**Load Balancer Integration with Oracle Kubernetes Engine (OKE)**

**1. Introduction to OCI Load Balancers**

Oracle Cloud Infrastructure (OCI) provides **Load Balancer as a Service (LBaaS)** to distribute traffic efficiently across backend servers or Kubernetes pods in an **Oracle Kubernetes Engine (OKE) cluster**. OCI Load Balancers operate at **two levels**:

1. **Layer 7 (Application Load Balancer - ALB)**
   * Works at the **application layer** (HTTP/HTTPS).
   * Supports **content-based routing** (e.g., route traffic based on URL path or host header).
   * Can perform **SSL termination**, meaning it handles HTTPS traffic and forwards plain HTTP to backend services.
   * Supports **Web Application Firewall (WAF)** integration.
2. **Layer 4 (Network Load Balancer - NLB)**
   * Works at the **transport layer** (TCP/UDP).
   * Supports **low-latency traffic forwarding**.
   * Suitable for applications that require **high performance** and do not need HTTP-specific features.
   * Does **not** perform SSL termination or content-based routing.
   * Can distribute traffic **directly to pods** in an OKE cluster.

**2. Configuring Load Balancers to Route Traffic in OKE**

To integrate OCI Load Balancers with **OKE**, follow these steps:

**A. Deploy an OKE Cluster**

* Ensure you have an **OKE cluster running** with worker nodes.
* Nodes should have the **OCI Cloud Controller Manager (CCM)** and the **OCI Load Balancer Controller** enabled.

**B. Create a Kubernetes Service of Type LoadBalancer**

OCI Load Balancers are automatically provisioned when a Kubernetes **Service** with type: LoadBalancer is created.

1. **Define a Service Manifest (service.yaml)**:

apiVersion: v1

kind: Service

metadata:

name: my-loadbalancer

annotations:

service.beta.kubernetes.io/oci-load-balancer-shape: "flexible"

service.beta.kubernetes.io/oci-load-balancer-internal: "true" # Set false for external

spec:

type: LoadBalancer

selector:

app: my-app

ports:

- protocol: TCP

port: 80

targetPort: 8080

1. **Apply the service to Kubernetes**:

bash

CopyEdit

kubectl apply -f service.yaml

1. **Verify the Load Balancer creation**:

bash

CopyEdit

kubectl get svc my-loadbalancer

This command will display the **external IP** assigned by the OCI Load Balancer.

**C. Load Balancer Annotations for Custom Configurations**

OCI supports annotations to customize Load Balancer behavior:

| **Annotation** | **Description** |
| --- | --- |
| service.beta.kubernetes.io/oci-load-balancer-shape | Defines Load Balancer shape (e.g., flexible, 100Mbps, 400Mbps). |
| service.beta.kubernetes.io/oci-load-balancer-internal | Set true for internal, false for public LB. |
| service.beta.kubernetes.io/oci-load-balancer-ssl-ports | Defines SSL termination for specific ports. |
| service.beta.kubernetes.io/oci-load-balancer-policy | Sets LB routing policy (e.g., ROUND\_ROBIN, LEAST\_CONNECTIONS). |

**D. Exposing Services via an Ingress Controller**

For **advanced Layer 7 routing**, an **Ingress Controller** (e.g., NGINX Ingress Controller, Traefik) can be used.

1. **Deploy an Ingress Controller**:

bash

CopyEdit

helm install nginx-ingress ingress-nginx/ingress-nginx

1. **Create an Ingress Resource (ingress.yaml)**:

yaml

CopyEdit

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: my-ingress

spec:

rules:

- host: myapp.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: my-loadbalancer

port:

number: 80

1. **Apply the Ingress Resource**:

bash

CopyEdit

kubectl apply -f ingress.yaml

1. **Verify the Ingress Configuration**:

bash

CopyEdit

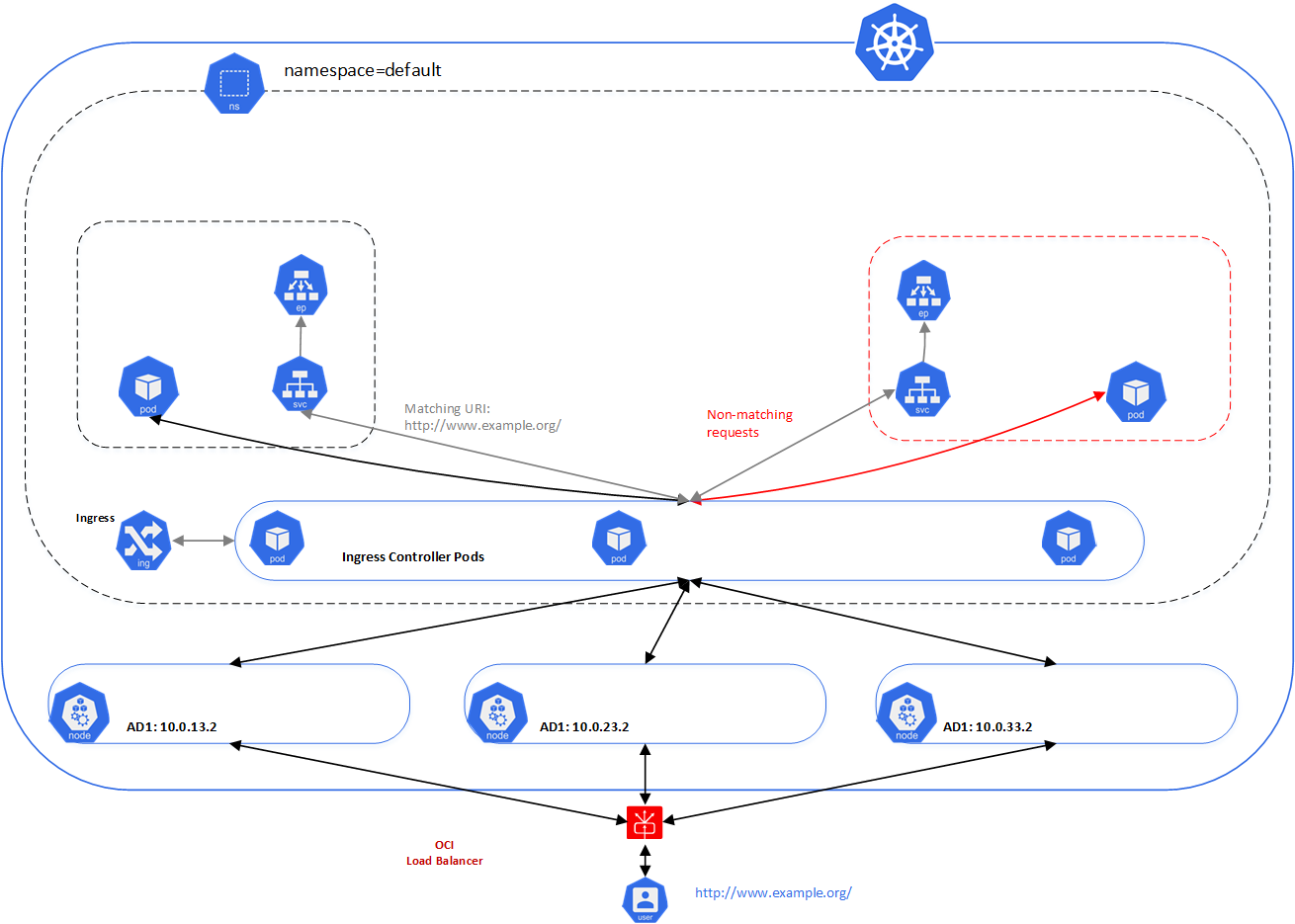
kubectl get ingress

**E. Load Balancer Security & Scaling**

1. **Security:**
   * **Network Security Groups (NSGs)** can control traffic to and from the load balancer.
   * **SSL Termination** should be configured for secure communication.
   * **OCI Web Application Firewall (WAF)** can be integrated for additional security.
2. **Scaling:**
   * OCI Load Balancers automatically scale **up to 8,000 concurrent connections per instance**.
   * Use **Flexible Load Balancer Shape** to dynamically allocate bandwidth.

