**Shared Compartments for OKE (Oracle Kubernetes Engine)**

In Oracle Cloud Infrastructure (OCI), **compartments** are logical containers used to organize and isolate resources. They help manage access control, billing, and resource organization. When working with **Oracle Kubernetes Engine (OKE)**, shared compartments play a key role in enabling better **resource isolation** and **collaboration** across teams or projects.

**What are Shared Compartments?**

Shared compartments are OCI compartments that allow multiple teams, projects, or applications to share resources while maintaining a level of isolation. For OKE, this means you can create a shared compartment where Kubernetes clusters, node pools, and related resources (like load balancers, storage, and networking) are managed collectively but accessed by different teams or users based on their permissions.

**Key Benefits of Shared Compartments for OKE**

1. **Resource Isolation**:
   * Shared compartments allow you to isolate resources for different teams or projects within the same compartment.
   * For example, you can have multiple OKE clusters in a shared compartment, each serving a different application or team, while ensuring that resources are not mixed up.
2. **Collaboration Across Teams**:
   * Teams can share access to the same compartment while maintaining their own Kubernetes clusters and workloads.
   * This is useful in scenarios where teams need to collaborate on shared infrastructure (e.g., shared networking, storage, or databases) but still require autonomy over their Kubernetes environments.
3. **Centralized Management**:
   * Shared compartments enable centralized management of resources, such as networking (VCNs, subnets, security lists) and IAM policies.
   * Admins can define policies at the compartment level to control access and usage for all teams sharing the compartment.
4. **Cost Optimization**:
   * By sharing resources like networking or storage across multiple OKE clusters, you can reduce costs compared to creating separate compartments for each team or project.
5. **Simplified Access Control**:
   * OCI Identity and Access Management (IAM) policies can be applied at the compartment level, making it easier to manage permissions for multiple teams or users accessing the shared compartment.

**Use Cases for Shared Compartments in OKE**

1. **Multi-Tenant Environments**:
   * In organizations with multiple teams or departments, shared compartments allow each team to have its own OKE cluster while sharing common resources like VCNs, subnets, or load balancers.
2. **DevOps and CI/CD Pipelines**:
   * Shared compartments can be used to host OKE clusters for different stages of a CI/CD pipeline (e.g., dev, test, prod) while maintaining isolation between environments.
3. **Cross-Team Collaboration**:
   * Teams working on related projects can share a compartment to access common resources (e.g., databases, object storage) while maintaining separate OKE clusters for their workloads.
4. **Centralized Networking**:
   * Shared compartments allow you to define a single Virtual Cloud Network (VCN) and subnets that can be used by multiple OKE clusters, reducing the complexity of managing separate networks.

**How to Implement Shared Compartments for OKE**

1. **Create a Shared Compartment**:
   * Use the OCI Console or CLI to create a compartment specifically for shared resources.
2. **Define IAM Policies**:
   * Create policies to control access to the shared compartment. For example:

Copy

Allow group <Team-A> to manage cluster-family in compartment <Shared-Compartment>

Allow group <Team-B> to use virtual-network-family in compartment <Shared-Compartment>

1. **Set Up Networking**:
   * Create a VCN and subnets within the shared compartment. These can be used by all OKE clusters in the compartment.
2. **Deploy OKE Clusters**:
   * Create OKE clusters within the shared compartment. Each cluster can be configured for a specific team or project.
3. **Manage Resource Sharing**:
   * Use OCI Resource Manager or Terraform to automate the provisioning of shared resources (e.g., load balancers, storage) for OKE clusters.
4. **Monitor and Optimize**:
   * Use OCI Monitoring and Cost Management tools to track resource usage and optimize costs across the shared compartment.

**Best Practices for Using Shared Compartments with OKE**

1. **Granular IAM Policies**:
   * Define fine-grained IAM policies to ensure teams only have access to the resources they need within the shared compartment.
2. **Tagging Resources**:
   * Use OCI tags to label resources (e.g., OKE clusters, node pools) within the shared compartment. This helps in tracking usage and costs by team or project.
3. **Limit Shared Resources**:
   * Avoid overloading the shared compartment with too many resources. Use separate compartments for completely independent projects.
4. **Regular Audits**:
   * Periodically review IAM policies and resource usage to ensure compliance and optimal resource allocation.
5. **Backup and Disaster Recovery**:
   * Implement backup and disaster recovery plans for shared resources to minimize downtime for all teams using the compartment.

