Storm and Python

Jonathan Harrington

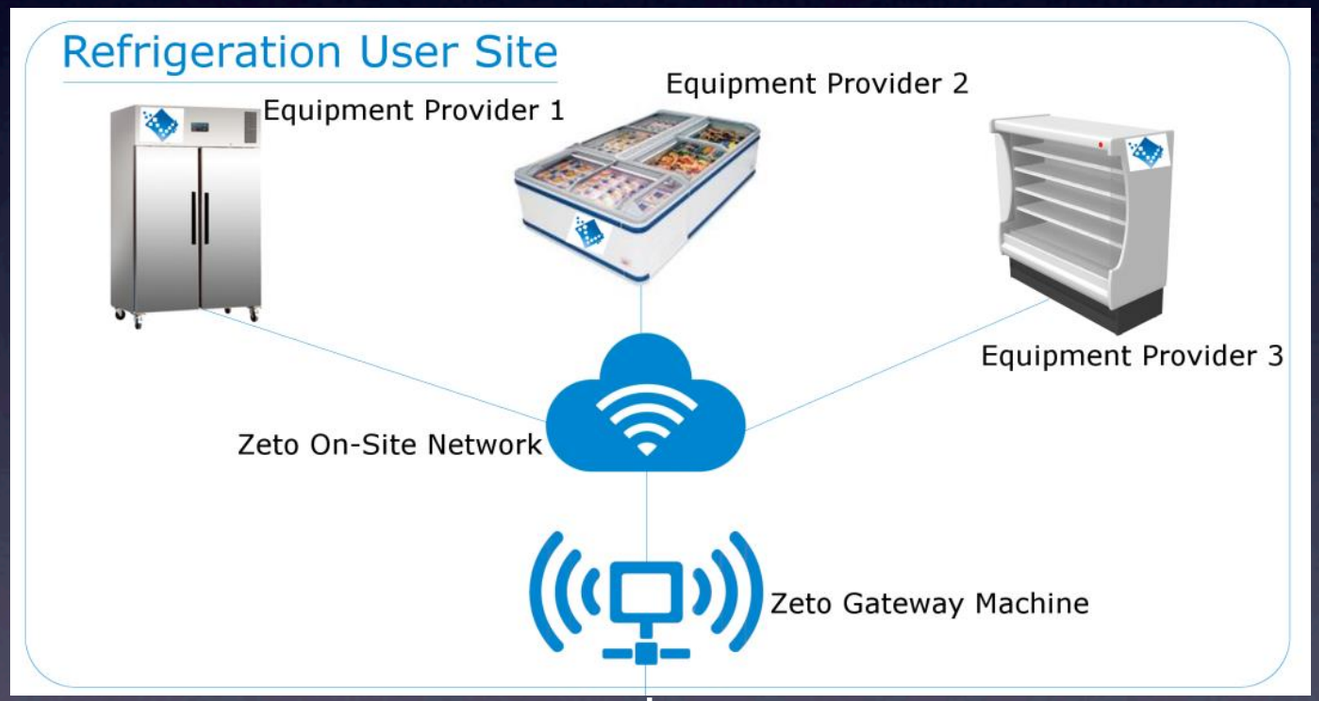
Overview

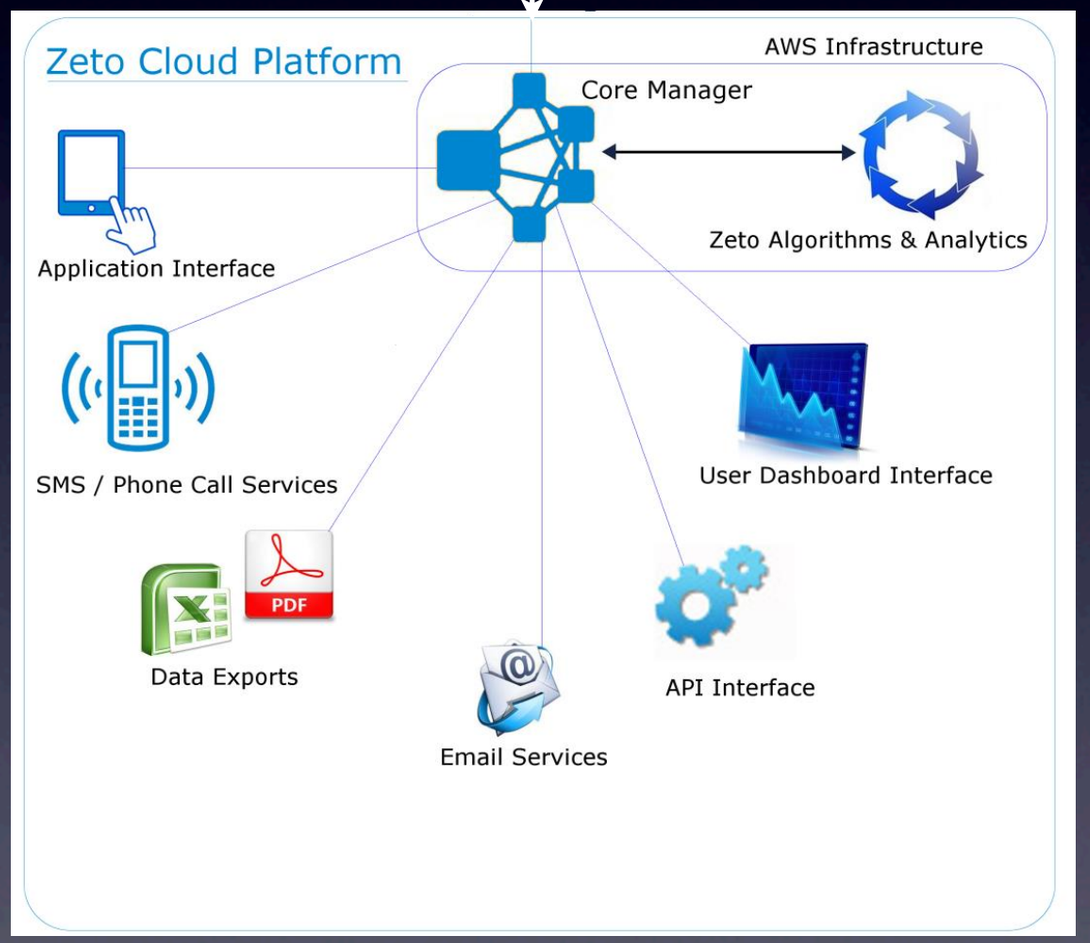
- Me
- Data is good, data is simple
- Real time computation: Storm
- Example
