CloudGuard

Terraformed Provisioned Azure Lab

Advanced Threat Prevention

Automated & Cloud-Native

Unified Management

Installing Terraform

Terraform is a Binary / Executable Application - Just download, update the path variable so the system can find the executable and run it.

Binary Download Available Here:

https://developer.hashicorp.com/terraform/install

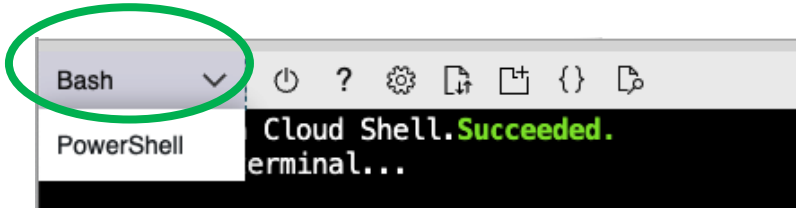
Step by Step Instructions (for Windows, Mac, and Linux) available here:

https://spacelift.io/blog/how-to-install-terraform

Running from Azure CLI (No need to Install Anything)

- ✓ Open Azure Web Portal and Log in
- ✓ Click on CLI icon

- ✓ Select “Bash”



Type terraform version

Download and Upgrade binary if needed (instructions below):

Type curl -O https://releases.hashicorp.com/terraform/1.6.5/terraform_1.6.5_linux_amd64.zip

Type unzip terraform_1.6.5_linux_amd64.zip

Type mkdir bin

Type mv terraform bin/

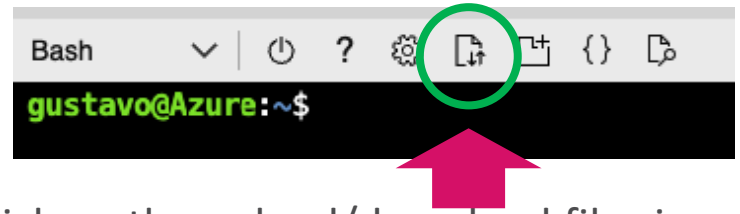
Close and re-open CLI

Type terraform version

CHECK POINT

Upload and Uncompress the AzureLab.tgz File

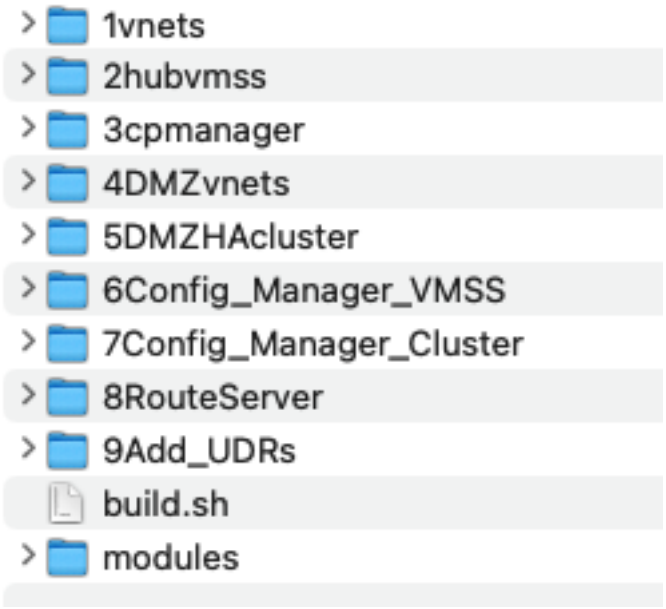
- ✓ Upload the File to Azure CLI (if not using your own environment)



- ✓ Click on the upload/download files icon

Type tar -xvf AzureLab.tgz to uncompress the playbooks

If using Windows, Mac or Linux, simply copy the AzureLab.tgz file to the desired directory and uncompress



CHECK POINT

Create or Select a Service Principal (Contributor Access)

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/guides/service_principal_client_secret

Store it as an Environmental Variable (Linux, Mac or Windows Terminal)

```
export TF_VAR_ARM_CLIENT_ID=”abcdefghijklmnop1234567890"
export TF_VAR_ARM_CLIENT_SECRET="abcdefghijklmnop1234567890"
export TF_VAR_ARM_TENANT_ID="abcdefghijklmnop1234567890"
export TF_VAR_ARM_SUBSCRIPTION_ID="abcdefghijklmnop1234567890"
```

Did the Variables get stored correctly?

```
printenv | grep ARM
```

Remove the Variables from the Environment (or close session)

```
unset TF_VAR_ARM_CLIENT_ID="abcdefghijklmnop1234567890"
unset TF_VAR_ARM_CLIENT_SECRET="abcdefghijklmnop1234567890"
unset TF_VAR_ARM_TENANT_ID="abcdefghijklmnop1234567890"
unset TF_VAR_ARM_SUBSCRIPTION_ID="abcdefghijklmnop1234567890”
```

CHECK POINT™

Terraform Commands

terraform init

Initializes the directory, loads the providers. Can be run multiple times. Recommended if changes have been made to any of the terraform files in the directory.

terraform plan

Creates an execution plan, which lets you preview the changes that Terraform plans to make to your infrastructure. Checks the “state” and identifies differences. Displays a list of actions to be performed.

terraform apply (-auto-approve option removes interactive request)

Executes the actions detailed in the terraform plan

terraform destroy (-auto-approve option removes interactive request)

Destroys all objects / elements created with the apply command

CHECK POINT™

1 VNETS

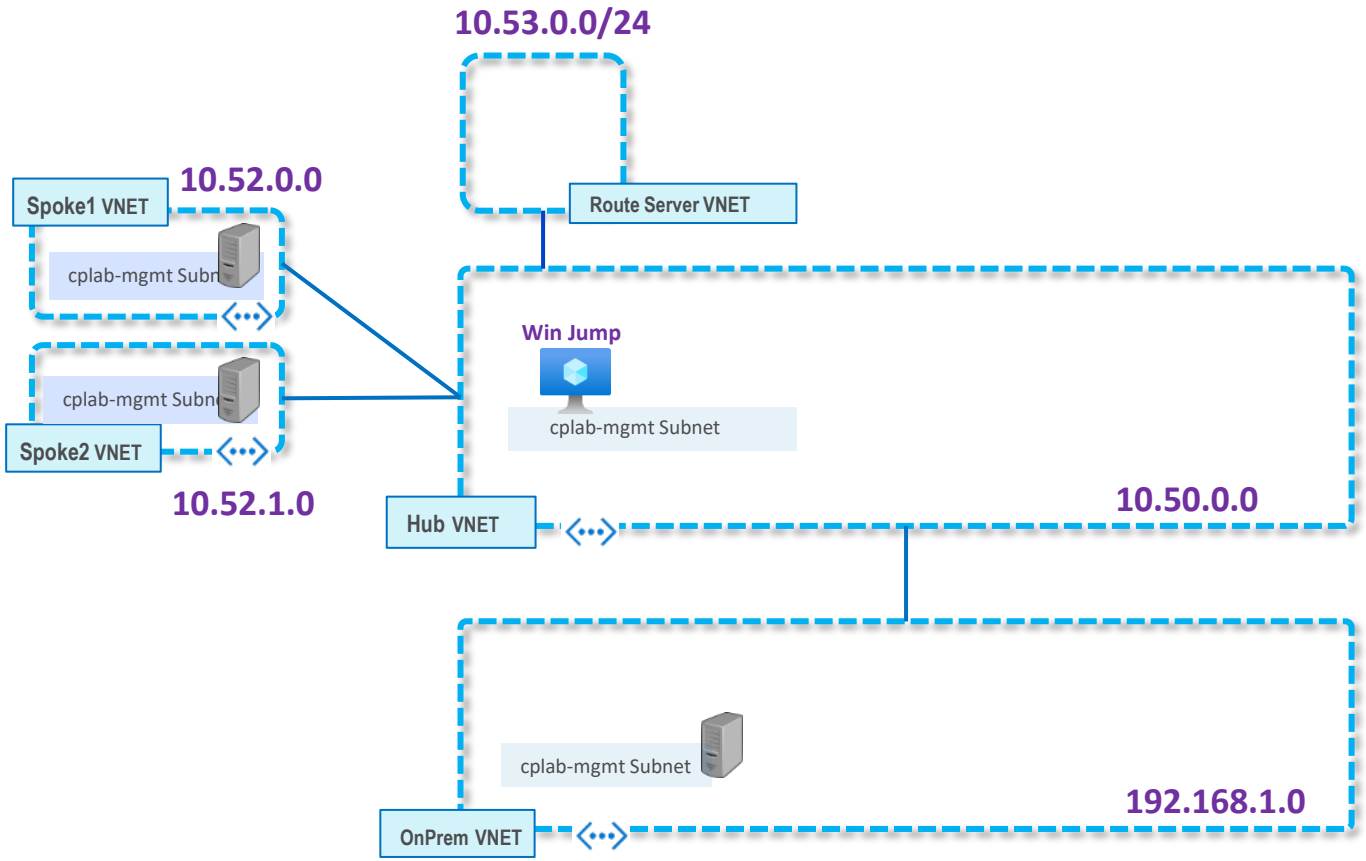
The vnets directory builds a security landing zone, a vnet that mimics an onPrem datacenter, plus two spokes containing either app or test VMs.

