
数字货币交易所技术分享

大纲

- 1 业务简介
- 2 整体架构
- 3 应用框架
- 4 核心流程
- 5 其他

业务简介

公司业务简介

- 区块链技术服务商(https://www.chainup.com/zh-CN/)
  - 交易所，钱包，流动性，挖矿，FileCoin，算力云

- ToC：全球300家，覆盖国家20+
- 交易所
  - 主站 Bitwind https://www.bitwind.com/zh_CN/
  - 其他
    - BiKi：https://www.biki.com/
    - 陌陌：https://www.momoex.com/

- **币圈**和链圈

- 行业头部公司
  - 火币(我常用) https://www.huobi.com/zh-cn/
  - 币安 https://www.binance.com/zh-CN
  - OKEX https://www.okex.com/

如何购买比特币？ 这可能是你们最想听的

- 1 场外交易 https://otc.bitwind.com/zh_CN/
  - 场外人民币购买btc,btc到场外账户
  - 场外账户转到币币账户
  - 开始币币交易
  - 或者转到别的交易所(提现到火币钱包地址)
- 2 自己挖
  - 前提要有矿机
- 3 别人转
  - 前提需要有自己钱包

- 如果感兴趣，可以私下交流

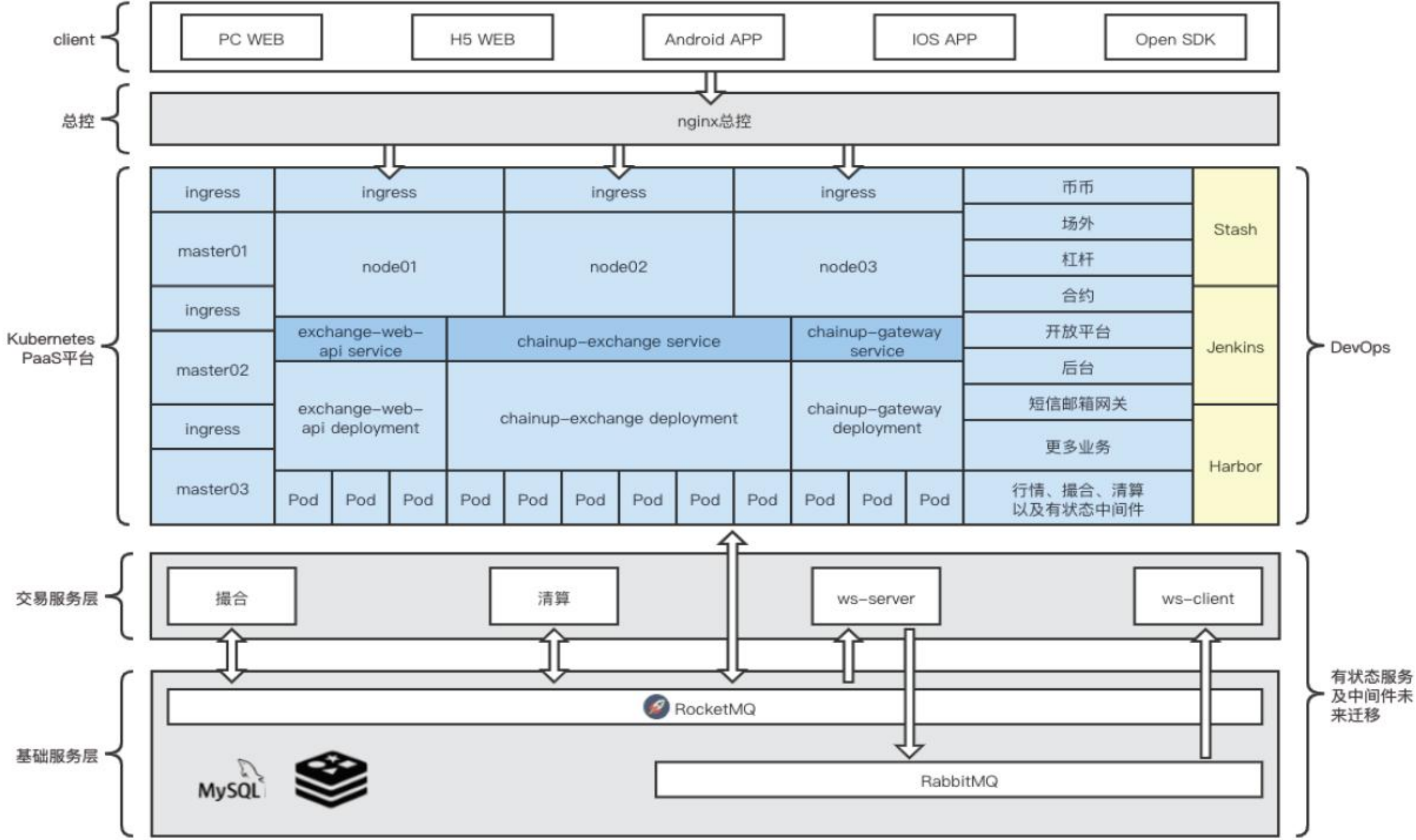
整体架构

平台架构

- 1 基于Kubernetes、Docker、Rancher构建容器云PaaS平台，Helm进行交易所一键部署、版本升级、批量更新、回滚等项目。结合Harbor、Jenkins、Atlassian-Stash(类似于gitlab)进行DevOps交付流程。
