John Moody and Jason Gilmore

Building Microapps with Citrix Workspace and DreamFactory

Contents

Introduction

Increasingly, customers are looking at useful, interesting and innovative use-cases that they can integrate with Microapps in Citrix Workspace. But what if your application does not have a RESTful API? What if your application is backed by a SQL Database and users are asking for a secure API?

As with many application deployments, there's usually a batch of applications that cannot be upgraded and are critical to a company's business. These "last mile" applications need integration into your digital workspace, but the application does not provide the required interfaces to do so.

This is where DreamFactory (https://www.dreamfactory.com) comes in. Using DreamFactory, organizations can automatically create API endpoints for those systems that do not have a RESTful interface. These APIs can in turn can be leveraged by Citrix Workspace to create single use workflows, or "Microapps" for end users to consume directly from their workspace without having to open, login to and access the original application.

This guide uses a case study of a fictional company named Global Distributors. Like a lot of other companies, Global Distributors is not a cloud-first company, but rather one that depends upon a hybridized mix of SaaS, Windows, on-premises, and public cloud applications.

Prerequisites

There are a number of prerequisites that you'll ideally complete before starting this guide:

- Request a 30-day Citrix Microapps Developer instance at https://developer.cloud.com/citrixworkspace/citrix-workspace-platform
- Configure Authentication to Workspace https://developer.cloud.com/citrixworkspace/citrix-workspace-platform/build-workspace-microapp-integrations/docs/getting-started
- Request a 14-day DreamFactory instance at https://genie.dreamfactory.com

About the Authors

John Moody

John Moody is a Domain Specialist at Citrix. Having worked with Workspace technologies for over 20-years, John is passionate about removing unnecessary barriers at work so we can focus on 'real' creative work.

John believes work should be about belonging, growth and achieving our personal goals, not about getting bogged down in clutter and distractions. John is passionate about how we can use technology to help support human sustainability and our mental health. In his spare time, John is a stand-up comedian.

Jason Gilmore

Jason Gilmore is the CTO of DreamFactory Software. He has spent the past 20 years helping companies of all sizes build amazing products.

Jason is the author of nine books, including the bestselling “Beginning PHP and MySQL, Fourth Edition”, “Easy Laravel 5”, “Easy PHP Websites with the Zend Framework, Second Edition”, and “Easy Active Record for Rails Developers”.

Jason is cofounder of the wildly popular CodeMash Conference, the largest multi-day developer event in the Midwest. Away from the keyboard, you’ll often find Jason playing with his children, hunched over a chess board, or having fun with DIY electronics.

Chapter 1. Introducing Microapps

If we look at the apps we use in our personal lives on a day to day basis, the Facebooks, TikToks, SnapChats, Airbnbs, and Spotifys of the world - they don't come with user manuals and yet we instinctively know how to use them. Our teenagers didn't read a 30 page manual on uploading photos to Instagram - because there isn't one. The best consumer apps are inherently intuitive and easy to use - that's why we use them! So why aren't enterprise applications the same?

To most employees there's nothing more demoralizing than hard to use, complex enterprise systems. We've all been there - that application that your organisation has, that you only need a specific piece of information out of, or perhaps don't use that often. Research has shown that employees in an organisation will spend 20% just searching for information, and only use four or five of all available business application functions. In the midst of this, we're constantly distracted by notifications, messages and other interruptions, that we find ourselves spending the majority of our working day "context switching". When do we get the time to spend on the real work, or rather, the work you want (and get paid) to do? In the end, it's all rather frustrating and depressing.

A well designed Microapp serves one purpose, a single use-case or workflow. They should have a common consistent look and feel regardless of the underlying application(s) it is accessing. From the perspective of the end-user it should be self-explanatory and simple to use - like Instagram!

A Day in the Life With Really Hard Enterprise Apps

Global Distributors is a large distributor of many thousands of products to retail outlets around the world. Global Distributors use a legacy ERP system called Products ERP which is backed by a MySQL database and which is accessed using a legacy JAVA client. The Products ERP system can only be accessed from within the office. End-users constantly complain that it's difficult to use. On a recent staff survey, end users complained that accessing Products ERP and navigating around the complicated UI made their jobs less satisfying. Global Distributors has tried to mitigate this in the past with in-depth user guides, however the employees find these guides complicated and there is a perception that "it's a waste of valuable time".

As part of this guide, we're going to focus on three key user communities within Global Distributors that use the Products ERP system. In the real world there are likely to be many more across different business units. Once you have implemented the basic case studies in this guide using DreamFactory and Citrix Microapps, you can explore other use cases to build out your Microapps. The three use cases we're going to consider as part of this guide are:

- The Marketing Manager - responsible for all the product descriptions for each product held within Products ERP.

- Sales Representatives - within Global Distributor need to be aware of when the details of a product change, with the nature of the business - is fairly frequently.
- Product Manager - responsible for adding new products to Products ERP.

The Marketing Manager

Using a combination of screen-shots and email, the Marketing Manager emails the entire sales force when the product description for a new product is added. This provides the Sales team with additional information about each product. Despite this laborious process, the Marketing Manager finds themself dealing with queries from Sales Reps all the time as they in turn don't have the time nor skills to search the ERP for new product information.

The Sales Team

On a recent employee survey at Global Distributors, the biggest complaint from Sales Reps, particularly new hires, was that it was impossible to know every product that Global Distributors sold globally. Accessing the ERP to obtain the information was incredibly difficult; not only did they have to connect to the datacenter in head office over VPN, but to add to the misery the JAVA based ERP client only works on Windows. There's no mobile app of any sort, making the retrieval of product information while on customer calls an incredibly challenging process.

The Products Manager

When a new Product is added to Products ERP, the Products manager then notifies the Marketing Manager that a new product has been added - this is usually done via email or the in-house instant messaging system. Different Product Managers will use different methods of communication.

Microapp Use Cases

As discussed, a Microapp should be a straightforward easy to use workflow for the end-users. Our above example surfaces a number of use-cases that we can transform into useful workflows within a Microapp. There are probably several use-cases with this example, however the ones we will tackle as part of this book are:

- Allow the Marketing Manager to approve Product Descriptions without having to access the ERP and remove the need to use email to send product updates.
- Provide Sales Reps with a simple interface to search for Products without access to the ERP.
- Provide Sales Reps with a notification that Product has been updated without using email.
- Allow a Products Manager to add new products to the database.

There are other use cases that can build upon the examples above - we could add the ability to check stock/orders etc using straightforward CRUD actions against the Products database. Using Dreamfactory we could also combine other databases into the same API call or introduce more application logic using SQL Stored Procedures.

Conclusion

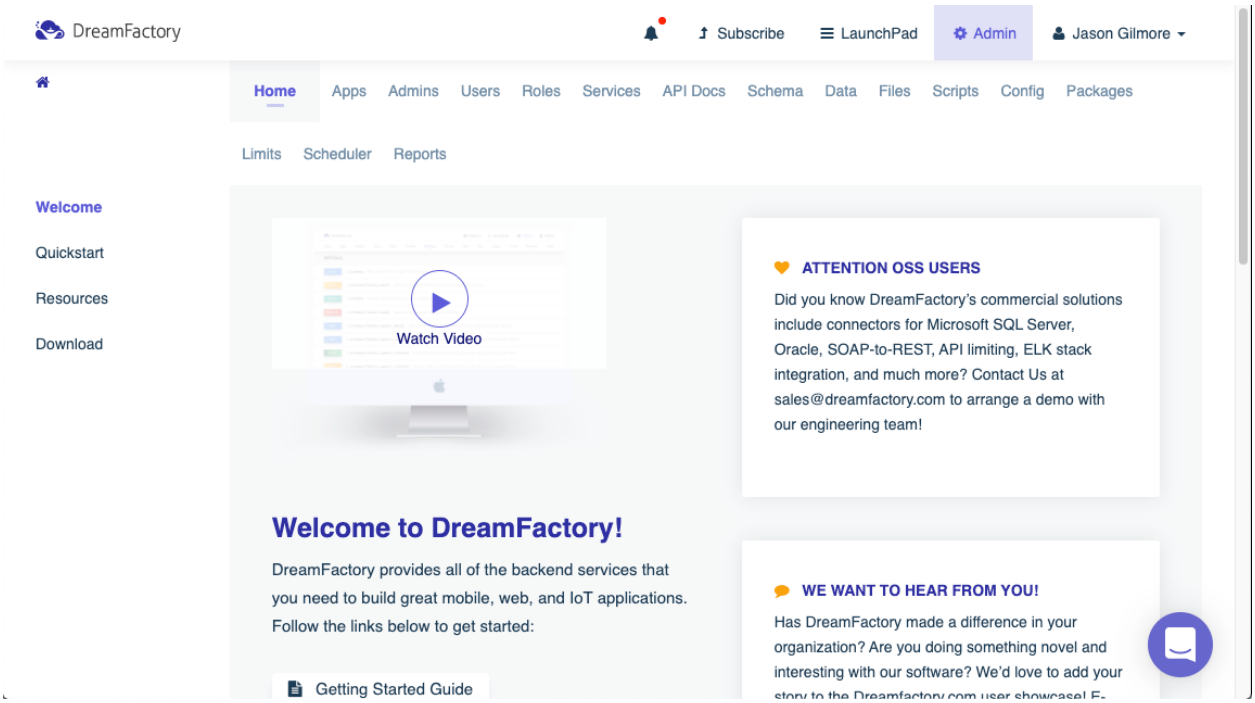
Now that you have a basic understanding of the challenges enterprise software users face, and the role microapps can play in resolving workflow issues, let's move on to creating a few example microapps! In the next chapter we'll start by generating a database-backed API using the DreamFactory platform.

Chapter 2. Generating an API with DreamFactory

We'll begin by generating a secure API which will expose the product data to Citrix Workspace. This is easily accomplished using DreamFactory's point-and-click administration interface.

Keep these credentials handy and login to your hosted DreamFactory instance. If you don't remember the URL and login credentials, they're also found in the welcome email. The URL will look something like this: https://prefix.apps.dreamfactory.com.

After logging into your DreamFactory instance, you'll be presented with the administration homepage.



The DreamFactory home page

At the top, you'll see a navigation bar containing links to the administration interface's key sections. These include:

- **Apps**: DreamFactory does not support public APIs, requiring at minimum an API key be passed for authentication purposes. These API keys are managed under this tab, and are auto-generated by DreamFactory.

- **Admins**: Manage DreamFactory administrators. In the enterprise (on-premise) DreamFactory version, a root administrator can be created which can additionally delegate limited privileges to other administrators. You'll manage administrators under this tab.
- **Users**: In addition to authenticating via API keys, DreamFactory supports a wide array of authentication identity providers including Active Directory, Okta, LDAP, OpenID Connect, OAuth, and others. Additionally, users may authenticate using Basic HTTP authentication by authenticating with their email address and password. Those accounts are managed via this tab.
- **Roles**: All DreamFactory-managed APIs are protected by a role-based access control (RBAC). Each RBAC is defined using a point-and-click interface found under this tab. RBACs give you endless control over how your APIs are accessed; for instance you can easily create a database-based read-only API, restrict access to a few specific tables, views, or stored procedures, or selectively combine CRUD (create, read, update, delete) access to various endpoints.
- **Services**: All APIs generated by and mounted to DreamFactory are managed under this tab. Here you can choose from dozens of native connectors to generate APIs from data sources such as databases and file systems, as well as mount third-party APIs and SOAP services, and even script your own APIs using programming languages such as PHP, Python, and NodeJS.
- **API Docs**: Under this tab you'll find interactive OpenAPI documentation for your APIs. In the case of auto-generated APIs, DreamFactory will automatically generate the associated documentation. For mounted and scripted APIs, you have the option to upload the companion documentation, resulting in it also being displayed under this tab.
- **Schema**: When DreamFactory first connects to a database as part of the API generation process, it will create a snapshot of the schema, including the table names, column data types, column attributes, and foreign key relationships. You can use this tab to review and edit the schemas, as well as configure virtual relationships useful for executing cross-database joins via API calls.
- **Data**: This tab provides a simple CRUD interface for data exposed through a mounted database.
- **Files**: Like the Data tab, this tab provides a simple interface for viewing files exposed through a mounted file system.
- **Scripts**: It's often useful to modify API endpoint behavior to validate input, transform responses, and perform more complicated tasks such as API chaining and API composition. You can use the Scripts tab to add this custom logic. NodeJS, PHP, and Python (versions 2 and 3) are supported. You can also manage your scripts in version control using the Bitbucket, GitLab, and GitHub integrations.
- **Config**: The Config tab contains information about the DreamFactory instance, including the version number, installation path, and IP address. You can also perform various administrative tasks here, including managing the instance's CORS configuration, the cache, and global lookup keys.

Packages: You can use the Packages tab to import and export various DreamFactory settings, including users and APIs.

- **Limits**: You'll often want to impose volume restrictions on third-parties to reduce the possibility of excessive resource usage. The Limits tab provides a simple point-and-click interface for imposing access limits on specific services, users, roles, endpoints, and even HTTP methods.

- **Scheduler**: Many DreamFactory use cases involve real-time API calls being made to applications such as dashboards and mobile apps. Sometimes though you'll want to periodically call API endpoints without requiring a human to do so. You can use the Scheduler tab to schedule API calls. It's also possible to pass payloads into scheduled calls.
- **Reports**: The Reports tab provides a historical summary of API lifecycles, logging the creation timestamp, creator username, and similar information for any subsequent service modifications or deletions. This information is stored by default in the system database however it can optionally be sent to a remote database for auditing and security purposes.

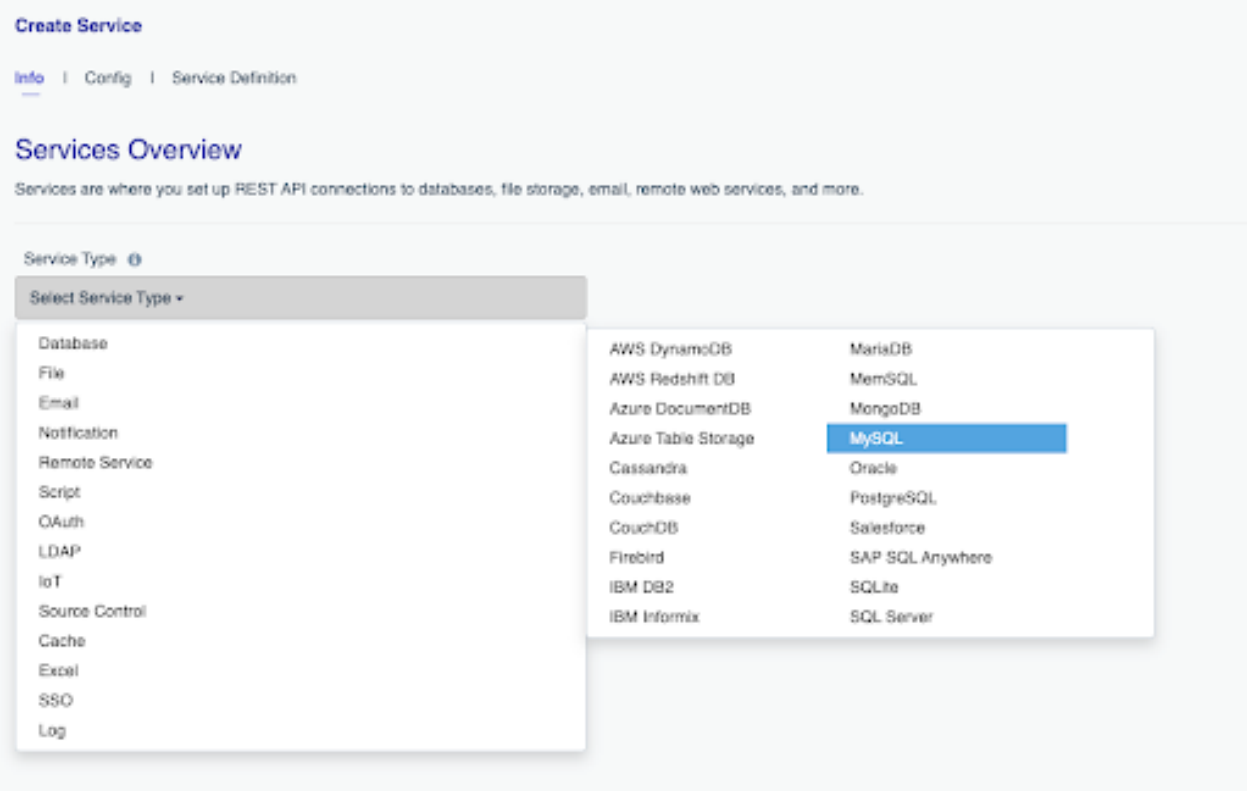
For this exercise we'll explore the Services, Roles, and Apps tabs, in that order.

Creating a Service

DreamFactory offers native API generators for a wide variety of data sources, including databases such as Microsoft SQL Server, MySQL, and Oracle, file systems such as AWS S3 and SFTP, and source control solutions such as GitHub and Bitbucket. Generating an API from these native generators requires you to supply a set of credentials which allows DreamFactory to connect to the destination data source. In the case of MySQL, you'll need access to the database's host name, in addition to account credentials.

When you signed up for the DreamFactory hosted trial, a test MySQL database containing more than five million records was generated for you. While you're not strictly required to use this test database to follow along with the examples in this book, it will help because the screenshots and other related content will reflect the data found in this example database. If you'd like to use the example database, you'll need access to the MySQL database credentials included in your DreamFactory hosted trial welcome email. These credentials include the database hostname (this will match your DreamFactory instance URL prefix), account username and password, database name, and password.

To generate the MySQL API, click on the Services tab, and then click `Create`. You'll be presented with a select box containing all supported databases. Choose the MySQL option (see below figure).



