SKILLS | DevOps and Cloud Computing

Introduction to Kubernetes

Objective

- Understand the importance of container orchestration and the history of Kubernetes.
- Compare Kubernetes with Docker Swarm and identify key differences.
- Learn about master and worker node components that dri Kubernetes functionality.
- Explore Kubernetes Pods, Nodes, Services, Deployments, and Namespaces.
- Gain hands-on experience creating and managing Pods using YAML files.
- Understand ReplicaSets and how to scale Pods with Deployments.

Explaining the need for container orchestration and challenges in managing large-scale containerized applications.

Let’s see

Container orchestration is essential for managing large-scale containerized applications due to the following reasons:

Need for Container Orchestration:

1. Automation and Scalability
2. Resilience and Availability
3. Resource Optimization
4. Simplified Management
5. Enhanced Security

Let’s see

Challenges in Managing Large-Scale Containerized Applications:

- Complexity
- Scalability Issues
- Security Concerns
- Failure Management
- Resource Management

PW SKILLS

Discuss the history of Kubernetes, its origin at Google, and how it became the leading orchestration tool

Let’s discuss

Kubernetes, the leading container orchestration tool, has a fascinating history rooted in Google's internal infrastructure innovations.

Origin at Google:

1. **Borg and Omega Inspiration:** Kubernetes was inspired by Google's Borg and Omega systems, which were internal cluster managers used to handle massive workloads efficiently.
2. **Development and Announcement:** In 2013, Google engineers Craig McLuckie, Joe Beda, and Brendan Burns pitched the idea of an open-source container management system based on Borg. Kubernetes was officially announced on June 6, 2014.
3. **Open Source Release:** Kubernetes version 1.0 was released on July 21, 2015, with Google donating it to the Cloud Native Computing Foundation (CNCF), ensuring community-driven development.

Let’s discuss

Rise to Dominance:

1. **Community and Ecosystem:** Kubernetes quickly gained traction due to its flexibility, scalability, and open-source nature. Major tech companies like Red Hat, Microsoft, IBM, and Docker joined the effort early on.
2. **Broad Adoption:** By 2017, competitors like AWS and VMware integrated Kubernetes into their platforms, solidifying its position as the standard for container orchestration.
3. **Global Impact:** Today, Kubernetes is one of the largest open-source projects globally, used by 71% of Fortune 100 companies for managing microservices and cloud-native applications.

PW SKILLS

Introducing key Kubernetes features such as self-healing, auto-scaling, and declarative configuration

Kubernetes features

Kubernetes offers several key features that make it a powerful container orchestration tool:

1. **Self-Healing:**

Kubernetes automatically detects and resolves issues with containers. It restarts failed containers, replaces unhealthy ones, and ensures that only ready containers serve requests. This guarantees application resiliency and minimizes downtime.

Kubernetes features

2. Auto-Scaling:

Kubernetes dynamically adjusts the number of running containers based on resource usage (e.g., CPU or memory). Horizontal scaling allows applications to handle increased demand efficiently without manual intervention.

3. Declarative Configuration:

Users define the desired state of their applications using YAML or JSON files. Kubernetes continuously works to maintain this state, automating rollouts, rollbacks, and updates while ensuring consistency across deployments.

PW SKILLS

Compare Kubernetes and Docker Swarm in terms of:

Kubernetes v/s Docker swarm

1. **Architecture and Complexity**

- Kubernetes: Uses a master-worker architecture with a central control plane (master) managing worker nodes. It offers extensive features but has a steep learning curve due to its complexity and configuration requirements.
- Docker Swarm: Employs a manager-worker architecture, where manager nodes control the cluster. It is simpler to set up and use, making it ideal for smaller teams or workloads.

Kubernetes v/s Docker swarm

2. Scaling and Networking

Scaling:

- Kubernetes supports horizontal autoscaling natively, allowing dynamic scaling based on resource utilization or traffic.
- Docker Swarm allows manual scaling via commands but lacks built-in autoscaling capabilities.

Networking:

- Kubernetes provides advanced networking features like network policies, service meshes, and customizable plugins for fine-grained traffic control.
- Docker Swarm has a simpler networking model with built-in overlay networks and encrypted communication but fewer advanced features.

