**To debug connectivity issues or any kind of issues related to an Azure Kubernetes Service (AKS) cluster, follow these detailed step-by-step instructions:**

**Step 1: Verify AKS Cluster Status**

1. Check the AKS Cluster's Health: Use the Azure CLI to check the health status of your AKS cluster

az aks show --resource-group <ResourceGroupName> --name <AKSClusterName> --query 'provisioningState'

Ensure that the cluster is in a Succeeded or Running state.

1. Check Node Health: Ensure that the cluster nodes are healthy and running.

kubectl get nodes

Look for nodes that are in a Ready state. If a node is in NotReady, you might have an issue with that node.

**Step 2: Check Pod Health and Connectivity**

1. Check All Pods' Status: Run the following command to list all pods across all namespaces and check their status.

kubectl get pods --all-namespaces

Check if any pods are in a Pending, CrashLoopBackOff, or Error state, indicating issues in the pods themselves.

1. Get Detailed Pod Information: If you find a pod in an error state, inspect it for details.

kubectl describe pod <PodName> -n <Namespace>

Look for any event logs or error messages indicating why the pod is failing.

1. Check Pod Logs: Logs are useful for identifying runtime issues or configuration problems.

kubectl logs <PodName> -n <Namespace>

Use the logs to troubleshoot any potential issues related to container startup or crashes.

**Step 3: Verify Network Connectivity**

1.Check Network Policies: If you have network policies enabled in the AKS cluster, ensure they are not blocking connectivity between pods.

kubectl get networkpolicies -n <Namespace>

Review any policies to ensure traffic is allowed as expected.

2.Run DNS Resolution Check: Use the DNS utilities within a pod to check DNS resolution.

kubectl exec -it <PodName> -- nslookup <ServiceName>

Ensure that the internal services are resolvable via DNS.

3.Ping Other Pods/Services: You can test the network connection between pods and services by executing a ping command from one pod to another.

kubectl exec -it <PodName> -- ping <ServiceIP>

This will help identify if there are networking issues between your services or pods.

4.Test Node to Pod Connectivity: If pod-to-pod connectivity is fine, but you suspect node issues, SSH into an AKS node and check connectivity:

az aks ssh --resource-group <ResourceGroupName> --name <AKSClusterName>

Once logged into the node, you can perform standard networking tests like curl, ping, or telnet to pods or external services.

**Step 4: Troubleshoot Load Balancer and Ingress**

1.Verify Ingress Resource: If you're using Ingress for external connectivity, ensure your ingress resource is correctly configured.

kubectl get ingress -n <Namespace>

Then, describe the ingress resource to inspect any events or errors.

kubectl describe ingress <IngressName> -n <Namespace>

2.Check External Load Balancer: If you're exposing services via an external load balancer, verify the status.

kubectl get svc -n <Namespace>

Check for an external IP or load balancer provisioned. If the external IP is pending, there may be an issue with provisioning the load balancer.

3.Check Azure Network Security Groups (NSGs): Ensure that the NSGs associated with your AKS nodes or subnets are not blocking required traffic. You can check this in the Azure Portal or use the Azure CLI:

az network nsg show --resource-group <ResourceGroupName> --name <NSGName>

Ensure ports like 80/443 for HTTP/HTTPS or other custom ports are open.

**Step 5: Verify Kubernetes Service and Configurations**

1.Check Service Configuration: If your service is not reachable, ensure it’s properly configured by describing the service.

kubectl describe svc <ServiceName> -n <Namespace>

Review the Endpoints section to confirm that the service is correctly bound to the pod.

2.Port Forward for Debugging: Temporarily forward a service port to your local machine to check its response.

kubectl port-forward svc/<ServiceName> <LocalPort>:<ServicePort> -n <Namespace>

Test connectivity by accessing localhost:<LocalPort>.

3.Check Service Endpoints: Verify that your services have the correct endpoints:

kubectl get endpoints -n <Namespace>

**Step 6: Analyze Azure Activity Logs**

1.Review Azure Activity Logs: If the issue is related to the underlying AKS infrastructure or any Azure-level actions, review the activity logs.

* Go to the Azure Portal.
* Navigate to the AKS resource group.
* Go to Activity Logs to see any recent failures, configuration changes, or warnings that could be affecting your AKS cluster.

2.Azure Monitor for AKS: Use Azure Monitor to analyze performance and logs from your AKS cluster. It provides metrics, logs, and insights to identify issues.

