**Setup Nginx Ingress Controller on Kubernetes using Helm, Metallb and GoDaddy**

Kubernetes Ingress is an API object that manages external access to the services in a cluster, typically HTTP/HTTPS. Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

Ingress may provide load balancing, SSL termination and name-based (path-based) virtual hosting. With Ingress, you can easily set up rules for routing traffic without creating a bunch of Load Balancers or exposing each service on the node within you Kubernetes cluster.

Within a Kubernetes cluster an Ingress resource allows you to share a single public IP address and route your application via URLs or URI, commonly known as http/https routing. If your Kubernetes cluster has Ingress Controller resource deployed and once it’s picked, you will be able to specify the URLs or URI that you need to route, and the controller will look after it for you.

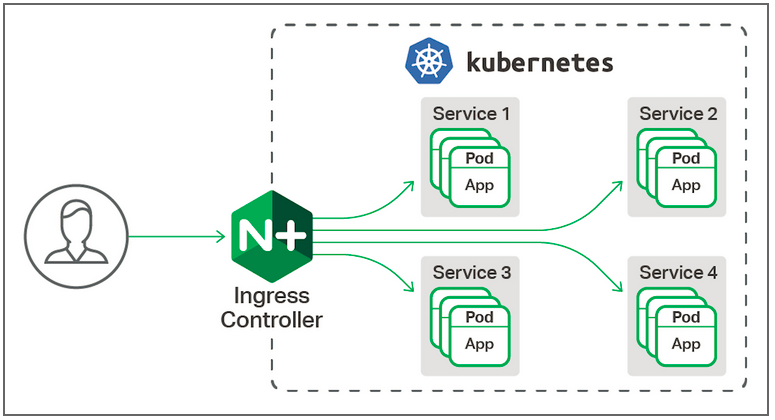
**NOTE**: An Ingress does not expose arbitrary ports or protocols. Exposing services other than HTTP and HTTPS to the internet typically uses a service of type Service.Type=NodePort or Service.Type=LoadBalancer

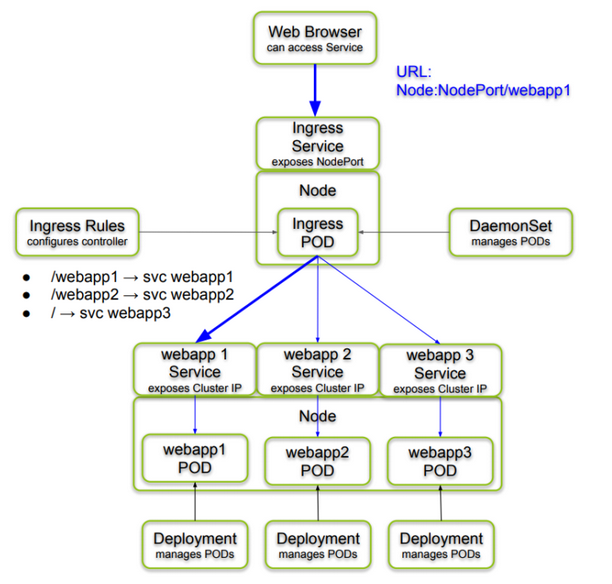
**Why we need Ingress resource?**

When we can simple expose a service and set the service type as LoadBalancer why do we need ingress in Kubernetes?

The exposed service will then talk to cloud provider of your choice and spin up the Load Balancer for you which will have a Public IP and that’s all. End users will be able to access your application using that public IP or with associated DNS name.

Now for a minute just imagine that you are running 100’s of services which are exposed to the public on using one of the public cloud providers. Each service would spin up its own load balancer and public IP address. It will be soon become a headache for you to handle them and eventually become unmanageable. Keep one thing in mind it will shoot up your BILL like anything from your cloud provider for all the extra load balancers and Public IP addresses.





Prerequisites:

* A 3 node Kubernetes cluster up and running
* Helm “The Kubernetes package manager” version 3 installed
* An ON-PERM (Bare Metal) load balancer solution up and running (I will explain setting up the same in a separate section below)

My Setup

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Node Name** | **Operating System** | **IP Address** | **kubectl client/server version** | **Helm version** | **Docker version** |
| k8smaster.local | Centos 7.7 | 192.168.185.71 | v1.20.0 | v3.4.2 | 20.10.0 |
| k8sworker1.local | Centos 7.7 | 192.168.185.72 | v1.20.0 | NA | 20.10.0 |
| k8sworker2.local | Centos 7.7 | 192.168.185.73 | v1.20.0 | NA | 20.10.0 |

### **Installing an in-house Bare metal Load Balancer solution "MetalLB"**

MetalLB is a load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols.

If you’re using kube-proxy in IPVS mode, since Kubernetes v1.14.2 you have to enable strict ARP mode.

