K8s CERT-MANAGER

WHAT IS A SELF SIGNED CERTIFICATE

A **self-signed SSL certificate** is a certificate that is signed by the person who created it rather than a trusted certificate authority. Self-signed certificates can have the same level of encryption as the trusted CA-signed SSL certificate.

Web browsers do not recognize the self-signed certificates as valid. When using a self-signed certificate, the web browser shows a warning to the visitor that the web site certificate cannot be verified.

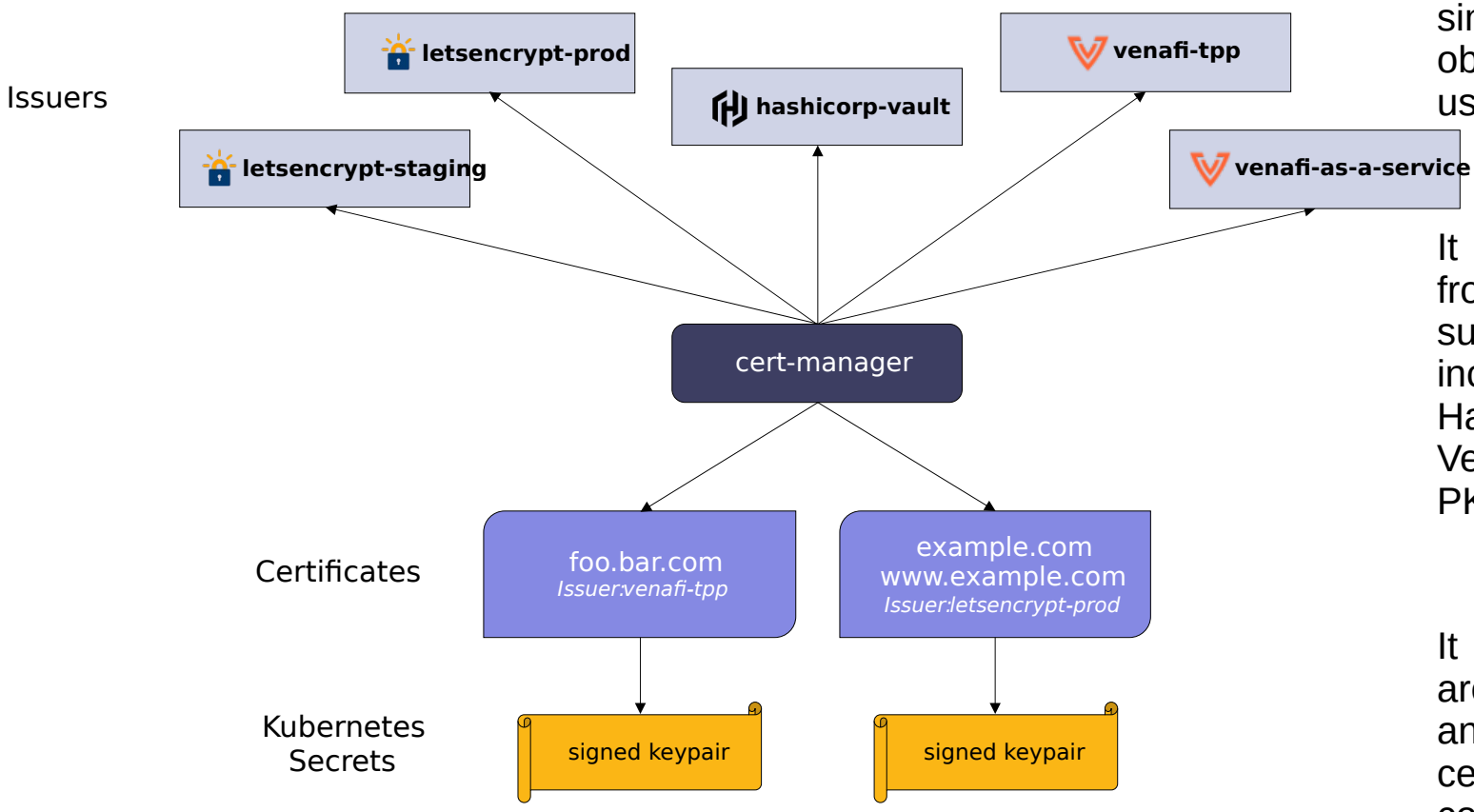
→ **that's why we need a certificate authority**: In cryptography, a certificate authority or certification authority (CA) is an entity that stores, signs, and issues digital certificates.

