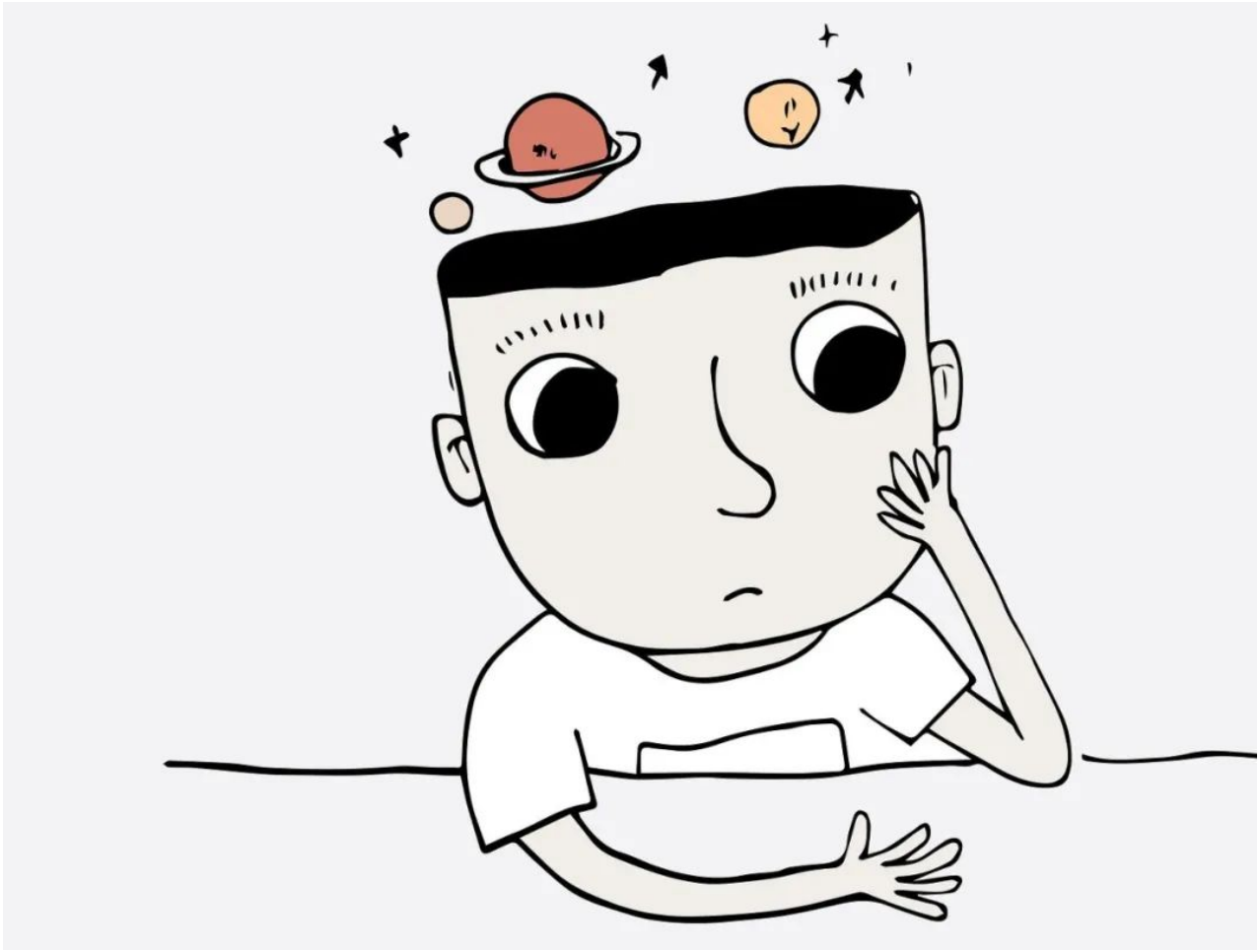
k8s Traefik简介与部署

mp.weixin.qq.com/s/UL0IE_AG7uU5dIOOMpypKQ



什么是Traefik

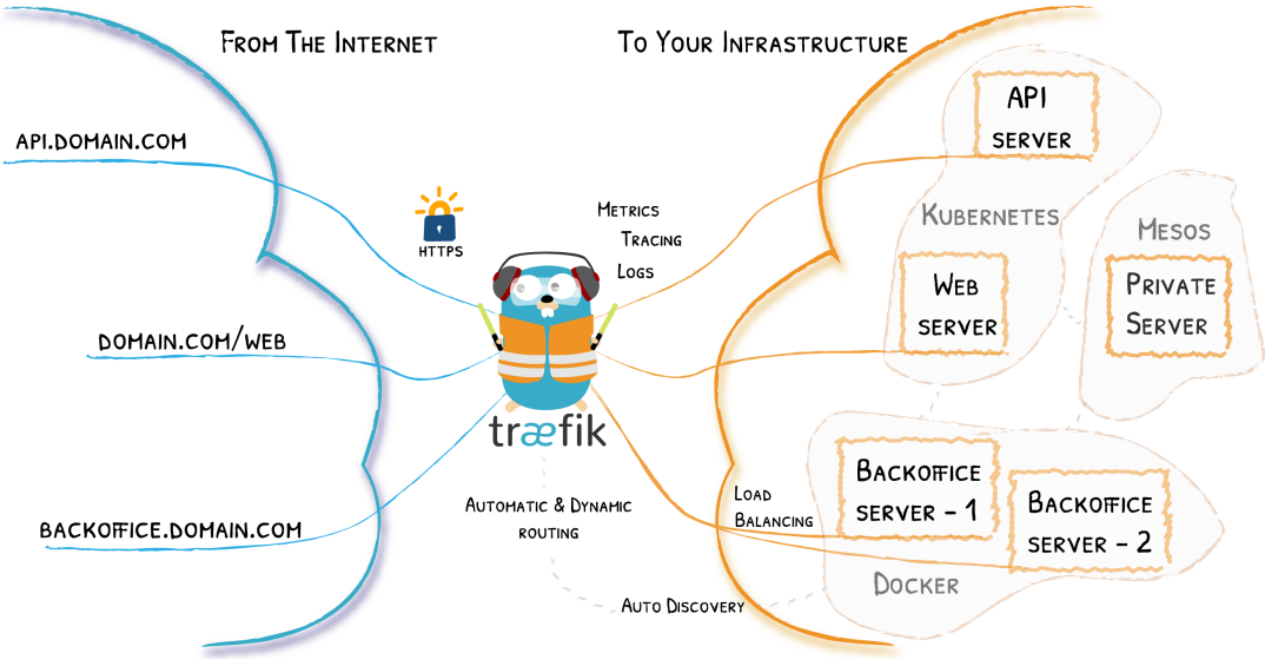
1. 官方文档

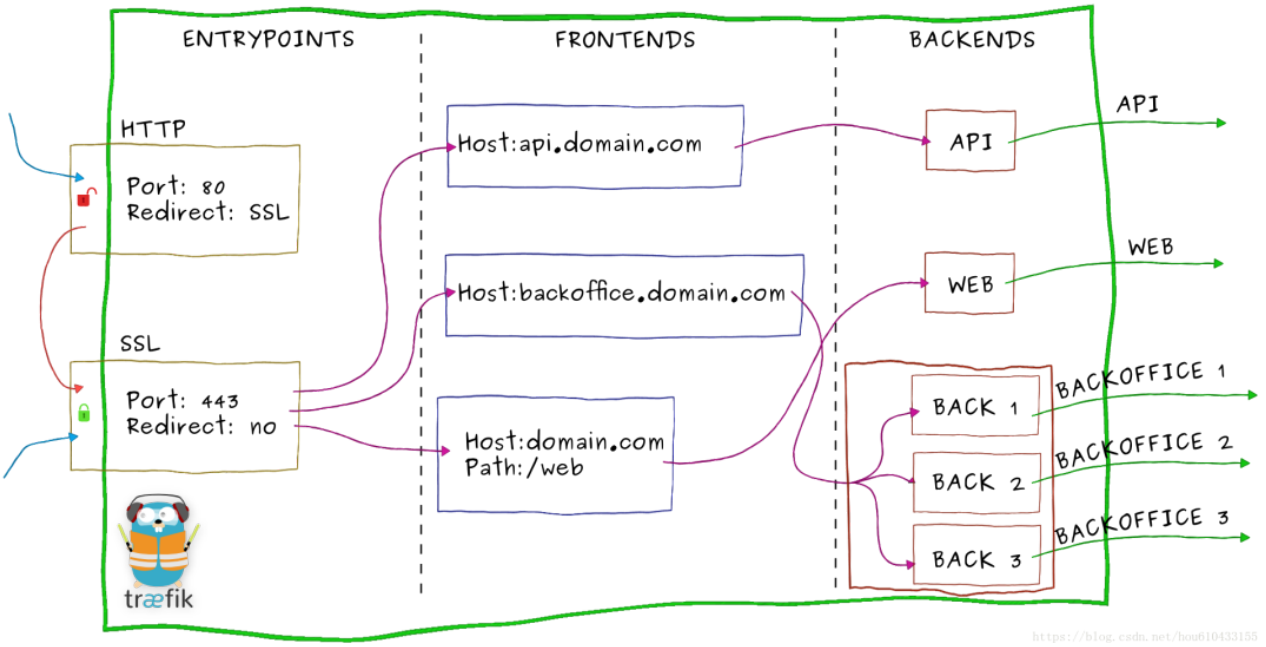
https://doc.traefik.io/traefik/

2. 简介

Traefik是一个为了让部署微服务更加便捷而诞生的现代HTTP反向代理、负载均衡工具。它支持多种后台 (Docker, Swarm, Kubernetes, Marathon, Mesos, Consul, Etcd, Zookeeper, BoltDB, Rest API, file...) 来自动化、动态的应用它的配置文件设置。

