Practical Use-Cases of Solr's AutoScaling Framework

Varun Thacker
Lucidworks Inc.

Lucidworks

Overview

AutoScaling Policies and Preferences

AutoScaling API

Triggers and Listeners

Autoscale a Solr Cluster to a trillion documents with minimal human intervention

Why Auto-Scaling ?

- Operations is hard to scale
- It’s time consuming
- Tough to build expertise
- Tooling is required

In a nutshell

Lucidworks

- Provide my cluster assumptions and constraints
- Define constraints in terms of Disk / CPU / Search latency / Update throughput

Auto-Scaling helps you with :

- Which operations can take us to the desired state?
- When to perform those operations?

Describing your cluster

Define the layout and characteristics of the cluster e.g.

- All replicas must be on unique nodes
- All replicas of each shard must be on the same rack
- At least 2 replicas of each shard must be on the same rack
- No replicas should be added to a node that already has 4 Solr cores

Describing your cluster

Lucidworks

We must also define the load on each node

- System load average
- Free disk space
- Heap Usage
- Number of existing Solr cores

Auto-Scaling Policies

Lucidworks

- Policy defines the desired layout of the cluster
- Policies are at the cluster level and augmented for collections if necessary
- Replaces the old “Rule based replica placement” framework

Auto-Scaling Policies

Specify policies for a node

- Any node *must* have a maximum of 5 solr cores

```
{node:#ANY, cores:<6}
```

Auto-Scaling Policies

Specify a policy that applies to collections

- All replicas must be on unique nodes

```
{node:#ANY, collection:my_coll, replica:<2}
```

- Do not place more than 1 replica of a shard on the same node

```
{node:#ANY, collection:my_coll, shard:#EACH, replica:<2}
```

Auto-Scaling Policies

Lucidworks

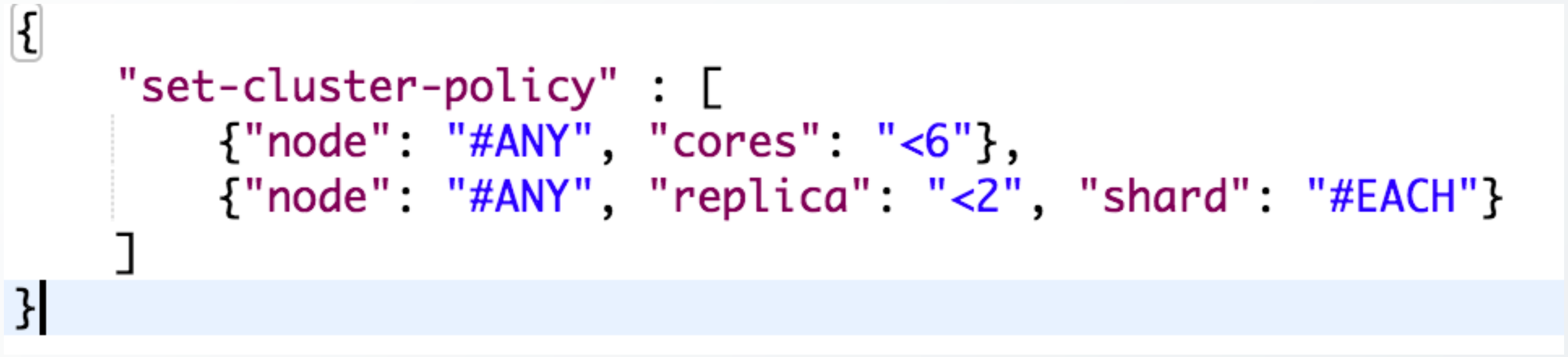
- Spread two replicas for each shard across availability zones
- Start every Solr node with a system property to identify it’s availability zone
  - Example “-Davailability_zone=us-east-1a”

```
    {"replica": "<2", "shard": "#EACH",
"sysprop.availability_zone": “us-east-1a"}

    {"replica": "<2", "shard": "#EACH",
"sysprop.availability_zone": "us-east-1b"}
```

