From Zero to Production in Kubernetes - Workshop – Lab Guide

Table of Content

INTRODUCTION 2

GET STARTED 3

LAB 1 - DEPLOYING MUSHOP LOCALLY USING DOCKER.....13

LAB 2 – DEPLOY OKE CLUSTER WITH TERRAFORM 30

LAB 3 - OCI VAULT INTEGRATION WITH OKE.....33

LAB 4 - INSTALL MUSHOP USING HELM 46

LAB 5 - INSTALL INGRESS CONTROLLER 51

LAB 6 - INSTALL CERTMANAGER.....52

LAB 7 – DEPLOYMENT AUTOSCALING 57

LAB 8 – MONITORING THE DEPLOYMENT.....64

LAB – 9 - CREATE ALARMS TO DETECT UNSCHEDULED PODS 71

LAB 10 - USE TAINTS AND TOLERATIONS FOR RESOURCE ALLOCATION (OPTIONAL) 77

From Zero to Production in Kubernetes - Workshop – Lab Guide

Introduction

Build and Deploy the MuShop App

Learn how to deploy, Maintain and Troubleshoot the MuShop App into a cloud-native microservice-based development framework leveraging the Oracle Cloud Native solutions. You will deploy an eCommerce application with polyglot microservices for each component of a highly scalable architecture leveraging Kubernetes and Oracle Cloud Infrastructure components.

Objectives

In this lab, you will:

- Create Resources and setup a Developer Enviornment based on OKE
- Deploy an eCommerce microservices-based solution to Kubernetes (MuShop App)
- Working with infrastructure as a code tools
- Troubleshooting challenges
- Explore your App
- Expose your app publicly
- Integrate the application with Logging, monitoring and 3rd party services

Prerequisites

1. Proficient knowledge and experience with Kubernetes
2. Familiarity with YAML, commands, manifests, and Kubernetes architecture
3. Experienced with Helm charts
4. An Oracle Free Tier (Trial) Account

Get Started

Introduction

Before you get started, you will need an Oracle Cloud account. This lab walks you through the steps of getting an Oracle Cloud Free Tier account and signing in.

Existing Cloud Accounts

If you already have access to an Oracle Cloud account, skip to **Task 2** to sign in to your cloud tenancy.

Objectives

- Create an Oracle Cloud Free Tier account
- Sign in to your account

Prerequisites

- A valid email address
- Ability to receive SMS text verification (only if your email isn't recognized)

Note: Interfaces in the following screenshots may look different from the interfaces you will see.

Task 1: Create Your Free Trial Account

If you already have a cloud account, skip to **Task 2**.

1. Validate that you received email in the following format from Oracle's marketing and click the link for registration or copy it to your browser.

בהמשך להרשמתך לקורס ההכרות עם ענן אורקל שמתקיים מחר
הסדנא כוללת תרגול מעשי בסביבת הענן של אורקל. לקראת הסדנא אנחנו מעניקים עבור הנרשמים **400 אירו קרדיט בסביבת ענן חדשה כהטבה לנרשמי הסדנא**.

****יש לפתוח את חשבון הענן לפני הגעתכם לסדנא!**

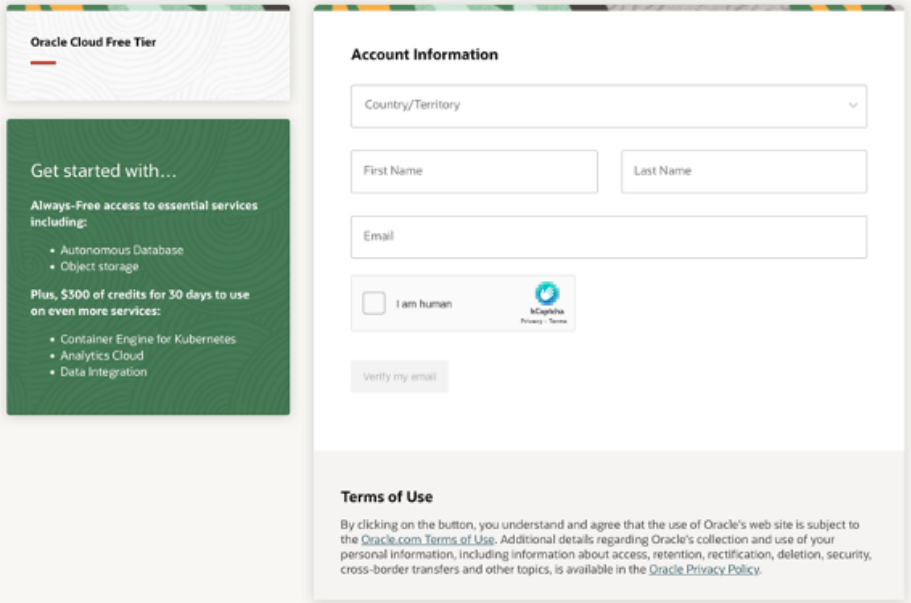
ניתנה לך הרשאה התקפה ל-30 יום
יש צורך להזין פרטי כרטיס אשראי עבור זיהוי המשתמש אך הוא לא יחויב ו/או ישמר כאמצעי חיוב.

מצורפות הנחיות לפתיחת החשבון:

קישור לפתיחת החשבון:
[signup.oraclecloud.com] Oracle Cloud Free Tier Signup
חשוב: בהרשמה השתמשו באימייל איתו נרשמתם לקורס.
במידה ובעבר נפתח עבורכם חשבון ענן באורקל עם האימייל שנרשמתם, יש צורך לשלוח לנו אימייל אחר שאיתו תוכלו לקבל גישה.

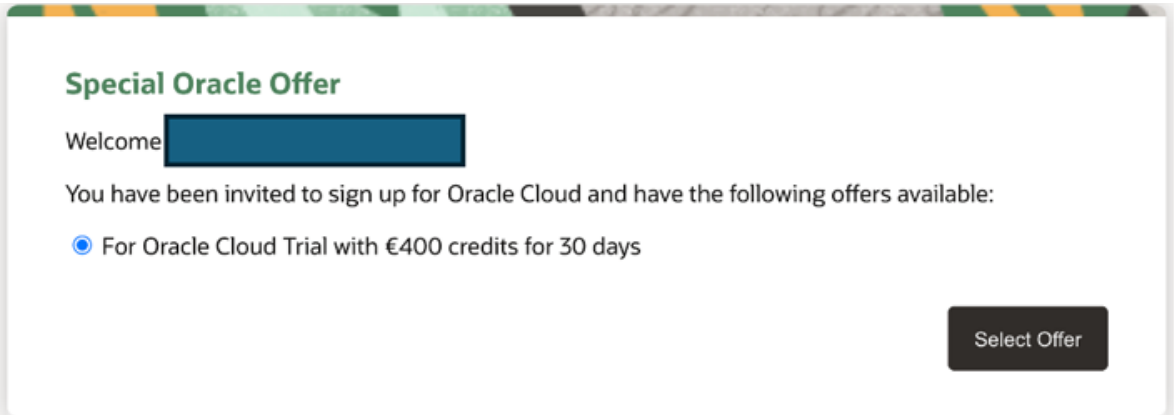
בנוסף מצורף לינק לסרטון הדרכה קצר איך להגדיר את החשבון:
[youtube.com] https://www.youtube.com/watch?v=Yr1Cz-U-Kok

2. An alternative option in case of no marketing email available is to Open up a web browser to access the Oracle Cloud account registration form at oracle.com/cloud/free.
3. In both cased above you You will be presented with a registration page.

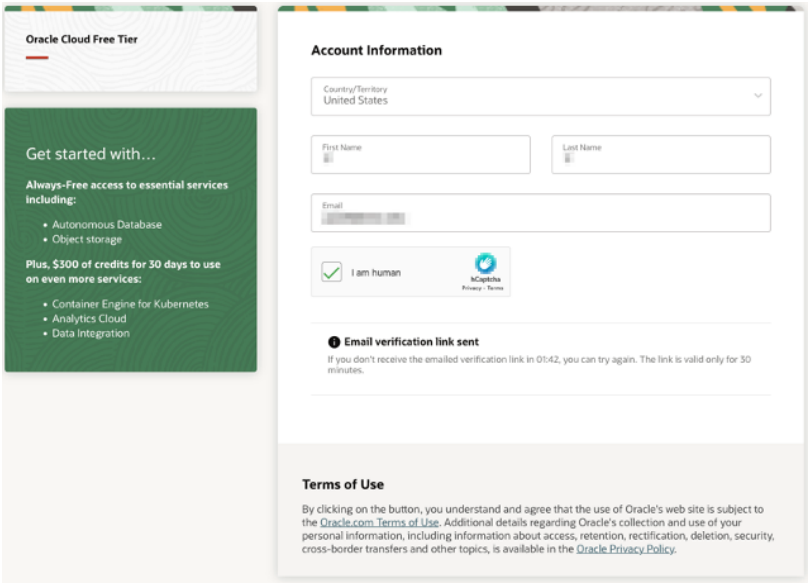


4. Enter the following information to create your Oracle Cloud Free Tier account.
    - Choose your **Country**
    - Enter your **Name** and **Email**
    - Use hCaptcha to verify your identity

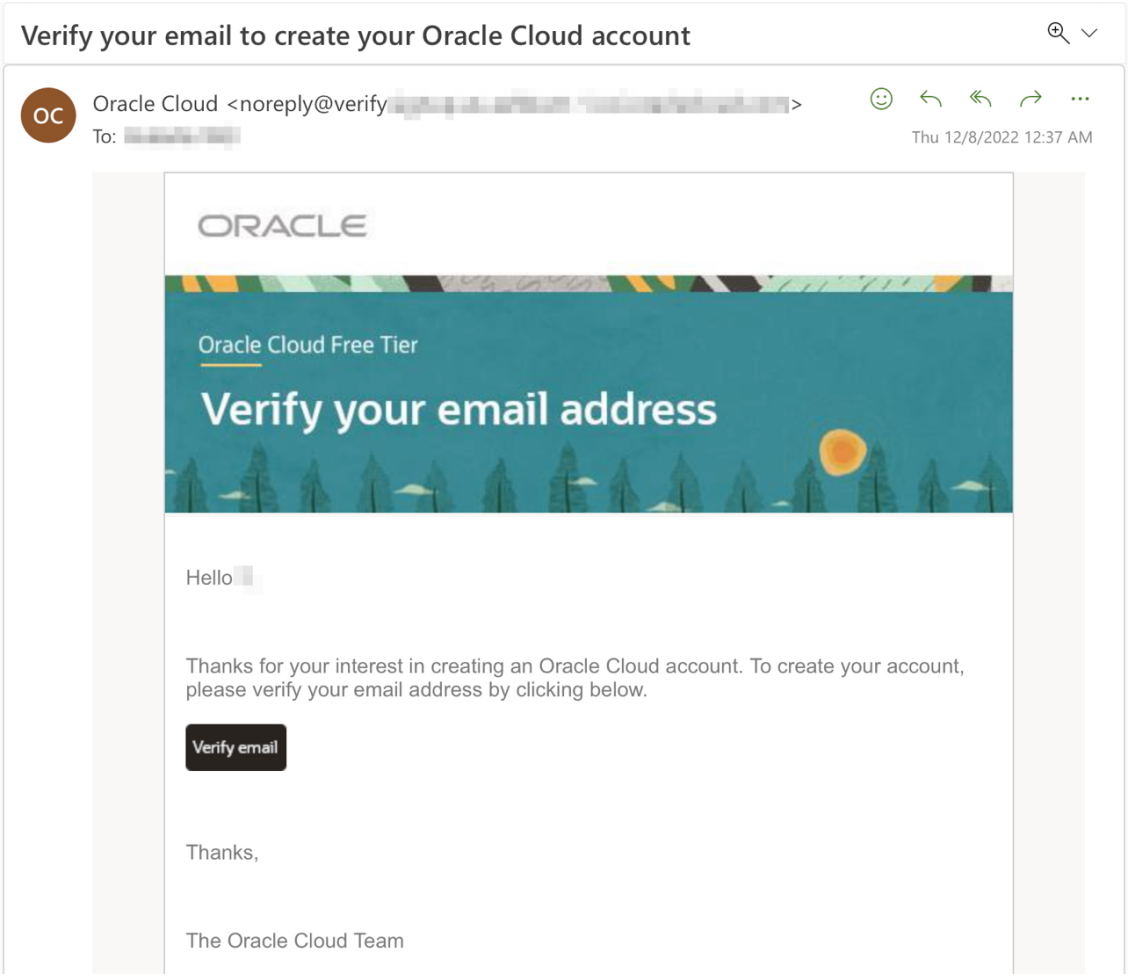
5. Once you have entered a valid email address, select the **Verify my email** button. The screen will appear as follows and you should click **Select Offer**



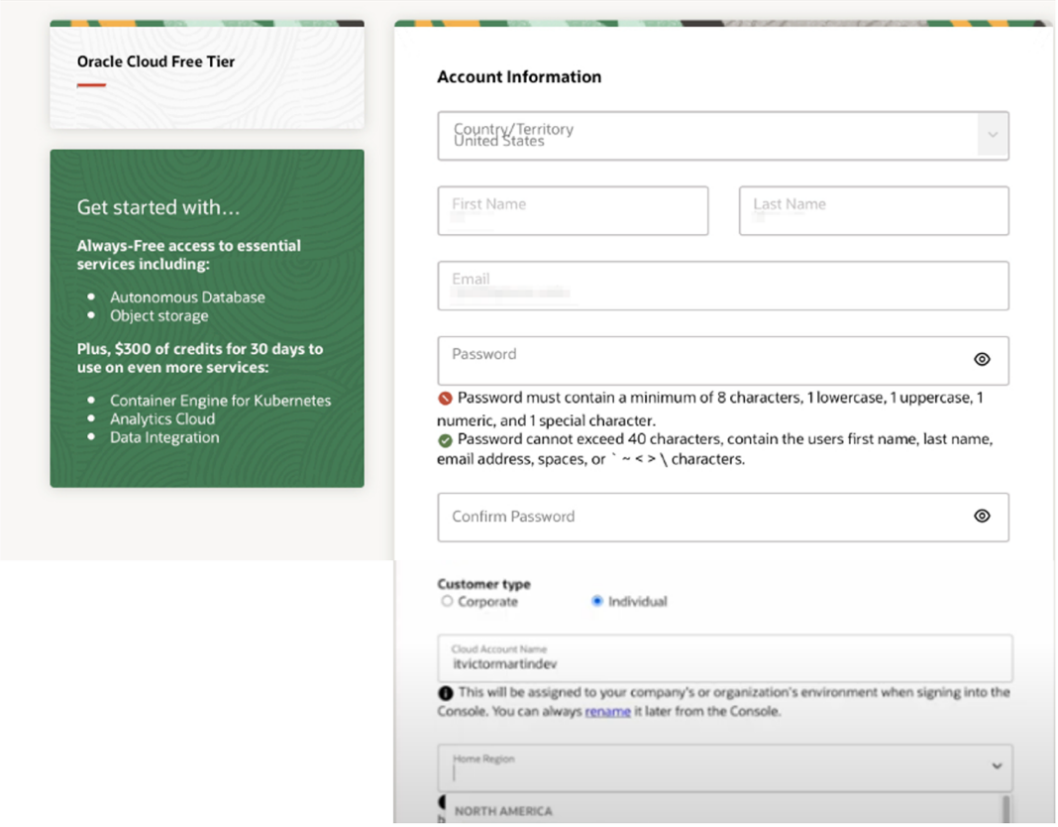
Wait for **Email verification link sent** notification



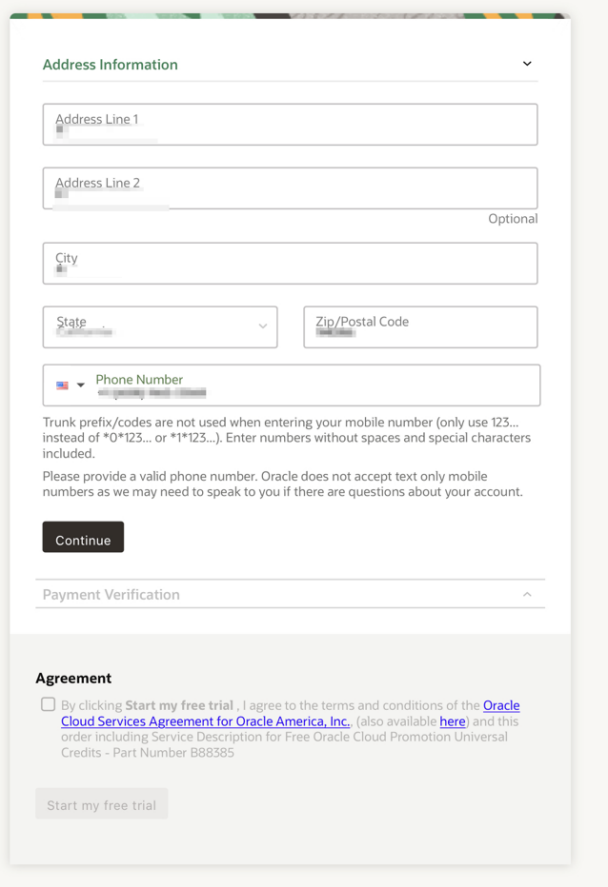
6. Go to your email. You will see an account validation email from Oracle in your inbox. The email will be similar to the following:



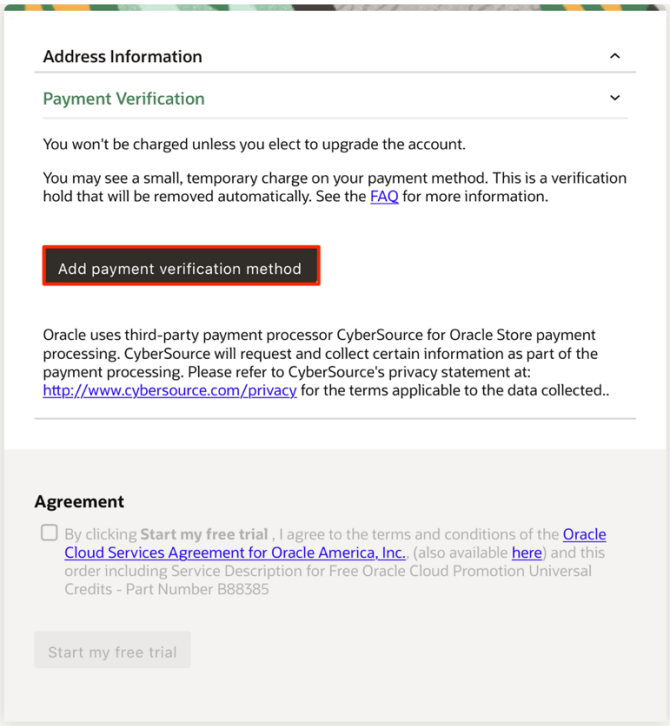
7. Click **Verify email**.
8. Enter the following information to create your Oracle Cloud Free Tier account.
    - Choose a **Password**
    - Customer Type : **Change to Individual**
    - Your **Cloud Account Name** will generate automatically based on your inputs. You can change that name by entering a new value. **Remember what you wrote. You'll need this name later to sign in.**
    - Choose a **Home Region** : **Frankfurt** ! Your Home Region cannot be changed once you sign-up.
    - Checkbox the acknowledge message
    - Click **Continue**



9. Enter your address information. Choose your country and enter a phone number. Click **Continue**.

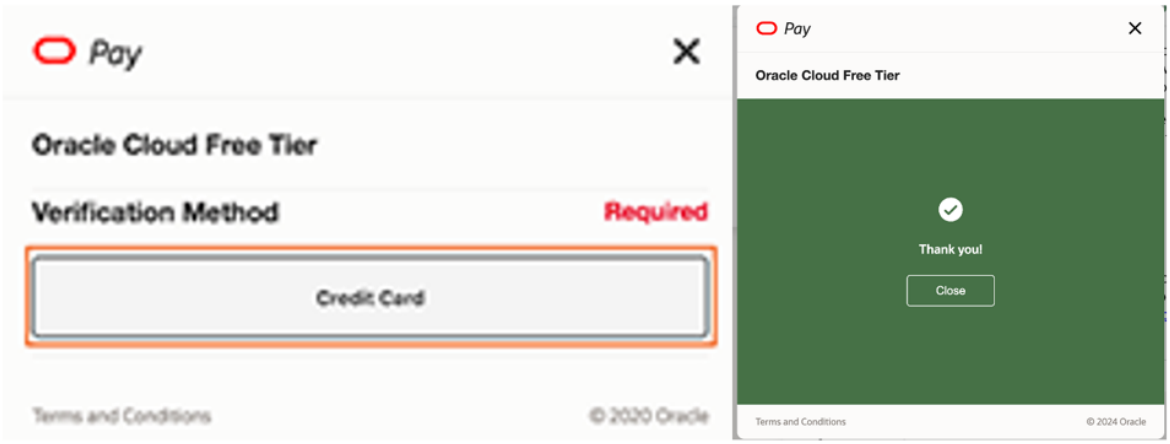


10. Click the **Add payment verification method** button.

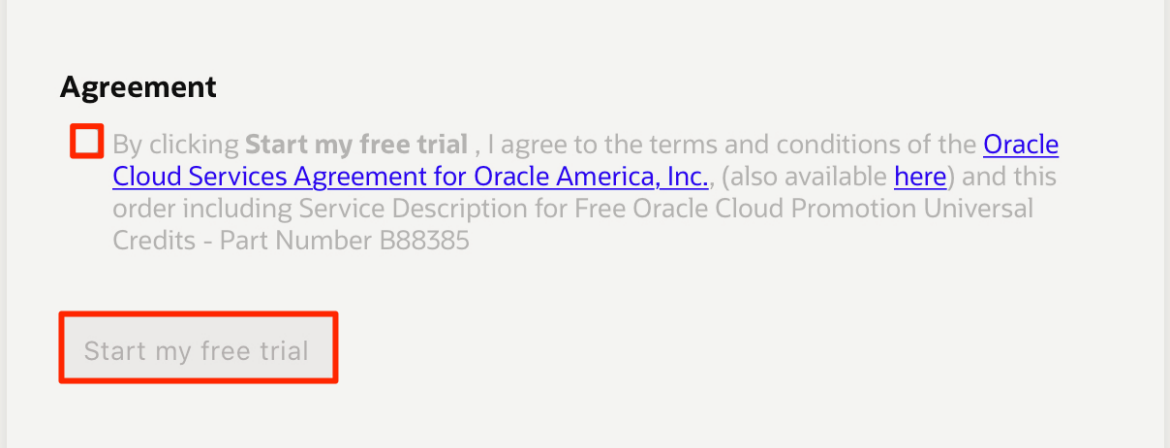


11. Choose the verification method. In this case, click the **Credit Card** button. Enter your information and payment details.

    Note: This is a free credit promotion account. You will not be charged unless you elect to upgrade the account.



12. Once your payment verification is complete, review and accept the agreement by clicking the check box. Click the Start my free trial button.

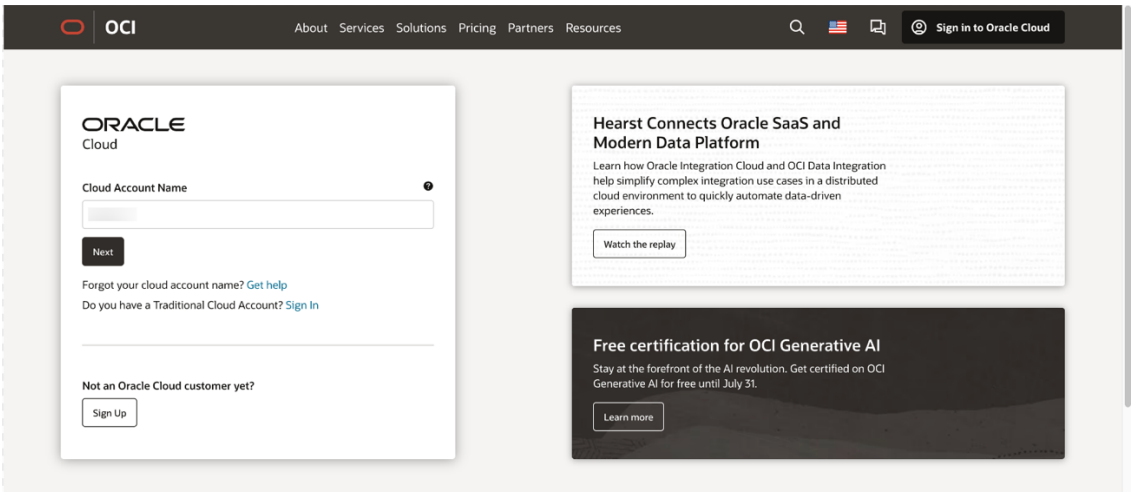


13. Your account is provisioning and should be available soon! You might want to log out as you wait for your account to be provisioned. You'll receive an email from Oracle notifying you that provisioning is complete, with your cloud account and username.

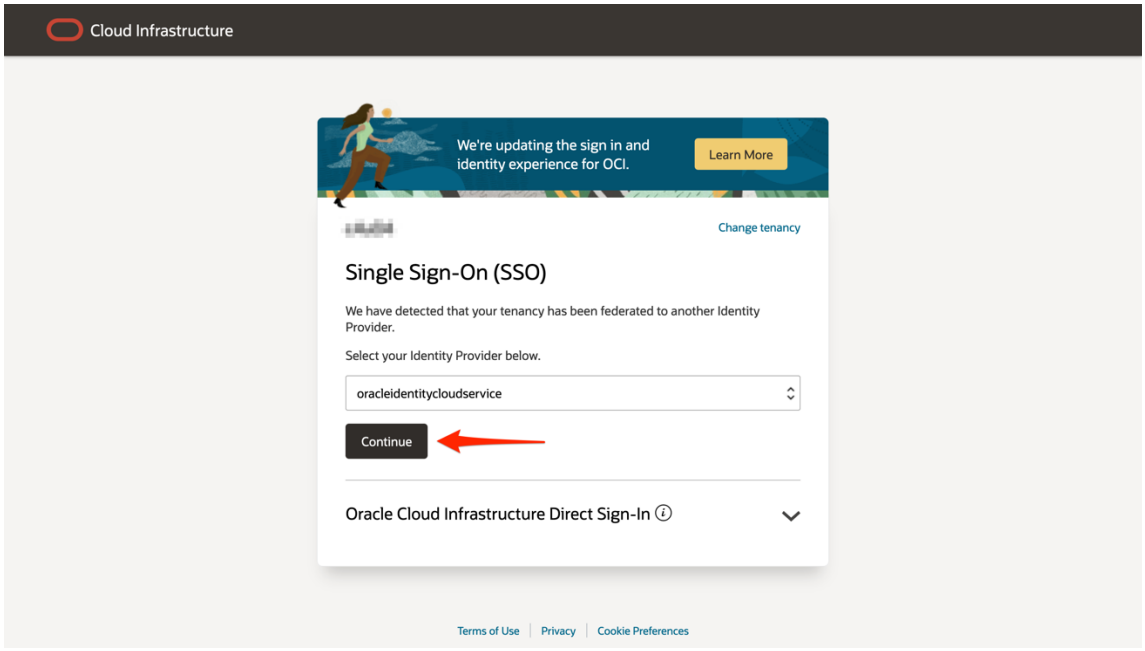
Task 2: Sign in to Your Account

Please note that while your tenancy is initially created, you will only see a direct login. Once your tenancy is fully provisioned, you will see the screens as described below.

