Kong Gateway Operations on Kubernetes

Intro to Kong Ingress Controller

Course Agenda

1. Kong Gateway Installation on K8s
2. **Intro to Kong Ingress Controller**
3. Securing Kong Gateway
4. Securing Services on Kong
5. OIDC Plugin
6. Kong Vitals
7. Advanced Plugins Review
8. Troubleshooting
9. Test your Knowledge

Learning Objectives

1. Describe what a Kubernetes Ingress Controller does
2. Describe the purpose of each Kong CRD
3. Describe how to create a Consumer via KongConsumer CRD
4. Describe how to create a Gateway Service via a K8s Annotated Service with K8s Ingress
5. Describe how to create a Gateway Route via a K8s Annotated Ingress and how to use the KongIngress CRDS
6. Describe how to implement a KongPlugin CRD to a Gateway Object
7. Deploy a new Kong Ingress Controller to enable a new Workspace

Reset Environment

Before we start this lesson, be sure to reset your environment:

```
$ cd /home/labuser
$ git clone https://github.com/Kong/edu-kgac-202.git
$ source ~/edu-kgac-202/base/reset-lab.sh
```

If the git repo already exists, then `git clone` will give an error - you can disregard this.

Kubernetes Ingress Controllers

What is a Kubernetes Ingress Controller?

Kubernetes Ingress Controllers are not deployed to a k8s cluster by default. The reason is there are many available and each of them work in coordination with the Kubernetes Ingress resource to activate the Ingress rules on their corresponding product.

The first step is to deploy a Kubernetes Ingress Controller to your cluster that fits your use case. In our case, we will be deploying Kong Ingress Controller for Kubernetes (KIC) via our Kong helm chart and its values file. The specific configuration each controller interprets and acts on from the Ingress resource is proprietary to the controller's individual tech. For example, the NGINX Ingress Controller configures Nginx accordingly. The NGINX Ingress Controller cannot be used to configure our Kong Gateway. Only our KIC can do this for us.

The remaining steps involve creating your Kubernetes Ingress yaml and tagging your Kubernetes Ingress Controller. Your Controller will then respond after the Kubernetes API creates the resource and it will configure its targets accordingly.

We will be building our yamls in this lesson to hit home these concepts specifically for KIC.

What is a Kubernetes Ingress Controller?

Kubernetes Ingress Controllers are optional components that activate the Ingress rules on the system

There are many Ingress Controllers available, and the specific configuration for each is proprietary to the controller's individual tech. For example, the NGINX Ingress Controller configures Nginx - it cannot be used to configure our Kong Gateway.