  - Kiali
  - Jaeger
  - Isito

- 2 调用链路
  - 之前：client -> nginx总控 -> 各项目nginx -> 各业务模块
  - 接入K8S：client -> nginx总控 -> k8s节点 -> ingress -> Service -> Pod(测试环境)

- 3 部署
  - AWS
    - EKS 弹性K8S服务
    - ELB 弹性负载君合
    - EC2 云服务器
  - Aliyun
    - SLS日志服务 Logtail 数据采集
    - OSS 对象存储
    - ECS 云服务器


client
PC WEB
H5 WEB
Android APP
IOS APP
Open SDK
总控
nginx总控
Kubernetes
PaaS平台
ingress
ingress
ingress
ingress
master01
node01
node02
node03
ingress
exchange-web-
api service
chainup-exchange service
chainup-gateway
service
master02
exchange-web-
api deployment
chainup-exchange deployment
chainup-gateway
deployment
ingress
master03
Pod
币币
场外
杠杆
合约
开放平台
后台
短信邮箱网关
更多业务
行情、撮合、清算
以及有状态中间件
Stash
Jenkins
Harbor
DevOps
交易服务层
撮合
清算
ws-server
ws-client
基础服务层
RocketMQ
MySQL
RabbitMQ
有状态服务
及中间件未
来迁移