The playbooks will deploy three Linux hosts (1 vCPU each).

The landing zone will also include a Windows Jump server that can be optionally used to run the Check Point Smart Console.

There is a third spoke vnet that can be used for testing, but it is meant to optionally hold the Azure Router Service.

Lab Address Space 10.50/10.51/10.52/10.53/10.54

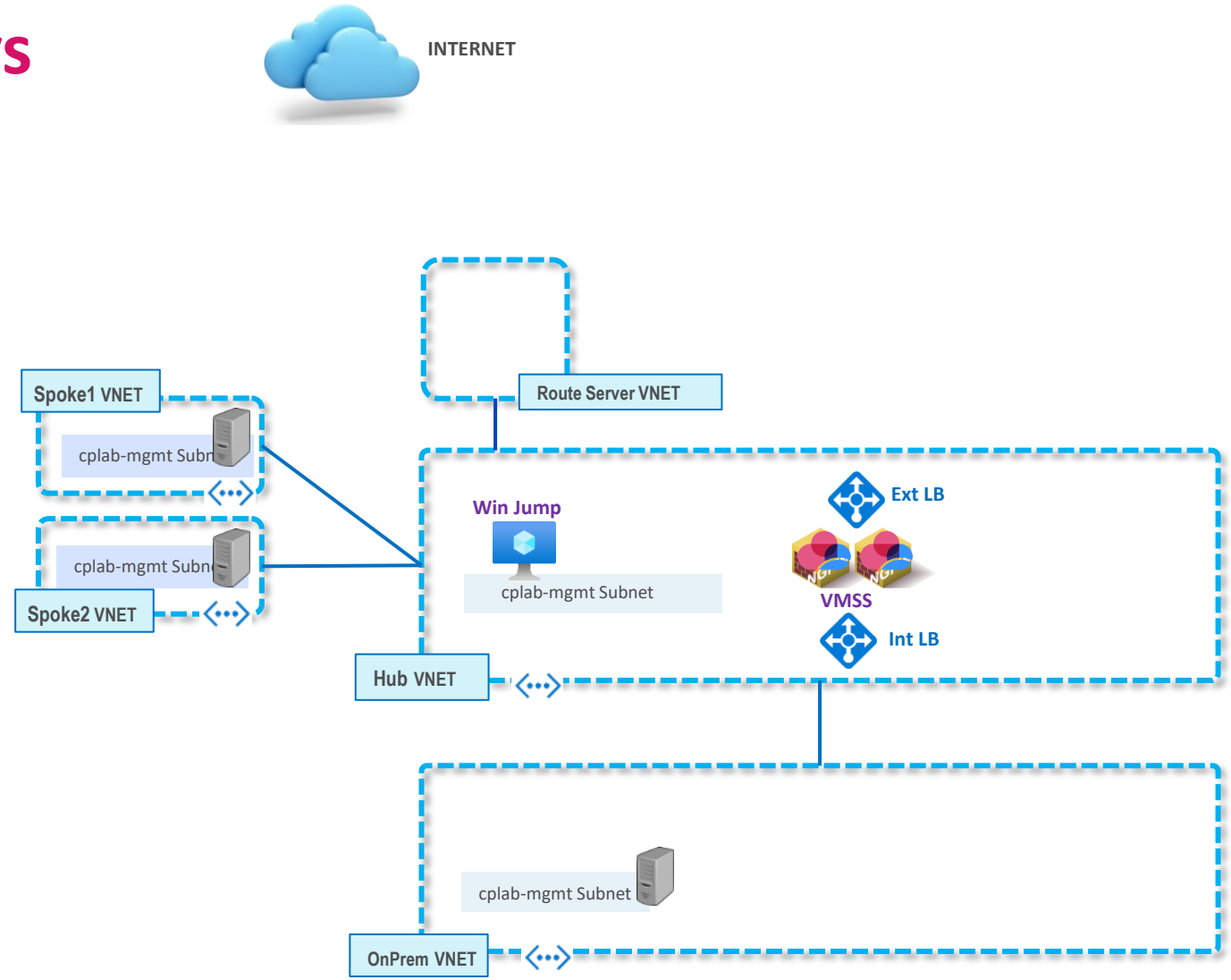


CHECK POINT

2 VMSS CP Gateways

This directory builds a two member CloudGuard Scale Set. The scale set can be used for inspection of Ingress, Egress internal VNET to VNET traffic as well as traffic between OnPrem and Azure.

The scale set can also be used for testing dynamic auto scaling (scale up and scale down events) and understanding how the management server automatically provisions the security gateways.

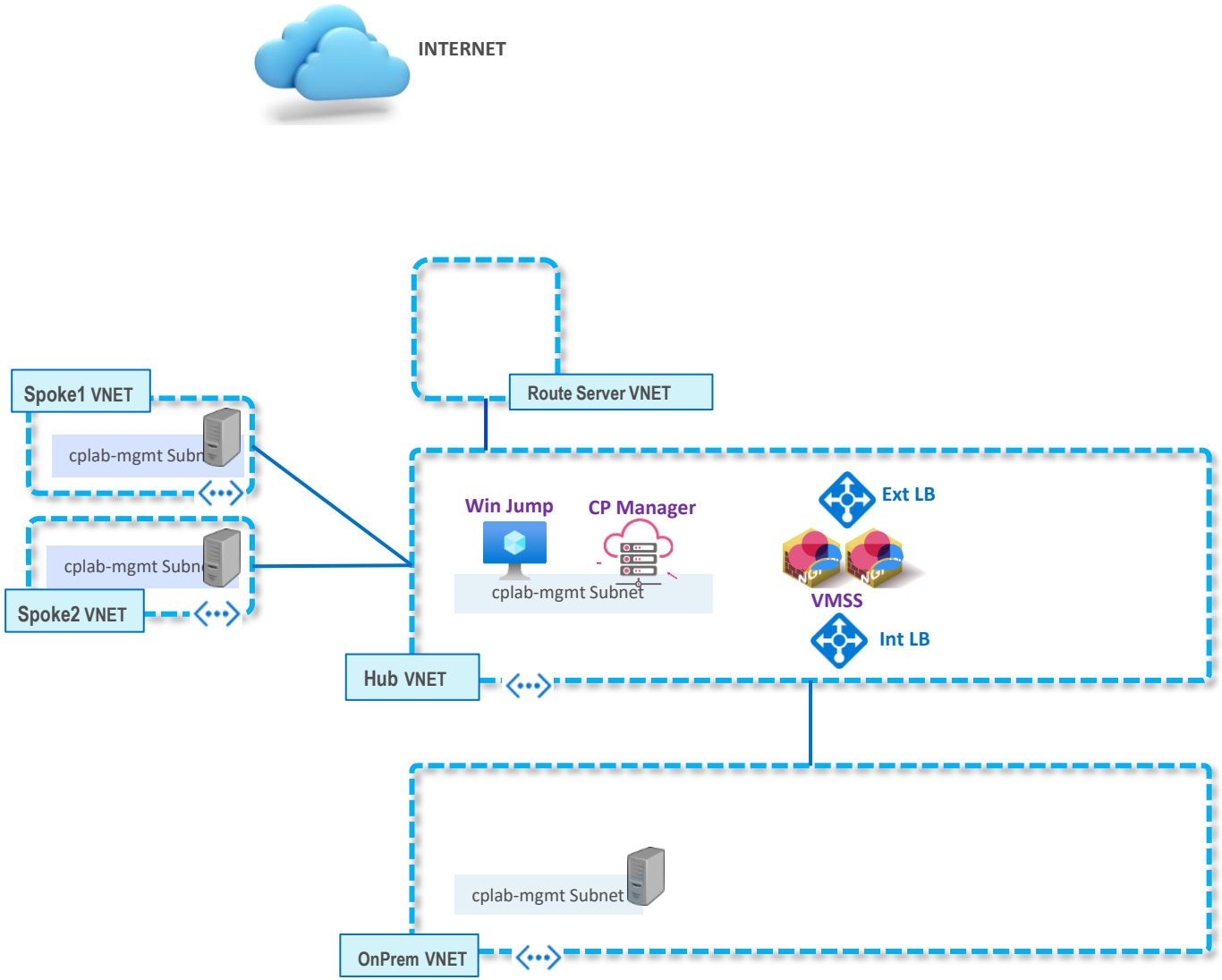


CHECK POINT

3 CP Manager

This directory optionally deploys a Check Point Management server on the same subnet as the Windows Jump Station.

