Getting Started with Traefik and the New Kubernetes Service APIs

SOME IMAGE?

As we already introduced in June last year, there has been a movement inside the Kubernetes Community to work on a next iteration for defining and managing Ingress Traffic. As a result, there is a new set of Service API which feature the so-called Gateway-AP to tackle that task. This post will feature an "how to use" approach of that set of APIs with Traefik. For more information about the whole standard on its own, you can find more information on the old post.

Prerequisites

- Kubernetes Cluster
- Traefik official docs
- Kubeconfig file to access your Kubernetes Cluster through `kubectl`

Configuration files for this tutorial can be found here: https://github.com/traefik-tech-blog/k8s-service-apis

Installing the CRDs

To install the CRD's, you can just use the current released version 0.10

```
kubectl apply -k "github.com/kubernetes-sigs/service-apis/config/crd?
ref=v0.1.0"
```

Install and configure Traefik to use Service APIs

To install Traefik v2.4 (or later) and have it configured to enable the new provider, best way is to install Traefik through our helm chart

```
helm repo add traefik https://helm.traefik.io/traefik
helm repo update
helm install traefik --set experimental.kubernetesGateway.enabled=true
traefik/traefik
```

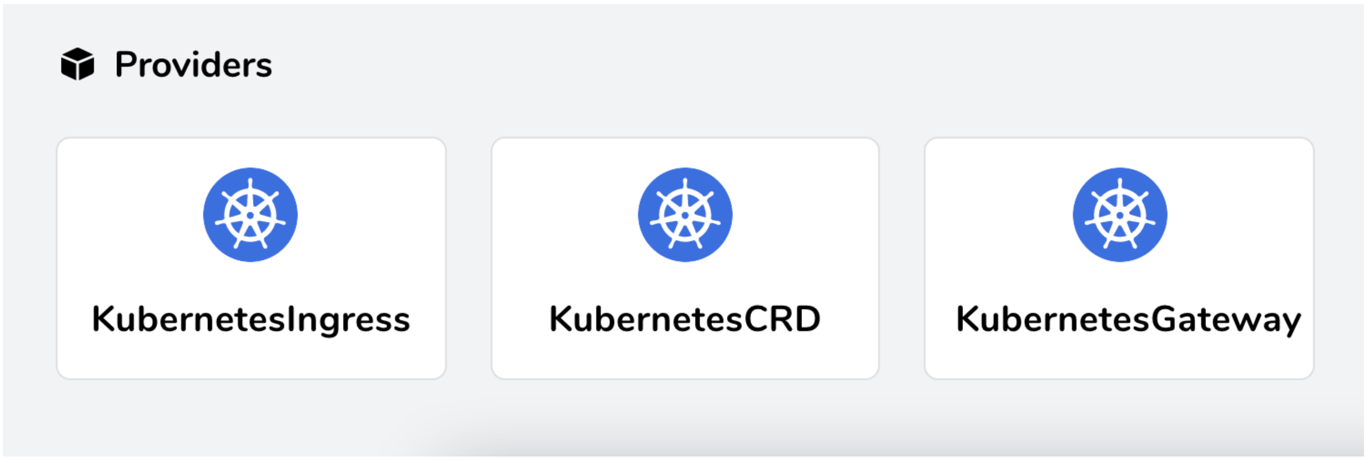
More customization options for the installation, such as the labeSelector or TLS Certificates (which we see later) are visible in the values file. (put link).

That will install Traefik 2.4, enable the new provider and also make sure that the creation of GatewayClasses and a GateWay instance is taken care of.

Then you can port forward to the dashboard to check if the provider is activated and ready to serve.

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

Your dashboard, should show all Kubernetes related providers like that then:



From here, we are ready to go.

