Kubernetes Documentation

Kubernetes is a powerful open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). Here are the key aspects of Kubernetes:

Key Concepts:

1. **Containerization:**
    - Kubernetes is built around the concept of containers, which are lightweight, portable, and consistent environments for running applications and their dependencies.
2. **Nodes:**
    - A Kubernetes cluster consists of nodes. Nodes can be physical or virtual machines and are divided into two types: master nodes and worker nodes.
3. **Master Node (Control Plane):**
    - The master node is responsible for managing the overall state of the cluster.
    - Components on the master node include the API server, etcd, controller manager, and scheduler.
4. **Worker Node:**
    - Worker nodes, or minions, host the containers that run the actual applications.
    - Each worker node has the Kubelet, Kube Proxy, and a container runtime (e.g., Docker) installed.
5. **Pod:**
    - The smallest deployable unit in Kubernetes.
    - A pod represents a single instance of a running process in a cluster and encapsulates one or more containers.

6. **Controller:**
    - Controllers are responsible for maintaining the desired state of the cluster.
    - Examples include ReplicaSet (ensures a specified number of replicas of a pod are running) and Deployment (manages rolling updates and rollbacks).
7. **Service:**
    - An abstraction that defines a logical set of pods and a policy by which to access them.
    - Enables load balancing and service discovery within the cluster.
8. **Namespace:**
    - Provides a way to organize and scope resources within a cluster.
    - Allows multiple virtual clusters to coexist on the same physical cluster.

Architecture:

- **API Server:**
    - Exposes the Kubernetes API and serves as the front end for the Kubernetes control plane.
- **etcd:**
    - Consistent and highly-available key-value store used for storing all cluster data.
- **Controller Manager:**
    - Manages controllers that regulate the state of the system.
- **Scheduler:**
    - Assigns pods to nodes based on resource availability and constraints.
- **Kubelet:**
    - Ensures that containers are running in a Pod on a node.
- **Kube Proxy:**
    - Maintains network rules on nodes.

- **Container Runtime:**
    - Software responsible for running containers (e.g., Docker).

Workflow:

1. **Desired State Declaration:**
    - Users declare the desired state of their applications using YAML or JSON manifests.
2. **API Server:**
    - The manifests are submitted to the Kubernetes API server.
3. **etcd:**
    - The API server stores the configuration in etcd.
4. **Controller Manager and Scheduler:**
    - Controllers (e.g., ReplicaSet) and the scheduler work together to maintain the desired state.
5. **Kubelet and Container Runtime:**
    - Kubelet on worker nodes communicates with the container runtime to start or stop containers.
6. **Kube Proxy:**
    - Manages network rules to enable communication between pods and external traffic.

Benefits:

- **Scalability:**
    - Kubernetes scales easily, allowing you to deploy and manage applications across clusters of machines.
- **Fault Tolerance:**
    - High availability is achieved by distributing workloads across multiple nodes.
- **Portability:**

    - Kubernetes abstracts away the underlying infrastructure, making it easier to move applications between environments.

- **Declarative Configuration:**
    - Describes the desired state of the system, and Kubernetes takes care of making it a reality.
- **Self-healing:**
    - Kubernetes automatically replaces failed containers or reschedules them to healthy nodes.

Ecosystem:

Kubernetes has a vibrant ecosystem with a rich set of tools and projects, including Helm (package manager), Istio (service mesh), and Prometheus (monitoring).

In summary, Kubernetes simplifies the deployment and management of containerized applications, providing a robust and scalable platform for modern, cloud-native development.

In detail Master Node Components and Processes:

1. **API Server:**
    - The central management entity that exposes the Kubernetes API.
    - Responsible for validating and processing API requests.
2. **etcd:**
    - A distributed key-value store that stores configuration data and state of the entire cluster.
    - Ensures high availability and consistency.
3. **Controller Manager:**
    - Manages controllers that regulate the state of the cluster.
    - Examples include ReplicationController (ensures a specified number of pod replicas) and NodeController (monitors node status).
4. **Scheduler:**
    - Assigns pods to nodes based on resource availability and constraints.

    - Takes into account factors like affinity, anti-affinity, and resource requirements.

5. **Cloud Controller Manager (optional):**
    - Interacts with the underlying cloud provider's API.
    - Manages external cloud resources like load balancers and volumes.

