**Terraform About**

Terraform is an infrastructure as code tool that lets you build, change, and version infrastructure safely and efficiently. This includes low-level components like compute instances, storage, and networking, as well as high-level components like DNS entries and SaaS features.

**What is Terraform?**

Terraform is an infrastructure as code tool that lets you build, change, and version cloud and on-prem resources safely and efficiently.

HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

=========

**Terraform Cheat Sheet**

Terraform command accepts a variety of subcommands such as terraform init or terraform plan. Some of the widely used commands and subcommands I have listed below.

**Show version**

terraform –version Shows terraform version installed

**Initialize infrastructure**

terraform init Initialize a working directory

terraform init -input=true Ask for input if necessary

terraform init -lock=false Disable locking of state files during state-related operations

**Get**

terraform get downloads and update modules mentioned in the root module

terraform get -update=true modules already downloaded will be checked for updates and updated

**Provision infrastructure**

terraform plan Creates an execution plan (dry run)

terraform plan -out=path save generated plan output as a file

terraform plan -destroy Outputs a destroy plan

terraform apply Executes changes to the actual environment

terraform apply –auto-approve Apply changes without being prompted to enter ”yes”

terraform apply -refresh=true Update the state for each resource prior to planning and applying

terraform apply -input=false Ask for input for variables if not directly set

terraform apply -var ‘foo=bar’ Set a variable in the Terraform configuration, can be used multiple times

terraform apply -var-file=foo Specify a file that contains key/value pairs for variable values

terraform apply -target Only apply/deploy changes to the targeted resource

terraform destroy –auto-approve Destroy/cleanup without being prompted to enter ”yes”

terraform destroy -target Only destroy the targeted resource and its dependencies

**Terraform Workspaces**

terraform workspace new Create a new workspace and select it

terraform workspace select Select an existing workspace

terraform workspace list List the existing workspaces

terraform workspace show Show the name of the current workspace

terraform workspace delete Delete an empty workspace

**Format and validate Terraform Code**

terraform fmt Format code as per HCL canonical standard

terraform validate validate configuration files for syntax

**Inspect Infrastructure**

terraform graph creates a resource graph listing all resources in your configuration and their dependencies.

terraform output List all the outputs for the root module

terraform output instance\_public\_ip List only the specified output

terraform output -json List all the outputs in JSON format

terraform show provide human-readable output from a state or plan file

First, install the required dependencies using the following command:

apt-get install wget curl unzip software-properties-common gnupg2 -y

Next, download and add the HashiCorp signed gpg keys to your system:

curl -fsSL https://apt.releases.hashicorp.com/gpg | apt-key add -

Next, add the HashiCorp repository to the APT using the following command:

apt-add-repository "deb [arch=$(dpkg --print-architecture)] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"

Next, update the repository using the command given below:

apt-get update -y

Finally, install the Terraform by running the following command:

apt-get install terraform -y

Once the Terraform has been installed, verify it using the following command:

terraform -v

You will get the Terraform version in the following output:

Terraform v1.1.2

on linux\_amd64

Install Azcli

Step 1: Get required packages for Azure CLI installation

sudo apt-get update

sudo apt-get install ca-certificates curl apt-transport-https lsb-release gnupg

Step 2: Download and install Microsoft signing key

curl -sL https://packages.microsoft.com/keys/microsoft.asc |

gpg --dearmor |

sudo tee /etc/apt/trusted.gpg.d/microsoft.gpg > /dev/null

Step 3: Add Azure CLI software repository

AZ\_REPO=$(lsb\_release -cs)

echo "deb [arch=amd64] https://packages.microsoft.com/repos/azure-cli/ $AZ\_REPO main" |

sudo tee /etc/apt/sources.list.d/azure-cli.list

Step 4: Update repository information

sudo apt-get update

Step 5: Install the azure-cli package

sudo apt-get install azure-cli

Step 6: Verify the installation

az login

On successful installation above command should open Azure sign-in page in default browser.

