[**Data Store**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#top)

The simplest use case of Kinvey is storing and retrieving data to and from your cloud backend.

The basic unit of data is an *entity* and entities of the same kind are organized in *collections*. An entity is a set of key-value pairs which are stored in the backend in JSON format. Kinvey's libraries automatically translate your native objects to JSON.

Kinvey's data store provides simple CRUD operations on data, as well as powerful filtering and aggregation.

The Appdata API forms one of the core REST services provided by the Kinvey backend.

[**Entities**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#entities)

Entities are the basic unit of data, stored as data objects using a JSON representation in the backend. The full set of CRUD operations is available for manipulating entities.

[**Create**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#create)

Stores the JSON document passed in the request body at the specified path. The response will be HTTP 201 Created with a Location header containing the URL for future updates.

This request allows only a *single entity* to be uploaded. For bulk upload, see the CSV/JSON import feature on the Kinvey [console](https://console.kinvey.com/) (navigate to the collection, click Settings, then click Import Data).

Entities can be created using either a POST request or a PUT request that includes the entity \_id in the path.

In the POST case, if an \_id is specified in the request body, the entity is created with the supplied \_id. If an \_id is not specified, one is automatically generated.

POST /appdata/:appKey/:collectionName HTTP/1.1

Host: baas.kinvey.com

Authorization: [user credentials]

Content-Type: application/json

<JSON-Document-representing-Entity>

HTTP/1.1 201 OK

Content-Type: application/json

<JSON-Document-representing-Entity>

In the PUT case, if an entity with that \_id already exists in the collection, the entity is updated with the supplied JSON. If an entity with such \_id does not exist, a new entity is created with the \_id supplied in the path.

PUT /appdata/:appKey/:collectionName/:id HTTP/1.1

Host: baas.kinvey.com

Authorization: [user credentials]

Content-Type: application/json

<JSON-Document-representing-Entity>

HTTP/1.1 201 OK

Content-Type: application/json

<JSON-Document-representing-Entity>

[Required headers](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Requiredheaders)

* Authorization
* Content-Type: application/json

[**Retrieve**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Fetching)

A GET to /appdata/:appKey/:collectionName/:id lets an app retrieve a previously created entity.

GET /appdata/:appKey/:collectionName/:id HTTP/1.1

Host: baas.kinvey.com

Authorization: [user credentials]

HTTP/1.1 200 OK

Content-Type: application/json

<JSON-Document-representing-Entity>

If you omit the entity ID, you get all entities in the collection, or you can replace the entity ID with a [query expression](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#querying) to limit the results down to the entities that match your criteria.

It's worth noting that an empty array is returned when the query matches zero entities but the response code remains the same.

Results from a single GET query are limited to 10,000 entities. If your query produces more than that number of matching entities, only the first 10,000 will be returned.

[Required headers](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Requiredheaders1)

* Authorization

[**Update**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Saving)

A PUT to /appdata/:appKey/:collectionName/:id acts as *upsert*. If the request contains an existing entity ID, it updates the entity. The request creates a new entity if the ID is not recognized.

The JSON body needs to include all entity attributes—both the ones you are updating and the ones that will remain unchanged. The backend will store the entire JSON body exactly as passed in the body.

PUT /appdata/:appKey/:collectionName/:id HTTP/1.1

Host: baas.kinvey.com

Authorization: [user credentials]

Content-Type: application/json

<JSON-Document-representing-Entity>

HTTP/1.1 200 OK

Content-Type: application/json

<JSON-Document-representing-Entity>

[Required headers](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Requiredheaders2)

* Authorization
* Content-Type: application/json

[**Delete**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Deleting)

Deletes an explicitly identified entity located at the specified URI or multiple entities in the collection filtered using a query.

Never apply [query modifiers](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#modifiers) to DELETE queries. Modifiers are ignored with this query type, which may result in a data loss.

When using API version 1 or below, the delete operation returns a '204 No Content' response.

DELETE /appdata/:appKey/:collectionName/:id?query=... HTTP/1.1

Host: baas.kinvey.com

X-Kinvey-API-Version: 1

Authorization: [user credentials]

HTTP/1.1 204 No Content

Starting wth API version 2, a delete returns a '200 OK' with a count of the number of entities deleted in the response body.

