# Table of Contents

**DevOps**

- DevOps overview ................................................................. 7
- Install DevOps ........................................................................... 12
- Setting up DevOps tools ........................................................... 15
- Accelerating DevOps change .................................................... 155
- DevOps dashboard and Pipeline UI ........................................... 181
- User-created DevOps integrations ............................................ 205
- DevOps reference ................................................................. 240

**Index** ...................................................................................... a
DevOps

Use the ServiceNow® DevOps application with your DevOps toolchain to provide data insights, accelerate change, and increase visibility in your DevOps environment using a single system.

Achieve enterprise DevOps

Companies are turning to DevOps to speed up product delivery and innovation. DevOps is integral to delivering differentiation, improving customer responsiveness, and building competitive advantage.

In addition to Now Platform® core features, DevOps integrates with these ServiceNow applications:

- Now Intelligence Performance Analytics
- IT Service Management Change Management
- IT Business Management Agile Development
- Governance, Risk, and Compliance Risk Management

View and download the full infocard for a highlight of the DevOps features.
approval to continue deployment through the execution pipeline automatically.

### Evaluate and monitor DevOps process and results
Use the data-driven DevOps Insights dashboard to view change results reports, pipeline value stream, and evaluate the overall DevOps process.

### Visualize interactions and results across a pipeline execution
Use the DevOps Pipeline UI view to show pipeline stage progression and details for each app.

### Set up user-created integrations for additional planning, coding, and test tools
Integrate planning, coding, and test tools not included in the integrations provided with the DevOps application.

### Automate deployments by accelerating change

Change requests are automatically created for stages under change control. You can also enable automatic approval of DevOps changes using workflows, such as Change Approval Policy and DevOps Change Policy.
Evaluate and monitor DevOps process and results

Use the DevOps Insights dashboards to analyze operational and business reports and determine the overall efficiency and growth associated with your development processes.

Visualize interactions and results across a pipeline execution

Quickly view how everything is connected to see exactly what's happening with the pipeline and when. View the commits, the committers, and other details for a change request in one place.
Set up user-created integrations for additional planning, coding, and test tools

The DevOps application includes tool definitions for integrating some common planning, coding, and test tools. You can also set up user-created integrations for additional tools in your DevOps environment.

Get started

- Work with an implementation specialist to streamline your DevOps setup process. To learn more, visit the Customer Success Center.
- See the DevOps overview for information on how to request DevOps and begin setup.
- Take a DevOps course to learn about DevOps fundamentals and how to implement DevOps. To sign up, visit ServiceNow training and certification.

Applications and features

- DevOps

DevOps

Use the ServiceNow® DevOps application with your DevOps toolchain to provide data insights, accelerate change, and increase visibility in your DevOps environment using a single system.
Important:
Starting with version 1.31, the DevOps (sn_devops) application is renamed to DevOps Data Model. Download and install the DevOps Change Velocity application from the ServiceNow Store. The following dependencies are automatically installed or upgraded as part of the DevOps Change Velocity app:

- DevOps Data Model (previously called DevOps)
- DevOps Integrations
- DevOps Insights

For more information, see DevOps.

The DevOps application collects data across the entire set of lifecycle activities to provide visibility for DevOps teams so they can own the end-to-end process (plan, develop, build, test, deploy, and operate). Visibility across the entire DevOps toolchain, versus a single tool, exposes overall performance, including areas in the process that may need improvement.

As a result of this visibility, enterprise governance can be accelerated using DevOps, including automating change decisions.

ServiceNow DevOps Overview

To learn more about how your ServiceNow instance supports DevOps, see DevOps overview for a general understanding of the application and how it is used.

Start gaining insight using DevOps

Connect your tools and configure your pipeline to automate lifecycle activities. View pipeline stage progression and details for each app using the DevOps Pipeline UI. Monitor pipeline modeling and change acceleration benefits in your environment using the DevOps Insights dashboard.
• Connect your planning, coding, and orchestration tools, and model your pipeline

Tool support includes Azure DevOps Boards, Jira, and ServiceNow Agile Development 2.0 planning tools, Azure DevOps Repos, GitHub, Bitbucket Server (version 5.10.0), and GitLab SCM coding tools, and Azure DevOps Pipelines, Jenkins, and GitLab CI/CD orchestration tools.

• Automate deployments by accelerating change

Automatically create change requests at any stage for deployments that require change control in your environment, and use change approval policies to automate change request approval to continue deployment through the execution pipeline automatically.

• Evaluate and monitor DevOps process and results

Use the data-driven DevOps Insights dashboard to view change results reports, pipeline value stream, and evaluate the overall DevOps process.

• Visualize interactions and results across a pipeline execution

Use the DevOps Pipeline UI view to show pipeline stage progression and details for each app.

• Set up user-created integrations for additional planning, coding, and test tools

Integrate planning, coding, and test tools not included in the integrations provided with the DevOps application.

Understand DevOps concepts

• DevOps overview and getting started
• Install DevOps
• Setting up DevOps tools
• Accelerating DevOps change
• DevOps dashboard and Pipeline UI
• User-created DevOps integrations
• DevOps API

Find DevOps resources
• DevOps release notes
• Ask or answer questions in the Now Community

Get help from DevOps resources
• Search the Known Error Portal for known error articles
• Contact Customer Service and Support
• Upgrade to Quebec

DevOps overview
Set up DevOps roles, connections, integrate with external tools, then use the Insights dashboard to analyze operational and business reports and gain insight into your DevOps environment.

Use the change acceleration feature of DevOps to automatically create a change request for a stage in your development pipeline to accelerate change.

DevOps integrations

<table>
<thead>
<tr>
<th>Tool Type</th>
<th>Tool</th>
</tr>
</thead>
</table>
| Planning  | • Azure Boards (Azure DevOps Server 2020)  
           | • Jira (ver 7.x and 8.x)  
           | • ServiceNow Agile Development (ver 2.x) |
| Coding    | • Azure Repos (Azure DevOps Server 2020)  
<pre><code>       | • Bitbucket Server/Enterprise (ver 5.10.0) |
</code></pre>
<table>
<thead>
<tr>
<th>Tool Type</th>
<th>Tool</th>
</tr>
</thead>
</table>
| GitHub and GitHub Enterprise (ver 2.x) | ° Basic authentication  
                  ° OAuth           |
| GitLab SCM (ver 12.8.0)        |                                                                      |
| Orchestration   | ° Azure Pipelines (Azure DevOps Server 2020) Jobs supported:          |
|                 | ° Agent job                                                          |
|                 | ° Agentless (server) job                                             |
|                 | ° Jenkins (ver 2.138) Jobs supported:                                |
|                 | ° Freestyle project                                                  |
|                 | ° Folder (default is 3 levels)                                       |
|                 | ° Pipeline                                                          |
|                 | ° Multibranch Pipeline                                               |
|                 | ° GitLab CI/CD (ver 12.8.0) basic CI pipelines                      |

**DevOps applications**

The **DevOps application** uses these related DevOps applications:
- **DevOps Insights application**

  Use the **DevOps Insights dashboard** to manage and track results.
DevOps Integrations application

Use DevOps Integrations application for Azure DevOps integration with DevOps and Jenkins integration with DevOps (starting with version 1.12), and GitLab integration with DevOps (starting with version 1.16).

DevOps extensions

Use these extensions to model your pipeline in DevOps:

**ServiceNow DevOps extension for Azure DevOps**

Starting with version 1.12, the ServiceNow DevOps extension on Visual Studio Marketplace is required to integrate your Azure DevOps pipeline with ServiceNow DevOps.

**Jenkins plugin for ServiceNow DevOps**

A Jenkins plugin is provided to enable change acceleration so your orchestration tool can communicate with ServiceNow DevOps and control certain aspects of pipeline executions. Visit the Ancillary Software section on the ServiceNow Store website to download the Jenkins plugin for ServiceNow DevOps.

DevOps workflow

DevOps integrates with external planning, coding, and orchestration tools and automatically creates change requests at any stage for deployments that require change control.

Change approval policies can be used to automate change request approval to continue deployment through the execution pipeline automatically.

Performance and efficiency in your DevOps environment are monitored and analyzed using the DevOps Insights dashboard. Pipeline executions are visualized on the DevOps Pipeline UI view.

DevOps Concepts

These concepts are useful to understand with respect to the DevOps application.

**Pipeline**

A pipeline is a set of steps that begins with planning (plans for the work to be done). An app is the item being worked on, and the work is done via commits to a code repository. Once committed, an orchestration tool picks up the change and sends it through a series of pipeline steps, including production.
Pipeline steps can include quality checks, like functional, security, load, and behavioral tests as well as deployments, and infrastructure provisioning. The result is fully tested development features delivered as quickly as possible to production.

Integrations

The DevOps application integrates with external tools by exposing REST endpoints to receive webhook notifications, or direct REST calls from tools.

API

The DevOps application includes a DevOps API that allows integration with any coding, planning, or orchestration tools.

Getting started with DevOps

Get started using the DevOps application to integrate with your existing DevOps toolchain, manage processes, accelerate changes, and view operational and business insights.

Integrate with external planning, coding, orchestration, and testing tools. Out-of-the-box integrations collect lifecycle events and data from your existing DevOps toolchain.

Tool support includes Azure DevOps Boards, Jira, and ServiceNow Agile Development 2.0 planning tools, Azure DevOps Repos, GitHub, Bitbucket Server (version 5.10.0) and GitLab SCM coding tools, and Azure DevOps Pipelines, Jenkins, and GitLab CI/CD orchestration tools.

DevOps requirements

- ServiceNow IntegrationHub Runtime (com.glide.hub.integration.runtime) plugin
- ServiceNow IntegrationHub Action Step - REST (com.glide.hub.action_step.rest) plugin
- MID Server

Note: You must set up a MID Server for on-premise tools (for example, GitHub, Jira, Jenkins).

© 2021 ServiceNow, Inc. All rights reserved. ServiceNow, the ServiceNow logo, Now, and other ServiceNow marks are trademarks and/or registered trademarks of ServiceNow, Inc., in the United States and/or other countries. Other company names, product names, and logos may be trademarks of the respective companies with which they are associated.
DevOps integration and system users are automatically created in the DevOps application and are used to integrate with external tools.

- DevOps Integration User [devops.integration.user]

  Note: You must configure the password before the DevOps Integration User can configure a tool.

- DevOps System [devops.system] user

### DevOps roles

Before you can use the DevOps application, an admin user must assign DevOps roles.

- DevOps Administrator [sn_devops.admin] role is required for DevOps administration and setting up tools.
- DevOps integration [sn_devops.integration] role is required for DevOps inbound access to the integration tools in your environment.
- DevOps manager [sn_devops.manager] role is required for DevOps operation management, including monitoring performance in your DevOps environment.
- DevOps viewer [sn_devops.viewer] role is required to access the DevOps navigation menu and view DevOps information.

Note:

Starting from version 1.27, you need to include the sn_devops.integration' user role for read and write (create / update) operations on any custom ACLs that you may have configured, due to changes in our security model.

### Initial DevOps Setup

Once the DevOps application is installed and roles are assigned, complete the setup configuration.

System admin:
• Configure an HTTP connection in CreateDevOpsTool (DevOps connection and credential alias provided) to automatically connect to your tools
• Add admin credentials to CreateDevOpsTool to allow access to your tools

DevOps admin:
• Create a DevOps tool record (required to connect to and discover tool configuration)
  ◦ Application plans
  ◦ Repositories
  ◦ Orchestration tasks
  ◦ Pipelines
• Model your pipeline in DevOps (associating pipeline steps is required to receive notifications from your tool)
• Set up change acceleration
• Integrate test tools, if applicable

**View the DevOps Insights dashboard**
Use the DevOps Insights dashboard to manage and track results.

**View the DevOps Pipeline UI**
Use the DevOps Pipeline UI for a graphical view of pipeline executions.

**Install DevOps**
Install the DevOps application from ServiceNow Store applications. Visit the ServiceNow Store website to view all the available apps and for information about submitting requests to the store. For cumulative release notes information for all released apps, see the ServiceNow Store version history release notes.

**Before you begin**
Role required: admin
About this task

Note: For all new installations of the DevOps application on Quebec, you must set the `com.snc.change_management.change_model.type_compatibility` property to True.

To install the DevOps application on a subprod instance, navigate to the app on the ServiceNow Store and click Request Install to send your request.

Procedure

1. Navigate to System Applications > All Available Applications > All.
2. Click the Not Installed tab to view a list of applications available for installation.
3. Locate the DevOps application and click Install.
4. Optional: For Azure DevOps and Jenkins tool support (starting with version 1.12), and for GitLab tool support (starting with version 1.16), Install DevOps Integrations.

Install DevOps Integrations

For Azure DevOps and Jenkins tool support (starting with version 1.12), and for GitLab tool support (starting with version 1.16), install the DevOps Integrations application from ServiceNow Store applications. Visit the ServiceNow Store website to view all the available apps and for information about submitting requests to the store. For cumulative release notes information for all released apps, see the ServiceNow Store version history release notes.

Before you begin
Role required: admin

About this task
The DevOps Integrations application is required to integrate with these tools:

- Jenkins (starting with version 1.12)
- Azure DevOps (starting with version 1.12)
  - Azure Boards
  - Azure Repos
  - Azure Pipelines
- GitLab (starting with version 1.16)
GitLab SCM
GitLab CI/CD (basic CI pipelines)

⚠️ Note: You must Install DevOps before installing DevOps Integrations.

Procedure
1. Navigate to System Applications > All Available Applications > All.
2. Click the Not Installed tab to view a list of applications available for installation.
3. Locate the DevOps Integrations application and click Install.

Install DevOps Insights
Install the DevOps Insights Standard dashboard application from ServiceNow Store applications. Visit the ServiceNow Store website to view all the available apps and for information about submitting requests to the store. For cumulative release notes information for all released apps, see the ServiceNow Store version history release notes.

Before you begin

⚠️ Note: You must install the Service Portfolio Management Foundation (com.snc.service_portfolio) plugin before installing the DevOps Insights application to see demo data.

Role required: admin

About this task
The DevOps Insights application uses Performance Analytics to track performance metrics over time.

⚠️ Note: Starting with version 1.12, the DevOps Insights Standard dashboard is the sole dashboard for the DevOps application. You must install the DevOps Insights application to view the Insights module.

Procedure
1. Navigate to System Applications > All Available Applications > All.
2. Click the Not Installed tab to view a list of applications available for installation.
3. Locate the DevOps Insights application and click Install.
4. For new DevOps Insights installations, if you have already been using the DevOps app and you Installed DevOps Insights at a later date, run the [DevOps] Historical Data Collection job to collect historical Insights data.

   Note: This job is a one-time job to collect historical DevOps data, and not meant to be on a schedule. It might take awhile, so plan on running this job during a period of low usage.

   a. Navigate to Performance Analytics > Data Collector > Jobs and open the [DevOps] Historical Data Collection record.
   b. Select Active and set the Run As credentials to System Administrator.
   c. Click Execute Now.

5. Starting with version 1.14, unless the [DevOps] Daily Data Collection job was previously customized, set it up to collect daily Insights data.

   Note: This is a scheduled job (to be run on a regular basis) to collect daily DevOps data. For optimal performance, set this job to run during periods of low usage.

   b. Select Active and set the Run As credentials to System Administrator.
   c. Set the run time of day to a period of low usage.
   d. Click Execute Now.

   The DevOps > Insights module is shown with the Standard dashboard within it.

**Setting up DevOps tools**

Set up your environment to connect DevOps to your tools, discover tool configuration, and import historical data into the DevOps application.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Required DevOps applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azure DevOps</td>
<td>DevOps application and DevOps Integrations application</td>
</tr>
<tr>
<td>• Azure Boards</td>
<td></td>
</tr>
<tr>
<td>• Azure Repos</td>
<td></td>
</tr>
<tr>
<td>• Azure Pipelines</td>
<td></td>
</tr>
<tr>
<td>Bitbucket Server/Enterprise</td>
<td>DevOps application</td>
</tr>
<tr>
<td>Tool</td>
<td>Required DevOps applications</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>GitHub and GitHub Enterprise</td>
<td>DevOps application</td>
</tr>
<tr>
<td>GitLab</td>
<td>DevOps application and DevOps Integrations application</td>
</tr>
<tr>
<td>• SCM</td>
<td></td>
</tr>
<tr>
<td>• CI/CD</td>
<td></td>
</tr>
<tr>
<td>Jenkins</td>
<td>DevOps application and DevOps Integrations application</td>
</tr>
<tr>
<td>Jira</td>
<td>DevOps application</td>
</tr>
<tr>
<td>ServiceNow Agile Development 2.x</td>
<td>DevOps application</td>
</tr>
</tbody>
</table>

**Connection and credential alias**

Before you set up your tool records in DevOps, your admin configures the DevOps connection and credential alias (CreateDevOpsTool) to gain access to the tools in your environment.

⚠️ **Note:** You only need to configure the CreateDevOpsTool connection and credential alias once to set up all of your tools in DevOps.

**Tool actions**

A common set of actions is required to integrate DevOps with each tool in your environment.
<table>
<thead>
<tr>
<th>Action</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>DevOps accesses the tool with correct credentials and gets the webhook URL.</td>
</tr>
<tr>
<td>Discover</td>
<td>DevOps discovers all tool information.</td>
</tr>
<tr>
<td></td>
<td>• Planning tool application plans</td>
</tr>
<tr>
<td></td>
<td>• Coding tool repositories</td>
</tr>
<tr>
<td></td>
<td>• Orchestration tool orchestration tasks and pipelines</td>
</tr>
<tr>
<td>Configure</td>
<td>DevOps configures the webhook URL in the source tool so notifications from the tool can be received by DevOps.</td>
</tr>
<tr>
<td>Import</td>
<td>DevOps imports all tool data and enables tracking.</td>
</tr>
<tr>
<td></td>
<td>• Planning tool application plan work item data (and plan versions, features)</td>
</tr>
<tr>
<td></td>
<td>• Coding tool repository branch and commit data</td>
</tr>
<tr>
<td></td>
<td>• Orchestration tool task execution data</td>
</tr>
</tbody>
</table>

**DevOps Retry Policy**

Starting with version 1.19, enable the DevOps Custom HTTP Retry Policy for most tool communication from ServiceNow flows to automatically retry failed requests when a step encounters an intermittent issue, such as a network failure or request rate limit.

Navigate to the action in **Flow Designer > Designer > Actions** and in the REST step, select **Enable Retry Policy**. For the DevOps Custom HTTP Retry Policy to take effect, override the default policy for the alias and select the DevOps retry policy.

To customize the retry configuration, access the DevOps Custom HTTP Retry Policy settings in **IntegrationHub > Retry Policy**.
Manual configuration mode

As an alternative to making a connection using the standard setup process, you can use manual configuration mode to set up a webhook manually.

For example, if you do not have admin privileges for a tool (to allow automatic configuration the webhook URL), you can send an email to the admin of the tool requesting the ServiceNow instance be added to the webhook. Once the instance is added, you can Enter Manual Configuration Mode and change the Connection state field to Connected (to connect manually).

This way you only need read-only permission to the tool. Once the connection is made, click Exit Manual Configuration Mode.

Note: The Connection state field can only be edited in manual configuration mode.

All planning, coding, and orchestration tool connections support manual configuration mode.

Note: Connection state automatically changes to disconnected if there is a change in configuration, such as URL, credentials, alias, type, or MID Server settings, for example.

Commits and task executions

Run commits in DevOps are associated to a task execution.

Note: For a commit to show up as a run commit, the commit record must exist in ServiceNow prior to the job/pipeline run.

In the event that jobs are rerun on the same commit, these conditions apply.

- Azure DevOps does not show any run commits.
- GitLab displays only the last commit as a run commit.
- Jenkins displays only the last commit as a run commit on which it was run. The difference of all commits is not shown.

Multiple commits in a single payload (commit arrays) have these limitations.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Max commits per payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azure DevOps</td>
<td>25</td>
</tr>
<tr>
<td>GitHub</td>
<td>1000</td>
</tr>
<tr>
<td>GitLab</td>
<td>20</td>
</tr>
</tbody>
</table>
Associating multiple work items to a commit

Starting with version 1.22, multiple work items for a commit are supported in DevOps for Azure DevOps, Bitbucket, GitHub, and GitLab.

Work item syntax in the commit message can be customized to reflect the processes in your organization using the DevopsCommitMessageParserSNC script included in the System Definition > Script Includes module.

Linking work items to a commit using the Azure DevOps user interface is also supported in DevOps.

You can view the list of associated work items in the DevOps Commit record, and in the Pipeline UI.

Deleting DevOps records

Starting with version 1.21, cascade record deletion is implemented to delete all dependent lower level DevOps records whenever a parent or higher level DevOps entity is deleted.

Note: Deleting a parent record can trigger the deletion of many child records. See the DevOps record deletion cascade section for more details.

For example, when a Plan record is deleted, all dependent Work Item, Plan Version, and many-to-many relation (like App to Plan, and Work Item to Plan Version) records are deleted.
Create new DevOps tool from the Tools list view

Create a default connection alias, while creating a DevOps tool from the Tools list view. Use base system configuration templates to create connections and credentials under the child connection alias that is created. Eliminate creating a connection alias as a prerequisite before creating a new DevOps tool.

Before you begin

ℹ️ Note: This feature works on the Paris release or higher.

Role required: sn_devops.admin

About this task

Starting from the release 1.26, the DevOps application provides base system parent aliases and the following corresponding connection templates:

- Basic Auth
- API Key
- OAuth- Authorization Code

You can use these templates to create connection and credential under the child connection alias that is created while creating the new tool. By default, a new child connection alias is automatically created when you create a new tool. This eliminates the need to create a new alias or use an existing connection alias.

Since a new child alias is created, you do not need to use the base system child aliases that are seeded as part of the DevOps application. This way you do not use up all existing aliases as a new alias is created every time you create a DevOps tool. Based on the tool you select in the form, a corresponding authentication mechanism and Connections and Credentials template displays. This template based approach also simplifies the process to create certain connection protocols such as OAuth 2.0. For more information see, Create a Connection and Credential alias

Procedure

1. Navigate to a Tools list view. For instance, DevOps > Orchestration Tools.
2. Click New in the header menu.
3. On the form, fill in the fields.
Create new tool

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter a name for the tool you're creating.</td>
</tr>
<tr>
<td>Tool</td>
<td>Click the Search icon to select the tool from the lookup list.</td>
</tr>
<tr>
<td>Connection and Credential</td>
<td>Click the Search icon to select the tool from the lookup list or create a</td>
</tr>
<tr>
<td>Template</td>
<td>template.</td>
</tr>
<tr>
<td></td>
<td>• The options in the lookup list depend on the tool you select.</td>
</tr>
<tr>
<td></td>
<td>• This field does not appear if you select an agile tool integration.</td>
</tr>
</tbody>
</table>

4. **Click Submit.**
   The new tool record is created and the tools list view displays.

5. **Open the tool record you just created.**
   Creating a child alias after creating a tool may take some time. If the connection alias is not yet created the following information message displays.

   **Important:** No Connection Alias created for the tool. If the tool was just created, it can take few seconds for alias to be created.

The **Connection Alias** field is auto-populated with the child alias that is created during tool creation.

6. **Click the Create New Connection & Credential related link.**
   Based on the tool and the template you select, a relevant Create Connection and Credential form displays.

7. **On the form, fill in the fields and create the connection.**

**Results**
You have successfully created a connection alias while creating a new DevOps tool.
Example:

OAuth 2.0 Connection and Credential template form from the new tool form

Create Connection and Credential

Please Enter the Connection Information

* Connection Name:
  test1

* Connection URL:
  https://<provider-domain-name>.com

Use Mid:
false

Please Enter the Credential Information

* Credential Name:
  test1

* OAuth Client ID:
  Client ID for provider

* OAuth Client Secret:
  Client Secret for provider

* OAuth Redirect URL:
  https://d1.service-now.com/oauth_redirect.do

* Auth URL:
  https://<provider-name>.com/login/oauth/authorize

* Token URL:
  https://<provider-name>.com/login/oauth/access_token
Onboarding DevOps tools and apps using self service catalog

Use the onboarding catalog items as a self-service approach to add your tools and apps to the service catalog.

Use the ServiceNow Service Catalog to request the onboarding of tools and apps. Fill out the onboarding form details for a tool or an app and submit the request.

Prior to the request creation, an automated workflow approves or rejects the request.

- When the workflow is approved, a success message displays.
- When rejected, an inbound event is created capturing error logs. By ensuring the log is error free, you can create a request again.

Starting with version 1.31, you can onboard DevOps tools and apps using self-service catalogs.

- Set the Active state to True to activate the DevOps Tool Onboarding maintain item. By default the catalog items (App Onboarding) is disabled.
- Activate the Request for onboarding approval flow from the Flow Designer, for the tool onboarding requests to be approved by default.
- Customize roles for approvals of the request for onboarding approval flow from the flow designer.

Activate and configure DevOps tool onboarding

Use the service catalog to onboard DevOps tools.

Before you begin

- Upgrade to ServiceNow DevOps version 1.31.

Role required: sn_devops.admin

About this task

Supported tools:

- Sonar
- BitBucket
- GitHub
- Jenkins
- Jira
• Azure
• Agile

Note: GitLab is not supported for this version.

Procedure
1. Navigate to Service catalog > Catalog Definitions > Maintain Items > DevOps Tool Onboarding > Activate.
2. Navigate to DevOps Tool Onboarding > Available For, select the users you want to provide access to the onboarding requests. Similarly, use the Not Available For related list to restrict the user access. This allows the users with specified roles to have access to the onboarding tool while creating tool requests.

Results
You have successfully activated and configured the DevOps tool onboard catalog item.

What to do next
Starting with version 1.31, self-service onboarding of tools and apps is supported.
• By default the catalog items shipped with 1.31 are in inactive state, you need to activate the catalog items for DevOps tool onboarding.
• By default the Request for onboarding approval is available from the Flow Designer for DevOps tool onboarding requests.
• You can also configure the tools to self-service catalog.

Create a tool onboarding request
Use the service catalog to create an onboarding request of a tool.

Before you begin
• Upgrade to ServiceNow DevOps version 1.31.

Role required: sn_devops.admin

Procedure
1. Navigate to Self-Service > Service Catalog > DevOps Tool Onboarding.
2. On the form, fill in the fields.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool Name</td>
<td>Enter a name for the tool you are creating.</td>
</tr>
<tr>
<td>Tool Integration</td>
<td>Select the supported active tool.</td>
</tr>
<tr>
<td>Tool URL</td>
<td>Enter the selected devops tool URL in the appropriate format:</td>
</tr>
<tr>
<td></td>
<td>For example: <a href="https://sonarqube.sbdevops.xyz">https://sonarqube.sbdevops.xyz</a></td>
</tr>
<tr>
<td>Tool password/Access</td>
<td>Tool password or Personal access token or OAuth 2.0</td>
</tr>
<tr>
<td>token</td>
<td></td>
</tr>
<tr>
<td>Credential Type</td>
<td>OAuth 2.0</td>
</tr>
<tr>
<td></td>
<td>Preferred credential type for GitHub alone.</td>
</tr>
<tr>
<td>Tool Credential</td>
<td>Enter the OAuth 2.0 credentials</td>
</tr>
<tr>
<td>Integration User Name</td>
<td>Select an integration user from the list of assigned users.</td>
</tr>
<tr>
<td>Integration user Password</td>
<td>Password of the selected integration user.</td>
</tr>
</tbody>
</table>

3. Click **Order Now**. A request item is created for the placed order.

**What to do next**
- You can verify the newly created tool at DevOps > **Tools**.
- Ensure the flow approver has access to the sc_request table. This will inform the flow approvals context.
- The onboarding requester can configure the **Comments and work notes** field of the sc_request table to view the approver comments. This also displays tool registration success and failure notifications.

**Activate and configure DevOps app onboarding**
Use the service catalog app onboarding item to add pipelines, repositories, and plans for orchestration, coding, and planning tools respectively.

**Before you begin**
- Upgrade to ServiceNow DevOps version 1.31.

Role required: sn_devops.admin
About this task
Supported tools:
• Sonar
• BitBucket
• GitHub
• Jenkins
• Jira
• Azure
• Agile

⚠️ Note: GitLab is not supported for this version.

Procedure
1. Navigate to Service catalog > Catalog Definitions > Maintain Items > DevOps App Onboarding > Activate.
2. Navigate to DevOps App Onboarding > Available For, select the users you want to provide access to the onboarding requests. Similarly, use the Not Available For related list to restrict the user access.
   This allows the users with specified roles to have access to the app onboarding tool while creating app requests.

Results
You have successfully activated and configured the DevOps app onboard catalog item.

What to do next
Starting with version 1.31, self-service onboarding of tools and apps is supported.
• By default the catalog items shipped with 1.31 are in inactive state, you need to activate the catalog items for DevOps app onboarding.
• By default the Request for onboarding approval is available from the Flow Designer for DevOps app onboarding requests.
• You can also configure the tools to self-service catalog.

Create an app onboarding request
Use the service catalog to create an onboarding request of an app.

Before you begin
• Upgrade to ServiceNow DevOps version 1.31.
Role required: sn_devops.admin

Procedure
1. Navigate to **Self-Service > Service Catalog > DevOps App Onboarding**.
2. On the form, fill in the details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Options</th>
</tr>
</thead>
</table>
| Are you creating a new app or adding to an existing one? | • Add an existing app if you have any.  
• Create a new app. |
| App                                        | Enter a name to the app.                                                |
| Onboarding Pipelines                       | Select an existing orchestration tool.                                  |
| Pipelines                                  | Select and move the pipeline from the available list to the selected list.  
The field appears when the orchestration tool is chosen. |
| Onboarding Repositories                    | Select an existing coding tool.                                         |
| Repositories                               | Select and move the repository from the available list to the selected list.  
The field appears when the coding tool is chosen. |
| Onboarding Plans                           | Select an existing planning tool.                                       |
| Plans                                      | Select and move the plan from the available list to the selected list.  
The field appears when the planning tool is chosen. |

ℹ **Note:** You can request onboarding for a single or all categories from the form.

3. Click **Order Now**.
   All the added pipeline, repository, and plans are mapped with the application. A request item is created for the placed order. You can verify the newly created tool at **DevOps > Tools**.
   If the request item is failed or rejected, you can verify the error logs at **DevOps > Administration > Inbound events**.
What to do next

- You can verify the newly created tool at DevOps > Apps and Pipelines.
- Ensure the flow approver has access to the sc_request table. This will inform the flow approvals context.
- The onboarding requester can configure the Comments and work notes field of the sc_request table to view the approver comments. This also displays tool registration success and failure notifications.

Renaming integration objects

Rename a repository or planning item without duplicating the objects in the coding or the planning tools.

In previous versions, when a repository or planning item is renamed externally, the integration objects are created as new objects in ServiceNow DevOps. Starting from version 1.31, ServiceNow DevOps supports renaming of the objects without creating duplicate objects. The objects are represented by a Native ID to check for duplication before creating or updating a repository or a planning item.

The following coding tools are supported:
- GitHub
- GitHub Enterprise
- BitBucket
- Azure DevOps
- GitLab

The following pipeline tools are supported:
- Azure DevOps
- GitLab

Renaming of Jenkins pipeline objects cannot be done, as it doesn't populate a Native ID. In case you cannot avoid renaming, you are required to manually rename all the affected objects to continue with the same pipeline. Otherwise, a new pipeline is created where in the historical data of the renamed pipeline cannot be associated with the new pipeline.

Rename the coding tools

Rename the repository
Before you begin

- Upgrade to ServiceNow DevOps version 1.31.

Role required: sn_devops.admin

About this task

Starting with version 1.31, a unique identifier called **Native ID** is introduced to the repository table for all the supported coding tools. The native ID acts as an identifier whenever an object is renamed.

Advantages of renaming:

- When a repository is renamed, the historical data from past 9 months reflects the rename changes. Following URLs are considered:
  - commitURL
  - commitAPIURL
  - TagURL
  - BranchURL

Following scenarios may result in creating duplicate objects:

- Always run **Discover** after import, configure, or upgrade actions on a repository.
- Avoid renaming an object before or during an upgrade.
- GitHub does not support branch renaming.

Procedure

1. Navigate to **Tools > Coding Tools**.
2. Select a coding tool.
3. In the **Repositories** related link, rename the name of the repository, and then click **Discover**.

   **Note:** Always **Discover** after renaming a repository. This ensures the new name is reflected in all the related objects of the repository.

4. **Optional:** You can also rename a repository in the source tool. In such cases, you must run **Discover** in ServiceNowDevOps for the rename changes to reflect for all affected objects.

Results

Repository is renamed successfully.
Rename the orchestration tools

Rename the pipelines of an orchestration tool to avoid creating duplicate pipelines.

Before you begin

- Upgrade to ServiceNow DevOps version 1.31.

Role required: sn_devops.admin

About this task

Starting with version 1.31, a unique identifier called Native ID is introduced to the pipeline table for all the supported orchestration tools. The native ID acts as an identifier whenever an object is renamed.

Advantages of renaming:

- When a pipeline is renamed, the historical data from past 9 months reflects the rename changes. Following URLs are considered:
  - PipelineURL
  - Task Execution URL

Following scenarios may result in creating duplicate objects:

- Always run Discover after import, configure, or upgrade actions on a pipeline.
- Avoid renaming a pipeline object before or during an upgrade.
- GitHub does not support branch renaming.

Procedure

1. Navigate to Tools > Pipeline Tools.
2. Select a pipeline tool.
3. In the Pipelines related link, rename the pipeline, and then run Discover.

   ❗ Note: Always Discover after renaming a pipeline. This ensures the new name is reflected in all the related objects.

4. Optional: You can also rename a repository in the source tool. In such cases, you must run Discover in ServiceNow DevOps for the rename changes to reflect for all affected objects.

Results

Repository is renamed successfully.
Azure DevOps integration with DevOps

Integrate Azure DevOps tools with DevOps using the DevOps Integrations application.

💡 Note: You must install the DevOps Integrations application to integrate Azure DevOps tools with DevOps.

Azure DevOps tools supported:

- Azure Boards (planning)
- Azure Repos (coding)
- Azure Pipelines (orchestration)
  - Build (CI) pipelines - agent and agentless (server) jobs
  - Release (CD) pipelines

Starting with version 1.24, capture tags from Azure DevOps coding tool commits.

Azure DevOps extension

You can use the ServiceNow DevOps extension for Azure DevOps on Visual Studio Marketplace to integrate your Azure pipeline with the ServiceNowDevOps application.

The ServiceNow DevOps extension includes:

- ServiceNow DevOps service connection
- ServiceNow DevOps Release Gate (starting with version 1.18)
- Azure build (CI) pipeline agent and server job custom tasks

Set DevOps change request details

Starting with version 1.24, set closure code and change request fields from within the Azure pipeline.

Bulk commits in Azure DevOps

Starting with version 1.23, large commits are supported with Azure DevOps.

To support large commits, perform these actions:
• Install the ServiceNow IntegrationHub Action Template - Data Stream (com.glide.hub.action_type.datastream) plugin.

• For optimal performance, disable flow logging by setting the Flow Designer com.snc.process_flow.reporting.level property to Off.

• For MID Server settings, view the MID Server support for Data Stream actions section.

The Azure DevOps data stream can currently process up to 8000-9000 commits per code push. The number of run commits listed for a task execution are limited to 200.

**Azure DevOps test tool integration**

Starting with version 1.15, JUnit test type integration is supported for Azure DevOps.

**Test tool integration** lets you view test results in DevOps for Azure DevOps unit, functional, and performance tests.

**Connect DevOps to Azure DevOps tools**

Configure a connection and credential alias and create an Azure DevOps tool record in DevOps.

⚠️ **Note:** You can create a new tool and simultaneously create a new connection alias. Since the child alias is created during tool creation, you do not need an admin role nor do you need to create a new connection alias as a prerequisite. Create the new connection alias as an sn_devops.admin user using **Connections and Credentials template** which are dynamically populated based on the tool you select. For more information, see Create new DevOps tool from the Tools list view.

**Configure connection and credential alias - Azure DevOps**

Before you set up a tool record in DevOps, your admin configures the DevOps CreateDevOpsTool connection and credential alias to allow access to the tools in your environment.

**Before you begin**

Starting with version 1.15, DevOps connection and credential aliases (DevOpsAlias1 through DevOpsAlias10) are used to connect automatically when you set up your DevOps planning, coding, and orchestration tools.

⚠️ **Note:** DevOpsAlias aliases take the place of the sn_devops.GitHub, sn_devops.Jenkins, sn_devops.JIRA, sn_devops.BitBucket aliases in the tool setup.
To connect to more than 10 tools, or if you receive an error saying all DevOpsAlias connection and credential aliases are being used, an admin can create additional aliases in the `sn_devops` application scope.

**Note:** If additional sys_alias records for DevOps are needed, they must be created by an admin in the `sn_devops` application scope.

You can reuse a DevOpsAlias alias no longer in use by inactivating the HTTP connection, if needed.

Role required: admin, sn_devops.admin

About this task
Your admin must configure an HTTP connection in the CreateDevOpsTool connection and credential alias provided, and add admin credentials to connect automatically when you set up your DevOps planning, coding, and orchestration tools:

- **URL:** `https://<instance name>.service-now.com/

- **Credentials:** admin

**Note:** Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

You only need to configure the CreateDevOpsTool connection and credential alias once to set up all of your tools in DevOps.

The CreateDevOpsTool connection and credential alias is used with the DevOps Create a Tool subflow in Flow Designer. You can view executions in Flow Designer for information regarding flow results.

Procedure

1. Navigate to **Connections & Credentials > Connection & Credential Aliases** and open the CreateDevOpsTool record.

2. In the Connections related list, create a record and enter a **Name** for the connection.

3. On the Connection form, click the **Credential** field lookup list, and then click **New** to create an admin credential.
a. Click Basic Auth Credentials and enter a Name.

b. Enter admin username and password (required to access the tools in your DevOps environment).

Note: Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

4. On the Connection form, enter https://<instance name>.service-now.com/ for the Connection URL.

Create an Azure DevOps tool record in DevOps
Create an Azure DevOps tool record in DevOps to connect, discover, and import Azure DevOps tool data.

Before you begin
Role required: sn_devops.admin

About this task
Actions:
• **Connect** to Azure DevOps and get the webhook URL when you submit a DevOps tool record.
• **Discover** plans, repositories, orchestration tasks, and pipelines.
• **Configure** webhooks in Azure DevOps.
• **Import** work item, plan version, and feature records, branch and commit records, and task execution and step execution records.

To customize Azure Boards import of work item states or types, use the DevOpsAzureDevOpsWorkItemHelper script include.

Procedure
1. Create a tool record in DevOps to automatically connect to Azure DevOps and get the webhook URL.

Note: You need at least two unused connection and credential aliases to create an Azure DevOps tool (one for a build pipeline and one for a release pipeline).

a. Navigate to DevOps > Tools > Create New and create a record.

b. Enter a Tool Name and fill in the tool details.
<table>
<thead>
<tr>
<th>Tool Integration</th>
<th>Azure DevOps</th>
</tr>
</thead>
</table>
| Tool URL         | Azure DevOps tool URL in the format:  
https://dev.azure.com/organization/project |
| Tool Username    | Azure DevOps user name |
| Access Token     | Personal access token (PAT) |

**Note:** Only a personal access token is supported.

When you generate a Personal access token (PAT) for Azure DevOps, you must select the scopes to authorize if you are not granting complete access.

**Note:** MID Server is optional. Select MID Server for tools on premise attached to a MID Server. Application is automatically set to DevOps and capability is set to REST.

c. Click Submit.
The tool is automatically **Connected Successfully** using a connection alias, and HTTP tool connection (basic authentication credential).

**Note:** If you do not have global admin privileges for your tool (to allow automatic configuration of the webhook URL), you may need to have the tool admin user configure it for you (cut and paste the webhook URL into the tool configuration). Once the webhook is configured in the tool, **Enter Manual Configuration Mode** to connect to the tool manually, then exit.

2. Click Discover to discover tool configuration, including existing application plans (projects associated with the tool), repositories, orchestration tasks, and pipelines.
Records are added to the corresponding related lists.

3. Automatically configure webhook URLs in Azure DevOps.

a. Click Configure.

b. Enter your Integration user (DevOps Integration User) and Password.
The DevOps Integration User is provided with the DevOps application and is used for inbound authentication from your tool to your ServiceNow instance.

