----

activation :

account : activate ur profile and reset the password ...

---- loggin in :

    techlanders : acocunt name

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

**Overview of Kubernetes and OKE Cluster Concepts**

**Key Features of Kubernetes and Its Orchestration Benefits**

Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. It provides a robust framework for running distributed systems resiliently, handling scaling and failover, and offering deployment patterns and tools for managing application workloads.

**Key Features of Kubernetes:**

2. **Automated Scheduling:**

* Kubernetes automatically      schedules containers based on resource requirements and constraints,      ensuring optimal utilization of cluster resources.

1. **Self-Healing:**

* Kubernetes automatically      restarts failed containers, replaces and reschedules containers when      nodes die, and kills containers that don’t respond to user-defined health      checks.

1. **Horizontal Scaling:**

* Applications can be scaled up or      down manually or automatically based on CPU usage or other select      metrics.

1. **Service Discovery and Load     Balancing:**

* Kubernetes can expose a      container using the DNS name or using their own IP address. If traffic to      a container is high, Kubernetes is able to load balance and distribute      the network traffic so that the deployment is stable.

1. **Automated Rollouts and Rollbacks:**

* Kubernetes progressively rolls      out changes to your application or its configuration, while monitoring      application health to ensure it doesn’t kill all your instances at the      same time. If something goes wrong, Kubernetes will rollback the change      for you.

1. **Storage Orchestration:**

* Kubernetes allows you to      automatically mount a storage system of your choice, such as local      storages, public cloud providers, and more.

1. **Secret and Configuration     Management:**

* Kubernetes lets you store and      manage sensitive information, such as passwords, OAuth tokens, and SSH      keys. You can deploy and update secrets and application configuration      without rebuilding your container images, and without exposing secrets in      your stack configuration.

**Orchestration Benefits:**

* **Efficiency:** Kubernetes optimizes the     use of hardware resources, reducing costs and improving performance.
* **Portability:** Applications can be moved     across different environments (on-premises, hybrid, or public cloud)     without changes.
* **Scalability:** Kubernetes can handle the scaling     of applications seamlessly, ensuring that they can meet demand without     manual intervention.
* **Reliability:** With features like     self-healing and automated rollouts, Kubernetes ensures that applications     are highly available and resilient to failures.

**OKE as a Fully Managed Kubernetes Service on Oracle Cloud Infrastructure (OCI)**

Oracle Kubernetes Engine (OKE) is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications to the cloud. OKE leverages the power of Kubernetes while removing the operational burden of managing the control plane and infrastructure.

**Key Features of OKE:**

2. **Fully Managed Control Plane:**

* Oracle manages the Kubernetes      control plane, including the API server, etcd, scheduler, and controller      manager, ensuring high availability and security.

1. **Integrated with OCI Services:**

* OKE is deeply integrated with      other OCI services such as Oracle Cloud Infrastructure Registry (OCIR),      Oracle Cloud Infrastructure Monitoring, and Oracle Cloud Infrastructure      Logging, providing a seamless experience for deploying and managing      applications.

1. **Simplified Cluster Management:**

* OKE provides a simple and      intuitive interface for creating, scaling, and managing Kubernetes      clusters. You can create a cluster with just a few clicks or using the      OCI CLI and SDKs.

1. **High Availability and     Reliability:**

* OKE ensures high availability by      distributing the control plane and worker nodes across multiple      availability domains within a region.

1. **Security:**

* OKE provides robust security      features, including network policies, IAM integration, and encryption at      rest and in transit. It also supports private clusters, where the control      plane and worker nodes are not exposed to the public internet.

1. **Cost Efficiency:**

* With OKE, you only pay for the      compute, storage, and networking resources that your applications use.      There are no additional charges for the Kubernetes control plane.

1. **Flexibility and Portability:**

* OKE supports both Linux and      Windows containers, and you can use any Kubernetes-compatible tool or      application. This ensures that your applications are portable across      different environments.

**Benefits of Using OKE:**

* **Reduced Operational Overhead:** By offloading the     management of the Kubernetes control plane to Oracle, you can focus on developing     and deploying your applications.
* **Scalability:** OKE allows you to easily     scale your applications up or down based on demand, ensuring that you only     use the resources you need.
* **Integration with OCI Ecosystem:** OKE’s deep integration with     other OCI services provides a comprehensive solution for building,     deploying, and managing cloud-native applications.
* **Enterprise-Grade Security:** OKE provides     enterprise-grade security features, ensuring that your applications and     data are protected.

