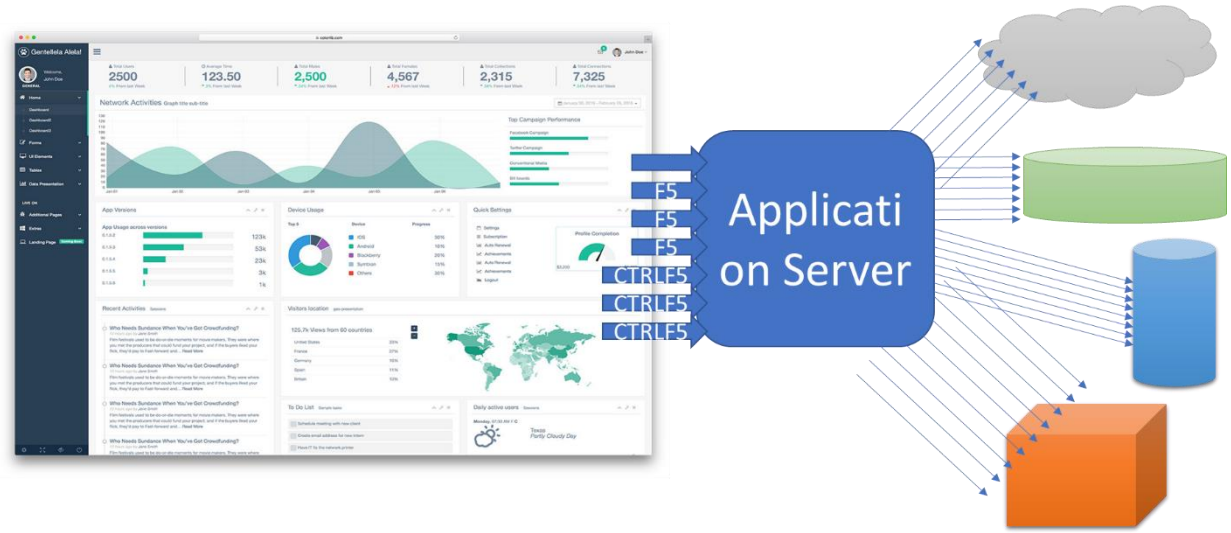
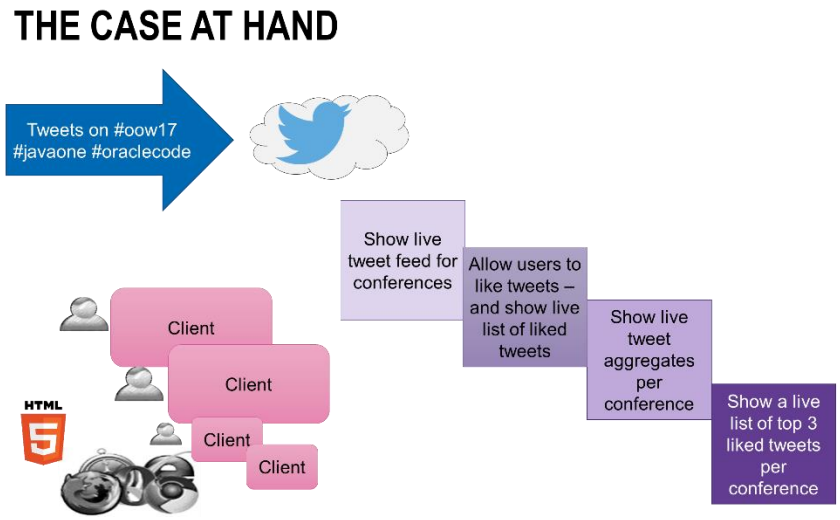
Real Time UI with Apache Kafka Streaming Analytics of Fast Data and Server Push



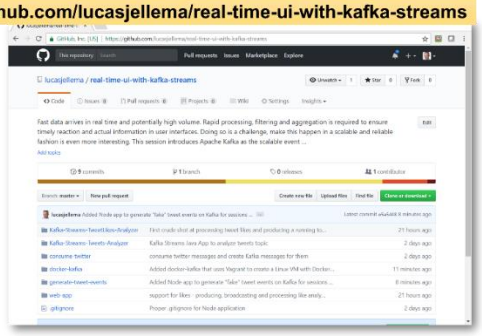
Fast data and active UI

- Handle influx
- Publish findings instantaneously
- Update UI & notify end user immediately
- Analyze in real time
- Decoupled components
- No data loss when a component is temporarily down
- Scalable with volume of events and of number of clients

