Lighthouse Documentation

Release 0.1.0

the Lighthouse Authors, see the AUTHORS.md file

Dec 23, 2018

Contents

Star Lighthouse on Github: Fork Lighthouse on Github:

CHAPTER 1

Tour of Features

This article constitutes a tour of features of Lighthouse, starting with simplest use cases and moving on to more advanced features associated with more complicated problems.

Lighthouse currently defines a single HTTP JSON RPC endpoint named `/assign-workloads`.

1.1 Basic Scheduling

You can schedule workloads onto nodes like this:

```
curl \
    -X POST \
    --data-binary '

{
    "nodes": [
        {
            "id": "node-1",
            "resources": {
                "cpu": 1.0,
                "mem": 8.0
                }
        },
        {
            "id": "secondnode",
            "resources": {
                "cpu": 5.0,
                "mem": 4.0
            }
        }
    ],
    "workloads": [
        {
```

(continues on next page)

(continued from previous page)

```
        "id": "firstreq",
        "requirements": {
            "cpu": 5.0
            }
        }
    ]
}' \
<sphinx-endpoint>/assign-workloads

# =>
#{
#    "successful": true,
#    "assignments": {
#        "firstreq": "secondnode"
#    }
#}
```

As you can see, `assign-workloads` takes a list of nodes and workloads and attempts to assign workloads to nodes. If there is enough room, it returns with `successful` as `true` and gives a list of assignments using `json`.

Note that the requirements in a workload need not include all the types of resources found in nodes. In the above example, each node has `mem` and `cpu` attributes, but only a `cpu` attribute is required by the workload.

1.2 Failure to Schedule

If it fails, it will simply return a json blob with `successful` as `false` and an empty `assignments` dictionary:

```
curl \
    -X POST \
    --data-binary '
{
    "nodes": [
        {
            "id": "node-1",
            "resources": {
                "cpu": 1.0,
                "mem": 8.0
                }
        },
        {
            "id": "secondnode",
            "resources": {
                "cpu": 5.0,
                "mem": 4.0
            }
        }
    ],
    "workloads": [
        {
        "id": "firstreq",
        "requirements": {
            "cpu": 6.0
            }
        }
    ]
```

(continues on next page)

(continued from previous page)

```
}' \
<sphinx-endpoint>/assign-workloads

# =>
#
#{
#    "successful": false,
#    "assignments": {}
#}
```

In the above example, workload placement failed because the workload needed too much CPU. It needed `6.0` units of `cpu`, but any single available node given only had `5.0`.

Placement of workloads onto nodes is not guaranteed. That is, simply because room exists for all workloads, this does not mean that Lighthouse will be able to figure this out. You can help Lighthouse get better at packing nodes tightly using the *BinPack* strategy, and you can also increase the capacity of the nodes.

1.3 Placement Strategies

You can tell Lighthouse to use one of several placement strategies when placingj workloads onto nodes. `Prioritized` is the default, because it is the simplest, but it is not the best. `BinPack` is, in general, recommended.

The following example will be referred to when discussing each of the placement strategies below:

```
curl \
    -X POST \
    --data-binary '
{
  "nodes": [
    {
      "id": "node-1",
      "resources": {
        "cpu": 2,
        "mem": 8,
        "disk": 60
      }
    },
    {
      "id": "node-2",
      "resources": {
        "cpu": 6,
        "mem": 6,
        "disk": 20
      }
    },
    {
      "id": "node-3",
      "resources": {
        "cpu": 4,
        "mem": 2,
        "disk": 40
      }
    }
  ],
  "workloads": [
```

(continues on next page)

(continued from previous page)

```
    {
      "id": "req-1",
      "requirements": {
        "cpu": 1,
        "mem": 2,
        "disk": 10
      }
    },
    {
      "id": "req-2",
      "requirements": {
        "cpu": 3,
        "mem": 2,
        "disk": 5
      }
    },
    {
      "id": "req-3",
      "requirements": {
        "cpu": 2,
        "mem": 4,
        "disk": 50
      }
    }
  ],
  "strategy": "<strategy>"
}
' <lighthouse-endpoint>/assign-workloads

# =>
#{"successful":true,"assignments":{"req-1":"node-2","req-3":"node-1","req-2":"node-3"}
↪}
```

1.3.1 Prioritized

With a strategy of `Prioritized`, Lighthouse will attempt to assign workloads to nodes in the order they appear in the given list of nodes, and in the order the workloads appear.

