## Step 1 - Watcher APIs

# 1.1 - Inspect Watcher stats

GET \_xpack/watcher/stats

## Step 2 - Excessive shard count logging

# 2.1 - Inspect cluster stats

GET \_cluster/stats

# 2.2 - Create shard count Watch

# Version 1 - only schedule and input

PUT \_xpack/watcher/watch/cluster\_shard\_count

{

"metadata": {

"max\_shards": 20

},

"trigger": {

"schedule": { "interval": "1m" }

},

"input": {

"http" : {

"request" : {

"host" : "localhost",

"port" : 9200,

"scheme": "https",

"auth": {

"basic": {

"username": "elastic",

"password": "changeme"

}

},

"path" : "/\_cluster/stats"

}

}

}

}

# 2.3 Check results of watch

GET .watcher-history-\*/\_search

{

"query": {

"match\_all": {}

},

"sort": { "trigger\_event.triggered\_time": "desc"}

}

# 2.4 Version 2 - Add condition and action

PUT \_xpack/watcher/watch/cluster\_shard\_count

{

"metadata": {

"max\_shards": 30

},

"trigger": {

"schedule": { "interval": "1m" }

},

"input": {

"http" : {

"request" : {

"host" : "localhost",

"port" : 9200,

"scheme": "http",

"auth": {

"basic": {

"username": "elastic",

"password": "changeme"

}

},

"path" : "/\_cluster/stats"

}

}

},

"condition" : {

"compare" : { "ctx.payload.indices.shards.total" : { "gt" : "{{ctx.metadata.max\_shards}}" } }

},

"actions" : {

"log\_warn" : {

"logging" : {

"text" : "The current number of shards in the cluster ({{ctx.payload.indices.shards.total}}) exceeds the set threshold ({{ctx.metadata.max\_shards}}).",

"level": "warn"

}

}

}

}

# 2.5 Add index with shards

PUT test

{

"settings" : {

"index" : {

"number\_of\_shards" : 20,

"number\_of\_replicas" : 0

}

}

}

# 2.6 Delete created index

DELETE test

# 2.7 Version 3 - Throttle logging messages to max one every 10 minutes

PUT \_xpack/watcher/watch/cluster\_shard\_count

{

"metadata": {

"max\_shards": 30

},

"trigger": {

"schedule": { "interval": "1m" }

},

"input": {

"http" : {

"request" : {

"host" : "localhost",

"port" : 9200,

"scheme": "http",

"auth": {

"basic": {

"username": "elastic",

"password": "changeme"

}

},

"path" : "/\_cluster/stats"

}

}

},

"condition" : {

"compare" : { "ctx.payload.indices.shards.total" : { "gt" : "{{ctx.metadata.max\_shards}}" } }

},

"actions" : {

"log\_warn" : {

"throttle\_period": "5m",

"logging" : {

"text" : "The current number of shards in the cluster ({{ctx.payload.indices.shards.total}}) exceeds the set threshold ({{ctx.metadata.max\_shards}}).",

"level": "warn"