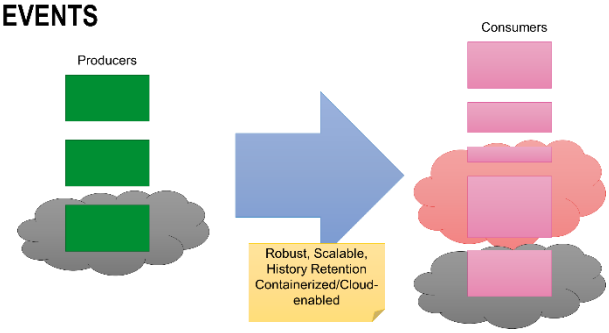


THE CODE

https://github.com/lucasjellema/real-time-ui-with-kafka-streams



EVENTS



Requirements for Event capability

- Provide decoupling between publisher and consumer
- Generally accessible for all consumers
    - Using standardized protocols and formats for communications and event payload (http, JSON)
- Scalable (handle high loads)
- Available (allow speedy event publication)
- Reliable (do not lose events, at least once delivery)
- Event Ordering (deliver events in the order of publication)
- Manageable at scale
- Retain Event History
    - For consumers that are late to the game
    - To construct state from all historic events: Event Sourcing

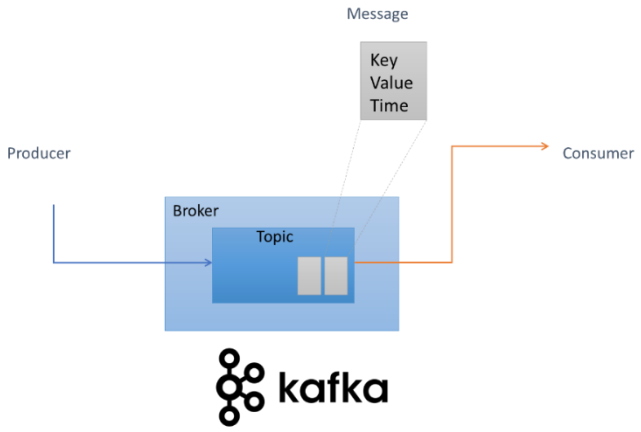
Introducing Apache Kafka

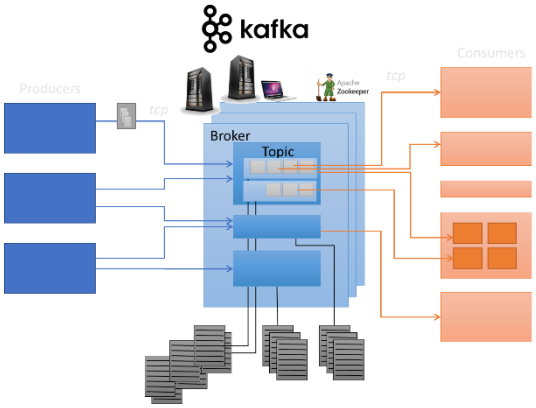
- ..- 2010 – creation at Linkedin
- Message Bus | Event Broker
- High volume, low latency, highly reliable, cross technology
    - Scalable, distributed, strict message ordering, ….
- 2011/2012 – open source under the Apache Incubator/ Top Project
- Kafka is used by many large corporations:
    - Walmart, Cisco, Netflix, PayPal, LinkedIn, eBay, Spotify, Uber, Sift Science
    - And embraced by many software vendors & cloud providers

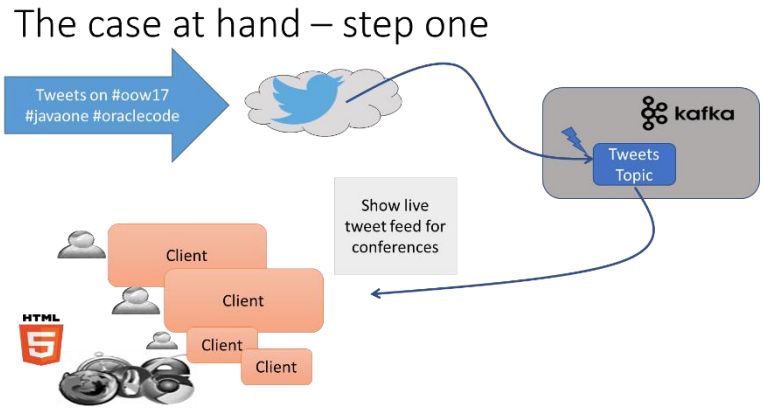
- Client libraries available for NodeJS, Java, C++, Python, Ruby, PHP and many more

