
TECH
University
ebay
CHINA CENTER
OF EXCELLENCE
elastic

CONTENTS

Agenda

D1

	Agenda	Speaker
AM	Overview of ES & Pronto & ES Basics	Hongsi
PM	ES Basic Ops and Monitoring	Hongsi
	Search & ES in Advance (1)	Peter

D2

	Agenda	Speaker
AM	Search & ES in Advance (2)	Peter
PM	ELKB & Workshop	Hongsi
	Use case sharing / Pronto Tools	Peter

1

Elasticsearch Basics

Session 1 Elasticsearch Basics

- Elasticsearch & Pronto Overview
- Components of Elasticsearch
- CRUD of Elasticsearch
- Distribution of an Index

A search engine that prevails in databases

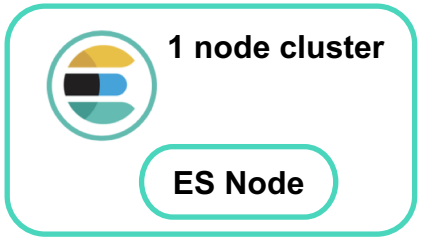
Rank			DBMS	Database Model	Score		
Aug 2018	Jul 2018	Aug 2017			Aug 2018	Jul 2018	Aug 2017
1.	1.	1.	Oracle	Relational DBMS	1312.02	+34.24	-55.85
2.	2.	2.	MySQL	Relational DBMS	1206.81	+10.74	-133.49
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1072.65	+19.24	-152.82
4.	4.	4.	PostgreSQL	Relational DBMS	417.50	+11.69	+47.74
5.	5.	5.	MongoDB	Document store	350.98	+0.65	+20.48
6.	6.	6.	DB2	Relational DBMS	181.84	-4.36	-15.62
7.	7.	↑ 9.	Redis	Key-value store	138.58	-1.34	+16.68
8.	8.	↑ 10.	Elasticsearch	Search engine	138.12	+1.90	+20.47
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	129.10	-3.48	+2.07
10.	10.	↓ 8.	Cassandra	Wide column store	119.58	-1.48	-7.14

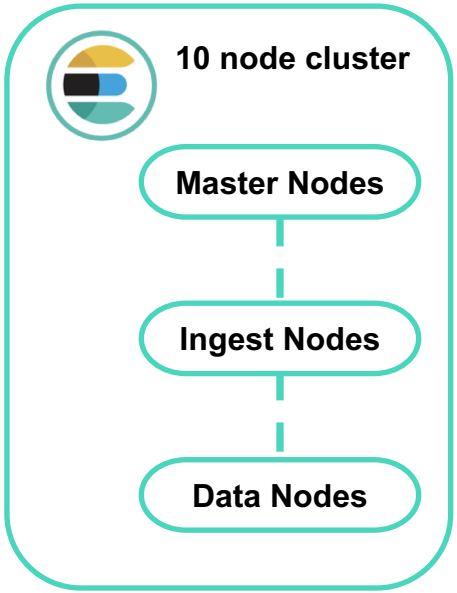
https://db-engines.com/en/ranking

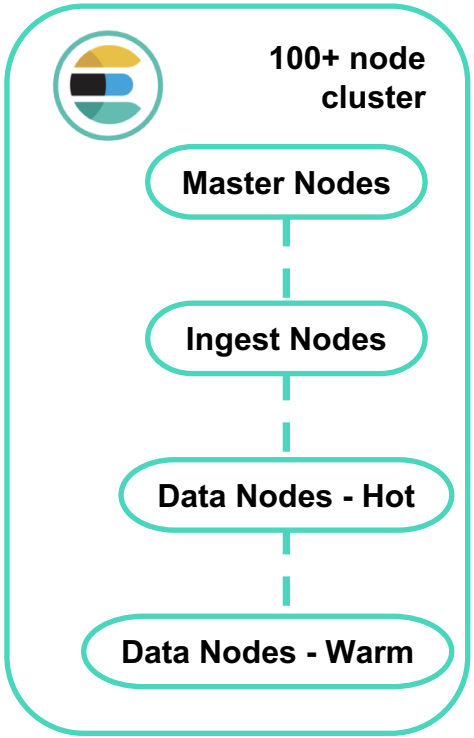
The Birth of Elasticsearch

- In 2004,Shay Banon developed a product called **Compass**
  - Built on top of Lucene, Shay's goal was to have search
- integrated into Java applications as simply as possible
- The need for ***scalability*** became a top priority
- In 2010, Shay completely rewrote Compass with two main objectives:
  1. ***distributed from the ground up in its design***
  2. ***easily used by any other programming language***
- Today Elasticsearch is the most popular enterprise search engine

Distributed Search







A ***node*** is an instance of Elasticsearch

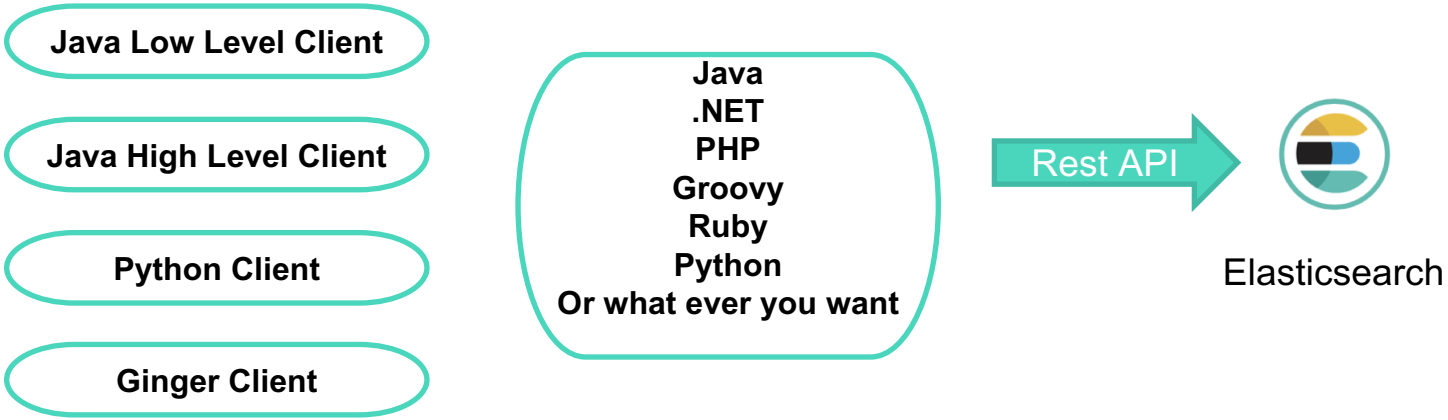
A ***cluster*** is a collection of Elasticsearch nodes

Your cluster can grow as your needs grow

Easily Used by Other Languages

Elasticsearch provides REST APIs for communicating with a cluster over HTTP

- Allows client applications to be written in any language



Use cases of Elasticsearch

- Search
  - GitHub
    - 15 billion lines codes
  - StackOverflow
    - 40 million articles
  - Wikipedia
    - 40 million articles

- Mornitor & Analyze
  - Sprint
    Driving mission-critical business decisions by analyzing over 200 dashboards covering 3 billion events a day
  - citibank
    - Monitoring application infrastructure across a major financial institution

Pronto use cases

- Search
  - SEO: 300k search per minute
- Log & Analysis
  - 7 clusters, ~ 500 data nodes
  - > 10 billion log events, > 10TB data volume every day
- Near Realtime Aggregation & Reporting
  - NGRL: 200 data nodes

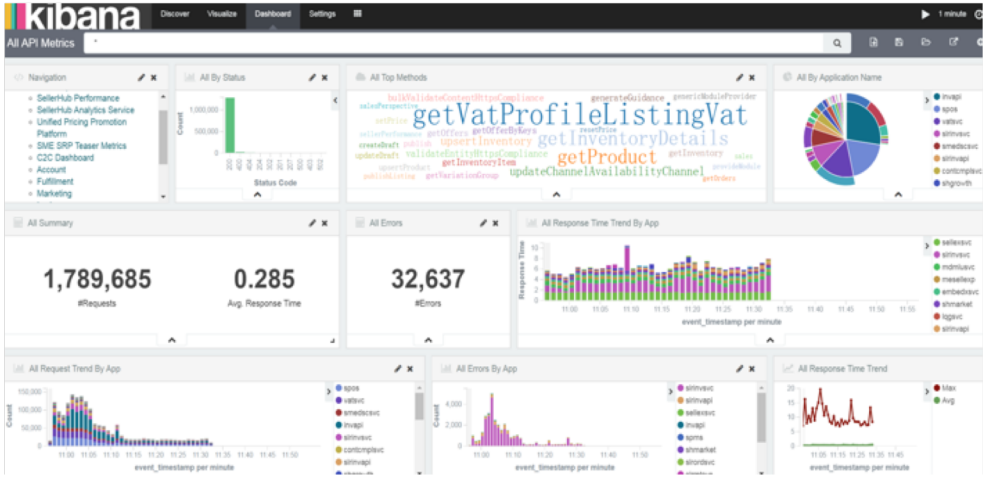
Selling team near real-time operational and behavioral data monitoring and analysis

Business Value

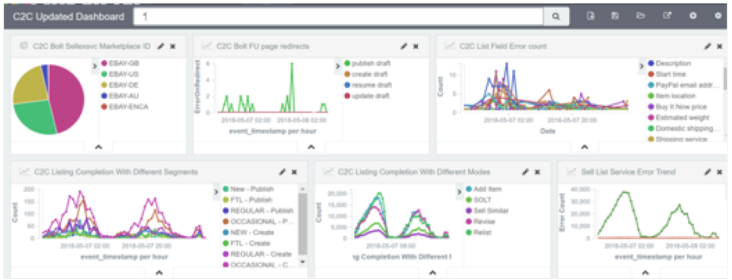
- Near-real time troubleshooting/triage
- Near-real time service stack analysis
- Near-real time business metrics monitoring and analysis

For more than 30+ services

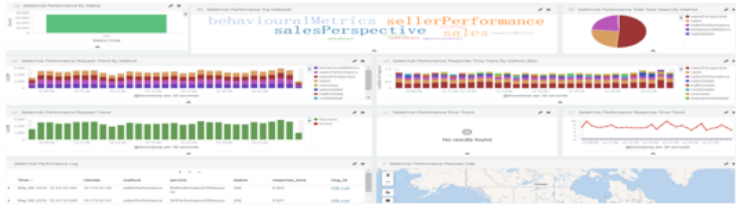
Overview Dashboard



c2c dashboard



SellerHub Performance dashboard



Pricing Dashboard

ebay CHINA CENTER OF EXCELLENCE

TECH University

What is Elasticsearch, and what is not

1. Elasticsearch is a distributed, RESTful search and real time analytics engine

2. Elasticsearch is not designed for data storage

3. Elasticsearch is not RDBMS or KV

4. Not as single source of truth data

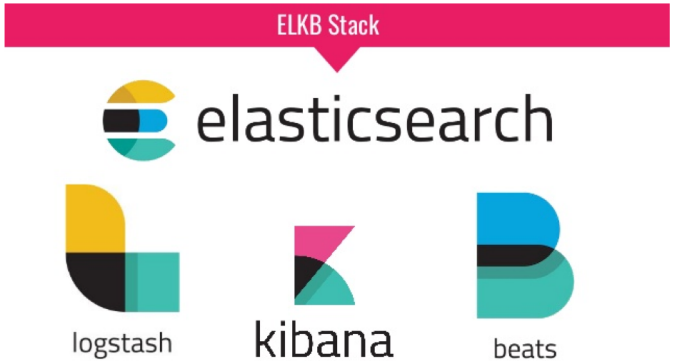
Elasticsearch History

- **0.4:** first version was released in February, 2010
- **1.0:** released in January, 2014
- **2.0:** released in October, 2015
- **5.0:** released in October, 2016
- **6.0**: released in October, 2017

Why the big jump in version number?

