

**Lab Manual- Using Azure Application Gateway as Ingress Controller for AKS**

**Prepared for**:

**Date:** 18th Dec 2023

**Prepared by:**

Document Name: Lab Manual **Document Number** AZLabn916

**Contributor:**

Contents

[1. Objective 3](#_Toc159582072)

[1. How AGIC works ? 3](#_Toc159582073)

[2. What are the pros and cons of using AGIC when compared with Nginx Ingress Controler ? 4](#_Toc159582074)

[2. App-Gateway SAMPLE ARCHITECTURE 4](#_Toc159582075)

[3. Azure Well-Architected Framework review - Azure Application Gateway v2 5](#_Toc159582076)

[1. Reliability 5](#_Toc159582077)

[1.1. Design checklist 5](#_Toc159582078)

[2. Security 5](#_Toc159582079)

[1.2. Design checklist 6](#_Toc159582080)

[3. Cost optimization 6](#_Toc159582081)

[1. Design checklist 6](#_Toc159582082)

[4. Operational excellence 6](#_Toc159582083)

[1. Design checklist 6](#_Toc159582084)

[5. Performance efficiency 6](#_Toc159582085)

[1. Design checklist 7](#_Toc159582086)

[4. AKS Cluster with single node and Azure CNI 7](#_Toc159582087)

[5. Enable Azure Application Gateway Ingress Controller 8](#_Toc159582088)

[6. Deploy a simple app with ingress 12](#_Toc159582089)

[7. To Check Logs File 17](#_Toc159582090)

# Objective

Kubernetes exposes services securely (HTTPS) through an ingress controller. Kubernets supports the ingress resources. But users should provide and install a plugin to handle ingress traffic. There are lots of plugins available like Nginx Ingress Controller or also Azure Application Gateway Ingress Controller (AGIC).

In this demonstration, we will enable AGIC extension in AKS and use it to expose a sample application to the internet or internal network.

## How AGIC works ?

Application Gateway will act as the frontend that will receive customer traffic. Then it will route the traffic to the pods directly.

How that is possible ?

This achievable because the Application Gateway and the AKS cluster should be in the **same network**. Either in **2 separate subnets** within the same VNET or in **2 different peered VNETs**. So the App Gateway can reach the Pods through their private IPs.

**But how the App Gateway could know the private IPs of the pods ?**

Here comes the AGIC extension. AGIC will be installed into the AKS cluster as a **pod** within **kube-system namespace**. Its role is to listen for ingress resources creation, get pod IPs then use it to control the configuration of the App Gateway. This means it will connect to the App Gateway and authenticate and authorize using a User Assigned Managed Identity created within the node resource group. AGIC will create the listeners and backend configuration for App Gateway. The following picture shows the workflow.

## What are the pros and cons of using AGIC when compared with Nginx Ingress Controler ?

There are a lot of features available for App Gateway and for Nginx IC. Here I will put only the most relevant ones. (this is not a refence for comparing the 2 tools).

|  | **Application Gateway** | **Nginx Ingress Controller** |
| --- | --- | --- |
| Support HTTPS/TLS | Yes | Yes |
| TLS decryption | outside the cluster | inside the cluster |
| Scale out | Outside the cluster | Inside the cluster (HPA) |
| Consume cluster resources | No | Yes |
| Cost | Cost of Azure resource (more expensive) | Cost of pods inside cluster (cheaper) |
| WAF | Supported with SKU WAF\_v2 | Very basic, needs Nginx Plus license |

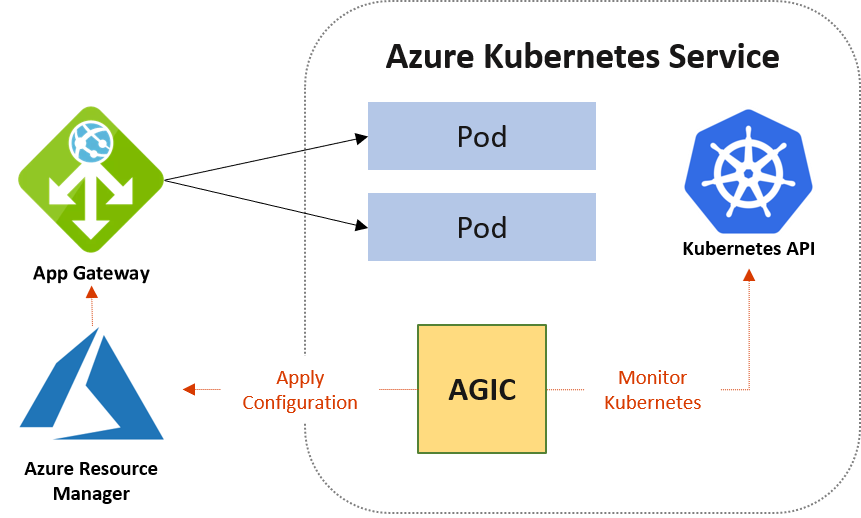
Note: The App Gateway will not consume resources from the cluster when doing TLS termination or scale out.

