# Continuous resource-localization for YARN containers

Jian He with inputs from Vinod Kumar Vavilapalli, Hitesh Shah, Varun Vasudev

## 1) Goal

The goal is to enable NodeManagers to localize resources while container is running. Today, resource-localization is always the first step before starting a container. It will be useful if YARN can localize the resources continuously even while container is running.

## 2) Use cases

- Upgrade a running container (e.g. a zookeeper instance) with new artifacts. YARN should enable applications to reuse an existing allocation but restart the container with a modified set of *LocalResources*.
- While a container (e.g. Apache Tez task) is running, it may require new jars at runtime to access various other components.
- Re-configuring a container: Download new config files for a container while it is running.

## 3) Proposal

The general idea is to make localization a first-class API concept in YARN

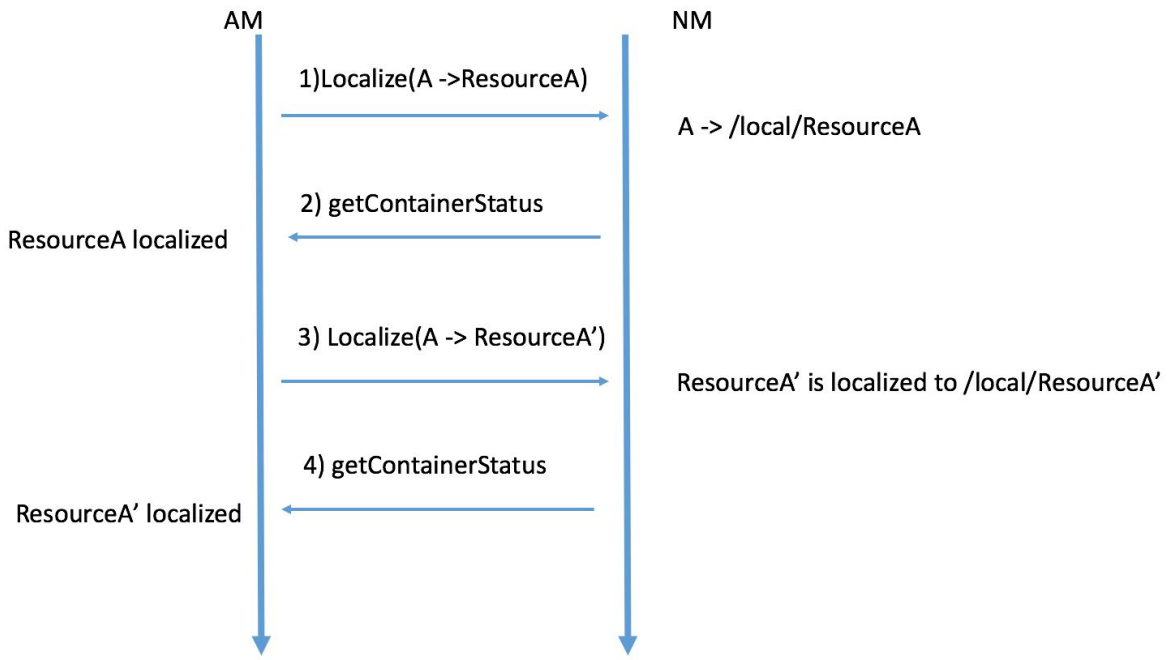
1) AM can request new resources for a container as it runs.
2) NM downloads the new set of resources.
3) NM creates the symlink file which points to the newly downloaded resources. (This is an existing mechanism for containers to access the localized resources from within their current working-directory even though NM may have downloaded them into a completely different filesystem hierarchy)
    a) If it's a new symlink, NM just creates the new symlink file.
    b) If it's an existing symlink, the container may be still using the resource. Blindly updating symlink will cause issues. This is mainly for the use case of container upgrade - container may need to restart with a newer version of resources, and we want to download the resources before triggering restart. We may update the symlink to point to the newer version of resources as part of the container

launching process. How to implement this partially falls into the effort of container upgrade feature. As part of the work for continuous resource-localization, we'll skip updating the symlink for now.

4) The implementation will largely flow with the existing resource-localization code to bring minimal disruptions.

# 4) Workflow

Below diagram describes the continuous localization workflow.




1) AM is running, it asks to localize new ResourceA with symlink as A. NM, when receiving this call, will try to download the ResourceA into "/local" directory, and create symlink A in the container CWD to point to the ResourceA.
2) AM can use *getContainerStatus* API to query whether the previous asked resources are localized properly
    a) My original thought is to have a separate *getLocalizationStatus* API to query resource statuses instead of reusing the getContainerStatus API. Given that today the resources are still tying to the containers, I chose to include it as part of ContainerStatus for simplicity. We can open this up later if need be.
3) AM again asks to localize new resource A' with the same symlink A. NM downloads resource A' and postpone updating the symlink to point to new ResourceA', because the container may still be using the old ResourceA.
4) AM gets informed that the new ResourceA' is localized.

5) During this process, AM itself needs to track how many resources it has asked and how many resources are localized/pending.

# 5) API Changes

## 5.1) New API in ContainerManagementProtocol

```
localize(ResourceLocalizationRequestProto) returns (ResourceLocalizationResponseProto);

message ResourceLocalizationRequestProto {
  optional ContainerIdProto container_id = 1;
  repeated StringLocalResourceMapProto local_resources = 2;
}

message ResourceLocalizationResponseProto {
}
```

## 5.2) Change ContainerStatus to include resource localization status

```
message ContainerStatusProto {
   optional ContainerIdProto container_id = 1;
   optional ContainerStateProto state = 2;
   optional string diagnostics = 3 [default = "N/A"];
   optional int32 exit_status = 4 [default = -1000];
   optional ResourceProto capability = 5;
   optional ExecutionTypeProto executionType = 6 [default = GUARANTEED];
+ repeated LocalizationStatusProto localization_status = 7;
}

message LocalizationStatusProto {
  optional LocalResourceProto resource = 1;
  optional LocalizationStateProto state = 2;
}

enum LocalizationStateProto {
   PENDING = 0;
   COMPLETED = 1;
```

```
   FAILED = 2;
}
```