生产环境方案

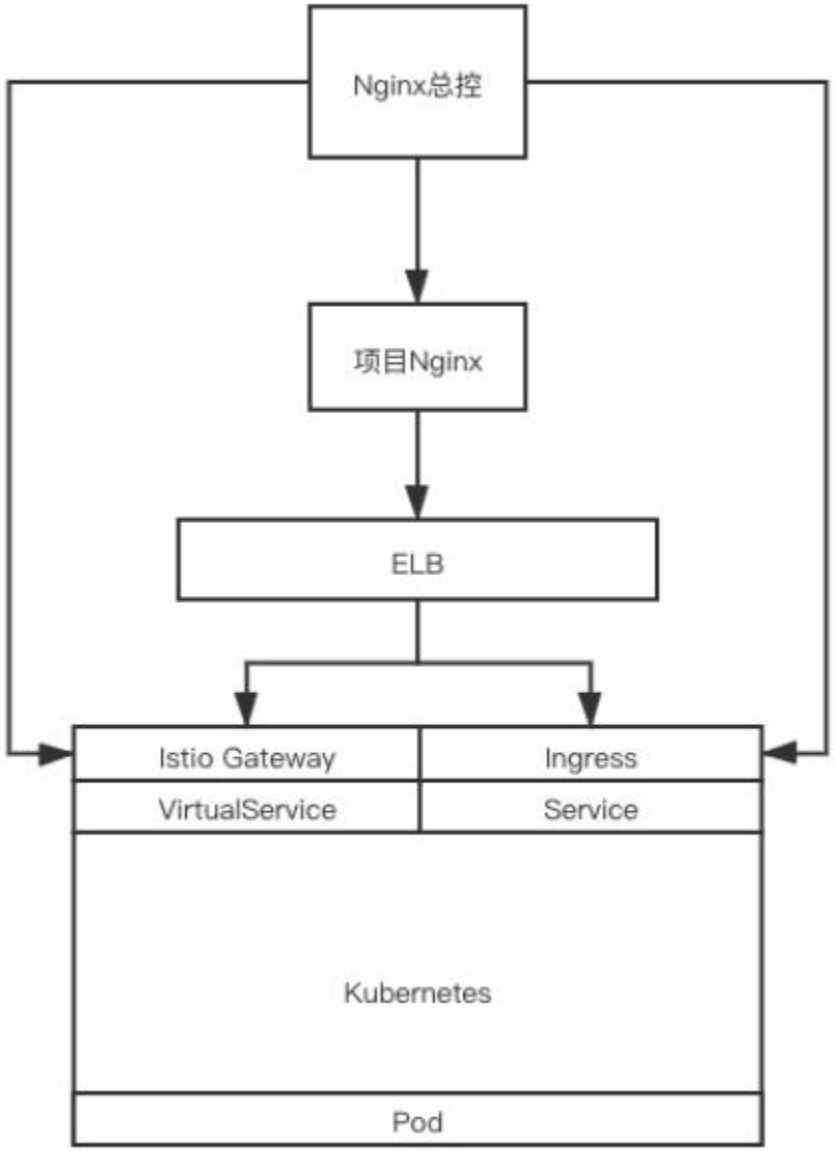
- 采用AWS EKS Kubernetes 1.15托管集群，接入Rancher 2.4GA 管理EKS集群，master节点由AWS维护，我方只需购买EC2增加worker节点
- 两套集群
  - 第一套helm部署rancher HA
    - 由Ingress接入流量，以Deployment形式部署，设置Replicas高可用，创建LoadBalance暴露NodePort提供给ELB转发，域名解析至ELB提供外界访问。(提供页面访问)
  - 第二套承接业务层项目，由Ingrss或者Isito gateway接入流量。
    - Ingress接入流量，以DaemonSet形式部署，为k8s node创建Taints，DaemonSet调度至Taints所在node，由ELB解析至所在node，域名解析至ELB提供外界访问。
    - Istio Gateway接入流量提供灰度发布、故障注入、流量控制、熔断等能力，以Deployment形式部署，创建LoadBalance暴露NodePort提供给ELB转发，由Nginx总控导入流量至ELB。
- Harbor直推生产环境AWS ECR云服务

生产环境方案

- 链路：Host -> Nginx -> Elb -> Loadbalance -> Isito gateway -> VirtualService DestinationRule -> Service Pod

