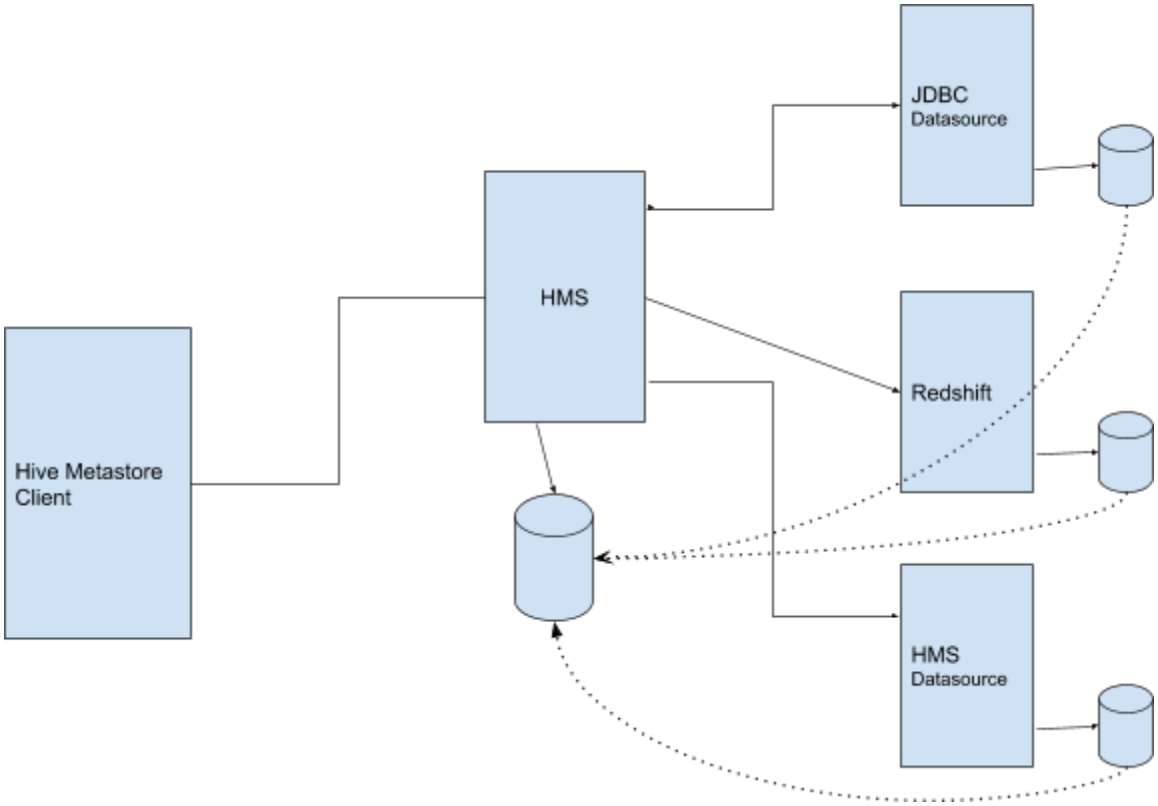
Feature Summary:

Support the ability to map a catalog or database that resides in an external datasource into a local Hive Metastore. These external data sources can be of different types, like MySQL, Postgres, Redshift etc or even other Hive Metastore instances. Metastore currently has support for external tables using a JDBCStorageHandler but the mapping needs to be configured for each table. To be able to map a remote data source with large volumes of tables, this can be very painful.

So this feature work is to be able to support configuring a mapping at a catalog level or a database level. When an external catalog is mapped to hive metastore, all its databases and their constituent tables are automatically visible from the hive metastore. Similarly, when an external database is mapped, all its constituent tables are visible from the hive metastore. When a Hive Metastore client requests a table, that resides in external datasource, hive metastore will fetch the table's definition from the external data source and returns a [JDBCStorageHandler-based] Table object just like it would return a native hive table.

Currently, Hive metastore support for Catalogs is very limited in functionality and not user-facing. Enriching this support is a lot of work in terms of API changes and also client side changes. So, while we do want to add support for mapping Catalogs in the future, our immediate goal is to support external databases.


JDBC
Datasource
HMS
Redshift
Hive Metastore
Client
HMS
Datasource

Named DataConnectors:

A dataconnector is a top level HMS object that contains definition/configuration of a remote datasource. This object contains all the necessary properties that can be used to connect to this datasource, like hostname, user credentials, type etc).