Auto-Scaling Policy - API

```
curl -X POST http://host:port/api/cluster/autoscaling -d
```



```
{
    "set-cluster-policy" : [
        {"node": "#ANY", "cores": "<6"},
        {"node": "#ANY", "replica": "<2", "shard": "#EACH"}
    ]
}
```

Auto-Scaling Preferences

- Preferences is a language to define load
- Preferences are only at the cluster level
- Not hard conditions
- You can choose to either maximise or minimise on a metric to order nodes
  - cores - number of Solr cores on a node
  - freedisk - the amount of free disk space
  - heapUsage - heap used / heap allocated
  - sysLoadAvg - the system load average reported by JVM

Auto-Scaling Preferences

```
{minimize:cores}

{maximize:freedisk}

{minimize:sysLoadAvg}

{<sort-order>:<metric>, precision:<val>}
```

Auto-Scaling Preferences

Lucidworks

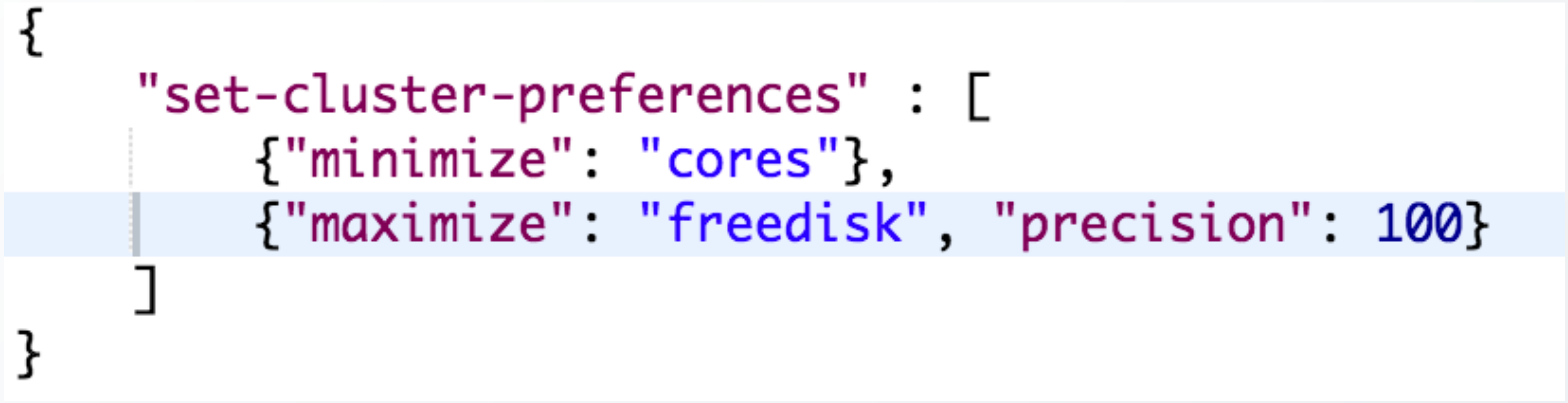
```
{maximize:freedisk, precision:10}
```

If the difference between the values of free disk for two nodes is within the precision value then they are considered equivalent.

The precision allows Solr to introduce ties so that the rest of the preferences can be applied. Metrics having floating point values must have precision if they are not the last preference.

