

The YARN Timeline Service v.2

- [Overview](#)
 - [Introduction](#)
 - [Architecture](#)
 - [Current Status](#)
- [Deployment](#)
 - [Configurations](#)
 - [Enabling the Timeline Service v.2](#)
- [Publishing of application specific data](#)
- [Timeline Service v.2 REST API](#)
 - [Query Flows](#)
 - [Query Flow Runs](#)
 - [Query Flow Run](#)
 - [Query Apps for a Flow](#)
 - [Query Apps for a Flow Run](#)
 - [Query App](#)
 - [Query Generic Entities](#)
 - [Query Generic Entity](#)

Overview

Introduction

YARN Timeline Service v.2 is the next major iteration of Timeline Server, following v.1 and v.1.5. V.2 is created to address two major challenges of v.1.

Scalability

V.1 is limited to a single instance of writer/reader and storage, and does not scale well beyond small clusters. V.2 uses a more scalable distributed writer architecture and a scalable backend storage.

YARN Timeline Service v.2 separates the collection (writes) of data from serving (reads) of data. It uses distributed collectors, essentially one collector for each YARN application. The readers are separate instances that are dedicated to serving queries via REST API.

YARN Timeline Service v.2 chooses Apache HBase as the primary backing storage, as Apache HBase scales well to a large size while maintaining good response times for reads and writes.

Usability improvements

In many cases, users are interested in information at the level of “flows” or logical groups of YARN applications. It is much more common to launch a set or series of YARN applications to complete a logic application. Timeline Service v.2 supports the notion of flows explicitly. In addition, it supports aggregating metrics at the flow level.

Also, information such as configuration and metrics is treated and supported as a first-class citizen.

Architecture

YARN Timeline Service v.2 uses a set of collectors (writers) to write data to the backend storage. The collectors are distributed and co-located with the application masters to which they are dedicated. All data that belong to that application are sent to the application level timeline collectors with the exception of the resource manager timeline collector.

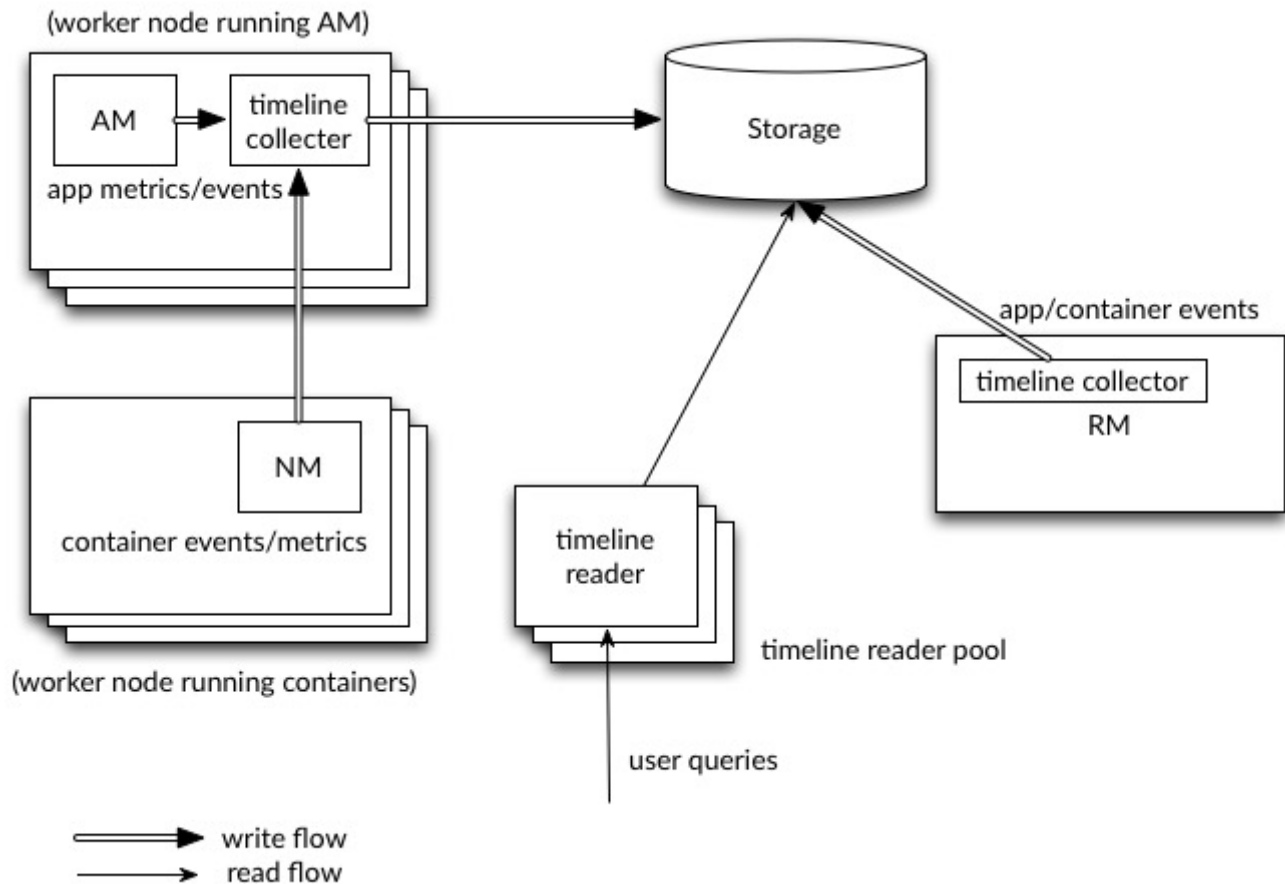
For a given application, the application master can write data for the application to the co-located timeline collectors (which is an NM auxiliary service in this release). In addition, node managers of other nodes that are running the containers for the application also write data to the timeline collector on the node that is running the application master.

The resource manager also maintains its own timeline collector. It emits only YARN-generic lifecycle events to keep its volume of writes reasonable.

The timeline readers are separate daemons separate from the timeline collectors, and they are dedicated to serving queries via

REST API.

The following diagram illustrates the design at a high level.



Current Status and Future Plans

YARN Timeline Service v.2 is currently in alpha. It is very much work in progress, and many things can and will change rapidly. Users should enable Timeline Service v.2 only on a test or experimental cluster to test the feature.

A complete end-to-end flow of writes and reads should be functional, with Apache HBase as the backend. You should be able to start generating data. When enabled, all YARN-generic events are published as well as YARN system metrics such as CPU and memory. Furthermore, some applications including Distributed Shell and MapReduce write per-framework data to YARN Timeline Service v.2.

The REST API comes with a good number of useful and flexible query patterns (see below for more information).

Although the basic mode of accessing data is via REST, it also comes with a basic web UI based on the proposed new YARN UI framework. Currently there is no support for command line access, however.

The collectors (writers) are currently embedded in the node managers as auxiliary services. The resource manager also has its dedicated in-process collector. The reader is currently a single instance. Currently, it is not possible to write to Timeline Service outside the context of a YARN application (i.e. no off-cluster client).

When YARN Timeline Service v.2 is disabled, one should expect no functional or performance impact on any other existing functionality.