* Go to Monitor in the Azure Portal.
* Select Kubernetes Services.
* Review metrics for CPU, memory, and network usage.

**Step 7: Diagnose DNS and CoreDNS Issues**

1.Check CoreDNS Logs: CoreDNS is responsible for DNS resolution in AKS. If you're facing DNS issues, check the CoreDNS pod logs

kubectl logs -n kube-system -l k8s-app=kube-dns

Look for any errors related to DNS queries.

2.CoreDNS Configuration: Ensure that the CoreDNS configuration is correct. You can describe the config map to inspect it:

kubectl describe configmap coredns -n kube-system

**Step 8: Restarting Pods/Nodes**

1.Restart Unhealthy Pods: If you've identified a pod that is malfunctioning and can't determine the cause, try restarting it.

kubectl drain <NodeName> --ignore-daemonsets --force

Then cordon the node and restart it from the Azure portal or CLI.

**Step 9: Use Azure Diagnostics Tools**

1.Azure Network Watcher: Use Azure's Network Watcher to diagnose networking issues related to the AKS cluster. Set up the following:

* Connection monitors to check end-to-end connectivity.
* Packet capture on specific VMs to investigate packet-level issues.

kubectl get roles,rolebindings,clusterroles,clusterrolebindings -n <Namespace>

Adjust permissions if required

**Let's go step-by-step with more detailed instructions and include examples to help debug connectivity and other issues in an Azure Kubernetes Service (AKS) cluster.**

**Step 1: Verify AKS Cluster and Node Health**

**1.1 Check the AKS Cluster Health**

You can check the overall health of your AKS cluster using the Azure CLI.

**Command:**

az aks show --resource-group <ResourceGroupName> --name <AKSClusterName> --query 'provisioningState'

**Example:**

az aks show --resource-group MyResourceGroup --name MyAKSCluster --query 'provisioningState'

**Expected Output:**

If the cluster is healthy, the output should be Succeeded. If there are any issues, it may show Failed, Updating, or Degraded, which indicates there are problems with the cluster’s provisioning.

**1.2 Check Node Health**

Next, check if the nodes within your AKS cluster are running correctly.

**Command:**

kubectl get nodes

**Example Output:**

| **NAME** | **STATUS** | **ROLES** | **AGE** | **VERSION** |
| --- | --- | --- | --- | --- |
| aks-nodepool1-12345678-vmss000000 | Ready | agent | 25d | v1.21.2 |
| aks-nodepool1-12345678-vmss000000 | Ready | agent | 25d | v1.21.2 |

The STATUS column should show Ready. If it's NotReady, further investigation is required. Nodes that are NotReady may indicate a networking or resource allocation issue.

**Step 2: Check Pod Health and Logs**

**2.1: Get the Status of All Pods**

Check the status of all pods to see if any are having issues.

**Command:**

kubectl get pods --all-namespaces

**Example Output:**

sql

| **NAMESPACE** | **NAME** | **STATUS** | **ROLES** | **RESTARTS** | **AGE** |
| --- | --- | --- | --- | --- | --- |
| kube-system | coredns-7f9c69d78c-5lx9r | 1/1 | Running | 0 | 25d |
| default | nginx-deployment-5d898d5c89-fbtcx | 0/1 | CrashLoopBackOff | 3 | 5m |
| default | myapp-abc123 | 1/1 | Running | 0 | 10h |

The STATUS column will indicate if any pod is stuck in states like Pending, CrashLoopBackOff, or Error.

**2.2: Describe the Problematic Pod**

If you notice any pods in CrashLoopBackOff or another error state, describe the pod to get more details.

**Command:**

kubectl describe pod <PodName> -n <Namespace>

**Example:**

kubectl describe pod nginx-deployment-5d898d5c89-fbtcx -n default

**Key Information to Look for:**

* Events at the bottom of the output can indicate if the pod failed due to insufficient memory, failed image pull, or other issues.
* Look for issues like:

-ImagePullBackOff: Indicates the container image cannot be pulled from the registry.  
-OOMKilled: Indicates the pod was killed due to an out-of-memory issue.

**2.3: Check Logs for the Pod**

Logs help diagnose runtime issues within a container.

**Command:**

kubectl logs <PodName> -n <Namespace>

**Example:**

kubectl logs nginx-deployment-5d898d5c89-fbtcx -n default

Check for errors in the output, such as misconfigurations in the container or service issues.

