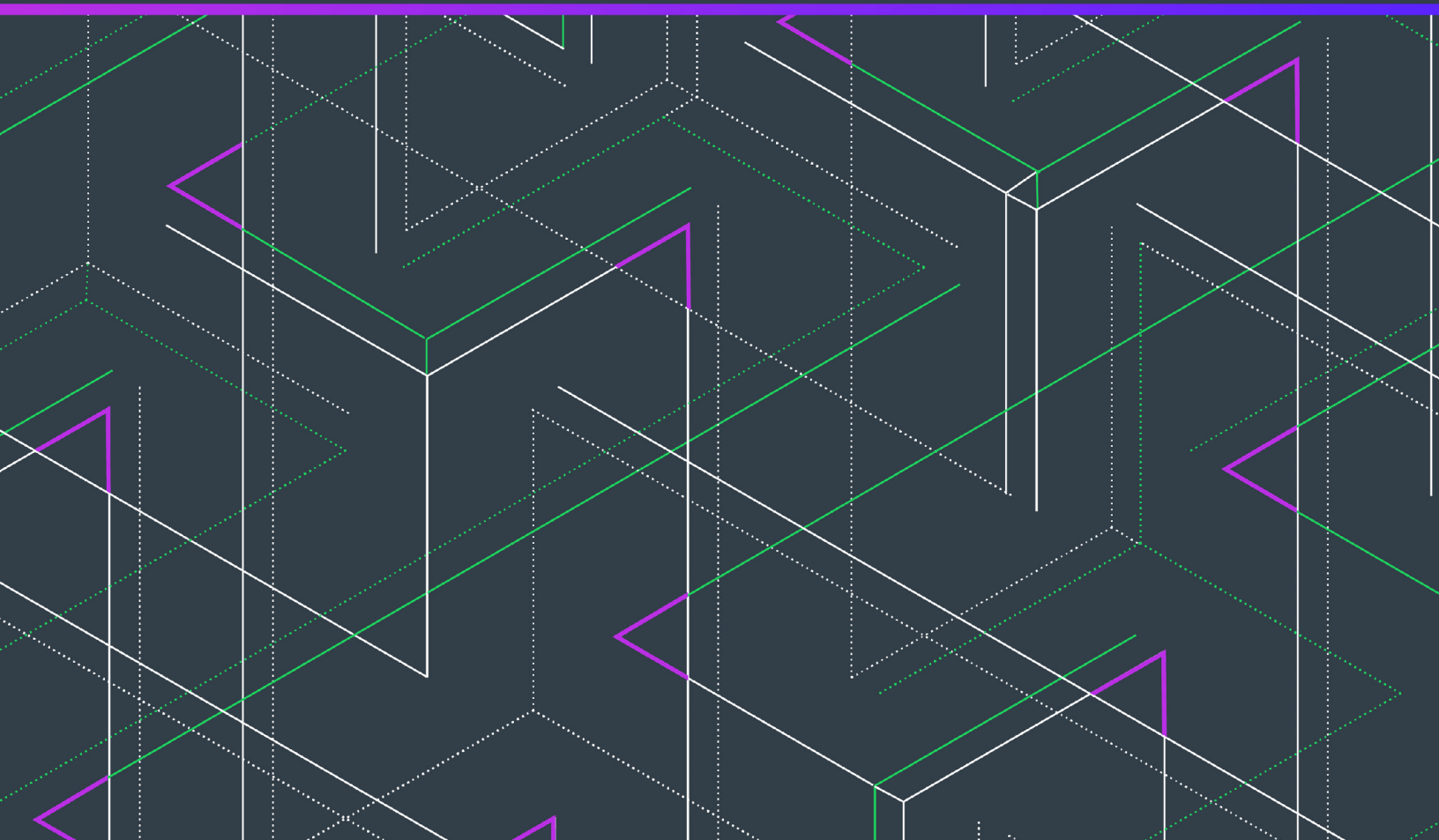
revenera™

Code Insight 2021 R2

Plugins Guide



Legal Information

Book Name: Code Insight 2021 R2 Plugins Guide

Part Number: RCI-2021R2-PG00

Product Release Date: May 2021

Copyright Notice

Intellectual Property

Restricted Rights Legend

Contents

1

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:

About 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 with using this product:

- Revenera Product Documentation
- Revenera Community
- Revenera Learning Center
- Revenera Support

Revenera Product Documentation

You can find documentation for all Revenera products on the Revenera Product Documentation site:

https://docs.revenera.com

Revenera Community

On the Revenera 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 Revenera's product solutions, you can access forums, blog posts, and knowledge base articles.

https://community.revenera.com

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 making selections on the **Get Support** menu of the Revenera Community.

https://community.revenera.com

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

2

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

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 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.

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.
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.
Binary Repositories	JFrog Artifactory Plugin	Artifactory repositories to identify non-compliant artifacts.
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 to Version 2.0 in Code Insight 2020 R3

Code Insight scan-agent plugins were upgraded from version 1.x to 2.0 starting in the Code Insight 2020 R3 release.

The 1.*x* plugins require an inventory-only project on the Code Insight Core Server to which to send only the inventory results from the remote scans. However, the 2.0 plugins, whose scans capture both inventory and codebase-file information, send 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, introduced in Code Insight 2020 R3, see "Legacy Projects" in the *Code Insight User Guide*.

Note that the configuration of all 2.0 plugins requires a new "alias" property to identify the scan agent to Code Insight. A new "host" property might also be required for certain plugins (see Note About Rescans Performed by 2.0 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
- JFrog Artifactory
- 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. See "Creating a Project" in the *Code Insight User Guide* for details.

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 Revenera Community. The following procedure assumes you have a login and password to access the Revenera Community.

Task ***To download the plugin zip file, do the following:***

1. Access the Revenera Community site and sign in:

   https://community.revenera.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 `CodeInsight`*`version`*`Plugins.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, Artifactory, 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

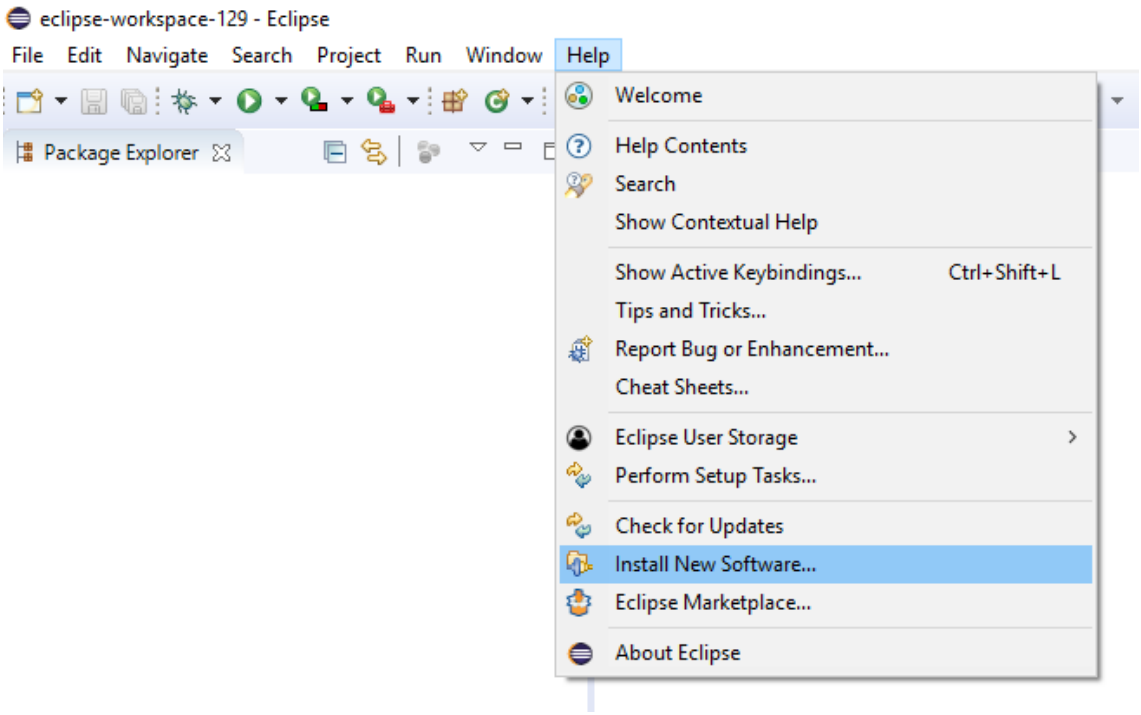
Before you can install and configure the Code Insight plugin for Eclipse, perform the required tasks described in Preparing to Use the Plugins for details.