Setup a dummy service

In order to have a target to route Traefik to, we will quickly install the famous whoami service in order to have something to use for testing purposes later.

```
# 01-whoami.yaml
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: whoami

spec:
  replicas: 2
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
    spec:
      containers:
        - name: whoami
          image: traefik/whoami:v1.6.0
          ports:
            - containerPort: 80
              name: http

---
```

```
apiVersion: v1
kind: Service
metadata:
  name: whoami

spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: http
  selector:
    app: whoami
```

Simple Host

Everything is set and ready now, to deploy our first simple `HTTPRoute` to see the action going.

```
# 02-whoami-httproute.yaml
---
kind: HTTPRoute
apiVersion: networking.x-k8s.io/v1alpha1
metadata:
  name: http-app-1
  namespace: default
  labels:
    app: traefik
spec:
  hostnames:
    - "whoami"
  rules:
    - matches:
        - path:
            type: Exact
            value: /
      forwardTo:
        - serviceName: whoami
          port: 80
          weight: 1
```

This HTTPRoute will catch requests going on `whoami` and forward them to the service, which is our simple whoami service as mentioned above. All of that is possible through the labelSelector of `app: traefik`. This is set during the installation phase mentioned above and can be customized with the Helm chart.

If you know emit a request for that hostname, you will see something like this:

```
curl -H "Host: whoami" http://localhost
Hostname: whoami-9cdc57b6d-pfpxs
IP: 127.0.0.1
```

```
IP: ::1
IP: 10.42.0.13
IP: fe80::9c1a:a1ff:fead:2663
RemoteAddr: 10.42.0.11:33658
GET / HTTP/1.1
Host: whoami
User-Agent: curl/7.64.1
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 10.42.0.1
X-Forwarded-Host: whoami
X-Forwarded-Port: 80
X-Forwarded-Proto: http
X-Forwarded-Server: traefik-74d7f586dd-xxr7r
X-Real-Ip: 10.42.0.1
```

Simple Host with Paths

The example above can easily be enhanced to only react on a given subpath.

```yaml
# 03-whoami-httproute-paths.yaml
---
apiVersion: networking.x-k8s.io/v1alpha1
kind: HTTPRoute
metadata:
  labels:
    app: traefik
  name: http-app-1
  namespace: default
spec:
  hostnames:
    - whoami
  rules:
    -
      forwardTo:
        -
          port: 80
          serviceName: whoami
          weight: 1
      matches:
        -
          path:
            type: Exact
            value: /foo
```

The result will look like that:

```
curl -H "Host: whoami" http://localhost/foo
Hostname: whoami-9cdc57b6d-pfpxs
IP: 127.0.0.1
IP: ::1
IP: 10.42.0.13
IP: fe80::9c1a:a1ff:fead:2663
RemoteAddr: 10.42.0.11:34424
GET /foo HTTP/1.1
Host: whoami
User-Agent: curl/7.64.1
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 10.42.0.1
X-Forwarded-Host: whoami
X-Forwarded-Port: 80
X-Forwarded-Proto: http
X-Forwarded-Server: traefik-74d7f586dd-xxr7r
X-Real-Ip: 10.42.0.1
```

