Configure OCI alerting system using function

High Level Steps

1. Create a docker repository on OCI (menu -> developer services)
2. Configure docker on local machine
3. Use docker login to log into OCI docker registry
4. Configure function on local machine
5. Create python environment and create function
6. Create docker image from function
7. Push docker image to the OCI registry
8. Create application and function end point from push docker image
9. Configure alarm pointing to function end point

What is Function => Fn is a lightweight Docker-based serverless functions platform you can run on your laptop, server, or cloud. It’s mainly a docker image with invoking end point configured. Due to any specific reason when this endpoint is invoked then this docker image kicks in, does it’s job and send response if required. more info -> <https://fnproject.io/>

configuring OCI docker registry :

1. create an auth token for use with Oracle Cloud Infrastructure Registry
2. create a new repository
3. log in to Oracle Cloud Infrastructure Registry from the Docker CLI. For e.g. docker login -u '<compartment>/<user-name>' iad.ocir.io
4. pull a test image from DockerHub
5. tag the image
6. push the image to Oracle Cloud Infrastructure Registry using the Docker CLI
7. verify the image has been pushed to Oracle Cloud Infrastructure Registry using the Console

For further info -- <https://www.oracle.com/webfolder/technetwork/tutorials/obe/oci/registry/index.html>

Functions QuickStart on Local Host:

1. Set up your tenancy

1. Create groups and users

2. Create compartment

3. Create VCN and subnets

4. Create policy for group and service

Policies required to run function to the group as below

1. Allow group <group-name> to use cloud-shell in tenancy
2. Allow group <group-name> to manage repos in tenancy
3. Allow group <group-name> to read objectstorage-namespaces in tenancy
4. Allow group <group-name> to manage logging-family in tenancy
5. Allow group <group-name> to read metrics in tenancy
6. Allow group <group-name> to manage functions-family in tenancy
7. Allow group <group-name> to use virtual-network-family in tenancy
8. Allow group <group-name> to use apm-domains in tenancy
9. Allow group <group-name> to read vaults in tenancy
10. Allow group <group-name> to use keys in tenancy
11. Allow service faas to use apm-domains in tenancy
12. Allow service faas to read repos in tenancy where request.operation='ListContainerImageSignatures'
13. Allow service faas to {KEY\_READ} in tenancy where request.operation='GetKeyVersion'
14. Allow service faas to {KEY\_VERIFY} in tenancy where request.operation='Verify'

More info for Function policies -- <https://docs.oracle.com/en-us/iaas/Content/Functions/Tasks/functionscreatingpolicies.htm>

1. Create application

Create your first application

1. Sign in to the Console as a functions developer.
2. In the Console, open the navigation menu and click Developer Services. Under Functions, click Applications.
3. Select the region you are using with Oracle Functions.
4. Click Create Application.

Graphical user interface, application

Description automatically generated

1. This image shows the New Applicatoin dialog, with empty Name, VCN, and Subnets fields. Specify:

helloworld-app as the name for the new application. You'll deploy your first function in this application, and specify this application when invoking the function.

The VCN and subnet in which to run the function.

1. Click Create.
2. Set up your local host dev/prod environment
3. Install and start docker
   1. Check => docker version
   2. Check if docker is working => docker run hello-world
4. Set up API signing key and OCI profile
   1. Sign in to the Console as a functions developer
   2. Open the Profile menu (User menu icon) and click User Settings
   3. Under Resources, click API Keys, and then click Add API Key.
   4. Select Generate API Key Pair in the Add API Key dialog.

Graphical user interface, text, application, email

Description automatically generated

* 1. Click **Download Private Key** and save the private key file (as a .pem file) in the ~/.oci directory. (If the ~/.oci directory doesn't already exist, create it now).
  2. Click Add to add the new API signing key to your user settings.

The Configuration File Preview dialog is displayed, containing a configuration file snippet with basic authentication information for a profile named DEFAULT (including the fingerprint of the API signing key you just created).