1. Go to cloud.oracle.com. Enter your Cloud Account Name and click Next. This is the name you chose while creating your account in **Task 1/Section 8** . It's NOT your email address. If you've forgotten the name, see the confirmation email.

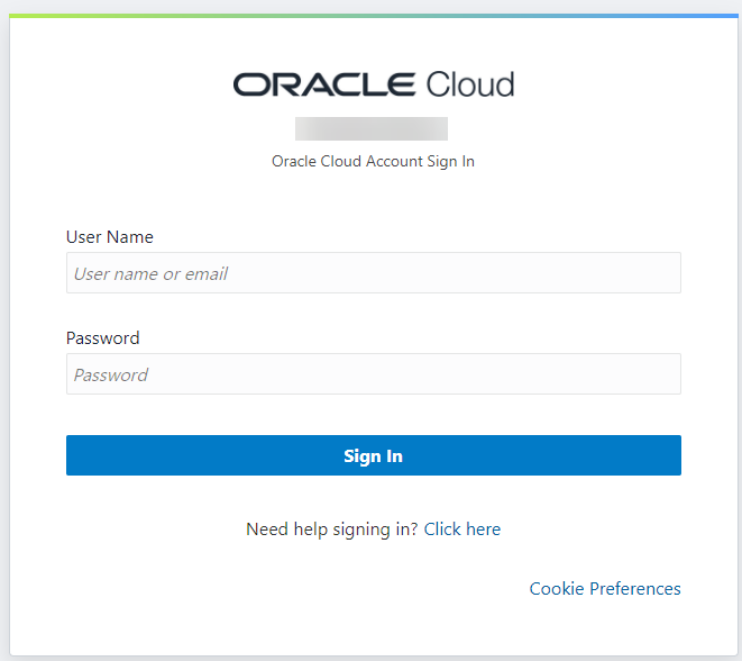


2. Click **Continue** to sign in using the *"oracleclouddidentityservice"*.

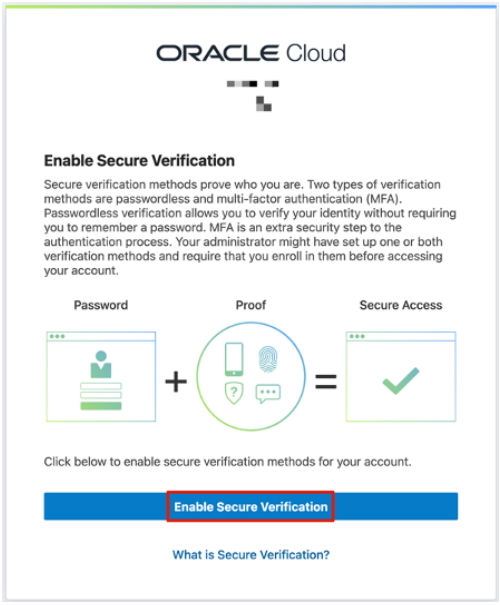


When you sign up for an Oracle Cloud account, a user is created for you in Oracle Identity Cloud Service with the username and password you selected. You can use this single sign-on option to sign in to Oracle Cloud Infrastructure and then navigate to other Oracle Cloud services without re-authenticating. This user has administrator privileges for all the Oracle Cloud services included with your account.

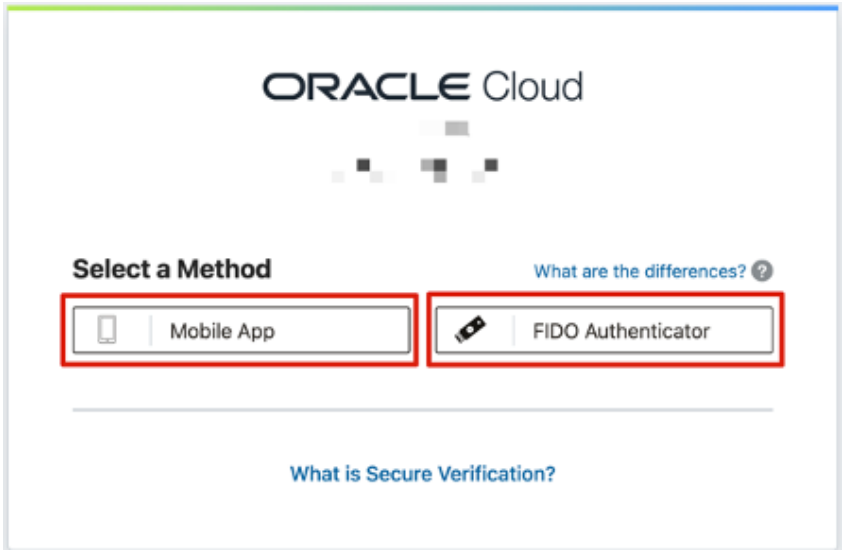
3. Enter your Cloud Account credentials and click **Sign In**. Your username is your email address. The password is what you chose when you signed up for an account.



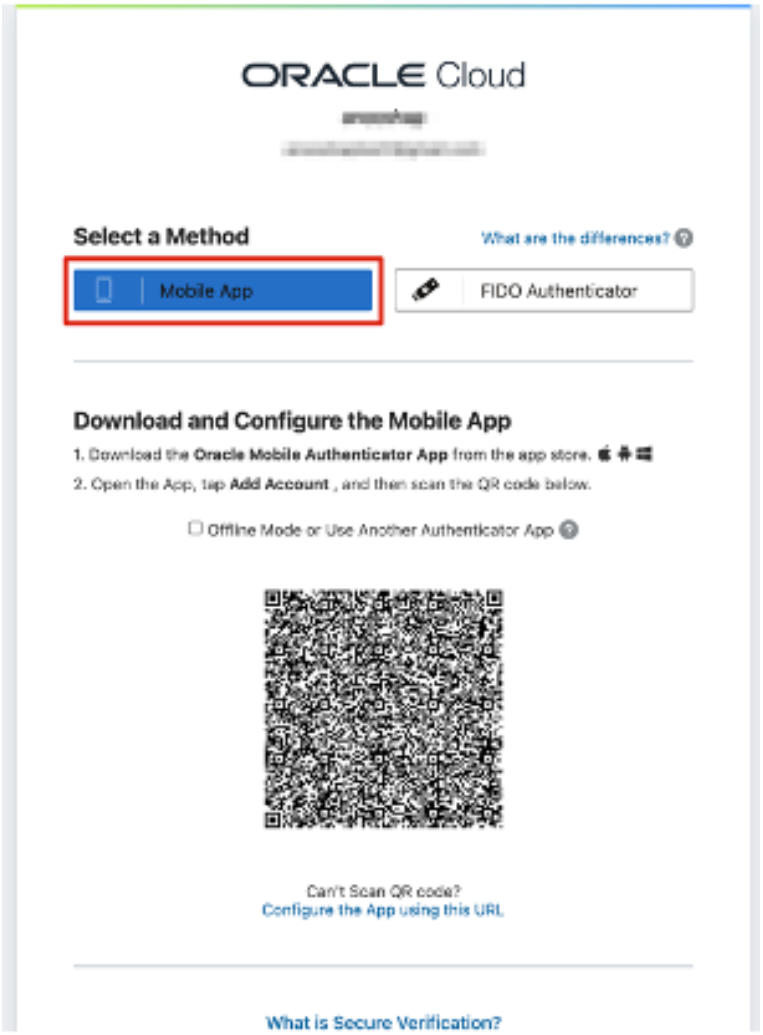
4. You will be prompted to enable secure verification. Click **Enable Secure Verification**. For more details, refer the [Managing Multifactor Authentication documentation]



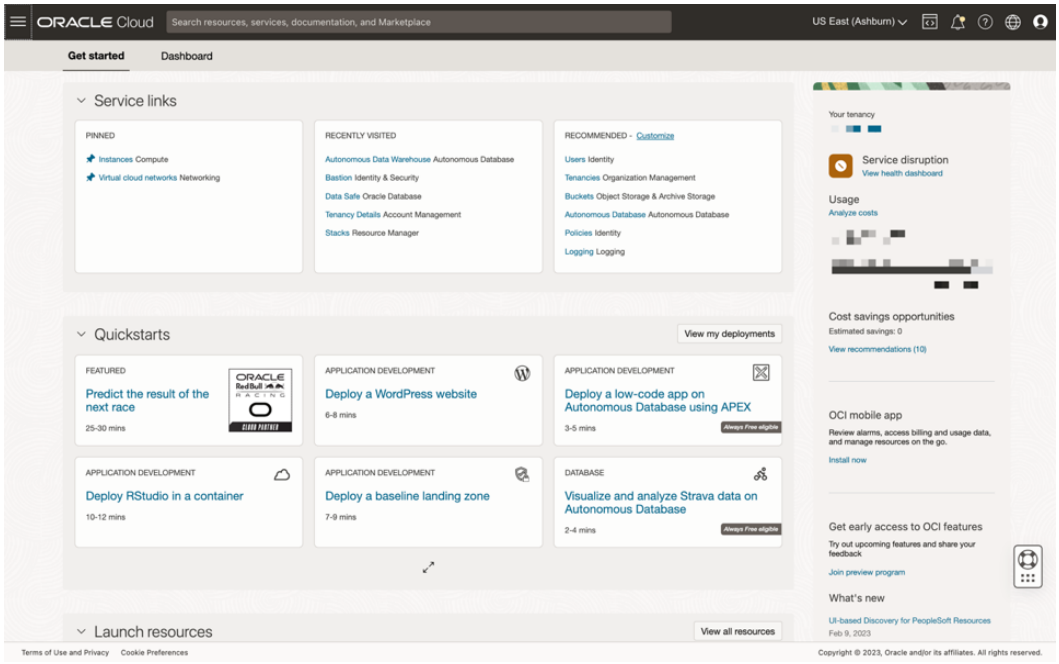
5. Select a method - **Mobile App** or **FIDO Authenticator** to enable secure verification.



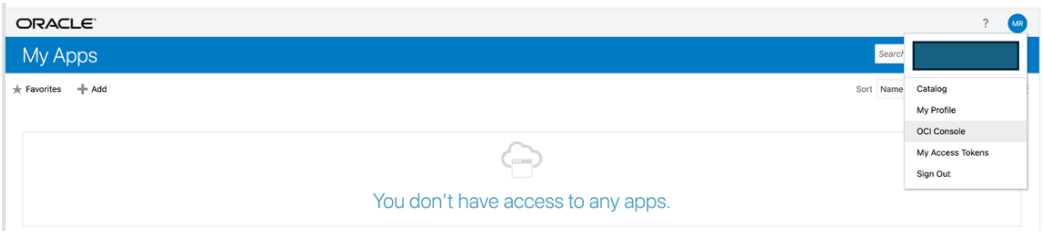
6. Choose **Mobile App –** Follow the steps as shown in the screenshot to setup authentication.



7. Once you have verified authentication, you will now be signed in to Oracle Cloud!



8. In case you received the following view go to Profile and Click OCI Console



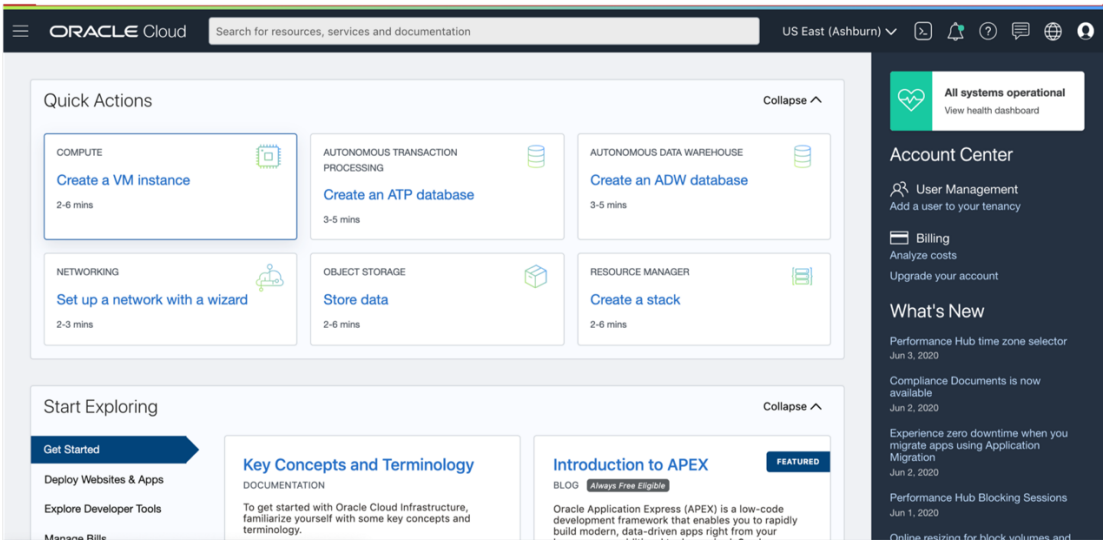
LAB 1 - Deploying MuShop Locally Using Docker

Introduction

Task 1: Log into OCI Tenancy

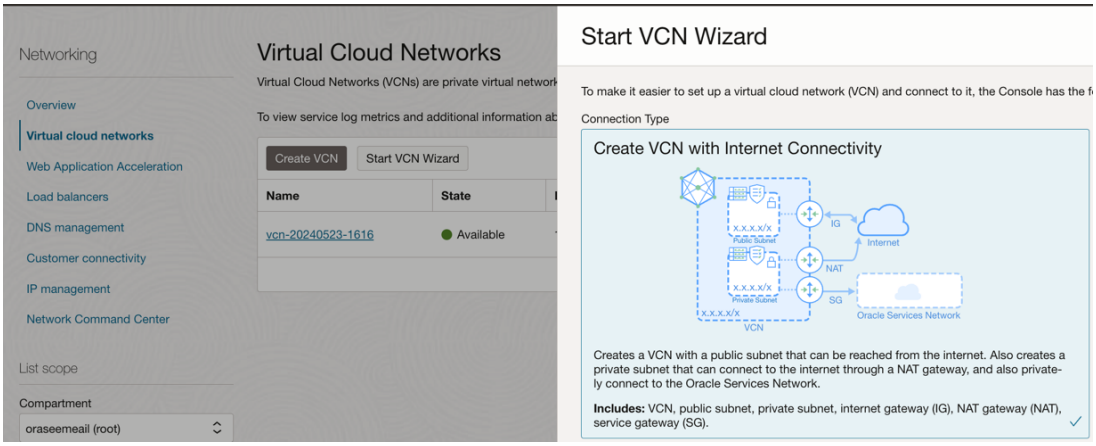
Log in to your OCI dashboard and retrieve information required to create resources.

1. Once you log in you will see a page similar to the one below. Click on "Infrastructure Dashboard."



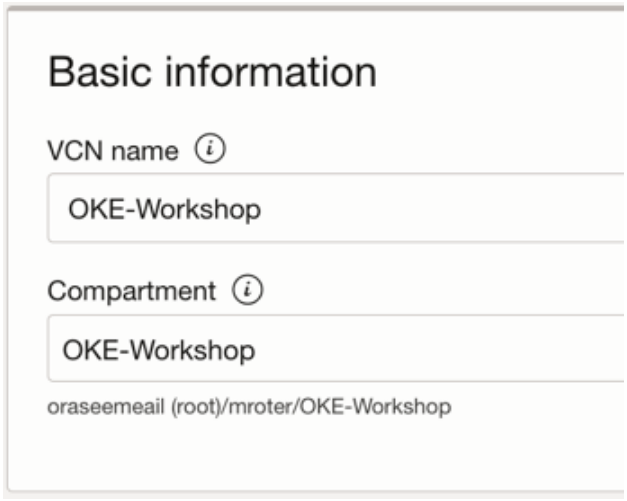
Task 2: Create Virtual Cloud Network (VCN)

1. Navigate to **Networking** > **Virtual Cloud Networks >** Click **Start VCN Wizard** > Select **VCN with Internet Connectivity** > Start **VCN Wizard**

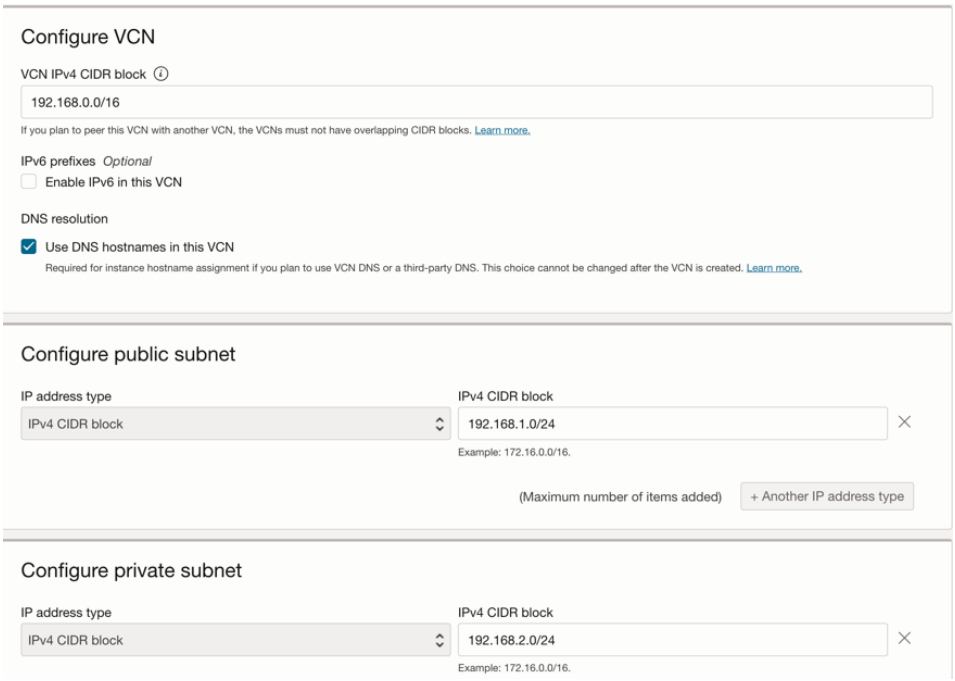


2. **Enter the following:**

   - VCN Name **:** `okeworkshop`
   - Compartment : `choose the root compartment`



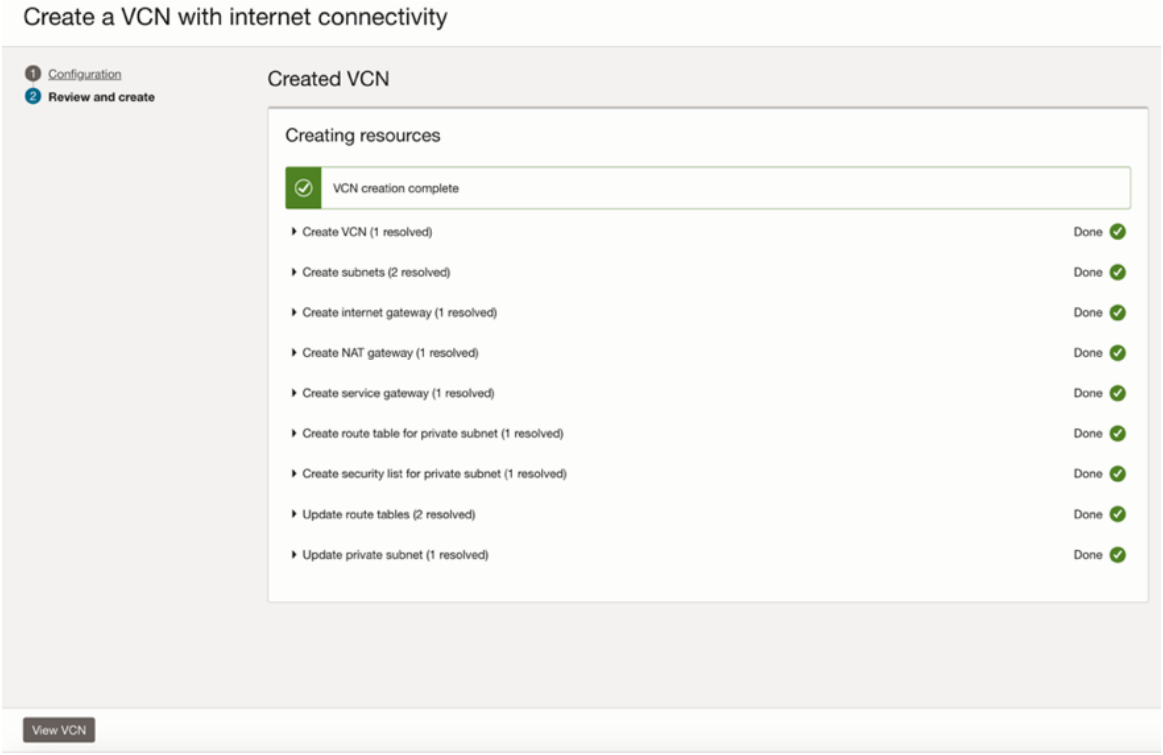
   - CIDR Block: `192.168.0.0/16`
   - Public CIDR Block: `192.168.1.0/24`
   - Private CIDR Block: `192.168.2.0/24`



   - Click **Next**
   - Validate all the information completed and Click **Create**

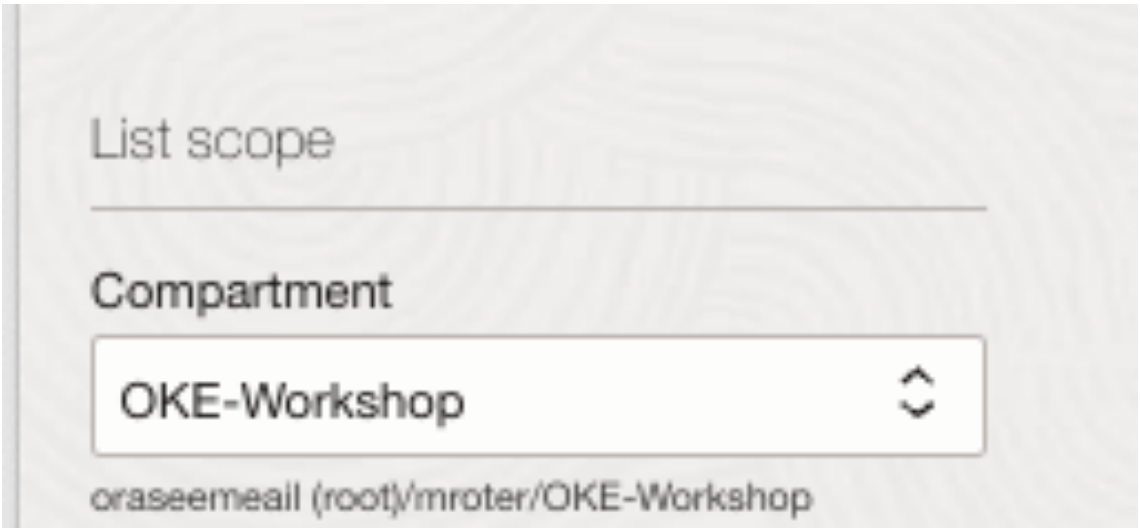
3. Click **View VCN** on the bottom left

Note: This process creates a Virtual Network with the following Components: VCN, Public subnet, Private subnet, Internet gateway (IG), NAT gateway (NAT), Service gateway (SG)



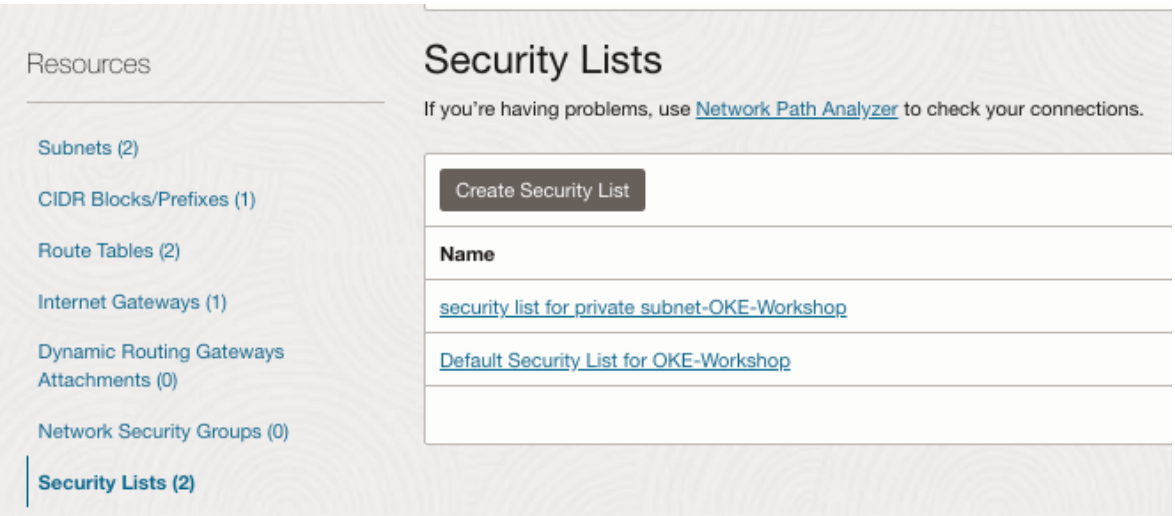
4. **The following view shows the configured VCN in details**

Note: if no information available go to List scope > Compartment > Validate the proper compartment is in place



Task 3: Configure Security List for Public Access

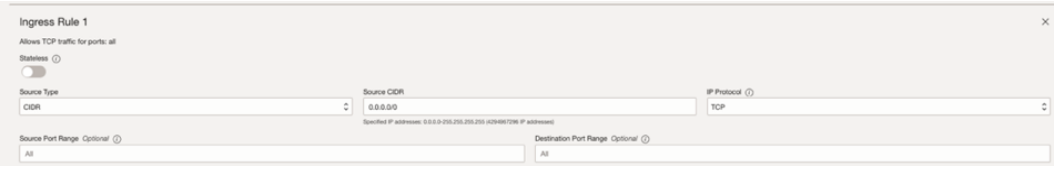
1. In the VCN view navigate to **Resources** > **Security Lists >** Click **Default Security list for** < `VCN Name`>



