我们检测到你可能使用了 AdBlock 或 Adblock Plus，它的部分策略可能会影响到正常功能的使用（如关注）。
你可以设定特殊规则或将知乎加入白名单，以便我们更好地提供服务。（为什么？）

知乎 首发于 云音乐前端技术团队专栏 关注专栏 写文章 ...

前端领域的 Docker 与 Kubernetes

云音乐前... 已认证的官方帐号 + 关注他

36 人赞同了该文章

看完本文希望读者能够了解到，Docker 的基本原理，Kubernetes 是怎么工作的， 对于前端 Kubernetes 有哪些优势与玩法。

Docker 和传统部署方式最大的不同在于，它将不会限制我们使用任何工具，任何语言，任何版本的 runtime，Docker 将我们的应用看成一个只提供网络服务的盒子(也即容器)，Kubernetes 则是对这些盒子进行更多自动化的操作，自动创建，自动重启，自动扩容，自动调度，这个过程称之为容器编排。

在今天，容器编排技术给 Web 应用带来了巨大的灵活性，让我们轻松创建需要的程序对外提供服务。和传统的 IaaS 相比，不需要去关心云主机申请，云主机配置等信息，也不需考虑云主机故障导致的服务不可用，由 Kubernetes 的副本控制器帮我们完成云主机故障发生后容器迁移。

本篇文章和大家一起，回顾一下从 Docker 到 Kubernetes 的一些相关内容，最后再看看 Kubernetes 在前端领域有哪些优势和新玩法。

Docker 安装

- Linux Debian/Ubuntu， 安装 社区版DockerCE
- Windows 一键安装

如果是 Windows10， Windows7 将会使用 VirtualBox 安装 Linux 作为 Docker 的宿主机。
Windows10 Pro 会使用 Hyper-V 安装 Linux 作为 Docker 的宿主机。

- macOS 一键安装

Docker 基本信息

▲ 赞同 36 ▼ 添加评论 分享 喜欢 收藏 ...

默认 Docker 存储位置为 `/var/lib/docker`，所有的镜像，容器，卷都会在这里，如果你使用了多硬盘，或者挂载了 SSD 不在 `/` 上，需要修改默认路径（graph）到合适位置，配置文件为 `/etc/docker/daemon.json`， 例如

```
{
  "bip": "192.168.0.1/16",
  "graph": "/mnt/ssd/0/docker"
}
```

Docker 在安装过程中会自动创建好 docker0 网卡，并分配 ip 给他。 上面指定的 bip 是指定了 docker0 网卡的 ip， 如果不指定那么在创建 docker0 时会自动根据主机 ip 选取一个合适的 ip，不过由于网络的复杂性，特别是机房网络内很容易发现地址选取冲突，这时候就需要手动指定 bip 为一个合适的值。docker 的 ip 选取规则这篇文章分析的很好， 可以参考 blog.csdn.net/longxing_... 。

安装并启动后可以通过 `docker info` 查看Docker的一些配置信息。

Docker hello world

Docker 检查安装是否正常的第一个测试命令很简单。

```
docker run hello-world
```

首先他会去 Docker Hub 上下载 hello-world 这个镜像，然后在本地运行这个镜像，启动后的这个 Docker 服务称之为容器。容器创建后就会执行规定的入口程序，程序执行向流中输出了一些信息后退出，容器也会随着这个入口程序的结束而结束。

- 查看所有容器

```
docker ps -a
```

输出如下:

```
cf9a6bc212f9        hello-world                    "/hello"                 28
```

第一列为容器 id， 很多针对容器的操作都需要这个 id， 例如下面一些常用的操作。

```
docker rm container_id
docker stop container_id
docker start container_id
docker describe container_id
```

这里有个 `docker start container_id`， 启动一个容器，说明容器即使退出后其资源依然存在，还可以使用 `docker start` 重启这个容器。要想让容器退出后自动删除可以在 `docker run` 时指定 `--rm` 参数。

当我们运行这个命令时 Docker 会去下载 hello-world 这个镜像缓存到本地，这样当下次再运行这条命令时就不需要去源中下载。

- 查看本地镜像

```
docker images
```

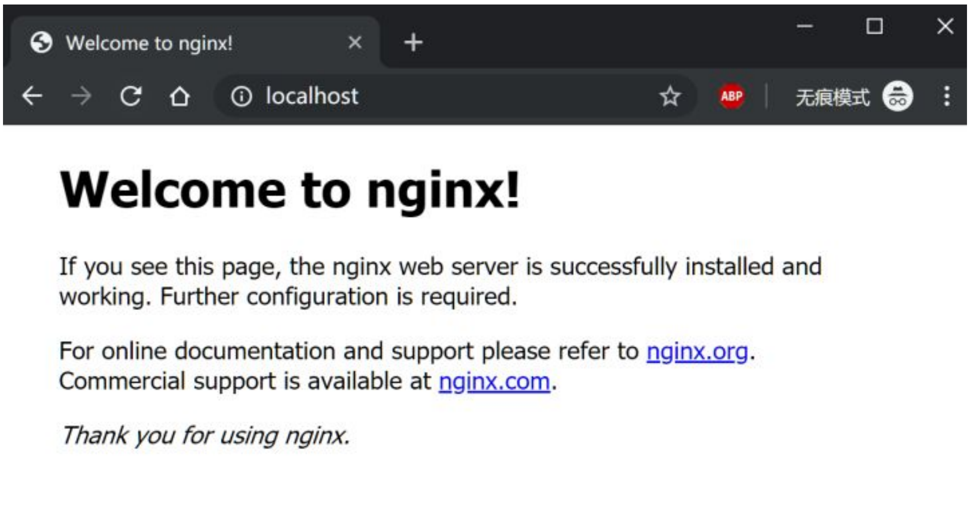
运行 Nginx

Nginx 作为使用广泛的 W
证网络配置情况， 使用下

▲ 赞同 36 ▼ 添加评论 分享 喜欢 收藏 ...

访问 `localhost:80` 端口即可看到 Ônğ฿Ú服务启动，控制台中可以看到 Ônğ฿Ú服务的日志输出。

因为 k”\ạæX̧H内的网络与外部世界是隔离的，所以我们需要手动指定端口转发 `-p 80:80` 来显式将宿主机的' ⁰ H前ɪ转发到容器的' ⁰ 端口，暴露端口是我们提供服务最常用的使用方式之一。也有一些其他类型的服务，例如日志处理，数据收集需要共享数据卷才能提供服务，所有这些都需要我们在启动容器时显式指定。



一些常见的启动参数'H `-p` *本机端口 容器端口 映射本地端口到容器* `-P` 将容器端口映射为本机随机端口 `-v` *本地路径或卷名 容器路径 将本地路径或者数据卷挂载到容器的指定位置* `-it` 作为交互式命令启动 `-d` *将容器放在后台运行* `--rm` 容器退出后清除资源

k”\ạæX̧H是如何工作的

k”\ạæX̧H的底层核心原理是利用了 ǂạ̄ęőŰ内核的 ę ȟ XķH \X 以及 \nJ̈H őH特性，其中 ę ȟ XķH \X 进行资源隔离，\nJ̈H őH进行资源配额， 其中 ǂạ̄ęőŰ内核中一共有 ˙ 种 ę ȟ XķH \X，分别对应如下。

