Microsoft

Cloud Native Transformation of Applications Using Azure Kubernetes

Week 3 – AKS Networking

About the Speakers

Steve Caravajal, Ph.D. is Chief Cloud Solution Architect with extensive cloud experience, and 15+ yrs with Microsoft. He is focused on leading digital transformation and cloud native modernization for Microsoft's large strategic accounts. Steve is also Cloud Security lead, SME for multi-vendor cloud strategy, and K8s certified (CKA, CKAD).

David Hoerster, a former 6-time .NET MVP, has been working with the Microsoft.NET Framework since the early 1.0 betas and has recently found his next passion in Open Source technologies. He is currently a Cloud Solutions Architect at Microsoft specializing in application development and identity. He also recently earned his CKA and CKAD.

4 Week Agenda Overview

Week 1 – July 14
Containers, Azure Kubernetes Service (AKS), Azure Container Registry (ACR)
Establish foundation to enable advanced implementations and configuration in Weeks 2-4.

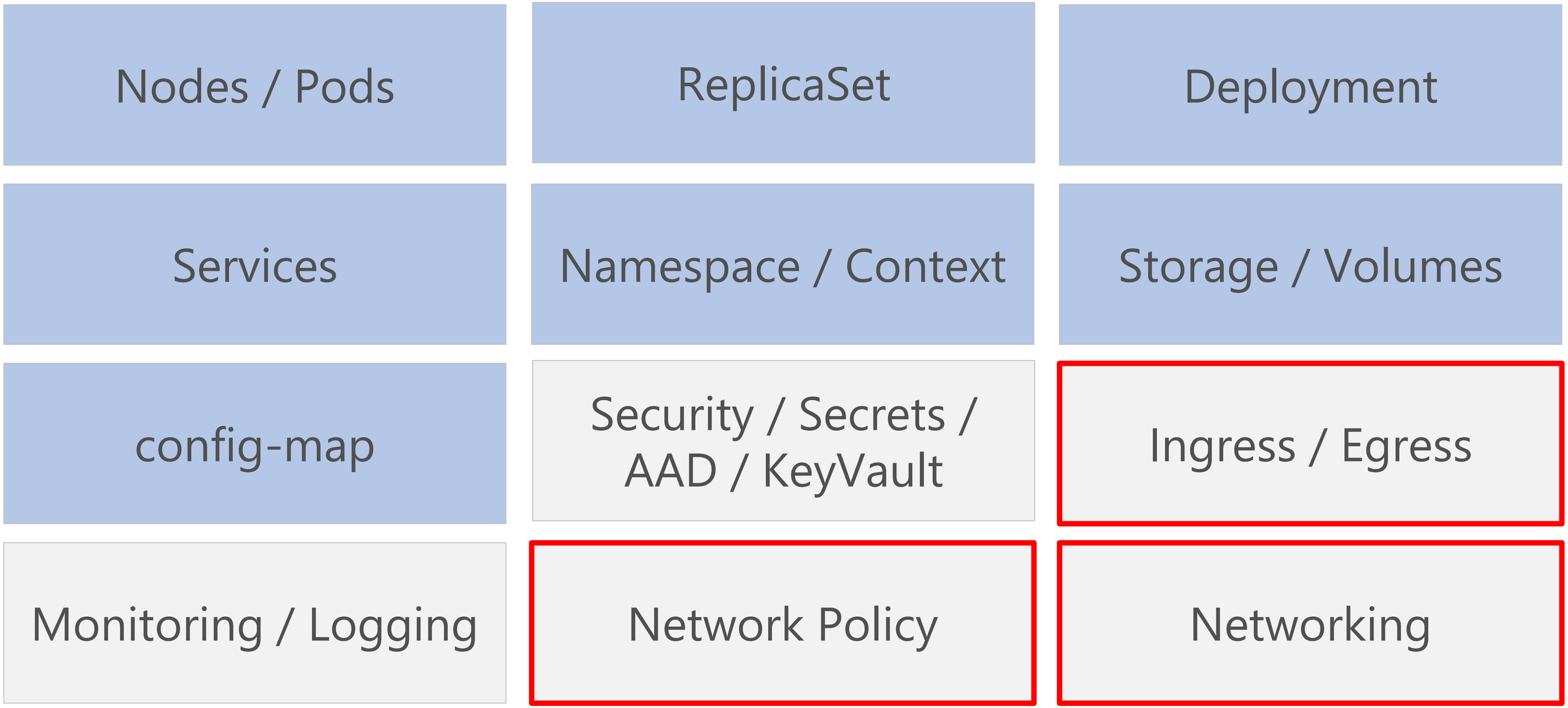
Week 2 – July 21
Storage, Config-maps, Namespace, Packaging and Deployment Templates and YAML

Week 3 – July 28
AKS Networking, Managing Ingress/Egress, Network Policy, Private Cluster

Week 4 – Aug 4
Deploying a Distributed Application, Security, Monitoring, and Service Mesh

Workshop Content and Labs
https://github.com/Azure/sg-aks-workshop

Kubernetes Topics – Week 3



Nodes / Pods

ReplicaSet

Deployment

Services

Namespace / Context

Storage / Volumes

config-map

Security / Secrets / AAD / KeyVault

Ingress / Egress

Monitoring / Logging

Network Policy

Networking

Agenda

- AKS Networking Options
- Services
- Ingress, Egress, & Ingress Controllers
- Network Policies
- Advanced Networking Scenario

Microsoft

Kubernetes Networking

Networking in Kubernetes

Kubernetes knows 4 methods of communications

- **Container to Container** - containers within the same Pod are in the same network namespace, including their IP address, and can all reach each other's ports on localhost.

- **Pod-to-Pod -** communication directly by IP address.

- **Pod-to-Service** - communication is directed to service virtual IP by kube-proxy process (running on all hosts) and directed to the correct Pod.

- **External-to-Internal** - communication – external access is captured by an external load balancer which targets nodes in a cluster. The Pod-to-Service flow stays the same.

Microsoft

AKS Networking Models

Network Configuration Options

Options

- ✓ Basic
- ✓ Advanced

When you create an Azure Kubernetes Service (AKS) cluster, you can choose between two networking options:

- ***Kubenet* or Basic Networking** - Azure manages the virtual network resources as the cluster is deployed and uses the **kubenet** Kubernetes plugin.

- ***Azure Container Networking Interface (CNI)* or Advanced Networking** - Deploys into an existing virtual network and uses the **CNI** Kubernetes plugin.

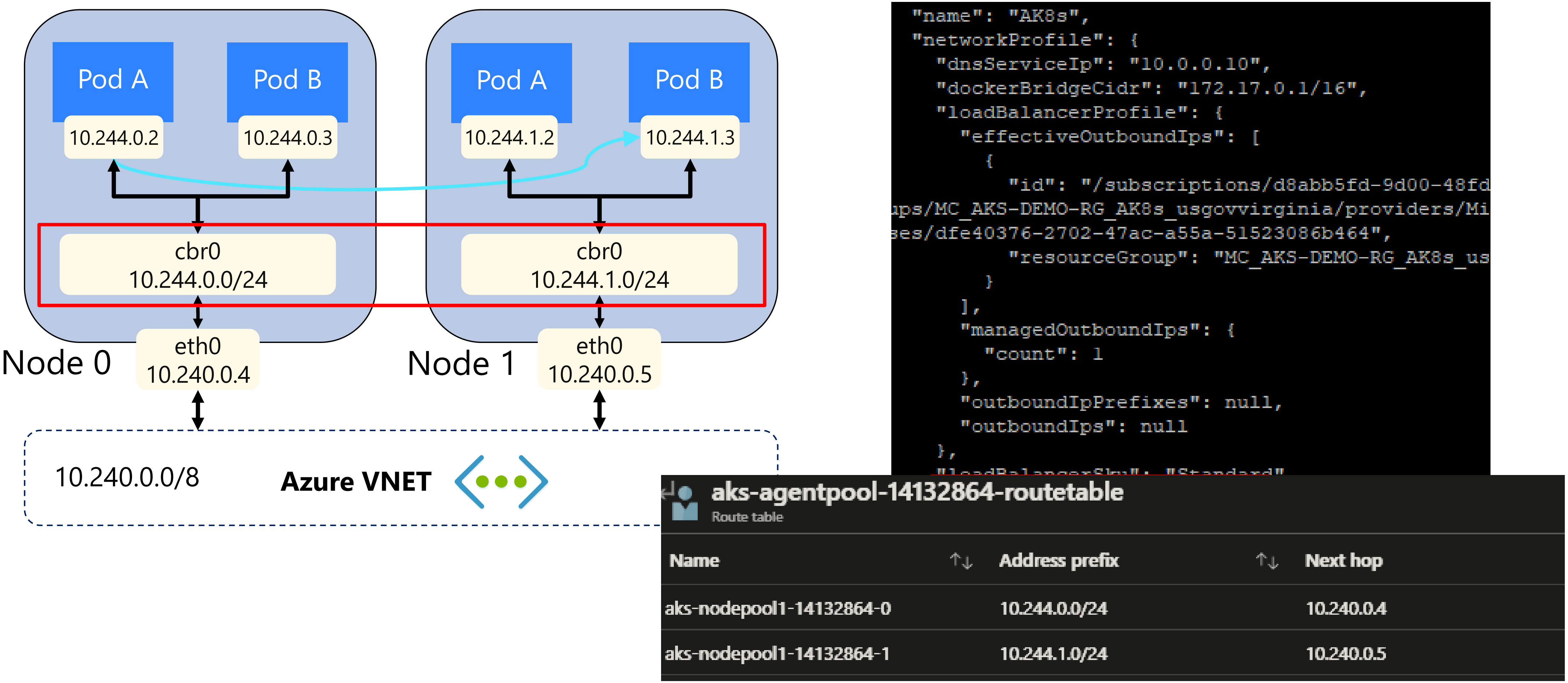
Basic or Kubenet Networking

Options

✓ Basic

- Uses the kubenet Kubernetes plugin.
- Default configuration for AKS
- Deploy into new or existing VNET
- Pods assigned IP from logically different address space to Azure subnet
- Uses NAT to route traffic from Pods to external resources
- Reduces number of reserved IP addresses

Kubenet or Basic Networking



```
 "name": "AK8s",
 "networkProfile": {
   "dnsServiceIp": "10.0.0.10",
   "dockerBridgeCidr": "172.17.0.1/16",
   "loadBalancerProfile": {
     "effectiveOutboundIps": [
       {
         "id": "/subscriptions/d8abb5fd-9d00-48fd
ups/MC_AKS-DEMO-RG_AK8s_usgovvirginia/providers/Mi
ses/dfe40376-2702-47ac-a55a-51523086b464",
         "resourceGroup": "MC_AKS-DEMO-RG_AK8s_us
       }
     ],
     "managedOutboundIps": {
       "count": 1
     },
     "outboundIpPrefixes": null,
     "outboundIps": null
   },
   "loadBalancerSku": "Standard"
```

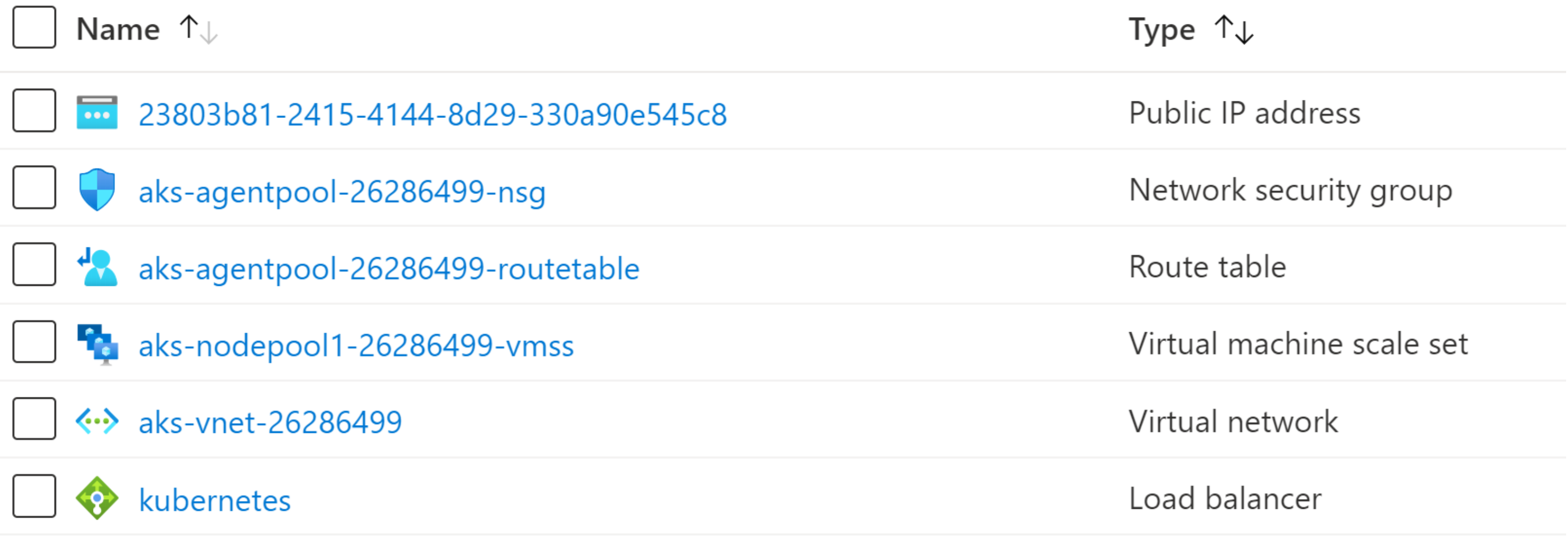
aks-agentpool-14132864-routetable
Route table