Installing the Eclipse Plugin

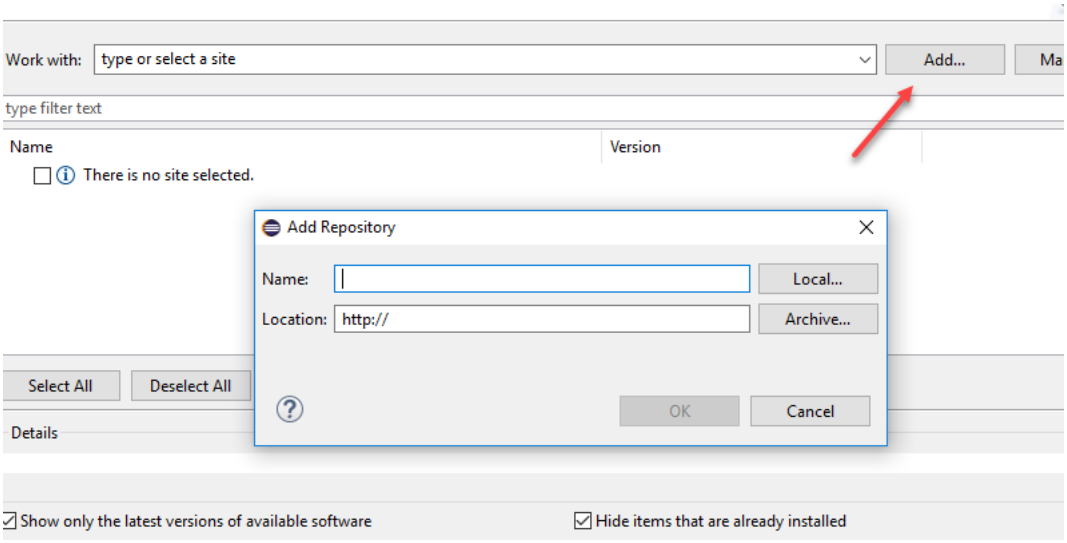
Once you have ensured that the prerequisites are met, use this procedure to install the Eclipse plugin in your Eclipse environment.

Task ***To install the Eclipse plugin, do the following:***

1. Ensure that you have extracted the `code-insight-eclipse-scan` folder from the `CodeInsight`*`version`*`Plugins.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**.



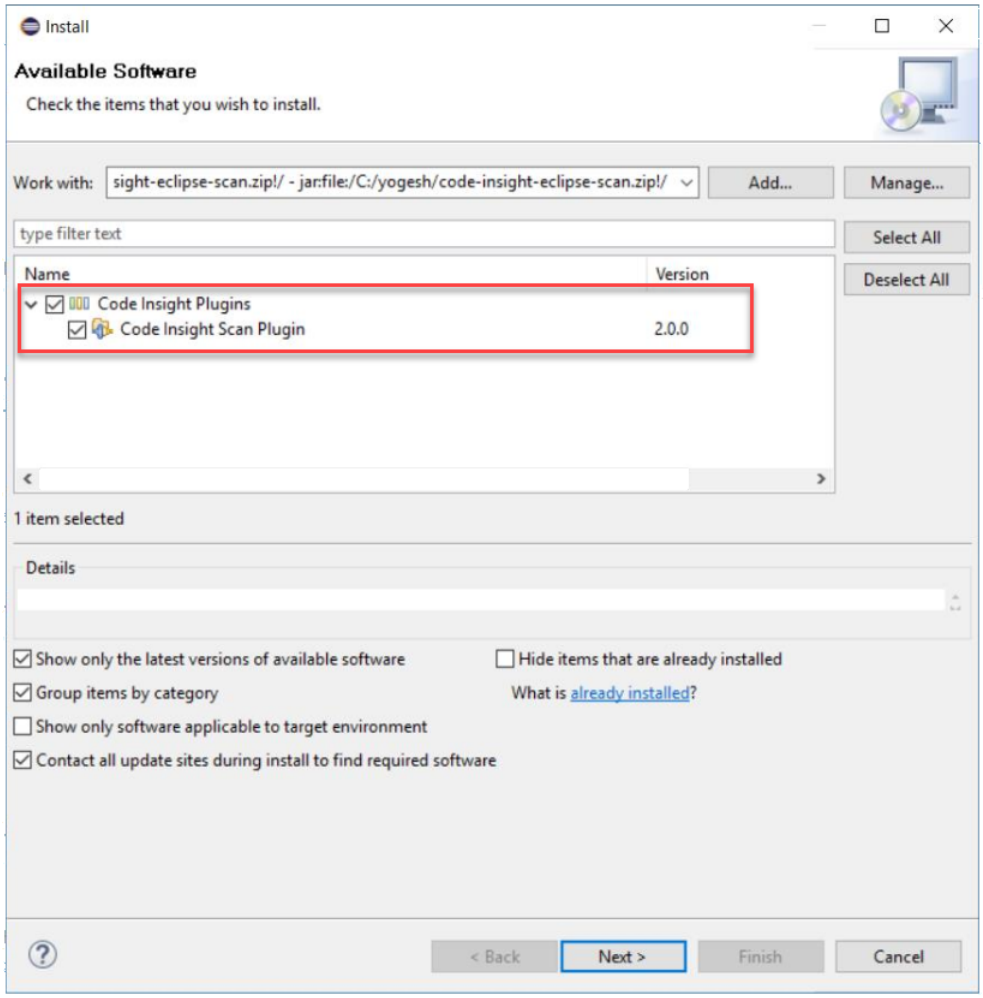
3. On the **Available Software** window, click the **Add** to display the **Add Repository** popup:



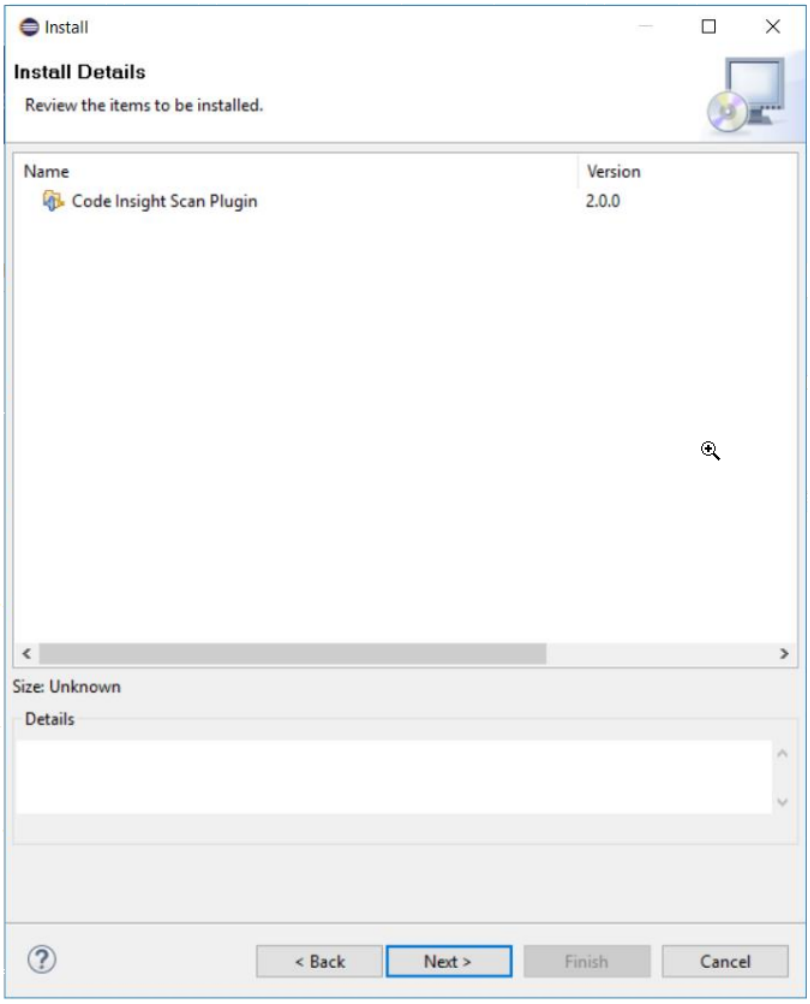
4. On the popup, click **Archive**.

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.