This is the result if the above were run with `<strategy>` were run with `Prioritized`:

```
{"successful":true,"assignments":{"req-1":"node-1","req-3":"node-1","req-2":"node-1"}}
```

In this example, all nodes are assigned to `node-1` because they can all fit on `node-1` and it appears first in the list of nodes given.

1.3.2 RoundRobin

With a strategy of `RoundRobin`, assignment of workloads is done in the order given in the list, but placement attempts for each successive load starts on the node just after the successful placement of the previous load – in a "round robin" fashion.

This is the result if the above were run with `<strategy>` as `RoundRobin`:

```
{"successful":true,"assignments":{"req-1":"node-1","req-3":"node-3","req-2":"node-2"}}
```

1.3.3 BinPack

This strategy requires additional information in the JSON blob that is given to `/assign-workloads`. A `rubric` must be specified. In discussing the example above, we will assume in our discussion that the following was also sent to the RPC endpoint:

```
"strategy": "BinPack",
"rubric": {
        "cpu": 1,
        "mem": 0.5,
        "disk": 0.025
      }
...
```

BinPack attempts to pack in as many requirements into as few nodes as possible. In order to do so, the caller must specify a `rubric`. This specifies that certain attributes need to be present in all nodes as resources and all workloads as requirements, and gives quantities that will be used to score each workload and node by multiplying each quantity for a given node or workload and summing the results. This score is computed for each node and workload and semantically corresponds to the node or load's "size". If any node or workload doesn't have all the attributes in the rubric, the call to `/assign-workloads` will not be successful. In future versions of `/assign-workloads`, specifying negative values in the rubric will not be allowed and in the current version if this happens the result is undefined.

If `BinPack` was used in the above example, the result would look like this:

```
{"successful":true,"assignments":{"req-1":"node-2","req-3":"node-1","req-2":"node-3"}}
```

In this example, all workloads were assigned to `node-3`, since `node-3` had the least room in it going into scheduling, since it had the least disk space.

1.4 Placement Enforcement

At the time of placement of a workload onto a node, the requirements are subtracted from the node's resources so as to keep track of what nodes still have room left for more assignments. In particular, all attributes associated with the *node* must register with a quantity at or above zero in order for the assignment to succeed at *assignment time*.

This allows for some interesting possibilities for how to enforce where workloads can be assigned in your cluster of nodes.

1.4.1 Node Tagging

Sometimes it is desirable to mark a particular node as specifically dedicated to a particular type of workload. When this is desired, it is simply a matter of adding a resource to a node with zero as the quantity:

```
...
"nodes": [
    {
        "id: "node1",
        "resources": {
           "dedicated": 0.0,
           ...
        }
    }
]
```

Then, simply place a similar attribute in the requirements dictionary of the workloads that should be run on the dedicated nodes:

```
...
"workloads": [
    {
        "id": "workload1",
        "requirements": {
            "dedicated": 0.0,
            ...
        }
    }
]
```

This works because all requirements listed for a workload must be present on the node and none may be allowed to be below zero, but zero is okay.

1.4.2 Deficits and Tolerations

This concept is similar to Kubernetes' Taints and Tolerations idea, but also has nuances to it that make it more flexible.

The idea is to mark a particular set of nodes as unavailable for workloads unless those workloads specifically opt into being run on those nodes.

We do this in Lighthouse using Defecits and Tolerations.

It is perfectly fine to list negative values for resources at call time on a node; however, as has been previously explained, if there are any resources in a node with negative quantity at *assignment time* of a workload, the workload is not able to be attached.

Negative resources can be overcome by a resource in one of two ways.

First, for negative resource of *finite* this can be overcome by simply listing a negative requirement. That way, when one is subtracted from the other, the result will be zero:

```
...
"nodes": [
    {
        "id: "node1",
        "resources": {
           "flies": -5.0,
           ...
        }
    }
],
"workloads": [
    {
        "id": "workload1",
        "requirements": {
            "flies": -5.0,
            ...
        }
    }
]
```

This may be used to list "shortcomings" of a node that precludes it from having workloads scheduled on it unless at least one workload has a sufficient tolerance to the shortcoming.