ℹ️ **Note:** You must configure the password before the DevOps Integration User can configure a tool.

Notifications are sent from Azure DevOps tools to DevOps as the DevOps Integration User.

The **Track** field is automatically set to **True** in configured plan and repository records.

4. For discovered plans, import historical data from the tool and add an app to tie your tools together.

a. Open the plan record from the Plans related list and click the **Import** related link.
   
   Records are created for the plan in the Work Items, Plan Versions, and Features related lists.
   
   • Work items related list shows tasks, bugs, and stories.
     
     **Native State** and **Native Type** fields of the work item contain the original state and type values from the source tool.
   
   • Plan Versions related list shows releases.
   
   • Features related list shows epics and features.

ℹ️ **Note:** Historical import of Azure DevOps work items is not supported for Agile Boards CMMI process.

b. In the Apps related list, click **Edit...** to select an App to associate with the plan (project), or click **New** to create one.

ℹ️ **Note:** A plan must have an associated App record to complete the planning tool setup.
Example:

Azure DevOps tool record in DevOps

What to do next

For Azure pipelines, **Model an Azure pipeline in DevOps** to complete the configuration in DevOps.

- Map each pipeline to a specific app.
- Create pipeline steps and map each step to an Azure pipeline job.
- Configure change control.

**Model an Azure pipeline in DevOps**

Starting with version 1.12, model an Azure pipeline by mapping the pipeline to an app, and mapping DevOps pipeline steps to Azure pipeline jobs.

**Before you begin**

Role required: sn_devops.admin
Procedure

1. Map your pipeline to an app in DevOps.
   
   a. Navigate to **DevOps > Apps & Pipelines > Apps** and open the application record to associate with the pipeline.
   
   b. In the Pipelines related list, click **Edit...** to select a pipeline to associate with the app, or click **New** to create the pipeline. For a new pipeline, fill in the **Orchestration pipeline** field using the project name and pipeline name as specified in Azure DevOps Pipelines in path format.

   For example, **My Project/My Classic Build Pipeline**.

   ❗ **Note:** The project name must be specified with the pipeline because there could be multiple pipelines with the same name in different projects.

   c. Click **Submit**.

2. Open the pipeline record again and select the **Track** check box so events from the pipeline are received.

   ❗ **Note:** The **Track** check box must be selected to integrate the pipeline with DevOps.

3. Create DevOps steps to map to each Azure pipeline job so an orchestration task can be created. Steps can be created in one of the following ways.

   • Starting with version 1.18, automatically create and map pipeline steps in DevOps by running your Azure pipeline.

   Pipeline steps are automatically created, mapped, and associated when DevOps receives step notifications from your Azure pipeline during the run.

   • Manually create and map each pipeline step to an Azure pipeline job.

   In the Steps related list, click **New** to create a DevOps step for each Azure pipeline job (**Orchestration stage** field).

   ❗ **Note:** The **Orchestration stage** field value of each step is case-sensitive and must match the original name of the corresponding Azure pipeline job.

---

© 2021 ServiceNow, Inc. All rights reserved.

ServiceNow, the ServiceNow logo, Now, and other ServiceNow marks are trademarks and/or registered trademarks of ServiceNow, Inc., in the United States and/or other countries. Other company names, product names, and logos may be trademarks of the respective companies with which they are associated.
<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Pipeline in which the step is configured.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Pipeline step type.</td>
</tr>
<tr>
<td></td>
<td>◦ Build and Test</td>
</tr>
<tr>
<td></td>
<td>◦ Test</td>
</tr>
<tr>
<td></td>
<td>◦ Deploy</td>
</tr>
<tr>
<td></td>
<td>◦ Deploy and Test</td>
</tr>
<tr>
<td></td>
<td>◦ Manual</td>
</tr>
<tr>
<td></td>
<td>◦ Prod Deploy</td>
</tr>
<tr>
<td>Order</td>
<td>Order in which the steps are run.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The step order determines the</td>
</tr>
<tr>
<td></td>
<td>order of the cards in the Pipeline UI.</td>
</tr>
<tr>
<td></td>
<td>Starting with version 1.18, the order of</td>
</tr>
<tr>
<td></td>
<td>the cards in the Pipeline UI is by task</td>
</tr>
<tr>
<td></td>
<td>execution.</td>
</tr>
<tr>
<td>Orchestration stage</td>
<td>Azure pipeline job name (case-sensitive).</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> For step association with Azure</td>
</tr>
<tr>
<td></td>
<td>pipeline jobs, the Orchestration stage</td>
</tr>
<tr>
<td></td>
<td>field must be configured.</td>
</tr>
<tr>
<td>Business service</td>
<td>Configuration service that applies to the</td>
</tr>
<tr>
<td></td>
<td>step.</td>
</tr>
</tbody>
</table>

Once orchestration tasks are created, associate each orchestration task in the Orchestration Tasks related list with a DevOps pipeline step.

4. **Optional:** Select the **Change control** check box in a step to enable change acceleration and the corresponding configuration fields.
   For Azure release (CD) pipelines, enable change control in the first step of the required stage only. Change control for Azure release (CD) pipelines is supported only in pre-deployment gates.

**Note:** ServiceNow Change Management must be installed for change acceleration.

| Change receipt | (Optional) Starting with version 1.20, select to enable change receipt for the step so the pipeline doesn’t pause when a change request is created. |
All pipeline data is included in the change, but approval is not required for the pipeline to proceed.

<table>
<thead>
<tr>
<th>Change approval group</th>
<th>(Optional) Approval group for the change request. The change approval group becomes the Assignment group in the DevOps change request.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change type</td>
<td>Change request type to create.</td>
</tr>
<tr>
<td>(Optional) Template</td>
<td>(Optional) List of templates to use to auto populate fields for Normal or Emergency change requests. Select a template or create a new one.</td>
</tr>
<tr>
<td>(Optional) Standard change template</td>
<td>List of standard change templates to use for Standard change requests.</td>
</tr>
<tr>
<td>Change controlled branches</td>
<td>(Optional) (Multibranch only) Comma-separated list of branches under change control. Wildcards are supported.</td>
</tr>
</tbody>
</table>

Note: Ensure that the selected group has members and a group manager so the approver field is not empty.

You can set up change control in your Azure DevOps YAML or Classic Azure pipeline using the Azure Invoke REST API or the ServiceNow DevOps extension for Azure DevOps.

5. Navigate to DevOps > Tools > Orchestration Tools and in the Azure tool record, copy the DevOps Webhook URL field value. The webhook URL contains the DevOps location for Azure DevOps Pipelines to send messages, including the sys_id for the tool:

https://<devops.integration.user>:<password>@<your-instance>.service-now.com/api/sn_devops/v1/devops/tool/event/{sys_id of the record azure tool that was created}
Example:

DevOps app

App My App

- Name: My App
- Owner: System Administrator
- Application service

Related Links

Pipeline UI

Pipelines

- Name: MyPipeline
  - Description: My Classic Build Pip...
- Name: MyPipeline Demo Yaml
  - Description: Demo Yaml Pipeline
- Name: MyPipeline Yaml
  - Description: 423.azure...

DevOps pipeline

Pipeline CorpSite

- Name: CorpSite
- Type: 
- Order: 100
- Change control: false
- Change approval group: (empty)
- Configuration item: (empty)
- Change controlled branches: *
What to do next
Configure the Azure pipeline for DevOps

Configuring the Azure pipeline for DevOps
Change control, artifacts, and packages can be configured in the Azure pipeline for integration with DevOps.

You can configure change control in Azure pipelines two ways.
- Use the Azure Invoke REST API.
- Use the ServiceNow DevOps extension for Azure DevOps.

The ServiceNow DevOps extension for Azure DevOps includes:
- ServiceNow DevOps service connection
- ServiceNow DevOps Release Gate (starting with version 1.18)
- Azure build (CI) pipeline agent and server job custom tasks

Configuring change control using the Azure Invoke REST API
You can use the Azure Invoke REST API in your YAML or Classic Azure pipeline to configure change control for DevOps.

For Azure Invoke REST API details, visit the Microsoft documentation site and search for Invoke HTTP REST API task - Azure Pipelines.
Note:
Starting from DevOps version 1.29, if you have duplicate or reused job names in your pipeline execution steps, ensure that the `stageName` attribute contains `stageName/jobName` in it's value, i.e. `stageName = azureStageName/jobName`. The artifact registration tasks send both stage and job names to associate the artifact version to the correct task execution.

Generic service connection
Using the Azure Invoke REST API requires the creation of a generic service connection in Azure DevOps.

**Edit service connection**

Server URL

https://[].service-now.com/api/sn_devops/v1/devops/orches

Authentication

Username (optional)

`devops.integration.user`

Username for connecting to the endpoint

Password/Token Key (optional)

`******`

Password/Token Key for connecting to the endpoint

Details

Service connection name

`change1`

Description (optional)


Security

Grant access permission to all pipelines

YAML Azure pipeline

In Azure DevOps, a server task must be created with the service connection as the change control endpoint.
Invoke REST API payload requirements

<table>
<thead>
<tr>
<th>Azure pipeline type</th>
<th>Values</th>
</tr>
</thead>
</table>
| Build               | • buildNumber  
                    | • isMultiBranch 
                    | • branchName     |
| Release             | • releaseNumber |
|                     | • projectName |

ℹ️ Note: For release pipelines, set the Pre-deployment conditions > Advanced > Completion event field to Callback.

Build pipeline:

```powershell
- task: InvokeRESTAPI@1
  inputs:
    connectionType: 'connectedServiceName'
    serviceConnection: 'change1'
    method: 'POST'
    body: |
    {
      "buildNumber": "$(build.buildId)",
      "isMultiBranch": "true",
      "branchName": "$(build.sourceBranchName)"
    }
    waitForCompletion: 'true'
```

Release pipeline:

```powershell
- task: InvokeRESTAPI@1
  inputs:
    connectionType: 'connectedServiceName'
    serviceConnection: 'change1'
    method: 'POST'
    body: |
    {
      "releaseNumber": "$(Release.ReleaseId)",
      "projectName": "$(System.TeamProject)"
    }
    waitForCompletion: 'true'
```
Classic Azure pipeline

For a Classic Azure pipeline, an Invoke REST API server task must be added.

Classic Azure build pipeline example

Classic Azure release pipeline example
Use the ServiceNow DevOps extension for Azure DevOps

Install and configure ServiceNow DevOps extension for Azure DevOps to send build and release notifications from your Azure pipeline to ServiceNow DevOps application.

About this task
You can use ServiceNow DevOps extension on Visual Studio Marketplace to integrate your Azure pipeline with the ServiceNowDevOps application.

- **ServiceNow DevOps** service connection
  Required to connect the Azure pipeline to ServiceNow

- Starting with version 1.18, **ServiceNow DevOps Release Gate**
  Required to enable change control in Azure release (CD) pipelines (in pre-deployment conditions only)

Azure build (CI) pipeline custom tasks:

- Agentless (server) job
  🏷️ **ServiceNow DevOps Server Job Notification** custom task
  Required (in DevOps versions 1.16 and earlier) for agentless (server) jobs to send notifications from the Azure pipeline to ServiceNow DevOps.

  🔄 **Note:** Starting with version 1.17, job notification tasks are only required when using the ServiceNow DevOps Change Acceleration task.

  🏷️ **ServiceNow DevOps Server Change Acceleration** custom task
  Required for agentless (server) jobs to automatically create a change request in ServiceNow Change Management as part of the Azure pipeline.

  🔄 **Note:** Starting with version 1.19, the ServiceNow DevOps Server Change Acceleration task does not require ServiceNow DevOps Server Job Notification tasks.

  🏷️ Starting with version 1.13, **ServiceNow DevOps Server Package Registration** custom task
  Required for agentless (server) jobs to register a package in the ServiceNow instance

  🏷️ Starting with version 1.13, **ServiceNow DevOps Server Artifact Registration** custom task
  Required for agentless (server) jobs to register an artifact in the ServiceNow instance

- Agent job
- **ServiceNow DevOps Agent Job Notification** custom task
  Required (in DevOps versions 1.16 and earlier) for agent jobs to send notifications from the Azure pipeline to ServiceNow DevOps.

  **Note:** Starting with version 1.17, job notification tasks only required when using the **ServiceNow DevOps Change Acceleration** task.

- **Starting with version 1.13, ServiceNow DevOps Agent Package Registration** custom task
  Required for agent jobs to register a package in the ServiceNow instance

- **Starting with version 1.13, ServiceNow DevOps Agent Artifact Registration** custom task
  Required for agent jobs to register an artifact in the ServiceNow instance

**Procedure**

1. Go to [Visual Studio Marketplace](https://marketplace.visualstudio.com), search for the **ServiceNow DevOps** extension, and click **Get it free**.

   ![ServiceNow DevOps Extension](https://marketplace.visualstudio.com/id=9d2df620-1a0c-4d25-86ba-6d00e5474f4d)

   **Overview** | **Q & A** | **Rating & Review**
   --- | --- | ---
   This extension allows you to integrate Azure Pipelines with ServiceNow DevOps.
   [Get it free](https://marketplace.visualstudio.com/id=9d2df620-1a0c-4d25-86ba-6d00e5474f4d)

2. In Azure DevOps Pipelines, navigate to the service connections section in your project settings and create a **New service connection** using the **ServiceNow DevOps** service connection.

   | ServiceNow URL | https://<your-instance>.service-now.com/ |
   | Tool ID | The sys_id of the orchestration tool. You can copy this value from the webhook URL (toolid value), or obtain it directly |
using the `Copy sys_id` command on the Orchestration Tool form.

<table>
<thead>
<tr>
<th>Username</th>
<th>devops.integration.user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>Password for DevOps Integration User.</td>
</tr>
<tr>
<td>Service connection name</td>
<td>DevOps Connection</td>
</tr>
<tr>
<td>Grant access permission to all pipelines</td>
<td>Select check box.</td>
</tr>
</tbody>
</table>

**Example:**

**Azure pipeline: ServiceNow DevOps service connection**

Set up an Azure build (CI) pipeline in DevOps

Use the ServiceNow DevOps extension for Azure DevOps to configure job notifications, change control, and artifacts and packages in your Azure build (CI) pipeline.

**Procedure**

1. For DevOps versions 1.16 and earlier, in Azure DevOps Pipelines, navigate to the Tasks section of your Azure pipeline jobs and **Add (+)** the appropriate ServiceNow DevOps job notification custom task at the beginning and end of each job in your build (CI) pipeline.
Note: ServiceNow DevOps job notification custom tasks must be added twice per job (to indicate both the beginning, and end of each job) for notifications to be sent to DevOps successfully.

- For **Build** and **Test** job types, use the **ServiceNow DevOps Agent Job Notification** custom task.
- For **Server** job type, use the **ServiceNow DevOps Server Job Notification** custom task.

2. For DevOps versions 1.16 and earlier, in Azure DevOps Pipelines, configure ServiceNow DevOps job notification custom tasks.

<table>
<thead>
<tr>
<th>Display name</th>
<th>ServiceNow Job Notification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNow endpoint</td>
<td>My Connection</td>
</tr>
<tr>
<td>Upstream Job</td>
<td>Indicates the previous job in line. For example, the job before Test may be Build.</td>
</tr>
<tr>
<td>Job Execution Phase</td>
<td>Indicates where this ServiceNow Job Notification custom task is placed in the list of job tasks.</td>
</tr>
</tbody>
</table>

- **STARTED**
  First task in the job.
- **COMPLETED**
  Last task in the job.

Note: The ServiceNow Job Notification custom task must be added to the task list as both the first task of the job and the last task of the job.

3. Optional: In Azure DevOps Pipelines, Add (+) the **ServiceNow DevOps Server Change Acceleration** custom task to the Tasks section of your Azure pipeline agentless (server) job to configure change acceleration.

Note: The **ServiceNow DevOps Server Change Acceleration** task must be between the **ServiceNow DevOps Server Job Notification** tasks.

Starting with version 1.19, the server job notification tasks are no longer required when using the change acceleration task.
<table>
<thead>
<tr>
<th>Display name</th>
<th>ServiceNow Change Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNow endpoint</td>
<td>My Connection</td>
</tr>
<tr>
<td>Upstream job executed</td>
<td>Indicates the previous job in line. (Optional) For example, the job before Server may be Test.</td>
</tr>
<tr>
<td>Change request details</td>
<td>(Optional) Starting with version 1.24, set closure code and change request fields from within the pipeline. Click the information icon to view sample input.</td>
</tr>
</tbody>
</table>

4. **Optional:** In Azure DevOps Pipelines, **Add (+) the ServiceNow DevOps server or agent artifact and package registration custom tasks to the Tasks section of your Azure pipeline server or agent job to configure artifacts and packages.** See [DevOps change acceleration for releases](#) for more information regarding artifacts.

**Example:**

**Azure pipeline: ServiceNow DevOps job notification custom tasks**

- ![ServiceNow DevOps Agent Job Notification](image)
  - Add as the first and last task in each agent job to send notifications to ServiceNow DevOps.
- ![ServiceNow DevOps Server Job Notification](image)
  - Add as the first and last task in each server job to send notifications to ServiceNow DevOps.
```yml
trigger:
  - master
name: $(Date:yyyyMMdd)$Rev:

stages:
  - stage: Build
    jobs:
      - job: 'Build'
        steps:
          - task: ServiceNow DevOps Agent Job Notification@2
            inputs:
              connectedServiceName: 'devopsdev'
              Phase: 'STARTED'

            Settings
            - task: Bash@3
              inputs:
                targetType: 'inline'
                script: |
                # Write your commands here
                echo 'Hello world'
            - task: ServiceNow DevOps Agent Job Notification@2
              inputs:
                connectedServiceName: 'devopsdev'
                Phase: 'COMPLETED'
```

---

## Azure pipeline: ServiceNow DevOps Change Acceleration custom task

Add a server job to create a change request as part of the Azure pipeline.
Change request details example (starting with version 1.24)

```
steps:
  task: ServiceNow-DevOps-Server-Change-AccelerationExample
  inputs:
    connectedServiceName: 'AgileOne-ServiceNow DevOps Service Connection'
    UpstreamJob: 'Test'
    changeRequestDetails: |
      | "setCloseCode": true,
      | "setAttribute": |
      | "sys_created_by": |
      | "sys_update_by": |
      | "sys_created_on": "2021-03-09 18:58:41",
      | "priority": "2",
      | "work_start": "2021-03-05 00:00:00",
      | "work_end": "2021-03-08 00:00:00",
      | "comments": "This update for work notes is from ADD job",
      | "work_notes": "Update this to work notes",
      | "start_date": "2021-03-01 00:00:00",
      | "end_date": "2021-03-08 00:00:00"
```  

Set up an Azure release (CD) pipeline in DevOps

Starting with version 1.18, use the ServiceNow DevOps extension for Azure DevOps to configure change control, and artifacts and packages in your Azure release (CD) pipeline.

**About this task**

*Note:* Change control in Azure release (CD) pipelines is supported in pre-deployment gates only. Pre-deployment gate change requests are mapped to the step execution of the first job in that stage.

Phases in an Azure release pipeline are mapped to a step. The step name for a multi-config or multi-agent job must include the phase name only, and not the actual job name derived at pipeline execution run time.

Task executions for skipped jobs are marked as failed.

**Procedure**

1. In Azure DevOps Pipelines, navigate to your release pipeline and open the Pre-deployment conditions window.
2. Enable the Gates setting and click **+Add**.
3. Click the **ServiceNow DevOps Release Gate** and select the ServiceNow endpoint.
4. Exit the release gate configuration, and expand the Evaluation options section to configure the timing fields.
If the change request is canceled or rejected, the Azure pipeline release gate keeps re-evaluating the change request status at the configured interval until timeout. No user intervention is required for further attempts of re-evaluation after the change request has already been approved, rejected, or canceled.

<table>
<thead>
<tr>
<th>Timeout after which gates fail</th>
<th>Timeout value in minutes, hours, or days.</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the pre-deployment conditions fail, the subsequent jobs of the stage are marked as failed, and the start and end time defaults to the current system time.</td>
<td></td>
</tr>
</tbody>
</table>

5. **Optional:** Configure artifacts in your Azure release (CD) pipeline.
To set up artifacts using the build pipeline as the source, these names must match.

- Repository name, and build pipeline name (for example, DeployableRepo).
- Name property of the build pipeline artifact, and source alias property of the release pipeline artifact (for example, BuildDrop).

In addition to the build pipeline, you can select artifacts from any other eight sources. To track commit & work item details, follow these rules.

- When the source is the build pipeline, the *semantic version* property of the artifacts should be in the format **MAJOR.MINOR.PATCH** (for example 5.1.3).
- When the source is not the build pipeline, define a *semantic version* by implementing the **DevOpsArtifactSemanticVersionAPI** extension interface.

See [DevOps change acceleration for releases](https://example.com) for more information regarding artifacts.

**Example:**

![ServiceNow DevOps extension for Azure DevOps - Release Gate](https://example.com)
Azure release pipeline pre-deployment gate configuration

- Gates
  - Define gates to evaluate before the deployment.
  - Learn more
  - The delay before evaluation
    - 0
    - Minutes
  - Deployment gates
    - + Add
      - ServiceNow DevOps Release Gate
        - Enabled
      - Evaluation options
        - The time between re-evaluation of gates
          - 5
          - Minutes
        - Minimum duration for steady results after a successful gates evaluation
          - 0
          - Minutes
        - The timeout after which gates fail
          - 15
          - Minutes
      - Gates and approvals
        - Before gates, ask for approvals
        - On successful gates, ask for approvals
        - Ignore gates outcome and ask for approvals

Artifact setup - build pipeline source

```yaml
$ master

DeployableRepo / azure-pipelines.yml

- task: ServiceNow-DevOps-Agent-Artifact-Registration@1
- inputs:
  - connectedServiceName: Artifacts payload
  - artifactsPayload: |
  
  "artifacts": [ 
  
  { 
  
  "name": "BuildDrop", 
  "version": "$Build.BuildId", 
  "semanticVersion": "$Build.BuildId.0", 
  "BuildNumber": "$Build.BuildNumber", 
  "repositoryName": "DeployableRepo"
  }
  ]
```
Artifact setup - release pipeline

Associate Azure pipeline steps in DevOps

For manually created DevOps steps, associate each orchestration task in the Orchestration Tasks related list with a DevOps pipeline step to track the activity of each stage in your Azure pipeline.

Before you begin
Role required: sn_devops.admin

Procedure

1. Navigate to DevOps > Tools > Orchestration Tools, and open the Azure DevOps tool record.

2. In the Orchestration Tasks related list, enter the corresponding DevOps pipeline step in the Step field of each orchestration task.

Note: The Track field is set to True by default when you discover orchestration tasks and pipelines. Tracking is required to receive job notifications from Azure DevOps.

Azure pipeline job run notifications are sent to the DevOps application. Each task execution notification corresponds to an orchestration task and, since orchestration tasks are mapped to a certain step in your DevOps pipeline, you can track the activity in each stage of your pipeline.
Example:

DevOps associate steps

Configure SonarQube scans on Azure DevOps pipelines

Configure SonarQube or SonarCloud scans on MS Azure DevOps pipelines. Check Azure DevOps pipeline executions for SonarQube scans on every stage of the pipeline's execution and fetch lists and details of scans from any stage to ServiceNow® DevOps. Drill down on the Scan Details based on categories.

Before you begin

- Upgrade to DevOps version 1.27
- Connect, configure the Azure DevOps tool and discover existing repositories, orchestration tasks, and pipelines.
- Create a SonarQube tool record. For more information, see Create a SonarQube tool record.
- Install the SonarQube extension from the Visual Studio Marketplace and configure branch analysis to use the Azure Devops tasks in your build definitions to analyze your projects. For more information, see https://docs.sonarqube.org/latest/analysis/azuredevops-integration/
- Install the following custom extension tasks on your Azure Devops instance.
ServiceNow extension to integrate Azure Pipelines with ServiceNow DevOps. For more information, see Use the ServiceNow DevOps extension for Azure DevOps

ServiceNow DevOps Build Sonar Registration (for Build pipelines)
ServiceNowDevOps Release Sonar Registration task (for Release pipelines)

Roles required:
- admin or sn_devops.admin in ServiceNow DevOps
- admin in Azure Devops
- admin role in SonarQube with access to all projects that the sonar scans are configured on.

About this task
By default, Azure DevOps provides you with the following tasks to run a Sonar scan on build and release pipelines:
- Prepare analysis on SonarCloud
- Run Code analysis
- Publish Quality Gate Result

Add and configure custom ServiceNow extension tasks in order to them default tasks, to fetch the scan details to ServiceNowDevOps from Azure DevOps build and release pipelines. The scan details that are fetched are:
- pipelineName
- buildNumber
- stageName
- branchName
- sonarProjectKey
- sonarInstanceUrl
Procedure

1. Set up and configure the Azure DevOps and Sonar Integration to fetch scan analysis from Build pipelines.

   a. In the Azure DevOps console, navigate to Organization > SonarIntegrations > Pipelines > Jobs.

   b. Click the Add Tasks icon (+), and search for the ServiceNow extensions in the Add tasks search bar.

   c. Add the following tasks.

      • ServiceNow DevOps Build Sonar Registration task.
      • ServiceNow DevOps Release Sonar Registration task.

   d. On the form, fill in the fields.

<p>| ServiceNow DevOps Build/ Release Sonar Registration |
|----------------------------------|-----------------------------------------------|
| Field                            | Description                                   |
| Display name                     | Auto-populates on entering key and URL.        |
| ServiceNow endpoint              | The ServiceNow instance endpoint that is automatically created during Tool configuration. Use the same service connection for this task. |
| Sonar project Key                | Enter the same project key value which you used while configuring the Prepare analysis on SonarCloud task. |
| Sonar Instance Url               | Enter the same URL which you used to connect to the Sonar tool during tool creation. |
| Job Name                         | The name of the Azure DevOps pipeline job.     |</p>
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This field appears only for the ServiceNow DevOps Release Sonar Registration task.</td>
</tr>
</tbody>
</table>

**Important:** In the list of tasks for the pipeline job, ensure that the ServiceNow DevOps Build Sonar Registration or ServiceNow DevOps Release Sonar Registration task is added after the Run Code Analysis task.

You have configured the build pipelines to send sonar scan results to your DevOps application. Scan results are mapped to the Software Quality capability are processed by an associated base system sub flow (FetchSonarScanId), once an event is created and processed in the Inbound Events table.

2. Associate Azure pipelines steps in DevOps.

3. Run the pipeline.

**Results**

Based on the scan results on various stages of pipeline's execution, the results are shown in the corresponding step of the pipeline in ServiceNow DevOps.

Inbound events are created for notifications and subflows are triggered based on the notification type and capability.

**What to do next**

- View scan details as part of Task Executions. View details of all the Sonar scans that are part of the task execution mapped to a build or release pipeline execution step.

  1. Navigate to DevOps > Orchestrate > Task Execution click a relevant Task Execution record.

  2. Click the Software Quality Summary related list.

  3. Click a relevant Scan ID record.

  The Software Quality Scan Summary and Scan Details display. For more information see, Software Quality Scan Summary

- View scan details as part of Change Request. View all the scans that were part of this build/release pipeline in the Software Quality Results > Software Quality Summary related list.
1. Navigate to **DevOps > Orchestrate > Pipeline Change Requests**

2. Click the Software Quality Summary related list.

3. Click a relevant Scan ID record.
   The Software Quality Scan Summary and Scan Details display. For more information see, **Software Quality Scan Summary**

**Restarting failed build or release pipeline jobs and stages**

Rerun or redeploy Azure DevOps build, release changes, or pipelines that are failed or canceled in that stage or pipeline. The reattempts display on the DevOps pipeline UI as continuous runs instead of creating new executions.

**Rerun Azure DevOps pipelines or stages**

Starting with version 1.30, you can rerun a failed or canceled build or release pipelines or change jobs in Azure DevOps. The reruns are processed as part of the same pipeline execution as the first run in ServiceNowDevOps. You can rerun entire pipelines or specific failed or canceled jobs and stages. You can now choose to reuse a change request instead of creating a new change request each time you restart a stage or a pipeline.

⚠ **Note:** Upgrade all plugins and extensions to the 1.30 versions for the base system Test API, Artifact registration and package registration functionality to work for reruns.

An **attemptNumber** parameter is added to the payload which helps us track reruns. Associated test summary, software quality scan results, commits, work items corresponding to every rerun attempt is also updated in ServiceNow DevOps.

If you are using the **Azure Invoke REST API** you must add the attempt number parameter to your payload body in the specified syntax format for build and release pipelines. If you do not specify the attempt id parameter, the default attempt number is set to 1.

Example attempt number parameter in build pipeline payload:

```
"attemptNumber": "$\{system.jobAttempt\}"
```

Example attempt number parameter in the release pipeline payload:

```
"attemptNumber": "$\{Release.AttemptNumber\}"
```

⚠ **Note:** Do not use the existing started and completed notifications for the stage jobs. If your jobs consider the started and completed notifications the rerun functionality does not work.
Reusing Change Requests

If a change enabled job is rerun, and a change request exists for the previous run/attempt, you can choose to reuse the previous change request or create a new change request, using the base system ‘DevOps Change Request Reusability Decision Subflow’. The default implementation of this subflow, allows you to reuse a change request from the previous attempt if the change request is in implement, or post-implement states. If the Change request is in any other state, by default, a new change request is created when you rerun the job. Per existing behavior, all associated details such as test summaries, and scans, are newly generated while commits and work items are retained unchanged for new change requests.

For example, when a pipeline fails at a specific stage after the change request is approved, and you rerun that stage. The change request is reused, the associated test summary and software quality scans, and the commits and work items associated to the artifact are associated with the same change request which you approved.

To apply a custom logic for reusability, you can copy the existing subflow, make the changes, publish it, and update the new subflow name under DevOps Properties > DevOps Change Request Reusability Decision Subflow.

In the regular base system flow when a change is created, ‘DevOps Model Change Request flow’ is used to update the State field of step execution record after a decision is taken on the change request. However, when you reuse a change, the first trigger condition of a change request being created is not met. A base system subflow ‘DevOps Change Request Reusability Model Subflow’ is triggered instead, whenever a change request is reused when a job is a rerun. The default implementation of this subflow is similar to the DevOps Model Change Request flow. You can create a custom subflow and update the subflow name at DevOps Properties > DevOps Change Request Reusability Model Subflow.

Pipeline UI changes

Starting with version 1.30, ServiceNow DevOps now synchronizes all the changes that are caused when you restart or rerun a stage or a job, and displays them in the DevOps pipeline UI.

- Click a card to view the latest attempt of that stage.
- Click the View all attempts link to see all the step executions and related information associated to the step or stage that is run more than once.
- The View change link displays the change request associated with the latest attempt.
In previous releases, failed jobs were either ignored or a new pipeline execution job was created for reruns and processed accordingly. For more information, see DevOps Pipeline UI.

**Execution sequence and waiting logic for rerun jobs**

Processing sequence and waiting logic for rerun jobs are different when you reuse or create a change request as part of a rerun job.

**Existing Considerations**

- A change request should not exist in a stage which contains parallel jobs.
- If more than one stage is running in parallel, change request should not be the first job in both the stages.

ℹ️ **Note:** Parallel jobs appear in the ServiceNow DevOps pipeline UI in a sequential order or the configured run order, though the jobs are processed in parallel.

**Upgrade Considerations**

There is no change in the functionality or execution when you run the first pipeline attempt. All stages are processed sequentially and associated tests, software quality scans and change requests are executed and created as modeled.

ℹ️ **Note:**

- Run a new pipeline after you upgrade if you have rerun stages and pipelines before upgrading. Rerun attempts and failed events prior to the upgrade are ignored by ServiceNow DevOps for reattempts, even after you upgrade to version 1.30.
- If you have only run the pipeline once before upgrade, you can rerun the stage or the pipeline. The rerun functionality applies as designed and is saved in ServiceNow DevOps.

**Execution Sequence and processing logic**

- If the same artifact version registration call is received in reattempt, the registration call is ignored.
- Package registration calls with same package name are not ignored. A new package associated to artifact versions and pipeline execution is created during reattempt. The artifacts that are associated to the latest package, will be shown on the change request.
From the Azure DevOps GUI, if you rerun a stage in a build pipeline the subsequent stages reruns are also triggered in it’s specified sequence. If you reattempt processing a pipeline before all the stages of the previous attempt are not complete. The subsequent attempt waits until all the events in the previous attempt are processed.

For release pipelines, the stages are run in the specified sequence only during the first run. For subsequent rerun attempts manually run each stage. In release pipelines even if stages are running in parallel in Azure DevOps, from the second attempt onwards the events are processed in the specified sequence.

- When a new change request is created for a reattempt stage job, and the stage that you are reattempting includes a test and a software quality scan only the latest Test Summary and Software Quality scan results display on the change request related list.

- When a change request is reused for a rerun stage job, the Test Summary and Software Quality scan results for each attempt displays in the change request related list.

**Parallel stages in Azure DevOps release pipelines**

Parallel stages in a release pipeline are now processed concurrently and displayed in the DevOps pipeline UI in real time. Base system pre-deployment conditions and release gates enable you to create change requests that include details from parallel stages.

**Base system parallel stage support for Azure DevOps**

Organizations use parallel stages to automate and speed up release processes for tasks that can be performed in parallel. For example, a release pipeline has integrated multiple test and software quality tools and has jobs configured to run in parallel. Not running each job sequentially significantly speeds up the release pipeline execution.

Starting with version 1.31, ServiceNow DevOps supports processing parallel stages in release pipelines and displays the stages in a parallel view in the DevOps pipeline UI. Effectively, the DevOps pipeline UI replicates the Azure DevOps GUI in real time.

You can also see the details of the processed stages in the pipeline UI.

**Important:** Support for parallel stages is restricted to Release pipelines. Build pipelines continue to appear in a sequential or serial manner in the DevOps pipeline UI, even if parallel stages are configured for build pipelines in Azure DevOps.
Until version 1.30, parallel stages or jobs were processed serially or sequentially, or even ignored in ServiceNowDevOps. As a result, the DevOps pipeline UI did not display the stages in a parallel view. Since ServiceNowDevOps did not support processing a single stage with multiple upstream stages, change requests that were created before the production deploy stage would contain details from only single upstream stages.

**ServiceNow® DevOps release gate in pre-deployment conditions to create Change requests**

Starting with version 1.31, a base system ServiceNowDevOps Release Gate is added along with pre-deployment conditions. Enable base system deployment gates that are configured to call the Now Platform instance, to create a change request before the deploy to production stage. Change requests are now created after all previous (upstream) stages complete processing. The change request captures relevant details from all upstream stages and displays them in the following corresponding related lists.

- Commits
- Work Items
- Test Summaries
- Software Quality Summary
- Artifact Versions

After the pipeline execution completes processing the parallel stages preceding the production deployment stage, a change request is automatically created and mapped to the deploy to production stage in the Pipeline Executions view. The production stage completes processing once the change request is approved.

From the **Pipeline Execution** view of the relevant pipeline, click the **Pipeline UI** related link to view the real-time state of the pipeline as it appears in Azure DevOps. The associated artifact details which are sourced from the build pipeline, Test Results, Software Quality Summary Results display on the pipeline UI.

**Upgrade Considerations**

Ensure that you review the following considerations before upgrading to version 1.31.

> **Important:** A change request should not exist in a stage which contains parallel jobs.
• Starting with version 1.31, the **Upstream execution** column in the Task Executions table is no longer displayed for fresh installations. Any customizations that you have made using the **Upstream execution** column prior to the upgrade are unaffected.

• If stages are running in parallel, a change request should not be the first job in any stage.

• New release pipeline executions after upgrading to version 1.31, will process parallel stages concurrently and display parallel stages and associated details in the pipeline UI. Azure DevOps release pipelines that are already executed and stored in ServiceNow DevOps prior to the upgrade remain unaffected and continue to display parallel stages (that are already executed and persisted) in ServiceNowDevOps serially.

• If the pre-deployment ServiceNow DevOps release gate is enabled in more than one start stage in a release pipeline with multiple start stages, it may result in multiple pipeline executions.

**Note:** A package is created for each start stage but any one package is associated per pipeline execution.

**Azure DevOps PAT scopes for DevOps**

Scope access levels are required when using a personal access token to access Azure DevOps during setup.

Mandatory scope access level settings.

• User Profile: Read

• Project & Team: Read

**Note:** Read-level access must be set for **User Profile** and **Project & Team** scopes to access Azure DevOps for tool setup.

Other scope access level settings are based on the capability you have configured. Set the corresponding access level for seamless functionality.

**Scope access level settings per capability**

<table>
<thead>
<tr>
<th>Capability</th>
<th>Scope</th>
<th>Access level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boards</td>
<td>Work item</td>
<td>Read</td>
</tr>
<tr>
<td>Repos</td>
<td>Code</td>
<td>Read</td>
</tr>
<tr>
<td>Build pipelines</td>
<td>Build</td>
<td>Read &amp; Execute</td>
</tr>
</tbody>
</table>

© 2021 ServiceNow, Inc. All rights reserved.

ServiceNow, the ServiceNow logo, Now, and other ServiceNow marks are trademarks and/or registered trademarks of ServiceNow, Inc., in the United States and/or other countries. Other company names, product names, and logos may be trademarks of the respective companies with which they are associated.
Scope access level settings per capability (continued)

<table>
<thead>
<tr>
<th>Capability</th>
<th>Scope</th>
<th>Access level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release pipelines</td>
<td>Release</td>
<td>Read</td>
</tr>
<tr>
<td>Release gates</td>
<td>Token administration</td>
<td>Read &amp; Manage</td>
</tr>
<tr>
<td>Test build and release pipelines</td>
<td>Test management</td>
<td>Read</td>
</tr>
</tbody>
</table>

DevOps work item import for Azure Boards

Azure Boards work items are mapped to default ServiceNow DevOps states and types during import. You can use the DevOpsAzureDevOpsWorkItemHelper script include to customize the mappings.

Default Azure Boards work item mapping

Native State and Native Type fields of the work item contain the original state and type values from the source tool.

Work item type mapping

<table>
<thead>
<tr>
<th>ServiceNow DevOps</th>
<th>Azure Boards Basic</th>
<th>Azure Boards Agile</th>
<th>Azure Boards Scrum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Task</td>
<td>Task</td>
<td>Task</td>
</tr>
<tr>
<td>Bug</td>
<td>Issue</td>
<td>Bug</td>
<td>Impediment</td>
</tr>
<tr>
<td>Story</td>
<td>--</td>
<td>User story</td>
<td>Product backlog item</td>
</tr>
<tr>
<td>Epic</td>
<td>Epic</td>
<td>Epic</td>
<td>Epic</td>
</tr>
<tr>
<td>Feature</td>
<td>--</td>
<td>Feature</td>
<td>Feature</td>
</tr>
</tbody>
</table>

Note: Historical import of Azure DevOps work items is not supported for Agile Boards CMMI process.
### Work item state mapping

<table>
<thead>
<tr>
<th>ServiceNow DevOps</th>
<th>Azure Boards Basic</th>
<th>Azure Boards Agile</th>
<th>Azure Boards Scrum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned</td>
<td>To Do</td>
<td>New</td>
<td>New Open Approved Committed To do</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WIP</td>
<td>Doing</td>
<td>Active Design</td>
<td>In Progress Design</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete</td>
<td>Done</td>
<td>Ready Closed</td>
<td>Done Ready Closed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deleted</td>
<td>Deleted</td>
<td>Completed Deleted</td>
<td>Removed</td>
</tr>
</tbody>
</table>

**Note:** When an imported Azure Boards work item type or state is not recognized, the value is set to **Other**.

### Customize Azure Boards state and type mappings

Access the DevOpsAzureDevOpsWorkItemHelper script include in the **System Definition > Script Includes** module.