Example:

```
# View pods in the default namespace
kubectl get pods

# Scale a deployment
kubectl scale deployment <deployment-name> --replicas=3
```

In detail Worker Node Components and Processes:

1. **Kubelet:**
    - An agent that runs on each worker node.
    - Ensures containers are running in a Pod and reports node status to the master.

2. **Kube Proxy:**
    - Maintains network rules on nodes.
    - Enables communication between pods on different nodes and external traffic.

3. **Container Runtime:**
    - Software responsible for running containers (e.g., Docker, containerd).
    - Pulls container images, creates containers, and manages their lifecycle.

Example:

```
# View nodes in the cluster
kubectl get nodes

# Describe a node
kubectl describe node <node-name>
```

High-Level Example Workflow:

1. A user deploys a Kubernetes manifest specifying desired pod configurations.
2. The manifest is submitted to the API server.

3. The API server stores the configuration in etcd.
4. The Controller Manager detects the desired state and instructs the Kubelet on worker nodes to start or stop containers.
5. Kubelet communicates with the container runtime to execute the desired state.
6. Kube Proxy manages network rules, enabling communication between pods.

Different Kubernetes important resources, Deployment, Service, Secrets, ConfigMap, and Ingress—with example YAML configurations.

1. Deployment:

A Deployment in Kubernetes describes a desired state for a set of identical pods. It allows you to declaratively manage applications, including their replicas, updates, and rollbacks.

Example Deployment YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
      - name: example-container
        image: nginx:latest
```

- `replicas`: Number of desired replicas.
- `selector`: Label selector to match pods controlled by this deployment.
- `template`: Pod template with its own metadata and spec.

2. Service:

A Service in Kubernetes exposes a set of pods as a network service. It provides a stable endpoint to access the application.

Example Service YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  selector:
    app: example
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

- `selector`: Label selector to define which pods the service targets.
- `ports`: Port configuration to access the service.

3. Secrets:

Secrets in Kubernetes are used to store sensitive information, such as passwords or API keys. They can be mounted into pods as volumes or used as environment variables.

Example Secret YAML:

```
apiVersion: v1
kind: Secret
metadata:
  name: example-secret
type: Opaque
data:
  username: YWRtaW4=  # base64-encoded 'admin'
  password: cGFzc3dvcmQ=  # base64-encoded 'password'
```

- `type`: Opaque indicates a generic secret.
- `data`: Base64-encoded key-value pairs.

4. ConfigMap:

ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.

Example ConfigMap YAML:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-configmap
data:
  app.config: |
    key1: value1
    key2: value2
```

- `data`: Key-value pairs representing the configuration.

5. Ingress:

An Ingress in Kubernetes exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. It allows external access to services.

Example Ingress YAML:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /app
        pathType: Prefix
        backend:
          service:
            name: example-service
            port:
              number: 80
```

- `rules`: Define host-based routing.
- `http.paths`: Specify path-based routing to services.

Service Types

In Kubernetes, Services are used to expose applications running in a set of Pods as a network service. There are different types of Services, each with its own use case. Here are some common Service types:

1. ClusterIP:

A ClusterIP Service exposes the Service on an internal IP address within the cluster. This type of Service is only reachable from within the cluster.

Example YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: example-clusterip-service
spec:
  selector:
    app: example
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

2. NodePort:

A NodePort Service exposes the Service on each Node's IP at a static port. It makes the Service accessible externally by connecting to any Node's IP on the specified NodePort.

Example YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: example-nodeport-service
spec:
  selector:
    app: example
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: NodePort
```

3. LoadBalancer:

A LoadBalancer Service automatically provisions an external load balancer in cloud environments (e.g., AWS, GCP) and assigns a public IP to the Service. It is useful for exposing services to the internet.

Example YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: example-loadbalancer-service
spec:
  selector:
    app: example
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

4. ExternalName:

An ExternalName Service maps a Service to a DNS name. It is used for accessing external services by name.

