

融数数据基于K8S的微服务治理和构建平台

- 现任融数数据北京研发中心CTO，负责公司大数据平台、微服务框架以及DevOps平台的研发工作；
- 毕业于天津大学，毕业后一直从事软件相关研发和架构设计工作，曾经在普元软件任资深架构师、IBM GBS任咨询经理、亚马逊任架构师等，后加入创业公司，从事研发和管理工作；
- 热爱编程，喜欢钻研新技术，对于微服务、企业架构、大数据以及DevOps有浓厚的兴趣



- 融数数据基于gRPC的微服务框架介绍
- 融数数据微服务治理和监控
- 融数数据基于K8S的微服务构建、开发和部署
- 打造适合微服务的技术团队

- 融数数据基于gRPC的微服务框架介绍
- 融数数据微服务治理和监控
- 融数数据基于K8S的微服务构建、开发和部署
- 打造适合微服务的技术团队

社区热度

- 文档多
- 坑少
- 比较容易找到人

架构成熟度

- 方便开发
- 方便迁移
- 多协议支持
- 多语言支持

学习曲线

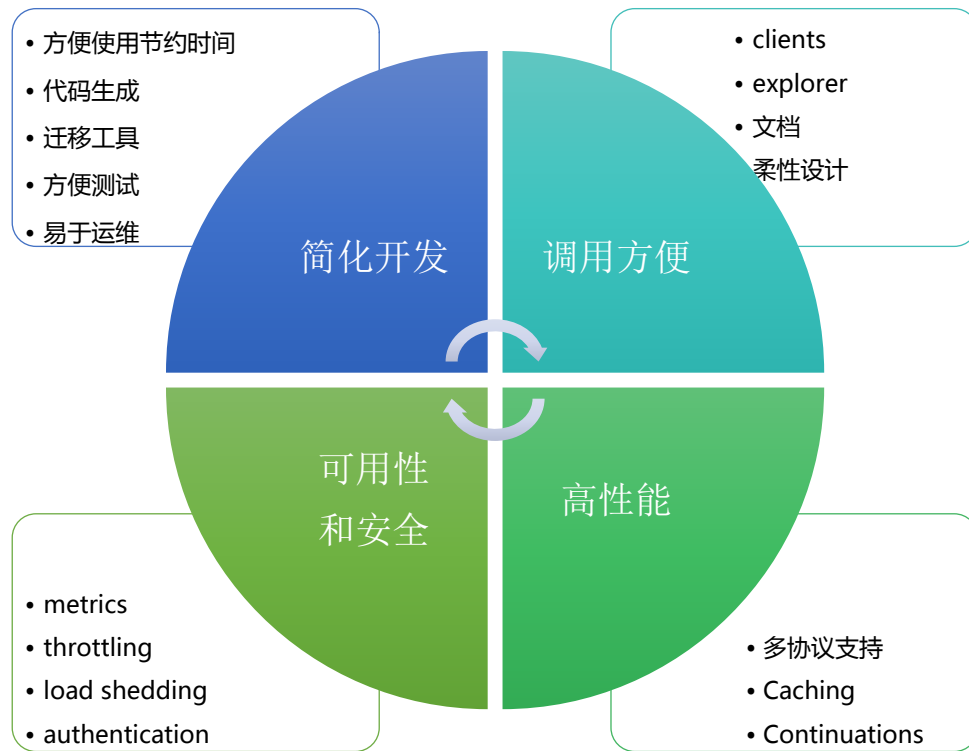
- 基于主流技术
- 现有知识传承

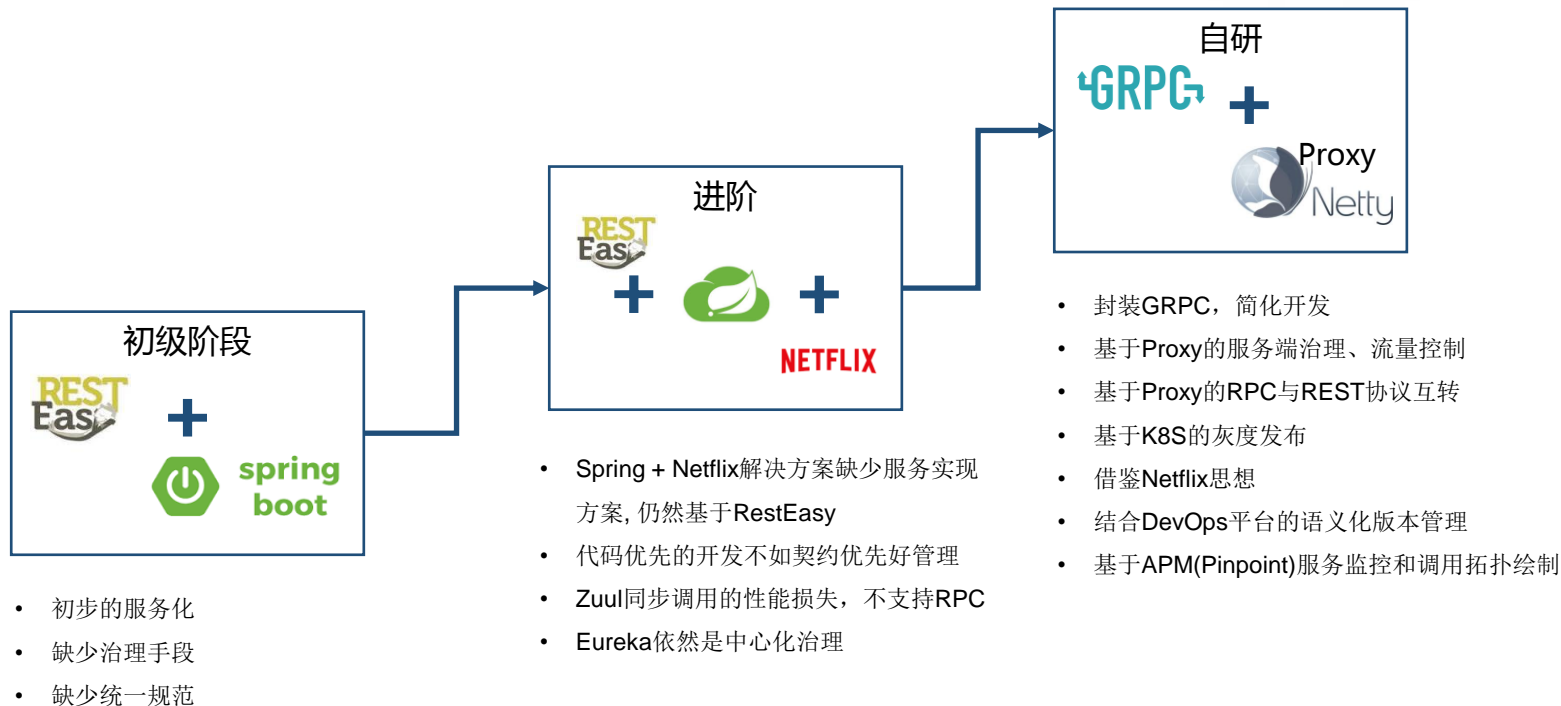
可维护性

- 监控能力
- 运维能力

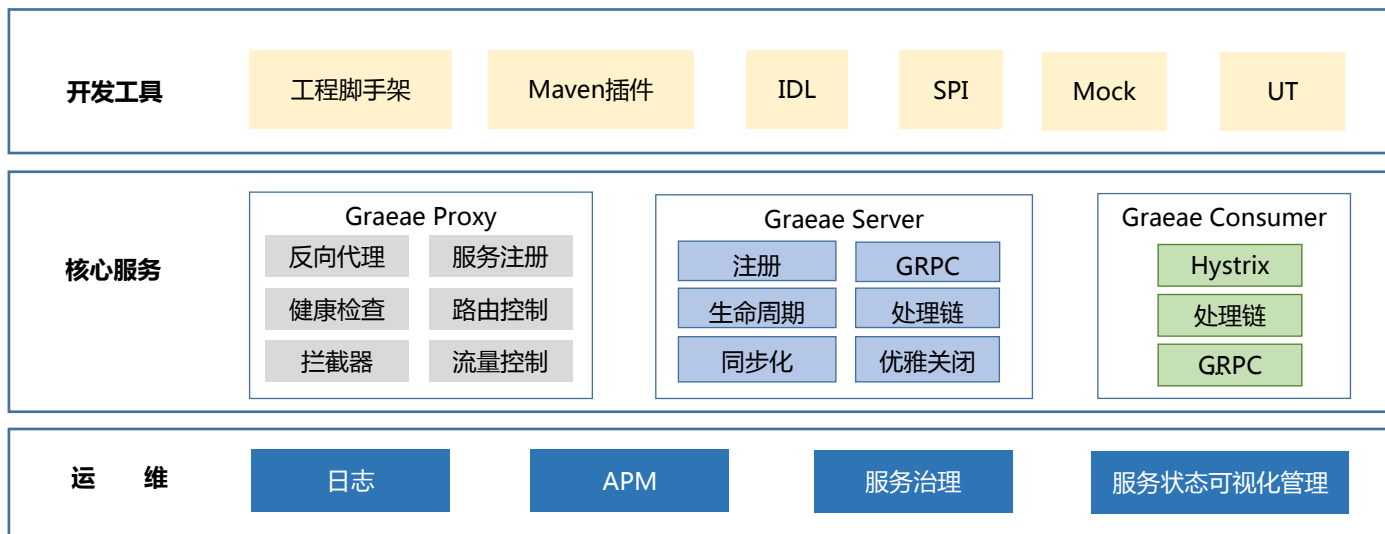
功能点/服务框架	备选方案				
	Netflix/Spring cloud	Motan	gRPC	Thrift	Dubbo/DubboX
功能定位	微服务框架	RPC框架,但整合了ZK或Consul,实现集群环境的基本的服务注册/发现	RPC框架	RPC框架	服务框架
支持Rest	是	否	否	否	是
支持RPC	否	是(Hession2)	是	是	是
支持多语言	是(Rest形式)	否	是	是	否
服务注册/发现	Eureka服务注册表, Karyon服务端框架支持服务自注册和健康检查	是(zookeeper/consul)	否	否	是
负载均衡	是(服务端Zuul+客户端Ribbon)	是(客户端)	否	否	是(客户端)
配置服务	Netflix Archaius Spring Cloud Config Server集中配置	是(zookeeper提供)	否	否	是
服务调用链监控	否 Zuul提供边缘服务,API网关	否	否	否	否
高可用/容错	是(服务端Hystrix+客户端Ribbon)	是(客户端)	否	否	是(客户端)
典型应用案例	Netflix	Sina	eBay/CoreOS	Facebook	用户多
社区活跃程度	高	一般	高	一般	已经不维护了
学习难度	中等	低	高	高	低
文档丰富度	高	一般	一般	一般	高
其他	Spring Cloud Bus	支持降级	Netflix准备集成gRPC	IDL定义	实践的公司比较多,许多企业已放弃,如京东