===================================================

Test Case,

Create storage account using azcli command or create from portal.

az group create --location $location --name $resourceGroupName

az storage account create --name $storageAccountName --resource-group $resourceGroupName --location $location --sku Standard\_LRS

az storage container create --name state --account-name $storageAccountName

az storage account blob-service-properties update --account-name $storageAccountName --enable-change-feed --enable-versioning true

**Create AKS Cluster with windows node using Azure Portal**

Create an AKS cluster

Sign in to the Azure portal.

On the Azure portal menu or from the Home page, select Create a resource.

In the Categories section, select Containers > Azure Kubernetes Service (AKS).

On the Basics page, configure the following options:

Project details:

Select an Azure Subscription.

Create an Azure Resource group, such as myResourceGroup. While you can select an existing resource group, for testing or evaluation purposes, we recommend creating a resource group to temporarily host these resources and avoid impacting your production or development workloads.

Cluster details:

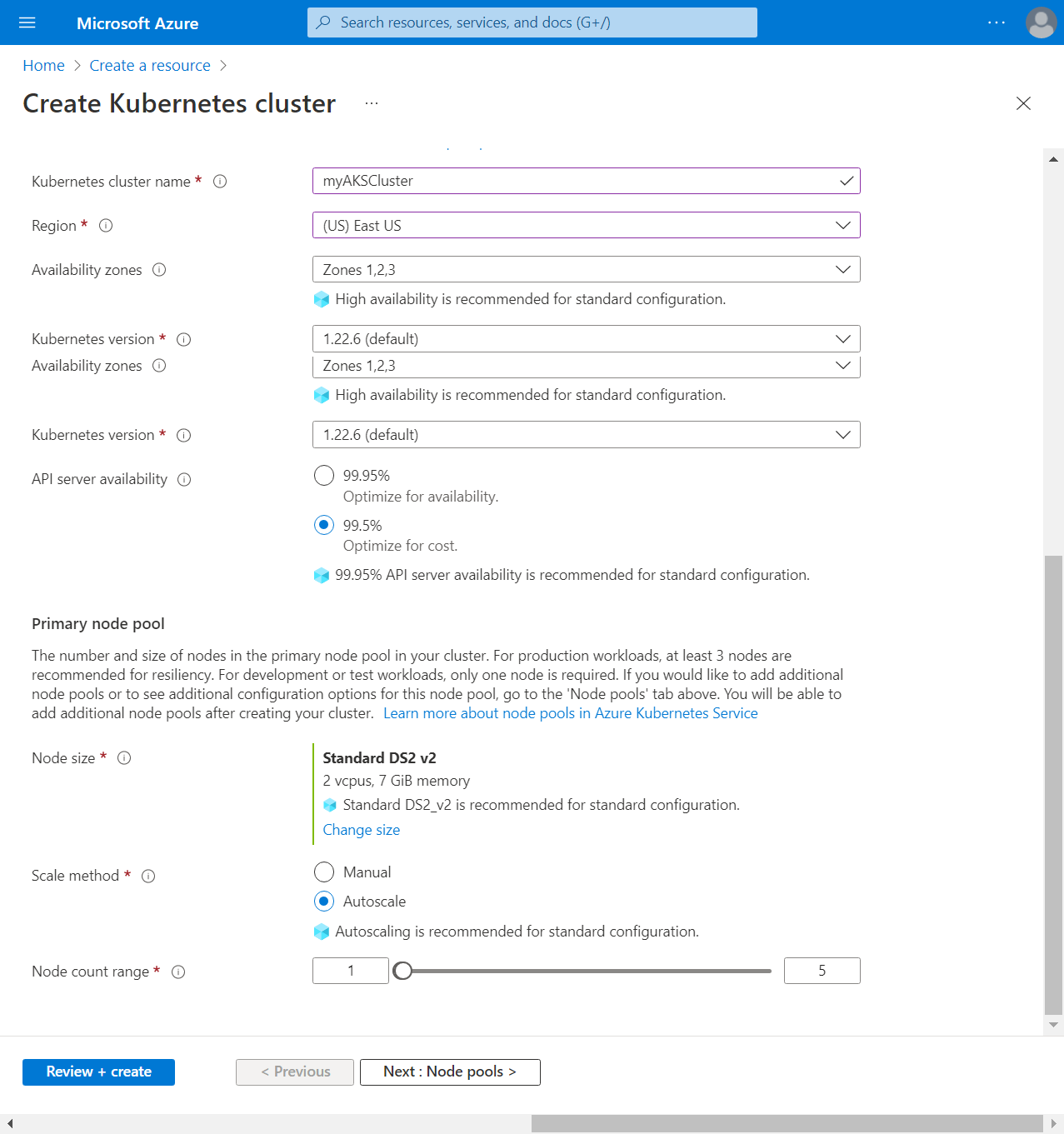
Ensure that the Preset configuration is Standard ($$). For more details on preset configurations, see Cluster configuration presets in the Azure portal.

