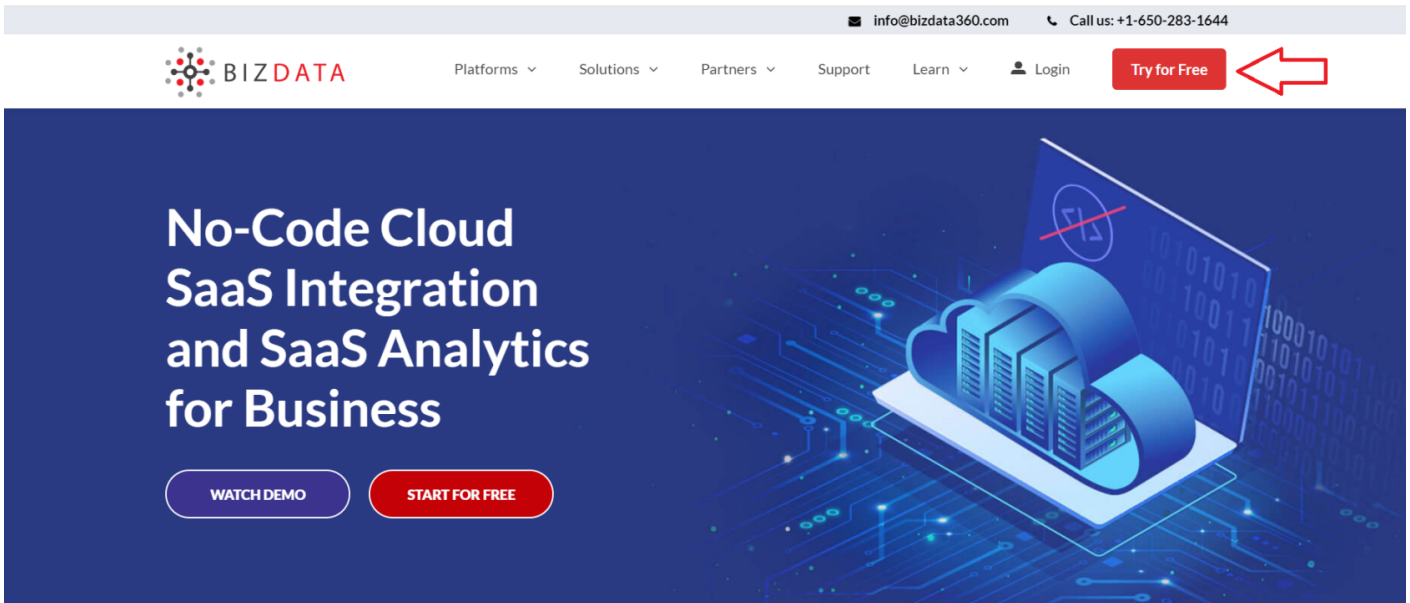


Onboarding

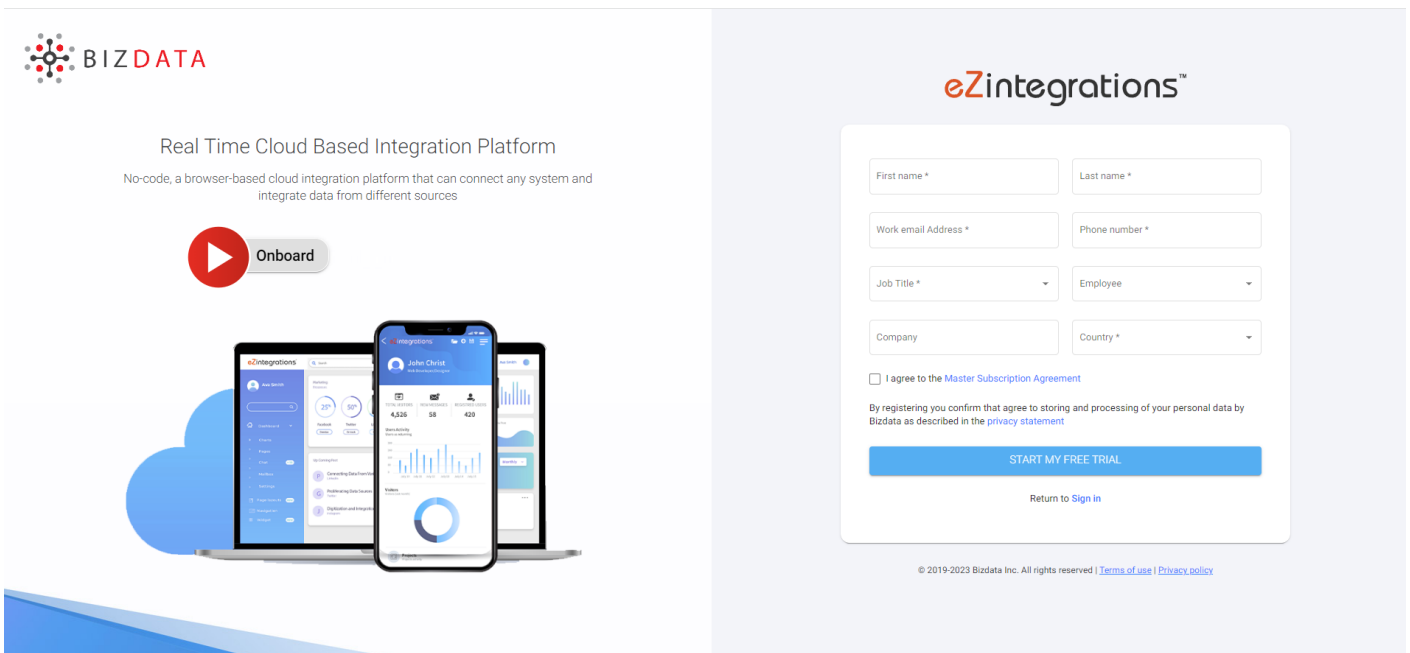
- [Login for new users](#)
- [Multi-Factor Authentication](#)
- [Adding Users under Organization Admin : Step-by-Step Guide](#)

Login for new users

Step 1: Click on the 'Try for free' button on our Bizdata website to launch our eZintegration application.



Step 2: You will be redirected to the eZintegration sign up window. Click on 'Onboard' button on the left-hand side panel to watch a demonstration video for easy sign up. After watching the video, you can start with filling the details in the Sign-up form.



Step 3: Fill up the form with details mentioned: First name, Last name, work email address, Phone number, and company. Select your job title, number of employees in the company, and country of your organization.

Read the **Master Subscription Agreement** and select the check box to agree to the agreement. Click on '**Start my free trial**' button to submit the form.

eZintegrations™

First name * Demo	Last name * User
Work email Address * testuser@bizdata360.com	Phone number * 0000000000
Job Title * IT Manager	Employee 101 - 500 Employees
Company Bizdata	Country * India

I agree to the [Master Subscription Agreement](#)

By registering you confirm that agree to storing and processing of your personal data by Bizdata as described in the [privacy statement](#)

START MY FREE TRIAL

[Return to Sign in](#)

© 2019-2023 Bizdata Inc. All rights reserved | [Terms of use](#) | [Privacy policy](#)

Step 4: After signing up, you will receive a welcome email with a link to verify your account. Click on the 'Verify account' button to start the account verification process. You will be redirected to the account verification page.

Step 5: You will receive a verification code over email. Enter the code under verification code text box to verify your email id. If you haven't received the code yet, click on the 'Resend code' button. Click on 'verify'.

Step 6: You will be redirected to the password change window. Enter a new password for your account. Confirm the password by entering the same password. Select a security question from the list provided. Provide an answer to the security question which should not be complete answer or can be guessed easily. Click on 'Set password' button.

Step 7: Login to the portal by entering your username and password. Select the 'Remember me' check box for easy login in future. You will be logged in to the eZintegration portal to explore various integration options.

Multi-Factor Authentication

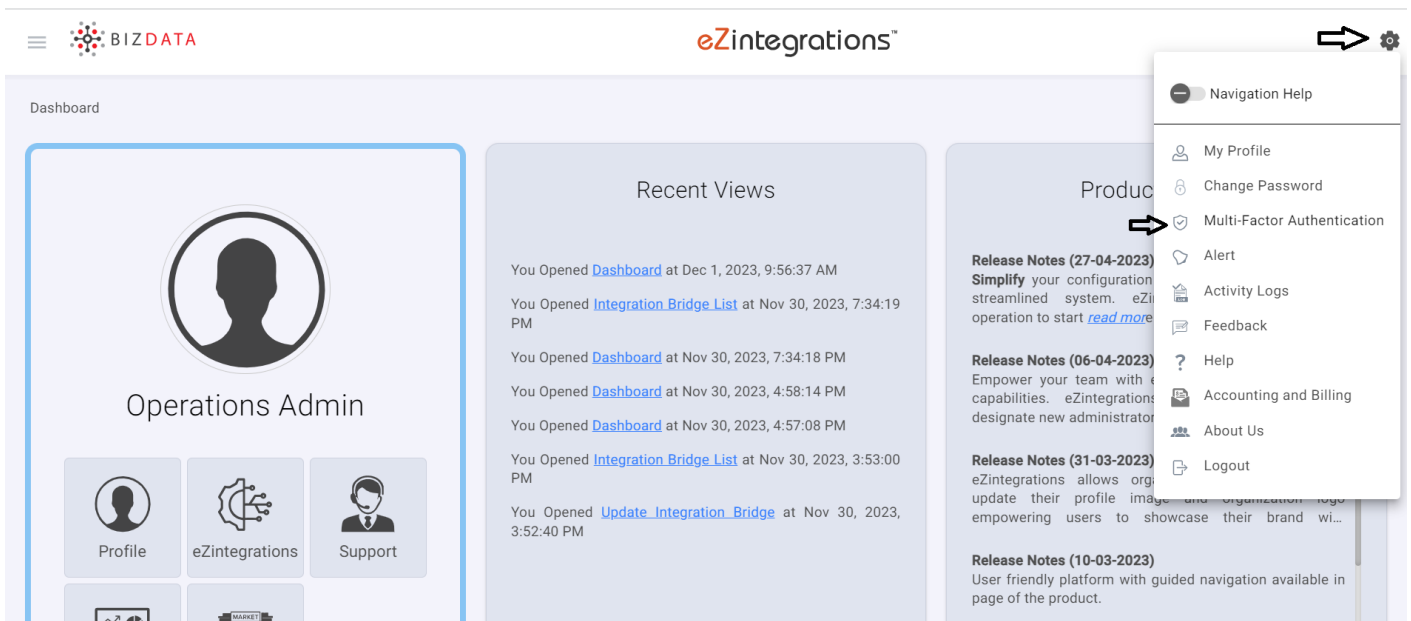
Multi-factor authentication (MFA) is a multi-step account login process that requires users to enter more information than just a password. For example, along with the password, users might be asked to enter a code sent to their email, answer a secret question, or scan a fingerprint. A second form of authentication can help prevent unauthorized account access if a system password has been compromised.

In eZintegrations, we use Two-factor authentication (2FA). 2FA is a specific type of multi-factor authentication (MFA) that strengthens access security by requiring two methods to verify your identity. These factors can include something you know — like a username and password — plus something you have — like a smartphone app — to approve authentication requests. 2FA protects against phishing, social engineering and password brute-force attacks and secures your logins from attackers exploiting weak or stolen credentials.

Enabling multifactor authentication:

Step 1: Log in to eZintegration.

Step 2: Click on “Settings” in the right-side panel and navigate to “Multi-Factor Authentication”.

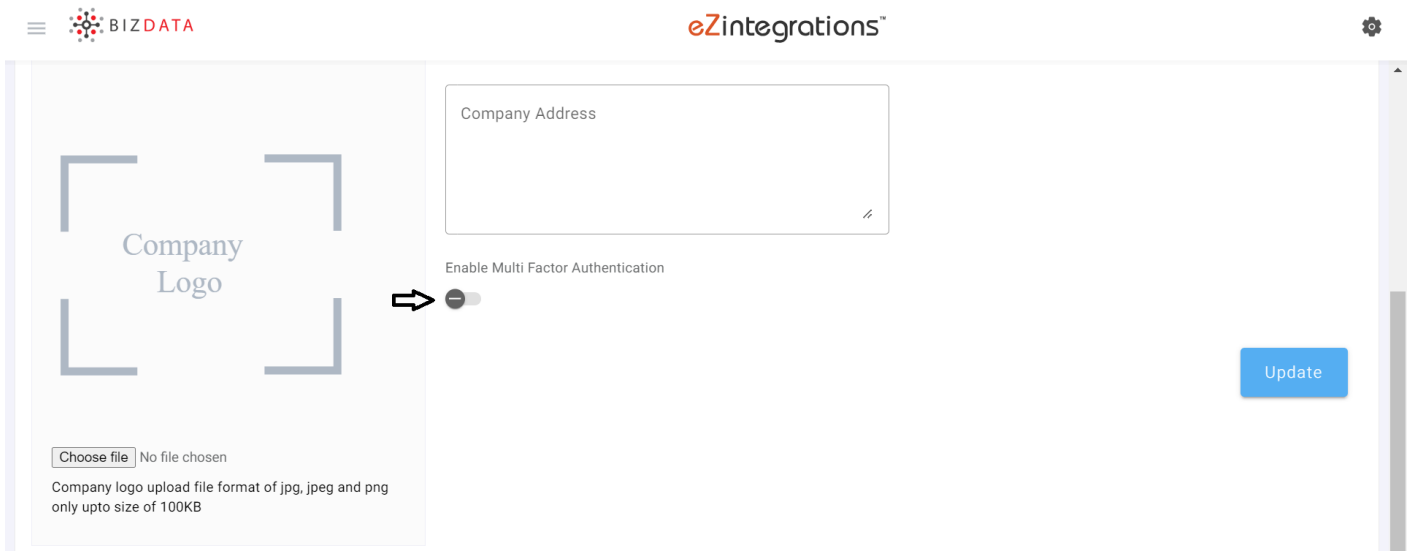


The screenshot displays the eZintegrations dashboard interface. At the top left, there is a 'BIZDATA' logo and a hamburger menu icon. The main header area includes the 'eZintegrations' logo and a settings gear icon. The dashboard content is divided into three main sections: a user profile card on the left, a 'Recent Views' log in the center, and a 'Products' section on the right. The user profile card for 'Operations Admin' features a profile picture placeholder and three quick-action buttons: 'Profile', 'eZintegrations', and 'Support'. The 'Recent Views' section lists several recent page visits with timestamps. The 'Products' section contains multiple 'Release Notes' with dates and brief descriptions. A settings dropdown menu is open on the right side, listing various user management options. An arrow points to the 'Multi-Factor Authentication' option in this menu.

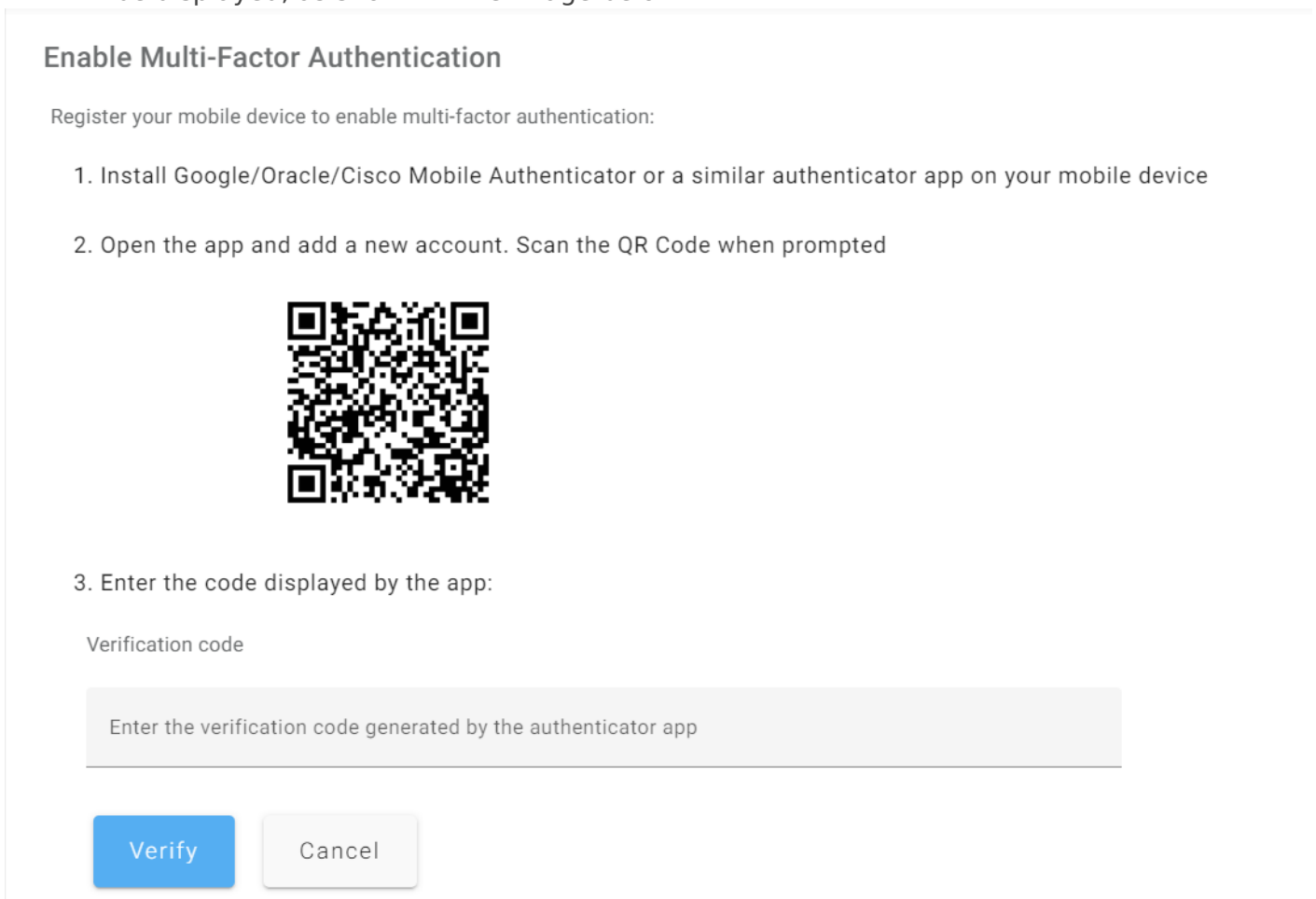
Navigation Help

- My Profile
- Change Password
- Multi-Factor Authentication
- Alert
- Activity Logs
- Feedback
- Help
- Accounting and Billing
- About Us
- Logout

Step 3: Toggle the “Enable Multi Factor Authentication” button.