**Conclusion**

Shared compartments in OCI provide a powerful way to balance **resource isolation** and **collaboration** when using OKE. By organizing resources in a shared compartment, teams can work together efficiently while maintaining control over their Kubernetes environments. Proper planning, IAM policies, and resource management are key to successfully leveraging shared compartments for OKE.

**Lab 5 : Demonstrating Shared Compartments for OKE**

This lab guide will walk you through setting up **shared compartments** in Oracle Cloud Infrastructure (OCI) and using them with **Oracle Kubernetes Engine (OKE)**. The goal is to demonstrate how shared compartments enable **resource isolation** and **collaboration** across teams or projects.

**Lab Objectives**

1. Create a shared compartment in OCI.
2. Set up a Virtual Cloud Network (VCN) and subnets in the shared compartment.
3. Deploy two OKE clusters in the shared compartment for two different teams.
4. Configure IAM policies to control access for each team.
5. Demonstrate resource sharing (e.g., networking) and isolation (e.g., Kubernetes workloads).

**Prerequisites**

1. An OCI tenancy with administrative privileges.
2. Access to the OCI Console.
3. OCI CLI installed and configured (optional but recommended).
4. Basic knowledge of Kubernetes and OCI networking.

**Step 1: Create a Shared Compartment**

1. **Log in to the OCI Console**:
   * Go to the [OCI Console](https://cloud.oracle.com/).
2. **Navigate to Compartments**:
   * In the OCI Console, go to **Identity & Security > Compartments**.
3. **Create a New Compartment**:
   * Click **Create Compartment**.
   * Fill in the details:
     + **Name**: Shared-OKE-Compartment
     + **Description**: "Shared compartment for OKE network resources."
     + **Parent Compartment**: Choose the root compartment or a parent compartment where this will reside.
   * Click **Create**.
4. Create two more compartments team-A-compartmnt and team-B-compartmnt

**Step 2: Set Up Networking in the Shared Compartment**

1. **Create a Virtual Cloud Network (VCN)**:
   * Go to **Networking > Virtual Cloud Networks**.
   * Click **Create VCN**.
   * Fill in the details:
     + **Name**: Shared-OKE-VCN
     + **Compartment**: Select Shared-OKE-Compartment.
     + **CIDR Block**: 10.0.0.0/16 (or any CIDR range you prefer).
   * Click **Create**.
2. **Create Subnets**:
   * Inside the Shared-OKE-VCN, create two subnets:
     + **Subnet 1**:
       - **Name**: Team-A-Subnet
       - **CIDR Block**: 10.0.0.0/22
       - **Subnet Type**: Regional
     + **Subnet 2**:
       - **Name**: Team-B-Subnet
       - **CIDR Block**: 10.0.4.0/22
       - **Subnet Type**: Regional
   * Ensure both subnets are created in the Shared-OKE-Compartment.
3. **Configure Security Lists**:
   * Create a security list to allow traffic between the subnets and external access for OKE.
   * Example rules:
     + Allow inbound TCP traffic on port 6443 (Kubernetes API).
     + Allow inbound TCP traffic on port 22 (SSH).
     + Allow all outbound traffic.

**Step 3: Configure IAM Policies for Teams**

1. **Create Groups for Teams**:
   * Go to **Identity & Security > Groups**.
   * Create two groups:
     + **Group Name**: Team-A-Group
     + **Group Name**: Team-B-Group
2. **Create Policies**:
   * Go to **Identity & Security > Policies**.
   * Create a policy in the root compartment to allow access to the shared compartment:

# Allow Team A to manage clusters in their own compartment

Allow group Team-A-Group to manage cluster-family in compartment Team-A-Compartment

# Allow Team B to manage clusters in their own compartment

Allow group Team-B-Group to manage cluster-family in compartment Team-B-Compartment

# Allow both teams to use shared networking resources

Allow group Team-A-Group to use virtual-network-family in compartment Shared-OKE-Compartment

Allow group Team-B-Group to use virtual-network-family in compartment Shared-OKE-Compartment

1. **Add Users to Groups**:
   * Add users to Team-A-Group and Team-B-Group based on their roles.

