Table of Contents

Introduction

ClusterFlux is a distributed InfluxDB node management and distribution library built entirely in Go programming language. It enables InfluxDB to be operated in a distributed clustered environment. ClusterFlux registers/deregisters InfluxDB nodes, syncs metadata across InfluxDB nodes, and enables InfluxDB nodes to perform peer to peer read/write of data (shards).

Using the Clusterflux Service built inside InfluxDB, InfluxDB communicates with the Clusterflux Application through http APIs. Clusterflux stores InfluxDB metadata as well as node info in a distributed key-value store called etcd.

Setup

ClusterFlux requires etcd version 2 for its setup. You can install etcd by following instructions on CoreOS etcd repository on Github. Current implementation supports etcd version 2. Once you have etcd cluster up and running you can verify the installation by running: `curl -L http://127.0.0.1:2379/version`.

If you are running etcd on server other than localhost, you can set the environment variable by running: `export CLUSTERFLUX_ETCD_ENDPOINTS=http://<etcd-endpoint>:port`

Next you need to generate ClusterFlux binaries from the ClusterFlux repository. You will need golang and might have to install dependencies. You can follow the steps below to generate the binaries:

1. Clone the repository in `$GOPATH/src/revision.aeip.apigee.net/monitoring/clusterflux`
2. Run `glide install`
3. Run `go install $(glide nv)`
4. Run `clusterflux`

ClusterFlux looks for the following set of environment variables which can be provided through a TOML file named config.toml and placed in `/etc/clusterflux/` or `$HOME/.clusterflux` or the current working directory from which it is being executed from. The configurables variables are listed below:

1. `CLUSTERFLUX_ETCD_ENDPOINTS` : used to specify the etcd etcd-endpoint

2. `CLUSTERFLUX_ETCD_TTL` : used to specify the time to keep a node marked alive without getting any heartbeat from that node

The InfluxDB binaries can be generated from Apigee's Github repository by following the steps below (You should also have git and mercurial installed):

1. Clone the repository in `$GOPATH/src/github.com/influxdata/influxdb`
2. InfluxDB uses gdm to manage dependencies. Install it by running the following:

```
go get github.com/sparrc/gdm
```

   i. Get the dependencies for the project by execute the following commands. The changes for ClusterFlux application are currently on `cflux-v1.0.0-beta3` branch, and as such you'd need to checkout the branch before proceeding further.

```
cd $GOPATH/src/github.com/influxdata/influxdb
git checkout --track -b cflux-v1.0.0-beta3 origin/cflux-v1.0.0-beta3
gdm restore
```

   i. To then build and install the binaries, run the following command.

```
go clean ./...
go install ./...
```

   The InfluxDB binary (called `influxd`) will be generated in the `$GOPATH/bin` path.

3. Run `influxd` to start the influxdb server.
4. You can test the cluster by hitting the URL: `http://localhost:8000/nodes`

For easiest setup, install Kubernetes and use the etcd.yaml file provided to start up three instances of etcd. All the yaml files are provided in the kubernetes folder in the ClusterFlux repository.

To generate the InfluxDb image run: `GO_VER=1.6.2 ./build-docker.sh` . This will generate an image called `influxdb` .

To generate ClusterFlux image, run: `docker build -t cflux .` . This will generate an image called cflux which will be used by the yaml files for kubernetes setup.

The cflux.yaml file can be used to start ClusterFlux pods. The cflux-deployment.yaml and cflux-rc.yaml can be used to start ClusterFlux deployments and replication-controller respectively. It's recommended to provide a load balancer or a service for the ClusterFlux application; the cflux-service.yaml can be used to start the ClusterFlux service at port 30000. The influx.yaml file can be used to start InfluxDb pods. If you are not using these yaml files, make sure you open the required ports for the communication.

All environment variables can be modified inside of meta.go or through the command line.

Introduction

Following are the list of APIs that ClusterFlux application provides to the InfluxDB nodes.

Register

`/nodes/{cluster}`

Registers an InfluxDB node with a specified cluster assigns an ID, and stores influxDB metadata and node data in etcd with a default time to live set for 15 seconds.

Automatically called from ClusterFlux Service when InfluxDB starts up.

Deregister

`/nodes/{cluster}/{id}`

Deregisters and deletes InfluxDB node data with a specified cluster and id from etcd.

