Changelog

2021.11.10

1. Ingress apiVersion的变化。从Kubernetes v1.8+ 开始，apiVersion变更为networking.k8s.io/v1beta1。废弃extensions/v1beta1。
2. 结构体的变化。移除了spec.backend字段；backend下的service也会接着细化。主要修改为：
   - 移除了spec.backend字段
   - backend.serviceName，backend.servicePort 更换为了backend.service.name，backend.service.port.number
3. 新增了必填字段：pathType。其功能作用是精确匹配和前缀匹配的区别。

例如 `path: /wss，pathType: Prefix，等同于path: /wss*` 。具体详细内容：kubernetes.io/zh/docs/con...

4. 更新 ingress 版本到最新 stable 1.0.4 版本，原有的镜像源会失效。我也没找到好的镜像，希望大家如果有挖掘可以提供

前言

在上一章，我们部署了一套 `Kubernetes` 集群环境，这一章我们就来部署自己的第一个 `Kubernetes` 应用并实现访问。

声明一份配置清单

在开始部署前，我们先要声明一份 `配置清单` ，清单的文件格式为 `YAML` 文件格式。在 Kubernetes 中，应用部署完全可以通过 `YAML` 配置清单来进行部署。

新建一个文件夹，名称叫 `deployment` ，并在文件夹内创建一份 `yaml` 文件，名称为 `v1` ：

shell 复制代码

```shell
mkdir deployment && cd deployment
vim v1.yaml
```

接着在配置文件中，写入以下内容：

yaml 复制代码

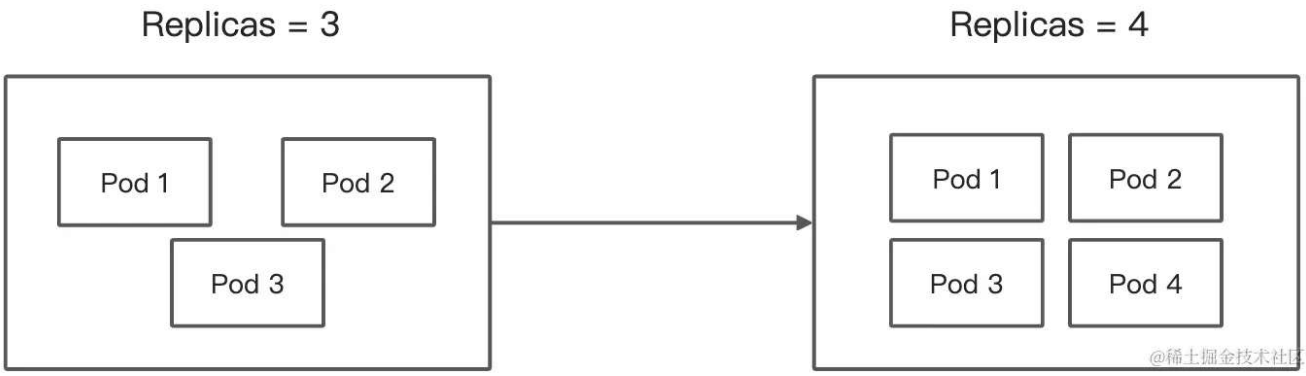
```yaml
metadata:
  name: front-v1
spec:
  selector:
    matchLabels:
      app: nginx-v1
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx-v1
    spec:
      containers:
      - name: nginx
        image: registry.cn-hangzhou.aliyuncs.com/janlay/k8s_test:v1
        ports:
        - containerPort: 80
```

我们关注下 YAML 文件中的 kind 字段。这是在声明 Kubernetes 的资源类型。在这里，我们的 `kind` 值为 `deployment` 。那 `deployment` 又是什么呢?

