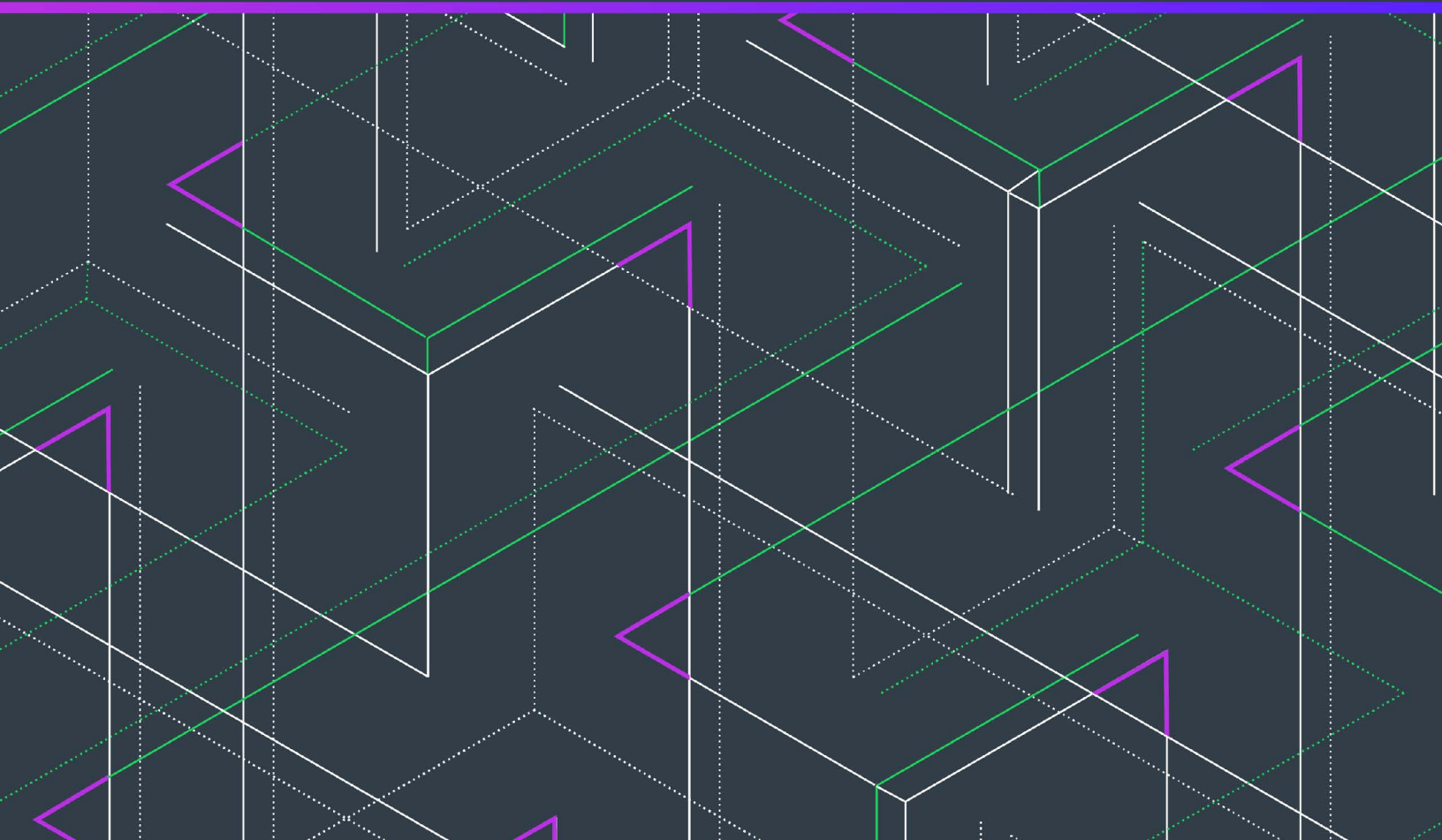


Code Insight 2025 R3

Plugins Guide



Legal Information

Book Name: Code Insight 2025 R3
Part Number: RCI-2025R3-PG00
Product Release Date: August 2025

Copyright Notice

Copyright © 2025 Flexera Software

This publication contains proprietary and confidential information and creative works owned by Flexera Software and its licensors, if any. Any use, copying, publication, distribution, display, modification, or transmission of such publication in whole or in part in any form or by any means without the prior express written permission of Flexera Software is strictly prohibited. Except where expressly provided by Flexera Software in writing, possession of this publication shall not be construed to confer any license or rights under any Flexera Software intellectual property rights, whether by estoppel, implication, or otherwise.

All copies of the technology and related information, if allowed by Flexera Software, must display this notice of copyright and ownership in full.

Code Insight incorporates software developed by others and redistributed according to license agreements. Copyright notices and licenses for these external libraries are provided in a supplementary document that accompanies this one.

Intellectual Property

For a list of trademarks and patents that are owned by Flexera Software, see <https://www.revenera.com/legal/intellectual-property.html>. All other brand and product names mentioned in Flexera Software products, product documentation, and marketing materials are the trademarks and registered trademarks of their respective owners.

Restricted Rights Legend

The Software is commercial computer software. If the user or licensee of the Software is an agency, department, or other entity of the United States Government, the use, duplication, reproduction, release, modification, disclosure, or transfer of the Software, or any related documentation of any kind, including technical data and manuals, is restricted by a license agreement or by the terms of this Agreement in accordance with Federal Acquisition Regulation 12.212 for civilian purposes and Defense Federal Acquisition Regulation Supplement 227.7202 for military purposes. The Software was developed fully at private expense. All other use is prohibited.

Contents

1	Code Insight Plugins Guide	7
	Scan-Agent Plugins	7
	Contents of this Book	8
	Product Support Resources	8
	Contact Us	9
2	Installing and Configuring Standard Plugins	11
	About Scan-Agent Plugins	12
	Overview of Available Plugins	12
	Important Plugin Upgrade in Code Insight 2020 R3	13
	Preparing to Use the Plugins	14
	Application of Scan Profile Settings	14
	Providing an Authorization Token	16
	Downloading Plugins	16
	Plugins for Integrated Development Environments (IDEs)	17
	Eclipse Plugin	17
	Prerequisites for the Eclipse Plugin	17
	Installing the Eclipse Plugin	17
	Configuring the Eclipse Plugin	20
	Running a Scan within Your Eclipse Environment	22
	Uninstalling the Eclipse Plugin	22
	Visual Studio Plugin	23
	Prerequisites for the Visual Studio Plugin	23
	Installing the Visual Studio Plugin	23
	Configuring the Visual Studio Plugin	25
	Configuration Using Visual Studio IDE	25
	Configuration Using MSBuild	28
	Executing a Scan	29

<i>Scan Execution Using Visual Studio IDE</i>	30
<i>Scan Execution Using MSBuild</i>	30
Disabling or Uninstalling the Visual Studio Plugin	31
Plugins for Continuous Integration (CI) Tools	32
Azure DevOps Extension	32
Prerequisites for the Azure DevOps Extension	33
Installing the Azure DevOps Extension	33
Adding a Scan Task to Your Azure DevOps Agent Job	33
Bamboo Plugin	36
Prerequisites for the Bamboo Plugin	36
Installing and Configuring the Bamboo Plugin	37
GitLab Plugin	39
Prerequisites for the GitLab Plugin	39
Integrating the Generic Scan-Agent Plugin with GitLab Runner	40
Using “run_scan.bat” to Configure the CI/CD Pipeline	40
Using “codeinsight-<generic-plugin-version>.jar” to Configure the CI/CD Pipeline	42
Executing the Build	45
Jenkins Plugin	45
Prerequisites for the Jenkins Scan-Agent Plugin	46
Support for Jenkins Systems Running on Java 11	46
Setting Heap Size for the Jenkins Scan-Agent Plugin	46
Setting Up the Code Insight Jenkins Scan-Agent Plugin	47
Configuring the Publication of a Code Insight Report in Jenkins	49
Support for the Jenkins Pipeline	50
Providing the Pipeline Script for the Scan Step	50
Pipeline Code Examples for Running the Scan	51
Scan Scheduler Plugin for Jenkins	52
TeamCity Plugin	53
Prerequisites for TeamCity Plugin	53
Installing the Generic Scan-Agent on the Team City Build Agent	54
Configuring a Build to Run a Code Insight Scan	54
Executing the Build	56
Configuring SSL for Azure DevOps, Bamboo, and Jenkins Scan Agents	56
Exporting the Certificate Used by Code Insight	56
Importing the Certificate to the Java Home for the Scan Agent	58
Plugins for Package Managers and Build Tools	58
Apache Ant Plugin	58
Prerequisites for the Apache Ant Plugin	59
Configuring the Apache Ant Plugin	59
Executing the Scan	59
Gradle Plugin	61
Prerequisites for the Gradle Plugin	61
Installing and Configuring the Gradle Plugin	61
Maven Plugin	63
More About the Maven Plugin	64
Prerequisites for the Maven Plugin	64

Installing and Configuring the Maven Plugin	64
Cleaning the Application Project	66
Running the Maven Goal for the Scan	67
Plugins for Container Platforms	67
Docker Images Plugin	67
Prerequisites for the Docker Images Plugin	67
Syft Integration in the Docker Images Plugin	68
Obtaining the Docker Images Plugin	71
Running a Docker Images Plugin Scan	71
Running a Docker Images Plugin Scan with a Properties File	71
Running a Docker Images Plugin Scan with Command-Line Options	74
Specifying a Temporary Directory of Your Choice	77
Important Note About an Image Name Containing a "/" in the Command	77
Process of a Docker Images Plugin Scan	78
Generic Scan-Agent Plugin	78
Prerequisites for the Generic Scan-Agent Plugin	79
Support for Systems Running on Java 11	79
Running the Generic Scan-Agent Plugin	79
Status of the Import of Remote Scan Results into Your Project	81
Support for Processing Remote Scan Results in the Background	81
Note About Rescans Performed by v2.0 and Later Plugins	83
3 Developing Custom Plugins	85
Scan Agent Framework	85
Features Provided by the Framework	86
Available Classes and Methods in the Framework	86
Property Settings	87
Downloading the Scan Agent Toolkit	89
Contents of the Scan Agent Toolkit	89
Writing a Custom Scan-Agent Plugin	89
Deploying a Custom Scan-Agent Plugin	90

Code Insight Plugins Guide

Code Insight empowers organizations to take control of and manage their use of open source software (OSS) and other third-party components. It helps development, legal, and security teams use automation to create a formal OSS strategy that balances business benefits and risk management.

The *Code Insight Plugins Guide* describes how to install and configure a Code Insight standard scan-agent plugin (or develop your own custom plugin) directly in your development environment to scan your product source files and post-build artifacts for OSS and third-party components.

This chapter covers the following information:

- [Scan-Agent Plugins](#)
- [Contents of this Book](#)
- [Product Support Resources](#)
- [Contact Us](#)

Scan-Agent Plugins

Code Insight supports a *scan-agent plugin*, which is installed directly in your development environment to perform scans on your product's source files and built artifacts as part of your software development process. This type of scan is an alternative to the standard scan, which is performed source codebase files that are uploaded to Code Insight. (Scans on codebases uploaded to the Scan Server are described in *Code Insight User Guide*.)

The scan-agent plugin is configured to scan a specific set of files within the context of an Engineering application (such as an IDE, artifact repository, CI tool, a build, testing, or installation tool, or a source-management application). Once configured, the plugin can be invoked to run a scan as part of the build process. The scan results, sent back to Code Insight, include scanned-file information and published inventory awaiting review, management, and remediation. Just as with published inventory produced by the Code Insight Scan Server, published inventory produced by a scan-agent plugin can be automatically reviewed by license or security policies during the scan. Inventory not reviewed by policy can be reviewed manually by legal or security experts. Security alerts with corresponding email notifications are automatically generated for any inventory item with new security vulnerabilities.

Code Insight offers a set of standard scan-agent plugins that are pre-built and ready for immediate deployment. It also provides a generic scan-agent plugin (also pre-built) that can be used as a standalone scan-agent to scan arbitrary file systems or integrated with certain Engineering applications for automatic code scanning.

Additionally, Code Insight offers a Scan Agent toolkit that enables you to create a custom scan-agent plugin that integrates with your development ecosystem.

Contents of this Book

The *Code Insight Plugins Guide* includes the following chapters.

Table 1-1 ■ Code Insight Plugins Guide Contents

Topic	Content
Installing and Configuring Standard Plugins	Describes how to install and configure the Code Insight standard scan-agent plugins.
Developing Custom Plugins	Describes how to use the Scan Agent toolkit to develop custom scan-agent plugins.

Product Support Resources

The following resources are available to assist you:

- [Reverera Product Documentation](#)
- [Reverera Community](#)
- [Reverera Learning Center](#)
- [Reverera Support](#)

Reverera Product Documentation

You can find documentation for all Reverera products on the [Reverera Product Documentation](#) site:

<https://docs.reverera.com>

Reverera Community

On the [Reverera Community](#) site, you can quickly find answers to your questions by searching content from other customers, product experts, and thought leaders. You can also post questions on discussion forums for experts to answer. For each of Reverera's product solutions, you can access forums, blog posts, and knowledge base articles.

<https://community.reverera.com/s/>

Revenera Learning Center

The Revenera Learning Center offers free, self-guided, online videos to help you quickly get the most out of your Revenera products. You can find a complete list of these training videos in the Learning Center.

<https://learning.revenera.com>

Revenera Support

For customers who have purchased a maintenance contract for their product(s), you can submit a support case or check the status of an existing case by first logging into the [Revenera Community](#), clicking **Support** on the navigation menu to open the **Support Hub** page, and then clicking the **Open New Case** or **Case Portal** button.

Contact Us

Revenera is headquartered in Itasca, Illinois, and has offices worldwide. To contact us or to learn more about our products, visit our website at:

<http://www.revenera.com>

You can also follow us on social media:

- [Twitter](#)
- [Facebook](#)
- [LinkedIn](#)
- [YouTube](#)
- [Instagram](#)

Installing and Configuring Standard Plugins

Code Insight supports *scan-agent plugins*, which are installed directly in development environments to perform scans on pre-build source files or post-build artifacts as part of the software development process. These remote plugin scans provide an alternative to scans performed on codebases that are uploaded to the Code Insight, as described in the *Code Insight User Guide*.

The following sections discuss the download, installation, and configuration of the plugins available for various types of build environments:

- [About Scan-Agent Plugins](#)
- [Overview of Available Plugins](#)
- [Preparing to Use the Plugins](#)
- [Providing an Authorization Token](#)
- [Downloading Plugins](#)
- [Plugins for Integrated Development Environments \(IDEs\)](#)
- [Plugins for Continuous Integration \(CI\) Tools](#)
- [Plugins for Package Managers and Build Tools](#)
- [Plugins for Container Platforms](#)
- [Generic Scan-Agent Plugin](#)
- [Status of the Import of Remote Scan Results into Your Project](#)
- [Support for Processing Remote Scan Results in the Background](#)
- [Note About Rescans Performed by v2.0 and Later Plugins](#)

About Scan-Agent Plugins

Once a Code Insight scan-agent plugin is installed and the scan is configured as part of your build process, the scan agent, when run, collects and sends the scan results back to a project in Code Insight. The results provide information about the scanned files (including any license evidence found) and published inventory awaiting review, management, and remediation through Code Insight user interface. As with published inventory generated by the Code Insight scan server, published inventory generated by a scan-agent plugin can be automatically reviewed by license or security policies as part of the scan and, for inventory not reviewed by policy, can be reviewed manually by legal or security experts. Security alerts with corresponding email notifications will be generated for any inventory item with new security vulnerabilities.

No Backward Compatibility with Code Insight Releases Previous to 2022 R2

The scan-agent plugins released in the current version of Code Insight are not compatible with Code Insight versions previous to 2022 R2. That is, if you attempt to run a 2022 R2 or later scan-agent plugin with Code Insight 2022 R1 or earlier, the plugin scan will fail.

Supported Rescan Type

After the initial full scan of a file system, subsequent scans by a scan-agent plugin will be incremental only. No forced full rescans are supported.



Note ▪ Scan-agent plugin support for incremental rescans was introduced in Code Insight 2022 R2. Prior to 2022 R2, scan agent plugins supported only full rescans for both the initial scan and all rescans. These previous plugins are still compatible with Code Insight 2022 R2 and later, but they will continue to run full scans only (with no support for incremental scanning).

Overview of Available Plugins

Code Insight provides the following plugins that enable data (codebase files) on remote servers to be scanned:

Table 2-1 ▪ Overview of the Standard Plugins

Build Environment	Code Insight Plugin	Performs automated scanning of...
IDEs	Eclipse Plugin	An Eclipse workspace in the Eclipse IDE environment.
	Visual Studio Plugin	A Visual Studio solution.

Table 2-1 ■ Overview of the Standard Plugins (cont.)

Build Environment	Code Insight Plugin	Performs automated scanning of...
CI Tools	Azure DevOps Extension	An Azure DevOps workspace as part of the build process.
	Bamboo Plugin	A Bamboo workspace as part of the build process (on Local Agents only)
	GitLab Plugin	GitLab projects as part of the build process.
	Jenkins Plugin	A Jenkins workspace as part of the build process. A separate plugin is available (called the Scan Schedule Plugin) that enables you to simply schedule the scan of a codebase residing on the Code Insight scan server via the Jenkins scheduler.
	TeamCity Plugin	TeamCity projects as part of the build process.
Package Manager and Build Tools	Apache Ant Plugin	Apache Ant as part of the build process.
	Gradle Plugin	Gradle projects as part of the build process.
	Maven Plugin	Maven projects as part of the build process.
Container Platforms	Docker Images Plugin	Docker images on a Docker server.

Additionally, a generic scan-agent plugin is available with Code Insight that enables you to scan arbitrary file systems of your choice. It also easily integrates with certain Engineering systems, such as TeamCity and GitLab, to perform scans as part of a build process or can serve as an example for developing your own scan-agent plugin (as described in the chapter [Developing Custom Plugins](#)). All the scan-agent plugins send results to Code Insight for further review and action.

Important Plugin Upgrade in Code Insight 2020 R3

An important Code Insight plugin change occurred when Code Insight scan-agent plugins were upgraded from version 1.x to 2.0 in the Code Insight 2020 R3 release.

A v1.x plugins requires an inventory-only project on the Code Insight Core Server to which to send only the inventory results from the remote scans. However, a v2.0 (or later) plugin, whose scans capture both inventory and codebase-file information, sends scan results to a new project type that is capable of managing the data from both server scans (performed by the Scan Server) and remote scans. For information about this new project type, see “Legacy Projects” in the *Code Insight User Guide*.

Note that the configuration of all v2.0 and later plugins requires a new “alias” property to identify the scan agent to Code Insight. A new “host” property is also required for certain plugins. See [Note About Rescans Performed by v2.0 and Later Plugins](#) for details.

Code Insight continues to support existing inventory-only projects, enabling users to scan these projects using version 1.x plugins installed from previous Code Insight releases. However, inventory-only projects will be deprecated in a future release. If you want to manually migrate your inventory-only projects to the new project type, refer to the following Knowledge Base article in the Revenera Community:

<https://community.flexera.com/t5/FlexNet-Code-Insight-Customer/Code-Insight-2020-R3-Changes-to-Projects/ta-p/160059>

Plugins Not Yet Upgraded

In this release, the following scan-agent plugins have been not been upgraded. These 1.x plugins continue to require inventory-only projects, sending only inventory information in the scan results.

- GitLab
- TeamCity

As these scan-agent plugins are updated, you can retrieve the updated documentation from either the [Revenera Product Documentation](#) site or the Flexera Product and Licensing Center.

Preparing to Use the Plugins

Before configuring and using the scan-agent plugins, complete the following tasks:

- Ensure that the Code Insight server is installed and running, as described in the *Code Insight Installation & Configuration Guide*. (Take note of the server's URL, as you will need this information to configure the plugin to access the server.)
- Generate a JSON Web Token (JWT) for a user registered on the Code Insight server. See [Providing an Authorization Token](#) for more information.
- Create a project on the Code Insight server to which the results of scans run by the scan-agent plugin are sent. See "Creating a Project" in the *Code Insight User Guide* for details.
- Ensure that the scan profile associated with the project you created to receive the scan results is configured appropriately to handle dependencies and file exclusions during the scan. See the next section, [Application of Scan Profile Settings](#).

Application of Scan Profile Settings

During a scan on a remote file system, a scan-agent plugin processes certain options from the scan profile currently associated with the Code Insight project created to store the scan results. Ensure that these options are configured according to your requirements. In general, the **Perform Package/License Discovery in Archives**, **Dependency Support**, and **Scan Exclusions** settings are applied to scans performed by scan-agent plugins.



Note - Scan-agent plugin support for the processing of a project's scan-profile settings was introduced in Code Insight 2022 R2. Scan-agent plugins released prior to 2022 R2 are still compatible with Code Insight 2022 R2 and later. However, they do not process scan profile settings, including settings for dependency processing, scan exclusions, and archive processing. Also, when if you are using a pre-2022 R2 plugin to scan files in Code Insight 2022 R2 or later, dependency processing with use of a `codeaware.properties` file is no longer supported.

The following sections provide more information about the supported/non-supported scan-profile options:

- [Support for Archive Processing](#)
- [Support for Dependency Processing](#)
- [Support for Scan Exclusions](#)
- [Support for Incremental Rescans](#)
- [Addition of Files to Related to Inventory Not Supported](#)
- [Code Matching and Search-Term Processing Not Supported](#)

For more information about these options, see “Managing Scan Profiles” in the *Code Insight Installation & Configuration Guide*.