Note: With Kubenet, the cluster route table should be attached to the App Gateway subnet to reach pods. More details here:

<https://azure.github.io/application-gateway-kubernetes-ingress/how-tos/networking/>

<https://azure.github.io/application-gateway-kubernetes-ingress/>

# App-Gateway SAMPLE ARCHITECTURE



# Azure Well-Architected Framework review - Azure Application Gateway v2

## Reliability

In the cloud, we acknowledge that failures happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component. Use the following information to minimize failed instances.

### Design checklist

As you make design choices for Application Gateway, review the [Reliability design principles](https://learn.microsoft.com/en-gb/azure/well-architected/resiliency/principles).

* Deploy the instances in a [zone-aware configuration](https://learn.microsoft.com/en-us/azure/application-gateway/application-gateway-autoscaling-zone-redundant), where available.
* Use Application Gateway with Web Application Firewall (WAF) within a virtual network to protect inbound HTTP/S traffic from the Internet.
* In new deployments, use Azure Application Gateway v2 unless there is a compelling reason to use Azure Application Gateway v1.
* Plan for rule updates
* Use health probes to detect backend unavailability
* Review the impact of the interval and threshold settings on health probes
* Verify downstream dependencies through health endpoints

## Security

Security is one of the most important aspects of any architecture. Application Gateway provides features to employ both the principle of least privilege and defense-in-defense. We recommend you review the [Security design principles](https://learn.microsoft.com/en-gb/azure/well-architected/security/principles).

### Design checklist

* Set up a TLS policy for enhanced security
* Use AppGateway for TLS termination
* Use Azure Key Vault to store TLS certificates
* When re-encrypting backend traffic, ensure the backend server certificate contains both the root and intermediate Certificate Authorities (CAs)
* Use an appropriate DNS server for backend pool resources
* Comply with all NSG restrictions for Application Gateway
* Refrain from using UDRs on the Application Gateway subnet
* Be aware of Application Gateway capacity changes when enabling WAF

## Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. We recommend you review the [Cost optimization design principles](https://learn.microsoft.com/en-gb/azure/well-architected/cost-optimization/principles).

### Design checklist

* Familiarize yourself with Application Gateway pricing
* Review underutilized resources
* Stop Application Gateway instances that are not in use
* Have a scale-in and scale-out policy
* Review consumption metrics across different parameters

## Operational excellence

Monitoring and diagnostics are crucial for ensuring operational excellence of your Application Gateway and the web applications or backends behind the gateway. You can not only measure performance statistics but also use metrics to troubleshoot and remediate issues quickly. We recommend you review the [Operational Excellence design principles](https://learn.microsoft.com/en-gb/azure/well-architected/operational-excellence/principles).

### Design checklist

* Monitor capacity metrics
* Enable diagnostics on Application Gateway and Web Application Firewall (WAF)
* Use Azure Monitor Network Insights
* Match timeout settings with the backend application
* Monitor Key Vault configuration issues using Azure Advisor
* Configure and monitor SNAT port limitations
* Consider SNAT port limitations in your design

## Performance efficiency

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. We recommend you review the [Performance efficiency principles](https://learn.microsoft.com/en-gb/azure/well-architected/scalability/principles).

### Design checklist

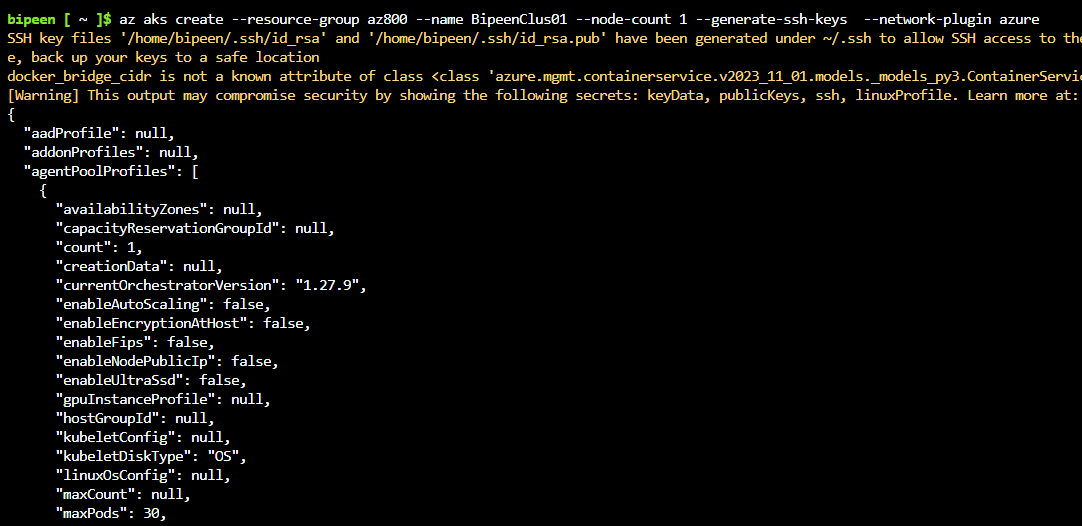
* Estimate the Application Gateway instance count
* Define the maximum instance count
* Define the minimum instance count
* Define Application Gateway subnet size
* Take advantage of Application Gateway V2 features for autoscaling and performance benefits

