What Is An Ingress Controller?

Ingress Controller

An Introduction to Ingress Controller

Kubernetes ingress is an object with rules for routing and controlling the ways that external users access services running in a Kubernetes cluster. You can expose applications in Kubernetes to external users taking one of three basic approaches:

- A NodePort type of Kubernetes service exposes the application on a port across each node
- A Load Balancer Kubernetes service points users to Kubernetes services in the cluster
- A Kubernetes Ingress Resource and Ingress controller can together expose the application

NodePort

Each cluster node has an open NodePort which exposes the service on that Node's IP. Kubernetes routes incoming traffic on the NodePort to services, and is the most basic way to provide access.

Users running in Google Cloud and other public cloud providers may have to edit firewall rules to make the system functional, but every Kubernetes cluster supports the basic NodePort functions.

However, if the port isn't specified, Kubernetes will choose it at random, which is not always advantageous. It is less convenient that the system generally assigns the value of any NodePort randomly from a pool of cluster-configured NodePort ranges between 30000 and 32767. This range stays safely non-standard and out of the way of well-known ports, and for most UDP or TCP clients it is not an issue.

But compared to the typical ports 80 and 443 for HTTP and HTTPS, respectively, these NodePort values mean HTTP or HTTPS traffic will be exposed on a non-standard port. In addition, particularly when the system sets a unique, random port for every service, not knowing these random values in advance is its own challenge, which in turn makes configuring firewall rules, NAT, etc. more difficult.

The NodePort is a handy abstraction for situations when you don't need a production-level URL, such as during development. It is intended as a building block for higher-order ingress models such as load balancers.

Load Balancer

The Load Balancer is another option. An external load balancer is deployed automatically when the load balancer service type is in use. This external load balancer routes external traffic to a Kubernetes service in your cluster and is associated with a specific IP address.

That said, it only works if you are operating in a cloud-hosted environment; not all cloud providers support the load balancer service type; and the load balancer's exact implementation relies upon the cloud provider. Moreover, it's necessary to supply load balancer implementation to deploy Kubernetes on bare metal. Perhaps least advantageous: for every service with this type, a hosted load balancer along with a new public IP address is spun up, which adds costs.

However, the load balancer service type is often the simplest, safest way to route traffic in environments that support it.

Ingress Controllers and Ingress Resources

Kubernetes supports Ingress, a high level abstraction which enables simple URL or host based HTTP routing. In Kubernetes, the Ingress resource is the official means of exposing HTTP-based services.

Although it remains in beta, an ingress is a core Kubernetes concept. Nevertheless, an ingress is always implemented by an ingress controller, a third party proxy responsible for reading and processing Ingress Resource information. Various ingress controllers support additional, distinct use cases by extending the specification in unique ways.

The Kubernetes cluster must have a running ingress controller for the Ingress resource to function. However, although the ingress controller provides additional control and routing behind an external load balancer, it does not typically replace it.

The Kubernetes cluster does not start ingress controllers automatically, in contrast to other kube-controller-manager binary varieties of controllers. The Kubernetes project maintains and supports GCE, AWS ALB ingress controllers, and NGINX ingress controllers, but there are many other options. Most, such as the AWS ingress controller, are open source.

Various facts bolster the idea that the Ingress resource is less well designed than other Kubernetes resources. For example, the Ingress resource has existed since version 1.1, for the past 18 Kubernetes versions, as a beta resource. Furthermore, there have been many modifications and extensions to Ingress resource behavior from Ingress-controller-specific annotations over the years such as Kubernetes ingress cache control annotation.

Scaling is a particular problem of the ingress resource. For instance, the ingress resource combines three key issues into one resource definition:

- Identity or domain name
- Authentication or TLS certificate
- Routing or which URL paths and Kubernetes services are routed to each other

For optimal results managing a more complex site where multiple independent work groups manage components, split and delegate those key issues to different roles:

Infrastructure/Security manages identity or domain names plus authentication and TLS certificates

Site admin covers application/component routing to individual management teams

Application teams manage routing within versions of applications, testing cycles, etc.