Support for Archive Processing

During a scan, the scan-agent plugin processes archives according to the **Perform Package/License Discovery in Archives** scan profile setting.

Support for Dependency Processing

Based on the **Dependency Support** value defined in the scan profile, the scan-agent plugin does one of the following to process dependencies of the top-level inventory item during a scan:

- Processes direct (first-level) dependencies only
- Processes both direct and transitive dependencies
- Performs no dependency processing

Support for Scan Exclusions

During a scan, the scan-agent plugin processes the list of file extensions defined in the **Scan Exclusions** field in the scan profile to exclude any files using those extensions from the scan.

Support for Incremental Rescans

Once the scan-agent plugin performs the initial full scan on a file system, any subsequent rescans it performs are *always incremental* (that is, only those files changed or new since the most recent scan are scanned). Forced full rescans are not supported. Hence, the plugin ignores the **Rescan Options** field in the scan profile but applies all other supported scan-profile settings to those files that are scanned during a rescan.

If a file has been deleted since the previous scan, its retention during a rescan is based on the project setting **On the data import or rescan, delete inventory with no associated files**. See for more information about this option, see “Editing the Project Definition and General Settings” and the related topic “Edit Project: General Tab” in the *Code Insight User Guide*.

Addition of Files to Related to Inventory Not Supported

Scan-agent plugins do not support the association of additional files to existing inventory items during scans and therefore ignore the **Automatically Add Related Files to Inventory** scan profile option.

Code Matching and Search-Term Processing Not Supported

Scan-agent plugins neither support the matching of exact files or code snippets against the Code Insight Data Library nor use search terms to locate possible third-party code or content. Therefore, the **Exact Matches**, **Source Code Matches**, or **Search Terms** profile options are ignored during scans.

Providing an Authorization Token

Code Insight uses a JSON Web Token (JWT) to authorize user access to the Code Insight public REST interface. Several of the scan-agent plugins make use of the REST APIs and thus require a JWT. For information about obtaining this authorization token, see “Managing Authorization Tokens” in the “Using Code Insight” chapter in the *Code Insight User Guide*.

Downloading Plugins

The scan-agent plugins are provided in a .zip file that is not included with the Code Insight installation. You can access the plugins .zip file from the Reverera Community. The following procedure assumes you have a login and password to access the Reverera Community.



Task

To download the plugin zip file, do the following:

1. Access the Reverera Community site and sign in:
<https://community.reverera.com>
2. In **Find My Product**, click **Code Insight**.
3. Under **Product Resources** on the right, click **Download Product and Licenses**.
4. Once in the Product and License Center, navigate to **Your Downloads** and select **FlexNet Code Insight**. The **Download Packages** page is displayed.
5. Select the version of Code Insight from the list. The **Downloads** page appears.
6. Select the Code Insight Plugins version, and download its associated `CodeInsightversionPlugins.zip` file.
7. When the download finishes, extract and copy the desired plugin subfolder to your build environment (or the location that this document might specify for a particular plugin):
 - For a standard scan-agent plugin (such as Ant, Bamboo, Docker, Gradle, Jenkins, Maven, and others), extract the subfolder that identifies the plugin (such as `code-insight-docker-images-plugin` for the Docker Images plugin).
 - For the Code Insight generic scan-agent plugin (required for Team City or GitLab scans), extract the subfolder `code-insight-agent-sdk-generic-plugin`. This plugin can also be used to scan arbitrary files or can serve as a basis for developing custom scan-agent plugins.

Ensure that you copy the entire subfolder to your build location, so you have all necessary files to implement the plugin.

Plugins for Integrated Development Environments (IDEs)

Currently, Code Insight support for scan integration with an integrated development environment (IDE) includes the following:

- [Eclipse Plugin](#)
- [Visual Studio Plugin](#)

Eclipse Plugin

The Eclipse plugin enables development teams to perform Code Insight scans within their Eclipse IDE environment. The scan results are automatically sent to the Code Insight server for inventory review, management, remediation, and security-alerting through the Code Insight user interface.

To enable this functionality, you need to install the Eclipse plugin and configure it for your Eclipse project:

- [Prerequisites for the Eclipse Plugin](#)
- [Installing the Eclipse Plugin](#)
- [Configuring the Eclipse Plugin](#)
- [Running a Scan within Your Eclipse Environment](#)
- [Uninstalling the Eclipse Plugin](#)

Prerequisites for the Eclipse Plugin

Before you can install and configure the Code Insight plugin for Eclipse, perform the required tasks described in [Preparing to Use the Plugins](#) for details.

Installing the Eclipse Plugin

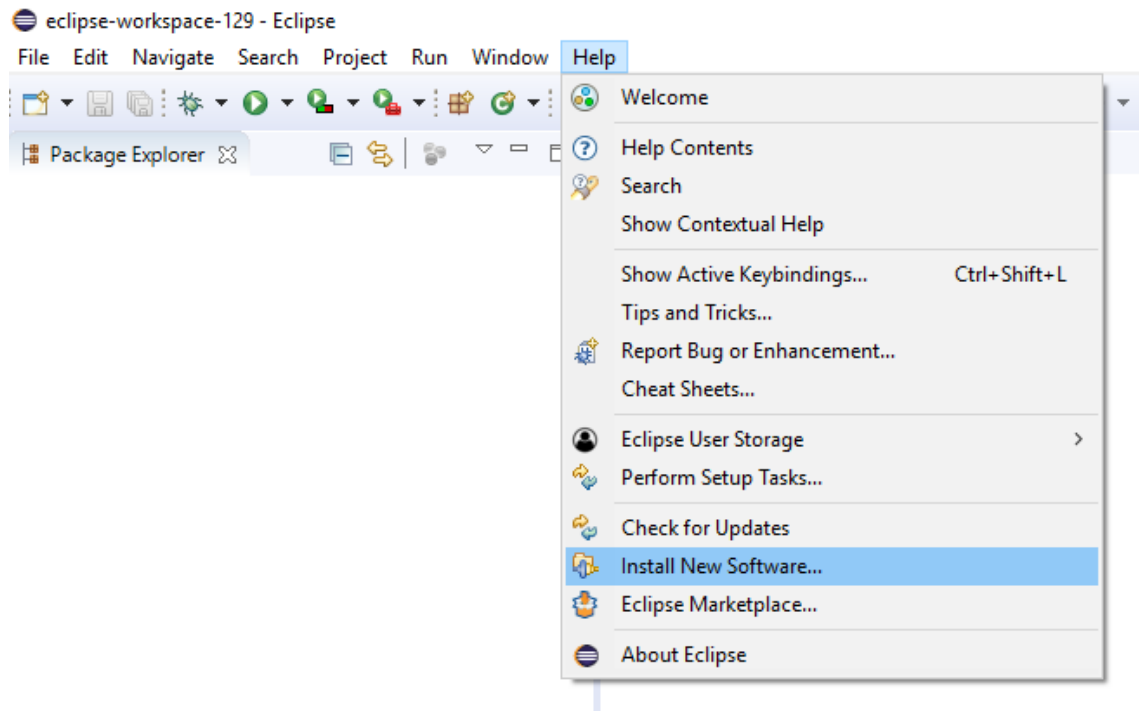
Once you have ensured that the prerequisites are met, use this procedure to install the Eclipse plugin in your Eclipse environment.



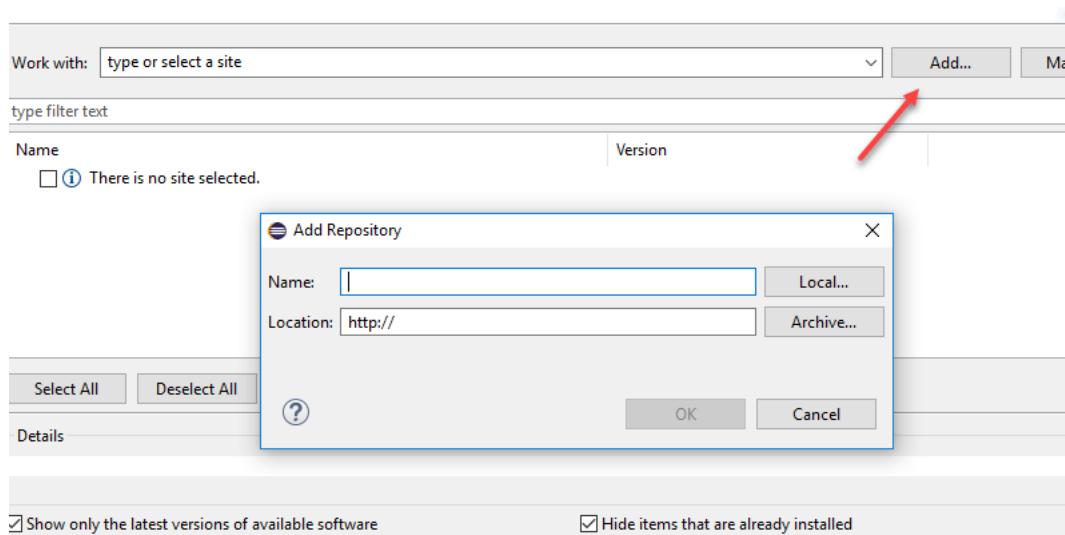
Task

To install the Eclipse plugin, do the following:

1. Ensure that you have extracted the code-insight-eclipse-scan folder from the CodeInsightversionPlugins.zip file and have placed it in a location accessible to your environment. For more information, see [Downloading Plugins](#).
2. Open Eclipse, and select **Help | Install New Software**.

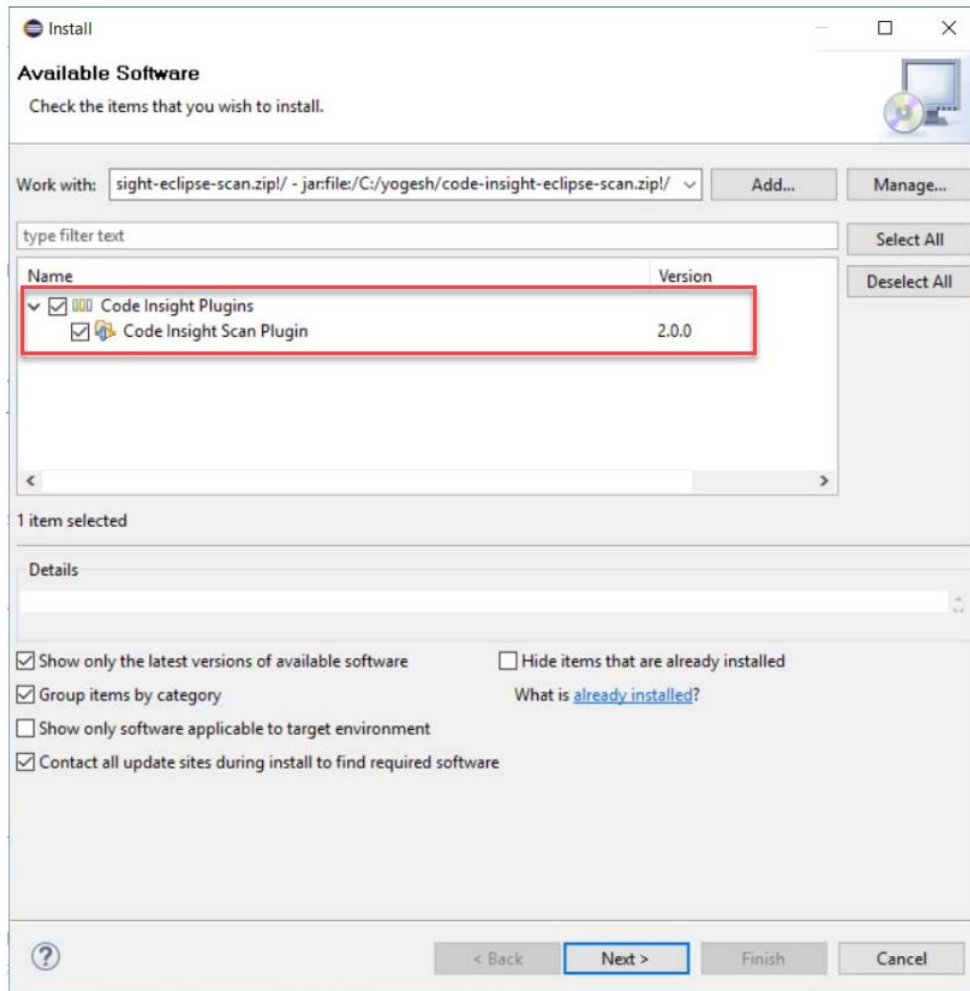


3. On the **Available Software** window, click the **Add** to display the **Add Repository** popup.

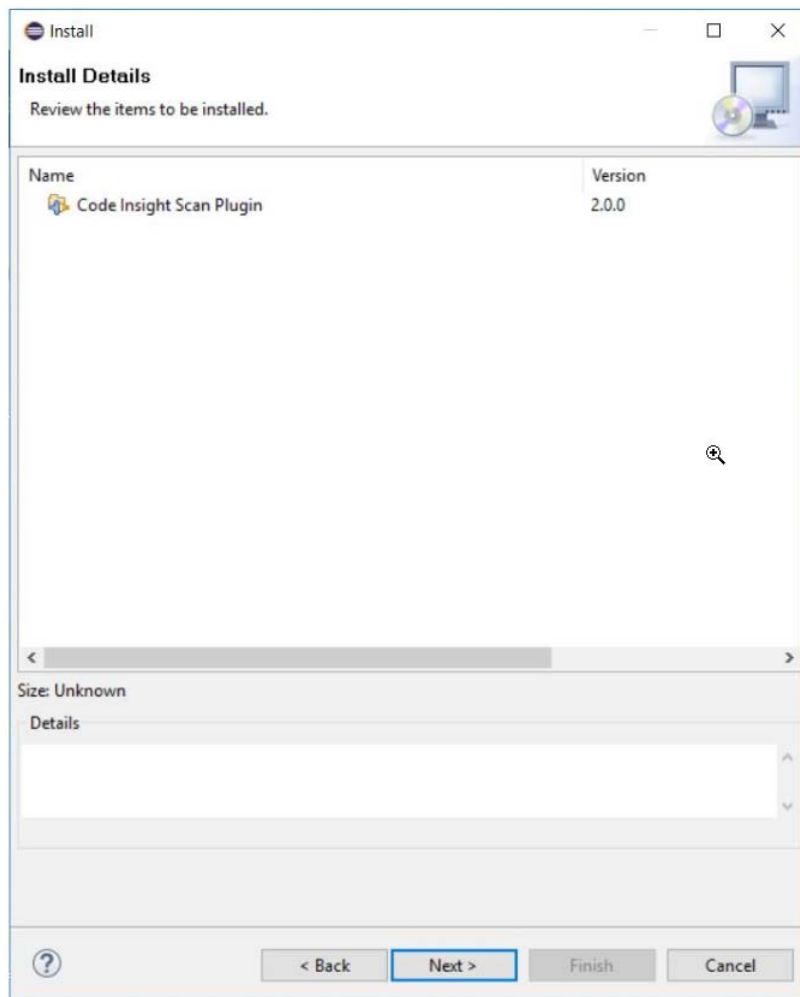


4. On the popup, click **Archive**.
5. Browse to the code-insight-eclipse-scan/code-insight-eclipse-scan.zip file, select it, and click **Open**. Then click **OK**.

The **Available Software** window is redisplayed, showing the plugin information you selected.



6. Click the checkbox next to the **Code Insight Scan Plugin** entry, and click **Next**. The **Install Details** window is displayed, listing the plugin you are preparing to install.



7. Click **Next** to display the **Review Licenses** window.
8. Accept the license agreement terms, and click **Finish**.
9. When the installation is completed, restart Eclipse.

Configuring the Eclipse Plugin

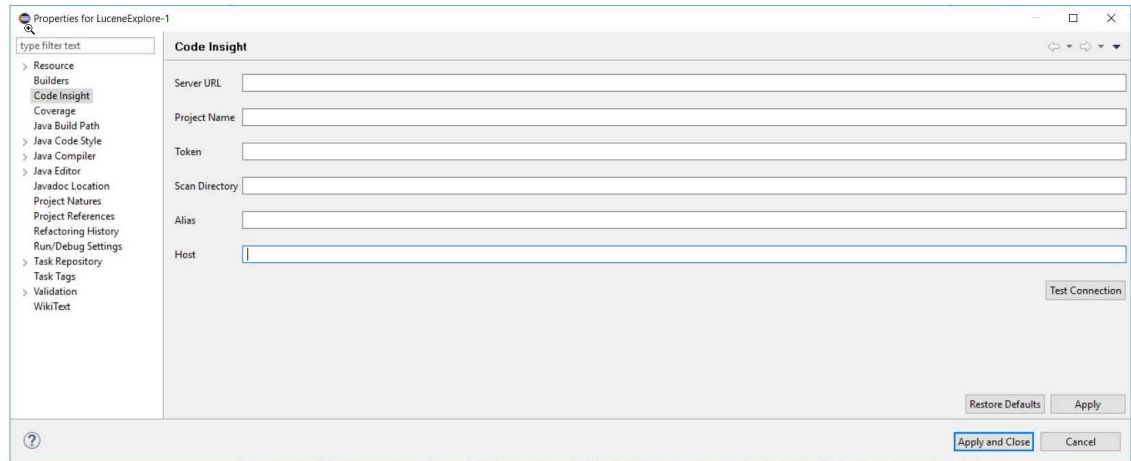
Once you have installed the Eclipse plugin and have restarted Eclipse, follow this procedure to configure the plugin to perform codebase scans on your project in Eclipse.



Task

To configure the Eclipse plugin, do the following:

1. In Eclipse, select the project whose codebase you want to scan, right-click the project entry, and select **Properties**.
2. On the **Properties** window for your project, select **Code Insight** in the left pane to display the properties needed for a Code Insight scan.



3. In the **Properties** window for your project, provide the following property values:

Field	Description
Server URL	The URL for the Code Insight core server (for example, http://codeinsightserver.myorg.org:8888/codeinsight/). Ensure that the URL is publicly accessible and that the port is available.
Project Name	The name of the project that was created in the Code Insight user interface (for example, ScanProject2_eclipse).
Token	The JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it in this field. For more information, see Providing an Authorization Token .
Scan Directory	The folder containing the code to scan (for example, C:\Users\user1\eclipse-workspace\project2_code).
Alias	A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the Analysis Workbench . This name must be unique within the project.
Host	<p>(Optional) A user-defined name for the instance where the scan-agent plugin is configured to run scans. This property along with the <code>alias</code> property will remain unchanged for each subsequent rescan.</p> <p>Although optional in general, this value is required if you are running the scan in a dynamic host environment. See Note About Rescans Performed by v2.0 and Later Plugins.</p>

4. Click **Test Connection** to ensure you can connect to the Code Insight server. A message box is displayed, indicating whether the test is successful. If the test is unsuccessful, adjust the property values as needed and retest the connection.

- When the plugin has been properly configured, click **Apply and Close**.

Running a Scan within Your Eclipse Environment

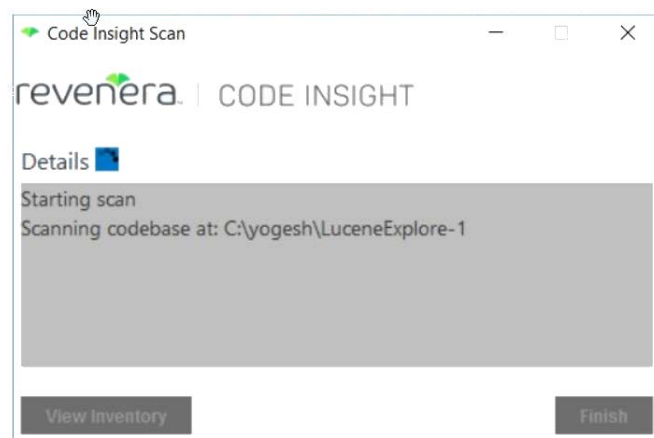
Once the Eclipse plugin has been properly configured, you can invoke a scan on your codebase.



Task

To run a scan on your codebase, do the following:

- In Eclipse, select the project whose codebase you want to scan, right-click the project entry, and select **Code Insight | Scan Project**. The Code Insight scan window is opened, enabling you to keep track of the scan progress.



- When the scan completes, do one of the following:
 - Click **Finish** to close the Code Insight scan window.
 - Click **View Inventory** to connect to Code Insight, which opens to the **Project Inventory** tab for the project created for the scan. From here you can review, manage, and remediate the inventory resulting from the scan. For further instructions, refer to “Reviewing Published Inventory” in the “Using Code Insight” chapter in the *Code Insight User Guide*.

Uninstalling the Eclipse Plugin

Use the following procedure to uninstall the Eclipse plugin.



Task

To uninstall the Eclipse plugin, do the following:

- Open Eclipse, and select **Help | About Eclipse**.
- In the **About Eclipse** window, click **Installation Details**.
- In the **Eclipse Installation Details** window, select **Code Insight Scan Plugin**, and click **Uninstall....**
The **Uninstall** window is displayed, listing the plugin.
- Click **Finish** to confirm that you want to uninstall the plugin and to start the process.

5. Restart Eclipse when prompted to do so.
6. Navigate to the plugins folder in Eclipse (under either *eclipse-home/plugins* or *userhome/.p2/pool/plugins*); and remove the *code-insight-eclipse-scan_X.0.0* folder.
7. Navigate to and open the *artifacts.xml* file in Eclipse (under either *eclipse-home/artifacts.xml* or *userhome/.p2/pool/artifacts.xml*), and remove the *codeinsight-Eclipse-scan* and *flexNet.code.insight.scan.plugin* sections from the file.
8. Navigate to the features folder in Eclipse (under either *userhome/.p2/pool* or *eclipse-home*), and remove the *flexNet.code.insight.scan.plugin_X.0.0* folder (if it exists).