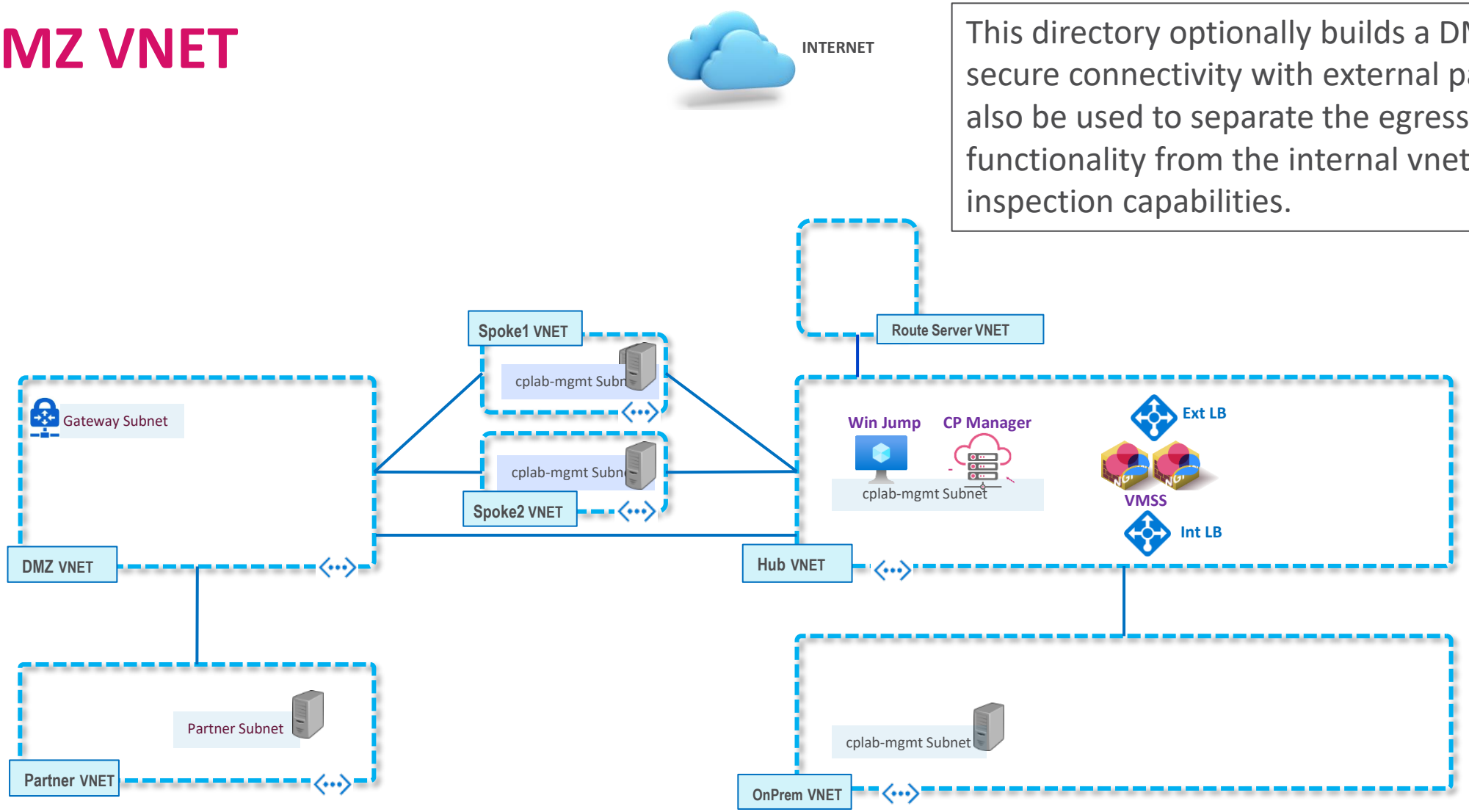
While the lab gateways can be managed from an outside (pre-existing) management server, this directory provides the option of using a dedicated management server for testing cloud functionality such as creating tag-based rules or playing with the API (or the auto provisioning functionality).



CHECK POINT

4 DMZ VNET

This directory optionally builds a DMZ vnet to allow secure connectivity with external partners and can also be used to separate the egress and ingress functionality from the internal vnet to vnet inspection capabilities.

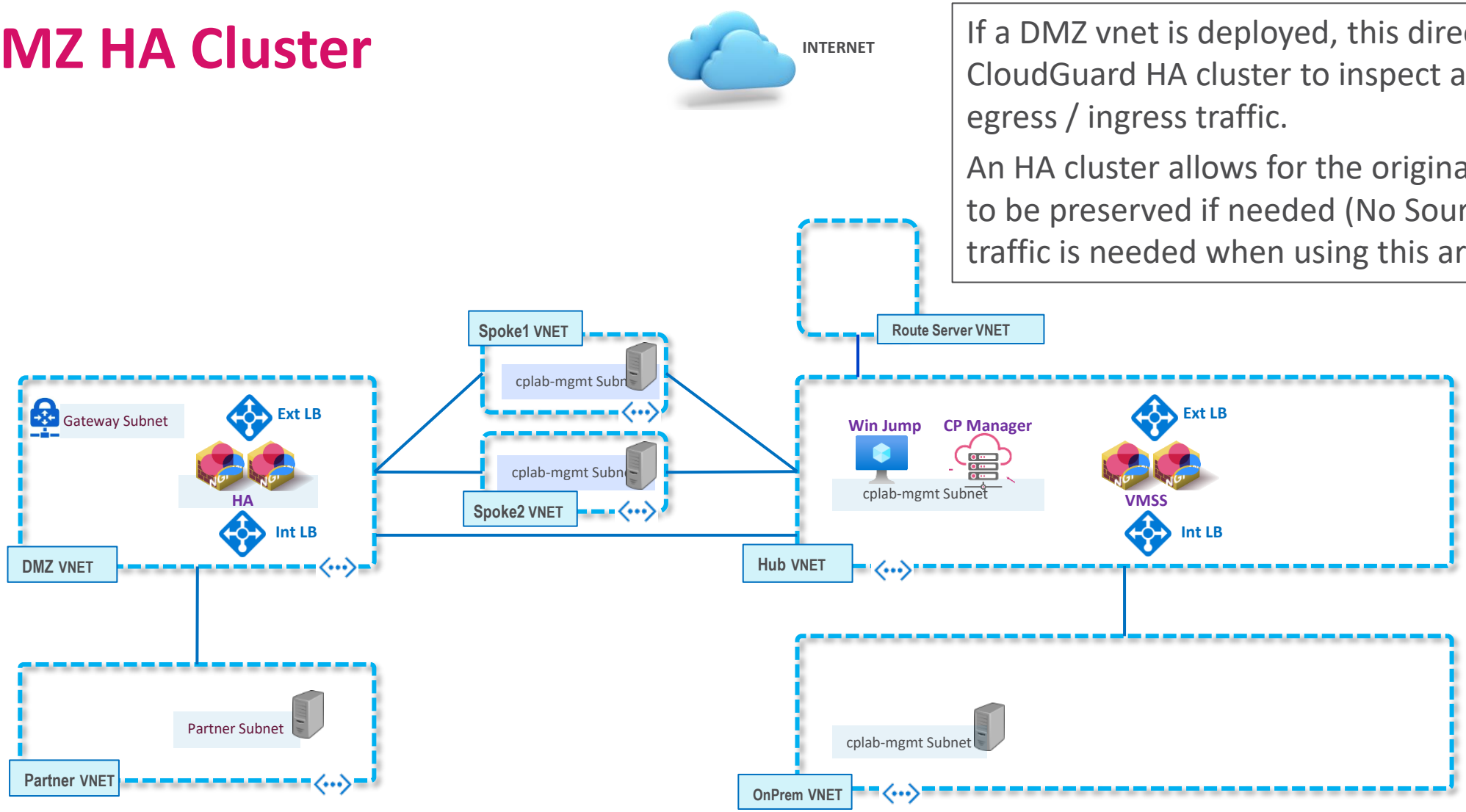


CHECK POINT

5 DMZ HA Cluster

If a DMZ vnet is deployed, this directory builds a CloudGuard HA cluster to inspect any partner / egress / ingress traffic.

