DDOS DETECTION W/ KAFKA AND PYTHON

Jake Schwertel for phData

PROBLEM OVERVIEW

- Problem Description
  - A customer runs a website and is periodically attacked by a botnet in a DDOS attack
  - Using Apache log files, create a system that will detect the DDOS attack in real time
- Solution Requirements
  - Use Apache web server logs as input
  - System must be real-time (can detect attack within minutes)
  - Message system (like Kafka) must be used
  - Malicious IP addresses should be output to a directory for further processing

GENERAL SOLUTION

- To find malicious IP addresses – count them!
- For each log:

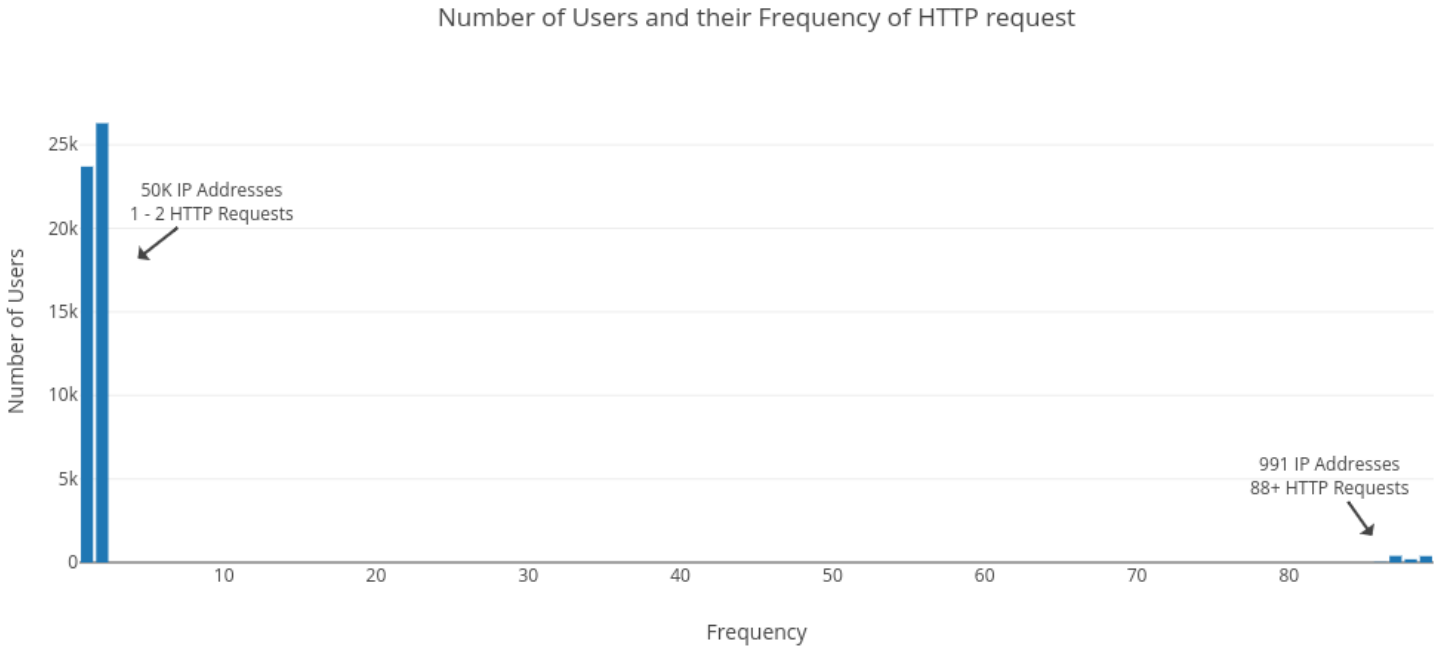
  Parse into JSON document

  Trim document to IP and TS

  Put IP and TS in sliding window

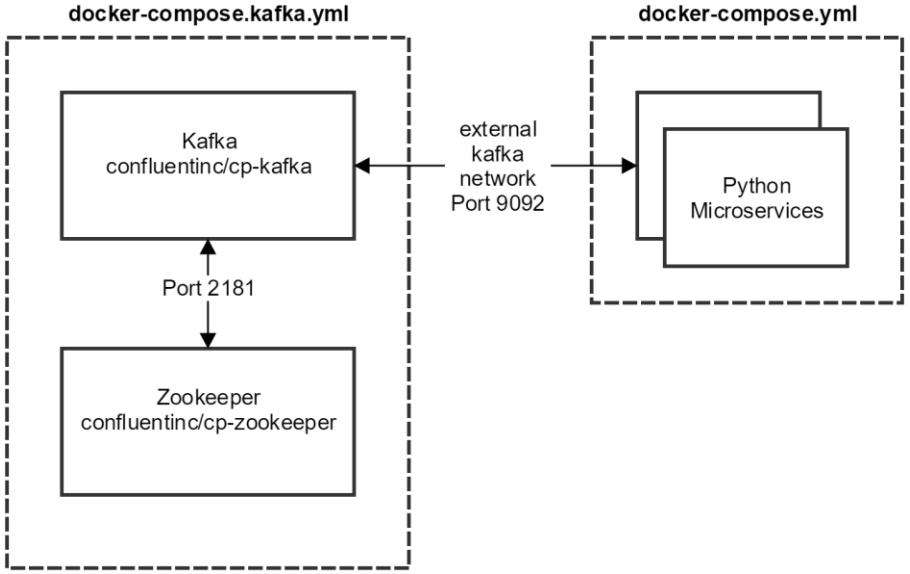
  If num(TS) > threshold:

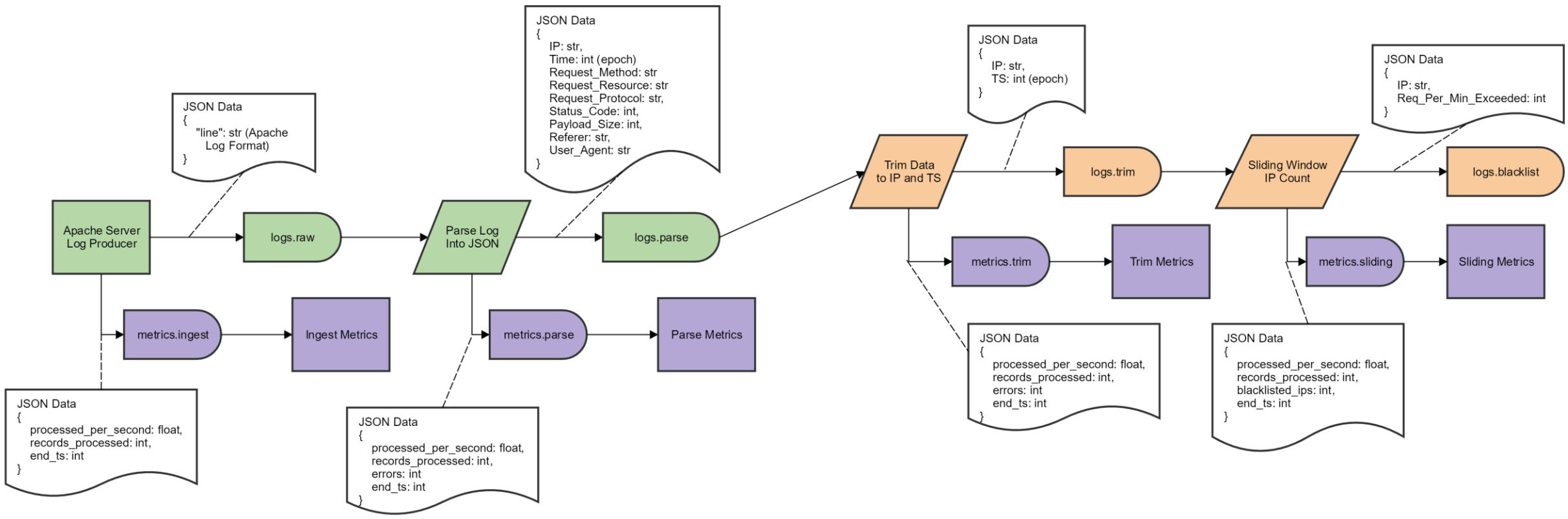
    Write IP address to malicious dir



TECHNOLOGY USED

- Docker
  - Local dev is simple
  - Lots of options with compose
  - Deployment to Cloud is straightforward
- Kafka
  - Distributed & real-time messaging
  - Fault tolerant and persistent
  - High throughput / concurrency
- Python
  - I ♡ Python
  - Kafka Library, up and running quick

