# Ingress

**FEATURE STATE:** `Kubernetes v1.1` beta

An API object that manages external access to the services in a cluster, typically HTTP.

Ingress may provide load balancing, SSL termination and name-based virtual hosting.

## Terminology

For clarity, this guide defines the following terms:

- Node: A worker machine in Kubernetes, part of a cluster.
- Cluster: A set of Nodes that run containerized applications managed by Kubernetes. For this example, and in most common Kubernetes deployments, nodes in the cluster are not part of the public internet.
- Edge router: A router that enforces the firewall policy for your cluster. This could be a gateway managed by a cloud provider or a physical piece of hardware.
- Cluster network: A set of links, logical or physical, that facilitate communication within a cluster according to the Kubernetes networking model.

## What is Ingress?

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.

```
    internet
        |
   [ Ingress ]
   --|-----|--
   [ Services ]
```

An Ingress may be configured to give Services externally-reachable URLs, load balance traffic, terminate SSL / TLS, and offer name based virtual hosting. An Ingress controller is responsible for fulfilling the Ingress, usually with a load balancer, though it may also configure your edge router or additional frontends to help handle the traffic.

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.

## Prerequisites

You must have an ingress controller to satisfy an Ingress. Only creating an Ingress resource has no effect.

You may need to deploy an Ingress controller such as ingress-nginx. You can choose from a number of Ingress controllers.

Ideally, all Ingress controllers should fit the reference specification. In reality, the various Ingress controllers operate slightly differently.

**Note:** Make sure you review your Ingress controller's documentation to understand the caveats of choosing it.

## The Ingress Resource

A minimal Ingress resource example:

```
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          serviceName: test
          servicePort: 80
```

As with all other Kubernetes resources, an Ingress needs `apiVersion`, `kind`, and `metadata` fields. The name of an Ingress object must be a valid DNS subdomain name. For general information about working with config files, see deploying applications, configuring containers, managing resources. Ingress frequently uses annotations to configure some options depending on the Ingress controller, an example of which is the rewrite-target annotation. Different Ingress controller support different annotations. Review the documentation for your choice of Ingress controller to learn which annotations are supported.

The Ingress spec has all the information needed to configure a load balancer or proxy server. Most importantly, it contains a list of rules matched against all incoming requests. Ingress resource only supports rules for directing HTTP traffic.

## Ingress rules

Each HTTP rule contains the following information:

- An optional host. In this example, no host is specified, so the rule applies to all inbound HTTP traffic through the IP address specified. If a host is provided (for example, foo.bar.com), the rules apply to that host.
- A list of paths (for example, `/testpath`), each of which has an associated backend defined with a `serviceName` and `servicePort`. Both the host and path must match the content of an incoming request before the load balancer directs traffic to the referenced Service.
- A backend is a combination of Service and port names as described in the Service doc. HTTP (and HTTPS) requests to the Ingress that matches the host and path of the rule are sent to the listed backend.

A default backend is often configured in an Ingress controller to service any requests that do not match a path in the spec.

## Default Backend

An Ingress with no rules sends all traffic to a single default backend. The default backend is typically a configuration option of the Ingress controller and is not specified in your Ingress resources.

## Path Types

Each path in an Ingress has a corresponding path type. There are three supported path types:

- *`ImplementationSpecific`* (default): With this path type, matching is up to the IngressClass. Implementations can treat this as a separate `pathType` or treat it identically to `Prefix` or `Exact` path types.
- *`Exact`*: Matches the URL path exactly and with case sensitivity.
- *`Prefix`*: Matches based on a URL path prefix split by `/`. Matching is case sensitive and done on a path element by element basis. A path element refers to the list of labels in the path split by the `/` separator. A request is a match for path *p* if every *p* is an element-wise prefix of *p* of the request path.

> **Note:** If the last element of the path is a substring of the last element in request path, it is not a match (for example: `/foo/bar` matches `/foo/bar/baz`, but does not match `/foo/barbaz`).

### Multiple Matches

In some cases, multiple paths within an Ingress will match a request. In those cases precedence will be given first to the longest matching path. If two paths are still equally matched, precedence will be given to paths with an exact path type over prefix path type.

