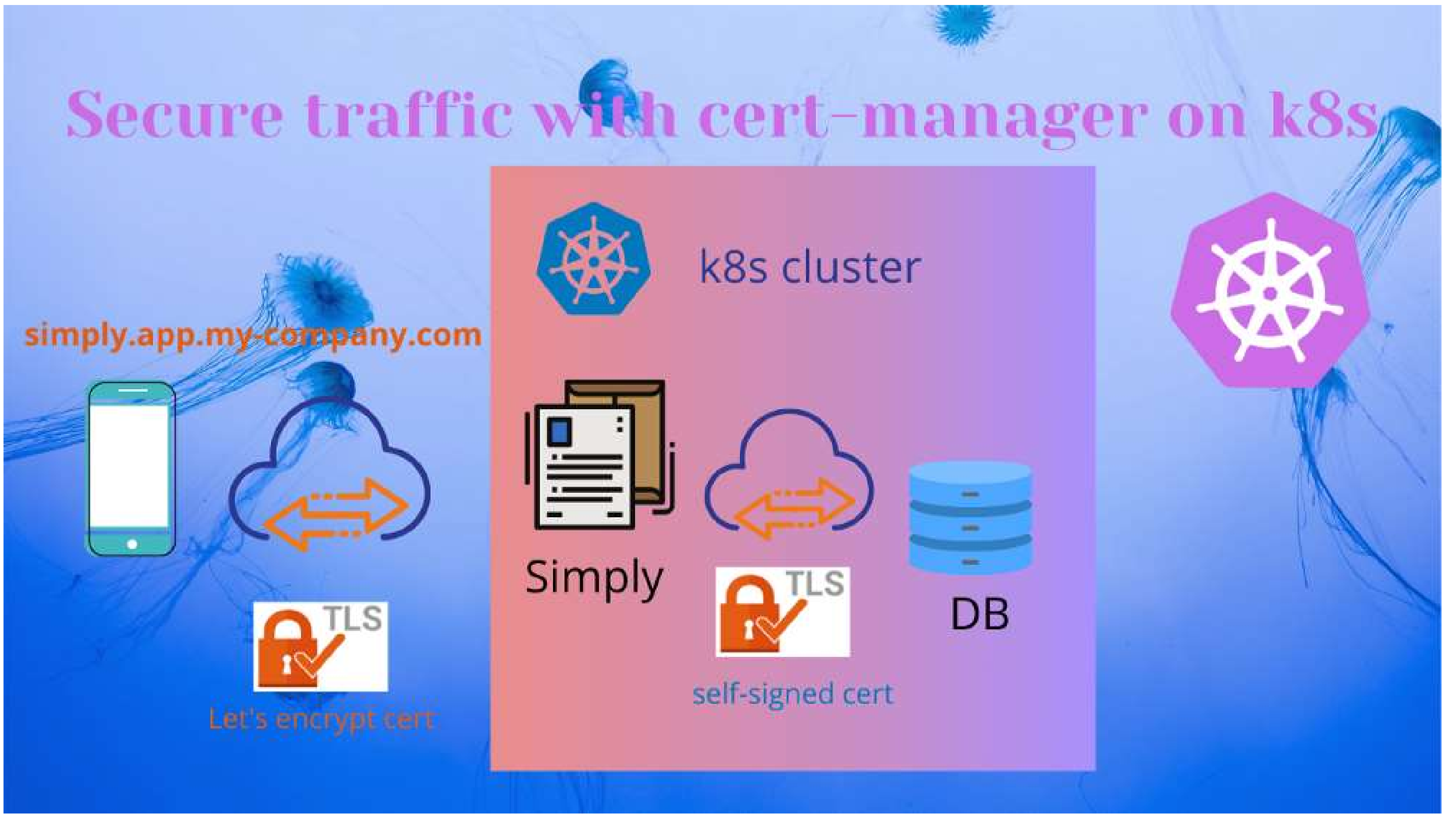
Secure traffic to your application with Kubernetes and cert-manager

Igor Zhivilo
Apr 6 · 5 min read


Secure traffic with cert-manager on k8s
k8s cluster
simply.app.my-company.com
Simply
DB
TLS
Let's encrypt cert
TLS
self-signed cert

As a DevOps engineer at Cloudify.co, I am working on the creation of the Production Kubernetes (EKS) cluster with all needed mechanisms for our SaaS, including the TLS certificates issuing for secure communication to and within the k8s cluster.

In this tutorial, I will show how to secure external and internal traffic coming to your application on the Kubernetes cluster by issuing the TLS certificates with the Cert-Manager.

To simplify things let's introduce the '**simply**' web app which sits on the Kubernetes cluster, gets external traffic, and talks to DB (internal traffic) which also installed on the same k8s cluster.

Let's make an assumption that we are able to reach the '**simply**' app on this **simply.app.my-company.com** domain using the HTTP protocol (port 80).

What I am trying to achieve?

