Kubernetes Cert-Manager

The `cert-manager` is the modern replacement for jetstack's previous `kube-lego` project. The purpose of this project is to automate TLS certificate renewal on Kubernetes via LetsEncrypt.

We can currently set up wildcard TLS via LetsEncrypt manually in the cluster using Craig's fantastic instructions: Wildcard Certs via LetsEncrypt

If cert-manager can be used in a similar fashion to automate this setup entirely, that would be even better.

NOTE - These instructions were taken mostly from some random blog post: https://itnext.io/automated-tls-with-cert-manager-and-letsencrypt-for-kubernetes-7daaa5e0cae4

Prerequisites

- A 1+ node Kubernetes cluster
- Helm and Tiller installed on Kubernetes cluster
- NGINX Ingress Controller running in Kubernetes cluster
- A DNS record setup to point at the IP of the NGINX controller

Let's Encrypt: Staging vs Production

Before we get into any of this, I'd like to point out that we used the staging server for this example because it is much more forgiving about how often you can request a new certificate.

Their production ACME server has a **HARD LIMIT** of 20 certificate requests (per domain) every 7 days. I think this limit is even lower for wildcard certificates (possibly closer 5 requests every 7 days).

In short, DO NOT use LetsEncrypt production servers for testing - you will have a bad time.

Instead, use their staging server to verify your setup, and then only migrate to production once you're sure everything is working.

For more information on the rate limits in place on LetsEncrypt, see https://letsencrypt.org/docs/rate-limits/

Step 1: Install cert-manager via helm

Create `cert-manager-values.yaml` on disk:

cert-manager-values.yaml

```
---
ingressShim.defaultIssuerName: letsencrypt-staging
ingressShim.defaultIssuerKind: ClusterIssuer
```

Then install `cert-manager` using the official helm chart:

```
$ helm install --name my-release -f cert-manager-values.yaml cert-manager
```

Once the helm chart completes its installation, you should see that the `cert-manager` Deployment has been created in the `kube-system` namespace, and that it has started a Pod that is now running:

```
$ kubectl get all -n kube-system -l app=cert-manager
NAME                                 READY     STATUS    RESTARTS   AGE
pod/cert-manager-75577f4545-xbz5j   1/1       Running   0          3m

NAME                                 DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/cert-manager   1         1         1            1           3m

NAME                                            DESIRED   CURRENT   READY     AGE
replicaset.extensions/cert-manager-75577f4545   1         1         1         3m

NAME                            DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cert-manager   1         1         1            1           3m

NAME                                       DESIRED   CURRENT   READY     AGE
replicaset.apps/cert-manager-75577f4545   1         1         1         3m
```

If errors occur while issuing certificates, they will appear in these pod logs - this should always be your first stop in debugging this process:

```
$ kubectl logs -f deploy/cert-manager -n kube-system
```

Step 2: Create a ClusterIssuer

An `Issuer` or `ClusterIssuer` represents the CA that will distribute certificates for your ingress rules - in our case, this is LetsEncrypt.

NOTE: An Issuer is namespace-bound, whereas a ClusterIssuer can work with any namespace.

Create a `letsencrypt-staging.yaml` pointing at the LetsEncrypt ACME staging server:

letsencrypt-staging.yaml

```
apiVersion: certmanager.k8s.io/v1alpha1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
spec:
  acme:
    # The ACME server URL
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    # Email address used for ACME registration
    email: myemail@gmail.com
    # Name of a secret used to store the ACME account private key
    privateKeySecretRef:
      name: letsencrypt-staging
    # Enable the HTTP-01 challenge provider
    http01: {}
```

Then create the `ClusterIssuer` in Kubernetes:

```
$ kubectl create -f letsencrypt-staging.yaml
clusterissuer.certmanager.k8s.io/letsencrypt-staging created
```

