ActiveData

Components and Capabilities

Desired Features

Service

Data

Analysis

Desired Features

Service

Data

Analysis

Data is just a function call away

Less Problems

- Centralization point
- redundancy
- backup and recovery
- security updates
- scaling
- latency problems

Desired Features

Service

Data

Analysis

Shape and Organization

- The shape of the data is important for manipulation.
- Organization of data is important for discovery.
- Locating data; required data is sparse
- Names of columns/fields
- Relations to connected data
- Similar data
- Shape of the data; standard data
- Schema management

Desired Features

Service

Data

Analysis

Well featured language to manipulate data

- Support complex queries
- Link/join to other data
- Compare data to expected values
- Update and fix data
- Transform for presentation
- Apply business rules

Service: Database?

Problems

- Queries consume resources, locking database from other transactions
- Does not contain all enterprise data
- Human-managed schema
- Too many data shape choices (what to index, 1-1 table or use nulls, denormalize for query speed)
- Limited to a single machine
- SQL is not suited for simple queries
- SQL computationally unbounded (QoS)

Service: Elasticsearch?

- No locking, non-transactional - low query latency
- Everything indexed - low query latency
- Scales to multiple machines
- Redundancy and recovery - hot backup
- Offset query load from other systems
- Document database is denormalized, is a hierarchical database
- Handle multitude of schemas from many sources
- No loops, no joins, computationally bounded

Elasticsearch on Github

Service: Elasticsearch?

Problems

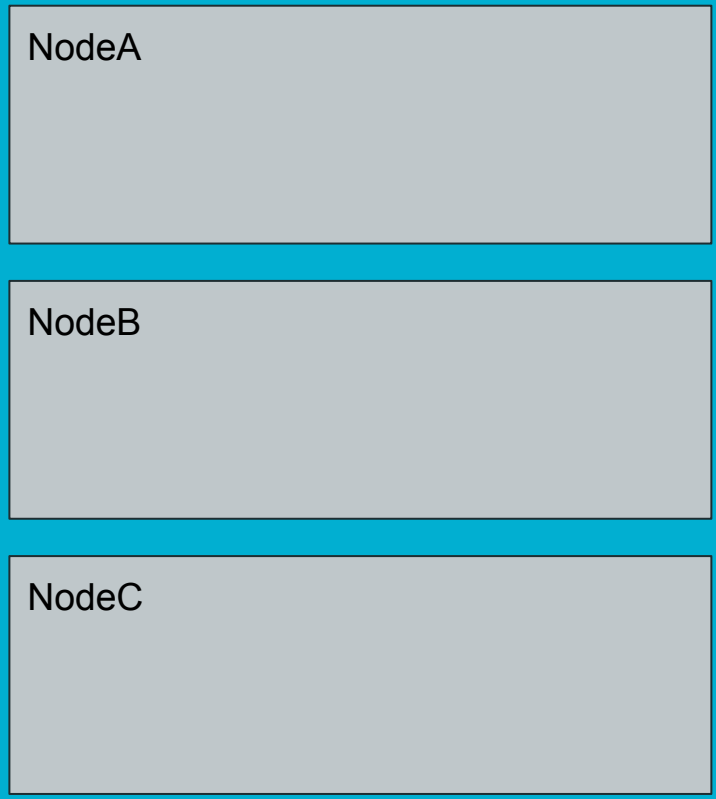
- No security: Can not exposed to public
- Query language not suited for analysis
- Separate service
  - Operational costs
  - Data transfer is required
- Many machines is expensive
- OutOfMemory exceptions

Elasticsearch on Github

Service: Elasticsearch (overview)

How to get data onto many nodes?

?



Service: Elasticsearch (overview)

Split data into shards

Shard 1

Shard 2

Shard 3

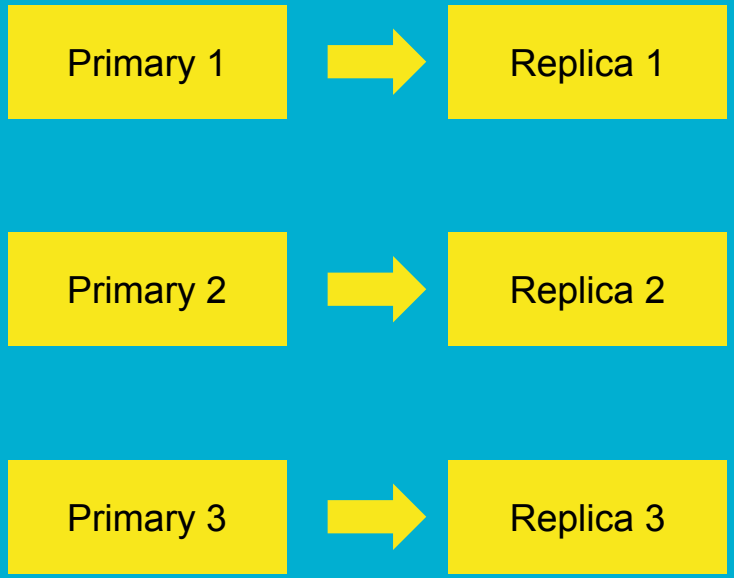
NodeA

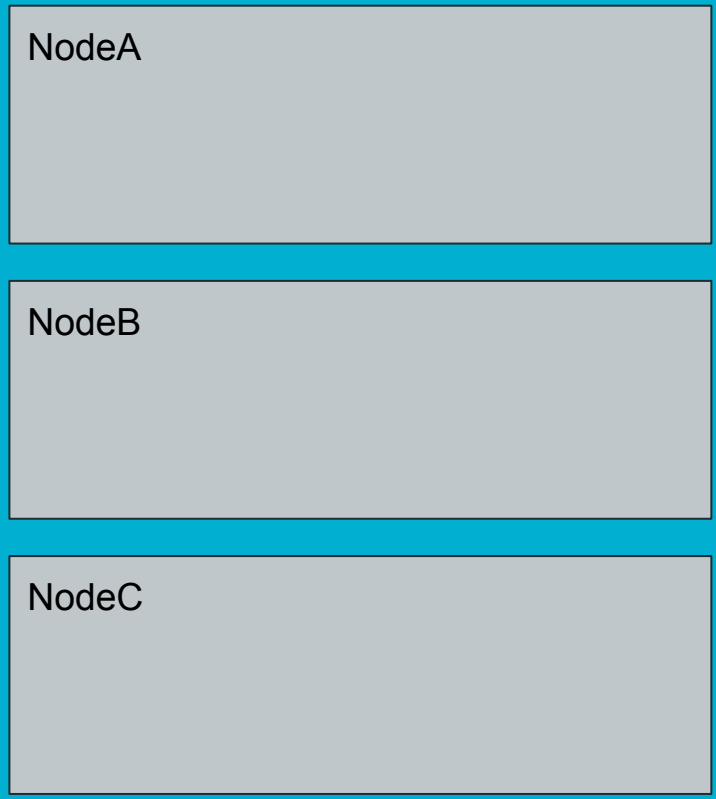
NodeB

NodeC

Service: Elasticsearch (overview)

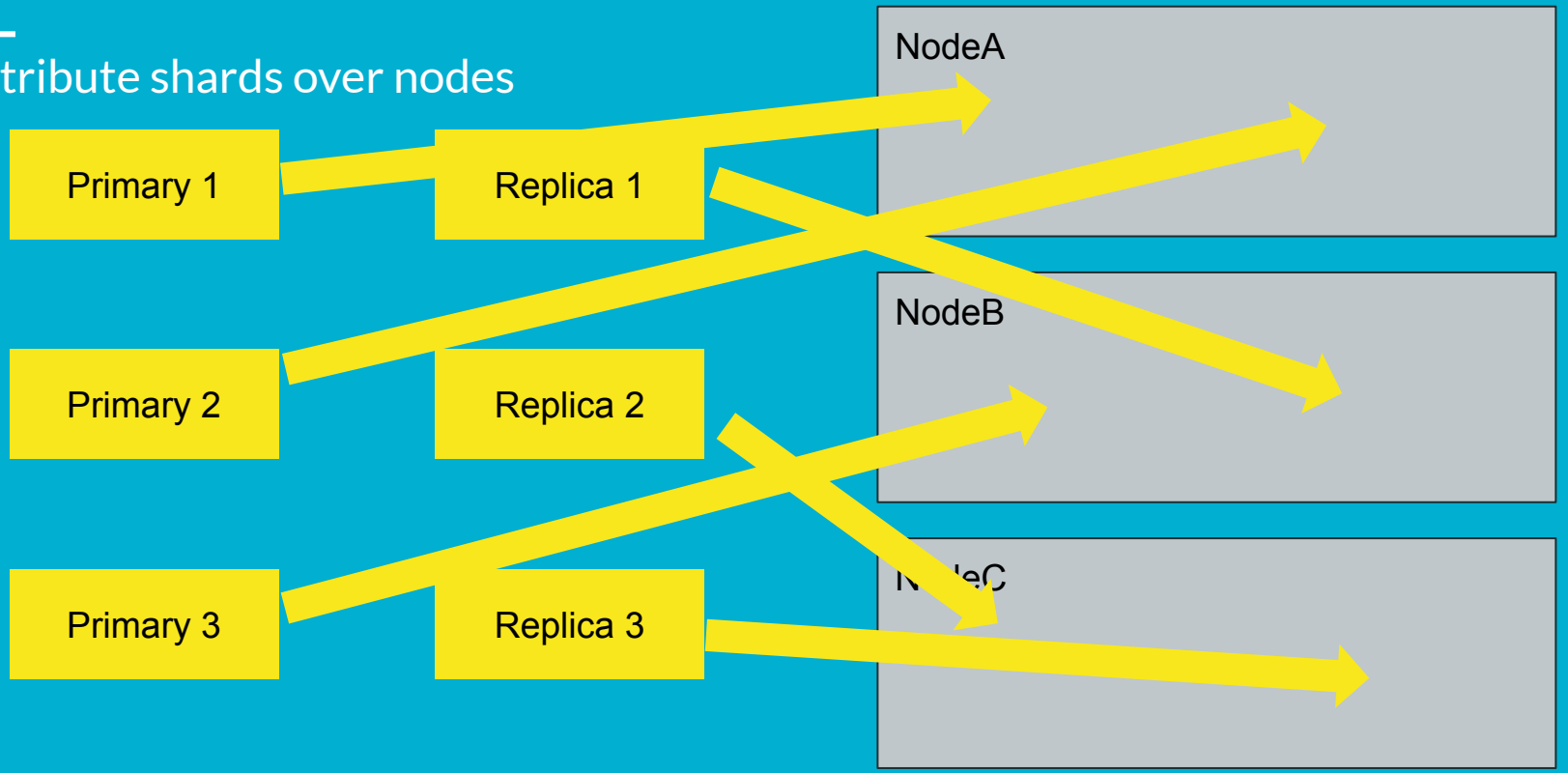
Make replica of each shard





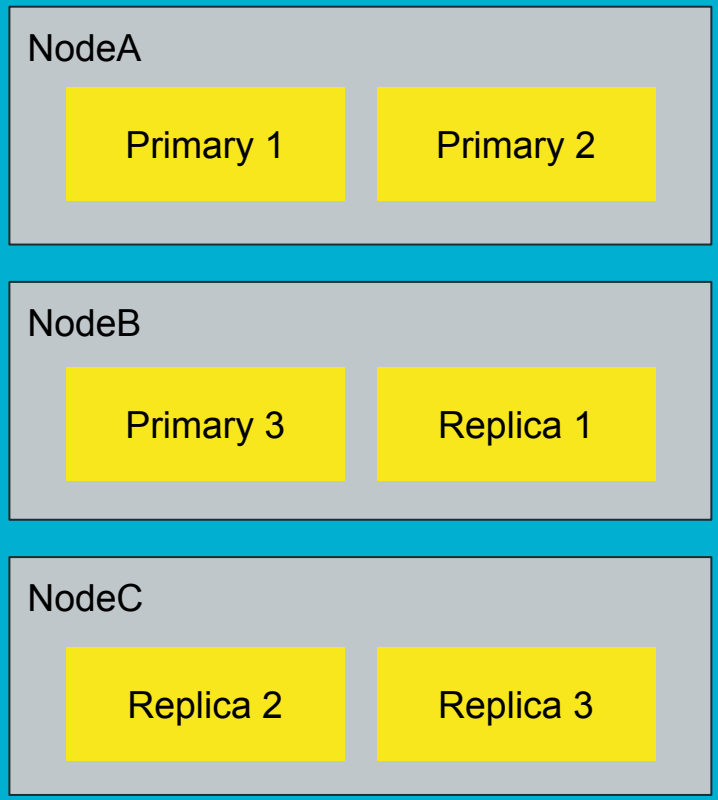
Service: Elasticsearch (overview)