Namespace	系统调用函数	隔离内容
UTS	CLONE_NEWUTS	主机与域名
IPC	CLONE_NEWIPC	信号量、消息队列和共享内存
PID	CLONE_NEWPID	进程编号
Network	CLONE_NEWNET	网络设备、网络栈、端口等
Mount	CLONE_NEWNS	挂载点(文件系统)
User	CLONE_NEWUSER	用户和用户组

在系统调用中有三个与ę ȟ XķH \X有关的函数'H

⁵ ĥ\ȄęXȟ ę(ñ Hmj̧Ȅęő Ű ȟ ęő̈H nJXm̃

如果我想让子进程拥有独立的网络地址，ĶbĒıÆ协议栈，可以下面这样指定。

```
clone(cb, *stack , CLONE_NEWNET, 0)
```

⁵ ĥőęķŪ ĦXȟ ę(ñ Hmj̧Ȅęő Ű ȟ ęő̈H nJXm̃

▲ 赞同 ⁷ˉ ▼ 添加评论 分享 喜欢 收藏 ⋯

将当前进程转移到新的 ę ň XķḤ ʅX 中， 例如使用 NḶǽ或 ţ NḶǽ创建的进程将默认共享父级资源，使用 őęķŨ ḷX 将子进程从父级取消共享。

⁵ ṅķXṃęķ ň ę ń ḷṁ ḷǽőŰň ęőḤ nḷXṁ

给指定的ĒÆ指定 ę ň XķḤ ʅX， 通常用于共享 ę ň XķḤ ʅX。

ṫǽőŨ在内核层支持了在系统调用中隔离 ę ň XķḤ ʅX， 通过给一个进程分配单独的 ę ň XķḤ ʅX 从而让其在各个资源维度进行隔离，每个进程都能获取到自己的主机名，Æb， ĒÆ， Æ， 根文件系统，用户组等信息，就像在一个独占系统中，不过虽然资源进行了隔离，但是内核还是共享同一个，这也是比传统虚拟机轻量的原因之一。

另外只有资源进行隔离还不够，要想保证真正的故障隔离，互不影响， 还需要对针对 bĒń， 内存，ōĒń 等进行限制，因为如果一个程序出现死循环或者内存泄露也会导致别的程序无法运行。资源配额是使用内核的 ʅnḤ őḤ 特性来完成，想了解细节的同学可以参考 Ḥ ʅę6Ēnķʅ ň ık ň ň ŭőǽıḤṁ。 （另外强烈推荐在 ṫǽőŰ ň 以上的内核跑容器，ṫǽőŰ ṅŰ中有已知内核不稳定导致主机重启的问题）

k ” ʅǽXḤ网络

一个容器要想提供服务，就需要将自身的网络暴露出去。k ” ʅǽXḤ是与宿主机上的环境是隔离的，要想暴露服务就需要显示告诉 k ” ʅǽXḤ哪些端口允许外部访问，在运行 `docker run -p 80:80 nginx` 时这里就是将容器内部的 ⁾ ⁰ 端口暴露到宿主机的 ⁾ ⁰ 端口上，具体的端口转发下面会具体分析一下。容器的网络部分是容器中最重要的部分，也是构建大型集群的基石，在我们部署 k ” ʅǽXḤ的应用时，需要要对网络有个基本的了解。

k ” ʅǽXḤ提供了四种网络模式，分别为 `Host` ， `Container` ， `None` ， `Bridge` 使用 `--net` 进行指定

§ ” ķṁ模式 Ḥ

```
docker run --net host nginx
```

§ ” ķṁ模式不会单独为容器创建 ęXṁ ” ḷǽę ň XķḤ ʅX， 容器内部直接使用宿主机网卡，此时容器内获取 ǽḤ 为宿主机 ǽḤ， 端口绑定直接绑在宿主机网卡上，优点是网络传输时不用经过 Ô2Ķ 转换，效率更高速度更快。

b ” ęṁǽXḤ模式 Ḥ

```
docker run --net container:xxx_containerid nginx
```

和指定的 ʅ ” ęṁǽXḤ共享 ęXṁ ” ḷǽę ň XķḤ ʅX， 共享网络配置，ǽḤ 地址和端口，其中无法共享网络模式为 § ” ķṁ的容器。

Ô” ęX 模式 Ḥ

```
docker run --net none busybox ifconfig
```

指定为 Ô” ęX 模式的容器内将不会分配网卡设备，仅有内部 `lo` 网络。

赞同 ⁷ ˙ ▼ 添加评论 分享 喜欢 收藏 ⋯

```
D:\
λ docker run --rm --net none busybox ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

D:\
```

^ ̧ǟ nX 模式

```
docekr run ——net bridge busybox ifconfig
```

```
master@ubuntuserver2:~$ docker run --net bridge --rm busybox ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:12:00:02
          inet addr:172.18.0.2  Bcast:172.18.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:180 (180.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

master@ubuntuserver2:~$
```

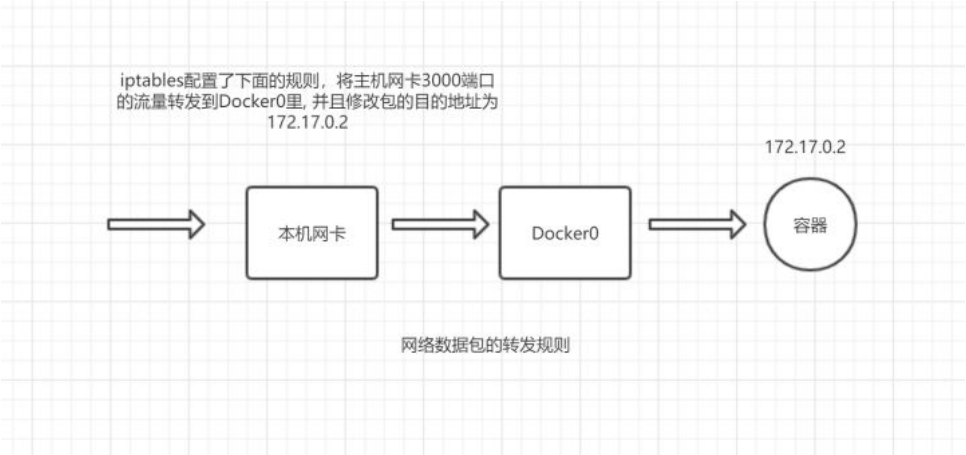
该模式为默认模式，容器启动时会被分配一个单独的 ȩXmı̧"ļǣȩ ȟ XkḤ ɭX，同时 k"ɭǣXḤ在安装ı初始化时会在宿主机上创建一个名为 l"ɭǣXḤ 的网桥，该网桥也作为容器的默认网关，容器网络会在该网关网段内进行 ǟḤ的分配。

