Docker+K8S+DevOps 微服务架构师

学神 IT 教育：从零基础到实战，从入门到精通！

联系方式：

学神 IT 教育官方网站: http://www.xuegod.cn

学神 K8S 精英学习 11 群 QQ 群: 957231097

学习顾问：小语老师　　学习顾问：边边老师　　学神微信公众号

微信扫码添加学习顾问微信，同时扫码关注学神公众号了解最新行业动态，获取更多学习资料及答疑就业服务！

第 7 章　在 k8s 中部署电商平台：对关键业务模块实现全链路监控

本节所讲内容：

实战：在 k8s 部署 Springcloud 项目：

1、部署 gateway 网关服务

2、部署 portal 前端服务

3、部署 order 订单服务

4、部署 product 产品服务

5、部署 stock 库存服务

7.1 全链路监控系统概述

7.2 介绍典型的全链路监控工具

7.3 全链路监控工具对比分析

7.4 安装 pinpoint 全链路监控服务

实战 1：在 k8s 集群中部署大型电商项目：模拟京东购物平台

实战 2：通过 Pinpoint 实现电商平台功能模块全链路监控

实战 3：介绍 Pinpoint web 界面使用技巧

实战：在 k8s 平台部署 SpringCloud 框架的电商项目：

1.在 k8s 中部署网关 Gateway 服务

1）构建镜像

[root@xuegod63 k8s]# cd /root/microservic-test/gateway-service/

[root@xuegod63 gateway-service]#docker build -t 192.168.1.62/microservice/gateway:v1 .

[root@xuegod63 gateway-service]#docker push 192.168.1.62/microservice/gateway:v1

2）部署服务

[root@xuegod63 gateway-service]# cd /root/microservic-test/k8s

修改 gateway.yaml 文件，把镜像变成 image: 192.168.1.62/microservice/gateway:v1

3）更新 yaml 文件

[root@xuegod63 gateway-service]# kubectl apply -f gateway.yaml

4）查看 pod 状态

[root@xuegod63 gateway-service]# kubectl get pods -n ms　| grep gateway

5）看到如下 running 说明 pod 运行正常

gateway-c94f4d95c-2dqvw　1/1　Running　0　ns　31s

gateway-c94f4d95c-l4jmq　1/1　Running　0　31s

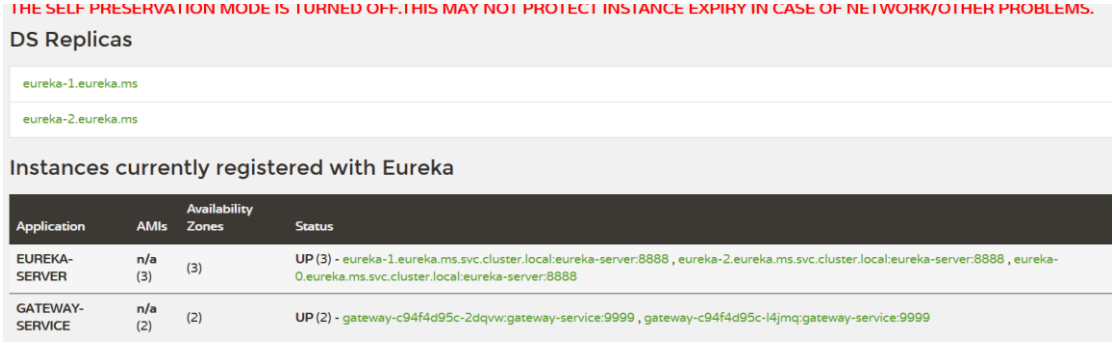
6）配置 hosts 文件

gateway 的域名是 gateway.ctnrs.com，需要在电脑找到 hosts 文件，再增加一行如下：

192.168.1.64　gateway.ctnrs.com

在浏览器访问 eureka.ctnrs.com

可看到 GATEWAY-SERVICE 已经注册到 eureka 了：



2.在 k8s 中部署前端 portal 服务

1）构建镜像

[root@xuegod63 k8s]# cd　/root/microservic-test/portal-service

[root@xuegod63 portal-service]# docker build -t

192.168.1.62/microservice/portal:v1 .

[root@xuegod63 portal-service]# docker push　192.168.1.62/microservice/portal:v1

2）部署服务

[root@xuegod63 portal-service]# cd /root/microservic-test/k8s

修改 portal.yaml 文件，把镜像变成 image: 192.168.1.62/microservice/portal:v1

3）更新 yaml 文件

[root@xuegod63 k8s]# kubectl apply -f portal.yaml

4）查看 pod 状态

[root@xuegod63 k8s]# kubectl get pods -n ms　| grep portal

看到如下 running 说明 pod 运行正常：

portal-5f59cb767c-fv67r　1/1　Running　1　5m26s

5）配置 hosts 文件

要在自己电脑找到 hosts 文件，再增加一行如下内容：

192.168.1.64　portal.ctnrs.com

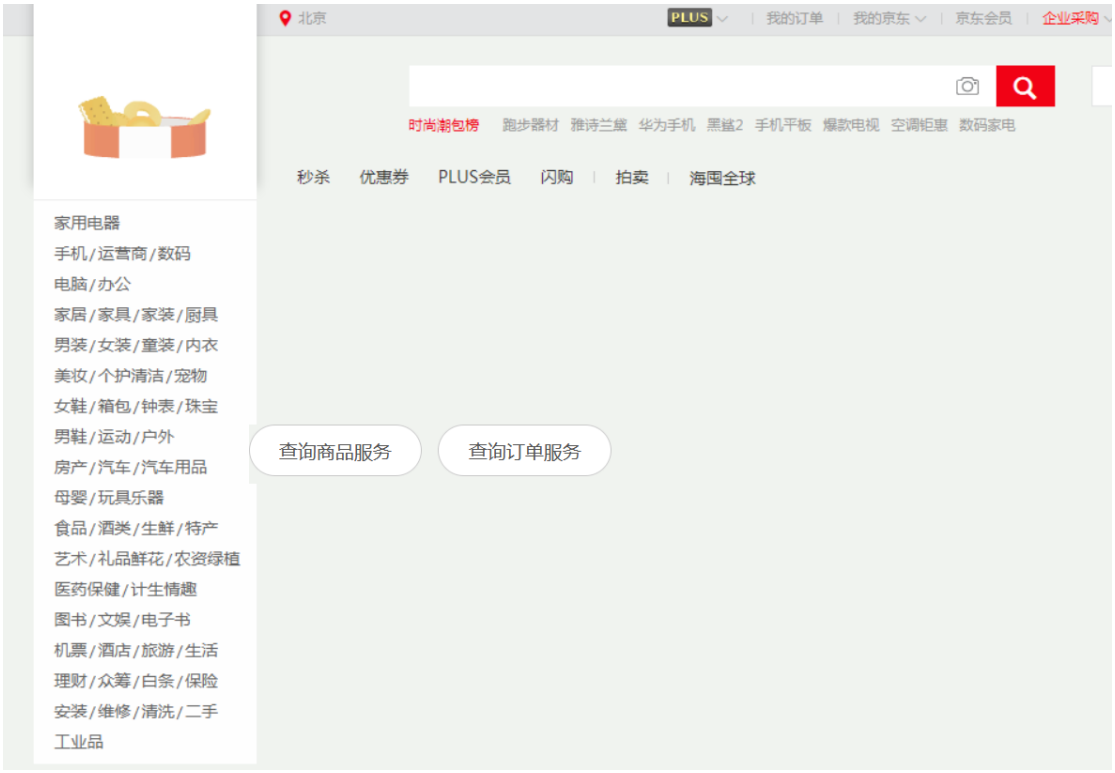
6）查看 portal 是否注册到 eureka 中

在浏览器访问 eureka.ctnrs.com 可看到 portal 服务已经注册到 eureka 了

7）访问前端页面

在浏览器访问 portal.ctnrs.com

可看到如下页面：



3.在 k8s 中部署订单 order 服务

1）构建镜像

[root@xuegod63 k8s]# cd /root/microservic-test/order-service/order-service-biz

[root@xuegod63 order-service-biz]# docker build -t 192.168.1.62/microservice/order:v1 .

[root@xuegod63 order-service-biz]# docker push 192.168.1.62/microservice/order:v1

2）部署服务

[root@xuegod63 order-service-biz]# cd /root/microservic-test/k8s

修改 order.yaml 文件，把镜像变成 image: 192.168.1.62/microservice/order:v1

3）更新 yaml 文件

[root@xuegod63 k8s]# kubectl apply -f order.yaml

4）查看 pod 状态

[root@xuegod63 k8s]# kubectl get pods -n ms | grep order

看到如下 running 说明 pod 运行正常：

order-75d9c4bbd7-f6lhx 1/1 Running 0 3m10s

4.在 k8s 中部署产品 product 服务

1）构建镜像

[root@xuegod63 k8s]# cd /root/microservic-test/product-service/product-service-biz

[root@xuegod63 product-service-biz]# docker build -t 192.168.1.62/microservice/product:v1 .

[root@xuegod63 product-service-biz]# docker push 192.168.1.62/microservice/product:v1

2）部署服务

[root@xuegod63 product-service-biz]# cd /root/microservic-test/k8s

修改 product.yaml 文件，把镜像变成 image: 192.168.1.62/microservice/product:v1

3）更新 yaml 文件

[root@xuegod63 k8s]# kubectl apply -f product.yaml

4）查看 pod 状态

[root@xuegod63 k8s]# kubectl get pods -n ms | grep product

看到如下 running 说明 pod 运行正常：

product-775f5f74d9-4nssp 1/1 Running 0 18s

5.在 k8s 中部署库存 stock 服务

1）构建镜像

[root@xuegod63 k8s]# cd /root/microservic-test/stock-service/stock-service-biz

[root@xuegod63 stock-service-biz]# docker build -t 192.168.1.62/microservice/stock:v1 .

