**Native OCI Tools for Monitoring and Logging in OKE**

**1. OCI Monitoring Service**

The **OCI Monitoring Service** is a fully managed service that provides comprehensive visibility into the performance and health of your OKE clusters and the workloads running on them. It collects, aggregates, and analyzes metrics from various resources in your OKE environment.

**Key Features:**

* **Resource Metrics Collection**:
  + The OCI Monitoring Service automatically collects metrics from OKE components, such as:
    - Kubernetes nodes (worker nodes).
    - Pods, containers, and deployments.
    - CPU, memory, disk, and network usage.
    - Cluster autoscaler metrics (if enabled).
  + These metrics are collected at regular intervals and stored for analysis.
* **Health Tracking**:
  + The service monitors the health of your OKE cluster and its resources, such as:
    - Node status (e.g., ready, not ready).
    - Pod status (e.g., running, pending, failed).
    - Cluster events (e.g., scaling events, failures).
  + Alerts can be configured to notify you of any anomalies or critical issues.
* **Custom Metrics**:
  + You can push custom metrics from your applications running on OKE to the OCI Monitoring Service using the OCI SDKs or APIs.
* **Dashboards and Visualization**:
  + The service provides pre-built dashboards for OKE, allowing you to visualize metrics such as:
    - Cluster resource utilization.
    - Pod and container performance.
    - Node health and capacity.
  + You can also create custom dashboards tailored to your specific needs.
* **Alarms and Notifications**:
  + You can set up alarms based on specific thresholds (e.g., CPU usage > 80%) and configure notifications via email, OCI Notifications, or integration with third-party tools like PagerDuty or Slack.

**Use Cases for OCI Monitoring Service in OKE:**

* Proactively identify and resolve performance bottlenecks.
* Monitor cluster autoscaling and resource allocation.
* Track the health of workloads and ensure high availability.
* Optimize resource utilization to reduce costs.

**2. OCI Logging Service**

The **OCI Logging Service** is a fully managed service that enables you to collect, store, and analyze log data generated by your OKE clusters and applications. It provides a centralized platform for log management, making it easier to troubleshoot issues and gain insights into your Kubernetes environment.

**Key Features:**

* **Log Data Aggregation**:
  + The OCI Logging Service collects logs from various sources in your OKE cluster, including:
    - Kubernetes control plane logs (e.g., API server, scheduler, controller manager).
    - Worker node logs (e.g., kubelet, container runtime).
    - Application logs (from pods and containers).
    - Audit logs (for tracking API requests and changes to the cluster).
  + Logs are aggregated into a centralized log repository for easy access and analysis.
* **Log Search and Analysis**:
  + The service provides powerful search and filtering capabilities to help you quickly locate specific log entries.
  + You can use the OCI Logging Query Language (LQL) to perform advanced queries and analyze log data.
* **Log Retention and Archiving**:
  + Logs are stored securely in OCI Object Storage, with configurable retention periods.
  + You can archive logs for long-term storage and compliance purposes.
* **Integration with OCI Monitoring**:
  + Logs can be correlated with metrics from the OCI Monitoring Service to provide a comprehensive view of your OKE environment.
* **Custom Logs**:
  + You can configure custom log sources to collect logs from specific applications or services running on OKE.
* **Alarms and Notifications**:
  + Similar to the Monitoring Service, you can set up alarms based on log patterns (e.g., error logs) and receive notifications via email, OCI Notifications, or third-party tools.

**Use Cases for OCI Logging Service in OKE:**

* Troubleshoot issues in your Kubernetes cluster by analyzing control plane and node logs.
* Monitor application logs to identify errors or performance issues.
* Track changes to your cluster using audit logs for security and compliance.
* Gain insights into cluster behavior and optimize operations.

