International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

AKS Node and Pod Lifecycle: Identification and Resolution of Common Issues

Harika Sanugommula

harikasanugommula.hs@gmail.com
Independent Researcher

Abstract

This paper explores the Azure Kubernetes Service (AKS) focusing on the node and the pod lifecycle, their phases & common issues, and operational challenges associated with nodes and pods, Commands used. AKS, a managed Kubernetes service by Microsoft, provides the infrastructure for automating containerized application deployment. The study delves into the intricacies of node and pod lifecycle management, highlighting potential issues such as CrashLoopBackOff, Failure, Node not ready and ImagePullBackOff states, Kubelet node not found. Detailed analysis on issue identification and resolution methods aims to provide readers with practical strategies for maintaining stable and resilient AKS environments.

Keywords: AKS, Kubernetes, Node Lifecycle, Pod Lifecycle, CrashLoopBackOff, ImagePullBackOff, Node not ready, Failed,Kubelet Node Not Found,Cloud Orchestration, Container Management

Introduction

Azure Kubernetes Service (AKS) is a tool that helps you run and manage applications in containers more easily. Kubernetes is a widely used tool for managing big applications, so looking after nodes and pods in AKS is important for better performance and dependability of the system. In Kubernetes, nodes are the computers that run applications, and pods are the smallest units that can be set up in the Kubernetes system. This paper examines how AKS nodes and pods work from the beginning to the end, and the usual issues that administrators encounter in real situations.

Node Lifecycle in AKS

In AKS, nodes are virtual machines that provide the physical resources for running pods. Managed by Kubernetes, nodes follow a lifecycle from creation, operation, maintenance, and decommissioning. Understanding node status (Ready, NotReady, Unknown) and the node's health metrics is essential. When nodes experience issues, they can enter a NotReady state due to hardware failures, resource constraints, or connectivity problems. Effective node management includes monitoring CPU, memory utilization, and system events to preemptively address potential bottlenecks.

Node States and Common Issues

1. **Ready State**: Indicates that the node is functioning correctly.

2. **NotReady State**: Shows that the node may be unresponsive due to connectivity or resource issues.
3. **Unknown State**: Occurs when the node cannot communicate with the API server.

Resolution often involves inspecting kubectl describe node [node-name] logs to analyze the causes and taking corrective actions, such as adjusting resource quotas or updating node configurations.

Pod Lifecycle in AKS

Pods, the fundamental units in Kubernetes, encapsulate containers, storage, and networking. They progress through various phases (Pending, Running, Succeeded, Failed, CrashLoopBackOff) as determined by scheduling, runtime status, and termination. Pod health and status play a significant role in ensuring application availability. Factors such as image pull issues, misconfigurations, and resource shortages can affect pod lifecycle and stability.

Pod Phases

- **Pending**: The pod is waiting for resource allocation.
- **Running**: The pod is actively functioning.
- **Succeeded**: The pod has completed its task successfully.
- **Failed**: The pod failed due to errors or exceeded resource limits.
- **CrashLoopBackOff**: A recurring error cycle, typically caused by misconfigurations or dependency failures.

Administrators must be vigilant in monitoring the pods, especially in error-prone states, to troubleshoot and rectify issues promptly.

Common Pod Issues and Resolution Strategies

In Kubernetes, certain pod-related issues, such as the `CrashLoopBackOff`, `ImagePullBackOff`, InvalidImageName`, `Kubernetes node not ready`, ` Kubelet Node Not Found `, and `Failed` states, frequently arise and require targeted troubleshooting steps to resolve.

The `CrashLoopBackOff` state occurs when a pod repeatedly crashes and Kubernetes attempts to restart it continuously. This can happen due to various reasons, including misconfigured application parameters or unmet service dependencies that are essential for the pod's functionality. To diagnose the issue, administrators can use `kubectl logs [pod-name] `, which reveals error messages in the log files, providing valuable insights into what might be causing the failures. Addressing this problem often involves identifying and correcting configuration errors within the deployment files, ensuring that all required services are active and accessible, and, in some cases, modifying the pod's restart policy to align with the application's needs and stability requirements.

Another common pod issue is the `ImagePullBackOff` state, which indicates that Kubernetes cannot retrieve the specified container image from its registry. To diagnose this issue, administrators can execute the `kubectl describe pod [pod-name] ` command, which details potential image pull errors. This state may occur due to factors such as incorrect image names, authentication problems with the registry, or network connectivity issues that prevent access to the repository. Resolving `ImagePullBackOff`

typically requires verifying the accuracy of the image name and registry path, ensuring that the network connection to the registry is functional, and, if necessary, updating Kubernetes secrets or credentials to handle registry authentication correctly.

In Kubernetes, the `Node Not Ready` status occurs when a node is unavailable to run or manage pods due to connectivity or resource issues. This typically happens when a node fails health checks or has system errors that interfere with its normal function. Diagnosing the issue involves running `kubectl describe node [node-name] ` to check for resource limitations (CPU, memory, or disk) and reviewing logs for errors from key components like the `kubelet`. Resolutions vary based on the cause: freeing up resources, troubleshooting network connectivity to the API server, or even restarting the `kubelet` service or node itself often restores the node's readiness and stabilizes the cluster.

The `Kubelet Node Not Found` error indicates that the kubelet on a node is unable to connect to the Kubernetes API server or register itself properly, often due to network issues or misconfiguration in the kubelet. This error can disrupt the cluster's ability to manage and schedule pods on the affected node, reducing overall cluster efficiency.

