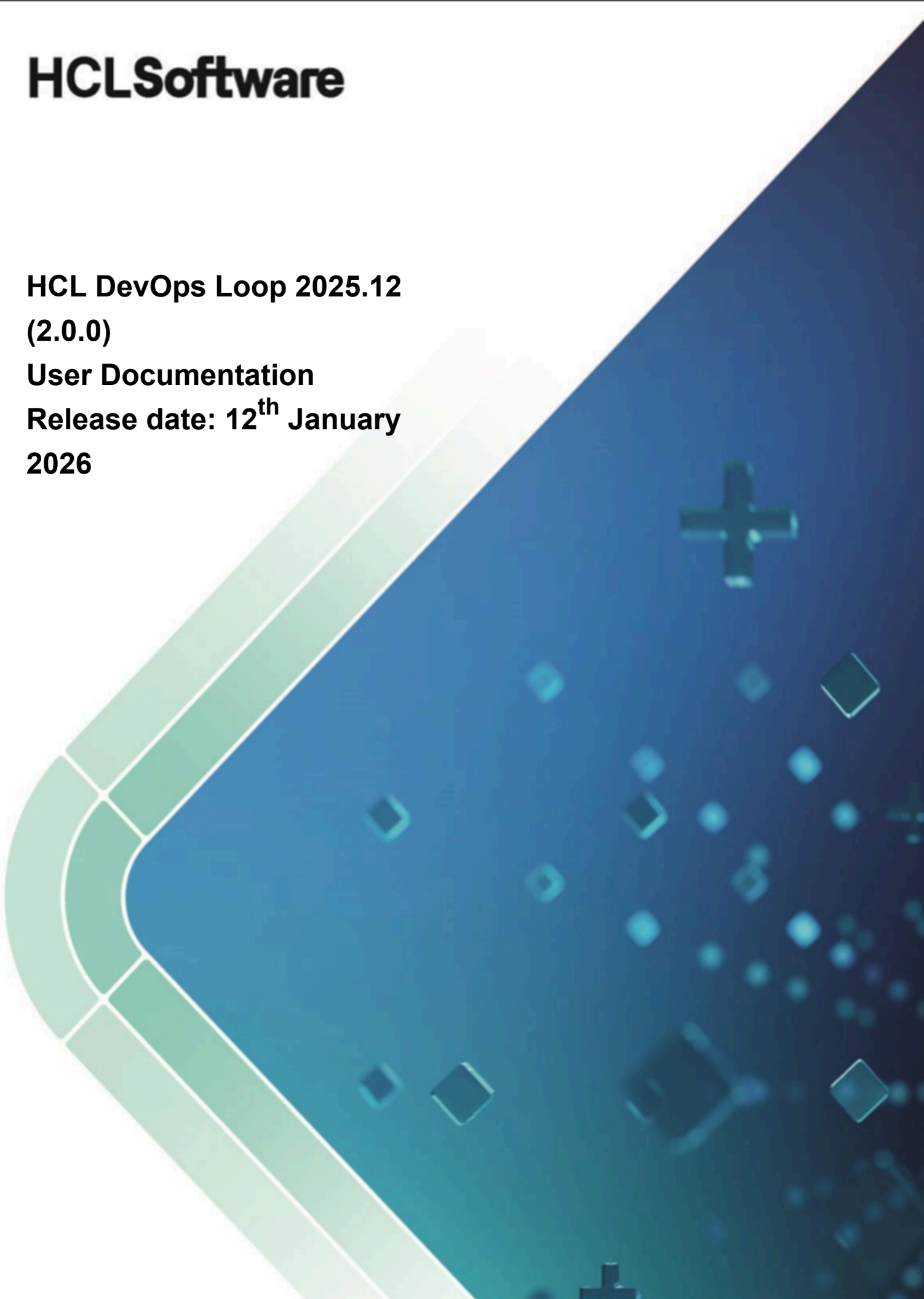
HCLSoftware

# HCL DevOps Loop 2025.12 (2.0.0)

## User Documentation

**Release date: 12th January 2026**



# Special notice

Before using this information and the product it supports, read the information in Notices on page clxi.

# Contents

# Chapter 1. Release notes for DevOps Loop 2025.12 (2.0.0)

This document includes information about the features introduced, a product overview, and installation instructions, as well as contact information of HCL Customer Support for this version of HCL DevOps Loop.

### Product description

DevOps Loop is a cloud-based continuous integration platform. DevOps Loop is built on modern, cloud-native technologies that enables product teams to plan, code, test, build and deploy the applications efficiently. DevOps Loop provides a holistic view of the progress in the DevOps cycle, allowing teams to track development, monitor workflows, and manage releases effectively. For more information, see Overview of DevOps Loop on page 21.

### What's new

You can find information about the features introduced, enhancements, or other changes in DevOps Loop. See What's New on page 5.

### Product download and installation

If you have purchased the licenses to use the product, you can download the entitled software packages from HCL Harbor container registry.

For instructions about installing the product software, see Installation of DevOps Loop on page 30.

### Known issues

For the list of known issues that are identified in this version and the previously published known issues that are still applicable, see Known issues on page 10.

### Contacting HCL support

- For technical assistance, contact HCL Customer Support.
- Before you contact HCL support, you must gather the background information that you might need to describe your problem. When you describe a problem to an HCL Support specialist, be as specific as possible and include all relevant background information so that the specialist can help you solve the problem efficiently. To save time, know the answers to these questions:
    - What software versions were you running when the problem occurred?
    - Do you have logs, traces, or messages that are related to the problem?
    - Can you reproduce the problem? If so, what steps do you take to reproduce it?
    - Is there a workaround for the problem? If so, be prepared to describe the workaround.

# What's New

You can find information about the features introduced, enhancements, or other changes made in HCL DevOps Loop.

## What's New in DevOps Loop 2025.12 (2.0.0)

**Release date: 12th January 2026**

The following table lists the new features, enhancements, or other changes made in DevOps Loop:

| Feature title | Description |
|---|---|
| **GKE platform support** | You can now install DevOps Loop on Google Kubernetes Engine, with full support for Filestore NFS, DNS configuration, load balancers, and Helm-based installation. See Installing DevOps Loop on Google Kubernetes Engine (GKE) on page 43. |
| **Learning Loop enhancements** | Learning Loop now provides an automated CI/CD demonstration flow from code changes to build, deployment, and a live updated webpage served through a new NgineX web server pod with a unique URL for each loop. See Learning Loop overview on page 97. |
| **Support for DevOps Loop MCP server integration** | You can now integrate the DevOps Loop MCP server with external MCP clients such as VS Code and Claude Desktop to retrieve and update loop resources directly from those tools. See Connecting VS Code to the DevOps Loop MCP server and Connecting Claude Desktop to the DevOps Loop MCP server on page 81. |
| **Enhanced Loop Genie with multi-agent workflow engine** | Loop Genie now uses a multi-agent workflow engine that can perform multi-step automation, run complex tasks, and produce well-formatted responses by using models such as **Claude Desktop**, **OpenAI**, and **Gemini**. |
| **Support for a centralized Loop Details page** | You can now view all loop configuration details in one place through the new **Loop Details** page, including linked resources and team members, with options to add additional test projects and repositories. See Loop Genie - Tech Preview on page 154. |
| **AI provider support for Loop Genie** | You can now configure AI providers such as Claude Desktop and Gemini in DevOps Loop to enhance the AI capabilities of Loop Genie. See Configuring Claude Desktop integration on page 70, Configuring Gemini integration on page 71. |
| **UI and accessibility enhancements** | DevOps Loop UI is now improved with consistent color schemes, fonts, pagination, and alignment which provides a uniform and accessible user experience. |
| **Enhanced search capabilities** | The Search feature has been enhanced with new capabilities that make it easier to find and understand information across your loops. You can now search multiple artifact types including issues, builds, tool requests, commits, and deployments. You can also filter results by selecting a loop. See AI-powered search on page 101. |
| **Support for mixed licensing tiers** | As an administrator, you can now assign users to different licensing tiers: Essential, Standard, or Premium with access and feature availability automatically adjusted based on the selected tier.<br><br>Non-admin users receive a tier during onboarding, and admin users automatically receive the highest tier available. See DevOps Loop licensing information on page 52. |

| Feature title | Description |
|---|---|
| **Plan** | |
| **UI enhancements for query links and work item subscriptions** | You can now copy query links to the clipboard and more easily subscribe other users to work items. |
| **Enhancements to Schema Designer** | You can now apply new schema revisions directly to your application, including adding and editing record-type Fields, Actions, and States. |
| **Code** | |
| **Viewing metrics in the running dev containers** | You can now get real-time visibility into key dev container resource metrics to help optimize performance. See Working with dev containers on page 107. |
| **Support for automated SSH authentication by using the Open with DevOps Code UI option** | You can now securely clone a Control Git repository by using the DevOps Code UI with SSH authentication. See Authenticating with DevOps Code UI on page 114. |
| **Support for temporary GitHub Copilot in dev containers** | You can now configure a dev container to support GitHub Copilot for AI-powered code suggestions and chat assistance. See Installing GitHub Copilot in dev containers on page 129. |
| **Build** | |
| **Automatic Build agent configuration** | Build agents are now automatically configured and added to the agent pool during the teamspace initialization. |

You can find information about the features introduced, enhancements, or other changes made in the previous versions of DevOps Loop in the following sections.


- What's New in DevOps Loop 2025.09 (1.0.3) on page 7
- What's new in DevOps Loop 2025.06 (1.0.2) on page 9
- What's new in DevOps Loop 2025.03 (1.0.1) on page 10


## What's New in DevOps Loop 2025.09 (1.0.3)

**Release date: 31st October 2025**

The following table lists the new features, enhancements, or other changes made in DevOps Loop.

| Feature title | Description |
|---|---|
| **Support for installation of DevOps Loop in an air-gapped environment** | You can install DevOps Loop in environments without internet access. See Installation of DevOps Loop in an air-gapped environment on page 47. |

| Feature title | Description |
|---|---|
| **Support for private CA and self-signed certificates** | You can now configure DevOps Loop to trust private or self-signed certificates. See Private CA and self-signed certificate support on page 61. |
| **Support for backup and restoration of DevOps Loop data** | You can now back up and restore DevOps Loop data by using Velero and Minio for disaster recovery and upgrades. See Backup and restoration of DevOps Loop on page 58. |
| **Support for upgrading DevOps Loop** | You can now upgrade DevOps Loop with support for restoring configurations and resources if needed. See Upgrading DevOps Loop on page 64. |
| **Support for Learning Loop with built-in sample data** | You can now use sample data for Build, Control, Deploy, and Plan by enabling Learning Loop for training and experimentation without affecting production data. See Learning Loop overview on page 97. |
| **Support for IBM watsonx integration for Loop Genie** | You can now integrate IBM watsonx in DevOps Loop to enhance the AI capabilities of Loop Genie. See Configuring IBM watsonx integration on page 72. |
| **Enhanced Loop Genie AI assistant - Tech Preview** | You can now use Loop Genie as an intelligent, chat-based assistant to manage teamspace users, loops, and perform control actions such as creating branches or managing pull requests. New features also include voice interaction and improved tracking capabilities. See Loop Genie - Tech Preview on page 154, Prompt references for Loop Genie on page 156. |
| **Support for customizing Dashboards** | You can now edit dashboards, swap panels, clone panels, and add titles directly within DevOps Loop, which provides enhanced customization and visibility. See Dashboards and insights on page 99. |
| **Support for Search across issues** | You can now search for issues such as epics, defects, and stories across loops and teamspaces. Search results include issue metadata, associated URLs, and AI-generated summaries with results filtered by your access permissions. See AI-powered search on page 101. |
| **Support for loop enabling or disabling functionality** | You can now enable or disable loops. Disabling a loop moves it to a **Disabled** tab and stops associated plugins and webhooks, while re-enabling the loop restores all previously disabled functionality. See Disabling a loop on page 95 and Enabling a loop on page 96. |
| **Support for Build integration** | Build is now automatically set up during loop and teamspace creation, which streamlines resource setup and CI/CD steps. |
| **Support for UI and accessibility enhancements** | You can now experience improved UI consistency across DevOps Loop, including updated color schemes, fonts, pagination, and alignment which provides a uniform and accessible user experience. |
| **Support for removing members from Teamspaces and Loops** | You can now remove members from loops, which automatically removes them from all associated capabilties of DevOps Loop. After you remove members from loops, you can |

| Feature title | Description |
|---|---|
| | also remove the members from their teamspaces. See Removing a member from a loop on page 95 and Removing a member from a teamspace on page 85. |
| **Measure** | |
| **Support for integration with Measure to export test execution metrics** | You can now integrate Test Hub with Measure by using a webhook and specify additional parameters related to the test run so that you can view the summary metrics of the test run in the Measure dashboard. The results are categorized based on the type of test – Functional, Unit tests, API, and Performance. |
| **Code** | |
| **Support for launching Dev Containers via URL** | You can now launch Dev Containers directly from a shared URL in DevOps Code, not just from the landing page, enabling quicker access and simplified collaboration. |
| **Support for viewing running Dev Containers** | A new Running Containers tab is available on the landing page to show active Dev Containers in DevOps Code for improved visibility and management. |
| **Display of Dev Container descriptions in UI** | Each Dev Container tile on the landing page now displays its description, helping users understand the container's purpose before launching it. |
| **Inclusion of sample Dev Containers in new teamspaces** | Each newly created teamspace is now prepopulated with a set of sample Dev Containers, enabling users to explore and start development quickly without manual setup. |
| **Support for Git device code authentication** | You can now authenticate Git operations using device code authentication on page 118, ideal for secure or restricted environments. |
| **Support for embedding preinstalled extensions in Dev Container images** | As an admin, you can now embed extensions directly into Dev Container images, ensuring they are preinstalled at launch - ideal for air-gapped environments without access to online extension registries on page 130. |

## What's new in DevOps Loop 2025.06 (1.0.2)

The following section lists the new features, enhancements, or other changes made in DevOps Loop:

| Feature title | Description |
|---|---|
| **Support for teamspace creation** | You can now create teamspaces for your teams in DevOps Loop and the linked teamspaces in all the integrated applications such as DevOps Plan, DevOps Control, DevOps Code, DevOps Test Hub, DevOps Deploy, DevOps Measure, and DevOps Release. See Teamspace management on page 81, Creating a teamspace on page 82, and Adding or inviting members to a teamspace on page 84. |
| **Support for OpenSearch dashboards** | You can now view the visualization of the data that is gathered through the DevOps activities in the integrated applications. See Dashboards and insights on page 99. |

| Feature title | Description |
|---|---|
| **Integration of AI providers - Tech Preview** | You can now integrate AI providers such as OpenAI and Ollama to enable Loop Genie to search and summarize the project data in a loop. See AI provider integration for Loop Genie - Tech Preview on page 65, Configuring OpenAI integration on page 68, Configuring Ollama integration on page 74, and Integration management. |
| **Integration of Loop Genie - Tech Preview** | You can now use the Loop Genie chatbot to answer your queries related to the project in a loop. See Interacting with Loop Genie on page 155. |
| **Revoking licenses** | You can now revoke licenses for users when they are no longer required in a project. See Revoking a license on page 55. |
| **Integration of Build** | You can now configure and use DevOps Build as part of the platform for managing continuous integration and build processes. |
| **Support for custom dev containers** | DevOps Code now supports custom dev containers. A teamspace owner can configure the dev containers accessible to the members. The configuration is based on the Dev Containers specification. You can push specifications of Dev containers to a Control Git repository to version the containers. |
| **Support for a new landing page** | DevOps Code now includes a new landing page. From the landing page, you can select the teamspace that you want to work and choose the branch from where you can fetch the available dev containers. You can launch, open, or terminate Dev containers directly from this landing page. |

### What's new in DevOps Loop 2025.03 (1.0.1)

The following section lists the features, enhancements, or other changes made in DevOps Loop:

| Feature title | Description |
|---|---|
| **Integration of DevOps Code** | You can now use  DevOps Code (Code) as part of the platform. Code is a cloud-based integrated development environment for creating and managing your code. See Code on page 103. |

## Known issues

You can find the known issues that are identified in this version of HCL DevOps Loop.

