

Hand Written Kubernetes Notes on

Ingress Controller & Ingress Resource (A Strong Student Guide to K8s Networking)

First Understand the Problem Statement (Understanding the Need for Ingress)

Imagine you have a Kubernetes cluster running multiple services, such as a frontend that handles user interactions, a backend that processes requests, and a database that stores information. To make these services accessible to users outside the cluster, you need to expose them.

One way to expose services is by using NodePort, which assigns a unique port on every node. However, this approach has a major drawback—each service requires an open port, increasing security risks by making cluster nodes directly accessible from the internet. Another option is using a LoadBalancer, which provides external access by creating a dedicated cloud-based load balancer for each service. While this method is effective, it becomes expensive and difficult to manage as the number of services grows.

Managing multiple NodePorts or LoadBalancers for different services leads to security vulnerabilities, increased operational complexity, and high costs. Instead of exposing every service separately, a more efficient approach is needed.

And the solution? **Ingress!**

Ingress acts as a single entry point that routes external requests to the appropriate services based on defined rules. Instead of assigning a separate LoadBalancer or NodePort for each service, Ingress allows controlled and secure access while simplifying traffic management, reducing costs, and improving security.

Prerequisites Before Using Ingress in Your Cluster

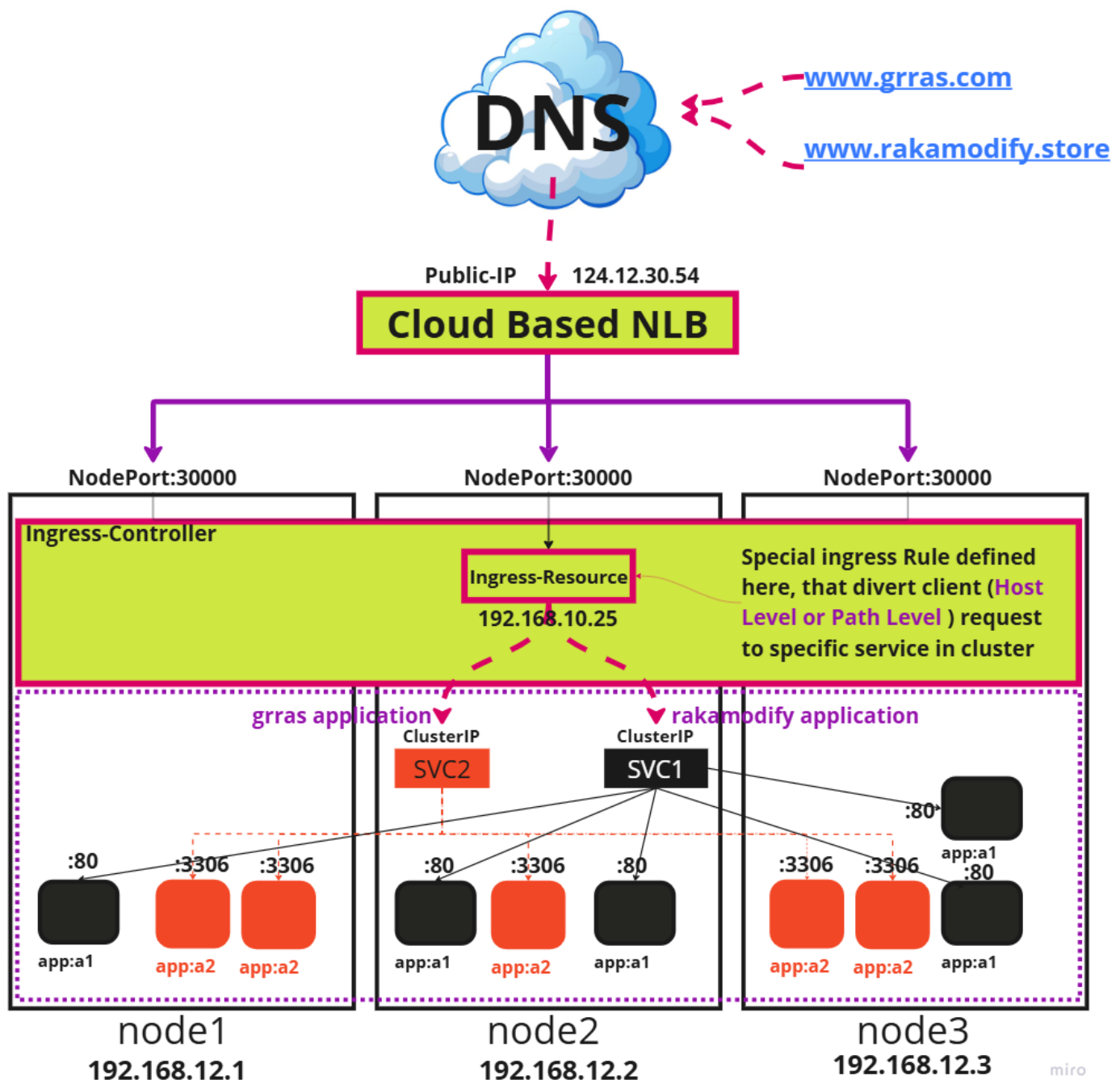
Before implementing Ingress to manage external traffic in your Kubernetes cluster, there are two essential components you must have in place. Without these, Ingress cannot function properly.