More information about what part of a request can be matched are visible on the official Service API documentation. TLS with static certificates Until here, we have created a simple HTTP Route. For the next step, we want to secure this route through TLS. For that, we need to create a secret first with a dummy certificate.

```
# 04-tls-dummy-cert.yaml
---
apiVersion: v1
data:
  tls.crt:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVVVENDQXJtZ0F3SUJBZ0lRV2pNZ2Q4OU
xOUXIwVC9WMDdGR1pEREFOQmdrcWhraUc5dzBCQVFzRkFEQ0IKaFRFZU1Cd0dBMVVFQ2hNVmJX
dGpaWEowSUdSbGRtVnNiM0J0Wlc1MElFTkJNUzB3S3dZRFZRUUxEQ1JxWW1SQQpaSEpwZW5wME
lDaEtaV0Z1TFVKaGNIUnBjM1JsSUVSdmRXMWxibXB2ZFNreE5EQXlCZ05WQkFNTUsyMXJZMlZ5
CmRDQnFZbVJBWkhKcGVucDBJQ2hLWldGdUxVSmhjSFJwYzNSbElFUnZkVzFsYm1wdmRTa3dIaG
NOTWpBeE1qQTAKTVRReE1qQXpXaGNOTWpNd016QTBNVFF4TWpBeldqQllNU2N3SlFZRFZRUUtF
eDV0YTJObGNuUWdaR1YyWld4dgpjRzFsYm5RZ1kyVnlkR2xtYVdOaGRHVXhMVEFyQmdOVkJBc0
1KR3BpWkVCa2NtbDZlblFnS0VwbFXNHRRbUZ3CmRHbHpkR1VnUkc5MWJXVnVhbTkxS1RDQ0FT
SXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUIKQU12bEc5d0ZKZklRSWRreD
RXUy9sNGhQTVRQcmVUdmVQOS9MZlBYK2h2ekFtVC90V1BJbGxGY2JJNnZzemp0NQpEWlZUMFFu
QzhHYzg0K1lPZXZHcFpNaTg0M20zdTdFSUlmY3dETUF4WWQ0ZjJJcENNLVW9jSFNtVGpOaVhDSn
hwCjVNd2tlVXdEc1dvVVZza1RxeVpOcWp0RWVIbGNuQTFHaGZSa3dEUkZxd1QxeVhaUTBoZHpk
QzRCeFhhaVk0VEQKaFQ1dnFXQmlnUlh0M1VwSkhEL1NXUG4wTEVQOHM3ckhjUkZPY0RhV3ZWMW
1jTkxNZUpveWNYUTJ0M2Z1Q0Fsegp3UWZOSjFQSk45QWlLalFJcXJ1MGFnMC9wU0kyQ3NkbEUz
UTFpM29tZGpCQkZDcmxNMTZyY0wwNDdtWXZKOEVvCjFMdDVGQkxnVURBZkttIOFRsaXU0ZG9jQ0
F3RUFBYU5wTUdjd0RnWURWUjBQQVFIL0JBUURBZ1dnTUJNR0ExVWQKSlFRTU1Bb0dDQ3NHQVFV
RkJ3TUJNQXdHQTFVZEV3RUIvd1FDTUFBd0h3WURWUjBqQkJnd0ZvQVV5cWNiZGhDego3Nm4xZj
FtR3BaemtNb2JJOYnJ3d0VRWURWUjBSQkFvd0NJSUdkMmh2WVcxcE1BMEdDU3FHU0liM0RRRUJD
d1VBCkE0SUJnUUFzWlBndW1EdkRmNm13bXR1TExkWlZkZjdYWk13TjVNSkk5SlpUQ1NaRFRQRj
RsdG91S2RCV0gxYm0Kd003VUE0OXVWSHplNVNDMDNlQ294Zk9Ddlczby94SFZjcDZGei9qSldl
YlY4SWhJRi9JbGNRRyszTVRRMVJaVApwNkZOa3kvOEk3anF1R2V2b0xsbW9KamVRV2dxWGtFL0
```

```
d1MFloVCtudVBJY1pGa0hsKzFWOThEUG5WaTJ3U0hHCkIwVU9RaFdxVkhRU0RzcjJLVzlPbmhT
RzdKdERBcFcwVEltYmNCaWlXOTlWNG9Ga3VNYmZQOE9FTUY2ZXUzbW0KbUVuYk1pWFFaRHJUMW
llMDhwWndHZVNhcTh1Rk82djRwOVVoWHVuc3Vpc01YTHJqQzFwNmlwaDdpMTYwZzRWawpmUXlY
T09KY0o2WTl2a2drYzRLYUxBZVNzVUQvRDR1bmd6emVWQ3k0ZXhhMmlBakpzVHVRS3JkOFNUTG
NNbUJkCnhtcXVKZXFWSEpoZEVMNDBMVGtEY1FPM1NzOUJpbjRaOEFXeTJkdkR1a1gwa084dm9I
UnN4bWVKcnVyZ09MVmIKamVvbTVQMTVsMkkwY3FKd2lNNHZ3SlBsb25wMTdjamJUb0IzQTU5Rj
