**Twitter API**

# API v2

**Twitter APIv2** is the most recent endpoint of Twitter. The REST endpoint uses the GET method to return information about a Tweet or group of Tweets, specified by a [Tweet ID](https://developer.twitter.com/en/docs/twitter-ids). The response returns one or many [Tweet objects](https://developer.twitter.com/en/docs/twitter-api/data-dictionary/object-model/tweet), which deliver fields such as the Tweet text, author, media attachments, and more. To access this endpoint, you must have an approved [developer account](https://developer.twitter.com/en/docs/developer-portal/overview), and have [activated the new developer portal](https://developer.twitter.com/content/developer-twitter/en/portal/opt-in). When authenticating, you must use keys and tokens from a [developer App](https://developer.twitter.com/en/docs/apps) that is located within a [Project](https://developer.twitter.com/en/docs/projects).

The Twitter API v2 endpoints are equipped with a new set of parameters called **fields**, which allows you to select just the data that you want from each of our objects in your endpoint response. For example, if you only need to retrieve a Tweet’s created date, or a user’s bio description, you can specifically request that data to return with a set of other default fields without the full set of fields that associate with that data object. This provides a higher degree of customization by enabling you to only request the fields you require depending on your use case.

Each object has its own parameter which is used to specifically request the fields that are associated with that object. Here are the different fields parameters that are currently available:

* Tweet → tweet.fields
* User → user.fields
* Media → media.fields
* Poll → poll.fields
* Place → place.fields

With **expansions**, developers can expand objects referenced in the payload.

Objects available for expansion are referenced by ID. For example, the referenced\_tweets.id and author\_id fields returned in the [Tweets lookup](https://developer.twitter.com/en/docs/twitter-api/tweets/lookup/introduction.html) payload can be expanded into complete objects. If you would like to request fields related to the user that posted that Tweet, or the media, poll, or place that was included in that Tweet, you will need to pass the related expansion query parameter in your request to receive that data in your response.

Entity **annotations** are programmatically defined entities that are nested within the entities field and are reflected as annotations in the payload. The entity annotations can have the following types: Person, Place, Product, Organization and Other.

## I.I User

The **RESTful endpoint** uses the GET method to return information about a user or group of users, specified by a user ID or a username. The response includes one or many [user objects](https://developer.twitter.com/en/docs/twitter-api/data-dictionary/object-model/user), which deliver fields such as the Follower count, location, pinned Tweet ID, and profile bio. Responses can also optionally include expansions to return the full Tweet object for a user’s pinned Tweet, including the Tweet text, author, and other Tweet fields.

This endpoint is commonly used to receive up-to-date details on a user, to verify that a user exists, or to update your stored details following a compliance event.

To access this endpoint, you must have an approved [developer account](https://developer.twitter.com/en/portal/dashboard), and have [activated the new developer portal](https://developer.twitter.com/en/portal/opt-in).

## I.II Tweets

The Twitter API v2 recent **search endpoint** provides developers with API access to public Tweets posted over the last week. The endpoint, receiving a single search query and responding with matching Tweets. The recent search endpoint serves a wide variety of use cases; from providing fundamental research data, to near-real-time 'listening' applications, this endpoint can be used to collect Tweets related to topics of interest.

This RESTful endpoint can deliver to 100 Tweets per request in reverse-chronological order, and [pagination](https://developer.twitter.com/en/docs/twitter-api/tweets/search/integrate/paginate) tokens are provided for paging through large sets of matching Tweets. [Search queries](https://developer.twitter.com/en/docs/twitter-api/tweets/search/integrate/build-a-rule) are created with operators that match on Tweet and user attributes, such as message keywords, hashtags, and URLs. Operators and rule clauses can be combined into queries with boolean logic and parentheses to help refine the filter's matching behavior, which can add up to 512 characters long.

The **filtered stream endpoint** group enables developers to filter the real-time stream of public Tweets. This endpoint group’s functionality includes multiple endpoints that enable you to create and manage rules and apply those rules to filter a stream of real-time Tweets that will return matching public Tweets. This endpoint group allows users to listen for specific topics and events in real-time, monitor the conversation around competitions, understand how trends develop in real-time, and much more.

The **sampled stream endpoint** delivers a roughly 1% random sample of publicly available Tweets in real-time. With it, you can identify and track trends, monitor general sentiment, monitor global events, and much more.

# API v1.1

All Twitter APIs that return Tweets provide that data encoded using JavaScript Object Notation (JSON). JSON is based on key-value pairs, with named attributes and associated values.