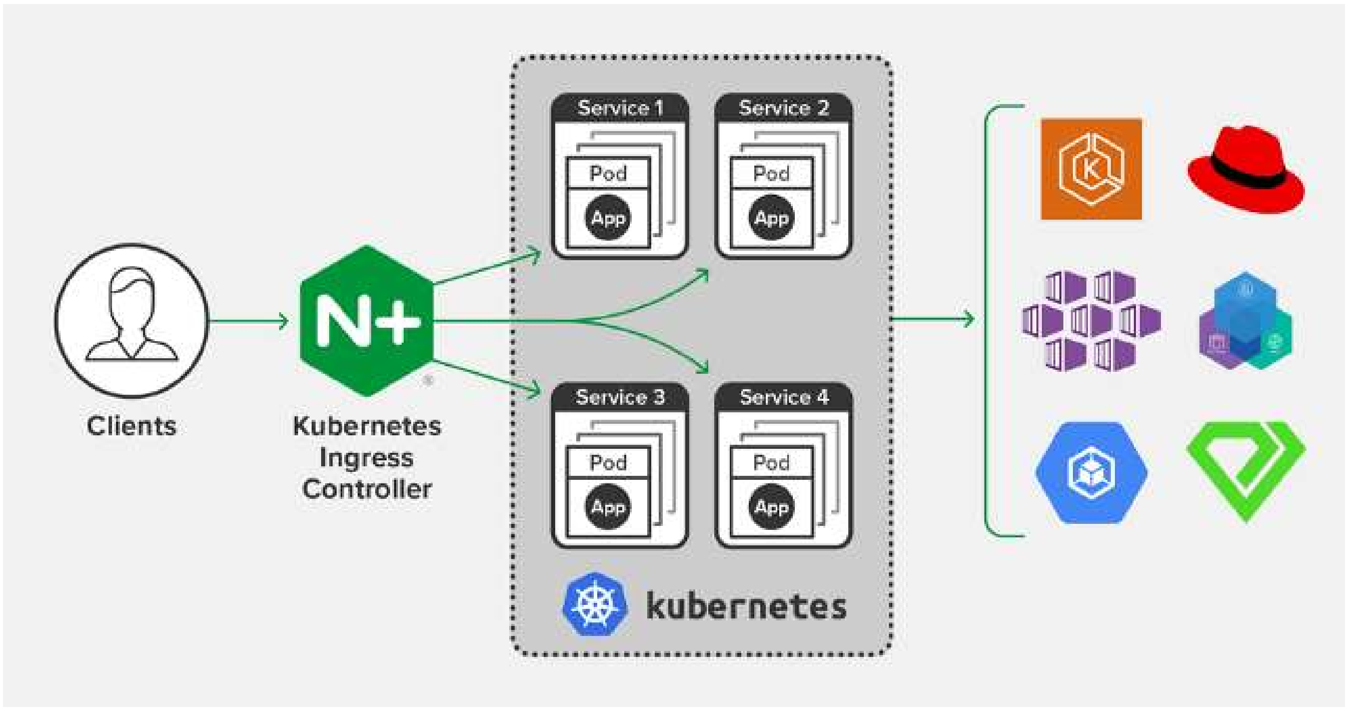
1. Secure all external traffic to the 'simply' app using TLS certificate from Let's Encrypt and adding communication through port 443 (HTTPS)

2. Secure internal traffic between the 'simply' app and DB using the self-signed certificate.

What is NGINX Ingress?

> ingress-nginx is an Ingress controller for Kubernetes using NGINX as a reverse proxy and load balancer.

https://github.com/kubernetes/ingress-nginx



To install Nginx Ingress we will use the Helm package manager.

What is cert-manager?

> *cert-manager is a native Kubernetes certificate management controller. It can help with issuing certificates from a variety of sources, such as Let's Encrypt, HashiCorp Vault, Venafi, a simple signing key pair, or self signed.*

> *It will ensure certificates are valid and up to date, and attempt to renew certificates at a configured time before expiry.*
>
> *It is loosely based upon the work of kube-lego and has borrowed some wisdom from other similar projects such as kube-cert-manager.*

So how to achieve this?

Prerequisites

- Kubernetes cluster (EKS in my case)
- Ingress-Nginx installed to Kubernetes cluster
- Cert-Manager Installed to Kubernetes cluster
- Configured 'A' record which points to your Load Balancer created by Ingress, basically, an alias which points to created LB by Ingress.

You can read about the installation and configuration of all those components in details in my post: https://medium.com/@warolv/building-the-ci-cd-of-the-future-nginx-ingress-cert-manager-945f3dc6b12e

Configure Cluster Issuer to issue Let's Encrypt certificates

To obtain Let's Encrypt certificates we need to create an issuer, it may be Issuer or ClusterIssuer, the difference is that an Issuer is scoped to a single namespace and ClusterIssuer is a cluster-wide version of an Issuer.