Graphical user interface, text, application

Description automatically generated

* 1. Copy the configuration file snippet shown in the text box, and close the Configuration File Preview dialog.
  2. In a text editor, open the ~/.oci/config file and paste the snippet into the file. (If the ~/.oci/config file doesn't already exist, create it now).

1. In the text editor, change the profile in the snippet you've just pasted, as follows:

Change the name of the profile from [DEFAULT] to a name of your choosing (for example, [functions-developer-profile]). Note that the ~/.oci/config file cannot contain two profiles with the same name.

Change the value of the key\_file parameter of the profile to specify the path of the private key file (the .pem file) you downloaded earlier.

1. In the text editor, save the changes you've made to the ~/.oci/config file, and close the text editor.
2. In a terminal window, change permissions on the private key file (the .pem file) to ensure that only you can read it, by entering:
   1. chmod go-rwx ~/.oci/<private-key-file-name>.pem
3. Install Fn Project cli
   1. curl -LSs https://raw.githubusercontent.com/fnproject/cli/master/install | sh
   2. brew update && brew install fn (this is for Mac User then skip step a)
   3. fn version (confirming fn is working)
   4. fn start --log-level DEBUG (To start the fn server)

Step 4 & 5, not mandatory if you are running local fn server

1. Set up Fn Project CLI context provider –oracle
   1. Create a new Fn Project CLI context by entering:

fn create context <my-context> --provider oracle

* 1. fn use context <my-context>
  2. fn update context oracle.profile <profile-name> (more info -- <https://docs.oracle.com/en-us/iaas/Content/Functions/Tasks/functionscreatefncontext.htm#Create_an_Fn_Project_CLI_Context_to_Connect_to_Oracle_Cloud_Infrastructure>)

1. Complete Fn Project CLI context configuration
   1. fn update context oracle.compartment-id <compartment-ocid>
   2. fn update context api-url <api-endpoint>
   3. fn update context api-url <https://functions.us-phoenix-1.oci.oraclecloud.com>

More Info -- <https://docs.oracle.com/en-us/iaas/Content/Functions/Tasks/functionscreatefncontext.htm#Create_an_Fn_Project_CLI_Context_to_Connect_to_Oracle_Cloud_Infrastructure>

1. Generate Auth Token
   1. Sign in to the Console as a functions developer.
   2. Open the User menu and go to User Settings. On the Auth Tokens page, click Generate Token.
   3. Enter a meaningful description for the auth token in the Generate Token dialog, and click Generate Token. The new auth token is displayed.
   4. Copy the auth token immediately to a secure location from where you can retrieve it later, because you won't see the auth token again in the Console.
   5. Close the Generate Token dialog.

More info -- <https://docs.oracle.com/en-us/iaas/Content/Functions/Tasks/functionsgenerateauthtokens.htm#Generate_an_Auth_Token_to_Enable_Login_to_Oracle_Cloud_Infrastructure_Registry>

1. Log into Registry
   1. docker login -u '<tenancy-namespace>/<user-name>' <region-key>.ocir.io
   2. When prompted for a password, enter the Oracle Cloud Infrastructure auth token that you created and copied earlier. For example, 6aN...6MqX

d. Create, deploy, and invoke your function

Creating Function

1. fn init --runtime python pythonfn

2. A directory called pythonfn is created, containing:

A function definition file called func.yaml

A func.py will be created, which will contain handle function from where all the execution starts, we need to edit this func.py file for our customization

A requirements.txt file will be created which contain the list of python modules with versions which are required to run func.py

Deploying the function (this actually create a docker image with version)

1. fn --verbose deploy --app pythonapp –local

Tagging function/docker, or preparation for pushing the image to OCI docker registry

1. docker tag fndemouser/pythonfn:0.0.1 iad.ocir.io/vbauto1/vbcs-func-repo/pythonfn:0.0.1

Pushing the docker image to repository

1. docker push iad.ocir.io/vbauto1/vbcs-func-repo/pythonfn:0.0.1

Functions Development

Structure of the python function as below :

import io

import json

from fdk import response

def handler(ctx, data: io.BytesIO=None):

name = "World"

try:

body = json.loads(data.getvalue())

name = body.get("name")

except (Exception, ValueError) as ex:

print(str(ex))

return response.Response(

ctx, response\_data=json.dumps(

{"message": "Hello {0}".format(name)}),

headers={"Content-Type": "application/json"}

)

Main function called during execution starts from handler and argument passes through “data” on handler function and “data” need to be processed.

