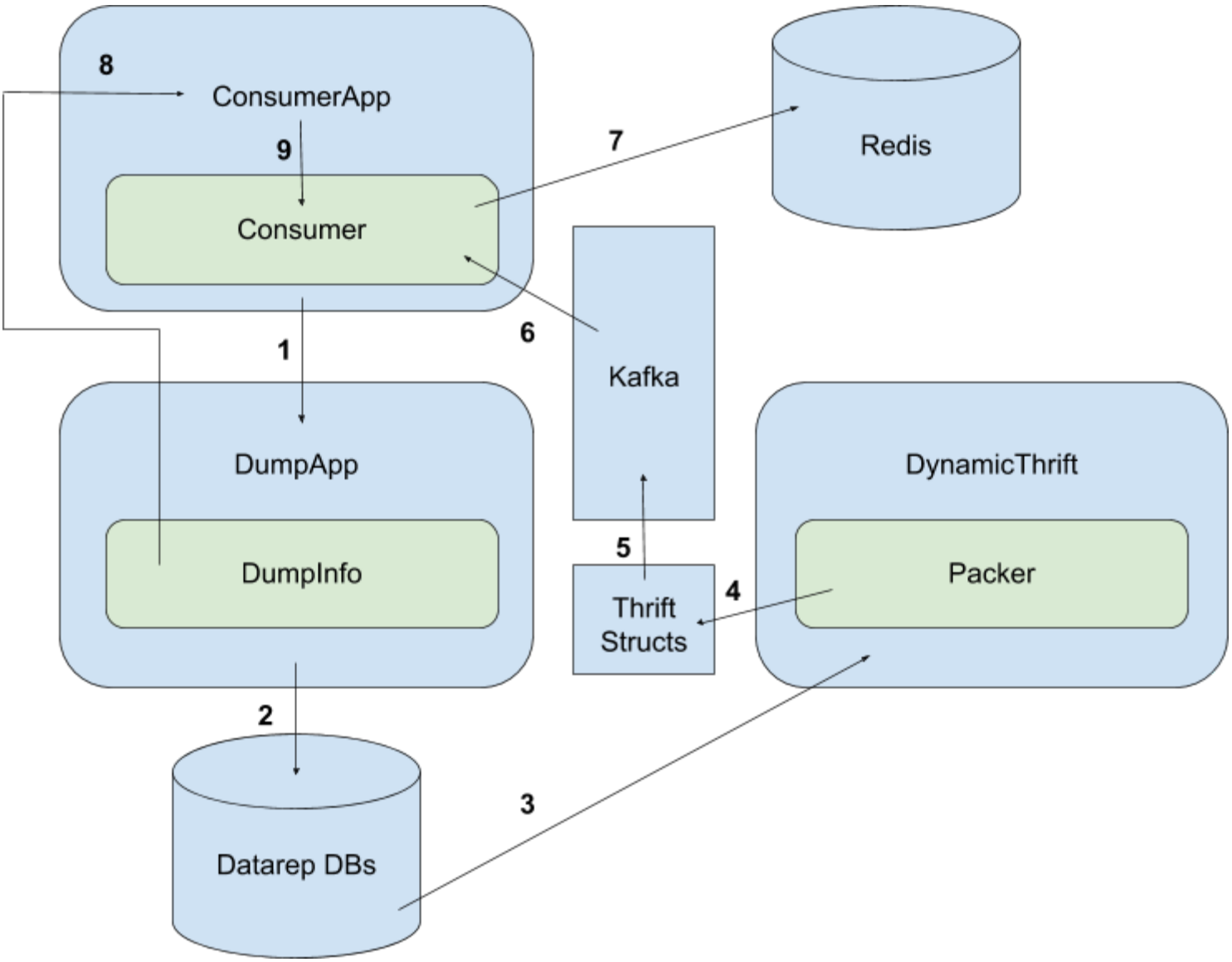
DList, DumpApp, Dynamic Thrift

git.rsglab.com/bware/dlist-dumpapp-dthrift



Background

This document covers all three applications because their use cases are tightly coupled, in the end however they will sit as stand alone services leveraging one another's capabilities.

Right now, Data Systems is working closely with Data Science to build useful applications leveraging the pipeline so that real-time events can lead to real-time predictions. An example need the Data Science team had was data mappings that could be queried in a cacheable manner (ie emailAddr → listIDs). In order to do this, an application would also be needed to update the history of this data source with the datarep dbs current state. Finally, in order to generically do this with all Thrift struct objects, a library was needed to generate Thrift objects given row data from mysql tables. Thus, the 3 projects of DList, DumpApp, and Dynamic Thrift were born.

DList is a consumer application that reads off the Prologue Third Stage kafka topics. The application stores the mappings of UserID → ListIDs into a Redis instance, taking care of the struct dependencies (ie ListEmail → Email).

DumpApp performs as it sounds: a standalone application that grabs all of the structs from the datarep mysql tables and produces them to a kafka topic for DList to read, as well as the original topic offsets. DumpApp depends on a topic to produce its data and offsets to as well as a list of Struct objects to know what to tables to clone from the datarep mysql tables.

DynamicThrift is a library leveraged by the DumpApp to produce the proper Thrift objects based on the mysql table data. Ported over from Prologue Third Stage into Java.

Goals

DList

1. Store UserID → ListIDs mappings into Redis
2. Update kafka topic offsets given by DumpApp
3. Run many consumers at once all reading from the thrift_uid_mailchimp topics

DumpApp

1. Successfully connect to the datarep dbs through a thread pool
2. Query the dbs without failure and through a server-side cursor and properly generate the right thrift structs
3. Store the third stage topic offsets before dumping the datarep dbs and send all the data through the ephemeral kafka topic

DynamicThrift

1. Generic a thrift struct given a row of mysql data
2. Be reusable by different projects within Data Systems

Design

DList

The design of DList is a consumer meant to constantly read from the PTS kafka topics and store Email → ListIDs mappings in Redis. DList can be configured through a properties file and is run through the ConsumerApp class. ConsumerApp leverages Consumer, Producer, DLConfig, and Topic to drive the logic for reading off kafka and storing mappings in Redis. Most of the logic sits in the Consumer class, where the reactive streams are created and thrift structs are read off the stream and properly stored. DList is meant to be an example consumer application in case others want to create similar mappings for easier querying later.

DumpApp

DumpApp was designed to copy all of the datarep dbs no matter the struct type passed as an argument. DumpApp has to be run by another application, and is driven by the DList app in my project. However it was generically designed to only be given a list of struct strings and a topic and will go grab all the datarep information and also store the kafka offsets for the consumer application. DumpApp was designed to be a generic tool that gives back metadata about the dump's success, topic offsets, and the kafka topics it used to push the data for the consumer to read from.

DynamicThrift

DynamicThrift was also designed to be leveraged by a consumer application. DynamicThrift simply provides an API to be called given certain params that returns a Thrift object packed with the correct data. It pairs closely with the DumpApp as after the mysql row data has been retrieved it then leverages DynamicThrift to correctly pack that information into a struct. DynamicThrift is meant to be a Java library for use of many applications to come.

Assign

All 3 applications have currently been written by the intern, Bradford Ware. After he leaves, the Data Systems team will pick up ownership of the repository and determine any other necessary features that would be useful.

Bradford Ware - bware@rsglab.com
Sean Sawyer - ssawyer@rsglab.com
Coty Rosenblath - crosenblath@rsglab.com

Schedule

A prototype application has been built by Bradford Ware that will be finished by May 11th. After that tests will be run to see if it should be converted into a job running in production. Needs will mostly be determined by the Data Science and Data Systems teams.