Refresh

`/nodes/{cluster}/{id}`

Refreshes the time to live (TTL) for an InfluxDB node with {cluster} and {id}.

Automatically called from ClusterFlux Service every 10 seconds to tell ClusterFlux manager that the node is alive.

List

`/nodes`

Lists all InfluxDB nodes and ClusterFlux nodes

```
{
    "cflux": [
    {
        "id": 8278,
        "ip": "192.168.64.1,10.224.16.214,",
        "hostname": "cflux-host",
        "bind-address": "10.224.16.214:8000",
        "alive": true
    }
    ],
    "influx": [
    {
        "id": 8319,
        "ip": "10.224.16.214,",
        "hostname": "influx-host",
        "bind-address": "10.224.16.214:8888",
        "alive": true
    }
    ]
}
```

Cluster

`/nodes/{cluster}`

Lists all InfluxDB nodes in a specfied cluster.

```
[
     {
       "id": 8319,
       "ip": "10.224.16.214,",
       "hostname": "influx-host",
       "bind-address": "10.224.16.214:8888",
       "alive": true
    }
]
```

Write

`/clusters/{cluster}/versions/{version}`

Writes InfluxDB metadata from a specified cluster with a specified version. Uses etcd's prevIndex option. This allows a user to specify what version the metadata should be, so that old versions do not rewrite new versions. On a successful write, the version is

incremented from inside the ClusterFlux Manager. If an old version tries to write over a newer version, ClusterFlux will return the most recent version of the metadata to the node requesting to update.

Read

`/clusters/{cluster}/versions/{version}`

Reads metadata from a specified cluster and version. Read uses etcd's watcher to track changes on a specific data version. If a version is updated, the new metadata automatically gets returned.

Introduction

The Clusterflux Service (`cflux.Client`) is a wrapper on InfluxDB's `meta.Client` . The wrapper, while wraps some of the methods of `meta.Client` to implement the Clusterflux service, uses the existing implementations for all the methods; thus not introducing any new behavior to the InfluxDB itself. This wrapper service is responsible for registering the InfluxDB node in the cluster and for sending a heartbeat to the Clusterflux manager reporting it's alive status. Also, the service syncs the InfluxDB metadata with the remote Clusterflux manager. All http requests from the node to the remote manager is done with exponential backoff to avoid overwhelming the manager. A total of 5 attempts are made before reporting failure to connect.

Node Registration and Heartbeat

When a new InfluxDB node comes up, it checks the environment variables called `CLUSTER` and `CFLUX_ENDPOINT` . It uses the `CLUSTER` variable to register to the specified cluster. If no value is specified, it registers to a cluster named `default` . The `CFLUX_ENDPOINT` environment variable specifies the address of the remote Clusterflux manager. This address could be the load balancer's (for the Clusterflux managers) address or address of any one of the managers. These environment variables can be set through a configuration file or through the commandline itself.

The service reads the hostname and IP addresses of the InfluxDB machine and reports this metadata to the Clusterflux manager for book keeping. In future, this service will also report the bind address of the machine for the service-manager communication. Currently the bind address is set to the remote address of the http request sent from the node. Once the Clusterflux receives the node data sent by InfluxDB node, it creates a unique key for this node. This unique key (node ID) is then used for the heartbeat and all future communications. A heartbeat with this node ID is sent every 5 seconds to the Clusterflux manager reporting the alive status for the node.

Cluster Sync Service

The `cflux.Client` starts a sync service for the InfluxDB metadata syncing. It makes a http GET request to the manager with it's cluster name and current version of local metadata. The manager returns an updated metadata (with the new version) if the version reported in the request was older than the version on remote. If no new changes are found on remote in an interval of 5 minutes, the clusterflux manager sends a Gateway Timeout (to be changed to 204 - No content), a new request is made from the service.

Updating remote metadata

On any local changes to the InfluxDB metadata, a new database metadata snapshot is pushed to the cluster. The metadata version is maintained by the local as well as the remote Clusterflux manager. If the remote metadata is higher than the local version, the local metadata is updated followed by applying the new changes on the updated metadata. A retry to update the remote is then made. The Clusterflux manager on receiving a new update, updates it's metadata and version, notifying other InfluxDB nodes to update themselves.

