AppHub

Contents

Copyright GE Digital

About AppHub

AppHub Overview

You can use the AppHub service to combine multiple microapps into a unified interface with global navigation and a common mechanism for user authentication.

As web applications grow in size and complexity, it eventually makes sense to break a large application into smaller, separately created and deployed pieces known as microapps. Microapps are free of technical interdependencies, and can receive requests, apply appropriate logic, and produce a response. These functionalities are coordinated by using simple RESTful web service protocols to interact with a collection of loosely coupled microservices. The two protocols most commonly used are HTTP request-response with resource APIs and lightweight messaging. Authentication is typically implemented with API keys.

This means that you can use microapps with a bare minimum of centralized management, which supports application development using a variety of programming languages and data storage technologies. AppHub service instances provide the following features to manage microapps:

- Easy navigation within and between microapps
- Common user authentication and session management logic for microapps
- Notifications and alerts for microapps displayed to users
- Independent deployment and scaling for microapps
- Microapp reuse across product packages

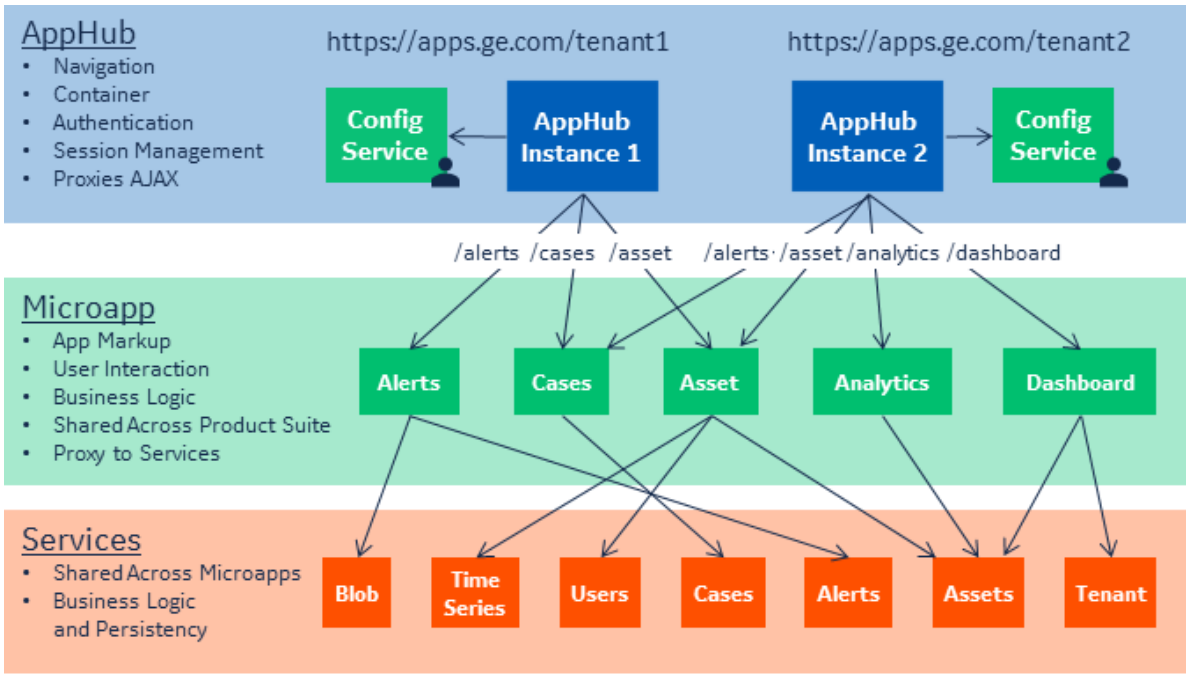

Figure 1: AppHub Overview

Related concepts

What is Predix Platform?

Predix Microservices

Get Started With AppHub

AppHub Service Setup

Specific accounts and software are required before you can complete the tasks needed to set up the AppHub service.

Before You Begin

Accounts

To use Predix services, you must have the following accounts.

- A Predix.io account. Go to https://www.predix.io/registration/.
  When you register for a Predix.io account, an org and space is created for you in Cloud Foundry.
- A Github account. Go to https://github.com/join.

Software

The following software is required to set up the AppHub service.

Table 1: Common Tools for Cloud Development

Software	Version	Description
Cloud Foundry CLI	Latest stable binary version	You use the Cloud Foundry CLI to deploy and manage applications and services. Download the latest stable binary from https://github.com/cloudfoundry/cli#downloads.
Git	Latest	Download Git from https://git-scm.com/downloads.

Note: The command-line syntax provided throughout this documentation is for Macintosh or Linux environments. To use the same syntax in Windows environments, use Git Bash.

About AppHub Authentication

Authentication for the AppHub service is controlled by the designated trusted issuer and is managed by the User Account and Authentication (UAA) web service, in the same manner as for other Predix platform services. You must set up a UAA service instance as the trusted issuer before you can set up the AppHub service. For more information, see About the User Account and Authentication Security Service.

Task Roadmap

Complete these tasks to set up the AppHub service.

#	Task	Information
1	(Optional) Configure your proxy settings.	Depending on your location and network configuration, you might need to configure your proxy settings to access remote resources. See Defining Proxy Connections to Remote Resources.
2	Create a UAA trusted issuer.	See Creating a Predix UAA Service Instance and Service Key on page 3.
3	Create an OAuth2 client.	See Creating an OAuth2 Client for AppHub on page 4.
4	Create an AppHub service instance.	See Creating a Predix AppHub Service Instance on page 6.
6	Create an AppHub service key.	See Creating an AppHub Service Key on page 8.

Creating a Predix UAA Service Instance and Service Key

You can create up to 10 instances of the UAA service in your space. If you need additional instances, you must delete an older unused instance and create a new one.

1. Use the Cloud Foundry CLI to log into Cloud Foundry.

   ```
   cf login -a <API_Endpoint>
   ```

   Note: If you are a GE employee, you must use the cf login --sso command to log into Cloud Foundry. After you enter your SSO, you will receive a one-time passcode URL. Copy this URL and paste it in a browser to retrieve your one-time passcode. Use this code with the cf command to complete the CF login process.

   Depending on your Predix.io registration, the value of <API_Endpoint> is one of the following:

   - Predix US-West
     https://api.system.aws-usw02-pr.ice.predix.io
   - Predix US-East
     https://api.system.asv-pr.ice.predix.io
   - Predix Europe
     https://api.system.aws-eu-central-1-pr.ice.predix.io

   For example,

   ```
   cf login -a https://api.system.aws-usw02-pr.ice.predix.io
   ```

2. Create a new predix-uaa service instance.

   ```
   cf create-service predix-uaa <service-plan> <uaa-instance-name>
   -c '{"adminClientSecret":"<admin-client-secret>"}'
   ```

   where:

   - cf is an alias for the cloud foundry CLI command.
   - <service-plan> is the service plan you want to select for your UAA instance.

- <uaa-instance-name> is the name you want to use for your UAA instance, for example, my-uaa-instance.
- -c is used to specify additional parameters. <admin-client-secret> is the character string you want to use as the client secret for your UAA instance.