The work to make it production-ready continues. Some key items include

- More robust storage fault tolerance
- Security
- Support for off-cluster clients
- More complete and integrated web UI
- Better support for long-running apps
- Offline (time-based periodic) aggregation for flows, users, and queues for reporting and analysis
- Timeline collectors as separate instances from node managers
- Clustering of the readers
- Migration and compatibility with v.1

Deployment

Configurations

New configuration parameters that are introduced with v.2 are marked bold.

Basic configuration

Configuration Property	Description
<code>yarn.timeline-service.enabled</code>	Indicate to clients whether Timeline service is enabled or not. If enabled, the <code>TimelineClient</code> library used by applications will post entities and events to the Timeline server. Defaults to <code>false</code> .
<code>yarn.timeline-service.version</code>	Indicate what is the current version of the running timeline service. For example, if “ <code>yarn.timeline-service.version</code> ” is 1.5, and “ <code>yarn.timeline-service.enabled</code> ” is true, it means the cluster will and should bring up the timeline service v.1.5 (and nothing else). On the client side, if the client uses the same version of timeline service, it should succeed. If the client chooses to use a smaller version in spite of this, then depending on how robust the compatibility story is between versions, the results may vary. Defaults to 1.0f.
<code>yarn.timeline-service.writer.class</code>	The class for the backend storage writer. Defaults to a filesystem storage writer, therefore it should be overridden.
<code>yarn.timeline-service.reader.class</code>	The class for the backend storage reader. Defaults to a filesystem storage reader, therefore it should be overridden.
<code>yarn.system-metrics-publisher.enabled</code>	The setting that controls whether yarn system metrics is published on the Timeline service or not by RM And NM. Defaults to <code>false</code> .
<code>yarn.rm.system-metrics-publisher.emit-container-events</code>	The setting that controls whether yarn container metrics is published to the timeline server or not by RM. This configuration setting is for ATS V2. Defaults to <code>false</code> .

Advanced configuration

Configuration Property	Description
<code>yarn.timeline-service.hostname</code>	The hostname of the Timeline service web application. Defaults to <code>0.0.0.0</code>
<code>yarn.timeline-service.address</code>	Address for the Timeline server to start the RPC server. Defaults to <code>\${yarn.timeline-service.hostname}:10200</code> .
<code>yarn.timeline-service.webapp.address</code>	The http address of the Timeline service web application. Defaults to <code>\${yarn.timeline-service.hostname}:8188</code> .
<code>yarn.timeline-service.webapp.https.address</code>	The https address of the Timeline service web application. Defaults to <code>\${yarn.timeline-service.hostname}:8190</code> .
<code>yarn.timeline-service.writer.flush-interval-seconds</code>	The setting that controls how often the timeline collector flushes the timeline writer. Defaults to 60.
<code>yarn.timeline-service.app-collector.linger-period.ms</code>	Time period till which the application collector will be alive in NM, after the application master container finishes. Defaults to 1000 (1 second).
<code>yarn.timeline-service.timeline-client.number-of-async-entities-to-merge</code>	Time line V2 client tries to merge these many number of async entities (if available) and then call the REST ATS V2 API to submit. Defaults to 10.
<code>yarn.timeline-service.coprocessor.app-final-value-retention-milliseconds</code>	The setting that controls how long the final value of a metric of a completed app is retained before merging into the flow sum. Defaults to 259200000 (3 days).

Enabling the Timeline Service v.2

Preparing Apache HBase cluster for storage

The first part is to set up or pick an Apache HBase cluster to use as the storage cluster. Once you have an HBase cluster ready to use for this purpose, perform the following steps.

First, add the timeline service jar to the HBase classpath in all HBase machines in the cluster. It is needed for the coprocessor as well as the schema creator. For example,

```
cp hadoop-yarn-server-timelineservice-3.0.0-SNAPSHOT.jar /usr/hbase/lib/
```

Then, enable the coprocessor that handles the aggregation. To enable it, add the following entry in region servers' hbase-site.xml file (generally located in the conf directory) as follows:

```
<property>
  <name>hbase.coprocessor.region.classes</name>
  <value>org.apache.hadoop.yarn.server.timelineservice.storage.flow.FlowRunCopro
</property>
```

Restart the region servers and the master to pick up the timeline service jar as well as the config change. In this version, the coprocessor is loaded statically (i.e. system coprocessor) as opposed to a dynamically (table coprocessor).

Finally, run the schema creator tool to create the necessary tables:

```
bin/hbase org.apache.hadoop.yarn.server.timelineservice.storage.TimelineSchemaCr
```

The TimelineSchemaCreator tool supports a few options that may come handy especially when you are testing. For example, you can use -skipExistingTable (-s for short) to skip existing tables and continue to create other tables rather than failing the schema creation.

Enabling Timeline Service v.2

Following are the basic configurations to start Timeline service v.2:

```
<property>
  <name>yarn.timeline-service.version</name>
  <value>2.0f</value>
</property>

<property>
  <name>yarn.timeline-service.enabled</name>
  <value>true</value>
</property>

<property>
  <name>yarn.timeline-service.writer.class</name>
  <value>org.apache.hadoop.yarn.server.timelineservice.storage.HBaseTimelineWrit
</property>

<property>
  <name>yarn.timeline-service.reader.class</name>
  <value>org.apache.hadoop.yarn.server.timelineservice.storage.HBaseTimelineRead
</property>

<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle,timeline_collector</value>
```

```

</property>

<property>
  <name>yarn.nodemanager.aux-services.timeline_collector.class</name>
  <value>org.apache.hadoop.yarn.server.timelineservice.collector.PerNodeTimeline
</property>

<property>
  <description>The setting that controls whether yarn system metrics is
  published on the Timeline service or not by RM And NM.</description>
  <name>yarn.system-metrics-publisher.enabled</name>
  <value>true</value>
</property>

<property>
  <description>The setting that controls whether yarn container events are
  published to the timeline service or not by RM. This configuration setting
  is for ATS V2.</description>
  <name>yarn.rm.system-metrics-publisher.emit-container-events</name>
  <value>true</value>
</property>

```

In addition, you may want to set the YARN cluster name to a reasonably unique name in case you are using multiple clusters to store data in the same Apache HBase storage:

```

<property>
  <name>yarn.resourcemanager.cluster-id</name>
  <value>my_research_test_cluster</value>
</property>

```

Also, add the `hbase-site.xml` configuration file to the client Hadoop cluster configuration so that it can write data to the Apache HBase cluster you are using.

Running Timeline Service v.2

Restart the resource manager as well as the node managers to pick up the new configuration. The collectors start within the resource manager and the node managers in an embedded manner.

The Timeline Service reader is a separate YARN daemon, and it can be started using the following syntax:

```
$ yarn-daemon.sh start timelinereader
```

Enabling MapReduce to write to Timeline Service v.2

To write MapReduce framework data to Timeline Service v.2, enable the following configuration in `mapred-site.xml`:

```

<property>
  <name>mapreduce.job.emit-timeline-data</name>
  <value>true</value>
</property>

```

Publishing application specific data

This section is for YARN application developers that want to integrate with Timeline Service v.2.

