QATIP Intermediate
Azure BonusLab02
Deploying AKS resources using Terraform

Contents

Overview

The objective of this lab is to provision an Azure Kubernetes Service (AKS) cluster and deploy a sample NGINX application into it using Terraform. The challenge focuses on understanding the core components required to set up Kubernetes infrastructure in Azure and manage workloads using Infrastructure as Code (IaC) principles. This lab is designed to help you understand the different approaches and considerations when deploying and managing applications in Kubernetes using Terraform and Helm. The lab is divided into two main phases:

1. **Phase 1:** Deploy applications using Terraform with three modifications:
    - **Run 1:** Deployment without pod resource controls or deployment autoscaling.
    - **Run 2:** Deployment with pod resource controls but no deployment autoscaling
    - **Run 3:** Deployment with pod resource controls and deployment autoscaling.
2. **Phase 2:** Deploy applications using Helm charts, managed by Terraform.

Components to be Created

- **Azure Resource Group**: A container to manage related resources in Azure.
- **AKS Cluster**: A managed Kubernetes cluster hosted in Azure.
- **Kubernetes Deployment**: A deployment to manage the desired state of the NGINX application.
- **Kubernetes Service**: A LoadBalancer service to expose the NGINX application externally.
- **Terraform Outputs**: Outputs to retrieve important details like the cluster kubeconfig and service IP.
- **Helm chart**: A single resource for all NGINX requirements.

Before you begin

Ensure you have completed Lab0 before attempting this lab.

In the IDE terminal pane, enter the following command...

cd c:\azure-tf-int\lab\bonus02

This shifts your current working directory to labs\bonus02. Ensure all commands are executed in this directory

Close any open files and use the Explorer pane to navigate to and open the labs\bonus02 folder.

Review the pre-provisioned files...

main.tf - defines the AKS cluster resource

providers.tf - defines three providers; azurerm, kubernetes and helm

outputs.tf - outputs kubernetes cluster config and nginx service IP

helm.tf1 - Helm chart (with tf1 extension, so not used initially)

deployments.tf -Kubernetes nginx deployment, service and autoscaler

Phase1:Run1 Deployment without pod resource controls

Introduction

1. The current terraform files will deploy a 2-node AKS cluster in East US (main.tf). Into this cluster, it will then deploy a kubernetes deployment comprising of 2 replicas of a nginx pod which will sit behind a Loadbalancer service, front-faced with a public IP address (see deployments.tf).

Steps

2. In your IDE terminal, ensure you have navigated to **c:\azure-tf-int\labs\bonus02**

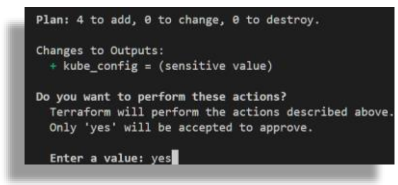
3. Update line **11** of **provider.tf** with your subscription id and then initialize terraform

   terraform init

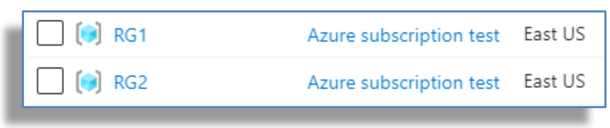
4. Once initialized run the following commands to view, and then apply the deployment

