**Q & A**

**Q1. How NodePort Works**

When you access a Kubernetes **Service** using the **public IP of a Node** and the **Service port**, you are using what's called a **NodePort** type of service.

**Example Setup**

Let’s say:

* You have a **NodePort service** exposing your app.
* Service listens on **port 30080**.
* Your Node has a **public IP**: 203.0.113.10.
* You access: http://203.0.113.10:30080

**How It Works Internally (Step-by-Step)**

**1. NodePort Service**

When you create a Kubernetes Service of type NodePort, Kubernetes:

* Opens a **high port (usually 30000–32767)** on **every node** in the cluster.
* Maps this port to the **targetPort** on the **Pod’s container**.

type: NodePort

ports:

port: 80 # service port (optional)

targetPort: 8080 # container port

nodePort: 30080 # exposed on node's IP

**2. Access via Public IP**

When you hit http://203.0.113.10:30080, the request goes to:

* **Node’s OS IP tables (or IPVS)** which detect that this is a NodePort.
* Kubernetes networking rules **route the traffic to the service**.
* The Service **load balances** the request to one of the matching Pods across the cluster.

**3. Pod Selection**

The Service uses **label selectors** to find matching Pods and forwards the request to one of them using:

* kube-proxy (on each node)
* Either **iptables** or **IPVS** to route traffic
* It doesn’t matter if the Pod is on the same node or a different one

**2. How many labels can one service handle? Does it have to be 1:1? (Shreya)**

There is **no hard limit** on the **number of labels** a Kubernetes **Service** can have, but practical and architectural limits apply.

**🔹 Technically:**

* You can assign **multiple labels** (dozens or more) to a service under metadata.labels.
* Kubernetes **stores labels in etcd**, and each object must stay within the etcd object size limit of **1 MB**.
* Each **label key and value**:
  + Must be ≤ 63 characters
  + Full label key (with prefix) ≤ 253 characters

**🔹 Practically:**

* It's common to use **3–10 labels** per service.
* Excessive labeling makes things complex and may hurt API performance, label-based selectors, and RBAC clarity.

**2. Does label matching have to be 1:1 between Service and Pod?**

No, it **does not** have to be a 1:1 match.

**🔹 Here’s how it works:**

* A **Service’s selector** (e.g., app: my-app) is used to **select matching Pods**.
* The **Service does not need to match all Pod labels** — only the **labels listed in the selector**.
* The **Pod can have more labels** than the selector requires — that’s okay!

**Example:**

**Pod with multiple labels:**

metadata:

labels:

app: my-app

tier: frontend

version: v1

**Service selecting just one:**

spec:

selector:

app: my-app

This Service will still match the Pod successfully.

**But this will NOT work:**

If the **selector** expects more than the Pod provides:

**Service selector:**

spec:

selector:

app: my-app

version: v2

**Pod label:**

metadata:

labels:

app: my-app

No match, since version: v2 is missing on the Pod.