Developers can continue to use the `TimelineClient` API to publish per-framework data to the Timeline Service v.2. You only need to instantiate the right type of the client to write to v.2. On the other hand, the entity/object API for v.2 is different than v.1 as the object model is significantly changed. The v.2 timeline entity class is `org.apache.hadoop.yarn.api.records.timeline.service.TimelineEntity` whereas the v.1 class is `org.apache.hadoop.yarn.api.records.timeline.TimelineEntity`. The methods on `TimelineClient` suitable for writing to the Timeline Service v.2 are clearly delineated, and they use the v.2 types as arguments.

Timeline Service v.2 `putEntities` methods come in 2 varieties: `putEntities` and `putEntitiesAsync`. The former is a blocking operation which should be used for writing more critical data (e.g. lifecycle events). The latter is a non-blocking operation. Note that neither has a return value.

Creating a `TimelineClient` for v.2 involves passing in the application id to the factory method.

For example:

```
// Create and start the Timeline client v.2
TimelineClient client = TimelineClient.createTimelineClient(appId);
client.init(conf);
client.start();

try {
    TimelineEntity myEntity = new TimelineEntity();
    myEntity.setEntityType("MY_APPLICATION");
    myEntity.setEntityId("MyApp1")
    // Compose other entity info

    // Blocking write
    client.putEntities(entity);

    TimelineEntity myEntity2 = new TimelineEntity();
    // Compose other info

    // Non-blocking write
    timelineClient.putEntitiesAsync(entity);

} catch (IOException e) {
    // Handle the exception
} catch (RuntimeException e) {
    // In Hadoop 2.6, if attempts submit information to the Timeline Server fail mo
    // a RuntimeException will be raised. This may change in future releases, being
    // replaced with a IOException that is (or wraps) that which triggered retry fa
} catch (YarnException e) {
    // Handle the exception
} finally {
    // Stop the Timeline client
    client.stop();
}
```

As evidenced above, you need to specify the YARN application id to be able to write to the Timeline Service v.2. Note that currently you need to be on the cluster to be able to write to the Timeline Service. For example, an application master or code in the container can write to the Timeline Service, while an off-cluster MapReduce job submitter cannot.

You can create and publish your own entities, events, and metrics as with previous versions.

Application frameworks should set the "flow context" whenever possible in order to take advantage of the flow support Timeline

Service v.2 provides. The flow context consists of the following:

- Flow name: a string that identifies the high-level flow (e.g. "distributed grep" or any identifiable name that can uniquely represent the app)
- Flow run id: a monotonically-increasing sequence of numbers that distinguish different runs of the same flow
- (optional) Flow version: a string identifier that denotes a version of the flow

If the flow context is not specified, defaults are supplied for these attributes:

- Flow name: the YARN application name (or the application id if the name is not set)
- Flow run id: the application start time in Unix time (milliseconds)
- Flow version: "1"

You can provide the flow context via YARN application tags:

```
ApplicationSubmissionContext appContext = app.getApplicationSubmissionContext();

// set the flow context as YARN application tags
Set<String> tags = new HashSet<>();
tags.add(TimelineUtils.generateFlowNameTag("distributed grep"));
tags.add(TimelineUtils.generateFlowVersionTag("3df8b0d6100530080d2e0decf9e528e57c"));
tags.add(TimelineUtils.generateFlowRunIdTag(System.currentTimeMillis()));

appContext.setApplicationTags(tags);
```

Timeline Service v.2 REST API

Querying the Timeline Service v.2 is currently only supported via REST API; there is no API client implemented in the YARN libraries.

The v.2 REST API is implemented at under the path, `/ws/v2/timeline/` on the Timeline Service web service.

Here is an informal description of the API.

Root path

```
GET /ws/v2/timeline/
```

Returns a JSON object describing the service instance and version information.

```
{
  "About": "Timeline Reader API",
  "timeline-service-version": "3.0.0-SNAPSHOT",
  "timeline-service-build-version": "3.0.0-SNAPSHOT from fb0acd08e6f0b030d82eeb7cb",
  "timeline-service-version-built-on": "2016-04-11T23:15Z",
  "hadoop-version": "3.0.0-SNAPSHOT",
  "hadoop-build-version": "3.0.0-SNAPSHOT from fb0acd08e6f0b030d82eeb7cbfa54043763",
  "hadoop-version-built-on": "2016-04-11T23:14Z"
}
```

The following shows the supported queries on the REST API.

Query Flows

With Query Flows API, you can retrieve a list of active flows that had runs most recently. If REST endpoint without cluster name is used, cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken

HTTP request:

```
GET /ws/v2/timeline/clusters/{cluster name}/flows/
```

or

```
GET /ws/v2/timeline/flows/
```

Query Parameters Supported:

1. `limit` - If specified, defines the number of flows to return. The maximum possible value for limit is maximum value of Long. If it is not specified or has a value less than 0, then limit will be considered as 100.
2. `dateRange` - If specified is given as "[startdate]-[enddate]" (i.e. start and end date separated by "-") or single date. Dates are interpreted in yyyyMMdd format and are assumed to be in GMT. If a single date is specified, all flows active on that date are returned. If both startdate and enddate is given, all flows active between start and end date will be returned. If only startdate is given, flows active on and after startdate are returned. If only enddate is given, flows active on and before enddate are returned.

For example :

"daterange=20150711" returns flows active on 20150711.

"daterange=20150711-20150714" returns flows active between these 2 dates.

"daterange=20150711-" returns flows active on and after 20150711.

"daterange=-20150711" returns flows active on and before 20150711.

Example JSON Response:

```
[
  {
    "metrics": [],
    "events": [],
    "id": "test-cluster/1460419200000/sjlee@ds-date",
    "type": "YARN_FLOW_ACTIVITY",
    "createdtime": 0,
    "flowruns": [
      {
        "metrics": [],
        "events": [],
        "id": "sjlee@ds-date/1460420305659",
        "type": "YARN_FLOW_RUN",
        "createdtime": 0,
        "info": {
          "SYSTEM_INFO_FLOW_VERSION": "1",
          "SYSTEM_INFO_FLOW_RUN_ID": 1460420305659,
          "SYSTEM_INFO_FLOW_NAME": "ds-date",
          "SYSTEM_INFO_USER": "sjlee"
        },
        "isrelatedto": {},
        "relatesto": {}
      },
    ],
  },
]
```



```

{
  "metrics": [],
  "events": [],
  "id": "sjlee@ds-date/1460420587974",
  "type": "YARN_FLOW_RUN",
  "createdtime": 0,
  "info": {
    "SYSTEM_INFO_FLOW_VERSION": "1",
    "SYSTEM_INFO_FLOW_RUN_ID": 1460420587974,
    "SYSTEM_INFO_FLOW_NAME": "ds-date",
    "SYSTEM_INFO_USER": "sjlee"
  },
  "isrelatedto": {},
  "relatesto": {}
}
],
"info": {
  "SYSTEM_INFO_CLUSTER": "test-cluster",
  "UID": "test-cluster!sjlee!ds-date",
  "SYSTEM_INFO_FLOW_NAME": "ds-date",
  "SYSTEM_INFO_DATE": 1460419200000,
  "SYSTEM_INFO_USER": "sjlee"
},
"isrelatedto": {},
"relatesto": {}
}
]

```

Response Codes

1. If successful, a HTTP 200 (OK) response is returned.
2. If any problem occurs in parsing request, HTTP 400 (Bad Request) is returned.
3. For all other errors while retrieving data, HTTP 500 (Internal Server Error) is returned.

