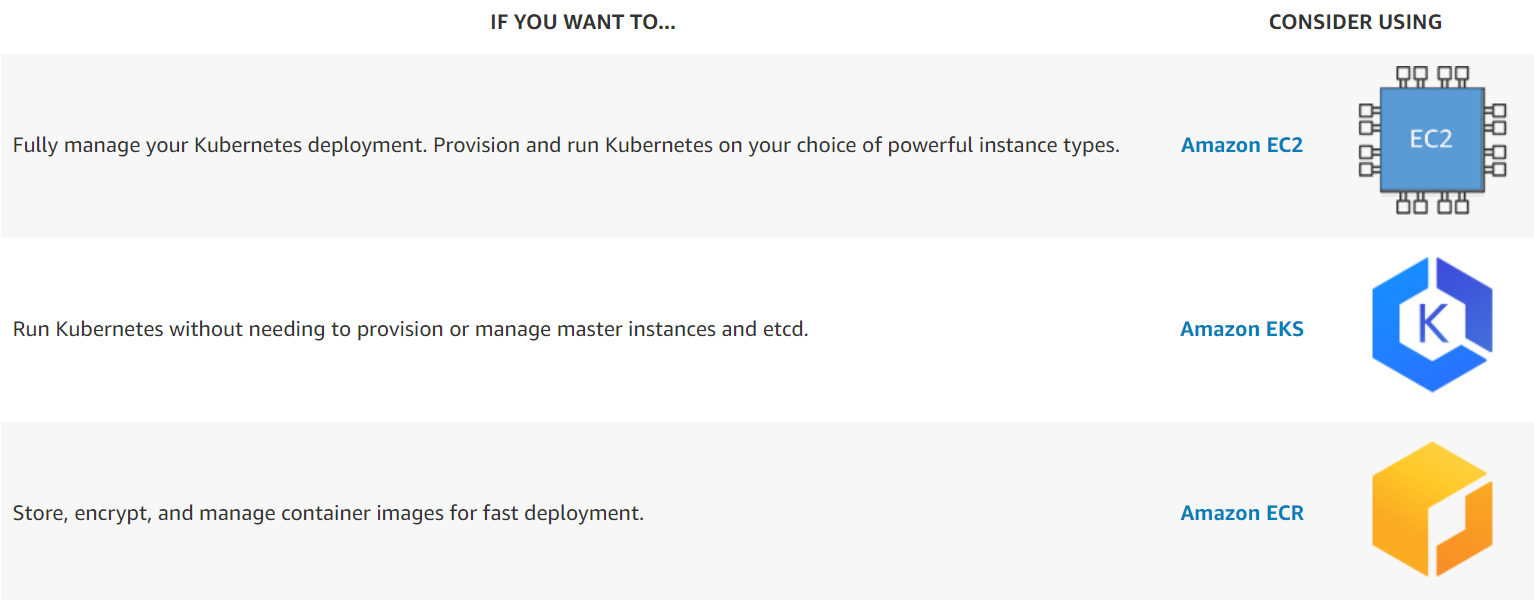
EKS (Amazon Elastic Kubernetes services)

Kubernetes is open source software that allows you to deploy and manage containerized applications at scale. Kubernetes manages clusters of Amazon EC2 compute instances and runs [containers](https://aws.amazon.com/what-are-containers/) on those instances with processes for deployment, maintenance, and scaling. Using Kubernetes, you can run any type of containerized applications using the same toolset on-premises and in the cloud.

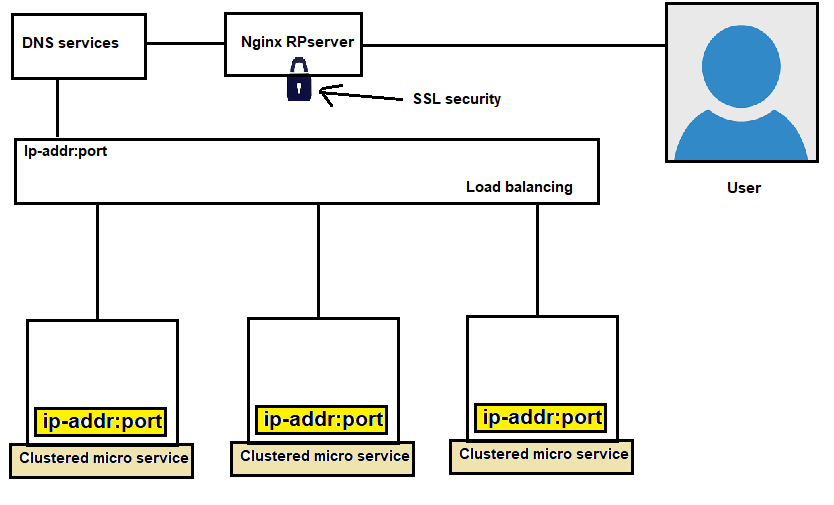


Similar features would be provided by google and azure.(GCE- Google Kubernetes engine, and azure Kubernetes services)

Ingress network:

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.



