Installation with Manifests

This document describes how to install the NGINX Ingress Controller in your Kubernetes cluster using Kubernetes manifests.

Prerequisites

1. Make sure you have access to the Ingress Controller image:
   - For NGINX Ingress Controller, use the image `nginx/nginx-ingress` from DockerHub.
   - For NGINX Plus Ingress Controller, see here for details on how to pull the image from the F5 Docker registry.
   - To pull from the F5 Container registry in your Kubernetes cluster, configure a docker registry secret using your JWT token from the MyF5 portal by following the instructions from here.
   - It is also possible to build your own image and push it to your private Docker registry by following the instructions from here.

2. Clone the Ingress Controller repo and change into the deployments folder:

```
$ git clone https://github.com/nginxinc/kubernetes-ingress.git --branch v3.1.1
$ cd kubernetes-ingress/deployments
```

1. Configure RBAC

1. Create a namespace and a service account for the Ingress Controller:

```
$ kubectl apply -f common/ns-and-sa.yaml
```

2. Create a cluster role and cluster role binding for the service account:

```
$ kubectl apply -f rbac/rbac.yaml
```

3. (App Protect only) Create the App Protect role and role binding:

```
$ kubectl apply -f rbac/ap-rbac.yaml
```

4. (App Protect DoS only) Create the App Protect DoS role and role binding:

```
$ kubectl apply -f rbac/apdos-rbac.yaml
```

Note: To perform this step you must be a cluster admin. Follow the documentation of your Kubernetes platform to configure the admin access. For GKE, see the Role-Based Access Control doc.

2. Create Common Resources

In this section, we create resources common for most of the Ingress Controller installations: **NOTE**: Installing the `default-server-secret.yaml` is optional and is required only if you are using the default server TLS secret command line argument. It is recommended that users provide their own certificate. Otherwise, step 1 can be ignored.

1. Create a secret with a TLS certificate and a key for the default server in NGINX (below assumes you are in the `kubernetes-ingress/deployment` directory):

```
$ kubectl apply -f ../examples/shared-examples/default-server-secret/default-server-secret.yaml
```

   Note: The default server returns the Not Found page with the 404 status code for all requests for domains for which there are no Ingress rules defined. For testing purposes we include a self-signed certificate and key that we generated. However, we recommend that you use your own certificate and key.

2. Create a config map for customizing NGINX configuration:

```
$ kubectl apply -f common/nginx-config.yaml
```

3. Create an IngressClass resource:

```
$ kubectl apply -f common/ingress-class.yaml
```

If you would like to set the Ingress Controller as the default one, uncomment the annotation `ingressclass.kubernetes.io/is-default-class`. With this annotation set to true all the new Ingresses without an ingressClassName field specified will be assigned this IngressClass.

Note: The Ingress Controller will fail to start without an IngressClass resource.

3. Create Custom Resources

Note: By default, it is required to create custom resource definitions for VirtualServer, VirtualServerRoute, TransportServer and Policy. Otherwise, the Ingress Controller pods will not become `Ready`. If you'd like to disable that requirement, configure `-enable-custom-resources` command-line argument to `false` and skip this section.

1. Create custom resource definitions for VirtualServer and VirtualServerRoute, TransportServer and Policy resources:

```
$ kubectl apply -f common/crds/k8s.nginx.org_virtualservers.yaml
$ kubectl apply -f common/crds/k8s.nginx.org_virtualserverroutes.yaml
$ kubectl apply -f common/crds/k8s.nginx.org_transportservers.yaml
$ kubectl apply -f common/crds/k8s.nginx.org_policies.yaml
```

2. If you would like to use the TCP and UDP load balancing features of the Ingress Controller, create the following additional resources:

Create a custom resource definition for GlobalConfiguration resource:

```
$ kubectl apply -f common/crds/k8s.nginx.org_globalconfigurations.yaml
```

3. If you would like to use the App Protect WAF module, create the following additional resources:

Create a custom resource definition for `APPolicy`, `APLogConf` and `APUserSig`:

```
$ kubectl apply -f common/crds/appprotect.f5.com_aplogconfs.yaml
$ kubectl apply -f common/crds/appprotect.f5.com_appolicies.yaml
$ kubectl apply -f common/crds/appprotect.f5.com_apusersigs.yaml
```

4. If you would like to use the App Protect DoS module, create the following additional resources:

Create a custom resource definition for `APDosPolicy`, `APDosLogConf` and `DosProtectedResource`:

```
$ kubectl apply -f common/crds/appprotectdos.f5.com_apdoslogconfs.yaml
$ kubectl apply -f common/crds/appprotectdos.f5.com_apdospolicy.yaml
$ kubectl apply -f common/crds/appprotectdos.f5.com_dosprotectedresources.yaml
```

3. Deploy the Ingress Controller

We include two options for deploying the Ingress Controller:

- *Deployment*. Use a Deployment if you plan to dynamically change the number of Ingress Controller replicas.
- *DaemonSet*. Use a DaemonSet for deploying the Ingress Controller on every node or a subset of nodes.

Before creating a Deployment or Daemonset resource, make sure to update the command-line arguments of the Ingress Controller container in the corresponding manifest file according to your requirements.