Second, we list a node up front at call time with a resource that has infinite negative value:

```
...
"nodes": [
    {
        "id: "node1",
        "resources": {
           "spiders": -inf,
           ...
        }
    }
]
```

In this scenario, workloads will not be able to overcome the shortcoming no matter how finitely resilient the workload is. However, we can list a `toleration` on the workload.

A `toleration` in a workload tells Lighthouse to ignore whatever value exists for a resource in a node at assignment time of the workload. So, in order to schedule a workload on the node listed above, we can simply add `"spiders"` to the toleration list for the workload:

```
...
"workloads": [
    {
        "id": "workload1",
        "requirements": {
            ...
        },
        "tolerations": [
            "spiders",
            ...
        ]
    }
]
```

1.5 Aversion Groups

Aversion Groups correspond to anti-affinity groups in other scheduling schemes.

Put simply, any aversion group listed for a workload causes that workload to "prefer" to be scheduled on a node without any other workloads listed as "belonging" to the same aversion group, like this::

```
...
"nodes": [
    {
        "id: "node1",
        "resources": {
           ...
        }
    },
    {
        "id: "node2",
        "resources": {
           ...
        }
    }

],
```

(continues on next page)

(continued from previous page)

```
"workloads": [
    {
        "id": "workload1",
        "requirements": {
            ...
        },
        "aversion_groups": [
            "io-bound",
            ...
        ]
    },
    {
        "id": "workload2",
        "requirements": {
            ...
        },
        "aversion_groups": [
            "io-bound",
            ...
        ]
    }
]
```

In the above example, both `workload1` and `workload2` will try really hard to be scheduled on different nodes, becuase they both list the `io-bound` aversion group in their aversion groups list.

CHAPTER 2

Changelog

All notable changes to this project will be documented here.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

2.1 Unreleased

2.1.1 Added

2.1.2 Changed

2.1.3 Fixed

2.2 0.1.0

2.2.1 Added

- Added *assign-workloads* RPC endpoint, allowing the assignment of workloads

2.2.2 Changed

2.2.3 Fixed

CHAPTER 3

Contributor Covenant Code of Conduct

3.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

3.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

3.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

3.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

3.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at "djhaskin987 at gmail.com" . All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

3.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant homepage, version 1.4.

CHAPTER 4

Contributing Guide

Thank you so much for considering contribution to Lighthouse!

First, please read the Lighthouse *Contributor Covenant Code of Conduct*. This project will not take any contribution coming from those who do not abide by the code of conduct. This means that if a person is currently under disciplinary action via avenues set forth in that document, we will ignore that person's PR and/or any issues they may log.

A contribution can be large or small, code or non-code. To make a contribution, first log a GitHub issue. Talk about what you want, and ask for other's opinions.

When you go to make the PR, please use the following checklist to test whether or not it is likely to be accepted:

1. **Is it based on the ‘‘develop‘‘ branch?** Lighthouse uses the git-flow framework for branch management. Please make PRs to the `develop` branch.
2. **Do you have tests in your PR, and do they pass?** Tests are in two places in lighthouse: the `test/Lighthouse` directory, where more or less normal unit tests reside; and the `api-test/` directory, where REST API-level tests reside. You must have at least an API-level test as a "spot-check" of your feature if the PR is to be merged. The test need not be elaborate; simple tests are better than no tests.
3. **Is your PR backwards compatible?** The biggest feature Lighthouse provides is backwards compatibility. If Lighthouse breaks a build, it is a bug. A PR is herein defined to be "backwards incompatible" if 1) it significantly changes the content of any previously merged unit or script test and 2) if it breaks any of them.
4. **Did you add documentation around the feature in your PR?** Generally this at least means adding something to the *Tour of Features* document.
5. **Did you add an entry to the Changelog?** This project keeps a curated *changelog*.

There are some exceptions to the above rules. If your patch is less than two lines' difference from the previous version, your PR may be a "typo" PR, which may qualify to get around some of the above rules. Just ask the team on your GitHub issue.

CHAPTER 5

Authors and Contributions

We thank everyone that has contributed to Degasolv!

5.1 Authors

- Daniel Jay Haskin

CHAPTER 6

Indices and tables

- genindex
- modindex
- search