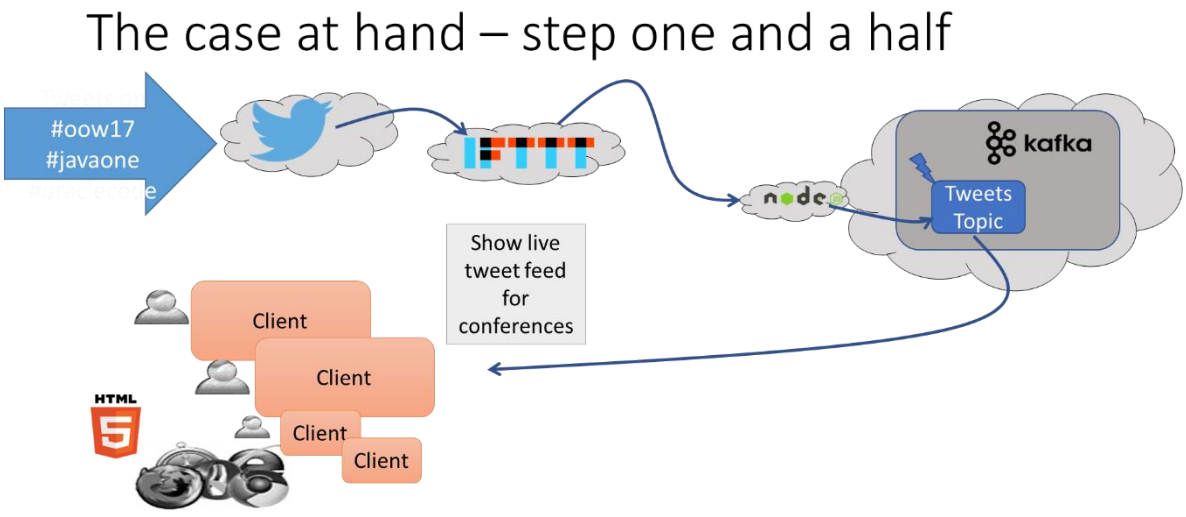
Kafka terminology

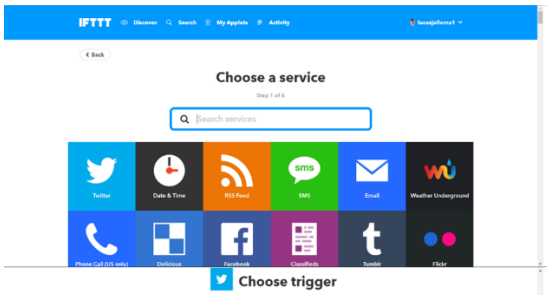
- Topic
  - partition
- Message
  - == ByteArray
- Broker
  - replicated
- Producer
- Consumer
  - Working together in Consumer Groups