**Step 4: Deploy OKE Clusters in the Shared Compartment**

1. **Create OKE Cluster for Team A**:
   * Go to **Developer Services > Kubernetes Clusters (OKE)**.
   * Click **Create Cluster**.
   * Fill in the details:
     + **Name**: Team-A-Cluster
     + **Compartment**: Team-A-compartment
     + **VCN**: Shared-OKE-VCN (change compartment to see the network vcn)
     + **Subnets**: Select Team-A-Subnet.
     + **Node Pool**: Configure a node pool with 2 nodes.
   * Click **Create**.
2. **Create OKE Cluster for Team B**:
   * Repeat the same steps to create a second cluster:
     + **Name**: Team-B-Cluster
     + **Subnets**: Select Team-B-Subnet.
3. **Wait for Clusters to Be Active**:
   * It may take a few minutes for the clusters to be provisioned.

**Step 5: Access and Test the OKE Clusters**

1. **Download kubeconfig for Each Cluster**:
   * For each cluster, go to the cluster details page and click **Access Cluster**.
   * Follow the instructions to download the kubeconfig file.
2. **Set Up kubectl**:
   * Merge the kubeconfig files into your local ~/.kube/config file.
   * Use kubectl config use-context to switch between clusters.
3. **Deploy Sample Workloads**:
   * For Team-A-Cluster:

bash

Copy

kubectl create deployment team-a-app --image=nginx

kubectl expose deployment team-a-app --port=80 --type=LoadBalancer

* + For Team-B-Cluster:

bash

Copy

kubectl create deployment team-b-app --image=nginx

kubectl expose deployment team-b-app --port=80 --type=LoadBalancer

1. **Verify Isolation**:
   * Check that workloads in Team-A-Cluster are isolated from Team-B-Cluster.

**Step 6: Demonstrate Resource Sharing**

1. **Shared Networking**:
   * Show that both clusters use the same VCN but different subnets.
   * Use oci network subnet list to verify.
2. **Shared Load Balancer**:
   * Check that the Load Balancers for both clusters are created in the shared compartment.
3. **IAM Policies in Action**:
   * Log in as a user from Team-A-Group and verify they cannot modify resources in Team-B-Cluster.

**How This Works**

* **Team A** can only manage resources in Team-A-Compartment.
* **Team B** can only manage resources in Team-B-Compartment.
* Both teams can still use shared networking resources in Shared-OKE-Compartment.

**Step 7: Clean Up**

1. **Delete OKE Clusters**:
   * Go to **Developer Services > Kubernetes Clusters (OKE)**.
   * Delete Team-A-Cluster and Team-B-Cluster.
2. **Delete Networking Resources**:
   * Delete the subnets and VCN in the shared compartment.
3. **Delete the Shared Compartment**:
   * Go to **Identity & Security > Compartments**.
   * Delete Shared-OKE-Compartment.

**Conclusion**

This lab demonstrates how shared compartments in OCI enable resource isolation and collaboration for OKE clusters. By following this guide, you’ve learned how to:

* Create and manage shared compartments.
* Set up networking and IAM policies for shared resources.
* Deploy and manage OKE clusters in a shared compartment.

This approach is ideal for multi-team environments where resource sharing and isolation are both required.