[root@xuegod63 stock-service-biz]# docker push 192.168.1.62/microservice/stock:v1

2）部署服务

[root@xuegod63 stock-service-biz]# cd /root/microservic-test/k8s

修改 stock.yaml 文件，把镜像变成 image: 192.168.1.62/microservice/stock:v1

3）更新 yaml 文件

[root@xuegod63 k8s]# kubectl apply -f stock.yaml

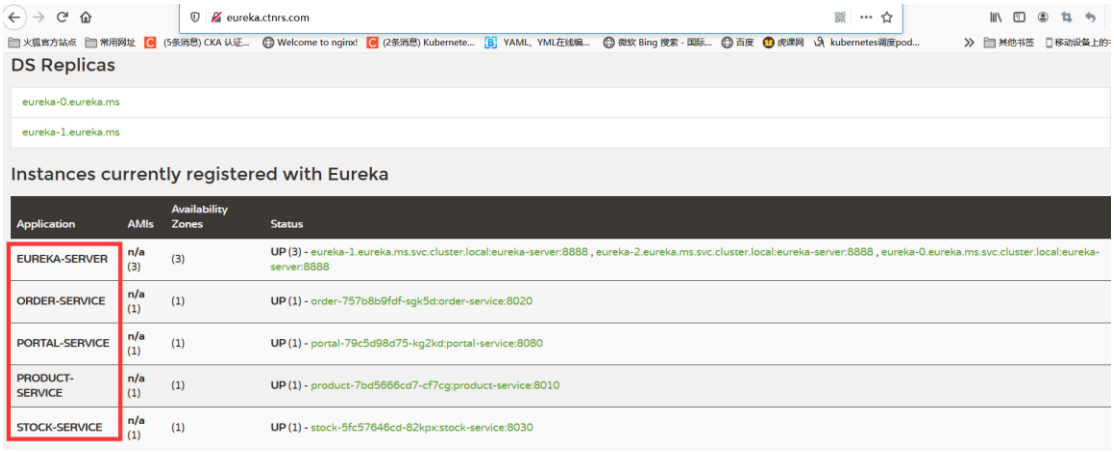
4）查看 pod 状态

[root@xuegod63 k8s]# kubectl get pods -n ms | grep stock

看到如下 running 说明 pod 运行正常：

stock-984756d8d-ss8tq 1/1 Running 0 26s

上面都部署成功之后，在浏览器访问 eureka.ctnrs.com 可看到 gateway、portal、product、order、stock 等服务都已经注册到 eureka 了



在浏览器访问 portal.ctnrs.com 登陆前端页面，可看到如下内容：

点击查询商品服务，出现如下：

学神 IT 教育官方 QQ 群：957231097 或唐老师 QQ：3340273106 领取更多资料

选择手机，点击购买，然后再点击查询订单服务，出现如下：

互动 1：　什么是全链路监控?

全链路性能监控从整体维度到局部维度展示各项指标，将跨应用的所有调用链性能信息集中展现，可方便度量整体和局部性能，并且方便找到故障产生的源头，生产上可极大缩短故障排除时间。

互动 2：　全链路监控跟传统的 prometheus 监控有何区别?

Prometheus 监控解决了基本指标和报警问题（cpu、内存、磁盘、网络流量等指标），全链路监控的解决链路追踪的问题，两者各司其职，是互相的补充。

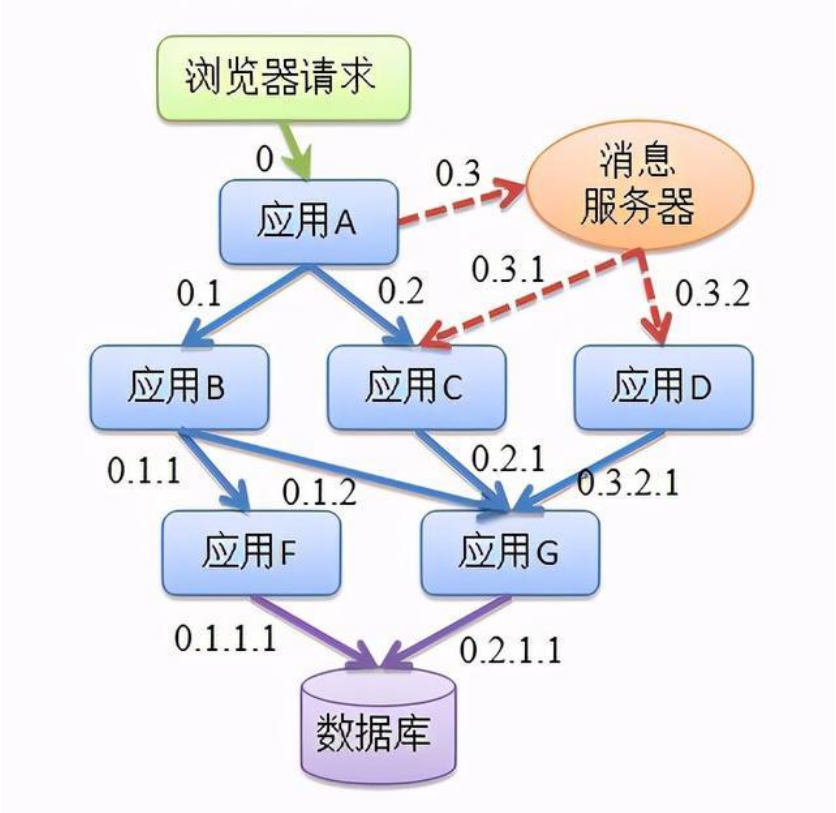
互动 3：　为什么要进行全链路监控?

随着服务架构的流行，尤其是 k8s 的大量使用，导致各服务间的调用关系越来越复杂，一次请求往往需要涉及到多个服务，这些服务有可能是由不同的团队开发、可能使用不同的编程语言来实现、有可能分布在了几千台服务器，横跨多个不同的数据中心。分布式部署架构带来的问题就会迅速凸显出来。尤其线上出现问题，不知道如何排查，问题出现在哪个服务？如何快速定位问题？如何跟踪业务调用链路？于是就有了全链路监控

7.1　全链路监控系统概述

7.1.1　什么是全链路监控系统?

在分布式微服务架构中，系统为了接收并处理一个前端用户请求，需要让多个微服务应用协同工作，其中的每一个微服务应用都可以用不同的编程语言构建，由不同的团队开发，并可以通过多个对等的应用实例实现水平扩展，甚至分布在横跨多个数据中心的数千台服务器上。单个用户请求会引发不同应用之间产生一串顺序性的调用关系，如果要对这些调用关系进行监控，了解每个应用如何调用，这就产生了全链路监控。



7.1.2 为什么要进行全链路监控?

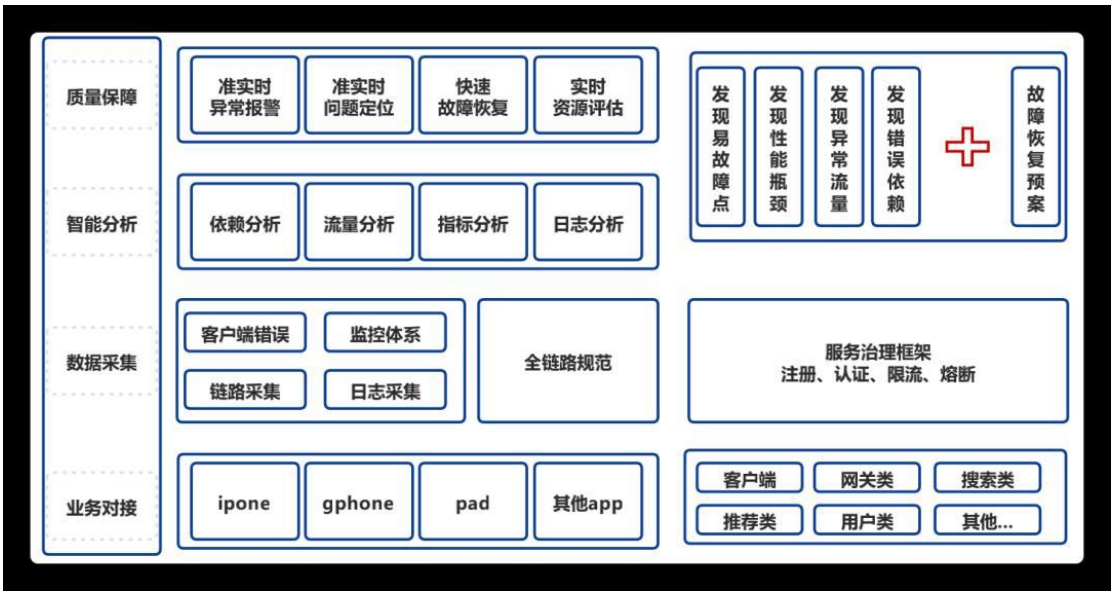
在微服务架构中，服务会被拆分成多个模块，这些模块可能由不同的开发团队开发、维护，也可能使用不同的编程语言来实现、也有可能分布在多台服务器上，由于服务的拆分，单个用户的请求会经过多个微服务，相互之间形成复杂的调用关系，传统的监控手段已经不能实现如此复杂的链路之间的监控了，因此，就需要一些可以帮助理解系统行为、用于分析性能问题的工具，以便发生故障的时候，能够快速定位和解决问题。

7.1.3 全链路监控系统可以解决哪些问题?

