Session 3:
Ingress Controllers and ACME

What’s an ingress?

- Is an object that manages external access to a service in the cluster, typically through HTTP
- Even though this API is built in on the apiserver by default, it is one of those resources that does **nothing** with the default setup.
- For an ingress to actually do something, you need to set up an **ingress controller**

What’s an ingress?

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        backend:
          serviceName: test
          servicePort: 80
```

What’s an ingress?

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: s1
          servicePort: 80
  - host: bar.foo.com
    http:
      paths:
      - backend:
          serviceName: s2
          servicePort: 80
```

Ingress Controller

- You can look at an ingress as rules to expose something to the outer world
- The **ingress controller** actually does the work
  - There are two heavily supported ingress controllers in GKE:
    - Nginx (An nginx that auto-configures itself based on ingress resources)
    - GCE (A L7 load balancer that resides outside the cluster)
- Incoming requests to the IC are routed to the appropriate service based on what the ingress resources say.

Default Backend

- When an incoming request is not routed to any service based on ingress resources, it is routed to the **default backend**
- The default backend is a service that will reply to any route a **404 response**.
- It can reply any HTML it desires, as long as the response is a 404.
- You **must** deploy a default backend when you deploy an ingress controller.

GCE Ingress Controller

- https://github.com/kubernetes/ingress-gce
- Enabled by default, but can be disabled when creating a cluster
- If you are going to use another ingress controller in GKE, you must either disable this one or use annotations on your ingress.
- You will be **billed** for each L7 load balancer that is created through this. Be sure to check pricing and how many LB will be created (I believe is 1 per ingress resource)

Nginx Ingress Controller

- https://github.com/kubernetes/ingress-nginx
- Can be quickly installed through Helm:
  - https://github.com/kubernetes/charts/tree/master/stable/nginx-ingress
- If you have multiple ingress controllers installed, you must annotate your ingresses so they’re picked up by this one
  - kubernetes.io/ingress.class: "nginx"
- Some additional configuration can be added through annotations
- Deeper configuration can be done through ConfigMap

TLS

- An ingress can specify if it should allow TLS connections:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  tls:
  - secretName: tls-secret
    hosts:
    - something.org
  rules:
  - http:
      paths:
      - path: /testpath
        backend:
          serviceName: test
          servicePort: 80
```

TLS

- The secret would have something like this:

```
apiVersion: v1
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
kind: Secret
metadata:
  name: tls-secret
  namespace: default
type: Opaque
```

TLS

- Using this method you can add any existing TLS certificate, which is a good path when migrating existing applications.
- However, we can make this even simpler.
- The ingress controller cares only that the secret with the proper certificate data exists. It cares not what put the secret there.
- Hence, we have a way of automatically creating certificates based on our ingress data

ACME

- The ACME protocol (Automated Certificate Management Environment) was created to automate the whole process of certificate request, verification and submission
- There’s ACMEv1 and ACMEv2. v2 is not backwards compatible with v1
  - v2 supports wildcard domains which v1 did not.
  - Most ACME clients are still supporting only v1 (at May 2018)

ACME Workflow

- Full spec: https://ietf-wg-acme.github.io/acme/draft-ietf-acme-acme.html
- Client requests a certificate from a server
- Server states a list of challenges that the client must surpass in order to verify the identity. These can be:
  - HTTP requests
  - DNS records (through TXT records)
- The client does what it needs to make the challenges succeed
  - In an HTTP challenge this would involve providing a response string with a key at a particular url.
- The client notifies the server that the challenges can be reviewed
- The server verifies the challenges
- If the challenges succeed, a new certificate is minted and transferred to the client
- The client install the certificate, and periodically renews it by making renewal requests to the server.

ACME Server

- If we want our certificates to be recognized, the ACME server we interact with must be well-respected and known
- Otherwise we will have valid certificates that nobody trusts
- Currently, the most popular (and free) ACME server is Let's Encrypt

ACME Client

- We need a client that will:
  - Read our ingress resources
  - Request certificates based on that information
  - Automatically add non-HTTPS ingresses to handle challenges
  - Store resulting certificates on Kubernetes secrets

ACME Client

- Clients:
  - Kube-lego (deprecated)
  - Cert-Manager
    - We will focus on this one

ACME Client

- We state that we want an ingress to have auto-generated ACME certificates with an annotation:
  - kubernetes.io/tls-acme: “true”
  - This annotation was introduced by kube-lego, and Cert Manager supports it as well (with extra configuration on ingress-shim)

Cert Manager

- https://github.com/jetstack/cert-manager
- Flexible add-on, can be configured with multiple providers (i.e. Issuers)
- Can be installed through Helm:
  - https://github.com/kubernetes/charts/tree/master/stable/cert-manager
- You will need to install an ingress shim in order to load data from ingresses. Installing though Helm installs this as well.

Cert Manager

- After you install it you must add an issuer to your cluster:

```
apiVersion: certmanager.k8s.io/v1alpha1
kind: ClusterIssuer
metadata:
  name: test-issuer
spec:
  acme:
    email: info@mahisoft.com
    server: https://acme-v01.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: issuerKey
    http01: {}
```

Cert Manager

- You can specify what Issuer to use on your ingress through annotations:
  - certmanager.k8s.io/cluster-issuer: "test-issuer"
- You can also provide a default issuer by passing arguments to the ingress-shim pod (or via Helm)
  - --default-issuer-name=test-issuer,--default-issuer-kind=ClusterIssuer

Questions?

Assignment

- Create a GKE cluster (you can use your script from last week)
- Install an Nginx ingress controller through helm. Make sure to set the service type to **LoadBalancer**
- Install Cert Manager and create a Cluster Issuer pointing to let's encrypt **staging** environment. You can install this through helm as well
- Add a deployment (anything that replies through http) and create an ingress for it.
- Send me your load balancer's IP address in an email, as well as the output of kubectl describe ingress <your-ingress>