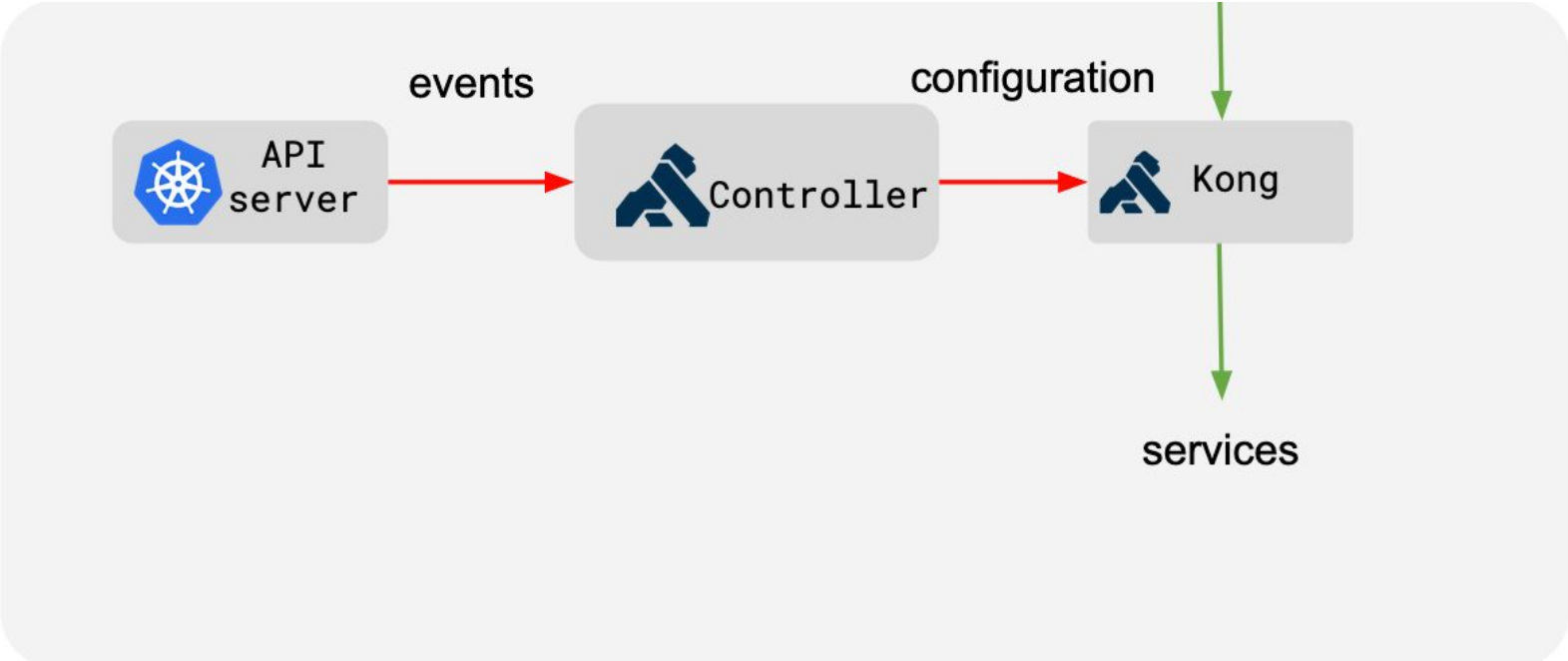
We will deploy Kong Ingress Controller for Kubernetes (KIC) via our Kong helm chart and its values file.

The remaining steps involve creating your Kubernetes Ingress yaml and tagging your Kubernetes Ingress Controller. Your Controller will then configure its targets accordingly.

We will be building our yamls in this lesson to hit home these concepts, specifically for KIC.

KIC Flows

As CRDs are created via the K8s API, the KIC responds by modifying the Kong Gateway configuration.



Kong Custom Resource Definitions (CRDs)

While **deck** can still be used to configure the gateway via the command line, Kong has created and recommends the usage of Custom Resource Definitions (CRDs). The Ingress Controller can configure Kong specific features using several Custom Resource Definitions(CRDs).

Following CRDs enables users to declaratively configure all aspects of Kong:

- KongPlugin: This resource corresponds to the Plugin entity in Kong.
- KongIngress: This resource provides fine-grained control over all aspects of proxy behaviour like routing, load-balancing, and health checking. It serves as an “extension” to the Ingress resources in Kubernetes.
- KongConsumer: This resource maps to the Consumer entity in Kong.
- TCPIngress: This resource can configure TCP-based routing in Kong for non-HTTP services running inside Kubernetes.
- UDPIngress: This resource can configure UDP-based routing in Kong.
- KongClusterPlugin: KongClusterPlugin resource is exactly same as KongPlugin, except that it is a Kubernetes cluster-level resources instead of being a namespaced resource.

KongConsumer CRD

KongConsumer CRD: K8s Secret

A typical KongConsumer deployment involves creating the K8s Secret first and then creating the KongConsumer CRD which references the created secret by name. However, some authentications via KongPlugins might have other designs.

```
apiVersion: v1
kind: Secret
metadata:
  name: jane-apikey
  namespace: httpbin-demo
type: Opaque
data:
  key: SmFuZVBhc3N3b3Jk
  kongCredType: a2V5LWF1dGg=
```

There are different kinds of kongCredTypes to include key-auth, jwt, and acl. Refer to the plugin documentation to determine which value to use. The values in the YAML are base64 encoded.

```
$ echo "SmFuZVBhc3N3b3Jk" | base64 -d
JanePassword

$ echo "a2V5LWF1dGg=" | base64 -d
key-auth
```

