GUIDEWIRE DEV TRAILS

Team : Pentagon

Content

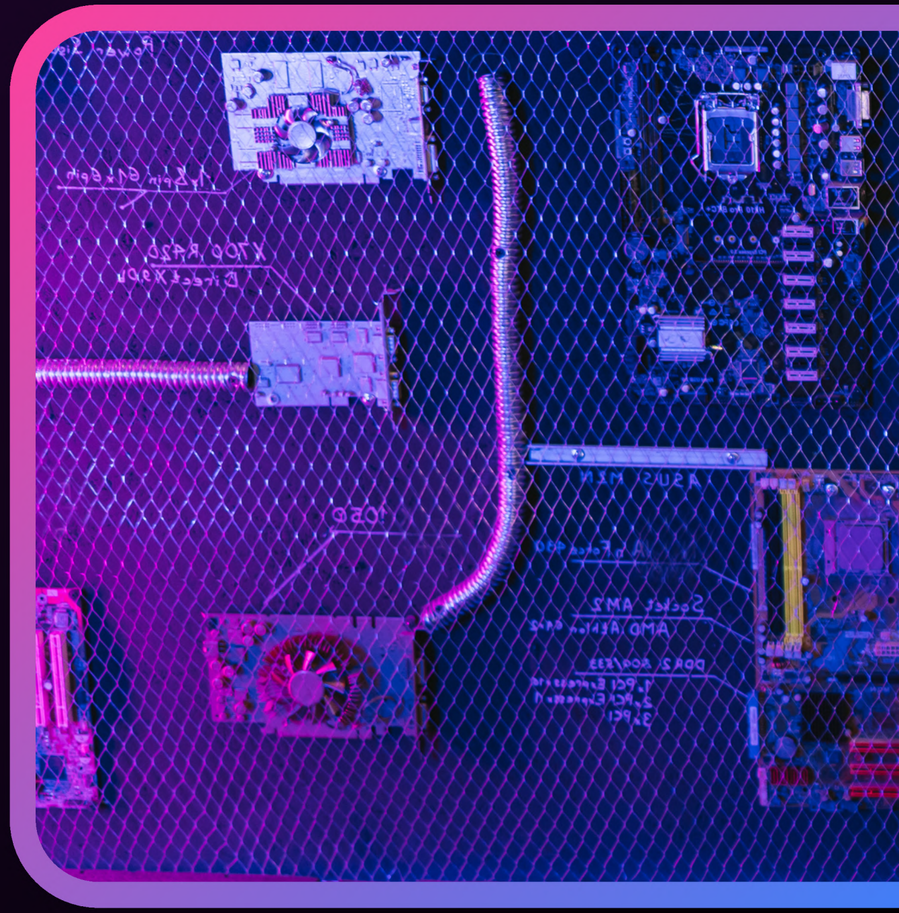
Problem Statement

Kubernetes clusters are complex and can encounter failures such as:

- Pod crashes due to misconfigurations or resource limits.
- Resource bottlenecks (CPU, memory, disk exhaustion).
- Network connectivity issues affecting service communication.
- Service disruptions caused by failures in logs and events.

Challenge:

- Predict these failures before they happen using AI/ML.
- Analyze historical and real-time cluster metrics.
- Improve cluster reliability and minimize downtime



Dataset overview

Source : Simulated Kubernetes failure dataset (GitHub Repository)

Description: The dataset contains synthetic Kubernetes cluster metrics, including system performance indicators and failure events.

Key Features:

- Timestamp – When the data was recorded.
- Node ID – Unique identifier for a Kubernetes node.
- Pod ID – Unique identifier for a running pod.
- CPU Usage (%) – CPU consumption of a pod/node
- Memory Usage (MB) – RAM consumption of a pod/node.
- Disk Usage (GB) – Storage consumption of a node.
- Network I/O (MB/s) – Ingress and egress network traffic.
- Pod Status – Running, Pending, Failed, or Unknown.
- Node Status – Healthy, Unhealthy, or Unknown.
- Failure Type – Type of failure (if any), such as CPU Exhaustion, Memory Overload, Network Failure, etc.

