Large Scale User Behavior Analytics by Flink

Data Driven Security

Hao WU

Jan, 2017

Agenda

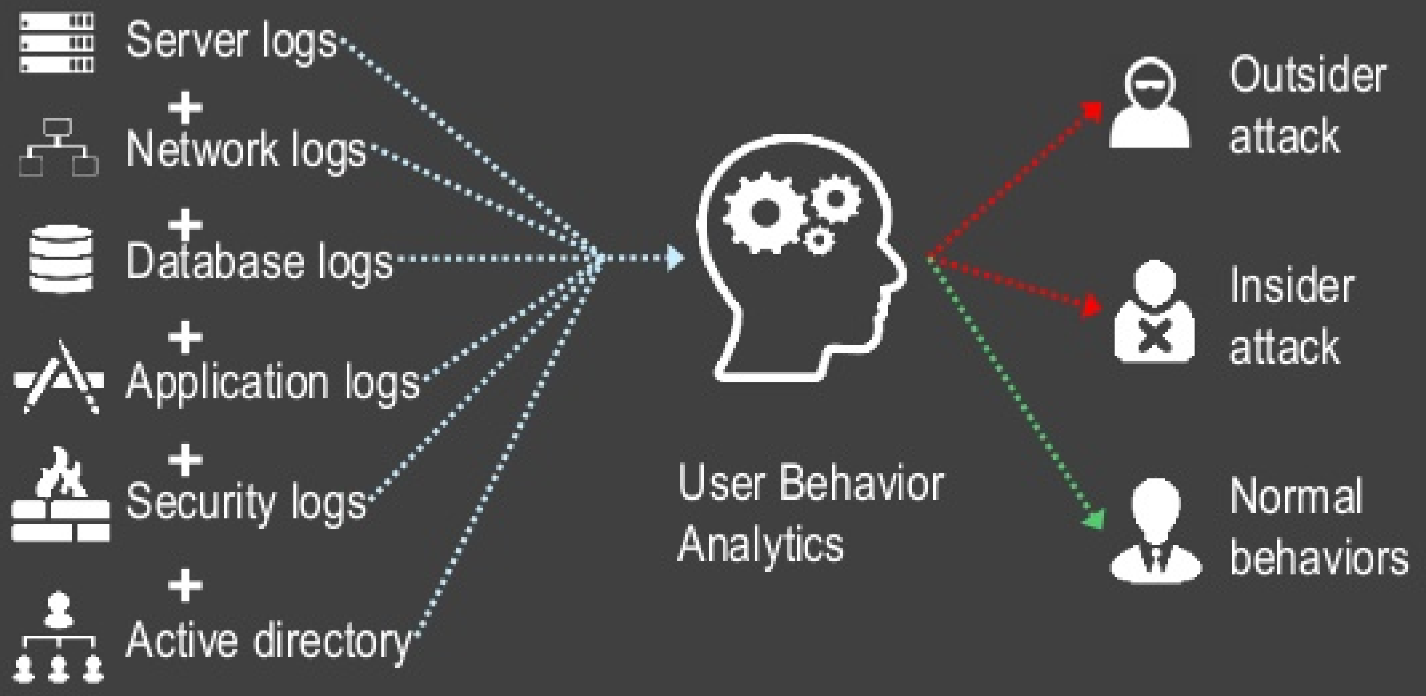
- User Behavior Analytics (UBA) in Cybersecurity
- Technical Challenges for Real-Time Large Scale UBA
- CEP Engine with Rules
- Flink Modifications

Who am I

- Work in HanSight, a leading cybersecurity startup in China
- Software architect of data processing team for user behavior analytics
- Live in Chengdu, city of panda
- Hobbies: data processing technologies, information visualization
- hao_wu@hansight.com

What's User Behavior Analytics in Cybersecurity?