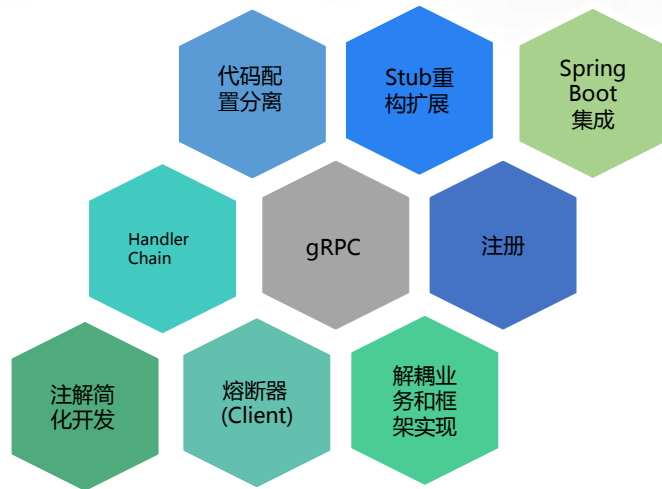
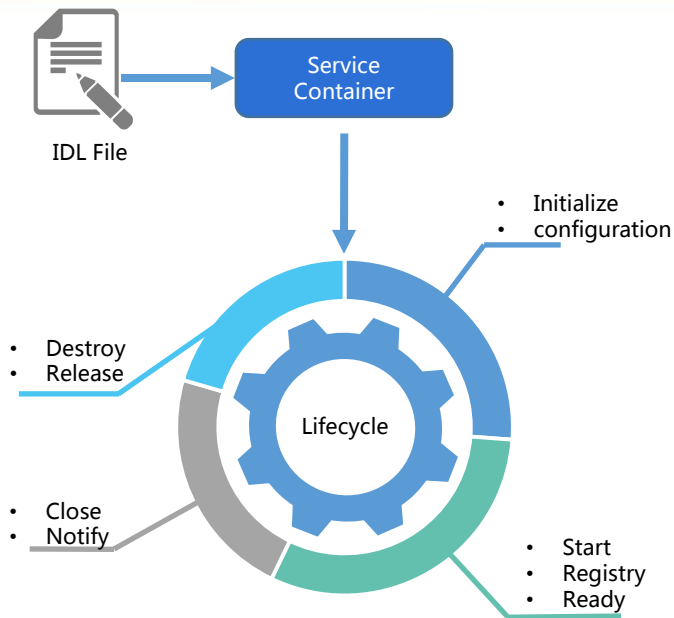
... 那么，实现微服务框架，我们希望得到什么？





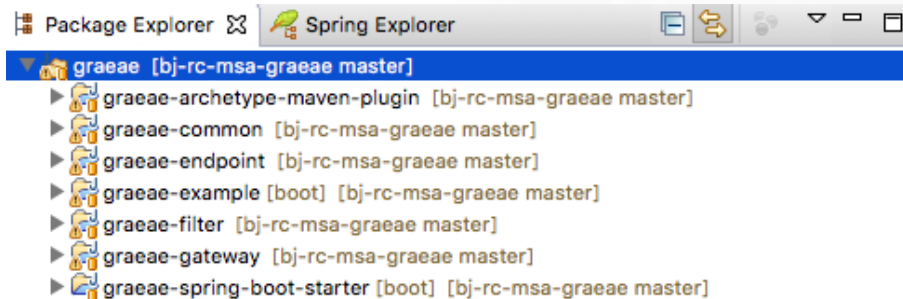
Graeae['gri:i]为希腊神话中可知过去、现在、未来的三盲人女妖,却用同一只眼睛看世界.
取此名亦想体现**微服务**共存共依,又相互独立的特点.

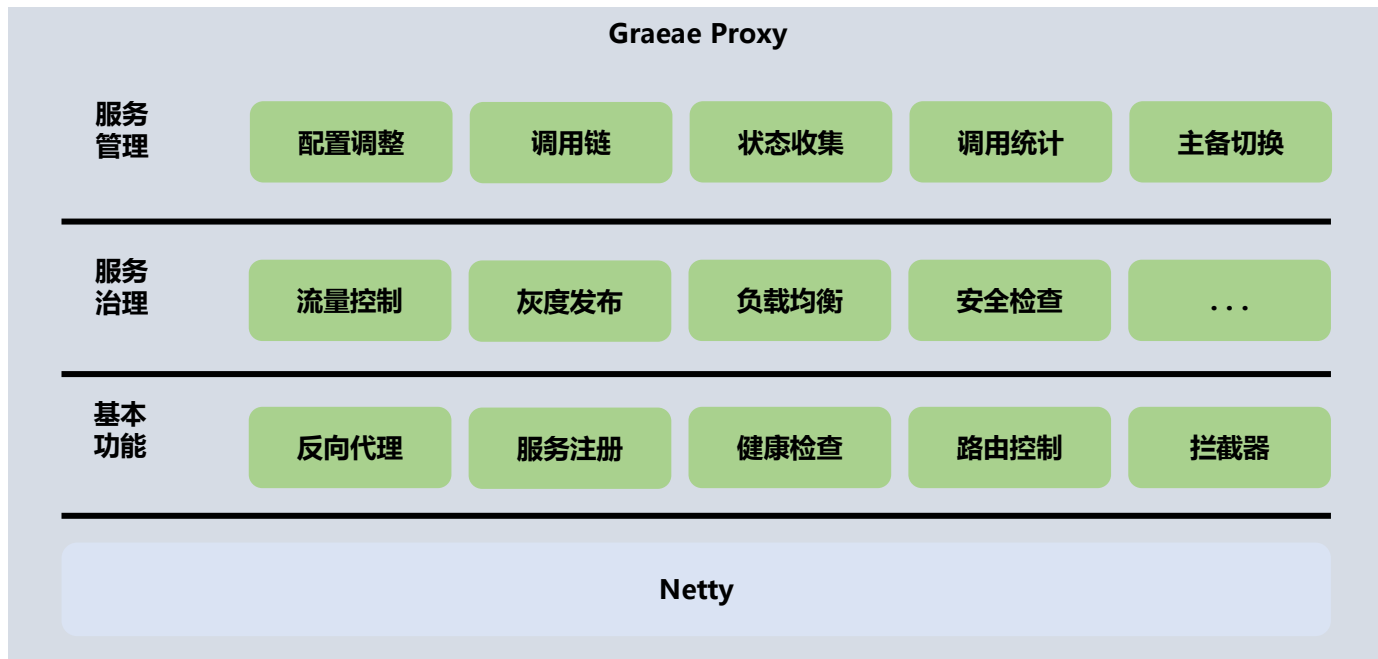




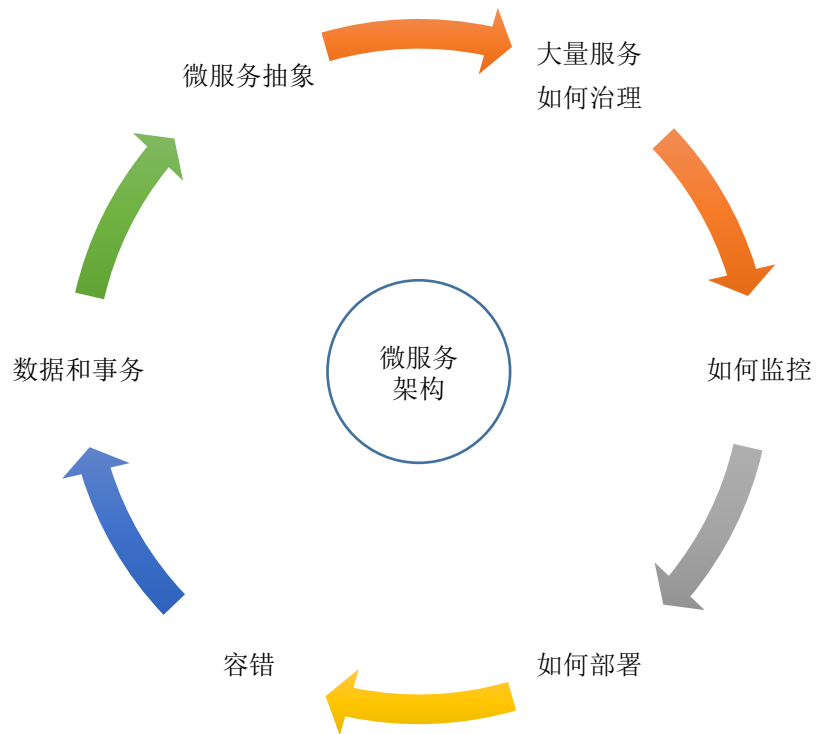
- 抽象基于生命周期的服务容器概念，将服务运行时划分为生命周期的各个阶段
- 在生命周期的各个阶段完成对服务上下文的构建与管理
- 提供对服务端治理的注册、寻址支持
- 提供对部署层的代码、配置分离
- 底层基于gRPC，在gRPC基础上对易用性及功能性进行加强
- 基于annotation标注及stub重新构建，打断业务实现与gRPC的紧耦合
- 重构stub，简化方法调用，屏蔽gRPC stub易用性间隙
- 客户端集成Netflix的Hystrix熔断器，提供fast-fail能力

- 重构gRPC原生代码的生成结构，去掉了内部类和基类继承
 - GrpcClientBuidler
 - GrpcServerBuidler
- 使用annotation简化client，server和service的开发
 - @GraeaeClient
 - @GraeaeServer
 - @GraeaeService
- SpringBoot Starter集成
 - @EnableGraeae
- Client端Hystrix集成
 - HystrixFaultTolerance
- 简化原生gRPC的基于观察者模式的回调实现方式，提供同步和异步两种调用方法
 - getStub().<methodName>InBlock(Parameters)
 - getStub().<methodName>Async(Parameters)
- 基于调用链(FilterChain)的SPI扩展点