Enter a Kubernetes cluster name, such as myAKSCluster.

Select a Region for the AKS cluster, and leave the default value selected for Kubernetes version.

Primary node pool:

Leave the default values selected.



Select Next: Node pools when complete.

Add a Windows Server node pool

Select Add node pool.

Enter a Node pool name, such as **npwin**.

For Mode, select User.

For OS type, select Windows.

Select a Node size, such as Standard\_D2s\_v3.

Select Next: Networking and set the Network policy to Azure.

Select Review + create > Create.

On the Node pools page, leave the default options and then select Next: Access.

On the Access page, configure the following options:

Select Next: Networking when complete.

**Network policy : - Azure**

Keep the default Networking options, which uses the kubenet networking plug-in, and then select Next: Integrations.

Keep the default Integrations options and then select Next: Advanced.

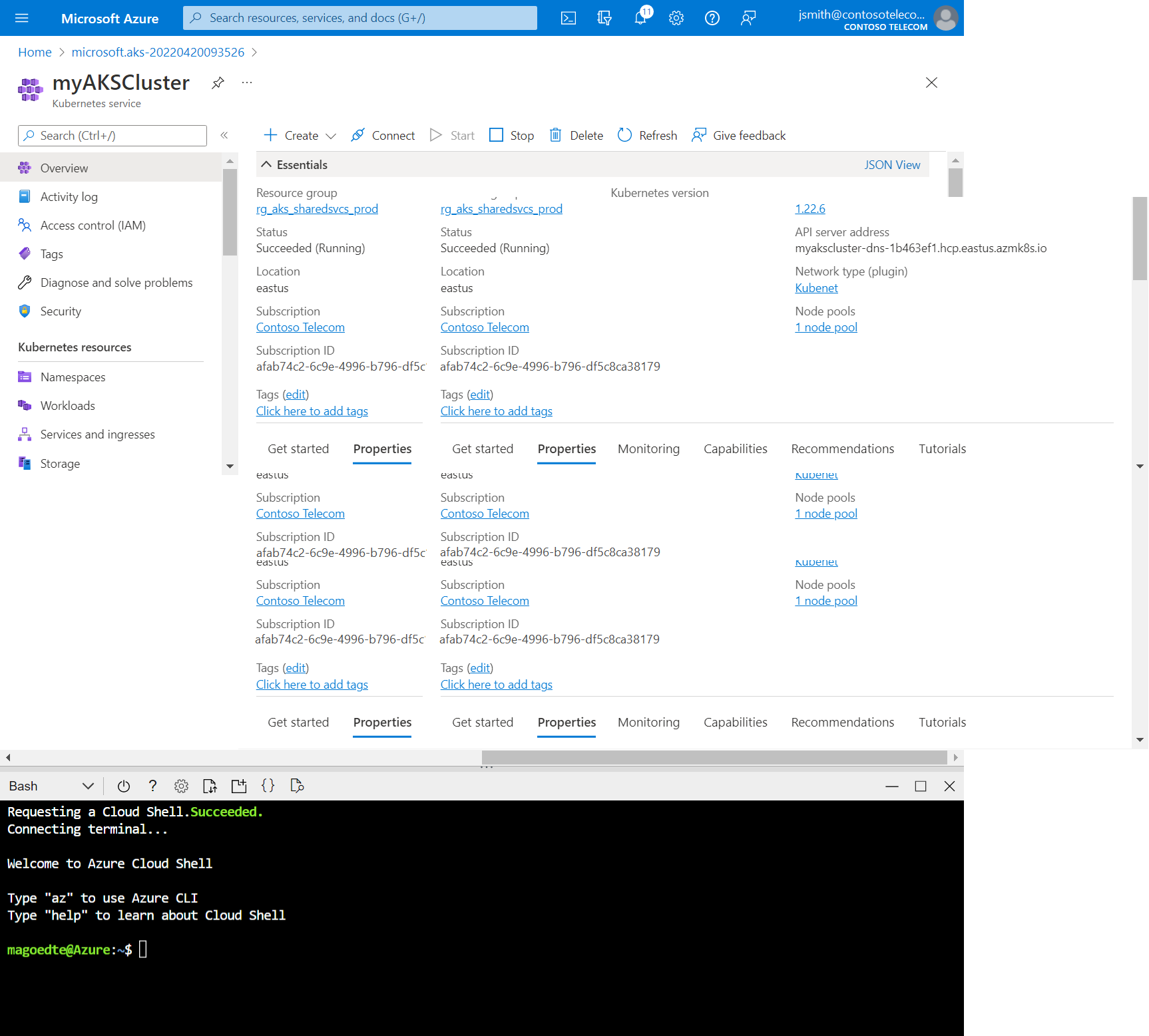
Keep the default Advanced options and then select Next: Tags.

