BarCodeApp Application Tutorial

Kurento Media Server + Proton CEP Integration

This document provides a description on how to develop an application integrating Proton Complex Event Processing (CEP) and Kurento. For a brief description of the example, is used a Kurento filter that detects ZBar codes and trigger events. This events are given to the CEP, which process them and return feedback for the end user.

Architecture

The architecture of the example uses a modular design and is based on Kurento and CEP architectures. The following Figure 1 illustrates in detail the modules used and the communication channels between them.

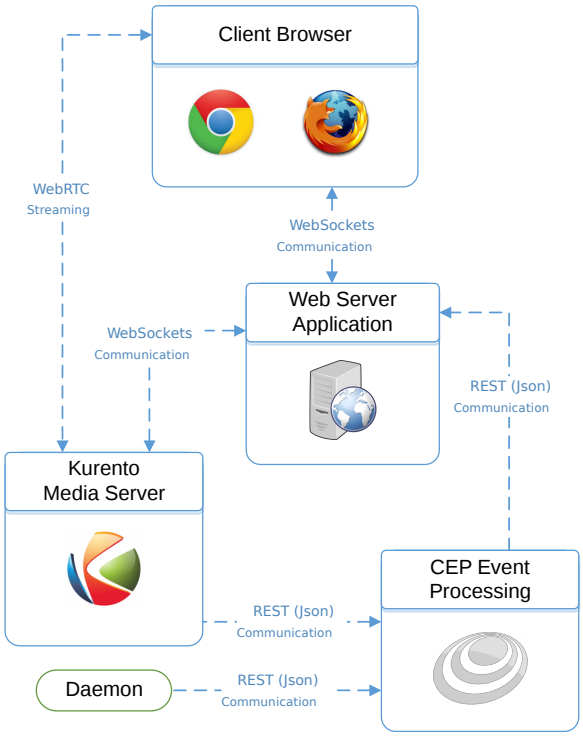

Figure 1: Architecture of Kurento and CEP example

The architecture is compose by the following modules:

- **Client Browser:** this module represents the end user browsers, which are capable to support WebRTC communications.

- **Web Server Application (WSA):** this module represents a web server that implements a sample application of Kurento. It can communicate using websockets and rest interfaces.

- **Kurento Media Server (KMS):** this module represents the Kurento Media Server that is responsible for web real-time communication.

- **CEP Event Processing:** this module represents the Complex Event Processing (CEP) that is responsible for processing events originated by the web sample application.

- **Daemon:** this module represents a Unix Daemon that is responsible for processing events originated in files logs of Kurento Media Server.

At the start of the application, the client browser connects to the WSA, using for communication web sockets and trading messages in json format. This steps allows the establishment of a communication channel between the client and the server.

When the client wants to begin a real-time communication, the WSA is used as an intermediary to create a communication channel with KMS. After the channel is set up, the media streaming communication, between the client and the media server, is based in the WebRTC schema.

At the same time that the client is performing a stream communication with the media server, a demon present in the KMS or the WSA gather information and send it to CEP, using a Representational State Transfer (REST) interface and messages in json format.The CEP then process the information that was receive and provide feedback to the client, using the WSA as an intermediary.

Generated Documentation

This application provides an example to how to integrate the Kurento with CEP. The main goal is to use the real-time communication samples, from the Kurento, to create events, which can be processed by the CEP, generating some feedback for the end user. Given that Kurento was a filter that detects and reads ZBar codes, it is possible to develop an application that allow the user to show a code bar that will trigger an event, similar to the cash registration machines of the supermarkets.

The example is a ZBar code analyser that from the values given, in the format of bar codes, presents results to the end user. The end user shows barcodes to the

Webcam with values, which are identified and decoded by the Kurento. This values are then used by the CEP that, for a time window, calculates the number of view, sum and average of the values. The results are then send to the end user as feedback.

This is a basic example showing how is possible to use image filters, provided from Kurento, to analyses and then process events, created from image collection. In the future, with more different filters it is possible to develop more advance analyze mechanism that are capable of processing other type of data, identification and processing events generated from car plates or face recognition.