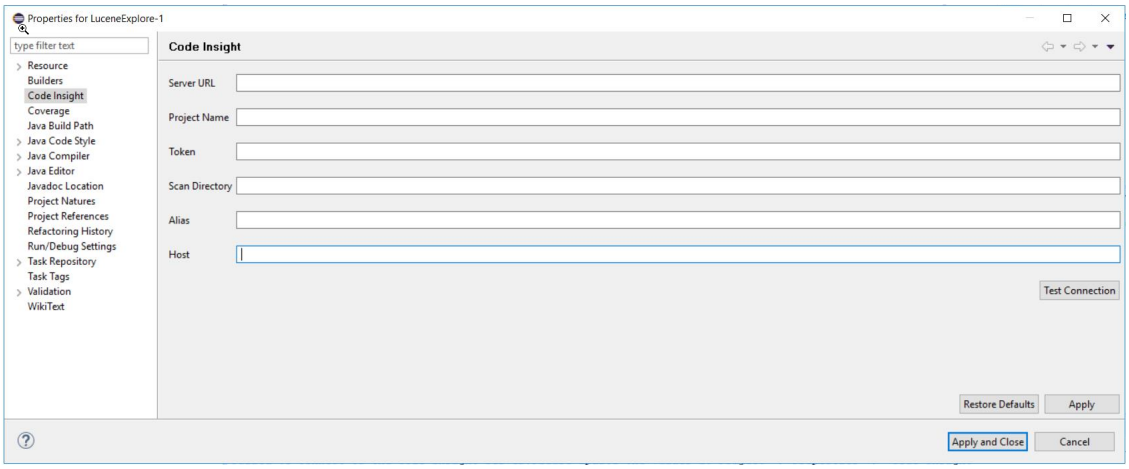
7. Click **Next** to display the **Review Licenses** window.
8. Accept the license agreement terms, and click **Finish**.
9. When the installation is completed, restart Eclipse.

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	(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 2.0 Plugins.

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.

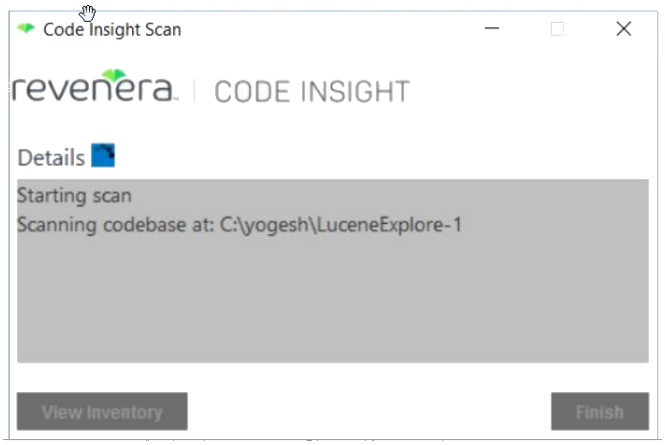
5. When the plugin has been properly configured, click **Apply and Close**.

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:***

1. 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.



2. 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:***

1. Open Eclipse, and select **Help | About Eclipse**.
2. In the **About Eclipse** window, click **Installation Details**.
3. In the **Eclipse Installation Details** window, select **Code Insight Scan Plugin**, and click **Uninstall...**.

   The **Uninstall** window is displayed, listing the plugin.
4. 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.

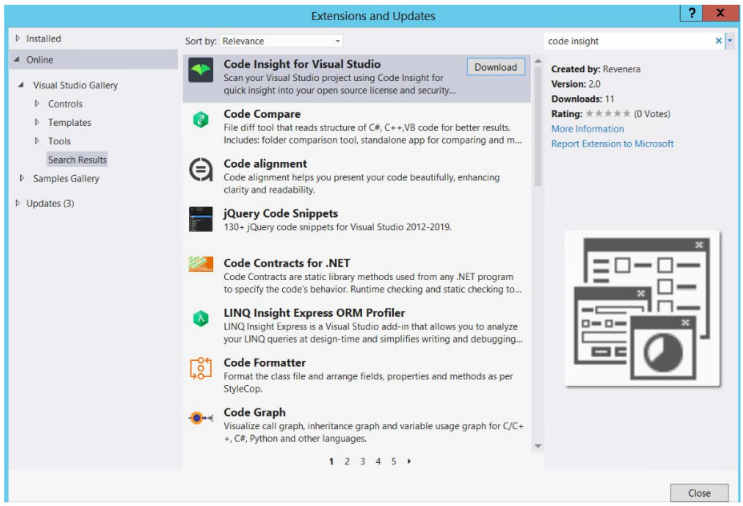
Note ▪ *Microsoft Visual Studio Express does not support the Visual Studio plugin.*

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:***

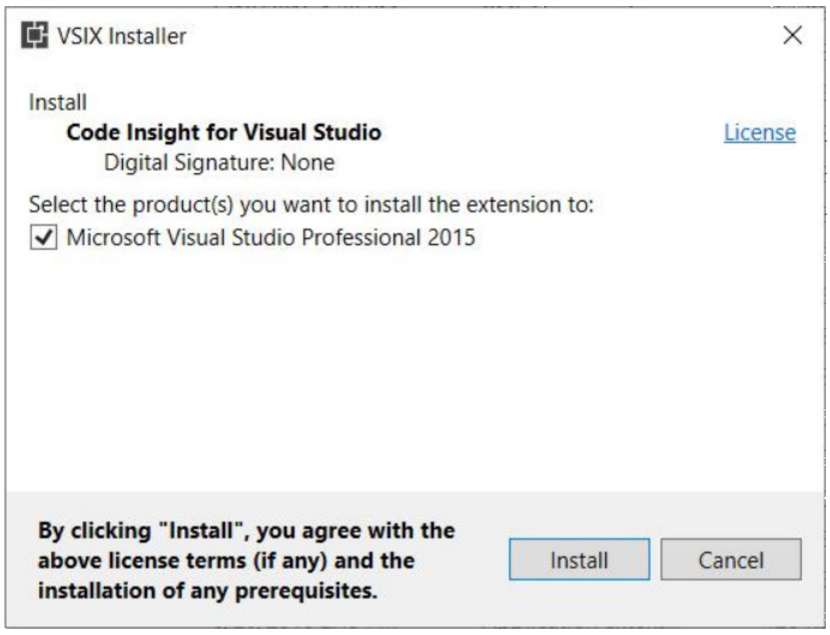
1. In Visual Studio IDE, select **Tools | Extensions and Updates**.
2. In the **Extensions and Updates** tree, search for the **Code Insight for Visual Studio** extension in the **Online | Visual Studio Marketplace** category.



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

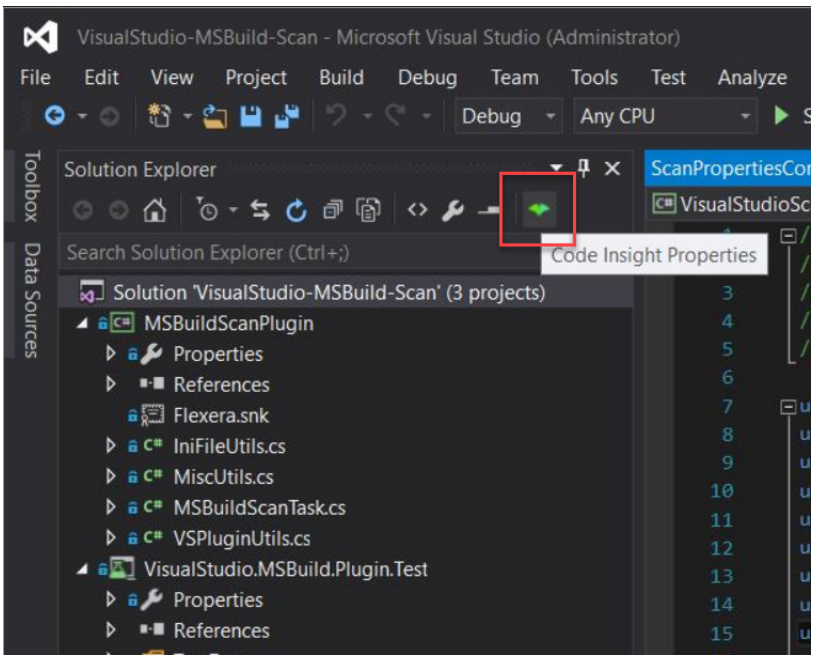
Configuration Using Visual Studio IDE

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.

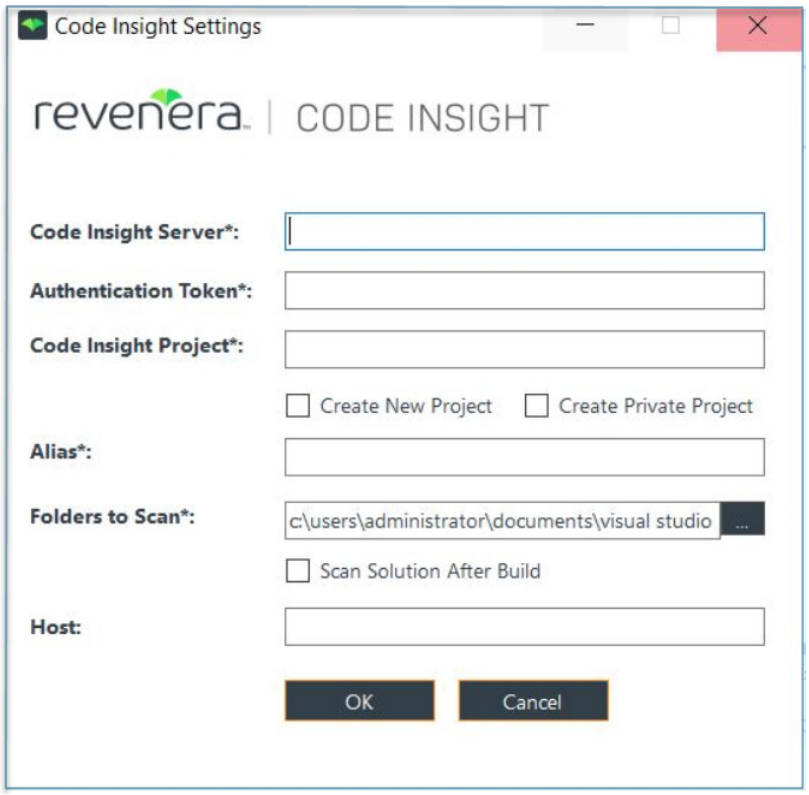
The following steps describe this configuration process.