什么是 Deployment

如果你将 `k8s` 看作是一个大型机场，那么 `deployment` 刚好就是机场内的**停机坪**。

根据飞机的种类进行划分停机坪，不同的停机坪都停着不同类型的飞机。只不过， `deployment` 要比停机坪还要灵活，随时可以根据剩余的空地大小（服务器剩余资源）和塔台的指令，增大/变小停机坪的空间。**这个“增大变小停机坪空间的动作”，在k8s中就是 `deployment` 对它下面所属容器数量的扩容/缩小的操作。**

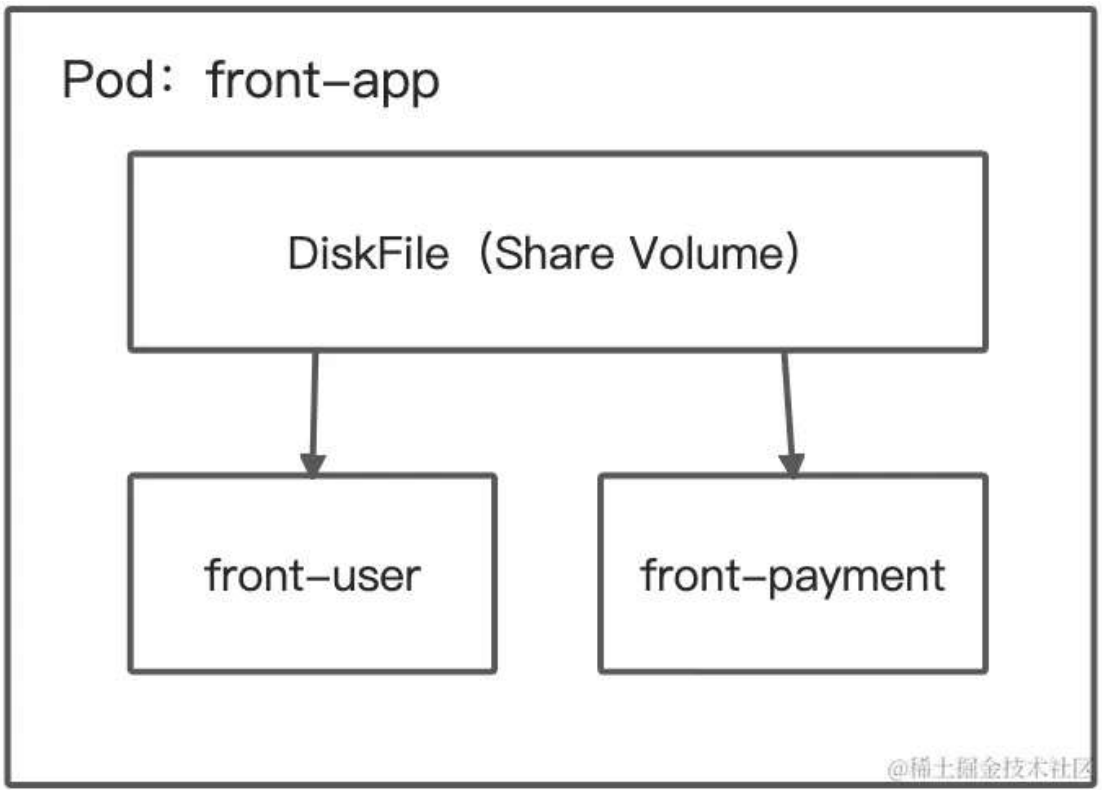


那么这也就代表， `deployment` **是无状态的，也就不会去负责停机坪中每架飞机之间的通信和组织关系。**只需要根据塔台的指令，维护好飞机的更新和进出指令即可。**这个根据指令维护飞机更新和进出的行为，在k8s中就是 `deployment` 对他下面的容器版本更新升级，暂停和恢复更新升级的动作。**

在这里的**容器**，并不等于 Docker 中的容器。它在K8S中被称为 `Pod` 。那么 `Pod` 是什么？

什么是 Pod

但这个 IP 会随着Pod的重启，创建，删除等跟着改变，所以不固定且不完全可靠。这也就是 Pod 的 IP 漂移问题。这个问题我们可以使用下面的 Service 去自动映射



我们经常会把 Pod 和 Docker 搞混，这两者的关系就像是豌豆和豌豆荚，Pod 是一个容器组，里面有很多容器，容器组内共享资源。

分析配置文件构成

那么相信大家对 `deployment` 有大体的概念了。当然， `kind` 字段不只可以声明 `deploymnt` ，还可以声明其他的资源类型。重要的我们在后面的章节中都会写到。

了解了 `deployment` 是啥后，我们来看看配置清单中的字段都代表的是啥。我们将配置分成三段去进行阅读：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: front-v1
```

最上面的第一段声明了当前资源配置的 API 版本，资源类型和资源名称：

- API 配置版本： `apps/v1`
- 资源类型： `deployment`
- 资源名称： `deplyment` 的名称叫 `front-v1`

> 该怎么选择 apiVersion 的值: matthewpalmer.net/kubernetes-...