Auto-Scaling Preferences - API

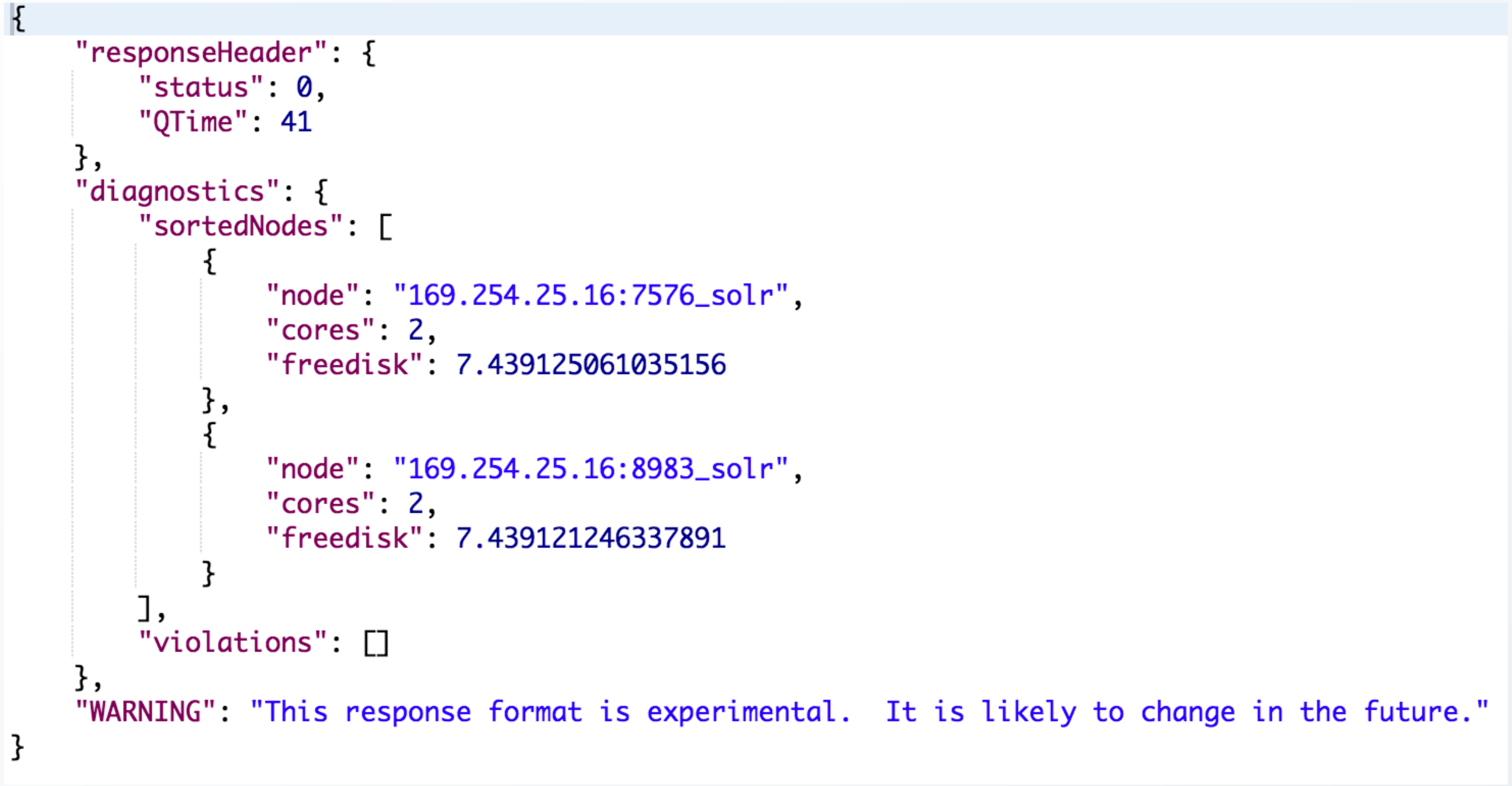
```
curl -X POST http://host:port/api/cluster/autoscaling -d
```



```
{
    "set-cluster-preferences" : [
        {"minimize": "cores"},
        {"maximize": "freedisk", "precision": 100}
    ]
}
```

Auto-Scaling Preferences - API

```
curl http://host:port/api/cluster/autoscaling/diagnostics
```



```
{
    "responseHeader": {
        "status": 0,
        "QTime": 41
    },
    "diagnostics": {
        "sortedNodes": [
            {
                "node": "169.254.25.16:7576_solr",
                "cores": 2,
                "freedisk": 7.439125061035156
            },
            {
                "node": "169.254.25.16:8983_solr",
                "cores": 2,
                "freedisk": 7.439121246337891
            }
        ],
        "violations": []
    },
    "WARNING": "This response format is experimental.  It is likely to change in the future."
}
```

Collection APIs

Lucidworks

Collection APIs automatically use policy and preferences to place replicas in desired nodes

- Create Collection
- Add Replica
- Split Shard
- Create Shard
- Restore Collection

Collection Replication Factor

Lucidworks

- Define Solr collections which maintain it’s replication factor
- When enabled, Solr auto-creates relevant triggers and adds replicas to keep the replication factor satisfied
- Uses the defined policies and preferences while adding the new replica
- Add **&autoAddReplicas=true** while creating your collection
- False by default

Auto-Scaling Triggers

Triggers, once activated, perform actions such as evaluating the system against the autoscaling configuration.