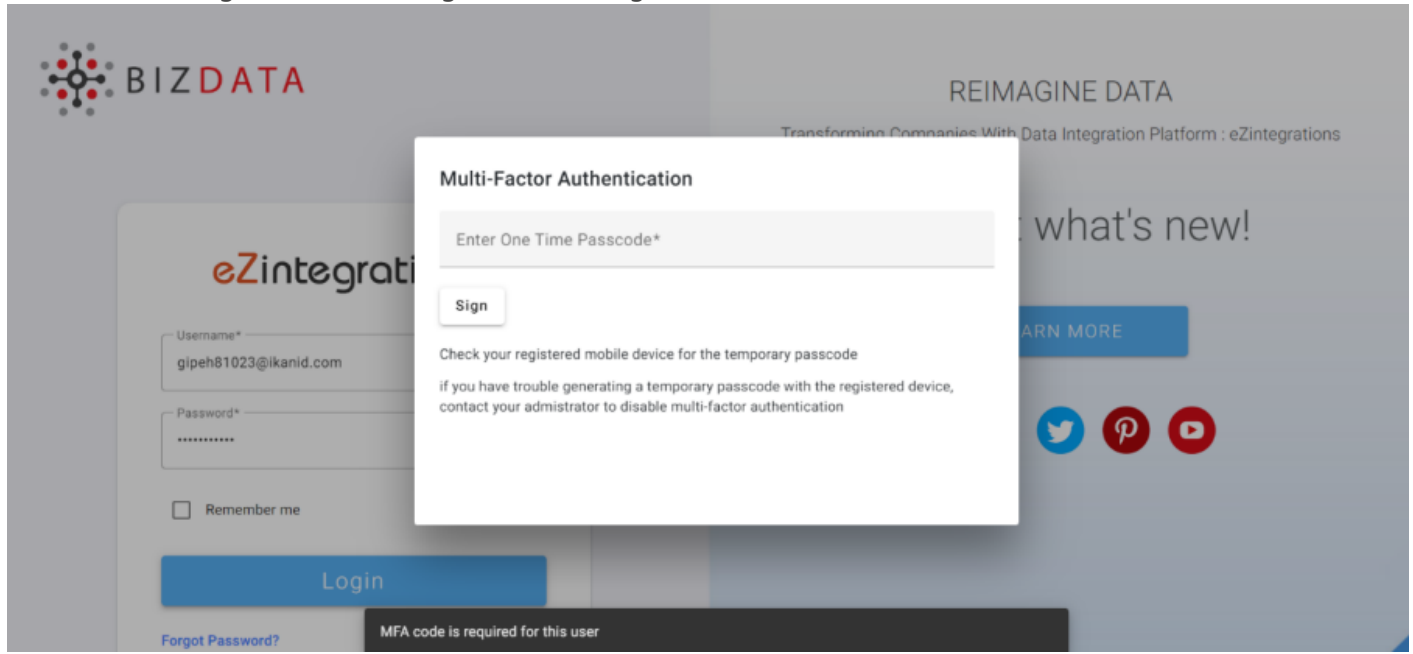
Step 4: On toggling the “Enable Multi Factor Authentication” button, all the information regarding 2FA will be displayed, as shown in the image below.



Step 5: Download and install the Google Authenticator app on your smartphone. Complete the necessary initial app setup. Click on “Add a Code” or the “+” (plus) symbol, then choose “Scan the QR code”. Scan the QR code displayed on the 2FA information. After successful scanning, the authenticator app will generate a time-based code every 30 seconds. Enter the code from the app and click on the “Verify” button.

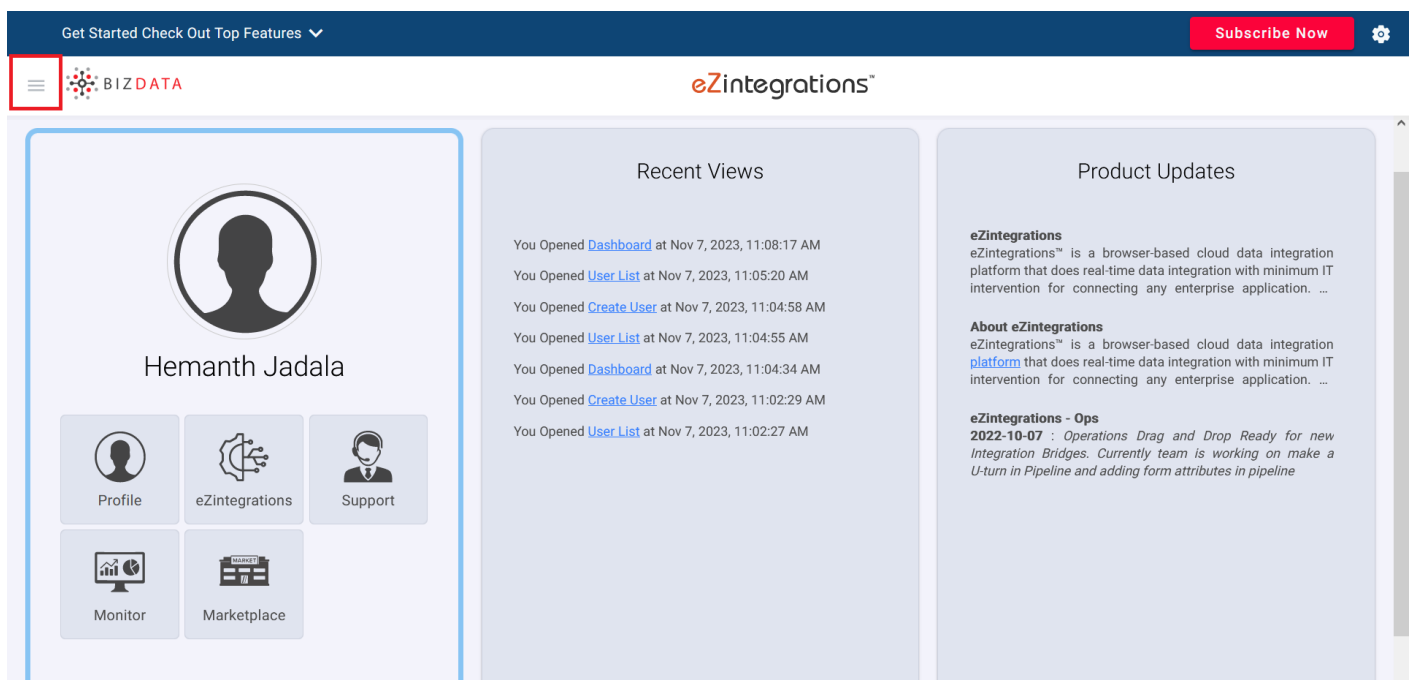
Step 6: Upon successful verification, a popup will appear, indicating “MFA enabled”. The “Multi Factor Authentication” button will be enabled. Click on the “Update” button, and a popup will confirm “Successfully Updated profile”. If you choose not to enable MFA, toggle the “Multi Factor Authentication” button. A popup will appear, stating “MFA disabled”. Click on the “Update” button at the bottom of the page, and another popup will confirm “Successfully Updated profile”.

Step 7: Whenever you log out and attempt to log in again, a popup will prompt you for the Multi Factor Authentication One Time Passcode. Enter the code displayed on the authenticator app and click on the “Sign” button to log in to eZintegration.

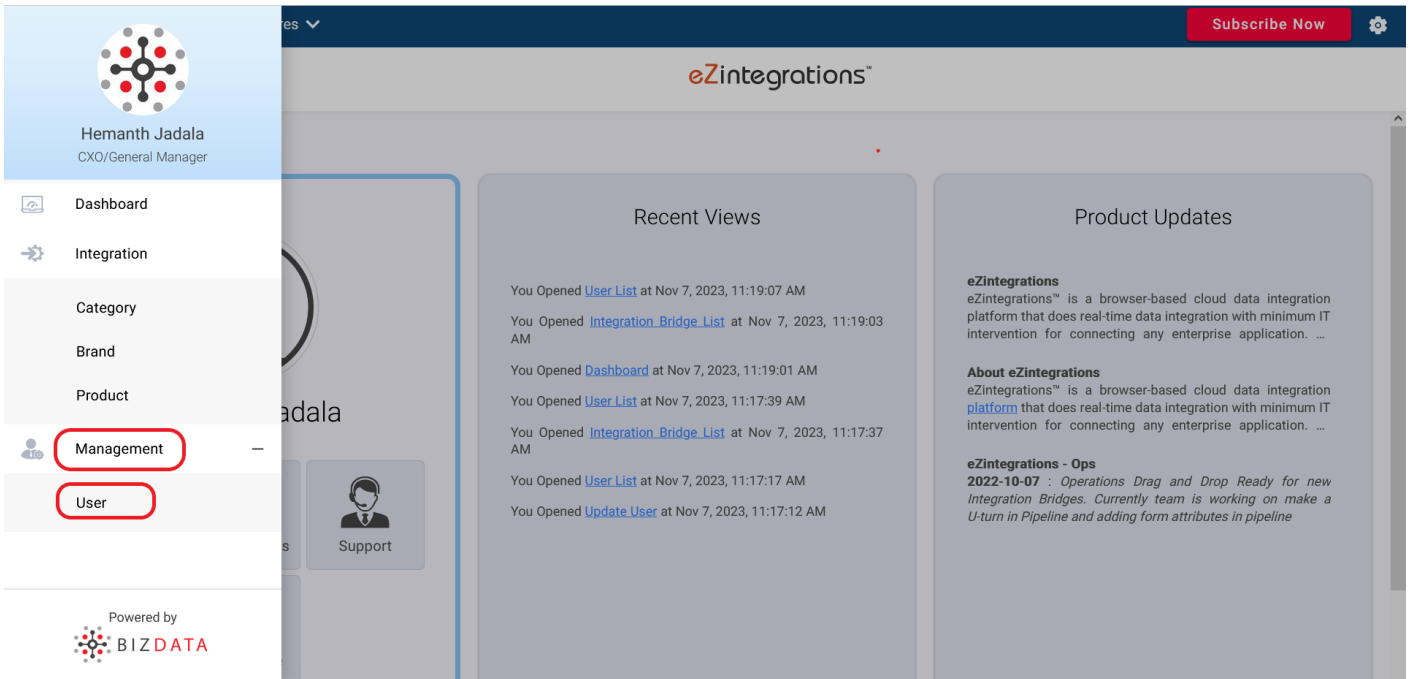


Adding Users under Organization Admin : Step-by-Step Guide

Step 1 : Click on the three horizontal lines located to the left of the screen, just after the Bizdata logo.

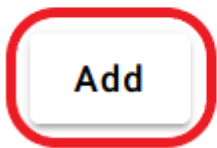


Step 2 : Click on the "Management" option, which becomes visible when you click on the three horizontal lines. Then, under the "Management" menu, click on "User."



Step 3 : Click on the "Add" button, and fill in all the necessary details of the user.

User



User / Add

User

First name*

Last name*

Email*

Select Permission*

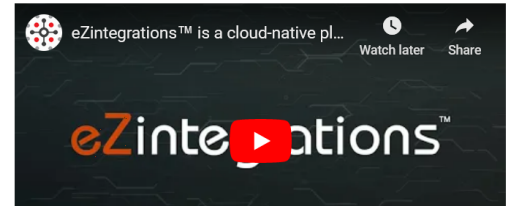
 Make Admin

Cancel

Add

About eZintegrations

eZintegrations™ is a browser-based cloud data integration platform that does real-time data integration with minimum IT intervention for connecting any enterprise application. By leveraging eZintegrations™ no-code iPaaS, users can connect data from fragmented sources and empower organizations to consolidate multiple cloud solutions into a single platform. It can integrate any system traditional RDBMS, NoSQL, Cloud Databases, Data Warehouse, Data Lake, REST API, SOAP API or logs, videos, voice notes or stats from Linux commands as well as various protocols from IoT sensors.



Step 4 : After filling in the First name, Last name, and Email, select the user permission.

Select Permission*

 List (User) Add (User) Edit (User) Details (User) List (Integration Bridge) Add (Integration Bridge)

Access Levels:

1. User Access:

- LIST: View a list of users under the organization.
- ADD: Add new users to the organization.
- EDIT: Modify user details within the organization.
- DELETE: Remove users from the organization.

- DETAILS: Access detailed information of users within the organization.
2. Integration Bridge Access (Same controls as User Access):
- LIST: View a list of users under the organization.
 - ADD: Add new users to the organization.
 - EDIT: Modify user details within the organization.
 - DELETE: Remove users from the organization.
 - DETAILS: Access detailed information of users within the organization.
 - VIEW STREAMING LOGS: Access real-time streaming logs.
 - DOWNLOAD LOGS: Download logs for analysis.

Step 5 : Click on "Add." The new user will receive a verification email, through which they can change the password and begin using their account.


Note: Only new users, not existing ones, can be added to the organization


Accounting & Billing



- Dashboard
- Plan
- Current Plan
- Payment Method
- Billing Information
- Invoices

Dashboard

This page contains information of the users billing amount, billing cycle, billing period and the details of the payment method.

Get Started Check Out Top Features 

[Subscribe Now](#) 

Accounting and Billing

[Dashboard](#) [Plan](#) [Current Plan](#) [Payment Method](#) [Billing information](#) [Invoices](#)

Your Next Payment

A total of will be charged on

*Indicates estimated sales tax or VAT, calculated based on your primary company address. Your recurring fees are subject to increase based on your usages

Billing Cycle	Billing Period	Payment Method
Monthly	4/29/2023 - 5/29/2023	VISA ending in

Need help?

Visit the [Knowledge base](#) for answers to FAQs or [Contact us](#) for additional support.

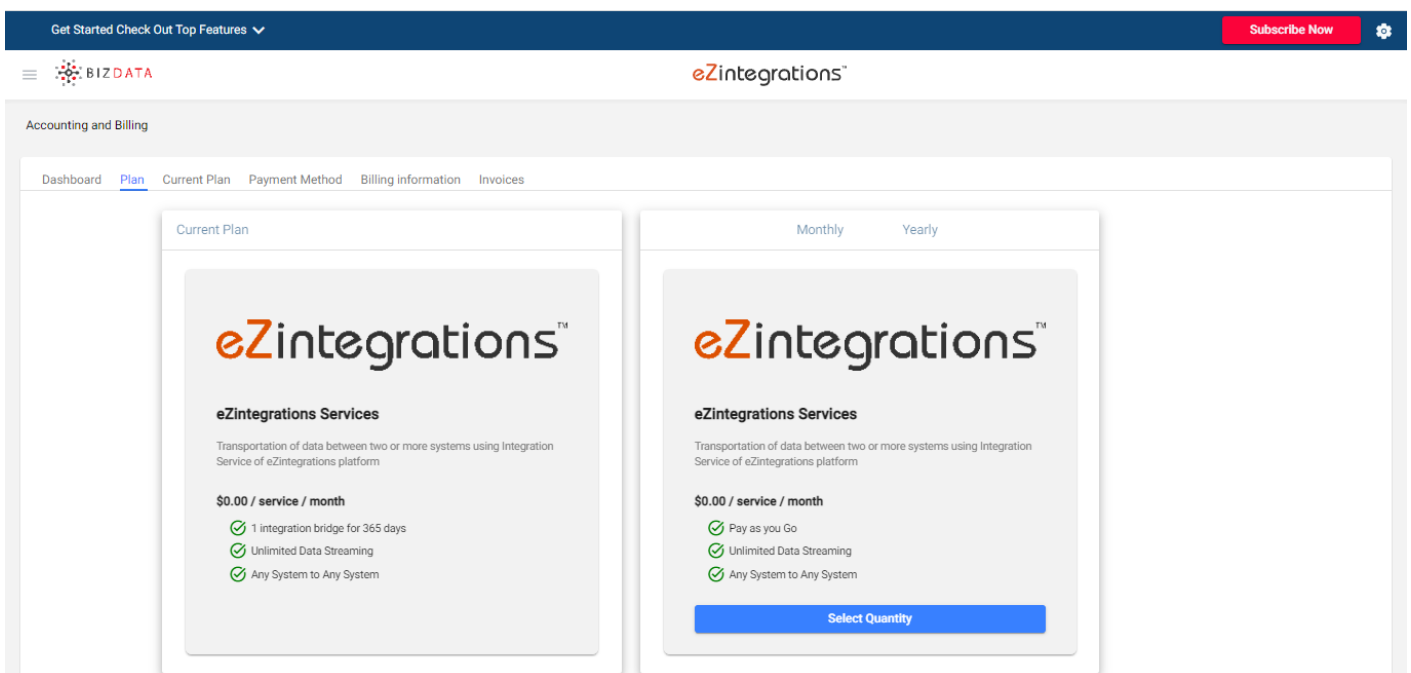
Plan

First Time User

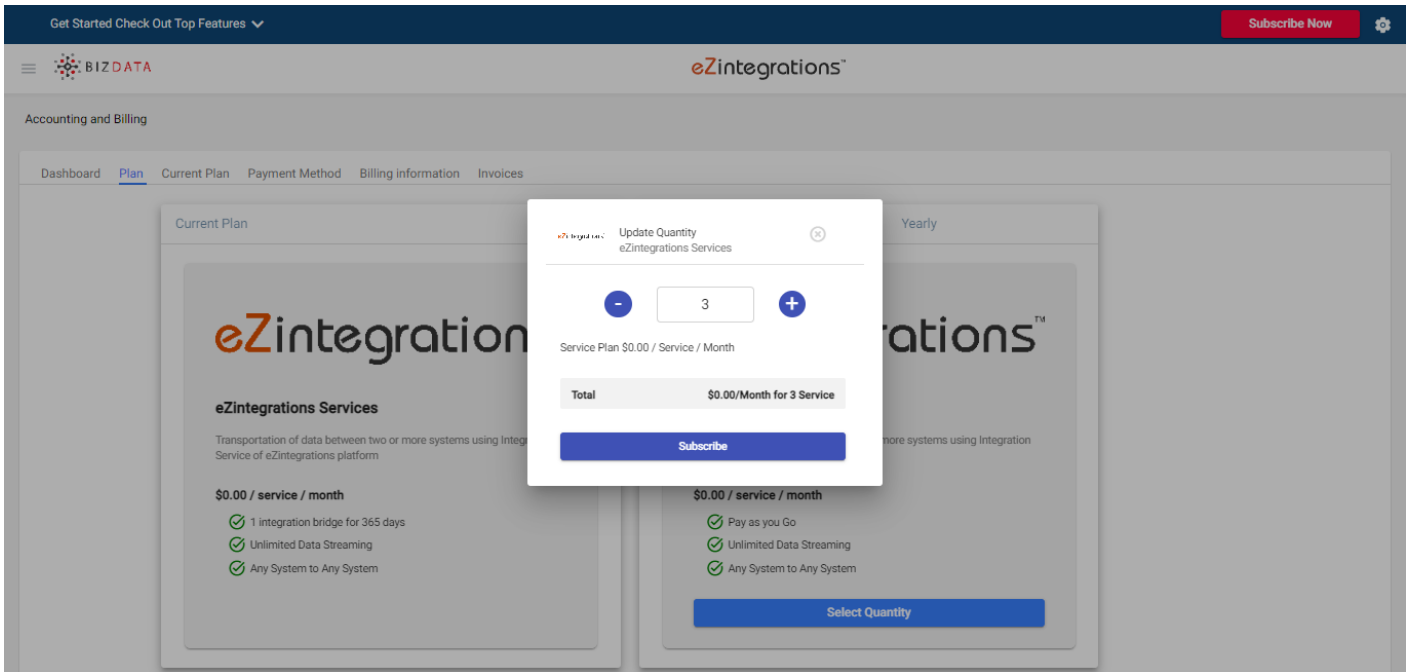
This page has information related to the plans, pricing and payment options.

