# Tweetypie

## Overview

Tweetypie is the core Tweet service that handles the reading and writing of Tweet data. It is called by the Twitter clients (through GraphQL), as well as various internal Twitter services, to fetch, create, delete, and edit Tweets. Tweetypie calls several backends to hydrate Tweet related data to return to callers.

## How It Works

The next sections describe the layers involved in the read and create paths for Tweets.

### Read Path

In the read path, Tweetypie fetches the Tweet data from [Manhattan](https://blog.twitter.com/engineering/en\_us/a/2014/manhattan-our-real-time-multi-tenant-distributed-database-for-twitter-scale) or [Twemcache](https://blog.twitter.com/engineering/en\_us/a/2012/caching-with-twemcache), and hydrates data about the Tweet from various other backend services.

#### Relevant Packages

- [backends](src/main/scala/com/twitter/tweetypie/backends/): A "backend" is a wrapper around a thrift service that Tweetypie calls. For example [Talon.scala](src/main/scala/com/twitter/tweetypie/backends/Talon.scala) is the backend for Talon, the URL shortener.

- [repository](src/main/scala/com/twitter/tweetypie/repository/): A "repository" wraps a backend and provides a structured interface for retrieving data from the backend. [UrlRepository.scala](src/main/scala/com/twitter/tweetypie/repository/UrlRepository.scala) is the repository for the Talon backend.

- [hydrator](src/main/scala/com/twitter/tweetypie/hydrator/): Tweetypie doesn't store all the data associated with Tweets. For example, it doesn't store User objects, but it stores screennames in the Tweet text (as mentions). It stores media IDs, but it doesn't store the media metadata. Hydrators take the raw Tweet data from Manhattan or Cache and return it with some additional information, along with hydration metadata that says whether the hydration took place. This information is usually fetched using a repository. For example, during the hydration process, the [UrlEntityHydrator](src/main/scala/com/twitter/tweetypie/hydrator/UrlEntityHydrator.scala) calls Talon using the [UrlRepository](src/main/scala/com/twitter/tweetypie/repository/UrlRepository.scala) and fetches the expanded URLs for the t.co links in the Tweet.

- [handler](src/main/scala/com/twitter/tweetypie/handler/): A handler is a function that handles requests to one of the Tweetypie endpoints. The [GetTweetsHandler](src/main/scala/com/twitter/tweetypie/handler/GetTweetsHandler.scala) handles requests to `get\_tweets`, one of the endpoints used to fetch Tweets.

#### Through the Read Path

At a high level, the path a `get\_tweets` request takes is as follows.

- The request is handled by [GetTweetsHandler](src/main/scala/com/twitter/tweetypie/handler/GetTweetsHandler.scala).

- GetTweetsHandler uses the TweetResultRepository (defined in [LogicalRepositories.scala](src/main/scala/com/twitter/tweetypie/config/LogicalRepositories#L301)). The TweetResultRepository has at its core a [ManhattanTweetRespository](src/main/scala/com/twitter/tweetypie/repository/ManhattanTweetRepository.scala) (that fetches the Tweet data from Manhattan), wrapped in a [CachingTweetRepository](src/main/scala/com/twitter/tweetypie/repository/ManhattanTweetRepository.scala) (that applies caching using Twemcache). Finally, the caching repository is wrapped in a hydration layer (provided by [TweetHydration.hydrateRepo](src/main/scala/com/twitter/tweetypie/hydrator/TweetHydration.scala#L789)). Essentially, the TweetResultRepository fetches the Tweet data from cache or Manhattan, and passes it through the hydration pipeline.

- The hydration pipeline is described in [TweetHydration.scala](src/main/scala/com/twitter/tweetypie/hydrator/TweetHydration.scala), where all the hydrators are combined together.

### Write Path

The write path follows different patterns to the read path, but reuses some of the code.

#### Relevant Packages

- [store](src/main/scala/com/twitter/tweetypie/store/): The store package includes the code for updating backends on write, and the coordination code for describing which backends need to be updated for which endpoints. There are two types of file in this package: stores and store modules. Files that end in Store are stores and define the logic for updating a backend, for example [ManhattanTweetStore](src/main/scala/com/twitter/tweetypie/store/ManhattanTweetStore.scala) writes Tweets to Manhattan. Most of the files that don't end in Store are store modules and define the logic for handling a write endpoint, and describe which stores are called, for example [InsertTweet](src/main/scala/com/twitter/tweetypie/store/InsertTweet.scala) which handles the `post\_tweet` endpoint. Modules define which stores they call, and stores define which modules they handle.

#### Through the Write Path

The path a `post\_tweet` request takes is as follows.

- The request is handled in [PostTweet.scala](src/main/scala/com/twitter/tweetypie/handler/PostTweet.scala#L338).

- [TweetBuilder](src/main/scala/com/twitter/tweetypie/handler/TweetBuilder.scala) creates a Tweet from the request, after performing text processing, validation, URL shortening, media processing, checking for duplicates etc.

- [WritePathHydration.hydrateInsertTweet](src/main/scala/com/twitter/tweetypie/config/WritePathHydration.scala#L54) passes the Tweet through the hydration pipeline to return the caller.

- The Tweet data is written to various stores as described in [InsertTweet.scala](src/main/scala/com/twitter/tweetypie/store/InsertTweet.scala#L84).