elastic

Elastic Machine Learning Workshop

Rich Collier
Solutions Architect

Challenges that Anomaly Detection Solves

- IT Operations
  - How do I know my systems are behaving normally?
  - Where to set thresholds for good alerting?
  - How to find the root cause of problems when I don’t know what to look for?
- IT Security
  - Do I have systems that are compromised with malware?
  - Which users could be an insider threat?
- Internet of Things / SCADA / Other
  - Is my factory working normally?
  - What do I do with thousands of time-series data points?
  - Which traffic incidents are causing the most delay?

Overview

- Anomaly Detection Concepts
- How to Learn “Normal”
- Lab 1: The Single Metric Job
- Splitting the Analysis
- Lab 2: Multi-Metric Jobs
- The Anomaly Explorer View
- Custom URLs
- Forecasting
- Lab 3: Forecasting
- Population Analysis
- Custom Rules
- Calendars
- Advanced Jobs
- Other Types of Analysis
  - Rarity
  - Categorization
- Lab 4: Choose Your Own Adventure
  - Root Cause Analysis
  - Rogue IPs in Weblogs
  - NYC Taxi Ridership

Anomaly Detection Concepts

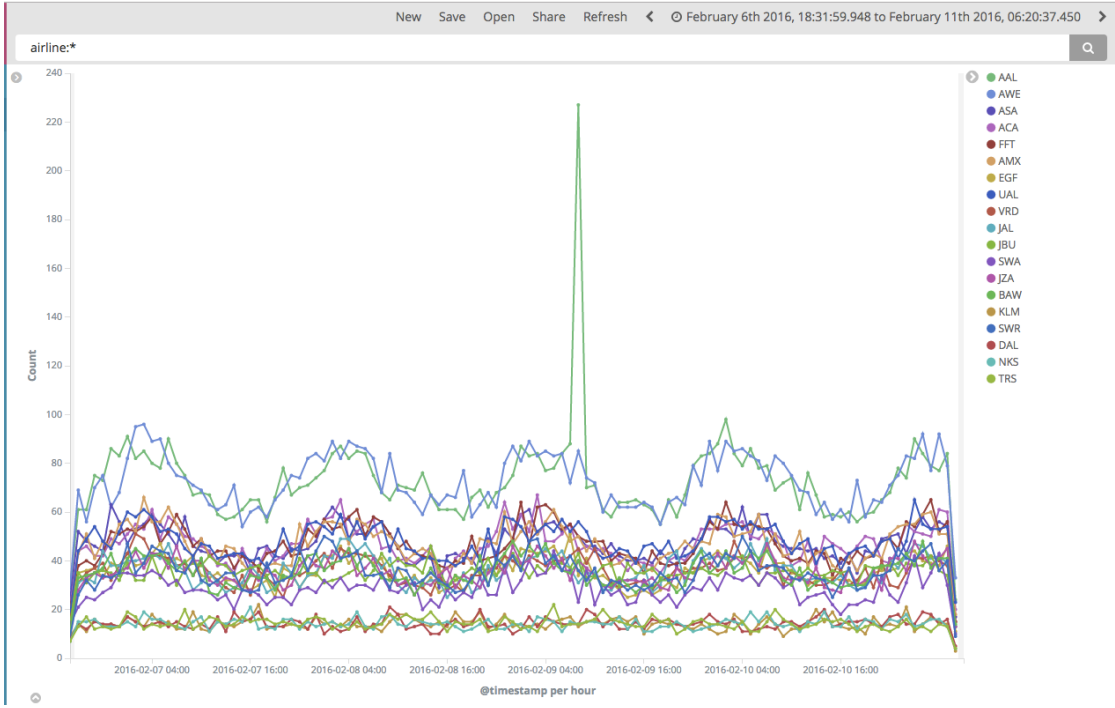
Terminology

- ***Machine Learning***
  - Broad term, but Elastic Machine Learning is automated anomaly detection and forecasting for time-series data (for now).
- ***Anomaly Detection***
  - Discovery of what's "weird" or "different", not what's "bad"
- ***Unsupervised Learning***
  - Learning without human-labeled examples (without being "taught"). Rely only on the data
- ***Bayesian***
  - An approach based on probability in which prior results are used to calculate probabilities of certain present or future events

What is “Abnormal”?

What’s abnormal here?

Why?

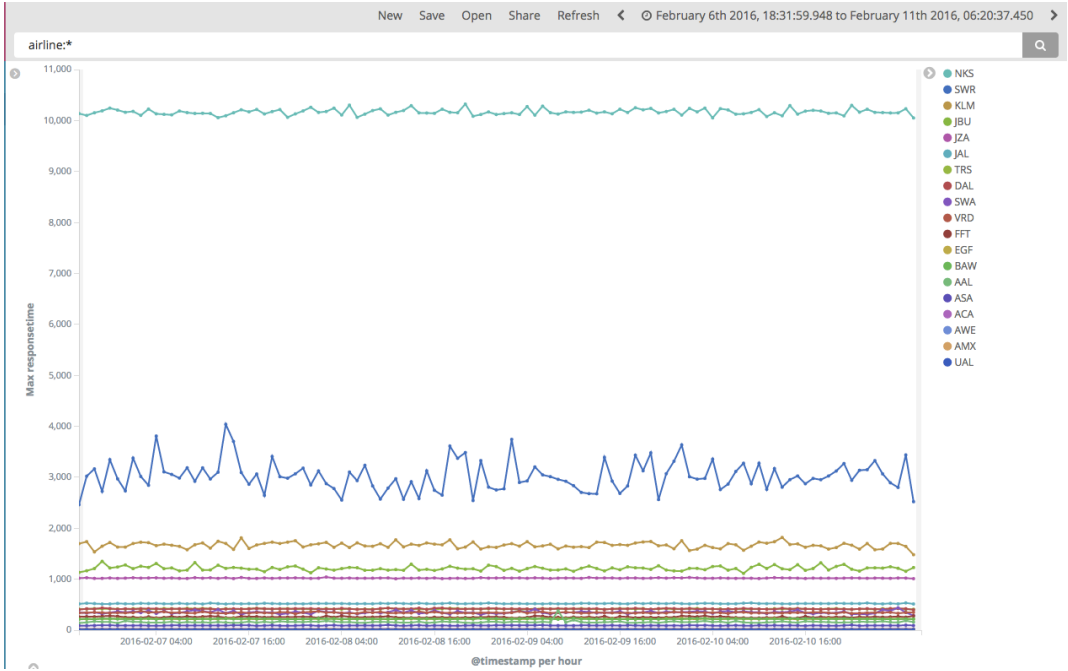


elastic

What is “Abnormal”?

What’s abnormal here?

Why?



elastic

What is “Abnormal”?

What’s abnormal here?

Why?

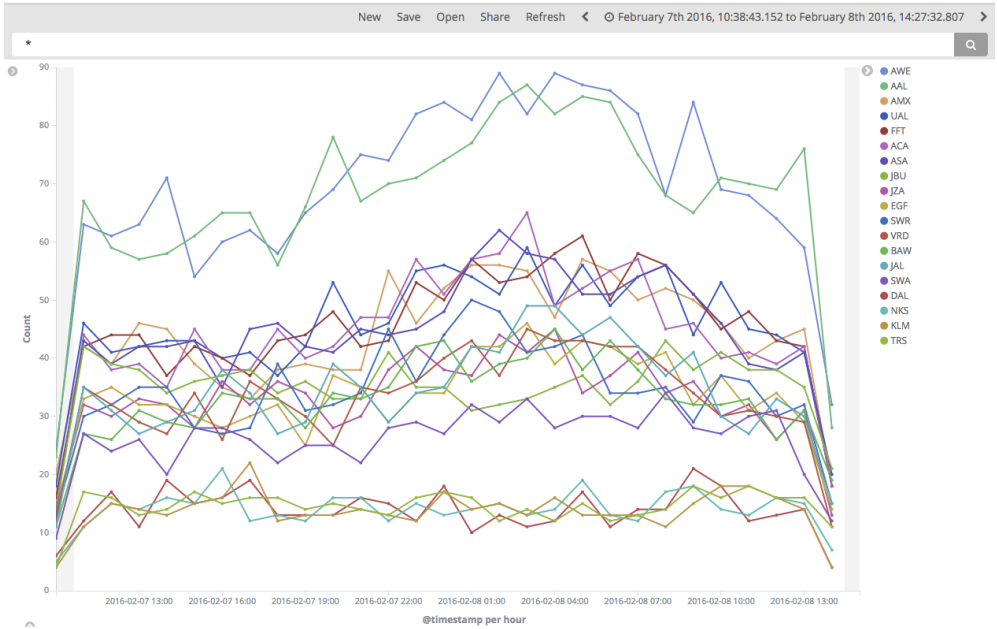
elastic

What is “Normal”?

In general, this question can be answered in two ways:

1) Something behaves in a consistent way with respect to itself, over time

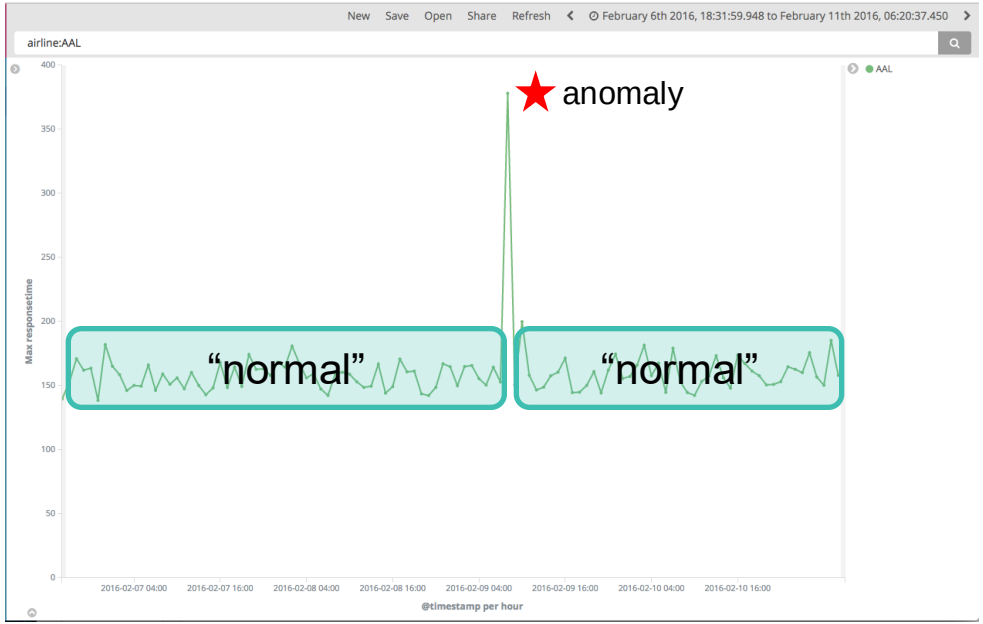
2) Something behaves in a consistent way compared against similar entities



elastic

What is “Abnormal”?

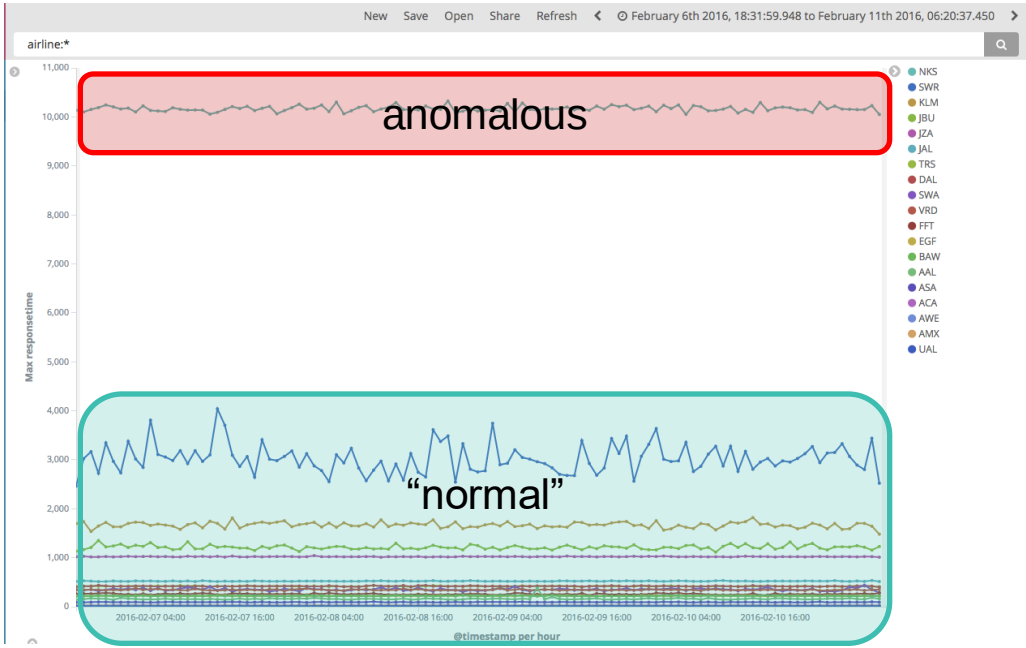
1) If something changes its behavior, compared to its own history – that change is ***anomalous***.



elastic

What is “Abnormal”?

2) If something is drastically different that others within a population, then that entity is ***anomalous***.

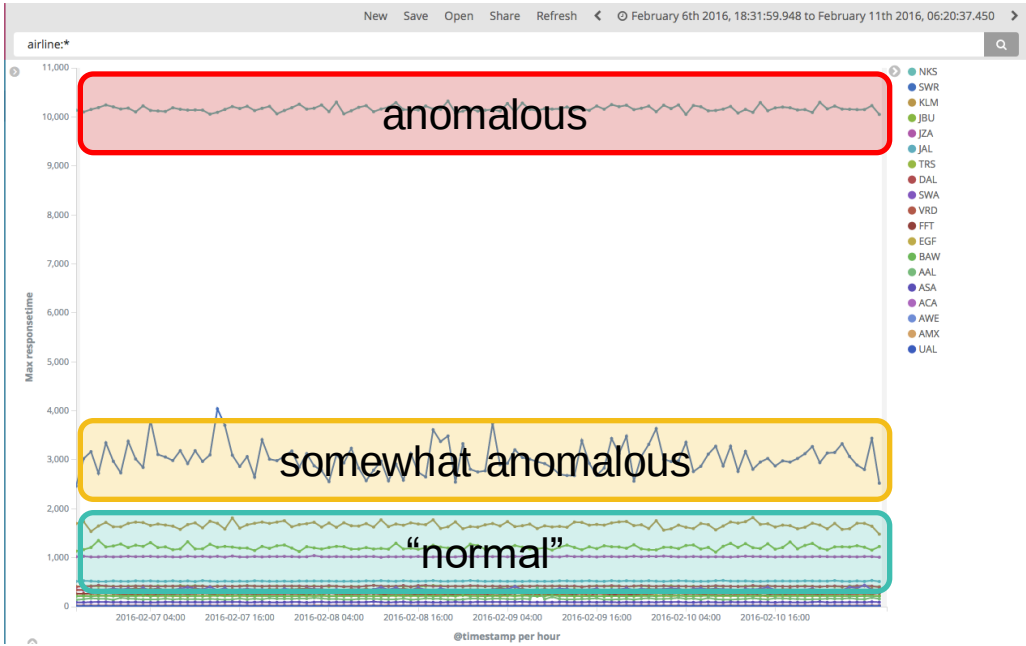


elastic

What is “Abnormal”?

2) If something is drastically different that others within a population, then that entity is ***anomalous***.

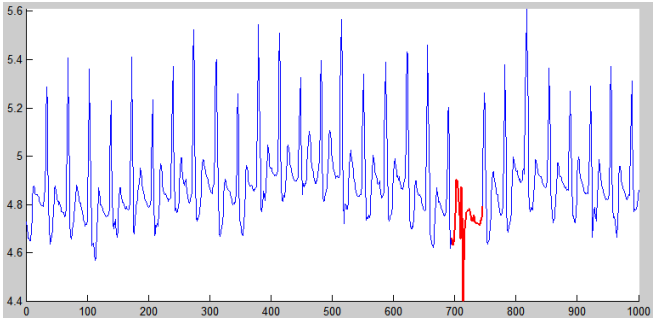
There’s also the concept of being “somewhat anomalous”



elastic

In Summary, Anomalousness is:

1) When an entities' ***behavior changes*** significantly and suddenly



2) When an entity is drastically ***different than others*** within a population

elastic

How to Learn “Normal”

An Analogy

- How could I learn how much Postal-mail you get daily and how could I use that information to predict how much you might get tomorrow?

elastic

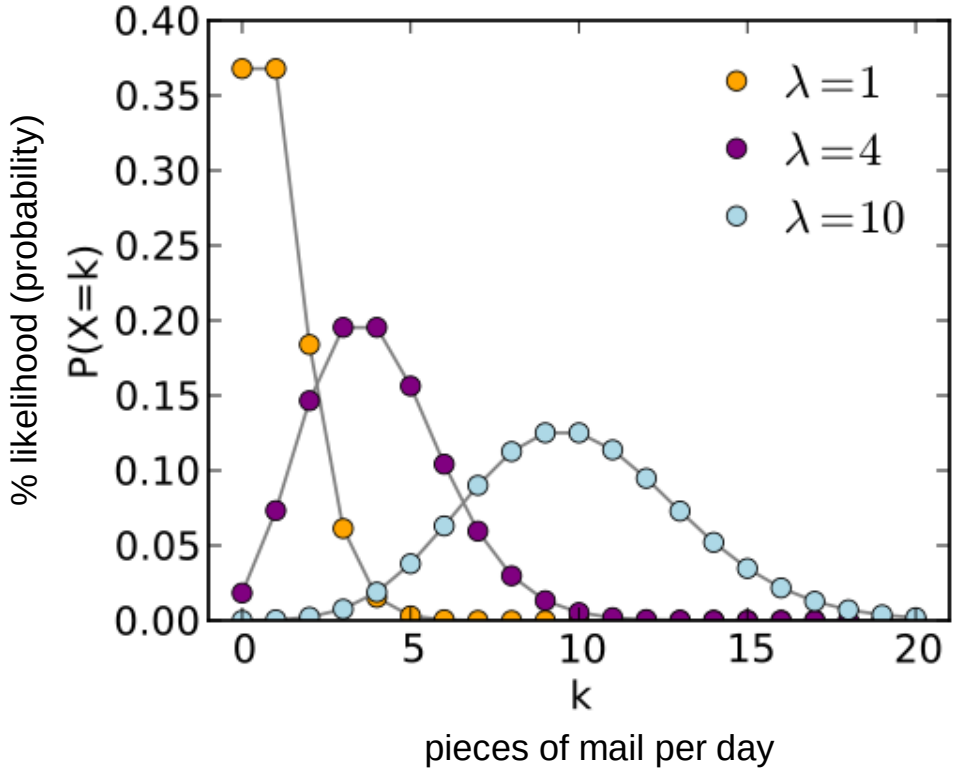
I could stand on your front porch and…

- Watch your mail delivery volume for a while, but how long?
  - 1 day?
  - 1 week?
  - 1 month?

- Notice, that you intuitively feel like you'll gain accuracy in your predictions with more data that you see.