- Detect outsider and insider attacks by finding user behavior anomalies
- Outsider attack: e.g. external hacker cracks VPN password and takes over the accounts of employees
- Insider attack: e.g. disgruntled employee steals sensitive information
- Used to be purely rule-based analysis in offline batch mode, now most vendors use some forms of machine learning (unsupervised outlier analysis) methods to do online/streaming analysis



Technical Challenges

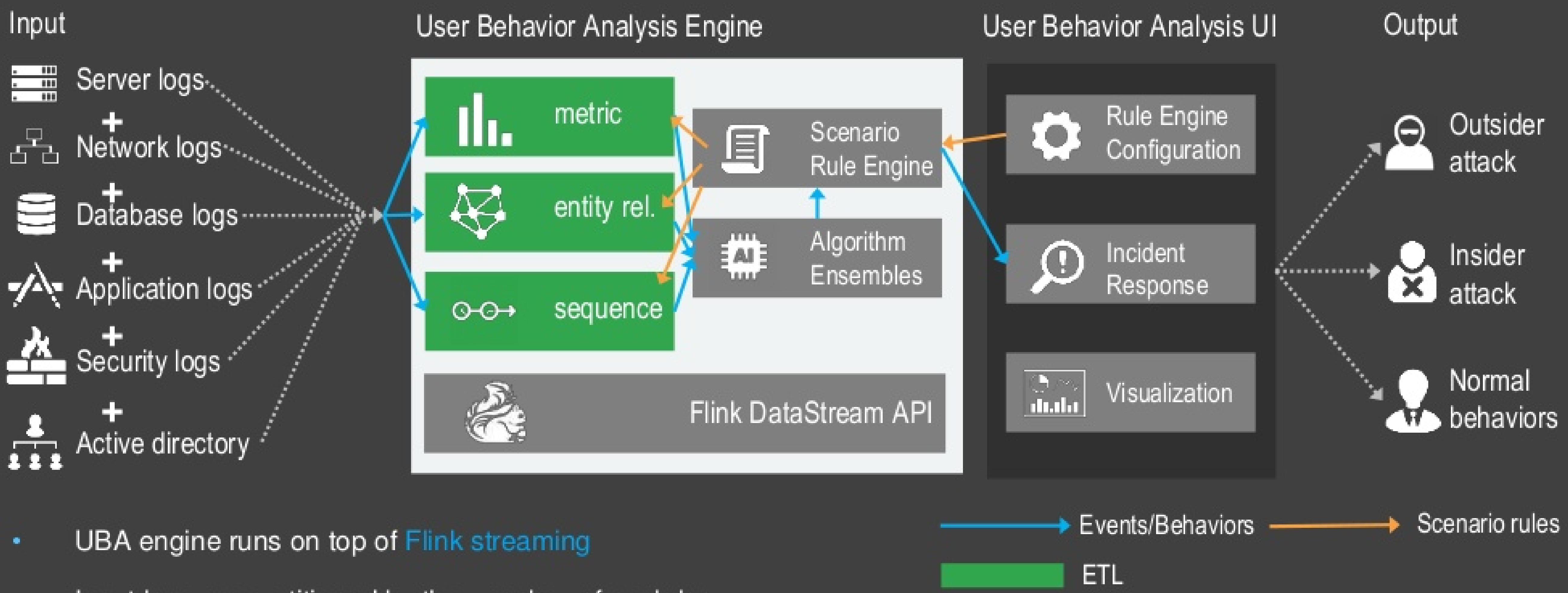
1. Typical UBA deployment in large enterprise needs to handle analyzing 10K+ unique users over 10+ dimensions
2. Detection of some attacks needs to be real time
3. Detection logic is a mixture of blacklists, rules, and machine learning algorithms
4. Detection logic needs to be customizable in near real time

While 1 and 2 can easily be solved by Flink, 3 and 4 needs extra work

Why Flink?

- Streaming in nature
- High-variety of data sources supported (i.e. CSV, Kafka, Hbase, Socket, etc)
- RocksDB as data storage backend
- High throughput (100K TPS)
- Flexible windowing capability (tumbling, sliding and the combination)
- Highly customizable operators to meet specific business logic

UBA Architecture



- UBA engine runs on top of Flink streaming
- Input logs are partitioned by the *user* key of each log
- Uses a modified version of Drools as the scenario rule engine
- Kafka and Elasticsearch are omitted from the diagram for simplicity

HanSight 瀚思

Why Drools instead of Flink Native CEP

- Rule engine is the most complicated component
- Flink native CEP API is generic, powerful and easy to use.
- Perfect for real-time streaming events analysis.