In summary, Kubernetes is a powerful platform for managing containerized applications, and OKE simplifies the use of Kubernetes on Oracle Cloud Infrastructure by providing a fully managed, secure, and scalable service. This allows developers to focus on building and deploying applications without worrying about the underlying infrastructure.

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

 ways :

 gui , cli , api/sdk , terraform . ansible

 ---- cluster creation : quick create clsuetr ....

     --- managed ..

--- under advanced :

    --- ssh >> generate an ssh keypair

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

    ---- oci cli installation

<https://github.com/oracle/oci-cli/releases>

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

**Authentication and Authorization in Oracle Kubernetes Engine (OKE)**

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. A critical aspect of managing Kubernetes clusters is ensuring secure access to the cluster and its resources. This is achieved through **authentication** and **authorization** mechanisms. Below, we will explore these concepts in detail, focusing on **user authentication**, **cluster access control**, and **Role-Based Access Control (RBAC)** in Kubernetes clusters.

**1. Authentication in OKE**

Authentication is the process of verifying the identity of a user, service account, or system component attempting to access the Kubernetes cluster. In OKE, authentication ensures that only legitimate users and entities can interact with the cluster.

**Key Concepts in Authentication:**

* **Users**: These are typically human users (e.g., developers, administrators)     who interact with the Kubernetes cluster.
* **Service Accounts**: These are non-human entities     (e.g., applications, pods) that require access to the Kubernetes API.
* **Authentication Methods**: Kubernetes supports multiple     authentication mechanisms, and OKE integrates with OCI Identity and Access     Management (IAM) for seamless authentication.

**Authentication Methods in OKE:**

2. **OCI IAM Integration**:

* OKE leverages OCI IAM for user      authentication. Users are authenticated using their OCI credentials      (username/password, API keys, or federated identity).
* OCI IAM provides Single Sign-On      (SSO) capabilities, enabling users to access OKE clusters without      managing separate credentials.
* Users are mapped to Kubernetes      roles via OCI IAM policies.

1. **Kubernetes Service Accounts**:

* Service accounts are used by      pods and applications to authenticate with the Kubernetes API.
* Each namespace in a Kubernetes      cluster has a default service account, and custom service accounts can be      created as needed.
* Service accounts are associated      with tokens (stored as Kubernetes secrets) that are used for      authentication.

1. **X.509 Client Certificates**:

* Kubernetes supports client      certificate-based authentication. Users can present a valid X.509      certificate to authenticate themselves.
* This method is less common in      OKE due to the integration with OCI IAM.

1. **Static Token Files**:

* Kubernetes can authenticate      users using static token files. However, this method is not recommended      for production environments due to security concerns.

1. **OpenID Connect (OIDC)**:

* OKE can integrate with      OIDC-compliant identity providers (e.g., Google, Azure AD) for      authentication.
* This method is useful for organizations      that use third-party identity providers.

**2. Authorization in OKE**

Authorization determines what actions an authenticated user or service account can perform within the Kubernetes cluster. In OKE, authorization is primarily managed using **Role-Based Access Control (RBAC)**.

**Key Concepts in Authorization:**

* **Roles**: Define a set of permissions (e.g., create, read, update, delete)     for resources within a specific namespace.
* **ClusterRoles**: Similar to Roles but apply to     cluster-wide resources or across all namespaces.
* **RoleBindings**: Associate a Role with a user,     group, or service account within a specific namespace.
* **ClusterRoleBindings**: Associate a ClusterRole with a     user, group, or service account across the entire cluster.

**Role-Based Access Control (RBAC) in OKE:**

RBAC is the standard method for managing permissions in Kubernetes clusters, including OKE. It provides fine-grained control over who can perform specific actions on specific resources.