- Summary

Me

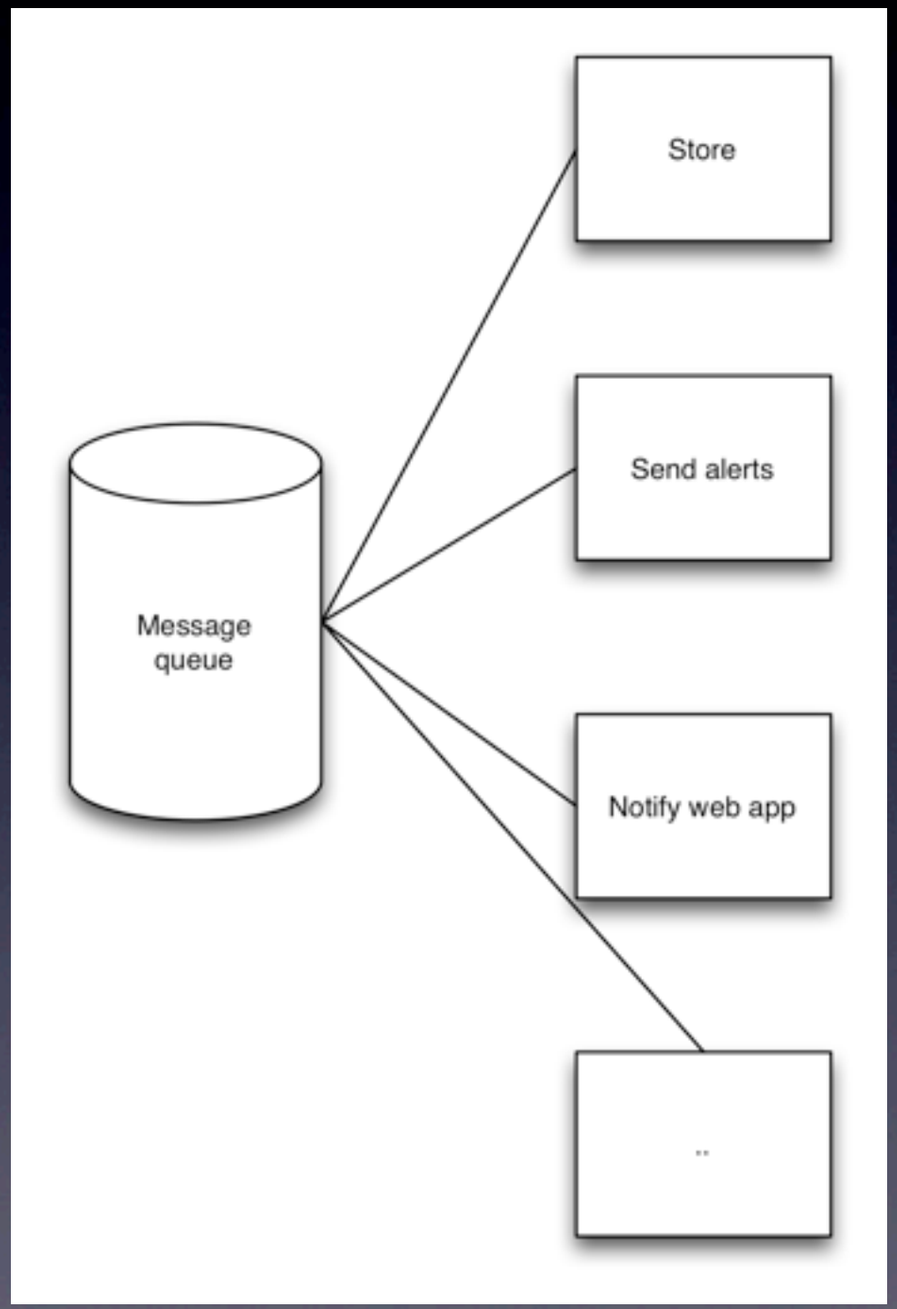
- CTO of Zeto, based in Cork
- Vender neutral refrigeration management
- We use “the right tool for the job”
  - Python, Java, Clojure, C#
- We’re hiring!