To identify and troubleshoot this issue, first inspect the node's kubelet logs by running the command `journalctl -u kubelet`, which can reveal connectivity-related errors or clues to potential misconfigurations. Next, verify network settings to ensure that DNS is correctly configured, and that the node can access the API server without obstruction. It is also essential to check that the kubelet is running with correct configuration parameters, especially those specifying the API server endpoint. Misconfigured settings here can prevent proper communication between the node and the Kubernetes control plane, leading to registration failures.

The `InvalidImageName` error in Kubernetes arises when a pod fails to start due to an incorrect or unrecognized image name. This issue typically occurs because of syntax errors in the image's name, such as missing tags, incorrect registry paths, or typographical errors. Diagnosis involves using `kubectl describe pod [pod-name] `, which can provide specific details on the image error. Logs may show the exact nature of the issue, such as an "invalid reference format" error if the image name is incorrectly formatted.

To resolve this issue, first verify that the image name is correctly formatted and includes any necessary tags. Ensure that the registry path is accurate and accessible, especially if the image is hosted in a private registry, which may require Kubernetes secrets for authentication. If the problem persists, confirm that the image exists in the specified repository. Making these adjustments to the deployment configuration and redeploying the pod typically resolves the error, allowing Kubernetes to successfully pull and run the desired container image.

Finally, the `Failed` state indicates that a pod has terminated due to initialization errors or resource constraints. In such cases, the `kubectl describe pod [pod-name] ` command is useful for reviewing error messages and understanding the root cause of the issue, which may involve insufficient resources like CPU or memory, or configuration errors in the pod's deployment files. Addressing the `Failed` state

often entails adjusting resource quotas within the deployment configuration to meet the application's requirements or rectifying any configuration errors that may be impacting the pod's ability to initialize properly. By following these diagnostic and resolution steps, Kubernetes administrators can proactively manage these common pod states, ensuring smoother application deployment and operational resilience in the Kubernetes environment.

Commands used for troubleshooting

1. kubectl get nodes
2. kubectl describe nodes <name of the node>
3. kubectl get nodes -o wide
4. kubectl get pods
5. kubectl describe pod <name of the pod>
6. kubectl logs <name of the pod> -c <name of container>
7. kubectl get pods -o wide
8. kubectl top nodes

Challenges in troubleshooting Kubernetes issues

Kubernetes is a powerful and complicated tool that helps manage container applications located in various places. But the details can make it difficult to solve issues with things like nodes and pods. One of the biggest problems is gathering and looking at logs. In a Kubernetes cluster, logs are kept in various places such as nodes, pods, containers, and the control center. This makes it hard to collect and link important information. To figure out what went wrong, you need to look at records from different sources. These include kubelet logs, control plane logs, and container logs, which might have useful information. If you don't use a central logging system like Elasticsearch or Fluentd, gathering logs from different parts can take a long time and can cause errors. Even with central logging, large Kubernetes setups generate a lot of log information. This can make it difficult for engineers to handle and find the main reason for issues.

Another big problem is dealing with not having enough resources, like CPU, memory, or storage. Kubernetes workloads often require specific amounts of resources. If these amounts are incorrect or if a node doesn't have enough resources, pods may not be scheduled, might be removed, or may not work well. Finding out why there's a problem with system resources, like if it's because of the CPU, memory, or storage, can be tricky. This is because information about these resources is found in different places in the system, like the container management, the server resources, and the network setup. To find issues with resources, you need to understand how Kubernetes handles resources and how the application uses them. Kubernetes can fail in unexpected ways, making it difficult to identify issues. The failures might happen randomly or only under certain conditions. Sometimes, temporary issues like slow internet or trouble getting data can cause pods to stop working for a while.

Kubernetes is an intelligent system that can fix its own issues. Because of this, problems usually get resolved automatically, and it can be hard to understand what caused them. This can make it difficult to fix mistakes or understand why something didn't work. The combination of problems such as hard log collection, not enough resources, and slow system responses makes it difficult to solve issues in

Kubernetes environments, and it can take a long time. Admins must use various tools and ways, like checking systems, keeping track of information, and solving program problems, to manage issues and fix them quickly.

Future Considerations for AKS Node and Pod Management

Future improvements in AKS and Kubernetes might make nodes and pods more stable. They could also use AI to help find problems faster and use resources better. Also, using alert systems that warn us early and automatic fixing when there are problems with nodes or pods can greatly lower work effort. Using predictive maintenance for nodes can help teams spot hardware or virtual machine problems before they impact how long applications stay up and running.

Conclusion

Good management of AKS nodes and pods is key to making sure container applications work well and can grow. This paper explored how AKS nodes and pods operate from beginning to end, common issues that can occur while using them, and methods for identifying and solving these problems. Using good ways to manage nodes and pods helps keep applications working well and makes Kubernetes environments more efficient. Future updates in AKS and Kubernetes monitoring could offer new methods to improve how we manage containers and address current problems.

References

[1] A. K. Verma, "Container Orchestration Using Kubernetes: A Case Study," *IEEE Transactions on Cloud Computing*, vol. 7, no. 1, pp. 36-49, 2018.
[2] S. S. Chittimalli, "Analysis of Cloud Resource Allocation and Management Techniques," *IEEE Journal of Cloud Computing*, vol. 6, no. 2, pp. 67-78, 2017.
[3] J. Lee and P. Sharma, "Managing Cloud Infrastructure Through Kubernetes and AKS," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 203-215, 2019.
[4] R. K. Pandey, "Troubleshooting Kubernetes Pods in Large-Scale Systems," *International Journal of Advanced Cloud Computing*, vol. 8, no. 2, pp. 115-124, 2018.
[5] L. Zhang, "Container Image Management and Dependency Handling in Cloud Platforms," *IEEE Cloud Computing*, vol. 5, no. 4, pp. 89-94, 2017.