1

Outline

- Review of Kubernetes Services
- What is an Ingress ?
- SetUp Traefic Ingress Controller
- Accessing Internal Services

2

Review of Kubernetes Services

3

Service Types

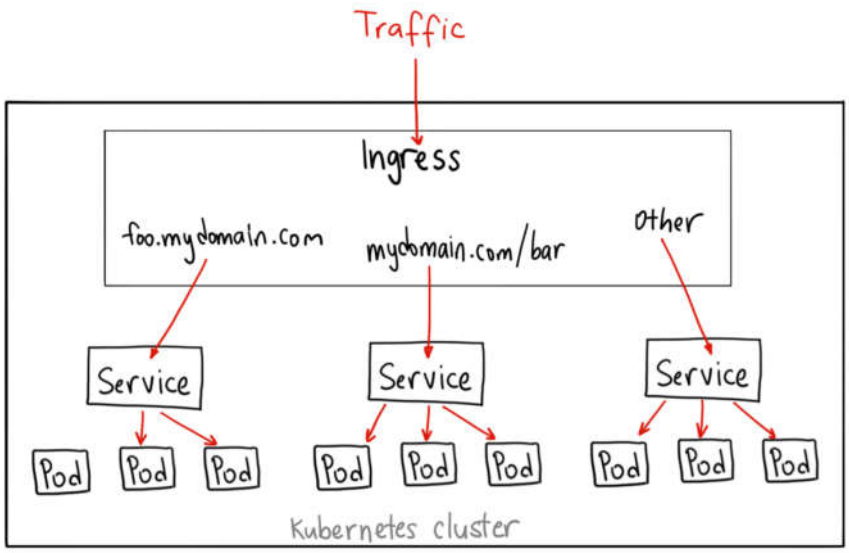
1. **ClusterIP (default)**
   - A virtual IP address is allocated for the service
   - This IP address is reachable only from within the cluster (nodes and pods)
   - Perfect for internal communication, within the cluster
2. **NodePort**
   - NodePort services extend the ClusterIP service.
   - Exposes a port on every node's IP.
   - Port can either be statically defined, or dynamically taken from a range between 30000-32767.
3. **LoadBalancer**
   - LoadBalancer services extend NodePort
   - Works in conjunction with an external system to map a cluster external IP to the exposed service (typically a cloud load balancer, e.g. ELB on AWS, GLB on GCE ...)

4

What is an Ingress ?

5

Ingess



6

The Ingres API Object

- **An Ingres is an API object that manages external access to the services in a cluster**
  - Provides load balancing, SSL termination and name/path-based virtual hosting
  - Gives services externally-reachable URLs
- **They are specifically for HTTP services(not TCP or UDP)**
- **They can also handle TLS certificates, URL rewriting ...**

```
# Path based  routing Examle
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: service1
          servicePort: 4200
      - path: /bar
        backend:
          serviceName: service2
          servicePort: 8080
```


7

Ingress Controller

- **The Ingress manifest doesn't actually do anything on its own; you must deploy an Ingress Controller into your cluster to watch for these declarations and act upon them.**
- **Ingress controllers are pods, just like any other application, so they're part of the cluster and can see other pods. They're built using reverse proxies that have been active in the market for years.**
  - So, you have your choice of an **HAProxy, traefic**, **NGINX** Ingress Controller, and so on. The underlying proxy gives it Layer 7 routing and load balancing capabilities.
- **Being inside the cluster themselves, Ingress Controllers are susceptible to the same walled-in jail as other Kubernetes pods.**
  - You need to expose them to the outside via a Service with a type of either NodePort or LoadBalancer.
  - However, now you have a single entrypoint that all traffic goes through: one Service connected to one Ingress Controller, which, in turn, is connected to many internal pods.
  - The controller, having the ability to inspect HTTP requests, directs a client to the correct pod based on characteristics it finds, such as the URL path or the domain name.

8

Set up Traefic Ingress controller

9

Traefic Ingress Controller

- **Traefik is a modern HTTP reverse proxy and load balancer that makes deploying microservices easy.**
  - Traefik integrates with your existing infrastructure components (Docker, Swarm mode, Kubernetes, Amazon ECS, ...) and configures itself automatically and dynamically. Pointing Traefik at your orchestrator should be the only configuration step you need.

