The YARN Timeline Service v.2

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.

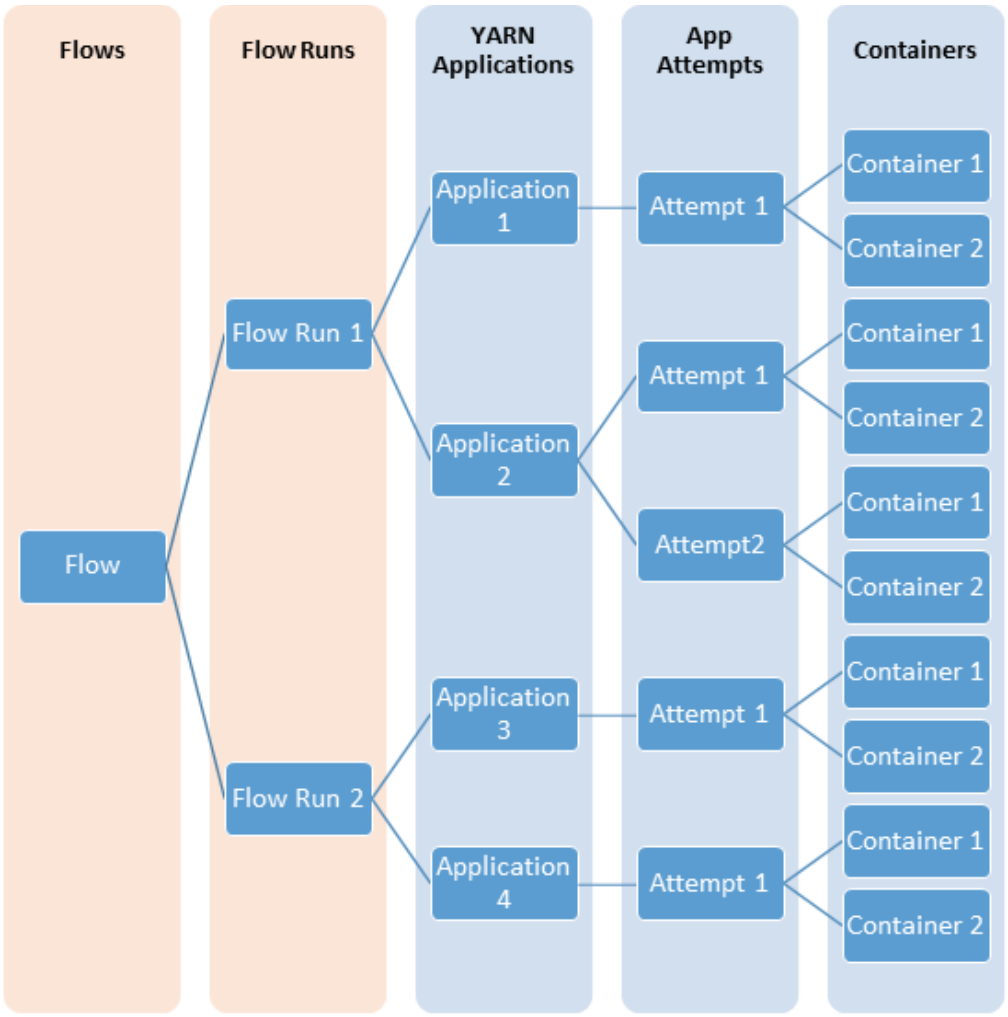
YARN Timeline Service v.2 chooses Apache HBase as the primary backing storage, as Apache HBase scales well to a large size while maintaining good response times for reads and writes.

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 logical 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 first-class citizens.

The following diagrams illustrates the relationship between different YARN entities modelling flows.