On the tags page, leave the default option and then select Next: Review + create.

When you navigate to the Review + create tab, Azure runs validation on the settings that you have chosen. If validation passes, you can proceed to create the AKS cluster by selecting Create. If validation fails, then it indicates which settings need to be modified.

It takes a few minutes to create the AKS cluster. When your deployment is complete, navigate to your resource by either:

**Connect to the Cluster**



az aks get-credentials --resource-group myResourceGroup --name myAKSCluster

kubectl get nodes

NAME STATUS ROLES AGE VERSION

aks-agentpool-87331340-vmss000000 Ready agent 8m53s v1.25.6

aks-agentpool-87331340-vmss000001 Ready agent 8m51s v1.25.6

aks-agentpool-87331340-vmss000002 Ready agent 8m57s v1.25.6

Create application Yaml

Vi sampleapp.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: sample

labels:

app: sample

spec:

replicas: 1

template:

metadata:

name: sample

labels:

app: sample

spec:

nodeSelector:

"kubernetes.io/os": windows

containers:

- name: sample

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

ports:

- containerPort: 80

selector:

matchLabels:

app: sample

---

apiVersion: v1

kind: Service

metadata:

name: sample

spec:

type: LoadBalancer

ports:

- protocol: TCP

port: 80

selector:

app: sample

kubectl apply -f sampleapp.yaml

root@NCSUSNAUJPWS01:~/akswin# kubectl get node

NAME STATUS AGE VERSION

aks-lin-10005819-vmss000000 Ready 2h v1.26.6

akswin000000 Ready 1h v1.26.6

root@NCSUSNAUJPWS01:~/akswin#

root@NCSUSNAUJPWS01:~/akswin# kubectl get pod

NAME READY STATUS RESTARTS AGE

sample-65c7dcd9b7-pfsmk 1/1 Running 0 13m

root@NCSUSNAUJPWS01:~/akswin# kubectl get svc

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes 10.0.0.1 <none> 443/TCP 2h

sample 10.0.66.1 52.191.217.125 80:31805/TCP 13m

Check

http://52.191.217.125