**Conclusion**

* OCI Load Balancers provide **Layer 7 (ALB) and Layer 4 (NLB)** traffic management.
* Kubernetes **Service of type LoadBalancer** automatically provisions an OCI Load Balancer.
* **Annotations** help customize load balancer behavior.
* **Ingress Controllers** enable advanced routing for multiple services.
* **Security and scaling** should be considered to optimize performance.



**Lab 3 : Integrating load balancer**

**Step 1: Deploy NGINX and HTTPD Applications**

Follow the same steps as in the original lab to create the NGINX and HTTPD deployments and services.

**Step 2: Deploy the NGINX Ingress Controller**

**30 k create ns ingress**

**32 helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx**

**33 helm repo update**

1. **Install the NGINX Ingress Controller using Helm:**  
   Use the following command to install the NGINX Ingress Controller and configure it to automatically create an OCI Load Balancer:

helm install my-release ingress-nginx/ingress-nginx \

--namespace ingress \

--set controller.replicaCount=2 \

The controller.service.type=LoadBalancer flag tells Kubernetes to create a Service of type LoadBalancer, which will automatically provision an OCI Load Balancer.

1. **Verify the Ingress Controller and Load Balancer:**  
   Check the status of the Ingress Controller and the Load Balancer:

bash

Copy

kubectl get svc ingress-nginx-controller

You should see an output like this:

Copy

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

ingress-nginx-controller LoadBalancer 10.96.XXX.XXX <OCI-LB-IP> 80:XXXXX/TCP,443:XXXXX/TCP 1m

The EXTERNAL-IP column will show the public IP address of the OCI Load Balancer that was automatically created.

**Step 3: Create the two depployments , services and an Ingress Resource**

Create the Ingress resource as before to route traffic to the NGINX and HTTPD services.

Save the following YAML as ingress.yaml:

root@master:~/raman# cat ingdeploy.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: test-app

labels:

app: test-app

spec:

replicas: 3

selector:

matchLabels:

app: test-app

template:

metadata:

labels:

app: test-app

spec:

containers:

- name: test-app

image: nginx:latest

ports:

- containerPort: 80

resources:

limits:

cpu: 100m

memory: 128Mi

requests:

cpu: 50m

memory: 64Mi

---

apiVersion: v1

kind: Service

metadata:

name: raman-service

spec:

type: NodePort

ports:

- port: 80

selector:

app: test-app

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: test-app2

labels:

app: test-app2

spec:

replicas: 3

selector:

matchLabels:

app: test-app2

template:

metadata:

labels:

app: test-app2

spec:

containers:

- name: test-app2

image: httpd

ports:

- containerPort: 80

resources:

limits:

cpu: 100m

memory: 128Mi

requests:

cpu: 50m

memory: 64Mi

---

apiVersion: v1

kind: Service

metadata:

name: raman-service2

spec:

type: NodePort

ports:

- port: 80

selector:

app: test-app2

---

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: test-app-ingress

namespace: ingress

annotations:

nginx.ingress.kubernetes.io/ssl-redirect: "false"

nginx.ingress.kubernetes.io/use-regex: "true"

nginx.ingress.kubernetes.io/rewrite-target: /$2

spec:

ingressClassName: nginx

rules:

- http:

paths:

- path: /raman(/|$)(.\*)

pathType: ImplementationSpecific

backend:

service:

name: raman-service

port:

number: 80

- http:

paths:

- path: /raman2(/|$)(.\*)

pathType: ImplementationSpecific

backend:

service:

name: raman-service2

port:

number: 80

* K apply -f deployments.yaml -n ingress

**Step 5: Test the Setup**

1. **Access the NGINX Application:**  
   Open a browser and navigate to ExternalIpofmy-release-ingress-nginx-controller/raman You should see the NGINX welcome page.
2. **Access the HTTPD Application:**  
   Open a browser and navigate to ExternalIpofmy-release-ingress-nginx-controller/raman2 You should see the Apache HTTPD welcome page.