Architecture

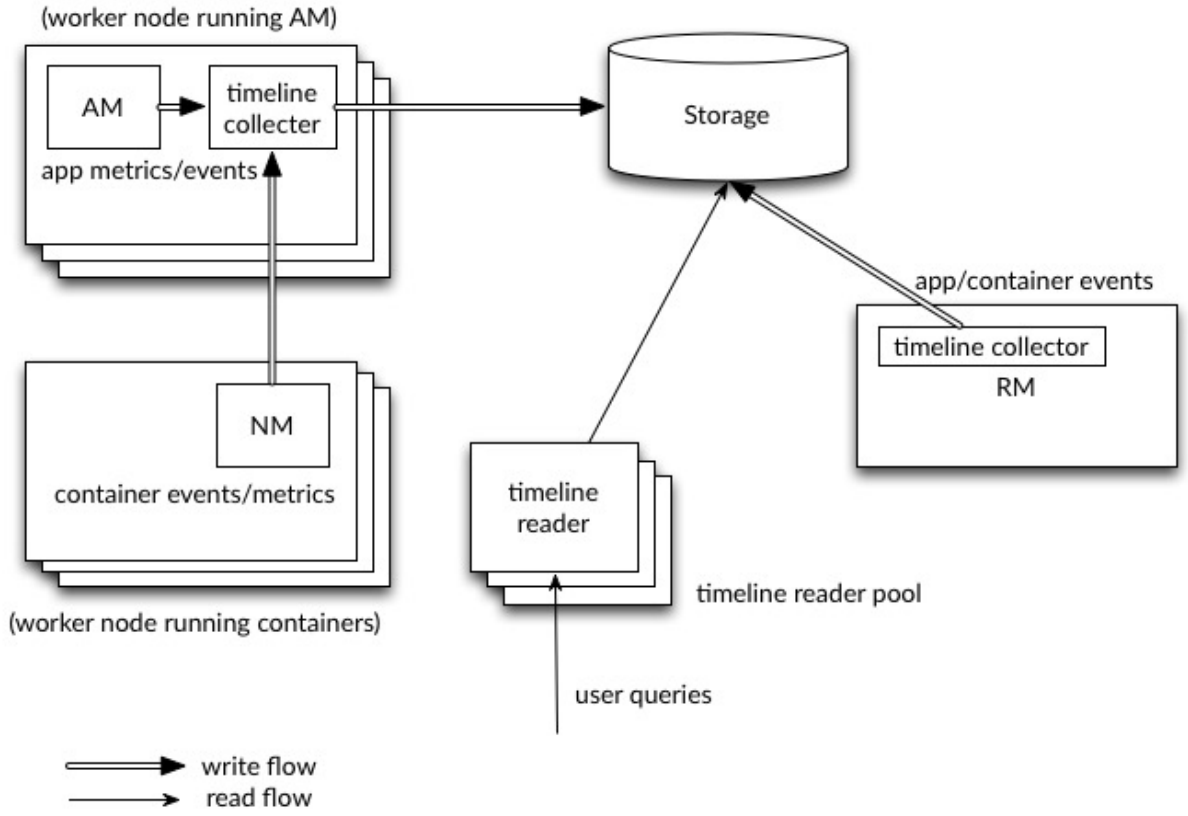
YARN Timeline Service v.2 uses a set of collectors (writers) to write data to the backend storage. The collectors are distributed and co-located with the application masters to which they are dedicated. All data that belong to that application are sent to the application level timeline collectors with 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 ("alpha 1"). It is very much work in progress, and many things can and will change rapidly. Users must enable Timeline Service v.2 only on a test or experimental cluster to test the feature.

Most importantly, **security is not enabled**. Do not set up or use Timeline Service v.2 until security is implemented if security is a requirement.

A complete end-to-end flow of writes and reads is 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 can write per-framework data to YARN Timeline Service v.2.

The basic mode of accessing data is via REST. Currently there is no support for command line access. The REST API comes with a good number of useful and flexible query patterns (see below for more information).

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 can expect no functional or performance impact on any other existing functionality.

The work to make it truly 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

Configuration Property	Description	Description

Configurations

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

Basic configuration

Configuration Property	Description
`yarn.timeline-service.enabled`	Indicate to clients whether Timeline service is enabled or not. If enabled, the `TimelineClient` library used by applications will post entities and events to the Timeline server. Defaults to `false`.
`yarn.timeline-service.version`	Indicate what is the current version of the running timeline service. For example, if "yarn.timeline-service.version" is 1.5, and "yarn.timeline-service.enabled" is true, it means the cluster will and must 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 must 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`.
`yarn.timeline-service.writer.class`	The class for the backend storage writer. Defaults to HBase storage writer.
`yarn.timeline-service.reader.class`	The class for the backend storage reader. Defaults to HBase storage reader.
`yarn.system-metrics-publisher.enabled`	The setting that controls whether yarn system metrics is published on the Timeline service or not by RM And NM. Defaults to `false`.