### Known issues in DevOps Loop 2025.12 (2.0.0)

The known issues are as follows:

| ID | Description |
|---|---|
| NEXUS00004152 | When you receive a clarification question from Loop Genie, the test box might not get enabled, preventing you from entering a response. |

| ID | Description |
|---|---|
| | To work around this problem, you must close and restart Loop Genie. |
| NEXUS00004133 | When you use IBM watsonX, it might not provide proper responses for multistep workflows. The issue is currently observed only with workflows involving multiple steps. Single-step workflows work as expected.<br><br>To work around this problem, you must use IBM watsonX only for single-step workflows until the issue is resolved. |
| NEXUS00004078 | When loop creation fails, and the API returns a message indicating multiple workflows with the same name (for example, `"Multiple workflows found with the same name: Space1~LoopM"`), two or more VSMs might get created with the same name. A duplicate VSM can cause failures in subsequent loop creation attempts.<br><br>To work around this problem, you must perform the following steps:<br><br>1. Go to the **Measure** page and identify the duplicated VSM.<br>2. Delete one of the duplicates, ensuring that only one VSM with that name remains.<br>3. Return to the Loop home page and click the **Retry** button to create the loop again. |
| NEXUS00003988 | When you use Claude Desktop, the application might occasionally encounter connection issues due to a session timeout. Claude Desktop does not currently prompt the user to re-authenticate or refresh the session automatically.<br><br>To work around this problem, you must reconnect the connector to restore the connection. If reconnecting fails, you must delete the existing connector and create a new one, and then restore the connection. |

### Known issues from earlier versions

The known issues identified in the earlier versions of DevOps Loop that are still applicable are as follows:

| ID | Description | Indetified in version | Applicable in and until version |
|---|---|---|---|
| NEXUS00003212 | After a Build server or an agent pod restarts, the previously configured agents are no longer available in Build.<br><br>To work around this problem, you must reconfigure the agents | 1.0.3 | 1.0.3 |

| ID | Description | Indetified in version | Applicable in and until version |
|---|---|---|---|
| | manually from the Build UI after restarting any pod. For more information, refer to Configuring agents. | | |
| NEXUS00003275 | When an administrator is added to a loop by another administrator before logging into Measure for the first time or before creating their own Loop, the administrator cannot log in to Measure.<br><br>To work around this problem, you must log in to Measure as another administrator (for example, the one who created the loop) and assign the global *Product Administrator* role to the administrator who cannot log in. | 1.0.3 | |
| NEXUS00003375 | When you upgrade DevOps Loop from version 1.0.2 to version 1.0.3, teamspace creation might fail at the Build step if the teamspace contains existing build resources with naming conventions that conflict with the resources created during the upgrade.<br><br>To work around this problem, rename the conflicting resources in Build and then click the **Retry** option. | 1.0.3 | 1.0.3 |
| NEXUS00001925 | When a user is invited through the add user process as part of teamspace management, and then the user completes the sign-up process after clicking the invite link received over email, DevOps Loop displays just the confirmation | 1.0.2 | |

| ID | Description | Indetified in version | Applicable in and until version |
|---|---|---|---|
| | message instead of displaying the login link.<br><br>To work around this problem, the user must manually enter the URL to log in to the platform. | | |
| NEXUS00002053 | When you navigate to **Settings > User Administration > Users**, and if you try to filter the user list by using the drop-down lists available on the page, the result might not display correctly. | 1.0.2 | 1.0.3 |
| NEXUS00002082 | When you perform the following steps in DevOps Loop, you might see the error page instead of the DevOps Measure home page:<br><br>1. Create a teamspace.<br>2. Create a loop.<br>3. Navigate to Measure.<br><br>An error page is displayed.<br><br>To work around this problem, you must perform the following tasks:<br><br>1. Configure nginx to increase its buffer size with the following values:<br><br>proxy_buffer_size 32k;<br>proxy_buffers 8 16k;<br><br>**Note:** You can run the following commands to do the configuration:<br><br>`kubectl get pods \| grep velocity-router`<br>`kubectl edit cm nginx-conf` | 1.0.2 | 1.0.3 |

| ID | Description | Indetified in version | Applicable in and until version |
|---|---|---|---|
| |  `kubectl delete pod <velocity-router-id>`<br><br>2. Verify whether the issue is resolved. If not, go to step 3 on page 14.<br>3. Resync the user's access by performing the following steps:<br>a. Log in to DevOps Measure and delete the user from the team that is created through teamspace or loop by navigating to **Settings > Users**.<br>The user can log in to Measure but loses access to teams and resources associated with the teamspace or loop in DevOps Measure.<br>b. In DevOps Loop, add the user again through the teamspace and loop creation process. | | |
| NEXUS00002086 | When a teamspace owner invites an admin user through the invite process as part of the teamspace creation, and when the invited user logs in to DevOps Loop and navigates to DevOps Measure through the switcher, the login page displays instead of the home page of DevOps Measure. | 1.0.2 | 1.0.3 |

| ID | Description | Indetified in version | Applicable in and until version |
|---|---|---|---|
| | To work around this problem, you must perform one of the following tasks:<br><br>• Click **Login with OIDC**.<br>• Clear your cache, cookies, and local browser data.<br><br>If the issue is still not resolved, resync the user's access by performing the following steps:<br><br>1. Log in to DevOps Measure and delete the user from the team that is created through teamspace or loop by navigating to **Settings > Users**.<br><br>The user can to log in to DevOps Measure but loses access to teams and resources associated with the teamspace or loop in DevOps Measure.<br><br>2. In DevOps Loop, add the user again through the teamspace and loop creation process. | | |

# Chapter 2. System Requirements for DevOps Loop 2025.12 (2.0.0)

This document includes information about hardware and software requirements for HCL DevOps Loop.

### Contents



### Disclaimers

This report is subject to the Terms of Use and the following disclaimers:

The information contained in this report is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied, including but not limited to the implied warranties of merchantability, non-infringement, and fitness for a particular purpose. In addition, this information is based on HCL's current product plans and strategy, which are subject to change by HCL without notice. HCL shall not be responsible for any direct, indirect, incidental, consequential, special or other damages arising out of the use of, or otherwise related to, this report or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from HCL or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of HCL software.

References in this report to HCL products, programs, or services do not imply that they will be available in all countries in which HCL operates. Product release dates and/or capabilities referenced in this presentation may change at any time at HCL's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Discrepancies found between reports and other HCL documentation sources may or may not be attributed to different publish and refresh cycles for this tool and other sources. Nothing contained in this report is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth, savings or other results. You assume sole responsibility for any results you obtain or decisions you make as a result of this report.

Notwithstanding the Terms of Use users of this site are permitted to copy and save the reports generated from this tool for such users own internal business purpose. No other use shall be permitted.

# Hardware

You can find information about the hardware requirements for HCL DevOps Loop.

The following table lists the requirements to run DevOps Loop on the Kubernetes Service (K8S) or IBM Cloud Kubernetes Service (IKS) platform.

| **Resource** | **Requirement** | **Notes** |
|---|---|---|
| Number of worker nodes | 3 | If you are using Ollama, worker nodes must have at least 32Gib memory. |
| Processor | 8 CPUs per node | |
| Memory | 64 GiB total<br><br>In addition, you might require 200 MB for each DevOps Code Dev Container used by a user. | |
| Disk space | 256 GiB<br><br>In addition, you might require 20 GB for each DevOps Code Dev Container used by a user. | |
| Network | 1 Gbps | |

Related information

System Requirements for DevOps Loop 2025.12 (2.0.0) on page 16

# Operating systems and containers

You can find details about the supported operating systems and containers for HCL DevOps Loop.

### Contents

### Bit version support

| Bitness | Description |
|---|---|
| 64-Exploit | The product or part of the product runs as a 64-bit application in the 64-bit platforms listed as supported. |

### Operating systems

| Operating system | Hardware | Bitness | Notes |
|---|---|---|---|
| RHEL 8.8 | x86-64 | 64-Exploit | |
| RHEL 8.10 | x86-64 | 64-Exploit | |
| RHEL 9.0 | x86-64 | 64-Exploit | |
| RHEL 9.2 | x86-64 | 64-Exploit | |
| RHEL 9.4 | x86-64 | 64-Exploit | |

### Container Platforms

You can find details about the supported container platforms.

| Platform | Version | Note |
|---|---|---|
| Kubernetes Service (K8S) | 1.32 | The storage class must support ReadWriteOnce (RWO) and ReadWriteMany (RWX). |

Related information

System Requirements for DevOps Loop 2025.12 (2.0.0) on page 16

# Host prerequisites

You can find the prerequisites that support the operating capabilities for HCL DevOps Loop.

### Contents



### Installation

### Licensing

| License Server | Version | Notes |
|---|---|---|
| My HCLSoftware (MHS) | Latest Cloud version | |
| HCL Local License Server | 5.1 | |

### Runtime environment

| Supported software | Version | Notes |
|---|---|---|
| Helm CLI | 3.17 | Required locally for Helm. |

### Web browsers

| Browser | Version |
|---|---|
| Google Chrome | 129 or later |
| Microsoft Edge | 129 or later |
| Mozilla Firefox | 130 or later |


Related information

System Requirements for DevOps Loop 2025.12 (2.0.0) on page 16


# Supported software

You can find information about the supported software in HCL DevOps Loop.

## Supported software in DevOps Loop 2025.12 (2.0.0)

| Supported software | Version |
|---|---|
| HCL DevOps Plan | 3.0.6 |
| HCL DevOps Test Hub | 11.0.7 |
| HCL DevOps Deploy | 8.2.0.0 |
| HCL DevOps Velocity | 5.2.0 |
| HCL DevOps Build | 7.1.0.2 |

Related information

System Requirements for DevOps Loop 2025.12 (2.0.0) on page 16

# Chapter 3. Getting Started

HCL DevOps Loop is a platform designed to support the full DevOps lifecycle. The Getting Started section introduces key concepts and walks through the initial steps to help new users begin working with the platform.

Before exploring the capabilities or performing tasks, ensure that the DevOps Loop is installed and properly configured. See Installation of DevOps Loop on page 30.

## Overview of DevOps Loop

HCL DevOps Loop is a unified platform that brings every stage of the software development lifecycle into a continuous feedback loop. It connects teams, processes, and applications across planning, coding, testing, releasing, deploying, and measuring to deliver software faster, with greater visibility and governance.

DevOps Loop is built on cloud-native technologies that support CI/CD at scale and enhance automation with Loop Genie. This AI-powered assistant that provides search results across applications in the Loop and accelerates workflows and helps accelerate delivery.

With dashboards and analytics, DevOps Loop provides real-time insights into progress, quality, and delays across the toolchain. These insights help teams track value creation, enhance decision-making, and continuously improve delivery.

For security and compliance, DevOps Loop offers role-based access control so administrators can manage permissions and enforce governance across projects.

Enterprises can install DevOps Loop on IBM Red Hat OpenShift, IBM Cloud Kubernetes Service (IKS), and Kubernetes Service (K8S) and Air-Gapped environments to achieve flexibility and scalability across environments.

In addition to its platform-wide features, DevOps Loop includes a suite of integrated capabilities such as **Plan, Code, Control, Build, Test, Release, Deploy, and Measure** which work together to support every stage of the software development lifecycle.

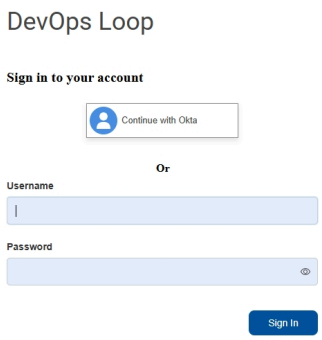
To learn more about the specific capabilities, see Capabilities of DevOps Loop on page 103.

## User interface

You can find an overview of the HCL DevOps Loop user interface.

**Login page**

The login page provides an option to sign in to the DevOps Loop

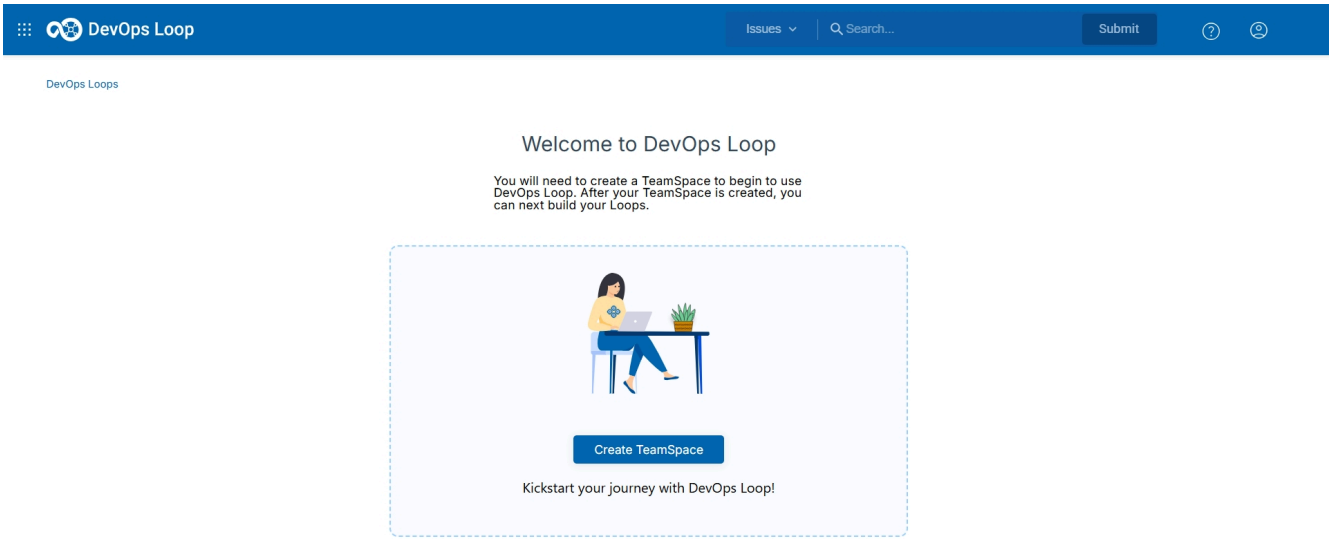




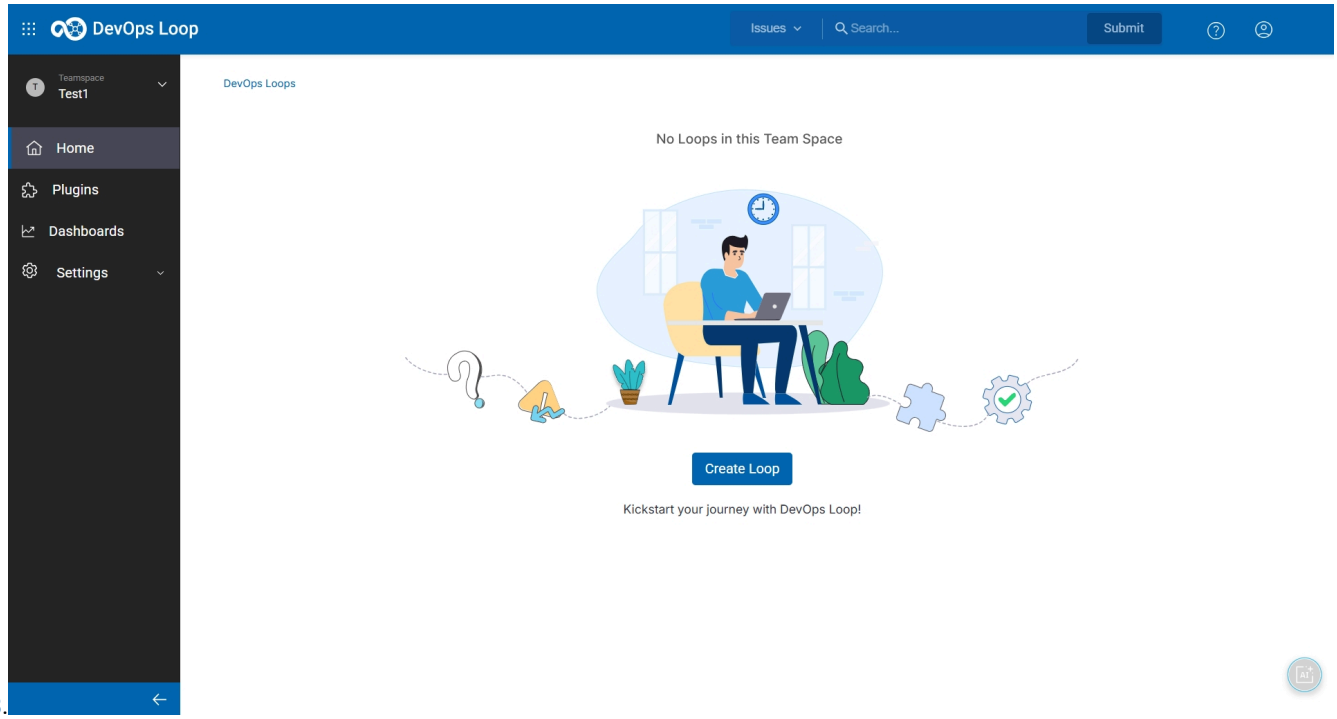
platform.