   terraform plan

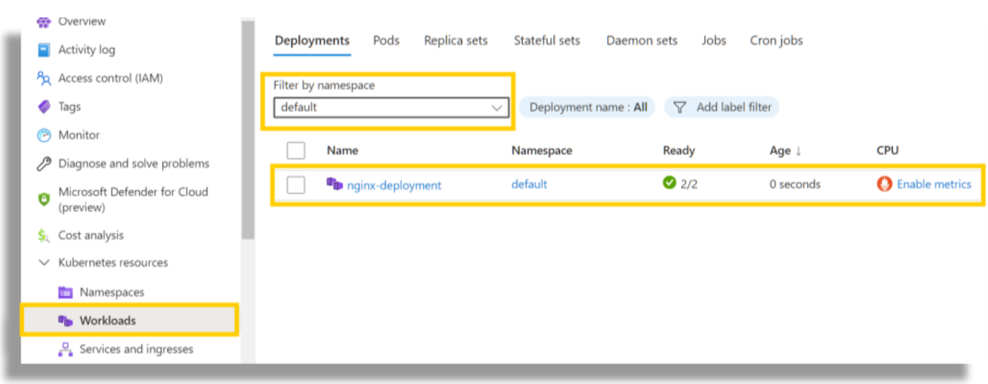
   terraform apply followed by **yes**

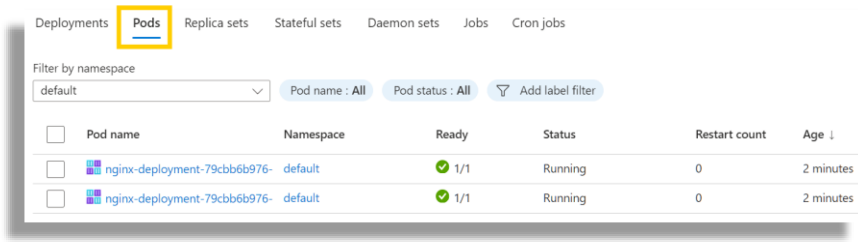


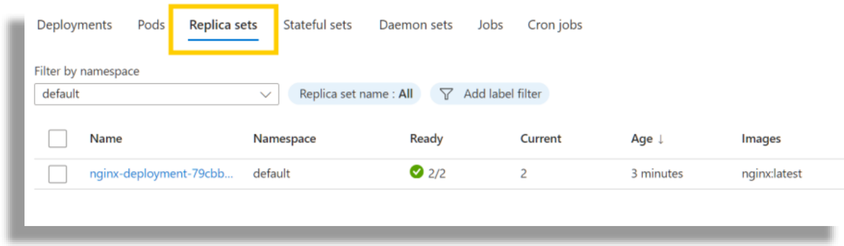
5. The deployment may take up to 5 minutes to complete. Navigate to the Azure Console and verify the creation of 2 resource groups. The main resource group, RG1, is where the core cluster components are created, whilst the 2nd one, RG2, is the node pool resource group and is created automatically. It is auto-named by default, but its name can be defined, as has been done here to comply with resource group naming policy restrictions...

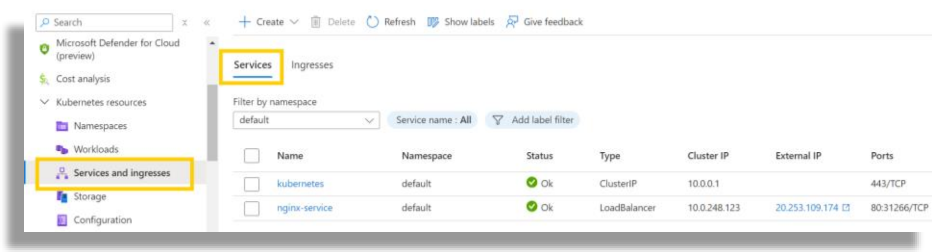


6. Review the deployed kubernetes resources, selecting the 'default' namespace; 1 deployment, 2 pods, 1 Replica set and 1 Service ...





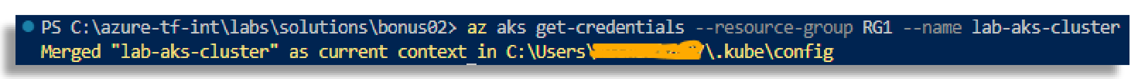




7. Aside from the GUI, **kubectl** can be used to interact with the cluster API service for cluster administration tasks. Running **kubectl** commands against the cluster requires authentication and cluster identification. These details are stored in a config file in the .kube folder of your home directory by default. The following command will get your cluster details and credential and create a new .kube\config file in your home directory...