## Ingress Class

Ingresses can be implemented by different controllers, often with different configuration. Each Ingress should specify a class, a reference to an IngressClass resource that contains additional configuration including the name of the controller that should implement the class.

```
apiVersion: networking.k8s.io/v1beta1
kind: IngressClass
metadata:
  name: external-lb
spec:
  controller: example.com/ingress-controller
  parameters:
    apiGroup: k8s.example.com/v1alpha
    kind: IngressParameters
    name: external-lb
```

## Deprecated Annotation

Before the IngressClass resource and `ingressClassName` field were added in Kubernetes 1.18, Ingress classes were specified with a `kubernetes.io/ingress.class` annotation on the Ingress. This annotation was never formally defined, but was widely supported by Ingress controllers.

The newer `ingressClassName` field on Ingresses is a replacement for that annotation, but is not a direct equivalent. While the annotation was generally used to reference the name of the Ingress controller that should implement the Ingress, the field is a reference to an IngressClass resource that contains additional Ingress configuration, including the name of the Ingress controller.

## Default Ingress Class

You can mark a particular IngressClass as default for your cluster. Setting the `ingressclass.kubernetes.io/is-default-class` annotation to `true` on an IngressClass resource will ensure that new Ingresses without an `ingressClassName` field specified will be assigned this default IngressClass.

**Caution:** If you have more than one IngressClass marked as the default for your cluster, the admission controller prevents creating new Ingress objects that don't have an `ingressClassName` specified. You can resolve this by ensuring that at most 1 IngressClasess are marked as default in your cluster.

## Types of Ingress

### Single Service Ingress

There are existing Kubernetes concepts that allow you to expose a single Service (see alternatives). You can also do this with an Ingress by specifying a *default backend* with no rules.

service/networking/ingress.yaml

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test-ingress
spec:
  backend:
    serviceName: testsvc
    servicePort: 80
```

```
kubectl get ingress test-ingress
```

```
NAME           HOSTS     ADDRESS           PORTS     AGE
test-ingress   *         203.0.113.123   80        59s
```

Where `203.0.113.123` is the IP allocated by the Ingress controller to satisfy this Ingress.

> **Note:** Ingress controllers and load balancers may take a minute or two to allocate an IP address. Until that time, you often see the address listed as `<pending>`.

## Simple fanout

A fanout configuration routes traffic from a single IP address to more than one Service, based on the HTTP URI being requested. An Ingress allows you to keep the number of load balancers down to a minimum. For example, a setup like:

```
foo.bar.com -> 178.91.123.132 -> / foo    service1:4200
                                 / bar    service2:8080
```

would require an Ingress such as:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: simple-fanout-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: service1
          servicePort: 4200
      - path: /bar
        backend:
          serviceName: service2
          servicePort: 8080
```

When you create the Ingress with `kubectl apply -f`:

```
kubectl describe ingress simple-fanout-example
```

```
Address:          178.91.123.132
Default backend:  default-http-backend:80 (10.8.2.3:8080)
Rules:
  Host         Path  Backends
  ----         ----  --------
  foo.bar.com
               /foo   service1:4200 (10.8.0.90:4200)
               /bar   service2:8080 (10.8.0.91:8080)
Annotations:
  nginx.ingress.kubernetes.io/rewrite-target:  /
Events:
  Type     Reason  Age                From                     Message
  ----     ------  ----               ----                     -------
  Normal   ADD     22s                loadbalancer-controller  default/test
```

The Ingress controller provisions an implementation-specific load balancer that satisfies the Ingress, as long as the Services ( `service1` , `service2` ) exist. When it has done so, you can see the address of the load balancer at the Address field.

**Note:** Depending on the Ingress controller you are using, you may need to create a default-http-backend Service.

## Name based virtual hosting

Name-based virtual hosts support routing HTTP traffic to multiple host names at the same IP address.

```
foo.bar.com --|                 |-> foo.bar.com service1:80
              | 178.91.123.132  |
bar.foo.com --|                 |-> bar.foo.com service2:80
```

The following Ingress tells the backing load balancer to route requests based on the Host header.

```
  name: name-virtual-host-ingress
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: service1
          servicePort: 80
  - host: bar.foo.com
    http:
      paths:
      - backend:
          serviceName: service2
          servicePort: 80
```

If you create an Ingress resource without any hosts defined in the rules, then any web traffic to the IP address of your Ingress controller can be matched without a name based virtual host being required.

