RedBorder CEP documentation

redborder

Version 1.0.0

Índice

Capítulo 1: Introducción a redBorder CEP

En este capítulo daremos una breve introducción a redBorder CEP y a su arquitectura.

RedBorder CEP

Un procesador de eventos complejos o CEP (De sus siglas en inglés: Complex Event Processing), es un procesador de eventos que combina los datos de varias fuentes para inferir eventos o patrones que sugieran las circunstancias más complejas.

redBorder CEP permite la ejecución de un conjunto de reglas, las cuales pueden inferir eventos, patrones y secuencias. Dispone de una API Rest a través de la cual un usuario puede añadir, eliminar o listar las reglas en tiempo real.

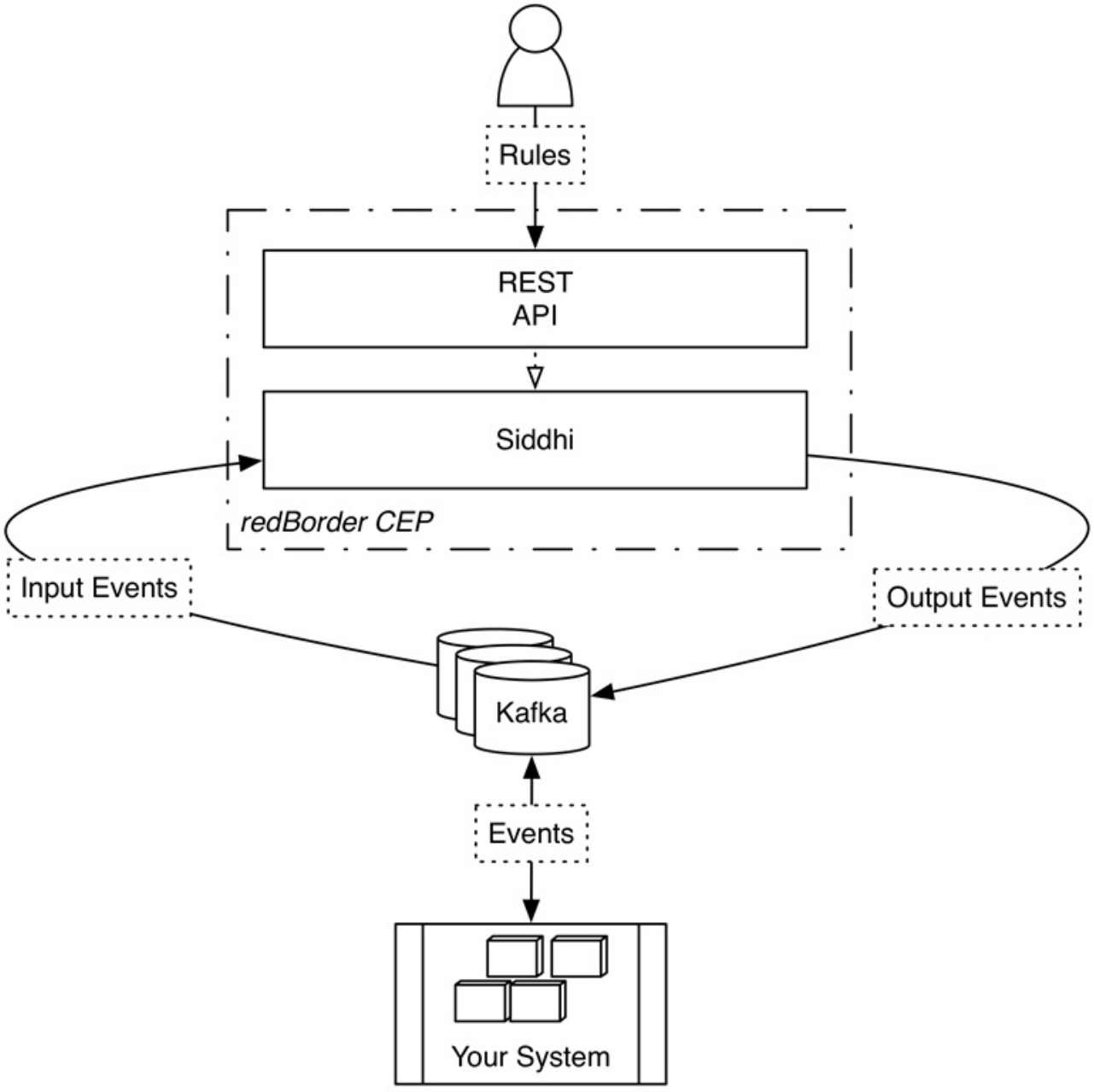

Figure 1. Arquitectura de redBorder CEP

El motor encargado de realizar las funciones del CEP es Siddhi. Siddhi CEP es un motor de procesamiento de eventos complejos en tiempo real creado por WSO2 ligero y fácil de usar escrito en Java bajo licencia Apache v2.0. Siddhi puede recibir eventos de fuentes externas y analizarlos en base a las especificaciones del usuario para posteriormente notificar los eventos apropiados.

En la arquitectura de redBorder CEP los eventos consumidos por Siddhi provienen de uno o varios topics de entrada de Kafka. El resultado de los cálculos y eventos son insertados en un nuevo topic o conjunto de topics de Kafka.

Siddhi tiene muchas características entre ellas podemos destacar:

- Funciones de agregación como sumas, máximos, mínimos, media, desviación estándar y conteo.
- Filtro y proyección de consultas usando expresiones matemáticas y lógicas.
- Valores por defecto en los atributos.
- Funciones integradas utiles para el procesamiento de eventos como, por ejemplo, ventanas de tiempo.
- Renombrado de atributos == Capítulo 2: Correlación de Eventos Complejos

En este capítulo introduciremos algunos conceptos básicos sobre la correlación de eventos complejos. En primer lugar definiremos los conceptos básicos y elementos.

Flujo de eventos

Siddhi utiliza el concepto de flujo de eventos para referirse a una secuencia de eventos ordenados en el tiempo con un esquema de atributos definido. Uno o varios de estos eventos pueden ser importados y manipulados usando consultas para identificar condiciones de eventos complejos. Estas consultas generarán nuevos eventos que son notificados como respuesta.

Un flujo de eventos está identificado por un nombre y tiene un conjunto de atributos asociados inequívocamente identificables, definiendo su esquema. Los atributos tienen un nombre único dentro del flujo de eventos y pueden ser de los siguientes tipos: `string`, `int`, `long`, `float`, `double`, `bool` u `object`.

Los eventos están asociados únicamente a un flujo y tienen un conjunto de atributos idénticos con tipos específicos. Un evento debe contener una marca de tiempo y los atributos de acuerdo a su esquema.