2. Click **Ingress Rules > Add Ingress Rules**

3. **Enter the following :**

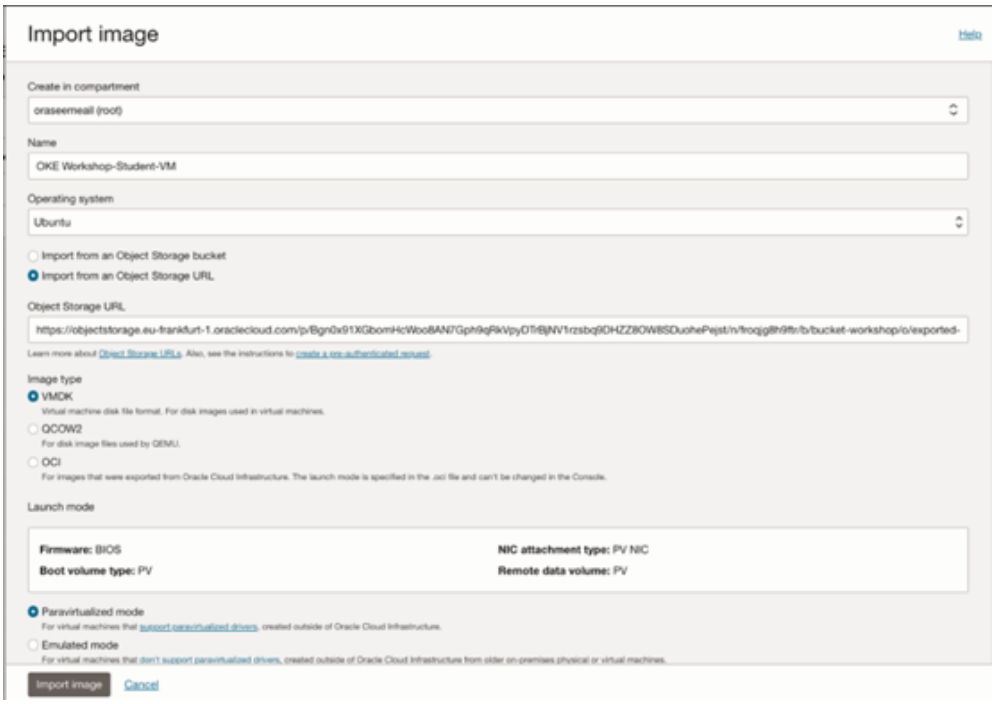
   - Source CIDR: `0.0.0.0/0`
   - Destination Port Range: `All (default)`



   - Click **Add Ingress Rules** and Validate rule is in the list
   - Egrees Rule : `0.0.0.0/0 (default)`

Task 4: Import custom Image

1. Navigate to **Menu** > **Compute** > **Custom Images** in OCI Console.
2. Select **Import Image** and enter the following :
    - Compartment: `Select the appropriate compartment.`
    - Name: `Assign a descriptive name. Avoid confidential information`
    - Operating System: `Choose Ubuntu.`
    - Check the option **"**`Import from an Object Storage URL`**"**
    - Object Storage URL: `Paste the URL` : *https://objectstorage.uk-london-1.oraclecloud.com/p/2o164n0KlkY--neE-r78QVLUySU1ELPfxPB4cPhKTaApNEDzQzhWm86En4aFGaaR/n/oraseemeail/b/bucket-oke-workshop/o/exported-image-20241117-1641*
    - Image Type: `VMDK (default)`
    - Launch Mode: `Paravirtualized (default).`
3. Click **Import Image** to complete.
    - In case of "invalid source URL " error message please check if you copy the URL without spaces in the beginning and in the end

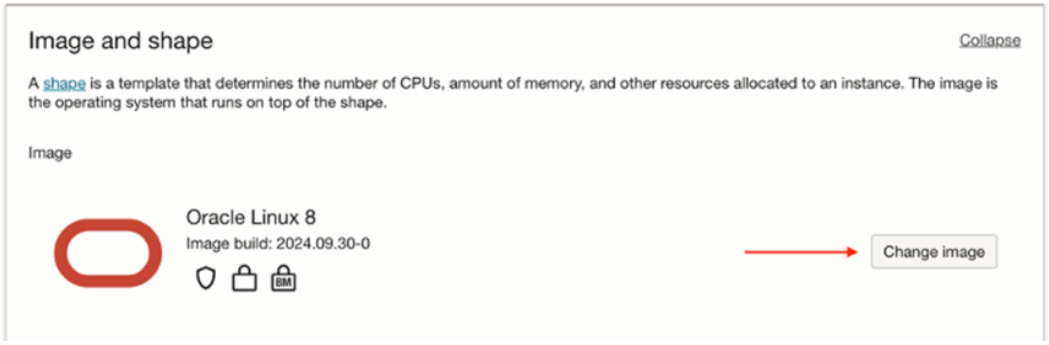


Now Wait until the image uploaded before moving to Task 5

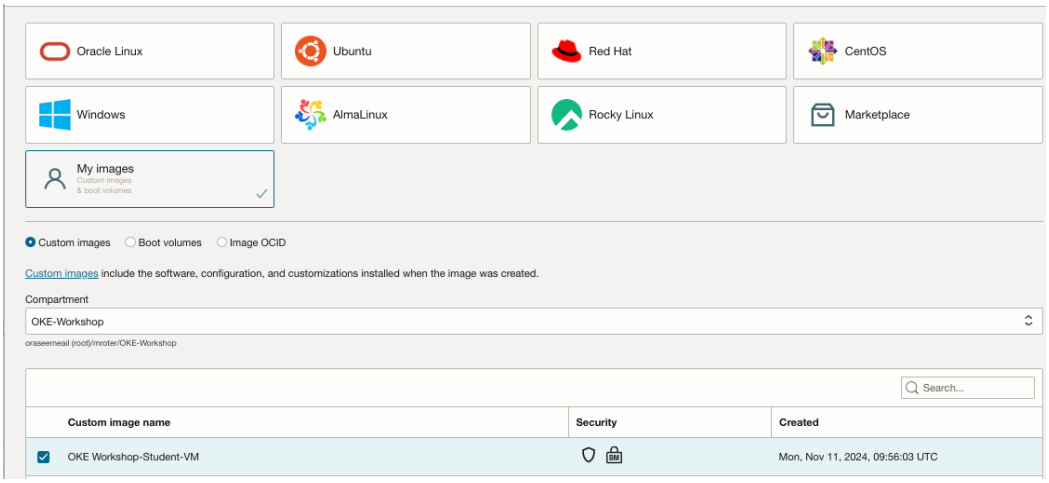
Task 5: Create a Compute Instance

1. Navigate to **Compute** > **Instances** > **Create Instance**.
2. Enter the following to setup the Instance:

    - Instance Name and Compartment: `Input relevant details.`
    - Placement : `Frankfurt-AD1 (default)`
    - Security: `Default`
    - Image Source: `Select Image and shape > Change image`

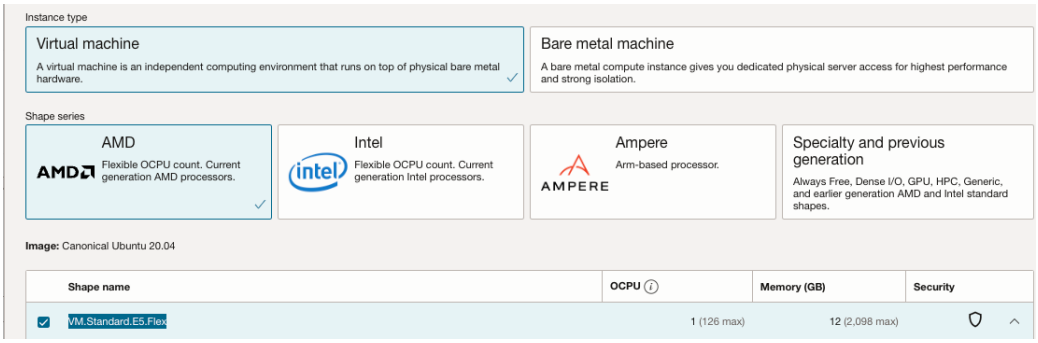


    - Select **My images > Custom images > Uploaded in Task 1 > click select image**

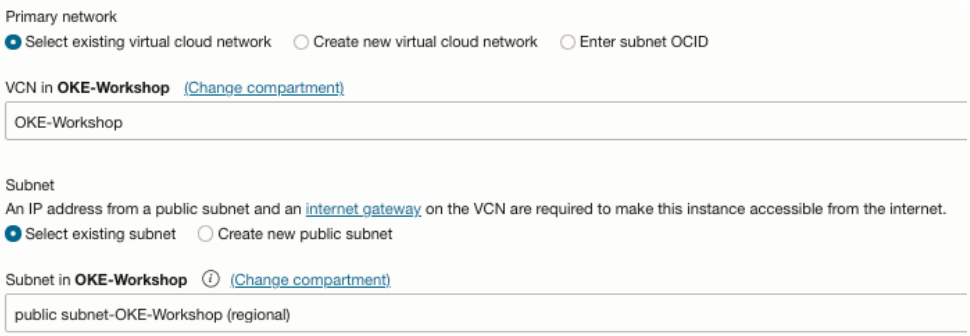


    - **VM Shape :** Select **Image and shape** > **Change shape**

- **Select the following Parameters**: **Intance type**: `Virtual machine` , **Shape series**: `AMD` and **Shape name**: `VM.Standard.E5.Flex`

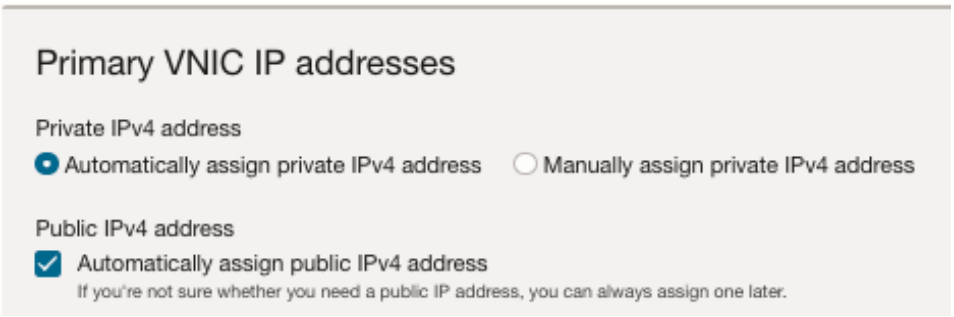


- **Network Settings** (Primary VNIC information):
  - **Primary network** - Select **existing virtual cloud network** > Choose the **VCN created in Task 2.**
  - **Subnet** - Select **existing subnet** > Choose the **Public Subnet** created in Task 2.

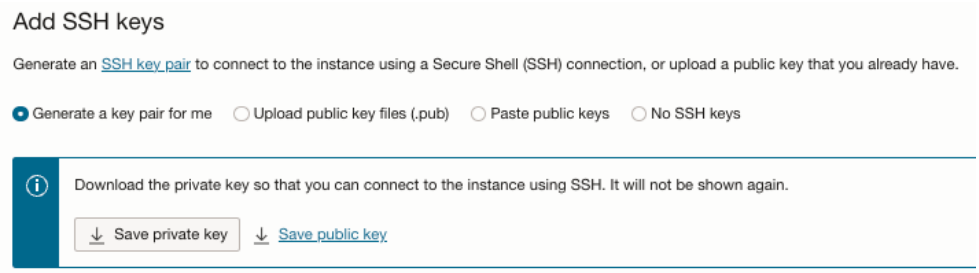


  - **Note** : Validate the Compartment name if the VCN/Subnet is not in the drop-down list

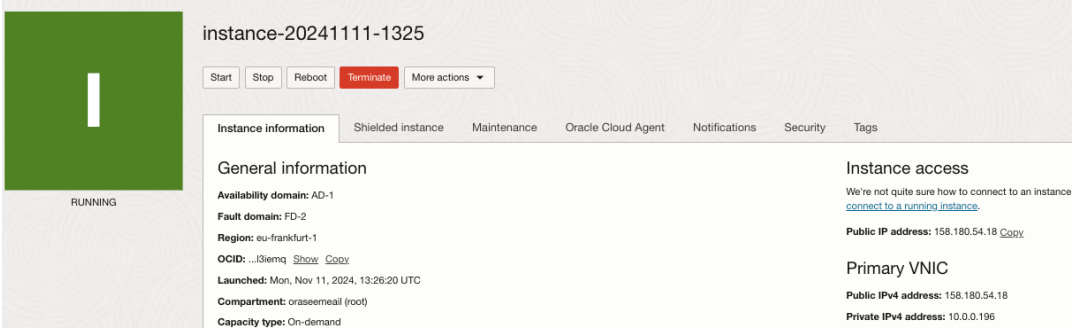
- **Primary VNIC IP addresses:**
  - **Private Ipv4 address >** Checkbox **Automatically assign private Ipv4 address** (default)
  - **Public Ipv4 address** > Checkbox **Automatically assign public Ipv4 address** (default).



- **Add SSH Keys:**
  - Choose **Generate a key pair for me** > Save both **private** and **public keys**



- **Boot Volume**: `Unchecked values (default)`
- **Block Volume**: `No volumes (default)`
- **Live migration**: `Enabled (default)`
- Click **Create** and validate instance is in Running state
- **Make a note of public ip address**



Task 6: Connect the VM via SSH

1. From terminal (On local computer – MacOS, Windows, Linux) enter the following command to change permission for the downloaded Private Key before SSH to the VM

   ```
   chmod 700 ssh_private_key.key
   ```

   ```
   % chmod 700 ssh-key-2024-11-17.key
   %
   ```

2. From the terminal SSH to the VM by executing the following command:

```
ssh -i <path/to/private key/ssh_private_keyname> ubuntu@<PUBLIC_IP_OF_COMPUTE>
```

Note : Replace the <PUBLIC_IP_OF_COMPUTE> with the instance assigned public IP

Task 7: Configure OCI CLI on VM

1. Execute the following command on the VM for OCI CLI setup :

```
oci setup config
```

```
PS D:\nadeem> oci setup config
```

```
This command provides a walkthrough of creating a valid CLI config file.

The following links explain where to find the information required by this
script:

User OCID and Tenancy OCID:

    https://docs.us-phoenix-1.oraclecloud.com/Content/API/Concepts/apisigningkey.htm#Other

Region:

    https://docs.us-phoenix-1.oraclecloud.com/Content/General/Concepts/regions.htm

General config documentation:

    https://docs.us-phoenix-1.oraclecloud.com/Content/API/Concepts/sdkconfig.htm
```

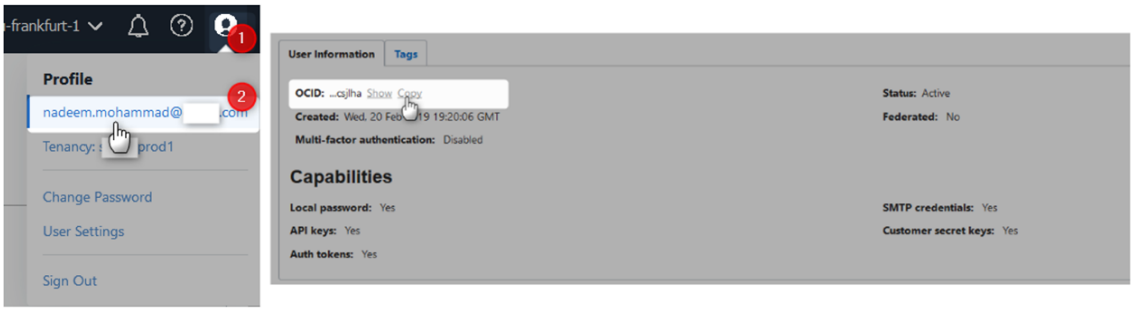
2. Choose **default location** > **Click Enter**

```
Enter a location for your config [C:\Users\nadeem\.oci\config]:
```

3. Enter **User OCID** when prompted:

```
Enter a user OCID: ocid1.user.oc1..xxxxxxxxxxxxxxxplxgzkpmwgjzdbgg2szjpfllllllllll
```

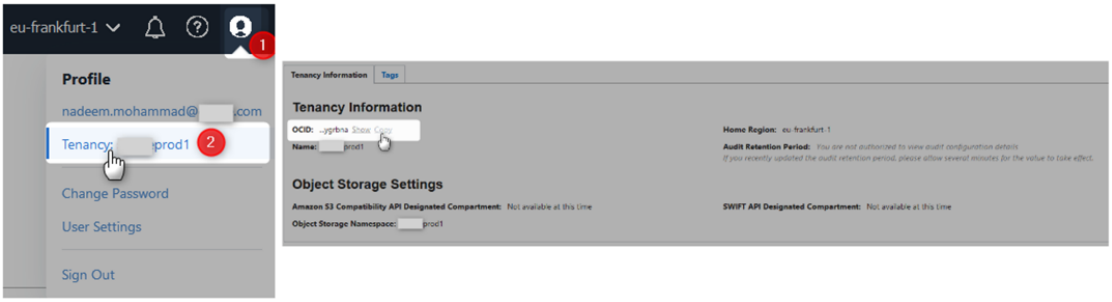
4. Obtain the **User OCID** as follow :



5. Enter **Tenancy OCID** when prompted:

```
Enter a tenancy OCID: ocid1.tenancy.oc1..xxxxxxxxxxxxxxxx4bp2xinnnndy25ps6csc7yyyyyyyyyyy
```

6. Obtain the **Tenancy OCID** as follow :



7. Enter **Region** when prompted**:** `27 (eu-frankfurt-1)`

8. **Generate API Keys** as prompted:

   Do you want to generate a new RSA key pair? (If you decline you will be asked to supply the path to an existing key.) [Y/n]: **Y**

   Enter a directory for your keys to be created [\Users\nadeem\.oci]:

   Enter a name for your key [oci_api_key]:

   Public key written to: \.oci\oci_api_key_public.pem

   Enter a passphrase for your private key (empty for no passphrase): **N/A**

   Private key written to: \Users\nadeem\.oci\oci_api_key.pem

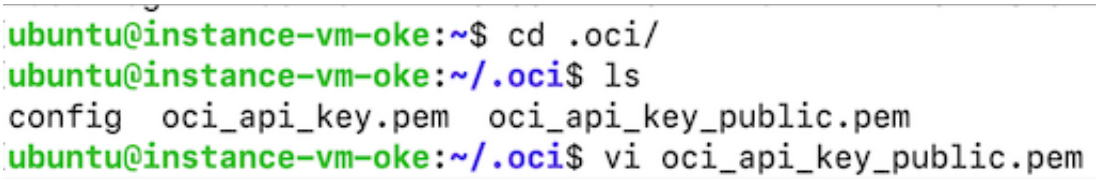
   Fingerprint: b2:04:c3:ee:22:d0:85:83:b6:fa:24:9e:93:2f:c5:27

   Config written to \Users\nadeem\.oci\config

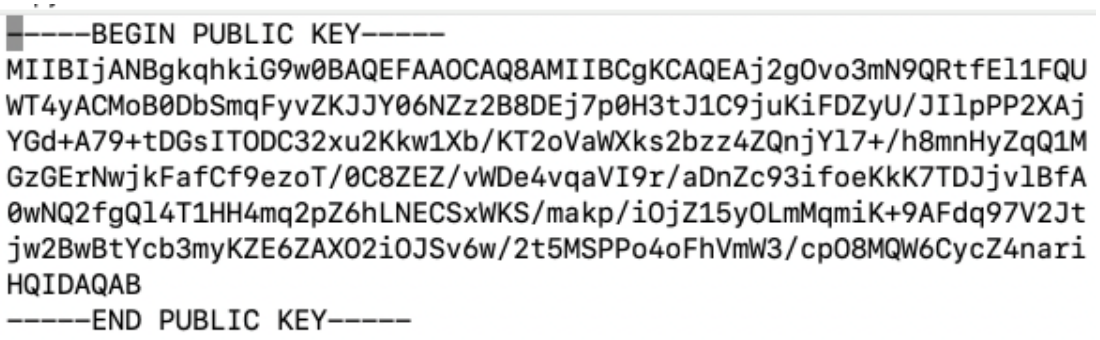
9. Copy **public key**

   - **Execute the following command:**

```
ubuntu@instance-vm-oke:~$ cd .oci/

ubuntu@instance-vm-oke:~/.oci$ 1s

config oci_api_key.pem   oci_api_key_public.pem

ubuntu@instance-vm-oke:~/.oci$ vi oci_api_key_public.pem
```



   - Copy the file

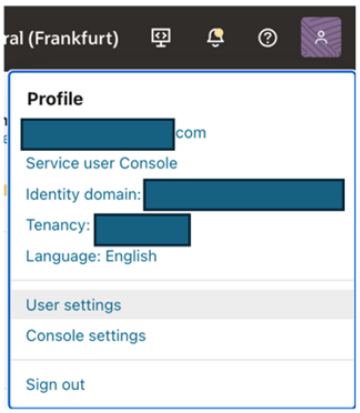


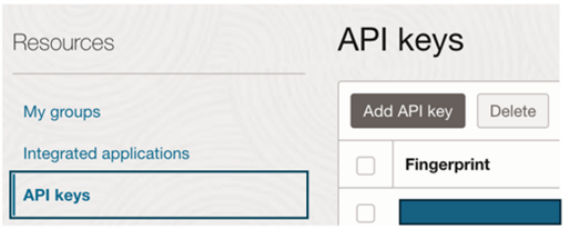
   - Exit the file

```
Press ESC and type :q
```

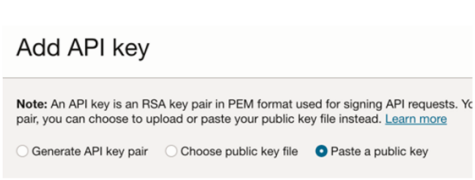
10. Upload **Public key**

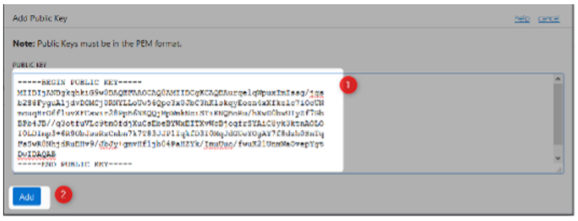
- In the OCI Console **Navigate to User Setting >** Click **API keys > Add API key**



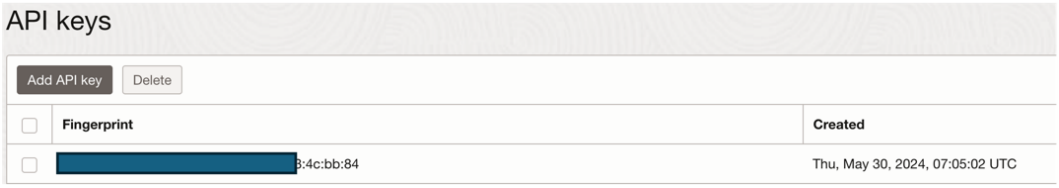


- Choose **Paste a public key > Paste the public key** copied earlier





- **Validate** public key added



- **Validate** the OCI config file with the following command:

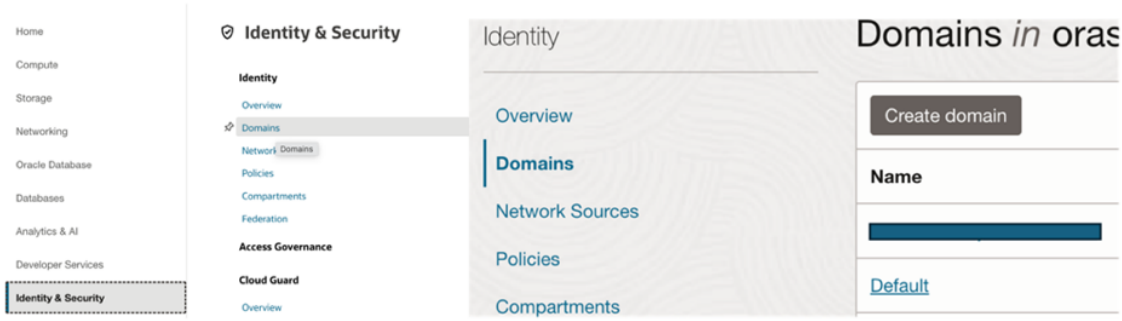
```
oci os ns get
```

```
PS C:\WINDOWS\system32> oci os ns get
{
  "data": "prod1"
}
```

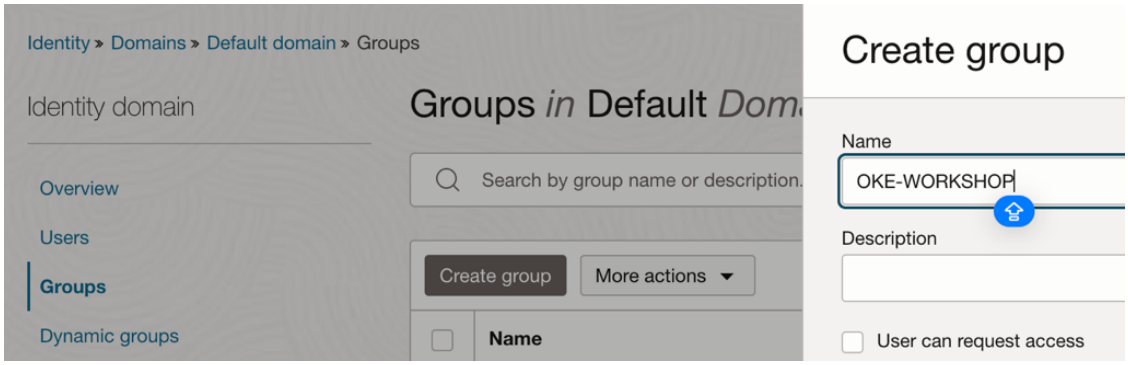
Note : Wait few minutes for the Key to be updated before moving to Task 8

Task 8: Set Up Policy to manage OCI Resources

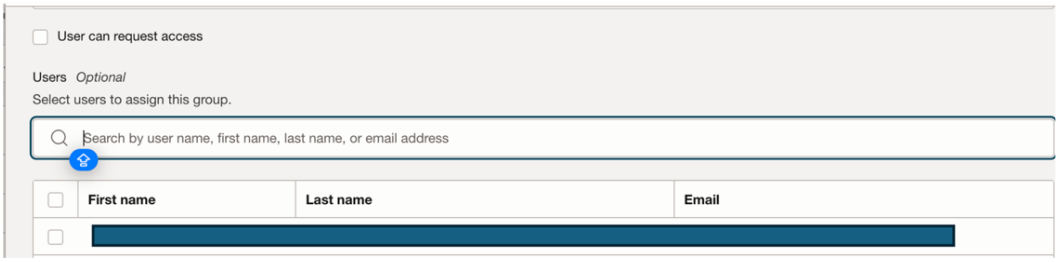
1. Navigate to **Identity & Security** > **Domains** > **Click Default** > **Groups** > **Click Create Group** and **Add User** to the Group



2. **Create Group**

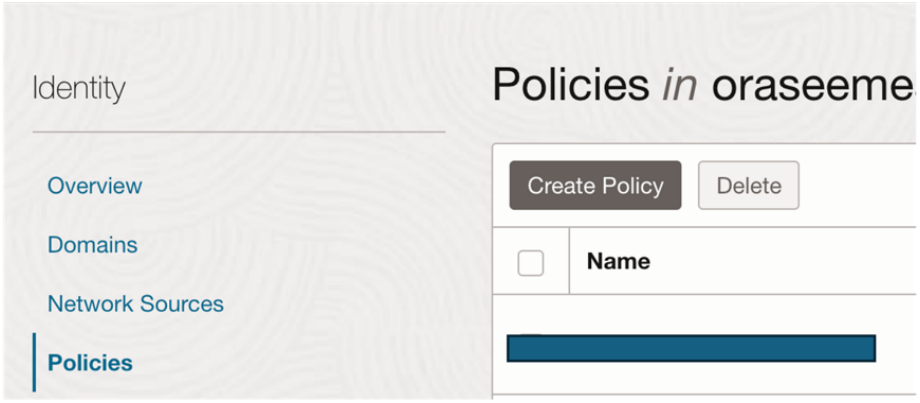


   - Add the User to the Group via the Create group tab**:** Search **Username >** checkbox **the username**



3. **Create Policy**

   - Navigate to **Identity & Security** > **Domains** > **Policies > Create Policy**



   - **Add Policy :**

     ```
     Allow group <GroupName> to manage all-resources in tenancy <Tenancy Name>
     ```

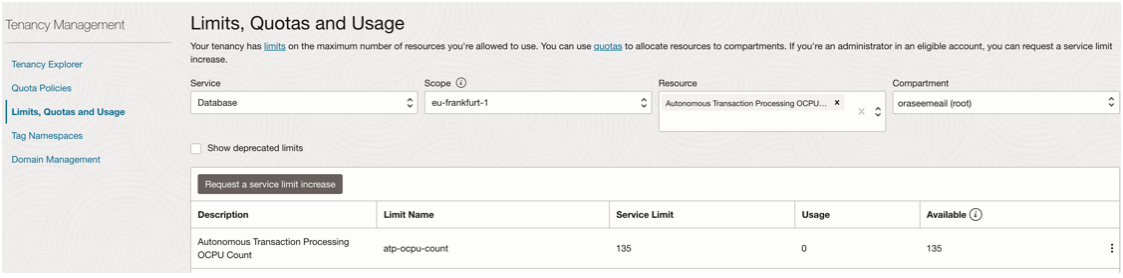
     Change the view to manual builder
     Notes: Replace `<GroupName>` with the newly created group and `<CompartmentName>` with the root compartment name for the purpose of this lab.