   Remove-Item -Path "$HOME\.kube\config" -Force

   az aks get-credentials --resource-group RG1 --name lab-aks-cluster



Note: Re-run these commands if you destroy and then recreate the cluster

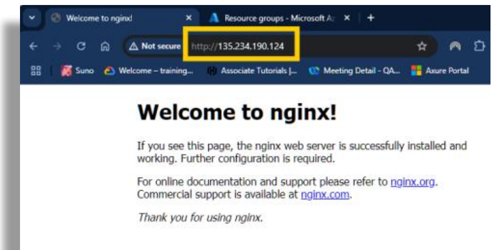
8. To view the results of the Kubernetes deployment using terraform, run the following commands...

   kubectl get services

```
PS C:\azure-tf-int\lab1> kubectl get services
NAME            TYPE           CLUSTER-IP    EXTERNAL-IP       PORT(S)        AGE
kubernetes      ClusterIP      10.0.0.1      <none>            443/TCP        5m9s
nginx-service   LoadBalancer   10.0.42.144   135.234.190.124   80:30261/TCP   2m57s
PS C:\azure-tf-int\lab1>
```

This command reveals the public IP address of the LoadBalancer service that is sitting in front of the nginx pods.

Browse to this address using http://<LoadBalancer External IP> ....



kubectl get deployments

```
PS C:\azure-tf-int\lab1> kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2/2     2            2           5m34s
PS C:\azure-tf-int\lab1>
```

This command shows that the nginx deployment is for 2 replica pods and that both are up to date and available.

kubectl get nodes

```
PS C:\azure-tf-int\lab1> kubectl get nodes
NAME                                STATUS   ROLES    AGE     VERSION
aks-default-19681873-vmss000000     Ready    <none>   7m35s   v1.30.7
aks-default-19681873-vmss000001     Ready    <none>   7m30s   v1.30.7
PS C:\azure-tf-int\lab1>
```

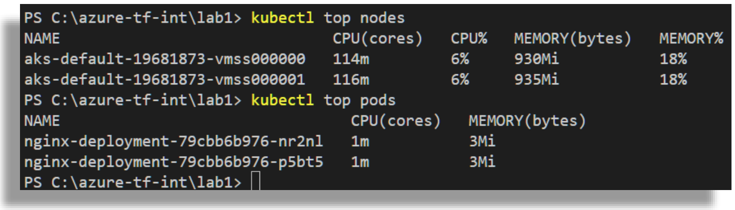
This command shows the 2 nodes that make up the cluster.

kubectl get pods

```
PS C:\azure-tf-int\lab1> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-79cbb6b976-nr2nl   1/1     Running   0          8m7s
nginx-deployment-79cbb6b976-p5bt5   1/1     Running   0          8m7s
PS C:\azure-tf-int\lab1>
```

This command identifies the name of the current pods that make up this 2 replica deployment

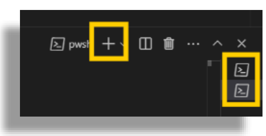
kubectl top nodes and **kubectl top pods**


```
PS C:\azure-tf-int\lab1> kubectl top nodes
NAME                           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
aks-default-19681873-vmss000000   114m         6%     930Mi           18%
aks-default-19681873-vmss000001   116m         6%     935Mi           18%
PS C:\azure-tf-int\lab1> kubectl top pods
NAME                                CPU(cores)   MEMORY(bytes)
nginx-deployment-79cbb6b976-nr2nl   1m           3Mi
nginx-deployment-79cbb6b976-p5bt5   1m           3Mi
PS C:\azure-tf-int\lab1>
```

These commands show the current CPU and memory utilization levels being supported by the nodes as generated by the pods

9. We will now access a pod and run a CPU intensive process to view and understand how Kubernetes deals with this. Note that the pod specification (lines 28 to 36 of **deployments.tf)** currently has no resource limits applied to the container that runs inside the pods.