- 监控：用Rancher + Prometheus + Grafana提供k8s原生监控体系。结合阿里云SLS配置事件报警

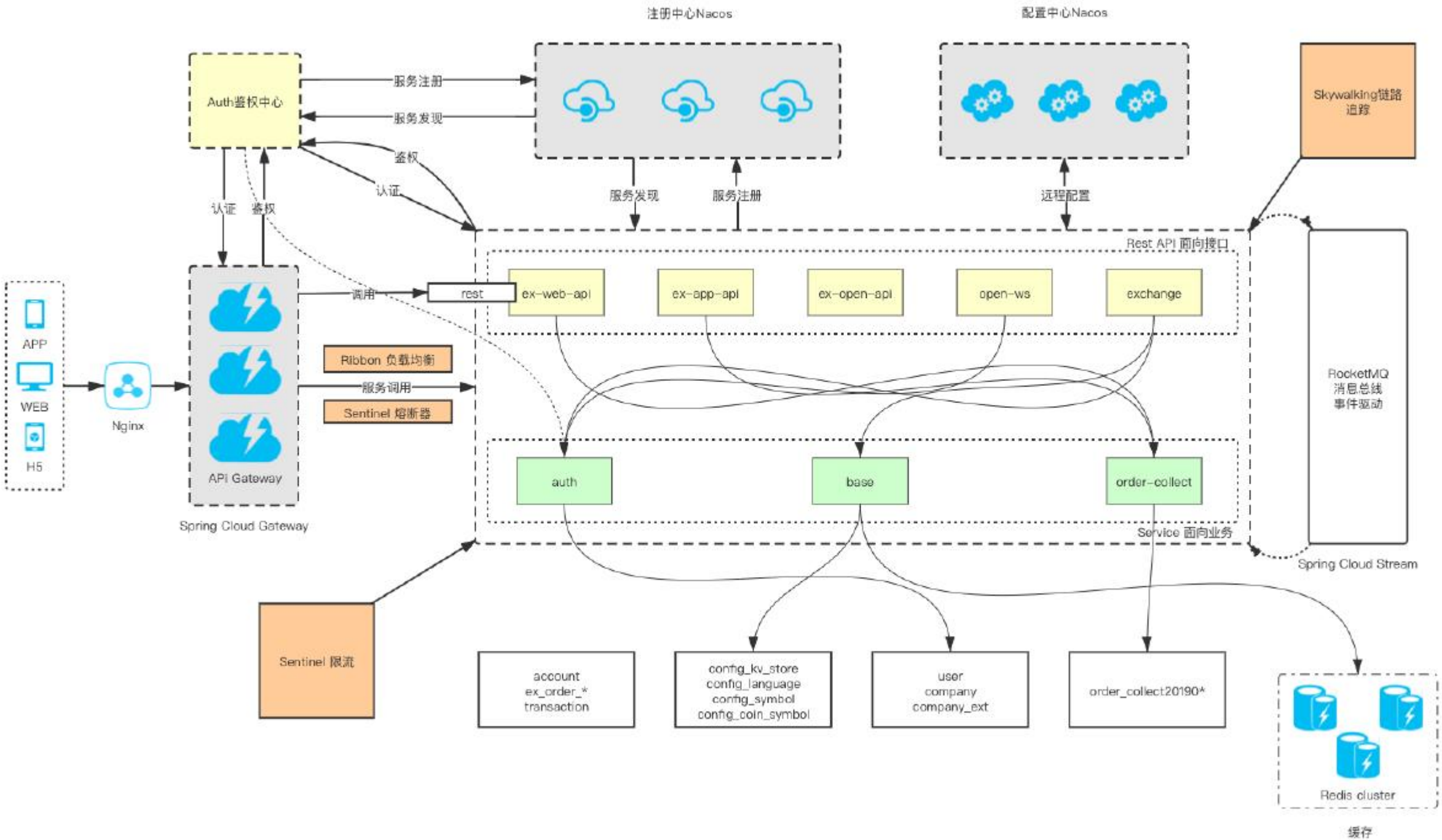
- 日志采集
  - Helm部署阿里云SLS，直接采集Kubernetes集群日志至SLS
  - Helm部署node-problem-detector，采集事件日志至SLS Kubernetes事件中心提供Kubernetes事件报警、监控、统一报表