Typically, the self-signed certificates are used for testing purposes or internal usage. You should not use a self-signed certificate in production systems that are exposed to the Internet.

Learn how to create your self signed certificate:

https://linuxize.com/post/creating-a-self-signed-ssl-certificate/

WHAT IS CERT MANAGER



cert-manager adds **certificates** and **certificate issuers** as resource types in Kubernetes clusters, and simplifies the process of obtaining, renewing and using those certificates.

It can issue certificates from a variety of supported sources, including Let's Encrypt, HashiCorp Vault, and Venafi as well as private PKI.

It will ensure certificates are valid and up to date, and attempt to renew certificates at a configured time before expiry.

ISSUERS == CERT AUTHORITY

The first thing you'll need to configure after you've installed cert-manager is an Issuer or a ClusterIssuer. These are resources that represent **certificate authorities** (CAs) able to sign certificates in response to certificate signing requests. A very common issuer used for self signed certificates is **Let’s Encrypt**.

Rank	Issuer	Usage	Market Share
1	IdenTrust	38.5%	43.6%
2	DigiCert Group	13.1%	14.5%
3	Sectigo (Comodo Cybersecurity)	12.1%	13.4%
4	GlobalSign	16.1%	16.7%
5	Let's Encrypt	5.8%	6.4%
6	GoDaddy Group	4.8%	5.3%

```
$ k get clusterissuer
NAME              READY   AGE
letsencrypt-prod  True    58d
```

ACME

ACME stands for Automatic Certificate Management Environment

Public Key Infrastructure using X.509 (PKIX) certificates are used for a number of purposes, the most significant of which is the authentication of domain names. Thus, **certification authorities** (CAs) in the Web PKI are trusted to verify that an applicant for a certificate legitimately represents the domain name(s) in the certificate. As of this writing, this verification is done through a collection of ad hoc mechanisms. ACME is a protocol that a CA and an applicant can use to automate the process of verification and certificate issuance. The protocol also provides facilities for other certificate management functions, such as certificate revocation.

Identifier Validation Challenges

Challenges provide the server with assurance that an account holder is also the entity that controls an identifier.

HTTP-01 challenge

This is the most common challenge type today. Let's Encrypt gives a token to your ACME client, and your ACME client puts a file on your web server at http://<YOUR_DOMAIN>/.well-known/acme-challenge/<TOKEN>. That file contains the token, plus a thumbprint of your account key. Once your ACME client tells Let's Encrypt that the file is ready, Let's Encrypt tries retrieving it (potentially multiple times from multiple vantage points). If our validation checks get the right responses from your web server, the validation is considered successful and you can go on to issue your certificate. If the validation checks fail, you'll have to try again with a new certificate.