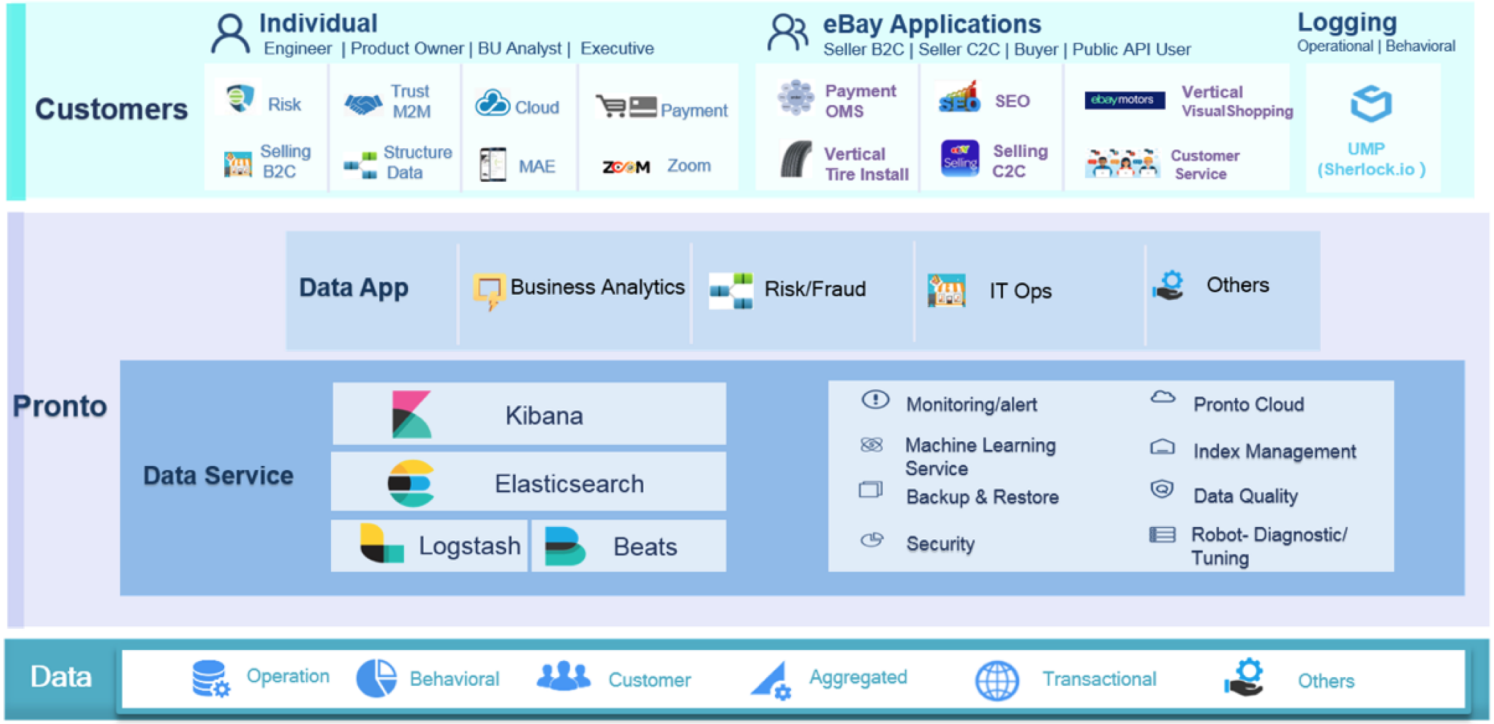
The Elastic Stack...

ELKB Stack

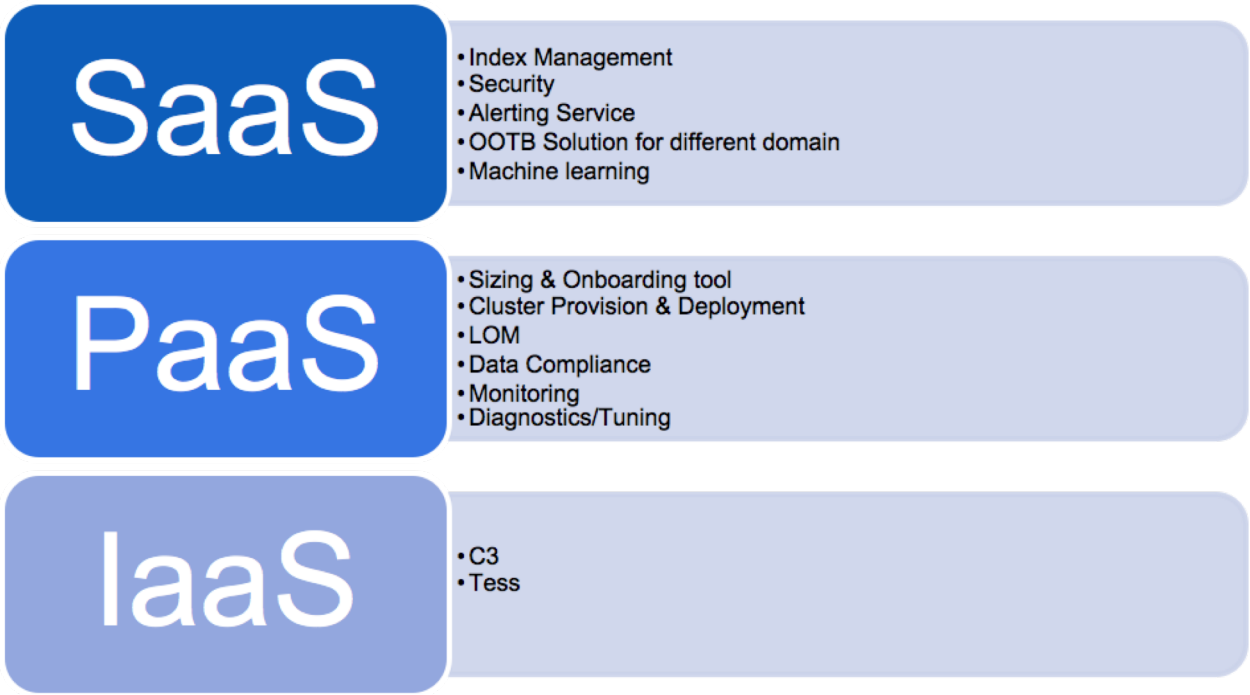


- Security
- Alerting
- Monitoring
- Reporting
- Graph
- Machine Learning

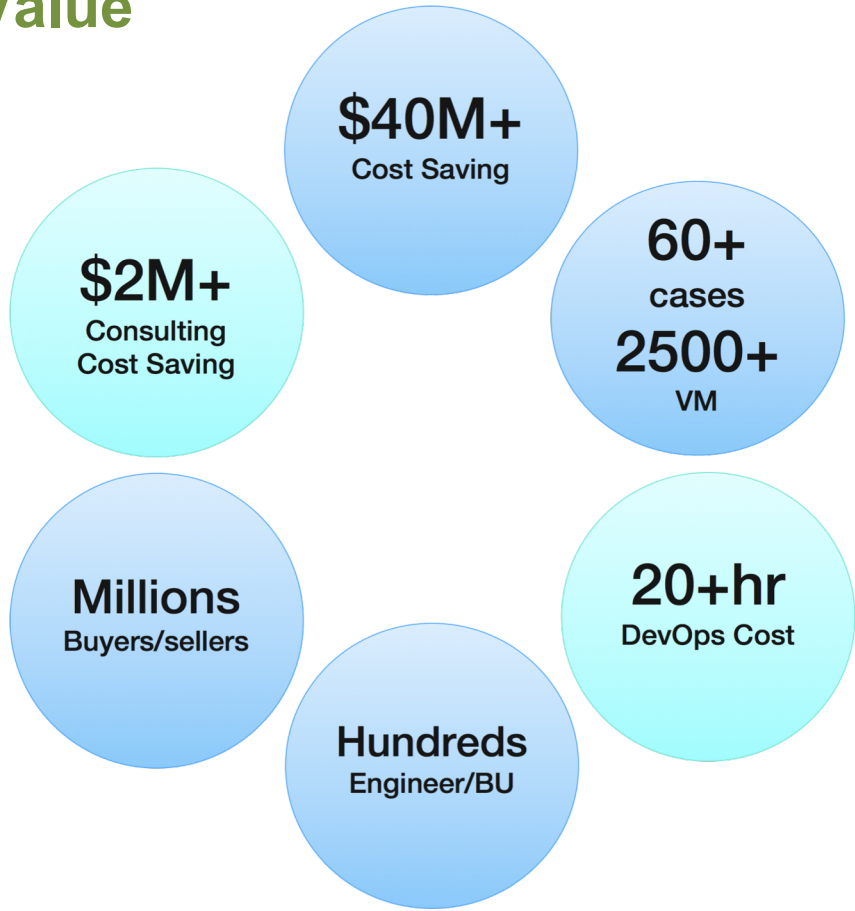
ELKB in eBay



Pronto



Business Value



Session 1 Elasticsearch Basics

- Elasticsearch Overview
- **Components of Elasticsearch**
- CRUD of Elasticsearch
- Distribution of an Index

The Components of Elasticsearch

– ***node***: an instance of Elasticsearch

– ***cluster***: one or more nodes that work together to serve the same data and API requests

– ***document***: a piece of data that you want to search

– ***index***: a collection of documents that have somewhat similar

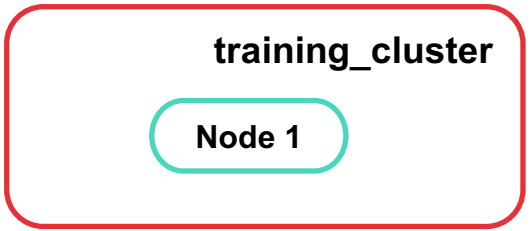
– ***shard***: is a worker unit that holds data and can be assigned to nodes

Cluster

- Everything in Elasticsearch happens in the context of a ***cluster***
  - Every cluster has a name (e.g. ***training_cluster***)
  - Default cluster name is "**elasticsearch**"
  - Every node belongs to a cluster

- You can set the **cluster.name** that a node belongs to in **elasticsearch.yml** or on the command line

```
bin/elasticsearch -E node.name=node1 -E cluster.name=training_cluster
```



What is a node?

