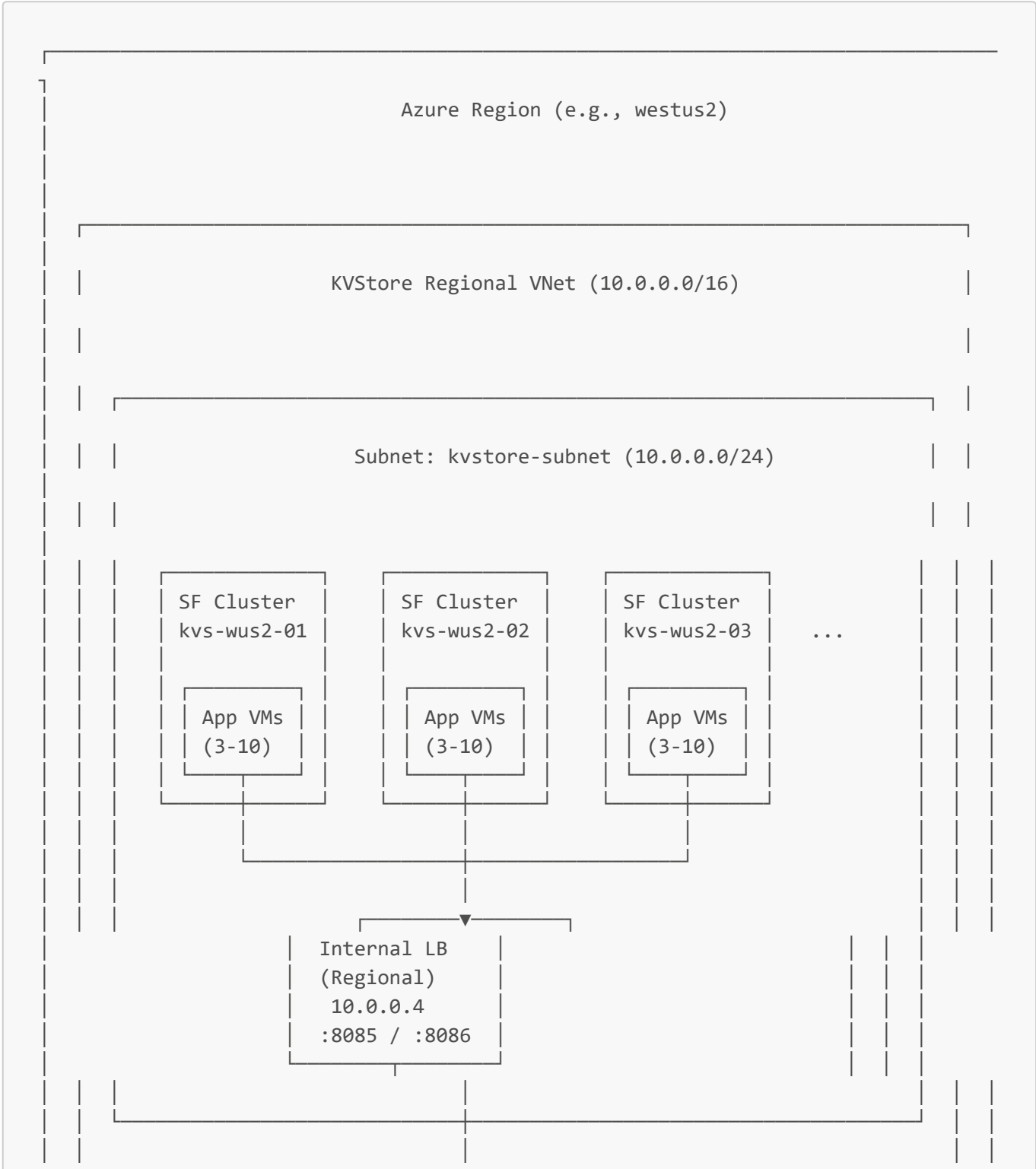