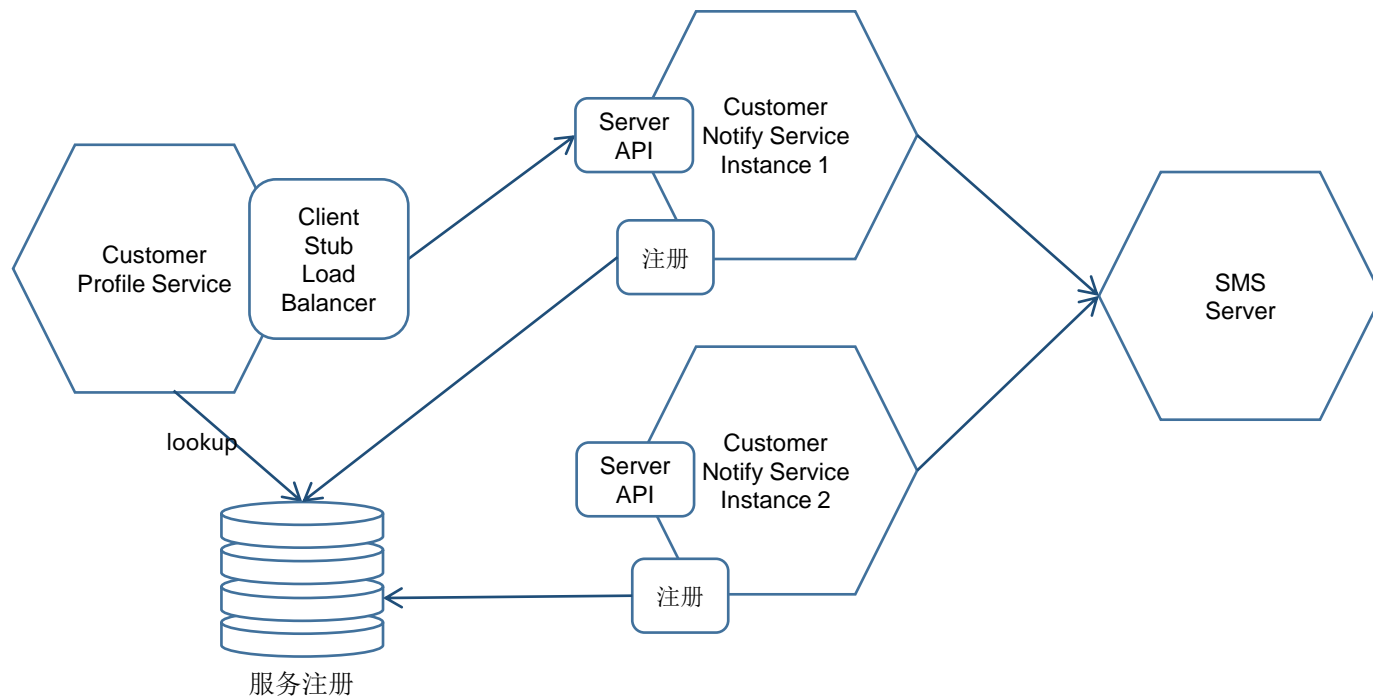


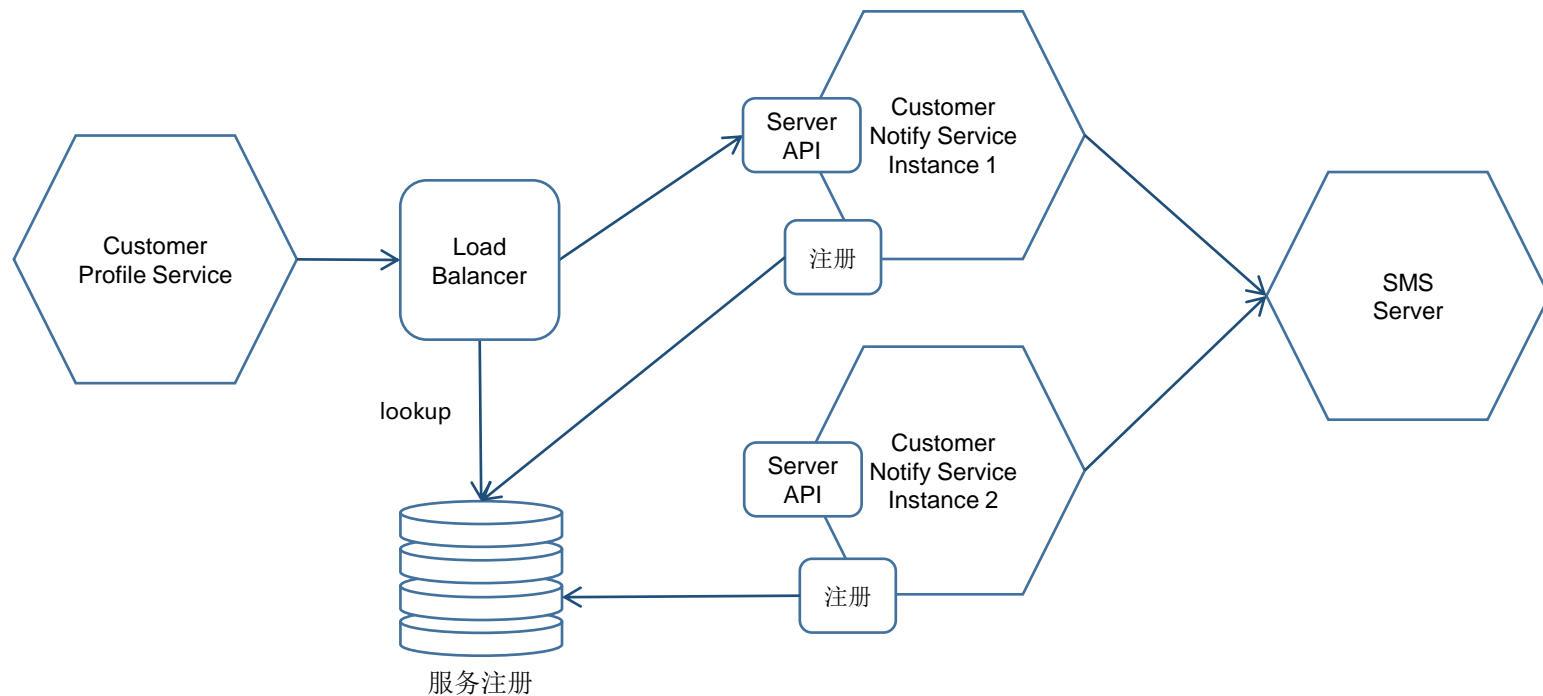


- 融数数据基于gRPC的微服务框架介绍
- 融数数据微服务治理和监控
- 基于K8S的微服务构建、开发和部署
- 打造适合微服务的技术团队

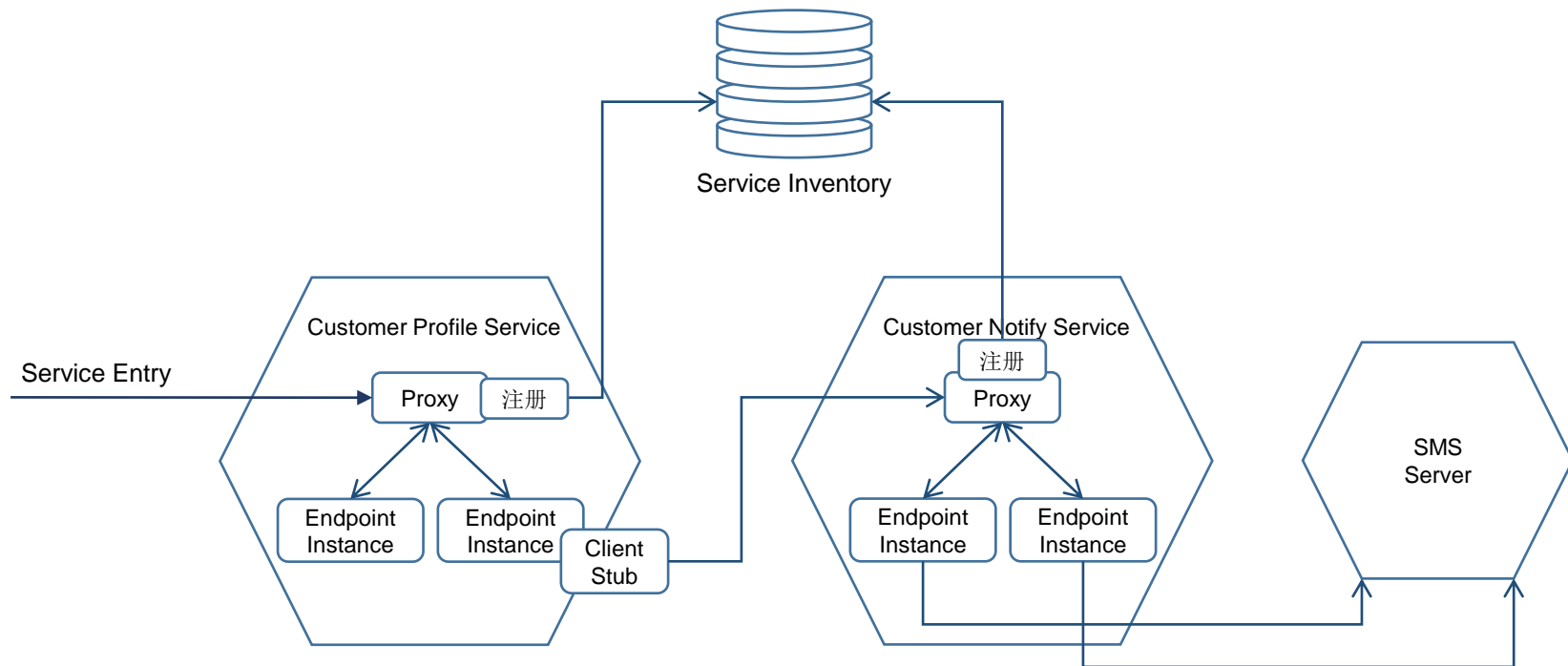


微服务如何治理？

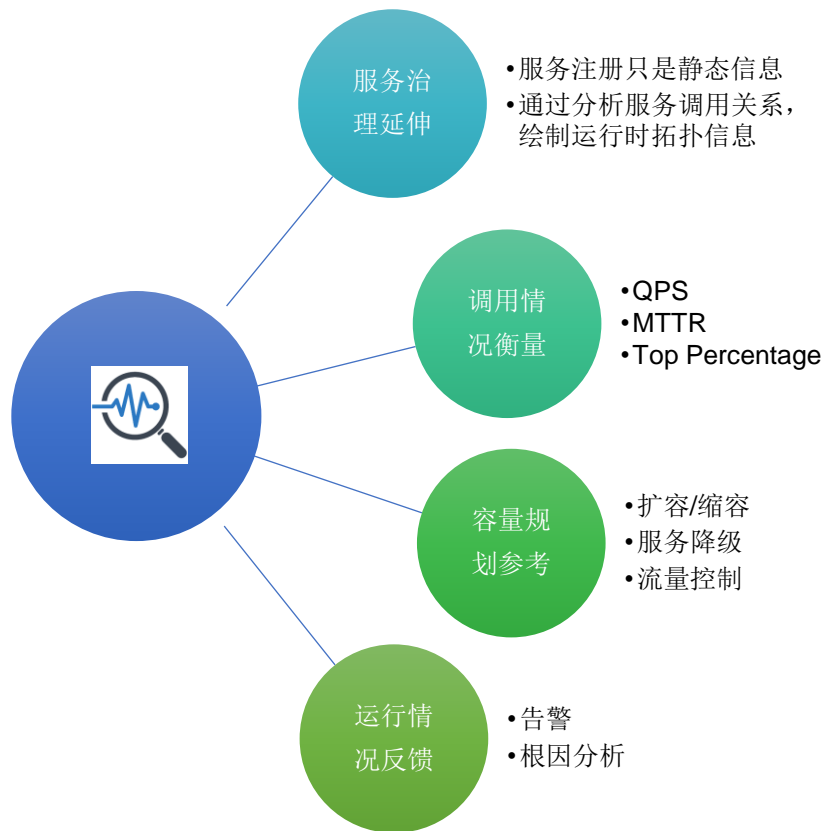




我们如何做？



如何监控？



融数微服务框架监控第一代 zipkin + brave

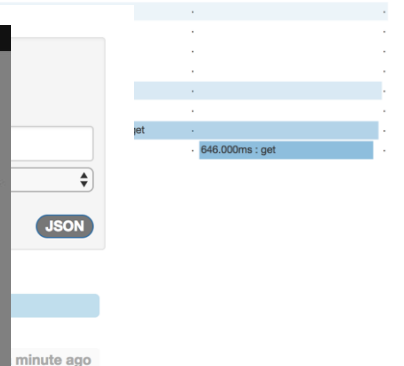
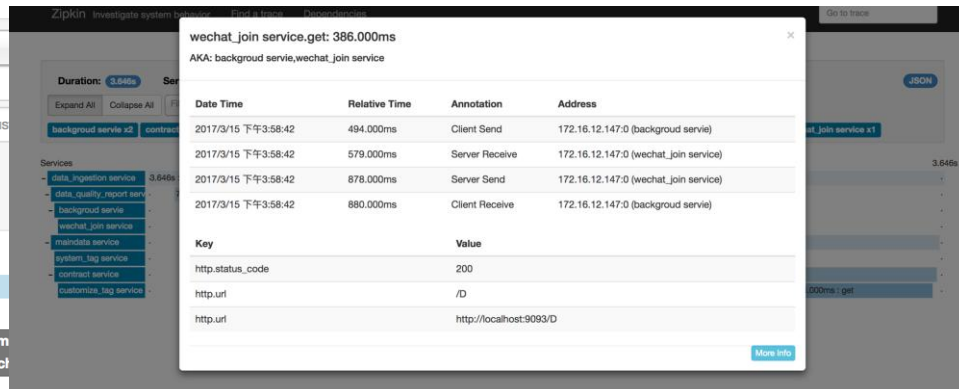
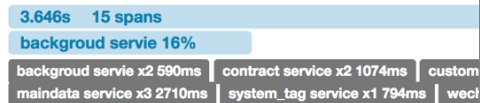
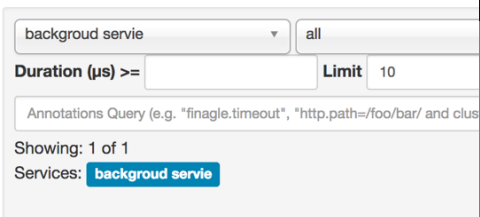
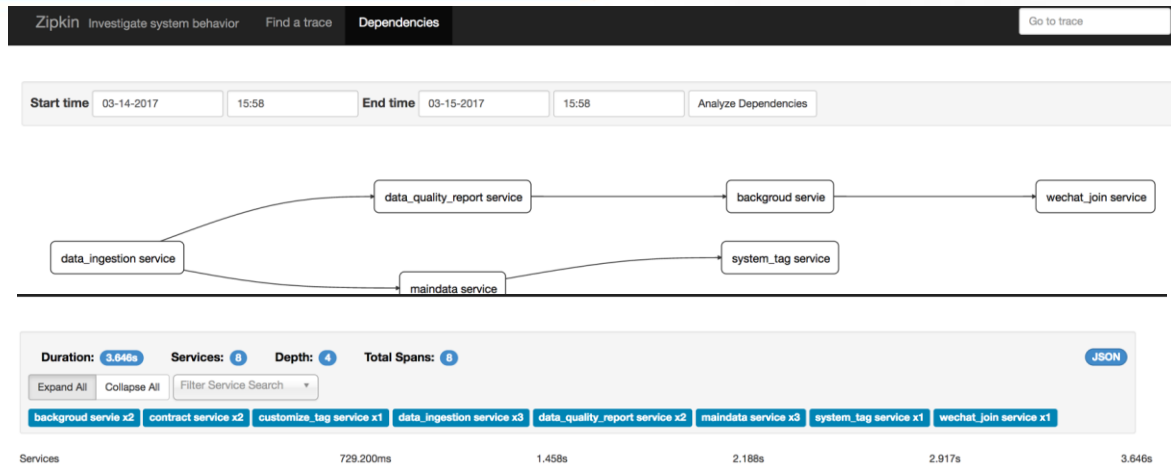
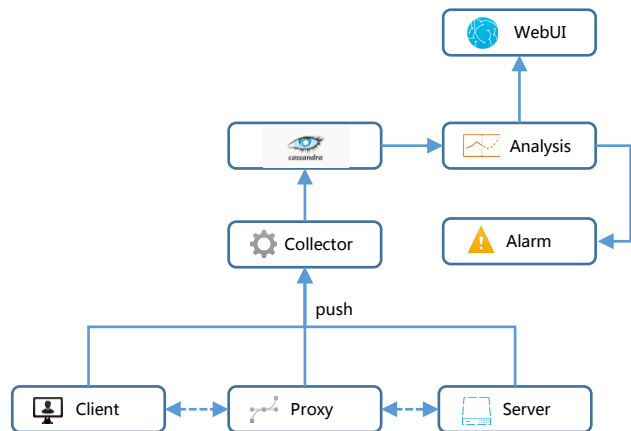
```
@Slf4j
public class BraveUtils {

    private static Reporter<Span> reporter() {
        return new GraeaeLoggingReporter();
    }

    public static Brave brave(int port) {
        Map<String, String> configMap = ConfigLoader.load();
        String serviceName = configMap.get(GraeaeConfigConstant.ZIPKIN_SERVICE_NAME_KEY);
        String sampleRateStr = configMap.get(GraeaeConfigConstant.ZIPKIN_SAMPLER_RATE_KEY);
        String zipkinTestEnableStr = configMap.get(GraeaeConfigConstant.ZIPKIN_TEST_ENABLE_KEY);
        boolean zipkinTestEnable = false;
        float sampleRate = GraeaeConfigConstant.ZIPKIN_SAMPLER_RATE;
        if (null == serviceName) {
            serviceName = GraeaeConfigConstant.ZIPKIN_SERVICE_NAME;
        }
        if (null != sampleRateStr) {
            sampleRate = Float.parseFloat(sampleRateStr);
        }
        if (null != zipkinTestEnableStr) {
            zipkinTestEnable = Boolean.parseBoolean(zipkinTestEnableStr);