- A ***node*** is an instance of Elasticsearch
  - It is a java process that runs in a JVM

- Every node has a name
  - you can set **node.name** in the **elasticsearch.yml** file
  - or on the command line

- Each node has a UID assigned at startup
  - And persisted in the data folder for future restarts

Node Type

- Master Node
  - a master node is a node eligible to be elected as master
  - The one and only master can publish cluster status

- Data Node
  - When **node1** starts, it is a data node by default
  - You can disable by setting **node.data: false**

- Coordinating Node
  - The node that handles requests
  - Every node is a coordinating node when it receives request.
  - It inspects the request and determines what needs to be done – It makes sure the request is executed and sends the response
  - When a node is not master/data it's dedicated coordinate node

Document

A ***document*** can be any text or numeric data you want to search and/or analyze
For example:

- an entry in a log file
- a comment made by a customer on your website
- the weather details from a weather station at a specific moment

Specifically, a ***document*** is a top-level object that is serialized into JSON and stored in Elasticsearch

Every document has a ***unique ID*** – which either you provide
- or Elasticsearch generates for you

Documents must be JSON objects

- Numbers, Strings, Booleans
- Is Array / Nested JSON Supported?
  - –Yes
- Can we have infinite fields?
  - –No, big metadata is dangerous

Definition of an index

An ***index*** in Elasticsearch is a ***logical*** way of grouping data:

- An index contains ***mappings*** that define the documents' field names and data types of the index
- an index is a ***logical namespace*** that maps to where its contents are stored in the cluster

There are two different concepts in this definition:

- an index has some type of data schema mechanism
- an index has some type of mechanism to distribute data across a cluster

An unsuitable analogy

Relational DB	Elasticsearch
Database	Cluster
Table	Index
Row	Document
Column	Field
Schema	Mapping
SQL	Query DSL

- In 6.0 & later, type is deprecated gradually
- No Trasaction, thus no ACID, no roll back
- Although enhanced greatly in reliability, ES is not recommended to be treated as single source of truth

You can have a lot of indices

- Indices are fairly light weight in Elasticsearch, so it is not unusual to create lots of them

- For example, you might define a new index every day for searching logs
- Using multiple indices allows you to organize your data ***logically***:
  - **logs-2018-08-01**
  - **logs-2018-08-02**
  - **logs-2018-08-03**
- **Question: can we create infinite indices?**

Session 1 Elasticsearch Basics

- Elasticsearch Overview
- Components of Elasticsearch
- **CRUD of Elasticsearch**
- Distribution of an Index

CRUD of Elasticsearch

- Indexing
- Get / Simple Search
- Update
- Delete

Test Cluster

- http://go/test_cluster
- http://go/test_cluster_kibana

Define an Index

- Clients communicate with a cluster using Elasticsearch's REST APIs
  - There is a collection of ***Document APIs*** for working with indexes and documents

- An index is defined using the **Create Index API**, which can be accomplished with a simple **PUT** command:

```
curl -XPUT '[endpoint]/comments' -H "Content-Type: application/json"
```

- Let's create a new index named "[yourname]_**comments**":

Index a Document


```
curl -XPUT ‘[endpoint]/honhu_comments/_doc/1' -i -H "Content-Type: application/json" -d '
{
    "username": ”honhu",
    ”item title": ”iPhoneX 256GB",
    "comment": ”I love it",
    "rating": 4
}'
```

- Index / type will be created automatically if it doesn’t exist, even it’s not defined
- If the id not exists, the new document will be created. Otherwise, it’s reindexed.
    - Full doc overwrite, not update by field
- Use **_create API** if you want to make sure the id doesn’t exist.

Indexing without specifying an ID

HTTP/1.1 201 **Created**
Location: /comments/_doc/Mh-IeV4Bhv5TOUuTrQX3 content-type: application/json; charset=UTF-8
content-length: 220


```
{
    "_index": "comments",
    "_type": "doc",
    "_id": "Mh-IeV4Bhv5TOUuTrQX3",
    "_version": 1,
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0 },
    "_seq_no": 0,
    "_primary_term": 1
}
```

Retrieve a document

- Use **GET** to retrieve an indexed document
  - Returns 200 if the document is found
  - Returns a 404 error if the document is not found

```
curl -XGET] '[endpoint]/comments/_doc/1' -H "Content-Type: application/json"
```

- Use **_source API** to get document content only

```
curl -XGET '[endpoint]/comments/_doc/1/_source' -H "Content-Type: application/json"
```

Multi get api

- The ***Multi Get API*** allows you to GET multiple documents in a single request
  - Avoid multiple round trips

```
curl -XPOST '[endpoint]/_mget' -H "Content-Type: application/json" -d '
{
    "docs" : [
        {
            "_index" : "INDEX_NAME1",
            ["_type": "TYPE1",]
            "_id" : "ID1"
        }, {
            "_index" : "INDEX_NAME2",
            ["_type": "TYPE2",]
            "_id" : "ID2"
        }
]}'
```

Update a Document

- Use Index API to overwrite
- Use _update API to update fields

```
POST my_index/_doc/2/_update
{
   "doc" : {
      "comment" : ”Good quality."
   }
 }
```

- There’s no in-place update in ES

Delete

- Delete a document

```
DELETE my_index/_doc/2
```

- Delete an index

```
DELETE my_index
```

- There’s no rollback in ES, be very careful to implement deletions.

The bulk API

- The ***Bulk API*** makes it possible to perform many write operations in a single API call, greatly increases the indexing speed
  - useful if you need to index a data stream such as log events, which can be queued up and indexed in batches of hundreds or thousands

- Four actions: **create**, **index**, **update** and **delete**

- The response is a large JSON structure with the individual results of each action that was performed
  - The failure of a single action does not affect the remaining actions

Example of _bulk

- **_bulk** has a unique syntax based on lines of commands:
  - Each command appears on a single line

```
curl -XPOST '[endpoint]:9200/_bulk' -H "Content-Type: application/x-ndjson" -i -d '
{"index" : {"_index":"comments", "_type":"doc", "_id":4}}
{"username": ”scott", ”item": ”Surface Pro 3 I7","comment": ”a lot of problems", "rating": 2}
{"index" : {"_index":"comments", "_type":"doc", "_id":5}}
{"username": ”honhu", ”item": ”Dyson V8 Vacuum","comment": ”The best vacuum I’ve ever used",
"rating": 5}
{"update" : {"_index":"comments", "_type":"doc", "_id":1}}
{"doc": {"rating": 5}}
{"delete": {"_index":"my_index", "_type":"doc", "_id":4}}
'
```

Each operation specifies the index, type and id

Followed by a document on a single line, except for ‘delete’

The final \n is necessary

Bulk Error Messages

ISSUE	REASON	ACTION
Unable to connect	networking issue or cluster down	retry until connection is available (round robin across nodes)
5xx error	internal server error in Elasticsearch	retry until successful (using round robin)
4xx error	invalid JSON or incorrect request structure	fix bad request before retrying
429 error	Elasticsearch is too busy	retry (ideally with linear or exponential backoff)
Connection unexpectedly closed	Node died or network issue	Retry (with resk of creating a duplicate document)

Session 1 Elasticsearch Basics

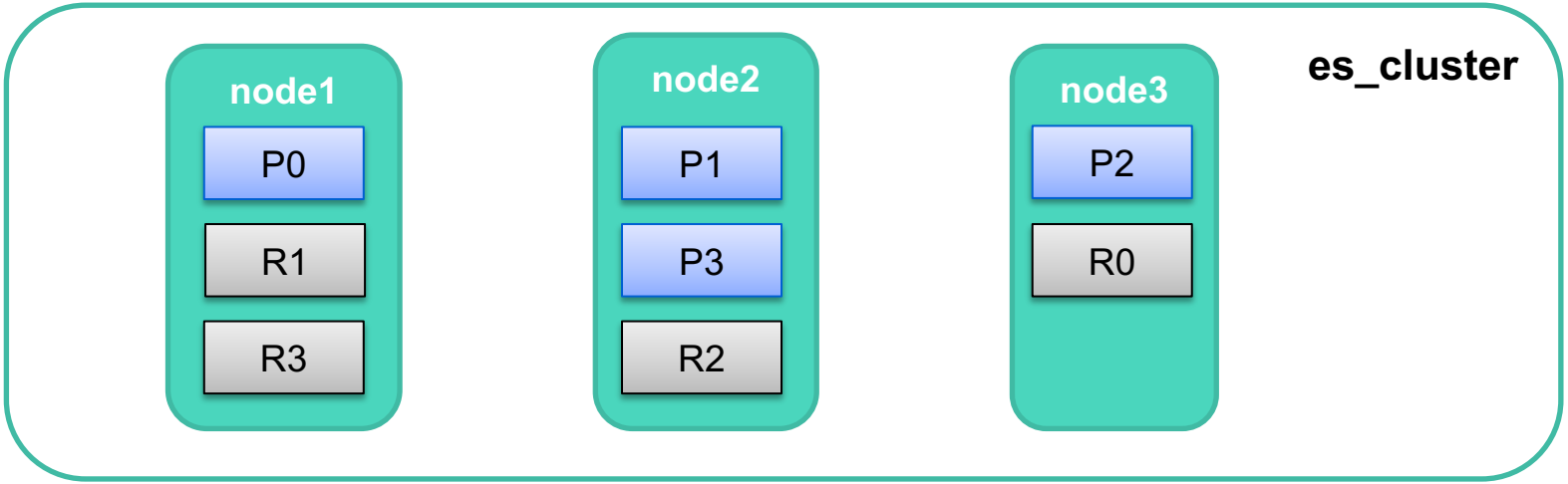
- Elasticsearch & Pronto Overview
- Components of Elasticsearch
- CRUD of Elasticsearch
- **Distribution of an Index**

Shards: Distribution of an Index

- **Shard:**
  - A shard is a worker unit that holds data and can be assigned to nodes
  - An index is split into shards before any documents are indexed
  - The default number of shards for an index is 5
  - Every shard is an standalone index of Lucene
  - The elected master node decides which shards to allocate to which nodes (published via cluster state)
  - Documents are routed to shards using the following formula:
    - » **Shard = hash(_routing) % number_of_primary_shards**
    - » default **_routing** value is document id

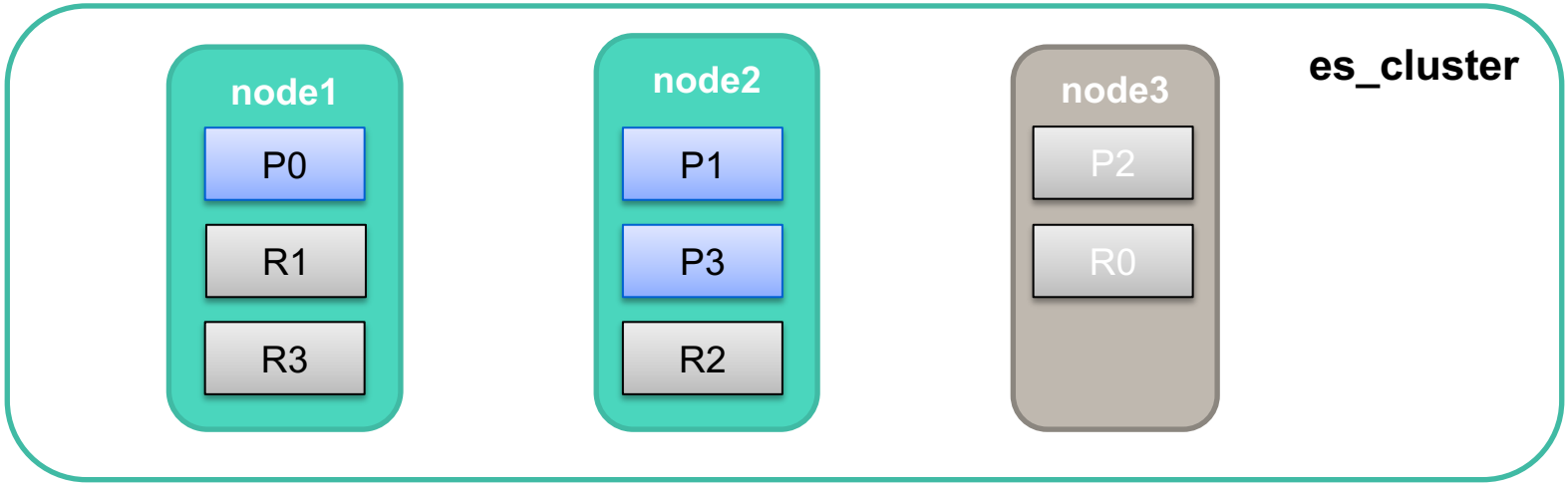
Primary vs Replica

- Two types of shards
  - **primary:** the original shards of an index
    - » Write throughput. Reallocation.
  - **replicas**: copies of the primary
    - » HA, Read throughput