Note: Cloud Foundry CLI syntax can differ between Windows and Linux operating systems. See the Cloud Foundry help for the appropriate syntax for your operating system. For example, to see help for the create service command, run cf cs.

Your UAA instance is created with:

- A client identifier (admin).

  Note: An admin client is created for bootstrap purposes. You can create additional clients to use with your application.
- The client secret that you specified when you created the service.

3. Create a new predix-uaa service key for the instance.

```
cf create-service-key <uaa-instance-name> <uaa-instance-name>-service-key
```

   where:

   - cf is an alias for the cloud foundry CLI command.
   - <uaa-instance-name> is the name you previously specified for your UAA instance.
   - <uaa-instance-name>-service-key is the name you want to use for your UAA instance's service key, for example, my-uaa-instance-service-key.
4. View the service key details.

```
cf service-key <uaa-instance-name> <uaa-instance-name>-service-key
```

5. To confirm successful UAA service instance creation, open a web browser, navigate to the dashboardUrl value in your UAA service key, and log into the **UAA Dashboard** with the admin client ID and client secret you defined for your new instance.

Creating an OAuth2 Client for AppHub

After you create a UAA service instance, you must create an OAuth2 client to manage user authentication and authorization for AppHub and create UAA users.

You use the **Client Management** tab of the **UAA Dashboard** to create an OAuth2 client for AppHub, and then use the **User Management** tab to create users.

1. In the Predix.io **Console** view, select the space in which you created your UAA service instance.
2. In the **Service Instances** page, select your UAA instance, and then select the **Configure Service Instance** option.
3. Log into the **UAA Dashboard** with your admin client secret, and then select the **Client Management** tab.

   The **Client Management tab** has two views, **Clients** and **Services** . The **Services** view displays only the services bound to the UAA instance that you are currently confguring.
4. Complete the **Create Client** form.

Field	Description
Client ID	Specify a name for the OAuth2 client you are creating, for example, apphub-oauthclient.
Authorized Grant Types	Select the following grant types: • **authorization_code**: The client directs the resource owner to UAA, which in turn directs the resource owner back to the client with the authorization code. • **client_credentials:** The OAuth2 endpoint in UAA accepts the client ID and client secret and provides access tokens. • **refresh_token:** Refresh tokens are credentials used to obtain access tokens. For this grant type, UAA provides a refresh token that you can then use to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope. For more information on grant types, see RFC 6749.
Client Secret	Specify the client secret. Be sure to make note of this password, which cannot be retrieved if lost.
Confirm Client Secret	Re-enter the client secret.
Redirect URI	Specify a URI (Unique Resource Identifier) to which to redirect the client after login or logout. When you use UAA as the service provider for your external identity provider. this value is provided for the /oauth/authorize and /logout endpoints. The **Redirect URI** value is required when you use the **authorization_code** grant type. This grant type uses the **Redirect URI** value as the endpoint or callback for your application that requires user authorization code. Unique Resource Identifiers consist of: • Access protocol, such as http or https • Domain or IP address • Access port, such as 80 or 443 • Path When you create an OAuth2 client for AppHub, set **Redirect URI** to http://localhost:3000.
Scopes	Scopes are permissions associated with an OAuth client to control user access to a resource through an application for the **authorization_code** grant type. By default, the admin client is assigned all required scopes. You can add scopes based on client requirements at creation time, or modify these settings later on. When you create an OAuth2 client for AppHub, the **uaa.user** scope is assigned by default. Do not add any other scopes at this time. For a list of available scopes, see Scopes Authorized by the UAA.
Authorities	Authorities are permissions for the **client_credentials** authorization grant type. These permissions are associated with an OAuth2 client when an application or API accesses a resource with its own credentials without user involvement. By default, the admin client is assigned all required authorities. You can add authorities based on client requirements at creation time, or modify these settings later on. When you create an OAuth2 client for AppHub, the **uaa.user** authority is assigned by default. Do not add any other authorities at this time. The list of authorities matches the list of scopes. For a list of available UAA scopes, see Scopes Authorized by the UAA. **Note:** Admin clients are not assigned the authority to change the user password by default. To change the user password, you must add the **uaa.admin** authority to your admin client.
Auto Approved Scopes	(Optional) These are scopes that can be automatically approved for the client without explicit approval from a resource owner. When you create an OAuth2 client for AppHub, leave this field empty.

Field	Description
Allowed Providers	(Optional) These are the names of the external identity providers, if any. This field is required if you use external identity providers with UAA as a service provider. When you create an OAuth2 client for AppHub, leave this field empty.
Access Token Validity	(Optional) This is the access-token expiration time in ms. When you create an OAuth2 client for AppHub, leave this field empty.
Refresh Token Validity	(Optional) This is the refresh-token expiration time in ms. When you create an OAuth2 client for AppHub, leave this field empty.

5. Click **Save**, and then in the **User Management** tab, click **Create User**.
6. Complete the **New User** form.

Field	Description
Regular User	Choose this option to create local users in your UAA instance when you are not using an external identity provider (IdP).
Shadow User	Choose this option to create local users in UAA that correspond to users defined in your external IdP. This is useful if you need to create a whitelist to authenticate only a subset of users set up in your identity provider. For more information, see configuring a new IdP.
User Name	Specify a username. If you are creating a shadow user, this value must match the user name defined in your IdP.
Email	Specify an email address. If you are creating a shadow user, this value must match the email address defined in your IdP.
Password	Specify a password. Administrators can set password policies. For more information, see Creating Password Policies. This option is not required if you are creating a shadow user.
Given Name	(Optional) Specify the user's first name.
Family Name	(Optional) Specify the user's last name.
Origin	Specify the IdP to which this user is configured. This option is available only when you create a shadow user.
Groups	When you create an OAuth2 client for AppHub, leave this field empty. For more information on groups, see Creating Groups in a UAA Instance.
Active	Select this option to allow the newly created user to log in.
Verified	Select this option to verify this user with an autogenerated email invitation sent from UAA at the time of account creation.

Creating a Predix AppHub Service Instance

To complete this task, you must first create a UUA service instance, create an OAuth2 client, and add users.