**LAB 6 : SECRETS :**

**encrypt the data first of all**

**password :**

**echo 'ramankhanna123' | base64**

**username :**

**echo 'ramankhanna' | base64**

**462 cat secret.yaml**

**463 kubectl apply -f secret.yaml**

**464 kubectl get secret**

**465 kubectl describe secret my-secrets**

**466 kubectl describe secret my-secrets -o yaml**

**467 vi secret-pod.yaml**

**468 kubectl apply -f secret**

**469 kubectl apply -f secret-pod.yaml**

**470 kubectl get pods**

**471 kubectl exec -it myapp-pod1 -- /bin/bash**

**472 kubectl exec -it myapp-pod2 -- /bin/bash**

**#to access username and password as env variables in pod2 :**

**echo $SECRET\_USERNAME**

**echo $SECRET\_PASSWD**

**473 cat secret-pod.yaml**

**474 cat secret.yaml**

**475 history**

**root@tech-master:/gagan# cat secret.yaml**

**apiVersion: v1**

**kind: Secret**

**metadata:**

**name: my-secrets**

**type: Opaque**

**data:**

**username: cmFtYW5raGFubmEK**

**password: cmFtYW5raGFubmExMjMK**

**root@tech-master:/gagan# cat secret-pod.yaml**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: myapp-pod1**

**labels:**

**app: myapp**

**spec:**

**containers:**

**- name: httpd-container**

**image: httpd**

**volumeMounts:**

**- name: credentials**

**mountPath: /tmp/creds**

**readOnly: true**

**volumes:**

**- name: credentials**

**secret:**

**secretName: my-secrets**

**---**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: myapp-pod2**

**labels:**

**app: myapp**

**type: front-end**

**spec:**

**containers:**

**- name: httpd-container**

**image: httpd**

**env:**

**- name: SECRET\_USERNAME**

**valueFrom:**

**secretKeyRef:**

**name: my-secrets**

**key: username**

**- name: SECRET\_PASSWD**

**valueFrom:**

**secretKeyRef:**

**name: my-secrets**

**key: password**

**root@tech-master:/gagan#**

**===========**

**===========**

**LAB 7 : CONFIG MAPS:**

**CONFIGMAPS:**

**echo "Hello from production" > prod.html**

**echo "Hello from dev" > dev.html**

**kubectl get configmaps**

**kubectl create configmap prod.cmap --from-file=prod.html**

**kubectl create configmap dev.cmap --from-file=dev.html**

**kubectl get configmaps**

**kubectl get configmap dev.cmap -o yaml**

**kubectl get configmap prod.cmap -o yaml**

**vi pods.yaml**

**----**

**root@gagan-master:/gagan/cmap# cat pods.yaml**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: prod-nginx**

**labels:**

**app: prod-nginx**

**spec:**

**containers:**

**- name: nginx**

**image: nginx:latest**

**ports:**

**- containerPort: 80**

**volumeMounts:**

**- name: config-volume**

**mountPath: /usr/share/nginx/html**

**volumes:**

**- name: config-volume**

**configMap:**

**name: prod.cmap**

**items:**

**- key: prod.html**

**path: index.html**

**---**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: dev-nginx**

**labels:**

**app: dev-nginx**

**spec:**

**containers:**

**- name: nginx**

**image: nginx:latest**

**ports:**

**- containerPort: 80**

**volumeMounts:**

**- name: config-volume**

**mountPath: /usr/share/nginx/html**

**volumes:**

**- name: config-volume**

**configMap:**

**name: dev.cmap**

**items:**

**- key: dev.html**

**path: index.html**

**root@gagan-master:/gagan/cmap#**

**Network Concepts in OKE Clusters**

Oracle Kubernetes Engine (OKE) is a managed Kubernetes service provided by Oracle Cloud Infrastructure (OCI). It simplifies the deployment, management, and scaling of containerized applications using Kubernetes. Networking is a critical component of OKE clusters, as it ensures seamless communication between pods, nodes, and external systems. Below is a detailed explanation of the networking concepts in OKE clusters, including OCI networking, Dynamic Routing Gateway (DRG), Dedicated Region Cloud@Customer (DRCC), and Full Stack Disaster Recovery (FSDR).

**1. Introduction to Networking in OCI**

Networking in OCI is designed to provide secure, scalable, and high-performance connectivity for workloads running in the cloud. Key components of OCI networking include:

* **Virtual Cloud Network (VCN):**
  + A VCN is a software-defined network that you can configure in OCI. It acts as a private network for your cloud resources, such as compute instances, databases, and OKE clusters.
  + A VCN includes subnets, route tables, security lists, and gateways to control traffic flow.
  + Subnets can be public (accessible from the internet) or private (isolated from the internet).
* **Subnets:**
  + Subnets are subdivisions of a VCN and are used to segment resources based on their function or security requirements.
  + In OKE, worker nodes and load balancers are typically deployed in different subnets for better isolation and security.
* **Security Lists and Network Security Groups (NSGs):**
  + Security Lists and NSGs are used to define firewall rules for inbound and outbound traffic.
  + NSGs provide more granular control by allowing you to apply rules to specific resources (e.g., pods or nodes).
* **Load Balancers:**
  + OCI provides both public and private load balancers to distribute traffic across multiple instances or pods in an OKE cluster.
  + Load balancers are essential for exposing applications running in OKE to external users.
* **Internet Gateway (IGW):**
  + An IGW allows resources in a public subnet to communicate with the internet.
  + It is used for inbound and outbound internet traffic.
* **NAT Gateway:**
  + A NAT Gateway allows resources in a private subnet to access the internet for updates or downloads while preventing inbound internet traffic.
* **Service Gateway:**
  + A Service Gateway provides secure access to OCI services (e.g., Object Storage) without requiring internet access.
* **FastConnect:**
  + FastConnect is a dedicated private network connection between your on-premises data center and OCI, offering higher bandwidth and lower latency compared to internet-based connections.

**2. Dynamic Routing Gateway (DRG)**

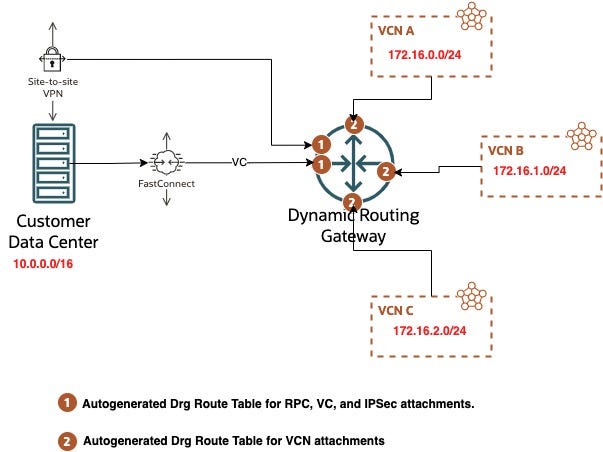
The **Dynamic Routing Gateway (DRG)** is a virtual router that provides a path for network traffic between your VCN and external networks, such as on-premises data centers, other VCNs, or third-party networks.

**Key Features of DRG:**

* **Connectivity Between On-Premises and OCI:**
  + DRG enables secure and high-performance connectivity between your on-premises network and OCI using VPN or FastConnect.
  + VPN Connect provides an encrypted connection over the internet, while FastConnect offers a dedicated private connection.
* **Inter-VCN Routing:**
  + DRG allows you to connect multiple VCNs within the same region or across regions, enabling communication between resources in different VCNs.
* **Peering with Third-Party Networks:**
  + DRG supports peering with third-party networks, such as other cloud providers or SaaS platforms.
* **Route Tables:**
  + DRG uses route tables to control the flow of traffic between your VCN and external networks.

**Use Cases for DRG in OKE:**

* **Hybrid Cloud Deployments:**
  + DRG enables OKE clusters to communicate with on-premises applications or databases.
* **Multi-Region Deployments:**
  + DRG facilitates communication between OKE clusters in different OCI regions.
* **Disaster Recovery:**
  + DRG is used in disaster recovery scenarios to ensure connectivity between primary and secondary sites.



**3. Dedicated Region Cloud@Customer (DRCC)**

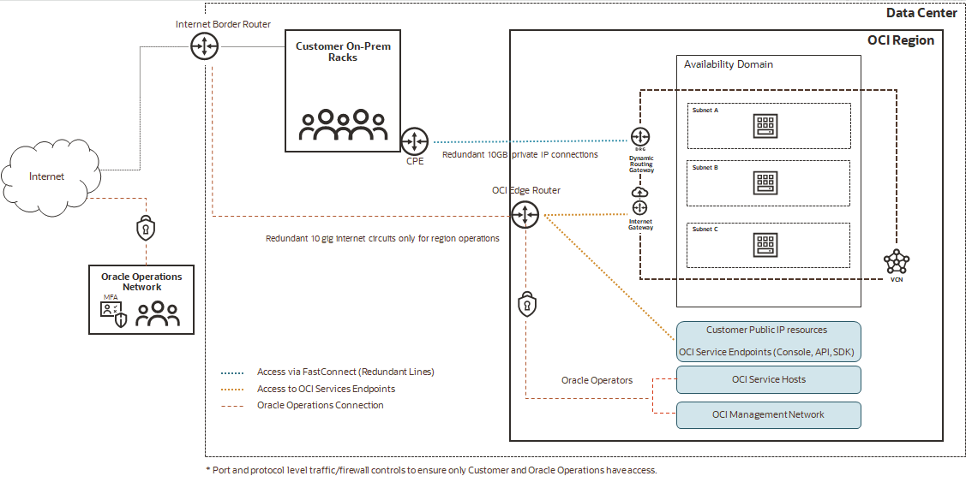
**Dedicated Region Cloud@Customer (DRCC)** is a fully managed OCI region that is deployed in your on-premises data center. It brings the full power of OCI, including compute, storage, networking, and services, to your premises while maintaining the same APIs, tools, and SLAs as the public OCI regions.