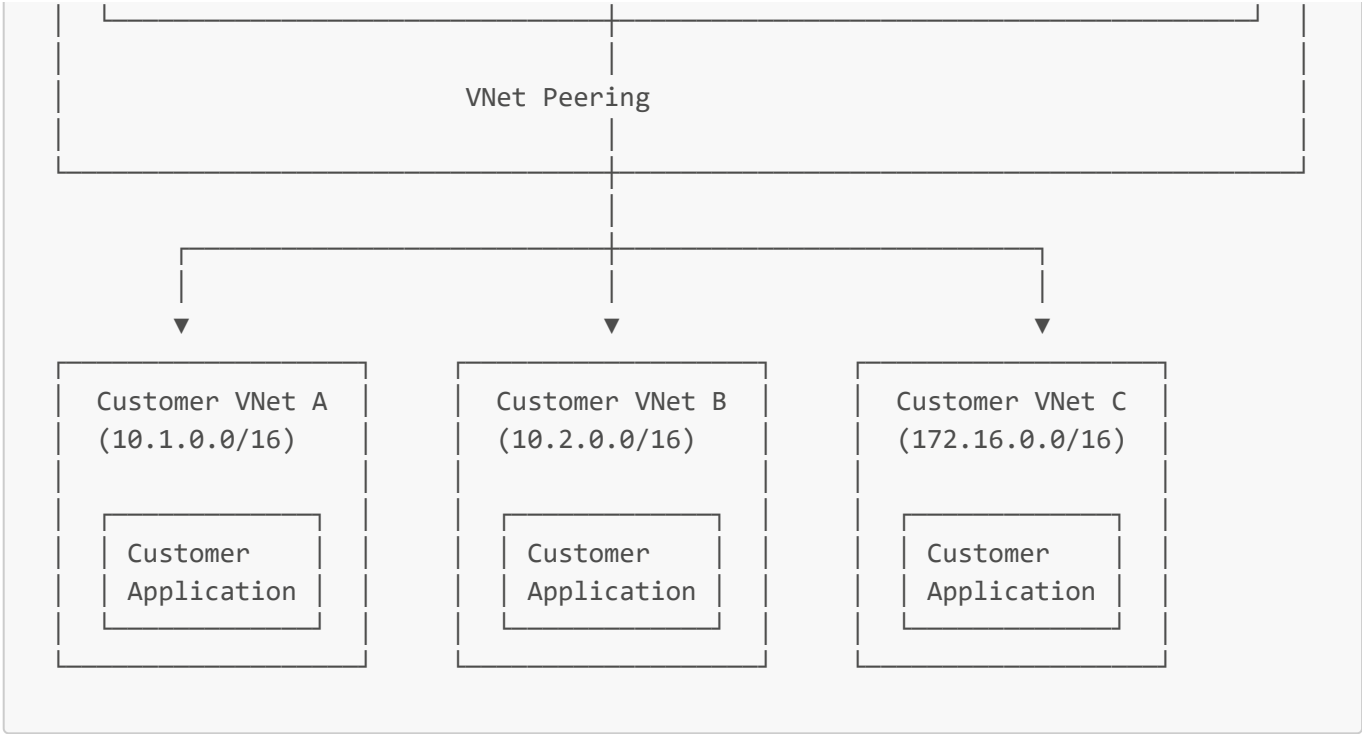
KVStore Data Plane - Regional Topology