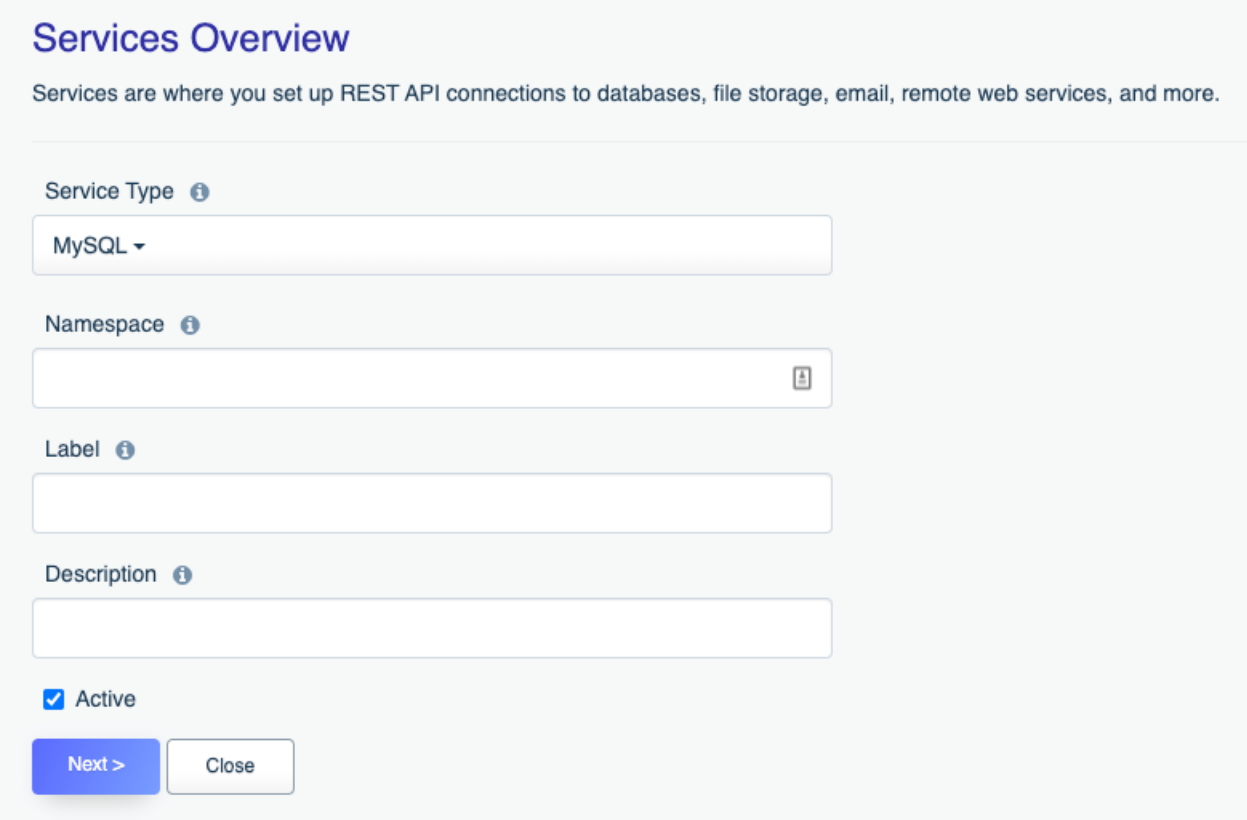
Choosing the MySQL Service Connector

In order to generate the MySQL API you'll need to supply several key pieces of information, beginning with the following items (see below screenshot):

- **Name**: The name will form part of your API URI, so you'll want to use a lowercase string with no spaces or special characters. Further, you'll want to typically choose something which allows you to easily identify the API's purpose. For instance for your MySQL-backed API you might choose a name such as mysql, corporate, or store. Keep in mind lowercasing the name is a requirement.
- **Label**: The label is used for referential purposes within the administration interface and system-related API responses. You can use something descriptive here, such as "MySQL-backed Corporate Database API".

Description: Like the label, the description is used for referential purposes within the administration interface and system-related API responses.

- **Active**: This determines whether the API is active. By default it is set to active however if you're not yet ready to begin using the API or would like to later temporarily disable it, just return to this screen and toggle the checkbox.



Configuring the MySQL Connector

Once you've completed these items, click `Next` to continue. On this screen (see below screenshot), you'll be prompted to enter the aforementioned connection details:

- **Host**: The database server's host address. This may be an IP address or domain name.
- **Port Number**: The database server's port number. For instance on MySQL this is 3306.
- **Database**: The name of the database you'd like to expose via the API.
- **Username**: The username associated with the database user account used to connect to the database.
- **Password**: The password associated with the database user account used to connect to the database.

Configuring the Authentication Credentials

Following the credential-related fields you'll find a laundry list of other configurable options. None of these are relevant to our present project so we'll forgo a lengthy description, however you can learn more about them at https://guide.dreamfactory.com/[1].

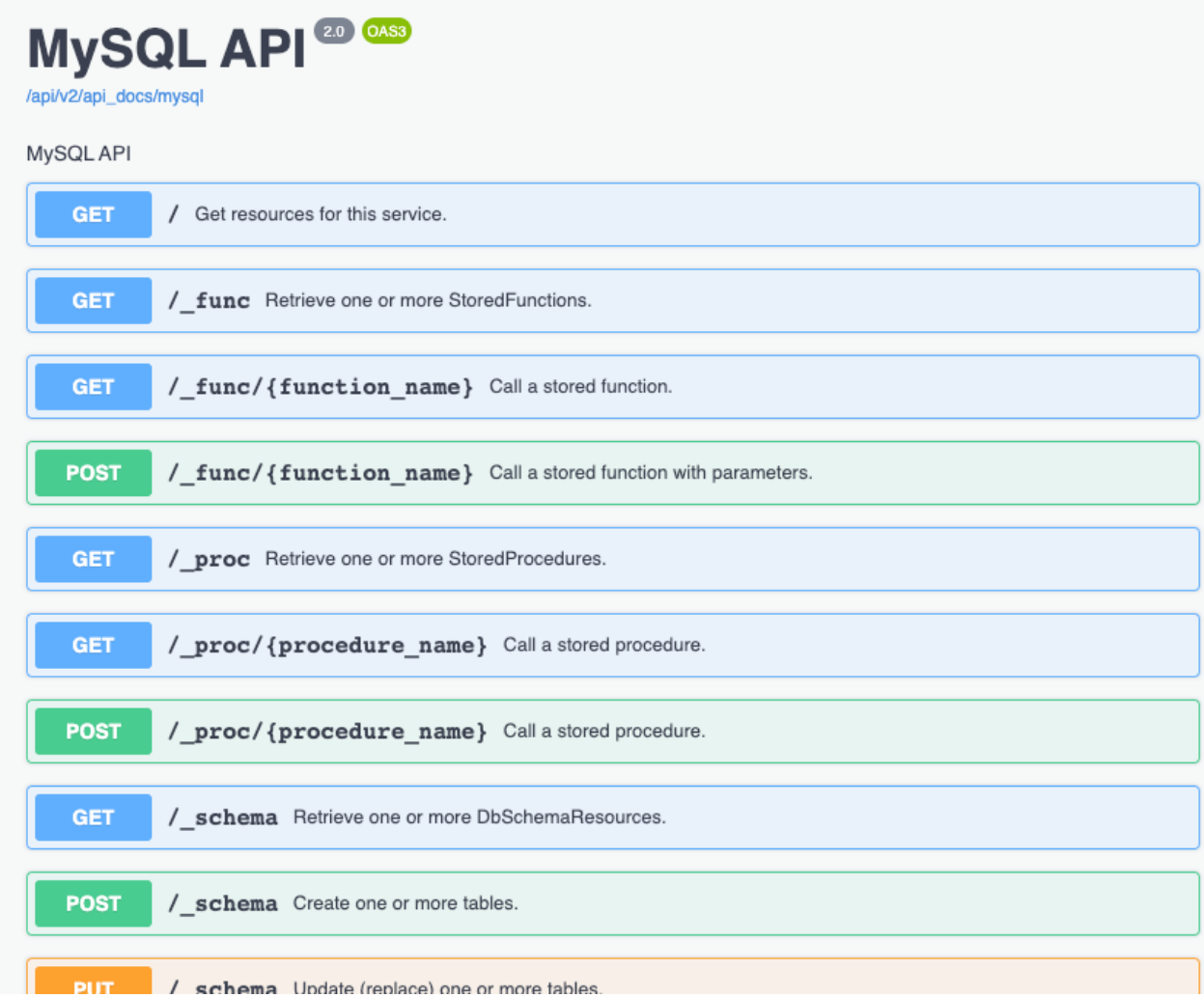
After entering the credentials, scroll down to the bottom of the page and press the `Save` button to generate your MySQL API!

Reviewing the API Documentation

After generating the API, click on the API Docs tab to review the associated documentation. DreamFactory autogenerates interactive OpenAPI documentation for any data source supported through a native connector, MySQL included. To access the documentation, click on the API Docs tab and then search for your newly created API using the search field at the top of the page.

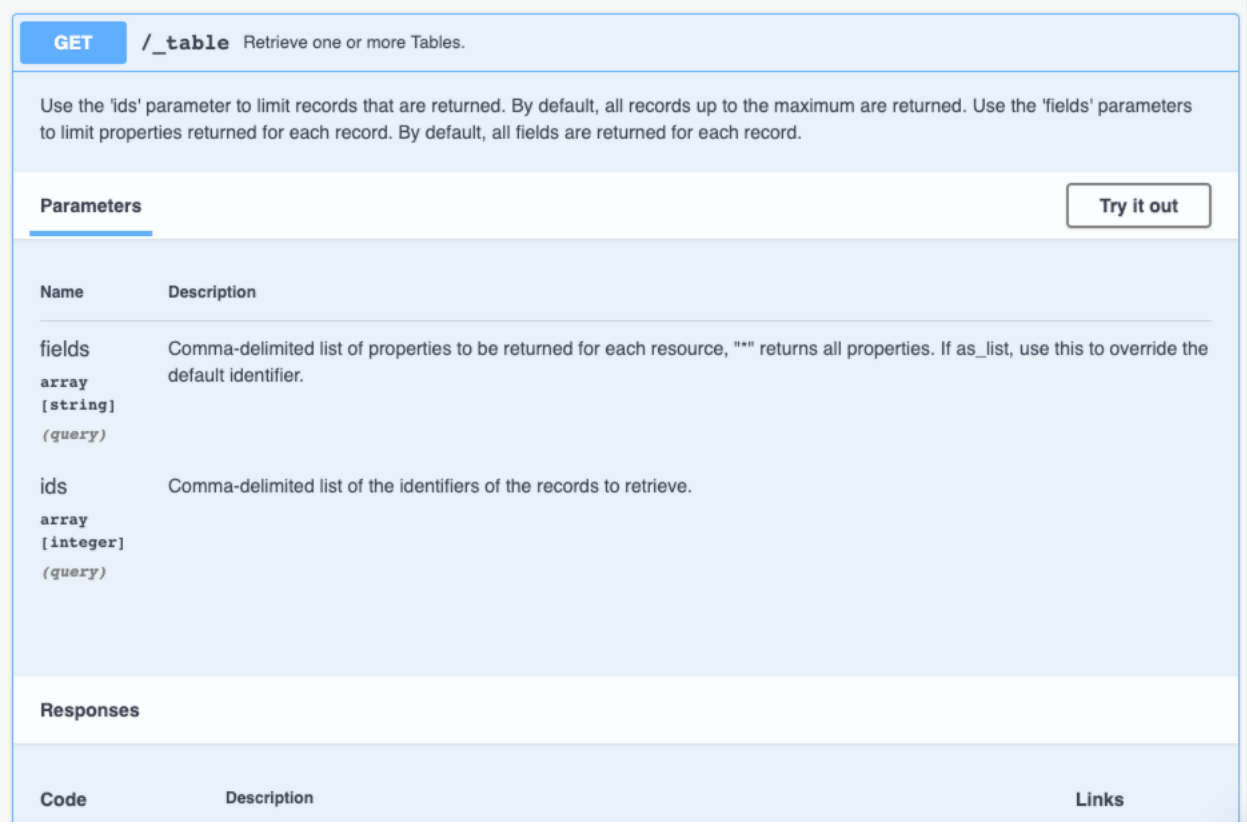
You'll be presented with a list of 39 generated endpoints for this API (see below screenshot).

[1] https://guide.dreamfactory.com/



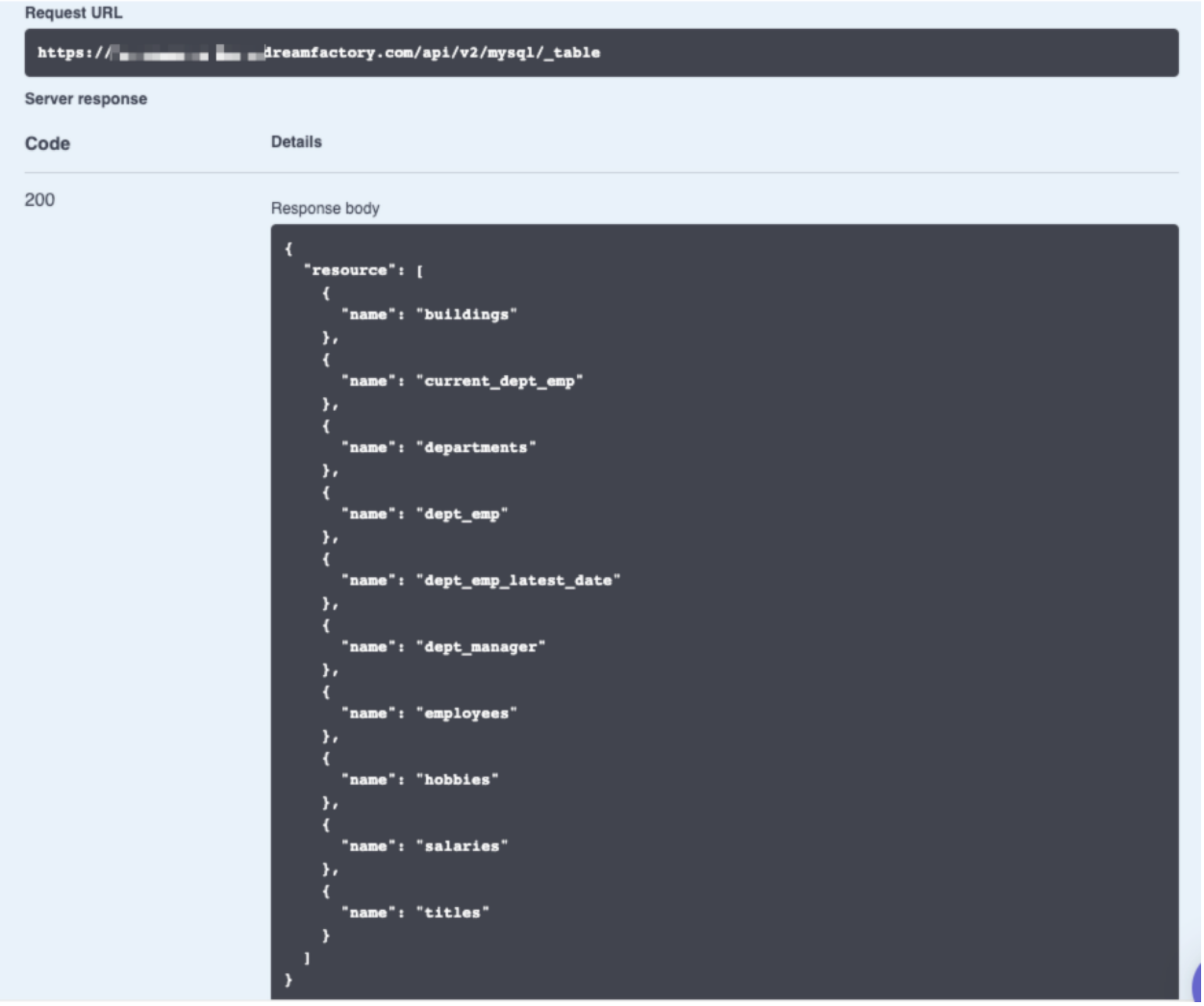
Viewing the API Documentation

Click on any endpoint and you'll be able to interact with it using the point-and-click interface. For instance try clicking on the `GET /_table` endpoint to retrieve a list of tables found in the example database. You'll be presented with the interface found in the following screenshot. Click the `Try it out` button and then press the `Execute` button.



Retrieving Table Endpoints

When you press `Execute`, DreamFactory will initiate a call to the chosen endpoint, and present the results within the interface. For instance, the below screenshot presents the `GET /_table` response. Note the response is returned in JSON format (DreamFactory also supports XML and CSV).

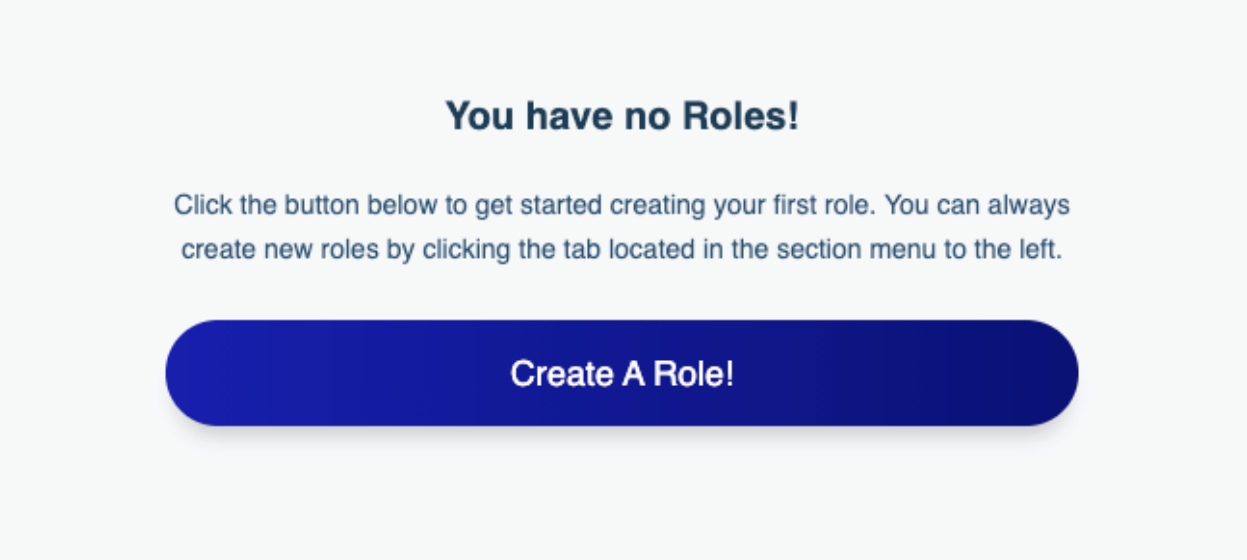


The Table Endpoint Response

Consider spending some time interacting with the other endpoints to get a sense for what you can do with the API!