当我执行 `docker run -p 3000:80 nginx` 时，k"ɭǣXḤ会在宿主机上创建下面一条 ǟḤm6ǠX转发规则。

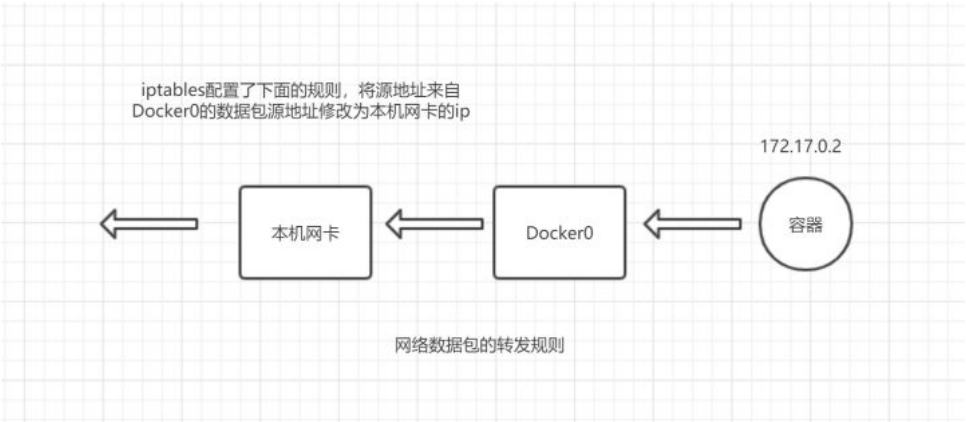
```
master@ubuntufordechang:~$ sudo iptables -L -t nat|grep 3000
DNAT       tcp  --  anywhere             anywhere             tcp dpt:3000 to:172.17.0.3:80
master@ubuntufordechang:~$
```

最底下的规则显示当外部请求主机网卡 ⁷⁰⁰⁰ 端口时将它进行目的地址转换ʜk Ô2Ḳɪ， 目的地址修改为 `172.18.0.2`，端口修改为 ⁾⁰，修改好目的地址后流量会从本机默认网卡经过 l"ɭǣXḤ 转发到对应的容器，这样当外部请求宿主机的 ⁷⁰⁰⁰ 端口，内部会将流量转发给内部容器服务，从而实现服务的暴露。

赞同 ⁷· ▼ 添加评论 分享 喜欢 收藏 ···



同样 k”\æXH内部访问外部接口也会进行源地址转换HgÔ2Ķr， 容器内部请求 `google.com`， 服务器上收到的将是主机网卡的 ãḤ。



^ĻāḤnjX模式由于多了一层 Ô2Ķ 转换所以效率会比 §”ķṁ模式差一些，但是能够很好的隔离外部网络环境，让容器独享 ãḤ且具有完整的端口空间。

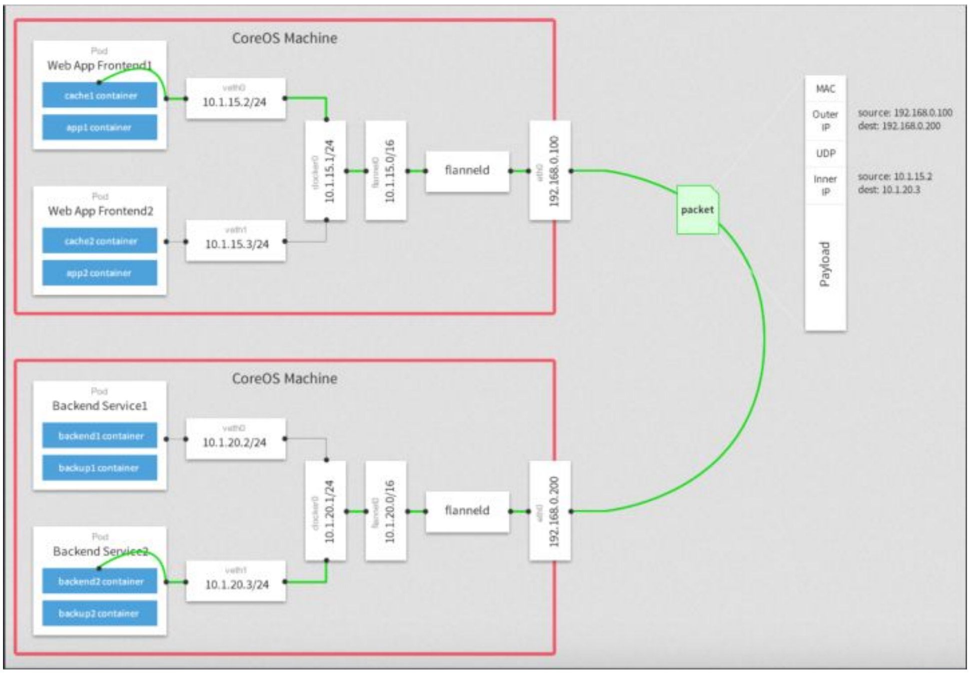
上面四种网络模式是 k”\æXH自带的几种工作方式，但是部署 Ÿő6XţęXṁk 需要所有的容器都工作在一个局域网中，所以在部署集群时需要多主机网络插件的支持。

ôÈęęXÈ

多主机网络解决方案有 bÔbô 推出的 bÔÁ规范以及 k”\æXH自带的 bÔÎ 方案，但是目前大家用的最多的还是 bÔÁ规范，其中一种实现就是 ôÈęęXÈ。

ôÈęęXÈ使用了报文嵌套技术来解决多主机网络互通问题，将原始报文进行封包，指定包ãṭ为目的主机地址，等包到达主机后再进行拆包传送到对应的容器。下图显示 NÈęęXÈ使用效率更高的 ńkĒ 协议来在主机间传输报文。

▲ 赞同 7 · ▼ 添加评论 分享 喜欢 收藏 ···



目前主流跨主机通信目前常用的有三种，各有优缺，视场景选择'H

- "A `overlay`， 即上面的报文嵌套。
- "A `hostgw` 通过修改主机路由表实现转发，不需要拆包和封包，效率更高，但同样限制比较多，只适合在相同局域网中的主机使用。
- "A使用软件实现的 `BGP` （边界网关协议）以此向网络中的路由器广播路由规则。和 Ű' ḳṁ]Ţ 一样不需要拆包，但是实现成本较高。

有了bÔÆ才能在此基础上构建 Ÿő6 XḫẹXṁẊḳ 集群。

Ÿő6 XḫẹXṁẊḳ 介绍

在小规模场景下使用 k" \æXḪ可以一键部署应用确实很方便，达到了一键部署的目的，但是当出现需要在几百台主机上进行多副本部署，需要管理这么多主机的运行状态以及服务的故障时需要在其他主机重启服务，想象一下就知道手动的方式不是一种可取的方案，这时候就需要利用 Ÿő6 XḫẹXṁẊḳ 这种更高维度的编排工具来管理了。Ÿő6 XḫẹXṁẊḳ 简称 Ÿ' ḁ， 简单说 Ÿ' ḁ 就是抽象了硬件资源，将 Ô 台物理机或云主机抽象成一个资源池，容器的调度交给 Ÿ' ḁ 就像亲妈一样照顾我们的容器，bĒń 不够用就调度到一台足够使用的机器上，内存不满足要求就会寻找一台有足够内存的机器在上面创建对应的容器，服务因为某些原因挂了， Ÿ' ḁ 还会帮我们自动迁移重启， 简直无微不至，至尊享受。我们作为开发者只关心自己的代码，应用的健康由 Ÿ' ḁ 保证。