ZqekdONWtCbjZTaWVmR3VLM21hVWdKegoxWndjamFjPQotLS0tLUVORCBDRVJUSUZJQ0FURS0t
LS0t
  tls.key:
LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1JSUV2Z0lCQURBTkJna3Foa2lHOXcwQkFRRU
ZBQVNDQktnd2dnU2tBZ0VBQW9JQkFRREw1UnZjQlNYeUVDSFoKTWVGa3Y1ZUlUekV6NjNrNzNq
L2Z5M3oxL29iOHdKay83Vmp5SlpSWEd5T3I3TTQ3ZVEyVlU5RUp3dkJuUE9QbQpEbnJ4cVdUSX
ZPTjV0N3V4Q0NIM01BekFNV0hlSDlpS1FpbEtIQjBwazR6WWx3aWNhZVRNSkhsTUE3RnFGRmJK
CkU2c21UYW83UkhoNVhKd05Sb1gwWk1BMFJhc0U5Y2wyVU5JWGMzUXVBY1Yyb21PRXc0VStiNm
xnWW9FVjdkMUsKU1J3LzBsajU5Q3hEL0xPNngzRVJUbkEybHIxZFpuRFN6SGlhTW5GME5yZDM3
Z2dKYzhFSHpTZFR5VGZRSWlvMApDS3E3dEdvTlA2VWlOZ3JIWlJOME5ZdDZKbll3UVJRcTVUTm
VxM0M5T081bUx5ZkJLTlM3ZVJRUzRGQXdIeWgvCkU1WXJ1SGFIQWdNQkFBRUNnZ0VCQUl5SWpv
bzQxaTJncHVQZitIMkxmTE5MK2hyU0cwNkRZajByTVNjUVZ4UVEKMzgvckZOcFp3b1BEUmZQek
ZUWnl1a1VKYjFRdUU2cmtraVA0S1E4MTlTeFMzT3NCRTVIeWpBNm5CTExYbHFBVwpEUmRHZ05U
K3lhN2xiemU5NmdaOUNtRVdackJZLzBpaFdpdmZyYUNKK1dJK1VGYzkyS1ZoeldSa3FRR2VYME
RiCnVSRXRpclJzUXVRb1hxNkhQS1FIeUVITHo2aWVVMHJsV3IyN0VyQkJ4RlRKTm51MnJ1MHV1
Ly8wdG1SYjgzZWwKSUpXQnY1V1diSnl4dXNnMkhkc0tzTUh0eEVaYWh1UlpTNHU2TURQR3dSdj
RaU0xpQm1FVVc3RUMwUEg3dCtGaAoxUDcrL0Yyd1pGSDAvSzl6eXUyc0lOMDJIbTBmSWtGejBx
b09BSzQ5OXhrQ2dZRUE2SC9nVUJoOG9GUSt2cmZKCnQvbXdMeFBHZHhWb3FWR1hFVjhlQzNWbm
xUSXJlREpNWm81b1hKZHNuQ0d2S1NaWUhXZ3o3SVpwLzRCL29vSWsKTDl4TEJSVTJwS0d1OGxB
T1ZhYnpaVDk0TTZYSE1PTGQ0ZlUrS3ZqK1lLVm5laEM3TVNQL3RSOWhFMjN1MnRKZwp1eUdPRk
lFVlptNHZxS1hEelU3TTNnU0R5WXNDZ1lFQTRJRVFyZDl2MXp0T2k5REZ6WEdnY05LVmpuYmFT
WnNXCm9JNm1WWFJZS1VNM1FyWUw4RjJTVmFFM0Y0QUZjOXRWQjhzV0cxdDk4T09Db0xrWTY2Nj
ZqUFkwMXBWTDdXeTMKZXpwVEFaei9tRnc2czdic3N3VEtrTW5MejVaNW5nS3dhd3pRTXVoRGxL
TmJiUi90enRZSEc0NDRrQ2tQS3JEbQphOG40bUt6ZlRuVUNnWUFTTWhmVERPZU1BS3ZjYnpQSl
F6QkhydXVFWEZlUmtNSWE2Ty9JQThzMGdQV245WC9ICk12UDE4eC9iNUVMNkhIY2U3ZzNLUUFi
QnFVUFQ2dzE3OVdpbG9EQmptQWZDRFFQaUxpdTBTOUJUY25EeFlYL3QKOUN5R1huQkNEZy9ZSE
1FWnFuQ1RzejM4c0VqV05VcSt1blN0SkVFUmdDUVl0Y2hxSS9XaWxvWGQyd0tCZ1FEQworTlBY
YlBqZ1h5MHoxN2d4VjhFU3VwQVFEY0E5dEdiT1FaVExHaU9Ha2sxbnJscG9BWnVZcWs0Q0pyaV
ZpYUlyCkJvRElWWpDcjVNK3FnR3VqU3lPUnpSVU40eWRRWkdIZjN1Zkp3NEM3L1k3SlY0amlz
R3hSTSt3Rk9yQ0EydmIKVEdGMEZZLcThaN0o2N3dQRVliUUNobDB4TmJkcVIvK1ZGTzdGQ1QxV0
VRS0JnQThUaE9hZmNEUmdpd0IxRFdyRgozZ1lmT3I0dERENExrNjRYZlF6ajdtRXQyYlJzOFNE
YXYwVGZPclVUUlpFTTkyTVFZMnlrbzhyMDJDbmpndmxCCm1aYnZCTEFYaVZLa0laai9TTkNYUn
hzOFZkZ3psTkpzYVNZTUtsNloxK1Z3MnZUdDNQSnI0TXlhRWpHYUxlSmMKRGRTQjdYOU9ESk5a
cW10bGpoRzc5eXpQCi0tLS0tRU5EIFBSSVZBVEUgS0VZLS0tLS0=
kind: Secret
metadata:
  name: mysecret
  namespace: default
type: kubernetes.io/tls
```