This document explains the architecture and topology of the KVStore data plane, including how regions are set up, how clusters are deployed, and how customers connect via private networking.

Overview

The KVStore data plane is organized into **regions**, each containing shared infrastructure and one or more Service Fabric managed clusters. Customer access is provided through **private endpoints** using VNet peering and internal load balancers.





Shared Regional Assets

Each region contains shared infrastructure that is used by all clusters in that region.

1. Prompt Metadata Storage Account

The regional **prompt metadata storage account** stores configuration metadata and is accessed by all clusters.

Property	Example Value
Name	`promptdatawestus2`
Resource Group	`azureprompt_dp`
Container	`dataplane-metadata`
Purpose	Cluster configs, routing tables, customer mappings
Access	UAMI with Storage Blob Data Contributor

2. User Content Storage Accounts

User-generated content (KV data) is stored in **customer-specific storage accounts** in a dedicated resource group. This separation provides:

- Isolation of customer data
- Independent scaling per customer
- Simplified access control

Property	Example Value
Resource Group	`promptservice-regional-storage-westus2`

Property	Example Value
Storage Accounts	One per customer (e.g., `customerAstorage`, `customerBstorage`)
Purpose	Customer KV data (keys, values, embeddings)
Access	UAMI with Storage Blob Data Contributor

3. User-Assigned Managed Identity (UAMI)

A shared managed identity that Service Fabric clusters use to access Azure resources.

Property	Example Value
Name	`kvstore-svc-westus2`
Client ID	`4c1be51a-e029-46b7-a595-360d4628454a`
Resource Group	`azureprompt_dp`

UAMI Role Assignments

The UAMI is granted permissions across multiple resource groups:

Scope	Role	Purpose
`promptdatawestus2` (storage)	Storage Blob Data Contributor	Read/write dataplane metadata
`promptservice-regional-storage-westus2` (RG)	Storage Blob Data Contributor	Read/write all customer storage accounts
`kvstore-kv-westus2` (key vault)	Key Vault Secrets User	Read certificates and secrets

> **Note**: The UAMI is assigned to the entire `promptservice-regional-storage-westus2` resource group, which automatically grants access to all current and future customer storage accounts in that RG.

4. Key Vault

Stores certificates and secrets for cluster authentication.

Property	Example Value
Name	`kvstore-kv-westus2`
Resource Group	`azureprompt_dp`
Server Cert	Used by SF clusters for management
Client Cert	Used by deployment scripts

5. Virtual Network (VNet)

A dedicated VNet for the KVStore data plane in each region.

Property	Example Value
Name	`kvstore-vnet-westus2`
Resource Group	`azureprompt_dp`
Address Space	`10.0.0.0/16`
Subnet	`kvstore-subnet` (`10.0.0.0/24`)
NSG	`kvstore-nsg-westus2`

6. Internal Load Balancer (ILB)

A shared internal load balancer that provides the regional private endpoint.

Property	Example Value
Name	`kvs-wus2-01-ilb`
Resource Group	`azureprompt_dp`
Private IP	`10.0.0.4`
Port (NUMA 0)	`8085` (gRPC)
Port (NUMA 1)	`8086` (gRPC)
Backend Pool	All App node VMs

> **Note**: Each App VM runs two NUMA-pinned KVStoreServer processes:
>
> - **NUMA Node 0** listens on port `8085`
> - **NUMA Node 1** listens on port `8086`
>
> The ILB has separate rules and health probes for each port. Clients connect to either `10.0.0.4:8085` or `10.0.0.4:8086` depending on which NUMA node they want to target.

Resource Group Summary

The regional infrastructure spans multiple resource groups:

Resource Group	Purpose	Contents
`azureprompt_dp`	Dataplane infrastructure	VNet, ILB, Key Vault, UAMI, Prompt Metadata Storage, SF Clusters
`promptservice-regional-storage-westus2`	User content storage	Per-customer storage accounts
`SFC_<cluster-guid>`	SF managed resources	VMSS, managed disks (auto-created by SF)

Service Fabric Cluster Architecture

Each cluster uses the **BYOVNET + BYOLB** (Bring Your Own VNet + Bring Your Own Load Balancer) pattern.

BYOVNET (Bring Your Own VNet)

Clusters are deployed INTO the shared regional VNet using the `subnetId` property. This allows:

- All clusters to share the same network space
- Consistent network policies via shared NSG
- Single VNet peering point for customers

BYOLB (Bring Your Own Load Balancer)

The App node type uses `frontendConfigurations` to point to the shared internal load balancer:

- Traffic on port 8085 is routed to NUMA Node 0 processes
- Traffic on port 8086 is routed to NUMA Node 1 processes
- Each VM runs two KVStoreServer processes, each pinned to a NUMA node and listening on its respective port
- Default 5-tuple hash load distribution
- Separate health probes for each port ensure only healthy instances receive traffic

Dual-Port NUMA Architecture

Instead of multi-NIC binding (which has asymmetric routing issues with Azure LB), we use a **dual-port architecture**:

NUMA Node	Port	Purpose
0	8085	Process pinned to NUMA 0 (40 cores)
1	8086	Process pinned to NUMA 1 (40 cores)

This provides:

- **Full NUMA utilization**: Each NUMA node has dedicated memory and CPU cores
- **Independent health monitoring**: Each port has its own LB health probe
- **Simple client targeting**: Clients can choose which NUMA node to target by port
- **Reliable LB distribution**: No asymmetric routing issues

> **Note**: The previous multi-NIC approach (both NICs on port 8085) didn't work because Azure's health probe (168.63.129.16) only routes via the primary NIC, causing the secondary NIC's health probe to fail.

Node Types

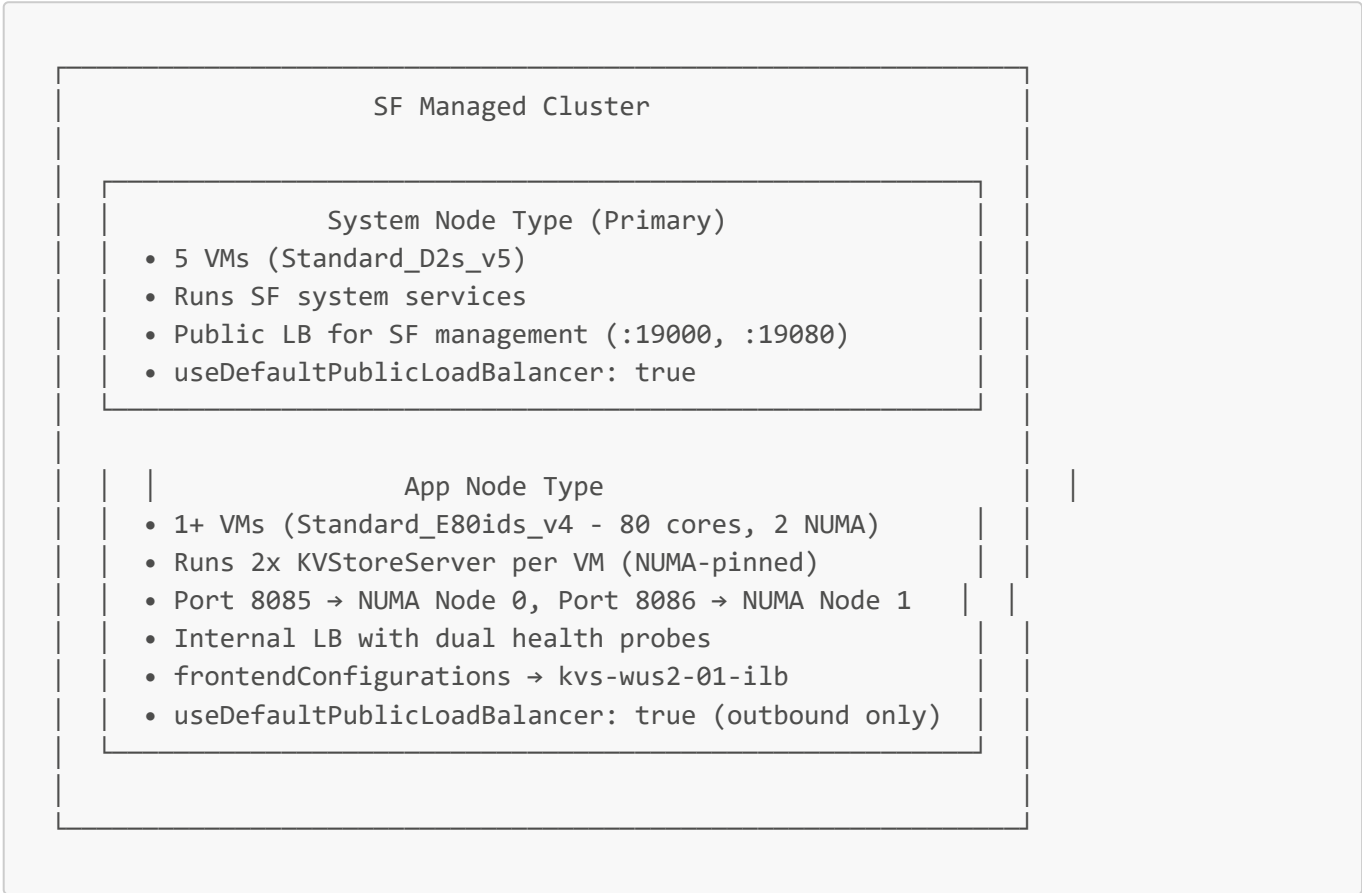
Each cluster has two node types:

Node Type	Purpose	VM Size	Count	Load Balancer
System	SF system services	Standard_D2s_v5	5	Public (SF managed)

Node Type	Purpose	VM Size	Count	Load Balancer
App	KVStoreServer application	Standard_E80ids_v4	1+	Internal (BYOLB)

> **Note**: The App node type uses `Standard_E80ids_v4` isolated VMs with 80 vCPUs, 80 Gbps network, and 2 NUMA nodes. Each VM runs two KVStoreServer processes:
>
> - NUMA Node 0 process listens on port 8085
> - NUMA Node 1 process listens on port 8086

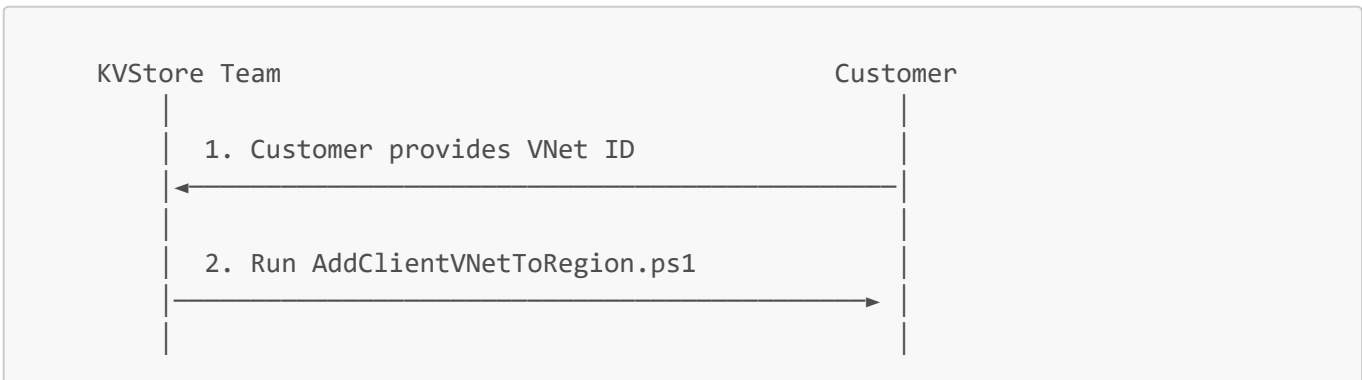
Cluster Configuration



Customer Connectivity

Customers connect to the KVStore service via **VNet peering** to access the private internal load balancer endpoint.

VNet Peering Process



```
|  3. Provide PromptServiceVNetId + Endpoint    |
|──────────────────────────────────────────────►|
|                                               |
|  4. Customer runs PeerToPromptService.ps1     |
|◄──────────────────────────────────────────────|
|                                               |
|         Bidirectional Peering Complete        |
|◄─────────────────────────────────────────────►|
|                                               |
|  5. Customer accesses 10.0.0.4:8085 or :8086  |
|◄──────────────────────────────────────────────|
```

Peering Requirements

Requirement	Description
No Overlapping Address Space	Customer VNet must not use the same CIDR as the KVStore VNet or any other peered VNet
Network Contributor	Both parties need Network Contributor on their respective VNets
NSG Rules	Customer NSG must allow outbound TCP to ports 8085 and 8086

Example Endpoint Access

After peering is complete, customers can access the service:

```
# From customer's VM - NUMA Node 0
./KVClient 10.0.0.4:8085

# From customer's VM - NUMA Node 1
./KVClient 10.0.0.4:8086

# gRPC endpoints
grpc://10.0.0.4:8085  # NUMA 0
grpc://10.0.0.4:8086  # NUMA 1
```

> **Note**: Two ports are exposed to clients (8085 for NUMA Node 0, 8086 for NUMA Node 1). The load balancer distributes traffic for each port to the corresponding NUMA-pinned process on each App VM.