Example YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: example-externalname-service
spec:
  type: ExternalName
  externalName: example.com
```

5. Headless Service:

A Headless Service is used when you don't need load balancing or a single IP. It provides DNS resolution for the set of Pods but doesn't allocate a cluster IP.

Example YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: example-headless-service
spec:
  clusterIP: None
  selector:
    app: example
  ports:
```

```
    - protocol: TCP
      port: 80
      targetPort: 8080
```

6. Ingress:

While not a Service type per se, an Ingress is often used to expose HTTP and HTTPS routes to services within the cluster. It provides more advanced routing capabilities compared to basic Services.

Example Ingress YAML:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /app
        pathType: Prefix
        backend:
          service:
            name: example-service
            port:
              number: 80
```

Choose the Service type that best fits your application's requirements and the desired level of exposure. Always consider security implications when exposing services externally.

Setup K8 Cluster

Setting up Kubernetes involves several steps, and the process can vary depending on your platform and requirements. Below are the general steps for setting up a basic Kubernetes cluster. Please note that these instructions are for educational purposes, and in a production environment, you may want to use a managed Kubernetes service or follow more advanced configurations based on your needs.

Prerequisites:

1. **Linux Environment:**
    - Ensure you have a Linux-based system. Popular choices include Ubuntu, CentOS, or a specialized Kubernetes distribution like Rancher.
2. **Docker:**
    - Install Docker to run containerized applications. Follow the official Docker installation guide for your OS: Install Docker.
3. **Kubernetes Tools:**
    - Install `kubectl` (Kubernetes command-line tool) and `kubeadm` (Kubernetes cluster management tool):

      ```
      sudo apt-get update && sudo apt-get install -y kubectl kubeadm
      ```

Setting Up a Single-Node Cluster (for Development):

```
# Initialize the cluster
sudo kubeadm init

# Configure kubectl to use the new cluster
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

# Apply a pod network (e.g., Calico)
kubectl apply -f https://docs.projectcalico.org/v3.23/manifests/calico.yaml

# Allow scheduling pods on the control plane node (not recommended for production)
kubectl taint nodes --all node-role.kubernetes.io/master-

# Check the cluster status
kubectl get nodes
kubectl get pods --all-namespaces
```

Joining Worker Nodes:

```
# On each worker node, run the kubeadm join command obtained from the master node
initialization output
sudo kubeadm join <master-node-ip>:<master-node-port> --token <token> --discovery-
token-ca-cert-hash <hash>
```

Setting Up a Multi-Node Cluster (for Production):

1. Follow the above steps to set up the master node.
2. On each worker node, run the `kubeadm join` command obtained from the master node initialization output.
3. If using a cloud provider, you may need to set up networking and security groups accordingly.
4. Install a network plugin suitable for your environment. Example: Calico, Flannel, or Weave.

Verifying the Cluster:

```
# On the master node, check node status
kubectl get nodes

# List all pods in all namespaces
kubectl get pods --all-namespaces
```

These commands provide a basic setup. Depending on your use case, you may want to configure additional components like Ingress controllers, storage, or monitoring tools. Always refer to the official Kubernetes documentation for the most up-to-date and specific instructions for your environment.

Most Important K8 Commands with examples

Certainly! Below are 50 commonly used Kubernetes commands with descriptions and examples. These commands cover various aspects of managing Kubernetes clusters and applications. Please note that some commands might require specific resources or configurations in your cluster.

Cluster Information and Status:

1. **View Kubernetes cluster information:**

   ```
   kubectl cluster-info
   ```

2. **View nodes in the cluster:**

   ```
   kubectl get nodes
   ```

3. **View all resources in a namespace:**

   ```
   kubectl get all -n <namespace>
   ```

4. **Describe a specific resource:**

   ```
   kubectl describe <resource_type> <resource_name>
   ```

5. **View pod logs:**

   ```
   kubectl logs <pod_name>
   ```

Deployments and Pods:

6. **Create a deployment:**

   ```
   kubectl create deployment <deployment_name> --image=<image_name>
   ```

7. **Scale a deployment:**

   ```
   kubectl scale deployment <deployment_name> --replicas=<replica_count>
   ```

8. **Rolling restart of a deployment:**

   ```
   kubectl rollout restart deployment <deployment_name>
   ```

9. **Exposing a deployment as a service:**

   ```
   kubectl expose deployment <deployment_name> --port=<port>
   ```

10. **Run a one-time command in a pod:**

    ```
    kubectl exec -it <pod_name> -- <command>
    ```

Services:

11. **Create a service:**

    ```
    kubectl create service <service_type> <service_name> --
    tcp=<port>:<target_port>
    ```

12. **Expose a deployment using a service:**

    ```
    kubectl expose deployment <deployment_name> --type=<service_type> --
    port=<port>
    ```

13. **Describe a service:**

    ```
    kubectl describe service <service_name>
    ```

14. **Delete a service:**

    ```
    kubectl delete service <service_name>
    ```

Configurations:

15. **View ConfigMap details:**

