Kubernetes: A Simplified Guide

Chapter 1: Important Fundamentals, Core Concepts, and Components in Kubernetes Architecture

1.1 What is Kubernetes?

- **Orchestrator:** Kubernetes (often called k8s) is an open-source platform designed to automate deploying, scaling, and managing containerized applications.
- **Containers:** Think of containers as lightweight, portable, and self-sufficient packages that contain everything an application needs to run (code, libraries, dependencies). Docker is a popular containerization technology.
- **Abstraction:** Kubernetes provides an abstraction layer over your infrastructure (physical or virtual servers), allowing you to treat them as a single pool of resources.

1.2 Core Concepts

- **Cluster:** A Kubernetes cluster is a set of machines (physical or virtual), called nodes, that work together to run your applications.
- **Node:** A node is a worker machine in Kubernetes. It can be a physical server or a virtual machine.
- **Pod:** The smallest deployable unit in Kubernetes. A Pod represents a single instance of an application and can contain one or more containers. Containers in the same Pod share the same network and storage resources.
- **ReplicaSet:** Ensures that a specified number of Pod replicas are running at any given time. It helps with scaling and self-healing.
- **Deployment:** A higher-level abstraction that manages ReplicaSets. Deployments allow you to describe the desired state of your application (e.g., number of replicas, container image, etc.). Kubernetes will then automatically create/update the necessary ReplicaSets and Pods to match this state.
- **Service:** An abstraction that defines a logical set of Pods and a policy by which to access them. Services provide a stable IP address and DNS name for your application, even if the underlying Pods are recreated or moved.
- **Namespace:** A way to divide cluster resources between multiple users or teams. Namespaces provide a scope for names and help prevent naming conflicts.
- **ConfigMap:** Used to store configuration data as key-value pairs. This allows you to decouple configuration from your application code.
- **Secret:** Similar to ConfigMaps, but used to store sensitive data, such as passwords or API tokens. Secrets are encoded and can be encrypted for better security.
- **Volume:** A way to provide persistent storage to Pods. Volumes can be backed by various storage systems (e.g., local disk, cloud storage, network file systems).
- **Ingress:** Manages external access to the services in a cluster, typically HTTP. Ingress can provide load balancing, SSL termination, and name-based virtual hosting.

1.3 Kubernetes Architecture

A Kubernetes cluster consists of two main parts:

- **Control Plane (Master Node):** The brain of the cluster. It's responsible for managing the cluster state, scheduling applications, and responding to events.
    - **kube-apiserver:** The primary interface for interacting with the cluster. It exposes the Kubernetes API and handles authentication, authorization, and admission control.
    - **etcd:** A distributed key-value store that stores the cluster's configuration data and state.
    - **kube-scheduler:** Responsible for assigning Pods to Nodes based on resource availability and constraints.
    - **kube-controller-manager:** Runs various controller processes that regulate the state of the cluster (e.g., Node Controller, Replication Controller, Endpoints Controller).
    - **cloud-controller-manager:** An optional component that integrates with cloud providers to manage cloud-specific resources (e.g., load balancers, storage).
- **Worker Nodes:** The machines that run your applications.
    - **kubelet:** An agent that runs on each node and ensures that containers are running in a Pod as described in the PodSpec.
    - **kube-proxy:** A network proxy that maintains network rules on nodes and allows network communication to your Pods from inside or outside of your cluster.
    - **Container Runtime:** The software responsible for running containers (e.g., Docker, containerd, CRI-O).

1.4 How Kubernetes Works (Simplified)

1. **Define Desired State:** You describe the desired state of your application using YAML or JSON files (e.g., Deployment, Service).
2. **API Server Interaction:** You use `kubectl` (command-line tool) to send these definitions to the kube-apiserver.
3. **Control Plane Actions:**
    - The API server validates and stores the configuration in etcd.
    - The scheduler assigns Pods to suitable nodes.
    - Controllers ensure that the correct number of Pods are running.
4. **Node Actions:**
    - kubelet on each node receives instructions from the API server.
    - kubelet interacts with the container runtime to start/stop containers.
    - kube-proxy configures network rules for Pod communication.