Deploy Arbitrator for NGINX App Protect DoS

If you would like to use the App Protect DoS module, you will need to deploy the Arbitrator.

- Build your own image and push it to your private Docker registry by following the instructions from here.
- Run the Arbitrator by using a Deployment and Service

```
$ kubectl apply -f deployment/appprotect-dos-arb.yaml
$ kubectl apply -f service/appprotect-dos-arb-svc.yaml
```

3.1 Run the Ingress Controller

- *Use a Deployment*. When you run the Ingress Controller by using a Deployment, by default, Kubernetes will create one Ingress Controller pod.

  For NGINX, run:

```
$ kubectl apply -f deployment/nginx-ingress.yaml
```

For NGINX Plus, run:

```
$ kubectl apply -f deployment/nginx-plus-ingress.yaml
```

Note: Update the `nginx-plus-ingress.yaml` with the chosen image from the F5 Container registry; or the container image that you have built.

- *Use a DaemonSet*: When you run the Ingress Controller by using a DaemonSet, Kubernetes will create an Ingress Controller pod on every node of the cluster.

  See also: See the Kubernetes DaemonSet docs to learn how to run the Ingress Controller on a subset of nodes instead of on every node of the cluster.

  For NGINX, run:

  ```
  $ kubectl apply -f daemon-set/nginx-ingress.yaml
  ```

  For NGINX Plus, run:

  ```
  $ kubectl apply -f daemon-set/nginx-plus-ingress.yaml
  ```

  Note: Update the `nginx-plus-ingress.yaml` with the chosen image from the F5 Container registry; or the container image that you have built.

3.2 Check that the Ingress Controller is Running

Run the following command to make sure that the Ingress Controller pods are running:

```
$ kubectl get pods --namespace=nginx-ingress
```

4. Get Access to the Ingress Controller

If you created a daemonset, ports 80 and 443 of the Ingress Controller container are mapped to the same ports of the node where the container is running. To access the Ingress Controller, use those ports and an IP address of any node of the cluster where the Ingress Controller is running.

If you created a deployment, below are two options for accessing the Ingress Controller pods.

4.1 Create a Service for the Ingress Controller Pods

- *Use a NodePort service*.

  Create a service with the type *NodePort*:

  ```
  $ kubectl create -f service/nodeport.yaml
  ```

  Kubernetes will randomly allocate two ports on every node of the cluster. To access the Ingress Controller, use an IP address of any node of the cluster along with the two allocated ports.

  Read more about the type NodePort in the Kubernetes documentation.

- *Use a LoadBalancer service*:

1. Create a service using a manifest for your cloud provider:

   - For GCP or Azure, run:

     ```
     $ kubectl apply -f service/loadbalancer.yaml
     ```

   - For AWS, run:

     ```
     $ kubectl apply -f service/loadbalancer-aws-elb.yaml
     ```

     Kubernetes will allocate a Classic Load Balancer (ELB) in TCP mode with the PROXY protocol enabled to pass the client's information (the IP address and the port). NGINX must be configured to use the PROXY protocol:

     - Add the following keys to the config map file `nginx-config.yaml` from the Step 2:

       ```
       proxy-protocol: "True"
       real-ip-header: "proxy_protocol"
       set-real-ip-from: "0.0.0.0/0"
       ```

     - Update the config map:

```
kubectl apply -f common/nginx-config.yaml
```

Note: For AWS, additional options regarding an allocated load balancer are available, such as the type of a load balancer and SSL termination. Read the Kubernetes documentation to learn more.

Kubernetes will allocate and configure a cloud load balancer for load balancing the Ingress Controller pods.

2. Use the public IP of the load balancer to access the Ingress Controller. To get the public IP:

   - For GCP or Azure, run:

     ```
     $ kubectl get svc nginx-ingress --namespace=nginx-ingress
     ```

   - In case of AWS ELB, the public IP is not reported by `kubectl`, because the ELB IP addresses are not static. In general, you should rely on the ELB DNS name instead of the ELB IP addresses. However, for testing purposes, you can get the DNS name of the ELB using `kubectl describe` and then run `nslookup` to find the associated IP address:

     ```
     $ kubectl describe svc nginx-ingress --namespace=nginx-ingress
     ```

     You can resolve the DNS name into an IP address using `nslookup`:

     ```
     $ nslookup <dns-name>
     ```

     The public IP can be reported in the status of an ingress resource. See the Reporting Resources Status doc for more details.

   Learn more about type LoadBalancer in the Kubernetes documentation.

Uninstall the Ingress Controller

1. Delete the `nginx-ingress` namespace to uninstall the Ingress Controller along with all the auxiliary resources that were created:

   ```
   $ kubectl delete namespace nginx-ingress
   ```

2. Delete the ClusterRole and ClusterRoleBinding:

   ```
   $ kubectl delete clusterrole nginx-ingress
   $ kubectl delete clusterrolebinding nginx-ingress
   ```

3. Delete the Custom Resource Definitions:

   Note: This step will also remove all associated Custom Resources.

   ```
   $ kubectl delete -f common/crds/
   ```