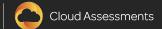


Ingress is an API object that manages external access to the services in a cluster, usually HTTP.

It can provide load balancing, SSL termination and name-based virtual hosting.

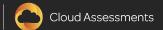
For our purposes, an Edge router is a router that enforces the firewall policy for your cluster.





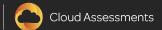
This could be a gateway managed by a cloud provider or a physical piece of hardware.

Our Cluster network is a set of links, either logical or physical, that facilitate communication within a cluster according to the Kubernetes networking model. Examples of a Cluster network include overlays such as Flannel, like we're using in our Linux Academy Cloud server cluster, or SDNs such as OpenVSwitch.





A Service is a Kubernetes Service that identifies a set of pods using label selectors. Unless mentioned otherwise, Services are assumed to have virtual IPs only routable within the cluster network.





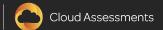
What is Ingress?

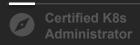
Services and pods have IPs only routable by the cluster network.

So an Ingress is a collection of rules that allow inbound connections

Can be configured to give services externally-reachable URLs, load balance traffic, terminate SSL, offers name-based virtual hosting, and the like.

Users request ingress by POSTing the Ingress resource to the API server. An Ingress controller is responsible for fulfilling the Ingress, usually by way of a load balancer, though it may also configure the edge router or additional front ends to help handle the traffic in a Highly Available manner.

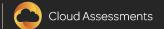




Relatively new resource and not available in any Kubernetes release prior to 1.1. Ingress controller to satisfy an Ingress object

Most cloud providers deploy an ingress controller on the master.

Each ingress pod must be annotated with the appropriate class





```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
name: test-ingress
annotations:
 ingress.kubernetes.io/rewrite-target:/
spec:
- http:
   paths:
   - path: /path
    backend:
     serviceName: test
     servicePort: 80
```







apiVersion: extensions/v1beta1

kind: Ingress

metadata:

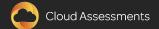
name: test-ingress

spec:

backend:

serviceName: testsvc

servicePort: 80





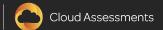
kubectl create -f <filename.yaml>

kubectl get ing

NAME RULE BACKEND ADDRESS

test-ingress - testsvc:80 107.178.254.228

Where 107.178.254.228 is the IP allocated by the Ingress controller to satisfy this Ingress. The RULE column shows that all traffic sent to the IP is directed to the Kubernetes Service listed under BACKEND.

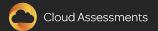




apiVersion: extensions/v1beta1 kind: Ingress metadata: name: test annotations: ingress.kubernetes.io/rewrite-target:/ spec: rules: - host: some.example.com http:

paths: - path: /service1 backend: serviceName: s1 servicePort: 80 - path: /service2 backend: serviceName: s2 servicePort: 80







kubectl get ing

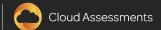
NAME RULE BACKEND ADDRESS

test -

some.example.com

/service1 s1:80

/service2 s2:80





apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: test

spec:

rules:

- host: service1.example.com

http:

paths:

- backend:

serviceName: s1

servicePort: 80

- host: service2.example.com

http:

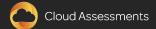
paths:

- backend:

serviceName: s2

servicePort: 80

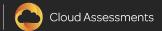






How to secure an Ingress:

- Specify secret.
 - TLS private key
 - Certificate
- Port 443 (Assumes TLS Termination).
- Multiple hosts are multiplexed on the same port by hostnames specified through the SNI TLS extension.
- The TLS secret must contain keys named tls.crt and tls.key that contain the certificate and private key to use for TLS.





apiVersion: v1

data:

tls.crt: base64 encoded cert

tls.key: base64 encoded key

kind: Secret

metadata:

name: supersecret

namespace: default

type: Opaque

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: no-rules-map

spec:

tls:

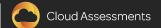
- secretName: supersecret

backend:

serviceName: s1

servicePort: 80







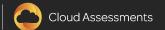
An Ingress controller is bootstrapped with a load balancing policy that it applies to all Ingress objects (e.g. load balancing algorithm, backend weight scheme, etc)

Persistent sessions and dynamic weights not yet exposed

The service load balancer may provide some of this

Health checks are not exposed directly through the Ingress

Readiness probes allow for similar functionality





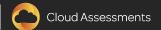
```
kubectl get ing
```

NAME RULE BACKEND ADDRESS

test - 178.91.123.132

services.example.com

/s1 s1:80





kubectl get ing

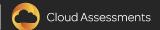
NAME RULE BACKEND ADDRESS

test - 178.91.123.132

services.example.com

/s1 s1:80

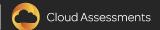
kubectl edit ing test





```
spec:
rules:
- host: services.example.com
 http:
  paths:
  - backend:
     serviceName: s1
     servicePort: 80
    path: /s1
- host: newhost.example.com
```

```
http:
 paths:
 - backend:
   serviceName: s2
   servicePort: 80
  path: /s2
```





```
kubectl get ing
```

NAME RULE BACKEND ADDRESS

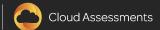
test - 178.91.123.132

services.example.com

/s1 s1:80

newhost.example.com

/s2 s2:80





Alternatively, kubectl replace -f on a modified Ingress yaml file.

This command updates a running Kubernetes object.

Remember that Ingress is a relatively new concept in Kubernetes

Other ways to expose a service that doesn't directly involve the Ingress resource:

- Use Service.Type=LoadBalancer
- Use Service.Type=NodePort
- Use a Port Proxy.