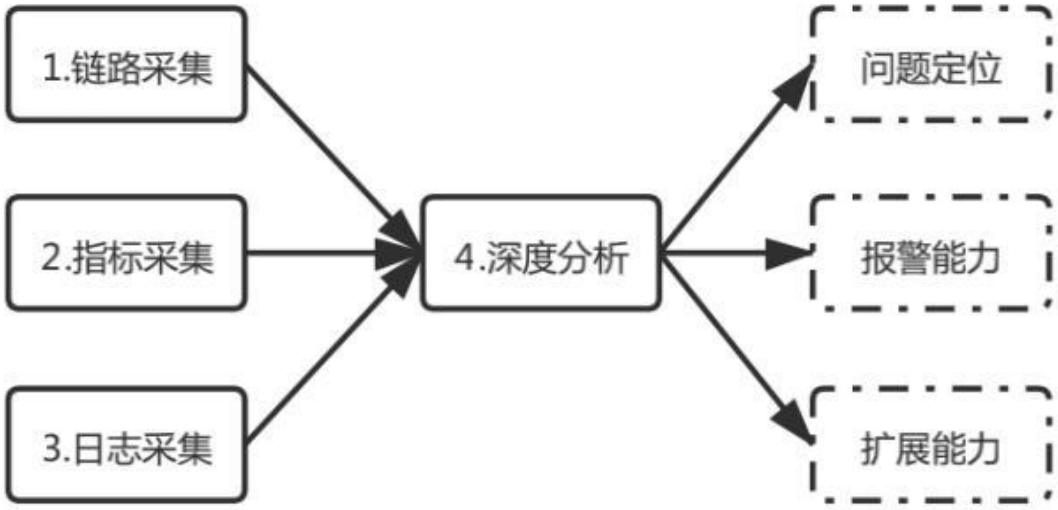
1. 请求链路追踪，故障快速定位：可以通过调用链结合业务日志快速定位错误信息。
2. 可视化： 各个阶段耗时，进行性能分析。
3. 依赖优化：各个调用环节的可用性、梳理服务依赖关系以及优化。
4. 数据分析，优化链路：可以得到用户的行为路径，汇总分析应用在很多业务场景。

7.1.4 全链路监控可监控哪些指标?

全链路监控的四部分：链路采集、指标采集、日志采集、深度分析



链路采集包括调用链和服务拓扑，是全链路分析的串联器。
指标采集整合到服务链路上，使全链路具备基础监控能力。
日志采集的数据源，也是全链路分析的数据源。
深度分析包括离线、在线模块，满足全链路的问题定位需求。



在微服务架构中，不同维度有不同的监控方式。

（1）健康检查。健康检查是对应用本身健康状况的监控，检查服务是否还正常存活。

（2）日志。日志是排查问题的主要方式，日志可以提供丰富的信息用于定位和解决问题。

（3）调用链监控。调用链监控可以完整的呈现出一次请求的全部信息，包括服务调用链路、所耗时间等。

（4）指标监控。指标是一些基于时间序列的离散数据点，通过聚合和计算后能反映出一些重要指标的趋势。

在上述 4 种监控方式中，健康检查是云平台等基础设施提供的能力，日志则一般有单独的日志中心进行日志的采集、存储、计算和查询，调用链监控一般也有独立的解决方案进行服务调用的埋点、采集、计算和查询。

指标监控选择 Prometheus 的主要原因：

（1）成熟的社区支撑。Prometheus 是一个开源的监控软件，拥有活跃的社区，能够很好地与云原生环境搭配。

（2） 易于部署和运维。Prometheus 核心只有一个二进制文件，没有其他的第三方依赖，部署运维均十分方便。

（3）采用 Pull 模型，通过 HTTP 的 Pull 方式从各个监控目标拉取监控数据。Push 模型一般通过 Agent 方式去采集信息并推送到收集器中，每个服务的 Agent 都需要配置监控数据项与监控服务端的信息，在大量服务时会加大运维难度；另外，采用 Push 模型，在流量高峰期间监控服务端会同时接收到大量请求和数据，会给监控服务端造成很大压力，严重时甚至服务不可用。

（4）强大的数据模型。Prometheus 采集到的监控数据均以指标的形式存在于内置的时序数据库中，除了基本的指标名称外，还支持自定义的标签。通过标签可以定义出丰富的维度，方便进行监控数据的聚合和计算。

（5）强大的查询语言 PromQL。通过 PromQL 可以实现对监控数据的查询、聚合、可视化、告警。

（6）完善的生态。常见的操作系统、数据库、中间件、类库、编程语言，Prometheus 都提供了接入方案，并且提供了 Java/Golang/Ruby/Python 等语言的客户端 SDK，能够快速实现自定义的监控逻辑。

（7）高性能。Prometheus 单一实例即可处理数以百计的监控指标，每秒处理数十万的数据，在数据采集和查询方面有着优异的性能表现。

注意：查看爱奇艺全链路自动化监控平台的探索与实践的落地

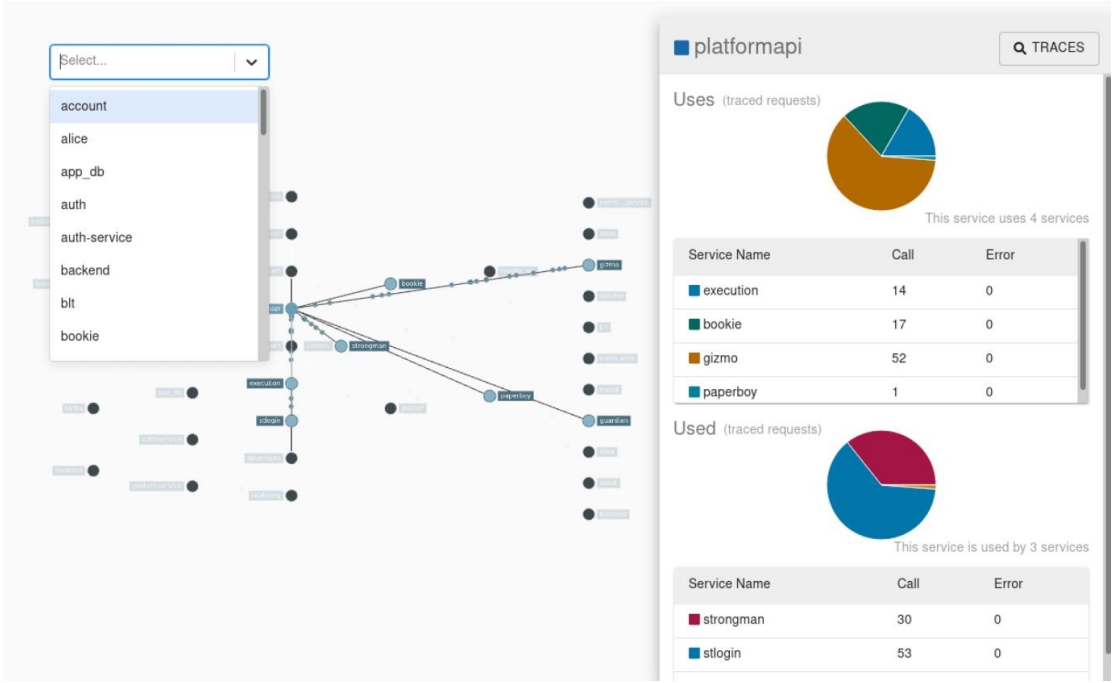
7.2 常见的全链路监控工具有哪些?

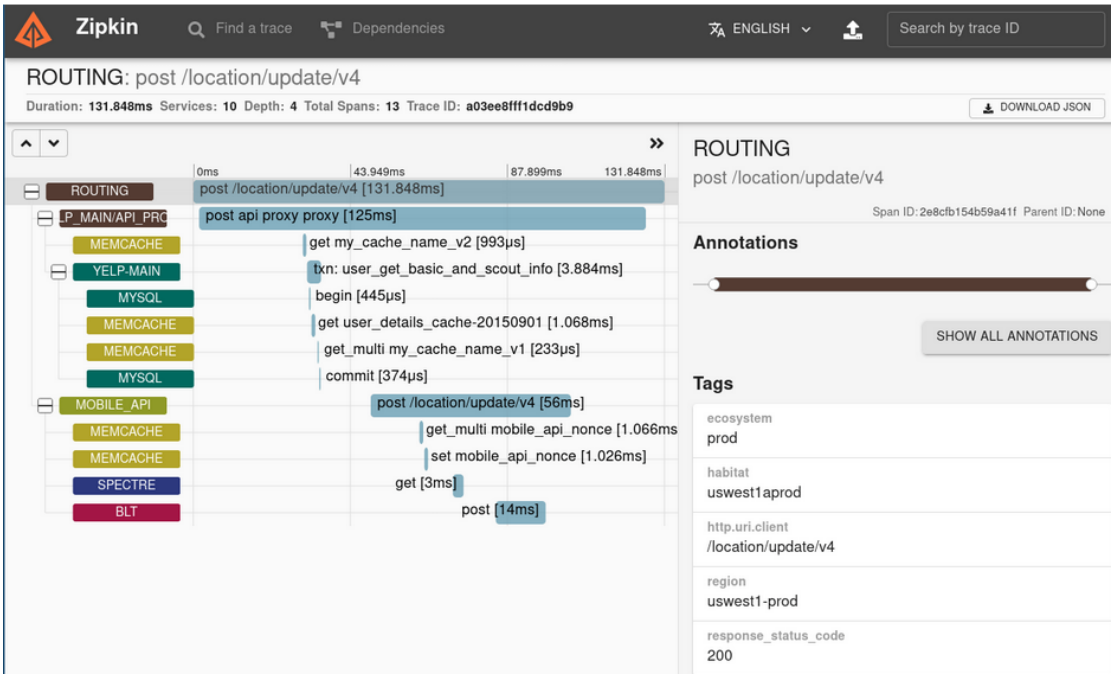
7.2.1 zipkin

github：
https://github.com/openzipkin/zipkin

zipkin 是一个分布式的追踪系统，它能够帮助你收集服务架构中解决问题需要的时间数据，功能包括收集和查找这些数据。如果日志文件中有跟踪 ID，可以直接跳转到它。否则，可以根据服务、操作名称、标记和持续时间等属性进行查询。例如在服务中花费的时间百分比，以及哪些环节操作失败。特点是轻量，使用部署简单。