Query Flow Runs

With Query Flow Runs API, you can drill further down to get the runs (specific instances) of a given flow. This returns the most recent runs that belong to the given flow. If REST endpoint without cluster name is used, cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken.

HTTP request:

```

GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/runs/
or
GET /ws/v2/timeline/users/{user name}/flows/{flow name}/runs/

```

Query Parameters Supported:

1. `limit` - If specified, defines the number of flows to return. The maximum possible value for limit is maximum value of Long. If it is not specified or has a value less than 0, then limit will be considered as 100.
2. `createdTimeStart` - If specified, matched flow runs should not have been started before this timestamp.
3. `createdTimeEnd` - If specified, matched flow runs should not have started after this timestamp.

4. **metricsToRetrieve** - If specified, defines which metrics to retrieve or which ones not to retrieve and send back in response. **metricsToRetrieve** can be an expression of the form :
 (<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)
 This specifies a comma separated list of metric id prefixes. Only metrics matching either of the prefixes will be retrieved. Brackets are optional.
 Or, it can be an expression of the form :
 !(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)
 This specifies a comma separated list of metric id prefixes. Only metrics not matching either of the prefixes will be retrieved. If **metricsToRetrieve** is specified, metrics will be retrieved irrespective of whether **METRICS** is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.
5. **fields** - Specifies which fields to retrieve. For querying flow runs, only **ALL** or **METRICS** are valid fields. Other fields will lead to HTTP 400 (Bad Request) response. If not specified, in response, id, type, createdtime and info fields will be returned.

Example JSON Response:

```
[
  {
    "metrics": [],
    "events": [],
    "id": "sjlee@ds-date/1460420587974",
    "type": "YARN_FLOW_RUN",
    "createdtime": 1460420587974,
    "info": {
      "UID": "test-cluster!sjlee!ds-date!1460420587974",
      "SYSTEM_INFO_FLOW_RUN_ID": 1460420587974,
      "SYSTEM_INFO_FLOW_NAME": "ds-date",
      "SYSTEM_INFO_FLOW_RUN_END_TIME": 1460420595198,
      "SYSTEM_INFO_USER": "sjlee"
    },
    "isrelatedto": {},
    "relatesto": {}
  },
  {
    "metrics": [],
    "events": [],
    "id": "sjlee@ds-date/1460420305659",
    "type": "YARN_FLOW_RUN",
    "createdtime": 1460420305659,
    "info": {
      "UID": "test-cluster!sjlee!ds-date!1460420305659",
      "SYSTEM_INFO_FLOW_RUN_ID": 1460420305659,
      "SYSTEM_INFO_FLOW_NAME": "ds-date",
      "SYSTEM_INFO_FLOW_RUN_END_TIME": 1460420311966,
      "SYSTEM_INFO_USER": "sjlee"
    },
    "isrelatedto": {},
    "relatesto": {}
  }
]
```

Response Codes

1. If successful, a HTTP 200 (OK) response is returned.
2. If any problem occurs in parsing request or if an invalid field is specified in fields query param, HTTP 400 (Bad Request) is returned.
3. For all other errors while retrieving data, HTTP 500 (Internal Server Error) is returned.

Query Flow Run

With this API, you can query a specific flow run identified by cluster, user, flow name and run id. If REST endpoint without cluster name is used, cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken. Metrics are returned by default while querying individual flow run.

HTTP request:

```
GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/r
or
GET /ws/v2/timeline/users/{user name}/flows/{flow name}/runs/{run id}
```

Query Parameters Supported:

1. `metricsToRetrieve` - If specified, defines which metrics to retrieve or which ones not to retrieve and send back in response. `metricsToRetrieve` can be an expression of the form :
`(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)`
This specifies a comma separated list of metric id prefixes. Only metrics matching either of the prefixes will be retrieved. Brackets are optional.
Or, it can be an expression of the form :
`!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)`
This specifies a comma separated list of metric id prefixes. Only metrics not matching either of the prefixes will be retrieved. Please note that URL unsafe characters such as spaces will have to be suitably encoded.

Example JSON Response:

```
{
  "metrics": [
    {
      "type": "SINGLE_VALUE",
      "id": "org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter:BYTES_R",
      "aggregationOp": "NOP",
      "values": {
        "1465246377261": 118
      }
    },
    {
      "type": "SINGLE_VALUE",
      "id": "org.apache.hadoop.mapreduce.lib.output.FileOutputFormatCounter:BYTES",
      "aggregationOp": "NOP",
      "values": {
        "1465246377261": 97
      }
    }
  ],
  "events": [],
  "id": "varun@QuasiMonteCarlo/1465246348599",
  "type": "YARN_FLOW_RUN",
  "createdtime": 1465246348599,
  "isrelatedto": {},
  "info": {
    "UID": "yarn-cluster!varun!QuasiMonteCarlo!1465246348599",
    "SYSTEM_INFO_FLOW_RUN_END_TIME": 1465246378051,
    "SYSTEM_INFO_FLOW_NAME": "QuasiMonteCarlo",
    "SYSTEM_INFO_USER": "varun",
    "SYSTEM_INFO_FLOW_RUN_ID": 1465246348599
  }
}
```

```
    },
    "relatesto": {}
  }
}
```

Response Codes

1. If successful, a HTTP 200(OK) response is returned.
2. If any problem occurs in parsing request, HTTP 400 (Bad Request) is returned.
3. If flow run for the given flow run id cannot be found, HTTP 404 (Not Found) is returned.
4. For all other errors while retrieving data, HTTP 500 (Internal Server Error) is returned.

Query Apps for a flow

With this API, you can query all the YARN applications that are part of a specific flow. If REST endpoint without cluster name is used, cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken. If number of matching applications are more than the limit, most recent apps till the limit is reached, will be returned.

HTTP request:

```
GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/a
or
GET /ws/v2/timeline/users/{user name}/flows/{flow name}/apps
```

Query Parameters Supported:

1. **limit** - If specified, defines the number of applications to return. The maximum possible value for limit is maximum value of Long. If it is not specified or has a value less than 0, then limit will be considered as 100.
2. **createdTimeStart** - If specified, matched applications should not have been created before this timestamp.
3. **createdTimeEnd** - If specified, matched applications should not have been created after this timestamp.
4. **relatesTo** - If specified, matched applications should relate to or not relate to given entities associated with a entity type. `relatesto` is represented as an expression of the form :
`"(<entitytype>:<entityid>:<entityid>...,<entitytype>:<entityid>:<entityid>...) <op> !(<entitytype>:<entityid>:<entityid>...,<entitytype>:<entityid>:<entityid>...)"`.
 Here, "!" means all the entity type - entity id(s) relations specified within the enclosing brackets proceeding "!" should not exist. If "!" is not specified, they should exist. `op` is a logical operator and can be either AND or OR. entity type can be followed by any number of entity id(s). And we can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together.
For example : `relatesTo` can be `"(((type1:id1:id2:id3,type3:id9) AND !(type2:id7:id8)) OR (type1:id4))"`.
 Please note that URL unsafe characters such as spaces will have to be suitably encoded.
5. **isRelatedTo** - If specified, matched applications should be related to or not related to given entities associated with a entity type. `isRelatedTo` is represented in the same form as `relatesTo`.
6. **infofilters** - If specified, matched applications should have exact matches to the given info key and should be either equal or not equal to given value. The info key is a string but value can be any object. `infofilters` are represented as an expression of the form :
`"(<key> <compareop> <value>) <op> (<key> <compareop> <value>)"`.
 Here `op` can be either of AND or OR. And `compareop` can be either of "eq", "ne" or "ene".
 "eq" means equals, "ne" means not equals and existence of key is not required for a match and "ene" means not equals but existence of key is required. We can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together. *For example* : `infofilters` can be `"(((infokey1 eq value1) AND (infokey2 ne value1)) OR (infokey1 ene value3))"`.
 Please note that URL unsafe characters such as spaces will have to be suitably encoded.
7. **confilters** - If specified, matched applications should have exact matches to the given config name and should be either

equal or not equal to the given config value. Both the config name and value must be strings. confilters are represented in the same form as infilters.

8. **metricfilters** - If specified, matched applications should have exact matches to the given metric and satisfy the specified relation with the metric value. Metric id should be a string and metric value should be an integral value. metricfilters are represented as an expression of the form :
`"(<metricid> <compareop> <metricvalue>) <op> (<metricid> <compareop> <metricvalue>)"`.
 Here op can be either of AND or OR. And compareop can be either of "eq", "ne", "ene", "gt", "ge", "lt" and "le".
 "eq" means equals, "ne" means not equals and existence of metric is not required for a match, "ene" means not equals but existence of metric is required, "gt" means greater than, "ge" means greater than or equals, "lt" means less than and "le" means less than or equals. We can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together. *For example* : metricfilters can be `"(((metric1 eq 50) AND (metric2 gt 40)) OR (metric1 lt 20))"`.
 This in essence is an expression equivalent to `"(metric1 = 50 AND metric2 > 40) OR (metric1 < 20)"` Please note that URL unsafe characters such as spaces will have to be suitably encoded.
9. **eventfilters** - If specified, matched applications should contain or not contain the given events depending on the expression. eventfilters is represented as an expression of the form :
`"(<eventid>,<eventid>) <op> !(<eventid>,<eventid>,<eventid>)"`.
 Here, "!" means none of the comma-separated list of events within the enclosed brackets proceeding "!" should exist for a match to occur. If "!" is not specified, the specified events within the enclosed brackets should exist. op is a logical operator and can be either AND or OR. We can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together.
For example : eventfilters can be `"(((event1,event2) AND !(event4)) OR (event3,event7,event5))"`.
 Please note that URL unsafe characters such as spaces will have to be suitably encoded.
10. **metricsToRetrieve** - If specified, defines which metrics to retrieve or which ones not to retrieve and send back in response. metricsToRetrieve can be an expression of the form :
`(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)`
 This specifies a comma separated list of metric id prefixes. Only metrics matching either of the prefixes will be retrieved. Brackets are optional.
 Or, it can be an expression of the form :
`!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)`
 This specifies a comma separated list of metric id prefixes. Only metrics not matching either of the prefixes will be retrieved. If metricsToRetrieve is specified, metrics will be retrieved irrespective of whether METRICS is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.
11. **confsToRetrieve** - If specified, defines which configs to retrieve or which ones not to retrieve and send back in response. confsToRetrieve can be an expression of the form :
`(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)`
 This specifies a comma separated list of config name prefixes. Only configs matching either of the prefixes will be retrieved. Brackets are optional.
 Or, it can be an expression of the form :
`!(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)`
 This specifies a comma separated list of config name prefixes. Only configs not matching either of the prefixes will be retrieved.
 If confsToRetrieve is specified, configs will be retrieved irrespective of whether CONFIGS is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.
12. **fields** - Specifies which fields to retrieve. Possible values for fields can be EVENTS, INFO, CONFIGS, METRICS, RELATES_TO, IS_RELATED_TO and ALL. All fields will be retrieved if ALL is specified. Multiple fields can be specified as a comma-separated list. If fields is not specified, in response, app id, type (equivalent to YARN_APPLICATION), app createdtime and UID in info field will be returned.
13. **metricsLimit** - If specified, defines the number of metrics to return. Considered only if fields contains METRICS/ALL or metricsToRetrieve is specified. Ignored otherwise. The maximum possible value for metricsLimit can be maximum value of Integer. If it is not specified or has a value less than 1, and metrics have to be retrieved, then metricsLimit will be considered as 1 i.e. latest single value of metric(s) will be returned.

➡ Example JSON Response:

```
[
  {
    "metrics": [ ],
    "events": [ ],
    "type": "YARN_APPLICATION",
    "id": "application_1465246237936_0001",
    "createdtime": 1465246348599,
    "isrelatedto": { },
    "configs": { },
    "info": {
      "UID": "yarn-cluster!application_1465246237936_0001"
    },
    "relatesto": { }
  }
]
```

```

    },
    {
      "metrics": [ ],
      "events": [ ],
      "type": "YARN_APPLICATION",
      "id": "application_1464983628730_0005",
      "createdtime": 1465033881959,
      "isrelatedto": { },
      "configs": { },
      "info": {
        "UID": "yarn-cluster!application_1464983628730_0005"
      },
      "relatesto": { }
    }
  ]

```

Response Codes

1. If successful, a HTTP 200 (OK) response is returned.
2. If any problem occurs in parsing request, HTTP 400 (Bad Request) is returned.
3. For all other errors while retrieving data, HTTP 500 (Internal Server Error) is returned.

Query Apps for a flow run

With this API, you can query all the YARN applications that are part of a specific flow run. If REST endpoint without cluster name is used, cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken. If number of matching applications are more than the limit, most recent apps till the limit is reached, will be returned.

HTTP request:

```

GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/runs/{run id}/apps/
or
GET /ws/v2/timeline/users/{user name}/flows/{flow name}/runs/{run id}/apps/

```

Query Parameters Supported:

1. **limit** - If specified, defines the number of applications to return. The maximum possible value for limit is maximum value of Long. If it is not specified or has a value less than 0, then limit will be considered as 100.
2. **createdTimeStart** - If specified, matched applications should not have been created before this timestamp.
3. **createdTimeEnd** - If specified, matched applications should not have been created after this timestamp.
4. **relatesTo** - If specified, matched applications should relate to or not relate to given entities associated with a entity type. `relatesto` is represented as an expression of the form :
`"(<entitytype>:<entityid>:<entityid>...,<entitytype>:<entityid>:<entityid>...) <op> !(<entitytype>:<entityid>:<entityid>...,<entitytype>:<entityid>:<entityid>...)".`
 Here, "!" means all the entity type - entity id(s) relations specified within the enclosing brackets proceeding "!" should not exist. If "!" is not specified, they should exist. `op` is a logical operator and can be either AND or OR. entity type can be followed by any number of entity id(s). And we can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together.
For example : `relatesTo` can be `"(((type1:id1:id2:id3,type3:id9) AND !(type2:id7:id8)) OR (type1:id4))".`
 Please note that URL unsafe characters such as spaces will have to be suitably encoded.
5. **isRelatedTo** - If specified, matched applications should be related to or not related to given entities associated with a entity type. `isRelatedTo` is represented in the same form as `relatesTo`.
6. **infofilters** - If specified, matched applications should have exact matches to the given info key and should be either

equal or not equal to given value. The info key is a string but value can be any object. infofilters are represented as an expression of the form :

"(<key> <compareop> <value>) <op> (<key> <compareop> <value>)"

Here op can be either of AND or OR. And compareop can be either of "eq", "ne" or "ene".