**How RBAC Works:**

2. **Define Roles and ClusterRoles**:

* Roles and ClusterRoles specify      the permissions (verbs) that can be performed on resources (e.g., pods,      services, deployments).
* Example of a Role:

yaml

Copy

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

  namespace: default

  name: pod-reader

rules:

- apiGroups: [""]

  resources: ["pods"]

  verbs: ["get", "watch", "list"]



* Example of a ClusterRole:

yaml

Copy

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

  name: cluster-admin

rules:

- apiGroups: [""]

  resources: ["\*"]

  verbs: ["\*"]

2. **Bind Roles to Users or Groups**:

* RoleBindings and      ClusterRoleBindings associate Roles or ClusterRoles with users, groups,      or service accounts.
* Example of a RoleBinding:

yaml

Copy

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

  name: read-pods

  namespace: default

subjects:

- kind: User

  name: "john.doe@example.com"

  apiGroup: rbac.authorization.k8s.io

roleRef:

  kind: Role

  name: pod-reader

  apiGroup: rbac.authorization.k8s.io



* Example of a ClusterRoleBinding:

yaml

Copy

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

  name: admin-binding

subjects:

- kind: User

  name: "admin@example.com"

  apiGroup: rbac.authorization.k8s.io

roleRef:

  kind: ClusterRole

  name: cluster-admin

  apiGroup: rbac.authorization.k8s.io

2. **OCI IAM Policies and RBAC**:

* In OKE, OCI IAM policies can be      used to map OCI users and groups to Kubernetes roles.
* Example OCI IAM policy:

Copy

Allow group <group-name> to manage cluster-family in compartment <compartment-name>



* This policy grants the specified      OCI group permissions to manage Kubernetes clusters in the specified      compartment.

**3. Managing User Authentication and Cluster Access Control in OKE**

**Steps to Manage Authentication and Access Control:**

2. **Configure OCI IAM**:

* Create OCI IAM users and groups.
* Define IAM policies to grant      access to OKE clusters.

1. **Create Kubernetes Roles and RoleBindings**:

* Define Roles and ClusterRoles      based on the required permissions.
* Bind these roles to OCI IAM      users or groups using RoleBindings or ClusterRoleBindings.

1. **Use Service Accounts for     Applications**:

* Create service accounts for applications      that need to interact with the Kubernetes API.
* Bind appropriate roles to these      service accounts.

1. **Enable OIDC Integration     (Optional)**:

* Configure OIDC with an external      identity provider for additional authentication options.

1. **Audit and Monitor Access**:

* Use Kubernetes audit logs and      OCI logging to monitor access and actions performed within the cluster.

**4. Best Practices for Authentication and Authorization in OKE**

2. **Least Privilege Principle**:

* Grant users and service accounts      the minimum permissions required to perform their tasks.

1. **Use OCI IAM for Centralized     Management**:

* Leverage OCI IAM for user      authentication and policy management to simplify access control.

1. **Regularly Review Roles and     Bindings**:

* Periodically audit Roles,      ClusterRoles, RoleBindings, and ClusterRoleBindings to ensure they align      with current requirements.

1. **Secure Service Accounts**:

* Avoid using the default service      account for pods. Create dedicated service accounts with limited      permissions.

1. **Enable Multi-Factor     Authentication (MFA)**:

* Use MFA for OCI IAM users to      enhance security.

1. **Monitor and Log Access**:

* Enable Kubernetes audit logging      and OCI logging to track access and changes to the cluster.

**Conclusion**

Authentication and authorization are fundamental to securing Kubernetes clusters in OKE. By leveraging OCI IAM for authentication and RBAC for authorization, organizations can ensure that only authorized users and applications have access to the cluster and its resources. Properly configuring and managing these mechanisms is essential for maintaining a secure and compliant Kubernetes environment.

    LAB :

    ruser1@cloudshell:.kube (us-ashburn-1)$ k create ns raman

-- creating a user  dev-user .

-- group :dev-group , admin-group

**Step 3: Define IAM Policies for OKE Access**

2. **Navigate to Policies**:

* Go to **Identity &      Security > Policies**.

1. **Create Policies**:

* Create a      policy to allow admin-group full access to OKE:
* Name: admin-oke-policy
* Description: "Policy for       admin-group to manage OKE clusters."
* Policy       Statements:

Copy

Allow group admin-group to manage cluster-family in tenancy



* Create a      policy to allow dev-group read-only access to OKE:
* Name: dev-oke-policy
* Description: "Policy for       dev-group to read OKE clusters."
* Policy       Statements:

Copy

Allow group dev-group to read cluster-family in tenancy



* For each      policy:
* Click **Create Policy**.
* Enter the name, description,       and policy statements.
* Click **Create**.