Zeto Platform





First design



What I don't want to talk about

What I don't want to talk about

- Is it "Big Data"?
- Is it "Long Data"?
- Is it "Small Data"?

What I don't want to talk about

- Is it "Big Data"?
- Is it "Long Data"?
- Is it "Small Data"?

What I don't want to talk about

- Is it "Big Data"?
- Is it "Long Data"?
- Is it "Small Data"?
- <insert Star Trek Joke>
- I don't really care

What I do want to talk about

- Takeaways from todays talk
  - Data is easy to read
  - Data is easy to ‘debug’
  - Data is easy to manipulate
  - More data, less code

Unix

- 40+ years old and still going strong
- Simple philosophies
  - Everything is a file (incl stdout and stdin)
  - Commands do one thing
- cat access.log | awk '{print $1}' | sort -n | uniq | wc -l

Storm

- "To do for real-time computation what hadoop did for batch computation"
- Hadoop, distributed map-reduce, based on Google research paper
- Hadoop for real-time

Implementation

- Nathan Marz (Backtype, Twitter)
- JVM, Java & Clojure
- Distributed “cluster”
- Zookeeper, Nimbus
- Fault tolerant
- Fail Fast

Getting started

- Local mode vs Remote mode
- storm cli tool
  - deploy/status/start/stop
- Storm deploy
  - Ec2

Terminology

- Spouts
- Streams
- Bolts
- Topologies

Streams

- A sequence of (named) tuples
- Storm gives you the tools to transform a stream in a distributed, reliable fashion
- A stock ticker stream could be transformed into a stream of buy/sell/hold messages

Spouts

- The source of a stream
- Normally reads data from a source
  - Redis, Kestrel, RabbitMQ
- Emits a sequence of (named) tuples
- Reliable vs Unreliable
  - ack and fail