Distribute shards over nodes



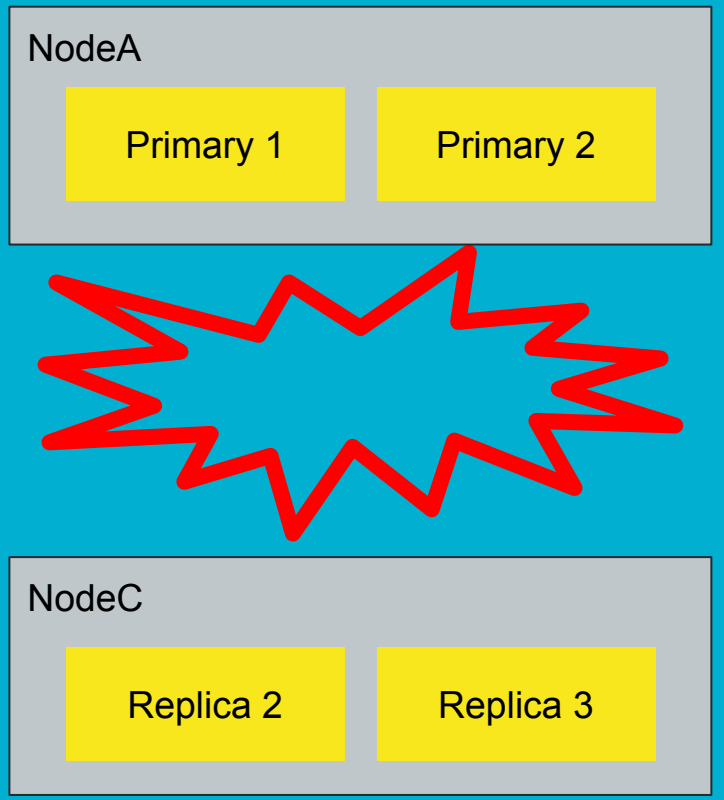
Service: Elasticsearch (overview)

Done!



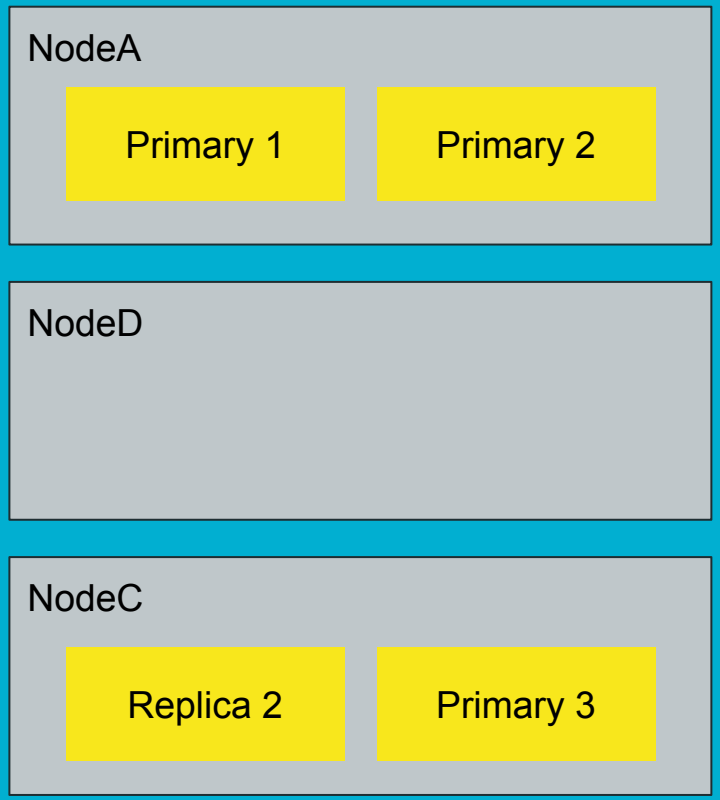
Service: Elasticsearch (overview)

If we loose a node...



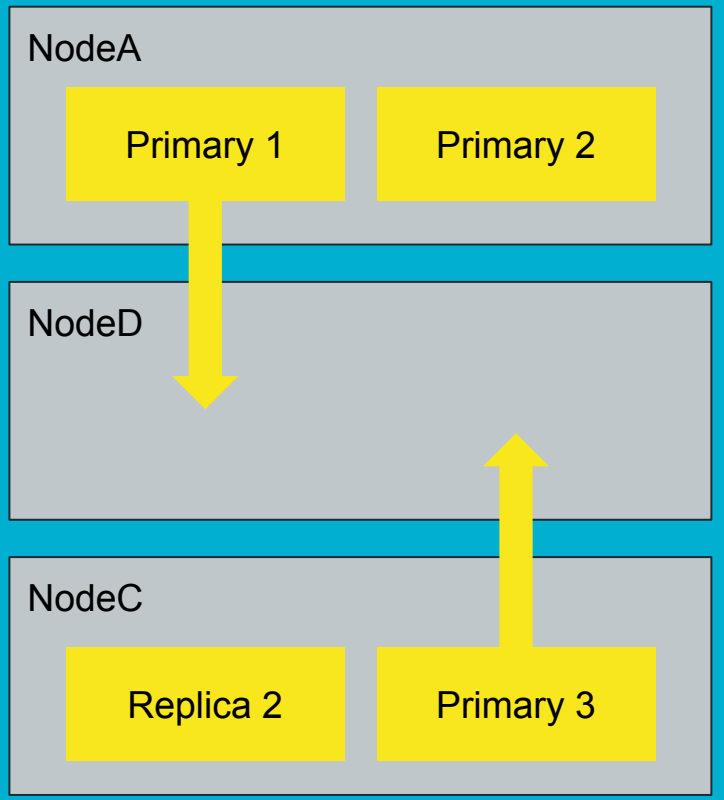
Service: Elasticsearch (overview)

...and a new node joins the cluster...



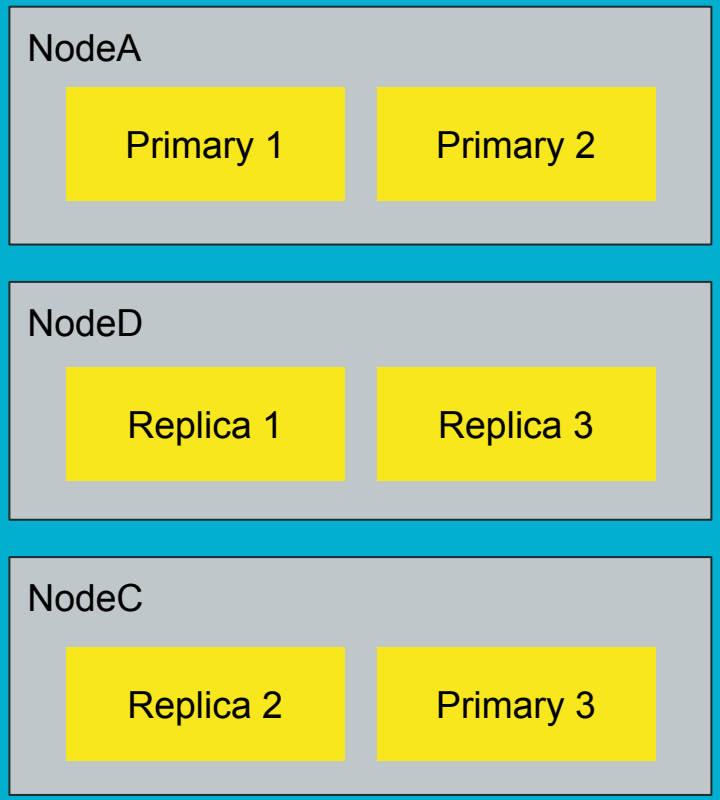
Service: Elasticsearch (overview)

.... recovery is automatic



Service: Elasticsearch (overview)

Repaired!

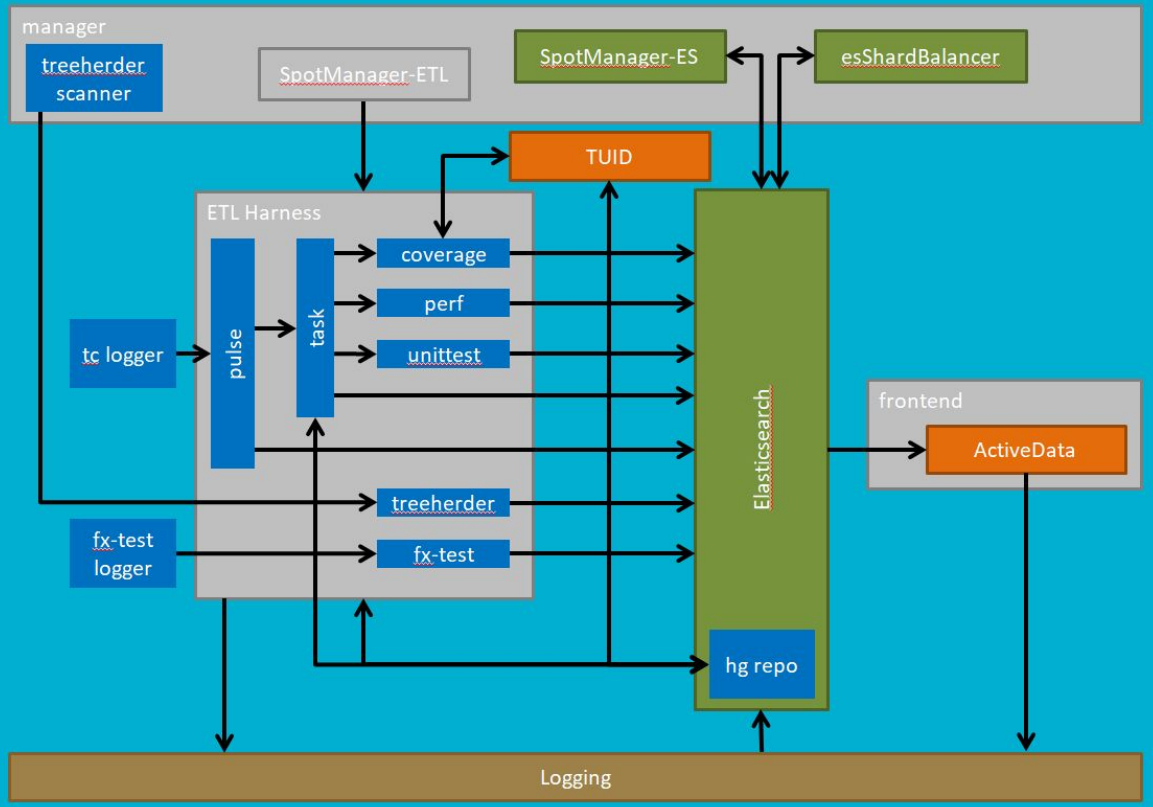


Service: ActiveData

Solves the limitations of Elasticsearch

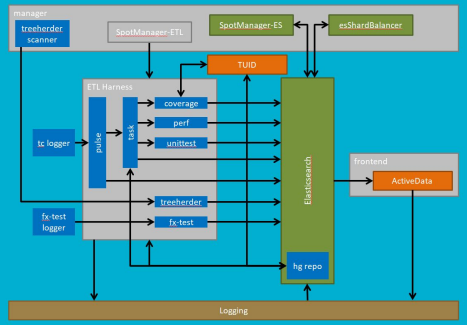
- Public gateway for Elasticsearch
- Manage the multitude of machines
- Ingests data from various sources
- Simpler data model (shape reduces choices)
- Simpler query language