   - This policy is broad for simplicity; consider refining it for real-world usage.

Task 9: View the Tenant Service Limits

1. Ensure OCI account has adequate limits, particularly for Autonomous Database configurations.

2. Navigate to **Governance & Administration > Limits, Quotas and Usage > Service Database > Scope eu-frankfurt-1 > Resource Autonomous Transaction Processing OCPU > Count > 1**

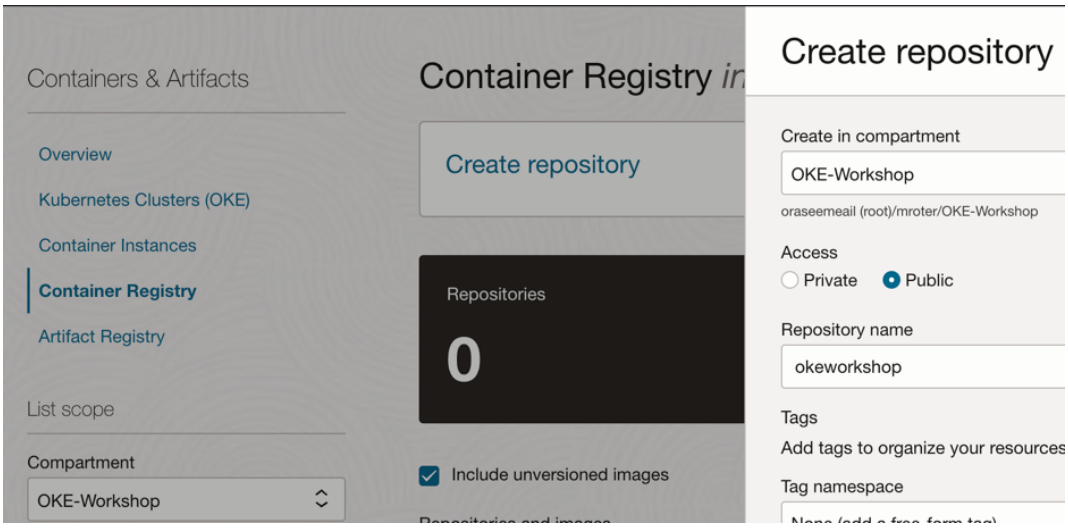


Task 10: Running Containers Locally with Docker

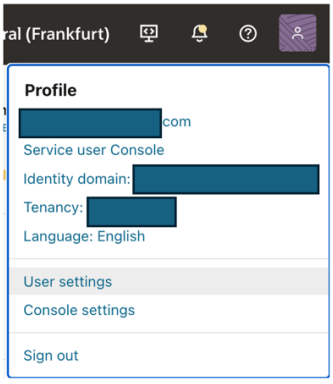
Introduction

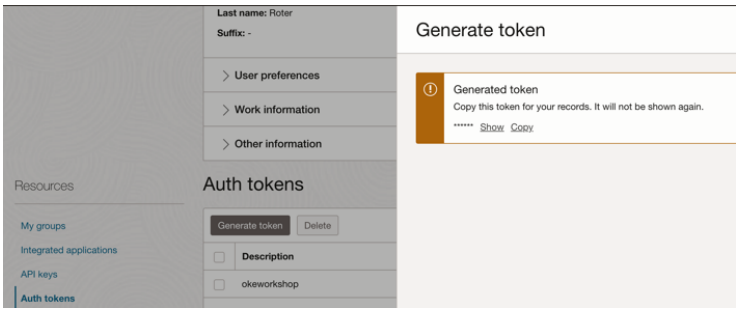
This Task demonstrate how to use, build the code and microservices on the created VM and push them to OCI Container Registry and run them using Docker Compose.

1. Create **Container Registry:**
    - Navigate to **Developer Services** > **Container Registry** > Click **Create Repository** > Select **root compartment** > **Access Type Public** > **Specify Repository name**
    - Click **Create**



2. Generate **Auth Token:** Click **Profile** on the top-right corner > **User Settings** > **Auth Tokens**, generate a token and copy it for future use.





3. **Staging Docker Images Locally:**

   - Login to the Repository by Using the following command

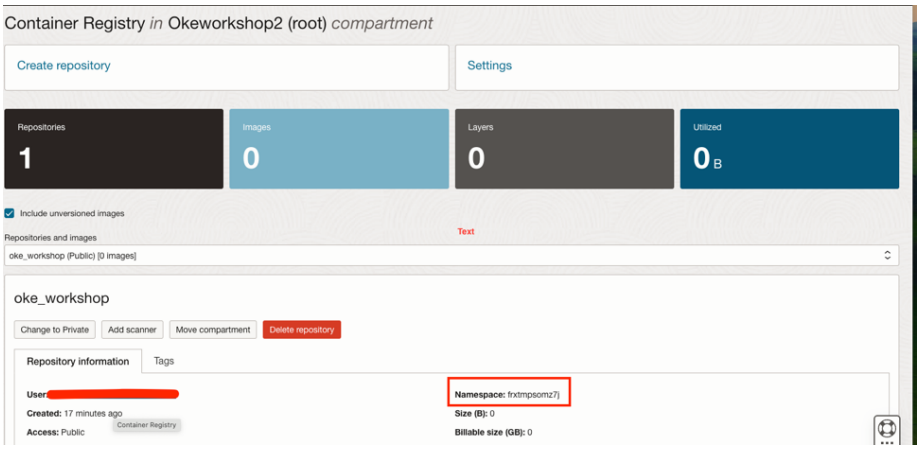
     `docker login <region_code>.ocir.io`

     Example for Frankfurt region - `docker login fra.ocir.io`

     - User Name

       `<Registry-namespace>/default/<username>`

       Example - `froqjg8h9ftr/default/user@domain.com`



     - Password: **Provide the auth token when prompted**

Task 11: Build and Deploy Using Docker

1. Create **folder "**`oci_workshop`**"**

   ```
   mkdir oci_workshop
   cd oci_workshop
   ```

2. Clone **Git_Hub Folder with the following command:**

   ```
   git clone https://github.com/oci-oke-workshop-il/oci-cloudnative-ext.git
   ```

3. Go to **SRC folder and validate the following microservices exists:**

```
ke:~/oci-cloudnative-internal/src$ ls -l

 ubuntu 5927 Nov  3 12:01 README.md
 ubuntu 4096 Nov  3 12:01 api
 ubuntu 4096 Nov  3 12:01 assets
 ubuntu 4096 Nov  3 12:01 carts
 ubuntu 4096 Nov  3 12:01 catalogue
 ubuntu 4096 Nov  3 12:01 dbtools
 ubuntu 4096 Nov  3 12:01 docs
 ubuntu 4096 Nov  3 12:01 edge-router
 ubuntu 4096 Nov  3 12:01 events
 ubuntu 4096 Nov  3 12:01 fulfillment
 ubuntu 4096 Nov  3 12:01 functions
 ubuntu 4096 Nov  3 12:01 load
 ubuntu 4096 Nov  3 12:01 orders
 ubuntu 4096 Nov  3 12:01 payment
 ubuntu 4096 Nov  3 12:01 storefront
 ubuntu 4096 Nov  3 12:01 user
```

4. **Build Docker images as follow :**

   - **Build docker for "api" microservice using:**

     ```
     docker build -t give_name_of_docker_image:version .
     ```

     Example: `docker build -t mushop_api:v1 .`

   - **Tag the newly created images as follow:**

     ```
     docker tag give_name_of_docker_image:version

     oci_region/registry_namespace/registry_name/name_of_docker_image:version
     ```

     Example:

     ```
     docker tag mushop_api:v1
     fra.ocir.io/froqjg8h9ftr/oke_workshop/mushop_assets:v1
     ```

   - **Push Docker images to the OCI container registry as follow:**

     ```
     docker push oci_region/registry-
     namespace/registry_name/name_of_docker_image:version
     ```

     Example:

     ```
     docker push fra.ocir.io/froqjg8h9ftr/oke_workshop/mushop_api:v1
     ```

5. **Repeat tasks 4-6 for each image to ensure all microservices are uploded**

6. **Validate all images are built by executing the following command:**

   ```
   docker images -a
   ```

LAB 2 – Deploy OKE Cluster with Terraform

Introduction

You will take on the persona of an Operations Engineer. You will initiate the Oracle cloud environment that will be used to create and deploy your microservices applications. This environment will be contained within a cloud Compartment, and communication within the Compartment will be via a Virtual Cloud Network (VCN). The Compartment and VCN will isolate and secure the overall environment. You will deploy the Oracle Cloud Infrastructure Container Engine for Kubernetes(OKE).

Task 1: Go to Terraform Directory

1. Change directory to the `**Terraform-templates**` folder within the cloned repository:

   ```
   cd oci-cloudnative-ext/Terraform-templates
   ```

   Explanation: The `cd` command navigates to the `Terraform-templates` directory where you will configure the deployment templates.

Task 2: Edit `terraform.tfvars`

1. Open `**terraform.tfvars**` using a text editor:

   ```
   vi terraform.tfvars
   ```

   Explanation: This file contains the variables required for your Terraform deployment. You need to modify it to specify details about your OCI environment.

2. Update **the following variables:**

   `compartment_id`: This is the OCID of the compartment where you want to deploy your cluster.

   `availability_domain`: Specify the availability domain (e.g., `AD-1`) where OKE cluster will be deployed

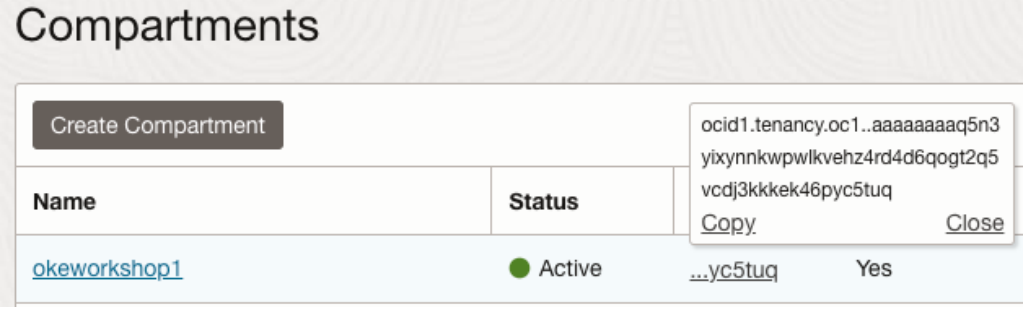
   `tenancy_ocid` : This is the OCID of your tenancy.

   Example of edited `**terraform.tfvars**`:

   compartment_id = `ocid1.compartment.oc1..exampleuniqueID`
   availability_domain = `XXXX:EU-FRANKFURT-1-AD-1`
   tenancy_ocid = `ocid1.tenancy.oc1..exampleuniqueID`

   Explanation: Replacing these variables with the correct OCIDs ensures that Terraform knows where to deploy your OKE cluster within your OCI environment.

3. **Obtain the `compartment_id` (OCID)**
    - Navigate to **Identity & Security** > **compartments**

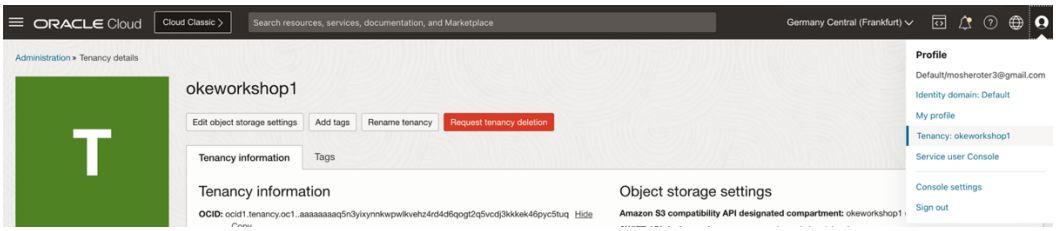


4. **Obtain the `availability_domain'**
    - Navigate to **Compute** > **Instances > Create Instance > Locate the available AD's**

      Note: locate and use the string for the AD in the following format - XXXX:EU-FRANKFURT-1-AD-1

    - **For this LAB, we are using AD1**

5. **Obtain the `tenancy_ocid`**
    - Navigate to **Profile** > Click on **Tenancy name > Locate tenancy OCID**



Task 4: Apply Terraform Configuration to deploy OKE Cluster

1. Initialize the directory:

   ```
   terraform init
   ```

   Explanation: This command initializes your Terraform configuration by downloading the required plugins and preparing the environment.

2. **Execute the Terraform plan command to preview changes:**

   ```
   terraform plan
   ```

   Explanation: This Task allows you to check for potential issues and confirm that the deployment configuration is correct.

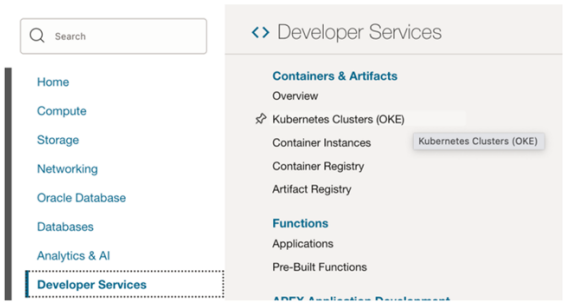
3. **Deploy the OKE cluster**

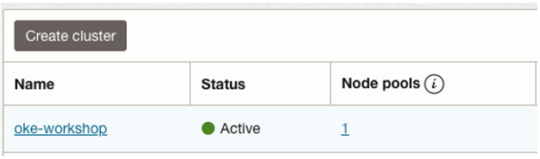
   ```
   terraform apply
   ```

   Explanation: This command starts the deployment of the OKE cluster. Review the changes when prompted and type `yes` to confirm.

Task 5: Validate Successful Deployment

1. Navigate to Developer Services > Kubernetes Clusters (OKE) > Click Cluster name > Review **cluster details** and





   Explanation: Verifying in the OCI Console ensures that the cluster has been deployed as expected.

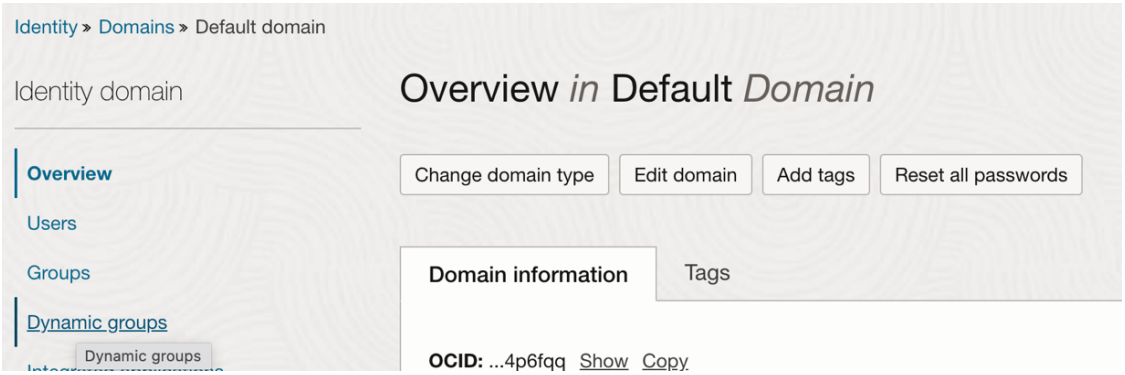
LAB 3 - OCI Vault integration with OKE

Introduction

This lab provides the relevant configuration Tasks for Vault service integration with OCI services (e.g. Oracle Autonomous database)

Task 1: Create Dynamic Group to group the OKE cluster nodes to automaticlly assign policies to the group

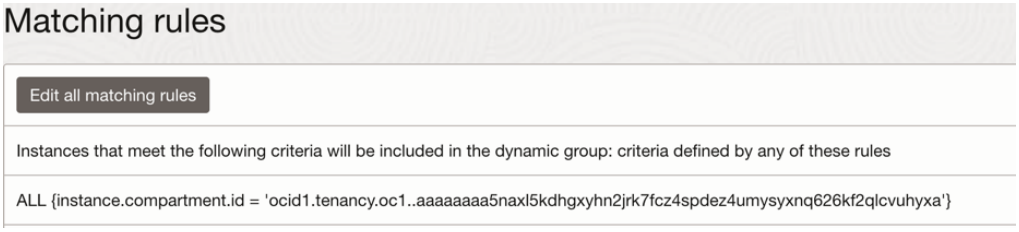
1. Navigate to **Identity & Security** > **Domains** > Click **Domain Name >** Click **Dynamic groups**



2. Click **Create dynamic group** and fill the following information
   - Name : `OKE-DG`
   - Description : `Include OKE nodes in the same group`
   - **Matching rules > Rule 1 > add the following rule :**

     ```
     ALL {instance.compartment.id = 'insert compartment ocid'}
     ```

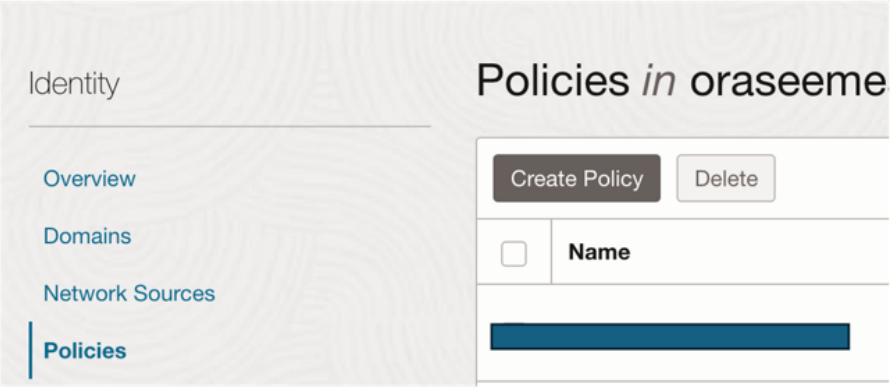
     Example



   - Click **Create**

Task 2: Create Policy to allow the OKE cluster access the Vault resource

1. Navigate to **Identity & Security** > **Policies > Create Policy**



2. Add **Policies:**

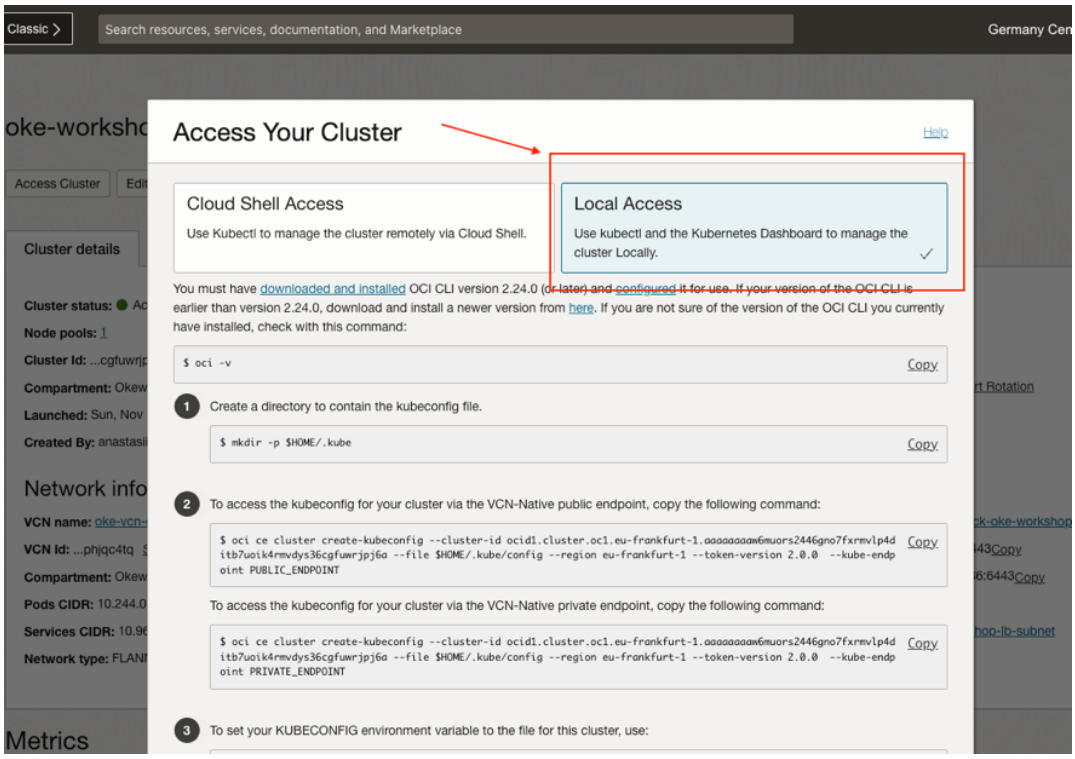
```
Allow dynamic-group <Dynamic-group-name> to read secret-bundles in compartment
id <compartment-ocid>

Allow dynamic-group <Dynamic-group-name> to use keys in compartment id
<compartment-ocid>

Allow dynamic-group <Dynamic-group-name> to manage secret-family in compartment
id <compartment-ocid>
```

Notes:
Replace `<Dynamic-group-name>` with the newly created dynamic group and `<compartment-ocid>` with the root compartment ocid for the purpose of this lab.

3. Click **Create**

Task 3: Install External Secrets Operator on OKE

1. In OCI console Navigate to the **Developer Services** > **Kubernetes clusters (OKE)**.
2. Choose the **OKE cluster** you have previously created with terraform.
3. Follow the instruction for **Quick Access** > **Local Accsess** > Do the Action from the VM



4. **Execute the following commands in the VM Terminal:**

```
helm repo add external-secrets https://charts.external-secrets.io
helm install external-secrets \
 external-secrets/external-secrets \
 -n external-secrets \
 --create-namespace
```

Task 4: Configuring ESO for use with OCI Vault

1. **Create a oci-secret-store.yaml:**

   - **Open editor in the VM terminal using VI command and create the following file:**

```
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
 name: workshop-vault
spec:
 provider:
  oracle:
   vault: <vault-OCID> #Replace <vault-OCID> with the Vault ocid created by the terraform
   region: eu-frankfurt-1
```

   - **Execute the following command to deploy the yaml in the oke cluster**

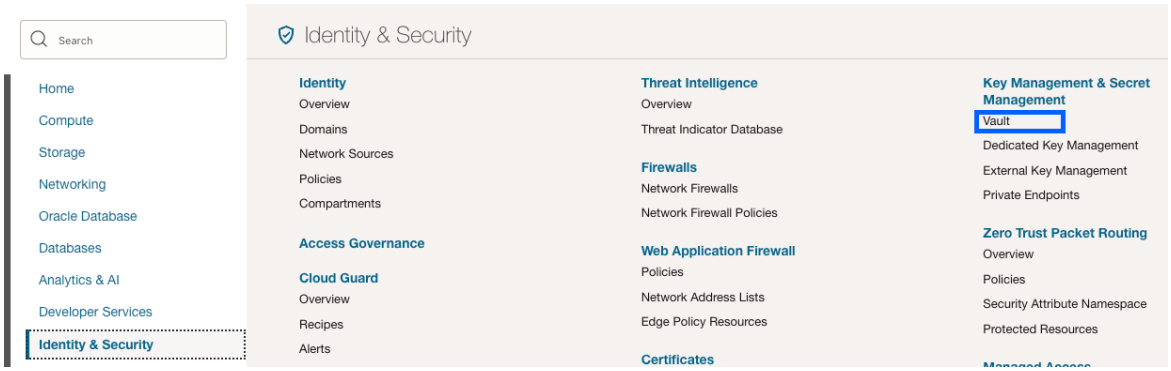
```
kubectl create -f oci-secret-store.yaml
```

- **Validate that SecretStore resource created in the cluster:**

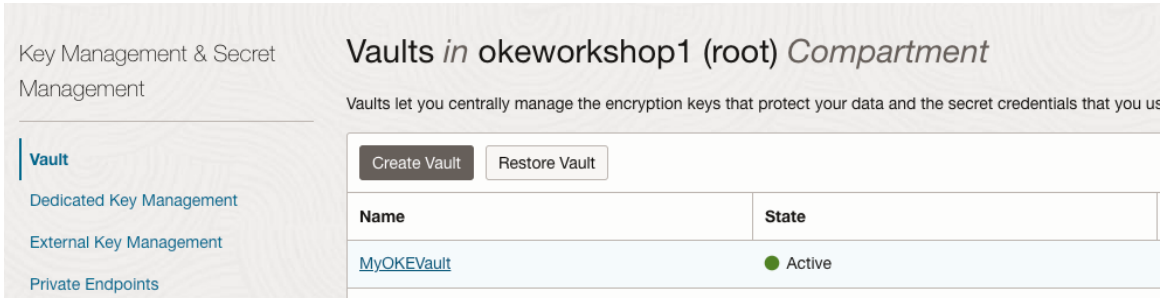
```
kubectl get secretstore -A
```

Task 5: Create secrets in OCI Vault

1. Navigate to **Identity & Security** > **Vault**

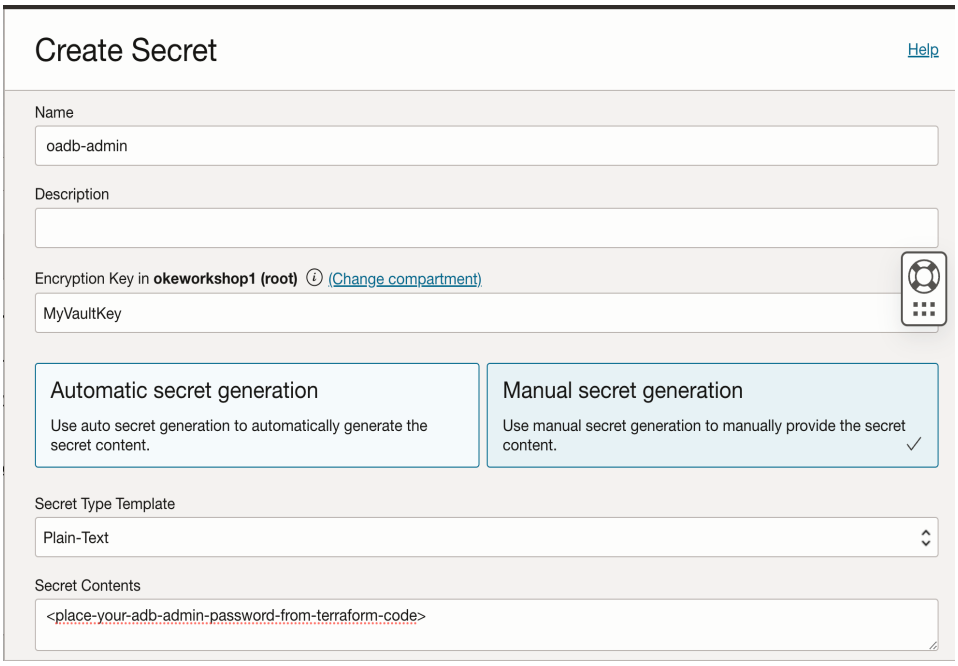


2. Validate **compartment name (root)** and Click on the **Vault instance** created by terraform



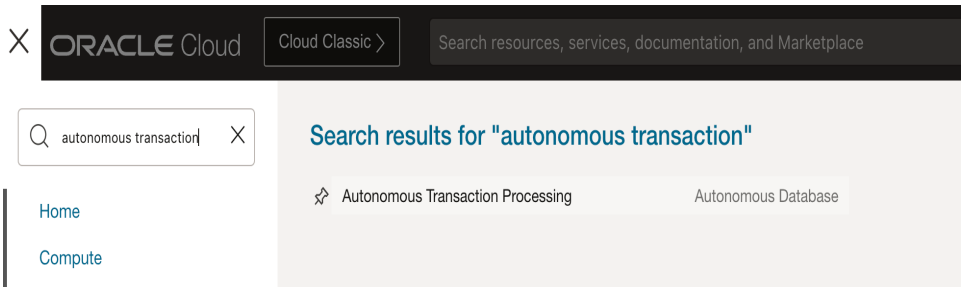
3. Click **Secrets**
4. **Create the following secrets :** `oadb-connection,oadb-wallet & oadb-admin`