DNS-01 challenge

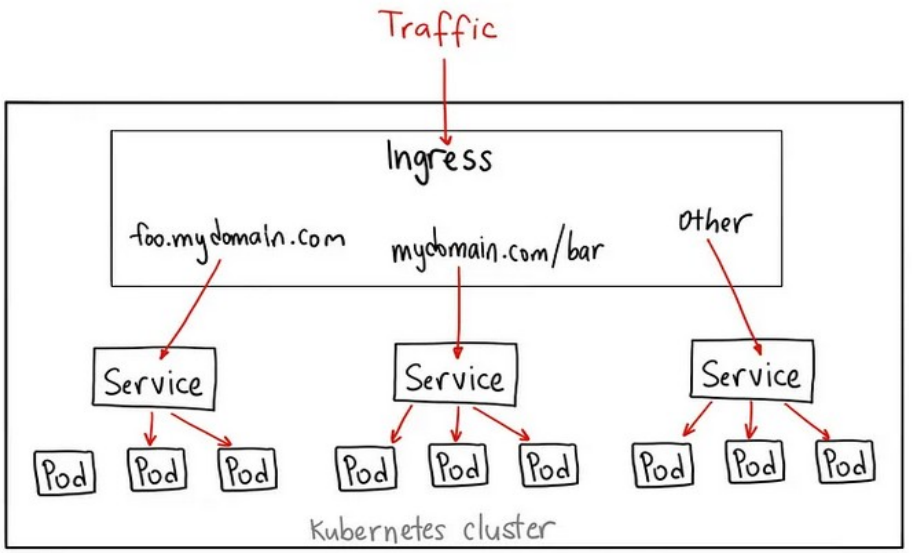
This challenge asks you to prove that you control the DNS for your domain name by putting a specific value in a TXT record under that domain name. It is harder to configure than HTTP-01, but can work in scenarios that HTTP-01 can't. It also allows you to issue wildcard certificates. After Let's Encrypt gives your ACME client a token, your client will create a TXT record derived from that token and your account key, and put that record at _acme-challenge.<YOUR_DOMAIN>. Then Let's Encrypt will query the DNS system for that record. If it finds a match, you can proceed to issue a certificate!

INGRESS

Unlike all the above examples, Ingress is actually NOT a type of service. Instead, it sits in front of multiple services and act as a “smart router” or entrypoint into your cluster.

You can do a lot of different things with an Ingress, and there are many types of Ingress controllers that have different capabilities.

Nginx ingress controller will spin up a Load Balancer for you. This will let you do both path based and subdomain based routing to backend services. For example, you can send everything on foo.yourdomain.com to the foo service, and everything under the yourdomain.com/bar/ path to the bar service.



CERT-MANAGER K8S OBJECTS

Orders resources are used by the ACME issuer to manage the lifecycle of an ACME 'order' for a signed TLS certificate. More details on ACME orders and domain validation can be found on the Let's Encrypt website here. An order represents a single certificate request which will be created automatically once a new CertificateRequest resource referencing an ACME issuer has been created. CertificateRequest resources are created automatically by cert-manager once a Certificate resource is created, has its specification changed, or needs renewal.

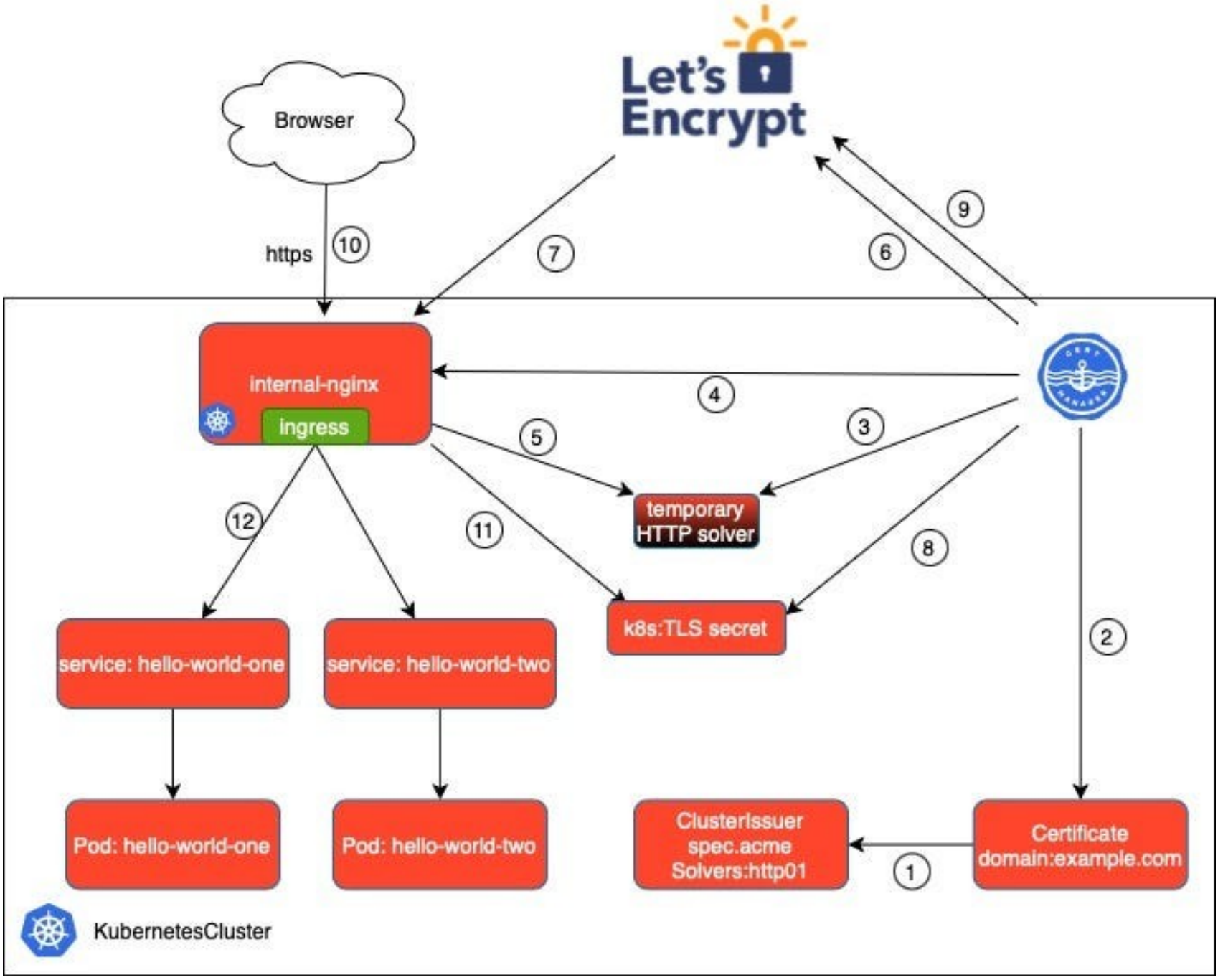
Challenge resources are used by the ACME issuer to manage the lifecycle of an ACME 'challenge' that must be completed in order to complete an 'authorization' for a single DNS name/identifier.