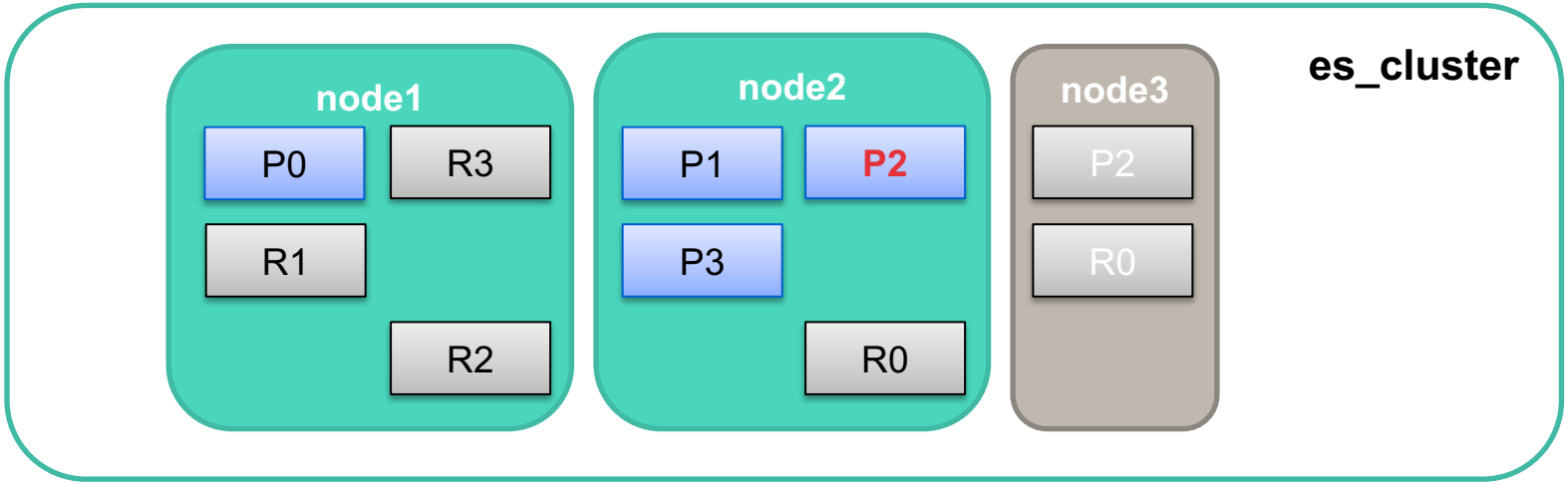
HA of data

- When node3 fails



HA of data (2)

- Promotion of replicas
- Replicating unassigned replicas
- Shards of same ID will **never** be allocated to the same node



Routing

- Manually specify how to distribute the docs to shards

```
POST profile/_doc?routing=hongsi
{
    "user" : "hongsi",
    "favourite_food" : "fried noodles"
}
```

- Useful to optimize performance by organizing data likely to be fetched together

```
GET profile/_doc/222?routing=hongsi
```

- But be careful, data may get imbalanced, thus load got imbalanced

Segment

- Basic Storage Unit, a Lucene inverse index
- Shards are actually stored in segments
- Why ES doesn’t like update & delete?

Why ES doesn't like update / delete

- No roll back
- Performance will be impacted
  - segments become sparse
  - merge is IO intensive
- Avoid frequent update / delete, and use

```
POST {index}/_forcemerge
```

when load is low

Session 1 Review

- **Elasticsearch & Pronto Overview**
- **Components of Elasticsearch**
  - cluster, node, index, shard, document
- **CRUD of Elasticsearch**
  - index, get, update/overwrite, delete
- **Distribution of an Index**
  - shard mechanism, segment

QUIZ & QA

1. Can you **PUT** a document into an index without specifying an id?

2. What is the only valid type for an index created in ES 6.x?

3. Suppose you have a cluster with 10 data nodes, you created a new index with ‘PUT /play’, then insert 1,000,000 documents. How many total shards will the index has?

4. Suppose you have a cluster with 5 data nodes, what’s the max replica number you can set?

3

Elasticsearch Ops Basics

Session 4 Elasticsearch Ops Basics

- Elasticsearch Settings
- Health & Statistics API of Elasticsearch
- Monitoring & Alerting, Diagnostic Tool of Pronto

Settings of Elasticsearch

- Settings are configured in Elasticsearch at three levels
  - Index
  - Node
  - Cluster

- ***static*** settings can only be set in **elasticsearch.yml** or from the command line
- ***dynamic*** settings can be updated on a live cluster using the ***Cluster Update API***
  - Index level setting can only be set dynamically

Index Settings

- Index settings can be set on creation or be updated
  - On Creation

```
PUT my_index
{
  "settings": {
    "number_of_shards": 10
  }
}
```

  - Update on existing index

```
PUT my_index_*/_settings
{
  "number_of_replicas": 3
}
```

Wildcard updating is available

Configuring shards of an index

- You can change the "***number_of_replicas***" at any time using the ***_settings*** endpoint (**dynamic**)
- You can **NOT** change the number of primary shards after an index is created, so choose wisely! (**fixed**)
- But sometimes, we do need to change shard number, for various reasons.
  - Reindex
  - Some index manage mechanism

All Settings

- This setting prevent index reassigned too soon when node need maintenance.
- Note that _all or * won’t apply on new indices after set.

```
PUT _all/_settings
{
    "settings": {
       "index.unassigned.node_left.delayed_timeout": ”35m"
     }
}
```

Node Settings

- ***Node settings*** include:
  - file paths, labels, network interfaces (settings specific to the node)
  - typically configured in **elasticsearch.yml** or command line

- Settings in **elasticsearch.yml** have 2 styles
  - **Flat**

```
cluster.name: my_cluster_name
```

  - **Indented**

```
cluster:
    name: my_cluster_name
```

A glance of elasticsearch.yml

- **cluster.name**
  - defaults to “**elasticsearch**”
  - ES node use it to identify if another node belongs to the same cluster
  - So it’s recommended to give it a unique & meaningful name
  - A typical pronto cluster name is [poc]es[ver][team][usage]-[az], e.g.: es6umpdiscovery-lvs02

- **node.name**
  - Default to first 7-characters of a UUID
  - Recommended to be set as hostname

- **node.attr**
  - User defined attributes, can be very useful if you want to extend es functionalities

Node Roles

- **Master eligible**
- **Data**
- **Ingest**
- **Coordinating Only**
- **Machine Learning**
- **Tribe (deprecated)**

  – Nodes can take on multiple roles at the same time
  – or they can be ***dedicated*** nodes that only take on a single role

Node Role Configurations

- **node.master**
  - Only if True, the node is eligible to be elected as master
- **node.data**
  - Only if True, the node is allowed to have shards allocated in it.
- **node.ingest**
  - Only if True, the node is allowed to use pipeline to pre-process document indexed
- **node.ml**
- All settings above are default to **TRUE**

Master Nodes

- The ***master node*** is responsible for:
  - cluster-wide actions such as creating or deleting an index
  - tracking which nodes are part of the cluster
  - deciding which shards to allocate to which nodes

- An ES Cluster can have multiple master eligible nodes, but only one master node.
  - Which needs a mechanism to elect master, and ensure there's one working master.

- It's a good practice to have multiple, dedicated master eligible nodes

Discovery

- Discovery is the action a node takes to discover other nodes within a cluster
- Default Discovery method is **zen**
- With ***zen discovery***, a node issues ping requests to find other nodes on the same network.
  - start with a list of hosts to act as *gossip routers,* configured with **discovery.zen.ping.unicast.hosts**

```
discovery.zen.ping.unicast.hosts: [‘node1’, ‘node2’]
```

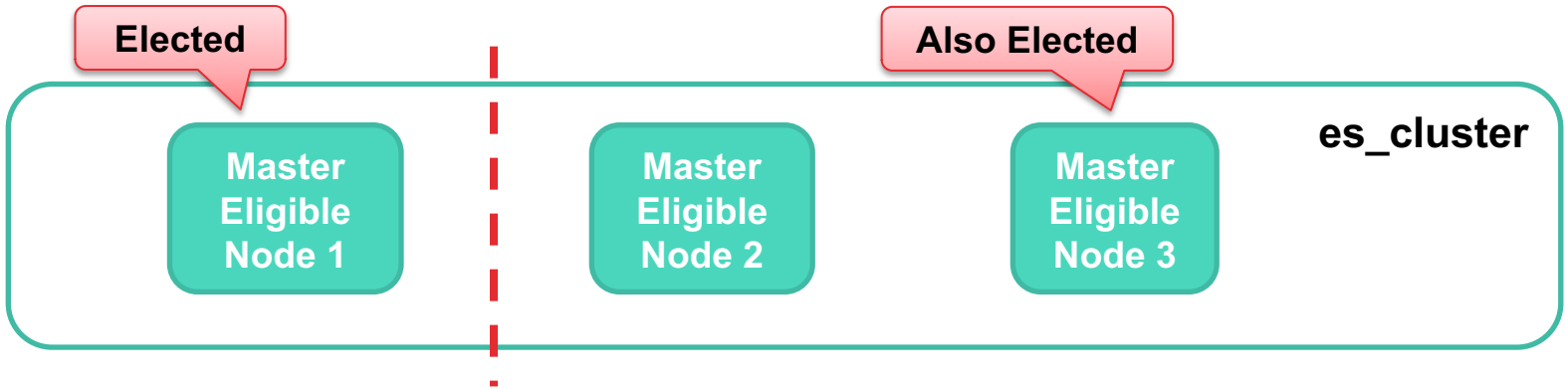
  - If another node is reachable, and have same cluster.name, current node will join the cluster.
  - Recommended to set as the list of dedicated master eligible nodes.

Master Election

- Master election is a part of ping process
- Always elect the node with lowest node id
- Once a master is elected, other master eligible nodes can still join the cluster, but not as the master
- When a elected master node fails, other master eligible node will use the same method to elect.

Split Brain

- Even though the master node is the single source of truth, when it comes to the cluster state, problems can still happen
  - In particular, a big concern is if the cluster gets ***partitioned***
  - Multiple masters could be elected, which is called "***split brain***"
  - Split brain can lead to inconsistency.



Avoiding Split Brain

- A master-eligible node needs at least **minimum_master_nodes** votes to win an election
  - Setting it to a quorum helps avoid the split brain scenario

- It's recommended to set

  minimum_master_nodes = [num of master eligible node]/2+1

- Pronto cluster always have 3 dedicated master eligible nodes, thus

```
discovery.zen.minimum_master_nodes: 2
```

Dedicated Master Nodes

- ***Dedicated master nodes*** are less likely to have JVM heap and memory issues:
  - they are not holding any shards (data)
  - they are not handling all the API connections/requests
  - they are not involved with high-frequency index and search operations, so they are not a scaling bottleneck

```
node.master: true
node.data: false
node.ingest: false
```

Data Nodes

- Data node is allowed to hold shards

- So, all data-related requests put load on some data nodes eventually
  - If you need more storage capacity, or more computing power, increase data nodes
  - Dedicated data node will help to let then focus on processing data & client requests

```
node.master: false
node.data: true
node.ingest: false
```

Coordinating Nodes

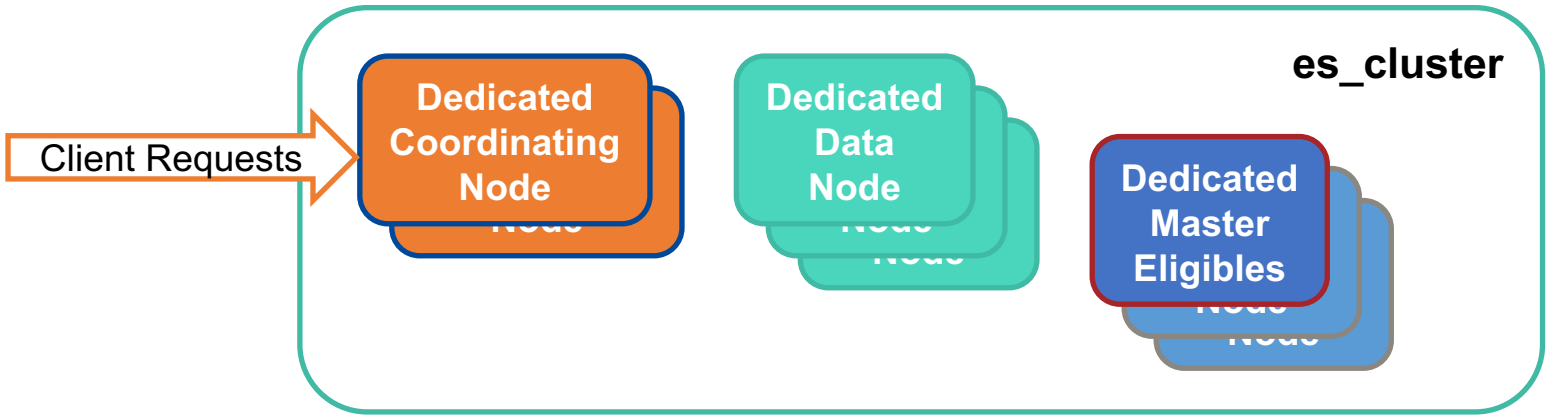
- A ***coordinating node*** is the node that receives a specific client request
  - every node is implicitly a coordinating node
  - forwards the request to other relevant nodes, then combines those results into a single result
  - So, client node will have high load in aggregation intensive cases