zipkin 还提供了一个 UI 界面，它能够显示通过每个应用程序的跟踪请求数。这有助于识别聚合行为，包括错误路径或对不推荐使用的服务的调用。





应用程序需要“检测”才能向 Zipkin 报告跟踪数据。这通常意味着需要配置一个用于追踪和检测的库。最流行的向 Zipkin 报告数据的方法是通过 http 或 Kafka，尽管还有许多其他选项，如 apache，activemq、gRPC 和 RabbitMQ。提供给 UI 存储数据的方法很多，如存储在内存中，或者使用受支持的后端（如 apachecassandra 或 Elasticsearch）持久存储。

7.2.2 skywalking

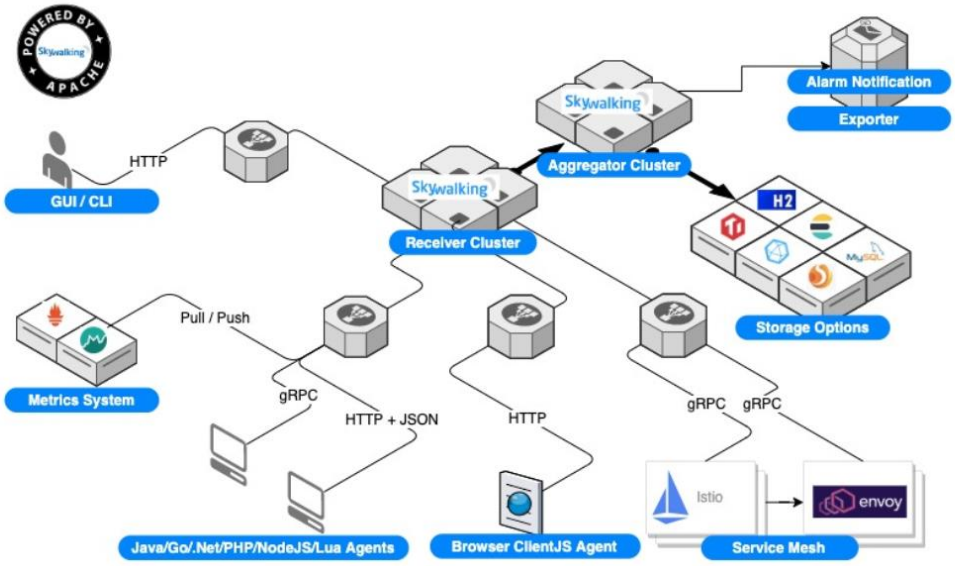
github:

https://github.com/apache/incubator-skywalking

skywalking 是本土开源的调用链追踪系统，包括监控、跟踪、诊断功能，目前已加入 Apache 孵化器，专门为微服务、云本地和基于容器（Docker、Kubernetes、Mesos）架构设计。

主要功能如下：

1）服务、服务实例、端点指标数据分析
2）根本原因分析，在运行时评测代码
3）服务拓扑图分析
4）服务、服务实例和端点依赖性分析
5）检测到慢速服务和终结点
6）性能优化
7）分布式跟踪和上下文传播
8）数据库访问度量。检测慢速数据库访问语句（包括 SQL 语句）。
9）报警
10）浏览器性能监视



7.2.3 pinpoint

github:

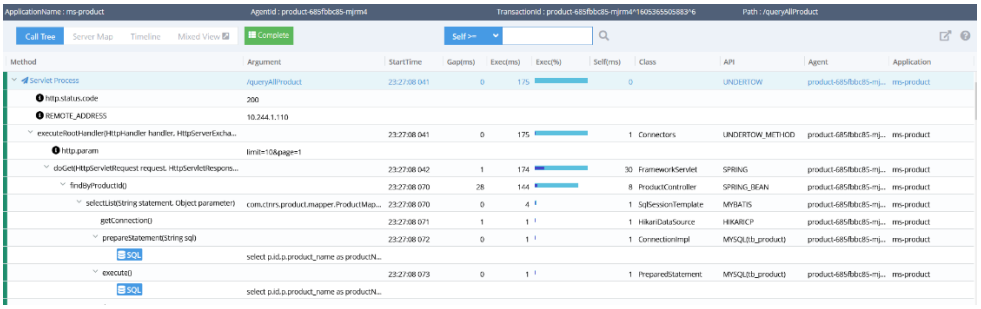
https://github.com/naver/pinpoint

pinpoint 是韩国人开源的基于字节码注入的调用链分析，以及应用监控分析工具。Pinpoint 提供了一个解决方案，可以帮助分析系统的整体结构，以及通过跟踪分布式应用程序中的事务来分析其中的组件是如何相互连接的。

功能如下：

1）一目了然地了解应用程序拓扑
2）实时监视应用程序
3）获得每个事务的代码级可见性
4）安装 APM 代理程序，无需更改一行代码
5）对性能的影响最小（资源使用量增加约 3%）

Pinpoint 的可视化 UI 界面：



7.3 全链路监控工具对比分析

市面上的全链路监控理论模型大多都是借鉴 Google Dapper 论文，下面重点关注以下三种 APM 组件：APM = ApplicationPerformance Management,中文即应用性能管理

1）Zipkin：由 Twitter 公司开源，开放源代码分布式的跟踪系统，用于收集服务的定时数据，以解决微服务架构中的延迟问题，包括：数据的收集、存储、查找和展现。

2）Pinpoint：一款对 Java 编写的大规模分布式系统的 APM 工具，由韩国人开源的分布式跟踪组件。

3）Skywalking：国产的优秀 APM 组件，是一个对 JAVA 分布式应用程序集群的业务运行情况进行追踪、告警和分析的系统。

7.3.1 全面的调用链路数据分析

全面的调用链路数据分析，提供代码级别的可见性以便轻松定位失败点和瓶颈。

1.zipkin

zipkin 的链路监控粒度相对没有那么细，调用链中具体到接口级别，再进一步的调用信息未普及。

2.skywalking

skywalking 支持 20+的中间件、框架、类库，比如：主流的 dubbo、Okhttp，还有 DB 和消息中间件。skywalking 链路调用分析截取的比较简单，网关调用 user 服务，由于支持众多的中间件，所以 skywalking 链路调用分析比 zipkin 完备些。

3.pinpoint

pinpoint 应该是这三种 APM 组件中，数据分析最为完备的组件。提供代码级别的可见性以便轻松定位失败点和瓶颈，上图可以看到对于执行的 sql 语句，都进行了记录。还可以配置报警规则等，设置每

个应用对应的负责人，根据配置的规则报警，支持的中间件和框架也比较完备。

7.3.2 pinpoint 和 zipkin 对比分析

差异性:

1. Pinpoint 是一个完整的性能监控解决方案，有从探针、收集器、存储到 Web 界面等全套体系，而 Zipkin 只侧重收集器和存储服务，虽然也有用户界面，但其功能与 Pinpoint 不可同日而语。反而 pinpoint 提供有 Query 接口，更强大的用户界面和系统集成能力，可以基于该接口二次开发实现。

2. Zipkin 官方提供有基于 Finagle 框架（Scala 语言）的接口，而其他框架的接口由社区贡献，目前可以支持 Java、Scala、Node、Go、Python、Ruby 和 C# 等主流开发语言和框架；但是 Pinpoint 目前只有官方提供的 Java Agent 探针，其他的都在请求社区支援中。

3. Pinpoint 提供有 Java Agent 探针，通过字节码注入的方式实现调用拦截和数据收集，可以做到真正的代码无侵入，只需要在启动服务器的时候添加一些参数，就可以完成探针的部署，而 Zipkin 的 Java 接口实现 Brave（zipkin 的 java 客户端），只提供了基本的操作 API，如果需要与框架或者项目集成的话，就需要手动添加配置文件或增加代码。

4. Pinpoint 的后端存储基于 Hbase，而 Zipkin 基于 Cassandra（*Cassandra* 是一套开源分布式 NoSQL 数据库系统）。

相似性:

