FLIP-7: Expose metrics to WebInterface

Status

Current state: *Released*

Discussion thread: -

JIRA: *FLINK-4389 - Expose metrics to Webfrontend* CLOSED

Released: Flink 1.2

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

With the introduction of the metric system it is now time to make it easily accessible to users. As the WebInterface is the first stop for users for any details about Flink, it seems appropriate to expose the gathered metrics there as well.

Public Interfaces

There will be no changes to public interfaces.

Proposed Changes

The changes can be roughly broken down into 4 steps:

1. Create a data-structure on the Job-/TaskManager containing a metrics snapshot
2. Transfer this snapshot to the WebInterface back-end
3. Store the snapshot in the WebRuntimeMonitor in an easily accessible way
4. Expose the stored metrics to the WebInterface via REST API

Note that this FLIP does not include the display of metrics.

Creating the snapshot on the Job-/TaskManager

The MetricRegistry will contain a MetricQueryService, which acts like an unscheduled reporter.
The service is a separate actor that creates and returns a Key-Value representation of the entire metric space when queried.

The keys represent the name of the metric; formatted according to the following scope format strings:

metrics.scope.jm	0:<user_scope>.<name>
metrics.scope.tm	1:<tm_id>:<user_scope>.<name>
metrics.scope.jm.job	2:<job_id>:<user_scope>.<name>
metrics.scope.tm.job	2:<job_id>:<user_scope>.<name>
metrics.scope.tm.task	3:<job_id>:<task_id>:<subtask_index>:<user_scope>.<name>
metrics.scope.tm.operator	4:<job_id>:<task_id>:<subtask_index>:<operator_name>:<user_scope>.<name>

The initial number serves as a category for the WebInterface, and allows for faster handling as we don't have to parse the entire string before deciding what category it belongs to.
0 = JobManager
1 = TaskManager
2 = Job
3 = Task
4 = Operator

The scope generation will be hard-coded into the separate metric groups, as ScopeFormats are a bit overkill for this. The created scopes are cached to avoid frequent re-computation.

The Value is the value returned by the metric or a method of the given metric (as Histograms expose multiple methods).

The Key-Value pairs can be stored in simple list-like data structure like an Object array.

Transfer to the WebInterface

With the "upcoming" separation of JobManager and WebInterface the transfer of TaskManager metrics should not be done completely through the JobManager, thus the TaskManagers should communicate directly with the WebInterface.

As there aren't any details as to how the separation will work, specifically whether a TaskManager -> WebInterface heartbeat will exist, i will assume that there is no message that we can piggyback on.

The WebRuntimeMonitor will contain a MetricFetcher which queries the JobManager for all available TaskManagers, and then query each of them for a metric dump. Metrics are only fetched if they actually accessed via REST calls, with a minimum time period (10 seconds) between updates.

The fetched metrics are merged and kept in a central location inside the MetricFetcher, available to different handlers.

Storage in the WebRuntimeMonitor

My proposal for a data-structure is the following:

```
MetricStore {
        void addMetric(String name, Object value);

        JobManagerMetricStore  jobMan

        class JobManagerMetricStore {
                Map<String, Object> metrics;
        }

        Map<String, TaskManagerMetricStore> taskmanagers;

        class TaskManagerMetricStore {
                Map<String, Object> metrics;
        }

        Map<String, JobMetricStore> jobs;

        class JobMetricStore {
                Map<String, Object> metrics;
                Map<String, TaskMetricStore > tasks;
        }

        class TaskMetricStore {
                Map<String, Object> metrics;
                Map<String, SubtaskMetricStore>;
        }

        class SubtaskMetricStore  {
                Map<String, Object> metrics;
        }
}
```

Note that at any given time only one of these objects will exist.

The WebInterface will execute a variety of queries regarding metrics of

- the job manager
- a specific task manager
- a specific job
- a specific (sub-)task

The proposed data structure supports the these queries without requiring any joins of sub structures. For example, in order to access a task metric we do not have to iterate through structures representing the different task managers, which would be the case if we were to mirror the MetricGroup structure.

Access to the structure should be guarded by a lock, so that not every map must a concurrent one.

Data Retirement

- JobManager metrics are kept indefinitely, as they have a limited size
- TaskManager metrics are kept as long as the given TaskManager is registered on the JobManager
- Job and Task metrics are kept as long as they are running or archived. In other words, if the job is listed in the WebInterface as either running or completed, the metrics are still available.

Access from the WebInterface

Several new Handlers will be added (one for each category) that will access the central MetricStore.

The REST calls for the list of available metrics look like this:

- JobManager: "/jobmanager/metrics"
- TaskManager: "/taskmanagers/:taskmanagerid/metrics"
- Job: "/jobs/:jobid/metrics"
- Task: "/jobs/:jobid/vertices/:vertexid/metrics"

This will return a JSON array containing the names of all available metrics.

The values for a list of metrics can be requested by appending "?get=[<metric_name1>[,<metric_nameX>]]"

This will return a JSON array containing "id":"values" pairs.

Prototype

A working prototype that follows this FLIP can be found here.

It allows the WebInterface to display task/operator metrics. Credits to Piotr Godek who provided the code to display of task metrics in the WebInterface.

Compatibility, Deprecation, and Migration Plan

-

Test Plan

Everything can be tested with unit tests.

Rejected Alternatives

-