}

}

}

}

# 2.8 Deactivate watch

PUT \_xpack/watcher/watch/cluster\_shard\_count/\_deactivate

## Step 3 - Alert on process existance

# 3.1 Verify Metricbeat is logging data successfully

GET metricbeat-\*/\_search

{

"query": {

"match\_all": {}

}

}

# 3.2 - Inspect Metricbeats data

GET metricbeat-\*/\_search

{

"query" : {

"term": {

"system.process.name": {"value": "Google Chrome"}

}

}

}

# 3.3 Version 1 - Record search

PUT /\_xpack/watcher/watch/my\_process\_watch

{

"trigger" : {

"schedule" : {"interval" : "60s"}

},

"input" : {

"search" : {

"request" : {

"indices" : ["metricbeat\*"],

"body" : {

"query" : {

"bool" : {

"must" : {

"term": {

"system.process.name": {"value": "Google Chrome"}

}

},

"filter": {

"range": {

"@timestamp": { "gt" : "now-2m"}

}

}

}

}

}

}

}

},

"condition" : {

"compare" : {"ctx.payload.hits.total" : { "gt" : 0}}

}

}

# 3.4 Add index for keeping results

PUT my\_alert\_index

{

"mappings" : {

"alert" : {

"properties" : {

"alert\_name" : {"type" : "text"},

"alert\_text" : {"type" : "keyword"},

"alert\_time" : {"type" : "date"}

}

}

}

}

# 3.5 Version 2 - Write results to separate index

PUT /\_xpack/watcher/watch/my\_process\_watch

{

"trigger" : {

"schedule" : {"interval" : "60s"}

},

"input" : {

"search" : {

"request" : {

"indices" : ["metricbeat\*"],

"body" : {

"query" : {

"bool" : {

"must" : {

"term": {

"system.process.name": {"value": "Google Chrome"}

}

},

"filter": {

"range": {

"@timestamp": { "gt" : "now-2m"}

}

}

}

}

}

}

}

},

"condition" : {

"compare" : {"ctx.payload.hits.total" : { "gt" : 0}}

},

"actions": {

"index\_payload" : {

"transform": {

"script": "return ['alert\_name': ctx.watch\_id , 'alert\_text': \"Alerted and action taken\" , 'alert\_time': ctx.trigger.triggered\_time] "

},

"index" : {

"index" : "my\_alert\_index",

"doc\_type" : "alert"

}

}

}

}

# 3.6 Look up indexed results

GET /my\_alert\_index/\_search

{

"query" : {

"term": {

"alert\_name": {"value": "my\_process\_watch"}

}

}

}

# 3.8 Deactivate watch

PUT \_xpack/watcher/watch/my\_process\_watch/\_deactivate

# 4.1 Missing things

PUT \_xpack/watcher/watch/flights\_watch/

{

"trigger": {

"schedule": {

"interval": "1h"

}

},

"input": {

"search": {

"request": {

"search\_type": "query\_then\_fetch",

"indices": [

"flights-\*"

],

"types": [],

"body": {

"query": {

"range": {

"{{ctx.metadata.timefield}}": {

"gte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.window\_period}}",

"lte": "now-{{ctx.metadata.offset}}"

}

}

},

"aggs": {

"periods": {

"filters": {

"filters": {

"history": {

"range": {

"{{ctx.metadata.timefield}}": {

"gte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.window\_period}}",

"lte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.last\_period}}"

}

}

},

"last\_period": {

"range": {

"{{ctx.metadata.timefield}}": {

"gte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.last\_period}}",

"lte": "now-{{ctx.metadata.offset}}"

}

}

}

}

},

"aggs": {

"things": {

"terms": {

"field": "{{ctx.metadata.thing}}",

"size": 10000

}

}

}

}

},

"size": 0

}

}

}

},

"condition": {

"script": {

"inline": "return ctx.payload.aggregations.periods.buckets.history.things.buckets.size() > ctx.payload.aggregations.periods.buckets.last\_period.things.buckets.size();",

"lang": "painless"

}

},

"actions": {

"log": {

"transform": {

"script": {

"inline": "def last\_period=ctx.payload.aggregations.periods.buckets.last\_period.things.buckets.stream().map(e -> e.key).collect(Collectors.toList()); return ctx.payload.aggregations.periods.buckets.history.things.buckets.stream().map(e -> e.key).filter(p -> !last\_period.contains(p)).collect(Collectors.toList());",

"lang": "painless"

}

},

"logging": {

"level": "info",

"text": "{{ctx.metadata.thing}} missing in the last {{ctx.metadata.last\_period}} that were present in the prior day: {{#ctx.payload.\_value}}{{.}} {{/ctx.payload.\_value}}"

}

}

},

"metadata": {

"timefield": "@timestamp",

"last\_period": "24h",

"offset": "2y",

"window\_period": "48h",

"thing": "DEST"

},

"throttle\_period\_in\_millis": 300000

}