**Note**, you don’t need this if you’re using kube-router as service-proxy because it is enabling strict arp by default.

You can achieve this by editing kube-proxy config in current cluster and set:

kubectl edit configmap -n kube-system kube-proxy

apiVersion: kubeproxy.config.k8s.io/v1alpha1

kind: KubeProxyConfiguration

mode: "ipvs"

ipvs:

strictARP: true

If you are trying to automate this change, these shell snippets may help you:

# see what changes would be made, returns nonzero returncode if different

kubectl get configmap kube-proxy -n kube-system -o yaml | \

sed -e "s/strictARP: false/strictARP: true/" | \

kubectl diff -f - -n kube-system

# actually apply the changes, returns nonzero returncode on errors only

kubectl get configmap kube-proxy -n kube-system -o yaml | \

sed -e "s/strictARP: false/strictARP: true/" | \

kubectl apply -f - -n kube-system

**MetalLB Installation**

kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.5/manifests/namespace.yaml

kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.5/manifests/metallb.yaml

kubectl create secret generic -n metallb-system memberlist --from-literal=secretkey="$(openssl rand -base64 128)"

**Define and deploy a configMap**

**Since I am using the CIDR for internal weave networking for kubernetes cluster 10.10.120.0/24. I have used a range of IP’s reserved for the Load Balancers.**

cat > metalLb-configmap.yml << EOF

apiVersion: v1

kind: ConfigMap

metadata:

namespace: metallb-system

name: config

data:

config: |

address-pools:

- name: default

protocol: layer2

addresses:

- 192.168.185.75-192.168.185.85

EOF

**kubectl apply -f metalLb-configmap.yml**

**Test the Load balancer functionality by creating a Deployment and exposing the service**

kubectl create deployment nginx --image=nginx

kubectl expose deployment nginx --port 80 --type LoadBalancer

kubectl get svc

It’s working as expected as our nginx service is getting LoadBalancer IP assigned 10.10.120.190.

### **Deploy the Nginx Ingress controller**

To deploy the NGINX Ingress controller using helm, run the following command:

**helm install nginx-ingress stable/nginx-ingress**

Verify the installation

kubectl get all -- It’s up and running as we see.

**Deploy a sample web app**

****

**kubectl apply -f ingress-demo-deploy-v1.yaml**

**kubectl get deployment**

Expose the Deployment: **kubectl expose deployment ingress-demo-deploy-v1 --port 80**

**kubectl get svc**

### **Create an Ingress resource**

Time to Create an Ingress resource using following manifest file that sends traffic to your Service via **demo.ingress.example.com**

****

**kubectl apply -f ingress-resource-demo.yaml**

Verify the Ingress resource status

**kubectl describe ingress ingress-resource-demo**

**Add the following line to the bottom of the /etc/hosts file or to Windows hosts file C:\Windows\System32\Drivers\etc\hosts and in k8s nodes**

182.18.182.90 demo.ingress.khaleel.com

Try accessing it through your web browser --- **http://demo.ingress.khaleel.com**

**Deploy Second Sample web app**

****

**kubectl apply -f ingress-demo-deploy-v2.yaml**

**kubectl get deployment**

Expose the Deployment: **kubectl expose deployment ingress-demo-deploy-v2 --port 80**

**kubectl get svc**

## **Edit Ingress resource**

**Now since we have created a new Deployment we need to edit our current Ingress Resource. Add following part to your ingress resource file.**

- path: /v2

backend:

serviceName: ingress-demo-deploy-v2

servicePort: 80

kubectl apply -f ingress-resource-demo.yaml

**Verify the Ingress deployment status -- kubectl describe ingresses.networking.k8s.io**

**Test Your Ingress**

**Try accessing the version v1 of our web application using following URL.**

[**http://demo.ingress.khaleel.com/**](http://demo.ingress.khaleel.com/)

**Try accessing the version v2 of our web application using following URL.**

[**http://demo.ingress.khaleel.com/v2**](http://demo.ingress.khaleel.com/v2)

## **Securing the Ingress Using Cert-Manager**

Install the Cert Manager

kubectl create namespace cert-manager

helm repo add jetstack https://charts.jetstack.io

helm repo update

kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.1.0/cert-manager.crds.yaml