**How OCI Monitoring and Logging Services Work Together in OKE**

* **Unified Observability**:
  + The OCI Monitoring and Logging Services provide a unified observability platform for your OKE clusters. By combining metrics and logs, you can gain a holistic view of your cluster's performance and health.
* **Correlation of Metrics and Logs**:
  + For example, if the Monitoring Service detects a spike in CPU usage, you can use the Logging Service to investigate the corresponding logs and identify the root cause (e.g., a misbehaving application or a resource-intensive process).
* **Proactive Management**:
  + With alarms and notifications, you can proactively address issues before they impact your applications or users.

**Benefits of Using OCI Monitoring and Logging Services in OKE**

* **Fully Managed**: No need to set up or maintain additional monitoring or logging infrastructure.
* **Scalable**: Automatically scales with your OKE cluster and workloads.
* **Integrated**: Seamlessly integrates with other OCI services and third-party tools.
* **Cost-Effective**: Pay-as-you-go pricing model based on usage.

**Conclusion**

The OCI Monitoring and Logging Services are essential tools for managing and optimizing your OKE clusters. They provide deep visibility into your cluster's performance, health, and logs, enabling you to troubleshoot issues, ensure high availability, and optimize resource utilization. By leveraging these native OCI tools, you can simplify operations and focus on delivering value through your Kubernetes workloads.

**Lab 8: Monitoring and Logging in OKE using OCI Native Tools**

**Lab Objectives**

1. Set up an OKE cluster.
2. Deploy a sample application on the OKE cluster.
3. Configure OCI Monitoring Service to collect and visualize metrics.
4. Configure OCI Logging Service to collect and analyze logs.
5. Set up alarms and notifications for metrics and logs.

**Prerequisites**

1. **OCI Account**: Access to an Oracle Cloud Infrastructure (OCI) tenancy.
2. **Permissions**: Ensure you have the necessary permissions to create and manage OKE clusters, OCI Monitoring, and Logging Services.
3. **OCI CLI**: Install and configure the OCI CLI on your local machine.
4. **kubectl**: Install kubectl to interact with the OKE cluster.
5. **Docker**: Install Docker to build and push container images (optional, if you want to use a custom application).

**Step 1: Create an OKE Cluster**

1. Log in to the **OCI Console**.
2. Navigate to **Developer Services > Kubernetes Clusters (OKE)**.
3. Click **Create Cluster** and choose **Quick Create**.
4. Provide a name for your cluster (e.g., oke-monitoring-lab).
5. Configure the following:
   * **Kubernetes Version**: Choose the latest stable version.
   * **Node Shape**: Select a VM shape (e.g., VM.Standard.E4.Flex).
   * **Number of Nodes**: Set to 2.
6. Click **Create**.
7. Once the cluster is created, download the **kubeconfig** file and configure kubectl to access the cluster:

bash

Copy

oci ce cluster create-kubeconfig --cluster-id <cluster-ocid> --file $HOME/.kube/config --region <region>

1. Verify access to the cluster:

bash

Copy

kubectl get nodes

**Step 2: Deploy a Sample Application**

1. Create a simple Kubernetes deployment and service:

yaml

Copy

# sample-app.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: sample-app

spec:

replicas: 3

selector:

matchLabels:

app: sample-app

template:

metadata:

labels:

app: sample-app

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: sample-app-service

spec:

selector:

app: sample-app

ports:

- protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

1. Deploy the application:

bash

Copy

kubectl apply -f sample-app.yaml

1. Verify the deployment:

bash

Copy

kubectl get pods,svc

**Step 3: Configure OCI Monitoring Service**

1. Navigate to **Observability & Management > Monitoring** in the OCI Console.
2. **Explore Metrics**:
   * Go to **Metrics Explorer**.
   * Select your OKE cluster's compartment and namespace (oci\_oke).
   * Choose metrics
   * Visualize the metrics on a dashboard.

 By default, **only node-level metrics** are enabled in OCI Monitoring.

 To enable **pod-level metrics**, you need to **install metric server**

* **Enable metric server add on ..**
* **K top nodes , pods**

**✅ If this works, it confirms that Kubernetes Metrics Server is running!**

1. **Create a Custom Dashboard**:
   * Create a new dashboard and add widgets for the metrics you want to monitor (e.g., CPU, memory, and network usage).

