d 
POd 
Z BPON 
I apON 
」 STID 

The **control plane** is the central management layer responsible for maintaining the desired state of the cluster. It manages and orchestrates all cluster activities, including scheduling, scaling, and maintaining the lifecycle of workloads.

The control plane components include:

* **API Server**: The communication hub for cluster management.
* **etcd**: The key-value store for cluster state.
* **Scheduler**: Allocates pods to appropriate nodes.
* **Controller Manager**: Ensures desired cluster state by managing controllers (e.g., nodes, deployments).
* **Cloud Controller Manager** (optional): Integrates with cloud-specific APIs for resources like load balancers.

 02/12/2024   10:21.07   /home/mobaxterm  kubectl get componentstatus

Warning: v1 ComponentStatus is deprecated in v1.19+

NAME STATUS MESSAGE ERROR

scheduler Healthy ok

controller-manager Healthy ok

etcd-0 Healthy ok

 02/12/2024   10:46.14   /home/mobaxterm  kubectl describe componentstatus scheduler

Warning: v1 ComponentStatus is deprecated in v1.19+

Name: scheduler

Namespace:

Labels: <none>

Annotations: <none>

API Version: v1

Conditions:

Message: ok

Status: True

Type: Healthy

Kind: ComponentStatus

Metadata:

Creation Timestamp: <nil>

Events: <none>

✓

 02/12/2024   10:59.49   /home/mobaxterm 

✓

 02/12/2024   11:08.15   /home/mobaxterm  kubectl describe componentstatus scheduler

Warning: v1 ComponentStatus is deprecated in v1.19+

Name: scheduler

Namespace:

Labels: <none>

Annotations: <none>

API Version: v1

Conditions:

Message: ok

Status: True

Type: Healthy

Kind: ComponentStatus

Metadata:

Creation Timestamp: <nil>

Events: <none>

✓

 02/12/2024   11:08.30   /home/mobaxterm  kubectl describe componentstatus controller-manager

Warning: v1 ComponentStatus is deprecated in v1.19+

Name: controller-manager

Namespace:

Labels: <none>

Annotations: <none>

API Version: v1

Conditions:

Message: ok

Status: True

Type: Healthy

Kind: ComponentStatus

Metadata:

Creation Timestamp: <nil>

Events: <none>

✓

 02/12/2024   11:08.58   /home/mobaxterm  kubectl describe componentstatus etcd-0

Warning: v1 ComponentStatus is deprecated in v1.19+

Name: etcd-0

Namespace:

Labels: <none>

Annotations: <none>

API Version: v1

Conditions:

Message: ok

Status: True

Type: Healthy

Kind: ComponentStatus

Metadata:

Creation Timestamp: <nil>

Events: <none>

Here are short explanations of the key **control plane components** in Kubernetes and how to check their status:

1. **API Server**:  
   Manages communication between Kubernetes components and is the entry point for all administrative tasks.
   * **Check status**: kubectl get componentstatus or inspect logs using kubectl logs -n kube-system <apiserver-pod>.
2. **etcd**:   
   A distributed key-value store that stores cluster state and configuration data.
   * **Check status**: Use kubectl exec -n kube-system <etcd-pod> -- etcdctl endpoint health.
3. **Scheduler**:  
   Assigns pods to nodes based on resource availability and constraints.
   * **Check status**: Review pod logs using kubectl logs -n kube-system <scheduler-pod>.
4. **Controller Manager**:  
   Manages core control loops (e.g., node, replication controllers) to maintain desired cluster state.
   * **Check status**: Inspect logs using kubectl logs -n kube-system <controller-manager-pod>.
5. **Cloud Controller Manager** (if cloud-provider integration is enabled):  
   Manages cloud-specific logic like load balancers and storage provisioning.
   * **Check status**: Use kubectl logs -n kube-system <cloud-controller-manager-pod>.

