



Persistent UNITE for GitHub

Installation and Configuration Guide
Version 2.0.2

Statement of confidentiality

© Copyright Persistent Systems 2020

This document is provided with the explicit understanding that the contents will not be divulged to any third party without prior written consent from Persistent Systems. The contents of this document are proprietary and confidential information of Persistent Systems. All registered trademarks are acknowledged. All other company logos or product names mentioned are used for identification purposes only and may be trademarks of their respective owners.

Contents

Statement of confidentiality.....	2
Revision history	4
Overview	5
Prerequisites	6
Software requirements	7
Deploying the connector on application server	8
Establishing JAS connection with <i>Persistent UNITE</i>	10
Configuring the connector instance	12
Establishing the cross-server communication	19
Limitations	23

Revision history

Changes to this document are summarized in the following table in chronological order.

Version	Date	Short description
2.0.2	31 August 2020	<p>Includes the following details:</p> <ul style="list-style-type: none">• Deploying the GitHub connector on the application server• Establishing Jazz Authorization Server (JAS) connection with <i>Persistent UNITE</i>• Configuring the connector instance• Establishing the cross-server communication

Overview

Persistent UNITE™ for GitHub integrates GitHub with the applications in the IBM Engineering Lifecycle Management (ELM) solution. This solution was previously known as the Internet of Things Continuous Engineering Solution (CE) and the Rational solution for Collaborative Lifecycle Management (CLM). For more information about name change, see [Renaming the IBM Continuous Engineering Portfolio](#).

When you work in an integrated environment, it is important to understand the terminology used by the applications. GitHub and ELM use slightly different terms for the same elements.

In GitHub, a *branch* is an independent line of development in a project. In ELM, it is called as *stream*. You can add branches in the global configuration streams.

A *tag* in GitHub is a development branch that does not change. It is a pointer to a specific commit. In ELM, it is called as *baseline*. In ELM, you can add tags to the global configuration streams and components. It helps you to find them easily in a search or to group them in a way that makes sense for your team.

Prerequisites

1. ELM server must be installed, configured with JAS, and functional.
2. JAS must be installed and functional.
3. *Persistent UNITE* must be installed, configured, and functional.
4. IBM WebSphere Liberty server must be installed and configured with *Persistent UNITE*.
5. Database instance is functional and communicates with *Persistent UNITE*.
6. GitHub enterprise server must be installed, configured, and functional.
7. Google Chrome or Mozilla Firefox browser is available and configured to allow pop-up windows.
8. Installation package `UniteForGitHub-v2.0.2.zip` must be available.

Software requirements

The following table provides the software requirements.

Software	Version
Operating system	Windows 2012 R2, Red Hat Enterprise Linux (RHEL) 7
IBM ELM	6.0.6, 6.0.6.1 or 7.0 Note: For ELM version 7.0, iFix002 interim fix or later must be installed.
IBM WebSphere Liberty	18.0.0.1
Database	Derby 10.15, Oracle 11g
Persistent UNITE	2.0.2
GitHub enterprise server	github.com cloud, GitHub server 2.19 or 2.21
Web browser	Google Chrome v81 or later, Mozilla Firefox v76 or later

Deploying the connector on application server

Important: *Persistent UNITE* must be configured in WebSphere Application Server Liberty. Then, you can run the installer and it copies the relevant artifacts to unite liberty profile server.

Liberty profile server path is found in deployment server as, <liberty-server-installation-directory>\wlp\usr\servers\<server-name>

1. Stop the liberty server if it is running.

```
C:\wlp-webProfile7-18.0.0.1\wlp\bin>server.bat stop unite
Stopping server unite.
Server unite stopped.

C:\wlp-webProfile7-18.0.0.1\wlp\bin>
```

Note: The term `unite` in the command is the server name where *Persistent UNITE* is deployed. Directory names available under the location of <liberty-server-installation-directory>\wlp\usr\servers are considered as available server names.

2. Extract `UniteForGitHub-v2.0.2.zip` into a temporary directory.
3. Go to the extracted directory and open `github-config.properties` file.
4. Update the `github.server.publicurl` property to GitHub enterprise server's public URL.

For example, `github.server.publicurl=https://github.persistent.com`

Note: You can configure only one GitHub public URL for each GitHub connector in *Persistent UNITE*. See [Limitations](#) for the details.

5. Go to the extracted directory and run one of the following scripts based on the target operating system:

- For Windows operating system,
`install-connector.bat`
- For Linux and UNIX based operating systems,
`# chmod +x install-connector.sh`
`# ./install-connector.sh`

6. Enter the liberty server profile path when prompted:

- For Windows operating system,
`<liberty-server-installation-dir>\wlp\usr\servers\unite`
- For Linux and UNIX based operating systems,
`<liberty-server-installation-dir>/wlp/usr/servers/unite`

Note: The term `unite` in the liberty server profile path, is the server name where *Persistent UNITE* is deployed. You must replace the placeholders according to the server installation directory.

Connector is installed. You can see the installation log in `cli-log.log` file.

7. Start the liberty server to complete the connector deployment.

```
C:\wlp-webProfile7-18.0.0.1\wlp\bin>server.bat start unite
Starting server unite.
Server unite started.

C:\wlp-webProfile7-18.0.0.1\wlp\bin>
```

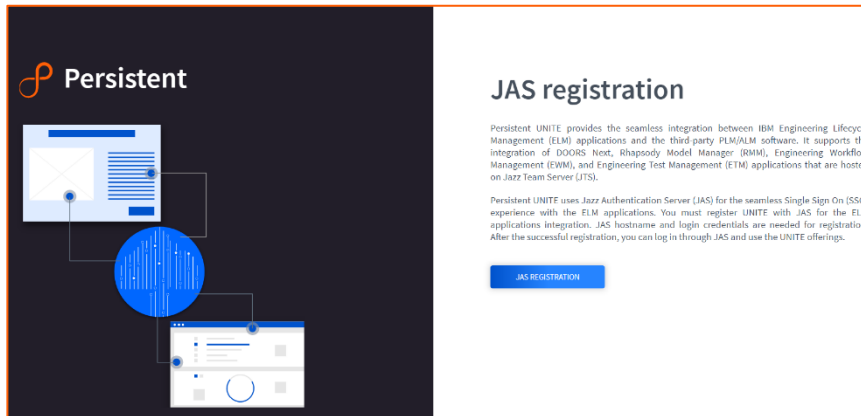
Establishing JAS connection with *Persistent UNITE*

Important: Do the following steps only if the JAS connection is not established with *Persistent UNITE* previously.

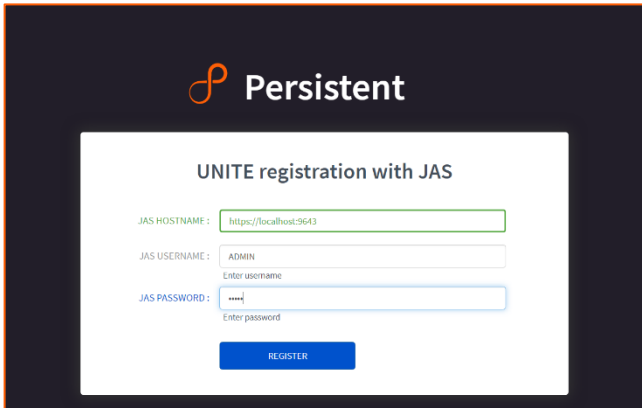
1. In a web browser, enter `https://<unite_host_name>:port/unite` to go to *Persistent UNITE*.

Note: The `unite_host_name` is the hostname with the DNS domain reference of the server where the WebSphere Application Server Liberty is installed.

2. Click **JAS REGISTRATION**.

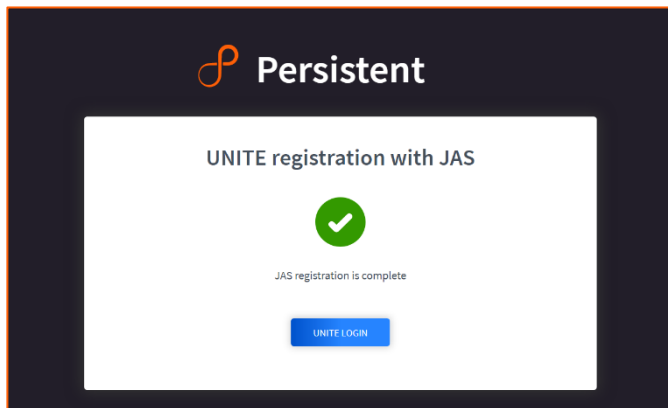


3. Enter the following parameters:
 - **JAS HOSTNAME:** Enter as `https://<jas_host_name>:port`
 - **JAS USERNAME**
 - **JAS PASSWORD**

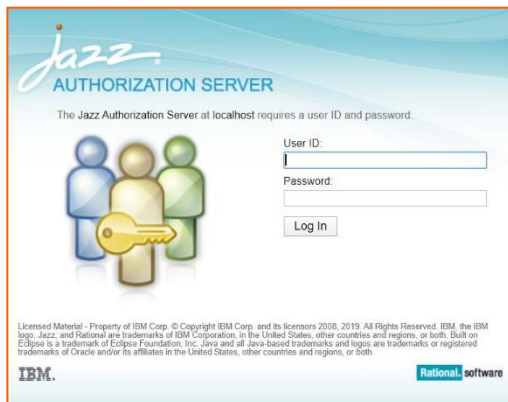
The screenshot shows the Persistent logo at the top. Below it, the title "UNITE registration with JAS" is centered. The form contains three input fields: "JAS HOSTNAME:" with the value "https://localhost:9643", "JAS USERNAME:" with the value "ADMIN", and "JAS PASSWORD:" with masked characters "****". Below each field is a small hint text: "Enter username" and "Enter password". At the bottom center, there is a blue button labeled "REGISTER".

4. Click **REGISTER**. JAS registration is completed.

5. Click **UNITE LOGIN**. JAS SSO page opens.



6. Enter **User ID** and **Password** and click **Log In**. After the JAS login, *Persistent UNITE* home page opens.



Configuring the connector instance

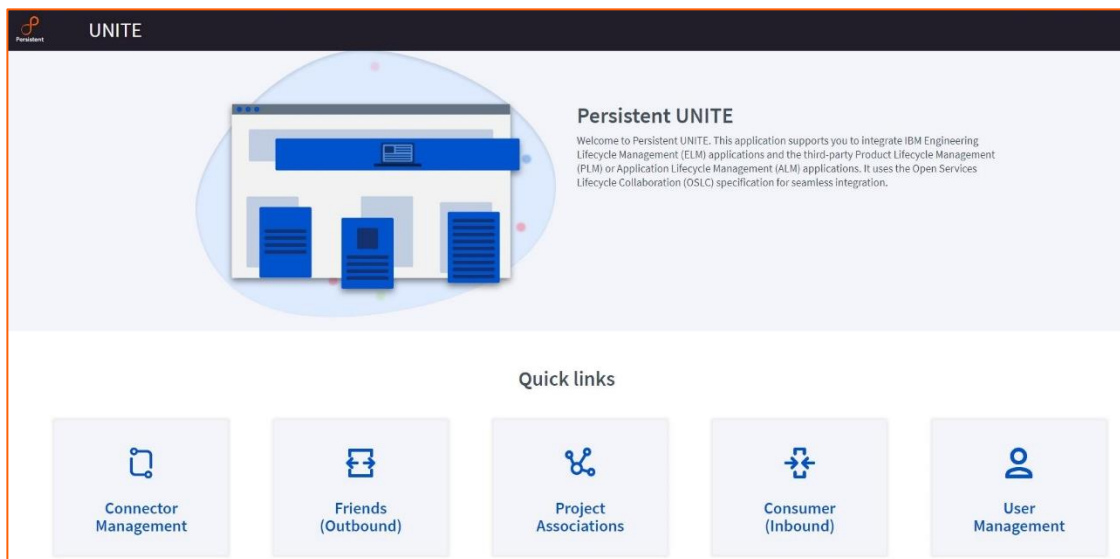
Follow the steps to create a GitHub connector instance in *Persistent UNITE*.

1. In a web browser, enter `https://<unite_host_name>:port/unite` to go to *Persistent UNITE*.
2. Enter **User ID** and **Password** and click **Log In**. After the JAS login, *Persistent UNITE* home page opens.

Important: The JAS user account must require *JazzAdmin* role that is assigned in ELM.



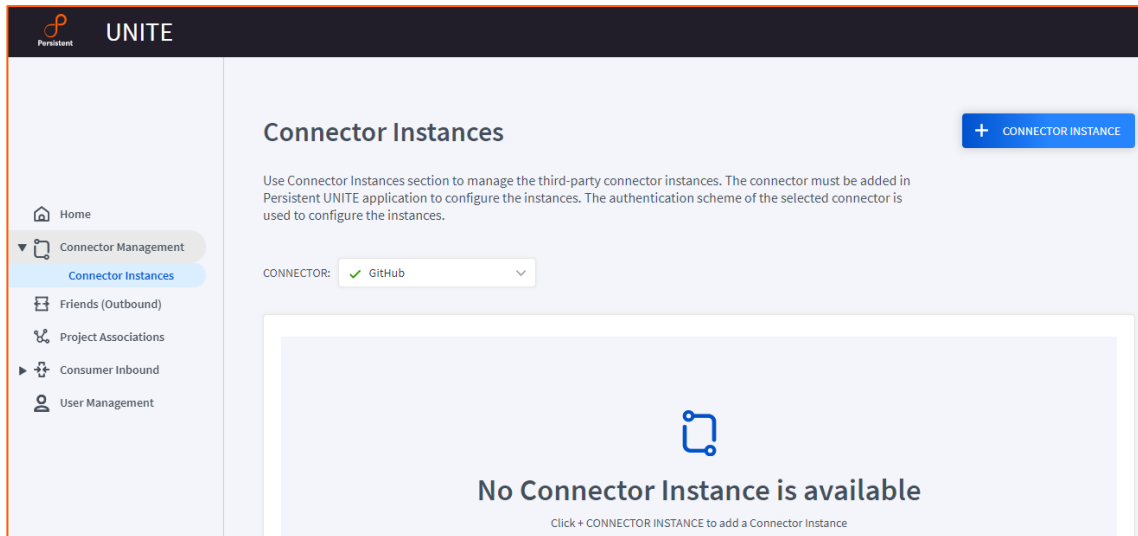
3. Click **Connector Management**. Connector Instances page opens.



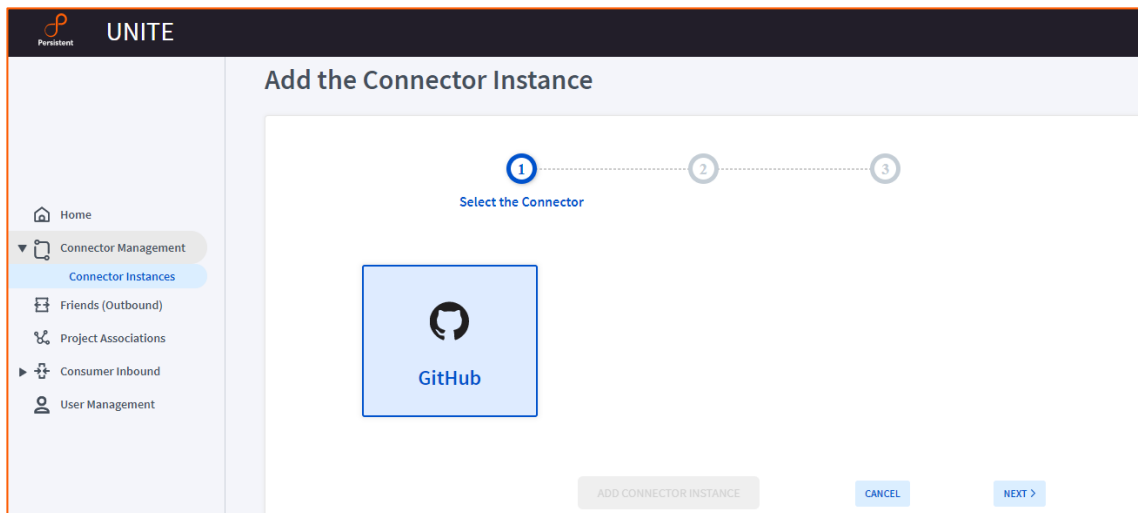
4. Select GitHub from **CONNECTOR** list and click **CONNECTOR INSTANCE**.

Select the Connector page opens.

Note: If you cannot see GitHub in **CONNECTOR** list, you need to repeat the connector installation steps. See [Deploying the connector on application server](#) for the details.



5. Select GitHub and click **NEXT**. Configure the Connector Instance page opens.



6. Enter the following details:

- **CONNECTOR INSTANCE NAME**
- **GITHUB BASE URL:** Enter the GitHub server API base URL that is used in the connector.
- **HOST NAME / IP ADDRESS:** This field is populated according to **GITHUB BASE URL**.

- **CONNECTOR INSTANCE DESCRIPTION:** Enter the detailed description of connector instance. This field is optional.
- **AUTHENTICATION SCHEME:** This field is populated as OAUTH2.
- **OAUTH SCOPE:** Enter `user repo` to provide the user and repo permissions.
- **CALLBACK URL:** This field is populated by *Persistent UNITE* application.

Note: You need to create an application in the connector according to OAuth2 specification for the seamless integration. The remaining fields are updated after the new OAuth application is created in GitHub server.

The screenshot shows the 'UNITE' web application interface. On the left is a sidebar menu with options: Home, Connector Management (selected), Connector Instances (sub-selected), Friends (Outbound), Project Associations, Consumer Inbound, and User Management. The main content area is titled 'Add the Connector Instance'. It features a progress bar at the top with three steps: 1. Add the Connector Instance, 2. Configure the Connector Instance (active), and 3. Verify the Connector Instance. Below the progress bar, the form contains the following fields:

- CONNECTOR INSTANCE NAME:
- GITHUB BASE URL:
- HOSTNAME / IP ADDRESS:
- CONNECTOR INSTANCE DESCRIPTION:
- AUTHENTICATION SCHEME:
- OAUTH SCOPE:
- CALLBACK URL:

 Each field has a small help icon to its right.

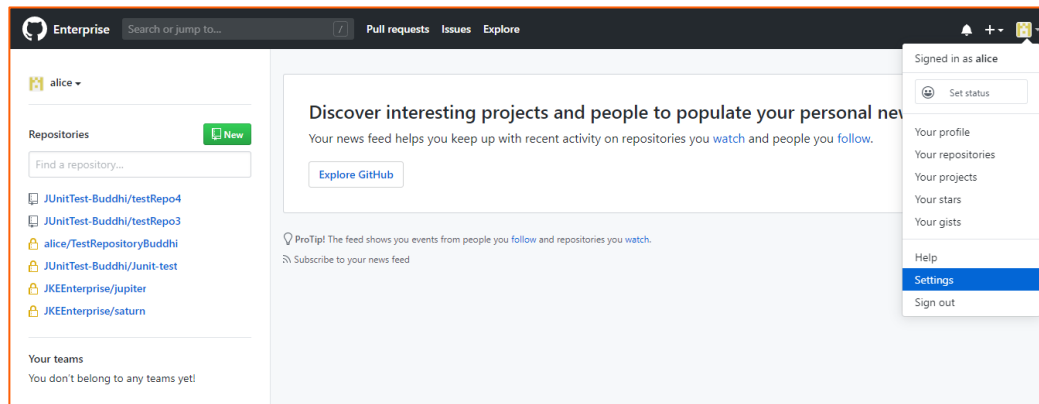
7. Copy **CALLBACK URL** field to clipboard for registering the application with the connector.
8. In a web browser, open the GitHub server URL.
9. Enter **Username or email address** and **Password**. Click **Sign in**.

Note: The user requires administrator access to proceed with the next steps on GitHub server.

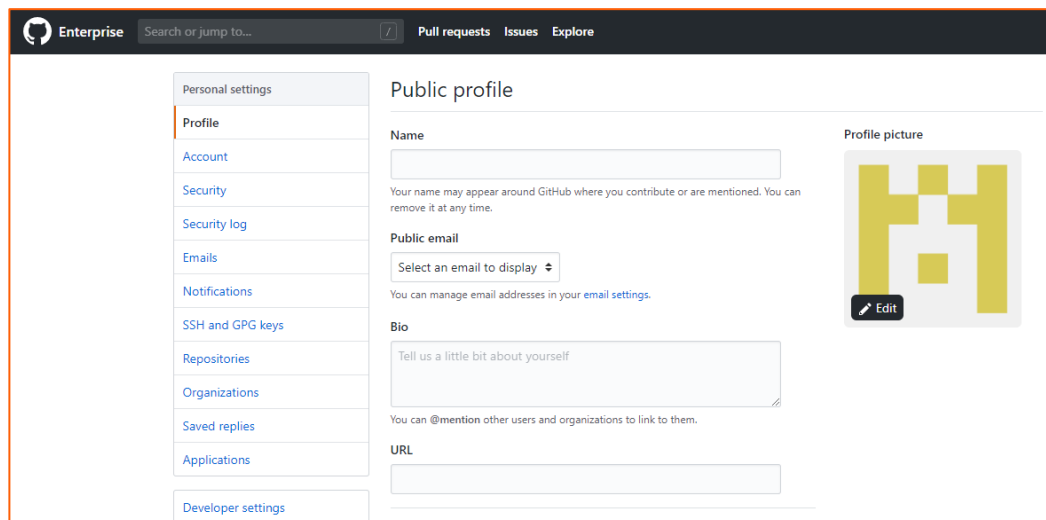
The screenshot shows the GitHub Enterprise sign-in page. It has a header with the GitHub logo and 'Enterprise'. Below it is the text 'Sign in to your account'. The form contains:

- A label 'Username or email address' above a text input field.
- A label 'Password' above a password input field, with a 'Forgot password?' link to its right.
- A green 'Sign in' button at the bottom of the form.

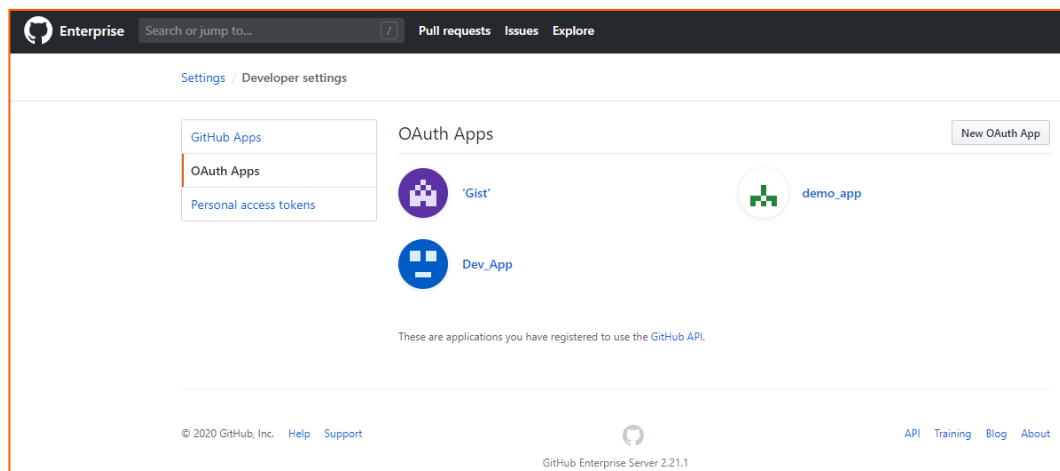
10. Click **Settings** in the **Profile** menu.



11. Click **Developer settings**.

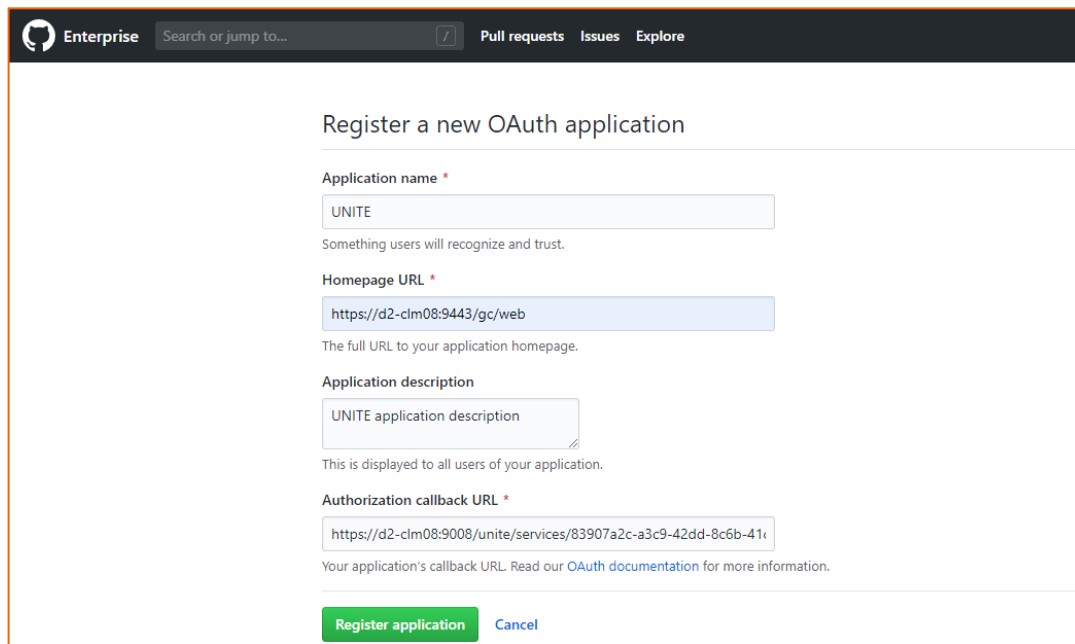


12. Click **New OAuth App** from **OAuth Apps** menu.



13. Enter the following details:

- **Application name:** Enter the application name as `UNITE`.
- **Homepage URL:** Enter the ELM URL.
- **Application description:** Enter the detailed description. This field is optional.
- **Authorization callback URL:** Paste the **CALLBACK URL** field that was copied from Configure the Connector Instance page in *Persistent UNITE*.



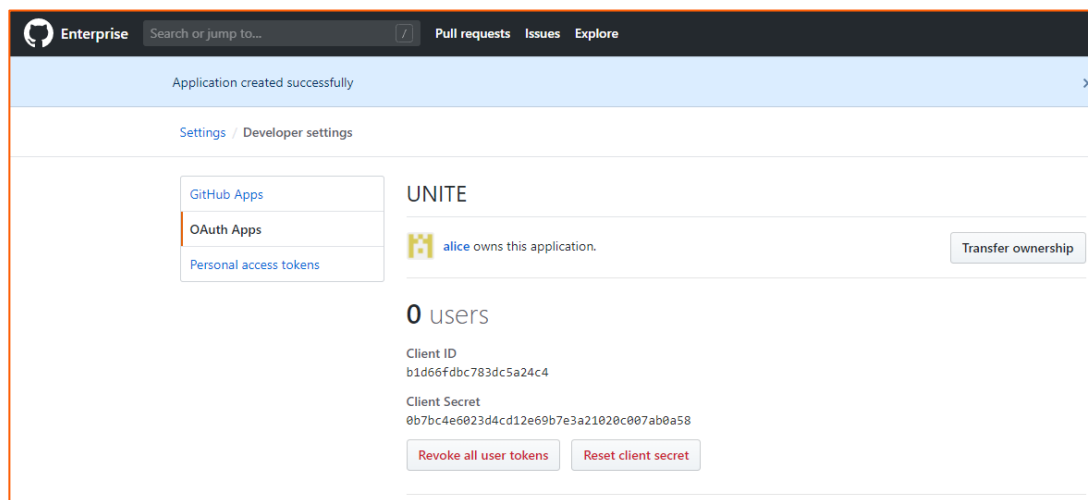
The screenshot shows the 'Register a new OAuth application' page in GitHub Enterprise. The form includes the following fields and values:

- Application name:** UNITE
- Homepage URL:** https://d2-clm08:9443/gc/web
- Application description:** UNITE application description
- Authorization callback URL:** https://d2-clm08:9008/unite/services/83907a2c-a3c9-42dd-8c6b-41c

At the bottom of the form are two buttons: 'Register application' (green) and 'Cancel' (blue).

14. Click **Register application**. Application is created.

A success message is shown. The application details are shown in the following page.



The screenshot shows the 'Developer settings' page for the 'UNITE' application. A blue banner at the top indicates 'Application created successfully'. The page layout includes:

- Left sidebar:** A menu with 'GitHub Apps', 'OAuth Apps' (highlighted), and 'Personal access tokens'.
- Main content area:**
 - Header:** 'UNITE' with a 'Transfer ownership' button.
 - Ownership:** A message stating 'alice owns this application.'.
 - Users:** A section titled '0 users'.
 - Client ID:** b1d66fdb783dc5a24c4
 - Client Secret:** 0b7bc4e6023d4cd12e69b7e3a21020c007ab0a58
 - Buttons:** 'Revoke all user tokens' and 'Reset client secret'.

15. Copy the following fields from the new application details page and paste them in sequence in Configure the Connector Instance page of *Persistent UNITE*.

- **CLIENT NAME:** Enter the client name as UNITE.
- **Client ID to CLIENT ID**
- **Client Secret to CLIENT SECRET**

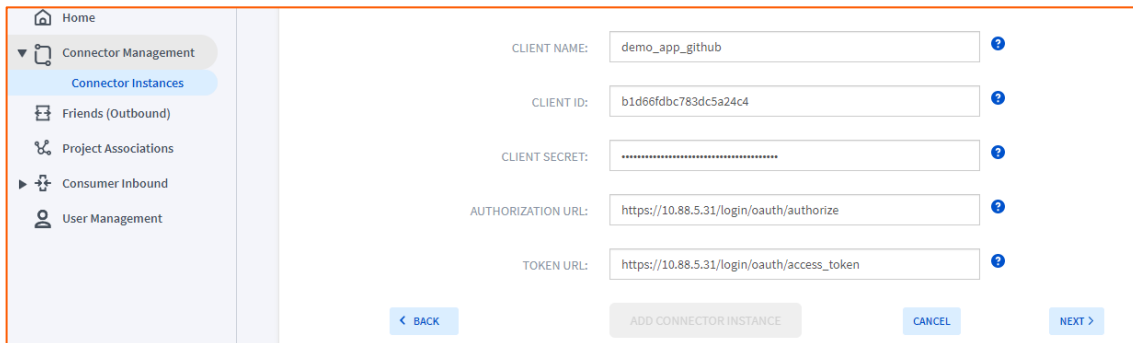
16. Enter the following fields and click **NEXT** in Configure the Connector Instance page.

- **AUTHORIZATION URL:** Obtained from third-party connector documentation.
- **TOKEN URL:** Obtained from third-party connector documentation.

Note: The authorization and token URL for GitHub enterprise are typically in the following format:

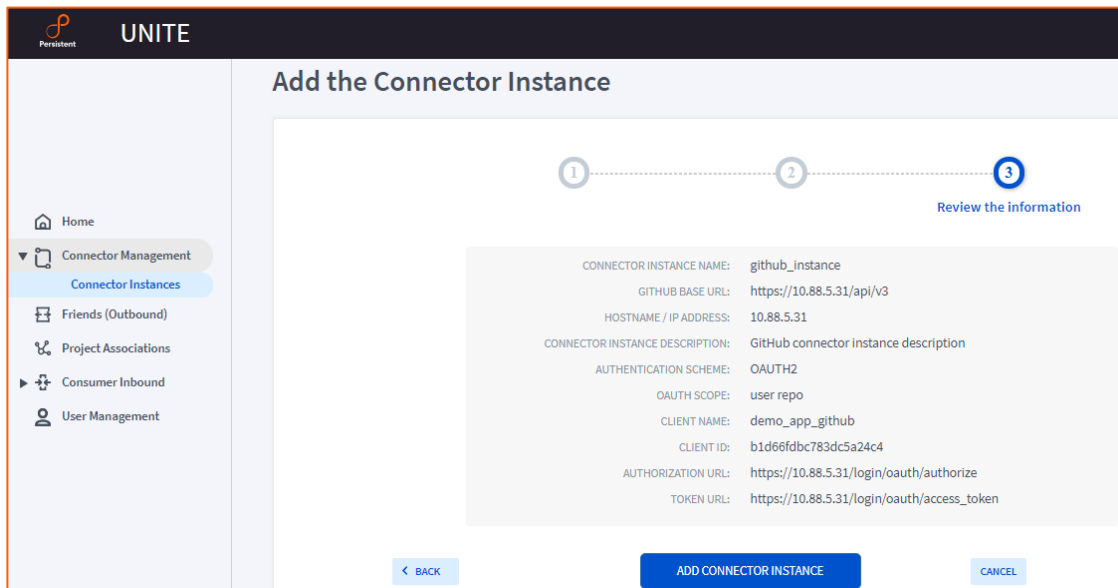
Authorization URL: `https://<hostname:port>/login/oauth/authorize`

Token URL: `https://<hostname:port>/login/oauth/access_token`

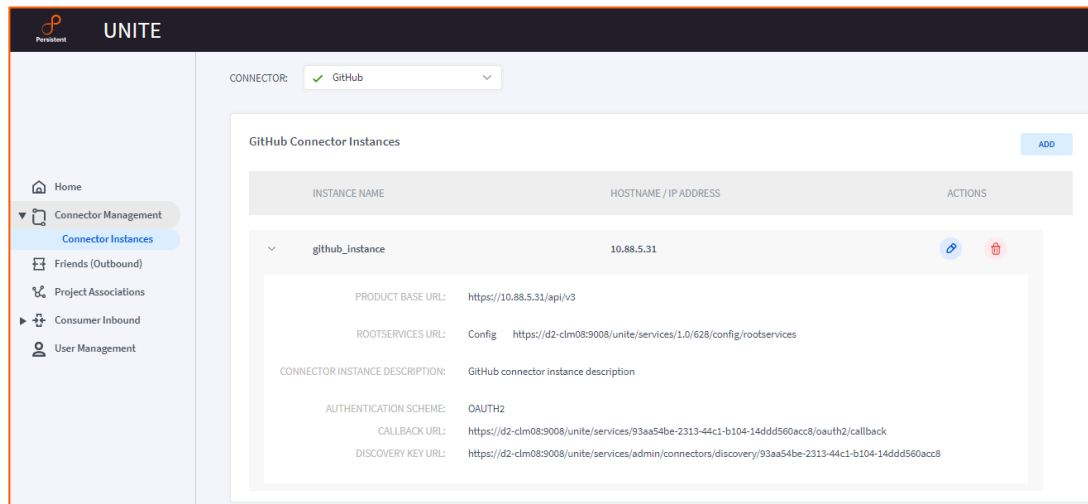


17. Verify the details and click **ADD CONNECTOR INSTANCE**.

Connector instance is added and shown in Connector Instances page.



18. Click the new connector instance name to see the details.



19. Copy **ROOTSERVICES URL Config** domain link. It is used in [Establishing the cross-server communication](#).

Establishing the cross-server communication

You need to establish the cross-server communication between the Global Configuration (GC) Management application of IBM ELM and *Persistent UNITE* where GitHub connector is configured. This relationship between the servers is called as friend relationship. It indicates that the servers can communicate each other and requests between the servers can be trusted. This friend relationship is required to add the configurations from GitHub connector to GC application.

Root Services URL is used to make outbound friend request between the ELM server and connector. You must log in to ELM server by using an account that has Jazz Administrator privileges.

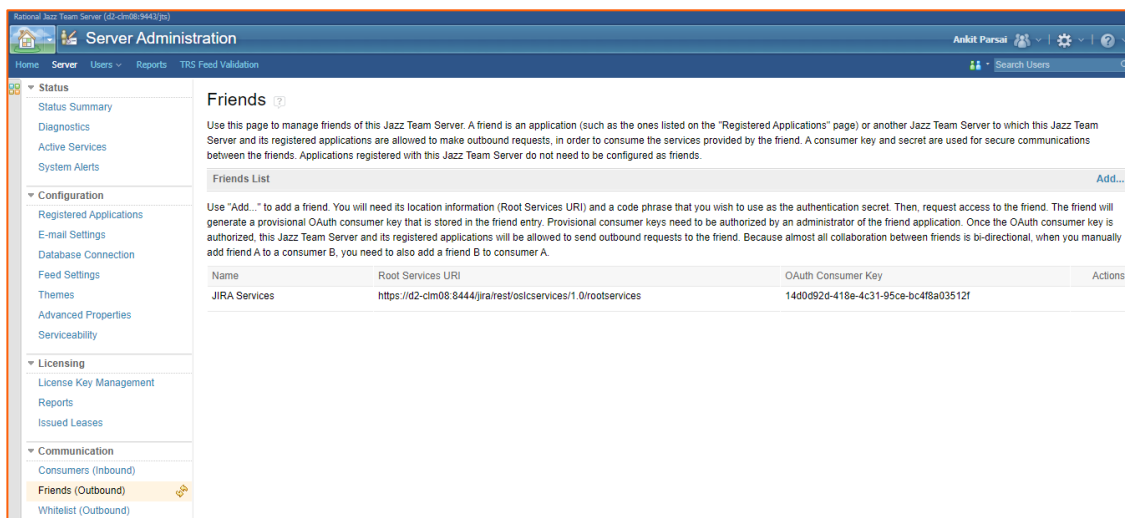
1. In a web browser, go to `https://<elm_host_name>:port/jts/admin` to log in to ELM home page.

Note: The term `elm_host_name` in the command is the hostname with DNS reference of server where ELM server is installed.

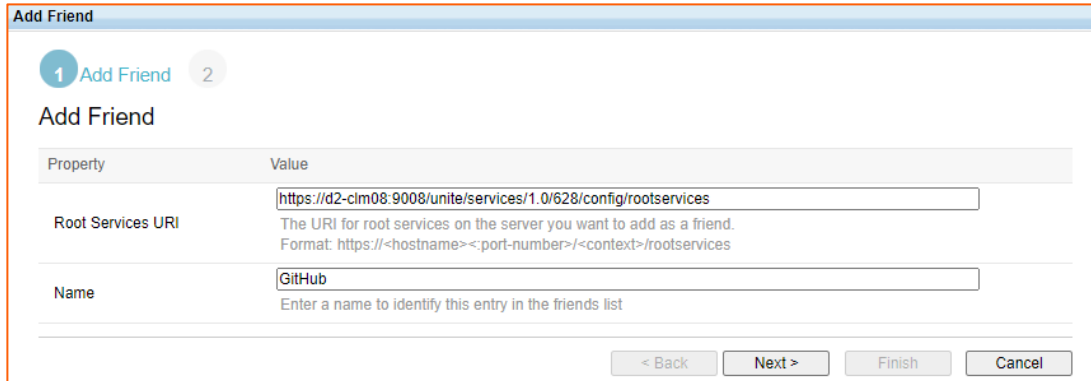
2. Enter **User ID** and **Password**. Click **Log In**.



3. In **Server** tab, under **Communication** section, click **Friends (Outbound)**.



4. Click **Add**. Add Friends page opens.
5. Enter the following inputs:
 - **Root Services URI**: Paste the Root Services URL that was copied from [Configuring the Connector Instance](#).
 - **Name**: Enter the name to identify the friend server. For example, GitHub.
6. Click **Next**. Add Friend page 2 opens.



Add Friend

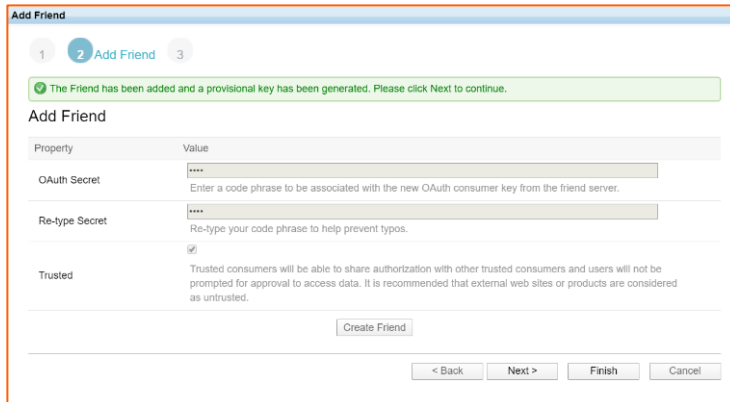
1 Add Friend 2

Add Friend

Property	Value
Root Services URI	https://d2-clm08-9008/unite/services/1.0/628/config/rootservices <small>The URI for root services on the server you want to add as a friend. Format: https://<hostname><port-number>/<context>/rootservices</small>
Name	GitHub <small>Enter a name to identify this entry in the friends list</small>

< Back Next > Finish Cancel

7. Enter the following inputs:
 - **OAuth Secret**
 - **Re-type Secret**
 - **Trusted**
8. Click **Create Friend**. A success message is shown for the friend addition and provisional key generation.



Add Friend

1 2 Add Friend 3

✓ The Friend has been added and a provisional key has been generated. Please click Next to continue.

Add Friend

Property	Value
OAuth Secret	***** <small>Enter a code phrase to be associated with the new OAuth consumer key from the friend server.</small>
Re-type Secret	***** <small>Re-type your code phrase to help prevent typos.</small>
Trusted	<input checked="" type="checkbox"/> <small>Trusted consumers will be able to share authorization with other trusted consumers and users will not be prompted for approval to access data. It is recommended that external web sites or products are considered as untrusted.</small>

Create Friend

< Back Next > Finish Cancel

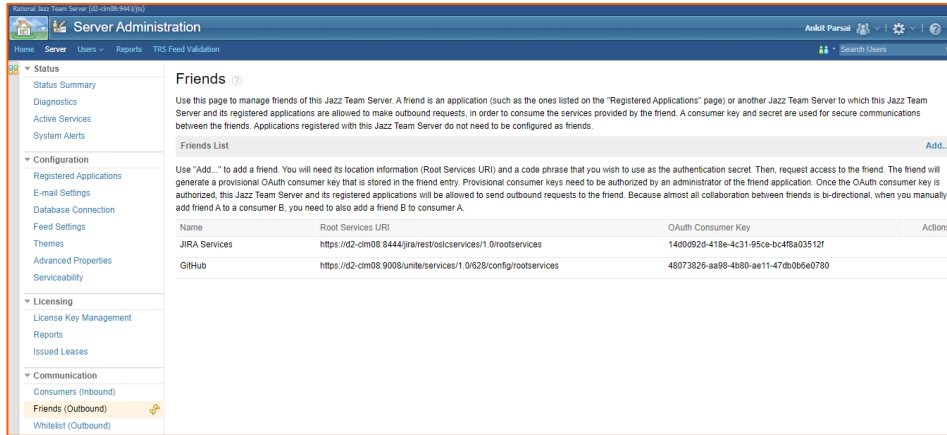
9. Click **Next**. Authorize Provisional Key page opens.

10. Click **Grant access for the provisional key** link.
11. If you see the following page, select **openid** and click **Allow, remember my decision** or **Allow once**.

12. If you see the following page, select **Trusted** and click **Allow**. A success message is shown to indicate that the provisional key is approved.

13. Click **Finish**.

14. In **Friends (Outbound)** page, the new friend is added in the **Friends List** table.



Server Administration

Home Server Users Reports TRS Feed Validation

Search Users

Friends ?

Use this page to manage friends of this Jazz Team Server. A friend is an application (such as the ones listed on the "Registered Applications" page) or another Jazz Team Server to which this Jazz Team Server and its registered applications are allowed to make outbound requests, in order to consume the services provided by the friend. A consumer key and secret are used for secure communications between the friends. Applications registered with this Jazz Team Server do not need to be configured as friends.

Friends List [Add...](#)

Use "Add..." to add a friend. You will need its location information (Root Services URI) and a code phrase that you wish to use as the authentication secret. Then, request access to the friend. The friend will generate a provisional OAuth consumer key that is stored in the friend entry. Provisional consumer keys need to be authorized by an administrator of the friend application. Once the OAuth consumer key is authorized, this Jazz Team Server and its registered applications will be allowed to send outbound requests to the friend. Because almost all collaboration between friends is bi-directional, when you manually add friend A to a consumer B, you need to also add a friend B to consumer A.

Name	Root Services URI	OAuth Consumer Key	Actions
JIRA Services	https://d2-clm08.8444/jira/rest/ostoolservices/1.0/rootservices	140052d-418e-4c31-95ce-bc4f8a03512f	
GitHub	https://d2-clm08.9008/units/services/1.0/528/config/rootservices	48073826-aa98-4b80-ae11-47db0b6e0780	

Navigation Menu:

- Status
 - Status Summary
 - Diagnostics
 - Active Services
 - System Alerts
- Configuration
 - Registered Applications
 - E-mail Settings
 - Database Connection
 - Feed Settings
 - Themes
 - Advanced Properties
 - Serviceability
- Licensing
 - License Key Management
 - Reports
 - Issued Licenses
- Communication
 - Consumers (Inbound)
 - Friends (Outbound)**
 - Whitelist (Outbound)

Limitations

Multiple GitHub connector instances cannot be configured in *Persistent UNITE*

Persistent UNITE allows to configure multiple connector instances for the installed GitHub connector. However, only one connector instance is functional as the connector public URL is managed by GitHub connector in a config file.



Persistent

Persistent Systems (BSE & NSE: PERSISTENT) builds software that drives our customers' business; enterprises and software product companies with software at the core of their digital transformation.

www.persistent.com