**Ingress Controller**

Ingress controller is a necessary Kubernetes feature that plays a vital role in the functioning of Ingress resources. The Ingress resources deployed in the cluster are controlled by the Ingress controller. As mentioned in [Kubernetes Ingress controller page](https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/): "Unlike other types of controllers which run as part of the kube-controller-manager binary, Ingress controllers are not started automatically when a cluster is created." It must be deployed into the cluster manually and configured according to the prescribed requirements.

**NGINX Ingress Controller for Kubernetes**

According to the [NGINX Ingress Controller page](https://www.nginx.com/products/nginx/kubernetes-ingress-controller), "The NGINX Ingress Controller for Kubernetes provides enterprise-grade delivery of services for Kubernetes applications, with benefits for users of both open source NGINX and Kubernetes." We are using this NGINX Ingress Controller.

**Deploying a NGINX Ingress Controller**

 kubectl apply -f

apiVersion: v1

kind: Namespace

metadata:

name: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

---

kind: ConfigMap

apiVersion: v1

metadata:

name: nginx-configuration

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

---

kind: ConfigMap

apiVersion: v1

metadata:

name: tcp-services

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

---

kind: ConfigMap

apiVersion: v1

metadata:

name: udp-services

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

---

apiVersion: v1

kind: ServiceAccount

metadata:

name: nginx-ingress-serviceaccount

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRole

metadata:

name: nginx-ingress-clusterrole

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

rules:

- apiGroups:

- ""

resources:

- configmaps

- endpoints

- nodes

- pods

- secrets

verbs:

- list

- watch

- apiGroups:

- ""

resources:

- nodes

verbs:

- get

- apiGroups:

- ""

resources:

- services

verbs:

- get

- list

- watch

- apiGroups:

- "extensions"

resources:

- ingresses

verbs:

- get

- list

- watch

- apiGroups:

- ""

resources:

- events

verbs:

- create

- patch

- apiGroups:

- "extensions"

resources:

- ingresses/status

verbs:

- update

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: Role

metadata:

name: nginx-ingress-role

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

rules:

- apiGroups:

- ""

resources:

- configmaps

- pods

- secrets

- namespaces

verbs:

- get

- apiGroups:

- ""

resources:

- configmaps

resourceNames:

# Defaults to "<election-id>-<ingress-class>"

# Here: "<ingress-controller-leader>-<nginx>"

# This has to be adapted if you change either parameter

# when launching the nginx-ingress-controller.

- "ingress-controller-leader-nginx"

verbs:

- get

- update

- apiGroups:

- ""

resources:

- configmaps

verbs:

- create

- apiGroups:

- ""

resources:

- endpoints

verbs:

- get

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: RoleBinding

metadata:

name: nginx-ingress-role-nisa-binding

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: nginx-ingress-role

subjects:

- kind: ServiceAccount

name: nginx-ingress-serviceaccount

namespace: ingress-nginx

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: nginx-ingress-clusterrole-nisa-binding

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: nginx-ingress-clusterrole

subjects:

- kind: ServiceAccount

name: nginx-ingress-serviceaccount

namespace: ingress-nginx

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-ingress-controller

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

spec:

replicas: 1

selector:

matchLabels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

template:

metadata:

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

annotations:

prometheus.io/port: "10254"

prometheus.io/scrape: "true"

spec:

serviceAccountName: nginx-ingress-serviceaccount

containers:

- name: nginx-ingress-controller

image: quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.24.1

args:

- /nginx-ingress-controller

- --configmap=$(POD\_NAMESPACE)/nginx-configuration

- --tcp-services-configmap=$(POD\_NAMESPACE)/tcp-services

- --udp-services-configmap=$(POD\_NAMESPACE)/udp-services

- --publish-service=$(POD\_NAMESPACE)/ingress-nginx

- --annotations-prefix=nginx.ingress.kubernetes.io

securityContext:

allowPrivilegeEscalation: true

capabilities:

drop:

- ALL

add:

- NET\_BIND\_SERVICE

# www-data -> 33

runAsUser: 33

env:

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

ports:

- name: http

containerPort: 80

- name: https

containerPort: 443

livenessProbe:

failureThreshold: 3

httpGet:

path: /healthz

port: 10254

scheme: HTTP

initialDelaySeconds: 10

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 10

readinessProbe:

failureThreshold: 3

httpGet:

path: /healthz

port: 10254

scheme: HTTP

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 10

---

This will deploy the Ingress Controller deployment and other associated resources in the **ingress-nginx**namespace.

 kubectl apply -f  :

kind: Service

apiVersion: v1

metadata:

name: ingress-nginx

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

spec:

externalTrafficPolicy: Local

type: LoadBalancer

selector:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

ports:

- name: http

port: 80

targetPort: http

- name: https

port: 443

targetPort: https

---

This will deploy the Ingress Controller LoadBalancer type Kubernetes service in the **ingress-nginx**namespace.