Dataset overview - snippet of some rows and columns

Timestamp	datetime64[ns]
Pod Name	int64
CPU Usage (%)	float64
Memory Usage (%)	float64
Pod Status	int64
Pod Restarts	int64
Ready Containers	int64
Total Containers	int64
Pod Event Type	int64
Pod Event Reason	int64
Pod Event Age	timedelta64[ns]
Pod Event Source	int64
Pod Event Message	object
Node Name	int64
Event Reason	int64
Event Age	timedelta64[ns]
Event Source	int64
Event Message	object
Memory Usage (MB)	float64
Active Memory Requests (MB)	int64
Memory Requests (%)	float64
Memory Usage (Cache) (MB)	float64
Network Receive Bytes	float64
Network Transmit Bytes	float64
Network Receive Packets (p/s)	float64
Network Transmit Packets (p/s)	float64
Network Receive Packets Dropped (p/s)	float64
Network Transmit Packets Dropped (p/s)	float64

```
[ ] import pandas as pd
    df=pd.read_csv("dataSynthetic.csv")
```

```
df.head()
```

	Timestamp	Pod Name	CPU Usage (%)	Memory Usage (%)	Pod Status	Pod Reason	Pod Restarts	Ready Containers	Total Containers	Pod Event Type	...	Network Receive Packets (p/s)	Network Transmit Packets (p/s)	Network Receive Packets Dropped (p/s)	Network Transmit Packets Dropped (p/s)	FS Reads Total (MB)	FS Writes Total (MB)	FS Reads/Writes Total (MB)	FS Reads Bytes Total (MB)	FS Writes Bytes Total (MB)	FS Reads/Writes Bytes Total (MB)
0	2024-04-19 21:05:48	opentelemetry-demo-redis-68779558bb-4mjtr	0.230313	53.101612	Running	NaN	0.0	1.0	1.0	No recent events	...	3.768307	1.135357	2.153989	1.185063	0.000485	0.001514	0.001785	0.000310	0.000000	0.000331
1	2024-04-19 20:58:41	opentelemetry-demo-frontend-76f486559f-szlsh	14.612577	0.141409	Running	NaN	2.0	1.0	1.0	Normal	...	156.855681	356.949251	1.460284	10.050480	0.001362	0.000666	0.001240	0.000049	0.000000	0.000000
2	2024-04-18 22:13:38	opentelemetry-demo-frauddetectionservice-64cb6...	0.603365	12.191371	Running	NaN	0.0	1.0	1.0	Normal	...	4.474349	3.042058	3.968989	4.581325	0.001177	0.328530	0.357402	0.000326	0.003142	0.003002
3	2024-04-22 12:30:14	opentelemetry-demo-frauddetectionservice-64cb6...	0.077004	0.201204	Unknown	NaN	0.0	0.0	0.0	No recent events	...	1.393119	1.345098	0.861935	5.033863	0.000000	0.000771	0.000000	0.000396	0.000000	0.000205
4	2024-04-22 15:54:45	opentelemetry-demo-recommendationservice-7697d...	1.060736	43.678164	Running	NaN	0.0	1.0	1.0	No recent events	...	0.050636	0.094615	2.542782	7.614585	0.000000	0.001226	0.000493	0.000000	0.000279	0.000000

5 rows × 38 columns

Data Preprocessing & Data Preparation

1) Handled Missing Values
Checked for null values using .isnull().sum().
Found no missing values in most columns, so imputation wasn’t necessary.
2) Feature Selection & Cleaning
Dropped redundant columns:
"Memory Usage (MB).1" (high correlation with "Memory Usage (MB)").
Unnecessary log/event details that don’t impact predictions.
Standardized column names for consistency.
3) Data Type Conversions
Converted Event Age (HHMMSS format) to timedelta
Converted timestamps to datetime format for time-based analysis.
4)Numerical Transformations
CPU & Memory Usage: Rounded to 3 decimal places for consistency.
Highly skewed features (e.g., Memory Requests %): Considered log transformation but left unchanged as tree-based models handle skewness well.
5) Encoded Categorical Data
Mapped categorical columns like Pod Status, Event Type, Event Source to numeric values using dictionary mappings.