Name	Address prefix	Next hop
aks-nodepool1-14132864-0	10.244.0.0/24	10.240.0.4
aks-nodepool1-14132864-1	10.244.1.0/24	10.240.0.5

Node Resource Group

AKS cluster provisioning using defaults

AKS creates and manages the VNet and Subnet addressing, an NSG associated with the AKS subnet, and the route table for UDR and IP forwarding



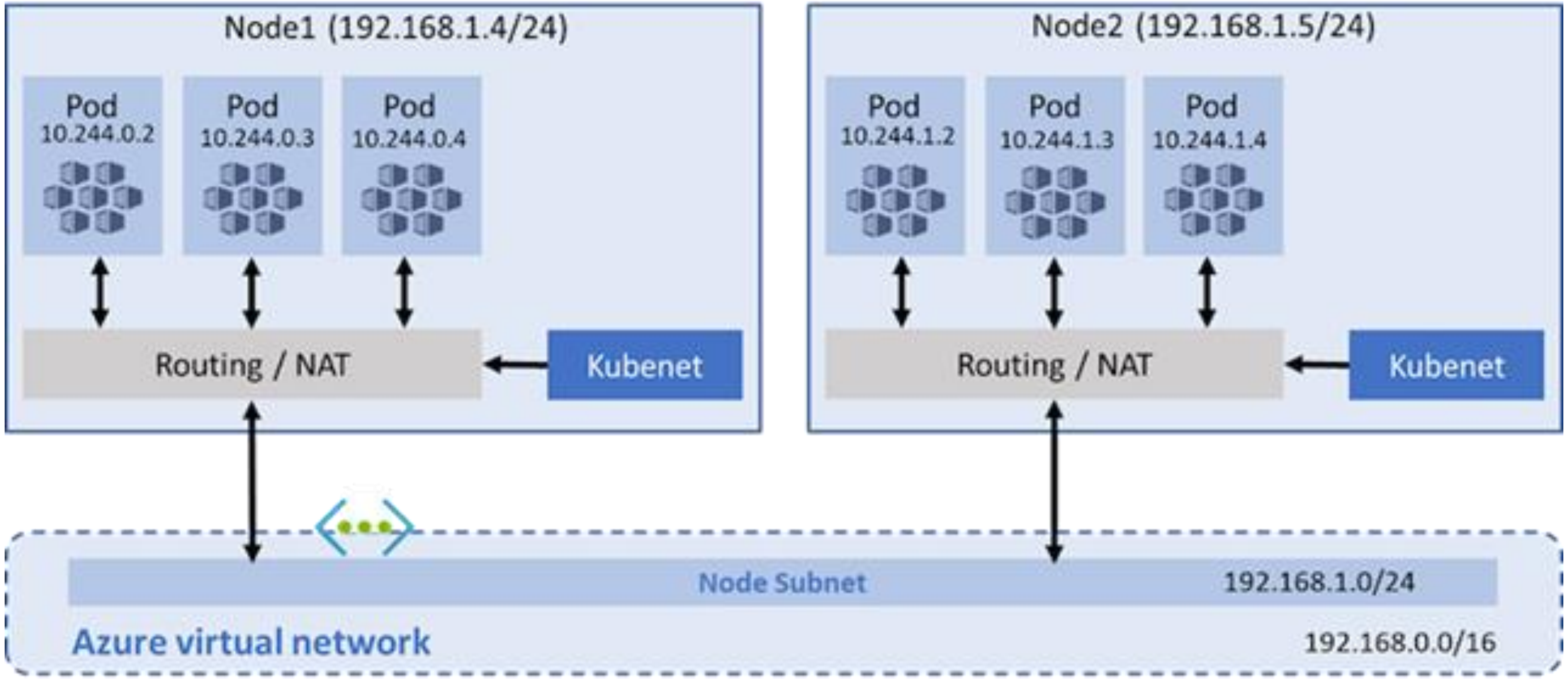
□	Name ↑↓	Type ↑↓
□	23803b81-2415-4144-8d29-330a90e545c8	Public IP address
□	aks-agentpool-26286499-nsg	Network security group
□	aks-agentpool-26286499-routetable	Route table
□	aks-nodepool1-26286499-vmss	Virtual machine scale set
□	aks-vnet-26286499	Virtual network
□	kubernetes	Load balancer

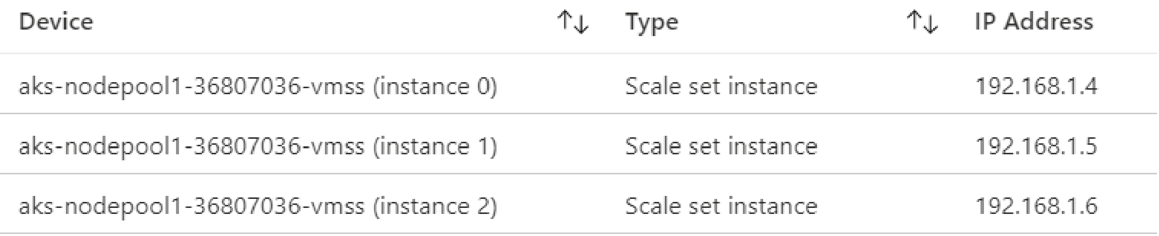
Kubenet Basic Networking with your own IP address

```
az network vnet create \
    --resource-group myResourceGroup \
    --name myAKSVnet \
    --address-prefixes 192.168.0.0/16 \
    --subnet-name myAKSSubnet \
    --subnet-prefix 192.168.1.0/24

az aks create \
    --resource-group myResourceGroup \
    --name myAKSCluster \
    --node-count 3 \
    --network-plugin kubenet \
    --service-cidr 10.0.0.0/16 \
    --dns-service-ip 10.0.0.10 \
    --pod-cidr 10.244.0.0/16 \
    --docker-bridge-address 172.17.0.1/16 \
    --vnet-subnet-id $SUBNET_ID \
    --service-principal d2929ec9-9b53-4dad-8c88-684f68b84817 \
    --client-secret ldpVhnM7v.lLfr.D77NZbWCy_~Yg~LJhuP
```

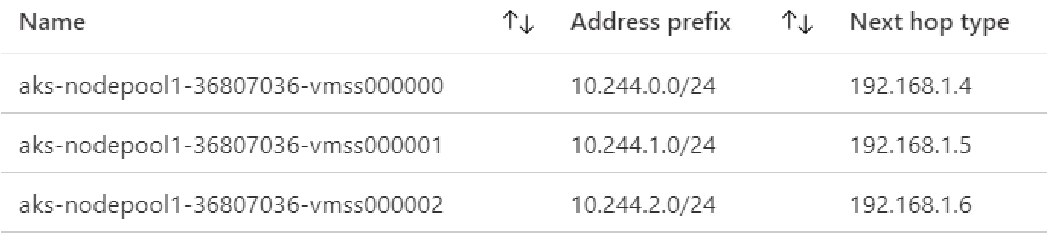
Node Resource Group

Basic Networking





Device ↑↓	Type ↑↓	IP Address
aks-nodepool1-36807036-vmss (instance 0)	Scale set instance	192.168.1.4
aks-nodepool1-36807036-vmss (instance 1)	Scale set instance	192.168.1.5
aks-nodepool1-36807036-vmss (instance 2)	Scale set instance	192.168.1.6



Name ↑↓	Address prefix ↑↓	Next hop type
aks-nodepool1-36807036-vmss000000	10.244.0.0/24	192.168.1.4
aks-nodepool1-36807036-vmss000001	10.244.1.0/24	192.168.1.5
aks-nodepool1-36807036-vmss000002	10.244.2.0/24	192.168.1.6

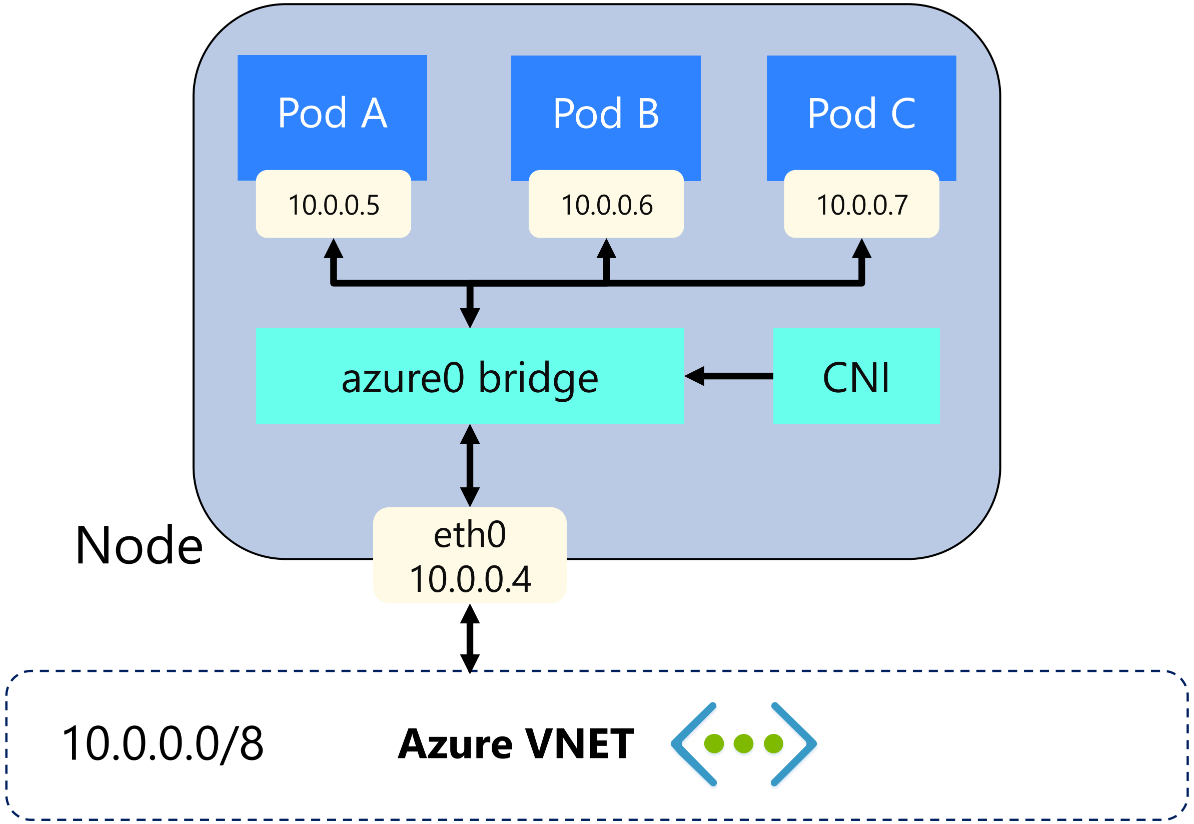
CNI or Advanced Networking

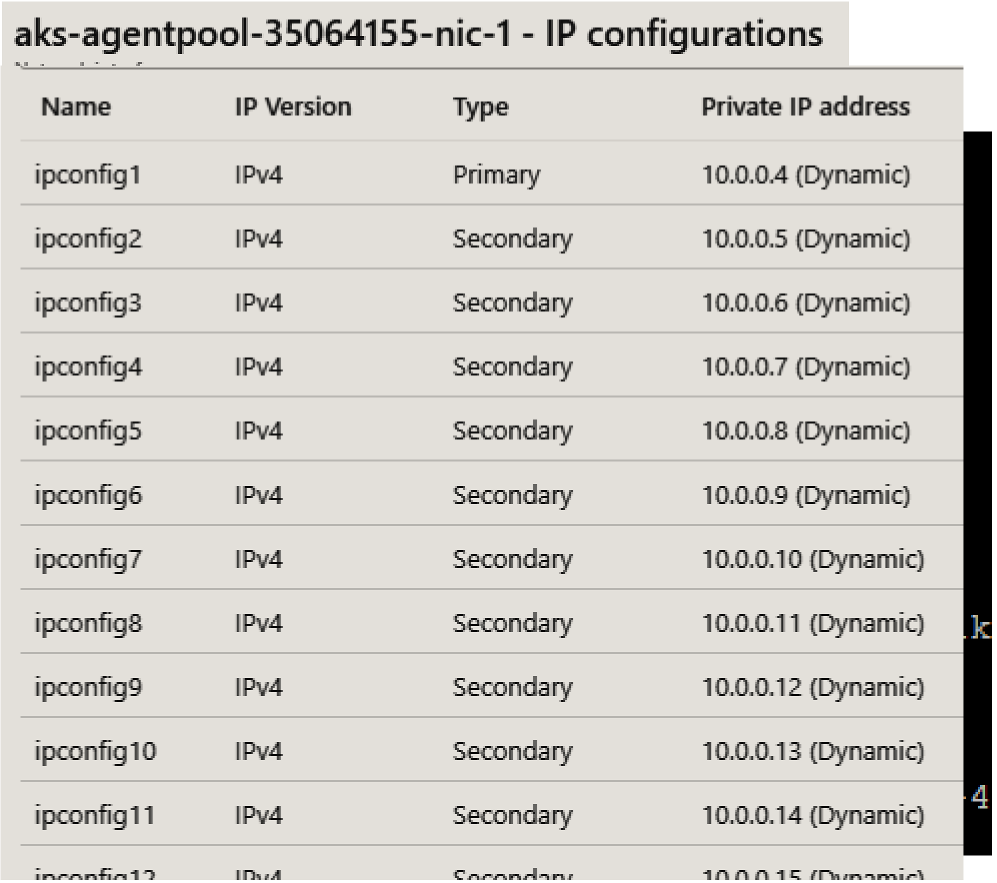
Options

✓ Advanced

- Uses the Azure Container Networking Interface (CNI) Kubernetes plugin
- Assigns Pods an IP address directly from the subnet in Azure
- Provides automatic connectivity to VNet resources
- Requires more IP addresses

