6 - Detección de Fraude con Kapacitor y Monitoreo en Vivo

December 29, 2020

1 6 - Detección de Fraude con Kapacitor y Monitoreo en Vivo

Este cuaderno se utiliza para simular datos de entrada utilizando el dataset de test. Se generan datos de nuevas operaciones periodicamente que se cargan en InfluxDB y pueden ser visualizadas en Grafana.

Además, se utiliza el servicio Kapacitor (y Cronograph como administrador de sus reglas y alertas activas) para ejecutar un servicio de procesamiento básico que por cada nuevo dato recibido *TMV_CREDIT_OPERATION* lo clasifica utilizando el servicio REST presentado anteriormente y genera una nueva variable llamada *TMV_CREDIT_OPERATION_RESULT* en la base de datos con el resultado del análisis.

El script de Kapacitor 'FraudDetection.tick' en este caso es sencillo:

```
dbrp "my_application"."autogen"

stream
    |from()
        .measurement('TMV_CREDIT_OPERATION')
    @udf_ml_model()
    |influxDBOut()
        .database('my_application')
        .measurement('TMV_CREDIT_OPERATION_RESULT')
```

Kapacitor opera de dos maneras. Por lote (batch) o por flujo (stream). En el primer caso se ejecuta una regla para un grupo de muestras ocurridas en un intervalo de tiempo. En el segundo caso se activa una regla ante la llegada de una nueva muestra. Aquí se utiliza el modo flujo porque interesa reaccionar ante cada nueva operación para determinar si debe activarse la alarma de fraude. El código anterior se suscribe a las variables llamadas TMV_CREDIT_OPERATION y ante cada nueva muestra le aplicará la función *udf_ml_model* (esta función se comunica por medio de un socket con un agente de Kapacitor que permite ejecutar funciones externas escritas en otros lenguajes, como por ejemplo Python o Go).

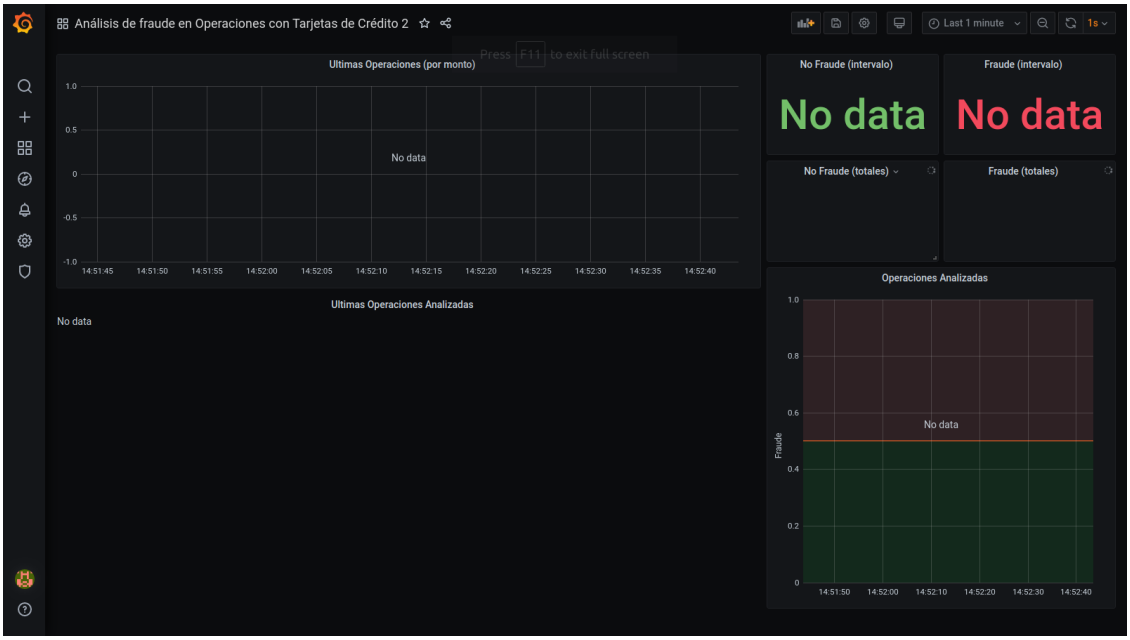
La función *udf_ml_model()* está implementada en un container llamado kapacitor_udf siguiendo los instructivos y tutoriales del sitio oficial: Kapacitor - Kapacitor is a real-time streaming data processing engine.

1.1 6.1 Configuración de InfluxDB, Kapacitor, UDF y Grafana

A continuación se indica como configurar los servicios y monitorear operaciones simuladas.

1. Abrir dashboard Grafana - analisis-de-fraude-en-operaciones-con-tarjetas-de-credito. (En caso de que no esté importarlo del directorio 'dashboards' adentro de compose/grafana/dashboards).

Cuando no se están recibiendo datos se debería observar algo así:



Por problemas en la secuencia de inicio de servicios, a veces Kapacitor y el agente de Kapacitor para las User Defined Functions (UDF) pueden no haberse iniciado por errores de conexión o de acceso a recursos. Para solucionarlo, basta con reiniciarlos en este orden:

2. Reiniciar UDF de kapacitor

```
cd compose
./restart-kapacitor-udf.sh
```

Debe verse:

```
2020-12-29 02:02:12,257 INFO:root: Started server. Listening at: /tmp/comm/udf_ml_model.sock
```

3. Reiniciar kapacitor.

```
cd compose
./restart-kapacitor.sh
```

4. Conectarse a Cronograph y verificar que la conexión con InfluxDB y Kapacitor está establecida.
5. Verificar que el TICKScript FraudDetection está habilitado.

