Kubernetes Traefik Ingress Controller

Installation Kubernetes Traefik Ingress Controller

Zur einfachen Installation des Kubernetes Traefik Ingress Controllers solltest Du Helm verwenden. Helm bezeichnet sich selbst als Paketmanager für Kubernetes-Anwendungen. Neben der Installation bietet Helm auch einfache Updates seiner Anwendungen. Wie auch bei kubectl brauchst Du nur die K8s-Config, um direkt loszulegen:

```
    $ helm repo add traefik https://helm.traefik.io/traefik
    $ helm repo update
    $ helm install --create-namespace --namespace=traefik-system traefik
traefik/traefik
```

NOTES: The ingress-traefik controller has been installed. It may take a few minutes for the LoadBalancer IP to be available. You can watch the status by running 'kubectl –namespace traefik-system get services -o wide -w traefik'

An example Ingress that makes use of the controller:

```
  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    name: example
    namespace: foo
  spec:
    ingressClassName: traefik
    rules:
      - host: www.example.com
        http:
          paths:
            - pathType: Prefix
              backend:
                service:
                  name: exampleService
                  port:
                    number: 80
              path: /
    # This section is only required if TLS is to be enabled for the Ingress
    tls:
      - hosts:
        - www.example.com
        secretName: example-tls
```

Es ist auch möglich ein Ingress per create zu erzeugen. Hier ein kleines Beispiel dazu

```
kubectl create ingress webserver-red --namespace red --rule
'ingress-0.training.lab/red*=webserver-red:80' --dry-run=client -o yaml
```

Bei einem Aufrug mit Hostname ingress-0.training.lab und path red/* gebe die Anfrage weiter an den Service mit Namen webserver-red auf Port 80 Wenn man das * hinter red/ wegläst dann wird „pathType: Exact“ gesetzt statt „pathType: Prefix“

If TLS is enabled for the Ingress, a Secret containing the certificate and key must also be provided:

```
k create secret tls stirlingpdf-tls --key
/home/marko/Downloads/pdftools.tuxnet.lan.key --cert
/home/marko/Downloads/pdftools.tuxnet.lan.crt -n stirlingpdf-testing
```

or

```
  apiVersion: v1
  kind: Secret
  metadata:
    name: example-tls
    namespace: foo
  data:
    tls.crt: <base64 encoded cert>
    tls.key: <base64 encoded key>
  type: kubernetes.io/tls
```

Es ist auch Möglich ein default TLS in einem TLSStore zu hinterlegen

```
apiVersion: traefik.io/v1alpha1
kind: TLSStore
metadata:
  name: default
  namespace: default
spec:
  defaultCertificate:
    secretName: ingress-certificate
```

Forwarding SSH traffic inside Kubernetes using Traefik

Are you running a Gitea or Gitlab instance inside your Kubernetes cluster? And you want to reach it not only via HTTPS, but also via SSH for easier pulling and pushing?

This article describes how to setup Traefik as ingress controller to do that, using Gitea as an example.

The necessary steps are:

1. enable the gitea-ssh service

2. configure Traefik to recognize and open the new ports
3. create an ingressrouteTCP resource to tell Traefik what to do
4. configure your client
5. make sure everything is working

In case you are new to Traefik, we recently published a (German-only) introduction to Traefik.

enable the gitea-ssh service

If you installed gitea from the official helm chart, then you just need to add some lines to your values.yaml to enable the gitea-ssh service:

```
service:
  ssh:
    type: ClusterIP
    port: 22
    clusterIP:
```

The empty clusterIP is not an error, the default for that values is none, and in this way we overwrite it and tell it to use a real ClusterIP.

After modifying the values.yaml, upgrade gitea using helm, e.g. helm upgrade gitea gitea/gitea -f values.yaml and wait for everything to fully settle. Make sure the pods are up and the service is shown:

```
$ kubectl get svc
NAME                         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE
gitea-http                   ClusterIP   None           <none>
3000/TCP   6d3h
gitea-postgresql-headless    ClusterIP   None           <none>
5432/TCP   6d3h
gitea-postgresql             ClusterIP   10.62.43.161   <none>
5432/TCP   6d3h
gitea-ssh                    ClusterIP   10.62.71.156   <none>        22/TCP
3d7h
```

configure Traefik to recognize the gitea-ssh port

If you want your Gitea instance to be reachable via port 22, which is the default port for SSH, you run into two problems:

First, the Traefik pods is running with low privileges, i.e. as a non-root user. That means it cannot bind to ports under 1024. That is intentional and good admin practice. Second, your kubernetes node might be using port 22 for its own SSH connection.