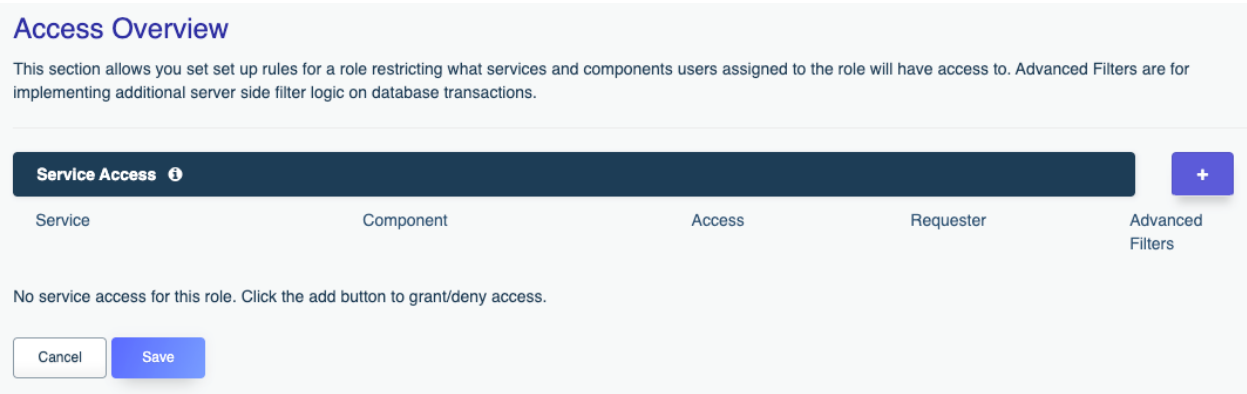
Creating a Role

With the MySQL API generated, let's create an associated role-based access control (RBAC). An RBAC determines what exactly a client can do with your API. For instance you might want the API to be read-only, or perhaps access should be limited to a select few tables, views, or stored procedures. This is easily accomplished using the point-and-click interface. Click on the `Roles` tab to get started and you'll be presented with the interface found in the following screenshot.



Creating a Role

Click the `Create a Role!` button and you'll be prompted to enter a role name and description. Unlike the service name, the role name is only used for human consumption so be sure to name it something descriptive such as `MySQL Role`. You can ignore the `DN (Distinguished Name)` field because it's used only when associating a Microsoft Active Directory group with a role. Click the `Next` button. You'll be prompted to identify the API(s) which should be associated with this role, and define the restrictions. The default interface looks like that presented in the below screenshot.



The Role Access Tab

The `Service` select box contains all of the APIs you've defined this far, including a few which are automatically included with each DreamFactory instance (`system`, `api_docs`, etc). Select the `mysql` service. After selecting the `mysql` service, click on the `Component` select box. You'll see this select box contains a list of all assets exposed through this API. If you leave the `Component` select box set to `*`, then the role will have access to all of the APIs assets. However, you're free to restrict the role's access to one or several assets by choosing for instance `_table/employees/*`. This would limit this role's access to just performing CRUD operations on the `employees` table. Further, using the `Access`

select box, you can restrict which methods can be used by this role, selecting only GET, only POST, or any combination thereof.

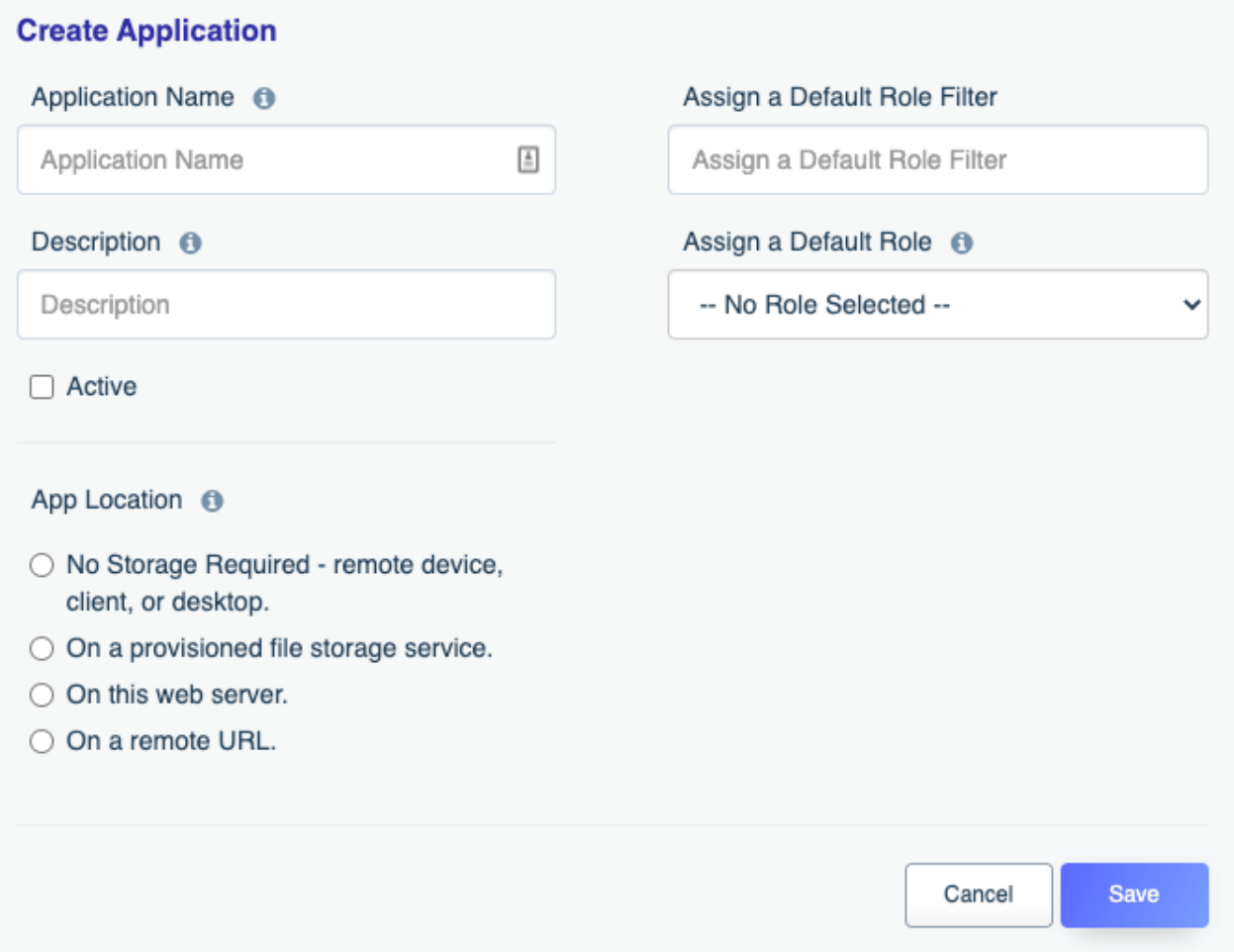
For our project you should configure the following settings:

- **Service**: Choose the namespace designation associated with the generated MySQL API.
- **Component**: Select the asterisk (*), meaning the microapp will have access to all endpoints.
- **Access**: Select all five checkboxes. When done, this form control will be set to `5 Selected`.
- **Requester**: Leave this set to `API`.

After setting these values, click the `Save` button to generate your role!

Creating an API Key

With the API and role generated, it's time to generate an API key. This API key will accompany all API requests, and will be associated with the newly created role. DreamFactory will consult the associated role with each request to determine whether the client is authorized to complete the desired task. To generate the API key, click on the `Apps` tab. You'll be presented with a table identifying a few API keys associated with predefined services such as `api_docs` and `file_manager`. Click the `Create` link located on the left side of the screen to generate a new key. You'll be presented with the following screen:



Creating an API Key

You'll need to complete the fields presented on this screen to generate the API key:

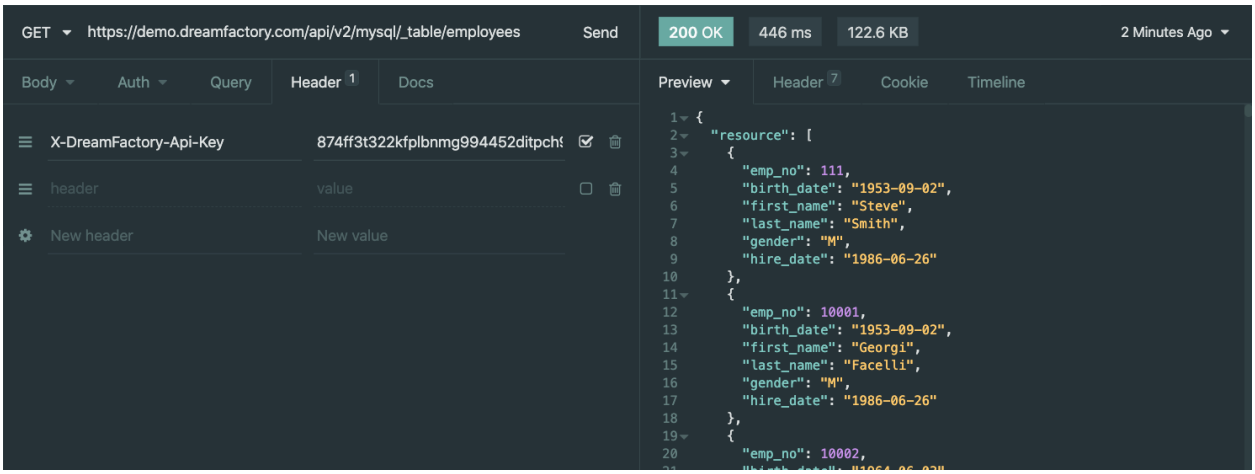
- **Application Name and Description**: The application name and description are used purely for human consumption, so feel free to complete these as you see fit.
- **Active**: This checkbox can be used to toggle availability of the API key, which will be generated and presented to you when the application is saved.
- **App Location**: This field presents four options for specifying the application's location. The overwhelming majority of users will choose No Storage Required because the API key will be used in conjunction with a mobile or web application, or via a server-side script.
- **Assign a Default Role Filter**: Some of our customers manage dozens and even hundreds of roles within their DreamFactory environment! To help them quickly find a particular role we added this real-time filtering feature which will adjust what's displayed in the Assign a Default Role select box. You can leave this blank for now.
- **Assign a Default Role**: It is here where you'll assign the newly created role to your application. Click on this select box and choose the role according to the name used in the last section.

Click the `Save` button and the new API key will be generated. Click the clipboard icon next to your new API key to select the key, and then copy it to your clipboard, because in the next section we'll use it to interact with the API.

Testing the API

With the API generated and associated RBAC and API key created, it's time to begin interacting with the MySQL data via an API call! There are a few easy ways to do this, however the most straightforward involves downloading an API client such as Postman (https://www.postman.com/[2]) or Insomnia (https://insomnia.rest/[3]).

To begin interacting with your API, you'll need to specify the HTTP method, desired API endpoint, and the API key. The API key is securely passed along in an HTTP header. The header should be named `X-DreamFactory-Api-Key`. In the below screenshot you'll see an example involving querying a MySQL API.



Using Insomnia

Conclusion

With the API, role-based access control, and API key generated, it's time to begin building the microapp!

[2] https://www.postman.com/
[3] https://insomnia.rest/

Chapter 3. Creating a Service Integration with Citrix Workspace

You will need to complete a number of prerequisites before you dive into building Microapps. Your end-users will access Microapps and workflows from their Citrix Workspace. In the real world, Citrix Workspace is used to access applications, workflows, data and desktops from any location. One service that is provisioned through Workspace is the Microapp functionality.

In a development scenario, we want to build and test Microapp integrations and deploy them to test subscribers. To do this, we need to configure Identity and Access Management - the authentication our end-users will use to access Workspace. We then need to configure Citrix Workspace to display our Microapp integrations so end-users can access them. There are a number of guides in the Citrix Production Documentation that cover this subject:

- Request a 30-day Citrix Microapps Developer instance - https://developer.cloud.com/citrixworkspace/citrix-workspace-platform[4]
- Configure Authentication to Workspace - https://developer.cloud.com/citrixworkspace/citrix-workspace-platform/build-workspace-microapp-integrations/docs/getting-started[5]
- Enable Microapps in Workspace - https://docs.citrix.com/en-us/citrix-microapps/getting-started.html#enable-microapps-service-in-workspace[6]
- Configure Citrix Workspace - https://docs.citrix.com/en-us/citrix-workspace/configure.html[7]

Introducing the Microapp Service

The Citrix Microapp Service is part of the overall Citrix Workspace platform. Citrix Workspace allows organizations to provide users with a single, easy to use digital workspace where all their applications, desktops and content can be securely delivered to any device. The Citrix Microapp Service takes Workspace to the next level by exposing workflows out of those applications so users can save time and deal with specific tasks without having to logon to the full application.

The Microapp Service is a cloud hosted solution within Citrix Cloud. Typically, organizations would use the Microapp Service in conjunction with other services provided by Citrix Cloud.

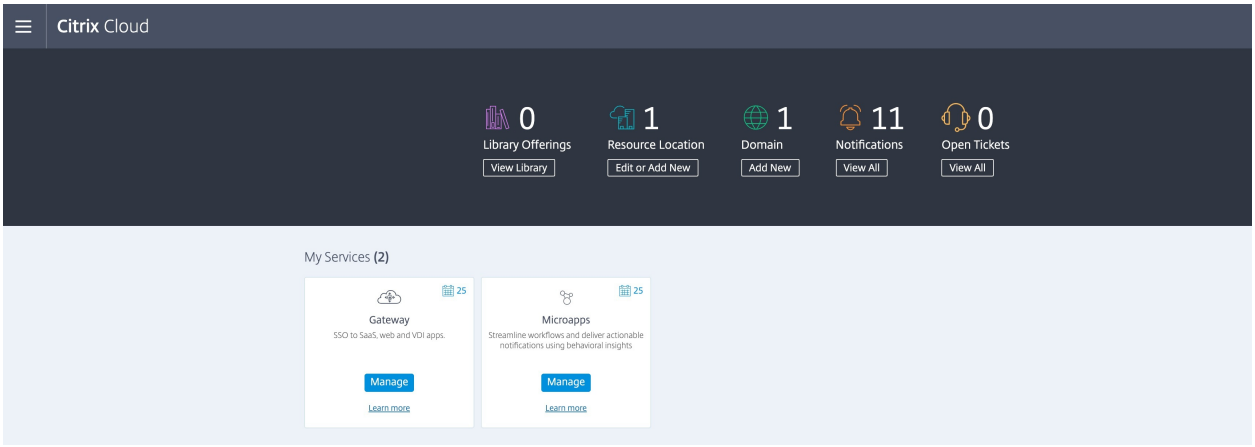
[4] https://developer.cloud.com/citrixworkspace/citrix-workspace-platform
[5] https://developer.cloud.com/citrixworkspace/citrix-workspace-platform/build-workspace-microapp-integrations/docs/getting-started
[6] https://docs.citrix.com/en-us/citrix-microapps/getting-started.html#enable-microapps-service-in-workspace
[7] https://docs.citrix.com/en-us/citrix-workspace/configure.html

Accessing the Microapp Designer

You'll build Microapps using the Microapp designer - a cloud hosted service in Citrix Cloud that allows you to configure out-of-box integrations, and as is the case with this example, custom connections using the HTTP Connector.

Once you have requested your Microapp developer instance from developer.cloud.com, login to Citrix cloud from citrix.cloud.com using the credentials you used when signing up.

You will be presented with the Citrix Cloud main dashboard.



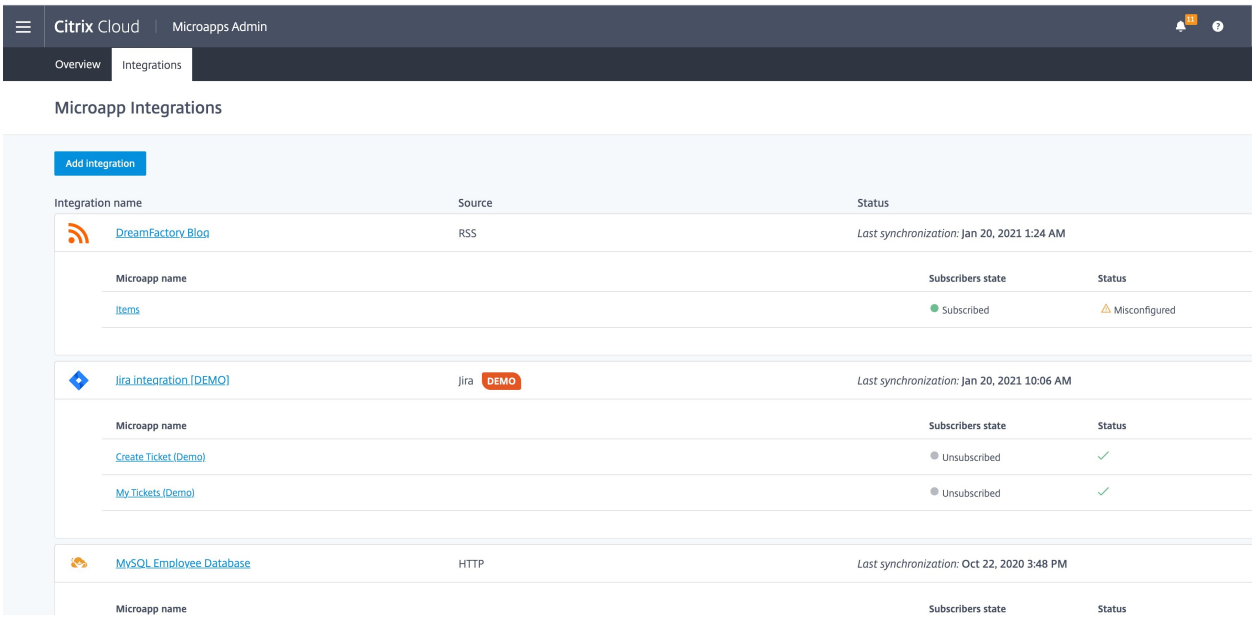
The Citrix Cloud Dashboard

The Citrix Cloud dashboard displays a list of all the services you are currently subscribed to. This guide assumes you have configured Identity and Access Management and also Citrix Workspace Configuration (See Chapter 3: Prerequisites).

