Managing HTTP(s) traffic in K8s

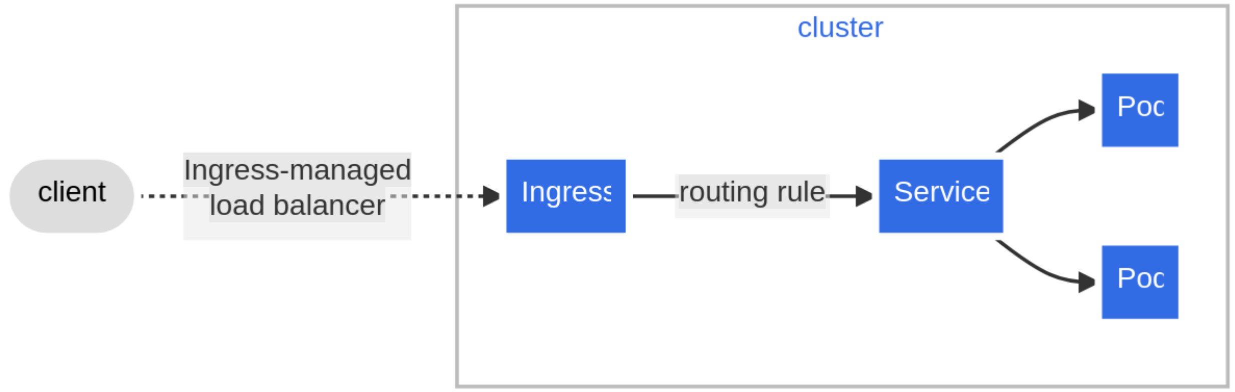
Ingresses and Ingress controllers

Francesco Cappa, Jacopo Marino
Politecnico di Torino

The reason behind Ingress

- Something which goes beyond Service object:
  - An application made up of different Services
  - Need of redirecting traffic to a specific service
  - Need of the same ports (e.g., 80 and 443) with the same IP for several services (public IPs are expensive)
- 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.
- What is an 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.
- 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.

Example of Ingress



- 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.