```
kubectl get configmap <configmap_name>
```

16. **Create a ConfigMap from file:**

```
kubectl create configmap <configmap_name> --from-file=<path/to/files>
```

17. **View Secret details:**

```
kubectl get secret <secret_name>
```

18. **Create a Secret from literal values:**

```
kubectl create secret generic <secret_name> --from-literal=key1=value1 --
from-literal=key2=value2
```

Networking:

19. **Get information about services and their endpoints:**

```
kubectl get svc
```

20. **Create an Ingress resource:**

```
kubectl apply -f ingress.yaml
```

21. **View Ingress details:**

```
kubectl get ingress
```

22. **Network policy details:**

```
kubectl get networkpolicies
```

23. **Enable or disable resource access for a pod:**

```
kubectl apply -f network-policy.yaml
```

Storage:

24. **View Persistent Volumes (PVs):**

```
kubectl get pv
```

25. **View Persistent Volume Claims (PVCs):**

```
kubectl get pvc
```

26. **Create a Persistent Volume Claim:**

```
kubectl apply -f persistent-volume-claim.yaml
```

Scaling and Autoscaling:

27. **Autoscale a deployment:**

```
kubectl autoscale deployment <deployment_name> --min=<min_replicas> --
max=<max_replicas> --cpu-percent=<cpu_percent>
```

28. **View horizontal pod autoscaler details:**

```
kubectl get hpa
```

29. **Manually trigger horizontal pod autoscaler:**

```
kubectl scale deployment <deployment_name> --replicas=<new_replica_count>
```

StatefulSets:

30. **Create a StatefulSet:**

```
kubectl apply -f statefulset.yaml
```

31. **Scale a StatefulSet:**

```
kubectl scale statefulset <statefulset_name> --replicas=<replica_count>
```

32. **Delete a StatefulSet and associated pods:**

```
kubectl delete statefulset <statefulset_name>
```

Debugging and Troubleshooting:

33. **Run a debug container in a pod:**

```
kubectl debug <pod_name> -it --image=<debug_image>
```

34. **Get events in the cluster:**

```
kubectl get events
```

35. **Check pod resource usage:**

```
kubectl top pod
```

Security:

36. **View pod security policies:**

```
kubectl get psp
```

37. **View pod security context:**

```
kubectl get pods --output=jsonpath='{range
.items[*]}{"\n"}{.metadata.name}{":"}{.spec.securityContext.runAsUser}{end}
'
```

38. **Create a service account:**

```
kubectl create serviceaccount <serviceaccount_name>
```

Namespaces:

39. **List all namespaces:**

```
kubectl get namespaces
```

40. **Switch to a different namespace:**

```
kubectl config set-context --current --namespace=<namespace_name>
```

41. **Create a namespace:**

```
kubectl create namespace <namespace_name>
```

Helm (Kubernetes Package Manager):

42. **Install a Helm chart:**

```
helm install <release_name> <chart_name>
```

43. **List Helm releases:**

```
helm list
```

44. **Upgrade a Helm release:**

```
helm upgrade <release_name> <chart_name>
```

Monitoring and Logging:

45. **View pod logs with timestamps:**

```
kubectl logs <pod_name> --timestamps
```

46. **Enable metrics server (for resource usage metrics):**

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-
server/releases/latest/download/components.yaml
```

47. **View resource usage metrics for nodes:**

```
kubectl top nodes
```

48. **View resource usage metrics for pods:**

```
kubectl top pods
```

Clean Up:

49. **Delete a resource:**

```
50. kubectl delete <resource_type> <resource_name>
51.
```

52. **Delete all resources in a namespace:**

```
kubectl delete all --all -n <namespace>
```

Errors in K8 & how to troubleshoot them

Kubernetes, like any complex system, can encounter various errors and issues during setup, configuration, and runtime. Here are 20 common errors in Kubernetes and ways to troubleshoot them:

1. Error: Unable to connect to the cluster

Troubleshooting:

- Verify your `kubectl` configuration using `kubectl config view`.
- Ensure that the cluster API server is accessible from your machine.
- Check if the cluster is running (`kubectl cluster-info`).