Advanced Networking - CNI



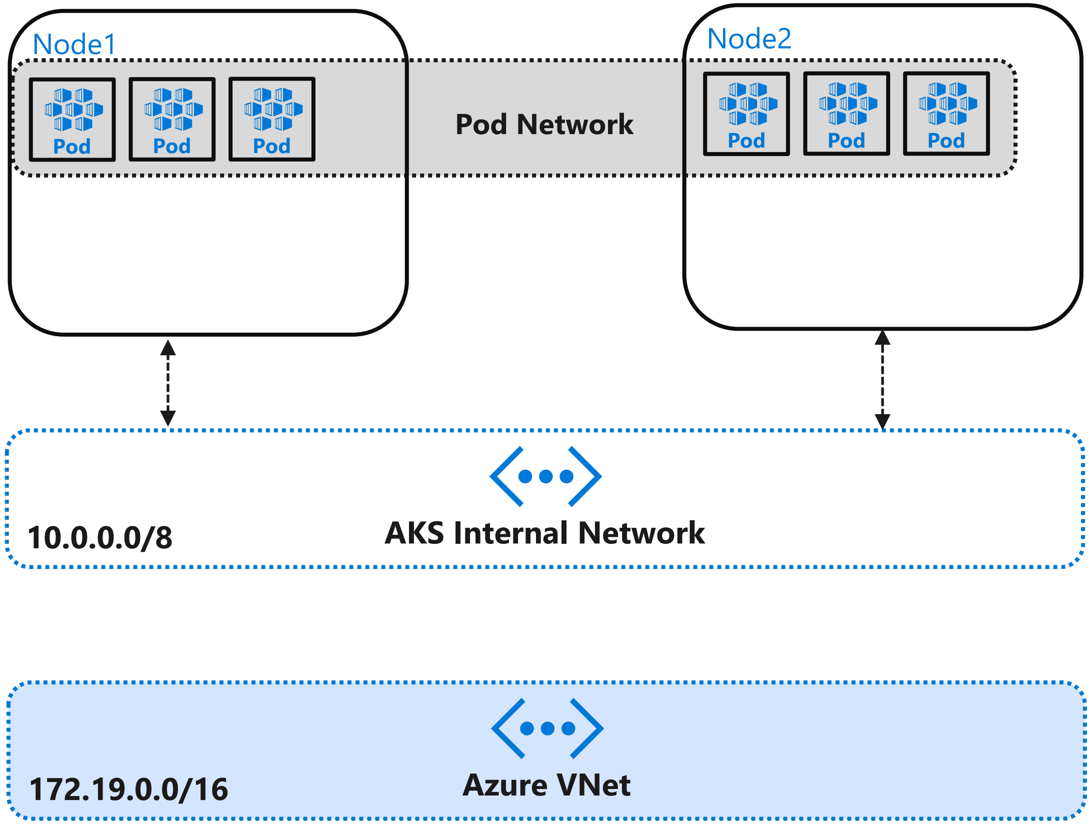


aks-agentpool-35064155-nic-1 - IP configurations

Name	IP Version	Type	Private IP address
ipconfig1	IPv4	Primary	10.0.0.4 (Dynamic)
ipconfig2	IPv4	Secondary	10.0.0.5 (Dynamic)
ipconfig3	IPv4	Secondary	10.0.0.6 (Dynamic)
ipconfig4	IPv4	Secondary	10.0.0.7 (Dynamic)
ipconfig5	IPv4	Secondary	10.0.0.8 (Dynamic)
ipconfig6	IPv4	Secondary	10.0.0.9 (Dynamic)
ipconfig7	IPv4	Secondary	10.0.0.10 (Dynamic)
ipconfig8	IPv4	Secondary	10.0.0.11 (Dynamic)
ipconfig9	IPv4	Secondary	10.0.0.12 (Dynamic)
ipconfig10	IPv4	Secondary	10.0.0.13 (Dynamic)
ipconfig11	IPv4	Secondary	10.0.0.14 (Dynamic)
ipconfig12	IPv4	Secondary	10.0.0.15 (Dynamic)

Basic Networking - Implications

- Managed Network.
- Must manually manage and maintain user-defined routes (UDRs).
- Uses Kubernetes internal or external load balancer to reach pods from outside of the cluster.
- Conserves IP address space.
- Appropriate for deployments that do not require custom VNet integration.

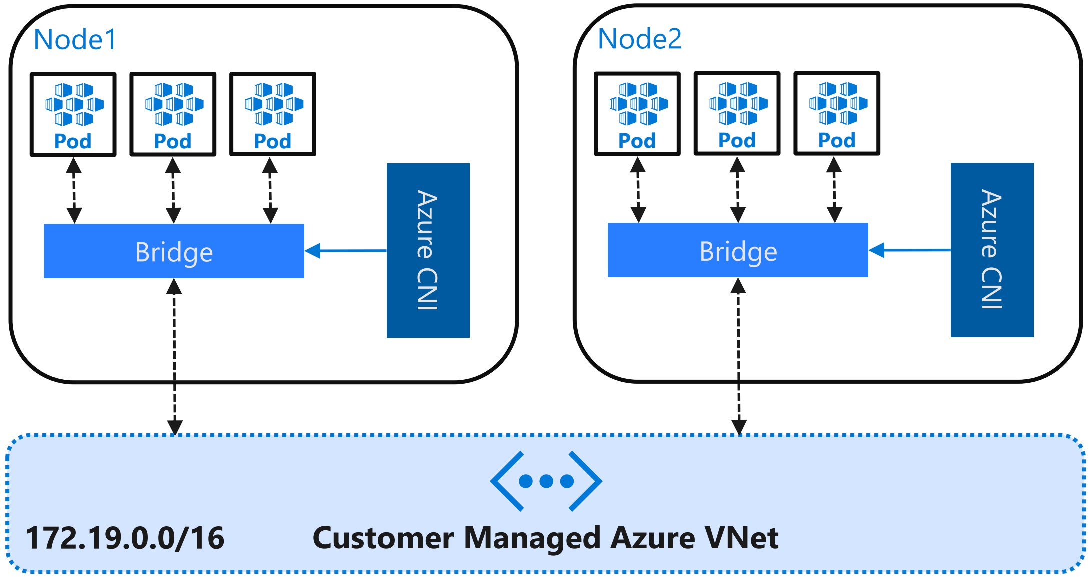


Advanced Networking - Implications

- Advanced networking places pods in an Azure Virtual Network (VNet) that you configure.

- Provides automatic connectivity to VNet resources and integration with the rich set of VNet capabilities.

- Requires more IP address space.



Summary

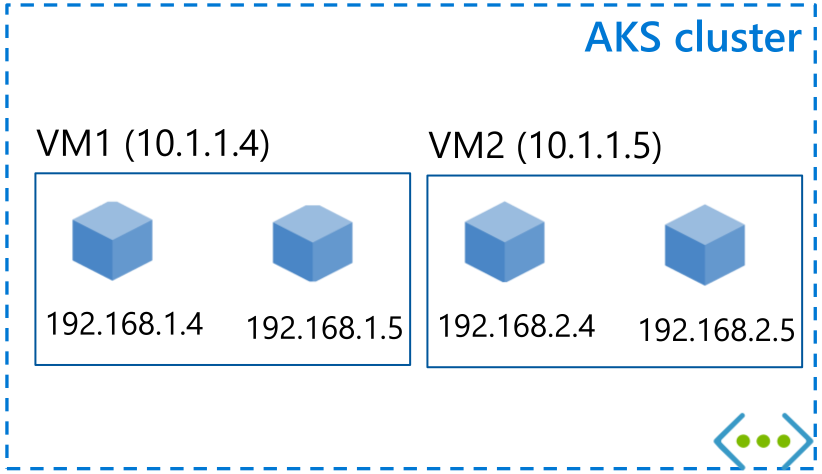
Basic

- ✓ Pods on internal bridge network
- ✓ Conserves IP space
- ✓ UDR based routing

Advanced

- ✓ Deploy into an existing VNET
- ✓ Pods direct attach to subnet
- ✓ Access all VNet resources

New VNet

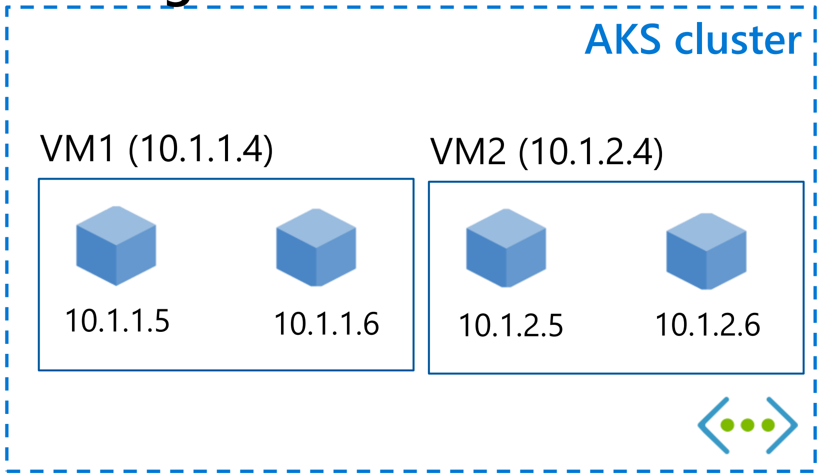


Basic

Dest	Next Hop
192.168.1.4/24	10.1.1.4
192.168.2.4/24	10.1.1.5

UDR

Existing VNet



Advanced

Azure CNI

Azure CNI plugin present on every Kubernetes node. Assigns an IP address directly from the VNET/Subnet

Which Networking Model do I choose?

A balance between flexibility and advanced configuration needs

✓ Decision required

Use *kubenet* when:

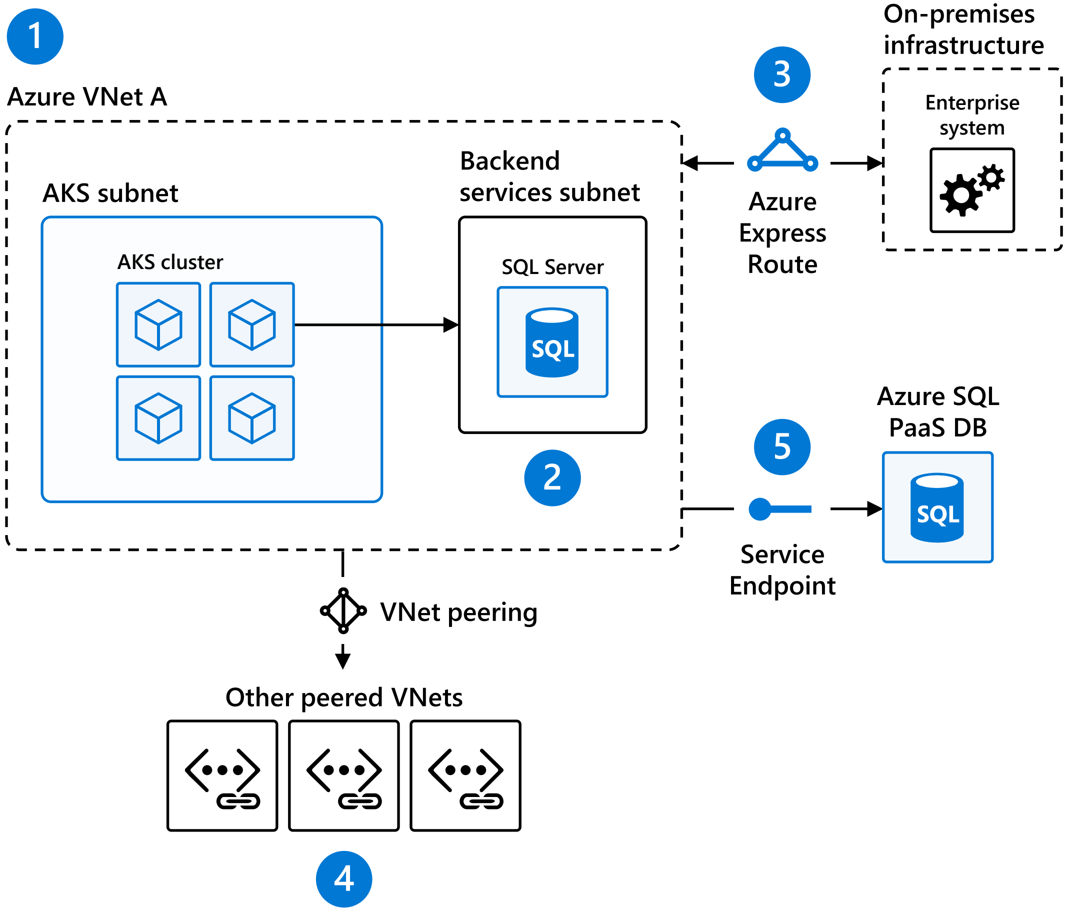
- You have limited IP address space.
- Pod communication is predominantly within the cluster.
- You don't need advanced AKS features such as virtual nodes or Azure Network Policy. Use Calico network policies.