Creating a Custom HTTP Integration

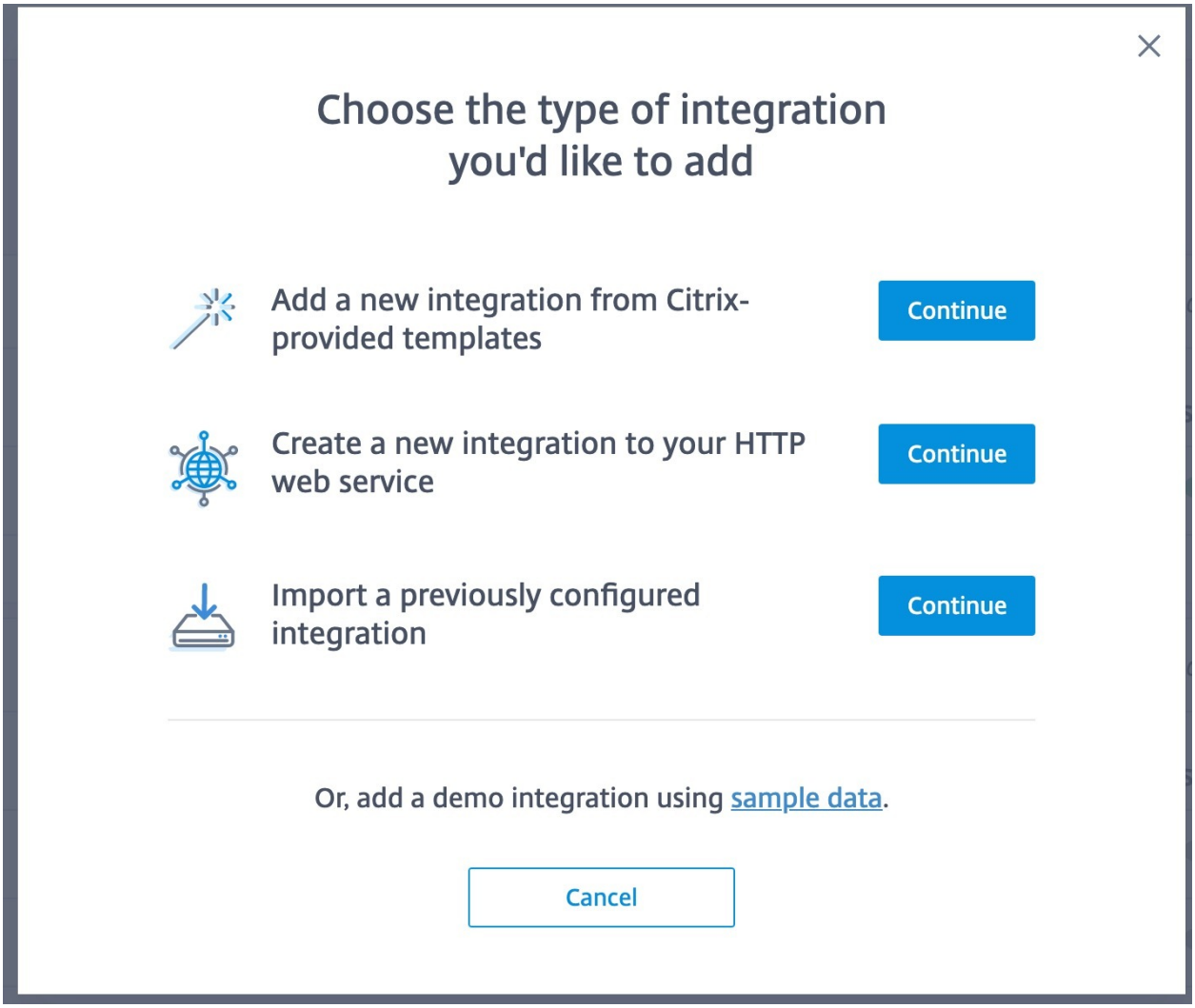
From the Citrix Cloud dashboard, click the "Manage" button from the Microapps panel.

From the Microapps Admin page, select "Integrations" from the main menu. This then opens the Microapp Integrations page where all your integrations will be listed.



The Microapp Admin Page

From the top of the Microapp integrations page, select “Add Integration”. The “Choose Integration Type” dialog box will be displayed. For the purposes of this guide, select the “Create a new integration to your HTTP web service”.

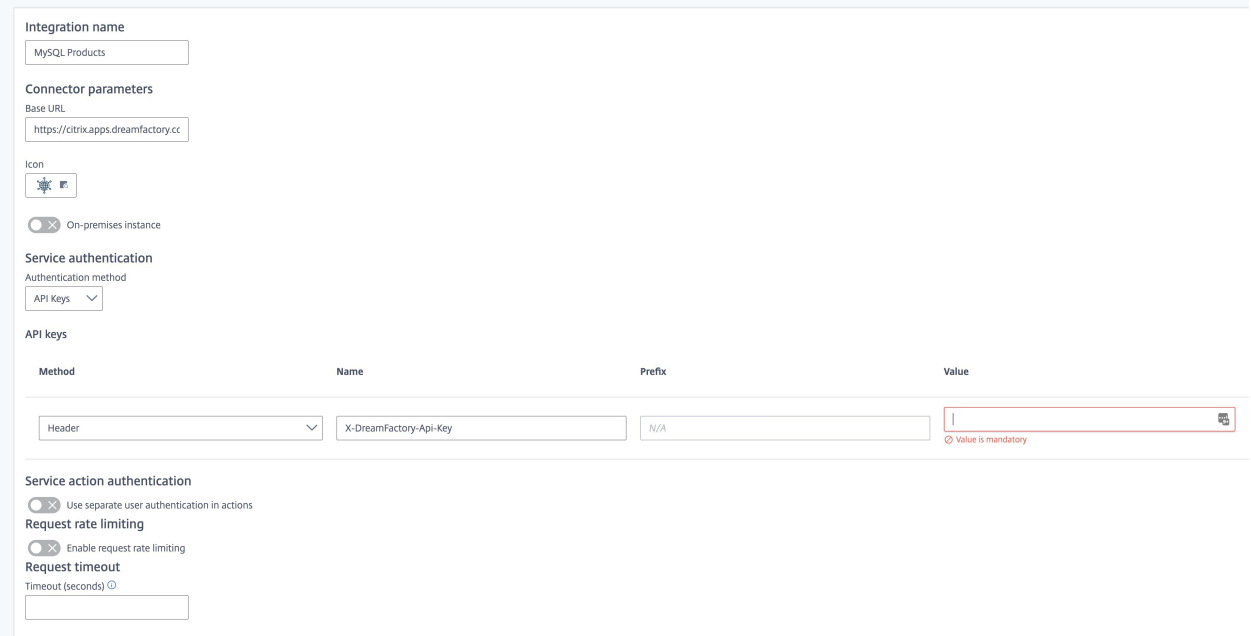


Choosing Your Integration Type

The "Add HTTP Integration" page will be displayed.

Add HTTP Integration

We first need to configure the basic integration. This includes configuring the REST endpoint we want to connect to and how we will authenticate. It will be useful at this point to have your DreamFactory configuration to hand as you will need the endpoint URL and `X-DreamFactory-API-Key`.



Configuring the Integration

For the Integration Name, enter a logical and meaningful name for your integration, typically this is the name of your back-end system.

Configure the Base URL as per your DreamFactory instance. This guide assumes you are using the cloud-hosted version of DreamFactory. You can find the base URL from the API Docs of your configured DreamFactory Service (see: Testing The API). When using the hosted environment that base URL will look like this:

```
1 https://YOURINSTANCE.apps.dreamfactory.com/api/v2/
```

Select "API Keys" from the Service Authentication method pull down menu, you will then need to change the Method to "Header" from the API Keys selection. In the name enter `X-DreamFactory-Api-Key`.

In the Value box, enter the `X-DreamFactory-Api-Key` as configured when you configured a Role in DreamFactory (see chapter 1). Click `Save` to save the initial service configuration. You will be returned to the main configuration screen.

Adding Data Endpoints

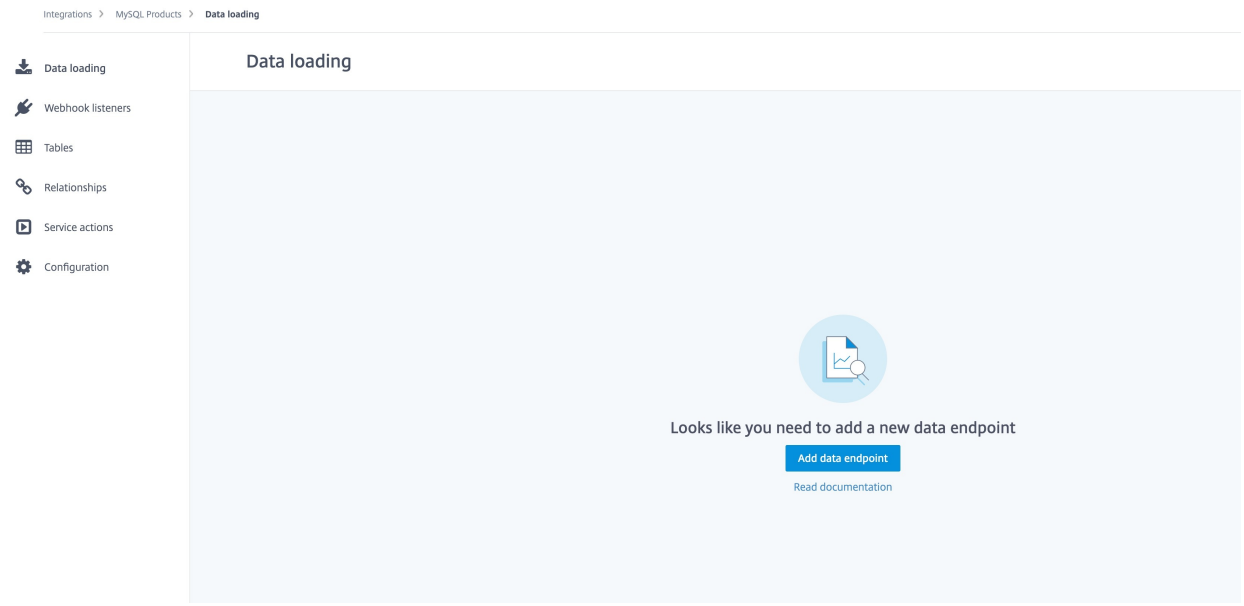
The next step is to configure the data endpoint we wish to synchronize with the Microapp cache and use within our Microapp integrations. We do this through a "Data Endpoint", typically this is a REST api destination or combination of different api destinations that build our tables in the Microapp cache. In the example we're using as part of this guide, it may be a specific table(s) or procedure in our MySQL database that has been exposed through DreamFactory.

At a configured interval, the Microapp service will synchronize the Microapp cache with the data endpoint. The synchronization interval is configurable. This is worth bearing in mind since our Microapps will be operating from a cache as opposed to working in real-time. The cache follows a typical relational database model. It is mandatory to configure primary keys on tables within the cache. Developers are able to link tables together and perform Child API calls on other end-points to build the cache.

You are able to force a synchronization of the cache after we make a change. For example, if we were creating a Microapp for adding new products to the ERP, once we've submitted our new product using a POST Service Action, we can force the Microapp cache to perform a synchronization. This is covered in more detail in "Creating Service Actions".

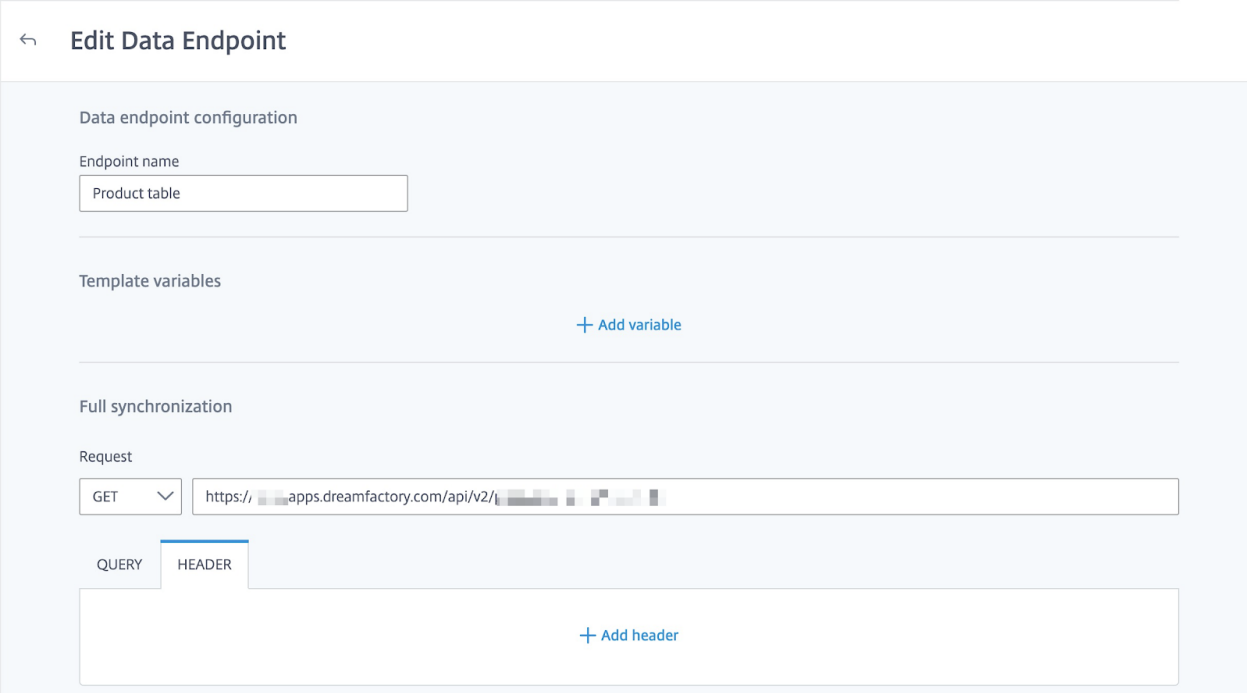
We are going to configure a data load using an HTTP GET Request to our DreamFactory endpoint, specifically, the Products table in our MySQL database.

From the main integration screen, select "Data Loading":



Loading the Data

From the Data loading page, select "Add data endpoint"

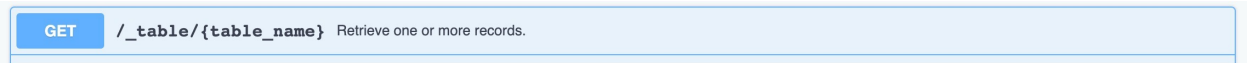


Adding the Data Endpoint

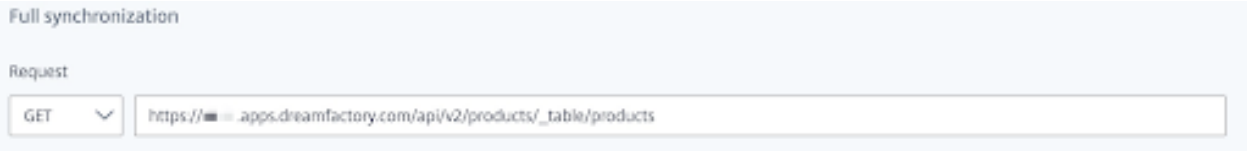
In the Endpoint name field, enter a descriptive name for the endpoint, this can be anything and is not seen by the end-user using the Microapp. You may wish to use a naming convention at this point to maintain consistency across different endpoints, for the purposes of this guide, we're using the name of the MySQL table we wish to talk too.

It is possible to use dynamic values in the HTTP request we're about to make if we want to use dynamic values as part of our HTTP GET request. For example, if we only wanted to retrieve records where a field is above a certain value, then we could configure a template variable to do this. Template variables can then be used using "double moustache" notation - `{{my_variable}}` - in any part of our GET request, including the URL, Query parameters or Header information.

For now, enter the request URL from where we'll GET our data, you can use the API Docs in DreamFactory to retrieve this information. In our example, we're using the GET endpoint for the Products table.