So, unless you have important reasons for really using port 22, the safer and easier approach is to use a free high port. In the following example we will be using 55522. See the footnote below if you really need port 22. But be warned…

In case you installed Traefik via the official helm chart, add the following section to your values.yaml file and upgrade Traefik (e.g. helm upgrade traefik traefik/traefik -f values.yaml or similar).

```
ports:
  gitea-ssh:
    port: 55522
    expose: true
```

Verify that the settings are reflected in the Traefik deployment:

```
$ kubectl get deployment traefik -n traefik-system -o yaml
[...]
    spec:
      containers:
      - args:
        - --global.checknewversion
        - --global.sendanonymoususage
        - --entrypoints.gitea-ssh.address=:55522/tcp
        - --entrypoints.traefik.address=:9000/tcp
        - --entrypoints.web.address=:8000/tcp
        - --entrypoints.websecure.address=:8443/tcp
[...]
```

create an IngressRouteTCP resource

Kubernetes itself only offers a very limited featureset for ingresses, mainly HTTP/HTTPS. This is where Traefik shines, as it comes with its own CRDs (custom resource definitions) that can be used until the Kubernetes ApiGateway specification is stable enough. This ApiGateway is intended as a successor to ingresses and will (hopefully) be supported by all ingress controllers, while the Traefik CRDs are only working with Traefik.

The following IngressRouteTCP resource configures Traefik to forward TCP traffic that is arriving on port 55522 to the gitea-ssh service that is listening on port 22:

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRouteTCP
metadata:
  name: gitea-ssh
  namespace: gitea
spec:
  entryPoints:
    - gitea-ssh
  routes:
  - match: HostSNI(`*`)
    services:
    - name: gitea-ssh
      port: 22
```

Side note: The HostSNI option is mandatory for an IngressRouteTCP object, even if it is completely useless in case of SSH. As SSH is not using the TLS protocol, it is not sending any SNI information. This means, that you cannot forward traffic to different recipient services depending on the hostname that is given, like you are used to with HTTP traffic (in webserver terminology this is called virtual hosts or host-based routing). In case of SSH, you need a separate port for each of your services.

configure your client to use the proper port

As we are not using port 22, we need to configure our clients to use the correct port.

This can be done either during cloning of the repository with git or by setting the appropriate parameters in your ~/.ssh/config file.

And, of course, before you can do that, you need to add your public SSH key to your user's profile in your Gitea instance.

For git, use the full URL that looks something like ssh:*git@gitea.example.org:55522/my-gitea-user/my-repository.git: <code bash> $ git clone ssh:*git@gitea.example.org:55522/my-gitea-user/my-repository.git Cloning into 'my-repository'... remote: Enumerating objects: 3, done. remote: Counting objects: 100% (3/3), done. remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 Receiving objects: 100% (3/3), done. </code> If you want to configure this once so you do not need to remember for each clone, add this to your ~/.ssh/config file:

```
Host gitea.example.org
     Port 55522
```

Then just clone using the URL without the port number:

```
$ git clone git@gitea.example.org:/my-gitea-user/my-repository.git
Cloning into 'my-repository'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

Hooray, it works!

Before you go off to party and celebrate your success: In case it is not only you that will be using this gitea instance, it might make sense to distribute the SSH settings to other users. Those can easily be added to your /etc/ssh_config (or the equivalent on your operation system) via your company's configuration management system. Ask your local IT administrator and send her my best wishes!

Footnotes

In case you really need Gitea to be reachable on port 22, use the following snippet for your Traefik values.yaml:

```
ports:
```

```
  gitea-ssh:
    port: 22
    expose: true

securityContext:
  capabilities:
    drop: [ALL]
    add: [NET_BIND_SERVICE]
  runAsGroup: 0
  runAsNonRoot: false
  runAsUser: 0
```

And yes, this is really NOT A GOOD IDEA...

Ingress zusammen mit dem cert-manager

Um ein für Ingress benötigtes TLS-Zertifikat mit dem cert-manager automatisch zu erstellen, muss zunächst ein Issuer oder ClusterIssuer definiert werden. Der cert-manager stellt die Zertifikate basierend auf diesen Issuers aus.

Beispiel für einen ClusterIssuer mit Let's Encrypt:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: your-email@example.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
      - http01:
          ingress:
            class: nginx
```

Sobald der ClusterIssuer vorhanden ist, kann die Ingress-Ressource so konfiguriert werden, dass sie automatisch ein Zertifikat anfordert, indem die entsprechenden Annotationen hinzugefügt werden:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
```

```
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - example.com
    secretName: example-tls  # cert-manager speichert das Zertifikat in
diesem Kubernetes-Secret
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: example-service
            port:
              number: 80
```

Dank der Annotation `cert-manager.io/cluster-issuer: „letsencrypt-prod“` erkennt der cert-manager automatisch, dass ein TLS-Zertifikat benötigt wird. Er erstellt daraufhin eine `Certificate`-Ressource und kümmert sich um die Ausstellung und Erneuerung des Zertifikats. Das Zertifikat wird im angegebenen Secret (`example-tls`) gespeichert und vom Ingress-Controller für HTTPS verwendet.

Trouble Shooting

Um Traefik-Ingresse in ArgoCD nicht dauerhaft als „Progressing“ angezeigt zu bekommen, folgende Values im Traefik Helm Chart setzen:

```
globalArguments:
  - "--providers.kubernetesingress.ingressendpoint.publishedservice=traefik-
system/traefik"
```

From:
https://www.cooltux.net/ - **TuxNet DokuWiki**

Permanent link:
https://www.cooltux.net/doku.php?id=it-wiki:kubernetes:ingress-traefik-loadbalancer&rev=1740116764

Last update: **2025/02/21 05:46**