- Set dedicated coordinating node

```
node.master: false
node.data: false
node.ingest: false
```

Dedicated Coordinating Nodes

- behave like smart load balancers
- perform the gather/reduce phase of search queries
- lightens the load on data nodes

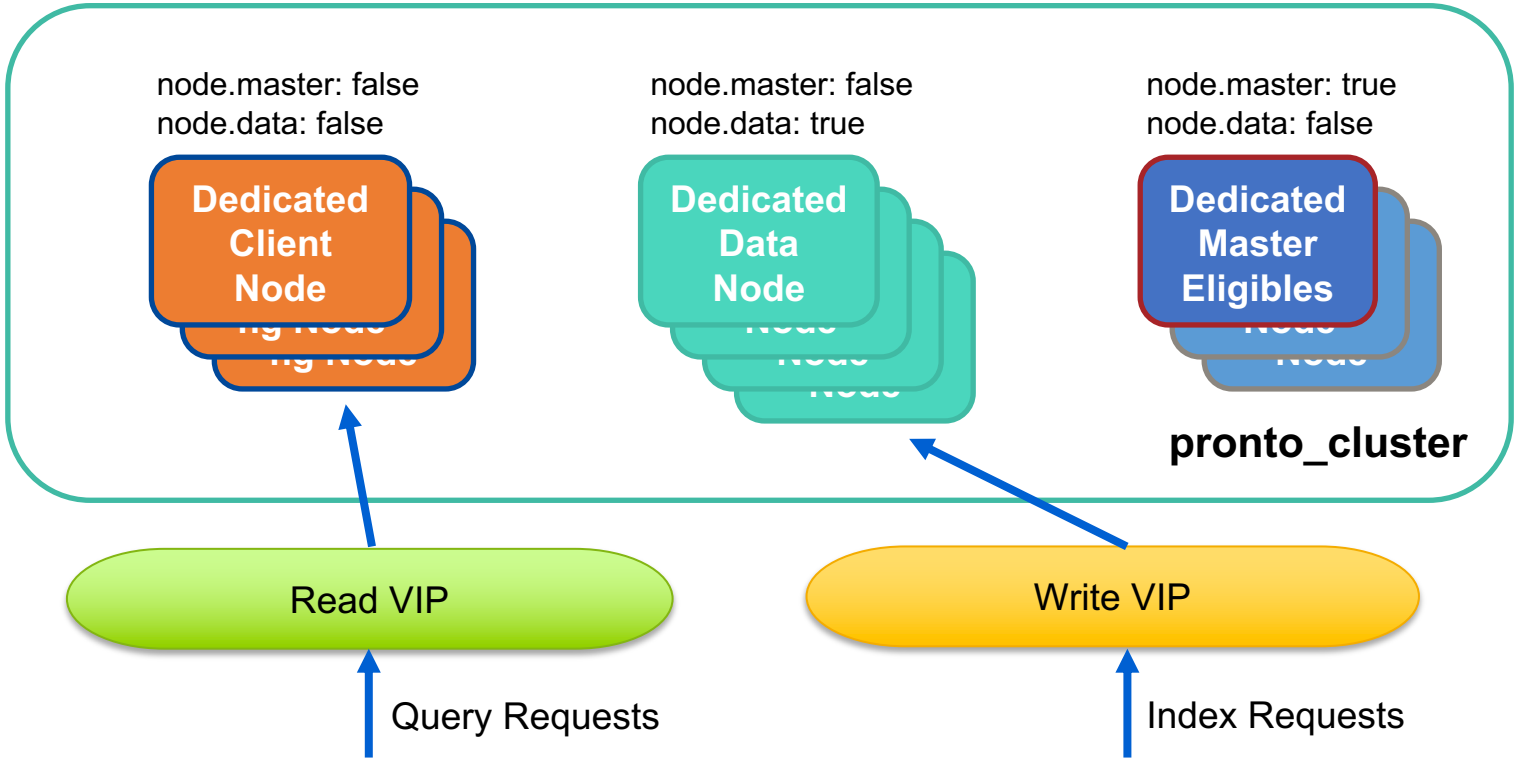


Ingestion Nodes

- An ***ingest node*** is used for pre-processing documents right before they get indexed
- Documents are passed through a ***pipeline*** of ***processors*** that you configure
- If the ingestion operation is intensive, dedicated ingestion node is recommended

```
node.master: false
node.data: false
node.ingest: true
```

Pronto's practice



Cluster Settings

- ***Cluster settings*** include:
  - logging levels, index templates, scripts, shard allocation, etc
  - typically configured using the REST APIs

```
PUT /_cluster/settings
{
   "transient" : {
      "cluster.routing.allocation.enable" : "none"
   }
}
```

- **persistent:** survives restarts
- **transient:** will be erased when cluster restarts

Dynamic Settings

- An index will dynamically be created during a document index request if the index does not already exist.
- You can disable this dynamic behavior completely using the dynamic "***action.auto_create_index***" setting:

```
PUT /_cluster/settings
{
    "persistent" : {
        "action.auto_create_index" : false
    }
}
```

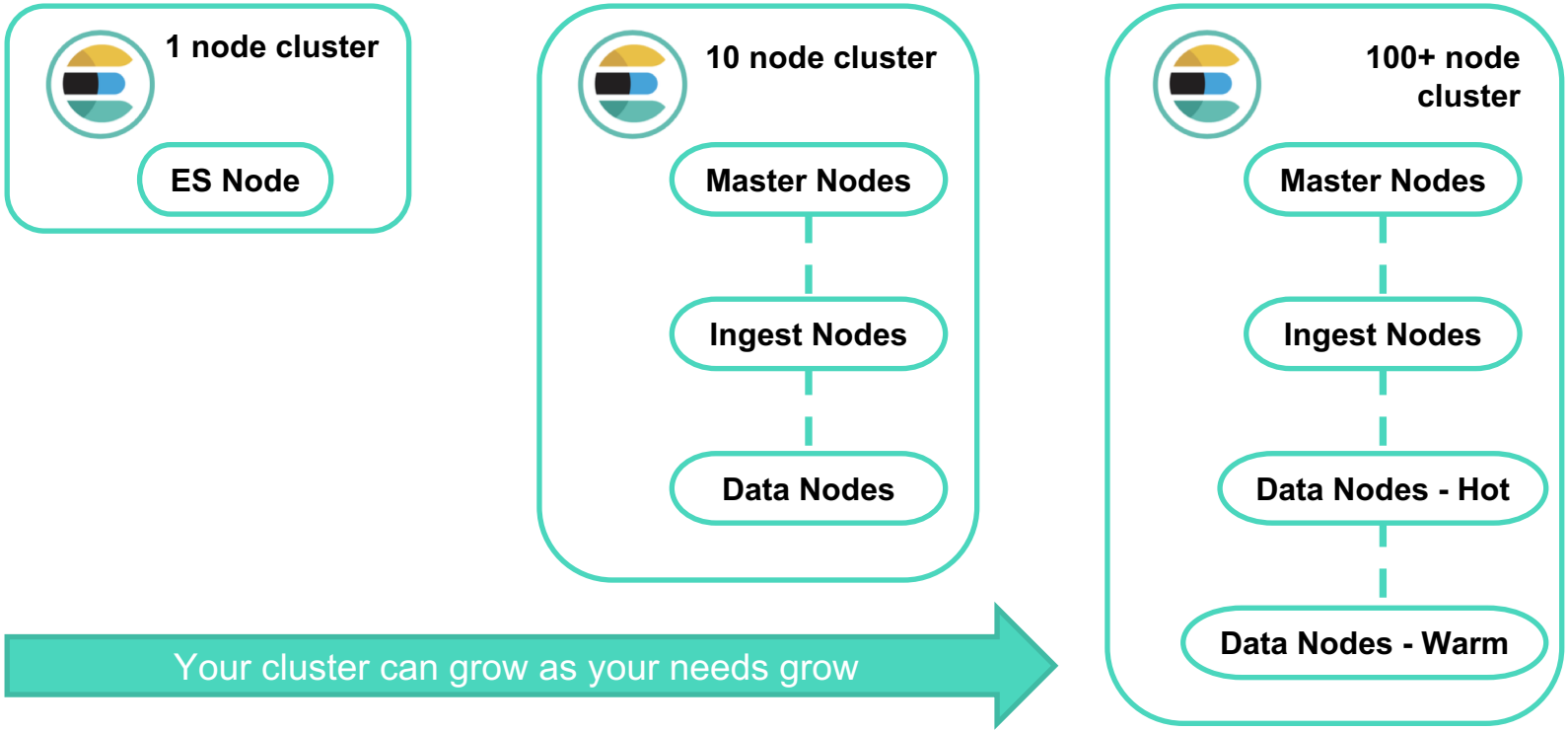
- Or, you can whitelist certain patterns:

```
PUT /_cluster/settings
{
    "persistent" : {
        "action.auto_create_index" :  "+logstash-*"
    }
}
```

Precedence of Settings

- ***Transient settings*** take precedence over all
  - ***Persistent settings*** take precedence over
    - ***Command-line settings***, which take precedence over
      - **elasticsearch.yml** settings

Dance with settings: Hot-warm architecture



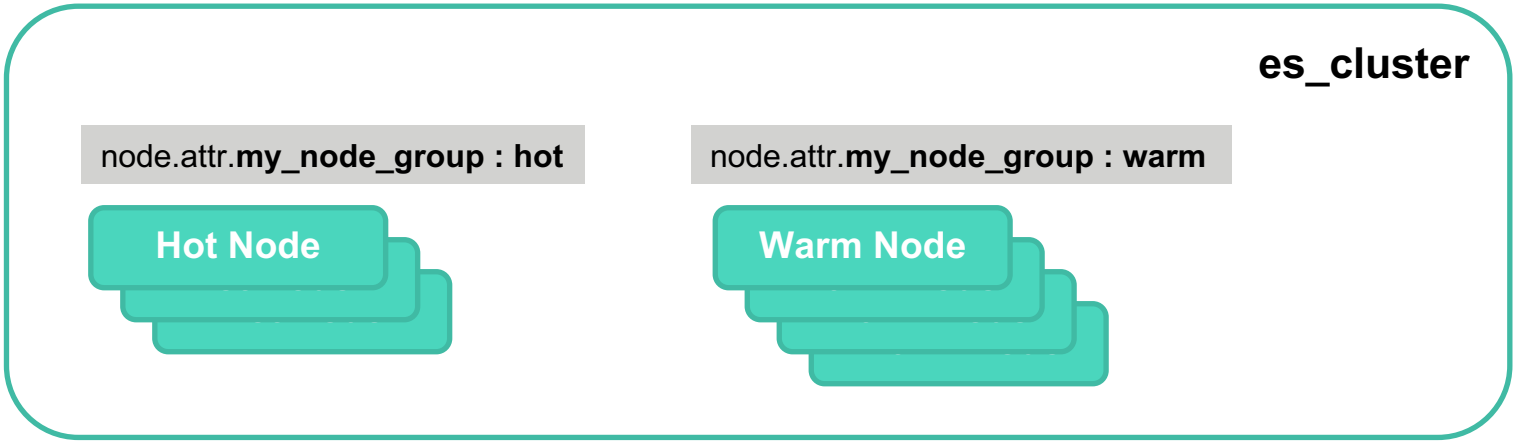
Hot-warm Architecture

- **Hot nodes** for supporting the indices with new documents being written to
  - indexing is a CPU and IO intensive operation, so hot nodes should be powerful servers
  - faster storage than the warm nodes