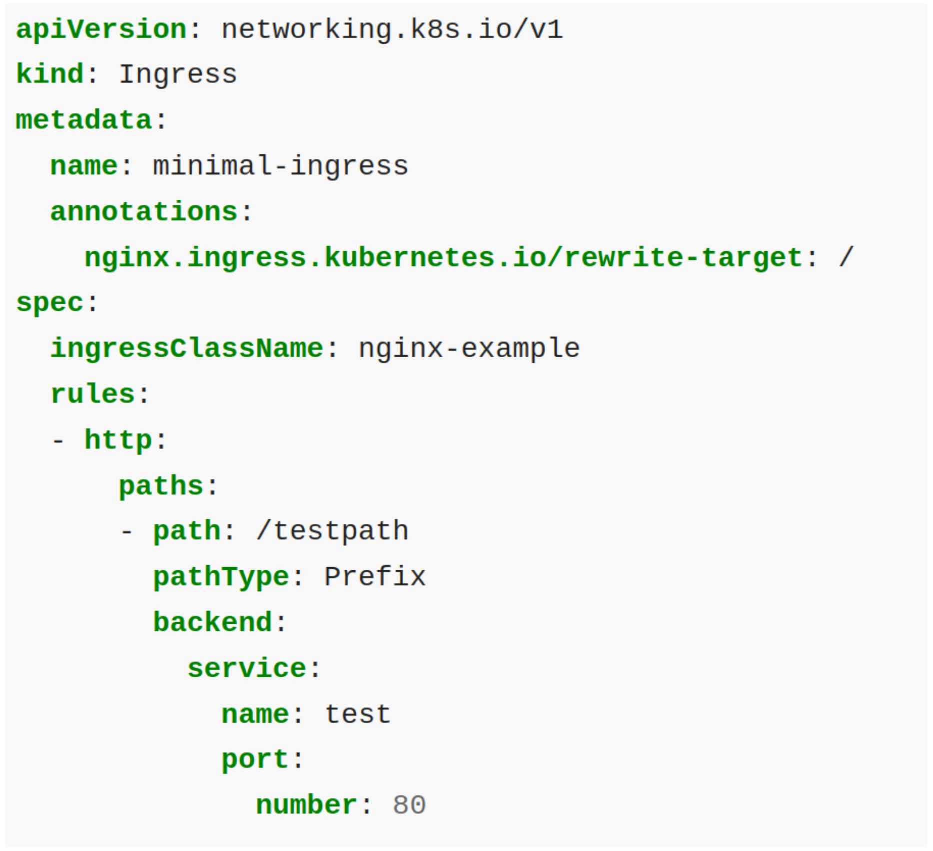
Prerequisites

- You must have an Ingress controller to satisfy an Ingress. Only creating an Ingress resource has no effect.
- Ideally, all Ingress controllers should fit the reference specification. In the reality, the various Ingress controllers operate slightly differently.
- You may need to deploy an Ingress controller such as ingress-nginx. You can choose from a number of Ingress controllers.

Ingress Controllers

- In order for the Ingress resource to work, the cluster must have an ingress controller running.
- Unlike other types of controllers which run as part of the **kube-controller-manager** binary, Ingress controllers are not started automatically with a cluster.
- Kubernetes as a project supports and maintains AWS, GCE, and nginx ingress controllers.
- Using multiple Ingress controllers:
  - You may deploy any number of ingress controllers using ingress class within a cluster.
  - If you do not specify an IngressClass for an Ingress, and your cluster has exactly one IngressClass marked as default, then Kubernetes applies the cluster's default IngressClass to the Ingress.
  - You mark an IngressClass as default by setting the ingressclass.kubernetes.io/is-default-class annotation on that IngressClass, with the string value "true".

The Ingress resource



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

- 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(S) traffic.
- If the ingressClassName is omitted, a default Ingress class should be defined.

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 service.name and a service.port.name or service.port.number. 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 or a custom resource backend by way of a CRD. HTTP (and HTTPS) requests to the Ingress that match the host and path of the rule are sent to the listed backend.
  - A defaultBackend is often configured in an Ingress controller to service any requests that do not match a path in the spec.

Resource backends

- A Resource backend is an ObjectRef to another Kubernetes resource within the same namespace as the Ingress object.
- A Resource is a mutually exclusive setting with Service, and will fail validation if both are specified.
- A common usage for a Resource backend is to ingress data to an object storage backend with static assets.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-resource-backend
spec:
  defaultBackend:
    resource:
      apiGroup: k8s.example.com
      kind: StorageBucket
      name: static-assets
  rules:
    - http:
        paths:
          - path: /icons
            pathType: ImplementationSpecific
            backend:
              resource:
                apiGroup: k8s.example.com
                kind: StorageBucket
                name: icon-assets
```

Path types

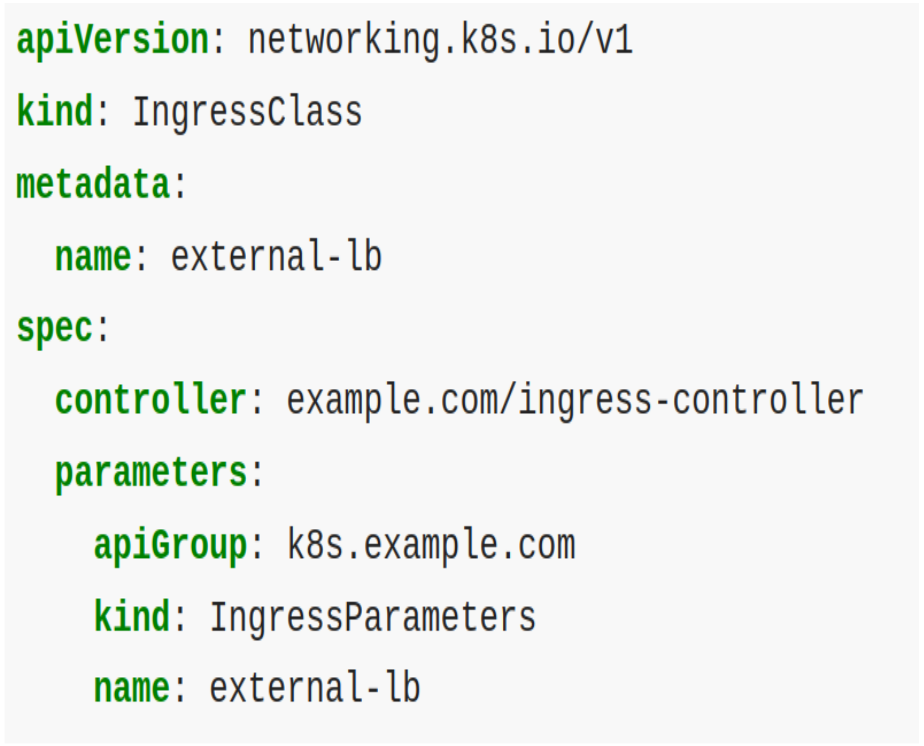
- Each path in an Ingress is required to have a corresponding path type.
- Paths that do not include an explicit pathType will fail validation.
- There are three supported path types:
  1. ImplementationSpecific: up to the Ingress Controller implementation
  2. Exact: Matches the URL path exactly and with case sensitivity.
  3. Prefix: Matches based on a URL path prefix split by /

Hostname wildcards

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-wildcard-host
spec:
  rules:
  - host: "foo.bar.com"
    http:
      paths:
      - pathType: Prefix
        path: "/bar"
        backend:
          service:
            name: service1
            port:
              number: 80
  - host: "*.foo.com"
    http:
      paths:
      - pathType: Prefix
        path: "/foo"
        backend:
          service:
            name: service2
            port:
              number: 80
```

- Hosts can be precise matches (for example “foo.bar.com”) or a wildcard (for example “*.foo.com”).
- Precise matches require that the HTTP host header matches the host field.
- Wildcard matches require the HTTP host header is equal to the suffix of the wildcard rule.

Ingress class



```yaml
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: external-lb
spec:
  controller: example.com/ingress-controller
  parameters:
    apiGroup: k8s.example.com
    kind: IngressParameters
    name: external-lb
```

- 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.
- The .spec.parameters field of an IngressClass lets you reference another resource that provides configuration related to that IngressClass.
- The specific type of parameters to use depends on the ingress controller that you specify in the .spec.controller field of the IngressClass.

IngressClass scope

- Depending on your ingress controller, you may be able to use parameters that you set cluster-wide, or just for one namespace.
- Cluster:
  - The default scope for IngressClass parameters is cluster-wide.
  - The kind (in combination the apiGroup) of the parameters refers to a cluster-scoped API (possibly a custom resource), and the name of the parameters identifies a specific cluster scoped resource for that API.
- Namespaced:
  - If you set the .spec.parameters field and set .spec.parameters.scope to Namespace, then the IngressClass refers to a namespaced-scoped resource.
  - You must also set the namespace field within .spec.parameters to the namespace that contains the parameters you want to use.
  - The kind (in combination the apiGroup) of the parameters refers to a namespaced API (for example: ConfigMap), and the name of the parameters identifies a specific resource in the namespace you specified in namespace.

IngressClass scope (cont'd)

CLUSTER-WIDE

```yaml
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: external-lb-1
spec:
  controller: example.com/ingress-controller
  parameters:
    # The parameters for this IngressClass are specified in a
    # ClusterIngressParameter (API group k8s.example.net) named
    # "external-config-1". This definition tells Kubernetes to
    # look for a cluster-scoped parameter resource.
    scope: Cluster
    apiGroup: k8s.example.net
    kind: ClusterIngressParameter
    name: external-config-1
```

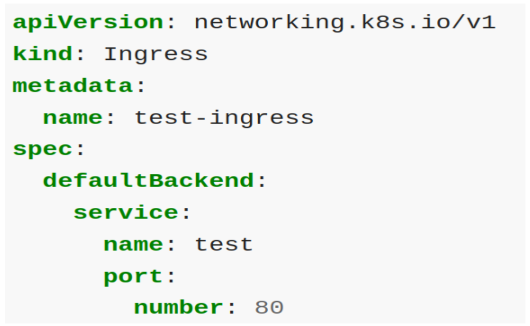
NAMESPACED

```yaml
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: external-lb-2
spec:
  controller: example.com/ingress-controller
  parameters:
    # The parameters for this IngressClass are specified in an
    # IngressParameter (API group k8s.example.com) named "external-
    # that's in the "external-configuration" namespace.
    scope: Namespace
    apiGroup: k8s.example.com
    kind: IngressParameter
    namespace: external-configuration
    name: external-config
```

Namespace-scoped parameters help the cluster operator delegate control over the configuration (for example: load balancer settings, API gateway definition) that is used for a workload.

Types of Ingress

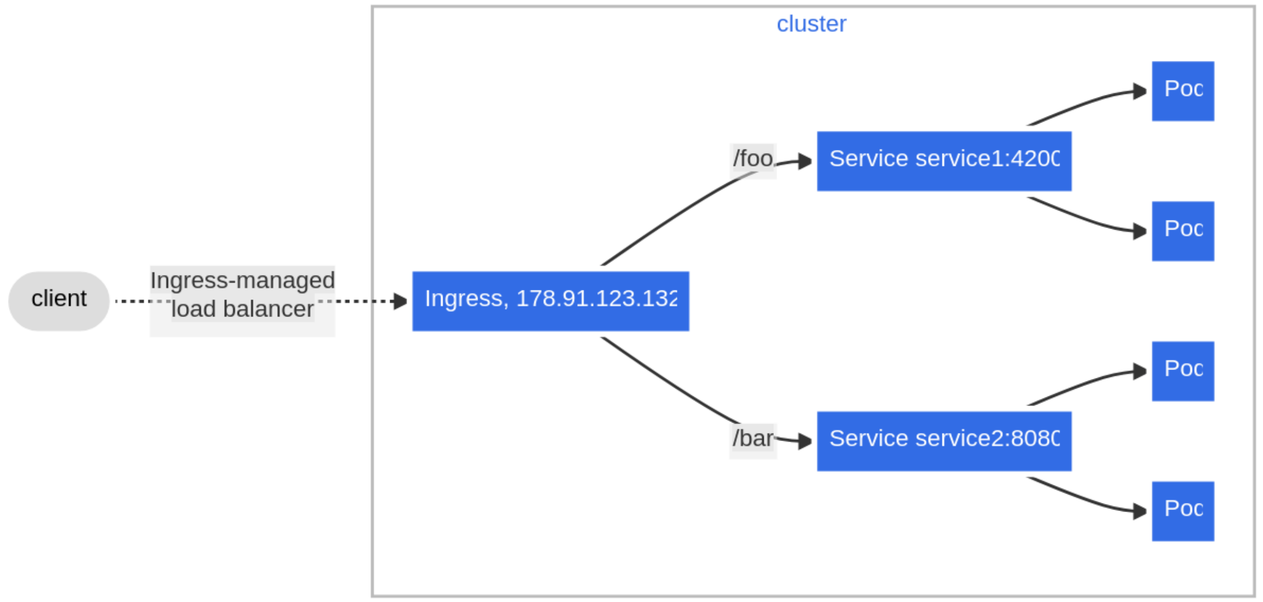
- Ingress backed by a single Service:
  - There are existing Kubernetes concepts that allow you to expose a single Service (see alternatives). You can also do this with an Ingress by specifying a default backend with no rules.



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-ingress
spec:
  defaultBackend:
    service:
      name: test
      port:
        number: 80
```

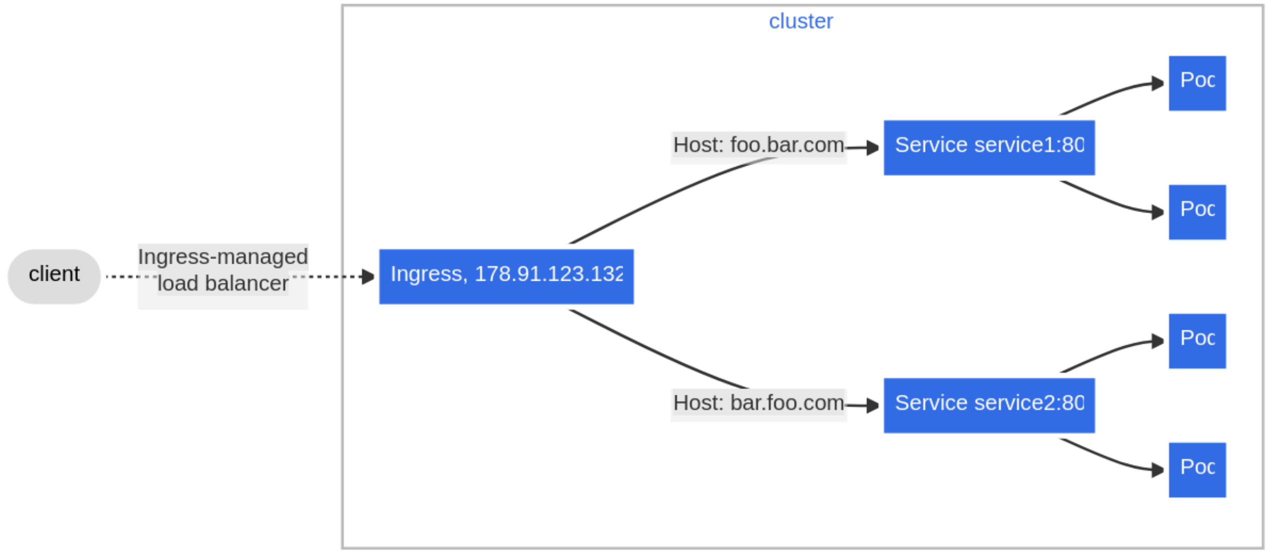
- Simple fanout:
  - A fanout configuration routes traffic from a single IP address to more than one Service, based on the HTTP URI being requested.
  - An Ingress allows you to keep the number of load balancers down to a minimum.
- Name based virtual hosting:
  - Name-based virtual hosts support routing HTTP traffic to multiple host names at the same IP address.
  - 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.

Types of Ingress: Simple fanout



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 4200
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 8080
```

Types of ingress: name based virtual hosting



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: name-virtual-host-ingress
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: service1
            port:
              number: 80
  - host: bar.foo.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: service2
            port:
              number: 80
```

TLS

- You can secure an Ingress by specifying a Secret that contains a TLS private key and certificate.
- The Ingress resource only supports a single TLS port, 443, and assumes TLS termination at the ingress point (traffic to the Service and its Pods is in plaintext).
- The TLS secret must contain keys named tls.crt and tls.key that contain the certificate and private key to use for 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 https-example.foo.com.

```
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
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-example-ingress
spec:
  tls:
  - hosts:
      - https-example.foo.com
    secretName: testsecret-tls
  rules:
```

Load Balancing

- 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.
- It's also worth noting that even though health checks are not exposed directly through the Ingress, there exist parallel concepts in Kubernetes such as readiness probes that allow you to achieve the same end resul

Questions