3. 流量示意图

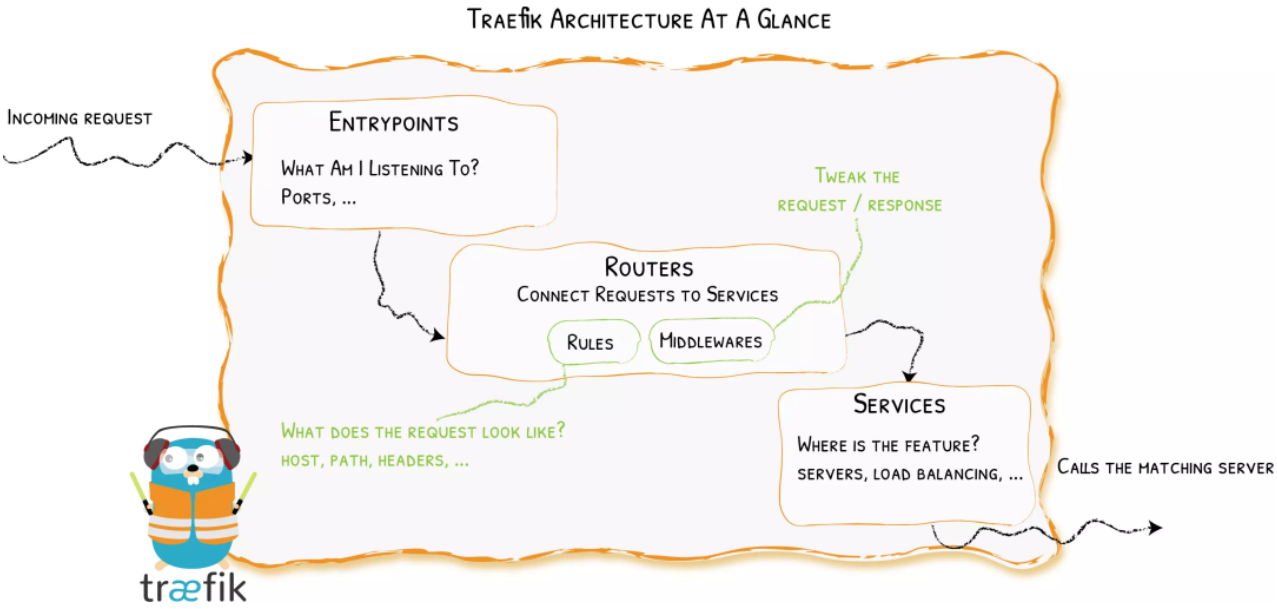




4. 核心概念

当启动Traefik时，需要定义 `entrypoints` ，然后通过entrypoints的路由来分析传入的请求，来查看他们是否是一组规则匹配，如果匹配，则路由可能将请求通过一系列的转换过来在发送到服务上去。

- `Providers` 用来自动发现平台上的服务，可以是编排工具、容器引擎
- `Entrypoints` 监听传入的流量，是网络的入口点，定义了接受请求的端口(HTTP或者TCP)
- `Routers` 分析请求(host,path,headers,SSL等)，负责将传入的请求连接到可以处理这些请求的服务上去

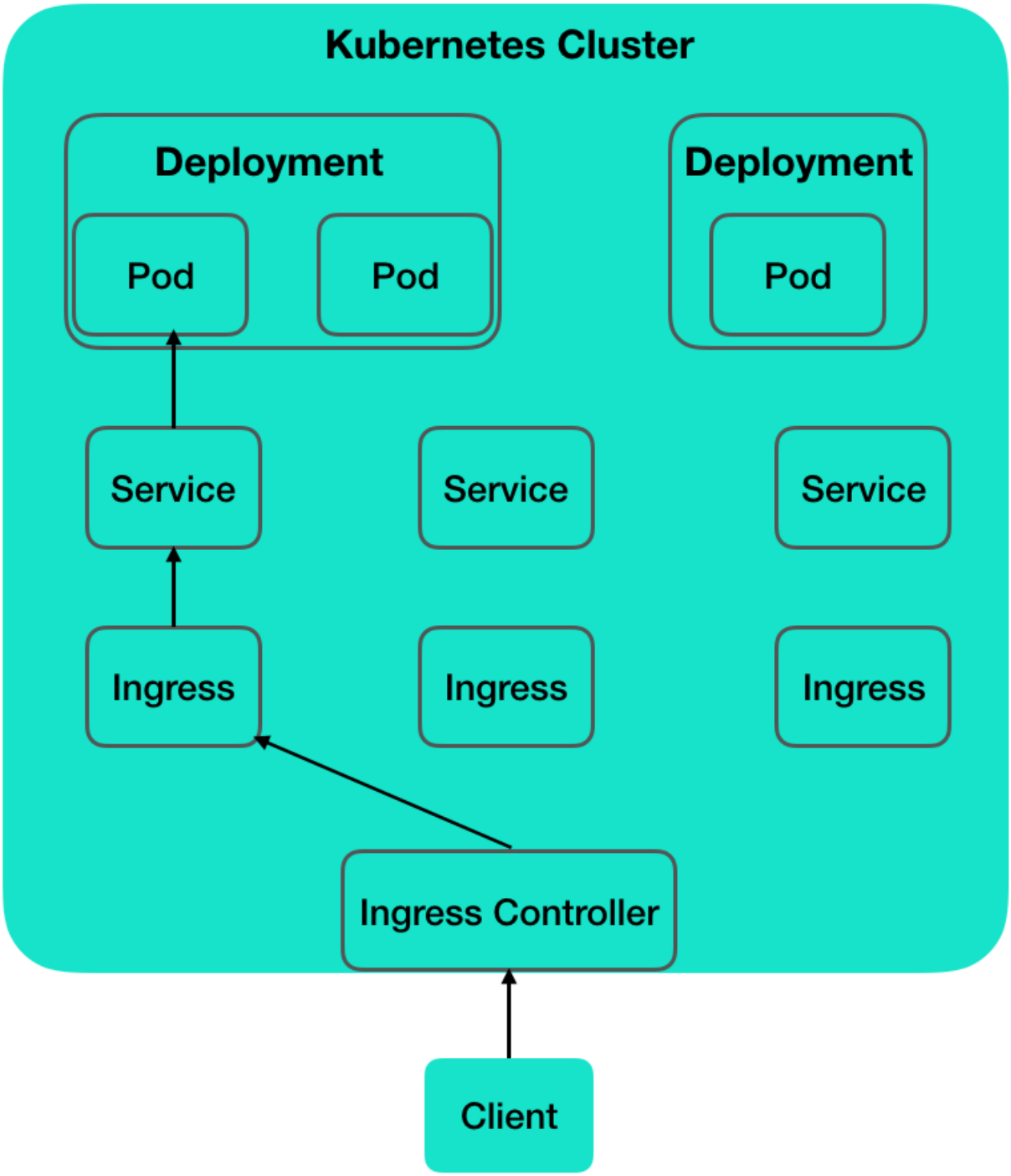


- `Service` 将请求转发给应用，负责配置如何最终将处理传入请求的实际服务
- `Middlewares` 中间件，用来修改请求或者根据请求来做出判断，中间件被附件到路由上，是一种在请求发送到服务之前调整请求的一种方法