- However, our scenarios requires more dynamicity
- Drools rules can be hot deployed to a running system without recompile and restart of the system

HanSight 瀚思

What is Drools?

- Drools is a business rule management system (BRMS) with a forward and backward chaining inference based rules engine

```
package com.example;
import com.example.Person

rule "example rule"
when
    p: Person( name=="Michael" )
then
    p.name = "other";
    System.out.println(p.name);
end

declare EventA
    @role( event )
end

rule "Timeout EventA"
when
    $a1 : EventA()
    $a2 : EventA(this after[5s,10s] $a1)
then
    retract($a1);
end
```

UBA with Drools (Pros and Cons)

- Pros
  - Simple rule language, easy to write rules
  - Seamless integration with Java
  - Dynamic configuration of drools rules
- Cons
  - Built in aggregation functionality is slow -> use Flink for tumbling window aggregation
  - Built in event series processing is resource consuming -> UBA manages the lifecycle of events

ETLs Needed by the Scenario Rule Engine

Each raw event/log is processed by 3 types of ETL: *metric*, *entity-relationship*, and *sequence* to be converted into behaviors:

1. Metric: aggregated value within a specific tumbling window, e.g., how many logons in one hour
2. Entity-relationship: connections between two entities, e.g., user uses which device
3. Sequence: logs filtered, then sorted by timestamp, and finally converted to *states*, and ordered

All 3 ETLs are defined in the scenario rules and thus hot-deployable

Hot Deployment

- Use *CoFlatMapFunction* to process two streams: rules and logs
- ETL processors parse only related configurations (*type* and *expression* in each behavior definition) from rules
- Scenario rule engine parses the whole rules

ETL Defined in Rules

```
rule "Suspicious VPN activity"
when
    $b1: Behavior(type == "metric/1h",
                expression == "vpn.type == \"logon\" && vpn.result == \"failure\"",
                anomalyScore > 50)
    $b2: Behavior(type == "er",
                expression == "[vpn.user, vpn.device]",
                anomalyScore > 50,
                this coincides $b1)
    $b3: Behavior(type == "sequence",
                anomalyScore > 50,
                this coincides $b1)
then
{
    alertCollector.collect($b1, $b2, $b3)
}
end
```

1 hour tumbling window metric over failed VPN logons

Entity relationship between the vpn user and the used device

Sequence using default expression settings

* *VPN* events are part of the built-in ontology dictionary

HanSight 瀚思

Long Term Behaviors

- By default, algorithms analyze long term (> 3 month) behaviors to calculate *anomalyScore*
- Possible ways of accessing long term behaviors previously generated by the *metric* or *entity relationship* ETL:
  1. In Drools engine
  2. In an external DB
  3. As persistent operator state

Problems

- Need to maintain previous window state (as intermediate result for drools rule engine) for a certain amount of time.
- Flink built in window mechanism emits output and clears the window state when window is over.
- Flink built in RocksDB backend deletes records when window is purged.
- Results from Flink aggregation flood into drools rule engine for evaluation and you may run out of memory quickly

