secure access to AKS Nodes via different methods

Securing access to AKS Nodes is essential for secure and controlled Nodes. By default, AKS won’t expose any nodes to public internet. However, still you can set it more securely using SSH methods and Others.

Below are a few best practices for managing SSH access to AKS nodes.

# 1. Using Azure Bastion for Secure SSH Access

Azure Bastion provides security and SSH, RDP access to VM directly in the Azure Portal over SSL. It acts as a jump server without needing the Public IP.

Steps:

1. **Deploy Azure Bastion**:
   * Go to the Azure portal and navigate to your virtual network (VNet) that contains your AKS nodes.
   * Under "Operations," select "Bastion" and click "Create."
   * Follow the prompts to deploy Azure Bastion in the selected VNet.
2. **Connect to AKS Nodes via Bastion**:
   * In the Azure portal, navigate to the Virtual Machine Scale Set (VMSS) that backs your AKS nodes.
   * Select the specific instance (node) you want to access.
   * Click "Connect," then choose "Bastion" and use the SSH option to securely access the node.

**Benefits:**

* **No Public IP Required**: Nodes remain isolated from the public internet.
* **Secure Connection**: Connections are encrypted using SSL, ensuring secure communication.

# 2. Using Private IP with VPN or ExpressRoute

# Create an AKS cluster with SSH key

* + 1. **Configure SSH Access During Cluster Creation**

When creating an AKS cluster, you can configure SSH access to your nodes by specifying an SSH public key. This method is useful if you anticipate needing SSH access to nodes regularly.

Steps:

1. Generate an SSH Key Pair:

* On your local machine, generate an SSH key pair if you don't already have one:

|  |
| --- |
| Bash Copy code  **ssh-keygen -t rsa -b 4096 -C "your\_email@example.com** |

1. Specify the SSH Key During AKS Cluster Creation:

* When creating the AKS cluster via the Azure CLI, provide the SSH public key:

|  |
| --- |
| Bash Copy code  **az aks create \**  **--resource-group <resource-group-name> \**  **--name <aks-cluster-name> \**  **--ssh-key-value ~/.ssh/id\_rsa.pub** |

1. Access the Nodes:

* After the cluster is created, you can SSH into any node using:

|  |
| --- |
| Bash Copy code  **ssh azureuser@<node-private-ip>** |

Benefits:

* **Controlled Access**: SSH access is limited to those with the specified SSH key.
* **Flexibility**: SSH access is baked into the cluster from the start.

# 4. Disable SSH Root Access and Enforce Key-Based Authentication

To further secure SSH access, you should disable root login and ensure that only key-based authentication is allowed.

Steps:

1. Disable Root Login:

* SSH into the AKS node and edit the SSH configuration file:

|  |
| --- |
| Bash Copy code  **sudo nano /etc/ssh/sshd\_config** |

* Set PermitRootLogin to **‘no’**:

|  |
| --- |
| Bash Copy code  **PermitRootLogin no** |

1. Enforce Key-Based Authentication:

* Ensure PasswordAuthentication is set to no:

|  |
| --- |
| Bash Copy code  **PasswordAuthentication no** |

* Restart the SSH service to apply changes:

|  |
| --- |
| Bash Copy code  **Sudo systemctl restart sshd** |

Benefits:

* **Enhanced Security**: Prevents unauthorized access through brute-force password attacks.
* **Best Practices**: Aligns with security best practices for SSH.

# Update SSH public key on an existing AKS cluster

This operation updates the key on all node pools. You can either specify a key or a key file using the --ssh-key-value argument.

* To specify a new SSH public key value, include the --ssh-key-value argument:

|  |
| --- |
| Azure CLI Copy    **az aks update --name myAKSCluster --resource-group MyResourceGroup --ssh-key-value 'ssh-rsa AAAAB3Nza-xxx'** |

* To specify an SSH public key file, specify it with the --ssh-key-value argument:

|  |
| --- |
| Azure CLI Copy  **az aks update --name myAKSCluster --resource-group MyResourceGroup --ssh-key-value ~/.ssh/id\_rsa.pub** |

! Important

When you update the SSH Key for an existing AKS cluster, then new SSH key, AKS doesn’t automatically update your node pool. To apply the updated SSH key to all nodes in the pool, you need to perform a node pool update operation. it typically done through a **node image upgrade** or by manually updating the node pool.