The installation and the configuration detailed in this section are for a deployment in local machines in your own premises and could be easily adapted for a deployment in the FIWARE Cloud.

Install Kurento

To install and start the Kurento Media Server follow the installation procedures described in Kurento Installation and Administration Guide (StreamOriented - Installation and Administration Guide). The Web Server Application sample can be download and run from GitHub using the following command:

```
git clone https://github.com/htfonseca/kurento-tutorial-java.git
cd kurento-tutorial-java/kurento-magic-mirror-ZBar
mvn compile exec:java
```

The sample will stars on port 8080 in the localhost by default. Therefore, to run the client the URL to use is http://localhost:8080/ in a WebRTC compliant browser (Chrome, Firefox). The REST interface will be receiving information in the URL http://localhost:8080/message. The configuration of the Web Server Application sample can be change using the following command:

```
mvn compile exec:java -Dserver.port=<custom-port>
```

Install Proton CEP

To install and start the Proton CEP engine (CEP GE) follow the installation procedures described in CEP Installation and Administration Guide (CEP GE - IBM Proactive Technology Online Installation and Administration Guide).

The default Proton CEP instance is called *ProtonOnWebServer*, and the server is listening by default on port 8080 (the default Apache Tomcat server port). If you change the Tomcat port you also need to change the port in the `tomcat-server-`

`port={port}` configuration option on the file `{Apache Tomcat directory}/webapps/ProtonOnWebServerAdmin/ProtonAdmin.properties`, as described in the CEP Installation and Administration Guide.

You can check that CEP started accessing the CEP Web Development User Interface (CEP Web UI) at: http://{host}:{port}/AuthoringTool/Main.html in your browser. The host is the IP or hostname of the machine were the the CEP was deployed, The default port is 8080, unless it was changed according to the instruction detailed above.

Import a CEP Project with the configurations needed for the application. A Project is a collection of definitions for the incoming events, event processing agents, contexts, event consumers and event producers.

In the CEP Web UI click in the "*Import Project...*" button and select the file `BarCodeApp_CEPProject.json` located in BarCodeApp repository. The CEP Web UI should now list the different project components like shown in Figure 2.

Install Kurento Log Parser

The parsing and processing of the KMS logs is done using Parsible. This Pyhton log parser was extended via plugins to parse the KMS logs. The Parsible application including the KMS plugins can be found in the `parsible` folder inside the BarCodeApp project folder.
To configure the Parsible KMS plugin you need to change the CEP REST URL, to point to the host, port and CEP instance described in the above section. From the root directory of Parsible, edit the file `./parsible/plugins/outputs/cep_publisher.py`. And change the variables shown below according to your CEP configuration:

```
CEP_HOSTNAME = "10.0.1.100"
CEP_PORT = "8080"
CEP_INSTANCE_NAME = "ProtonOnWebServer"
```

After having configured the Parsible KMS plugin you can start it using by executing the command bellow from the Parsible root directory (you may need to change path for the KMS log file according the configuration of your system):

```
python parsible.py --log-file /var/log/kurento-media-server/media-server.log  --parser parse_kurento
```

While Parsible is running it monitors KMS log file, whenever a new line, that matches the regular expression defined in the parser, comes in, the tool will publish an event to the CEP REST input interface.