Service: ActiveData



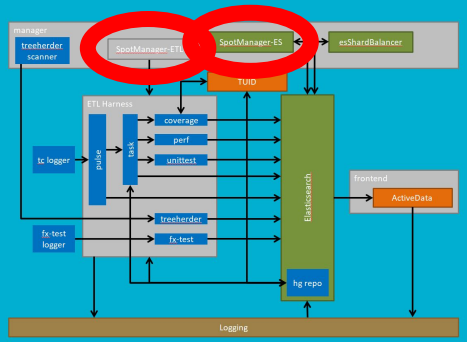
Logical architecture

Service: ActiveData



Logical architecture

ActiveData: SpotManager

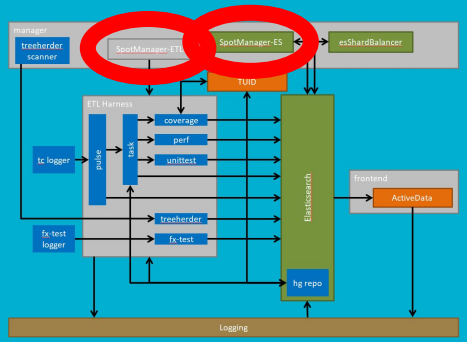


Provides many machines at a low price

- Spot instances are 1/4 to 1/10 the price
- Number of instances limited by budget
- Monitors price fluctuations to target desired uptime
- Set machine diversity to mitigate price fluctuations
- Setup ephemeral (local) drives
- Request number of EBS drives

more on SpotManager

ActiveData: SpotManager



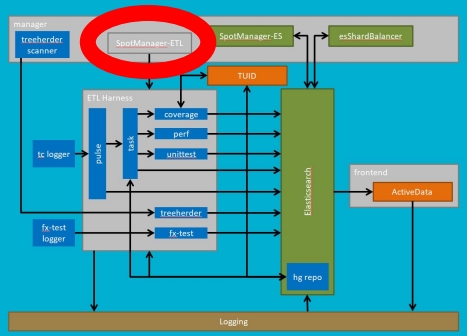
Inputs:

1. Instance Type -> Utility score
2. Current desired utility
3. Script to setup machine

more on SpotManager

ActiveData: SpotManager

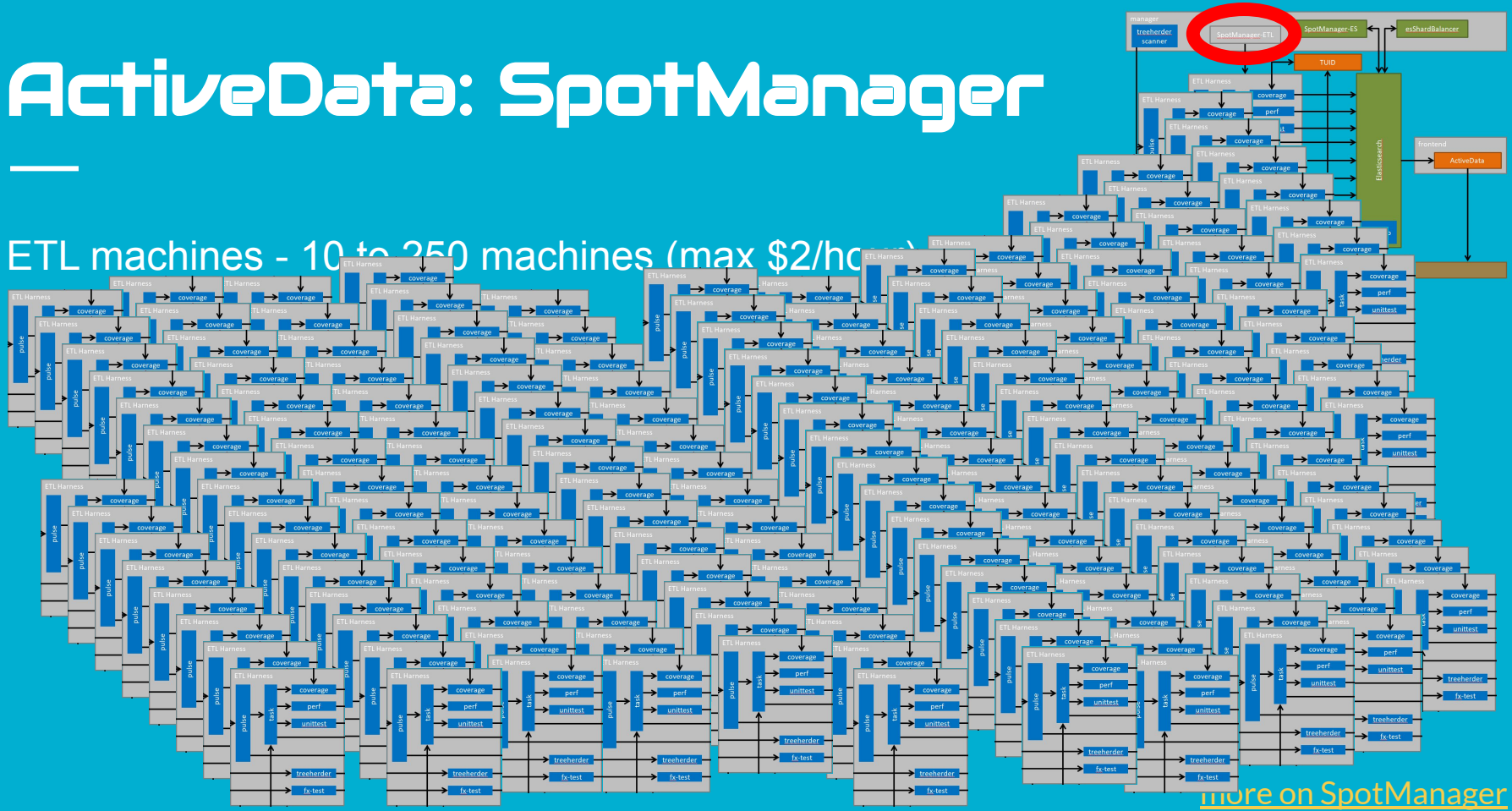
ETL machines - 10 to 250 machines (max $2/hour)



more on SpotManager

ActiveData: SpotManager

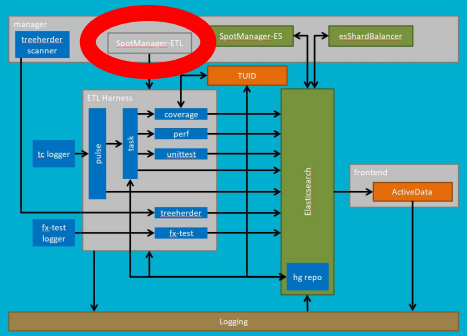
ETL machines - 10 to 250 machines (max $2/hour)



More on SpotManager

ActiveData: SpotManager

ETL machines - cpu count -> Utility score

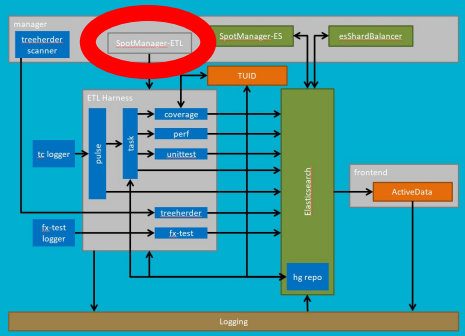



```
{"instance_type": "m2.xlarge", "num_drives": 1, "memory": 17.1, "cpu": 2, "utility": 2},
{"instance_type": "m2.2xlarge", "num_drives": 1, "memory": 34, "cpu": 4, "utility": 4},
{"instance_type": "m2.4xlarge", "num_drives": 2, "memory": 68, "cpu": 8, "utility": 8},

{"instance_type": "m3.medium", "num_drives": 1, "memory": 3.75, "cpu": 1, "utility": 1},
{"instance_type": "m3.large", "num_drives": 1, "memory": 7.5, "cpu": 2, "utility": 2},
{"instance_type": "m3.xlarge", "num_drives": 2, "memory": 15, "cpu": 4, "utility": 4},
{"instance_type": "m3.2xlarge", "num_drives": 2, "memory": 30, "cpu": 8, "utility": 8},
```


more on SpotManager

ActiveData: SpotManager

ETL machines - Current desired utility

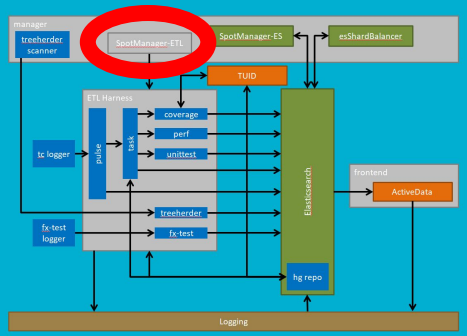



```
def required_utility(self, current_utility=None):
    queue = aws.Queue(self.settings.work_queue)
    pending = len(queue)

    tod_minimum = None
    if Date.now().dow not in [6, 7] and Date.now().hour not in [4, 5, 6, 7, 8, 9, 10, 11]:
        tod_minimum = 100
    minimum = max(self.settings.minimum_utility, tod_minimum)

    if current_utility < pending / 20:
        # INCREASE
        return max(minimum, Math.ceiling(pending / 20))  # ENSURE THERE IS PLENTY OF WORK BEFORE MACHINE IS DEPLOYED
    else:
        # DECREASE
        target = max(minimum, min(current_utility, pending*2))
        return target + int((current_utility-target) / 2)
```

more on SpotManager

ActiveData: SpotManager

ETL machines - Script to setup machine

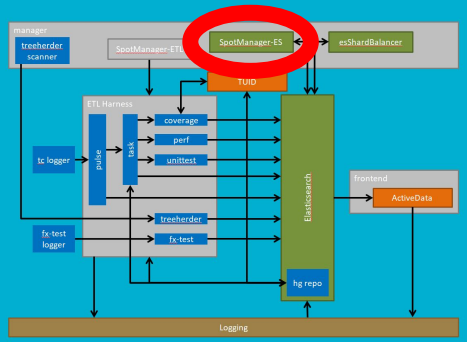



```
def setup_etl_supervisor(self, conn, cpu_count):
    # INSTALL supervsor
    conn.sudo("apt-get install -y supervisor")
    # with fabric_settings(warn=True:
    conn.sudo("service supervisor start")

    # READ LOCAL CONFIG FILE, ALTER IT FOR THIS MACHINE RESOURCES, AND PUSH TO REMOTE
    conf_file = File("./examples/config/etl_supervisor.conf")
    content = conf_file.read_bytes()
    find = between(content, "numprocs=", "\n")
    content = content.replace("numprocs=" + find + "\n", "numprocs=" + str(cpu_count) + "\n")
    File("./temp/etl_supervisor.conf.alt").write_bytes(content)
    conn.sudo("rm -f /etc/supervisor/conf.d/etl_supervisor.conf")
    conn.put("./temp/etl_supervisor.conf.alt", '/etc/supervisor/conf.d/etl_supervisor.conf', use_sudo=True)
    conn.run("mkdir -p /home/ubuntu/ActiveData-ETL/results/logs")

    # POKE supervisor TO NOTICE THE CHANGE
    conn.sudo("supervisorctl reread")
    conn.sudo("supervisorctl update")
```

more on SpotManager

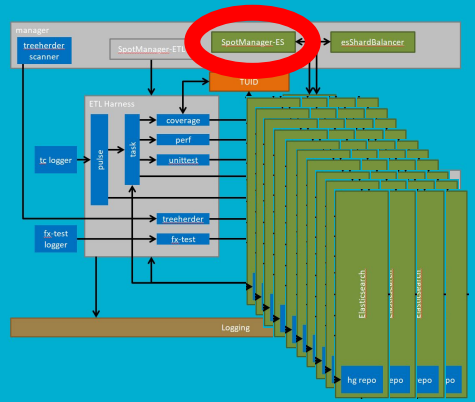
ActiveData: SpotManager

ES machines - 40 machines (max $4/hour)



