Job group management

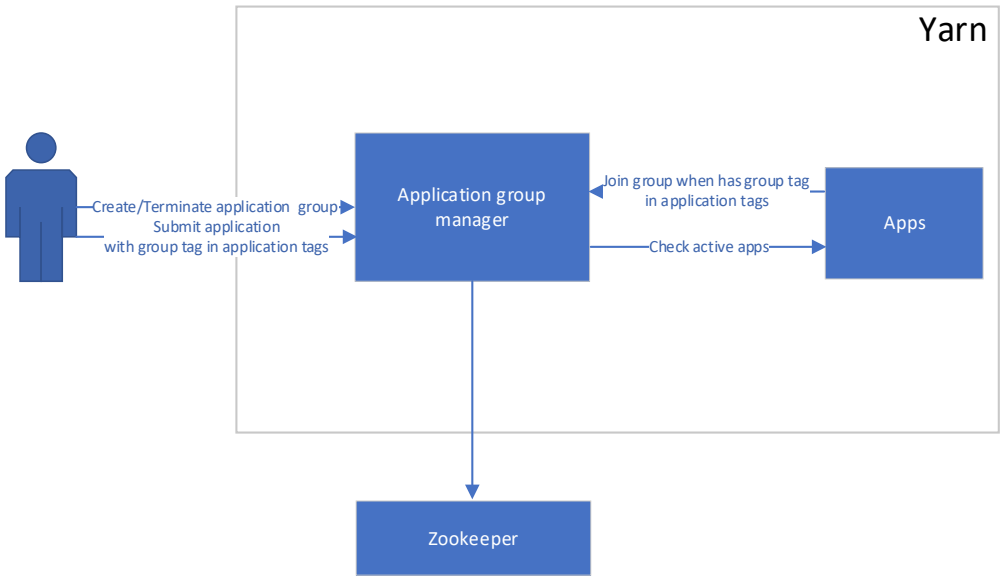
Contents

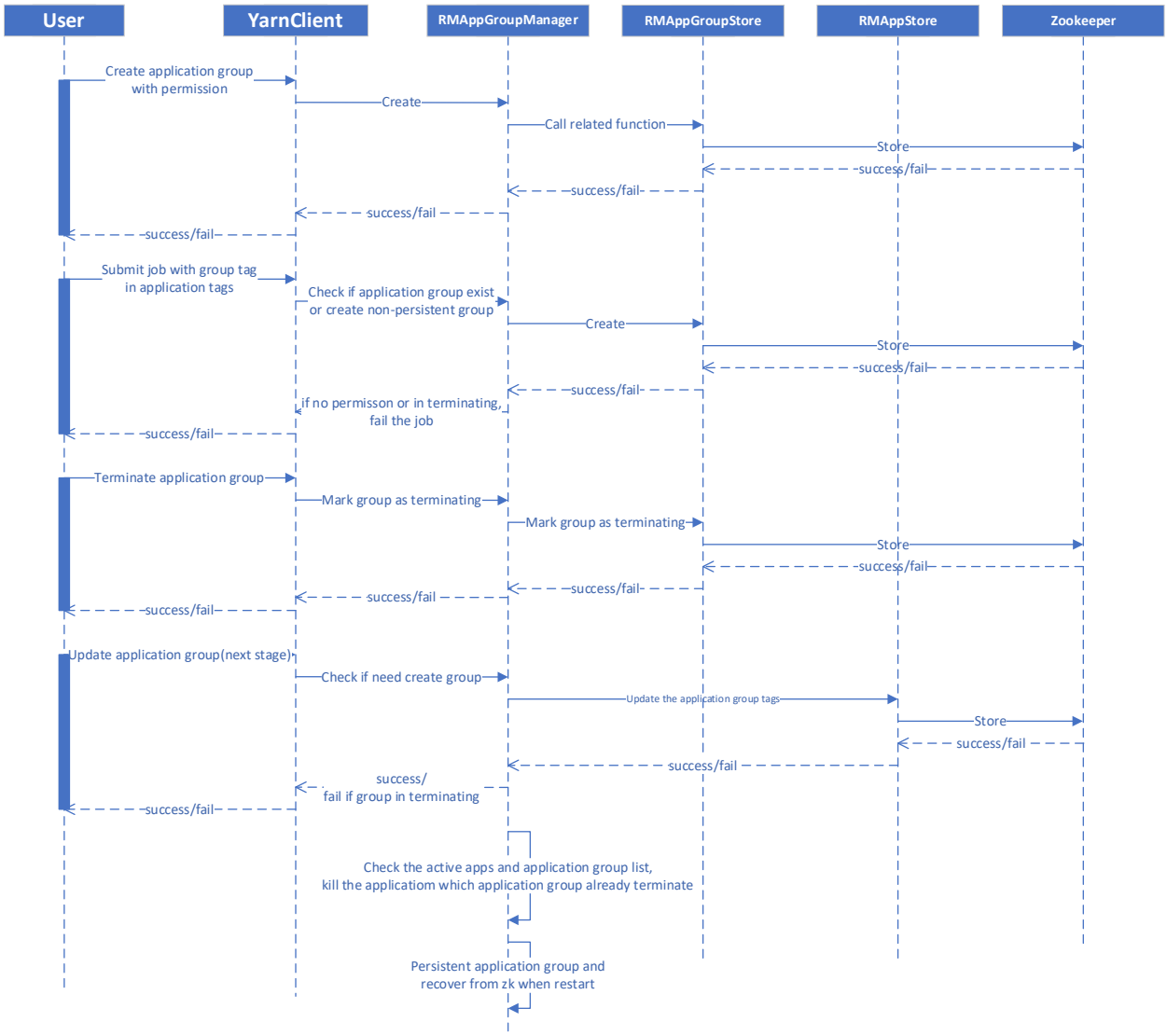
1. Abstracts:

In current yarn job management, we don't have an efficient mechanism to manage several jobs together. For example, one batch job may trigger several sub-jobs to running at the same time, like one job to process the data and another one monitor job metrics. And when we want to cancel these jobs, we must kill them one by one in current design. I proposal a job group concept to handle such parent-child jobs as one unit.
In this doc, we design creating, terminating, and listing group status API for user to operate. And user use group tag(GID- as prefix) in application tags to organize job group. If user want to kill all the parent-child jobs, they can call the terminate API, then yarn will scan all the jobs and kill them.

2. Architect and Workflow design:

In this section, we introduce the architect of app group and the workflow for creating, terminating, and listing group status API for user.

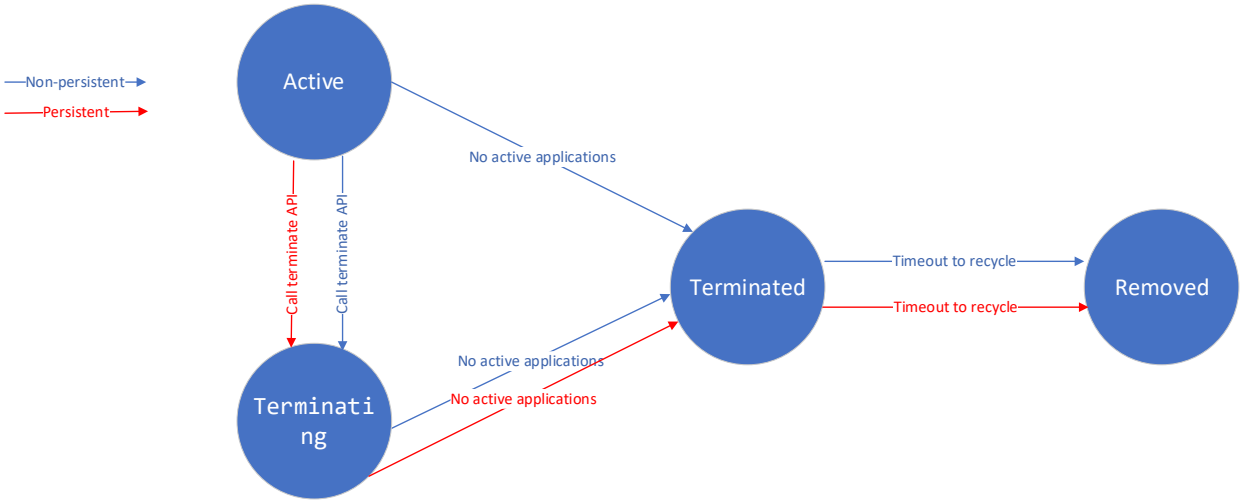




1) We provide command line and rest API to let user create and terminate application groups. The create operation will have permission support. And we provide two type application groups, one is persistent and another is non-persistent. Persistent application group can only be created by API.
2) First, user can create an application group with permission (currently, we support * and alias) by API or use implicit creation by submitting job with application tags.
3) When user submit job with group application tags (start with GID-), application group manager will check the application group existent(if not exist, will create one non-persistent implicitly) and permission. Or if group in terminating or terminated, will reject the job submission with error msg.
4) If user call terminate group API, application group manager will mark the group with terminating. And application group manager will scan all active apps list to see if app's group in terminating status. If so, will kill it.
5) Application group manager will recover job group list from zookeeper when RM restart.

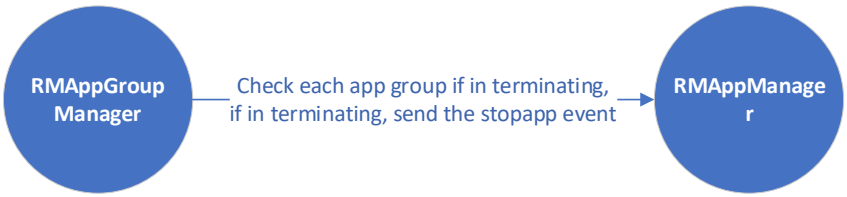