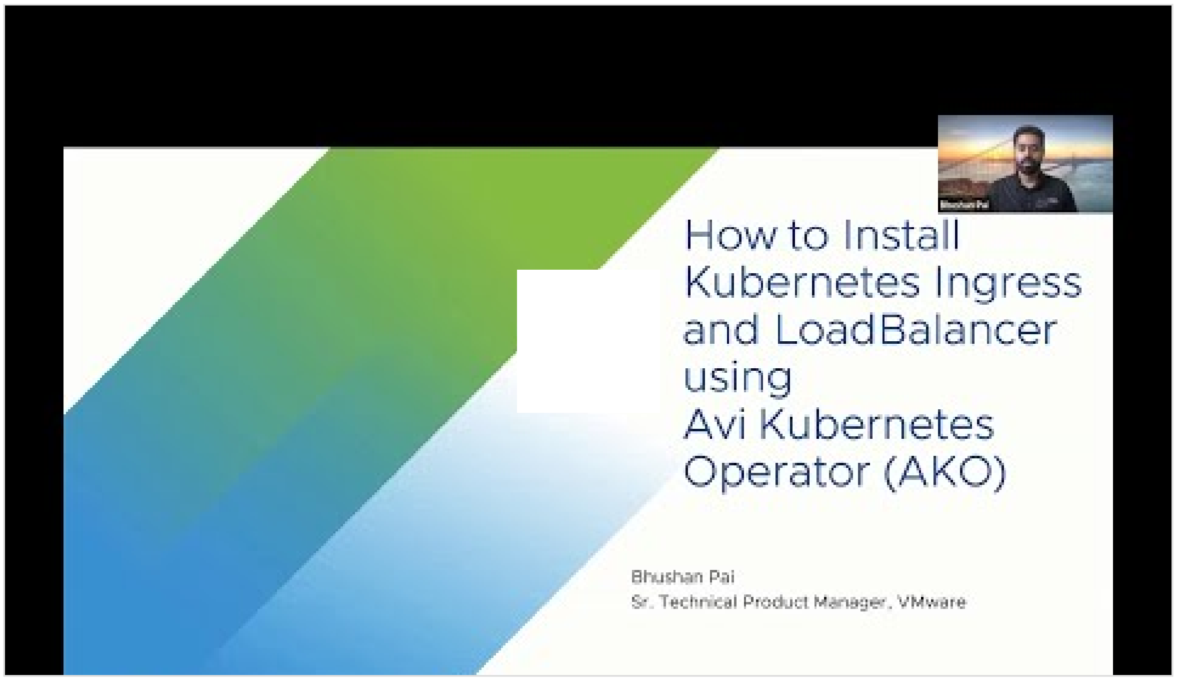
If the application team wants to run a blue/green test, for example, this is complicated by the single resource definition problem. The Kubernetes ingress resource defines the routing to Kubernetes services, TLS certificate, and domain name within a single object.

This means that the team will require access to the global ingress resource for the whole site to modify it just to conduct their testing, which has implications for both stability and security. For instance, the whole site will be inaccessible should the application team introduce a syntax error into the ingress resource.

Some ingress controllers support a multi-role setup and empower simpler scaling in Kubernetes.

For a typical configuration of Kubernetes ingress controller architecture, see the Kubernetes ingress controller diagram below.

For more on the actual implementation of Kubernetes Ingress, check out our Application Delivery How-To Videos or watch the Kubernetes Ingress and Load Balancer How To Video here:



How to Install Kubernetes Ingress and LoadBalancer using Avi Kubernetes Operator (AKO)

Bhushan Pai
Sr. Technical Product Manager, VMware

What Is an Ingress Controller?

In Kubernetes, as pods are created, selector labels are assigned to them. Then, it's typical to group them under a Service, rather than making the pods directly accessible. This makes the pods available, although only from within the same cluster, at a single cluster IP address.

The pods can be replaced or scaled up or down at any time. The Service hides the ephemeral nature of the pods, acting both as a layer of abstraction and performing very basic round-robin load balancing.

However, a Service is accessible only to nearby pods inside the cluster. This presented a new challenge for Kubernetes operators: granting clients outside the cluster access to services.

Allocating an external load balancer or random port is relatively simple to implement, but presents unique challenges. Both defining multiple NodePort services and Load Balancer services can lead to increased use of resources and too much complexity. The next idea was creating a new layer of abstraction that might contain or reduce this complexity so that many internal services could be exposed via one load balancer or random port.

The result was the Ingress, a single entrypoint behind which many services could be consolidated. The Ingress itself declares the user wants clients to be routed to services. The Ingress rule or manifest doesn't do anything itself; to watch for these declarations and act upon them users must deploy ingress controllers.

An Ingress is a rule that governs how a service securely inside the Kubernetes cluster can safely access the outside world to allow user access. An ingress controller, a proxy, sits at the edge of the cluster's network waiting for new rules. It processes them and maps each service to a specific domain name or URL path for public use. It is the Kubernetes project itself that develops and maintains the Ingress, but other open source projects develop ingress controllers, implement them, and create unique features.

Just like other applications, ingress controllers are pods, visible parts of the cluster. Ingress controllers are built using underlying reverse proxies that lend them load balancing and Layer 7 routing capabilities. Each proxy is its own product with a unique set of features.