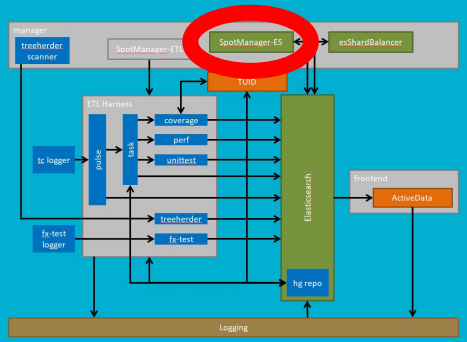
ActiveData: SpotManager

ES machines - 40 machines (max $4/hour)



ActiveData: SpotManager

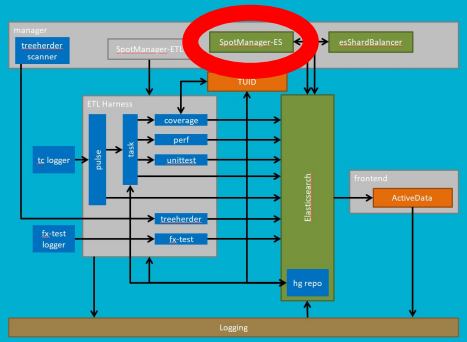
ES machines - General configuration




```
{
    "budget": 4.00,                 //MAXIMUM SPEND PER HOUR FOR ALL INSTANCES
    "max_utility_price": 0.02,      //MOST THAT WILL BE SPENT ON A SINGLE UTILITY POINT
    "max_new_utility": 120,         //MOST NEW UTILITY THAT WILL BE REQUESTED IN A SINGLE RUN
    "max_requests_per_type": 2,     //LIMIT THE NUMBER OF NET-NEW REQUESTS BY TYPE
    "max_percent_per_type": 0.50,   //ALL INSTANCE TYPES MAY NOT GO OVER THIS AS A PERCENT OF T
    "uptime":{
        "history": "week",          //HOW MUCH HISTORY TO USE
        "duration": "day",          //HOW LONG WE WOULD LIKE OUR MACHINE TO TO STAY UP
        "bid_percentile": 0.95      //THE PROBABILITY WE ACHIEVE OUR UPTIME
    },
    "price_file": "resources/aws/prices.json",
    "run_interval": "10minute",     //HOW LONG BEFORE NEXT RUN
    "availability_zone": "us-west-2c",
    "product":"Linux/UNIX (Amazon VPC)",
    "aws": {
        "$ref": "//~/private.json#aws_credentials"
    },
    "more_drives":[
        {"path":"/data1", "size":1000, "volume_type":"standard"}
    ],
    "1_ephemeral_drives":[
        {"path":"/data1", "device":"/dev/sdb"}
```

ActiveData: SpotManager

ES machines - Machine type -> Utility score



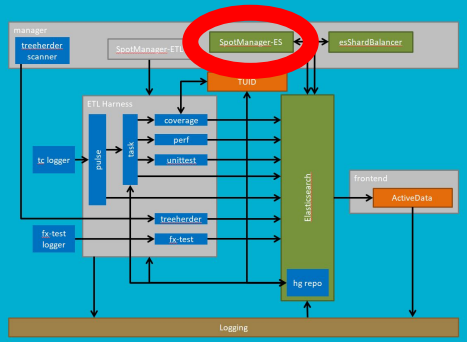

```
{"instance_type": "d2.8xlarge", "storage": 48000, "drives": {"$ref": "#8_ephemeral_drives"}, "discount": 0, "ecu": 116, "num_drives": 24, "memory": 244, "cpu": 36, "utility": 6
{"instance_type": "d2.xlarge", "storage": 6000, "drives": {"$ref": "#3_ephemeral_drives"}, "discount": 0, "ecu": 14, "num_drives": 3, "memory": 30.5, "cpu": 4, "utility": 30.5}
{"instance_type": "g2.2xlarge", "storage": 60, "drives": {"$ref": "#1_ephemeral_drives"}, "discount": 0, "ecu": 26, "num_drives": 1, "memory": 15, "cpu": 8, "utility": 1.2},
{"instance_type": "g2.8xlarge", "storage": 240, "drives": {"$ref": "#2_ephemeral_drives"}, "discount": 0, "ecu": 104, "num_drives": 2, "memory": 60, "cpu": 32, "utility": 4.8},
{"instance_type": "hi1.4xlarge", "storage": 2048, "drives": {"$ref": "#2_ephemeral_drives"}, "discount": 0, "ecu": 35, "num_drives": 2, "memory": 60.5, "cpu": 16, "utility": 40
{"instance_type": "hs1.8xlarge", "storage": 48000, "drives": {"$ref": "#8_ephemeral_drives"}, "discount": 0, "ecu": 35, "num_drives": 24, "memory": 117, "cpu": 16, "utility": 6
{"instance_type": "i2.2xlarge", "storage": 1600, "drives": {"$ref": "#2_ephemeral_drives"}, "discount": 0, "ecu": 27, "num_drives": 2, "memory": 61, "cpu": 8, "utility": 32.0},
{"instance_type": "i2.4xlarge", "storage": 3200, "drives": {"$ref": "#4_ephemeral_drives"}, "discount": 0, "ecu": 53, "num_drives": 4, "memory": 122, "cpu": 16, "utility": 60}
{"instance_type": "i2.8xlarge", "storage": 6400, "drives": {"$ref": "#8_ephemeral_drives"}, "discount": 0, "ecu": 104, "num_drives": 8, "memory": 244, "cpu": 32, "utility": 60}
```


more on SpotManager

ActiveData: SpotManager

ES machines - Current desired utility

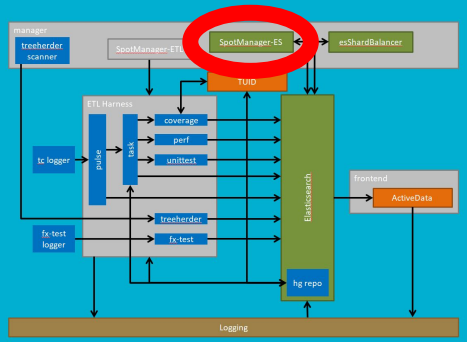
```
    def required_utility(self, current_utility=None):
        return self.minimum_utility
```



more on SpotManager

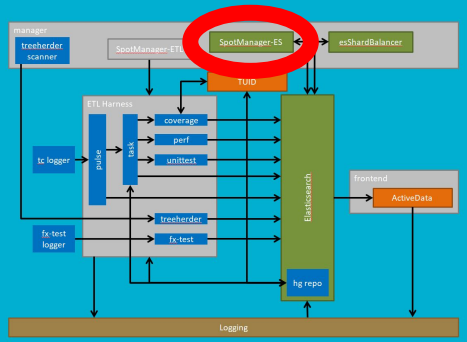
ActiveData: SpotManager

ES machines - Script to setup machine




```
# MOUNT AND FORMAT THE VOLUMES (list with `lsblk`)
for i, k in enumerate(volumes):
    if not conn.exists(k.path):
        # ENSURE DEVICE IS NOT MOUNTED
        conn.sudo('sudo umount '+k.device, warn=True)

        # (RE)PARTITION THE LOCAL DEVICE, AND FORMAT
        conn.sudo("parted " + k.device + " --script \"mklabel gpt mkpart primary ext4 2048s 100%\"")
        conn.sudo('yes | sudo mkfs -t ext4 '+k.device)

        # ES AND JOURNALLING DO NOT MIX
        conn.sudo('tune2fs -o journal_data_writeback '+k.device)
        conn.sudo('tune2fs -O ^has_journal '+k.device)

        # MOUNT IT
        conn.sudo('mkdir '+k.path)
        conn.sudo('sudo mount '+k.device+' '+k.path)
        conn.sudo('chown -R ec2-user:ec2-user '+k.path)

        # ADD TO /etc/fstab SO AROUND AFTER REBOOT
        conn.sudo("sed -i '$ a\\"+k.device+"    "+k.path+"       ext4    defaults,nofail  0   2' /etc/fstab")
```

ActiveData: SpotManager

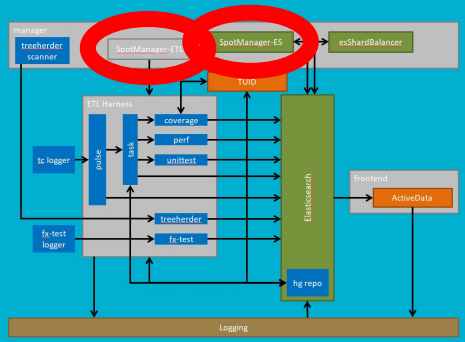


ES machines - Script to setup machine

First time user must setup drive partitions!

```
# MOUNT AND FORMAT THE VOLUMES (list with `lsblk`)
for i, k in enumerate(volumes):
    if not conn.exists(k.path):
        # ENSURE DEVICE IS NOT MOUNTED
        conn.sudo('sudo umount '+k.device, warn=True)

        # (RE)PARTITION THE LOCAL DEVICE, AND FORMAT
        conn.sudo("parted " + k.device + " --script \"mklabel gpt mkpart primary ext4 2048s 100%\"")
        conn.sudo('yes | sudo mkfs -t ext4 '+k.device)

        # ES AND JOURNALLING DO NOT MIX
        conn.sudo('tune2fs -o journal_data_writeback '+k.device)
        conn.sudo('tune2fs -O ^has_journal '+k.device)

        # MOUNT IT
        conn.sudo('mkdir '+k.path)
        conn.sudo('sudo mount '+k.device+' '+k.path)
        conn.sudo('chown -R ec2-user:ec2-user '+k.path)

        # ADD TO /etc/fstab SO AROUND AFTER REBOOT
        conn.sudo("sed -i '$ a\\"+k.device+"    "+k.path+"       ext4    defaults,nofail  0   2' /etc/fstab")
```

ActiveData: SpotManager

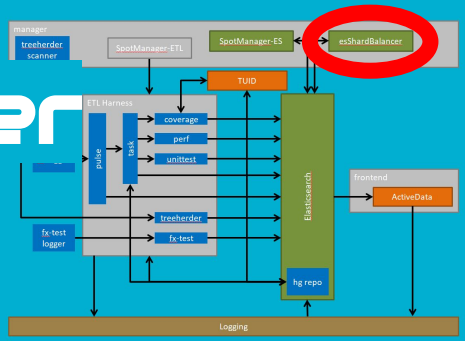


Future Work

- Add Test suite
- Use instance type datasource (not config files)

more on SpotManager

ActiveData: esShardBalancer



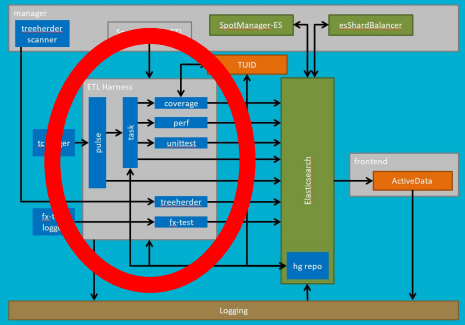
Better version of ElasticSearch's shard balance logic

- Balance query load
- Balance network load
- Prioritize redundancy (when nodes are lost)
- Handle diverse machine types
- Handle diverse index sizes
- Handle zones with different numbers of nodes

more on esShardBalancer

ActiveData: ETL

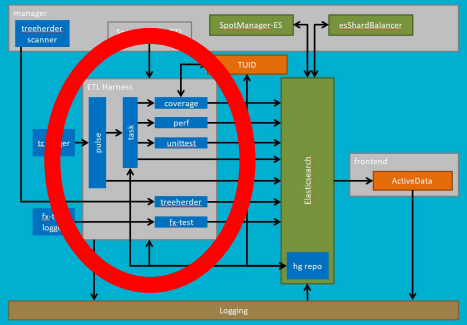
ETL - Extract Transform Load

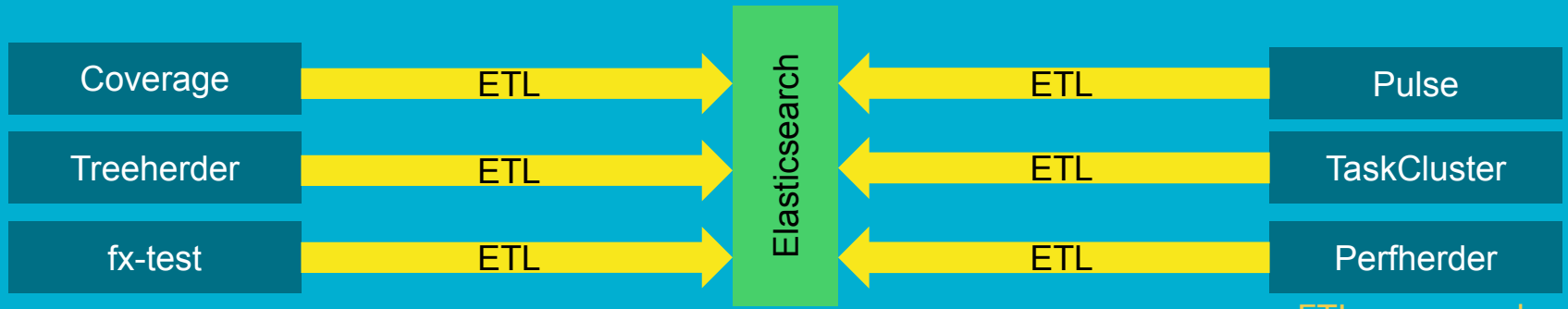


[ETL source code]