DELETE /appdata/:appKey/:collectionName/?query=... HTTP/1.1

Host: baas.kinvey.com

X-Kinvey-API-Version: 2

Authorization: [user credentials]

HTTP/1.1 200 OK

Content-Type: application/json

{

"count": 5

}

[Required headers](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Requiredheaders3)

* Authorization

[**Deleting a collection**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#deletecollection)

To permanently delete an entire collection, send a POST request containing the name of the collection you wish to delete to /rpc/:appKey/remove-collection. To ensure that this destructive action is not performed accidentally, you must authenticate using the [master secret](https://devcenter.kinvey.com/rest/guides/security#credentials), as well as include the X-Kinvey-Delete-Entire-Collection header as part of the request.

POST /rpc/:appKey/remove-collection HTTP/1.1

Host: baas.kinvey.com

Content-Type: application/json

Authorization: [master credentials]

X-Kinvey-Delete-Entire-Collection: true

{

"collectionName": "your-collection-name"

}

If the collection was successfully deleted, Kinvey will respond with a '200 OK', with a body containing the number of collections deleted, which will always be 1.

HTTP/1.1 200 OK

Content-Type: application/json

{

"count": 1

}

[Required headers](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Requiredheaders4)

* Authorization
* X-Kinvey-Delete-Entire-Collection: true
* Content-Type: application/json

[**Metadata**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Metadata)

Every entity has a \_kmd (Kinvey Metadata) JSON object associated with it, which provides meta-information about an entity. The format of this object is:

1. "\_kmd": {
2. "lmt": "2013-03-14T15:55:00.329Z",
3. "ect": "2013-03-14T15:55:00.329Z"
4. }

The object exposes the following properties:

* lmt (last modified time) - holds the time the entity was last updated on the server.
* ect (entity creation time) - holds the time the entity was created on the server. This property will be added to an entity when it is created using POST or PUT, and will remain unchanged for the life of the entity.

[**Querying**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#querying)

Kinvey's REST API is backed by MongoDB. This gives you access to the power of MongoDB's [query language](http://www.mongodb.org/display/DOCS/Querying), exposed through the ?query={} URL parameter. Similarly, you have access to the power of MongoDB's [aggregation features](http://docs.mongodb.org/manual/core/single-purpose-aggregation/#group).

For example, an HTTP GET request to the following endpoint retrieves a subset of documents that match the criteria specified through [filters](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#operators) and [modifiers](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#modifiers).

1. /appdata/:appKey/:collectionName/?query=[filters]&[modifiers]

When sending queries to Kinvey's REST API, an app needs to [URLEncode](http://en.wikipedia.org/wiki/Percent-encoding#Percent-encoding_in_a_URI) the query strings, which most tools and libraries easily support. For readability, we don't do that in the examples below.

[**Filters**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#operators)

Filters allow an app to specify a range of MongoDB [query predicates](http://docs.mongodb.org/manual/reference/operator/query/) for filtering data stored in the collection. Queries respect Kinvey's [access control](https://devcenter.kinvey.com/rest/guides/security#authorization) settings as well, and will only retrieve entities to which the authenticated user has access.

[Query basics](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#querybasics)

To query for a single property with a specific value, use a JSON object of the form {"propertyName":"value"}:

1. ?query={"firstName":"James"}
2. ?query={"age":15}

To query for nested JSON sub-objects, use the dot notation. For example, if the entity contains the following JSON:

1. {
2. "\_id": "12345",
3. "author": {
4. "firstName": "Terry",
5. "lastName": "Pratchett"
6. }
7. }

It can be retrieved by the following query:

1. ?query={"author.firstName":"Terry"}

[Logical operators](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#logicaloperators)

You can filter by more than one condition using a single query. Separating your queries with a comma (,) will perform an *implicit* logical AND operation -- in other words, the query will return any entities that match ALL of the conditions. For example, the query below will return an entity with a first name of James and a last name of Bond, but will **not** return an entity with a first name of James and a last name of Smith.

1. ?query={"firstName":"James", "lastName":"Bond"}

You may also use *explicit* logical AND and OR operators. For example, the query above can also be written as:

1. ?query={"$and":[{"firstName":"James"}, {"lastName":"Bond"}]}

Similarly, to perform a logical OR, use the $or operator. The following query will return all entities that have a first name of James (*even if the last name is not Bond*), as well as all entities having a last name of Bond (*even if the first name is not James*):

1. ?query={"$or":[{"firstName":"James"}, {"lastName":"Bond"}]}

For more information on logical operators, see the [MongoDB reference documentation](http://docs.mongodb.org/manual/reference/operator/query/#logical).