Visual Studio Plugin

The Code Insight plugin for Visual Studio (called the *Visual Studio plugin*) enables development teams perform Code Insight scans within their Microsoft Visual Studio IDE environment. The scan results are automatically sent to the Code Insight server for inventory review, management, remediation, and security-alerting through the Code Insight user interface.

To enable this functionality, you need to install the Visual Studio plugin and configure it for your Visual Studio solution, as described in the following topics. Plugin configuration and scan execution can be performed through either the Visual Studio IDE interface or MSBuild.

- [Prerequisites for the Visual Studio Plugin](#)
- [Installing the Visual Studio Plugin](#)
- [Configuring the Visual Studio Plugin](#)
- [Executing a Scan](#)
- [Disabling or Uninstalling the Visual Studio Plugin](#)

Prerequisites for the Visual Studio Plugin

Before you can install and configure the Visual Studio plugin, perform the required tasks described in [Preparing to Use the Plugins](#). Note that one of these required tasks is to create a project on the Code Insight server in which to store scan results for analysis and review in Code Insight. If you prefer, you can have the configuration process for the Visual Studio plugin create this project for you, as described later in [Configuring the Visual Studio Plugin](#).



Note - Microsoft Visual Studio Express does not support the Visual Studio plugin.

Installing the Visual Studio Plugin

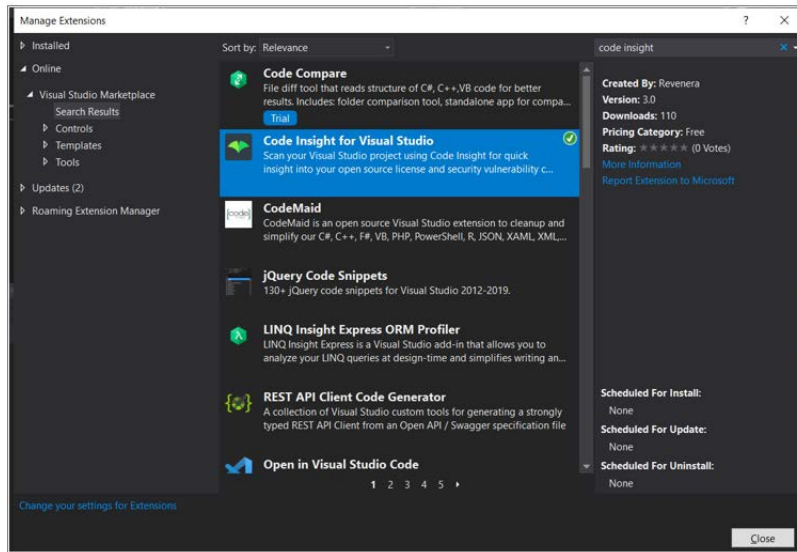
Use Visual Studio IDE to install the Visual Studio plugin extension.



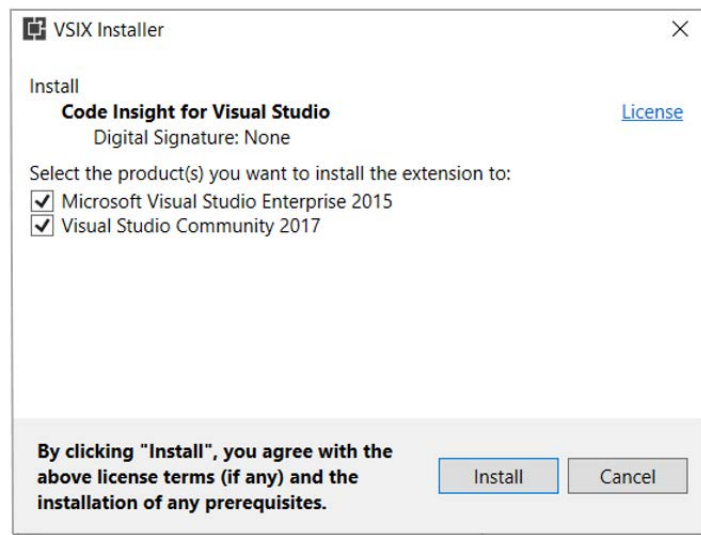
Task

To install the Visual Studio plugin, do the following:

1. In Visual Studio IDE, select **Extensions | Manage Extensions**.
2. In the **Extensions and Updates** tree, search for the **Code Insight for Visual Studio** extension in the **Online | Visual Studio Marketplace** category (or the **Visual Studio Gallery** category, depending on the Visual Studio version). The following shows the menu flow for the extension in Visual Studio 2019.



3. When you locate the extension, click **Download** to download and automatically launch the plugin installer. You are prompted to select the Visual Studio version to which you are installing the plugin.
4. Select the appropriate Visual Studio version, and click **Install**.



5. When the installation completes, restart Visual Studio IDE to apply the extension.

Configuring the Visual Studio Plugin

After the plugin is installed, it must be configured in Visual Studio to enable codebase scans for a specific solution. If you have not already created a Code Insight project in which to store the scan results on the Code Insight server, the configuration process can create this project for you.

You can configure the plugin either in Visual Studio IDE or through the MSBuild command interface:

- [Configuration Using Visual Studio IDE](#)
- [Configuration Using MSBuild](#)


Configuration Using Visual Studio IDE

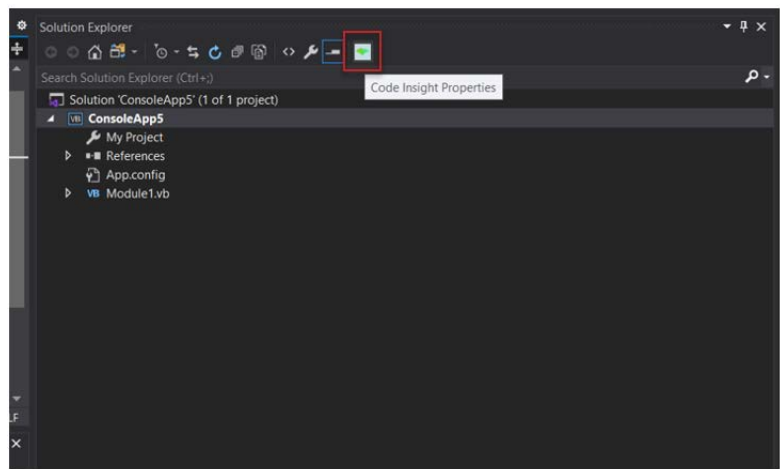
During the Visual Studio plugin installation, the **Code Insight Properties** icon  is added to the **Solution Explorer** toolbar in Visual Studio IDE. This icon provides access to the settings needed to configure the plugin at the solution level.

The following steps describe this configuration process.



Task *To use the Visual Studio IDE interface to configure the Visual Studio plugin, do the following:*

1. In Visual Studio IDE, open the Visual Studio solution that you intend to scan.
2. In the **Solution Explorer** toolbar, click the **Code Insight Properties** icon .



The **Code Insight Settings** dialog is displayed.

The screenshot shows the 'Code Insight Settings' dialog box. It features the 'revenera | CODE INSIGHT' logo at the top. The settings are organized into several sections: 'Code Insight Server*' with an empty text field; 'Authentication Token*' with an empty text field; 'Code Insight Project*' with an empty text field and two checkboxes below it, 'Create New Project' and 'Create Private Project'; 'Alias*' with an empty text field; 'Folders to Scan*' with a text field containing 'c:\users\administrator\documents\visual studio' and a file explorer button, and a checkbox 'Scan Solution After Build'; and 'Host' with an empty text field. At the bottom are 'OK' and 'Cancel' buttons.

3. Complete the following fields on the dialog to configure the Visual Studio plugin for the current solution. All fields with an asterisk are required.

Field	Description
Code Insight Server	<p>Enter the URL for the Code Insight core server in the format <code>http://<SERVERHOSTNAME>:<PORT>/codeinsight/</code> (for example, http://codeInsightServer.myorg.org:8888/codeinsight/).</p> <p>Ensure that the URL is publicly accessible and that the port is available.</p>
Authentication Token	<p>Provide the JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. You generate this token using the Code Insight Web UI and then copy and paste it in this field. For more information, see Providing an Authorization Token.</p>
Code Insight Project	<p>Enter the name of the Code Insight project that already exists or that you want this configuration process to create on the Code Insight server to store scan results.</p> <p>If you want the configuration process to create the specified Code Insight project, select Create New Project (see the next table entry).</p> <p>If the specified project already exists on the Code Insight server, do <i>not</i> select Create New Project or Create Private Project (see the next table entries).</p>

Field	Description
Create New Project	<p>Select this option if you want the configuration process to create a project in which to store scan results on the Code Insight server. The project is the name specified for Code Insight Project. The Project Owner is the user who generated the JWT provided for Authentication Token.</p> <p>If you want this new project to be <i>public</i>—that is, viewable by all Code Insight users—leave the next option, Create Private Project, unselected.</p> <p>Otherwise, select Create Private Project (described next).</p>
Create Private Project	<p>Select this option if you selected Create a New Project and want this new project to be a <i>private</i>—that is, accessible by only the Project Contact and users assigned to project roles. The Project Contact is the user who generated the JWT provided for Authentication Token.</p> <p>Leave this option unselected if you want the new project to be public.</p>
Alias	<p>A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the Analysis Workbench. This name must be unique within the project.</p>
Folders to Scan	<p>The auto-generated list of codebase folders to scan, based on the current output configuration of the Visual Studio project. To append additional codebase folders to scan, specify their absolute paths, separating each with a comma.</p> <div data-bbox="631 1131 667 1176" data-label="Image"> </div> <p>Note ▪ This field is pre-populated with output directories for only those project types that have configuration support. For project types that do not support configuration, such as Silverlight or JavaScript, you must specify the absolute path for each folder to scan, separating each path with a comma.</p>
Scan Solution After Build	<p>Select this option to execute the Code Insight scan automatically after the Build or Rebuild Solution step. If this option is not selected, you must start each scan manually, as described in Executing a Scan.</p>
Host	<p>(Optional) A user-defined name for the instance where the scan-agent plugin is configured to run scans. This property along with the <code>alias</code> property will remain unchanged for each subsequent rescan.</p> <p>Although optional in general, this value is required if you are running the scan in a dynamic host environment. See Note About Rescans Performed by v2.0 and Later Plugins.</p>

- Click **OK** to save the plugin configuration.

Configuration Using MSBuild

The following steps describe how to use MSBuild to configure the Visual Studio plugin once it is installed.



Task

To use the Visual Studio IDE interface to configure the Visual Studio plugin, do the following:

1. Launch Visual Studio IDE in **Run As Administrator** mode to enable the Visual Studio plugin for MSBuild and create a project/solution. You need to perform this step only once.
2. Copy the template configuration file `codeinsight_scan_settings.ini` from `<LOCAL_APP_DATA>\Local\Microsoft\VisualStudio` to the Visual Studio solution folder you want to scan.

The following is an example solution folder to which to copy the file: `C:\Users\jsmith\Documents\Visual Studio 2015\Projects\MyProject`.

3. In a text editor, open the `codeinsight_scan_settings.ini` that you copied to the solution folder, and provide the following values in the Settings section:

Property	Description
CodeInsight Server	<p>Provide the URL for the Code Insight core server in the format <code>http://<SERVERHOSTNAME>:<PORT>/codeinsight/</code> (for example, <code>http://codeInsightServer.myorg.org:8888/codeinsight/</code>).</p> <p>Ensure that the URL is publicly accessible and that the port is available.</p>
AuthenticationToken	<p>Provide the JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. You generate this token using the Code Insight Web UI and then copy and paste it in this field. For more information, see Providing an Authorization Token.</p>
CodeInsight Project	<p>Provide the name of the Code Insight project that already exists or that you want this configuration process to create for you on the Code Insight server to store scan results.</p> <p>If you want the configuration process to create the Code Insight project you specified here, also set CreateNewProject to True (see the later table entry).</p> <p>If the specified project already exists on the Code Insight server, ensure that CreateNewProject and CreatePrivateProject are set to False.</p>
ScanFolders	<p>Specify the absolute paths for the project output folders (or any additional folders) to scan for the solution, separating each path with a comma.</p>

Property	Description
CreateNewProject	<p>If the Code Insight project you specified for CodeInsightProject already exists, set this property to False.</p> <p>However, if you want the plugin-configuration process to create a new project in which store scan results on the Code Insight server, set this property to True. The project is the name specified for CodeInsightProject. The Project Owner is the user who generated the JWT provided for AuthenticationToken.</p> <p>Use the CreatePrivateProject property to define the new project as public or private.</p>
CreatePrivateProject	<p>Determine whether the new project is to be created as public or private:</p> <ul style="list-style-type: none"> • If you want this new project to be <i>public</i>—that is, viewable by all Code Insight users, set this property to False. • If you want this project to private to <i>private</i>—that is, viewable and managed by the Project Owner and select users, set this property to True. (The Project Owner is the user who generated the JWT provided for AuthenticationToken.)
ScanAfterBuild	<p>Specify True to have the Code Insight scan execute automatically after the Build or Rebuild Solution step.</p> <p>Specify False to start each scan manually, as described in Executing a Scan.</p>
Alias	<p>A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the Analysis Workbench. This name must be unique within the project.</p>
Host	<p>(Optional) A user-defined name for the instance where the scan-agent plugin is configured to run scans. This property along with the <code>alias</code> property will remain unchanged for each subsequent rescan.</p> <p>Although optional in general, this value is required if you are running the scan in a dynamic host environment. See Note About Rescans Performed by v2.0 and Later Plugins.</p>

4. Save the file changes.

Executing a Scan

Once the Visual Studio plugin has been properly configured, you can manually invoke a scan on your solution codebase whenever needed. Trigger the scan from either Visual Studio IDE or through MSBuild commands:

- [Scan Execution Using Visual Studio IDE](#)
- [Scan Execution Using MSBuild](#)

The Visual Studio plugin can also be configured to trigger a scan automatically at the end of each build or rebuild. (See [Configuring the Visual Studio Plugin](#) for configuration details.) When the scan completes, you can click the URL at the end of the build output to connect to Code Insight. You are opened to the **Project Inventory** tab, where you can review and remediate the project inventory resulting from the scan. Refer to “Reviewing Published Inventory” in the “Using Code Insight” chapter in the *Code Insight User Guide*.

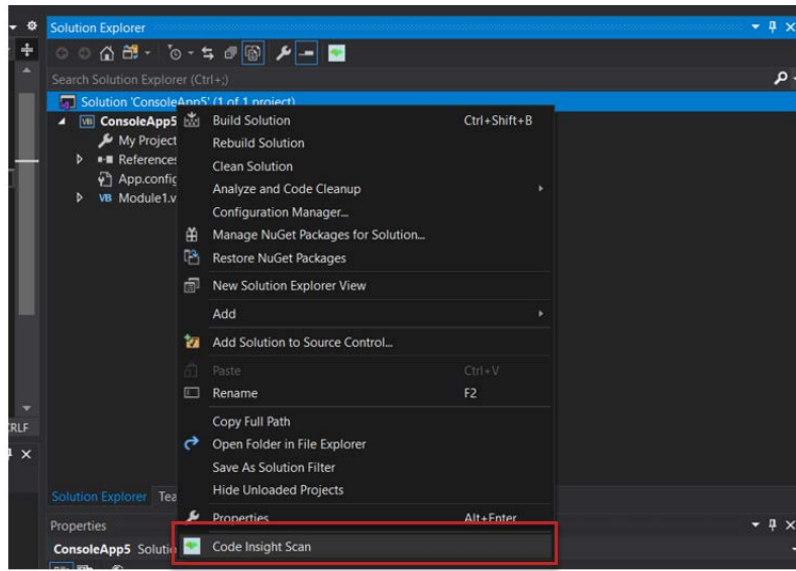
Scan Execution Using Visual Studio IDE

Use the following procedure to manually invoke a Code Insight scan on your solution codebase in Visual Studio IDE.



Task To run a scan using Visual Studio IDE, do the following:

1. In the Solution Explorer in your Visual Studio IDE, navigate to the solution you want to scan.
2. Right-click the solution entry, and select **Code Insight Scan** to start the scan.



3. When the scan completes, click the URL at the end of the build output to connect to Code Insight. You are opened to the **Project Inventory** tab for the Code Insight project created for the scan. From here you can review, manage, and remediate the inventory resulting from the scan. For further instructions, refer to “Reviewing Published Inventory” in the “Using Code Insight” chapter in the *Code Insight User Guide*.

Scan Execution Using MSBuild

The following procedure uses MSBuild commands to manually invoke a Code Insight scan on your solution codebase.



Task

To run a scan using MSBuild, do the following:

1. Open a command-line prompt as an administrator, navigate to the MSBuild directory.

Note that these directories might vary based on your Visual Studio version:

Visual Studio Version	MSBuild Directory
VS2017	c:\Program Files (x86)\Microsoft Visual Studio\2017\<INSTALLED_EDITION>\MSBuild\15.0\Bin
VS2015	C:\Windows\Microsoft.NET\Framework\<FRAMEWORK_VERSION> C:\Program Files (x86)\MSBuild\14.0\Bin
VS2019	C:\Program Files (x86)\Microsoft Visual Studio\2019\<INSTALLED_EDITION>\MSBuild\Current\Bin

2. Enter the following command:

```
MSBuild.exe "<PATH_OF_SOLUTION_FILE>" /p:CodeInsightScan=true
```

The command uses these parameters:

- **<PATH_OF_SOLUTION_FILE>**—The absolute path of the solution directory you are scanning.
- **CodeInsightScan**—The parameter indicating that you want to run a Code Insight scan. Set this parameter to **true**. If you omit this option or set it to **false**, no scan is run.
- **CodeInsightConfig**—(Optional) The absolute path of the .ini configuration file if it does not reside in the solution directory specified for <PATH_OF_SOLUTION_FILE>. If you provide a value for this option, use the following command syntax:

```
MSBuild.exe "<PATH_OF_SOLUTION_FILE>" /p:CodeInsightScan=true;  
CodeInsightConfig=<ABSOLUTE_PATH_TO_INI_CONFIG_FILE>
```

3. When the scan completes, click the URL at the end of the build output to connect to Code Insight. You are opened to the **Project Inventory** tab for the Code Insight project created for the scan. From here you can review, manage, and remediate the inventory resulting from the scan. For further instructions, refer to “Reviewing Published Inventory” in the “Using Code Insight” chapter in the *Code Insight User Guide*.

Disabling or Uninstalling the Visual Studio Plugin

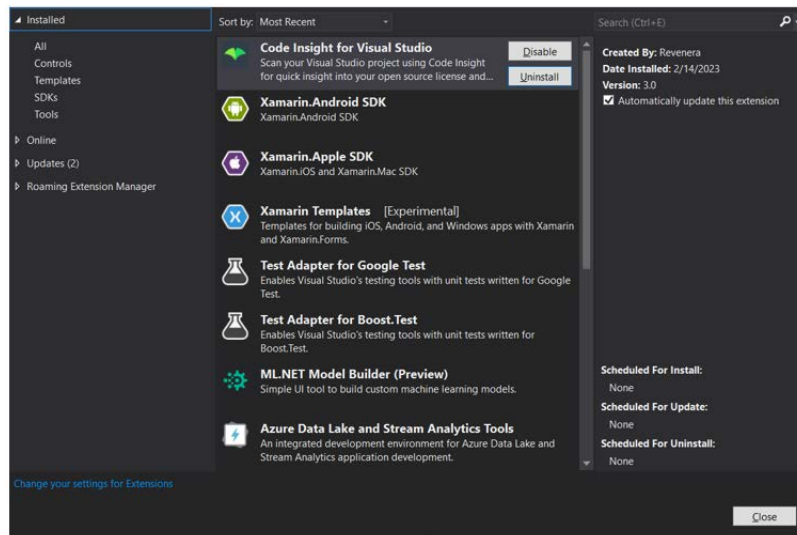
Use the following procedure to disable or uninstall the Visual Studio plugin.



Task

To uninstall the Visual Studio plugin, do the following:

1. Open Visual Studio IDE, and select **Tools | Extensions and Updates**.
2. Select the **Installed** category in the left pane.
3. From the list of applications, select **Code Insight Scan for Visual Studio**, and click **Disable** or **Uninstall**.



4. Restart Visual Studio IDE to verify that the plugin has been disabled or removed from the list of applications.

Plugins for Continuous Integration (CI) Tools

This section described how to configure the Code Insight scan-agent plugins that integrate with continuous integration (CI) tools:

- [Azure DevOps Extension](#)
- [Bamboo Plugin](#)
- [GitLab Plugin](#)
- [Jenkins Plugin](#)
- [Scan Scheduler Plugin for Jenkins](#)
- [TeamCity Plugin](#)
- [Configuring SSL for Azure DevOps, Bamboo, and Jenkins Scan Agents](#)

Azure DevOps Extension

A Code Insight extension for Azure DevOps is available for downloading from the Visual Studio Marketplace. This extension allows development teams to easily integrate Code Insight scanning into their Azure DevOps build process and send scan results to the Code Insight server for inventory review, management, remediation, and security-alerting through the Code Insight user interface.

To enable this functionality, you need to install the extension and configure the build process to include the scan:

- [Prerequisites for the Azure DevOps Extension](#)
- [Installing the Azure DevOps Extension](#)
- [Adding a Scan Task to Your Azure DevOps Agent Job](#)

Prerequisites for the Azure DevOps Extension

Before you can install and configure the Azure DevOps extension, perform the required tasks described in [Preparing to Use the Plugins](#) for details.