This script example adds new states and types for custom processes MyScrum and CustomBasic. CustomBasic inherits the state and type defined for Basic process.

```javascript
var DevOpsAzureDevOpsWorkItemHelper = Class.create();

DevOpsAzureDevOpsWorkItemHelper.prototype =
Object.extendsObject(DevOpsAzureDevOpsWorkItemHelperSNC, {

setDefaultProcess: function (projectProcess) {

DevOpsAzureDevOpsWorkItemHelperSNC.prototype.setDefaultProcess.call(this, projectProcess);

//set custom states and types
var newStates, newWITypes;
```

© 2021 ServiceNow, Inc. All rights reserved.
if (projectProcess == 'NPScrum'){
  // no parent process set. So type and states available will be limited to newStates
  // and newWITypes
  newStates = {
    'Delayed': 'planned',
    'Approved': 'wip'
  };
  newWITypes= {
    'Request': 'story',
    'Incident': 'task'
  };
}
else if (projectProcess == 'CustomBasic'){
  // set parent process to Basic to inherit basic states and types
  this.setParentProcess('Basic');
  newStates = {
    'Auto-Approved': 'wip'
  };
  newWITypes= {
    'UserStory': 'story'
  };
}

this.setStates(newStates);
  this.setWorkItemTypes(newWITypes);
},
type: 'DevOpsAzureDevOpsWorkItemHelper'
});

Track specific pipelines in Azure DevOps

Enable and configure specific pipelines in Azure DevOps that you want to track. Select the pipelines you want to monitor and receive job notifications from Azure DevOps.

Before you begin

- Connect DevOps to Azure DevOps tools.
- Ensure that you have modeled, configured, and associated Azure pipelines to DevOps.
- Review how to configure form layouts. For more information, see Configuring the form layout
Role required: admin, personalize_form

About this task
By default, the Track Specific Pipeline field is set to False when you discover Azure DevOps orchestration tasks and pipelines. The Track flag for all Azure DevOps pipelines are also disabled. This leads to all pipelines sending job notifications to DevOps and increased processing times, which could cause the pipeline you really want to track to be delayed. Enable the Track Specific Pipeline check box to process only notifications for the pipeline events that you specify to track, and ignore all other pipeline notifications.

Procedure
1. Navigate to DevOps > Tools > Orchestration Tools, and open the Azure DevOps tool record.
2. Click the Additional Actions icon (Ξ), and select Configure > Form Layout.
3. Use the slushbucket, and move the Track Specific Pipeline field to the selected list.
4. Click Save.
   You are redirected back to the DevOps Tool > Azure DevOps form, and the Track Specific Pipeline check box appears.
5. Select the Track Specific Pipeline check box.
6. Navigate to the Pipelines related list.
7. Click the Update Personalized List icon (⚙).
8. In the Personalize List Columns form, use the slushbucket to move the Track field to the Selected list.
9. Click OK.
   You are redirected back to the Pipelines related list, where the Track field now appears.

10. Navigate to the pipeline step which you want to track, double-click the Track field.

11. Change the Track field’s value to true, and click Save.

**Results**
Only the pipelines that have Track field set to true in the Pipelines related list are now being tracked and sent to the Inbound Events table [sn_devops_inbound_event_list.do]. All pipelines that have the Track field set to false are ignored.

**Note:**
- If the Track Specific Pipeline check box is selected only those pipelines that have the Track field value set to true are tracked and sent to the Inbound Events table. If Track field is set to false for all pipelines none of the pipelines are tracked.
- If the Track Specific Pipeline check box is unselected, all events of the pipeline are tracked and sent to the Inbound Events table.
Bitbucket integration with DevOps

Integrate Bitbucket coding tool with DevOps by configuring a connection and credential alias, and creating a tool record in DevOps.

⚠️ Note: You can create a new tool and simultaneously create a new connection alias. Since the child alias is created during tool creation, you do not need an admin role nor do you need to create a new connection alias as a prerequisite. Create the new connection alias as an sn_devops.admin user using Connections and Credentials template which are dynamically populated based on the tool you select. For more information, see Create new DevOps tool from the Tools list view.

Configure connection and credential alias - Bitbucket

Before you set up a tool record in DevOps, your admin configures the DevOps CreateDevOpsTool connection and credential alias to allow access to the tools in your environment.

Before you begin

Starting with version 1.15, DevOps connection and credential aliases (DevOpsAlias1 through DevOpsAlias10) are used to connect automatically when you set up your DevOps planning, coding, and orchestration tools.


To connect to more than 10 tools, or if you receive an error saying all DevOpsAlias connection and credential aliases are being used, an admin can create additional aliases in the sn_devops application scope.

⚠️ Note: If additional sys_alias records for DevOps are needed, they must be created by an admin in the sn_devops application scope.

You can reuse a DevOpsAlias alias no longer in use by inactivating the HTTP connection, if needed.

Role required: admin, sn_devops.admin

About this task

Your admin must configure an HTTP connection in the CreateDevOpsTool connection and credential alias provided, and add admin credentials to connect automatically when you set up your DevOps planning, coding, and orchestration tools:
• URL: https://<instance name>.service-now.com/

• Credentials: admin

ℹ️ **Note:** Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

You only need to configure the CreateDevOpsTool connection and credential alias once to set up all of your tools in DevOps.

The CreateDevOpsTool connection and credential alias is used with the DevOps Create a Tool subflow in Flow Designer. You can view executions in Flow Designer for information regarding flow results.

**Procedure**

1. Navigate to Connections & Credentials > Connection & Credential Aliases and open the CreateDevOpsTool record.

2. In the Connections related list, create a record and enter a Name for the connection.

3. On the Connection form, click the Credential field lookup list, and then click New to create an admin credential.

   a. Click Basic Auth Credentials and enter a Name.

   b. Enter admin username and password (required to access the tools in your DevOps environment).

      ℹ️ **Note:** Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

4. On the Connection form, enter https://<instance name>.service-now.com/ for the Connection URL.

**Create a Bitbucket tool record in DevOps**

Create a Bitbucket tool record in DevOps to connect and discover repositories.

**Before you begin**

Role required: sn_devops.admin

**About this task**

Actions:
• Connect to Bitbucket and get the webhook URL when you submit a DevOps tool record.
• Discover repositories.
• Configure the webhook in the Bitbucket repository.
• Import branch and commit records.

Starting with version 1.13, capture tags from Bitbucket coding tool commits.

Procedure
1. Create a tool record in DevOps to automatically connect to Bitbucket and get the webhook URL.

   a. Navigate to DevOps > Tools > Create New and create a record.

   b. Enter a Tool Name and fill in the tool details.

<table>
<thead>
<tr>
<th>Tool Integration</th>
<th>Bitbucket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool URL</td>
<td>Bitbucket tool URL</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td><a href="https://bitbucket.com">https://bitbucket.com</a></td>
</tr>
<tr>
<td>Tool Username</td>
<td>Bitbucket username</td>
</tr>
<tr>
<td>Tool Password / Access Token</td>
<td>Bitbucket password or access token</td>
</tr>
</tbody>
</table>

   MID Server is optional. Select MID Server for tools on premise attached to a MID Server. Application is automatically set to DevOps and capability is set to REST.

   c. Click Submit.
   The tool is automatically Connected Successfully using a connection alias, and HTTP tool connection (basic authentication credential).

   Tip: If you do not have global admin privileges for your tool (to allow automatic configuration of the webhook URL), you may need to have the tool admin user configure it for you (cut and paste the webhook URL into the tool configuration). Once the webhook is configured in the tool, Enter Manual Configuration Mode to connect to the tool manually, then exit.

2. Click Discover to discover existing repositories for the coding tool.
Repository records are added to the Repositories related list.

3. Automatically configure the webhook URL in a Bitbucket repository to send notifications to the DevOps tool.

   a. Open the discovered repository record from the Repositories related list and click **Configure**.

   b. Enter your **Integration user** (DevOps Integration User) and **Password**. The DevOps Integration User is provided with the DevOps application and is used for inbound authentication from your tool to your ServiceNow instance.

      ❗ **Note:** You must configure the password before the DevOps Integration User can configure a tool.

      Notifications are sent from the repository to DevOps as the DevOps Integration User.

      The **Track** field is automatically set to **True** in the repository record.

4. Click **Import** to import historical data from the repository.

   ❗ **Note:** Import by date range is not supported for Bitbucket coding tool.

   Imported branch records and commit records from the repository are added to the corresponding related lists.

5. In the **App** field, click the lookup list and select an App record to associate with the repository, or click **New** to create one.
GitHub integration with DevOps

Integrate GitHub coding tool with DevOps by configuring a connection and credential alias, and creating a tool record in DevOps.

Both GitHub and GitHub Enterprise are supported.

Connect using:

- Basic authentication
- OAuth 2.0 credentials (Starting with version 1.17)

⚠️ Note: You can create a new tool and simultaneously create a new connection alias. Since the child alias is created during tool creation, you do not need an admin role nor do you need to create a new connection alias as a prerequisite. Create the new connection alias as an sn_devops.admin user using Connections and Credentials template which are dynamically populated based on the tool you select. For more information, see Create new DevOps tool from the Tools list view.

Connect DevOps to GitHub

Configure a connection and credential alias and create a GitHub tool record in DevOps.
Configure connection and credential alias - GitHub

Before you set up a tool record in DevOps, your admin configures the DevOps CreateDevOpsTool connection and credential alias to allow access to the tools in your environment.

Before you begin

Starting with version 1.15, DevOps connection and credential aliases (DevOpsAlias1 through DevOpsAlias10) are used to connect automatically when you set up your DevOps planning, coding, and orchestration tools.


To connect to more than 10 tools, or if you receive an error saying all DevOpsAlias connection and credential aliases are being used, an admin can create additional aliases in the sn_devops application scope.

Note: If additional sys_alias records for DevOps are needed, they must be created by an admin in the sn_devops application scope.

You can reuse a DevOpsAlias alias no longer in use by inactivating the HTTP connection, if needed.

Role required: admin, sn_devops.admin

About this task

Your admin must configure an HTTP connection in the CreateDevOpsTool connection and credential alias provided, and add admin credentials to connect automatically when you set up your DevOps planning, coding, and orchestration tools:

• URL: https://<instance name>.service-now.com/

• Credentials: admin

Note: Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

You only need to configure the CreateDevOpsTool connection and credential alias once to set up all of your tools in DevOps.

The CreateDevOpsTool connection and credential alias is used with the DevOps Create a Tool subflow in Flow Designer. You can view executions in Flow Designer for information regarding flow results.
Procedure

1. Navigate to Connections & Credentials > Connection & Credential Aliases and open the CreateDevOpsTool record.

2. In the Connections related list, create a record and enter a Name for the connection.

3. On the Connection form, click the Credential field lookup list, and then click New to create an admin credential.
   - Click Basic Auth Credentials and enter a Name.
   - Enter admin username and password (required to access the tools in your DevOps environment).

   **Note:** Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

4. On the Connection form, enter https://<instance name>.service-now.com/ for the Connection URL.

Create a GitHub tool record in DevOps

Create a GitHub tool record in DevOps to connect and discover repositories.

Before you begin
Role required: sn_devops.admin

About this task
Actions:
- **Connect** to GitHub and get the webhook URL when you submit a DevOps tool record.
- **Discover** repositories.
- **Configure** the webhook in the GitHub repository.
- **Import** branch and commit records.

Starting with version 1.17, using OAuth 2.0 credentials with GitHub integration is supported.

Starting with version 1.13, capture tags from GitHub coding tool commits.
Procedure

1. Create a tool record in DevOps to automatically connect to GitHub and get the webhook URL.

a. Navigate to DevOps > Tools > Create New and create a record.

b. Enter a Tool Name and fill in the tool details.

<table>
<thead>
<tr>
<th>Tool Integration</th>
<th>GitHub</th>
</tr>
</thead>
<tbody>
<tr>
<td>(GitHub Enterprise is for on-premise configuration.)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub tool URL.</td>
</tr>
<tr>
<td>For example: <a href="https://api.github.com">https://api.github.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GitHub credential type (starting with version 1.17)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Basic Auth</td>
</tr>
<tr>
<td>◦ GitHub username</td>
</tr>
<tr>
<td>◦ Personal access token</td>
</tr>
</tbody>
</table>

**Note:** Only a personal access token is supported with basic authentication.

When you generate a Personal access token (PAT) for GitHub, you must select the scopes to authorize if you are not granting complete access.

<table>
<thead>
<tr>
<th>GitHub Tool Credential</th>
</tr>
</thead>
<tbody>
<tr>
<td>• OAuth</td>
</tr>
</tbody>
</table>

See Setting up GitHub OAuth 2.0 credentials for DevOps.

MID Server is optional. Select MID Server for tools on premise attached to a MID Server. Application is automatically set to DevOps and capability is set to REST.

c. Click Submit.
The tool is automatically **Connected Successfully** using a connection alias, and HTTP tool connection (basic authentication or OAuth credential).

**Note:** If you do not have global admin privileges for your tool (to allow automatic configuration of the webhook URL), you may need to have the tool admin user configure it for you (cut and paste the webhook URL into the tool configuration). Once the webhook is configured in the tool, **Enter Manual Configuration Mode** to connect to the tool manually, then exit.

2. Click **Discover** to discover existing repositories for the coding tool. Repository records are added to the Repositories related list.

3. Automatically configure the webhook URL in a GitHub repository to send notifications to the DevOps tool.

   a. Open the discovered repository from the Repositories related list and click **Configure**.

   b. Enter your **Integration user** (DevOps Integration User) and **Password**. The DevOps Integration User is provided with the DevOps application and is used for inbound authentication from your tool to your ServiceNow instance.

   **Note:** You must configure the password before the DevOps Integration User can configure a tool.

   Notifications are sent from the repository to DevOps as the DevOps Integration User.

   The **Track** field is automatically set to **True** in the repository record.

4. Click **Import** to import historical data from the repository. Imported branch records and commit records from the repository are added to the corresponding related lists.

5. In the **App** field, click the lookup list and select an App record to associate with the repository, or click **New** to create one.
Example:

GitHub tool record in DevOps

<table>
<thead>
<tr>
<th>Name</th>
<th>GitHub</th>
<th>Connection state</th>
<th>Last discovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool</td>
<td>GitHub</td>
<td>Connected</td>
<td>2020-03-05 10:43:02</td>
</tr>
<tr>
<td>Connection alias</td>
<td>GitHub</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tool URL: [https://github.com](https://github.com)
Webhook URL: [https://<DevOps integration user ID>:<integration user password>@github.com/api/v3/hooks](https://<DevOps integration user ID>:<integration user password>@github.com/api/v3/hooks)

Setting up GitHub OAuth 2.0 credentials for DevOps

Starting with version 1.17, integrate your GitHub account with your ServiceNow instance by creating a custom OAuth application in GitHub and authenticating requests from ServiceNow DevOps.

You can set up OAuth 2.0 credentials for GitHub Apps and OAuth Apps.

- **GitHub Apps**
  - JWT
  - Authorization Code

- **OAuth Apps**

OAuth 2.0 credentials for GitHub Apps - JWT

Starting with version 1.17, configure your GitHub account, generate and attach the JKS certificate, create a JWT signing key and provider, register GitHub, and create a credential record for the GitHub App provider.

Role required: oauth_admin

Configure the GitHub App in your GitHub account (JWT)

Create a custom GitHub App from your GitHub account to enable OAuth 2.0 authentication with your ServiceNow instance.
Before you begin

GitHub requirements:

• GitHub account
• GitHub App configured to integrate with ServiceNow

About this task

Complete these steps from your GitHub account. See Building GitHub Apps on the GitHub Developer site for instructions on creating and configuring custom applications.

Procedure

1. From your GitHub account, create your GitHub App by navigating to Developer Settings > GitHub Apps.

2. In the Homepage URL field, enter https://<instance-name>.service-now.com.

3. In the User authorization callback URL field, enter https://<instance-name>.service-now.com/oauth_redirect.do.

4. In the Identifying and authorizing users section, uncheck the Expire user authorization tokens field.

5. In the Webhook section, uncheck the Active field.

6. Leave the remaining fields empty (default).

7. In the Repository permissions section, configure these settings.

<table>
<thead>
<tr>
<th>Contents</th>
<th>Read-only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td>Read-only</td>
</tr>
<tr>
<td>Webhooks</td>
<td>Read and write</td>
</tr>
</tbody>
</table>

8. Leave the remaining permissions as No access (default).

9. After creating the GitHub App, generate a new private key and save it to your machine.

10. Install the newly created GitHub App on the accounts of your choice.

Generate the JKS certificate for GitHub

Generate a JKS certificate for the JWT authentication of ServiceNow DevOps requests for GitHub.

Before you begin

Role required: admin
Procedure

1. Make a copy of the GitHub App private key previously generated, and rename it to use the .key extension.
2. Create a CA signed certificate using the GitHub App private key:
   
   ```bash
   openssl req -new -x509 -key <file-name>.key -out <certificate-name>.pem -days 1095
   ```
3. Enter the required details.
4. Create the PKCS 12 file using the GitHub App private key and CA signed certificate:
   
   ```bash
   openssl pkcs12 -export -in <certificate-name>.pem -inkey <file-name>.key -certfile <certificate-name>.pem -out <PKCS-12-file-name>.p12
   ```
5. Provide the export password.
6. Create the JKS file:
   
   ```bash
   keytool -importkeystore -srckeystore <PKCS-12-file-name>.p12 -srcstoretype pkcs12 -destkeystore <JKS-certificate-filename>.jks -deststoretype JKS
   ```
7. Provide the destination and source keystore passwords.

Attach the GitHub Java Key Store certificate to your instance

Enable the JWT Bearer Grant token authentication by attaching the valid GitHub Java KeyStore (JKS) certificate to your instance.

Before you begin

Ensure the availability of a valid Java KeyStore certificate.

Role required: admin

Procedure

1. Navigate to System Definition > Certificates.
2. Click New.
3. Complete the form.

X.509 Certificate form fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter a name to uniquely identify the record. For example, My GitHub App Certificate.</td>
</tr>
<tr>
<td>Notify on expiration</td>
<td>Define users to be notified when the certificate expires.</td>
</tr>
</tbody>
</table>
**Field** | **Description**
---|---
Warn in days to expire | Enter the number of days to send a notification before the certificate expires.
Active | Enable
Type | Select **Java Key Store**.
Expires in days | Enter the number of days until the certificate expires.
Key store password | Enter a password associated with the certificate (hint: the destination keystore password previously created).
Short description | Enter a summary about the certificate.

4. Click the attachments icon ( 📄 ) and attach a JKS certificate.
5. Click **Validate Stores/Certificates**.

**Create a JWT signing key for the GitHub JKS certificate**

Create a JSON Web Token (JWT) signing key to assign to your GitHub Java KeyStore certificate.

**Procedure**
1. Navigate to **System OAuth > JWT Keys**.
2. Click **New**.
3. Complete the form.

**JWT Keys form fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter a name to uniquely identify the JWT signing key. For example, <strong>My GitHub App JWT Key</strong>.</td>
</tr>
<tr>
<td>Signing Keystore</td>
<td>Select the valid JKS certificate attached in the previous task. For example, <strong>My GitHub App Certificate</strong>.</td>
</tr>
<tr>
<td>Key Id</td>
<td>Enter a key Id to identify which key is used when multiple keys are used to sign tokens.</td>
</tr>
<tr>
<td>Signing Algorithm</td>
<td>Select an algorithm to sign with the JWT key (hint: RSA 256).</td>
</tr>
</tbody>
</table>
Create a JWT provider for your GitHub signing key

Add a JSON Web Token (JWT) provider to your ServiceNow instance for GitHub.

**Procedure**

1. Navigate to **System OAuth > JWT Providers**.
2. Click **New**.
3. On the **JWT Provider** form, fill the values.

**JWT Provider form fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter a name to uniquely identify the JWT provider. For example, My GitHub App JWT Provider.</td>
</tr>
<tr>
<td>Expiry Interval (sec)</td>
<td>Enter a number in seconds to set the lifespan of JWT provider tokens (Hint: You can leave it as default).</td>
</tr>
<tr>
<td>Signing Configuration</td>
<td>Select the JWT signing key previously created. For example, My GitHub App JWT Key.</td>
</tr>
</tbody>
</table>

4. Right-click the form header, and click **Save**.

5. Enter your GitHub App **App ID** (available in the **About** section of your GitHub App configuration in GitHub) as the value of the **iss** claim, in the Standard Claims related list.

6. Leave **aud** and **sub** values blank (default).

**Register GitHub as an OAuth Provider (JWT)**

Use the information generated during GitHub App account configuration to register GitHub as an OAuth provider and allow the instance to request OAuth 2.0 tokens.
Procedure

1. Navigate to **System OAuth > Application Registry**.
2. Click **New**.
   The system displays the message **What kind of OAuth application?**
3. Select **Connect to a third party OAuth Provider**.
   The system displays a empty Application Registries form.
4. Complete the form.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter any name to uniquely identify the record. For example, enter My GitHub App Provider.</td>
</tr>
<tr>
<td>Client ID</td>
<td>Enter the client ID of your GitHub App (hint: available in the About section of your GitHub App configuration in GitHub).</td>
</tr>
<tr>
<td>Client Secret</td>
<td>Enter the client secret of your GitHub App (hint: available in the About section of your GitHub App configuration in GitHub).</td>
</tr>
<tr>
<td>OAuth API script</td>
<td>Select OAuthDevOpsGitHubHandler.</td>
</tr>
<tr>
<td>Default Grant type</td>
<td>Select JWT Bearer.</td>
</tr>
<tr>
<td>Authorization URL</td>
<td>Enter <a href="https://github.com/login/oauth/authorize">https://github.com/login/oauth/authorize</a>. For an on-premise deployment, use the proper GitHub host URL.</td>
</tr>
<tr>
<td>Token URL</td>
<td>Enter <a href="https://github.com/login/oauth/access_token">https://github.com/login/oauth/access_token</a>. For an on-premise deployment, use the proper GitHub host URL.</td>
</tr>
</tbody>
</table>

5. Leave the rest of the form fields as default.
6. Right-click the form header, and click **Save**.

- The system validates the OAuth credentials and populates the **Redirect URL** (Hint: It should match the User authorization callback URL previously provided in your GitHub App configuration).
- The system populates **OAuth Entity Profile** with **Grant Type** as JWT Bearer. For example, **OAuth Entity Profile** is created with default **Name**, **My GitHub App Provider default_profile**
7. Open the default profile created in the **OAuth Entity Profiles** related list.

8. Populate the **JWT Provider** field with the JWT provider previously created, uncheck the **Is default** flag and save the form.

9. Back on the **Application Registry** form, in the **OAuth Entity Profiles** related list, create a new OAuth Entity Profile with the **Is default** flag set to True, and **Grant type** set to Authorization code.

**Create a credential record for GitHub App provider (JWT)**

Create a credential record to the GitHub App provider previously created to authorize actions.

**Procedure**

1. Navigate to **Connections & Credentials > Credentials**.

2. Click **New**.
   The system displays the message **What type of Credentials would you like to create?**.

3. Select **OAuth 2.0 Credentials**.
   The pop-up window displays an empty OAuth 2.0 Credentials form.

4. Fill in these values.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter any name to uniquely identify the record. For example, enter My GitHub App Credential.</td>
</tr>
<tr>
<td>Active</td>
<td>Enable</td>
</tr>
<tr>
<td>OAuth Entity Profile</td>
<td>Select the default OAuth Entity profile you created where <strong>Grant type</strong> is Authorization Code.</td>
</tr>
<tr>
<td>Applies to</td>
<td>Select the MID Servers that can use this credential. For example, select <strong>All MID Servers</strong>.</td>
</tr>
<tr>
<td>Order</td>
<td>Select the order to apply this credential. For example, enter 100.</td>
</tr>
</tbody>
</table>

5. Save the record.

6. Click the **Get OAuth Token** related link to generate the OAuth token.
OAuth 2.0 credentials for GitHub Apps - Authorization Code

Starting with version 1.17, configure your GitHub account, register GitHub, and create a credential record for the GitHub App provider.

Role required: oauth_admin

Configure the GitHub App in your GitHub account (Authorization Code)

Create a custom GitHub App from your GitHub account to enable OAuth 2.0 authentication with your ServiceNow instance.

Before you begin
GitHub requirements:
• GitHub account
• GitHub App configured to integrate with ServiceNow

About this task
Complete these steps from your GitHub account. See Building GitHub Apps on the GitHub Developer site for instructions on creating and configuring custom applications.

Procedure
1. From your GitHub account, create your GitHub App by navigating to Developer Settings > GitHub Apps.
2. In the Homepage URL field, enter https://<instance-name>.service-now.com.
3. In the User authorization callback URL field, enter https://<instance-name>.service-now.com/oauth_redirect.do.
4. In the Identifying and authorizing users section, uncheck the Expire user authorization tokens field.
5. In the Webhook section, uncheck the Active field.
6. Leave the remaining fields empty (default).
7. In the Repository permissions section, configure these settings.

<table>
<thead>
<tr>
<th>Contents</th>
<th>Read-only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td>Read-only</td>
</tr>
<tr>
<td>Webhooks</td>
<td>Read and write</td>
</tr>
</tbody>
</table>

8. Leave the remaining permissions as No access (default).
9. Install the newly created GitHub App on the accounts of your choice.
Register GitHub as an OAuth Provider (Authorization Code)

Use the information generated during GitHub App account configuration to register GitHub as an OAuth provider and allow the instance to request OAuth 2.0 tokens.

Procedure

1. Navigate to System OAuth > Application Registry.
2. Click New.
   The system displays the message What kind of OAuth application?
3. Select Connect to a third party OAuth Provider.
   The system displays an empty Application Registries form.
4. Complete the form.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter any name to uniquely identify the record. For example, enter My GitHub App Provider.</td>
</tr>
<tr>
<td>Client ID</td>
<td>Enter the client ID of your GitHub App (hint: available in the About section of your GitHub App configuration in GitHub).</td>
</tr>
<tr>
<td>Client Secret</td>
<td>Enter the client secret of your GitHub App (hint: available in the About section of your GitHub App configuration in GitHub).</td>
</tr>
<tr>
<td>OAuth API script</td>
<td>Select OAuthDevOpsGitHubHandler.</td>
</tr>
<tr>
<td>Default Grant type</td>
<td>Select Authorization Code.</td>
</tr>
<tr>
<td>Authorization URL</td>
<td>Enter <a href="https://github.com/login/oauth/authorize">https://github.com/login/oauth/authorize</a>. For an on-premise deployment, use the proper GitHub host URL.</td>
</tr>
<tr>
<td>Token URL</td>
<td>Enter <a href="https://github.com/login/oauth/access_token">https://github.com/login/oauth/access_token</a>. For an on-premise deployment, use the proper GitHub host URL.</td>
</tr>
</tbody>
</table>

5. Leave the rest of the form fields as default.

6. Right-click the form header, and click Save.
• The system validates the OAuth credentials and populates the **Redirect URL** (Hint: It should match the **User authorization callback URL** previously provided in your GitHub App configuration).

• The system populates **OAuth Entity Profile** with **Grant Type** as **Authorization Code**. For example, **OAuth Entity Profile** is created with default **Name**, My GitHub App Provider **default_profile**

**Create a credential record for GitHub App provider (Authorization Code)**

Create a credential record to the GitHub App provider previously created to authorize actions.

**Procedure**

1. Navigate to **Connections & Credentials > Credentials**.

2. Click **New**.
   
   The system displays the message **What type of Credentials would you like to create?**.

3. Select **OAuth 2.0 Credentials**.
   
   The pop-up window displays an empty OAuth 2.0 Credentials form.

4. Fill in these values.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter any name to uniquely identify the record. For example, enter <strong>My GitHub App Credential</strong>.</td>
</tr>
<tr>
<td>Active</td>
<td>Enable</td>
</tr>
<tr>
<td>OAuth Entity Profile</td>
<td>Select the default OAuth Entity profile you created previously.</td>
</tr>
<tr>
<td>Applies to</td>
<td>Select the MID Servers that can use this credential. For example, select <strong>All MID Servers</strong>.</td>
</tr>
<tr>
<td>Order</td>
<td>Select the order to apply this credential. For example, enter <strong>100</strong>.</td>
</tr>
</tbody>
</table>

5. Save the record.

6. Click the **Get OAuth Token** related link to generate the OAuth token.

**OAuth 2.0 credentials for OAuth Apps**

Starting with version 1.17, configure your GitHub account, register GitHub, and create a credential record for the GitHub App Oath provider.
Role required: oauth_admin

Configure the OAuth App in your GitHub account
Create a custom OAuth App from your GitHub account to enable OAuth 2.0 authentication with your ServiceNow instance.

Before you begin
GitHub requirements:
• GitHub account
• GitHub OAuth App configured to integrate with ServiceNow

About this task
Complete these steps from your GitHub account. See Building OAuth Apps on the GitHub Developer site for instructions on creating and configuring custom applications.

Procedure
1. From your GitHub account, create your OAuth App by navigating to Developer Settings > OAuth Apps.
2. In the Homepage URL field, enter https://<instance-name>.service-now.com.

Register GitHub as an OAuth Provider
Use the information generated during GitHub OAuth App account configuration to register GitHub as an OAuth provider and allow the instance to request OAuth 2.0 tokens.

Procedure
1. Navigate to System OAuth > Application Registry.
2. Click New.
   The system displays the message What kind of OAuth application?
3. Select Connect to a third party OAuth Provider.
   The system displays an empty Application Registries form.
4. Complete the form.
<table>
<thead>
<tr>
<th>Field</th>
<th>Value required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter any name to uniquely identify the record. For example, enter My GitHub OAuth App Provider.</td>
</tr>
<tr>
<td>Client ID</td>
<td>Enter the client ID of your OAuth App.</td>
</tr>
<tr>
<td>Client Secret</td>
<td>Enter the client secret of your OAuth App.</td>
</tr>
<tr>
<td>OAuth API script</td>
<td>Select OAuthDevOpsGitHubHandler.</td>
</tr>
<tr>
<td>Default Grant type</td>
<td>Select Authorization Code.</td>
</tr>
<tr>
<td>Authorization URL</td>
<td>Enter <a href="https://github.com/login/oauth/authorize">https://github.com/login/oauth/authorize</a>. For an on-premise deployment, use the proper GitHub host URL.</td>
</tr>
<tr>
<td>Token URL</td>
<td>Enter <a href="https://github.com/login/oauth/access_token">https://github.com/login/oauth/access_token</a>. For an on-premise deployment, use the proper GitHub host URL.</td>
</tr>
</tbody>
</table>

5. Leave the rest of the form fields as default.

6. Right-click the form header, and click Save.
   - The system validates the OAuth credentials and populates the Redirect URL (Hint: It should match the Authorization callback URL previously provided in your OAuth App configuration).
   - The system populates OAuth Entity Profile with Grant Type as Authorization Code. For example, OAuth Entity Profile is created with default Name, My GitHub OAuth App Provider default_profile

7. In the OAuth Entity Scopes related list, create an entry where Name is set to scope and OAuth scope is set to repo, and save the form.

8. In the OAuth Entity Profiles related list, open the default profile.

9. In the OAuth Entity Profiles Scopes related list, select the OAuth Entity Scope you created, and save the form.

Create a credential record for OAuth App provider
Create a credential record to the GitHub App OAuth provider previously created to authorize actions.
Procedure
1. Navigate to **Connections & Credentials > Credentials**.

2. Click **New**.
   The system displays the message **What type of Credentials would you like to create?**.

3. Select **OAuth 2.0 Credentials**.
   The pop-up window displays an empty OAuth 2.0 Credentials form.

4. Fill in these values.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter any name to uniquely identify the record. For example, enter <strong>My OAuth App Credential</strong>.</td>
</tr>
<tr>
<td>Active</td>
<td>Enable</td>
</tr>
<tr>
<td>OAuth Entity Profile</td>
<td>Select the default OAuth Entity profile you created previously.</td>
</tr>
<tr>
<td>Applies to</td>
<td>Select the MID Servers that can use this credential. For example, select <strong>All MID Servers</strong>.</td>
</tr>
<tr>
<td>Order</td>
<td>Select the order to apply this credential. For example, enter <strong>100</strong>.</td>
</tr>
</tbody>
</table>

5. Save the record.

6. Click the **Get OAuth Token** related link to generate the OAuth token.

**GitLab integration with DevOps**
Integrate GitLab tools with DevOps using the DevOps Integrations application.

ℹ️ **Note:** You must install the **DevOps Integrations application** to integrate GitLab tools with DevOps.

GitLab tools supported:

* GitLab SCM (coding)
* GitLab CI/CD (orchestration) basic CI pipelines

ℹ️ **Note:** Multi-project pipelines are not supported.

Starting with version 1.24, tags are captured from GitLab coding tool commits.
Limitations

- A pipeline must run to completion one time before change control is enforced.

- If a manual job in GitLab is canceled, or times out before completion, the corresponding change remains in **Open** state until it is manually reconciled.

- Pipeline discovery is limited to the first 100 results using the current filter.
  
  To discover additional pipelines, modify the project filter to expand the results for the discovery request.

- Historical import of commits or events is not supported.
  
  Those created or updated after the project has been discovered and configured are tracked normally.

  GitLab Issues is not supported.

**Bulk commits in GitLab**

Starting with version 1.24, bulk commits are supported with GitLab.

To support large commits, perform these actions:

- Install the ServiceNow IntegrationHub Action Template - Data Stream (com.glide.hub.action_type.datastream) plugin.

- For optimal performance, disable flow logging by setting the Flow Designer `com.snc.process_flow.reporting.level` property to **Off**.

- For MID Server settings, view the MID Server support for Data Stream actions section.

  GitLab code push webhook sends a maximum of 20 commits in a notification.
  
  If the number of commits in the push are less than 20, a single inbound event is created and processed in the ServiceNow instance.

  If the number of pushed commits are equal or greater than 20, multiple inbound events are created from the original event with each new event containing a batch of 19 commits. The original inbound event is marked as ignored.

  Currently, GitLab Data stream action can process up to 10,000 commits in a single push.

**GitLab test tool integration**

Starting with version 1.23, JUnit test type integration is supported for GitLab.

Test tool integration lets you view test results in DevOps for GitLab unit, functional, and performance tests.
Connect DevOps to GitLab tools

Configure a connection and credential alias and create a GitLab tool record in DevOps.

Important: You can create a new tool and simultaneously create a new connection alias. Since the child alias is created during tool creation, you do not need an admin role nor do you need to create a new connection alias as a prerequisite. Create the new connection alias as an sn_devops.admin user using Connections and Credentials template which are dynamically populated based on the tool you select. For more information, see Create new DevOps tool from the Tools list view.

Configure connection and credential alias - GitLab

Before you set up a tool record in DevOps, your admin configures the DevOps CreateDevOpsTool connection and credential alias to allow access to the tools in your environment.

Before you begin

Starting with version 1.15, DevOps connection and credential aliases (DevOpsAlias1 through DevOpsAlias10) are used to connect automatically when you set up your DevOps planning, coding, and orchestration tools.


To connect to more than 10 tools, or if you receive an error saying all DevOpsAlias connection and credential aliases are being used, an admin can create additional aliases in the sn_devops application scope.

Important: If additional sys_alias records for DevOps are needed, they must be created by an admin in the sn_devops application scope.

You can reuse a DevOpsAlias alias no longer in use by inactivating the HTTP connection, if needed.

Role required: admin, sn_devops.admin

About this task

Your admin must configure an HTTP connection in the CreateDevOpsTool connection and credential alias provided, and add admin credentials to connect automatically when you set up your DevOps planning, coding, and orchestration tools:
• URL: https://<instance name>.service-now.com/

• Credentials: admin

ℹ️ Note: Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

You only need to configure the CreateDevOpsTool connection and credential alias once to set up all of your tools in DevOps.

The CreateDevOpsTool connection and credential alias is used with the DevOps Create a Tool subflow in Flow Designer. You can view executions in Flow Designer for information regarding flow results.

Procedure

1. Navigate to Connections & Credentials > Connection & Credential Aliases and open the CreateDevOpsTool record.

2. In the Connections related list, create a record and enter a Name for the connection.

3. On the Connection form, click the Credential field lookup list, and then click New to create an admin credential.

   a. Click Basic Auth Credentials and enter a Name.

   b. Enter admin username and password (required to access the tools in your DevOps environment).

      ℹ️ Note: Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

4. On the Connection form, enter https://<instance name>.service-now.com/ for the Connection URL.

Create a GitLab tool record in DevOps

Create a GitLab tool record in DevOps to connect, discover, and import GitLab tool data.

Before you begin
Role required: sn_devops.admin
About this task

• **Connect** to GitLab and get the webhook URL when you submit a DevOps tool record.
• **Discover** repositories, orchestration tasks, and pipelines.
• **Configure** webhooks in GitLab (starting with version 1.18).
• **Import** branch records, task execution and step execution records.

ℹ️ **Note:** Import of GitLab commits is not supported.

Procedure

1. Create a tool record in DevOps to automatically connect to GitLab and get the webhook URL.

   a. Navigate to DevOps > Tools > Create New and create a record.

   b. Enter a **Tool Name** and fill in the tool details.

<table>
<thead>
<tr>
<th>Tool Integration</th>
<th>GitLab</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool URL</td>
<td>GitLab tool URL.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td><a href="https://gitlab.com">https://gitlab.com</a></td>
</tr>
<tr>
<td>Tool Username</td>
<td>GitLab username</td>
</tr>
<tr>
<td>Tool Password / Access Token</td>
<td>GitLab access token</td>
</tr>
</tbody>
</table>

ℹ️ **Note:** Only personal access token is supported.

   If you are not granting complete access, select scope:

   api: Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.

**MID Server** is optional. Select MID Server for tools on premise attached to a MID Server. Application is automatically set to DevOps and capability is set to REST.
c. Click **Submit**.
   The tool is automatically **Connected Successfully** using a connection alias and HTTP tool connection (basic authentication credential), and an **API key credential** is created.

   **Note:** If you do not have global admin privileges for your tool (to allow automatic configuration of the webhook URL), you may need to have the tool admin user configure it for you (cut and paste the webhook URL into the tool configuration). Once the webhook is configured in the tool, **Enter Manual Configuration Mode** to connect to the tool manually, then exit.

2. Click **Discover** to discover tool configuration, including existing repositories, orchestration tasks, and pipelines, and fill in the fields in the project Search Filter window.

   **Note:** GitLab repos and pipelines discovered are determined by these filter conditions, and are limited to the first 100 results. To discover additional repos, modify the project filter to expand the results.

<table>
<thead>
<tr>
<th>Filter Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owned by me (recommended)</td>
<td>Searches for the repositories in the project that the current user owns.</td>
</tr>
<tr>
<td>Currently member of</td>
<td>Searches for the repositories in the project that the current user is a member of.</td>
</tr>
<tr>
<td>Search</td>
<td>Searches for the repositories in projects with the specified text string.</td>
</tr>
</tbody>
</table>

   **Note:** GitLab repositories and pipelines discovered are also determined by the credentials (level of access) configured in the connection.

   Records are added to the corresponding related lists.

3. Starting with version 1.18, automatically configure webhook URLs in GitLab.

   a. Click **Configure**.

   b. In the GitLab Configure window, select the applicable projects (containing existing repos and pipelines) to configure webhooks for, and click **Continue**.

   c. Enter your **Integration user** (DevOps Integration User) and **Password**.

      The DevOps Integration User is provided with the DevOps application and is used for inbound authentication from your tool to your ServiceNow instance.
Note: You must configure the password before the DevOps Integration User can configure a tool.

Notifications are sent from GitLab tools to DevOps as the DevOps Integration User.

The Track field is automatically set to True in configured repository records.

4. For discovered plans, import historical data from the tool and add an app to tie your tools together.

a. Open the plan record from the Plans related list and click the Import related link.
Records are created for the plan in the Work Items, Plan Versions, and Features related lists.
• Work items related list shows tasks, bugs, and stories.
  Native State and Native Type fields of the work item contain the original state and type values from the source tool.
• Plan Versions related list shows releases.
• Features related list shows epics and features.

b. In the Apps related list, click Edit... to select an App to associate with the plan (project), or click New to create one.

Note: A plan must have an associated App record to complete the planning tool setup.
Example:

**GitLab tool record in DevOps**

![GitLab tool record](image)

- **Name**: GitLab
- **Tool**: GitLab
- **Connection alias**: sn_devops DevOps Alias

**Tool URL**: [http://gitlab](http://gitlab)

**Webhook URL**: `https://<DevOps integration user ID>/<integration user password>@<password_name>.service-now.com/gitlab_devops/v1/devops/tool/4/codes/plan/artifact/orchestration/seg1?toolId=2997f8d39464e6b0a3f3a1c9f2f32329

What to do next

For GitLab pipelines, Model a GitLab basic CI pipeline in DevOps to complete the configuration in DevOps.

