Installation Guide ¶

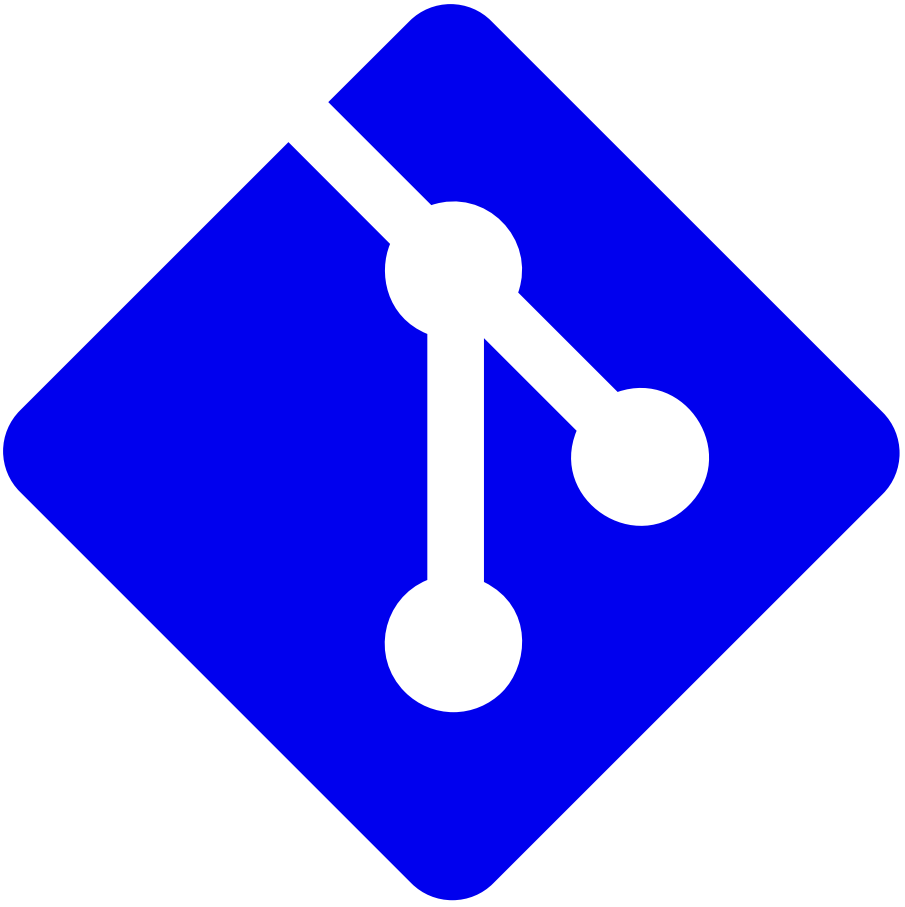
kubernetes.github.io/ingress-nginx/deploy

Skip to content

Ingress-Nginx Controller

Installation Guide

Type to start searching



kubernetes/ingress-nginx

- controller-v1.13.3
- 19k
- 8.4k

- Welcome
- Deployment
- User Guide
- Examples
- Developer Guide

- FAQ

There are multiple ways to install the Ingress-Nginx Controller:

- with Helm, using the project repository chart;
- with `kubectl apply`, using YAML manifests;
- with specific addons (e.g. for minikube or MicroK8s).

On most Kubernetes clusters, the ingress controller will work without requiring any extra configuration. If you want to get started as fast as possible, you can check the quick start instructions. However, in many environments, you can improve the performance or get better logs by enabling extra features. We recommend that you check the environment-specific instructions for details about optimizing the ingress controller for your particular environment or cloud provider.

Contents ¶

Quick start ¶

If you have Helm, you can deploy the ingress controller with the following command:

```
helm upgrade --install ingress-nginx ingress-nginx \
  --repo https://kubernetes.github.io/ingress-nginx \
  --namespace ingress-nginx --create-namespace
```

It will install the controller in the `ingress-nginx` namespace, creating that namespace if it doesn't already exist.

