

ATSv2 Authentication

Varun Saxena

This document is intended to cover design for ATsv2 Authentication

Authentication will be achieved via Kerberos Authentication and delegation tokens. If authentication is enabled, each node collector manager will load a TimelineAuthenticationFilter. We will reuse this filter from ATsv1. The filter does the following:

1. Login with Kerberos Principal and uses HTTP SPNEGO to authenticate HTTP requests.
2. After Kerberos authentication is successful, generates a delegation token(DT) using the supplied timeline delegation token manager, for following communication.
3. Identify the Delegation token in subsequent requests and if it is valid, allow the request to pass through to the Web Service.

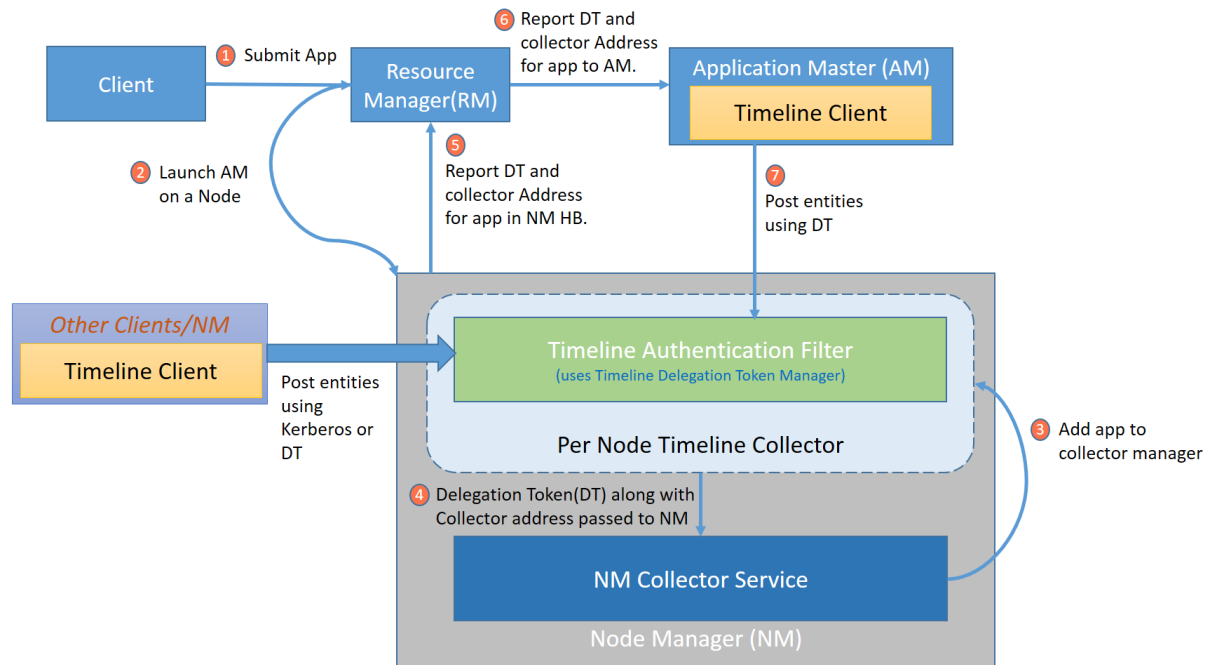
We will reuse the DelegationTokenAuthenticatedURL class for communication from TimelineClient to ensure above flow works. Using this, TimelineClient will use Kerberos authentication for first request (as DT is not present) and delegation token thereafter.

This should be enough for communication from clients where Kerberos is available but for posting entities from AM where Kerberos may not be available we need a mechanism to pass delegation token to AM so that it can be authorized to write entities to YARN ATS.

This can be done as under:

1. Whenever an app is submitted to RM, AM container will be allocated on a particular node.
2. NM on receiving an AM container request, adds the App level timeline collector via the Per node timeline collector aux service to Node collector manager. Collector Manager binds to any port.
3. After the application is added to collector manager, the collector address is returned back to NM. Collector manager though can additionally generate a delegation token (using timeline delegation token manager shared with timeline authentication filter) and pass it on to NM.
4. The delegation token generated in step above and collector address can then be passed to RM and later RM can pass on this information to AM.
5. AM can update this token when it receives it in Allocate Response and use it to post entities to Collector.
6. Delegation Token Manager inside Collector Manager can manage the lifecycle of a token as it will be notified when an app is removed (which in turn means the respective app collector is removed).

This is explained further via the diagram below.



Note: Currently app collectors run within NM. All the collectors are managed by NMCollectorManager. In future though, a collector can run outside NM as a standalone process (which handles a set of apps for writing entities to backend) or we can even consider launching app level collectors as containers. Both will lead to additional communication, not captured in the diagram above. Please note, if app collectors are launched as containers, there will be a separate timeline auth filter and corresponding delegation token manager for each app collector instead of having it at the collector manager level.

PerNodeTimelineCollector displayed a separate box in the diagram above is to indicate that the auxiliary service can run outside NM as well.

Other possible approaches:

Approach 2: RM generates timeline token on behalf of Collector and loads it in Application Submission Context

One of the drawbacks of first approach is that AM needs to do special handling for loading timeline tokens received in Allocate Response.

We can however let RM generate a timeline token as soon as an App is submitted. And add it in Application Submission Context. When a request for starting AM container is sent to NM, the collector will be started as well. We can thus extract the timeline token from the context and add it to timeline delegation token manager (corresponding to the node collector manager).

The token will be made available to AM as well as it is in the launch context. But as collector is not the one generating the token, IDs' may clash. To handle this, the kind of token generated by RM and collector manager can be different. But, required changes can be made to timeline authentication filter so that it will be able to identify both kinds of tokens for authentication.

Approach 3: NM generates timeline token on behalf of Collector and loads it in AM launch Context

Same as approach 2, but instead of RM, NM generates the token and populates it in AM container launch context. The same token can then be passed onto collector manager.

Alternatively, NM can wait for collector manager to generate a token before launching the AM. This however may lead to delay in launching of AM especially when collectors run outside NM. So it's not a feasible option.

Authentication for Timeline Reader

Like collectors, Timeline Reader will load the TimelineAuthenticationFilter on startup as well. And using it, requests sent to reader can be authenticated.

Additional Points:

1. Like for other modules, configurations will be introduced for loading keytab files and Kerberos principals in collectors and reader.
2. Delegation tokens need to be persisted to a state store to handle scenarios where the collector or NM (if collector is running inside NM) goes down. We can use Level DB to achieve this.