Use *Azure CNI* when:

- You have available IP address space.
- Pod communication is to resources outside of the cluster.
- You don't want to manage UDRs for pod connectivity.
- You need AKS advanced features such as virtual nodes or Azure Network Policy. Use Calico network policies.

Scenarios enabled by Advanced Networking

1. Uses Azure subnet for both your containers and cluster VMs
2. Connectivity to existing Azure services in the same VNet
3. Express Route to on-premises infrastructure
4. VNet peering
5. Privately access to other Azure resources using VNet endpoints

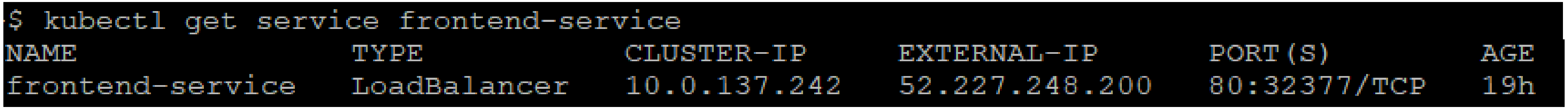


Microsoft

Kubernetes Services

Motivation behind Services

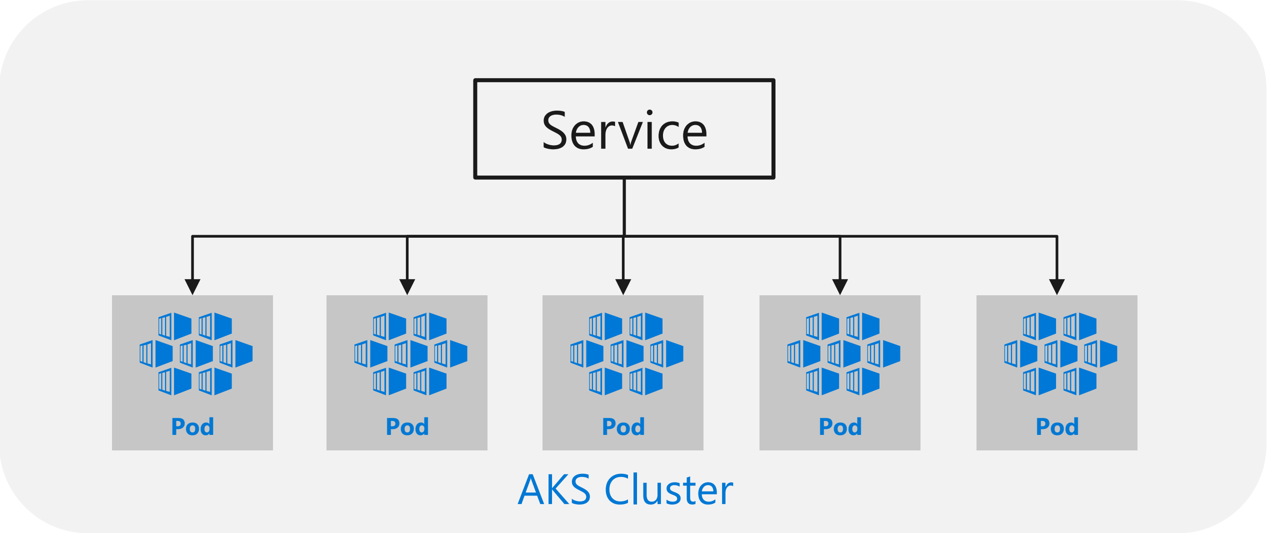
- Services logically group pods to allow for direct access via an IP address or DNS name and on a specific port
- Services provide for cluster ingress and egress.
- Pods can communicate to any other Pod by IP
- Pods are mortal and IPs can change, so Services provide stable communication regardless of IP

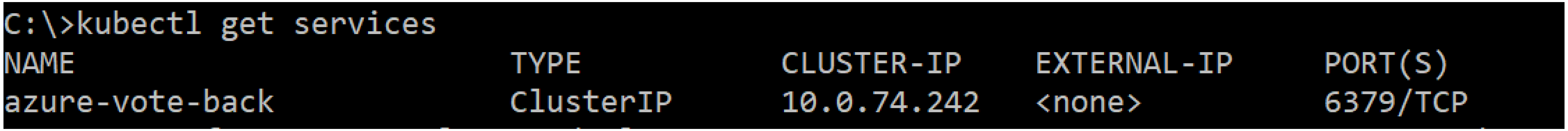


```
$ kubectl get service frontend-service
NAME               TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
frontend-service   LoadBalancer   10.0.137.242   52.227.248.200   80:32377/TCP   19h
```

Exposing Services

Inside the cluster (ClusterIP)

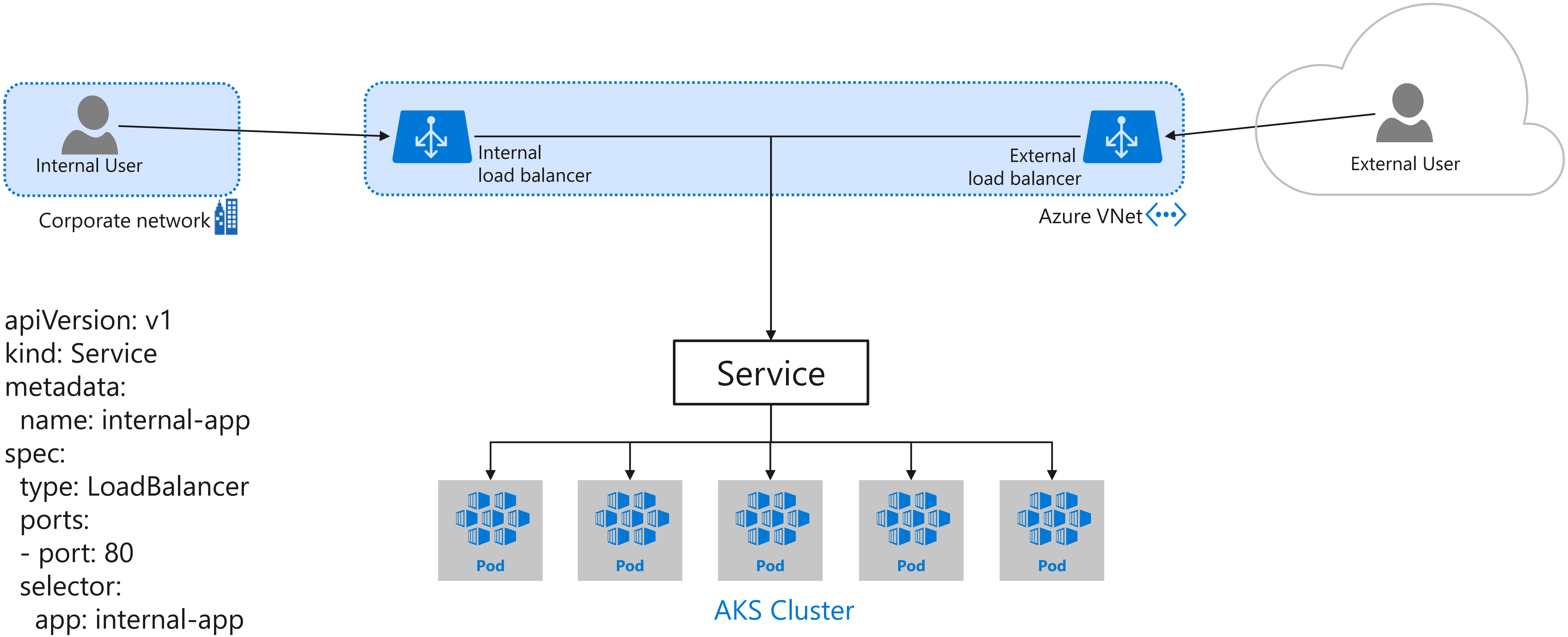




```
C:\>kubectl get services
NAME                        TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)
azure-vote-back             ClusterIP      10.0.74.242   <none>          6379/TCP
```

Exposing Services

Outside the cluster – internal or external network (layer 4)

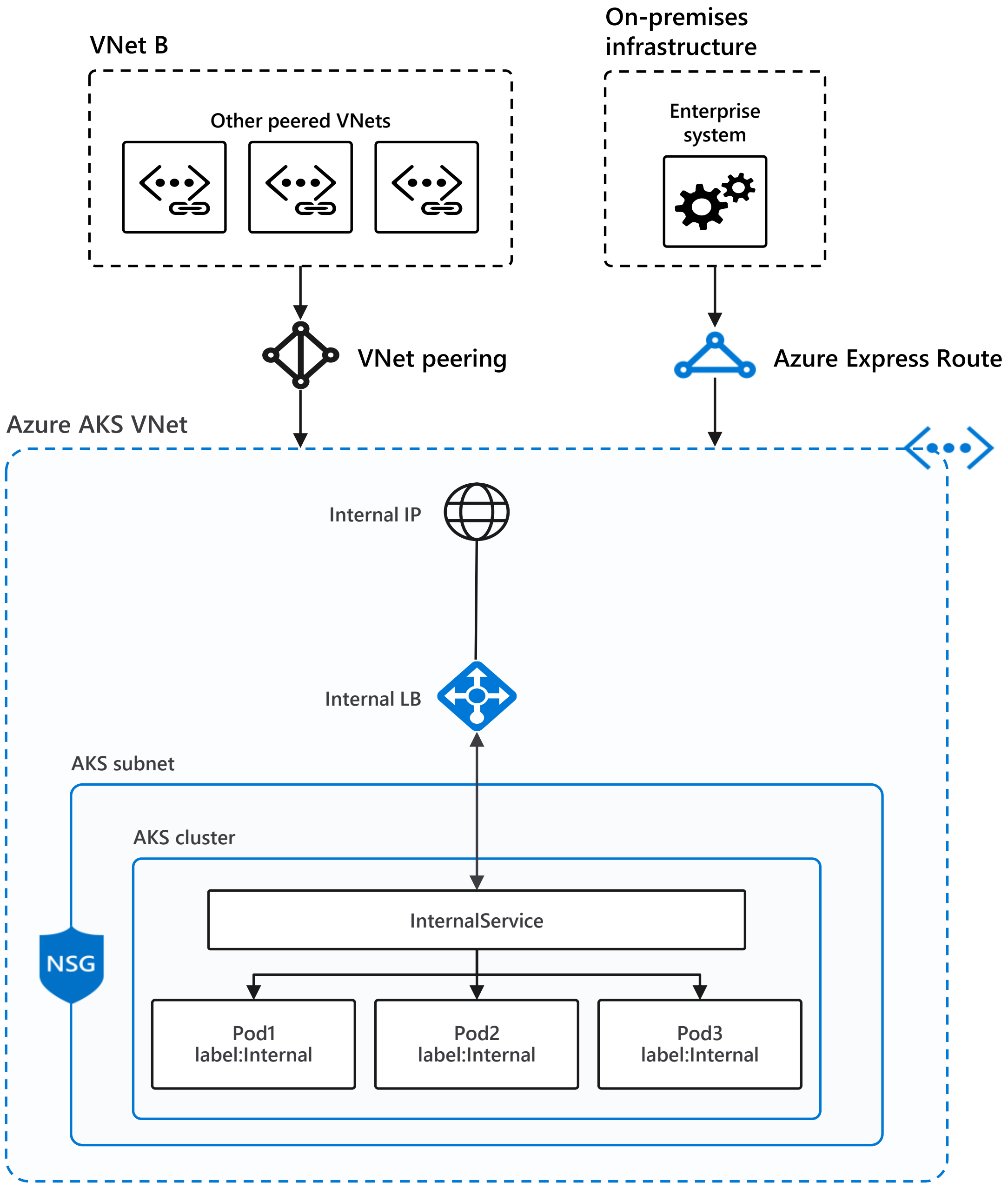


```
apiVersion: v1
kind: Service
metadata:
  name: internal-app
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: internal-app
```