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 Settings** icon .



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



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	Enter the URL for the Code Insight core server in the format `http://<SERVERHOSTNAME>:<PORT>/codeinsight/` (for example, `http://codeInsightServer.myorg.org:8888/codeinsight/`). Ensure that the URL is publicly accessible and that the port is available.
Authentication Token	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.
Code Insight Project	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. If you want the configuration process to create the specified Code Insight project, select **Create New Project** (see the next table entry). If the specified project already exists on the Code Insight server, do *not* select **Create New Project** or **Create Private Project** (see the next table entries).

Field	Description
Create New Project	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**. If you want this new project to be *public*—that is, viewable by all Code Insight users—leave the next option, **Create Private Project**, unselected. Otherwise, select **Create Private Project** (described next).
Create Private Project	Select this option if you selected **Create a New Project** and want this new project to be a *private*—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**. Leave this option unselected if you want the new project to be public.
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.
Folders to Scan	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.  ***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.*
Scan Solution After Build	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.
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 2.0 Plugins.

4. 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	Provide the URL for the Code Insight core server in the format `http://<SERVERHOSTNAME>:<PORT>/codeinsight/` (for example, `http://codeInsightServer.myorg.org:8888/codeinsight/`). Ensure that the URL is publicly accessible and that the port is available.
AuthenticationToken	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.
CodeInsight Project	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. 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). If the specified project already exists on the Code Insight server, ensure that **CreateNewProject** and **CreatePrivateProject** are set to `False`.
ScanFolders	Specify the absolute paths for the project output folders (or any additional folders) to scan for the solution, separating each path with a comma.

Property	Description
CreateNewProject	If the Code Insight project you specified for **CodeInsightProject** already exists, set this property to `False`. 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**. Use the **CreatePrivateProject** property to define the new project as public or private.
CreatePrivateProject	Determine whether the new project is to be created as public or private: ● If you want this new project to be *public*—that is, viewable by all Code Insight users, set this property to `False`. ● If you want this project to private to *private*—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	Specify `True` to have the Code Insight scan execute automatically after the Build or Rebuild Solution step. Specify `False` to start each scan manually, as described in Executing a 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 API output and in the file tree in the **Analysis Workbench**. 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 optional in general, this value is required if you are running the scan in a dynamic host environment. See Note About Rescans Performed by 2.0 Plugins.

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

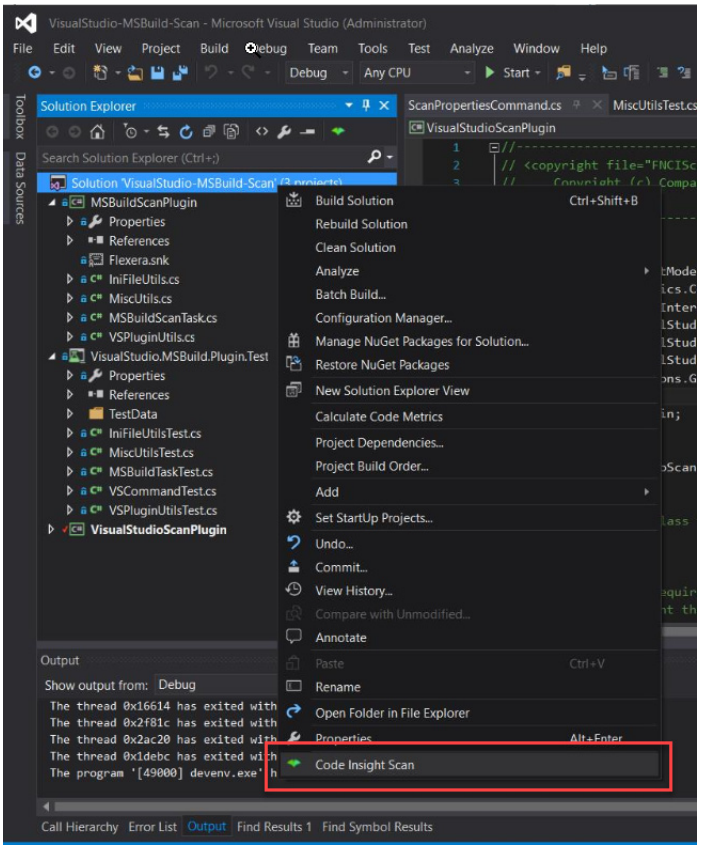
The Visual Studio plugin can also be configured to trigger a scan automatically at the end of each build or rebuild. (See Configuring the Visual Studio Plugin for configuration details.) When the scan completes, you can click the URL at the end of the build output to connect to Code Insight. You are 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`
VS2012, VS2015	`C:\Windows\Microsoft.NET\Framework\<FRAMEWORK_VERSION>`
VS2015	`C:\Program Files (x86)\MSBuild\14.0\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>
     ```

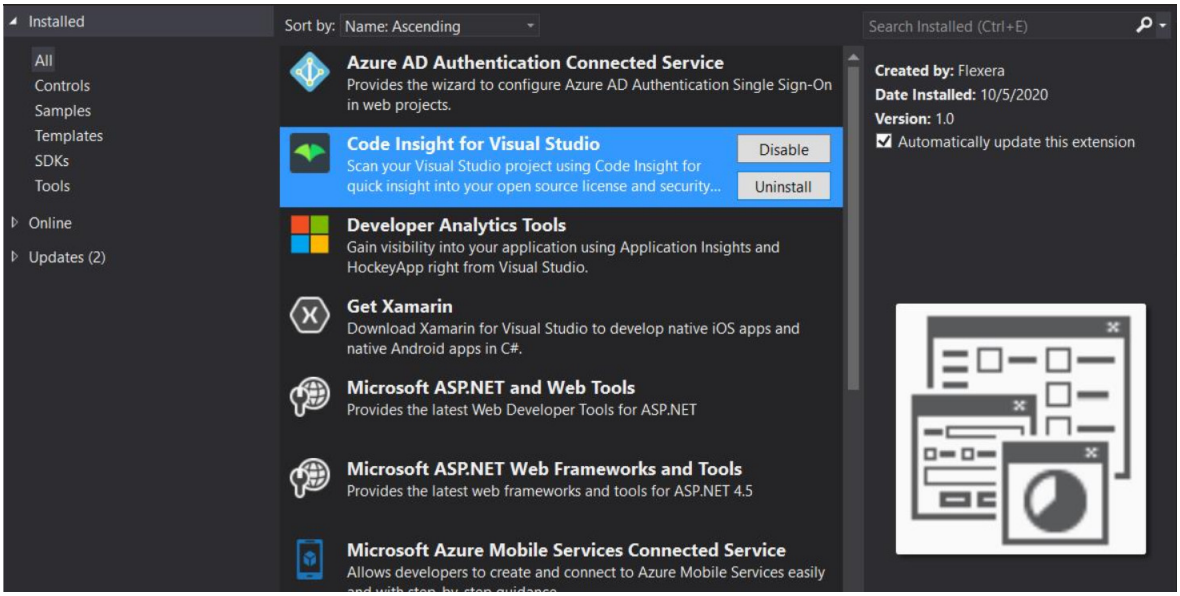
3. When the scan completes, click the URL at the end of the build output to connect to Code Insight. You are opened to the **Project Inventory** tab for the Code Insight project created for the scan. From here you can review, manage, and remediate the inventory resulting from the scan. For further instructions,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

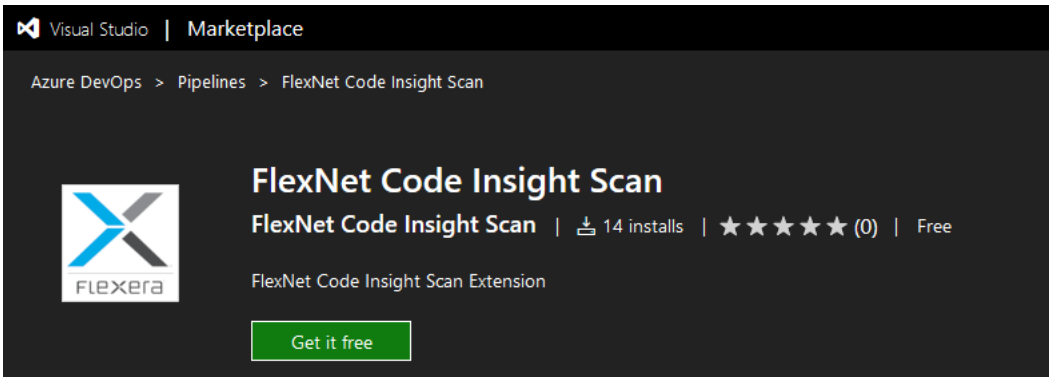
To obtain and install the Azure DevOps extension, perform the following steps.