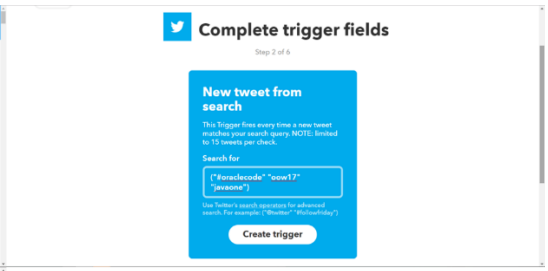


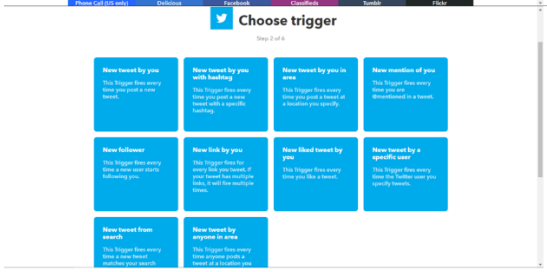


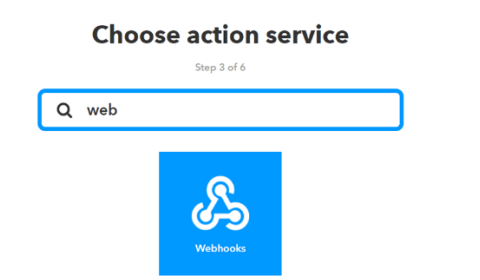












IFTTT consumer => KAFKA Producer

```
var eventBridge = module.exports;
var apiURL = "/eventBridge";
var moduleName = "accs.eventBridge";
var version = "0.8.1";

eventBridge.registerListeners =
    function (app) {
        app.post(apiURL + '/ifttt-tweet', function (req, res) {
            console.log('EventBridge IFTTT-TWEET POST');
            console.log('body in request' + JSON.stringify(req.body));
            eventBridge.postNewTweet(req, res, req.body);
        });//post messages
    }//registerListeners
eventBridge.postNewTweet = function (req, res, tweet) {
    produceTweetMessage(tweet);
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write("Thank you for your tweet");
        res.write("incoming headers" + JSON.stringify(req.headers));
    res.end();
}
```

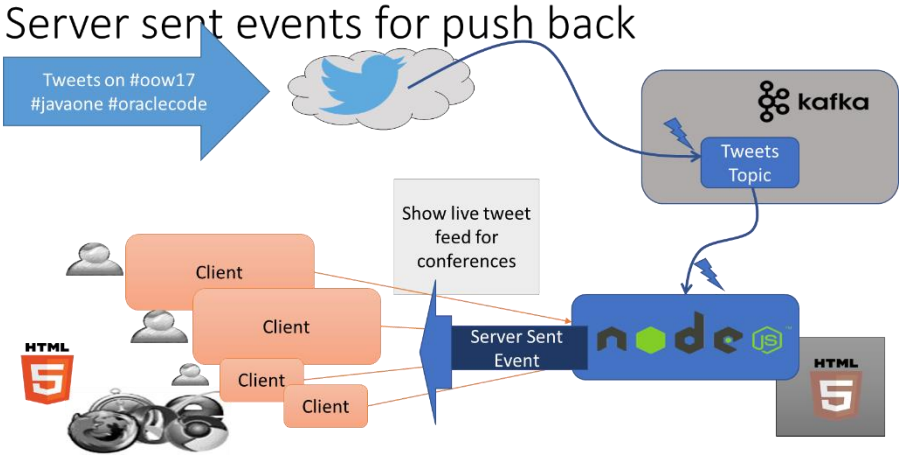
```
var EVENT_HUB_PUBLIC_IP = '129.144.150.24';
var TOPIC_NAME = 'partnercloud17-microEventBus';
var ZOOKEEPER_PORT = 2181;

var kafka = require('kafka-node');
var Producer = kafka.Producer;
var client = new kafka.Client(EVENT_HUB_PUBLIC_IP + ':'+ZOOKEEPER_PORT);
var producer = new Producer(client);

let payloads = [
    { topic: TOPIC_NAME, messages: '*', partition: 0 }
  ];

producer.on('ready', function () {
    console.log("producer  is ready");
});

function produceTweetMessage(tweet) {
    var tweetEvent = {
        "eventType": "tweetEvent"
        , "tweet": tweet
      };
      KeyedMessage  = kafka.KeyedMessage,
      tweetKM = new KeyedMessage(tweetEvent.id, JSON.stringify(tweetEvent)),
      payloads[0].messages = tweetKM;

    producer.send(payloads, function (err, data) {
```

Kafka consumer in node
get events pushed into application

```
var kafka = require('kafka-node');
var tweetListener = module.exports;
var subscribers = [];

tweetListener.subscribeToTweets = function (callback) {
  subscribers.push(callback);
}
var KAFKA_ZK_SERVER_PORT = 2181;
var EVENT_HUB_PUBLIC_IP = '192.168.188.102';
var TOPIC_NAME = 'tweetsTopic';

var consumerOptions = {
  host: EVENT_HUB_PUBLIC_IP + ':' + KAFKA_ZK_SERVER_PORT,
  groupId: 'consume-tweets-for-web-app',
  sessionTimeout: 15000,
  protocol: ['roundrobin'],
  fromOffset: 'earliest' // equivalent of auto.offset.reset valid values are 'none', 'latest
};

var topics = [TOPIC_NAME];
var consumerGroup = new kafka.ConsumerGroup(Object.assign({ id: 'consumer1' }, consumerOptio
consumerGroup.on('error', onError);
consumerGroup.on('message', onMessage);

function onMessage(message) {
  console.log('%s read msg Topic="%s" Partition=%s Offset=%d', this.client.clientId, message
  console.log("Message Value " + message.value)
```

```
var tweetCache = {};
tweetListener.subscribeToTweets((message) => {
  var tweetEvent = JSON.parse(message);
  tweetCache[tweetEvent.tweetId] = tweetEvent;
  updateSseClients(tweetEvent);
})
```