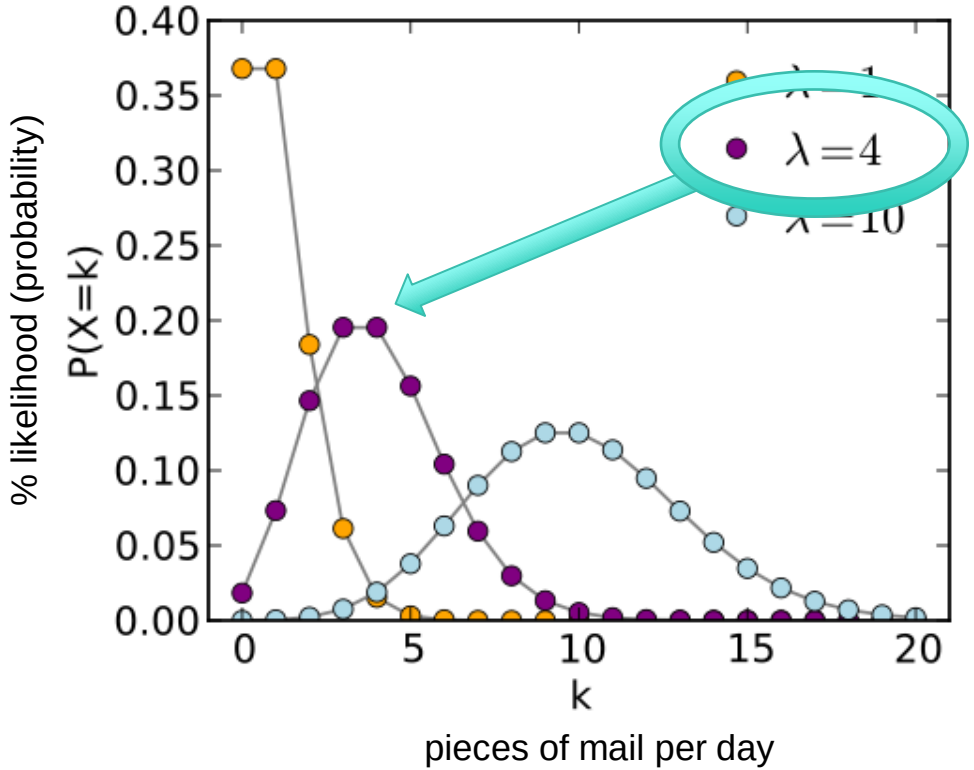
- Use those observations to create a model…

Probability Distribution Function



elastic

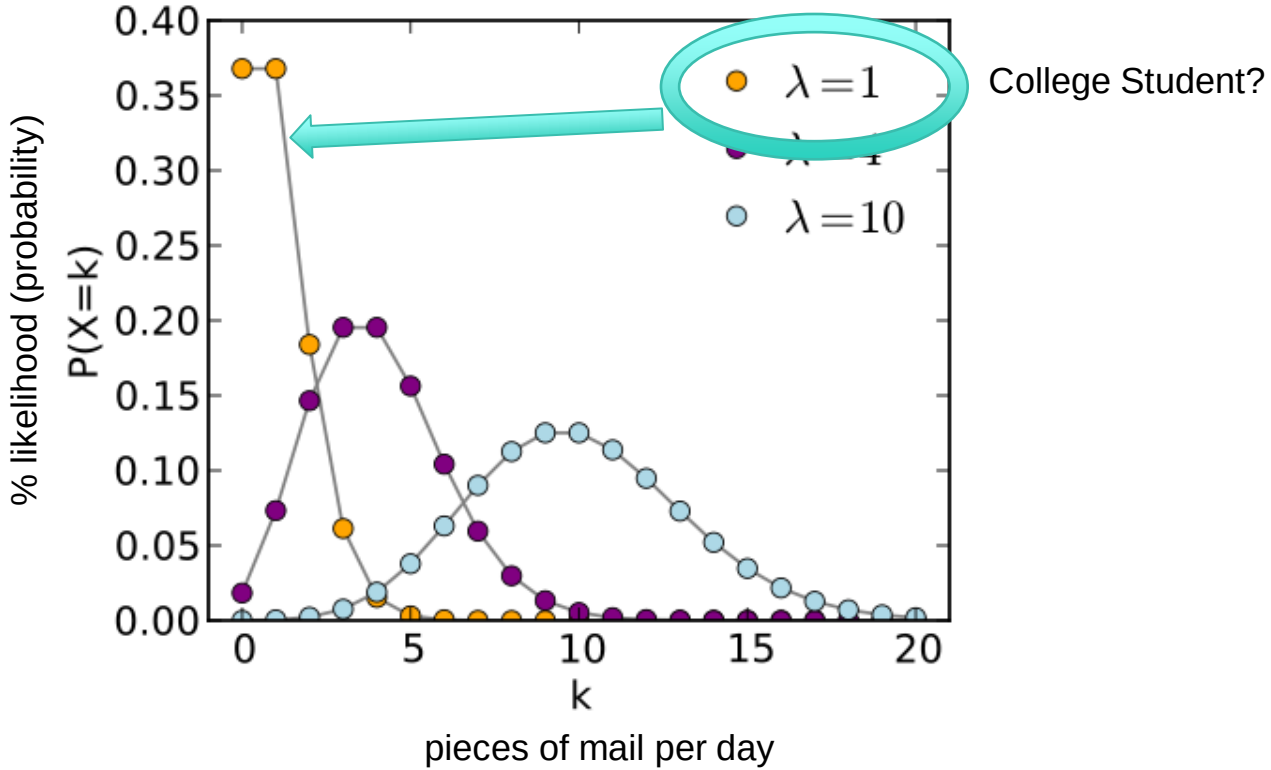
Probability Distribution Function



Best for my house

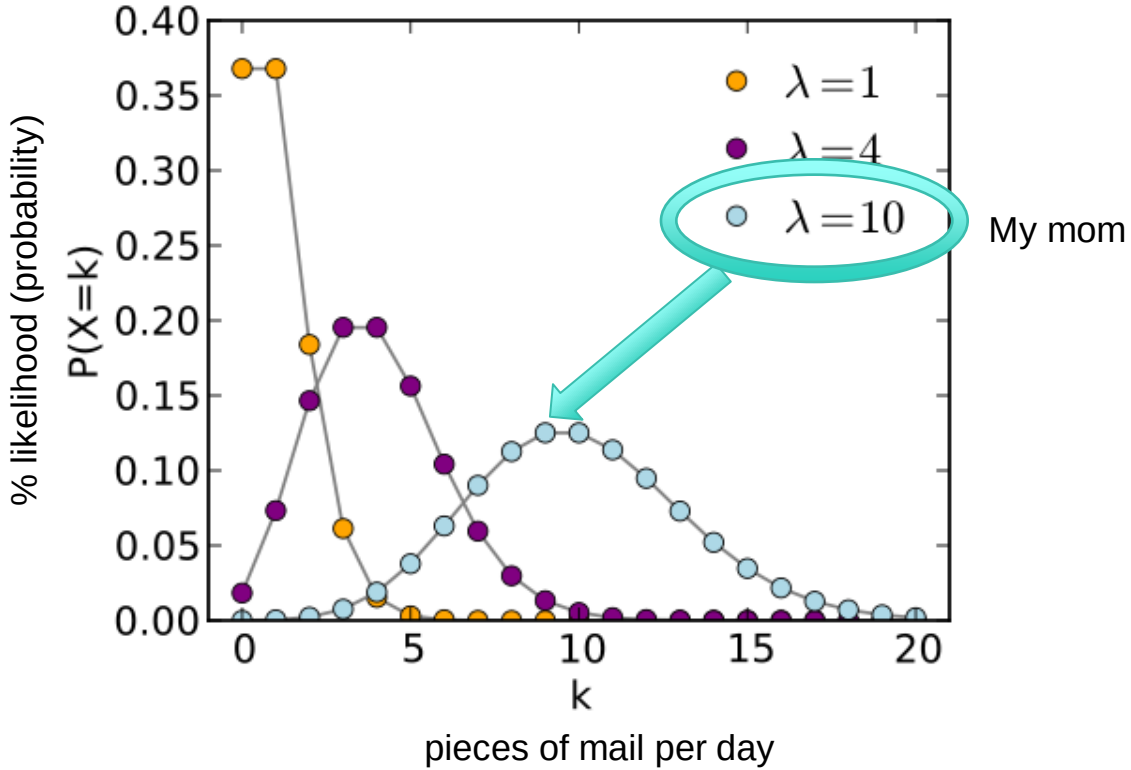
elastic

Probability Distribution Function



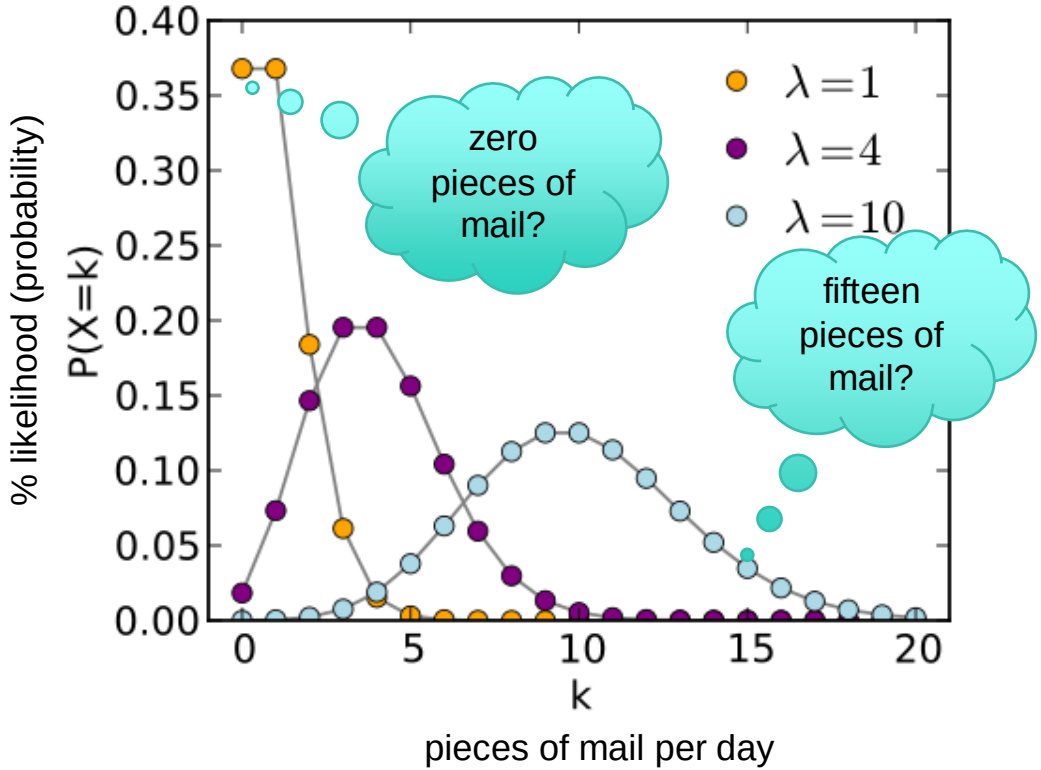
elastic

Probability Distribution Function



elastic

Using the Model to Find What is Unexpected



elastic

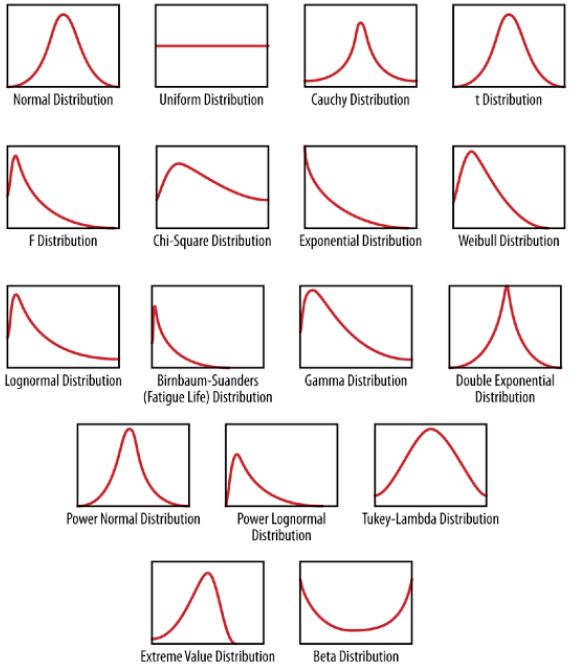
Relate back to IT/Security data

- # Pieces of mail = # events of a certain type
  - number of failed logins
  - number of errors in a log
  - number of events with certain status codes
- Or, metrics
  - response time
  - utilization
  - orders per minute

=> Every kind of data will need its own unique “model” (probability distribution function)

How does one pick a model?

Which one best fits your data?

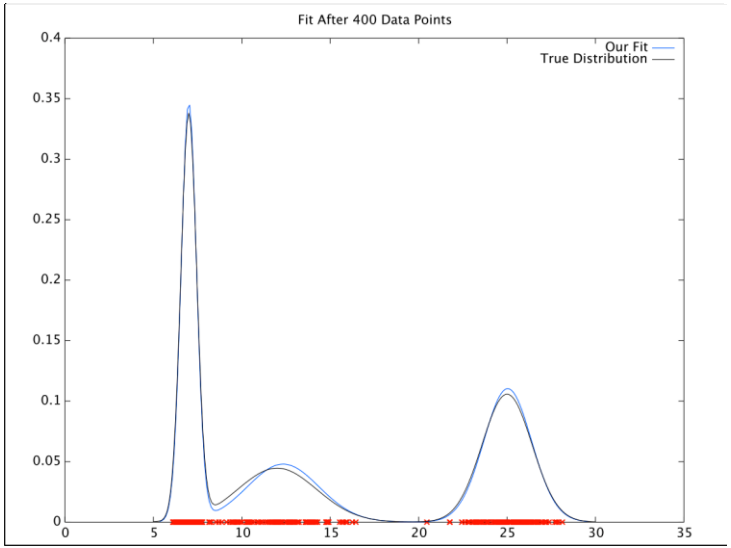


source: “Doing Data Science” O’Neil & Schutt

elastic

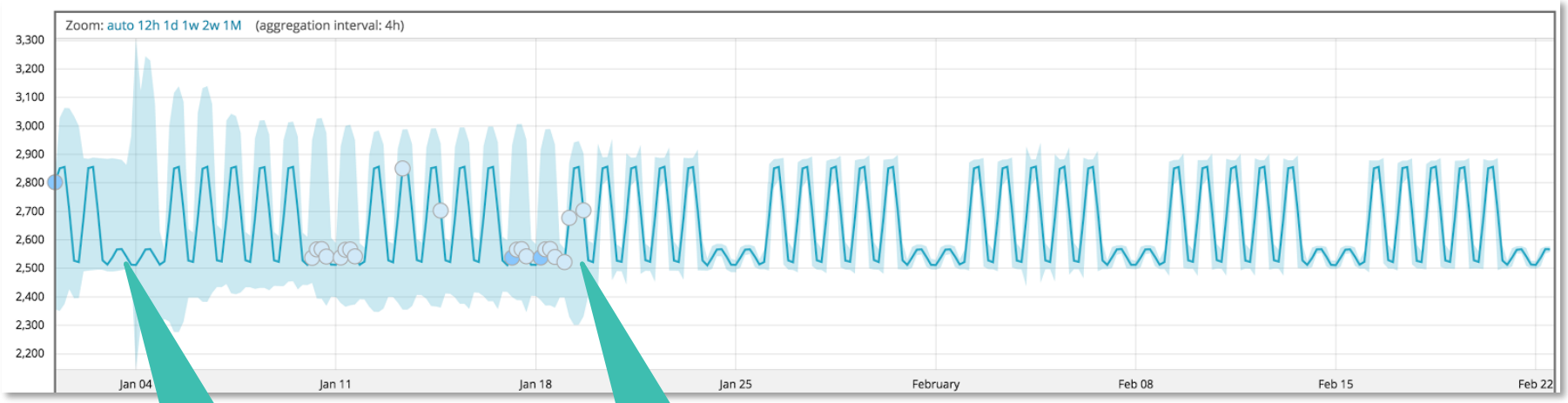
Machine Learning picks it for you

- ML uses sophisticated machine-learning techniques to best-fit the right statistical model for your data.
- Better models = better outlier detection = less false alarms
- Anomalies occur when observation is in low probability area



elastic

Also, model needs to account for any periodicity



After 2 full days, daily periodicity has been learned.

After 2 full weeks, weekly periodicity has been learned.

elastic

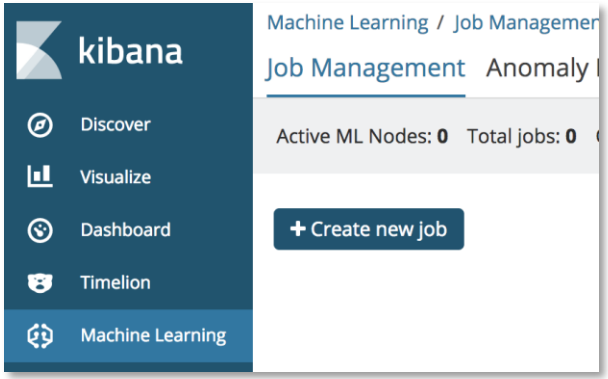
Lab 1: The Single Metric Job

Goal: Detect anomalies in a single time series

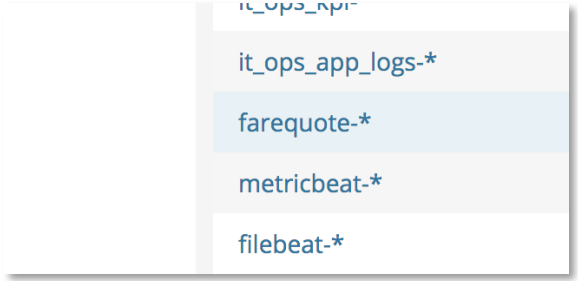
elastic

Steps to Complete

1) In Machine Learning, Create new job



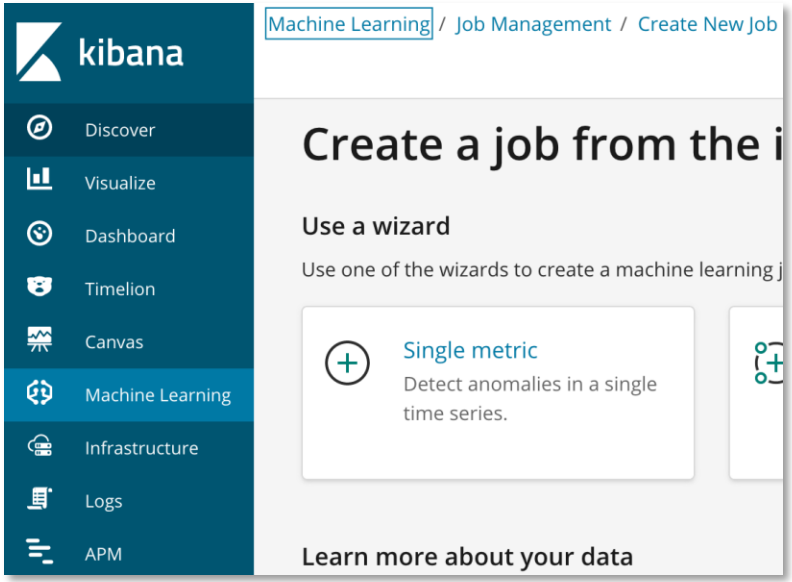
2) Choose the “farequote-*” index pattern



elastic

Steps to Complete

3) pick the Single Metric Wizard



elastic

Steps to Complete

4) Aggregation: count

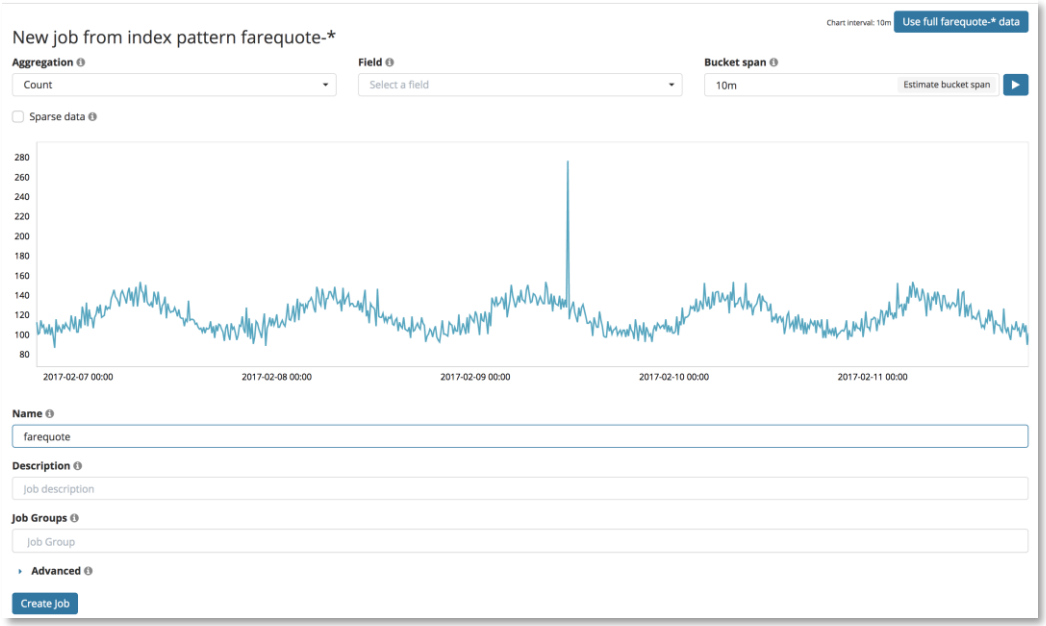
5) Field: <leave blank>

6) Bucket span: 10m

7) Click the “use full farequote data” button

8) Name: “farequote”