10. Open a new terminal session, identify one of the pods in the deployment, connect to it and initialize a CPU intensive process, “stress”....



kubectl get pods
kubectl exec -it <pod name> -- bash

Then inside the pod:

apt-get update
apt-get install -y stress
stress --cpu 4 --timeout 600

11. Switch to your initial terminal

kubectl top pods and **kubectl top nodes**

```
PS C:\azure-tf-int\lab1> kubectl top pods
NAME                                CPU(cores)   MEMORY(bytes)
nginx-deployment-79cbb6b976-nr2nl   1844m        21Mi
nginx-deployment-79cbb6b976-p5bt5   1m           3Mi
PS C:\azure-tf-int\lab1> kubectl top nodes
NAME                              CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
aks-default-19681873-vmss000000   143m         7%     940Mi           18%
aks-default-19681873-vmss000001   2000m        105%   976Mi           19%
PS C:\azure-tf-int\lab1>
```

You may need to re-run these command a few times to get updated values. Notice how the one pod is consuming all of the CPU on one of the cluster nodes.

12. Switch to second terminal and kill the "stress" process

Ctrl+c
exit

13. Switch back to initial terminal

kubectl top pods and **kubectl get nodes**

Periodically re-run these commands....

```
PS C:\azure-tf-int\lab1> kubectl top pods
NAME                                CPU(cores)   MEMORY(bytes)
nginx-deployment-79cbb6b976-nr2nl   1m           21Mi
nginx-deployment-79cbb6b976-p5bt5   1m           3Mi
PS C:\azure-tf-int\lab1> kubectl top nodes
NAME                              CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
aks-default-19681873-vmss000000   115m         6%     975Mi           19%
aks-default-19681873-vmss000001   112m         5%     1003Mi          19%
PS C:\azure-tf-int\lab1>
```

Analysis

Without constraints, a pod can demand excessive resources from the node on which it is running. This will affect the overall capacity of your cluster and is not best practice.

Phase1:Run2 Deployment with pod resource controls

Introduction

1. In this run you will impose resource constraints on the pods that make up the deployment and test the impact.

Steps

2. Un-comment lines 29-36 in **deployments.tf** to set resource limits on your pods...

```
# Add resource requests and limits
resources {
  requests = {
    cpu = "50m"
  }
  limits = {
    cpu = "60m"
  }
}
```

3. Apply these changes: **terraform apply** followed by **yes**

```
~ container {
      name                          = "nginx"
      # (10 unchanged attributes hidden)

    ~ resources {
        ~ limits   = {
            + "cpu" = "60m"
          }
        ~ requests = {
            + "cpu" = "50m"
          }
      }
```

```
Plan: 0 to add, 1 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

4. Switch to your second terminal session, identify one of the pods in the deployment (pods are immutable so the changes to the deployment will have destroyed the original pods and recreated new ones), connect to it and initialize the cpu intensive process, “stress”….,

   kubectl get pods
   kubectl exec -it <pod name> -- bash

   Then inside the pod:

   apt-get update
   apt-get install -y stress
   stress --cpu 4 --timeout 600

5. Switch back to your initial terminal session..

   kubectl top pods/nodes

```
NAME                              CPU(cores)   MEMORY(bytes)
nginx-deployment-d8745b45b-tvss9  60m          21Mi
nginx-deployment-d8745b45b-wvqlf  1m           3Mi
```

```
NAME                         CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
aks-default-33338017-vmss000000   121m         6%     912Mi           18%
aks-default-33338017-vmss000001   173m         9%     916Mi           18%
```

Repeat these commands several time and observe the increase in pod cpu utilization and corresponding increase in node cpu utilization. The pod will not be granted more than 60m of CPU, as defined in the pod specification...

6. Switch to your second terminal and kill the stress processes

   Ctrl+c
   exit

7. Kill this terminal (right click over highlighted terminal and select Delete), leaving 1 terminal session..

8. Monitor the load decrease...

   kubectl top pods and **kubectl top nodes**

   Repeat these commands several time and observe the decrease in pod cpu demand and corresponding decrease in node cpu utilization.