**Step 6: Clean Up**

1. Delete the Ingress:

bash

Copy

kubectl delete -f ingress.yaml

1. Delete the Deployments and Services:

bash

Copy

kubectl delete -f nginx-deployment.yaml

kubectl delete -f httpd-deployment.yaml

1. Uninstall the NGINX Ingress Controller:

bash

Copy

helm uninstall ingress-nginx

This will automatically delete the OCI Load Balancer that was provisioned.

**Key Points**

* By setting the Service type to LoadBalancer, Kubernetes automatically provisions an OCI Load Balancer.
* This approach is fully automated and eliminates the need for manual intervention.
* The OCI Load Balancer is managed by Kubernetes, so it will be deleted when the Ingress Controller is uninstalled.

**Storage Concepts in Oracle Kubernetes Engine (OKE)**

Oracle Kubernetes Engine (OKE) provides multiple storage options to manage and persist data for containerized applications. In Kubernetes, storage solutions can be categorized as ephemeral (temporary) or persistent (durable across pod restarts and rescheduling). OKE integrates with Oracle Cloud Infrastructure (OCI) storage services to provide robust, scalable, and managed storage solutions.

**1. Understanding Persistent Storage in Kubernetes**

Kubernetes offers persistent storage to ensure that application data is retained even when containers or pods are restarted or rescheduled.

**Persistent Volumes (PVs) and Persistent Volume Claims (PVCs)**

* **Persistent Volume (PV):** A cluster-wide storage resource provisioned by an administrator or dynamically by the storage class.
* **Persistent Volume Claim (PVC):** A request for storage by a pod, which is bound to an available PV based on size, access mode, and other constraints.
* **Storage Classes:** Define different types of storage (e.g., block, file, or object) with different characteristics like performance, replication, and retention policies.

**Types of Storage in Kubernetes (OKE)**

OKE supports multiple storage options:

1. **OCI Block Storage (for PVs)**
   * Best for stateful applications like databases (e.g., MySQL, PostgreSQL).
   * Can be dynamically provisioned with Storage Classes.
   * Supports ReadWriteOnce (RWO) access mode (single-node writable).
2. **OCI File Storage Service (FSS)**
   * Network File System (NFS)-based shared storage.
   * Used for applications requiring shared access among multiple pods.
   * Supports ReadWriteMany (RWX) access mode (multi-node writable).
3. **OCI Object Storage**
   * Not directly mountable but useful for storing application backups and logs.

**2. Integrating with File Storage Service (FSS) for Shared Storage**

OCI **File Storage Service (FSS)** is a fully managed, elastic NFS file system that allows multiple OKE pods to share the same storage. It is ideal for workloads that require shared access to files, such as web applications, content management systems, and machine learning models.

**Benefits of Using FSS in OKE**

* **High Availability & Durability:** Data is replicated across multiple availability domains.
* **Scalability:** Auto-scales up to petabytes of storage.
* **Multi-Pod Access:** Multiple Kubernetes pods can read and write to the same file system simultaneously.
* **Backup & Restore Capabilities:** OCI provides automated snapshots for data protection.