Bolts

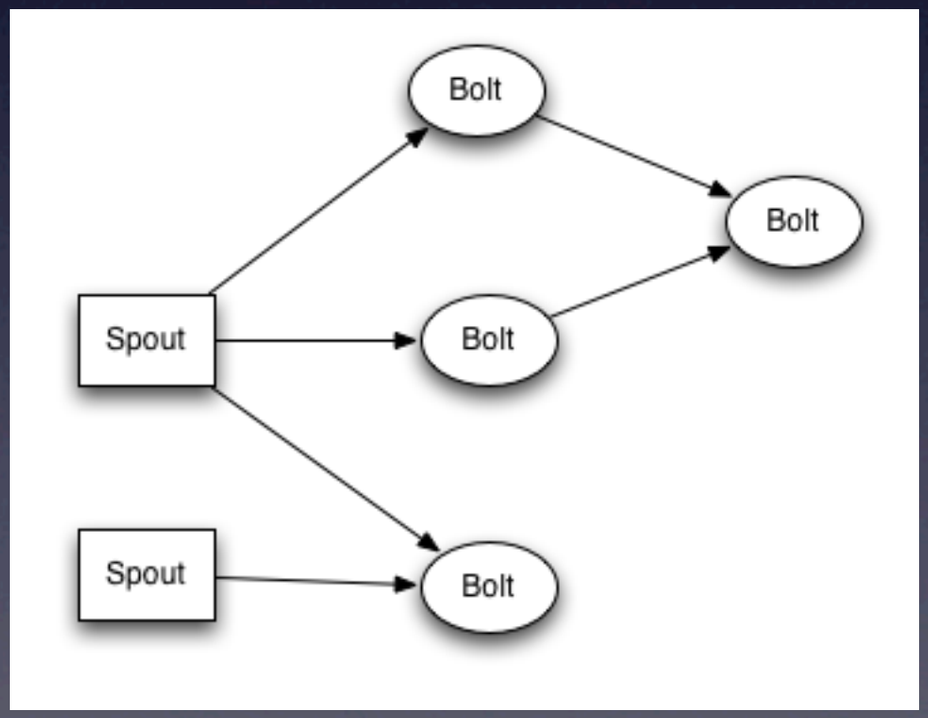
- A bolt consumes one or more input streams
- Does some computation
- Possibly persists some data (local state or db)
- Possibly outputs a stream

Stream Groupings

- How streams are distributed to bolts
  - Shuffle
  - Field
  - Custom (java)

Topology

- "A graph of computation"
  - The description of our system
- Nodes execute in parallel



Multi-lang Protocol

- stdin and stdout
- JSON
- Java wrapper (ShellX)
- PHP, Ruby and Python wrappers

Topology (Code)

- Topologies can be created in Python out of the box, they are just Apache Thrift structures

```
(defn mk-topology []
  (topology
   {"spout" (spout-spec sentence-spout)
   {"split" (bolt-spec {"spout" :shuffle}
                       split-sentence
                       :p 5)
    "count" (bolt-spec {"split" ["word"]}
                       word-count
                       :p 6)}))
```

Spout

- Spouts can be written in any language

```
public static class SentenceSpout extends ShellSpout implements IRichSpout
{
  public SentenceSpout() {
      super("python", "sentencespout.py");
  }
    ...
}
```

```
from random import choice
import storm

class SentenceSpout(storm.Spout):
    sentences = ["a little brown dog", "the man petted the dog",
                 "four score and seven years ago",
                 "an apple a day keeps the doctor away"]

    def nextTuple(self):
        storm.emit([choice(self.sentences)])

SentenceSpout().run()
```

Bolts

- Bolts can be written in any language

```
public static class SplitSentence extends ShellBolt implements IRichBolt {
    public SplitSentence() {
        super("python", "splitsentence.py");
    }

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }
}
```

```
import storm

class SplitSentenceBolt(storm.BasicBolt):
    def process(self, tup):
        words = tup.values[0].split(" ")
        for word in words:
          storm.emit([word])

SplitSentenceBolt().run()
```

Word Count Bolt

```
from collections import defaultdict
import storm

class WordBolt(storm.BasicBolt):
    def initialize(self, conf, context):
        self._count = defaultdict(int)

    def process(self, tup):
        word = tup.values[0]
        self._count[word] += 1
        storm.emit([word, self._count[word]])

WordBolt().run()
```

Petrel

- Open source (created by AirSage)
- Allows writing, submitting, debugging and monitoring pure Python topologies
- Native Python (not Jython), no Java wrappers required
- Actively supported?

