Replicate Functions

There are multiple ways of creating a function for ex:

```
create function toupper as 'com.anishek.MyUpper' using jar
'hdfs://localhost:9000/user/anishek/myudf.jar';
```

-- OR --

```
add jar 'hdfs://localhost:9000/user/anishek/myudf.jar'; create function
another_toupper as 'com.anishek.MyUpper';
```

Problem Statement:
Replicate permanent functions to destination cluster. We will not Support Temporary Functions.

Solution Proposal:
Approach 1:
The solution has to take care of bootstrap and incremental replication of the following two components of the create function spec.

1. Metadata defining the function :
    a. Events leading to change in metadata of functions is restricted to CREATE and DROP. If we capture these events in the notification log we should be able to recreate the definition on the destination cluster
2. Actual implementation of the function definition:
    a. The actual implementation for a function resides in a class / jar file situated either on
        - ~~local HS2 instance~~ : result of 1 & 2
        - HDFS
        - On web as specified via IVY file
    b. The implementation can change if a new binary is updated on the source, which can be done outside the scope of hive system. Hence we have to track these changes via a change management strategy similar to data file changes for tables. This will lead to storing the checksum of the implementation as part of the function definition in hive. This is only valid for jars on ~~**local HS2 instance**~~ or **HDFS**.
    Generally should not affect IVY mode, except in cases where the maven repository is internally maintained enterprise repo and someone publishes a new artifact with same version, specifically true for SNAPSHOTS?

    c. The hive system can know about these changes
        - When hive is Restarted
        - When a `reload functions` statement is executed in a session.

Capturing the metadata definition of the function definition is easier to capture and is already being done.This mostly includes name of the function, reference to the underlying class as a FQN as implementation.
For the actual jar defined as here, we should at the time of function creation, create a copy of the jar and store it under a filesystem managed by replication component. This is important for two reasons:

- The Jar modification is external to the hive system and we wont have a hook to apply *copy on write* strategy as we do for tables.
- These jars are binaries and it will not be a huge overhead to store a copy of it at function creation time.

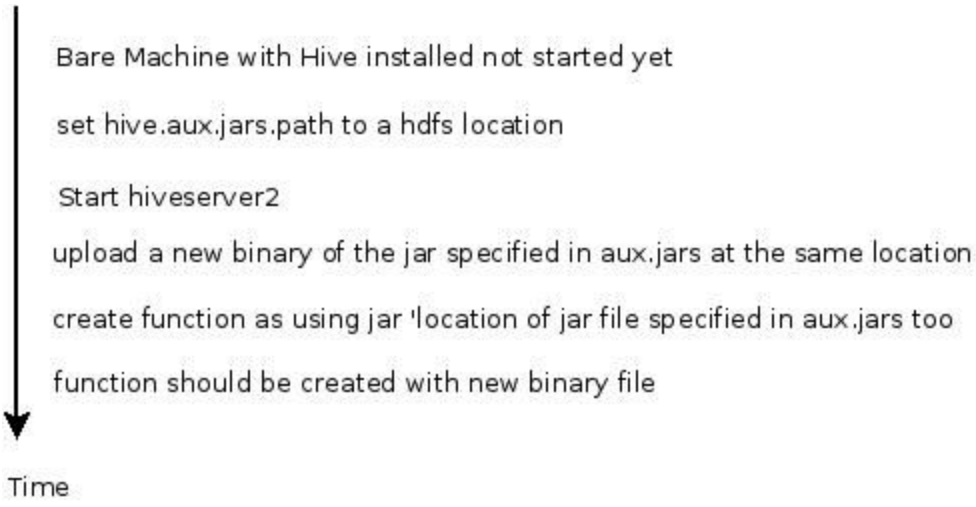
We have to make sure though that the FS managing the copy of the jar incorporates the event id semantics when storing it as multiple events as described here will affect what binary to use. This will lead to change in the notification log structure for create function since the resource URI's in the Function Object as represented in the metastore will be the user given URI (including schemes like hdfs/ivy), this however is something we have to definitely change for 'hdfs' scheme to instead use the change management (cmfs) uri scheme for notification log for replication. This approach will work for incremental repl, I have to understand how the `repl dump` command works for bootstrap to make sure it will not pose additional challenges.

Questions:

1. The implementation of a function can be provided via two modes:
    - Mode 1 : `add jar 'location'`
    - Mode 2 : `create function as 'some.class' using jar 'location'`

    We should be able satisfy (*2.b)* in *Solution Proposal* for either modes. Supporting Mode 1 would require capturing metadata for Mode 1 in metastore. This will need more analysis of various cases relating to (udf implementations vs serde implementations) and session scope of follow up events since they can be used to create temporary functions / tables. Leaning towards the use Mode 2 only, for creating permanent functions (the function spec be changed or we support only mode 2 for replication) ?
    Decision : Use Mode 2 of operation

2. How does hive perform in standalone non replication setup for the steps below:



~~Do we need to look at ordering of resources in class loaders for create function such that the new implementation is picked?~~
-- OR --
~~Do we use the current mode where since the resource was already added it will just load the older implementation of the function ? (have to verify this)~~
-- OR --
To get the new implementation, use `reload functions` after create function ?