这里就不介绍具体的安装方式了，如果使用 Ŷ ậl "Ţḳ 或者 Î \` ḁ 可以直接使用 k" \æXḪk XḳáṁḪ 下的 Ÿő6 XḫẹXṁẊḳ 选项一键安装单主机集群，也可以使用 ǽl 工具 在本地模拟多集群 Ÿ' ḁ。

Ÿ' ḁ 调度的基本单位为 Ḥ' l， 一个 Ḥ' l 表示一个或多个容器。引用一本书里所说

> 之所以没有使用容器作为调度单位，是因为单一的容器没有构成服务的概念；例如 Ŷ X6 应用做了前后端分例，需要一个 Ô" l X÷ḁ 与 Ķ' ȟ \ ṁ才能组成一个完整的服务，这样就需要部署两个容器来实现一个完整的服务，虽然也可以把他们都放到一个容器里，但这显然违反了一个容器即一个进程的核心思想 ǘǎ《ḁXḤlậXÎ XḳǕ实战 ǎ用 ậṁă 软负载实现服务网格》

Ÿ' ḁ 与传统 Æ ḁ 系统的不同'H

Æ ḁ 就是 ÆÑḤ ḳṁő \ ṁőḷX ḳ ḳXḤ ậX， 所谓基础设施即服务，开发者想要上线一个新应用需要申请主机，ậḤ， 域名等一系列资源，然后登录主机自行搭建所需环境，部署应用上线，这样不仅不利于大规模操作，而且还增加了出错的可能，运维或开发这常常自己写脚本自动化完成，遇到一些差异再手动修改脚本，非常痛苦。

▲ 赞同 ▼ 添加评论 分享 喜欢 收藏 ···

Ÿ g 则是将基础设施可编程化，由原来的人工申请改为一个清单文件自动创建，开发者只需要提交一份文件，Ÿ g 将会自动为你分配创建所需的资源。对这些设施的 bĖńk 都可以通过程序的方式自动化操作。

为了了解 Ÿ g 的基础概念，下面来部署一个 Ô" I Xgg Ė 应用'H

初始化应用模板

```
npm install create-next-app
npx create-next-app next-app
cd next-app
```

创建好工程后给添加一个 k" l æ X ¦ H 用来构建服务的镜像

k" l æ X ¦ H

```
FROM node:8.16.1-slim as build

COPY ./ /app

WORKDIR /app
RUN npm install
RUN npm run build
RUN rm -rf .git


FROM node:8.16.1-slim

COPY --from=build /app /

EXPOSE 3000
WORKDIR /app

CMD ["npm", "start"]
```

这个 k" l æ X ¦ H 做了两部分优化

[5] ñ 使用精简版的 ę" I X 基础镜像， 大大减少镜像体积
[6] ñ 使用分步构建的方式， 能够减少镜像层数以及移除临时文件从而减少了镜像体积。

构建镜像

```
docker build  . --tag next-app
```

之后我们就可以向 Ÿ ó 6 X ¦ ę X m k 提出我们应用的要求了。为了保证高可用，服务至少创建两个副本，我们还需要一个应用的域名当这个域名请求到我们集群上时自动转发到我们的服务上。那么我们对应的配置文件就可以这么写

k X ¦ Ë ü ň X ę m ů ň È

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: app-ingress
spec:
  rules:
  - host: next-app-server
    http:
      paths:
      - backend:
          serviceNa
```

▲ 赞同 7 · ▼ 添加评论 分享 喜欢 收藏 ···

```
          servicePort: 80

---
kind: Service
apiVersion: v1
metadata:
  name: app-service
spec:
  selector:
    app: web
  ports:
  - port: 80
    targetPort: 3000

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - image: next-app
        name: next-app
          imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 3000
```

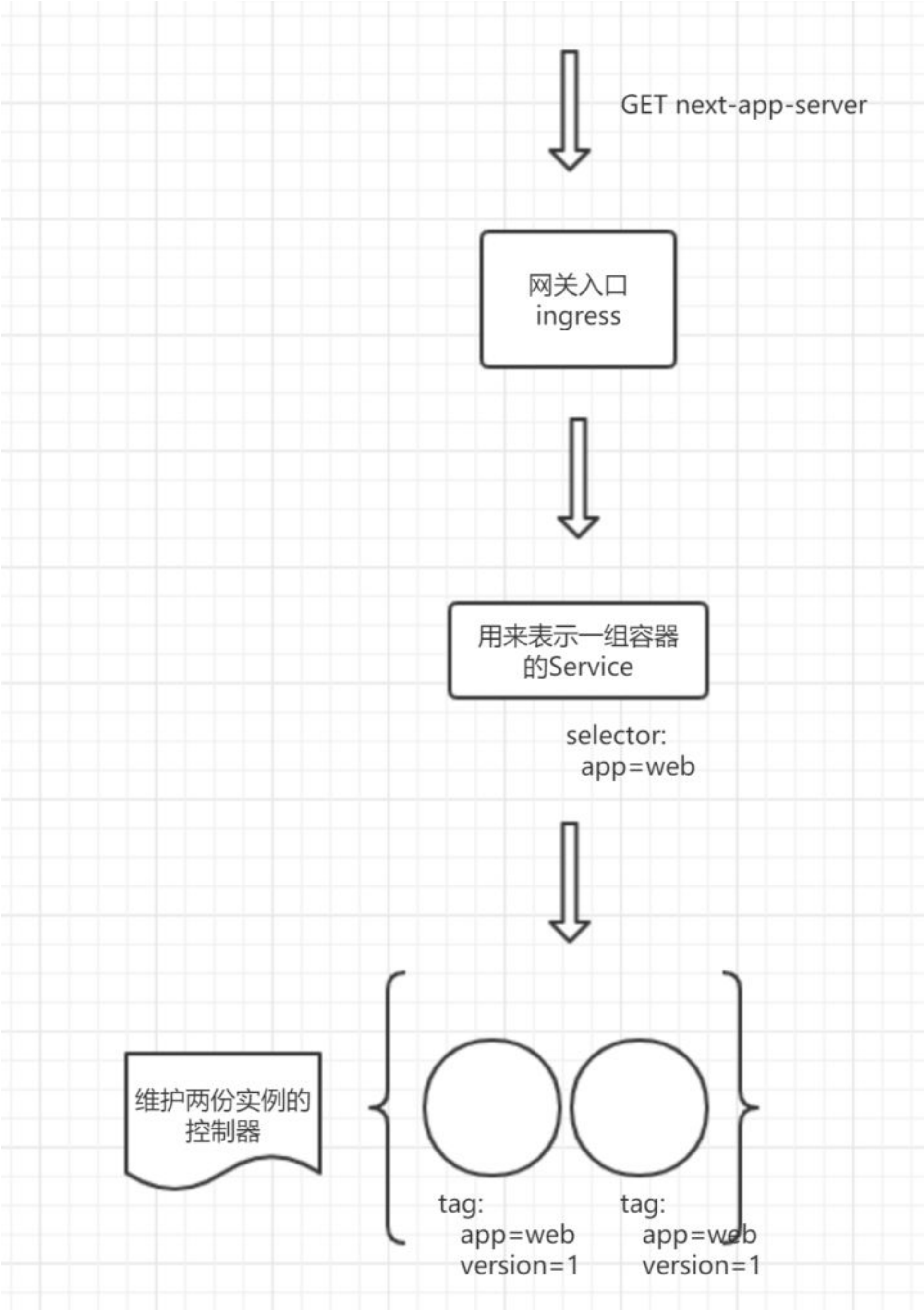
上面这个清单告诉 Ÿ' g'H

"A首先需要一个 k XŧĖüň Xęnł控制器，镜像为 `next-app`， 服务端口为 ⁷⁰⁰⁰，给我创建两个副本。
"A还需要创建一个 gXŧąX, 这个 gXŧąX 指向由副本控制器创建的几个 `next-app` 。
"A申请一个 ÆnJXk 入口， 域名为 `next-app-server`， 其指向刚刚的 gXŧąX。

提交这份申请给 Ÿ' g。