The DreamFactory Table Name Endpoint

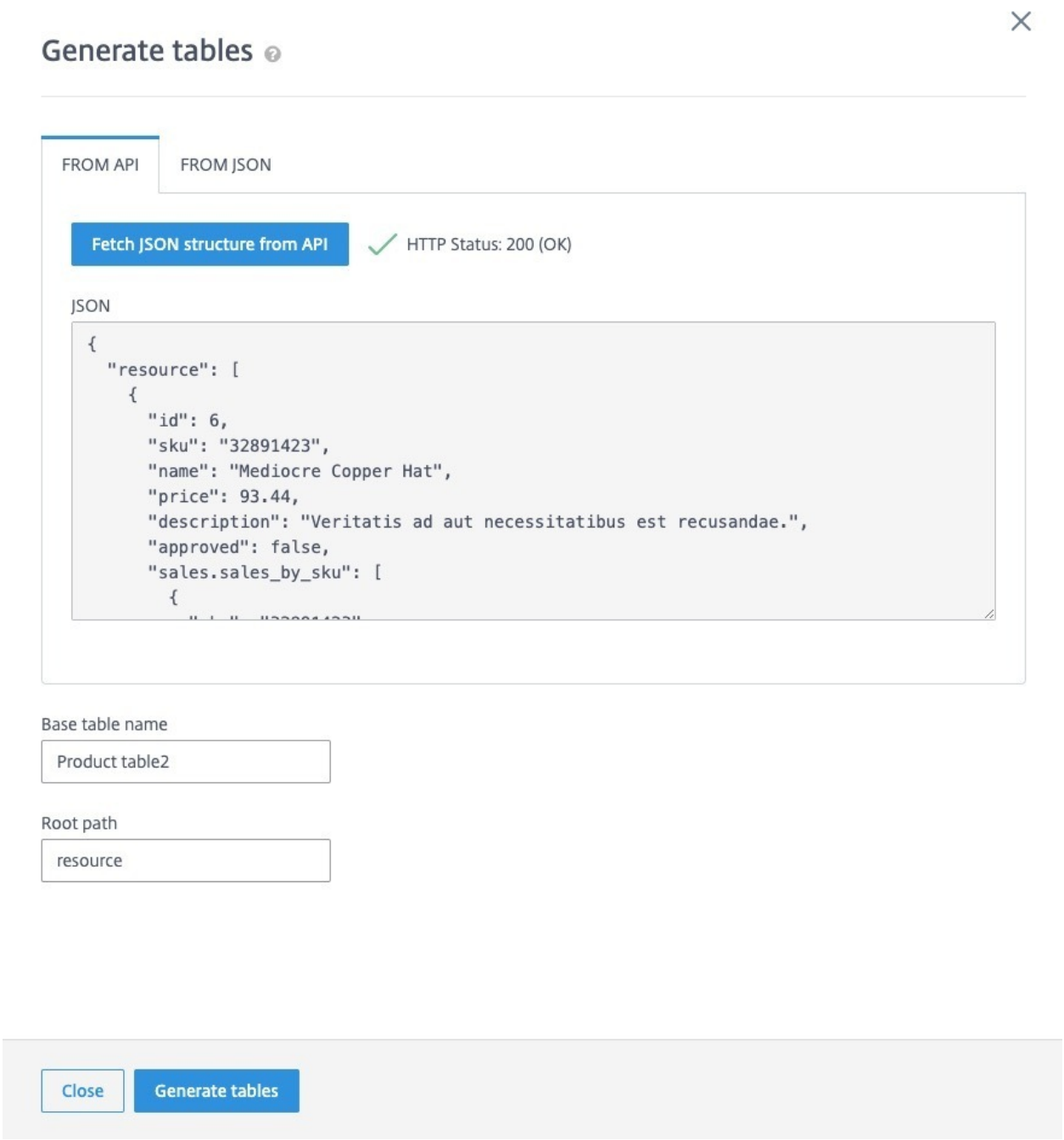


Adding the Endpoint

It is good practice to test your GET Request using a tool such as Postman or Insomnia.

Click the `Test with parameters` button to check the configuration, click `Check Service` in the "Test service with parameters" blade. You will then see a success message, details of the GET request to the endpoint including a HTTP status of 200. If your request fails, check our HTTP GET configuration using a tool such as Postman or Insomnia.

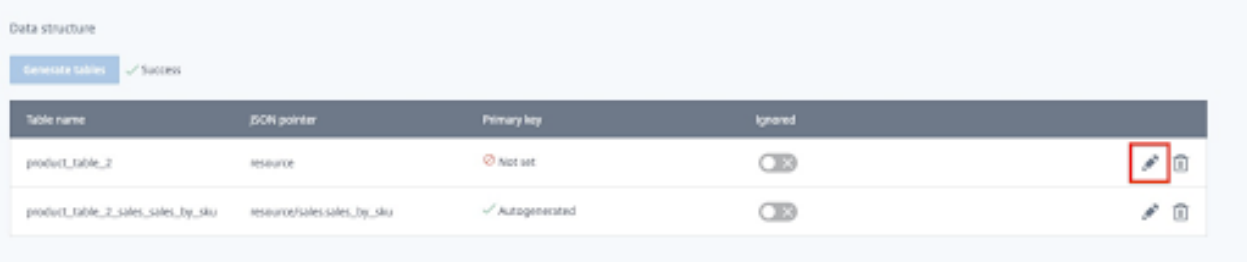
We now need to create a data structure that we'll use as part of our Microapps. This is essentially a relational database within the Microapp cache that is populated by the requests made to the endpoint. Click `Generate tables` to display the Generate tables blade.



Reviewing a Test Request

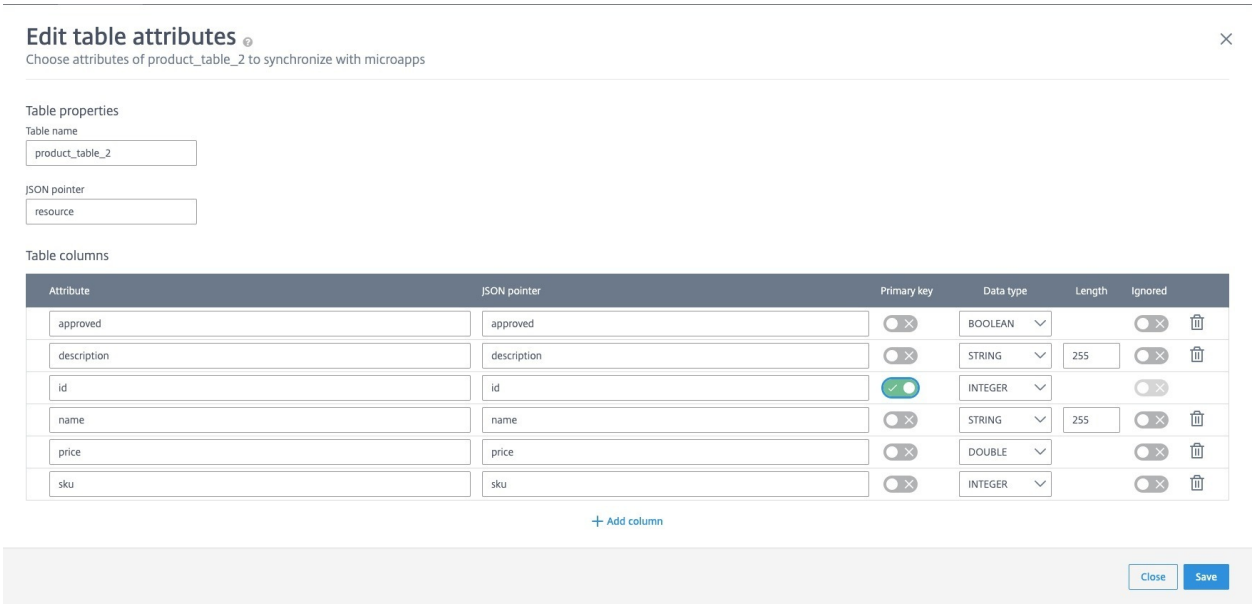
Click the `Fetch JSON structure from API` button. This uses the JSON structure to define the fields and data types for the tables the Microapp service will use from the cache. The JSON structure will be displayed. We can see from the example above that individual records are contained within the `resource` array, therefore we need to add `resource` as the "Root path" for individual records. We can also change the base name of the table if required. This is useful if the Base table name provided

automatically is not clear. Click `Generate tables`.



Generating the Tables

Once the tables have been configured, we will need to set a Primary Key for the table. Click on the `Edit tables` button (highlighted above) to set the Primary Key, in this example, we're going to set the field `ID` as the Primary Key.



Editing Table Attributes

Select the Primary Key and click `Save`. From the Add data endpoint screen, click `Add`. You will then be returned to the Data loading page.

Adding Service Actions

In chapter 1, we discussed the use-cases that we want to provide with our Microapps, these were:

- Allow the Marketing Manager to approve/edit/create Product Descriptions without having to access the ERP and remove the need to use email to send product updates.
- Provide Sales Reps with a simple interface to search for Products without access to the ERP.

- Provide Sales Reps with a notification that Product has been updated without using email.
- Allow the Products Manager to add new products.

In the first example, we require our Microapp to make changes (writebacks) to our Products Database, specifically, updating a specific field in an existing record. This is done using a HTTP PUT to our DreamFactory provided end-point. Within Microapps, a user initiated HTTP POST, PUT, DELETE etc is typically done using a Service Action.

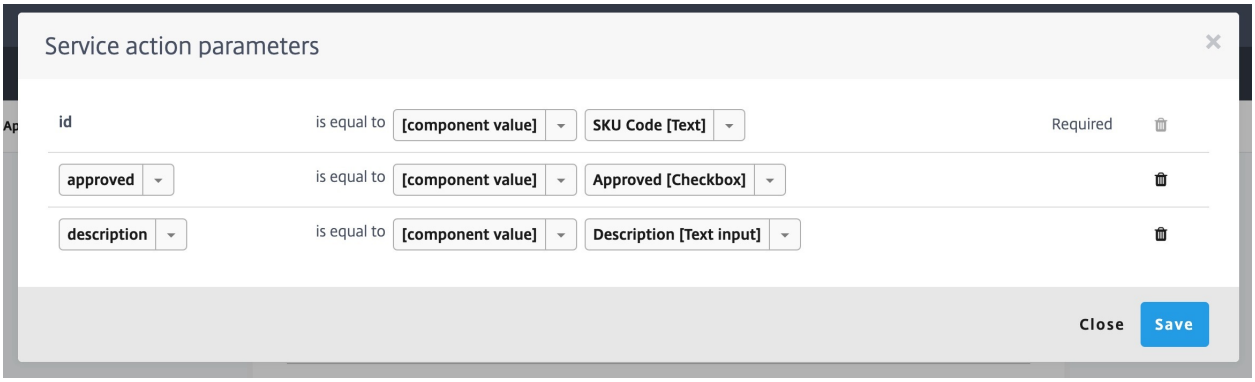
Service Action - Approve Product Descriptions

Service Actions are extremely flexible and are configured similar to Data Endpoints. We will create a Service Action that allows the Marketing Manager to approve changes to Product Descriptions.

From the menu, select “Service Actions”, and click “Add Service action”. The “Add service action” page will be displayed.

In the “Action name” field, enter a name for the Service Action.

We now need to configure the Service Action parameters that will compose our HTTP PUT to the API endpoint.



Service Action Parameters

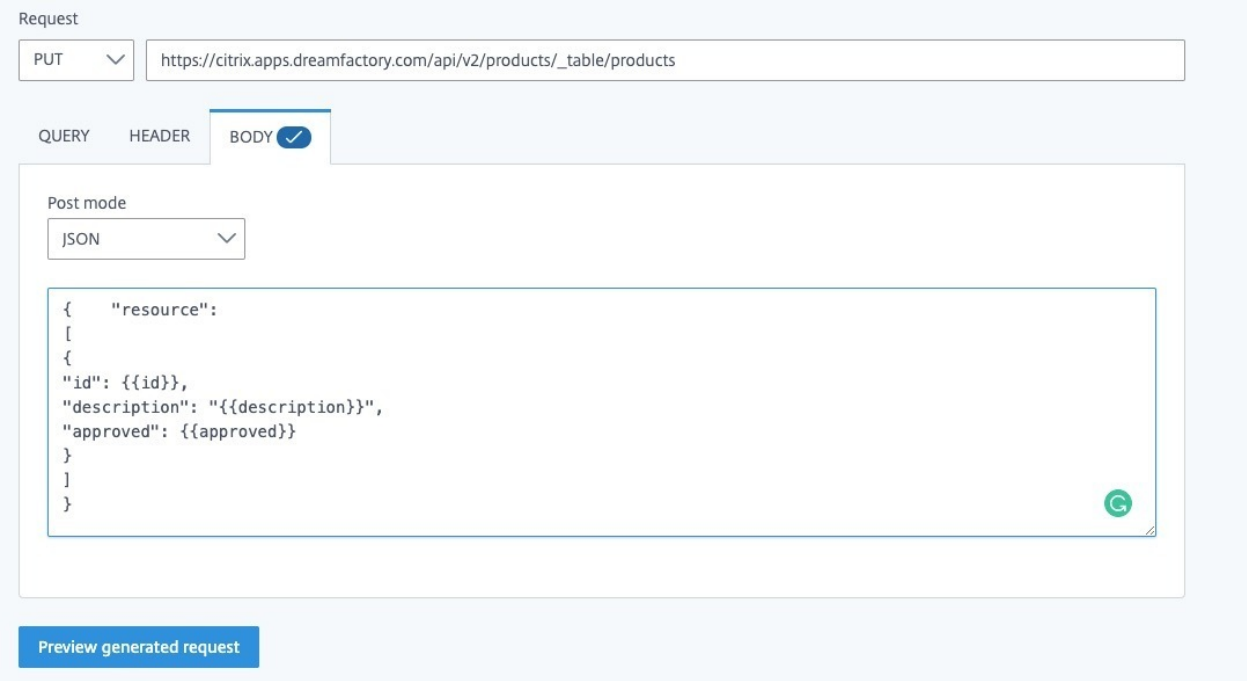
In our example, we want to update two fields, “Approved” and “Description”. We’ll add these as Service Action parameters along with the ID of the record we’re updating.

We then move to configure the Action Sequence. In advanced configurations, we may wish to configure data cache synchronization prior to our PUT request. We can also configure Post action data updates that would synchronize the cache following an update. For now, we’ll configure the Action execution.

Select the Request type as “PUT” and configure the URL endpoint as the same one used in the initial Data Loading since this is the table within MySQL we want to update. We can now start to build up our HTTP PUT request, it is possible to configure Query parameters, Header parameters etc. However, these are not required for this example, however, we do want to configure the Body so our HTTP PUT request is in the correct format. Select the “Post mode” as JSON. In the body, place the JSON template as per the below:

```
{  "resource":
  [
    {
      "id": {{id}},
      "description": "{{description}}",
      "approved": {{approved}}
    }
  ]
}
```

The double-moustached variables, such as `{{description}}` refer to the Service Action parameters we configured above. When we come to build the pages of our Microapp, we will reference these Service Action parameters once more on our Product Description approval form.



Configuring a PUT Request

We can also click on the "Preview generated request" button to open the preview request blade to see what our request will look like, we can add in some test data and "Regenerate preview" will preview the request.

Now that we've configured our Service Action, we can hit the "Save" button and return to the main integration page.

Service Action - Add New Products

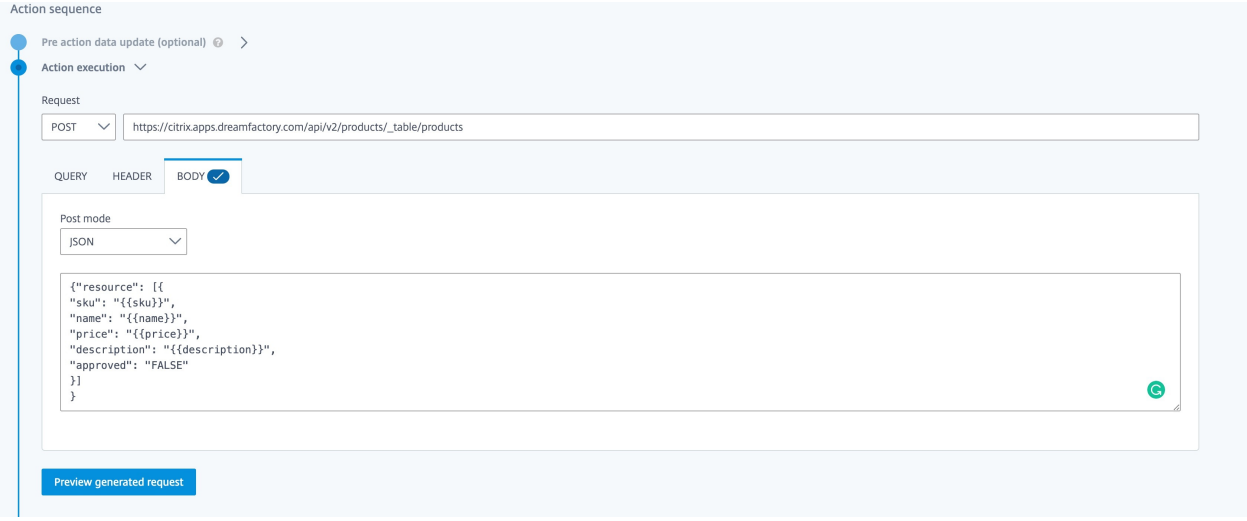
From the menu, select "Service Actions", and click "Add Service action". The "Add service action" page will be displayed. In the "Action name" field, enter a name for the Service Action.

We now need to configure the Service Action parameters that will compose our HTTP POST to the API endpoint.

We want to add all the fields that make up the Products table so we can add a new record, we'll add these as Service Action parameters.

