Playing with Elasticsearch data pipelines

•••

A story of metrics in a distributed architecture environment

Whoami?

- Elasticsearch user and administrator since 2013
- Sysadmin/devops 10+ years
- Infosec enthousiast
- Love computers
- Love building things

https://twitter.com/@moonbocal

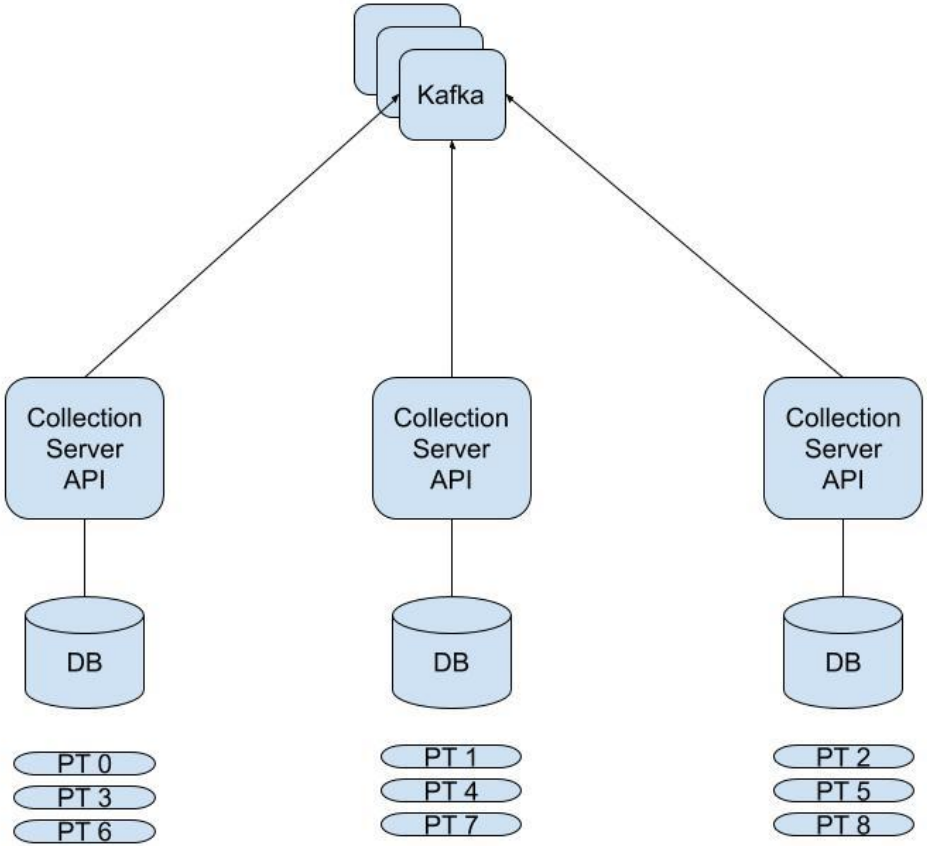
https://medium.com/ebuschini

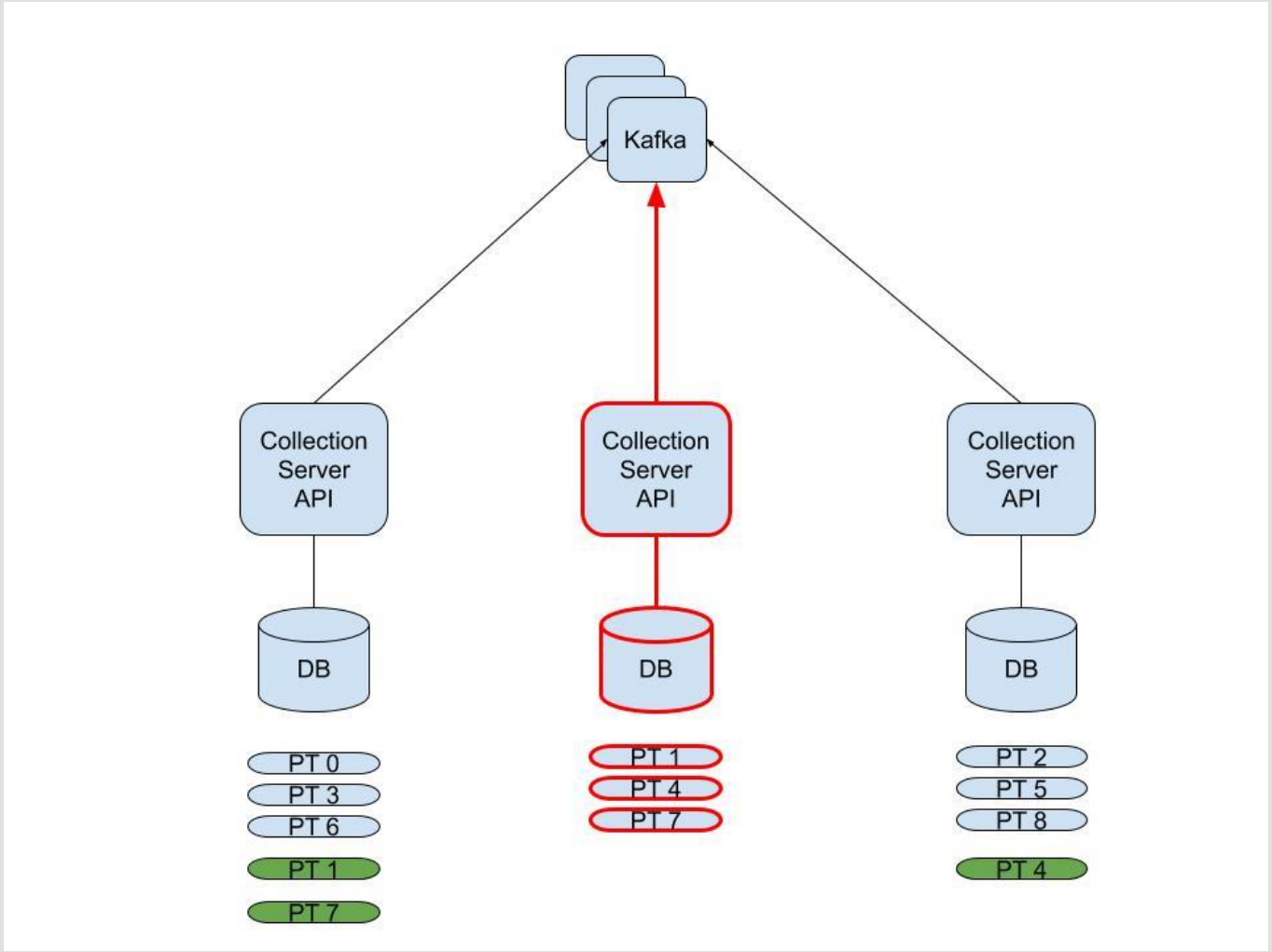
Agenda

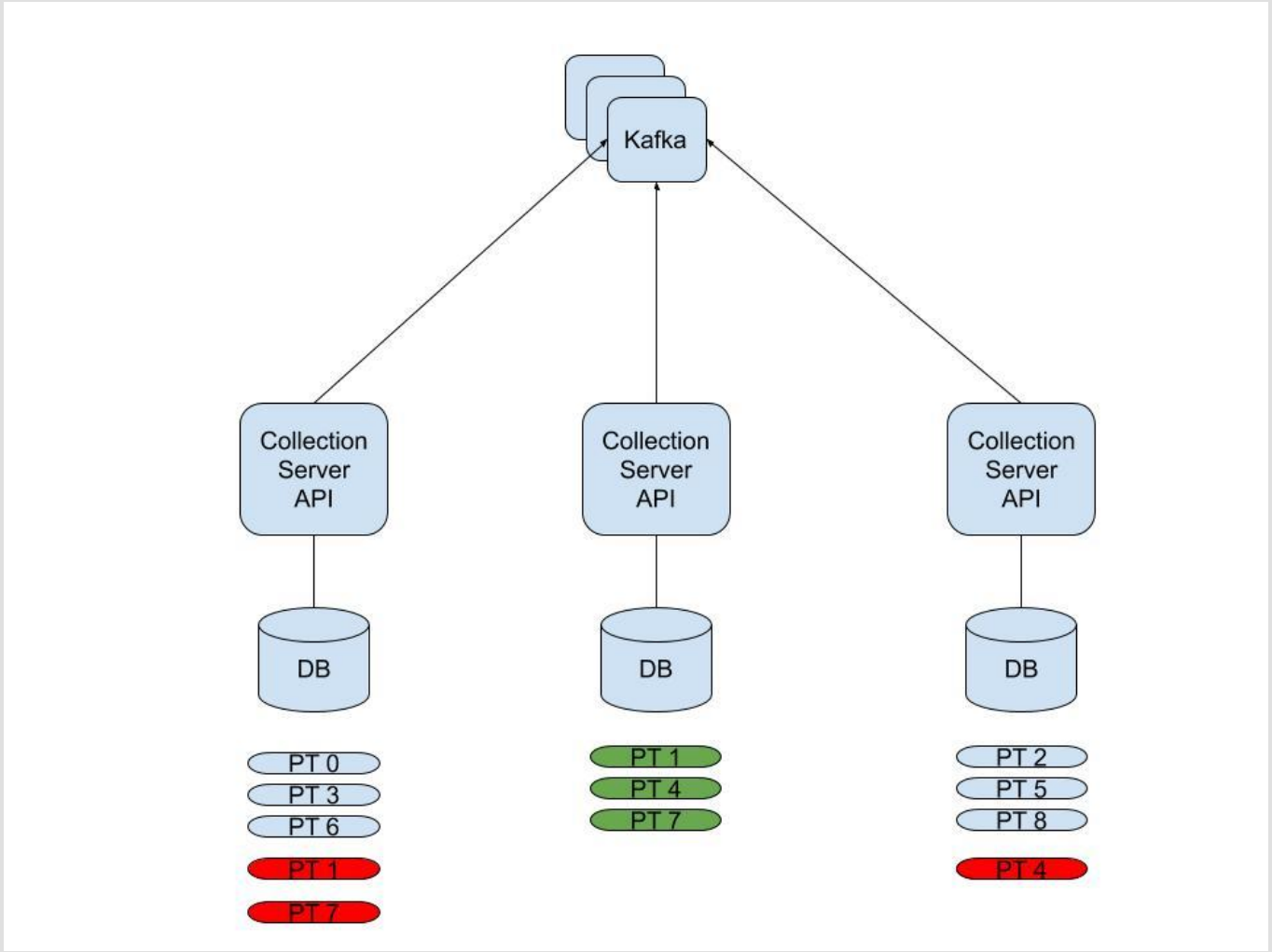
- Distributed architecture
  - Use case
  - Proposed solution
- The tools
  - JQ
  - Cryptocli
  - Json Lines
- Pipeline approach
  - Export data