An HA cluster allows for the original inbound client IP to be preserved if needed (No Source NAT of Ingress traffic is needed when using this architecture).

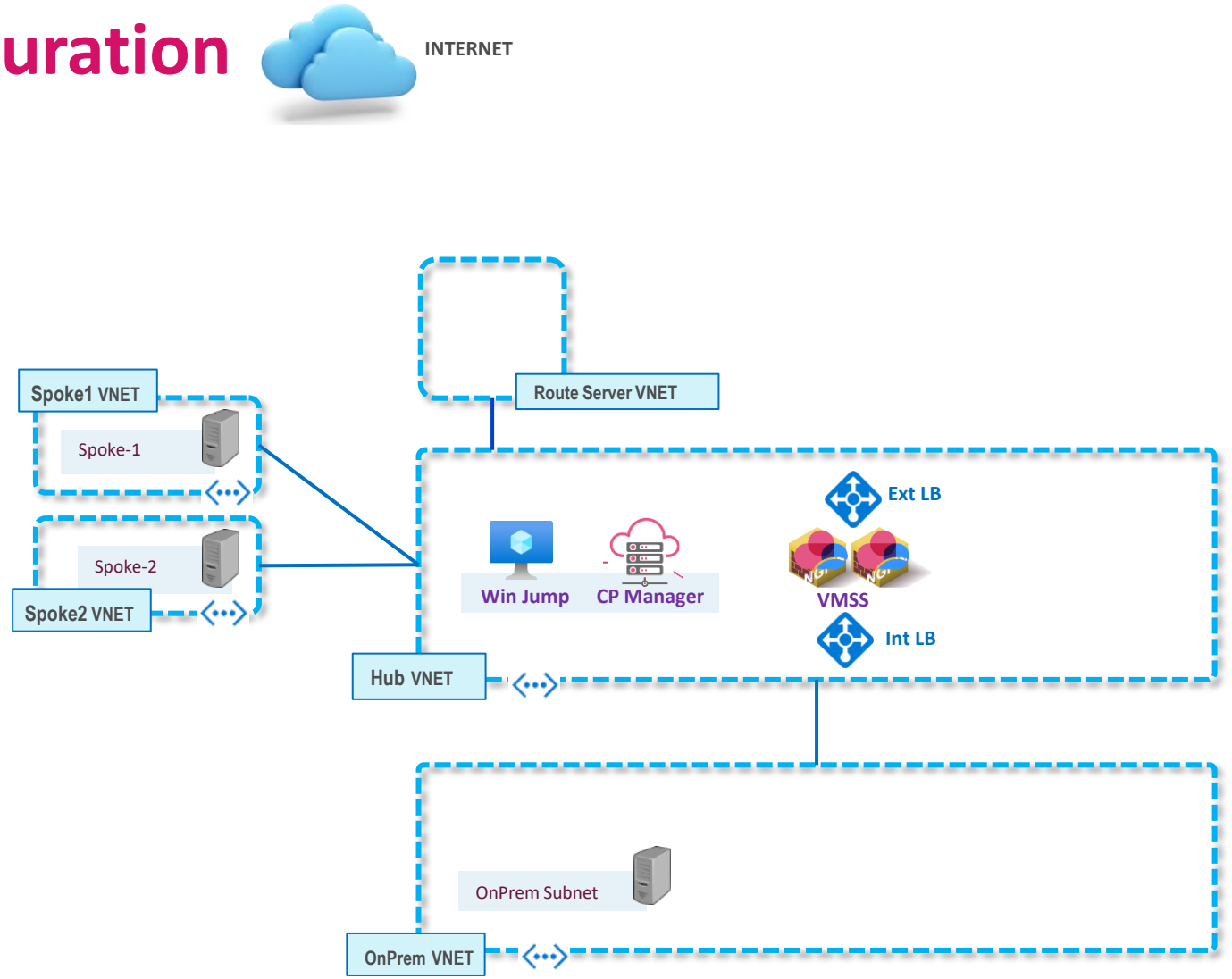


6 CP Manager Configuration

This directory optionally configures auto provisioning on the lab Management server.

It mainly runs a configuration script that builds an auto provisioning template for the scale set, creates a datacenter connection between the management server and Azure (to allow tag-based security policies), and adds a few basic rules to the security policy (to allow vnet to vnet connectivity as well as NAT hide for egress to the Internet).

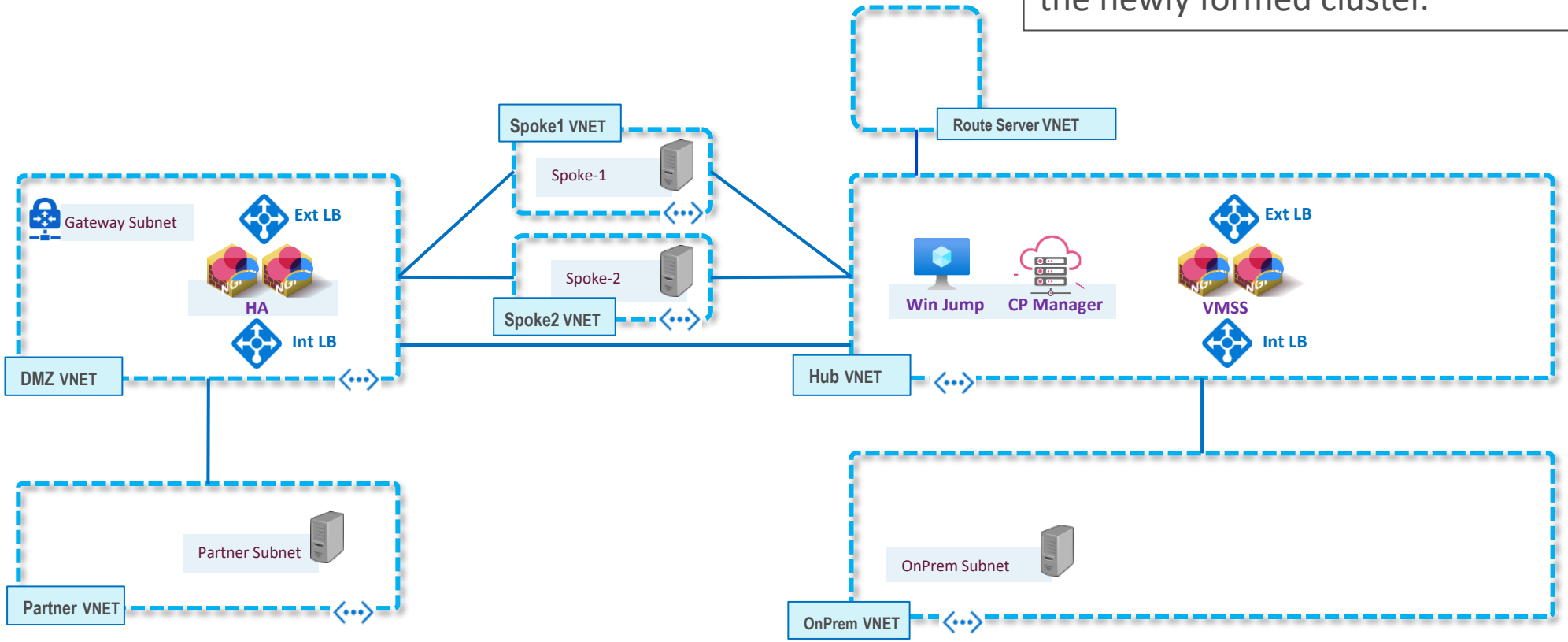
While all this can be done manually, the script is a potential time saver.



