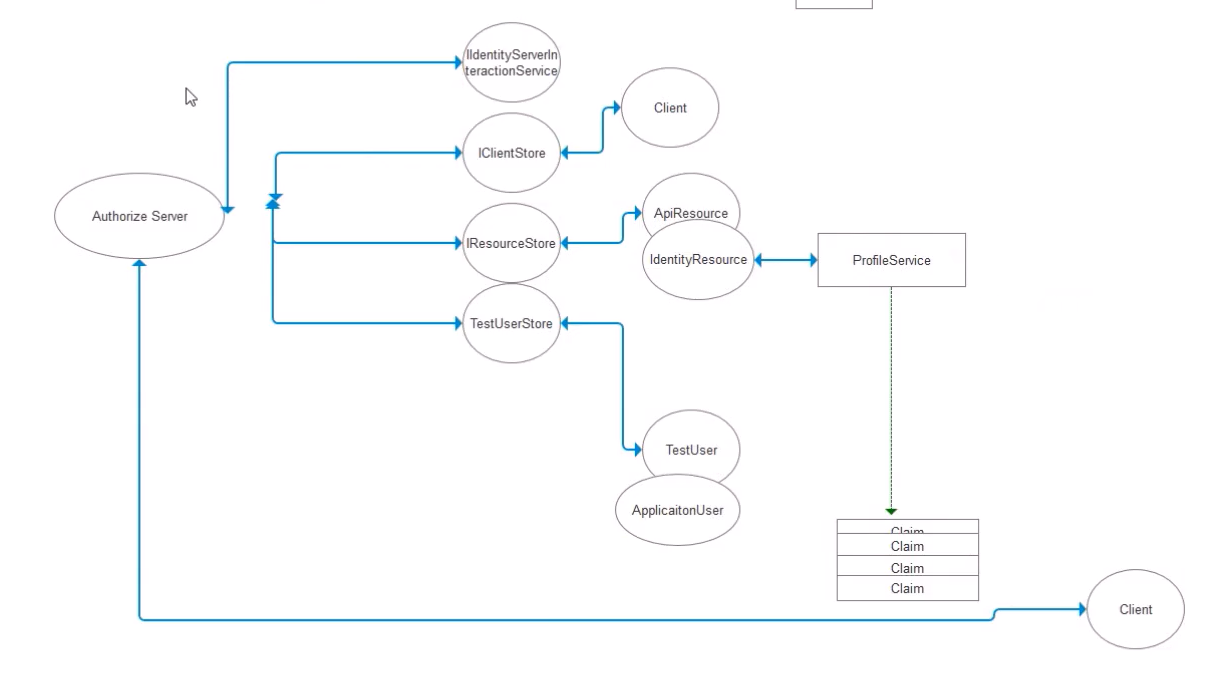
项目（MvcCookieAuthSample3）

## 任务16：oauth2 + oidc 实现 client部分

项目（MvcClient）

应用自己比较信用的应用

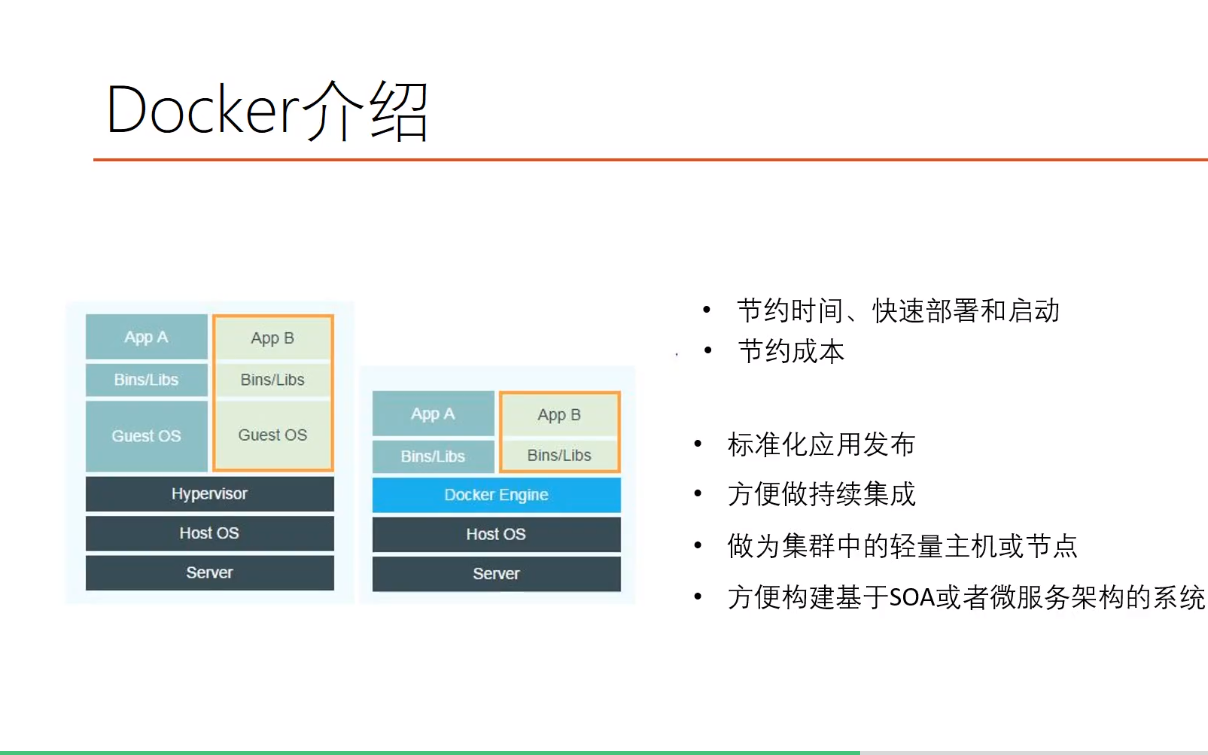
## 任务17：Identity Server 4回顾



## 任务18：Consent 实现思路介绍

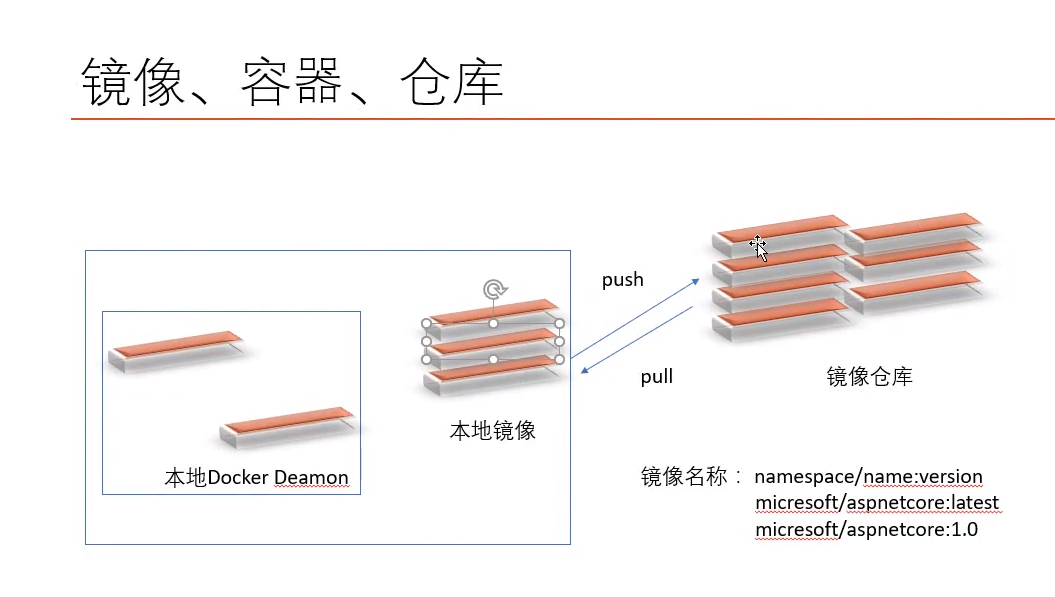
## 任务19：Consent视图制作

## 任务28：Docker介绍与目录

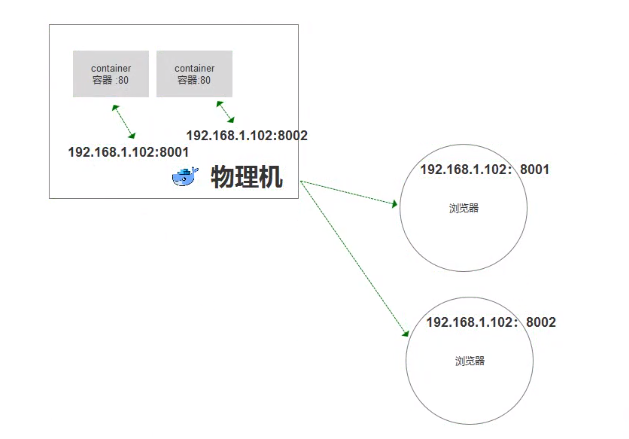




## 任务29：镜像容器与仓库



## 任务30：构建单机多容器环境-故障





## 任务32：Docker安装MongoDb

查看使用的端口：netstat

查看端口是否被使用：netstat -aon|findstr “27016”

删除容器：docker rn mymongo

创建mongo：docker run -d -p 27016:27016 –name mymongo mongo

查看本地镜像 ：docker ps -a

查看运行的镜像 docker ps

## 任务33：Docker安装Mysql及用户配置

Mysql 镜像拉取：docker pull mysql/mysql-server

安装：docker run -d -p 3306:3306 –name mysql101 mysql/mysql-server

## 34:Docker安装Mysql参数及环境变量使用

## 35:Docker安装Mysql挂载Host Volume(正式环境)

## 36:Docker安装MongoDB挂载Host Volume(正式环境)

## 37:Docker制作dotnet core控制台程序镜像(完成)

重点：如果运行不了容器，改下版本号

镜像打包

搜索dcker build 命令

<https://www.runoob.com/docker/docker-build-command.html>

打包命令

docker build -t runoob/ubuntu:v1 .