**Step 3: Verify Network Connectivity**

**3.1: Check DNS Resolution**

Within a pod, test whether it can resolve services via DNS.

**Command:**

kubectl exec -it <PodName> -- nslookup <ServiceName>

**Example:**

kubectl exec -it myapp-abc123 -- nslookup nginx-service`

**Output:**

yaml

Server: 10.0.0.10  
Address: 10.0.0.10#53

**Non-authoritative answer:**

Name: nginx-service.default.svc.cluster.local  
Address: 10.0.149.1

This confirms that the service is resolvable using DNS. If it fails, there may be a CoreDNS issue or misconfigured service.

**3.2: Ping a Pod or Service**

You can also test network connectivity between pods.

**Command:**

kubectl exec -it <PodName> -- ping <ServiceIP>

**Example:**

kubectl exec -it myapp-abc123 -- ping 10.0.149.1

If the ping fails, there could be a network policy or firewall blocking traffic between pods.

**Step 4: Troubleshoot Load Balancer and Ingress**

**4.1: Verify Ingress Configuration**

If you are using an Ingress resource for external access, describe the ingress to see if it’s properly configured.

**Command:**

kubectl describe ingress <IngressName> -n <Namespace>

**Example:**

kubectl describe ingress nginx-ingress -n default

**Example Output:**

vbnet

Name: nginx-ingress  
Namespace: default  
Address: 20.51.10.21

**Rules:**

| **Host** | **Path Backends** |
| --- | --- |
| [myapp.com](http://myapp.com/) | nginx-service:80 (10.0.149.1:80) |

* Check if the Address is populated with an external IP. If it’s not, the Ingress might not have a valid load balancer assigned.

**4.2: Check Service and External Load Balancer**

To verify if your service is properly exposed with a load balancer, check the service status.

**Command:**

kubectl get svc -n <Namespace>

**Example Output:**

scss

| **NAME** | **TYPE** | **CLUSTER-IP** | **EXTERNAL-IP** | **PORT(S)** | **AGE** |
| --- | --- | --- | --- | --- | --- |
| nginx-service | LoadBalancer | 10.0.149.1 | 20.51.10.21 | 80:30001/TCP | 10m |

* Ensure the EXTERNAL-IP field is assigned. If it says pending, it means the load balancer hasn’t been provisioned correctly.

**4.3: Check Azure NSGs**

If services or pods cannot be accessed externally, ensure that Azure Network Security Groups (NSGs) are not blocking traffic.

You can check NSGs associated with the cluster via the Azure CLI or Portal.

**Command:**

az network nsg show --resource-group <ResourceGroupName> --name <NSGName>

**Example:**

az network nsg show --resource-group MyResourceGroup --name MyNSG

Make sure that rules allow inbound traffic on ports like 80, 443, or custom ports used by your services.

**Step 5: Diagnose CoreDNS Issues**

**5.1: Check CoreDNS Pod Logs**

If you’re facing DNS resolution issues, check the logs for the CoreDNS pods running in the AKS cluster.

**Command:**

kubectl logs -n kube-system -l k8s-app=kube-dns

**Example Output:**

csharp

[INFO] 10.244.1.1:60107 - 46275 "A IN nginx-service.default.svc.cluster.local. udp 65 false 512" NOERROR qr,aa,rd 111 0.000447657s

Look for errors related to DNS queries failing.

**5.2: Check CoreDNS Config**

CoreDNS uses a config map to manage DNS configuration. You can inspect the CoreDNS config map as follows.

**Command:**

kubectl describe configmap coredns -n kube-system

Check for any misconfigurations in DNS settings.

**Step 6: Use Azure Diagnostics**

**6.1: Review Activity Logs**

If your AKS cluster or associated services are experiencing issues, review the Azure Activity Logs for any failed operations or warnings.

1. Go to the Azure Portal.
2. Navigate to the Resource Group where your AKS cluster is deployed.
3. Go to Activity Logs to review recent activity like node failures, cluster scaling issues, etc.

**6.2: Use Azure Monitor for AKS**

Azure Monitor can give you a deep dive into the AKS cluster’s health, including CPU, memory, and network performance.

1. Go to Monitor in the Azure Portal.
2. Select Kubernetes Services.
3. You can view metrics for node performance, pod health, and network traffic.