Moving to appveen data.stack 1.0 from Omni-Data Platform 3.9

Introduction

appveen data.stack 1.0 (data.stack) made significant changes to the APIs and data. These changes prevent a direct upgrade from Omni-Data Platform (ODP) 3.9 to data.stack.

This document is intended to be used as a migration guide for teams planning to move away from ODP to data.stack.

Important

We are going to take certain liberties w.r.t., naming and versions in this document. This is purely done to make it easier to consume the document.

1. **data.stack**, refers to appveen data.stack 1.0
2. **ODP**, refers to Omni-Data Platform 3.9

If you are on an earlier version of ODP, please upgrade to ODP version 3.9 first before moving to data.stack 1.0

Breaking changes

Data service API changes

We changed the way in which data service APIs are structured in data.stack. The API design focuses on the primary and secondary operations that can be performed on a data.

Operations that deal with the data saved within a data service is available at a root level ("/"). These are the CRUD APIs of the data service. We also managed to consolidate a few of these APIs, particularly around bulk updates and deletes.

Other operations on the data like simulate or file export have been clubbed under utils ("/utils").

Though changes to the APIs doesn't affect the user who predominantly uses Author and AppCenter UI, it affects anyone who is directly consuming the APIs.

Documented below are the API level changes for a data service.

We make the following assumption to better explain the changes.

App	Adam
Data service	Users
API endpoint	/users

Changes to endpoints

CRUD

ODP	data.stack	Description
/api/c/Adam/users	/api/c/Adam/users	Introduced *countOnly* as a query param to fetch only the count
/api/c/Adam/users	/api/c/Adam/users	
/api/c/Adam/users/count	---	Removed
/api/c/Adam/users/{id}	/api/c/Adam/users/{id}	
/api/c/Adam/users/{id}	/api/c/Adam/users/{id}	Option to bulk update with an array of documents is now supported.
/api/c/Adam/users/{id}	/api/c/Adam/users/{id}	Option to bulk delete with an array of ids is now supported.

Utils

ODP	data.stack	Description
/api/c/Adam/users/simulate	/api/c/Adam/users/utils/simulate	
/api/c/Adam/users/bulkShow	---	Removed
/api/c/Adam/users/securedFields	/api/c/Adam/users/utils/securedFields	
/api/c/Adam/users/bulkUpdate	---	Removed
/api/c/Adam/users/bulkDelete	---	Removed
/api/c/Adam/users/{id}/math	/api/c/Adam/users/utils/math/{id}	
/api/c/Adam/users/file/download/{id}	/api/c/Adam/users/utils/file/download/{id}	
/api/c/Adam/users/export/download/{id}	/api/c/Adam/users/utils/export/download/{id}	
/api/c/Adam/users/export	/api/c/Adam/users/utils/export	
/api/c/Adam/users/utils/aggregate	/api/c/Adam/users/utils/aggregate	
/api/c/Adam/users/experienceHook	/api/c/Adam/users/utils/experienceHook	

Changes in API behaviour

METHOD	URL PARAM	REQ. PAYLOAD	RES. PAYLOAD	RESULT
POST	No Effect	Object	Object	Insert single document
	No Effect	Array of Objects	Array of Objects	Insert multiple documents. Returns error at the same location in the array as the input
PUT	ID	Object	Object	Apply changes to a single document which matched the ID
	No Effect	Array of Objects	Array of Objects	Apply changes to multiple documents. Returns error at the same location in the array as the input
	Filter	Object	Object	Apply the same change to multiple documents that matches the filter query
DELETE	ID	No Effect	No Effect	Deletes a single document which matched the ID
	Filter	No Effect	No Effect	Deletes multiple documents that match the filter query.

Unsupported APIs

These APIs have been permanently removed from data.stack.

- `GET /api/c/Adam/users/count`
- `GET /api/c/Adam/users/bulkShow`
- `PUT /api/c/Adam/users/bulkUpdate`
- `DELETE /api/c/Adam/users/bulkDelete`

Date and Time format

With ODP we had the following challenges with saving date and time.