9) click “Create Job”



elastic

Steps to Complete

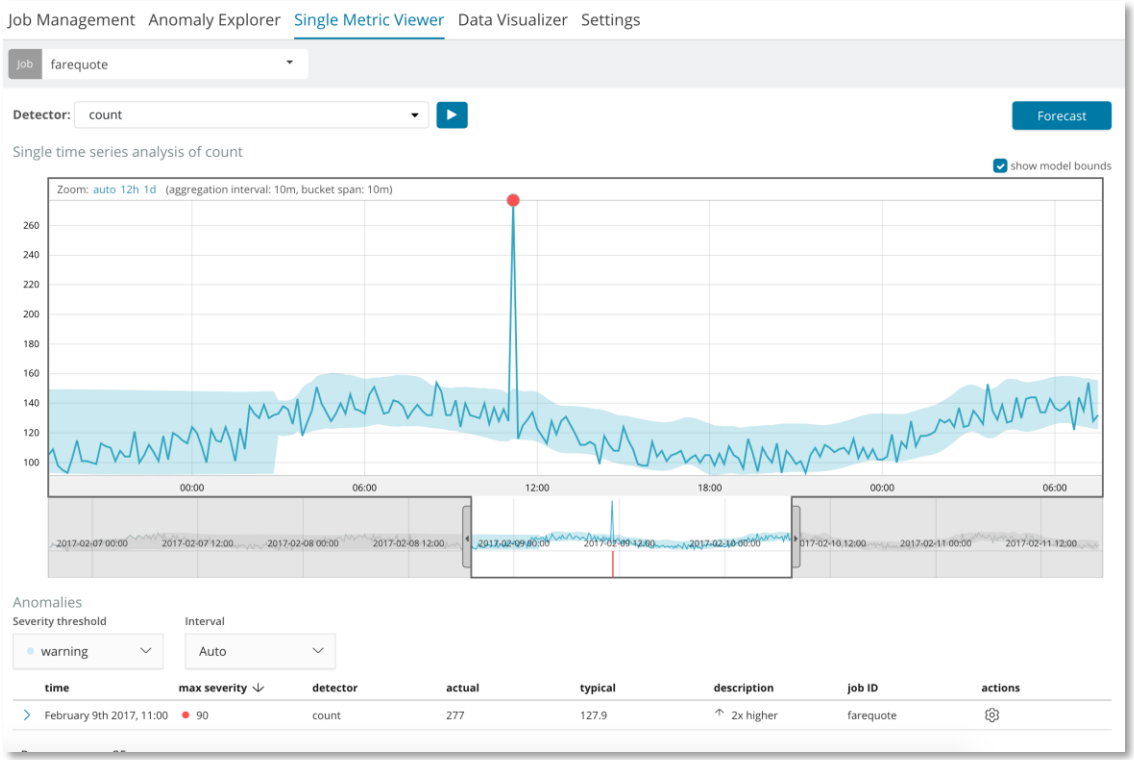
10) See animated learning

11) click “View Results”

elastic

Steps to Complete

12) Zoom in on anomaly



Splitting the Analysis

Feature Selection and Selection

- Which attributes of a this data could be used to judge its unusualness?

raw logs

```
2016/02/08 06:20:43 INFO  [http-8680]: FareQuoteImpl - FareQuoteImpl.getFare(AAL): exiting: 92.5638
2016/02/08 06:20:44 INFO  [http-8680]: FareQuoteImpl - FareQuoteImpl.getFare(JZA): exiting: 990.4628
2016/02/08 06:20:46 INFO  [http-8680]: FareQuoteImpl - FareQuoteImpl.getFare(JBU): exiting: 877.5927
...
```

ingest

document


```
{
  "_index": "farequote",
  "_type": "response",
  "_id": "AVNQ1__XRcuaRiYtw-jH",
  "_score": 3.290889,
  "_source": {
    "sourcetype": "farequote",
    "airline": "AAL",
    "responsetime": "92.5638",
    "time": "2016-02-08T06:20:43+0000"
  }
}
```


“airline” (categorical)

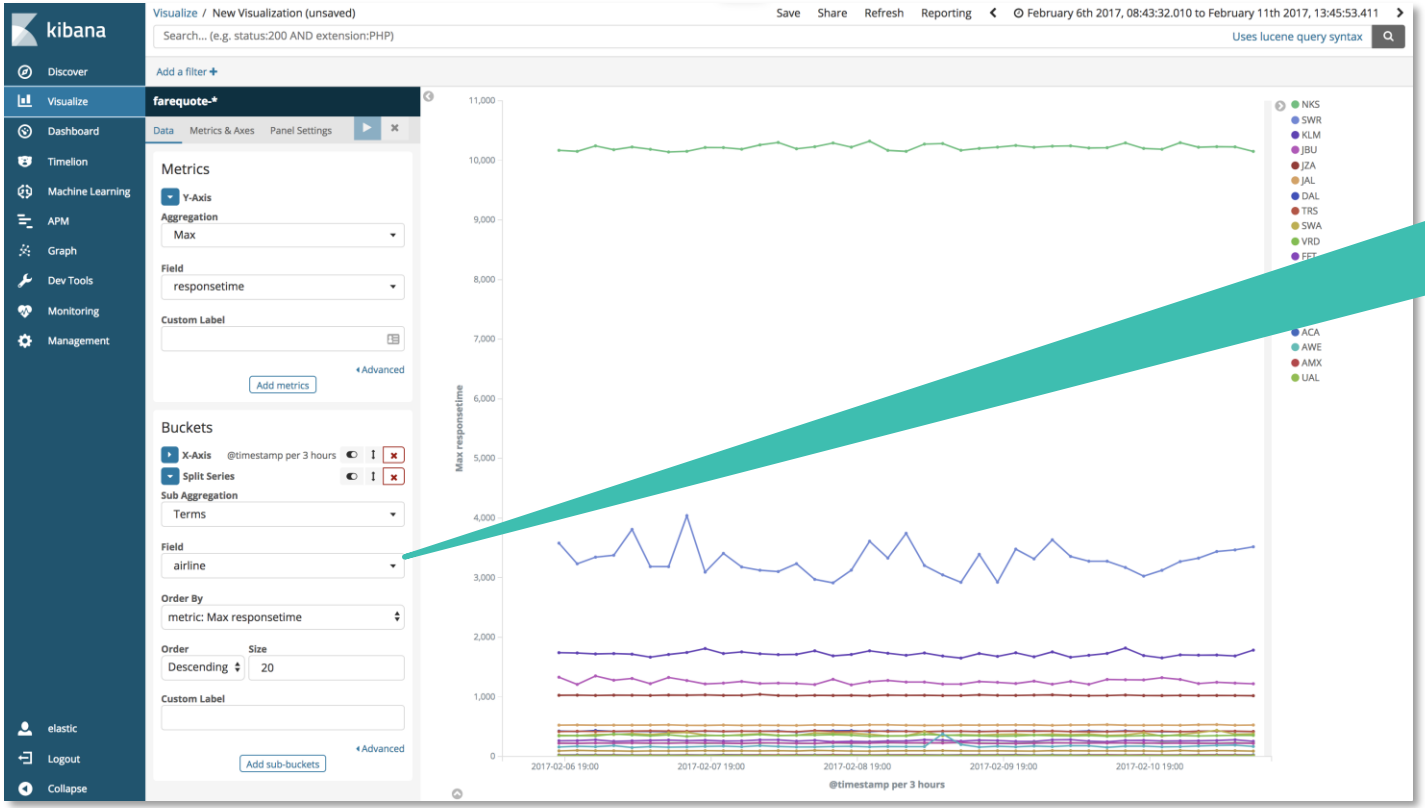
“responsetime” (metric)

elastic

What Kinds of Questions Can be Answered?

QUESTION	ANSWERABLE?
Is there an unusual ***amount*** of requests per unit time (total)?	Yes
Is there any particular airlines with unusual ***amounts*** of requests per unit time?	Yes
Is the ***total*** response time of ***all*** API calls unusually long?	Yes
Is the response time of API calls ***per airline*** unusually long?	Yes
Are there any airlines with excessive take-off delays?	No

In Kibana: How to Answer that Question



ML has the same notion of "splitting" along a categorical field

elastic

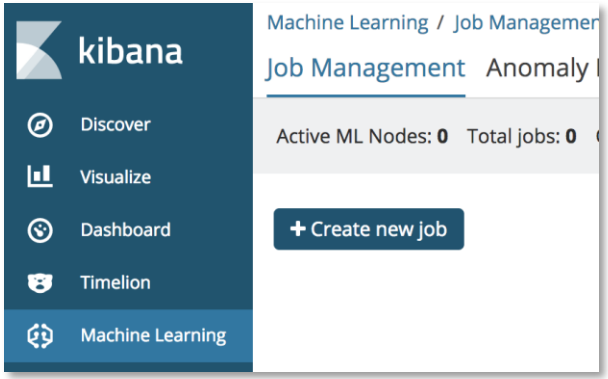
Lab 2: Multi-Metric Jobs

Goal: Detect anomalies in parallel time series

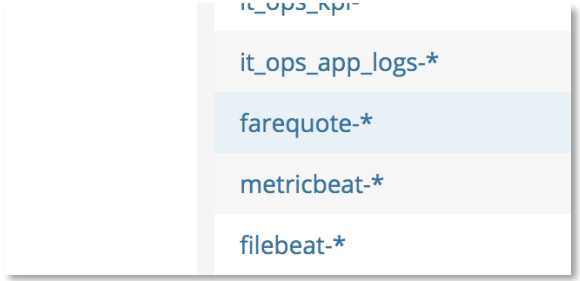
elastic

Steps to Complete

1) In Machine Learning, Create new job



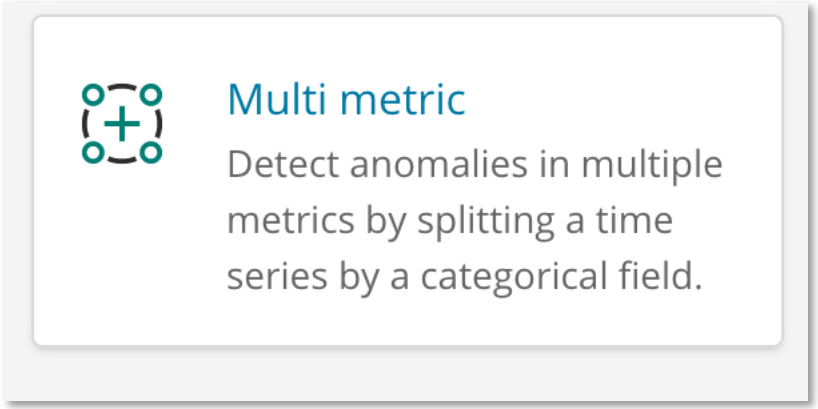
2) Choose the “farequote-*” index pattern



elastic

Steps to Complete

3) pick Multi metric Job Wizard



elastic

Steps to Complete

4) choose
- event rate, count
- responsetime, max

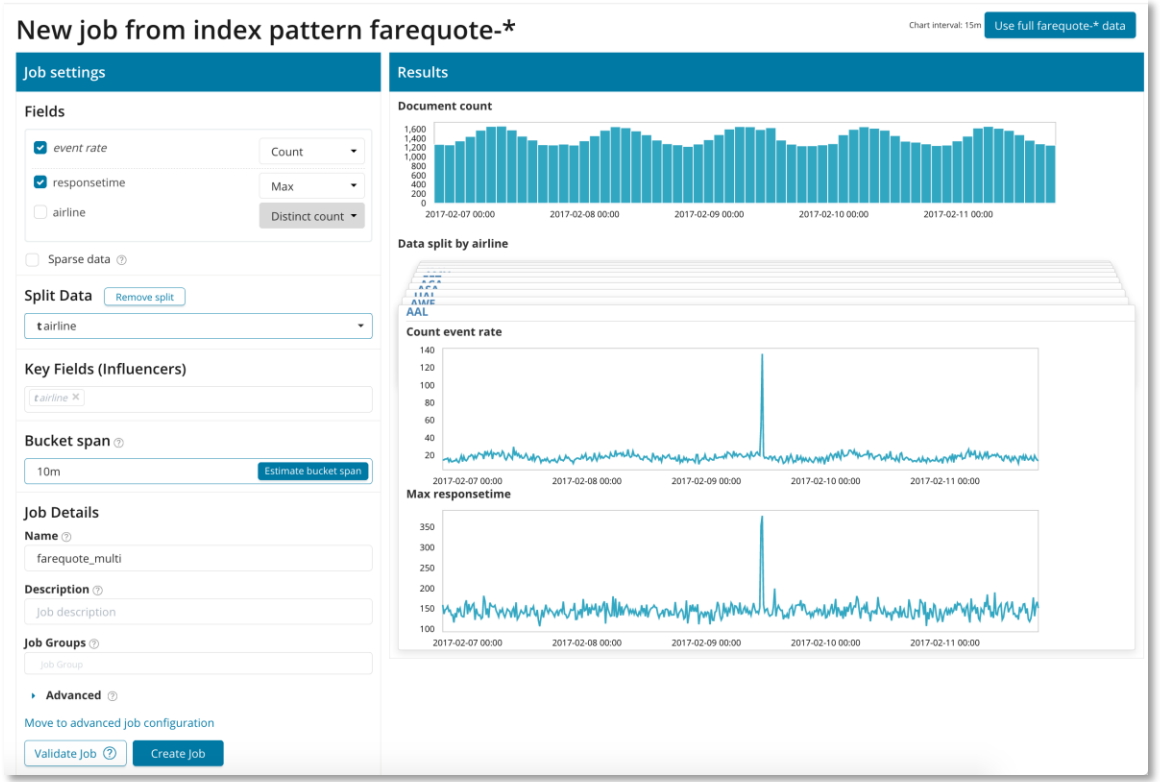
5) bucket span: 10m

6) Split Data: airline

7) click “use full farequote data”

8) Name: “farequote_multi”

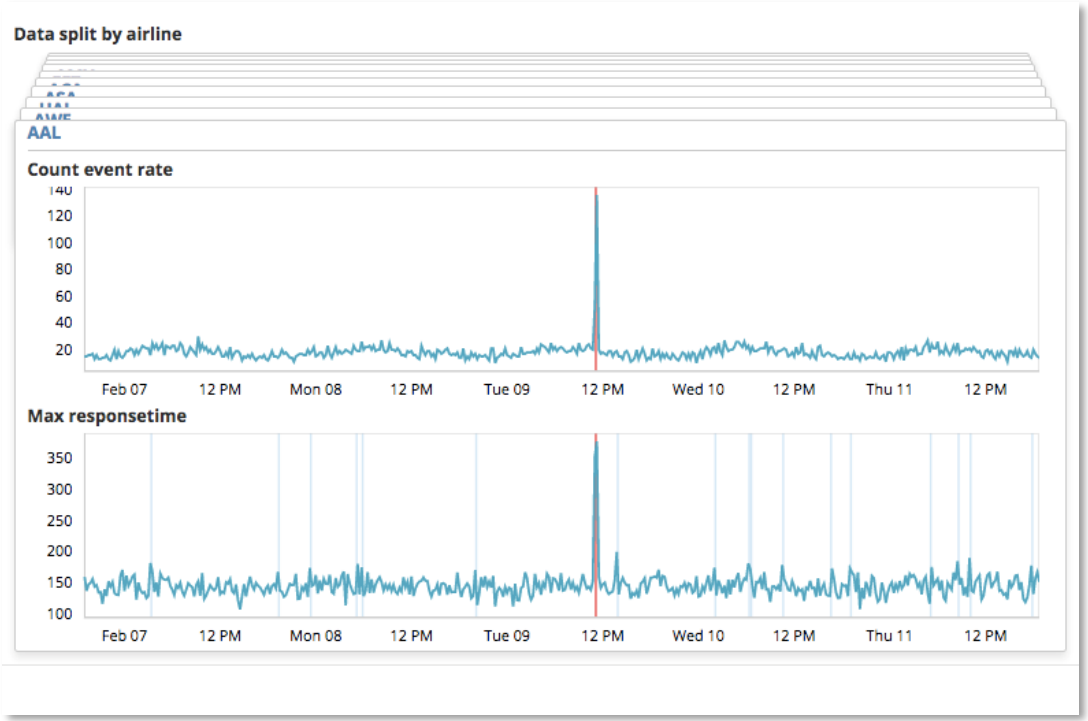
9) click “Create Job”



Steps to Complete

10) See animated learning

11) click “View Results”