- **Warm nodes** for handling read-only indices that are not as likely to be queried frequently
  - tend to utilize large attached disks (usually spinning disks)
  - larger amounts of data may require additional nodes to meet performance requirements

Shard Filtering

- ***Shard filtering*** refers to the ability to control which nodes the shards for an index are allocated:
  - Step 1: use ***node.attr*** to tag your nodes



Shard Filtering (2)

– use ***index.routing.allocation*** to assign indexes to nodes

```
PUT prontologs-2018-08-02/_settings
{
   "index.routing.allocation.require.my_node_group" : "hot"
}
```

dynamic setting	assign the index to a node whose {attr} has
index.routing.allocation.include.{attr}	at least one of the values
index.routing.allocation.exclude.{attr}	none of the values
index.routing.allocation.require.{attr}	all of the values

Quiz & QA

1. If you have 4 master eligible nodes in you cluster, what should you set minimum_master_nodes to?
2. True or False: Every node in a cluster needs to enable the HTTP protocol in order to communicate with other nodes.
3. True or False: persistent settings take precedence over transient settings

Session 4 Elasticsearch Ops Basics

- Elasticsearch Settings
- **Health & Statistics API of Elasticsearch**
- Monitoring & Alerting, Diagnostic Tool of Pronto

Start from /_cat/health

GET /_cat/health?v

```
epoch      timestamp cluster status node.total node.data shards  pri relo init unassign pending_tasks max_task_wait_time active_shards_percent
1533718884 02:01:24  esmon   green           24         17  11639 5810    8    0        0             0                  -                 100.0%
```

Cluster status

shards in different state

Green, Yellow, Red

- The cluster is green only if all indices in the cluster are intact.
- The cluster is yellow when there are replica shards not ok.
- The cluster is red when there is a primary shard missing / broken.
  - How to find index with trouble?

```
GET /_cluster/health?level=indices
```

  - How es defines the status of a shard?
  - Let's take a look at

```
GET /_cat/shards?v
```

The life-cycle of a shard

- Shards go through several states:
  - **UNASSIGNED**
    - The shard not yet / failed to be assigned to any node
  - **INITIALIZING**
    - Shard is assigned to a node, but not finish writing
  - **STARTED**
    - The shard has finished initializing
  - **RELOCATING**
    - Node join/left, allocation settings changed
    - if shard size is over 50GB, it'll be very difficult for es to relocate!

Thinking about the colors

- All shards of an index are **STARTED**, then the index is **GREEN**.
- All indices are GREEN, the cluster is GREEN.
- Cluster status always reflects the worst shard status.
- Is YELLOW / RED always bad?
- Does GREEN mean everything ok?

Yellow / Red: Don't panic

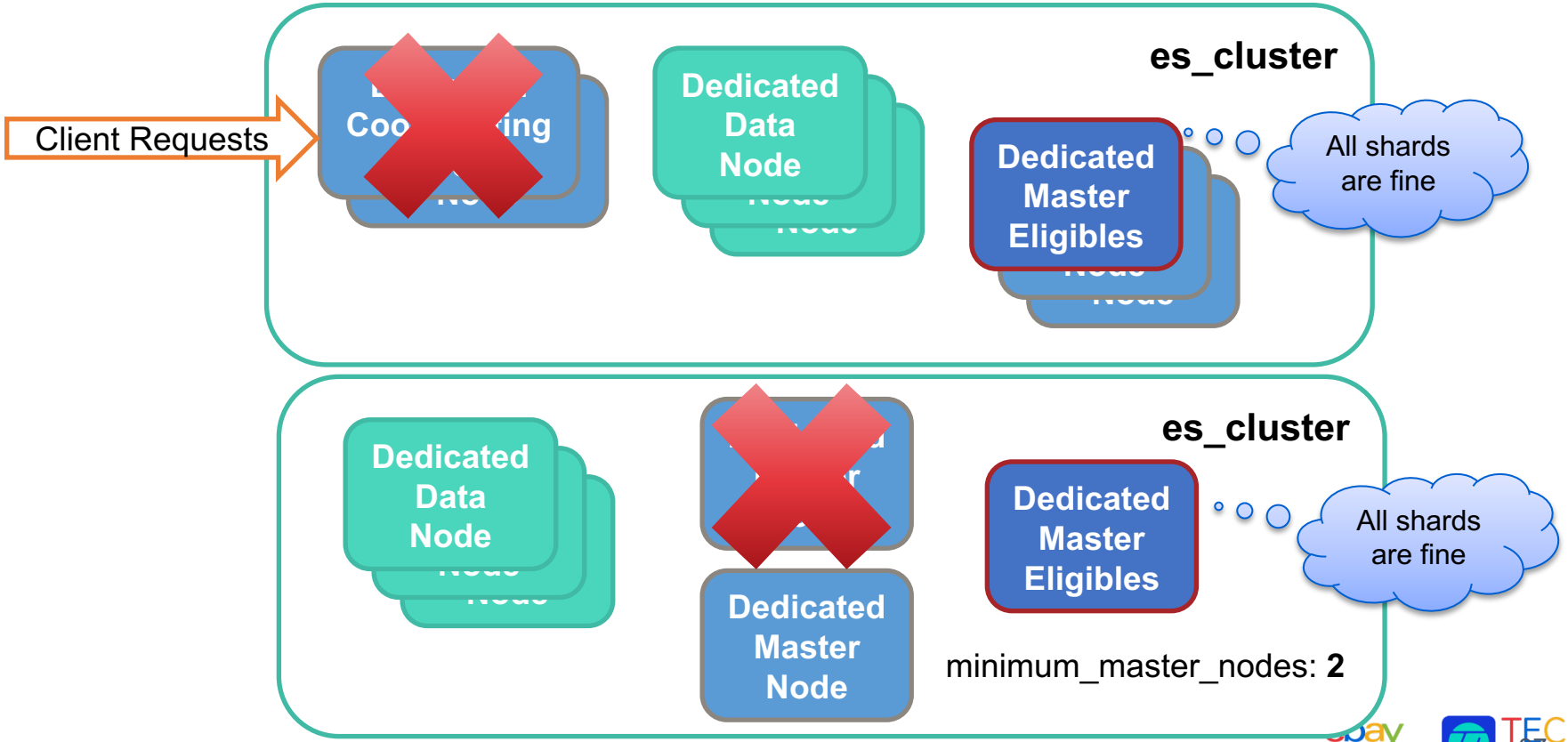
- **Yellow** means there's no data loss yet. It's very common when the index is big
- Most **Yellow** State can return to **Green** automatically
- Even **Red** doesn't necessarily mean data loss. Dangling index.
- Real threat: Disk full, Big Shards
- **Yellow** / **Red** definitely mean you need to pay attention, but don't pannic

```
GET _cluster/allocation/explain
```

Green status won't tell you…

- If replica number is set to 0, Green is no better than yellow.
- CPU Load / GC Time / Network latency is high
- Data allocation is skewed / not safe.
- Coordinating nodes are all down
- One master eligible node is down
- Shards are too big to be relocated, etc.

Don’t trust green



Stats APIs

- Elasticsearch has several APIs for monitoring:
  - **Node Stats**: _nodes/stats
    - _nodes/{node}/stats, _nodes/_local/stats
    - _nodes/{node}/stats/thread_pool,jvm,os,fs,indices
  - **Cluster Stats**: _cluster/stats
  - **Index Stats**: my_index/_stats
  - **Pending Cluster Tasks API**: _cluster/pending_tasks

- All these api return JSON

Frequently used APIs

- _cat/nodes
- _cat/shards
- _cat/allocation
- _cat/thread_pool, _nodes/{node}/hot_threads
- _cluster/health/{index}?level=shards
- _cluster/state

Quiz & QA

1. Can we have 2 clusters with same cluster name in the same environment?

2. Suppose we have a index rolling daily, data volume is 20GB every day. What actions should we do if we found the volume of our data indexing into es is increasing dramatically? 200GB? 2TB?

3. Is it possible to get the actual indexing rate (doc indexed per minute) of a cluster? What about search rate (search request per minute)?

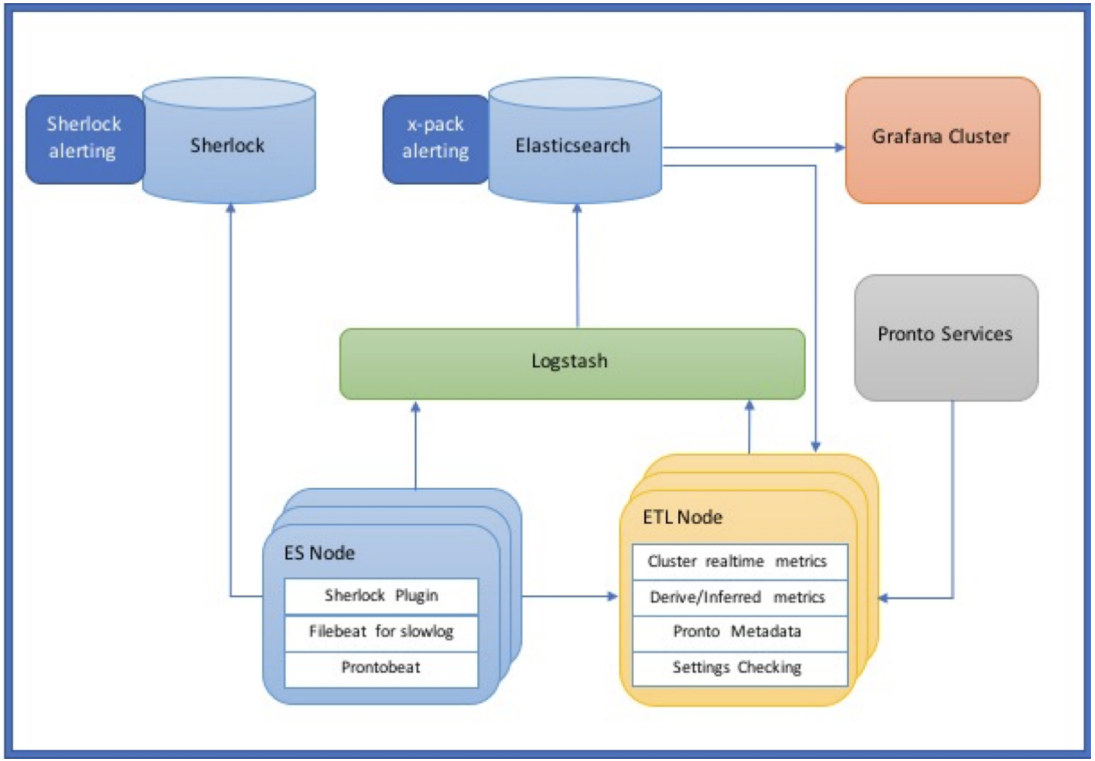
Session 4 Elasticsearch Ops Basics

- Elasticsearch Settings
- Health & Statistics API of Elasticsearch
- **Monitoring & Alerting, Diagnostic Tool of Pronto**