When an Order resource is created, the order controller will create Challenge resources for each DNS name that is being authorized with the ACME server.

As an end-user, you will never need to manually create a Challenge resource. Once created, a Challenge cannot be changed. Instead, a new Challenge resource must be created.

INGRESS

```
$ k get certificaterequest -n asinatra
NAME                    APPROVED   ISSUER  REQUESTOR                                        AGE
webapp-general-tls-7nf2c   True          letsencrypt-prod   system:serviceaccount:cert-manager:cert-manager   26d

aureliano@aureliano-N141CU ~/git/terraform (main)$ k get challenge -A
NAMESPACE   NAME                                                  STATE     DOMAIN                    AGE
default     testnginx-general-tls-mnrwr-955462142-2377448811      pending   jeremy.ynov.ddns.net      12d
lbattais    webapp-general-tls-6b9d6-3871570138-3572040395        pending   lucas-webapp.ynov.ddns.net   26d
mjust       mjust-webapp-general-tls-zwdpn-1829819536-1932917636  pending   mjust.webapp.ynov.ddns.net   26d
sidie       webapp-general-tls-ss7qh-1331647779-3726548383        pending   sidi-webapp.ynov.ddns.net    5d18h

$ kubectl get order -n asinatra
NAME                                 STATE   AGE
webapp-general-tls-7nf2c-3037855297   valid   26d

$ k get certificate -n asinatra
NAME                READY   SECRET              AGE
webapp-general-tls   True    webapp-general-tls   26d

$ k get secret -n asinatra -o yaml

$ k get ingress -n asinatra
NAME                CLASS   HOSTS                      ADDRESS         PORTS     AGE
example-webapp-test   nginx   webapp.ynovsophia.ddns.net   159.223.241.14   80, 443   26d
```


Let's Encrypt
Browser
https
10
7
6
9
internal-nginx
ingress
4
5
3
temporary HTTP solver
12
11
8
k8s:TLS secret
service: hello-world-one
service: hello-world-two
2
Pod: hello-world-one
Pod: hello-world-two
ClusterIssuer
spec.acme
Solvers:http01
1
Certificate
domain:example.com
KubernetesCluster