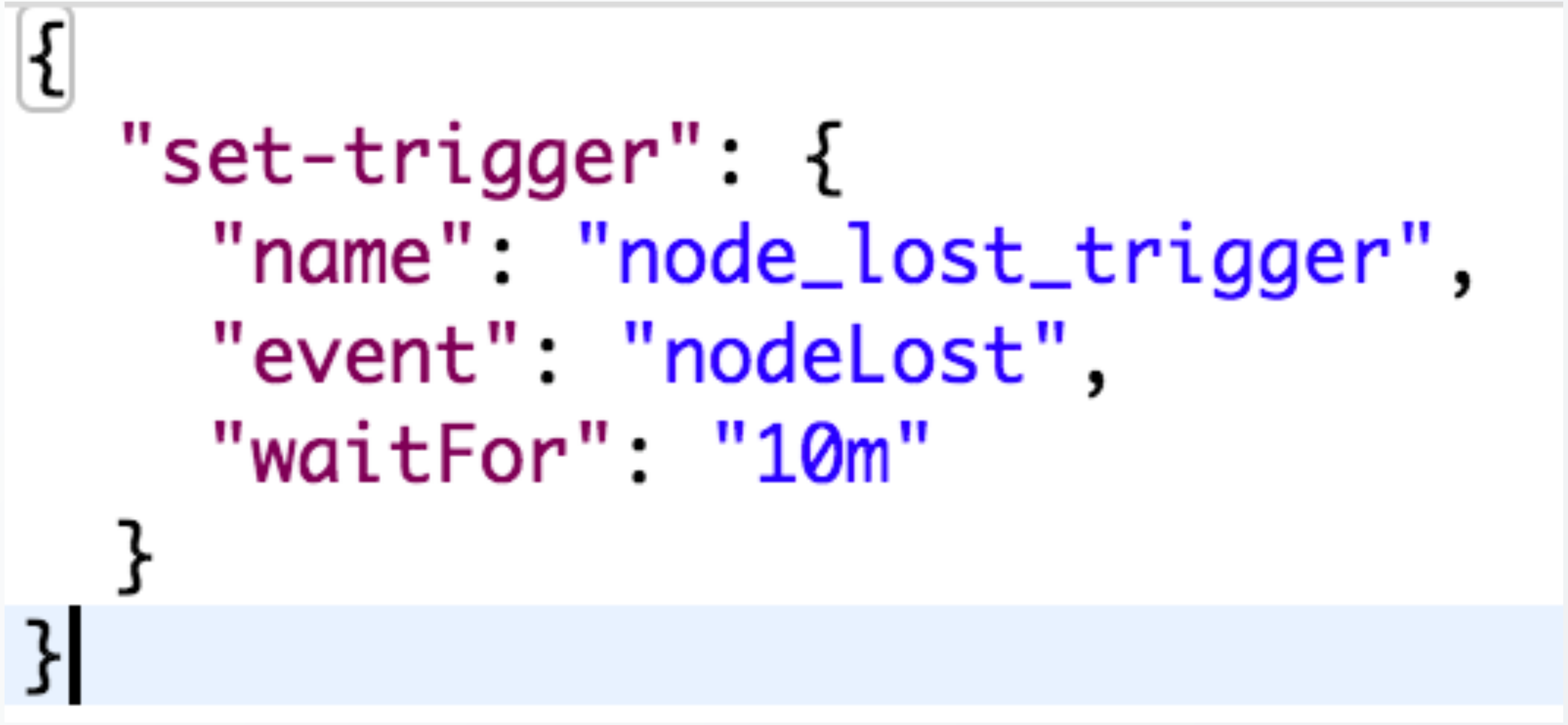
Solr 7.1 introduced two triggers that activate on a node joining or leaving a cluster. Both move replicas around to balance load.