**Step 4: Configure OCI Logging Service**

1. Navigate to **Observability & Management > Logging** in the OCI Console.
2. **Create a Log Group**:
   * Create a new log group (e.g., oke-logs).
3. **Create a Log**:
   * Go to logs ; enable service logs
   * Configure the log to collect:
     + Kubernetes control plane logs.
     + Worker node logs.
4. **View Logs**:
   * Use the **Log Search** feature to query logs (e.g., search for errors or specific events).

* **Go to logging analytics , enable it >> go to dashboards /solutions >> enable the OKE dashboard**
* **Explore the dashboard**

**Step 5: Generate Load and Analyze Metrics/Logs**

1. Generate load on the sample-app:
   * Use a tool like curl or ab (Apache Benchmark) to send requests to the application's external IP.

bash

Copy

ab -n 1000 -c 10 http://<external-ip>/

1. Monitor the metrics in the **OCI Monitoring Service**:
   * Observe CPU, memory, and network usage.
   * Check if alarms are triggered.
2. Analyze logs in the **OCI Logging Service** using log analytics explorer service ….
   * Look for application logs or errors generated during the load test.

**Step 6: Clean Up**

1. Delete the sample-app deployment:

bash

Copy

kubectl delete -f sample-app.yaml

1. Delete the OKE cluster:
   * Navigate to **Developer Services > Kubernetes Clusters (OKE)**.
   * Select your cluster and click **Delete**.
2. Delete the log group and alarms created in the OCI Logging and Monitoring Services.

**Lab Summary**

In this lab, you:

1. Created an OKE cluster and deployed a sample application.
2. Configured OCI Monitoring Service to collect and visualize metrics.
3. Configured OCI Logging Service to collect and analyze logs.
4. Set up alarms and notifications for metrics and logs.
5. Generated load on the application and analyzed the results.

This lab demonstrates how to use OCI's native monitoring and logging tools to gain visibility into your OKE clusters and applications, enabling you to troubleshoot issues and optimize performance.

=====================================================

**Dynamic Group Concepts in Oracle Cloud Infrastructure (OCI)**

Dynamic Groups in OCI are a powerful feature that allows you to group **compute instances** or **other resources** based on **attributes** rather than manually assigning them. This enables you to define **policies** that grant permissions to resources dynamically, based on their characteristics (e.g., compartment, tags, or resource types). Dynamic Groups are particularly useful for automating access control and managing permissions for workloads running in OCI, such as those in Oracle Kubernetes Engine (OKE).

**1. Creating Dynamic Groups to Control Resource Access Dynamically**

**What is a Dynamic Group?**

A Dynamic Group is a logical grouping of **OCI resources** (e.g., compute instances, functions, or other services) that share common characteristics. Instead of manually adding resources to the group, you define **matching rules** based on attributes such as:

* **Compartment ID**: Resources in a specific compartment.
* **Instance OCIDs**: Specific compute instances.
* **Tags**: Resources with specific tags.
* **Resource types**: All resources of a certain type (e.g., all compute instances).

Once a Dynamic Group is created, you can write **IAM policies** to grant permissions to the group. These policies are automatically applied to any resource that matches the group's rules.

**Steps to Create a Dynamic Group**

1. **Define Matching Rules**:
   * Go to the OCI Console → **Identity & Security** → **Dynamic Groups**.
   * Click **Create Dynamic Group**.
   * Define the matching rules using the following syntax:
     + ALL {resource.type = 'instance', compartment.id = 'ocid1.compartment.oc1..example'}: All compute instances in a specific compartment.
     + ANY {resource.type = 'instance', tag.department.value = 'finance'}: Any compute instance with a specific tag.
     + ALL {resource.type = 'fnfunc', compartment.id = 'ocid1.compartment.oc1..example'}: All functions in a specific compartment.
2. **Name and Description**:
   * Provide a name and description for the Dynamic Group.
3. **Create the Group**:
   * Click **Create** to finalize the Dynamic Group.