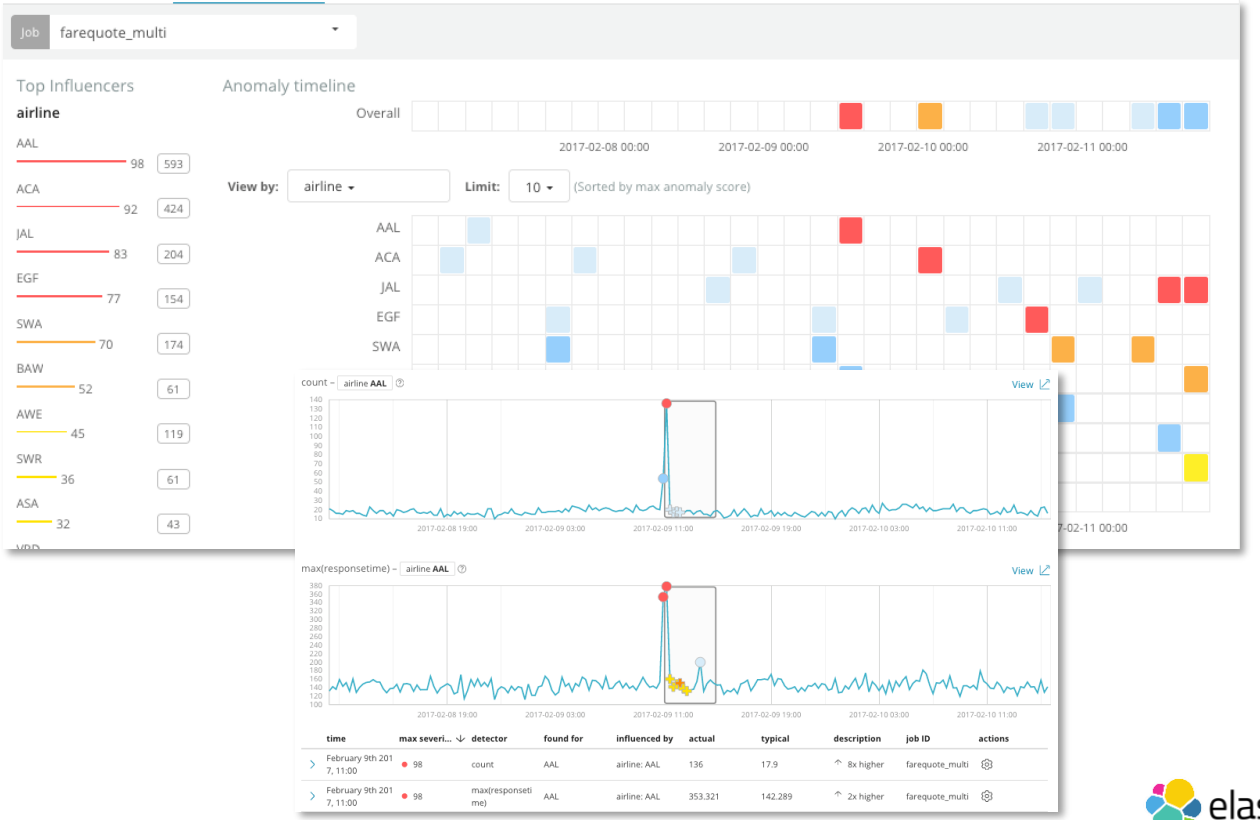


elastic

Steps to Complete

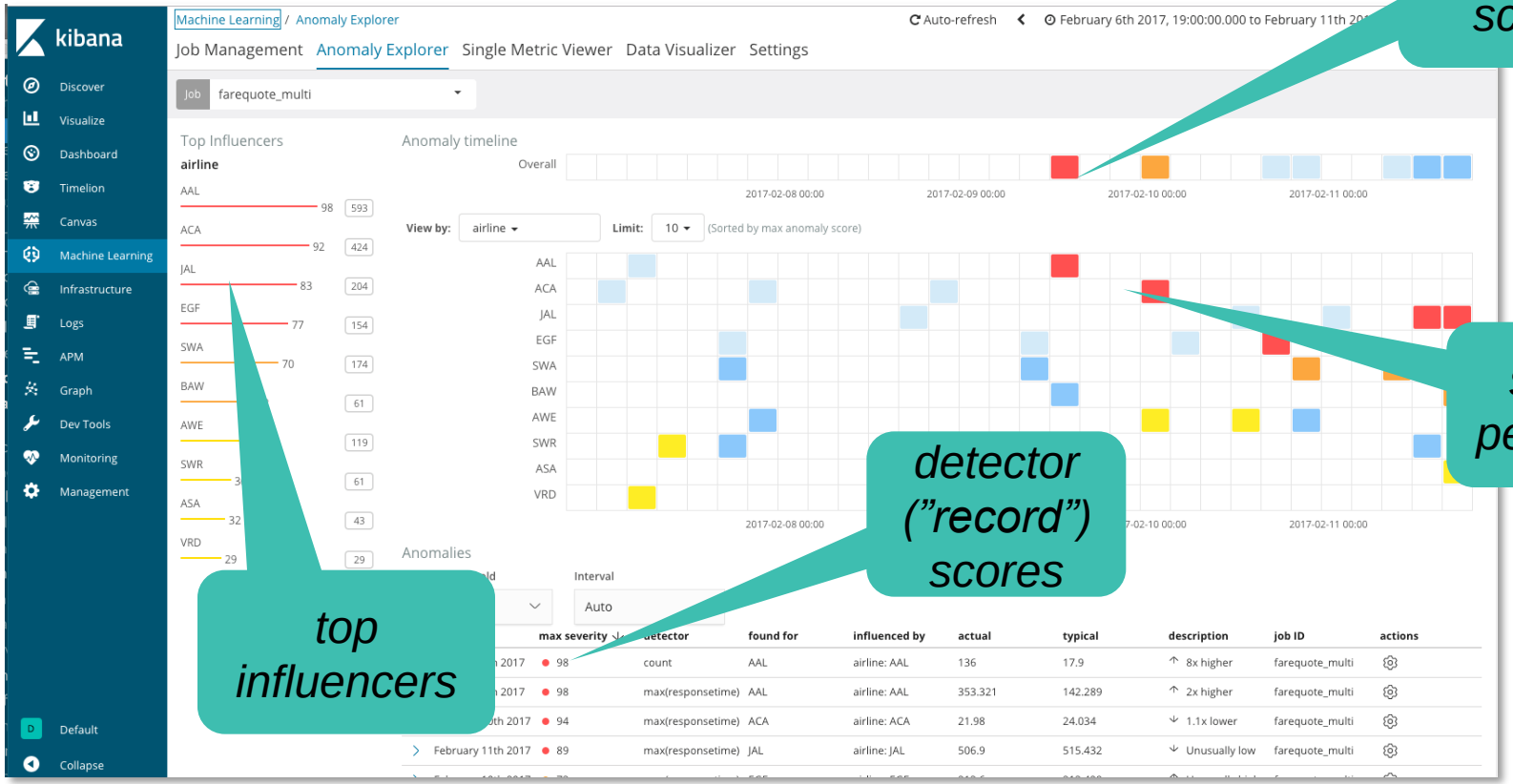
12) Result:

top unusual airline: AAL anomalies
in both count
and response time



The Anomaly Explorer View

Anomaly Explorer



elastic

Concept: What is an Influencer?

- An Influencer is a field, selected at configuration time, that would be a logical entity ”to blame” if an anomaly were to exist
- Doesn’t have to be a field in the actual detector, but fields used to split the data are often good candidates
- Will get its own score based upon how influential that entity is on the anomaly

Scoring

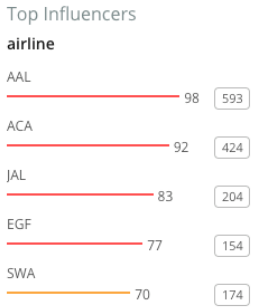
- Overall Job score is 90
  - How unusual is that bucket, given all airlines?
- Detector scores are 98
  - How unusual is the response time and count of airline=AAL?
- airline=AAL is the top influencer in this time range
  - 98 is the max score
  - 593 is the sum of all scores over the time range

Anomalies

Severity threshold: warning

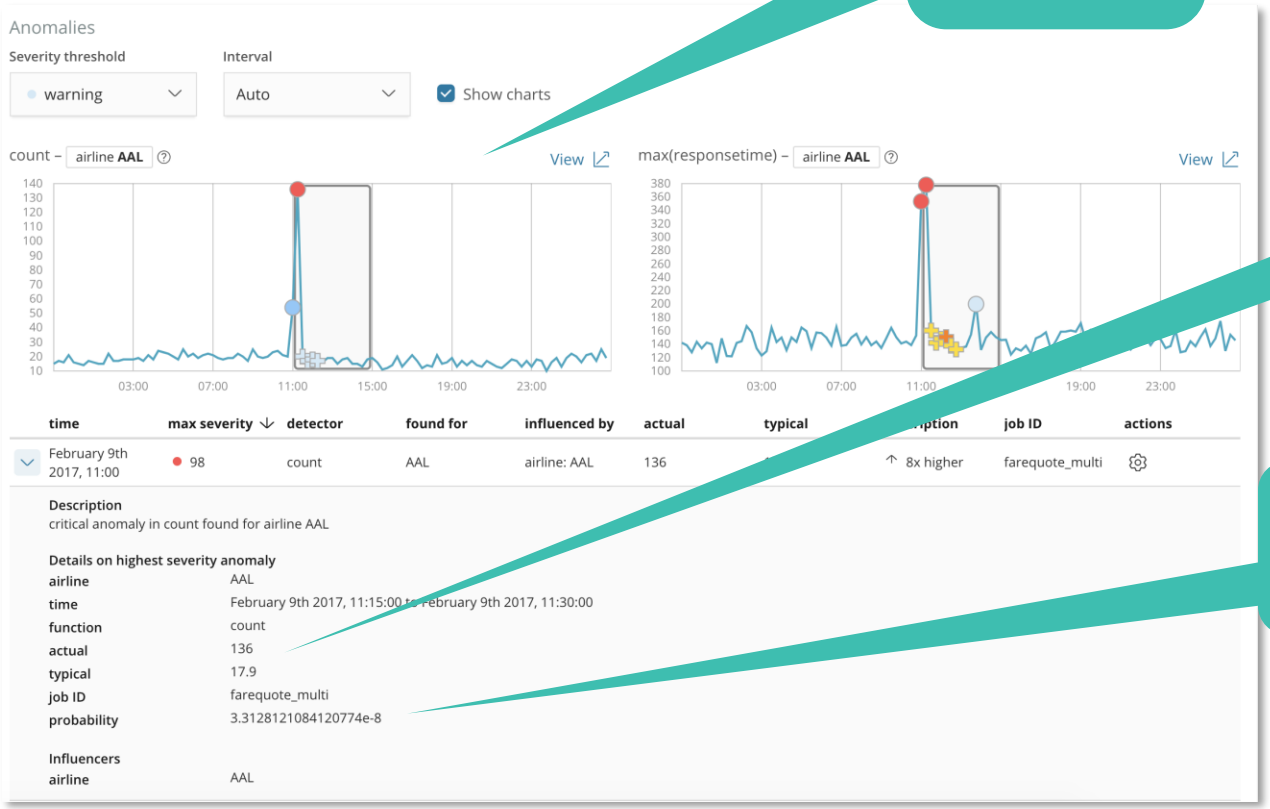
Interval: Auto

time	max severity	detector	found for	influenced by	actual	typical	description	job ID	actions
February 9th 2017	98	count	AAL	airline: AAL	136	17.9	8x higher	farequote_multi	
February 9th 2017	98	max(responsetime)	AAL	airline: AAL	353.321	142.289	2x higher	farequote_multi	



elastic

Anomaly Details



view individual anomalies

actual vs. "typical"

raw probability

elastic

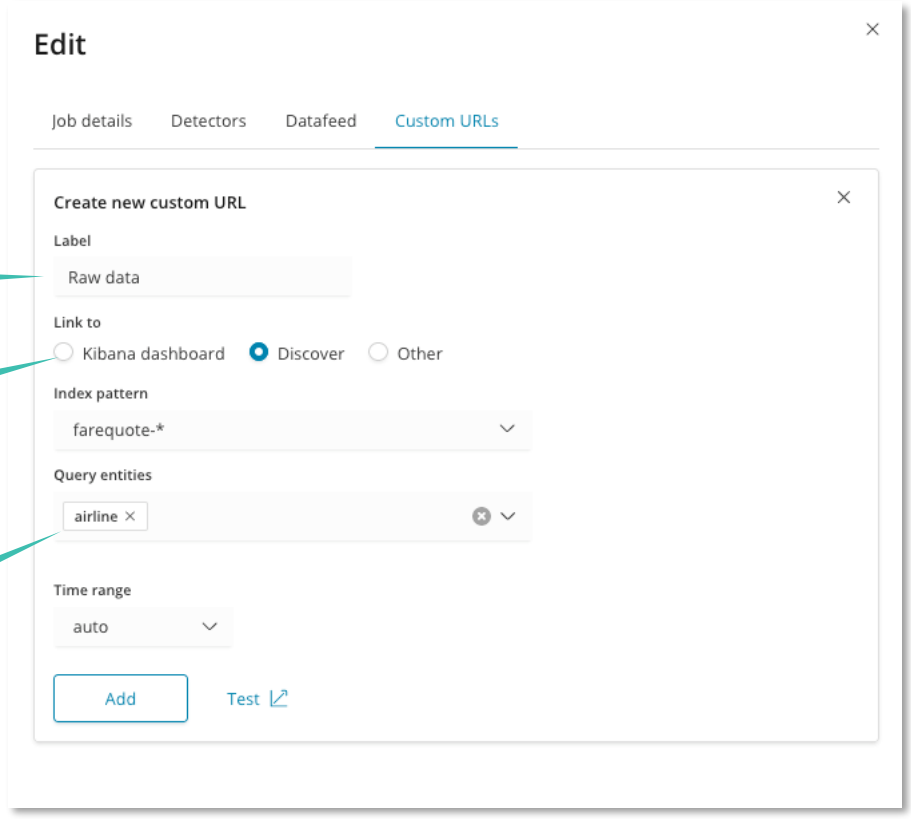
Custom URLs: Linking to other places, in context

Custom URLs in Job config

Name the label

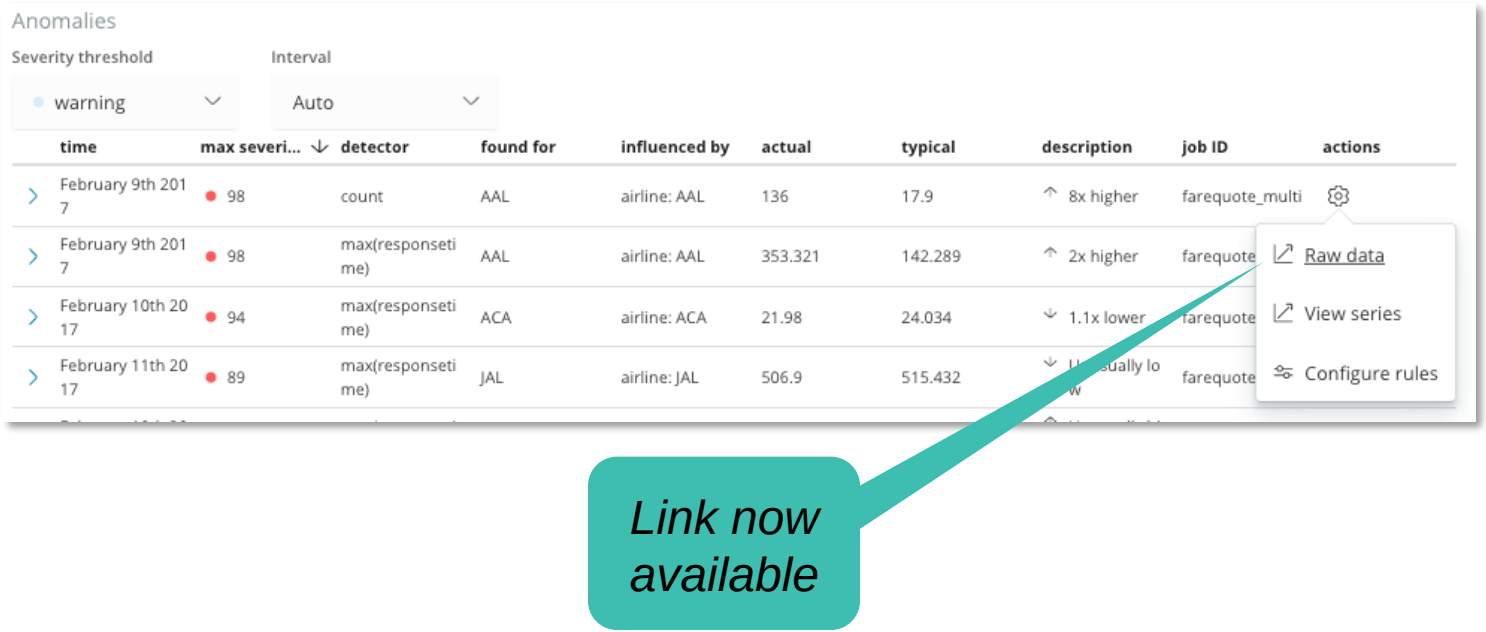
where to link

fields for filtering

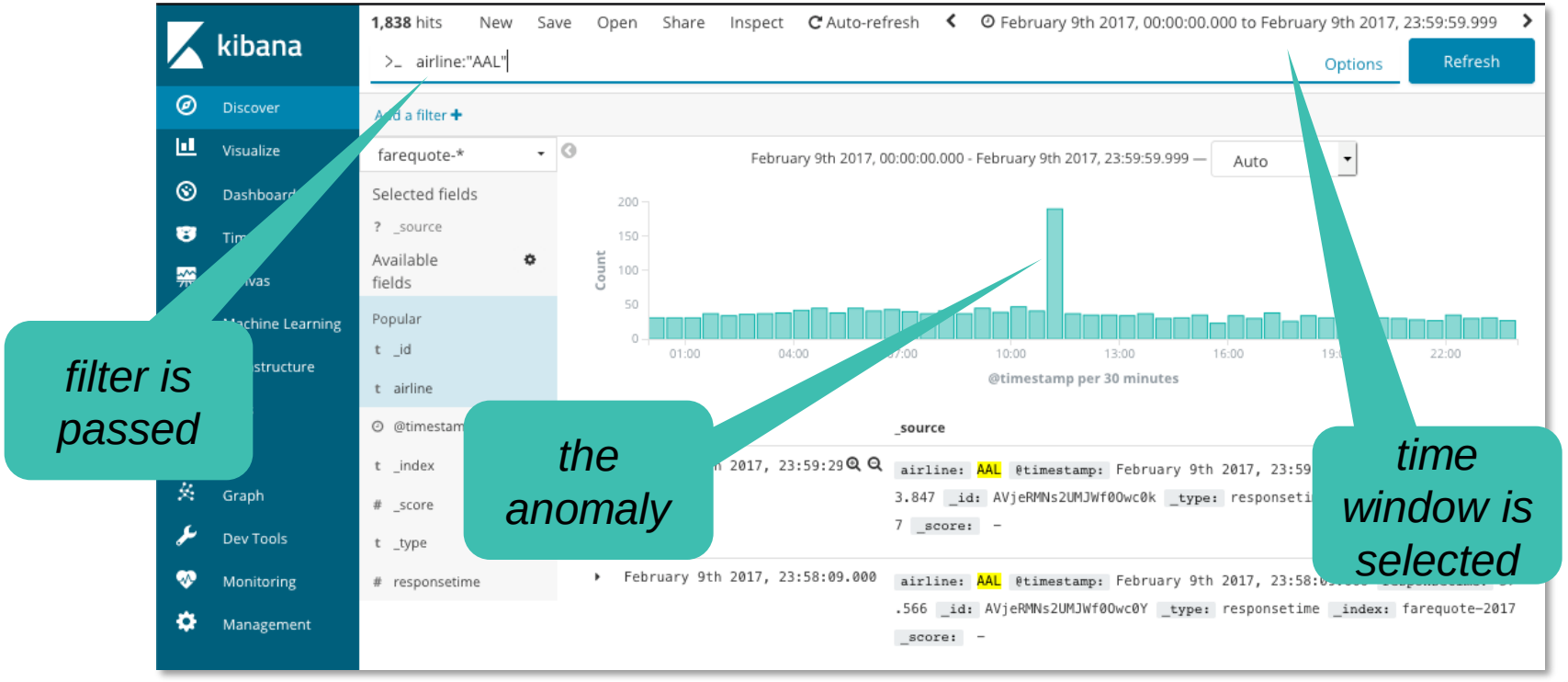


elastic

In Anomaly Explorer



Clicking on Custom URL, passing context

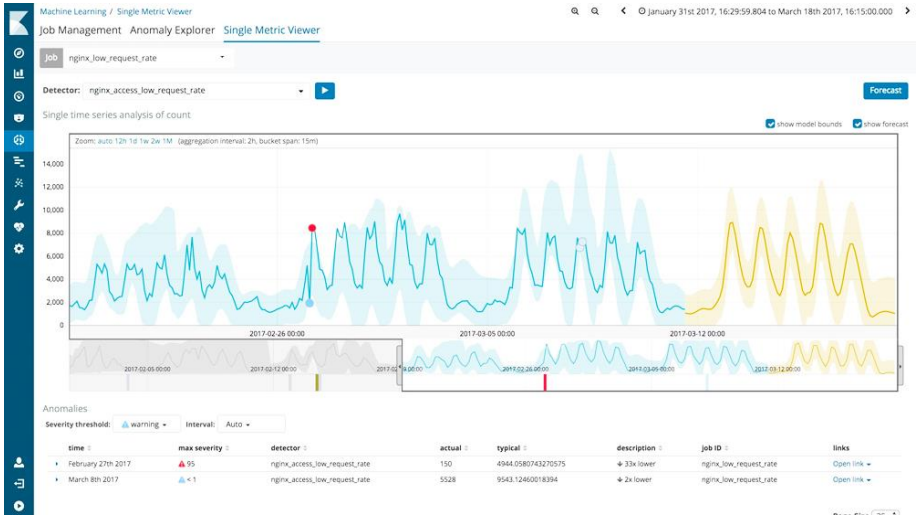


elastic

Forecasting

elastic

Extrapolate into the future with forecasting



Example use cases

- I want to know when capacity will exceed 80%

- What is my expected visitor count next Saturday at 8am?