- **Features:**
  - Continuous update of configuration (no restarts),
  - Support for multiple load balancing algorithms,
  - Web UI, metrics export,
  - Support for various protocols, REST API, canary releases and so on.
  - The support for Let's Encrypt certificates right out of the box is another nice feature.

10

Setup Traefic

https://doc.traefik.io/traefik/getting-started/install-traefik/

- **Add Traefik's chart repository to Helm:**

```
helm repo add traefik https://helm.traefik.io/traefik
```

- **You can update the chart repository by running:**

```
helm repo update
```

- **And install it with the helm command line:**

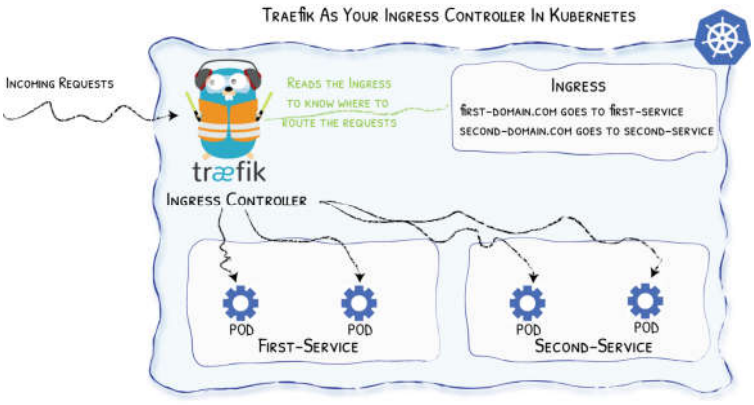
```
helm install traefik traefik/traefik
```

- **Exposing the Traefik dashboard**
  - This HelmChart does not expose the Traefik dashboard by default, for security concerns. Thus, there are multiple ways to expose the dashboard. For instance, the dashboard access could be achieved through a port-forward :

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


11

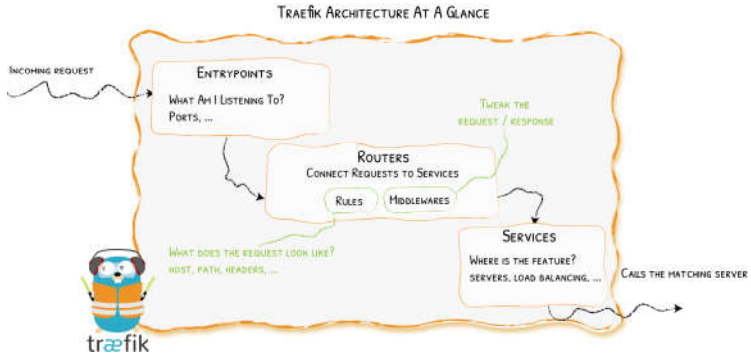
How Traefik works ?



12

Exposing Services with Ingress Resources

Traefik 2 Architecture



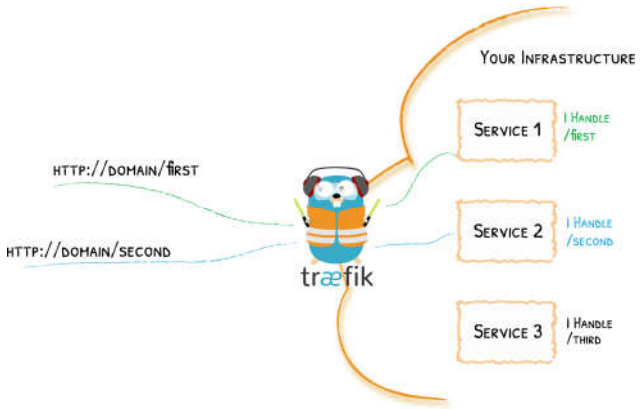
- **Providers** discover the services that live on your infrastructure (their IP, health, ...)
- **Entrypoints** listen for incoming traffic (ports, ...)
- **Routers** analyse the requests (host, path, headers, SSL, ...)
- **Services** forward the request to your services (load balancing, ...)
- **Middlewares** may update the request or make decisions based on the request (authentication, rate limiting, headers, ...)