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 **FlexNet Code Insight Scan** extension.



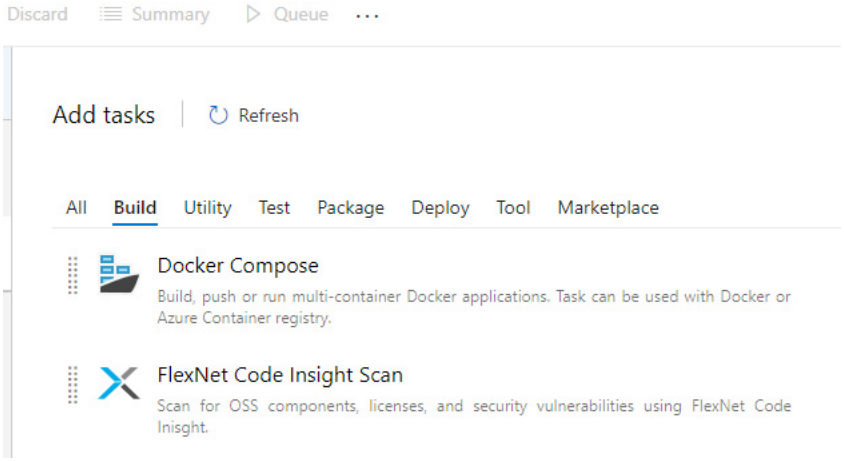
3. Download and install this extension into Azure DevOps.

Adding a Scan Task to Your Azure DevOps Agent Job

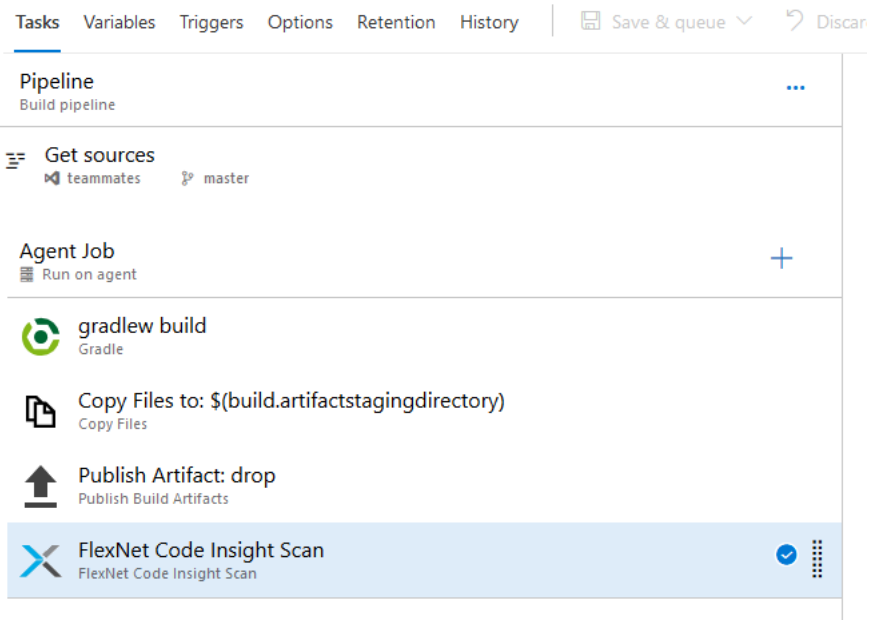
After the Azure DevOps extension has been installed, you need to add a Code Insight scan task to your Azure DevOps agent job so that the scan is automatically performed as part of your build process.

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

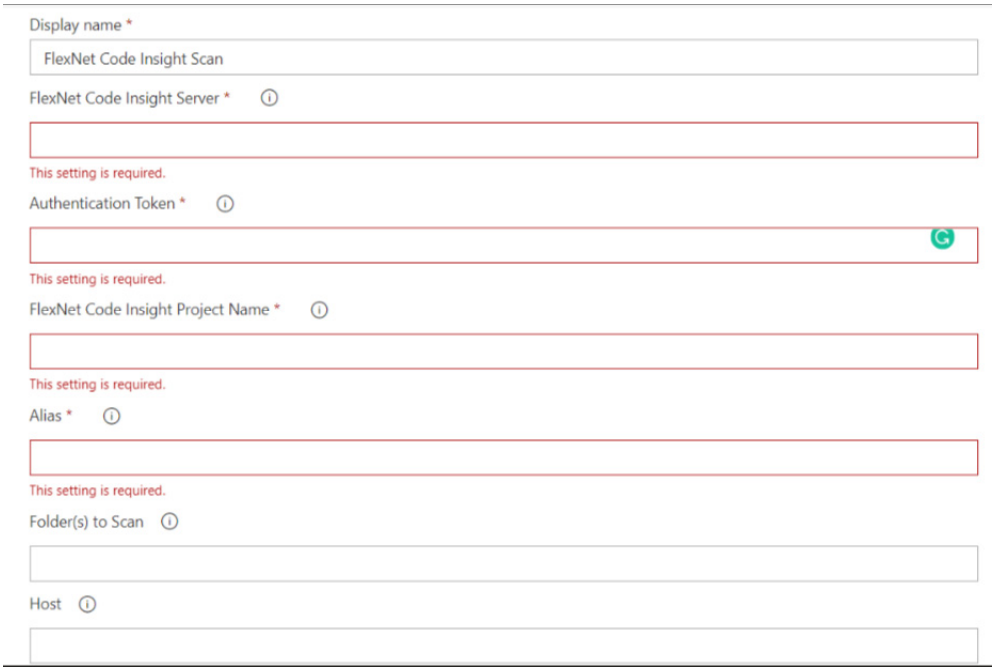
1. Create a build pipeline for your Azure DevOps project.
2. Locate the **FlexNet Code Insight Scan** task under the **Builds** section in the task catalog.



3. Add the **FlexNet Code Insight** task at any point after the build task has completed.



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



The following describes the task properties:

Field	Description
FlexNet 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.
FlexNet Code Insight Project Name	The name of the project that was created in the Code Insight user interface (for example, `ScanProject2_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.
Folder(s) to Scan	The folder containing the code to scan. Typically, you would use `$(build.artifactstagingdirectory)`, which is the location where the build output is staged during the build process.

Field	Description
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 2.0 Plugins.