        }
        try {
            int ip = InetAddressUtilities.toInt(InetAddressUtilities.getLocalHostLANAddress());
            final Brave.Builder builder = new Brave.Builder(ip, port, serviceName);
            builder.traceSampler(Sampler.create(sampleRate));
            Reporter<Span> reporter = false == zipkinTestEnable ? reporter() : testReporter();
            builder.reporter(reporter);
            return builder.build();
        } catch (Exception e) {
            log.error(e.getMessage(), e);
        }
        return null;
    }
}
```

```
@Slf4j
public class GraeaeLoggingReporter implements Reporter<zipkin.Span> {

    @Override
    public void report(zipkin.Span span) {
        checkNotNull(span, "Null span");
        log.info(span.toString());
    }
}
```



- 基于Spark Job离线绘制拓扑图，实时性不好
- Zipkin + Brave功能比较单一，只有调用链信息
- Zipkin需要依赖被监控框架或者中间件的interceptor机制，代码入侵性较强，需要对现有系统进行配置的改造

融数微服务框架监控第二代 - Pinpoint

- 与Zipkin一样基于Google Dapper构建
- 基于Java Instrument技术，代码无入侵
- Plugins机制，方便扩展
- 支持模块多
- 虚拟机状态监控

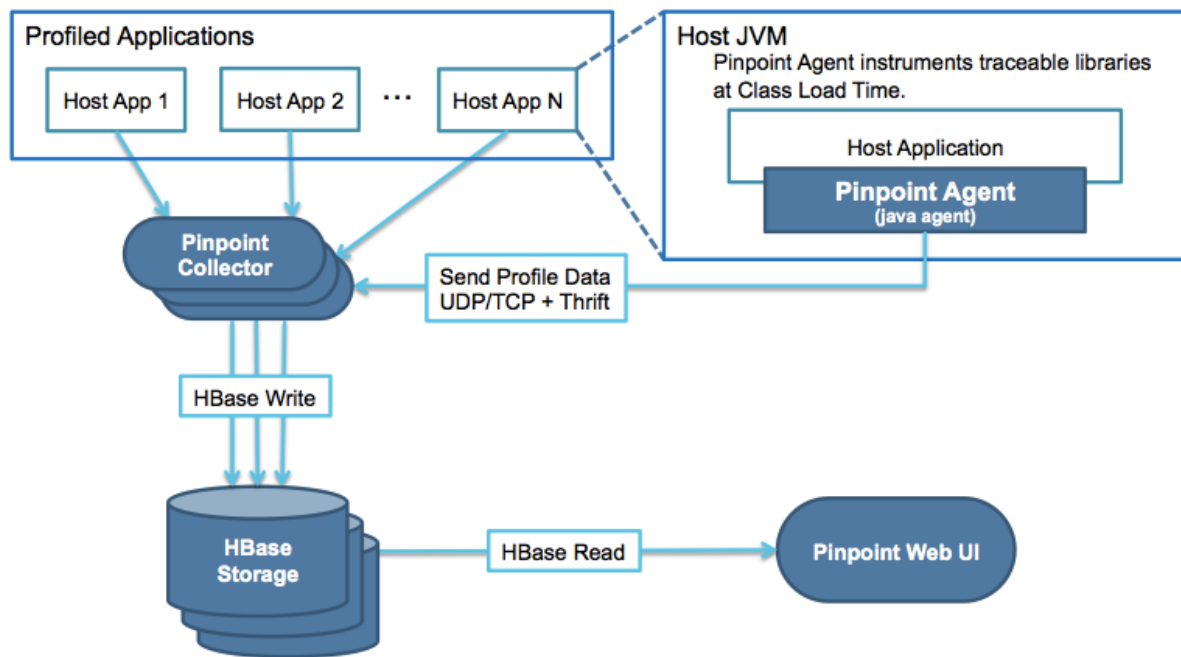
Supported Modules

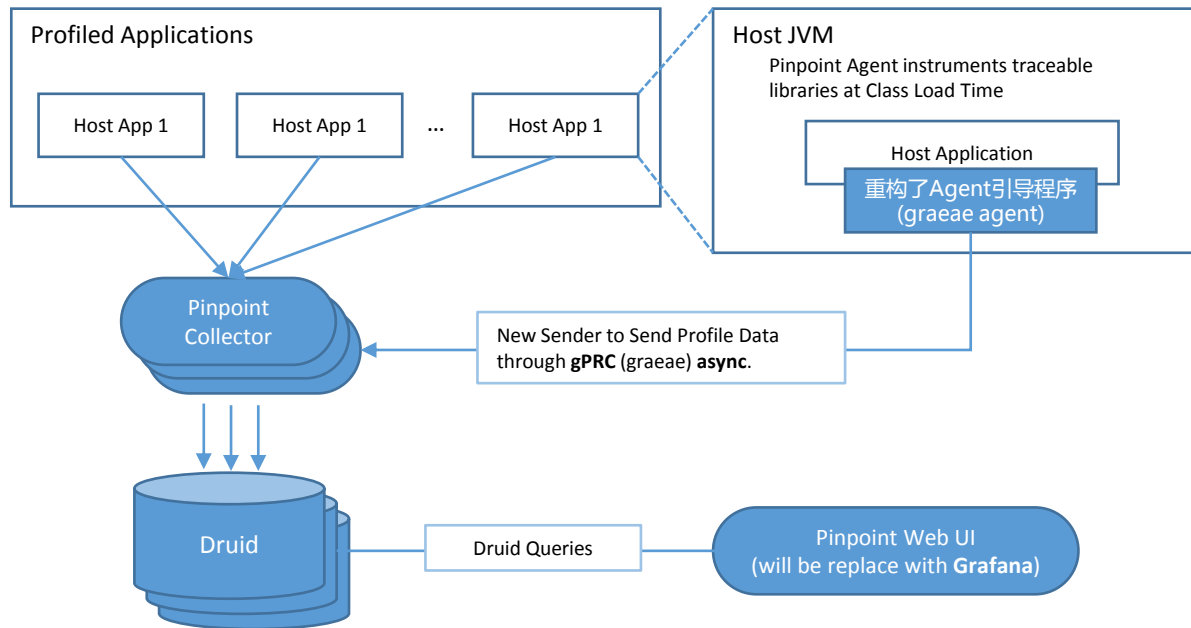
- JDK 6+
- Tomcat 6/7/8, Jetty 8/9, JBoss EAP 6, Resin 3/4
- Spring, Spring Boot
- Apache HTTP Client 3.x/4.x, JDK HttpConnector, GoogleHttpClient, OkHttpClient, NingAsyncHttpClient
- Thrift Client, Thrift Service, DUBBO PROVIDER, DUBBO CONSUMER
- MySQL, Oracle, MSSQL, CUBRID, DBCP, POSTGRESQL, MARIA
- Arcus, Memcached, Redis, CASSANDRA
- iBATIS, MyBatis
- gson, Jackson, Json Lib
- log4j, Logback