"eq" means equals, "ne" means not equals and existence of key is not required for a match and "ene" means not equals but existence of key is required. We can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together. *For example* : infofilters can be "(((infokey1 eq value1) AND (infokey2 ne value1)) OR (infokey1 ene value3))".

Please note that URL unsafe characters such as spaces will have to be suitably encoded.

7. **confilters** - If specified, matched applications should have exact matches to the given config name and should be either equal or not equal to the given config value. Both the config name and value must be strings. confilters are represented in the same form as infofilters.
8. **metricfilters** - If specified, matched applications should have exact matches to the given metric and satisfy the specified relation with the metric value. Metric id should be a string and metric value should be an integral value. metricfilters are represented as an expression of the form :
 "(<metricid> <compareop> <metricvalue>) <op> (<metricid> <compareop> <metricvalue>)"
 Here op can be either of AND or OR. And compareop can be either of "eq", "ne", "ene", "gt", "ge", "lt" and "le".
 "eq" means equals, "ne" means not equals and existence of metric is not required for a match, "ene" means not equals but existence of metric is required, "gt" means greater than, "ge" means greater than or equals, "lt" means less than and "le" means less than or equals. We can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together. *For example* : metricfilters can be "(((metric1 eq 50) AND (metric2 gt 40)) OR (metric1 lt 20))".
 This in essence is an expression equivalent to "(metric1 = 50 AND metric2 > 40) OR (metric1 < 20)" Please note that URL unsafe characters such as spaces will have to be suitably encoded.
9. **eventfilters** - If specified, matched applications should contain or not contain the given events depending on the expression. eventfilters is represented as an expression of the form :
 "(<eventid>,<eventid>) <op> !(<eventid>,<eventid>,<eventid>)"
 Here, "!" means none of the comma-separated list of events within the enclosed brackets proceeding "!" should exist for a match to occur. If "!" is not specified, the specified events within the enclosed brackets should exist. op is a logical operator and can be either AND or OR. We can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together.
For example : eventfilters can be "(((event1,event2) AND !(event4)) OR (event3,event7,event5))".
 Please note that URL unsafe characters such as spaces will have to be suitably encoded.
10. **metricsToRetrieve** - If specified, defines which metrics to retrieve or which ones not to retrieve and send back in response. metricsToRetrieve can be an expression of the form :
 (<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)
 This specifies a comma separated list of metric id prefixes. Only metrics matching either of the prefixes will be retrieved. Brackets are optional.
 Or, it can be an expression of the form :
 !(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)
 This specifies a comma separated list of metric id prefixes. Only metrics not matching either of the prefixes will be retrieved. If metricsToRetrieve is specified, metrics will be retrieved irrespective of whether METRICS is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.
11. **confsToRetrieve** - If specified, defines which configs to retrieve or which ones not to retrieve and send back in response. confsToRetrieve can be an expression of the form :
 (<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)
 This specifies a comma separated list of config name prefixes. Only configs matching either of the prefixes will be retrieved. Brackets are optional.
 Or, it can be an expression of the form :
 !(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)
 This specifies a comma separated list of config name prefixes. Only configs not matching either of the prefixes will be retrieved.
 If confsToRetrieve is specified, configs will be retrieved irrespective of whether CONFIGS is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.
12. **fields** - Specifies which fields to retrieve. Possible values for fields can be EVENTS, INFO, CONFIGS, METRICS, RELATES_T0, IS_RELATED_T0 and ALL. All fields will be retrieved if ALL is specified. Multiple fields can be specified as a comma-separated list. If fields is not specified, in response, app id, type (equivalent to YARN_APPLICATION), app createdtime and UID in info field will be returned.
13. **metricsLimit** - If specified, defines the number of metrics to return. Considered only if fields contains METRICS/ALL or metricsToRetrieve is specified. Ignored otherwise. The maximum possible value for metricsLimit can be maximum value of Integer. If it is not specified or has a value less than 1, and metrics have to be retrieved, then metricsLimit will be considered as 1 i.e. latest single value of metric(s) will be returned.

Example JSON Response:

```
[
  {
    "metrics": [],
    "events": [],
```



```

    "id": "application_1460419579913_0002",
    "type": "YARN_APPLICATION",
    "createdtime": 1460419580171,
    "info": {
      "UID": "test-cluster!sjlee!ds-date!1460419580171!application_1460419579913_0002",
    },
    "configs": {},
    "isrelatedto": {},
    "relatesto": {}
  }
}

```

Response Codes

1. If successful, a HTTP 200 (OK) response is returned.
2. If any problem occurs in parsing request, HTTP 400 (Bad Request) is returned.
3. For all other errors while retrieving data, HTTP 500 (Internal Server Error) is returned.

Query app

With this API, you can query a single YARN application identified by cluster and application ID. If REST endpoint without cluster name is used, cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken. Flow context information i.e. user, flow name and run id are not mandatory but if specified in query param can preclude the need for an additional operation to fetch flow context information based on cluster and app id.

HTTP request:

```

GET /ws/v2/timeline/clusters/{cluster name}/apps/{app id}

or

GET /ws/v2/timeline/apps/{app id}

```

Query Parameters Supported:

1. `userid` - User id which should match for the application. Part of flow context information and if specified alongwith `flowname` and `flowrunid`, we would not need to fetch flow context information while executing this query.
2. `flowname` - Flow name which should match for the application. Part of flow context information and if specified alongwith `flowrunid` and `userid`, we would not need to fetch flow context information while executing this query.
3. `flowrunid` - Run id which should match for the application. Part of flow context information and if specified alongwith `flowname` and `userid`, we would not need to fetch flow context information while executing this query.
4. `metricsToRetrieve` - If specified, defines which metrics to retrieve or which ones not to retrieve and send back in response. `metricsToRetrieve` can be an expression of the form :
 (<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)
 This specifies a comma separated list of metric id prefixes. Only metrics matching either of the prefixes will be retrieved. Brackets are optional.
 Or, it can be an expression of the form :
 !(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)
 This specifies a comma separated list of metric id prefixes. Only metrics not matching either of the prefixes will be retrieved. If `metricsToRetrieve` is specified, metrics will be retrieved irrespective of whether `METRICS` is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.
5. `configsToRetrieve` - If specified, defines which configs to retrieve or which ones not to retrieve and send back in response. `configsToRetrieve` can be an expression of the form :
 (<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)
 This specifies a comma separated list of config name prefixes. Only configs matching either of the prefixes will be retrieved.