Users can select the required plan (Monthly or Yearly) and Quantity of integrations and click 'Subscribe'. Users will be redirected to payment gateway where details regarding payment method should be entered. Once the inputs are correctly entered, click 'Subscribe' to avail the services.

Step 1: Select plan (Monthly or Yearly)



Step 2: Select the required quantity and click 'Subscribe'



Step 3: Enter the card details and other required details and click 'Subscribe'.

BIZ DATA TEST MODE

Subscribe to eZintegrations Services

\$130.00 per month
until coupon expires

Transportation of data between two or more systems using
Integration Service of eZintegrations platform

eZintegrations Services	\$195.00
Qty 3, Billed monthly	\$65.00 per Qty.
Subtotal	\$195.00
BizdataONE	-\$65.00
\$65.00 off for 12 months	
Total due today	\$130.00

Powered by stripe | Terms | Privacy

Pay with card

Email

Card information REQUIRED

1234 1234 1234 1234


MM / YY CVC

Name on card

Country or region

India

Securely save my information for 1-click checkout
Pay faster on Bizdata Inc and everywhere Link is accepted.

 1234 567 891

By saving my info, I agree to the [Link Terms](#) and [Privacy Policy](#).

[link](#) · [More info](#)

Subscribe

If quantity of integrations selected is 4, users will be charged only for 3 integrations since the first integration is free. Charge of the first integration will be auto deducted from the bill with code BizdataONE as displayed on payment gateway screen.

Plan Upgradation


In case of upgradation of the previously selected plan, users can select the required quantity from the 'Select Quantity' option and increase the quantity as per requirement.



If the user selected quantity 2 (Integrations) previously and wants to upgrade to 5 then a total of quantity 5 should be selected in the 'Select Quantity' option. Pricing of only the increased quantity would be added to the previously calculated bill.

The screenshot displays the eZintegrations user interface. At the top, there is a dark blue navigation bar with the text "Get Started Check Out Top Features" and a "Subscribe Now" button. Below this, the "BIZ DATA" logo and "eZintegrations" brand name are visible. The main content area is titled "Accounting and Billing" and contains a navigation menu with "Dashboard", "Plan", "Current Plan", "Payment Method", "Billing information", and "Invoices". The "Plan" section is active, showing a "Monthly" plan card for "eZintegrations Services". The card includes the eZintegrations logo, a description of the service, the price "\$65.00 / service / month", and three features: "Pay as you Go", "Unlimited Data Streaming", and "Any System to Any System". A blue "Select Quantity" button is located at the bottom of the card.

Current Plan

This page contains information of the current selected plan.

Get Started Check Out Top Features 

Accounting and Billing

[Dashboard](#) [Plan](#) [Current Plan](#) [Payment Method](#) [Billing information](#) [Invoices](#)

eZintegrations Services

Free

Quantity : 1

Payment Method

This page has information of the payment method used by the user.

Users can update/ change their payment method from edit option available beside their payment method information.

The screenshot displays the eZintegrations user interface. At the top, there is a dark blue navigation bar with the text "Get Started Check Out Top Features" and a "Subscribe Now" button. Below this, the "BIZ DATA" logo and "eZintegrations" branding are visible. The main content area is titled "Accounting and Billing" and contains a navigation menu with "Dashboard", "Plan", "Current Plan", "Payment Method" (highlighted), "Billing information", and "Invoices". The "Payment Method" section shows a single entry for "VISA" with a red checkmark, "Ending in xxxx", and "Expires: 12/2025". A small pencil icon next to the entry is highlighted by a large black arrow, indicating the edit option.

Billing Information

This page contains Billing Information of the selected plan in the format as mentioned below

First Name:

Last Name:

Work Email Address:

Phone Number:

Job Title:

Employee:

Company:

Country:

Subscription Start Date:

Accounting and Billing					
Dashboard	Plan	Current Plan	Payment Method	Billing information	Invoices
First Name	Pushpi				
Last Name	Biz				
Work Email Address					
Phone Number	+1 214-599-7271				
Job Title	Consultant				
Employee	101 - 500 Employees				
Company	Biz				
Country	United States				
Subscription Start Date	5/3/2023				

Invoices

This page contains information of the Invoices of the plans subscribed.

Users can find Invoice ID, Invoice Date, Amount, Status of the selected plan and option to download the Invoice.

Get Started Check Out Top Features ▼ Subscribe Now ⚙️

⌵ BIZ DATA eZintegrations™

Accounting and Billing

Dashboard Plan Current Plan Payment Method Billing information Invoices

Search

Invoice ID	Invoice Date	Amount	Status	Download
			paid	
			paid	

Data Source - API

- Test the Data Source
- Authorization
- Time Parameters
- Numeric Parameters
- Pagination
- Environment Settings
- Response Params
- Pre- request Scripts

Test the Data Source

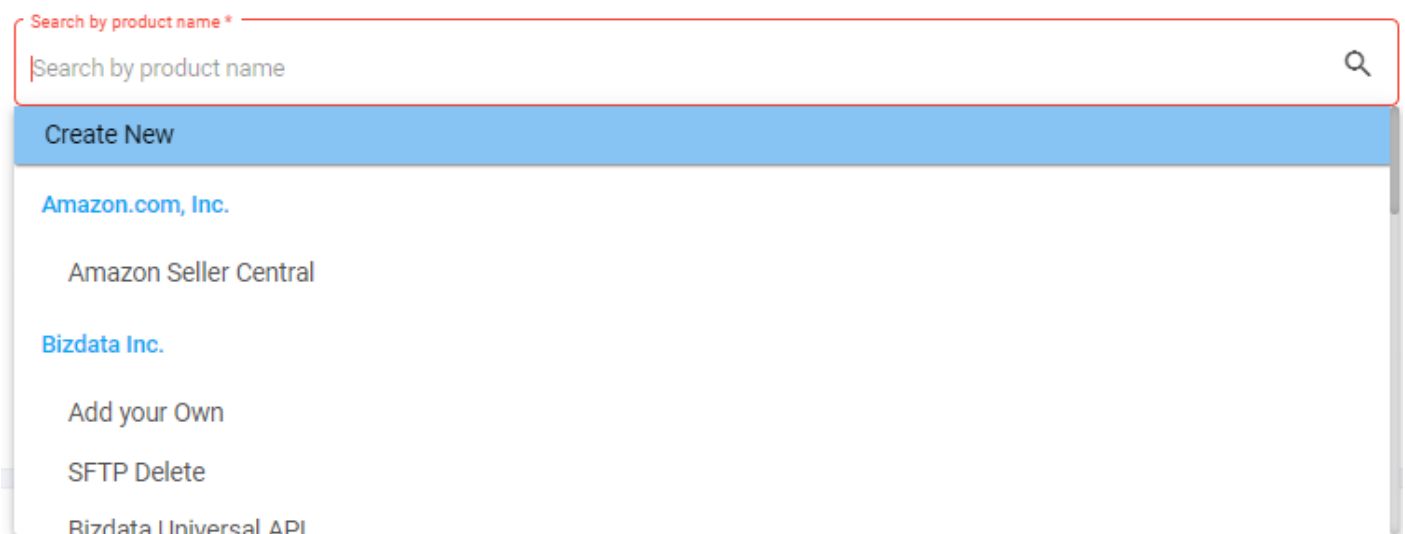
To test the data source-

Step 1:

Specify or add the data source from Catalog

What is your Data Source?

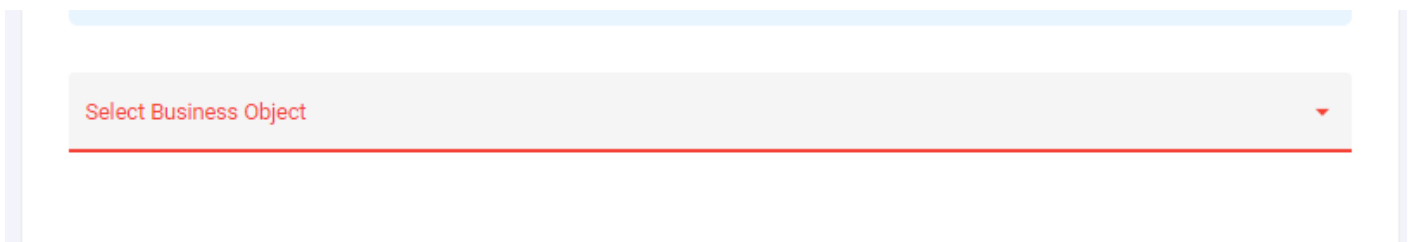
Please specify or select the Data Source from Catalog



The screenshot shows a search bar with the placeholder text "Search by product name" and a magnifying glass icon on the right. Below the search bar is a dropdown menu with a blue header "Create New". The menu items are: "Amazon.com, Inc." (with "Amazon Seller Central" indented below it), "Bizdata Inc." (with "Add your Own" and "SFTP Delete" indented below it), and "Bizdata Universal API".

Step 2:

Select the business object



The screenshot shows a dropdown menu with a light blue header and a grey body. The text "Select Business Object" is displayed in red, and a small red downward arrow is visible on the right side of the dropdown.

Step 3:

URL will be generated. All the fields pre-defined in API catalog so click on 'Test' to get response.

GET

Request URL

Test

< Params Authorization Headers Body Pre-request Script Pagination Numeri >

Key

Value

Description

+ -

Response

1	{}
---	----

Ln: 1 Col: 1

Authorization

eZintegrations platform supports all sort of Authorization for any standard APIs like REST API, SOAP API, gRPC or GraphQL.

The major Authorizations are :-

- API Key
- Basic Authentication
- OAuth1.0
- OAuth2.0
- AWS Signature
- NTLM Authentication
- Custom Algorithm Signature
- Any other custom architecture based Authorization

API Key

API Key Authorization is a method used to secure access to an API (Application Programming Interface) by requiring clients to include a unique API key when making requests. This key is typically a long alphanumeric string that is issued to the client by the API provider. API keys serve as a form of authentication and authorization, allowing the API provider to identify and control access to their API.

The API provider generates a unique API key for each client or application that needs access to their API. This key is often associated with specific usage limits or permissions.

Clients must include their API key in the headers or query parameters of their API requests. This key is used to identify the client making the request.

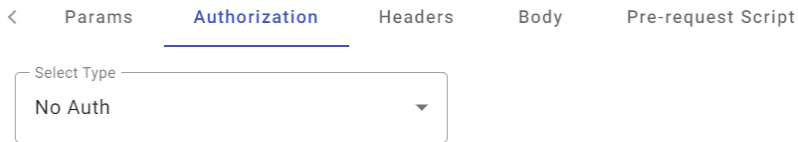
API Key is key-value pair based authorization where the key is **Authorization** and the value which is one time string generated from Application while enabling the API for defined set of action with roles and privileges.

There are various application whose API Key are static and its a one time activity to enable those keys, keep them in secret mode and use them for API Authorization

Authorization :

```
noKWB57FVhzjh11FRlv9HEghY94gXHyoFPPS3QIHETbGIZnMqDsPjsCiWkoDNE7W6UA2gcITtZTs4dZbiP20a9shY06lX5icKkRSdcQBnDTyaRno5BQphBdW0sX6sQto
```


In eZntegrations API Data Source user can select Type as **No Auth** under **Authorization** and under **Headers** call the key as **Authorization** and the value as you get from the application



The screenshot shows the 'Authorization' tab selected in a configuration interface. At the top, there are tabs for 'Params', 'Authorization', 'Headers', 'Body', and 'Pre-request Script'. Below the tabs, there is a dropdown menu labeled 'Select Type' with 'No Auth' selected.

And under Headers add the key-value pair

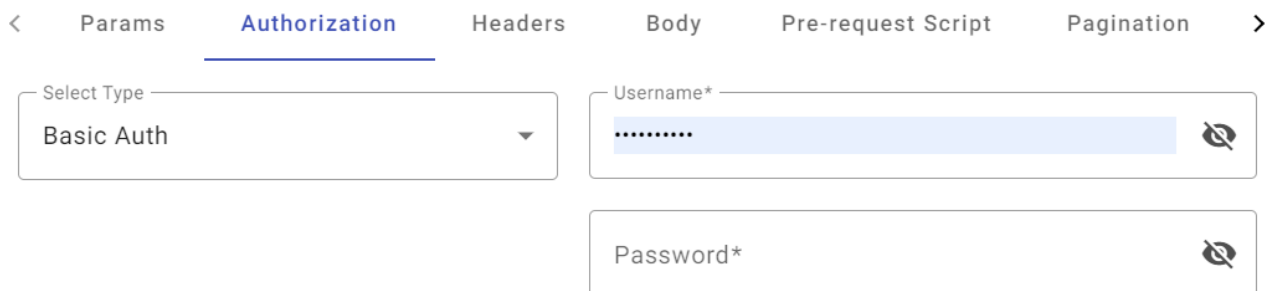


The screenshot shows the 'Headers' tab selected in the configuration interface. At the top, there are tabs for 'Params', 'Authorization', 'Headers', 'Body', and 'Pre-request Script'. Below the tabs, there are three input fields: 'Authorization' with the value 'noKWB57FVhzh11FRlv9HEghY', and 'Description'.

Basic Authentication

Basic Authentication is a simple method of authenticating access to an API (Application Programming Interface) by sending a username and password with each HTTP request.

In eZintegrations for Source, Operations and Target APIs user can select Type as **Basic Auth** under **Authorization** Tab



The screenshot shows the 'Authorization' tab selected in the configuration interface. At the top, there are tabs for 'Params', 'Authorization', 'Headers', 'Body', 'Pre-request Script', and 'Pagination'. Below the tabs, there is a dropdown menu labeled 'Select Type' with 'Basic Auth' selected. To the right, there are two input fields: 'Username*' with a masked value and a clear icon, and 'Password*' with a clear icon.

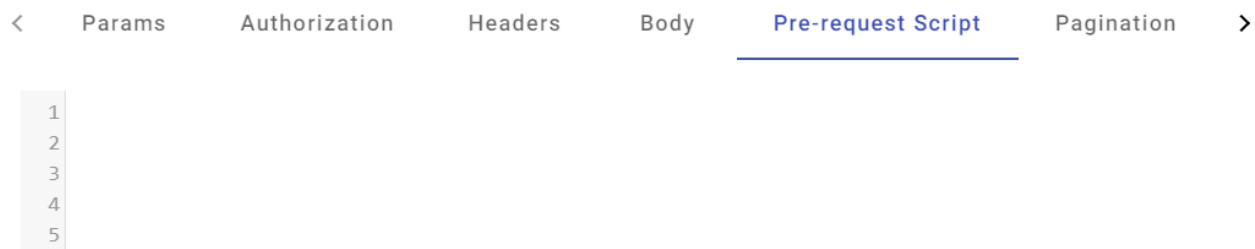
OAuth 1.0

OAuth 1.0: OAuth 1.0 is an open standard protocol that allows third-party applications to access a user's resources (e.g., data) on a server without exposing the user's credentials. It relies on cryptographic signatures and token-based authentication for secure API access.

Here are some key components and steps in the OAuth 1.0 process:

1. **Consumer:** The consumer is the third-party application or service that wants to access a user's data on a resource server.
2. **Service Provider (Provider):** The service provider is the entity that hosts the user's resources and manages access to them. It includes popular platforms like Twitter and Flickr.
3. **User:** The user is the resource owner who has data on the service provider's platform.
4. **Token:** OAuth 1.0 uses temporary tokens (OAuth token) to grant access to resources. These tokens are issued by the service provider.
5. **Signature:** OAuth 1.0 requires the use of cryptographic signatures to ensure the integrity and authenticity of API requests.

Since OAuth 1.0 is versatile and hard to predict, eZintegrations provides flexibility to write Python code under **Pre-request Script** to achieve the token or signature for Authorization requirement. If user is not aware as how to use the OAuth 1.0, plenty of examples available under section **Pre-Request Script**



OAuth 2.0

OAuth 2.0: OAuth 2.0 is an authorization framework that enables third-party applications to access a user's data or perform actions on a user's behalf through a secure and standardized process. It defines various authorization grant types and flows for different use cases.

Key components and steps in the OAuth 2.0 process:

1. **Resource Owner:** The resource owner is the user who has control over their resources and data.
2. **Client:** The client is the third-party application or service that wants to access the user's resources on a resource server.
3. **Resource Server:** The resource server hosts the user's data and resources, which the client wants to access.
4. **Authorization Server:** The authorization server is responsible for authenticating the user and issuing access tokens to the client after the user grants consent.
5. **Access Token:** OAuth 2.0 uses access tokens, which are short-lived credentials that the client presents to the resource server to access protected resources. They can also include scopes that define the level of access the client has.

In eZintegrations for Source, Operations and Target API user can select the **Type** as **OAuth 2.0**

This Authorization will help you to generate a `Access Token` based on `Refresh Token`

The screenshot shows a configuration interface for an authorization request. At the top, there are tabs for 'Params', 'Authorization' (which is selected), 'Headers', 'Body', 'Pre-request Script', and 'Pagination'. Below the tabs, there is a 'Select Type' dropdown menu set to 'OAuth 2.0'. To the right of this menu are two input fields: 'Refresh Token URL*' and 'Refresh Token Method*'. Below these fields is a section titled 'Refresh Token Endpoint Header' with a table containing one row and one column, with the number '1' in the cell.

Refresh Token URL :

The "Refresh Token URL" is the specific endpoint on the authorization server where the client sends a request to obtain a new access token using a refresh token. This URL is provided by the authorization server as part of the OAuth 2.0 protocol.

Example : <https://www.googleapis.com/oauth2/v4/token>

Refresh Token Method :

In OAuth 2.0, when a client application needs to obtain a new access token using a refresh token, it typically makes a POST request to a specific endpoint known as the "Token Endpoint" rather than a "Refresh Token URL." The Token Endpoint is where the client exchanges its refresh token for a new access token

HTTP POST Request: The client sends an HTTP POST request to the Token Endpoint of the authorization server. This request includes the following parameters:

- `grant_type` : This parameter should be set to "refresh_token" to indicate that the client is using a refresh token to obtain a new access token.
- `refresh_token` : The refresh token issued to the client during the initial authorization process.
- `client_id` and `client_secret` (optional): These are the credentials of the client application if it's required by the authorization server. Not all OAuth 2.0 implementations use client

credentials in this step.

eZintegrations provides all method available.

Refresh Token Endpoint Header:

Some of the common headers that can be used during the process of getting an access token via a refresh token:

Request Headers

Authorization Header: If the client needs to authenticate itself with the authorization server, it includes the client's credentials (client_id and client_secret) in the Authorization header using the "Basic" authentication scheme. For example:

```
{"Authorization": "Basic base64-encoded(client_id:client_secret)"}
```

This header is required in cases where the client authentication method involves client credentials.