**Home page**

After logging in, the Home page provides an option to create Teamspace. You must create a Teamspace before start using DevOps Loop. See Teamspace management on page 81.

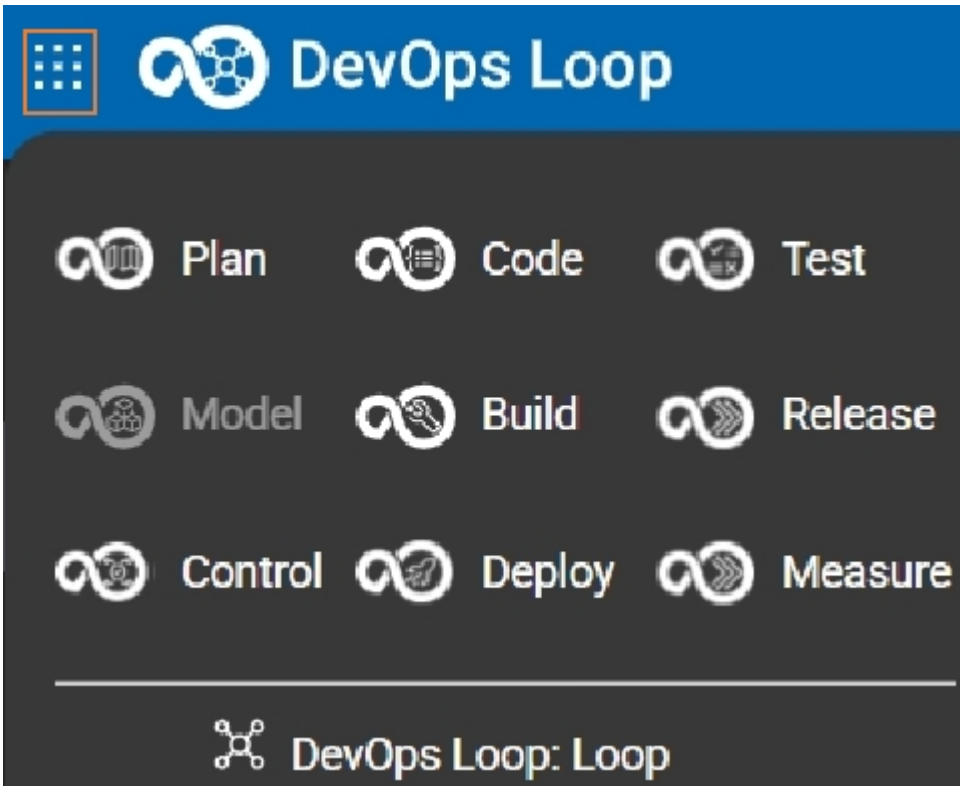


Within your Teamspace, you can create a loop to begin setting up your DevOps workflow. See Loop management on page 88.




**Capability selector**

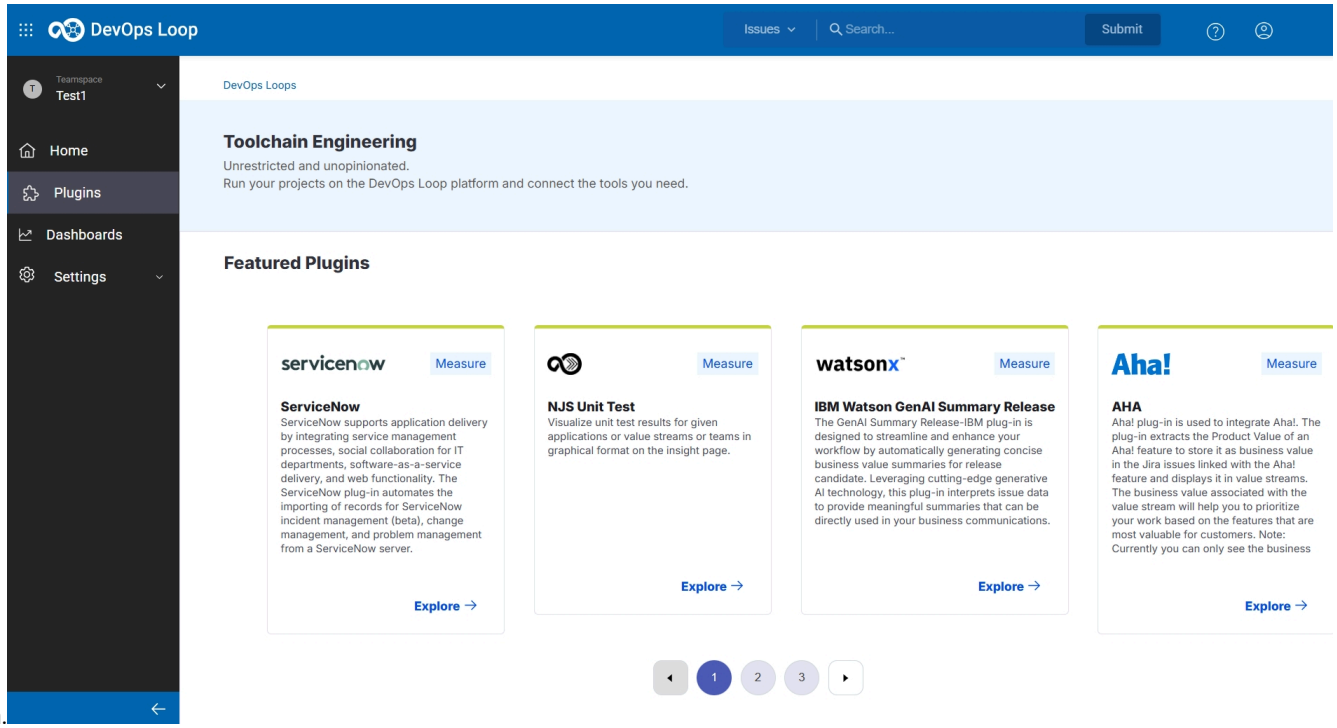
By using the capability selector icon , you can quickly access all DevOps Loop capabilities, such as Plan, Code, Test, Build, Release , Control, Deploy, and Measure from a single menu.




**Plugins**

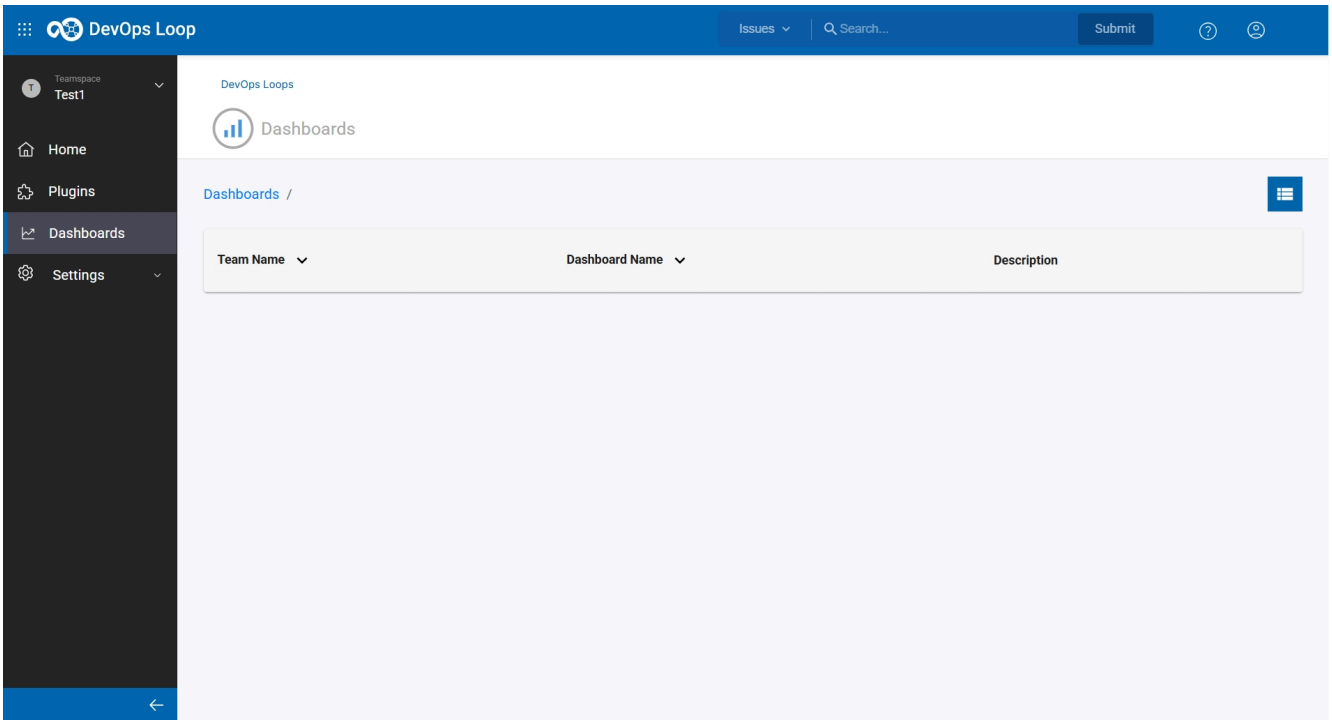
The sidebar provides the  option for you to navigate to the **Toolchain Engineering** page that lists the **Featured plugins** and **All plugins** sections. You can choose a plugin that is

required for a specific solution from the **All plugins** section and click  to download the plugin.




### Dashboards

The Dashboards page provides a centralized view where teams can access, filter, and manage all their DevOps Loop dashboards.



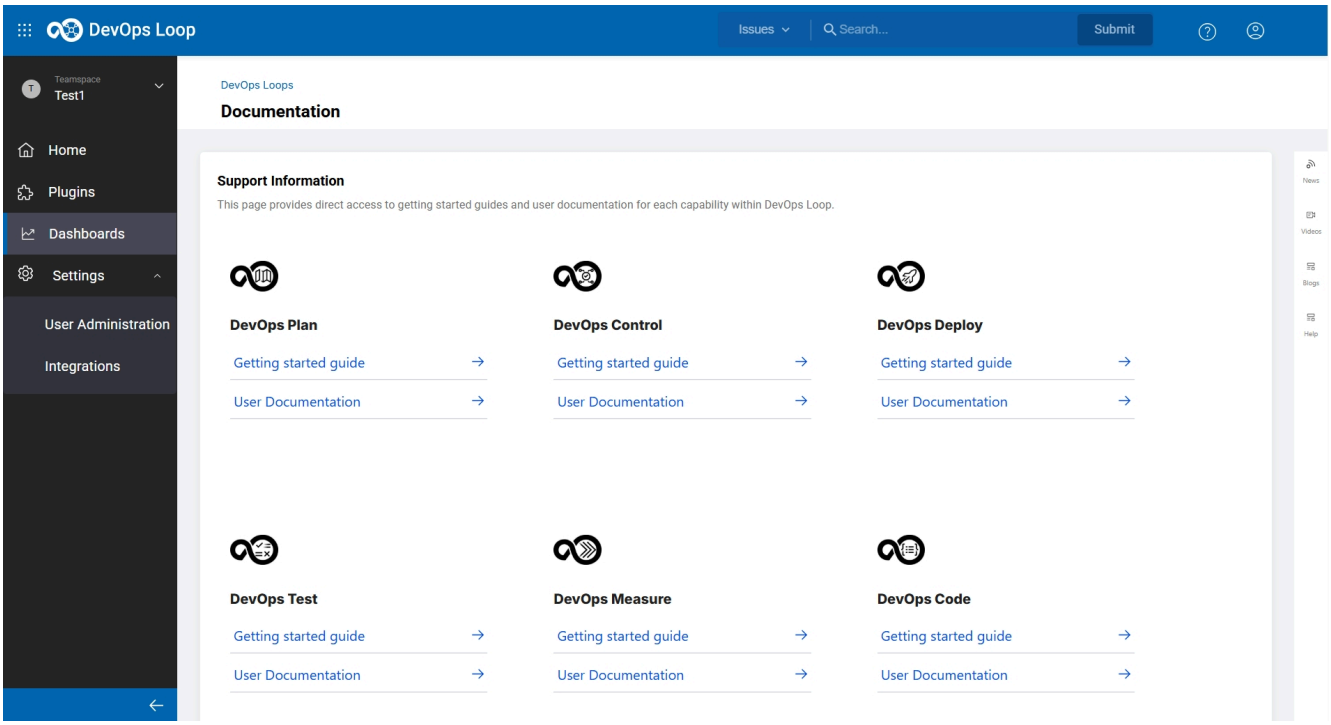

**Settings**

The sidebar provides you the  option to administer users through the **User Administration** page and manage integrations through the **Integrations** page. See Revoking a license on page 55 and AI provider integration for Loop Genie - Tech Preview on page 65.

**Help**

You can click the **Help** icon  to quickly access the product information and documentation resources.

You can select **Documentation** option in the **Help** menu to view getting-started guides and detailed user documentation for every capability within DevOps Loop.




### Profile

The **Profile**  displays the logged-in user's information and provides the option to log out of DevOps Loop.

To log out, you can click the profile icon  , and then click **Logout**. The following logout page is displayed.

You can click **Login** to log in again.

# Accessibility features

Accessibility features help users who have physical disabilities, such as visual and hearing impairment, or limited mobility, to use the software products successfully.

### Accessibility compliance

The product documentation is published by using Oxygen XML WebHelp Responsive. To understand the accessibility compliance status for Oxygen XML WebHelp Responsive, refer to WebHelp Responsive VPAT Accessibility Conformance Report.

### Accessing UI elements

DevOps Loop supports navigation in the UI by using different methods such as a mouse, keyboard, or touchpad.

You can use the keyboard keys such as **Tab**, arrow keys such as **UP**, **DOWN**, **LEFT**, and **RIGHT** to navigate to the different pages in the **Navigation** pane or to the different action labels in the right pane on the UI.

# Chapter 4. Administration

This section provides information on administration, installation, configuration, and integrations and teamspace management.

## Configuring external databases for the capabilities in DevOps Loop

You can configure external databases for the capabilities in DevOps Loop and provide the database URL in the helm chart to connect the capabilities to external database.

### Overview

You can configure external databases using Helm charts for the capabilities in DevOps Loop. Each product may require different settings or secrets, which are defined in Helm chart values and configuration files. This topic summarizes how to configure databases for Deploy, Plan, Build, Measure, Release, and Test capabilities.

### Plan

To configure an external database for Plan. Refer to Helm chart external database configuration.