6) User can also use API to update application group(next stage).

3. Group life cycle:



We have different life cycle for persistent application group and non-persistent application group. For persistent group, it can only be recycled by cmd manually. And non-persistent application group will be recycled when no active applications.

4. Terminating logic:



RMAppGroupManager will have a thread polling each apps' group in RMAppGroupManager, if the app group in terminating status, will send stop-app event to kill the app.

5. API design:

1) Create application group:
    a. CMD: yarn applicationgroup -create <groupname> [-permission < permission>, -persistent, default permission is alias(owner). Currently support *, alias.
    b. Rest-API: post http://rm-http-address:port/ws/v1/cluster/appgroups

    Headers: {

    Authorization: 'Bearer ${token}',

    }

Body: {

"groupName":"testgroup",

"permission":"joinACL:*;adminACL:*",

"persistent":"true"

}

Response:

HTTP 200 OK

{

"message":"Create $groupId successfully."

}

HTTP 400 bad request

{

"message":"$groupId already exist."

}

Status: 500

{

"code": "UnknownError",

"message": "*Upstream error messages*"

}

2) Terminate application group:
   a. CMD: yarn applicationgroup -terminate <groupname>
   b. Delete http://rm-http-address:port/ws/v1/cluster/appgroups/:groupname

```
Headers: {

    Authorization: 'Bearer ${token}',

}
```

Response:

```
HTTP 202 OK

{

"message":" terminate $groupId successfully, accepted, will stop related
jobs shortly"

}
```

```
-----------------or-----------------
HTTP 404 Not Found
{}
Status: 500
{
  "code": "UnknownError",
  "message": "*Upstream error messages*"
}
```

3) List all application groups status:
   a. CMD: yarn applicationgroup -list
   b. Get http://rm-http-address:port/ws/v1/cluster/appgroups

```
Headers: {
    Authorization: 'Bearer ${token}',
}
Response:
HTTP 200 OK
{
   "groupName":"testgroup",
    "permission":"joinACL:*,terminateACL:*",
    "persistent":"true",
    "status":"active",
},
…
]
}
Status: 500
{
  "code": "UnknownError",
  "message": "*Upstream error messages*"
}
```