- **Ingress Controller** – This is the actual component that processes incoming traffic and enforces the routing rules defined in Ingress Resources. Kubernetes does not come with an Ingress Controller by default, so you need to install and configure one separately.
- **Ingress Resource** – This is a Kubernetes object that defines the rules for routing traffic, such as which requests should be sent to which service based on domain names or URL paths. However, an Ingress Resource alone does not handle traffic; it requires an Ingress Controller to function.

Without an Ingress Controller, Ingress Resources are just configurations that Kubernetes cannot act upon. In the next section, we will dive deeper into both **Ingress Controller** and **Ingress Resource** to understand how they work together to manage external traffic efficiently.

I have created an **image diagram below** to visually explain how **Ingress** and **Ingress Controller** function with two deployment services inside your Kubernetes cluster.

Ex: Image Diagram:



What is an Ingress Controller?

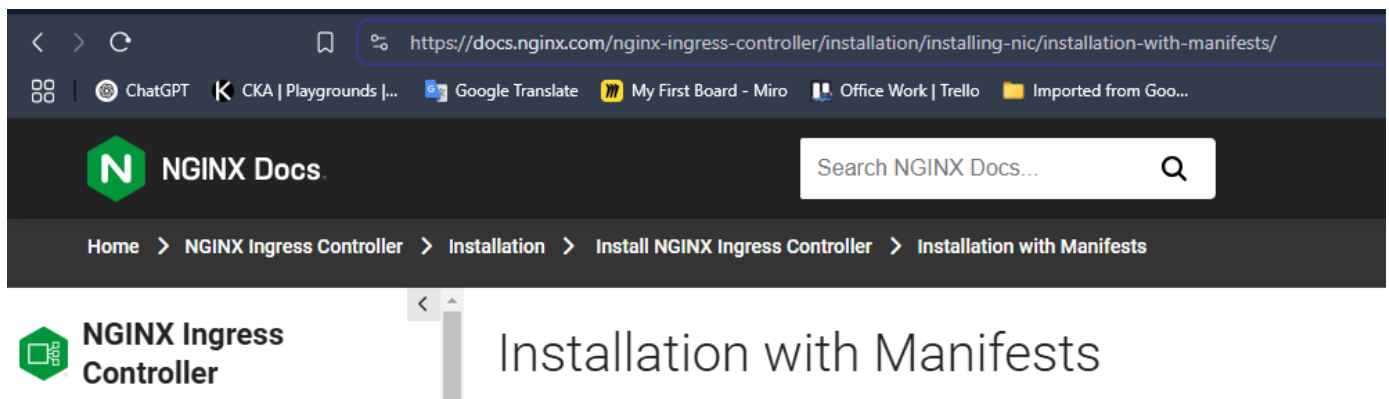
In Kubernetes Ingress Resource alone cannot process requests. It requires an **Ingress Controller** to implement and enforce the rules defined in Ingress.