ActiveData: ETL

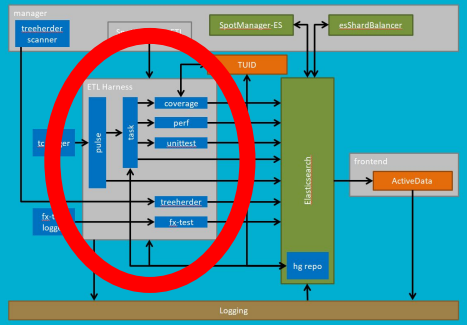
ETL - Extract Transform Load





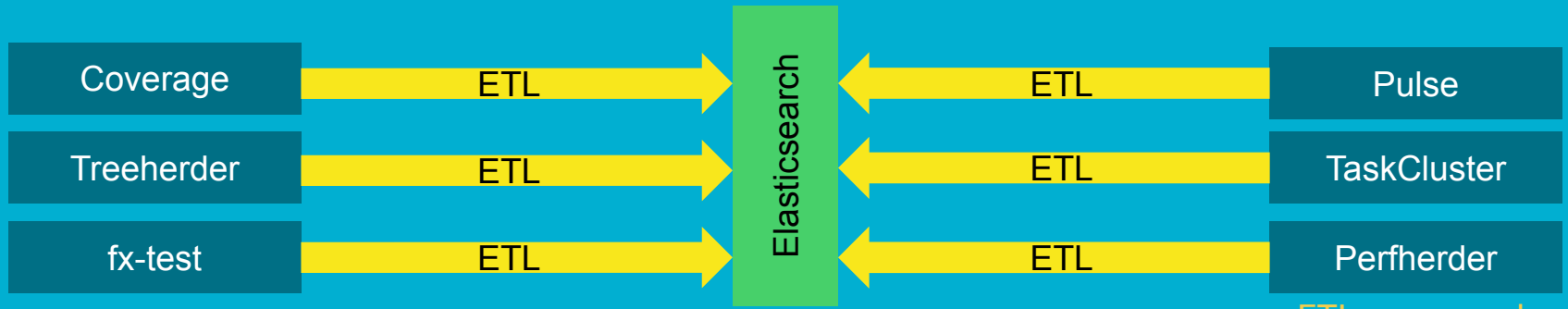
ETL source code

ActiveData: ETL



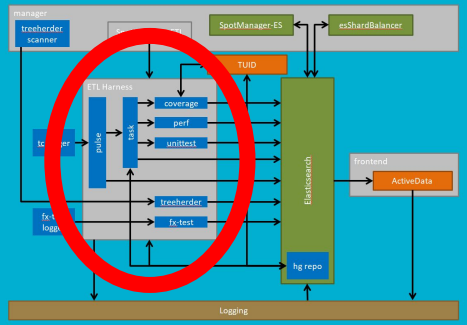
ETL - Extract Transform Load

- Centralize all data



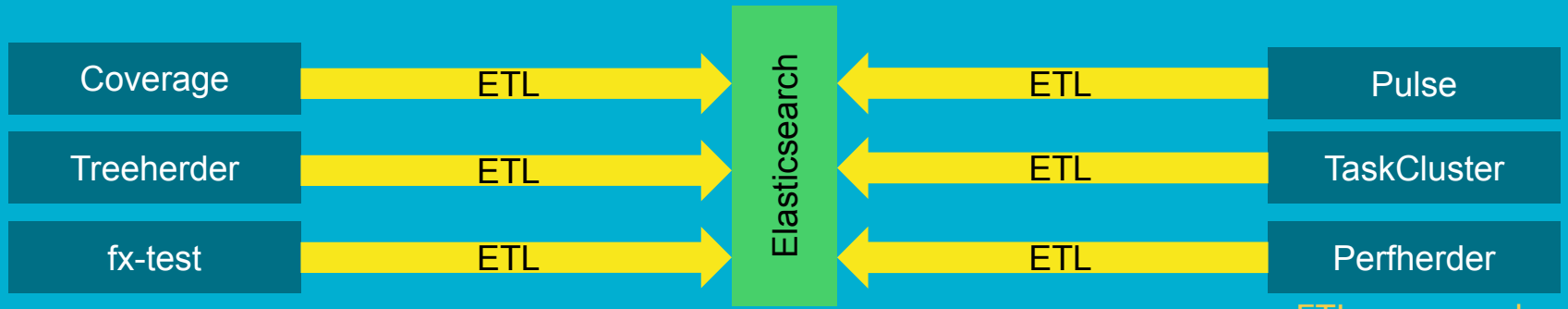
ETL source code

ActiveData: ETL



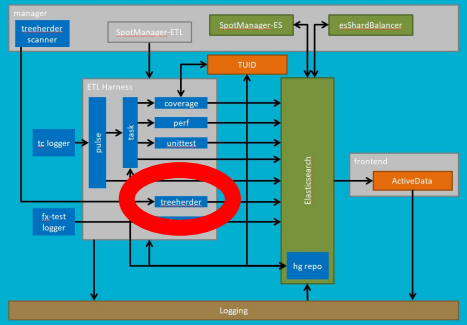
ETL - Extract Transform Load

- Centralize all data
- Decouple transactional from analytical



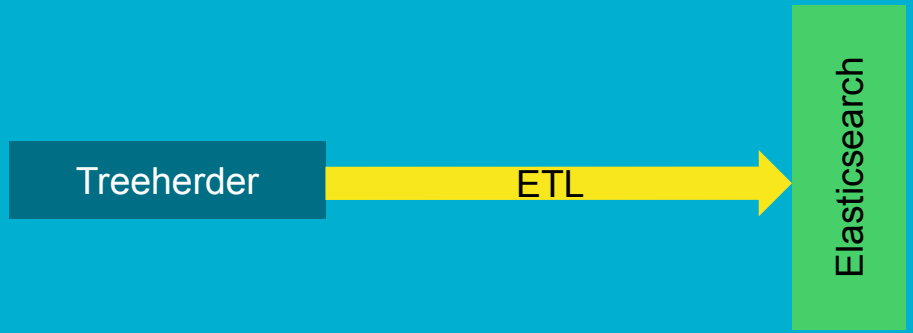
ETL source code

ActiveData: ETL



ETL - Extract Transform Load

- Centralize all data
- Decouple transactional from analytical

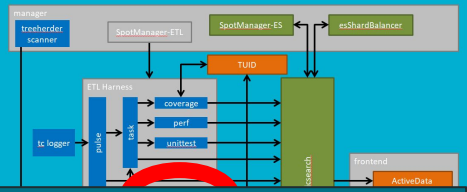


[MySQL-to-S3 (a denormalizer)]

ActiveData: ETL

ETL - Extract Transform Load

- Centralize all data
- Decouple transactional from analyt
- Pre-join data (denormalization)



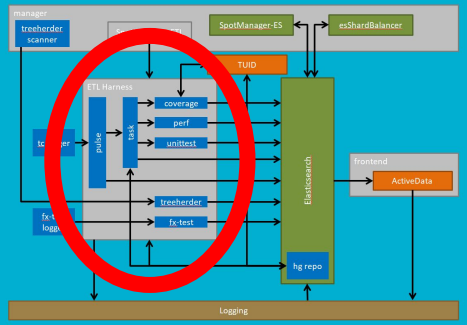

```
FROM job AS t1
LEFT JOIN option_collection AS t2 ON t2.option_collec...
LEFT JOIN failure_classification AS t3 ON t3.id = t1....
LEFT JOIN reference_data_signatures AS t4 ON t4.id = ...
LEFT JOIN build_platform AS t5 ON t5.id = t1.build_pl...
LEFT JOIN job_group AS t6 ON t6.id = t1.job_group_id
LEFT JOIN job_type AS t7 ON t7.id = t1.job_type_id
LEFT JOIN machine AS t8 ON t8.id = t1.machine_id
LEFT JOIN machine_platform AS t9 ON t9.id = t1.machin...
LEFT JOIN product AS t10 ON t10.id = t1.product_id
LEFT JOIN push AS t11 ON t11.id = t1.push_id
LEFT JOIN repository AS t12 ON t12.id = t1.repository_id
LEFT JOIN OPTION AS t13 ON t13.id = t2.option_id
LEFT JOIN repository AS t14 ON t14.id = t11.repositor...
```

MySQL-to-S3 (a denormalizer)

ActiveData: ETL

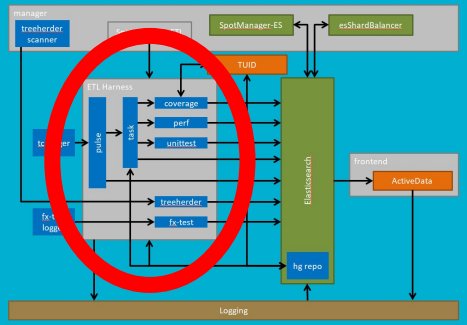
ETL - Extract Transform Load

- Centralize all data
- Decouple transactional from analytical
- Pre-join data (denormalization)
- Standardize data shape



ETL source code

ActiveData: ETL



ETL - Extract Transform Load

- Centralize all data
- Decouple transactional from analyt
- Pre-join data (denormalization)
- Standardize data shape

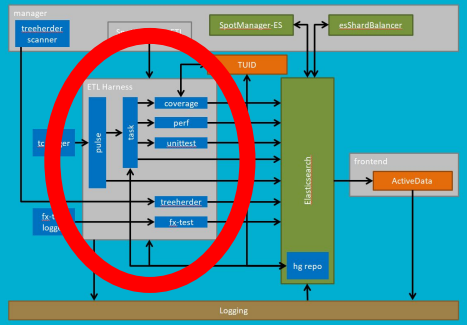
- No nulls (use missing properties)
- No 1-1 relations (use optional properties)
- 1-many are nested object arrays
- many-many demands a different granularity
- All properties are "indexed"
- Datatype does not matter

ETL source code

ActiveData: ETL

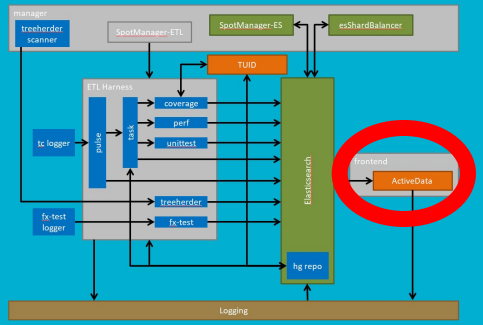
ETL - Extract Transform Load

- Centralize all data
- Decouple transactional from analytical
- Pre-join data (denormalization)
- Standardize data shape



ETL source code

ActiveData: Frontend



- Public facing service
- Simplify data model - everything is JSON
- Simple query language
- Consider whole-business needs...

ActiveData frontend source code

- Consider whole-business needs...