**Key Features of DRCC:**

* **On-Premises OCI Region:**
  + DRCC provides all OCI services locally, enabling you to run workloads on-premises while leveraging OCI's capabilities.
* **Consistent API and Tooling:**
  + DRCC uses the same APIs, CLI, and management tools as the public OCI regions, ensuring a seamless experience.
* **Data Residency and Compliance:**
  + DRCC ensures that data remains on-premises, addressing data residency and compliance requirements.
* **Hybrid Cloud Integration:**
  + DRCC can be integrated with public OCI regions using DRG and FastConnect for hybrid cloud deployments.

**Use Cases for DRCC in OKE:**

* **Regulated Industries:**
  + DRCC is ideal for industries with strict data residency requirements, such as healthcare, finance, and government.
* **Low-Latency Applications:**
  + DRCC enables low-latency access to applications and data by running OCI services on-premises.
* **Hybrid Kubernetes Deployments:**
  + OKE clusters can be deployed in DRCC to run containerized applications on-premises while integrating with public OCI regions.



**4. Full Stack Disaster Recovery (FSDR)**

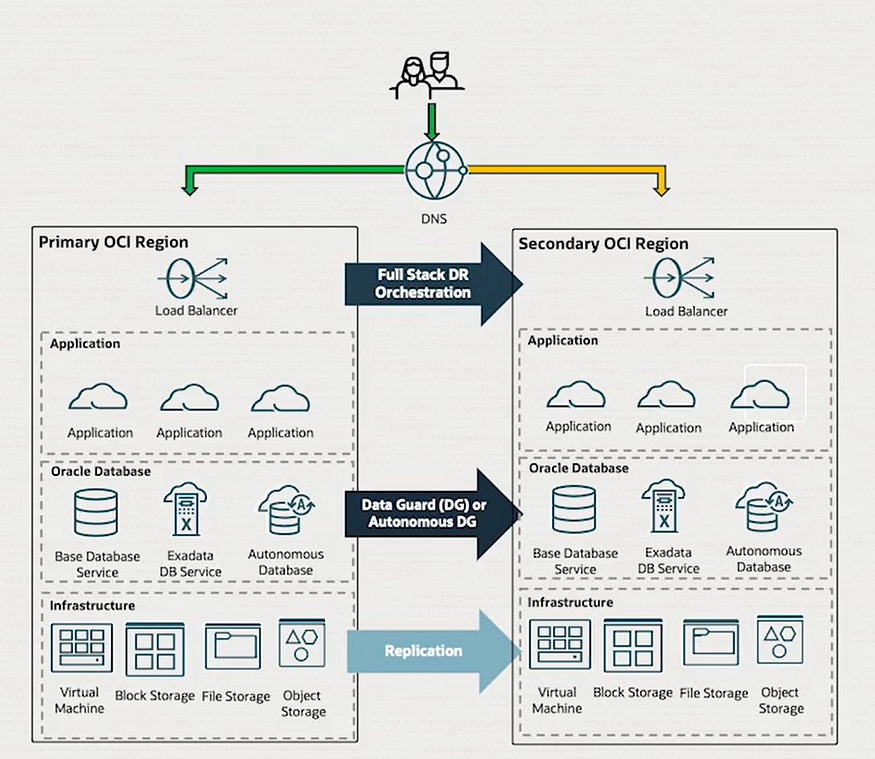
**Full Stack Disaster Recovery (FSDR)** is a comprehensive solution for ensuring business continuity and disaster recovery for applications and infrastructure running in OCI. FSDR provides automated failover and failback capabilities for both infrastructure and applications.