1. Datetime was always saved in UTC. This meant that we lose the timezone information that was part of the data
2. Exported data was in UTC. We never could export original data because the timezone information was missing. A temporary fix had to be put in where all time was converted into browser timezone.
3. Date and time queries had to be written in UTC. For a user who is in India (IST +5:30) had to make sure that to get all the records created on 1st April 2020, the query had to be modified to 31st March 2020 18:30 to 1st April 2020 18:30.
4. No time zone information could be captured without additional data model changes.

So we have redesigned the way in which date and time related values are handled in data.stack. This design in particular takes care of the vital timezone related information in date and time.

Important

This change is only for the date and time saved part of the data service schema.

- `_metadata.createdAt` and `_metadata.lastUpdated` fields would always be in Zulu time.
- **data.stack**, like ODP, must be deployed on a server where the server time is set to UTC/Zulu time

Changes to schema and data

In **data.stack**, Date and Date-time fields will cease to exist as a simple key-value pair. Instead the field will be saved as an object in the backend.

The following are the new set of attributes,

Field	Type	Description
`rawData`	`String`	Raw data entered by the user.
`tzData`	`JS Date`	Date-time form with time zone
`tzInfo`	`String`	Name of the timezone
`utc`	`JS Date`	A string representing the given date in the ISO 8601 format according to universal time.
`unix`	`Number`	Number of milliseconds since 1 January, 1970, 00:00:00 UTC, with leap seconds ignored.

E.g., for a field like executionTime, the data will be saved as

```
{
  "rawData": "2021-01-01T05:30:00.000Z",
  "tzData": "2021-01-01T14:30:00+09:00",
  "tzInfo": "Asia/Tokyo",
  "utc": "2021-01-01T05:30:00.000Z",
  "unix": 1609479000000
}
```

If you have existing date and time fields, they would have to be migrated to the new format, data and logs, to be able to work with data.stack.

Authentication modes

With data.stack we have removed the capability to switch to other authentication modes, viz, AzureAD and LDAP from the UI. Instead all these authentication modes can coexist and has to be configured via environment variables.

- RBAC_USER_AUTH_MODES
  - The supported authentication modes
  - Supported values - *local, azure, ldap*
- LDAP
  - The LDAP connection JSON

```
{
  "BASE_DN": "ou=internal,dc=example,dc=com",
  "BASE_FILTER": "(objectClass=inetOrgPerson)",
  "BIND_DN": "uid=admin,ou=system",
  "BIND_PASSWORD": "secret",
  "SERVER_URL": "ldap://myldapserver.com:389",
  "USER_ID_ATTRIBUTE": "cn",
  "USER_NAME_ATTRIBUTE": "sn",
  "USER_EMAIL_ATTRIBUTE": "mail"
}
```

- Azure
  - The Azure connection JSON

```
{
  "AD_USER_ATTRIBUTE": "mail",
  "B2C_TENANT": "myorg.onmicrosoft.com",
  "CLIENT_ID": "57ceab52-xxxx-xxxx-xxxx-25dad05xxxxx",
  "CLIENT_SECRET": "-XXXXXX_8m1-A.XXXXXX9Oo70_fpLXXXXX"
}
```

Because of this change you would have to update the documents under users collection (`userMgmt.users`). Each document must have the auth type defined.

Eg. for local

```
{ "auth": {"authType": "local"} }
```

Without this change the users won't be able to login.

Request payload changes to post- and pre-hook

Sample pre-hook payload


```
{
  "trigger": {
    "source": "presave",
    "simulate": false
  },
  "operation": "POST",
  "txnId": "Z1bIhdJ",
  "user": "jerry@appveen.com",
  "data": {
    "_id": "USE1001",
    "name": "Alice",
    "age": 23,
    "active": true,
    "dateOfBirth": {
      "rawData": "1990-01-01T00:00:00Z",
      "tzInfo": "Zulu"
    },
    "accountExpiry": {
      "rawData": "2029-01-31T00:16:24Z",
      "tzInfo": "Zulu"
    },
    "_metadata": {
      "deleted": false,
      "version": {
        "release": "1.0.0",
        "document": 1
      },
      "createdAt": "2021-03-03T18:46:35.520Z",
      "lastUpdated": "2021-03-03T18:46:35.520Z"
    }
  },
  "service": {
    "id": "SRVC2063",
    "name": "Users"
  },
  "name": "Validate",
  "app": "appveen-adam",
  "docId": "USE1001",
  "properties": {}
}
```