**Example Use Case**

Suppose you have a set of compute instances running in a compartment called Prod\_Compartment, and you want to grant them access to read objects in a specific bucket in Object Storage. You can:

1. Create a Dynamic Group with the rule:

Copy

ALL {resource.type = 'instance', compartment.id = 'ocid1.compartment.oc1..example'}

1. Create an IAM policy to grant the Dynamic Group access:

Copy

Allow dynamic-group Prod\_Instances to read objects in compartment Prod\_Compartment where target.bucket.name = 'prod-bucket'

This way, any compute instance in the Prod\_Compartment will automatically have read access to the prod-bucket.

**2. Automating Policies for User and Service Principals in OKE**

Oracle Kubernetes Engine (OKE) is OCI's managed Kubernetes service. When working with OKE, you often need to grant permissions to:

* **User Principals**: Human users or administrators.
* **Service Principals**: Non-human entities like compute instances, functions, or OKE worker nodes.

Dynamic Groups can be used to automate access control for OKE resources, ensuring that the right permissions are granted dynamically based on the workload or environment.

**Steps to Automate Policies for OKE**

1. **Create a Dynamic Group for OKE Worker Nodes**:
   * Define a Dynamic Group for OKE worker nodes using a matching rule like:

Copy

ALL {resource.type = 'instance', compartment.id = 'ocid1.compartment.oc1..example'}

* + This ensures that all worker nodes in the specified compartment are part of the Dynamic Group.

1. **Create IAM Policies for the Dynamic Group**:
   * Write policies to grant the Dynamic Group the necessary permissions for OKE. For example:

Copy

Allow dynamic-group OKE\_Worker\_Nodes to manage cluster-family in compartment OKE\_Compartment

This policy allows the worker nodes to manage Kubernetes clusters.

1. **Grant Permissions to Service Principals**:
   * If you have service principals (e.g., CI/CD pipelines or other automated processes), you can create a Dynamic Group for them and grant permissions like:

Copy

Allow dynamic-group CI\_CD\_Pipelines to use clusters in compartment OKE\_Compartment

1. **Automate User Access**:
   * For human users, you can create IAM policies that grant permissions based on their roles or group memberships. For example:

Copy

Allow group OKE\_Admins to manage cluster-family in compartment OKE\_Compartment

**Example Use Case for OKE**

Suppose you have an OKE cluster running in a compartment called OKE\_Compartment, and you want to:

* Allow worker nodes to pull images from OCI Container Registry.
* Allow a CI/CD pipeline to deploy applications to the cluster.

1. **Dynamic Group for Worker Nodes**:
   * Create a Dynamic Group with the rule:

Copy

ALL {resource.type = 'instance', compartment.id = 'ocid1.compartment.oc1..example'}

* + Grant the group permissions to pull images:

Copy

Allow dynamic-group OKE\_Worker\_Nodes to read repos in compartment OKE\_Compartment

1. **Dynamic Group for CI/CD Pipeline**:
   * Create a Dynamic Group for the CI/CD pipeline (e.g., based on a tag or compartment).
   * Grant the group permissions to deploy to the cluster:

Copy

Allow dynamic-group CI\_CD\_Pipelines to use clusters in compartment OKE\_Compartment

**Benefits of Using Dynamic Groups in OCI**

1. **Automation**: Permissions are automatically applied to resources that match the Dynamic Group rules.
2. **Scalability**: As your environment grows, you don’t need to manually update permissions for new resources.
3. **Security**: Fine-grained access control ensures that only authorized resources can perform specific actions.
4. **Simplified Management**: Reduces the overhead of managing individual permissions for each resource.