### Build

Build uses the same MySQL database as Deploy. You must configure the Deploy Helm chart with MySQL. To configure the Build Helm chart to use this external database, you must update the following Helm values:

#### Helm Values for External MySQL

You must use the following values under `HELM_OPTIONS`:

```
--set hcl-devops-build.externalDB.enabled=true \
--set hcl-devops-build.externalDB.hostname=<MYSQL_HOST> \
--set hcl-devops-build.externalDB.port=<MYSQL_PORT> \
--set hcl-devops-build.externalDB.database=<DATABASE_NAME> \
--set hcl-devops-build.externalDB.user=<DATABASE_USER> \
--set hcl-devops-build.secret.name=build-secret
```

You must replace the following placeholders with your actual MySQL configuration:

- `<MYSQL_HOST>` – Hostname or IP of the MySQL server
- `<MYSQL_PORT>` – MySQL port (default: 3306)
- `<DATABASE_NAME>` – Name of the MySQL database (e.g., `devops_build`)
- `<DATABASE_USER>` – MySQL username

#### Setting the database password

You must create a Kubernetes secret that contains the database password by running the following command:

```
BUILD_DB_PASSWORD=<DATABASE_PASSWORD>
kubectl create secret generic build-secret -n devops-loop --from-literal=dbpassword="$BUILD_DB_PASSWORD"
```

“

#☺ **Important:** The `BUILD_DB_PASSWORD` must match the password assigned to the MySQL user.

”

### Deploy

To configure an external database for Deploy, refer to the Prerequisites and Parameters sections of the README file in Helm chart. Helm values related to database configuration begin with `database`.

### Measure and Release

To configure an external database for Measure and Release, create a secret named `mongodb-url-secret` with a property named `password` after installing the MongoDB instance. The value should follow this format:

```
mongodb://<user>:<password>@<host>:<port>/<auth_database>
```

Example command:

```
kubectl create secret generic mongodb-url-secret --namespace devops-automation
 --from-literal=password="${MONGO_URL}"
```

Set the Helm chart value to reference the secret:

```
--set ibm-ucv-prod.secrets.database=mongodb-url-secret
```

For more information, refer to Velocity installation prerequisites.

### Test

Support for configuring external databases in Test will be provided in an upcoming release.

# Installation of DevOps Loop

To get started working with HCL DevOps Loop, you must first install DevOps Loop.

You can install DevOps Loop on the following platforms:

- IBM Cloud Kubernetes Service (IKS)
- Kubernetes Service (K8S)
- Air-Gapped Environment

To learn more about the installation of DevOps Loop on the supported platforms refer to the following topics:


- Installing on IKS by using a newly created cluster on page 31
- Installing on a K8S cluster that has load balancer resources available on page 34
- Installing on a K8S cluster with an upstream L7 load balancer on page 37

- Installing on a K8S cluster that has an upstream L7 load balancer and expects data to be re-encrypted on page 39
- Installation of DevOps Loop in an air-gapped environment on page 47

**Note:** Before you get started with the installation, if you want to configure the external databases in the production environment, see Configuring external databases for the capabilities in DevOps Loop on page 29.

## Installing DevOps Loop on IBM Cloud Kubernetes Service (IKS)

You can find information about the tasks that you can perform to install HCL DevOps Loop on IBM Cloud Kubernetes Service (IKS) by using a newly created cluster. You can use the Helm chart to perform the installation.

**Before you begin**

You must have completed the following tasks:

- Read and understood System Requirements for DevOps Loop 2025.12 (2.0.0) on page 16.
- Installed the following CLI tools:
  - Kubectl
  - IBM Cloud CLI
- Installed Helm on the system from which you access the Kubernetes cluster. For more information, refer to Installing Helm.
- Set up a Kubernetes cluster. For more information, refer to Getting started with IBM Cloud Kubernetes Service.
- Set up the Secrets Manager in your Kubernetes Service cluster. For more information refer to IBM Cloud Secrets Manager.
- Obtained valid public certificates issued by trusted Certificate Authorities (CAs). Also, read and understood about managing certificates and secrets. For more information refer to Managing TLS and non-TLS certificates and secrets.

1. Navigate to **Cluster Management > Clusters > Overview > Actions > Connect via CLI** in your IBM Cloud account and associate the kubectl context with your cluster.
2. Navigate to **Cluster Management > Clusters > Ingress > Ingress Controllers (ALB)** and note down the IP address of the public ingress controller.
3. Navigate to **Cluster Management > Clusters > Ingress > Domain** and note down the region in the domain name of your cluster, which is in the format, `<cluster_name-id>.<region>.containers.appdomain.com`.
4. Click **Create** in the **Domain** tab and perform the following steps:
   a. Provide a domain name in the format, `<custom_name>.<region>.containers.appdomain.com`, by using the region noted down in the previous step.
   b. Provide the IP address of the public ingress controller noted down in step 2 on page 31.
   c. Set the domain as the default domain.

**Note:** If you do not set this as the default domain, you must delete and re-create the domain to set it as the default domain.

5. Make the certificate and key available as a Kubernetes secret in the Kubernetes namespace that you use for DevOps Loop.

   **Note:** If a helm lookup error occurs on a secret, likely the certificate was not imported to the Loop namespace.

   To correct this error, import the certificate into your namespace: `ibmcloud ks ingress secret create --cluster <cluster_name_or_ID> --cert-crn <crn> --name <secret_name> --namespace <namspace>`

   You must use the IBM Secrets Manager to manage the life cycle of the certificate. You must also note down the name of the secret that contains the TLS certificate and key. For more information, refer to https://cloud.ibm.com/docs/containers?topic=containers-secrets&interface=ui#tls-custom.
6. Perform the following steps to install Emissary-ingress in your cluster:
   a. Run the following commands to set the Ambassador Edge Stack Helm chart:

   ```
   helm repo add datawire https://app.getambassador.io
   helm repo update
   ```

   b. Run the following commands to create a namespace and install the Ambassador Edge stack:

   ```
   kubectl create namespace emissary && \
   kubectl apply -f https://app.getambassador.io/yaml/emissary/3.9.1/emissary-crds.yaml
   kubectl wait --timeout=90s --for=condition=available deployment emissary-apiext -n
    emissary-system
   ```

   c. Run the following command to create `emissary-ports.yaml`:

   ```
   cat <<EOF > emissary-ports.yaml
   service:
     ports:
       - name: https
         port: 443
         targetPort: 8443
         #nodePort: <optional>
       - name: http
         port: 80
         targetPort: 8080
         #nodePort: <optional>
       - name: deploy-wss
         port: 7919
         targetPort: 7919
         #nodePort: <optional>
       - name: control-ssh
         port: 9022
         targetPort: 9022
         #nodePort: <optional>
   EOF
   ```

   d. Install Emissary-ingress by running the following command:

```
helm install emissary-ingress --namespace emissary datawire/emissary-ingress -f
 emissary-ports.yaml && \
kubectl -n emissary wait --for condition=available --timeout=90s deploy
 -lapp.kubernetes.io/instance=emissary-ingress
```

7. Note down the domain name for your cluster, which is in the format, `<custom_name>.<region>.containers.appdomain.com`.
   You can navigate to **Containers > Cluster Management > Clusters > Ingress > Domains** in the IBM Cloud console or run the following command to list the domains in your cluster:

   ```
   ibmcloud ks ingress domain ls --cluster <CLUSTER_NAME>
   ```

8. Perform the following steps to access the HCL Harbor container registry:
   a. Get a key to the HCL Harbor container registry.
   b. Log in to HCL Harbor container registry with the HCL ID and password that are associated with the entitled software.
   c. Copy the pre-generated CLI secret from the **User Profile** page.
   d. Create the following three secrets in the target namespace to pull images from the HCL Harbor container registry:

   ```
   kubectl create secret docker-registry hcl-entitlement-key \
       --namespace [namespace_name] \
       --docker-username=<Harbor User ID> \
       --docker-password=<CLI secret> \
       --docker-server=hclcr.io
   ```

   **Note:** Secrets are namespace-specific and they are required to install DevOps Plan.

9. Run the following command to view the `README.md` file:

   ```
   helm show readme oci://hclcr.io/devops-automation-helm/hcl-devops-loop --version 2.0.000
   ```

10. Update the following parameters and the other required parameters in the script in the Helm README with the correct values:
    - DOMAIN
    - TLS_CERT_SECRET_NAME
    - RWO_STORAGE_CLASS=nfs-client
    - RWX_STORAGE_CLASS=nfs-client

    For DOMAIN and TLS_CERT_SECRET_NAME, you must provide the values noted down in the previous steps.

11. Run the script in the Helm README for K8 installation.

    **Note:** If the installation fails due to timeout, run the installation script again.

12. Perform the following steps to enable non-HTTP and additional special services:
    a. Run the following command to display the IP of the L4 load balancer installed as part of DevOps Loop:

    ```
    kubectl get svc --namespace emissary emissary-ingress -o
     jsonpath='{.status.loadBalancer.ingress[0].ip}'
    ```

b. Perform the following steps to create a new domain by using the IP of the L4 load balancer:
   i. Navigate to **Cluster Management > Clusters > Ingress > Domains >** .
   ii. Copy the default domain name.
   iii. Click **Create**.
   iv. Enter `service-<copied_domain_name>` in **Name**.
   v. Enter the IP of the L4 load balancer in **IP address**.
   vi. Click **Create**.

You can also run the following command from CLI:

```
ibmcloud ks ingress domain create --cluster CLUSTER [--crn CRN] [--is-default] [--domain
 DOMAIN] [--hostname HOSTNAME] [--ip IP] [--output OUTPUT] [--domain-provider PROVIDER] [-q]
 [--secret-namespace NAMESPACE] [--zone ZONE]
```

**Note:** You must ensure that the domain created in this step is not the default domain.

**Results**

You have installed DevOps Loop on IKS by using a newly created cluster.

## Installing DevOps Loop on Kubernetes Service (K8S)

You can find information about the tasks that you can perform to install HCL DevOps Loop on Kubernetes Service (K8S). You can use the Helm chart to perform the installation.

**Before you begin**

You must have completed the following tasks:

- Read and understood System Requirements for DevOps Loop 2025.12 (2.0.0) on page 16.
- Installed the Kubernetes CLI tool, Kubectl.
- Installed Helm on the system from which you access the Kubernetes cluster. For more information, refer to Installing Helm.
- Set up a Kubernetes cluster. For more information, refer to Kubernetes Documentation.
- Read and understood administering a cluster and managing TLS certificates in a cluster. For more information, refer to Administer a Cluster and Manage TLS Certificates in a Cluster.
- Set up the cert-manager in your Kubernetes cluster. For more information refer to Kubernetes documentation.

**About this task**

You can select one of the following methods to install DevOps Loop on K8S:

- Installing on a K8S cluster that has load balancer resources available on page 34
- Installing on a K8S cluster with an upstream L7 load balancer on page 37
- Installing on a K8S cluster that has an upstream L7 load balancer and expects data to be re-encrypted on page 39

## Installing on a K8S cluster that has load balancer resources available

**Before you begin**

You must have completed the following tasks:

- Ensured your cluster supports L4 load balancer resources.
- Ensured that an external fully qualified domain name with a certificate signed by a well-known CA or an intermediary is available.

1. Associate the kubectl context with your cluster by using the following commands:

```
kubectl config set-context <context_name> --namespace=<namespace-name> --cluster=<cluster-name>
 --user=<user-name>
kubectl config use-context <context-name>
```

2. Obtain the certificate and key for the domain that you use for DevOps Loop.
3. Make the certificate and key available as a Kubernetes secret in the Kubernetes namespace that you use for DevOps Loop.

   You should use the cert-manager or any standard mechanism to manage the life cycle of the certificate. You must also note down the name of the secret that contains the TLS certificate and key.
4. Perform the following steps to install Emissary-ingress in your cluster:
   a. Run the following commands to set the Ambassador Edge Stack Helm chart:

```
helm repo add datawire https://app.getambassador.io
helm repo update
```

   b. Run the following commands to create a namespace and install the Ambassador Edge stack:

```
kubectl create namespace emissary && \
kubectl apply -f https://app.getambassador.io/yaml/emissary/3.9.1/emissary-crds.yaml
kubectl wait --timeout=90s --for=condition=available deployment emissary-apiext -n
 emissary-system
```

   c. Perform the following step to create emissary-ports.yaml:

```
cat <<EOF > emissary-ports.yaml
service:
  ports:
    - name: https
      port: 443
      targetPort: 8443
      #nodePort: <optional>
    - name: http
      port: 80
      targetPort: 8080
      #nodePort: <optional>
    - name: deploy-wss
      port: 7919
      targetPort: 7919
      #nodePort: <optional>
    - name: control-ssh
      port: 9022
      targetPort: 9022
      #nodePort: <optional>
EOF
```

   d. Install Emissary-ingress:

```
helm install emissary-ingress --namespace emissary datawire/emissary-ingress -f
 emissary-ports.yaml && \
kubectl -n emissary wait --for condition=available --timeout=90s deploy
 -lapp.kubernetes.io/instance=emissary-ingress
```

5. Open the ports in your firewall to the external ports and the node ports configured in the previous step.
   You can run the following command to determine the node ports if they are configured automatically:

   ```
   kubectl get svc emissary-ingress --namespace emissary -o jsonpath='{range .spec.ports[*]}{.name}:
    {.nodePort}{"\n"}{end}'
   ```

6. Perform the following steps to access the HCL Harbor container registry:
   a. Get a key to the HCL Harbor container registry.
   b. Log in to HCL Harbor container registry with the HCL ID and password that are associated with the entitled software.
   c. Copy the pre-generated CLI secret from the **User Profile** page.
   d. Create the following three secrets in the target namespace to pull images from the HCL Harbor container registry:

      ```
      kubectl create secret docker-registry hcl-entitlement-key \
          --namespace [namespace_name] \
          --docker-username=<Harbor User ID> \
          --docker-password=<CLI secret> \
          --docker-server=hclcr.io
      ```

      **Note:** Secrets are namespace-specific and they are required to install DevOps Plan.

7. Run the following command to view the `README.md` file:

   ```
   helm show readme oci://hclcr.io/devops-automation-helm/hcl-devops-loop --version 2.0.000
   ```

8. Update the following parameters and the other required parameters in the script in the Helm README with the correct values:
   - DOMAIN
   - TLS_CERT_SECRET_NAME
   - RWO_STORAGE_CLASS=nfs-client
   - RWX_STORAGE_CLASS=nfs-client

   For DOMAIN and TLS_CERT_SECRET_NAME, you must provide the values noted down in the previous steps.

9. Run the script in the Helm README for K8 installation.

   **Note:** If the installation fails due to timeout, run the installation script again.

10. Perform the following steps to enable non-HTTP and additional special services:

a. Configure the DNS to route traffic from a second FQDN that is service-<DOMAIN> to the L4 load balancer that you created as a prerequisite.

   The DOMAIN value is the same as the value used in the helm chart.

b. Configure your L4 node balancer to forward the ports configured in 4.c on page 35 and determined in 5 on page 36 to your cluster.

## Installing on a K8S cluster with an upstream L7 load balancer

**Before you begin**

You must have completed the following tasks:

- Ensured that the external L7 load balancer and cluster support for L4 load balancer resources are available.
- Ensured that an external fully qualified domain name with a certificate signed by a well-known CA or an intermediary is available.

1. Perform the following steps to install Emissary-ingress in your cluster:
   a. Run the following commands to set the Ambassador Edge Stack Helm chart:

```
helm repo add datawire https://app.getambassador.io
helm repo update
```

   b. Run the following commands to create a namespace and install the Ambassador Edge stack:

```
kubectl create namespace emissary && \
kubectl apply -f https://app.getambassador.io/yaml/emissary/3.9.1/emissary-crds.yaml
kubectl wait --timeout=90s --for=condition=available deployment emissary-apiext -n
 emissary-system
```

   c. Perform the following step to create emissary-ports.yaml:

```
cat <<EOF > emissary-ports.yaml
service:
  type: LoadBalancer  #Set to NodePort when using an external L4 load balancer
  ports:
    - name: http
      port: 80
      targetPort: 8080
      #nodePort: <optional>
    - name: deploy-wss
      port: 7919
      targetPort: 7919
      #nodePort: <optional>
    - name: control-ssh
      port: 9022
      targetPort: 9022
      #nodePort: <optional>
EOF
```

   d. If the support for load balancer resources is not available in your cluster, edit the `emissary-ports.yaml` to change the type to NodePort.