- Map each pipeline to a specific app.
- Create pipeline steps and map each step to a GitLab pipeline job.
- Configure change control.

Setting up GitLab OAuth 2.0 credentials for DevOps

Starting with version 1.30, integrate your GitLab account with your ServiceNow instance by creating a custom OAuth application in GitLab and authenticating requests from ServiceNow DevOps.

Configure your GitLab account, register GitLab in the application registry, and create a credential record for the GitLab App provider.

Role required: oauth_admin

**Configure the GitLab App in your GitLab account (Authorization Code)**

Create a custom GitLab App from your GitLab account to enable OAuth 2.0 authentication with your ServiceNow instance.
Before you begin
GitLab requirements:

• GitLab account
• GitLab App configured to integrate with ServiceNow

About this task
Complete these steps from your GitLab account. See GitLab as an OAuth 2.0 authentication service provider on the GitLab docs site for instructions on using GitLab applications for OAuth 2.0 authentication.

Procedure
1. From your GitLab account, create your GitLab App by navigating to Edit Profile > Applications.
2. In the Add new application form, specify a Name, and in Redirect URI field, enter https://<instance-name>.service-now.com/oauth_redirect.do.
3. In the Scopes section, ensure that you select the api check box.
4. Leave the remaining fields empty (default).
5. Click Save application.
   The application is created. You can open the application to access the Application ID, Secret key and Callback URL
6. Install the newly created GitLab App on the accounts of your choice.

Register GitLab as an OAuth Provider (Authorization Code)
Use the information generated during GitLab App account configuration to register GitLab as an OAuth provider and allow the instance to request OAuth 2.0 tokens.

Procedure
1. Navigate to System OAuth > Application Registry.
2. Click New.
   The system displays the message What kind of OAuth application?
3. Select Connect to a third party OAuth Provider.
   The system displays an empty Application Registries form.
4. Complete the form.
<table>
<thead>
<tr>
<th>Field</th>
<th>Value required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter any name to uniquely identify the record. For example, enter My GitLab App Provider.</td>
</tr>
<tr>
<td>Client ID</td>
<td>Enter the Application ID of your GitLab App.</td>
</tr>
<tr>
<td>Client Secret</td>
<td>Enter the secret key of your GitLab App.</td>
</tr>
<tr>
<td>OAuth API script</td>
<td>Select OAuthDevOpsGitLabHandler.</td>
</tr>
<tr>
<td>Default Grant type</td>
<td>Select Authorization Code.</td>
</tr>
<tr>
<td>Authorization URL</td>
<td>Enter <a href="https://gitlab.com/oauth/authorize">https://gitlab.com/oauth/authorize</a>. For an on-premise deployment, use the proper GitLab host URL.</td>
</tr>
<tr>
<td>Token URL</td>
<td>Enter <a href="https://gitlab.com/oauth/token">https://gitlab.com/oauth/token</a>. For an on-premise deployment, use the proper GitLab host URL.</td>
</tr>
</tbody>
</table>

5. Leave the rest of the form fields as default.

6. Right-click the form header, and click Save.

- The system validates the OAuth credentials and populates the Redirect URL (Hint: It should match the Redirect URI value previously provided in your GitLab App configuration).
- The system populates OAuth Entity Profile with Grant Type as Authorization Code. For example, OAuth Entity Profile is created with default Name, My GitLab App Provider default_profile

7. Validate that the OAuth Entity Scope related list contains the api scope.

Create a credential record for GitLab App provider (Authorization Code)
Create a credential record to the GitHub App provider previously created to authorize actions.

Procedure

1. Navigate to Connections & Credentials > Credentials.

2. Click New.
   - The system displays the message What type of Credentials would you like to create?.

© 2021 ServiceNow, Inc. All rights reserved.
ServiceNow, the ServiceNow logo, Now, and other ServiceNow marks are trademarks and/or registered trademarks of ServiceNow, Inc., in the United States and/or other countries. Other company names, product names, and logos may be trademarks of the respective companies with which they are associated.
3. Select **OAuth 2.0 Credentials**.
   The pop-up window displays an empty OAuth 2.0 Credentials form.

4. Fill in these values.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter any name to uniquely identify the record. For example, enter My GitLab App Credential.</td>
</tr>
<tr>
<td>Active</td>
<td>Enable</td>
</tr>
<tr>
<td>OAuth Entity Profile</td>
<td>Select the default OAuth Entity profile you created previously.</td>
</tr>
<tr>
<td>Applies to</td>
<td>Select the MID Servers that can use this credential. For example, select All MID Servers.</td>
</tr>
<tr>
<td>Order</td>
<td>Select the order to apply this credential. For example, enter 100.</td>
</tr>
</tbody>
</table>

5. Save the record.

6. Click the **Get OAuth Token** related link to generate the OAuth token.
   A successful token generation indicates that you can now authenticate connection between ServiceNow DevOps and GitLab via OAuth.

**Model a GitLab basic CI pipeline in DevOps**

Starting with version 1.16, model a GitLab basic CI pipeline by mapping the pipeline to an app, and mapping DevOps pipeline steps to GitLab pipeline jobs.

**Before you begin**
Role required: sn_devops.admin

**Procedure**

1. Map your pipeline to an app.
   a. Navigate to **DevOps > Apps & Pipelines > Apps** and open the application record to associate with the pipeline.
   
   b. In the Pipelines related list, click **Edit...** to select a pipeline to associate with the app, or click **New** to create the pipeline.
   For a new pipeline, fill in the **Orchestration pipeline** field using the group name, subgroup name (if applicable), and project name as specified in GitLab.
For example, My Group/My SubGroup/My Project.

If a project is not under a group, simply specify My Project.

c. Click Submit.

2. Open the pipeline record again and create DevOps steps to map to each GitLab pipeline job so an orchestration task can be created.

Steps can be created in one of the following ways.

- Starting with version 1.18, automatically create and map pipeline steps in DevOps by running your GitLab pipeline.

  Pipeline steps are automatically created, mapped, and associated when DevOps receives step notifications from your GitLab pipeline during the run.

- Manually create and map each pipeline step to a GitLab pipeline job.

  In the Steps related list, click New to create a DevOps step for each GitLab pipeline job (Orchestration stage field).

  **Note:** The Orchestration stage field value of each step is case-sensitive and must match the original name of the corresponding GitLab pipeline job.

<table>
<thead>
<tr>
<th>Name</th>
<th>Name of the pipeline step.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>Pipeline in which the step is configured.</td>
</tr>
<tr>
<td>Type</td>
<td>Pipeline step type.</td>
</tr>
<tr>
<td></td>
<td>◦ Build and Test</td>
</tr>
<tr>
<td></td>
<td>◦ Test</td>
</tr>
<tr>
<td></td>
<td>◦ Deploy</td>
</tr>
<tr>
<td></td>
<td>◦ Deploy and Test</td>
</tr>
<tr>
<td></td>
<td>◦ Manual</td>
</tr>
<tr>
<td></td>
<td>◦ Prod Deploy</td>
</tr>
<tr>
<td>Order</td>
<td>Order in which the steps are run.</td>
</tr>
<tr>
<td>Orchestration stage</td>
<td>GitLab pipeline job name (case-sensitive).</td>
</tr>
</tbody>
</table>

© 2021 ServiceNow, Inc. All rights reserved.

ServiceNow, the ServiceNow logo, Now, and other ServiceNow marks are trademarks and/or registered trademarks of ServiceNow, Inc., in the United States and/or other countries. Other company names, product names, and logos may be trademarks of the respective companies with which they are associated.
For step association with GitLab CI pipeline jobs, the **Orchestration stage** field must be configured.

<table>
<thead>
<tr>
<th>Business service</th>
<th>Configuration service that applies to the step.</th>
</tr>
</thead>
</table>

Once orchestration tasks are created, associate each orchestration task in the Orchestration Tasks related list with a DevOps pipeline step.

### 3. Optional: Select the Change control check box in a step to enable change acceleration and the corresponding configuration fields.

**Note:** ServiceNow Change Management must be installed for change acceleration.

<table>
<thead>
<tr>
<th>Change receipt</th>
<th>(Optional) Starting with version 1.20, select to enable change receipt for the step so the pipeline doesn’t pause when a change request is created. All pipeline data is included in the change, but approval is not required for the pipeline to proceed.</th>
</tr>
</thead>
</table>

| Change approval group | (Optional) Approval group for the change request. The change approval group becomes the **Assignment group** in the DevOps change request. **Note:** Ensure that the selected group has members and a group manager so approver field is not empty. |

<table>
<thead>
<tr>
<th>Change type</th>
<th>Change request type to create.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Normal (default)</td>
</tr>
<tr>
<td></td>
<td>• Standard</td>
</tr>
<tr>
<td></td>
<td>• Emergency</td>
</tr>
</tbody>
</table>

| (Optional) Template | (Optional) List of templates to use to auto populate fields for Normal or Emergency change requests. |

© 2021 ServiceNow, Inc. All rights reserved. ServiceNow, the ServiceNow logo, Now, and other ServiceNow marks are trademarks and/or registered trademarks of ServiceNow, Inc., in the United States and/or other countries. Other company names, product names, and logos may be trademarks of the respective companies with which they are associated.
Select a template or create a new one.

(Optional) Standard change template

List of standard change templates to use for Standard change requests.

Change controlled branches

(Optional) (Multibranch only) Comma-separated list of branches under change control. Wildcards are supported.

You can set up change control in GitLab for manual GitLab jobs.

4. For versions 1.17 and earlier, navigate to DevOps > Tools > Orchestration Tools and in the GitLab tool record, copy the DevOps Webhook URL field value. The webhook URL contains the DevOps location for GitLab CI pipelines to send messages, including the sys_id for the tool:

   http://<devops.integration.user>:<integration user password>@<your-instance>.service-now.com/api/sn_devops/v1/devops/tool/{code | plan | artifact | orchestration | test}?toolId={sys_id of the GitLab tool record}.

Example:

Configure the GitLab pipeline for DevOps

Configuring the GitLab pipeline for DevOps

Webhooks are required in GitLab to send job and push notifications to DevOps. Change control can be configured in GitLab for a manual job.
Configure webhooks in GitLab

Version 1.17 and earlier, configure webhooks in GitLab to send job and push notifications to the DevOps application.

Before you begin
Verify that you have copied the Webhook URL field from the DevOps GitLab tool form into your clipboard.
Role required: sn_devops.admin

About this task
Create webhooks for these two triggers in GitLab for every project you want to track.
• Push events
• Job events

In GitLab, navigate to Project > Settings > Webhooks to create webhooks.

Procedure
1. Create a webhook for push events in GitLab.

   a. Paste the DevOps GitLab tool webhook in the URL field, and modify the URL to specify only the code parameter.
      That is, replace {code | plan | artifact | orchestration | test} with code.
      For example:
      http://<devops.integration.user>:<integration user password>@<your-instance>.service-now.com/api/sn_devops/v1/devops/tool/code?toolId={sys_id of the GitLab tool record}.

      i Note: Your integration username and password must be specified in the URL.
b. Select the **Push events** check box.

2. Create a webhook for job events in GitLab.

   a. For job events, paste the DevOps GitLab tool webhook in the **URL** field, and modify the URL to specify only the `orchestration` parameter. That is, replace `{code | plan | artifact | orchestration | test}` with `orchestration`.

   For example:

   ```
   http://<devops.integration.user>:<integration user password>@<your-instance>.service-now.com/api/sn_devops/v1/devops/tool/orchestration?
   toolId={sys_id of the GitLab tool record}.
   ```

   **Note:** Your integration username and password must be specified in the URL.
b. Select the **Job events** check box.

**Example:**

**GitLab webhooks for DevOps integration**

- **Project Hooks (3)**
  - [URL](https://devops.integration.user:_____@_____.service-now.com/api/sn_devops/v1/devops/tool/orchestration?toolId=7f8806b4d6e490101d0455535e961926)
    - Job Events
    - SSL Verification: enabled
  - [URL](https://devops.integration.user:_____@_____.service-now.com/api/sn_devops/v1/devops/tool/code?toolId=7f8806b4d6e490101d0455535e961926)
    - Push Events
    - SSL Verification: enabled
Change acceleration in GitLab

Change acceleration is supported in DevOps for manual GitLab jobs. The GitLab job under change control must have these instructions for the pipeline execution to be resumed or canceled via the change request:

- `when: manual`
- `allow_failure: false`

For example:

```
deploy:
  stage: deploy
  tags:
    - local-runner1
  when: manual
  allow_failure: false
  script:
    - echo 'Deploy'
```

Refer to GitLab CI/CD pipeline configuration reference for more information on how to configure a GitLab job.

Additional considerations:

- If `allow_failure` is set to `true`, the pipeline continues even when the change is rejected.
- A user with the appropriate role access in GitLab can unblock and continue a pipeline regardless of the change request state.

<table>
<thead>
<tr>
<th>Manual execution</th>
<th>Change acceleration in step</th>
<th>Change request approved</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>If the manual job is under change control, the change is automatically created.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Yes</td>
<td>The manual job is automatically executed.</td>
</tr>
</tbody>
</table>
### GitLab change acceleration behavior (continued)

<table>
<thead>
<tr>
<th>Manual execution</th>
<th>Change acceleration in step</th>
<th>Change request approved</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
<td>The manual job is automatically rejected/failed.</td>
</tr>
<tr>
<td>No</td>
<td>N/A</td>
<td>No</td>
<td>The manual job waits for manual intervention from the pipeline owner via the GitLab UI (default behavior).</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
<td>The job runs, and the change request is created with the applicable related list data. However, approval or rejection of the change request has no impact on the pipeline run.</td>
</tr>
</tbody>
</table>

**Note:** Parallel jobs are displayed sequentially, based on the order in which the jobs are queued for execution.

### Associate GitLab pipeline steps in DevOps

For manually created DevOps steps, associate each orchestration task in the Orchestration Tasks related list with a DevOps pipeline step to track the activity of each stage in your GitLab pipeline.

**Before you begin**

Role required: sn_devops.admin
Procedure

1. Navigate to **DevOps > Tools > Orchestration Tools**, and open the GitLab tool record.

2. In the Orchestration Tasks related list, enter the corresponding DevOps pipeline step in the **Step** field of each orchestration task.

   **Note:** The **Track** field is set to **True** by default when you discover orchestration tasks and pipelines. Tracking is required to receive job notifications from GitLab.

GitLab pipeline job run notifications are sent to the DevOps application. Each task execution notification corresponds to an orchestration task and, since orchestration tasks are mapped to a certain step in your DevOps pipeline, you can track the activity in each stage of your pipeline.

**Example:**

DevOps associate steps

<table>
<thead>
<tr>
<th>DevOps Tool</th>
<th>GitLab</th>
<th>Connection state</th>
<th>Last discovery</th>
<th>Configuration Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Gitlab</td>
<td>Connected</td>
<td>2020-07-24 22:04:50</td>
<td></td>
</tr>
<tr>
<td>Tool</td>
<td>GitLab</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tool URL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Webhook URL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Jenkins integration with DevOps

Integrate Jenkins orchestration tool with DevOps using the DevOps Integrations application.

**Note:** You must install the **DevOps Integrations application** and the Jenkins plugin for ServiceNow DevOps to integrate Jenkins with DevOps.
Connecting DevOps to Jenkins requires configuring a connection and credential alias, and creating a Jenkins tool record in DevOps.

Setting up the DevOps connection for Jenkins
Once you have connected DevOps to Jenkins, complete the integration by configuring the Jenkins pipeline for DevOps, and modeling the Jenkins pipeline in DevOps.

Modeling the Jenkins scripted pipeline in Devops

**Jenkins plugin for ServiceNow DevOps**
The Jenkins plugin for ServiceNow DevOps is provided to enable change acceleration so your orchestration tool can communicate with ServiceNow DevOps and control certain aspects of pipeline executions.

Visit the Ancillary Software section on the ServiceNow Store website to download the Jenkins plugin for ServiceNow DevOps.

**ServiceNowDevOps Jenkins plugin on the Jenkins Marketplace**
The ServiceNowDevOps Jenkins plugin is provided to enable change acceleration so your orchestration tool can communicate with ServiceNow DevOps and control certain aspects of pipeline executions. Starting with DevOps version 1.29, the plugin is available in the Jenkins Marketplace.

• Navigate to **Manage Jenkins > Manage Plugins > Available** tab, search for and install the ServiceNowDevOps plugin.

**Set DevOps change request details**
Starting with version 1.24, set closure code and change request fields from within the Jenkins pipeline.

**Jenkins test tool integration**
Starting with version 1.15, JUnit test type integration is supported for Jenkins.

Test tool integration lets you view test results in DevOps for Jenkins unit, functional, and performance tests.

**Connect DevOps to Jenkins**
Configure a connection and credential alias and create a Jenkins tool record in DevOps.
Note: You can create a new tool and simultaneously create a new connection alias. Since the child alias is created during tool creation, you do not need an admin role nor do you need to create a new connection alias as a prerequisite. Create the new connection alias as an sn_devops.admin user using Connections and Credentials template which are dynamically populated based on the tool you select. For more information, see Create new DevOps tool from the Tools list view.

Configure connection and credential alias - Jenkins

Before you set up a tool record in DevOps, your admin configures the DevOps CreateDevOpsTool connection and credential alias to allow access to the tools in your environment.

Before you begin
Starting with version 1.15, DevOps connection and credential aliases (DevOpsAlias1 through DevOpsAlias10) are used to connect automatically when you set up your DevOps planning, coding, and orchestration tools.


To connect to more than 10 tools, or if you receive an error saying all DevOpsAlias connection and credential aliases are being used, an admin can create additional aliases in the sn_devops application scope.

Note: If additional sys_alias records for DevOps are needed, they must be created by an admin in the sn_devops application scope.

You can reuse a DevOpsAlias alias no longer in use by inactivating the HTTP connection, if needed.

Role required: admin, sn_devops.admin

About this task
Your admin must configure an HTTP connection in the CreateDevOpsTool connection and credential alias provided, and add admin credentials to connect automatically when you set up your DevOps planning, coding, and orchestration tools:

• URL: https://<instance name>.service-now.com/
• Credentials: admin

Note: Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.
You only need to configure the CreateDevOpsTool connection and credential alias once to set up all of your tools in DevOps.

The CreateDevOpsTool connection and credential alias is used with the DevOps Create a Tool subflow in Flow Designer. You can view executions in Flow Designer for information regarding flow results.

**Procedure**

1. Navigate to **Connections & Credentials > Connection & Credential Aliases** and open the CreateDevOpsTool record.
2. In the Connections related list, create a record and enter a **Name** for the connection.
3. On the Connection form, click the **Credential** field lookup list, and then click **New** to create an admin credential.
   a. Click **Basic Auth Credentials** and enter a **Name**.
   b. Enter admin username and password (required to access the tools in your DevOps environment).

   **Note:** Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

4. On the Connection form, enter `https://<instance name>.service-now.com/` for the **Connection URL**.

**Create a Jenkins tool record in DevOps**

Create a Jenkins tool record in DevOps to connect, discover, and import orchestration tasks and pipelines.

**Before you begin**

Role required: sn_devops.admin

**About this task**

Actions:

- **Connect** to Jenkins and get the webhook URL when you submit a DevOps tool record.
- **Discover** orchestration tasks and pipelines.
- **Import** task execution and step execution records.
Note: Starting from version 1.28, you can authenticate your connection with Jenkins using Jenkins API tokens. For more information, see Connect to Jenkins using API token authentication.

Procedure

1. Create a tool record in DevOps to automatically connect to Jenkins and get the webhook URL.

   a. Navigate to DevOps > Tools > Create New and create a record.

   b. Enter a Tool Name and fill in the tool details.

      Note: Freestyle project, Pipeline, Multibranch Pipeline, and Folder type jobs are supported. Default number of levels for Folder type is three.

<table>
<thead>
<tr>
<th>Tool Integration</th>
<th>Jenkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool URL</td>
<td>Jenkins tool URL</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td><a href="https://jenkins.com">https://jenkins.com</a></td>
</tr>
<tr>
<td>Tool Username</td>
<td>Jenkins user name</td>
</tr>
<tr>
<td>Tool Password / Access Token</td>
<td>Jenkins password, access token or the API token you generate.</td>
</tr>
</tbody>
</table>

MID Server is optional. Select MID Server for tools on premise attached to a MID Server. Application is automatically set to DevOps and capability is set to REST.

   c. Click Submit.

   Note: Only one Jenkins connection per connection alias is supported. You must create a new connection alias for a second Jenkins connection.
The tool is automatically **Connected Successfully** using a connection alias, and HTTP tool connection (basic authentication credential).

**Note:** If you do not have global admin privileges for your tool (to allow automatic configuration of the webhook URL), you may need to have the tool admin user configure it for you (cut and paste the webhook URL into the tool configuration). Once the webhook is configured in the tool, **Enter Manual Configuration Mode** to connect to the tool manually, then exit.

2. Click **Discover** to discover the existing orchestration tasks (Jenkins stages) and pipelines. Records are added to the corresponding related lists.

3. Open a discovered record from the Orchestration Tasks related list and click the **Import** related link to import historical data from the orchestration task. Imported task execution records and step execution records are added to the corresponding related lists.

**Example:**

**Jenkins tool record in DevOps**

![Jenkins tool record example](image)

**Connect to Jenkins using API token authentication**

Connect to Jenkins using API token authentication instead of user name and password.
Before you begin
Role required:
• Jenkins: admin or any user with overall Read and Job Read roles
• ServiceNowDevOps: sn_devops.admin in

About this task
Use Jenkins API tokens to connect JenkinsDevOps to Jenkins or create a new Jenkins tool in ServiceNowDevOps.

Procedure
1. In the Jenkins banner frame, click your user name to open the user menu.
   a. Navigate to **Your Username > Configure > API Token**.
   b. Click **Add new Token**.
   c. Click **Generate**.
   d. Copy the API token that is generated
2. In the ServiceNowDevOps > **Create Tool** form, use this as the password when you login to Jenkins or connect or create a new Jenkins.

Note:
• Regenerate the tokens every 6 months (depending on your context). Jenkins displays an indicator concerning the age of the token
• Use a different token for each application so that if an application is compromised you can revoke its token individually.
• If your token expires, regenerate the token and update it in the Now Platform instance
• ServiceNowDevOps DevOps does not support using Legacy API Tokens since Jenkins does not recommend the use of Legacy API Token. For more information, see the [Jenkins blog post](#).

What to do next
**Model a Jenkins pipeline in DevOps** to complete the configuration in DevOps.
• Map each pipeline to a specific app.
• Create pipeline steps and map each step to a Jenkins pipeline stage.
• Configure change control.
Model a Jenkins pipeline in DevOps

Model a Jenkins pipeline by mapping the pipeline to an app, and mapping DevOps pipeline steps to Jenkins stages.

Before you begin
The Jenkins plugin for ServiceNow DevOps is provided to enable change acceleration so your orchestration tool can communicate with ServiceNow DevOps and control certain aspects of pipeline executions.

Visit the Ancillary Software section on the ServiceNow Store website to download the Jenkins plugin for ServiceNow DevOps.

Role required: sn_devops.admin

About this task
Both scripted pipelines (Jenkinsfile) and freestyle jobs are supported.

For Jenkinsfile pipelines, pipeline steps are created, mapped, and associated to orchestration tasks automatically, instead of manually.

Procedure
1. Map your pipeline to an app in DevOps.

   a. Navigate to DevOps > Apps & Pipelines > Apps and open the application record to associate with the pipeline.

   b. In the Pipelines related list, click Edit... to select a pipeline to associate with the app, or click New to create the pipeline.
      For a new pipeline, fill in the Orchestration pipeline field using the full project name as specified in Jenkins.

   c. Click Submit.

2. Open the pipeline record again and select the Track check box so events from the pipeline are received.

   Note: The Track check box must be selected to integrate the pipeline with DevOps.

3. Create DevOps steps to map to each Jenkins pipeline stage so an orchestration task can be created.
   Steps can be created in one of the following ways.
Starting with version 1.19, for declarative or scripted pipelines, run your Jenkins pipeline to automatically create and map pipeline steps in DevOps. Pipeline steps are automatically created, mapped, and associated when DevOps receives step notifications from your Jenkins pipeline during the run.

For freestyle jobs, manually create and map each pipeline step to a Jenkins pipeline job.

In the Steps related list, click New to create a DevOps step for each Jenkins pipeline stage (Orchestration stage field)

**Note:** The **Orchestration stage** field value of each step is case-sensitive and must match the original name of the corresponding Jenkins pipeline stage.

<table>
<thead>
<tr>
<th>Name</th>
<th>Name of the pipeline step.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>Pipeline in which the step is configured.</td>
</tr>
<tr>
<td>Type</td>
<td>Pipeline step type.</td>
</tr>
<tr>
<td></td>
<td>◦ Build and Test</td>
</tr>
<tr>
<td></td>
<td>◦ Test</td>
</tr>
<tr>
<td></td>
<td>◦ Deploy</td>
</tr>
<tr>
<td></td>
<td>◦ Deploy and Test</td>
</tr>
<tr>
<td></td>
<td>◦ Manual</td>
</tr>
<tr>
<td></td>
<td>◦ Prod Deploy</td>
</tr>
<tr>
<td>Order</td>
<td>Order in which the steps are run.</td>
</tr>
</tbody>
</table>

**Note:** The step order determines the order of the cards in the Pipeline UI.

Starting with version 1.18, the order of the cards in the Pipeline UI is by task execution.

<table>
<thead>
<tr>
<th>Orchestration stage</th>
<th>Jenkins pipeline stage name (case-sensitive).</th>
</tr>
</thead>
</table>

**Note:** For step association with Jenkins pipeline stages, the **Orchestration stage** field must be configured.
Once orchestration tasks are created, associate each orchestration task in the Orchestration Tasks related list with a DevOps pipeline step.

4. **Optional:** Select the **Change control** check box in a step to enable change acceleration and the corresponding configuration fields.

⚠️ **Note:** ServiceNow Change Management must be installed for change acceleration.

<table>
<thead>
<tr>
<th><strong>Change receipt</strong></th>
<th>(Optional) Starting with version 1.20, select to enable change receipt for the step so the pipeline doesn’t pause when a change request is created. All pipeline data is included in the change, but approval is not required for the pipeline to proceed.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Change approval group</strong></td>
<td>(Optional) Approval group for the change request. The change approval group becomes the Assignment group in the DevOps change request.</td>
</tr>
</tbody>
</table>
| **Change type** | Change request type to create.  
• Normal (default)  
• Standard  
• Emergency | |
| **(Optional) Template** | (Optional) List of templates to use to auto populate fields for Normal or Emergency change requests. Select a template or create a new one. | ➤ **Note:** This field is shown only when Change type is Normal or Emergency. |
| **(Optional) Standard change template** | List of standard change templates to use for Standard change requests. | ➤ **Note:** This field is shown only when Change type is Standard. |
Change controlled branches

(Optional) (Multibranch only) Comma-separated list of branches under change control. Wildcards are supported.

Note: This field is required for Standard change type.

5. Navigate to DevOps > Tools > Orchestration Tools and in the Jenkins tool record, copy the DevOps Webhook URL field value. The webhook URL contains the DevOps location for Jenkins to send messages, including the sys_id for the tool:

https://<devops.integration.user>:<password>@<your-instance>.service-now.com/api/sn_devops/v1/devops/tool/event/{sys_id of the record}

Example:
**DevOps pipeline**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Order</th>
<th>Change control</th>
<th>Change approval group</th>
<th>Configuration Item</th>
<th>Change controlled branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>CorpSite: CI</td>
<td>Build and Test</td>
<td>100</td>
<td>false</td>
<td>(empty)</td>
<td>(empty)</td>
<td>*</td>
</tr>
<tr>
<td>CorpSite: UAT</td>
<td>Deploy</td>
<td>200</td>
<td>false</td>
<td>(empty)</td>
<td>(empty)</td>
<td>*</td>
</tr>
<tr>
<td>CorpSite: UAT Test</td>
<td>Test</td>
<td>300</td>
<td>false</td>
<td>(empty)</td>
<td>(empty)</td>
<td>*</td>
</tr>
<tr>
<td>CorpSite: Prod</td>
<td>Prod Deploy</td>
<td>400</td>
<td>true</td>
<td>Change Management</td>
<td>(empty)</td>
<td>*</td>
</tr>
</tbody>
</table>

**DevOps pipeline step**

<table>
<thead>
<tr>
<th>Name</th>
<th>Pipeline</th>
<th>Type</th>
<th>Order</th>
<th>Orchestration stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prod</td>
<td>CorpSiteCD</td>
<td>Prod Deploy</td>
<td>20</td>
<td>Prod</td>
</tr>
</tbody>
</table>

**What to do next**

Configure the Jenkins pipeline for DevOps

**Configuring the Jenkins pipeline for DevOps**

Jenkins tool integration includes configuring the Jenkins pipeline.

**Configure Jenkins to send notifications to DevOps**

Configure Jenkins to send build notifications to the DevOps application.
Before you begin
Role required: admin or user with overall Read and Job-Read access in Jenkins, sn_devops.admin

About this task
You need to configure your Jenkins instance to be able to securely connect and communicate with ServiceNow DevOps. Starting from version 1.28, you can use Jenkins API tokens to authenticate your connection with Jenkins. For more information, see Connect to Jenkins using API token authentication

Procedure
1. In Jenkins, navigate to Manage Jenkins > Configure System, select the ServiceNow DevOps Enabled check box in the ServiceNow DevOps Configuration section, and fill in the fields.
When enabled, Jenkins starts sending events to DevOps as inbound events.

i Note: Values are case-sensitive.

<table>
<thead>
<tr>
<th>Field</th>
<th>URL format/value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance URL</td>
<td>https://&lt;your-instance&gt;.service-now.com</td>
</tr>
<tr>
<td>API Version</td>
<td>v1</td>
</tr>
<tr>
<td>Orchestration Tool ID</td>
<td>The sys_id of the orchestration tool. You can copy this value from the webhook URL (toolid value), or obtain it directly using the Copy sys_id command on the Orchestration Tool form.</td>
</tr>
<tr>
<td>Artifact Tool ID</td>
<td>The sys_id of the artifact tool.</td>
</tr>
<tr>
<td>Username</td>
<td>• devops.integration.user</td>
</tr>
<tr>
<td></td>
<td>• The user name you specify while generating the API token.</td>
</tr>
<tr>
<td>Field</td>
<td>URL format/value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>• Password for DevOps Integration User.</td>
</tr>
<tr>
<td></td>
<td>• The API token that you generated and copied. For more information see, Connect to Jenkins using API token authentication</td>
</tr>
<tr>
<td><strong>Debug Mode</strong></td>
<td>Select check box.</td>
</tr>
<tr>
<td></td>
<td>This field appears only until Jenkins plugin version 1.28.</td>
</tr>
<tr>
<td><strong>Log Level</strong></td>
<td>The level of log messages you want to capture in the ServiceNow log recorder.</td>
</tr>
<tr>
<td></td>
<td>Select from one of the following options:</td>
</tr>
<tr>
<td></td>
<td>• inherit</td>
</tr>
<tr>
<td></td>
<td>• off</td>
</tr>
<tr>
<td></td>
<td>• severe</td>
</tr>
<tr>
<td></td>
<td>• info</td>
</tr>
<tr>
<td></td>
<td>• config</td>
</tr>
<tr>
<td></td>
<td>• fine</td>
</tr>
<tr>
<td></td>
<td>• finer</td>
</tr>
<tr>
<td></td>
<td>• finest</td>
</tr>
<tr>
<td></td>
<td>• all</td>
</tr>
<tr>
<td></td>
<td>This field appears only after you upgrade to the Jenkins plugin version 1.29.</td>
</tr>
<tr>
<td></td>
<td>For more information, see ServiceNow log levels and Log Recorders</td>
</tr>
<tr>
<td><strong>Force Tracking Check</strong></td>
<td>• Select the check box to make a REST API call to Jenkins for each pipeline execution to determine whether the pipeline is tracked or not.</td>
</tr>
</tbody>
</table>
2. Click **Test Connection**.
   Verify that the connection successful message displays.

3. **Optional:** Set the ServiceNow job-level properties in the **View Configuration > General** tab for the job.

<table>
<thead>
<tr>
<th>Field</th>
<th>URL format/value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear the check box to store details in the snPipelineInfo.json file and stop making API calls for every pipeline execution. By default, this check box is selected. For more information, see Reduce calls from Jenkins to ServiceNow DevOps to fetch pipeline information. This field appears only after you upgrade to the Jenkins plugin version 1.29.</td>
<td></td>
</tr>
<tr>
<td>ServiceNow DevOps Enabled</td>
<td>(Optional) Clear the check box to disable build notifications for a specific job. Default is selected.</td>
</tr>
<tr>
<td><strong>Note:</strong> Starting with version 1.19, this option is no longer applicable because the build notification setting is configured on the DevOps Pipeline form while modeling the Jenkins pipeline (Track check box).</td>
<td></td>
</tr>
<tr>
<td>Ignore ServiceNow DevOps errors</td>
<td>Select the check box to prevent job failure if there is an error, incorrect snDevOpsStep ID, for example. Default is not selected.</td>
</tr>
<tr>
<td>Change Request Details (only used for freestyle/maven projects)</td>
<td>Starting with version 1.24, set closure code and change request fields from within the pipeline.</td>
</tr>
</tbody>
</table>

**Note:** You can set these properties in a scripted or declarative pipeline, but if they are also set here, these check boxes take precedence.
4. For change control, navigate to Manage Jenkins > Configure System > Jenkins Location, and provide the host name in the Jenkins URL field.

**Note:** The Jenkins URL location is required for change acceleration functionality.

Example:

**Set up Jenkins to send notifications to DevOps**

**Job-level settings example (starting with version 1.24, change request details)**

**Jenkins location**

**Reduce calls from Jenkins to ServiceNow DevOps to fetch pipeline information**

Enable the Force Tracking Check field in the Jenkins configuration form to create a pipeline tracking file in Jenkins. ServiceNowDevOps makes a REST call to Jenkins to update the tracking file when the Track field in a pipeline is modified.
Force Tracking Check
Starting with the ServiceNow DevOps Jenkins plugin 1.29, a new **Force Tracking Check** check box is introduced in the ServiceNowDevOps configuration section in Jenkins to reduce the number of calls made from Jenkins to ServiceNowDevOps to fetch pipeline information such as pipelines being tracked. The following base system flows are also added as part of the 1.29 plugin:

- DevOps Jenkins File Update- Track flow
- DevOps Jenkins File Update- Test Info flow

How it works
In previous versions, a REST call fetched the pipeline information for every Jenkins build triggered. If you had multiple pipelines in your Jenkins environment and were tracking only a few of them, this meant that a call was made to fetch the tracking information for each pipeline even if you were tracking a few of them.

After you upgrade to DevOps version 1.29, clear the **Force Tracking Check** check box in the Jenkins > ServiceNowDevOps configuration section. The first time you trigger a Jenkins build or pipeline execution, Jenkins makes a pipeline information API call and creates `snPipelineInfo.json` file in `/{JENKINS_HOME}/jobs/{jobName}` directory. For each subsequent pipeline execution Jenkins checks the information available in the `snPipelineInfo.json` file before making a pipeline info API call.

If you disable the **Force Track Check** check box:

- The DevOps Jenkins File Update- Track flow triggers when you update the **Track** field on the pipeline form. The **Track** field info is updated in the `snPipelineInfo.json` file.
- DevOps Jenkins File Update- Test Info flow triggers when you update Test type mapping for Jenkins tool integration and verify that Test info is updated in `snPipelineInfo.json` file.

If you enable the **Force Track Check** check box:

- Jenkins makes pipeline info API calls to ServiceNow DevOps even if track/test information exists in the `snPipelineInfo.json` file.

Using a declarative or scripted pipeline in DevOps
When you use a Jenkinsfile, steps are created, mapped, and associated to orchestration tasks automatically instead of manually.

Jenkinsfile is a text file that contains the definition of a Jenkins pipeline and is checked into source control.
Each root-level stage configured in the Jenkinsfile is discovered as a separate orchestration task in DevOps that is mapped to an individual step.

⚠️ Note: The Track field for the pipeline must be set to True in DevOps to receive job notifications from Jenkins.

**DevOps Jenkinsfile commands**

⚠️ Note: Starting with version 1.19, the `snDevOpsStep` command is no longer required to set up Jenkins pipeline steps because the steps are created automatically when the pipeline is run.

- `snDevOpsStep(enabled:{true/false}, ignoreErrors:{true/false})`
  Where:
  - `enabled` specifies the setting for build notifications property (true/false)
  - `ignoreErrors` specifies the setting to prevent job failure if there is an error (true/false)

  ⚠️ Note: Step properties can also be set using check boxes in Jenkins (job configuration), which take precedence over the values specified here.

- `snDevOpsChange(ignoreErrors:{true/false}, changeRequestDetails:{setCloseCode:{true/false}, attributes:})`
  Where `ignoreErrors` specifies the setting to prevent job failure if there is an error (true/false)
  Where `changeRequestDetails` specifies closure code and change request fields from within the pipeline (starting with version 1.24)

Enables change control for each root-level stage that is mapped to a DevOps step.

- Starting with version 1.11, `snDevOpsArtifact`
  Registers artifacts when configuring DevOps change acceleration for releases.

- Starting with version 1.11, `snDevOpsPackage`
  Creates a package for artifacts when configuring DevOps change acceleration for releases.

⚠️ Note: Stage mapping is only supported for stages at the root level, not nested or parallel stages.

**Example Jenkinsfile**

```java
node {
}
```
stage("build") {
    snDevOpsStep(enabled:false)
}

stage("test") {
    snDevOpsStep(ignoreErrors:false)
}

stage("deploy") {
    snDevOpsStep()
    snDevOpsChange()
}

Change request details example (starting with version 1.24)

```python
stage('Deploy'){
    agent any
    steps{
        snDevOpsChange(changeRequestDetails: ""
        "setCloseCode": false,
        "attributes": {
            "sys_created_by": "",
            "sys_updated_by": "",
            "watch_list": [{
                "name": "DevOps System"
            }, ""],
            "work_notes_list": [""],
            "category": "Service",
            "sys_created_on": "2021-02-09 18:58:41",
            "priority": "2",
            "work_start": "2021-01-05 08:00:00",
            "work_end": "2021-01-08 08:00:00",
            "comments": "This update for work notes is from jenkins file",
            "work_notes": "Update this to work_notes",
            "start_date": "2021-01-05 08:00:00",
            "end_date": "2021-01-08 08:00:00"
        }
    }
}
```

ℹ️ Note: Historical import of a declarative or scripted pipeline is not supported.
Parallel and sub-stage support

When a stage (or set of parallel stages) is nested within a pipeline stage defined by the `snDevOpsStep()` command, these rules apply:

- Any action from the nested stage is processed as part of the parent root-level stage
- Only one change request is created (at the parent root level) even if multiple stages nested under the parent root-level stage trigger a change
- Orchestration tasks created are always associated with the parent root-level stage (not the nested stage)

Example: Sub stage

In this sub-stage example, if a change request gets created from the sub stage (deploy PROD), the details of the parent root-level stage (deploy) are used in the change request, and orchestration tasks are also associated with the parent root-level stage (deploy).

```java
stage("deploy") {
    stages{
        stage('deploy UAT') {
            when{
                branch 'dev'
            }
            steps{
                snDevOpsStep ()
                echo "deploy in UAT"
            }
        }
        stage('deploy PROD') {
            when {
                branch 'master'
            }
            steps{
                snDevOpsStep ()
                echo "deploy in prod"
                snDevOpsChange()
            }
        }
    }
}
```
Example: Parallel stage

In this parallel stage example, if a change request is created from a sub stage (UAT test-1 and/or UAT static code test), only the first change request is created (using the details of the parent root-level stage, UAT test) regardless of whether both sub stages (UAT test-1 and UAT static code test) get triggered.