Advanced configuration

Configuration Property	Description
`yarn.timeline-service.hostname`	The hostname of the Timeline service web application. Defaults to `0.0.0.0`
`yarn.timeline-service.address`	Address for the Timeline server to start the RPC server. Defaults to `${yarn.timeline-service.hostname}:10200`.
`yarn.timeline-service.webapp.address`	The http address of the Timeline service web application. Defaults to `${yarn.timeline-service.hostname}:8188`.
`yarn.timeline-service.webapp.https.address`	The https address of the Timeline service web application. Defaults to `${yarn.timeline-service.hostname}:8190`.
`yarn.timeline-service.writer.flush-interval-seconds`	The setting that controls how often the timeline collector flushes the timeline writer. Defaults to `60`.
`yarn.timeline-service.app-collector.linger-period.ms`	Time period till which the application collector will be alive in NM, after the application master container finishes. Defaults to `1000` (1 second).
`yarn.timeline-service.timeline-client.number-of-async-entities-to-merge`	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`.
`yarn.timeline-service.hbase.coprocessor.app-final-value-retention-milliseconds`	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). This should be set in the HBase cluster.
`yarn.rm.system-metrics-publisher.emit-container-events`	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 `false`.

Enabling 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. The version of Apache HBase that is supported with Timeline Service v.2 is 1.1.x. The 1.0.x versions do not work with Timeline Service v.2. The 1.2.x versions have not been tested.

Once you have an Apache 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-alpha1-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.FlowRunCoprocessor</valu
</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.TimelineSchemaCreator
```

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.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.PerNodeTimelineCollectorsAu
</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 value 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.timelineservice.TimelineEntity` whereas the v.1 class is `org.apache.hadoop.yarn.api.records.timeline.TimelineEntity`. The methods on `TimelineClient` suitable for writing to 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 must 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 more than the
  // a RuntimeException will be raised. This may change in future releases, being
  // replaced with a IOException that is (or wraps) that which triggered retry failures.
} 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.

After creating the timeline client, user also needs to set the timeline collector address for the application. If `AMRMClient` is used then by registering the timeline client by calling `AMRMClient#registerTimelineClient` is sufficient.

```
amRMClient.registerTimelineClient(timelineClient);
```

Else address needs to be retreived from the AM allocate response and need to be set in timeline client explicitly.

```
timelineClient.setTimelineServiceAddress(response.getCollectorAddr());
```

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

TimelineEntity objects have the following fields to hold timeline data:

- events: A set of TimelineEvents, ordered by the timestamp of the events in descending order. Each event contains one id and a map to store related information and is associated with one timestamp.
- configs: A map from a string (config name) to a string (config value) representing all configs associated with the entity. Users can post the whole config or a part of it in the configs field. Supported for application and generic entities. Supported for application and generic entities.
- metrics: A set of metrics related to this entity. There are two types of metrics: single value metric and time series metric. Each metric item contains metric name (id), value, and what kind of aggregation operation should be performed in this metric (noop by default). Supported for flow run, application and generic entities.
- info: A map from a string (info key name) to an object (info value) to hold up related information for this entity. Supported for application and generic entities.
- isrelatedtoEntities and relatestoEntities: It is also possible to represent relationships between entities. Each entity contains a relatestoEntities and isrelatedtoEntities fields to represent relationships to other entities. Both fields are represented by a map from a string (the name of the relationship) to a timeline entity. In this way relationships among entities can be represented as a DAG.

Note that when posting timeline metrics, one may choose how each metric should be aggregated through the `TimelineMetric#setRealtimeAggregationOp()` method. The word "aggregate" here means applying one of the `TimelineMetricOperation` for a set of entities. Timeline service v2 provides built-in application level aggregation, which means aggregating metrics from different timeline entities within one YARN application. Right now, there are two kinds of operations supported in `TimelineMetricOperation`:

- `MAX`: Getting the maximum value among all `TimelineMetric` objects.
- `SUM`: Getting the sum of all `TimelineMetric` objects.

By default, the `NOP` operation means not performing any real-time aggregation operation.

Application frameworks must 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. Flow version can be used to identify changes in the flows, such as code changes or script changes.

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("3df8b0d6100530080d2e0decf9e528e57c42a90a"));
tags.add(TimelineUtils.generateFlowRunIdTag(System.currentTimeMillis()));

appContext.setApplicationTags(tags);
```

Timeline Service v.2 REST API

Querying 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-alpha1-SNAPSHOT",
  "timeline-service-build-version":"3.0.0-alpha1-SNAPSHOT from fb0acd08e6f0b030d82eeb7cbfa540
  "timeline-service-version-built-on":"2016-04-11T23:15Z",
  "hadoop-version":"3.0.0-alpha1-SNAPSHOT",
  "hadoop-build-version":"3.0.0-alpha1-SNAPSHOT from fb0acd08e6f0b030d82eeb7cbfa5404376313e60
  "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 the REST endpoint without the cluster name is used, the cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken. If none of the flows match the predicates, an empty list will be returned.