CHECK POINT

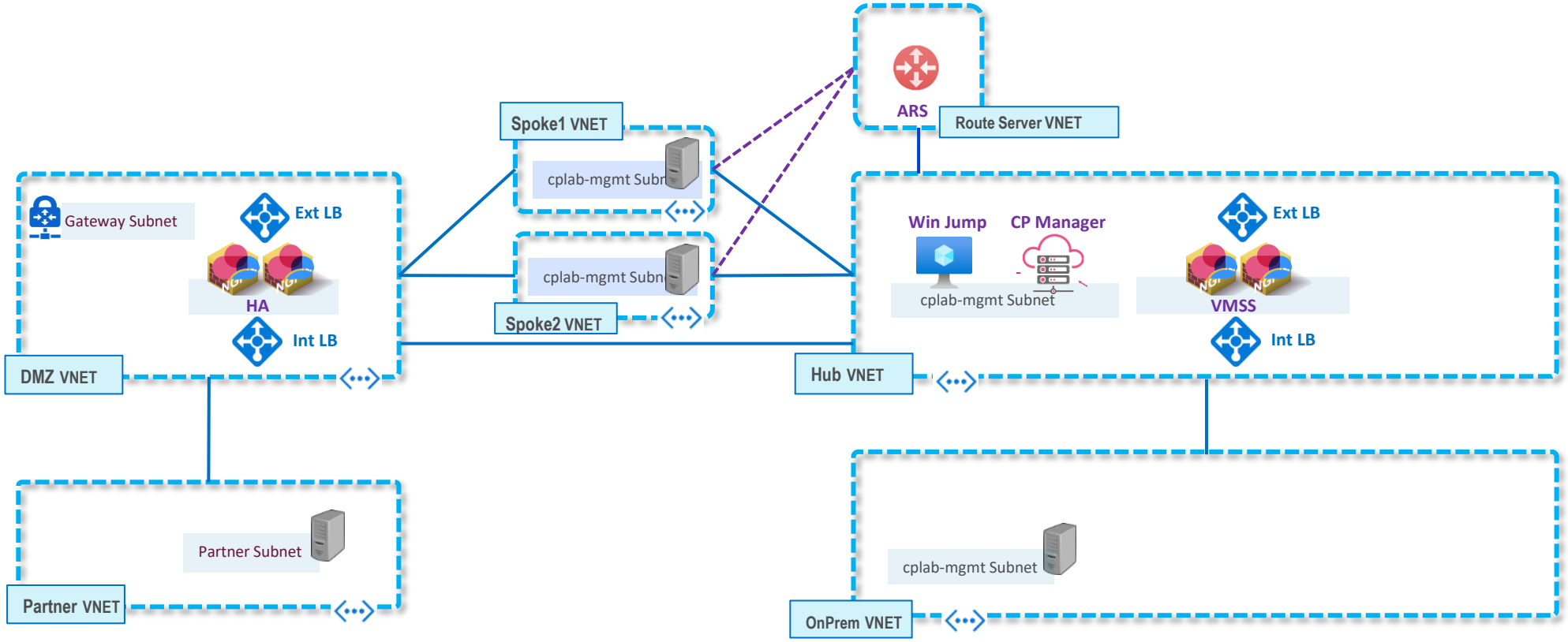
7 CP Manager Cluster Configuration

If an HA Cluster is deployed, this directory optionally executes a terraform script that builds the HA cluster object in Smart Console, runs SIC and pushes the rule base created by the configuration script on step 5 to the newly formed cluster.



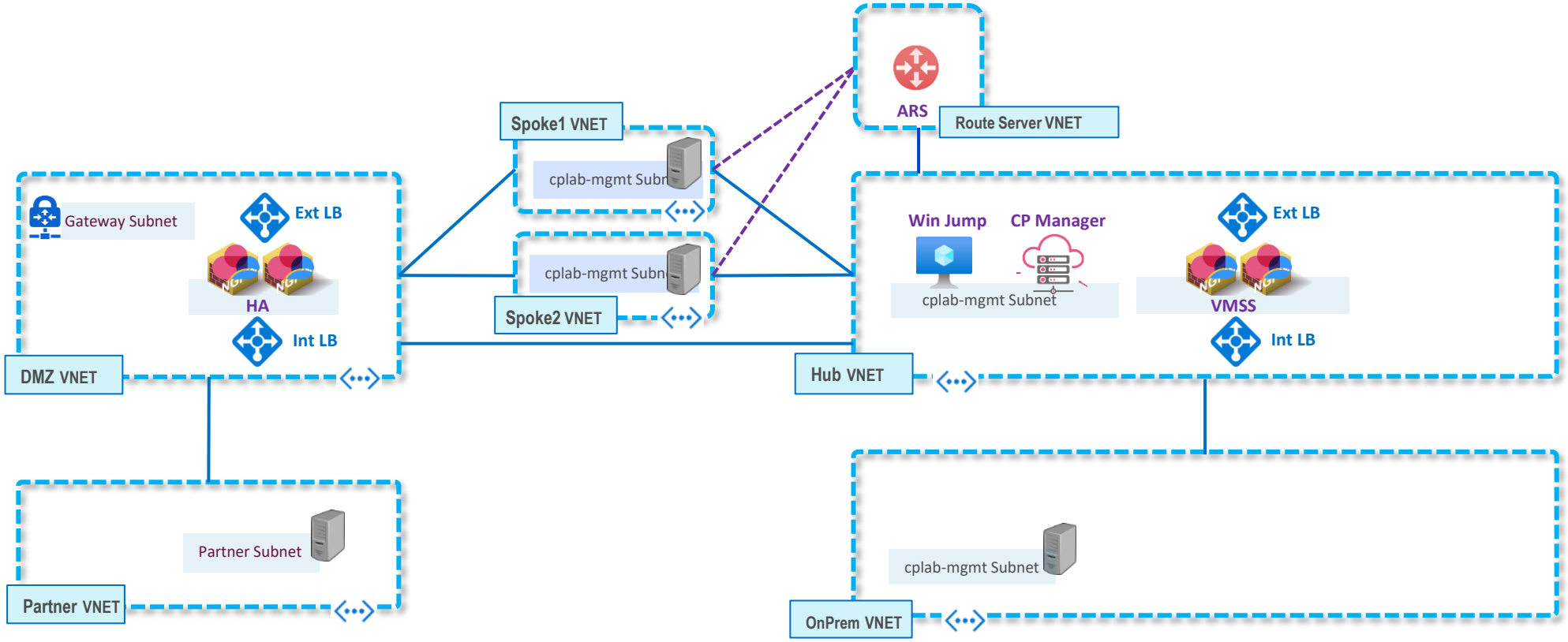
8 Add Route Server

If one of the objectives of the lab is to test removing the needs for UDRs. This directory deploys the Azure BGP service and configures a BGP connection to the Cloud Guard VMSS.



9 Add UDR Routes

If the Azure Route Server will not be used to advertise a default route to the spokes, this section builds the necessary static route tables.

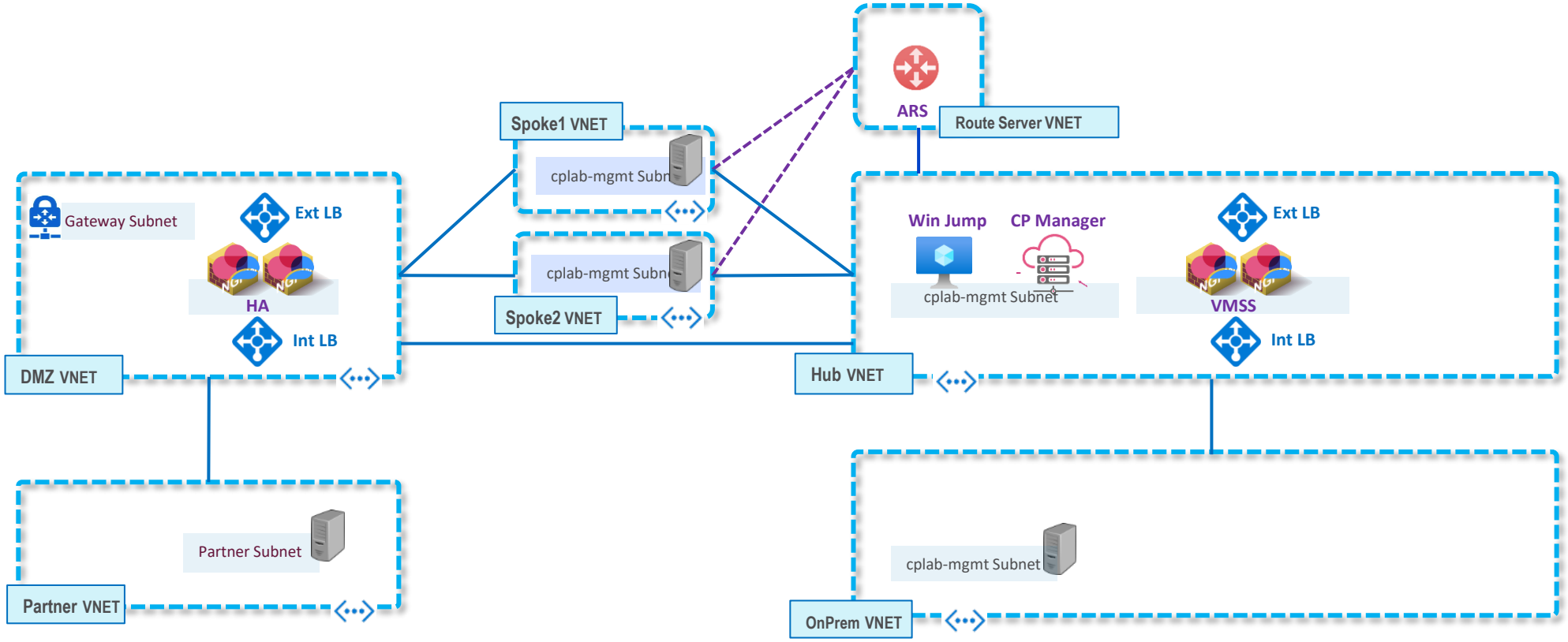


CHECK POINT

Runbook Errors/Warnings

These runbooks have been executed many times without error, but sometimes errors and warnings still occur. Warnings are usually notifications of future changes and can be safely ignored.