Model Selection and Approach

Pod failures

TYPE OF MODEL USED

- Anomaly Detection: Isolation Forest (Unsupervised Learning)
- Time-Series Forecasting: LSTM (Long Short-Term Memory Network) (Supervised Learning)

```python
# Build LSTM Model
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(seq_length, X.shape[2])),
    Dropout(0.2),
    LSTM(32, return_sequences=False),
    Dropout(0.2),
    Dense(1, activation="sigmoid")
])

# Compile Model
model.compile(loss="binary_crossentropy", optimizer=Adam(learning_rate=0.001), metrics=["accuracy"])

# Train Model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(**kwargs)
Epoch 1/10
887/887 ━━━━━━━━━━━━━━━━━━━━ 12s 9ms/step - accuracy: 0.7437 - loss: 0.5390 - val_accuracy: 0.8964 - val_loss: 0.3361
Epoch 2/10
887/887 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.7499 - loss: 0.5192 - val_accuracy: 0.8964 - val_loss: 0.3374
Epoch 3/10
887/887 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - accuracy: 0.7474 - loss: 0.5193 - val_accuracy: 0.8964 - val_loss: 0.3368
```

```python
# Train Isolation Forest on training data
iso_forest = IsolationForest(n_estimators=100, contamination=0.05, random_state=42)
iso_forest.fit(X_train.reshape(X_train.shape[0], -1))  # Flatten time-series

# Predict anomalies (-1 means anomaly, 1 means normal)
anomaly_scores = iso_forest.predict(X_test.reshape(X_test.shape[0], -1))
anomaly_scores = np.where(anomaly_scores == -1, 1, 0)  # Convert to binary (1 = anomaly)

# Combine LSTM and Isolation Forest results
final_predictions = np.logical_or(model.predict(X_test).flatten() > 0.5, anomaly_scores).astype(int)

# Evaluatez
print("Accuracy:", accuracy_score(y_test, final_predictions))
print(classification_report(y_test, final_predictions))
```

```
222/222 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
Accuracy: 0.8420236753100339
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      6361
           1       0.05      0.03      0.04       735

    accuracy                           0.84      7096
   macro avg       0.47      0.48      0.48      7096
weighted avg       0.81      0.84      0.82      7096
```

Detecting pod failures

OVERVIEW OF IMPLEMENTATION:

- Data Preprocessing – Load dataset, normalize features (CPU, memory, network), convert pod status to binary, and create time-series sequences.
- Anomaly Detection (Isolation Forest) – Identifies abnormal resource usage patterns that could indicate pod failures.
- Time-Series Prediction (LSTM) – Uses past 10 timestamps to predict future pod failures based on sequential trends.
- Hybrid Model Combination – Merges predictions from both models, flagging failures if either model detects an issue.
- Evaluation & Results – Validates performance using accuracy, classification report, and anomaly detection effectiveness.

Justification for chosen approach

1. ISOLATION FOREST FOR ANOMALY DETECTION

- Detects rare or abnormal patterns in Kubernetes pod behavior.
- Works well for high-dimensional data without requiring labeled failure examples.
- Identifies unexpected spikes in resource usage (CPU, memory, network) leading to failures.

2. LSTM FOR TIME-SERIES FORECASTING

- Captures temporal dependencies in sequential Kubernetes metrics.
- Predicts pod failures by learning long-term trends in resource utilization.
- Uses past behavior (last 10 timestamps) to classify future pod status as normal or failure.

3. HYBRID APPROACH (COMBINING BOTH)

- Isolation Forest flags anomalies in test data.
- LSTM Model predicts failures based on learned sequential patterns.
- Final prediction is a combination of both methods, improving accuracy and robustness.