<https://learn.microsoft.com/en-gb/azure/well-architected/service-guides/azure-application-gateway>

# AKS Cluster with single node and Azure CNI

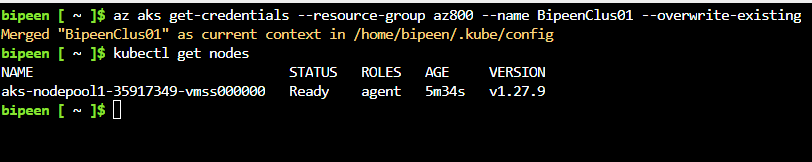
App Gateway works with both **Azure CNI** and **Kubenet** plugins.

az aks create --resource-group az800 --name BipeenClus01 --node-count 1 --generate-ssh-keys --network-plugin azure



az aks get-credentials --resource-group az800 --name BipeenClus01 --overwrite-existing

kubectl get nodes



# Enable Azure Application Gateway Ingress Controller

With Cli:

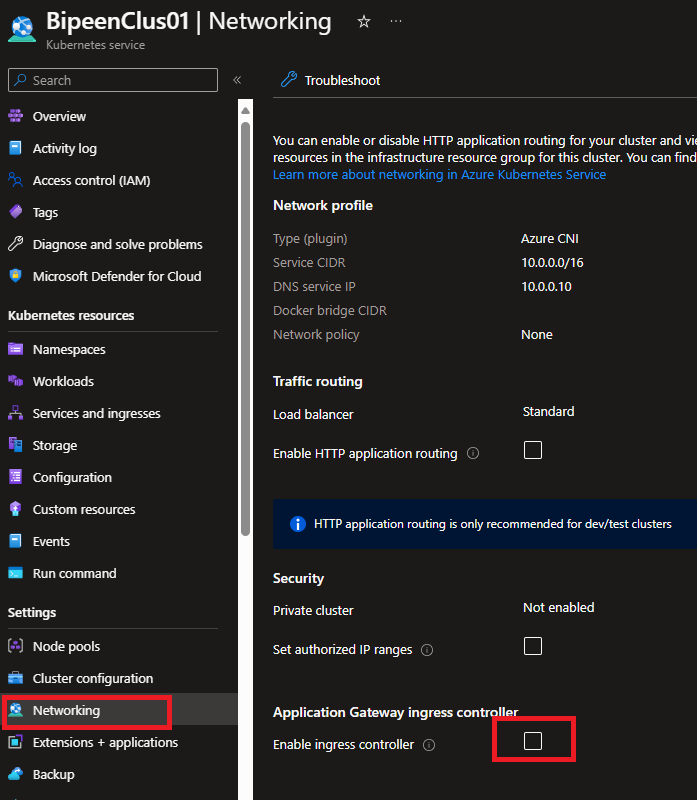
az aks addon enable -n aks-cluster -g rg-aks-cluster `