一些资料

Docker中文文档 [https://yeasy.gitbooks.io/docker\_practice/content/](https://yeasy.gitbooks.io/docker_practice/content/" \t "http://video.jessetalk.cn/course/5/thread/_blank)

官方中文文档[https://docs.docker-cn.com/](https://docs.docker-cn.com/" \t "http://video.jessetalk.cn/course/5/thread/_blank)

开发阶段用这个

Microsoft/dotnet:sdk

发布阶段用这个(打包比较下)

Microsoft/dotnet:<version>-runtime

### 开发的时候打包的方式

Dockerfile 文件说明

FROM microsoft/dotnet --使用哪种方式打包

WORKDIR /app --指定一个工作目录，没有的话就创建

COPY . /app --把所有的文件拷贝过去

CMD ["dotnet","run"] --实行命令

运行容器

Docker run --name console-dev lcc/console:v1

### 生产打包

优化的打包镜像

FROM microsoft/dotnet:sdk AS build-env

WORKDIR /code

COPY \*.csproj /code

RUN dotnet restore

COPY . /code

RUN dotnet publish -c Release -o out

FROM microsoft/dotnet:runtime

WORKDIR /app

COPY --from=build-env /code/out /app

ENTRYPOINT ["dotnet","ConsoleApp1.dll"]

回到跟目录

Cd ./

## 任务38：Mysql EF Core 快速构建 web api

--安装mysql

docker run -d -p 3306:3306 -e MYSQL\_USER="jesse" -e MYSQL\_PASSWORD="123456" -e MYSQL\_ROOT\_PASSWORD="123456" --name mysql02 mysql:5.6 --character-set-server=utf8 --collation-server=utf8\_general\_ci

--mysql命令

docker exec -it mysql01 bash

连接：mysql -ujesse -p

pwd123456

ALTER USER 'root'@' localhost' IDENTIFIED BY '123456' 修改密码

GRANT ALL PRIVILEGES ON \*.\* TO 'jesse'@'%' WITH GRANT OPTION; 给权限

退出：exit

## 任务39：ASPNETCORE WEB API与MYSQL互联(构建镜像:不被推荐)

构建镜像

docker run -t jesse/aspnetcore:prod .

运行容器

docker run -d -p 8002:80 --name aspnetcore --link mysql02:db jesse/aspnetcore:prod

docker exec -it aspnetcore bash



查看端口



Docker inspect mysql02

## 任务40：Docker Network（构建镜像运行）

Bridge(单机)



1 Docker network create -d bridge mybridge

Docker network ls



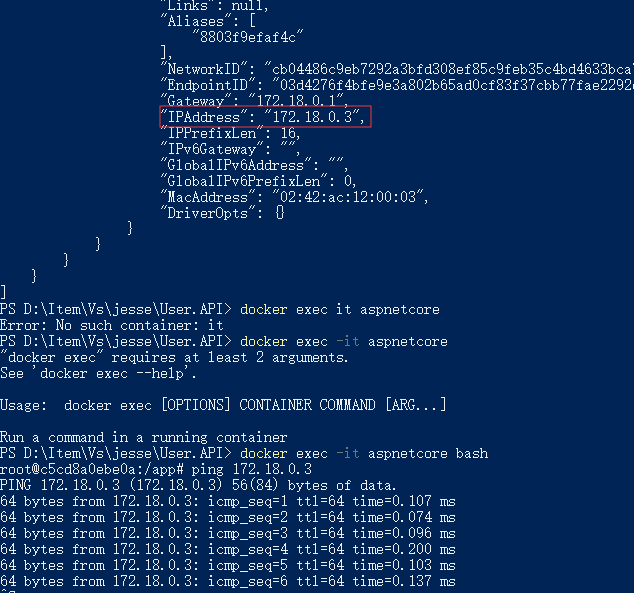
2 docker run -d -p 8002:80 --net mybridge --name aspnetcore jesse/aspnetcore:prod



docker network connect mybridge mysql02

docker exec -it aspnetcore bash

ping 下端口



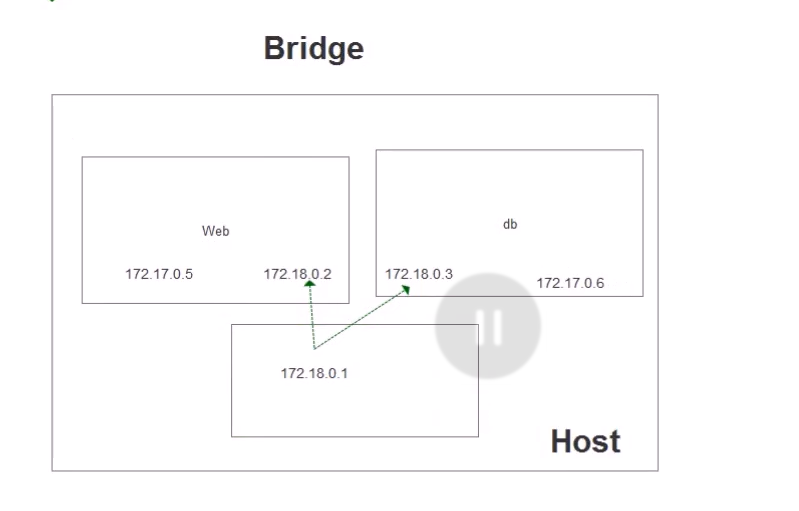
3 docker rename mysql02 db

第二种方式

改配置文件

第二种方式

挂载资料卷（推荐）



Overlay

## 任务41：制作 docker compose

制作镜像

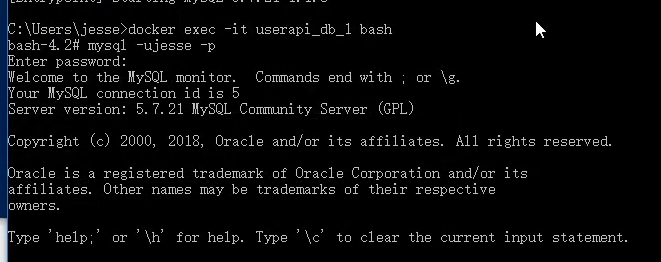
如制作不了，删除其他镜像

Docker-compose down

docker-compose build

docker-compose up

是否能登录成功





## 任务42：docker compose 启动容器(重点)

1 docker-compose up

重启

Docker-compose restart

Docker network ls

Docker inspect mysql02

给用户赋予权限

GRANT ALL PRIVILEGES ON \*.\* TO 'jesse'@'%' WITH GRANT OPTION;

先要把数据库创建好，需要给mysql用户赋予权限

## 任务43：启动问题解决方式

构造数据库的时候延时下

## 任务44：Gitlab安装

<https://blog.csdn.net/MonoBehaviour/article/details/84852984>

centos安装

链接

https://www.cnblogs.com/wyt007/p/11147233.html



Windows docker 安装

# docker pull gitlab/gitlab-ce:latest

# 官方版本是：gitlab/gitlab-ce:latest，为了提升速度我们这里使用阿里云的仓库

docker pull registry.cn-hangzhou.aliyuncs.com/imooc/gitlab-ce:latest

docker tag registry.cn-hangzhou.aliyuncs.com/imooc/gitlab-ce gitlab/gitlab-ce

docker run --detach --hostname localhost --publish 443:443 --publish 80:80 --publish 22:22 --name gitlab --restart always gitlab/gitlab-ce:latest

--查看安装日志

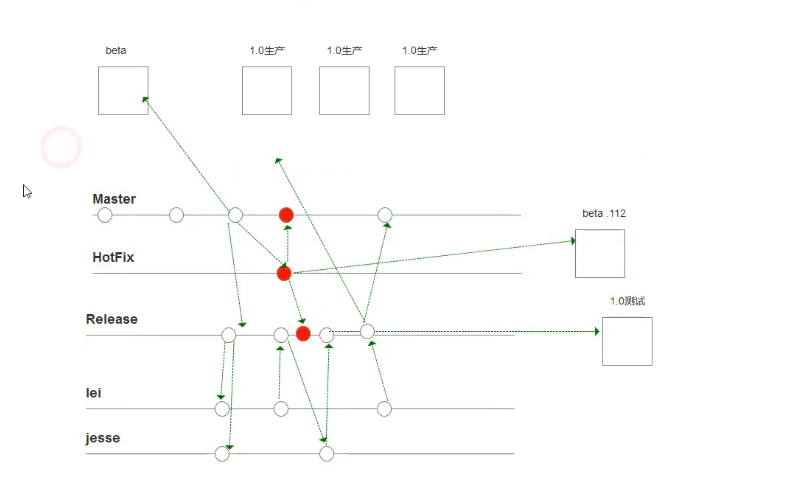
docker logs --follow gitlab

默认账号：root

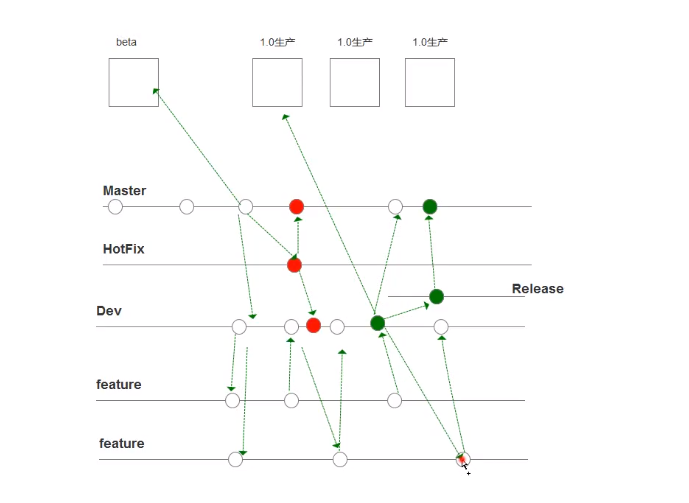
密码：chao411123

## 任务45：GitFlow的三种流程

一个团队的时候



多个团队时候



Feature 代表的是功能分支

## 任务46：GitLab Runner注册和安装

GitLab CI(持续集成)