Installing the Azure DevOps Extension

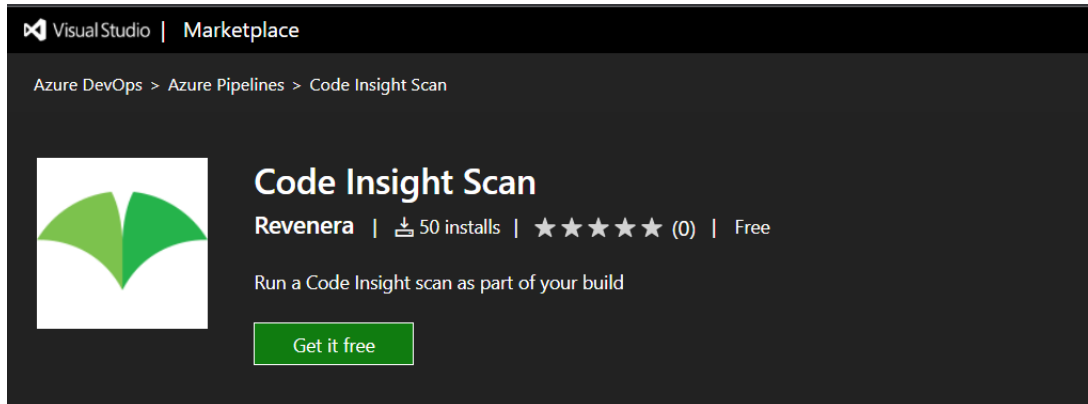
To obtain and install the Azure DevOps extension, perform the following steps.



Task

To obtain and install the extension, do the following:

1. Open the Visual Studio Marketplace:
<https://marketplace.visualstudio.com/>
2. In the **Azure DevOps** section, search for the **Code Insight Scan** extension.



3. Download and install this extension into Azure DevOps.

Adding a Scan Task to Your Azure DevOps Agent Job

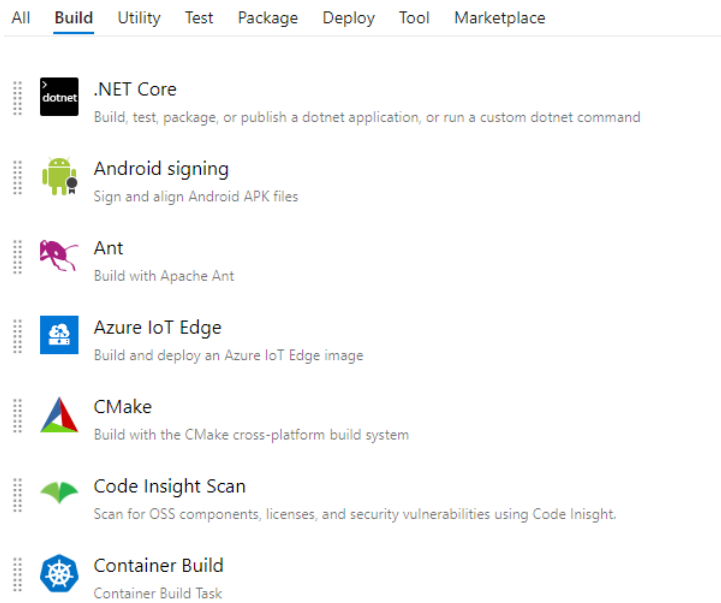
After the Azure DevOps extension has been installed, you need to add a Code Insight scan task to your Azure DevOps agent job so that the scan is automatically performed as part of your build process.



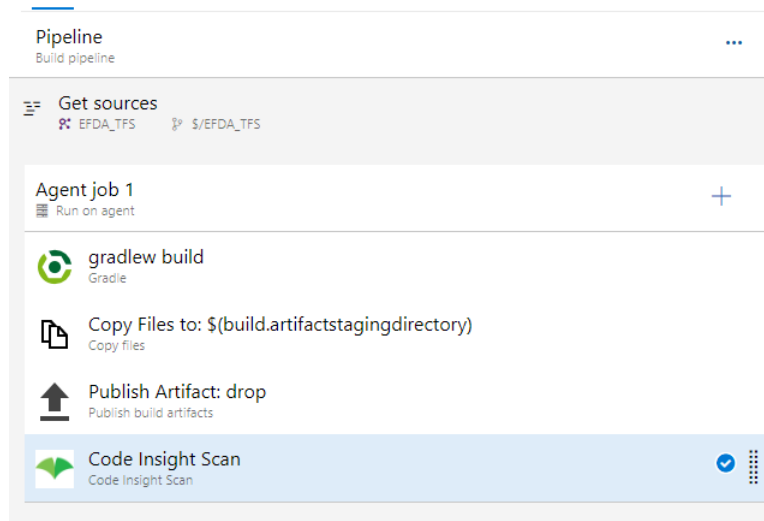
Task

To add a scan task to your DevOps agent job, do the following:

1. Create a build pipeline for your Azure DevOps project.
2. Locate the **Code Insight Scan** task under the **Builds** section in the task catalog.



3. Add the **Code Insight Scan** task at any point after the build task in the pipeline build definition.



4. Define the scan task properties on the **Code Insight Scan** window.

Code Insight Scan ⓘ

[Link settings](#) [View YAML](#) [Remove](#)

Task version 2.* ▾

Display name *

Code Insight Scan

Code Insight Server * ⓘ

This setting is required.

Authentication Token * ⓘ

This setting is required.

Code Insight Project Name * ⓘ

This setting is required.

Alias * ⓘ

This setting is required.

Folder(s) to Scan ⓘ

\$(build.artifactstagingdirectory)

Host * ⓘ

This setting is required.

The following describes the task properties. An asterisk indicates that a value is required.

Field	Description
Display name	The name of the scan task for display purposes.
Code Insight Server	The URL for the core server (for example, http://codeInsightServer.myorg.org:8888/codeinsight/). Ensure that the URL is publicly accessible and that the port is available.
Authorization Token	The JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it in this field. For more information, see Providing an Authorization Token .
Code Insight Project Name	The name of the project that was created in the Code Insight user interface (for example, ScanProject_AzureDevOps).
Alias	A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the Analysis Workbench . This name must be unique within the project.

Field	Description
Folder(s) to Scan	<p>The directory containing the code to scan. Typically, you would use one of the following:</p> <ul style="list-style-type: none">• \$(Build.ArtifactStagingDirectory)—The directory where the build output is staged during the build process• \$(Build.SourcesDirectory)—The directory where the source code files are downloaded• \$(Build.BinariesDirectory)—The output directory for the compiled binaries <p>For details about these directory variables, refer to this site: https://learn.microsoft.com/en-us/azure/devops/pipelines/build/variables?view=azure-devops&tabs=yaml</p>
Host	<p>(Optional) A user-defined name for the instance where the scan-agent plugin is configured to run scans. This property along with the alias property will remain unchanged for each subsequent rescan.</p> <p>Although optional in general, this value is required if you are running the scan in a dynamic host environment. See Note About Rescans Performed by v2.0 and Later Plugins.</p>

5. Save and queue the build definition.

The scan will be performed in the build environment as part of the build process, and the results will be sent to the project you configured on the Code Insight server. The resulting inventory items can be viewed and managed in the Code Insight user interface.

Bamboo Plugin

Code Insight provides a Bamboo plugin that allows automated scanning of a Bamboo workspace as part of your application build process. The scan results are sent to Code Insight for inventory creation, review, and security alerting.

The Bamboo plugin scans only the application root folder.

The following topics describe how to install and configure this plugin on the Bamboo build server:

- [Prerequisites for the Bamboo Plugin](#)
- [Installing and Configuring the Bamboo Plugin](#)

Prerequisites for the Bamboo Plugin

Before you install and configure the Bamboo plugin, ensure that the following prerequisites are met:

- All Code Insight prerequisite tasks for plugins have been performed, as described in [Preparing to Use the Plugins](#).
- Bamboo 5.2 or higher is installed and configured as explained in the Bamboo installation documentation.

- The Bamboo server instance on which you are running the plugin can be a Local Agent or Remote Agent.
- Maximum heap memory size is set to 4 GB for the Bamboo server (Local or Remote Agent, wherever the plugin is running). This size can be configured using the following property in `wrapper.conf`:

```
wrapper.java.maxmemory=4096
```

Note that this is the recommended size value. However, heap size is relative to the size of the codebase. A large codebase requires this value to be increased 6GB or 8 GB.

- Event details for scans run on a Local Agent are logged to the `<BAMBOO_HOME>/logs/atlassian-bamboo.log` file. On a Bamboo Remote Agent, the events are logged to the `<BAMBOO_REMOTE_DIR>/logs/atlassian-bamboo.log` file.

Installing and Configuring the Bamboo Plugin

The following procedure covers installing and configuring the Bamboo plugin, which requires you to perform actions in both Bamboo and Code Insight.



Task

To install and configure the Bamboo plugin, do the following:

1. Extract the Bamboo plugin from the `CodeInsightversionPlugins.zip` file. For more information, see [Downloading Plugins](#).
2. Access your Bamboo server instance.
3. From the **Bamboo Administration** icon, click **Add-ons**.
4. Click **Upload add-on**.
5. Browse to the `code-insight-bamboo-scan.jar` and click **Upload**. The Bamboo jar file is located wherever the zip file containing the plugins was extracted.
6. Create a project in Bamboo by creating the plan, adding a job, and then adding a Code Insight Scan task. To create the task, access the **FlexNet Code Insight Scan Task Configuration** window.

FlexNet Code Insight Server URL*

Enter the Code Insight Server Base URL as follows: `http://HOST_NAME:PORT/codeinsight`

Authentication Token*

Enter your Code Insight authentication token.

Project name*

Enter the Code Insight project name corresponding to this Bamboo Task.

Alias*

Enter a unique alias value for your project. The alias represents a container in which all the files scanned in this instance will be shown.

Folder(s) to scan

Enter the folders to scan relative to Job working directory. If you wish to scan the entire working directory, leave this field empty. If you wish to scan a folder say 'artifacts' relative to working directory, enter artifacts in the field. You can provide multiple path separated by comma.

Host

(Optional) A user-defined name for the instance where the scan-agent plugin is configured to run agent scans. You are strongly recommended to provide this value if you are running the scan in a dynamic host environment. (Note that this property along with the alias property will remain unchanged for each subsequent rescan.)

7. Enter the following information in the **FlexNet CodeInsight Scan Task Configuration** window:

- **Task description**—A label for this scan task.
- **Disable this task**—The option to disable or enable the scan task as needed.
- **Server URL**—The URL for the Code Insight core server (for example, <http://codeInsightServer.myorg.org:8888/codeinsight/>).
- **Authentication Token**—The JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it in this field. For more information, see [Providing an Authorization Token](#).
- **Project Name**—The name of the project that you created in the Code Insight to associate with this scan task.
- **Alias**—A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the **Analysis Workbench**. This name must be unique within the project.
- **Folder(s) to scan**—The one or more folders to scan. If you want to scan the entire working folder, leave this field blank. However, to scan specific folders in the working directory, list the path for each folder as relative to the working folder, using commas to separate multiple folders.

For example, the working directory `opt/atlassian/workingDirectory/project` has the following source sub-folders:

```
/opt/atlassian/workingDirectory/project/source1  
/opt/atlassian/workingDirectory/project/source2
```

```
/opt/atlassian/workingDirectory/project/source3  
/opt/atlassian/workingDirectory/project/source4/source4a
```

- To scan the source1 folder only, enter **source1**.
- To scan both source1 and source2, enter the following:
source1, source2
- To scan the source4a folder, enter **source4/source4a**.
- **Host**—(Optional) A user-defined name for the instance where the scan-agent plugin is configured to run scans. This property along with the `alias` property will remain unchanged for each subsequent rescan.

Although optional in general, this value is required if you are running the scan in a dynamic host environment. See [Note About Rescans Performed by v2.0 and Later Plugins](#).

8. Click **Save**. If the **Server URL** and **Token** values are correct, the task will be saved. The next time you run the plan, the automated scan of the workspace will be executed for the configured project as part of the plan.



Note ▪ The scan task should be placed after the build task in the plan's task sequence.

GitLab Plugin

This section explains how to configure GitLab to integrate with the Code Insight generic scan-agent plugin to perform an automatic composition scan as part of the build process in a Windows environment. The following topics are covered.

- [Prerequisites for the GitLab Plugin](#)
- [Integrating the Generic Scan-Agent Plugin with GitLab Runner](#)
- [Executing the Build](#)

Prerequisites for the GitLab Plugin

The following prerequisites are required to integrate GitLab Runner with the Code Insight generic scan-agent plugin:

- A GitLab Runner properly installed on Windows. (Refer to [Install GitLab Runner](#) for instructions.)
- All the prerequisites listed in [Prerequisites for the Generic Scan-Agent Plugin](#).
- The generic scan-agent plugin (as described in [Integrating the Generic Scan-Agent Plugin with GitLab Runner](#)).

Integrating the Generic Scan-Agent Plugin with GitLab Runner

To integrate the Code Insight generic scan-agent plugin with GitLab Runner, you must configure the CI/CD pipeline to perform a scan using the plugin. Use either of these methods to configure the pipeline:

- Using “run_scan.bat” to Configure the CI/CD Pipeline
- Using “codeinsight-<generic-plugin-version>.jar” to Configure the CI/CD Pipeline



Note • The generic scan-agent plugin currently integrates with the GitLab Runner installed only in a Windows environment. See [Prerequisites for the GitLab Plugin](#).

Using “run_scan.bat” to Configure the CI/CD Pipeline

The following procedure uses the run_scan.bat file to configure the CI/CD pipeline to run a generic scan-agent plugin scan as part of your application build.



Task To use “run_scan.bat” to configure the CD/CI pipeline to run a generic scan-agent plugin scan, do the following:

1. Download and extract the contents of the CodeInsightversionPlugins.zip file, as described in the previous section, [Downloading Plugins](#).
2. Locate the generic-plugin-binary folder (in the code-insight-agent-sdk-generic-plugin folder), and copy it to the GitLab-runner folder (that is, your GitLab installation folder).
3. Under the GitLab-Runner folder, create an output folder. For example, create a folder named output.

GitLab Runner uses this folder to store logs and codebase information as part of the background process during the build.

4. Locate the run_scan.bat file in the Gitlab-Runner\generic-plugin-binary folder.
5. Open the file in an editor, and update the following properties to match your environment:
 - For the SET_ROOT_PATH property, provide the path for the output folder created in Step 4. For example:

```
SET ROOT_PATH=D:\GitLab-Runner\output
```

(Adjust the path as needed to match your GitLab Runner installation.)

- For the cd command (in the next line after the SET_ROOT_PATH property), provide the location of the run_scan.bat file in GitLab Runner. For example:

```
cd D:\GitLab-Runner\generic-plugin-binary
```

(Adjust the path as needed to match your GitLab Runner installation.)



Important • If the “cd” command is currently a comment, be sure to uncomment the line.

6. Save the run_scan.bat file.

7. Restart GitLab Runner using the following command:

```
.\gitlab-runner.exe restart
```

8. Locate the `gitlab.ci.yml` file for the repository containing the code you want to scan, and open this file in an editor.

9. Update this file with information pertaining to the plugin scan.

- a. Add the following information defining the scan job:

```
codeinsight_scan:
  stage: test
  only:
    - main
  tags:
    - <tag_for_your_GitLab-Runner>
  script:
    - cmd /Q /C <path\to\generic-plugin-binary\run_scan.bat> "$CODEINSIGHT_PROJECT"
      "$CODEINSIGHT_SERVER" "$AUTH_TOKEN" <codebase_folder_paths> <alias>
```

- b. In the script section, replace the following with information required for the plugin scan:

- **<path\to\generic-plugin-binary\run_scan.bat>**—The path of the `run_scan.bat` file integrated with GitLab Runner, as in the example:

`D:\GitLab-Runner\generic-plugin-binary\run_scan.bat`
- **<Codebase_folder_paths>**—The path of the specific codebase folder to be scanned. Separate multiple folder paths with commas. These folders must be located on the local machine.
- **<alias>**—A user-defined name for the scan-agent plugin, such as `Eportal_Remote on abc.com`. The alias is simply used to represent the “container” (scan root) under which all the scanned files will be listed in the API output and in the file tree in the **Analysis Workbench**. The name must be unique within the Code Insight project.



Note ■ Do nothing with the variables `$CODEINSIGHT_PROJECT`, `$CODEINSIGHT_SERVER`, and `$AUTH_TOKEN` in the script section. These variables will be defined in the pipeline-schedule setup for the plugin scan (described in Step 11) and passed to the command listed.

10. Save the `gitlab.ci.yml` file.
11. When configuring the CI/CD pipeline schedule in the UI to run a Code Insight plugin scan, define the variables required for the scan:

- a. Navigate to the **Variables** section in the UI for the pipeline schedule.
- b. Select the specific variable from the dropdown and provide its value, as described in this list:
 - **CODEINSIGHT_SERVER**—The URL of the Code Insight Core Server (for example, `http://1.1.1.1:8888/codeinsight`).
 - **CODEINSIGHT_PROJECT**—The project you created in Code Insight to store the scan results (for example, `gitlabProject2`).

- **AUTH_TOKEN**—Your JSON Web Token (JWT) used to authorize user access to the Code Insight functionality, as in "eyJhbGciOiJIUzUxMiJ9..." (not showing the complete token).

Generate this token using the Code Insight Web UI and then copy and paste it in this field. For more information, see [Providing an Authorization Token](#).



Note - Do not precede the JWT with the term "Bearer", as it is already provided in the `run_scan.bat` file.

- c. Save the pipeline schedule once it is properly set up.

When triggered, the job for the plugin scan should run successfully.

Using "codeinsight-<generic-plugin-version>.jar" to Configure the CI/CD Pipeline

The following procedure uses the `codeinsight-<generic-plugin-version>.jar` file to configure the CI/CD pipeline to run a generic scan-agent plugin scan as part of your application build.



Task

To use "codeinsight-<generic-plugin-version>.jar" to configure the CD/CI pipeline to run a generic scan-agent plugin scan, do the following:

1. Download and extract the contents of the `CodeInsightversionPlugins.zip` file, as described in the previous section, [Downloading Plugins](#).
2. Locate the `generic-plugin-binary` folder (in the `code-insight-agent-sdk-generic-plugin` folder), and copy it to the `GitLab-Runner` folder (that is, your GitLab Runner installation folder).
3. Restart GitLab Runner using the following command:

```
.\gitlab-runner.exe restart
```
4. Locate the `gitlab.ci.yml` file for the repository containing the code you want to scan, and open this file in an editor.
5. Add the following code to define the job for Code Insight plugin scan.

```
codeinsight_scan:
  stage: test
  only:
    - main
  tags:
    - <tag_for_your_GitLab-Runner>
  script:
    - cmd /Q /C cd <path/to/generic-plugin-binary/>
    - cmd /Q /C java -jar <path/to/generic-plugin-binary/codeinsight-generic-<version>>.jar -server
      "$CODEINSIGHT_SERVER" -project "$CODEINSIGHT_PROJECT" -token "Bearer $AUTH_TOKEN" -root
      "<codebase_path>" -scandirs "<codebase_folder_paths>" -alias <alias>
  variables:
    CODEINSIGHT_SERVER: "<http://<codeinsight_host>:8888/codeinsight>"
    AUTH_TOKEN: "<token>"
    CODEINSIGHT_PROJECT: "<codeinsight_project_name>"
```

See [Plugin Scan Information to Provide in gitlab.ci.yml File](#) for a description of the properties and variables that you need to define in this content for the scan job.

6. Save `gitlab.ci.yml`.
7. Configure the CI/CD pipeline schedule in the UI to run the job for the Code Insight plugin scan.

When triggered, the job for the plugin scan should run successfully.

Plugin Scan Information to Provide in `gitlab.ci.yml` File

The following tables describes the properties and variables that you need to define to set up the job for the Code Insight plugin scan in the `gitlab.ci.yml` file. Refer to Step 5 in the previous section for the context of these properties and variable definitions.

- Required Properties in the “`codeinsight_scan`” Section of “`gitlab.ci.yml`”
- Required Variable Definitions in the “`variables`” Section of “`gitlab.ci.yml`”

Required Properties in the “`codeinsight_scan`” Section of “`gitlab.ci.yml`”

This table describes the required properties in the `codeinsight_scan` section of the `gitlab.ci.yml` file.

Table 2-2 ■ Required Properties in the “`codeinsight_scan`” Section of “`gitlab.ci.yml`”

Property Section/Keyword		Description
tags	- <code><tag_for_your_GitLab-Runner></code>	Replace <code><tag_for_your_GitLab-Runner></code> with the tag defined for your GitLab Runner. For example: - <code>hello</code>
script	Define the properties used in the script commands that run the Code Insight plugin scan.	
	-cmd (first)	Replace <code><path/to/generic-plugin-binary/></code> with the location of the generic scan-agent plugin in GitLab. For example: -cmd <code>D:\GitLab-Runner\generic-plugin-binary</code>
	-cmd (second)	Define the properties required in the second command of the script. -jar <code><path/to/generic-plugin-binary/codeinsight-generic-<version>>.jar</code> For the <code>jar</code> property, replace <code><path/to/generic-plugin-binary/codeinsight-generic-<version></code> with the path and version of the <code>codeinsight-generic-<version>.jar</code> file. For example: -jar <code>D:\GitLab-Runner\generic-plugin-binary\codeinsight-generic-3.1.12.jar</code>

Table 2-2 ■ Required Properties in the “codeinsight_scan” Section of “gitlab.ci.yml” (cont.)