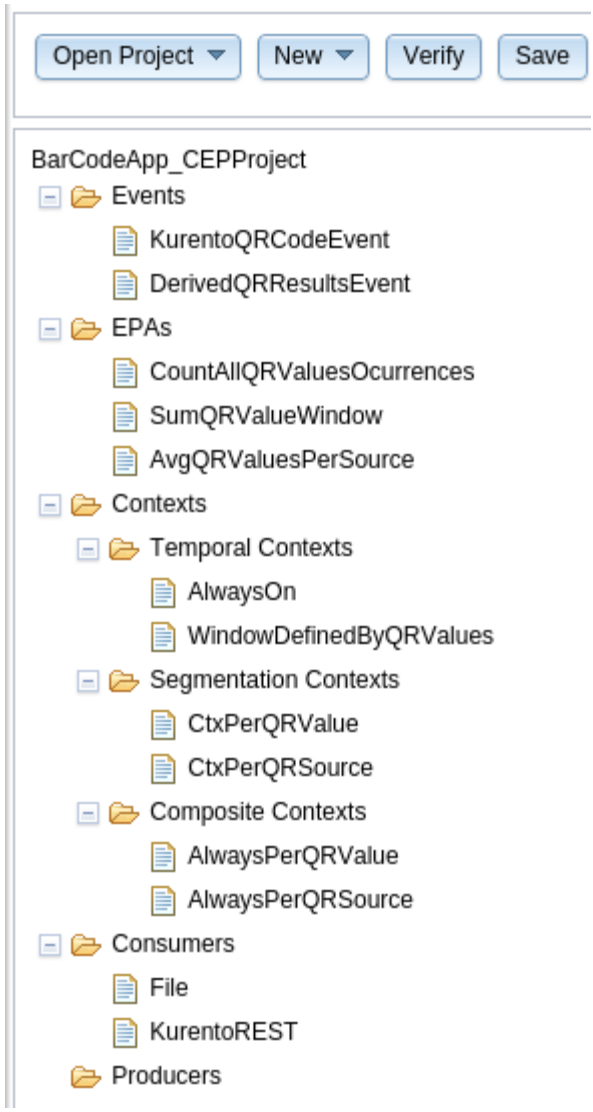

Figure 2: CEP Web UI after importing the Project file

Kurento Web Server Application Sample Project

This section contains the tutorial of the sample code of the Web Server Application. The sample was made in Java and is based in the following tutorial “WebRTC magic mirror” (http://www.kurento.org/docs/current/tutorials/java/tutorial-2-magicmirror.html). Therefore, this section is only going to cover the use of ZBar Code filter and the handling of the events it creates.

The filter perform media processing, computer vision, augmented reality, and so on. The ZBarFilter detects QR and bar codes in a video feed. When a code is found, the filter raises an event. Clients can add a listener to this event using the following method:

```
// WebRtcEndpoint is a media element with sen/receive capabilities for
// WebRTC media strems.
WebRtcEndpoint webRtcEndpoint = new
WebRtcEndpoint.Builder(pipeline).build();

// The filter used for detecting Zbar codes
ZBarFilter zBarFilter = new ZBarFilter.Builder(pipeline).build();

// When is detect a code this event is raise.
zBarFilter.addCodeFoundListener(new EventListener<CodeFoundEvent>() {

// This method is call is the event is raise.
@Override
public void onEvent(CodeFoundEvent event) {
    log.info("Code Found " + event.getValue());
    // ...
    });
}

//Connect the WebRtcEndpoint to the filter
webRtcEndpoint.connect(zBarFilter);
zBarFilter.connect(webRtcEndpoint);
```

How to create the Proton CEP Project

In this section we describe how the CEP Project (downloaded and imported in section Installing Proton CEP) was created as well as the purpose of each one of the project components. This will allow for a better understanding of the CEP configuration. After completing the steps below you will obtain a CEP project similar to the one you downloaded in the section Installing Proton CEP. The projects are created using Proton CEP Web UI, to start creating a new project go to the URL: http://{host}:{port}/AuthoringTool/Main.html in your browser and follow the steps below:

Create a New Empty project:

1. Click New, choose *Project*

2. Enter a *Project* name, we are going to call our project **`BarCodeApp_CEPProject`**.
3. Click *Add*.

A new project will be displayed in the navigator area with the new name and all the required folders.
After creating a new project we are going to define CEP input and output events.

Add New Events:

Create a new event that represents the events received from the KMS log parser, through CEP REST interface.