Using Petrel

```
def create(builder):
    builder.setSpout("spout", randomsentence.RandomSentenceSpout(), 1)
    builder.setBolt("split", splitsentence.SplitSentenceBolt(), 1)
           .shuffleGrouping("spout")
    builder.setBolt("count", wordcount.WordCountBolt(), 1)
    .fieldsGrouping("split", ["word"])
```

Petrel Spout

```
class RandomSentenceSpout(Spout):
    @classmethod
    def declareOutputFields(cls):
        return ['sentence']

    sentences = [
        "the cow jumped over the moon",
        "an apple a day keeps the doctor away",
        "four score and seven years ago",
        "snow white and the seven dwarfs",
        "i am at two with nature"
    ]

    def nextTuple(self):
        time.sleep(0.25);
        sentence = choice(self.sentences)
        log.debug('randomsentence emitting: %s', sentence)
        storm.emit([sentence])
```

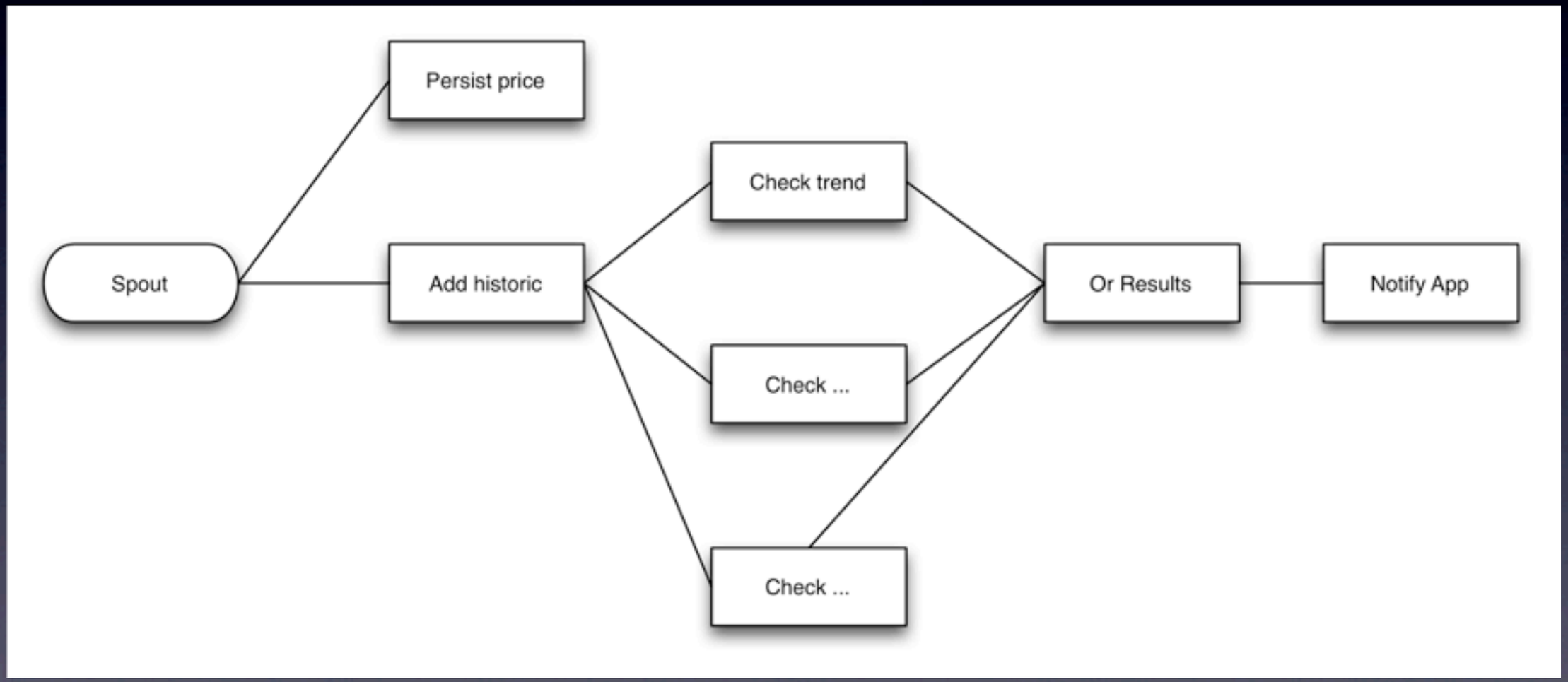
Petrel Bolt

```
class WordCountBolt(BasicBolt):
    def __init__(self):
        super(WordCountBolt, self).__init__()
        self._count = defaultdict(int)

    @classmethod
    def declareOutputFields(cls):
        return ['word', 'count']

def process(self, tup):
    log.debug('WordCountBolt.process() called with: %s', tup)
    word = tup.values[0]
    self._count[word] += 1
    log.debug('WordCountBolt.process() emitting: %s',
              [word, self._count[word]])
    storm.emit([word, self._count[word]])
```

Stock ticker Example

- Read stock ticker data
- Persist latest quote
- Run number of algorithms on the stream
- Determine the position we will take
- Notify external app of position

Topology


Persist price
Check trend
Spout
Add historic
Or Results
Notify App
Check ...
Check ...