Service – Internal Network

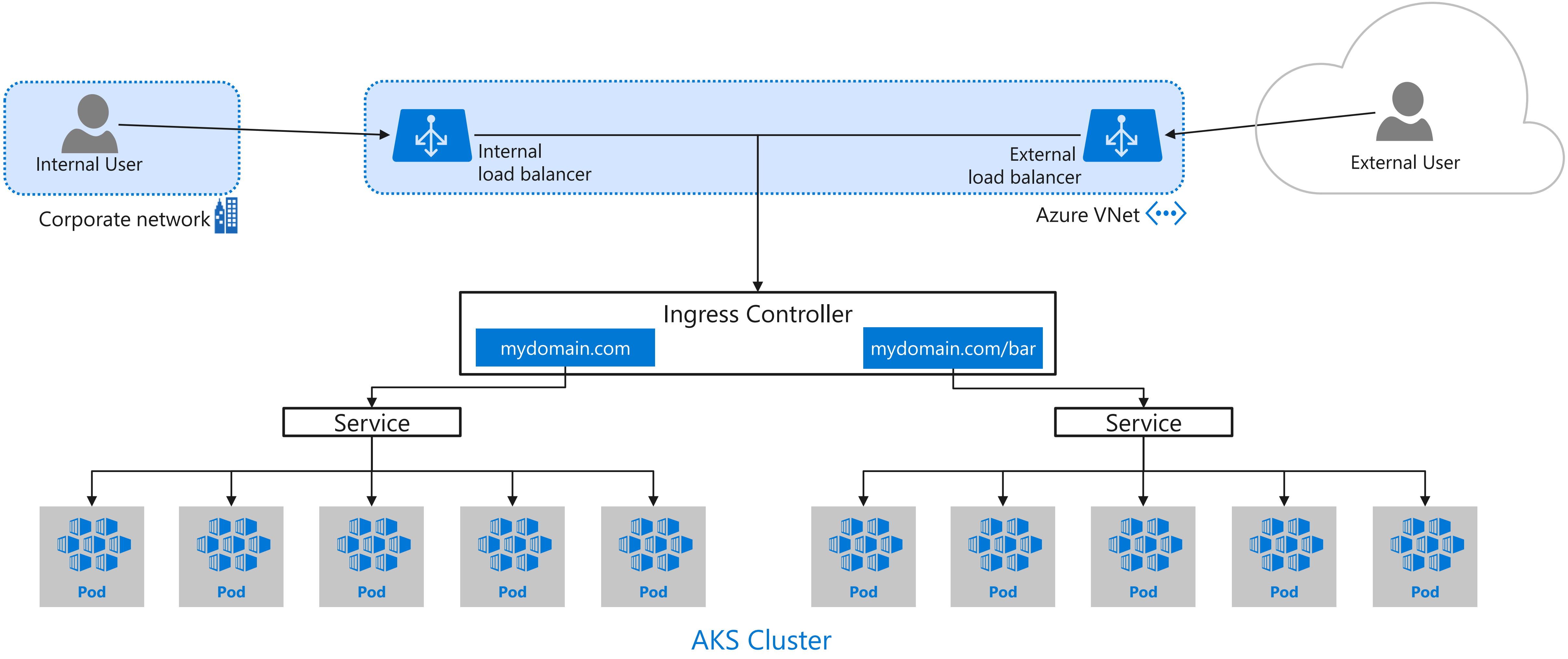
- Used for internal services that should be accessed by other VNETs or On-Premise only

```
apiVersion: v1
kind: Service
metadata:
  name: internalservice
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal: "true"
spec:
   type: LoadBalancer
   loadBalancerIP: 10.240.0.25
   ports:
   - port: 80
   selector:
    app: internal
```

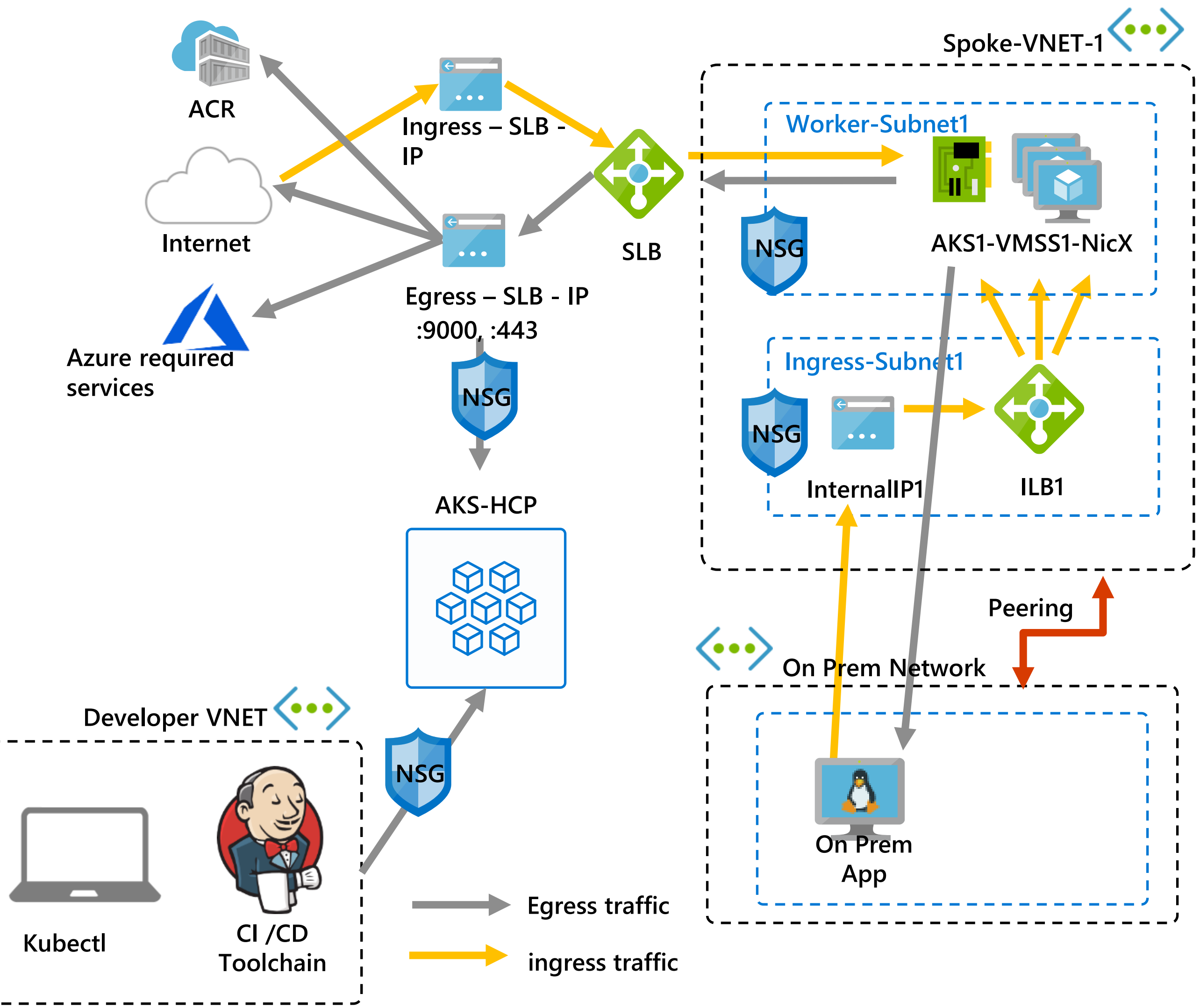


Exposing Services

Ingress Controller



Understanding Ingress and Egress flows in AKS

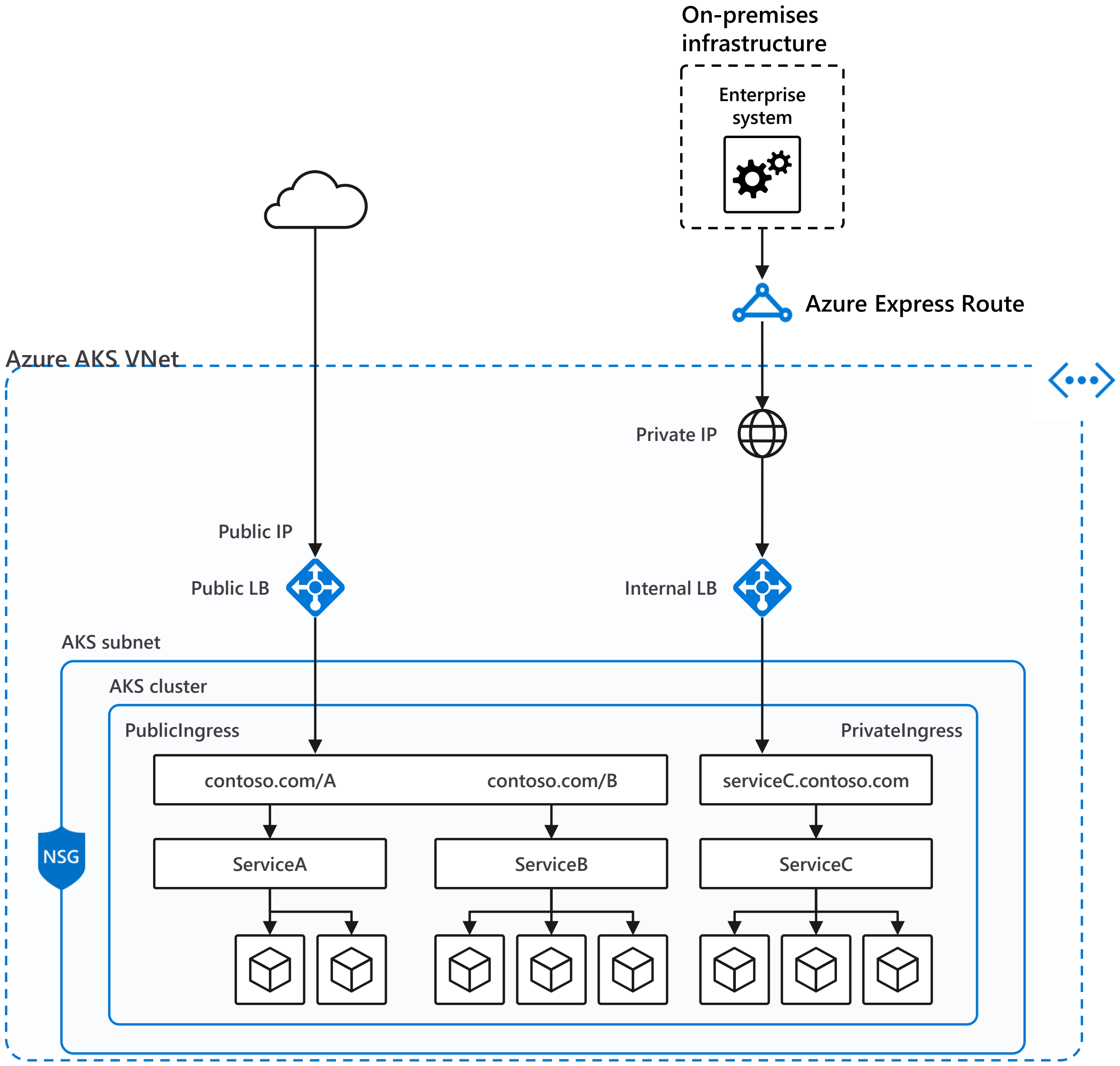


Microsoft

Managing Ingress & Egress

Ingress

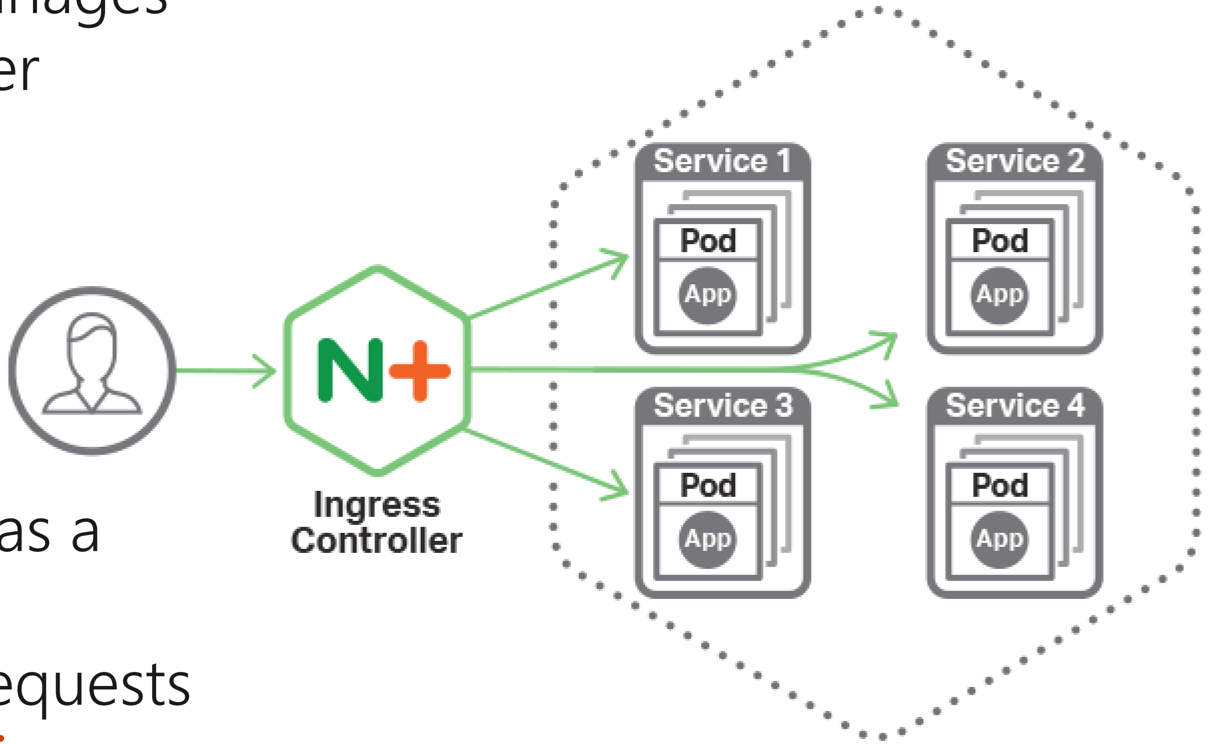
```
kind: Ingress
metadata:
 name: contoso-ingress
   annotations: kubernetes.io/ingress.class: ”PublicIngress"
spec:
 tls:
 - hosts:
   - contoso.com
   secretName: contoso-secret
 rules:
   - host: contoso.com
     http:
      paths:
      - path: /a
        backend:
         serviceName: servicea
         servicePort: 80
      - path: /b
        backend:
         serviceName: serviceb
         servicePort: 80
```



Ingress and Ingress Controllers

- Ingress is a Kubernetes API object that manages external access to the services in the cluster
  - Supports HTTP and HTTPs
  - Path and Subdomain based routing
  - SSL Termination

- Ingress controller is a daemon, deployed as a Kubernetes Pod, that watches the Ingress Endpoint for updates. Its job is to satisfy requests for ingresses. Most popular one being Nginx.

