Key IoT Challenges

- **Volume & Velocity** – ingesting, processing and storing infinite data in real-time.
- **Connectivity & Bandwidth** – Sensor and video data can hog the internet and cost a fortune, edge analytics must be used to scale and lower response time.
- **Combining Real-time data & Historical state** – cannot separate stream processing from historical context or auxiliary data, need real-time data joins.
- **Real-Time Actions** – Systems need to produce actions in real-time
- **Security** – devices may control vital systems or send private data. New paradigms, strict authentication and access control must be implemented.
Distributing The IoT Problem

Analytic PODs
Per city, building, factory, ..

Cloud (Aggregation)

Backup between neighbor PODs
Different Ways to Build an IoT Analytics POD

SILO & PIPELINE APPROACH
- Complex and expensive
- Impossible governance
- Slow time to insight
- Long app development and integration

DATA HUB APPROACH
- Extreme simplicity and efficiency
- Full tractability, secure data sharing
- Real-time insights
- Instant app deployment
Live Demo: Real-Time IoT End-to-End

Real-Time Data
- Streaming
- State & config
- Remote Control

Real-Time Insights
- Live streams
- Messages
- Real-time Queries

Server-less Event Processing
- Event Driven Code
- Security
- Queries
- Data Objects
- Data Lifecycle

Interactive Analytics
- Apache Zeppelin
- Real-time Predictions

Near Real-Time Analytics
- Spark
- Data Frames
- Vector ops

© 2016 iguaz.io
Real-Time Insights & Actions

Up streams
Events, statistics, Video

Down streams
Commands

Data Objects/Tables
Configuration, State, Auto-aggregations

Up streams
Commands

Down streams
Notifications, video, Enriched Event Streams

Data Objects/Tables
Object State, info
Live Geo/Status Map
Time Series Queries
Accelerated Spark OLAP Queries
System Message Flow

- Triggers Queue (all cars, sharded)
- Real-time Enrichment
- Event Driven Code
- Event Queue (per car)
- Command Queue (per car)

@end
Elastic Scaling of Micro-Services & Spark Executors

Monitor Workers
Load & Lag

PUT /<container>/<topic_path>/<shard_id> HTTP/1.1
Content-Type: application/json
X-igz-function: GetRecords

{  "OpaqueOffset": 86,
   "LagInBytes": 1780,
   "LagInRecord": 3,
   "lagInMilli": 21,
   "RecordsNum": 2,
   "Records": [    
      {        "SequenceNumber": 1,
      "Data": "SGVsbG8=",
      "ClientEventTimeSec": 0,
      "ClientEventTimeNSec": 0
      }
   ]
}

Each response returns Queue lag in time, bytes, and msg count

If Lag increase Create New Instance & Redistribute Shards

Response

© 2016 iguaz.io
Device Tables Data Flow

- Configuration
- State Updates
- Time Series Updates

Cars Table
- Query Car Table (Data Frames)
- Update Predictions (per car)

Region Table

Machine Learning
Simple RESTful APIs for Simple Development

**Add/Update a new record (DynamoDB API)**

```
PUT /<container>/<table_path>/ HTTP/1.1
Content-Type: application/json
X-igz-function: PutItem
{
  "Key": {"Key_Name" : { "S":"Key_value" } },
  "ConditionExpression": "",
  "Item": { "Attr_1" : { "S": "Value_string"}, "Attr_2" : {"N": "6547"} }
}
```

**Run a Query (DynamoDB API)**

```
POST /<container>/<table_path>/ HTTP/1.1
Content-Type: application/json
X-igz-function: GetItems
{
  "TableName" : "Cars",
  "Limit" : 1000,
  "AttributesToGet" : "carId, temp, distance, speed",
  "FilterExpression": "alarm = true AND region = 3"
}
```

**Time Series Aggregations (DynamoDB API)**

```
POST /<container>/<table_path>/ HTTP/1.1
Content-Type: application/json
X-igz-function: UpdateItem
{
  "TableName" : "Cars",
  "Key": {"ID" : {"N" : "1705"}},
  hour_cnt[3] 1 , minute_cnt[35] 1 SET hour_temp_min min(37.2) hour_temp_max max(37.2)"
}
```

**Or if you are really lazy, you can use our SDK (e.g. Python)**

```
# Initialize SDK Context
v3io = V3ioSDK(v3io_addr=v3io_address)

# Open a (Kinesis) Stream, and register a call-back for incoming messages
car_msg = v3io.open_stream(bucket=bucket, topic="/msg/car_msg/"+car_id)
car_msg.set_handler(shard_ids=[0], handler_func=stream_handler)

# Upload file as an (S3) Object
v3io.upload_file(bucket, file_path, dest_path)
```
function getStreamRecords(streamPath, marker) {
  return $http.put(
    // path
    ConfigService.baseUrl + '/' + streamPath,
    // body data
    {
      OpaqueOffset: marker,
      StartingSequenceNumber: 0,
      MaxRecords: 999999999
    },
    // config
    {
      headers: {
        'Content-Type': 'application/json',
        'X-igz-function': 'GetRecords',
        'Cache-Control': 'no-cache'
      }
    }
  );
}
Real-Time Enrichment in the Data Source (Example with Spark)

```scala
sparkContext.cassandraTable("myappspace", "cars_location")
  .joinWithCassandraTable("myappspace", "drivers")
  .collect()

val carsEnrich = Enrichment("/mystat/cars", "CarID", "ID", ...)
val vendorsEnrich = Enrichment("/mystat/vendors", "VendorID", "ID", ...)

spark.readStream("v3io://myapp/mystream/")
  .enrich(carsEnrich, vendorsEnrich).select("<fields>")
```

Cassandra JOIN:
- Using RDD (not Data Frames)
- Override Spark Context (Proprietary)
- Only Table JOINs (No Streams)

Enrichment Approach:
- Use Data Frames
- Generic, Can be added to any data source
- Work across streams and tables
Aggregating Data in Real-Time Using Vector Ops

**THE OLD WAY**
- Minutes, Batch
- CPU Resources: High
- ACID: No

**Real-Time Way**
- Sub-Second, Stream
- CPU Resources: Low
- ACID: Yes (+ Exactly Once Semantics)
- Simple: 10 lines of Spark Code!
# Read text file, split, and reduce
val wc = sc.textFile("/path/to/file.txt")
  .flatMap(_.split(" "))
  .map( word => (word, 1))
  .reduceByKey(_ + _)

# Define output Schema
val schema = StructType(List(
  StructField("word", StringType, false),
  StructField("count", IntegerType, false)))

# Write and aggregate the results
val df = sql.createDataFrame(wc, schema).write.format("v3tbl")
  .option("map", "count=_+_")
  .save("/path/to/words_table")
Thank You!