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 the yyyyMMdd format and are assumed to be in UTC. 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 non-recoverable 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 the REST endpoint without the cluster name is used, the cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken. If none of the flow runs match the predicates, an empty list will be returned.

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, then only flow runs started after this timestamp are returned.
3. `createdtimeend` - If specified, then only flow runs started before this timestamp are returned.
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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be of the form :
   !(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>…)
   This specifies a comma separated list of metric id prefixes. Only metrics not matching any 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 non-recoverable 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 the REST endpoint without the cluster name is used, the cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken. Metrics are returned by default while querying individual flow runs.

HTTP request:

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

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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be of the form :
   !(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>…)
   This specifies a comma separated list of metric id prefixes. Only metrics not matching any 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_READ",
      "aggregationOp": "NOP",
      "values": {
        "1465246377261": 118
      }
    },
    {
      "type": "SINGLE_VALUE",
```

```
      "id": "org.apache.hadoop.mapreduce.lib.output.FileOutputFormatCounter:BYTES_WRITTEN",
      "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 non-recoverable 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 the REST endpoint without the cluster name is used, the cluster specified by the configuration `yarn.resourcemanager.cluster-id` in `yarn-site.xml` is taken. If the number of matching applications are more than the limit, the most recent apps up to the limit will be returned. If none of the apps match the predicates, an empty list will be returned.

HTTP request:

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

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, then only applications created after this timestamp are returned.
3. `createdtimeend` - If specified, then only applications created before this timestamp are returned.
4. `relatesto` - If specified, matched applications must 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>…)".
   If relatesto expression has entity type - entity id(s) relations specified within enclosing brackets proceeding "!", this means

apps with these relations in its relatesto field, will not be returned. For expressions or subexpressions without "!", all apps which have the specified relations in its relatesto field, will be returned. "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 must 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 must have exact matches to the given info key and must 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. `conffilters` - If specified, matched applications must have exact matches to the given config name and must 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.
8. `metricfilters` - If specified, matched applications must have exact matches to the given metric and satisfy the specified relation with the metric value. Metric id must be a string and and metric value must 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 must 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 "!" must exist for a match to occur. If "!" is not specified, the specified events within the enclosed brackets must 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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be of the form:
!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>…)
This specifies a comma separated list of metric id prefixes. Only metrics not matching any 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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be 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 any 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 non-recoverable 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 the REST endpoint without the cluster name is used, the 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, the most recent apps up to the limit will be returned. If none of the apps match the predicates, an empty list will be returned.

HTTP request:

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

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, then only applications created after this timestamp are returned.
3. `createdtimeend` - If specified, then only applications created before this timestamp are returned.
4. `relatesto` - If specified, matched applications must 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>…)".
   If relatesto expression has entity type - entity id(s) relations specified within enclosing brackets proceeding "!", this means apps with these relations in its relatesto field, will not be returned. For expressions or subexpressions without "!", all apps which have the specified relations in its relatesto field, will be returned. "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 must 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 must have exact matches to the given info key and must 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. `conffilters` - If specified, matched applications must have exact matches to the given config name and must 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.
8. `metricfilters` - If specified, matched applications must have exact matches to the given metric and satisfy the specified relation with the metric value. Metric id must be a string and and metric value must 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 must 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 "!" must exist for a match to occur. If "!" is not specified, the specified events within the enclosed brackets must 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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be of the form :
    !(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>…)
    This specifies a comma separated list of metric id prefixes. Only metrics not matching any 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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be 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 any 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": [],
    "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 non-recoverable 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 the cluster and the application ID. If the REST endpoint without the cluster name is used, the 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` - If specified, only applications belonging to this user will be returned. This query param must be specified along with flowname and flowrunid query params, otherwise it will be ignored. If userid, flowname and flowrunid are not specified, we would have to fetch flow context information based on cluster and appid while executing the query.
2. `flowname` - Only applications belonging to this flowname will be returned. This query param must be specified along with userid and flowrunid query params, otherwise it will be ignored. If userid, flowname and flowrunid are not specified, we would have to fetch flow context information based on cluster and appid while executing the query.
3. `flowrunid` - Only applications belonging to this flow run id will be returned. This query param must be specified along with userid and flowname query params, otherwise it will be ignored. If userid, flowname and flowrunid are not specified, we would have to fetch flow context information based on cluster and appid while executing the 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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be of the form :
   !(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>…)
   This specifies a comma separated list of metric id prefixes. Only metrics not matching any 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. `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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be 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 any 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.
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_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. 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 non-recoverable 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 the REST endpoint without the cluster name is used, the 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, the most recent entities up to the limit 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`. If none of the entities match the predicates, an empty list will be returned.