Kubernetes v/s Docker swarm

3. **Deployment and Service Discovery**

Deployment:

- Kubernetes uses YAML manifests to define deployments with robust abstractions like Pods and StatefulSets. It supports rolling updates and rollbacks.
- Docker Swarm uses docker-compose.yml files for declarative configurations. It also supports rolling updates but is less feature-rich compared to Kubernetes.

Service Discovery:

- Kubernetes assigns services DNS names within the cluster, with flexible load balancing configurations via Ingress resources.
- Docker Swarm provides straightforward service discovery with DNS names and automatic load balancing using its routing mesh.

Discussing why Kubernetes is the industry-standard for production environments

Let’s discuss

Kubernetes has become the industry standard for production environments due to its robust features and ability to meet the demands of modern, large-scale applications. Key reasons include:

1. Scalability and High Availability
2. Flexibility and Portability
3. Advanced Orchestration Features
4. Security and Resource Management
5. Ecosystem and Community Support

PW SKILLS

Explaining the role of master node components in controlling the cluster

Let’s see

The master node in Kubernetes plays a critical role in controlling the cluster through its core components:

1. **API Server**

- Acts as the front-end for all Kubernetes operations, exposing a RESTful API for communication.
- Processes and validates requests from users, tools (like kubectl), and other components, ensuring state consistency by interacting with etcd.
- Serves as the central communication hub, managing authentication, authorization, and state updates.

Let’s see

2. **Controller Manager**

Runs multiple controllers to maintain the desired state of the cluster.

- Node Controller: Monitors node health and availability.
- Replication Controller: Ensures the correct number of pod replicas are running.
- Endpoint Controller: Updates service endpoints with active pods.

Operates in a reconciliation loop to align the cluster's current state with its desired state.

Let’s see

3. **Scheduler**

- Assigns workloads (pods) to nodes by evaluating resource availability and constraints.
- Uses scoring algorithms to select the most suitable node for each pod and informs the API Server of its decision.
- Ensures efficient resource utilization across the cluster.

4. **etcd**

- A distributed key-value store that holds all cluster configuration and state data.
- Stores critical information like pod specifications, service configurations, and secrets, ensuring high availability and consistency across the cluster.

PW SKILLS

Explaining worker node components responsible for running applications

Let’s see

Worker nodes in Kubernetes are responsible for running containerized applications. Their key components include:

1. **kubelet**
- Acts as an agent on each node, communicating with the API Server to ensure containers within pods are running and healthy.
- Receives pod specifications (PodSpecs) and instructs the container runtime to manage containers accordingly.

2. **kube-proxy**
- Manages network rules on nodes to enable communication between pods and services within or outside the cluster.
- Uses operating system packet filtering or forwards traffic directly to implement Kubernetes Service networking.

Take A 5-Minute **Break!**

- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes

PW SKILLS

Explaining Pods as the smallest deployable unit in Kubernetes

Let’s see

In Kubernetes, Pods are the smallest deployable units of computing. They are designed to encapsulate one or more containers along with shared storage, networking, and specifications for running the containers. Here’s a brief explanation:

Structure:

- A Pod acts as a "logical host," co-locating tightly coupled containers that share resources like storage volumes and network namespaces. Containers within a Pod can communicate using localhost and share a single IP address.

Let’s see

Types:

- Single-container Pods
- Multi-container Pods

Lifecycle: Pods are ephemeral; they remain on a node until terminated, deleted, or the node fails. Kubernetes manages pod replication for scaling and high availability.

PW SKILLS

Discussing how Nodes function as physical or virtual machines running workloads.

Let’s discuss

In Kubernetes, Nodes are physical or virtual machines that run containerized workloads as part of a cluster. Here’s how they function:

Workload Hosting: Nodes host Pods, which are the smallest deployable units in Kubernetes, containing one or more containers. Each node can run multiple Pods based on its resource capacity.

Let’s discuss

Components:

- kubelet
- kube-proxy
- Container Runtime

Control Plane Management: Nodes are managed by the control plane, which schedules workloads based on available resources and ensures nodes meet health requirements.

