How do you get a static IP for a Kubernetes load balancer?

In Kubernetes, a LoadBalancer service typically gets a **dynamic IP** from the cloud provider. But if you want a **static (reserved) IP**, especially for DNS or stability reasons, you need to **manually reserve it** and **attach it to your service**.

💡 Why Use a Static IP?

- ✅ You want a **predictable IP** for a LoadBalancer (e.g., for whitelisting).
- ✅ You plan to associate a **custom domain** (DNS A record).
- ✅ Avoid IP change every time the service restarts.

🧭 Approach Depends on the Cloud Provider

Here's how it works with different environments:

🟦 1. On GKE (Google Kubernetes Engine)

✅ **Steps:**

1. **Reserve a static IP** on GCP:

```
bash
----------
gcloud compute addresses create my-static-ip --region=us-central1
```

2. **Get the reserved IP address**:

```
bash
----------
gcloud compute addresses describe my-static-ip --region=us-central1 --
format="get(address)"
```

3. **Use it in your Kubernetes service**:

```
yaml
----------
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: LoadBalancer
  loadBalancerIP: <STATIC_IP>
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

⚠️ The IP must be in the **same region** as your cluster.

🟨 2. On AWS EKS

AWS does not allow you to directly assign static IPs to LoadBalancers, but you can:

- Create an **Elastic IP** (EIP)
- Use a **NLB (Network Load Balancer)** with Kubernetes service annotations and assign the EIP

It involves an **AWS Load Balancer Controller** and IAM setup.

🟪 3. On Azure AKS

1. **Reserve a static public IP**:

```
bash
---------
az network public-ip create \
  --name myStaticIP \
  --resource-group myResourceGroup \
  --allocation-method static \
  --sku standard
```

2. **Get the IP resource ID**:

```
bash
---------
az network public-ip show \
  --name myStaticIP \
  --resource-group myResourceGroup \
  --query id \
  --output tsv
```

3. **Attach to Kubernetes service** using annotation:

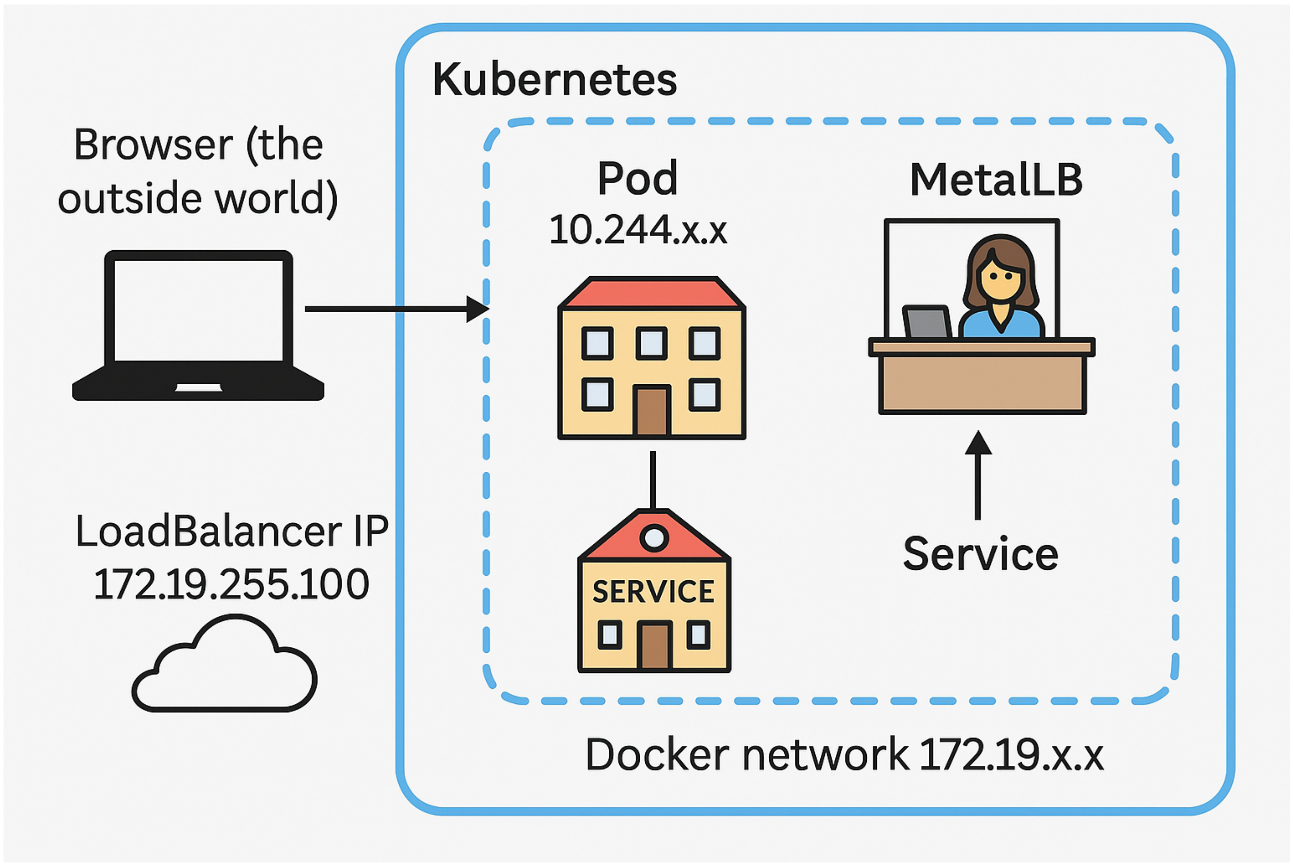
```
yaml
---------
metadata:
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-resource-group: 
myResourceGroup
```

Imagine This:

You're setting up a mini city (Kubernetes cluster) inside your laptop. This city has:

- **Buildings** (Pods – your applications)
- **Streets** (Networks so apps can talk to each other)
- **Post Office** (Services – handles who gets what message)
- **City Rules** (Kubernetes Configs)

Now, to run this mini city, you're using a tool called **Kind**, which runs Kubernetes inside **Docker containers**.


Browser (the outside world)
LoadBalancer IP 172.19.255.100
Kubernetes
Pod 10.244.x.x
SERVICE
MetalLB
Service
Docker network 172.19.x.x

1. Kind creates the city inside a container

Your city (cluster) is surrounded by a **boundary wall** (Docker bridge network). Everything inside uses special private streets (IP addresses) that you define.

- **podSubnet** = Where buildings (Pods) live.
- **serviceSubnet** = Where internal post offices (Services) live.

These IPs are **only used inside the city**.

2. Docker's job: Create the wall around the city

Docker makes a **bridge** (a virtual router) around your city and gives it a range of IPs like `172.19.0.0/16`.

This is the **"outer city" network**—your laptop's way of reaching inside.

---

3. The Problem: No way for the outside world to talk to the city

By default:

- If someone tries to visit your web app (like from a browser), they **can't get in** because there's no external door (IP address) open.

---

4. MetalLB

MetalLB is like a smart receptionist you install inside your city.

- It knows how to **talk to Docker's outer network** (the `172.19.x.x` range).
- It says: "Give me a few IPs from your range so I can share them with services inside."

Now when you expose your app using a LoadBalancer service and give it a MetalLB IP, like:

```
yaml

loadBalancerIP: 172.19.255.100
```

MetalLB connects that outer IP to the app inside the city.

🎉 Final :

Now when you visit `http://172.19.255.100` in your browser:

- MetalLB picks up the request at the Docker level,
- Routes it through the Kubernetes city gates,
- Takes it to the right building (Pod),
- And you see your web page!

---

4. On Bare Metal / Kind / Minikube

`Setup for Kind + MetalLB + Static IP Demo`

1. Create Kind Cluster with Custom Network Configuration

You need to ensure your Kind cluster uses a fixed subnet (because you're assigning IPs manually).

Create a Kind cluster config (`kind-config.yaml`):

```
yaml
---------
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
networking:
  # Same subnet range where you'll assign MetalLB IPs
  podSubnet: "10.244.0.0/16"
  serviceSubnet: "10.96.0.0/12"
nodes:
  - role: control-plane
    extraPortMappings:
      - containerPort: 80
        hostPort: 80
        protocol: TCP
```

Create the cluster:

```
bash
---------
kind create cluster --config kind-config.yaml
```

1. `kind config file sets` Kubernetes-level subnets

These settings:

```
yaml

networking:
  podSubnet: "10.244.0.0/16"
  serviceSubnet: "10.96.0.0/12"
```

Tell Kubernetes how to assign:

- `10.244.x.x` → for Pods (inside the cluster)
- `10.96.x.x` → for Services (ClusterIP etc.)

These **do not affect Docker network subnets**.

Docker-level subnet (`docker network inspect kind`)

```
json
CopyEdit
"Subnet": "172.19.0.0/16"
```

This is the **bridge network** Docker creates to host your Kind control-plane node. This is where MetalLB gets involved.

> 🔥 MetalLB assigns IPs **from the Docker bridge network**, not from Kubernetes serviceSubnet.

So why does MetalLB need IPs in `172.19.x.x`?

Because:

- MetalLB advertises **external LoadBalancer IPs** on the host (Docker side),
- These must be reachable from the host system,
- So you must pick IPs from Docker's bridge network (in this case, `172.19.0.0/16`)

🧱 2. Install MetalLB

Make sure the cluster is ready, then apply MetalLB:

```
bash
---------
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.13.10/config/manifests/meta
llb-native.yaml
```

Wait for pods in `metallb-system` namespace to be ready.

```
bash
---------
kubectl get pods -n metallb-system
```

📦 3. Configure MetalLB IP Pool & L2Advertisement

```
yaml
---------
# metallb-config.yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: my-ip-pool
  namespace: metallb-system
spec:
  addresses:
    - 172.18.255.1-172.18.255.250
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: example
  namespace: metallb-system
```

Apply the config:

```
bash
---------
kubectl apply -f metallb-config.yaml
```

> ✅ Make sure the IPs (like `172.18.255.X`) are in the Docker network used by Kind (`docker network inspect kind` to verify). If not, adjust to match your setup.

🚀 4. Deploy Your App and LoadBalancer Service

Deployment:

```
yaml
---------
# my-app.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: nginx
        ports:
        - containerPort: 80
```

Service with static IP:

```
yaml
---------
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: LoadBalancer
  loadBalancerIP: 172.18.255.100
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 80
```

Apply both:

```
bash
---------
kubectl apply -f my-app.yaml
```

🌐 5. Test Access Using Static IP

Now, run:

```
bash
---------
curl http://172.19.255.100
```

Or open in browser if you're on Linux/macOS:

```cpp
http://172.18.255.100
```

If you're on Windows or using WSL, you may need to **use port-forwarding or map the IP to localhost** via `/etc/hosts`.

✅ Notes

- To access from **outside Docker**, map ports (done in Kind config).
- To demo to others on your network, you might need to tweak **iptables** or use **ngrok** for external exposure.
- `kubectl get svc` will show the `EXTERNAL-IP` as `172.18.255.100` if all is configured correctly.
-

✅ Summary

Platform	Static IP Steps
GKE	Reserve IP in GCP → set `loadBalancerIP`
AWS EKS	Use EIP with NLB + annotations (via AWS LB Controller)
Azure AKS	Reserve IP in Azure → use annotation
Bare Metal	Use MetalLB with IP pool + `loadBalancerIP`

Q: Why does a LoadBalancer service in a Kind cluster not work out-of-the-box?

A. Because Kind doesn't support YAML files
B. Because LoadBalancer requires a cloud provider by default
C. Because Docker doesn't allow inbound traffic
D. Because MetalLB is installed by default

✅ **Correct Answer: B**
📘 **Explanation:** LoadBalancer type needs a cloud provider (like AWS, GCP). In Kind or bare metal, you need something like MetalLB.

Q: What is the purpose of MetalLB in a Kubernetes cluster?

A. To manage DNS names
B. To expose internal metrics

C. To provide LoadBalancer functionality on bare-metal or local setups
D. To schedule pods across nodes

✅ **Correct Answer: C**
📘 **Explanation:** MetalLB is used to emulate LoadBalancer services in environments without a built-in cloud load balancer.