Nginx-Ingress和Traefik区别

1. Ingress Controller

k8s 是通过一个又一个的 controller 来负责监控、维护集群状态。Ingress Controller 就是监控 Ingress 路由规则的一个变化，然后跟 k8s 的资源操作入口 api-server 进行通信交互。K8s 并没有自带 Ingress Controller，它只是一种标准，具体实现有多种，需要自己单独安装，常用的是 Nginx Ingress Controller 和 Traefik Ingress Controller。
Ingress Controller 收到请求，匹配 Ingress 转发规则，匹配到了就转发到后端 Service，而 Service 可能代表的后端 Pod 有多个，选出一个转发到那个 Pod，最终由那个 Pod 处理请求。



2. nginx-ingress

由于微服务架构以及 Docker 技术和 kubernetes 编排工具最近几年才开始逐渐流行，所以一开始的反向代理服务器比如 nginx、apache 并未提供其支持，所以才会出现 Ingress Controller 这种东西来做 kubernetes 和前端负载均衡器如 nginx 之间做衔接；即 Ingress Controller 的存在就是为了能跟 kubernetes 交互，然后写入nginx 配置，最后reload。
使用nginx作为前端负载均衡，通过ingress controller不断的和kubernetes api交互，实时获取后端service，pod等的变化，然后动态更新nginx配置，并刷新使配置生效，达到服务发现的目的。

3. traefik

traefik本身设计的就能够实时跟kubernetes api交互，感知后端service，pod等的变化，自动更新配置并重载。

4. traefik优点

- 速度快
- 不需要安装其他依赖，使用 GO 语言编译可执行文件
- 支持最小化官方 Docker 镜像
- 支持多种后台，如 Docker, Swarm mode, Kubernetes, Marathon, Consul, Etcd, Rancher, Amazon ECS 等等
- 支持 REST API
- 配置文件热重载，不需要重启进程
- 支持自动熔断功能
- 支持轮训、负载均衡
- 提供简洁的 UI 界面
- 支持 Websocket, HTTP/2, GRPC
- 自动更新 HTTPS 证书
- 支持高可用集群模式

5. Nginx和Traefik横向对比

	Nginx Ingress	**Traefik ingress**
协议	http/https、http2、grpc、tcp/udp	http/https、http2、grpc、tcp、tcp+tls
路由匹配	host、path	host、path、headers、query、path prefix、method
命名空间支持	-	共用或指定命名空间
部署策略	-	金丝雀部署、蓝绿部署、灰度部署
upstream探测	重试、超时、心跳探测	重试、超时、心跳探测、熔断
负载均衡算法	RR、会话保持、最小连接、最短时间、一致性hash	WRR、动态RR、会话保持

	Nginx Ingress	Traefik ingress
优点	简单易用，易接入	Golang编写，部署容易，支持众多的后端，内置WebUI
缺点	没有解决nginx reload，插件多，但是扩展性能查差	这么一看好像没啥缺点

helm部署traefik

1. 参考文档

- 官方文档：https://doc.traefik.io/traefik/getting-started/install-traefik/
- gtihub地址：https://github.com/traefik/traefik-helm-chart

2. 必要条件

- Kubernetes版本1.14+
- Helm版本3+

3. 安装traefik

```
# 添加repo
[root@k8s-1 ~]# helm repo add traefik https://helm.traefik.io/traefik
# 更新repo仓库资源
[root@k8s-1 ~]# helm repo update
# 查看repo仓库traefik
[root@k8s-1 ~]# helm search repo traefik
NAME            CHART VERSION   APP VERSION     DESCRIPTION
stable/traefik  1.87.7          1.7.26          DEPRECATED - A Traefik based
Kubernetes ingress...
traefik/traefik 9.12.3          2.3.6           A Traefik based Kubernetes ingress
controller
# 创建traefik-v2名称空间
[root@k8s-1 ~]# kubectl create ns traefik
# 安装traefik
[root@k8s-1 ~]# helm install --namespace=traefik traefik traefik/traefik
# 查看helm列表
[root@k8s-1 ~]# helm list -n traefik
NAME    NAMESPACE       REVISION        UPDATED
STATUS          CHART           APP VERSION
traefik traefik-v2      1               2020-12-29 12:16:57.773419046 +0800 CST
deployed        traefik-9.12.3  2.3.6
# 查看pod资源信息
[root@k8s-1 ~]# kubectl get pod -n traefik
NAME                       READY   STATUS    RESTARTS   AGE
traefik-6657f6775d-v6s7b   1/1     Running   0          36s
```

4. 暴露traefik dashboard服务

默认情况下，由于安全考虑，不会公开 Traefik 仪表板。可以通过端口转发实现仪表板访问：

`kubectl port-forward $(kubectl get pods --selector "app.kubernetes.io/name=traefik" --output=name -n traefik) -n traefik --address 0.0.0.0 9000:9000`

```
# 如果出现以下报错信息无法访问
Forwarding from 0.0.0.0:9000 -> 9000
Handling connection for 9000
E1229 13:10:40.655810  243434 portforward.go:400] an error occurred forwarding
9000 -> 9000: error forwarding port 9000 to pod
de96185927c5a2a6150c67130c14ef0f97f14900144389122570ccfc4b5b3179, uid : unable to
do port forwarding: socat not found
Handling connection for 9000
E1229 13:11:06.847006  243434 portforward.go:400] an error occurred forwarding
9000 -> 9000: error forwarding port 9000 to pod
de96185927c5a2a6150c67130c14ef0f97f14900144389122570ccfc4b5b3179, uid : unable to
do port forwarding: socat not found
Handling connection for 9000

# 解决办法在node节点安装socat
yum install socat
```