**Best Practices**

1. **Use Tags**: Tag resources consistently to make Dynamic Group rules more flexible and scalable.
2. **Least Privilege**: Grant only the minimum permissions required for each Dynamic Group.
3. **Audit Regularly**: Review Dynamic Group rules and policies periodically to ensure they align with your security requirements.
4. **Combine with Compartments**: Use compartments to logically organize resources and simplify Dynamic Group rules.

**Lab 9 : Dynamic Groups and IAM Policies for Oracle Kubernetes Engine (OKE)**

This lab demonstrates how to use **Dynamic Groups** and **IAM Policies** to automate access control for resources in Oracle Kubernetes Engine (OKE). By the end of this lab, you will:

1. Create a Dynamic Group for OKE worker nodes.
2. Create IAM policies to grant permissions to the Dynamic Group.
3. Deploy a sample application on OKE and verify access control.

**Lab Prerequisites**

1. **OCI Account**: Access to an Oracle Cloud Infrastructure (OCI) tenancy.
2. **Permissions**: Ensure you have the necessary permissions to create compartments, Dynamic Groups, IAM policies, and OKE clusters.
3. **OKE Cluster**: An existing OKE cluster or the ability to create one.
4. **OCI CLI**: Installed and configured on your local machine.
5. **kubectl**: Installed and configured to interact with the OKE cluster.

**Lab Steps**

**Step 1: Set Up the Environment**

1. **Create a Compartment** (if not already available):
   * Go to **Identity & Security** → **Compartments** → **Create Compartment**.
   * Name: OKE\_Lab\_Compartment.
   * Description: Compartment for OKE lab resources.
2. **Create an OKE Cluster** (if not already available):
   * Go to **Developer Services** → **Kubernetes Clusters (OKE)** → **Create Cluster**.
   * Choose **Quick Create** and fill in the details:
     + Compartment: OKE\_Lab\_Compartment.
     + Name: OKE\_Lab\_Cluster.
     + Kubernetes Version: Default.
     + Node Shape: VM.Standard.E4.Flex (or any available shape).
     + Number of Nodes: 3.
   * Click **Create**.
3. **Configure kubectl**:
   * After the cluster is created, go to the cluster details page and click **Access Cluster**.
   * Follow the instructions to set up kubectl and connect to the cluster.

**Step 2: Create a Dynamic Group for OKE Worker Nodes**

* Go to [Identity](https://cloud.oracle.com/identity) >> [Domains](https://cloud.oracle.com/identity/domains) >> [Default domain](https://cloud.oracle.com/identity/domains/ocid1.domain.oc1..aaaaaaaajcd3ojubdkqsk4wzhl5anngoocqnajfjdpnfdfnj5xgr77qstanq) >> Dynamic groups

1. Fill in the details:
   * Name: OKE\_Worker\_Nodes.
   * Description: Dynamic Group for OKE worker nodes.
   * Matching Rule:

Copy

ALL {resource.type = 'instance', compartment.id = 'ocid1.compartment.oc1..<your\_compartment\_ocid>'}

Replace <your\_compartment\_ocid> with the OCID of the OKE\_Lab\_Compartment.

**Explanation:**

* ALL: Applies the rule to all instances that match the conditions.
* resource.type = 'instance': Ensures the rule only applies to Compute Instances.
* resource.compartment.id = '<your\_compartment\_ocid>': Restricts instances to a specific compartment.

1. Click **Create**.

**Step 3: Create IAM Policies for the Dynamic Group**

1. Go to **Identity & Security** → **Policies** → **Create Policy**.
2. Fill in the details:
   * Name: OKE\_Worker\_Policies.
   * Description: Policies for OKE worker nodes.
   * Compartment: OKE\_Lab\_Compartment.
   * Policy Statements:

Copy

Allow dynamic-group OKE\_Worker\_Nodes to manage objects in compartment OKE\_Lab\_Compartment

Allow dynamic-group OKE\_Worker\_Nodes to read buckets in compartment OKE\_Lab\_Compartment

1. Click **Create**.

**Step 4: Verify Dynamic Group and Policies**

1. **Verify Dynamic Group**:
   * Go to **Identity & Security** → **Dynamic Groups**.
   * Ensure the OKE\_Worker\_Nodes group is listed and has the correct matching rule.
2. **Verify Policies**:
   * Go to **Identity & Security** → **Policies**.
   * Ensure the OKE\_Worker\_Policies policy is listed and has the correct statements.