elastic

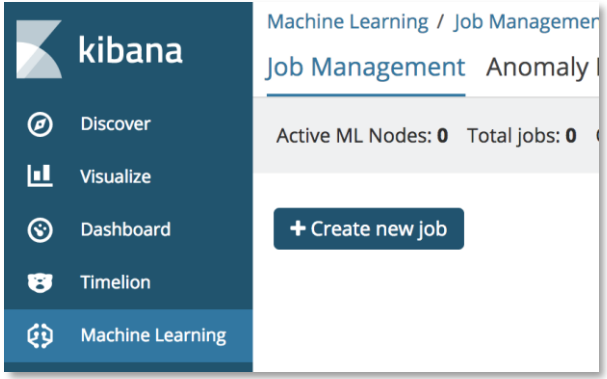
Lab 3: Forecasting

Goal: Extrapolate into the future

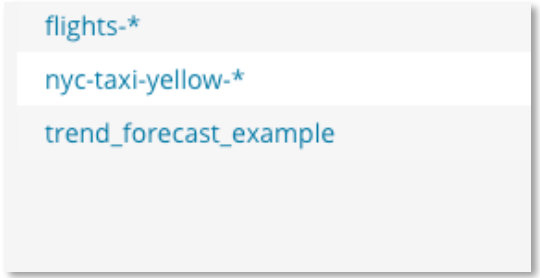
elastic

Steps to Complete

1) In Machine Learning, Create new job



2) Choose the “trend_forecast_example” index pattern



elastic

Steps to Complete

3) single metric job

4) sum of field “amount”

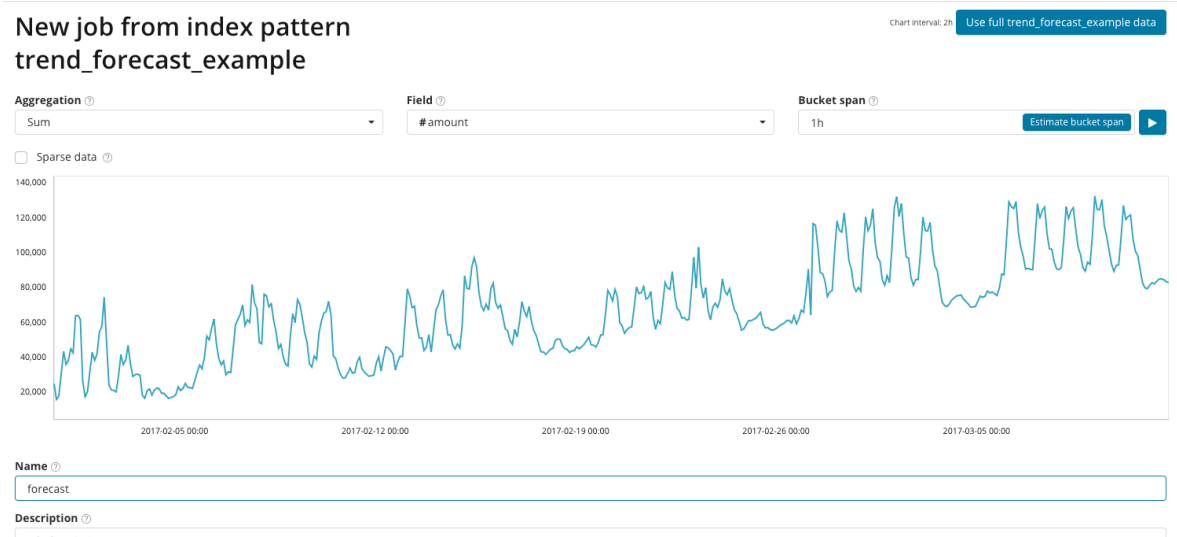
5) bucket span: 1h

6) click “use full data”

7) Name: “forecast”

8) click “Create Job”

9) click “View Results”



elastic

Steps to Complete

10) click “Forecast” button

11) Enter “7d” for duration

12) click “Run” button

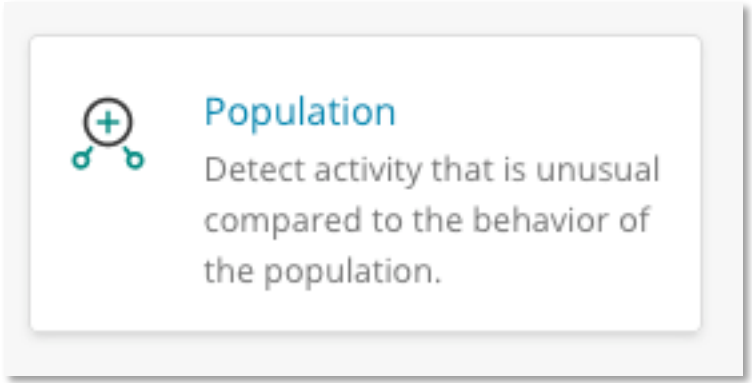
13) see forecast in UI
(results also available via API)

elastic

Population Analysis

Population Analysis

- Use when:
  - Compare entities against peers
  - Not against own history
- Helpful when:
  - Entities have high-cardinality (i.e. external IP addresses)
  - Data for specific entities may be sparse in time (individual customers placing orders)
  - The behavior of the population as a whole is mostly homogeneous
- Not appropriate when:
  - Members of the population have vastly different behavior inherently.



elastic

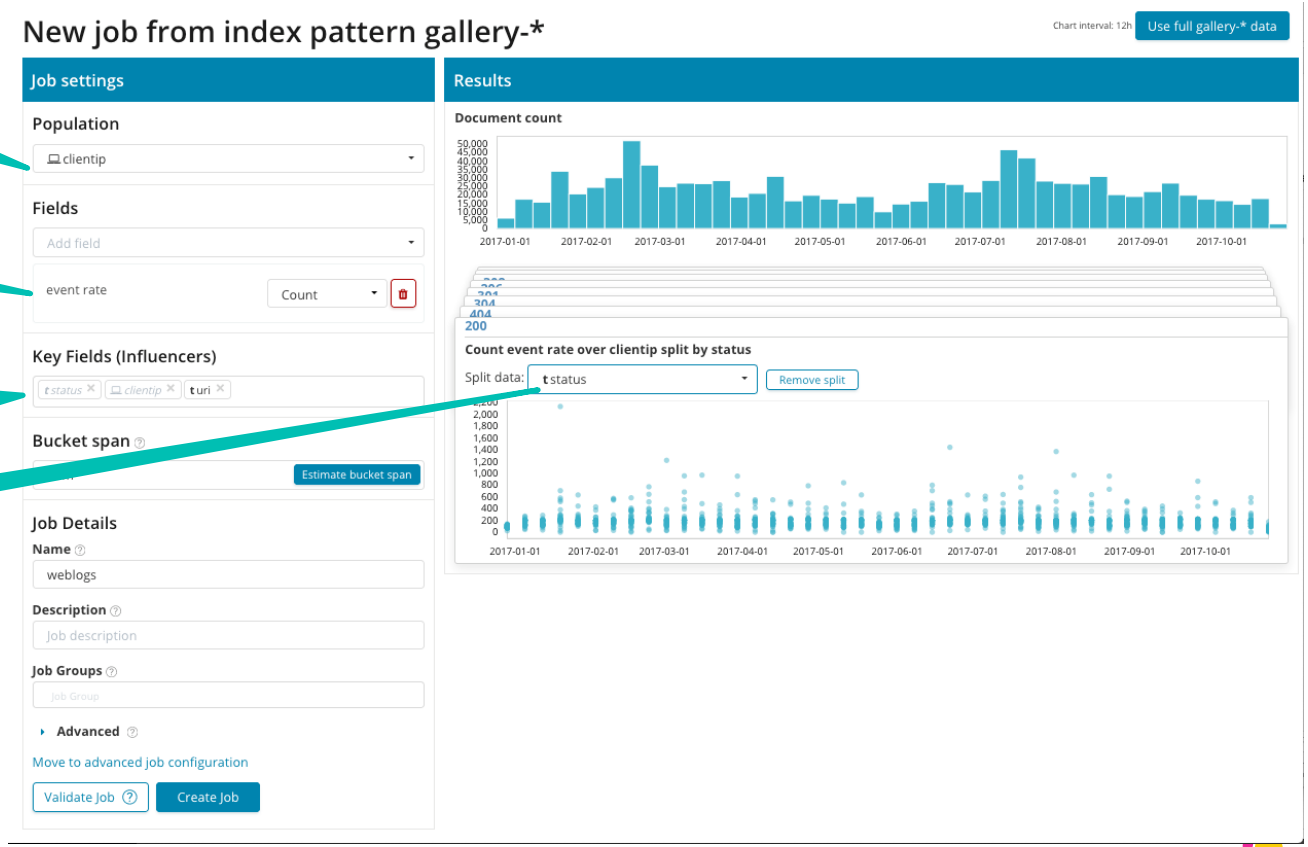
Population Analysis

population field

the detector

influencers

optional split



elastic

Population Analysis

clientip: 173.203.78.60
status: 404
uri: /wp-login.php

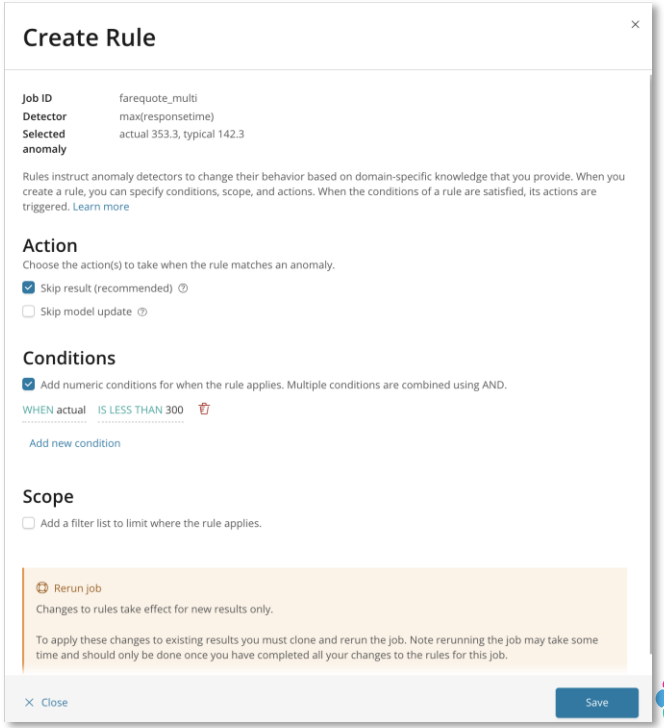
Custom Rules

Adding Custom Rules to ML
(v6.4+)

Define rules that:
- Ignore the creation of anomalies
- Disqualify data from being modeled

Meet certain conditions:
- on “actual”, ”typical” or difference

Optional scoping
- limit when rule is applied



elastic

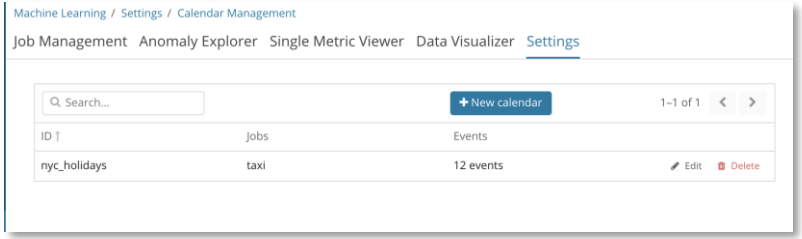
Calendars

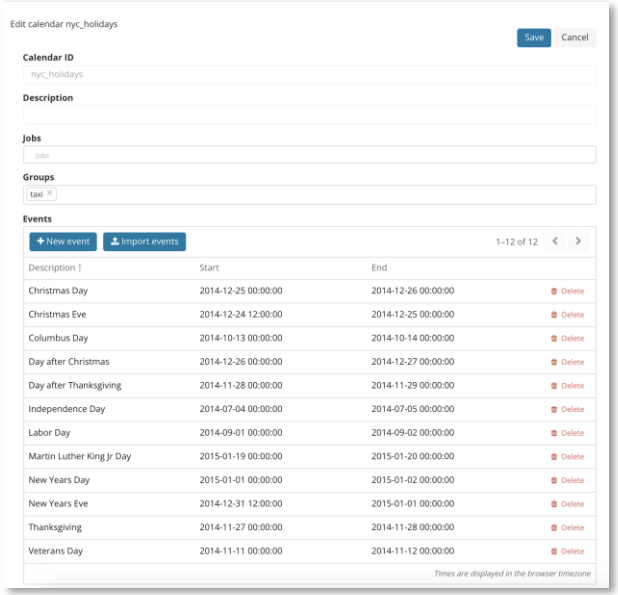
Ignoring Timeframes with Calendars

Define timeframes for ML to
- not create anomalies
- not allow model to see new data

Good for known, upcoming times

Calendars can apply to individual jobs or job groups

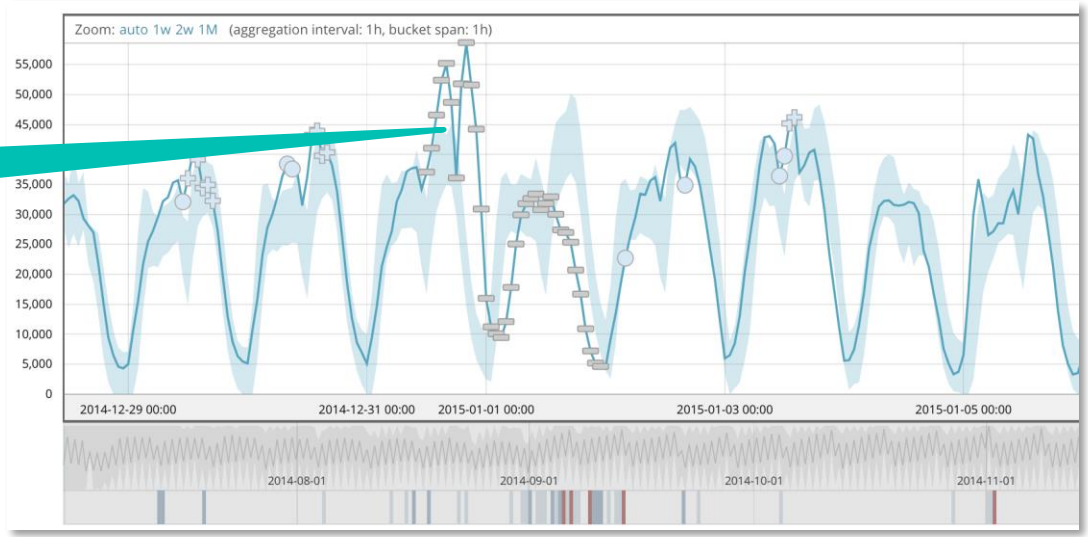




elastic

Ignoring Timeframes with Calendars

Ignored times marked with "—" symbol

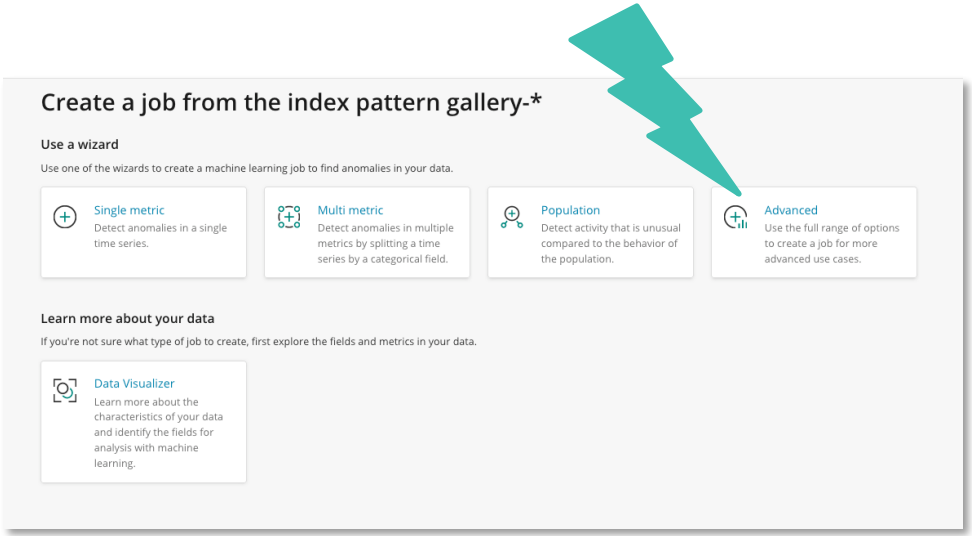


elastic

Advanced Jobs

Advanced Jobs

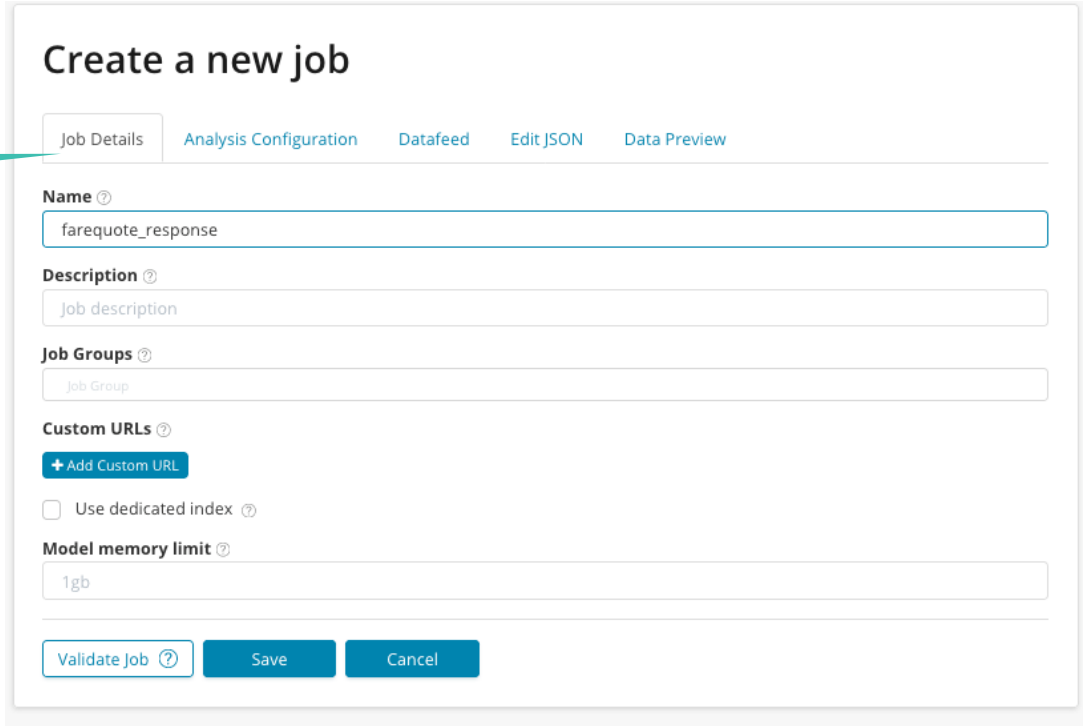
- Gives more flexibility
- Access to additional capabilities
- Requires more understanding of configuration parameters



elastic

Configuration of Advanced Job

Tabs of different information

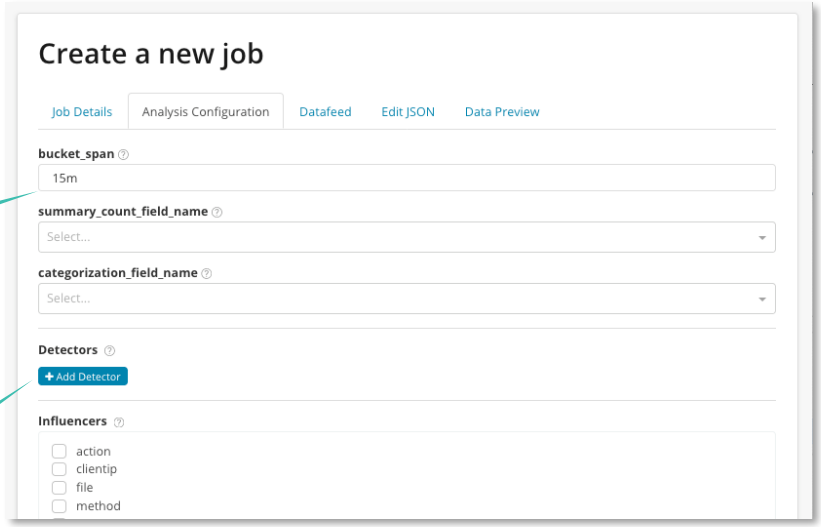


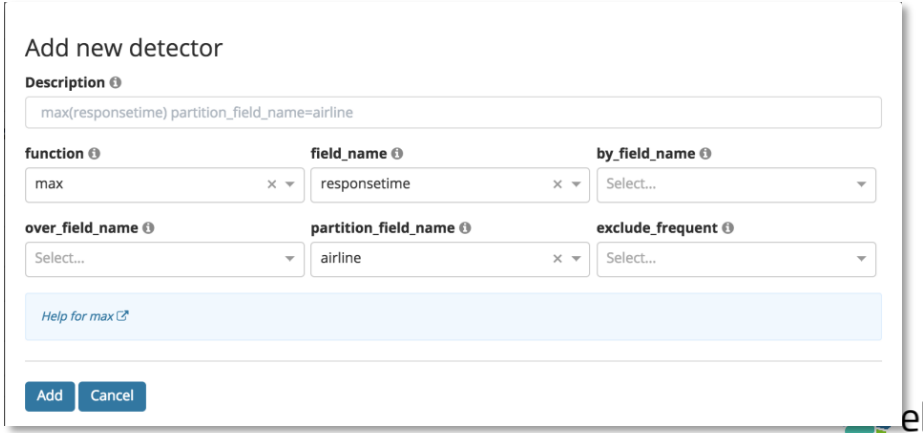
elastic

Analysis Configuration

Settting of bucket_span

Creation of "detector"