Solr 7.3 introduced triggers that activate based on search rate / a schedule / metric based

Auto-Scaling Triggers - API

Lucidworks

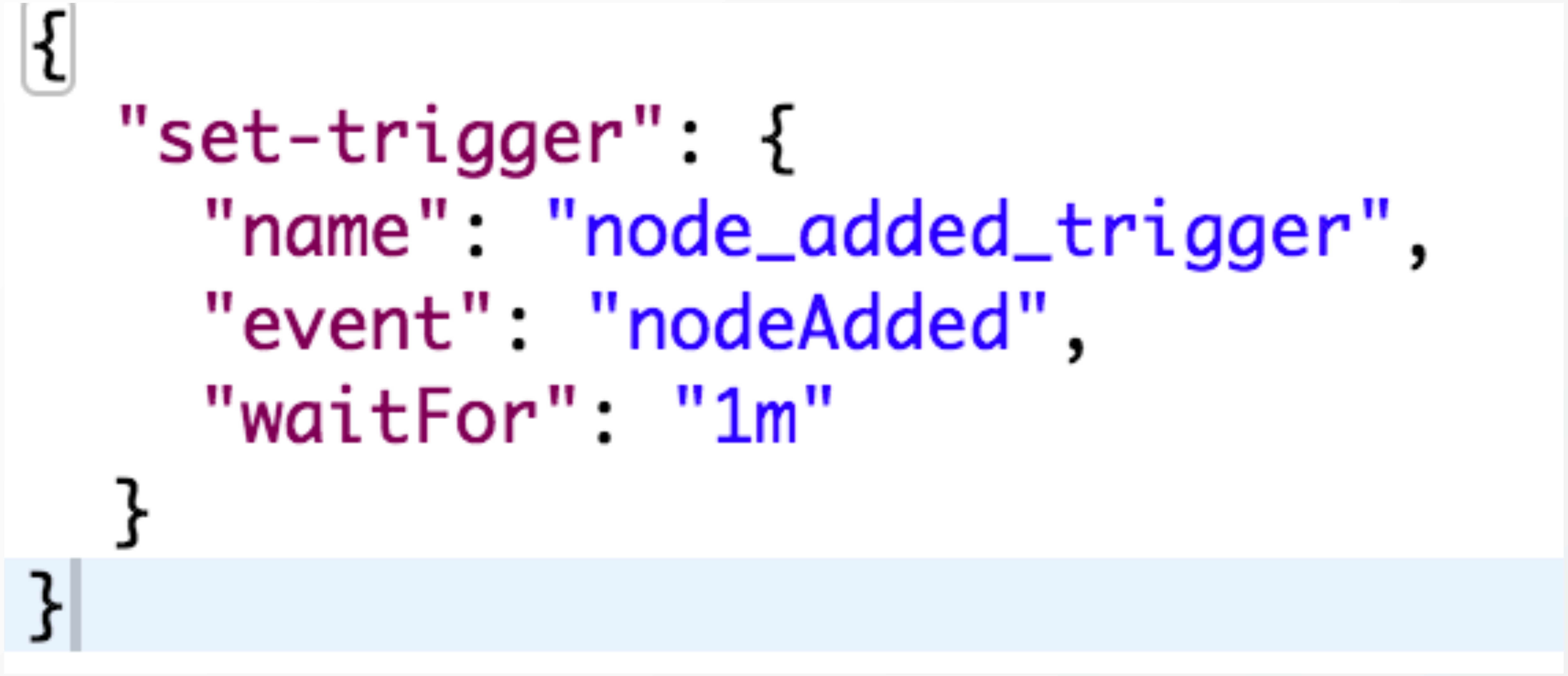
```
curl -X POST http://host:port/api/cluster/autoscaling -d
```



```
{
  "set-trigger": {
    "name": "node_lost_trigger",
    "event": "nodeLost",
    "waitFor": "10m"
  }
}
```