There is no indication of which parallel stage generated the change, and orchestration tasks are associated with the parent root-level stage (UAT test).

```java
stage('UAT test') {
  parallel {
    stage('UAT test-1') {
      steps {
        snDevOpsStep()
        snDevOpsChange()
        // 'UAT test-1' tasks
      }
      post {
        success {
          // post success tasks. E.g.: junit '**/target/surefire-reports/*.xml'
        }
      }
    }
    stage('UAT static code test') {
      steps {
        snDevOpsStep()
        snDevOpsChange()
        // 'UAT static code test' tasks
      }
    }
  }
}
```

Associate pipeline steps and Jenkins freestyle jobs in DevOps

For manually created DevOps steps, associate each orchestration task in the Orchestration Tasks related list with a DevOps pipeline step to track the activity of each stage in your Jenkins freestyle pipeline.

Before you begin

Role required: sn_devops.admin
Procedure

1. Navigate to DevOps > Tools > Orchestration Tools and open the Jenkins tool record.

2. In the Orchestration Tasks related list, enter the corresponding DevOps pipeline step in the Step field of each orchestration task.

   Note: The Track field for the pipeline must be set to True in DevOps to receive job notifications from Jenkins.

Jenkins job run notifications are sent to the DevOps application. Each task execution notification corresponds to an orchestration task and, since orchestration tasks are mapped to a certain step in your DevOps pipeline, you can track the activity in each stage of your pipeline.

Example:

![DevOps associate steps](image)

Configure SonarQube scans on Jenkins pipelines

Configure SonarQube scans on Jenkins pipelines.

Before you begin

Ensure that you meet the following pre-requisites before configuring your Jenkins pipelines for Sonar scans

- SonarCloud or SonarQube version 8.6.1 community edition
- SonarQube Scanner for Jenkins 2.4 or higher is installed on your Jenkins instance/environment.
- ServiceNow DevOps plugin version 1.27 for Jenkins.
- Ensure that Sonar scans are configured and exist on your Jenkins pipelines using the SonarQube Scanner plugin for Jenkins. For more information, see SonarQube Scanner for Jenkins and SonarScanner for Jenkins.
- A Sonar (software quality tool) is created and is connected to the SonarQube server. For more information, see Create a SonarQube tool record in DevOps

Role required: sn_devops.admin
About this task
When you run a pipeline which has Sonar scan executions, the details are fetched into ServiceNow DevOps from the Jenkins pipeline. Using the Jenkins plugin, from the ServiceNow Store we check if the scan execution is configured in the Jenkins pipeline and check for Sonar scans on every orchestration stage, using the withSonarQubeEnv tag. If a Sonar analysis has happened on any stage of the pipeline’s execution, as part of our end notification we add a model with the scanID and url details for every scan that occurs in a particular stage. These scan analytics or details are correlated and displayed in the Software Quality Summary related list from Change requests and Task executions.

Procedure
1. In Jenkins, navigate to Manage Jenkins > Configure System, select the Enable injection of SonarQube server configuration as build environment variables check box in the SonarQube Servers section, and fill in the fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>A unique name for the configuration/project.</td>
</tr>
<tr>
<td>Server URL</td>
<td>URL or IP address of the SonarQube server.</td>
</tr>
<tr>
<td>Server authentication</td>
<td>Select the relevant authentication token type for the configuration.</td>
</tr>
</tbody>
</table>

Note: Use these same SonarQube server details or values while creating the Sonar tools in ServiceNow DevOps as well as in the Jenkins pipeline.

2. Click Test Connection.
   Verify that the connection successful message is shown.

3. Click Apply and Save.

4. Run the pipeline.

Results
Based on the scan results on various stages of pipeline’s execution, the results are shown in the corresponding step of the pipeline. Inbound events are created for notifications and subflows are triggered based on the notification type and capability.
What to do next

Note:
Navigate to the pipeline UI to view the scan details under the Quality > Software Quality Results.

- View scan details as part of Task Executions. View details of all the Sonar scans that are part of the task execution mapped to a build or release pipeline execution step.
  1. Navigate to DevOps > Orchestrate > Task Execution click a relevant Task Execution record.
  2. Click the Software Quality Summary related list.
  3. Click a relevant Scan ID record.
     The Software Quality Scan Summary and Scan Details display. For more information see, Software Quality Scan Summary

- View scan details as part of Change Request. View all the scans that were part of this build/release pipeline in the Software Quality Results > Software Quality Summary related list.
  1. Navigate to DevOps > Orchestrate > Pipeline Change Request
  2. Click the Software Quality Summary related list.
  3. Click a relevant Scan ID record.
     The Software Quality Scan Summary and Scan Details display. For more information see, Software Quality Scan Summary

Notify change request rejection or cancellation reason to Jenkins pipeline
Send change request rejection or cancellation reason to Jenkins pipeline logs.

Before you begin
- Ensure that you have upgraded to ServiceNow DevOps version 1.28.
- Have an active Jenkins integration.

Role required: sn_devops.admin

About this task
Starting from version 1.28, you can send change request rejection or cancellation reasons or comments to the Jenkins pipeline logs.
• Ensure that you enter appropriate reasons or comments when rejecting or canceling a Change Request manually.

• If you have loaded demo data during upgrade, and are using the DevOps Demo Change Automation flow or a custom flow based on it, a notification with default message values is sent to the Jenkins pipeline logs.

**Note:** Starting from version 1.30, the Approver name and time stamp of the cancellation/rejection is also automatically sent to the Jenkins pipeline logs.

**Procedure**

1. To manually reject or cancel change requests, follow these steps:
   a. Navigate to **DevOps > Orchestrate > Pipeline Change Requests > Change Request record**.
   b. Open the required Change Request record.
      • From the context menu, click **Cancel Change**. In the **Cancel Change Request > Reason** field, enter an appropriate reason for cancelling the change, and click **Save**.
      • In the Approvers related list, provide your inputs in the **Comment** field, right-click the record, and click **Reject**.
      The change request is canceled/rejected and the reason for canceling the change is added to the Comments field and sent to the Jenkins pipeline log.

2. To send custom messages (from auto-rejected change requests) to Jenkins, follow these steps:
   a. Navigate to **Flow Designer > DevOps Demo Change Automation flow > DevOps Demo Change Policy**.
   b. Navigate to the **DevOps Auto Reject decision >DevOps Apply Change Approval Definition subflow > DevOps Create Auto Approval Record action**.
   c. Modify the action’s input script for the approval.comments attribute value. By default, the auto-rejected change requests stores and sends the approval.comments = 'Auto ' + state + ' via Change Policy'; variables as messages to the Jenkins pipeline as notifications.

3. In Jenkins, navigate to the pipeline (that is corresponding to the rejected change request) > **Console Output**.
   The Change Request’s rejection or cancellation comments that are stored as part of the step execution reflect in the Jenkins Console Output.
Notify ServiceNow DevOps change request number to Jenkins pipelines

Send change request numbers to the Jenkins pipeline logs when a change request is created in ServiceNowDevOps.

Sending ServiceNowDevOps change request to Jenkins pipeline

Starting from version 1.31, change requests that are created as part of Jenkins pipeline executions for both scripted and free-style pipelines are notified to the Jenkins pipeline console.

When you trigger a scripted Jenkins pipeline and a change request is created as part of the pipeline execution stage in ServiceNowDevOps, the change request number is notified to Jenkins. The change request number displays in the Jenkins pipeline job logs for scripted pipelines.

Upon triggering a free-style Jenkins pipeline, the change request that is created as part of the ServiceNowDevOps pipeline execution stage waits for the approval action to be performed on the change. The change request number and details display against the relevant build in the Jenkins console's Build History section.
Note: After an approval or rejection action is performed on the change request in ServiceNow DevOps, the change request displays in the Jenkins pipeline build logs, similar to scripted pipelines.

**ServiceNow log levels and Log Recorders**

Set log levels for the ServiceNow DevOps Jenkins plugin based on the extent of log detail you need for debugging.

Starting from the ServiceNow DevOps Jenkins plugin 1.29, you can set the level of detail that you capture in your logs. In previous versions of the plugin, you could only enable or disable the **Debug** mode, without the ability to select the log level that you want to capture in your logs. Often, this leads to excessive logging that can rapidly flood your systems and bring your production servers down. For more information, see Configuring the Jenkins pipeline for DevOps.

For users installing DevOps for the first time starting from the DevOps version 1.29, by default the log level is turned off.

On successfully upgrading to the ServiceNowDevOps Jenkins plugin 1.29 from a previous version:
• If the **Debug** check box was previously selected, then the log level is enabled and set to **Info** by default.

• If the **Debug** check box was not selected previously, then by default, log levels are tuned off.

**Note:**
The Jenkins log does not record log messages that are lower than **Info**. The ServiceNowDevOps log recorders are used to record these messages.

You can select a log level from the ServiceNowDevOps Configuration form > Log level list. A base system ServiceNowDevOps log recorder is created in the Jenkins Log Recorders list. For more information, see Jenkins documentation on viewing logs.

**Note:** Do not modify the log recorder from the Jenkins > Log Recorders > Configure log recorders. Editing the log recorder from Jenkins creates a duplicate entry for any changes to the log levels that you make. Use the list of log levels in the ServiceNowDevOps Configuration form, to modify the log levels.

**Jira integration with DevOps**
Integrate Jira planning tool with DevOps by configuring a connection and credential alias, and creating a tool record in DevOps.

**Note:** You can create a new tool and simultaneously create a new connection alias. Since the child alias is created during tool creation, you do not need an admin role nor do you need to create a new connection alias as a prerequisite. Create the new connection alias as an sn_devops.admin user using Connections and Credentials template which are dynamically populated based on the tool you select. For more information, see Create new DevOps tool from the Tools list view.

**Configure connection and credential alias - Jira**
Before you set up a tool record in DevOps, your admin configures the DevOps CreateDevOpsTool connection and credential alias to allow access to the tools in your environment.

**Before you begin**
Starting with version 1.15, DevOps connection and credential aliases (DevOpsAlias1 through DevOpsAlias10) are used to connect automatically when you set up your DevOps planning, coding, and orchestration tools.

To connect to more than 10 tools, or if you receive an error saying all DevOpsAlias connection and credential aliases are being used, an admin can create additional aliases in the sn_devops application scope.

Note: If additional sys_alias records for DevOps are needed, they must be created by an admin in the sn_devops application scope.

You can reuse a DevOpsAlias alias no longer in use by inactivating the HTTP connection, if needed.

Role required: admin, sn_devops.admin

About this task
Your admin must configure an HTTP connection in the CreateDevOpsTool connection and credential alias provided, and add admin credentials to connect automatically when you set up your DevOps planning, coding, and orchestration tools:

- URL: https://<instance name>.service-now.com/
- Credentials: admin

Note: Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

You only need to configure the CreateDevOpsTool connection and credential alias once to set up all of your tools in DevOps.

The CreateDevOpsTool connection and credential alias is used with the DevOps Create a Tool subflow in Flow Designer. You can view executions in Flow Designer for information regarding flow results.

Procedure
1. Navigate to Connections & Credentials > Connection & Credential Aliases and open the CreateDevOpsTool record.

2. In the Connections related list, create a record and enter a Name for the connection.

3. On the Connection form, click the Credential field lookup list, and then click New to create an admin credential.
a. Click **Basic Auth Credentials** and enter a **Name**.

b. Enter admin username and password (required to access the tools in your DevOps environment).

**Note:** Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

4. On the Connection form, enter https://<instance name>.service-now.com/ for the **Connection URL**.

**Create a Jira tool record in DevOps**

Create a Jira tool record in DevOps to connect and discover application plans.

**Before you begin**

Role required: sn_devops.admin

**About this task**

Actions:
- **Connect** to Jira, get the webhook URL, and configure the webhook in Jira when you submit a DevOps tool record.
- **Discover** plans.
- **Import** work item, plan version, and feature records.

**Procedure**

1. Create a tool record in DevOps to automatically connect to Jira and get the webhook URL.

   a. Navigate to **DevOps > Tools > Create New** and create a record.

   b. Enter a **Tool Name** and fill in the tool details.

<table>
<thead>
<tr>
<th>Tool Integration</th>
<th>Jira</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool URL</td>
<td>Jira tool URL.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td><a href="https://jira.com">https://jira.com</a></td>
</tr>
<tr>
<td>Tool Username</td>
<td>Jira username</td>
</tr>
<tr>
<td>Tool Password / Access Token</td>
<td>Jira password or access token</td>
</tr>
</tbody>
</table>
**MID Server** is optional. Select MID Server for tools on premise attached to a MID Server. Application is automatically set to DevOps and capability is set to REST.

c. Click **Submit**.
The tool is automatically **Connected Successfully** using a connection alias, and HTTP tool connection (basic authentication credential).

⚠ *Note:* If you do not have global admin privileges for your tool (to allow automatic configuration of the webhook URL), you may need to have the tool admin user configure it for you (cut and paste the webhook URL into the tool configuration). Once the webhook is configured in the tool, **Enter Manual Configuration Mode** to connect to the tool manually, then exit.

2. Click **Discover** to discover existing application plans (projects associated with the tool). Plan records are added to the Plans related list.

3. Automatically configure the webhook URL in Jira.

   a. Click **Configure**.

   b. Enter your **Integration user** (DevOps Integration User) and **Password**. The DevOps Integration User is provided with the DevOps application and is used for inbound authentication from your tool to your ServiceNow instance.

      ⚠ *Note:* You must configure the password before the DevOps Integration User can configure a tool.

      Notifications are sent from the Jira tool to DevOps as the DevOps Integration User.

4. Open the discovered application plan from the Plans related list and click the **Import** related link to import historical data from the application. Imported work item records, plan version records, and feature records are added to the corresponding related lists.

   • Work items related list shows tasks, bugs, and stories.

   **Native State** and **Native Type** fields of the work item contain the original state and type values from the source tool.

   • Plan Versions related list shows releases.

   • Features related list shows epics and features.
5. In the Apps related list, click New to create an App record to associate with the plan (project).

**Note:** A plan must have an associated App record to complete the planning tool setup.

**Example:**

*Jira tool record in DevOps*

![Jira tool record in DevOps](image)

**ServiceNow Agile Development 2.0 integration with DevOps**

Integrate Agile Development 2.0 planning tool with DevOps by configuring a connection and credential alias, and creating a tool record in DevOps.

**Note:** You can create a new tool and simultaneously create a new connection alias. Since the child alias is created during tool creation, you do not need an admin role nor do you need to create a new connection alias as a prerequisite. Create the new connection alias as an sn_devops.admin user using Connections and Credentials template which are dynamically populated based on the tool you select. For more information, see Create new DevOps tool from the Tools list view.

**Configure connection and credential alias - Agile Development 2.0**

Before you set up a tool record in DevOps, your admin configures the DevOps CreateDevOpsTool connection and credential alias to allow access to the tools in your environment.
Before you begin

Starting with version 1.15, DevOps connection and credential aliases (DevOpsAlias1 through DevOpsAlias10) are used to connect automatically when you set up your DevOps planning, coding, and orchestration tools.


To connect to more than 10 tools, or if you receive an error saying all DevOpsAlias connection and credential aliases are being used, an admin can create additional aliases in the sn_devops application scope.

Note: If additional sys_alias records for DevOps are needed, they must be created by an admin in the sn_devops application scope.

You can reuse a DevOpsAlias alias no longer in use by inactivating the HTTP connection, if needed.

Role required: admin, sn_devops.admin

About this task

Your admin must configure an HTTP connection in the CreateDevOpsTool connection and credential alias provided, and add admin credentials to connect automatically when you set up your DevOps planning, coding, and orchestration tools:

- **URL**: https://<instance name>.service-now.com/
- **Credentials**: admin

Note: Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

You only need to configure the CreateDevOpsTool connection and credential alias once to set up all of your tools in DevOps.

The CreateDevOpsTool connection and credential alias is used with the DevOps Create a Tool subflow in Flow Designer. You can view executions in Flow Designer for information regarding flow results.

Procedure

1. Navigate to **Connections & Credentials > Connection & Credential Aliases** and open the **CreateDevOpsTool** record.

2. In the Connections related list, create a record and enter a **Name** for the connection.
3. On the Connection form, click the **Credential** field lookup list, and then click **New** to create an admin credential.

   a. Click **Basic Auth Credentials** and enter a **Name**.

   b. Enter admin username and password (required to access the tools in your DevOps environment).

   ✋ **Note:** Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

4. On the Connection form, enter https://<instance name>.service-now.com/ for the **Connection URL**.

**Create an Agile Development 2.0 tool record in DevOps**

Create an Agile Development 2.0 tool record in DevOps to connect and discover application plans.

**Before you begin**
Role required: sn_devops.admin

**About this task**

Actions:
- **Connect** to Agile Development 2.0, get the webhook URL, and configure the webhook in Agile Development 2.0 when you submit a DevOps tool record.
- **Discover** plans.
- **Import** work item, plan version, and feature records.

**Procedure**

1. Create a tool record in DevOps to automatically connect to Jira and get the webhook URL.

   a. Navigate to **DevOps > Tools > Create New** and create a record.

   b. Enter a **Tool Name** and fill in the tool details.

<table>
<thead>
<tr>
<th>Tool Integration</th>
<th>Agile Development 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool Username</td>
<td>Agile username</td>
</tr>
<tr>
<td>Tool Password / Access Token</td>
<td>Agile password or access token</td>
</tr>
</tbody>
</table>
**MID Server** is optional. Select MID Server for tools on premise attached to a MID Server. Application is automatically set to DevOps and capability is set to REST.

c. Click Submit.
   The tool is automatically **Connected Successfully** using a connection alias, and HTTP tool connection (basic authentication credential).

   🔄 **Note:** If you do not have global admin privileges for your tool (to allow automatic configuration of the webhook URL), you may need to have the tool admin user configure it for you (cut and paste the webhook URL into the tool configuration). Once the webhook is configured in the tool, **Enter Manual Configuration Mode** to connect to the tool manually, then exit.

2. Click **Discover** to discover existing application plans (projects associated with the tool).
   Plan records are added to the Plans related list.

3. Automatically configure the webhook URL in Agile Development 2.0.

   **a. Click Configure.**

   **b. Enter your Integration user** (DevOps Integration User) and **Password**.
   The DevOps Integration User is provided with the DevOps application and is used for inbound authentication from your tool to your ServiceNow instance.

   🔄 **Note:** You must configure the password before the DevOps Integration User can configure a tool.
   Notifications are sent from the Agile Development 2.0 tool to DevOps as the DevOps Integration User.

4. Open the discovered application plan from the Plans related list and click the **Import** related link to import historical data from the application.
   Imported work item records, plan version records, and feature records are added to the corresponding related lists.

   • Work items related list shows tasks, bugs, and stories.

   **Native State** and **Native Type** fields of the work item contain the original state and type values from the source tool.

   • Plan Versions related list shows releases.

   • Features related list shows epics and features.
5. On the Apps related list, click New to create an App record to associate with the plan (project).

**Note:** A plan must have an associated App record to complete the planning tool setup.

**Example:**

**Agile tool record in DevOps**

![Image of an Agile tool record](https://example-url.com/)

<table>
<thead>
<tr>
<th>Name</th>
<th>ITBM - Agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool</td>
<td>Agile Development 2.0</td>
</tr>
<tr>
<td>Connection alias</td>
<td>Agile</td>
</tr>
<tr>
<td>Connection state</td>
<td>Connected</td>
</tr>
<tr>
<td>Last discovery</td>
<td>2020-03-05 17:37:20</td>
</tr>
</tbody>
</table>

**SonarQube integration with DevOps**

Integrate SonarQube or SonarCloud DevOps tools with ServiceNowDevOps using the ServiceNow DevOps Integrations plugin. This integration introduces code quality checks on your DevOps pipelines.

**Important:** You must install the DevOps Integrations application to integrate SonarQube tools with DevOps.

**Overview**

- Configure SonarQube scans on Jenkins and Azure DevOps pipelines. Scan results are fetched from the pipeline in ServiceNow DevOps.

- View summary results for code quality tools that can be viewed in a related list of a Change Request or a Task Execution flow. Use code quality results for change automation, this includes a way to make an automation policy against the results.
Important: You can set up branch analysis to enable SonarCloud to analyze branches in your projects apart from the main branch. You cannot set up or perform branch analysis on SonarQube community edition licenses. Upgrade to SonarQube Developer or Enterprise editions to set up branch analysis on SonarQube on-premise implementations.

Note: You can create a new tool and simultaneously create a new connection alias. Since the child alias is created during tool creation, you do not need an admin role nor do you need to create a new connection alias as a prerequisite. Create the new connection alias as an sn_devops.admin user using Connections and Credentials template which are dynamically populated based on the tool you select. For more information, see Create new DevOps tool from the Tools list view.

Verify base system data availability

Before you set up a tool record in ServiceNow® DevOps, verify that the seeded data is available to configure the tool in your instance.

Before you begin

Upgrade to ServiceNow DevOps version 1.27.

Role required: sn_devops.admin

About this task

Verify that the following menu items and user interfaces display after you upgrade to Devops version 1.27.

Procedure

1. Navigate to Connections and & Credentials > Connection & Credential Aliases > CreateDevOpsTool record.

The CreateDevOpsTool connection alias record exists.

2. Navigate to DevOps > Integrations > Tool Integrations > Sonar.
3. Navigate to DevOps > Integrations > Tool Integration Capabilities and check for softwarequality record.
4. Navigate to DevOps > Tool Capability Mappings and click the Sonar record.

Configure connection and credential alias - SonarQube

Before you set up a tool record in DevOps, your admin configures the DevOps CreateDevOpsTool connection and credential alias to allow access to the tools in your environment.

Before you begin
Starting with version 1.15, DevOps connection and credential aliases (DevOpsAlias1 through DevOpsAlias10) are used to connect automatically when you set up your DevOps planning, coding, and orchestration tools.


To connect to more than 10 tools, or if you receive an error saying all DevOpsAlias connection and credential aliases are being used, an admin can create additional aliases in the sn_devops application scope.

Note: If additional sys_alias records for DevOps are needed, they must be created by an admin in the sn_devops application scope.

You can reuse a DevOpsAlias alias no longer in use by inactivating the HTTP connection, if needed.

Role required: admin, sn_devops.admin
About this task
Your admin must configure an HTTP connection in the CreateDevOpsTool connection and credential alias provided, and add admin credentials to connect automatically when you set up your DevOps planning, coding, and orchestration tools:

- **URL**: `https://<instance name>.service-now.com/`
- **Credentials**: admin

ℹ️ **Note**: Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

You only need to configure the CreateDevOpsTool connection and credential alias once to set up all of your tools in DevOps.

The CreateDevOpsTool connection and credential alias is used with the DevOps Create a Tool subflow in Flow Designer. You can view executions in Flow Designer for information regarding flow results.

Procedure
1. Navigate to **Connections & Credentials > Connection & Credential Aliases** and open the **CreateDevOpsTool** record.
2. In the Connections related list, create a record and enter a **Name** for the connection.
3. On the Connection form, click the **Credential** field lookup list, and then click **New** to create an admin credential.
   a. Click **Basic Auth Credentials** and enter a **Name**.
   b. Enter admin username and password (required to access the tools in your DevOps environment).

   ℹ️ **Note**: Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

4. On the Connection form, enter `https://<instance name>.service-now.com/` for the **Connection URL**.

Create a SonarQube tool record in DevOps
Create an Sonar DevOps tool record in DevOps to connect, discover, and import SonarQube DevOps tool data.
Before you begin
Ensure that you have

- Project-level access to your SonarQube instance, in order to configure scans for all projects on your instance.

Role required: sn_devops.admin, Administrator role in SonarQube or SonarCloud

About this task
You create a SonarQube tool and a connection alias only to connect and fetch details through REST calls. Unlike other DevOps tools you cannot configure or discover using the tool and the connection.

Actions:
- **Connect** to SonarQube and get the webhook URL when you submit a DevOps tool record.

Procedure
Create a tool record in DevOps to automatically connect to GitLab and get the webhook URL.

a. Navigate to DevOps > Tools > Create New and create a record.

b. Enter a **Tool Name** and fill in the tool details.

<table>
<thead>
<tr>
<th>Tool Integration</th>
<th>Sonar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool URL</td>
<td>SonarQube/SonarCloud instance URL.</td>
</tr>
<tr>
<td></td>
<td>For example, <a href="https://sonarcloud.io">https://sonarcloud.io</a></td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Do not enter slash (/) character at the end of your Sonar tool URL.</td>
</tr>
<tr>
<td>Tool Password / Access Token</td>
<td>Sonar user token</td>
</tr>
</tbody>
</table>
Note: Only user tokens generated at User > My Account > Security from your Sonar instance is supported.

Before generating the token for a Sonar user, ensure that you grant Browse permission for all your Sonar projects.

api: Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.

Note: MID Server is optional. Select MID Server for tools on premise attached to a MID Server. Application is automatically set to DevOps and capability is set to REST. For more information, see Setting up the MID server.

c. Click Submit.
   The tool is automatically Connected Successfully using a connection alias and HTTP tool connection (basic authentication credential), and an API key credential is created.

You have successfully configured the SonarQube software quality DevOps tool and a connection alias. Use this connection alias to make the REST API calls to fetch details from the Sonar tool.

Example:
What to do next

- Configure SonarQube scans on Azure DevOps pipelines
- Configure SonarQube scans on Jenkins pipelines

Accelerating DevOps change

Enable the Change Acceleration feature of DevOps for automatic change request creation in your pipeline, and use change approval policies to automate approval under certain conditions.

Note: ServiceNow Change Management must be installed for change acceleration.

Enable and set up change control when you model your pipeline in DevOps:

- Model an Azure pipeline in DevOps
- Model a GitLab basic CI pipeline in DevOps
- Model a Jenkins pipeline in DevOps

You can view details for active change requests by navigating to DevOps > Orchestrate > Pipeline Change Requests.

Change control process

When change control is enabled for a job in your DevOps development pipeline, a change request is automatically created and set to Assess state to request approval for the execution of the current stage or job.

Starting with version 1.20, pipeline steps can be configured to create change receipts, which do not pause the pipeline. Change receipts include all pipeline data, but do not require approval to proceed in the pipeline.

Change requests can be approved automatically by configuring conditions in a change approval policy. The DevOps change approval policy (DevOps Change Policy) and workflow (Change Request - DevOps) are provided with the DevOps application.

The difference in the DevOps workflow (Change Request - DevOps) from the Normal workflow (Change Request - Normal) is that the change approval policy in the DevOps workflow is the DevOps Change Policy instead of the Normal Change Policy and, therefore, does not use Technical approvals nor Risk approvals activities that are part of the Normal workflow.

Once approved, either automatically or manually, change requests move to Implement state and the job is run. Once the job is run, the change request
is moved to Closed with Close code as Successful on a successful job run, or Unsuccessful on error in the job run.

If a change request is not approved and moved into canceled state or closed state, the associated Jenkins job is marked as failed and a console message is shown:

[ServiceNow DevOps] Job was not approved for execution

**Automatic approval of change requests using Change Policy workflows**

You can enable automatic approval of DevOps changes using the Change Approval Policy workflow and the DevOps Change Policy workflow.

**Change Approval Policy workflow**

A change request is automatically approved for low risk changes, when calculated risk and impact are below threshold values (set on the Step form for the pipeline). The state is moved to Implement.

When calculated risk and impact values are at or above threshold values, the normal change stays in the Assess state until approved.

**Note:** If the Standard change template field on the Step form is set, and the calculated risk and impact are below threshold values, a standard change request is created. Otherwise the change is always normal.

**DevOps Change Policy workflow**

A change request is automatically approved when certain conditions in the DevOps auto approval cycle are met. The state is moved to Implement. DevOps Change Policy approval is useful for less critical changes that do not require manual approval.

Change Approval Policy is mapped to DevOps Change Policy (instead of Normal Workflow Policy), and Risk Approvals is mapped to DevOps Change Policy.

This policy is inactive by default.

**DevOps Demo Change Policy workflow**

Starting with version 1.14, when demo data is installed, Flow Designer #DevOps# Demo# Change Automation flow# is available where# normal type change requests can be auto-approved based on the decision policies.

As a part of demo data, the decision policies available are:
• Low risk auto approval policy, where the failed test value is zero.
• High risk manual approval policy, where the failed test value is greater than zero.

See DevOps Demo Change Automation flow for more details.

**Custom change request templates**
Starting with version 1.14, when you enable change control in the ServiceNow DevOps step, you can select a custom template to populate fields automatically while creating the change request. The change request Category field is automatically set to DevOps.

ℹ️ **Note:** Do not configure the Category and changeType fields from the custom template.

The type of change request corresponds to the change request table in global scope.

**DevOps Model Change Request flow**
Starting with version 1.14, customize the DevOps Model Change Request flow using a flow or a script.

Once the change request is approved, rejected, or canceled, the state of the step execution can be updated by calling Flow Designer action **Update the state of step execution table** even before implementing the change request. If a change request is not approved, and moved into canceled or rejected state, the associated job is marked as canceled or failed and a console message is shown.

Flow Designer action **Update the state of step execution table** serves as a trigger for Change Control Callback flow, which is used to notify the change decision to the orchestration tool. Hence, calling the Flow Designer action **Update the state of step execution table** is mandatory.

**Automatic change request related lists**
For a change request automatically created by DevOps, the Category field is automatically set to DevOps, and these related lists are added:

- **Commits**
  Commits associated with the change request.

- **Work Items**
  Work items associated with the change request.

- **Test Results (build test summaries)**, starting with version 1.13
List of build test summaries for the tests run after the associated package was created and linked.

If there is no package, or the change request is created before package creation, then the list of build test summaries includes tests that were run after any associated artifacts were created and linked.

If no package or artifact versions are linked to the pipeline execution, then the list is empty.

**Artifact Versions, starting with version 1.13**

List of artifact versions associated to the package linked to pipeline execution for packages created before the change request is approved.

If no package is linked to the pipeline execution, then the list is empty.

**Test Summaries (replaces Test Results related list), starting with version 1.15**

List of test summaries for a pipeline execution associated with an artifact, package, or task execution before the change request.

See [Test Results](#) for more details.

---

**Note:** Implementation details from the orchestration tool are automatically added to the **Work notes** field on the change request form. Detail added to the work notes is limited to 5 KB of the task execution log for the step.

---

**Custom change request process**

These **DevOps change properties** are available to customize your change request flow.

- DevOps change request implement state
- DevOps change request post implement state
• DevOps change request cancel state
• DevOps change request approval text

To customize your change request flow, you must first create a System Definition > Choice List. For example, DevOps_Implement (value = 10).

Then, add the choice list to System Definition > Script Include > ChangeRequestStateHandlerSNC.

Once you have created the choice list and added it to the script include, you can update DevOps change properties with the new choice list values. For example, DevOps change request implement state -10.

**DevOps Risk Condition**

You can use the DevOps risk and impact calculation based on committer risk score.

This condition is inactive by default.

**Test Results related list**

Lists the tests that were run in a pipeline after a package was created. If no package was created, then the list includes the tests that were run after an artifact version was created.

Scenarios:

**A package is created in the pipeline, but no artifact versions are registered.**

- If the change request is created in the package creation stage:
  
  No test results are displayed because a package is not yet linked to the pipeline execution.

- If the change request is created in a stage after the package creation stage:

  Build test summaries include those associated with stages after the package creation stage, up to the change-controlled stage.

**Artifact versions are registered, but no package is created.**

- If the change request is created in the artifact version stage:

  No test results will be displayed, because no tests are associated until the task execution is completed.
• If the change request is created in a stage after the artifact version stage:
  Build test summaries include those in the artifact version stage, as well as the stages afterward, up to the change-controlled stage.

Both artifact versions and package are created in the pipeline.

• If the change request is part of the stage after artifact version and package creation stages:
  Build test summaries include those associated with the package creation stage, as well as the stages afterward, up to the change-controlled stage.

• If the change request is part of the package creation stage, and artifact versions are created as part of an earlier stage; or
  the change request is created in a stage (not package creation) after the artifact version stage, but before package creation stage; or
  the change request is part of the package creation stage and artifact versions are created as part of an earlier stage:
  Build test summaries include those associated with the artifact version stage, as well as stages afterward, up to the change-controlled stage.

Pipeline executions view
You can view pipeline activity by navigating to DevOps > Orchestrate > Pipeline Executions.
DevOps change acceleration for releases

Starting with version 1.11, DevOps change acceleration for releases lets you view all commits and work items in the change request for the DevOps release when approving, rather than having commits spread across multiple task executions.

When you associate commits with an artifact version (CI pipeline), and define an artifact package (CD pipeline), all artifact versions generated since the last time the app was deployed to production are included in the list of commits for the change. Consolidation of these items is helpful, especially when there are multiple CI builds before the deployment.

DevOps change request attributes:

- **Category** DevOps or the **Categorize DevOps change requests on "DevOps Change" field** check box is selected in DevOps properties. For more information, see DevOps change request without mandating category as DevOps

- **Commits and Work Items related lists**
Artifact setup
1. (Optional) Create an artifact tool record in DevOps.

   Note: An artifact tool is not necessary unless a webhook or user-created integration subflow configuration is required to look up artifact versions.

2. Register artifacts in the CI pipeline.
3. Create a package in the CD pipeline.

   Note: The package creation step must be before the Prod Deploy step.

Artifact registration
Configure artifact registration in a scripted pipeline or freestyle job using the DevOps API /artifact/registration endpoint. Multiple artifact versions are supported.

For Jenkins pipeline:

• Scripted and declarative pipeline (snDevOpsArtifact Jenkinsfile command)

   For example:

   ```java
   snDevOpsArtifact(artifactsPayload: ""{"artifacts": [{"name": "sa-web.jar", "version": "1.9", "semanticVersion": "1.9.0", "repositoryName": "services-1031"}], "branchName": "master"}]"")
   ```

• Freestyle job (Register Artifact build step)

   For example:

   ```json
   {"artifacts": [{"name": "sentiment-analysis-web2", "version": "1.9", "semanticVersion": "1.9.0", "repositoryName": "maven-releases"}]}
   ```

Artifact package creation
Configure artifact package creation in a scripted pipeline or freestyle job using the DevOps API /package/registration endpoint.

   Note: Package name must be specified.

For Jenkins pipeline:

• Freestyle job (Create Package build step)

   For example:

   ```json
   {"artifacts": [{"name": "sentiment-analysis-web2", "version": "1.9", "repositoryName": "maven-releases"}]}
   ```
• Declarative and scripted pipeline (\texttt{snDevOpsPackage} Jenkinsfile command)

Package with more than one artifact (from different repos) in the payload, for example:

\begin{verbatim}
snDevOpsPackage(name: "sentimentpackage", artifactsPayload: "{"artifacts":[{"name": "sa-web.jar", "version": "1.9", "repositoryName": "services-1031"},{"name": "sa-db.jar", "version": "1.3.2", "repositoryName": "services-1032"}],"branchName": "master"}"
\end{verbatim}

\textbf{Note:} When an artifact version is not available during the build, build details (pipelineName or projectName, taskExecutionNumber, stageName, branchName) are used to look up the artifact version in the task execution.

Jenkins plugin step \texttt{includeBuildInfo} can be used to include build details in the API call.

\textbf{Artifact workflow and objects}

The orchestration tool job publishes the new artifact (consisting of versions) to the artifact repository. Each artifact version is associated with a task execution
(consisting of the related code commits). A package is created for the release (consisting of specific artifact versions flagged for deployment) and, once the deployment stage completes, the package is marked as deployed to production.

These objects are part of the artifact structure.

**Artifact tool**

Used to support artifact repository managers such as JFrog Artifactory and Nexus, for example.

ℹ️ **Note:** An artifact tool is not necessary unless a webhook or user-created integration subflow configuration is required to look up artifact versions.

**Artifact repository**

Target for artifacts generated in a build, and also a source of artifacts required by a build.

Can be created manually, or through the RegisterArtifact API as new artifacts are published under new repositories in a tool.

**Artifact**

Artifact name for which different builds (artifact versions) are generated.

Can be created manually, or through the RegisterArtifact API. Artifacts (versions) are associated with a task execution and published to an artifact repository.

**Artifact version**

Specific version of the artifact. Deployable component of an application generated by a CI build. When provided, semantic version is used.

Can be created through discovery, or through the RegisterArtifact API. Artifacts (versions) are associated with a task execution and published to a tracked artifact repository.

**Semantic version**

Optional attribute of an artifact version that, when provided, is used to determine commits for a change.

Semantic version format is (MAJOR.MINOR.PATCH).

**Package**

Collection of artifact versions used as input to a CD pipeline, or for associating test results.
Package creation is triggered by the CreatePackage API call from the orchestration tool and contains the name, version, and repository name of all the artifact versions included in the package. A check box indicates whether the package has been deployed to production.

DevOps change acceleration for releases

DevOps change request – category DevOps

DevOps change request – commits and work items
DevOps artifact version list

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Semantic version</th>
<th>Artifact</th>
<th>Built by</th>
<th>Built on</th>
</tr>
</thead>
<tbody>
<tr>
<td>globe-web-war</td>
<td>2.30.0</td>
<td>2.30.0</td>
<td>globe-web-war</td>
<td>TE0001128</td>
<td>2020-02-20 13:32:51</td>
</tr>
<tr>
<td>globe-web-war</td>
<td>2.29.0</td>
<td>2.29.0</td>
<td>globe-web-war</td>
<td>TE0001114</td>
<td>2020-02-20 13:29:50</td>
</tr>
<tr>
<td>globe-web-war</td>
<td>2.28.0</td>
<td>2.28.0</td>
<td>globe-web-war</td>
<td>TE0001112</td>
<td>2020-02-20 13:28:20</td>
</tr>
<tr>
<td>globe-web-war</td>
<td>2.27.0</td>
<td>2.27.0</td>
<td>globe-web-war</td>
<td>TE0001108</td>
<td>2020-02-20 13:17:50</td>
</tr>
<tr>
<td>globe-web-war</td>
<td>2.26.0</td>
<td>2.26.0</td>
<td>globe-web-war</td>
<td>TE0001106</td>
<td>2020-02-20 13:14:51</td>
</tr>
<tr>
<td>globe-web-war</td>
<td>2.25.0</td>
<td>2.25.0</td>
<td>globe-web-war</td>
<td>TE0001104</td>
<td>2020-02-20 13:04:20</td>
</tr>
</tbody>
</table>

DevOps artifact version – commits

<table>
<thead>
<tr>
<th>Artifact Version</th>
<th>URL</th>
<th>Artifact</th>
<th>Semantic version</th>
<th>Version</th>
<th>Built on</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.16.0</td>
<td>com.globus.web/globe-web/2.16.0</td>
<td>globe-web</td>
<td>0.16.0</td>
<td>0.16.0</td>
<td>2020-01-14 09:43:11</td>
</tr>
</tbody>
</table>

DevOps artifact version – packages

<table>
<thead>
<tr>
<th>Artifact Version</th>
<th>URL</th>
<th>Artifact</th>
<th>Semantic version</th>
<th>Version</th>
<th>Built on</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.16.0</td>
<td>com.globus.web/globe-web/2.16.0</td>
<td>globe-web</td>
<td>0.16.0</td>
<td>0.16.0</td>
<td>2020-01-14 09:43:11</td>
</tr>
</tbody>
</table>
DevOps change request without mandating category as DevOps

Enable categorizing DevOps change request from DevOps properties. Change requests created as part of pipeline execution steps can now be created with a category other than DevOps.

Before you begin
Role required: sn_devops.admin

About this task
In previous versions, a change request created from the DevOps application automatically had the Category field set to DevOps. Starting from version 1.29, change requests created in the DevOps application are no longer mandatorily created with Category as DevOps.

Use related templates for specific categories such as Hardware, Software, Database and so on, to create DevOps change requests. You can now categorize DevOps change requests from the DevOps properties.

New change requests that are created in the DevOps application after you enable the property can continue to use custom change policies and approval workflows, in addition to the base system's DevOps Demo Change Automation flow, and the DevOps Model Change Request flow. If you do not enable this property, change requests continue to be created and processed with the category attribute set to DevOps.

Procedure
1. Navigate to DevOps > Administration > Properties > Categorize DevOps changes requests on "DevOps change" field.
   By default, the property check box is cleared.
2. Select the check box to enable categorizing any change request created from the DevOps application, as a DevOps change request.

⚠️ Note: DevOps Insights capabilities are currently disabled when this property is selected.

Results
Any new change request that is created from the DevOps application is considered as a DevOps change irrespective of the category attribute and processed accordingly. DevOps change request creation and processing no longer depend only on the category attribute being set to DevOps.

How commits are determined for a release in DevOps
Starting with version 1.11, the DevOps artifact package and its associated artifact versions are used to determine which commits are included in a DevOps change.