-------------------------------------------------------

**Step 4: Test IAM Policies**

2. **Log in as dev-user** on the portal

U will see that we can only do read on cluster

(( for testing :

    Allow group dev-group to read virtual-network-family in tenancy))

dev-user \_\_\_\_ raman@email >>>> dev-group >>> iam (policy ) read the oke cluster

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

RBAC for fine grained access inside oke ckuster :

---- we have the dev-user .....

---  we have to create a role in oke :

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

  namespace: dev-namespace

  name: read-only-role

rules:

- apiGroups: [""]

  resources: ["pods", "services"]

  verbs: ["get", "list", "watch"]

---- binded above role to the user/group in oci :

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

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

  name: read-only-binding

  namespace: dev-namespace

subjects:

- kind: User

  name: dev-user

  apiGroup: rbac.authorization.k8s.io

roleRef:

  kind: Role

  name: read-only-role

  apiGroup: rbac.authorization.k8s.io

   alias k=kubectl

   17  k api-resources

   18  clear

   19  k get pods

   20  k get ns

   21  k get pods

   22  k get roles

   23  k create ns raman

   24  k get roles

   25  k get rolebindings

   26  ls

   27  kubectl create namespace dev-namespace

   28  alias k=kubectl

   29  k get pods -A

   30  vi role.yaml

   31  k api-resources

   32  clear

   33  k create -f role.yaml

   34  k get role

   35  k get role -n dev-namespace

   36  k describe  role -n dev-namespace

   37  vi rolebinding

   38  vi rolebinding.yml

   39  cat rolebinding.yml

   40  k create -f rolebinding.yml

   41  k get rolebinding -A

   42  clear

   43  k describe rolebinding read-only-binding

   44  k describe rolebinding read-only-binding -A

   45  k describe rolebinding read-only-binding -n dev-namesapce

   46  k describe rolebinding read-only-binding -n dev-namespace

   47  kubectl run testapp --image=httpd -n dev-namespace --as="dev-user"

   48  kubectl auth can-i get pods -n dev-namespace --as="dev-user"

   49  kubectl auth get pods -n dev-namespace --as="dev-user"

   50  kubectl get pods -n dev-namespace --as="dev-user"

   51  kubectl get pods --as="dev-user"

   52  vi role.yaml

   53  k apply -f role

   54  k apply -f role.yaml

   55  kubectl get pods -n dev-namespace --as="dev-user"

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

nodeport service : which exposes our pods/deploymnets >>> outside the cluster at a specific port on the worker nodes

clusterip service : exposes the application to other deployments/application inside the clsuetr

load balancer : external load balancer

ingress controller ::::

    internal load balancer for utkubernets cluster

**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**:

2. **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.

1. **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.

2. **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

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

bash

CopyEdit

kubectl apply -f service.yaml

2. **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.

2. **Deploy an     Ingress Controller**:

bash

CopyEdit

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

2. **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

2. **Apply the     Ingress Resource**:

bash

CopyEdit

kubectl apply -f ingress.yaml

2. **Verify the     Ingress Configuration**:

bash

CopyEdit

kubectl get ingress

**E. Load Balancer Security & Scaling**

2. **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.

1. **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 :

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

**30  k create ns ingress**

**32  helm repo add ingress-nginx**[**https://kubernetes.github.io/ingress-nginx**](https://kubernetes.github.io/ingress-nginx)

**33  helm repo update**

2. **Install the NGINX Ingress     Controller using Helm:**
3. 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.

2. **Verify the Ingress Controller and     Load Balancer:**
3. 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.