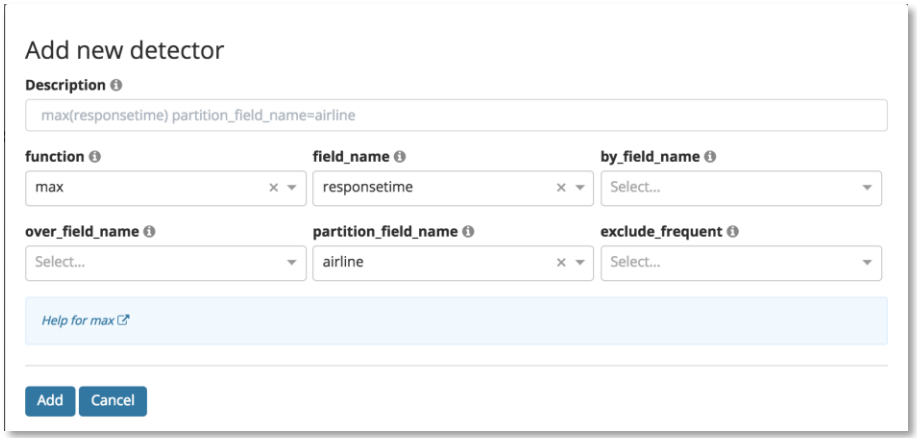




elastic

Detector Details

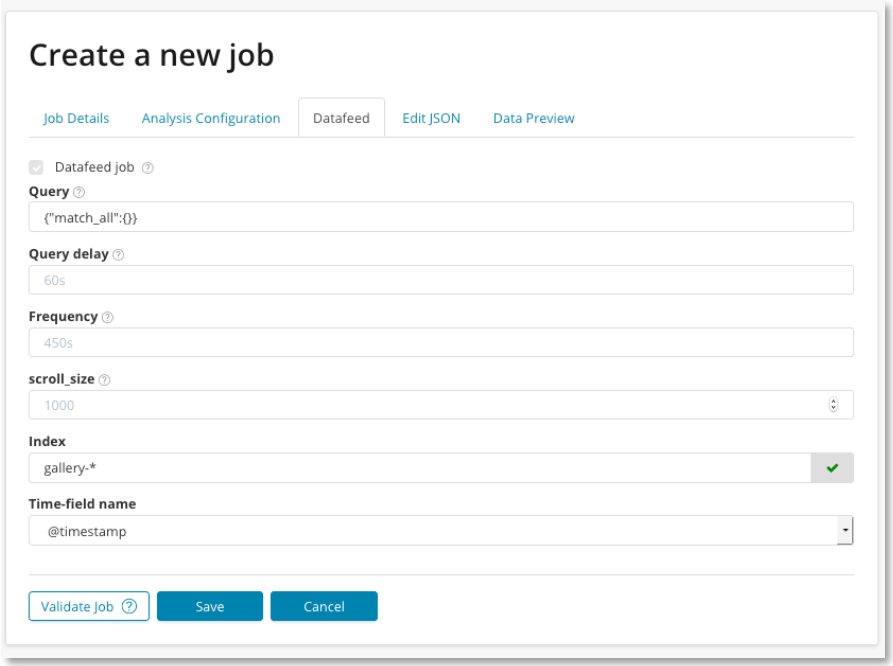
- **function** – min, max, mean, etc.
- **field_name** –field **function** “operates on”
- **by_field_name** –“dependent” split
- **partition_field_name** –“independent” split
- **over_field_name** – defines a population



elastic

Datafeed Details

- Datafeed manages how ML queries Elasticsearch
- See ML docs for configuration options



elastic

Other Types of Analysis

Rarity Analysis

Rare Analysis

- Finding items that rarely occur is also often useful
  - Rarely occurring log messages
  - Rare running process names
  - Rare connection destinations
- ML has a rare function, but it should be noted that:
  - It is relative, i.e. it takes into account the frequency of other field values

A,B,A,B,A,C,B,A,B,A,C,A,B,C,A,C,B,A,C,B,A,C,X <=X is rare
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X <= X is not rare

  - Therefore it works best when there are plenty of routine messages to contrast the rare ones

Example of Rare Analysis

- Use Case: Security team @ services company
- Wanted to profile typical processes on each host using netstat

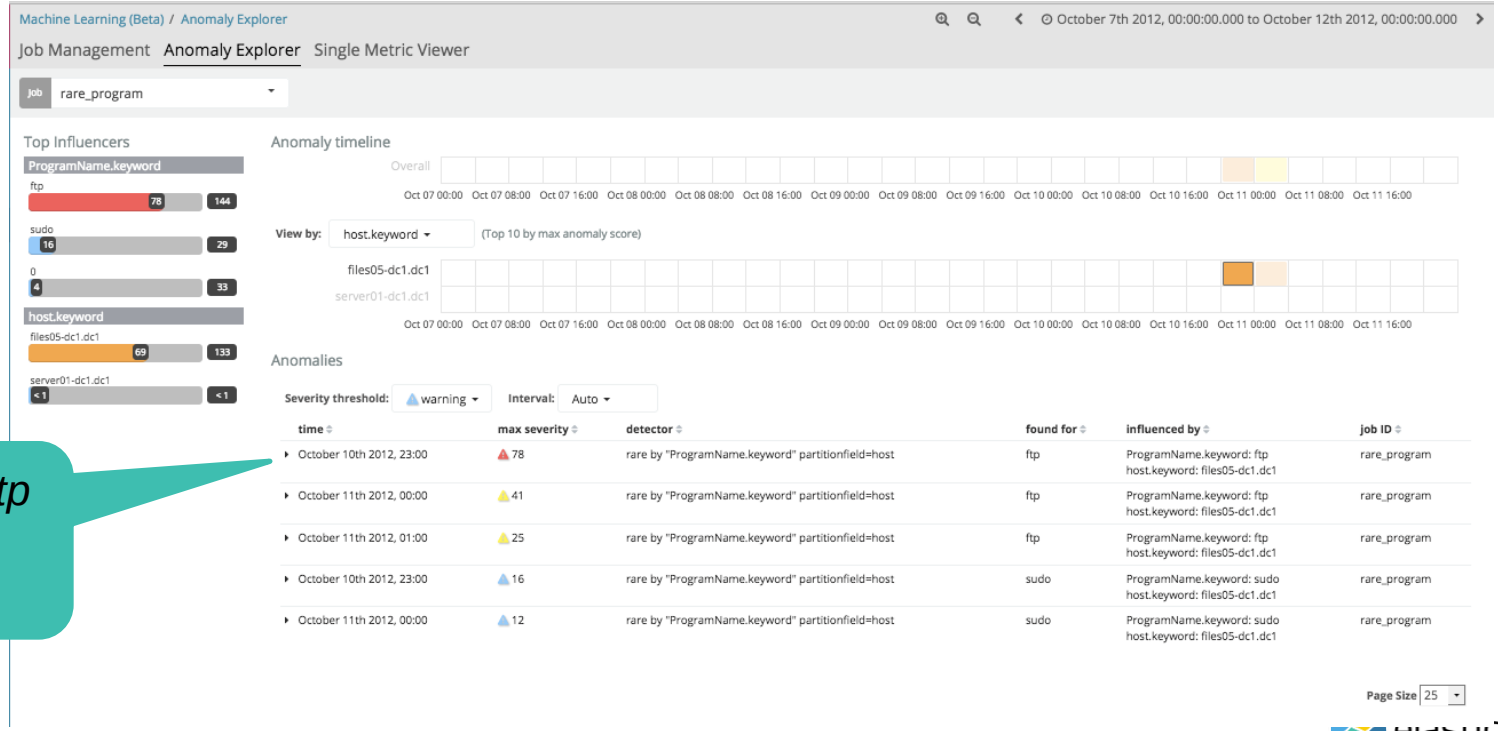
```
Active Internet connections (servers and established)
(index=netstat host="ids01-dc2" State=LISTEN (7/16/13 1:32:51AM))
Proto Recv-Q Send-Q Local Address               Foreign Address             State       PID/Program name
tcp        0      0  10.220.174.41:561         0.0.0.0:*                   LISTEN      8776/argus
tcp        0      0  0.0.0.0:22                0.0.0.0:*                   LISTEN      2033/sshd
tcp        0      0  0.0.0.0:1241              0.0.0.0:*                   LISTEN      4472/nessusd
tcp        0      0  0.0.0.0:8089              0.0.0.0:*                   LISTEN      4238/splunkd
tcp        0      0  127.0.0.1:25              0.0.0.0:*                   LISTEN      4194/master
tcp        0      0  0.0.0.0:8000              0.0.0.0:*                   LISTEN      4361/python
tcp        0      0  0.0.0.0:8834              0.0.0.0:*                   LISTEN      4472/nessusd
tcp        0      0  127.0.0.1:199             0.0.0.0:*                   LISTEN      2022/snmpd
```

- Goal was to identify rare processes that “start up and communicate” for each host, individually

elastic

Example of Rare Analysis

detector: *rare by ProgramName partition_field=host*



ProgramName: ftp
host: files05
rarely occurs

Categorization

Categorization

- What is it?
  - Automatically grouping similar log messages into the same “category”
- How?
  - Via an algorithm that looks at string similarity and clusters similar log lines
- Why?
  - Bring structure to semi-structured data so that it can be analyzed

Benefit: Find anomalies in logs without knowing what they contain

Example

```
Oct 22 18:17:58 localhost sshd[8903]: Invalid user admin from 41.43.112.199 port 41805
Oct 22 18:17:58 localhost sshd[8903]: input_userauth_request: invalid user admin [preauth]
Oct 22 18:17:59 localhost sshd[8903]: Connection closed by 41.43.112.199 port 41805 [preauth]
Oct 22 20:58:03 localhost sshd[2074]: Received disconnect from 72.93.85.203 port 53552:11: disconnected by user
Oct 22 20:58:03 localhost sshd[2074]: Disconnected from 72.93.85.203 port 53552
Oct 22 20:58:03 localhost sshd[2072]: pam_unix(sshd:session): session closed for user ec2-user
Oct 22 21:32:54 localhost sshd[8944]: pam_unix(sshd:session): session opened for user ec2-user by (uid=0)
Oct 22 21:35:15 localhost runuser: pam_unix(runuser-l:session): session closed for user ec2-user
Oct 22 21:35:15 localhost runuser: pam_unix(runuser-l:session): session opened for user ec2-user by ec2-user(uid=0)
Oct 22 21:35:16 localhost runuser: pam_unix(runuser-l:session): session closed for user ec2-user
```

Step 1 – remove mutable text

```
Oct 22 18:17:58 localhost sshd[8903]: Invalid user admin from 41.43.112.199 port 41805
Oct 22 18:17:58 localhost sshd[8903]: input_userauth_request: invalid user admin [preauth]
Oct 22 18:17:59 localhost sshd[8903]: Connection closed by 41.43.112.199 port 41805 [preauth]
Oct 22 20:58:03 localhost sshd[2074]: Received disconnect from 72.93.85.203 port 53552:11: disconnected by user
Oct 22 20:58:03 localhost sshd[2074]: Disconnected from 72.93.85.203 port 53552
Oct 22 20:58:03 localhost sshd[2072]: pam_unix(sshd:session): session closed for user ec2-user
Oct 22 21:32:54 localhost sshd[8944]: pam_unix(sshd:session): session opened for user ec2-user by (uid=0)
Oct 22 21:35:15 localhost runuser: pam_unix(runuser-l:session): session closed for user ec2-user
Oct 22 21:35:15 localhost runuser: pam_unix(runuser-l:session): session opened for user ec2-user by ec2-user(uid=0)
Oct 22 21:35:16 localhost runuser: pam_unix(runuser-l:session): session closed for user ec2-user
```

elastic

Step 2 – cluster similar messages together

Oct 22 18:17:58 localhost sshd[8903]: Invalid user admin from 41.43.112.199 port 41805

Oct 22 18:17:58 localhost sshd[8903]: input_userauth_request: invalid user admin [preauth]

Oct 22 18:17:59 localhost sshd[8903]: Connection closed by 41.43.112.199 port 41805 [preauth]

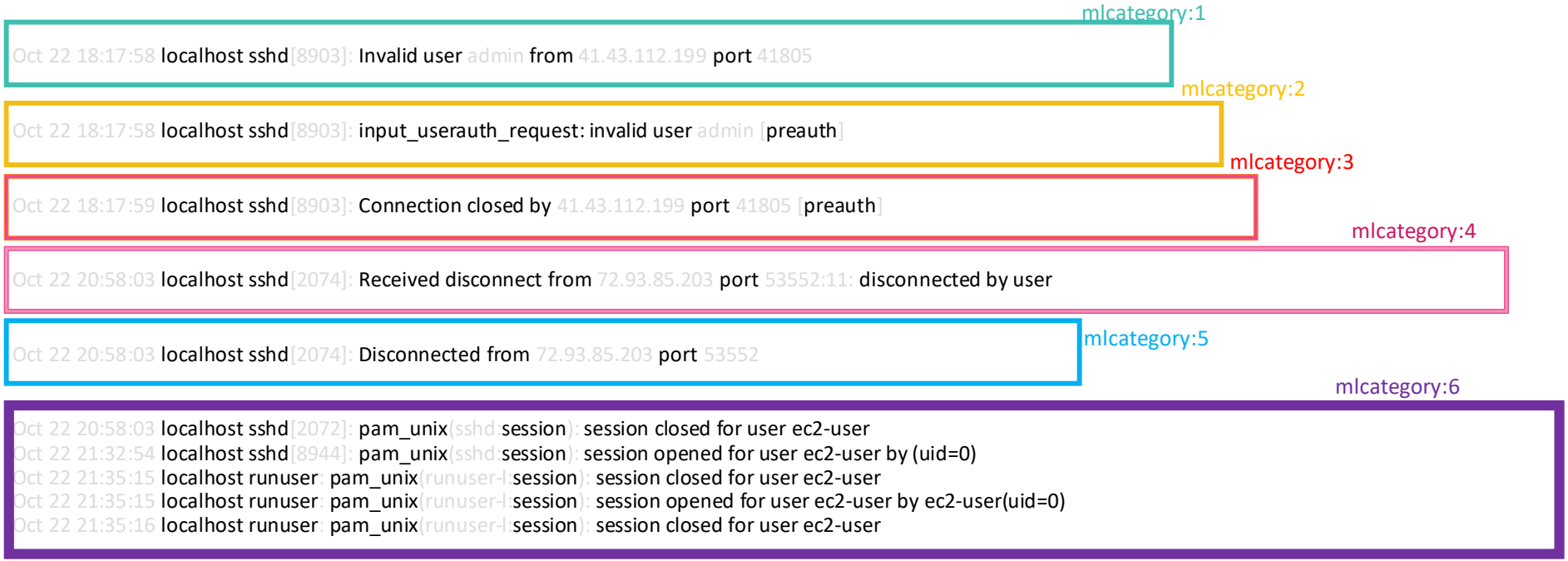
Oct 22 20:58:03 localhost sshd[2074]: Received disconnect from 72.93.85.203 port 53552:11: disconnected by user

Oct 22 20:58:03 localhost sshd[2074]: Disconnected from 72.93.85.203 port 53552

Oct 22 20:58:03 localhost sshd[2072]: pam_unix(sshd:session): session closed for user ec2-user
Oct 22 21:32:54 localhost sshd[8944]: pam_unix(sshd:session): session opened for user ec2-user by (uid=0)
Oct 22 21:35:15 localhost runuser: pam_unix(runuser-l:session): session closed for user ec2-user
Oct 22 21:35:15 localhost runuser: pam_unix(runuser-l:session): session opened for user ec2-user by ec2-user(uid=0)
Oct 22 21:35:16 localhost runuser: pam_unix(runuser-l:session): session closed for user ec2-user