**Step 4: Create an Object Storage Bucket**

1. Go to **Storage** → **Buckets** → **Create Bucket**.
2. Fill in the details:
   * Compartment: OKE\_Lab\_Compartment.
   * Name: oke-lab-bucket.
   * Storage Tier: Standard.
3. Click **Create**.

**Step 5: Deploy a Pod to Interact with Object Storage**

1. **Create a Namespace**:
   * Run the following command to create a namespace for the application:

bash

Copy

kubectl create namespace oke-lab

1. **Deploy a Pod with OCI CLI**:
   * Create a file named oci-cli-pod.yaml with the following content:

yaml

Copy

apiVersion: v1

kind: Pod

metadata:

name: oci-cli-pod

namespace: oke-lab

spec:

hostNetwork: true

containers:

- name: oci-cli

image: oraclelinux:8

command: ["sleep", "3600"]

* + Apply the deployment:

bash

Copy

kubectl apply -f oci-cli-pod.yaml

1. **Install OCI CLI in the Pod**:
   * Exec into the pod:

bash

Copy

kubectl exec -it oci-cli-pod -n oke-lab -- /bin/bash

* + Install the OCI CLI:

bash

Copy

yum install -y oraclelinux-developer-release-el8

yum install -y python36-oci-cli

1. **Configure OCI CLI**:
   * Generate an API key in the OCI Console:
     + Go to **Identity & Security** → **Users** → **User Details** → **API Keys** → **Add API Key**.
     + Download the private key and configuration file.

[DEFAULT]

user=ocid1.user.oc1..aaaaaaaatoxqw3xribsf75z5lqps6df37fntyvsgklj6n3mvxcqswdp3qp6q

fingerprint=af:72:90:63:af:57:47:62:52:bd:e0:cd:3f:80:bc:14

tenancy=ocid1.tenancy.oc1..aaaaaaaak2bppm6h6xpghpco4evq7kzyb6z54sa4piforkfto6khbcu7aota

region=us-ashburn-1

key\_file=<path to your private keyfile> # TODO

**Step 2: Copy the .oci Directory to the Pod**

1. **Create the .oci Directory Locally**:
   * On your local machine, create a directory called .oci in your home directory (if it doesn’t already exist):

bash

Copy

mkdir -p ~/.oci

* + Move the private key (oci\_api\_key.pem) and configuration file (config) into the .oci directory:

bash

Copy

mv oci\_api\_key.pem ~/.oci/

mv config ~/.oci/

For reference :

ruser1@cloudshell:.oci (us-ashburn-1)$ ls

config [ruser1@techlanders.com\_2025-02-25T19\_20\_52.296Z.pem](mailto:ruser1@techlanders.com_2025-02-25T19_20_52.296Z.pem)

[root@oke-ctbf3kfangq-npynr7l6q5a-sgaqqrjeafq-0 .oci]# cat config

[DEFAULT]

user=ocid1.user.oc1..aaaaaaaatoxqw3xribsf75z5lqps6df37fntyvsgklj6n3mvxcqswdp3qp6q

fingerprint=af:72:90:63:af:57:47:62:52:bd:e0:cd:3f:80:bc:14

tenancy=ocid1.tenancy.oc1..aaaaaaaak2bppm6h6xpghpco4evq7kzyb6z54sa4piforkfto6khbcu7aota

region=us-ashburn-1

key\_file=/root/.oci/ruser1@techlanders.com\_2025-02-25T19\_20\_52.296Z.pem

1. **Copy the .oci Directory to the Pod**:
   * Use the kubectl cp command to copy the .oci directory from your local machine to the pod:

bash

Copy

kubectl cp ~/.oci oci-cli-pod:/root/.oci -n oke-lab

* + This command copies the .oci directory (containing the config file and private key) to the /root/.oci directory inside the pod.