**Key Features of FSDR:**

* **Automated Failover and Failback:**
  + FSDR automates the process of switching over to a secondary site (failover) and returning to the primary site (failback) in the event of a disaster.
* **Application-Centric Recovery:**
  + FSDR ensures that not only the infrastructure but also the applications and data are recovered in a consistent state.
* **Multi-Region Support:**
  + FSDR supports disaster recovery across multiple OCI regions, ensuring high availability and resilience.
* **Integration with OKE:**
  + FSDR can be used to protect OKE clusters by replicating the cluster configuration, workloads, and data to a secondary region.

**Use Cases for FSDR in OKE:**

* **Mission-Critical Applications:**
  + FSDR ensures that mission-critical applications running in OKE are protected against regional outages or disasters.
* **Compliance and Business Continuity:**
  + FSDR helps organizations meet compliance requirements for disaster recovery and business continuity.
* **Zero Data Loss:**
  + FSDR ensures zero data loss by replicating data and workloads to a secondary region in real-time.



**Integration of Networking Concepts in OKE Clusters**

In an OKE cluster, the above networking concepts work together to provide a robust and scalable environment for running containerized applications:

1. **VCN and Subnets:**
   * OKE clusters are deployed within a VCN, with worker nodes and load balancers placed in specific subnets.
   * Security Lists and NSGs are used to control traffic to and from the cluster.
2. **DRG for Hybrid Connectivity:**
   * DRG enables OKE clusters to communicate with on-premises systems or other VCNs, supporting hybrid and multi-region deployments.
3. **DRCC for On-Premises OKE:**
   * DRCC allows OKE clusters to be deployed on-premises, providing low-latency access and data residency compliance.
4. **FSDR for Disaster Recovery:**
   * FSDR ensures that OKE clusters and their workloads are protected against disasters by replicating them to a secondary region.

**Conclusion**

Networking in OKE clusters is a critical aspect of ensuring secure, scalable, and high-performance deployments. By leveraging OCI's networking components, such as VCN, DRG, DRCC, and FSDR, organizations can build resilient and hybrid cloud architectures that meet their specific requirements. Whether you are running OKE clusters in the public cloud, on-premises, or in a hybrid environment, OCI provides the tools and services to ensure seamless connectivity and disaster recovery.

**Lab: Networking in OKE Clusters**

**Lab Objectives**

1. Create an OKE cluster within a Virtual Cloud Network (VCN).
2. Configure a Dynamic Routing Gateway (DRG) for hybrid connectivity.
3. Simulate a Dedicated Region Cloud@Customer (DRCC) setup.
4. Implement Full Stack Disaster Recovery (FSDR) for an OKE cluster.

**Lab Prerequisites**

1. **OCI Account:** Access to an Oracle Cloud Infrastructure (OCI) account with sufficient privileges to create resources.
2. **OCI CLI and Terraform:** Install and configure the OCI CLI and Terraform for automation.
3. **Kubectl:** Install kubectl to interact with the OKE cluster.
4. **FastConnect or VPN:** Access to a FastConnect or VPN connection for hybrid connectivity (optional for simulation).
5. **Two OCI Regions:** Access to two OCI regions (e.g., Phoenix and Ashburn) for FSDR.

**Lab Setup**

**Step 1: Create a Virtual Cloud Network (VCN)**

1. Log in to the OCI Console.
2. Navigate to **Networking > Virtual Cloud Networks**.
3. Click **Create VCN** and configure the following:
   * **Name:** oke-vcn
   * **CIDR Block:** 10.0.0.0/16
   * **Create Subnets:** Enable (this will create a public and private subnet).
4. Create the following subnets manually:
   * **Public Subnet:** 10.0.1.0/24 (for load balancers and internet-facing resources).
   * **Private Subnet:** 10.0.2.0/24 (for worker nodes).
5. Configure **Security Lists** to allow:
   * Inbound traffic on port 6443 (Kubernetes API server).
   * Inbound traffic on port 80/443 (for application access).
   * Outbound traffic to the internet (for worker nodes to pull container images).