KongConsumer CRD: YAML and Annotations

Here is a typical KongConsumer YAML that creates a Consumer in the Kong Gateway and references the previously created K8s Secret for this Consumer. Be sure that the K8s Secret is created in the same namespaces as the KongConsumer so the KIC can find it.

```
apiVersion: configuration.konghq.com/v1
kind: KongConsumer
metadata:
  name: jane
  namespace: httpbin-demo
  annotations:
    kubernetes.io/ingress.class: kong
username: jane
credentials:
  - jane-apikey
```

Current list of annotations for the KongConsumer CRD:

Following annotations are supported on KongConsumer resources:

ANNOTATION NAME	DESCRIPTION
REQUIRED `kubernetes.io/ingress.class`	Restrict the KongConsumers that a controller should satisfy
`konghq.com/plugins`	Run plugins for a specific consumer

Kong Gateway Services, Routes, and Upstreams via KIC

Creating Kong Gateway Services/Routes/Upstreams via KIC

Kong Gateway Services, Routes, and Upstreams are triggered to be created from the Kubernetes Ingress objects. Creating a Kubernetes Service and annotating it with some Kong annotations will not trigger the KIC to create an Service object.

Kong Service objects are created from the KIC interpreting what is in the Kubernetes Ingress object. KIC will review what is in the Kubernetes Ingress yaml and then create the appropriate objects needed to honor the declaration.

We will go over this in the following Ingress section.

Kubernetes Ingress and KongIngress CRD

Creating Ingress with Kubectl for Kong Gateway

KIC uses the Kubernetes Ingress object as the entry point into the creation of Kong Services, Upstreams, and Routes. KIC does not currently have a 1:1 mapping between traditional Kong Gateway objects and CRDs. For example, there is no equivalent KIC CRD for a Service, Route, or Upstream Kong Admin API calls.

Instead, KIC looks at the traditional Kubernetes Ingress object and uses additional CRDs such as KongIngress to further configure the Gateway.

This means that a translation from Kong Admin API calls to KIC needs to occur in order to achieve the equivalency.

Kubernetes Ingress with KongIngress Example

```
apiVersion: v1
kind: Service
metadata:
  name: httpbin-service
  namespace: httpbin-demo
  annotations:
    konghq.com/path: "/headers"
spec:
  type: ExternalName
  externalName: httpbin.default.svc.cluster.local
```

A typical pattern is to first create the Kubernetes Service object with a type of ExternalName pointing to endpoint of the upstream service.

Here we are creating the service in the httpbin-demo namespace with the external endpoint being the previously deployed httpbin application to the default namespace. This simulates an external service to Kong to proxy and could easily be an application endpoint on the internet or somewhere else in your datacenter. Note that no Kong Gateway objects are created yet.

Kubernetes Ingress with KongIngress Example

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: httpbin-ingress
  namespace: httpbin-demo
  annotations:
    konghq.com/strip-path: "true"
spec:
  ingressClassName: kong
  rules:
  - http:
      paths:
      - path: /httpbin
        pathType: Prefix
        backend:
          service:
            name: httpbin-service
            port:
              number: 80
```

Once this Kubernetes Ingress object is created with the ingressClassName being kong, the KIC will take action and create appropriate Service, Route, and Upstream if applicable.

Some things to note about this yaml:

- “path: /httpbin” refers to the Route that will be created in the Gateway
- “ingressClassName: kong” needs to be present in order for KIC to take action
- KIC will examine the K8s service object, httpbin-service, to determine if it will create just a Service in the Gateway or also an Upstream if ExternalName is used

Kubernetes Ingress with KongIngress Example