If you check the `cert-manager` Pod logs, you will see that it has picked up our new ClusterIssuer:

```
$ kubectl logs -f deploy/cert-manager -n kube-system
I0723 15:29:15.514705       1 start.go:63] starting cert-manager v0.4.0 (revision
e66648a7a70befe4096274f3f99bda248a7c371b)
I0723 15:29:15.516948       1 controller.go:111] Using the following nameservers for DNS01 checks: [10.96.0.10:
53]
I0723 15:29:15.517140       1 server.go:68] Listening on http://0.0.0.0:9402
I0723 15:29:15.518036       1 leaderelection.go:175] attempting to acquire leader lease  kube-system/cert-
manager-controller...
I0723 15:29:15.542612       1 leaderelection.go:184] successfully acquired lease kube-system/cert-manager-
controller
I0723 15:29:15.543680       1 controller.go:53] Starting ingress-shim controller
I0723 15:29:15.544348       1 controller.go:53] Starting issuers controller
I0723 15:29:15.544469       1 controller.go:53] Starting certificates controller
I0723 15:29:15.546478       1 controller.go:53] Starting clusterissuers controller
I0723 15:29:15.644004       1 controller.go:152] ingress-shim controller: syncing item 'default/cloud9'
I0723 15:29:15.644053       1 sync.go:49] Not syncing ingress default/cloud9 as it does not contain necessary
annotations
I0723 15:29:15.644067       1 controller.go:166] ingress-shim controller: Finished processing work item "default
/cloud9"
I0723 15:29:15.644057       1 controller.go:152] ingress-shim controller: syncing item 'cis-hackathon-2018/cis-
girder'
I0723 15:29:15.644118       1 controller.go:152] ingress-shim controller: syncing item 'cis-dev/cis-girder'
I0723 15:29:15.644135       1 sync.go:49] Not syncing ingress cis-dev/cis-girder as it does not contain
necessary annotations
   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
I0726 19:24:19.755803       1 controller.go:138] clusterissuers controller: syncing item 'letsencrypt-staging'
I0726 19:24:20.099674       1 acme.go:162] getting private key (letsencrypt-staging->tls.key) for acme issuer
kube-system/letsencrypt-staging
I0726 19:24:20.102043       1 setup.go:46] letsencrypt-staging: generating acme account private key
"letsencrypt-staging"
I0726 19:24:21.025144       1 logger.go:67] Calling GetAccount
I0726 19:24:22.755104       1 logger.go:62] Calling CreateAccount
I0726 19:24:23.076269       1 setup.go:73] letsencrypt-staging: verified existing registration with ACME server
I0726 19:24:23.076346       1 helpers.go:134] Setting lastTransitionTime for ClusterIssuer "letsencrypt-
staging" condition "Ready" to 2018-07-26 19:24:23.076326488 +0000 UTC m=+273307.575648035
I0726 19:24:23.086113       1 controller.go:152] clusterissuers controller: Finished processing work item
"letsencrypt-staging"
```

You should also see that cert-manager has stored your LetsEncrypt ACME private key in a secret:

```
$ kubectl get secret -n kube-system | grep letsencrypt
NAME                                          TYPE                                  DATA      AGE
letsencrypt-staging                           Opaque                                1         4m
```

If you see all of the above, then you should be ready to issue some certificates!

Step 3: Annotate Your Ingress

In the startup logs for the cert-manager, you may have seen info message about missing annotations, for example:

```
I0723 20:54:03.473218       1 sync.go:49] Not syncing ingress default/cloud9 as it does not contain necessary
annotations
```

This is because our ingress rules need to have a special annotation for cert-manager to operate on them. I believe the intention here is to prevent unauthorized persons from another namespace from modifying your ingress rules without your permission, and for larger/heterogeneous clusters to safely label their workloads as appropriate.

Simply Edit one of your ingress rules to add a single line - you'll need to point the `certmanager.k8s.io/cluster-issuer` annotation at the name of your `ClusterIssuer`.

For example, with `kubectl edit -n cis-dev ing cis-girder` I would add the following annotation:

cis-dev.ingress.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    certmanager.k8s.io/cluster-issuer: letsencrypt-staging    <---- Add this line
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
  name: cis-girder
  namespace: cis-dev
spec:
  rules:
  - host: dev.cis.ndslabs.org
    http:
      paths:
      - backend:
          serviceName: cis-girder
          servicePort: 80
        path: /
      - backend:
          serviceName: cis-girder
          servicePort: 8080
        path: /girder
      - backend:
          serviceName: cis-girder
          servicePort: 8080
        path: /static
      - backend:
          serviceName: cis-girder
          servicePort: 8080
        path: /api
  tls:
  - hosts:
    - dev.cis.ndslabs.org
    secretName: cis-tls-secret
```

NOTE: The rest of the ingress fields should be able to stay the same. Only those rules with the annotation will be affected by cert-manager, so only add it to ingress rules for which you'd like for it to control/renew TLS.

Step 4: Issue a Certificate

Create `dev.cis.ndslabs.org-certificate.yaml` on disk:

dev.cis.ndslabs.org-certificate.yaml

```
apiVersion: certmanager.k8s.io/v1alpha1
kind: Certificate
metadata:
  name: cis-tls-cert
spec:
  secretName: cis-tls-secret
  dnsNames:
  - dev.cis.ndslabs.org
  acme:
    config:
    - http01:
        ingressClass: nginx
      domains:
      - dev.cis.ndslabs.org
  issuerRef:
    name: letsencrypt-staging
    kind: ClusterIssuer
```

Then pass this file to `kubectl create`:

```
$ kubectl create -f dev.cis.ndslabs.org-certificate.yaml
```

NOTE: I suspect that this is required, but did not attempt otherwise - as you can see above, I did explicitly specify a namespace here to make sure my certificate/secret was created in the same namespace as my ingress rule. I have no idea how it would otherwise figure out which namespaces needs which secrets.

What should you see? (Console)

That's it! If everything worked correctly, you should see some successful log messages in the cert-manager logs:

```
$ kubectl logs -f -n kube-system deploy/cert-manager
   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
I0726 19:29:40.287760       1 controller.go:152] ingress-shim controller: syncing item 'cis-dev/cis-girder'
I0726 19:29:40.287796       1 sync.go:124] Certificate "cis-tls-secret" for ingress "cis-girder" already exists
I0726 19:29:40.287857       1 sync.go:127] Certificate "cis-tls-secret" for ingress "cis-girder" is up to date
I0726 19:29:40.287914       1 controller.go:166] ingress-shim controller: Finished processing work item "cis-dev
/cis-girder"
I0726 19:29:44.287733       1 controller.go:177] certificates controller: syncing item 'cis-dev/cis-tls-secret'
I0726 19:29:44.287981       1 sync.go:259] Preparing certificate cis-dev/cis-tls-secret with issuer
I0726 19:29:44.288047       1 acme.go:162] getting private key (letsencrypt-staging->tls.key) for acme issuer
kube-system/letsencrypt-staging
I0726 19:29:44.288886       1 prepare.go:247] Cleaning up previous order for certificate cis-dev/cis-tls-secret
I0726 19:29:44.288918       1 prepare.go:263] Cleaning up old/expired challenges for Certificate cis-dev/cis-
tls-secret
I0726 19:29:44.288981       1 logger.go:22] Calling CreateOrder
I0726 19:29:44.862177       1 acme.go:196] Created order for domains: [{dns dev.cis.ndslabs.org}]
I0726 19:29:44.862261       1 logger.go:52] Calling GetAuthorization
I0726 19:29:44.959729       1 prepare.go:263] Cleaning up old/expired challenges for Certificate cis-dev/cis-
tls-secret
I0726 19:29:44.959768       1 helpers.go:188] Found status change for Certificate "cis-tls-secret" condition
"ValidateFailed": "False" -> "False"; setting lastTransitionTime to 2018-07-26 19:29:44.959762093 +0000 UTC
m=+273629.459083531
I0726 19:29:44.959795       1 sync.go:266] Issuing certificate...
I0726 19:29:44.959830       1 acme.go:162] getting private key (letsencrypt-staging->tls.key) for acme issuer
kube-system/letsencrypt-staging
I0726 19:29:44.960303       1 logger.go:27] Calling GetOrder
I0726 19:29:45.258232       1 logger.go:37] Calling FinalizeOrder
I0726 19:29:46.062504       1 issue.go:104] successfully obtained certificate: cn="dev.cis.ndslabs.org"
altNames=[dev.cis.ndslabs.org] url="https://acme-staging-v02.api.letsencrypt.org/acme/order/6536636/4852241"
I0726 19:29:46.079900       1 sync.go:285] Certificate issued successfully
I0726 19:29:46.079951       1 helpers.go:188] Found status change for Certificate "cis-tls-secret" condition
"Ready": "False" -> "True"; setting lastTransitionTime to 2018-07-26 19:29:46.079945046 +0000 UTC m=+273630.
579266468
I0726 19:29:46.080259       1 sync.go:191] Certificate cis-dev/cis-tls-secret scheduled for renewal in 1438
hours
I0726 19:29:46.091762       1 controller.go:191] certificates controller: Finished processing work item "cis-dev
/cis-tls-secret"
```

You should also see that a new secret has been created for your certificate:

```
$ kubectl get certificate,secret -n cis-dev
NAME                                          AGE
certificate.certmanager.k8s.io/cis-tls-secret   5m

NAME                         TYPE                                  DATA      AGE
secret/cis-tls-secret        kubernetes.io/tls                     2         2m
secret/default-token-p2sxz   kubernetes.io/service-account-token   3         5d
```

What should your users see? (Browser)

Navigating to your ingress rule, you should see that you now have a LetsEncrypt TLS certificate.

I would recommend incognito, as TLS certs appear to be aggressively cached in Chrome.

To examine the certificate in Chrome, simply click the "Secure" / "Not Secure" badge to the immediate left of the address bar.

This will expand a dropdown that offers a `Certificate` option. Clicking this option will pop-out a window allowing you to examine the details of the certificate.

This will show you who issued the certificate, and can also shed light on who is the owner of a particular domain.

Certificate Differences: LetsEncrypt staging vs production

If you are using the LetsEncrypt staging server, the issued certificate will still appear `Not Secure` at first glance. Examining it, you should see that it was `Issued by: Fake LE Intermediate X1`

Real certificates (issued via the production server) are truly `Secure`, and will show as `Issued by: Let's Encrypt Authority X3`

Remember: DO NOT use LetsEncrypt production servers for testing - you will have a bad time.

For more information on the differences between staging and production on LetsEncrypt, see https://letsencrypt.org/docs/rate-limits/

Migrating from Staging to Production

If you've made it this far using staging, then you are probably ready to start issuing real certificates - simply change `https://acme-staging-v02.api.letsencrypt.org/directory` above to `https://acme-v02.api.letsencrypt.org/directory`

You may need to manually delete your old secret to get cert-manager to create a new one with the real TLS certificate and, once it has been recreated, touch your ingress rule.

Deleting secret out from under the Ingress Controller puts it into a bad state, but this is corrected when the Ingres Controller detects a change to the rule and reloads it.