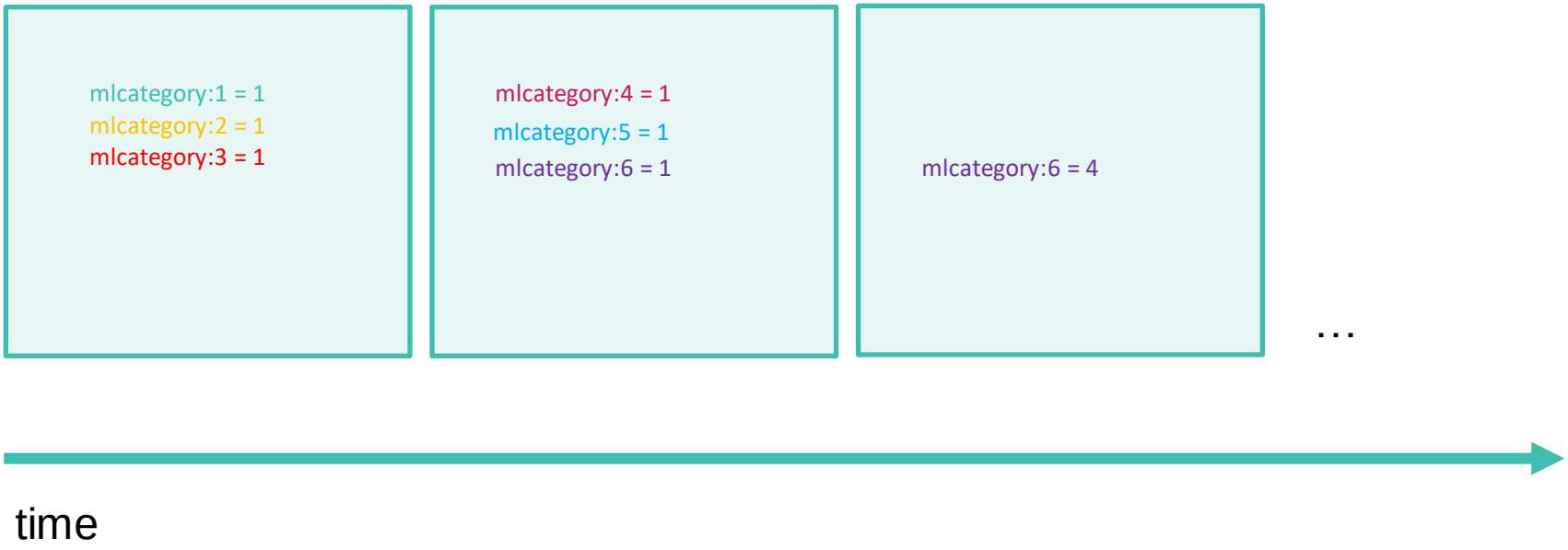
elastic

Step 2 – cluster similar messages together



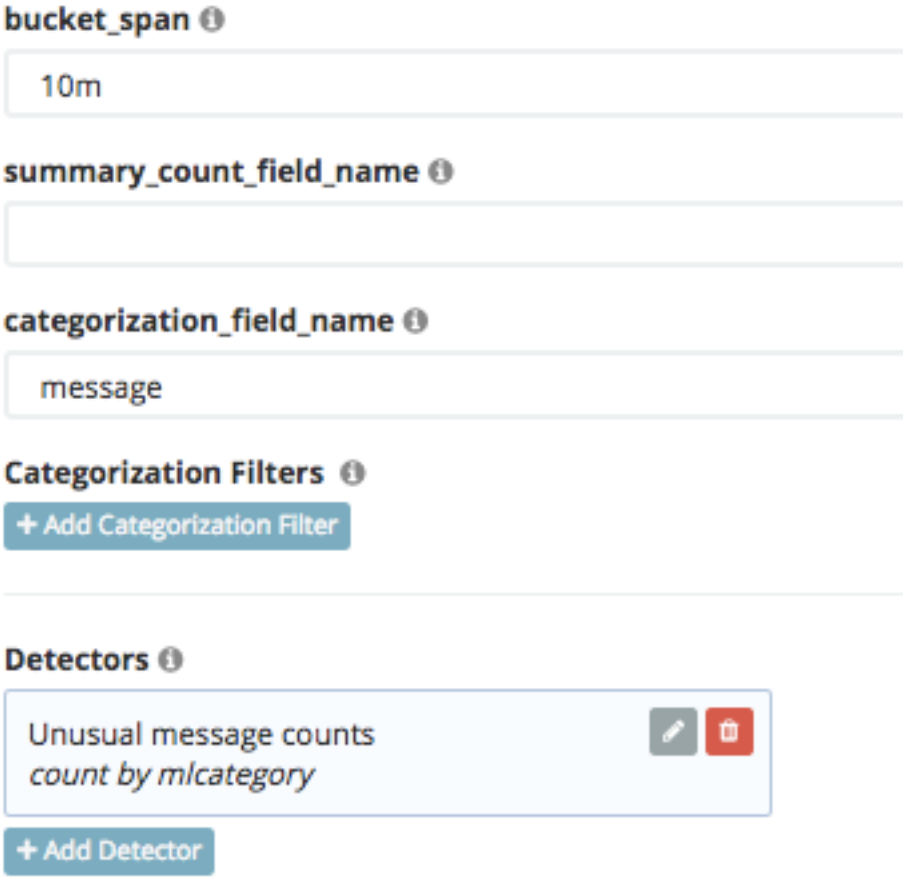
elastic

Step 3 – count per time bucket

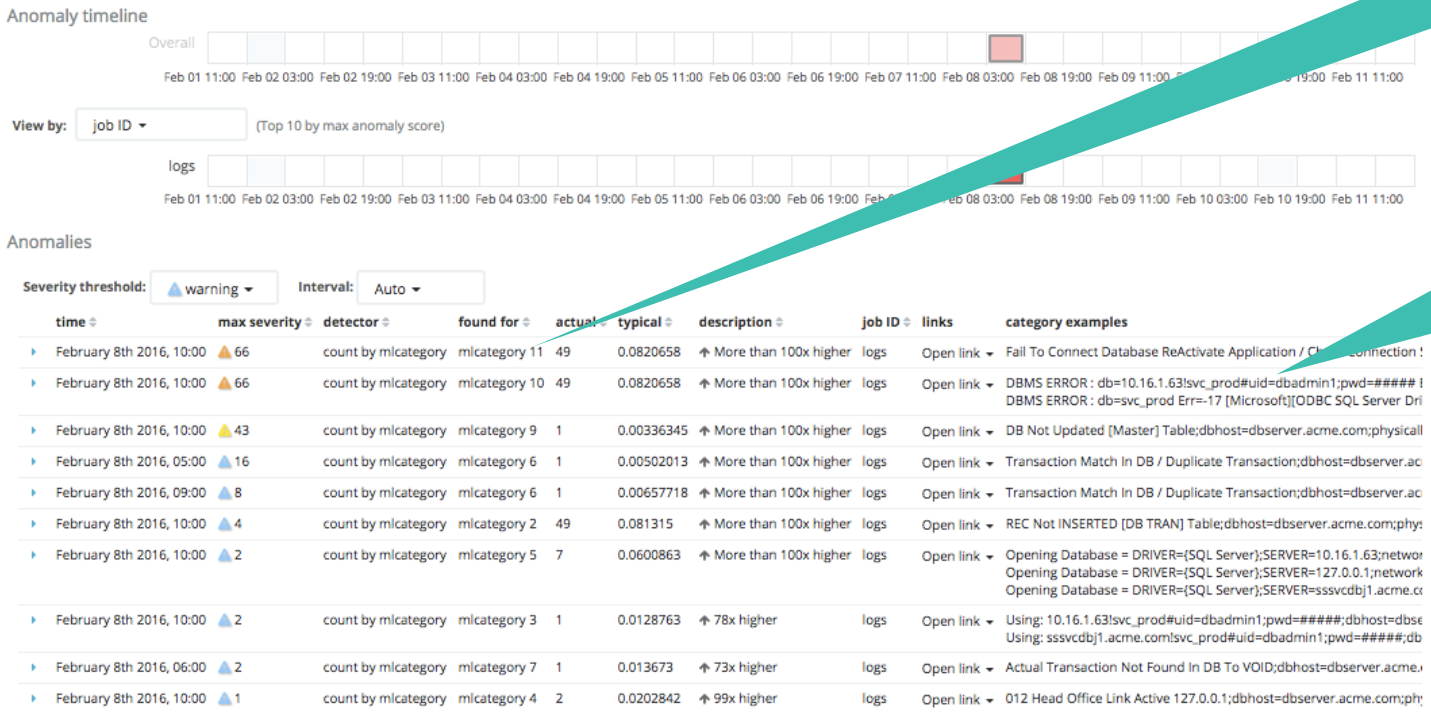


Categorization Example

- Configure an ML job to use:
  - “message” as the categorization_field_name
  - there will be a new, “magic” field called “mlcategory” that is dynamically created by ML to group similar messages together



Categorization Example – count by mlcategory



category name

example matching log messags

elastic

Lab 4: Choose your own Adventure

elastic

Options

- Track 1: Root Cause Analysis (the 4 it_ops-* indices)
  - Analyze logs and metrics together to solve a real-world problem
- Track 2: Analyze Web logs (gallery-* index)
  - Look for a brute-force authentication attack
- Track 3: Analyze NYC Taxi data (nyc-taxi-yellow-* index)
  - Find anomalies in taxicab ridership

Lab 4, Track 1: Root Cause Analysis

elastic

Lab 4: Track 1 - Problem

- Transaction processing app records # transactions per minute in an index called “it_ops_kpi-*”. Find the anomaly in it.
- Also, SQL Database and Network performance metrics were gathered for the application environment (it_ops_network-* and it_ops_sql-*). Use multi-metric jobs to see what might be going on in there.
- Lastly, there’s an index (it_ops_app_logs-*) that has the application’s log lines in them. See if there’s anything anomalous in there as well.
- View all jobs correlated together in the Anomaly Explorer

Lab 4, Track 2: Analyze Web Logs

elastic

Lab 4: Track 2 - Problem

- There’s been suspicion that automated bots are slamming the web site from time to time.
- Use a population analysis job to find any rogue IPs and see what URIs were being requested
- Data is in gallery-* index

Lab 4, Track 3: Analyze NYC Taxis

elastic

Lab 4: Track 3 - Problem

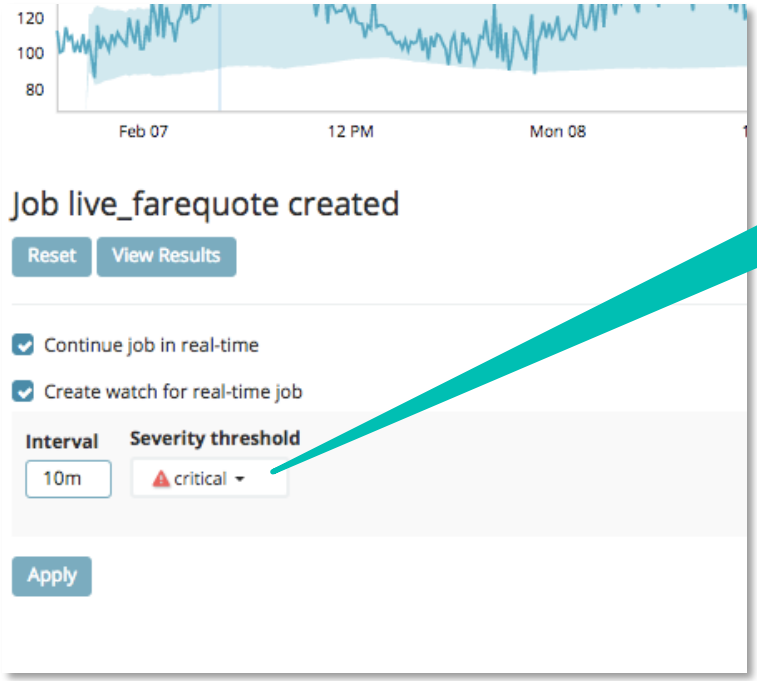
- Taxi ridership for NYC Yellow cabs is in the nyc-taxi-yellow-* index
- Determine time periods in which an unusual number of people take cab rides
- If those times are to be expected, use the Calendar to exclude them

Appendix

Alerting

Alerting from the ML User Interface

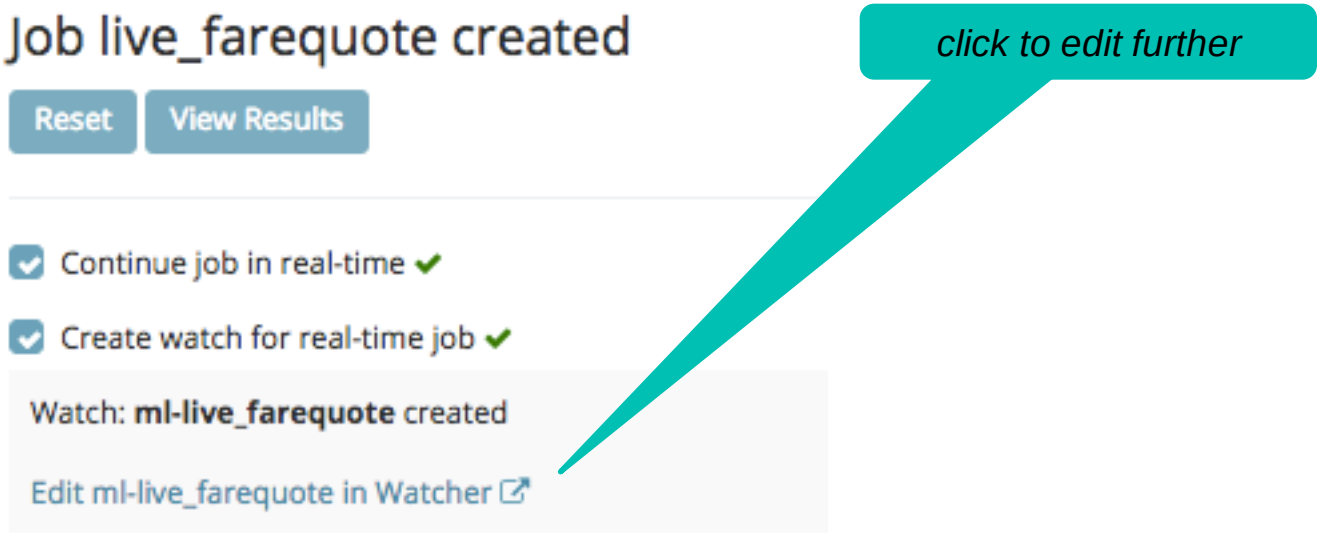
- After job configuration, or starting a datafeed, see option for creating a “watch” on the live data:



elastic

Alerting from the ML User Interface

- Clicking “Apply”:

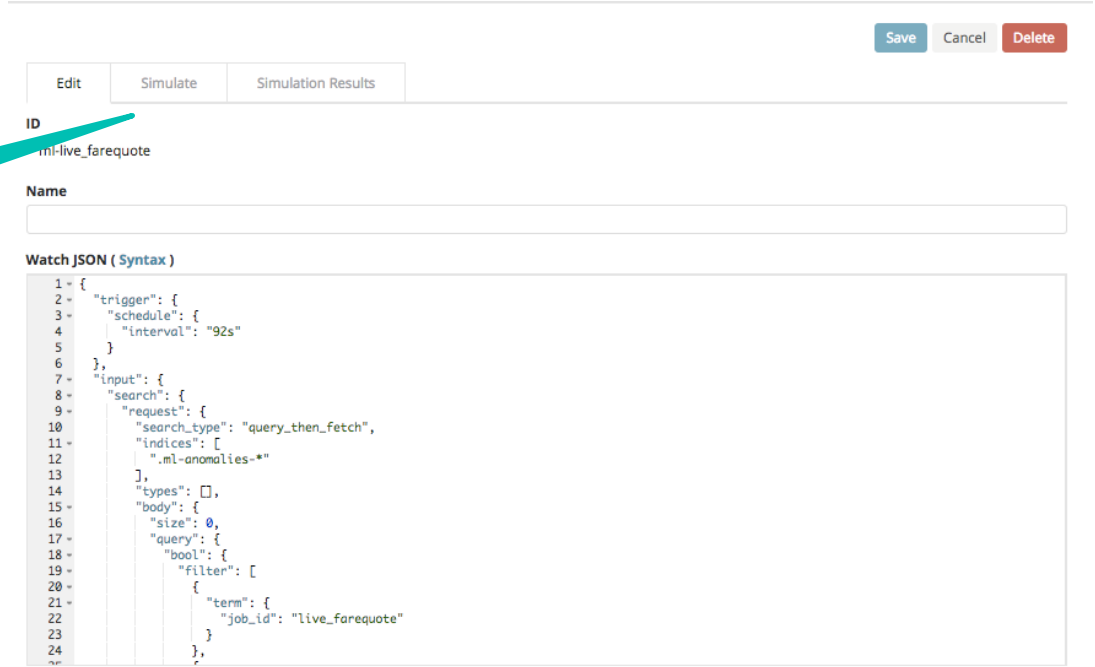


elastic

Alerting from the ML User Interface

- Now can edit Watch JSON to further customize things like action, etc.

or to test/simulate!



elastic

Under the Hood: Connecting ML to Alerting

- For all jobs - all anomalies are stored in **.ml-anomalies-*** indicies
- Every job also creates an index alias called **.ml-anomalies-<jobname>**
- You can configure Watches to query this index in 3 different ways:
  - "bucket" level
    - Answers: *How unusual was the job in a particular bucket of time?*
    - Essentially an aggregated anomaly score – useful for rate-limited alerts
  - "record" level
    - Answers: *What individual anomalies are present in a range of time?*
    - All the detailed anomaly information, but records can be numerous in big data
  - "influencers"
    - Answers: *What are the most unusual entities in a range of time?*
- *See https://github.com/elastic/examples/tree/master/Alerting/ml_examples*

elastic

Example – Querying for bucket results

- If we execute the query:

```
#farequote results bucket search
GET .ml-anomalies-*/_search
{
    "query": {
                "bool": {
                 "filter": [
                        { "range" : { "timestamp" : { "gte": "now-2y" } } },
                        { "term" :  { "result_type" : "bucket" } },
                        { "term" :  { "job_id" : "farequote_response" } },
                        { "range" : {"anomaly_score" : {"gte" : "75"}}}

                 ]
                }
    }
}
```

Example – An anomaly bucket

- We get the following:
  - anomaly_score: anomaly score of a bucket at time = timestamp
  - at this point, can also query “records” or “influencers” at this time for more detail
    - querying for records or influencers right away could lead to a lot of results for jobs with many detectors or entities in by/partition/over fields


```
{
  "took": 48,
  "timed_out": false,
  "_shards": {
    "total": 25,
    "successful": 25,
    "failed": 0
  },
  "hits": {
    "total": 1,
    "max_score": 0,
    "hits": [
      {
        "_index": ".ml-anomalies-shared",
        "_type": "result",
        "_id": "farequote_response_1455034500000_300",
        "_score": 0,
        "_source": {
          "job_id": "farequote_response",
          "timestamp": 1455034500000,
          "anomaly_score": 90.7,
          "bucket_span": 300,
          "initial_anomaly_score": 85.08,
          "record_count": 2,
          "event_count": 179,
          "is_interim": false,
          "bucket_influencers": [
            {
              "job_id": "farequote_response",
              "result_type": "bucket_influencer",
              "influencer_field_name": "airline",
              "initial_anomaly_score": 85.08,
              "anomaly_score": 90.7,
              "raw_anomaly_score": 43.1559,
              "probability": 2.661e-44,
              "timestamp": 1455034500000,
              "bucket_span": 300,
              "sequence_num": 5,
              "is_interim": false
            },
            {
              "job_id": "farequote_response",
              "result_type": "bucket_influencer",
              "influencer_field_name": "bucket_time",
```

A Simple Watch

- Trigger at a rate equal to or faster than bucket_span
- Define the input section as “search”
  - obviously make the “range” more “real-time” by using “now-1m”. It is longer in example because of date range of demo data.
- For the “condition” section, check to see if you get any payload hits
- For actions, can obviously do things more sophisticated than log (such as email, etc.)

```
# basic bucket watch
POST _xpack/watcher/watch/_execute
{
    "watch": {
    "trigger" : {
      "schedule" : { "interval" : "5m" }
    },
    "input" : {
      "search" : {
        "request" : {
          "indices" : [ ".ml-anomalies-*" ],
          "body" : {
            "query": {
              "bool": {
                "filter": [
                    { "range" : { "timestamp" : { "gte": "now-2y" } } },
                    { "term" :  { "result_type" : "bucket" } },
                    { "term" :  { "job_id" : "farequote_response" } },
                    { "range" : {"anomaly_score" : {"gte" : "75"}}}

                ]
              }
            }
          }
        }
      }
    },
    "condition" : {
      "compare" : { "ctx.payload.hits.total" : { "gt" : 0 }}
    },
    "actions" : {
      "log" : {
        "logging" : {
          "text" : "Anomalies:\n{{#ctx.payload.hits.hits}}score={{_source.anomaly_score}} at time={{_source.timestamp}}\n{{/ctx.payload.hits.hits}}"
        }
      }
  }
    }
  }
```