下载安装

<https://docs.gitlab.com/runner/install/windows.html>

命令

gitlab.exe register

<http://localhost/>

bBD\_xoYxeSwnrP5Syimc

lacal-win

Shell

## 47:Gitlab-CI Build Docker-compose自动发布

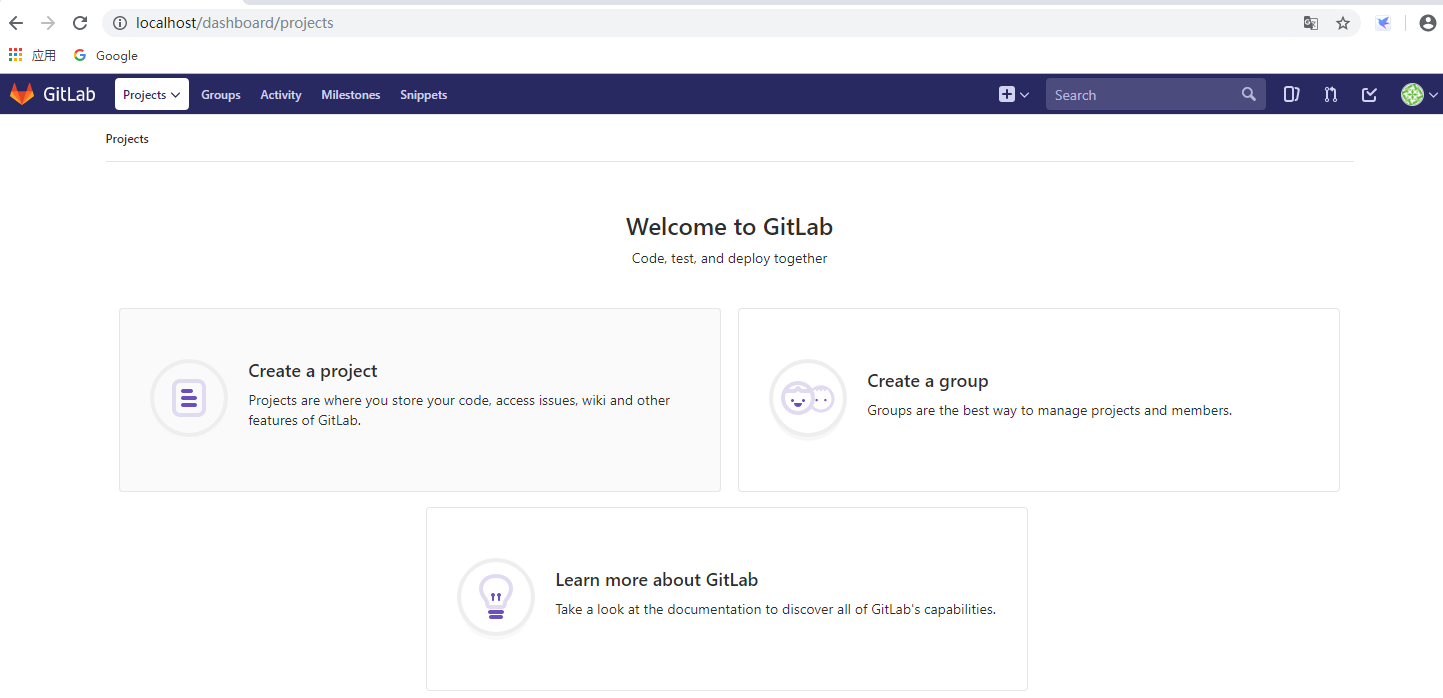
1：先注册

2：拿到SSH Keys

先看本地是否存在路径（C:\Users\Administrator\.ssh）

如没有SSH,则生成 生成SSH KEY

新建项目：



上传代码：



编辑gitlab命令



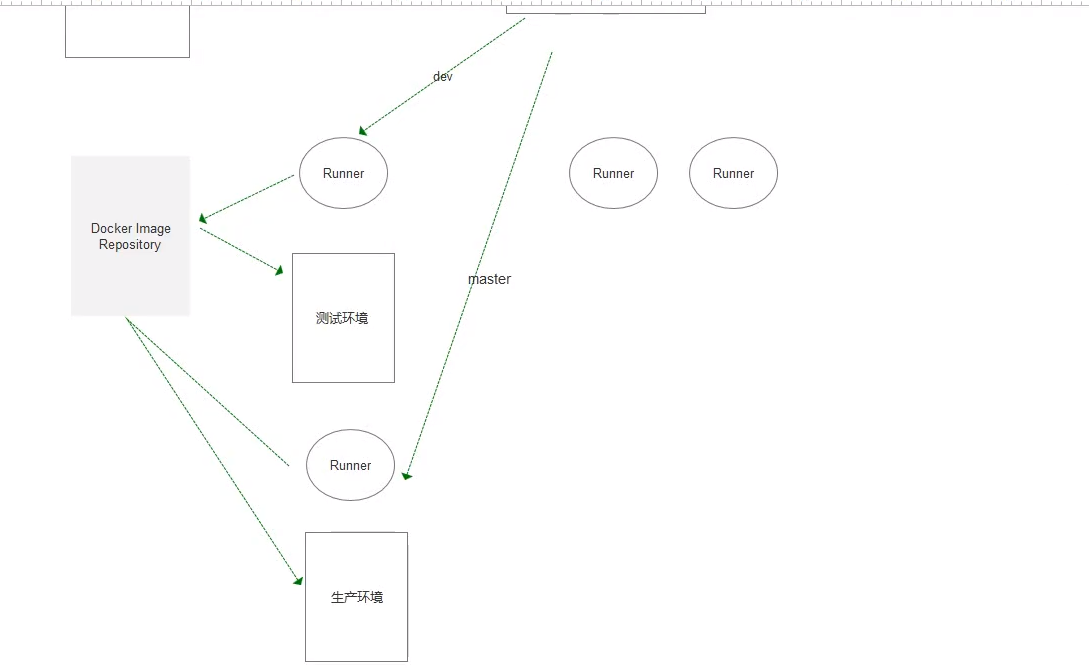
## 48：Giitlab-CI yml语法说明以及第二章总结

没有操作，一些解释说明

## 49:Docker仓库介绍以及阿里云镜像仓库的使用（需要重新看）

进入阿里云，搜索容器服务，创建镜像





## 50: 本地部署Registry

1、



2、



3、



4、(移除本地的)



5、



6、（重新拉取下来）



可视化工具（不推荐用这个），使用这个（Port）

7、

docker run -d --name registry-web --link registry:registry -e ENV\_DOCKER\_REGISTRY\_HOST=registry -e ENV\_DOCKER\_REGISTRY\_PORT=5000 -p 8080:80 konradkleine/docker-registry-frontend:v2