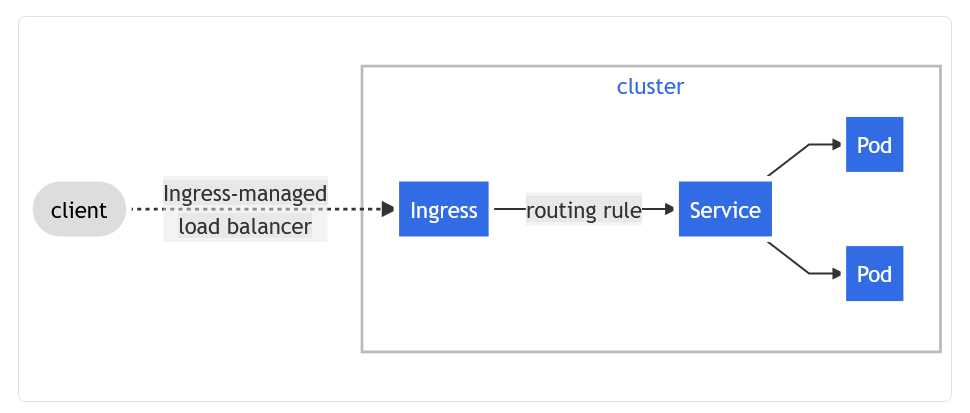
## Terminology

For clarity, this guide defines the following terms:

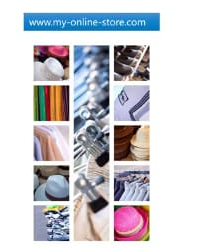
* Node: A worker machine in Kubernetes, part of a cluster.
* Cluster: A set of Nodes that run containerized applications managed by Kubernetes. For this example, and in most common Kubernetes deployments, nodes in the cluster are not part of the public internet.
* Edge router: A router that enforces 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](https://kubernetes.io/docs/concepts/cluster-administration/networking/).
* Service: A Kubernetes [Service](https://kubernetes.io/docs/concepts/services-networking/service/) that identifies a set of Pods using [label](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels) selectors. Unless mentioned otherwise, Services are assumed to have virtual IPs only routable within the cluster network.

## What is Ingress?

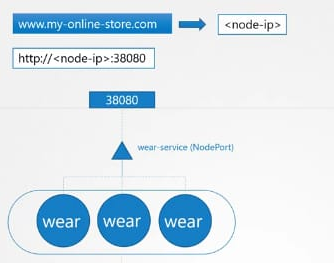
[Ingress](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.21/#ingress-v1-networking-k8s-io) exposes HTTP and HTTPS routes from outside the cluster to [services](https://kubernetes.io/docs/concepts/services-networking/service/) within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.



Suppose you are having online e-commerce store, and you have implemented to by Kubernetes.



Here, you have used cluster-ip, node-port and DNS services. As you are using node port, so you can access that website by node-ip and port.



Now, your side is growing, and simultaneously you have launched multiple other products as well.

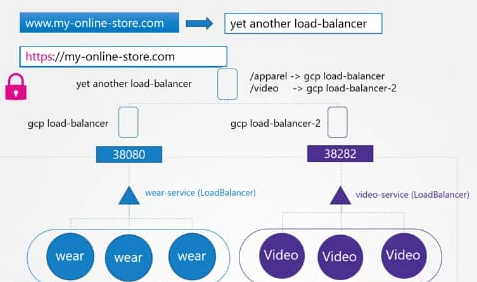
Here, by using cluster-ip, node-port and load balancer service, you can access your website outside, but still there are many more things such as SSL, and security thing.

DNS: The **Domain Name System** (**DNS**) is a [hierarchical](https://en.wikipedia.org/wiki/Hierarchy) and [decentralized](https://en.wikipedia.org/wiki/Decentralised_system) naming system for computers, services, or other resources connected to the [Internet](https://en.wikipedia.org/wiki/Internet) or a private network. It associates various information with [domain names](https://en.wikipedia.org/wiki/Domain_name) assigned to each of the participating entities. Most prominently, it translates more readily memorized domain names to the numerical [IP addresses](https://en.wikipedia.org/wiki/IP_address) needed for locating and identifying computer services and devices with the underlying [network protocols](https://en.wikipedia.org/wiki/Communication_protocol). By providing a worldwide, [distributed](https://en.wikipedia.org/wiki/Distributed_computing) [directory service](https://en.wikipedia.org/wiki/Directory_service), the Domain Name System has been an essential component of the functionality of the Internet since 1985.