Watch Output

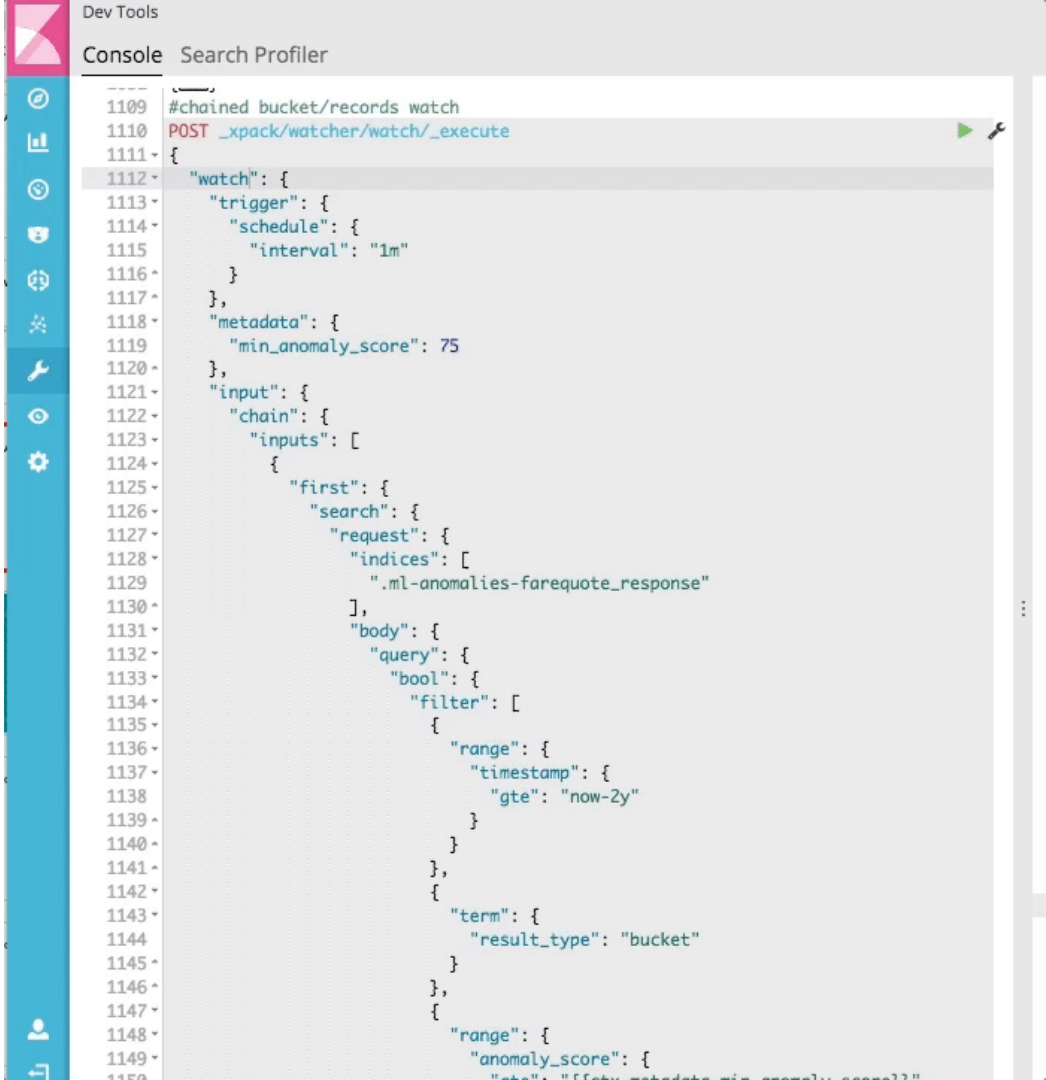
- The following is the resolved output of the watch, as seen in elasticsearch.log:

[2017-06-06T09:37:56,604][INFO][o.e.x.w.a.l.ExecutableLoggingAction] [GJgDody] Anomalies:score=90.7 at time=1455034500000

A Chained Watch

- Same idea, but use “chained” inputs to make 2 queries:
  - First to “bucket” level to test aggregated anomaly score
  - Second to “records” for that bucket time to return the records, if the bucket score was worthy
  - Output:

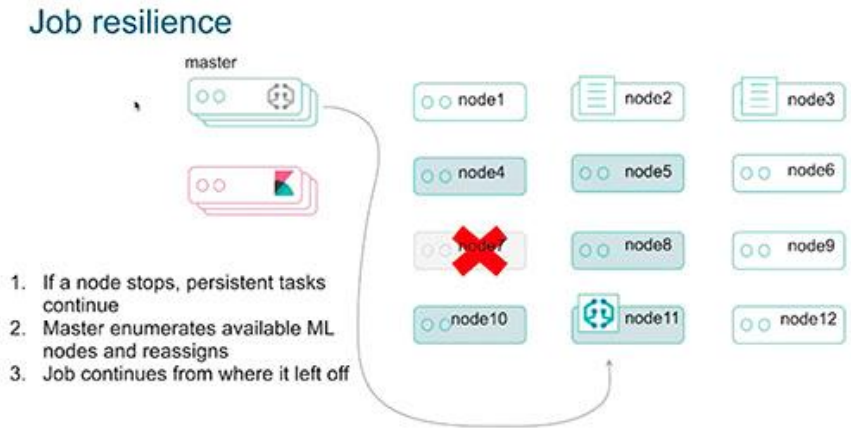
Anomaly of score=90.7 at 2016-02-09 11:15:00 Influenced by:
airline=AAL: score=95.4659, responsetime=296.19ms (typical=99.2962ms)
airline=AWE: score=0.00871093, responsetime=19.1644ms (typical=19.9918ms)



Architecture

Architecture

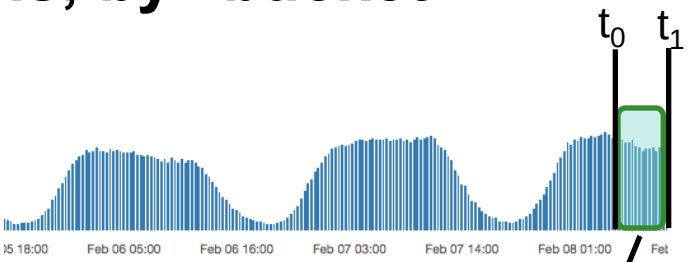
- ML jobs run on any ML-eligible node (node.ml:true)
- Runs ***outside*** of JVM (because it is C++)
- Best practice: Dedicated ML Nodes with ample CPU/RAM
- Jobs are tied to a specific node, but can failover to other ML-eligible nodes
- Job state (model) is stored in .ml-state index on data nodes of cluster
- Job results (anomalies) stored in .ml-anomalies index on data nodes of cluster

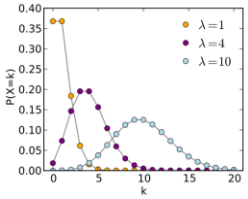


elastic

Analysis, by “bucket”

$t_n – t_{n-1}$ is called the “bucket_span”





2. get new data

1. read model

analysis

.ml-state

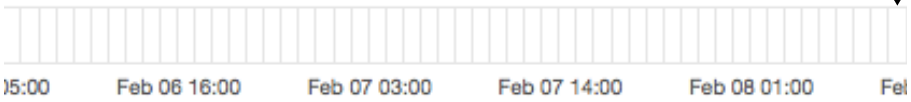
4. update model

3. analyze vs. model

5. store results

.ml-anomalies

results bucket t_0



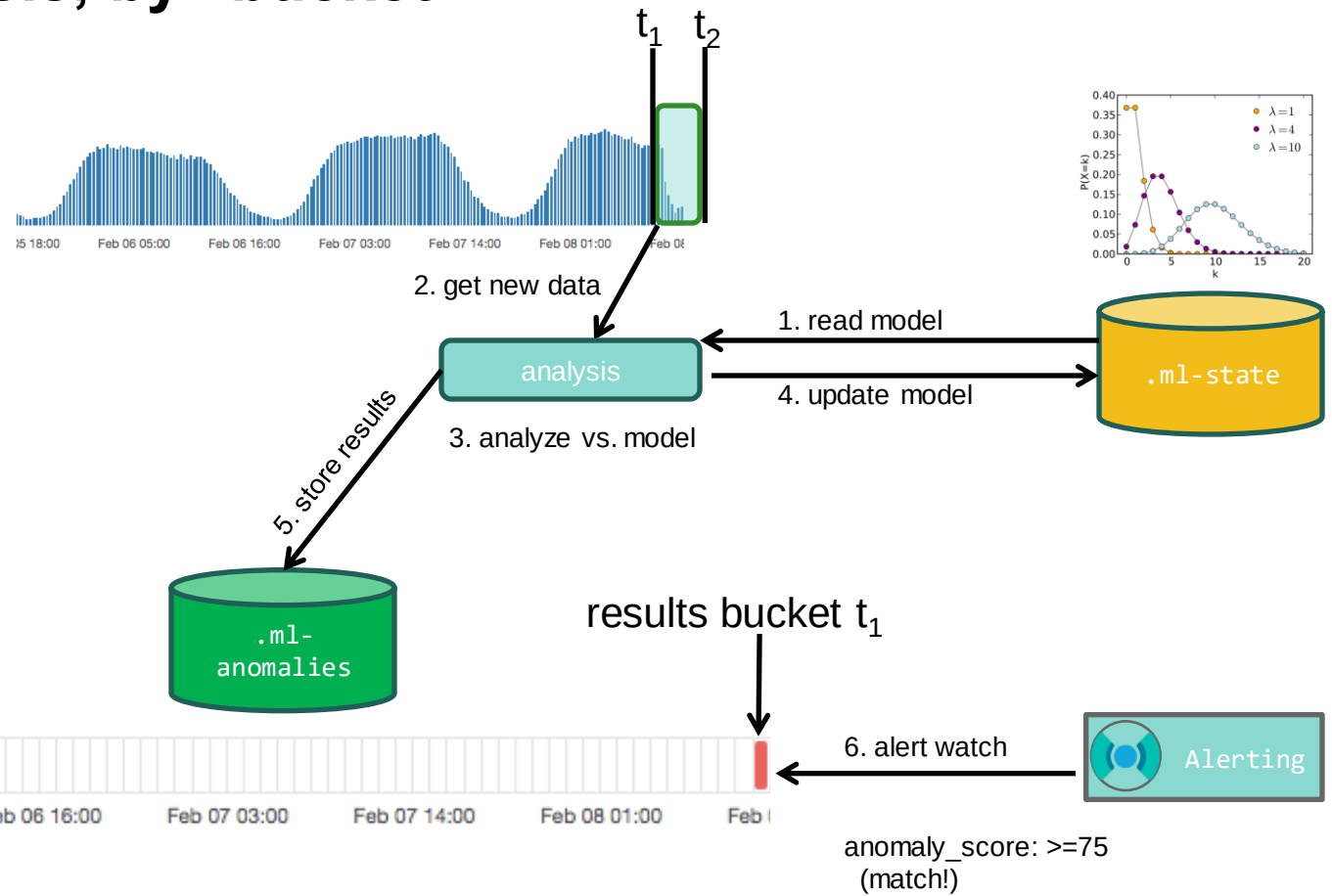
6. alert watch

Alerting

anomaly_score: >=75
(no match)

elastic

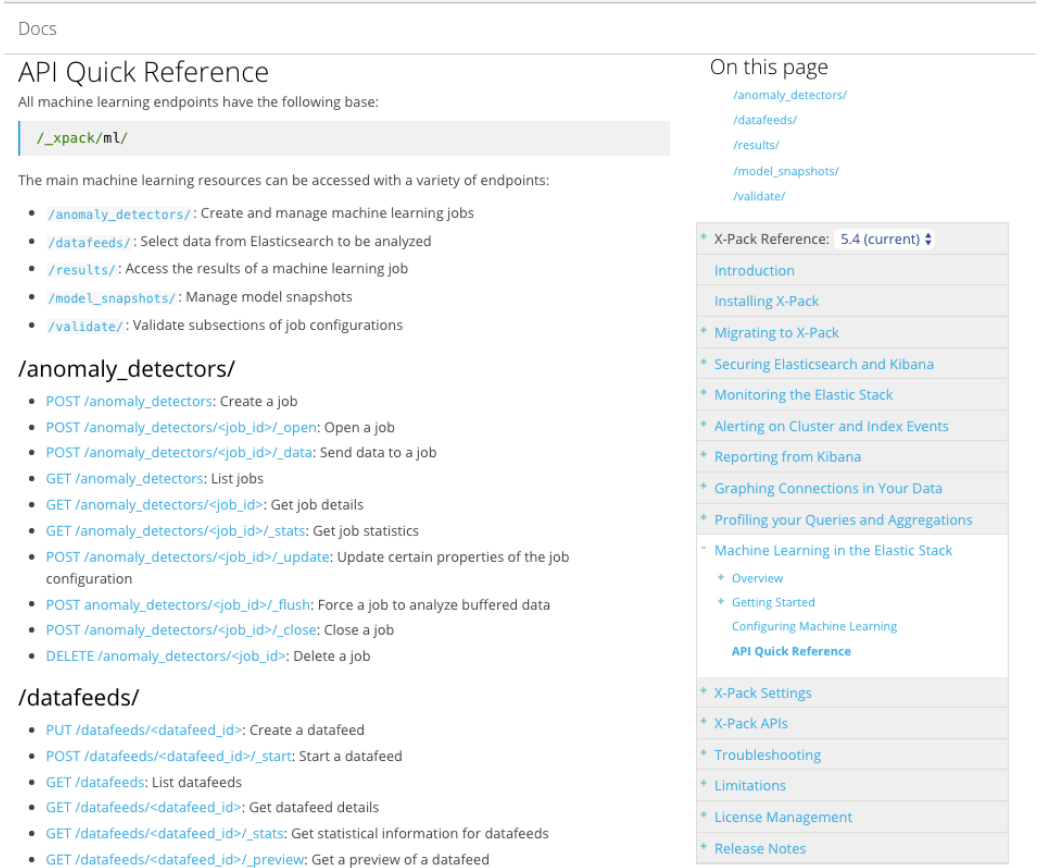
Analysis, by “bucket”



API Control

Controlling ML via API

- Full documentation of API available at https://www.elastic.co/guide/en/x-pack/current/ml-api-quickref.html



Docs

API Quick Reference

All machine learning endpoints have the following base:

```
/_xpack/ml/
```

The main machine learning resources can be accessed with a variety of endpoints:

- `/anomaly_detectors/`: Create and manage machine learning jobs
- `/datafeeds/`: Select data from Elasticsearch to be analyzed
- `/results/`: Access the results of a machine learning job
- `/model_snapshots/`: Manage model snapshots
- `/validate/`: Validate subsections of job configurations

/anomaly_detectors/

- POST /anomaly_detectors: Create a job
- POST /anomaly_detectors/<job_id>/_open: Open a job
- POST /anomaly_detectors/<job_id>/_data: Send data to a job
- GET /anomaly_detectors: List jobs
- GET /anomaly_detectors/<job_id>: Get job details
- GET /anomaly_detectors/<job_id>/_stats: Get job statistics
- POST /anomaly_detectors/<job_id>/_update: Update certain properties of the job configuration
- POST anomaly_detectors/<job_id>/_flush: Force a job to analyze buffered data
- POST /anomaly_detectors/<job_id>/_close: Close a job
- DELETE /anomaly_detectors/<job_id>: Delete a job

/datafeeds/

- PUT /datafeeds/<datafeed_id>: Create a datafeed
- POST /datafeeds/<datafeed_id>/_start: Start a datafeed
- GET /datafeeds: List datafeeds
- GET /datafeeds/<datafeed_id>: Get datafeed details
- GET /datafeeds/<datafeed_id>/_stats: Get statistical information for datafeeds
- GET /datafeeds/<datafeed_id>/_preview: Get a preview of a datafeed

On this page

/anomaly_detectors/
/datafeeds/
/results/
/model_snapshots/
/validate/

X-Pack Reference: 5.4 (current)

Introduction
Installing X-Pack
Migrating to X-Pack
Securing Elasticsearch and Kibana
Monitoring the Elastic Stack
Alerting on Cluster and Index Events
Reporting from Kibana
Graphing Connections in Your Data
Profiling your Queries and Aggregations
Machine Learning in the Elastic Stack
Overview
Getting Started
Configuring Machine Learning
API Quick Reference
X-Pack Settings
X-Pack APIs
Troubleshooting
Limitations
License Management
Release Notes

Example API Control

- All major operations are available via API
  - Create/Delete jobs and datafeeds
  - Job control (start/stop)
- Plus actions that are ONLY available via API
  - Model snapshot/restore

```
printf "\n\n== Creating job... \n"
curl -u elastic:changeme -s -X PUT -H 'Content-Type: application/json' ${JOBS}/${JOB_ID}?pretty -d '{
    "description" : "Unusual responsetimes by airlines",
    "analysis_config" : {
        "bucket_span": "5m",
        "detectors" :[{"function":"max", "field_name":"responsetime","by_field_name":"airline"}],
        "influencers" : [ "airline" ]
    },
    "data_description" : {
      "time_field":"@timestamp"
    }
}'

printf "\n\n== Creating datafeed... \n"
curl -u elastic:changeme -s -X PUT -H 'Content-Type: application/json' ${DATAFEEDS}/datafeed-${JOB_ID}?pretty -d '{
      "job_id" : "'"$JOB_ID"'",
      "indexes" : [
        "farequote"
      ],
      "types" : [
        "responsetime"
      ],
      "scroll_size" : 1000
}'


printf "\n\n== Opening job for ${JOB_ID}... "
curl -u elastic:changeme -X POST ${JOBS}/${JOB_ID}/_open

printf "\n\n== Starting datafeed-${JOB_ID}... "
curl -u elastic:changeme -X POST "${DATAFEEDS}/datafeed-${JOB_ID}/_start?start=1970-01-02T10:00:00Z&end=2017-01-01T00:00:00Z"

printf "\n\n== Finished ==\n\n"
```

elastic

The End

Extra Slides

bucket_span

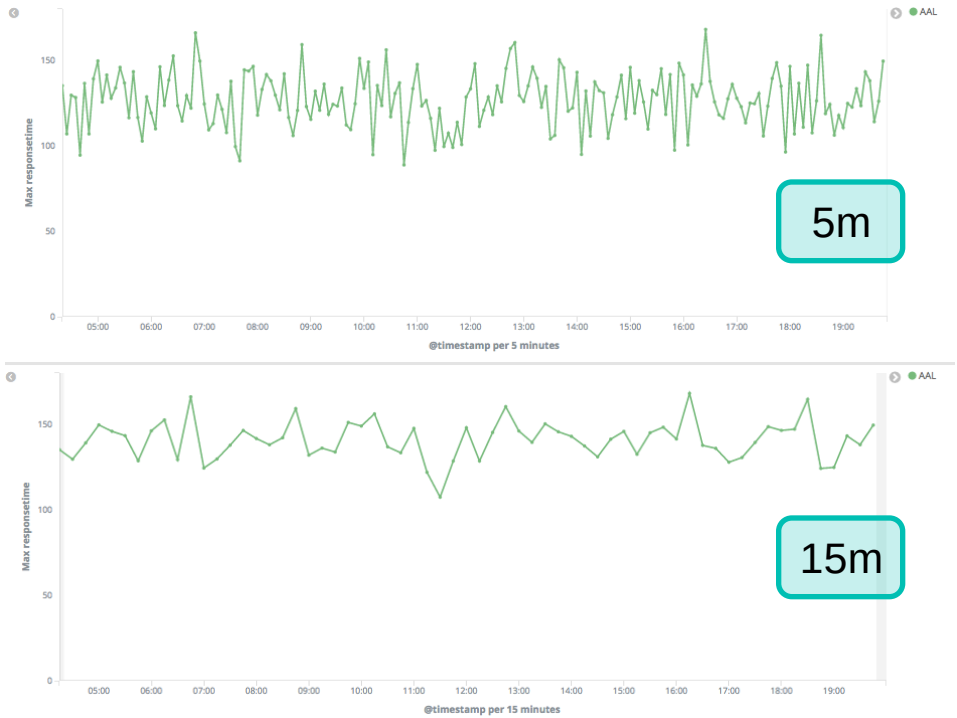
- Similar to a “Date Histogram” aggregation for an x-axis visualization in Kibana
- Defines the resolution (in time) that the data should be aggregated to

Date Histogram

elastic

What will bucket_span control?

- Granularity of data being analyzed
- Ultimately defines the volume of data analyzed per bucket
- How often the data is queried, when running in “real-time”



elastic

bucket_span: Rules of Thumb

- If you have “sampled” data (i.e. monitoring data every X minutes) then you could make bucket_span=X
- If you have arbitrary-rate data (i.e. log events), then pick a bucket_span that balances
  - timeliness (time to analyze/alert)
  - enough time to count, sum, etc. those events up to make the number of samples per bucket meaningful, given the velocity of data
- In other words, pick a bucket_span that’s approximately equivalent to the duration of an anomaly that you would care about.

By vs. Partition

- Both methods split data to establish separate baselines.
- Can be applied together in one detector.

- **By**
  - Creates attributes in existing model
  - Easier to scale
  - Best used to find anomalies in system as a whole
  - Scoring considers history of other by-fields
  - Required if using “rare” function

- **Partition**
  - Establishes different models
  - Requires more memory to store models
  - Best used to find anomalies in individual behavior
  - Scoring is more independent

Complex Detector Example

- high_count by ErrorCode over Host partition_field=App
  - Partition: App
    - Split the data into separate data sets for each App. Create a model (and population) for each.
  - Over: Host
    - For each App, create a population model out of all the Hosts and detect when a Host stands out.
  - By: ErrorCode
    - When modeling and evaluating Hosts, baseline each ErrorCode separately.
  - Function: High_Count
    - Measure the number of times an ErrorCode is observed in the time bucket.