Chapter 2: Important Kubernetes Commands (Cheat Sheet)

2.1 Cluster and Node Management

- `kubectl get nodes`: List all nodes in the cluster.
- `kubectl describe node <node-name>`: Get detailed information about a specific node.
- `kubectl top node`: Display resource usage (CPU/memory) for nodes.
- `kubectl cluster-info`: Display information about the cluster.
- `kubectl version`: Show the Kubernetes client and server versions.

2.2 Pod Management

- `kubectl get pods`: List all pods in the current namespace.

- `kubectl get pods -n <namespace>`: List pods in a specific namespace.
- `kubectl get pods -A`: List pods across all namespaces
- `kubectl describe pod <pod-name>`: Get detailed information about a specific pod.
- `kubectl logs <pod-name>`: View the logs of a container in a pod.
- `kubectl logs <pod-name> -c <container-name>`: View logs for a specific container.
- `kubectl exec -it <pod-name> -- /bin/bash`: Get a shell inside a running container.
- `kubectl delete pod <pod-name>`: Delete a pod.
- `kubectl run <pod-name> --image=<image-name>`: Run a new pod with given image

2.3 Deployment and ReplicaSet Management

- `kubectl get deployments`: List all deployments in the current namespace.
- `kubectl describe deployment <deployment-name>`: Get details about a deployment.
- `kubectl create deployment <deployment-name> --image=<image-name>`: Create a new deployment.
- `kubectl scale deployment <deployment-name> --replicas=<number>`: Scale a deployment up or down.
- `kubectl rollout status deployment/<deployment-name>`: Check the rollout status of a deployment
- `kubectl set image deployment/<deployment-name> <container-name>=<new-image>`: Update a deployment with a new image.
- `kubectl rollout undo deployment/<deployment-name>`: Rollback to a previous deployment version.
- `kubectl get replicasets`: List all ReplicaSets.

2.4 Service Management

- `kubectl get services`: List all services in the current namespace.
- `kubectl describe service <service-name>`: Get detailed information about a service.
- `kubectl expose deployment <deployment-name> --type=LoadBalancer --port=80`: Expose a deployment as a new service (e.g., LoadBalancer type).
- `kubectl delete service <service-name>`: Delete a service.

2.5 Namespace Management

- `kubectl get namespaces`: List all namespaces.
- `kubectl create namespace <namespace-name>`: Create a new namespace.
- `kubectl delete namespace <namespace-name>`: Delete a namespace.
- `kubectl config set-context --current --namespace=<namespace>`: Set the default namespace for kubectl commands

2.6 ConfigMap and Secret Management

- `kubectl get configmaps`: List all ConfigMaps.
- `kubectl create configmap <configmap-name> --from-literal=<key>=<value>`: Create a ConfigMap.

- `kubectl get secrets`: List all Secrets.
- `kubectl create secret generic <secret-name> --from-literal=<key>=<value>`: Create a Secret.

2.7 Other Useful Commands

- `kubectl apply -f <filename.yaml>`: Apply a configuration file to create/update resources.
- `kubectl delete -f <filename.yaml>`: Delete resources defined in a configuration file.
- `kubectl explain <resource>`: Get documentation about a specific resource type (e.g., `kubectl explain pod`).
- `kubectl get events`: View events in the cluster.

Example: Creating and Exposing a Deployment

1. **Create a Deployment YAML file (e.g., `my-deployment.yaml`):**

   ```yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: my-nginx-deployment
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: my-nginx
     template:
       metadata:
         labels:
           app: my-nginx
       spec:
         containers:
         - name: nginx
           image: nginx:latest
           ports:
           - containerPort: 80
   ```

2. **Create the Deployment:**

   ```
   kubectl apply -f my-deployment.yaml
   ```

3. **Expose the Deployment as a Service:**

   ```
   kubectl expose deployment my-nginx-deployment --type=LoadBalancer --
           port=80
   ```

4. **Get the external IP of the service:**

   ```
   kubectl get service my-nginx-deployment
   ```