Property Section/Keyword		Description
script (continued)	-cmd (second) (continued)	<p>-root "<codebase_path>"</p> <p>For the root property, replace <codebase_path> with the root path for the codebase to be scanned. For example:</p> <p>-root D:\GitLab-Runner\output</p>
	-scandirs "<codebase_folder_paths>"	<p>For the scandirs property, replace <codebase_folder_paths> with the path of the specific folder to be scanned. Separate multiple folder paths with commas.</p>
	-alias <alias>	<p>For the alias property, replace <alias> with a user-defined name for the scan-agent plugin. For example:</p> <p>-alias Eportal_Remote on abc.com</p> <p>The alias is simply used to represent the “container” (scan root) under which all the scanned files will be listed in the API output and in the file tree in the Analysis Workbench. The name must be unique within the Code Insight project.</p>

Required Variable Definitions in the “variables” Section of “gitlab.ci.yml”

This table describes the variable values that must be defined in the **variables** section of the gitlab.ci.yml file. These values are passed to the script and to the pipeline-schedule configuration required for the scan.

Table 2-3 ■ Variable Definitions Required in the “variables” Section of “gitlab.ci.yml”

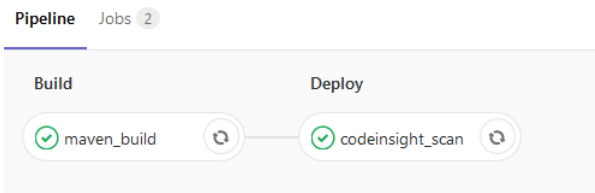
Variable Definition	Description
CODEINSIGHT_SERVER: "<http://<codeinsight_host>:8888/codeinsight>"	<p>Replace <http://<codeinsight_host>:8888/codeinsight> with the URL for the Code Insight Core Server, such as:</p> <p>CODEINSIGHT_SERVER: "http://1.1.1.1:8888/codeinsight"</p>
AUTH_TOKEN: "<token>"	<p>Replace <token> with the JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. For example:</p> <p>AUTH_TOKEN: "eyJhbGciOiJIUzUxMiJ9..." (not showing the complete token)</p> <p>Do not precede the token with the term “Bearer” as it is already indicated in the script command.</p> <p>You can generate this token using the Code Insight Web UI and then copy and paste it in this field. For more information, see Providing an Authorization Token.</p>

Table 2-3 ■ Variable Definitions Required in the “variables” Section of “gitlab.ci.yml” (cont.)

Variable Definition	Description
CODEINSIGHT_PROJECT: <codeinsight_project_name>"	Replace <codeinsight_project_name> with the name of the project you created in Code Insight to store the scan results. For example: CODEINSIGHT_PROJECT: "gitlabProject2"

Executing the Build

The next time your build is executed, a Code Insight agent scan is performed at the end of the build process. If you have scheduled the Code Insight scan job after a Maven build, for example, you should see something like this in your GitLab pipeline:



Note ■ Note that the first time a scan is performed using the generic scan-agent plugin, a data snapshot is downloaded from the National Vulnerability Database (NVD) to generate an index of the latest security vulnerabilities.

Jenkins Plugin

Code Insight provides a Jenkins scan-agent plugin that enables automated scanning of the Jenkins workspace as part of the build process or Jenkins Pipeline process. The scan results are sent to Code Insight for inventory creation, review, and security alerting. Optionally, you can have Code Insight generate a report based on the scan results and send it back to Jenkins for review.

The Jenkins scan-agent plugin installation and configuration process proceeds in the following manner:

- Phase 1**—Address the prerequisites for the Jenkins scan-agent plugin. See [Prerequisites for the Jenkins Scan-Agent Plugin](#).
- Phase 2**—Set the heap size. See [Setting Heap Size for the Jenkins Scan-Agent Plugin](#).
- Phase 3**—Set up the Jenkins scan-agent plugin. See [Setting Up the Code Insight Jenkins Scan-Agent Plugin](#).
- Phase 4**—(Optional) Configure the publication of a Code Insight report in Jenkins as part of the build scan. See [Configuring the Publication of a Code Insight Report in Jenkins](#).

For examples on how to include the Code Insight scan as a part of a Jenkins Pipeline, see [Support for the Jenkins Pipeline](#).

Prerequisites for the Jenkins Scan-Agent Plugin

Before you install and configure the Jenkins scan-agent plugin, ensure that the following prerequisites are met:

- The Jenkins scan-agent plugin is installed and configured properly in your environment.
- The Code Insight tasks required for running scans with the plugin are complete, as described in [Preparing to Use the Plugins](#).
- If you want to view the results of the build scan in a Code Insight report made available in Jenkins, the Jenkins HTML Publisher must be installed and configured in the build environment. For instructions on downloading and installing the HTML Publisher, go this site:

<https://plugins.jenkins.io/htmlpublisher/>

The following product versions are required for the generation and publication of a Code Insight report as part of the scan build process:

- Code Insight 2021 R1 or later
- Jenkins Server 2.332.1 or later
- Jenkins HTML Publisher 1.25 or later

See [Configuring the Publication of a Code Insight Report in Jenkins](#) for details about configuring the HTML Publisher for the report generation feature.

Support for Jenkins Systems Running on Java 11

Depending on its version, a Jenkins system requires either Java 11 (Oracle or OpenJDK 11) or Java 8. Jenkins versions 2.357 or later (and Long Term Support versions 2.361.1 or later) require Java 11, while Jenkins versions prior to 2.357 require Java 8.

The Jenkins scan-agent plugin available with Code Insight 2023 R1 or later supports Jenkins systems that run on either Java 11 or Java 8. (The plugin available with previous Code Insight versions supports only Jenkins systems that run on Java 8.)



Note ▪ The Code Insight Core and Scan Servers continue to run on Java 8 only.

Setting Heap Size for the Jenkins Scan-Agent Plugin

The Jenkins scan-agent plugin requires a minimum of 4GB heap for scanning. The heap size may need to be adjusted based on the number of parallel scans to be executed. In addition, ensure that you are using a 64-bit Java virtual machine (JVM) to support that amount of heap space and that you run the scan agent as a Jenkins agent, which is a Java executable that usually runs on a remote machine. The procedure for setting the heap size differs depending upon the environment you are using, Windows or Linux. Follow the procedure for your environment.

On Windows

Use this procedure to set the heap size in a Windows environment.



Task *To set the heap size in Windows, do the following:*

1. Open the `jenkins.xml` configuration file.
2. Update the `<EXECUTABLE>` value to point to your 64-bit JVM:

```
<EXECUTABLE>C:\Java\jdk1.8\jre\bin\java</EXECUTABLE>
```

3. Update the JVM arguments (`-Xmx` value) to allocate a minimum heap size of 4GB:

```
<ARGUMENTS>-Xrs -Xmx4g -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle -jar  
"%BASE%\jenkins.war" --httpPort=8080 --webroot="%BASE%\war"</ARGUMENTS>
```



Note ▪ The heap size may have to be adjusted based on the number of parallel scans to be executed.

On Linux

Use this procedure to set the heap size in a Linux environment.



Task *To set the heap size in Linux, do the following:*

1. Open the `/etc/default/jenkins` file.
2. Update the JVM arguments to allocate a minimum heap size of 4GB:

```
JAVA_ARGS="-Xmx4096m"
```



Note ▪ The heap size might need to be adjusted based on the number of parallel scans to be executed.

Setting Up the Code Insight Jenkins Scan-Agent Plugin

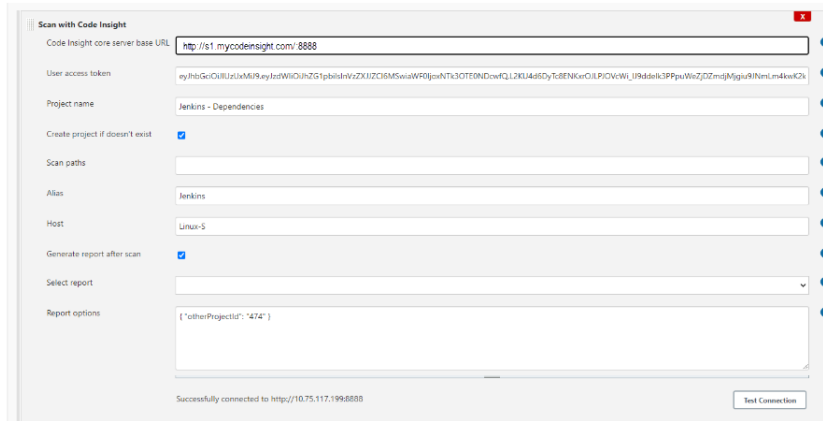
The following procedure describes how to configure the Jenkins scan-agent plugin.



Task *To set up the Jenkins scan-agent plugin, do the following:*

1. Extract the Jenkins scan-agent plugin from the `CodeInsightversionPlugins.zip` file. For more information, see [Downloading Plugins](#).
2. Access your Jenkins server instance and navigate to **Manage Jenkins -> Manage Plugins -> Advanced tab -> Upload Plugin**.
3. Browse to the `code-insight-scan-plugin.hpi` file, and click **Upload**.
4. Restart the Jenkins server after installing the plugin.
5. Open an existing Jenkins project, or use these steps to create a new Jenkins project:
 - a. Click **New Item**.

- b. Enter a name.
 - c. Select a project type.
 - d. Click **OK**.
6. To configure the project, select **Add post-build action** from the **Post-build action** menu, and select **Scan with Code Insight**. The **Scan with Code Insight** dialog is displayed.



7. Enter the following information in the **Scan with Code Insight** dialog:
- **Code Insight core server base URL**—Enter the URL for the Code Insight core server (for example, <http://codeInsightServer.myorg.org:8888>).
 - **User access token**—Enter the JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it in this field. For more information, see [Providing an Authorization Token](#).
 - **Project name**—Enter the name of the Code Insight project to which to send the scan results (for example, `Jenkins - Dependencies`). This project can be one that already exists on the Code Insight core server or one that the configuration process will create on the core server for you (if you have also selected **Create project if doesn't exist**).
 - **Create project if doesn't exist**—(Optional) Select this option to create the Code Insight project (specified for **Project name**) as part of the configuration process if it does not already exist on the Code Insight core server. If the project already exists, this option is ignored.
 - **Scan paths**—(Optional) To scan a subset of the Jenkins workspace, enter the path for each folder to scan, using commas to separate the paths. Each path should be either a path relative to the workspace or an absolute path.

If this field is left blank, the entire workspace considered for the scan.
 - **Alias**—A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the file tree in the **Analysis Workbench** and in the API output. This name must be unique within the project.
 - **Host**—(Optional) A user-defined name for the instance where the scan-agent plugin is configured to run scans. This property along with the **Alias** property will remain unchanged for each subsequent rescan.

Although generally optional, this value is required if you are running the scan in a dynamic host environment. See [Note About Rescans Performed by v2.0 and Later Plugins](#).

8. (Optional) Complete these next fields only if the Jenkins HTML Publisher is installed *and* you intend to have a report generated in Code Insight and made available in Jenkins as part of the build scan process.

For information about the Jenkins HTML Publisher, see [Prerequisites for the Jenkins Scan-Agent Plugin](#). In addition to the following fields, you must also define parameters for the publication of the report in Jenkins (see [Configuring the Publication of the Code Insight Report in Jenkins](#)).

- **Generate report after scan**—Select this option to have a Code Insight report generated as part of the Jenkins build process once the scan completes. The report, based on the scan results, is generated in Code Insight and returned to Jenkins so that you can view it.

At this point, you must click **Test Connection** to test your connection to the Code Insight server. If the connection succeeds, the **Select report** dropdown list is populated with available reports.

If the connection fails (or **Generate report after scan** is not selected), **Select report** remains blank and you are unable to continue with the report generation configuration.

- **Select report**—Select the specific Code Insight report that you want to generate as part of the Jenkins scan build process once the scan completes. The dropdown list includes all standard and custom reports available on your Code Insight system.
- **Report options**—If the report you selected is a custom report that enables users to specify additional report parameters in JSON format, enter those parameters here. For example, if the selected report is defined to require another project (that is, the **enableProjectPicker** value in the report definition is **true**), then enter the following required parameter in this field:

```
{ "otherProjectId": "1" }
```

where `otherProjectId` is the ID of the second project.

9. If you have not configured report generation (see the previous step), click **Test Connection** to test your connection to Code Insight.
10. Click **Save**. The next time you build, the scan will be performed after the build action. (Before running a scan, ensure that you have met all the requirements in [Prerequisites for the Jenkins Scan-Agent Plugin](#) and [Setting Heap Size for the Jenkins Scan-Agent Plugin](#).)

Configuring the Publication of a Code Insight Report in Jenkins

Optionally, the Jenkins scan-agent plugin can be configured to generate a Code Insight report of your choice as part of the Jenkins build scan. After the plugin sends the scan results to Code Insight, the specified report is generated in Code Insight and sent back to Jenkins, where you can then view it as part of the build process. Thus, the report provides a means to view the results of the build scan within Jenkins itself without having to go to Code Insight to see the results. (The report is also available in Code Insight on the **Reports** tab for the project.)

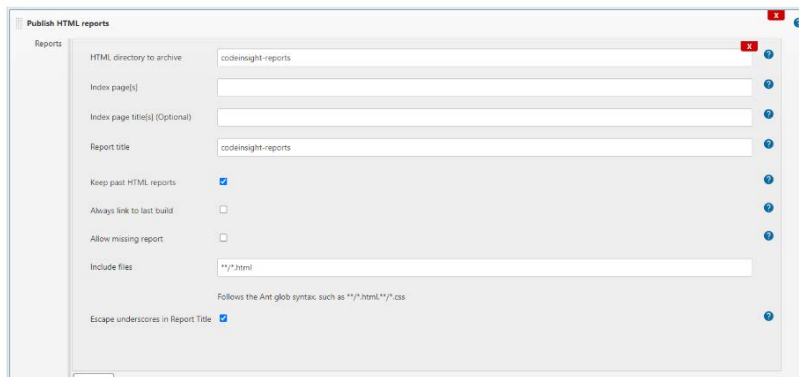
Viewing the Code Insight report as part of the build scan requires that the Jenkins HTML Publisher extension be installed in the Jenkins build environment. See [Prerequisites for the Jenkins Scan-Agent Plugin](#).

To configure the report generation as part of the build scan, you must do the following:

- Define the report settings for the Code Insight Jenkins scan-agent plugin (see [Setting Up the Code Insight Jenkins Scan-Agent Plugin](#)).
- Configure the publication of the report in Jenkins. See the following section for instructions.

Configuring the Publication of the Code Insight Report in Jenkins

To configure the publication of the Code Insight report once it is sent to Jenkins post-build, go the **Post-build Actions** section of the build scan job and open the **Build HTML reports** page.



Define the following required parameters:

- **HTML directory to archive**—Enter `codeinsight-reports`.
- **Report title**—Enter any meaningful title for the report (for example, `Code Insight Report`).

Click **Publishing options...** to display these required parameters:

- **Include file**—Enter the file extension for the report in the format `**/*.<file-extension>`, as in the example `**/*.html`. This option is required to keep past HTML reports.
- **Keep past HTML reports**—Select this option to view the report as part of each build.

Support for the Jenkins Pipeline

The Code Insight plugin for Jenkins supports the inclusion of the Code Insight scan in a Jenkins Pipeline, as described in the following topics:

- [Providing the Pipeline Script for the Scan Step](#)
- [Pipeline Code Examples for Running the Scan](#)

See the previous section for a description of the properties used in the Pipeline commands.

Providing the Pipeline Script for the Scan Step

Once you build the Pipeline job, you need to include the Pipeline script for the scan step, `StartScan`, in your Pipeline code. (The next section, [Pipeline Code Examples for Running the Scan](#), provides examples of Pipeline code that include this script.)

To create the Pipeline script for the `StartScan` step, you can use one of these methods:

- Go to the Snippet Generator, select the **StartScan: Scan workspace and send results to FlexNet Code Insight** step, and generate the script. Then copy and paste the generated script into the Pipeline code.
- Simply create the script for the `StartScan` step as highlighted in the Pipeline code examples.

See [Setting Up the Code Insight Jenkins Scan-Agent Plugin](#) for a description of the properties (base URL, project name, and JWT) used in the Pipeline script.

Pipeline Code Examples for Running the Scan

Jenkins supports two syntax types for the development of Pipeline code:

- **Scripted syntax**—The “traditional” syntax used to develop the Pipeline as a script using Groovy as the domain-specific language.
- **Declarative syntax**—A simple, user-friendly syntax with a predefined hierarchy of statements that makes Pipeline development easier than with the Scripted syntax. Additionally, it does not require knowledge of the Groovy language. Jenkins support for the Declarative syntax was introduced with Jenkins Pipeline Plugin 2.5.

The following examples show both types of Pipeline code syntax in which the Pipeline script for the scan has been incorporated:

- [Example Declarative Pipeline Code to Run the Scan](#)
- [Example Scripted Pipeline Code to Run the Scan](#)

The Pipeline script for the scan step is highlighted in each example.

Example Declarative Pipeline Code to Run the Scan

The following is an example of Declarative code used to run the Code Insight scan as a StartScan step in the Pipeline process:

```
pipeline {
  agent any
  stages {
    stage('Checkout build and scan project1') {
      steps {
        git credentialsId: 'abcd', url: 'git://git.company.com/organization/repository1.git'
        sh "'PATH_TO_MAVEN/bin/mvn' clean install"

        StartScan (baseUrl: '<http://HOST_NAME:PORT/>', projectName: '<CODEINSIGHT_PROJECT_NAME>', alias:
        '<SCAN-AGENT_ALIAS>', host: '<SCAN-AGENT_HOST>', token: '<JWT_TOKEN>')
      }
    }
  }
}
```

Example Scripted Pipeline Code to Run the Scan

The following is an example of Scripted Pipeline code used to run the Code Insight scan as a StartScan step in the Pipeline process. The example also shows how to set up individual scans within a single Pipeline job by specifying multiple directories.

```
node {
  checkout1()
  checkout2()
}

def checkout1(){
```

```
dir("project-1"){
    stage ('Checkout project 1'){
        git credentialsId: 'abcd', url: 'git://git.company.com/organization/repository1.git'
    }
    stage ('Build Project 1'){
        build()
    }
    stage ('Scan Project 1'){
        StartScan (baseUrl: '<http://HOST_NAME:PORT/>', projectName:
            '<CODEINSIGHT_PROJECT_NAME>', alias: '<SCAN-AGENT_ALIAS>', host:
            '<SCAN-AGENT_HOST>', token: '<JWT_TOKEN>')
    }
}

def checkout2(){
    dir("project-2"){
        stage ('Checkout project 2'){
            git credentialsId: 'abcd', url: 'git://git.company.com/organization/repository2.git'
        }
        stage ('Build Project 2'){
            build()
        }
        stage ('Scan Project 2'){
            StartScan (baseUrl: '<http://HOST_NAME:PORT/>', projectName:
                '<CODEINSIGHT_PROJECT_NAME>',
                alias: '<SCAN-AGENT_ALIAS>', host: '<SCAN-AGENT_HOST>', token:
                '<JWT_TOKEN>')
        }
    }
}

def build(){
    sh "'PATH_TO_MAVEN/bin/mvn' clean install"
}
```

Scan Scheduler Plugin for Jenkins

The Jenkins Scan Scheduler plugin enables you to schedule the scan of a codebase residing on the Code Insight scan server via the Jenkins scheduler. Before you install and configure the Jenkins Scan Scheduler plugin, ensure that the following prerequisites are met:

- Jenkins must be installed and configured properly in your environment.
- The project of interest must be set up in Code Insight. For information on creating a Code Insight project, see “Creating a Project” in the *Code Insight User Guide*.



Task

To install the Code Insight Scan Scheduler for Jenkins, do the following:

1. Sign into Jenkins CI.
2. Navigate to **Manage Jenkins > Manage Plugins > Advanced**. The **Upload Plugin** dialog appears.
3. Click **Choose File** and select the `code-insight-scan-scheduler.hpi` file.

4. Click **Upload**.
5. Restart the Jenkins server after uploading the plugin.
6. Create a new Jenkins project:
 - Click **New Item**.
 - Enter a name.
 - Select a project type.
 - Click **OK**.
7. To configure the project, select **Add build step** from the **Build** menu, and then select **Schedule a Code Insight Scan**. The **Schedule a Code Insight Scan** dialog is displayed.
8. Enter the following information in the **Schedule a Code Insight Scan** dialog:
 - **Server URL**—The URL for the Code Insight Core server. For example, `http://codeInsightServer.myorg.org:8888/codeinsight/`.
 - **Token**—The JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it in this field. For more information, see [Providing an Authorization Token](#).
 - **Project ID**—The ID of the project that was created in the Code Insight Web UI.
9. Click **Test Connection** to test your connection to Code Insight.
10. Click **Save**. The next time you build, the scan will be scheduled on the Code Insight server for the configured project as part of the build.

TeamCity Plugin

This section explains how to configure TeamCity to integrate with the Code Insight's generic scan-agent plugin to perform an automatic composition scan as part of the build process. The following topics are covered. Note that the scan requires the generic scan-agent plugin.

- [Prerequisites for TeamCity Plugin](#)
- [Installing the Generic Scan-Agent on the Team City Build Agent](#)
- [Configuring a Build to Run a Code Insight Scan](#)
- [Executing the Build](#)

Prerequisites for TeamCity Plugin

The following prerequisites are required to integrate TeamCity with the Code Insight generic scan-agent plugin:

- A TeamCity build agent properly installed. (Refer to <https://confluence.jetbrains.com/display/TCD10//Setting+up+and+Running+Additional+Build+Agents> for instructions.)
- All the prerequisites listed in [Prerequisites for the Generic Scan-Agent Plugin](#).