**Step 3: Verify the Files in the Pod**

1. **Exec into the Pod**:
   * Run the following command to exec into the pod:

bash

Copy

kubectl exec -it oci-cli-pod -n oke-lab -- /bin/bash

1. **Check the .oci Directory**:
   * Verify that the .oci directory and its contents were copied successfully:

bash

Copy

ls -l /root/.oci

* + You should see the config file and the private key file (e.g., oci\_api\_key.pem).
* To fix the perm. On both files , run below :

[root@oke-ctbf3kfangq-npynr7l6q5a-sgaqqrjeafq-0 .oci]# oci setup repair-file-permissions --file /root/.oci/config

[root@oke-ctbf3kfangq-npynr7l6q5a-sgaqqrjeafq-0 .oci]# oci setup repair-file-permissions --file /root/.oci/ruser1@techlanders.com\_2025-02-25T19\_20\_52.296Z.pem

* + Test the OCI CLI:

bash

Copy

oci os bucket list --compartment-id <compartment-ocid>

1. **Interact with Object Storage**:
   * List objects in the bucket:

bash

Copy

oci os object list --bucket-name oke-lab-bucket

* + Upload a file to the bucket:

bash

Copy

echo "Hello, OKE!" > test.txt

oci os object put --bucket-name oke-lab-bucket --file test.txt

**Step 6: Verify Dynamic Group Permissions**

1. **Check Pod Logs**:
   * Verify that the pod can interact with Object Storage without errors.
   * If the Dynamic Group and policies are correctly configured, the pod should be able to list and upload objects to the bucket.
2. **Test Without Permissions**:
   * Remove the IAM policies for the Dynamic Group and verify that the pod can no longer access Object Storage.

**Lab Cleanup**

1. Delete the OKE cluster:
   * Go to **Developer Services** → **Kubernetes Clusters (OKE)**.
   * Select the OKE\_Lab\_Cluster and click \*\*Delete`.
2. Delete the Object Storage bucket:
   * Go to **Storage** → **Buckets**.
   * Select the oke-lab-bucket and click **Delete**.
3. Delete the Dynamic Group and Policies:
   * Go to **Identity & Security** → **Dynamic Groups** and delete OKE\_Worker\_Nodes.
   * Go to **Identity & Security** → **Policies** and delete OKE\_Worker\_Policies.
4. Delete the Compartment:
   * Go to **Identity & Security** → **Compartments** and delete OKE\_Lab\_Compartment.

**Conclusion**

This revised lab demonstrates how to **test Dynamic Groups without using OCIR**. By using OCI Object Storage as the target resource, you can verify that the Dynamic Group and IAM policies are working as expected. This approach avoids the need to create a container registry and simplifies the lab setup.

**Add-ons for OKE Clusters: Enhanced Functionality and Monitoring**

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. To further enhance the functionality and monitoring capabilities of OKE clusters, you can install various **cluster add-ons**. These add-ons are additional tools, services, or plugins that integrate with your OKE cluster to provide advanced features, improve observability, and streamline operations.

**Key Benefits of Cluster Add-ons:**

1. **Enhanced Functionality**: Add-ons extend the capabilities of your OKE cluster, enabling features like logging, monitoring, security, networking, and more.
2. **Improved Monitoring and Observability**: Add-ons provide insights into cluster performance, resource utilization, and application health.
3. **Simplified Operations**: Pre-configured add-ons reduce the effort required to set up and manage additional tools.
4. **Security and Compliance**: Add-ons can help enforce security policies, manage secrets, and ensure compliance with industry standards.