All commits for artifact versions in the package that were generated after the last deployment date (up to the version currently being deployed) are included in the change request. Previous major versions are not included.

⚠️ Note: Patch and hotfix versions are excluded.

When provided, semantic version is used to determine commits for a change. Format is (MAJOR.MINOR.PATCH). For example, semantic version 2.0.1 is read:

- Major version 2
- Minor version 0
- Patch/hotfix version 1

Failed task executions between the previous and current artifact versions that did not build an artifact but have associated commits are also associated to the created artifact version.

Example: Commits and packages
This example consists of:

- Three build pipelines (A, B, and C)
- A release pipeline (ABC) under change control, with four packages:
  1. Build pipeline A (major version 1)
  2. Build pipelines A and B (major version 1)
  3. Build pipelines B and C (major version 1)
  4. Build pipelines A, B, and C (major version 1)
Package 1 (A 1.1.0)

<table>
<thead>
<tr>
<th>Commit</th>
<th>Build pipeline</th>
<th>Semantic version</th>
<th>Included in package</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>1.0.0</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>1.0.1</td>
<td>--</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>1.1.0</td>
<td>X</td>
</tr>
</tbody>
</table>

**Note:** Commit 2 (build A, semantic version 1.0.1) is not included in the package because the semantic version syntax indicates a patch or hotfix.

Package 2 (A 1.2.0, B 1.1.0)

<table>
<thead>
<tr>
<th>Commit</th>
<th>Build pipeline</th>
<th>Semantic version</th>
<th>Included in package</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>A</td>
<td>1.1.1</td>
<td>--</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>1.2.0</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>1.2.0</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>1.0.0</td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td>B</td>
<td>1.0.0</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>B</td>
<td>1.1.0</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>B</td>
<td>1.1.0</td>
<td>X</td>
</tr>
</tbody>
</table>

**Note:** Commit 4 (build A, semantic version 1.1.1) is not included because the semantic version syntax indicates a patch or hotfix.

Package 3 (B 1.2.0, C 1.0.0)

<table>
<thead>
<tr>
<th>Commit</th>
<th>Build pipeline</th>
<th>Semantic version</th>
<th>Included in package</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>A</td>
<td>1.3.0</td>
<td>--</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
<td>1.2.0</td>
<td>X</td>
</tr>
<tr>
<td>13</td>
<td>B</td>
<td>1.2.0</td>
<td>X</td>
</tr>
<tr>
<td>14</td>
<td>C</td>
<td>1.0.0</td>
<td>X</td>
</tr>
<tr>
<td>15</td>
<td>C</td>
<td>1.0.0</td>
<td>X</td>
</tr>
<tr>
<td>16</td>
<td>C</td>
<td>1.0.0</td>
<td>X</td>
</tr>
</tbody>
</table>
### Package 3 (B 1.2.0, C 1.0.0) (continued)

<table>
<thead>
<tr>
<th>Commit</th>
<th>Build pipeline</th>
<th>Semantic version</th>
<th>Included in package</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Commit 11 (build A, semantic version 1.3.0) is not included because the package does not specify build A.

### Package 4 (A 1.4.0, B 1.3.0, C 1.2.0)

<table>
<thead>
<tr>
<th>Commit</th>
<th>Build pipeline</th>
<th>Semantic version</th>
<th>Included in package</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>A</td>
<td>1.4.0</td>
<td>X</td>
</tr>
<tr>
<td>18</td>
<td>A</td>
<td>1.4.0</td>
<td>X</td>
</tr>
<tr>
<td>19</td>
<td>B</td>
<td>1.3.0</td>
<td>X</td>
</tr>
<tr>
<td>20</td>
<td>B</td>
<td>1.3.0</td>
<td>X</td>
</tr>
<tr>
<td>21</td>
<td>C</td>
<td>1.1.0</td>
<td>X</td>
</tr>
<tr>
<td>22</td>
<td>C</td>
<td>1.1.0</td>
<td>X</td>
</tr>
<tr>
<td>23</td>
<td>C</td>
<td>1.2.0</td>
<td>X</td>
</tr>
<tr>
<td>24</td>
<td>C</td>
<td>1.2.0</td>
<td>X</td>
</tr>
</tbody>
</table>

**Note:** Commit 11 is also included in this package because it is part of the changes in major version 1 of build A since the last package (including major version 1 of build A), package #2, was deployed to production.

### Setting up a DevOps artifact tool record

Starting with version 1.11, set up a DevOps artifact tool connection to discover artifact repositories when webhook or user-created integration subflow configuration is required to look up artifact versions.

### Configure connection and credential alias - artifact tool

Before you set up your tool records in DevOps, your admin configures the DevOps CreateDevOpsTool connection and credential alias to allow access to the tools in your environment.

### Before you begin

Starting with version 1.15, DevOps connection and credential aliases (DevOpsAlias1 through DevOpsAlias10) are used to connect automatically when you set up your DevOps planning, coding, and orchestration tools.

To connect to more than 10 tools, or if you receive an error saying all DevOpsAlias connection and credential aliases are being used, an admin can create additional aliases in the sn_devops application scope.

Note: If additional sys_alias records for DevOps are needed, they must be created by an admin in the sn_devops application scope.

You can reuse a DevOpsAlias alias no longer in use by inactivating the HTTP connection, if needed.

Role required: sn_devops.admin

About this task
Your admin must configure an HTTP connection in the CreateDevOpsTool connection and credential alias provided, and add admin credentials to connect automatically when you set up your DevOps planning, coding, and orchestration tools:

• URL: https://<instance name>.service-now.com/api/now/table
• Credentials: admin

Note: Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

You only need to configure the CreateDevOpsTool connection and credential alias once to set up all of your tools in DevOps.

The CreateDevOpsTool connection and credential alias is used with the DevOps Create a Tool subflow in Flow Designer. You can view executions in Flow Designer for information regarding flow results.

Procedure
1. Navigate to Connections & Credentials > Connection & Credential Aliases and open the CreateDevOpsTool record.
2. In the Connections related list, create a record and enter a Name for the connection.
3. On the Connection form, click the Credential field lookup list, and then click New to create an admin credential.
a. Click **Basic Auth Credentials** and enter a **Name**.

b. Enter admin username and password (required to access the tools in your DevOps environment).

   **Note:** Starting with version 1.15, admin role is not necessary. A user with connection_admin role can configure an HTTP connection.

4. On the Connection form, enter `https://<instance name>.service-now.com/api/now/table` for the **Connection URL**.

**Create an artifact tool record in DevOps**

Create an artifact tool record in DevOps to discover artifact repositories when webhook or user-created integration subflow configuration is required to look up artifact versions.

**Before you begin**

Role required: sn_devops.admin

**About this task**

   **Note:** An artifact tool is not necessary unless a webhook or user-created integration subflow configuration is required to look up artifact versions.

**Procedure**

1. Navigate to **DevOps > Tools > Create New** and create a record.
2. Enter a **Name**, **Tool Integration**, and **Tool URL**.

   **Note:** If you do not have global admin privileges for your tool (to allow automatic configuration of the webhook URL), you may need to have the tool admin user configure it for you (cut and paste the webhook URL into the tool configuration). Once the webhook is configured in the tool, **Enter Manual Configuration Mode** to connect to the tool manually, then exit.

After submitting, the artifact tool is automatically **Connected Successfully** using a connection alias, and HTTP tool connection (basic authentication credential).
Example:

**DevOps artifact tool setup**

![DevOps artifact tool setup](image)

### Configuring DevOps change request details within the pipeline

Starting with version 1.24, set closure code and change request fields from within Azure and Jenkins pipelines.

**Note:** Configuring change request details from within a GitLab pipeline is not supported.

Closure information and change request attributes are contained with the `changeRequestDetails` object.

### Closure code

Set the `setCloseCode` parameter to `true/false` based on the desired behavior.

If set to true, standard change request behavior occurs once the job or pipeline has completed.

If set to false, once the job or pipeline has completed the change request is not closed:

- Closure Information in the change request is not set (`Close code` and `Close notes` fields are left empty).
- A link to the step execution is added to the `Work notes`.

### Change request fields

Set change request field values within the pipeline for the change request template specified.

- Use the `attributes` parameter to set field values.
- Use the DevOps - POST `/devops/orchestration/changeControl` endpoint of the DevOps API.
Note:

- If a specified field has a dependent field that is mandatory, you must set that attribute as well.

- If the attribute for the dependent mandatory field is not set, change request and related step execution are canceled, and work notes are updated.

Field values within the attributes: parameter are key-value pairs. Meaning, the key is the field name within the template and the value is the information to populate in the field.

Starting with version 1.27, you can use the changeControl API to specify fields such as type, cmdb_ci, template, assignment_group business_service, standard_change_template and create a change request.

Note: Please ensure that you keep in mind the following points while configuring change requests from within the pipeline:

- Any attributes that you set in the changeControl API overrides any values set on the step.

- Change type and template fields are always taken from one source, that is you cannot use a combination attributes from the API request and the change request form.

All fields in the Change Request [change_request] table are supported except where specified.

<table>
<thead>
<tr>
<th>Change request fields supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsupported fields</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Supported fields</td>
</tr>
</tbody>
</table>

Note: The attribute name must match the change request field name, and the value specified must be valid.
Sample JSON payload

```json
{
  "callbackURL":"http://192.168.0.4:3000/jenkins/sn-devops/pipeline_839b7605-b98d-4831-bc87-96829de7da37",
  "orchestrationTaskURL":"http://192.168.0.4:3000/jenkins/job/java_sample_tests#deploy",
  "isMultiBranch":"false",
  "orchestrationTaskName":"java_sample_tests#deploy",
  "orchestrationTaskDetails":{
    "triggerType":"upstream",
    "upstreamTaskExecutionURL":"http://192.168.0.4:3000/jenkins/job/java_sample_tests/129/execution/node/35/wfapi/describe",
    "taskExecutionURL":"http://192.168.0.4:3000/jenkins/job/java_sample_tests/129/execution/node/50/wfapi/describe"
  },
  "changeRequestDetails":{
    "setCloseCode":false,
    "attributes":{
      "sys_created_by":"1832fbe1d701120035ae23c7ce610369",
      "sys_updated_by":"56826bf03710200044e0bfc8bcbe5dca",
      "requested_by":{
        "name":"Abel Tuter"
      },
      "watch_list":[
        {
          "name":"Abel Tuter"
        },
        {
          "name":"Aileen Mottern"
        },
        {
          "name":"Alejandra Prenatt"
        },
        "56826bf03710200044e0bfc8bcbe5dca"
      ],
      "work_notes_list":[
        "56826bf03710200044e0bfc8bcbe5dca",
        "465f9e9f9e198101d5e0ed9ad6f6e7",
        "d8f57f140b20220050192f15d6673a98"
      ],
      "assigned_to":"1832fbe1d701120035ae23c7ce610369"
    }
  }
}
```
Pipeline examples

Change request details - Azure pipeline

Job-level settings - Jenkins
Using DevOps Model Change Request flow

Starting with version 1.14, customize or recreate the DevOps Model Change Request flow based on your requirements using a flow or a script.

In the DevOps Model Change Request flow, the state of step execution is changed based on the change approval. However, you can customize or recreate this flow based on your requirements.

After the change request state is moved to approve, canceled, or rejected (either manually or by using a change policy), call the **Update state of step execution based on change approval** Flow Designer action to update the State field of step execution record.

You can use either a flow or a script to call the action.

**Calling the Flow Designer action using a flow**

Calling the **Update state of step execution based on change approval** Flow Designer action is required to update the state of the step execution record according to the approval field in the change request record.

This action serves as a trigger for the **Change Control Callback** flow, which is used to notify the change decision to the orchestration tool.
Calling the Flow Designer action using a script

Method to call the Flow Designer action from a script:

```javascript
sn_fd.FlowAPI.executeAction('sn_devops.name_of_FD_action', inputs);
```
Default Change Handler subflow
Use the Default Change Handler subflow to populate these change request fields with default values.

- Requested by
- Justification
- Implementation Plan
- Backout plan
- Test Plan
- Short Description
- Description
- Start Date
- End Date
- Risk Impact Analysis

The Default Change Handler subflow overrides the field values that were populated using a template while creating the change request record.

If desired, you can write a custom subflow in place of this flow by modifying the [sn_devops.change_request_handler_subflow] DevOps property.

DevOps Demo Change Automation flow
When demo data is installed, Flow Designer DevOps Demo Change Automation flow is available where normal type change requests can be auto-approved based on the decision policies.

As a part of demo data, the decision policies available are:
• Low risk auto approval policy, where the failed test count is zero.

• High risk manual approval policy, where the failed test count is greater than zero.

As a part of the demo flow, Flow Designer actions available are:

• **DevOps Set Change Window**
  
  Action used to set the change request start date.

• **Get Change Policy User/Group**
  
  Action used to fetch the user/group based on the change request approval policy.

• **DevOps Create Auto Approval Record**
  
  Action used by the user to create Auto-Approve/Auto-Reject approval records.
DevOps dashboard and Pipeline UI

Use the DevOps Insights dashboard to evaluate the results of your overall DevOps process. Use the DevOps Pipeline UI view to visualize your pipeline executions.
**DevOps Insights dashboard**

The DevOps Insights dashboard consists of a flexible graphical view that you can tailor to your needs, which is helpful for DevOps administrators when planning and providing updates.

ℹ️ **Note:** You must [Install DevOps Insights](#) to view the Insights Standard dashboard.

Use this dashboard to analyze operational and business reports and determine the overall efficiency and growth associated with your development processes.

- Monitor activities, KPIs, and metrics in a central dashboard.
- Track performance over time using ServiceNow Performance Analytics.
- Access breakdowns by pipeline, app, step, or team.

**DevOps Pipeline UI view**

The Pipeline UI view lets you visualize interactions and results across a pipeline execution. This graphical view shows pipeline step progression and other details, including change requests, step execution, artifact versions, test summaries, and commits.

**DevOps Insights Standard dashboard**

Use the DevOps Insights application with ServiceNow Performance Analytics to gain insight into your DevOps environment.

ℹ️ **Note:** Starting with version 1.14, unless these jobs were previously customized, you must select **Active** and set the **Run As** credentials to **System Administrator** to collect daily or historical Insights data:

- **[DevOps] Daily Data Collection (Daily)** (inactive by default)
  
  ℹ️ **Note:** This is a scheduled job (to be run on a regular basis) to collect daily DevOps data. For optimal performance, set this job to run during periods of low usage.

- **[DevOps] Historical Data Collection (On Demand)** (inactive by default)
  
  For new DevOps Insights installations, if you have already been using the DevOps app and you Installed DevOps Insights at a later date, run the Historical Data Collection job to collect historical Insights data.

  ℹ️ **Note:** This job is a one-time job to collect historical DevOps data, and not meant to be on a schedule. It might take awhile, so plan on running this job during a period of low usage.
Reports in the dashboard tabs get updated when the dashboard is refreshed.

**Change Acceleration**

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Changes Submitted - Yearly</td>
<td>Change Request</td>
<td>Total DevOps changes submitted yearly.</td>
</tr>
<tr>
<td>Avg Time to Close - Last 30 days</td>
<td>Change Request</td>
<td>Average time to close DevOps changes in the last 30 days.</td>
</tr>
<tr>
<td>Change Approval Rate - Last 30 days</td>
<td>Change Request</td>
<td>DevOps average change success rate for change requests in the last 30 days:</td>
</tr>
</tbody>
</table>

**Note:** This widget uses average aggregation and does not support multi-element selection.
<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Acceleration (continued)</td>
<td></td>
<td>[[([DevOps Change Success]] / [[DevOps Change]]) * 100</td>
</tr>
<tr>
<td>Note: This widget uses average aggregation and does not support multi-element selection.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-DevOps Change Approval Rate - Last 30 days</td>
<td>Change Request</td>
<td>Non-DevOps change approval rate for change requests in the last 30 days:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(([[Non-DevOps Change Approval]] / [[Non-DevOps Change]]) * 100</td>
</tr>
<tr>
<td>Note: Filter is not applicable to this widget.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change Request Volume</td>
<td>Change Request</td>
<td>Volume of change requests created for DevOps in the last 7 days.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compare the number of change requests created after you transition to DevOps so you can see the advantage of running DevOps in your environment.</td>
</tr>
<tr>
<td>Pending Changes per Pipeline</td>
<td>Step Execution</td>
<td>Number of change requests that have not been closed for each pipeline.</td>
</tr>
</tbody>
</table>
Change Acceleration (continued)

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Request</td>
<td></td>
<td>See the blockages in each pipeline that are keeping the change request from being completed so you can investigate the cause.</td>
</tr>
<tr>
<td>Average Time to Close Changes</td>
<td>Change Request</td>
<td>Average time to close DevOps changes by app.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compare DevOps change request statistics with non-DevOps change requests to see that DevOps change requests are getting resolved faster.</td>
</tr>
<tr>
<td>Changes Awaiting Approval</td>
<td>Change Request</td>
<td>Number of DevOps changes awaiting approval by date range.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compare DevOps change request statistics with non-DevOps change requests to see that DevOps change requests are getting resolved faster.</td>
</tr>
<tr>
<td>Non-DevOps Changes Awaiting Approval</td>
<td>Change Request</td>
<td>Number of Non-DevOps changes awaiting approval by date range.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compare DevOps change request statistics with non-DevOps change requests to see that DevOps change requests are getting resolved faster.</td>
</tr>
</tbody>
</table>
**Accelerate Metrics**

Starting with version 1.15, the Accelerate Metrics tab shows deployment frequency, lead time, MTTR, and change failure rate info.

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment Frequency - Monthly</td>
<td>Step Execution</td>
<td>Number of successful production deployments in a month.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Applies to steps of type <strong>Prod Deploy</strong> that are in completed state.</td>
</tr>
<tr>
<td>Average Lead Time</td>
<td>Pipeline Execution</td>
<td>Average of: ([Time the code is successfully pushed to production] - [Earliest commit time])</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Applies to steps of type <strong>Prod Deploy</strong> that are in completed state.</td>
</tr>
<tr>
<td>Report</td>
<td>Source list</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Mean Time to Resolve - Last 30 days</td>
<td>Database view joined by Incident, Change Request, Step Execution, Step, Pipeline, and App lists.</td>
<td>Average resolve time for an incident caused by a DevOps change in the last 30 days.</td>
</tr>
<tr>
<td>Change Failure Rate - Monthly</td>
<td>Change Request</td>
<td>Average change failure rate in a month.</td>
</tr>
<tr>
<td>Deployment Frequency</td>
<td>Step Execution</td>
<td>Number of successful production deployments in the last 30 days. Applies to steps of type <strong>Prod Deploy</strong> that are in completed state.</td>
</tr>
<tr>
<td>Change Failure Rate</td>
<td>Change Request</td>
<td>Average change failure rate in the last 30 days.</td>
</tr>
<tr>
<td>Mean Time to Resolve Trend</td>
<td>Database view joined by Incident, Change Request, Step Execution,</td>
<td>Daily average resolve time for an incident caused by a DevOps change.</td>
</tr>
</tbody>
</table>
Accelerate Metrics (continued)

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Step, Pipeline, and App lists.</td>
<td></td>
</tr>
</tbody>
</table>

**Lead Time**  
Pipeline Execution  
([Time the code is successfully pushed to production] - [Earliest commit time])  
Applies to steps of type **Prod Deploy** that are in completed state.

**Operational Stability**  
Starting with version 1.14, the Operational Stability tab shows:

- Service availability  
  Average service availability and daily service availability
- Mean time to resolve (MTTR)  
  Mean time to resolve and daily mean time to resolve

**Note:** You must install the Service Portfolio Management Foundation (com.snc.service_portfolio) plugin to see service availability widgets.

Demo data is also provided for the service availability widgets.

**Note:** You must install the Service Portfolio Management Foundation (com.snc.service_portfolio) plugin before installing the DevOps Insights application to see demo data.
### Operational Stability

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Time to Resolve - Last 30 days</td>
<td>Database view joined by Incident, Change Request, Step Execution, Step, Pipeline, and App lists.</td>
<td>Average resolve time for an incident caused by a DevOps change in the last 30 days.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This widget uses average aggregation and does not support multi-element selection.</td>
</tr>
<tr>
<td>Incidents - Monthly</td>
<td>Incident</td>
<td>Number of incidents in a month (based on pipeline steps of type <strong>Prod Deploy</strong>) linked to business service in CMDB.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This report provides an indication of environment stability.</td>
</tr>
<tr>
<td>Average Service Availability - Last 30 days</td>
<td>Database view joined by Service Availability, Service Offering, Business</td>
<td>Average service availability in the last 30 days (based on pipeline steps of type <strong>Prod Deploy</strong>) linked to business service in CMDB.</td>
</tr>
</tbody>
</table>
### Operational Stability (continued)

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service, Step, Pipeline, and App lists.</td>
<td>pipeline steps of type <strong>Prod Deploy</strong> linked to application service in CMDB. This report provides an indication of environment stability.</td>
<td></td>
</tr>
<tr>
<td>Outages - Monthly</td>
<td>Outage</td>
<td>Number of outages in a month (based on pipeline steps of type <strong>Prod Deploy</strong> linked to business service in CMDB. This report provides an indication of environment stability.</td>
</tr>
<tr>
<td>Mean Time to Resolve Trend</td>
<td>Database view joined by Incident, Change Request, Step Execution, Step, Pipeline, and App lists.</td>
<td>Daily average resolve time for an incident caused by a DevOps change.</td>
</tr>
<tr>
<td>Service Availability Trend</td>
<td>Database view joined by Service Availability, Service Offering, Business Service, Step, Pipeline, and App lists.</td>
<td>Daily average service availability (based on pipeline steps of type <strong>Prod Deploy</strong> linked to application service in CMDB. This report provides an indication of environment stability.</td>
</tr>
<tr>
<td>Incidents Trend</td>
<td>Incident</td>
<td>Daily number of incidents (based on pipeline steps of type <strong>Prod Deploy</strong> linked to business service in CMDB.</td>
</tr>
</tbody>
</table>
Operational Stability (continued)

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outage Trend</td>
<td>Outage</td>
<td>Daily number of outages (based on pipeline steps of type <strong>Prod Deploy</strong>) linked to business service in CMDB. This report provides an indication of environment stability.</td>
</tr>
</tbody>
</table>

**Development**

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit Frequency</td>
<td>Commit</td>
<td>Number of commits measured daily. Smaller more frequent commits are preferred</td>
</tr>
<tr>
<td>Report</td>
<td>Source list</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Average Branches per Repository</td>
<td>Branches, Repository</td>
<td>Average branches per repository on a given day.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This widget uses formula and does not support multi-element selection.</td>
</tr>
</tbody>
</table>
| Average Commits per Pipeline Execution      | Commit, Pipeline Execution | Average commits per pipeline on in the last 30 days:  
\[
\frac{[\text{Total number of commits}]}{[\text{Number of pipeline executions}]}  
\]  
**Note:** This widget uses average aggregation and does not support multi-element selection.  
A low number is preferable, which indicates a concentrated effort, versus switching from task to task without completion. |
| Commits without Work Item                   | Commit              | Commits made that are not tied to a work item, grouped by committer, in the last 30 days.  
This report is useful for investigating and |
Development (continued)

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>resolving why a commit is not tied to a work item, since all commits should be tied to a work item.</td>
</tr>
</tbody>
</table>

**Work Items**

**Work Item**

Number of work items that are complete or working in progress in the last 30 days.

**Note:** Filter is not applicable to this widget.

**Commit Insights**
## Commit Insights

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
</table>
| Active Committers              | Commit                      | Committers that submitted commits in the last 30 days. Shows how many active committers there are.  
|                                |                             | **Note:** This widget uses count distinct aggregation and does not support multi-element selection. |
| Average Commits per Committer  | Commit                      | Total number of commits in the last 30 days / Active committers. Shows how often committers are committing. A higher value is more favorable.  
|                                |                             | **Note:** This widget uses formula and does not support multi-element selection. |
| Average Files Added per Commit | Commit, Commit Details      | Total number of files added in the last 30 days / Total number of commits in the last 30 days. Shows how few files are committed at a time. A lower value is more favorable. |
## Commit Insights (continued)

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Note:" /> This widget uses formula and does not support multi-element selection.</td>
</tr>
<tr>
<td>% Commits Reverted</td>
<td>Commit</td>
<td>Commits reverted in the last 30 days / Total number of commits in the last 30 days. Shows how many commits have been reverted. A lower value is more favorable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Note:" /> This widget uses formula and does not support multi-element selection.</td>
</tr>
<tr>
<td>Top Committers</td>
<td>Commit</td>
<td>Committers with the highest number of commits in the last 30 days. Provides visibility into the users that commit the most.</td>
</tr>
<tr>
<td>Top Reverters</td>
<td>Commit</td>
<td>Committers with the highest number of reverts in the last 30 days. Provides visibility into the users that revert commits the most.</td>
</tr>
<tr>
<td>Commits Added per App</td>
<td>Commit, Commit Details</td>
<td>Number of commits added per app in the last 30 days.</td>
</tr>
</tbody>
</table>
Commit Insights (continued)

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Provides visibility into the development activity for each app.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: Filter is not applicable to this widget.</td>
</tr>
</tbody>
</table>

Deployments

Deployment frequency lets you know how often you are delivering value based on production deployments. Typically, more frequent deployments are desired.

Note: Metrics are based off production deployments (Type field is set to Prod Deploy in the app step).

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment Frequency - Monthly</td>
<td>Step Execution</td>
<td>Number of successful production deployments in a month.</td>
</tr>
<tr>
<td>Report</td>
<td>Source list</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Deployment Success Rate | Step Execution | Deploymnts success rate over the last 30 days.  
Deployment Success Rate = (Number of Successful Deployments in the last 30 days / Total Number of Deployments in the last 30 days) * 100  
Applies to steps of type **Prod Deploy** in completed state. |
| Failed Deployments      | Step Execution | Number of failed production deployments in the last 30 days.  
Applies to steps of type **Prod Deploy** that are in failed or user-canceled state. |
| Average Lead Time       | Pipeline Execution | Average of:  
((Time the code is successfully pushed to production) - [Earliest commit time])  
Applies to steps of type **Prod Deploy** that are in completed state. |
## Deployments (continued)

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
</table>
| Successful Production Deployments | Step Execution | Frequency of successful production deployments over time broken down by app.  
More frequent production deployments are preferred. |
| Failed Production Deployments   | Step Execution | Frequency of failed production deployments over time broken down by app.  
| Commit-to-Deploy Lead Time      | Pipeline Execution | Duration from the earliest commit time to production deployment (for a successful pipeline execution).  
Minimizing the time it takes from committing code to successfully running it in production is preferable.  
When the lead time is high, you can investigate the pipeline to identify the slowest steps. For example, a manual change approval process could increase lead time. |

**Note:** This widget uses average aggregation and does not support multi-element selection.
## System Health

### Task Execution Success Rate

- **Success Rate**: 70.87%
- **Change**: -0.55% (May 22: 71.42%)

### Number of Task Executions

- **Default**: 30 days

### Number of API calls

- **Default**: 30 days

### Report

<table>
<thead>
<tr>
<th>Report</th>
<th>Source list</th>
<th>Description</th>
</tr>
</thead>
</table>
| Task Execution Success Rate  | Task Executions | Success rate for tasks run by the execution tools over time: 
\[
\frac{[\text{Task Execution Success}]}{[\text{Task Execution}]} \times 100
\]

**Note:** This widget does not support multi-element selection.

<table>
<thead>
<tr>
<th>Number of Task Executions</th>
<th>Task Executions</th>
<th>Default number of tasks executions in the last 30 days.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of API calls</td>
<td>Event</td>
<td>Default number of API calls in the last 30 days.</td>
</tr>
</tbody>
</table>
DevOps Pipeline UI

Use the Pipeline UI to visualize interactions and results across a pipeline execution. This graphical view shows pipeline step progression and other details for each pipeline.

From DevOps, get a quick view of how everything is connected to see exactly what is happening with the pipeline and when. From the ServiceNow Change Management application, you can access the Pipeline UI and quickly see the commits, the committers, and other details for the change request in one place.

Starting with version 1.30, the Pipeline UI shows all attempts of any stage or job that has been rerun or restarted. For more information, see Restarting failed build or release pipeline jobs and stages.

Starting with version 1.19, the Pipeline UI shows the pipeline steps that ran instead of the steps configured in DevOps.

You can access the Pipeline UI using the related link from within certain DevOps forms, and also from a DevOps change request form:
- DevOps Pipeline form
- DevOps Pipeline Execution form
- Change Request form created by DevOps

Note: You must reload the view to update the status buttons in the pipeline execution History.

<table>
<thead>
<tr>
<th>Step run states (task execution status)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Green</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Grey</strong></td>
</tr>
<tr>
<td><strong>Yellow</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Step run states (task execution status) (continued)

Red  Failed.  At least one step execution failed.  Task execution end date is populated even when the change is rejected.

Note: The order the cards appear in the Pipeline UI is determined by the Order field in each pipeline step when you modeled your pipeline in DevOps. Skipped stages are not shown.

Starting with version 1.18, the order the cards appear in the Pipeline UI is by task execution.

<table>
<thead>
<tr>
<th>UI feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline steps</td>
<td>Timing.</td>
</tr>
<tr>
<td></td>
<td>• Start</td>
</tr>
<tr>
<td></td>
<td>• Last run</td>
</tr>
<tr>
<td></td>
<td>• Duration for each step</td>
</tr>
<tr>
<td></td>
<td>When the downstream task execution starts immediately after the upstream task execution, the duration is 0 seconds.</td>
</tr>
<tr>
<td></td>
<td>• Wait times in between task executions.</td>
</tr>
<tr>
<td>UI feature</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Wait time</td>
<td>Wait time is calculated as: Start time of the task execution minus the end time of the upstream task execution.</td>
</tr>
<tr>
<td>View change request</td>
<td>Change request record. Click directly into the change request of the step that was created by DevOps to view details of the change and take action.</td>
</tr>
<tr>
<td>Note:</td>
<td>• Starting with version 1.30, the change request record for the associated with the latest task execution displays.</td>
</tr>
<tr>
<td></td>
<td>• Starting with version 1.21, commits reverted in the same pipeline execution are not shown in the commit list.</td>
</tr>
<tr>
<td>Pipeline history</td>
<td>Pipeline Execution. Click a history tile to view the previous step details for a pipeline execution.</td>
</tr>
<tr>
<td>View all attempts</td>
<td>All attempts that the job has run in a step. Click the link in the relevant step to view all attempt details.</td>
</tr>
<tr>
<td>Artifacts</td>
<td>• Artifact versions  ◦ Work items  ◦ Commits  ◦ Packages  • Commits  • Work items</td>
</tr>
<tr>
<td>UI feature</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>Starting with version 1.21, commits reverted in the same pipeline execution are not shown in the commit list.</td>
</tr>
<tr>
<td><strong>Quality</strong></td>
<td>• Test Results. View the build test results to see what tests passed or failed. Starting with version 1.15, the quality card contains test summaries:</td>
</tr>
<tr>
<td></td>
<td>◦ Test type and test category in the format:</td>
</tr>
<tr>
<td></td>
<td>test type/test category</td>
</tr>
<tr>
<td></td>
<td>◦ Native ID of the step</td>
</tr>
<tr>
<td></td>
<td>◦ Test pass percentage (unit and functional tests only)</td>
</tr>
<tr>
<td></td>
<td>◦ Throughput (performance tests only)</td>
</tr>
<tr>
<td></td>
<td>◦ Step name</td>
</tr>
<tr>
<td></td>
<td>• Software Quality Results</td>
</tr>
<tr>
<td></td>
<td>Starting with version 1.27, view all the software quality (Sonar scan) results grouped by project name that were fetched as part of the selected pipeline. You can view the scan results for all categories in a pipeline execution step.</td>
</tr>
<tr>
<td></td>
<td>You can customize the priority of categories from the <strong>DevOps &gt; Administration &gt; Properties &gt; DevOps Properties Category</strong> &gt; Software quality categories shown by default in the Pipeline UI view field in a comma separated format.</td>
</tr>
<tr>
<td>UI feature</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>◦ Click the gear icon to modify the category view in the pipeline UI.</td>
<td></td>
</tr>
<tr>
<td>◦ Click the step execution record to view the corresponding Software Quality Scan Summary record.</td>
<td></td>
</tr>
<tr>
<td>◦ Click the Vulnerabilities count record to view the Software Quality Scan Detail record and the corresponding sub category details.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** When you reattempt running a stage or a pipeline job which contains a test or a software quality scan, the results get appended with the attempt number.

Starting with version 1.16, click directly into DevOps change requests, step executions, artifacts, artifact versions, work items, test summaries, and reattempts in flyout windows.

**Pipeline UI - Artifact version flyout**

![Pipeline UI - Artifact version flyout](image-url)
User-created DevOps integrations

User-created integrations are for integrating additional planning, coding, and test tools not included in the integrations provided with the DevOps application. The DevOps application includes tool definitions for integrating some common planning, coding, and test tools, but you can also set up user-created integrations for additional tools in your DevOps environment.

⚠ Note: User-created integrations for orchestration tools is not supported.

DevOps integration objects

DevOps tool integration consists of these objects.
Tool capability actions

- Starting with version 1.9, **Connect** action:

  When connecting, the subflow for the specific tool is called and the connection state is updated. The connection status message is shown on the form.

  See [Connect capability subflow](#) for more details.

- Starting with version 1.9, **Discover** action:

  When discovering, an import request record is created and the subflow for the specific tool is called (as defined in the Integration Capability record). **Detail** and **Status** fields in the Import Request record are updated with the number of items discovered, updated, and failed.

  The transformed payload consists of an array of objects as a JSON string. Elements vary depending on the tool type.

  See [Discover capability subflow](#) for more details.

  **Note:** Pagination is not supported in the Discover action for user-created integrations.

- Starting with version 1.12, **Import** action:

  Import action does not support historical import functionality.

- Starting with version 1.12, **Lookup** action:

  The Lookup main flow is provided to support artifact tool type in a subflow created by your integration developer.

- **Notification** (webhook) action:

  The source tool is configured manually (by your integration developer) to send raw data to the ServiceNow instance. The raw payload is then transformed into a standard JSON object using a subflow.

  See the [Notification capability subflow](#) and the [DevOps - POST /devops/tool/{capability}](#) endpoint of the [DevOps API](#) for more details.

  **Note:** Starting with version 1.11, if a subflow is not specified, default handling of notifications occurs (original payload is automatically copied to transformed payload).

  This behavior is useful when the transactional data of the tool is supported by ServiceNow DevOps as is.

  See the expected standard payloads in the [Notification capability subflow](#) for more details.
DevOps integration configuration overview

Tool integration configuration can be completed by your integration developer and your DevOps admin.

Integration developer

- Create a tool integration record in DevOps to define the tool you are integrating (source tool).
- Create a Flow Designer subflow to collect and transform data from the tool you are integrating (source tool).
- Create a tool capability mapping record in DevOps to map the tool integration record to the tool type capability.

⚠️ Note: Notifications (webhook) capability is supported. Starting with version 1.9, connect and discover capabilities are also supported.

- Create an integration capability record in DevOps to specify the action for the tool type capability.

DevOps admin

- Create a (planning, coding, or test) tool record in DevOps to connect to the tool you are integrating (source tool).

⚠️ Note: The tool integration record must be specified in the Tool field of the tool record.

- Configure the source tool with the webhook and credentials.

Inbound events

An inbound event serves as a staging area for the notifications flow that supports reprocessing of failed payloads. Meaning, a record in an error state from a failed integration or transformation can be retried.

If an Inbound Event record is in the Error state, the flow was not able to insert the record successfully into the core DevOps tables.

Common errors can be resolved with these actions.

Inbound event error states

<table>
<thead>
<tr>
<th>Error</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing required fields</td>
<td>The transformed payload does not match the standard payload.</td>
</tr>
</tbody>
</table>

© 2021 ServiceNow, Inc. All rights reserved.
ServiceNow, the ServiceNow logo, Now, and other ServiceNow marks are trademarks and/or registered trademarks of ServiceNow, Inc., in the United States and/or other countries. Other company names, product names, and logos may be trademarks of the respective companies with which they are associated.
## Inbound event error states (continued)

<table>
<thead>
<tr>
<th>Error</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repository not marked for tracking</td>
<td>The commit cannot be inserted. The DevOps admin needs to track the repository.</td>
</tr>
<tr>
<td>[Subflow] has not been published within application scope [app_scope]</td>
<td>The subflow is created but not published yet.</td>
</tr>
<tr>
<td>Timeout exception</td>
<td>The subflow takes more time than the value set in the property: <code>com.glide.hub.flow_api.default_execution_time</code></td>
</tr>
<tr>
<td></td>
<td>See [FlowAPI - executeSubflowQuick(String name, Map inputs, Number timeout)] for more details.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Starting with version 1.10, the execution of the subflow exceeds the value set in the <strong>Timeout</strong> field in the Integration Capability record.</td>
</tr>
<tr>
<td>Did not find a matching subflow for notification capability and [tool_integration_sys_id] tool integration</td>
<td>The flow was not able to find the matching subflow. Verify the integration setup procedure.</td>
</tr>
<tr>
<td>Payload does not match the expected capability.</td>
<td>The <strong>Original payload</strong> (payload being sent) is a different <strong>Capability</strong> type than the tool type capability configured in your tool capability mapping.</td>
</tr>
<tr>
<td></td>
<td>The payload type must match the tool type capability configured in your tool integration.</td>
</tr>
</tbody>
</table>
Note: An inbound event record is not created when any of the following conditions occur:

- Source tool has not passed the tool ID as a query parameter.
- Source tool has passed on a tool ID, but there is no matching tool ID in the instance.

**Tool mappings**

A tool can be mapped to multiple capabilities.

<table>
<thead>
<tr>
<th>Tool Integration</th>
<th>Tool Type Capability</th>
<th>Tool Capability Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile Development 2.0</td>
<td>Plan</td>
<td>Plan</td>
</tr>
<tr>
<td>Azure DevOps</td>
<td>Code</td>
<td>• Agile Development 2.0 - Plan</td>
</tr>
<tr>
<td>Bitbucket</td>
<td>Orchestration</td>
<td>• Azure DevOps - Plan</td>
</tr>
<tr>
<td>GitHub</td>
<td>Artifact</td>
<td>• Jira - Plan</td>
</tr>
<tr>
<td>GitHub Enterprise</td>
<td>Test</td>
<td></td>
</tr>
<tr>
<td>GitLab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jenkins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jira</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Agile DevOps - Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bitbucket - Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GitHub - Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GitHub Enterprise - Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GitLab - Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Azure DevOps - Orchestration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Jenkins - Orchestration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GitLab - Orchestration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Azure DevOps - Test</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Jenkins - Test</td>
</tr>
</tbody>
</table>

A tool capability mapping can be mapped to multiple actions.
<table>
<thead>
<tr>
<th>Tool Capability Mapping</th>
<th>Tool Action</th>
<th>Integration Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Agile Development 2.0 - Plan</td>
<td>• Connect</td>
<td>Agile Development 2.0</td>
</tr>
<tr>
<td></td>
<td>• Discover</td>
<td>• Agile Development 2.0 - Plan - Connect</td>
</tr>
<tr>
<td></td>
<td>• Import</td>
<td>• Agile Development 2.0 - Plan - Discover</td>
</tr>
<tr>
<td></td>
<td>• Lookup</td>
<td>• Agile Development 2.0 - Plan - Import</td>
</tr>
<tr>
<td></td>
<td>• Notification</td>
<td>• Agile Development 2.0 - Plan - Notification</td>
</tr>
<tr>
<td>• Azure DevOps - Plan</td>
<td></td>
<td>Azure DevOps</td>
</tr>
<tr>
<td>• Azure DevOps - Code</td>
<td></td>
<td>• Azure DevOps - Plan - Connect</td>
</tr>
<tr>
<td>• Azure DevOps - Orchestration</td>
<td></td>
<td>• Azure DevOps - Plan - Discover</td>
</tr>
<tr>
<td>• Bitbucket - Code</td>
<td></td>
<td>• Azure DevOps - Plan - Notification</td>
</tr>
<tr>
<td>• GitHub - Code</td>
<td></td>
<td>• Azure DevOps - Code - Discover</td>
</tr>
<tr>
<td>• GitHub Enterprise - Code</td>
<td></td>
<td>• Azure DevOps - Code - Notification</td>
</tr>
<tr>
<td>• GitLab - Code</td>
<td></td>
<td>• Azure DevOps - Orchestration - Discover</td>
</tr>
<tr>
<td>• GitLab - Orchestration</td>
<td></td>
<td>• Azure DevOps - Orchestration - Notification</td>
</tr>
<tr>
<td>• Jenkins - Orchestration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Jira - Plan</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2021 ServiceNow, Inc. All rights reserved.
ServiceNow, the ServiceNow logo, Now, and other ServiceNow marks are trademarks and/or registered trademarks of ServiceNow, Inc. in the United States and/or other countries. Other company names, product names, and logos may be trademarks of the respective companies with which they are associated.
<table>
<thead>
<tr>
<th>Tool Capability Mapping</th>
<th>Tool Action</th>
<th>Integration Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub Enterprise</td>
<td></td>
<td>• GitHub - Code - Connect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GitHub - Code - Discover</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GitHub - Code - Import</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GitHub - Code - Notification</td>
</tr>
<tr>
<td>GitLab</td>
<td></td>
<td>• GitLab - Code - Connect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GitLab - Code - Discover</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GitLab - Code - Notification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GitLab - Orchestration - Notification</td>
</tr>
<tr>
<td>Jenkins</td>
<td></td>
<td>• Jenkins - Orchestration - Connect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Jenkins - Orchestration - Discover</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Jenkins - Orchestration - Import</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Jenkins - Orchestration - Notification</td>
</tr>
<tr>
<td>Jira</td>
<td></td>
<td>• Jira - Plan - Connect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Jira - Plan - Discover</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Jira - Plan - Import</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Jira - Plan - Notification</td>
</tr>
</tbody>
</table>

Multiple test types can be mapped to each tool integration.