Ingress controller deployments are themselves inside the cluster, walled-in like other Kubernetes pods. A service with a type of either load balancer or NodePort is required to expose ingress controllers to the outside. However, now many internal pods connect to one ingress controller, which itself connects to one

Service: a single entrypoint for all traffic. The ingress controller inspects HTTP requests, and identifies the correct pod for each client based on the domain name, the URL path, or other characteristics it detects.

The ingress controller is tasked with fulfillment based on the declarations in the ingress. This declarative aspect of the ingress manifest allows users to specify goals and needs without needing to hash out those fulfillment specifics. As new Ingress rules are issued, the ingress controller identifies them and the corresponding routes, and configures its underlying proxy in response.

After the ingress controller install and the definition of ingress manifests, there isn't much to managing the controller. The ingress controller instantly wires up the manifests once they are defined distinct from the service they refer to, and the controller works quietly in the background, managing when the public may access the service.

In Kubernetes environments, an ingress controller is a kind of specialized load balancer. For managing containerized applications, Kubernetes has become the de facto standard, but moving production workloads into Kubernetes creates application traffic management additional complexities for many businesses. An ingress controller serves as a bridge between Kubernetes and external services by abstracting the complexity of Kubernetes application traffic routing away.

Kubernetes ingress controllers basics:

- Continuously monitor the Kubernetes pods, and as pods are added or removed from a service, automatically update load balancing rules
- Accept outside traffic and load balance it to containers running inside the Kubernetes platform
- Manage in-cluster egress traffic for services which need to communicate with outside services
- Use the Kubernetes API to deploy Ingress resources and configuration files, and the kubectl apply and other syntax tools to run basic commands

Ingress controllers help route external traffic to Kubernetes clusters. But in practice, except for all but the simplest cloud applications, Kubernetes services typically also impose other requirements on ingress. For example:

Security needs such as authentication and access control, TLS termination, and allowlist/denylist

Need for traffic management and service discovery, including content-based routing, such as routing based on request headers, namespace ingress, HTTP method, or other specific request properties

Resilience challenges, such as DOS detection and mitigation, timeouts and rate limiting

Multiple protocols in need of support

Demand for observability and insights

To manage these concerns at the service level inside Kubernetes, many Kubernetes ingress controller options exist and there are several approaches. Any Kubernetes ingress controller comparison necessarily should consider supported protocols, traffic routing capabilities, underlying software, upstream probes, namespace limitations, traffic distribution, authentication, load balancing algorithms, WAF capabilities, customizability, and other features. This is why in Kubernetes multiple ingress controllers exist.

When choosing an ingress controller, in most situations it pays to begin with an external load balancer, regardless of ingress strategy. This load balancer provides an IP address—a stable endpoint—for external traffic to access. The load balancer then routes traffic to an ingress or Kubernetes service on the Kubernetes cluster to conduct service-specific routing.

NodePorts are not designed to be directly used for production, and both Kubernetes services and ingress controllers require an external load balancer.

Ingress Controller vs Service

This is the same contrast between Ingress controller vs API gateway. Ingress strategies center on service-specific ingress management—selecting the right options for managing traffic between services and the external load balancer. Businesses can develop and deploy their own custom ingress controller or other configurations, but most instead choose either a standard ingress controller or an API gateway deployed as a Kubernetes service.

The choice focuses on actual capabilities. Because the Ingress resource is an imprecise, less than portable standard trained on basic functionality, many Kubernetes ingress controller options have extended the Ingress resource with custom annotations.

Ingress Controller vs Load Balancer

An ingress is merely the set of protocols or rules for ingress to be deployed. Without either an ingress controller or a load balancer service that is configured to listen for and process these ingress rules, nothing will happen after deploying them.

In other words, a load balancer, like an ingress controller, can be configured to process and act on ingress rules, enabling ingress to function. However, unlike a reverse proxy or API gateway, which routes requests to specific backend services based on particular criteria, a load balancer distributes requests among multiple backend services of the same type.

Ingress Controller vs Service Mesh

Adoption of service mesh has become more mainstream as organizations heighten investment into containerized apps and microservices. According to the Cloud Native Computing Foundation's 2020 survey: adoption of service mesh is rising rapidly; increased container use indicates more organizations need advanced security tools and traffic management, and might benefit from a service mesh; and three of the top container challenges are interrelated.

However, implementing a service mesh or other advanced API management solution too soon simply adds expense and risk that outweigh any benefits. When it comes to the ingress controller vs service mesh vs API gateway comparison, consider these points:

> How invested the user is in Kubernetes. Whether the user has moved the production environment into Kubernetes already or is just starting to test migrating apps to container workloads, consider whether the long-term roadmap for application management includes Kubernetes.