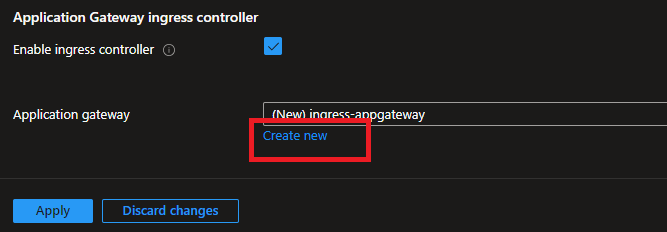
--addon ingress-appgw `

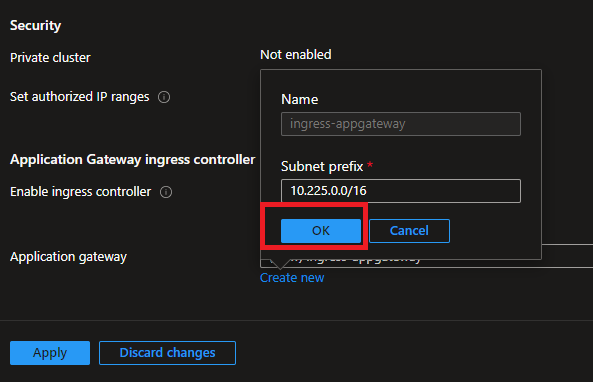
--appgw-subnet-cidr 10.225.0.0/16 `

--appgw-name gateway

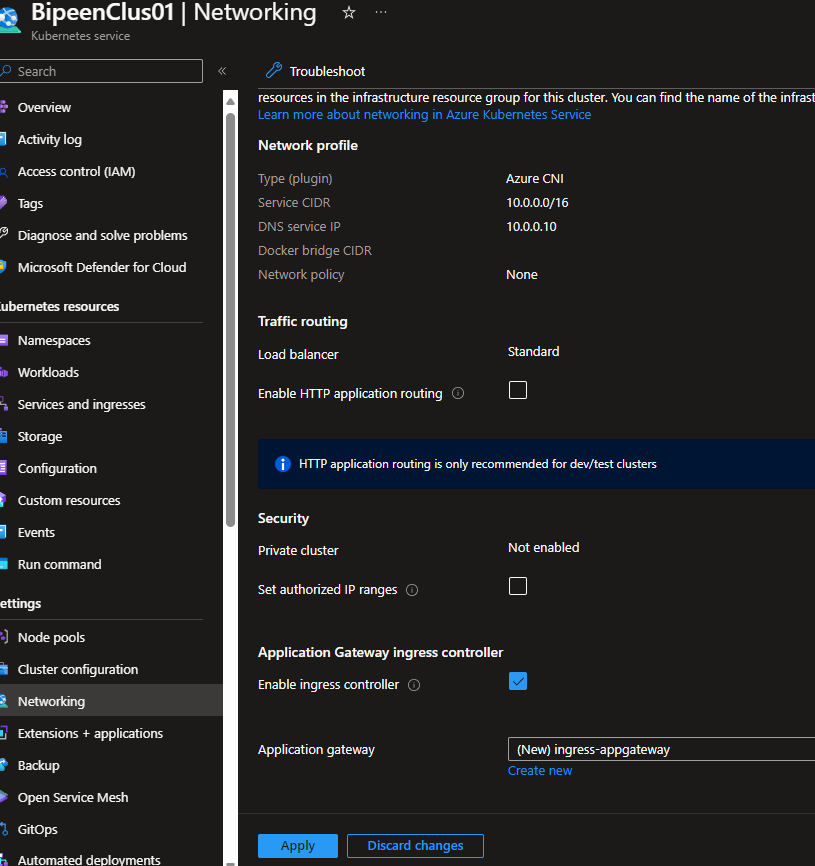
With Azure Portal Follow below Guidance



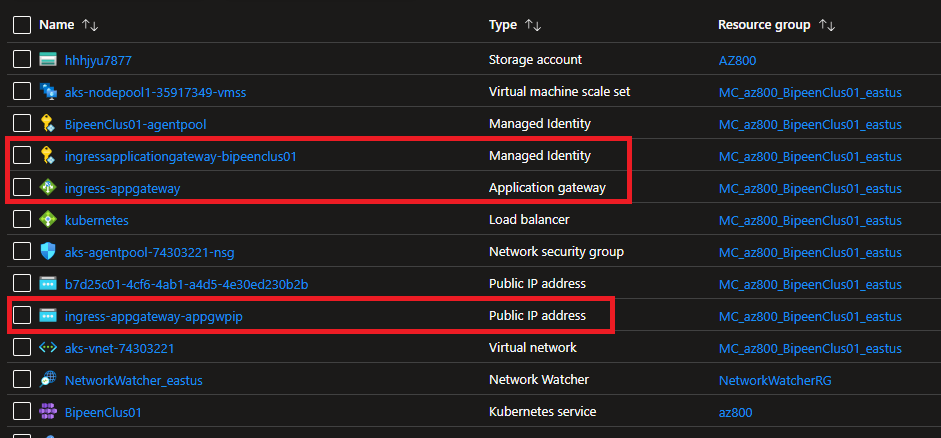




**Click Apply to Create the Application Gateway**

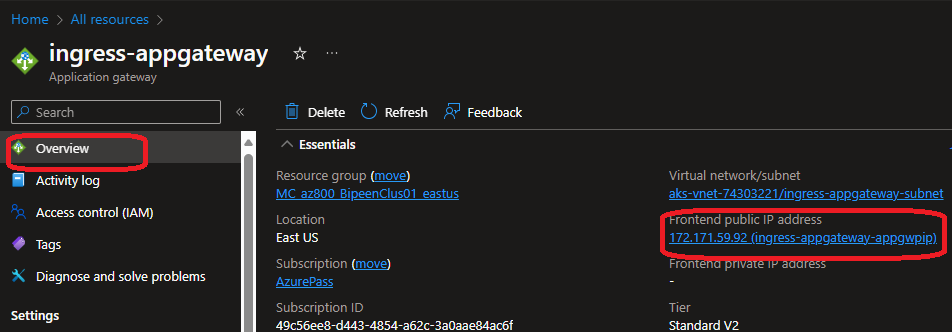


It Take Approx 5-10 Min to enable Application Gateway. Once Completed you can verify the Identity , Gateway and Public IP Associated with it

