# Expose SDK Harness status to runner

https://s.apache.org/beam-fn-api-harness-status
Yichi Zhang, Ankur Goenka
zyichi@google.com, goenka@google.com

*TL;DR: This doc describes the proposal to expose sdk harness status for better debuggability and manageability in runner with portability framework, particularly Dataflow runner.*

## Background

For runners with portability framework, typically there is a 1:N mapping between runner worker and SDK harness. Both may expose their job execution statuses separately.

Take Dataflow runner for example, both streaming worker and batch worker expose essential information such as worker's thread dump, memory and cpu usage etc through http sevlets for debugging purposes. The status of SDK Harness may be exposed separately (e.g. Dataflow python SDK). It is often quite inconvenient to look up the ports exposed by SDK Harness and gather all the information individually.

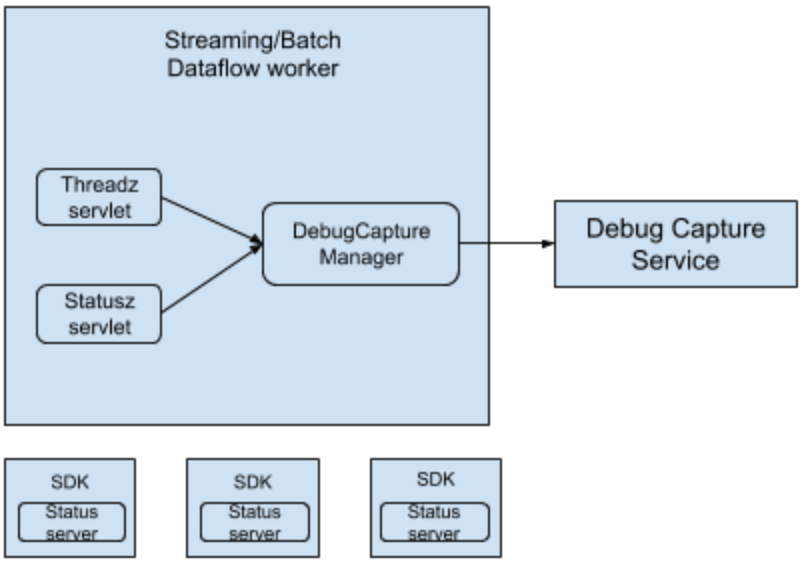
In addition, runners often have access to external status registry or sinks that enable storing snapshotted information which SDK harnesses do not have (such as Dataflow debug capture service), when debugging a finished pipeline, it is only possible to track the history of the runner.

We propose to expose the statuses of SDK Harnesses to runner through FnApi (thus can be captured and fed to the user in a more convenient way by runner) to improve the overall debuggability of portability framework.

## Current State(Dataflow)

We'll use Dataflow runner for the discussion below. Other runners may choose to implement similar functionalities when then debuggability improvement deems necessary.

The overall architecture related to status debugging of a Dataflow runners/SDKs is shown in the diagram below.




The Dataflow Runner implemented several servlets that can directly provide status pages, namely:

- HealthZ servlet, respond to /healthz with health information for runner jvm.
- HeapZ servlet, Respond to /heapz with a page allowing downloading of the heap dumps of runner jvm.
- StatusZ servlet, general servlet for providing a bunch of information on the statusz page.
- ThreadZ servlet, respond to /threadz with the stack traces of all running threads in runner jvm.

The python SDK implemented a StatusServer which provides a live thread dump of the SDK Harness. However users have to lookup the server address in worker log, which is often not straightforward.

While these http handlers provides current status information, the DebugCapture manager in Dataflow stream/batch worker also captures snapshots some of these debug info pages, thus making it possible to trace the status of runner in history. On the other hand, the status of SdkHarness is queryable while the pipeline is running, but the history will be untraceable if the pipeline failed or was canceled.

### Proposed Improvement

To allow traceback of SDK harness status when debugging a failed/stuck pipeline. We propose to add another api to Beam portability framework that allows runner to inspect the status of each SDK container that it connects to.

On a high level, there are two options how we can implement the rpc call:

**Option 1. Push model: SDK Harness pushes status info to runner periodically**

In this push model, SDK harness process will use a client to periodically report its relevant debug information to runner worker. Runner will record and update the sdk status cache it maintains. Debug capture manager will capture all latest responses from each individual sdk harness and report to debug capture service periodically.

Cons:

- SDK has to implement and expose separate http handler individually if we need to query live sdk status.

**Option 2(Implemented). Pull model: Runner pulls status info from SDKHarness periodically**

In this pull model, dataflow runner is responsible to initiate a status report RPC to SDK harnesses. SDK harnesses returns relevant status info in the response.