Sample post-hook payload


```
{
```

```
  "_id": "003b2f473304df4990ad96dbc5c9ffd1",
  "txnId": "Z1bIhdJ",
  "user": "jerry@appveen.com",
  "status": "Pending",
  "message": null,
  "retry": 0,
  "operation": "POST",
  "type": "PostHook",
  "trigger": {
    "source": "postSave",
    "simulate": false
  },
  "service": {
    "id": "SRVC2063",
    "name": "Users"
  },
  "callbackUrl":
"/api/c/Adam/users/utils/callback/003b2f473304df4990ad96dbc5c9ffd1",
  "properties": {},
  "docId": "USE1001",
  "data": {
    "old": null,
    "new": {
      "_id": "USE1001",
      "name": "prehook-name - 1614797195501",
      "age": 23,
      "active": true,
      "dateOfBirth": {
        "rawData": "1990-01-01T00:00:00Z",
        "tzData": "1990-01-01T00:00:00Z",
        "tzInfo": "Zulu",
        "utc": "1990-01-01T00:00:00.000Z",
        "unix": 631152000000
      },
      "accountExpiry": {
        "rawData": "2029-01-31T00:16:24Z",
        "tzData": "2029-01-31T00:16:24Z",
        "tzInfo": "Zulu",
        "utc": "2029-01-31T00:16:24.000Z",
        "unix": 1864512984000
      },
      "_metadata": {
        "deleted": false,
        "version": {
          "release": "1.0.0",
          "document": 1
        },
        "createdAt": "2021-03-03T18:46:35.520Z",
        "lastUpdated": "2021-03-03T18:46:35.520Z"
      },
      "__v": 0
    }
  },
```

```
  "name": "Post hook"
}
```

We also consolidated all the hook responses under an app to a single collection. With ODP all the hook logs used to be under the logs db under a collections

- `<AppName>.<data service name>.postHook`
- `<AppName>.<data service name>.pretHook`

With data.stack this will be under `<AppName>.hook`, under the logs DB.

Data service definition

ODP saved data service definition as a stringified JSON. Starting with data.stack we have started to save the definition as an array of attribute definitions. With this change we can maintain the same order of attributes at all time.

Here's a sample code to convert the existing definition to the new format.

```
function convert(definition) {
    let defList = [];
    if (definition && typeof definition == 'string') {
        definition = JSON.parse(definition);
    }
    if (Array.isArray(definition)) {
        return definition;
    }
    Object.keys(definition).forEach(key => {
        const def = definition[key];
        def.key = key;
        if (def.type === 'Object' || def.type === 'Array') {
          def.definition = convert(def.definition);
        }
        if (def.type === 'Date') {
            def.properties["defaultTimezone"] = "Zulu"
            def.properties["supportedTimezones"] = []
        }
        defList.push(def);
    });
    return defList;
}
```

Also note that the `_id` field now needs to have an explicit type defined as `String`. For the same code above, if will look something like this.

```
let newDefinition = convert(definition)
newDefinition[0]["type"] = "String"
```

Relationships

Relationships now depend on the new dataKey and dataPath property of an attribute. If you have existing relationships that stopped working, you would have to edit the data service by following these steps

1. Removed the relationship, either by removing the attribute or changing the attribute type to, say Text
2. Save
3. Add the relationship back.
4. Save and Deploy

If this still doesn't solve the problem,

1. Check the definition structure saved in the DB and makes sure that both data services have `dataKey` and `dataPath`
2. Check the definition of the data service which has the relationship defined and makes sure that the `relatedSearchField` under `properties` is set.

Getting help

If issues still exist please reach out to support@appveen.com.