-----------------------------------------------------------------------------------

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

k create -f 25thdeploy.yaml -n ingress

-----------------------------------------------------------------------------

history for refrence :

    k get nodes

  148  clear

  149  k get pods -A

  150  k create ns ingress

  151  k get ns

  152  clear

  153  helm repo add ingress-nginx <https://kubernetes.github.io/ingress-nginx>

  154  helm repo list

  155  helm install my-release ingress-nginx/ingress-nginx  --namespace ingress  --set controller.replicaCount=2

  156  helm list

  157  clear

  158  helm list

  159  helm list -n ingress

  160  k get all -n ingress

  161  clear

  162  k get all -n ingress

  163  ls

  164  vi 25thdeploy.yml

  165  k create -f deploy.yml

  166  k delete -f deploy.yml

  167  k get pods

  168  k get ns

  169  k get pods -n ingress

  ---

  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

  ####

**Key Features in Your Ingress YAML**

1. **Annotations:**

* nginx.ingress.kubernetes.io/ssl-redirect: "false"
* → Disables automatic HTTPS redirection (keeps HTTP traffic).
* nginx.ingress.kubernetes.io/use-regex: "true"
* → Enables regex in the path field for flexible URL matching.
* nginx.ingress.kubernetes.io/rewrite-target: /$2
* → Rewrites the incoming request to remove the first part (/raman or /raman2) and forward only the extracted portion to the backend service.

1. **Rules and Path Matching:**

* **Rule 1**:
* Requests matching /raman/... are routed to raman-service on port 80.
* The regex /raman(/|$)(.\*) ensures that:
* /raman → is rewritten to /
* /raman/test → is rewritten to /test
* **Rule 2**:
* Requests matching /raman2/... are routed to raman-service2 on port 80.
* The regex /raman2(/|$)(.\*) behaves the same as above.

**How Requests Are HandledIncoming RequestRewritten PathForwarded to Service/raman/raman-service/raman/hello/helloraman-service/raman2/raman-service2/raman2/info/inforaman-service2**

####

k apply -f 25thdeploy-yaml -n ingress

**Step 5: Test the Setup**

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

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

history for refrence :

      clear

  171  history

  172  clear

  173  k get all -n ingress

  174  k create -f deploy.yml -n ingress

  175  k get ns

  176  clear

  177  k get all

  178  k get pods all -n ingress

  179  k get all -n ingress

  180  vi 25thdeploy.yml

  181  k delete -f deploy.yml -n ingress

  182  clear

  183  ls

  184  k create -f 25thdeploy.yml -n ingress

  185  clear

  186  k get all -n ingress

  187  clear

  188  k get all -n ingress

  189  clear

  190  vi 25thdeploy.yml

  191  k apply -f 25thdeploy.yml -n ingress

  192  k get ingress -n ingress

  193  k describe ingress -n ingress

  194  k get pods -o wide  -n ingress

  195  k get svc -n ingress

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

  pv/pvc :

**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:

2. **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).

1. **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).

1. **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**

2. **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.

1. **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

2. **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

2. **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: "/"

2. **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

2. **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.

-------------------------------------------------------------------------------------------------------------------------------------------------------------

LAB :

     purpose : deploymnt : replicas : pods >>> multiple hosts  ......................commom storage

**Lab Setup**

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

2. **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**.

  --- TEST THE FSS :

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

n  Ssh on both cluster  nodes

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

* File Storage >> File Systems >> File System     details

n  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

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

-- login to worker nodes and test it...

2. **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

 NOTE : MAKE SURE PVC IN SAME NAMESPACE AS PODS/DEPLOYMENT

n  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:**

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

bash

Copy

kubectl apply -f deployment.yaml

2. 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.

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

history for refrence :

     cat pv

  240  cat pv.yaml

  241  vi pv.yaml

  242  k create -f pv.yaml

  243  k get pods

  244  k get pv

  245  k describe pv

  246  ls

  247  cat pvc.yaml

  248  k create -f pvc.yaml

  249  k get pvc

  250  k get pv

  251  LS

  252  ls

  253  vi deploy.yml

  254  k create -f deploy.yml

  255  k get pods

  256  k describe pod test-deployment-fss-54c975f585-5dhj9

  257  clear

  258  k get pods

  259  cat deploy.yml

  260  k get pods -o wide

  261  clear

  262  cat deploy.yml

  263  k get pods -o wide

  264  k exec -it test-deployment-fss-54c975f585-5dhj9 -- /bin/bash

  265  k exec -it test-deployment-fss-54c975f585-h2wbt -- /bin/bash

  266  k describe pod

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