1. Click *New*, choose *New Event*.
2. Enter an event name, we are going to call the events received from Kurento **`KurentoQRCodeEvent`**. NOTE: the event name must match the value for the attribute *Name* in the JSON event received from KMS.
3. By default new event is created with several default attributes: *Certainty*, *OccurenceTime*, *ExpirationTime*, *Cost* and *Duration*. As we are not going to use this attributes, delete them by clicking in the red X at the end of each row in the *Event Attributes Table*.
4. Now we are going to add new event attributes to **`KurentoQRCodeEvent`** event. The attributes to add for the input events must include all attributes present in the JSON event received from KMS, except the attribute *Name* (this already corresponds to the event name). In our example we are going to add the attributes: **`type`**, **`source`**, **`value`** and **`id`**. These are the attributes included in the JSON events received from Kurento and **`id`** is a new attribute used to identify these type of events in the Kurento Web Application. To add a new attribute click *Add New* in the *Event Attribute Table*. When clicking, a new row with empty fields will be added to the table.
5. Fill in the *Name* column with the event attribute name, you are going to add an attribute *`source`* with *Type String*. Repeat the process, and add the attributes *`value`* and *`type`* attributes, all of *Type String*.

We are also going to create another event, a *derived event*, this event is similar to input events but represent the event that is created by the processing agent when it detects a situation. This derived event is the one CEP will output, and send to the Kurento Web application with the outcome produced by the processing agents. The procedure to create the derived event is similar to the procedure described for the **`KurentoQRCodeEvent`** input event. But this one will be named **`DerivedQRResultsEvent`**. Delete all the event attributes created by default by Proton CEP and add the following new attributes: **`description`** (*Type String*), **`value`** (*Type Double*), **`source`** (*Type String*) and **`id`** (*Type String*) with *Default Value* **`"rest"`**.

Add Segmentation Contexts:

We are now going to create two *Segmentation Contexts*. These contexts group events that refer to the same entity, according to a set of attributes.
To create a *Segmentation Context* follow the steps below:

1. Click *New*, choose *Segmentation Context*.
2. Enter the *Segmentation Context* name as **`CtxPerQRValue`**.
3. In the *Participants Events* table Click *Add New*, a new row will be added to the table. In the *Event* column choose the **`KurentoQRCodeEvent`** and **`KurentoQRCodeEvent.value`** as the *Expression*. This context will group events by **`KurentoQRCodeEvent`** *value* attribute.
4. Repeat the steps above and add a new *Segmentation Context* named **`CtxPerQRSource`**. Add ***`KurentoQRCodeEvent`*** as *Participant Event* and *Expression* as **`KurentoQRCodeEvent.source`**. This context will group events by **`KurentoQRCodeEvent`** *source* attribute.

Add Temporal Contexts:

We are going to create two *Temporal Contexts*, these contexts define a time window in which the event-processing agent is relevant. They start with an *initiator* and end with a *terminator*. The first *Temporal Context* will be a context that starts when Proton CEP starts and only terminates with the CEP termination.
To create a *Temporal Context* as defined above, follow the steps below:

1. Click *New*, choose *Temporal Context*.
2. Enter the *Temporal Context* name as **`AlwaysOn`**.
3. In the *Initiators* section select the *"At Startup"* checkbox and in the *Terminators* section select the *"Never Ends"* checkbox.

The second *Temporal Context* is a time window that will start for a configured **`KurentoQRCodeEvent`** *value* and terminate for another **`KurentoQRCodeEvent`** *value we are going to define*.

To create the Context as defined above, follow the steps below:

1. Click *New*, choose *Temporal Context*.
2. Enter the *Temporal Context* name as **`WindowDefinedByQRValues`**.
3. In the *Event Initiators* table Click Add *New*. In the added row select **`KurentoQRCodeEvent`** as the initiator event, **`KurentoQRCodeEvent.value=="start"`** as the C*ondition* and ***`ignore`*** as the *Correlation Policy*.
4. In the *Event Terminators* table Click Add *New*. In the added row select **`KurentoQRCodeEvent`** as the *terminator* event, **`KurentoQRCodeEvent.value=="stop"`** as the *condition* and **`First`** as the *Quantifier* and **`Terminate`** as *Termination Type*.