```
apiVersion: configuration.konghq.com/v1
kind: KongIngress
metadata:
  name: httpbin-route-customize
  namespace: httpbin-demo
route:
  headers:
    X-with-ID:
    - "TRUE"
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: correlation
  namespace: httpbin-demo
  annotations:
    konghq.com/strip-path: "true"
    konghq.com/override: "httpbin-route-customize"
```

Sometimes, the information you can supply to the Kubernetes Ingress object may not be enough to achieve your application's goals. For example, header modification at the route level cannot be added to this object.

To help with this, KIC has a KongIngress CRD that can be created and then referenced by the Kubernetes Ingress object via an annotation.

To the left is an example with the rest of the Kubernetes Ingress stripped for visibility on this slide.

KongPlugin CRD

KongPlugin CRDs

Kong Plugins are declared via KIC by using the KongPlugin CRD. A typical pattern is to first create the KongPlugin CRD via kubectl and then reference that plugin with kubernetes annotations in Kubernetes Service and Ingress objects to apply that plugin.

To the right, we have an example of a KongPlugin getting created with its config. We then see a Kubernetes Service object with annotations to use that plugin. The appropriate Service/Route/Upstream are then created by annotations and configuration of a Kubernetes Ingress object like we have already seen.

```
apiVersion: configuration.konghq.com/v1
kind: KongPlugin
metadata:
  name: httpbin-acl
  namespace: httpbin-demo
plugin: acl
config:
  allow:
  - admins
---
apiVersion: v1
kind: Service
metadata:
  name: oidc-service
  namespace: httpbin-demo
  annotations:
    konghq.com/path: "/anything"
    konghq.com/plugins: "httpbin-oidc,httpbin-acl"
spec:
  type: ExternalName
  externalName: httpbin.default.svc.cluster.local
```

Summary

In this lesson we:

- Described what a Kubernetes Ingress Controller does
- Described the purpose of each Kong CRD
- Described how to create a Consumer via KongConsumer CRD
- Described how to create a Gateway Service via a K8s Annotated Service with K8s Ingress
- Described how to create a Gateway Route via a K8s Annotated Ingress and how to use the KongIngress CRDS
- Described how to implement a KongPlugin CRD to a Gateway Object

KICs and Gateway Workspaces

Relationship between KIC and Gateway Workspaces

KIC uses the Kubernetes Ingress object like other Ingress Controllers as the entry point into the creation of Kong Services, Upstreams, and Routes. KIC does not currently have a 1:1 mapping between traditional Kong Gateway APIs/Objects and CRDs. For example, there is no equivalent KIC CRD for a Service, Route, or Upstream Kong Admin API calls.

Instead, KIC looks at the traditional Kubernetes Ingress object and uses additional CRDs such as KongIngress to further configure the Gateway.

This means that a translation from Kong Admin API calls to KIC needs to occur in order to achieve the equivalency.

KICs and Kong Gateway Workspaces have a 1:1 mapping. Currently, if you are using KIC, it is assumed that you will use K8s RBAC instead of Gateway RBAC. There's currently no way for a since KIC to manage multiple Gateway Workspaces.

Single KIC Helm Values Example

```
deployment:
  kong:
    enabled: false
ingressController:
  ingressClass: "kong-workspace-a"
  enabled: true
  installCRDs: false
  image:
    repository: kong/kubernetes-ingress-controller
    tag: "2.7"
  env:
    kong_workspace: WorkspaceA
    kong_admin_url: "http://kong-kong-admin.kong.svc.cluster.local:8001"
    publish_service: kong-dp/kong-dp-kong-proxy
    kong_admin_token:
      valueFrom:
        secretKeyRef:
          name: kong-enterprise-superuser-password
          key: password
```

We are not interested in deploying the Gateway.

The ingressClass is what this KIC will be registered as inside your Kubernetes cluster and it is what you will use in your manifests.

Kong_workspace is the name of the Workspace inside the Gateway that this KIC will manage.

The password you used for the Gateway you already deployed.

Helm Install and Uninstall Example

```
$ # Add kong helm repo and update
$ helm repo add kong https://charts.konghq.com
$ helm repo update

$ # Install new KIC using:
$ # - Kong's Helm Chart version 2.12.0
$ # - a local values file called kic-helm-values.yaml
$ # - naming the deployment new-kic-deployment and placing in the kong namespace
$ helm install --version 2.12.0 -f kic-helm-values.yaml new-kic-deployment
kong/kong -n kong

$ # Uninstall the KIC we just deployed
$ helm uninstall new-kic-deployment -n kong
```

Questions?

What's next?

In the next section we will show you how to secure your Kong Gateway.

Thank You