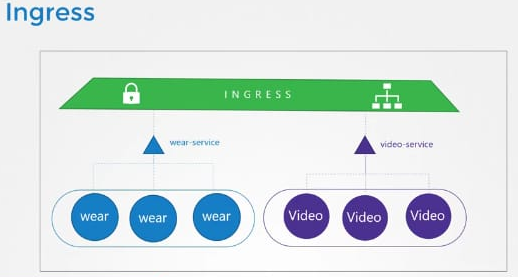
/etc/hosts >> Here we add entry for host whom we want to communicate.

/etc/resolve.conf >> instead of adding multiple entry, lets do it in centralized place.

During communication first it check entry on /etc/hosts then /etc/resolve.conf



Instead of manual managing these things, we can opt ingress network concept.





**FEATURE STATE:** Kubernetes v1.19

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](https://kubernetes.io/docs/concepts/cluster-administration/networking/).
* Service: A Kubernetes [Service](https://kubernetes.io/docs/concepts/services-networking/service/) that identifies a set of Pods using [label](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels) selectors. Unless mentioned otherwise, Services are assumed to have virtual IPs only routable within the cluster network.

What is Ingress?

[Ingress](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.20/#ingress-v1-networking-k8s-io) exposes HTTP and HTTPS routes from outside the cluster to [services](https://kubernetes.io/docs/concepts/services-networking/service/) within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

Here is a simple example where an Ingress sends all its traffic to one Service:

cluster

Ingress-managed  
load balancer

routing rule

Ingress

Pod

Service

Pod

client

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](https://kubernetes.io/docs/concepts/services-networking/ingress-controllers) 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](https://kubernetes.io/docs/concepts/services-networking/service/" \l "nodeport) or [Service.Type=LoadBalancer](https://kubernetes.io/docs/concepts/services-networking/service/" \l "loadbalancer).

Prerequisites

You must have an [Ingress controller](https://kubernetes.io/docs/concepts/services-networking/ingress-controllers) to satisfy an Ingress. Only creating an Ingress resource has no effect.

You may need to deploy an Ingress controller such as [ingress-nginx](https://kubernetes.github.io/ingress-nginx/deploy/). You can choose from a number of [Ingress controllers](https://kubernetes.io/docs/concepts/services-networking/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 resources

A minimal Ingress resource example: [**service/networking/minimal-ingress.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/service/networking/minimal-ingress.yaml)

**apiVersion**: networking.k8s.io/v1

**kind**: Ingress

**metadata**:

**name**: minimal-ingress

**annotations**:

**nginx.ingress.kubernetes.io/rewrite-target**: /

**spec**:

**rules**:

- **http**:

**paths**:

- **path**: /testpath

**pathType**: Prefix

**backend**:

**service**:

**name**: test

**port**:

**number**: 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](https://kubernetes.io/docs/concepts/overview/working-with-objects/names#dns-subdomain-names). For general information about working with config files, see [deploying applications](https://kubernetes.io/docs/tasks/run-application/run-stateless-application-deployment/), [configuring containers](https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/), [managing resources](https://kubernetes.io/docs/concepts/cluster-administration/manage-deployment/). Ingress frequently uses annotations to configure some options depending on the Ingress controller, an example of which is the [rewrite-target annotation](https://github.com/kubernetes/ingress-nginx/blob/master/docs/examples/rewrite/README.md). Different [Ingress controller](https://kubernetes.io/docs/concepts/services-networking/ingress-controllers) support different annotations. Review the documentation for your choice of Ingress controller to learn which annotations are supported.

The Ingress [spec](https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status) 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.

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](https://kubernetes.io/docs/concepts/services-networking/service/) or a [custom resource backend](https://kubernetes.io/docs/concepts/services-networking/ingress/#resource-backend) by way of a [CRD](https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/). HTTP (and HTTPS) requests to the Ingress that matches 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.

DefaultBackend

An Ingress with no rules sends all traffic to a single default backend. The defaultBackend is conventionally a configuration option of the [Ingress controller](https://kubernetes.io/docs/concepts/services-networking/ingress-controllers) and is not specified in your Ingress resources.

If none of the hosts or paths match the HTTP request in the Ingress objects, the traffic is routed to your default backend.

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.