Info

This command is *idempotent*:

- if the ingress controller is not installed, it will install it,
- if the ingress controller is already installed, it will upgrade it.

If you want a full list of values that you can set, while installing with Helm, then run:

```
helm show values ingress-nginx --repo https://kubernetes.github.io/ingress-nginx
```

Helm install on AWS/GCP/Azure/Other providers

The *ingress-nginx-controller helm-chart is a generic install out of the box*. The default set of helm values is **not** configured for installation on any infra provider. The annotations that are applicable to the cloud provider must be customized by the users.
See AWS LB Controller.
Examples of some annotations recommended (healthecheck ones are required for target-type IP) for the service resource of `--type LoadBalancer` on AWS are below:

```
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-target-group-attributes:
deregistration_delay.timeout_seconds=270
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-healthcheck-path: /healthz
    service.beta.kubernetes.io/aws-load-balancer-healthcheck-port: "10254"
    service.beta.kubernetes.io/aws-load-balancer-healthcheck-protocol: http
    service.beta.kubernetes.io/aws-load-balancer-healthcheck-success-codes: 200-
299
    service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
    service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp
    service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-
enabled: "true"
    service.beta.kubernetes.io/aws-load-balancer-type: nlb
    service.beta.kubernetes.io/aws-load-balancer-manage-backend-security-group-
rules: "true"
    service.beta.kubernetes.io/aws-load-balancer-access-log-enabled: "true"
    service.beta.kubernetes.io/aws-load-balancer-security-groups: "sg-something1
sg-something2"
    service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-name:
"somebucket"
    service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-prefix:
"ingress-nginx"
    service.beta.kubernetes.io/aws-load-balancer-access-log-emit-interval: "5"
```

If you don't have Helm or if you prefer to use a YAML manifest, you can run the following command instead:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-
nginx/controller-v1.13.3/deploy/static/provider/cloud/deploy.yaml
```

Info

The YAML manifest in the command above was generated with `helm template`, so you will end up with almost the same resources as if you had used Helm to install the controller.

Attention

If you are running an old version of Kubernetes (1.18 or earlier), please read this paragraph for specific instructions. Because of api deprecations, the default manifest may not work on your cluster. Specific manifests for supported Kubernetes versions are available within a sub-folder of each provider.

Firewall configuration ¶

To check which ports are used by your installation of ingress-nginx, look at the output of `kubectl -n ingress-nginx get pod -o yaml`. In general, you need:

- Port 8443 open between all hosts on which the kubernetes nodes are running. This is used for the ingress-nginx admission controller.
- Port 80 (for HTTP) and/or 443 (for HTTPS) open to the public on the kubernetes nodes to which the DNS of your apps are pointing.

Pre-flight check ¶

A few pods should start in the `ingress-nginx` namespace:

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

After a while, they should all be running. The following command will wait for the ingress controller pod to be up, running, and ready:

```
kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=120s
```

Local testing ¶

Let's create a simple web server and the associated service:

```
kubectl create deployment demo --image=httpd --port=80
kubectl expose deployment demo
```

Then create an ingress resource. The following example uses a host that maps to `localhost`:

```
kubectl create ingress demo-localhost --class=nginx \
  --rule="demo.localdev.me/*=demo:80"
```

Now, forward a local port to the ingress controller:

```
kubectl port-forward --namespace=ingress-nginx service/ingress-nginx-controller
8080:80
```

Info

A note on DNS & network-connection. This documentation assumes that a user has awareness of the DNS and the network routing aspects involved in using ingress. The port-forwarding mentioned above, is the easiest way to demo the working of ingress. The "kubectl port-forward..." command above has forwarded the port number 8080, on the localhost's tcp/ip stack, where the command was typed, to the port number 80, of the service created by the installation of ingress-nginx controller. So now, the traffic sent to port number 8080 on localhost will reach the port number 80, of the ingress-controller's service. Port-forwarding is not for a production environment use-case. But here we use port-forwarding, to simulate a HTTP request, originating from outside the cluster, to reach the service of the ingress-nginx controller, that is exposed to receive traffic from outside the cluster.
This issue shows a typical DNS problem and its solution.

At this point, you can access your deployment using curl ;