Introduction

This file contains design decisions that are difficult to explain through code.

Cluster Read

Do a Get before Watcher. In ClusterFlux Application we do an extra Get before starting the watcher for long polling on updates.

Reason: The Watcher can be set to watch after some index, but the influxDB node's index might be too old for etcd to remember. Etcd keeps track of only a limited amount of history across all nodes. On the other hand if we use 'start from the current index', we might be waiting for the next update even if there's an update upstream; if influxDB is version 2 and remote cluster is at version 4 and if we start watching from current version (=4) we won't update influxDB unless a version 5 comes in. So the best approach would be to do a Get and check for version mismatch. If there's a version mismatch, return with the update, if not, start the watcher.

Extra post to remote on starting service

Currently we do an extra Post to ClusterFlux when Open() method of ClusterFlux Client is called. This is done to create a key in Etcd. If this is not done, the ClusterFlux sync service (service that polls for updates) starts throwing key not found; if this is the first node coming up in a new cluster.

Introduction

This file contains list of pending features that needs to be implemented.

1. Test cases for cluster meta watcher. Currently it's only testing read behavior. Would be good to have more granular tests. Most of the code is not covered through tests. This should be of highest priority.
2. Handle all errors. Some of the errors are not handled properly.
3. Benchmark Protobuf (serializing slice of protobuf vs serilizing protobuf for a slice (eg. point vs points proto))
4. All configurations should be from TOML (eg. Bind Address). Most of our configuration is through environment variables.
5. Look at the _internal database replication (remote/local?).
6. Handle scenarios where node goes down. Move data from existing replicas to the replaced node.
7. Create StatMap for ClusterFlux and log ClusterFlux metrics (remote write, meta update success/failure) to _internal database.
8. Kubernetes (with petset)
9. Upgrade etcd2 to etcd3.
10. UI for admins to monitor alive nodes, etc.
11. Store last 10 data versions of metadata. store recent versions as /cluster/history/ to etcd, say last 10 versions for a cluster
12. URL QueryUnescape in ClusterFlux application. Cluster names can have some special characters which might not work with our API unless escaped. We should escape all names in the request and response.
13. Ability to stop cluster read thread. There's a done channel on which the service listens to. We can implement a method that users can call to put a message on this channel to cancel the read thread. This is a low priority as the maximum life of long poll is anyways 5 minutes.
14. Fix the following bug (occurred during preparing for final presentation):

```
[clusterflux] 2016/08/17 03:44:00 Polling for updates from Clusterflux.
panic: runtime error: invalid memory address or nil pointer dereference
[signal 0xb code=0x1 addr=0x58 pc=0x68b888]

goroutine 43 [running]:
panic(0xcb8560, 0xc820010080)
    /usr/local/go/src/runtime/panic.go:481 +0x3e6
github.com/influxdata/influxdb/gossip.(*TSDBStore).WriteToShard(0xc8201826c0, 0x1,
 0xc82042ff80, 0x6, 0x8, 0x0, 0x0)
    /gopath/src/github.com/influxdata/influxdb/gossip/tsdbstore.go:85 +0x88
github.com/influxdata/influxdb/coordinator.(*PointsWriter).writeToShard(0xc8200725
a0, 0xc82034ba20, 0xc82011bbd0, 0x9, 0xddfc28, 0x7, 0xc82042ff80, 0x6, 0x8, 0x0, .
..)
    /gopath/src/github.com/influxdata/influxdb/coordinator/points_writer.go:323 +0
x2c1
github.com/influxdata/influxdb/coordinator.(*PointsWriter).WritePoints.func1(0xc82
02a47e0, 0xc8200725a0, 0xc82034ba20, 0xc82011bbd0, 0x9, 0xddfc28, 0x7, 0xc82042ff8
0, 0x6, 0x8)
    /gopath/src/github.com/influxdata/influxdb/coordinator/points_writer.go:278 +0
x8d
created by github.com/influxdata/influxdb/coordinator.(*PointsWriter).WritePoints
    /gopath/src/github.com/influxdata/influxdb/coordinator/points_writer.go:279 +0
x3e6
```

1. bindAddress is actually advertised address; peer address of the node. Not the address at which it listens on. The address it listens on is currently hard coded to :8888.