  - Aggregate data
  - Inject data

Use case

Distributed architecture metrics


Kafka
Collection Server API
Collection Server API
Collection Server API
DB
DB
DB
PT 0
PT 3
PT 6
PT 1
PT 4
PT 7
PT 2
PT 5
PT 8


Kafka
Collection Server API
Collection Server API
Collection Server API
DB
DB
DB
PT 0
PT 3
PT 6
PT 1
PT 7
PT 1
PT 4
PT 7
PT 2
PT 5
PT 8
PT 4


Kafka
Collection Server API
Collection Server API
Collection Server API
DB
DB
DB
PT 0
PT 3
PT 6
PT 1
PT 7
PT 1
PT 4
PT 7
PT 2
PT 5
PT 8
PT 4

Use case -- summary

- Data is sharded
- Build a “consistent” view
- API is too expensive (mesh network)

Proposed solution: Jobs pipeline

- Export data + offsets from servers
- Aggregate data from servers
- Refine data based on offsets
- Aggregated view from latest export
- Re-use data for analytics

Tools

JQ

https://github.com/stedolan/jq

jq is like sed for JSON data - you can use it to slice and filter and map and transform structured data with the same ease that sed, awk, grep and friends let you play with text.

jq can mangle the data format that you have into the one that you want with very little effort, and the program to do so is often shorter and simpler than you'd expect.