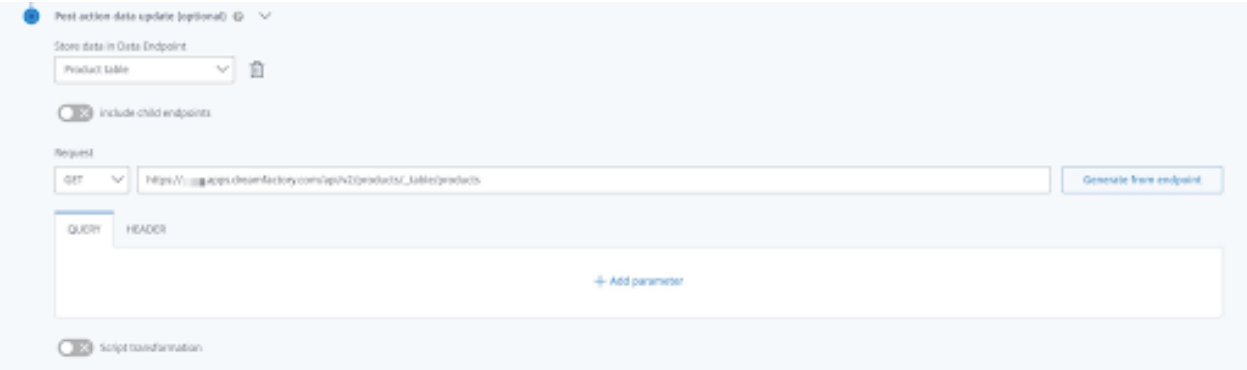
Select the Request type as "POST" and configure the URL endpoint as the same one used in the initial Data Loading since this is the table within MySQL we want to update. We can now start to build up our HTTP POST request, configure the Body so our HTTP POST request is in the correct format. Select the "Post mode" as JSON. In the body, place the JSON template as per the below:

```
{
"resource":
  [{
    "sku": "{{sku}}",
    "name": "{{name}}",
    "price": "{{price}}",
    "description": "{{description}}",
    "approved": "FALSE"
  }]
}
```



Configuring a POST Request

On this Service Action, when a new product is added to the database via the POST request, we should force a synchronization, so the data cache contains a copy of the new product. We can configure a "Post action data update" that will execute each time the Service Action completes successfully, accomplished by configuring the post action data update to GET all the records from the Products table.



Configuring a POST Action Data Update

Set the Request to "GET" and enter the URL for the Products table.

Now that we've configured our Service Action, we can hit the "Save" button and return to the main integration page.

In a production environment, we could configure the post action update further using queries and parameters. For example, we could limit the request to new records only by using `{{id}}` to only retrieve the record we've just added in to the cache:

```
https://<YOUR DREAMFACTORY.apps.dreamfactory.com/api/v2/products/_table/products/{{I\
D}}
```

Where ID is configured as the Primary Key of the record we've just added.

In summary - we should not have two Service Actions configured that:

- Allow the Marketing Manager to update records in the database using a HTTP PUT Request.
- Allow the Products Manager to add new records to the database using a HTTP POST request.

We're now ready to go and build some Microapps!

Chapter 4. Creating a Microapp with Citrix Workspace

Now that we have created our Service Integration, we can get to the good stuff and start building Microapps for our end-users. We're going to build three Microapps that service our use-cases we discussed in chapter 1. As a reminder, these are:

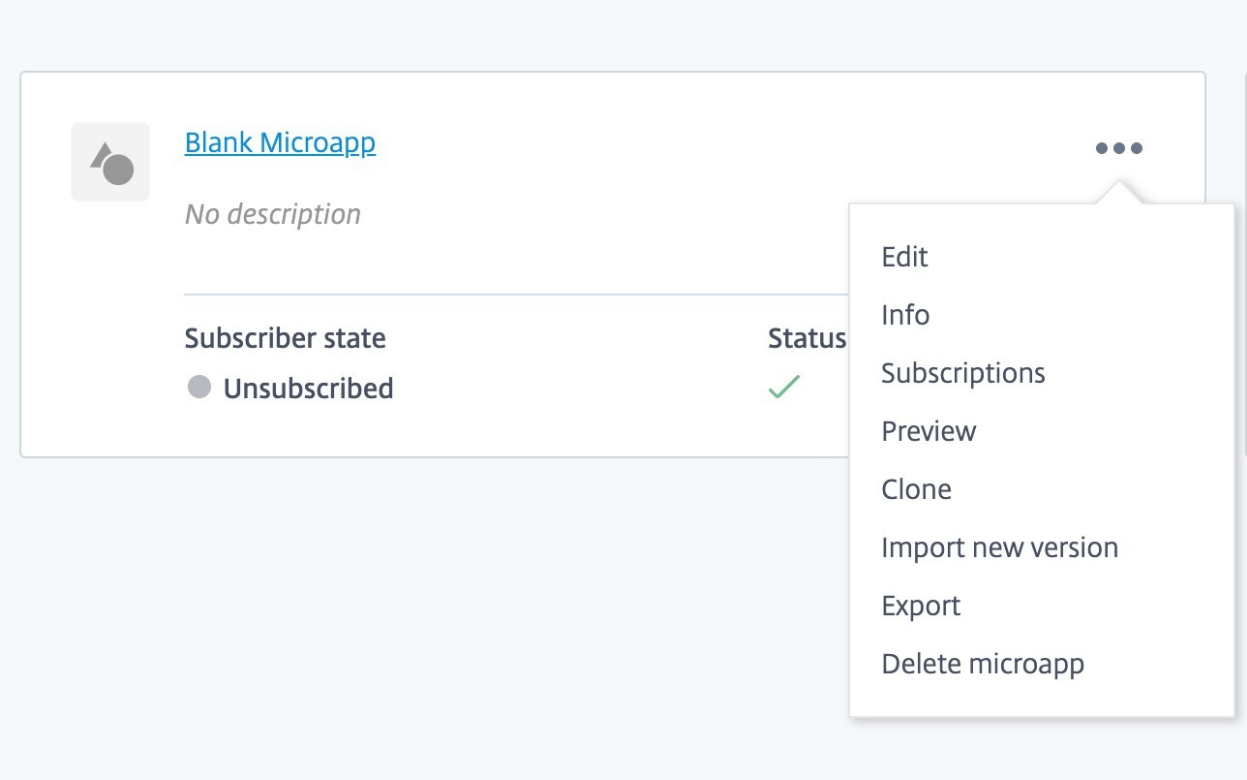
- Allow the Marketing Manager to approve Product Descriptions without having to access the ERP and remove the need to use email to send product updates.
- Provide Sales Reps with a simple interface to search for Products without access to the ERP.
- Provide Sales Reps with a notification that Product has been updated without using email.

Microapps comprise two main elements, notifications and actions. Notifications can be configured to populate the main "news feed" of Citrix Workspace when there is a change in data - for example, a new record is added, or a change is detected. An action is a user-initiated workflow, for example, logging a ticket. Notifications and Actions make up the end-user Microapp experience.

Microapps can vary in complexity, the Citrix Developer community provides a number of videos and walk-through guides that can really help to gain a thorough understanding of building complex Microapps to provide useful end-user workflows. For more information on the Microapp integrations learning journey, take a look at: https://developer.cloud.com/citrixworkspace/citrix-workspace-platform/build-workspace-microapp-integrations/docs/learning-journey[8].

Create a Microapp - Marketing Manager

We will use the Service Integration we created in Chapter 3 and add Microapps to it. Select the Integration from the Microapp Integrations page and click the "Add Microapp" button, this will then create a blank microapp in the integration. Select the dot menu and click "Edit".



Creating a New Microapp

Inside the Microapp, you will see four menus down the left hand side, Notifications, Pages, Localization and Properties.

[8] https://developer.cloud.com/citrixworkspace/citrix-workspace-platform/build-workspace-microapp-integrations/docs/learning-journey

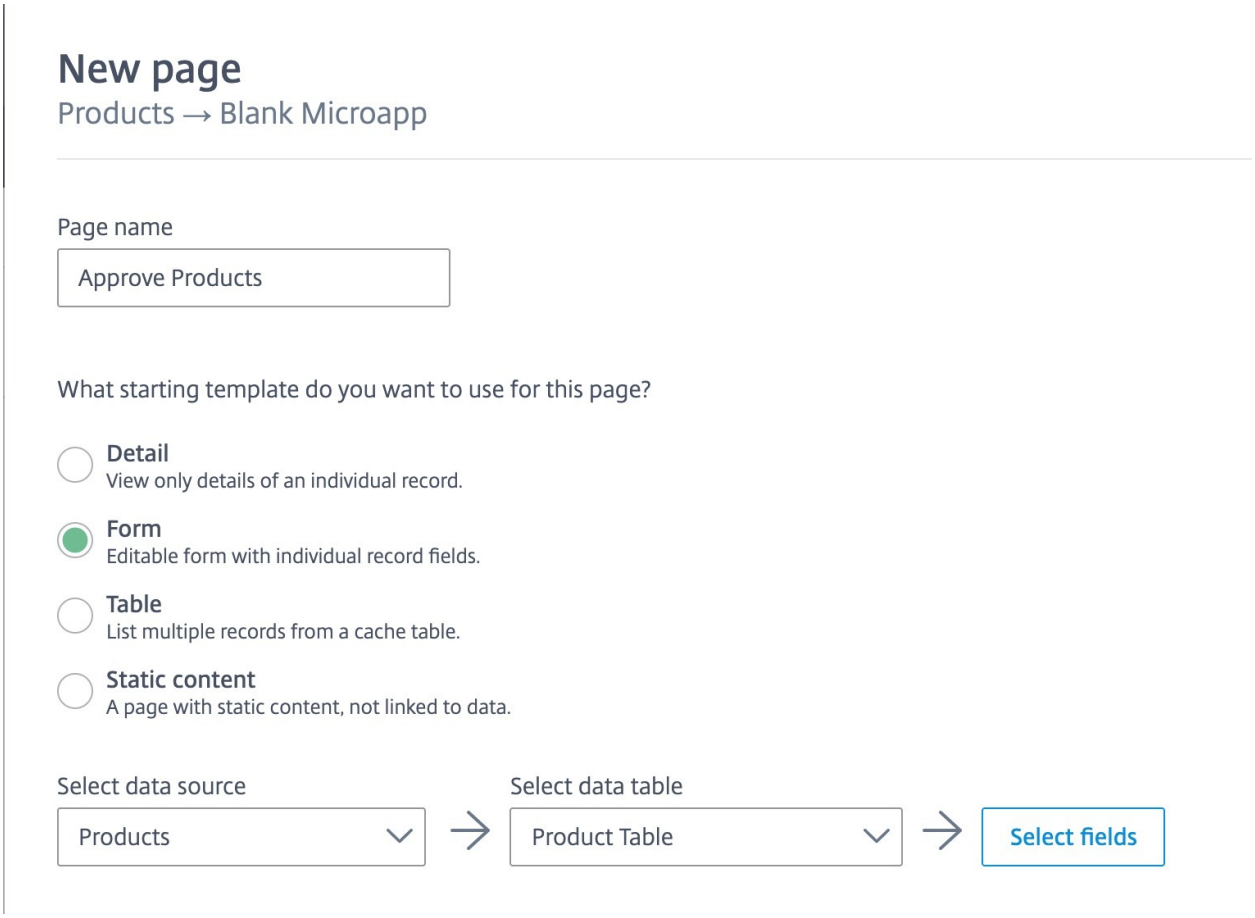
- Notifications allow us to configure just that - notifications we can send users when a particular event happens on our integration.
- Pages are the main way in which end-users interact with our workflows, they allow users to view data, submit forms etc.
- Localization allows us to add language files to our Microapps where we have a use case that requires another language other than English.
- Properties - Allow us to rename our Microapp to something useful, change the app icon and also "End of Life" a Microapp where we're able to turn it off, but leave notifications it has generated in the users feed. The Actions toggle allows us to enable our Microapp as an user initiated Action - This places the Microapp within the users "Actions" in Citrix Workspace. You can toggle the Action to "on" and then specify which page you want users to see first.

Firstly, we're going to configure two pages for our Marketing Manager. The first pages will allow the Marketing Manager to view all outstanding Product Descriptions that need approving. The second page will allow them to make changes to the Product Description and approve. The Marketing Manager will pick a product from the first page (a table) and it will open that record on a details (form) page. We will add the Service Action we created in the "Add Service Actions" section so we can allow the page to submit an update to the database.

Later, we will configure a notification for the Marketing Manager that will notify her when a new product has been added to the database and requires the Product Description approving, opening up the details page directly from the Notification.

Creating a Product Approval Page

Select "Pages" and then "Add Page", give the page a simple name that the end-user is going to understand, such as "Approve Product Descriptions", select the "Form" page type. We now need to configure the page to talk to the correct tables within our integration. From the "Select data source" pulldown menu, select "Products", from the Select data table, select the "Product table". Click Select Fields and select all fields and click "Set fields".



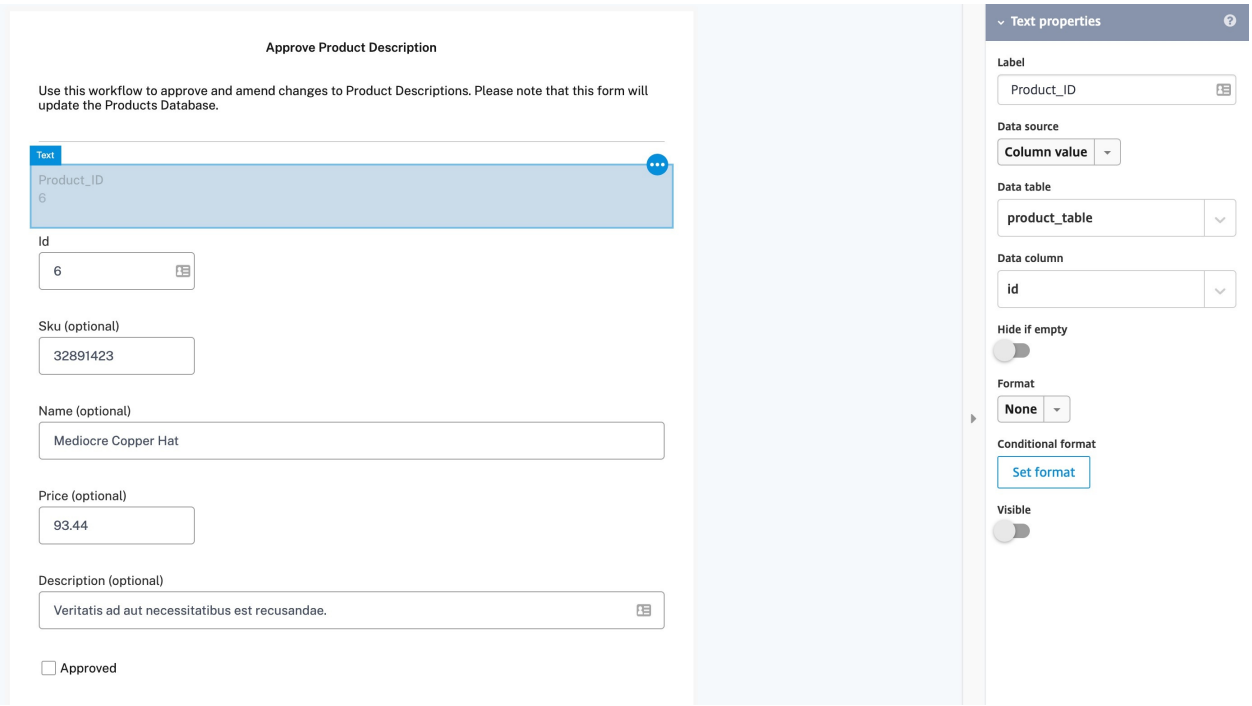
Creating a New Page

You will now be taken to the Page designer. You will notice that the fields that we selected are now present on the page as field inputs.

From the Page Designer we can add UI elements to our pages, it is possible to add images, tables, lookups and other common UI interface elements. By all means play around with the different elements, however, remember not to over complicate your Microapp - we want our workflows to be simple and easy to follow.

We need to change some of the fields within our page so end-users don't make changes to records they shouldn't. Whilst the Service Action we configured does not change these fields, from a UI perspective, we probably want to change them from Text Input fields to just displaying the text.

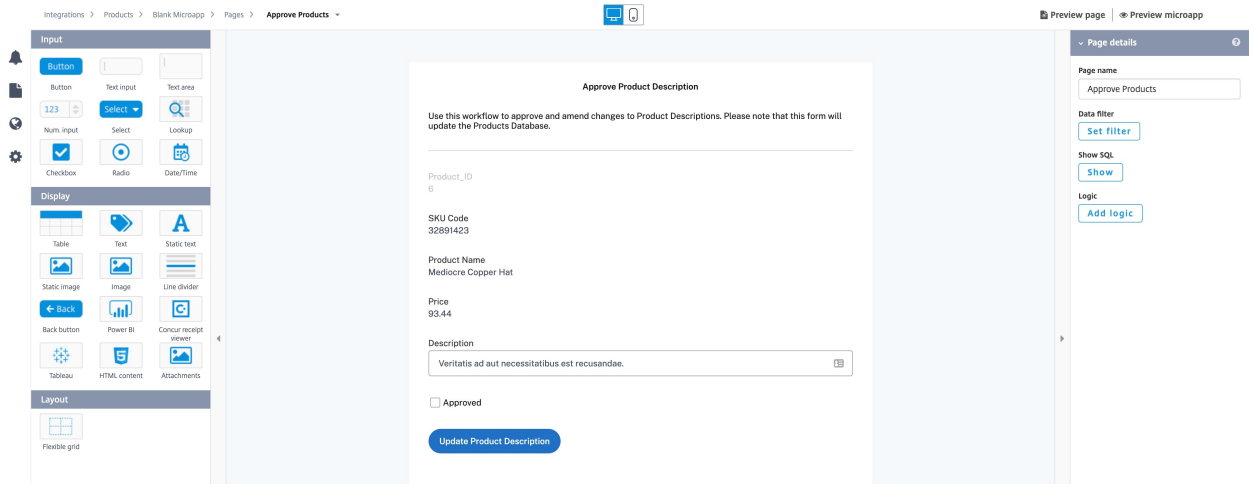
We're going to firstly change the Product ID field from a Text Input to a hidden text field (as we need the Product ID for our Service Action later). From the Display UI elements, drag a "Text" element to the page. Select it and on the right hand Text Properties blade, Update the "Label" to "Product_ID". Select the Data source as "Column value" and select the "Product_table" as the Data table. Select "ID" from the Data column. Lastly, click the "visible" toggle to "off" - this will hide the field element.



Changing the Product ID

You can now delete the original ID text input box. The eagle eyed amongst you may be asking "why did we select all the fields in the first place if we're just going to delete them?" This is true, in reality we only need to add the approve and description fields. However, by initially adding all the fields initially, it's easy to make decisions about which fields you want to see, and what UI element you want them to be.

As we don't want our Marketing Manager to be able to change the SKU, Product Name and Price as well as ID - we need to replace the Text Input fields for these with Text fields.