[Comparison operators](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#comparisonoperators)

MongoDB supports several comparison operators, used to compare values in entities. For example, the following query will find any entities having an age that is greater than or equal to 31 by using the $gte operator:

1. ?query={"age":{"$gte": 31}}

Also available are $gt (greater than), $lt (less than) and $lte (less than or equal), among others. For a complete list of supported comparison operators, see the [MongoDB documentation](http://docs.mongodb.org/manual/reference/operator/query/#comparison).

[**Modifiers**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#modifiers)

Modifiers allow an app to change the results of GET requests. They are included as URL parameters to the request and have various effects on the result set of your queries. For example, an app can use modifiers to limit the size of the result set, implement paging, or extract a subset of entity properties into the response.

Don't use modifiers with DELETE queries.

[Supported modifiers](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#supportedmodifiers)

Kinvey supports the following modifiers:

**limit** - set a maximum number of results to return. Examples:

1. # find the first 5 entities in the collection
2. ?query={}&limit=5
3. # find only the first 10 entities with first name James
4. ?query={"firstName":"James"}&limit=10

**sort** - sort the result set by one or more properties. Examples:

1. # sort results by age (ascending)
2. ?query={}&sort=age
3. # or, equivalently
4. ?query={}&sort={"age": 1}
5. # sort results by age (descending)
6. ?query={}&sort={"age": -1}
7. # find all entities with the first name James, and sort results by
8. # first name (ascending), and then by last name (descending) -- "James Zoo" will
9. # come before "James Bond", but "Abraham Lincoln" will come before both
10. ?query={"firstName":"James"}&sort={"firstName": 1, "lastName": -1}

**fields** - only return a certain set of properties from matching entities. Examples:

1. # only retrieve the age and lastName fields of matching documents
2. ?query={}&fields=age,lastName
3. # only retrieve the ages of people named James
4. ?query={"firstName":"James"}&fields=age

Certain fields that are essential to each entity will *always* be returned, even if they are not included in a fields modifier. These include \_id, \_acl and \_kmd.

**skip** - skips the first *N* results that match the query. Example:

1. # find the second youngest person with the first name James
2. ?query={"firstName":"James"}&sort={"age":1}&limit=1&skip=1
3. # find everyone except for the 10 oldest people
4. ?query={}&sort={"age":-1}&skip=10

[Paging](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#paging)

Using a combination of the skip and limit operators is a common way of paging through result sets -- that is, returning small chunks of results instead of the entire set at once. For example, the following set of requests page a result, returning 20 entities at a time:

1. ?query={}&limit=20&skip=0
2. ?query={}&limit=20&skip=20
3. ?query={}&limit=20&skip=40

Kinvey imposes a limit of 10,000 entities on a single request to fetch data stored in the backend. If you specify limit() > 10,000 in the example above, the backend will silently limit the results to only the first 10,000 entities. For this reason, we strongly recommend [fetching your data in pages](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#paging).

[**Restrictions and limitations**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#queryrestrictions)

Kinvey enforces several restrictions on queries.

[Regular expressions](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Regularexpressions)

Regular expressions need to be anchored (prefixed with ^), and case sensitive. To do case insensitive search, create a normalized (i.e. all lowercase) field in your collection and perform the match on that field.

[Disallowed operators](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Disallowedoperators)

The $where and $query operators are not supported, and using them will result in an error.

[Result set limits](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Resultsetlimits)

* Any query submitted to the REST API will return a maximum of 10,000 results. If your query matches more than 10,000 entities, only the first 10,000 will be returned.
* A single query can only retrieve 100mb of data, and will return an error if this limit is exceeded. Retrieving a very large amount of data can place a heavy burden on an app, and we encourage you to reconsider your design if you are hitting this limit. If your app absolutely needs to retrieve more than 100mb of data, please consider using [paging](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#paging).

[**Counting**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#counting)

To count the number of objects in a collection, make a GET to \_count.

GET /appdata/:appKey/:collectionName/\_count HTTP/1.1

Host: baas.kinvey.com

Authorization: [user credentials]

HTTP/1.1 200 OK

Content-Type: application/json

{"count": <count-of-entities>}

This means that an app won't be able to save an entity with id of \_count. ids that start with underscore are a reserved namespace.

Note that an app will get a {"count": 0} if the collection doesn't exist.