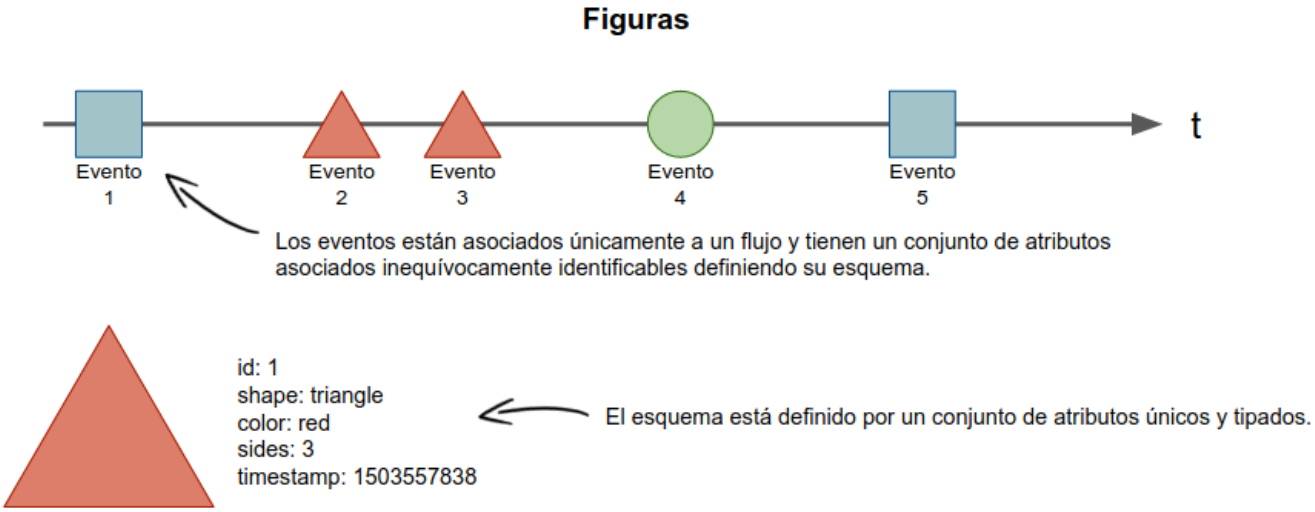

Figure 2. Flujo de eventos

Supongamos que definimos un flujo de eventos llamado "**figuras**" (Véase **Figure 1**) en el cuál se irá recibiendo una figura aleatoria a lo largo del tiempo. Un esquema válido para este flujo sería: figura = { id: int, shape: string, color: string, sides: int, timestamp: long }

Consultas

Siddhi dispone de un lenguaje DSL que permite operar con los eventos recibidos en un flujo. Con este lenguaje se pueden diseñar consultas.

Las consultas son construcciones lógicas que deriva nuevos streams al combinar streams ya existentes. Una consulta continene uno o varios flujos de eventos, manipuladores para modificar aquellos flujos de entrada, y un flujo de salida en el que publicar los eventos generados.

from <input stream name> **select** <attribute name>, <attribute name>, ... **insert into** <output stream name>

Cada consulta de Siddhi puede consumir uno o varios flujos de eventos y crea un nuevo flujo de salida a partir de ellos.

Proyecciones en las consultas

Una consulta de Siddhi tiene tres partes claramente diferenciables en sus consulta, una entrada (**from**), una salida (**insert into**) y una sección de proyección (**select**). Siddhi soporta las siguientes proyecciones:

- **Selección de los atributos necesario**

Permite seleccionar aquellos atributos del esquema definido que necesitamos.

from inputStream **select** attributeA, attributeB **insert into** outputStream

En la consulta anterior estamos seleccionando del flujo de eventos "inputStream" los atributos "attributeA" y "attributeB" y los estamos insertando en el flujo de salida "outputStream".

- **Selección de todos los atributos**

Permite seleccionar todos los atributos del esquema definido.

```
from inputStream select * insert into outputStream
```

En la consulta anterior estamos seleccionando del flujo de eventos "inputStream" todos los atributos y los estamos insertando en el flujo de salida "outputStrem". En este caso práctico la entrada es igual a la salida.

- **Renombrado de atributos**

Permite renombra los atributos seleccionados del esquema definido.

```
from inputStream select attributeA as fieldA, attributeB as fieldB insert into outputStream
```

En la consulta anterior estamos seleccionando del flujo "inputStream" los atributos "attributeA" y "attributeB" y los estamos insertando en el flujo de salida "outputStream" como atributos "fieldA" y "fieldB".

- **Introducir los valores por defecto**

Permite introducir valores por defecto, este valor sobreescribe el del atributo si este existiera.

```
from inputStream select attributeA, 'valueB' as attributeB insert into outputStream
```

En la consulta anterior estamos seleccionando del flujo "inputStream" el atributo "attributeA" y estamos insertando el atributo "attributeB" con valor por defecto "valueB" en el flujo de salida "outputStream".

- **Uso de expresiones matemáticas y lógicas**

Siddhi permite el uso de expresiones matemáticas y lógicas en sus proyecciones. En la siguiente tabla se hace referencia a ellas:

Operador	Descripción	Ejemplo
()	Alcance	`(cost + tax) * 0.5`
IS NULL	Comprueba si un atributo es nulo	is null
NOT	Negación lógica	not (price > 10)
* / %	Multiplicación, división y módulo	temp * 9/5 + 32