2. Error: ImagePullBackOff or ErrImagePull

Troubleshooting:

- Verify the image name and repository.
- Check image availability and permissions.
- Ensure network connectivity to the container registry.

3. Error: CrashLoopBackOff

Troubleshooting:

- Check pod logs (`kubectl logs <pod_name>`).
- Inspect the pod's events (`kubectl describe pod <pod_name>`).
- Verify resource constraints and application health.

4. Error: PersistentVolumeClaim is pending

Troubleshooting:

- Check storage class availability and status.
- Verify storage provider configuration.
- Ensure sufficient storage capacity in the cluster.

5. Error: Pod stays in Pending state

Troubleshooting:

- Check if there are enough resources in the cluster.
- Inspect pod events and describe the pod (`kubectl describe pod <pod_name>`).
- Examine the node's status and resource availability.

6. Error: Service is not accessible

Troubleshooting:

- Verify service configuration and ports.
- Check firewall rules and network policies.
- Ensure that the service selector matches pod labels.

7. Error: Unauthorized (RBAC-related issues)

Troubleshooting:

- Verify your user's RBAC permissions.
- Check roles and role bindings (`kubectl get roles`, `kubectl get rolebindings`).
- Ensure proper context and authentication settings in `kubectl`.

8. Error: Node NotReady

Troubleshooting:

- Check if kubelet is running on the node.
- Verify node status (`kubectl get nodes`).
- Examine kubelet logs for errors.

9. Error: Insufficient CPU or Memory

Troubleshooting:

- Check resource requests and limits in pod specs.
- Verify node capacity and resource utilization.
- Examine pod events and describe the pod.

10. Error: NamespaceNotFound

Troubleshooting:

- Ensure the specified namespace exists.
- Check your current context (`kubectl config current-context`).

11. Error: ConfigMap or Secret not found

Troubleshooting:

- Verify the resource name and namespace.
- Check if the resource was created (`kubectl get configmaps`).

12. Error: DNS Resolution Issues

Troubleshooting:

- Verify CoreDNS or kube-dns pods are running.
- Check DNS configuration in pods.
- Ensure that the service and pod names are correct.

13. Error: Invalid YAML Syntax

Troubleshooting:

- Use YAML linters to validate your configuration.
- Double-check indentation and syntax.
- Inspect error messages for specific details.

14. Error: NodeSelector does not match nodes

Troubleshooting:

- Check node labels and selectors in pod specifications.
- Verify that nodes match the required labels.

15. Error: Unable to access external services

Troubleshooting:

- Check network policies and firewall rules.
- Verify external service availability.
- Examine service endpoint configurations.

16. Error: ResourceQuota Exceeded

Troubleshooting:

- Check resource quotas and limits for namespaces.
- Verify resource utilization in the cluster.

17. Error: Ingress Controller Misconfiguration

Troubleshooting:

- Check Ingress resource for syntax errors.
- Verify Ingress controller logs.
- Ensure proper annotations and backend services.

18. Error: Custom Resource Definition (CRD) Not Found

Troubleshooting:

- Check if the CRD is installed (`kubectl get crds`).
- Verify API group and version.

19. Error: Unable to create LoadBalancer Service

Troubleshooting:

- Verify cloud provider credentials.
- Check if LoadBalancer services are supported in your environment.

20. Error: Node Affinity or Anti-Affinity Issues

Troubleshooting:

- Check node affinity or anti-affinity rules in pod specifications.
- Verify that nodes have the required labels.

When troubleshooting Kubernetes issues, it's essential to use a systematic approach, inspect logs, and make use of Kubernetes resources like `kubectl describe`, `kubectl logs`, and `kubectl get`. Additionally, monitoring tools and observability solutions can provide insights into the cluster's health and performance. Always refer to the official Kubernetes documentation for detailed troubleshooting guidance.