Auto-Scaling Triggers - API

```
curl -X POST http://host:port/api/cluster/autoscaling -d
```



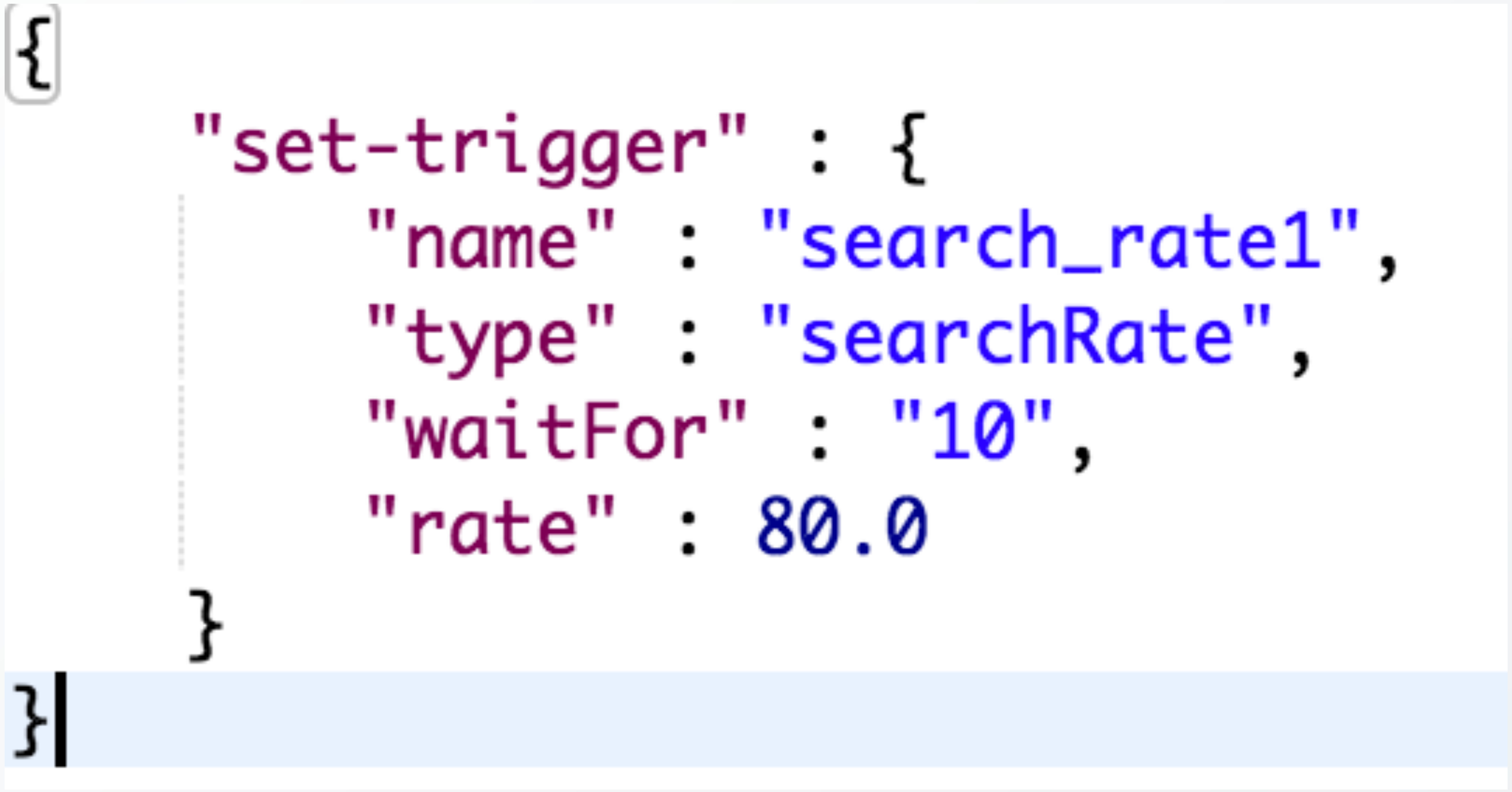
```
{
  "set-trigger": {
    "name": "node_added_trigger",
    "event": "nodeAdded",
    "waitFor": "1m"
  }
}
```