There are a set of **Data Dictionaries** for these fundamental Twitter objects. Reflecting the JSON hierarchy above, here are links and further descriptions of these Objects:

* [Tweet](https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object) - Also referred to as a ‘Status’ object, has many ‘root-level’ attributes, parent of other objects.
  + [**User**](https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/user-object) - Twitter Account level metadata. Will include any available account-level enrichments, such as [Profile geo](http://support.gnip.com/enrichments/profile_geo.html).
  + [**Entities**](https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/entities-object) - Contains object arrays of #hashtags, @mentions, $symbols, URLs, and media.
  + [**Extended Entities**](https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/extended-entities-object) - Contains up to four native photos, or one video or animated GIF.
  + [**Places**](https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/geo-objects) - Parent to ‘coordinates’ object.

## II.I Tweets

The following API endpoints can be used to programmatically retrieve Tweets:

* GET statuses/show/:id
* GET statuses/oembed
* GET statuses/lookup

### II.I.I Geo

Tweets can be associated with a location, generating a Tweet that has been ‘geo-tagged.’ Tweet locations can be assigned by using the Twitter user-interface or when posting a Tweet using the API. Tweet locations can be an exact ‘point’ location, or a Twitter Place with a ‘bounding box’ that describes a larger area ranging from a venue to an entire region.

There are two ‘root-level’ JSON objects used to describe the location associated with a Tweet: coordinates and place. Places are specific, named locations with corresponding geo coordinates.

Tweet location operators:

* **place**: Filter for specific Places by their name or ID. To discover “Places” associated with a specific area, use Twitter’s reverse\_geocode endpoint in the REST API. Then use the Place IDs you find with the place: operator to track Tweets that include the specific Place being referenced. If you use the Place name rather than the numeric ID, ensure that you quote any names that include spaces or punctuation.
* **place\_contains**: Rather than matching for specific Places, the place\_contains: operator performs a substring match against Place names sent through from Twitter. This can accommodate concepts like place\_contains:”, CO” to match Places that contain names like “Boulder, CO” or “Denver, CO”.
* **place\_country**: Each Twitter “Place” comes with a country code, indicating the country in which the Place is located. The country\_code: operator allows you to filter on this ISO alpha-2 character code.
* **has:geo**: The has:geo operator matches for the presence of either Point or Place geo information within the Twitter payload. Note that this does not allow you to specify specific locations or types of geo data, it simply requires that results have Tweet-specific location information of some kind.
* **point\_radius**: The point\_radius: operator allows you to specify a circular geographic area and match Tweets containing Tweet-specific location data that fall within that area. To use, define a central lon-lat coordinate, and then set the radius (up to 25 miles). Any Tweet containing a geo Point that falls within this region will be matched. Additionally, Tweets containing Twitter Places will match where the geo polygon defined for the Place falls fully within the defined point-radius area. Places whose polygons fall outside the defined point-radius area to any extent will not match.

Usage resembles the following: point\_radius:[lon lat radius]

* **bounding\_box**: The bounding\_box: operator allows you to specify a 4-sided geographic area and match Tweets containing Tweet-specific location data that fall within that area. To use, define lon-lat coordinates that represent the opposite corners of the box, such that each side of the box is up to 25 miles in length. Any Tweet containing a geo Point that falls within this region will be matched. Additionally, Tweets containing Twitter Places will match where the geo polygon defined for the Place falls fully within the defined point-radius area. Places whose polygons fall outside the defined point-radius area to any extent will not match.

For our purposes I will suggest using **point\_radius** approach because we can cover the Apulia region in these three circles.



We cannot list for all the cities in the region using **place\_contains**, it would be very unpractical.

For retrieving posts **regarding** the Apulia region (outside the region), we can query based on hashtags. In this case, we can list all the main cities of the region and filter for all the hashtag containing these words. Here is a possible filter list: “Puglia”, “Bari”, “Barletta”, “Andria”, “Trani”, “BAT”, “Brindisi”, “Foggia”, “Taranto”, “Lecce”. We can also add the mayors of the main cities like “decaro” and so on. It is true that if a post regards Cisternino and does not contain the hashtag Brindisi, it would not be retrieved but I have to say that often in hashtags appear the region or the province, in the case in which the post is written outside the region.

For this purpose, combining the point\_radius research with the hashtag query, we have to define circles for the rest of the nation outside the region like in this way.



For the retrieving of data in circles and for hashtag filtering, Crowd Pulse can help us.

## II.II User

The following API endpoints can be used to programmatically follow users, search for users, and get user information:

* GET users/lookup
* GET users/search
* GET users/show