For sample functions refer -> <https://github.com/oracle/oracle-functions-samples>

Function which is implemented as below

"""[summary]

This is the body of function

"""

import io

import json

import logging

from fdk import response

import smtplib

from email.mime.text import MIMEText

import datetime

import re

import pytz

import oci

import tempfile

import sys

regex = re.compile('tomcat[\S]\*[vb|db|wtss]', re.IGNORECASE)

port = 587

sender = 'XYZ@smtp.email.us-ashburn-1.oci.oraclecloud.com'

receiver = [ 'email-id@domain-name.com', 'email-id-02@domain-name.com' ]

compartment\_id = 'ocid1.compartment.oc1..SOMEMISSINGINFO4q'

bucket\_name = 'OSS-bucket'

msg = MIMEText('Secured Mail From OCI\_Alert - Not Initialized')

logFile = tempfile.NamedTemporaryFile()

file\_handler = logging.FileHandler(filename=logFile.name)

stdout\_handler = logging.StreamHandler(sys.stdout)

handlers = [file\_handler, stdout\_handler]

logging.getLogger().info('I m Here 101')

user = 'ocid1.user.oc1..aaaaaaaaeml2dtthkwSOMEMISSINGINFO.w3.com'

password = 'THISISFALSEPASSWORD'

def time\_now\_str():

"""[summary]

This will return present time in DD-MM-YYYY:HH-MM-SSTZ format as string

"""

just\_now = datetime.datetime.utcnow().replace(tzinfo=pytz.utc)

just\_now\_str = "{:02d}".format(just\_now.day) + '-' + "{:02d}".format(just\_now.month) + \

'-' + str(just\_now.year) + ':' + str(just\_now.hour) + '-' + str(just\_now.minute) + \

'-' + str(just\_now.second) + str(just\_now.tzname())

return just\_now\_str

def upload\_log\_oss(bucketName, objectName, content):

signer = oci.auth.signers.get\_resource\_principals\_signer()

client = oci.object\_storage.ObjectStorageClient(config={}, signer=signer)

namespace = client.get\_namespace().data

output=""

try:

object = client.put\_object(namespace, bucketName, objectName, json.dumps(content))

output = "Success: Put object '" + objectName + "' in bucket '" + bucketName + "'"

except Exception as e:

output = "Failed: " + str(e.message)

send\_alert(output)

return { "state": output }

def send\_alert(message = ""):

"""[summary]

Sending Email of Alerts to Designated user

"""

msg = MIMEText(message)

msg['Subject'] = 'Alert System Function - alert-function'

with smtplib.SMTP("smtp.email.us-ashburn-1.oci.oraclecloud.com", port) as server:

server.starttls() # Secure the connection

server.login(user, password)

server.sendmail(sender, receiver, msg.as\_string())

logging.getLogger().info("Alert Mail Successfully Sent")

return True

def validate\_real\_alert(alerts):

"""[summary]

This function will validate if the alerts are real by check it length

& it must contain alarmMetaData

"""

if type(alerts) is dict and len(alerts) > 0:

if 'alarmMetaData' in alerts:

return True

return False

def seperating\_multiple\_alerts(alerts):

"""[summary]