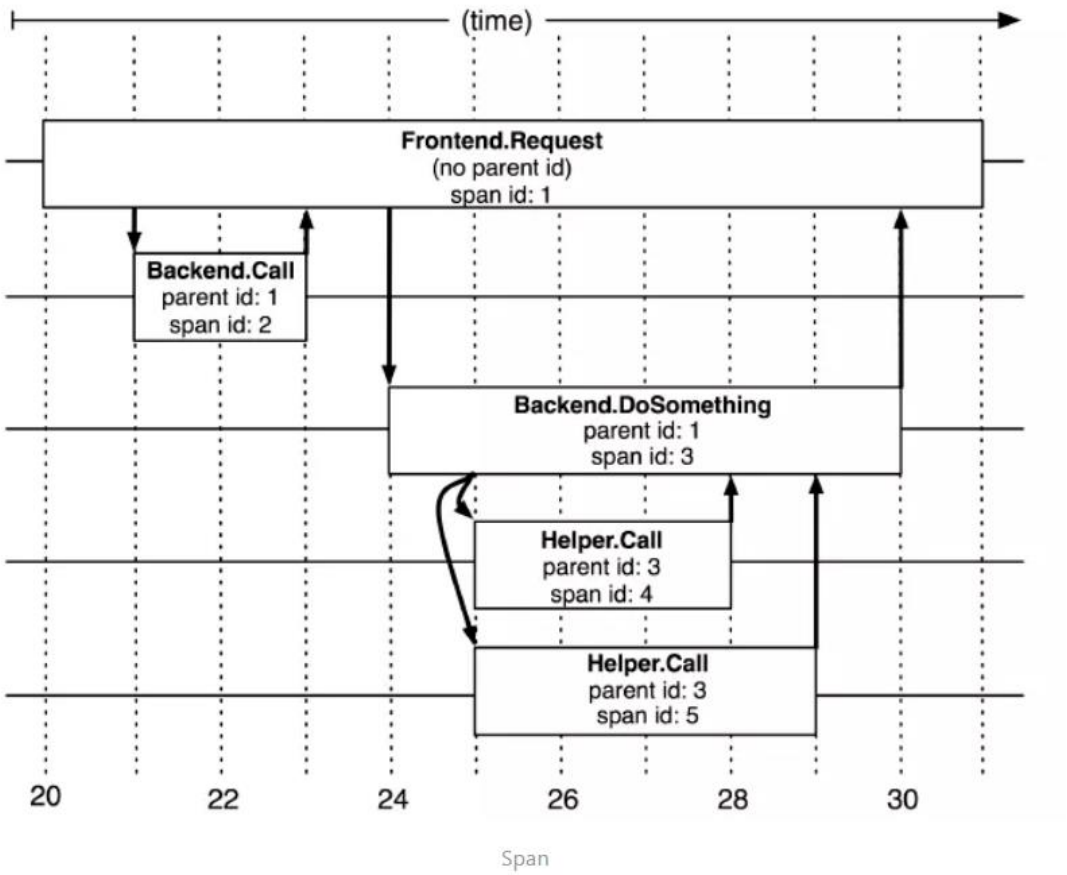
pinpoint 与 zipkin 都是基于 Google Dapper（google 的分布式追踪系统 dapper）的那篇论文，因此理论基础大致相同。两者都是将服务调用拆分成若干有级联关系的 Span，通过 SpanId 和 ParentSpanId 来进行调用关系的级联，最后再将整个调用链流经的所有的 Span 汇聚成一个 Trace，报告给服务端的 collector 进行收集和存储。

Pinpoint 所采用的概念也不完全与那篇论文一致。比如他采用 TransactionId 来取代 TraceId，而真正的 TraceId 是一个结构，里面包含了 TransactionId, SpanId 和 ParentSpanId。而且 Pinpoint 在 Span 下面又增加了一个 SpanEvent 结构，用来记录一个 Span 内部的调用细节（比如具体的方法调用等等），因此 Pinpoint 默认会比 Zipkin 记录更多的跟踪数据。但是理论上并没有限定 Span 的粒度大小，所以一个服务调用可以是一个 Span，那么每个服务中的方法调用也可以是个 Span，这样的话，其实 Brave 也可以跟踪到方法调用级别，只是具体实现并没有这样做而已。

互动:

1.span 是什么?

基本工作单元，一次链路调用（可以是 RPC，DB 等没有特定的限制）创建一个 span，通过一个 64 位 ID 标识它，uuid 较为方便，span 中还有其他的数据，例如描述信息，时间戳，key-value 对的（Annotation）tag 信息，parent_id 等,其中 parent-id 可以表示 span 调用链路来源。



Span

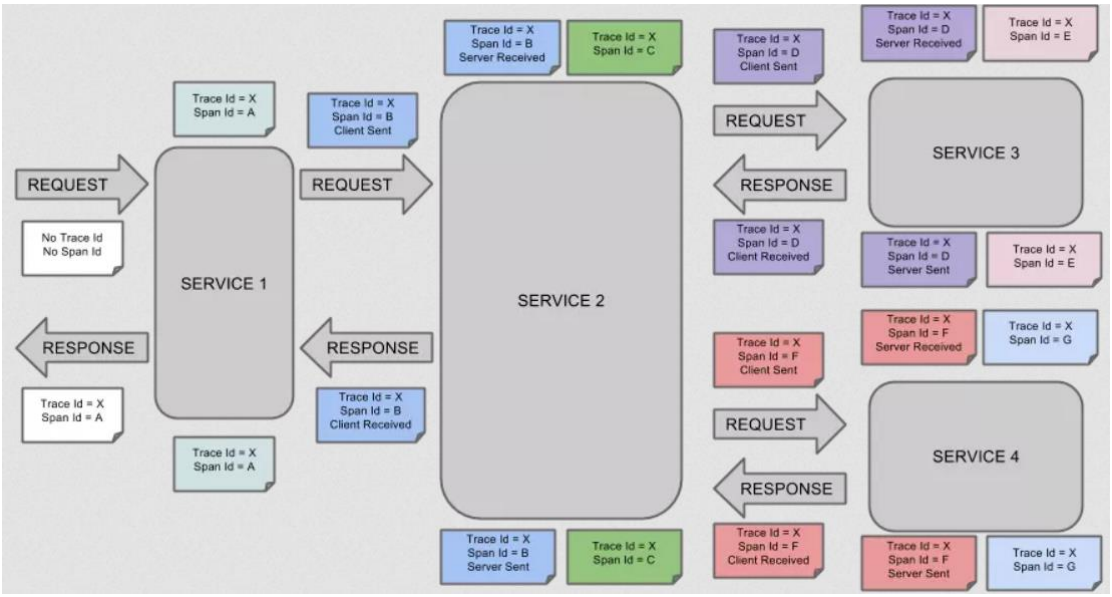
上图说明了 span 在一次大的跟踪过程中是什么样的。Dapper 记录了 span 名称，以及每个 span 的 ID 和父 ID，以重建在一次追踪过程中不同 span 之间的关系。如果一个 span 没有父 ID 被称为 root span。所有 span 都挂在一个特定的跟踪上，也共用一个跟踪 id。

Span 数据结构：

```
type Span struct {
TraceID     int64   #用于标示一次完整的请求 id
Name          string
ID             int64   #当前这次调用 span_id
ParentID    int64   #上层服务的调用 span_id   最上层服务 parent_id 为 null
Annotation []Annotation   #用于标记的时间戳
Debug         bool
}
```

2. Trace 是什么?

类似于树结构的 Span 集合，表示一次完整的跟踪，从请求到服务器开始，服务器返回 response 结束，跟踪每次 rpc 调用的耗时，存在唯一标识 trace_id。比如：你运行的分布式大数据存储一次 Trace 就由你的一次请求组成。



每种颜色的 note 标注了一个 span，一条链路通过 TraceId 唯一标识，Span 标识发起的请求信息。树节点是整个架构的基本单元，而每一个节点又是对 span 的引用。节点之间的连线表示的 span 和它的父 span 直接的关系。虽然 span 在日志文件中只是简单的代表 span 的开始和结束时间，他们在整个树形结构中却是相对独立的。

7.4 初始化实验环境

新创建一台虚拟机安装 pinpoint，配置如下：

主机名	ip	配置	网络
pinpoint	192.168.1.65	4vCPU/8G 内存/60G 硬盘	NAT

注： 新机器的初始化只需要按照上面课程步骤进行初始化即可

7.4.1 配置静态 IP

把虚拟机或者物理机配置成静态 ip 地址，这样机器重新启动后 ip 地址也不会发生改变。

在 harbor 节点配置网络

修改/etc/sysconfig/network-scripts/ifcfg-ens33 文件，变成如下：

```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
IPADDR=192.168.1.65
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
DNS1=192.168.1.2
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
```

IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
DEVICE=ens33
ONBOOT=yes
修改配置文件之后需要重启网络服务才能使配置生效，重启网络服务命令如下：
service network restart

7.4.2 修改 yum 源

下面的步骤在 pinpoint 节点操作
mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup
下载阿里的 yum 源
wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo
配置安装 k8s 需要的 yum 源
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=0
EOF
#添加 docker 需要的 yum 源
yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
安装软件包
yum -y install wget net-tools nfs-utils lrzsz gcc gcc-c++ make cmake libxml2-devel openssl-devel curl curl-devel unzip sudo ntp libaio-devel wget vim ncurses-devel autoconf automake zlib-devel python-devel epel-release openssh-server socat ipvsadm conntrack ntpdate yum-utils device-mapper-persistent-data lvm2 telnet

7.4.3 配置防火墙

关闭 firewalld 防火墙， centos7 系统默认使用的是 firewalld 防火墙，停止 firewalld 防火墙，并禁用这个服务。
systemctl stop firewalld && systemctl disable firewalld

7.4.4 时间同步

ntpdate cn.pool.ntp.org
编辑计划任务，每小时做一次同步
1）crontab -e
* */1 * * * /usr/sbin/ntpdate cn.pool.ntp.org
2）重启 crond 服务：
service crond restart