The most common deployment error is an “Allocation Error”. Sometimes Azure does not have enough resources available to build one component (or more) of the architecture. This error usually goes away after a while.



CHECK POINT

Connecting to the Management Server

You can download the SmartConsole Management client by connecting via https to either one of the cluster instances.

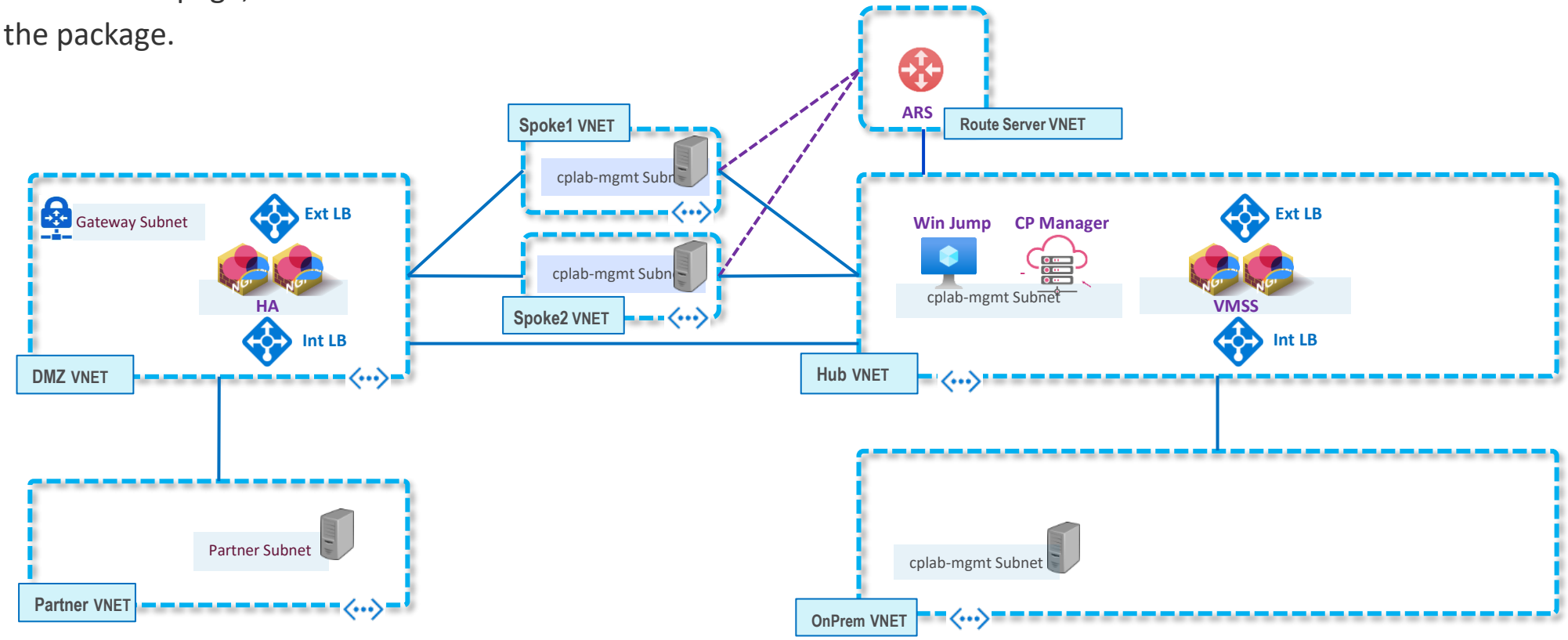
Log in using “admin” and the password you entered as the last of the environment variables.

In the navigation tree (menu on the left), click **Maintenance** > **Download SmartConsole**.

Click the **Download** button.

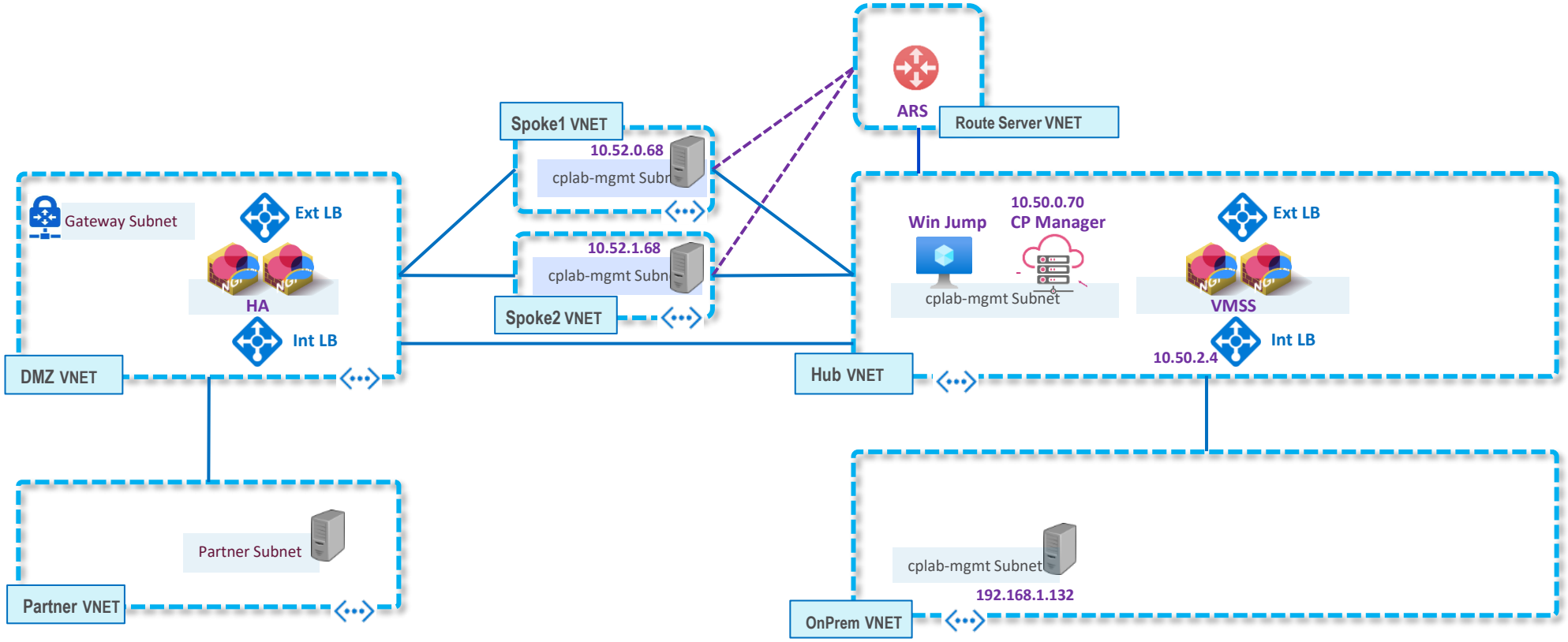
On the download page, click the **Download** button.

Save the package.