Resource Exhaustion

Models used:

1. LSTM (Long Short-Term Memory)
2. XGBoost Classifier

```
Accuracy: 0.9550610820244329
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      2189
           1       0.00      0.00      0.00       103

    accuracy                           0.96      2292
   macro avg       0.48      0.50      0.49      2292
weighted avg       0.91      0.96      0.93      2292
```

Detecting resource exhaustion

OVERVIEW OF IMPLEMENTATION:

- Loading and preprocessing the dataset containing system metrics (CPU usage, memory usage, disk operations).
- Normalising feature values using MinMaxScaler for better model performance.
- Creating binary labels for resource exhaustion detection, marking instances where CPU or memory usage exceeds 90%.
- Using LSTM (Long Short-Term Memory) to model time-series patterns and detect resource exhaustion.
- Training an XGBoost Classifier as an alternative model to compare performance.
- Evaluating model accuracy and generating a classification report to assess predictive capability.

Reason for choosing these models

1. LSTM (LONG SHORT-TERM MEMORY)

- Effective for sequential data (e.g., system resource logs).
- Captures temporal dependencies, identifying trends before resource exhaustion occurs.
- Unlike traditional models (e.g., logistic regression, decision trees), LSTMs can handle long-range dependencies.

2. XGBOOST CLASSIFIER

- Provides an alternative approach that handles structured data well.
- Works efficiently with tabular features and can offer fast predictions.
- Helps in comparative analysis to see if a deep learning model (LSTM) performs significantly better.

WHY NOT OTHER MODELS?

- Linear models assume independence between observations and miss sequential patterns.
- Random Forests / Decision Trees work well for classification but don’t capture time-series trends effectively.
- CNNs are suited for spatial data rather than sequential dependencies.

Network issues

OVERVIEW OF IMPLEMENTATION:

- Loading network-related system metrics from a dataset (e.g., network receive/transmit bytes, packets dropped).
- Converting timestamps to datetime format and sorts for time-series analysis.
- Computing total network packets dropped as a key indicator of network exhaustion.
- Normalising feature values using MinMaxScaler to improve model performance.
- Models used: LSTM to detect network resource exhaustion from sequential data, XGBoost Classifier

```
Accuracy: 0.9522452245224522
              precision    recall  f1-score   support

           0       0.33      0.00      0.00       954
           1       0.95      1.00      0.98     19044

    accuracy                           0.95     19998
   macro avg       0.64      0.50      0.49     19998
weighted avg       0.92      0.95      0.93     19998
```

Reason for choosing these models

1. LSTM (LONG SHORT-TERM MEMORY)

- Detects patterns in network activity over time.
- Captures long-term dependencies, allowing early detection of network issues.
- Works better than traditional models, which assume independent observations.

2. XGBOOST CLASSIFIER

- Provides a fast, tree-based alternative that performs well on structured data.
- Helps compare deep learning (LSTM) vs. machine learning (XGBoost).

WHY NOT OTHER MODELS?

- Logistic Regression & Decision Trees – Cannot model time dependencies.
- Random Forest – Good for tabular data but fails on sequential patterns.
- CNNs – Best suited for images rather than network activity trends.

CHALLENGES FACED

DIFFICULTY IN NLP-BASED VECTORIZATION OF POD EVENT MESSAGES

- The Pod Event Message field contains unstructured text, making it challenging to apply NLP techniques for meaningful vectorization.
- Messages often include varying details like timestamps, container statuses, and specific error logs, requiring extensive preprocessing and domain-specific embeddings.

LACK OF DIRECT CORRELATION BETWEEN EVENT REASON & EVENT MESSAGE

- Event Reason (e.g., OOMKilling, RegisteredNode) does not always map clearly to the Pod Event Message, making it difficult to establish a straightforward cause-effect relationship.
- This weak correlation affects predictive modeling, as event patterns are not easily extractable without deeper context analysis.