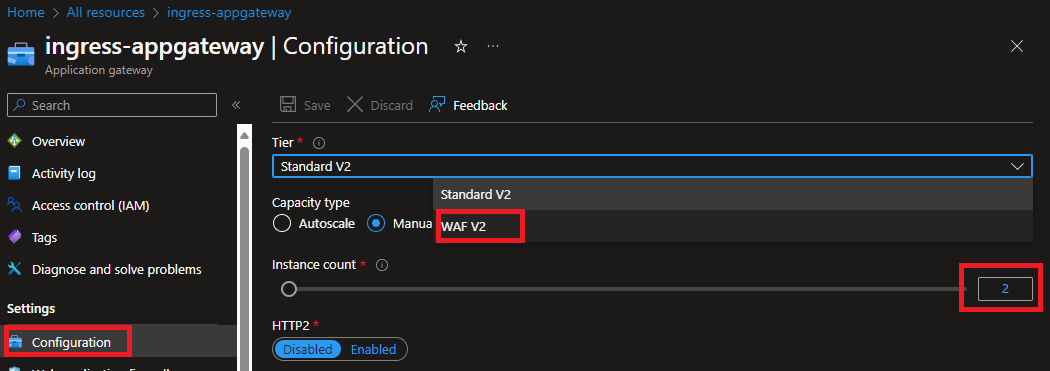


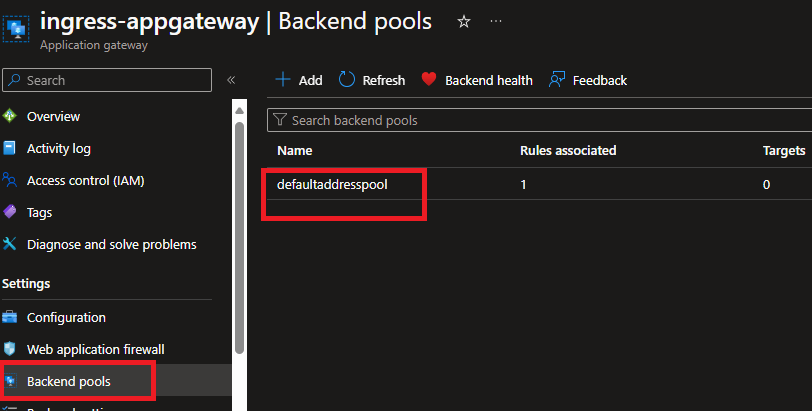
Check Application Gateway:

* 1. **Frontend IP**

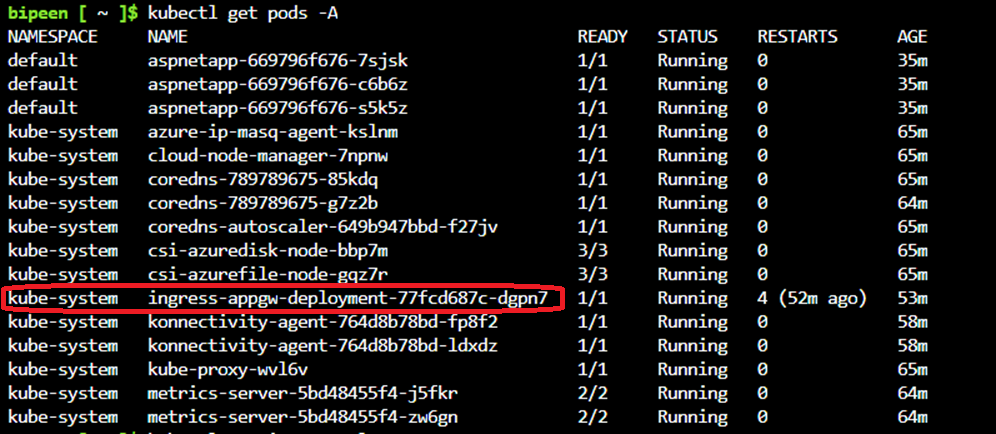


* 1. **Instance Count and Also can be integrated with WAF**

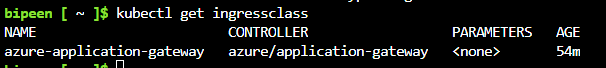


* 1. **Backend Pool**
* ****

kubectl get pods -A



kubectl get ingressclass



# Deploy a simple app with ingress

**ingress\_appgw.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: aspnetapp

  labels:

    app: aspnetapp

spec:

  replicas: 3

  selector:

    matchLabels:

      app: aspnetapp

  template:

    metadata:

      labels:

        app: aspnetapp

    spec:

      containers:

      - name: aspnetapp

        image: mcr.microsoft.com/dotnet/samples:aspnetapp

        ports:

        - containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

  name: aspnetapp

spec:

  selector:

    app: aspnetapp

  ports:

  - protocol: TCP

    port: 8080

    targetPort: 8080

---

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: aspnetapp

  # annotations:

    # kubernetes.io/ingress.class: azure/application-gateway

spec:

  ingressClassName: azure-application-gateway

  rules:

  - http:

      paths:

      - path: /

        backend:

          service:

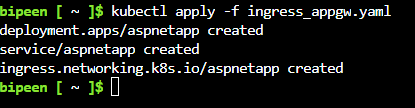
            name: aspnetapp

            port:

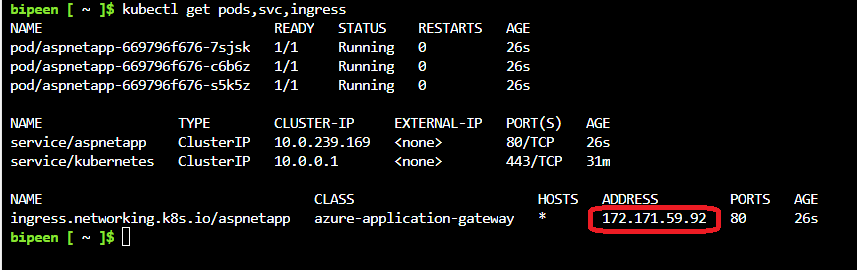
              number: 8080

        pathType: Exact

kubectl apply -f ingress\_appgw.yaml

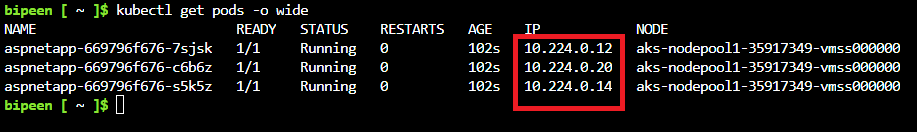


kubectl get pods,svc,ingress

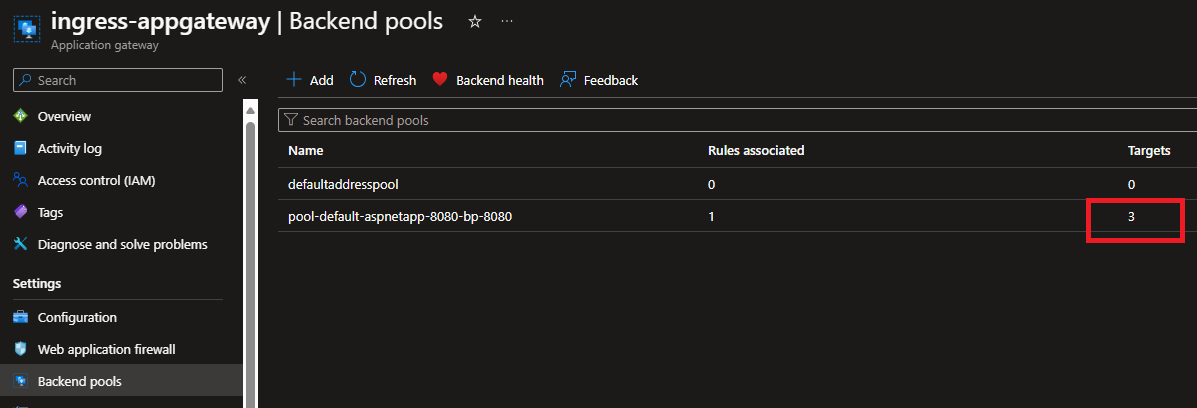


Note : The same IP which you see in App Gateway in Portal

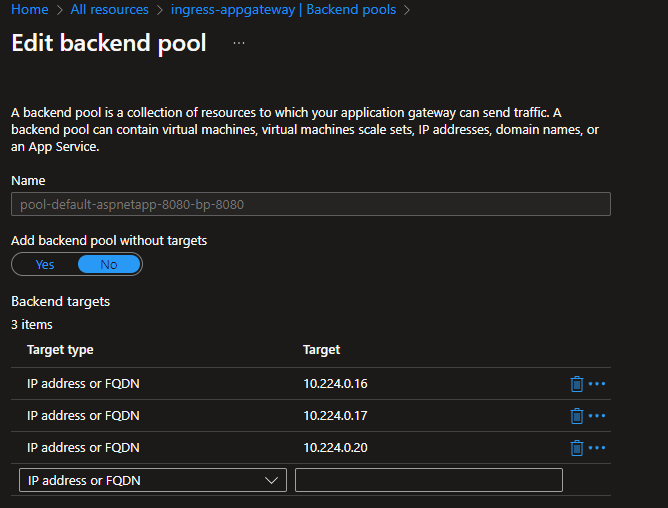
kubectl get pods -o wide



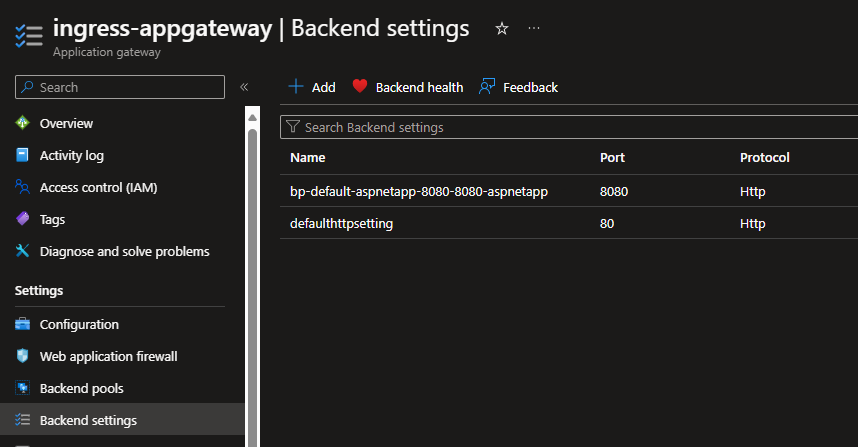
Note the IP and again go to App Gateway in Portal