Brackets are optional.

Or, it can be an expression of the form :

!(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)

This specifies a comma separated list of config name prefixes. Only configs not matching either of the prefixes will be retrieved.

If configsToRetrieve is specified, configs will be retrieved irrespective of whether CONFIGS is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.

6. **fields** - Specifies which fields to retrieve. Possible values for fields can be EVENTS, INFO, CONFIGS, METRICS, RELATES_TO, IS_RELATED_TO and ALL. All fields will be retrieved if ALL is specified. Multiple fields can be specified as a comma-separated list. If fields is not specified, in response, app id, type (equivalent to YARN_APPLICATION), app createdtime and UID in info field will be returned.
7. **metricsLimit** - If specified, defines the number of metrics to return. Considered only if fields contains METRICS/ALL or metricsToRetrieve is specified. Ignored otherwise. The maximum possible value for metricsLimit can be maximum value of Integer. If it is not specified or has a value less than 1, and metrics have to be retrieved, then metricsLimit will be considered as 1 i.e. latest single value of metric(s) will be returned.

➤ Example JSON Response:

```
{
  "metrics": [],
  "events": [],
  "id": "application_1460419579913_0002",
  "type": "YARN_APPLICATION",
  "createdtime": 1460419580171,
  "info": {
    "UID": "test-cluster!sjlee!ds-date!1460419580171!application_1460419579913_00",
  },
  "configs": {},
  "isrelatedto": {},
  "relatesto": {}
}
```

➤ Response Codes

1. If successful, a HTTP 200(OK) response is returned.
2. If any problem occurs in parsing request, HTTP 400 (Bad Request) is returned.
3. If flow context information cannot be retrieved or application for the given app id cannot be found, HTTP 404 (Not Found) is returned.
4. For all other errors while retrieving data, HTTP 500 (Internal Server Error) is returned.

Query generic entities

With this API, you can query generic entities identified by cluster ID, application ID and per-framework entity type. If REST endpoint without cluster name is used, cluster specified by the configuration yarn.resourcemanager.cluster-id in yarn-site.xml is taken. Flow context information i.e. user, flow name and run id are not mandatory but if specified in query param can preclude the need for an additional operation to fetch flow context information based on cluster and app id. If number of matching entities are more than the limit, most recent entities till the limit is reached, will be returned. This endpoint can be used to query containers, application attempts or any other generic entity which clients put into the backend. For instance, we can query containers by specifying entity type as YARN_CONTAINER and application attempts by specifying entity type as YARN_APPLICATION_ATTEMPT.

➤ HTTP request:

```
GET /ws/v2/timeline/clusters/{cluster name}/apps/{app id}/entities/{entity type}
or
```

```
GET /ws/v2/timeline/apps/{app id}/entities/{entity type}
```

Query Parameters Supported:

- userid** - User id which should match for the entities. Part of flow context information and if specified alongwith flowname and flowrunid, we would not need to fetch flow context information while executing this query.
- flowname** - Flow name which should match for the entities. Part of flow context information and if specified alongwith flowrunid and userid, we would not need to fetch flow context information while executing this query.
- flowrunid** - Run id which should match for the entities. Part of flow context information and if specified alongwith flowname and userid, we would not need to fetch flow context information while executing this query.
- limit** - If specified, defines the number of entities to return. The maximum possible value for limit is maximum value of Long. If it is not specified or has a value less than 0, then limit will be considered as 100.
- createdTimeStart** - If specified, matched entities should not have been created before this timestamp.
- createdTimeEnd** - If specified, matched entities should not have been created after this timestamp.
- relatesTo** - If specified, matched entities should relate to or not relate to given entities associated with a entity type. relatesTo is represented as an expression of the form :
`"(<entitytype>:<entityid>:<entityid>...,<entitytype>:<entityid>:<entityid>...) <op> !(<entitytype>:<entityid>:<entityid>...,<entitytype>:<entityid>:<entityid>...)"`
 Here, "!" means all the entity type - entity id(s) relations specified within the enclosing brackets proceeding "!" should not exist. If "!" is not specified, they should exist. op is a logical operator and can be either AND or OR. entity type can be followed by any number of entity id(s). And we can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together.
For example : relatesTo can be `"(((type1:id1:id2:id3,type3:id9) AND !(type2:id7:id8)) OR (type1:id4))"`.
 Please note that URL unsafe characters such as spaces will have to be suitably encoded.
- isRelatedTo** - If specified, matched entities should be related to or not related to given entities associated with a entity type. isRelatedTo is represented in the same form as relatesTo.
- infofilters** - If specified, matched entities should have exact matches to the given info key and should be either equal or not equal to given value. The info key is a string but value can be any object. infofilters are represented as an expression of the form :
`"(<key> <compareop> <value>) <op> (<key> <compareop> <value>)"`
 Here op can be either of AND or OR. And compareop can be either of "eq", "ne" or "ene".
 "eq" means equals, "ne" means not equals and existence of key is not required for a match and "ene" means not equals but existence of key is required. We can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together. *For example* : infofilters can be `"(((infokey1 eq value1) AND (infokey2 ne value1)) OR (infokey1 ene value3))"`.
 Please note that URL unsafe characters such as spaces will have to be suitably encoded.
- conffilters** - If specified, matched entities should have exact matches to the given config name and should be either equal or not equal to the given config value. Both the config name and value must be strings. conffilters are represented in the same form as infofilters.
- metricfilters** - If specified, matched entities should have exact matches to the given metric and satisfy the specified relation with the metric value. Metric id should be a string and metric value should be an integral value. metricfilters are represented as an expression of the form :
`"(<metricid> <compareop> <metricvalue>) <op> (<metricid> <compareop> <metricvalue>)"`
 Here op can be either of AND or OR. And compareop can be either of "eq", "ne", "ene", "gt", "ge", "lt" and "le".
 "eq" means equals, "ne" means not equals and existence of metric is not required for a match, "ene" means not equals but existence of metric is required, "gt" means greater than, "ge" means greater than or equals, "lt" means less than and "le" means less than or equals. We can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together. *For example* : metricfilters can be `"(((metric1 eq 50) AND (metric2 gt 40)) OR (metric1 lt 20))"`.
 This in essence is an expression equivalent to `"(metric1 = 50 AND metric2 > 40) OR (metric1 < 20)"` Please note that URL unsafe characters such as spaces will have to be suitably encoded.
- eventfilters** - If specified, matched entities should contain or not contain the given events depending on the expression. eventfilters is represented as an expression of the form :
`"(<eventid>,<eventid>) <op> !(<eventid>,<eventid>,<eventid>)"`
 Here, "!" means none of the comma-separated list of events within the enclosed brackets proceeding "!" should exist for a match to occur. If "!" is not specified, the specified events within the enclosed brackets should exist. op is a logical operator and can be either AND or OR. We can combine any number of ANDs' and ORs' to create complex expressions. Brackets can be used to club expressions together.
For example : eventfilters can be `"(((event1,event2) AND !(event4)) OR (event3,event7,event5))"`.
 Please note that URL unsafe characters such as spaces will have to be suitably encoded.
- metricsToRetrieve** - If specified, defines which metrics to retrieve or which ones not to retrieve and send back in response. metricsToRetrieve can be an expression of the form :
`(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)`
 This specifies a comma separated list of metric id prefixes. Only metrics matching either of the prefixes will be retrieved. Brackets are optional.
 Or, it can be an expression of the form :
`!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)`
 This specifies a comma separated list of metric id prefixes. Only metrics not matching either of the prefixes will be retrieved.