```
curl --resolve demo.localdev.me:8080:127.0.0.1 http://demo.localdev.me:8080
```

You should see a HTML response containing text like **"It works!"**.

Online testing ¶

If your Kubernetes cluster is a "real" cluster that supports services of type `LoadBalancer`, it will have allocated an external IP address or FQDN to the ingress controller.

You can see that IP address or FQDN with the following command:

```
kubectl get service ingress-nginx-controller --namespace=ingress-nginx
```

It will be the `EXTERNAL-IP` field. If that field shows `<pending>`, this means that your Kubernetes cluster wasn't able to provision the load balancer (generally, this is because it doesn't support services of type `LoadBalancer`).

Once you have the external IP address (or FQDN), set up a DNS record pointing to it. Then you can create an ingress resource. The following example assumes that you have set up a DNS record for `www.demo.io`:

```
kubectl create ingress demo --class=nginx \
  --rule="www.demo.io/*=demo:80"
```

Alternatively, the above command can be rewritten as follows for the `--rule` command and below.

```
kubectl create ingress demo --class=nginx \
  --rule www.demo.io/=demo:80
```

You should then be able to see the "It works!" page when you connect to http://www.demo.io/. Congratulations, you are serving a public website hosted on a Kubernetes cluster! 🎉

Environment-specific instructions ¶

Local development clusters ¶

minikube ¶

The ingress controller can be installed through minikube's addons system:

```
minikube addons enable ingress
```

MicroK8s ¶

The ingress controller can be installed through MicroK8s's addons system:

```
microk8s enable ingress
```

Please check the MicroK8s documentation page for details.

Docker Desktop ¶

Kubernetes is available in Docker Desktop:

- Mac, from version 18.06.0-ce
- Windows, from version 18.06.0-ce

First, make sure that Kubernetes is enabled in the Docker settings. The command `kubectl get nodes` should show a single node called `docker-desktop`.

The ingress controller can be installed on Docker Desktop using the default quick start instructions.

On most systems, if you don't have any other service of type `LoadBalancer` bound to port 80, the ingress controller will be assigned the `EXTERNAL-IP` of `localhost`, which means that it will be reachable on localhost:80. If that doesn't work, you might have to fall back to the `kubectl port-forward` method described in the local testing section.

Rancher Desktop ¶

Rancher Desktop provides Kubernetes and Container Management on the desktop. Kubernetes is enabled by default in Rancher Desktop.

Rancher Desktop uses K3s under the hood, which in turn uses Traefik as the default ingress controller for the Kubernetes cluster. To use Ingress-Nginx Controller in place of the default Traefik, disable Traefik from Preference > Kubernetes menu.

Once traefik is disabled, the Ingress-Nginx Controller can be installed on Rancher Desktop using the default quick start instructions. Follow the instructions described in the local testing section to try a sample.

Cloud deployments ¶

If the load balancers of your cloud provider do active healthchecks on their backends (most do), you can change the `externalTrafficPolicy` of the ingress controller Service to `Local` (instead of the default `Cluster`) to save an extra hop in some cases. If you're installing with Helm, this can be done by adding `--set controller.service.externalTrafficPolicy=Local` to the `helm install` or `helm upgrade` command.

Furthermore, if the load balancers of your cloud provider support the PROXY protocol, you can enable it, and it will let the ingress controller see the real IP address of the clients. Otherwise, it will generally see the IP address of the upstream load balancer. This must be done both in the ingress controller (with e.g. `--set controller.config.use-proxy-protocol=true`) and in the cloud provider's load balancer configuration to function correctly.

In the following sections, we provide YAML manifests that enable these options when possible, using the specific options of various cloud providers.

AWS ¶

In AWS, we use a Network load balancer (NLB) to expose the Ingress-Nginx Controller behind a Service of `Type=LoadBalancer`.

Info

The provided templates illustrate the setup for legacy in-tree service load balancer for AWS NLB. AWS provides the documentation on how to use Network load balancing on Amazon EKS with AWS Load Balancer Controller.

Network Load Balancer (NLB) ¶

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.13.3/deploy/static/provider/aws/deploy.yaml
```