```
kubectl apply -f ./Deployment.yaml
```

▲ 赞同 ⁷ · ▼ 添加评论 分享 喜欢 收藏 ···

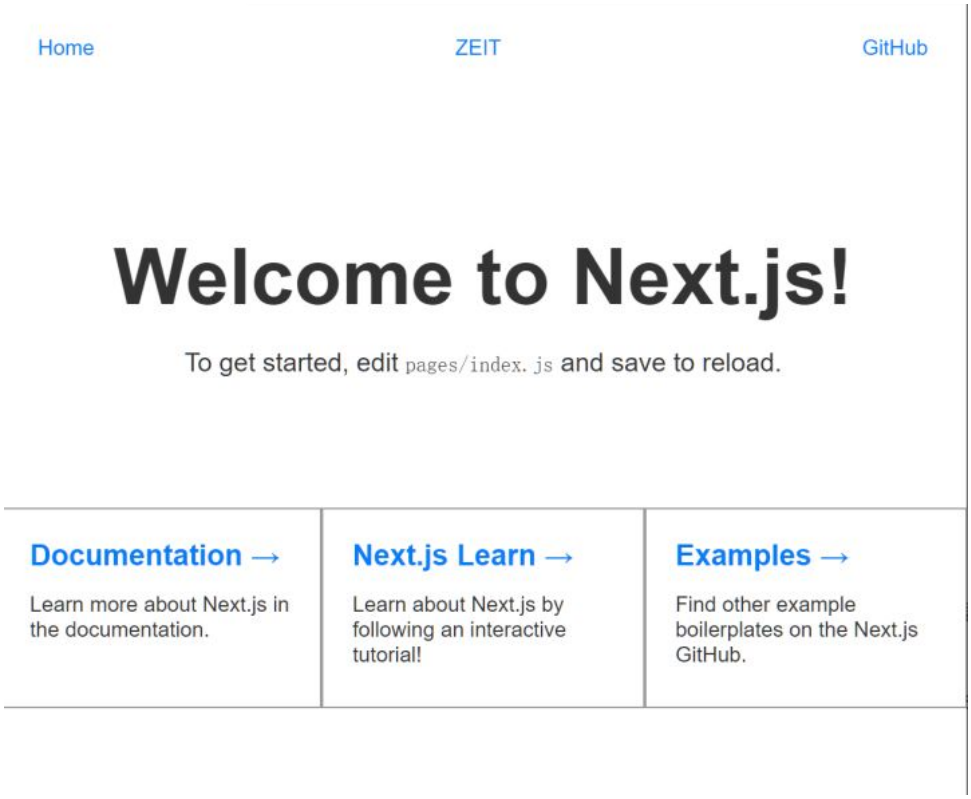


接着就可以看到已经部署的 Ḥ' l 。