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
  ```

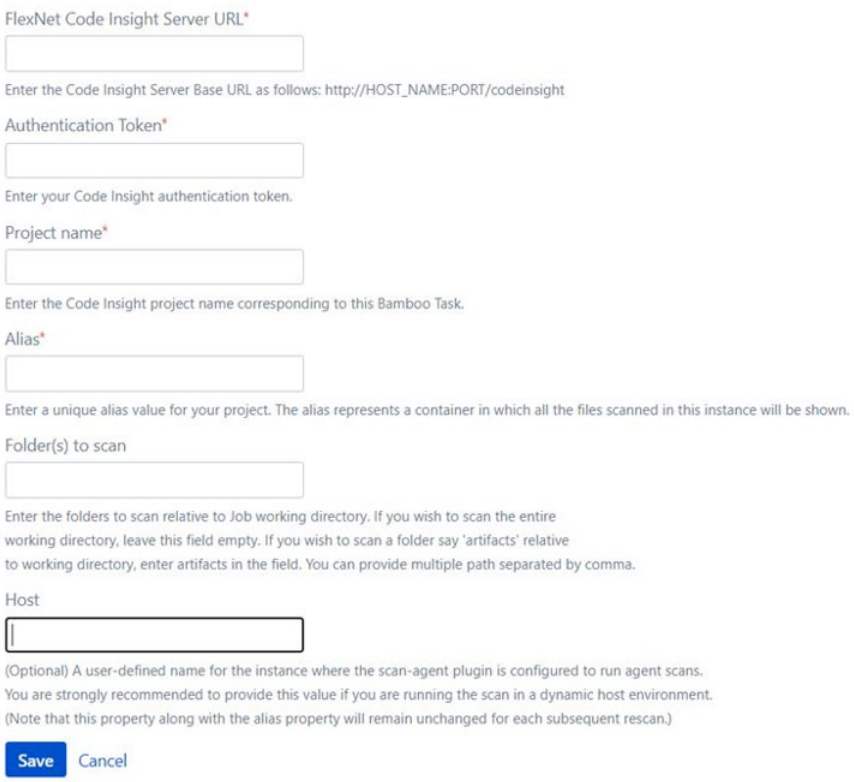
  Note that this is the recommended size value. However, heap size is relative to the size of the codebase. A large codebase requires this value to be increased 6GB or 8 GB.
- Event details for scans run on a Local Agent are logged to the `<BAMBOO_HOME>/logs/atlassian-bamboo.log` file. On a Bamboo Remote 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 `CodeInsight`*`version`*`Plugins.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.



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 2.0 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. Note that the scan requires the generic scan-agent plugin.

- Prerequisites for the GitLab Plugin
- Installing the Generic Scan Agent on GitLab Runner

- Configuring the CI/CD Pipeline to Run a Scan
- Executing the Build

Prerequisites for the GitLab Plugin

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

- A GitLab runner properly installed on Windows. (Refer to `https://docs.gitlab.com/runner/install/` 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 GitLab Runner).

Installing the Generic Scan Agent on GitLab Runner

Use these instructions to install the generic scan-agent plugin on the GitLab runner (installed on Windows).

Task ***To install the generic scan-agent plugin on the GitLab runner, do the following:***

1. Download and extract the contents of the `CodeInsight`*`version`*`Plugins.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 GitLab runner.
3. On the GitLab runner, update the following to match your environment:
   - The codebase root:

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

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

     ```
     cd C:\GitLab-Runner\GenericScanPlugin\example\bin
     ```

4. If you want the generic scan-agent plugin to detect transitive dependencies during scans, follow the procedure described in Enabling the Generic Scan-Agent Plugin to Detect Transitive Dependencies to configure this capability.

Configuring the CI/CD Pipeline to Run a Scan

To configure the CI/CD pipeline in your GitLab project to run a Code Insight scan, you need to edit your `.gitlab-ci.yml` file.

Task ***To edit the .gitlab-ci.yml file, do the following:***

Add the following contents to the file:

```
variables:
  CODEINSIGHT_SERVER: <CODEINSIGHT SERVER>
```

```
    AUTH_TOKEN:  <AUTH TOKEN>
    CODEINSIGHT_PROJECT: <CODEINSIGHT PROJECT NAME>

  codeinsight_scan:
    stage: test
    only:
      - mainline
    tags:
      - <tag for your GitLab-Runner>
    script:
      - cmd /Q /C C:\Gitlab-runner\GenericScanPlugin\example\bin\run_scan.bat %CODEINSIGHT_PROJECT%
%CODEINSIGHT_SERVER% %AUTH_TOKEN% %CI_PROJECT_DIR%
```

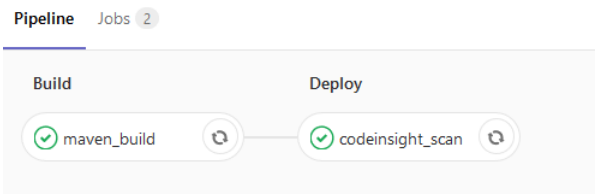
Replace the following variables with the appropriate information:

- **<CODEINSIGHT SERVER>**—The URL of the Code Insight Core Server (for example, `http://1.1.1.1:8888/codeinsight`).
- **<AUTH 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 store the scan results.
- **<tag for your GitLab-Runner>**—The tag for your GitLab runner.

`%CI_PROJECT_DIR%` is the GitLab variable for the project path where the code is built. You can replace it with the path of the folder containing the binaries of your built project.

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 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 plugin installation and configuration process proceeds in the following manner:

Phase 1—Address the prerequisites for the Jenkins plugin. See Prerequisites for the Jenkins Plugin.

Phase 2—Set the heap size. See Setting Heap Size for the Jenkins Plugin.

Phase 3—Set up the Jenkins plugin. See Setting Up the Code Insight Jenkins 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 Plugin

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

- The Jenkins 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 2.176.4 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.

Setting Heap Size for the Jenkins Plugin

The Jenkins 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.*

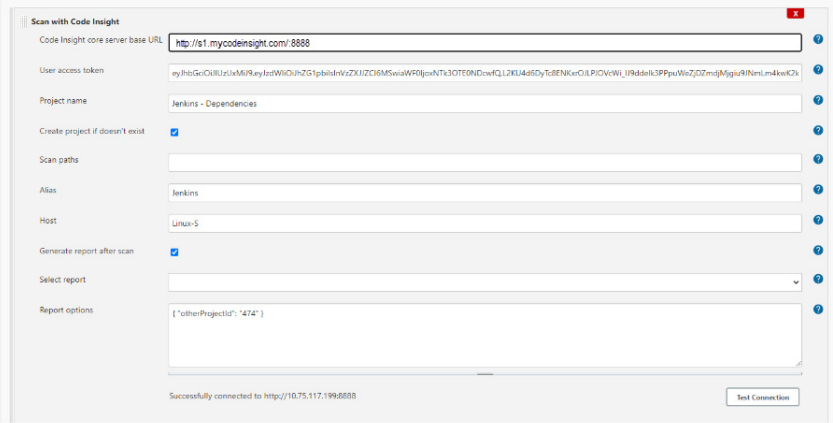
Setting Up the Code Insight Jenkins Plugin

The following procedure describes how to configure the Jenkins plugin.

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

1. Extract the Jenkins plugin from the `CodeInsight`*`version`*`Plugins.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** dropdown 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 2.0 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 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 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 Plugin and Setting Heap Size for the Jenkins Plugin.)

Configuring the Publication of a Code Insight Report in Jenkins

Optionally, the Jenkins 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 go to Code Insight to see the results. (The report is also available in Code Insight on the **Reports** tab for the project.)

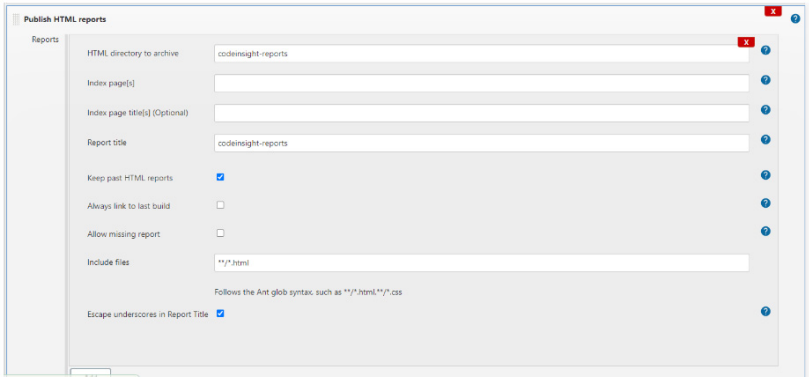
Viewing the Code Insight report as part of the build scan requires that the Jenkins HTML Publisher extension be installed in the Jenkins build environment. See Prerequisites for the Jenkins Plugin.

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 plugin (see Setting Up the Code Insight Jenkins 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 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** dropdown menu, and select **Schedule a Code Insight Scan**. The **Schedule a Code Insight Scan** dialog appears.
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 `CodeInsight`*`version`*`Plugins.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
     ```

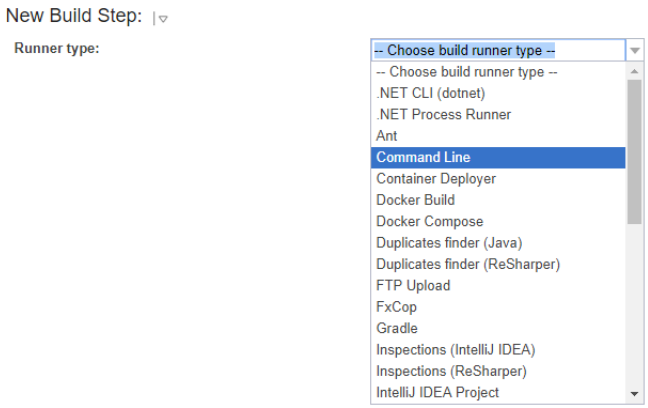
4. If you want the generic scan-agent plugin to detect transitive dependencies during scans, follow the procedure described in Enabling the Generic Scan-Agent Plugin to Detect Transitive Dependencies to configure this capability.

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**.



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.

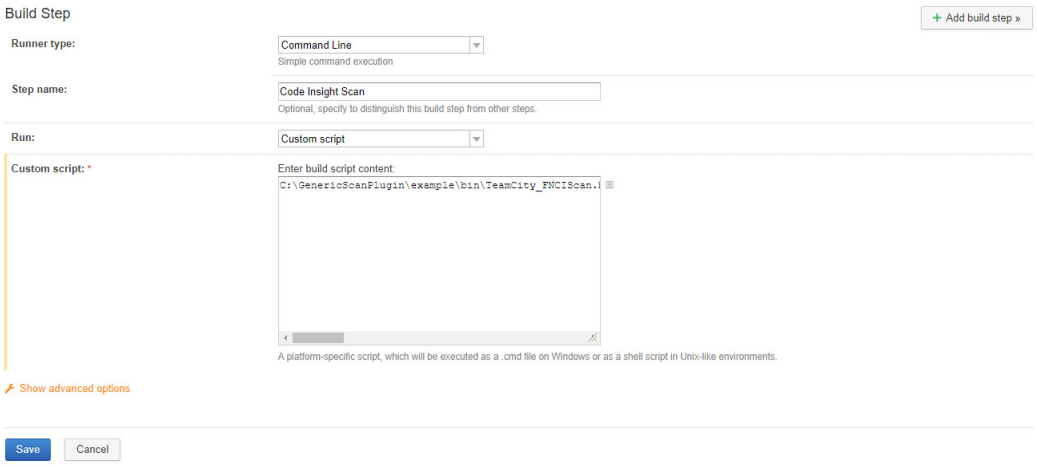
   b. In the **Run** field, select **Custom script**.

   c. In the **Custom script** field, provide the following:

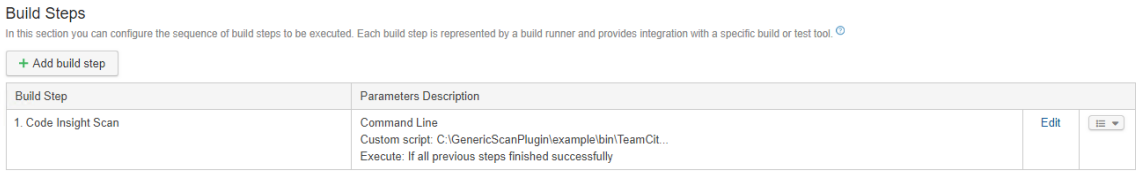
   ```
   C:\GenericScanPlugin\example\bin\TeamCity_FNCIScan.bat <CODEINSIGHT_PROJECT_NAME>
   <CODEINSIGHT_SERVER> <JWT_TOKEN> <SCAN_DIR>
   ```

   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.

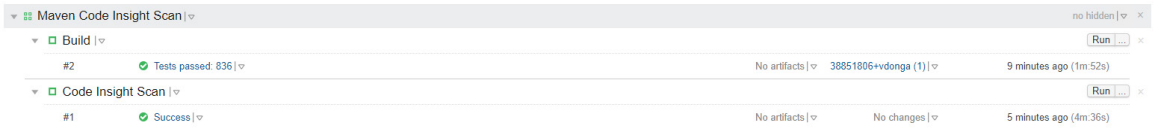


When complete, your build configuration should look like this:



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 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 `CodeInsight`*`version`*`Plugins.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>"
plugindescription="<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.eyJzdWIiOiJhZG1pbiIsInVzZXJJZCI6MSwia
  ```