      An external L4 load balancer is required in this installation scenario.

e. Install Emissary-ingress:

```
helm install emissary-ingress --namespace emissary datawire/emissary-ingress -f
 emissary-ports.yaml && \
kubectl -n emissary wait --for condition=available --timeout=90s deploy
 -lapp.kubernetes.io/instance=emissary-ingress
```

2. Open the ports in your firewall to the node ports configured in the previous step.
   You can run the following to determine the node ports if they are configured automatically:

```
kubectl get svc emissary-ingress --namespace emissary -o jsonpath='{range .spec.ports[*]}{.name}:
 {.nodePort}{"\n"}{end}'
```

3. Perform the following steps to access the HCL Harbor container registry:
   a. Get a key to the HCL Harbor container registry.
   b. Log in to HCL Harbor container registry with the HCL ID and password that are associated with the entitled software.
   c. Copy the pre-generated CLI secret from the **User Profile** page.
   d. Create the following three secrets in the target namespace to pull images from the HCL Harbor container registry:

```
kubectl create secret docker-registry hcl-entitlement-key \
    --namespace [namespace_name] \
    --docker-username=<Harbor User ID> \
    --docker-password=<CLI secret> \
    --docker-server=hclcr.io
```

**Note:** Secrets are namespace-specific and they are required to install DevOps Plan.

4. Run the following command to view the `README.md` file:

```
helm show readme oci://hclcr.io/devops-automation-helm/hcl-devops-loop --version 2.0.000
```

5. Update the following parameters and the other required parameters in the script in the Helm README with the correct values:
   - DOMAIN
   - RWO_STORAGE_CLASS=nfs-client
   - RWX_STORAGE_CLASS=nfs-client

   For DOMAIN, you must provide the values noted down in the previous steps.

6. Add the following parameter to the ADDITIONAL_HELM_OPTIONS section:

```
--set platform.emissary.l7Depth=<number_of_hops_to_load_balancer>
```

   By default the value is set to 0, which indicates that there is no upstream load balancer. You must set the value to 1 for a single hop to a direct upstream load balancer.

7. Run the script in the Helm README for K8 installation.
8. Perform the following steps to enable non-HTTP and additional special services:

a. If the load balancer resources are available in your cluster, then run the following command to determine the IP of the L4 load balancer:

```
kubectl get svc --namespace emissary emissary-ingress -o
 jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

If the load balancer resources are not available, then configure an external L4 load balancer to open the non-http/https ports in the `emissary-ingress.yaml` to direct traffic to your cluster.

b. Configure the DNS to route traffic from a second FQDN that is service-<DOMAIN> to the L4 load balancer that you created as a prerequisite.

A certificate is not required for this domain. The DOMAIN value is the same as the value used in the helm chart.

## Installing on a K8S cluster that has an upstream L7 load balancer and expects data to be re-encrypted

**Before you begin**

You must have completed the following tasks:

- Ensured that the external L7 load balancer, and cluster support for L4 load balancer resources or an external L4 load balancer are available.
- Ensured that an external fully qualified domain name with a certificate signed by a well-known CA or a self-signed certificate is available as required.

1. Obtain a certificate and key that covers all the nodes in your cluster through a Subject Alternative Name (SAN).
2. Configure your L7 load balancer to trust the certificate for the nodes in your cluster.

**Note:** A health check endpoint is available at `/automation/healthz` for your load balancer to reference for health checks.

3. Make the certificate and key available as a Kubernetes secret in the Kubernetes namespace that you use for DevOps Loop.
   You must use the cert-manager or any standard mechanism to manage the life cycle of the certificate. You must also note down the name of the secret that contains the TLS certificate and key.
4. Perform the following steps to install Emissary-ingress in your cluster:
   a. Run the following commands to set the Ambassador Edge Stack Helm chart:

```
helm repo add datawire https://app.getambassador.io
helm repo update
```

   b. Run the following commands to create a namespace and install the Ambassador Edge stack:

```
kubectl create namespace emissary && \
kubectl apply -f https://app.getambassador.io/yaml/emissary/3.9.1/emissary-crds.yaml
kubectl wait --timeout=90s --for=condition=available deployment emissary-apiext -n
 emissary-system
```

c. Perform the following step to create emissary-ports.yaml:

```
cat <<EOF > emissary-ports.yaml
service:
  type: LoadBalancer # NodePort if no LoadBalancer resources are available in your cluster
  ports:
    - name: https
      port: 443
      targetPort: 8443
      #nodePort: <optional unused if type Nodeport>
    - name: http
      port: 80
      targetPort: 8080
      #nodePort: <optional unused if type Nodeport>
    - name: deploy-wss
      port: 7919
      targetPort: 7919
      #nodePort: <optional unused if type Nodeport>
    - name: control-ssh
      port: 9022
      targetPort: 9022
      #nodePort: <optional unused if type Nodeport>
EOF
```

d. If no load balancer resources are available in your cluster, edit the `emissary-ports.yaml` to change the type to NodePort.

An external L4 load balancer is required in this installation scenario.

e. Install Emissary-ingress:

```
helm install emissary-ingress --namespace emissary datawire/emissary-ingress -f
 emissary-ports.yaml && \
kubectl -n emissary wait --for condition=available --timeout=90s deploy
 -lapp.kubernetes.io/instance=emissary-ingress
```

5. Open the ports in your firewall to the node ports configured in the previous step.
   You can run the following to determine the node ports if they are configured automatically:

```
kubectl get svc emissary-ingress --namespace emissary -o jsonpath='{range .spec.ports[*]}{.name}:
 {.nodePort}{"\n"}{end}'
```

6. Perform the following steps to access the HCL Harbor container registry:
   a. Get a key to the HCL Harbor container registry.
   b. Log in to HCL Harbor container registry with the HCL ID and password that are associated with the entitled software.
   c. Copy the pre-generated CLI secret from the **User Profile** page.
   d. Create the following three secrets in the target namespace to pull images from the HCL Harbor container registry:

```
kubectl create secret docker-registry hcl-entitlement-key \
    --namespace [namespace_name] \
    --docker-username=<Harbor User ID> \
```

```
    --docker-password=<CLI secret> \
    --docker-server=hclcr.io
```

**Note:** Secrets are namespace-specific and they are required to install DevOps Plan.

7. Run the following command to view the `README.md` file:

```
helm show readme oci://hclcr.io/devops-automation-helm/hcl-devops-loop --version 2.0.000
```

8. Update the following parameters and the other required parameters in the script in the Helm README with the correct values:
    - DOMAIN
    - TLS_CERT_SECRET_NAME
    - RWO_STORAGE_CLASS=nfs-client
    - RWX_STORAGE_CLASS=nfs-client

    For DOMAIN and TLS_CERT_SECRET_NAME, you must provide the values noted down in the previous steps.
9. Run the script in the Helm README for K8 installation.
10. Perform the following steps to enable non-HTTP and additional special services:
    a. If the load balancer resources are available in your cluster, then run the following command to determine the IP of the L4 load balancer installed as part of DevOps Loop:

    ```
    kubectl get svc --namespace emissary emissary-ingress -o
     jsonpath='{.status.loadBalancer.ingress[0].ip}'
    ```

    If the load balancer resources are not available, then configure an external L4 load balancer to open the non-http/https ports in `emissary-ingress.yaml` to direct traffic to your cluster.

    b. Configure the DNS to route traffic from a second FQDN that is service-<DOMAIN> to the L4 load balancer that you created as a prerequisite.

    A certificate is not required for this domain. The DOMAIN value is the same as the value used in the helm chart.

## Installation of DevOps Loop on Google Kubernetes Engine (GKE)

You can find information about tasks that you can perform to install DevOps Loop on Google Cloud Platform (GCP) by using the Google Kubernetes Engine (GKE) service.

## Preparing GCP infrastructure for installing DevOps Loop on GKE

You must set up the required Google Cloud Platform (GCP) infrastructure including DNS, a global static IP address, and RWX Storage, before deploying DevOps Loop on Google Kubernetes Engine (GKE).

**Before you begin**

You must have completed the following tasks:

- Ensured that your GCP environment includes the required roles and APIs:
  - The service account used for the GKE cluster must have the `roles/file.serviceAgent` IAM role.
  - Enabled the following APIs at the project level:
    - `file.googleapis.com`
    - `gkehub.googleapis.com`

      These APIs and permissions are required for filestore provisioning, CSI driver operations, and certificate management within GKE.

1. Perform the following steps to configure DNS and network resources:
   a. Create a host name in Cloud DNS for the DevOps Loop application.
   b. Reserve a **global static IPv4 address** in the same GCP project.
   c. Create a DNS A record that maps the domain name to the global static IP address.

   For example:
   - Name: `<host-name>`
   - Type: `A`
   - Value: `<GLOBAL_STATIC_IP>`

   **Note:** DNS must resolve and must be visible from the public internet or at least from GCP checkers for GKE-managed certificate provisioning to succeed.

2. Perform the following steps to configure `RWX StorageClass` for DevOps Loop:

   a. Create a custom `RWX StorageClass` with the following requirements:

   - Minimum 1 TiB storage
   - Provisioner: `filestore.csi.storage.gke.io`
   - VPC: must match the VPC network used by your GKE cluster

   **Note:** A custom `StorageClass` is required so filestore volumes are created within your cluster's VPC. Otherwise, GCP-provided filestore classes attempt provisioning under the default network.

3. Verify available `StorageClasses` by running the following command:

```
kubectl get storageclass
```

**Results**

You have prepared the required Google Cloud Platform (GCP) infrastructure for deploying DevOps Loop on Google Kubernetes Engine (GKE).

**What to do next**

You can now proceed with installing DevOps Loop on GKE.

## Installing DevOps Loop on Google Kubernetes Engine (GKE)

You can install HCL DevOps Loop on a newly created GKE cluster by using the Helm chart.

**Before you begin**

You must have completed the following tasks:

- Read and understood System Requirements for DevOps Loop 2025.12 (2.0.0) on page 16.
- Gained access to the cluster and installed the following tools:
  - Kubectl
  - Google Cloud SDK and Google Cloud CLI
- Installed Helm on the system from which you access the Kubernetes cluster. For more information, refer to Installing Helm.
- Set up a Kubernetes (GKE) cluster under the platform - Google Kubernetes Engine (GKE) and Image type - Container-Optimized OS with containerd (cos_containerd). For more information, refer to Create a cluster and deploy a workload in the Google Cloud console.

  **Note:** Ensured that you have enabled the Filestore CSI driver and Cloud Storage Fuse CSI driver while creating the cluster to use the GCP CSI driver storage class.

- Prepared the required infrastructure in Google Cloud Platform (GCP). See Preparing GCP infrastructure for installing DevOps Loop on GKE on page 41.
- Ensured that you have the following details:
  - **DOMAIN (FQDN)** – the hostname you created in Cloud DNS for the Loop installation.
  - **STATIC_IP_ADDRESS** – the global static IP mapped to the DNS A record.
  - **NAMESPACE** – the target Kubernetes namespace (for example: `devops-loop`).

1. Run the following command to download the `GKE-Infra-Setup.sh` script from the Helm chart to provision and configure the load balancer with a Google managed certificate.

   ```
   (TMP_DIR=$(mktemp -d) && \
   helm pull ibm-helm/ibm-devops-loop --untar --untardir "$TMP_DIR" --version 2.0.000 > /dev/null 2>&1 && \
   sh "$TMP_DIR/ibm-devops-loop/scripts/GKE/GKE-Infra-Setup.sh" && \
   rm -rf "$TMP_DIR")
   ```

   This command downloads and copies, and runs the `GKE-Infra-Setup.sh` script in your temporary directory.

   After you run the script, wait for certificate provisioning to complete. This process can take 10–60 minutes, depending on your cluster and DNS configuration.
2. Perform the following steps to access the IBM Entitled Registry:

a. Log in to the **My IBM Container Software Library** using the ID and password associated with your entitled software.
b. Navigate to the Entitlement keys section and select **Copy key** to copy your IBM Entitled Registry entitlement key to you clipboard.
c. Create a secret in the target namespace named `hcl-entitlement-key` to pull images from the HCL Harbor container registry:

```
kubectl create secret docker-registry hcl-entitlement-key \
    --namespace [namespace_name] \
    --docker-username=<Harbor User ID> \
    --docker-password=<CLI secret> \
    --docker-server=hclcr.io
```

3. Run the following command to add the repository:

```
helm repo add hclsoftware
https://us-artifactory1.nonprod.hclpnp.com/artifactory/hclsoftware-helm-dev --force-update
```

4. Run the following command to view the `README.md` file:

```
helm show readme hclsoftware-helm/hclsoftware-devops-loop
```

5. Update the following parameters in the Helm installation script with the correct values:
    - DOMAIN - the FQDN created in Cloud DNS
    - TLS_CERT_SECRET_NAME – Leave it blank to use the GCP-managed certificate.
    - RWO_STORAGE_CLASS=`standard-rwo` OR `premium-rwo` as required.
    - RWX_STORAGE_CLASS=custom storage class name you have created previously for RWX under the filestore csi driver. (e.g.: `custom-rwx`)
6. Run the script in the Helm README for installing DevOps Loop on GKE.

   If the installation fails due to a timeout, run the installation script again.

**Results**

You have installed DevOps Loop on Google Kubernetes Engine (GKE).

**What to do next**

You can add users and manage user access by logging in to the Keycloak instance that is installed with DevOps Loop. See User access and administration using Keycloak on page 55.

## Installing DevOps Loop on a RHEL system for a demo setup

**Disclaimer:** In this demo setup, DevOps Loop is deployed on a RHEL-based system with minimum hardware and software specifications. This demo setup is intended solely for evaluation purposes and might not accurately represent the performance, reliability, or feature set of the full production environment. The demo setup might not include enhanced security configurations typical of production systems. No warranty is implied regarding uptime, support, or continuity of access.

You can find information about the tasks that you can perform to install HCL DevOps Loop on a Red Hat Enterprise Linux (RHEL) system for a demo setup.

**Before you begin**

You must have completed the following tasks:

- Ensured that the RHEL system on which you are deploying DevOps Loop meets the following minimum requirements:
  - Operating system: RHEL 9.4 or later
  - Disk space: 300 GB
  - RAM: 64 GB
  - CPU: 16 Core
- Downloaded the Installation scripts.
- Obtained the user credentials of the licensed Docker account.

**Note:** If you do not provide the licensed docker account credentials, the system might take approximately 6 hours to become operational.

- Set these environment variables in your shell before you run the `RHEL9-K8S-Infra-Setup.sh` script:
  - export USERNAME=<docker_username>
  - export PASSWORD=<docker_password>

  The `<docker_username>` and `<docker_password>` are your Docker Hub credentials.

  If the above values are not set, you are prompted with login steps during the script run.

1. Run the `RHEL9-K8S-Infra-Setup.sh` script:

```
./RHEL9-K8S-Infra-Setup.sh
```

2. Run the following command to create a namespace:

```
kubectl create namespace devops-loop
```

3. Perform the following steps to access the HCL Harbor container registry:
   a. Log in to the HCL Harbor container registry using your HCL ID and password by selecting the **LOGIN VIA OIDC PROVIDER** authentication method.
   b. Copy the **CLI secret** by opening your **User Profile** from the top right corner of the page.

      This value serves as the password for the `docker-registry` secret creation command.

   c. Create an `imagePullSecret` to authenticate and pull images from the HCL Entitled Registry.

      **Note:** Secrets are namespace-scoped and must be created in each namespace where you plan to install DevOps Loop.

   Use the following command to create an `imagePullSecret` named `hcl-entitlement-key`:

```
kubectl create secret docker-registry hcl-entitlement-key \
    --docker-username=<username> \
    --docker-password=<password> \
    --docker-server=hclcr.io
    --namespace devops-loop \
```

For example:

```
kubectl create secret docker-registry hcl-entitlement-key \
    --docker-username=xxxx@hcl.com \
    --docker-password=AOve4CzW2Hs0yCnuQxxxxxxvX \
    --docker-server=hclcr.io
    --namespace devops-loop \
```

4. Update the following values in the `HCL-devops-Install.sh` script:
    - LICENSE_SERVER=
    - LICENSE_ID=
    - EMAIL_SERVER_HOST=
    - EMAIL_SERVER_PORT=
    - EMAIL_FROM_ADDRESS=
    - EMAIL_SERVER_USERNAME=""
    - EMAIL_SERVER_PASSWORD=""
5. Run the following command to start the installation script:

    ```
    ./HCL-devops-Install.sh
    ```

    After the successful installation, the following URLs are displayed:

    Application URL: `https://<system_IP_address>.nip.io`

    Keycloak URL to add users: `https://<system_IP_address>.nip.io/auth`

    **Note:** To onboard users to DevOps Loop, see User access and administration using Keycloak on page 55. After the user is onboarded, you can add the user to teamspaces and loops.

    **Note:** If you are installing DevOps Loop on a computer that is newly set up, then the installation might fail sometimes. If the installation fails, to resolve the issue, you must uninstall the application by running the `uninstall.sh` script and then reinstall it.

    **Note:** If the installation fails due to timeout, run the installation script again.

6. **Optional:** Run the following command to check whether all the pods are running:

    ```
    kubectl get pods -n devops-loop
    ```

7. Open a browser and enter the application URL to get started with DevOps Loop.
8. Run the following command to log out of Docker to remove any cached authentication tokens, after completing the DevOps Loop installation:

    ```
    docker logout
    ```

    This command removes the login credentials for `https://index.docker.io/v1/` and cleans up Docker credentials stored in the cache (for example, `/root/.docker/config.json`). Logging out is recommended for security, as Docker authentication tokens are otherwise retained on the system.

**Results**

You have installed DevOps Loop on a RHEL system for a demo setup.

## Installation of DevOps Loop in an air-gapped environment

An air-gapped environment is a network or system isolated from the public internet, typically for security or compliance reasons. In such environments, all external resources must be imported manually. DevOps Loop supports air-gapped installations by providing scripts and configurations to transfer required resources into your private environment.

Some deployments operate without internet access, requiring all container images, plugins, and dependencies to be preloaded into a private registry.

**Scripts and configurations for Air-gapped installation**

DevOps Loop provides the `airgap.sh` script along with Helm configuration parameters to support air-gapped setups.

| Parameter | Description | Example value |
|---|---|---|
| `global.imageRegistry` | Private registry for product images | `registry.local:5000/devops-loop` |
| `global.externalImageRegistry` | Private registry for unqualified/public images. The `airgap.sh` script places qualified and unqualified images in the same image repository, you can separate them if required. | `registry.local:5000/public` |
| `airgap.sh` | Script to list or copy required images to private registry | `./airgap.sh -\|` |

Following are the functions of `airgap.sh` script:

- List all the container images required by DevOps Loop (using `-l` option).
- Copy images from public registries to your private registry.

## Preparing images and resources for an air-gapped installation

Before you begin the air-gapped installation, you must prepare the images and resources. You must collect and transfer all container images and required plugins to your private registry or offline storage.

**Before you begin**

You must have completed following tasks:

- Ensured that you have the administrative privileges on the target Kubernetes namespace and the airgap container registry.
- Downloaded the DevOps Loop Helm chart and it is accessible.
- Downloaded the `airgap.sh` script and ensured that it is available in the DevOps Loop Helm chart.

- Allocated sufficient disk space to save product images and plugin files for transfer to the air-gapped environment.

**Note:** Product images will consume approximately 50 GB or more of disk space in your internal/air-gapped image registry.

**About this task**

To copy the DevOps Loop product images from public repositories to your specified internal or air-gapped image registry, follow the steps below. The system where you run these steps must have access to both the internet and the internal or air-gapped registry.

As part of the DevOps Loop installation, additional components such as **MongoDB** (used to store application and configuration data) and the **MySQL JDBC driver** (required by the DevOps Deploy server) are also needed. In air-gapped environments, these components cannot be downloaded automatically and must be prepared manually. The steps below guide you through copying the DevOps Loop images as well as preparing MongoDB and the JDBC driver for installation.

1. Locate the `airgap.sh` script in the scripts/airgap folder of the downloaded DevOps Loop Helm chart.
2. Run the following command to copy the DevOps Loop images into the internal/airgap image registry:

```
sh ./ibm-devops-loop/scripts/airgap/airgap.sh -r
        ibm-devops-loop -d registry.local:5000/devops-loop
```

3. Perform the following steps to prepare MongoDB for an air-gapped installation:

**Note:** The DevOps Loop installation script deploys MongoDB in the same namespace as DevOps Loop. In an air-gapped environment, you must download the Helm chart locally and update the script, because it cannot fetch the chart from public repositories. The MongoDB server image is already available in your air-gapped registry.

   a. Download and extract the MongoDB Helm chart version 14.13.0 from Bitnami Helm Charts into the same directory as the expanded `ibm-devops-loop` Helm chart.
   b. Update the installation script to set `MONGO_IMAGE_REPO=registry.local:5000/devops-loop` and replace `bitnami/mongodb` with `mongodb` in the `helm upgrade --install` command.

4. Perform the following steps to prepare the MySQL JDBC Driver for an air-gapped installation:

**Note:** The DevOps Deploy server, installed as part of DevOps Loop, creates a MySQL database to store various artifacts. In an air-gapped environment, the installation cannot automatically download the required JDBC driver. You must make the drive available manually before starting the DevOps Loop installation. You can also refer to the JDBC Driver section of the DevOps Deploy Prerequisites document. See the link provided in Related information.

   a. Download the required MySQL JDBC driver to a drive that is accessible in an air-gapped environment.
   b. Write a bash script, named `script.sh` that copies the JDBC driver from a location accessible from your cluster to `${UCD_HOME}/ext_lib/`.

c. Store the script `script.sh` in a yaml file describing a Kubernetes ConfigMap.

The Kubernetes ConfigMap is defined for copying the JDBC driver to your Persistent Volume (PV) during the chart installation process.

Below is an example ConfigMap yaml file that copies a MySQL .jar file from a web server using `wget`:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: user-script
data:
  script.sh: |
    #!/bin/bash
    echo "Running script.sh..."
    if [ ! -f ${UCD_HOME}/ext_lib/mysql-jdbc.jar ] ; then
      echo "Copying file(s)..."
      wget -L -O mysql-jdbc.jar http://webserver-example/mysql-jdbc.jar
      mv mysql-jdbc.jar ${UCD_HOME}/ext_lib/
      echo "Done copying."
    else
      echo "File ${UCD_HOME}/ext_lib/mysql-jdbc.jar already exists."
    fi
```

d. Create the ConfigMap in your cluster by running a command, such as:

```
oc create configmap <map-name> <data-source>
```

e. Specify the ConfigMap name in Helm using:

```
ibm-ucd-prod.extLibVolume.configMapName
```

**Results**

You have collected all the required DevOps Loop container images, plugins, and JDBC drivers, and transferred them to your airgap image or offline storage, making the environment ready for air-gapped installation.

---

Related information

DevOps Deploy Prerequisites

## Installing DevOps Loop in an air-gapped environment

After preparing the required images, plugins, and JDBC drivers in your air-gapped registry or offline storage, you can begin installing DevOps Loop in an air-gapped environment by using Helm.

**Before you begin**

You must have completed the following tasks:

- Preloaded images and resources. See Preparing images and resources for an air-gapped installation on page 47.
- Ensured that you have administrative privileges on the Kubernetes namespace where DevOps Loop will be installed.

- Installed and configured Helm on your system.
- Gained access to the airgap image containing the preloaded images.

1. Modify the `ADDIITIONAL_HELM_OPTIONS` variable in the DevOps Loop installation script, to set the following Helm values to point to your airgap registry:

```
--set global.imageRegistry=registry.local:5000/devops-loop
--set global.externalImageRegistry=registry.local:5000/devops-loop
--set platform.velocity.plugins.plan.image=ucv-ext-compass:1.2.2
--set platform.velocity.plugins.control.image=hclcr.io/accelerate/ucv-ext-control:1.1.1
--set platform.velocity.plugins.test.image=hclcr.io/accelerate/ucv-ext-onetest-server:1.0.32
--set platform.velocity.plugins.deploy.image=hclcr.io/accelerate/ucv-ext-launch:5.1.1
--set platform.velocity.plugins.build.image=hclcr.io/accelerate/ucv-ext-build:1.1.2
--set platform.velocity.plugins.yaml.image=hclcr.io/accelerate/ucv-ext-yaml-executor:0.0.13
```

2. Edit the DevOps Loop installation script to use the locally downloaded and expanded DevOps Loop Helm chart.
   For **OpenShift (OCP)** installations, no further changes to the installation script are required.
   For **Kubernetes (K8s)** installations, you must locate the `helm upgrade --install` command at the end of the installation script and replace `ibm-helm/ibm-devops-loop` with `ibm-devops-loop`.
3. Follow the DevOps Loop installation instructions based on your environment.

**Results**

You have installed DevOps Loop in the air-gapped environment.

**What to do next**

Before installing additional plug-ins, ensure the required images are available in your internal or air-gapped environment:

- If you want to install Measure plug-in images that are not included in the standard DevOps Loop set, download them from their public repositories (such as docker.io or hclrc.io) and upload them to your internal or air-gapped image registry.
- If you want to install Deploy plug-ins that are not included in the standard set, download them from their public repositories to local storage, then upload them to the DevOps Deploy server.

---

Related reference

Installation of DevOps Loop on page 30

## Managing DevOps Loop features after installation

After you install HCL DevOps Loop, you can manage and customize certain features based on your requirements.

During the installation of DevOps Loop, if you did not use any Helm parameters, you can still use those parameters even after installation is complete.

You can use the helm upgrade command to enable or disable server features after the installation of DevOps Loop is complete.

The sample code is to enable the Egress policy that restricts traffic to private IP addresses by retaining the values of other parameters that you used during the installation.

Similarly, you can use the other additional Helm parameters to enable or disable features based on your requirements. See Additional Helm parameters on page 51.

## Additional Helm parameters

You can find the information about Helm parameters that you can use during the installation of HCL DevOps Loop.

As a system administrator, you must complete the additional configuration for the capabilities by using Helm parameters. Each Helm parameter that is specific to the solution is prefixed by its Helm chart name as follows:

- Plan: `hcl-devopsplan-prod`
- Test: `hcl-devops`
- Deploy: `hcl-launch-server-prod`
- Measure: `hcl-velocity`

For example, to configure a property in Plan, use the following command:

```
bash
--set hcl-devopsplan-prod.property=value
```

You can run the following commands to view the required parameters:

- Plan: helm show readme hcl-helm/hcl-devopsplan-prod # DevOps Plan
- Deploy: helm show readme hcl-helm/hcl-launch-server-prod # DevOps Deploy
- Test: helm show readme hcl-helm/hcl-devops # DevOps Test
- Measure and Release: helm show readme hcl-helm/hcl-velocity # DevOps Velocity

# License management and user administration

After you install HCL DevOps Loop, you must consider how the platform manages licenses, users, and authentication.

**License management**

During installation, you can set the licensing information only by using the License server details in the Helm chart. See DevOps Loop licensing information on page 52.

The **Users** tab on the **User Administration** page displays the licensing details such as the total number of licenses in the active tier and the number of licenses in use. The tab also provides options to revoke and assign licenses.

**User administration**

As part of administration, you can add users to DevOps Loop and assign roles in Keycloak. See User access and administration using Keycloak on page 55 and Enabling the social sign-up and social login for DevOps Loop on page 54.

You can use the default user management provided by the platform and decide what additional controls you might want to add. If you manage users and authentication through social login, you can review how to use that server to manage users of DevOps Loop.

The admin user can manage users within tenants. Tenant represents an organization within which multiple teamspaces can be created and each teamspace can contain multiple loops. Admin users can access the **User Administration** page in DevOps Loop by clicking **User Administration** under **Settings**. The **User Administration** page displays the name of the tenant and teamspaces within the tenant on the left side, while the right side displays the **Details** and **Users** tabs.

- **Details**: The tab displays the number of users, number of user licenses, number of teamspaces, and number of loops associated with the tenant. The tab also displays the tenant details such as the name of the tenant, the tenant ID, and creation date.
- **Users**: The tab displays a list of users and user details. You can click a user to view their name, email ID, loops that are associated with the user, teamspaces that are associated with the user, and assigned roles. The **Users** tab facilitates license management. As an admin, you can revoke licenses for other users but not for yourself. Upon revoking a license, a confirmation message is displayed, and the **Revoke** option becomes disabled for that user. The **Licenses** section in the top bar displays the total number of licenses and the number of licenses that are in use in the active tiers. When you assign or revoke licenses for other users, the license information is also updated.

## DevOps Loop licensing information

You can find the details of licensing that HCL DevOps Loop supports.

DevOps Loop supports the following 3 tiers of license:

- **Essentials** (Tier 1): Plan, Control and Code
- **Standard** (Tier 2): Plan, Control, Code, Test, and Deploy
- **Premium** (Tier 3): Plan, Control, Code, Test, Deploy, Measure, and Release

Each tier is defined by the number of capabilities that are included in the DevOps Loop platform.

Licenses are consumed when the admin assigns them, and only released when the admin explicitly revokes them or DevOps Loop is uninstalled. License tier re-assignments may take up to 15 minutes to reflect.

By default, non-admin users must be assigned a tier before logging in, while admin users automatically receive the highest available tier. This per-user tier assignment allows different access levels to be enforced for different users, providing flexibility in license management.

**Configuration**

When you purchase one of these tiers, you must map the licenses to My HCLSoftware (MHS). MHS is a cloud-based web application that helps to manage software entitlements and licenses. When you install DevOps Loop, you must specify the License server URL and ID, which are configured only through the Helm chart. For more information about managing licenses for your product, refer to the following resources:

- My HCLSoftware
- Managing HCL Local License Server or Managing HCL Cloud License Server

**License allocation**

You must run the following command to retrieve information on the licensing allocation:

curl --cacert /path/to/your/certificate.crt https://FQDNhostname:LLSport/v1/licensepools/poolID

**Sample output:**

```
{
  "id": "MyPoolID",
  "features": {
    "DevOpsAutomationEssentials": {
      "total": 100,
      "leased": 0,
      "available": 100,
      "name": "DevOps Automation Essentials User"
      }
    }
}
```

When you run the command to retrieve the current license allocations, depending on the tier that you have opted for, the following values are displayed:

- **DevOpsAutomationEssentials**
- **DevOpsAutomationStandard**
- **DevOpsAutomationPremium**

The license feature names that are used by the License server are different from the product name.

**Usage**

After the Admin user creates a user, when the user logs in to DevOps Loop, then one license of the highest available tier is consumed. The user can access the products that are included in the licensing tier that is purchased.

DevOps Loop provides a user interface to view and manage licensing. You can revoke or assign a user's license through the **User Administration** page. See License management and user administration on page 51 and Revoking a license on page 55.

**Troubleshooting**

You can view the licensing logs for troubleshooting by running the following command:

```
kubectl logs <licensing_pod_name> -n <namespace>
```

Where, the *namespace* is defined by the Helm installation. You can find the *licensing_pod_name* by running the following command:

```
kubectl get pods -n <namespace>
```

The pod name that contains 'licensing' in the name is listed.

# Enabling the social sign-up and social login for DevOps Loop

You can enable social authentication for users to sign up and log in to HCL DevOps Loop by using a social network account. The Keycloak admin can configure this social sign-up in the Keycloak UI.

**Before you begin**

You must have completed the following tasks:

- Installed DevOps Loop.
- Assigned the role of a Keycloak Administrator.
- Copied the URL of the Keycloak UI, username, and password on completion of the installation of the platform.
- Obtained the credentials to access the dashboard of a social identity provider to configure the delegation of authentication to a social media account.
- Read and understood Integrating identity providers.