Content-Type Header

The Content-Type header specifies the format of the data being sent in the request body. For OAuth 2.0, the Content-Type is typically set to "application/x-www-form-urlencoded" for form-encoded data.

```
{"Content-Type":"application/x-www-form-urlencoded"}
```

Refresh Token Endpoint Params :

Refresh Token Endpoint Params are the parameters that may needed to extend in Refresh Token URL which may be needed for filtering the request. Below is such example to add in this field

```
{  
  "params1": "your_params1",  
  "params2": "your_params2"  
}
```

Refresh Token Endpoint Body:

In OAuth 2.0, the term "Refresh Token Endpoint Body" typically refers to the parameters that are included in the HTTP request body when a client application requests a new access token using a refresh token. This process is part of the OAuth 2.0 token refresh flow, where the client exchanges a valid refresh token for a fresh access token. The refresh token body contains specific parameters that convey the client's intent and identity to the authorization server.

Here are the key parameters that are commonly included in the refresh token endpoint body:

grant_type Parameter: This parameter is required in the request body and specifies the grant type being used. In the context of a token refresh request, it should be set to "refresh_token" to indicate that the client is using a refresh token to obtain a new access token.

Example:

```
"grant_type":"refresh_token"
```

refresh_token Parameter: The refresh_token parameter is also required and contains the refresh token issued to the client during the initial authorization process. This refresh token is used as proof that the client has previously been granted access and is eligible for a new access token.

Example:

```
"refresh_token":"your_refresh_token"
```

client_id Parameter (Optional): Some OAuth 2.0 implementations require the inclusion of the client_id parameter in the refresh token request to identify the client making the request. It depends on the authorization server's configuration.

Example:

```
"client_id":"your_client_id"
```

client_secret Parameter (Optional): If the authorization server requires client authentication (using a client secret), the client_secret parameter should be included in the request. Not all OAuth 2.0 implementations use client credentials in the token refresh flow.

Example:

```
"client_secret":"your_client_secret"
```

When a client sends a POST request to the token endpoint of the authorization server with the above parameters in the request body, the authorization server processes the request. If the refresh token is valid, and any required client authentication is successful, the authorization server issues a new access token in the response body.

Here is an example of a **Refresh Token Endpoint Body**:

```
{
  "grant_type": "refresh_token",
  "refresh_token": "your_refresh_token",
  "client_id": "your_client_id",
  "client_secret": "your_client_secret"
}
```

Refresh Token Endpoint Body can be given in **JSON** format or in **String** Format. Preferred one is in **JSON** format

It's important to note that the specific parameters required and the format of the refresh token body may vary depending on the OAuth 2.0 implementation and the authorization server's policies.

When working with OAuth 2.0 and sending data in URL-encoded format, user can provide values in the **Refresh Token Endpoint Body** field in the following JSON format :

```
{
  "grant_type": "password",
  "client_id": "your_client_id",
  "client_secret": "your_client_secret",
  "username": "your_username",
  "password": "your_password"
}
```

When dealing with OAuth 2.0 and sending data in URL-encoded format as a **string**, you can provide values in the **Refresh Token Endpoint Body** field using the following format:

```
"client_id=your_client_id&client_secret=your_client_secret&grant_type=refresh_token&refresh_to
ken=your_refresh_token"
```

Note with **string** in **Refresh Token Endpoint Body** it will not support special characters like **%, ^, *** etc.

For other Authorization like AWS Signature, NTLM and Custom Signature please refer the Pre-Request Script sections for examples.

Time Parameters

The attributes under Time Parameters of Data Source API are -

- Today Format
- Yesterday Format
- Rolling Time Format
- Rolling Frequency
- Time Offset Value

This parameters are needed when user does not have any means of retrieving data from a API with a given time window. Below are the cases where API architecture have shortfall in retrieving data

User needs to retrieve data between two time window. Example : From current timestamp to last 1 hour or last 15 minutes etc.

User needs to retrieve data from yesterday transaction only and the API does not provide any means to filter the yesterday's transactions dynamically

User needs to retrieve the data from today transaction only and the API does not provide any means to filter the today's transactions dynamically

User needs to retrieve the data by adjusting the Time Zone with the dataset time based attributes like create_date, update_date, invoice_date, due_date etc.

There are four reserved keywords which can be used in any API request to filter the time based data. These keywords can be used in API params, headers, JSON body, pre-request script etc. of any API request call

These four keywords are : `today` , `yesterday` , `min_time` , `max_time`

Today Format - A date format that returns the today's date. `%Y-%m-%dT00:00:01Z` . Use the keyword `today` to use with any date field in dataset. For example: `invoice_date = {%today%}` then it will filter all transaction or records by today's date

Yesterday Format - A date format that returns previous days's date. `%Y-%m-%dT00:00:01Z`. Use the keyword `yesterday` to use with any date field in dataset. For example: `invoice_date =`

{%yesterday%} then it will filter all transaction or records by yesterday's date

Rolling Time Format - A date format that matches with the date format of the date attribute which user needs to filter data based on time window. For example, user needs to filter and get the API response with all the invoices with `invoice_date` of past 15 minutes. So for such case the user will input the date format `%Y-%m-%dT%H:%M:00.000` which is equivalent to date format of `invoice_date`

Use keywords `min_time` and `max_time` to define the time window of any API request. For example filter the invoice based on `invoice_date` as

```
invoice_date={%min_time%}&invoice_date={%max_time%}
```

GET

```
https://example.com/api/orders?start_date={%min_time%}&end_date={%max_time%}
```

Rolling Frequency - The time window in seconds between the current timestamp and frequency input by user. Example : If `900` seconds is given then it will look for Rolling 15 minutes data based on `invoice_date`

Time Offset Value -

A `Time Offset Value` typically refers to a numerical representation of the time difference between two points in time, often expressed in hours, minutes, and seconds. In API data source of eZintegrations, it's in `seconds`. This value is used to adjust or compare time between different time zones, measure time intervals, or calculate the time elapsed between two events.

Time offset values are essential for tasks such as time zone conversions, scheduling, and calculating the duration between two events, especially when those events occur in different time zones or involve time intervals. Time offset values are usually added to or subtracted from a reference time to obtain the adjusted time. For example, if you have a time in New York (EST) and want to convert it to Tokyo time (JST), you would apply a time offset of +9 hours (for the time difference between EST and JST).

Example 1 : Use of `today` and `yesterday` in Endpoint JSON Body of a API request

```
"query":"SELECT * FROM table_name WHERE column_name = '{%today%}' OR column_name = '{%yesterday%}'"
```

Example 2 : Use of `min_time` and `max_time` in Endpoint JSON body of a API request

```
"query":"SELECT * FROM table_name WHERE column_name BETWEEN '{%min_time%}' AND '{%max_time%}'"
```


Numeric Parameters

Numeric Parameters of API Data Source has two attributes

Minimum Number

Maximum Number.

Minimum Number is to define the start number of records while flushing response from API request

Maximum Number is to define the maximum number of records in a given API request

Example 1 : In Oracle Netsuite REST API architecture the `Minimum Number` is 0 and `Maximum Number` is 1000 as it flushes 1000 records per request

Example 2 : In another API whose API response is like below where per page it is having 6 records. Here the `Minimum Number` is 1 and `Maximum Number` is 6

```
ROOT
├── page: 1
├── per_page: 6
├── total: 12
├── total_pages: 2
├── data: [Array]
│   ├── [0]: [Object]
│   ├── [1]: [Object]
│   ├── [2]: [Object]
│   ├── [3]: [Object]
│   ├── [4]: [Object]
│   └── [5]: [Object]
└── support: [Object]
```

There are two reserved keywords to use the Minimum Number and Maximum Number. Those are `min_num` and `max_num` respectively. These keywords can be used in API params, headers, JSON body, pre-request script etc. of any API request call, wherever there is need to control the flow of API response

Example : GET

`https://example.com/api/users?page={%min_num%}&per_page={%max_num%}`

page	{%min_num%}	Description	+ -
------	-------------	-------------	-----

per_page	{%max_num%}	Description	+ -
----------	-------------	-------------	-----

Pagination

Pagination returns the pages of response.

There are 6 types of Pagination in API Data Source of eZintegrations as Out of the box pagination settings. These are industry standard API architecture based pagination

- Next URL Pagination
- Offset Pagination
- Total Page Count
- Pagination with Body
- Cursor Pagination
- Encoded Next Token

Next URL Pagination

This pagination will apply when we have URL of next page in our API response. For example we have next link in this key `@odata.nextLink`

```
"@odata.nextLink": "https://graph.microsoft.com/v1.0/users?$skiptoken  
=RFNwdAIAAQAAABY6cGFua3VyQGJpemRhdGEzNjAuY29tKVVzZXJfZmUwNWE5NmUtYzhiNi00Zjh  
hLWJjNWUtZjllYTNhYzh0GE5uQAAAAAAAAAAAAA",
```

Select Next URL pagination from the dropdown and fill these values there.

< **cript** Environment Variable **Pagination** Numeric Params Time Params >

Select Type
Next URL Pagination ▼

User Key*
@odata.nextLink

Data Collection Key*
['value']

Another Response Parameter
['value']

- In the above example `@odata.nextLink` key helps to identify upto which page the pagination will be completed , so it will come in `User Key`
- `Data Collection Key` holds the data from the API and we will get the response in that key for example here it is `['value']`
- `Another Response Parameter` key will hold the response of the next url page, This value needs to same as that of `Data Collection Key`.

Typically being used in Microsoft Graph API, Dropbox, Box Cloud

Offset Pagination

This pagination we will apply when we have `"hasMore": true` in our API response.

```
"count": 1000,
"hasMore": true,
"items": [
```

Select offset pagination from the dropdown and fill these values there.

< :-request Script
Environment Variable
Pagination
Numeric Params
Time >

Select Type

Offset Pagination
▼

User Key*

hasMore

Key Name to Update*

offset

Data Collection Key*

['items']

- In the above example `hasMore` key helps to identify upto which page the pagination will be completed , so it will come in `User Key`
- As we are using `Offset pagination` so `Key Name to Update` will be `offset`
- `Data Collection Key` holds the data from the API and we will get the response in that key for example here it is `['items']`.

Typically being used in Oracle Netsuite, Oracle Fusion Cloud Applications etc.

Total Page Count

This pagination will apply when we have `total_pages` key in our API response. For example we have total page Count in this key `total_pages`

Example 1: When we have the user key in root keys

```
"page": 1,  
"per_page": 6,  
"total": 12,  
"total_pages": 2,
```

Select Total Page Count from the dropdown and fill these values there.

< cript Environment Variable **Pagination** Numeric Params Time Params >

Select Type Total Page Count ▾	User Key* total_pages
	Key Name to Update* page
	Data Collection Key* ['data']

- In the above example `total_pages` key helps to identify upto which page the pagination will be completed , so it will come in `User Key`
- As we will be updating `page` value for going to next page data so `Key Name to Update` will be `page`
- `Data Collection Key` holds the data from the API and we will get the response in that key for example here it is `['data']`.

Example 2: When we have the user key in nested form.

```
],
"meta": {
  "pagination": {
    "total": 22817,
    "count": 250,
    "per_page": 250,
    "current_page": 1,
    "total_pages": 92,
    "links": {
      "previous": "",
      "current": "?page=1&limit=250",
      "next": "?page=2&limit=250"
    }
  }
}
```

Select Total Page Count from the dropdown and fill these values there.

< Headers Body Pre-request Script Environment Variable Pagination >

Select Type
Total Page Count ▼

User Key*
['meta']['pagination']['total_pages']

Key Name to Update*
page

Data Collection Key*
['data']

- In the above example `total_pages` key helps to identify upto which page the pagination will be completed , so it will come in `User Key` , as we can see in the example data `total_page` key is deep nested so we will pass the `User Key` as above.
- As we will be updating `page` value for going to next page data so `Key Name to Update` will be `page`
- `Data Collection Key` holds the data from the API and we will get the response in that key for example here it is `['data']` .

Typically being used in all eCommerce Applications

Pagination with Body

This pagination will apply when we have `nextPageToken` key in our API response. For example we have next page token in this key `nextPageToken`

```

    ],
    "isDataGolden": true
  },
  "nextPageToken": "501"

```

Select Pagination with Body from the dropdown and fill these values there.

< Pre-request Script
Environment Variable
Pagination
Numeric Params
Tit >

Select Type

Pagination with Body
▼

User Key*

nextPageToken

Key Name to Update*

pageToken

Data Collection Key*

['reports'][0]['data']['rows']

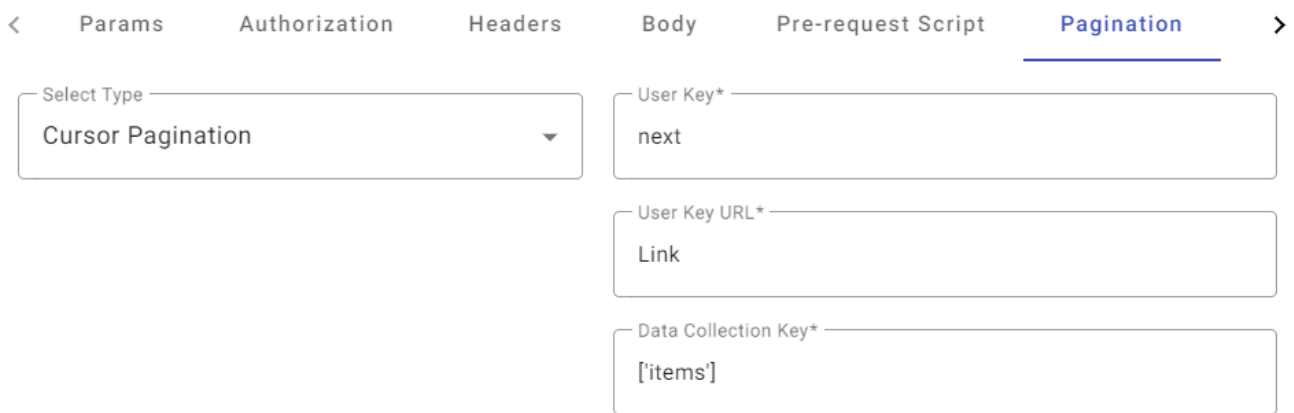
- In the above example `nextPageToken` key helps to identify upto which page the pagination will be completed , so it will come in `User Key`

- As we are getting a token from this pagination so `Key Name to Update` will be `pageToken`
- `Data Collection Key` holds the data from the API and we will get the response in that key for example here it is `['reports'][0]['data']['rows']`.

Typically being used in all Google Products like Google Analytics, Google Ads etc.

Cursor Pagination

This type of pagination usually sends a cursor in response header when a API is requested. This is typically used in User Interface based APIs. It provides a link for the next page but this comes in header response and not in the dataset



The screenshot shows a configuration interface for pagination. It has a horizontal menu with tabs: Params, Authorization, Headers, Body, Pre-request Script, and Pagination. The Pagination tab is selected and underlined. Below the tabs, there are four input fields:

- Select Type:** A dropdown menu with "Cursor Pagination" selected.
- User Key*:** A text input field containing "next".
- User Key URL*:** A text input field containing "Link".
- Data Collection Key*:** A text input field containing "['items']".

Typically being used in eCommerce Apps like Shopify, WooCommerce etc.

Encoded Next Token

This type of pagination is similar to Next URL Pagination but here the `NextToken` key is encoded

< Params Authorization Headers Body Pre-request Script Pagination >

Select Type
Encoded Next Token ▼

User Key*
NextToken

Data Collection Key*
[payload]['Orders']

Very rarely used in API but one such example is Amazon Seller Central APIs

Environment Settings

In eZintegrations, environment settings are used to store and manage dynamic values that can be reused across multiple requests and collections. These settings are helpful for streamlining your API testing and ensuring consistency in your requests. Environment settings can be categorized into different types, such as Parameters, Authorization, and Endpoint URL, based on their use. Here are some examples:

Parameters Environment settings:

Example: Let's say you have an environment variable called `apiKey`, which stores an API key. You can use this variable in your requests like this:

Request URL: `https://api.example.com/data?apikey={{apiKey}}`

Request Headers: You can set a header like `Authorization: Bearer {{apiKey}}`

In this example, the `{{apiKey}}` variable is used to pass the API key in the URL and as a header value for authentication.

Authorization Environment settings:

Example: You can store credentials for Basic Authentication in environment settings. For instance, you might have a variable called `basicAuth` with the value `Base64Encode(username:password)`. You can use it as follows:

In the request, you set the Authorization header to `Basic {{basicAuth}}`.

This approach allows you to keep sensitive information secure and make it easy to update credentials if they change.

Endpoint URL Environment settings:

Example: Suppose you have an environment variable called `baseUrl`, which represents the base URL for your API. You can use it in requests as follows:

Request URL: `{{baseUrl}}/endpoint`

By setting the `baseUrl` variable, you can easily switch between different environments (e.g., development, staging, production) without modifying all your requests.

Here's how you can create and manage environment settings in eZintegrations:

- Open eZintegrations and navigate to Click Add Integration Bridge and select a API from Data Source and click on the + button to add "Environment Settings".
- Create a new environment or select an existing one.
- Add settings with their names and values, and categorize them as needed (e.g., Parameters, Authorization, Endpoint URL). In your requests, you can reference these settings using double curly braces, like `{{variableName}}`.

By using environment settings in eZintegrations, you can make your API testing more efficient, maintainable, and adaptable to different testing scenarios and environments.

You can create Environment settings in **Data Source API, Operations API, Data Target APIs and Marketplace APIs**

All your Environment Settings will be saved only when you have saved the Integration Bridge, else you will loose the Environment Settings.

A newly created Environment Settings will be retained till the time the login session is alive.

Response Params

Response parameters:

It is an API Request parameter to getting the response as per our requirement.

We can send the Response type as Text, XML, and JSON and then it will give us response on that particular response parameters request. Once we will send the response parameter type in API Request it will process by backend python API and then it will give the particular response based on response type.

These are the following options we have for response params.

Text.

XML.

JSON.

Below image is the UI Representation of Response Parmas in our eZintegration product. You can get this feature inside IB (Integration Bridge) postman view in API as a source, operation & target.

Response Params



Text



Content



JSON

Note: Text in response params is selected by default. Users can change it as per their requirements.

Pre- request Scripts

Pre-request script is a piece of code that will run before the execution of a request. Pre-request scripts gives users a chance to modify the request after variables have been resolved but before the request is made.

Example- Generating signatures for authentication

Pre-Request Script of Python

Pre-processing tasks including setting parameters, variable values, body data and headers can be performed using the pre-request script. Pre-request scripts can also be used to debug the code, for example, by logging output to the console. Additionally, we may obtain the result of the function, such as the date, time, timestamp, etc., utilising the pre-request script notion.

The code that is run prior to sending an HTTP request using an application like requests is known as the "pre-request" script in Python. Before submitting the actual request, the pre-request script is used to change the response fields or headers.