7.4.5 关闭 selinux

关闭 selinux，设置永久关闭，这样重启机器 selinux 也处于关闭状态

可用下面方式修改：

```
sed -i  's/SELINUX=enforcing/SELINUX=disabled/' /etc/sysconfig/selinux
sed -i  's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
```

上面文件修改之后，需要重启虚拟机，如果测试环境可以用如下命令强制重启：

```
reboot -f
```

注：生产环境不要 reboot -f，要正常关机重启

查看 selinux 是否修改成功

重启之后登录到机器上用如下命令：

```
getenforce
```

显示 Disabled 说明 selinux 已经处于关闭状态

7.4.6 修改内核参数

```
cat <<EOF > /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl --system
```

注：*sysctl --system* 这个会加载所有的 *sysctl* 配置

7.4.7 修改主机名

在 192.168.1.65 上：

```
hostnamectl set-hostname pinpoint
```

7.5 安装 docker

7.5.1 查看 docker 版本

```
yum list docker-ce --showduplicates |sort -r
```

7.5.2 安装 docker

```
yum install -y docker-ce-19.03.7-3.el7
systemctl enable docker && systemctl start docker
#查看 docker 状态，如果状态是 active（running），说明 docker 是正常运行状态
systemctl status docker
```

7.5.3 修改 docker 配置文件

```
cat > /etc/docker/daemon.json <<EOF
{
"registry-mirrors":["https://rsbud4vc.mirror.aliyuncs.com","https://registry.docker-
  cn.com","https://docker.mirrors.ustc.edu.cn","https://dockerhub.azk8s.cn","http://hu
  b-mirror.c.163.com","http://qtid6917.mirror.aliyuncs.com"]
}
EOF
```

注：

```
"registry-mirrors":["https://rsbud4vc.mirror.aliyuncs.com","https://registry.docker-
cn.com","https://docker.mirrors.ustc.edu.cn","https://dockerhub.azk8s.cn","http://hub
```

-mirror.c.163.com","http://qtid6917.mirror.aliyuncs.com"]
上面配置的是 docker 镜像加速器

7.5.4 重启 docker 使配置生效

systemctl daemon-reload && systemctl restart docker && systemctl status docker

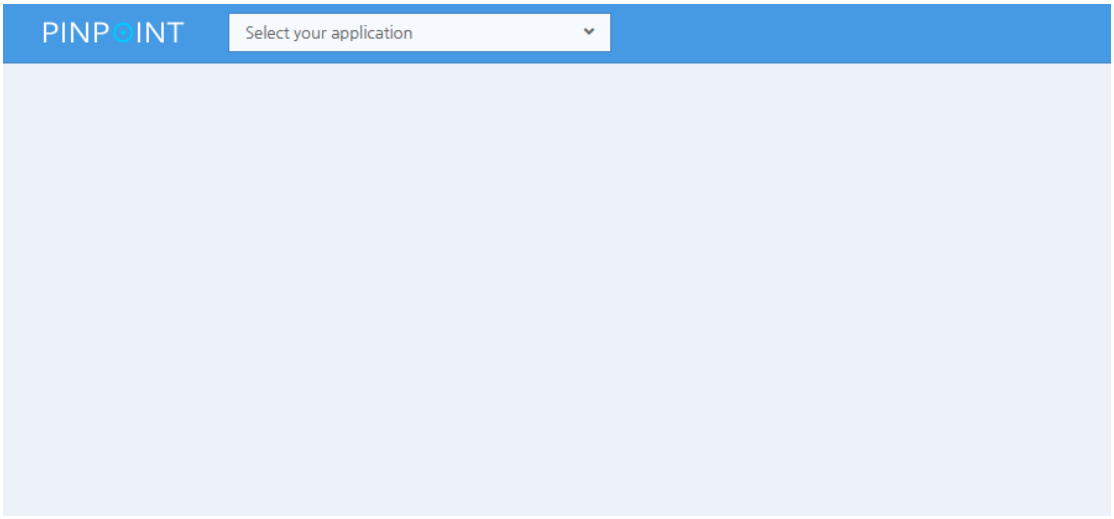
7.5.5 开启机器的 bridge 模式

```
#临时生效
echo 1 > /proc/sys/net/bridge/bridge-nf-call-iptables
echo 1 >/proc/sys/net/bridge/bridge-nf-call-ip6tables
#永久生效
echo """
vm.swappiness = 0
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
""" > /etc/sysctl.conf
sysctl -p
```

7.6 安装 pinpoint 服务

```
#把 pinpoint-docker-2.0.1.zip 上传到 pinpoint 机器上，unzip 解压
unzip pinpoint-docker-2.0.1.zip
cd pinpoint-docker-2.0.1

修改 docker-compose.yml 文件的 version 版本，如 2.2，变成自己支持的版本
version: "3.6" 变成 version: "2.2"
#拉取镜像
yum install docker-compose -y
# 拉取安装 pinpoint 服务需要的镜像，把 pinpoint-image.tar.gz 镜像压缩包上传到 pinpoint 机
  器上，手动解压：
[root@pinpoint ~]# docker load -i pinpoint-image.tar.gz
#通过下面命令也可以把安装 pinpoint 需要的镜像在线获取，但是速度较慢，可以通过上面手动解
压方式拉取
docker-compose pull
#启动服务
docker-compose up -d
#查看对应的服务是否启动
docker ps | grep pinpoint
#找到 pinpoint-web，可看到在宿主机绑定的端口是 8079，
在浏览器访问 ip:8079 即可访问 pinpoint 的 web ui 界面
```



7.7 部署电商项目：带 pinpoint agent 客户端

把 microservic-test-dev1.zip 上传到 k8s 的控制节点 xuegod63 上，解压：
unzip microservic-test-dev1.zip

7.7.1 修改源代码，更改数据库连接地址

1）修改库存数据库
[root@xuegod63 ~]# vim /root/microservic-test-dev1/stock-service/stock-service-biz/src/main/resources/application-fat.yml

jdbc:mysql://192.168.1.63:3306/tb_stock?characterEncoding=utf-8
#变成自己的 mysql 数据库所在机器的地址
2）修改产品数据库
[root@xuegod63 ~]# vim /root/microservic-test-dev1/product-service/product-service-biz/src/main/resources/application-fat.yml

jdbc:mysql://192.168.1.63:3306/tb_product?characterEncoding=utf-8
#变成自己的数据库地址
3）修改订单数据库
[root@xuegod63 ~]# vim /root/microservic-test-dev1/order-service/order-service-biz/src/main/resources/application-fat.yml

url: jdbc:mysql://192.168.1.63:3306/tb_order?characterEncoding=utf-8
#变成自己的数据库地址

#修改配置，指定 pinpoint 服务端
[root@xuegod63 ~]# cd /root/microservic-test-dev1/product-service/product-service-biz/pinpoint

打开 pinpoint.config，需要修改的地方如下：
profiler.collector.ip=192.168.1.65
#这个是安装 pinpoint 服务的 ip 地址

cd /root/microservic-test-dev1/order-service/order-service-biz/pinpoint
打开 pinpoint.config，需要修改的地方如下：
profiler.collector.ip=192.168.1.65

cd /root/microservic-test-dev1/stock-service/stock-service-biz/pinpoint
打开 pinpoint.config，需要修改的地方如下：
profiler.collector.ip=192.168.1.65

cd /root/microservic-test-dev1/portal-service/pinpoint
打开 pinpoint.config，需要修改的地方如下：
profiler.collector.ip=192.168.1.65

cd /root/microservic-test-dev1/gateway-service/pinpoint
打开 pinpoint.config，需要修改的地方如下：
profiler.collector.ip=192.168.1.65

cd /root/microservic-test-dev1/eureka-service/pinpoint
打开 pinpoint.config，需要修改的地方如下：
profiler.collector.ip=192.168.1.65

#上面修改的 ip 是部署 pinpoint 的机器的 ip 地址，也就是 pinpoint 的服务端

7.7.2 通过 Maven 编译、打包、构建代码

在 k8s 的控制节点 xeugod63 操作
修改源代码之后回到/root/microservic-test-dev1 目录下执行如下命令：
[root@xuegod63 pinpoint]# cd /root/microservic-test-dev1
[root@xuegod63 microservic-test-dev1]# mvn clean package -D maven.test.skip=true
#看到如下，说明代码已经打包完成

```
[INFO] simple-microservice ............................... SUCCESS [0.251s]
[INFO] basic-common ...................................... SUCCESS [0.006s]
[INFO] basic-common-core ................................. SUCCESS [5.560s]
[INFO] gateway-service ................................... SUCCESS [2.632s]
[INFO] eureka-service .................................... SUCCESS [2.485s]
[INFO] product-service ................................... SUCCESS [0.013s]
[INFO] product-service-api ............................... SUCCESS [1.161s]
[INFO] stock-service ..................................... SUCCESS [0.001s]
[INFO] stock-service-api ................................. SUCCESS [0.433s]
[INFO] product-service-biz ............................... SUCCESS [0.795s]
[INFO] stock-service-biz ................................. SUCCESS [0.536s]
[INFO] order-service ..................................... SUCCESS [0.001s]
[INFO] order-service-api ................................. SUCCESS [0.903s]
[INFO] order-service-biz ................................. SUCCESS [0.839s]
[INFO] basic-common-bom .................................. SUCCESS [0.001s]
[INFO] portal-service .................................... SUCCESS [1.028s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 17.558s
[INFO] Finished at: Wed Apr 07 22:01:18 CST 2021
[INFO] Final Memory: 79M/567M
```