**Step 2: Create an OKE Cluster**

1. Navigate to **Developer Services > Kubernetes Clusters (OKE)**.
2. Click **Create Cluster** and configure the following:
   * **Name:** oke-cluster
   * **VCN:** Select the oke-vcn created earlier.
   * **Subnets:** Assign worker nodes to the private subnet (10.0.2.0/24) and load balancers to the public subnet (10.0.1.0/24).
   * **Kubernetes Version:** Latest stable version.
   * **Node Pool:** Create a node pool with 2 worker nodes.
3. Click **Create** and wait for the cluster to be provisioned.
4. Download the kubeconfig file and configure kubectl:

bash

Copy

oci ce cluster create-kubeconfig --cluster-id <cluster-id> --file ~/.kube/config --region <region>

1. Verify the cluster:

bash

Copy

kubectl get nodes

**Step 3: Deploy a Sample Application**

1. Create a sample deployment:

yaml

Copy

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

Apply the deployment:

bash

Copy

kubectl apply -f nginx-deployment.yaml

1. Expose the deployment using a LoadBalancer service:

yaml

Copy

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

type: LoadBalancer

ports:

- port: 80

targetPort: 80

selector:

app: nginx

Apply the service:

bash

Copy

kubectl apply -f nginx-service.yaml

1. Get the public IP of the load balancer:

bash

Copy

kubectl get svc nginx-service

1. Access the application using the load balancer's public IP.

**Step 4: Configure Dynamic Routing Gateway (DRG)**

1. Navigate to **Networking > Dynamic Routing Gateways**.
2. Click **Create DRG** and name it oke-drg.
3. Attach the DRG to the oke-vcn:
   * Navigate to **DRG Attachments** and click **Create Attachment**.
   * Select the oke-vcn and the oke-drg.
4. Configure a **Route Table** in the VCN to route traffic to the DRG:
   * Add a route with the destination CIDR block of your on-premises network (e.g., 192.168.1.0/24) and the target as the DRG.
5. Simulate hybrid connectivity:
   * Use a VPN or FastConnect to connect your on-premises network to the DRG.
   * Verify connectivity by pinging an on-premises resource from a worker node in the OKE cluster.

**Step 5: Simulate Dedicated Region Cloud@Customer (DRCC)**

1. Simulate an on-premises OCI region by creating a second VCN in a different compartment:
   * Name: drcc-vcn
   * CIDR Block: 10.1.0.0/16
2. Create an OKE cluster in the drcc-vcn:
   * Follow the same steps as in Step 2.
3. Connect the drcc-vcn to the oke-vcn using a DRG:
   * Attach the drcc-vcn to the oke-drg.
   * Configure route tables to allow communication between the two VCNs.
4. Deploy a sample application in the DRCC OKE cluster and verify connectivity with the primary OKE cluster.

**Step 6: Implement Full Stack Disaster Recovery (FSDR)**

1. Create a second OKE cluster in a different OCI region (e.g., Ashburn).
2. Replicate the VCN and subnets in the secondary region:
   * Use Terraform or OCI Console to create a VCN with the same CIDR block and subnets.
3. Deploy the same application in the secondary OKE cluster.
4. Configure FSDR:
   * Use OCI's **Disaster Recovery** service to replicate the primary OKE cluster's configuration and workloads to the secondary region.
   * Simulate a disaster by deleting the primary OKE cluster.
   * Failover to the secondary OKE cluster and verify that the application is accessible.

**Lab Cleanup**

1. Delete the OKE clusters:

bash

Copy

oci ce cluster delete --cluster-id <cluster-id> --force

1. Delete the VCNs and subnets.
2. Detach and delete the DRG.
3. Remove any unused resources to avoid unnecessary charges.

**Lab Outcomes**

* Hands-on experience with OKE and OCI networking.
* Understanding of hybrid connectivity using DRG.
* Simulation of DRCC for on-premises use cases.
* Implementation of FSDR for disaster recovery.

This lab provides a comprehensive understanding of networking concepts in OKE clusters and their integration with OCI services.