Creation of production 'Cluster Issuer' for Let's Encrypt certificates:

```
# cluster-issuer.yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    # The ACME server URL
    server: https://acme-v02.api.letsencrypt.org/directory
    # Email address used for ACME registration
    email: admin@app.my-company.com
    # Name of a secret used to store the ACME account private key
    privateKeySecretRef:
      name: letsencrypt-prod
    # Enable the HTTP-01 challenge provider
    solvers:
      - http01:
          ingress:
            class: nginx
```

We using HTTP Validation and ACME protocol for ClusterIssuer

Deploy cluster-issuer.yaml

```
kubectl create -f cluster-issuer.yaml
```

Issue a new Let's Encrypt certificate with cert-manager for simply.app.my-company.com

To issue a new certificate we will create the Ingress-Nginx rule for the **simply.app.my-company.com** domain, which uses cert-manager and cluster-issuer we created to issue a certificate from Let's encrypt.

> *Pay attention to 'annotations' and 'tls' parts, you can see the magic of integration between ingress and cert-manager by using annotations.*

```
# simply-ingress.yaml
apiVersion: extensions/v1
kind: Ingress
metadata:
  name: simply-ing
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  tls:
  - hosts:
    - simply.app.my-company.com
    secretName: simply-tls-prod
  rules:
  - host: simply.app.my-company.com
    http:
      paths:
      - path: /
        backend:
          serviceName: simply-svc
          servicePort: 80
```

I make an assumption that you can reach the 'simply' app inside of the cluster via the 'simply-svc' service and port 80.

Deploy simply-ingress.yaml

```
kubectl create -f simply-ingress.yaml
```

Now you need to wait till Cert-Manager acquires a certificate for the **simply.app.my-company.com** domain, it may take some time. When the certificate will be acquired you will be able to reach **simply.app.my-company.com** using HTTPS.

You can check acquired certificates and status using 'kubectl':

```
$ kubectl get certificates -n cert-manager
```

Issue self-signed certificate (signed by CA) with cert-manager for securing internal traffic between 'simply' and DB

> *I will make an assumption that 'simply' will be deployed to '**simplyns**' namespace:*

```
$ kubectl create ns simplyns
```

Create self-signed certificate issuer:

> *notice that Issuer is scoped to a single namespace*

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-issuer
spec:
  selfSigned: {}
```

Create a CA certificate that will be used to sign other certificates:

> *'isCA' must be true*

```
apiVersion: cert-manager.io/v1
kind: Certificate
```

```
metadata:
  name: simplyns-ca
spec:
  secretName: simplyns-ca-tls
  commonName: simplyns.svc.cluster.local
  usages:
    - server auth
    - client auth
  isCA: true
  issuerRef:
    name: selfsigned-issuer
```

Check generated certificate/secret created:

```
kubectl get certificate simplyns-ca -n simplyns
kubectl get secret simplyns-ca-tls -n simplyns
```

Create an additional Issuer to issuer certificates signed by CA

> *notice that we using previously generated for CA secret*

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  namespace: simplyns
  name: simply-ca-issuer
spec:
  ca:
    secretName: simplyns-ca-tls
```

And finally, generate a certificate signed by CA for 'simply' and DB communication:

> *simply deployed to **simplyns** namespace*
>
> *db deployed to **dbns namespace***
>
> *'simply.simplyns.svc.cluster.local' is internal dns domain of 'simply'*
>
> *'db.dbns.svc.cluster.local' is internal dns domain of DB*

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
    namespace: simplyns
    name: simply-cert
  spec:
    secretName: simply-certs
    isCA: false
    usages:
      - server auth
      - client auth
    dnsNames:
    - "simply.simplyns.svc.cluster.local"
    - "db.dbns.svc.cluster.local"
    issuerRef:
      name: simply-ca-issuer
```

You can check generated secret is created:

```
  kubectl get secret simply-cert -n simplyns
```

Conclusion

In this tutorial, I explained how to secure external and internal traffic coming to your application on the Kubernetes cluster by issuing the TLS certificates with the Cert-Manager, creating Cluster Issuer for Let's Encrypt certificates, and self-signed certificates issuer for internal communications.

Thank you for reading, I hope you enjoyed it, see you in the next post.

If you want to be notified when the next post of this tutorial is published, please follow me here on medium and on Twitter (@warolv).

My personal blog: http://igorzhivilo.com

References

https://github.com/kubernetes/ingress-nginx

https://cert-manager.io/docs

https://medium.com/@warolv/building-the-ci-cd-of-the-future-nginx-ingress-cert-manager-945f3dc6b12e

👏 Join FAUN today and receive similar stories each week in your inbox! Get your weekly dose of the must-read tech stories, news, and tutorials.

Follow us on Twitter 🐦 and Facebook 👥 and Instagram 📷 and join our Facebook and Linkedin Groups 💬

If this post was helpful, please click the clap 👏 button below a few times to show your support for the author! ⬇

Kubernetes Cert Manager Tls Certificate Lets Encrypt K8s