Testing VNET to VNET

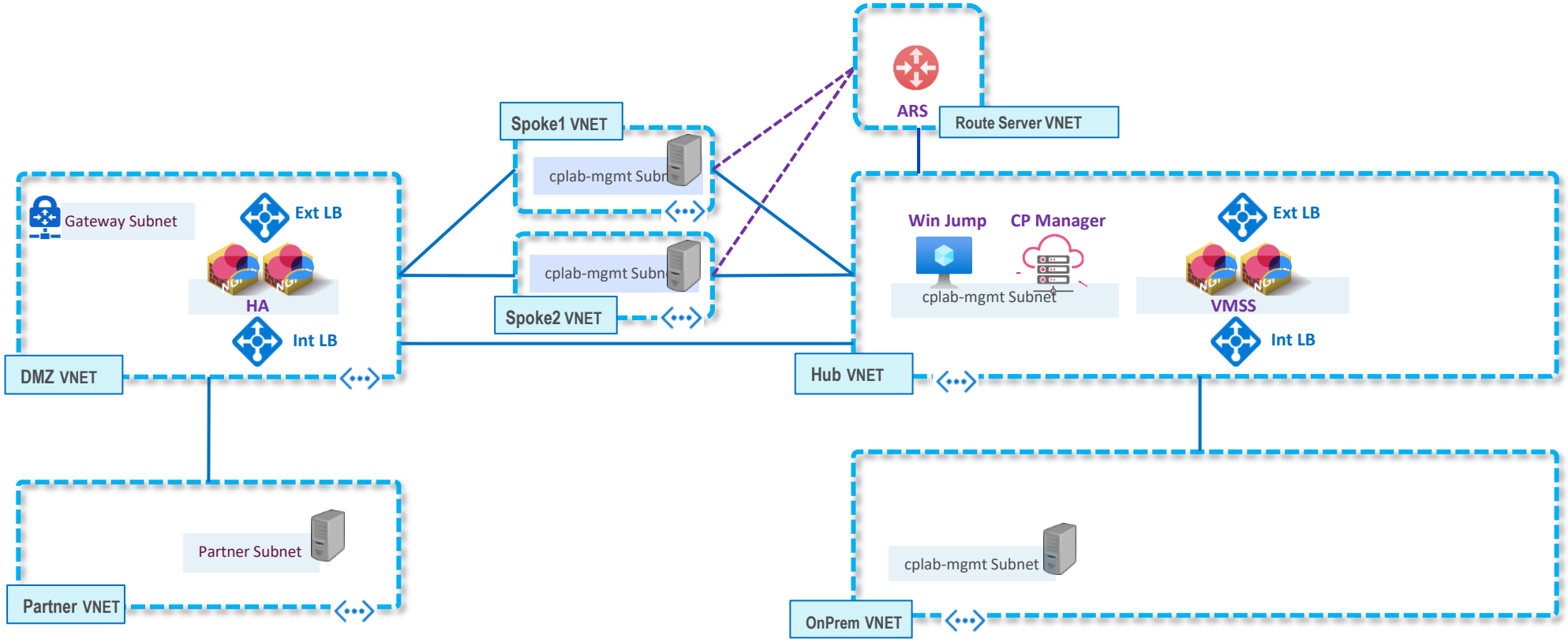
You should be able to connect to either of the spoke Linux hosts by using putty, MobaXterm, or similar application and connect via SSH. Once connected, try pinging or SSH from one spoke host to the other. After a successful connection, the traffic should appear on the Firewall logs.



CHECK POINT

Testing Egress

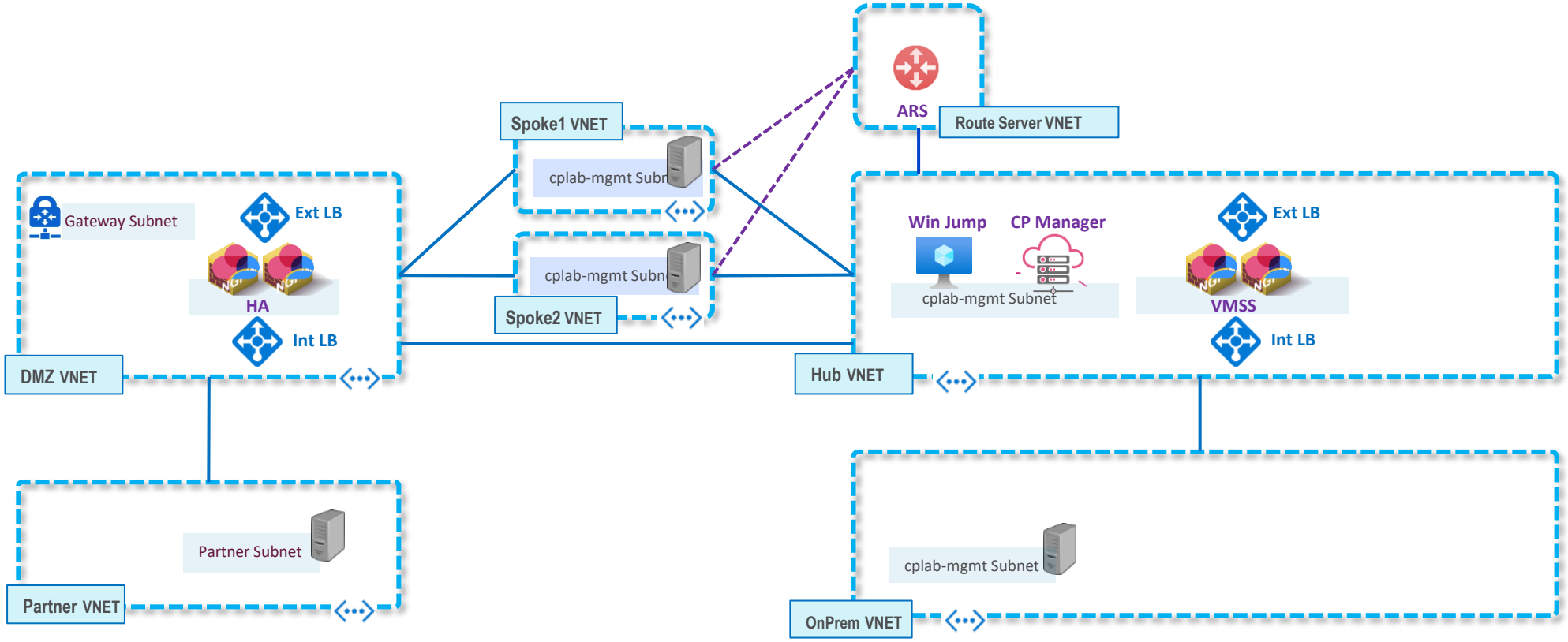
From either one of the spoke hosts, you should be able to ping or curl out to the Internet. The connection should appear on the management logs.



Testing Ingress

We did not set up a pre-configured “ingress” rule base or mapping. If you are familiar with the process, you can set it up. Or you can consult the “CloudGuard Network for Azure Virtual Machine Scale Sets (VMSS) Deployment Guide” available at this link:

https://sc1.checkpoint.com/documents/IaaS/WebAdminGuides/EN/CP_VMSS_for_Azure/Content/Topics-Azure-VMSS/Overview.htm

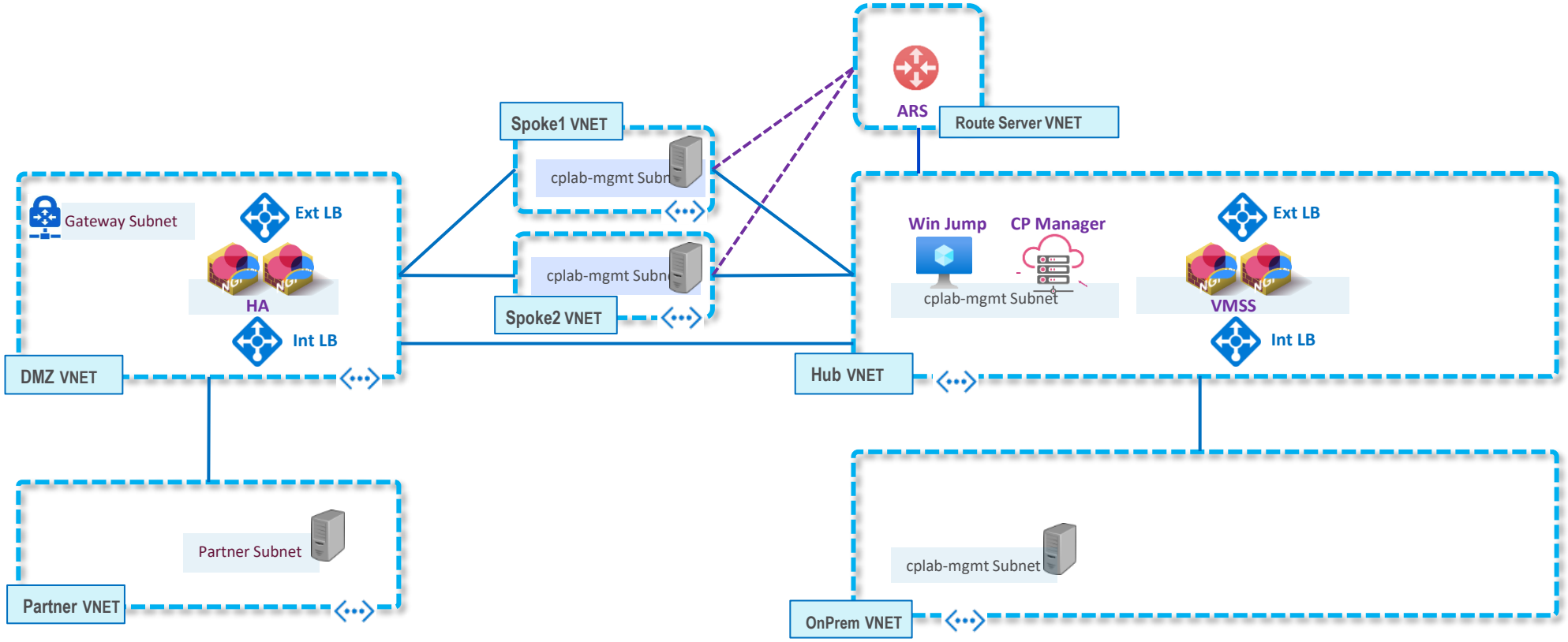


CHECK POINT

Playing with Tags

The terraform configuration script for the manager should create an Azure Datacenter. Simply click on the plus (+) sign on any rule base “Source” or “Destination” cell, pull down import and select a dynamic object from the Azure Database. For details in using Azure Dynamic objects please refer to the following link:

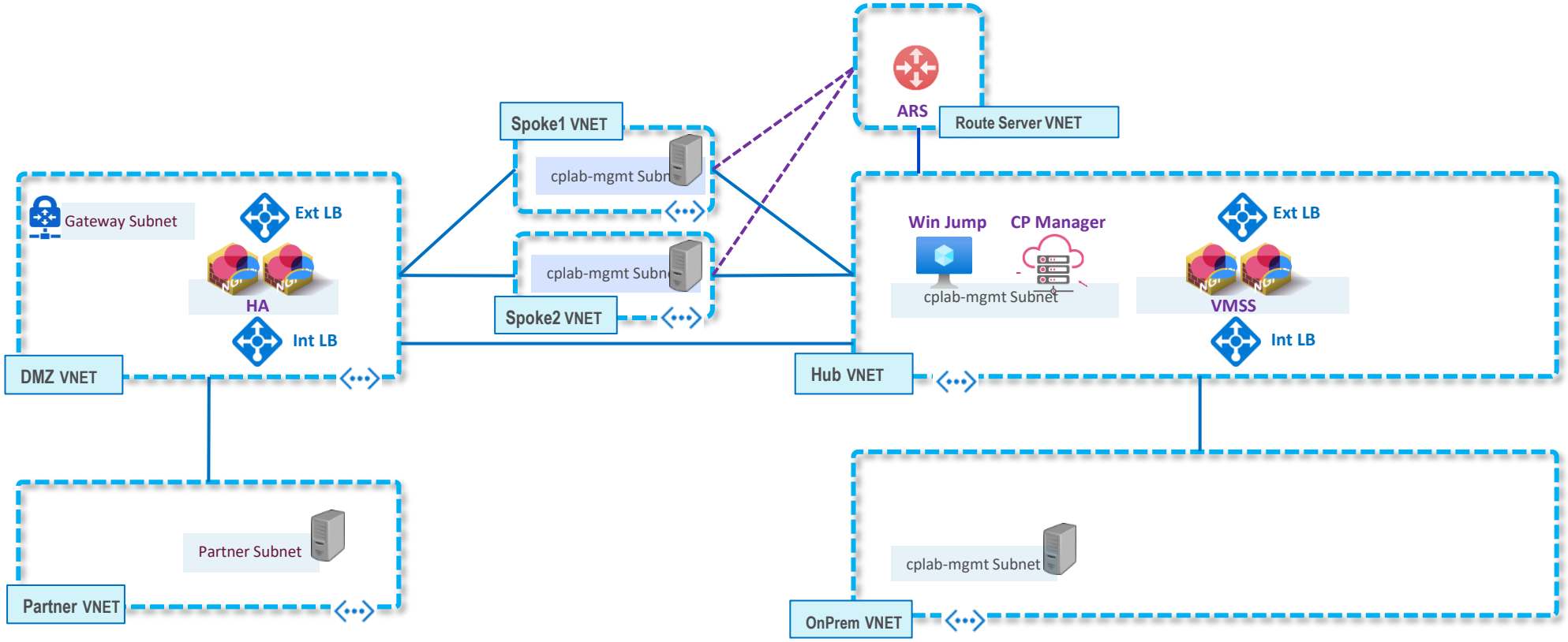
https://sc1.checkpoint.com/documents/R81.20/WebAdminGuides/EN/CP_R81.20_CloudGuard_Controller_AdminGuide/Content/Topics-CGRDG/Introduction.htm?tocpath=_____3



Playing with Auto Scaling

If you wish to test auto scaling, the easiest way to do it is to change the ”scaling” number of instances in the Azure portal. Change from 2 to 0, or from 2 to 4, or any other combination. The management server will track the changes and automatically adjust to the new number of gateways.

You can watch the CME in the management server track the activity using the tail command: tail –f /var/log/Cpcme/cme.log



CHECK POINT

Destroying the Lab

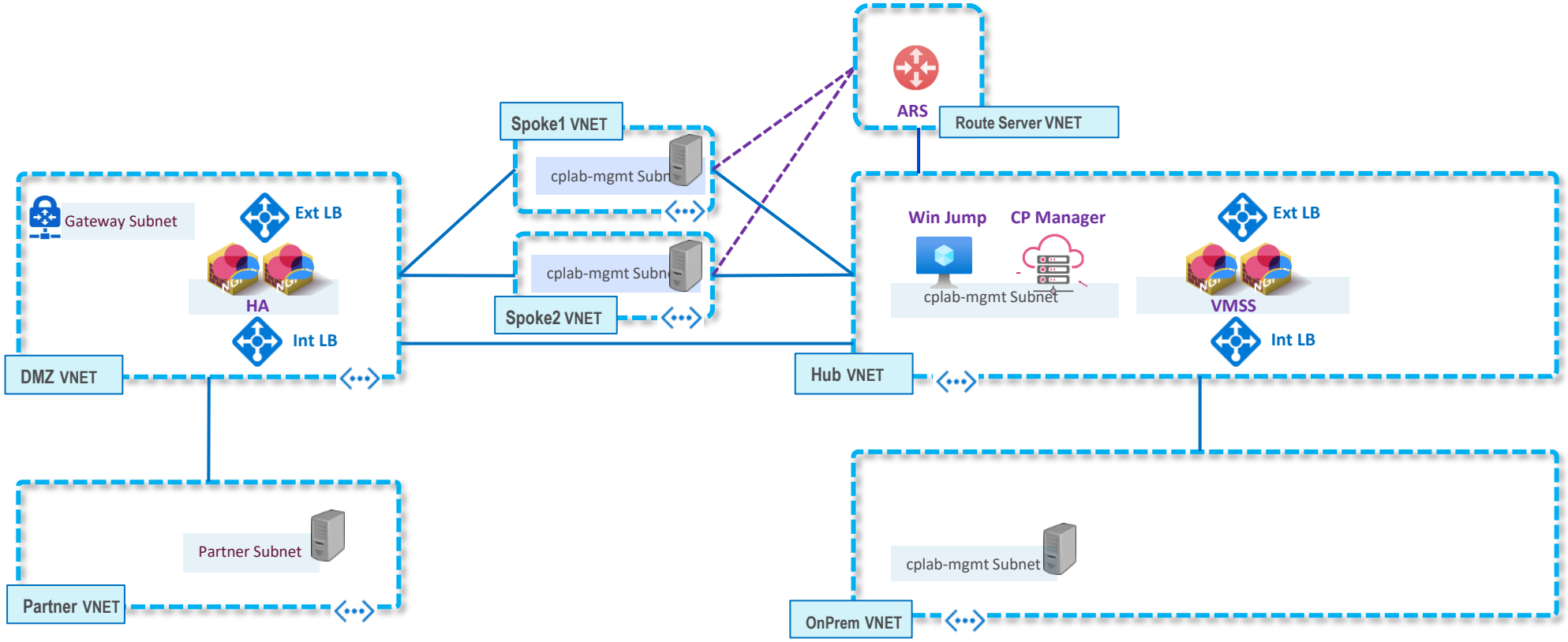
You can destroy the lab or portions of the lab using terraform destroy.

To destroy just a section, navigate to the directory for the object and run terraform destroy from inside that directory.

To destroy the entire lab, you can also run the teardown script located at the root level of the lab (./teardown.sh)

You can also delete the cplab named resource groups manually from the portal or the CLI.

If you destroy the lab manually from the portal or CLI, delete the terraform.tfstate and terraform.tfstate.backup files from the Azure_Lab directories (otherwise terraform will be confused as the current deployment will not match the remembered state).



CHECK POINT