helm install cert-manager \

--namespace cert-manager \

--version v1.1.0 \

--set "podDnsPolicy"="None" \

--set "podDnsConfig.nameservers[0]"="8.8.8.8" \

--set extraArgs='{--dns01-recursive-nameservers-only}' \

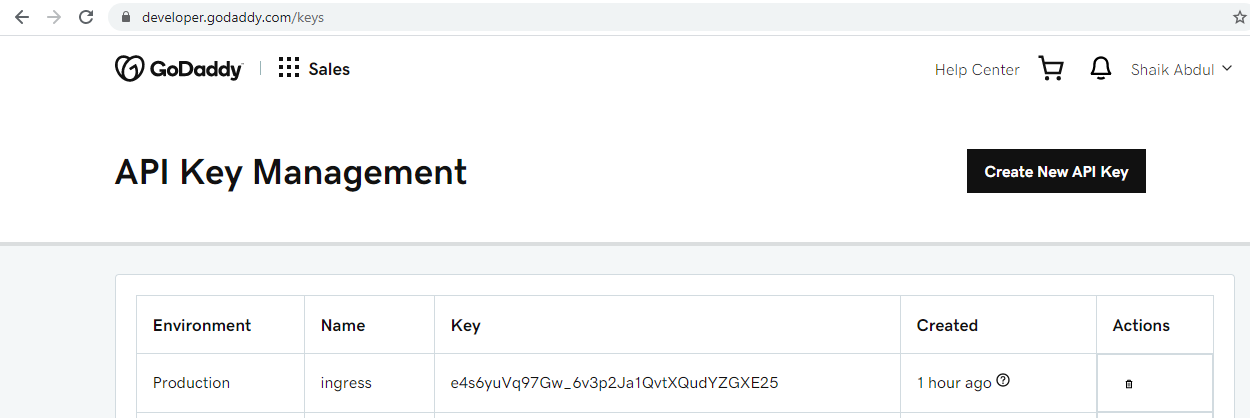
jetstack/cert-manager

kubectl get pods --namespace cert-manager

I am using Godaddy from these demo

Create the API Key and secret in GoDaddy

<https://developer.godaddy.com/keys>



- Deploy the ACME Webhook for GoDaddy

- helm install godaddy-webhook --namespace cert-manager ./deploy/godaddy-webhook

(OR)

git clone <https://github.com/khaleel-cloud786/godaddy-webhook.git>

cd godaddy-webhook

kubectl apply -f deploy/webhook-all.yml --validate=false

**Issuer:**

Create a Secret containing as key parameter the concatenation of the Godaddy Api and Secret separated by ":"

cat <<EOF > secret.yml

apiVersion: v1

kind: Secret

metadata:

name: godaddy-api-key

type: Opaque

stringData:

key: <GODADDY\_API:GODADDY\_SECRET>

EOF

kubectl appy -f secret.yml –n cert-manager

**ClusterIssuer**

Create a ClusterIssuer resource to specify the address of the ACME staging or production server to access. Add the DNS01 Solver Config that this webhook will use to communicate with the API of the Godaddy Server in order to create or delete an ACME Challenge TXT record that the DNS Provider will accept/refuse if the domain name exists.

cat <<EOF > clusterissuer.yml

EOF apiVersion: cert-manager.io/v1alpha2

kind: ClusterIssuer

metadata:

name: letsencrypt-prod

spec:

acme:

# prod : https://acme-v02.api.letsencrypt.org/directory

# staging : https://acme-staging-v02.api.letsencrypt.org/directory

server: https://acme-v02.api.letsencrypt.org/directory

# ACME Email address

email: khaleel@shaikabdulrayyan.com

privateKeySecretRef:

name: letsencrypt-prod

solvers:

- selector:

dnsNames:

- '\*.shaikabdulrayyan.com'

dns01:

webhook:

config:

apiKeySecretRef:

name: godaddy-api-key

key: key

production: true

ttl: 600

groupName: acme.shaikabdulkhaleel

solverName: godaddy

EOF

kubectl apply -f clusterissuer.yml –n cert-manager

Next, create for each of your domain where you need a signed certificate from the Letsencrypt authority the following certificate

cat <<EOF > certificate.yml

apiVersion: cert-manager.io/v1alpha2

kind: Certificate

metadata:

name: wildcard-shaikabdulrayyan-com

spec:

secretName: wildcard-shaikabdulrayyan-com-tls

renewBefore: 240h

dnsNames:

- '\*.shaikabdulrayyan.com'

issuerRef:

name: letsencrypt-prod

kind: ClusterIssuer

EOF

kubectl apply -f certificate.yml –n cert-manager

Next, deploy the following ingress resource

apiVersion: networking.k8s.io/v1beta1

kind: Ingress

metadata:

annotations:

kubernetes.io/ingress.class: nginx

cert-manager.io/issuer: "letsencrypt-prod"

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

name: ingress-resource-demo

spec:

tls:

- hosts:

- ingress.shaikabdulrayyan.com

secretName: wildcard-shaikabdulrayyanrayyan-com-tls

rules:

- host: ingress.shaikabdulrayyan.com

http:

paths:

- path: /

backend:

serviceName: ingress-demo-deploy-v1

servicePort: 80

- path: /v2

backend:

serviceName: ingress-demo-deploy-v2

servicePort: 80

kubectl describe certificates wildcard-shaikabdulrayyanrayyan-com -n cert-manager

kubectl describe certificaterequests wildcard-shaikabdulrayyanrayyan-com-wj56p -n cert-manager

kubectl describe order wildcard-shaikabdulrayyanrayyan-com-vz5zd-1794323654 -n cert-manager

kubectl describe challenge wildcard-shaikabdulrayyanrayyan-com-vz5zd-1794323654-826467321 -n cert-manager

kubectl describe secrets godaddy-api-key -n cert-manager