Add Composite Contexts:

A *Composite Context* groups several other contexts. We are are going to define two *Composite Contexts*. One that groups the **`AlwaysOn`** *Temporal Context* with the **`CtxPerQRValue`** *Segmentation Context* to group the events by QR Code value during CEP execution, the **`AlwaysPerQRValue`** context. Another that groups the **`AlwaysOn`** *Temporal Context* with the **`CtxPerQRSource`** *Segmentation Context* to group the events per *source* while CEP is running, the **`AlwaysPerQRSource`** context.
To create a *Composite Context*, follow the steps below:

1. Click *New*, choose *Composite Context*.
2. Enter the *Composite Context* name as **`AlwaysPerQRValue`**.
3. Add a new *Temporal Context* and choose **`AlwaysOn`**, add a new *Segmentation Context* and select **`CtxPerQRValue`**.
4. Repeat the process from steps 1 to 3 to add a new *Composite Context* named **`AlwaysPerQRSource`** with **`AlwaysOn`** as *Temporal Context* and **`CtxPerQRSource`** as *Segmentation Context*.

Add EPAs (Event Processing Agents):

These agents detect predefined situations and according to the configured rules generate derived events. We are going to define three EPAs:

1. **`CountAllQRValuesOcurrences`**: this EPA is will be configured to count the number of times a determined QR code event value enters the CEP engine. It will generate a derived event containing the number of times that value was detected since Proton CEP started.
2. **`SumQRValueWindow`**: this EPA will be configured to sum the QR code event values during a time window. When the time window ends a derived event with the sum of the QR codes event values, detected while the time window as active, will be generated. The time window will start when a QR code event with "`start`" value is detected and will terminate when a QR code event with a "`stop`" value is detected.
3. **`AvgQRValuesPerSource`**: this EPA will be configured to generate a derived event with the average event value per event source. The event source corresponds to the *WebRTC* session. Every time a QR code is detected an event with the average QR code value for that Session will be generated.

The description on how to create each one of the above EPAs is given below:

`CountAllQRValuesOcurrences`

1. Click *New*, choose *EPA*.
2. Enter EPA name as **`CountAllQRValuesOcurrences`**.
3. In the *General Section*, select *EPA Type* as **`Aggregate`** and *Composite Context* as **`AlwaysPerQRValue`**. This will aggregate the received events per QR Code value while CEP is running.

4. Add a new *Participant Event* in the *Event Selection Section*. Choose `KurentoQRCodeEvent` as *Event*. Insert `IsDigits(KurentoQRCodeEvent.value)` for the *Condition,* this will filter events with values that are not numbers. Add a new *Computed Variable* with `valuecounter` as *Name*, `Count` as *Aggregation Type* and `1` as *Expression*.
5. In the *Condition Section* add a new *Segmentation Context* and choose `CtxPerQRValue`.
6. In the *Derivation Section* add a new *Derived Event* by clicking the Add button after selecting `DerivedQRResultsEvent` in the dropdown menu.
7. In the Event Attributes table for `DerivedQRResultsEvent`, fill the attributes expressions fields as described: *description* - insert as expression: `"Code: " ++ context.CtxPerQRValue ++ " count"`, *value* - insert as expression: `valuecounter` and leave the *source* expression empty.