1. Use the Cloud Foundry CLI to log into Cloud Foundry.
2. Copy the required JSON payload template in the following example, add appropriate values, and save the file.

```
{
   "uaa": {
      "uaaUri": "",
      "clientID": "",
      "clientSecret": ""
   },
```

```
  "routeInfo": {
    "hostName": "",
    "shared": true/false,
    "context": ""
  },
  "appConfigURL": "",
  "customHeader": {},
  "applicationChrome": true/false,
  "apphubKey": ""
}
```

Table 2: JSON Payload Values

Key	Value	Required
uaa	Array of uaaUri, clientID, and clientSecret	n/a
uaaUri	The URI of your UAA service instance.	Yes
clientID	The login username for the UAA client.	Yes
clientSecret	The client secret for the UAA client.	Yes
routeInfo	Array of hostName, shared, and context	n/a
hostName	The host server for your UAA instance. Specify a string that includes only alphanumerics, hyphens, and underscores.	Yes
shared	true or false.	Yes
context	If shared is set to true, this value must also be set to true. If not, this value is optional and can be left blank.	Yes if shared is set to true, No otherwise.
appConfigURL	The URL for any service that returns the JSON payload required to AppHub to render the UI. If not specified, this value defaults to ARCS. For more information about ARCS, see About AppHub Microapp Configuration on page 13.	No
customHeader	If specified, this value must be a valid JSON object.	No
applicationChrome	true or false.	Yes
apphubKey	The service key for your AppHub instance.	No

3. Create a new predix-apphub service instance.

```
cf create-service predix-apphub <service-plan> <apphub-instance-name> -c <payload.json>
```

where:

- cf is an alias for the cloud foundry CLI command.
- <service-plan> is the service plan you want to select for your AppHub instance. The available plans for AppHub are Beta and Standard.

- <apphub-instance-name> is the name you want to use for your app instance, for example, my-apphub-instance.
- -c is used to specify additional parameters. <payload.json> is the fully qualified path to the JSON file you created in the previous step.

Note: Cloud Foundry CLI syntax can differ between Windows and Linux operating systems. See the Cloud Foundry help for the appropriate syntax for your operating system. For example, to see help for the create service command, run cf cs.

Creating an AppHub Service Key

You can retrieve your AppHub service instance details by creating a service key. You can then update your OAuth client with the information that you retrieve.

1. Use the Cloud Foundry CLI to log into Cloud Foundry.
2. Create the AppHub service key.

   ```
   cf create-service-key <apphub-instance-name> <apphub-instance-name>-service-key
   ```

   where:

   - <apphub-instance-name> is the name of your AppHub service instance.
   - <apphub-instance-name>-service-key is the name of the AppHub service key.

   Note: Cloud Foundry CLI syntax can differ between Windows and Linux operating systems. See the Cloud Foundry help for the appropriate syntax for your operating system. For example, to see help for the create service command, run cf cs.
3. View your newly created AppHub service key.

   ```
   cf service-key <apphub-instance-name> <apphub-instance-name>-service-key
   ```
4. Log into the **UAA Dashboard** for your UAA service instance, and then in the **Client Management** tab, select the OAuth2 client you created for your AppHub instance and add the following details from your AppHub service key:
   - **Scope:** The value of the oauth-scope attribute.
   - **Auto Approved Scope:** The value of the oauth-scope attribute.
   - **Authorities:** The value of the oauth-scope attribute.
   - **Redirect URI:** The value of the predix_apphub_url attribute.
5. Confirm that your AppHub service instance is authorized as a service within your UAA client. If needed, in the **Choose Service** box, add the instance manually.

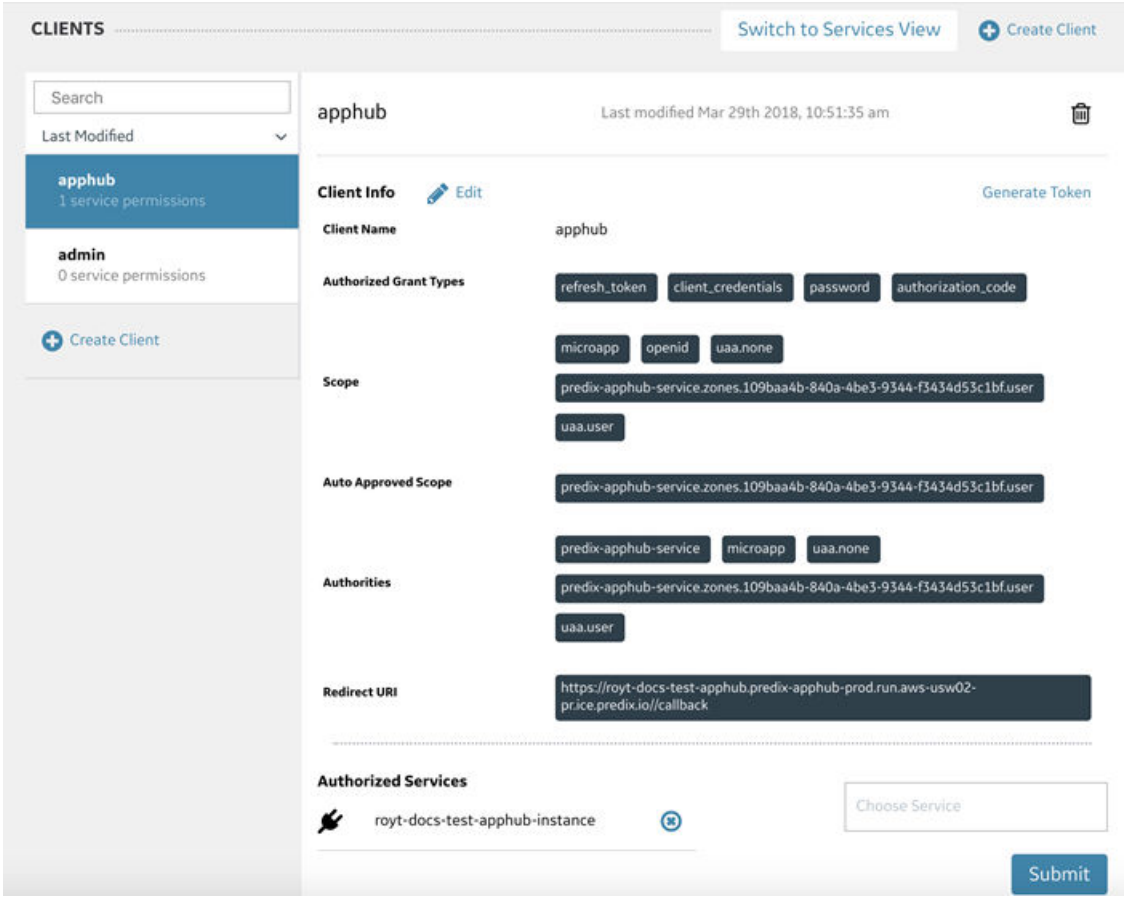

Figure 2: UAA Dashboard

6. Copy the value of the oauth-scope attribute from your AppHub service key, and then in the **User Management** tab, paste the value into the **Groups** field for each user.
7. Navigate to the predix_apphub_url value from your AppHub service key and log in as one of the UAA users you previously created.

   An AppHub page appears with only the default **Settings** microapp configured if you are using ARCS as your backend configuration service. For more information about ARCS, see About AppHub Microapp Configuration on page 13.

You can now add microapps and configuration settings to your AppHub instance. For more information, see About AppHub Microapp Configuration on page 13.

Working With Microapps

About Microapps and Predix Development

Microapps are typically single-page Web applications with GUIs that address specific tasks to produce results. For example, you can create microapps to register users for training, or report system status or device space allocations.

With AppHub, you can combine microapps into a unified UI with a single point of entry. You can use the provided microapp reference applications to easily start building microapps using Node.js. Additionally, you can load any existing microapps into the AppHub content area. AppHub includes the index.html for each microapp in the content section in its template, and proxies any further resource requests directly to the microapp itself.

Microapp Approach

A single-page application uses Asynchronous JavaScript (AJAX) requests and HTML5 to create fluid and responsive user interactions without constant page reloads. However, much of this work running in a browser occurs as complex state transitions following the Model View Controller architecture (see Modern Web Application for more information):

- There are global state changes (such as going offline in a real time application) .
- There are delayed results from AJAX requests that get returned at some point from back end operations
- There are changes to the Model function as values change.
- There are Domain Object Model (DOM) events that cause small state changes in the View function.
- There are changes to the Controller function that cause state changes in the View function..

The microapp approach implements a conventional single-page application as a suite of microservices. Microservices are atomic, self-contained services that typically perform a single operation on a back-end system, such as a retrieving a customer record.

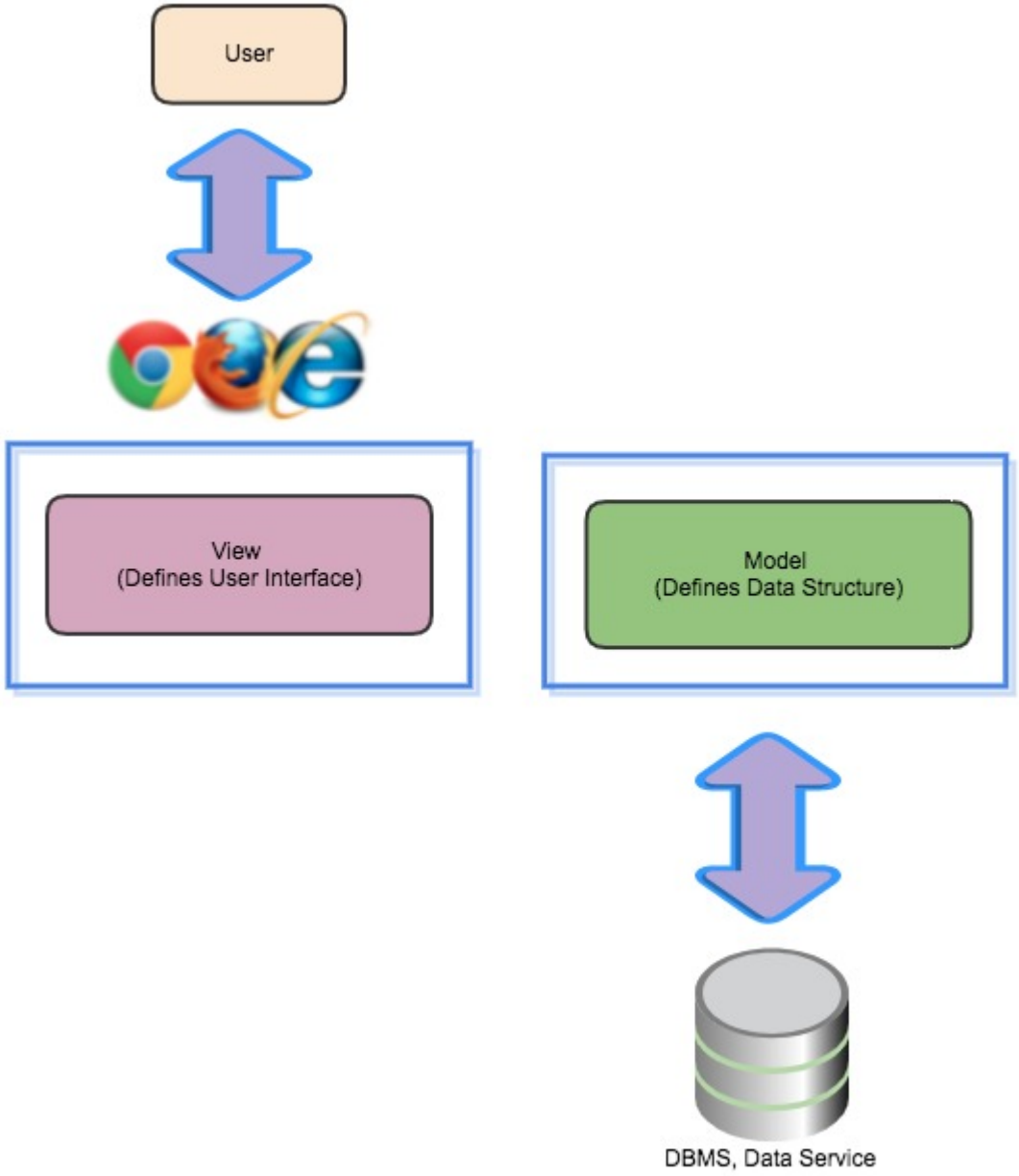

Figure 3: Microapp Approach

This means that microapps provide a range of capabilities that can be focused on specific tasks. For example, end users benefit from a View function that is tailored to their specific requirements.

See the following resources for more information:

- Why Application Development is Going Micro
- Understanding the Micro Apps Trend in Application Development
- About Microapps and Predix Development on page 10

Using Custom Microapps

You can create your own custom microapps or port existing microapps and easily integrate them with AppHub.

AppHub provides a seed microapp that you can use to easily start building your own custom microapps. The seed microapp is a blank stub that you can use as a starting point for custom microapp development. You can also port existing microapps that fit AppHub technical requirements.

To complete the following steps, be sure to log into the same Cloud Foundry environment you used to create your UAA and AppHub service instances.

1. Clone the repository for the seed microapp into your current working directory, change to the root directory of the cloned project, and check out the master branch.

   ```
   git clone https://github.com/predix/apphub-microapp-seed.git
   cd <seed-microapp-name>
   git checkout master
   ```

   Note: For more details, see the README files in the seed microapp repo.

2. Develop your microapp in the coding language of your choice, or modify and port an existing microapp.

   Keep the following points in mind:

   - The index page for your microapp should be similar to this boilerplate HTML, and should not include <html> and <body> tags.
   - Do not use CORS (cross-origin resource sharing).
   - The look and feel of your microapp should be compatible with that of AppHub.

   Note: When developing or porting microapps, always use relative URLs with no leading slashes. In AppHub, a path is prepended to microapp URLs to distinguish them from \the other microapps running in the same instance. The HTML tag <script src="foo/bar"> loads correctly in AppHub, while the tag <script src="/foo/bar"> generates an error, because the path cannot be correctly prepended with AppHub URL information.

3. Navigate to the root of the folder and install dependencies.

   ```
   npm install
   npm run bower
   ```

4. Confirm that the buildpack, configuration details, and environment variables for your microapp are correctly listed in manifest.yml, and then in the services section, list the Predix services that your microapp uses, specifically the UAA and AppHub instances that you previously created.

   Note: The services listed in manifest.yml are automatically bound to your AppHub instance when you push the microapp to Cloud Foundry.

   ```
   ---
   applications:
     - name: my-apphub-microapp-react-px-polymer-example
       buildpack: https://github.com/cloudfoundry/heroku-buildpack-nodejs.git
       memory: 512M
       stack: cflinuxfs2
       instances: 2
       path: .
       command: node server
       services:
         - my-microapp-seed-px-react-ge-twitter-uaa
         - my-microapp-seed-ge-twitter-apphub
   ```

```
env:
  NODE_ENV: 'production'
  REQUEST_LIMIT: 500kb
  SESSION_SECRET: mySecret
  COOKIE_NAME: myCookie
  SWAGGER_API_SPEC: /spec
```

5. Start the NodeJS development server with Webpack middleware.

```
npm run dev
```

6. To confirm that the microapp is running locally, in a web browser, navigate to http://localhost:9000.
7. Test client-server connectivity.

```
npm test
```

   - test:client: Use this option for client-side testing using Jest with snapshots and code coverage.
   - test:server: Use this option for server-side testing using Mocha with code coverage.

8. Build the microapp client and server for production.

```
npm run dist
```

9. From the root directory of your microapp, push your microapp to Cloud Foundry, and then confirm that the microapp is listed among the other apps in your Cloud Foundry environment.

```
cf push
cf apps
```

You can now configure your microapp to AppHub. For more information, see About AppHub Microapp Configuration on page 13.

About AppHub Microapp Configuration

After you create microapps, you must perform a few configuration tasks to associate them to your AppHub service instance.

You can choose to use ARCS (App Registry and Configuration Service) or a different configuration service to associate your microapps to your AppHub service instance. To make configuration requests, you use RESTful APIs for ARCS or your selected configuration service to configure microapps, tenants, themes, retrieval tokens, and so on. When you use ARCS as your configuration service, you can also make configuration settings by using the Settings microapp.

ARCS provides you with the ability to:

- Configure and persist microapp information.
- Configure and persist tenant configurations, for example, UAA, app name, logo, custom headers, themes, and so on.
- Configure and persist microapp-to-tenant mapping.
- Configure and persist microapp navigation information.
- Configure and persist preferences.
- Configure and persist details to retrieve user information.
- Create users for UAA instances at the tenant level.
- Map microapps to users or groups.
- Grant or restrict microapp access at the tenant level.

Adding and Configuring Microapps

The Settings microapp, a UI that you can use to configure services via ARCS, will be available with the GA release of AppHub. With the beta release of AppHub, you can use ARCS from the command line or import a dedicated Postman collection to add and configure microapps.

Note: For details on supported ARCS endpoints, see the API documentation.

1. Copy the value of the predix_apphub_config_uri attribute in your AppHub service key, and use it for the {{arcsURL}} value in the next steps.

   ```
   cf service-key <apphub-service-instance-name> <apphub-service-key-name>
   ```

2. POST your microapp to {{arcsUrl}}/apps with the correct header and body information.

   For example, to configure the react-polymer-px reference microapp:

   ```
   curl -X POST \
     {{arcsURL}}/apps \
     -H "authorization: Bearer {UAA_CLIENT_TOKEN}" \
     -H "content-type: application/json" \
     -H "predix-zone-id: {APPHUB_SERVICE_ZONE_ID}" \
     -d '[{
           "uri": "/apps/polymer-react",
           "id": "polymer-react",
           "host": "https://apphub-microapp-react-px-polymer-example.run.aws-usw02-pr.ice.predix.io",
           "path": "/polymer-react",
           "template": "/index.html",
           "location": "main",
           "navService": null,
           "items": [{
                 "label": "Polymer-React-Seed",
                 "path": "/polymer-react/#"
              }
           ],
           "order": 4,
           "default": false
       }
   ]'
   ```

3. Navigate to the AppHub URL listed in your AppHub service key, log in with your UAA user credentials, and verify that your microapp is properly configured and associated with your AppHub instance.

Configuring AppHub

Making Global Configuration Settings From the Command Line

You can make global configuration settings for your AppHub instance from the command line.

Global configuration settings include:

- Display name and logo
- Theme
- Enabled or disabled application chrome
- JavaScript injection

To make global configuration settings from the command line, you use the POST command to send a JSON payload to the URL for your configuration service.

- To make global configuration settings, use the following JSON payload.

```
curl -s -X POST <apphub-config-url>/globalconfig \
  -H "authorization: bearer <uaa-client-token>" \
  -H "content-type: application/json" \
  -H "predix-zone-id: <apphub-service-zone-id>" \
  -d '{
      "name": "<apphub-display-name>",
      "logo": true | false,
      "logoUri": "<logo-uri>",
      "applicationChrome": true | false,
      "globalScripts": [<script-url>]
   }'
```

Table 3: Global Configuration JSON Options

Key	Value	Description
authorization: bearer	<uaa-client-token>	The token for the UAA client for your AppHub instance.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.
name	<apphub-display-name>	The name for your AppHub instance that you want to show at the top of each microapp page.
logo	true or false	Set to true to show a custom logo at the top of each AppHub microapp page, or false to disable logo display. If you leave this option empty, the default AppHub logo appears.
logoUri	<logo-uri>	The URI to an image file to use as a logo next to the instance name.
applicationChrome	true or false	Set to true to show the left navigation pane, or false to hide it.
globalScripts	<script-url>	The fully qualified URL to a JavaScript file you want to inject into your AppHub instance.

About Configuring Themes From the Command Line

You can add, configure, and maintain themes from the command line.

Whether you use ARCS or another configuration service for your AppHub instance, you can perform the following theme configuration and maintenance tasks from the command line:

- Add themes
- Update themes
- Retrieve theme configuration settings
- Delete themes

If you are using ARCS, you can also perform many of these tasks by using the Settings microapp.

Adding Themes From the Command Line

You can add themes to your AppHub instance from the command line.

To add themes from the command line, you use the POST command to send a JSON payload to the URL for your AppHub instance. If ARCS is your configuration service, you can also use the Settings microapp to add themes.

- Use the following JSON payload:

```
[
   {
     "baseUri": "<apphub-uri>",
     "main": "<main-template>",
     "error": "<error-template>",
     "errorChromeless": "<errorchromeless-template>",
     "options": {
        "<option-name>": "<option-value>"
     },
     "displayName": "<theme-name>",
     "demoLink": "<theme-demo>",
     "description": "<theme-description>"
  }
]
```

You can use the following cURL command:

```
curl -X POST \
  <arcsurl>/themes \
  -H 'authorization: <uaa-token>' \
  -H 'cache-control: cache | no-cache' \
  -H 'content-type: application/json' \
  -H 'predix-zone-id: <apphub-service-zone-id>' \
  -d '[
     {
       "baseUri": "<apphub-uri>",
       "main": "<main-template>",
       "error": "<error-template>",
       "errorChromeless": "<errorchromeless-template>",
       "options": {
          "<option-name>": "<option-value>"
       },
       "displayName": "<theme-name>",
       "demoLink": "<theme-demo>",
```

```
    "description": "<theme-description>"
  }
]'
```

Table 4: Themes Configuration Options

Key	Value	Description
baseUri	<apphub-uri>	The URI where your AppHub instance is hosted.
main	<main-template>	The relative path to the HTML template to use for the main page view for the microapp.
error	<error-template>	The relative path to the HTML template to use for errors.
errorChromeless	<errorchromeless-template>	The relative path to the HTML template to use for chromeless errors.
options	Array of <option-name> and <option-value>	Names and values of theme options, for example, "color":"dark". You can define multiple options.
displayName	<theme-name>	The name to use for this theme in the UI.
demoLink	<theme-demo>	The URL to a demo AppHub instance with this theme applied.
arcsurl	<arcsurl>	The URL for your ARCS server or other configuration service.
description	<theme-description>	A brief description of the theme.
authorization	<uaa-token>	The token for the UAA client for your AppHub instance.
cache-control	cache or no-cache	Set cache to enable caching or no-cache to disable caching for this microapp.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.

Updating Themes From the Command Line

You can ujpdate themes in your AppHub instance from the command line.

To update themes from the command line, you use the PUT command to send a JSON payload to the URL for your AppHub instance. If ARCS is your configuration service, you can also use the Settings microapp to update themes.

- Do one of the following:
  - Use the following JSON payload:

```
[
  {
    "uri": <theme-uri>",
    "baseUri": "<apphub-uri>",
    "main": "<main-template>",
    "error": "<error-template>",
    "errorChromeless": "<errorchromeless-template>",
    "options": {
```

```
        "<option-name>": "<option-value>"
      },
      "displayName": "<theme-name>",
      "demoLink": "<theme-demo>",
      "description": "<theme-description>"
   }
]
```

- Use the following cURL command:

```
curl -X PUT \
  <arcsurl>/themes \
  -H 'authorization: <uaa-token>' \
  -H 'cache-control: cache | no-cache' \
  -H 'content-type: application/json' \
  -H 'predix-zone-id: <apphub-service-zone-id>' \
  -d '[
     {
      "baseUri": "<apphub-uri>",
      "main": "<main-template>",
      "error": "<error-template>",
      "errorChromeless": "<errorchromeless-template>",
      "options": {
         "<option-name>": "<option-value>"
      },
      "displayName": "<theme-name>",
      "demoLink": "<theme-demo>",
      "description": "<theme-description>"
    }
]'
```

Table 5: Themes Update Options

Key	Value	Description
uri	<theme-uri>	The URI where the theme is hosted.
baseUri	<apphub-uri>	The URI where your AppHub instance is hosted.
main	<main-template>	The relative path to the HTML template to use for the main page view for the microapp.
error	<error-template>	The relative path to the HTML template to use for errors.
errorChromeless	<errorchromeless-template>	The relative path to the HTML template to use for chromeless errors.
options	Array of <option-name> and <option-value>	Names and values of theme options, for example, "color":"dark". You can defiine multiple options.
displayName	<theme-name>	The name to use for this theme in the UI.
demoLink	<theme-demo>	The URL to a demo AppHub instance with this theme applied.
description	<theme-description>	A brief description of the theme.

Key	Value	Description
arcsurl	<arcsurl>	The URL for your ARCS server or other configuration service.
authorization	<uaa-token>	The token for the UAA client for your AppHub instance.
cache-control	cache or no-cache	Set cache to enable caching or no-cache to disable caching for this microapp.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.

Retrieving Theme Configuration Settings

You can retrieve theme configuration settings for your AppHub instance from the command line.

To retrieve theme configuration settings for your AppHub instance, you use cURL with the GET command to send a JSON payload to the ARCS URL for your instance.

- Use the following cURL command:

```
curl -X GET \
  <arcsurl>/themes \
  -H 'authorization: <uaa-token>' \
  -H 'cache-control: cache | no-cache' \
  -H 'content-type: application/json' \
  -H 'predix-zone-id: <apphub-service-zone-id>'
```

Table 6: Themes Configuration Options

Key	Value	Description
arcsurl	<arcsurl>	The URL for your ARCS server or other configuration service.
authorization	<uaa-token>	The token for the UAA client for your AppHub instance.
cache-control	cache or no-cache	Set cache to enable caching or no-cache to disable caching for this microapp.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.

Deleting Themes From the Command Line

You can delete themes from your AppHub instance from the command line.

To delete themes from your AppHub instance, you use cURL with the DELETE command to send application details to the ARCS URL for your instance.

- Use the following cURL command:

```
curl -X DELETE \
  <arcsurl>/themes \
  -H 'authorization: <uaa-token>' \
  -H 'cache-control: cache | no-cache' \
  -H 'content-type: application/json' \
```

```
-H 'predix-zone-id: <apphub-service-zone-id>'
```

Table 7: Themes Configuration Options

Key	Value	Description
arcsurl	<arcsurl>	The URL for your ARCS server or other configuration service.
authorization	<uaa-token>	The token for the UAA client for your AppHub instance.
cache-control	cache or no-cache	Set cache to enable caching or no-cache to disable caching for this microapp.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.

About Configuring Privileges From the Command Line

You can add, configure, and maintain privileges for your AppHub instance from the command line.

Whether you use ARCS or another configuration service for your AppHub instance, you can perform the following privileges configuration and maintenance tasks from the command line:

- Configure privileges
- Update privileges
- Retrieve privileges configuration settings
- Delete privileges

If you are using ARCS, you can also perform many of these tasks by using the Settings microapp.

Configuring Privileges From the Command Line

You can configure privileges for your AppHub instance from the command line.

To configure privileges from the command line, you use the POST command to send a JSON payload to the URL for your AppHub instance. If ARCS is your configuration service, you can also use the Settings microapp to configure privileges.

- Do one of the following:
  - Use the following JSON payload:

```
{
  "uri": "<privilege-uri>",
  "privilegeUri": "<privilege-host>"
}
```

  - Use the following cURL command:

```
curl -X POST \
  <arcsurl>/privileges/ \
  -H 'authorization: <uaa-token>' \
  -H 'cache-control: cache | no-cache' \
  -H 'content-type: application/json' \
  -H 'predix-zone-id: <apphub-service-zone-id>' \
  -d '[
    {
```

```
    "uri": "<privilege-uri>",
    "privilegeUri": "<privilege-host>"
  }
]'
```

Table 8: Privileges Configuration Options

Key	Value	Description
uri	<privilege-uri>	The fully qualified URL to a privileges server to use for access control for your AppHub service instance.
privilegeUri	<privilege-host>	The host server and port to use to connect with the privileges server.
arcsurl	<arcsurl>	The URL for your ARCS server or other configuration service.
authorization	<uaa-token>	The token for the UAA client for your AppHub instance.
cache-control	cache or no-cache	Set cache to enable caching or no-cache to disable caching for this microapp.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.

Updating Privileges From the Command Line

You can update privileges for your AppHub instance from the command line.

To update privileges from the command line, you use the PUT command to send a JSON payload to the URL for your AppHub instance. If ARCS is your configuration service, you can also use the Settings microapp to update privileges.

- Do one of the following:
  - Use the following JSON payload:

```
{
  "uri": "<privilege-uri>",
  "privilegeUri": "<privilege-host>"
  "uaaMetadata":
  {
    "uri":"<uaa-uri>",
    "clientId":"<uaa-admin-id>",
    "clientSecret":"<uaa-admin-secret>"
  }
}
```

  - Use the following cURL command:

```
curl -X PUT \
 <arcsurl>/privileges/ \
 -H 'authorization: <uaa-token>' \
 -H 'cache-control: cache | no-cache' \
 -H 'content-type: application/json' \
 -H 'predix-zone-id: <apphub-service-zone-id>' \
 -d '[
  {
    "uri": "<privilege-uri>",
    "privilegeUri": "<privilege-host>"
```

```
      "uaaMetadata":
         {
         "uri":"<uaa-uri>",
         "clientId":"<uaa-admin-id>",
         "clientSecret":"<uaa-admin-secret>"
         }
      }
]'
```

Table 9: Privileges Update Options

Key	Value	Description
uri	<privilege-uri>	The fully qualified URL to a privileges server to use for access control for your AppHub service instance.
privilegeUri	<privilege-host>	The host server and port to use to connect with the privileges server.
uaaMetadata	Array of uri, clientID, and clientSecret.	These options can be set only by updating privileges after you make initial settings by using POST.
uri	<uaa-uri>	The URI for the Predix UAA service instance associated with your AppHub instance.
clientID	<uaa-admin-id>	The admin client ID for the UAA service instance.
clientSecret	<uaa-admin-secret>	The admin client secret for the UAA service instance.
arcsurl	<arcsurl>	The URL for your ARCS server or other configuration service.
authorization	<uaa-token>	The token for the UAA client for your AppHub instance.
cache-control	cache or no-cache	Set cache to enable caching or no-cache to disable caching for this microapp.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.

Retrieving Privileges Configuration Settings

You can delete privilege settings for your AppHub instance from the command line.

To retrieve privilege configuration settings for your AppHub instance, you use cURL with the GET command to send a JSON payload to the ARCS URL for your instance.

- Use the following cURL command:

```
curl -X GET \
  <arcsurl>/privileges/ \
  -H 'authorization: <uaa-token>' \
  -H 'cache-control: cache | no-cache' \
  -H 'content-type: application/json' \
  -H 'predix-zone-id: <apphub-service-zone-id>' \
  -d '[
    {
```

```
      "uri": "<privilege-uri>",
      "privilegeUri": "<privilege-host>"
      "uaaMetadata":
         {
         "uri":"<uaa-uri>",
         "clientId":"<uaa-admin-id>",
         "clientSecret":"<uaa-admin-secret>"
         }
      }
]'
```

Table 10: Privileges Retrieval Options

Key	Value	Description
arcsurl	<arcsurl>	The URL for your ARCS server or other configuration service.
authorization	<uaa-token>	The token for the UAA client for your AppHub instance.
cache-control	cache or no-cache	Set cache to enable caching or no-cache to disable caching for this microapp.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.
uri	<privilege-uri>	The fully qualified URL to a privileges server to use for access control for your AppHub service instance.
privilegeUri	<privilege-host>	The host server and port to use to connect with the privileges server.
uaaMetadata	Array of uri, clientID, and clientSecret.	These options can be set only by updating privileges after you make initial settings by using POST.
uri	<uaa-uri>	The URI for the Predix UAA service instance associated with your AppHub instance.
clientID	<uaa-admin-id>	The admin client ID for the UAA service instance.
clientSecret	<uaa-admin-secret>	The admin client secret for the UAA service instance.

Deleting Privilege Settings From the Command Line

You can delete privilege configuration settings from your AppHub instance from the command line.

To delete privilege settings from your AppHub instance, you use cURL with the DELETE command to send application details to the ARCS URL for your instance.

- Use the following cURL command:

```
curl -X DELETE \
  <arcsurl>/privileges/ \
  -H 'authorization: <uaa-token>' \
  -H 'cache-control: cache | no-cache' \
  -H 'content-type: application/json' \
  -H 'predix-zone-id: <apphub-service-zone-id>'
```

About Configuring Preferences From the Command Line

You can add, configure, and maintain preferences for your AppHub instance from the command line.

Whether you use ARCS or another configuration service for your AppHub instance, you can perform the following preferences configuration and maintenance tasks from the command line:

- Configure preferences
- Update preferences
- Retrieve preferences configuration settings
- Delete preferences

If you are using ARCS, you can also perform many of these tasks by using the Settings microapp.

Configuring Preferences From the Command Line

You can configure preferences for your AppHub instance from the command line.

To configure preferences from the command line, you use the POST command to send a JSON payload to the URL for your AppHub instance. If ARCS is your configuration service, you can also use the Settings microapp to configure preferences.

- Do one of the following:
  - Use the following JSON payload:

```
{
  "uri": "<preference-uri>",
  "preferredLocale": "<preferred-locale>",
  "preferenceUri": "<preference-host>"
}
```

  - Use the following cURL command:

```
curl -X POST \
 <arcsurl>/preferences \
 -H 'authorization: <uaa-token>' \
 -H 'cache-control: cache | no-cache' \
 -H 'content-type: application/json' \
 -H 'predix-zone-id: <apphub-service-zone-id>' \
 -d '[
    {
      "uri": "<preference-uri>",
      "preferredLocale": "<preferred-locale>",
      "preferenceUri": "<preference-host>"
    }
]'
```

Table 11: Preferences Configuration Options

Key	Value	Description
uri	<preference-uri>	The fully qualified path to a preferences server for the specified default language.
preferredLocale	<preferred-locale>	The default language to use for your AppHub service instance.

Key	Value	Description
preferenceUri	<preference-host>	The host server and port to use to connect with the preferences server.
authorization	<uaa-token>	The token for the UAA client for your AppHub instance.
cache-control	cache or no-cache	Set cache to enable caching or no-cache to disable caching for this microapp.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.

Updating Preferences From the Command Line

You can update preferences for your AppHub instance from the command line.

To update preferences from the command line, you use the PUT command to send a JSON payload to the URL for your AppHub instance. If ARCS is your configuration service, you can also use the Settings microapp to update preferences.

- Do one of the following:
  - Use the following JSON payload:

```
{
  "uri": "<preference-uri>",
  "preferredLocale": "<preferred-locale>",
  "preferenceUri": "<preference-host>"
}
```

  - Use the following cURL command:

```
curl -X PUT \
  <arcsurl>/preferences \
  -H 'authorization: <uaa-token>' \
  -H 'cache-control: cache | no-cache' \
  -H 'content-type: application/json' \
  -H 'predix-zone-id: <apphub-service-zone-id>' \
  -d '[
    {
      "uri": "<preference-uri>",
      "preferredLocale": "<preferred-locale>",
      "preferenceUri": "<preference-host>"
    }
]'
```

Table 12: Preferences Configuration Options

Key	Value	Description
uri	<preference-uri>	The fully qualified path to a preferences server for the specified default language.
preferredLocale	<preferred-locale>	The default language to use for your AppHub service instance.
preferenceUri	<preference-host>	The host server and port to use to connect with the preferences server.

Key	Value	Description
arcsurl	<arcsurl>	The URL for your ARCS server or other configuration service.
authorization	<uaa-token>	The token for the UAA client for your AppHub instance.
cache-control	cache or no-cache	Set cache to enable caching or no-cache to disable caching for this microapp.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.

Retrieving Preferences Configuration Settings

You can delete preference settings for your AppHub instance from the command line.

To retrieve preference configuration settings for your AppHub instance, you use cURL with the GET command to send a JSON payload to the ARCS URL for your instance.

- Use the following cURL command:

```
curl -X GET \
  <arcsurl>/preferences \
  -H 'authorization: <uaa-token>' \
  -H 'cache-control: cache | no-cache' \
  -H 'content-type: application/json' \
  -H 'predix-zone-id: <apphub-service-zone-id>' \
  -d '[
    {
      "uri": "<preference-uri>",
      "preferredLocale": "<preferred-locale>",
      "preferenceUri": "<preference-host>"
    }
]'
```

Table 13: Preferences Retrieval Options

Key	Value	Description
arcsurl	<arcsurl>	The URL for your ARCS server or other configuration service.
authorization	<uaa-token>	The token for the UAA client for your AppHub instance.
cache-control	cache or no-cache	Set cache to enable caching or no-cache to disable caching for this microapp.
predix-zone-id	<apphub-service-zone-id>	The zone ID for your AppHub instance.
preferredLocale	<preferred-locale>	The default language to use for your AppHub service instance.
preferenceUri	<preference-host>	The host server and port to use to connect with the preferences server.

Working With the Settings Microapp

About the Settings Microapp

All new AppHub service instances that use ARCS include the **Settings** microapp, which you can use to perform many configuration and management tasks.

The **Settings** microapp has an easy, intuitive UI that simplifies AppHub configuration tasks and removes the need for command-line usage after you create your new instance. You can use the **Settings** microapp to:

- Configure the look and feel of your AppHub service instance.
- Inject JavaScript across all microapps in your service instance.
- Add new microapps and configure new and existing microapps.
- Define navigation and role-based access for each microapp.
- Add new themes and configure new and existing themes.
- Define the look and feel of new and existing themes.
- Set language and locale preferences for your service instance.
- Specify server-based authentication and access control for your service instance.

Configuring AppHub Settings

To use the **Settings** microapp, you must first create an AppHub service instance and create any microapps or themes you want to add to the instance.

1. Log into your AppHub service instance with an account that has administrator rights.
2. In the left navigation pane, at the bottom, click the gear icon, then select **Settings App**.
3. Make settings in any of the following tabs:
   - **Global Config**
   - **Apps**
   - **Themes**
   - **Preferences**
   - **Privileges**
4. Click **Save**.

Global Config Tab

In the **Global Config** tab, you can define and modify settings for your AppHub service instance.

Table 14: Global Config Tab Options

Section	Option	Description
General Settings	**Application Name**	The name for your AppHub service instance, which appears at the top of the left navigation pane. (Required)
	Logo URI	The URI to an image file to use as a logo next to the instance name. (Optional)
	Theme	The theme you want to apply to your AppHub instance. The available options are based on themes you configure in the **Themes** tab. If not set, the default AppHub theme is applied. (Optional)
	Enable application chrome	When selected, this option disables the left navigation pane.
Scripts	**Add** 	Adds a field you can use to specify a JavaScript file to inject into your AppHub instance.
	URL	The fully qualified URL to a JavaScript file to be injected into your AppHub instance. (Required)
	Remove 	Removes a field to delete a linked script.

Apps Tab

In the **Apps** tab, you can add and configure microapps for your AppHub service instance.

Table 15: Apps Tab Options

Section	Option	Description
Microapps subpane		A list of the microapps in your AppHub instance, with arrows to modify the display order and a delete button. Microapps appear in the AppHub left navigation pane in the order listed in this subpane.
	Add new app	Adds a new microapp entry at the bottom of the list. You can then make configuration settings in the right subpane.

Section	Option	Description
App details	**ID**	The microapp name or title. (Required)
	Path	The relative path to the microapp on the server. (Required)
	Hostname	The host server for the microapp. (Required)
	Template	The relative path to the main template file for the microapp on the server. (Required)
	Label	The display name for the microapp. (Required)
	Icon	The display icon for the microapp. (Required)
	Menu Location	Microapps can appear in the **Main**, in the **Profile**, or **Settings** menus. (Optional)
	Order	The display order for microapps in the left navigation pane. A value of 0 means the microapp is listed first. If no value is entered here, microapps appear in the list in the order they were added. (Optional)
	Navigation Service	The URL for the navigation server that you want to use for the microapp. When a value is set in this field, the **Sub-Navigation** links section is disabled. (Optional)
	Enable as Default Display App	When selected, this microapp is assigned order 0 and appears in the first AppHub view that users see after they log in.
Sub-Navigation	**Add** 	Adds a set of fields to define sub-navigation links or user groups for your microapp. (Optional)
	Label	The sub-navigation field text label. (Required)
	Path	The relative path for the sub-navigation link. (Required)
	Remove 	Deletes a set of fields to remove sub-navigation links or user groups.
Capabilities	**Group Name**	User groups that can access this microapp. The groups listed here should match what is defined in the OAuth2 client for the microapp.

Themes Tab

In the **Themes** tab, you can add and configure themes for your AppHub service instance.

Table 16: Themes Tab Options

Section	Option	Description
Microapps subpane		A list of the themes configured for your AppHub instance, with a delete button for each entry.
	Add new theme	Adds a new theme entry. You can then make configuration settings in the right subpane.
Theme	**Display Name**	The name to use for this theme in the UI. (Required)
	Demo Link	A clickable link to a demo AppHub instance with this theme applied. (Optional)
	Description	A brief description of the theme. (Optional)
	Base URL	The URL where the theme is hosted. (Required)
	Main Template	The relative path to the HTML template to use for the main page view for the microapp. (Required)
	Error Template	The relative path to the HTML template to use for errors. (Required)
	Minimal Error Template	The relative path to the HTML template to use for errors when running in chromeless mode. (Optional)
Theme Options	**Add** 	Adds a set of fields to define theme options to pass to your microapp. Values in these fields must reflect the skin options implemented in the theme. (Optional)
	Name	The option name. (Optional)
	Value	The option value. (Optional)
	Remove 	Deletes a set of fields to remove theme options.

Preferences Tab

In the **Preferences** tab, you can configure language preferences for your AppHub service instance.

Table 17: Preferences Tab Options

Section	Option	Description
Preferences	**Preferred Locale**	The default language to use for your AppHub service instance. (Optional)
	Preference URI	The fully qualified URL to a preferences server for the specified default language. (Required)

Privileges Tab

In the **Privileges** tab, you can configure server-based access for your AppHub instance.

Table 18: Privileges Tab Options

Section	Option	Description
Privileges	**Privileges URI**	The URL to the privileges server to use for access control for your AppHub service instance.
UAA Metadata	**URI**	The URI for the Predix UAA service instance associated with your AppHub instance.
	Client ID	The admin client ID for the UAA service instance.
	Client Secret	The admin client secret for the UAA service instance.