  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 `CodeInsight`*`version`*`Plugins.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.eyJzdWIiOiJhZG1pbiIsInVzZXJJZCI6MSwia
```

     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 `CodeInsight`*`version`*`Plugins.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 2-2 ▪ 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 2-2 ▪ 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 *application project* being scanned. This name will appear, along with the Code Insight project name, in the **Last Scan** field on the 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 Binary Repositories

Currently, Code Insight support for scan integration with binary repositories includes the JFrog Artifactory Plugin.

JFrog Artifactory Plugin

JFrog Artifactory is a binary repository manager where third-party artifacts are stored. The Artifactory repository is centralized, so all developers use the same repository to access artifacts, which provides faster access, control, and security of binary artifacts. The Artifactory plugin provided by Code Insight scans an Artifactory repository and sends the results, as inventory, to the Code Insight server for review, management, and remediation through the user interface. Because Artifactory can contain several repositories, the plugin can support a multiple-repository scan, generating inventory for each repository in a separate Code Insight project.

The following topics describe how to install and use the Artifactory plugin:

- Prerequisites for the Artifactory Plugin
- Installing the Artifactory Plugin
- Scanning an Artifactory Repository Using a Cron Job
- Scanning an Artifactory Repository Using REST API
- Scan Results

Prerequisites for the Artifactory Plugin

Before installing and using the Artifactory plugin, ensure that the following prerequisites are met:

- Your site uses JFrog Artifactory PRO 5.x or higher.
- All Code Insight prerequisite tasks for plugins have been performed, as described in Preparing to Use the Plugins.
- You have write access for the `etc/plugins` directory on the Artifactory server. If you do not have access to that directory, be sure to obtain access before attempting to install the plugin.

Installing the Artifactory Plugin

The Artifactory plugin is available from the Product and License Center. Use the following steps to install the plugin.

Task ***To install the Artifactory plugin, do the following:***

1. Download and extract the Artifactory plugin subfolder from the `CodeInsight`*`version`*`Plugins.zip` file. For more information, see Downloading Plugins.
2. Copy the following directory and files into the `<ARTIFACTORY_HOME>/etc/plugins` directory on the Artifactory server:
   - `libs` directory
   - `code-insight-scan-plugin.groovy`
   - `code-insight-scan.plugin.props`
3. Define the properties in the `code-insight-scan.plugin.props` file:

```
repoKeys=<REPOSITORY_PATH1>/,<REPOSITORY_PATH2>
      codeinsight.server= http(s)://<HOST>:<PORT>/
      codeinsight.auth.token=Bearer <JWT_TOKEN>
      codeinsight.project.name= <PROJECT_NAME1>,<PROJECT_NAME2>
      plugin.root.path=<./artifactory-pro-5.10.2/etc/plugins>
      plugin.project.description= will be set by plugin, can be left blank
      isScanCronJobEnabled=disabled
      isPluginEnabled=enabled
      cronJobTime=1 * * * * ?
      artifactory_url= http(s)://<HOST>:<PORT>/ARTIFACTORY/
```

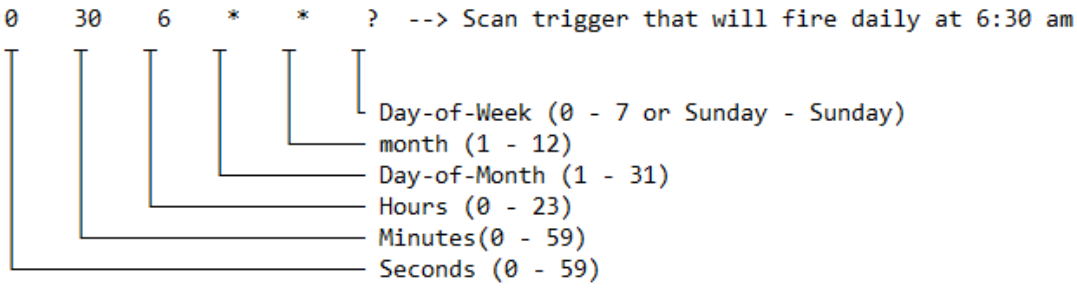
4. Determine if you want to execute the scan with a cron job or by calling REST API:
   - To execute a scan with a cron job, see Scanning an Artifactory Repository Using a Cron Job.
   - To execute a scan by calling REST API, see Scanning an Artifactory Repository Using REST API.

Scanning an Artifactory Repository Using a Cron Job

You can use the following procedure to schedule an Artifactory repository scan to run periodically.

Task ***To execute an Artifactory scan using a cron job, do the following:***

1. Open the `code-insight-scan.plugin.props` file.
2. Modify the property `isScanCronJobEnabled=disabled` to `isScanCronJobEnabled=`**`enabled`**.
3. Set the `cronJobTime` property to schedule the scan. Use the following diagram and the example it provides to help you set the property.



4. Restart the Artifactory server.

Scanning an Artifactory Repository Using REST API

You can call REST API to scan all Artifactory repositories listed in the `code-insight-scan.plugin.props` file or to scan a specific repository instead. The following topics describe how to scan repositories using REST API:

- Requirements When Using REST API to Scan Artifactory Repositories
- Scanning All Artifactory Repositories
- Scanning a Specific Artifactory Repository
- Reloading the Artifactory Plugin

Requirements When Using REST API to Scan Artifactory Repositories

The following lists the requirements for using REST APIs to scan Artifactory repositories.

Prerequisite for Scanning Repositories

As prerequisite for using REST API to scan Artifactory repositories, ensure that the properties in `code-insight-scan.plugin.props` are properly defined according to the instructions in Installing the Artifactory Plugin and according to any specific instructions listed in the procedures.

Required Option When Using the “https” Protocol

The REST API calls used in the next sections use the `http` protocol. To use the `https` protocol instead, be sure to include the option -k in the call:

```
curl -X POST -u<USER_NAME>:<PASSWORD> -k "https://<ARTIFACTORY_HOST>:8081/artifactory/api/plugins/
    execute/CodeInsightScan"
```

Scanning All Artifactory Repositories

The following command scans all repositories listed in the `code-insight-scan.plugin.props` file.

Task ***To scan all repositories, do the following:***

Use the following API call to scan all repositories:

```
curl -X POST -u<USER_NAME>:<PASSWORD> "http://<ARTIFACTORY_HOST>:8081/artifactory/api/plugins/
    execute/CodeInsightScan"
```

Scanning a Specific Artifactory Repository

The following procedure scans a specific repository.

Task ***To scan a specific repository, do the following:***

1. Ensure that the following properties are also defined in the `code-insight-scan.plugin.props` file: `codeinsight.server`, `Codeinsight.auth.token`, `plugin.root.path`, `isPluginEnabled`, and `artifactory_url`.
2. Use the following API call to scan the repository:

```
curl -X POST -u<USER_NAME>:<PASSWORD> "http://<ARTIFACTORY_HOST>:8081/artifactory/api/plugins/
    execute/
    CodeInsightSingleScan?params=repoKey=<REPOSITORY_NAME>%7cproject=<CODEINSIGHT_PROJECT_NAME>"
```

Reloading the Artifactory Plugin

If you have downloaded an updated version of the Code Insight plugin for Artifactory, you can use this REST API call to reload the plugin before running a scan:

```
curl -X POST --u<USER_NAME>:<PASSWORD> http://localhost:8081/artifactory/api/plugins/reload
```

Scan Results

When the scan completes, inventory is created in the corresponding Code Insight project. The **Scan Status** section on the **Summary** tab for the project provides information about the scan.

Similarly in Artifactory, information about the scan, such as the Code Insight project name, the scan status, and a link to the Code Insight project inventory are provided for each repository scanned. For more information about using plugins in Artifactory, see the following site:

`https://www.jfrog.com/confluence/display/RTF/User+Plugins`

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
- Installing the Docker Images Plugin
- Launching the Docker Images Plugin

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 plugin can only be executed on a server that already has an authenticated connection to the Docker server.

  ***Note** ▪ The Docker 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.

Installing the Docker Images Plugin

Use the following procedure to install the Docker Images plugin.

Task ***To install Docker Images plugin, do the following:***

1. Extract the Docker Images plugin subfolder from the `CodeInsight`*`version`*`Plugins.zip` file, and copy it the Docker server. For more information, see Downloading Plugins.
2. Open the `code-insight.docker.props` file in a text editor:

```
//required
codeinsight.server=http://127.0.0.1:8888
codeinsight.auth.token=Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsInVzZXJJZCI6MSwiaWF0IjoxNTExNDM1MTk4fQ.dHItJjJ2c89Dg5cVLvf
GR3fwJcR3yAlVE6k98dRZTdp3h6McDgv_PloVVE88eJ2GOG0tNDOnhU0ShDLUzdu3Pg
codeinsight.project.name=inv2
plugin.alias.name=scan-agent alias
plugin.root.path=/Users/ranimathur/Work/Scratch/
//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:
   - **codeinsight.server (required)**—The URL path to the Code Insight server.
   - **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.eyJzdWIiOiJhZG1pbiIsInVzZXJJZCI6MSwia
     ```

     For more information about generating this token, see Providing an Authorization Token.
   - **codeinsight.project.name (required)**—The name of the Code Insight project.
   - **codeinsight.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.root.path (required)**—The root path where the Docker plugin will be executing. This path must have writable privileges for the user executing the plugin.
   - **plugin.project.name (optional)**—A descriptive name to the project being scanned, that may be different from the project name specified in the Code Insight server. This text will appear on the **Summary** tab for the project in the Code Insight user interface.
   - **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 user interface.
   - **plugin.path.prefix (optional)**—The path prefix of the image being scanned. This prefix will be used to reference the file paths of the codebase on the **Project Inventory** page of the Code Insight GUI.

Launching the Docker Images Plugin

Use the following procedure to launch the Docker Images plugin to scan an image.

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

Task ***To launch the Docker Images plugin, do the following:***

Issue the following command to launch the Docker plugin from the command line:

```
% code-insight-docker-plugin.sh -image <DOCKER_IMAGE_NAME>
```

The `<DOCKER_IMAGE_NAME>` is the name given to the image that Code Insight is to scan.

Note ▪ *Only the downloaded Docker image is scanned.*

As it runs, the Docker plugin does the following:

- Contacts the Code Insight server to validate the connection and download a scanner.
- Extracts the Docker image.
- Scans the extracted Docker image contents.

The plugin sends the inventory results to Code Insight configured.

Generic Scan-Agent Plugin

The generic scan-agent plugin is an example ready-to-use plugin that is available in the `codeinsight-generic-version` toolkit installed with the standard Code Insight plugins. It can run as a standalone scan-agent plugin (as described in this section), enabling you to scan any file system instead of being restricted to specific Engineering systems, as you are with the standard scan-agent plugins. The plugin can also easily integrate with certain Engineering systems to perform scans as part of a build process, such a TeamCity or GitLab build process, or serve as a basis for developing your own scan-agent plugin (see the Developing Custom Plugins chapter).

The following describe the basics of using the generic scan-agent plugin as a standalone scan-agent:

- Prerequisites for the Generic Scan-Agent Plugin
- Running the Generic Scan-Agent Plugin
- Enabling the Generic Scan-Agent Plugin to Detect Transitive Dependencies

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 Java Runtime Environment JRE 8 or later
- 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.

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** ▪ 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.*

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

1. Download and extract the contents of the `CodeInsight`*`version`*`Plugins.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. If you want the generic scan-agent plugin to detect transitive dependencies during scans, follow the procedure described in Enabling the Generic Scan-Agent Plugin to Detect Transitive Dependencies to configure this capability.
4. To execute a scan using the plugin, run the following from a command line as a Java application:

   ```
   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”
   ```

   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.

  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 2.0 Plugins.

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.

Enabling the Generic Scan-Agent Plugin to Detect Transitive Dependencies

Use this procedure to enable the detection of transitive dependencies during scans performed by the generic scan-agent plugin. If you do not create and deploy the file described in this procedure, no transitive-dependency detection is performed.

Task ***To enable the generic scan-agent plugin to detect transitive dependencies, do the following:***

1. Create a text file whose content contains the following properties. These properties enable the various Code Insight analyzers to detect transitive dependencies during a scan:

   ```
   flx.enable.gradle.file.transitive.dependency=true
   flx.enable.npm.transitive.dependency=true
   flx.enable.nuget.transitive.dependency=true
   flx.enable.java.transitive.dependency=true
   ```

   If you do not want a specific analyzer to detect transitive dependencies, set the boolean value for its corresponding property to `false` (or remove the property from the file content). See the table below for more information.

2. Save the file as `codeaware.properties`.

3. Copy the file to the `code-insight-agent-sdk-generic/generic-plugin-binary` folder (where the `codeinsight-generic-build.jar` file is located).

Table 2-3 ▪ Properties Enabling the Generic Scan-Agent Plugin to Detect Transitive Dependencies During Scans

Package Analyzer	Property (with Boolean value) to Enable Transitive Dependency Detection
Gradle	`flx.enable.gradle.file.transitive.dependency={true | false}`
NPM	`flx.enable.npm.transitive.dependency={true | false}`
NuGet	`flx.enable.nuget.transitive.dependency={true | false}`
Java (pom.xml)	`flx.enable.java.transitive.dependency={true | false}`

Note About Rescans Performed by 2.0 Plugins

Starting with Code Insight version 2020 R3, scan agent plugins began support for an alias name. The 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 v2.0 of the 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 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.

3

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 `ExecutionContext.getInstance(...)` method.	**Methods:** `ExecutionContext getInstance(Properties props, PrintStream logger)` Initializes ExecutionContext for the current scan. **Parameters:** `props`: Properties required to scan and publish results `logger`: 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

Class	Description	Settings/Methods
ScanExecutor	Executes the scan and the publish results request made by client plugins. ***Note ▪*** *This class requires a valid* `ExecutionContext` *instance to be able to operate.*	**Methods:** • `ScanExecutor(ExecutionContext executionContext)` A constructor. • `String testConnection()` Validates the user authorization token and the connection to the Code Insight server. **Valid returned strings:** • `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`. **Parameters**: `paths`: list of absolute paths to be scanned **Return:** `SUCCESS` or `FAILURE` as a string. See the `logger` output stream for more details.

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: ● `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 2.0 Plugins.

Downloading the Scan Agent Toolkit

Use the following procedure to download and extract th 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 `CodeInsight`*`version`*`Plugins.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 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.