Below are the various examples of Pre-Request Scripts

Amazon SP API

Method - POST

```
import time
import datetime, hashlib, hmac
import json
access_key=' {{access_key}}'           # Provide Values
secret_key=' {{secret_key}}'         # Provide Values
host = ' {{host}}'                   # Provide Values
endpoint = ' {{hostname}}'           # Provide Values
canonical_uri = ' {{canonical_uri}}' # Provide Values
body = {{body}}                      # Provide Values
#####
#####
request_parameters =json.dumps(body)
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
```

```

datestamp = t.strftime('%Y%m%d')
method = 'POST'
service = 'execute-api'
region = 'us-east-1'
canonical_querystring = ''
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
signed_headers = 'host;x-amz-date'
payload_hash = hashlib.sha256((request_parameters).encode('utf-8')).hexdigest()
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n' +
canonical_headers + '\n' + signed_headers + '\n' + payload_hash
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' + amzdate + '\n' + credential_scope + '\n' +
hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
kDate = hmac.new(('AWS4' + secret_key).encode('utf-8'), datestamp.encode('utf-8'),
hashlib.sha256).digest()
kRegion = hmac.new(kDate, region.encode('utf-8'), hashlib.sha256).digest()
kService = hmac.new(kRegion, service.encode('utf-8'), hashlib.sha256).digest()
kSigning = hmac.new(kService, 'aws4_request'.encode('utf-8'), hashlib.sha256).digest()
signing_key = kSigning
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
hashlib.sha256).hexdigest()
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' + credential_scope +
', ' + 'SignedHeaders=' + signed_headers + ', ' + 'Signature=' + signature

```

Method - POST (with Body)

```

import time
import datetime, hashlib, hmac
import json
access_key='XXXXXXXXXXXXXXXXXXXX'
secret_key='XXXXXXXXXXXXXXXXXXXX'
method = 'POST'
service = 'execute-api'
host = 'sellingpartnerapi-na.amazon.com'
region = 'us-east-1'
endpoint = 'https://sellingpartnerapi-na.amazon.com'
body = {'reportType': 'GET_FLAT_FILE_ALL_ORDERS_DATA_BY_ORDER_DATE_GENERAL', 'dataStartTime':
' {%yesterday%}T00:00:01', 'dataEndTime': ' {%yesterday%}T23:59:59', 'marketplaceIds':
['XXXXXXXXXX']}

```

```

request_parameters =json.dumps(body)
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SΖ')
datestamp = t.strftime('%Y%m%d')
canonical_uri = '/reports/2021-06-30/reports'
canonical_querystring = ''
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
signed_headers = 'host;x-amz-date'
payload_hash = hashlib.sha256((request_parameters).encode('utf-8')).hexdigest()
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n' +
canonical_headers + '\n' + signed_headers + '\n' + payload_hash
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' + amzdate + '\n' + credential_scope + '\n' +
hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
kDate = hmac.new(('AWS4' + secret_key).encode('utf-8'), datestamp.encode('utf-8'),
hashlib.sha256).digest()
kRegion = hmac.new(kDate, region.encode('utf-8'), hashlib.sha256).digest()
kService = hmac.new(kRegion, service.encode('utf-8'), hashlib.sha256).digest()
kSigning = hmac.new(kService, 'aws4_request'.encode('utf-8'), hashlib.sha256).digest()
signing_key = kSigning
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
hashlib.sha256).hexdigest()
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' + credential_scope +
', ' + 'SignedHeaders=' + signed_headers + ', ' + 'Signature=' + signature

```

Method - GET

```

import datetime, hashlib, hmac
host = '{{host}}' # Provide Values
endpoint = '{{hostname}}' # Provide Values
access_key = '{{access_key}}' # Provide Values
secret_key = '{{secret_key}}' # Provide Values
canonical_uri = '{{canonical_uri}}' # Provide Values
#####
#####
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SΖ')
datestamp = t.strftime('%Y%m%d')
method = 'GET'

```



```

service = 'execute-api'
region = 'us-east-1'
canonical_headers = 'host:' + host + '\\n' + 'x-amz-date:' + amzdate + '\\n'
signed_headers = 'host;x-amz-date'
payload_hash = hashlib.sha256('').encode('utf-8')).hexdigest()
canonical_request = method + '\\n' + canonical_uri + '\\n' + canonical_headers + '\\n' +
signed_headers + '\\n' + payload_hash
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\\n' + amzdate + '\\n' + credential_scope + '\\n' +
hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
kDate = hmac.new(('AWS4' + secret_key).encode('utf-8'), datestamp.encode('utf-8'),
hashlib.sha256).digest()
kRegion = hmac.new(kDate, region.encode('utf-8'), hashlib.sha256).digest()
kService = hmac.new(kRegion, service.encode('utf-8'), hashlib.sha256).digest()
kSigning = hmac.new(kService, 'aws4_request'.encode('utf-8'), hashlib.sha256).digest()
signing_key = kSigning
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
hashlib.sha256).hexdigest()
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' + credential_scope +
', ' + 'SignedHeaders=' + signed_headers + ', ' + 'Signature=' + signature

```

Oracle Netsuite

Source

```

import datetime
import random
import string
import hashlib
import base64
import hmac
import urllib

oauth_consumer_id = '{{consumer_id}}' # Provide Values
oauth_consumer_key = '{{consumer_key}}' # Provide Values
oauth_consumer_secret = '{{consumer_secret}}' # Provide Values
oauth_token = '{{token}}' # Provide Values
oauth_token_secret = '{{token_secret}}' # Provide Values

#####

```

```
#####
request_method = 'POST'
url = 'https://{hostname}.suitetalk.api.netsuite.com/services/rest/query/v1/suiteql'
oauth_signature_method = 'HMAC-SHA256'
oauth_timestamp = str(int(datetime.datetime.now().timestamp()))
oauth_nonce = ''.join(random.choices(string.ascii_letters + string.digits, k = 11))
oauth_version = '1.0'
normalized_request_method = request_method.replace(' ', '')
normalized_string_url = urllib.parse.quote(url, safe = '')
normalized_params = {'oauth_consumer_key': oauth_consumer_key, 'oauth_token':
oauth_token, 'oauth_signature_method': oauth_signature_method, 'oauth_timestamp':
oauth_timestamp, 'oauth_nonce': oauth_nonce, 'oauth_version':
oauth_version, 'limit': max_num, 'offset': min_num}
sorted_params = dict(sorted(normalized_params.items()))
normalized_string_params = [k+'='+v for k,v in sorted_params.items()]
normalized_string_params = '&'.join([str(elem) for elem in normalized_string_params])
normalized_string_params.replace(' ', '')
normalized_string_params = urllib.parse.quote(normalized_string_params, safe = '')
base_string = request_method + '&' + normalized_string_url + '&' + normalized_string_params
base_string = str.encode(base_string)
signature_key = oauth_consumer_secret + '&' + oauth_token_secret
signature_key = str.encode(signature_key)
oauth_signature = hmac.new(signature_key, base_string, hashlib.sha256)
oauth_signature.hexdigest()
oauth_signature = base64.b64encode(oauth_signature.digest())
oauth_signature = oauth_signature.decode('UTF-8')
oauth_signature = urllib.parse.quote(oauth_signature, safe = '')
signature = 'OAuth
realm=" f' {oauth_consumer_id}", oauth_consumer_key=" f' {oauth_consumer_key}", oauth_token=" f' {o
auth_token}", oauth_signature_method=" f' {oauth_signature_method}", oauth_timestamp=" f' {oauth_t
imestamp}", oauth_nonce=" f' {oauth_nonce}", oauth_version=" f' {oauth_version}", oauth_signature="
 f' {oauth_signature}"'
```

Target / Operations

```
import os
import requests
import time
import hashlib
import hmac
```

```

import base64

# Set your environment variables (replace with your actual credentials)
account = '{{account}}'
consumerKey = '{{consumerKey}}'
consumerSecret = '{{consumerSecret}}'
tokenId = '{{tokenId}}'
tokenSecret = '{{tokenSecret}}'

# Generate timestamp and nonce
timestamp = str(int(time.time()))
nonce = ''.join([str(os.urandom(1)) for _ in range(11)])

# Create base string
baseString = f"{account}&{consumerKey}&{tokenId}&{nonce}&{timestamp}"

# Create key
key = f"{consumerSecret}&{tokenSecret}"

# Create signature
signature = base64.b64encode(hmac.new(key.encode('utf-8'), baseString.encode('utf-8'),
hashlib.sha256).digest()).decode('utf-8')

```

Azure Cosmos DB

Source & Target / Operations

```

from wsgiref.handlers import format_date_time
from datetime import datetime
from time import mktime
import base64
from urllib.parse import quote
import hmac
from hashlib import sha256
endpoint_url='{{hostname}}' # Provide Values
master_key = '{{master_key}}' # Provide Values
resource_type = '{{resource_type}}' # Provide Values
resource_id = '{{resource_id}}' # Provide Values
#####

```

```
#####
key = base64.b64decode(master_key)
endpoint_method = 'post'
now = datetime.now()
stamp = mktime(now.timetuple())
date = format_date_time(stamp)
text =
' {endpoint_method}\\n{resource_type}\\n{resource_id}\\n{date}\\n{other}\\n' . format(endpoint_me
thod=(endpoint_method.lower() or ''), resource_type=(resource_type.lower() or
''), resource_id=(resource_id or ''), date=date.lower(), other=' '.lower())
body = text.encode('utf-8')
digest = hmac.new(key, body, sha256).digest()
signature = base64.encodebytes(digest).decode('utf-8')
key_type = 'master'
version = '1.0'
uri = f' type={key_type}&ver={version}&sig={signature[:-1]}'
authorization = quote(uri)
```

Amazon S3

Method : PUT

Note: This pre_request_script is for loading data to Amazon S3 Bucket.

```
import hashlib
import hmac
import datetime
access_key = '{{access_key}}'          # Provide Values
secret_key = '{{secret_key}}'         # Provide Values
bucket = '{{bucket_name}}'           # Provide Values
region = '{{region}}'                 # Provide Values
payload = '{{payload}}'               # Provide Values
host = '{{host}}'                     # Provide Values
canonical_uri = '/{{canonical_uri}}'  # Provide Values
#####
#####
method = 'PUT'
amzdate = datetime.datetime.utcnow().strftime('%Y%m%dT%H%M%SZ')
datestamp = datetime.datetime.utcnow().strftime('%Y%m%d')
```

```

canonical_querystring = ''
payload_hash = hashlib.sha256(payload.encode()).hexdigest()
canonical_headers = 'host:' + host + '\\n' + 'x-amz-content-sha256:' + payload_hash + '\\n' +
'x-amz-date:' + amzdate + '\\n'
signed_headers = 'host;x-amz-content-sha256;x-amz-date'
canonical_request = method + '\\n' + canonical_uri + '\\n' + canonical_querystring + '\\n' +
canonical_headers + '\\n' + signed_headers + '\\n' +
hashlib.sha256(payload.encode()).hexdigest()
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/s3/aws4_request'
string_to_sign = algorithm + '\\n' + amzdate + '\\n' + credential_scope + '\\n' +
hashlib.sha256(canonical_request.encode()).hexdigest()
date_key = hmac.new(('AWS4' + secret_key).encode(), datestamp.encode(),
hashlib.sha256).digest()
region_key = hmac.new(date_key, region.encode(), hashlib.sha256).digest()
service_key = hmac.new(region_key, 's3'.encode(), hashlib.sha256).digest()
signing_key = hmac.new(service_key, 'aws4_request'.encode(), hashlib.sha256).digest()
signature = hmac.new(signing_key, string_to_sign.encode(), hashlib.sha256).hexdigest()
authorization_header = algorithm + ' Credential=' + access_key + '/' + credential_scope + ',
SignedHeaders=' + signed_headers + ', Signature=' + signature

```

Method : GET

Note: This pre_request_script is for retrieving data from Amazon S3 Bucket.

```

import hashlib
import hmac
import datetime

access_key = '{{access_key}}'          # Provide Values
secret_key = '{{secret_key}}'         # Provide Values
bucket = '{{bucket_name}}'           # Provide Values
region = '{{region}}'                 # Provide Values
host = '{{host}}'                     # Provide Values
canonical_uri = '/{{canonical_uri}}'  # Provide Values

#####
#####

method = 'GET'
service = 's3'
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
datestamp = t.strftime('%Y%m%d')

```

```
canonical_querystring = ''
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
signed_headers = 'host;x-amz-date'
payload_hash = hashlib.sha256('').encode('utf-8')).hexdigest()
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n' +
canonical_headers + '\n' + signed_headers + '\n' + payload_hash
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' + amzdate + '\n' + credential_scope + '\n' +
hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
date_key = hmac.new("AWS4" + secret_key).encode(), datestamp.encode(),
hashlib.sha256).digest()
region_key = hmac.new(date_key, region.encode(), hashlib.sha256).digest()
service_key = hmac.new(region_key, "s3".encode(), hashlib.sha256).digest()
signing_key = hmac.new(service_key, "aws4_request".encode(), hashlib.sha256).digest()
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
hashlib.sha256).hexdigest()
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' + credential_scope +
', ' + 'SignedHeaders=' + signed_headers + ', ' + 'Signature=' + signature
```

Data Source - Bizintel360

Data Lake

Bizintel360 Data Lake is a search engine based NO-SQL database owned by Bizdata. Users can ingest petabyte, zettabyte, yottabyte of records both structured and unstructured for Analytics, Storage, Machine Learning and deep learning.

Bizintel360 Data Lake Source is a connection pool in eZIntegrations platform to retrieve data in JSON format.

Response from Bizintel360 Data Lake source is stored in key `bizdata_dataset_response`. If you are using Single Line to Multiline Operations as a next operation in your pipeline then the `Chop key` will have value as

```
['bizdata_dataset_response']
```

Bizintel360 Data Lake Source have following Parameters :

Data Lake Version : Data Lake Version is the Data Lake Name and its version assigned to the organization by Bizdata

Index / Table Name: Index or Table name that you want to retrieve data from Data Lake. For the List of table name or index, check `Datalake` section in Bizintel360 Visualization product.

Pagination Wait Time : By default it is 2m, where m is minute. Pagination is a standard API capability. Bizintel360 Data Lake source retrieve data in paginated way. This parameter is to set as how long need to wait for next page. If the response of Table/Index is very high (having 100+ keys in a single record) then try increasing the `Pagination Wait Time`. In general `2m` is sufficient to stream data. Use this when you have high network traffic congestion.

Can also use `h` for hours and `s` for seconds

Timeout : By default it is `2m`, where m is minute. In general `2m` is high enough to get response from Bizintel360 Data Lake. Increase this when response from Bizintel360 Data Lake is slow. This can happen when the Data Lake Cluster size is small. Reach out to Bizdata support team to make a increase in cluster size of Bizintel360 Data Lake.

Can also use `h` for hours and `s` for seconds

Size : By default it is `1000`. Size is number of streaming record count from Bizintel360 Data Lake source. The source will stream the records inside pipeline in size of 1000 chunks and move to operations and finally to Data Target. This can be increased to max of 10,000 records in case of use case like one-time historical data loads.

For better performance and durability `1000` is recommended size. This size helps to realize 1000 records in target faster and makes perfection in real-time data processing.

Query : JSON Body based query to retrieve data from tables/index of Bizintel360 Data Lake.

Get all the Records from a table

This below example responds with all the records from a table. This is similar to sql ***select * from table***

```
{ "query": { "match_all": {} } }
```

Get all Records with specific columns/keys from a table

In the below example `store_number` and `customer_number` are two keys in the Data Lake table. It will respond with all the records with those keys only. This is similar to sql ***select store_number, customer_number from table***

```
{
  "_source": ["store_number", "customer_number"],
  "query": {
    "match_all": {}
  }
}
```

Get Specific column/keys from a table

In the below example `employee_id` is a key whose value is `130` and it responds with only two keys that is `employee_id` and `employee_name`. This is similar to sql ***select employee_id, employee_name from table where employee_id=130***

```
{
  "query": {
    "match": {
      "employee_id": 130
    }
  }
}
```



```
},
  "_source": {
    "includes": [ "employee_id", "employee_name" ]
  }
}
```

Get Specific Columns/keys and Filters from a table

Below example is similar to sql

select Project,title,Assigned To,Priority,Created By,createdDateTime,dueDateTime from table where Project='Project ABC' and Priority is not null and percentComplete=100

```
{"size": 50, "sort": [{}], "_source": ["Project", "title", "Assigned To", "Priority", "Created By", "createdDateTime", "dueDateTime"], "query": {"bool": {"must": [{"query_string": {"query": "*"}}, {"query_string": {"query": "Project: \"Project ABC\" AND Priority: [* TO *] AND NOT percentComplete: 100"}}, {"bool": {"should": []}}, {"must_not": []}]}}
```

Revision #18

Created 9 August 2023 11:51:41 by Bizdata Help

Updated 7 December 2023 05:51:13 by Bizdata Help

Data Source- Websocket

A communication protocol known as WebSocket allows a client and a server to have a single, persistent connection while offering full-duplex communication channels. It makes two-way communication possible in real-time. Unlike REST API, this Communication API does not require a new connection to be established for every message delivered between clients and servers. While REST API establishes distinct connections for each request-response cycle, messages can be sent and received constantly in WebSocket without interruption, once the connection is established.

Parameters used for WebSocket:

Request URL: Add the URL for your request. The URL used for WebSocket connections starts with `ws://` for unencrypted connections and `wss://` for encrypted (SSL/TLS) connections.

E.g.: Request URL: `wss://bizdatapi-sb.bizdata360.com/streaminglogs/1132`

Message: A message that needs to be sent to WebSocket API from the client. Pass the message content in string format.

E.g.: `{"message":"Hello world"}`

Params: To pass the additional information in the parameters. Add the values in key-value format along with the description, if any.

Headers: To pass additional information like authorization value in a key-value format in the headers.

Example:

Key

```
api-key
```

Value

```
{api-key value}
```

Revision #4

Created 1 December 2023 09:18:28 by Bizdata Help

Updated 7 December 2023 05:51:13 by Bizdata Help

Data Target - API

- Test the Data Target
- Pre-Request Script
- REST API Target
- Response Params

Test the Data Target

Pre-Request Script

Pre-request script is a piece of code that will run before the execution of a request. Pre-request scripts gives users a chance to modify the request after variables have been resolved but before the request is made.

Example- Generating signatures for authentication

Pre-Request Script of Python

Pre-processing tasks including setting parameters, variable values, body data and headers can be performed using the pre-request script. Pre-request scripts can also be used to debug the code, for example, by logging output to the console. Additionally, we may obtain the result of the function, such as the date, time, timestamp, etc., utilising the pre-request script notion.

The code that is run prior to sending an HTTP request using an application like requests is known as the "pre-request" script in Python. Before submitting the actual request, the pre-request script is used to change the response fields or headers.