Third party agents/plugins

These include agents and plugins that are not merged into this repository.
Take a look at them if you are interested and would like to help out.

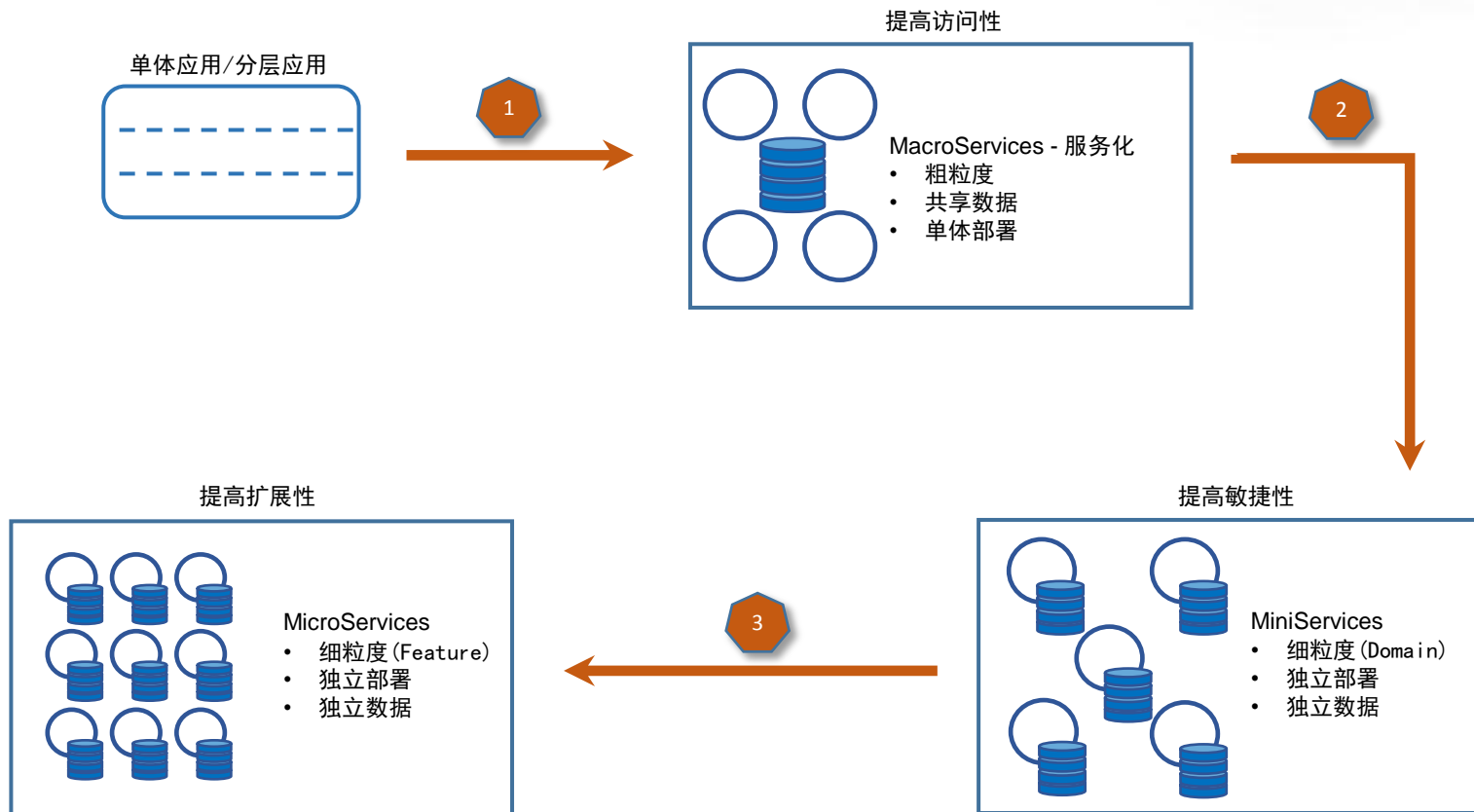
- Agents
 - NodeJS agent - <https://github.com/peaksnail/pinpoint-node-agent>
- Plugins
 - Websphere - https://github.com/sjmittal/pinpoint/tree/cpu_monitoring_fix/plugins/websphere
 - Hystrix - <https://github.com/jiaqifeng/pinpoint/tree/dev-hystrix-plugin/plugins/hystrix>
 - Kafka - <https://github.com/jiaqifeng/pinpoint/tree/dev-plugin-kafka/plugins/kafka>
 - RabbitMQ - https://github.com/jiaqifeng/pinpoint/tree/dev_plugin_rabbitmq/plugins/rabbitmq
 - RocketMQ - <https://github.com/ruizlake/pinpoint/tree/master/plugins/rocketmq>