## 51:Registry公网https带认证部署

在外网上部署

1. 需要申请证书（腾讯云免费的），会自动绑定https

docker run -d \

--restart=always \

--name registry \

-v `pwd`/cert:/certs \

-e REGISTRY\_HTTP\_ADDR=0.0.0.0:443 \

-e REGISTRY\_HTTP\_TLS\_CERTIFICATE=/certs/1\_hub.jessetalk.cn\_bundle.crt \

-e REGISTRY\_HTTP\_TLS\_KEY=/certs/2\_hub.jessetalk.cn.key \

-p 443:443 \

registry:2

1. 创建密码

mkdir auth

docker run \

--entrypoint htpasswd \

registry:2 -Bbn testuser testpassword > auth/htpasswd

3 Base Authentication 用户名和密码

docker run -d \

--restart=always \

--name registry \

-v `pwd`/auth:/auth \

-e "REGISTRY\_AUTH=htpasswd" \

-e "REGISTRY\_AUTH\_HTPASSWD\_REALM=Registry Realm" \

-e "REGISTRY\_AUTH\_HTPASSWD\_PATH=/auth/htpasswd" \

-v `pwd`/cert:/certs \

-e REGISTRY\_HTTP\_ADDR=0.0.0.0:443 \

-e REGISTRY\_HTTP\_TLS\_CERTIFICATE=/certs/1\_hub.jessetalk.cn\_bundle.crt \

-e REGISTRY\_HTTP\_TLS\_KEY=/certs/2\_hub.jessetalk.cn.key \

-p 443:443 \

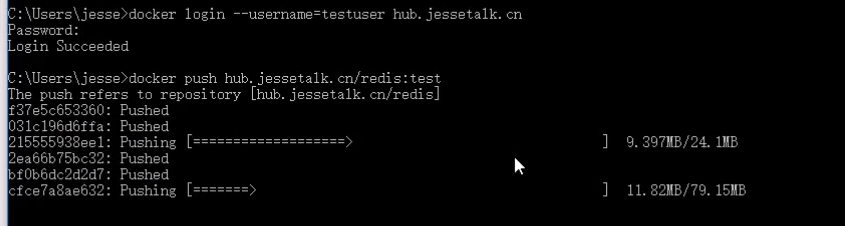
registry:2

3、



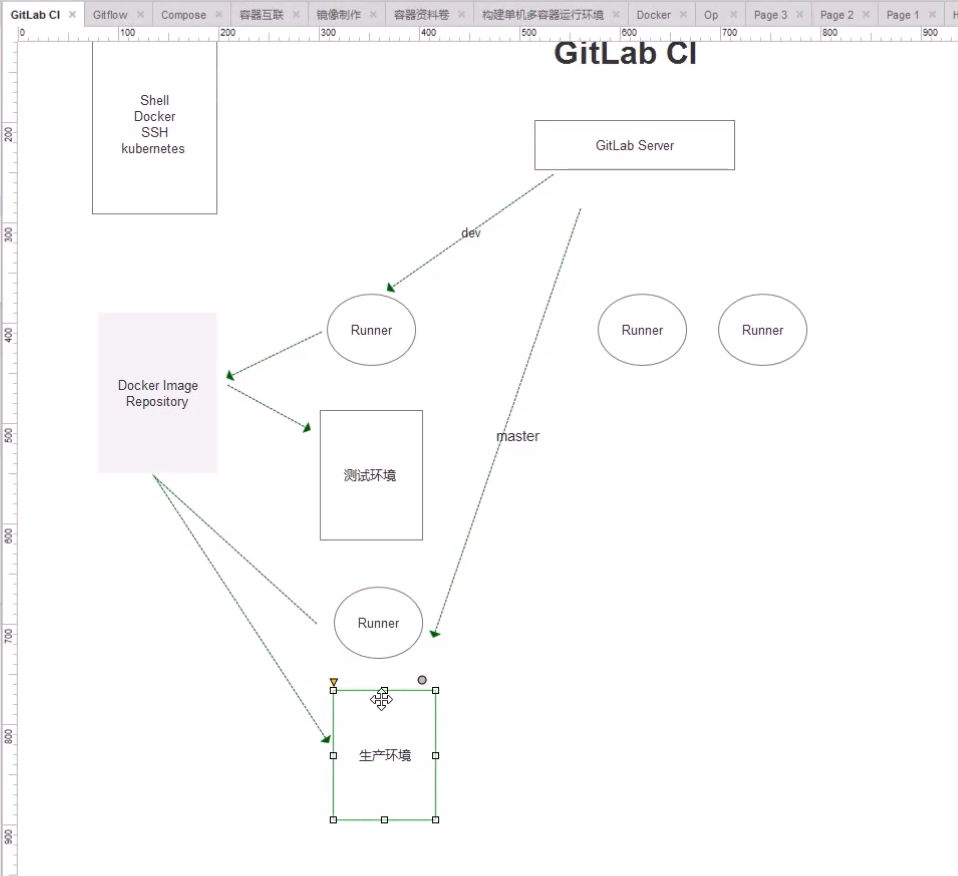
4、

需要输入密码才能推上去



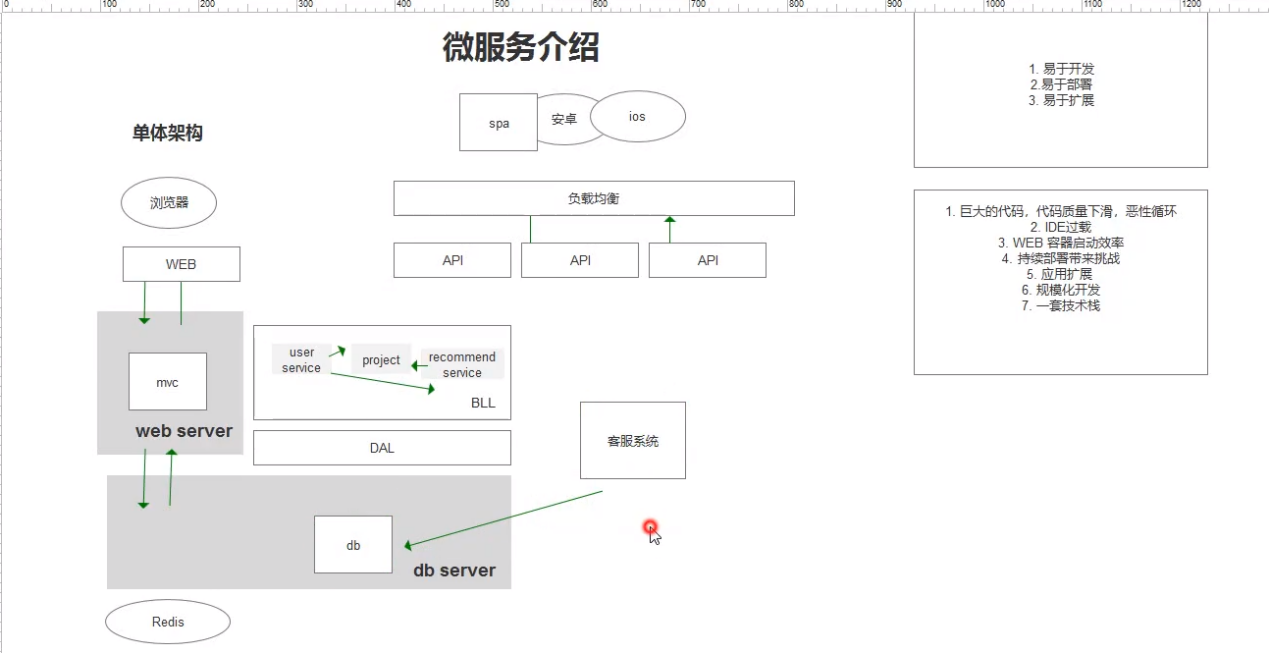
5、

Port(UI)用来管理镜像，可以支持用户名和密码



## 52：架构设计之单体架构

怎么是单体架构

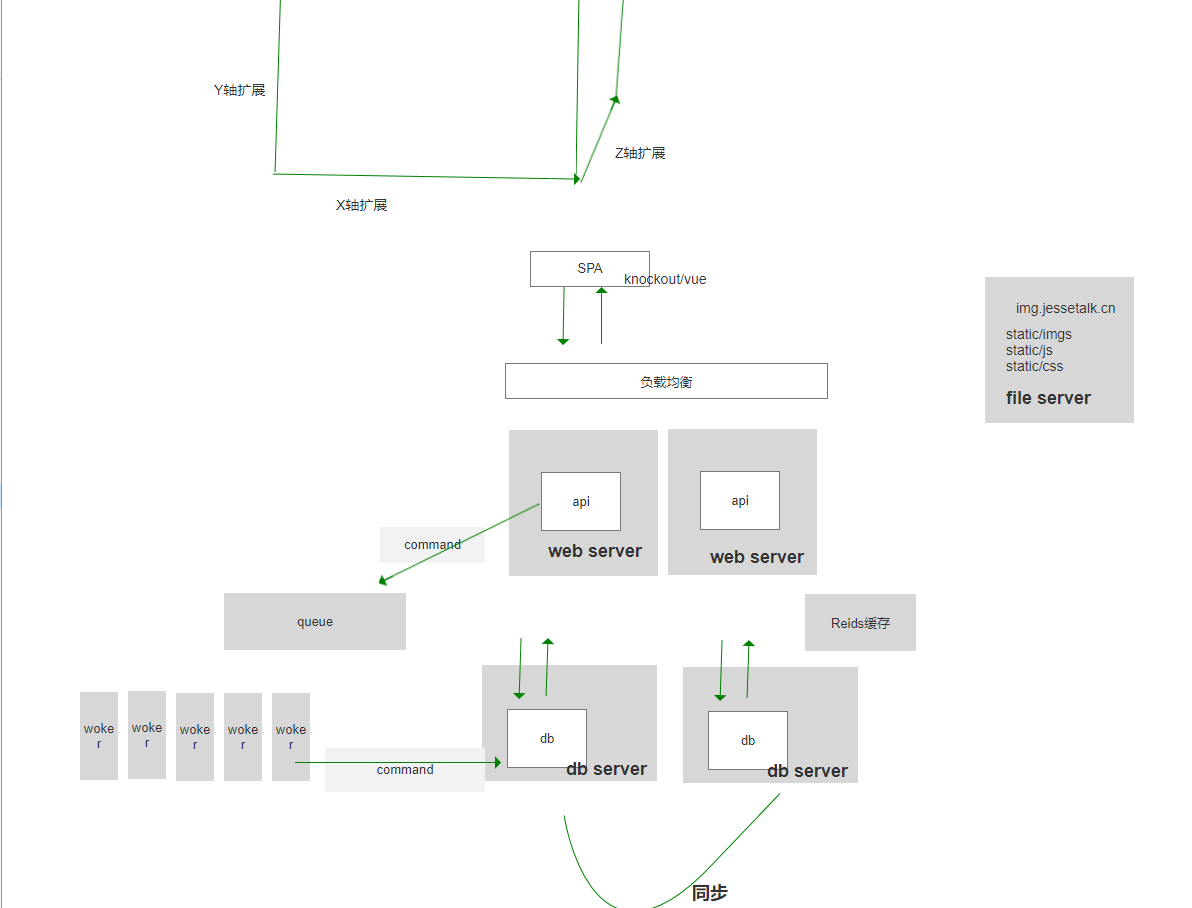


## 53:系统扩展升级演进思路

文档访问

[https://9o90oe.axshare.com/#g=1&p=%E5%8D%95%E4%BD%93%E6%BC%94%E8%BF%9B](https://9o90oe.axshare.com/" \l "g=1&p=%E5%8D%95%E4%BD%93%E6%BC%94%E8%BF%9B)

x,y,z轴扩展



## 54：微服务架构介绍

[https://9o90oe.axshare.com/#g=1&p=%E5%BE%AE%E6%9C%8D%E5%8A%A1%E4%BB%8B%E7%BB%8D](https://9o90oe.axshare.com/" \l "g=1&p=%E5%BE%AE%E6%9C%8D%E5%8A%A1%E4%BB%8B%E7%BB%8D)