```
spec:
  selector:
    matchLabels:
      app: nginx-v1
```

```
replicas: 3
template:
  metadata:
    labels:
      app: nginx-v1
  spec:
    containers:
    - name: nginx
      image: registry.cn-hangzhou.aliyuncs.com/janlay/k8s_test:v1
      ports:
      - containerPort: 80
```


左边这一段，告诉 `deployment` 我根据规则匹配相应的 `Pod` 进行控制和管理。这里使用 `matchLabels` 字段匹配 `Pod` 的 `label` 值。

右边配置则代表声明一个 Pod 组：

- replicas：要创建的 `Pod` 最大数量。数字类型
- labels.app：Pod 组的名称
- spec：组内创建的 Pod 信息
  - name：Pod 名称
  - image：以什么镜像创建 Pod。这里是 Docker 镜像地址
  - ports.containerPort：Pod 内容器映射的端口

这里的镜像，我使用了自己编译的一份 nginx 镜像作为演示，也可以换成你自己的镜像

启动第一个应用

好了，在我们了解完一份简单的 `deployment` 的配置清单后，我们就可以使用该清单创建我们的第一个应用。

在k8s中，我们使用 `kubectl apply` 来执行一份k8s的配置：

```shell
```

其中， `kubectl apply` 代表准备对资源进行配置。 `-f` 等于 `--filename` ，后面可以跟随多个配置文件。例如：

shell 复制代码

```shell
kubectl apply -f ./v1.yaml ./v1-service.yaml ./v1-ingress.yaml
```

当提示下面文字时，代表配置文件执行成功：

```
[root@master deployment]# kubectl apply -f ./v1.yaml
deployment.apps/front-v1 configured
[root@master deployment]#
```


如果你想看部署完毕后的 `Pod` 运行状态，可以使用 `kubectl get pod` 命令来获取所有 Pod 的信息：

shell 复制代码

```shell
kubectl get pod
```

你会得到一个表格，这是 你自己在 K8S 中部署的所有的Pod。

其中，name 是Pod的名称；READY 为容器状态，格式为可用容器/所有容器数量；STATUS 为 Pod 的运行状态；RESTARTS 为重启数量；AGE 为 Pod 运行时间；当状态都是 `Running` 时，代表 Pod 运行正常。