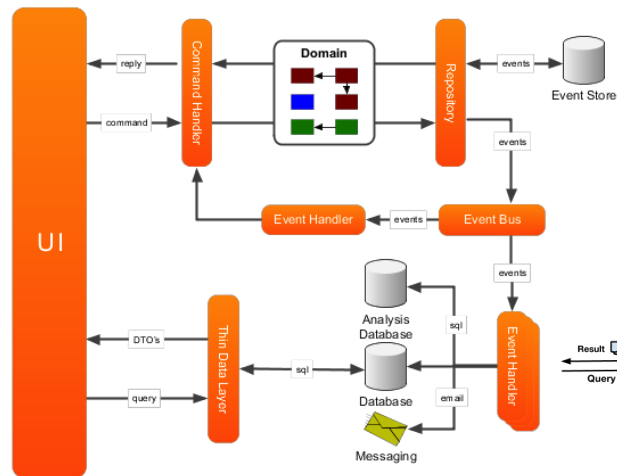
- 通信机制异步化调整
- 探针引导程序premain改造
- gRPC探针



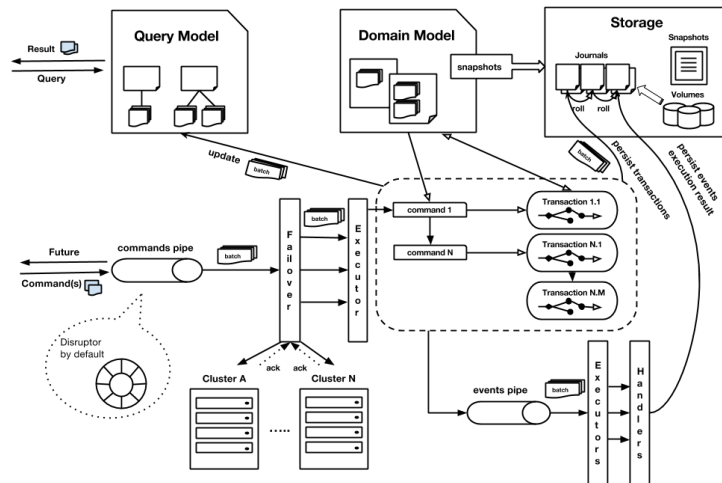




CQRS and Event Sourcing



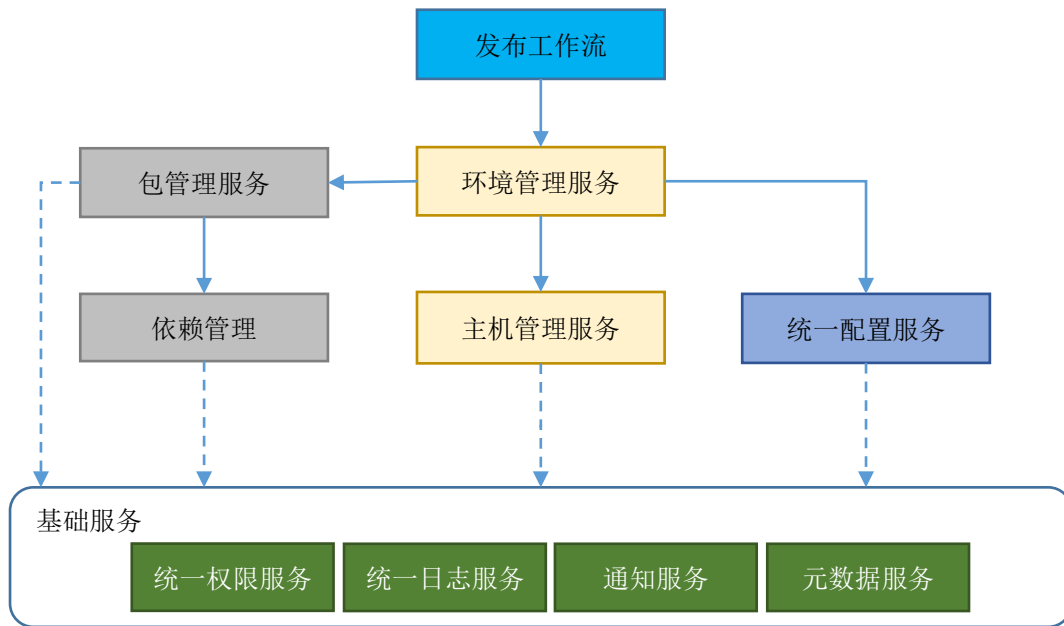
Axon Framework



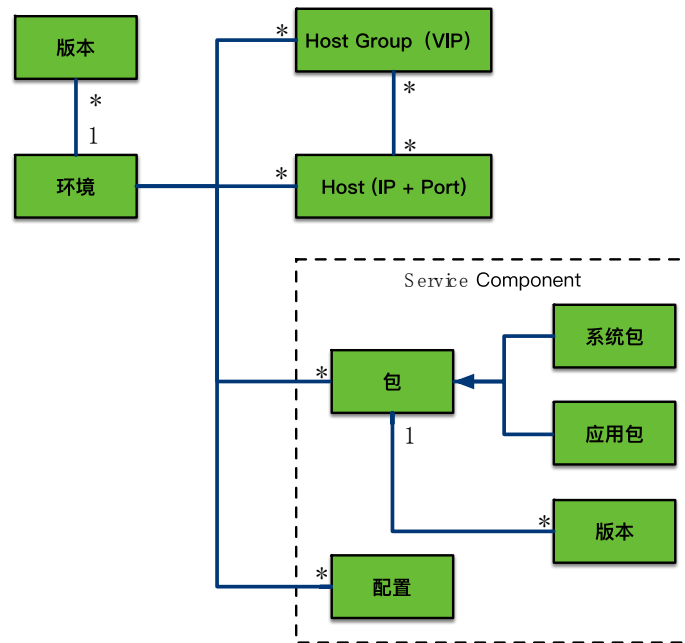
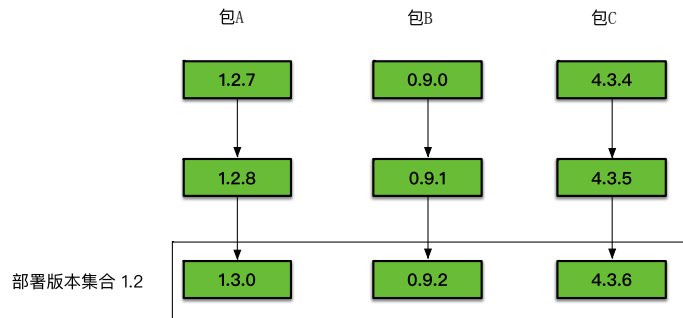
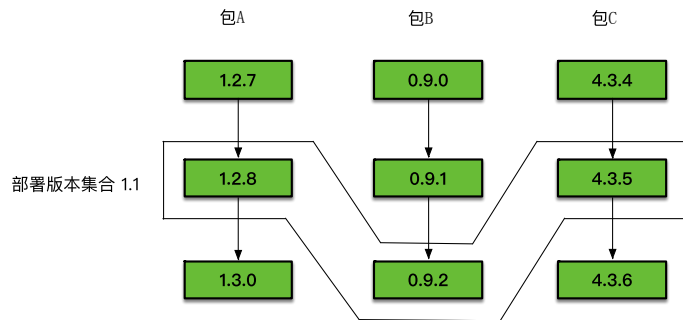
Reveno

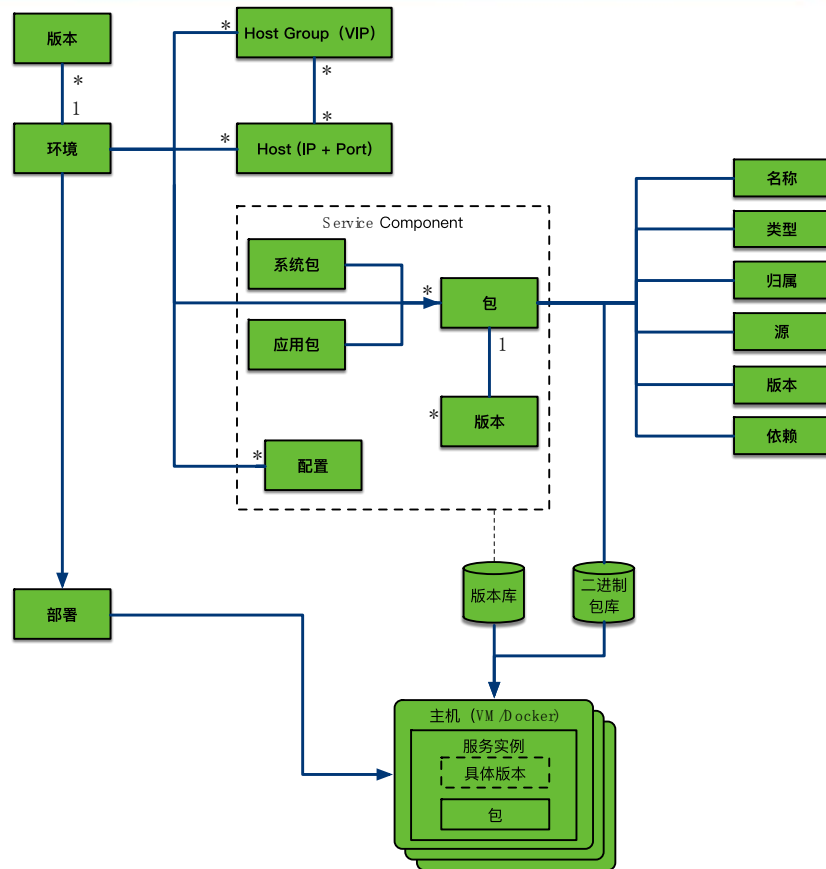
- 融数数据基于gRPC的微服务框架介绍
- 融数数据微服务治理和监控
- 基于**K8S**的微服务构建、开发和部署
- 打造适合微服务的技术团队

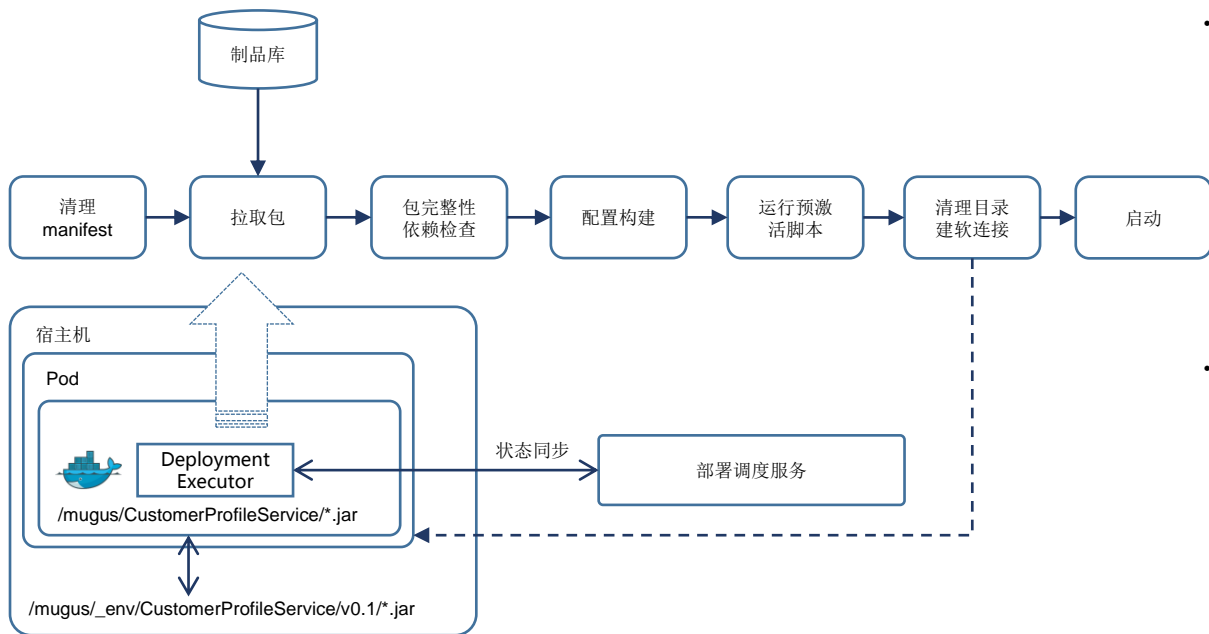
融数部署平台构建在kuberntes上，而kuberntes搭建在openstack私有云环境上



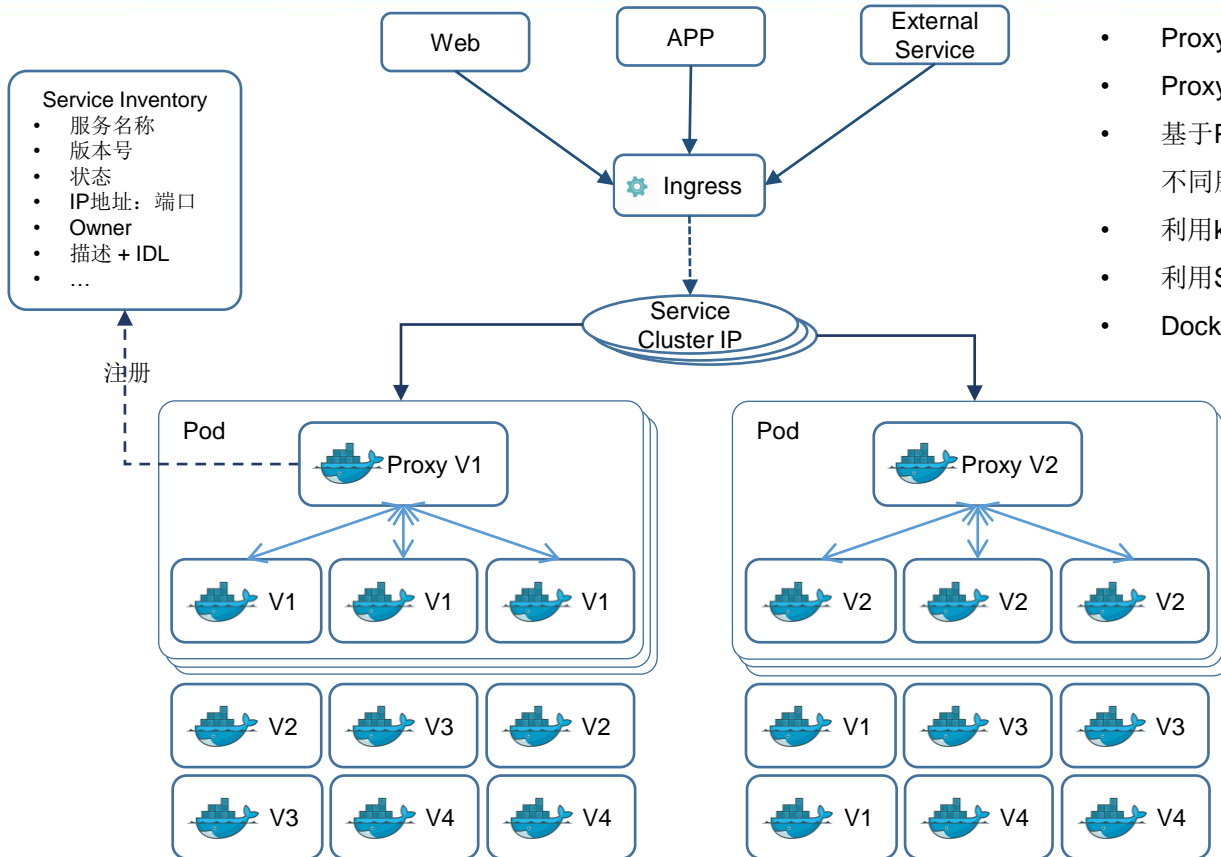
- 抽象包的概念
 - 构建包 – 描述代码、依赖和配置
 - 部署包 – 不同构建包的组合，描述每次部署所需要的不同包的版本的集合
- 版本化一切 – 编译、构建、测试、部署同源，方便回滚
 - 代码版本
 - 构建版本
 - 发布版本
 - 配置版本
- 逻辑环境和物理环境 – 一次构建，多次部署
 - 逻辑环境描述部署需要的包、配置以及所需资源的MANIFEST
 - 物理环境适配不同的主机环境（容器或者虚拟机），从而承载逻辑构建





