

Big Data Analytics on Container-Orchestrated Systems

Gerard Casas Saez

University of Colorado Boulder

July 20, 2017

Outline

Why?

Background

Problem statement

Approach

Demo

Evaluation & Results

Related work

Future work

Why?

Keeping up with data growth

Problem

- IOT & Social networks
- Internet traffic
 - Current: 72 petabytes/month
 - Prediction 2021: 232 petabytes/month.



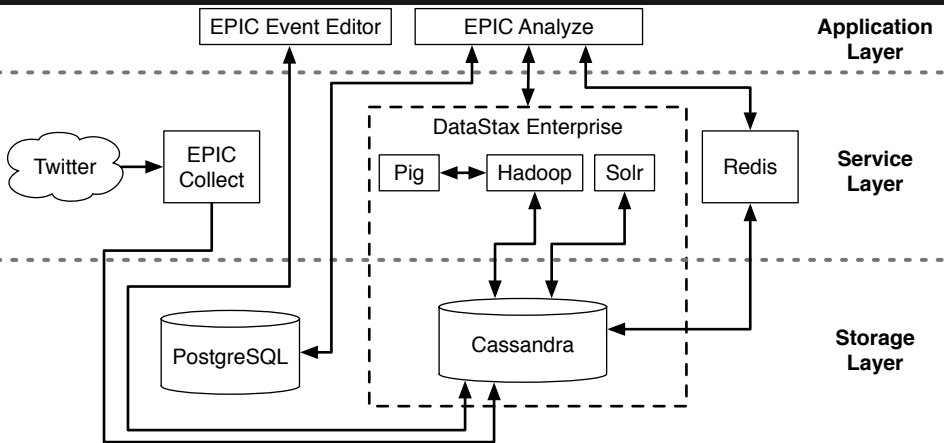
Another problem

- Need to scale Big Data Analytics System
- Keeping maintenance at low cost
- Container-orchestrated make infrastructure easier
- Migrate Project EPIC architecture to container-orchestration system

Background

Project EPIC

- EPIC Collect
- EPIC Analyze



Containerization

- Operating-system-level virtualization
- Use host machine system resources
- **Docker** most used alternative
- Development microservices

Container-orchestration systems

- Container interaction abstraction
- Great to deploy microservices architectures
- Apache Mesos vs **Kubernetes**



Microservices Architecture

- Small & specific
- Better scalability
- Loosely-coupled & highly-cohesive
- Orchestration <> **Coreography**

Coreography microservice architecture

- Easier to extend
- PubSub interaction
- Messaging system: **Apache Kafka**
- Asynchronous



Problem statement

Problem statement

1. Advantages and/or limitations from existing infrastructure
 - 1.1 More reliable?
 - 1.2 More scalable?
2. Lower maintenance costs than the existing infrastructure?
 - 2.1 Easier to deploy?
 - 2.2 Easier to upgrade?
 - 2.3 More resilient to failures?

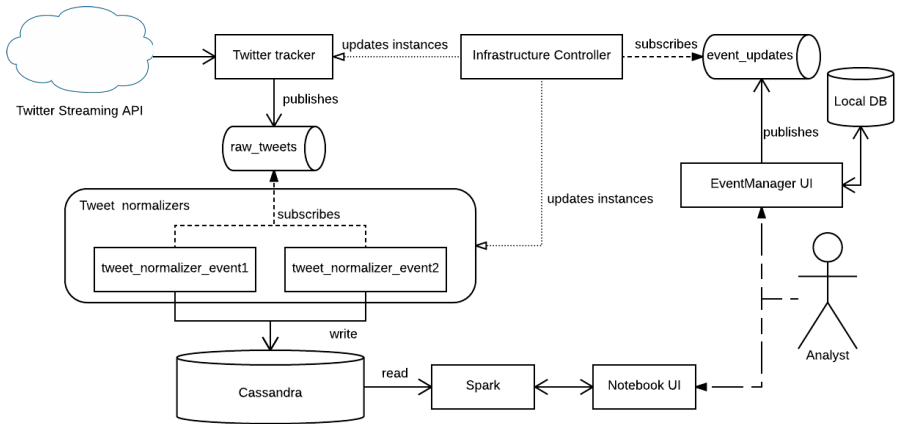
Approach

Features

- Event management
- Real-time collection of streaming Twitter data
- Real-time classification of incoming tweets
- Data Analysis

Non-functional requirements

- Less code
- Easier deployment
- More flexible
- Better scalability



Demo time!

Let's track an event...

Event Manager UI

...and analyze it!