PW SKILLS

Introducing Services for exposing Pods and enabling communication

Services

In Kubernetes, Services are used to expose Pods and enable communication between them or with external clients. Here’s a brief explanation:

Purpose: Pods are ephemeral and can be replaced dynamically, making their IP addresses unreliable for direct communication. Services provide a stable IP address and DNS name to ensure consistent access to a group of Pods performing the same function.

Functionality: Services act as an abstraction layer, routing network traffic to the appropriate Pods based on labels and selectors. This ensures that requests reach healthy and active Pods.

Services

Types: Kubernetes supports various Service types:

- ClusterIP:
- NodePort:
- LoadBalancer:
- ExternalName:

Explaining Deployments for managing application lifecycles

Let’s see

In Kubernetes, Deployments are used to manage the lifecycle of applications, ensuring reliability and automation. Here's a concise explanation:

- **Purpose:** Deployments define the desired state of an application, including container images, the number of replicas (Pods), and update strategies. They automate tasks like scaling, updating, and rolling back applications.
- **Automation:** The deployment controller continuously monitors and reconciles the actual state with the desired state. It automatically replaces failed Pods and ensures the specified number of replicas are running.

Let’s see

Update Strategies:

- Rolling Updates
- Recreate Strategy

Benefits: Deployments simplify application management by automating updates, scaling, and rollback processes, ensuring high availability and predictable behavior.

PW SKILLS

Discussing Namespaces for organizing and managing multi-tenant Kubernetes clusters.

Let’s discuss

Kubernetes Namespaces are a powerful feature for organizing and managing multi-tenant clusters by logically partitioning resources. Here’s a concise explanation:

- Segmentation and Isolation
- Resource Management
- Access Control
- Multi-Tenancy
- Simplified Administration

Explaining the difference between single-container Pods and multi-container Pods

Let’s see

Single-Container Pods

- **Definition:** Host only one container, making them simpler to create and manage.
- **Use Case:** Ideal for most applications where a single container encapsulates all necessary functionality.
- **Benefits:** Easier to scale and manage since Kubernetes directly controls Pods rather than individual containers.

Let’s see

Multi-Container Pods

- **Definition:** Host multiple containers that share resources like storage volumes and network namespaces.
- **Use Case:** Suitable for tightly coupled containers that need to work together, such as a main application container paired with a sidecar container for logging or monitoring.
- **Benefits:** Containers can communicate via localhost and share storage, enabling efficient collaboration. However, scaling is more complex as all containers in the Pod scale together.

Discussing how multi-container Pods share storage and networking, making them useful for sidecar patterns.

Let’s discuss

Multi-container Pods in Kubernetes share storage and networking, making them highly useful for sidecar patterns. Here's how:

- **Shared Storage:** Containers within a Pod can access shared volumes, enabling seamless data exchange.
- **Shared Networking:** All containers in a Pod share the same network namespace, including IP address and ports. This allows them to communicate using localhost, simplifying inter-container communication without external networking configurations.

Let’s discuss

- **Sidecar Pattern Use Case:** Sidecar containers complement the main application by performing auxiliary tasks, such as logging, monitoring, or data transformation. For instance, a sidecar might process logs generated by the main container or refresh files in a shared volume.

PW SKILLS

Discussing different Kubernetes installation methods

Let’s discuss

1. **Minikube for Local Development**

- **Purpose:** Simplifies Kubernetes setup on a local machine for development and testing.
- **Features:** Creates a single-node cluster using Docker or virtual machines. Supports multiple container runtimes (e.g., Docker, Containerd).
- **Use Case:** Ideal for developers testing applications in a lightweight, local environment.

Let’s discuss

2. **kubeadm for Setting Up Clusters**

- **Purpose:** Automates the setup of multi-node Kubernetes clusters by initializing and configuring control plane and worker nodes.
- **Features:** Provides flexibility for custom configurations and supports high availability setups.
- **Use Case:** Suitable for production environments where manual control over cluster components is needed.

Let’s discuss

3. **Managed Kubernetes Services (GKE, AKS, EKS)**

- **Purpose:** Cloud providers manage cluster infrastructure, scaling, upgrades, and maintenance.
- **Features:** Offers seamless integration with cloud services like AWS (EKS), Azure (AKS), and Google Cloud (GKE).
- **Use Case:** Ideal for production environments requiring minimal operational effort and scalability.