Configuration Files

Region Configuration (`config/<region>/region.config.json`)

Contains shared regional settings:

```
{
  "region": "westus2",
  "azure": {
```

```
    "resourceGroupName": "azureprompt_dp",
    "subscriptionId": "...",
    "location": "westus2"
  },
  "network": {
    "vnetName": "kvstore-vnet-westus2",
    "subnetName": "kvstore-subnet",
    "vnetAddressPrefix": "10.0.0.0/16",
    "subnetAddressPrefix": "10.0.0.0/24"
  },
  "identity": {
    "uamiName": "kvstore-svc-westus2",
    "uamiClientId": "..."
  },
  "dataplane": {
    "configStorageAccount": "promptdatawestus2",
    "grpcPortNuma0": 8085,
    "grpcPortNuma1": 8086,
    "grpcPorts": [8085, 8086]
  }
}
```

Cluster Configuration (`config/<region>/<cluster>.config.json`)

Contains cluster-specific settings:

```
{
  "cluster": {
    "name": "kvs-wus2-01",
    "endpoint": "kvs-wus2-01.westus2.cloudapp.azure.com:19000",
    "serverCertThumbprint": "..."
  },
  "byolb": {
    "internalLoadBalancerName": "kvs-wus2-01-ilb",
    "privateIp": "10.0.0.4"
  },
  "nodeTypes": {
    "system": { "name": "System", "instanceCount": 5 },
    "app": { "name": "App", "instanceCount": 3 }
  }
}
```

Scripts Reference

Region Setup

Script	Purpose
`scripts/init/InitRegion.ps1`	Initialize a new region with VNet, UAMI, Key Vault

Script	Purpose
scripts/dataplane/AddCluster.ps1	Add a new SF cluster to a region

Cluster Management

Script	Purpose
scripts/dataplane/AddCluster.ps1	Create cluster with BYOVNET + BYOLB
scripts/dataplane/DeployApp.ps1	Deploy KVStoreServer application
scripts/dataplane/RemoveCluster.ps1	Remove a cluster from a region

Customer VNet Peering

Script	Purpose	Run By
scripts/dataplane/AddClientVNetToRegion.ps1	Create peering from KVStore → Customer VNet	KVStore Team
scripts/dataplane/PeerToPromptService.ps1	Create peering from Customer → KVStore VNet	Customer

Testing & Benchmarking

Script	Purpose
scripts/run/run_azure_linux_cluster.ps1	Test from Linux VM via internal LB
scripts/run/benchmark_run.ps1	One-click benchmark sweep with chart generation
scripts/deploy/deploy_client.ps1	Deploy KVPlayground client + benchmark scripts to Linux VM

Benchmark Analysis

Script	Purpose
scripts/analysis/benchmark_sweep.sh	Multi-process benchmark sweep (runs on Linux VM)
scripts/analysis/generate_charts.py	Generate interactive HTML charts from results

Adding a New Cluster

To add a new cluster to an existing region:

```
# 1. Create the cluster with BYOVNET + BYOLB
.\scripts\dataplane\AddCluster.ps1 `
    -ClusterName "kvs-wus2-02" `
    -RegionName "westus2" `
    -AdminPassword (ConvertTo-SecureString "YourPassword!" -AsPlainText -Force) `
```

```
    -UseInternalLoadBalancer

# 2. Deploy the application
.\scripts\dataplane\DeployApp.ps1 `
    -ClusterName "kvs-wus2-02" `
    -RegionName "westus2"
```

The new cluster's App nodes will automatically be added to the shared internal load balancer's backend pool.