或者通过定义和应用入口路线 CRD：`kubectl apply -f dashboard.yaml`

```
#  dashboard.yamlapiVersion: traefik.containo.us/v1alpha1

kind: IngressRoute
metadata:
  name: dashboard
  namespace: traefik
spec:
  entryPoints:
    - web
  routes:
    - match: Host(`traefik.localhost`) && (PathPrefix(`/dashboard`) ||
PathPrefix(`/api`))
      kind: Rule
      services:
        - name: api@internal
          kind: TraefikService
```

修改host后访问http://traefik.localhost/dashboard/

手动部署traefik IngressRoute

> yaml手动部署traefik分为ingressRoute和ingress两种，ingressRoute是2.1以后新增功能，简单来说，他们都支持路径(path)路由和域名(host)HTTP路由，以及HTTPS配置，区别在于IngressRoute需要定义CRD扩展，他还支持TCP、UDP路由以及中间件等新特性

1. 环境准备

- k8s版本：1.18.14
- traefik版本：2.3.6
- 其中master节点充当边缘节点，安装两块网卡，eth0：k8s集群内网ip eth1公网ip

2. 创建CRD资源

> 在 traefik v2.1 版本后，开始使用 CRD（Custom Resource Definition）来完成路由配置等，所以需要提前创建 CRD 资源。

创建traefik-crd.yaml 文件

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: ingressroutes.traefik.containo.us
spec:
  group: traefik.containo.us
  version: v1alpha1
  names:
    kind: IngressRoute
    plural: ingressroutes
    singular: ingressroute
  scope: Namespaced

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: middlewares.traefik.containo.us
spec:
  group: traefik.containo.us
  version: v1alpha1
  names:
    kind: Middleware
    plural: middlewares
    singular: middleware
  scope: Namespaced

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: ingressroutetcps.traefik.containo.us
spec:
  group: traefik.containo.us
  version: v1alpha1
  names:
    kind: IngressRouteTCP
    plural: ingressroutetcps
    singular: ingressroutetcp
  scope: Namespaced

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: ingressrouteudps.traefik.containo.us
spec:
  group: traefik.containo.us
  version: v1alpha1
  names:
    kind: IngressRouteUDP
    plural: ingressrouteudps
    singular: ingressrouteudp
  scope: Namespaced

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
```

```
  name: tlsoptions.traefik.containo.us
spec:
  group: traefik.containo.us
  version: v1alpha1
  names:
    kind: TLSOption
    plural: tlsoptions
    singular: tlsoption
  scope: Namespaced

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: tlsstores.traefik.containo.us
spec:
  group: traefik.containo.us
  version: v1alpha1
  names:
    kind: TLSStore
    plural: tlsstores
    singular: tlsstore
  scope: Namespaced

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: traefikservices.traefik.containo.us
spec:
  group: traefik.containo.us
  version: v1alpha1
  names:
    kind: TraefikService
    plural: traefikservices
    singular: traefikservice
  scope: Namespaced
```

创建Traefik CRD 资源

```
# kubectl apply -f traefik-crd.yaml
```

3. 创建RBAC权限

> Kubernetes 在 1.6 版本中引入了基于角色的访问控制（RBAC）策略，方便对 Kubernetes 资源和 API 进行细粒度控制。Traefik 需要一定的权限，所以这里提前创建好 Traefik ServiceAccount 并分配一定的权限。

创建traefik名称空间

```
kubectl create ns traefik
```

创建traefik-rbac.yaml 文件

```
# ServiceAccount
apiVersion: v1
kind: ServiceAccount
metadata:
  name: traefik-ingress-controller
  namespace: traefik
---
# ClusterRole
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller
  namespace: traefik
rules:
  - apiGroups:
      - ""
    resources:
      - services
      - endpoints
      - secrets
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - extensions
      - networking.k8s.io
    resources:
      - ingresses
      - ingressclasses
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - extensions
    resources:
      - ingresses/status
    verbs:
      - update
  - apiGroups:
      - traefik.containo.us
    resources:
      - middlewares
      - ingressroutes
      - traefikservices
      - ingressroutetcps
      - ingressrouteudps
      - tlsoptions
      - tlsstores
    verbs:
      - get
      - list
      - watch

---
# ClusterRoleBinding
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
```

```
metadata:
  name: traefik-ingress-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: traefik-ingress-controller
subjects:
  - kind: ServiceAccount
    name: traefik-ingress-controller
    namespace: traefik
```

创建 Traefik RBAC 资源

