Kubernetes Architecture with EKS (Elastic Kubernetes Service)

Kubernetes (K8s) is a popular orchestration tool used to manage multiple nodes, enabling scalability by deploying pods.

Pods are abstractions of containers. When a node is deployed, it interacts with the master server, which decides the scheduling of pods. Here's how the process works:

- The **API server** informs the scheduler.
- The scheduler deploys the pod into the node.
- The **controller manager** ensures the desired number of pods are maintained.

To manage a Kubernetes cluster, we use:

- **kubectl**: A command-line tool to interact with the cluster.
- **eksctl**: A command-line tool for installing and managing EKS clusters.

Kubernetes Components

1. **Control Plane (Master Node)**
    - Decides and executes cluster actions.
2. **Data Plane (Worker Nodes)**
    - Executes actions like running containers.

Kubernetes is self-managed and open-source. However, managing and maintaining Kubernetes can be challenging as it involves deciding the number of master and worker nodes, updating Kubernetes versions, and troubleshooting issues.

Cloud providers such as AWS, GCP, and Azure offer **managed Kubernetes services** like EKS. These services reduce the overhead of maintenance and allow users to pay only for the resources used.

AWS EKS simplifies deployment and management through managed clusters.

- Resources like VPCs, subnets, security groups, and auto-scaling groups are created automatically.
- AWS also provides `kubectl` to interact with the cluster.

Steps to Create an EKS Cluster

1. **Install Prerequisites**

Install `eksctl` and `kubectl` on your system.
Copy code

```
eksctl version

kubectl version
```

2. **AWS Configuration**

Configure AWS CLI with your credentials:

```
aws configure
```

   - Provide your **Access Key ID** and **Secret Access Key**.

Verify credentials with:

```
aws sts get-caller-identity
```

   -
3. **Create a Cluster**

Use the following command to create an EKS cluster:

```
eksctl create cluster --name <cluster-name> --nodes <number-of-nodes>
--node-type <instance-type> --managed
```

   -

Example:

```
eksctl create cluster --name DevOpsCluster --nodes 3 --node-type
t3.medium --managed
```

   -

Check if the cluster is created:

```
eksctl get cluster
```

   -
4. **Verify in AWS Console**
   - Confirm resources like clusters, VPCs, security groups, and auto-scaling groups are created.

Managing Pods

Create a Pod

```
kubectl run <pod-name> --image=amazonlinux
```

-

Check Pods

```
kubectl get pods
```

-
- **Troubleshoot Pods**
    - If a pod crashes (e.g., **CrashLoopBackOff** error), follow these steps to diagnose:
        - **CrashLoopBackOff** occurs when a container repeatedly fails to start and stops unexpectedly. Common reasons include incorrect configurations, missing dependencies, or application errors.

Check pod details:

```
kubectl describe pod <pod-name>
```

        -

Review pod logs:

```
kubectl logs <pod-name>
```

        -
        - Example Errors:
            1. **ImagePullBackOff**: Failed to pull the image. Verify the image name and accessibility.
            2. **CrashLoopBackOff**: Look for missing environment variables, application startup errors, or resource limits causing the failure.

Writing YAML Manifests

To manage Kubernetes resources effectively, create YAML manifests. Use **VS Code** with the YAML extension for ease of writing and suggestions.

Example of a Pod manifest:

yaml

```
apiVersion: v1

kind: Pod

metadata:

  name: my-pod

spec:
```

```
  containers:
  - name: my-container
    image: amazonlinux
```

Save the file and apply it using:

```
kubectl apply -f <file-name>.yaml
```

Summary

By leveraging AWS EKS and its managed services, we can focus more on application deployment and less on cluster maintenance. Troubleshooting tools like `kubectl describe` and `kubectl logs` make debugging issues like **CrashLoopBackOff** easier. Managed services from cloud providers reduce complexity and allow developers to concentrate on scaling and delivering applications.