Cons:

- Need a mechanism to allow runner determine whether a particular SDK has implemented the status server for this rpc call.

The pull model has been chosen after discussion. The problem of determining SDK functional support from runner can largely be addressed by using a bidirectional streaming grpc, in which case SDK Harness initialize the connection and can be treated as a signal of being supportive for status report. And it also gives user convenience for real-time status query.

## Design considerations

**What content should the SDK status report contain**

Currently the most needed content is the thread dump from SDK harness which often provides good indication on the root cause of an issue. Other information such as pending bundle processed time, memory usage, etc, are also useful. It's up to the SDK harness to decide what are the relevant information to be included in the response, and the runner will not need to interpret the content of debug info, instead it can simply expose the content via an http page or write it to another sink.

Prefered status contents to include if applicable:

| Content | Description |
| --- | --- |
| Thread dump(Java, python), Goroutine stack traces(Go) | Snapshot of the state of all threads that are part of the SDK Harness process |
| Pending bundle processed time | Elapsed time for each bundle in processing if recorded |
| System | Memory/CPU usage etc |

Other ad-hoc info from the SDK can also be included as long as it is valuable for debugging and representative for SDK status. Note all the sections are optional and should only be included if they are truly relevant and accurate.

**What the Status API proto looks like**

In beam_fn_api.proto we define:

```
// Request from runner to SDK Harness asking for its status.
message WorkerStatusRequest {
  // (Required) Unique ID identifying this request.
  string id = 1;
}

// Response from SDK Harness to runner containing the debug related status info.
message WorkerStatusResponse {
  // (Required) Unique ID from the original request.
  string id = 1;
  // (Optional) Error message if exception encountered generating the status
response.
  string error = 2;
  // (Optional) Status debugging info reported by SDK harness worker.
  string status_info = 3;
}

// Fn Api for SDK harness to report its debug-related statuses to runner.
service BeamFnWorkerStatus {
  rpc WorkerStatus (stream WorkerStatusResponse)
    returns (stream WorkerStatusRequest) {}
}
```

To pass the status api endpoint on which the runner hosted the server, there are a few options:

1) Adding additional ApiServiceDescriptor information in provision API, so that the SDK harness can obtain it at boot time.
2) Reuse existing grpc channel such as the logging channel for the status api grpc stubs.
3) Adding additional flags to runner and SDK Harness container, and make sure at launch time the same value are shared between these containers. The same way as how logging and control endpoints are passed today.

The option 1) appears to be the more flexible one compared to the other two. The advantages: an empty ApiServiceDescriptor in provision response can be an indicator for SDK Harness to skip status report; less change required for the boot process of SDK Harness and runner.

Option 2) simplifies the port plumbing work, however, it prevents SDK Harness from knowing whether the runner has the status server hosted, thus requires extra error handling when runner is not supporting the API.

Options 3) has a major drawback that adding extra flags to boot.go in each SDK may end up breaking the backward-compatibility (passing new flags to older SDK container version will lead to fatal errors since they are unknown), extra version control flow is required for the job launching process.

For option 1), we will change the Provision API proto such as:

```
message ProvisionInfo {
    ...

    // (required) The artifact retrieval token produced by
    // ArtifactStagingService.CommitManifestResponse.
    string retrieval_token = 6;


    // (optional) The endpoint for runner to use for hosting the worker
    // status api server, specify this field only if the runner supports the
    // sdk worker status report.
    org.apache.beam.model.pipeline.v1.ApiServiceDescriptor status_endpoint = 7;

}
```

Note that there is a chance that provision server provides SDK harness with status_endpoint however runner doesn't support it, the SDK harness should skip connecting status report handler to the server if grpc connection error is encountered.

**Format requirement of status_info in the response**
There won't be a strong constraint on how the status content should be formatted, however it should be print-friendly and easy for user to identify each content section. Content should be formatted in a way that it should be appropriate to be used as an HTTP response body without additional rendering.

**Exposing SDK Harness status to user for pipeline**
Plan is to add another servlet in dataflow runner and expose an endpoint GET /sdk_status. For querying individual SDK Harness status, an id parameter should be specified such as GET /sdk_status?id=sdk0. When id parameter is missing, a summary of statuses from all connected SDK Harnesses rendered in html will be returned to user.

**Compatibility concerns**
As mentioned in previous sections, backward compatibility should be considered.
When SDK Harness supports the API but runner doesn't, sdk worker should fall back to skip this functionality. In another case, runner supports it and SDK harness doesn't, there won't be connections from SDK harness.