`SumQRValueWindow`

1. Click *New*, choose *EPA.*
2. Enter *EPA* name as `SumQRValueWindow`.
3. In the *General Section*, select *EPA Type* as `Aggregate` and *Temporal Context* as `WindowDefinedByQRValues`. This will aggregate the received events while the defined window is active.
4. Add a new *Participant Event* in the *Event Selection* Section. Choose `KurentoQRCodeEvent` as *Event*. Insert `IsDigits(KurentoQRCodeEvent.value)` for the *Condition*, this is needed to filter events with values that are not digits. Add a new *Computed Variable* with `sumvalue` as Name, `Sum` as *Aggregation Type* and `StringToInt(KurentoQRCodeEvent.value)` as *Expression*.
5. In the *Condition Section* change the *Evaluation Policy* from `Immediate` to `Deferred`, this way the derived event is only generated at the end of the *Temporal Context*, in this case, when the time window closes.
6. In the *Derivation Section* add a new *Derived Event* by clicking the *Add* button after selecting `DerivedQRResultsEvent` in the dropdown menu.
7. In the *Event Attributes* table for the `DerivedQRResultsEvent`, fill the attributes expressions fields as described: *description* - insert as expression: `"Value Sum in window (From Tag: start to Tag: stop)"`, *value* - insert as expression: `sumvalue` and leave the *source* expression empty.

`AvgQRValuesPerSource`

1. Click *New*, choose *EPA*.
2. Enter *EPA* name as `AvgQRValuesPerSource`.
3. In the General Section, select *EPA Type* as `Aggregate` and *Composite Context* as `AlwaysPerQRSource`. This will aggregate the received events by event attribute source while CEP is running.

4. Add a new *Participant Event* in the *Event Selection* section. Choose `KurentoQRCodeEvent` as *Event*. Insert `IsDigits(KurentoQRCodeEvent.value)` for the *Condition*, this is to filter events with values that are not numbers. Add a new *Computed Variable* with `avgvalue` as *Name*, `Average` as *Aggregation Type* and `StringToInt(KurentoQRCodeEvent.value)` as *Expression*.
5. In the *Derivation Section* add a new Derived Event by clicking the *Add* button after selecting `DerivedQRResultsEvent` in the drop-down menu.
6. In the *Event Attributes* table for the `DerivedQRResultsEvent`, fill the attributes expressions fields as described: *description* - insert as expression: `"Value Average in this Session"`, *value* - insert as expression: `avgvalue` and *source* - insert as expression: `context.CtxPerQRSource` (this way the *source* value will corresponde to the value of the `CtxPerQRSource` context, i.e, the Session ID of the context). Figure 3 shows how the Derived Event should look at the end of step 6.