```
PS C:\azure-tf-int\lab1> kubectl top pods
NAME                               CPU(cores)   MEMORY(bytes)
nginx-deployment-66757df766-hcr6v   1m           20Mi
nginx-deployment-66757df766-jf9h8   1m           3Mi
PS C:\azure-tf-int\lab1> kubectl top nodes
NAME                              CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
aks-default-32130711-vmss000000   144m         7%     1015Mi          20%
aks-default-32130711-vmss000001   118m         6%     970Mi           19%
PS C:\azure-tf-int\lab1>
```

Analysis

With constraints, a pod cannot demand excessive resources from the node on which it is running. This will ensure the nodes are not overutilized, but the application itself may perform poorly under heavy demand situations.

Phase1:Run3 Deployment with horizontal pod autoscaler

Introduction

1. In this run you will leave resource constraints on the pods in place but allow more capacity as necessary by utilizing a horizontal pod autoscaler (hpa) that allows more replicas of the pod to be initialized based on pod resource utilization levels.

   Note. In this lab, the target cpu utilization is set to 5%, indicating that a pod generating more that 5% CPU utilization be considered overworked. This threshold value would be much higher in production environments.

Steps

2. Un-comment lines 65-82 in **deployments.tf** to configure the horizontal pod autoscaler. Note that it specifies a minimum of **2** and a maximum of **4** replicas. It also sets the target cpu utilization at **5%**

```
resource "kubernetes_horizontal_pod_autoscaler" "nginx_hpa" {
  depends_on = [kubernetes_deployment.nginx]

  metadata {
    name = "nginx-hpa"
  }

  spec {
    max_replicas = 4
    min_replicas = 2
    scale_target_ref {
      api_version = "apps/v1"
      kind        = "Deployment"
      name        = kubernetes_deployment.nginx.metadata[0].name
    }
    target_cpu_utilization_percentage = 5 # Trigger scaling if CPU
  }
}
```

3. Plan and apply these changes..

   terraform plan

   terraform apply followed by **yes**

```
+ spec {
    + max_replicas                      = 4
    + min_replicas                      = 2
    + target_cpu_utilization_percentage = 5
```

```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

4. Generate web traffic against the load balancer by running 10 parallel web requests against it, replacing **<nginx_service_ip>** with the IP displayed as output in your terminal...

   for ($i = 0; $i -lt 10; $i++) { Start-Job -ScriptBlock { while ($true) { Invoke-WebRequest -Uri http://<nginx_service_ip> -UseBasicParsing | Out-Null } } }

```
PS C:\azure-tf-int\lab1> for ($i = 0; $i -lt 10; $i++) { Start-Job -ScriptBlock { while ($true) { Invoke-WebRequest -Uri http://172.171.165.255 -UseBasicParsing | Out-Null } } }

Id     Name            PSJobTypeName   State         HasMoreData     Location             Command
--     ----            -------------   -----         -----------     --------             -------
1      Job1            BackgroundJob   Running       True            localhost             while ($true) { Invoke-…
3      Job3            BackgroundJob   Running       True            localhost             while ($true) { Invoke-…
5      Job5            BackgroundJob   Running       True            localhost             while ($true) { Invoke-…
7      Job7            BackgroundJob   Running       True            localhost             while ($true) { Invoke-…
9      Job9            BackgroundJob   Running       True            localhost             while ($true) { Invoke-…
11     Job11           BackgroundJob   Running       True            localhost             while ($true) { Invoke-…
13     Job13           BackgroundJob   Running       True            localhost             while ($true) { Invoke-…
15     Job15           BackgroundJob   Running       True            localhost             while ($true) { Invoke-…
17     Job17           BackgroundJob   Running       True            localhost             while ($true) { Invoke-…
19     Job19           BackgroundJob   Running       True            localhost             while ($true) { Invoke-…
```

5. Periodically run **kubectl get hpa** to view the horizontal pod autoscaler increase the number of replicas from 2 to 3 or 4 as the cpu utilization of the initial pods exceeds 5%...

```
PS C:\azure-tf-int\lab1> kubectl get hpa
NAME        REFERENCE                     TARGETS     MINPODS   MAXPODS   REPLICAS
nginx-hpa   Deployment/nginx-deployment   cpu: 5%/5%  2         4         4
```