```
[root@master deployment]# kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
front-v1-5846db7d5b-gr9b5   0/1     Pending   0          3m16s
front-v1-5856484cc-clbx2    1/1     Running   0          6m12s
front-v1-bdfd88666-d24fz    1/1     Running   0          12m
front-v1-bdfd88666-rk6tj    1/1     Running   0          12m
[root@master deployment]#
```


令人费解的无状态

部署成功了，但怎么去访问具体应用呢?

前面我们写到， `deployment` 是无状态的。也就意味着， `deployment` 并不会对 `pod` 进行网络通信和分发。想访问服务，有以下两个办法：

1. 直接访问具体的 `Pod` ：这是一个办法，但是 `Pod` 太多了，达不到我们自动调度的效果。且 `Pod` 的 `IP` 在运行时还会经常进行漂移且不固定（后面会讲到）。
2. 使用 `Service` 组织统一的 `Pod` 访问入口。

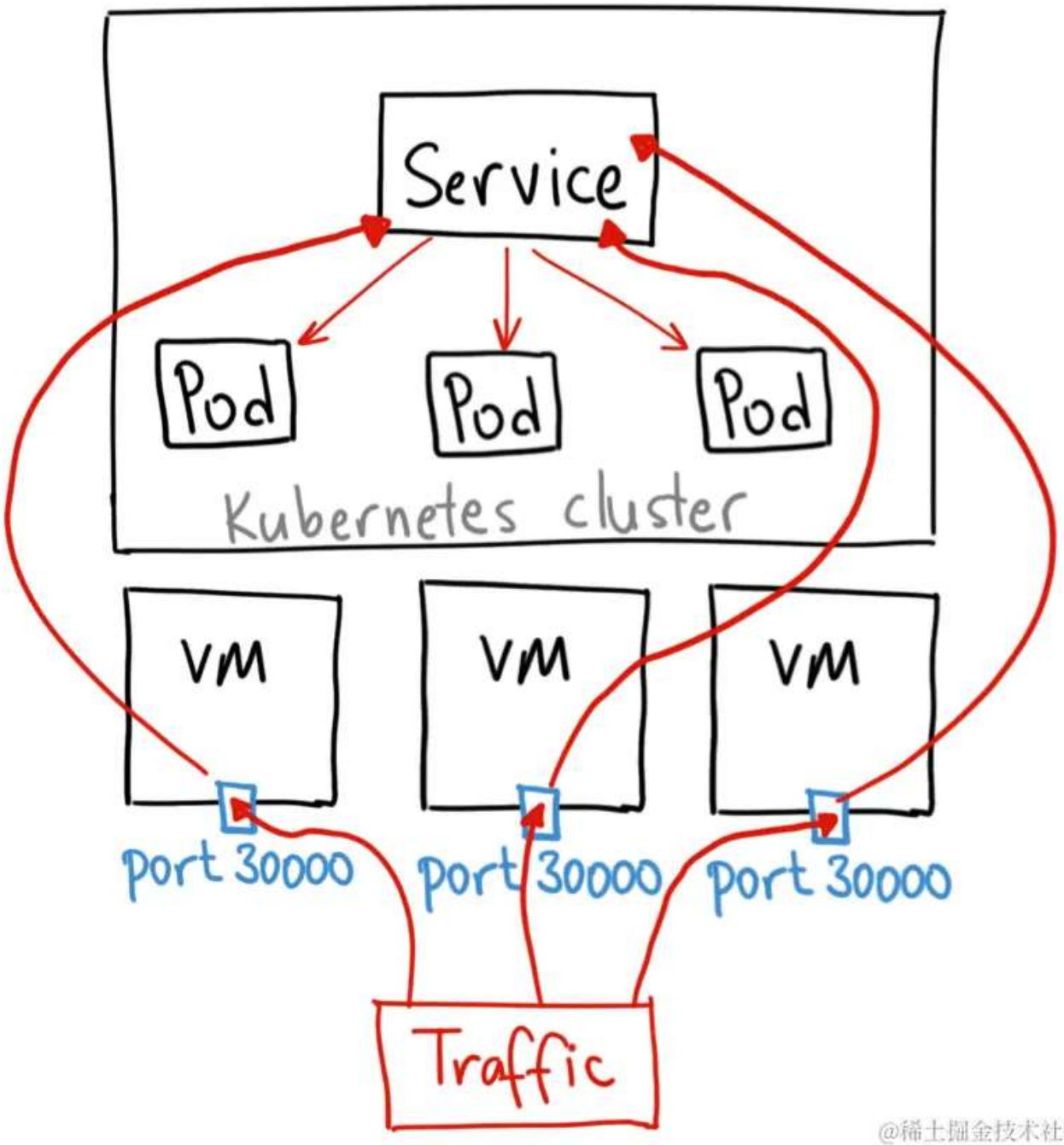
这里我们选择另一种资源类型 —— `Service` 来进行统一组织 `Pod` 服务访问

这里我们使用 k8s 的 Service 来组织我们的访问入口。那什么是 Service?

什么是 Service

`deployment` 是停机坪，那么 `Service` 则是一块停机坪的统一通信入口。它**负责自动调度和组织deployment中Pod 的服务访问。由于自动映射 Pod 的IP，同时也解决了 Pod 的IP漂移问题。**

下面这张图就印证了 `Service` 的作用。流量会首先进入 VM（主机），随后进入 Service 中，接着 Service 再去将流量调度给匹配的 Pod 。



Service 的配置

yaml 复制代码