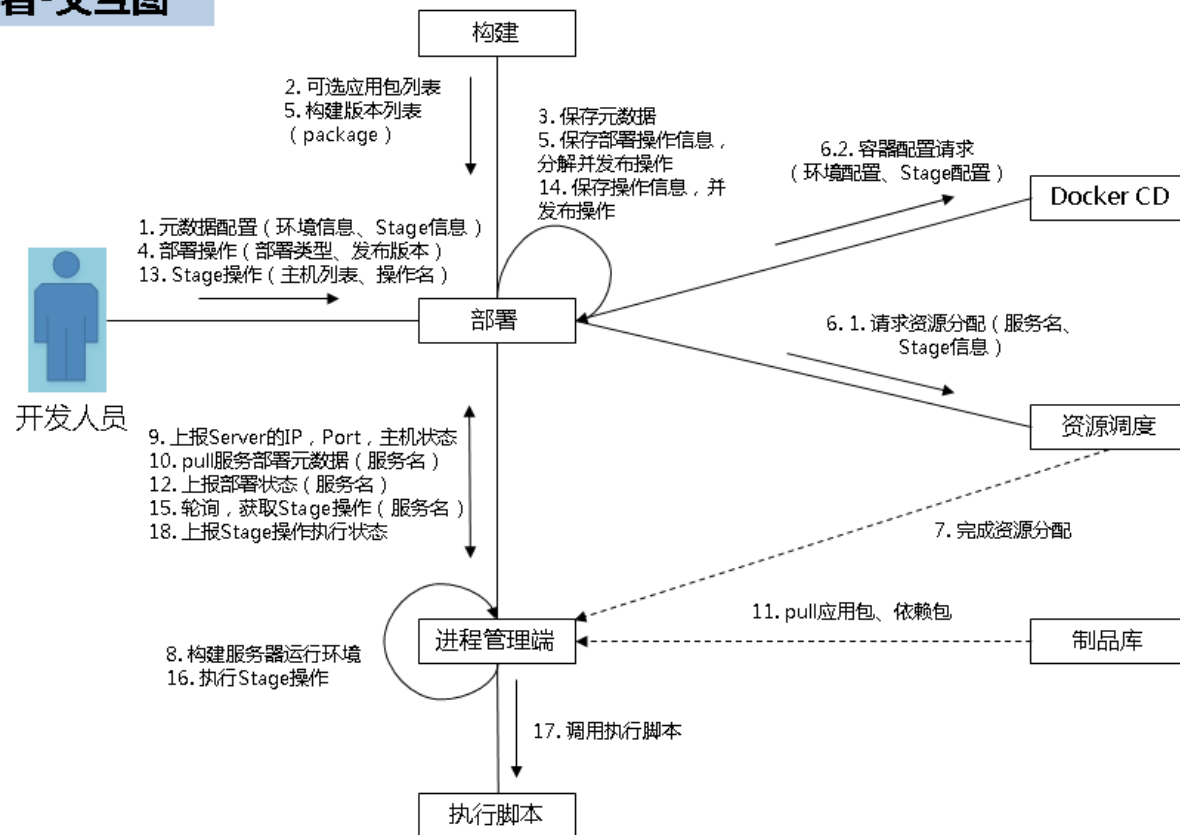


- 服务部署结构
 - Pod包含proxy和service endpoint instances
 - 通过service的cluster ip作为vip对外提供统一出口
 - 同一个服务可以部署在不同的pod上，共享数据库
 - 本地包缓存
 - 包的checksum
 - 在docker中的包通过宿主机共享目录存储
 - 不构建和部署的docker镜像，而只是通过k8s拉起基础docker镜像
- 使用Process Manager控制部署过程并报告部署状态
 - 构建于基础镜像之内
 - 过程分为
 - manifest清理，确保每个包都是最新的
 - pull包
 - 包完整性和依赖检查
 - 根据配置元数据构建配置
 - 运行预激活脚本
 - 删除并重建环境对应的目录结构，确保部署使用新的包集合
 - 启动程序



- Proxy是基于Netty的反向代理，是Client的访问入口点
- Proxy负责服务实例的信息收集并向Service Inventory注册
- 基于Proxy的路由功能结合语义化版本(X.Y.Z)的概念进行不同服务的版本管理
- 利用k8s简化部署
- 利用Service绑定(Virtual IP)来简化客户端寻址
- Docker通过挂载宿主机目录，增量拉取部署包

部署-交互图



← → ↻ ⚠ 不安全 https://k8s.in.dataengine.com/#/pod/msa/customer-profile-service-2811302657-whxsm?namespace=msa

⌵ **kubernetes** Workloads > Pods > customer-profile-service-2811302657-whxsm EDIT DELETE + CREATE

⌵ **kubernetes** Services and discovery > Services + CREATE

Admin

- Namespaces
- Nodes
- Persistent Volumes

Namespace

- msa**

Workloads

- Deployments
- Replica Sets
- Replication Controllers
- Daemon Sets
- Stateful Sets
- Jobs
- Pods

Services and discovery

- Services**
- Ingresses

Storage

- Persistent Volume Claims

Config

- Secrets
- Config Maps

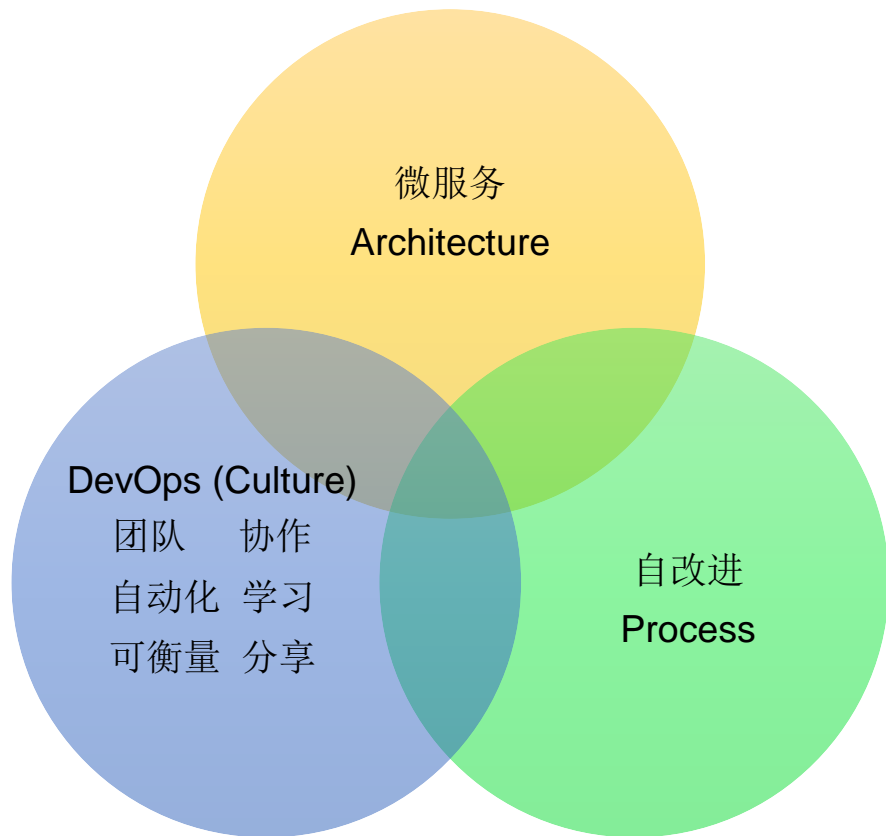
Services

Name	Labels	Cluster IP	Internal endpoints	External endpoints
✓ customer-profile-service-gateway	proxy: customer-profile-service	10.105.4.208	customer-profile-service.msa:10010 TCP customer-profile-service.msa:31759 TCP customer-profile-service.msa:12306 TCP customer-profile-service.msa:30208 TCP	-

Created by

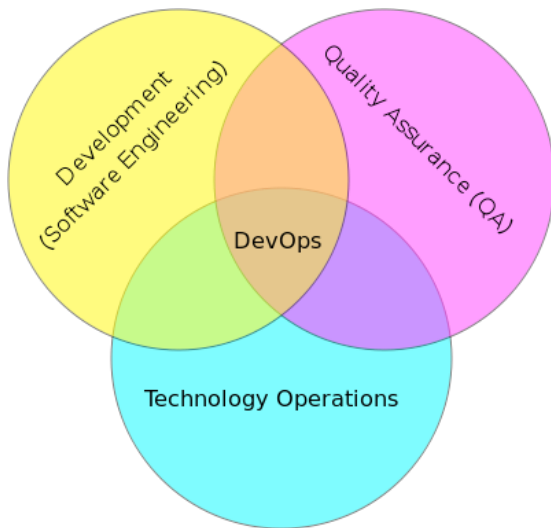
Name	Kind	Labels	Pods	Age	Images
✓ customer-profile-service-2811302657	Replica Set	pod-template-hash: 2811302657 proxy: customer-profile-service	1 / 1	9 days	msa-deploy/proxy:v0.1 msa-deploy/graeae:v0.1 msa-deploy/graeae:v0.1 msa-deploy/graeae:v0.1

- 融数数据基于gRPC的微服务框架介绍
- 融数数据微服务治理和监控
- 基于K8S的微服务构建、开发和部署
- 打造适合微服务的技术团队



微服务不仅仅是技术架构，
更是一种文化和自改进的交付模式，DevOps是微服务的基础

- 微服务的主要目的是为了敏捷
- 微服务表达了原子核心业务（Feature），但是也带了了新的挑战-将单体应用的复杂性从程序内部转移到了服务组件之间
- 因此，根据Martin Fowler提出的观点，微服务需要具备以下三个重要能力：
 - 硬件资源是否能够快速到位 – 环境管理的能力
 - 是否具备了微服务监控能力 – 分布式监控能力
 - 是否具有快速的开发部署流程 – 敏捷成熟度和自动化程度
- 服务粒度的细化，导致团队间的合作和沟通变得困难，当发生问题时，我们希望问题快速的暴露并得到迅速的解决，而DevOps是开发和运维团队的粘合剂，能够促进合作，提升解决问题的效率。