For example, the following Ingress resource will route traffic requested for `first.bar.com` to `service1`, `second.foo.com` to `service2`, and any traffic to the IP address without a hostname defined in request (that is, without a request header being presented) to `service3`.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: name-virtual-host-ingress
spec:
  rules:
  - host: first.bar.com
    http:
      paths:
      - backend:
          serviceName: service1
          servicePort: 80
  - host: second.foo.com
    http:
      paths:
      - backend:
          serviceName: service2
          servicePort: 80
  - http:
      paths:
      - backend:
          serviceName: service3
          servicePort: 80
```

## TLS

through the SNI TLS extension (provided the Ingress controller supports SNI). The TLS secret must contain keys named `tls.crt` and `tls.key` that contain the certificate and private key to use for TLS. For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: testsecret-tls
  namespace: default
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
type: kubernetes.io/tls
```

Referencing this secret in an Ingress tells the Ingress controller to secure the channel from the client to the load balancer using TLS. You need to make sure the TLS secret you created came from a certificate that contains a Common Name (CN), also known as a Fully Qualified Domain Name (FQDN) for `sslexample.foo.com`.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: tls-example-ingress
spec:
  tls:
  - hosts:
    - sslexample.foo.com
    secretName: testsecret-tls
  rules:
    - host: sslexample.foo.com
      http:
        paths:
        - path: /
          backend:
            serviceName: service1
            servicePort: 80
```

**Note:** There is a gap between TLS features supported by various Ingress controllers. Please refer to documentation on nginx, GCE, or any other platform specific Ingress controller to understand how TLS works in your environment.

## Loadbalancing

An Ingress controller is bootstrapped with some load balancing policy settings that it applies to all Ingress, such as the load balancing algorithm, backend weight scheme, and others. More advanced load balancing concepts (e.g. persistent sessions, dynamic weights) are not yet exposed through the Ingress. You can instead get these features through the load balancer used for a Service.

## Updating an Ingress

To update an existing Ingress to add a new Host, you can update it by editing the resource:

```
kubectl describe ingress test
```

```
Name:             test
Namespace:        default
Address:          178.91.123.132
Default backend:  default-http-backend:80 (10.8.2.3:8080)
Rules:
  Host         Path  Backends
  ----         ----  --------
  foo.bar.com
               /foo   service1:80 (10.8.0.90:80)
Annotations:
  nginx.ingress.kubernetes.io/rewrite-target:  /
Events:
  Type     Reason  Age                From                     Message
  ----     ------  ----               ----                     -------
  Normal   ADD     35s                loadbalancer-controller  default/test
```

```
kubectl edit ingress test
```

This pops up an editor with the existing configuration in YAML format. Modify it to include the new Host:

```
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: service1
          servicePort: 80
        path: /foo
  - host: bar.baz.com
    http:
      paths:
      - backend:
          serviceName: service2
          servicePort: 80
        path: /foo
..
```

verify this:

```
kubectl describe ingress test
```

```
Name:             test
Namespace:        default
Address:          178.91.123.132
Default backend:  default-http-backend:80 (10.8.2.3:8080)
Rules:
  Host         Path  Backends
  ----         ----  --------
  foo.bar.com
               /foo   service1:80 (10.8.0.90:80)
  bar.baz.com
               /foo   service2:80 (10.8.0.91:80)
Annotations:
  nginx.ingress.kubernetes.io/rewrite-target:  /
Events:
  Type     Reason  Age                From                     Message
  ----     ------  ----               ----                     -------
  Normal   ADD     45s                loadbalancer-controller  default/test
```

You can achieve the same outcome by invoking `kubectl replace -f` on a modified Ingress YAML file.

## Failing across availability zones

Techniques for spreading traffic across failure domains differs between cloud providers. Please check the documentation of the relevant Ingress controller for details. You can also refer to the federation documentation for details on deploying Ingress in a federated cluster.

## Future Work

Track SIG Network for more details on the evolution of Ingress and related resources. You may also track the Ingress repository for more details on the evolution of various Ingress controllers.

## Alternatives

You can expose a Service in multiple ways that don’t directly involve the Ingress resource:

- Use Service.Type=LoadBalancer
- Use Service.Type=NodePort