```yaml
apiVersion: v1
kind: Service
metadata:
  name: front-service-v1
spec:
  selector:
    app: nginx-v1
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: NodePort
```

其中比较熟悉的通用字段就不介绍了。有几个特有的字段需要关注下：

字段	解释
protocol	通信类型（TCP/UDP）
targetPort	原本 Pod 开放的端口
port	k8s 容器之间互相访问的端口
type	NodePort，Service的一种访问方式

在这里，Service的模式我们选择使用 `NodePort` 模式。其他模式可以参考：www.dockerone.com/article/488...

与 Deployment 配置文件合并

根据YAML语法，我们可以将Service和deployment合并为同一个配置文件。当然，新建一个文件也是可以的。 我们编辑原有的v1.yaml，在文件底部添加 `---` 继续拼接Service的配置：

shell 复制代码

```shell
vim ./v1.yaml
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: front-v1
spec:
  selector:
    matchLabels:
      app: nginx-v1
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx-v1
    spec:
     # hostNetwork: true
     containers:
      - name: nginx
        image: registry.cn-hangzhou.aliyuncs.com/janlay/k8s_test:v1
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: front-service-v1
spec:
  selector:
    app: nginx-v1
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: NodePort
```


编辑保存退出后，使用 `kubectl apply` 命令来更新配置：

shell 复制代码

```shell
kubectl apply -f ./v1.yaml
```

此时，Service 已经部署完毕。

查看 Service 的访问端口

在部署成功 `Service` 后，我们可以使用 `kubectl get svc` 来获取我们已经部署的 `Service` 列表

我们可以使用 `kubectl get svc` 去查看下具体打开的服务端口：

shell 复制代码

```shell
kubectl get svc
```

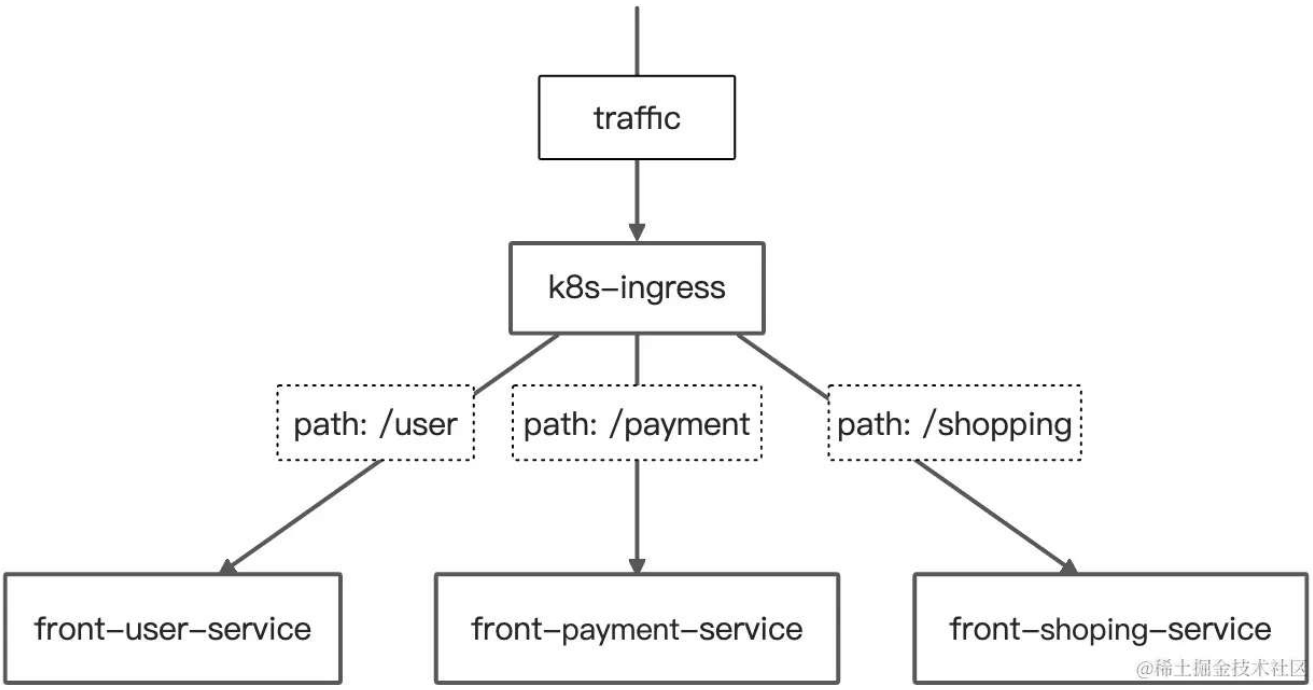
```
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
front-service-v1  NodePort    10.106.251.47   <none>        80:31048/TCP   3h25m
kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP        6h11m
[root@master deployment]#
```


其中， `PORT` 字段代表 `Service` 的访问端口。 `:` 前为映射到Pod的端口，31048 为访问端口。 我们访问 `Master节点IP + 端口` ，就可以访问到该服务。

ingress 是什么

在前面，我们部署了 `deployment` 和 `Service` ，实现了对服务的访问。但是在实际使用中，我们还会根据请求路径前缀的匹配，权重，甚至根据 `cookie/header` 的值去访问不同的服务。为了达到这种**负载均衡**的效果，我们可以使用 `k8s` 的另一个组件 —— `ingress`

在日常开发中，我们经常会遇到**路径分流**问题。例如当我们访问 `/a` 时，需要返回A服务的页面。访问 `/b` ，需要返回服务B的页面。这时候，我们就可以使用 `k8s` 中的 `ingress` 去实现。



在这里，我们选择 `ingress-nginx` 。 `ingress-nginx` 是基于 `nginx` 的一个 `ingress` 实现。当然也可以实现正则匹配路径，流量转发，基于 `cookie header` 切分流量（灰度发布）。

部署 ingress

首先进入 `master` 节点，下载 `ingress` 配置文件：