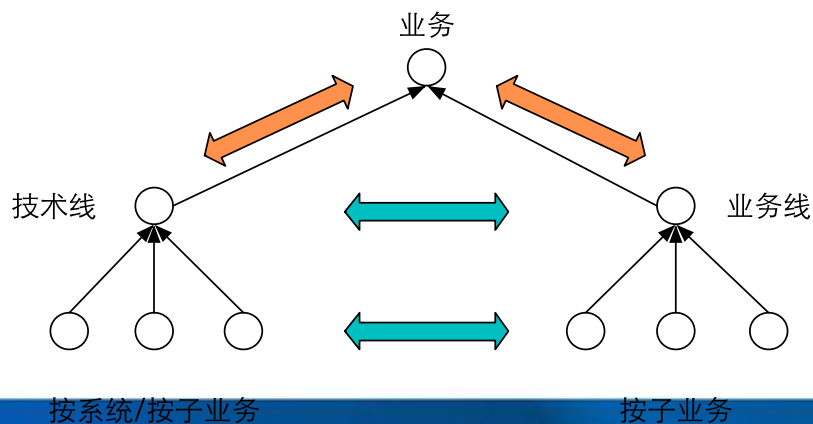
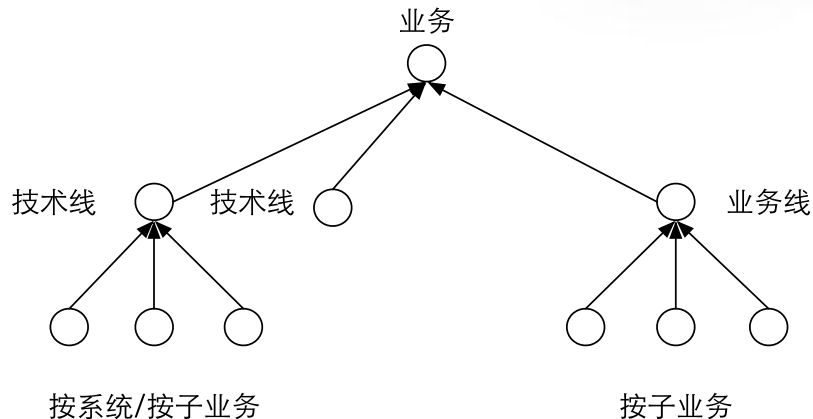
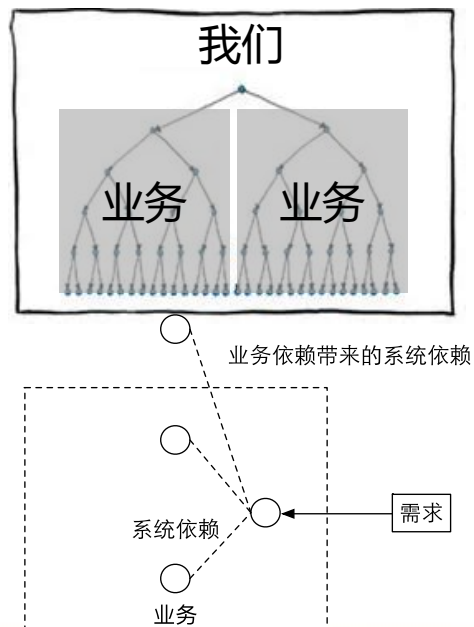
DevOps is CAMS

Culture
Automation
Measurement
Sharing

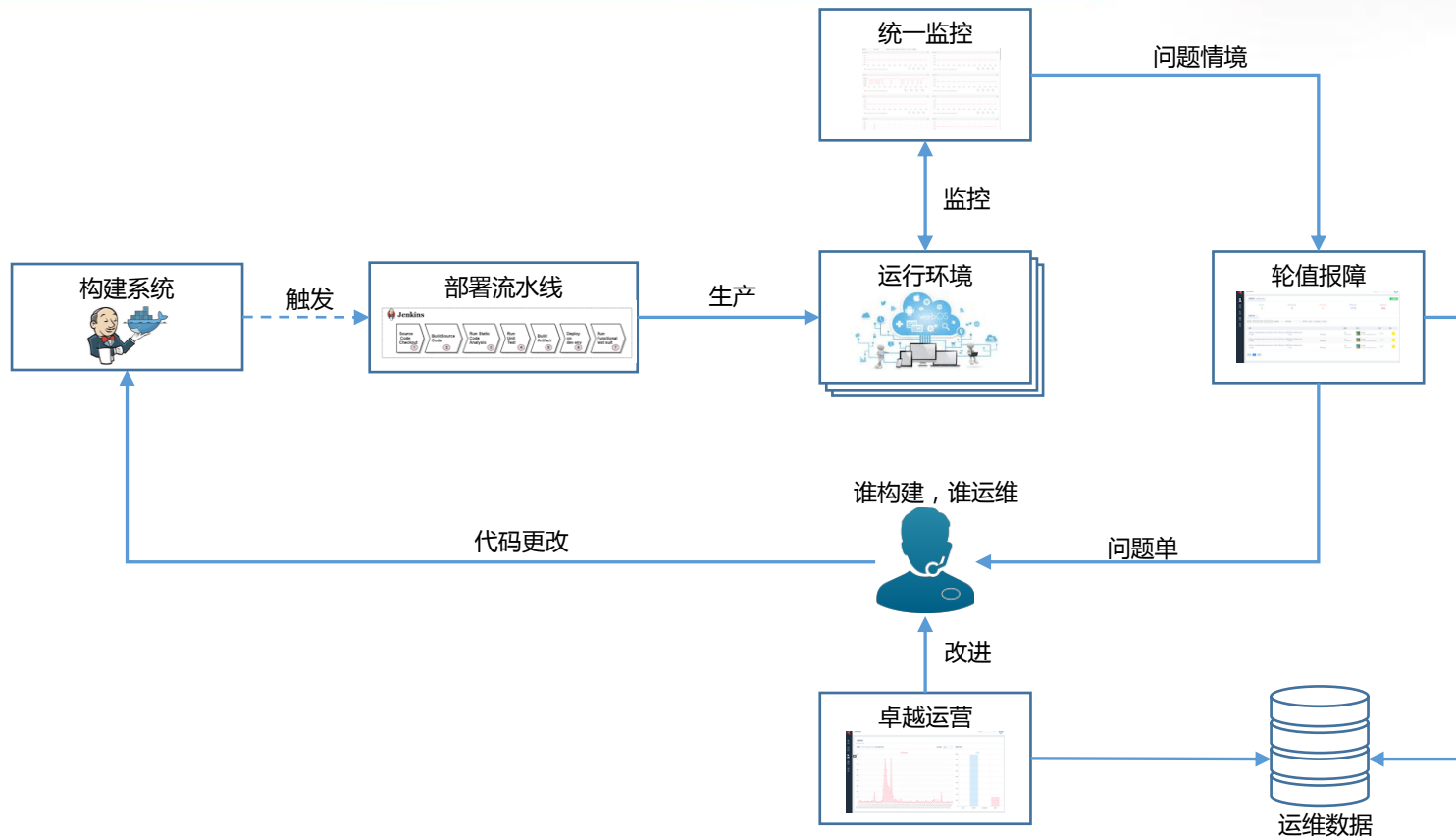
Culture Over Tools

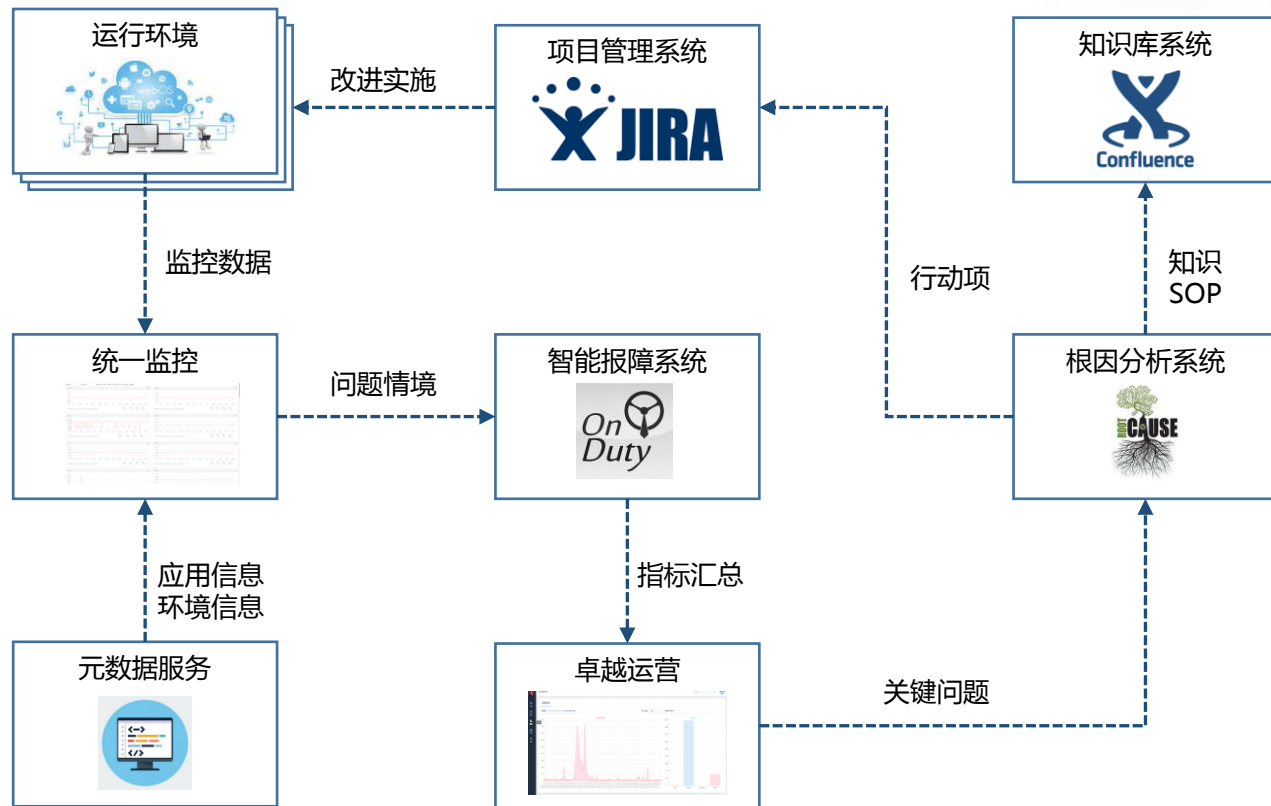
DevOps works for startups, but enterprises need it
more

- 业务驱动的团队划分
- 业务团队和技术团队水平沟通
- 向上汇报
- 业务协调，解决依赖



- 主人翁意识 (Ownership)
- 行动力 (Bias for Action)
- 吃自己的狗粮 (Eat your dog food)
 - 工程师负责从需求调研、设计、开发、测试、部署、维护、监控、功能升级等一系列的工作，也就是说软件**工程师负责服务的全生命周期的所有工作**
 - **运维是团队成员的第一要务**，在强大的自动化运维工具的支撑下，软件工程师必须负责服务或者应用的 SLA
- 让开发人员参与架构设计，而不是架构师参与开发
 - 研发人员是Owner，对业务和团队负责
 - 强调抽象和简化，将复杂的问题分解成简单的问题，并有效解决，防止过度设计
 - 鼓励用新技术解决问题，但强调掌控力





遇到问题，迅速跟进、快速解决

- 定制 SLA
- 7X24轮值
- 时效监控
- 多渠道通知
- 上报机制



在业务增加（系统功能增加）的前提下，每年的ticket数减少10%
相同系统维护的人员减少10%



演讲完毕，谢谢大家