微服务是什么

特点：一组小的服务，独立进程，轻量级通信，基于业务的能力，独立部署，无集中式管理

问题：

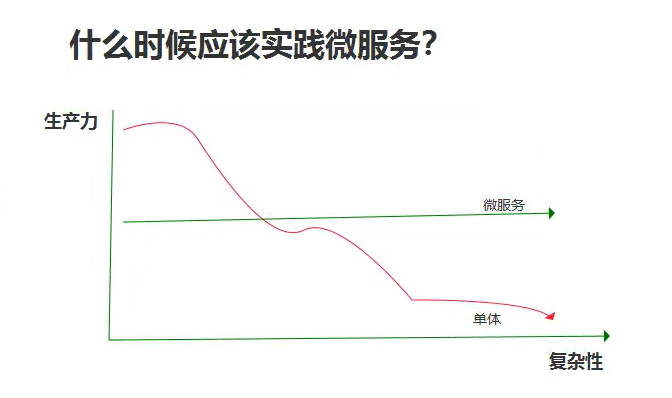
分布式所带来的问题：问题追踪，测试 ，部署，监控，分布式事务，

服务治理（注册，发现，跟踪）

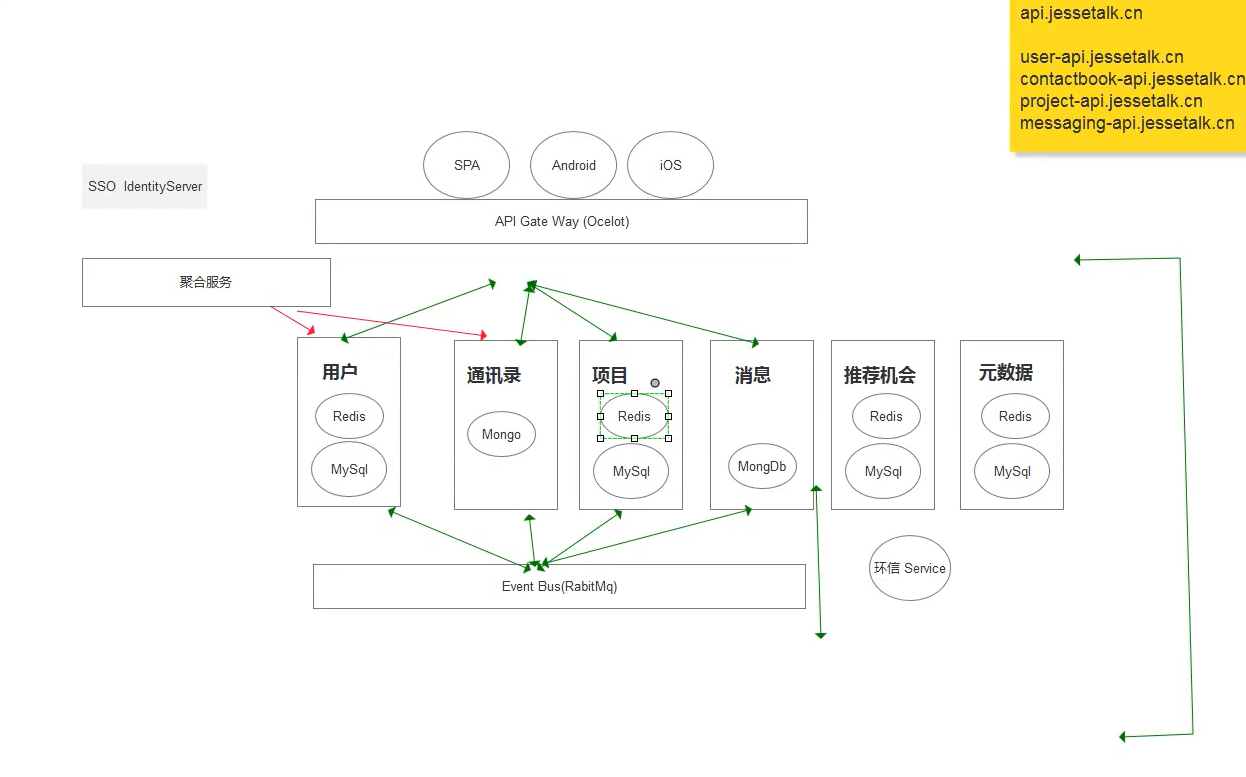
开发难度的加大（聚合查询，）

什么时候应该实践微服务

生产力（低），和复杂性（高）



## 55:面向微服务架构的单体架构

56: UML业务分析之用户

面临第一手业务，

根据用户的角色和行为（功能）来确定业务

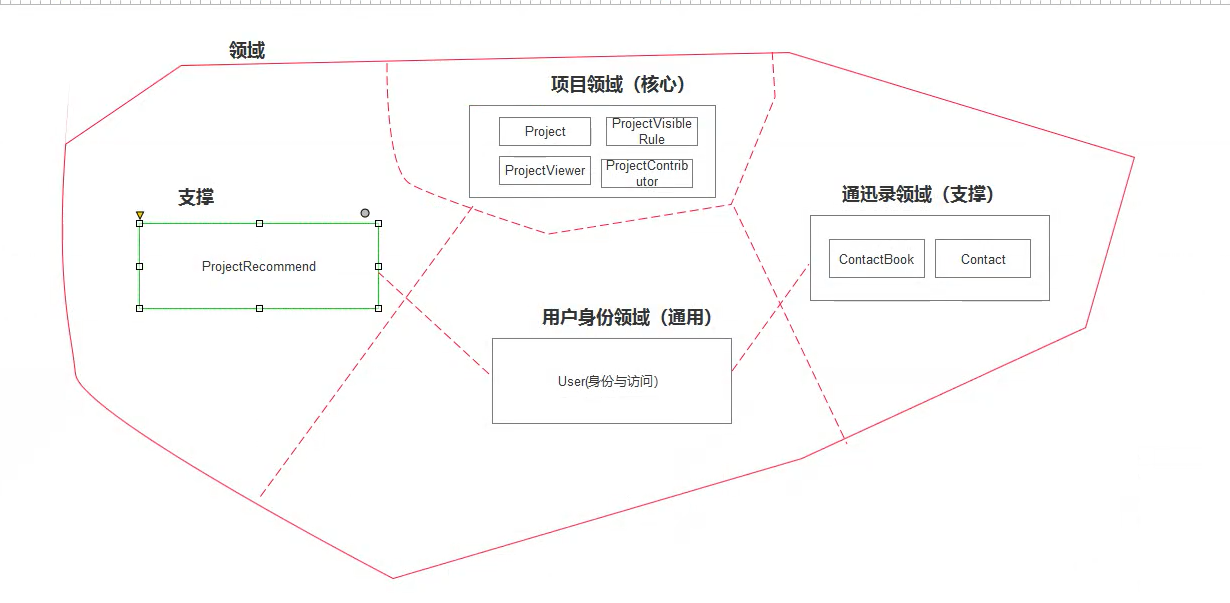
通过业务来确定角色

角色：

App用户

## 59:微服务拆分与界限上下文的确定

如果是跨服务查询，需不需要冗余？如更新不频繁，可以冗余



## 60:用户服务EF Core模型设计

创建一些表

聚合根

## 61:EF Core 数据库初始化

Docker windows 创建资料券

1 查看资料卷

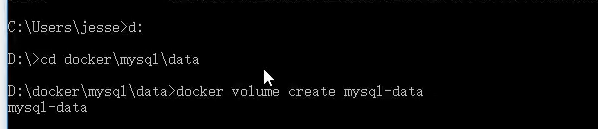
Docker volume ls

2 删除资料券

Docker volume rm mysql-comfig

Docker volume rm mysql-data

3 创建资料券，需要到指定位置





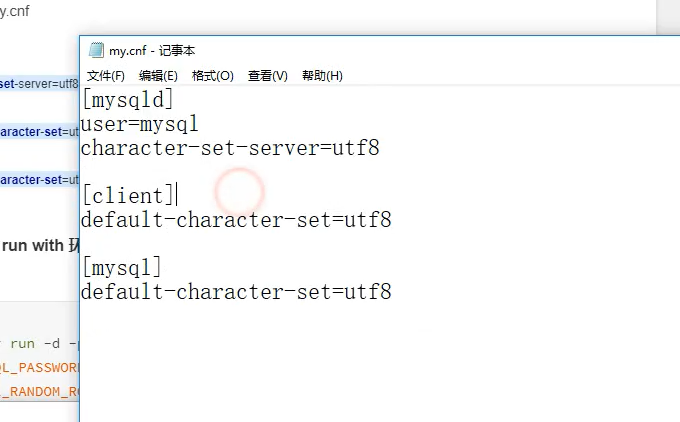
docker run -d -p 3306:3306 -e MYSQL\_USER=jesse -e MYSQL\_PASSWORD=123456 -e MYSQL\_ROOT\_PASSWORD=123456 -e MYSQL\_ROOT\_HOST=% -v c:/docker/mysql/config/my.cnf:/etc/my.cnf -v c:/docker/mysql/data:/var/lib/mysql --name mysql02 mysql:5.7

4：查看密码（docker logs mysql01），如果用过资料卷就看不到密码了用另一种方式

查看字符集

show variables like '%char%'

5找到文件保存(在d盘docker mysql )