**Ingresses**

Ingress is a Kubernetes object that allows access to your Kubernetes services from outside the Kubernetes cluster. You can configure access by creating a set of rules that defines which inbound internet traffic must reach which Kubernetes service in the cluster.

For Ingress resources to work within a cluster you must deploy an Ingress Controller prior to deploying an Ingress. The Ingress controller allows the Internet traffic to enter in the Kubernetes cluster. Once it is set up correctly we can create Ingress resources in the cluster and route internet traffic to the services. Note that Ingress resources are namespace-specific.

Below is a sample code for an Ingress resource:

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: service1-ingress

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

kubernetes.io/ingress.class: nginx

nginx.ingress.kubernetes.io/enable-cors: "true"

nginx.ingress.kubernetes.io/cors-allow-origin: "\*"

nginx.ingress.kubernetes.io/cors-allow-credentials: "true"

nginx.ingress.kubernetes.io/cors-allow-methods: "\*"

spec:

rules:

- host: aks.example.com

http:

paths:

- backend:

serviceName: service1

servicePort: 8089

**Common Issues Faced when Using Nginx Ingress Controller with Default Settings**

Headers and Body play vital roles in performing operations to an API service running in AKS. Ingresses allow the services running in AKS to talk to the outside world seamlessly. However one might face blockers when appropriate annotations and custom headers are not allowed in the configurations of the Nginx Ingress controller while the traffic hits the API service. There are a couple of ways to manipulate the configurations of the Ingress controller. One is by adding config maps that are connected to the Ingress controller deployment which affects the entire cluster, and the other is by adding necessary annotations to the Ingress resource, this way of manipulating configurations can affect only the namespace in which the Ingress resource is deployed.

Following are some of the common issues which we faced.

**Rewrite Target**

In the Ingress resource one must add **nginx.ingress.kubernetes.io/rewrite-target**annotations to redirect the internet traffic to a specific service of the application deployed in the cluster. The path key in the below YAML snippet has the value with which the URL will get rewritten. For instance, if the URL is **http://aks.example.com/service1,** then the URL will get rewritten as **http://aks.example.com** and then accessed inside the pod.

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: service1-ingress

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- host: aks.example.com

http:

paths:

- path: /service1

backend:

serviceName: service1

servicePort: 80

**Blocked Custom-Headers**

Ingresses, by default block the custom header which are used by a API service running in the AKS. These custom headers must be allowed through ingress controller so that it reaches the API service from the outside world. To fix this issue we make use of annotations and config maps that adhere to the Nginx ingress controller.

We create this config map following is YAML code for the config map.

apiVersion: v1

data:

X-Different-Name: "true"

X-Request-Start: t=${msec}

X-Using-Nginx-Controller: "true"

kind: ConfigMap

metadata:

name: custom-headers

namespace: ingress-nginx

We also need to add the annotation to the Ingress of that service as shown below.

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

annotations:

kubernetes.io/ingress.class: nginx

nginx.ingress.kubernetes.io/cors-allow-credentials: "true"

nginx.ingress.kubernetes.io/cors-allow-headers: Ocp-Apim-Subscription-Key,Ocp-Apim-Trace, Authorization, Content-Type

**Ocp-Apim-Subscription-Key**and **Ocp-Apim-Trace are** custom headers which we need to use to make use of Azure API management resource.

**Underscores in Headers Are Not Allowed**

Application developers have a habit of creating API services which accept headers that may have underscores in them, by default, the Nginx ingress controllers do not accept custom headers and headers containing underscores. To fix this issue, we edit the config maps that adheres to Nginx ingress controller.

When you create an Ingress controller it also creates a default config map know as **nginx-configuration**we edit this config map and add **data** to it**.**Following is YAML code for the config map.

apiVersion: v1

data:

enable-underscores-in-headers: "true"

proxy-set-headers: ingress-nginx/custom-headers

kind: ConfigMap

metadata:

labels:

app: ingress-nginx

name: nginx-configuration

namespace: ingress-nginx

**Limited Proxy Body Size**

In some cases, the body of an API service requires data as a payload which is of larger size than the limit. To increase the size of data supplied as the payload of the API we need to increase the PROXY BODY SIZE   allowed by the Ingresses.

To fix this issue we need to use the **annotations** in the ingresses. We use **nginx.ingress.kubernetes.io/proxy-body-size: 10m**. annotation in the Ingress of that particular service.

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

annotations:

nginx.ingress.kubernetes.io/proxy-body-size: 10m

**Enable Cross-Origin Resource Sharing (CORS) in Ingress**

According to Wikipedia, "Cross-origin resource sharing [(CORS)](https://en.wikipedia.org/wiki/Cross-origin_resource_sharing) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served".

To enable CORS we need to add the three main annotations in the ingresses as shown below.

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

annotations:

nginx.ingress.kubernetes.io/cors-allow-methods: '\*'

nginx.ingress.kubernetes.io/cors-allow-origin: '\*'

nginx.ingress.kubernetes.io/enable-cors: "true"

Similarly, based on application needs there are many annotations which you can use to enable the features in the Ingress.