```
sh-4.4$ kubectl get pod
NAME                               READY     STATUS    RESTARTS   AGE
app-deployment-594c48dbdb-4f4cg    1/1       Running   0          1m
app-deployment-594c48dbdb-snj54    1/1       Running   0          1m
```

然后浏览器打开 Ænj̥Xk̩ 里配置的域名即可访问对应的应用ʜ前提是这个域名能够打到你的 Ÿ ģ 集群节点上ɪ。

▲ 赞同 ▼ 添加评论 分享 喜欢 收藏 ···



上面的清单主要创建了三种最常见资源来保证服务的运行， 这也是 Ÿó6 XĻęXṁẋķ 的最主要的三类资源。

"AÆēnĮXķķ

‡ 层负载均衡配置， 可以根据不同的域名或者路径等信息指向不同的 gẊĻłåX， ÆēnĮXķķ 和 ÔnĝęÚ 很像，实际上 ÆēnĮXķķ 的一种实现就是 ÔnĝęÚ， 所以可以将 ÆēnĮXķķ 来当成 ÔnĝęÚ来用，只不过我们不需要手动修改 `nginx.conf` ，也不用手动重启 ÔnĝęÚ服务。

"AgẊĻłåX

一组 Ḥ"l 的抽象，用来选择提供同一服务的 Ḥ"l 。 由于 Ḥ"l 是不稳定的，销毁重建经常发生， Ḥ"l 的 åłł经常发生变化，所以需要一种抽象的资源 gẊĻłåX来表示 Ḥ"l 的位置。 gẊĻłåX也是Ÿ' g 内部服务发现机制，会自动将 gẊĻłåX名称写入内部 k Ôg 记录中。

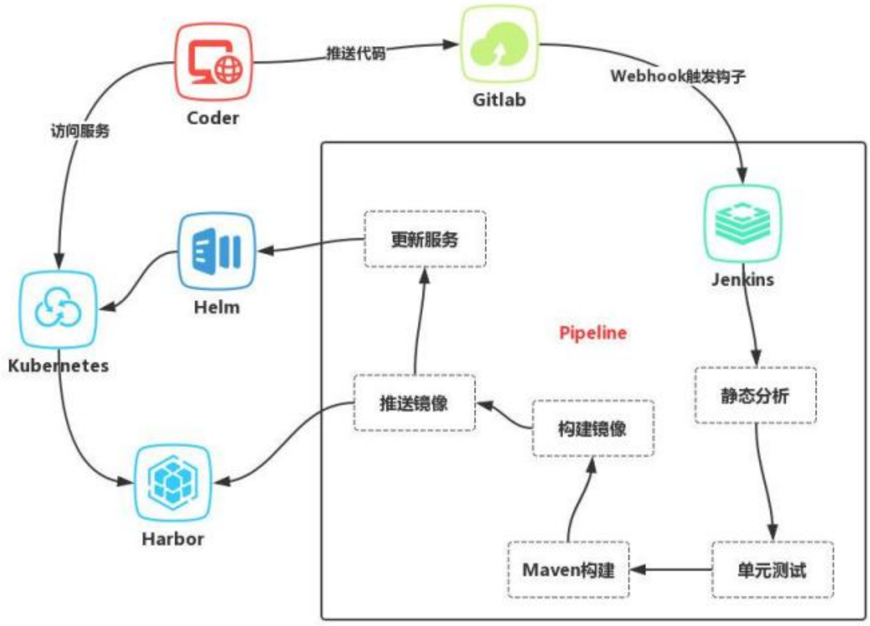
"Ak XłĖüň Xęṁ

副本控制器，用来管理维护 Ḥ"l 的一种机制。通过 k XłĖüň Xęṁ可以指定副本数量，发布策略， 记录发布日志并支持回滚。

应用发布系统

Ÿ' g 仅仅负责容器的编排，实际上如果部署应用还需要外部 ĒåłXẅęX 的支持，代码的构建，静态检查，镜像的打包由 ĒåłXẅęX 完成ñ

▲ 赞同 7 · ▼ 添加评论 分享 喜欢 收藏 ···

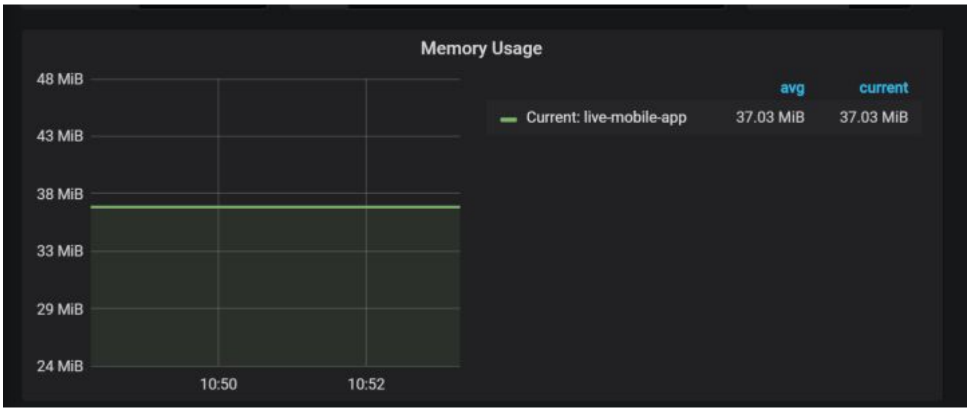


”H

目前国内用的比较多的发布系统常常由下面几个服务组成'Hб ąц ϭıõ ąŞ óϭἧ÷Xęąęķἧg”ę ļἧŞ ļϬ”Ḩ

Ÿ⁾ g 在前端的优势

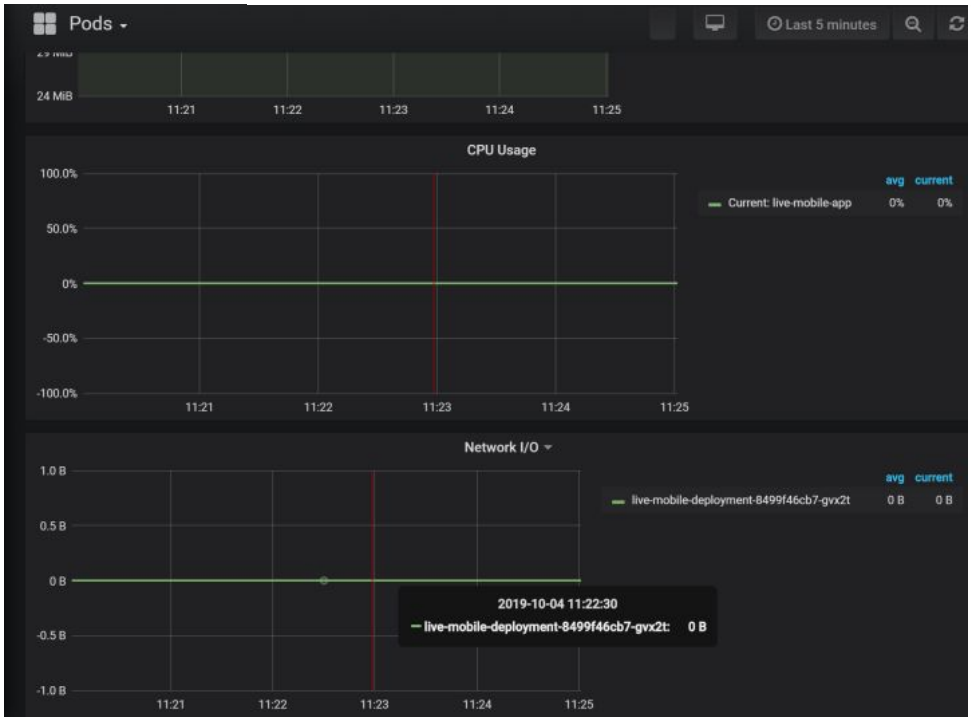
⁵ἧ首先前端应用和 ÷ ț 不同，一个小型 Ộ”I X÷g 服务占用内存仅 ⁸⁰Î 左右，这意味着如果我们有很多 Ộ”I X÷g 应用，使用 Ÿ⁾ g 将节省大量的硬件资源。



⁵ἧ使用容器的思想进行非侵入式日志，性能指标收集。

由于容器即是一个进程，所以对容器的监控可以看作对我们 Ộ”I X÷g 进程的监控，Ÿ⁾ g 生态里已经有很多成熟的容器监控方案，例如 ĒĻIἧ XἧXóķ "IõĻNę ， 使用此可以达到应用的非侵入式性能指标的收集包括'H网络Æı 磁盘Æı bĒἧ ıÎ tÎ 。

▲ 赞同 ▼ 添加评论 分享 喜欢 收藏 ···



同样对于日志收集，我们在代码中可以直接使用 `console` 的方式输出， 在容器维度再使用日志收集服务进行日志收集，同样的非侵入式， 代码层无感知，对开发者更加友好，将日志和服务解耦。

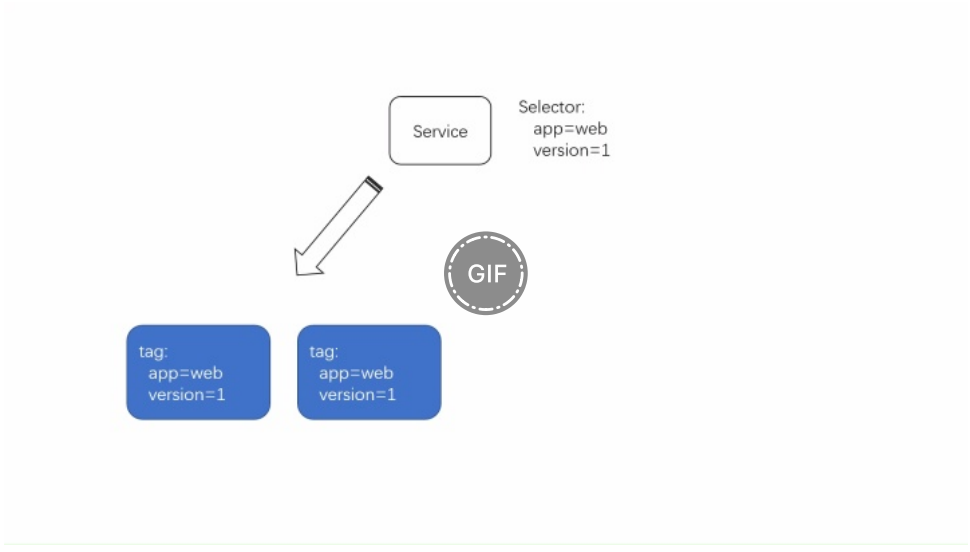
[5]ή前端微服务架构基础设施层。

微服务架构是近两年越来越流行的一种前端架构组织方式，微服务架构需要有一种更加弹性灵活的部署方式。 使用 k”\æ¼ħ让我们在复杂架构中抽象服务的最小单元，Ÿˀ g̨ 给自动维护大规模集群提供了可能。可以说微服务架构天然适合使用 Ÿˀ g̨。

Ÿˀ g̨ 新玩法， 流量分配

Ÿˀ g̨ 中使用 g̨X̧ŋāX来抽象一组 Ḥ’ I，而 g̨X̧ŋāX的选择器可以动态变更，所以了我们很多可能的玩法， 比如蓝绿发布系统。

蓝绿发布是指发布过程中新应用发布测试通过后，通过切换网关流量， 一键升级应用的发布方式， 在 Ÿˀ g̨ 中通过动态更新 g̨X̧ŋāX的选择器实现不同版本的一键切换

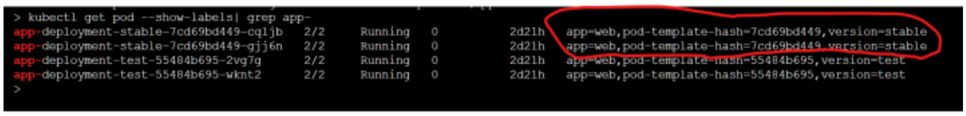


下面使用上面的 `Next.j`

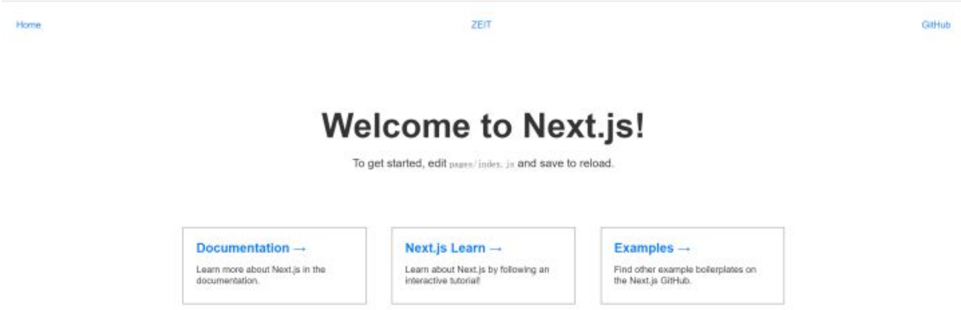
▲ 赞同 7 · ▼ 添加评论 分享 喜欢 收藏 ···