Topology

- Spout -> (‘GOOGL’, ‘2012-10-13’, 312.45)
- Historic -> (‘GOOGL’, ‘2012-10-13’, [312.45, 312.68, 311.87, 312.01, 312.92])
- Check X -> (‘GOOGL’, ‘2012-10-13’, ‘hold’)
- Or -> (‘GOOGL’, ‘2012-10-13’, ‘buy’)
- Notify -> /trader/v1/buy/googl?num=100

Topology

```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("spout", new StockSpout(), 5);

builder.setBolt("persist", new PersistBolt(), 5)
       .shuffleGrouping("spout");

builder.setBolt("historic", new HistoricBolt(), 5)
       .fieldsGrouping("spout", new Fields("symbol"));

builder.setBolt("check-trend", new TrendBolt(), 5)
       .fieldGroupings("historic", new Fields("symbol"));
builder.setBolt("check-change", new ChangeBolt(), 5)
       .fieldGroupings("historic", new Fields("symbol"));

builder.setBolt("or", new OrBolt(), 5)
       .fieldGroupings("check-trend", new Fields("symbol"))
       .fieldGroupings("check-change", new Fields("symbol"));

builder.setBolt("notify", new NotifyBolt(), 5)
       .shuffleGrouping("or");
```

StockSpout

```
import storm
import pandas as pd


class StockSpout(storm.Spout):
    def initialize(self, conf, context):
        df = pd.read_csv('table.csv')
        self.rows = df.iterrows()

    def nextTuple(self):
        try:
            row = self.rows.next()[1]
            storm.emit(['YHOO', row['Date'], row['Close']])
        except StopIteration:
            pass


StockSpout().run()
```

Persist Bolt

```
class PersistBolt(storm.BasicBolt):
    def process(self, tup):
        session = tick.Session()
        sym, datestr, price = tup.values
        date = parser.parse(datestr).date()
        item = tick.Tick(sym, date, price)
        session.add(item)
        session.commit()
        session.close()
        storm.log(str(tup))
        #storm.emit(tup.values)
```

Historic Bolt

```
class HistoricBolt(storm.BasicBolt):
    def process(self, tup):
        session = tick.Session()
        sym, date, price = tup.values
        prices = [quote.price for quote in
                  session.query(tick.Tick).order_by(tick.Tick.date)[:5]]
        prices.insert(0, price)
        session.close()
        storm.emit([sym, date, prices])
```

Check(Trend) Bolt

```
class CheckTrendBolt(storm.BasicBolt):
    def process(self, tup):
        sym, date, prices = tup.values
        if self.is_upward_trend(prices):
            position = 'buy'
        elif self.is_downward_trend(prices):
            position = 'sell'
        else:
            position = 'hold'
        storm.emit([sym, date, position])
```

Notify Bolt

```
class NotifyBolt(storm.BasicBolt):
    def process(self, tup):
        sym, date, position = tup.values
        url = '/trader/v1/{}/{}/'.format(position, sym)
        r = requests.put(url, auth=('user', 'pass'))
        storm.log(r.text)
```

Tips (Be like Unix)

- Where possible, make bolts stateless
- Bolts should emit streams
- Bolts should do only one "thing" (preferably a stream transform)
- Think '|'
- Easier debug when data flows from bolt to bolt

DRPC

- IMHO, a bad idea
- I've used DCOM, Cobra, JRI, XML-RPC etc.
- All bad
- Send a message, subscribe to the response

Clojure

- Great language, well worth a look
  - The language is data (Lisp)
- Even if you ignore me, search for Rich Hickey and watch his software design talks
  - http://thechangelog.com/rich-hickeys-greatest-hits/

Summary

- Data is simple
- Storm: Topologies, Spouts, Streams Bolts
- Native Python (Multi-Lang)
- Petrel Python wrapper
- DRPC

We're Hiring

jonathan@zeto.ie

Questions