TLS termination in AWS Load Balancer (NLB) ¶

By default, TLS is terminated in the ingress controller. But it is also possible to terminate TLS in the Load Balancer. This section explains how to do that on AWS using an NLB.

1. Download the deploy.yaml template

   ```
   wget https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.13.3/deploy/static/provider/aws/nlb-with-tls-termination/deploy.yaml
   ```

2. Edit the file and change the VPC CIDR in use for the Kubernetes cluster:

   ```
   proxy-real-ip-cidr: XXX.XXX.XXX/XX
   ```

3. Change the AWS Certificate Manager (ACM) ID as well:

   ```
   arn:aws:acm:us-west-2:XXXXXXXX:certificate/XXXXXX-XXXXXXX-XXXXXXX-XXXXXXXX
   ```

4. Deploy the manifest:

```
kubectl apply -f deploy.yaml
```

NLB Idle Timeouts ¶

The default idle timeout value for TCP flows is 350 seconds and can be modified to any value between 60-6000 seconds.

For this reason, you need to ensure the keepalive_timeout value is configured less than your configured idle timeout to work as expected.

By default, NGINX `keepalive_timeout` is set to `75s`.

More information with regard to timeouts can be found in the official AWS documentation

GCE-GKE ¶

First, your user needs to have `cluster-admin` permissions on the cluster. This can be done with the following command:

```
kubectl create clusterrolebinding cluster-admin-binding \
  --clusterrole cluster-admin \
  --user $(gcloud config get-value account)
```

Then, the ingress controller can be installed like this:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-
nginx/controller-v1.13.3/deploy/static/provider/cloud/deploy.yaml
```

Warning

For private clusters, you will need to either add a firewall rule that allows master nodes access to port `8443/tcp` on worker nodes, or change the existing rule that allows access to port `80/tcp`, `443/tcp` and `10254/tcp` to also allow access to port `8443/tcp`. More information can be found in the Official GCP Documentation.

See the GKE documentation on adding rules and the Kubernetes issue for more detail.

Proxy-protocol is supported in GCE check the Official Documentations on how to enable.

Azure ¶

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-
nginx/controller-v1.13.3/deploy/static/provider/cloud/deploy.yaml
```

More information with regard to Azure annotations for ingress controller can be found in the official AKS documentation.

Digital Ocean ¶

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-
nginx/controller-v1.13.3/deploy/static/provider/do/deploy.yaml
```

By default the service object of the ingress-nginx-controller for Digital-Ocean, only configures one annotation. Its this one `service.beta.kubernetes.io/do-loadbalancer-enable-proxy-protocol: "true"`. While this makes the service functional, it was reported that the Digital-Ocean LoadBalancer graphs shows `no data`, unless a few other annotations are also configured. Some of these other annotations require values that can not be generic and hence not forced in a out-of-the-box installation. These annotations and a discussion on them is well documented in this issue. Please refer to the issue to add annotations, with values specific to user, to get graphs of the DO-LB populated with data.

Scaleway ¶

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.13.3/deploy/static/provider/scw/deploy.yaml
```

Refer to the dedicated tutorial in the Scaleway documentation for configuring the proxy protocol for ingress-nginx with the Scaleway load balancer.