Decision: We will go with approach 3 above to use “reload functions” as one of the hooks to detect change in the udf binary for replication.

3. Local FS: we don’t allow creating permanent functions using the command below:

```
create function tolow as 'com.anishek.MyUpper' using file
'file:///Users/anagarwal/projects/udf/target/udf-1.0-SNAPSHOT.jar';
```

however we can do the following:

```
add jar /Users/anagarwal/projects/udf/target/udf-1.0-SNAPSHOT.jar';
create function tolow as 'com.anishek.MyUpper';
```

What is the non-local mode and why one is allowed and not another ?
[Sushanth]The reason the second case works is because we're cheating. We tell hive not to worry, that the jar would be available at runtime, and thus, since we don't specify the resource uris, it doesn't/can't check. This permanent function is not guaranteed to work in other sessions or for other users unless they do an ADD JAR every time. However, with the first usage format, if the jar was present in hdfs, we guarantee the function's reusability/permanency.

Again, I see this as a good indication of the design route we should take with this - and that is to ignore ADD JAR'ed resources, and treat only the permanent resources as being relevant. Anything ADD JARed is the user's responsibility to make available on destination as well.
[Anishek] Agreed though on the functionality side i think we should do similar checks for permanent functions when created using route 2 since all it does is check the filesystem scheme "file" on the provided resource URI, we have to at udf load time lookup the resource used to load the class + if that resource is present the hive.downloaded.resources.dir as specified in hiveconf then we allow or reject. This is not very important from hive replication point of view.

4. Where do we download the resource on the target cluster? Define a `repl.function.resources.dir` on the target cluster as the root directory with the following locations for different resource types:
    a. HDFS : ${ repl.function.resources.dir}/HDFS
    b. IVY : ${ repl.function.resources.dir}/IVY
    This will however lead to departure of configuration from the ivysettings.xml file that is used to dowload ivy resources. Hence we might want to bank on the ivysettings.xml file in target cluster to define where to put the replicated jars for this type on the target cluster ?

5. How are serde jar's for storage formats going to be supported ? -- result of decision , This is not going to be a supported case for hive replication, this has to be done explicitly by the customer to maintain sanctity of the table / data.

Approach 2:

Function replication will be supported when their implementations reside on hdfs or ivy uri's. We have no change management over these scheme's as of now. Any changes in these external UDF jars hosted on these scheme(hdfs/ivy) are loaded either when, hive system restarts or `reload functions` command is executed.

Since hive replication aims to provide DR(Disaster Recovery) capability, this will not be possible if binaries of functions on primary warehouse are not copied over to the replica warehouse for schemes other than *ivy*. To achieve this when users create functions on primary warehouse using jars on HDFS, hive replication will copy the corresponding jar to the HDFS filesystem on the replica warehouse. The SQL commands in hive which will trigger the above are create / drop function commands only. The replication sub system will not plug into reload Functions or hive restarts to find out the binary differences in jars.

IVY: these systems typically mandate a new revision of the binary, if there are any changes in the binary, which will lead to redefining the function.
HDFS: there is no way for hive to manage or know about the change of binaries here externally so based on what hive context can infer from we will only use CREATE / DROP FUNCTION commands to achieve this.

This will also allow us to honor the specific configurations for the schemes as defined at the the replicated warehouse, for ex: ivysettings.xml or `hive.downloaded.resource.dir` configuration, in effect at the replicated warehouse might be different that the primary wwarehouse and the above mode of replication will allow us to use the existing warehouse settings for downloading the jars in appropriate locations etc.
It will also make understanding the replication process for functions easy for users, as well as debugging for developers.

Decision: We have decided to keep the implementation simple such that we will go with approach 2.
Summarizing the discussion:

- Support `create function` only with `using jar/file/archive` clause for replication
- Support only addition of jars from `hdfs/ivy` scheme
- No support for `add jar/file/archive` in replication
- We will move all function definitions irrespective of how they were created on the primary warehouse to the replica warehouse, functions which depend on jars on the hive server instance have to be set up as part of the administrator setup of the replica environment and the replication subsystem will not take care of this.
- For functions other than file:// scheme , we use the CREATE / DROP function commands to capture the change in jar binaries, hence the correct sequence of steps to update a function definition are:
    - Add jar to hdfs location = > hdfs://loc1
    - CREATE function t1 … using 'hdfs://loc1';
    - Drop function t1;
    - Remove + add new jar to hdfs hdfs://loc1
    - CREATE function t1 … using 'hdfs://loc1';

Relevant Technical details:

- Since functions are associated with databases, they will be exported to the root folder denoted by `[dump_location]/[db_name]/_functions/[function_name]`. The function definition will be stored in _metadata file under the root directory effectively @ `[dump_location]/[db_name]/_functions/[function_name]/_metadata`
- Since function names are unique globally, repl load will fail on destination warehouse, if a function on destination warehouse is already created even in a different database.
- `USING` will check for the correct scheme of the url's provided hence no additional check for the scheme will be done in replication code flow.