   - Create **oadb-admin** secret as follow:
     - Name: `oadb-admin`
     - Compartment: `Use the root for the purpose of this lab`
     - Encryption Key: `Choose from drop-down list`
     - Secrets contents: **Challenge yourself here < place-your-adb-admin-password-from-terraform-code>**

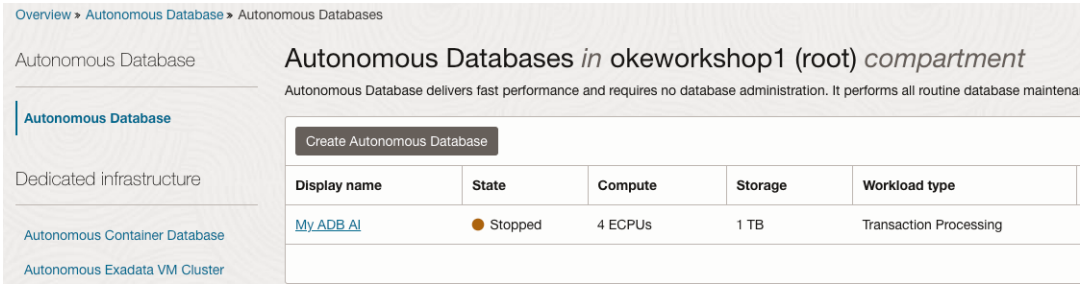


- Create **oadb-connection** secret as follow:
  - Name: `oadb-connection`
  - Compartment: `Use the root for the purpose of this lab`
  - Encryption Key: `Same as previous secret`
  - Secret Type Template: `< plain-text >`
  - Secrets contents: `Replace this value with the value of the adb-tnsname` **(refer to next section Obtain the adb-tnsname to identify the value)**
  - Create secret: **Click it when you finished to fill the information above)**

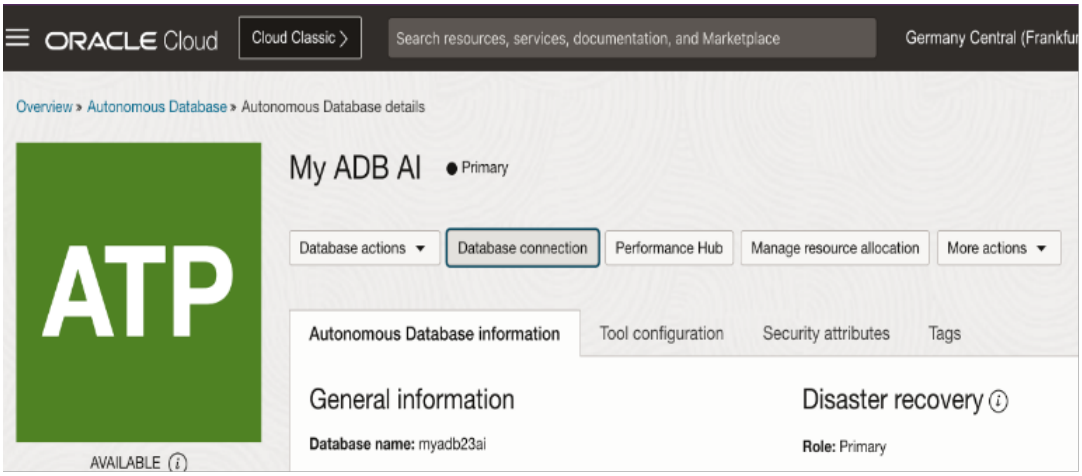
- Obtain the **adb-tnsname**
  - Duplicate the OCI portal window > Navigate to the **Search bar** and search for : Autonomous transaction



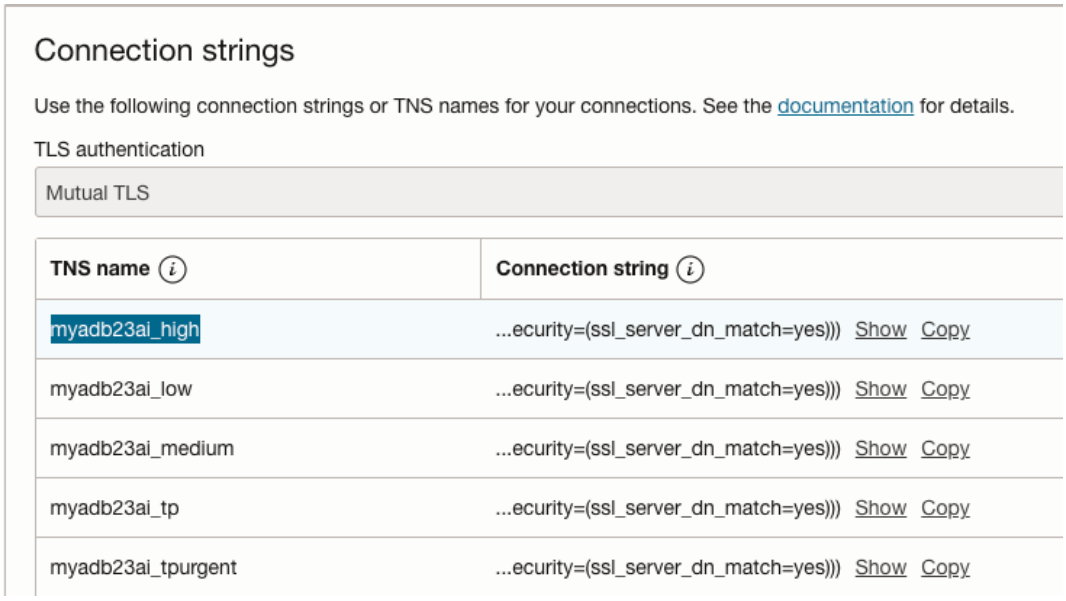
- Choose **the Databased created by the terraform**



- Click on **DB connection**



- In the Database connection view Scroll down to **Connection Strings** > Copy **TNS name_high** (Do not copy the connection string)

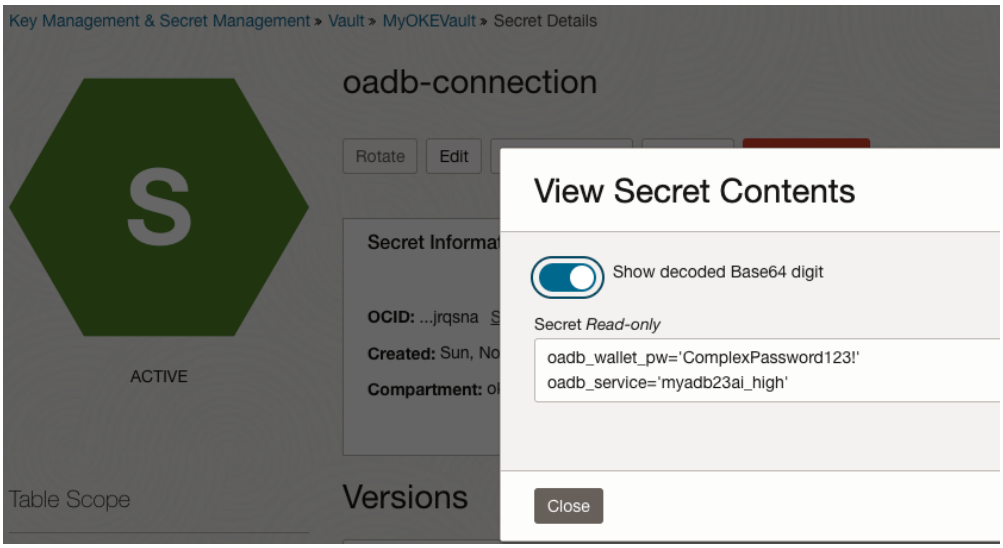


Do not close this window yet!

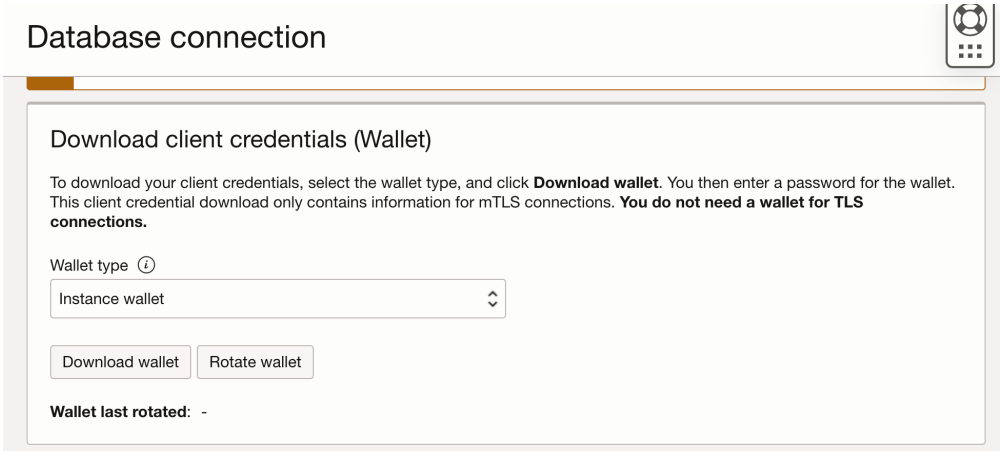
- Go to previous **Section** and use the copied **TNS name** to fill the **Secret Type Template** and Finish the Tasks to create the **oadb-connection**

Example

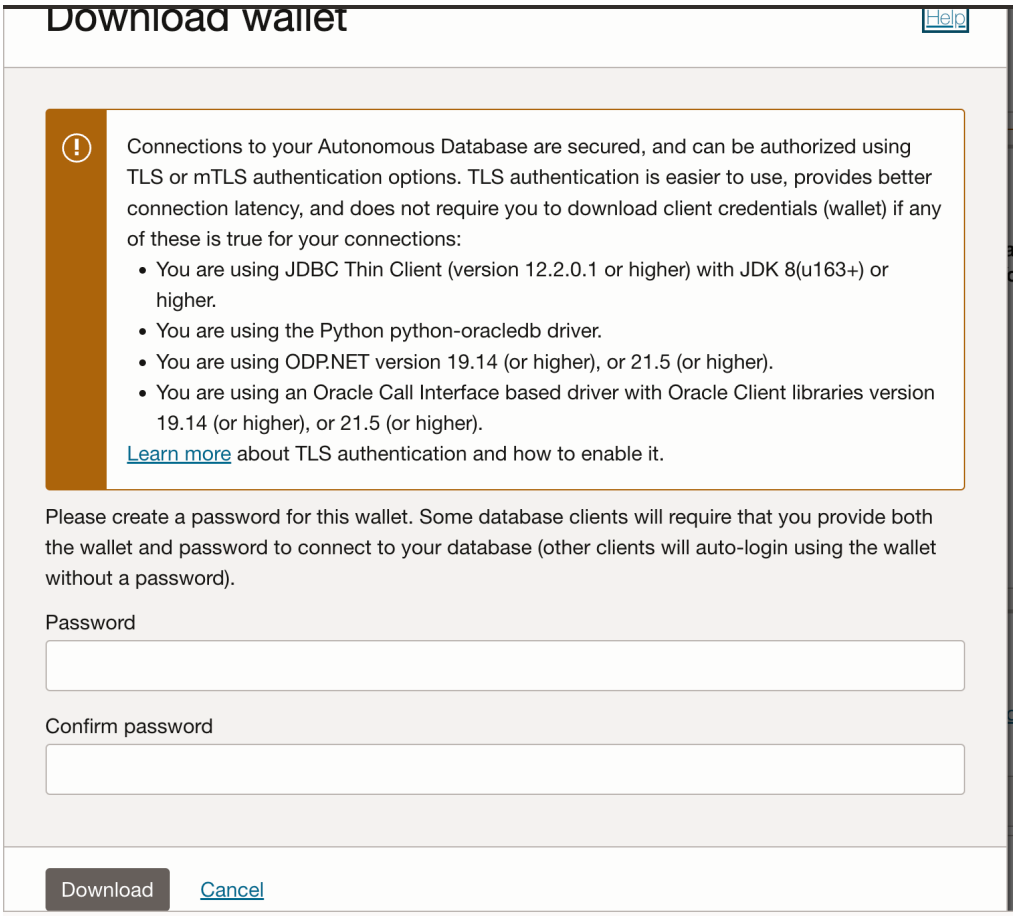
```
oadb_wallet_pw=’your-wallet-password'
oadb_service='myadb23ai_high' #Your TNSName
```



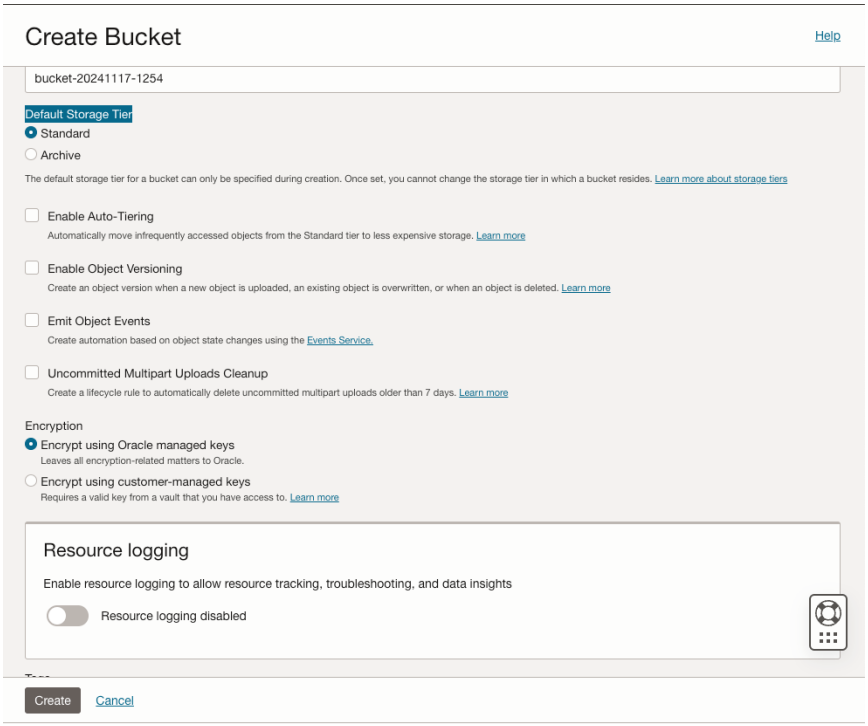
- Download **Client Credentials (Wallet)**
- Go back to **DB connection view page**
- Click **Download wallet**



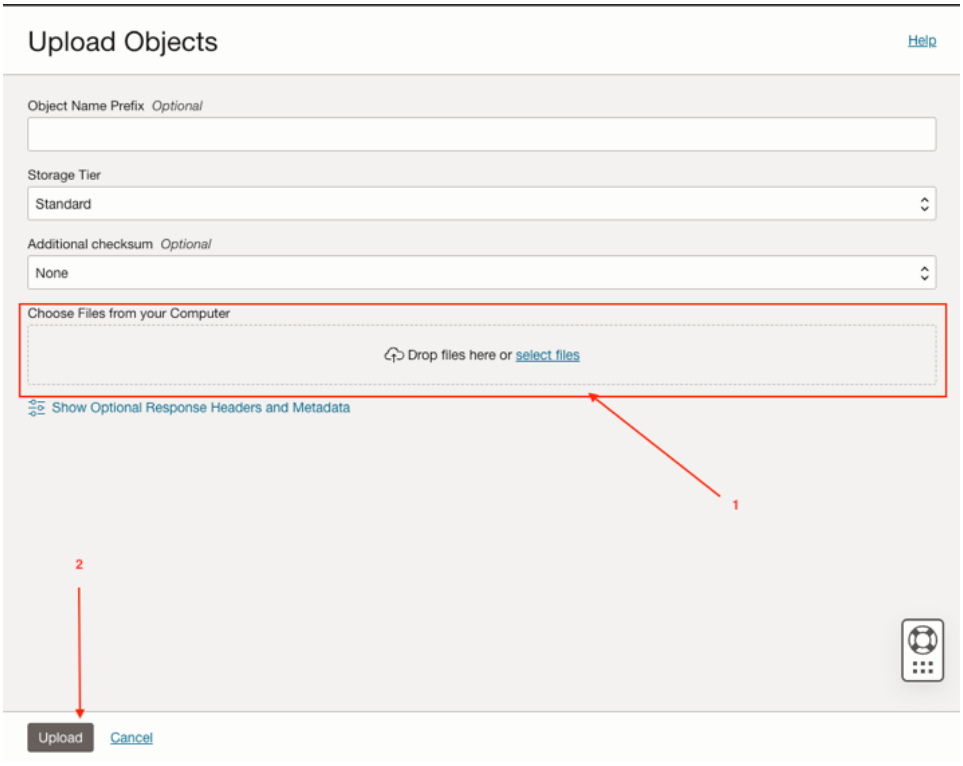
- Set Password and click **Download**



- **Upload the file to the virtual machine** In the OCI console Navigate to **Storage** > **Buckets** > click **Create Bucket**
  - Name: `my-bucket`
  - Default Storage Tier : `Standard`
  - Keep the default values for any additional values
  - Click **Create**



- **Upload the Zip folder** to the bucket as describe in picture below :



- **From the VM execute the following command:**

  ```
  oci os object bulk-download --bucket-name <BUCKET NAME> --download-dir
  <The DIR where you want to save the folder>
  ```

  Example:

  ```
  oci os object bulk-download --bucket-name my-bucket --download-dir
  ```

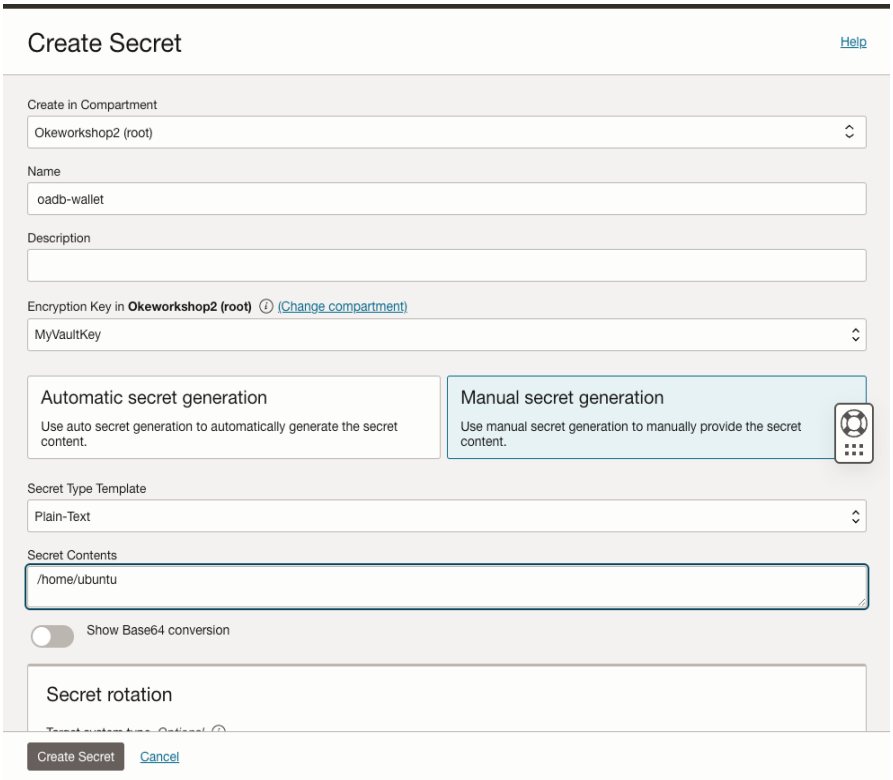
- **Unzip the folder locally from the VM with command:**

  ```
  unzip XXXX.zip
  ```

- **Update the oadb-wallet file: Copy the path of the directory where you save the unzipped files (execute the command pwd) .**

- **Create oadb- wallet secret as follow:**

  - Name: `oadb-wallet`
  - Compartment: `Use the root for the purpose of this lab`
  - Encryption Key: `Choose from drop-down list`
  - Secret Type Template: `< plain-text >`
  - Secrets contents: `The path should be the value for the oadb-wallet secret.  For Example: /home/ubuntu`



Task 6: Create ExternalSecret

1. **Create file named oadb-admin-secret.yaml as follow:**

```
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: oci-secret-admin
spec:
  refreshInterval: 0.03m
  secretStoreRef:
    kind: SecretStore
    name: workshop-vault # Must match SecretStore name deployed on the cluster
  target:
    name: oadb-admin # Must match adb secret name downloaded in Task 4-section 9
    creationPolicy: Owner
  data:
  - secretKey: key
    remoteRef:
      key: oadb-admin
```

2. **Create file named oadb-connection-secret.yaml as follow:**

```
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: oci-secret-connection
spec:
  refreshInterval: 0.03m
  secretStoreRef:
    kind: SecretStore
    name: workshop-vault # Must match SecretStore name deployed on the cluster
  target:
    name: oadb-connection # Must match adb secret name downloaded in Task 4-section 9
    creationPolicy: Owner
  data:
  - secretKey: key
    remoteRef:
      key: oadb-connection
```

3. **Create file named oadb-wallet-secret.yaml as follow:**

```
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: oci-secret-wallet
spec:
  refreshInterval: 0.03m
  secretStoreRef:
    kind: SecretStore
    name: workshop-vault # Must match SecretStore name deployed on the cluster
  target:
    name: oadb-wallet # Must match adb secret name downloaded in Task4-section 9

    creationPolicy: Owner
  data:
  - secretKey: key
    remoteRef:
      key: oadb-wallet
```

4. **Deploy Secrets Yaml files by executing the following commands :**

```
kubectl apply -f oadb-admin-secret.yaml
kubectl apply -f oadb-connection-secret.yaml
kubectl apply -f oadb-wallet-secret.yaml
```

Task 6: Validate external secrets created

1. **Execute the following commands:**

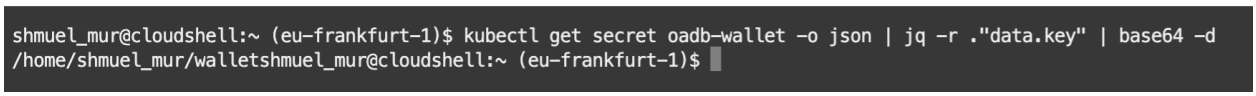
```
kubectl get es
```

```
shmuel_mur@cloudshell:~ (eu-frankfurt-1)$ kubectl get es -A
NAMESPACE   NAME                    STORE           REFRESH INTERVAL   STATUS         READY
default     oci-secret              oke-db-secret   0.03m              SecretSynced   True
default     oci-secret-connection   oke-db-secret   0.03m              SecretSynced   True
default     oci-secret-wallet       oke-db-secret   0.03m              SecretSynced   True
```

`kubectl get secrets` (validate that secrets created and sync with vault)

```
shmuel_mur@cloudshell:~ (eu-frankfurt-1)$ kubectl get secrets

NAME                                              TYPE                 DATA   AGE
oadb-admin                                        Opaque               1      3d1h
oadb-connection                                   Opaque               1      3d1h
oadb-wallet                                       Opaque               1      3d1h
sh.helm.release.v1.oci-kubernetes-monitoring.v1   helm.sh/release.v1   1      3d
```

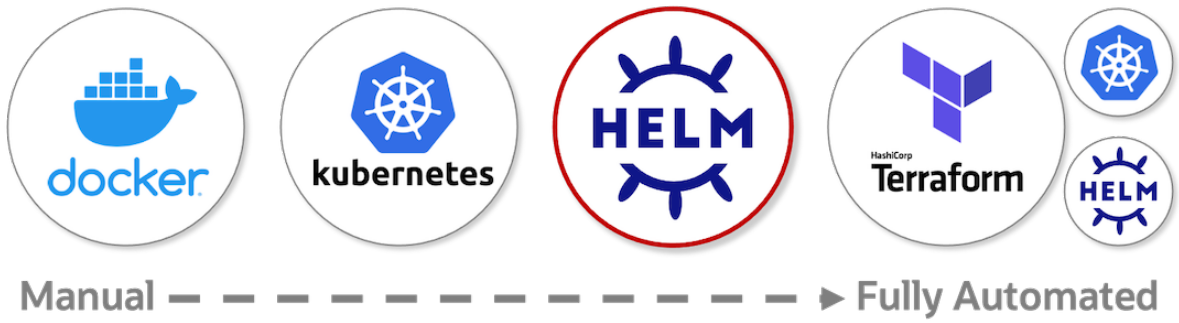
`kubectl get secret oadb-wallet -o json | jq -r ."data.key" | base64 -d`
(validate secrets in OKE synced with value)



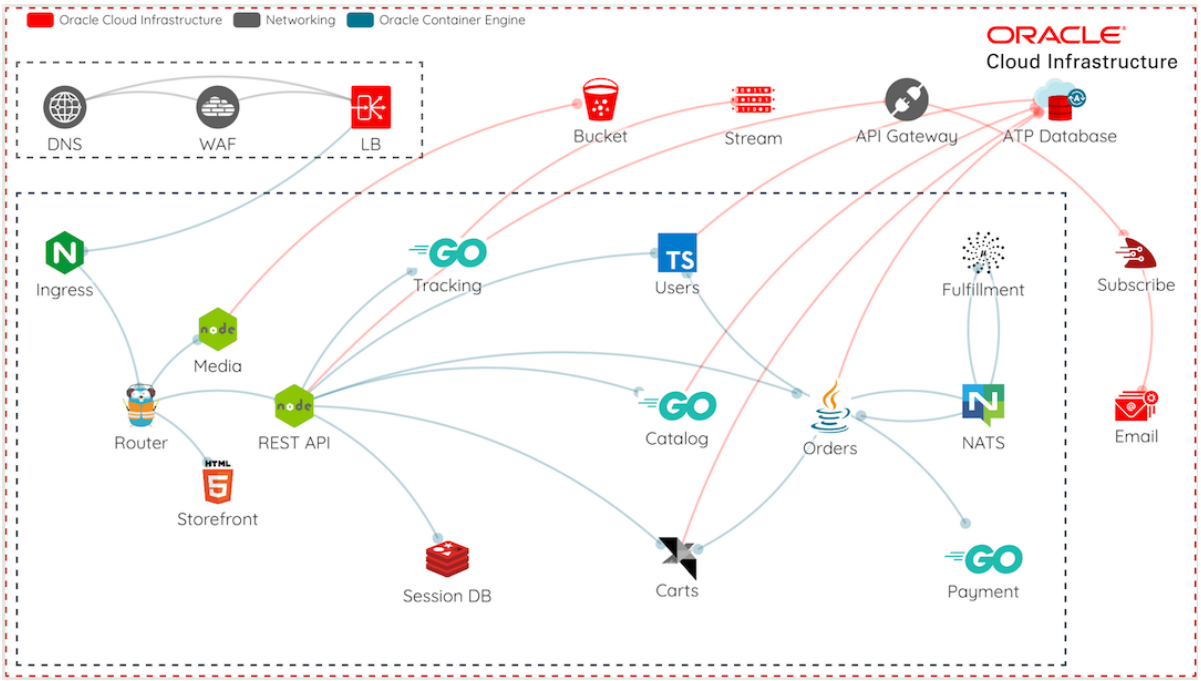
LAB 4 - Install Mushop Using Helm

Introduction

There are four options for deploying MuShop. They range from manual (docker), automated (Helm) to fully automated (Terraform).