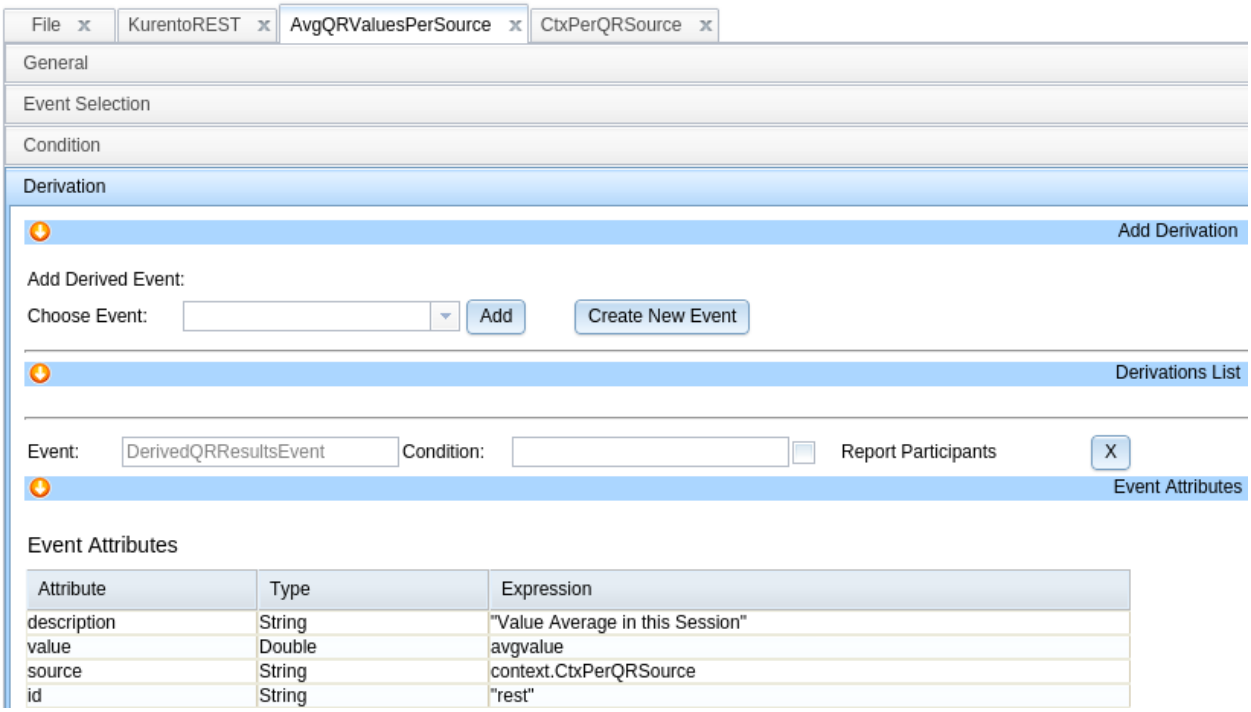

Figure 3: AvgQRValuesPerSource EPA Derived Event

Add Consumers:

A consumer is the Proton CEP building block responsible for consuming the events generated by the CEP engine and sending them to the outside world. In this application the derived events will be published using the CEP REST interface to the Kurento Web Application. We will show how to create a *File* consumer that will also

write the Derived events to a text file. This will work as a log with the derived events.

To create a *REST Consumer*, follow the steps below:

1. Click New, choose *Consumer*.
2. Enter *Consumer* name as **`KurentoREST`**.
3. Select **`Rest`** as *Type*.
4. Change the consumer properties as described, *URL* value: **`http://{KURENTO_APP_HOST}:{KURENTO_APP_PORT}/message`**, *contentType* value: **`application/json`**, *formatter* value: **`json`**. There are other *Properties* that were added by default by CEP you don't need to change them.
5. Add new *Received Events* and select **`DerivedQRResultsEvent`**.

For the *File Consumer*, follow the steps below:

1. Click New, choose *Consumer*.
2. Enter Consumer name as **`File`**.
3. Select **`File`** as *Type.*
4. Change the consumer properties as described: *filename* **-** insert the filename where to save the output, the path is relative to the Apache Tomcat root directory. (For example, in a Linux host, to save to the file named *`derived_log.txt`* in the directory *`samples`* inside Tomcat root directory the path will be `./sample/derived_log.txt`), *formatter* value - **`json`**.
5. Add new *Received Events* and select **`DerivedQRResultsEvent`**.

Running the BarCodeApp

After successfully started all the application components and configuring them as explained in the previous sections, the application is ready to run. To run the application you need to enter the BarCode Web App *URL/IP*:*Port* in your browser. You will be presented with a page containing a section like the one shown in Figure 4. You must use a web browser that supports WebRTC (like the latest versions of Chrome 23+ or Firefox 22+).