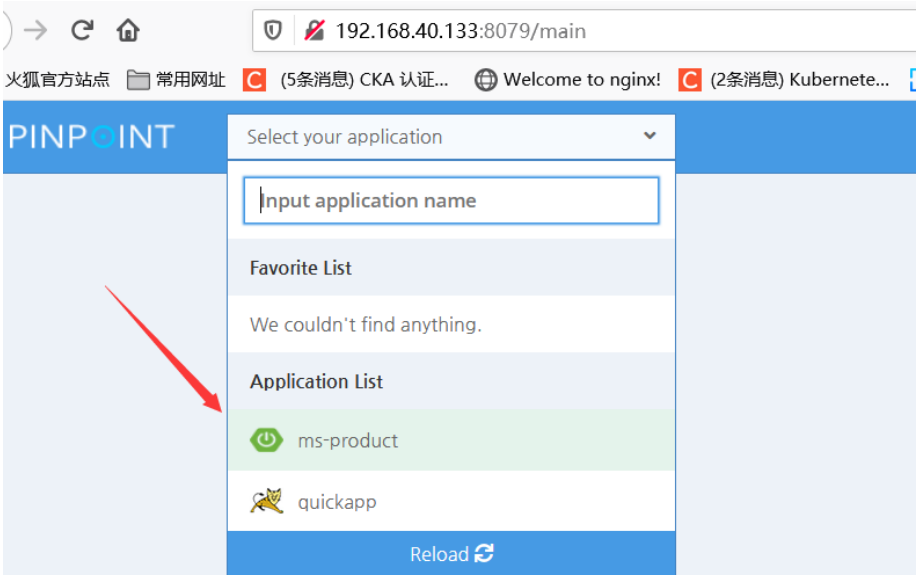
7.7.3 部署带 pinpoint agent 端的产品服务

[root@xuegod63 microservic-test-dev1]# cd /root/microservic-test-dev1/product-service/product-service-biz
docker build -t 192.168.1.62/microservice/product:v2 .
docker push 192.168.1.62/microservice/product:v2

[root@xuegod63 product-service-biz]# cd /root/microservic-test-dev1/k8s
修改 product.yaml 文件，把镜像变成 image: 192.168.1.62/microservice/product:v2

更新 yaml 文件
kubectl delete -f product.yaml
kubectl apply -f product.yaml
查看 pod 状态
kubectl get pods -n ms | grep product
看到如下 running 说明 pod 运行正常：
product-775f5f74d9-4nssp 1/1 Running 0 10s

上面部署成功之后，登陆 pinpoint web 界面，可看到有个新的应用是 ms-product



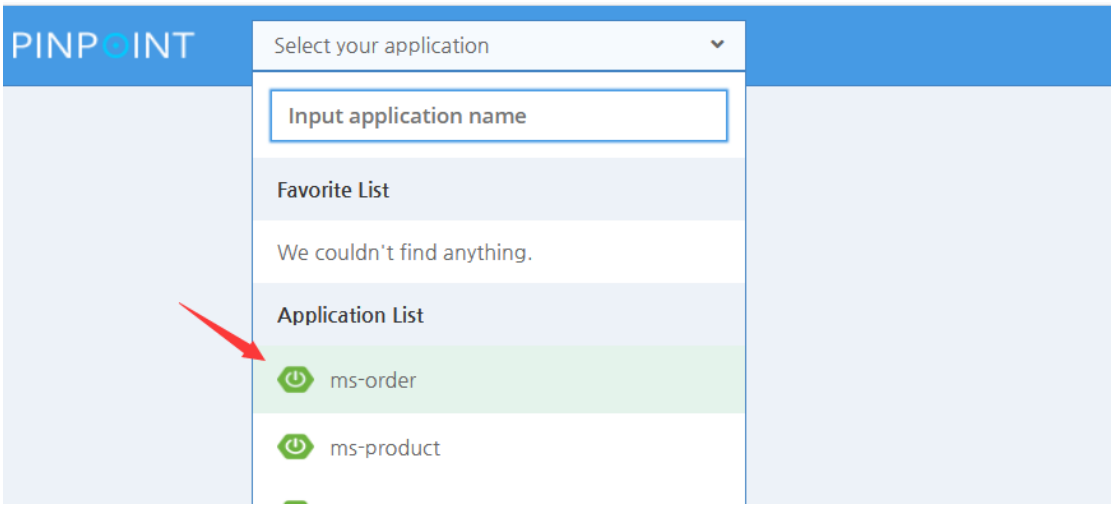
7.7.4 部署带 pinpoint agent 端的订单服务

[root@xuegod63 k8s]# cd /root/microservic-test-dev1/order-service/order-service-biz
[root@xuegod63 order-service-biz]# docker build -t 192.168.1.62/microservice/order:v2 .
docker push 192.168.1.62/microservice/order:v2
[root@xuegod63 order-service-biz]# docker push 192.168.1.62/microservice/order:v2

[root@xuegod63 order-service-biz]# cd /root/microservic-test-dev1/k8s
修改 order.yaml 文件，把镜像变成 image: 192.168.1.62/microservice/order:v2

更新 yaml 文件
kubectl delete -f order.yaml
kubectl apply -f order.yaml
查看 pod 状态
kubectl get pods -n ms | grep order
看到如下 running 说明 pod 运行正常：
order-775f5f74d9-4nssp 1/1 Running 0 18s

上面部署成功之后，登陆 pinpoint web 界面，可看到有个新的应用是 ms-order



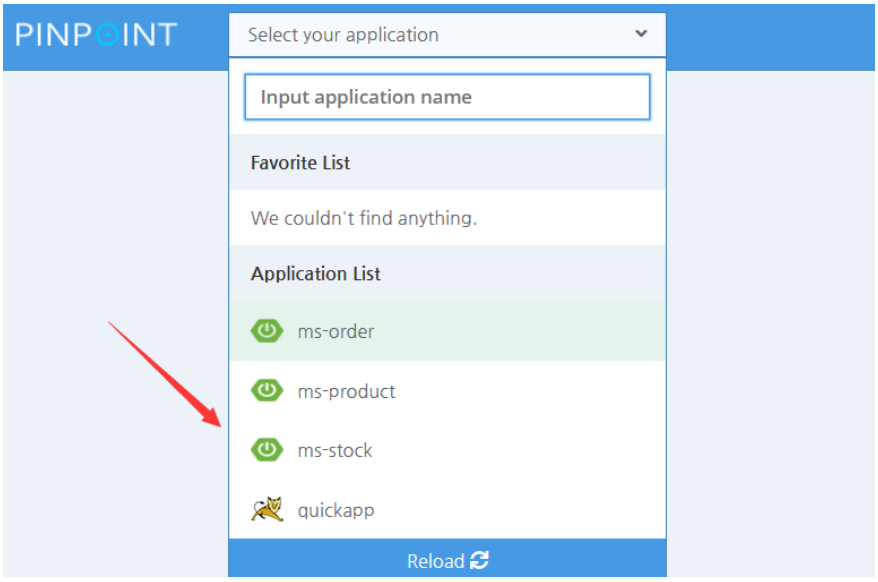
7.7.5 部署带 pinpoint agent 端的 stock 服务

cd /root/microservic-test-dev1/stock-service/stock-service-biz
docker build -t 192.168.1.62/microservice/stock:v2 .
docker push 192.168.1.62/microservice/stock:v2

cd /root/microservic-test-dev1/k8s
修改 stock.yaml 文件，把镜像变成 image: 192.168.1.62/microservice/stock:v2

更新 yaml 文件
kubectl delete -f stock.yaml
kubectl apply -f stock.yaml
查看 pod 状态
kubectl get pods -n ms | grep stock
看到如下 running 说明 pod 运行正常：
stock-775f5f74d9-4nssp 1/1 Running 0 18s

上面部署成功之后，登陆 pinpoint web 界面，可看到有个新的应用是 ms-stock



7.7.6　部署带 pinpoint agent 端的 portal 服务

cd　/root/microservic-test-dev1/portal-service/

docker build -t　192.168.1.62/microservice/portal:v2 .