**Steps to Integrate OCI FSS with OKE**

1. **Create an FSS File System in OCI**
   * Navigate to **OCI Console → File Storage → Create File System**
   * Assign it to a mount target and security list.
2. **Install and Configure OCI FSS CSI Driver on OKE**
   * Oracle provides a **Container Storage Interface (CSI) driver** for mounting FSS in Kubernetes.
   * Install the OCI FSS CSI driver using Helm:

sh

CopyEdit

helm repo add oracle https://oracle.github.io/helm-charts

helm install oci-fss-csi oracle/oci-fss-csi-driver

1. **Define a StorageClass for OCI FSS in Kubernetes**

yaml

CopyEdit

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: oci-fss

provisioner: fss.csi.oraclecloud.com

1. **Create a Persistent Volume (PV) for FSS**

yaml

CopyEdit

apiVersion: v1

kind: PersistentVolume

metadata:

name: fss-pv

spec:

capacity:

storage: 100Gi

accessModes:

- ReadWriteMany

nfs:

server: <FSS-IP-ADDRESS>

path: "/"

1. **Create a Persistent Volume Claim (PVC)**

yaml

CopyEdit

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: fss-pvc

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 100Gi

storageClassName: oci-fss

1. **Mount the PVC in a Kubernetes Deployment**

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: web-app

spec:

replicas: 2

selector:

matchLabels:

app: web

template:

metadata:

labels:

app: web

spec:

containers:

- name: web-container

image: nginx

volumeMounts:

- mountPath: "/usr/share/nginx/html"

name: fss-volume

volumes:

- name: fss-volume

persistentVolumeClaim:

claimName: fss-pvc

Once applied, all web server pods will share the same storage across different nodes.

**3. Managed Backups for OKE in OCI**

Backup and disaster recovery are crucial for stateful applications running in OKE. OCI provides **automated backup and restore** options to protect critical data.

**Backup Strategies in OKE**

1. **Backing up Persistent Volume Data**
   * **Block Storage Backups:** OCI allows automatic scheduled snapshots of block storage volumes (PVs).
   * **FSS Snapshots:** OCI File Storage supports point-in-time snapshots for shared storage.
2. **Backing up Kubernetes Cluster State**
   * Use **Velero**, an open-source Kubernetes backup tool.
   * It captures and restores Kubernetes objects, including PersistentVolumeClaims (PVCs).
   * Install Velero with the OCI plugin:

sh

CopyEdit

helm install velero vmware-tanzu/velero \

--set configuration.provider=aws \

--set configuration.backupStorageLocation.bucket=<OCI-BUCKET> \

--set credentials.secretContents.cloud=~/.oci/config

1. **Automating Kubernetes Backups in OCI**
   * **Use OCI Backup Policies:** Define policies for daily, weekly, or monthly backups.
   * **Use OCI Object Storage:** Store application configuration, logs, and database dumps.
   * **Enable OCI Vault for Encryption:** Secure backup data using OCI Vault keys.

**Restoring from a Backup**

* **For Block Storage:** Restore a backup to a new volume and attach it to a Kubernetes node.
* **For Kubernetes Cluster State:** Use Velero to restore cluster objects.

sh

CopyEdit

velero restore create --from-backup my-cluster-backup

**Conclusion**

* **Persistent storage in OKE** is essential for stateful applications and is managed using Persistent Volumes (PV) and Persistent Volume Claims (PVC).
* **OCI File Storage Service (FSS)** is a shared, scalable NFS solution that integrates well with OKE for multi-pod access.
* **Managed Backups** in OCI help automate data protection through snapshots, Velero, and object storage.
* **Automating storage and backup policies** enhances security, reliability, and disaster recovery for Kubernetes workloads in OCI.

**Lab 4 : Integ. with FSS**

1. **Integrate with File Storage Service (FSS)**: Set up a shared storage solution for OKE using OCI’s File Storage Service.
2. **Implement Managed Backups for OKE**: Automate the backup process for critical data in your OKE clusters.

**Prerequisites**

* An Oracle Cloud account with necessary permissions to create resources.
* Basic knowledge of Kubernetes and OCI.
* OCI CLI installed and configured on your local machine.
* Access to the OCI Console.

