I'd be glad to provide a comprehensive guide on implementing time series anomaly detection for GCP resource usage with TensorFlow Probability (TFP) and Vertex AI, incorporating the best aspects of the previous responses and addressing any potential shortcomings:

**1. Data Acquisition:**

* **Identify relevant metrics:** Choose metrics that align with your specific anomaly detection goals (e.g., CPU usage, memory utilization, network traffic, disk I/O).
* **Utilize Stackdriver Monitoring API:**
  + Use the API to retrieve historical and real-time metric data for the selected metrics.
  + Consider using the gcloud monitoring metrics command-line tool for efficient data retrieval.
* **Leverage Cloud Monitoring logs:**
  + Collect logs from relevant GCP services (e.g., Compute Engine, Cloud Load Balancing) to gain insights into system behavior and potential anomalies.
* **Explore Cloud Billing API:**
  + If cost-related anomalies are of interest, use the Cloud Billing API to access billing data.
* **Store data in BigQuery:**
  + Store the collected data in BigQuery for efficient storage, querying, and analysis.

**2. Data Preprocessing:**

* **Handle missing values:** Use appropriate techniques like imputation (e.g., mean, median, mode) to fill missing data points.
* **Address outliers:** Identify and handle outliers using methods like statistical techniques (e.g., Z-score, IQR) or machine learning-based approaches.
* **Normalize data:** Scale the data to a common range (e.g., 0-1) using techniques like min-max scaling or standardization.
* **Feature engineering:**
  + Create new features that might be informative for anomaly detection (e.g., time-based features, ratios, differences).
  + Consider using domain knowledge to identify relevant features.

**3. Model Training with TFP and Vertex AI:**

* **Import necessary libraries:**

Python

import tensorflow as tfimport tensorflow\_probability as tfpfrom vertexai.experimental.lineage import Lineage

Use code [with caution.](/faq" \l "coding" \t "_blank)

* **Define the model:**
  + Choose a suitable TFP model for time series anomaly detection, such as:
    - **Seasonal ARIMA:** For stationary time series with seasonal patterns.
    - **Bayesian Structural Time Series:** For modeling complex time series with multiple components (e.g., trend, seasonality, cycle).
    - **Gaussian Process Regression:** For capturing non-linear relationships and uncertainty.
  + Customize the model's hyperparameters based on your data and domain knowledge.
* **Train the model on Vertex AI:**
  + Create a Vertex AI Training Job using a custom training script.
  + Provide the preprocessed data and model definition to the training job.
  + Configure the training job's parameters (e.g., number of epochs, batch size).

**4. Anomaly Detection:**

* **Real-time or batch processing:**
  + Determine if you need to detect anomalies in real-time using a streaming pipeline or in batches for offline analysis.
* **Use the trained model:**
  + Pass new data points to the trained model to obtain anomaly scores.
  + Consider using probabilistic methods like posterior predictive checks or anomaly score thresholds to identify anomalies.
* **Set anomaly thresholds:**
  + Define thresholds based on the model's output distribution and domain knowledge.
  + Consider using statistical methods or expert judgment to set appropriate thresholds.

**5. Alerting and Visualization:**

* **Set up alerts:**
  + Use Cloud Monitoring alerts to trigger notifications (e.g., email, Slack) when anomalies are detected.
  + Customize alert conditions based on anomaly scores and other relevant criteria.
* **Data visualization:**
  + Create visualizations (e.g., time series plots, anomaly score plots) to monitor resource usage and identify anomalies.
  + Consider using tools like Looker Studio or Data Studio for interactive visualizations.

**Additional Considerations:**

* **Model evaluation:** Regularly evaluate the model's performance using metrics like precision, recall, F1-score, and ROC AUC.
* **Retraining:** Retrain the model periodically with new data to adapt to changing conditions.
* **Experimentation:** Explore different models, hyperparameters, and feature engineering techniques to optimize performance.
* **Consider using Vertex AI Pipelines:** Automate the entire workflow from data ingestion to anomaly detection and alerting.
* **Leverage Vertex AI Explainable AI:** Gain insights into the model's decision-making process to understand the reasons behind anomalies.