```
kubectl apply -f traefik-rbac.yaml
```

4. 创建traefik配置文件

> 在 Traefik 中有三种方式定义静态配置：在配置文件中、在命令行参数中、通过环境变量传递，由于 Traefik 配置很多，通过 CLI 定义不是很方便，一般时候选择将其配置选项放到配置文件中，然后存入 ConfigMap，将其挂入 traefik 中。

创建 traefik-config.yaml 文件

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: traefik-config
  namespace: traefik
data:
  traefik.yaml: |-
    serversTransport:
      insecureSkipVerify: true  ## Traefik 忽略验证代理服务的 TLS 证书
    api:
      insecure: true            ## 允许 HTTP 方式访问 API
      dashboard: true           ## 启用 Dashboard
      debug: false              ## 启用 Debug 调试模式
    metrics:
      prometheus: metrics       ## 配置 Prometheus 监控指标数据，并使用默认配置
    entryPoints:
      web:
        address: ":80"          ## 配置 80 端口，并设置入口名称为 web
      websecure:
        address: ":443"         ## 配置 443 端口，并设置入口名称为 websecure
      traefik:
        address: ":8080"        ## 配置 8080 端口，并设置入口名称为 dashboard
      metrics:
        address: ":8082"        ## 配置 8082 端口，作为metrics收集入口
      tcpep:
        address: ":8000"        ## 配置 8000 端口，作为tcp入口
      udpep:
        address: ":9000/udp"    ## 配置 9000 端口，作为udp入口
    providers:
      kubernetescrd:            ## 启用 Kubernetes CRD 方式来配置路由规则
        ingressclass: ""
      kubernetesingress:        ## 启动 Kubernetes Ingress 方式来配置路由规则
        ingressclass: ""
    log:
      filePath: "/etc/traefik/logs/traefik.log"               ## 设置调试日志文件存储路
径，如果为空则输出到控制台
      level: error              ## 设置调试日志级别
      format: ""                ## 设置调试日志格式
    accessLog:
      filePath: "/etc/traefik/logs/access.log"              ## 设置访问日志文件存储路
径，如果为空则输出到控制台
      format: ""                ## 设置访问调试日志格式
      bufferingSize: 0          ## 设置访问日志缓存行数
      filters:
        #statusCodes: ["200"]   ## 设置只保留指定状态码范围内的访问日志
        retryAttempts: true     ## 设置代理访问重试失败时，保留访问日志
        minDuration: 20         ## 设置保留请求时间超过指定持续时间的访问日志
      fields:                   ## 设置访问日志中的字段是否保留（keep 保留、drop 不保留）
        defaultMode: keep       ## 设置默认保留访问日志字段
        names:                  ## 针对访问日志特别字段特别配置保留模式
          ClientUsername: drop
        headers:                ## 设置 Header 中字段是否保留
          defaultMode: keep     ## 设置默认保留 Header 中字段
          names:                ## 针对 Header 中特别字段特别配置保留模式
            User-Agent: redact
            Authorization: drop
            Content-Type: keep
```

创建 Traefik ConfigMap 资源

```
kubectl apply -f traefik-config.yaml
```

5. 节点设置label标签

> 由于集群中master节点安装两块网卡充当边缘节点，需要提前给master节点设置Label，这样当程序部署时 Pod 会自动调度到设置 Label 的节点上。

给work1节点设置标签

```
kubectl label nodes k8s-master IngressProxy=true
```

查看节点label信息

```
[root@k8s-master traefik]# kubectl get nodes --show-labels
NAME         STATUS   ROLES    AGE    VERSION    LABELS
k8s-master   Ready    master   227d   v1.18.14
IngressProxy=true,beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kuberne
master,kubernetes.io/os=linux,node-role.kubernetes.io/master=
k8s-work1    Ready    <none>   227d   v1.18.14
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,
work1,kubernetes.io/os=linux
k8s-work2    Ready    <none>   227d   v1.18.14
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,
work2,kubernetes.io/os=linux
k8s-work3    Ready    <none>   227d   v1.18.14
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,
work3,kubernetes.io/os=linux
```

6. Deployment部署traefik

> 使用DeamonSet或者Deployment均可部署，此处使用Deployment方式部署 Traefik，副本数设置为1，调度至IngressProxy=true的那台master边缘节点

创建 traefik 部署文件 traefik-deploy.yaml

```
# Deployment
kind: Deployment
apiVersion: apps/v1
metadata:
  name: traefik
  namespace: traefik
  labels:
    app: traefik
spec:
  replicas: 1   #副本数为1，因为集群只设置一台work1为边缘节点
  selector:
    matchLabels:
      app: traefik
  template:
    metadata:
      labels:
        app: traefik
    spec:
      serviceAccountName: traefik-ingress-controller
      terminationGracePeriodSeconds: 1
      containers:
        - name: traefik
          image: traefik:v2.3
          args:
            - --configfile=/config/traefik.yaml
          ports:
            - name: web
              containerPort: 80
            - name: admin
              containerPort: 8080
            - name: tcpep
              containerPort: 8000
            - name: udpep
              containerPort: 9000
          securityContext:
            capabilities:              ## 只开放网络权限
              drop:
                - ALL
              add:
                - NET_BIND_SERVICE
          volumeMounts:
          - mountPath: "/config"
            name: "config"
          - mountPath: /etc/traefik/logs
            name: logdir
          - mountPath: /etc/localtime
            name: timezone
            readOnly: true
      volumes:
        - name: config
          configMap:
            name: traefik-config
        - name: logdir
          hostPath:
            path: /data/traefik/logs
            type: "DirectoryOrCreate"
        - name: timezone
          hostPath:
            path: /etc/localtime
```

```
            type: File
      tolerations:
        - operator: "Exists"          ## 设置容忍所有污点，防止节点被设置污点
      hostNetwork: true               ## 开启host网络，提高网络入口的网络性能
      nodeSelector:                   ## 设置node筛选器，在特定label的节点上启动
        IngressProxy: "true"          ## 调度至IngressProxy: "true"的节点
---
# Service
apiVersion: v1
kind: Service
metadata:
  name: traefik
  namespace: traefik
spec:
  type: NodePort                      ## 官网示例为LB，由于没有条件，所有改为NodePort
  selector:
    app: traefik
  ports:
    - protocol: TCP
      port: 80
      name: web
      targetPort: 80
    - protocol: TCP
      port: 8080
      name: admin
      targetPort: 8080
    - protocol: TCP
      port: 8000
      name: tcpep
      targetPort: 8000
    - protocol: UDP
      port: 9000
      name: udpep
      targetPort: 9000
```