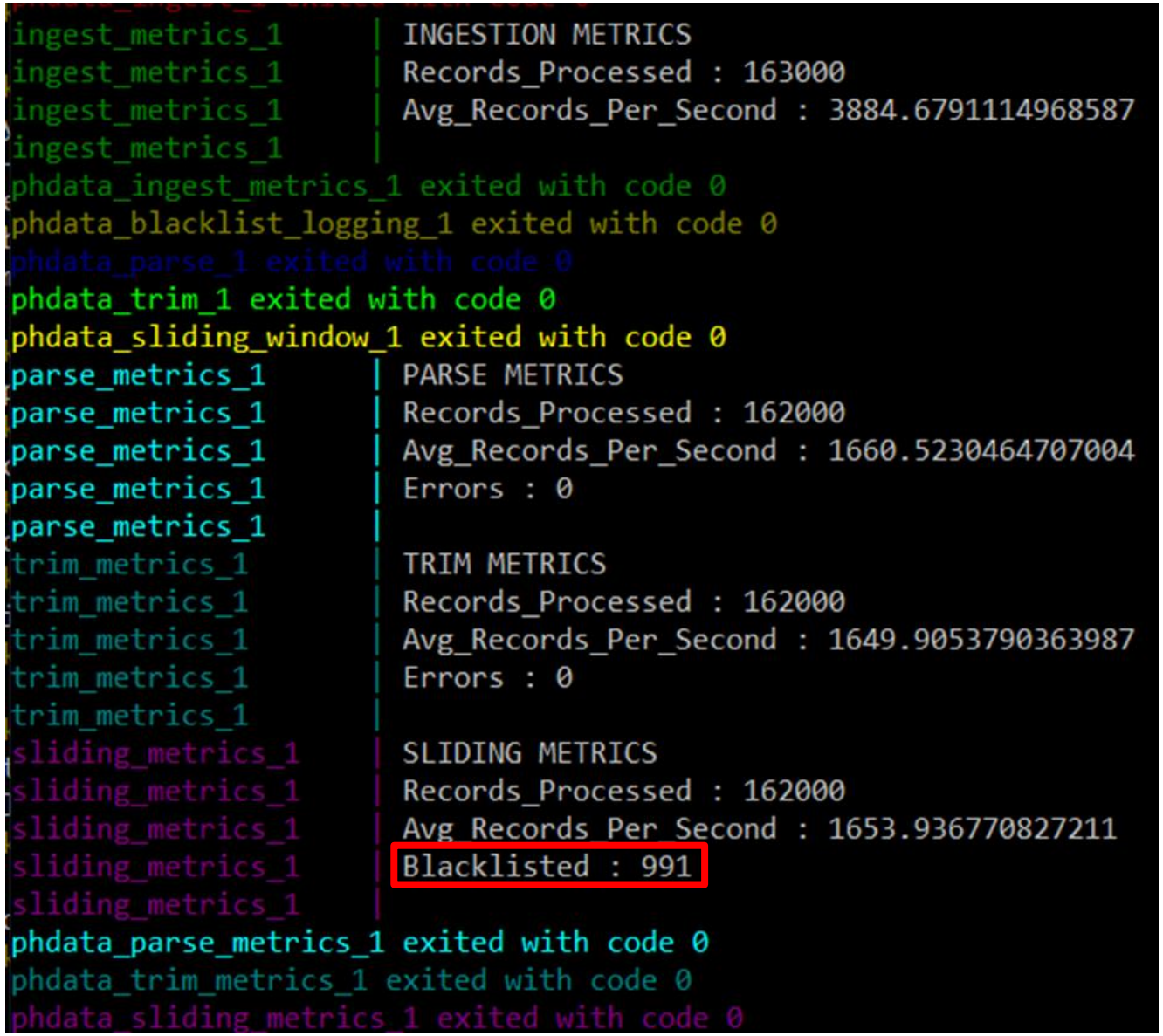




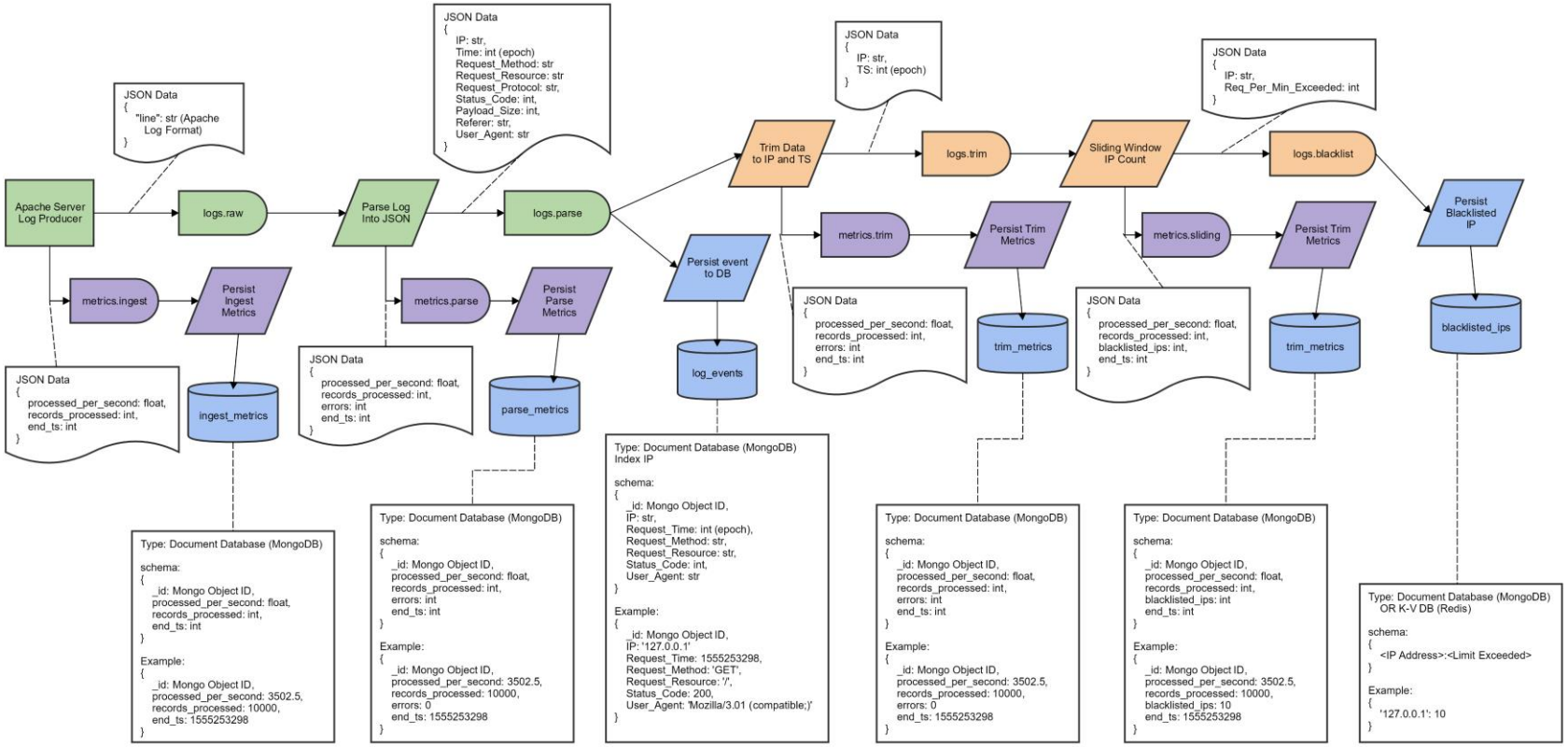
POC ARCHITECTURE

POC TESTING

- Unit Testing
  - In /test directory
  - Covers the important bits of the data transformation pipeline
- System Testing
  - Successfully found all 991 malicious IP addresses that accessed server > 10 times per minutes
  - Throughput *average* of 1600 records per second (4GB Ram / 4CPU)



PRODUCTION CHANGES



- Add Persistence
  - Document DB of some sort (Mongo)
- Persist the following:
  - Metric / Metadata of app performance
  - Full JSON document of parsed log
  - Blacklisted IP addressed
    - Could also use KV DB

SCALING

- Horizontally (many nodes)
- Kafka
  - Parallelism via partitions
  - Number of consumers (in a group) bound by number of partitions
  - Throughput increases, end-to-end latency may increase slightly
- Python Apps (Sliding Window)
  - Resides in memory and could become a concern
  - Only stateful computation – IP addresses must be Kafka Partition Key
  - Consider Spark Streaming
- Data Persistence
  - MongoDB can shard across nodes
    - Requires sharding key – I'd recommend IP address (maybe hashed)
  - Write heavy system – can sacrifice consistency for speed

LESSONS LEARNED / QUESTIONS REMAINING

- Metric aggregation architecture
  - One consumer per metric? One consumer for all metrics? Avoid copy/paste?
- Measuring App performance
  - First, kept cycle data in memory, aggregated and sent to topic
  - Then, measured only elapsed time and record count
  - Needless memory use, inaccurate reporting, average or averages is not the average
- Unit Testing
  - Revise code structure to better decouple data transformations from topic read/write
  - Requires complex mocking otherwise
- Kafka will get a lot more interesting at scale

THANK YOU!

Feedback / Questions?