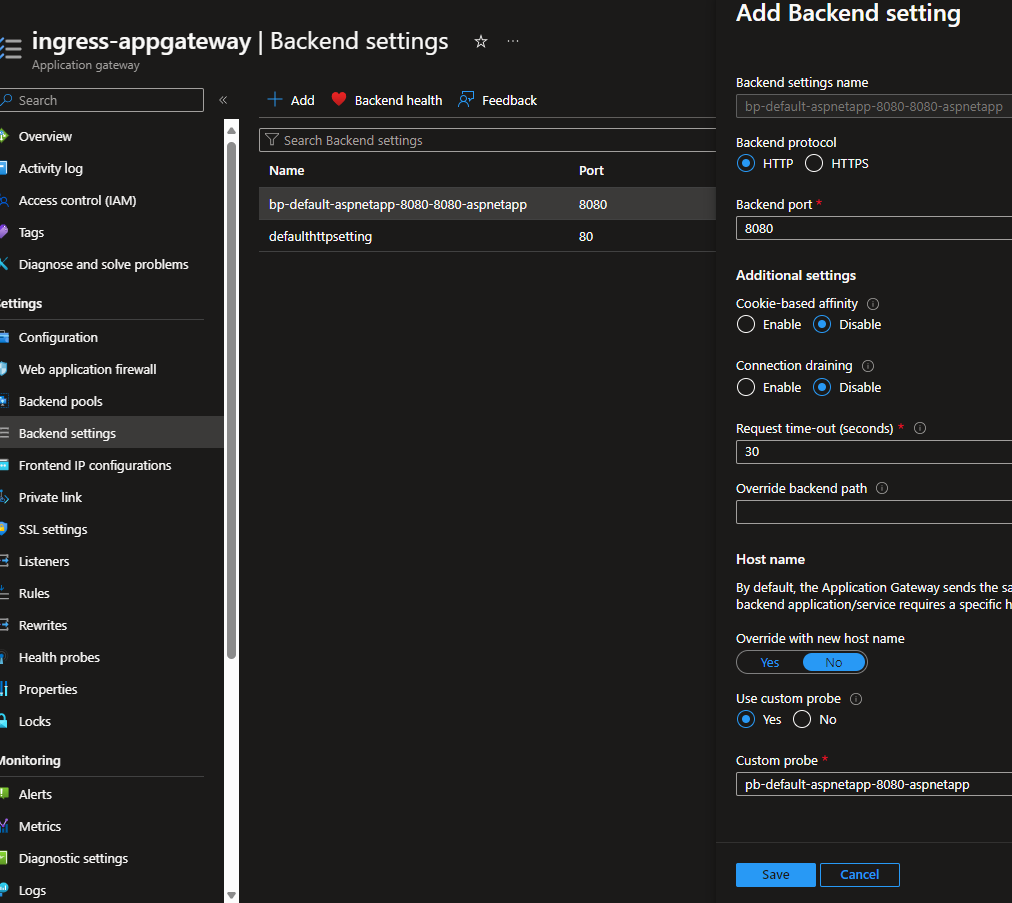


You can see the Same IP

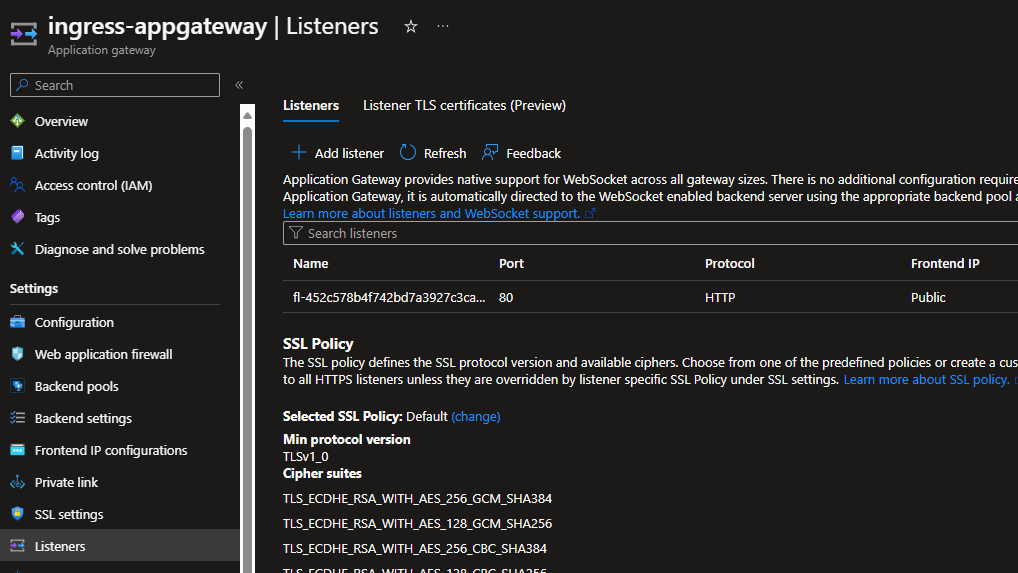


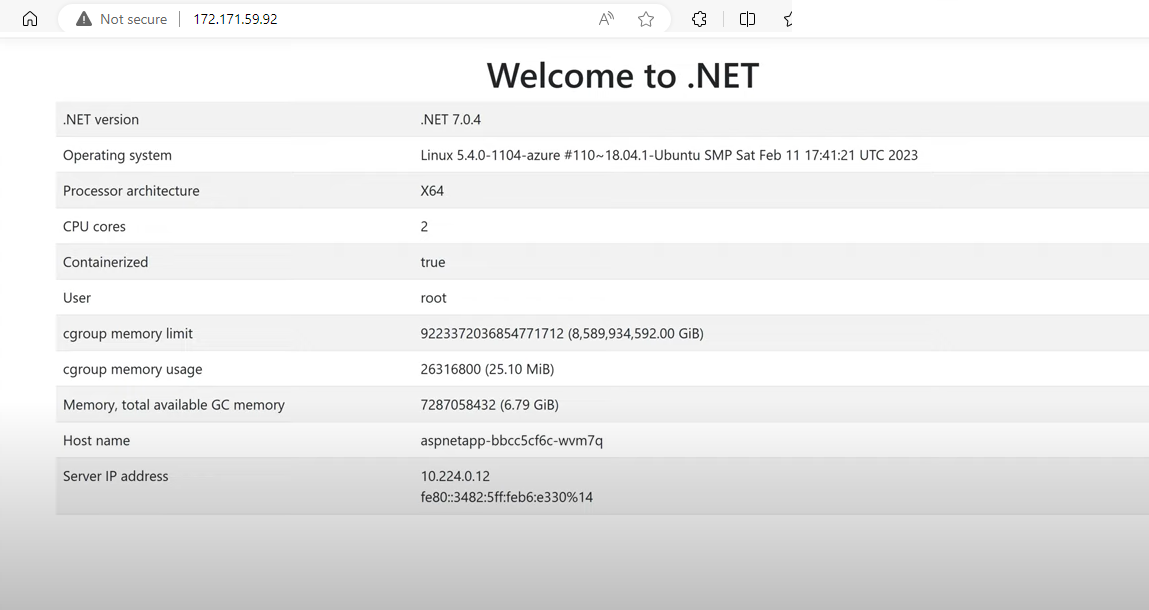
**Backend Setting**





**Ingress:**