6. List the currently running pods and their cpu load using **kubectl top pods** (Your values may differ)

```
PS C:\azure-tf-int\lab1> kubectl top pods
NAME                               CPU(cores)   MEMORY(bytes)
nginx-deployment-66757df766-8cd9t  5m           20Mi
nginx-deployment-66757df766-c92fh  5m           3Mi
nginx-deployment-66757df766-cwhj9  5m           3Mi
nginx-deployment-66757df766-nb7kh  5m           3Mi
```

The issue

7. Kubernetes has increased the pod count but the terraform files state there should be 2 replicas and plan/apply operations would propose destroying 2 pods, contradicting what the kubernetes control pane thinks the kubernetes deployment state should be.

   terraform plan

```
~ spec {
    ~ replicas                    = "4" -> "2"
      # (4 unchanged attributes hidden)
```

8. Stop the running jobs that are generating web-traffic..

   Get-Job | Where-Object { $_.Command -like "*Invoke-*" } | Stop-Job
   Get-Job | Where-Object { $_.State -eq 'Stopped' } | Remove-Job

9. Wait till the autoscaler scales down the deployment, this can take 5 minutes so grab a break or read ahead, then run **kubectl get hpa** and **kubectl top pods** periodically until replica count drops back to 2...

```
PS C:\azure-tf-int\lab1> kubectl top pods
NAME                               CPU(cores)   MEMORY(bytes)
nginx-deployment-66757df766-hcr6v   1m           20Mi
nginx-deployment-66757df766-jf9h8   1m           3Mi
PS C:\azure-tf-int\lab1> kubectl get hpa
NAME        REFERENCE                    TARGETS     MINPODS   MAXPODS   REPLICAS   AGE
nginx-hpa   Deployment/nginx-deployment   cpu: 1%/5%   2         4         2          29m
```

10. Re-run **terraform plan**...

```
No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your config
```

11. Note that *now* the kubernetes state matches the terraform state. This fluctuating behaviour can cause state inconsistencies

Analysis

12. While Terraform provides robust infrastructure management capabilities, managing Kubernetes deployments directly with Terraform introduces certain challenges. Terraform maintains a state

file to track the resources it manages. If changes are made outside of Terraform (e.g., scaling deployments with kubectl scale), the actual cluster state can drift from Terraform's known state.

13. Kubernetes resources, especially pods, are dynamic by nature. Terraform, designed for more static infrastructure, may struggle to track transient changes like pod restarts, rescheduling, or autoscaling activities. As deployments grow in complexity, managing all configurations (like ConfigMaps, Secrets, RBAC policies) within Terraform can become cumbersome compared to Kubernetes-native tools. Some Kubernetes resource configurations and advanced features may not be fully supported or exposed through the Terraform Kubernetes provider, requiring workarounds or custom solutions.

Phase2 Deployment using Helm chart

Introduction to Helm charts

Helm is a powerful package manager for Kubernetes that simplifies the deployment, configuration, and management of applications within a Kubernetes cluster. Much like how apt or yum manage packages on Linux systems, Helm manages Kubernetes applications through reusable templates known as **Helm charts**. These charts bundle Kubernetes manifests (like deployments, services, and config maps) with default configuration values, allowing users to deploy complex applications with just a few commands or configuration adjustments.

When managing Kubernetes with Terraform, Helm plays a crucial role in reducing complexity. While Terraform excels at infrastructure provisioning—such as setting up clusters, networking, and cloud resources—managing dynamic, application-level resources like deployments, services, and autoscalers can become cumbersome. Defining these resources directly with Terraform requires extensive boilerplate code, and handling updates, rollbacks, or versioning manually increases operational overhead.

Integrating Helm with Terraform bridges this gap. Terraform's **Helm provider** allows you to deploy and manage Helm charts declaratively, just like other infrastructure components. Instead of writing verbose Kubernetes resource definitions, you simply reference a Helm chart and specify key configuration values. This approach reduces code duplication, simplifies resource management, and leverages the Kubernetes community's best practices baked into popular charts. Additionally, Helm's versioning and rollback features provide greater flexibility in managing application lifecycle changes, making deployments more resilient and adaptable in dynamic environments.