The case at hand
Server sent events for push back



Server sent event – server side

```
const app = express()
  .use(bodyParser.urlencoded({ extended: true }))
  //configure sseMW.sseMiddleware as function to get a stab
  .use(sseMW.sseMiddleware)
  .use(express.static(__dirname + '/public'))
  .get('/updates', function (req, res) {
    var sseConnection = res.sseConnection;
    sseConnection.setup();
    sseClients.add(sseConnection);
  });
```

```
var tweetCache = {};
tweetListener.subscribeToTweets((message) => {
  var tweetEvent = JSON.parse(message);
  tweetCache[tweetEvent.tweetId] = tweetEvent;
  updateSseClients(tweetEvent);
})
```

```
// Realtime updates
var sseClients = new sseMW.Topic();

updateSseClients = function (message) {
  sseClients.forEach(function (sseConnection) {
    sseConnection.send(message);
  }
    , this // this second argument to forEach is the thisArg
  );
}
```

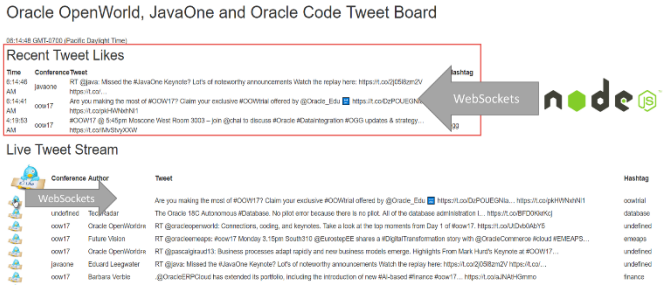
Server sent event – Client side

```
    <h2>Live Tweet Stream</h2>
    <table id="tweetsTable">
        <tr>
            <th><img src="images/like-tweet.jpg" width="100" /></th>
            <th>Conference </th>
            <th>Author </th>
            <th>Tweet </th>
            <th>Hashtag</th>
        </tr>
    </table>
    <br/>
</div>
<script src="js/sse-handler.js">
```

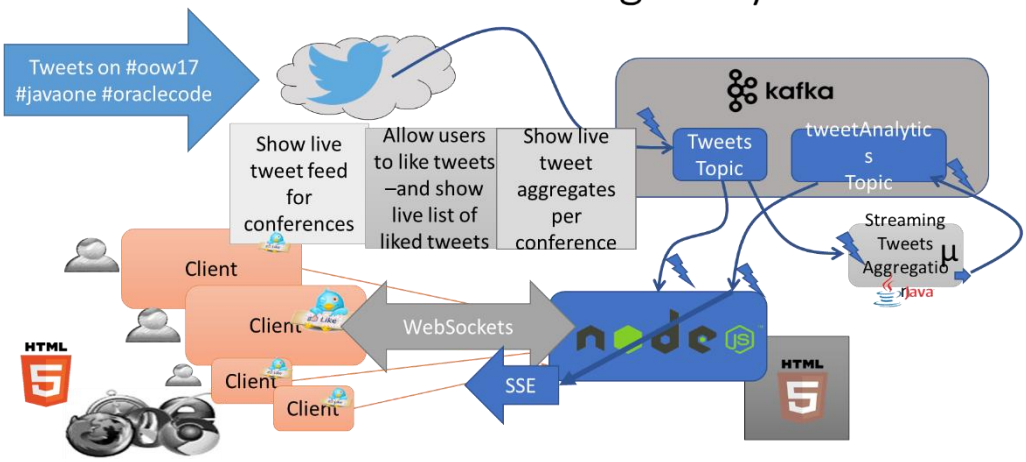
```
var source = new EventSource("../updates");
source.onmessage = function (event) {
    var data = JSON.parse(event.data);
    var table = document.getElementById("tweetsTable");
    var row = table.insertRow(1); // after header
    var likeCell = row.insertCell(0);
    var conferenceCell = row.insertCell(1);
    var authorCell = row.insertCell(2);
    var tweetCell = row.insertCell(3);
    var tagCell = row.insertCell(4);
    conferenceCell.innerHTML = data.tagFilter;
    authorCell.innerHTML = data.author;
    tweetCell.innerHTML = data.text;
    tagCell.innerHTML = data.hashtag;
    likeCell.innerHTML = `<img src="images/like-tweet.jpg" width="40" onclick="like('${data.
};//onMessage
```