Alerts may have multiple alerts combined,

This function will separate multple alerts to create individual

It will return array of alerts as b=objects fo JSON

"""

list\_of\_alerts = []

temp = {}

temp['dedupeKey'] = alerts['dedupeKey']

temp['title'] = alerts['title']

temp['type\_of\_alert'] = alerts['type']

temp['timestampEpochMillis'] = alerts['timestampEpochMillis']

temp['timestamp'] = alerts['timestamp']

for val in alerts['alarmMetaData']:

try:

temp['namespace'] = val['namespace']

except (IndexError, KeyError):

temp['namespace'] = 'None'

try:

temp['alert\_ocid'] = val['id']

except (IndexError, KeyError):

temp['alert\_ocid'] = 'None'

temp['status'] = val['status']

temp['severity'] = val['severity']

i = 0

while True:

try:

temp['resourceDisplayName'] = val['dimensions'][i]['resourceDisplayName']

except (IndexError, KeyError):

temp['resourceDisplayName'] = 'None'

try:

temp['resource\_ocid'] = val['dimensions'][i]['resourceId']

except (IndexError, KeyError):

temp['resource\_ocid'] = 'None'

list\_of\_alerts.append(temp.copy())

i = i + 1

print('i => ',i,' :::: len(val[dimensions]) => ',len(val['dimensions']) )

if i >= len(val['dimensions']):

break

logging.getLogger().info("list\_of\_alerts => {}",format(str(list\_of\_alerts)))

return list\_of\_alerts

def save\_delete\_alerts(alert):

"""[summary]

This function will be used to save live alerts

And delete cleared alerts, To a centralized oracle database

(Future)

"""

logging.getLogger().info('From save\_delete\_alerts\_Funtions Alerts => {}'.format(alert))

if str(alert['type\_of\_alert']).lower() == 'ok\_to\_firing': ## Save Alert

logging.getLogger().info('Saving First Alert') ## Saving to table -> livealert

if str(alert['type\_of\_alert']).lower() == 'firing\_to\_ok' or str(alert['type\_of\_alert']).lower() == 'reset': ## Delete Alert

if alert['resource\_ocid'] == 'None':

logging.getLogger().info('Deleting Multiple Alerts') ## deleting alert from table -> livealert

else:

logging.getLogger().info('Deleting One Alerts') ## deleting alert from table -> livealert

def filter\_tomcat\_alert(alerts):

"""[summary]

This function will take decision if generated alert is for tomcat or not

And Call function to save data to Oracle Database

And Send mail for tomcat alerts

"""

for data in seperating\_multiple\_alerts(alerts):

if regex.search(str(data['resourceDisplayName'])):

object\_name = data['resourceDisplayName'] + '\_' + time\_now\_str() + '.log'

if str(data['type\_of\_alert']).lower() == 'ok\_to\_firing':

logging.getLogger().info('Sending Mail As Alert ..... ')

# save\_delete\_alerts(data)

send\_alert(str(data))

upload\_log\_oss(bucket\_name, object\_name, data)

elif str(data['type\_of\_alert']).lower() == 'firing\_to\_ok' or str(data['type\_of\_alert']).lower() == 'reset':

# save\_delete\_alerts(data)

upload\_log\_oss(bucket\_name, object\_name, data)

logging.getLogger().info('Not Sending because of either type or name : {} => {}'.format(str(data['type\_of\_alert']), str(data['resourceDisplayName'])))

else:

logging.getLogger().info('Some Problem with filtering system')

else:

logging.getLogger().info('Alert Do not belong to tomcat :D')

def handler(ctx, data: io.BytesIO = None):

logging.getLogger().info('I m Here 105 {}'.format(handlers))

logging.basicConfig(handlers=handlers, format='%(asctime)s,%(msecs)d %(name)s \

%(levelname)s %(message)s',datefmt='%d/%m/%Y-%H:%M:%S', level=logging.INFO)

try:

body = json.loads(data.getvalue())

except (Exception, ValueError) as ex:

logging.getLogger().info('error parsing json payload: {}'.format(str(ex)))

logging.getLogger().info(body)

if validate\_real\_alert(body):

filter\_tomcat\_alert(body)

else:

logging.getLogger().info('Alerts are not valid .... Sorry .... :D')

logging.getLogger().info('Completing Execution .... :D')

return response.Response(

ctx, response\_data=json.dumps(

{"message": "{0}".format(body)}),

headers={"Content-Type": "application/json"}

)