M.Romdhani, 2021 **13**

13

Exposing Services with Ingress Resources

Path-based Routing

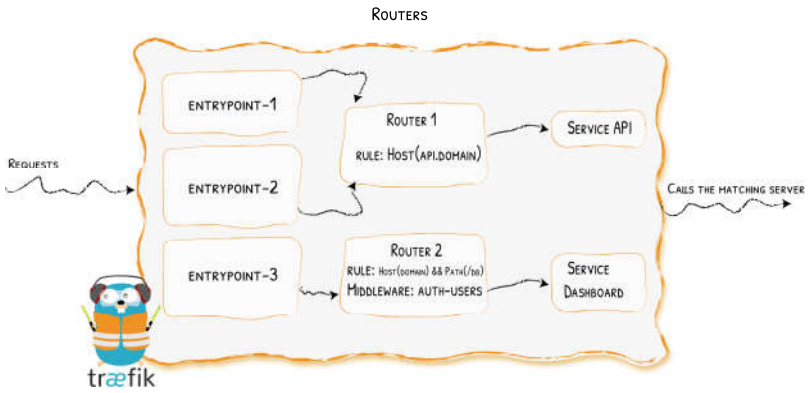


M.Romdhani, 2021 **14**

14

Exposing Services with Ingress Resources

Host-based Routing



M.Romdhani, 2021 15

15

Exposing Services with Ingress Resources

The New Ingress v1 syntax

- V1beta1 syntax

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: minimal-ingress-v1beta1
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          serviceName: test
          servicePort: 80
```

- V1 Syntax

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress-v1
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

M.Romdhani, 2021 16

16

Traefik IngressRoute CRD object

Standard Ingress Syntax

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress-path-based
  annotations:
    // Stripped
spec:
  rules:
  - http:
      paths:
        - path: /apple
          pathType: Prefix
          backend:
            service:
              name: apple-service
              port:
                number: 5678
        - path: /banana
          pathType: Prefix
          backend:
            service:
              name: banana-service
              port:
                number: 5678
```

Traefik IngressRoute Syntax

```yaml
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: example-ingress-path-based-crd
spec:
  entryPoints:
    - web
  routes:
    - match: PathPrefix(`/apple`)
      kind: Rule
      services:
        - name: apple-service
          port: 5678
          strategy: RoundRobin
          weight: 10
      middlewares:
        - name: do-stripprefix

    - match: PathPrefix(`/banana`)
      kind: Rule
      services:
        - name: banana-service
          port: 5678
          strategy: RoundRobin
          weight: 10
```


17

Ingress Definition for the BookStoreApp

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bookstore-ingress
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: bookstore.minikube
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
             name: frontend
             port:
               number: 80
      - path: /books
        pathType: Prefix
        backend:
          service:
             name: backend-api
             port:
               number: 80
      - path: /categories
        pathType: Prefix
        backend:
          service:
             name: backend-api
             port:
               number: 80
```


18

Accessing Internal Services

19

Accessing internal services

- **When we are logged in on a cluster node, we can access internal services**
  - As per the Kubernetes network model: all nodes can reach all pods and services)
- **When we are accessing a remote cluster, our local machine won't have access to the cluster's internal subnet. To overcome this:**
  - **kubectl proxy**: gives us access to the API, which includes a proxy for HTTP resources
  - **kubectl port-forward**: allows forwarding of TCP ports to arbitrary pods, services, ...

20

kubectl proxy

- **Running `kubectl proxy` gives us access to the entire Kubernetes API**
  - The API includes routes to proxy HTTP traffic
  - By default, the proxy listens on port 8001
- **These routes look like the following:**
  - `/api/v1/namespaces/<namespace>/services/<service>/proxy`
- **We just add the URI to the end of the request, for instance:**
  - `/api/v1/namespaces/<namespace>/services/<service>/proxy/index.html`
- **We can access services and pods this way !**
- ***Security considerations : kubectl proxy is intended for local use***
  - Running kubectl proxy openly is a huge security risk
  - It is slightly better to run the proxy where you need it (and copy credentials, e.g. ~/.kube/config, to that place)
  - It is even better to use a limited account with reduced permissions

21

kubectl port-forward

- **What if we want to access a TCP service?**
  - We can use **kubectl port-forward** instead
  - It will create a TCP relay to forward connections to a specific port (of a pod, service, deployment...)
- **The syntax is:**

```
kubectl port-forward service/name_of_service local_port:remote_port
```

  - If only one port number is specified, it is used for both local and remote ports

22