- Works with raw strings
- Map/Reduce/Filter
- Lightweight and portable
- Own JQ language
- Active development
- Replace most unix tools
- Easy to read
- Quick prototype
- Pipeline style programing
- Functions!!

jq is written in portable C, and it has zero runtime dependencies. You can download a single binary, scp it to a far away machine of the same type, and expect it to work.

recurse(.children[]) |= (.foo |= "boston")

- Recursively go through the `children` field
- Set the path `.foo` to be equal to `boston`

Cryptocli

https://github.com/tehmoon/cryptocli

Cryptocli

- Portable
- No dependencies
- Chain modules
- Lots of modules!
- Elasticsearch support!

```
cryptocli \
        -- http \
                --url https://test.domain/executable.zip \
        -- tee \
                --pipe “ dgst --algo sha256
                        -- hex -- encode
                        -- stdout ” \
        -- unzip \
        -- file \
                --path executable \
                --write \
                --mode 0755
```

JSON Lines

https://github.com/tehmoon/cheatsheet/blob/master/tools/json_lines.sh

JSON Lines

- Shell script
- Job framework
- JSON string all input lines into an array
- You decide the length of the array
- When EOF, send status == “stopped”
- Use netcat to send tcp
- Compatible with filebeat tcp module


```
"json_lines": {
  "created_at": "2019-08-22T01:24:01Z",
  "type": "users",
  "id": "cd64498ae3132b2843fbc153",
  “max_offset”: 10,
  “lines”: [{“line”: “test”, “line_offset”: 0}...],
  “length”: 10,
  "status": "running"
}
```

```
"json_lines": {
  "created_at": "2019-08-22T01:24:01Z",
  "type": "users",
  "id": "cd64498ae3132b2843fbc153",
  "status": "stopped",
  "took": 0.4199999999254942,
  "username": "nobody"
}
```

Tools -- summary

- Two simple and portable executable (windows/osx/linux/bsd)
- Really quick prototypes
- Easy to use
- Easy to learn
- Easy to switch from/to
- Stream oriented
- Create/filter on jobs
- Use beats as transport
- Create job framework

Example

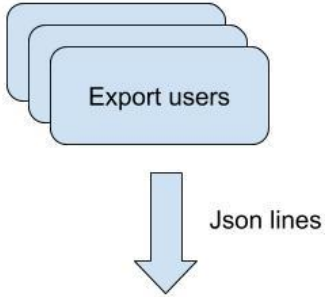
Let’s build a pipeline

Example

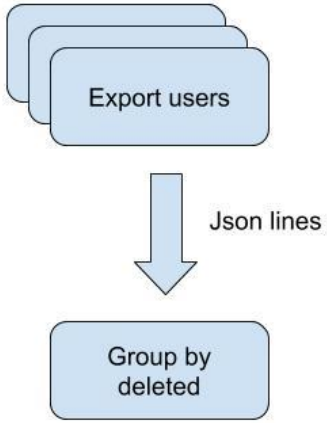
- Sharded database
- Export all users with status
- Aggregate users by status
- Count total of users by status
    - active/deleted
    - Store latest count
    - Store snapshot count for active users (aggregation issue)

Let’s glue tools together to build what we want

- 2 fields:
  - “deleted”: bool
  - “email”: string

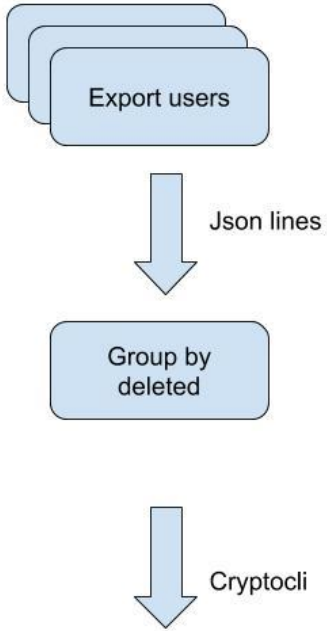


- 2 fields:
  - “deleted”: bool
  - “email”: string



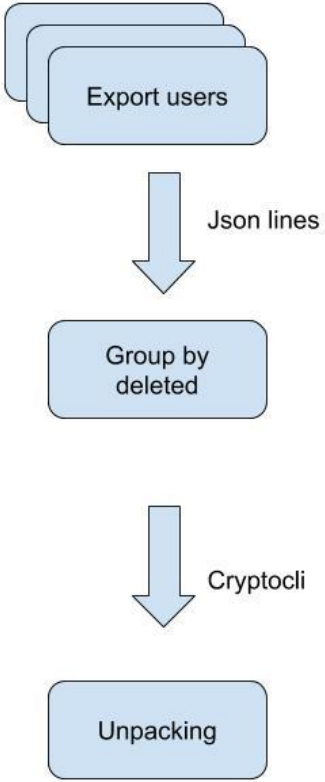
- 2 fields:
  - “deleted”: bool
  - “email”: string