Ingress Controller is a special pod that reads Ingress Resources and routes traffic accordingly.

- **Some Popular Ingress Controllers in Market:**
 - **NGINX Ingress Controller** (widely used)
 - **Traefik Ingress Controller**
 - **HAProxy Ingress Controller**
 - **Istio Gateway**

How to install Ingress Resource in your Kubernetes cluster?

Ingress Controller helps manage external traffic in a Kubernetes cluster. We are following the official nginx documentation for this Link: [https://docs.nginx.com/](https://docs.nginx.com/nginx-ingress-controller/installation/installing-nic/installation-with-manifests/) .

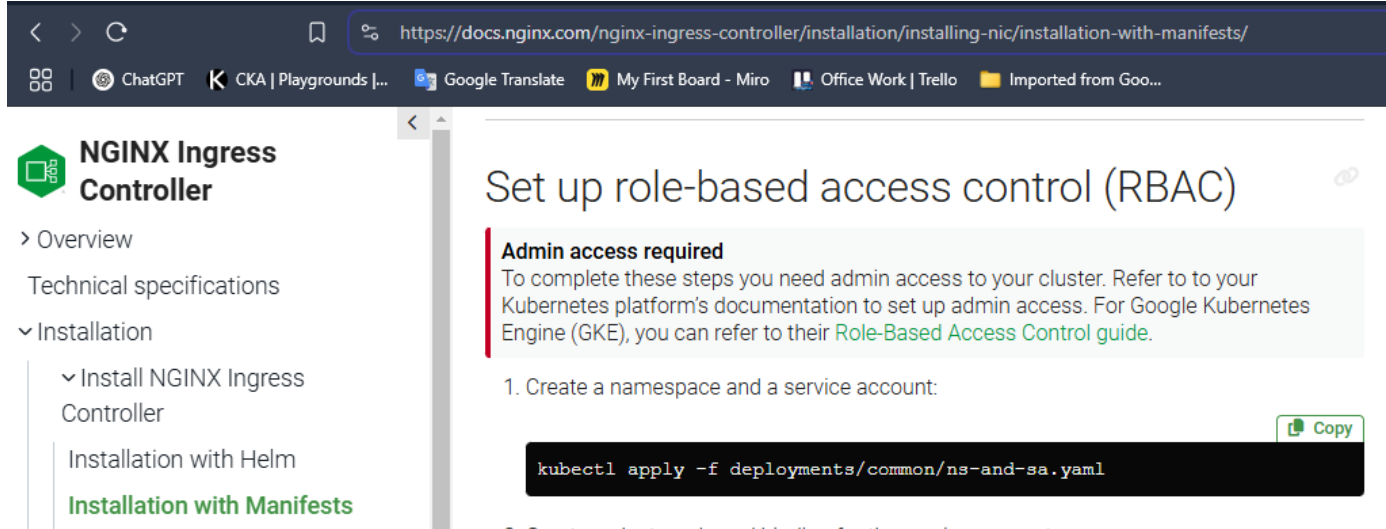


There are multiple ways to install the **NGINX Ingress Controller**:

1. **Installation with Helm**
2. **Installation with Manifests** (Recommended in this guide)
3. **Installation with NGINX Ingress Operator**
4. **Upgrade to NGINX Ingress Controller 4.0.0**

For detailed steps, follow the official documentation:

<https://docs.nginx.com/nginx-ingress-controller/installation/installing-nic/installation-with-manifests/>



NGINX Ingress Controller

- > Overview
- Technical specifications
- ▼ Installation
 - ▼ Install NGINX Ingress Controller
 - Installation with Helm
 - Installation with Manifests

Set up role-based access control (RBAC)

Admin access required
To complete these steps you need admin access to your cluster. Refer to your Kubernetes platform's documentation to set up admin access. For Google Kubernetes Engine (GKE), you can refer to their [Role-Based Access Control guide](#).

1. Create a namespace and a service account:

```
kubectl apply -f deployments/common/ns-and-sa.yaml
```

2. Create a clusterrole and binding for the service account.

What is an Ingress Resource?