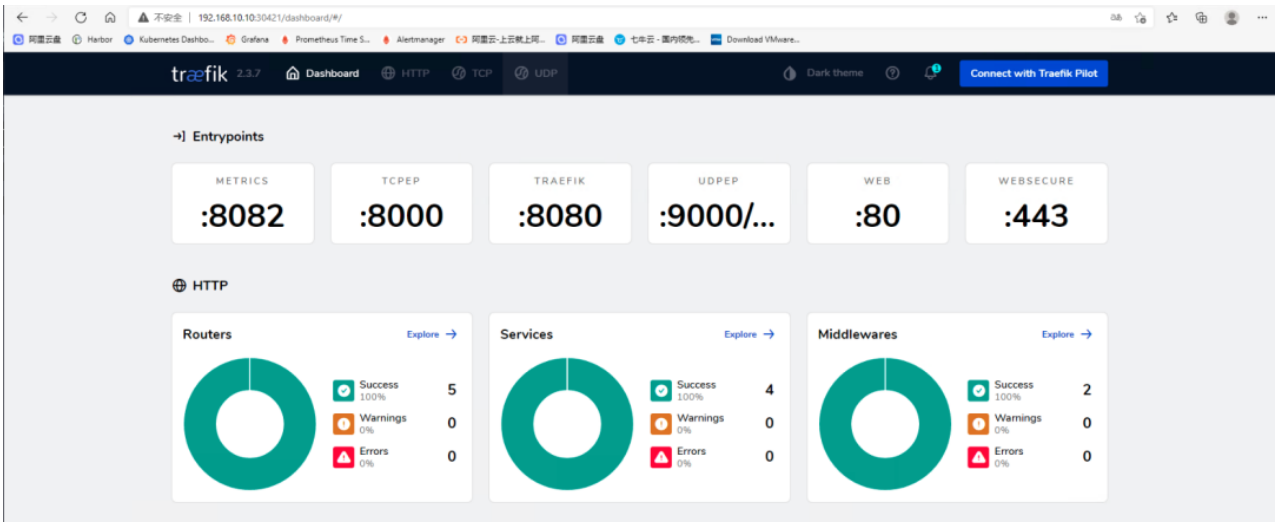
Kubernetes 部署 Traefik

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

查看资源信息

```
[root@k8s-master traefik]# kubectl get pod -n traefik
NAME                          READY   STATUS    RESTARTS   AGE
traefik-v2-7f797bccb6-cm46v   1/1     Running   0          3h7m
[root@k8s-master traefik]# kubectl get svc -n traefik
NAME            TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
traefik-v2      NodePort   10.101.231.184   <none>
80:32077/TCP,8080:30421/TCP,8000:30899/TCP   3h7m
traefikudp-v2   NodePort   10.111.159.51    <none>        9000:30579/UDP
3h7m
```

traefik的dashboard使用nodeport暴露服务，将8080端口映射为30421，现在访问 http://192.168.10.11:30421/



7. dashboard配置http域名访问

> Traefik 应用已经部署完成，treafik的Dashboard为svc类型是ClusterIP，接下来配置域名规则，通过traefik.local.com访问dashboard

创建 Traefik Dashboard 路由规则文件 traefik-dashboard-route.yaml

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: traefik-dashboard-route
  namespace: traefik
spec:
  routes:
    - match: Host(`traefik.local.com`)
      kind: Rule
      services:
        - name: api@internal
          kind: TraefikService
```

创建 Traefik Dashboard 路由规则对象

```
kubectl apply -f traefik-dashboard-route.yaml
```

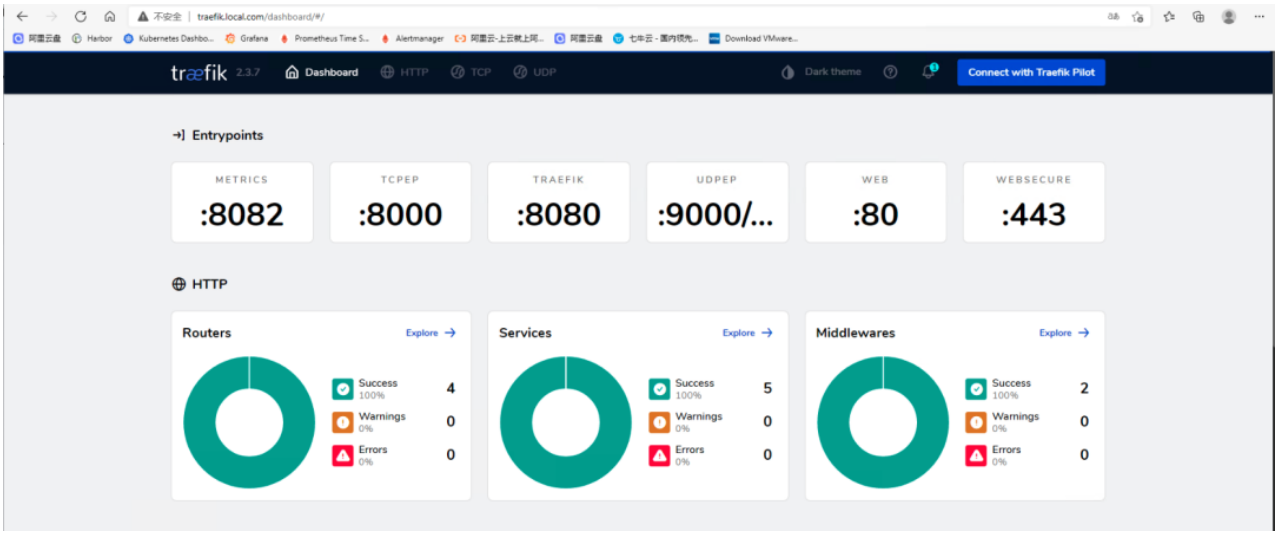
查找资源信息