docker push　192.168.1.62/microservice/portal:v2

cd /root/microservic-test-dev1/k8s

修改 portal.yaml 文件，把镜像变成 image: 192.168.1.62/microservice/portal:v2

更新 yaml 文件

kubectl delete -f portal.yaml

kubectl apply -f portal.yaml

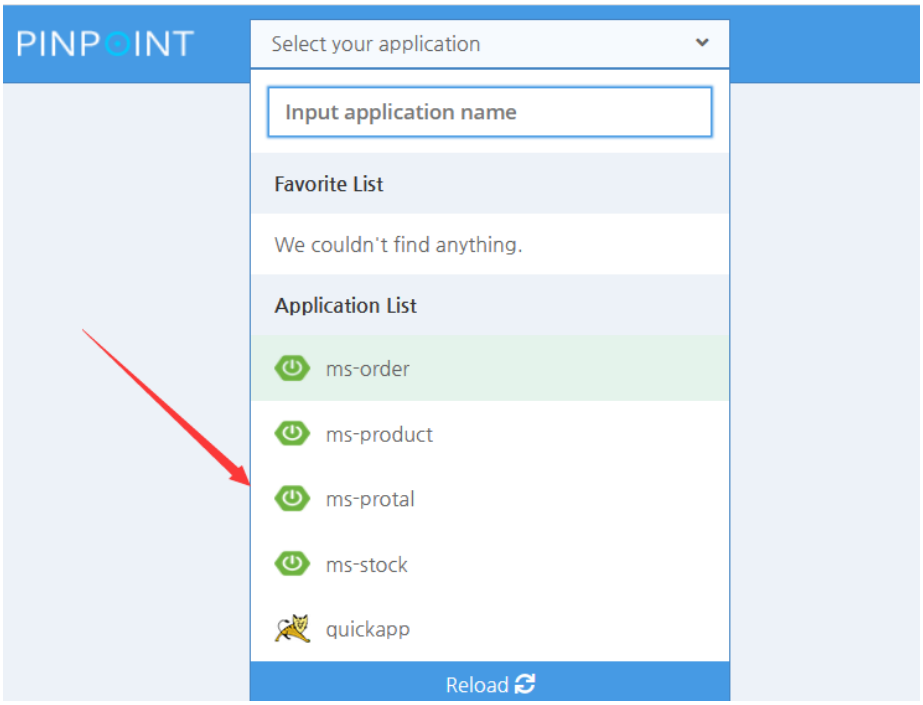
查看 pod 状态

kubectl get pods -n ms　| grep portal

看到如下 running 说明 pod 运行正常：

portal-775f5f74d9-4nssp　1/1　Running　0　18s

上面部署成功之后，登陆 pinpoint web 界面，可看到有个新的应用是 ms-portal



7.7.7　部署带 pinpoint agent 端的网关服务

1）构建镜像

cd　/root/microservic-test-dev1/gateway-service/

docker build -t　192.168.1.62/microservice/gateway:v2 .

docker push 192.168.1.62/microservice/gateway:v2

2）部署服务

cd /root/microservic-test-dev1/k8s

修改 gateway.yaml 文件，把镜像变成 image: 192.168.1.62/microservice/gateway:v2

3）更新 yaml 文件

kubectl delete -f gateway.yaml

kubectl apply -f gateway.yaml

4）查看 pod 状态

kubectl get pods -n ms　| grep gateway

5）看到如下 running 说明 pod 运行正常

gateway-c94f4d95c-2dqvw　1/1　Running　0　31s

gateway-c94f4d95c-l4jmq　1/1　Running　0　31s

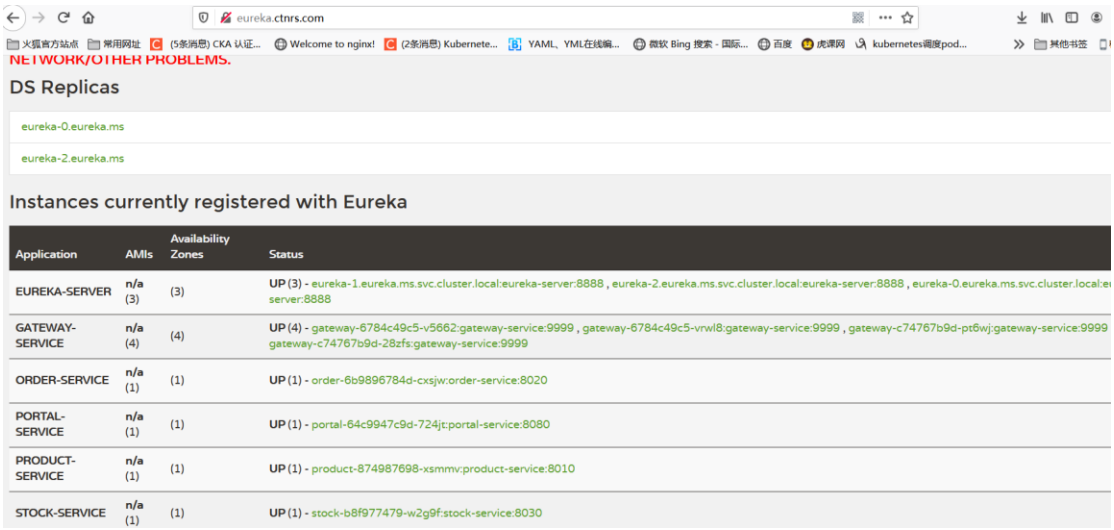
6）配置 hosts 文件

gateway 的域名是 gateway.ctnrs.com，需要在电脑找到 hosts 文件，再增加一行如下：

192.168.1.64　gateway.ctnrs.com

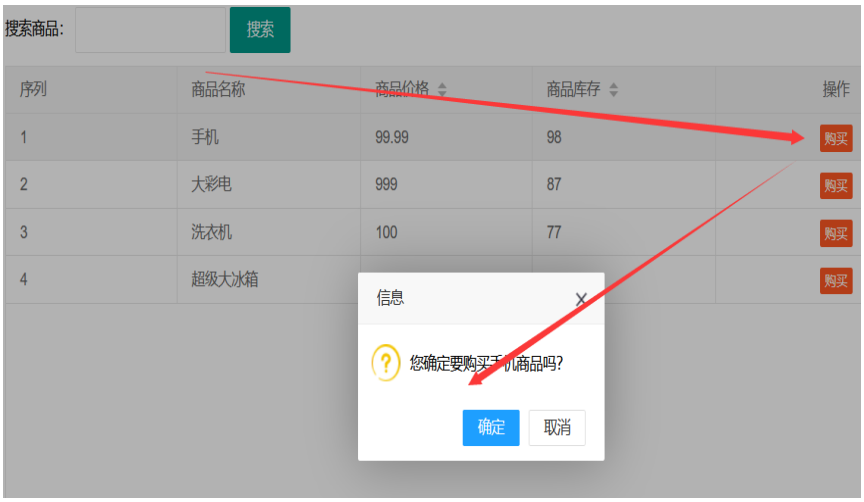
在浏览器访问 eureka.ctnrs.com

可看到 GATEWAY-SERVICE 已经注册到 eureka 了：

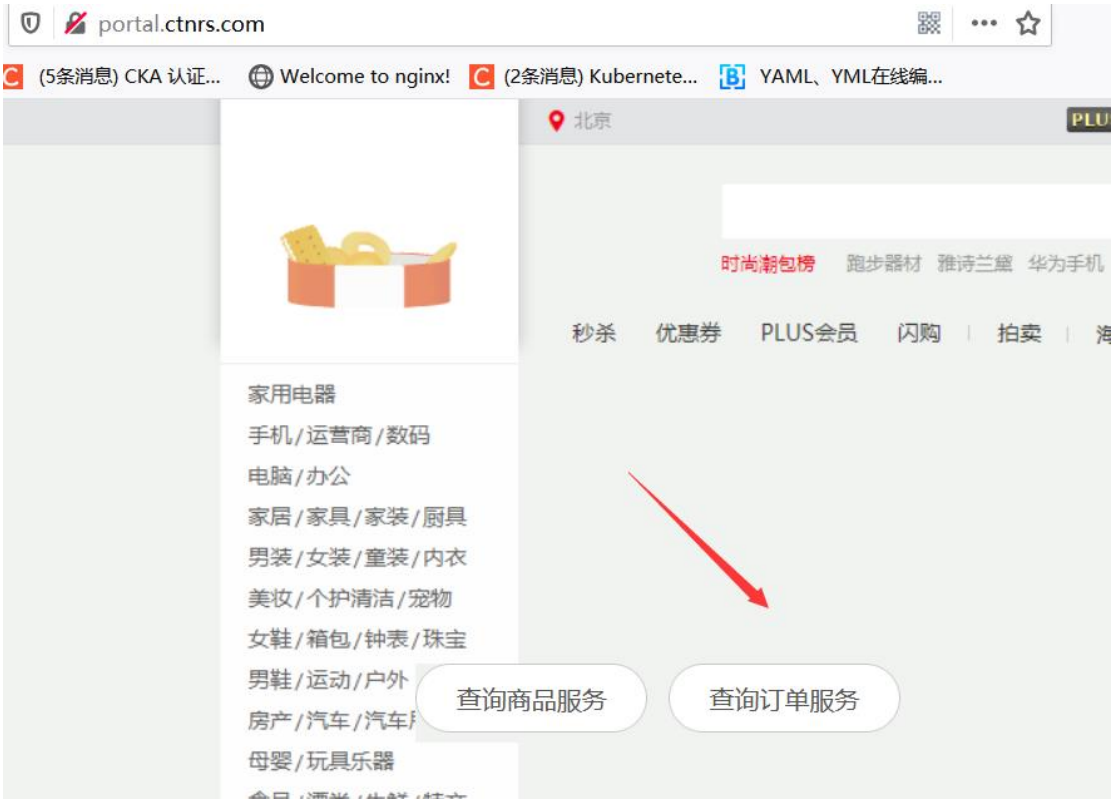


#访问前端页面

http://portal.ctnrs.com/



学神 IT 教育官方 QQ 群：957231097 或唐老师 QQ：3340273106 领取更多资料



7.8 pinpoint web 界面使用说明

总结:

实战：在 k8s 部署 Springcloud 项目：

1、部署 gateway 网关服务
2、部署 portal 前端服务
3、部署 order 订单服务
4、部署 product 产品服务
5、部署 stock 库存服务

7.1 全链路监控系统概述
7.2 介绍典型的全链路监控工具
7.3 全链路监控工具对比分析
7.4 安装 pinpoint 全链路监控服务

实战 1：在 k8s 集群中部署大型电商项目：模拟京东购物平台

实战 2：通过 Pinpoint 实现电商平台功能模块全链路监控

实战 3：介绍 Pinpoint web 界面使用技巧