How to Tackle

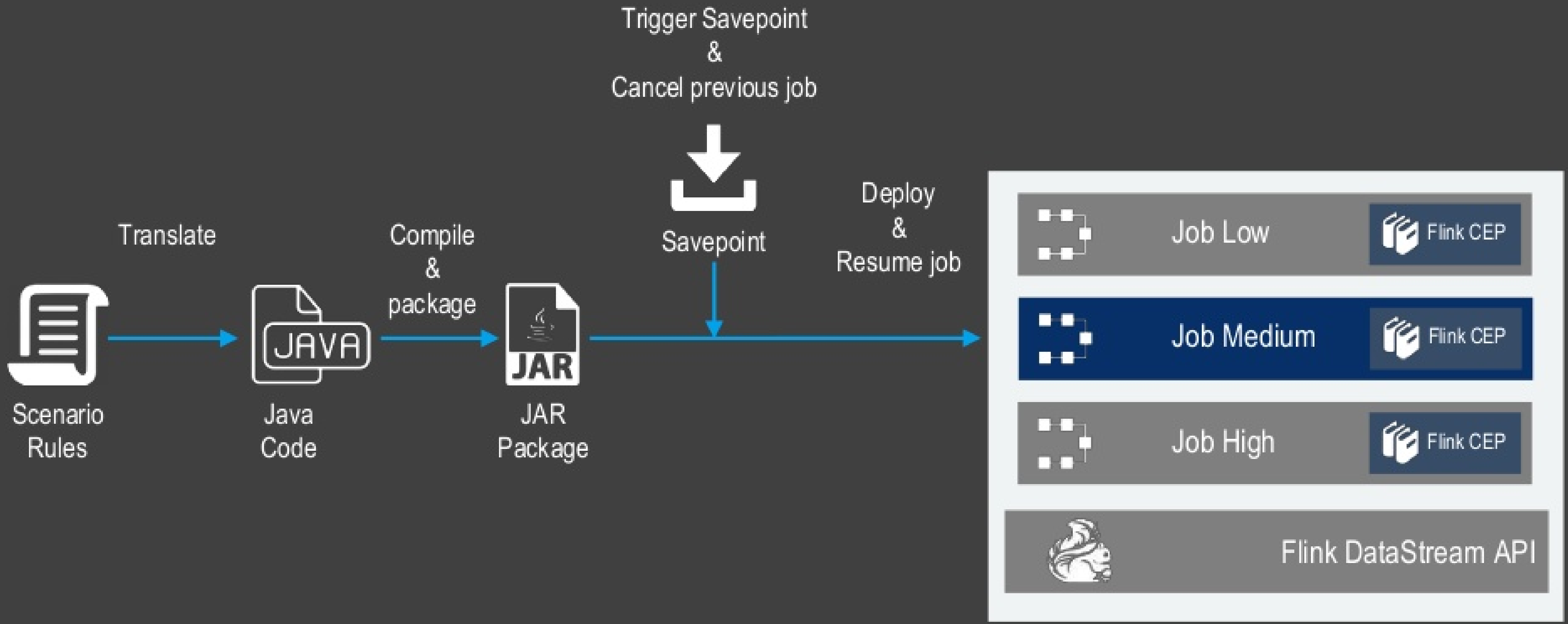
- Use external key/value stores like redis, memcached, etc for intermediate aggregation results.
    - Not an option due to throughput bottleneck
- Change the Flink RocksDB backend implementation.
    - Add “TTL” property to RocksDB instead of deleting its entries explicitly
    - Make “TTL” configurable on web interface
- Optimize drools memory management by setting a threshold for the number of facts allowed in memory and cleaning up unused facts when they are inactive.

Plan B: Flink CEP version

Flink recently add several new features which can be used to solve the "can't hot deployed" problem

1. Trigger savepoint, cancel job and resume job
2. State of unique identified operator in a savepoint
3. [FLINK-6927] Support pattern group in CEP Flink 1.4?
4. [FLINK-7129] Dynamically changing patterns open issue

Workflow



* Rules are package into several job Jars by their estimated complexity

HanSight 瀚思

Optimize DAG

- 1 pattern = 1 stream causes slow initialization and OOM for 1K+ patterns
- N pattern = 1 stream mode?
  - CEP API only allows 1 pattern = 1 stream
  - Merge multiple patterns into one using GroupPattern
  - No optimization for multiple patterns yet

```
Pattern<Event, Event> pattern5 =
        Pattern.<Event>begin("5").where(
            new SimpleCondition<Event>() {
                @Override
                public boolean filter(Event e) {
                    return e.type == "Logon";
                }
            });

Pattern<Event, Event> patterns =
        Pattern.begin(pattern1).optional()
                .next(pattern2).optional()
                .next(pattern3).optional()
                .next(pattern4).optional()
                .next(pattern5).optional()
```

Pros and Cons

- Pros
  - Simpler implementation, only 1/5 code of the Drools version
  - Better scalability and more parallelism, no more one big operator
  - Easier to get runtime metrics of each pattern
- Cons
  - Seconds delay for the savepoint-and-resume deployment
  - Slower performance in low/medium throughput scenario

Next Steps

- Direct Drools to Java translator
- Optimization at event pattern level for better performance
-

Q & A

Thank You | HanSight 瀚思

www.hansight.com

Phone: (+86 10) 8282 6616

Email: contact@HanSight.com