```
git clone https://github.com/Qquanwei/test-ab-deploy
cd test-ab-deploy
docker build . --tag next-app:stable
kubectl apply -f ./Deployment.yaml
```

这里会将 `next-app:stable` 这个镜像部署到集群中，并且给 Ḥ'I 打上 `version: stable` 的 ṁnj。



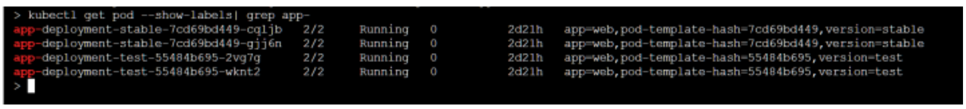
部署后打开显示如下。



接着，我们部署 mẊkṁ分支，这个分支我们会构建为 `next-app:test` 的镜像，并且部署时给这个 Ḥ'I 打上 `version: test` 的标签。

```
git checkout test
docker build . --tag next-app:test
kubectl apply -f ./Deployment.yaml
```

这时候我们一共部署了两个版本的应用，而且都已经就绪状态。



但是由于我们的 gẊḤ ậX为 `version=stable`， 所以所有的请求并不会打到 mẊkṁ版本上，仍然都会请求 ḳṁ6臣的服务。

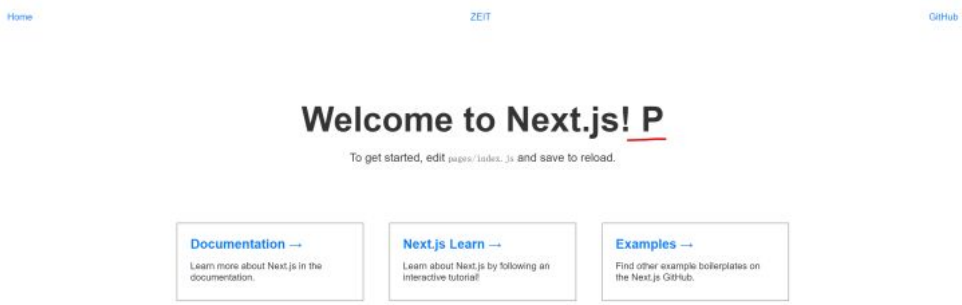
▲ 赞同 ▼ 添加评论 分享 喜欢 收藏 ···

```
> kubectl describe svc app-service
Name:              app-service
Namespace:         default
Labels:            cattle.io/creator=norman
Annotations:       field.cattle.io/creatorId: user-l7qkh
                   field.cattle.io/ipAddresses: null
                   field.cattle.io/targetDnsRecordIds: null
                   field.cattle.io/targetWorkloadIds: null
Selector:          app=web,version=stable
Type:              ClusterIP
IP:                10.43.81.3
Port:              http  80/TCP
TargetPort:        3000/TCP
Endpoints:         10.42.6.57:3000,10.42.7.134:3000
Session Affinity:  None
Events:            <none>
>
```

当我们用其他的方式已经验证 mXkm版本服务可用时， 例如配另外一个 gXtlaX用来测试ho""lı，这时候可以下面一条指令切换当前的 gXtlaX到 mXkm应用上。