Adding a New Customer VNet

To peer a customer VNet:

```
# 1. KVStore team runs (after customer provides their VNet ID)
.\scripts\dataplane\AddClientVNetToRegion.ps1 `
    -ClientVNetId "/subscriptions/.../virtualNetworks/customer-vnet" `
    -RegionName "westus2"

# 2. Customer runs (with info provided by KVStore team)
.\PeerToPromptService.ps1 `
    -PromptServiceVNetId "/subscriptions/.../virtualNetworks/kvstore-vnet-westus2"
`
    -ClientVNetName "customer-vnet" `
    -ClientVNetResourceGroup "customer-rg"
```

Scaling Considerations

Horizontal Scaling

- **Add more clusters**: Each cluster adds more App nodes to the regional backend pool
- **Scale out App nodes**: Increase `instanceCount` for the App node type
- **Scale out System nodes**: Increase count for higher SF management capacity

Regional Expansion

To add a new region:

1. Run `InitRegion.ps1` to create shared assets
2. Run `AddCluster.ps1` to create the first cluster
3. Run `DeployApp.ps1` to deploy the application
4. Set up VNet peering for customers in that region

Security Model

Layer	Protection
Network	Private VNet, NSG rules, no public ingress
Transport	gRPC over TLS

Layer	Protection
Identity	UAMI for Azure resource access
Data	Azure Storage encryption at rest

Client Testing Infrastructure

For benchmarking and testing, we use a **Linux VM** in a peered VNet that accesses the KVStore service via the internal load balancer.

Linux Client VM

Property	Value
Public IP	`20.115.133.84`
Private IP	`10.1.1.4`
VNet	Peered to `kvstore-vnet-westus2`
SSH Key	`kvClient.pem`
Working Dir	`~/kvgrpc/`

Client Components

```
~/kvgrpc/
├── KVPlayground          # Benchmark executable
├── conversation_tokens.json  # Test data (token blocks)
├── benchmark_sweep.sh    # Multi-process benchmark script
└── generate_charts.py    # Chart generator
```

Running Benchmarks

Option 1: One-click from Windows (recommended)

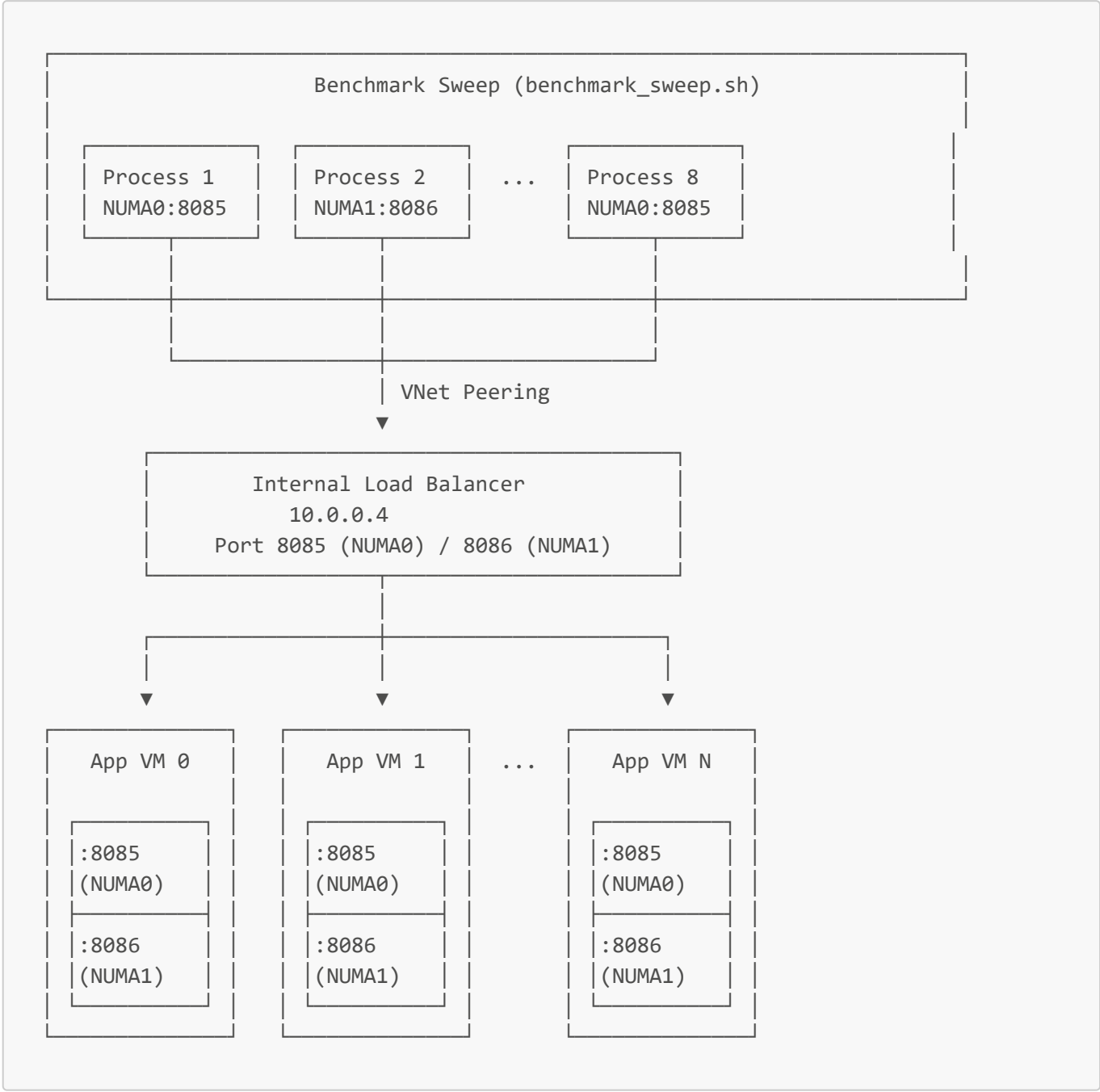
```
# Run benchmark sweep and generate charts
.\scripts\run\benchmark_run.ps1

# Just regenerate charts from existing data
.\scripts\run\benchmark_run.ps1 -SkipBenchmark -OpenChart
```