Enterprise Design Patterns

- Separate transactional from analytical
- Move code to data
- Machine compostability

Enterprise Design Patterns

Transactional vs Analytical

- RDBMS
- Normalized
- Daily business transactions
- Specific process information
- Fast insertion and updating
- Fast simple queries
- Data integrity important

- Data Warehouse
- Denormalized
- Analysis and prediction
- General business information
- Lagging ingestion, or batch insert
- Long running, complex queries
- Rely on sources' data

Enterprise Design Patterns

Transactional vs Analytical

- RDBMS
- Normalized
- Daily business
- Specific process information
- Fast insertion and updating
- Fast simple queries
- Data integrity important

Optimizing one, will hurt the other

- Data Warehouse
- Denormalized
- and prediction
- General business information
- Lagging ingestion, or batch insert
- Long running, complex queries
- Rely on sources' data

Enterprise Design Patterns

Move code to data

Enterprise Design Patterns

Move code to data (not data to the code)

Enterprise Design Patterns

Move code to data (not data to the code)

- Data locality is important to reduce latency
- Send queries to server, do not pull data to client
- Establish a set of primitive data operators (eg map(), filter(), and more)

Enterprise Design Patterns

Machine compostability (for sending code to data)

- Offload number crunching so clients require less CPU
- Promote automation that can compose requests
  - Simple
  - Orthogonal

Desired Features

Service

Data

Analysis

Data: Data Warehousing Problems

- No control of the type system(s)
- Many types of many different systems
- Types are changing often
- Processing unclean data

Data: Array Programming

All values are an array of values

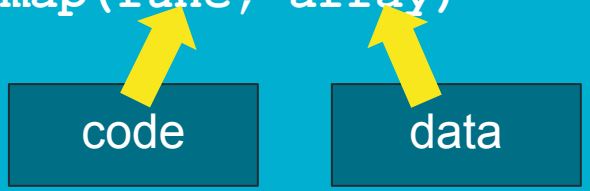
Encoding	Logical
NULL	[]
x	[x]
[x,y]	[x,y]

jQuery objects masquerade as arrays

Data: Array Programming

All values are an array of values

- Instead of passing data to the code: `func(x)`
- We apply the code over data: `map(func, array)`



Data: Array Programming

Property access is null safe

```
a = {}
x = a.b   # x is None
y = a.b.c # y is None
```

mo-dots
C# - null conditional operators
Lodash `.get()`
CoffeScript existential operator

Data: Array Programming

Property access is null safe

```
a = {}
x = a.b   # x is None
y = a.b.c # y is None
```

```
a = [{}]
x = [aa.get("b") for aa in a]
y = [xx.get("c") for xx in x]
```

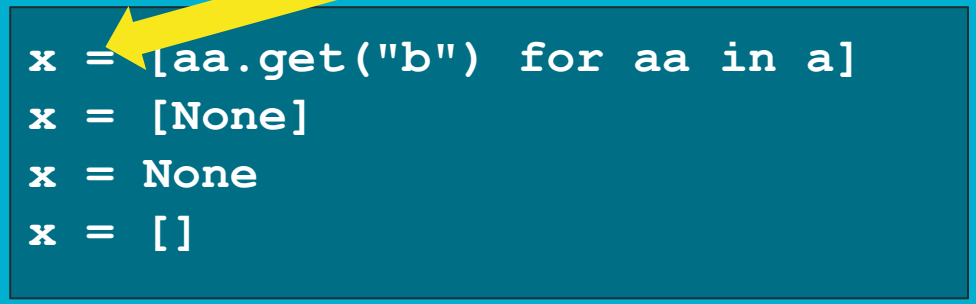
mo-dots
C# - null conditional operators
Lodash `.get()`
CoffeScript existential operator

Data: Array Programming

Property access is null safe

```
a = {}
x = a.b    # x is None
y = a.b.c # y is None
```

```
a = [{}]
x = [aa.get("b") for aa in a]
y = [xx.get("c") for xx in x]
```



mo-dots
C# - null conditional operators
Lodash `.get()`
CoffeScript existential operator

Data: Array Programming

Equality is set equality

eq	[]	[x]	[x, y]
[]	true	false	false
[x]	false	true	false
[x, y]	false	false	true

Data: Array Programming

Equality is set equality*

eq	[]	[x]	[x, y]
[]	true	false	false
[x]	false	true	false
[x, y]	false	false	true

* Not necessarily obvious: SQL `where NULL = NULL` is falsey

Data: Typed JSON

- Map from JSON to strict-typed systems
- Automated schema management
- Support automated migration
- Defer type management to query time

mo-json `typed_encoder.py`

Data: Typed JSON

All properties are marked with datatype

```
{"a": true  } -> {"a": {"~b~": true}}     # boolean
{"a": 1     } -> {"a": {"~n~": 1   }}     # number
{"a": "1"   } -> {"a": {"~s~": "1" }}     # string
{"a": {}    } -> {"a": {"~e~": 1   }}     # existence
{"a": [1, 2]} -> {"a": {"~N~": [          # Nested
                        {"~n~": 1},
                        {"~n~": 2},
                    ]}}
```

mo-json typed_encoder.py

Data: Typed JSON

All properties are marked with datatyp

Columnar storage means the data is not bigger, only the path names are longer:

`"a"` vs `"a.~N~.~n~"`

```
{"a": true  } -> {"a": {"~b~": true}}     # boolean
{"a": 1     } -> {"a": {"~n~": 1   }}     # number
{"a": "1"   } -> {"a": {"~s~": "1" }}     # string
{"a": {}    } -> {"a": {"~e~": 1   }}     # existence
{"a": [1, 2]} -> {"a": {"~N~": [          # Nested
                          {"~n~": 1},
                          {"~n~": 2},
                      ]}}
```

mo-json typed_encoder.py

Data: Typed JSON

All properties are marked with datatype

```
{"a": true  } -> {"a": {"~b~": true}}     # boolean
{"a": 1     } -> {"a": {"~n~": 1   }}     # number
{"a": "1"   } -> {"a": {"~s~": "1" }}     # string
{"a": {}    } -> {"a": {"~e~": 1   }}     # existence
{"a": [1, 2]} -> {"a": {"~N~": [          # Nested
                          {"~n~": 1},
                          {"~n~": 2},
                      ]}}
```

...well, except object existence, which is optional

mo-json `typed_encoder.py`

Data: Typed JSON

Any JSON is an object, and fits in a document store

```
[[1]]    ->  {"~N~": [
                  {"~N~": [
                      {"~n~": 1}
                  ]}
              ]}

"hello" ->  {"~s~": "hello"}
```

mo-json `typed_encoder.py`

Desired Features

Service

Data

Analysis

Analysis: JSON Expressions (JX)

Objective: Send code to data

- code is data, code is JSON
- work done by datastore, not by application

Additional Benefits

- Use in Javscript, Python, config files, and document stores
- Communicate rules between applications
- Simple enough so code can compound expressions
- Interpreters are easy to build

JSON expressions reference

Analysis: JSON Expressions (JX)

Objective: Send code to data

- code is data, code is JSON
- work done by datastore, not by application

Additional Benefits

- Use in Javscript, Python, config files, and document stores
- Communicate rules between applications
- Simple enough so code can compound expressi
- Interpreters are easy to build

JSON expressions reference

Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch

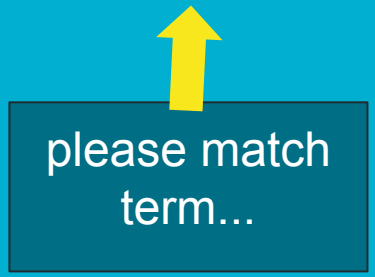
```
{"term":{"build.type":"opt"}}
```

JSON expressions reference

Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch

```
{"term":{"build.type":"opt"}}
```

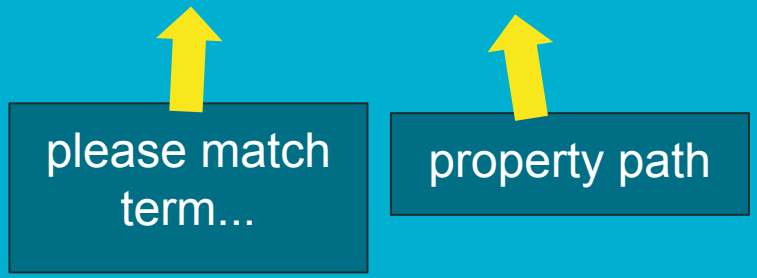


JSON expressions reference

Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch

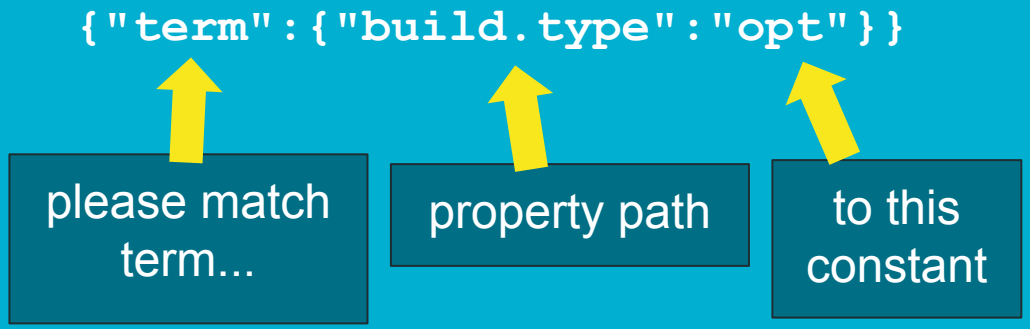
```
{"term":{"build.type":"opt"}}
```



JSON expressions reference

Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch



JSON expressions reference

Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch

```
{"term":{"build.type":"opt"}}
```

JSON Expression

```
{"eq":{"build.type":"opt"}}
```

JSON expressions reference

Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch

```
{"term":{"build.type":"opt"}}
```

JSON Expression

```
{"eq":{"build.type":"opt"}}
```

JSON expressions reference

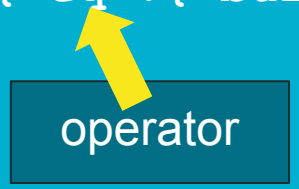
Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch

```
{"term":{"build.type":"opt"}}
```

JSON Expression

```
{"eq":{"build.type":"opt"}}
```



JSON expressions reference

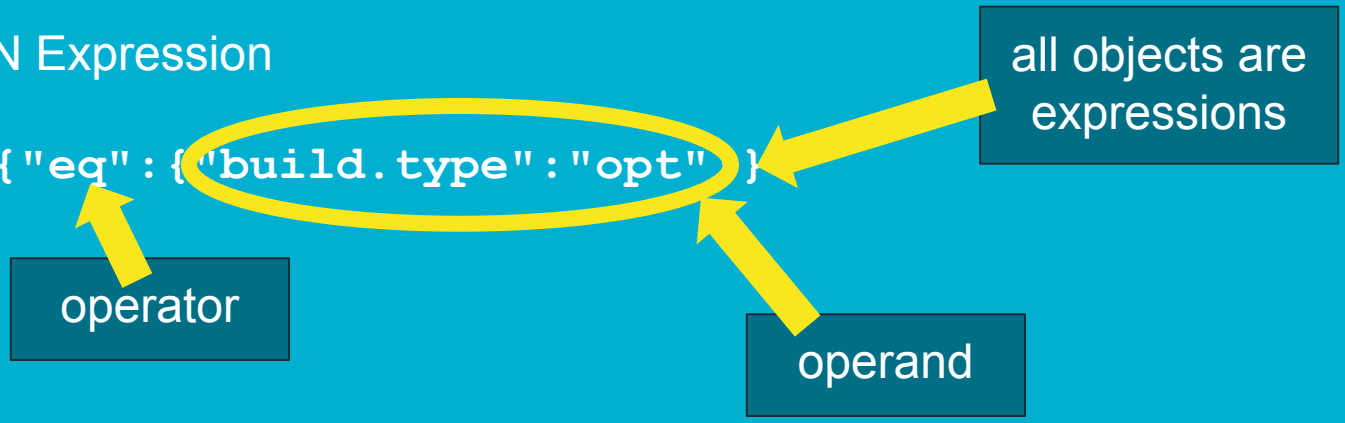
Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch

```
{"term":{"build.type":"opt"}}
```