- Get last job on each server
- Group by “deleted” field [0][1]:
  - “emails”: [string]
  - “deleted”: bool
  - “len”: int



- 2 fields:
  - “deleted”: bool
  - “email”: string

- Get last job on each server
- Group by “deleted” field [0][1]:
  - “emails”: [string]
  - “deleted”: bool
  - “len”: int



- 2 fields:
  - “deleted”: bool
  - “email”: string

- Get last job on each server
- Group by “deleted” field [0][1]:
  - “emails”: [string]
  - “deleted”: bool
  - “len”: int

- Get all group groupped by jobs
- Remove all emails
- Count total of users
- Cleanup unused fields

- 2 fields:
  - “deleted”: bool
  - “email”: string

Json lines

- Get last job on each server
- Group by “deleted” field [0][1]:
  - “emails”: [string]
  - “deleted”: bool
  - “len”: int

Cryptocli

- Get all users groupped by jobs
- Remove all emails
- Count total of users
- Cleanup unused fields

Cryptocli

- Use the same ID
- Use “latest” as ID

Cryptocli https://github.com/tehmoon/cryptocli

- Get all the documents that match the query
- From the last 10 days
- Aggregates the json_lines ids by timetstamp

```
cryptocli \
  -- elasticsearch-get \
    --index filebeat* \
    --server http://192.168.56.99:9200 \
    --from now-10d \
    --query 'json_lines.status: "stopped" AND json_lines.type: "users"' \
    --aggregation '{
      "terms": {
        "field": "json_lines.id",
        "size": 1,
        "order": {"timestamp": "desc"}
      },
      "aggs": {"timestamp": {"max": {"field": "@timestamp"}}}
    }' \
  -- stdout |
```

JQ

https://github.com/stedolan/jq

- Unmarshal JSON all the lines in an array
- Group by “deleted”
- For each entry
  - Pack all the email addresses
  - Count the length
  - Set the right deleted bool

Group by deleted

```
jq -crn '
  [ inputs |
    ._source |
    .json_lines.lines[].line |
    fromjson
  ] |
  group_by(.deleted) |
  map({
    "users": [.[].email],
    "len": (. | length),
  "deleted": .[0].deleted})
'
```

JQ

https://github.com/stedolan/jq

- Select if “message”
- “Users” equals to:
  - Unmarshal all the lines
  - “Deleted”: # deleted users
  - “Active”: # non deleted users
  - “Total”: “deleted” + “active”
- Set index to “users”
- Remove “message”
- Remove “json_lines”

Unpack

```
jq -r --unbuffered -c '
  select(._source.message) |
  ._source.users = (
    (._source.json_lines.lines[0].line | fromjson) |
    reduce .[] as $item (
      {};
      if $item.deleted == true
      then
        .deleted = ($item.len | tonumber)
      else
        .active = ($item.len | tonumber)
      end
    )
  ) |
  ._source.users.total = (._source.users.deleted + ._source.users.active) |
  ._index = "users" |
  del(._source.message) |
  del(._source.json_lines.lines)
' |
```

Cryptocli https://github.com/tehmoon/cryptocli

- Read from stdin
- Read lines
- Create new pipeline
  - Fork jq
    - Set ID to “latest”
  - Send to elasticsearch
- Send to elasticsearch

Save to elasticsearch

```
cryptocli \
  -- stdin \
  -- line \
  -- tee --pipe "
    fork jq -ncr --unbuffered '
      [inputs][0] |
      ._id = \"latest\"
    ' \
    -- elasticsearch-put \
      --server http://192.168.56.99:9200
  " \
  -- elasticsearch-put \
    --server http://192.168.56.99:9200
```

Demo

Questions

Thank you!!!

https://twitter.com/@moonbocal

https://github.com/tehmoon/cryptocli