[Required headers](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Requiredheaders5)

* Authorization

[**Aggregation**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#aggregation)

To perform a SQL-style GROUP BY aggregation, make a POST to \_group. The request body should contain a JSON object with four properties:

* key: an object that selects the field to group by
* initial: an object containing the initial structure of the document to be returned
* reduce: a JavaScript function that will be used to reduce down the result set
* condition: an optional filter applied to the result set *before* it is fed to the MapReduce operation

These are generally a little more complex than SQL GROUP BY, but an example goes a long way. Here, we groups the entities by lastName, applying the age filter, and return the count in each group.

POST /appdata/:appKey/:collectionName/\_group HTTP/1.1

Host: baas.kinvey.com

Authorization: [user credentials]

Content-Type: application/json

{

"key": {

"lastName":true

},

"initial": {

"count": 0

},

"reduce": "function(doc,out){ out.count++;}",

"condition": {

"age": { "$gt":31 }

}

}

[

{

"lastName": "Bond",

"count": 1

}

{

"lastName": "Jones",

"count": 3

}

]

[Required headers](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#Requiredheaders6)

* Authorization
* Content-Type: application/json

[**Location Querying**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#locationquerying)

See the [Location guide](https://devcenter.kinvey.com/rest/guides/location) for information on how to query data by location.

[**Delta Sync**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#DeltaSync)

The Kinvey REST API offers an endpoint that allows you to implement data differencing in your app. It works by returning the difference (data delta) between the local data state on the device and the data state on the backend. This saves bandwidth and shortens the request's response time, especially on slower networks.

The data delta contains a couple of arrays: one of entities that have been created or modified since the last time a specified query has been run and another listing the IDs of the entities that have been deleted. Using the delta and the current state of your local cache, you can reconstruct the latest server data state on the local device.

[**Configuring Delta Sync**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#ConfiguringDeltaSync)

The Delta Sync feature is configured per collection. The performance benefits of Delta Sync will be most noticeable on large collections that update infrequently. On the other hand, it may make sense to keep this feature turned off for small collections. This is because fetching the entire collection, if it's small, is expected to be faster than waiting for the server to calculate the delta and send it back.

Delta Sync is turned off by default for collections.

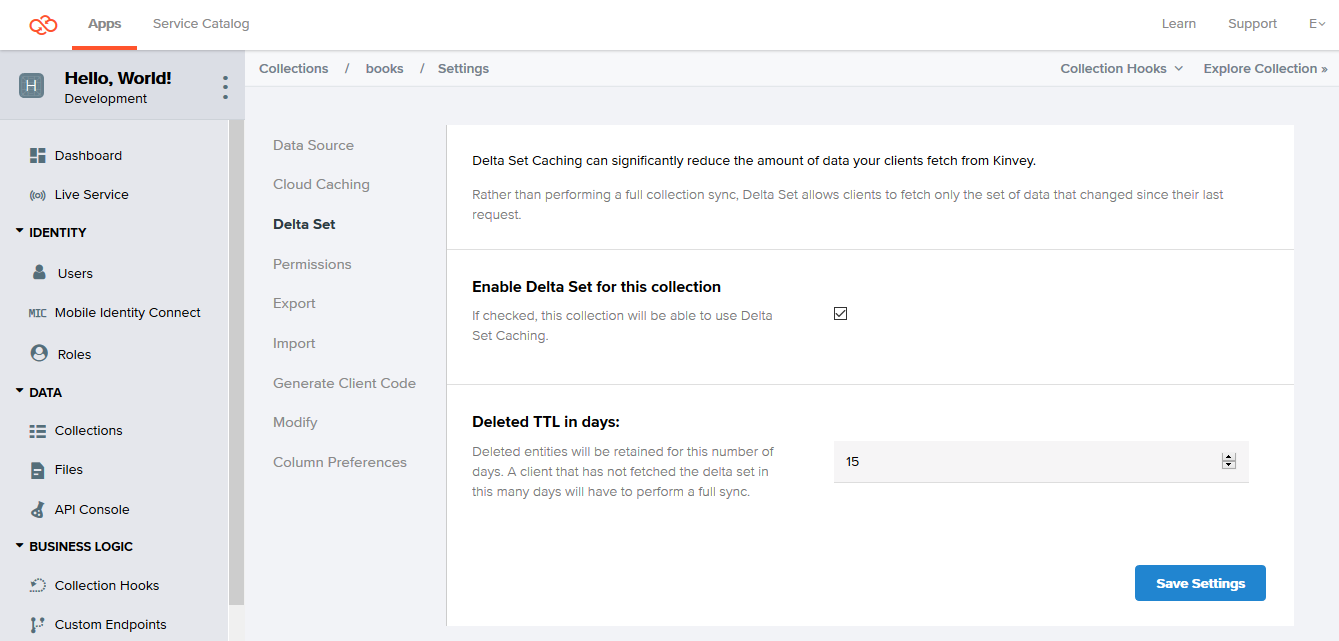
To turn on Delta Sync for a collection:

1. Log in to the Console.
2. Navigate to your app and select an environment to work with.
3. Under **Data**, click **Collections**.
4. On the collection card you want to configure, click the **Settings** icon.
5. From the Settings menu, click **Delta Set**.
6. Click **Enable Delta Set for this collection**.
7. Optionally, change the default **Deleted TTL in days** value.

The **Deleted TTL in days** option specifies the change history, or the maximum period for which information about deleted collection entities is stored. This change history is required for building a delta. Delta Sync queries requesting changes that precede this period return an error.

The maximum **Deleted TTL in days** you can set is 30 days.

Because Kinvey starts collecting data for Delta Sync only when you turn on the feature for a collection, the actual period for which a data delta can be retrieved can be shorter than the specified days. For example, if you turned on Delta Sync for a collection yesterday, you will only have one day's worth of change history instead of the configured 15. The change history will only reach and maintain its full size after the 15th day.



Delta Sync cannot be turned on for the User and Files collections.

Keeping and returning information about deletions is important, because without it, when receiving the data on the client, you won't be able to determine why the entity is missing from the data delta: because it has been deleted or because it has stayed unchanged.

Turning off Delta Sync for a collection results in permanently removing all information about deleted entities from the server. If you turn on Delta Sync again for the collection at a later stage, the accumulation of information about deleted entities starts from the beginning.

[**Using Delta Sync**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#UsingDeltaSync)

To request a data delta, send a GET request to the \_deltaset collection endpoint. As the queryparameter, include the URL-encoded query whose result set you want to sync. Finally, set the sincequery string parameter to the delta's starting point, which in most cases will be the last execution time of the request.

If you skip the query URL parameter, the endpoint returns a delta for the entire collection.

Usually, you would store the timestamp contained in the X-Kinvey-Request-Start header of the response to each GET request and pass it as the since parameter when requesting a delta for this query.

GET /appdata/:appKey/:collectionName/\_deltaset?since=2018-02-28T14:20:39.000Z&query=:urlEncodedQuery

HTTP/1.1

Host: baas.kinvey.com

Authorization: [user credentials]

Content-Type: application/json

{

changed: [

{ "\_id": "...", "myProp": "...", "\_acl": ... },

{ "\_id": "...", "myProp": "...", "\_acl": ... },

{ "\_id": "...", "myProp": "...", "\_acl": ... },

...

],

deleted: [

{ "\_id": "..." },

{ "\_id": "..." },

...

]

}

If the request features skip or limit modifiers, it returns an error.

Make sure the since timestamp is in the correct time zone for the Kinvey backend. An easy way to achieve this is to store the X-Kinvey-Request-Start header value contained in the response to the request's previous execution and pass it without modification.

The \_deltaset endpoint can return the following error responses:

| Error code | Error message | Description |
| --- | --- | --- |
| 400 | ParameterValueOutOfRange | Returned when the supplied since value predates the maximum change history. The history is configured per collection but can be shorter than specified if configured sooner than the specified days. |
| 400 | MissingRequestParameter | Returned when the since parameter is not found. |
| 400 | ResultSetSizeExceeded | Returned when the endpoint results in more than 10,000 entities. |
| 403 | MissingConfiguration | Returned when Delta Sync is turned off for the collection. |

If you receive 400 ResultSetSizeExceeded, Kinvey recommends requesting the full entity set and applying [paging](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#paging).

[**Related Samples**](https://devcenter.kinvey.com/rest/guides/datastore?fbclid=IwAR1OeZ6v9I6jKQgroo4wSobWAmj7PQe7l8cOH1CI5z9VqgrUj_MKqcmZgUs#RelatedSamples)

**[TestDrive](https://devcenter.kinvey.com/samples/testdrive)**