```
[root@k8s-master traefik]# kubectl get ingressroute -n traefik
NAME                     AGE
traefik-dashboard-route   44m
[root@k8s-master traefik]# kubectl describe ingressroute -n traefik
.....
Spec:
  Routes:
    Kind:   Rule
    Match:  Host(`traefik.local.com`)
    Services:
      Kind:  TraefikService
      Name:  api@internal
Events:      <none>
```

配置 Hosts，客户端想通过域名访问服务，必须要进行 DNS 解析，由于这里没有 DNS 服务器进行域名解析，所以修改 hosts 文件将 Traefik 指定的节点的 IP 和自定义 host 绑定。打开电脑的 Hosts 配置文件，往其加入下面配置：

```
192.168.10.10 traefik.local.com
```

配置完成后，打开浏览器输入地址：http://traefik.local.com 打开 Traefik Dashboard。



8. dashboard配置https域名访问

> 在实际生产环境中，大多数网站都需要使用https加密访问，可以使用CA机构颁发的证书，此处测试环境，直接使用openssl生成的证书即可，接下来以Traefik的dashboard为例，配置https路由

创建证书文件

```
#创建一个traefik证书目录
mkdir /root/k8s-install/traefik/tls && cd /root/k8s-install/traefik/tls


#创建证书
openssl req -newkey rsa:2048 -nodes -keyout tls.key -x509 -days 365 -out tls.crt
#这里信息都无所谓，直接一路回车就可以

 #当期目录下会产生一个crt文件和一个key文件
[root@k8s-master tls]# ls
tls.crt  tls.key
```

使用secret对象，将证书储存到secret中

```
kubectl create secret tls traefik-tls --cert=tls.crt --key=tls.key -n traefik
```

查看secret资源

```
[root@k8s-master tls]# kubectl get secrets -n traefik
NAME                                  TYPE
DATA   AGE
default-token-nf45n                   kubernetes.io/service-account-token   3
41m
traefik-ingress-controller-token-fzpq4   kubernetes.io/service-account-token   3
40m
traefik-tls                           kubernetes.io/tls                     2
43s
[root@k8s-master ssl]# kubectl describe secrets -n traefik
……
Data
====
tls.key:  1708 bytes
tls.crt:  1237 bytes
```

给traefik创建一个配置文件

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: traefik-dashboard-route-tls
  namespace: traefik
spec:
  routes:
    - match: Host(`traefik.local.com`)
      kind: Rule
      services:
        - name: api@internal
          kind: TraefikService
  tls:
    secretName: traefik-tls
```

创建 Kubernetes Dashboard 路由规则对象

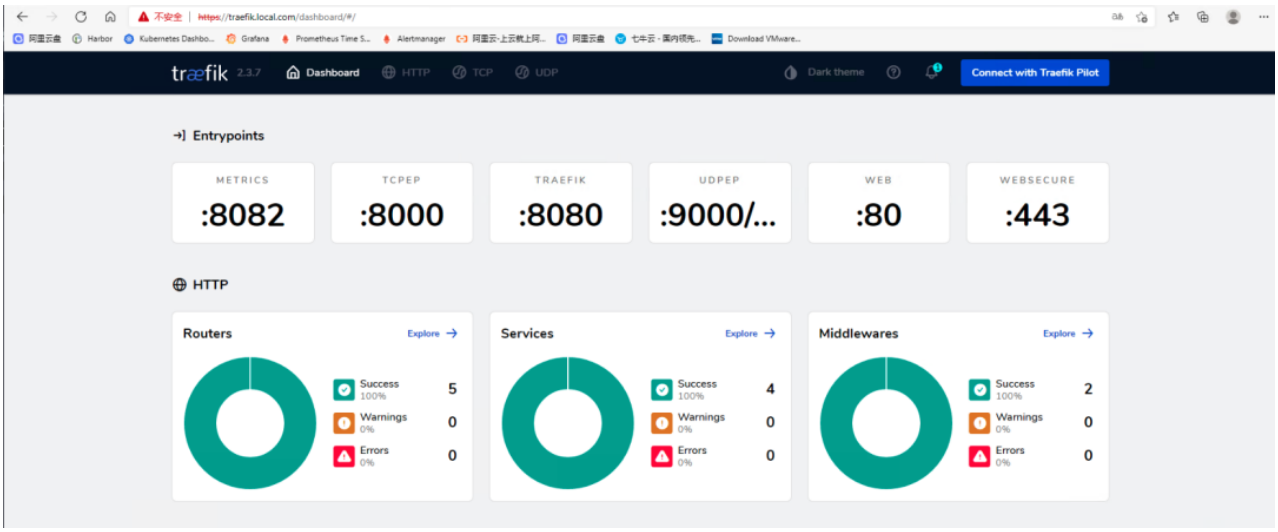
```
kubectl apply -f traefik-dashboard-route-tls.yaml
```

配置 Hosts 文件

```
192.168.10.10 traefik.local.com
```

配置完成后，打开浏览器输入地址：https://traefik.local.com 打开 Dashboard Dashboard。

至此，traefik部署已完成。


不安全 | https://traefik.local.com/dashboard/#/
traefik 2.3.7
Dashboard
HTTP
TCP
UDP
Dark theme
Connect with Traefik Pilot
Entrypoints
METRICS
:8082
TCPEP
:8000
TRAEFIK
:8080
UDPEP
:9000/...
WEB
:80
WEBSECURE
:443
HTTP
Routers
Explore
Success 100% 5
Warnings 0% 0
Errors 0% 0
Services
Explore
Success 100% 4
Warnings 0% 0
Errors 0% 0
Middlewares
Explore
Success 100% 2
Warnings 0% 0
Errors 0% 0