```
kubectl apply -f ./switch-to-test.yaml
```

执行完这条命令后，刷新页面可以看到如下。

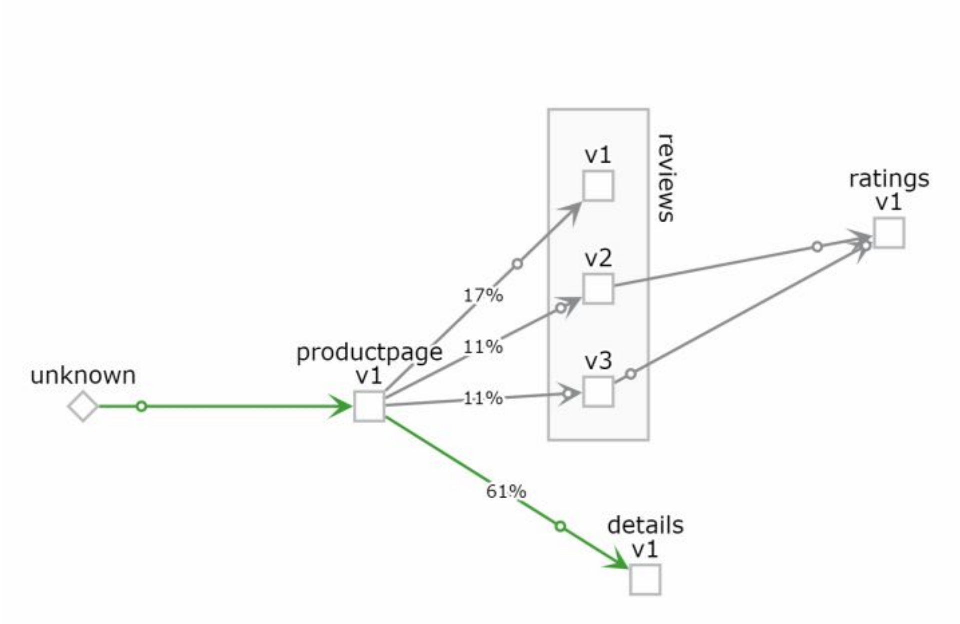


通过切换 gXtlaX的方式很轻松就实现了蓝绿发布的功能，而且是瞬间完成，因为 gXtlaX是 Ÿ g 里比较轻量的资源，不会和隔壁 ÔnjęÚ一样修改配置就要重启服务影响整个线上服务。当然实际生产环境会比演示更加严谨，可能有专门的平台以及审核人员进行每个操作的二次验证。

对于蓝绿， 灰度发布方式，使用 Ÿ g 可以较为轻松地实现，让我们能够有更多的方式去验证想法。不过如果想实现更加高级的流量分配方案（例如2ı^发布），需要复杂的流量管理策略 （鉴权，认证），就需要用到服务网格了。

Æŗā 是目前比较流行的服务网格框架，相比于 Ÿ g 注重运行容器的管理， Æŗā 则是更注重容器之间组成的服务网格的流量传输。

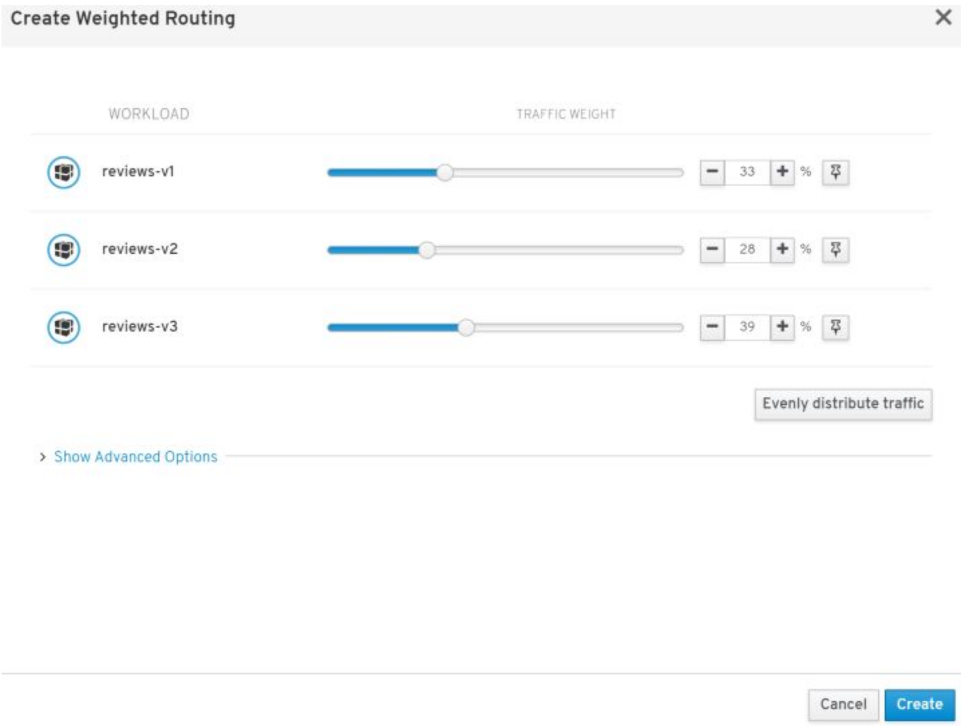
下图是 Æŗā 捕获的官方示例的 `bookinfo` 微服务中服务的拓扑结构和一些数据指标。



使用 Ąꜩă 有两个明显的好处'H

⁵ ῆĄꜩă 能够捕捉到服务间的调用链路，而且不入侵用户代码。
⁶ ῆĄꜩă 能够对每一条连接，进行单独的管理。

例如，我们可以轻松对的不同版本的 ꞩꭗ ăXꞀ 应用的 ṭ⁵， ṭ⁶， ṭ⁷ 版本进行动态权重分配。



不仅仅可以对流量权重分配，而且还可以制定一些2ı^方案，例如根据 ńĖ‡ 是否匹配请求不同的版本应用，或者根据 §X I Xꞩ种下的 b” ” ăꭗ 进行用户的区分，从而请求不同的应用。当然，面对行业场景不同，Ąꜩă 还会诞生很多有趣的玩法。

不过缺点同样存在，Ąꜩă 实际上也是一个很复杂的系统，会对性能造成影响，而且会占用不小的系统资源。

总结

▲ 赞同 ⁷ ▼ 添加评论 分享 喜欢 收藏 ···

Ÿ g 是划时代的，随着未来的发展微服务化，云原生化将会是我们的应用的主要形式，对于前端而言 Ÿ g 无疑会改变现有前端的开发方式和前端架构，让前端能够更迅速地扩展，更稳定地交付，应用之间的联系也会愈加紧密。沉寂已久的前端下一个三年相信将会是微服务架构的天下，Ÿ g 作为微服务架构基础设施层也将会被越来越多的公司团队所重视。

参考资料

ˇA《k" ˥æX̧H容器与容器云》
ˇA《Ÿ6X̧ȩXmX̧ ǟ 2 ˥ṝ ȩ》
ˇA《ģX̧Ḩ ǡ X Î X̧Ǖ实战 ά用 ǡ̧ ṝ 软负载实现服务网格》
ˇA阿里云 安装 k" ˥æX̧H̶ Ȅ n ǹ kl ȩ̄ ȩ Xm Ȅ ȩn Ǘ ȩn ǘṁ

> 本文发布自 网易云音乐前端团队，文章未经授权禁止任何形式的转载。我们一直在招人，如果你恰好准备换工作，又恰好喜欢云音乐，那就 加入我们！

编辑于 ⁶ ⁰ ⁶ ⁰ ǚ ⁵ ǚ ⁿ

k" ˥æX̧H　Ÿ6X̧ȩXmX̧　容器（虚拟化）

文章被以下专栏收录

云音乐前端技术团队专栏

关注专栏

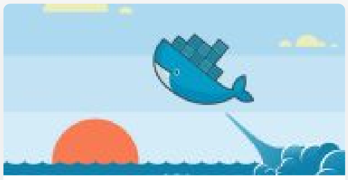
推荐阅读

再也不用担心学不会Ÿ g！⁵⁽ 个Ÿ g初学者必须掌握的知识点

ôǟb Ǖ ṁ

ǽ6X̧ȩXmX̧ 上手指南：概念篇

谢伟　发表于『õ" ḤǕṁ



⁵⁰ 分钟看懂k" ˥æX̧和Ÿ g

小枣君　发表于鲜枣课堂

从零开始网络概念

一、Ÿ6X̧ 来介绍一下 的一些想法 对于网络是 限制，也没 例。Ÿ6X̧

Î k̡Xṁ

还没有评论

写下你的评论ṁ

▲ 赞同 ⁷ ·　▼　添加评论　分享　喜欢　收藏　···