Search Rate Trigger

- Monitors 1-min average search rates
- Monitors rates per collection, per shard and per node
- Moves the replica with the highest rate to another node if rate is breached per-node
- Adds a replica if rate is breached per-collection or per-shard

Search Rate Trigger - API

```
curl -X POST http://host:port/api/cluster/autoscaling -d
```



```
{
    "set-trigger" : {
        "name" : "search_rate1",
        "type" : "searchRate",
        "waitFor" : "10",
        "rate" : 80.0
    }
}
```

Trigger Actions

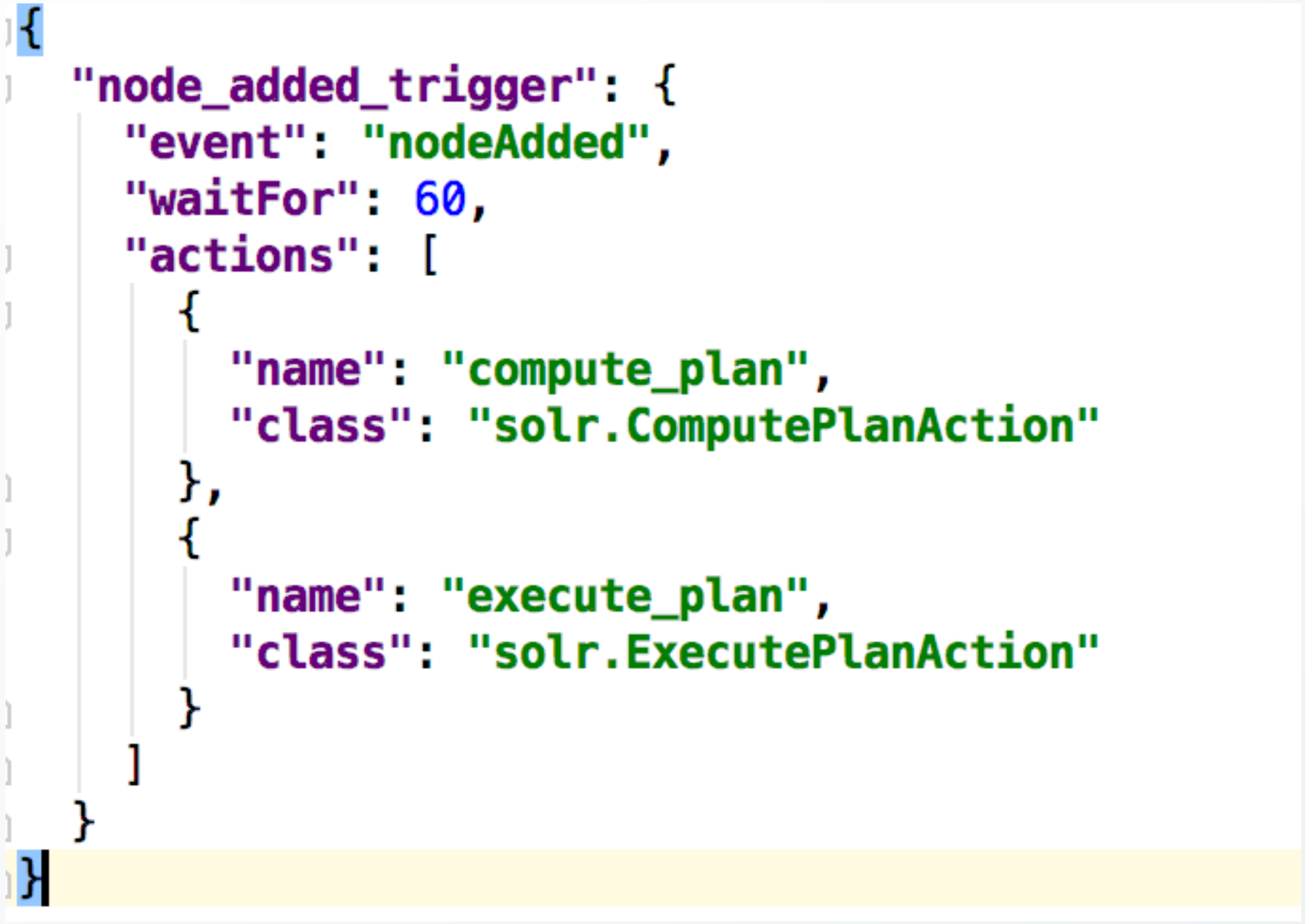
Lucidworks

Triggers Actions are executed in response to a trigger.