This object isolates the connection related properties from the Database and provides a better user interface. It

- Can be associated with multiple databases without having to re-define them for each mapped remote database.
- Provides a single source for any configuration changes. For example, user enables SSL for the JDBC transport. Just the dataconnector needs to be altered, no changes on mapped databases.
- Associated with database via dataconnector name rather than ID. This eliminates the need to alter the mapped database(s) when the transport related configuration is changed, like changes to host name of the DB server or change the dataconnector type altogether. For example, move from postgres to MySQL on the backend.
- DataConnectors can be dropped and re-created without the need to alter mapped databases.
- Better runtime performance and resource management as this enables hive to a connectionpools for the dataconnectors rather than for individual databases, which could be many more in volume than dataconnectors.
- This model fits well for the future work as well when we add support for remote catalogs. Catalogs can be defined using these dataconnector definitions as well.

HiveQL changes:

```
create connector mysql_ds1
     type MYSQL
     url <url>
     [COMMENT <description>]
     username <username>
     ( (password <password>) ||
      (keystore <keystore> && passwordkey <key>) )
     WITH DCPROPERTIES (
    "hive.sql.dbcp.username" = "hive"
    "hive.sql.dbcp.password" = "secret"
    "hive.sql.dbcp.maxActive" = "1"

SSL Support (Post-V1)
    "dataconnector.ssl.enabled" = "true"
    "dataconnector.ssl.trustStore" = "ds1.jks"
```

```
    "dataconnector.ssl.trustStorePassword" = "password"
    "dataconnector.ssl.keyStore" = "ds1_ks.jks"
    "dataconnector.ssl.keyStorePassword" = "welcome1"


CREATE [REMOTE] DATABASE mysql_db
    COMMENT <comment>
    USING <connectorName>
    WITH DBPROPERTIES
    (dataconnector.dbname=<remote db name>,
    dataconnector.tables.exclude = "table1,table2,table10")
```

IMetastoreClient API Modifications

createDatabase() → should use Request object
dropDatabase() → shallow drop of metadata from Hive Metastore for REMOTE DBs, no table deletions.
alterDatabase() // edit named dataconnector.

IMetastoreClient API Additions:
createDataConnector()
listDataConnector()
getDataConnector()
dropDataConnector()
alterDataConnector()

Thrift Changes:

```
struct DataConnector {
1: string name
2: string type
3: string url
4: optional string description
5: optional map<string,string> parameters
6: optional string ownerName
7: optional PrincipalType ownerType,
8: optional i32 createTime
}

// namespace for tables
struct Database {
 1: string name,
 2: string description,
 3: string locationUri,
 4: map<string, string> parameters, // properties associated with the database
 5: optional PrincipalPrivilegeSet privileges,
```

```
 6: optional string ownerName,
 7: optional PrincipalType ownerType,
 8: optional string catalogName,
 9: optional i32 createTime          // creation time of database in seconds since
epoch
 10: optional string managedLocationUri // directory for managed tables
11: optional string type                // enum of fixed types NATIVE || REMOTE
12: optional string dataConnectorName
13: optional string remoteDatabaseName
}

struct GetDatabaseRequest {
1: optional string name,
2: optional string catalogName,
3: optional list<string> processorCapabilities,
4: optional string processorIdentifier
}
```

Schema changes:

New tables:

```
DATACONNECTORS
(
        "DC_NAME" VARCHAR(128) NOT NULL, // PRIMARY KEY
        "TYPE" VARCHAR(32) NOT NULL,
        “URI” VARCHAR(4000) NOT NULL,
        "DESCRIPTION" VARCHAR(256),
        “OWNER_NAME” VARCHAR(128),
        “OWNER_TYPE” VARCHAR(10),
        "CREATE_TIME" INTEGER
)

"DATACONNECTOR_PARAMS"
(
        "DC_NAME" VARCHAR(128) NOT NULL,  // FOREIGN KEY
        "PARAM_KEY" VARCHAR(180) NOT NULL,
        "PARAM_VALUE" VARCHAR(4000)
);
```

Modifications to existing tables:

```
CREATE TABLE "APP"."DBS" (
 "DB_ID" BIGINT NOT NULL,
 "DESC" VARCHAR(4000),
 "DB_LOCATION_URI" VARCHAR(4000) NOT NULL,
 "NAME" VARCHAR(128),
 "OWNER_NAME" VARCHAR(128),
 "OWNER_TYPE" VARCHAR(10),
 "CTLG_NAME" VARCHAR(256) NOT NULL DEFAULT 'hive',
 "CREATE_TIME" INTEGER,
 "DB_MANAGED_LOCATION_URI" VARCHAR(4000),
```

```
 "TYPE" VARCHAR(128) DEFAULT "NATIVE",
 "DATACONNECTOR_NAME" VARCHAR(128),
 “REMOTE_DBNAME” VARCHAR(128)
);
```

Security:

- Filter output from “show dataconnectors”
- All authenticated users to be allowed to create a new data connector.
- “describe dataconnector” will require atz policies.