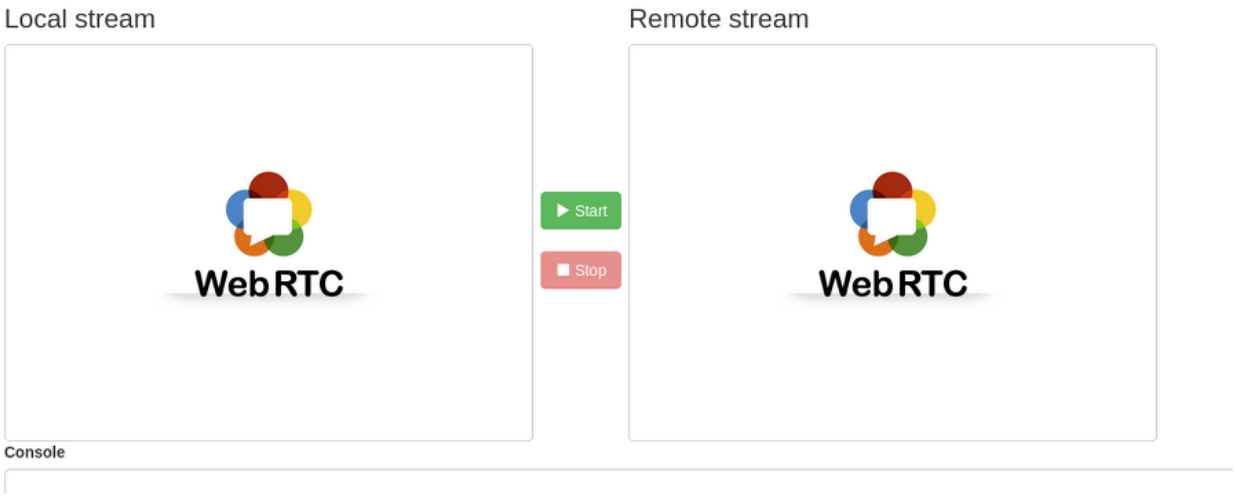

Figure 4: Bar Code WEB App start page

To start the application you click on the star button, you may need to grant permission for the browser to access the camera and microphone. When the application starts you will have something similar to the screen capture shown in Figure 5. At the left side it is displayed what the web camera is capturing, at the right side of the page is is displayed the webcam capture after passing through a ZBarCode filter. This filter is only detecting barcodes in the video stream so the video displayed will be similar to the one at your left. At the bottom of the page you will find a console that displays information related with the barcode detection and event processing messages generated by the Proton CEP.

Figure 5: Bar Code WEB App start page

To test the application you need to show a barcode, like a QR code, as shown in Figure 5. Inside of the `docs` folder of the BarCodeApp project you can find a page with several QR codes that you can print to test the application. Alternatively you can take photo of each code on your phone and then show the screen displaying the code in front of the webcam.

As previously described, CEP was configured with three different Event Processing Agents (EPA), each one of the detects a different situation. We will give different examples and show the outcome of each one, i.e., the expected output shown in the console for different QR code sequences.

The Figure 6 show a QR code sequence containing the start and stop encoded values. According to the **`SumQRValueWindow`** EPA configuration, the start value will trigger a time window that will allow to sum all the QR code values until a stop QR code value is detected. The expected output, shown in the console, for the **`SumQRValueWindow`** EPA when the sequence of QR codes shown is detected in the webcam, one by one, will be:

```
Received message: {"id":"rest","description":"Value Sum in window
(From Tag: start to Tag: stop)","value":"315.0"}
```

It should be noted that when a QR code is detected it can trigger an action (creation of a derived event) from more than one EPA at the same time. So other events from CEP can be shown in the console.

Figure 6: Example QR code sequence with start and stop codes

Assuming that the application was started and the first QR codes to be detected were the codes shown in Figure 7 the **`CountAllQRValuesOcurrences`** EPAs will send events (displayed in the console) with the total number of repetitions for each code. For example, when the second code with value 200 is detected a message like the one below is displayed informing that the code 200 had 2 occurrences since the start of the CEP.

Figure 7: Example QR code sequence

```
Received message: {"id":"rest","description":"Code: 200
count","value":"2.0"}
```

At the same time the **`AvgQRValuesPerSource`** EPA will generate events with the average of the QR code values during each session, assuming that all codes were shown in the same session (the session starts when clicking in the play button and ends when clicking the stop button) when the 5th code is detects the console show the message below:

```
Received message: {"id":"rest","description":"Value Average in this
Session","value":"123.0"}
```

The configured rules are just an example of what can be done combining the Kurento GE and Proton CEP GE. Much more interesting rules can be configured in the CEP to make the application more useful in a real world application. For example, adapting a camera to a conveyor belt in automated distribution and warehousing, were all the items tagged with QR codes, and with an adapted CEP Project could be used to do automatic stock management. the CEP allows manage the logic of the application directly from a Web UI without further programming.