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:

1. `userid` - If specified, only entities belonging to this user will be returned. This query param must be specified along with flowname and flowrunid query params, otherwise it will be ignored. If userid, flowname and flowrunid are not specified, we would have to fetch flow context information based on cluster and appid while executing the query.
2. `flowname` - If specified, only entities belonging to this flowname will be returned. This query param must be specified along with userid and flowrunid query params, otherwise it will be ignored. If userid, flowname and flowrunid are not specified, we would have to fetch flow context information based on cluster and appid while executing the query.
3. `flowrunid` - If specified, only entities belonging to this flow run id will be returned. This query param must be specified along with userid and flowname query params, otherwise it will be ignored. If userid, flowname and flowrunid are not specified, we would have to fetch flow context information based on cluster and appid while executing the query.
4. `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.
5. `createdtimestart` - If specified, then only entities created after this timestamp are returned.
6. `createdtimeend` - If specified, then only entities created before this timestamp are returned.
7. `relatesto` - If specified, matched entities must 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>…)".
   If relatesto expression has entity type - entity id(s) relations specified within enclosing brackets proceeding "!", this means entities with these relations in its relatesto field, will not be returned. For expressions or subexpressions without "!", all entities which have the specified relations in its relatesto field, will be returned. "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.
8. `isrelatedto` - If specified, matched entities must be related to or not related to given entities associated with a entity type. isrelatedto is represented in the same form as relatesto.
9. `infofilters` - If specified, matched entities must have exact matches to the given info key and must 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.

10. `conffilters` - If specified, matched entities must have exact matches to the given config name and must 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.
11. `metricfilters` - If specified, matched entities must have exact matches to the given metric and satisfy the specified relation with the metric value. Metric id must be a string and and metric value must 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.
12. `eventfilters` - If specified, matched entities must 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 "!" must exist for a match to occur. If "!" is not specified, the specified events within the enclosed brackets must 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.
13. `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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be of the form:
!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>…)
This specifies a comma separated list of metric id prefixes. Only metrics not matching any 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. `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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be 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 any 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.
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!appattempt
```

```
    },
    "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!appattempt
    },
    "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 non-recoverable 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 the REST endpoint without the cluster name is used, the 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}/{entity id}

or

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

Query Parameters Supported:

1. `userid` - If specified, entity must belong to this user. This query param must be specified along with flowname and flowrunid query params, otherwise it will be ignored. If userid, flowname and flowrunid are not specified, we would have to fetch flow context information based on cluster and appid while executing the query.
2. `flowname` - If specified, entity must belong to this flow name. This query param must be specified along with userid and flowrunid query params, otherwise it will be ignored. If userid, flowname and flowrunid are not specified, we would have to fetch flow context information based on cluster and appid while executing the query.
3. `flowrunid` - If specified, entity must belong to this flow run id. This query param must be specified along with userid and flowname query params, otherwise it will be ignored. If userid, flowname and flowrunid are not specified, we would have to fetch flow context information based on cluster and appid while executing the 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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be of the form:
   !(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>…)
   This specifies a comma separated list of metric id prefixes. Only metrics not matching any 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. `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 any of the prefixes will be retrieved. Brackets are optional for the simple expression. Alternatively, expressions can be 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 any 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.
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, entity id, entity type, 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": [ ],
  "type": "YARN_APPLICATION_ATTEMPT",
  "id": "appattempt_1465246237936_0001_000001",
  "createdtime": 1465246358873,
  "isrelatedto": { },
  "configs": { },
  "info": {
    "UID": "yarn-cluster!application_1465246237936_0001!YARN_APPLICATION_ATTEMPT!appattempt_1
  },
  "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 non-recoverable errors while retrieving data, HTTP 500 (Internal Server Error) is returned.