JSON Expression

```
{"eq":{"build.type":"opt"}}
```



JSON expressions reference

Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch

```
{"term":{"build.type":"opt"}}
```

JSON Expression

```
{"eq":{"build.type":"opt"}}
```

… or more generally:

```
{"eq":["build.type", "some_variable"]}
```

JSON expressions reference

Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch

```
{"term":{"build.type":"opt"}}
```

JSON Expression

```
{"eq":{"build.type":"opt"}}
```

… or more generally:

```
{"eq":["build.type", "some_variable"]}
```

"array form" expects expressions

JSON expressions reference

Analysis: JSON Expressions (JX)

Most basic example, from Elasticsearch

```
{"term":{"build.type":"opt"}}
```

JSON Expression

```
{"eq":{"build.type":"opt"}}
```

… or more generally:

```
{"eq":["build.type", "some_variable"]}
```

"object form" allows constants

"array form" expects expressions

JSON expressions reference

Analysis: JSON Query Expressions

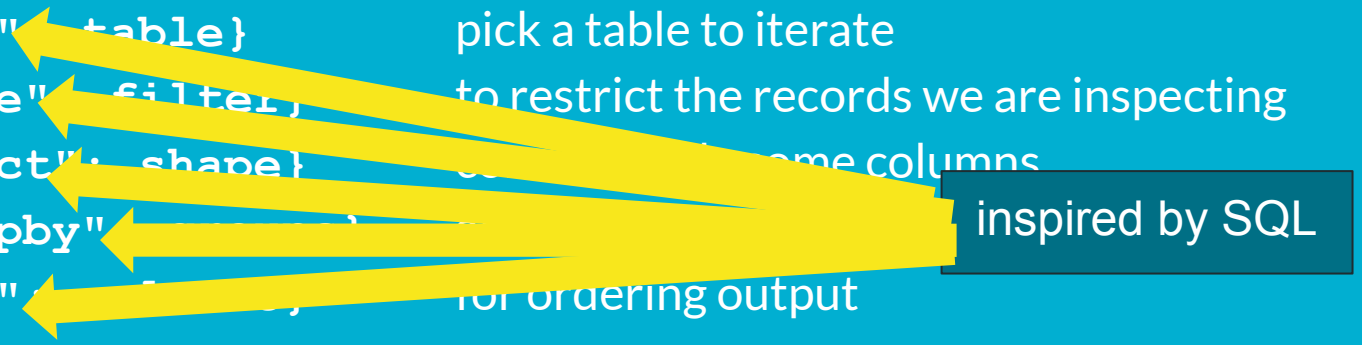
Project to define primitive operators on tables of data

- `{"from": table}` pick a table to iterate
- `{"where": filter}` to restrict the records we are inspecting
- `{"select": shape}` consider only some columns
- `{"groupby": groups}` aggregate
- `{"sort": columns}` for ordering output

Analysis: JSON Query Expressions

Project to define primitive operators on tables of data

- `{"from": table}` pick a table to iterate
- `{"where": filter}` to restrict the records we are inspecting
- `{"select": shape}` some columns
- `{"groupby": ...}`
- `{"sort": ...}` for ordering output



Analysis: JSON Query Expressions

No joins, no `explode()`

Spark explode()

Analysis: JSON Query Expressions

No joins, no `explode()`

```
{"example":[
     {"a": [
          {"b": 1}
          {"b": 2}
     ]}
]}
```

Analysis: JSON Query Expressions

No joins, no `explode()`

```
{"example":[
     {"a": [
          {"b": 1}
          {"b": 2}
     ]}
]}
```

```
{
    "from":"example.a",
    "select":"b"
}
```

Analysis: JSON Query Expressions

No joins, no `explode()`

```
{"example":[
     {"a": [
          {"b": 1}
          {"b": 2}
     ]}
]}
```

```
{
    "from":"example.a",
    "select":"b"
}
```

```
[1, 2]
```

Analysis: JSON Query Expressions

No joins, no `explode()`

```
{"example":[
     {"a": [
          {"b": 1}
          {"b": 2}
     ]}
]}
```

```
{
    "from":"example.a",
    "select":"b"
}
```

dot-delimited path to nested object array is just-another-table

```
[1, 2]
```

Analysis: JSON Query Expressions

datastores are logically JSON objects

No joins, no **explode()**

```
{"example":[
     {"a": [
          {"b": 1}
          {"b": 2}
     ]}
]}
```

```
{
    "from":"example.a",
    "select":"b"
}
```

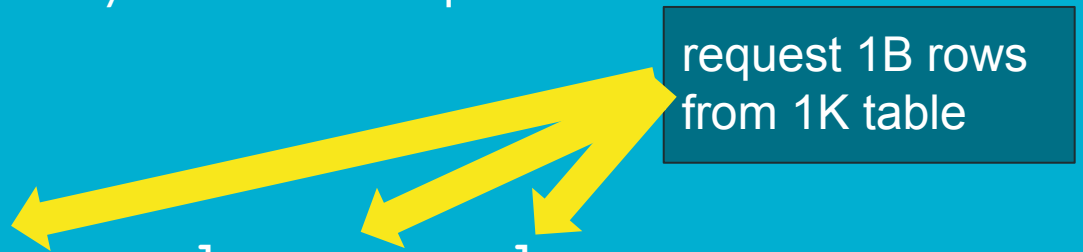
```
[1, 2]
```

Data: Why not SQL?

Data: Why not SQL?

Accepts computationally unbounded requests

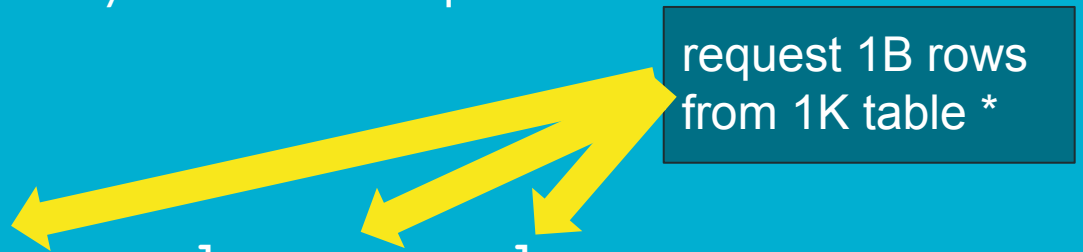
```
SELECT
    *
FROM
    employees, employees, employees
GROUP BY
    continent
```



Data: Why not SQL?

Accepts computationally unbounded requests

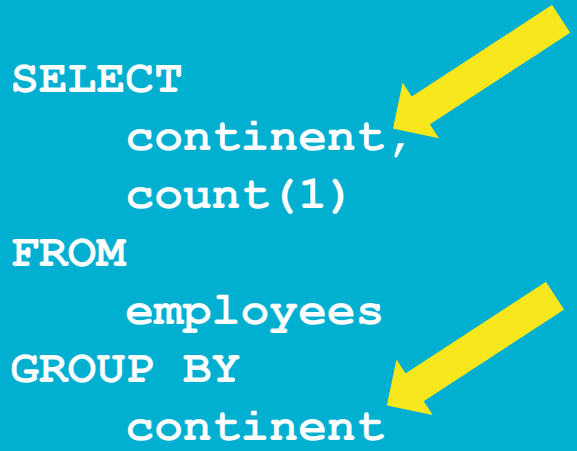
```
SELECT
    *
FROM
    employees, employees, employees
GROUP BY
    continent
```



* see CONNECT BY how to make it worse

Data: Why not SQL?

Redundant clause definition



```
SELECT
    continent,
    count(1)
FROM
    employees
GROUP BY
    continent
```

Data: Why not SQL?

```
SELECT
    continent,
    count(1)
FROM
    employees
GROUP BY
    continent
```

Data: Why not SQL?

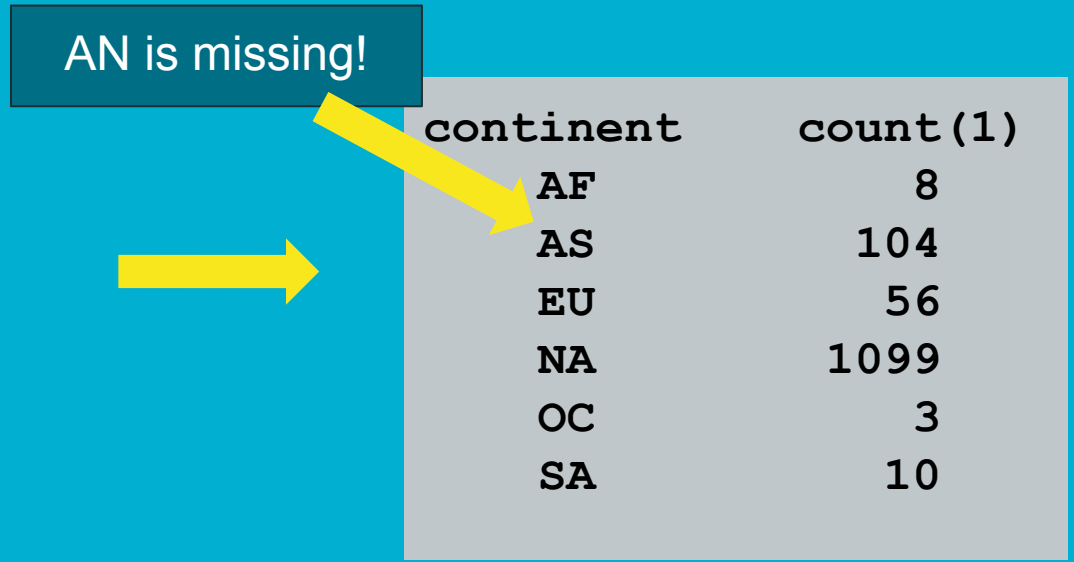
```
SELECT
    continent,
    count(1)
FROM
    employees
GROUP BY
    continent
```

continent	count(1)
AF	8
AS	104
EU	56
NA	1099
OC	3
SA	10

Data: Why not SQL?

Missing domains

```
SELECT
     continent,
     count(1)
FROM
     employees
GROUP BY
     continent
```



Data: Why not SQL?

```
SELECT
    c.name AS continent,
    count(1)
FROM
    employees e
RIGHT JOIN
    continents c ON c.code=e.continent
GROUP BY
    c.name
```

Data: Why not SQL?

```
SELECT
    c.name AS continent,
    count(1)
FROM
    employees e
RIGHT JOIN
    continents c ON c.code=e.c
GROUP BY
    c.name
```

continent	count(1)
Africa	8
Antarctica	NULL
Asia	208
N. America	1099
Oceania	3
S. America	10

Data: Why not SQL?

Mismatched domains

AN exists!

```
SELECT
    c.name AS continent,
    count(1)
FROM
    employees e
RIGHT JOIN
    continents c ON c.code=e.c
GROUP BY
    c.name
```

continent	count(1)
Africa	8
Antarctica	NULL
Asia	208
N. America	1099
Oceania	3
S. America	10

Data: Why not SQL?

Mismatching Joins

```
SELECT
    c.name AS continent,
    count(1)
FROM
    employees e
RIGHT JOIN
    continents c ON c.code=e.c
GROUP BY
    c.name
```

EU is missing!

continent	count(1)
Africa	8
Antarctica	NULL
Asia	208
N. America	1099
Oceania	3
S. America	10

Data: Why not SQL?

Double counting

```
SELECT
    c.name AS continent,
    count(1)
FROM
    employees e
RIGHT JOIN
    continents c ON c.code=e.c
GROUP BY
    c.name
```

Asia is double counted!

continen	count(1)
Africa	8
Antarctica	NULL
Asia	208
N. America	1099
Oceania	3
S. America	10

Data: Why not SQL?

Missing values

```
SELECT
    c.name AS continent,
    count(1)
FROM
    employees e
RIGHT JOIN
    continents c ON c.code=e.c
GROUP BY
    c.name
```

Should be zero

continent	count(1)
Africa	8
Antarctica	NULL
Asia	208
N. America	1099
Oceania	3
S. America	10

Data: Why not SQL?