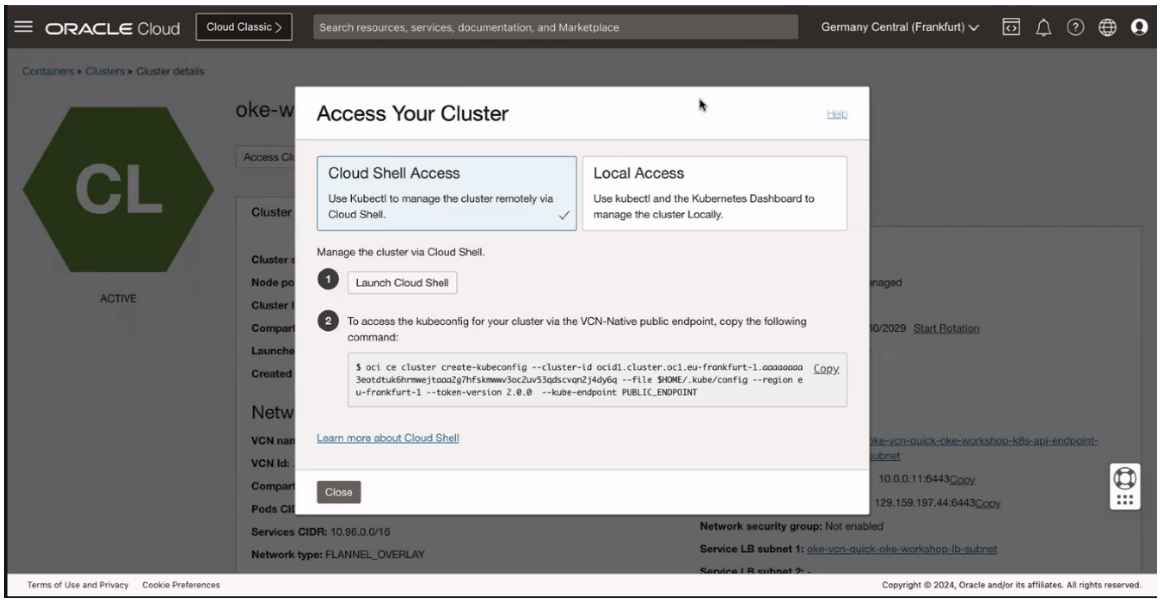
Designing in microservices offers excellent separation concerns and provides developer independence. While these benefits are clear, they can often introduce some complexity for the development environment. Services support configurations that offer flexibility, when necessary, and establish parity as much as possible. It is essential to use the same tools for development to production.



Note: This diagram contains services not covered by these labs.

Task 1: Access the OKE Cluster from Cloud Shell

1. In OCI console Navigate to the Developer Services > Kubernetes clusters (OKE).
2. Choose the OKE cluster you have previously created with terraform.
3. Connect to the OKE cluster from Quick Start with Cloud Shell.



Task 2: Obtain MuShop source code

1. Open Cloud Shell and clone the github repo.

   `git clone` https://github.com/oci-oke-workshop-il/oci-cloudnative-ext.git

   Sample response

   livelabs main

   ```
   Cloning into 'mushop'...
   remote: Enumerating objects: 542, done.
   remote: Counting objects: 100% (542/542), done.
   remote: Compressing objects: 100% (313/313), done.
   remote: Total 15949 (delta 288), reused 424 (delta 200), pack-reused 15407
   Receiving objects: 100% (15949/15949), 17.59 MiB | 33.71 MiB/s, done.
   Resolving deltas: 100% (9557/9557), done.
   ```

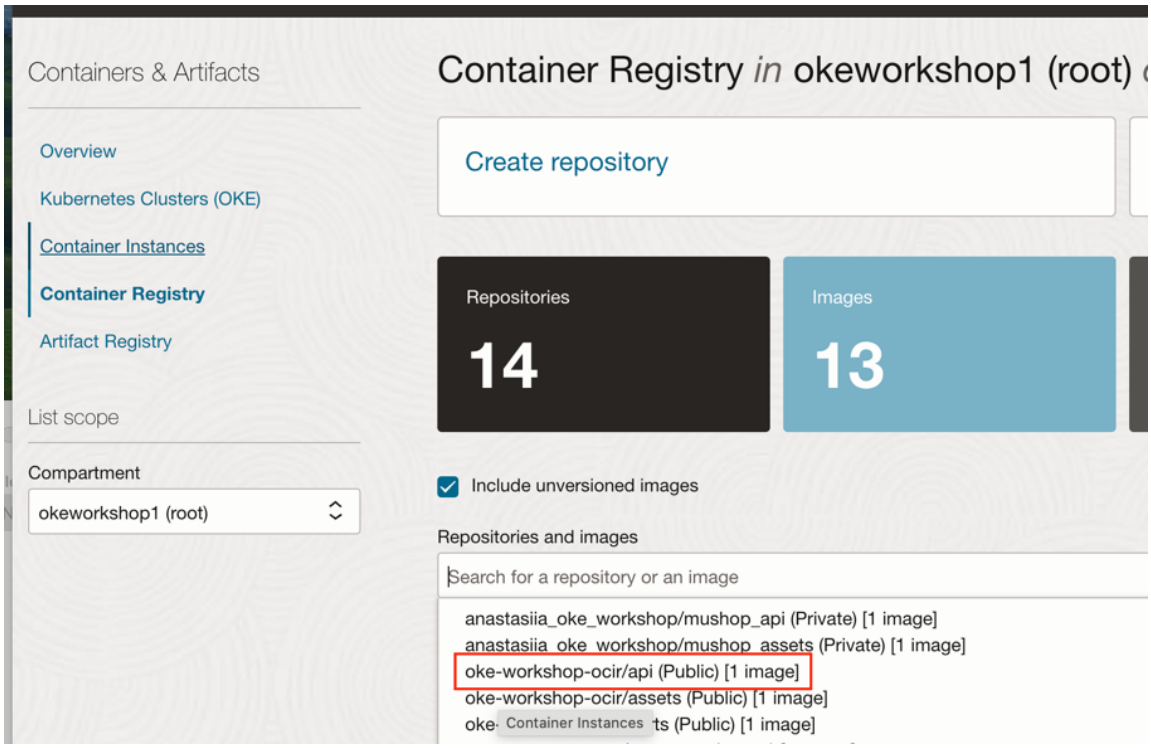
2. Change to the mushop directory containing the Helm charts:

   ```
   cd oci-cloudnative-ext/deploy/complete/helm-chart/mushop/charts
   ```

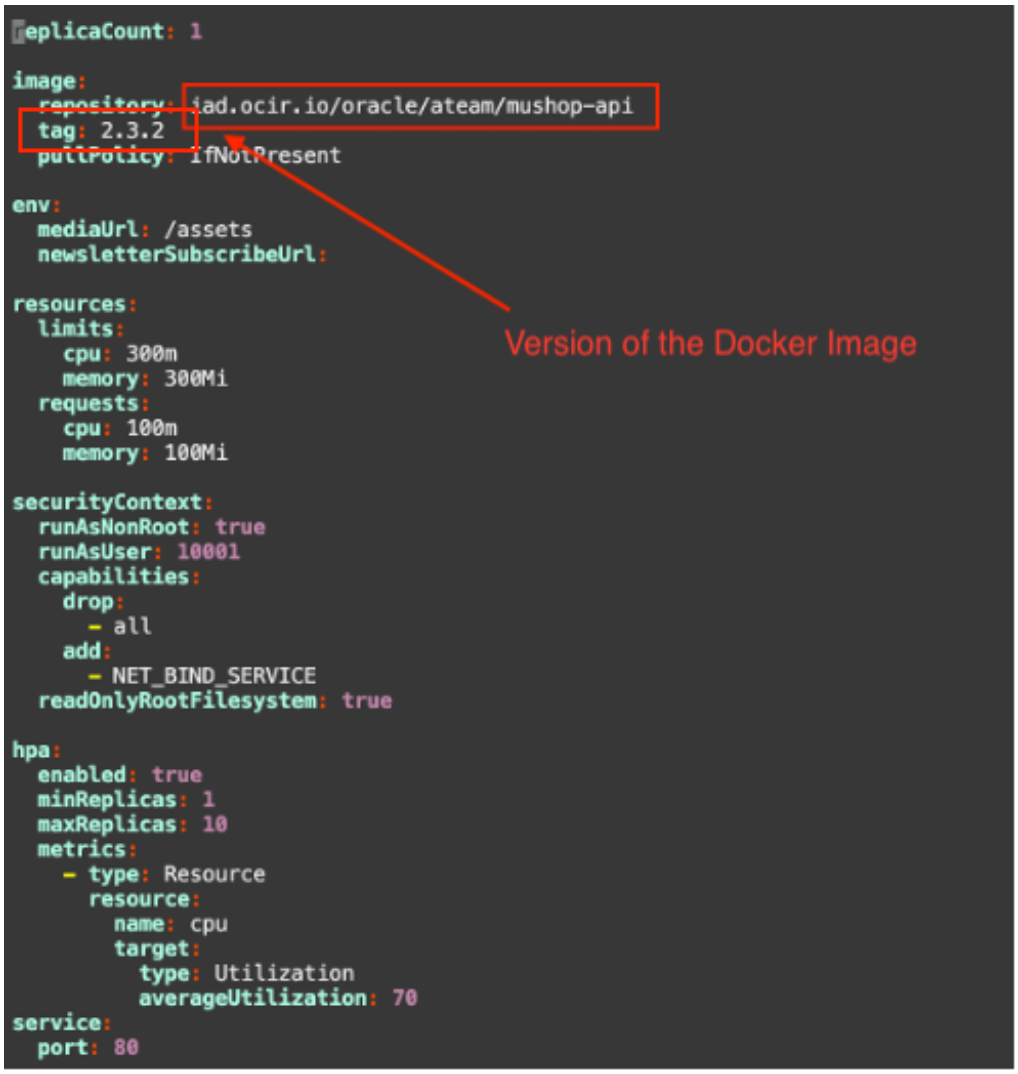
Task 4: Update the Image Repository

In this Task we will Update the image repository in the `**values.yaml**` files for the following microservices - **API, Storefront & Catalogue** . The new value must point to the container repository previously created in Lab 1

1. Enter every folder in `helm-chart/mushop/charts , For example `/api`.
2. Open the `values.yaml` file.
3. Update the `image.repository` field with the full path of the container repository created previously.
4. Update the Tag for the image that you pushed previously, For example v1



Example (`values.yaml` for one of the microservices):



5. **Repeat this process for API, Storefront & Catalogue microservices !!!**

Task 5: Deploy the eCommerce App with Helm

1. Deploy the application by Executing the following command:

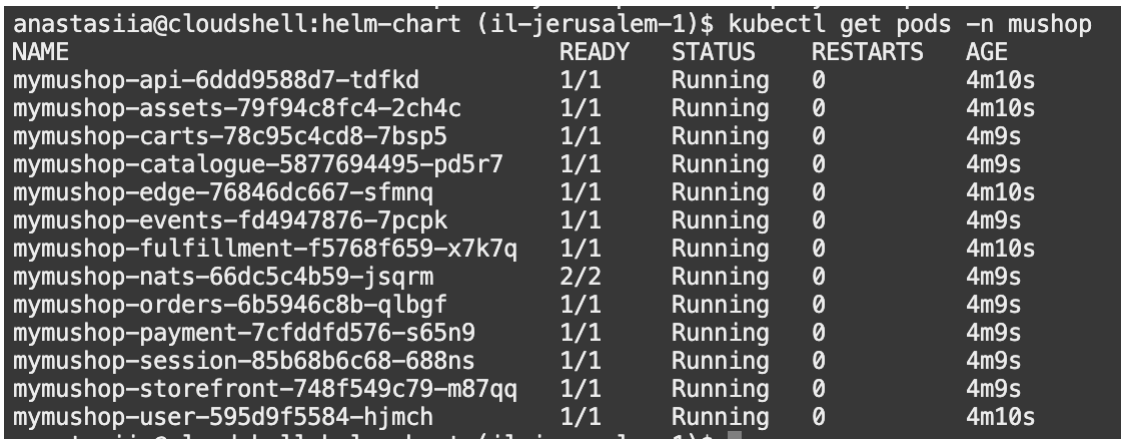
   helm install mymushop mushop \
   --namespace mushop \
   --create-namespace \
   --set global.mock.service=all

2. Please be patient. It may take a few moments to download all the application images

   Execute the following command:

   ```
   kubectl get pods -n mushop
   ```

Sample response to make sure all the pods are EXISTS and in the Running state



LAB 5 - Install Ingress Controller

Prerequisites : Running Kubernetes cluster ; kubectl ; Helm installed.

Task 1: Add NGINX Ingress Helm Repository

3. Execute the following command:

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
```

Task 2: Create a namespace for NGINX Ingress

1. Execute the following command:

```
kubectl create namespace nginx-ingress
```

Task 3: Install NGINX Ingress Controller using Helm

1. Execute the following command:

```
helm install nginx-ingress ingress-nginx/ingress-nginx --namespace nginx-ingress
```

Task 4: Validate installation:

1. Execute the following command:

```
kubectl get pods -n nginx-ingress
```

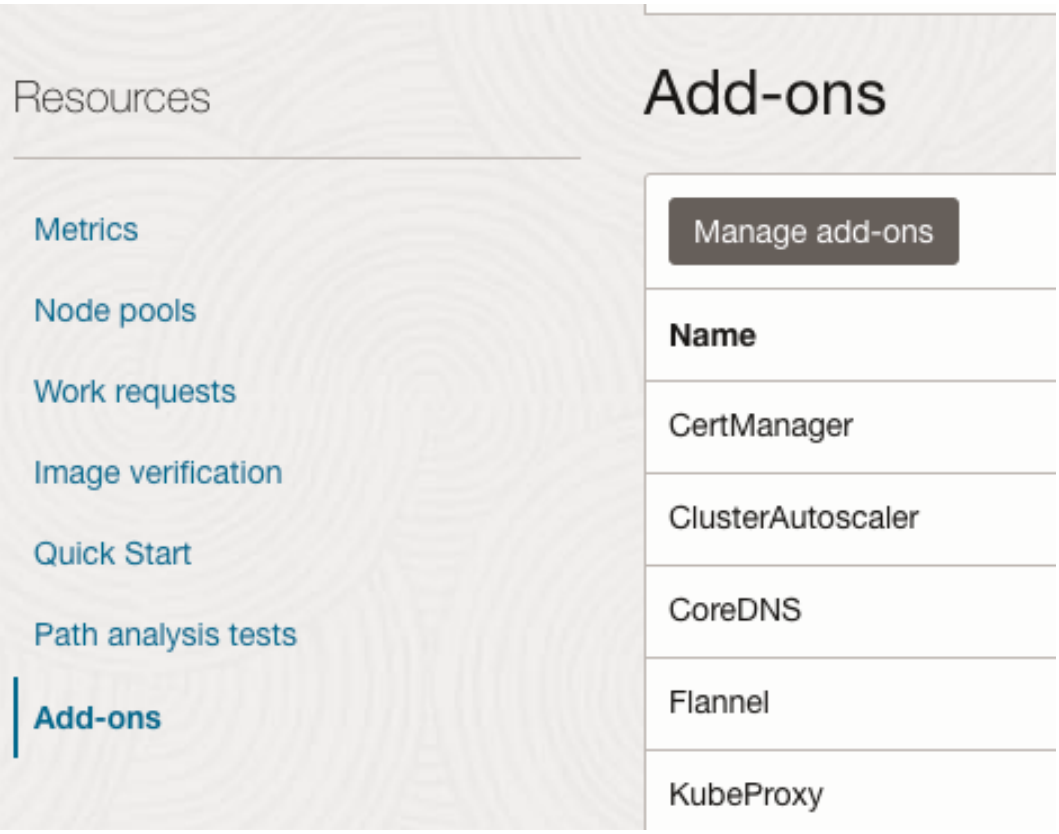
Sample response

```
NAME                                                READY
STATUS    RESTARTS   AGE
nginx-ingress-ingress-nginx-controller-xxxxxx     1/1
Running   0          2m
```

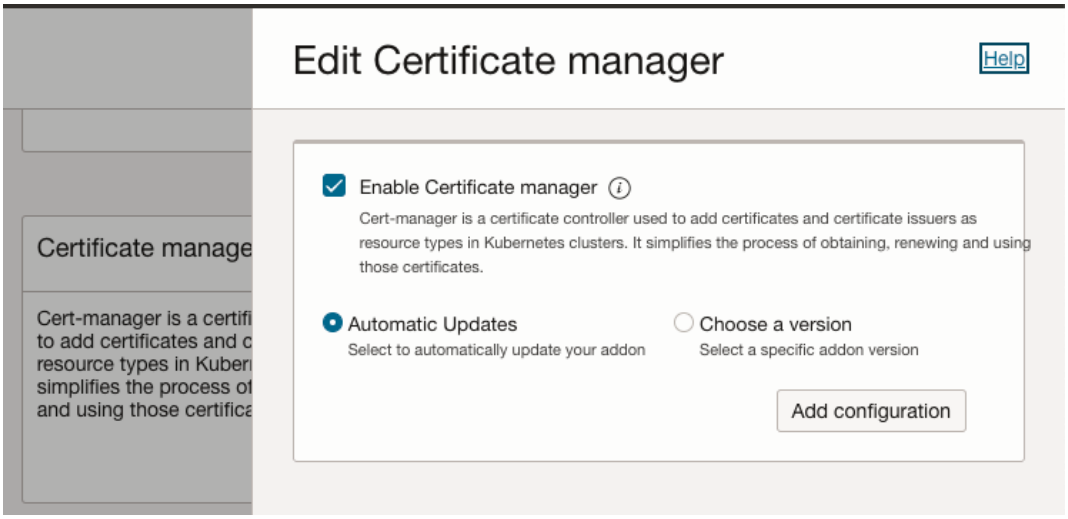
LAB 6 - Install CertManager

Task 1: Add Cert-Manager to OKE via Add-on:

1. Navigate to **Developer Services** > **OKE** > Click on **Cluster Name** > Click **Add-ons >** Click **Manage add-ons**



Task 2: Enable CertManager



Task 3: Validate CertManager installation

1. Verify that CertManager pods are running

```
kubectl get pods -n cert-manager
```

Sample response

```
NAME                                       READY   STATUS
RESTARTS   AGE
cert-manager-xxxxxx                        1/1     Running   0
2m
cert-manager-cainjector-xxxxxx             1/1     Running   0
2m

cert-manager-webhook-xxxxxx                1/1     Running   0
2m
```

Task 4: Create Let's Encrypt ClusterIssuer

1. Create a YAML file named : **letsencrypt-prod-clusterissuer.yaml**

```
vi letsencrypt-prod-clusterissuer.yaml
```

2. Enter the following:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: your-email@example.com # Replace with your email
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx
```

Task 5: Apply the ClusterIssuer:

```
kubectl apply -f letsencrypt-prod-clusterissuer.yaml
```

Task 6: Validate ClusterIssuer created:

```
kubectl get clusterissuer
```

Sample response

```
NAME              READY   AGE
letsencrypt-prod    True    2m
```

Task 7: Configure the MuShop Ingress Resource with TLS

1. Create YAML file named: mushop-dev-ingress.yaml

```
vi mushop-dev-ingress.yaml
```

2. Add the following content:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mushop-dev
  namespace: mushop
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-prod
spec:
  tls:
  - secretName: tls-secret
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: edge
            port:
              number: 8080
```

Task 8: Apply the Ingress Resource:

1. Apply the Ingress resource to your mushop namespace:

   ```
   kubectl apply -f mushop-dev-ingress.yaml
   ```

Task 9: Validate Ingress Setup:

2. Execute the following command:

   ```
   kubectl get ingress mushop-dev -n mushop
   ```

   Expected resaults after LoadBalancer deployment:

   - Ingress resource and an external IP
   - hostname

Task 10: Test HTTPs Access:

1. Find the **External IP** assigned to the ingress controller.

   ```
   kubectl get svc -n nginx-ingress
   ```

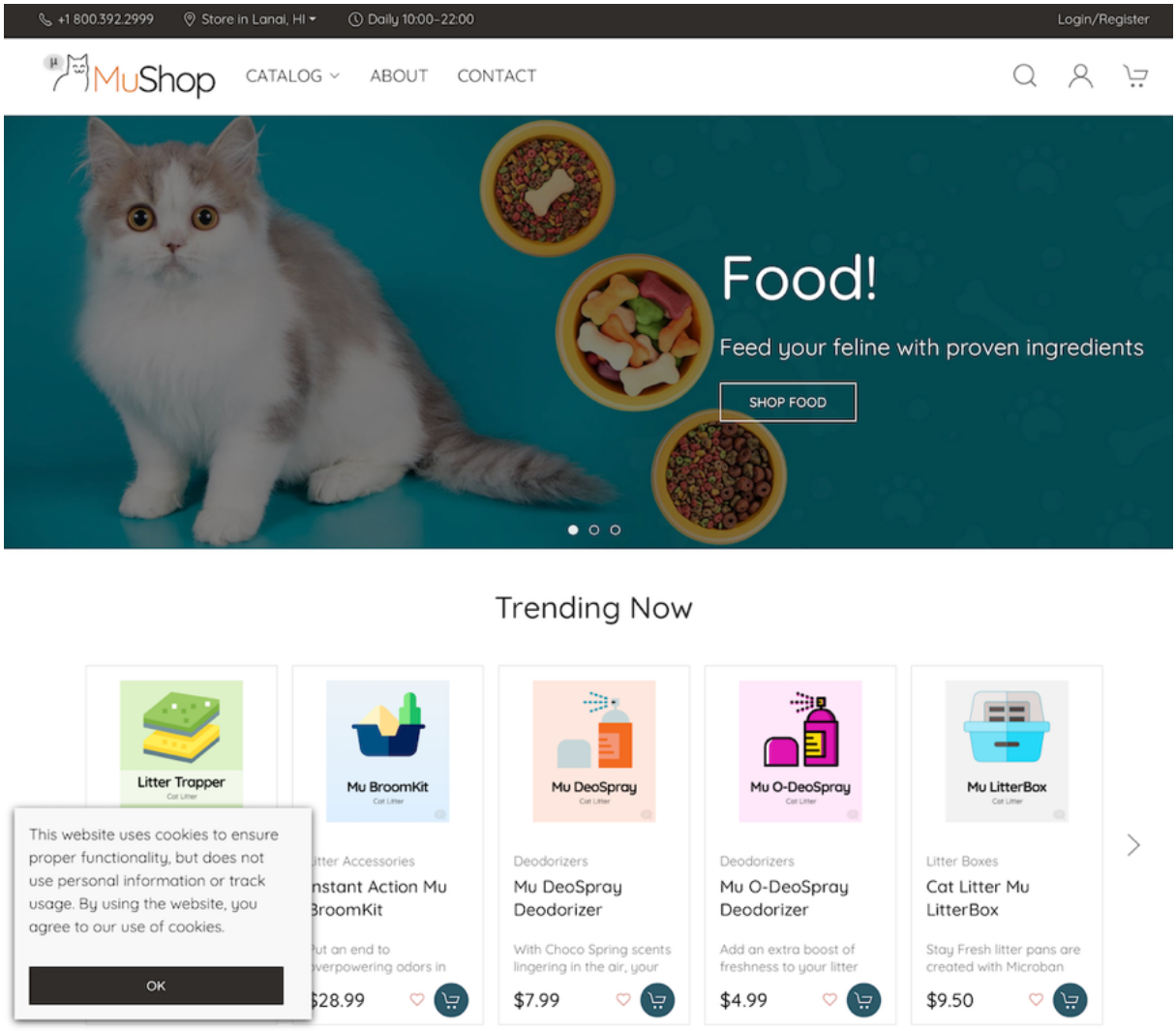
   Look for the EXTERNAL-IP in the output:

   Sample response

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

nginx-ingress-ingress-nginx-controller        LoadBalancer
10.96.25.104  your-external-ip   80:32456/TCP,443:31578/TCP
```

Task 11: Application Access:

1. Open to the MuShop Storefront by using your browser connecting to `https://< EXTERNAL-IP >`



LAB 7 – Deployment Autoscaling

Introduction

Scaling out a Deployment will ensure new Pods are created and scheduled to Nodes with available resources. Scaling will increase the number of Pods to the new desired state. Kubernetes also supports autoscaling of Pods, but it is outside of the scope of this tutorial. Scaling to zero is also possible, and it will terminate all Pods of the specified Deployment.