By combining Terraform's infrastructure-as-code capabilities with Helm's application management strengths, teams can achieve a more streamlined, efficient, and scalable approach to Kubernetes operations.

Steps

1. Run **terraform state list** to review the currently deployed resources...

```
PS C:\azure-tf-int\lab1> terraform.exe state list
azurerm_kubernetes_cluster.aks
azurerm_resource_group.rg
kubernetes_deployment.nginx
kubernetes_horizontal_pod_autoscaler.nginx_hpa
kubernetes_service.nginx
PS C:\azure-tf-int\lab1>
```

2. Rename **deployments.tf** to **deployments.tf1** (right-click over file and select rename), thus removing it from consideration by terraform.

3. Comment out lines 1-4 in **outputs.tf**

4. Run **terraform apply** followed by **yes** to delete the resources previously specified in the deployment file..

```
Plan: 0 to add, 0 to change, 3 to destroy.
```

5. Run **terraform state list** again to review the remaining deployed resources...

```
PS C:\azure-tf-int\lab1> terraform.exe state list
azurerm_kubernetes_cluster.aks
azurerm_resource_group.rg
PS C:\azure-tf-int\lab1>
```

6. Rename **helm.tf1** to **helm.tf**, thus bringing it into consideration by terraform

7. Apply these changes with **terraform plan**

```
          # (1 unchanged attribute hidden)
      }
    + set {
        + name  = "resources.limits.cpu"
        + value = "200m"
          # (1 unchanged attribute hidden)
      }
    + set {
        + name  = "resources.requests.cpu"
        + value = "100m"
          # (1 unchanged attribute hidden)
      }
    + set {
        + name  = "service.type"
        + value = "LoadBalancer"
          # (1 unchanged attribute hidden)
      }
  }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Notice that terraform will now create 1 resource as opposed to three previously. This is a helm chart for nginx and allows multiple values to be set. Scroll up to review.

8. Deploy the chart using **terraform apply**, followed by **yes**

9. Once deployed, review the kubernetes resources that have been created by running the following commands:

kubectl get services
kubectl get hpa
kubectl get pods
kubectl get deployments

```
PS C:\azure-tf-int\lab1> kubectl get services
NAME         TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
kubernetes   ClusterIP      10.0.0.1      <none>         443/TCP        32m
nginx        LoadBalancer   10.0.77.219   4.255.18.127   80:30230/TCP   4m18s
PS C:\azure-tf-int\lab1>
PS C:\azure-tf-int\lab1> kubectl get hpa
NAME    REFERENCE          TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
nginx   Deployment/nginx   cpu: 1%/80%  2         4         2          4m24s
PS C:\azure-tf-int\lab1>
PS C:\azure-tf-int\lab1> kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-65777545b-q2vwx    1/1     Running   0          4m29s
nginx-65777545b-wv9g2    1/1     Running   0          4m14s
PS C:\azure-tf-int\lab1>
PS C:\azure-tf-int\lab1> kubectl get deployments
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   2/2     2            2           4m34s
PS C:\azure-tf-int\lab1>
```

10. Run **terraform state list**

```
PS C:\azure-tf-int\lab1> terraform state list
azurerm_kubernetes_cluster.aks
azurerm_resource_group.rg
helm_release.nginx
```

11. The single helm chart deploys the same resources previously configured across several separate resource blocks, greatly simplifying administration.

12. The nginx chart offers many configuration values that can be set, see: [nginx helm charts]

Lab Clean-Up

1. Destroy all deployed resources with **terraform destroy** followed by **yes**

```
Plan: 0 to add, 0 to change, 3 to destroy.

Changes to Outputs:
  - kube_config = (sensitive value) -> null

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

2. Switch to the console and verify the deletion of the 2 resource groups associated with this lab.

Congratulations, you have completed this lab