By following these steps and incorporating the provided guidance, you can effectively implement time series anomaly detection for GCP resource usage using TFP and Vertex AI.



give me the complete end to end code to complete this task code

Show drafts



I'd be glad to provide a comprehensive end-to-end code example for implementing time series anomaly detection for GCP resource usage with TensorFlow Probability (TFP) and Vertex AI, incorporating the best aspects of previous responses and addressing potential shortcomings:

**1. Data Acquisition and Preprocessing:**

Python

import pandas as pdfrom google.cloud import monitoring\_v3from google.cloud import bigquery

# Replace with your GCP project ID and metric type

project\_id = "your-project-id"

metric\_type = "metric.googleapis.com/resource/cpu\_utilization"

# Retrieve metric data using Stackdriver Monitoring API

client = monitoring\_v3.MetricServiceClient()

time\_series = client.list\_time\_series(

request={

"filter": f'resource.type="gce\_instance" AND metric.type="{metric\_type}"',

"interval.start\_time": "2024-01-01T00:00:00Z",

"interval.end\_time": "2024-10-18T12:03:55Z",

}

)

# Extract timestamp and metric values

timestamps = []

values = []for series in time\_series.time\_series:

for point in series.points:

timestamps.append(point.interval.start\_time)

values.append(point.value.double\_value)

# Create a Pandas DataFrame

df = pd.DataFrame({"timestamp": timestamps, "value": values})

df["timestamp"] = pd.to\_datetime(df["timestamp"])

df.set\_index("timestamp", inplace=True)

# Preprocess the data (e.g., handle missing values, outliers, normalization)

df = df.fillna(method="ffill") # Forward fill missing values

df = df[(df["value"] > 0) & (df["value"] < 100)] # Remove outliers

df["value"] = (df["value"] - df["value"].min()) / (df["value"].max() - df["value"].min()) # Normalize

# Create additional features (e.g., time-based features, differences)

df["hour"] = df.index.hour

df["day\_of\_week"] = df.index.dayofweek

df["day\_of\_month"] = df.index.day

df["month"] = df.index.month

df["year"] = df.index.year

df["diff"] = df["value"].diff()

# Split into training and testing sets

train\_size = int(len(df) \* 0.8)

train\_data = df[:train\_size]

test\_data = df[train\_size:]

Use code [with caution.](/faq" \l "coding" \t "_blank)

**2. Model Training with TFP and Vertex AI:**

Python

import tensorflow as tfimport tensorflow\_probability as tfpfrom vertexai.experimental.lineage import Lineage

# Define the model (e.g., Seasonal ARIMA)

model = tfp.sts.SeasonalARIMA(

num\_timesteps=train\_data.shape[0],

num\_seasons=1,

period=24, # Assuming hourly data

exogenous=train\_data[["hour", "day\_of\_week", "day\_of\_month", "month", "year"]],

)

# Create a Vertex AI Training Job

training\_job = vertexai.training\_jobs.TrainingJob(

display\_name="time\_series\_anomaly\_detection",

training\_script\_path="your\_training\_script.py",

args=[

"--train-data", train\_data.to\_csv(index=False),

"--test-data", test\_data.to\_csv(index=False),

],

)

# Start the training job

training\_job.run()

Use code [with caution.](/faq" \l "coding" \t "_blank)

**3. Anomaly Detection:**

Python

import tensorflow as tf

# Load the trained model

model = tf.keras.models.load\_model("your\_model\_path")

# Make predictions on the test data

predictions = model.predict(test\_data[["hour", "day\_of\_week", "day\_of\_month", "month", "year"]])

# Calculate anomaly scores (e.g., using reconstruction error)

anomaly\_scores = tf.keras.losses.mean\_squared\_error(test\_data["value"], predictions)

Use code [with caution.](/faq" \l "coding" \t "_blank)

**4. Alerting and Visualization:**

Python

from google.cloud import monitoring\_v3

# Set up alerts based on anomaly scores

client = monitoring\_v3.AlertPolicyServiceClient()

alert\_policy = monitoring\_v3.AlertPolicy(

name="my\_alert\_policy",

display\_name="Anomaly Detection Alert",

condition=monitoring\_v3.Condition(

is\_is\_alerting\_condition=True,

condition\_threshold={

"operator": "GREATER\_THAN",

"value": anomaly\_threshold,

},

time\_series\_query="metric.type=\"custom.googleapis.com/anomaly\_score\"",

),

notification\_channels=["email:your\_email@example.com"],

)

client.create\_alert\_policy(parent="projects/your-project-id", alert\_policy=alert\_policy)

# Visualize anomaly scores and resource usage dataimport matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.plot(test\_data.index, test\_data["value"], label="Actual")

plt.plot(test\_data.index, predictions, label="Predicted")

plt.title("Anomaly Detection")

plt.xlabel("Timestamp")

plt.ylabel("Value")

plt.legend()

plt.show()

Use code [with caution.](/faq" \l "coding" \t "_blank)

Remember to replace placeholders like your-project-id, your\_training\_script.py, and your\_model\_path with your specific values. Adjust the model, hyperparameters, and alert conditions based on your requirements and data characteristics.