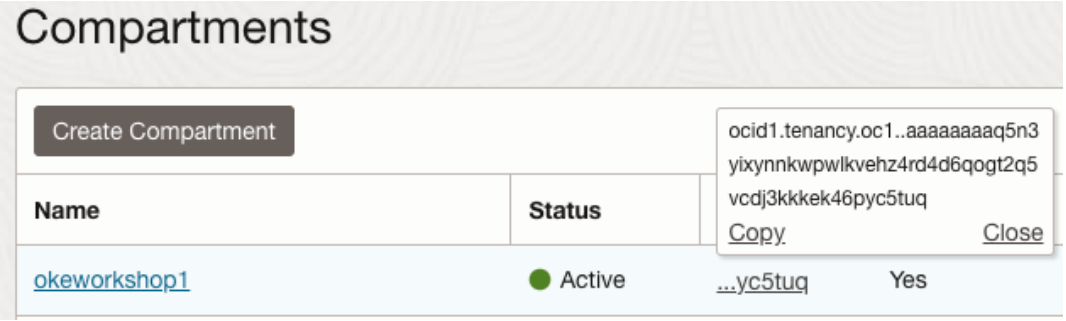
This step showcases the Horizontal Pod Autoscaling configurations deployed with the MuShop application.

Task 1: Create Policy to work with Autoscaler

1. Setting Up Instance Principal to Enable the Cluster Autoscaler Add-on to Access to Node Pools

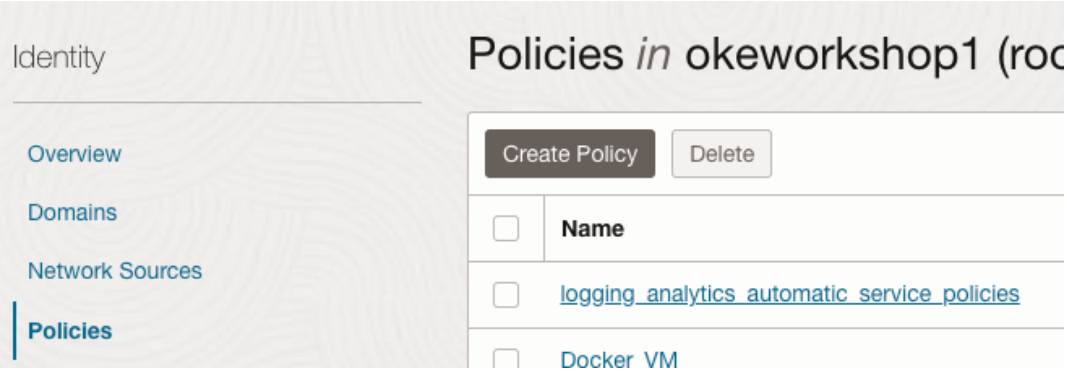
    - **Obtain the Cluster OCID (Cluster ID)**

    - **Obtain Compartment OCID -** Navigate to **Identity & Security** > **compartments**

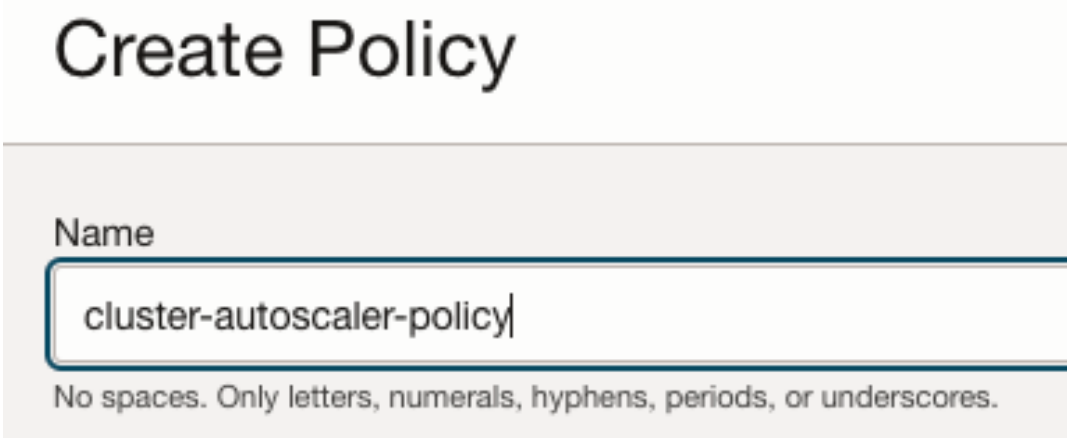


2. **Create the policy**

   - **Navigate** to **Identity & Security** > **Policies > Create Policy**



   - Fill the policy name (e.g cluster-autoscaler-policy)



- Enter policy statements to allow node pool management as follow:

Allow dynamic-group <dynamic-group-name> to manage cluster-node-pools in compartment id <compartment-ocid> where ALL {request.principal.type='workload', request.principal.namespace ='kube-system', request.principal.service_account = 'cluster-autoscaler', request.principal.cluster_id = '<cluster-ocid>'}
Allow dynamic-group <dynamic-group-name> to use subnets in compartment id <compartment-ocid> where ALL {request.principal.type='workload', request.principal.namespace ='kube-system', request.principal.service_account = 'cluster-autoscaler', request.principal.cluster_id = '<cluster-ocid>'}
Allow dynamic-group <dynamic-group-name> to read virtual-network-family in compartment id <compartment-ocid> where ALL {request.principal.type='workload', request.principal.namespace ='kube-system', request.principal.service_account = 'cluster-autoscaler', request.principal.cluster_id = '<cluster-ocid>'}
Allow dynamic-group <dynamic-group-name> to use vnics in compartment id <compartment-ocid> where ALL {request.principal.type='workload', request.principal.namespace ='kube-system', request.principal.service_account = 'cluster-autoscaler', request.principal.cluster_id = '<cluster-ocid>'}
Allow dynamic-group <dynamic-group-name> to inspect compartments in compartment id <compartment-ocid> where ALL {request.principal.type='workload', request.principal.namespace ='kube-system', request.principal.service_account = 'cluster-autoscaler', request.principal.cluster_id = '<cluster-ocid>'}
Allow dynamic-group <dynamic-group-name> to manage all-resources in tenancy

Replace:

- `<cluster-ocid>` with the cluster OCID obtained previously.
- `<compartment-ocid>` with the cluster OCID obtained previously.
- `<dynamic-group-name>` with the Dymanic group created on LAB 3 (Vault)

Task 2: Create the Cluster Autoscaler Add-on configuration

1. In OCI console navigate to **OKE cluster > Add-on > Manage Add-ons >** Click on **Cluster Autoscaler >** Click **"Enable Cluster Autoscaler >** Choose **nodes** from the option drop-down list **>** Fill the "Value" parameters

   The nodes parameter value has the following format:

2. For this lab Replace :

   - `<min-nodes> = 1` , The minimum number of nodes allowed in the node pool. The Kubernetes Cluster Autoscaler will not reduce the number of nodes below this number.
   - `<max-nodes> = 5` , The maximum number of nodes allowed in the node pool. The Kubernetes Cluster Autoscaler will not increase the number of nodes above this number. Make sure the maximum number of nodes you specify does not exceed the tenancy limits for the worker node shape defined for the node pool.
   - `<nodepool-ocid>` with the node pool OCIDs.

Task 3: Validate Cluster Autoscaler succesfully deployed

```
kubectl get deployments -n kube-system | grep cluster-autoscaler
```

```
shmuel_mur@cloudshell:deploy (eu-frankfurt-1)$ kubectl get deployments -n kube-system | grep cluster-autoscaler
cluster-autoscaler    1/1     1            1           2m39s
```

Task 4: Change the Value of the HPA autoscaaller

1. Execute the command:

   ```
   kubectl get hpa -n mushop
   ```

2. Edit the value of the HPA with the command **:**

   ```
   kubectl edit hpa mymushop-api -n mushop
   ```

```
ubuntu@instance-oke-workshop:~/valult$ kubectl get hpa -n mushop
NAME                 REFERENCE                       TARGETS              MINPODS   MAXPODS   REPLICAS
mymushop-api         Deployment/mymushop-api         cpu: <unknown>/70%   1         50        50
mymushop-assets      Deployment/mymushop-assets      cpu: <unknown>/70%   1         10        1
mymushop-catalogue   Deployment/mymushop-catalogue   cpu: <unknown>/70%   1         10        1
mymushop-edge        Deployment/mymushop-edge        cpu: <unknown>/70%   1         10        1
mymushop-events      Deployment/mymushop-events      cpu: <unknown>/70%   1         10        1
mymushop-storefront  Deployment/mymushop-storefront  cpu: <unknown>/70%   1         10        1
mymushop-user        Deployment/mymushop-user        cpu: <unknown>/70%   1         10        1
ubuntu@instance-oke-workshop:~/valult$ kubectl edit hpa mymushop-api -n mushop
Edit cancelled, no changes made.
ubuntu@instance-oke-workshop:~/valult$
```

3. Change the value to **50**

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  annotations:
    meta.helm.sh/release-name: mymushop
    meta.helm.sh/release-namespace: mushop
  creationTimestamp: "2024-11-17T13:22:46Z"
  labels:
    app.kubernetes.io/managed-by: Helm
  name: mymushop-api
  namespace: mushop
  resourceVersion: "19587"
  uid: 0e275e5c-b6cb-4a2c-8bae-b370afae3a32
spec:
  maxReplicas: 50
  metrics:
  - resource:
      name: cpu
      target:
        averageUtilization: 70
        type: Utilization
    type: Resource
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mymushop-api
status:
  conditions:
  - lastTransitionTime: "2024-11-17T13:23:01Z"
    message: the HPA controller was able to get the target's current scale
    reason: SucceededGetScale
```

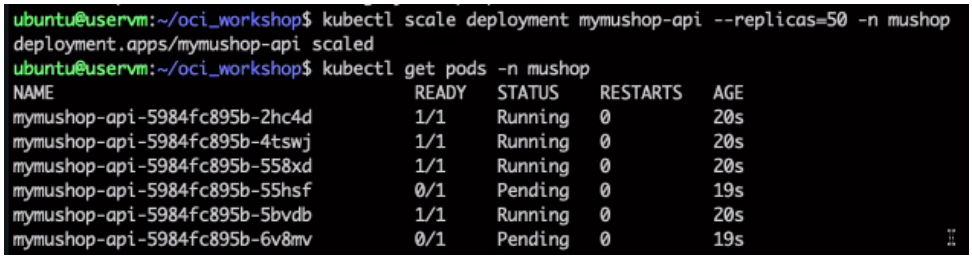
4. Save the changed with command **:wq**

Task 5: Increase replica count in the Mushop API service deployment to 50

1. Execute the following command:

   ```
   kubectl scale deployment mymushop-api --replicas=50 -n mushop
   ```

   Example – Status should be "Running" for all



2. Where Pod Status is "**Pending**" you might verify the reseaon by executing the following command :

   ```
   kubectl describe pod <add pod name>  -n mushop
   ```

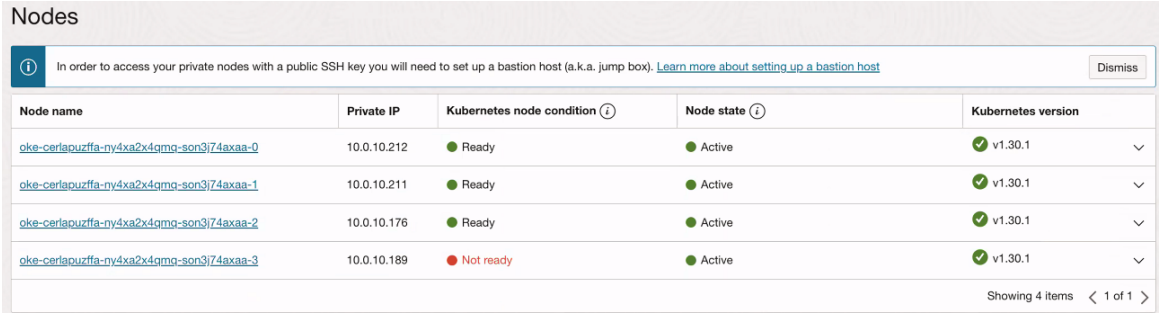
   Example – Status should be "Running" for all

Task 6: Validate replica size and new node were created in the nodepool

1. In the CLI Execute the following command:

   ```
   Kubectl get pods -n mushop
   ```

2. In OCI console navigate to **Developer services** > **Kubernetes Clusters (OKE)** > click **Cluster Name** > click **Node pools** > click **Node pool name** > validate the number of Nodes > 3



Task 7: Revert replica size

1. Execute the following command and validate that pods are terminated

   ```
   kubectl scale deployment mymushop-api --replicas=1 -n mushop
   ```

2. Validate the number of nodes in the OCI console as described in task 6.

Task 8: validate # pod revert to the original count

```
kubectl get pods -n mushop
```

LAB 8 – Monitoring the deployment

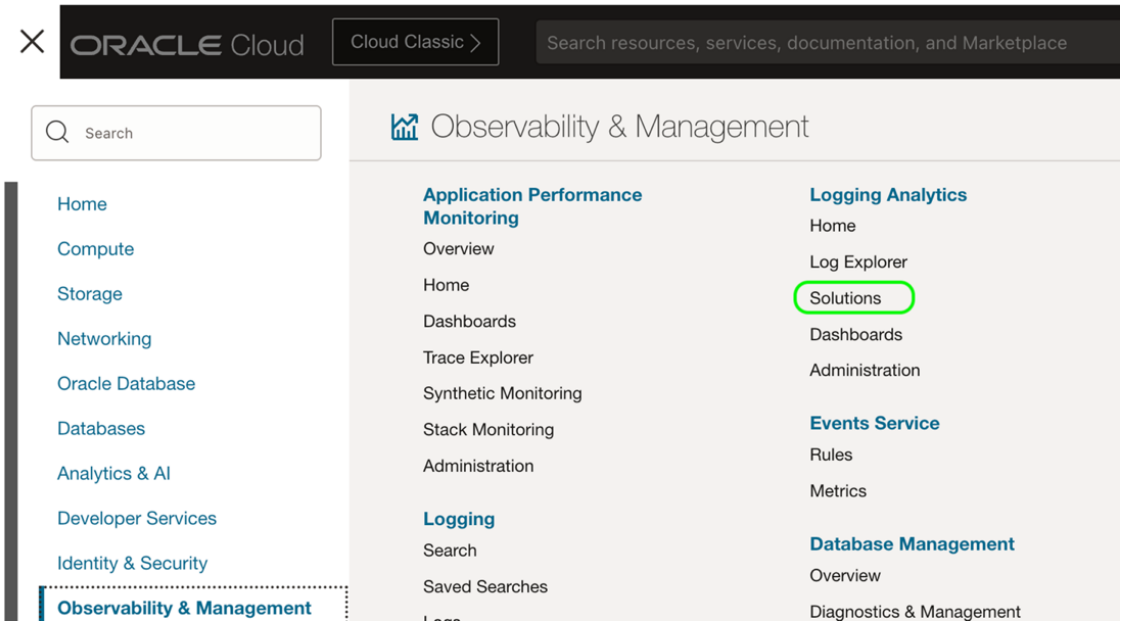
Introduction

Observability helps to understand and explain the state of your systems using a combination of logs, metrics and traces. It helps bringing better visibility into systems.

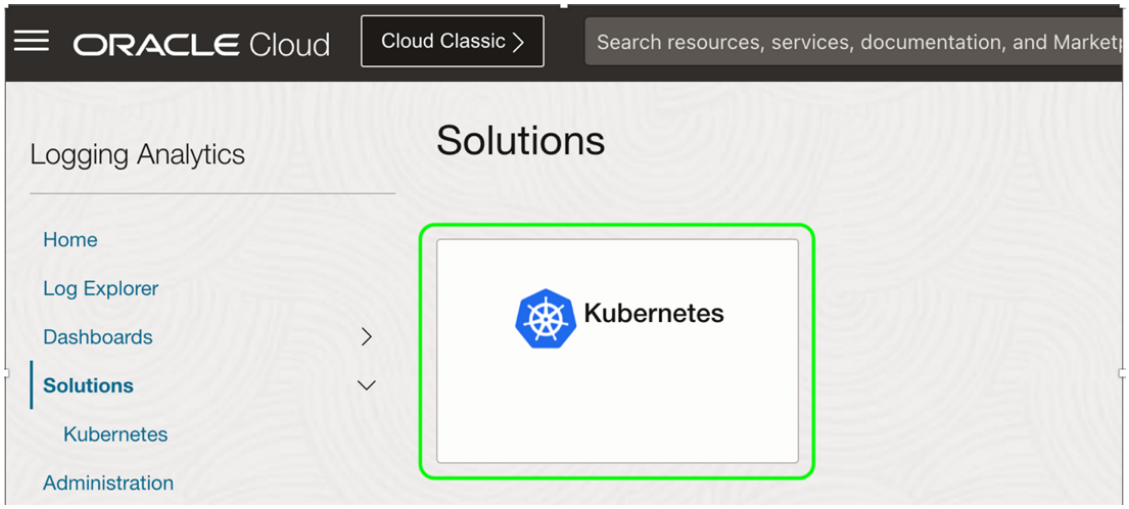
Pre-requisites: OCI CLI & Kubectl installed

Task 1: OKE Management Pack Deployment

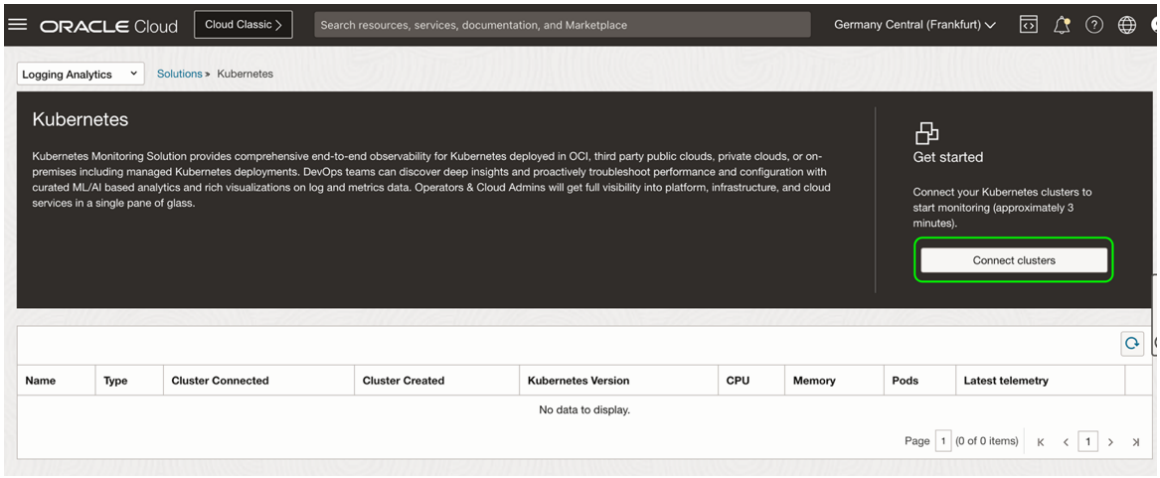
1. Navigate to **Observability & Management** > **Logging Analytics** > Click on **Solutions**



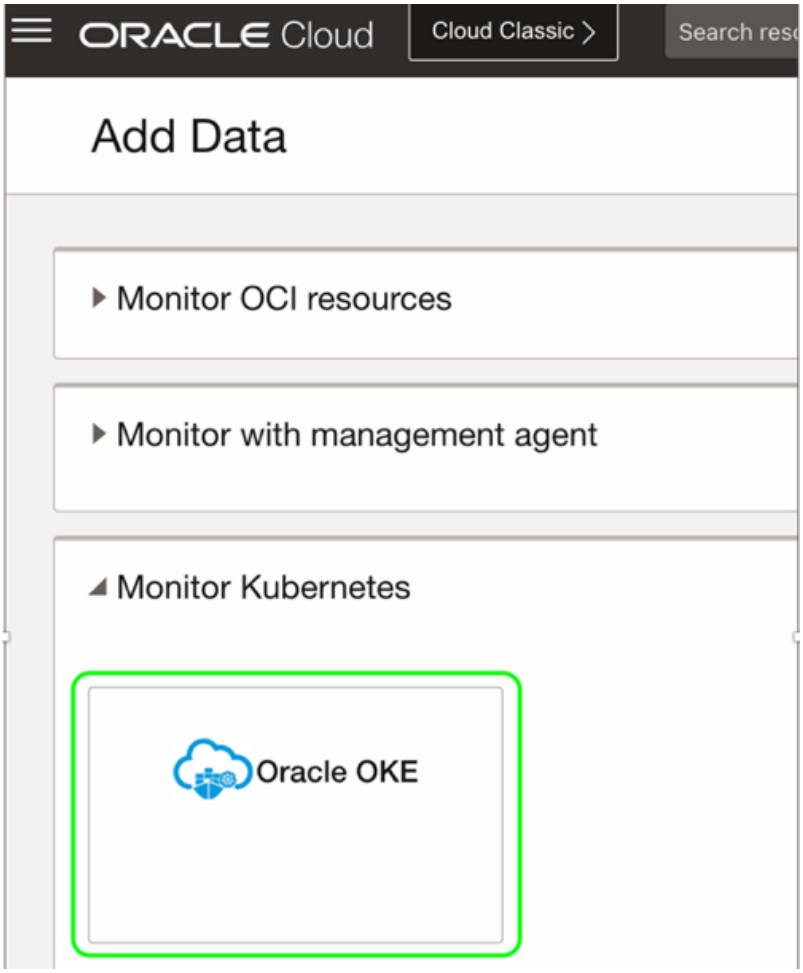
2. Choose **Kubernetes** solution



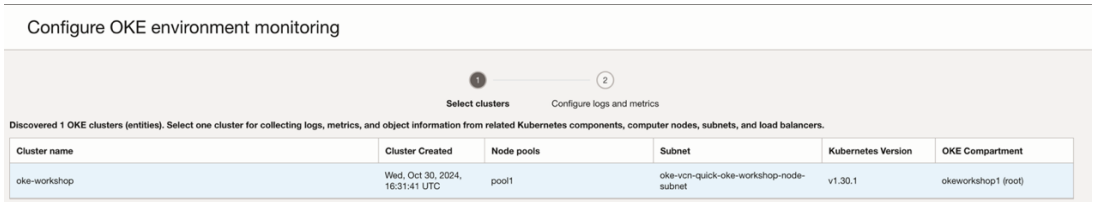
3. Click on **Connect clusters**



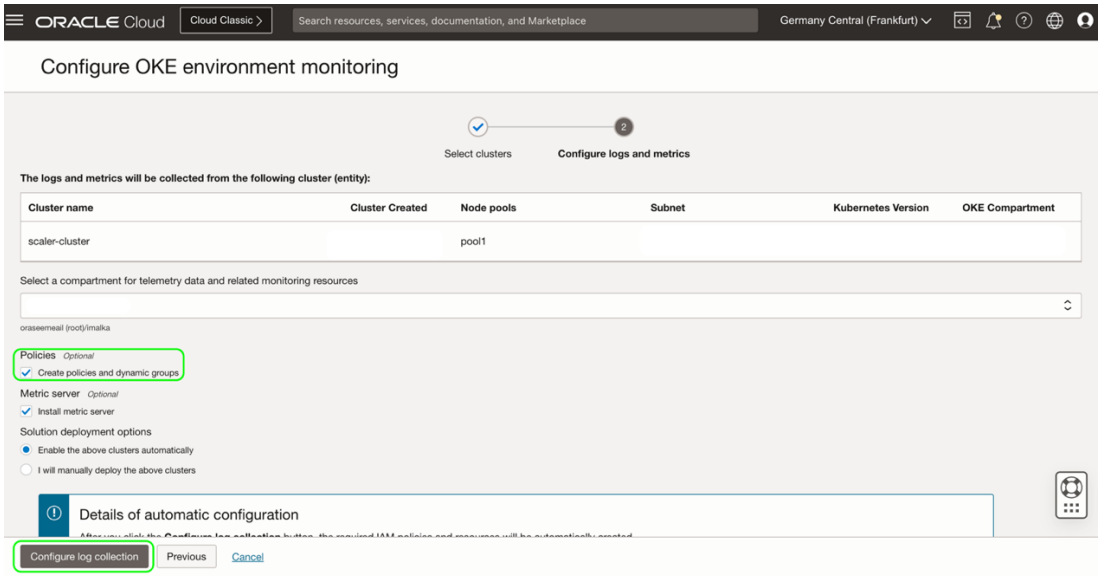
4. **Add Data** > **Monitor Kubernetes** > **Oracle OKE**



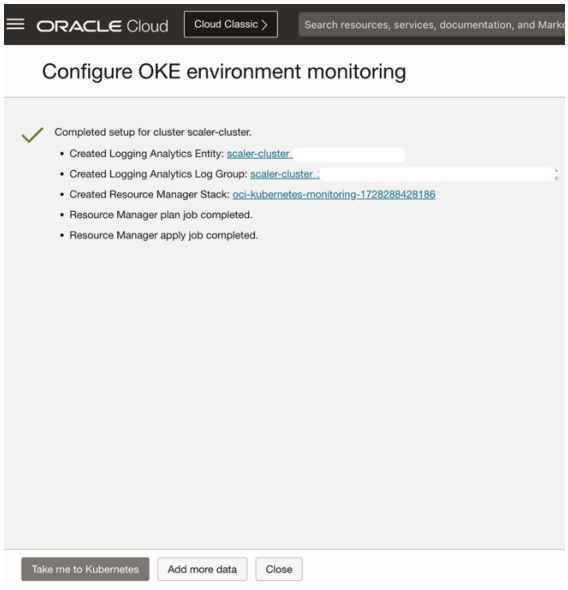
5. Select the **OKE cluster** from the clusters list and click **Next**



6. Validate all the details based on the following screen



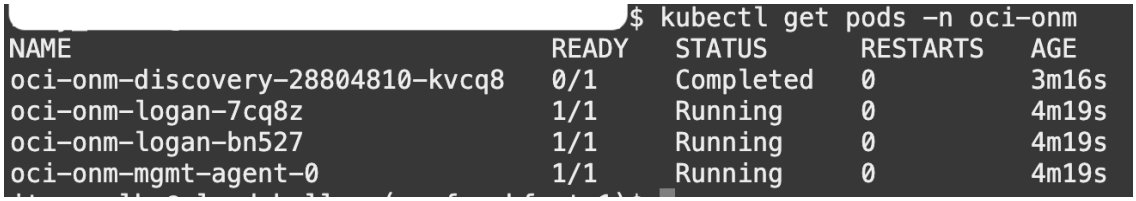
7. The following page after assure that OKE monitoring environment configured succefuly



8. Validate OKE components deployed successfully via CLI