Node Image Upgrade:

|  |
| --- |
| az aks nodepool upgrade \  --resource-group <your-resource-group> \  --cluster-name <your-aks-cluster-name> \  --name <your-node-pool-name> \  **--node-image-only** |

Important Considerations:

* **Downtime**: Depending on your workload and node pool configuration (e.g., the use of multiple availability zones, the number of replicas), upgrading node images may cause some downtime as nodes are drained and re-imaged.
* **Rolling Update**: The node pool update is performed as a rolling update, where nodes are updated one at a time to minimize disruption. However, plan for potential service interruptions.
* **SSH Key Management**: If you rotate SSH keys regularly for security reasons, remember to follow this process each time to ensure the new keys are applied to all nodes.
* **Terraform Configuration**: If you use Terraform, make sure to update your Terraform configuration with the new SSH key and apply the changes. After applying, use the node pool update process as described.

Summary:

Yes, you can create an AKS (Azure Kubernetes Service) cluster with SSH keys using Terraform. Below is a basic example of Terraform code that provisions an AKS cluster with SSH key access.

**Prerequisites:**

* Terraform installed on your local machine.
* Azure CLI configured and authenticated.
* An SSH key pair available locally **(‘~/.ssh/id\_rsa.pub’** for the public key).

Terraform code example

A screenshot of a computer program

Description automatically generated

If you choose to use Kubernetes (also known as **kubenet**) networking for your AKS cluster instead of Azure CNI, you need to modify the **‘network\_profile’** block in the Terraform configuration.

A black screen with white text

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **Azure CNI (Container Networking Interface)** |  * In Azure CNI, each pod in your AKS cluster gets an IP address from the Azure Virtual Network subnet. This means that pods citizens on a VNET and can communicate directly with other resources on VNET. AVMs and other resources. * Each pod gets an IP from the VNET, you need to maintain the VNET has enough IP addresses available, which is harder to maintain in large clusters. * Azure CNI provides a better performance to communication. Its directly integrated directly into Azure networking Stack. * **Use Case:** Azure CNI is ideal for tight integration with Azur resources, direct access from pods. If you want you can integrate NSG,FIREWALLS and others. | |  | | --- | | **Kubenet** |  * In Kubenet only the nodes (VMs) are assigned IP addresses from the Azure VNET, this means that pod IPs are assigned from a separate internal subnet that Kubernetes manages. * Kubenet is a simple, more traditional Kubernetes networking model. * Communication between pods across nodes is handled by Network Address Translation (NAT). * Kubenet is more efficient in terms of IP address utilization, especially in large clusters where IPs might be a concern. * Kubenet has less performance compared to Azure CNI for certain networking operations. * **Use Case:** kubenet is suitable for scenarios where you don’t need direct integration with Azure Networking, or when preserving IP addresses is a priority. |

**Key Differences:**

* **Pod IP Addresses**:
  + **Azure CNI**: Pods get IP addresses directly from the Azure VNet.
  + **Kubenet**: Pods get IP addresses from a Kubernetes-managed subnet, not directly from the Azure VNet.
* **Networking Features**:
  + **Azure CNI**: Full integration with Azure networking features like NSGs and route tables.
  + **Kubenet**: Limited integration with Azure networking; primarily relies on Kubernetes' networking features.
* **Performance**:
  + **Azure CNI**: Typically, better for pod-to-pod and pod-to-service communication.
  + **Kubenet**: May have lower performance due to the use of NAT.
* **Use Cases**:
  + **Azure CNI**: Better for clusters requiring advanced networking or direct integration with Azure services.
  + **Kubenet**: Better for large clusters where IP address conservation is important.

Disable SSH on an existing cluster & new cluster deployment & for a new node pool.

Re-enable SSH on an existing cluster & SSH for a specific node pool.