添加数据

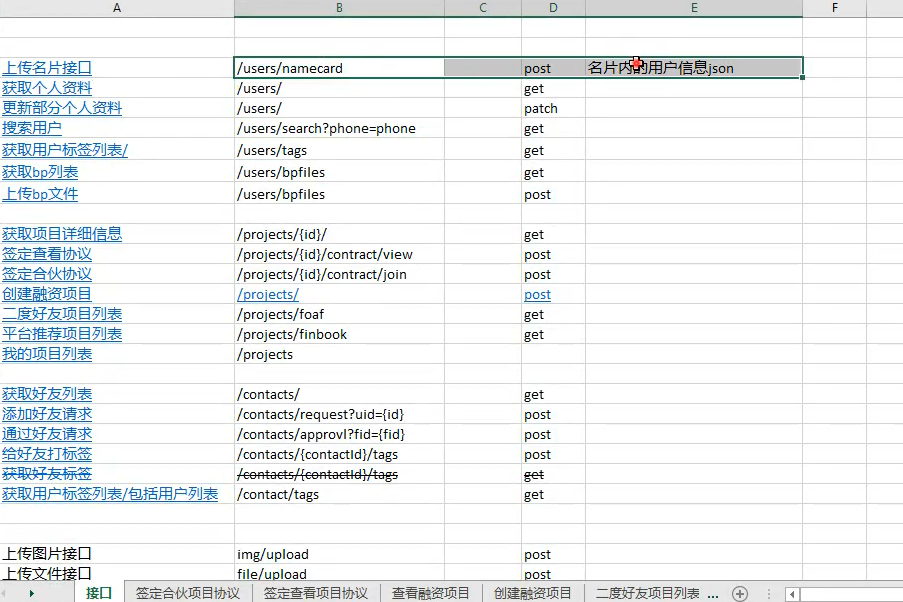
Add-

62 用户服务API设计

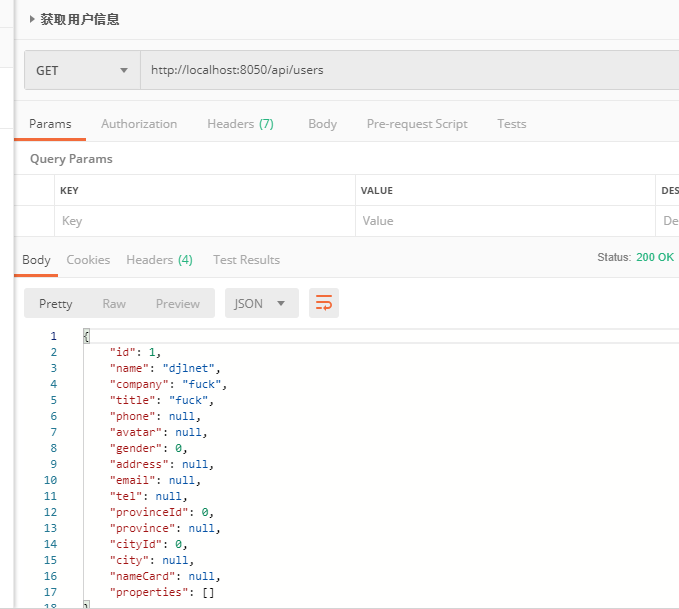
接口设计

：代码写完通过swagger生成

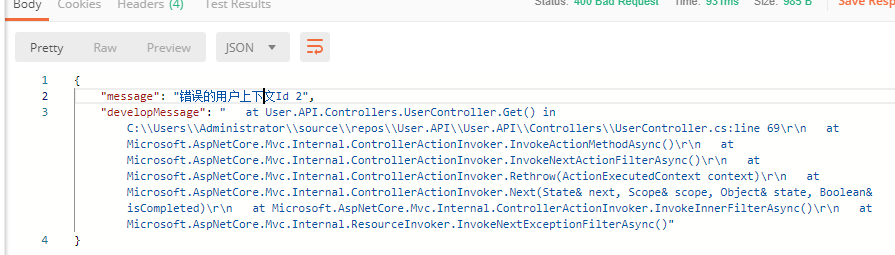
开始的时候通过execl文档编写，方便整理思路



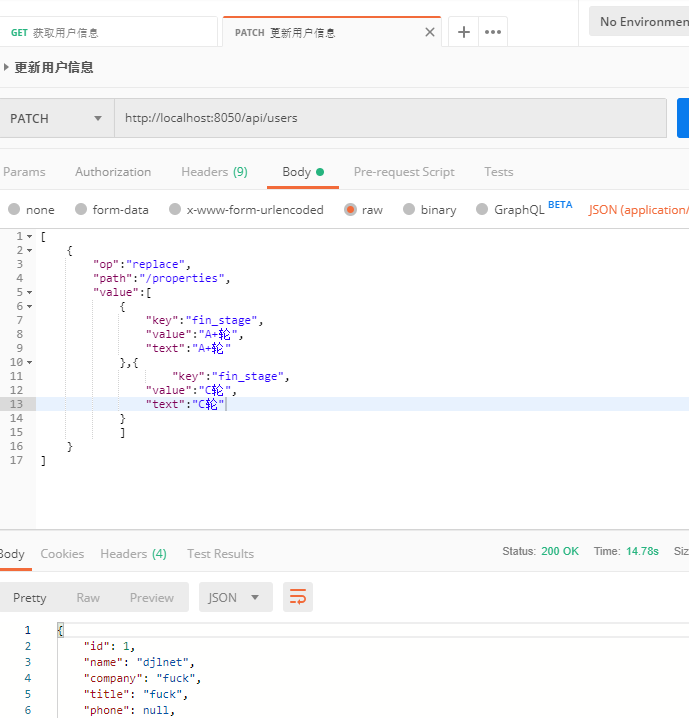
63:获取用户信息-数据访问与模型处理



## 64:全局异常处理与日志记录



## 65：通过HTTP Patch更新用户信息



66:单元测试初探

单元测试，集成测试，TDD

67xunit+Moq+MemoryEFDbContext测试UserController

每次测试都是新的数据源

68FluentAPI写UserController测试用例（上）

## 70:GitLab CI完整部署UserAPI到线上测试环境-上

先要把代码部署到GitLab上

1：先找到要测试的目录

查看有多少方法能测试

dotnet test -t

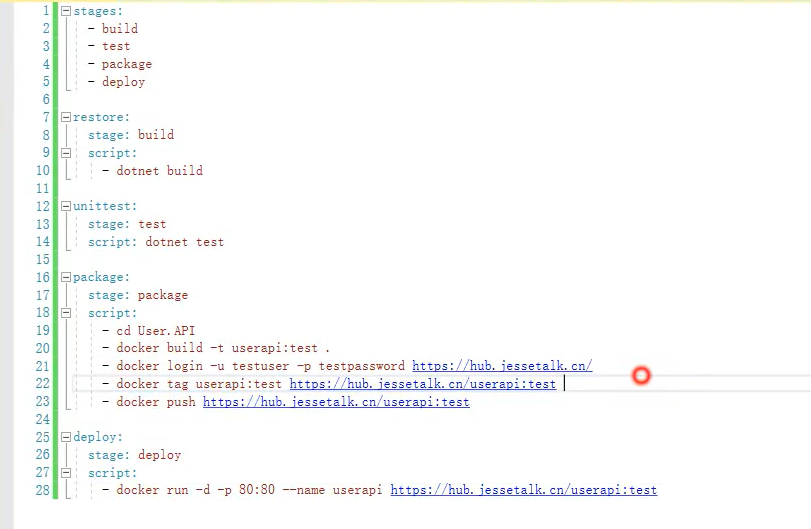


把代码推送到gitlab上

## 71GitLab CI完整部署UserAPI到线上测试环境-中

先要把镜像推送到线上

72 GitLab CI完整部署UserAPI到线上测试环境-下



检查是否能执行成功

Docker build -t userapi:test .

推送镜像

Docker push hub.jessetalk.cn/userapi:test

## 73:GitLab CI完整部署UserAPI到线上测试环境-Debug

服务器需要装sdk

是2.1的版本

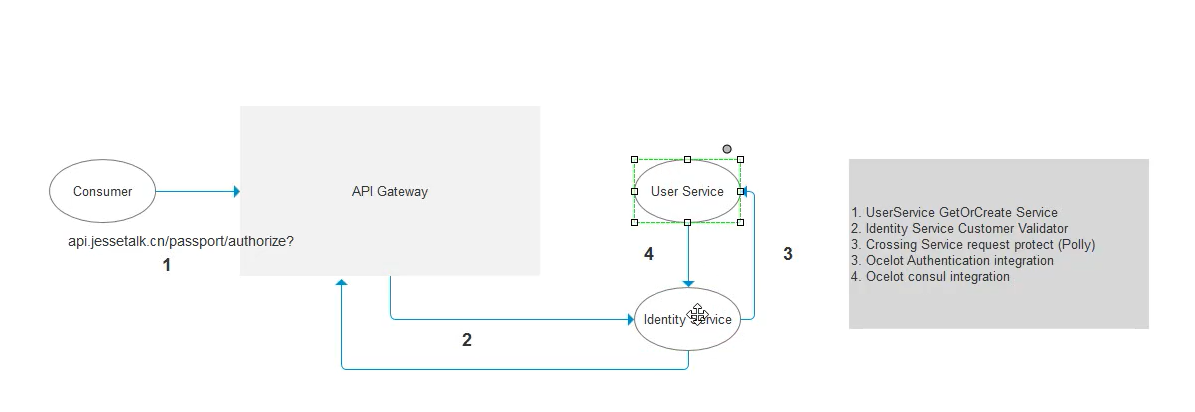
## 74:API网关与服务发现(重点)

<http://www.jessetalk.cn/2018/03/19/net-core-apigateway-ocelot-docs/>

Ocelot是一个用.NET Core实现并且开源的API网关，它功能强大，包括了：路由、请求聚合、服务发现、认证、鉴权、限流熔断、缓存、Header头，并内置了负载均衡器与Service Fabric、Butterfly Tracing集成

Consul 服务发现

## 75:用户注册与登录



## 76: IdentityService4自定义Validator

服务间需要实现接口

## 77: 自定义Validator与UserService互通

文档：

[www.jessetalk.cn/?p=287&preview=true](http://www.jessetalk.cn/?p=287&preview=true)