Below are the various examples of Pre-Request Scripts

Amazon SP API

Method - POST

```
import time
import datetime, hashlib, hmac
import json
access_key=' {{access_key}}'           # Provide Values
secret_key=' {{secret_key}}'          # Provide Values
host = ' {{host}}'                    # Provide Values
endpoint = ' {{hostname}}'            # Provide Values
canonical_uri = ' {{canonical_uri}}'  # Provide Values
body = {{body}}                       # Provide Values
#####
#####
request_parameters =json.dumps(body)
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
```

```

datestamp = t.strftime('%Y%m%d')
method = 'POST'
service = 'execute-api'
region = 'us-east-1'
canonical_querystring = ''
canonical_headers = 'host:' + host + '\\n' + 'x-amz-date:' + amzdate + '\\n'
signed_headers = 'host;x-amz-date'
payload_hash = hashlib.sha256((request_parameters).encode('utf-8')).hexdigest()
canonical_request = method + '\\n' + canonical_uri + '\\n' + canonical_querystring + '\\n' +
canonical_headers + '\\n' + signed_headers + '\\n' + payload_hash
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\\n' + amzdate + '\\n' + credential_scope + '\\n' +
hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
kDate = hmac.new(('AWS4' + secret_key).encode('utf-8'), datestamp.encode('utf-8'),
hashlib.sha256).digest()
kRegion = hmac.new(kDate, region.encode('utf-8'), hashlib.sha256).digest()
kService = hmac.new(kRegion, service.encode('utf-8'), hashlib.sha256).digest()
kSigning = hmac.new(kService, 'aws4_request'.encode('utf-8'), hashlib.sha256).digest()
signing_key = kSigning
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
hashlib.sha256).hexdigest()
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' + credential_scope +
', ' + 'SignedHeaders=' + signed_headers + ', ' + 'Signature=' + signature

```

Method - POST (with Body)

```

import time
import datetime, hashlib, hmac
import json
access_key='XXXXXXXXXXXXXXXXXXXX'
secret_key='XXXXXXXXXXXXXXXXXXXX'
method = 'POST'
service = 'execute-api'
host = 'sellingpartnerapi-na.amazon.com'
region = 'us-east-1'
endpoint = 'https://sellingpartnerapi-na.amazon.com'
body = {'reportType': 'GET_FLAT_FILE_ALL_ORDERS_DATA_BY_ORDER_DATE_GENERAL', 'dataStartTime':
' {%yesterday%}T00:00:01', 'dataEndTime': ' {%yesterday%}T23:59:59', 'marketplaceIds':
['XXXXXXXXXX']}

```

```

request_parameters =json.dumps(body)
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SΖ')
datestamp = t.strftime('%Y%m%d')
canonical_uri = '/reports/2021-06-30/reports'
canonical_querystring = ''
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
signed_headers = 'host;x-amz-date'
payload_hash = hashlib.sha256((request_parameters).encode('utf-8')).hexdigest()
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n' +
canonical_headers + '\n' + signed_headers + '\n' + payload_hash
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' + amzdate + '\n' + credential_scope + '\n' +
hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
kDate = hmac.new(('AWS4' + secret_key).encode('utf-8'), datestamp.encode('utf-8'),
hashlib.sha256).digest()
kRegion = hmac.new(kDate, region.encode('utf-8'), hashlib.sha256).digest()
kService = hmac.new(kRegion, service.encode('utf-8'), hashlib.sha256).digest()
kSigning = hmac.new(kService, 'aws4_request'.encode('utf-8'), hashlib.sha256).digest()
signing_key = kSigning
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
hashlib.sha256).hexdigest()
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' + credential_scope +
', ' + 'SignedHeaders=' + signed_headers + ', ' + 'Signature=' + signature

```

Method - GET

```

import datetime, hashlib, hmac
host = '{{host}}' # Provide Values
endpoint = '{{hostname}}' # Provide Values
access_key = '{{access_key}}' # Provide Values
secret_key = '{{secret_key}}' # Provide Values
canonical_uri = '{{canonical_uri}}' # Provide Values
#####
#####
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SΖ')
datestamp = t.strftime('%Y%m%d')
method = 'GET'

```

```

service = 'execute-api'
region = 'us-east-1'
canonical_headers = 'host:' + host + '\\n' + 'x-amz-date:' + amzdate + '\\n'
signed_headers = 'host;x-amz-date'
payload_hash = hashlib.sha256('').encode('utf-8')).hexdigest()
canonical_request = method + '\\n' + canonical_uri + '\\n' + canonical_headers + '\\n' +
signed_headers + '\\n' + payload_hash
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\\n' + amzdate + '\\n' + credential_scope + '\\n' +
hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
kDate = hmac.new(('AWS4' + secret_key).encode('utf-8'), datestamp.encode('utf-8'),
hashlib.sha256).digest()
kRegion = hmac.new(kDate, region.encode('utf-8'), hashlib.sha256).digest()
kService = hmac.new(kRegion, service.encode('utf-8'), hashlib.sha256).digest()
kSigning = hmac.new(kService, 'aws4_request'.encode('utf-8'), hashlib.sha256).digest()
signing_key = kSigning
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
hashlib.sha256).hexdigest()
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' + credential_scope +
', ' + 'SignedHeaders=' + signed_headers + ', ' + 'Signature=' + signature

```

Oracle Netsuite

Source

```

import datetime
import random
import string
import hashlib
import base64
import hmac
import urllib

oauth_consumer_id = '{{consumer_id}}' # Provide Values
oauth_consumer_key = '{{consumer_key}}' # Provide Values
oauth_consumer_secret = '{{consumer_secret}}' # Provide Values
oauth_token = '{{token}}' # Provide Values
oauth_token_secret = '{{token_secret}}' # Provide Values

#####

```



```
#####
request_method = 'POST'
url = 'https://{hostname}.suitetalk.api.netsuite.com/services/rest/query/v1/suiteql'
oauth_signature_method = 'HMAC-SHA256'
oauth_timestamp = str(int(datetime.datetime.now().timestamp()))
oauth_nonce = ''.join(random.choices(string.ascii_letters + string.digits, k = 11))
oauth_version = '1.0'
normalized_request_method = request_method.replace(' ', '')
normalized_string_url = urllib.parse.quote(url, safe = '')
normalized_params = {'oauth_consumer_key': oauth_consumer_key, 'oauth_token':
oauth_token, 'oauth_signature_method': oauth_signature_method, 'oauth_timestamp':
oauth_timestamp, 'oauth_nonce': oauth_nonce, 'oauth_version':
oauth_version, 'limit': max_num, 'offset': min_num}
sorted_params = dict(sorted(normalized_params.items()))
normalized_string_params = [k+'='+v for k,v in sorted_params.items()]
normalized_string_params = '&'.join([str(elem) for elem in normalized_string_params])
normalized_string_params = urllib.parse.quote(normalized_string_params, safe = '')
base_string = request_method + '&' + normalized_string_url + '&' + normalized_string_params
base_string = str.encode(base_string)
signature_key = oauth_consumer_secret + '&' + oauth_token_secret
signature_key = str.encode(signature_key)
oauth_signature = hmac.new(signature_key, base_string, hashlib.sha256)
oauth_signature.hexdigest()
oauth_signature = base64.b64encode(oauth_signature.digest())
oauth_signature = oauth_signature.decode('UTF-8')
oauth_signature = urllib.parse.quote(oauth_signature, safe = '')
signature = 'OAuth
realm=" f' {oauth_consumer_id}", oauth_consumer_key=" f' {oauth_consumer_key}", oauth_token=" f' {o
auth_token}", oauth_signature_method=" f' {oauth_signature_method}", oauth_timestamp=" f' {oauth_t
imestamp}", oauth_nonce=" f' {oauth_nonce}", oauth_version=" f' {oauth_version}", oauth_signature="
 f' {oauth_signature}"'
```

Target / Operations

```
import os
import requests
import time
import hashlib
import hmac
```

```

import base64

# Set your environment variables (replace with your actual credentials)
account = '{{account}}'
consumerKey = '{{consumerKey}}'
consumerSecret = '{{consumerSecret}}'
tokenId = '{{tokenId}}'
tokenSecret = '{{tokenSecret}}'

# Generate timestamp and nonce
timestamp = str(int(time.time()))
nonce = ''.join([str(os.urandom(1)) for _ in range(11)])

# Create base string
baseString = f"{account}&{consumerKey}&{tokenId}&{nonce}&{timestamp}"

# Create key
key = f"{consumerSecret}&{tokenSecret}"

# Create signature
signature = base64.b64encode(hmac.new(key.encode('utf-8'), baseString.encode('utf-8'),
hashlib.sha256).digest()).decode('utf-8')

```

Azure Cosmos DB

Source & Target / Operations

```

from wsgiref.handlers import format_date_time
from datetime import datetime
from time import mktime
import base64
from urllib.parse import quote
import hmac
from hashlib import sha256
endpoint_url='{{hostname}}' # Provide Values
master_key = '{{master_key}}' # Provide Values
resource_type = '{{resource_type}}' # Provide Values
resource_id = '{{resource_id}}' # Provide Values
#####

```

```
#####
key = base64.b64decode(master_key)
endpoint_method = 'post'
now = datetime.now()
stamp = mktime(now.timetuple())
date = format_date_time(stamp)
text =
' {endpoint_method}\\n{resource_type}\\n{resource_id}\\n{date}\\n{other}\\n' . format(endpoint_me
thod=(endpoint_method.lower() or ''), resource_type=(resource_type.lower() or
''), resource_id=(resource_id or ''), date=date.lower(), other=' '.lower())
body = text.encode('utf-8')
digest = hmac.new(key, body, sha256).digest()
signature = base64.encodebytes(digest).decode('utf-8')
key_type = 'master'
version = '1.0'
uri = f' type={key_type}&ver={version}&sig={signature[:-1]}'
authorization = quote(uri)
```

Amazon S3

Method : PUT

Note: This pre_request_script is for loading data to Amazon S3 Bucket.

```
import hashlib
import hmac
import datetime
access_key = '{{access_key}}'          # Provide Values
secret_key = '{{secret_key}}'         # Provide Values
bucket = '{{bucket_name}}'           # Provide Values
region = '{{region}}'                 # Provide Values
payload = '{{payload}}'               # Provide Values
host = '{{host}}'                     # Provide Values
canonical_uri = '/{{canonical_uri}}'  # Provide Values
#####
#####
method = 'PUT'
amzdate = datetime.datetime.utcnow().strftime('%Y%m%dT%H%M%SZ')
datestamp = datetime.datetime.utcnow().strftime('%Y%m%d')
```

```

canonical_querystring = ''
payload_hash = hashlib.sha256(payload.encode()).hexdigest()
canonical_headers = 'host:' + host + '\\n' + 'x-amz-content-sha256:' + payload_hash + '\\n' +
'x-amz-date:' + amzdate + '\\n'
signed_headers = 'host;x-amz-content-sha256;x-amz-date'
canonical_request = method + '\\n' + canonical_uri + '\\n' + canonical_querystring + '\\n' +
canonical_headers + '\\n' + signed_headers + '\\n' +
hashlib.sha256(payload.encode()).hexdigest()
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/s3/aws4_request'
string_to_sign = algorithm + '\\n' + amzdate + '\\n' + credential_scope + '\\n' +
hashlib.sha256(canonical_request.encode()).hexdigest()
date_key = hmac.new(('AWS4' + secret_key).encode(), datestamp.encode(),
hashlib.sha256).digest()
region_key = hmac.new(date_key, region.encode(), hashlib.sha256).digest()
service_key = hmac.new(region_key, 's3'.encode(), hashlib.sha256).digest()
signing_key = hmac.new(service_key, 'aws4_request'.encode(), hashlib.sha256).digest()
signature = hmac.new(signing_key, string_to_sign.encode(), hashlib.sha256).hexdigest()
authorization_header = algorithm + ' Credential=' + access_key + '/' + credential_scope + ',
SignedHeaders=' + signed_headers + ', Signature=' + signature

```

Method : GET

Note: This pre_request_script is for retrieving data from Amazon S3 Bucket.

```

import hashlib
import hmac
import datetime

access_key = '{{access_key}}'          # Provide Values
secret_key = '{{secret_key}}'         # Provide Values
bucket = '{{bucket_name}}'           # Provide Values
region = '{{region}}'                 # Provide Values
host = '{{host}}'                     # Provide Values
canonical_uri = '/{{canonical_uri}}'  # Provide Values

#####
#####

method = 'GET'
service = 's3'
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SΖ')

```

```
datestamp = t.strftime('%Y%m%d')
canonical_querystring = ''
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
signed_headers = 'host;x-amz-date'
payload_hash = hashlib.sha256('').encode('utf-8')).hexdigest()
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n' +
canonical_headers + '\n' + signed_headers + '\n' + payload_hash
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' + amzdate + '\n' + credential_scope + '\n' +
hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
date_key = hmac.new("AWS4" + secret_key).encode(), datestamp.encode(),
hashlib.sha256).digest()
region_key = hmac.new(date_key, region.encode(), hashlib.sha256).digest()
service_key = hmac.new(region_key, "s3".encode(), hashlib.sha256).digest()
signing_key = hmac.new(service_key, "aws4_request".encode(), hashlib.sha256).digest()
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
hashlib.sha256).hexdigest()
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' + credential_scope +
', ' + 'SignedHeaders=' + signed_headers + ', ' + 'Signature=' + signature
```

REST API Target

- When working with OAuth 2.0 and sending data in URL-encoded format, you can provide values in the 'refresh token endpoint body' field in the following format :

Refresh Token Endpoint Body :

```
{
  "grant_type": "password",
  "client_id": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "client_secret": "XXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "username": "XXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "password": "XXXXXXXXXXXXXXXXXXXXXXXXXXXX"
}
```

- When dealing with OAuth 2.0 and sending data in URL-encoded format as a string, you can provide values in the 'refresh token endpoint body' field using the following format:

Refresh Token Endpoint Body :

```
"client_id=XXXXXXXXXXXXXXXXXX&client_secret=XXXXXXXXXXXX&grant_type=refresh_token&refresh_token=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

Response Params

Response parameters:

It is an API Request parameter to getting the response as per our requirement.

We can send the Response type as Text, XML, and JSON and then it will give us response on that particular response parameters request. Once we will send the response parameter type in API Request it will process by backend python API and then it will give the particular response based on response type.

These are the following options we have for response params.

Text.

XML.

JSON.

Below image is the UI Representation of Response Parmas in our eZintegration product. You can get this feature inside IB (Integration Bridge) postman view in API as a source, operation & target.

Response Params



Text



Content



JSON

Note: Text in response params is selected by default. Users can change it as per their requirements.

Data Target- Database

- Database
- Data Target - Database SQL Examples

Database

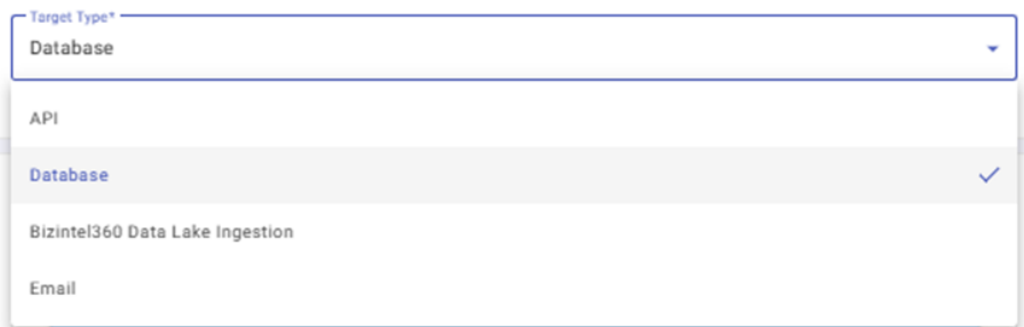
This page contains guide to keep database as a data target for the integration service. Upon successfully configuring the Data Source and Data Operation, users proceed to the Data Target stage to specify the integration bridge's intended destination, defining the necessary details for seamless integration.

To configure database as a Data Target:

Step 1: Select Target Type

To begin, users need to select Database as their Target Type

Select Target Type



The image shows a dropdown menu titled "Target Type*". The menu is open, displaying a list of options. The "Database" option is selected and highlighted in blue, with a blue checkmark to its right. Other options listed are "API", "Bizintel360 Data Lake Ingestion", and "Email".

Step 2: Select Storage Name

The user can select their preferred storage by choosing from the options listed under Select Storage Name.

Step 3: Version

Upon selecting the Storage Name, users will be presented with another dropdown menu to select the version.

Step 4: Host IP

The user needs to add the Host IP address associated with their database for establishing a seamless connection.

Step 5: Port Number

Users have the option to specify the port number pertinent to their database to establish a connection.

Step 6: Schema Name

Enter the schema name which contains the table where data is required to be inserted.

Step 7: Username

Enter the username linked to the database for authentication.

Step 8: Password

To access and authenticate the database, input the associated password.

Step 9: Order Set of Values (Tuple Key)

In tuple key, provide the key name that holds the tuple data.

Tuple key should be written inside square brackets enclosed within single quotes.

Example

[' Tuple Key']

Step 10: Batch Size

The Batch Size within the target system defines the number of streaming records moved from the source, optimizing transfer efficiency for real-time processing. A recommended value of 1000 facilitates smoother data flow between systems.

Step 11: SQL Statement

Once the necessary information required above has been furnished, users can input their SQL query in the designated section, adhering to the provided instructions.



The screenshot shows a text input field titled "SQL Statement" with a vertical scrollbar on the left side. Below the input field, there is a note: "Note: SQL Statement to be passed without any quotes, e.g: select * from source_table".

Step 12: Execute

After composing the SQL query, users can execute it by selecting the "Execute" option found at the top of the page.

Response

Users can check the results of their SQL query in the response section.

Data Target - Database SQL Examples

SQL Actions is Insert

This will go inside the SQL Statment of Database Target

```
""""Insert into table_name
(column1, column2, column3, column4) values
(?, ?, ?, ?)""", "column1", "column2", "column3", "column4"
```

SQL Actions is Update

This will go inside the SQL Statment of Database Target

```
""""UPDATE table_name SET PICKED_STATUS = 'Y',
column_name = systimestamp where column1= ? and column2= ? and column3= ?
and column4= ?""",
"column1", "column2", "column3", "column4"
```

SQL Actions is PLSQL Procedure with tuple

This will go inside the SQL Statment of Database Target

```
""""DECLARE
    column_name    VARCHAR2 (4000);
BEGIN
apps. xxxxxxxx.insert_prc(
            column_1 => ?
            ,column_2 => ?
            ,column_3 => ?
        );
EXCEPTION WHEN OTHERS THEN
    OUTPUT.PUT_LINE (' xxxxxx ' || xxxxx);
END;""", "column_1", "column_2", "column_3"
```

SQL Actions are not Insert, Update like Delete or Running a PLSQL Procedure

This will go inside the SQL Statment of Database Target

The field **Order Set of Values (Tuple Key)** should be empty in the Database Target

```
""""DECLARE
    column_name varchar2(1000); BEGIN
END;""""
```

SQL Action is Upsert or Merge

Below is the example of Merge query in Oracle Database . Insert and Update at same time in ORDERS_TEST table

t is a alias for target table

s is a alias for source table

This will go inside the SQL Statment of Database Target

```
""""MERGE INTO ORDERS_TEST t
    USING (
        SELECT
            ? AS AMAZON_ORDER_ID,
            TO_DATE(?, 'YYYY-MM-DD') AS LAST_SHIP_DATE,
            ? AS ORDER_TYPE,
            TO_DATE(?, 'YYYY-MM-DD') AS PURCHASE_DATE,
            ? AS BUYER_EMAIL,
            TO_DATE(?, 'YYYY-MM-DD') AS AMZ_LAST_UPDATE_DATE,
            ? AS IS_REPLACEMENT_ORDER,
            ? AS NUM_ITEMS_SHIPPED,
            ? AS SHIPMENT_SERVICE_LEVEL,
            ? AS ORDER_STATUS,
            ? AS SALES_CHANNEL,
            ? AS IS_BUSINESS_ORDER,
            ? AS NUM_ITEMS_UNSHIPPED,
            ? AS GLOBAL_EXPRESS_ENABLED,
            ? AS IS_SOLDBY_AB,
            ? AS IS_PREMIUM_ORDER,
            ? AS ORDER_TOTAL_AMOUNT,
```