New Resource type in Ranger called DataConnector. Access to DataConnector objects will require Atz provider policies as they contain sensitive data.

HiveMetaStore changes:

Add new APIs
create_dataconnector()
drop_dataconnector()
alter_dataconnector()
list_dataconnectors()
get_dataconnector()

Modifications to existing APIs/implementations
create_database(CreateDatabaseRequest)
Support additional properties for associating with a dataconnector
drop_database() → shallow drop of the database metadata in HMS for non-NATIVE DBs

`DataConnectorProviderFactory`: Takes a DataConnector object and has factory methods to return a DataConnectorProvider

- *`DataConnectorProvider getInstance(DataConnector)`*
  This method returns an instance of DataConnectorProvider based on the type of the dataconnector.
- *`shutdown()`*
  This method calls shutdown on each of the providers created that in turn can close any resources aka connectionpools etc. Ideally, this is called from a service shutdown

Uses an internal map to cache instantiated providers.

`DataConnectorProvider` interface:
(add all *Table* APIs to database)

- void open()
- void close()

- List<Table> getTables(String dbname, String regex)
- List<String> getTableNames(String dbname)
- Table getTable(String dbname, String tableName)
-
- createTable(String dbname, String tableName ...)
- dropTable(String dbname, String tableName ...)

- alterTable(String dbname, String tableName,)

Providers:

```
HiveDataConnectorProvider
MSSqlDataConnectorProvider*
PostgresDataConnectorProvider*
MySqlDataConnectorProvider*
OracleDataConnectorProvider*
RedshiftDataConnectorProvider
```

An Abstract class of the DataProvider that has utils methods for converting to hive object class and vice versa and a sub-class implementation in a JDBCDataSourceProvider that can serve as a super class for any specific provider implementation.

Tasks:

1. Add DataConnector support
    a. Define built-in types (SQLServer, postgres, redshift etc) and make new types.
        i. Check if the specified type could point to the driver directly. Drivers are packed in CDH.

    b. HMS schema changes, initialize scripts and thrift changes.
    c. Create/Alter/Show/Drop DataConnector support. Create a database with property support (create database property (dataconnector="mysql1")).
    d. Maybe I need to modify the table property? Adding support point to ds without removing current features. (lower priority)
    e. Add Ranger support for dataconnectors. New top level Ranger resource. Only owner and users of admin groups are allowed to modify existing dataconnectors. Default Ranger policy to filter out certain dataconnectors based on ownership and group membership.
    f. Alter table should deny any metadata changes to the tables from the external tables.
    g. getTable*() APIs should generate JDBCStorageHandler-based table metadata.
    h. DDL support to change "`type`" for databases should not be supported.
    i. Alter database DDL queries for REMOTE-type DBs have limited scope. For example, location changes should be denied.
2. Defining DataConnector interfaces with base JDBC implementations and SPIs for plugins
    a. SQL Server (top)
    b. Postgres
    c. Hive (integrate Jesus' patch with these changes)
    d.
3. Read is already supported

4. Write: We are not targeting support for write operations on these mapped tables. There are plans to support this for future release. There is an upstream jira (HIVE-22392) tracking this work.
5. Backward compatibility.
6. Unit Tests
7. create database foo with DBPROPERTIES (“dataconnector.name”=”mysql1”, dataconnector.exclude.tables=”tabl1,tabl2”);

Pending items

- Should REMOTE databases be eligible for replication?
    - If the can and should be replicated, then dataconnectors should also be replicated right? How do we accomplish this?
    - If they should not be replicated, how should we prevent this? What properties should be scrubbed off these databases to do so?

- Should DataConnector APIs require any special capabilities or a need for transformation?
- Determine the impact on impala especially with the new Database objects.
- Determine the impact on spark.

Potential future work

- Add new APIs to enrich Catalog support in Hive metastore, ability to create/alter/drop new catalogs as well as having catalog names in all existing read/write/search APIs.
- Add catalog-related HiveQL grammar support to create/drop/alter catalogs.
- Enhance existing HiveQL grammar for being able to include a catalog name in the table name specification.
- Add support for queries like "*show databases in <catalog>*"