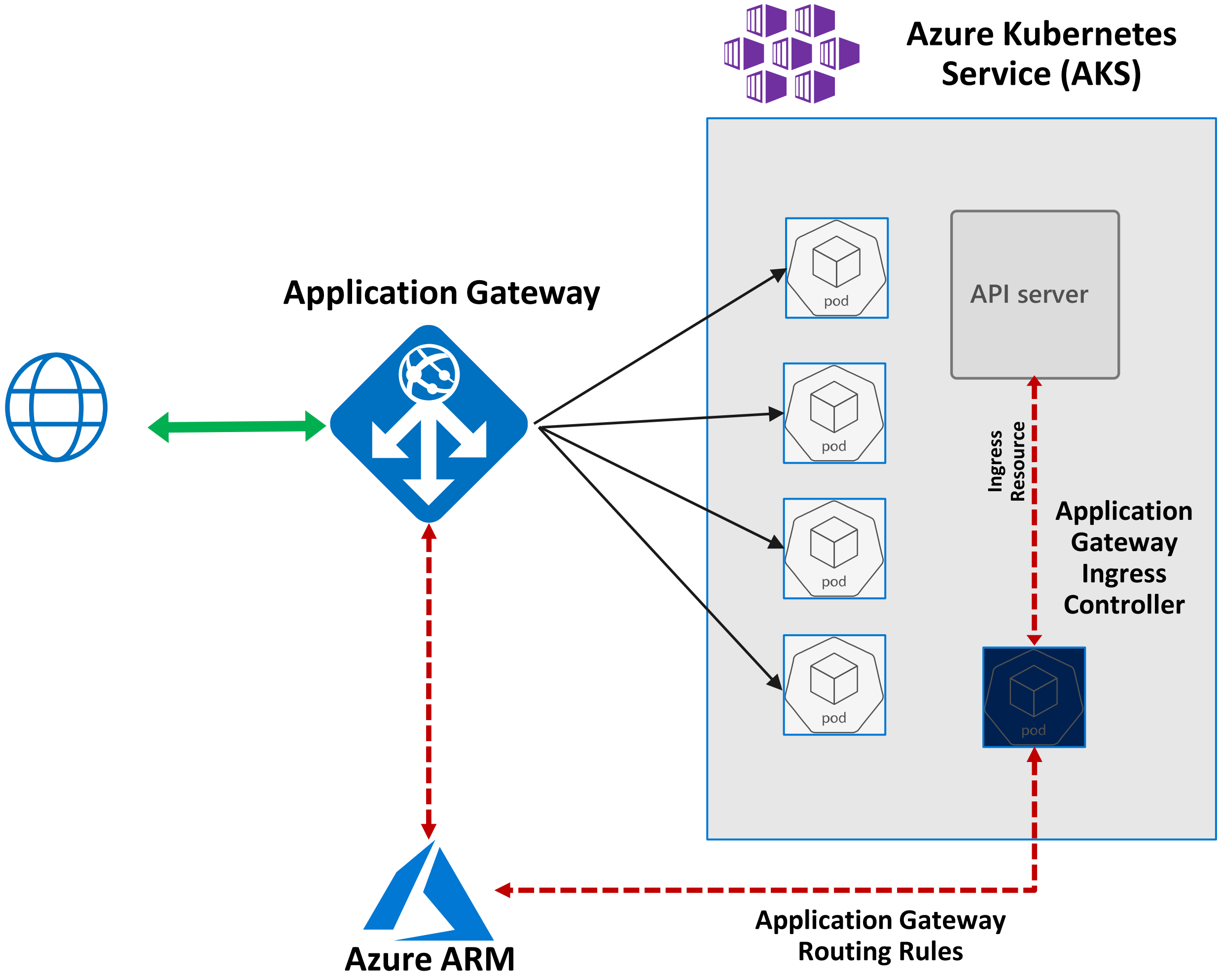


Azure App Gateway Ingress

- Layer 7 web traffic load balancer that enables you to manage traffic to your web applications.
- Exposes applications using native App Gateway
- Monitors Kubernetes cluster to continuously update App Gateway services exposed externally
- Provides
  - URL/subdomain routing
  - End to end SSL termination

Application Gateway Ingress Controller

- Attach Application Gateways to AKS Clusters
- Load Balance from the Internet to pods
- URL routing
- Cookie-based affinity
- SSL termination
- End-to-end SSL
- Support for public, private, and hybrid web sites
- Integrated web application firewall


Azure Kubernetes Service (AKS)
Application Gateway
pod
pod
pod
pod
API server
Ingress Resource
Application Gateway Ingress Controller
pod
Azure ARM
Application Gateway Routing Rules

Enabling Application Gateway Ingress Controller

1. Create a resource group

2. Create a new AKS cluster

```
az aks create -n myCluster -g myResourceGroup --network-plugin azure --enable-managed-identity
```

3. Create a new Application Gateway

```
az network public-ip create -n myPublicIp -g MyResourceGroup --allocation-method Static --sku Standard
```

```
az network vnet create -n myVnet -g myResourceGroup --address-prefix 11.0.0.0/8 --subnet-name mySubnet --subnet-prefix 11.1.0.0/16
```

```
az network application-gateway create -n myApplicationGateway -l canadacentral -g myResourceGroup --sku Standard_v2 --public-ip-address myPublicIp --vnet-name myVnet --subnet mySubnet
```

4. Enable the AGIC add-on in the existing AKS cluster using the existing Application Gateway

```
appgwId=$(az network application-gateway show -n myApplicationGateway -g myResourceGroup -o tsv --query "id")
```

```
az aks enable-addons -n myCluster -g myResourceGroup -a ingress-appgw --appgw-id $appgwId
```

5. Peer the Application Gateway VNet with the AKS cluster VNet

Egress

By default, AKS clusters have unrestricted outbound (egress) internet access.

- The cluster has outbound or egress dependencies.
- For management and operational purposes, nodes in an AKS cluster need to access certain ports and fully qualified domain names (FQDNs).
- These endpoints are required for the nodes to communicate with the API server, or to download and install core Kubernetes cluster components and node security updates.
- The lack of static addresses means that Network Security Groups can't be used to lock down the outbound traffic from an AKS cluster.

Standard Load Balancer

Default SKU when deploying a new cluster.

Standard Public IP address assigned to SLB so that the worker nodes can communicate with the API server, but it is only for allowing outbound Internet access.

SLB is a Layer4 LB that supports both inbound and outbound scenarios.

Secure By Default: This means SLB and Standard public IP addresses are closed to inbound flows unless opened by Network Security Groups.

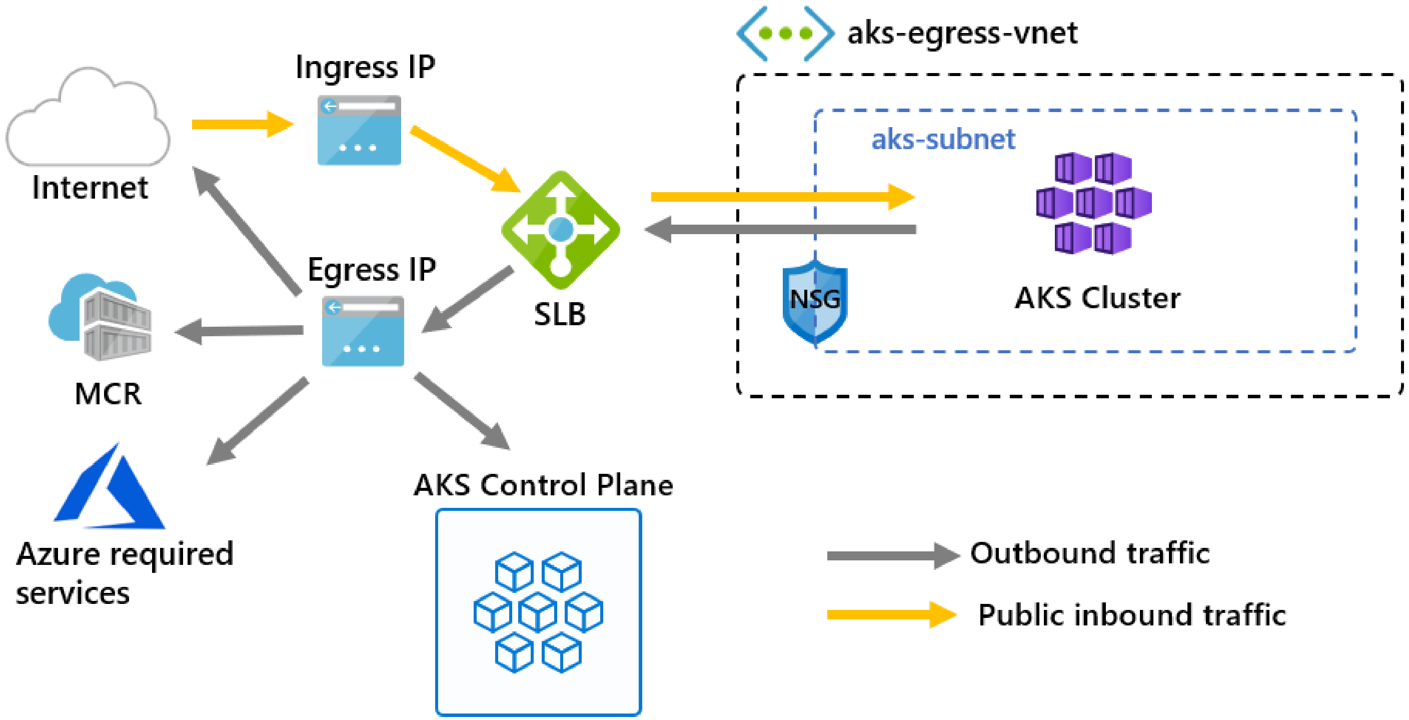
Supports larger backend pool size (1000 vs 100) vs Basic SKU

Supports Multiple Node Pools, Availability Zones

Standard Load Balancer

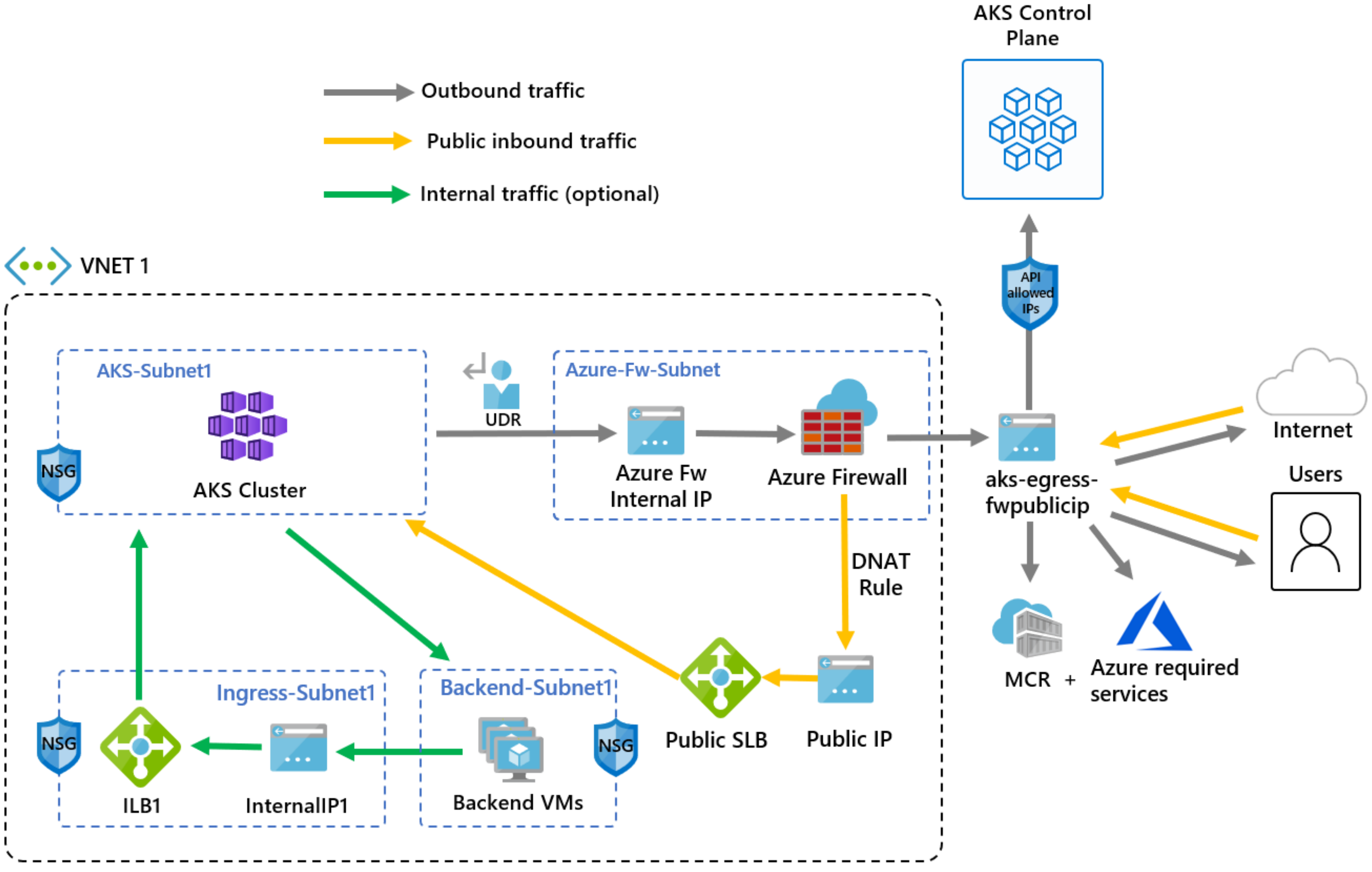
```
apiVersion: v1
kind: Service
metadata:
  name: public-svc
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: public-app
```