 02/12/2024   10:00.04   /home/mobaxterm  kubectl cluster-info

Kubernetes control plane is running at https://\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*.eu-west-1.eks.amazonaws.com

CoreDNS is running at https://\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*.eu-west-1.eks.amazonaws.com/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

wale-k8s-dashboard-kubernetes-dashboard is running at https://\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*eu-west-1.eks.amazonaws.com/api/v1/namespaces/kube-system/services/https:wale-k8s-dashboard-kubernetes-dashbo

ard:https/proxy

**1. Inspect Cluster Control Plane Using kubectl**

Control plane components like the API server, etcd, scheduler, and controller manager are managed as static pods or services in Kubernetes. To check their status:

**View Control Plane Pods**

1. Run the following to list pods in the kube-system namespace:

kubectl get pods -n kube-system

Look for pods like:

* + kube-apiserver
  + kube-scheduler
  + kube-controller-manager
  + etcd

1. Check details of any specific pod:

kubectl describe pod <pod-name> -n kube-system

1. Review logs for specific pods, for example:

bash

Copia codice

kubectl logs <pod-name> -n kube-system

**2. Access the API Server**

The API server is the core of the control plane. To verify it:

1. **Confirm the API Server Endpoint**:

kubectl cluster-info

Example output:

arduino

Kubernetes control plane is running at https://<api-server-endpoint>

1. **Check API Server Health**:
   * You can directly access the health endpoint:

curl -k https://<api-server-endpoint>/healthz

Expected output for a healthy API server:

Copia codice

ok

**3. Inspect Cluster Events**

Cluster events can indicate control plane activity:

kubectl get events --all-namespaces

Look for warnings or issues related to control plane components.

**4. Using Cloud-Specific Tools**

If your cluster is managed by a cloud provider (like EKS, AKS, GKE), you can inspect the control plane through the cloud console or CLI.

**Example:**

* **AWS EKS**: Use the AWS Management Console or the CLI to check the status of the control plane:

aws eks describe-cluster --name <cluster-name>

* **GKE**: Use the Google Cloud Console under Kubernetes Engine > Clusters or the following CLI command:

gcloud container clusters describe <cluster-name>

**5. Check Etcd (Cluster State Storage)**

Etcd is a critical part of the control plane. To inspect its health:

1. Access etcd pod logs:

kubectl logs <etcd-pod-name> -n kube-system

1. If running locally, use etcdctl to check cluster health:

etcdctl --endpoints=<etcd-endpoint> endpoint health

**6. Debug Issues with Control Plane**

For deeper debugging:

* **Check Control Plane Node Health**:

kubectl get nodes

Verify that all nodes, especially the master/control plane nodes, are in a healthy state.

* **Review System Logs** (if running locally or on bare metal):
  + Check logs under /var/log/ for kubelet or static pod errors.
  + Use journalctl for systemd-based logs.

Other essential components include **NODES**, which work together with the control plane to run and manage workloads. Here's a breakdown:

**Nodes (Worker Nodes)**

Nodes are the physical or virtual machines where application workloads are deployed. Each node contains these key components:

1. **Kubelet**: An agent that communicates with the control plane and ensures containers are running as expected.
2. **Kube-proxy**: A networking component responsible for forwarding requests to the appropriate pods and managing network rules.
3. **Container Runtime**: The software responsible for running containers (e.g., Docker, containerd).

**Pods**

Pods are the smallest deployable units in Kubernetes, consisting of one or more containers that share storage, network, and a specification for how to run.

**Cluster Add-ons**

Additional components that extend the functionality of a Kubernetes cluster:

* **DNS**: A cluster-wide DNS service for service discovery.
* **Ingress Controller**: Manages external HTTP(S) traffic routing into the cluster.
* **Metrics Server**: Provides resource metrics (e.g., CPU and memory usage) for auto-scaling.

**Storage and Networking**

* **Persistent Volumes (PVs) and Persistent Volume Claims (PVCs)**: Allow stateful applications to store data persistently.
* **Overlay Networks**: Facilitate communication between pods across nodes (e.g., using CNI plugins like Calico or Flannel).

Each of these components works in concert to provide Kubernetes its robust, scalable, and self-healing architecture.