With that secret in place, we can start securing. First, we need to update the Gateway to create a TLS listener with that certificate. That is possible through the `certificates` option on the helm chart which we can use for upgrading

```yaml
# 05-values.yaml
---
experimental:
  kubernetesGateway:
    appLabelSelector: traefik
    certificates:
      -
        group: "core"
        kind: "Secret"
        name: "mysecret"
    enabled: true
```

```
helm upgrade traefik -f values.yaml traefik/traefik
```

Once upgrades, lets see the result:

```
curl --insecure -H "Host: whoami" https://localhost/foo
Hostname: whoami-9cdc57b6d-pfpxs
IP: 127.0.0.1
IP: ::1
IP: 10.42.0.13
IP: fe80::9c1a:a1ff:fead:2663
RemoteAddr: 10.42.0.11:53158
GET /foo HTTP/1.1
Host: whoami
User-Agent: curl/7.64.1
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 10.42.0.1
X-Forwarded-Host: whoami
X-Forwarded-Port: 443
X-Forwarded-Proto: https
X-Forwarded-Server: traefik-74d7f586dd-xxr7r
X-Real-Ip: 10.42.0.1
```

That's it 😃

Canary Releases

The last feature we support out of the specification in terms of routing capabilities, is canary releases!

For that, we need a second service to run first. For the sake of this example, we will quickly spawn an nginx:

```yaml
# 06-nginx.yaml
---
kind: Deployment
```

```yaml
apiVersion: apps/v1
metadata:
  name: nginx

spec:
  replicas: 2
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: whoami
          image: nginx
          ports:
            - containerPort: 80
              name: http

---
apiVersion: v1
kind: Service
metadata:
  name: whoami

spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: http
  selector:
    app: nginx
```

The HTTPRoute has a weight option, which we can utilize for that.

```yaml
# 07-whoami-nginx-canary.yaml
---
apiVersion: networking.x-k8s.io/v1alpha1
kind: HTTPRoute
metadata:
  labels:
    app: traefik
  name: http-app-1
  namespace: default
spec:
  hostnames:
    - whoami
  rules:
    -
      forwardTo:
        -
          port: 80
          serviceName: whoami
          weight: 3
        -
          port: 80
          serviceName: nginx
          weight: 1
```

Now, every fourth curl request will show a different result 😃

Status Resources to the Rescue

The Service API specification heavily utilizes Status Resources to show issues with your configuration.

Some can easily be reproduced when you use a wrong port on your Gateway or when you utilize a not yet implemented protocol which will be handled as an invalid value error:

```
---
Spec:
  Controller: traefik.io/gateway-controller
Status:
  Conditions:
    ? "Last Transition Time"
    : 2021-01-27 15:22:07 +00:00
    Message: "Handled by Traefik controller"
    Reason: Handled
    Status: Unknown
    Type: InvalidParameters
```

There are plenty more, so we recommend checking them out on the official documentation.

Known Limitations and Future

Currently, our implementation is focussing on HTTP and HTTPS only. However, the spec also features TCP and in the future probably UDP as well which is something we will be working on. Also, we want to improve the need to know which ports Traefik has oben to do the exact matching on a Gateway Ressource. Also, more advanced cases such as traffic splitting are not yet implemented. Last but not least, there is some more logic required in terms of default values for extensions through configmaps. That's all on our list and will be improved eventually as the spec evolves.