**Lab Setup**

**Part 1: Integrating with File Storage Service (FSS)**

1. **Create a File System in FSS**
   * Navigate to the OCI Console.
   * Go to the **File Storage** section.
   * Click on File System for NFS
   * Choose a name and compartment for the file system.
   * Click on edit details on Mount target information
   * Select the region in which ur Kubernetes cluster is situated and subnet : oke-nodesubnet-quick\*

NOTE : MAKE SURE THE MOINT TARGET IS IN THE SAME SUBNET AND REGION AS UR WORKER NODES

* + Choose a subnet for the mount target and click **Create**.

1. **Configure Network Security**
   * Ensure that the security list or network security group for the subnet allows traffic on NFS (port 2049).
2. **Install NFS Client on OKE Nodes**
   * Access your both OKE cluster nodes.

HP@DESKTOP-3G12VKI MINGW64 ~/Downloads

$ ssh -i ssh-key-2025-02-24.key [opc@129.80.184.27](mailto:opc@129.80.184.27)

* Ssh on both cluster nodes

Got o file system details and see the mount commands to test the mount points

* [File Storage](https://cloud.oracle.com/fss/file-systems) >> [File Systems](https://cloud.oracle.com/fss/file-systems) >> [File System details](https://cloud.oracle.com/fss/file-systems/ocid1.filesystem.oc1.iad.aaaaaaaaaaky4avknfqwillqojxwiotjmfsc2ylefuzqaaaa/exports)
* Run below commands on all worker nodes in the cluster tocheck the file system mounting :

sudo yum install nfs-utils

sudo mkdir -p /mnt/FileSystem-20250224-1627-29

sudo mount 10.0.10.225:/FileSystem-20250224-1627-29 /mnt/FileSystem-20250224-1627-29

* Test the mount points by creating test files on both worker nodes

1. **Create a Persistent Volume (PV) and Persistent Volume Claim (PVC)**
   * Create a YAML file named fss-pv.yaml for the PV:

apiVersion: v1

kind: PersistentVolume

metadata:

name: fss-pv

spec:

capacity:

storage: 2Gi

accessModes:

- ReadWriteMany

nfs:

path: /path/to/mount/target

server: <mount-target-ip>

ruser1@cloudshell:~ (us-ashburn-1)$ cat pv.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: fss-pv

spec:

capacity:

storage: 2Gi

accessModes:

- ReadWriteMany

nfs:

path: /FileSystem-20250224-1536-23

server: 10.0.10.6

* + Create a YAML file named fss-pvc.yaml for the PVC:

ruser1@cloudshell:~ (us-ashburn-1)$ cat pvc.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: fss-pvc

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 2Gi

storageClassName: ""

* + Apply the PV and PVC:

kubectl apply -f fss-pv.yaml

kubectl apply -f fss-pvc.yaml

* Here’s an example of how to create a deplooymnet that uses the PVC:

ruser1@cloudshell:~ (us-ashburn-1)$ cat deploy.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: test-deployment

spec:

replicas: 3

selector:

matchLabels:

app: test-app

template:

metadata:

labels:

app: test-app

spec:

containers:

- name: test-container

image: nginx

volumeMounts:

- mountPath: "/mnt/data"

name: test-volume

volumes:

- name: test-volume

persistentVolumeClaim:

claimName: fss-pvc

**Steps to Apply the Deployment:**

1. Save the above YAML to a file, e.g., deployment.yaml.
2. Apply the Deployment using kubectl:

bash

Copy

kubectl apply -f deployment.yaml

1. Verify the Deployment, Pod, PVC, and PV status:

bash

Copy

kubectl get deployment

kubectl get pods

kubectl get pvc

kubectl get pv

kubectl exec -it deploymentpod1 -- /bin/bash

kubectl exec -it deploymentpod2 -- /bin/bash

**Expected Behavior:**

* The Deployment will create a Pod that uses the PVC (fss-pvc).
* Once the Pod is scheduled, the PVC will bind to the PV (fss-pv), and the PVC status will change to Bound.
* The PV will also reflect the claim in its CLAIM column.