Operador	Descripción	Ejemplo
+ -	Suma y resta	temp * 9/5 + 32
< ⇐ > ⇐	Comparaciones: menos que, mayor o igual que, mayor que, menor o igual que	totalCost >= price * quantity
== !=	Comparaciones: Igual, distinto	totalCost >= price * quantity
AND	Conjunción lógica (Y)	temp < 40 and (humidity < 40 or humidity >= 60)
OR	Disyunción lógica (O)	temp < 40 and (humidity < 40 or humidity >= 60)

from TempStream **select** roomNo, temp * 9/5 + 32 **as** temp, 'F' **as** scale, roomNo >= 100 and roomNo < 110 as isServerRoom **insert into** RoomTempStream;

Filtros

Siddhi tiene la capacidad de aplicar filtros a sus flujos de entrada permitiendo filtrar los eventos que se van recibiendo por el criterio que deseemos.

from <input stream name>[<filter condition>] **select** <attribute name>, <attribute name>, ... **insert into** <output stream name>

Las operaciones que pueden utilizarse para la creación de un filtro son:

- Operaciones de comparación menor que (<), menor o igual que (⇐), mayor que (>), mayor o igual que (>=), igual a (==) y distinto (**!**=).
- Operaciones lógicas de conjunción (**and**), disyunción (**or**) y negación (**not**)
- Operación de comprobación de contenido de cadena (**contains**)
- Operación de tipo de dato (**instanceof**)

from TempStream [(roomNo >= 100 and roomNo < 110) and temp > 40] **select** roomNo, temp **insert into** HighTempStream;

En el ejemplo anterior se está aplicando un filtro en el flujo de entrada **TempStream** en el cual se escifica que el número de habitación sea mayor o igual a 100 y menor a 110 y cuya temperatura sea de mayor de 40.

Ventanas

Siddhi dispone de funciones de ventanas que permiten recolectar y trabajar sobre un subconjunto limitado de eventos.

Los flujos de eventos de entradas sólo pueden tener una función de ventana asociada.

Las funciones de ventanas pueden emitir dos tipos eventos por cada evento que consumen: Los eventos actuales y los eventos que han expirado. La función de ventana emite los eventos actuales (**current-event**) cuando llega un nuevo evento y emite los eventos expirados (**expired-event**) cuando un evento expira debido al criterio de la ventana.

La salida de las funciones de ventana puede ser manipulada en base a la clasificación de los eventos.

> **from** <input stream name>[<filter condition>]#window.<windowFunction> **select** <attribute name>, <attribute name>, ... **insert** [***expired events*** | ***current events*** | ***all events***] **into** <output stream name>

Según la clasificación de los eventos:

- **expired events**: La ventana emitirá los eventos que hayan expirado.
- **current events**: La ventana emitirá los eventos cada vez que se reciba uno.
- **all events**: La ventana emitirá tanto los eventos actuales como los que han expirado.

Agregaciones

Siddhi dispone de una serie de funciones de agregación para llevar a cabo cálculos de los valores de los atributos indicados dentro de la ventana definida. Para ello los atributos tienen que ser de tipo `int`, `long`, `double` o `float`. Estas agregaciones son:

- sum: Calcula la suma ⟶ sum(<my-attribute>) as mySum
- average: Calcula la media ⟶ avg(<my-attribute>) as myAvg
- max: Obtiene el máximo valor ⟶ max(<my-attribute>) as myMax
- min: Obtiene el mínimo valor ⟶ min(<my-attribute>) as myMin
- count: Cuenta el número de eventos recibidos ⟶ count() as events
- stddev: Calcula la desviación estandar ⟶ stddev(<my-attribute>) as myStddev

Nótese que a las funciones de agregación se le aplica un renombrado, esto se debe a que siddhi no reconoce las funciones de agregación como atributos del flujo de eventos con el que se está trabajando.

Si no se aplicaran funciones de agregación entonces obtendríamos una salida igual a la entrada del

flujo, variando únicamente en el tiempo de emisión de los eventos que depende del tipo de ventana utilzada.

Tipos de ventanas

Siddhi ofrece las siguientes funciones de ventana:

- **Ventana de tiempo**: <event> time(<int|long|time> windowTime)

Ventana deslizante que mantiene los eventos que se han recibido durante el último periodo de tiempo "**windowTime**".