```

? AS ORDER_TOTAL_CURRENCY,
TO_DATE(?, 'YYYY-MM-DD') AS EARLIEST_SHIP_DATE,
? AS MARKETPLACE_ID,
? AS FULFILLMENT_CHANNEL,
? AS PAYMENT_METHOD,
? AS SHIPPING_CITY,
? AS SHIPPING_POSTAL,
? AS SHIPPING_STATE,
? AS SHIPPING_COUNTRY,
? AS IS_ISPU,
? AS IS_PRIME,
? AS SELLER_ORDER_ID,
? AS SHIPMENT_SERVICE_CATEGORY,
? AS NEXTTOKEN
FROM dual
) s
ON (t.AMAZON_ORDER_ID = s.AMAZON_ORDER_ID)
WHEN MATCHED THEN
UPDATE SET
t.LAST_SHIP_DATE = s.LAST_SHIP_DATE,
t.ORDER_TYPE = s.ORDER_TYPE,
t.PURCHASE_DATE = s.PURCHASE_DATE,
t.BUYER_EMAIL = s.BUYER_EMAIL,
t.AMZ_LAST_UPDATE_DATE = s.AMZ_LAST_UPDATE_DATE,
t.IS_REPLACEMENT_ORDER = s.IS_REPLACEMENT_ORDER,
t.NUM_ITEMS_SHIPPED = s.NUM_ITEMS_SHIPPED,
t.SHIPMENT_SERVICE_LEVEL = s.SHIPMENT_SERVICE_LEVEL,
t.ORDER_STATUS = s.ORDER_STATUS,
t.SALES_CHANNEL = s.SALES_CHANNEL,
t.IS_BUSINESS_ORDER = s.IS_BUSINESS_ORDER,
t.NUM_ITEMS_UNSHIPPED = s.NUM_ITEMS_UNSHIPPED,
t.GLOBAL_EXPRESS_ENABLED = s.GLOBAL_EXPRESS_ENABLED,
t.IS_SOLDBY_AB = s.IS_SOLDBY_AB,
t.IS_PREMIUM_ORDER = s.IS_PREMIUM_ORDER,
t.ORDER_TOTAL_AMOUNT = s.ORDER_TOTAL_AMOUNT,
t.ORDER_TOTAL_CURRENCY = s.ORDER_TOTAL_CURRENCY,
t.EARLIEST_SHIP_DATE = s.EARLIEST_SHIP_DATE,
t.MARKETPLACE_ID = s.MARKETPLACE_ID,
t.FULFILLMENT_CHANNEL = s.FULFILLMENT_CHANNEL,
t.PAYMENT_METHOD = s.PAYMENT_METHOD,

```

```
t. SHIPPING_CITY = s. SHIPPING_CITY,  
t. SHIPPING_POSTAL = s. SHIPPING_POSTAL,  
t. SHIPPING_STATE = s. SHIPPING_STATE,  
t. SHIPPING_COUNTRY = s. SHIPPING_COUNTRY,  
t. IS_ISPU = s. IS_ISPU,  
t. IS_PRIME = s. IS_PRIME,  
t. SELLER_ORDER_ID = s. SELLER_ORDER_ID,  
t. SHIPMENT_SERVICE_CATEGORY = s. SHIPMENT_SERVICE_CATEGORY,  
t. NEXTTOKEN = s. NEXTTOKEN
```

WHEN NOT MATCHED THEN

```
INSERT (  
    AMAZON_ORDER_ID,  
    LAST_SHIP_DATE,  
    ORDER_TYPE,  
    PURCHASE_DATE,  
    BUYER_EMAIL,  
    AMZ_LAST_UPDATE_DATE,  
    IS_REPLACEMENT_ORDER,  
    NUM_ITEMS_SHIPPED,  
    SHIPMENT_SERVICE_LEVEL,  
    ORDER_STATUS,  
    SALES_CHANNEL,  
    IS_BUSINESS_ORDER,  
    NUM_ITEMS_UNSHIPPED,  
    GLOBAL_EXPRESS_ENABLED,  
    IS_SOLDBY_AB,  
    IS_PREMIUM_ORDER,  
    ORDER_TOTAL_AMOUNT,  
    ORDER_TOTAL_CURRENCY,  
    EARLIEST_SHIP_DATE,  
    MARKETPLACE_ID,  
    FULFILLMENT_CHANNEL,  
    PAYMENT_METHOD,  
    SHIPPING_CITY,  
    SHIPPING_POSTAL,  
    SHIPPING_STATE,  
    SHIPPING_COUNTRY,  
    IS_ISPU,  
    IS_PRIME,  
    SELLER_ORDER_ID,
```

```
SHIPMENT_SERVICE_CATEGORY,  
NEXTTOKEN  
) VALUES (  
  s. AMAZON_ORDER_ID,  
  s. LAST_SHIP_DATE,  
  s. ORDER_TYPE,  
  s. PURCHASE_DATE,  
  s. BUYER_EMAIL,  
  s. AMZ_LAST_UPDATE_DATE,  
  s. IS_REPLACEMENT_ORDER,  
  s. NUM_ITEMS_SHIPPED,  
  s. SHIPMENT_SERVICE_LEVEL,  
  s. ORDER_STATUS,  
  s. SALES_CHANNEL,  
  s. IS_BUSINESS_ORDER,  
  s. NUM_ITEMS_UNSHIPPED,  
  s. GLOBAL_EXPRESS_ENABLED,  
  s. IS_SOLDBY_AB,  
  s. IS_PREMIUM_ORDER,  
  s. ORDER_TOTAL_AMOUNT,  
  s. ORDER_TOTAL_CURRENCY,  
  s. EARLIEST_SHIP_DATE,  
  s. MARKETPLACE_ID,  
  s. FULFILLMENT_CHANNEL,  
  s. PAYMENT_METHOD,  
  s. SHIPPING_CITY,  
  s. SHIPPING_POSTAL,  
  s. SHIPPING_STATE,  
  s. SHIPPING_COUNTRY,  
  s. IS_ISPU,  
  s. IS_PRIME,  
  s. SELLER_ORDER_ID,  
  s. SHIPMENT_SERVICE_CATEGORY,  
  s. NEXTTOKEN  
)
```

```
""
```

Database Target Troubleshoot

Not able to update or truncate the records in RDBMS tables like Oracle, MSSQL etc.

Below is the example as how you can overcome the update issue in Oracle Database.

Assume a user **X** is updating a record in an application with a username **appuser** and at the same time user **Y** is also updating the same record by using the same username as **appuser**. In such case there will be two session for making update. At this stage the Oracle database will go into locking mode and you will get error like below and the update will not happen.

```
ORA-00060: deadlock detected while waiting for resource
```

If one of the user commits, then the lock will get open and then update will happen. In many cases the sessions get locks when user uses application like Oracle sql developer , TOAD or DBeaver.

To check is the session is open and locked, use the below sql. At any given time the below query response should be of 0 records. If there are 0 records then the update statement and truncate statement will work perfectly

```
SELECT s.sid, s.serial#, s.username, s.program, s.machine
FROM v$session s
WHERE s.sid IN (
  SELECT DISTINCT l.sid
  FROM v$lock l
  JOIN dba_objects o ON l.id1 = o.OBJECT_ID
  WHERE o.OBJECT_NAME = 'YOUR_TABLE_NAME' AND o.OBJECT_TYPE = 'TABLE'
);
```

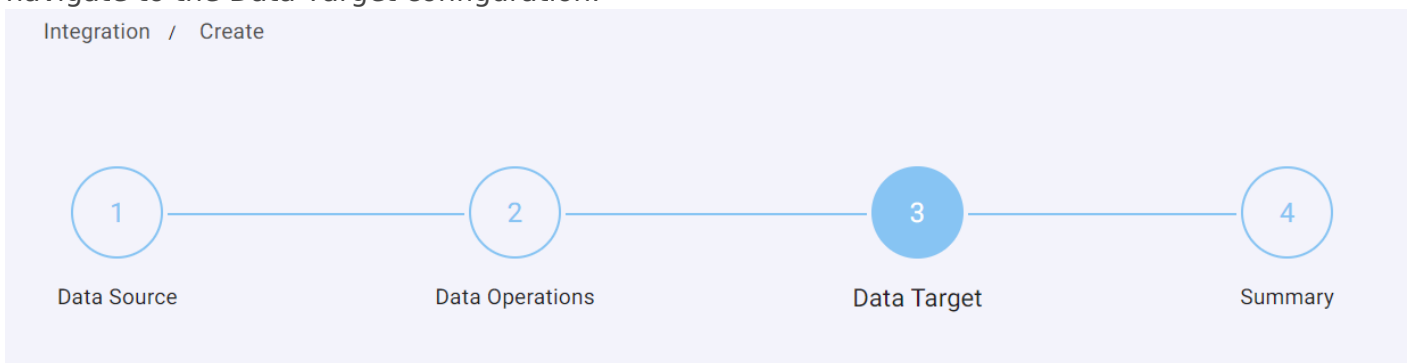
To kill the user session, use the below sql