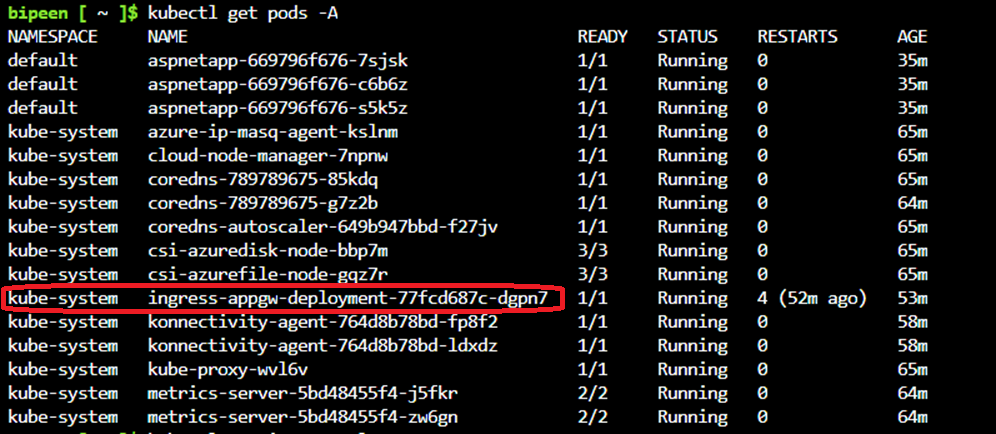




# To Check Logs File

**First see the PODs**

kubectl get pods -A

****

kubectl logs ingress-appgw-deployment-77fcd687c-dgpn7 -n kube-system | more