Zeppelin Notebook

Evaluation & Results

Reliability

Current vs Prototype

- EPIC Collect: multi threaded, auto-monitoring
- EPIC Analyze: tight dependency to external resources, manual deployment, no recovery procedure
- Kubernetes abstraction
- Controller manager: Auto-recovery, replica deployment, rolling update
- Scheduler: check memory and cpu and assign instance deployment

| NAMESPACE | NAME | READY | STATUS | RESTARTS | AGE |
|-------------|--|-------|---------|----------|-----|
| cassandra | cassandra-0 | 2/2 | Running | 1 | 2d |
| cassandra | cassandra-1 | 2/2 | Running | 2 | 11d |
| cassandra | cassandra-2 | 2/2 | Running | 7 | 22d |
| cassandra | spark-master-controller-bkd17 | 1/1 | Running | 0 | 22d |
| default | hearth-event-parser-2160998245-m19st | 1/1 | Running | 9 | 2d |
| default | k8s-controller-3919038388-75tkk | 1/1 | Running | 0 | 2d |
| default | smiley-event-parser-3033807940-c0cwj | 1/1 | Running | 9 | 2d |
| default | twitter-tracker-2482383360-s0kz1 | 1/1 | Running | 5 | 2d |
| frontend | eventmanager-ui-3464180876-h605f | 1/1 | Running | 0 | 23d |
| frontend | zeppelin-3633522582-t0kng | 1/1 | Running | 0 | 2d |
| kafka | kafka-0 | 1/1 | Running | 3 | 11d |
| kafka | kafka-1 | 1/1 | Running | 0 | 2d |
| kafka | zoo-0 | 1/1 | Running | 0 | 2d |
| kafka | zoo-1 | 1/1 | Running | 0 | 2d |
| kafka | zoo-2 | 1/1 | Running | 0 | 25d |
| kube-system | heapster-v1.3.0-4211727876-kv0cl | 2/2 | Running | 0 | 11d |
| kube-system | kube-dns-806549836-43lvx | 3/3 | Running | 0 | 11d |
| kube-system | kube-dns-autoscaler-2528518105-2wlsr | 1/1 | Running | 1 | 27d |
| kube-system | kube-proxy-gke-development-development-dcfa2eb3-2jhh | 1/1 | Running | 0 | 2d |
| kube-system | kube-proxy-gke-development-development-dcfa2eb3-l5xb | 1/1 | Running | 1 | 26d |

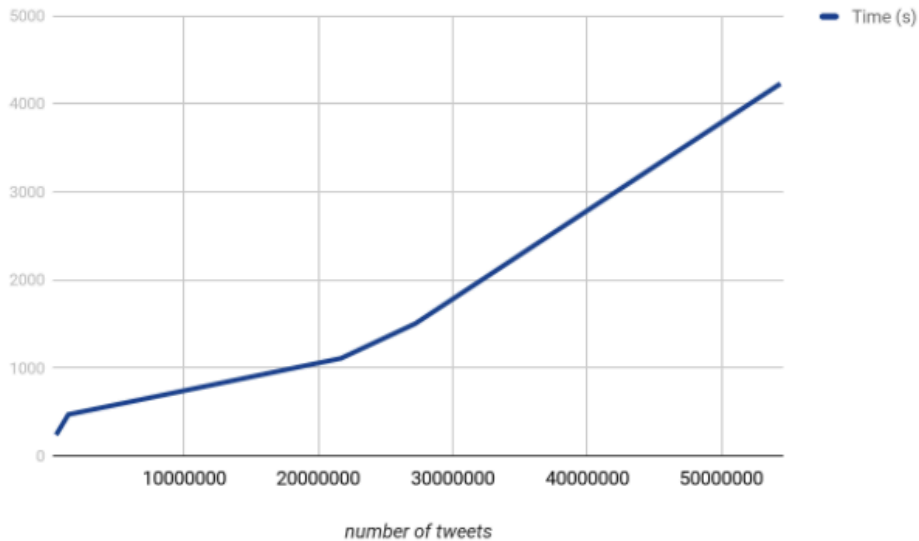
Scalability

Current vs Prototype

- EPIC Collect: manual, huge load, monolith replication
- EPIC Analyze: monolith, load balancing, shared session state
- Not easy and manual
- Replica specification
- Independently scalable
- Stateless microservices
- Auto scale
- Abstracted management interface

Performance

WordCount execution



Development and maintenance

Current vs Prototype

- Large and complex
- Strong use of frameworks
- Simple and small code (mostly)
- Independent microservices
- Technology flexibility
- YAML deployment description files



Results

- High scalability with microservices
- Scale components individually
- Abstraction over system management vs manual
- Monitoring built-in

Results

- Easier to maintain: less code, faster development
- Ease of configuration and deployment
- Flexibility on cloud provider deployment
- Resource usage optimizer
- Other features: Rolling-update, auto-scale...



Related work

Future work

Questions?

Thank you!

Twitter @casassaez

Website gerard.space

Repo github.com/casassg/thesis