- The generic scan-agent plugin (as described in [Installing the Generic Scan-Agent on the Team City Build Agent](#)).

Installing the Generic Scan-Agent on the Team City Build Agent

Use these instructions to install the generic scan-agent plugin on the Team City build agent.



Task

To install the generic scan-agent plugin on the Team City build agent, do the following:

1. Download and extract the contents of the `CodeInsightversionPlugins.zip` file, as described in the previous section, [Downloading Plugins](#).
2. Locate the `code-insight-agent-sdk-generic-plugin/generic-plugin-binary` folder, and copy it to the TeamCity build agent.
3. On the Team City build agent, update the following to match your environment:

- The codebase root:

```
SET ROOT_PATH=C:\Codebase\output
```

- The bin folder location for the generic scan-agent plugin:

```
cd C:\agent\GenericScanPlugin\example\bin
```

Configuring a Build to Run a Code Insight Scan

Follow these steps to configure a build that runs a Code Insight scan.



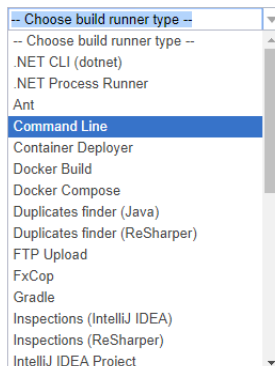
Task

To configure a build to run a Code Insight scan, do the following:

1. Log into TeamCity, select your project, and create a new Build Configuration.
2. To configure a build step to run a Code Insight scan, select on your build configuration, and click **Add Build Step**.
3. From the **Runner type** list, select **Command Line**.

New Build Step: | ▾

Runner type:



4. Configure the **Command line** build step for the Code Insight scan:
- a. Enter a value for **Step name** (for example, **Codeinsight Scan**) to identify the step.
 - b. In the **Run** field, select **Custom script**.
 - c. In the **Custom script** field, provide the following:

```
C:\GenericScanPlugin\example\bin\TeamCity_FNCIScan.bat <CODEINSIGHT_PROJECT_NAME>  
<CODEINSIGHT_SERVER> <JWT_TOKEN> <SCAN_DIR>
```

Replace the following variables in the script with the appropriate information:

- **<CODEINSIGHT_PROJECT>**—The name of the project you created in Code Insight to capture the inventory.
- **<CODEINSIGHT_SERVER>**—The URL of the Code Insight Core Server (for example, **http://1.1.1.1:8888/codeinsight**).
- **<JWT_TOKEN>**—Your JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it here. For more information, see [Providing an Authorization Token](#).
- **<SCAN_DIR>**—The directory that you want to scan.

Build Step

Runner type:

Command Line

Simple command execution

Step name:

Code Insight Scan

Optional, specify to distinguish this build step from other steps.

Run:

Custom script

Custom script:

Enter build script content:

C:\GenericScanPlugin\example\bin\TeamCity_FNCIScan.bat

A platform-specific script, which will be executed as a .cmd file on Windows or as a shell script in Unix-like environments.

Show advanced options

Save

Cancel

When complete, your build configuration should look like this:

Build Steps

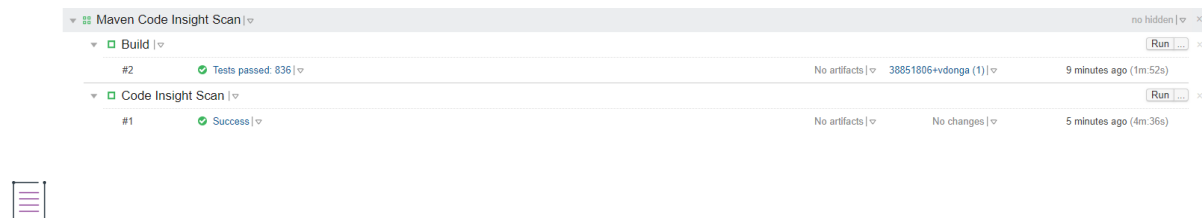
In this section you can configure the sequence of build steps to be executed. Each build step is represented by a build runner and provides integration with a specific build or test tool.

+ Add build step

Build Step	Parameters Description		
1. Code Insight Scan	Command Line Custom script: C:\GenericScanPlugin\example\bin\TeamCit... Execute: If all previous steps finished successfully	Edit	

Executing the Build

The next time your build is executed, a scan will be performed at the end of the build process. If you have scheduled the Code Insight scan job, after a Maven build, for example, you should see something like this in your TeamCity build queue:



Note - Note that the first time a scan is performed using the generic scan-agent plugin, a data snapshot is downloaded from the National Vulnerability Database (NVD) to generate an index of the latest security vulnerabilities.

Configuring SSL for Azure DevOps, Bamboo, and Jenkins Scan Agents

To configure SSL for secure communications between a Code Insight scan agent and the Code Insight Core Server, the SSL Certificate used by the Code Insight Core Server must be exported to a file and then imported to the Java home location of the scan agent. The following sections describe this process:

- [Exporting the Certificate Used by Code Insight](#)
- [Importing the Certificate to the Java Home for the Scan Agent](#)

Currently, these instructions apply to configuring SSL for the Azure DevOps, Bamboo, and Jenkins scan-agent plugins only. The plugin must be installed to configure SSL.

Exporting the Certificate Used by Code Insight

The following sections describe how to use the Google Chrome or Mozilla Firefox browser to export the SSL certificate used by the Code Insight Core Server.

- [Using Chrome](#)
- [Using Firefox](#)

To use another browser type to export the certificate, follow the instructions specific to that browser.


Using Chrome

The following steps describe how to use a Chrome browser to export the Code Insight SSL certificate.



Task

To export the certificate using a Chrome browser, do the following:

1. Access the Code Insight Web UI in a Chrome browser.
2. Click the padlock icon  next to the website URL.
3. From the pop-up window, select **Certificate**.
4. On the **Certificate** window, click the **Details** tab.
5. On the **Details** tab, select **Copy to File....**
6. Follow the wizard instructions to export the certificate to a file. (The wizard allows you to export the certificate contents in the format you require—DER, Base 64, or Cryptographic Message Syntax Standard.)
7. At the end of the wizard process, save the file to any location. If you save the file to a location not accessible by the machine on which the Code Insight scan agent resides, copy the file to such a location before beginning the import phase.

Using Firefox



Use the following steps to export the certificate in a Mozilla Firefox browser.

This browser exports the certificate as .cer content only. If you need to export the certificate contents to a DER format, perform the export process using a browser that supports this format.



Task

To export the certificate in a Firefox browser, do the following:

1. Access the Code Insight Web UI in a Firefox browser.
2. In the upper-right corner of the browser, click  to open the Firefox hamburger menu, and select **Options**.
3. In the **Find in Options** search field, enter **cer**, and click .
4. Scroll through the search results to locate the **Certificates** section, and click **View Certificates**.
5. On the **Certificate Manager** window, click **Servers** to list the available certificates.
6. Select the Code Insight Core Server certificate, and click **Export**.
7. Click **Save** to export the certificate to a file in any location. If you save the file to a location not accessible by the machine on which the Code Insight scan agent plugin resides, copy the file to such a location before beginning the import phase.
8. Click OK to close the **Certificate Manager** window.

Importing the Certificate to the Java Home for the Scan Agent

The following steps describe how to import the exported certificate to the Java home directory for the Code Insight scan agent.



Task

To import the certificate for the Code Insight agent, do the following:

1. Ensure that the exported certificate is in a location accessible by the machine where the Code Insight scan agent plugin is installed.
2. Verify the Java home location of the scan agent. (If necessary, use the instructions specific to your CI tool to determine the path.)
3. From the command line, navigate to the scan agent's Java home location. (On a Linux machine, enter `echo $JAVA_HOME`.)
4. Go to the bin directory of the scan agent's Java home directory, and run the following command to import the Code Insight certificate to the cacerts file.

```
keytool -import -file <path_to_certificate/<codeinsight_certificate_name>.cer> -alias <ALIAS> -  
keystore ../lib/security/cacerts -storepass
```

5. Restart the agent plugin.

Plugins for Package Managers and Build Tools

Code Insight provides the following scan-agent plugins that integrate with package manager and build tools:

- [Apache Ant Plugin](#)
- [Gradle Plugin](#)
- [Maven Plugin](#)

Apache Ant Plugin

Apache Ant is a tool to support the build process for Java projects. Ant is often used in conjunction with other build tools such as Maven. Code Insight provides the Apache Ant plugin to run a scan task along with the target of the build cycle and send the results, as inventory, to the Code Insight server for review, management, and remediation. The Apache Ant plugin scans only the application root folder.

The following topics describe how to install and configure the Gradle plugin:

- [Prerequisites for the Apache Ant Plugin](#)
- [Configuring the Apache Ant Plugin](#)
- [Executing the Scan](#)

Prerequisites for the Apache Ant Plugin

Before you install and configure the Apache Ant plugin, ensure that the following items are correctly installed and configured:

- JDK 1.8
- Apache Ant

Also ensure that all Code Insight prerequisite tasks for plugins have been performed, as described in [Preparing to Use the Plugins](#).

Configuring the Apache Ant Plugin

Use the following procedure to configure the Apache Ant Plugin to execute along with the build target.



Task

To configure the plugin, do the following:

1. Extract the Ant plugin from the `CodeInsightversionPlugins.zip` file. See [Downloading Plugins](#).
2. Configure `%ANT_HOME%` and add `%ANT_HOME%/bin` to the path variable.
3. To check the Ant plugin installation, run the following command:

```
ant -v
```

4. Add all the dependent jars from the `code-insight-ant-plugin` folder to the application's compile classpath.
5. To run the task `codeinsightantplugin` along with the compile target, paste the taskdef code snippet into the compile target and run the following command:

```
ant compile
```

For the code snippet, see [Executing the Scan](#).

6. Copy the `code-insight-ant.jar` into the path used for the compile task, and set the `classpath refid` of the `javac` task as the `classpathref` in the `codeinsightantplugin` task.

Executing the Scan

Use the following procedure to execute the scan along with the build target.



Task

To execute the scan, do the following:

1. Run the following command:

```
ant <targetname>
```

For example, you might enter:

```
ant compile
```

2. To execute the scan along with any target of the build lifecycle, apply the plugin inside the target in the `build.xml` of the Ant application as follows:

```
<taskdef name="codeinsightantplugin" classname="com.ant.plugin.CodeInsightAntPlugIn"
classpath=" " classpathref=" " />
<codeinsightantplugin fnciServer="<SERVER_URL>" fnciauthtoken="<BEARER_SERVER_TOKEN_VALUE>"
fnciprojectname="<CODE_INSIGHT_PROJECT_NAME>"
scanDirs="<DIRECTORIES_TO_BE_SCANNED_IN_RELATION_TO_BASE_APPLICATION_PROJECT>"
alias="<SCAN-AGENT_ALIAS>"
pluginRootPath="<PLUGIN_ROOT_PATH>"
pluginProjectName="<APPLICATION_PROJECT_TO_SCAN>"
pluginindescription="<APPLICATION_DESCRIPTION>"
pluginPathPrefix="<PLUGIN_PATH_PREFIX>">
</codeinsightantplugin>
```

See descriptions of these settings in [Installing and Configuring the Gradle Plugin](#).



Note ■ The Ant plugin project name can not include the ampersand (&) character.

The following is a description of the scan settings used to apply the plugin:

- **fnciServer**—(Required) The hosted server where the Code Insight application is running.
- **fnciAuthToken**—(Required) The JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it here. Be sure to include the command “Bearer” followed by the token value, as in the example:

```
Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pb2IsInVzZXJJZCI6MSwia
```

For more information about generating this token, see [Providing an Authorization Token](#).

- **fnciProjectName**—(Required) The name of the Code Insight project existing on the Code Insight server to contain the scan results.
- **scanDirs**—Each path to be scanned relative to the base directory of the Ant project. For example, if the base directory for the Ant project is `D:/worksapce/project` and you want to scan the directory `D:/worksapce/project/build`, specify `"/build"` for the value here. If multiple paths are to be scanned, separate them with commas: `"/build,/build2"`. To indicate that all paths under the base directory are to be scanned, enter `."` for the value.
- **alias**—A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the **Analysis Workbench**. This name must be unique within the project.
- **pluginRootPath**—(Required) The path where the plugin will be launched, usually the root of the application. An example value is `D:\\test\\Ant_test\\Ant_application`. This field is required.
- **pluginProjectName**—(Required) The name of Ant-based application whose codebase you want to scan.
- **pluginDescription**—A description of the application to display on the **Summary** tab for the project in Code Insight.
- **pluginPathPrefix**—The Code Insight server path (for example, `demo_workspace/`) used as a prefix for codebase file locations, as listed on the **Associated Files** tab for an inventory item in the Code Insight user interface. For example, `demo_workspace/`. This field is optional.

Note About “classpath”

Although specifying `taskdef.classpath` is not mandatory, you should set the path id of the `javac` task as the `Classpathref` in the `codeinsightantplugin` taskdef. If the application does not have a `javac` path-id defined in the `build.xml`, you must define one new path id referring to all compile time dependencies and use this as `Classpathref`. See the following example:

```
<path id= "cp" <fileset dir="lib">
  <include name="*.jar" />
</fileset>
</path>
```

In this case, use `"cp"` as the `Classpathref` in the taskdef.

Gradle Plugin

Gradle is a build automation system that uses the Groovy language to establish the configuration of the build project, rather than using XML as Maven does. The Gradle plugin provided by Code Insight scans a codebase created in Gradle and sends the results, as inventory, to the Code Insight server for review, management, and remediation through the user interface. The plugin scans only the following:

- Direct dependencies of a project
- Transitive dependencies of a project
- The distribution folder containing application jars



Note • The Gradle plugin does not scan the jars present in the `lib` folder, which contains plugin-dependent jars.

The following topics describe how to install and configure the Gradle plugin:

- [Prerequisites for the Gradle Plugin](#)
- [Installing and Configuring the Gradle Plugin](#)

Prerequisites for the Gradle Plugin

Before you install and configure the Code Insight Gradle plugin, ensure that the following items are correctly installed and configured:

- JDK1.8
- Gradle

Also ensure that all Code Insight prerequisite tasks for plugins have been performed, as described in [Preparing to Use the Plugins](#).

Installing and Configuring the Gradle Plugin

To use the Gradle plugin, you must configure settings in the application’s `build.gradle`. This section contains the procedure for installing and configuring the plugin.



Task

To install and configure the Gradle plugin, do the following:

1. Extract the Gradle plugin from the `CodeInsightversionPlugins.zip` file. See [Downloading Plugins](#).
2. Use these steps to add all the dependent jars in the `code-insight-scan-plugin` to the application class path:
 - a. Create a folder named `dependent_jars` within the application project.
 - b. Copy all jar files into that folder.
 - c. Add the following configuration in `build.gradle` so that the jars are available to the classpath:

```
buildscript {  
    dependencies {  
        classpath files(fileTree(dir: 'dependent_jars', includes: ['*.jar']))  
    }  
}
```

3. If the Java plugin is not already applied in the `build.gradle` script, do so by adding the appropriate configuration *at the beginning* of the script:

- For a single module project, add the following:

```
apply plugin: 'java'
```

- For a multi-modular project:

```
allprojects {  
    apply plugin: 'java'  
}
```

4. Apply the Gradle plugin in the `build.gradle` file:

```
apply plugin: 'code-insight-scan-plugin'  
  
scanSettings {  
    fnciServer= "<SERVER_URL>"  
    fnciAuthToken= "<BEARER_SERVER_TOKEN_VALUE>"  
    fnciProjectName= "<CODE_INSIGHT_PROJECT_NAME>"  
    alias=<SCAN-AGENT_ALIAS>  
    pluginRootPath= "<PLUGIN_ROOT_PATH>"  
    pluginProjectName= "<APPLICATION_PROJECT_TO_SCAN>"  
    pluginDescription= "<APPLICATION_DESCRIPTION>"  
    pluginPathPrefix= "<PLUGIN_PATH_PREFIX>"  
}
```

The following is a description of the scan settings used to apply the plugin:

- **scanSettings**—An extension to provide the Code Insight scan server settings.
- **fnciServer**—(Required) The hosted server where the Code Insight application is running.
- **fnciAuthToken**—(Required) The JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it here. Be sure to include the command “Bearer” followed by the token value, as in the example:

```
Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pb2IsInVzZXJJZCI6MSwia
```

For more information about generating this token, see [Providing an Authorization Token](#).

- **fnciProjectName**—(Required) The name of the Code Insight project existing on the Code Insight server to contains the scan results.
 - **alias**—A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the **Analysis Workbench**. This name must be unique within the project.
 - **pluginRootPath**—(Required) The path where the plugin will be launched, usually the root of the application. An example value is `D:\\test\\Gradle_test\\Gradle_application`. This field is required.
 - **pluginProjectName**—(Required) The name of Gradle-based application whose codebase you want to scan.
 - **pluginDescription**—A description of the application to display on the **Summary** tab for the project in Code Insight.
 - **pluginPathPrefix**—The Code Insight server path (for example, `demo_workspace/`) used as a prefix for codebase file locations, as listed on the **Associated Files** tab for an inventory item in the Code Insight user interface. For example, `demo_workspace/`. This field is optional.
5. Configure the `code-insight-scan` task to run during or after the build process. See [Important Note About Scanning Dependencies](#).

Important Note About Scanning Dependencies

Previous versions (1.x) of the Gradle scan-agent plugin scanned both the dependencies section *and* the project build directory of the Gradle project. The current plugin version (2.x), introduced in Code Insight 2020 R3, scans only the project build directory. Refer to the Gradle documentation for instructions on how to include dependencies as a part of build directory. An example install command for including dependencies might be:

```
task copyToLib(type: Copy) { into "$buildDir/output/lib" from configurations.runtime }
```

For this task, use the following command to run the scan agent from the Gradle application project:

```
gradle build copyToLib code-insight-scan
```

Maven Plugin

Maven is a tool that simplifies the building and management of Java-based projects. The Code Insight Maven plugin allows you to scan an application project during its build on Maven without disrupting the established build process. Once scanned, the codebase can be analyzed in the Code Insight user interface. The Maven plugin makes it easy to incorporate scanning and analysis into your development workflow.

For more information, refer to the following:

- [More About the Maven Plugin](#)
- [Prerequisites for the Maven Plugin](#)
- [Installing and Configuring the Maven Plugin](#)
- [Cleaning the Application Project](#)
- [Running the Maven Goal for the Scan](#)

More About the Maven Plugin

The Maven plugin scans only the following items:

- Direct dependencies of a project (see also [Important Note About Scanning Dependencies](#))
- Transitive dependencies of a project (see also [Important Note About Scanning Dependencies](#))
- Build folder containing the application jars

The plugin creates a Maven goal called `code-insight-scan`, which will be executed along with the *install* phase of the build cycle to get inventory details, as described later in [Running the Maven Goal for the Scan](#).

Prerequisites for the Maven Plugin

Before you install and configure the Code Insight Maven plugin, ensure that the following prerequisites are met:

- Maven and JDK 1.8 are installed.
- `%MAVEN_HOME%/bin` is configured and added to the path environment variable. (This prerequisite avoids SSL certification issues.) You can always check your Maven installation by running `mvn -v`.
- All Code Insight prerequisite tasks for plugins have been performed, as described in [Preparing to Use the Plugins](#).

Installing and Configuring the Maven Plugin

Use the following steps to install and configure the Code Insight Maven plugin.



Task

To install and configure the Code Insight Maven plugin, do the following:

1. From the `CodeInsightversionPlugins.zip` file that was downloaded from the Product and License Center, extract the Maven plugin subdirectory (`code-insight-maven-plugin`) to a location on your local disk. The recommended location to which to extract this subdirectory is the application project directory.
2. Execute the following commands to install the plugin into the Maven local repository:

```
mvn install:install-file -Dfile="$<PROJECT_DIRECTORY>/code-insight-maven-plugin/lib/code-insight-maven-scan-<PLUGIN_VERSION>.jar" -DpomFile="$<PROJECT_DIRECTORY>/code-insight-maven-plugin/lib/pom.xml" -DgroupId=com.flexnet.maven -DartifactId=code-insight-maven-scan -Dversion=<PLUGIN_VERSION> -Dpackaging=jar

mvn install:install-file -Dfile="$<PROJECT_DIRECTORY>/code-insight-maven-plugin/lib/codeinsight-agent-<AGENT_VERSION>.jar" -DgroupId=com.flexnet.codeinsight -DartifactId=codeinsight-agent -Dversion=<AGENT_VERSION> -Dpackaging=jar
```

Note the following variables:

- `<PROJECT_DIRECTORY>` is your application project directory (or the local directory to which you extracted the plugin).
- `<PLUGIN_VERSION>` is the latest version of the `code-insight-maven-scan` jar file.
- `<AGENT_VERSION>` is the latest version of the `codeinsight-agent` jar file.