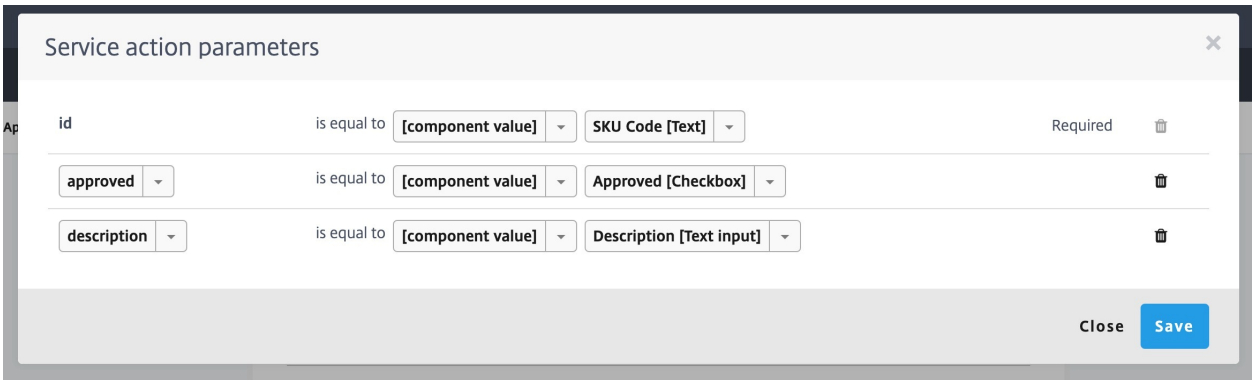
Designing the Page Overview

We want to leave our Approved as a check box (True or False) and our Description Text input fields as we want our Marketing Manager to make changes to these fields.

Finally, we need to add a Button element to the bottom of our page. We will then link this button to our Service Action so we can send the updated product description to the database.

Add the button from the Input pallet to the bottom of the page. Select the button and the "Button properties" blade will be displayed. Expand "Actions", select "Page action button". From the "Add action…" pulldown, select "Run service action". This configures the button to do something when selected by the user, in this case, it'll run the specified Service Action. Select the "Products" table from the "Data" pulldown, and select the "Approve Change" Service action from the "Action" pulldown.

We now need to configure what data we will pass the Service Action parameters we configured in "Add Service Actions", click the "Edit Parameters" button.



Configuring Service Action Parameters

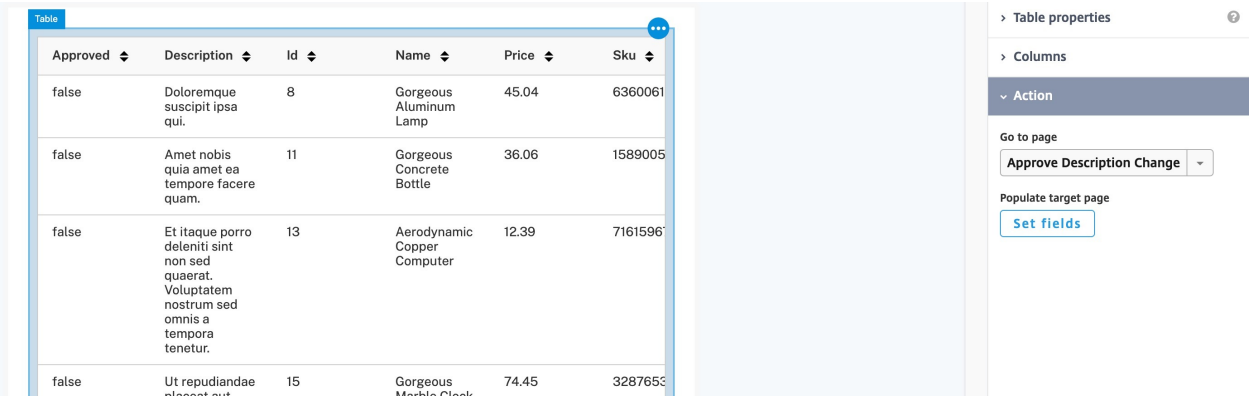
The Service action parameters box will be displayed, all we do is simply align the fields we have configured on the page with the Service action parameters we've already specified. For example, we make `Id is equal to [component value] id [Text]`

We then do the same for "Approved" and "Description", making sure they are equal to the correct UI elements from the page. Click "Save" when you're done.

Creating an Approvals Page

Follow the steps to create a new page, instead of adding a "Form" page, add a "Table" page and call it "Outstanding Approvals". Select "Products" as the data source and select the "Products" data table, add all the fields and click "Set Fields".

In your newly configured "Outstanding approvals" page, select the table that the designer has created, and select "Action" from the right-hand blade, in the "Goto page" pull down, select the previous page we just created (Approve Description Change).



Selecting the Page Design

We now need to enable our Microapp as a user initiated action, go back to the Products Description Approvals Microapp by following the breadcrumb menu. Select Properties from the menu on the left and click the "Enable as action" toggle and select the Action page as "Outstanding Approvals". This will allow the user to select the Product Description Approvals Action from their Workspace and be presented with the table of all approvals that are not completed (The approved field being "FALSE"). When they click on a record, they'll be taken to the Approve Product Description page to approve the change.

Creating a Notification

Notifications appear in the users activity feed and are events from the system of record. In our example, we're using events to notify the Marketing Manager that a new Product has been added to the database and requires a Product Description approval and also the Sales Reps that a Product has an updated description.

When configuring Notifications ensure that that the notification is actually relevant and useful, in a production environment, there may be several Systems of Record connected to the users Workspace. Filling the activity feed full of notifications that are not useful leads to a poor user experience. We'll focus on creating the first Notification so the Marketing Manager receives a notification of when a new Product is added to the database and requires a product description approval.

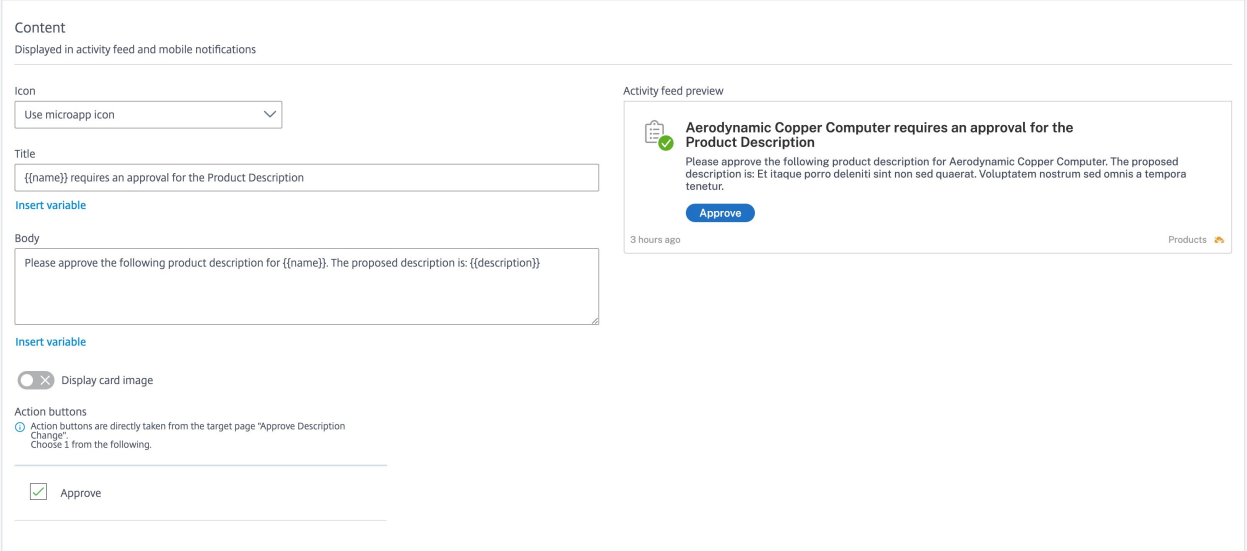
From the Product Description Approvals Microapp, select Notifications. Click "Add New Notification". The "New Notification" blade will open. Give the notification a name.

There are different event triggers that can be used, for our notification, we're going to use "New Records", so each time a new product record is added, we'll get a notification. Notifications will only appear for new products when they are synchronized with the data cache. If you add a new product to the MySQL database, be sure to perform a Data Synchronization to load those new products into the cache.

Select the data source as the "Products" data source and the data table as "Product Table", click "Add". You will now be taken to the "Edit Notification" page.

From the “Edit Notification” page, select the “Automatically run this event after the integration data change”. This will run the notification whenever the data cache is synchronized with the MySQL database.

We can now build what our Notification will look like in the “Content” box, as you build the Content, a preview is displayed under “Activity feed preview”.



Building the Notification Content

In the “Title” field, enter:

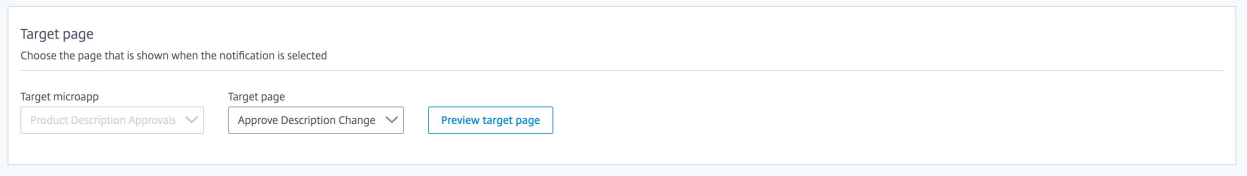
```
{{name}} requires an approval for the Product Description
```

In the Body field enter a descriptive message for the end user, for example

```
Please approve the following product description for {{name}}.
The proposed description is: {{description}}
```

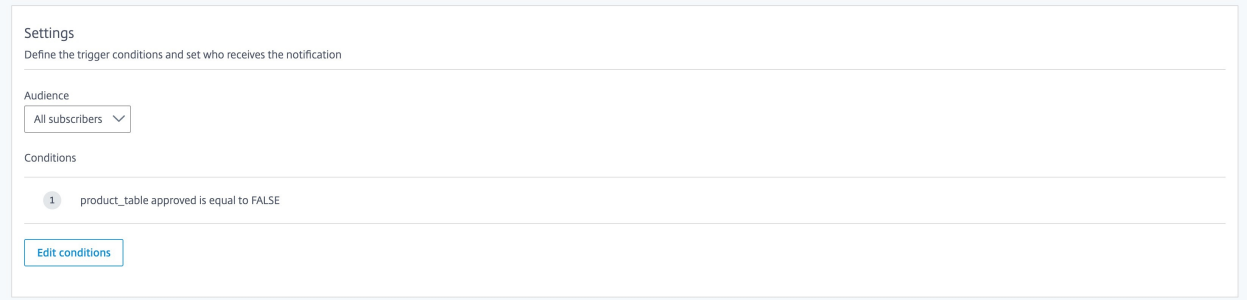
The double moustached variables relate to fields in the “Product” table, this allows you to get really creative in creating useful human readable notifications for end users.

Scroll down to the “Target page” box and select the “Target Page” as the “Approve Products” page we created earlier. This will enable an user to click on the notification in their activity feed and then open up the record directly in the blade from the Notification, there the user can review and edit the Product Description and approve.



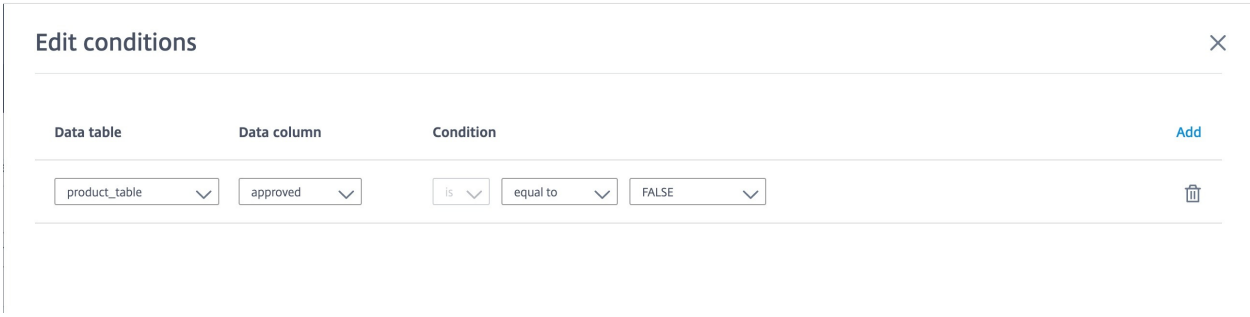
Choosing the Notification Target Page

We will configure our Notification so it appears for all subscribers to the Microapp, which is the default setting. We do need to add a condition however, so our notification only appears when the product description has not been approved.



Notification Settings

Click the “Edit conditions” button, the “Edit conditions blade will be displayed.



Notification Edit Conditions

Select the “Data table” as the product_table, “Data column as “approved” and the conditions as “is equal to FALSE”. Any record in the product table where the approved field is equal to FALSE will now trigger a notification for that record. Click “Save”.

Conditions for both displaying and deleting a notification allow you to fine tune when notifications appear and when they are deleted. This ensures that only notifications that are relevant to end-users are displayed.

We can set an expiration on our notifications, either when the record is deleted (i.e we remove a record from the `product_table` or after a specified time interval, or we can build a condition or multiple conditions. For the purposes of our example, we’re going to leave this as the default. Click “Save” from the top of the page to save the notification.

We've now completed the basic elements of our first Microapp.

Creating a Microapp for Sales Reps

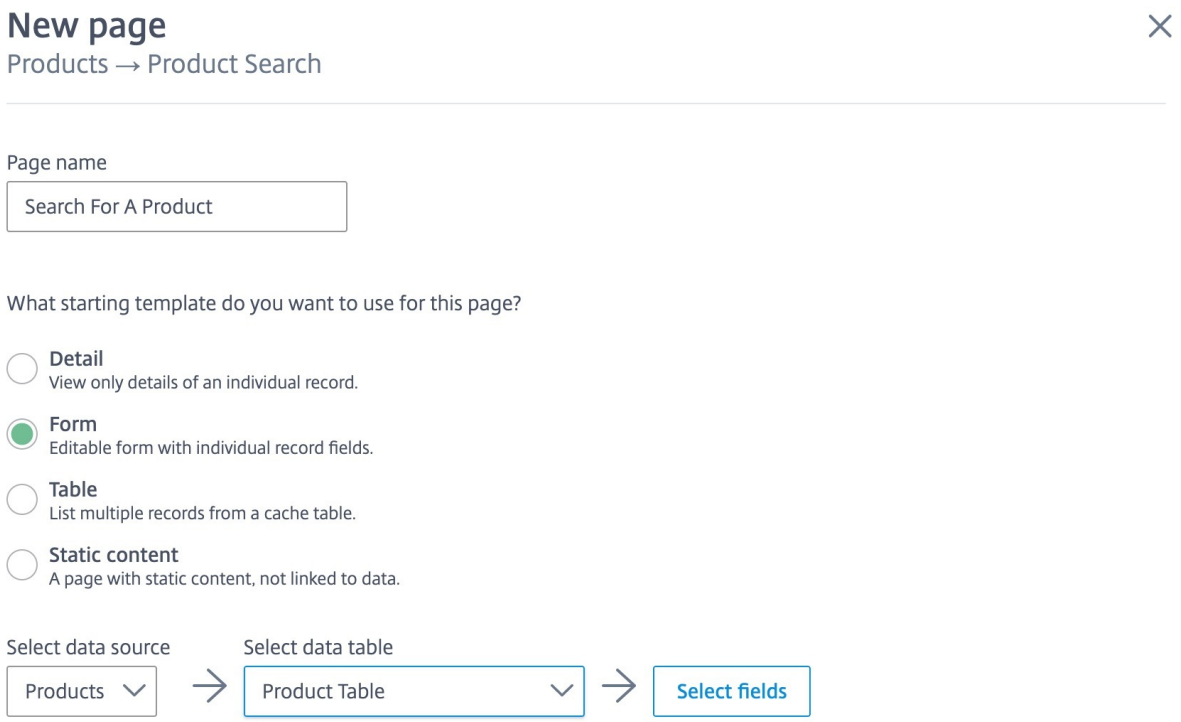
We've now gone through the basics of creating our first Microapp for the Marketing Manager, we now need to build additional Microapps for the Sales Reps so they can receive notifications of when a product has an updated description, and also search the Product database.

The Sales Rep Microapp will consist of the following:

- A Notification when a product has an updated Product Description
- A Page that displays the product with the updated description (this will be used by the Notification)
- A Page that is the Action Page to allow the Sales Rep to search the Products database.

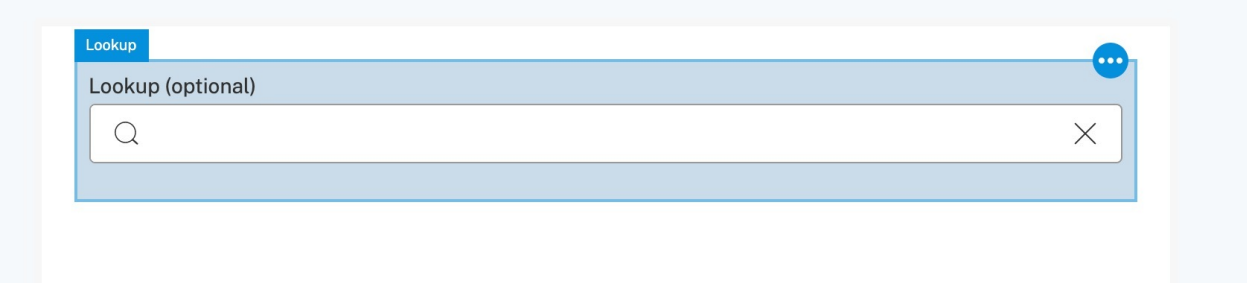
To build the Sales Rep Microapp:

- Create a Microapp for use by the Sales Reps.
- Add a page called "Display Product Detail", make this a "Detail" page, add the "Products" data source and all fields from the "Products" Table.
- Add a page called "Search Product Catalog", make this a "Form" page.