- **Network topology deployed in AKS clusters by default**



Custom Egress with Azure Firewall

- Outbound Type is set to User Defined Routing (UDR)
- az aks create …. –outbound-type userDefinedRouting



Microsoft

Kubernetes Network Policy

Network Pod Communication Defaults

- All Pods are non-isolated by default, they accept traffic from any source.
- Flat network. All pods can talk to other pods
- Accept traffic from anyone.
- Multi stage/zone project could expose security risks.
  - 3 tier webapp.
  - Front end could technically talk to DB

Network Policies

- **Rules that control the flow of traffic**
- **Uses labels to select Pods and create rules to specify allowed traffic**
- **Two options:**
  - **Azure Network Policies – CNI**
  - **Calico – Open Source**
- **Defined at cluster creation (cannot be changed)**

Network Policy Manifest

- Pod Selector
- PolicyTypes
    - Ingress, egress

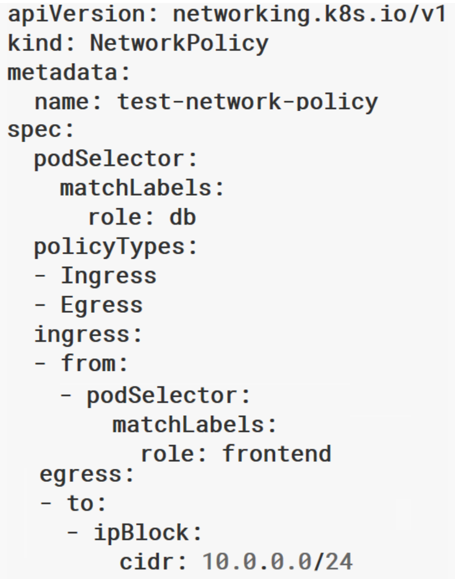
Ingress

- namespaceSelector
- podSelector
- ipBlock

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: test-network-policy
 namespace: default
spec:
 podSelector:
  matchLabels:
   role: db
 policyTypes:
 - Ingress
 - Egress
 ingress:
 - from:
  - ipBlock:
    cidr: 172.17.0.0/16
    except:
    - 172.17.1.0/24
  - namespaceSelector:
    matchLabels:
     project: myproject
  - podSelector:
    matchLabels:
     role: frontend
  ports:
  - protocol: TCP
   port: 6379
 egress:
 - to:
  - ipBlock:
    cidr: 10.0.0.0/24
  ports:
  - protocol: TCP
   port: 5978
```

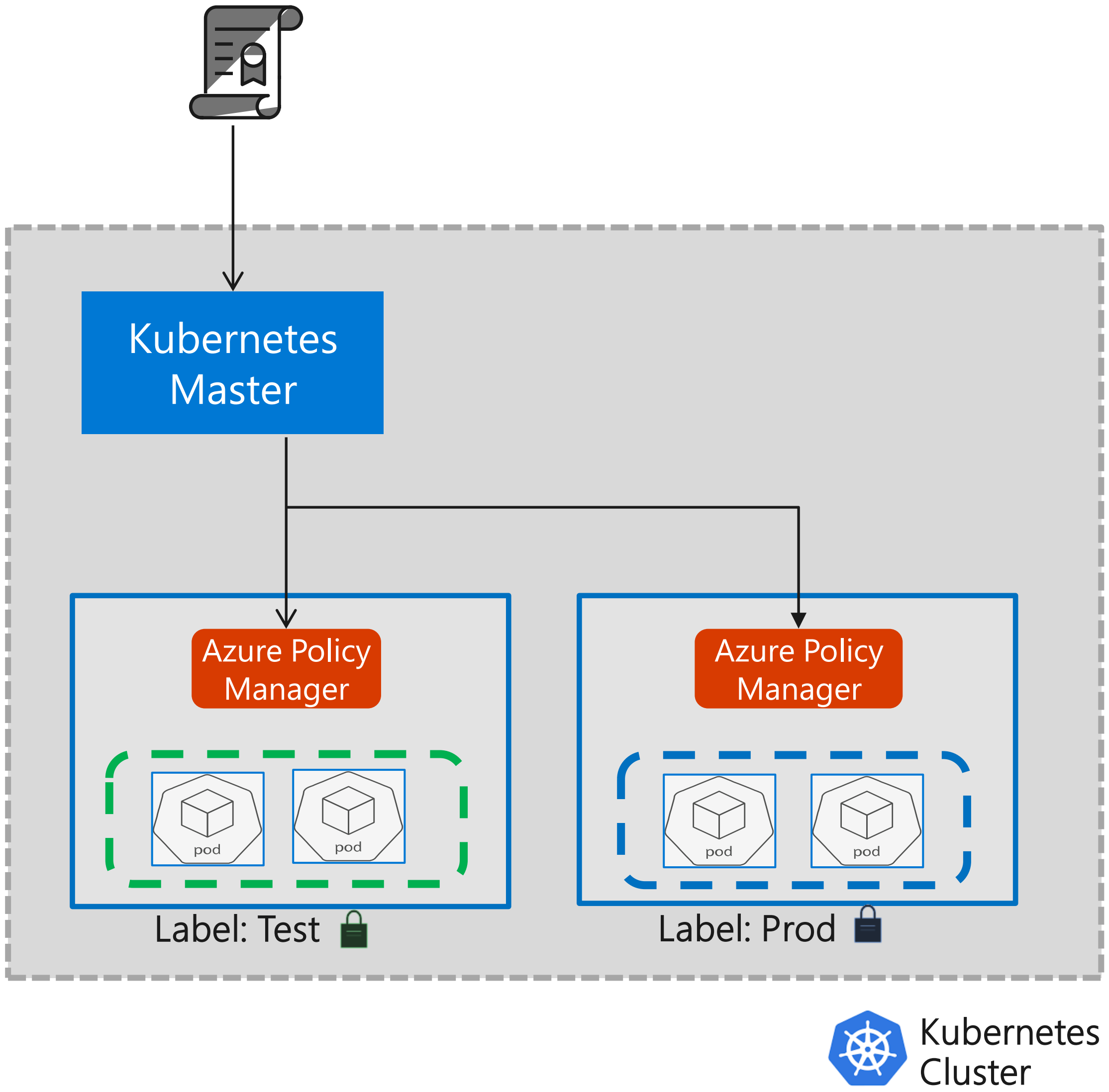
Azure Kubernetes Network Policies

- Micro-segmentation for containers – like NSGs for VMs
- Label-based selection of Pods
- Policy resource yaml file specifies Ingress and Egress policies
- Works in conjunction with Azure CNI



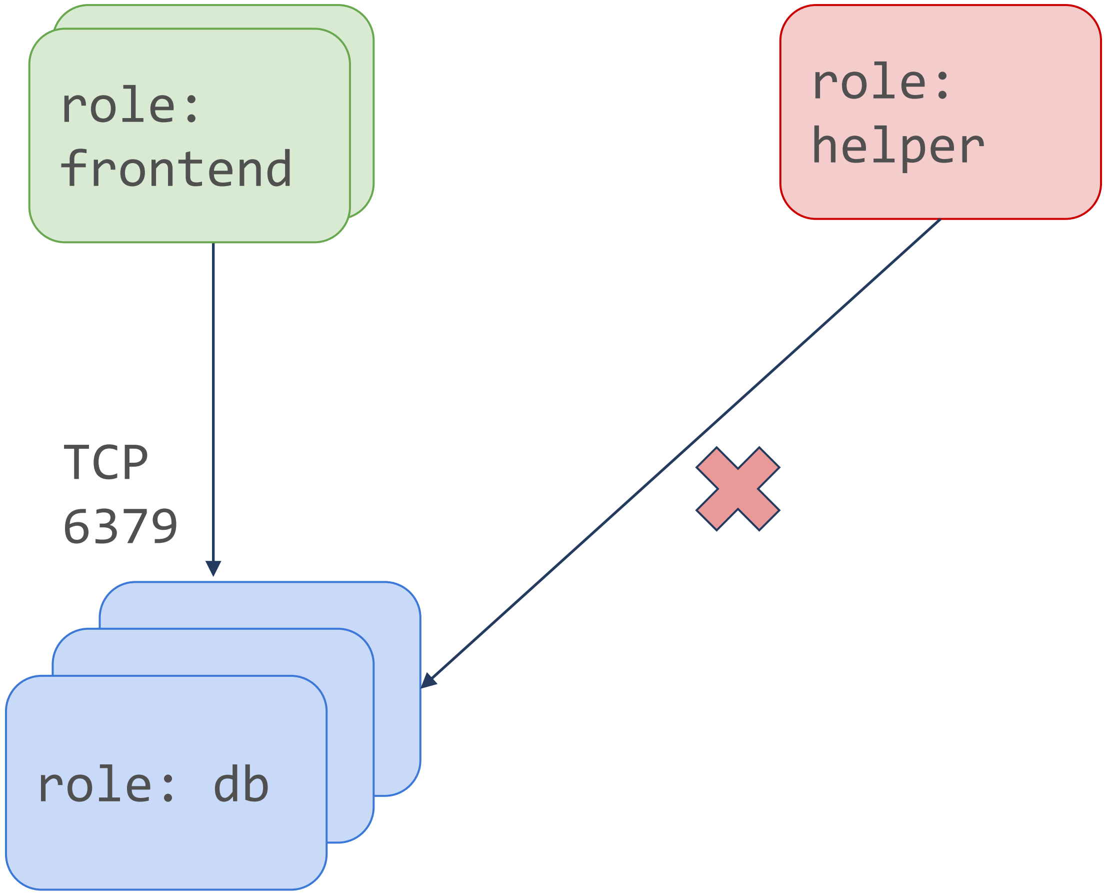
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
```

```
kubectl apply -f policy.yaml
```



Kubernetes Network Policy Example

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```



Network Policy Recommendations

Create default policies that apply to all pods

- Default deny all ingress traffic
- Default deny all egress traffic.
- Allow Specific Traffic (Ingress & Egress)

When to use Azure NSG vs. Kubernetes Network policy?

- Use Azure NSG to filter North-South traffic, that is, traffic entering and leaving your cluster subnet
- Use Kubernetes Network Policies to filter East-West traffic, that is, traffic between pods in your cluster

Let’s Create A Network Policy

- Deny all traffic to pod.
- Allow traffic based on pod labels.
- Allow traffic based on namespace.

- Enabled policy at cluster creation. You can't enable network policy on an existing AKS cluster.

- Use the Azure CNI plug-in.

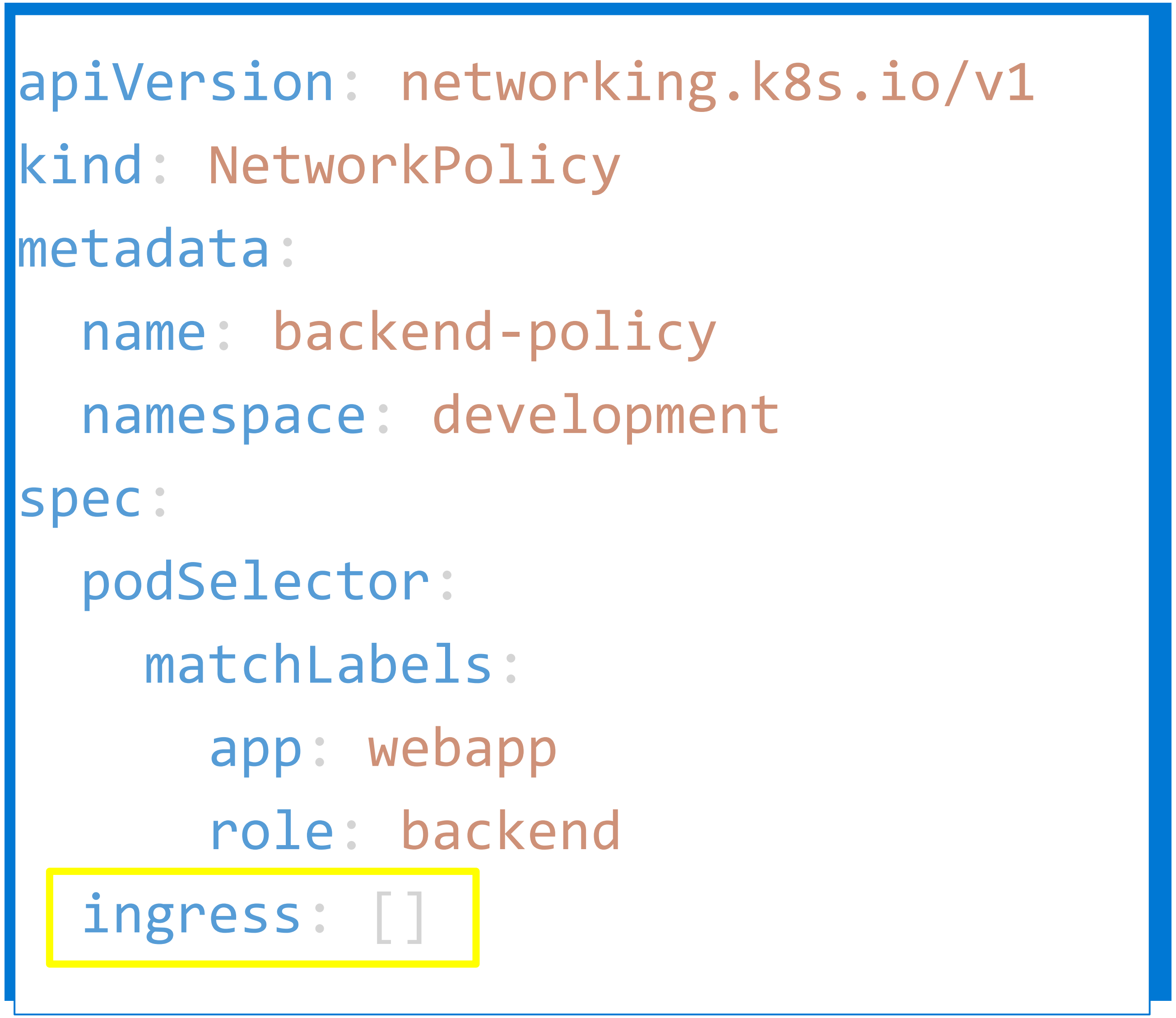
Process for enabling the cluster:

- Create a virtual network and subnet.
- Create an Azure Active Directory (Azure AD) service principal for use with the AKS cluster (could also use managed identities).
- Assign *Contributor* permission for service principal on the virtual network.
- Creates an AKS cluster in the defined virtual network and enables network policy.

