**SERVERLESS IOT DATA PROCESSING**

**PROJECT FEATURES:**

When implementing this architecture, consider your requirements for the following parameters:

* **Request throttling**

After creating the API gateway and deploying one or more APIs, you might want to limit the rate at which front-end clients can send requests to backend services. Decide the request-rate limit based on your requirement to maintain high availability and fair use by protecting the backend resources from being overwhelmed by too many requests. You might also need to prevent denial-of-service (DoS) attacks or control and constrain resource consumption. Ultimately, you apply a rate limit globally to all the routes in an API deployment specification.

* **Service limits**

When designing your architecture, consider the service limits for the Streaming and Functions services. See the Service Limits documentation listed in the Explore More section.

* **Scalability**
  + **Database**

You can manually scale the number of CPU cores of the database up or down at any time. The autoscaling feature of autonomous databases allows your database to use up to three times the current base number of CPU cores at any time. As demand increases, autoscaling automatically increases the number of cores in use. Autonomous databases allow you to scale the storage capacity at any time without affecting availability or performance.

* + **Application**

You can scale your Flask application by using the instance pool and autoscaling features.

Instance pools enable you to provision and create multiple compute instances based on the same configuration within the same region.

Use autoscaling to automatically adjust the number of compute instances in an instance pool based on performance metrics, such as CPU utilization. Autoscaling helps you provide consistent performance for users during periods of high demand and reduce your costs when the demand is low.

* + **Functions**

Oracle Functions creates and removes function containers automatically based on the request load. You pay only when the functions are invoked and for the duration that they run.

* **Application availability**

Fault domains provide the best resilience within an availability domain. If you need higher availability, consider using multiple availability domains or multiple regions where feasible.

* **Backups**
  + **Database**

Oracle Cloud Infrastructure automatically backs up autonomous databases and retains the backups for 60 days. You can restore and recover your database to any point in time during the retention period. You can also create manual backups to supplement the automatic backups. Manual backups are stored in an Oracle Cloud Infrastructure Object Storage bucket that you create, and are retained for 60 days.

* + **Application**

The Oracle Cloud Infrastructure Block Volumes service lets you create point-in-time backups of data on a block volume. You can restore these backups to new volumes at any time.

You can also use the service to make a point-in-time, crash-consistent backup of a boot volume without application interruption or downtime. Boot and block volumes have the same backup capabilities.

* **Security**
  + **Access control**

Use policies to restrict who can access your resources in the cloud and the actions that they can perform.

**ALGORITHM USED:**

Support Vector Machines (SVMs) are primarily used for binary classification tasks where the goal is to separate data points into two classes. However, SVMs can also be adapted for anomaly detection. The idea is to treat anomaly detection as a one-class classification problem, where the SVM learns to classify data points into a single class (the "normal" class) and identifies anomalies as data points that do not conform to this normal class. This is known as "One-Class SVM."

Using a One-Class SVM (Support Vector Machine) for anomaly detection in serverless IoT data processing can be an effective approach. Here's how you can implement it:

**1.Data Collection and Preprocessing:**

• Collect data from your IoT devices and preprocess it. This may involve cleaning, aggregating, and transforming the data into a suitable format.

**2.Feature Engineering:**

•Identify relevant features from your IoT data that can help in anomaly detection. This may include metrics like CPU usage, memory usage, network traffic, or any other parameters relevant to your use case.

**3.Normalization:**

•Normalize the data to ensure that all features have similar scales. One-Class SVM is sensitive to the scale of the features.

**4.Data Split:**

•Split your data into two sets: a training set and a testing set. The training set will be used to fit the One-Class SVM model, while the testing set will be used to evaluate its performance.

**5.One-Class SVM Model:**

•Train a One-Class SVM model on the training data. One-Class SVM aims to find a hyperplane that separates the normal data points from the outliers (anomalies).

from sklearn.svm import OneClassSVM

model = OneClassSVM(nu=0.05) .

model.fit(training\_data)

**6.Model Evaluation:**

•Use the trained model to make predictions on the testing data. The decision function will return a negative value for anomalies and a positive value for normal data points.

predictions = model.decision\_function(test\_data)

**7.Threshold Selection:**

•Choose a threshold for the decision function's output to classify data points as anomalies or normal. The threshold can be adjusted based on the desired trade-off between false positives and false negatives.

**8.Anomaly Detection**:

•Identify data points with decision function values below the chosen threshold as anomaliesanomalies = test\_data[predictions < threshold]

**9.Deployment in a Serverless Environment:**

•Deploy your One-Class SVM model in a serverless environment, such as AWS Lambda or Azure Functions, to perform real-time anomaly detection on incoming IoT data.

**10.Monitoring and Alerting:**

•Set up monitoring and alerting mechanisms to notify you when anomalies are detected. This can be done through various cloud services or custom solutions.

**11.Model Retraining:**

•Periodically retrain your model with new data to adapt to changing patterns and ensure its effectiveness over time.

**12.Tuning and Optimization:**

•Experiment with different hyperparameters, such as the kernel type and nu value, to optimize the One-Class SVM model for your specific use case.

Anomaly detection using One-Class SVM is a powerful technique for IoT data processing, as it can identify abnormal behavior in a variety of IoT applications, including serverless environments. Keep in mind that the choice of parameters and the quality of your features play a crucial role in the success of the anomaly detection system.

**FEATURES:**

**1.Relevance:** Features should be relevant to the problem you're trying to solve. They should capture information that can help distinguish between normal and anomalous data.

**2.Unimodal Data**: One-Class SVM works well when the data distribution is unimodal (single-peaked). Ensure that your selected features reflect this characteristic.

**3.Numerical Features**: One-Class SVM typically works with numerical features, so if your data includes categorical or text data, you may need to transform or encode it into numerical representations.

**4.Normalization/Standardization**: Normalize or standardize your features to ensure that they have a consistent scale. This is particularly important when using the radial basis function (RBF) kernel.

**5.Dimensionality Reduction**: If you have a high-dimensional dataset, consider using dimensionality reduction techniques like Principal Component Analysis (PCA) to reduce the number of features while preserving as much variance as possible.

**Team Members:**

1.Abinaya V-2021506003

2.Devadharshini B-2021506015

3.Monisha H-2021506051

4.Padmapriya S-2021506059

5.Rudhra Y-2021506076