**About this task**

The social sign-up link on the **Login** page of DevOps Loop is not visible when you install DevOps Loop by using the default values. The sign-up link is hidden to provide enhanced security to the platform. You can enable the sign-up link on the **Login** page when you want to provide self-registration of users in DevOps Loop.

The following procedure describes steps to be performed to enable the **Sign up** link by using Keycloak UI.

1. Log in to the Keycloak Admin Console.
2. Click **Identity providers** from the left navigation pane.
3. Select an identity provider under the **Social** section.
   The configuration page for the selected identity provider is displayed.
4. Enter the configuration details.
   The fields in the configuration page depend on the social identity provider that you have selected.
5. Click **Add** to apply the changes.

**Results**

You have enabled the social sign-up link on the **Login** page of DevOps Loop.

**What to do next**

You must add the self-registered users to the **Users** group in the Keycloak Admin Console. See User access and administration using Keycloak on page 55.

---

Related information

Keycloak

## Revoking a license

Each user is assigned a license when they log in to the DevOps Loop for the first time. As an admin, you can revoke the assigned license if the user is no longer part of the platform.

1. Log in to DevOps Loop.
2. Click  **Settings > User Administration**.
   The **User Administration** page is displayed.
3. Click the **Users** tab.
   All the users are displayed. You can search for a user by their name or email ID if the list is long.
4. Click the **Revoke** slider next to the username.
   Alternatively, you can select a user, and then click the **REVOKE LICENSE** button.

   A confirmation dialog is displayed.
5. Click **Yes, Revoke**.
   A confirmation message is displayed, and the slider changes to **Assign** using which you can assign the license again.

   Also, the **Licenses** section displays the updated number after revoking a license.

**Results**

You have revoked the assigned license for a user.

## User access and administration using Keycloak

HCL DevOps Loop uses Keycloak (https://www.keycloak.org/) to manage and authenticate users. You can manage user access by logging in to the Keycloak instance that is installed with DevOps Loop.

Keycloak uses the concept of a realm to manage and authenticate users. When you install DevOps Loop, a realm called *platform* is created for you in Keycloak. All server users belong to this realm and when they log in to DevOps Loop, they log into that realm.

As an administrator, it is important to consider the following points about the platform administration:

- To begin with, there is no administrator for DevOps Loop.

  Such an administrator is required for accessing additional functions, which include claiming ownership of projects and unarchiving them. However, you can assign administrative privileges to any user. You must assign the privilege by adding the admin role to the user in Keycloak.
- You must add a user that you want to be the administrator in Keycloak by logging in to the Keycloak Admin Console at `https://<fully-qualified-dns-name>/auth/`.

**Note:** Do not use that admin user to perform non-administration tasks. Instead, add another user.

The default username for the Keycloak administrator is `keycloak`. The password is randomly generated when the software is installed. You can see the password by using the following kubectl command:

```
kubectl get secret -n <namespace> <helm name>-keycloak -o jsonpath="{.data.password}" | base64 --decode;
 echo
```

- After you add the user that you want to be the administrator for DevOps Loop, you must make that user the administrator.

  In the Keycloak Admin Console, on the **Users** page, you can search and select the user that you want to make an administrator. Then, in the **Groups** tab, you can add the user to the **Admins** group.

- All users must have the **Users** group assigned to them to access DevOps Loop.

  In the Keycloak Admin Console, on the **Users** page, you can search and select a user, and then in the **Groups** tab, you can add the user to the **Users** group.

  For more information about assigning user roles, see Groups in the Keycloak documentation.

Now that you are the platform administrator, it is important to consider the following points about the default user management and authentication:

- Minimum password length is 8 characters
- Email verification of new users is turned off
- The Forgot Password feature is turned on by default but no instructions are sent to the user to reset their password
- Forgotten user passwords are changed by you if you do not enable Keycloak to send instructions to reset a password

**Note:** If a user is added through the user management of HCL DevOps Plan, then that user must have the **Users** or **Admins** group assigned in the Keycloak Admin Console to get access to DevOps Loop.

You can review the following sections about changing the default authentication controls.

### Email settings

The default status of the Forgot Password switch is ON in the *devops-automation* realm. However, as an administrator, you must enable Keycloak to send an email to the user with instructions to reset their password. If you want to verify an email, you must also enable Keycloak to send an email to the user to verify their email address.

You must provide SMTP server settings for Keycloak to send an email. After you log in to the Keycloak Admin Console, see Email Settings in the Keycloak documentation.

Then, to set up the email verification, see Forgot Password in the Keycloak documentation.

### Password policy

The *devops-automation* realm has a password policy where the minimum length of a password is 8. As an administrator, you can update password policies in Keycloak.

After you log in to the Keycloak Admin Console, see Password Policies in the Keycloak documentation.

### User password

When you create a user, you must create credentials for the user by clicking the **Set password** button in the **Credentials** tab. Then, you must share the username and password with the user. While setting the password, if the **Temporary** slider button is set to **On**, then the user must set a new password before logging in to the platform.

If you did not enable Keycloak to send instructions to a user about how to reset a password, you must use the Keycloak Admin Console to change their password for them.

After you log in to the Keycloak Admin Console, see User Credentials in the Keycloak documentation.

### User deletion

When a user is inactive or no longer needs to access the platform, you can delete that user.

After you log in to the Keycloak Admin Console, see Deleting Users in the Keycloak documentation.

## About user roles and access permissions

Users have different roles and access levels on the platform and in integrated capabilities, depending on their tasks and the settings in each application.

Two types of roles are supported in HCL DevOps Loop: admin and user. Admin users can create teamspaces and loops.

| Role | Permissions |
|---|---|
| Admin | Admins have full permission for the following features in the platform:<br>• User administration<br>• Integration<br>• Teamspace management<br>• Loop management<br>• Loop Genie<br>• Dashboards |
| User | Users have view permission for the following features in the platform: |

| Role | Permissions |
| --- | --- |
| | • View loops that the user is a part of<br>• Switch to the application<br>• Loop Genie<br>• Dashboards |

**Notes:**

- Loop creator is granted the highest permissions for the resources in the integrated applications that are created by the loop. The other invited users are granted basic access. The loop creator can elevate the other users' access as required.
- If you are an admin user in the platform but a loop member, in Deploy, you will be given the same privileges as the loop creator. For information on roles and permissions in Deploy, see DevOps Loop: Deploy security management.

# Backup and restoration of DevOps Loop

You must back up HCL DevOps Loop data before you uninstall the current version of DevOps Loop. You must back up the data to avoid data loss or inaccessibility.

A backup creates a copy of your data and stores it in secondary storage. You can use this copy to recover if the original data is lost or becomes inaccessible.

A restore copies backed-up data from secondary storage back to its original location, making it available again.

You must back up and restore the data when you perform the following tasks:

- Move the existing environment to a new system.
- Change the name of the release or namespace that you used during the installation of DevOps Loop.
- Minimize the downtime of DevOps Loop during disaster recovery.

## Tools supporting backup and restoration

DevOps Loop uses the following open source tools to perform backup and restore operations:

- **Velero**: It is a Kubernetes native backup and restore tool that captures and restores Kubernetes resources, persistent volumes, and application state. Velero also supports disaster recovery and migration scenarios between clusters.
- **MinIO**: It is an S3 compatible object storage system used as the backup storage target for Velero. MiniO stores the backups created by Velero and enables you to retrieve them for restore operations.

Velero is installed in your Kubernetes cluster and configured to use MinIO or any other S3-compatible storage backend.

**Note:** MinIO is provided here as an example S3-compatible storage solution for reference purposes only.

To install Velero and MiniO in your cluster, you must run the following script:

```
./ibm-devops-loop/scripts/backup-restore/install-minio.sh
./ibm-devops-loop/scripts/backup-restore/install-velero.sh
```

After installing MinIO, run the port-forward commands to make it accessible on your local machine:

```
```bash
kubectl port-forward --address 0.0.0.0 pod/minio -n loops-minio 9000:9000 9090:9090
```

Then, use the MinIO Web UI (http://localhost:9090) to create a bucket (for example, "loops-backup") to store Velero backups.

## Backing up DevOps Loop data

You can back up DevOps Loop by using Velero and Minio to support disaster recovery and upgrade scenarios.

**Before you begin**

You must have completed the following tasks:

- Ensured that all DevOps Loop services are running and in a healthy state.
- Gained administrative privileges.
- Installed Velero and configured with an S3-compatible storage backend (MinIO or AWS S3). See Tools supporting backup and restoration on page 58.
- Locally saved the Helm values for your current DevOps Loop deployment.

**About this task**

DevOps Loop utilizes both block and file storage, allowing the back up process to support CSI snapshots and fs-volume backups.

"

**Important:**If DevOps Loop pods that use file storage for fs-volume backups are not labeled, Velero will not include their data in the backup, which could result in incomplete or inconsistent backup data.

"

1. Run the following command to label DevOps Loop pods that use file storage for fs-volume backups:

   ```
   ./ibm-devops-loop/scripts/backup-restore/annotate-pods.sh
   ```
   ```

**Note:** If your DevOps Loop deployment only uses block storage volumes (CSI snapshots), you can skip this step.

2. Export the Helm configuration of your DevOps Loop deployment to a local file by running the following command:

```
helm get values devops-loop -n devops-loop > devops-loop-values-backup.yaml
```

3. Run a Velero backup of the DevOps Loop namespace, including PVCs, PVs, and Secrets, using the following command:

```
velero backup create devops-loop-backup \
  --include-namespaces devops-loop \
  --include-resources persistentvolumeclaims,persistentvolumes,secrets \
  -n velero
```

4. Run the following command to verify the backup status to ensure it completed successfully:

```
velero backup describe devops-loop-backup --details
```

5. **Optional** Backup any external databases with their native backup tools to ensure data consistency.
6. Optional: Check the status and logs of your restore jobs to troubleshoot any issues:

```
# Check backup status
velero get backup

# Describe specific backup
velero describe backup <backup-name>

# View backup logs
velero backup logs <backup-name>
```

**Results**

You have created a backup of the DevOps Loop namespace and Helm values for disaster recovery or upgrade scenarios.

## Restoring DevOps Loop data

You can restore DevOps Loop from a backup created with Velero and Helm values after a disaster recovery event or a failed upgrade.

**Before you begin**

You must have completed the following tasks:

- Ensured that the Kubernetes environment hosting DevOps Loop is running and accessible.
- Gained administrative privileges
- Installed Velero and configured with an S3-compatible storage backend (MinIO or AWS S3). See Tools supporting backup and restoration on page 58.
- Locally saved the Helm values for your current DevOps Loop deployment.

**About this task**

DevOps Loop utilizes both block and file storage, allowing the restore process to support CSI snapshots and fs-volume backups.

“

**Important:**If DevOps Loop pods that use file storage were not labeled during backup, their data will not be restored, which could result in incomplete or inconsistent recovery.

”

1. Run the following command to restore the DevOps Loop namespace from the Velero backup:

```
velero restore create --from-backup devops-loop-backup
```

2. Verify that the restore is completed successfully by running the following command:

```
velero restore describe devops-loop-backup --details
```

3. Reinstall or upgrade DevOps Loop using the saved Helm values if necessary:

```
helm upgrade --install devops-loop devops-loop-chart -n devops-loop -f devops-loop-values-backup.yaml
```

**Note:** If persistent volumes are larger than the default Helm chart sizes, override them using `--set` options during reinstall.

4. **Optional**: Restore any external databases used by DevOps Loop using their native restore tools to ensure data consistency.
5. Restart DevOps Loop to ensure all services reflect the restored state:

```
kubectl rollout restart deployment devops-loop -n devops-loop
```

6. Optional: Check the status and logs of your restore jobs to troubleshoot any issues:

```
# Check all restores
velero get restore

# Describe a specific restore
velero describe restore <RESTORE_NAME>

# View restore logs
velero restore logs <RESTORE_NAME>
```

**Results**

You have restored the DevOps Loop namespace, including all CSI snapshots and fs-volume backups, and any external databases (if applicable).

# Private CA and self-signed certificate support

A private Certificate Authority (CA) is an organization-specific service that issues and manages digital certificates for use within an enterprise’s private network. These certificates secure communication between internal applications and services.

By default, self signed or privately signed certificates are not automatically trusted by clients because they are not included in the public trust stores used by operating systems and browsers.

DevOps Loop supports configuring and trusting such private CA or self-signed certificates so that secure connections to internal systems can be established.

During installation or upgrade, DevOps Loop reads CA certificates from a Kubernetes secret specified by the Helm parameter `global.privateCaBundleSecretName`. These certificates are then added to internal trust store of DevOps Loop so that integrations and plugins can establish encrypted, trusted connections to systems that use private CA or self-signed certificates.

You can also reference the same secret using `global.ibmCertSecretName` when using self-signed certificates to simplify configuration.

---

Related information

Configuring trusted certificates in DevOps Loop on page 62

## Configuring trusted certificates in DevOps Loop

You can configure DevOps Loop to use private CA and self-signed certificates by creating or updating a Kubernetes secret with a PEM certificate bundle.

**Before you begin**

You must have completed the following tasks:

- Ensured that you have the administrator privileges to the Kubernetes namespace where DevOps Loop will be installed.
- Ensured that the private CA is available in a PEM format (`.pem`).

**Note:** In the scenarios below, `mycacrt.pem` is used as a sample PEM file name for a private CA or combined certificate bundle. You must replace `mycacrt.pem` with the actual name and path of your PEM file.

Perform any of the following actions based on the scenario:

| Scenario | Action | Notes |
|---|---|---|
| No certificate provided (auto-generate self-signed) | Perform the following steps:<br>a. Set `SELF_SIGNED=true` in the DevOps Loop installation script.<br>The script generates `key.pem` and `cert.pem` valid for 365 days with SAN for `$DOMAIN`. | No manual secret creation is required. The certificate is generated automatically. |

| Scenario | Action | Notes |
|---|---|---|
| | The script creates a Kubernetes secret named in `devops-loop-tls-secret` containing `ca.crt=cert.pem`, `tls.crt=cert.pem`, and `tls.key=key.pem`.<br><br>The script also sets `global.hclCertSecretName=devops-loop-tls-secret`, so that it is used to terminate TLS for the DevOps Loop instance. | |
| Using a private CA certificate bundle | Perform the following steps:<br><br>a. Combine multiple CA/intermediate certificates if needed:<br>`cat rootCA1.pem rootCA2.pem > mycacrt.pem`<br>b. Create the secret:<br>`kubectl create secret generic privateca-secret --from-file=ca.crt=/path/to/mycacrt.pem -n devops-loop`<br>c. Edit the `ADDITIONAL_HELM_OPTIONS` variable in the DevOps Loop installation script to add:<br>`--set global.privateCaBundleSecretName=privateca-secret -set hcl-devops-prod.ingress.cert.selfSigned=true`<br><br>The helm value<br>`hcl-devops-prod.ingress.cert.selfSigned=true`<br>is needed to work-around an issue with the latest shipping version of DevOps Test when using TLS certificates signed by a private CA. | Used when internal services are signed by a private CA. The `ca.crt` key is mandatory. |
| Updating an existing CA or self-signed certificate | Perform the following steps:<br><br>a. To prevent the need to manually restart pods, create a new secret with the updated certificate in the secret:<br>`kubectl create secret generic privateca2-secret \ --from-file=ca.crt=/path/to/mycacrt.pem -n devops-loop` | |

| Scenario | Action | Notes |
|---|---|---|
| | b. Edit the `ADDITIONAL_HELM_OPTIONS` variable in the DevOps Loop installation script to add:<br>`--set global.privateCaBundleSecretName=privateca2-secret -set hcl-devops-prod.ingress.cert.selfSigned=true`<br>The helm value<br>`hcl-devops-prod.ingress.cert.selfSigned=true`<br>is needed to work around an issue with the latest shipping version of DevOps Test when using TLS certificates signed by a private CA. | |

**Results**

You have configured DevOps Loop to use the specified trusted certificates.