PW SKILLS

Providing an overview of the basic setup process

Overview

Here's a concise overview of a basic Kubernetes setup process:

1. **Install Tools:**

- **For local development:** Install Minikube or Docker Desktop (includes Kubernetes).
- **For production:** Use kubeadm or a managed service (e.g., GKE/AKS/EKS).

2. **Cluster Initialization:**

- **Minikube:** Run minikube start to create a single-node cluster.
- **kubeadm:** Use kubeadm init to bootstrap a control plane and kubeadm join to add worker nodes.
- **Managed Services:** Use cloud provider tools (e.g., gcloud container clusters create for GKE).

Overview

3. **Configure Access:**

Set up kubectl (Kubernetes CLI) to interact with the cluster using the kubeconfig file.

4. **Deploy Applications:**

Apply YAML manifests (e.g., kubectl apply -f deployment.yaml) to create Pods, Services, and Deployments.

5. **Verify Setup:**

Check cluster status with kubectl get nodes and application health with kubectl get pods.

PW SKILLS

Explaining the declarative approach of Kubernetes using YAML files

Let’s see

The declarative approach in Kubernetes uses YAML files to define the desired state of resources, allowing Kubernetes to reconcile the actual state with the specified configuration. Here's a concise explanation:

Workflow:

1. Write a YAML file defining the desired state.
2. Apply it using kubectl apply -f <file>.yaml.
3. Kubernetes ensures the cluster matches the defined state, handling updates and scaling automatically.

PW SKILLS

Demonstrating writing a basic Pod configuration in YAML

Let’s do it

Here’s an example of a basic Pod configuration in YAML:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  labels:
    app: example
spec:
  containers:
  - name: example-container
    image: nginx:latest
    ports:
    - containerPort: 80
```

Discussing best practices for organizing Kubernetes YAML manifests

Let’s discuss

Here are best practices for organizing Kubernetes YAML manifests:

1. Use Clear and Consistent Naming Conventions
2. Specify Resource Requests and Limits
3. Version Control Your YAML Files
4. Group Related Resources
5. Leverage ConfigMaps and Secrets
6. Use Labels and Annotations
7. Validate YAML Files

PW SKILLS

Explaining ReplicaSets and their role in ensuring high availability of applications

Let’s explain

ReplicaSets in Kubernetes are controllers designed to ensure high availability of applications by maintaining a stable number of Pod replicas. Here's an explanation of their role:

- **Purpose:** ReplicaSets guarantee that the specified number of identical Pods are running at any given time. If a Pod fails, is deleted, or becomes unresponsive, the ReplicaSet automatically creates new Pods to restore the desired count.

Components:

- Replicas
- Selector
- Pod Template

PW SKILLS

Discussing how Deployments provide rolling updates and rollback mechanisms.

Let's discuss

1. **Rolling Updates:**

Process:

- The Deployment Controller starts replacing old Pods one at a time with new Pods based on the updated deployment manifest.
- Kubernetes monitors the health of new Pods using readiness probes before scaling down old Pods.
- Parameters like maxSurge (extra Pods allowed during updates) and maxUnavailable (Pods that can be offline during updates) can fine-tune the process.

Let’s discuss

2. **Rollback Mechanism:**

Process:

- Kubernetes records each Deployment version.
- If needed, use kubectl rollout undo deployment/<deployment-name> to restore the previous version.

Demonstrating how to scale a Deployment to increase Pod count

Let’s do it

1. **Check Current Deployment:**

Use the command:

```
kubectl get deployments
```

2. **Scale the Deployment:**

Run the following command to set the desired number of replicas:

```
kubectl scale deployment/<deployment-name> --replicas=<desired-count>
```

For example, to scale a Deployment named nginx-deployment to 4 replicas:

```
kubectl scale deployment/nginx-deployment --replicas=4
```

Let’s do it

3. **Verify Scaling:**

Check the updated Deployment status:

```
kubectl get deployments
```

Confirm the number of Pods:

```
kubectl get pods
```

PW SKILLS

Time for case study!

Important

- Complete the post-class assessment
- Complete assignments (if any)
- Practice the concepts and techniques taught in this sessior
- Review your lecture notes
- Note down questions and queries regarding this session ar consult the teaching assistants

Thanks!