Ingress Resource is a Kubernetes object that defines rules for routing HTTP and HTTPS traffic to the correct service inside the cluster.

It allows:

- URL-based routing
- Load balancing
- SSL/TLS termination
- Redirects and rewrites

Real-World project: (Traffic Flow in Kubernetes with Ingress)

Scenario Overview

In our Kubernetes cluster, two projects are running with the following domains:

- **www.grras.com** → Should be routed to **SVC2** (backend-service)
- **www.rakammodify.store** → Should be routed to **SVC1** (frontend-service)

How Does the Request Flow?

1. A client requests either **www.grras.com** or **www.rakammodify.store**.
2. The request first goes to the Public DNS (Domain Name System), which resolves the domain name to the public IP of the Network Load Balancer (NLB).
3. The NLB forwards the request to the attached Kubernetes cluster nodes using a **NodePort** service.
4. The Ingress Controller, running on the cluster, receives the request.
5. Based on the Ingress rules, the request is routed:
 - If the request is for **www.grras.com**, it is forwarded to **SVC2**.
 - If the request is for **www.rakammodify.store**, it is forwarded to **SVC1**.
6. The respective Pods behind the services process the request and respond back using the same route in reverse.

What is written in this traffic rule (Inside Ingress Resource)

This is an Ingress resource for routing traffic in the Kubernetes cluster.

It is created in the "amazon" namespace.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: project-ingress    # Name of the Ingress resource
  namespace: amazon       # Namespace where this Ingress is applied
spec:
  rules:
    - host: www.grras.com  # When a request comes for this domain
      http:                # Rule 1: Handles traffic for www.grras.com
        paths:
          - path: /         # Routes all traffic ("/" means root path)
            pathType: Prefix # Ensures the path-based routing
            backend:
              service:
                name: svc2    # Traffic is forwarded to the service "svc2"
                port:
                  number: 80  # The service port it should use
    - host: www.rakammodify.store # When a request comes for this domain
      http:                      # Rule 2: Handles traffic for www.rakammodify.store
        paths:
          - path: /             # Routes all traffic for this domain
            pathType: Prefix    # Path-based routing
            backend:
              service:
                name: svc1      # Traffic is forwarded to the service "svc1"
                port:
                  number: 80    # The service port it should use
```

```
# kubectl create -f ingress-resource.yml
```

Why Should we Use Ingress?

When deploying applications in Kubernetes, exposing services externally is a crucial challenge. Let's explore why Ingress is the best solution.

What was the problem without Ingress

1. Using NodePort:

- a) When you expose an application using NodePort, each service requires a unique port.
- b) If you have multiple services, you must open multiple NodePorts, which exposes your cluster nodes directly to the internet.
- c) More open NodePorts = More security risks (like open doors for hackers).

2. Using LoadBalancer for Each Service:

- a) Every LoadBalancer service creates a new cloud-based Load Balancer, which is expensive.
 - b) More services = More LoadBalancers = High cost & complex setup.
-

What is the solution: Ingress!

Ingress acts as a smart gateway that solves these issues efficiently.

1. Smart Traffic Routing

- Ingress allows you to route traffic based on domain names or URL paths instead of assigning a NodePort to every service.
- Example:
 - `www.grras.com` → Forwarded to `svc2`
 - `www.rakammodify.store` → Forwarded to `svc1`

2. Cost-Effective

- Instead of requiring a separate LoadBalancer for every service, Ingress works with a single LoadBalancer and intelligently directs traffic.
- Fewer LoadBalancers = Less cost & easy maintenance.

3. Improved Security

- Since Ingress handles traffic internally, you don't need to expose multiple NodePorts.
- Fewer open ports = Less attack surface = Better security.
- Also supports TLS termination (HTTPS), making communication secure.

4. Simplified Configuration

- **Ingress provides powerful features like:**
 - URL rewriting (e.g., `/old-path` → `/new-path`)
 - Redirections (e.g., `http` → `https`)
 - Traffic splitting (e.g., A/B testing).
-

<https://www.linkedin.com/in/rakeshkumarjangid/>