- Compute Plan:
  - Evaluates the policy/preferences against the current state of Solr
  - Computes the list of operations that push Solr towards the desired state
- Execute Plan: Executes the list of operations

Auto-Scaling Triggers - API

Lucidworks



```
{
  "node_added_trigger": {
    "event": "nodeAdded",
    "waitFor": 60,
    "actions": [
      {
        "name": "compute_plan",
        "class": "solr.ComputePlanAction"
      },
      {
        "name": "execute_plan",
        "class": "solr.ExecutePlanAction"
      }
    ]
  }
}
```

Trigger Listener

Trigger Listeners are attached to a trigger to be notified of important lifecycle events.

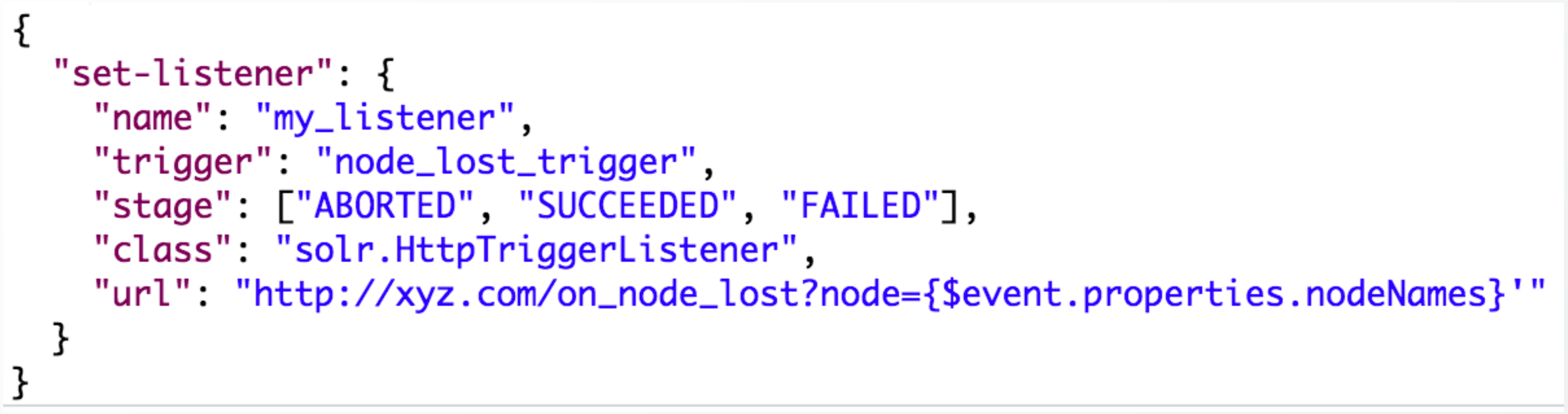
Examples of lifecycle include a trigger being activated, aborted and overall success or failure as well as start or finish of individual actions

Solr provides two listeners: one automatically stores event info to .system collection and second can be used to send events to external systems via HTTP

HTTP Trigger Listener - API

Lucidworks

```
curl -X POST http://host:port/api/cluster/autoscaling -d
```



```
{
  "set-listener": {
    "name": "my_listener",
    "trigger": "node_lost_trigger",
    "stage": ["ABORTED", "SUCCEEDED", "FAILED"],
    "class": "solr.HttpTriggerListener",
    "url": "http://xyz.com/on_node_lost?node={$event.properties.nodeNames}'"
  }
}
```

Lucidworks

Thank You

Lucidworks