polly使用

<https://github.com/app-vNext/Polly>

内部服务调用

请求的安全，容错，

## 78: 集成Identity Server4到Identity api并调试通过

认证授权

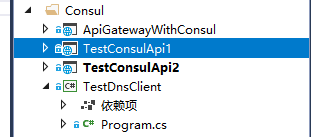
## 79: 添加Ocelot网关并集成identity server4认证

路由代理

添加认证信息

## 80:Ocelot集成Consul服务发现

负载均衡



## 81:微服务之间的服务发现

用来解耦请求地址，地址不被写死

## 82:Polly在eShopsOnContainer项目中的使用设计解析

限流，负载，重治，熔断，容错

## 83:实现ResilienceHttpClient

## 84:集成ResilienceHttpClient到IdentityAPI

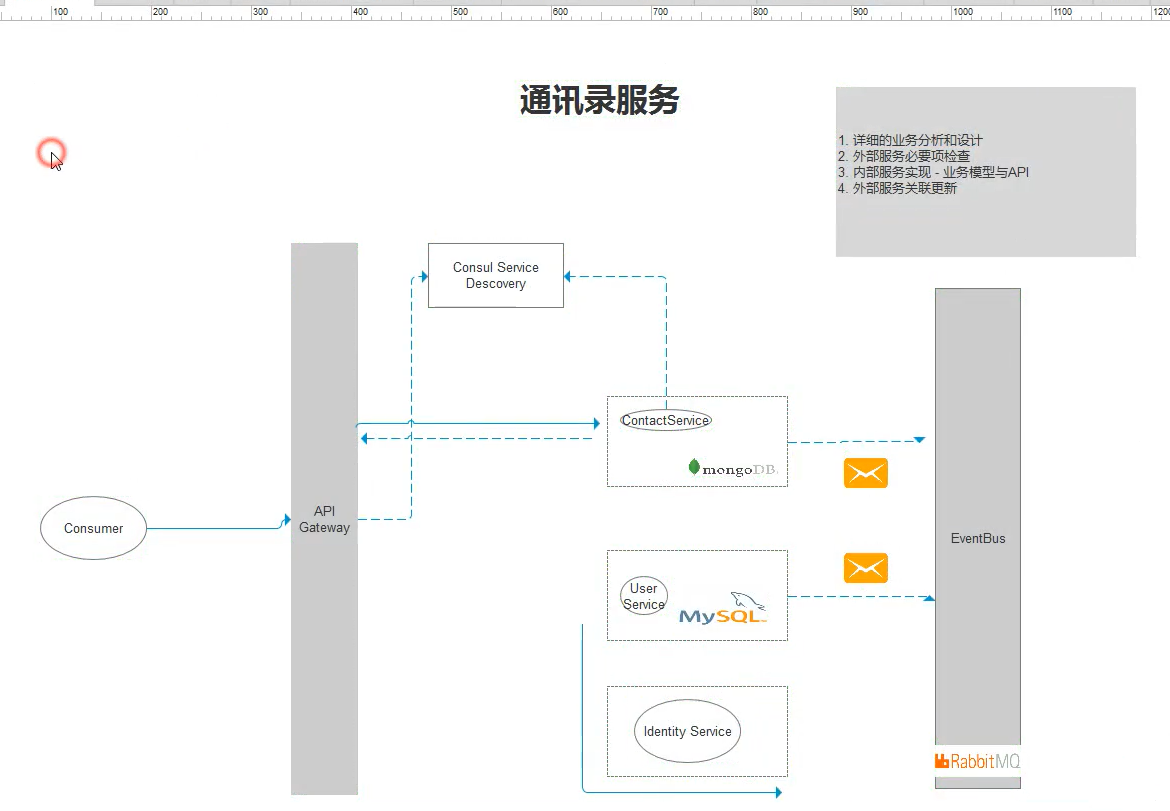
用来创建httpClient

## 85:调试ResilienceHttpClient和IdentityAPI

当api报错的时候，重新发起请求

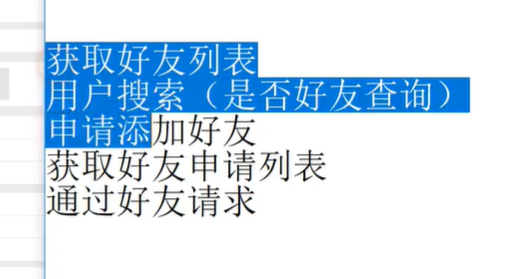
## 86:通讯录服务实现章节介绍

里面用到mongodb，和消息通信



## 87:详细业务分析设计及外部依赖检查

接口



## 88:外部依赖UserService实现

实现几个接口

获取用户标签

通过手机号获取用户信息

更新用户标签

## 89:内部实现 - 用户好友申请接口

实现几个接口

## 90:用户好友申请实现MongoRepository上

实现mongo数据上下文

## 91用户好友申请实现Mongo查询与更新的基本用法

Windows mongo 安装方式

<http://www.moguf.com/post/windockerrunmongo>

## 92:用户好友申请实现

## 93:用户好友通过实现

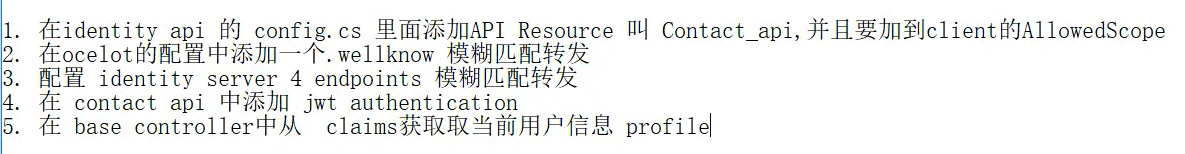
## 94:将ContactApi加入网关



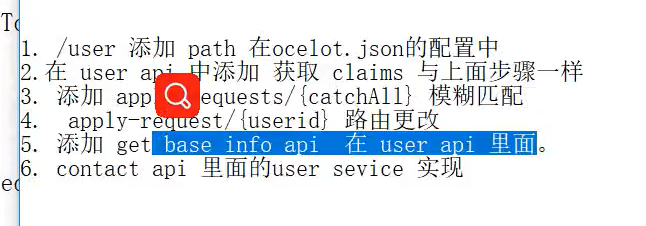
## 96:profile调试

把用户信息写入IdentityServer4中

## 97：在contact api 中获取profile



## 98Contact API 调试



## 99:实现Contact API 向 User API的调用(重点)

## 100:Contact API 完整运行

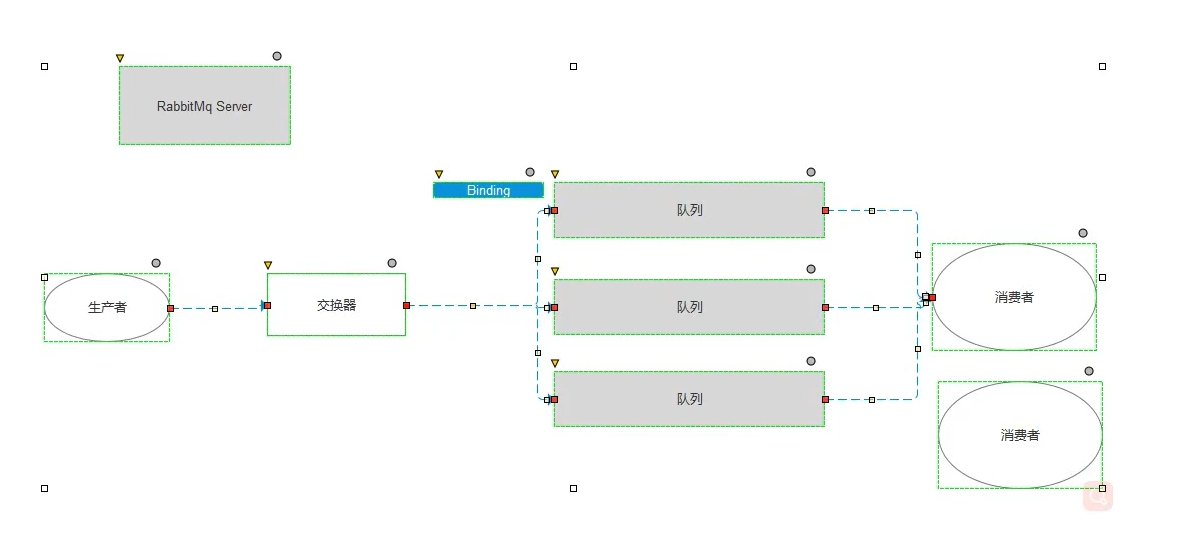
## 101:服务交互 - RabbitMQ介绍

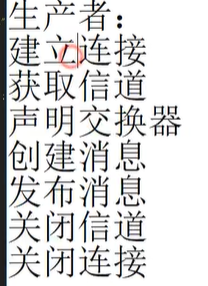
安装

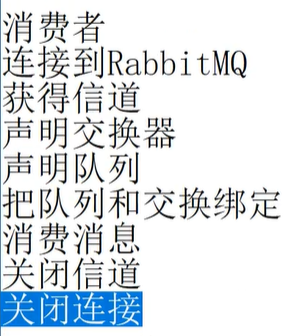
<https://www.cnblogs.com/saryli/p/9729591.html>

实现外部服务关联更新，

例子：有用户服务和通讯录服务，两个服务之间有自己独立的数据和业务，但是他们也有一些交互，当我们通讯录服务需要用到用户服务的数据的时候，我们做了冗余，在用户更新他的资料的时候需要同步过去，在这种机制下需要用到RabbitMQ