3. Add the following information to your application pom.xml file. Refer to [Plugin and Code Insight Server Settings](#) for a description of the values you need to provide for the plugin and fnciServerSettings sections.

```
<plugin>
  <groupId>com.flexnet.maven</groupId>
  <artifactId>code-insight-maven-scan</artifactId>
  <version>latest_codeinsight_maven_scan_jar_version</version>
  <inherited>false</inherited>
  <executions>
    <execution>
      <phase>install</phase>
      <goals>
        <goal>code-insight-scan</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <fnciServerSettings>
      <fnciServer>server_url</fnciServer>
      <fnciAuthToken>Bearer server_authentication_token_value</fnciAuthToken>
      <fnciProjectName>codeinsight_project_name</fnciProjectName>
      <alias>scan_agent_alias<alias>
      <pluginRootPath>plugin_root_path</pluginRootPath>
      <pluginProjectName>plugin_project_name</pluginProjectName>
      <pluginDescription>any_plugin_description</pluginDescription>
      <pluginPathPrefix>plugin_path_prefix</pluginPathPrefix>
    </fnciServerSettings>
  </configuration>
</plugin>
```

Plugin and Code Insight Server Settings

The following describes the settings that you need to define in the plugin and fnciServerSettings sections of the information you are adding to the application pom.xml file (as described in Step 3 of the previous procedure).

Table 2-4 ■ Code Insight Server Settings in the Application “pom.xml” File

Setting	Description
version	The version of the code-insight-maven-scan-<VERSION>.jar file included with the current plugin (for example, 1.0.2).
fnciServer	(Required) The URL path to the Code Insight server in the following format: http://<CODE_INSIGHT_SERVER_HOST_NAME>:<PORT_NUMBER>/codeinsight/
fnciAuthToken	(Required) The JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it here. Be sure to include the command “Bearer” followed by the token value, as in the example: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsInVzZXJJZCI6MSwia For information about generating this token, see Providing an Authorization Token .

Table 2-4 ■ Code Insight Server Settings in the Application “pom.xml” File (cont.)

Setting	Description
fnciProjectName	(Required) The name of the Code Insight project created on the Code Insight server for your application codebase scans.
alias	A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the Analysis Workbench . This name must be unique within the project.
pluginRootPath	Currently not used.
pluginProjectName	(Optional) The name of the <i>application project</i> being scanned. This name will appear, along with the Code Insight project name, in the Last Scan field on the Summary tab for the project in the Code Insight user interface. It provides a reference to help a reviewer or developer identify what codebase was scanned.
pluginDescription	(Optional) A description of the application project being scanned. This text will appear in the Description field on the Summary tab for the project in the Code Insight user interface.
pluginPathPrefix	(Optional) The path prefix for the codebase files being scanned. This prefix is used to reference the codebase file paths on the Project report generated from the Summary tab for the project in the Code Insight user interface.

Important Note About Scanning Dependencies

Previous versions (1.x) of the Maven scan-agent plugin scanned both the dependencies section *and* the `${project.build.directory}` of the Maven project. The current plugin version (2.x), introduced in Code Insight 2020 R3, scans only the `${project.build.directory}`. Refer to the Maven documentation for instructions on how to include dependencies as a part of build directory. An example install command for including dependencies might be:

```
maven-dependency-plugin install copy-dependencies ${project.build.directory}/project-dependencies
```

Cleaning the Application Project

During a build, Maven can cache an extensive amount of output, which, in turn, can have a negative impact on the performance of the Maven plugin. Therefore, before you run the Maven goal for the Code Insight scan, it is recommended that you clean the application project, a process that clears the cache of the artifacts of previous builds.



Task

To clean the application project, do the following:

Execute the following command:

```
mvn clean
```

Running the Maven Goal for the Scan

After you clean the application project, you can run the `code-insight-scan` Maven goal, which will perform a Code Insight scan on the codebase.



Task

To execute the goal that runs the Code Insight scan, do the following:

To build the application (and run the Code Insight scan as part of the build cycle), execute the following command:

```
mvn install
```

Alternatively, to execute the Code Insight scan only, run the specific goal:

```
mvn code-insightscan:code-insight-scan
```

Plugins for Container Platforms

Currently, Code Insight support for scan integration with a container platforms includes the [Docker Images Plugin](#).

Docker Images Plugin

Docker is a tool that packages applications and their dependencies into containers, which are comprised of static images. These images are themselves comprised of layers. Code Insight Docker Images scan-agent enables the scanning of Docker images on a Docker server and sends the results as inventory to the Code Insight server for review, management, and remediation.



Note - It is recommended that Docker images be scanned on a development, test, or staging server before being pushed to a production instance as part of the DevOps process flow.

The following topics describe how to install and launch the Docker Images plugin:

- [Prerequisites for the Docker Images Plugin](#)
- [Syft Integration in the Docker Images Plugin](#)
- [Obtaining the Docker Images Plugin](#)
- [Running a Docker Images Plugin Scan](#)

Prerequisites for the Docker Images Plugin

Before you install and configure the Docker Images plugin, ensure that the following prerequisites are met:

- The Docker server must be installed and configured properly in your environment. The Docker Images plugin can only be executed on a server that already has an authenticated connection to the Docker server.



Note ▪ The Docker Images plugin issues Docker commands without prompting for credentials.

- All Code Insight prerequisite tasks for plugins have been performed, as described in [Preparing to Use the Plugins](#).
- A minimum of 2GB of heap space is allocated on the Docker server, which must be configured with a 64-bit JRE to support that amount of heap space.
- The plugin sets the maximum JVM heap size to 4GB by default, a size sufficient for most images. However, if you are receiving errors due to heap size, you can increase this maximum by setting a higher `-Xmx` value in the last line of code in the `code-insight-docker-plugin.sh` file, located in the plugin folder. (You can also add a minimum heap size, using the `-Xms` option, if needed.) The following provides an example of how you might update this line to increase the maximum heap size:

```
java -Xmx10240m -Xms1024m -jar $DEBUG code-insight-docker-plugin.jar $*
```



Note ▪ If you are using a 64-bit JVM, you can increase the heap size to the size you need. If you are using a 32-bit JVM, you are limited to 4GB.

- The Docker image you are scanning must already be downloaded to your system.

Syft Integration in the Docker Images Plugin

Syft, an open-source scanning tool, is integrated in the Docker Images plugin (starting with the 2022 R4 release). This tool generates a Software Bill of Materials (SBOM) from the discovery of packages and libraries in container images, file systems, archives, and other artifacts. Its integration with the Docker Images plugin enables the plugin to report findings from Docker images containing Alpine, RPM, and Debian Linux distribution packages that reside in RedHat Enterprise Linux, Ubuntu, and CentOS Linux operating systems.



Important ▪ Support for CentOS operating system currently has a known issue in Code Insight in which not all codebase files are being displayed in the **Analysis Workbench** after a scan.

The following sections provide more information about the integration of Syft in the Docker Images plugin:

- [Forge Support](#)
- [Important Information About the Syft Integration](#)

Forge Support

The following table shows the supported forges (by package type) for inventory detected by Syft.

Table 2-5 ▪ Forge Support

Package	Forge Support for Inventory
RPM	<p>Inventory items are associated with components from the following forges:</p> <ul style="list-style-type: none"> • centos • fedora • gnu • fsf • github • savannah <p>If the component is not found in any of these forges, a custom component is created for the inventory item.</p>
Alpine	<p>Inventories are associated with components from the following forges:</p> <ul style="list-style-type: none"> • Alpine • centos • fedora • gnu • github <p>If the component is not found in any of these forges, a custom component is created for the inventory item.</p>
Debian	<p>Library collection is currently not available for Debian packages. All inventories detected in this package type are associated with a custom component.</p>

Important Information About the Syft Integration

As you configure and use the Docker Images plugin, keep the following in mind about the Syft integration in the plugin:

- [Syft as Part of Plugin Bundle](#)
- [Inventory Detection Notes](#)
- [Inventory Names with Licenses](#)

Syft as Part of Plugin Bundle

The Syft artifacts are bundled with the Docker Images plugin and are located in the syft directory (directly under the code-insight-docker-images-plugin directory in the Code Insight plugins zip file). Once the zip file is extracted, the syft directory must remain *as is* along side the code-insight-docker-plugin.jar file. By default, during a scan, Code Insight looks for the syft directory in the same location where this jar file (which runs the scan) is executing. If the syft directory is not found in this location, the scan fails.



Note - The Docker Images plugin uses the 1.9.0 version of Syft.

Inventory Detection Notes

An inventory item discovered by the Syft detection process shows the following system-generated content in the **Notes** field for the inventory item:

Detected By: SyftParser
Attributes:
Source: Syft parser attribute
This item is created as per scan data from Syft

The **Notes** field can include other Syft parser attributes as well, as shown in this example:

Notices Text

Notes

Associated Files (81)

Usage

Custom Fields

Detection

Notes:

Notes from Automated Finding:
As-found name: adduser
Detected By: SyftParser
Attributes:
Source: SyftParser
Licenses from Syft Artifact : {"urls": [], "spdxExpression": "GPL-2.0-only", "locations": [{"path": "/usr/share/doc/adduser/copyright", "layerID": "sha256:78658088978a7596a65f6e864cd30ae00697c62b2b0b82ebf766239ae9c2ed6d", "accessPath": "/usr/share/doc/adduser/copyright"}], "type": "declared", "value": "GPL-2"}
Source: Syft parser attribute
pUrl: pkg.deb/debian/adduser@3.118?arch=all&distro=debian-11
Source: Syft parser attribute
Maintainer: Debian Adduser Developers <adduser@packages.debian.org>
Source: Syft parser attribute
: This item is created as per scan data from Syft.

Inventory Names with Licenses

Only the names of those licenses reported by Syft that have matching names in the Code Insight Data Library are explicitly included in the inventory name. If multiple licenses are reported by Syft, the inventory name can include the names of up to three license matches along with an **or more** clause, indicating that more licenses (with matches or no matches in the data library) are available for you to review. You can see the names of all matched and non-matched licenses detected by Syft for a given inventory item by accessing the inventory's **Detection Notes** in the Code Insight UI.

If Syft reports only licenses that have no matches in the data library (or reports no licenses at all), the inventory name shows **Unknown License** in its name. You can still see the names of any reported non-matched licenses by reviewing the **Detection Notes** for the inventory item.

The following table shows examples of how inventory names in Code Insight reflect the licenses reported by Syft for a given component.

Table 2-6 ■ Inventory Names Reflecting Licenses Reported by Syft

Licenses Reported by Syft	Licenses Matched in Data Library	Example Inventory Name
"GPL-2", "LGPL-2"	GPL-2.0-only, LGPL-2.0-only	libgcrypt20 1.10.1-2 (GPL-2.0-only or LGPL-2.0-only)
"GPL-2", "LGPL-2", "MIT", "Apache-2"	GPL-2.0-only, LGPL-2.0-only, MIT, Apache-2.0	libgcrypt20 1.10.1-2 (GPL-2.0-only or LGPL-2.0-only or MIT or more)
"GP-2", "LL-2"	None	libgcrypt20 1.10.1-2 (Unknown License)
"GPL-2", "LL-2"	GPL-2.0-only	libgcrypt20 1.10.1-2 (GPL-2.0-only or more)
"GPL-2"	GPL-2.0-only	libgcrypt20 1.10.1-2 (GPL-2.0-only)
None	Not applicable	libgcrypt20 1.10.1-2 (Unknown License)

Obtaining the Docker Images Plugin

Extract the Docker Images plugin subfolder from the `CodeInsightversionPlugins.zip` file, and copy it the Docker server. For more information, see [Downloading Plugins](#).

Running a Docker Images Plugin Scan

You can run a Docker Images plugin scan either by providing a properties file that you update to configure the scan or by using command-line options that pass the required configuration information.

The following sections describe both methods for initiating a scan and describe the overall scan process:

- [Running a Docker Images Plugin Scan with a Properties File](#)
- [Running a Docker Images Plugin Scan with Command-Line Options](#)
- [Specifying a Temporary Directory of Your Choice](#)
- [Important Note About an Image Name Containing a "/" in the Command](#)
- [Process of a Docker Images Plugin Scan](#)

Running a Docker Images Plugin Scan with a Properties File

You can run a Docker Images plugin scan in conjunction with a properties file that you manually update before the executing scan.

You initiate the scan by executing the provided shell script or the plugin's jar file. The execute command supports an optional `-tmpdir` parameter, enabling you to specify a temporary folder of your choice for storing and processing the Docker image during the scan (instead of the using default `/tmp` directory).

See the following topics for details:

- [Manually Updating the Properties File for the Docker Images Plugin](#)
- [Initiating a Docker Images Plugin Scan When Using the Properties File](#)

Manually Updating the Properties File for the Docker Images Plugin

Use the following steps manually update the properties file used to configure the Docker Images plugin for scans.



Task

To configure the Docker Images plugin, do the following:

1. Navigate to the Docker Images plugin folder that you copied to the Docker server (as described in [Obtaining the Docker Images Plugin](#)).
2. Open the `code-insight.docker.props` file within a text editor. The following shows the file contents that are shipped with the plugin. You'll need to update the property values for your site.

```
//required
codeinsight.server=http://127.0.0.1:8888/codeinsight
codeinsight.auth.token=Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsInVzZXJJZCI6MSwiaWF0IjoxNTE4MjM0MTk4fQ.dHIJtJjJ2c89Dg5cVLvf
GR3fwJcR3yAlVE6k98dRZTdp3h6McDgv_PloVVE88eJ2GOG0tNDOnhU0ShDLUzdu3Pg
codeinsight.project.name=Docker
plugin.alias.name=dockerRemote
plugin.root.path=/Users/ranimathur/Work/Scratch/
plugin.name=Docker

//optional
plugin.project.name=plugin project name
plugin.project.description=plugin project description
plugin.path.prefix=$demo_workspace/
```

3. Edit the `code-insight.docker.props` file to specify the following information, replacing current values with values appropriate for your site:
 - **codeinsight.server (required)**—The URL path to the Code Insight server in the following format:
`<codeInsight_server_hostname>:<port>/<codeInsight_server_path>`
An example URL might be <http://1.1.1.1:8888/codeinsight>.
 - **codeinsight.auth.token (required)**—The JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it here. Be sure to include the command “Bearer” followed by the token value, as in the example:
[Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsInVzZXJJZCI6MSwiaWF0IjoxNTE4MjM0MTk4fQ.dHIJtJjJ2c89Dg5cVLvfGR3fwJcR3yAlVE6k98dRZTdp3h6McDgv_PloVVE88eJ2GOG0tNDOnhU0ShDLUzdu3Pg](#)
For information about generating this token, see [Providing an Authorization Token](#).
 - **codeinsight.project.name (required)**—The name of the project created in Code Insight to capture the scan results.
 - **plugin.alias.name (required)**—A custom name that you want to give to the scan-agent plugin. This alias is then used as the name of the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the **Analysis Workbench**. The name must be unique within the project.

- **plugin.root.path (required)**—The root path where the Docker Images plugin will be executing. This path must have writable privileges for the user executing the plugin.
- **plugin.name**—The name of the Docker Images plugin. This value must be **Docker**.
- **plugin.project.name (optional)**—A descriptive name of the project being scanned. This can be different from the name specified for the project on the Code Insight server. This text will appear on the **Summary** tab for the project in the Code Insight web UI.
- **plugin.project.description (optional)**—A description of the project being scanned. This text will appear on the **Summary** tab for the project in the Code Insight web UI.
- **plugin.path.prefix (optional)**—The path prefix of the image being scanned. This prefix will be used to reference the codebase file paths on the **Project Inventory** page in the Code Insight web UI.

4. Save the file.

Initiating a Docker Images Plugin Scan When Using the Properties File

To initiate a Docker Images plugin scan that uses the properties file that you configured, you can execute either the shell script or the Java jar file provided with the plugin.

The launch process automatically stores the Docker image in the `/tmp` directory on the machine where the Docker Images plugin is installed. The image contents are then extracted and scanned in this directory and, once the scan completes, the artifacts are deleted from the directory. Alternatively, you can specify a directory of your choice to store and process the image.

The following sections describe how to launch the plugin and scan an image:

- [Using the Shell Script to Launch the Plugin](#)
- [Using the Java jar File to Launch the Plugin](#)



Note ■ The Docker Images plugin must be launched whenever the Docker image is updated. The Docker Images plugin can be included in a script, so the image is scanned regularly.

Using the Shell Script to Launch the Plugin

Use this procedure to launch the Docker Images plugin using the shell script provided with the plugin.



Task

To use the shell script to launch the Docker Images plugin, do the following:

Issue *one* of these command options (where `<Docker_image_name>` is the name for the Docker image that the plugin is to scan).

Option 1: `sh code-insight-docker-plugin.sh -image <Docker_image_name>`

Option 2: `sh code-insight-docker-plugin.sh -image <Docker_image_name> -tmpdir <custom_path>`

Option 3: `./code-insight-docker-plugin.sh -image <Docker_image_name>`

Option 4: `./code-insight-docker-plugin.sh -image <Docker_image_name> -tmpdir <custom_path>`

Note that `-tmpdir <custom_path>` used in Option 2 and Option 4 designates the desired path for storing and processing the image (instead of using the default `/tmp` directory). For more information about this optional parameter, see [Specifying a Temporary Directory of Your Choice](#).

Also, issues can occur if the Docker image name contains a forward slash (`/`). See [Important Note About an Image Name Containing a “/” in the Command](#).

Using the Java jar File to Launch the Plugin

Use this procedure to launch the Docker Images plugin using the Java `.jar` file provided with the plugin.



Task

To use the `.jar` file to launch the Docker Images plugin, do the following:

Issue *one* of these command options (where `<Docker_image_name>` is the name for the Docker image that the plugin is to scan):

Option 1: `java -jar code-insight-docker-plugin.jar -image <Docker_image_name>`

Option 2: `java -jar code-insight-docker-plugin.jar -image <Docker_image_name> -tmpdir <custom_path>`

Note that `-tmpdir <custom_path>` used in Option 2 designates the desired path for storing and processing the image (instead of using the default `/tmp` directory). For more information about this optional parameter, see [Specifying a Temporary Directory of Your Choice](#).

Also, issues can occur if the Docker image name contains a forward slash (`/`). See [Important Note About an Image Name Containing a “/” in the Command](#).

Running a Docker Images Plugin Scan with Command-Line Options

The following procedure initiates a Docker Images plugin scan using command-line options instead of the properties file.

The launch process automatically stores the Docker image in the `/tmp` directory on the machine where the Docker Images plugin is installed. The image contents are then extracted and scanned in this directory and, once the scan completes, the artifacts are deleted from the directory.

Consider the following when using the command-line method to run a scan:

- The available command-line options basically correspond to the properties labeled as **required** (except for **plugin.name**) in properties file `code-insight.docker.props`. The command-line method does not support options corresponding to those properties labeled as **optional** in the file. (If you have used the properties file previously, you can reference it to help you determine what values to supply for the required command-line options.)
- The command must include all the options described in this section (with the exception of the `-tmpdir` parameter, which you can use to specify a custom temporary directory for storing Docker image files).
- During the scan, the command-line options overwrite the values of corresponding properties in `code-insight.docker.props`.



Task

To run a Docker Images plugin scan using command-line options, do the following:

1. Extract the Docker Images plugin subfolder from the CodeInsightversionPlugins.zip file, and copy it the Docker server. For more information, see [Downloading Plugins](#).
2. From a command line, run the a java command in this format:

```
java -Dflx.agent.logLevel=info -jar code-insight-docker-plugin-<VERSION>.jar -server
"<CODEINSIGHT_SERVER_HOSTNAME>:<PORT>/<CODEINSIGHT_SERVER_PATH>" -token "Bearer <JWT_TOKEN>"
- proj "<CODEINSIGHT_PROJECT_NAME>" -root "</path/to/the/codebase>" -image "<IMAGE_NAME>" -alias
"<SCAN_AGENT_ALIAS>" -tmpdir "<custom_path>"
```

For example:

```
java -Dflx.agent.logLevel=info -jar code-insight-docker-plugin-2.4.3.jar -server "http://
1.1.1.1:8888/codeinsight" -token "Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbGlzInVzZXJJZCI6MSwiaWF0IjoxNjg1OTQxNzg3fQ.IPuvTWeZyqRuj9P
AG8eiG7N7ddY4Nuc3QRPRCwMkBXhRofw4A7gOJ-JDdNGLfaXoNa0QEITesoSMxbER8pXlQuuTbe7e" - proj
"libc_proj_1" -root "/users/jsmith/work/scatch" -image "alpinelinux:4.4" -alias "docker45"
-tmpdir "c:/mystuff/scantemp"
```

Refer to the following table for a description of the arguments.

Table 2-7 ■ Command-Line Options for Running a Docker Images Plugin Scan

Command-Line Option	Description	Corresponding Property in Properties File
-jar code-insight-docker-plugin-<VERSION>.jar	Replace <VERSION> with the build version of the .jar file used to run the plugin scan. You can locate the version in the name of the .jar file installed with the plugin. For example: -jar code-insight-docker-plugin-2.4.3.jar	N/A
-server "<CODEINSIGHT_SERVER_HOSTNAME>:<PORT>/<CODEINSIGHT_SERVER_PATH>"	Replace the following with the appropriate values to identify the URL for Code Insight Core Server: <CODEINSIGHT_SERVER_HOSTNAME>:<PORT>/<CODEINSIGHT_SERVER_PATH> For example: -server "http://1.1.1.1:8888/codeinsight"	codeinsight.server

Table 2-7 ■ Command-Line Options for Running a Docker Images Plugin Scan (cont.)