<table>
<thead>
<tr>
<th>Test Type</th>
<th>Tool Integration</th>
<th>Test Type Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit:</td>
<td>• Azure DevOps</td>
<td>Azure DevOps</td>
</tr>
<tr>
<td></td>
<td>• JUnit</td>
<td></td>
</tr>
<tr>
<td>Functional:</td>
<td>• Integration</td>
<td>• Azure DevOps - JUnit</td>
</tr>
<tr>
<td></td>
<td>• Regression</td>
<td>• Azure DevOps - Integration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Azure DevOps - Regression</td>
</tr>
</tbody>
</table>
### DevOps test tool integration

Test tool integration lets you view test results in DevOps for Jenkins, Azure DevOps, and GitLab unit, functional, and performance tests.

Starting with version 1.15, JUnit test type integration is supported for Jenkins and Azure DevOps.

Starting with version 1.23, JUnit test type integration is supported for GitLab.

⚠️ **Note:** For other test types, use the `DevOps - POST /devops/tool/{capability}` endpoint of the DevOps API.

Starting with version 1.20:

- Selenium tests run and published using TestNG are reported by the Jenkins plugin for ServiceNow DevOps.
- Test type categorization is supported.
- Additional tests results reported by tools, such as Apache JMeter, can be processed in DevOps using custom Flow Designer subflows (no pipeline changes required).
<table>
<thead>
<tr>
<th>Category</th>
<th>Test type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit</td>
<td>JUnit (default)</td>
</tr>
<tr>
<td></td>
<td>You can change the default test type by modifying the [sn_devops.default_test_type] DevOps property.</td>
</tr>
<tr>
<td>Functional</td>
<td>• Integration</td>
</tr>
<tr>
<td></td>
<td>• Regression</td>
</tr>
<tr>
<td></td>
<td>• Smoke</td>
</tr>
<tr>
<td></td>
<td>• System</td>
</tr>
<tr>
<td></td>
<td>• User Acceptance</td>
</tr>
<tr>
<td>Performance</td>
<td>Load</td>
</tr>
</tbody>
</table>

**Test type mapping**

The test type mapping connects the test type and entity being tested with the DevOps tool (DevOps > Integrations > Test Type Mappings module.)

An accurate test type mapping ensures that the test type always appears as intended in the test summary results.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test type</td>
<td>• JUnit</td>
</tr>
<tr>
<td></td>
<td>• Integration</td>
</tr>
<tr>
<td></td>
<td>• Regression</td>
</tr>
<tr>
<td></td>
<td>• Smoke</td>
</tr>
<tr>
<td></td>
<td>• System</td>
</tr>
<tr>
<td></td>
<td>• User Acceptance</td>
</tr>
<tr>
<td></td>
<td>• Load</td>
</tr>
<tr>
<td>DevOps Entity Id</td>
<td>Table name</td>
</tr>
<tr>
<td></td>
<td>DevOps table name that contains the entity linked to the test results (in the test report payload).</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Document</td>
<td>Name of the entity specified in the selected table.</td>
</tr>
<tr>
<td></td>
<td>For example, the name of the step, pipeline, artifact, or package.</td>
</tr>
<tr>
<td>Test File Paths</td>
<td>Path to the generated test result file on the Jenkins server.</td>
</tr>
<tr>
<td>(Jenkins tests only)</td>
<td>This is useful so test reports with attributes that don't conform to JUnit or TestNG implementation, such as JMeter for example, can still be leveraged by DevOps. Separate multiple files by a comma.</td>
</tr>
<tr>
<td>Tool integration</td>
<td>Tool that's running the test.</td>
</tr>
<tr>
<td>DevOps Table</td>
<td>DevOps table that corresponds to the table name in the DevOps Entity Id setting.</td>
</tr>
</tbody>
</table>
Transforming a raw test payload

You can transform a raw test report (a report containing attributes that do not conform to JUnit or TestNG implementation), such as JMeter for example, by configuring the DevOps Test Subflow Policy decision table to call a custom subflow (Decision Tables > Decision Tables module).

ℹ️ Note: You must create the custom Flow Designer subflow that transforms the raw test payload.

If there is more than one test type in a performance stage, you can use the DevOps Test Type Policy decision table to configure the test type for each test so the test result payloads are transformed correctly.

### DevOps decision tables

<table>
<thead>
<tr>
<th>Decision table</th>
<th>Purpose</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevOps Test Subflow Policy</td>
<td>To automatically call a custom subflow that transforms the raw payload received by the tool.</td>
<td>Create a decision that specifies the custom subflow to call when the raw payload is received. Set the conditions to contain fields that would be included in the raw payload. For example, to call Jenkins BZ Performance Test custom subflow:</td>
</tr>
</tbody>
</table>
DevOps decision tables (continued)

<table>
<thead>
<tr>
<th>Decision table</th>
<th>Purpose</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Conditions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Test Type is Load (Load is the test type configured for performance tests)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Test Result Payload contains throughput</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Test Result Payload contains concurrency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Answer: Flow: Jenkins BZ Performance Test</td>
</tr>
<tr>
<td>DevOps Test Type Policy</td>
<td>To automatically set test types when more than one type of test is configured in a performance test stage. This is necessary so the results for the second test type get transformed correctly. For example, when both a Load performance test and a JUnit performance test are mapped in the same DevOps step, the JUnit test results won't get formatted correctly.</td>
<td>Create a decision for each type of test in the performance test stage to set the test type. Load test:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Conditions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ Step is Perf Tests</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ Test Result Payload contains throughput</td>
</tr>
<tr>
<td></td>
<td></td>
<td>◦ Test Result Payload contains concurrency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Answer: TestType: Load</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JUnit test:</td>
</tr>
</tbody>
</table>
DevOps decision tables (continued)

<table>
<thead>
<tr>
<th>Decision table</th>
<th>Purpose</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>unless a decision is created. Decision inputs:</td>
<td>• Conditions:</td>
</tr>
<tr>
<td></td>
<td>• Step</td>
<td>◦ Step is Perf Tests</td>
</tr>
<tr>
<td></td>
<td>• Test Result Payload</td>
<td>◦ Test Result Payload does not contain throughput</td>
</tr>
<tr>
<td></td>
<td>• Tool Integration</td>
<td>◦ Test Result Payload does not contain concurrency</td>
</tr>
<tr>
<td></td>
<td>• Pipeline</td>
<td>• Answer: TestType: JUnit</td>
</tr>
</tbody>
</table>

DevOps multiple performance test types

```java
1  stage('Checkout') {
2    snDevOpsStep()
3    snDevOpsChange()
4    node {
5      echo 'Checking out..'  
6        checkout scm
7    }
8 }
9  stage('Tests'){
10     snDevOpsStep()
11     node {
12       sh 'mvn clean test'
13       step([$class: 'Publisher', reportFilenamePattern: '**/testng-results.xml'])
14    }
15 }
16  stage('Perf Tests'){
17     snDevOpsStep()
18     node {
19       bzt "load_test1.xml"
20         junit '**/xunit.xml'
21    }
22 }
23 }
```

DevOps multiple test type mappings

<table>
<thead>
<tr>
<th>Test Type Mappings</th>
<th>Search</th>
<th>Name</th>
<th>Test Type</th>
<th>Tool Integration</th>
<th>Name</th>
<th>DevOps Entity Id</th>
<th>DevOps Table</th>
<th>Test File Paths</th>
</tr>
</thead>
</table>
DevOps decision table decision

Test summary results
You can view test summary results these ways.

- **DevOps > Test Results** module (Test Summaries and Performance Test Summaries).
- **DevOps change request** - Test Results related list.
- **DevOps Pipeline UI** - Quality tile.

DevOps performance test summary example

Expected standard JSON Notification capability payload - Test tool

Functional:

```json
{
  "name": "CorpSite-selenium#55",
  "duration": 78.802,
```
"passedTests": 4,
"failedTests": 0,
"skippedTests": 0,
"blockedTests": 0,
"totalTests": 4,
"startTime": "2020-06-30T18:12:31Z",
"finishTime": "2020-06-30T18:12:31Z",
"passingPercent": 100,

// Use Artifact OR Package OR Build + Stage + PipelineName Attributes
"packages": [{"name": "CorpSite-pkg1"}],
"artifacts": [{"name": "CorpSite-artifact", "version": "1.0.0"}],
"buildNumber": "55",
"stageName": "test",
"pipelineName": "CorpSite-selenium",
}

Performance:

{
"name": "Performance Tests",
"url": "http://abc.com",
"startTime": "2020-06-30T18:12:31Z",
"finishTime": "2020-06-30T18:12:31Z",
"duration": 78.802,
"maximumVirtualUsers": "",
"throughput": "",
"maximumTime": "",
"minimumTime": "",
"averageTime": "",
"ninetyPercent": "",
"standardDeviation": "",

// Use Artifact OR Package OR Build + Stage + PipelineName Attributes
"packages": [{"name": "CorpSite-pkg1"}],
"artifacts": [{"name": "CorpSite-artifact", "version": "1.0.0"}],
"buildNumber": "55",
"stageName": "test",
"pipelineName": "CorpSite-Performance",
}
Configure a test tool in DevOps

Configure a test tool in DevOps to view unit, functional, and performance test results.

Before you begin
Role required: sn_devops.admin

Procedure
1. Navigate to DevOps > Integrations > Test Type Mappings and create a record to map the test type to the integration tool.

2. In the Test Type Mapping record, use the search icon (🔍) to select **Test type** and **Tool integration** field values.

3. Starting with version 1.20, click the search icon (🔍) in the **DevOps Entity Id** field, and fill in the test details.

<table>
<thead>
<tr>
<th>Table name</th>
<th>DevOps table name that contains the entity linked to the test results.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>Name of the entity specified in the selected table.</td>
</tr>
</tbody>
</table>

See the Test type mapping section for more details.
4. **Optional:** Starting with version 1.20, to capture the raw test payload of non-JUnit or -TestNG tests, enter an .xml filename and path (comma separate multiple files).

(Optional) You can transform a raw payload using decision tables and a custom Flow Designer subflow.

ℹ️ **Note:** You must create the custom subflow that transforms the raw payload.

See the **Transforming a raw test payload** section for more details on configuring decision tables.

5. Run the test and view test results by navigating to **DevOps > Test Results** (Test Summaries and Performance Test Summaries).

---

**Example:**

```plaintext
1 stage('Checkout') {
2   snDevOpsStep()
3   snDevOpsChange()
4   node {
5     echo 'Checking out..'
6     checkout scm
7   }
8 }
9 stage('Tests'){
10   snDevOpsStep()
11   node {
12     sh 'mvn clean test'
13     step([$class: 'Publisher', reportFilenamePattern: '**/testng-results.xml'])
14   }
15 }
16 stage('Perf Tests'){
17   snDevOpsStep()
18   node {
19     bzt "load_test1.yml"
20     //junit '**/xunit.xml"
21   }
22 }
```
Create a DevOps tool integration

To create a DevOps tool integration, your integration developer configures DevOps tool integration settings, and a Flow Designer subflow to collect and transform data from the source tool. Then your DevOps admin configures your DevOps tool connections.

Before you begin

Note: When creating an integration as a scoped app, the system admin must assign these roles to the integration developer so the integration developer is able to create tool integration and integration capability records for the specific scope.

- Developer role for the scoped app
- DevOps admin role

Role required: sn_devops.admin

About this task

Creating a DevOps tool integration procedure involves configuration by both your integration developer and your DevOps admin.

- Your integration developer creates a tool integration record in DevOps, a Flow Designer subflow, a tool capability mapping, and an integration capability record in DevOps to map the capabilities and actions together.

Note: Notification, connect, and discover capabilities are supported.

- Your DevOps admin sets up DevOps connections (planning or coding tool), and configures the source tool with the webhook and credentials.

This procedure provides detailed steps to create your DevOps tool integration.

Procedure

1. Integration developer:
   Configure the source tool integration capabilities and actions, and a subflow.

   a. Navigate to DevOps > Integrations > Tool Integrations and create a record to define the tool you are integrating (source tool).

      Note: Do not edit the tool integration records provided with the DevOps application.
DevOps tool integration

<table>
<thead>
<tr>
<th>Tool label</th>
<th>Sample Code Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Code Tool [sn_devops_tool]</td>
</tr>
<tr>
<td>Use packageable integrations</td>
<td>Selected</td>
</tr>
<tr>
<td>Integration version</td>
<td>1.0</td>
</tr>
<tr>
<td>Active</td>
<td>Selected</td>
</tr>
</tbody>
</table>

b. Navigate to **Flow Designer** > **Designer** and create a subflow to collect and transform data from the tool you are integrating (source tool).

**Note:** The **Run As** field must be set to System User, and the **Inputs** label must be set to current variable.

<table>
<thead>
<tr>
<th>Notification subflow properties</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code Tool Notification</td>
</tr>
<tr>
<td>Application</td>
<td>Sample Integration App</td>
</tr>
<tr>
<td>Accessible From</td>
<td>All application scopes</td>
</tr>
<tr>
<td>Description</td>
<td>Code tool for integration app</td>
</tr>
<tr>
<td>Run As</td>
<td>System User</td>
</tr>
</tbody>
</table>

The subflow must contain Get More Data via API calls, and/or transform the original payload. Copy the transformed payload into the inbound events record.

**Note:** Do not edit the DevOps main flow.

c. Navigate to **DevOps** > **Integrations** > **Tool Capability Mappings** and create a record to map the tool integration record to the tool type capability.

<table>
<thead>
<tr>
<th>DevOps tool capability mapping</th>
<th>Tool integration</th>
<th>Sample tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool type capability</td>
<td></td>
<td>Code</td>
</tr>
</tbody>
</table>

d. Navigate to **DevOps** > **Integrations** > **Integration Capabilities** and create a record to specify the action for the tool capability mapping.
⚠ Note: Do not edit the integration capability records provided with the DevOps application.

<table>
<thead>
<tr>
<th>DevOps integration capability</th>
<th>Sample Code Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool integration</td>
<td>Sample Code Tool</td>
</tr>
<tr>
<td>Capability mapping</td>
<td>Sample Code Tool</td>
</tr>
</tbody>
</table>

⚠ Note: Do not edit tool type capability records.

<table>
<thead>
<tr>
<th>Action</th>
<th>Notification</th>
</tr>
</thead>
</table>

⚠ Note: Do not edit tool action records.

<table>
<thead>
<tr>
<th>Active</th>
<th>Selected</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Timeout (ms)</th>
<th>Timeout for the corresponding subflow. If execution of the subflow exceeds this value, a timeout exception occurs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>in milliseconds (ms). Default is 45,000 (45 seconds).</td>
</tr>
</tbody>
</table>

⚠ Note: Starting with version 1.10.

<table>
<thead>
<tr>
<th>Subflow name</th>
<th>x_snc_sample_integ.code_tool_notification</th>
</tr>
</thead>
<tbody>
<tr>
<td>The name is prefixed by the scope name and a dot (.) before the actual subflow name.</td>
<td></td>
</tr>
<tr>
<td>For example, given:</td>
<td></td>
</tr>
<tr>
<td>• connect_code_tool subflow</td>
<td></td>
</tr>
<tr>
<td>• my_app_scope scope</td>
<td></td>
</tr>
<tr>
<td>The value for this field is</td>
<td></td>
</tr>
<tr>
<td>my_app_scope.connect_code_tool.</td>
<td></td>
</tr>
</tbody>
</table>

⚠ Note: Starting with version 1.11, if the Subflow name field is left blank for a Notification capability, default handling of notifications occurs. |

See Notification tool capability action.

<table>
<thead>
<tr>
<th>Domain</th>
<th>global</th>
</tr>
</thead>
</table>

2. DevOps admin:

Configure the connection from DevOps to the source tool.
Refer to Setting up DevOps tools for more details on completing this step.

**a.** The tool record must contain:

- Reference to the tool integration record created by the integration developer (in the **Tool** field)
- Tool label
- Connection alias (connection and credential)

**b.** Copy the notification (webhook) created on the DevOps planning tool to the source tool service hook of the notification endpoint and set the credentials to devops.integration.user.

You can view the state of integration events in the Inbound Event list (**DevOps > Administration > Inbound Events**).

The inbound event record state is set to **Processed** once the object has been inserted into the DevOps Core table. Event states include New, In Progress, Processed, Unmatched, and Error.

**3.** If the tool integration record and the subflow are created in a different scope, the DevOps admin must create two new Cross scope privileges records to allow the app to access the inbound events table.

Navigate to **System Applications > Application Cross-Scope Access** and create read and write cross scope privileges records to allow your app to access the inbound events table.

<table>
<thead>
<tr>
<th>Field</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Scope</td>
<td>Auto populated based on the current application</td>
<td></td>
</tr>
<tr>
<td>Target Scope</td>
<td>DevOps</td>
<td></td>
</tr>
<tr>
<td>Target Name</td>
<td>sn_devops_inbound_event</td>
<td></td>
</tr>
<tr>
<td>Target Type</td>
<td>Table</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>Read</td>
<td>Write</td>
</tr>
<tr>
<td>Status</td>
<td>Allowed</td>
<td></td>
</tr>
</tbody>
</table>
Example:

DevOps tool integration

This record is in the DevOps Integrations application, but DevOps is the current application. To edit this record click here.

<table>
<thead>
<tr>
<th>Tool label</th>
<th></th>
<th>Tool name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Azure DevOps</td>
<td></td>
<td>sn_devops_Azure_DevOps</td>
<td></td>
</tr>
</tbody>
</table>

Use packageable integration

<table>
<thead>
<tr>
<th>Integration version</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td></td>
</tr>
</tbody>
</table>

Integration Capabilities (1)  Tool Capability Mappings (1)

Flow Designer subflow properties

Subflow Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlanView processor</td>
<td>-- None --</td>
</tr>
</tbody>
</table>

Application

Plan View Integration

Accessible From

All application scopes

Category

Description

Describe your subflow

Run As

System User
### DevOps tool capability mapping

**Tool label**: Azure DevOps  
**Tool name**: sn_devops_ints_Azure_DevOps  
**Integration version**: 5.1  
**Active**: Yes

#### Integration Capabilities (3)

<table>
<thead>
<tr>
<th>Tool type capability</th>
<th>Name</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Azure DevOps-Code</td>
<td>global</td>
</tr>
<tr>
<td>Orchestration</td>
<td>Azure DevOps-Orchestration</td>
<td>global</td>
</tr>
<tr>
<td>Plan</td>
<td>Azure DevOps-Plan</td>
<td>global</td>
</tr>
</tbody>
</table>

### Integration capability

**Tool Integration**: Azure DevOps  
**Capability mapping**: Azure DevOps-Plan  
**Action**: Discover  
**Active**: Yes  
**Timeout (ms)**: 45,000  
**Subflow name**: sn_devops_ints.azure_devops_discover Boards

### Planning tool

**Name**: PlanView  
**Tool**: PlanView  
**Connection alias**: sn_devops_planview  
**Tool URL**  
**Connection state**: Disconnected  
**Last discovery**:  
**Webhook URL**: [link](https://DevOps integration/user ID=Integration/user password=Integration/service-now.com/api/sn_devops/CI/DevOps/plan?tool=2250c-97b8c-4c-001f134e5a9f&cb=0)
Cross scope access records (read and write)

Creating DevOps subflows

For user-created integrations, create a DevOps Flow Designer subflow to collect and transform data from the tool you are integrating.

ServiceNow Flow Designer is a Now Platform feature your integration developer can use to automate processes using a sequence of actions. Trigger conditions start the flow, and variables pass information between actions.

See Flow Designer Subflows for information on how to create a subflow.

Notification capability subflow

<table>
<thead>
<tr>
<th>Item</th>
<th>Expected value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Label: current&lt;br&gt;Type: Reference.Inbound Event</td>
</tr>
<tr>
<td>Outputs</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Transform the original payload, and save the final payload in the `transformed_payload` field.

Expected standard JSON Notification capability payload - Planning tool

ℹ️ **Note:** The version attribute in the JSON payload is optional. Even if you do not provide the version attribute, the payload is processed successfully.

```json
{
    "id": "STR1234",
    "type": "Story",
    "shortDescription": "Planning API Spec",
    "state": "In-progress",
    "createdDateTime": "1970-01-01T08:15:30-05:00",
    "assignedTo": {
        "name": "Leo Neo",
        "userName": "lenn",
```
Expected standard JSON Notification capability payload - Coding tool

```json
{
  "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "url": "https://github.com/mycompany/mobileplatform/commit/3fa85f6457174562b3fc2c963f66afa6",
  "committedDate": "1970-01-01T08:15:30-05:00",
  "repository": {
    "name": "Platform-Mobile",
    "url": "https://github.com/mycompany/mobileplatform"
  },
  "branch": {
    "name": "master",
    "path": "refs/heads/master"
  },
  "committer": {
    "email": "lenn@smithworksinc.com"
  }
}
```
"details": [
{
"additions": 0,
"deletions": 0,
"totalChanges": 0,
"file": "src/test/java/com/mycompany/app/App.java",
"action": "modified",
"changes":
"%40%40%20-30%2C6%20%2B30%2C18%20%40%40%20public%20void%20testAppConstructo
r%28%29%20%7B%0A%20%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%20%7D%0A%20%0A%
2B%20%20%20%20%40Test%0A%2B%20%20%20%20public%20void%20testDatabase%28%29%0A%
2B%20%20%20%20%7B%0A%2B%09%2F%2FThis%20is%20a%20test%20function%0A%2B%20%20%2
0%20%20%20%20%20App.main%28null%29%3B%0A%2B%20%20%20%20%20%20%20%20try%20%7B%
0A%2B%20%20%20%20%20%20%20%20%20%20%20%20assertEquals%28%22Hello%20World%21%2
2%20%2B%20System.getProperty%28%22line.separator%22%29%2C%20outContent.toStri
ng%28%29%29%3B%0A%2B%20%20%20%20%20%20%20%20%7D%20catch%20%28AssertionError%2
0e%29%20%7B%0A%2B%20%20%20%20%20%20%20%20%20%20%20%20fail%28%22%22message%22%
20is%20not%20%22Hello%20World%21%22%22%29%3B%0A%2B%20%20%20%20%20%20%20%20%7
D%0A%2B%20%20%20%20%7D%0A%2B%0A%20%20%20%20%20%40Test%0A%20%20%20%20%20publi
c%20void%20testAppMain%28%29%0A%20%20%20%20%20%7B"
}
]
}

Expected standard JSON Notification capability payload - Orchestration tool
{
"toolId": "bc1d9454dbdb0810ae77f3c61d9619d1",
"buildNumber": "100",
"nativeId": "HILR/Prod #100",
"name": "HILR/Prod",
"id": "HILR/Prod #100",
"url":
"https://dev.azure.com/lenn/CorpSite-ADO/_build/results?buildId=100#Prod/",
"isMultiBranch": "false",
"orchestrationTaskUrl":
"https://dev.azure.com/lenn/CorpSite-ADO/_build?name=HILR#Prod",
"orchestrationTaskName": "CorpSite-ADO/HILR#Prod",
"upstreamTaskUrl":
"https://dev.azure.com/lenn/CorpSite-ADO/_build/results?buildId=100#UAT/",
"upstreamId": "CorpSite-ADO/HILR#UAT",
"result": "building",
"startDateTime": "2020-03-20 22:59:27"

© 2021 ServiceNow, Inc. All rights reserved.
ServiceNow, the ServiceNow logo, Now, and other ServiceNow marks are trademarks and/or registered trademarks of ServiceNow, Inc., in the United States and/or other countries.
Other company names, product names, and logos may be trademarks of the respective companies with which they are associated.

231


Expected standard JSON Notification capability payload - Test tool

Functional:

```json
{
    "name": "CorpSite-selenium#55",
    "duration": 78.802,
    "passedTests": 4,
    "failedTests": 0,
    "skippedTests": 0,
    "blockedTests": 0,
    "totalTests": 4,
    "startTime": "2020-06-30T18:12:31Z",
    "finishTime": "2020-06-30T18:12:31Z",
    "passingPercent": 100,

    // Use Artifact OR Package OR Build + Stage + PipelineName Attributes
    "packages": [{"name": "CorpSite-pkg1"}],
    "artifacts": [{"name": "CorpSite-artifact", "version": "1.0.0"}],
    "buildNumber": "55",
    "stageName": "test",
    "pipelineName": "CorpSite-selenium",
}
```

Performance:

```json
{
    "name": "Performance Tests",
    "url": "http://abc.com",
    "startTime": "2020-06-30T18:12:31Z",
    "finishTime": "2020-06-30T18:12:31Z",
    "duration": 78.802,
    "maximumVirtualUsers": "",
    "throughput": "",
    "maximumTime": "",
    "minimumTime": "",
    "averageTime": "",
    "ninetyPercent": "",
    "standardDeviation": "",

    // Use Artifact OR Package OR Build + Stage + PipelineName Attributes
    "packages": [{"name": "CorpSite-pkg1"}],
    "artifacts": [{"name": "CorpSite-artifact", "version": "1.0.0"}],
}
```
Expected standard JSON Notification capability payload - Artifact tool

Pipelines:

```json
{
  "artifacts": [
    {
      "name": "acm.jar",
      "version": "1.82",
      "semanticVersion": "1.82.0",
      "repositoryName": "acm-repo"
    }
  ],
  "pipelineName": "testMultiBranch/master",
  "taskExecutionNumber": "82",
  "stageName": "buildmbmaster",
  "branchName": "master"
}
```

Jenkins Freestyle/Maven Project:

```json
{
  "artifacts": [
    {
      "name": "mav1.jar",
      "version": "1.11",
      "semanticVersion": "1.11.0",
      "repositoryName": "mav-repo"
    }
  ],
  "projectName": "maven-test-proj",
  "taskExecutionNumber": "11"
}
```
Connect capability subflow

Starting with version 1.9, connect capability is supported.

<table>
<thead>
<tr>
<th>Item</th>
<th>Expected value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Label: current&lt;br&gt;Type: Reference.DevOps Tool&lt;br&gt;Tool record for which the <strong>Connect</strong> button action is clicked.</td>
</tr>
<tr>
<td>Outputs</td>
<td>• Label: connected&lt;br&gt;Flag indicating the success or failure of the connection made to the target tool (true/false).&lt;br&gt;• Label: errormessage</td>
</tr>
<tr>
<td>See subflow outputs.</td>
<td></td>
</tr>
<tr>
<td>Item</td>
<td>Expected value</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>String message displayed on the form for connection failure. The</td>
<td>variable is an empty string if the connection is successful.</td>
</tr>
<tr>
<td>Connect errors shown on the DevOps tool form:</td>
<td></td>
</tr>
<tr>
<td><strong>Connection failed</strong></td>
<td>Subflow was executed successfully but the connection could not be made.</td>
</tr>
<tr>
<td><strong>Error: Failed to get failure details from the tool specific connect flow</strong></td>
<td>Subflow execution failed for an unknown reason.</td>
</tr>
<tr>
<td><strong>Error updating the tool connect status</strong></td>
<td>The connection_state attribute could not be updated for an unknown reason.</td>
</tr>
</tbody>
</table>

**DevOps Connect capability flow diagram example - Orchestration tool**

**Discover capability subflow**
Starting with version 1.9, discover capability is supported.
<table>
<thead>
<tr>
<th>Item</th>
<th>Expected value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td><img src="image.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td><img src="image.png" alt="Image" /></td>
</tr>
<tr>
<td>Outputs</td>
<td><img src="image.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image.png" alt="Note" /></td>
<td><img src="image.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td><img src="image.png" alt="Image" /></td>
</tr>
</tbody>
</table>

**Note:** Pagination is not supported for the discover action.

See subflow outputs.
<table>
<thead>
<tr>
<th>Item</th>
<th>Expected value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>;</td>
</tr>
</tbody>
</table>

The Discover main flow is triggered during import request record creation. An import request has these states and messages.

<table>
<thead>
<tr>
<th>State</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requested</td>
<td>--</td>
</tr>
<tr>
<td>Processing</td>
<td>--</td>
</tr>
<tr>
<td>Completed</td>
<td>Updated &lt;number&gt; object(s)</td>
</tr>
<tr>
<td></td>
<td>Found &lt;number&gt; objects with invalid toolId</td>
</tr>
<tr>
<td></td>
<td>Found &lt;number&gt; objects failed validation</td>
</tr>
<tr>
<td>Error</td>
<td>• Unable to parse payload string from Subflow</td>
</tr>
<tr>
<td></td>
<td>• discoverpayload is expected to be an array of objects</td>
</tr>
<tr>
<td></td>
<td>• ImportRequest record does not have reference to tool table</td>
</tr>
<tr>
<td>Paused</td>
<td>--</td>
</tr>
<tr>
<td>Canceled</td>
<td>--</td>
</tr>
<tr>
<td>Unmatched</td>
<td>--</td>
</tr>
</tbody>
</table>

**Expected standard JSON Discover capability payload - Planning tool**

```json
[
  {
    "id":"REL1234567",
    "name": "REL NUMBERS",
    "url" : "https://jira.com/vult/browse/REL1234567",
    "nativeId": "1790e6cc-085b-4529-9cb8-47f393182226"
  },
  {
    "id": "TOR67",
    "name": "TOR 67"
  }
]```
Expected standard JSON Discover capability payload - Coding tool

```
[
  {
    "name": "nvm_repo",
    "url": "https://github.com/nvm_repo/",
    "externalCreatedDate": "2019-06-19 00:37:16"
  },
  {
    "name": "golang_util",
    "url": "https://github.com/golang_util/",
    "externalCreatedDate": "2019-06-19 00:37:16"
  }
];
```

Expected standard JSON Discover capability payload - Orchestration tool