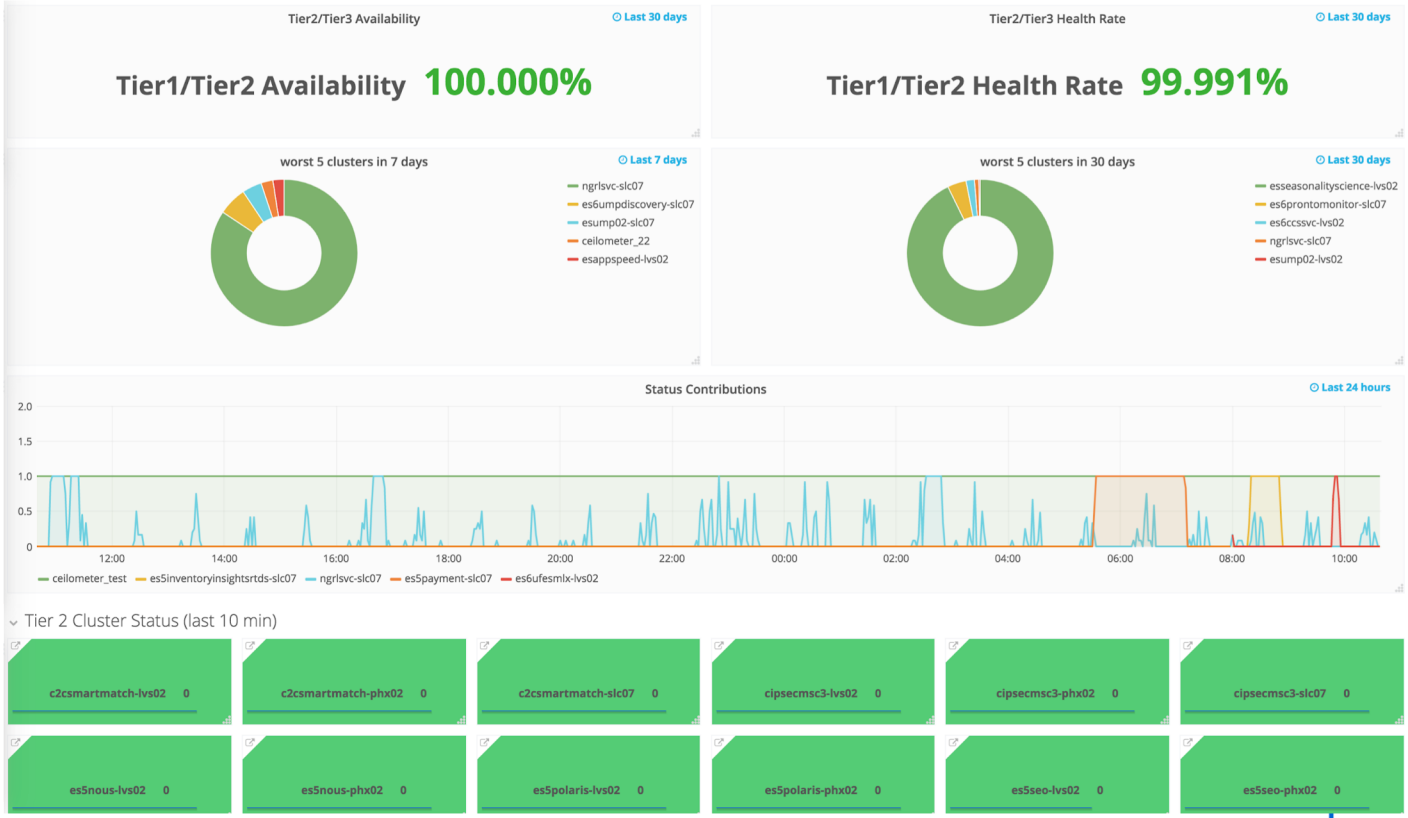
What a monitoring/alerting system should be like?

- Each Component should be High HA
- Each Component should be monitored
  - Not only heart beat
- Should be Alerting its own failures
- There should be at least one external monitoring
- Should be easy to scale and extend
- Agents should be robust and light-weighted

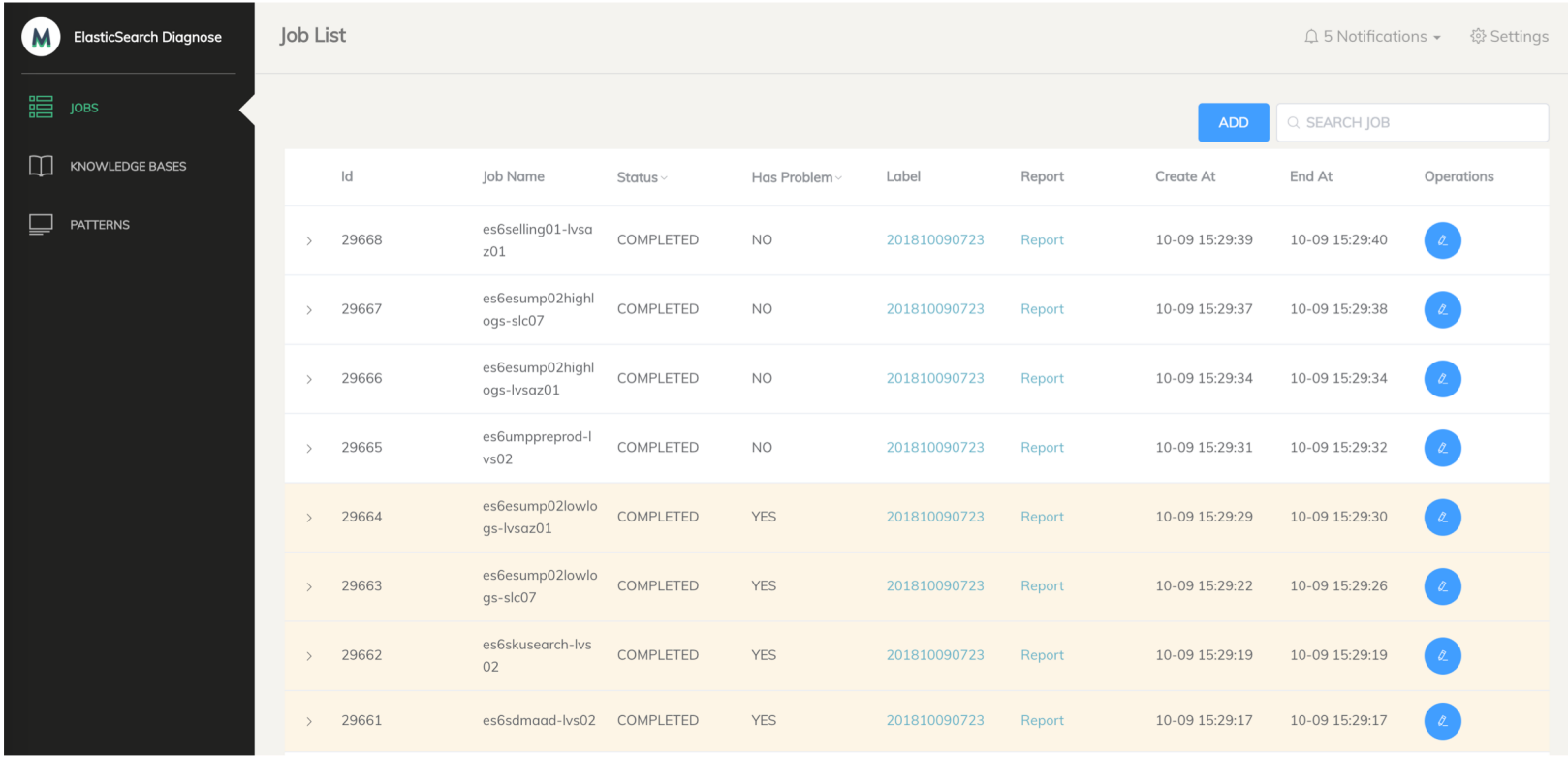
Architecture of Pronto Monitoring & Alerting



Pronto Dashboard



Diagnostic Tool



Id	Job Name	Status	Has Problem	Label	Report	Create At	End At	Operations
29668	es6selling01-lvsaz01	COMPLETED	NO	201810090723	Report	10-09 15:29:39	10-09 15:29:40	
29667	es6esump02highlogs-slc07	COMPLETED	NO	201810090723	Report	10-09 15:29:37	10-09 15:29:38	
29666	es6esump02highlogs-lvsaz01	COMPLETED	NO	201810090723	Report	10-09 15:29:34	10-09 15:29:34	
29665	es6umppreprod-lvs02	COMPLETED	NO	201810090723	Report	10-09 15:29:31	10-09 15:29:32	
29664	es6esump02lowlogs-lvsaz01	COMPLETED	YES	201810090723	Report	10-09 15:29:29	10-09 15:29:30	
29663	es6esump02lowlogs-slc07	COMPLETED	YES	201810090723	Report	10-09 15:29:22	10-09 15:29:26	
29662	es6skusearch-lvs02	COMPLETED	YES	201810090723	Report	10-09 15:29:19	10-09 15:29:19	
29661	es6sdmaad-lvs02	COMPLETED	YES	201810090723	Report	10-09 15:29:17	10-09 15:29:17	

Diagnostic Tool

- Knowledge Base is more important than tool itself
  - Cluster status should be green
  - Minimum master node should be set to eligible/2+1
  - Master eligible node count should be larger than 2
  - Shard size should be under 50G
  - Replica number should be larger than 0
  - Shard allocation should be enabled
  - Shard size should be balanced(no more than 30% delta than avg value)
  - A index should not have more than 1000 fields
  - Rack ID should be configured on every node
  - …

Review of Session 2

- **Elasticsearch Settings**
  - cluster settings, node settings, index settings
- **Health & Statistics API of Elasticsearch**
  - cluster status, _cat api, _cluster api
- **Monitoring & Alerting, Diagnostic Tool of Pronto**
  - pronto monitoring & alerting, diagnostic tool