Test Default Node Connectivity

```
kubectl create namespace development
kubectl label namespace/development purpose=development

kubectl run backend --image=nginx --labels app=webapp,role=backend --namespace development --
expose --port 80 --generator=run-pod/v1

        kubectl get pods --namespace=development
        NAME      READY   STATUS    RESTARTS   AGE
        backend    1/1    Running    0         71m

kubectl run --rm -it --image=alpine network-policy --namespace development --generator=run-pod/v1

wget -qO- http://backend

        <!DOCTYPE html>
        <html>
        <head>
        <title>Welcome to nginx!</title>
        [...]
```

Create a Policy to Deny Traffic



```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: backend-policy
  namespace: development
spec:
  podSelector:
    matchLabels:
      app: webapp
      role: backend
  ingress: []
```

This manifest uses a **podSelector** to attach the policy to pods that have the **app:webapp,role:backend** label, like our NGINX pod.

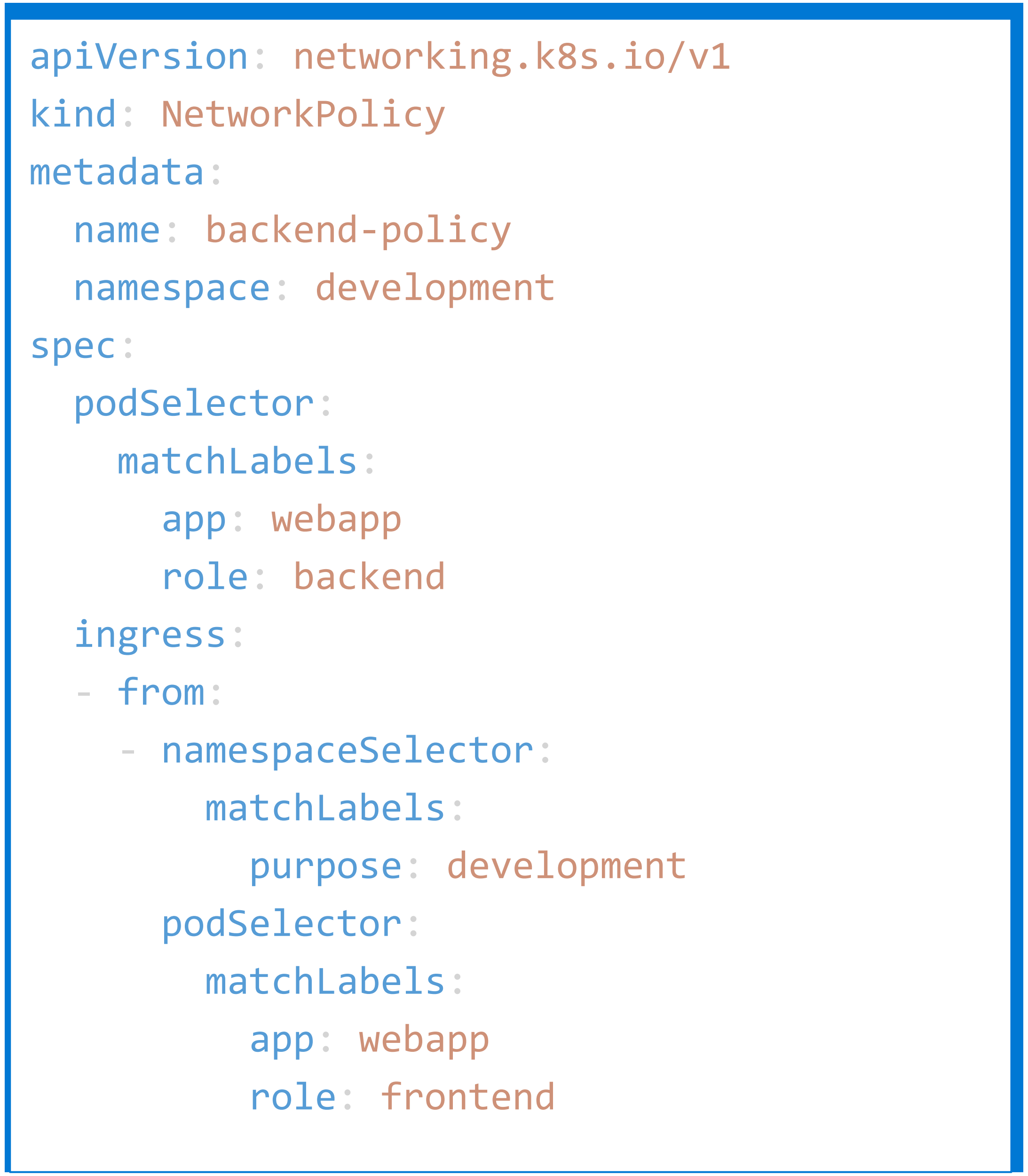
No rules are defined under **ingress**, so all inbound traffic to the pod is denied

Allow inbound traffic based on a pod label

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: backend-policy
  namespace: development
spec:
  podSelector:
    matchLabels:
      app: webapp
      role: backend
  ingress:
  - from:
    - namespaceSelector: {}
      podSelector:
        matchLabels:
          app: webapp
          role: frontend
```

Updated network policy to allow traffic from pods with the labels *app:webapp,role:frontend* and in any namespace.

Allow traffic only from within a defined namespace



```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: backend-policy
  namespace: development
spec:
  podSelector:
    matchLabels:
      app: webapp
      role: backend
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: development
      podSelector:
        matchLabels:
          app: webapp
          role: frontend
```

This network policy uses a **namespaceSelector** and a **podSelector** element for the ingress rule.

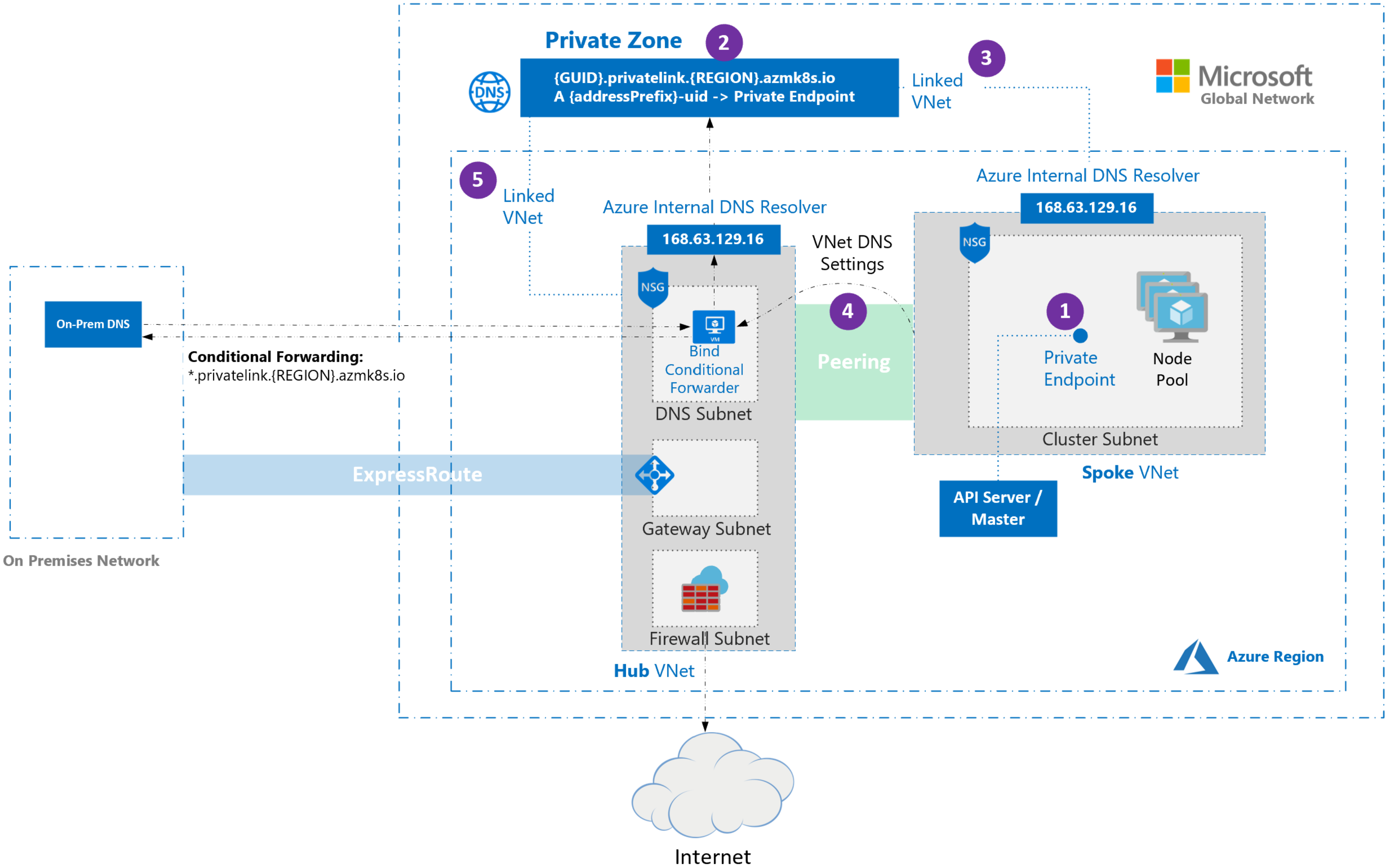
In this example, both elements must match for the ingress rule to be applied.

Private AKS Cluster

- Control plane or API server has internal IP addresses
- Network traffic between your API server and your node pools remains on the private network only
- API server and the cluster or node pool can communicate with each other through the Azure Private Link service in the API server virtual network and a private endpoint that's exposed in the subnet of the customer's AKS cluster

```
az aks create \
    --resource-group <private-cluster-resource-group> \
    --name <private-cluster-name> \
    --load-balancer-sku standard \
    --enable-private-cluster \
    --network-plugin azure \
    --vnet-subnet-id <subnet-id> \
    --docker-bridge-address 172.17.0.1/16 \
    --dns-service-ip 10.2.0.10 \
    --service-cidr 10.2.0.0/24
```

Example Hub n Spoke for Private Cluster



Microsoft

Let’s Summarize the Networking Decisions we need to make …

Networking Components Decisions

✓ Decision required

Type of Network	Basic(Kubenet) or Advances(Azure CNI)
Maximum Pods per Node	Basic: minimal 30, maximum 110 Advanced: minimal 30, maximum 250
Service Types	• ClusterIP – to microservices pods • Internal or External Load Balancer – Ingress Controller only • NodePort – to integrate with external services, if required.
Type of Ingress Controller	Azure App Gateway, Nginx, Traefik, Ambassador And single Ingress per cluster or per namespace/pool
Type of Load Balancer	• Internal LB (requires IP from AKS subnet • External LB (public facing). Requires Public IP from Azure.
Firewalls	Azure Firewall (using WAF?) or existing customer solution
Network Policy	Azure CNI or Calico
Egress Traffic	Azure Firewall or NVA
High Availability	Traffic manager with multiple regions or Azs with single regions. Or both solutions can be used.
Service Mesh	Istio, Consul or Linkerd

Summary

We discussed:

- The two different Networking Models and their advantages/limitations.
- Designing my network to govern ingress and egress in my AKS cluster.
- Controlling traffic flows inside the cluster using network policies.
- Designing a private cluster and private access to the Control Plane.
- Defined the key Networking decisions that need to be made.

Microsoft

Thank you. See you next week!
Stay safe!