User need for mutual TLS (mTLS) and a zero-trust production environment between services. Some users will be forced to increase their service-level security, while others need to maintain the zero-trust level of security they already rely on for production apps in the containerized environment.

Maturity of the CI/CD pipeline. Mature deployments include programmatic procedural use of Kubernetes apps and Kubernetes infrastructure.

Complexity in depth and number of services. Large, distributed apps with many API dependencies typically demand external dependencies.

Frequent deployment to production. This means daily, typically, with near-constant updates of apps in production.

DevOps team can manage service mesh. Because DevOps teams often handle administration within the cluster, even if the NetOps team will manage the service mesh, DevOps must also be ready to handle the service mesh in addition to the rest of their stack.

AVI Networks Now part of VMware

WEBINAR

Get to Production-Ready Kubernetes with Ingress Controller, Load Balancing, and Security from VMware NSX Advanced Load Balancer (Avi Networks)

Watch this webinar to learn how to modernize your applications and infrastructure -- provide scalable load balancer and container ingress services

WATCH HERE

Ingress Controller Troubleshooting

There are many ways you can troubleshoot the ingress controller, one of which involves the ingress controller logs. In order to determine the ingress controller logs are good, it says it is listening on port 3000. It's important to make sure your applications are up and running, and if they aren't then it's time to make your logs a little more expansive. Once you can confirm the logs are good and the applications are running smoothly, then you can check on the rest of the traffic flow for other errors.

Ingress Controller Performance

Ingress controllers hold a lot of power to improve performance and enhance security for Kubernetes clusters, as long as they're deployed and configured correctly. One of the benefits of ingress controllers is that they can handle many of the capabilities other tools provide, saving you time and money. Other solutions like load balancers and application delivery controllers (ADCs) need the extra work to adapt to the workings of Kubernetes, so it's best to use ingress controllers that are already designed for Kubernetes. Its versatility is why you can depend on its performance for the future of Kubernetes.

Ingress Security

Ingress security is a crucial part to Kubernetes and being able to secure it gives you the power to use its full potential. A part of securing the Kubernetes application is provisioning TLS within the Ingress resource itself, which is why Ingress plays such an important role in Kubernetes application security. In addition to securing Ingress, there are other forms like network policies which identifies and secures how pods communicate with each other, taking the application-centric approach.

Does Avi Offer a Kubernetes Ingress Controller?

A typical Kubernetes deployment presents numerous challenges, including:

- Multiple discrete products and solutions deployed at once;
- A lack of observability and analytics;
- Complex, tough to manage operations; and
- Only partial ability to scale.

Delivering an ingress gateway to applications based on microservices architecture demands a modern, distributed application services platform. Traditional appliance-based ADC solutions are no longer up to the demands presented by web-scale, cloud-native applications deployed using container technology as microservices.

Each Kubernetes pod may hold thousands of containers, and a Kubernetes container cluster can have hundreds of pods. Full functionality in Kubernetes requires elastic container services designed for K8s, policy driven deployments, and mandating full automation.

Avi Vantage is based on a software-defined, scale-out architecture that provides container services for Kubernetes beyond typical Kubernetes controllers, such as traffic management, security, observability and a rich set of tools to simplify application maintenance and rollouts.

Avi's ingress service solution meets each of these common Kubernetes deployment challenges:

- An integrated solution – delivering comprehensive load balancing, ingress, intrinsic security, WAF, GSLB, DNS, and IPAM
- Operational simplicity – easier troubleshooting from a single solution with central control
- Rich observability – real-time telemetry with application insights across all components
- Cloud-native automation with elasticity – closed-loop analytics and decision automation deliver elastic autoscaling

The Avi Controller is a central plane for management, control, and analytics that communicates with the Kubernetes controller, configures services, deploys and manages the lifecycle of data plane proxies, and aggregates telemetry analytics from the Avi Service Engines.

The Avi Service Engine is a service proxy providing various dataplane ingress services, such as WAF, load balancing, IPAM/DNS, and GSLB. As mentioned, the Avi Service Engine also reports real-time telemetry analytics to the Avi Controller.

Avi Networks provides containerized applications running in Kubernetes environments with a centrally orchestrated, elastic proxy services fabric for analytics, dynamic load balancing, micro-segmentation, security, service discovery, and more. Learn more about the Avi Networks elastic Kubernetes ingress controller and services here.

WHITE PAPER–JANUARY 2021
Deliver Elastic Kubernetes Ingress Controller and Services
A single platform for consolidated traffic management, security, and observability
vmware

Read this white paper to learn how Avi delivers a service mesh and an application services fabric for elastic Kubernetes clusters

DOWNLOAD HERE