Nginx总控
项目Nginx
ELB
Istio Gateway
Ingress
VirtualService
Service
Kubernetes
Pod

应用框架

框架总览

- 服务框架：Spring Cloud Alibaba 2.1.0.RELEASE、Spring Boot 2.1.5.RELEASE。
- 注册中心：Eureka -> Nacos 1.1.0 -> k8s service+ coreDNS
- 配置中心：Nacos 1.1.0
- 限流，降级：Guava RateLimter -> Sentinel 1.6.3　熔断： resilience4j
- 链路追踪：Sleuth
- RPC：OpenFeign
- DB与分库分表：Amazon Aurora，Sharding-JDBC
- 服务网关：Spring Cloud Gateway 2.1.0.RELEASE
- 日志搜集：阿里云 Logtail（类似于Logstash ）
- 日志服务：阿里云 SLS（类似于elasticsearch ）
- 中间件：Redis Cluster，RocketMQ，MySQL master-slave,
- 线上问题追踪：Arthas
- 唯一ID生成：uid-generator (https://github.com/baidu/uid-generator)
- 单元测试：junit5+assertj+Mockito+JSONAssert　性能测试：JMH
- 代码审查：Alibaba P3C, NCSS，PMD，SpotBugs，jdepend
- 数据库审计平台：Yearning(http://yearning.io/)
- 短信邮件网关：Gin Web Framework，Redigo，gomail，logrus，go-sql-driver，go.uuid
- 规则引擎：drools+mysql
- 其他：Disruptor(disruptor-spring-manager)，dragonboat(https://github.com/lni/dragonboat/) , SOFAJRaft，RoaringBitmap


注册中心Nacos
配置中心Nacos
Auth鉴权中心
服务注册
服务发现
鉴权
认证
认证
鉴权
服务发现
服务注册
远程配置
Skywalking链路追踪
Rest API 面向接口
APP
WEB
H5
Nginx
API Gateway
Spring Cloud Gateway
调用
rest
ex-web-api
ex-app-api
ex-open-api
open-ws
exchange
Ribbon 负载均衡
服务调用
Sentinel 熔断器
auth
base
order-collect
Service 面向业务
RocketMQ
消息总线
事件驱动
Spring Cloud Stream
Sentinel 限流
account
ex_order_*
transaction
config_kv_store
config_language
config_symbol
config_coin_symbol
user
company
company_ext
order_collect20190*
Redis cluster
缓存

核心流程

撮合引擎

- 演进：DB撮合 -> Java内存撮合+日志簿 -> Go撮合(基于Raft高可用和选主)

- 一个币对一个线程，进入Disruptor，撮合结束，发布事件,通知结算服务，订单状态落库

下单

- 老流程： 下单 -> DB -> 撮合 -> DB
- 新流程
  - 机器人,做市商下单(90%):下单 -> MQ -> 撮合 -> DB
  - 用户下单(10%): 下单-> DB -> 撮合 -> DB
- MQ顺序性: MessageListenerOrderly， 一个币对一个Queue，单币对有序
- MQ重复消费: 内存RoaringBitmap(精确去重)去重订单ID，重启加载没有成交订单。
  - Java Membership: HashSet -> BitSet -> BloomFilter -> RoaringBitmap
  - 大量数据统计 https://github.com/addthis/stream-lib (单机)
    - Cardinality(HyperLogLog)
    - Memership(BloomFilter)
    - Frequency(CountMinSketch)
- MQ伸缩性: 扩容复杂，新建Broker，重新建Topic，上币对来进行负载均衡。后期采用Plusar

分库分表，读写分离

- 分库分表
  - 1 每天千万订单(用户10%，机器人下单90%)，上亿流水
  - 2 分库
    - 按照交易数据与非交易数据分库，
  - 3 分表
    - 订单表：按照币对垂直分表，按照时间水平分表，每周归集，查询最近三个月，对于用户维度搜索不友好（可以单独在按照用户分一次表，或者导入别的存储，如es或者hbase等）
    - 流水表：每日一表，每月汇总
  - 4 订单唯一ID
- 读写分离
  - DAO层Aop+切换数据源+AbstractRoutingDataSource -》 sharding-JDBC读写分离(思路参考 https://blog.csdn.net/liu976180578/article/details/77684583)
  - 多数据源管理
  - 写后读主库

其他

- 1 大量使用ThreadLocal
  - 多商户数据隔离父子线程，线程池，TransmittableThreadLocal
  - 写后读主库，标记线程是否使用过主库
- 2 并行流使用
  - 滥用并行流，导致系统吞吐量下降，在不同ForkJoinPool中使用并行流
- 3 分布式锁
  - 定时任务唯一节点执行
- 4 Redis 热key，大key
  - 先监控
  - 后内存-> 缓存一致性问题
- 5 缓存 caffeine

QA&感谢聆听