1.2 6.2 Inyección de datos (mock)

El código que sigue es para generar datos periódicamente obtenidos del dataset de test (se utiliza muestreo aleatorio con reemplazo).

[1]:
```
DATASET_PATH = '/data/credit_fraud/'
```

[2]:
```
import pandas as pd
import numpy as np
```

[3]:
```
test_df = pd.read_csv(DATASET_PATH+"creditcard_test.csv")
```

[4]:
```
non_feature_cols = ['Unnamed: 0','time','row_id','class']
feature_cols = [x for x in test_df.columns if x not in non_feature_cols]
feature_cols
```

[4]:
```
['v1',
 'v2',
 'v3',
 'v4',
 'v5',
 'v6',
 'v7',
 'v8',
 'v9',
 'v10',
 'v11',
 'v12',
 'v13',
 'v14',
 'v15',
 'v16',
 'v17',
 'v18',
 'v19',
 'v20',
 'v21',
 'v22',
 'v23',
 'v24',
 'v25',
 'v26',
 'v27',
 'v28',
 'amount']
```

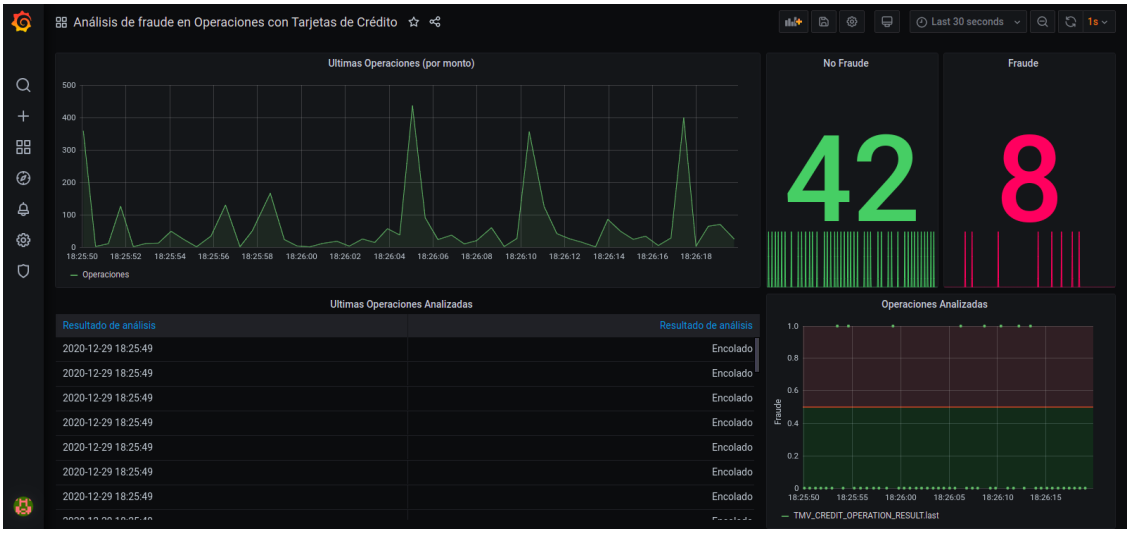
Mock para inyectar valores de prueba en InfluxDB

[5]:
```
import threading
import time
import math
import datetime
```

```
from influxdb import InfluxDBClient

class InfluxDBPublisher(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.keep_running = True
        self.client = InfluxDBClient(host='influxdb', port=8086,␣
 ↪username='root', password='root',
                                     database="my_application")
    def run(self):
        while self.keep_running:
            time.sleep(0.5)
            json_body = []
            ts = datetime.datetime.utcnow()
            row = test_df.sample(1)
            fields = {}
            for f in feature_cols:
                fields[f] = row[f].values[0]
            json_body.append(
                {
                    "measurement": f"TMV_CREDIT_OPERATION",
                    "time": ts,
                    "fields": fields
                }
            )
            self.client.write_points(json_body)

    def stop(self):
        self.keep_running = False
        self.join()


pub = InfluxDBPublisher()
pub.start()
input("Press any key to stop")
pub.stop()
```

```
Press any key to stop
```

Cuando están funcionando correctamente todos los servicios deberán verse tanto las operaciones publicadas como los resultados de análisis.


Análisis de fraude en Operaciones con Tarjetas de Crédito
Last 30 seconds
1s
Ultimas Operaciones (por monto)
500
400
300
200
100
0
18:25:50 18:25:52 18:25:54 18:25:56 18:25:58 18:26:00 18:26:02 18:26:04 18:26:06 18:26:08 18:26:10 18:26:12 18:26:14 18:26:16 18:26:18
Operaciones
No Fraude
42
Fraude
8
Ultimas Operaciones Analizadas
Resultado de análisis
Resultado de análisis
2020-12-29 18:25:49 Encolado
2020-12-29 18:25:49 Encolado
2020-12-29 18:25:49 Encolado
2020-12-29 18:25:49 Encolado
2020-12-29 18:25:49 Encolado
2020-12-29 18:25:49 Encolado
2020-12-29 18:25:49 Encolado
Operaciones Analizadas
1.0
0.8
0.6
0.4
0.2
0
Fraude
18:25:50 18:25:55 18:26:00 18:26:05 18:26:10 18:26:15
TMV_CREDIT_OPERATION_RESULT.last