Option 2: Manual SSH

```
# SSH to Linux VM
ssh -i C:\Users\kraman\Downloads\kvClient.pem azureuser@20.115.133.84
```

```
# Run benchmark
cd ~/kvgrpc
./benchmark_sweep.sh
```

Benchmark Architecture



Deploying Client Updates

```
# Deploy KVPlayground binary + benchmark scripts to Linux VM
.\scripts\deploy\deploy_client.ps1
```

This copies:

- `KVPlayground` binary (built from `build/KVPlayground/`)

- `benchmark_sweep.sh` (from `scripts/analysis/`)
- `generate_charts.py` (from `scripts/analysis/`)

Performance Results

See **PERFORMANCE_SUMMARY.md** and **benchmark_charts.html** for detailed benchmark results.

Summary:

- **Peak throughput:** ~94K tokens/sec (reads + writes)
- **Peak TPS:** ~82 iterations/sec
- **Sweet spot:** C=32 (8 processes × 4 concurrency each)
- **Lookup latency:** p50=4.7ms (metadata only)
- **Read latency:** p50=26ms at C=32 (Azure Storage bound)

Troubleshooting

Common Issues

1. **VNet Peering Fails - Address Space Overlap**
    - Check for stale/disconnected peerings
    - Ensure customer VNet doesn't overlap with `10.0.0.0/16`
2. **Cannot Connect to Internal LB**
    - Verify VNet peering is in "Connected" state on both sides
    - Check NSG allows outbound TCP on ports 8085 and 8086
    - Verify internal LB health probe is passing
3. **Application Not Responding**
    - Check SF cluster health via Explorer
    - Verify KVStoreServer is deployed and healthy
    - Check App node count > 0

Useful Commands

```
# Check cluster status
az sf managed-cluster show --cluster-name kvs-wus2-01 --resource-group
azureprompt_dp

# Check ILB backend pool
az network lb address-pool show --lb-name kvs-wus2-01-ilb --resource-group
azureprompt_dp --name appBackendPool

# Check VNet peering status
az network vnet peering list --resource-group azureprompt_dp --vnet-name kvstore-
vnet-westus2
```

```
# Test connectivity from peered VNet
./KVClient 10.0.0.4:8085  # or :8086 for NUMA node 1
```