```
{
  "orchestrationTasks": [
    {
      "name": "Build_APC_1",
      "url": "https://jenkins.wsf.xyz/job/Build_APC_1",
      "projectName": "Build_APC_1"
    },
    {
      "name": "CI_CD_Jenkins",
      "url": "https://pt1.jenkins.com/job/CI_CD_Jenkins",
      "projectName": "CI_CD_Jenkins"
    },
    {
      "name": "CI_deploy",
      "url": "https://pt2.jenkins.com/job/CI_deploy",
      "projectName": "CI_deploy"
    }
  ],
  "pipelines": [
    {
      "name": "Build_APC_1",
      "url": "https://jenkins.wsf.xyz/job/Build_APC_1",
      "projectName": "Build_APC_1"
    }
  ]
}
```
DevOps Discover capability flow diagram example - Planning tool
Update Import Request Flow Designer action

You can use the Update Import Request action in your Discover subflow to modify the Import Request record state, if desired.

<table>
<thead>
<tr>
<th>Input label</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>Reference.Import Request</td>
</tr>
<tr>
<td>state</td>
<td>(String)</td>
</tr>
<tr>
<td></td>
<td>• requested</td>
</tr>
<tr>
<td></td>
<td>• processing</td>
</tr>
<tr>
<td>details</td>
<td>(String)</td>
</tr>
</tbody>
</table>

Note: Flow execution is stopped (once the return is received from the subflow) when the state is set to processing.

• completed
• error
• paused
• canceled
• unmatched

DevOps reference

Reference additional details relating to DevOps functionality.

Agile product and DevOps

An agile product represents an item under development that organizes themes, epics, and stories into a single context.

An agile product (application model) is mapped to a plan in DevOps that associates work items.

<table>
<thead>
<tr>
<th>Agile table</th>
<th>DevOps table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Model</td>
<td>Plan</td>
</tr>
<tr>
<td></td>
<td>[cmdb_application_product]</td>
</tr>
<tr>
<td>Extends table:</td>
<td></td>
</tr>
</tbody>
</table>
DevOps plan tables are extended from the Base Planning Item [sn_devops_base_planning_item] table.

**Business application and DevOps**

A business application represents software (software models) used to perform a business function. Software is stored in the Software Model [cmdb_software_product_model] table.

A business application can be associated with an app in DevOps.

A DevOps app ties together plans, repositories, and pipelines that are related to the application.

**DevOps modules**

**Insights**

Analyze operational and business reports including change acceleration, system health, and development to gain insight into your DevOps environment.

**Tools**

Create and set up initial configuration connections including planning tools to connect to applications, coding tools to connect to repositories, orchestration tools to connect to tasks, artifact tools to use change acceleration for releases, and test tools.

**Apps & Pipelines**

Manage apps and create pipelines.
Plan
Access your integrated plans, features (mapped from epics), and versions.

Develop
Access your work items (mapped from your planning application stories), and integrated coding tool repositories. Review development activity including branches, commits, tags (tied to commits), commit details, and committers.

Orchestrate
Access your integrated orchestration tasks and task executions. Review pipeline change requests, registered callbacks, and pipeline executions.

Artifact
Access artifacts and packages to view all commits and work items in a change request for a DevOps release.

Test Results
View test summaries and performance test summaries from your integrated test tools.

Test
View build test summaries and test results.

Administration
Access administrative content such as DevOps properties, error log, inbound events, and payloads. You can also access quick start tests.

Integrations
Create and access DevOps tool integration setup.

Use a token to connect to a DevOps tool
Connect to a DevOps tool, such as GitHub or Jenkins, with basic authentication using a token instead of using a user name and password.

Before you begin
Role required: sn_devops.admin

About this task
Connecting using token-based authentication is useful not only to keep from having to enter your user name and password in a third-party tool, but you can also set specific privileges per token. And, if needed, tokens can be easily
revoked, which makes them a more flexible option if the sole purpose is for connecting a tool.

There are three parts to setting up a connection using a token:
• Generate a token in the tool (Jira, GitHub, or Jenkins)
• Set up basic authentication credentials in DevOps
• Use DevOps to connect to the tool

Procedure
1. Generate and copy a token in GitHub for a coding tool connection, or Jenkins for an orchestration tool connection.
   • Log in to GitHub and navigate to Settings > Developer settings.
     a. Select Personal access tokens and click Generate new token.
     b. Enter ServiceNow DevOps Coding Tool for the description and select repo for the scope.
     c. Click Generate Token.
   • Log in to Jenkins and navigate to Configure > API Token and click Add new Token.
     a. Enter ServiceNow Token for the name.
     b. Click Generate.

Note: You must copy the token at this time (for use in the DevOps coding configuration) because you do not have access to the token once the GitHub screen is closed.

2. Create an application connection to DevOps tools using basic authentication.
   a. Navigate to Connections and Credentials > Credentials and create a new record.
   b. Select HTTP(s) Connection and create a new Credential record using Basic Auth Credentials.
   c. Enter a name, corresponding GitHub or Jenkins user name, and the token generated in the tool previously and click Submit.

3. Create a DevOps coding tool connection to connect to GitHub, or an orchestration tool connection to connect to Jenkins.
a. For GitHub, navigate to DevOps > Configure > Coding Tools and create a record.

b. For Jenkins, Navigate to DevOps > Configure > Orchestration Tools and create a record.

c. Enter a name and select the corresponding tool (GitHub or Jenkins) for the Type field.

d. Use the connection Connection alias lookup list to select the basic authentication connection created previously and click Submit.

e. Open the new coding or orchestration tool record from the list and click Connect. The successful connection status message is shown.

Example:
DevOps connections setup using a token
Tracking DevOps notifications

Turn on tracking for a DevOps planning tool app, coding tool repository, or orchestration tool task to receive specific event notifications.

Track DevOps notifications to focus on specific events when you have multiple DevOps items configured for each tool. When tracking is enabled, you can see specific event details, such as code commits, for all events moving forward.

You can track events for these tool types:

- Planning applications
- Coding repositories
- Orchestration tasks

⚠️ Note: To enable the tracking feature, select the Track check box in the individual record (app, repository, or orchestration task). Tracking is automatically enabled after import.

Event payloads

Notifications for a tracked apps, repositories, and orchestration tasks are shown in the Event Payloads > All Payloads Events list.

You can view all payloads, or only unmatched payloads from the navigation menu. The unmatched payloads Events list shows events for tools not matched to a DevOps ID.

Event payload states

<table>
<thead>
<tr>
<th>Event State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ready</td>
<td>The event was received.</td>
</tr>
<tr>
<td>Processing</td>
<td>Tracking is enabled and the event is being processed.</td>
</tr>
</tbody>
</table>
Event payload states (continued)

<table>
<thead>
<tr>
<th>Event State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processed</td>
<td>The tracked event has completed processing.</td>
</tr>
<tr>
<td>Not Connected</td>
<td>The tool for this event is not connected. In this case, the event cannot be processed.</td>
</tr>
<tr>
<td>Ignored</td>
<td>The track feature is not enabled for the tool in the event. In this case, the event is ignored.</td>
</tr>
<tr>
<td>Error</td>
<td>Event processing failed. Processing can fail if, for example, there is an incorrect link for a tool item.</td>
</tr>
</tbody>
</table>

DevOps record deletion cascade

Starting with version 1.21, cascade record deletion is implemented to delete all dependent lower level DevOps records whenever a parent or higher level DevOps entity is deleted.

For example, when a Plan record is deleted, all dependent Work Item, Plan Version, and many-to-many relation (like App to Plan, and Work Item to Plan Version) records are deleted.

DevOps cascade deletion is implemented for these tables.

- Pipeline, Pipeline Execution, Step, Orchestration Task, Task Execution
- Repository, Commit, Branch, Tag
- Plan, Work item, Plan Version
- DevOps Tool, Artifact Repository, Artifact, Test Summary / Performance test summary

Delete action on a DevOps form

A user with the sn_devops.admin role can delete a DevOps record, but only if it meets the defined ACL criteria.
**Note:** To see the **Delete** button on a form, you must have the **sn_devops.admin** role, and the current record must meet the criteria defined in the scripted ACL.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Scripted ACL criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>A Pipeline record can be deleted only if no other pipeline executions from other pipelines are dependent on the artifact versions generated by the pipeline executions of this pipeline.</td>
</tr>
<tr>
<td>Pipeline Execution</td>
<td>A Pipeline Execution record can be deleted only if no other pipeline executions are dependent on the artifact versions generated by this pipeline execution.</td>
</tr>
</tbody>
</table>
| Task Execution        | A Task Execution record can be deleted if ALL of these conditions are met.  
  - There are no step executions referencing it.  
  - There are no downstream task executions referencing it.  
  - There are no pipeline executions dependent on the artifact versions built by this task execution. |
| Step                  | A Step record can be deleted only if there are no orchestration tasks or step executions referencing this step.                                             |
| Orchestration Task    | If the orchestration task has a step associated, it can be deleted only if there are no task executions referencing this orchestration task.  
  If the orchestration task does not have a step associated (example Jenkins freestyle job), it can be deleted only if no other pipeline executions are dependent on the artifact versions |
<table>
<thead>
<tr>
<th>Entity</th>
<th>Scripted ACL criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repository</td>
<td>A Repository record can be deleted only if none of the commits of this repository are associated to the artifact versions (Artifact Version to Commit table) or task executions (Run Commit table). Therefore, before cleaning up the repository, delete the dependent pipeline entities.</td>
</tr>
<tr>
<td>Branch</td>
<td>A Branch record can be deleted only if there are no commits associated to it in the Branch To Commit table.</td>
</tr>
<tr>
<td>Tag</td>
<td>A Tag record cannot be deleted by a sn_devops.admin.</td>
</tr>
<tr>
<td>Commit</td>
<td>A Commit record can be deleted if ALL of these conditions are met.</td>
</tr>
<tr>
<td></td>
<td>• The commit is not associated to the artifact version (Artifact Version to Commit table).</td>
</tr>
<tr>
<td></td>
<td>• The commit is not associated to task executions (Run Commit table).</td>
</tr>
<tr>
<td></td>
<td>• The commit is not being referenced by other commits as a revert commit.</td>
</tr>
<tr>
<td>Plan</td>
<td>A Plan record can be deleted only if none of the work items of this plan are associated or referenced by any commits.</td>
</tr>
<tr>
<td>Work Item</td>
<td>A Work Item record can be deleted if ALL of these conditions are met.</td>
</tr>
<tr>
<td>Entity</td>
<td>Scripted ACL criteria</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
|                              | • The work item is not being referenced by another work item as parent.  
|                              | • There are no commits referencing or associated to this work item.                                                                                   |
| Plan Version                 | A Plan Version record can be deleted only if there are no work items associated to it in the Work Item To Plan Version table.                      |
| Test Summary / Performance Test Summary | A Test Summary record can be deleted only if it doesn’t have an associated related record (Artifact version/Package/Task Execution) in the Test Summary Relations table. |
| Artifact                     | An Artifact Record can be deleted only if all the artifact versions belonging to it are deletable.                                                   |
| Artifact Repository          | An Artifact Repository record can be deleted only if all the artifacts belonging to it are deletable.                                             |
| Artifact Version             | An Artifact Version record can be deleted only if it’s built by task execution field is empty.                                                      |
| Artifact Staged Request      | An Artifact Staged Request record can be deleted when it is either an orphaned record, or the state is Processed / Error.                           |
| Package                      | A Package record can be deleted if ALL of these conditions are met.  
|                              | • There are no pipeline executions referenced to it.  
<p>|                              | • The built by task execution value on the Package record is null.                                                                                  |
| DevOps Tool                  | A DevOps Tool record cannot be deleted by a sn_devops.admin.                                                                                       |</p>
<table>
<thead>
<tr>
<th>Entity</th>
<th>Scripted ACL criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build Test Summary</td>
<td>These entities cannot be deleted by a sn_devops.admin.</td>
</tr>
<tr>
<td>Build Test Result</td>
<td></td>
</tr>
<tr>
<td>Commit Details</td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td></td>
</tr>
<tr>
<td>Inbound Event</td>
<td></td>
</tr>
</tbody>
</table>

**DevOps record delete cascade**

Deleting a record in a parent table cascade deletes all the child records in the hierarchy.

<table>
<thead>
<tr>
<th>Parent record being deleted</th>
<th>Cascade deleted child records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>Step</td>
</tr>
<tr>
<td></td>
<td>• Orchestration Task</td>
</tr>
<tr>
<td></td>
<td>Pipeline Execution</td>
</tr>
<tr>
<td></td>
<td>• Step Execution</td>
</tr>
<tr>
<td></td>
<td>• Callback</td>
</tr>
<tr>
<td></td>
<td>• Task Execution</td>
</tr>
<tr>
<td></td>
<td>◦ Package</td>
</tr>
<tr>
<td></td>
<td>◦ Run Commit</td>
</tr>
<tr>
<td></td>
<td>◦ Test Summary Relations</td>
</tr>
<tr>
<td></td>
<td>◦ Build Test Summary</td>
</tr>
<tr>
<td></td>
<td>▪ Build Test Result</td>
</tr>
<tr>
<td></td>
<td>◦ Artifact Version</td>
</tr>
<tr>
<td></td>
<td>▪ Artifact Staged Request</td>
</tr>
<tr>
<td></td>
<td>▪ Artifact Version To Commit</td>
</tr>
<tr>
<td></td>
<td>▪ Artifact Version To Package</td>
</tr>
<tr>
<td>Pipeline Execution</td>
<td>Step Execution</td>
</tr>
<tr>
<td></td>
<td>Callback</td>
</tr>
<tr>
<td></td>
<td>Task Execution</td>
</tr>
<tr>
<td>Parent record being deleted</td>
<td>Cascade deleted child records</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td></td>
<td>• Package</td>
</tr>
<tr>
<td></td>
<td>• Run Commit</td>
</tr>
<tr>
<td></td>
<td>• Test Summary Relations</td>
</tr>
<tr>
<td></td>
<td>• Build Test Summary</td>
</tr>
<tr>
<td></td>
<td>◦ Build Test Result</td>
</tr>
<tr>
<td></td>
<td>• Artifact Version</td>
</tr>
<tr>
<td></td>
<td>◦ Artifact Staged Request</td>
</tr>
<tr>
<td></td>
<td>◦ Artifact Version To Commit</td>
</tr>
<tr>
<td></td>
<td>◦ Artifact Version To Package</td>
</tr>
<tr>
<td>Step</td>
<td>None.</td>
</tr>
<tr>
<td></td>
<td>A Step record can be deleted only if there are no Orchestration Task or Step Execution records associated to it.</td>
</tr>
<tr>
<td>Orchestration Task</td>
<td>Task Execution</td>
</tr>
<tr>
<td></td>
<td>• Package</td>
</tr>
<tr>
<td></td>
<td>• Run Commit</td>
</tr>
<tr>
<td></td>
<td>• Test Summary Relations</td>
</tr>
<tr>
<td></td>
<td>• Build Test Summary</td>
</tr>
<tr>
<td></td>
<td>◦ Build Test Result</td>
</tr>
<tr>
<td></td>
<td>• Artifact Version</td>
</tr>
<tr>
<td></td>
<td>◦ Artifact Staged Request</td>
</tr>
<tr>
<td></td>
<td>◦ Artifact Version to Commit</td>
</tr>
<tr>
<td></td>
<td>◦ Artifact Version to Package</td>
</tr>
<tr>
<td>Repository</td>
<td>Tag</td>
</tr>
<tr>
<td></td>
<td>Branch</td>
</tr>
<tr>
<td></td>
<td>• Branch to Commit</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
</tr>
<tr>
<td>Parent record being deleted</td>
<td>Cascade deleted child records</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td></td>
<td>• Commit Details</td>
</tr>
<tr>
<td></td>
<td>• Branch to Commit</td>
</tr>
<tr>
<td></td>
<td>• Tag</td>
</tr>
<tr>
<td>Branch</td>
<td>Branch to Commit</td>
</tr>
<tr>
<td>Tag</td>
<td>No dependent child records.</td>
</tr>
<tr>
<td>Commit</td>
<td>Commit Details</td>
</tr>
<tr>
<td></td>
<td>Branch to Commit</td>
</tr>
<tr>
<td></td>
<td>Tag</td>
</tr>
<tr>
<td>Plan</td>
<td>Work Item</td>
</tr>
<tr>
<td></td>
<td>App to Plan</td>
</tr>
<tr>
<td></td>
<td>Plan Version</td>
</tr>
<tr>
<td></td>
<td>• Work Item to Plan Version</td>
</tr>
<tr>
<td>Work Item</td>
<td>Work Item to Plan Version</td>
</tr>
<tr>
<td>Plan Version</td>
<td>Work Item to Plan Version</td>
</tr>
<tr>
<td>Test Summary / Performance Test Summary</td>
<td>Test Summary Relations</td>
</tr>
<tr>
<td>Artifact</td>
<td>Artifact Version</td>
</tr>
<tr>
<td></td>
<td>• Artifact Staged Request</td>
</tr>
<tr>
<td></td>
<td>• Artifact Version to Commit</td>
</tr>
<tr>
<td></td>
<td>• Artifact Version to Package</td>
</tr>
<tr>
<td>Artifact Repository</td>
<td>Artifact</td>
</tr>
<tr>
<td></td>
<td>• Artifact Version</td>
</tr>
<tr>
<td></td>
<td>◦ Artifact Staged Request</td>
</tr>
<tr>
<td></td>
<td>◦ Artifact Version to Commit</td>
</tr>
<tr>
<td></td>
<td>◦ Artifact Version to Package</td>
</tr>
<tr>
<td>DevOps Tool</td>
<td>Event</td>
</tr>
<tr>
<td></td>
<td>Inbound Event</td>
</tr>
<tr>
<td></td>
<td>Test Summary / Performance Test Summary</td>
</tr>
</tbody>
</table>
Parent record being deleted | Cascade deleted child records
--- | ---

- Test Summary Relations

Plan
- Work Item
- App to Plan
- Plan Version
  - Work Item to Plan Version

Artifact Repository
- Artifact
  - Artifact Version
    - Artifact Staged Request
    - Artifact Version to Commit
    - Artifact Version to Package

Repository
- Tag
- Branch
  - Branch to Commit
- Commit
  - Commit Details
  - Branch to Commit
  - Tag

Pipeline
- Step
  - Orchestration Task
- Pipeline Execution
  - Step Execution
  - Callback
  - Task Execution
<table>
<thead>
<tr>
<th>Parent record being deleted</th>
<th>Cascade deleted child records</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>▪ Package</td>
</tr>
<tr>
<td></td>
<td>▪ Run Commit</td>
</tr>
<tr>
<td></td>
<td>▪ Test Summary Relations</td>
</tr>
<tr>
<td></td>
<td>▪ Build Test Summary</td>
</tr>
<tr>
<td></td>
<td>▪ Build Test Result</td>
</tr>
<tr>
<td></td>
<td>▪ Artifact Version</td>
</tr>
<tr>
<td></td>
<td>▪ Artifact Staged Request</td>
</tr>
<tr>
<td></td>
<td>▪ Artifact Version To Commit</td>
</tr>
<tr>
<td></td>
<td>▪ Artifact Version To Package</td>
</tr>
</tbody>
</table>

**DevOps record delete cascade exceptions**

These records are always deleted in the foreground.

<table>
<thead>
<tr>
<th>Parent record being deleted</th>
<th>Cascade deleted child records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artifact Version</td>
<td>These records are deleted along with Artifact Version.</td>
</tr>
<tr>
<td></td>
<td>▪ Artifact Staged Request</td>
</tr>
</tbody>
</table>

**Note:** For existing releases upgraded to version 1.21, Artifact Staged Request records are not deleted, and the Artifact Version reference is set to empty.

<table>
<thead>
<tr>
<th>Parent record being deleted</th>
<th>Cascade deleted child records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build Test Summary</td>
<td>These records are deleted along with Build Test Summary.</td>
</tr>
<tr>
<td></td>
<td>▪ Artifact Version To Commit</td>
</tr>
<tr>
<td></td>
<td>▪ Artifact Version to Package</td>
</tr>
<tr>
<td>Parent record being deleted</td>
<td>Cascade deleted child records</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td></td>
<td>• Build Test Result</td>
</tr>
<tr>
<td></td>
<td>• Note: For existing releases upgraded to version 1.21, you must delete any dependent Build Test Result records before deleting the Build Test Summary record.</td>
</tr>
<tr>
<td>Package</td>
<td>These records are deleted along with Package.</td>
</tr>
<tr>
<td></td>
<td>• Artifact Staged Request</td>
</tr>
<tr>
<td></td>
<td>• Note: For existing releases upgraded to version 1.21, Artifact Staged Request records are not deleted, and the Package reference is set to empty.</td>
</tr>
<tr>
<td></td>
<td>• Artifact Version to Package</td>
</tr>
</tbody>
</table>

**Foreground deletion threshold property**

Deletion of DevOps records occurs in the foreground (synchronously) by default. Meaning, other UI actions cannot be performed during synchronous deletion. Since the deletion of a parent record can result in the deletion of multiple child records, you can set a threshold value so the remaining records are deleted in the background.

⚠️ **Note:** Artifact Version, Build Test Summary, and Package records are always deleted in the foreground.

To set the DevOps **Cascade delete threshold** property, navigate to **DevOps > Administration > Properties**. Define the total number of records that can be cascade deleted in the foreground, after which the remaining records are deleted in the background. Default is 1000.

⚠️ **Note:** The records of tables in a hierarchy are deleted in a bottom-up manner. For example, it may be possible that even after triggering the delete action on a Repository record, it will still be available for read, write, and update in the system until deletion is complete.

DevOps record deletions do not trigger any business rules or workflows.
DevOps record deletion UI

Cascade delete of a DevOps record triggers multiple confirmation approvals.

Data archiving and table cleanup of inbound events data

Use table cleaners on the inbound events table to purge event data records beyond a specified period. Configure a retention policy with table rotation on the auto-flush form to manage table size growth and archiving data beyond the specified duration.
Overview
Starting from DevOps v1.25, a new data management feature allows you to enable archiving and purging of inbound event data. Database rotation and table cleanup configurations are enabled to ensure that eight weeks of event data is retained. Instance performance is preserved using Now Platform features such as Database rotation, table rotation, and table cleanup. Starting with DevOps version 1.26, a scheduled job is auto-enabled two weeks after you upgrade to DevOps v1.26.

Table rotation on processed inbound events table
A new Devops processed inbound events [sn_devops_processed_inbound_event_list.do] table is added in the DevOps v1.25 release. The processed events table contains copies of event data from the DevOps inbound events [sn_devops_inbound_list.do] table, that is in state 'processed' and 'ignored'.
The processed inbound events table has the table rotation feature enabled. In all, there are eight preconfigured shards, each containing a week's event data from the DevOps inbound event table. By default, once you have 56 days (7 days (Duration) *8 shards (Rotations) of data in the processed inbound table, table rotation is activated. For more information on applying table rotation, see Apply table rotation.

Tip: To modify table rotation configuration, navigate to System Definitions > Table Rotation and open the record for DevOps processed inbound events table.

Scheduled jobs and Table cleaners

Important: Starting with DevOps version 1.26, a scheduled job is auto-enabled two weeks after you upgrade. The scheduled job will run every seven days and is deactivated after the last update.

• Scheduled jobs activate tables cleaners on the inbound events if either of the following scenarios are met:
  ◦ A week after you upgrade to version 1.26, and the processed inbound table has records older than seven weeks.
  ◦ Two weeks after you upgrade to version 1.26.

The table cleanup feature for the inbound events table [sn_devops_inbound_table] has two cleaners that are part of the base system upgrade.
1. Table cleaners are activated and run on the inbound events 
   [sn_devops_inbound_table_list.do] for events in the following states.
   • Processed
   • Ignored
   based on the value specified in the Age in seconds field against the value in sys_created_on field.

2. Table cleaners are activated and run on the inbound events table 
   [sn_devops_inbound_table_list] for events in the following states.
   • New
   • Retry
   • In-progress
   based on the value specified in the Age in seconds field against the value in sys_created_on field.

Note: By default, table cleaners are run on the inbound events table for 
these states, 56 days after you upgrade to v1.26, or from the time the 
schedules jobs are activated. The default value in the Age in Seconds field 
is 4,838,400 seconds (56*24*60*60).

Auto-updates to table cleaner frequency
Events in states processed and ignored are backed up and copied to the 
processed inbound events [sn_devops_processed_inbound_event_list] table. By 
default, the process cleaners are auto-updated to an interval of 30 minutes or 
1800 seconds, on the inbound events table, if either of the following conditions 
are met:
• 56 days have passed since you’ve upgraded to v1.26.
• Two weeks have passed since the scheduled job is active and the processed 
inbound event table has event data older than 7 weeks.

Retry processing inbound events in Error state
Run base system or modified scheduled jobs to retry processing errored inbound 
events. Specify the exceptions or errors to retry events that are in an Error state.

Before you begin
Role required: sn_devops.admin

About this task
Starting from version 1.28, a scheduled job is introduced to retry processing 
inbound events that are in Error state. You can choose to use the scheduled job 
with it's base system properties or customize the properties to suit your needs.
• Run a scheduled job on inbound events that are in error state.
• Update the list of exceptions and errors so that you can retry processing errored inbound events with those exceptions.
• Specify the count for maximum attempts the retry jobs runs on an inbound event.
• Specify the time elapsed since the last retry job. The retry job runs again on the errored inbound events after the duration you specify.

Based on your error or exception, perform any of the following steps to retry processing the inbound events.

Procedure

1. Modify the **Retry Errored Inbound Events** schedule job frequency.

   a. Navigate to **System Definition > Scheduled Jobs**.

   b. Search and open the **Retry Errored Inbound Events** record.

   ![Note: By default, the scheduled job runs repeat every two minutes. You can modify the frequency (in minutes), as required.](image)

   c. **Optional:** Select the frequency to run the job, from the **Run** field, and configure the corresponding fields.

   d. Click **Update**.

   You have modified the frequency of the scheduled job.

2. Modify properties related to retry processing inbound events.

   a. Navigate to **DevOps > Administration > Properties**.

   b. **Optional:** Modify the **Maximum number of retries (for errored inbound events)** property.

   ![Note: The default value for this property is 3. You can modify the count as needed.](image)

   Processing inbound events are retried based on the count you specify.

   c. **Optional:** Modify the **Time elapsed (in minutes). The job will check errored inbound events** property.
Note: The default value for this property is 1440 minutes (24 hours). You can modify the count if required.

Based on the value in this property field, the system retries processing inbound events. For example, if the last attempt to process the inbound events was on 12 noon on 1 January, by default, the scheduled job retries processing errored events after 1440 minutes or 1 day from the timestamp of the last attempt to process the inbound event.

d. Optional: To update the exceptions list based on which errored events are retried for processing, modify the **Errors or exceptions (comma separated) for which errored events are set to Retry** property.

Note: The default values of this property are **TimeOutExceptions, FlowObjectAPIException**. You can update the exceptions you want to retry processing in a comma-separated format.

The exceptions you specify in the **Errors or exceptions (comma separated) for which errored events are set to Retry** property are checked for the **Processing Details** field in the inbound event's record and retries processing.

Results
The scheduled job to retry processing inbound events runs based on either the properties you have configured or the base system properties that are specified by default.

Quick start tests for DevOps
Starting with version 1.10, validate that DevOps still works after you make any configuration change such as apply an upgrade or develop an application in version 1.10, 1.11, and 1.12. Copy and customize these quick start tests to pass when using your instance-specific data.

DevOps quick start tests are available when you install the DevOps application from ServiceNow Store.

DevOps test suite

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
<th>Release version</th>
</tr>
</thead>
</table>
DevOps test suite (continued)

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
<th>Release version</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevOps Orchestration Flow with CR</td>
<td>Verify the Flow Designer flow for a DevOps orchestration tool that includes a change request.</td>
<td>Madrid</td>
</tr>
<tr>
<td>DevOps Orchestration Tool Flow</td>
<td>Verify the Flow Designer flow for a DevOps orchestration tool.</td>
<td>Madrid</td>
</tr>
<tr>
<td>DevOps Plan Tool Flow</td>
<td>Verify the Flow Designer flow for a DevOps planning tool.</td>
<td>Madrid</td>
</tr>
</tbody>
</table>

Related information

Quick start tests

DevOps troubleshooting

Troubleshooting actions can help resolve common issues when setting up or running the DevOps application.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import request not progressing</td>
<td>If an import request remains in the Requested state for too long while performing an import for a tool (such as Jenkins, Jira, or GitHub), delete the import request and try again.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You must delete the existing request to retry importing the same range.</td>
</tr>
<tr>
<td>Tool connection fails</td>
<td>Remove the trailing slash ('/') in the <strong>Connection URL</strong> field on the HTTP Connection form.</td>
</tr>
<tr>
<td>No change request is created for a Jenkins job under change control</td>
<td>Verify that:</td>
</tr>
<tr>
<td>Issue</td>
<td>Action</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>The tool integration in your instance is set up properly.</td>
<td>• The tool integration in your instance is set up properly.</td>
</tr>
<tr>
<td>The task has been synced in your instance.</td>
<td>• The task has been synced in your instance.</td>
</tr>
<tr>
<td>Tasks and app steps have been configured in your instance.</td>
<td>• Tasks and app steps have been configured in your instance.</td>
</tr>
<tr>
<td>Change request creation is not supported if the task is under change control:</td>
<td>Change request creation is not supported if the task is under change control:</td>
</tr>
<tr>
<td>• Is not part of a pipeline (is a standalone task, for example).</td>
<td>• Is not part of a pipeline (is a standalone task, for example).</td>
</tr>
<tr>
<td>• Is the first in the pipeline.</td>
<td>• Is the first in the pipeline.</td>
</tr>
<tr>
<td>• Is within the pipeline, but the user manually triggers, or does SCM checkout directly on the task under change control (thus not triggering the pipeline from the beginning).</td>
<td>• Is within the pipeline, but the user manually triggers, or does SCM checkout directly on the task under change control (thus not triggering the pipeline from the beginning).</td>
</tr>
</tbody>
</table>

**Jenkins does not block the job under change control (does not wait for change request approval)**

Verify that the Jenkins location is configured:  
Navigate to **Jenkins > Manage Jenkins > Configure System** and provide the hostname for the **Jenkins URL** field in the Jenkins Location section.

**Note:** To avoid caching issues, click **Save** even if the Jenkins URL field already contains a value when you first open the form.

**Events occurring in the payload log with state Not Connected**

If any of the following changes for a connection made manually (using manual configuration mode), the connection is automatically disconnected.

• Alias associated with the tool  
• Type of tool
<table>
<thead>
<tr>
<th>Issue</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>- New active HTTP connection for the same domain added to the alias</td>
<td>- Existing HTTP connection for the same domain activated</td>
</tr>
<tr>
<td>- Connection URL of the HTTP connection</td>
<td>- Credentials of the HTTP connection</td>
</tr>
<tr>
<td>- Use MID Server setting in the HTTP connection</td>
<td>Enter manual configuration mode and reconnect.</td>
</tr>
</tbody>
</table>

**Retry Inbound events that fail or error out due to REST API TimeoutException/FlowObjectAPIException**

Update the **RetryErrored Inbound Events** scheduled job to retry processing inbound events that are in **Error** state.

- Update the errors or exceptions list to specify exceptions that you want to retry event processing for.
- Modify the default **Maximum Retry** count.

For more information, see **Retry processing inbound events in Error state**

**Pipeline execution in ServiceNow DevOps does not move forward and waits indefinitely as the SonarQube scans do not take place due to the absence of Sonar tool.**

The software quality inbound event displays the following error message in the processing details field. "Check if the respective sonar tool is created successfully. If not create the sonar tool and retry the inbound event."

For all SonarQube steps in code quality scans, the user must create Sonar tool in the ServiceNowDevOps instance.

For more information, see **SonarQube integration with DevOps**
**Properties installed with DevOps**

Use these properties to configure settings in the DevOps application.

Role required: sn_devops.admin

These properties apply to the DevOps application (DevOps > Administration > Properties).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>[sn_devops.github.api_url]</td>
<td>GitHub API URL.</td>
<td><a href="https://api.github.com">https://api.github.com</a></td>
</tr>
<tr>
<td>[sn_devops.import.max.retries.per_page]</td>
<td>Maximum retries per page while importing.</td>
<td>3</td>
</tr>
<tr>
<td>[sn_devops.committer.score.default]</td>
<td>Default committer score.</td>
<td>50</td>
</tr>
<tr>
<td>[sn_devops.committer.score.multiply.factor]</td>
<td>Multiply factor for committer score.</td>
<td>1</td>
</tr>
<tr>
<td>[sn_devops.enable_debug]</td>
<td>Enable Debug Flag.</td>
<td>Yes</td>
</tr>
<tr>
<td>[sn_devops.import.max.processing.time.seconds.per_page]</td>
<td>Maximum processing time per page (in seconds), while importing.</td>
<td>600</td>
</tr>
<tr>
<td>[sn_devops.supported_webhook_capabilities]</td>
<td>Supported Webhook Capabilities.</td>
<td>code,plan,orchestration,artifact,test</td>
</tr>
<tr>
<td>[sn_devops.import.coding_tool.commits.per_page]</td>
<td>Coding tool commits per page.</td>
<td>100</td>
</tr>
<tr>
<td>[sn_devops.import.coding_tool.repos.per_page]</td>
<td>Coding tool repositories per page.</td>
<td>100</td>
</tr>
<tr>
<td>[sn_devops.cascade_delete_threshold]</td>
<td>Cascade delete threshold (recommended foreground limit 1000).</td>
<td>1000</td>
</tr>
<tr>
<td>[sn_devops.github.api_version_path]</td>
<td>GitHub API version path.</td>
<td>/api/v3</td>
</tr>
<tr>
<td>[sn_devops.import.orchestration_tool.executions.per_page]</td>
<td>Orchestration tool executions per page.</td>
<td>50</td>
</tr>
<tr>
<td>[sn_devops.github.url]</td>
<td>GitHub URL.</td>
<td><a href="https://github.com">https://github.com</a></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>[sn_devops.import.orchestration_tool.executions.maximum]</td>
<td>Maximum orchestration tool executions while importing.</td>
<td>1000</td>
</tr>
<tr>
<td>[sn_devops.import.save.payloads.as.attachments]</td>
<td>You can optionally save payloads as attachments on the Import Request Page record, set the <strong>Value</strong> field to true. Anything else is considered false.</td>
<td>false</td>
</tr>
<tr>
<td>[sn_devops.default_test_type]</td>
<td>Default test type</td>
<td>JUnit</td>
</tr>
<tr>
<td>[sn_devops.import.coding_tool.branches.per_page]</td>
<td>Coding tool branches per property.</td>
<td>19</td>
</tr>
<tr>
<td>[sn_devops.import.planning_tool.issues.per_page]</td>
<td>Planning tool issues per page.</td>
<td>100</td>
</tr>
<tr>
<td>[sn_devops.discovered.user.auto.assign.role]</td>
<td>The provided sn_devops role in the value field is automatically added to users who are active DevOps users (for example, making a commit). If a role not in the sn_devops scope is provided, it is not added. Leave empty if no role should be automatically provided.</td>
<td>sn_devops.viewer</td>
</tr>
<tr>
<td>[sn_devops.change_request.implement_state]</td>
<td>DevOps change request implement state. This property is useful for change management customizations. Default is -1 (Implement).</td>
<td>-1</td>
</tr>
<tr>
<td>[sn_devops.change_request.post_implement_state]</td>
<td>DevOps change request post implement state. Default is 0 (Review).</td>
<td>0</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>[sn_devops.change_request.cancel_state]</td>
<td>DevOps change request cancel state. This property is useful for change management customizations.</td>
<td>Default is 4 (Canceled).</td>
</tr>
<tr>
<td>[sn_devops.change_request_handler_subflow]</td>
<td>DevOps Change Request Handler subflow. This property is useful for change management customizations. Default is sn_devops.default_change_handler_subflow.</td>
<td></td>
</tr>
<tr>
<td>[sn_devops.change_request.approved_approval]</td>
<td>DevOps change request approval text. This property is useful for change management customizations. Approved</td>
<td></td>
</tr>
<tr>
<td>[sn_devops.import.planning_tool.itbm.issues.max.limit]</td>
<td>ITBM Agile planning tool issues import max limit. 1000</td>
<td></td>
</tr>
<tr>
<td>[sn_devops.max_retry_count_inbound_event]</td>
<td>Maximum number of retries for errored inbound events. 3</td>
<td></td>
</tr>
<tr>
<td>[sn_devops.inbound_events_retry_error_list]</td>
<td>Errors or exceptions (comma separated) for which errored inbound events are set to Retry. TimeOutException,FlowObjectAPIException</td>
<td></td>
</tr>
<tr>
<td>[sn_devops.sq_ui_category_preferences]</td>
<td>Software quality categories shown by default in the Pipeline UI view. (Comma separated) coverage,lines_of_code,bugs,code_smells,duplications,vulnerabilities</td>
<td></td>
</tr>
<tr>
<td>[sn_devops.tool_capabilities]</td>
<td>Tool Capabilities supported. (Comma separated) code,plan,orchestration,artifact,test</td>
<td></td>
</tr>
<tr>
<td>[sn_devops.custom_change_categorization]</td>
<td>Categorize DevOps change requests on DevOps change field. false</td>
<td>No</td>
</tr>
</tbody>
</table>
Select the check box (Yes) to enable categorizing change requests as DevOps change requests irrespective of the category selected in the change request.

**Components installed with DevOps**

Several types of components are installed with DevOps, including tables, users, roles, and scheduled jobs.

Demo data is available for this application.

**Users installed**

<table>
<thead>
<tr>
<th>User</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>devops.integration.user</td>
<td>DevOps Integration User</td>
</tr>
<tr>
<td>devops.system</td>
<td>DevOps System user</td>
</tr>
</tbody>
</table>

**Roles installed**

<table>
<thead>
<tr>
<th>Role title [name]</th>
<th>Description</th>
<th>Contains roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevOps Administrator [sn_devops.admin]</td>
<td>Sets up and configures the DevOps application.</td>
<td>• sn_devops.manager</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• connection_admin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• action_designer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• credential_admin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• flow_designer</td>
</tr>
<tr>
<td>DevOps integration [sn_devops.integration]</td>
<td>Has inbound access to the tools in your environment to</td>
<td>• None.</td>
</tr>
<tr>
<td>Role title [name]</td>
<td>Description</td>
<td>Contains roles</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>DevOps manager [sn_devops.manager]</td>
<td>Oversees the operation of the DevOps application and monitors performance in your DevOps environment.</td>
<td>• sn_devops.viewer  • cmdb_read</td>
</tr>
<tr>
<td>DevOps viewer [sn_devops.viewer]</td>
<td>Has access to the DevOps application to use in their environment.</td>
<td>None.</td>
</tr>
</tbody>
</table>

### Scheduled jobs installed

<table>
<thead>
<tr>
<th>Scheduled job</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[DevOps] Historical Data Collection</td>
<td>Collects data from DevOps tools on demand.</td>
</tr>
<tr>
<td>[DevOps] Daily Data Collection</td>
<td>Collects data from DevOps tools daily.</td>
</tr>
</tbody>
</table>

### Tables installed

<table>
<thead>
<tr>
<th>Name</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Schema Definition</td>
<td>[sn_devops_api_schema_definition]</td>
</tr>
<tr>
<td>App</td>
<td>[sn_devops_app]</td>
</tr>
<tr>
<td>Artifact</td>
<td>[sn_devops_artifact]</td>
</tr>
<tr>
<td>Artifact Repository</td>
<td>[sn_devops_artifact_repository]</td>
</tr>
<tr>
<td>Artifact Staged Request</td>
<td>[sn_devops_artifact_staging]</td>
</tr>
<tr>
<td>Artifact Version</td>
<td>[sn_devops_artifact_version]</td>
</tr>
<tr>
<td>Base Planning Item</td>
<td>[sn_devops_base_planning_item]</td>
</tr>
<tr>
<td>Branch</td>
<td>[sn_devops_branch]</td>
</tr>
<tr>
<td>Build Test Result</td>
<td>[sn_devops_build_test_result]</td>
</tr>
<tr>
<td>Name</td>
<td>Table</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>Build Test Summary</td>
<td>[sn_devops_build_test_summary]</td>
</tr>
<tr>
<td>Callback</td>
<td>[sn_devops_callback]</td>
</tr>
<tr>
<td>Commit</td>
<td>[sn_devops_commit]</td>
</tr>
<tr>
<td>Committer</td>
<td>[sn_devops_committer]</td>
</tr>
<tr>
<td>Commit Details</td>
<td>[sn_devops_commit_details]</td>
</tr>
<tr>
<td>Contributor Score Change Factor</td>
<td>[sn_devops_contributor_score_chg_factor]</td>
</tr>
<tr>
<td>Environment</td>
<td>[sn_devops_environment]</td>
</tr>
<tr>
<td>Event</td>
<td>[sn_devops_event]</td>
</tr>
<tr>
<td>Event Processor</td>
<td>[sn_devops_event_processor]</td>
</tr>
<tr>
<td>Import Filter</td>
<td>[sn_devops_import_filter]</td>
</tr>
<tr>
<td>Import Request</td>
<td>[sn_devops_import_request]</td>
</tr>
<tr>
<td>Import Request Page</td>
<td>[sn_devops_import_request_page]</td>
</tr>
<tr>
<td>Inbound Event</td>
<td>[sn_devops_inbound_event]</td>
</tr>
<tr>
<td>Integration Capability</td>
<td>[sn_devops_integration_capability]</td>
</tr>
<tr>
<td>Extends table Application File.</td>
<td></td>
</tr>
<tr>
<td>App to Plan</td>
<td>[sn_devops_m2m_app_plan]</td>
</tr>
<tr>
<td>Deployed Artifact to TaskExecution</td>
<td>[sn_devops_m2m_artifact_execution]</td>
</tr>
<tr>
<td>Artifact Version to Commit</td>
<td>[sb_devops_m2m_artifact_version_commit]</td>
</tr>
<tr>
<td>Artifact Version to Package</td>
<td>[sn_devops_m2m_artifact_version_package]</td>
</tr>
<tr>
<td>Branch To Commit</td>
<td>[sn_devops_m2m_branch_commit]</td>
</tr>
<tr>
<td>Run Commit</td>
<td>[sn_devops_m2m_commit_execution]</td>
</tr>
<tr>
<td>Work Item To Plan Version</td>
<td>[sn_devops_m2m_work_item_plan_version]</td>
</tr>
<tr>
<td>Orchestration Task</td>
<td>[sn_devops_orchestration_task]</td>
</tr>
<tr>
<td>Orchestration Task Definition</td>
<td>[sn_devops_orchestration_task_definition]</td>
</tr>
<tr>
<td>Package</td>
<td>[sn_devops_package]</td>
</tr>
<tr>
<td>Extends table Configuration Item.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Table</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Participant</td>
<td>[sn_devops_participant]</td>
</tr>
<tr>
<td>Pipeline</td>
<td>[sn_devops_pipeline]</td>
</tr>
<tr>
<td>Pipeline Execution</td>
<td>[sn_devops_pipeline_execution]</td>
</tr>
<tr>
<td>Plan</td>
<td>[sn_devops_plan]</td>
</tr>
<tr>
<td></td>
<td>Extends table Base Planning Item.</td>
</tr>
<tr>
<td>Plan Version</td>
<td>[sn_devops_plan_version]</td>
</tr>
<tr>
<td></td>
<td>Extends table Base Planning Item.</td>
</tr>
<tr>
<td>Repository</td>
<td>[sn_devops_repository]</td>
</tr>
<tr>
<td>Step</td>
<td>[sn_devops_step]</td>
</tr>
<tr>
<td>Step Execution</td>
<td>[sn_devops_step_execution]</td>
</tr>
<tr>
<td>Tag</td>
<td>[sn_devops_tag]</td>
</tr>
<tr>
<td>Task Execution</td>
<td>[sn_devops_task_execution]</td>
</tr>
<tr>
<td>Test Execution</td>
<td>[sn_devops_test_execution]</td>
</tr>
<tr>
<td>Test Result</td>
<td>[sn_devops_test_result]</td>
</tr>
<tr>
<td>Test Type</td>
<td>[sn_devops_test_type]</td>
</tr>
<tr>
<td>DevOps Tool</td>
<td>[sn_devops_tool]</td>
</tr>
<tr>
<td>Tool Action</td>
<td>[sn_devops_tool_action]</td>
</tr>
<tr>
<td>Tool Capability Mapping</td>
<td>[sn_devops_tool_capability_mapping]</td>
</tr>
<tr>
<td>DevOps Tool Integration</td>
<td>[sn_devops_tool_integration]</td>
</tr>
<tr>
<td></td>
<td>Extends table Application File.</td>
</tr>
<tr>
<td>Tool Type Capability</td>
<td>[sn_devops_tool_type_capability]</td>
</tr>
<tr>
<td>Work Item</td>
<td>[sn_devops_work_item]</td>
</tr>
<tr>
<td></td>
<td>Extends table Base Planning Item.</td>
</tr>
</tbody>
</table>

**Software Quality Results**

Software Quality Results display scan details from SonarQube scans configured on your Jenkins and Azure DevOps pipelines.
Software Quality Results

Starting from the ServiceNow DevOps version 1.27, you can configure Sonar scan results from your Jenkins and Azure DevOps pipelines to be fetched into ServiceNow DevOps. Sonar notifications create inbound events which are then processed by base system subflows associated with all inbound events with Software Quality capability.

After you have configured Sonar scans on your pipelines and configured the corresponding plugins, run the pipelines to fetch software quality scan results into ServiceNow DevOps. You can view the scan results by Scan ID for each SonarQube scan that was part of your build or release pipeline execution steps.

Software Quality Summaries

Software Quality Summaries shows you a summary of the scans that are run on the pipelines with Sonar scans on the DevOps app on your Now Platform instance.

Software Quality Summaries

1. Navigate to Software Quality Results > Software Quality Summaries.
2. Click a Scan ID record to view scan details.
   The following Software Quality Scan Summary Details appear for the Scan ID.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>The Scan record number in the CMDB.</td>
</tr>
<tr>
<td>Initiated By</td>
<td>The user who initiated the scan</td>
</tr>
<tr>
<td>Scan ID</td>
<td>The unique Scan ID</td>
</tr>
<tr>
<td>Project Name</td>
<td>The SonarQube project name or key.</td>
</tr>
<tr>
<td>Last Scanned</td>
<td>Date Time stamp of the last scan</td>
</tr>
<tr>
<td>Scan URL</td>
<td>The SonarQube project URL you configured in the extension task.</td>
</tr>
<tr>
<td>Scanner name</td>
<td>The code quality scanner tool name.</td>
</tr>
<tr>
<td>Domain</td>
<td>The domain name in which the scans are configured in the instance.</td>
</tr>
<tr>
<td>Tool</td>
<td>The name of the Sonar tool you created.</td>
</tr>
</tbody>
</table>
The Software Quality Scan Details related list (for each Scan ID) displays for the following categories with a corresponding value signifying the count value against the scan result category.

Software Quality Categories:

- Coverage
- Bugs
- Reliability Rating
- Code Smells
- Duplications
- Security Rating
- Lines of Code
- Vulnerabilities
- Security hotspots
- Maintainability rating

Software Quality Sub Categories

The Software Quality Scan Details related list (for each Scan ID) displays software quality sub categories which you can configure for results of category 'vulnerability' from the scan results.

Software Quality Sub Categories- Vulnerabilities

Use seeded sub-categories or create new categories to prioritise and list your scan results of category type- Vulnerabilities.

1. Navigate to DevOps > Integrations > Software Quality Sub Categories.

The Software Quality Sub Categories form displays with the following default categories:

- blocker
- critical
- high
- info
- major
- medium
- minor

These base system sub-categories are seeded to your DevOps app, on upgrading to version 1.27.
2. Click the **New** button to create custom sub categories.

3. On the form, fill in the fields.

**Software Quality Sub Category**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>A unique label for the sub category.</td>
</tr>
<tr>
<td>Domain</td>
<td>The domain in which the DevOps app is running.</td>
</tr>
<tr>
<td>Name</td>
<td>A unique name for the sub category.</td>
</tr>
<tr>
<td>Category</td>
<td>Select a category from the seeded base-system categories from the lookup list.</td>
</tr>
</tbody>
</table>

4. Click **Submit**.

You have successfully created a custom sub category.

To view the Software Quality Scan Detail and the Software Quality Category Details related list > Sub category, follow these steps:

- View scan details as part of Task Executions. View details of all the Sonar scans that are part of the task execution mapped to a build or release pipeline execution step.

1. Navigate to **DevOps > Orchestrate > Task Execution** click a relevant Task Execution record.

2. Click the Software Quality Summary related list.

3. Click a relevant Scan ID record.
The Software Quality Scan Summary and Scan Details display. For more information see, Software Quality Scan Summary

- View scan details as part of Change Request. View all the scans that were part of this build/release pipeline in the Software Quality Results > Software Quality Summary related list.

1. Navigate to DevOps > Orchestrate > Pipeline Change Requests
2. Click the Software Quality Summary related list.
3. Click a relevant Scan ID record.

The Software Quality Scan Summary and Scan Details display. For more information see, Software Quality Scan Summary
Index

D
DevOps 149
Jenkins
   API token 118