## 102:服务交互 - EventBus 之 MediatR,RawRabbit, eShopsOnContainer

## 103:集成Event Bus CAP（重点）

需要创建数据库

安装dotnet.cap

## 104:User API事件发送实现(重点)

事件的发布与消费

## 106:章节介绍项目推荐



## 107:DDD介绍-从三层到OOD OOP

## 108:DDD介绍-结合业务需求来认识各种名词

## 109:DDD介绍-分层

## 110：DDD介绍-CQRS基本介绍

汤雪华博客

读写分开

读的api部署，写的api部署

## 111:项目业务介绍 - 领域事件与集成事件

## 112:项目服务实现- 搭建框架

## 113:项目服务实现- 实现领域entity

## 114:MediatR与领域事件

## 115:创建查看加入Commands

## 116:通过MediatR完成API到Commands调用

## 117:添加EF Repository

## 118:实现EF Repository

## 119:EF 实体配置和migrations数据库生成

## 120:添加推荐服务调用（重点）

## 121:添加Dapper Queries

## 122: 添加Dapper Queries下

## 123:添加服务注册与发现debugging queries

## 124:debugging commands

## 125:响应领域事件发布集成事件上

## 126:响应领域事件发布集成事件下

## 127:添加集成事件处理器

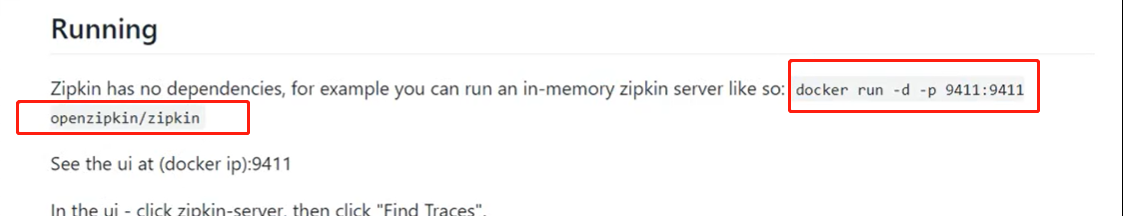
## 128:推荐服务实现 - 添加EF和生成数据库

## 133:分布式追踪介绍以及zipkin server初始化

查错，和性能掌握

用docker 来安装zipkin

安装：

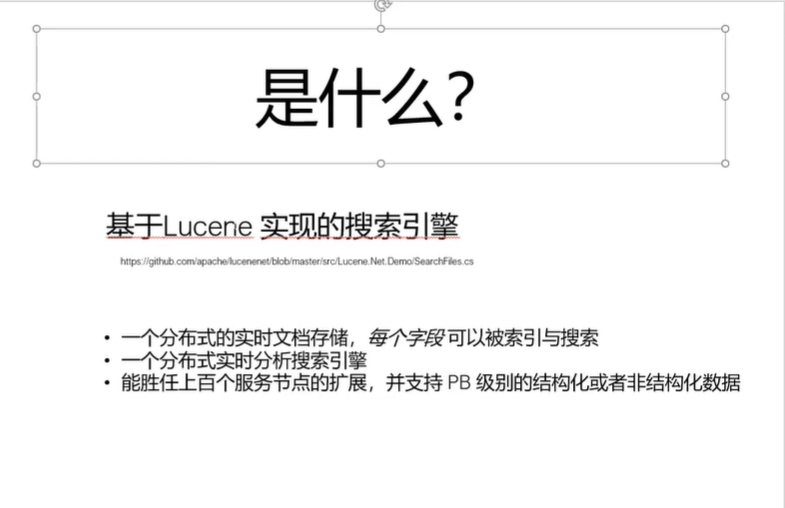


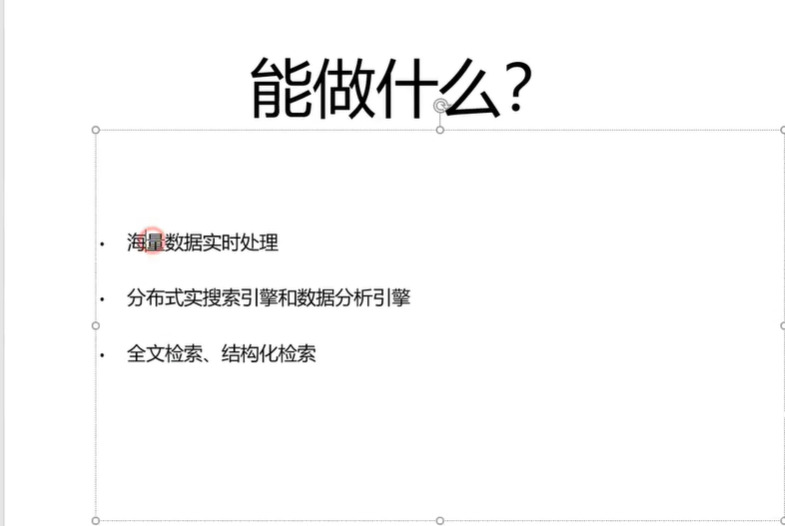
## 135:使用zipkin .NET客户端完成追踪和数据发送

## 137:Elastic Search30分钟快速入门

日志监控

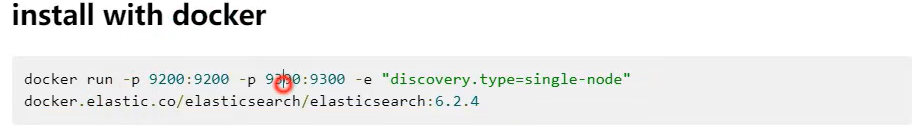


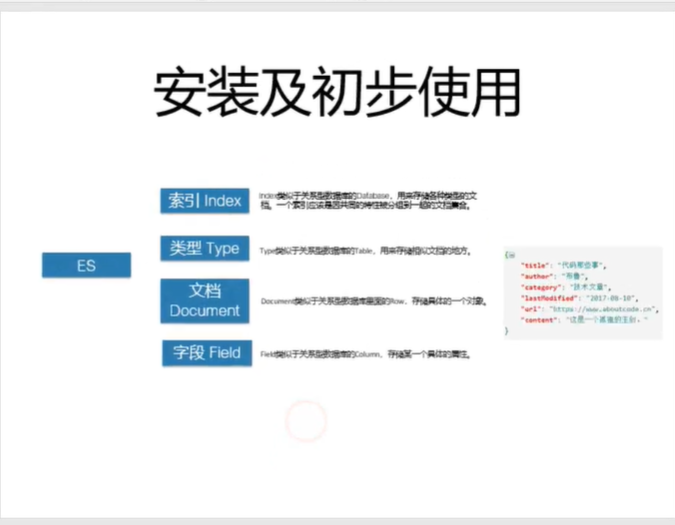




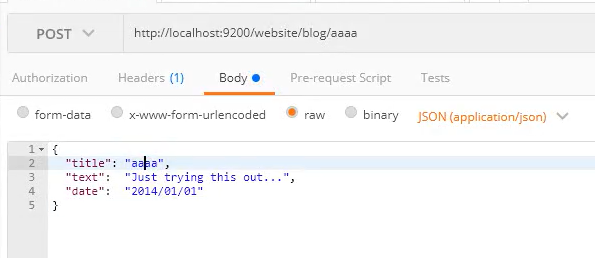


ES安装



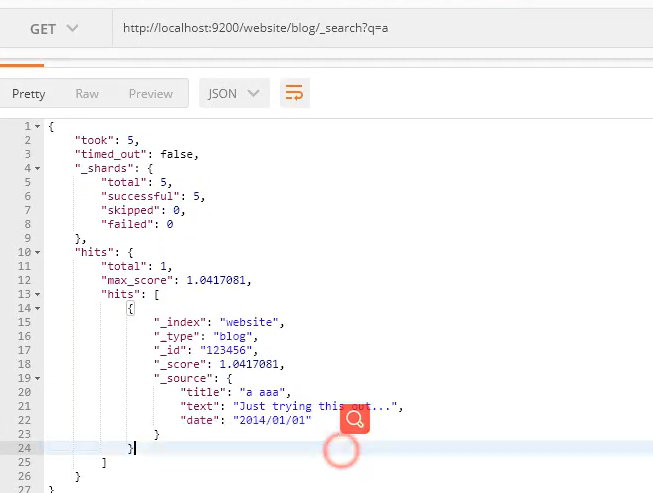


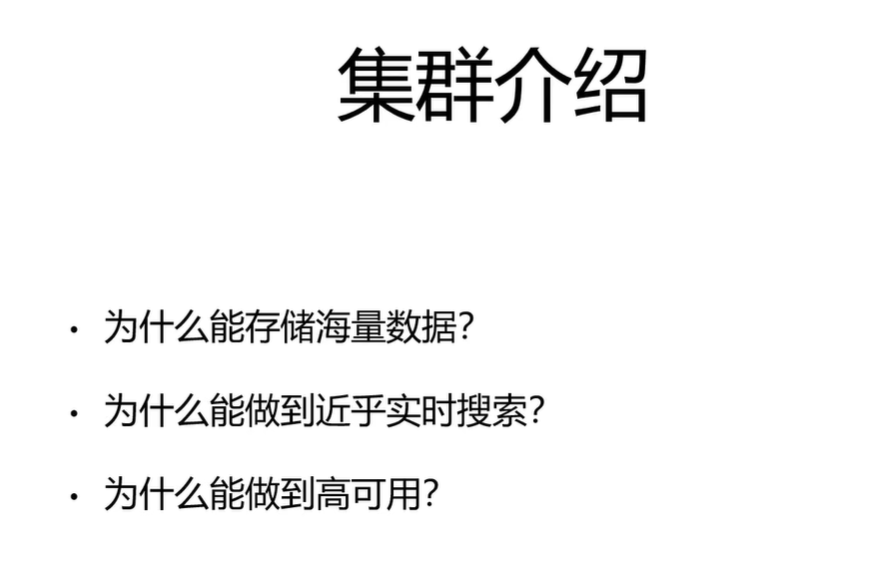
创建文档



查询文档

检索文档

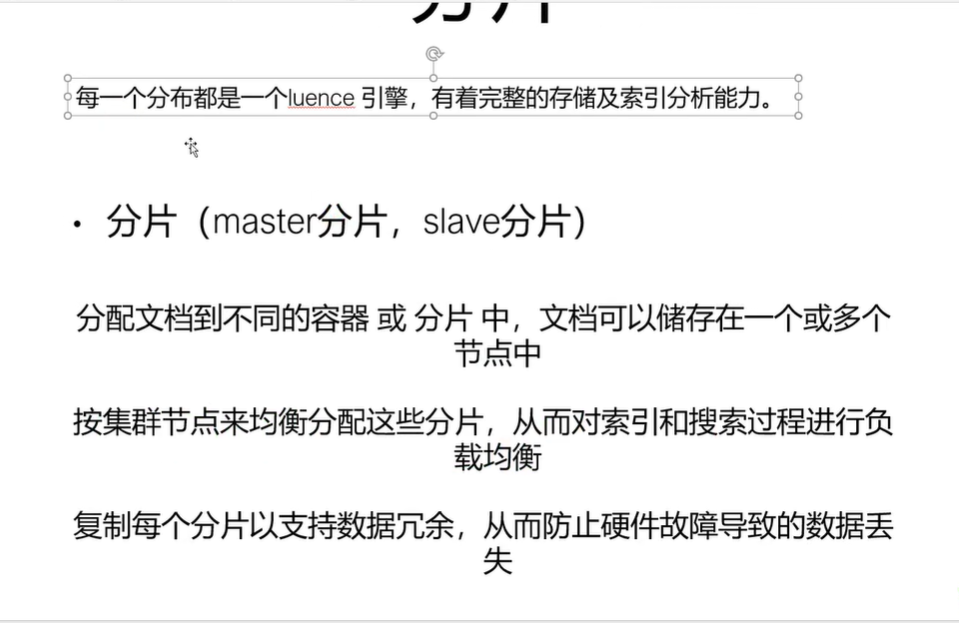


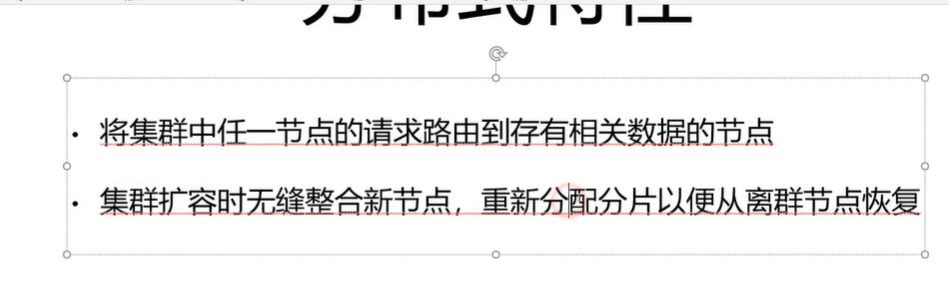






分片





## 138：日志系统开篇



## 139:日志的类别

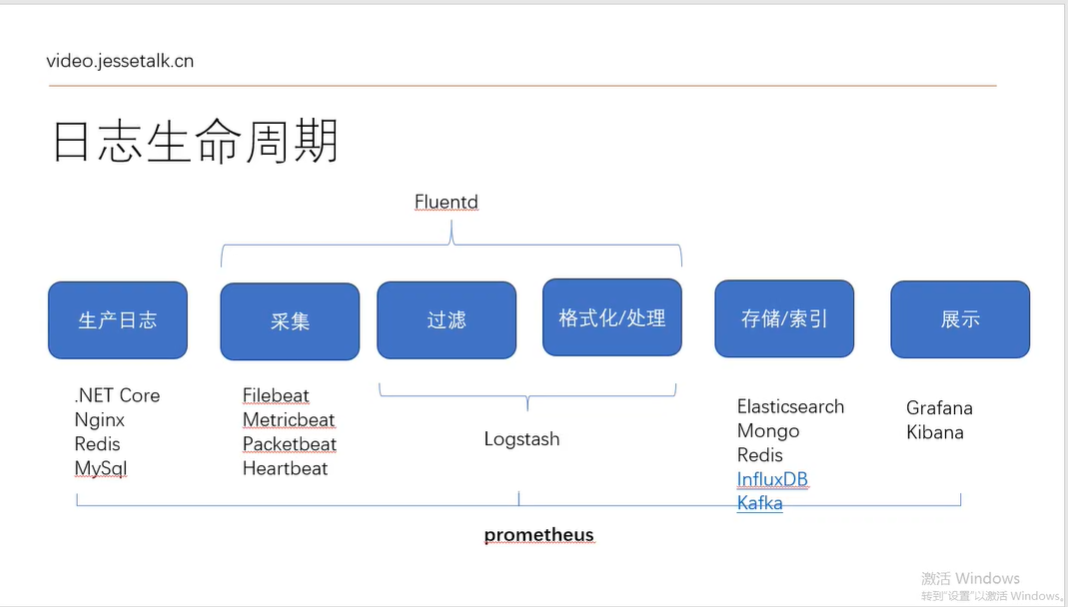


2：Web Sever， Mysql ，Redis 中间件日志

3：监控metrics ,CPU ,内存，

4：健康检查（可用状态）

## 140：日志的生命周期



## 141：EFK介绍(重点需要搭建)



## 142:Fluentd介绍(重点-搭建)

## 143:Serilog介绍(搭建)

## 144:Net Core日志通过 Kibana展示(搭建)

## 145:日志过滤与筛选

## 146:API性能与正确率搜集