Adding Product Search

On the "Search Product Catalog" page, add a "Look Up" UI element.

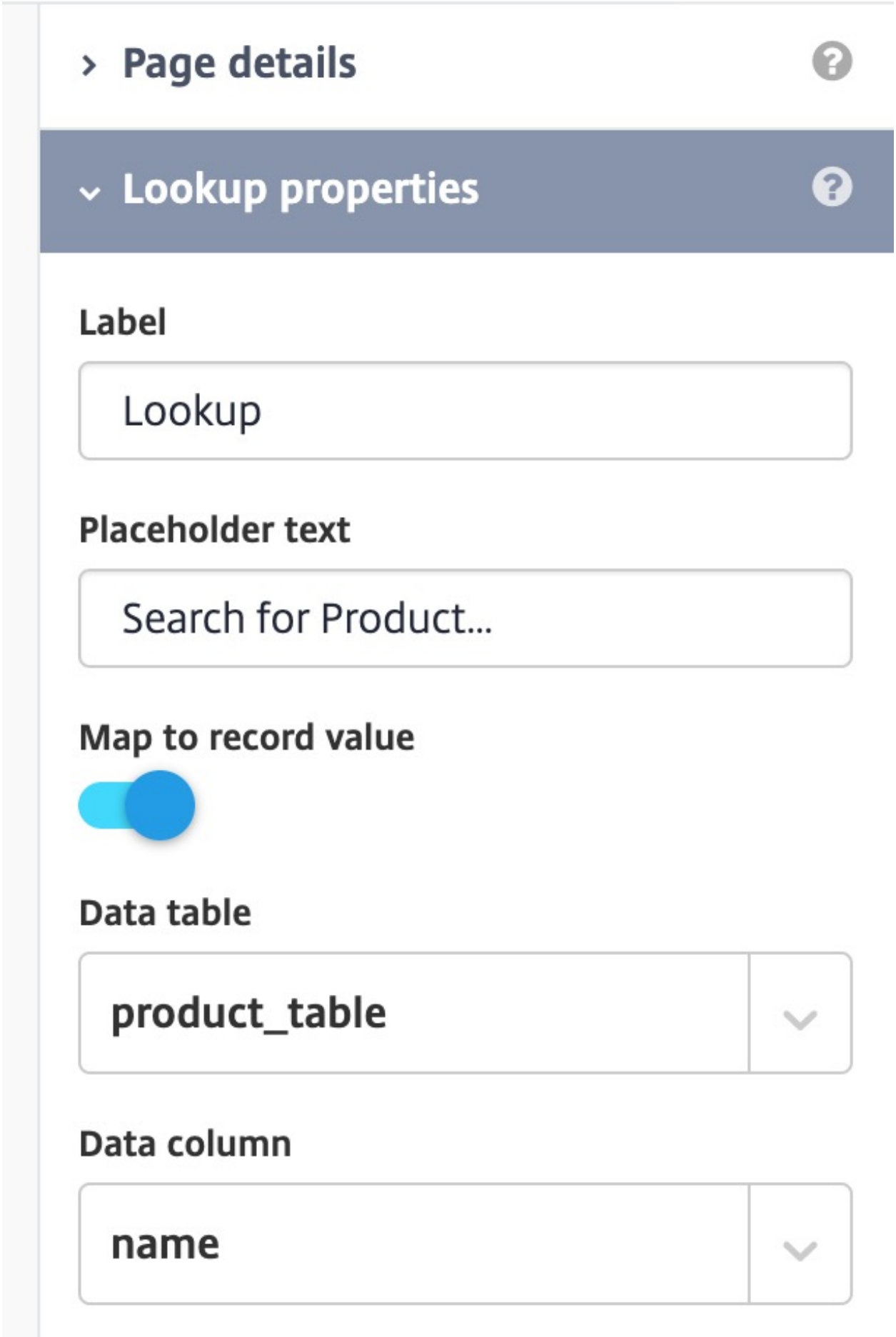


Adding a Lookup Element

Select the "Look Up" UI element properties and set the following values:

- Data table: `product_table`
- Data column: `name`
- Data source to search: `Products`
- Data table to search: `product_table`
- Data column to search: `name`

- Data column to use as value: `id`


Page details
Lookup properties
Label
Lookup
Placeholder text
Search for Product...
Map to record value
Data table
product_table
Data column
name

Create a new Notification called “New Product Description Notification”, configure it to run automatically after the integration data change, use the variables to create a useful notification for the Sales Rep when a product description has changed. Under settings, Select “Edit conditions” and select: `product_table approved becomes equal to TRUE`

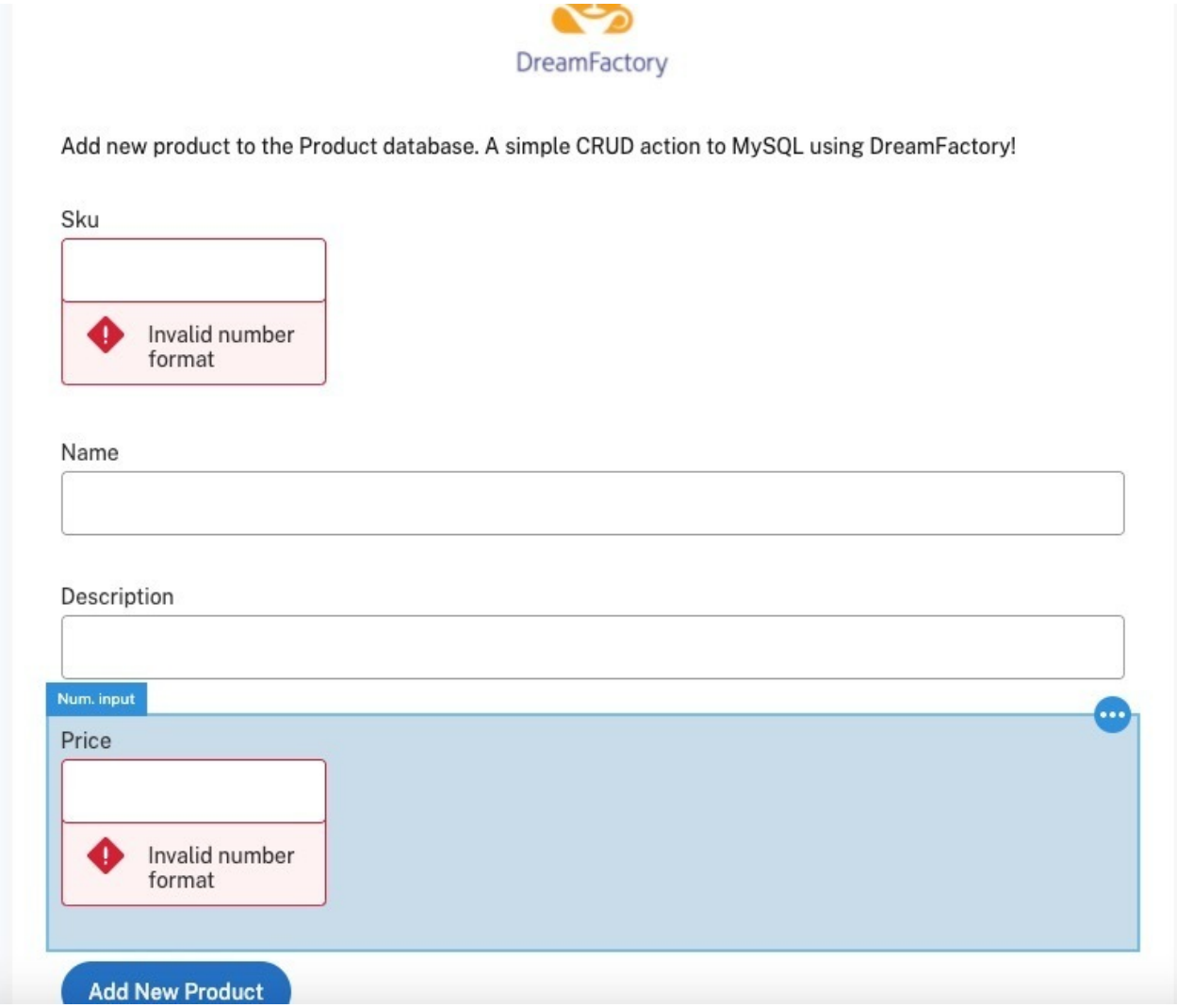
Creating a New Products Microapp

We’ve one last Microapp to add. In this Microapp, we’ll allow the Products Manager to add new Products to the database.

The Products Manager Microapp will consist of a “Form” Page to allow a new product to be added to the Products table.

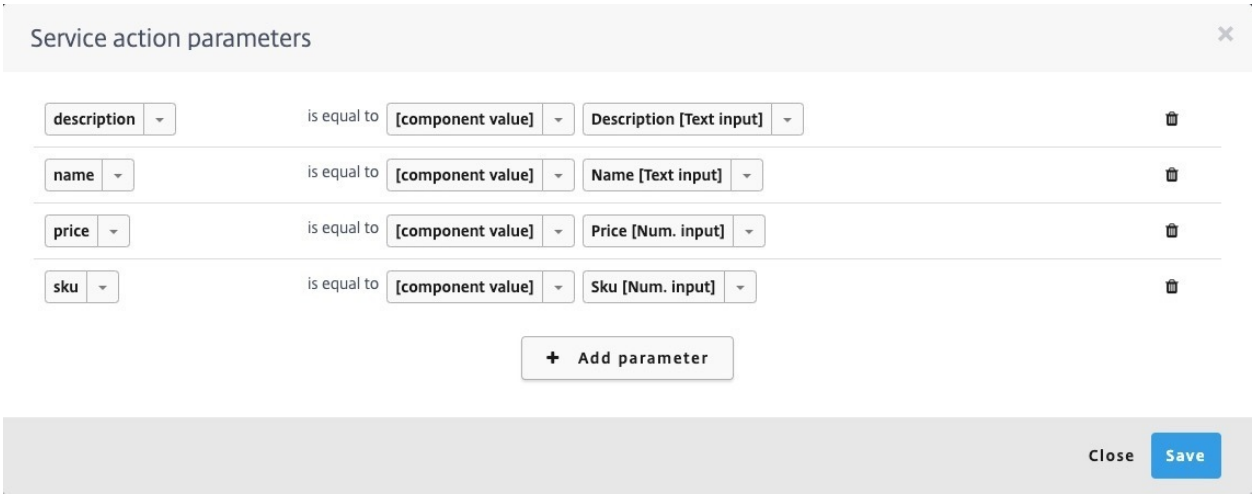
To build the Products Manager Microapp, follow these steps:

- Create a new Microapp for the Products Manager
- Add a new “Form” Page for that the Products Manager will use to add new products.
- Configure with the Text input fields:
- SKU (numeric)
- Name
- Description
- Price (numeric)



Adding a Product

Add a new Service Action button and configure the Data as “Products” and the action as per the Service Action that was created in section “Service Action - Add New Products”.



Product Manager Service Action Parameters

Once the page is complete, from the "Add new product" Microapp screen, select properties and ensure the "Enable as action" is selected and the "Add New Product" selected as the default page.

That concludes our basic Products Manager Microapp! As with all the Microapps we've built, you've many options to expand and improve them. You can also play around with the UI by adding in images, and descriptive text for what each page is therefore and help users to complete the workflows.

Adding Users to a Microapp

We subscribe users to each Microapp so the actions and notifications appear in their feed. In order to add users, you will need to ensure that Identity and Access Management is configured for Citrix Workspace.

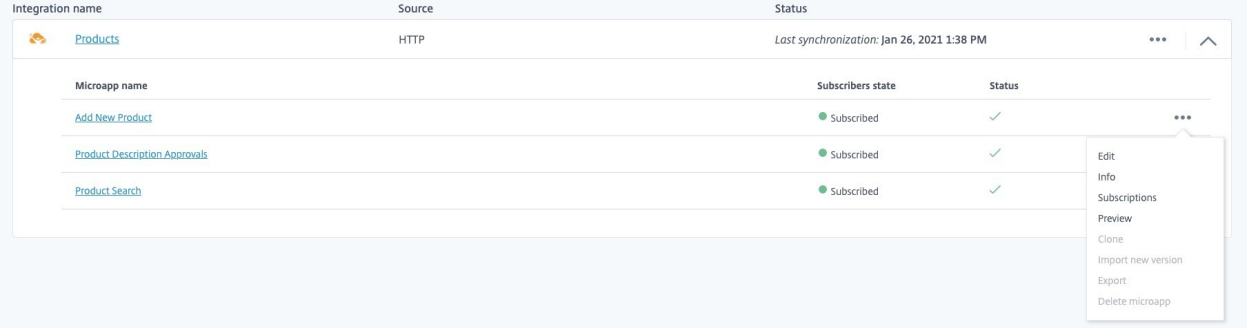
To use Azure Active Directory for Citrix Workspace authentication, see the following guides:

- https://docs.citrix.com/en-us/citrix-cloud/citrix-cloud-management/identity-access-management/connect-azure-ad.html#connect-citrix-cloud-to-azure-ad[9]
- https://developer.cloud.com/citrixworkspace/citrix-workspace-platform/build-workspace-microapp-integrations/docs/getting-started[10]

To add a user or group to a Microapp, select "Subscriptions" from the ellipsis menu:

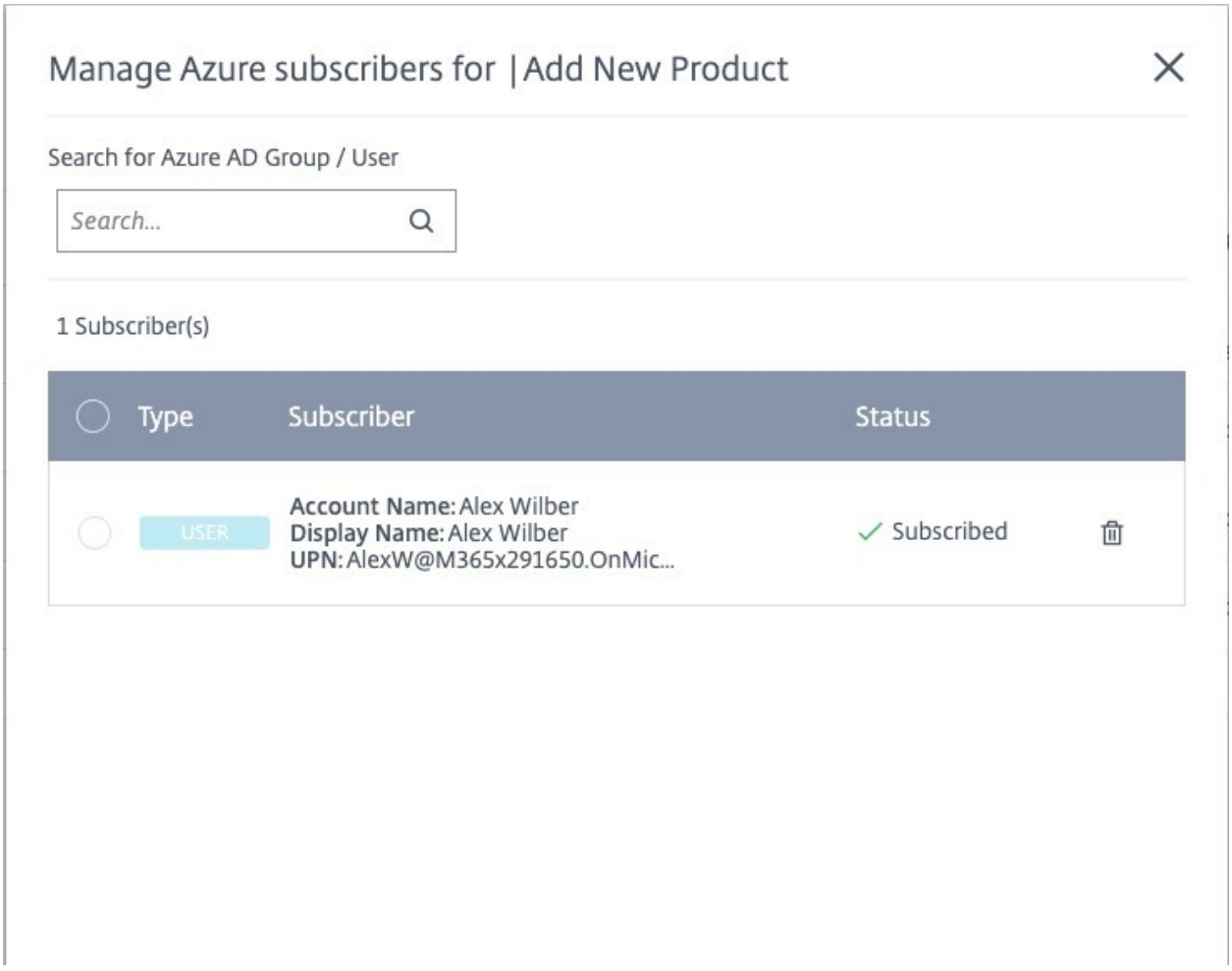
[9] https://docs.citrix.com/en-us/citrix-cloud/citrix-cloud-management/identity-access-management/connect-azure-ad.html#connect-citrix-cloud-to-azure-ad

[10] https://developer.cloud.com/citrixworkspace/citrix-workspace-platform/build-workspace-microapp-integrations/docs/getting-started



Adding Users

In the "Search" dialog box, enter the username or group name of the user or group you want to add. The username or group will appear as a subscriber.



Adding Subscribers Dialog

You can now login to your Citrix Workspace instance as your test user and test your Microapps!

Conclusion

Microapps can bring tremendous productivity gains to your enterprise. Thanks to solutions like Citrix Workspace and DreamFactory, it's possible to create and deploy powerful microapps faster than ever, and with remarkably little code. If you'd like to learn more about these solutions, contact John Moody (john.moody@citrix.com) and Jason Gilmore (jason.gilmore@dreamfactory.com)!