```
kubectl get pods -n oci-onm
```

Sample response



Task 2: Create (import) OKE Dashboards

1. Go into the cloned GitHub repository '**oci-cloudnative-ext**' or clone it again as follow :

   `git clone` https://github.com/oci-oke-workshop-il/oci-cloudnative-ext.git

2. Execute the following commad

   ```
   cd oci- cloudnative-ext/dashboards_json
   ```

3. Obtain the **OCID** of the compartment, where the dashboards need to be imported (same as the OKE cluster's compartment).

   `${compartment_ocid}` - Replace all the instances of the keyword in the JSON with Compartment OCID

4. The Following command is for quick reference that can be used in a linux/cloud-shell **environment:**

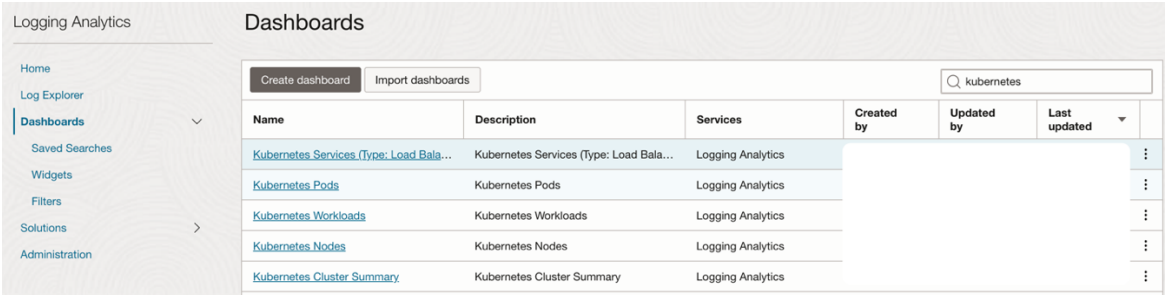
   ```
   sed -i "s/\${compartment_ocid}/<Replace-with-Compartment-OCID>/g" *.json
   ```

5. Execute the following commands to import the dashboards.

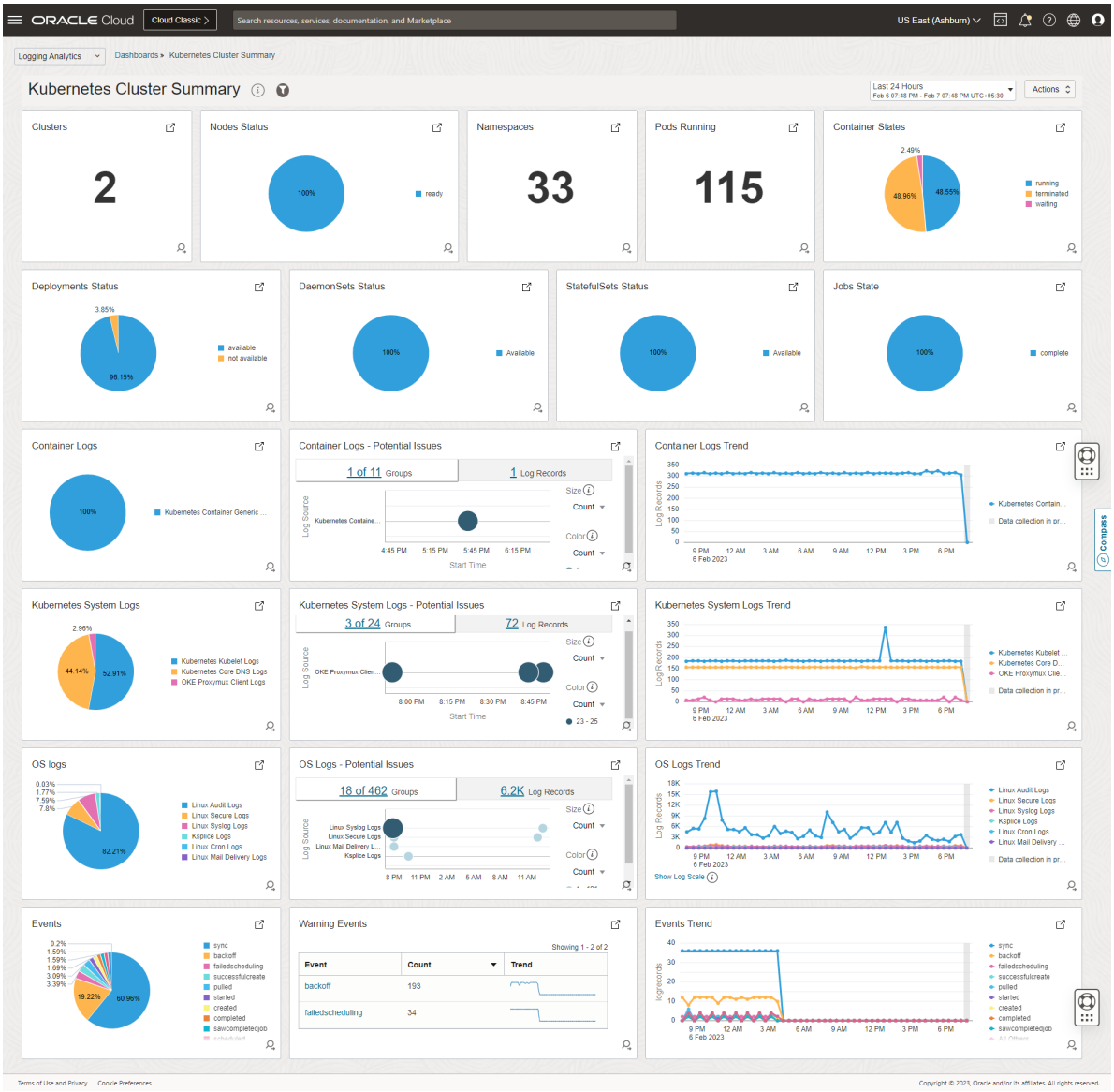
   ```
   oci management-dashboard dashboard import --from-json file://cluster.json
   oci management-dashboard dashboard import --from-json file://node.json
   oci management-dashboard dashboard import --from-json file://workload.json
   oci management-dashboard dashboard import --from-json file://pod.json
   oci management-dashboard dashboard import --from-json file://service-type-
   lb.json
   ```

Task 3: Review the OKE Dashboard

1. Navigate to **Observability & Management** > **Logging Analytics** > Click on **Dashboards**

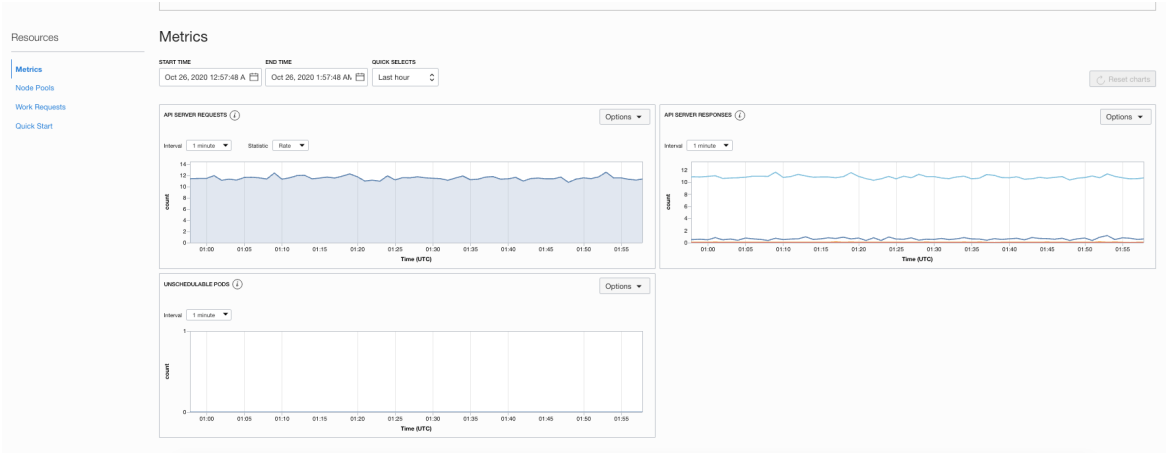


2. Click on the dashboards to view the cluster metrics, logs and status:



Task 3: Review OKE Metrics

1. OKE Cluster Metrics: **Navigate to Developer Services** > **Kubernetes Clusters** >
2. Under **Resources** -> **Metrics** observe the following metrics:
    - Unschedulable pods, which can be used to trigger node pool scale operations when there are insufficient resources on which to schedule pods.
    - API Server requests per second, which is helpful to understand any underlying performance issues seen in the Kubernetes API server.
3. These metrics can also be viewed from OCI Monitoring console under “oci_oke” namespace. Additionally, alarms can be created using industry standard statistics, trigger operators, and time intervals.



4. OKE Node Pool Metrics: Navigate to **Developer Services** > **Kubernetes Clusters** > **Node Pools** >

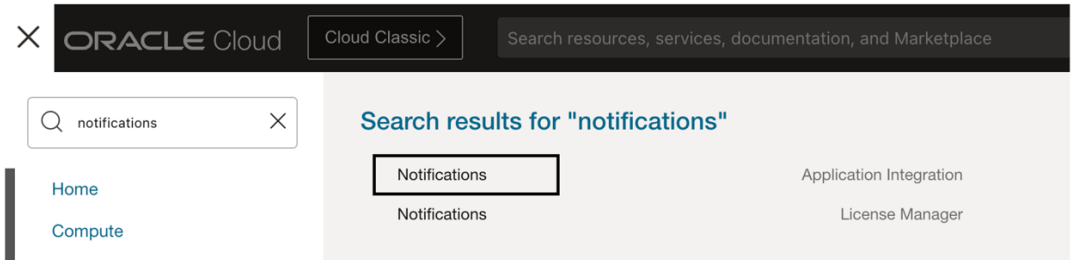
    Observe the following node pool metrics:
    - Node State (If your worker nodes are in Active state as indicated by OCI Compute Service)
    - Node condition (If your worker node are in Ready state as indicated by OKE API server)

5. OKE Worker Node Metrics: Navigate to **Developer Services** > **Kubernetes Clusters** > **Node Pools** > **Nodes** >
   Observe the following node metrics:
   - Activity level from CPU. Expressed as a percentage of total time (busy and idle) versus idle time. A typical alarm threshold is 90 percent.
   - Space currently in use. Measured by pages. Expressed as a percentage of used pages versus unused pages. A typical alarm threshold is 85 percent.
   - Activity level from I/O reads and writes. Expressed as reads/writes per second.
   - Read/Write throughput. Expressed as bytes read/Write per second.
   - Network receipt/transmit throughput. Expressed as bytes received/transmit per second.

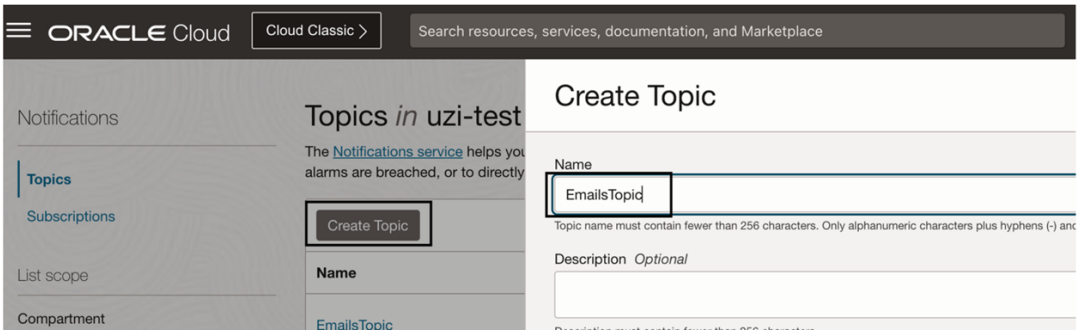
LAB – 9 - Create Alarms to Detect unscheduled Pods

Task 1: Create an OCI Topic & Subscription for email notifications

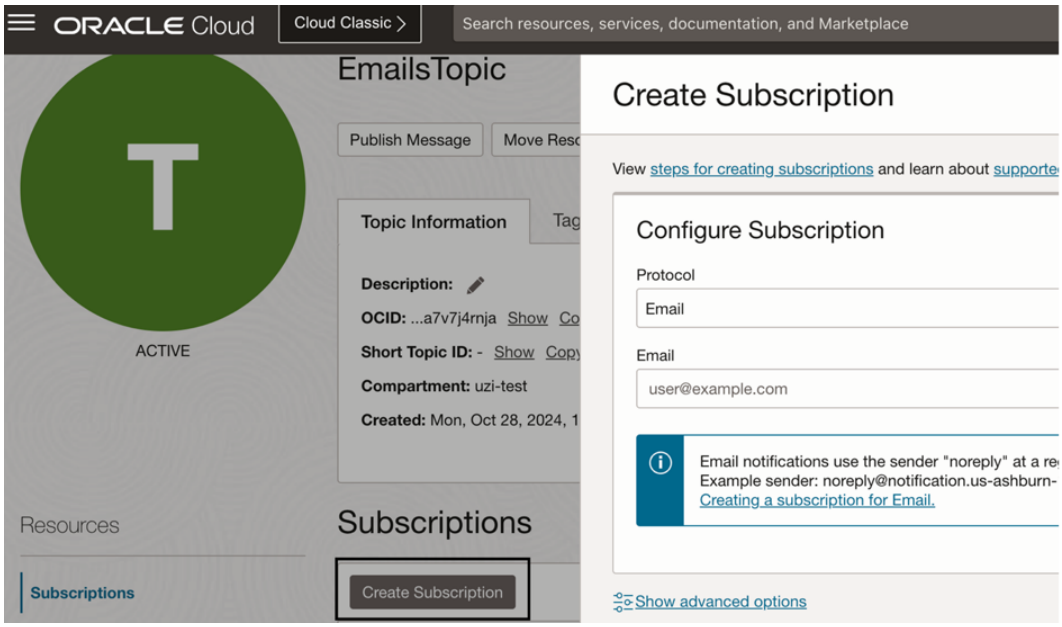
1. Search for **notifications** > click of Notifications under **Application Integration**



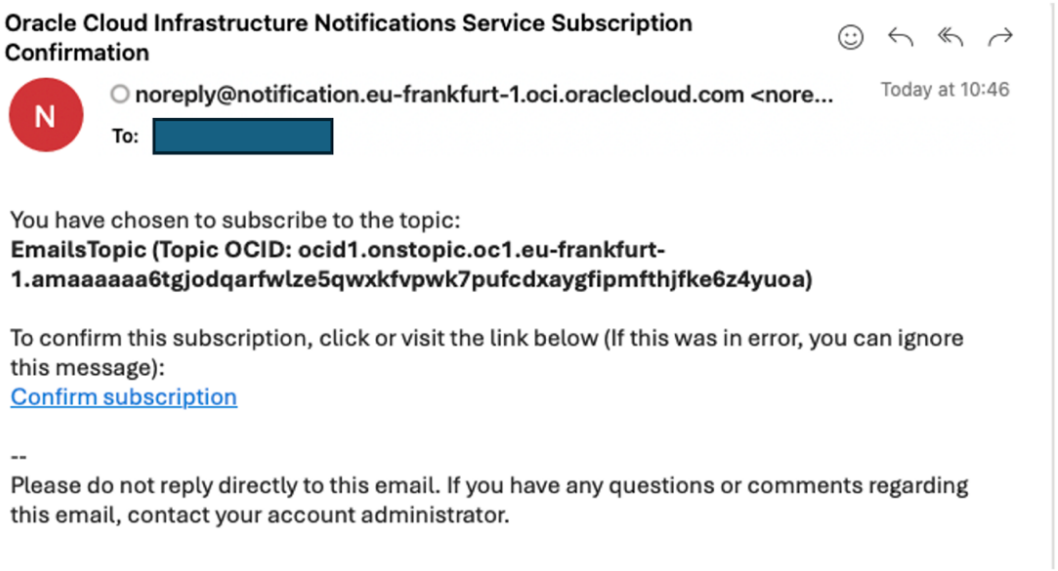
2. Click **Create topic** and name it – **EmailsTopic**



3. Click on the created topic > Select **Create Subscription** > Protocol `Email` > Email `{Your-Email-Address}` > Click **Create**



4. Go to your **email inbox** > **Confirm** the subscription.

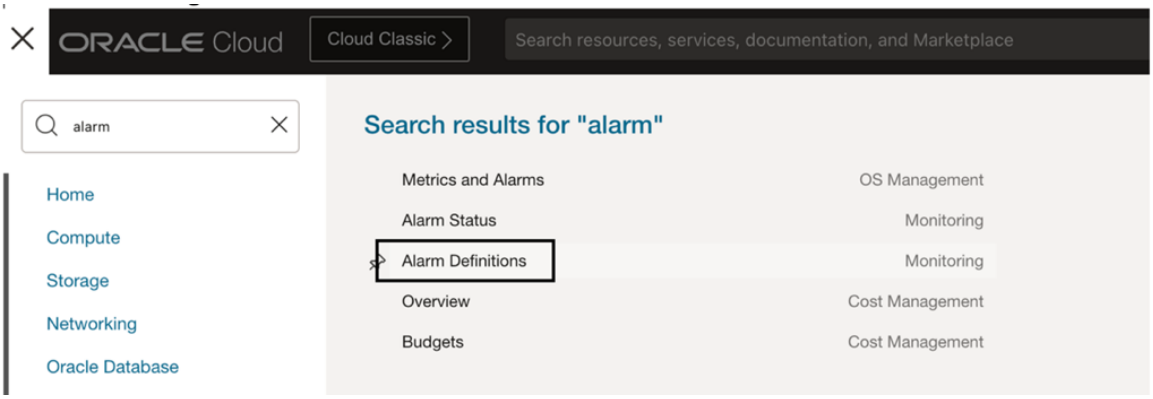


5. Validate Created Subscription in **Active** state

Task 2: Configure Alarm rule

The following alarm will generate a notification when the number of Kubernetes Nodes will be less than 3 Nodes which indicates a non-stable environment!

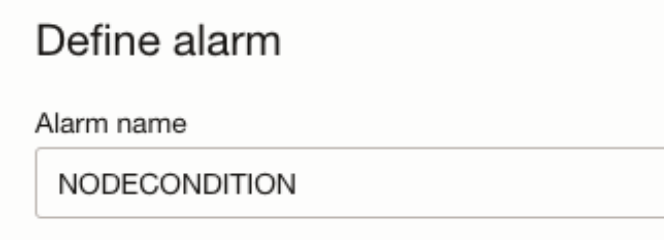
1. Search for **alarm** > click of **Alarm Definitions** under **Monitoring**



2. Click **Create Alarm**

3. **Enter the following information:**

   - Alarm Name: `NODECONDITION`

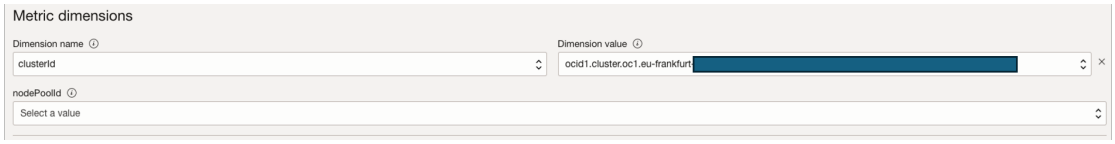


   - **Metric Description:**

     - Compartment: `<Your-Compartment>`
     - Metric Namespace: `oci_oke`
     - Metric name: `KubernetesNodeCondition`

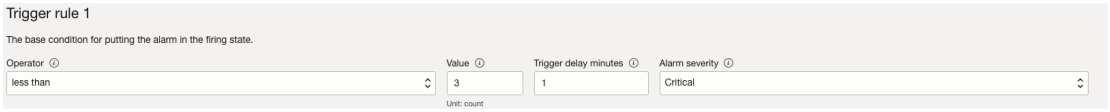
- **Metric dimensions:**

  - Dimension name: `clustereId`
  - Dimension value: `<OKE-Cluster-OCID>`
  - NodePoolId: `leave blank`

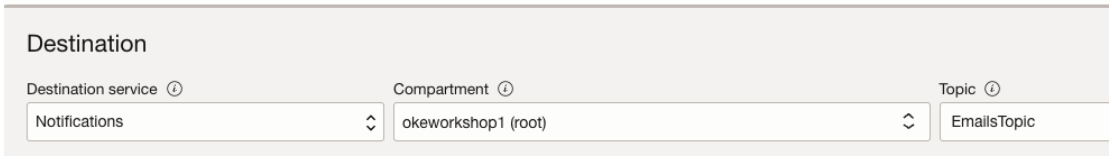


- **Trigger rule 1 Values:**

  - Operator: `Less than`
  - Value: `3`
  - Trigger delay minutes: **1**`(default)`
  - Alarm severity: `Critical`



- **Destination:**
  - Destination Service: `Notifications`
  - Compartment: **<**`Your-Compartment>`
  - Topic: `EmailsTopic`

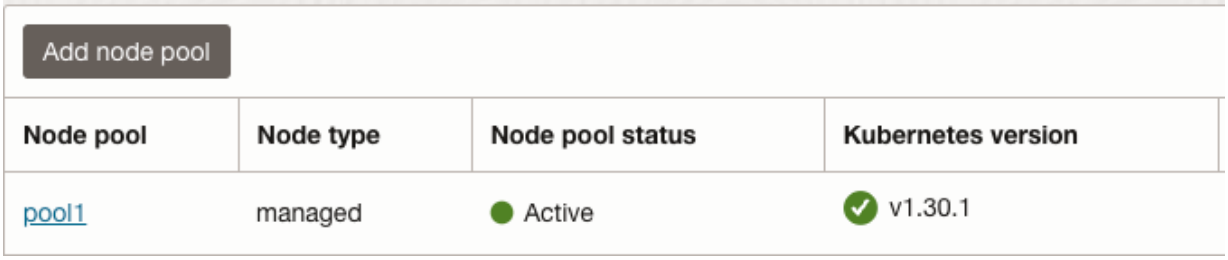


- Click **Save alarm**

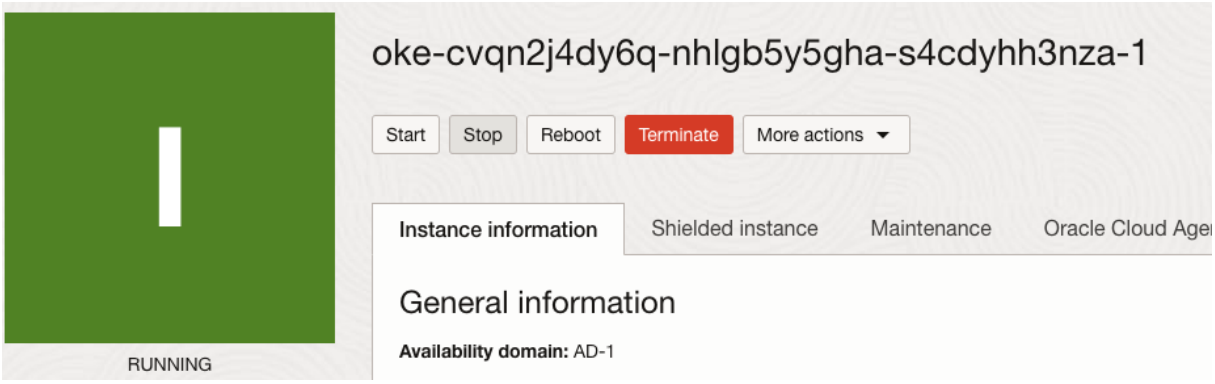
Task 3: Trigger Alarm manually by stopping one of the nodes in the pool

1. Navigate to **Developer Services > Kubernetes Clusters (OKE) >** Click **Node Pools**

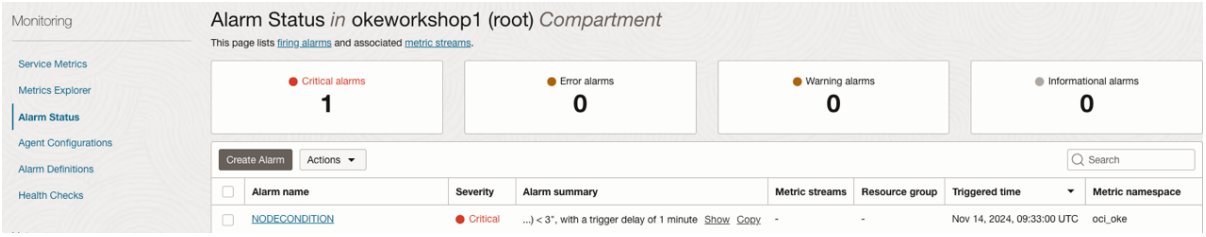
2. Click **Pool Name**



3. Click on **one of the nodes** > Click **Stop**

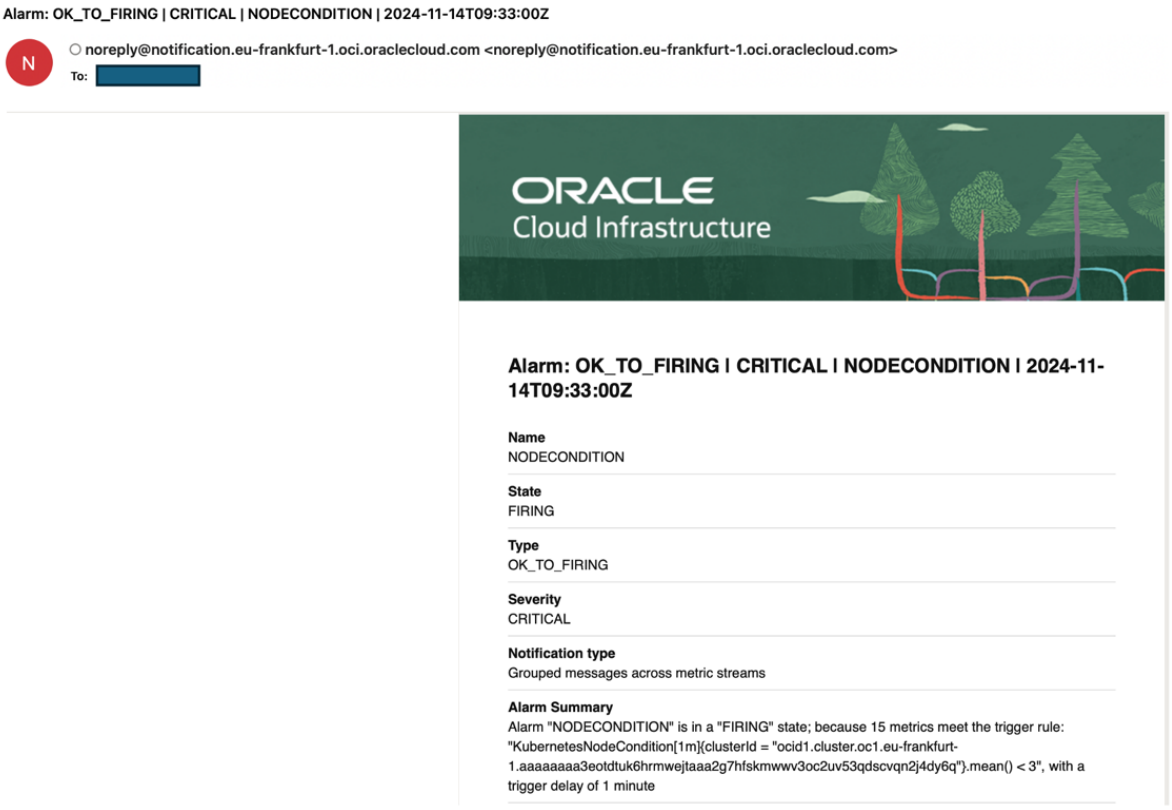


4. Go back to **Alarm** view to validate the system identify the false Node and trigger and Alarm



5. Validate Email notification service works correctly

Go to the **Email** and validate the following **email received:**



LAB 10 - Use Taints and Tolerations for Resource Allocation (Optional)

Open the following file:

`deploy/complete/helm-chart/mushop/charts/catalogue/values.yaml`

Add the toleration parameters:

tolerations:
key: "dedicated"
value: "test-taint"
effect: "NoSchedule"

Open the following file:

`deploy/complete/helm-chart/mushop/charts/catalogue/templates/catalogue-deployment.yaml`

Add the toleration parameters:
tolerations:

```
    {{- toYaml .Values.tolerations | nindent 8 }}
```

Redeploy the service and validate the following expected resaults: