KUBERNETES INGRESS EXPLAINED

In any Kubernetes cluster, applications must be able to connect with the outside world to provide end-users with access to the application. Kubernetes Ingress is one option available—let's take a look.

(This article is part of our Kubernetes Guide. Use the right-hand menu to navigate.)

What is Kubernetes Ingress?

Kubernetes Ingress is an API object that provides routing rules to manage access to the services within a Kubernetes cluster. This typically uses HTTPS and HTTP protocols to facilitate the routing.

Ingress is the ideal choice for a production environment. Users can expose services within the Kubernetes cluster without having to create multiple load balancers or manually exposing services.

Moreover, K8s Ingress offers a single entry point for the cluster, allowing administrators to manage the applications easily and diagnose any routing issues. This decreases the attack surface of the cluster, automatically increasing the overall security.

Some use cases of Kubernetes Ingress include:

- Providing externally reachable URLs for services
- Load balancing traffic
- Offering name-based virtual hosting
- Terminating SSL (secure sockets layer) or TLS (transport layer security)

Kubernetes Ingress supports multiple protocols, authentication, content-based routing, etc., and

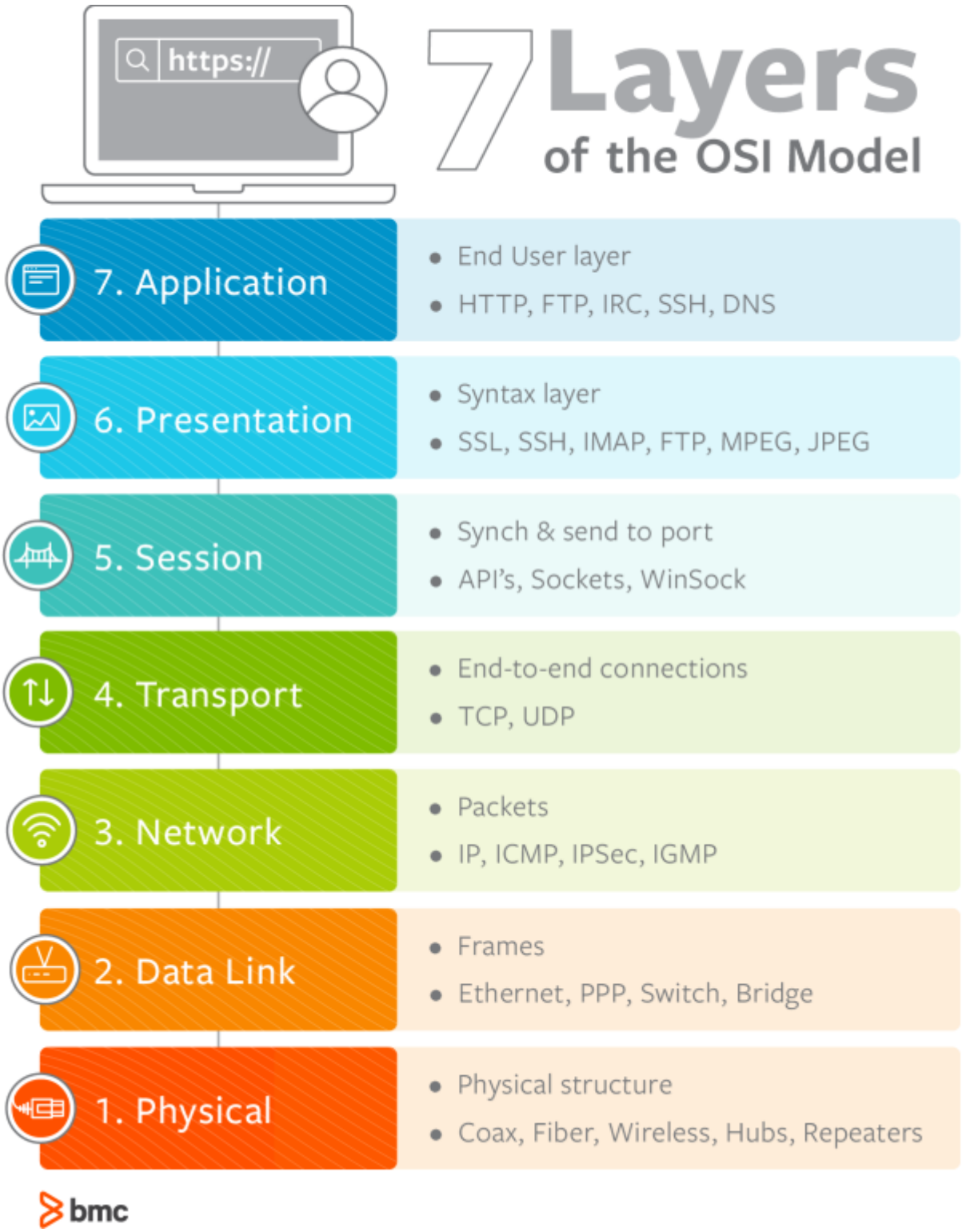
allows users to manage and configure them in Kubernetes clusters easily.

Kubernetes Ingress consists of two core components:

- **Ingress API object.** The API object indicates the services that need to be exposed outside the cluster. It consists of the routing rules.
- **Ingress Controller.** Ingress Controller is the actual implementation of Ingress. It is usually a load balancer that routes traffic from the API to the desired services within the Kubernetes cluster.

What is an Ingress controller?

The actual implementation of Kubernetes Ingress, the Ingress controller is responsible for the traffic routing from external sources to the appropriate services in the Kubernetes cluster. Ingress controllers support routing through both the transport layer (OSI - Layer 4) and the application layer (OSI - Layer 7) in the OSI model.



The application layer routing is preferred over simple transport layer routing because it offers greater control, such as load balancing external traffic based on requests. The Ingress controllers are neither automatically started with a Kubernetes cluster nor a part of the default "kube-controller-manager" binary.

Kubernetes offers built-in support for AWS, GCE, and nginx ingress controllers and also supports integration with third-party ingress controllers. The choice of the Ingress controller depends on:

- Your application requirements
- The host environment

Simply put, the Ingress controller is an application that runs within the Kubernetes cluster and

provisions a load balancer according to the requirements of Ingress. (The load balancer can be a software load balancer, external hardware load balancer, or cloud load balancer, each type requiring different controller implementations.)

Ingress vs alternatives for exposing services

Using Ingress is not the only way to expose services.

A Kubernetes service resource provides an abstract way to expose a set of targeted application pods as a network service. At the same time, Kubernetes uses ClusterIP, NodePort, and Load Balancer methods to expose services both internally within nodes and externally outside the cluster.

- **ClusterIP** is the default service type that exposes services using an internal cluster IP. This method exposes only the service within the cluster and is unreachable externally.
- **NodePort** exposes the service using a static port associated with the desired node. NodePort configuration will automatically create the ClusterIP to route the traffic internally. Users can access the service externally by using the IP address of the node and the exposed port.
- **LoadBalancer** exposes the service using the cloud provider's load balancer and automatically creates the necessary NodePorts and ClusterIPs to route the desired traffic.

(Learn about NodePort, Port & TargetPort.)

Individually exposing and managing services is inefficient and not scalable in large Kubernetes clusters. That's how Ingress becomes the ideal solution—by acting as the entry point for the whole Kubernetes cluster. This enables users to:

- Manage all the routing rules from a centralized resource
- Expose multiple services under a single IP address

Creating an Ingress resource

Now let's have a look at how to create an Ingress resource.

As with any other resource in Kubernetes, Ingress resource also consists of apiVersion, kind, and metadata fields with the desired rules and configurations under the spec section.

Simple Ingress configuration

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/app-root: /test_app
  name: testapproot
  namespace: default
spec:
  rules:
  - host: testapproot.mycluster.com
    http:
      paths:
```

```
    - backend:
        serviceName: http-svc
        servicePort: 80
      path: /
```

When defining an ingress resource, the name of the ingress object needs to adhere to valid subdomain DNS naming conventions. Furthermore, annotations can be used to configure ingress resources with specific options. In the above YAML file, we use the "nginx.ingress.kubernetes.io/app-root" option to define the application root that the ingress controller needs to redirect when the root path is called (path: /).

Different types of ingress controls will include different annotation support, and the user needs to modify the annotations to suit the desired controller.

INGRESS RULES

Ingress rules are the core of Kubernetes Ingress. They define what happens to the incoming traffic and to which services that traffic should be directed. Each HTTP rule consists of multiple fields to define the rule properly.

- **Host** is an optional parameter that defines which host the rule gets applied. When the host parameter is not defined, the rule is applied to all incoming HTTP traffic.
- **Paths** refers to the list of paths that are associated with a backend. Both the host and path should match to route the traffic to the desired service properly.
- **Backend** is **t**he destination to which the incoming traffic should be ultimately directed. This can be a Kubernetes service or a custom resource backend to indicate another resource under the same namespace (e.g., storage bucket).

Any request that does not match any of the defined rules will be directed to the default backend. This default backend is a part of the ingress controller itself and not specified in the ingress resource.

Defining multiple Ingress rules

Now that we have a better understanding of the ingress resource, we can see how to define multiple ingress rules with different hosts and backends.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: myapp
spec:
  rules:
  - host: "app.mycluster.com"
    http:
      paths:
      - pathType: Prefix
        path: "/home"
        backend:
```

```
          serviceName: http-app
          servicePort: 80
      - pathType: Prefix
        path: "/admin"
        backend:
          serviceName: http-app
          servicePort: 8080
  - host: "data.mycluster.com"
    http:
      paths:
      - pathType: Prefix
        path: "/charts"
        backend:
          serviceName: http-charts
          servicePort: 81
```

In the above configuration, we have configured three routing rules for two different hosts that consume different services.

host: "app.mycluster.com"

This section consists of two rules that only apply to traffic directed to app.mycluster.com. If the URL path matches either /home or /admin, the traffic will get directed to different ports of the same service.

host: "data.mycluster.com"

The single rule configured will apply to traffic directed at data.mycluster.com host, and if the URL path matches /charts, it will direct the traffic to port 81 of the "http-charts" service.

Kubernetes Ingress path types

When defining rules, the path must include a corresponding path type to validate the rules properly. Kubernetes has three path types:

- This matches the configurations of an IngressClass defined in an Ingress Controller. It will be treated as a separate path type or the same as Prefix or Exact types, depending on the implementation of the class.
- We can use this type when the user needs to match a URL path exactly with case sensitivity.
- This will match the URL based on the path prefix split by the forward-slash (/). This is also case sensitive, and the matching process is done from element to element in the URL.

Let's now look at some path matching scenarios.

Path Type	Path	Sample Request	Match
Exact	/test	www.example.com/test	Yes
Exact	/test	www.example.com/test/	No
Prefix	/test	www.example.com/test www.example.com/test/	Yes (Both URLs)

Prefix	/test/user	www.example.com/test/us	No

In instances where multiple matches are available for a request, the request with the longest matching path will be given preference. If both paths are an exact match, the ingress controller will select the exact path type over the prefix to apply the necessary ingress rule.

(Dive deeper into all the matching scenarios in the official Kubernetes documentation.)

Types of Ingress

There are three types of Ingress in Kubernetes;

- **Single service.** This is Ingress backed by a single service where a single service is exposed. To define this, specify a default backend without any rules.
- **Simple fanout.** The fanout type is where a single entry point (single IP address) is used to expose multiple services. There, URL traffic gets routed to the desired destination depending on the request. This method allows users to reduce the number of load balancers and easily manage the routing within the cluster.
- **Name-based virtual hosting.** This method can be used to route traffic from a single entry (IP address) to different hosts within the cluster. Here, the traffic is first targeted towards a specified host before any other routing is done.

Users can secure an ingress by specifying a secret with a supported TLS key and a certificate. The ingress resource supports a single TLS port (443) and assumes TLS termination at the ingress point. Then the internal communications happen via plain text.

Manage services better, with improved security

Kubernetes Ingress provides a unified resource to manage and route traffic to your applications in a Kubernetes cluster without having to manage services and load balances manually.

Related reading

- BMC DevOps Blog
- Kubernetes Best Practices for Enhanced Cluster Efficiency
- How Containers & Kubernetes Work Together
- Kubernetes vs Docker Swarm: Comparing Container Orchestration Tools
- Docker Security: 14 Best Practices for Securing Docker Containers
- The State of Containers Today: A Report Summary