```
shell 复制代码
```

接着编辑下部署文件，将 `ingress` 的 `nodePort` 端口改为 `31234` ，以方便后面访问：

shell 复制代码

```shell
vim ./deploy.yaml
```

在下图所示位置添加 `nodePort` 字段为 `31234` ，https为 `31235` 。

Hyper

```yaml
---
# Source: ingress-nginx/templates/controller-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    helm.sh/chart: ingress-nginx-2.11.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.34.1
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  type: NodePort
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
    - name: https
      port: 443
      protocol: TCP
      targetPort: https
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/component: controller
---
```


接着执行命令使 `ingress` 生效：

shell 复制代码

```shell
kubectl apply -f deploy.yaml
```

接下来会自动拉取 `ingress` 镜像，自动部署 `ingress` 。可以使用 `kubectl` 命令查看部署状态：

如果显示以下信息，则代表部署成功。

```
[root@master deployment]# kubectl get pods -n ingress-nginx \
>   -l app.kubernetes.io/name=ingress-nginx --watch
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-2cnxf        0/1     Completed   0          5h24m
ingress-nginx-admission-patch-6kcsq         0/1     Completed   0          5h24m
ingress-nginx-controller-6b6497d95d-hkdr4   1/1     Running     0          5h25m
```


输入以下命令，检查配置是否生效：

shell 复制代码

```
kubectl -n ingress-nginx get svc
```

如果看到以下信息，代表生效：

```
[root@master ~]# kubectl -n ingress-nginx get svc
NAME                                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
ingress-nginx-controller             NodePort    10.97.150.187   <none>        80:31234/TCP,443:31235/TCP   8h
ingress-nginx-controller-admission   ClusterIP   10.101.226.10   <none>        443/TCP                      8h
```


配置 ingress

初识配置文件

同样的， `ingress` 服务的配置也是使用 `yaml` 文件进行管理。

我们新建一个 `ingress` 文件夹，将 `ingress` 的配置放在里面：

shell 复制代码

```
mkdir ingress && cd ingress && vim base.yaml
```

拷贝以下内容进去：

yaml 复制代码

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-demo
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - http:
      paths:
```

```
15            service:
16              name: front-service-v1
17              port:
18                number: 80
```