**Common Types of Add-ons for OKE Clusters:**

**1. Monitoring and Observability Add-ons**

* **OCI Logging and Monitoring**: Integrate OKE with OCI's native logging and monitoring services to collect and analyze logs, metrics, and traces from your cluster and applications.
* **Prometheus and Grafana**: Deploy Prometheus for metrics collection and Grafana for visualization and dashboards.
* **OpenTelemetry**: Enable distributed tracing for microservices-based applications.

**2. Networking Add-ons**

* **OCI Load Balancer**: Automatically provision load balancers for Kubernetes services.
* **Ingress Controllers**: Use NGINX, Traefik, or other ingress controllers to manage external access to your applications.
* **CNI Plugins**: Enhance networking capabilities with plugins like Calico or Cilium for advanced network policies and security.

**3. Security Add-ons**

* **OCI Vault**: Integrate with OCI Vault for managing secrets, encryption keys, and certificates.
* **Kubernetes RBAC**: Enhance role-based access control (RBAC) for fine-grained permissions.
* **Pod Security Policies (PSP)**: Enforce security policies for pods running in the cluster.

**4. Storage Add-ons**

* **OCI Block Volume**: Dynamically provision persistent storage for stateful applications.
* **CSI Drivers**: Use Container Storage Interface (CSI) drivers to integrate with OCI storage services.

**5. CI/CD and Automation Add-ons**

* **Jenkins or GitLab Runners**: Set up CI/CD pipelines for automated application deployment.
* **ArgoCD**: Implement GitOps workflows for continuous delivery.

**6. Service Mesh Add-ons**

* **Istio**: Deploy a service mesh for advanced traffic management, security, and observability.
* **Linkerd**: Use a lightweight service mesh for microservices communication.

**7. Backup and Disaster Recovery Add-ons**

* **Velero**: Backup and restore Kubernetes cluster resources and persistent volumes.
* **OCI Object Storage**: Store backups in OCI Object Storage for durability and scalability.

**How to Install Add-ons in OKE Clusters:**

1. **OCI Console**: Use the OCI console to enable pre-configured add-ons like OCI Logging and Monitoring.
2. **Helm Charts**: Many add-ons can be installed using Helm, a package manager for Kubernetes. For example:

bash

Copy

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm install prometheus prometheus-community/prometheus

1. **Manual Deployment**: Deploy add-ons using Kubernetes manifests (YAML files) for custom configurations.
2. **OCI Marketplace**: Explore and deploy third-party add-ons available in the OCI Marketplace.

**Best Practices for Using Add-ons:**

* **Evaluate Requirements**: Choose add-ons based on your specific use case, such as monitoring, security, or networking.
* **Keep Add-ons Updated**: Regularly update add-ons to ensure compatibility with your OKE cluster and Kubernetes version.
* **Monitor Resource Usage**: Some add-ons may consume cluster resources, so monitor their impact on performance.
* **Follow Security Best Practices**: Ensure add-ons are configured securely to avoid vulnerabilities.

By leveraging these add-ons, you can significantly enhance the functionality, monitoring, and management of your OKE clusters, making them more robust, secure, and efficient for running containerized workloads.

**Lab 10 : Kubernetes dashboard add on**

* + **Add the Kubernetes dashboard add on in the cluster**
  + **Expose the dashboard service to load balancer**
  + **Retrieve the Token for Authentication**

The Dashboard will prompt you for a token. To retrieve the token, follow these steps:

1. **Create a Service Account** (if not already created):  
   Create a Service Account with the necessary permissions to access the Dashboard. For example:

yaml

Copy

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

Apply the manifest:

bash

Copy

kubectl apply -f service-account.yaml

1. **Bind the Service Account to a ClusterRole**:  
   Bind the Service Account to a ClusterRole with administrative privileges:

yaml

Copy

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: admin-user

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: admin-user

Apply the manifest:

bash

Copy

kubectl apply -f cluster-role-binding.yaml

1. **Retrieve the Token**:  
   Get the token for the admin-user Service Account:

bash

Copy

kubectl create token admin-user

This command will output a token. Copy the token.

* + **Expose the token on the Kubernetes dashboard and access it**