Sensitive to messy data

```
SELECT
    *
FROM
    continents
```

Duplicate*

code	name
AF	Africa
AN	Antarctica
AS	Asia
AS	Asia
NA	N. America
OC	Oceania
SA	S. America

Data: MDX

MultiDimensional eXpressions

```
SELECT
     [continent] ON ROWS
     [Count]
FROM
     [employee]
```

more on MDX

Data: MDX

Domain aware

```
SELECT
     [continent] ON ROWS
     [Count]
FROM
     [employee]
```

continent	count(1)
Africa	8
Antarctica	0
Asia	104
Europe	56
N. America	1099
Oceania	3
S. America	10

more on MDX

Analysis: JSON Query Expressions

Project to define primitive operators on tables of data

- `{"from": table}` pick a table to iterate
- `{"where": filter}` to restrict the records we are inspecting
- `{"select": shape}` consider only some columns
- `{"groupby": groups}` aggregate
- `{"sort": columns}` for ordering output

Analysis: JSON Query Expressions

Project to define primitive operators on tables of data

- `{"from": table}` pick a table to iterate
- `{"where": filter}` to restrict the records we are inspecting
- `{"select": shape}` consider only some columns
- `{"groupby": groups}` aggregate
- `{"sort": columns}` for ordering output
- `{"edges": domains}` domain-aware grouping

Analysis: JSON Query Expressions

Project to define primitive operators on tables of data

- `{"from": table}` pick a table to iterate
- `{"where": filter}` to restrict the records we are inspecting
- `{"select": shape}` consider only some columns
- `{"groupby": groups}` agg
- `{"sort": columns}`
- `{"edges": domains}` do

- include whole domain
- count everything
- ...and only once

SQL has GROUP BY CUBE()

Analysis: JSON Query Expressions

Project to define primitive operators on tables of data

- `{"from": table}` pick a table to iterate
- `{"where": filter}` to restrict the records we are inspecting
- `{"select": shape}` consider only some columns
- `{"groupby": groups}` aggregate
- `{"sort": columns}` for ordering `output`
- **`{"edges": domains}`** domain-aware grouping

Analysis: JSON Query Expressions

Project to define primitive operators on tables of data

- `{"from": table}` pick a table to iterate
- `{"where": filter}` to restrict the records we are inspecting
- `{"select": shape}` consider only some columns
- `{"groupby": groups}` aggregate
- `{"sort": columns}` for ordering `output`
- `{"edges": domains}` domain-aware grouping
- **`{"window": sub-query}`** add aggregate to each record

Analysis: JSON Query Expressions

Implementations

- jx-elasticsearch - core of the ActiveData service
- jx-python - for processing data in Python
- jx-sqlite - pour JSON documents into a well structured database, use JX to query them out again

Analysis: JSON Query Expressions

Future work? Take advantage of large number of hot machines

- Compound queries - perform further computation on results before returning them to the client
- Machine learning primitive operators?
  - a large amount of data
  - a large number of features
  - can the number crunching be split among machines?
  - what do the operators look like?
  - minimum first step is in a student project

Future: ActiveData?

Future: Google BigQuery?

Database offering: $5/terabyte scanned (+ almost-free storage)

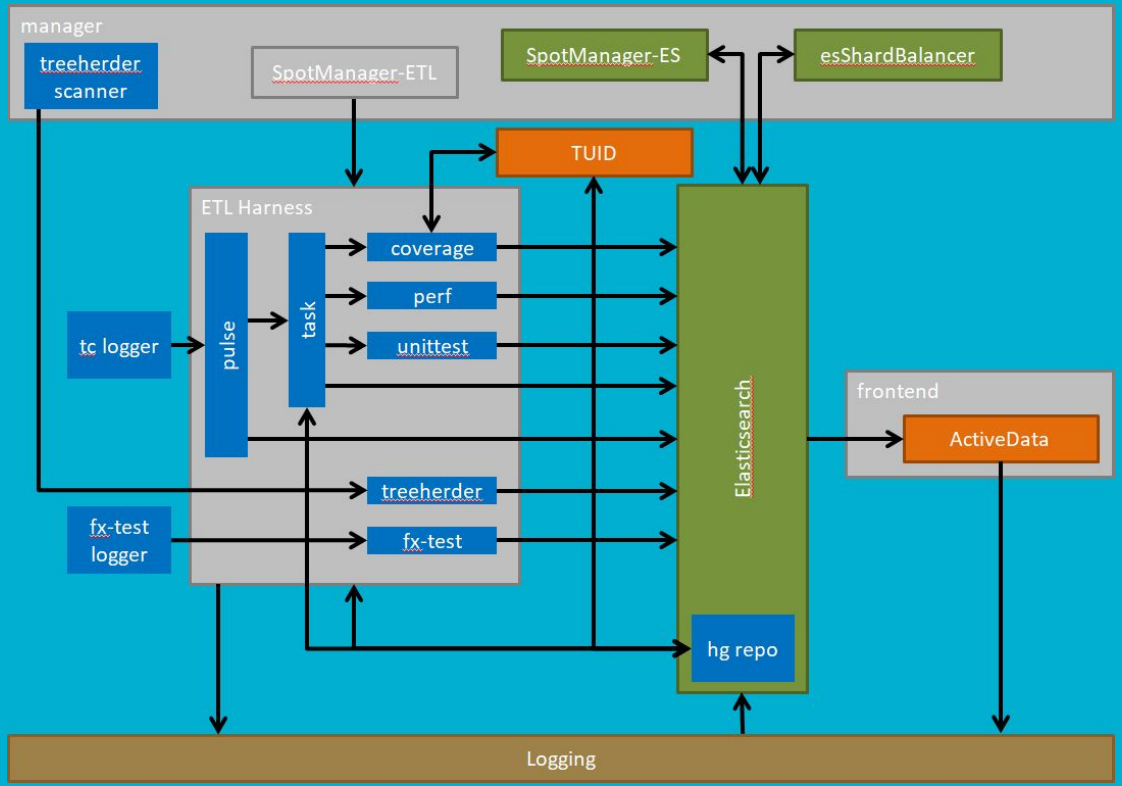
- 3rd party service
- centralized destination for data
- offset query load from transactional systems
- standard query language (SQL)

Future: Google BigQuery?

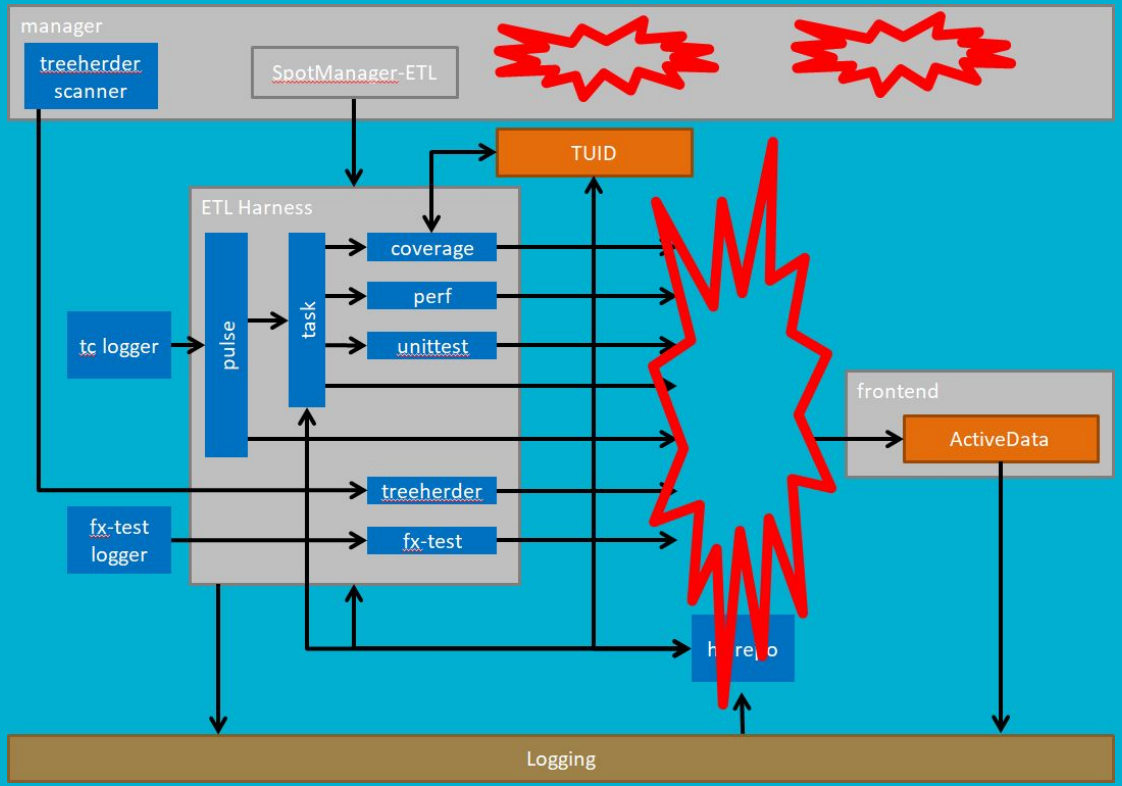
Problems

- query latency?
- how expensive is it in practice?
- Human-managed schema
- SQL is not suited for queries over unclean data
- SQL Computationally unbounded
- ETL ingestion from various sources

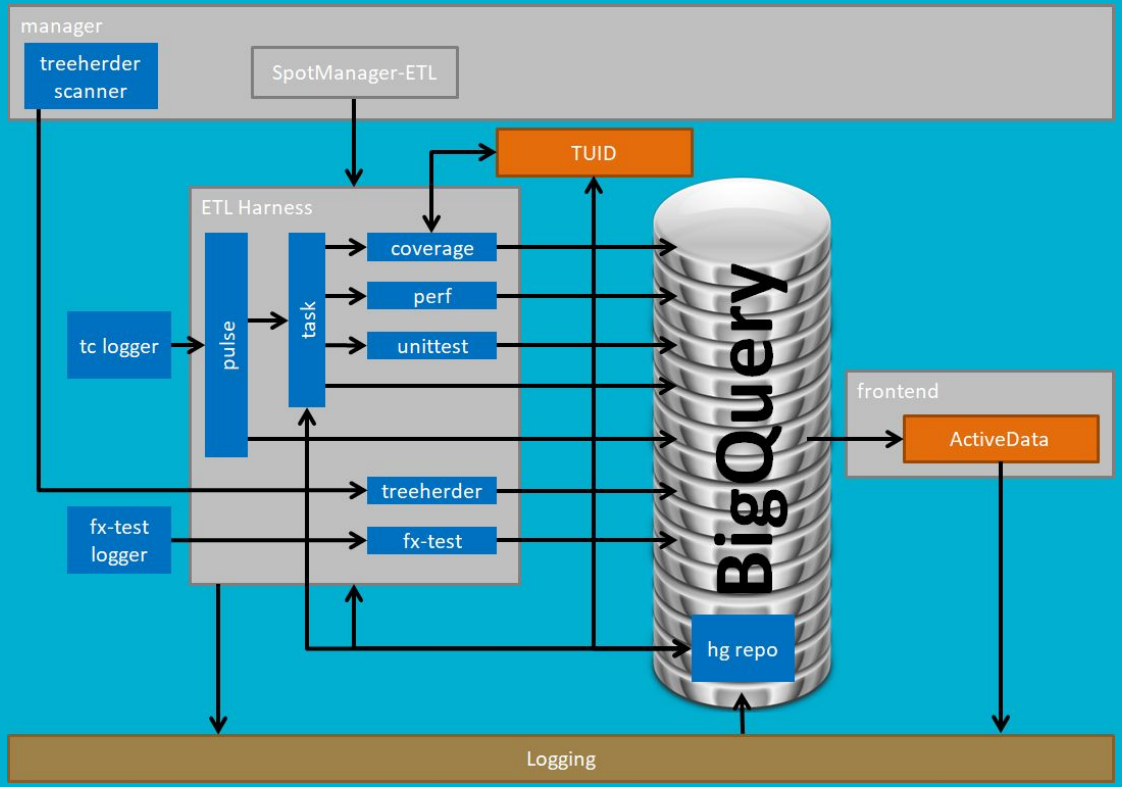
Future: Google BigQuery?



Future: Google BigQuery?



Future: Google BigQuery?



ActiveData

Components and Capabilities