If metricsToRetrieve is specified, metrics will be retrieved irrespective of whether METRICS is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.

14. **configsToRetrieve** - If specified, defines which configs to retrieve or which ones not to retrieve and send back in response. configsToRetrieve can be an expression of the form :
 (<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)
 This specifies a comma separated list of config name prefixes. Only configs matching either of the prefixes will be retrieved. Brackets are optional.
 Or, it can be an expression of the form :
 !(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)
 This specifies a comma separated list of config name prefixes. Only configs not matching either of the prefixes will be retrieved.
 If configsToRetrieve is specified, configs will be retrieved irrespective of whether CONFIGS is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.
15. **fields** - Specifies which fields to retrieve. Possible values for fields can be EVENTS, INFO, CONFIGS, METRICS, RELATES_TO, IS_RELATED_TO and ALL. All fields will be retrieved if ALL is specified. Multiple fields can be specified as a comma-separated list. If fields is not specified, in response, entity id, entity type, createdtime and UID in info field will be returned.
16. **metricsLimit** - If specified, defines the number of metrics to return. Considered only if fields contains METRICS/ALL or metricsToRetrieve is specified. Ignored otherwise. The maximum possible value for metricsLimit can be maximum value of Integer. If it is not specified or has a value less than 1, and metrics have to be retrieved, then metricsLimit will be considered as 1 i.e. latest single value of metric(s) will be returned.

➤ Example JSON Response:

```
[
  {
    "metrics": [ ],
    "events": [ ],
    "type": "YARN_APPLICATION_ATTEMPT",
    "id": "appattempt_1465246237936_0001_000001",
    "createdtime": 1465246358873,
    "isrelatedto": { },
    "configs": { },
    "info": {
      "UID": "yarn-cluster!application_1465246237936_0001!YARN_APPLICATION_ATTEMPT",
    },
    "relatesto": { }
  },
  {
    "metrics": [ ],
    "events": [ ],
    "type": "YARN_APPLICATION_ATTEMPT",
    "id": "appattempt_1465246237936_0001_000002",
    "createdtime": 1465246359045,
    "isrelatedto": { },
    "configs": { },
    "info": {
      "UID": "yarn-cluster!application_1465246237936_0001!YARN_APPLICATION_ATTEMPT",
    },
    "relatesto": { }
  }
]
```

➤ Response Codes

1. If successful, a HTTP 200(OK) response is returned.
2. If any problem occurs in parsing request, HTTP 400 (Bad Request) is returned.
3. If flow context information cannot be retrieved, HTTP 404 (Not Found) is returned.
4. For all other errors while retrieving data, HTTP 500 (Internal Server Error) is returned.

Query generic entity

With this API, you can query a specific generic entity identified by cluster ID, application ID, per-framework entity type and entity ID. If REST endpoint without cluster name is used, cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken. Flow context information i.e. user, flow name and run id are not mandatory but if specified in query param can preclude the need for an additional operation to fetch flow context information based on cluster and app id. This endpoint can be used to query a single container, application attempt or any other generic entity which clients put into the backend. For instance, we can query a specific YARN container by specifying entity type as `YARN_CONTAINER` and giving entity ID as container ID. Similarly, application attempt can be queried by specifying entity type as `YARN_APPLICATION_ATTEMPT` and entity ID being the application attempt ID.

HTTP request:

```
GET /ws/v2/timeline/clusters/{cluster name}/apps/{app id}/entities/{entity type}/
or
GET /ws/v2/timeline/apps/{app id}/entities/{entity type}/{entity id}
```

Query Parameters Supported:

- userid** - User id which should match for the entity. Part of flow context information and if specified alongwith flowname and flowrunid, we would not need to fetch flow context information while executing this query.
- flowname** - Flow name which should match for the entity. Part of flow context information and if specified alongwith flowrunid and userid, we would not need to fetch flow context information while executing this query.
- flowrunid** - Run id which should match for the entity. Part of flow context information and if specified alongwith flowname and userid, we would not need to fetch flow context information while executing this query.
- metricsToRetrieve** - If specified, defines which metrics to retrieve or which ones not to retrieve and send back in response. `metricsToRetrieve` can be an expression of the form :
 (<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)
 This specifies a comma separated list of metric id prefixes. Only metrics matching either of the prefixes will be retrieved. Brackets are optional.
 Or, it can be an expression of the form :
 !(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)
 This specifies a comma separated list of metric id prefixes. Only metrics not matching either of the prefixes will be retrieved. If `metricsToRetrieve` is specified, metrics will be retrieved irrespective of whether `METRICS` is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.
- confsToRetrieve** - If specified, defines which configs to retrieve or which ones not to retrieve and send back in response. `confsToRetrieve` can be an expression of the form :
 (<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)
 This specifies a comma separated list of config name prefixes. Only configs matching either of the prefixes will be retrieved. Brackets are optional.
 Or, it can be an expression of the form :
 !(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)
 This specifies a comma separated list of config name prefixes. Only configs not matching either of the prefixes will be retrieved.
 If `confsToRetrieve` is specified, configs will be retrieved irrespective of whether `CONFIGS` is specified in fields query param or not. Please note that URL unsafe characters such as spaces will have to be suitably encoded.
- fields** - Specifies which fields to retrieve. Possible values for fields can be `EVENTS`, `INFO`, `CONFIGS`, `METRICS`, `RELATES_TO`, `IS_RELATED_TO` and `ALL`. All fields will be retrieved if `ALL` is specified. Multiple fields can be specified as a comma-separated list. If `fields` is not specified, in response, entity id, entity type, createdtime and UID in info field will be returned.
- metricsLimit** - If specified, defines the number of metrics to return. Considered only if fields contains `METRICS/ALL` or `metricsToRetrieve` is specified. Ignored otherwise. The maximum possible value for `metricsLimit` can be maximum value of Integer. If it is not specified or has a value less than 1, and metrics have to be retrieved, then `metricsLimit` will be considered as 1 i.e. latest single value of metric(s) will be returned.

Example JSON Response:

```
{
  "metrics": [ ],
  "events": [ ],
  "type": "YARN_APPLICATION_ATTEMPT",
  "id": "appattempt_1465246237936_0001_000001",
  "createdtime": 1465246358873,
  "isrelatedto": { },
  "configs": { },
  "info": {
    "UID": "yarn-cluster!application_1465246237936_0001!YARN_APPLICATION_ATTEMPT!",
  },
  "relatesto": { }
}
```

➡ Response Codes

1. If successful, a HTTP 200 (OK) response is returned.
2. If any problem occurs in parsing request, HTTP 400 (Bad Request) is returned.
3. If flow context information cannot be retrieved or entity for the given entity id cannot be found, HTTP 404 (Not Found) is returned.
4. For all other errors while retrieving data, HTTP 500 (Internal Server Error) is returned.