Command-Line Option	Description	Corresponding Property in Properties File
-token "Bearer <JWT_TOKEN>"	<p>Replace <JWT_TOKEN> with a JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it here. Be sure to include the command "Bearer" followed by the token value.</p> <p>For example:</p> <pre>-token "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsInVzZXJJZ..."</pre> <p>For more information about generating this token, see Providing an Authorization Token.</p>	codeinsight.auth.token
-proj "<CODEINSIGHT_PROJECT_NAME>"	<p>Replace <CODEINSIGHT_PROJECT_NAME> with the name of the project created in Code Insight to capture the scan results.</p> <p>For example:</p> <pre>-proj "libc_proj_1"</pre>	codeinsight.project.name
-root "</path/to/the/codebase>"	<p>Replace </path/to/the/codebase> with path where the Docker Images plugin will be executing.</p> <p>For example:</p> <pre>-root "/users/jsmith/work/scatch"</pre> <p>This path must have writable privileges for the user executing the plugin.</p>	plugin.root.path
-image "<IMAGE_NAME>"	<p>Replace <IMAGE_NAME> with the name of the Docker image that the plugin is to scan.</p> <p>For example:</p> <pre>-image "alpinelinux:4.4"</pre> <p>If the image name contains a forward slash (/), ensure a valid tag is used with the name. See Important Note About an Image Name Containing a "/" in the Command.</p>	N/A

Table 2-7 ■ Command-Line Options for Running a Docker Images Plugin Scan (cont.)

Command-Line Option	Description	Corresponding Property in Properties File
-alias "<SCAN_AGENT_ALIAS>"	<p>Replace <SCAN_AGENT_ALIAS> with the name that you want to give to the scan-agent plugin. This alias is then used as the name of the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the Analysis Workbench. The name must be unique within the project.</p> <p>For example:</p> <pre>-alias "docker45"</pre>	plugin.alias.name
-tmpdir "<CUSTOM_PATH>"	<p>(Optional) Replace <CUSTOM_PATH> with the desired path for storing and processing the image. (By default, the scan uses the /tmp directory.) For more information about this optional parameter, see Specifying a Temporary Directory of Your Choice.</p>	N/A

Specifying a Temporary Directory of Your Choice

By default, the Docker Images plugin both stores the specified image file and scans its extracted contents in the /tmp directory on the machine where the plugin is installed. However, this directory might not always have sufficient space to process these files.

You have the option to specify a different directory in which to store the image and scan its extracted files. As with the /tmp directory, the artifacts are deleted from this directory once the scan completes. If you choose to specify such a directory, do the following:

- Provide the directory path with the -tmpdir token as shown in the previous examples. If either the path or the token is missing, the scan fails.
- Specify an absolute path or a path relative to the plugin installation.
- Do not include spaces in the path as in `/home /admin/Docker2023r/custom_dir`.
- Ensure that the path is a valid path on the machine.

Important Note About an Image Name Containing a “/” in the Command

If the image name in the command contains a forward slash (/), you must provide a valid tag for name, such as “latest”, to avoid issues during the scan. The following example commands illustrate the correct <name>:<tag> format to use when the image name contains a forward slash:

```
./code-insight-docker-plugin.sh -image alpine/linhttpd:latest
```


Prerequisites for the Generic Scan-Agent Plugin

The following prerequisites are required to use the Code Insight generic scan-agent plugin:

- A minimum of 4 GB Java heap space required for scanning (allocated to the JVM under which the scan-agent plugin will be executing)
- 64-bit JRE 8 or JDK 11
- Code Insight Core Server
- Code Insight prerequisites described in [Preparing to Use the Plugins](#)
- Internet access (recommended but not required):
 - If Internet access is available, the scan-agent will periodically download the latest security vulnerability definitions from the National Vulnerability Database (NVD).
 - If Internet access is not available, then the default signatures that were released with the latest version of Code Insight will be used.

Support for Systems Running on Java 11

Certain systems (such as Jenkins) on which you intend to run scans require Java.

The generic plugin available with Code Insight 2023 R1 or later supports such systems running on Java 11 (Oracle or OpenJDK 11) or Java 8. (The plugin available with the previous Code Insight versions support only Java 8 for these systems.)



Note • The Code Insight Core and Scan Servers continue to run on Java 8 only.

Running the Generic Scan-Agent Plugin

The generic scan-agent plugin can scan any file system of your choice, without your being limited to a specific build system as you are with the standard scan-agent plugins.

The scan returns the results back to Code Insight, where the discovered inventory items for your project can be reviewed automatically via policies or manually reviewed by various stakeholders. Security alerts with corresponding email notifications will be generated for any inventory items with new security vulnerabilities.



Note • The first time a scan is performed using the generic scan-agent plugin, a data snapshot is downloaded from the National Vulnerability Database (NVD) to generate an index of the latest security vulnerabilities.



Task

To run the generic scan-agent plugin, do the following:

1. Download and extract the contents of the `CodeInsightversionPlugins.zip` file, as described in the previous section, [Downloading Plugins](#).
2. Locate the `code-insight-agent-sdk-generic/generic-plugin-binary` folder and copy it to your hard drive.

3. Execute the required command from the command line as a Java application to perform the scan and report download operations using the plugin.

- a. To execute a scan using the plugin, run the following command:

```
java -Dflx.agent.logLevel=info -jar codeinsight-generic-<VERSION>.jar -server  
    "<CODEINSIGHT_SERVER_HOSTNAME>:<PORT>/<CODEINSIGHT_SERVER_PATH>" -token "Bearer  
<JWT_TOKEN>" -proj "<CODEINSIGHT_PROJECT_NAME>" -root "</path/to/the/codebase>" -scandirs  
    "</path/to/the/codebase/PROJECT>" -alias "<SCAN_AGENT_ALIAS>" -host "<SCAN_AGENT_HOST>"  
    -Djavax.net.ssl.trustStore="<JAVA_HOME>\lib\security\cacerts
```

- b. To download a report using the plugin, run the following command:

```
java -Dflx.agent.logLevel=info -jar codeinsight-generic-<VERSION>.jar -server  
    "<CODEINSIGHT_SERVER_HOSTNAME>:<PORT>/<CODEINSIGHT_SERVER_PATH>" -token "Bearer  
<JWT_TOKEN>" -proj "<CODEINSIGHT_PROJECT_NAME>" -root "</path/to/the/codebase>" -scandirs  
    "</path/to/the/codebase/PROJECT>" -alias "<SCAN_AGENT_ALIAS>" -host "<SCAN_AGENT_HOST>"  
    -Djavax.net.ssl.trustStore="<JAVA_HOME>\lib\security\cacerts -reportType "<REPORT_TYPE>"  
    -reportDownloadLocation "<REPORT_DOWNLOAD_LOCATION>" -reportOptions "<REPORT_OPTION_FILE>"
```

Replace the following variables with the appropriate information:

- **<VERSION>**—The build version of the .jar file used to run the scan agent. The version is shown in the name of .jar file, which is located in the code-insight-agent-sdk-generic/generic-plugin-binary folder.
- **<CODEINSIGHT_SERVER_HOSTNAME>:<PORT>/<CODEINSIGHT_SERVER_PATH>**—The URL for the Code Insight Core Server (for example, <http://1.1.1.1:8888/codeinsight>).
- **<JWT_TOKEN>**—Your JSON Web Token (JWT) used to authorize user access to the Code Insight functionality. Generate this token using the Code Insight Web UI and then copy and paste it in this field. For more information, see [Providing an Authorization Token](#).
- **<CODEINSIGHT_PROJECT NAME>**—The project you created in Code Insight to capture the inventory.
- **<path/to/the/codebase>**—The root path for the codebase to be scanned.
- **</path/to/the/codebase/PROJECT>**—The specific directories to be scanned.
- **<SCAN_AGENT_ALIAS>**—A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the **Analysis Workbench**. This name must be unique within the project.
- **<SCAN_AGENT_HOST>**—(Optional) A user-defined name for the instance where the scan-agent plugin is configured to run scans. This property along with the alias property will remain unchanged for each subsequent rescan.

The -host property (for which you need to enter this variable) is required only if you are running the scan in a dynamic host environment. See [Note About Rescans Performed by v2.0 and Later Plugins](#).

- **<JAVA_HOME>**—(Optional) The path defined for your JAVA_HOME environmental variable on the instance running the Code Insight Core Server. The -Djavax.net.ssl.trustStore property (for which you need to enter this variable) is required only if SSL is configured on the instance.
- **<REPORT_TYPE>**—(Optional) The name of a Code Insight report that you want to be generated as a part of the scan build process once the scan completes.
- **<REPORT_DOWNLOAD_LOCATION>**—(Optional) The directory where you want the report to be downloaded.

- **<REPORT_OPTION_FILE>**—(Optional) The directory to the JSON file that includes additional parameters for custom reports. Specifying the JSON file directory using the `-reportOptions` property allows the generic scan-agent plugin to apply the parameters from that file to customize the report generation process.



Note ▪ The `-reportOptions` property is required only when generating customized reports and is not needed for standard report generation.

Alternatively, you run a scan using one of two scripts, `run_scan.bat` or `run_scan.sh`, provided with the generic scan-agent plugin. The scripts located in the generic-plugin-binary folder.

Status of the Import of Remote Scan Results into Your Project

To view the status of the import of scan results into your Code Insight project from the most recent remote scan run for your project, navigate to the project's **Summary** page. In the **Scan Status** section, locate the status and timestamp of the import. (For a description of these remote-scan details, see the “Summary Tab” topic under “Code Insight User Roles and Permissions” section in the *Code Insight User Guide*.)

Scan Status
Scan Server Status: No scan scheduled. Click [here](#) to schedule a scan for this project
Last Server Scan: Scan of project Spotal_Generic **failed**.
Past Server Scans: Click [here](#) to view the scan history for this project.
Last Remote Scan: Scan Summary : 1,620 Files | 245.48 MB **Completed on 2022-01-07 12:59:18**

Support for Processing Remote Scan Results in the Background

For scans performed by the generic or Docker Images scan-agent plugin only, the phase in which the scan results are processed in Code Insight for a given project is run as a background job that users can track in the **Jobs** queue. During such scans, the plugin sends the scan results in JSON format to Code Insight as a `.txt` file to be stored temporarily in Code Insight. Once the results are successfully sent, the job ID for processing the results in Code Insight is created and the job is added to the job queue. (The ID is also returned to the plugin to indicate that the results were successfully sent.) Users can track the progress of the results-processing as a **Remote Scan** job in the **Jobs** queue. (Progress is shown as **Active**, **Scheduled**, **Completed**, or **Failed**.) When Code Insight completes the processing, the temporary file is removed.

For more information about the **Jobs** queue, see the “Monitoring the Code Insight Jobs Queue” section in the *Code Insight User Guide*.

For all other plugin scans, as soon as Code Insight receives the results, they are processed in the foreground.

About Running a Other Jobs Along with a Job That Processes Remote Scan Results

Note the following about running a job for remote-scan processing along with other jobs:

- A **Remote Scan** job for a given project can run simultaneously with a job for Scan Server scan (**Project Scan**) or rescan (**Project Re-Scan**) on the same project or on different projects.
- Only one **Remote Scan** job can run on a project at a given time. All other **Remote Scan** jobs for the project are placed in a **Scheduled** state and run in scheduled order.
- When a **Remote Scan** job for a specific project is in progress or waiting to run (with an **Active**, **New**, **Waiting on Update**, or **Scheduled** status), a job for a project import, SBOM Insights export, report generation, or project deletion for the same project is placed in a **Scheduled** state and run in scheduled order. Likewise, when any of these other jobs are currently in progress or waiting to run, a new **Remote Scan** job for the same project is placed in a **Scheduled** state and run in scheduled order.
- When a **Remote Scan** job for a specific project is in progress or waiting to run (with an **Active**, **New**, **Waiting on Update**, or **Scheduled** status), a branching process cannot be performed on the same project. (The **Branch Project** option on the **Manage Project** menu for the project is disabled.) The user must wait for the **Remote Scan** job to complete before attempting to run the branching process again.

Similarly, if a branching process is currently in progress or waiting to run for a specific project and a remote scan is attempted for the same project, the target project displays the message “Cannot import data for remote scan since project branching is in progress for the same project <projectid>”.

- When a **Remote Scan** job for a specific project is in progress or waiting to run (with an **Active**, **New**, **Waiting on Update**, or **Scheduled** status), a copy of this project cannot be performed. If a user attempts to run a project copy, a pop-up is displayed, explaining this situation. The user must wait for the **Remote Scan** job to complete before attempting to run the copy again.

Similarly, if a project copy is currently in progress or waiting to run for a specific project and a remote scan is attempted for the same project, both the source and the target project display the message “Cannot import data for remote scan since project copy is in progress for the same project <projectid>”.

- If an Electronic Update is currently in progress, all subsequent **Remote Scan** jobs are placed in a **Scheduled** state. Once the update is finished, these jobs are run based on the scheduled order.
- If a Library Refresh is currently scheduled or in progress and a **Remote Scan** job is added to the queue, the job fails. You must wait until the Library Refresh is completed before having the scan-agent plugin rerun the scan.

Note About Rescans Performed by v2.0 and Later Plugins

Scan agent v2.0 and later plugins support alias names. An alias name is used to uniquely identify a remote scan agent for a given project as well as differentiate the codebase scanned via the agent from other project codebase files. Alias names are unique within a given project and cannot be shared across multiple scan agents.

In general, a rescan performed by a v2.0 or later scan-agent plugin uses the same alias and hostname that the previous scan used. However, in a dynamic host environment (such as is supported by CI tools, where hosts are dynamically allocated as needed for cloud or linked builds), the instance on which a rescan is run might be different from the instance used in the previous scan, in turn causing the rescan to fail.

Therefore, for those v2.0 and later plugins used for Engineering platforms that support dynamic host environments, you must provide a value for the new “host” property in the plugin configuration. This value should be a user-defined name for the host instance on which the scan will be run. The value will then remain the same even if the instance used for a rescan is different from the one used in the previous scan.

This property is currently available for the Jenkins, Bamboo, Azure DevOps, and the generic scan-agent plugins.

Developing Custom Plugins

While Code Insight provides standard scan-agent plugins that are ready to deploy for codebase scanning on remote Engineering systems, it also provides a Scan Agent toolkit that implements the Code Insight Scan Agent Framework, which enables you to write a custom scan-agent plugin that integrates with your development ecosystem. The following sections provide guidance in creating a scan-agent plugin:

- [Scan Agent Framework](#)
- [Downloading the Scan Agent Toolkit](#)
- [Contents of the Scan Agent Toolkit](#)
- [Writing a Custom Scan-Agent Plugin](#)
- [Deploying a Custom Scan-Agent Plugin](#)

Scan Agent Framework

The Scan Agent Framework comprises a set of Java APIs that provide the ability to scan codebases at various phases of the development process on remote Engineering systems, within the appropriate context. The Framework provides the backbone for all the processing that a user-created scan-agent plugin requires.

The source code for a generic scan-agent plugin is provided in the toolkit to demonstrate the API flow. By creating a plugin that takes advantage of the Scan Agent Framework, you can tailor Code Insight's powerful scanning capabilities to your computing environment and incorporate them into your business process flow.

The following describes the Scan Agent Framework:

- [Features Provided by the Framework](#)
- [Available Classes and Methods in the Framework](#)
- [Property Settings](#)

Features Provided by the Framework

The Scan Agent Framework provides the following functionality for the custom scan-agent plugin:

- Tests the connection from your plugin to the Code Insight server and provides error handling with error messages. These messages include the Code Insight version, any invalid URLs passed, invalid user access tokens, and invalid project names.
- Passes environmental and system properties.
- Downloads and installs a remote scanner on the Engineering system where the scan-agent plugin is executed.
- Invokes the scan called for in the plugin, and sends the scan results back to Code Insight.
- Processes and displays logging content to a system console in the scan-agent plugin environment.
- Generates a verbose scanner log for further information and debugging of failed scans.
- Automatically uploads the output of the plugin to the Code Insight server, where it is then available for inventory review, management, and security alerting via the Code Insight user interface.


Available Classes and Methods in the Framework

The Scan Agent Framework provides the following classes and methods that can be used to build a scan plugin:

Table 3-1 ■ Available Classes and Methods

Class	Description	Settings/Methods
ExecutionContext	Stores all the information needed for a scan to run and publish results to Code Insight server. This class should be initialized only by calling the <code>ExecutionContext.getInstance(...)</code> method.	Methods: <code>ExecutionContext getInstance(Properties props, PrintStream logger)</code> Initializes ExecutionContext for the current scan. Parameters: <code>props</code> : Properties required to scan and publish results <code>logger</code> : Output stream where all the useful logging will be sent; if logger is null, all logging is redirected to console output. For more information about parameters for this class, see Property Settings .

Table 3-1 ■ Available Classes and Methods (cont.)

Class	Description	Settings/Methods
ScanExecutor	<p>Executes the scan and the publish results request made by client plugins.</p>  <p>Note ■ This class requires a valid <i>ExecutionContext</i> instance to be able to operate.</p>	<p>Methods:</p> <ul style="list-style-type: none"> ● ScanExecutor(ExecutionContext executionContext) A constructor. ● String testConnection() Validates the user authorization token and the connection to the Code Insight server. <p>Valid returned strings:</p> <ul style="list-style-type: none"> ● Success ● Server Not Found ● Invalid Auth Token Found ● User not authorized to access the project ● String scanCodebase(List<String> paths) Updates the embedded scanner if updates are found, scans the paths provided as input, and publishes the results to the Code Insight server as provided for ExecutionContext. <p>Parameters:</p> <p>paths: list of absolute paths to be scanned</p> <p>Return:</p> <p>SUCCESS or FAILURE as a string. See the logger output stream for more details.</p>

Property Settings

The following table lists property settings that you can update:

Table 3-2 ■ Property Settings

Property	Required/Optional?	Description
codeinsight.server	Required	The Code Insight server URI (for example, http(s)://your.codeinsight.server:port/).
codeinsight.auth.token	Required	Authorized user token generated in the Code Insight user interface.
codeinsight.project.name	Required	The Code Insight project name where all the inventory information will be published.

Table 3-2 ▪ Property Settings (cont.)

Property	Required/Optional?	Description
plugin.root.path	Required	The local root path that can be replaced with path prefix.
plugin.project.name	Optional	The name of the plugin project (for example, “Codeinsight Jenkins”).
plugin.project.description	Optional	The plugin’s instance name/build number or tag that can be sent to the Code Insight server.
plugin.path.prefix	Optional	Prefix to be added to file paths for display to the user. For example, it could be the URL for a Jenkins workspace.
plugin.alias.name	Required	A name that you define for the scan-agent plugin. The alias is used to represent the “container” (scan root) under which all the files scanned in this instance will be listed in the API output and in the file tree in the Analysis Workbench . This name must be unique within the project.
plugin.proxy.host	Required when using proxy server	The IP address or Hostname of the proxy server.
plugin.proxy.port	Required when using proxy server	The port used for the proxy server.
plugin.proxy.user	Required when using proxy server	The user name used to authenticate the proxy.
plugin.proxy.password	Required when using proxy server	The password used to authenticate the proxy.
plugin.has.proxy	Required when using proxy server	The value indicating whether the proxy is enabled for the plugin: <ul style="list-style-type: none"> ● true enables the proxy. ● false disables it.
plugin.host	Optional	(Optional) A user-defined name for the instance where the scan-agent plugin is configured to run scans. This property along with the alias property will remain unchanged for each subsequent rescan. Although optional in general, this value is required if you are running the scan in a dynamic host environment. See Note About Rescans Performed by v2.0 and Later Plugins .

Downloading the Scan Agent Toolkit

Use the following procedure to download and extract the Scan Agent toolkit.



Task

To download the Scan Agent toolkit, do the following:

1. Access the Revenera Community site and sign in:
<https://community.revenera.com>
2. In **Find My Product**, click **FlexNet Code Insight**.
3. Under **Product Resources** on the right, click **Download Product and Licenses**.
4. Once in the Product and License Center, navigate to **Your Downloads** and select **FlexNet Code Insight**. The **Download Packages** page is displayed.
5. Select the version of Code Insight from the list. The **Downloads** page appears.
6. Select the Code Insight Plugins version, and download its associated `CodeInsightversionPlugins.zip` file.
7. When the download finishes, extract the subfolder `code-insight-agent-sdk-generic-plugin`, which contains the toolkit.

Ensure that you extract the entire subfolder into your installation directory, so you have all necessary files to create the plugin.

Contents of the Scan Agent Toolkit

The contents of the Scan Agent toolkit includes the following:

- **/readme.txt**: Instructions on how to use the SDK.
- **/lib**: All dependent jar files needed to run the plugin.
- **/generic-plugin-binary**: An example binary—the generic scan-agent plugin—developed with the Scan Agent Framework.
- **/generic-plugin-source**: The source code for the example binary.
- **/generic-plugin-source/pom.xml+assembly.xml**: Maven build files used to compile and build the plugin.

Writing a Custom Scan-Agent Plugin

Writing a custom scan-agent plugin involves the following tasks:

- **Task 1**— Review the example generic scan-agent plugin for its code structure and build configuration. (In essence, you can use this plugin as a template for creating your own.)
- **Task 2**—Identify the Engineering system with which you will be integrating the scan agent. (This section and the next will use *Integration Server* as an example Engineering system).

- **Task 3**—Using the APIs provided by the Integration Server, write code to pull in all the codebase files that you want to scan into a single folder. This will be the folder path that you will be passing to the plugin.
- **Task 4**—Use the Scan Agent Framework APIs to connect to the Code Insight server.
- **Task 5**—Scan the folder that contains the desired code base files on the Integration Server.



Note ▪ *The plugin will typically execute on the same computer system as the Integration Server.*

Deploying a Custom Scan-Agent Plugin

Deployment of a custom scan-agent plugin involves the following tasks:

- **Task 1**—Install and configure the Code Insight server.
- **Task 2**—Ensure you adhere to the system prerequisites for the generic scan-agent plugin. See [Prerequisites for the Generic Scan-Agent Plugin](#).
- **Task 3**—Invoke the remote scans on the Integration Server, and review the results in the Code Insight user interface.