Ejemplo: Calcular la suma del valor del atributo "**attrA**" de los eventos que estén dentro de la ventana de 15 segundos.

```
from inputStream#window.time(15 sec) select sum(attrA) insert into outputStream
```

- **Ventana de tiempo por lotes**: <event> timeBatch(<int|long|time> windowTime)

Ventana que procesa los eventos por lotes. Recolecta los eventos recibidos dentro del último periodo "**windowTime**" y los agrupa en un único lote.

Ejemplo: Calcular la suma del valor del atributo "**attrA**" de los eventos recolectados cada 15 segundos.

```
from inputStream#window.timeBatch(15 sec) select sum(attrA) insert into outputStream
```

- **Ventana de longitud**: <event> length(<int> windowLength)

Ventana deslizante que mantiene los últimos elementos recibidos en una ventana de tamaño "**windowLength**".

Ejemplo: Calcular la suma del valor del atributo "**attrA**" de los 5 últimos evetnos recibidos.

```
from inputStream#window.length(5) select sum(attrA) insert into outputStream
```

- **Ventana de longitud por lotes**: <event> lengthBatch(<int> windowLength)

Ventana deslizante que emite los eventos como un lote a la llegada del i-ésimo evento en una ventana de tamaño "**windowLength**".

Ejemplo: Calcular la suma del valor del atributo "**attrA**" cada 5 eventos recolectados.

```
from inputStream#window.lengthBatch(5) select sum(attrA) insert into outputStream
```

- **Ventana deslizante de tiempo externo**: <event> time(<long> timestmap, <int|long|time>

windowTime)

Ventana deslizante que trabaja con el tiempo proporcionado por el flujo de eventos de entrada en lugar de utilizar el de la máquina.

Ejemplo: Calcular la suma del valor del atributo "attrA" utilizando una ventana de tiempo de 15 segundos y cuya marca de tiempo vendrá determinada por el campo "timestamp" del flujo de eventos entrante que está procesando.

```
from inputStream#window.externalTime(timestamp, 15 sec) select sum(attrA) insert into outputStream
```

- **Ventana deslizante con cron**: <event> cron(<string> cronExpression)

Ventana deslizante que procesa los eventos en base a un patrón de tiempo basado en una expresión de cron.

Ejemplo: Calcular la suma del valor del atributo "attrA" cada 5 segundos.

```
from inputStream#window.cron('*/5 * * * * ?') select sum(attrA) insert into outputStream
```

- **Ventana deslizante de primeras únicas ocurrencias**: <event> firstUnique(<string> attribute)

Ventana que mantiene únicamente los primeros eventos que son únicos en base a un atributo especificado.

Ejemplo: Obtener el primer evento de todas las IPs.

```
from inputStream#window.firstUnique(ip) select * insert into outputStream
```

- **Ventana deslizante de últimas ocurrencias únicas**: <event> unique(<string> attribute)

Ventana que mantiene los últimos eventos que son únicos en base a un atributo especificado.

Ejemplo: Obtener el último evento único de todas las IPs.

```
from inputStream#window.unique(ip) select * insert into outputStream
```

- **Ventana deslizante de ordenamiento**: <event> sort(<int> windowLength, <string> attribute, <string> order, .. , <string> attributeN, <string> orderN)

Ventana que ordena los atributos especificados de forma ascendete o descendente.

Ejemplo: Mantener 5 eventos ordenados por precio de forma ascendente.

```
from inputStream#window.sort(5, price, asc) select * insert into outputStream
```

- **Ventana deslizante de frecuencia**: <event> frequent(<int> eventCount, <string> attribute, .. , <string> attributeN)

Ventana que devuelve los últimos eventos más frecuentes recibidos. El cálculo de la frecuencia se basa en el algoritmo de contenio Misra-Gries.

Ejemplo: Obtener los 3 eventos más frecuentes.

```
from inputStream#window.frequent(3) select * insert into outputStream
```

- **Ventana deslizante de frecuencia con pérdidas**: <event> lossyFrequent(<double> supportThreshold, <double> errorBound)

Ventana que identificará y devolverá todos los eventos cuya frecuencia exceda el valor "**supportThreshold**". El cálculo de la frecuencia se basa en el algoritmo de *Lossy Counting*

```
from inputStream#window.lossyFrequent(0.1, 0.01) select * insert into outputStream
```