这是一份简单的 `ingress` 配置文件。配置主要分三部分：

annotations

`annotations` 是 `ingress` 的主要配置项目，可以用来修改这些配置来修改 `ingress` 的行为。我们可以通过修改这些配置来实现灰度发布，跨域资源，甚至将 `www.abc.com` 重定向到 `abc.com` 。

具体详细配置解释，可以翻阅官网文档：kubernetes.github.io/ingress-ngi...

yaml 复制代码

```yaml
annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
```

rules

`rules` 是 `ingress` 配置路径转发规则的地方。 `path` 可以是一个路径字符串，**也可以是一个正则表达式。** `backend` 则是 `k8s` 的 `service` 服务， `serviceName` 是服务名称， `servicePort` 是服务端口。

当我们去访问 `/wss` 时， `ingress` 就会帮我们调度到 `front-service-v1` 这个 `service` 上面。

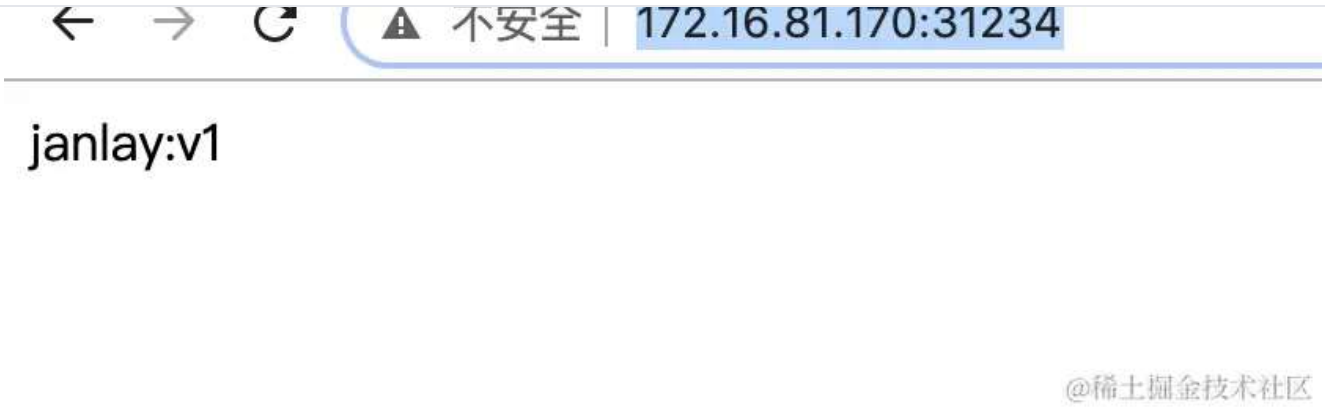
yaml 复制代码

```yaml
rules:
  - http:
      paths:
       - path: /wss
         pathType: Prefix
         backend:
           service:
             name: front-service-v1
             port:
               number: 80
```

然后执行命令，使配置项目生效：

shell 复制代码

```shell
kubectl apply -f ./base.yaml
```



结尾

到这里，我们就成功地部署了自己的第一个Kubernetes应用，并实现了访问。

但是在实际开发中，我们还需要零宕机发布，设置灰度环境等需求。下一章我们就会讲解下，如何使用 `Kubernetes` 配置你自己的灰度和滚动发布环境

上一章 下一章

留言

输入评论（Enter换行，Ctrl + Enter发送）

发表评论

全部评论（27）

晓之初 LV.2 JY.4 前端 10月前

【云服务器部署】都不想说的，但是还是给后来者填点坑，关于ingress
1. down下来deploy.yaml 后，手动改一下里面的镜像。 1.23.6的k8s可以用网上多数 1.1.1 的ingress版本。如果你忘记了改镜像直接就apply，那么恭喜你陷入删除ns的操作了。
先删ingress-nginx 这个namespace，再apply。
2. ingress-nginx-controll 的service配置那里改一下type: NodePort，拉下来的心的配置文件不是这个类型

点赞 1 ···

amandakel... 10月前

老哥这神预测.....我陷入删除ns搞了很久，一环扣一环
然后新拉下来的配置，type都是LoadBalancer，不知道信谁的

点赞 回复 ···