```
--1551 is sid and 4184 is serial#
ALTER SYSTEM KILL SESSION '1551,4184';
commit;
```


Data Target- Bizintel360 Datalake Ingestion

Data lake ingestion as Data target allows you to efficiently transfer and store data from various sources directly into the Bizintel360 Datalake.

After configuring Data Source and Data operations for the integration bridge configuration, we navigate to the Data Target configuration.

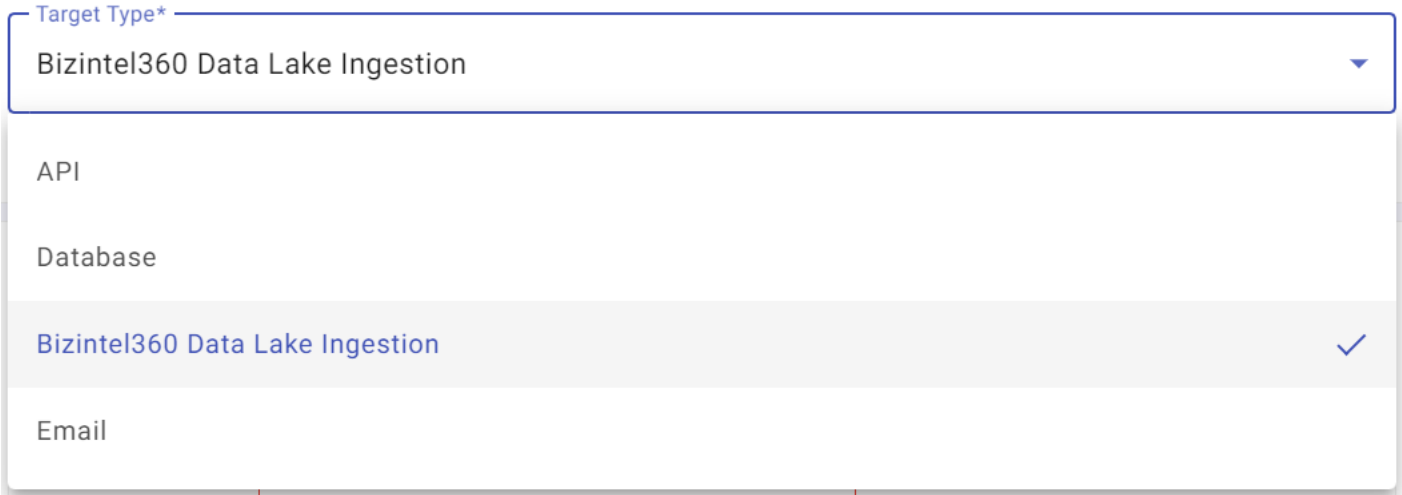


Steps to Configure

Step 1: Select Target Type

Select "Bizintel360 Data Lake Ingestion" in Select Target type option from Target Type drop down list as shown below

Select Target Type



Target Type*

Bizintel360 Data Lake Ingestion

API

Database

Bizintel360 Data Lake Ingestion

Email

Step 2: Select Data Lake Version

Select the required Datalake version from the drop-down menu as shown

Step 3: Add Index Name/Table Name

Write the Index or table name in which the data should be ingested.

bizintel360-

Index / Table Name*

Step 4: Select Action Type

Select the action type from the drop-down menu as per the requirement. Below mentioned is the description for all the action types available for selection:

- **Upsert-** The "Upsert" action type combines "update" and "insert" functionalities, allowing data to be updated if it exists or inserted if it doesn't, streamlining data management.
- **Update-** The "update" action type modifies existing data in the database, providing the ability to change specific values within a record.
- **Delete-** The "update" action type delete or removes specific set of data or entire record count from the datalake.
- **Create-** The "create" action type initiates the addition of new records or entities into datalake.
- **Insert-** The "insert" action type specifically adds new data into a database, appending records or entities into existing datasets.

Step 5: Insert Primary Key

Define the Primary key of the Index which will help in reducing data duplication. Primary key should be inserted in the case of UPSERT, UPDATE and DELETE.

Step 6: Select Ingestion Type

Select the ingestion type from the drop-down menu as per the requirement. Below mentioned is the description for all the ingestion types available for selection:

- Parallel Computing- Parallel computing as an ingestion type involves simultaneously processing and inputting large volumes of data across multiple computational resources for faster data intake and processing.
- Streaming Computing- Streaming computing as an ingestion type involves continuous and real-time processing of data as it flows into a system, enabling immediate analysis and action on incoming data streams.
- Bump Computing- Bump computing as an ingestion type involves ingesting data one batch at a time, based on batch defined in source bumps the data into Data Lake.

Revision #8

Created 30 November 2023 11:36:55 by Bizdata Help

Updated 7 December 2023 05:51:13 by Bizdata Help

Data Target- Email

Setting up Data Target as Email in eZintegrations:

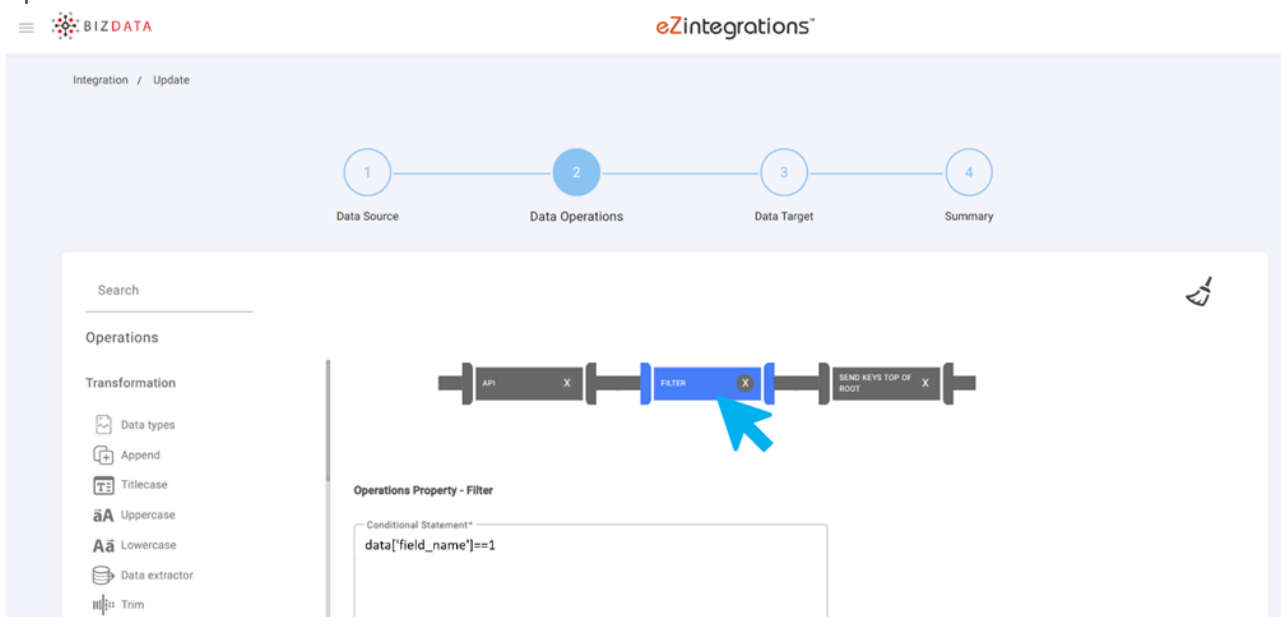
This feature enables users to receive timely email alerts indicating the success or failure of specific operations within the pipeline. This guide aims to provide a clear, step-by-step walkthrough, ensuring both new and experienced users can easily configure and manage email notifications.

Email notifications are a crucial aspect of pipeline management, ensuring that users are promptly informed about the status of their data operations. If a critical data transformation process fails, users can be immediately notified via email, allowing them to take corrective actions promptly.

Pre requisites:

- Before setting up email notifications, ensure you have the necessary credentials and permissions to access and configure the pipeline.
- Verify that your user account has the permission to enable the configuration of email alerts.
- Please make sure to add Filter operation and give the condition to trigger the Email.

Please refer: [Data Pipeline Controls | Bizdata Help \(bizdata360.com\)](https://bizdata360.com/help/Data-Pipeline-Controls) to configure filter operation.

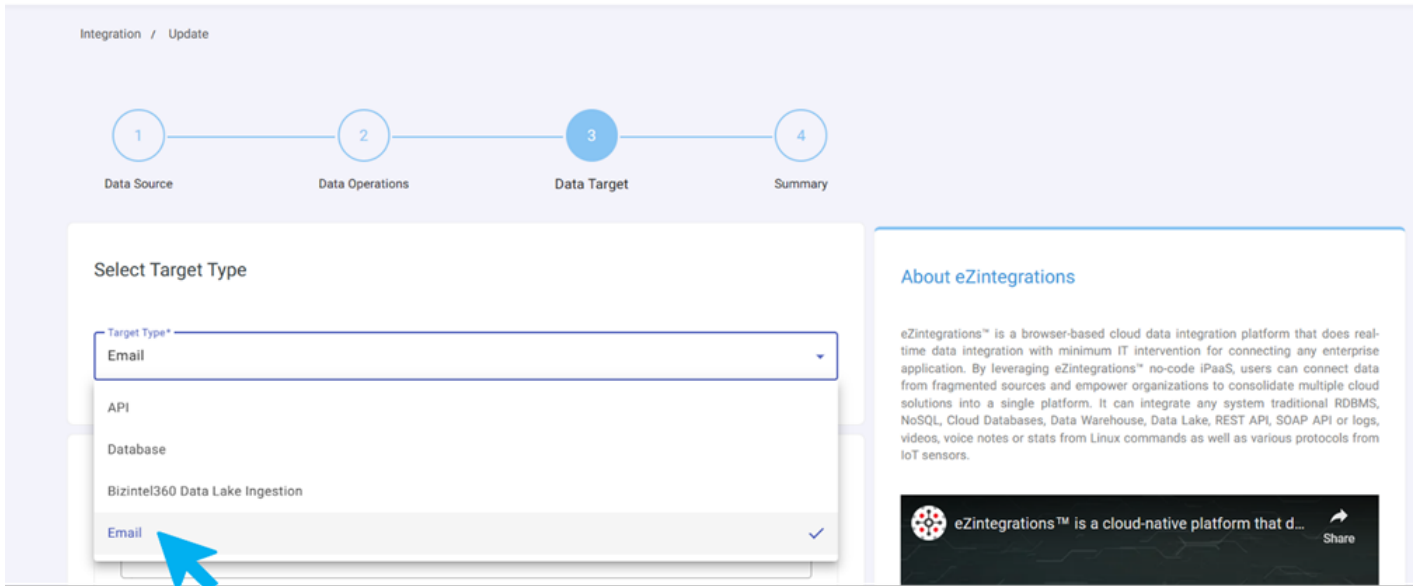


The screenshot displays the eZintegrations interface for configuring a pipeline. At the top, there is a progress bar with four steps: 1. Data Source, 2. Data Operations, 3. Data Target, and 4. Summary. The 'Data Operations' step is currently active. Below the progress bar, there is a search bar and a list of operations under the 'Transformation' category, including Data types, Append, Titlecase, Uppercase, Lowercase, Data extractor, and Trim. The main area shows a pipeline diagram with three operations: API, FILTER, and SEND KEYS TOP OF ROOT. A blue arrow points to the FILTER operation. Below the diagram, the 'Operations Property - Filter' section is visible, showing a 'Conditional Statement*' with the value 'data['field_name']==1'.

Steps to configure:

Step 1:

Select Target Type - Select target as "Email" from the dropdown.



Step 2:

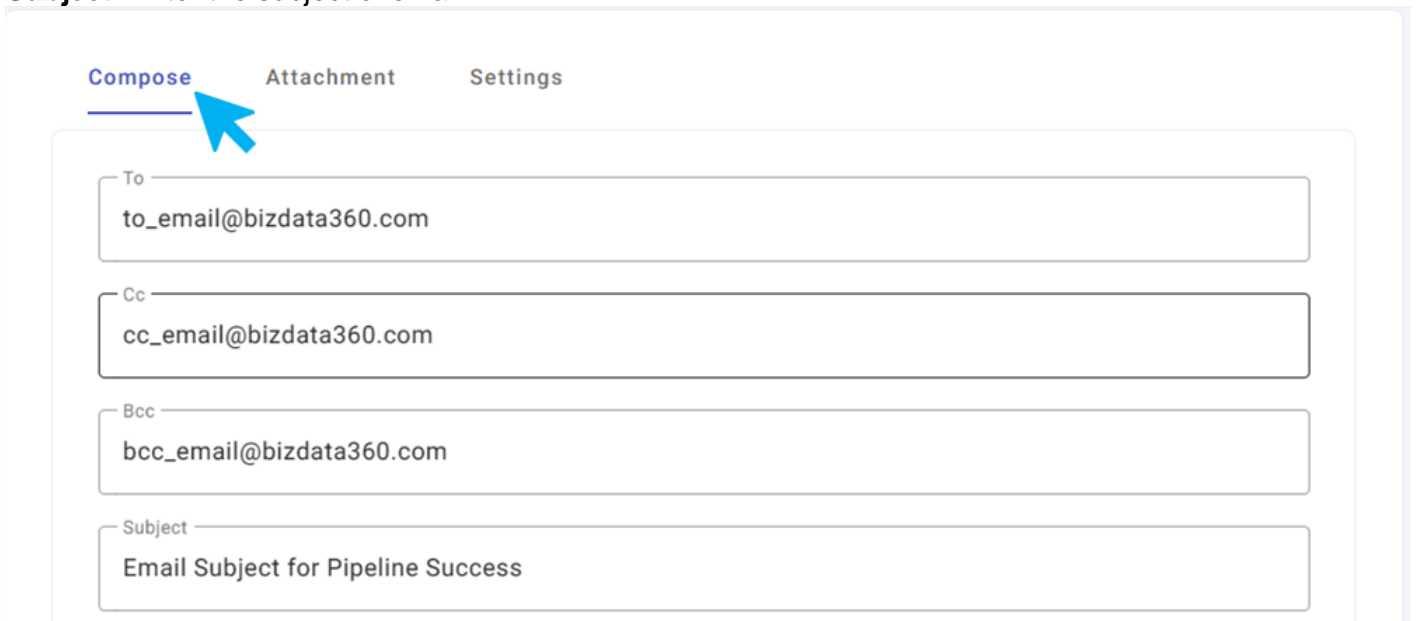
Compose - Configure Email Settings, Enter the recipient email addresses and email subject.

To- Enter the recipients email address.

Cc- Enter the recipients email address to whom carbon copy of the email is to be sent.

Bcc- Enter the recipients email address to whom carbon copy of the email is to be sent.

Subject- Enter the subject of email.



Step 3:

Specify conditions under which email alerts should be triggered (e.g., operation success, failure, specific error codes).

Subject
Email Subject for Pipeline Success

Source Key

Source **B** **I** **S** **I_x** | | | |

| Styles | Format

Date	Job ID	Error	State
{%today_date%}	{%jobId%}	{%errors%}	{%state%}

Step 4:

Attachment

Is Attachment?: In the 'Is Attachment?' parameter, a toggle option is provided for users to specify whether an attachment is included with the email. Users can select 'Yes' to indicate the presence of an attachment, or 'No' if no attachment is included with the email.

Compose **Attachment** Settings

Is Attachment ?

File name: In the 'File Name' parameter, users can input the name of the attachment. This parameter is enabled only when the user selects 'Yes' in the 'Is Attachment?' parameter, signifying that an attachment is included with the email.

Compose **Attachment** Settings

Is Attachment ?

File Name
file_name.tsv

Step 5:

Settings

From: In the 'From' parameter, users are required to input the sender's email address.

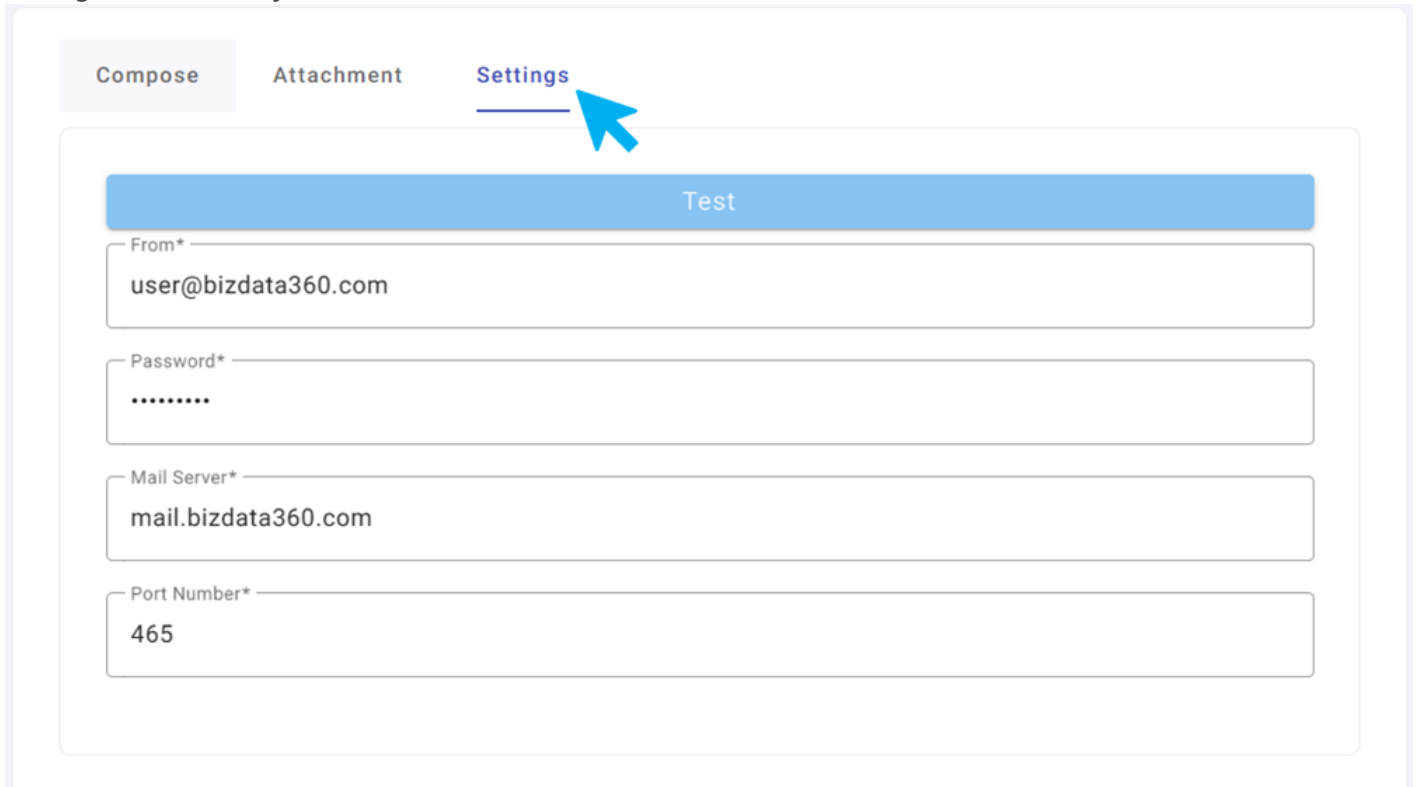
Password:

In the 'Password' parameter, users need to input the password associated with the sender's email address.

Mail Server: In the 'Mail Server' parameter, users are prompted to specify the address or hostname of the email server that will be used to send the email.

Port Number: In the 'Port Number' parameter, users are required to input the specific port number that corresponds to the chosen mail server.

Test: The 'Test' button in the UI serves as a valuable tool for evaluating the response of the configured Email Settings. Utilize the "Test" feature to send a sample email and ensure notifications are configured correctly.



The screenshot shows a web interface for configuring email settings. At the top, there are three tabs: 'Compose', 'Attachment', and 'Settings'. The 'Settings' tab is selected and highlighted with a blue arrow. Below the tabs is a large blue button labeled 'Test'. Underneath the 'Test' button are four input fields, each with a label and a value:

- From*:** user@bizdata360.com
- Password*:**
- Mail Server*:** mail.bizdata360.com
- Port Number*:** 465

Step 6:

Verify the content and format of the email message.

Managing Email notification preferences:

Edit or update email notification settings as needed. Disable email notifications for a specific data target if required.

If you need to update the recipient email address, revisit the configuration page, make the necessary changes, and click "Save."

Troubleshooting:

Troubleshoot common issues related to email notifications. Refer to the provided REST API Target documentation for advanced configurations.

If you're not receiving email notifications, check your spam folder and ensure that the email settings are correctly configured. For advanced troubleshooting, refer to the REST API Target | Bizdata Help (bizdata360.com) documentation.

Revision #1

Created 1 December 2023 09:40:33 by Bizdata Help

Updated 7 December 2023 05:51:13 by Bizdata Help

Release Notes

Release Notes (July 4, 2023)

Improvement:

- Alphanumeric values are now allowed in First name and Last name in My Profile page and Register page.

Bug Fix:

- Integration Bridge Update Issue with Database Source Type Resolved - Fixed the problem where users were unable to update the integration bridge when selecting the database as the source type, ensuring smooth editing and maintenance of integration bridges.
- Improved Usability of Key-Value Pair Addition in "params" and "header" Objects in Data Operation - Addressed the issue where the icon representing the functionality to add a key-value pair to "params" and "header" objects was relatively small and lacked clear recognition, resulting in difficulty for users to identify and utilize the feature effectively.
- Marketplace API Testing Section Issue Resolved - Fixed the problem where the API Testing section in the Marketplace was not functioning properly for certain APIs, ensuring reliable and accurate testing capabilities for all APIs within the Integration Bridge platform.
- Enhanced Display of Pre-request Script in View Mode - Addressed the issue where the pre-request script was displayed as a single line in view mode, despite users entering it as multi line text.
- Integration Bridge Creation Issue Resolved - Fixed the problem encountered when attempting to create a Integration Bridge with Data Source as Oracle Database and Data Target as Data Lake, resulting in an error message stating "Unable to create Integration Bridge."

Release Notes (June 28, 2023)

Bug Fix:

- Resolved Data Source Display Issue - Resolved the problem where multi line XML or JSON bodies were displayed as a single line in view mode when users selected the body as XML, ensuring accurate representation of the data structure.
- Integration Bridge Data Source and Target Display Issue Resolved - Fixed the issue where data target details were not being displayed when editing an integration bridge with a database as the data source type and an API as the target type. Additionally, resolved the problem where a cross icon was shown in the search by target name field, ensuring a

seamless user experience during target selection.

Release Notes (June 23, 2023)

Bug Fix:

- Resolved the issue of incorrect values of source and target key when using the Email operation.
- Resolved the issue of viewing response in UI when using API in operation.
- Resolved the issue in view section of the Integration bridge. Now, value is reflecting when given target type as database.
- Resolved the issue in view mode of the Integration bridge when user applies the dl ingestion operation. The operation is now visible in the view mode.
- New users can directly verify their account by clicking the verify link that comes in the verification mail.

Release Notes (June 16, 2023)

Bug Fix:

- In Append operation, users can now pass JSON as a value for keys.
- Resolved the issue of values getting dropped from Integration Bridge when using total_pages_pagination.
- Resolved the issue of user being able to update the integration bridge, without filling up all the mandatory fields, when the Data source type is database.
- Resolved the issue of user being able to edit the SQL statement and response field in view more mode when the source type is selected as Database.

Release Notes (June 13, 2023)

New Feature:

- Introducing Database as source option in Integration Bridge's source type Dropdown, empowering users to select databases as a data source for seamless integration.
- Introducing Encode-Decode Operation in UI Transformation, enabling users to encode or decode data within keys for enhanced data processing capabilities.

Improvement:

- Enhanced User Experience with Idle Timeout set to 15 Minutes, ensuring automatic logout from the system after a period of inactivity for improved security and resource management.

Bug Fix:

- Resolved the issue of not being able to submit the Integration Bridge with Date Analytics operation.

Release Notes (June 8, 2023)

New Feature:

- Introducing Filter operation under Pipeline Controls, Filter operation helps user enabling conditional execution of subsequent operations and streamlining data flow from source to target.

Bug Fix:

- Resolved the issue of operations getting disappeared after creating the Integration Bridge.

Release Notes (May 29, 2023)

New Feature:

- Users can now seamlessly integrate with the Bizdata Data Lake in the Integration Bridge using Bizdata360 Data Lake Ingestion option in the target type.
- Enhanced Integration Bridge with Database as Target Type, empowering users to seamlessly incorporate databases as their integration bridge targets.

Release Notes (May 24, 2023)

Improvement:

- Powerful search option specifically designed to ease out search operations in integration bridge.

Release Notes (May 12, 2023)

Bug Fix:

- Resolved Search Result Issue in Marketplace.

Release Notes (April 27, 2023)

New Feature:

- Find exactly what you need with eZintegrations marketplace search bar. With eZintegrations' user-friendly search functionality, you'll be able to quickly and easily locate the products or categories you're looking for. No more endless scrolling or navigating through confusing menus - eZintegrations search bar makes finding what you want a breeze.
- Marketplace in mobile view has an intuitive filtering system that lets users easily narrow down search results or listings, so one can find exactly what one is looking for.
- eZintegrations includes a convenient feature on the marketplace details page, allowing users to add their selection to their integration bridge with just one click and seamlessly

integrate their marketplace selection with eZintegrations' 'Add to Integration Bridge' button.

- eZintegrations includes a sort button in the marketplace list page, allowing users to sort products according to their criteria and streamline their searching process.
- Users can provide additional information or context about their integration bridge through the description option in summary page.
- Explore eZintegrations' extensive marketplace with ease thanks to eZintegrations user-friendly pagination feature. With quick and easy navigation through search results or product listings, you'll never miss out on finding your perfect item

Release Notes (April 6, 2023)

New Feature:

- Empower your team with eZintegrations flexible admin capabilities. eZintegrations allows users to easily designate new administrators, streamlining the management process and increasing productivity.
- eZintegrations platform includes a user-friendly search function for both the source and target fields in integration bridge creation and update, simplifying users integration process with searchable business object feature.

Release Notes (March 31, 2023)

New Feature:

- eZintegrations allows organization admins to easily update their profile image and organization logo empowering users to showcase their brand with customizable profile and logo feature.
- Protect your organization with eZintegrations enhanced security feature. eZintegrations now limits the ability to block users to only organization admins, ensuring that only authorized users can manage user access and prevent unauthorized access to your system.

Release Notes (March 10, 2023)

New Feature:

- User friendly platform with guided navigation available in page of the product.

Release Notes (February 24, 2023)

New Feature:

- Users can create one integration bridge for free after sign up to the free trial version.

Release Notes (February 17, 2023)

New Feature:

- Experience eZintegrations our one-year free trial version. During the trial, users will have access to all of our features and can create one integration bridge to see how our system works for them.

Release Notes (February 10, 2023)

New Feature:

- Simplify your integration setup with eZintegrations' brand-wise sorting drop-down list. eZintegrations' searchable drop-down list with brand-wise sorting makes it easy to find and select the brands you need when configuring integrations.

Release Notes (February 3, 2023)

New Feature:

- eZintegrations includes a product section, allowing users to browse and manage their product information with ease. eZintegrations API catalog has also been updated to fetch product names based on brand selection.






Revision #13

Created 11 May 2023 07:02:26 by Bizdata Help

Updated 7 December 2023 05:51:13 by Bizdata Help

Security Matrix

A	User A	Owner of the Customer Organization who signs up for eZintegrations from Sign Up Page
B	User B	The Employee of Customer Organization who has view Access to IB
C	User C	The Employee of Customer Organization who has edit Access to IB
D	User D	The Employee of Customer Organization who got delegated access from User-A

	Have access
	No access
	Access with all records all organization
	Future release
	Should have values only for customer facing business object

		A	B	C	D	
Customer Facing Business Objects	User	List	✓	×	×	✓
		Add	✓	×	×	✓
		Edit	✓	×	×	✓
		Details	✓	×	×	✓
		Total record at assigned organization	✓	×	×	✓
		User permission attribute	🔒	×	×	🔒
		Delete	×	×	×	×
	Integration Bridge	List	✓	✓	✓	✓
		Add	✓	×	✓	✓
		Edit	✓	×	✓	✓
		Copy	✓	×	✓	✓
		Details	✓	✓	✓	✓
		Start/Stop	✓	×	✓	✓
		Refresh	✓	×	✓	✓
		Status	✓	×	✓	✓
		Download	✓	×	✓	✓
		Live logs	✓	×	✓	✓
		Total record at assigned organisation	✓	×	✓	✓
		Delete	×	×	×	×
		Organization	List	✓	×	×
	Add		×	×	×	×
	Edit		✓	×	×	✓
	Details		✓	×	×	✓
	Total Record(1)		✓	×	×	✓
	Delete		×	×	×	×

Revision #3

Created 23 March 2023 09:33:09 by Bizdata Help

Updated 4 December 2023 06:36:09 by Bizdata Help