Exoscale ¶

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/exoscale/deploy.yaml
```

The full list of annotations supported by Exoscale is available in the Exoscale Cloud Controller Manager documentation.

Oracle Cloud Infrastructure ¶

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.13.3/deploy/static/provider/cloud/deploy.yaml
```

A complete list of available annotations for Oracle Cloud Infrastructure can be found in the OCI Cloud Controller Manager documentation.

OVHcloud ¶

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm -n ingress-nginx install ingress-nginx ingress-nginx/ingress-nginx --create-namespace
```

You can find the complete tutorial.

Bare metal clusters ¶

This section is applicable to Kubernetes clusters deployed on bare metal servers, as well as "raw" VMs where Kubernetes was installed manually, using generic Linux distros (like CentOS, Ubuntu...)

For quick testing, you can use a NodePort. This should work on almost every cluster, but it will typically use a port in the range 30000-32767.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-
nginx/controller-v1.13.3/deploy/static/provider/baremetal/deploy.yaml
```

For more information about bare metal deployments (and how to use port 80 instead of a random port in the 30000-32767 range), see bare-metal considerations.

Miscellaneous ¶

Checking ingress controller version ¶

Run `/nginx-ingress-controller --version` within the pod, for instance with `kubectl exec`:

```
POD_NAMESPACE=ingress-nginx
POD_NAME=$(kubectl get pods -n $POD_NAMESPACE -l app.kubernetes.io/name=ingress-
nginx --field-selector=status.phase=Running -o name)
kubectl exec $POD_NAME -n $POD_NAMESPACE -- /nginx-ingress-controller --version
```

Scope ¶

By default, the controller watches Ingress objects from all namespaces. If you want to change this behavior, use the flag `--watch-namespace` or check the Helm chart value `controller.scope` to limit the controller to a single namespace. Although the use of this flag is not popular, one important fact to note is that the secret containing the default-ssl-certificate needs to also be present in the watched namespace(s).

See also "How to install multiple Ingress controllers in the same cluster" for more details.

Webhook network access ¶

Warning

The controller uses an admission webhook to validate Ingress definitions. Make sure that you don't have Network policies or additional firewalls preventing connections from the API server to the `ingress-nginx-controller-admission` service.

Certificate generation ¶

Attention

The first time the ingress controller starts, two Jobs create the SSL Certificate used by the admission webhook.

This can cause an initial delay of up to two minutes until it is possible to create and validate Ingress definitions.

You can wait until it is ready to run the next command:

```
kubectl wait --namespace ingress-nginx \
 --for=condition=ready pod \
 --selector=app.kubernetes.io/component=controller \
 --timeout=120s
```

Running on Kubernetes versions older than 1.19 ¶

Ingress resources evolved over time. They started with `apiVersion: extensions/v1beta1`, then moved to `apiVersion: networking.k8s.io/v1beta1` and more recently to `apiVersion: networking.k8s.io/v1`.

Here is how these Ingress versions are supported in Kubernetes:

- before Kubernetes 1.19, only `v1beta1` Ingress resources are supported
- from Kubernetes 1.19 to 1.21, both `v1beta1` and `v1` Ingress resources are supported
- in Kubernetes 1.22 and above, only `v1` Ingress resources are supported

And here is how these Ingress versions are supported in Ingress-Nginx Controller:

- before version 1.0, only `v1beta1` Ingress resources are supported
- in version 1.0 and above, only `v1` Ingress resources are

As a result, if you're running Kubernetes 1.19 or later, you should be able to use the latest version of the NGINX Ingress Controller; but if you're using an old version of Kubernetes (1.18 or earlier) you will have to use version 0.X of the Ingress-Nginx Controller (e.g. version 0.49).

The Helm chart of the Ingress-Nginx Controller switched to version 1 in version 4 of the chart. In other words, if you're running Kubernetes 1.19 or earlier, you should use version 3.X of the chart (this can be done by adding `--version='<4'` to the `helm install` command).