5

ELKB Stack & Workshop

Session 5 ELKB Stack & Workshop

- Kibana
- Logstash
- Pipeline
- Beats

Kibana

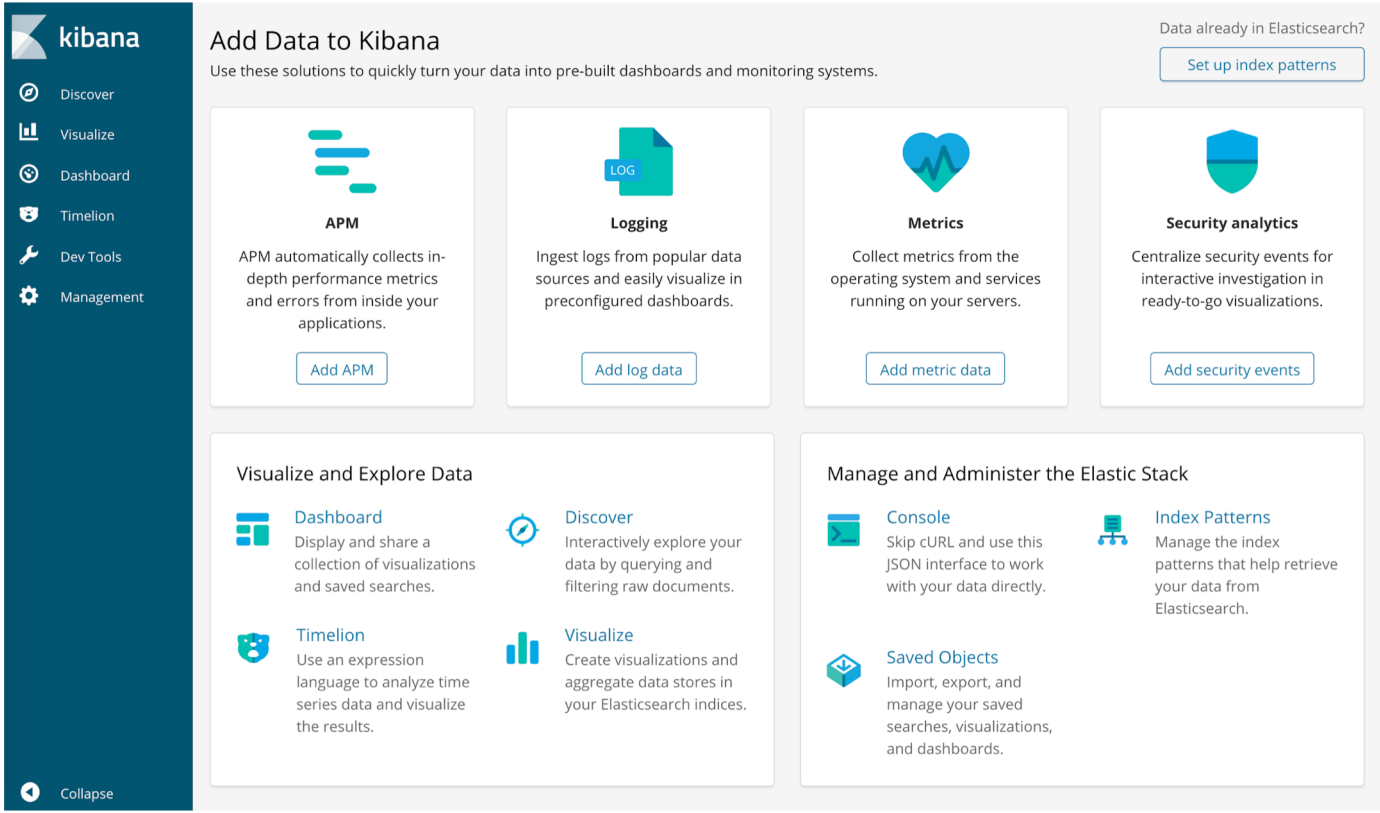
- Kibana is an open source analytics and visualization platform designed to work with Elasticsearch
  - You use Kibana to search, view, and interact with data stored in Elasticsearch indices. You can easily perform advanced data analysis and visualize your data in a variety of charts, tables, and maps.

- Kibana is actually a portal & hub of ES
  - It integrates es features & advanced functionalities

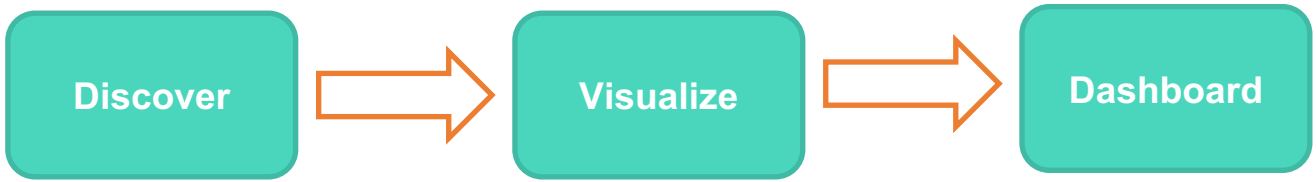
Install & Run Kibana

- Get Kibana at:
  - https://www.elastic.co/downloads/kibana
- For deb installation:
  - Config path is: /etc/kibana, Bin path is /usr/share/kibana/bin
- For tar installation:
  - Config path is: {kibana_path}/config, Bin path is {kibana_path}/bin
- Config ES endpoint in kibana.yml

Kibana UI



Typical Kibana Working Process



Logstash: Collect, Enrich and Transport

- **The ingestion workhorse for Elasticsearch and more**
  - Horizontally scalable data processing pipeline with strong Elasticsearch and Kibana synergy
- **Pluggable pipeline architecture**
  - Mix, match, and orchestrate different inputs, filters, and outputs to play in pipeline harmony
- **Community-extensible and developer-friendly plugin ecosystem**
  - Over 200 plugins available, plus the flexibility of creating and contributing your own

Install & Run Logstash

- Get Logstash at:
  - https://www.elastic.co/downloads/logstash
- For deb installation:
  - Config path is: /etc/logstash, Bin path is /usr/share/logstash/bin
- For tar installation:
  - Config path is: {logstash_path}/config, Bin path is {logstash_path}/bin
- Config logstash conf

Get Started with Logstash

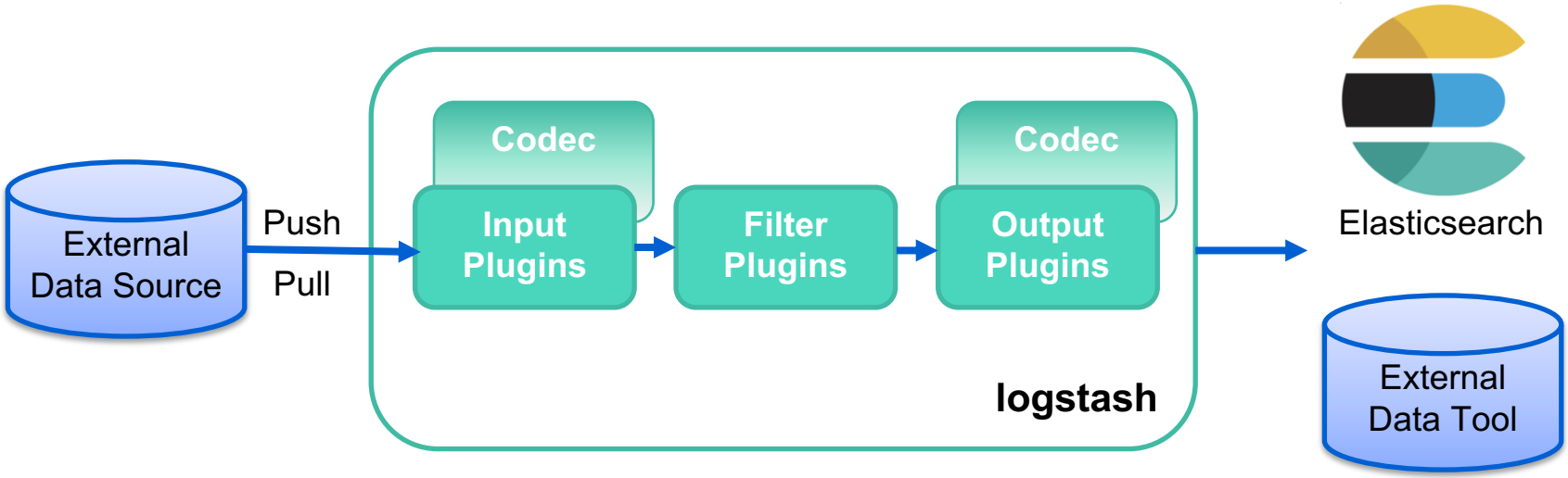
```
logstash -e 'input { stdin { } } output { stdout { codec => rubydebug } }'
```

```
Settings: Default pipeline workers: 2
Pipeline main

> I love elkb

{
       "message" => ”I love elkb",
      "@version" => "1",
 "@timestamp" => "2018-08-02T01:24:55.513Z",
              "host" => "localhost.localdomain"
}
```

How Logstash Works



Logstash Configuration Files

- settings files
  - specify options that control Logstash startup and execution
- pipeline configuration files
  - define the Logstash processing pipeline
- Let's see an example of pipeline configuration

An Example of Logstash Configuration File

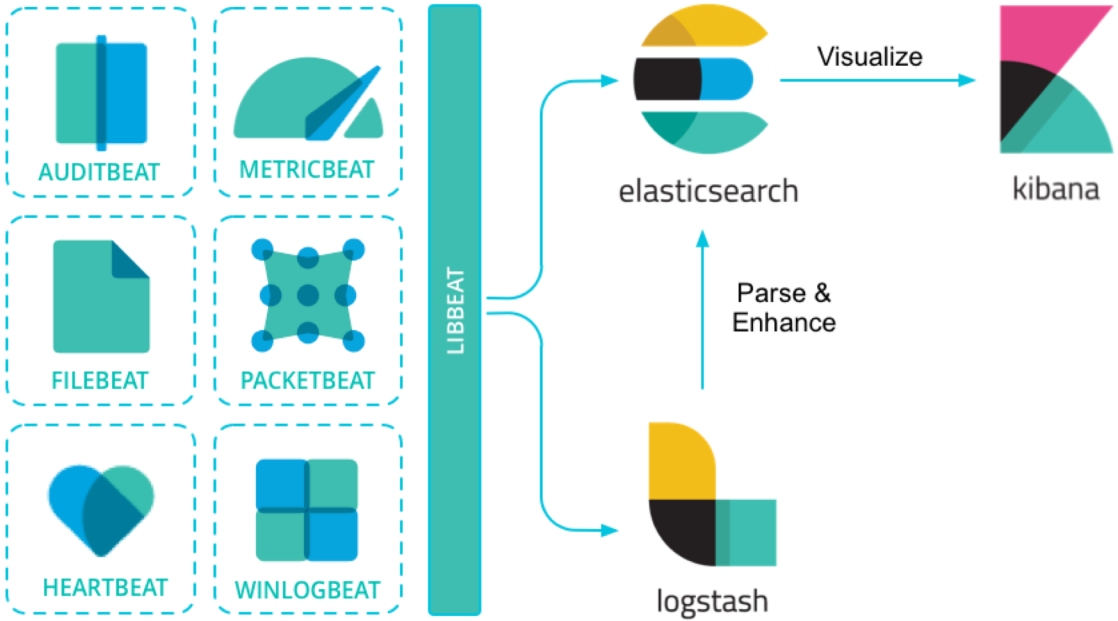
```
input {
    beats {
        port => 5044
    }
    http {
        port => 8080
        codec => json_lines
    }
}
filter {
    date {
        match => ["metrictime", "UNIX", "UNIX_MS", "yyyy-MM-dd'T'HH:mm:ss,SSS"]
    }
}
output {
    elasticsearch {
        id => "es_1"
        hosts => ["vip"]
        index => "clustermetrics-%{+YYYY.MM.dd}"
        document_id => "%{[@metadata][es_id]}"
    }
}
```

Too many roles for Logstash?

- As a agent, it’s too heavy
- As a server, it’s too simple, there’s no cluster mode.
- What if we just want a component with single functionality?

Beat: Collect, Parse & Ship

- Beats are open source data shippers that you install as agents on your servers to send operational data to Elasticsearch / Logstash
- It's written in GoLang, and it's light weighted, easy to deploy and use.



Filebeat Example

```
filebeat.inputs:
  - type: log
    paths:
      - /mnt/pronto/index_search_slowlog.log
    include_lines: ["\\[INFO\\s*\\]", "\\[WARN\\s*\\]"]
    pipeline: "search-slowlog-pipeline-filebeat"

  - type: log
    paths:
      - /mnt/pronto/indexing_slowlog.log
    include_lines: ["\\[INFO\\s*\\]", "\\[WARN\\s*\\]"]
    pipeline: "index-slowlog-pipeline-filebeat"

output.elasticsearch:
  hosts: [”localhost:9200"]
  username: "filebeat_internal"
  password: "YOUR_PASSWORD”
  index: slowlogs
```

Elasticsearch Pipeline

- Ingest Nodes are able to run a ***pipeline***
  - pipeline is configuration that defines a series *processors*

- Processors transform a document in some way. For example
  - rename a field
  - split a field into multiple fields

Pipeline Processor

- Like filter of logstash, have different funcationalities.

```
{
  "description" : "Ingest etl logs.",
  "processors" : [
    {
      "date" : {
        "field" : "logtime",
        "target_field" : "@timestamp",
        "formats" : ["UNIX_MS"]
      }
    }, {
      "remove": {
        "field": "logtime"
      }
    },{
      "date_index_name": {
        "field" : "@timestamp",
        "index_name_prefix" : "etl_log-",
        "date_rounding" : "d"
     }}
  ]}
```

Workshop: a Bitcoin Price Dashboard