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

**Part 2 (not tested yet ): Implementing Backups for OKE (Oracle Kubernetes Engine) using velero..**

**Step 1: Set Up OCI Object Storage for Backups**

1. **Create an OCI Object Storage Bucket:**

BUCKET\_NAME=velero-backups

1. **Generate an API Key for Velero Authentication:**

mkdir -p ~/.oci

openssl genrsa -out ~/.oci/oci\_api\_key.pem 2048

openssl rsa -pubout -in ~/.oci/oci\_api\_key.pem -out ~/.oci/oci\_api\_key\_public.pem

1. **Get the API Key Fingerprint:**

bash

CopyEdit

openssl rsa -in ~/.oci/oci\_api\_key.pem -pubout -outform DER | openssl md5 -c

1. **Add API Key to OCI User:**
   * Navigate to **OCI Console** → **Identity** → **Users** → (Your User).
   * Under **API Keys**, upload the oci\_api\_key\_public.pem.

**Step 2: Create IAM Policy for Velero**

1. **Navigate to Policies:**
   * In the OCI Console, go to **Identity** → **Policies**.
2. **Create a New Policy:**
   * Click on **Create Policy**.
   * Provide a **Name** (e.g., VeleroBackupPolicy).
   * Set the **Compartment** to the same compartment where your bucket is located.
3. **Define the Policy Statements:**
   * Add the following policy statements to grant access to the bucket:

Allow group Administrators to manage object-family in tenancy

Allow group Administrators to manage object-family in tenancy

* Replace <Your\_User\_Group> /user with the appropriate user group name.

1. **Click Create:**
   * Save the policy.

**Step 3: Install Velero on OKE**

1. **Install Velero CLI:**

curl -LO <https://github.com/vmware-tanzu/velero/releases/download/v1.10.1/velero-v1.10.1-linux-amd64.tar.gz>

tar zxvf velero-v1.10.1-linux-amd64.tar.gz

sudo mv velero /usr/local/bin/

1. **Deploy Velero Using Helm:**

helm repo add vmware-tanzu https://vmware-tanzu.github.io/helm-charts

helm repo update

helm install velero vmware-tanzu/velero \

--namespace velero --create-namespace \

--set configuration.provider=oci \

--set configuration.backupStorageLocation.name=default \

--set configuration.backupStorageLocation.bucket=velero-backups \

--set configuration.backupStorageLocation.config.region=us-ashburn-1 \

--set credentials.secretContents.cloud="$(base64 ~/.oci/ruser1@techlanders.com\_2025-02-25T19\_20\_52.296Z.pem)"

1. **Verify Velero Installation:**

kubectl get pods -n velero

Ensure the Velero pod is running.

**Step 4: Perform a Kubernetes Backup**

1. **Deploy an Example Application:**

bash

CopyEdit

cat <<EOF | kubectl apply -f -

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

EOF

1. **Verify Deployment:**

bash

CopyEdit

kubectl get deployments

1. **Create a Backup:**

bash

CopyEdit

velero backup create nginx-backup --include-namespaces default

1. **Verify Backup:**

bash

CopyEdit

velero backup get

**Step 5: Restore Kubernetes Resources**

1. **Delete Deployment:**

bash

CopyEdit

kubectl delete deployment nginx-deployment

1. **Restore from Backup:**

bash

CopyEdit

velero restore create --from-backup nginx-backup

1. **Verify Restore:**

bash

CopyEdit

kubectl get deployments

**Notes**

* By using your user credentials with the appropriate policies, you can perform backup and restore operations without creating a dynamic group.
* Ensure that your user account has the necessary permissions to manage the resources in OCI.
* Test your backup and restore processes to confirm everything is working as expected.