Live tweet stream

Tweet likes broadcasting



The case at hand - Streaming analysis of Tweets



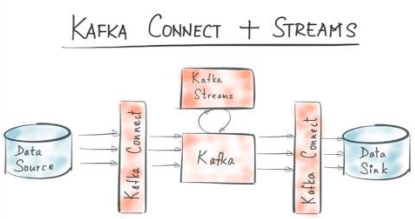
Kafka streams

Real Time Event [Stream] Processing integrated into Kafka
- Aggregations & Top-N
- Time Windows
- Continuous Queries
- Latest State (event sourcing)

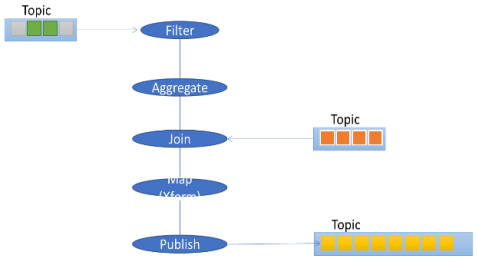
Turn Stream (of changes) into Table (of most recent or current state)
- Part of the state can be quite old

A Kafka Streams client will have state in memory
- Always to be recreated from topic partition log files

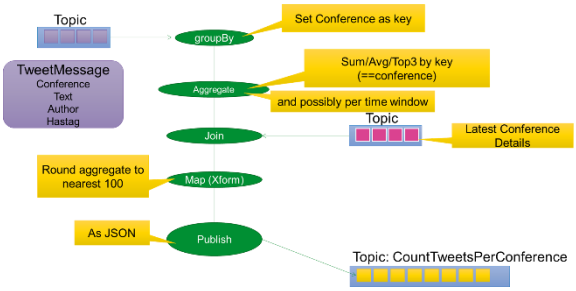
Note: Kafka Streams is relatively new
- Only support for Java clients



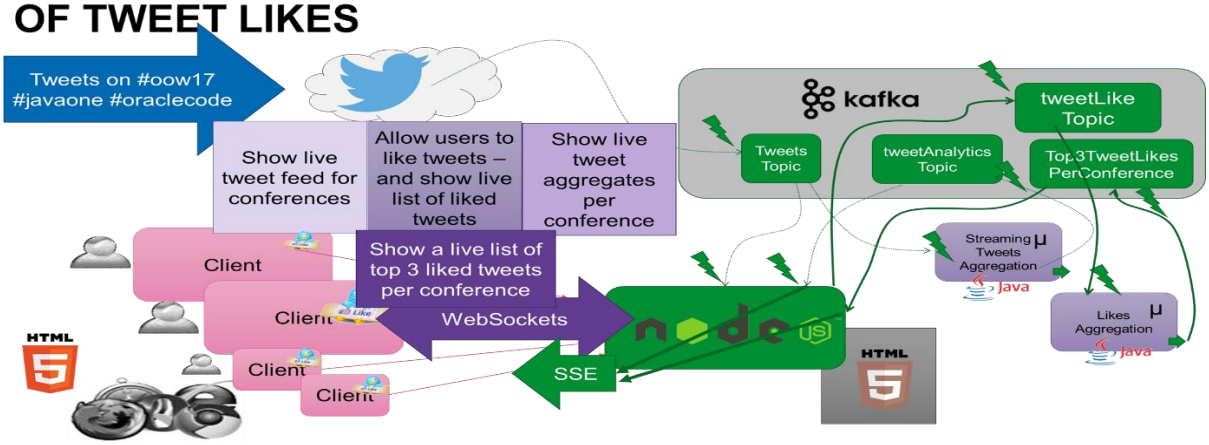
Kafka streams



EXAMPLE OF KAFKA STREAMS



THE CASE AT HAND - STREAMING ANALYSIS OF TWEET LIKES



END TO END FLOW CLOUD ENABLED