**What to do next**

You must run the DevOps Loop installation script. See Installation of DevOps Loop on page 30.

# Upgrading DevOps Loop

After you complete certain backup and pre-upgrade steps, you can upgrade DevOps Loop to a newer version using Helm.

**Before you begin**

You must have completed the following tasks:

- Performed a backup of DevOps Loop. For detailed steps, see Backup and restoration of DevOps Loop on page 58.
- Ensured that you have access to the Kubernetes cluster and Helm CLI.
- Verified that the Helm chart repository is reachable from the environment.

Perform the following steps to upgrade DevOps Loop:

1. Run the following commands to set environment variables for the upgrade:

```
HELM_NAME=devops-loop
NAMESPACE=devops-loop
VERSION=2.0.000
```

2. Run the following command to initialize the additional Helm options:

```
export ADDITIONAL_HELM_OPTIONS="--set option1=value1 --set option2=value2"
```

3. Run the following command to retrieve the existing Helm values:

```
helm get values ${HELM_NAME} --namespace ${NAMESPACE} > user_input_values.yaml
```

4. Retrieve the existing Helm values again to verify the configuration by running the following command:

```
ADDITIONAL_HELM_OPTIONS=${ADDITIONAL_HELM_OPTIONS:-}
helm get values ${HELM_NAME} --namespace ${NAMESPACE}
```

5. Run the following command to upgrade DevOps Loop:

```
ADDITIONAL_HELM_OPTIONS=${ADDITIONAL_HELM_OPTIONS:-}
helm get values ${HELM_NAME} --namespace ${NAMESPACE} > user_input_values.yaml
helm upgrade "${HELM_NAME}" oci://hclcr.io/devops-automation-helm/hcl-devops-loop --namespace
 ${NAMESPACE} --version=${VERSION} -f user_input_values.yaml ${ADDITIONAL_HELM_OPTIONS}
```

6. Verify that all DevOps Loop pods and components are running the following command:

```
kubectl get pods -n devops-loop
```

**Results**

You have upgraded DevOps Loop.

# Integrations in DevOps Loop

DevOps Loop provides a set of integrations to extend the platform's capabilities and connect your loops to external systems and intelligent tools. These integrations enable you to automate tasks, enhance productivity, and interact with loop resources from outside the DevOps Loop interface.

### AI provider integration

DevOps Loop supports AI-assisted features through external AI providers such as **OpenAI**, **Claude Desktop**, **Gemini**, **IBM watsonx**, and **Ollama**. When configured, these integrations enable Loop Genie to generate summaries, insights, and contextual information across work items, build results, deployments, and other supported capabilities.

### MCP server integration

The MCP server (Model Context Protocol Server) allows external MCP-compatible tools such as **VS Code (Copilot)** and **Claude Desktop** to connect directly to DevOps Loop. Through this integration, you can query, retrieve, and modify loop resources from supported external tools. For more information, see DevOps Loop MCP server on page 74.

## AI provider integration for Loop Genie - Tech Preview

**Disclaimer:**

This release contains access to the AI provider configurations for Loop Genie feature in HCL DevOps Loop as a Tech Preview. The Tech Preview is intended for you to view Loop Genie's capabilities by integrating with major LLM providers such as OpenAI, Claude Desktop, Gemini, IBM watsonx, and Ollama and provide your feedback to the

product team. You are permitted to use the information only for evaluation purposes and not for use in a production environment. HCL provides the information without obligation of support and "as is" without warranty of any kind.

You can configure AI providers such as **OpenAI**, **Claude Desktop**, **Gemini**, **IBM watsonx**, and **Ollama** with DevOps Loop. Configuring any one of these providers activates Loop Genie for your environment. Only one provider can be active at a time.

Loop Genie relies on external AI providers to power natural language understanding, multi-agent orchestration, and conversational task execution. By configuring an AI provider, you enable Loop Genie to process prompts, retrieve and summarize data, and perform supported actions across your DevOps Loop environment.

Loop Genie becomes available once at least one AI provider is configured and active. Switching providers immediately changes which LLM Loop Genie uses for processing prompts. All provider configurations remain saved even when disabled.

With at least one AI provider configured, Loop Genie can:

- Interpret natural language prompts
- Execute multi-step, multi-capability queries
- Retrieve data such as issues, repositories, branches, and commit history
- Perform supported automation tasks (For example: create work items, create branches, list repositories, edit files, manage pull requests)
- Produce structured, readable output
- Provide context-aware responses within a conversational thread.

You can manage integrations that are configured in HCL DevOps Loop by editing, deleting, enabling, or disabling them as required.

You can use the following options on the **Integrations** page:

| Actions | Description |
|---|---|
| Search for an integration | You can search for an integration that is added by entering the name in the **Search plugin** field when the list of integrations is lengthy. |
| Enable or disable a provider | You can enable or disable an AI provider by clicking the toggle button in the **Provider** column.<br><br>At any point in time, only one provider can be enabled. Other providers in the list are disabled automatically. |
| Edit an integration | You can modify the integration details by clicking  in the **Actions** column. |
| View an integration | You can view the integration details by clicking  in the **Actions** column. |

| Actions | Description |
| --- | --- |
| Delete an integration | You can delete an integration when it is no longer required by clicking  in the **Actions** column. |

## Prerequisites for setting up AI providers for Loop Genie

When you want to configure an AI provider such as OpenAI, IBM watsonx, Gemini, Claude or Ollama with HCL DevOps Loop, ensure that the prerequisites are met.

### Requirements for OpenAI integration

Ensure the following steps are completed before integrating OpenAI with DevOps Loop:

1. **Create an OpenAI account**: Sign up on OpenAI's platform and obtain API access.
2. **Generate an API key**: Navigate to the OpenAI dashboard and create a secret API key for authentication.
3. **Install OpenAI library**: Use Python or another programming language to install the OpenAI package.

### Requirements for Claude Desktop integration

Ensure the following steps are completed before integrating Claude Desktop with DevOps Loop:

1. **Create an Claude account**: Sign up on Claude's platform and obtain API access.
2. **Generate an API key**: Navigate to the Claude dashboard and create a secret API key for authentication.
3. **Install Claude library**: Use Python or another programming language to install the Claude Desktop AI package.

### Requirements for Gemini integration

Ensure the following steps are completed before integrating Gemini with DevOps Loop:

1. **Create an Gemini account**: Sign up on Gemini's platform and obtain API access.
2. **Generate an API key**: Navigate to the Gemini dashboard and create a secret API key for authentication.
3. **Install Gemini library**: Use Python or another programming language to install the Gemini package.

### Requirements for IBM watsonx integration

Ensure that you have the following details:

1. API Key
2. Project ID
3. Endpoint URL

### Requirements for Ollama integration

Ensure the following steps are completed before integrating Ollama with DevOps Loop:

1. **Install Ollama**: Download the installer and install it.

   You must have read and understood the system requirements for each model of Ollama. for more information, refer to https://ollama.com.

2. **Download a model**: Download the required large language model (LLM) such as Llama 2 or Mistral.
3. **Set up Python environment**: Create and activate a virtual environment and install dependencies.

   **Note:**

   If you are using the Ollama instance that is bundled with DevOps Loop, you must follow these steps:

   a. Enable the instance in the Helm chart by setting the value to `true` in the `values.yaml` file:

   ```
   llama:
       enabled: true
   ```

   b. Specify the required large language model (LLM) name in the Helm chart by providing the values in the `values.yaml` file:

   ```
   llama:
       ollama:
           models:
           pull:
               - <model name>
               - <model name>
           run:
               - <model name>
               - <model name>
   ```

   Where, the pull command is used to download a model, and the run command is used to run the model.

## Configuring OpenAI integration

You can configure the OpenAI integration in HCL DevOps Loop to connect to the provider and use Loop Genie services.

**Before you begin**

You must have completed the following tasks:

- Ensured that you have administrative privileges in DevOps Loop.
- Completed the steps to prepare your OpenAI account for integration with DevOps Loop. See Requirements for OpenAI integration on page 67.

1. Click **Settings > Integrations**.
   The **Integrations** page is displayed.
2. Click 
   
   .
   The **New Integration** pane is displayed with the **Overview** tab.
3. Enter a name to identify the integration in **Name**.

4. Select **Open AI** from the **Provider** list.
5. Select one of the following options in the **Credential Type** list:
    - API Key: Go to step 6 on page 69.
    - Password: Go to step 7 on page 69.
6. Enter the API key of the AI provider account to authenticate the connection in **API Key**.
7. Enter the details in the following fields:
    - Username
    - Password
    - Confirm Password
8. Click **Next**.

   The **Configure** tab is displayed.
9. Perform the following steps in the Configure tab to set the parameters:
    a. Enter the **API Endpoint** URL used by your OpenAI account.
    b. Select **Chat** in **Mode** to set the mode of communication.
    c. Select an LLM to be used from the **Model** list.
    d. Enter the **Organization ID** configured in your OpenAI account for your organization.
    e. Enter the **Project ID** configured in your OpenAI account for your organization.
    f. Enter one or more **Stop Sequences** to define where the model should stop generating text. For example: `"stop"` or `"end"`.
    g. Use the **Temperature** slider to control the creativity of the response:
        - Lower values produce more precise, consistent answers.
        - Higher values produce more creative or varied output.
    h. Set the **Max Length** slider to define the maximum number of tokens the model can generate.
    i. Use the **Top P** slider to adjust the diversity of the output:
        - Lower values focus on the most likely words.
        - Higher values allow more varied responses.
    j. Set the **Frequency Penalty** slider to reduce repetition of words or phrases.
        - Higher values result in less repetitive output.
    k. Set the **Presence Penalty** slider to encourage the model to introduce new topics rather than repeating previous ones.
    l. Set the **Best Of** slider to choose how many internal responses the model generates before returning the best one.
10. Click **Save** to apply the configuration.

**Results**

You have configured the OpenAI integration with DevOps Loop.

## Configuring Claude Desktop integration

You can configure the Claude Desktop integration in HCL DevOps Loop to enable Loop Genie services by connecting to the Claude Desktop provider.

**Before you begin**

You must have completed the following tasks:

- Ensured that you have administrative privileges in DevOps Loop.
- Completed the steps to prepare your Claude Desktop account for integration with DevOps Loop. See Prerequisites for setting up AI providers for Loop Genie on page 67.

1. Click **Settings > Integrations**.
   The **Integrations** page is displayed.
2. Click 

   .
   The **New Integration** pane is displayed with the **Overview** tab.
3. Enter a name to identify the integration in **Name**.
4. Select **Claude** from the **Provider** list.
5. Select one of the following options in the **Credential Type** list:
   - API Key: Go to step 6 on page 70.
   - Password: Go to step 7 on page 70.
6. Enter the API key of your Claude account to authenticate the connection in **API Key** and go to step 8 on page 70.
7. Enter the details in the following fields:
   - Username
   - Password
   - Confirm Password
8. Click **Next**.
   The **Configure** tab is displayed.
9. Perform the following steps in the **Configure** tab to set the parameters:
   a. Enter the **API Endpoint** URL used by your Claude account.
      - URL = https://api.anthropic.com
   b. Select **Chat** in **Mode** to set the mode of communication.
   c. Select an LLM to be used from the **Model** list.
   d. Enter the **Organization ID** configured in your Claude account for your organization.
   e. Enter the **Project ID** configured in your Claude account for your organization.
   f. Enter one or more **Stop Sequences** to define where the model should stop generating text. For example: `"stop"` or `"end"`.
   g. Use the **Temperature** slider to control the creativity of the response:
      - Lower values produce more precise, consistent answers.
      - Higher values produce more creative or varied output.

h. Set the **Max Length** slider to define the maximum number of tokens the model can generate.
i. Use the **Top P** slider to control the diversity of the generated output:
   - Lower values focus on the most likely words.
   - Higher values allow more varied responses.
j. Set the **Frequency Penalty** slider to reduce repetition of words or phrases.
   - Higher values result in less repetitive output.
k. Set the **Presence Penalty** slider to encourage the model to introduce new topics rather than repeating previous ones.
l. Set the **Best Of** slider to choose how many internal responses the model generates before returning the best one.

10. Click **Save** to apply the configuration.

**Results**

You have configured the Claude Desktop integration with DevOps Loop.

## Configuring Gemini integration

You can configure Gemini integration in HCL DevOps Loop to connect to the provider and use Loop Genie services.

**Before you begin**

You must have completed the following tasks:

- Ensured that you have administrative privileges in DevOps Loop.
- Completed the steps to prepare your Gemini account for integration with DevOps Loop. See Requirements for Gemini integration on page 67.

1. Click **Settings > Integrations**.
   The **Integrations** page is displayed.
2. Click 

   .
   The **New Integration** pane is displayed with the **Overview** tab.
3. Enter a name to identify the integration in **Name**.
4. Select **Gemini** from the **Provider** list.
5. Select one of the following options in the **Credential Type** list:
   - API Key: Go to step 6 on page 71.
   - Password: Go to step 7 on page 71.
6. Enter the API key of your Gemini account to authenticate the connection in **API Key** and go to step 8 on page 72.
   - Name = `Google_API_Key`
   - Value = `AIzaSyDySjxBpUlHqUhdIvh6VnXbxOYBJ5wout4`
7. Enter the details in the following fields:

- Username
- Password
- Confirm Password

8. Click **Next**.
   The **Configure** tab is displayed.
9. Perform the following steps in the **Configure** tab to set the parameters:
   a. Enter the **API Endpoint** URL used by your Gemini account.
      - URL = https://generativelanguage.googleapis.com
   b. Select **Chat** in **Mode** to set the mode of communication.
   c. Select an LLM to be used from the **Model** list.
   d. Enter the **Organization ID** configured in your Gemini account for your organization.
   e. Enter the **Project ID** configured in your Gemini account for your organization.
   f. Enter one or more **Stop Sequences** to define where the model should stop generating text. For example: `"stop"` or `"end"`.
   g. Use the **Temperature** slider to control the creativity of the response:
      - Lower values produce more precise, consistent answers.
      - Higher values produce more creative or varied output.
   h. Set the **Max Length** slider to define the maximum number of tokens the model can generate.
   i. Use the **Top P** slider to adjust the diversity of the output:
      - Lower values focus on the most likely words.
      - Higher values allow more varied responses.
   j. Set the **Frequency Penalty** slider to reduce repetition of words or phrases.
      - Higher values result in less repetitive output.
   k. Set the **Presence Penalty** slider to encourage the model to introduce new topics rather than repeating previous ones.
   l. Set the **Best Of** slider to choose how many internal responses the model generates before returning the best one.
10. Click **Save** to apply the configuration.

**Results**

You have configured the Gemini integration with DevOps Loop.

## Configuring IBM watsonx integration

You can configure the IBM watsonx integration in HCL DevOps Loop to connect to the provider and use Loop Genie services.

**Before you begin**

You must have completed the following tasks:

- Ensured that you have administrative privileges in DevOps Loop.
- Ensured that you have details such as API Key, Project ID and Endpoint URL.

1. Navigate to **Settings** > **Integrations**.

   The **Integrations** page is displayed.

2. Click 

.

   The **New Integration** pane is displayed with the **Overview** and **Configure** tabs.

3. Perform the following steps in the **Overview** tab:

   a. Enter a name to identify the integration in the **Name** field.

   b. Select **Watson X** in the **Provider** list.

   c. Select the API key in the **Credentials** list.

   d. Enter the API Key in the **API Key** field.

   e. Click **Next**.

      The **Configure** tab is displayed.

4. Perform the following steps in the **Configure** tab:

   a. Enter the API Endpoint URL in the **API Endpoint** field.

   b. Enter the release version of the IBM watsonx.ai model in the **Version** field to specify the model to which DevOps Loop should connect.

   c. Select a **Model** from the list.

   d. Enter the project ID in the **Project ID** field.

   e. Enter the string to be used to stop the generation of text by AI in the **Stop Sequences** field.

      For example, you can set "stop" or "end" as the stop sequence.

   f. Select a **Decoding** method from the list to control how the model generates output.
      The following options are available:
      - **Greedy:** Produces the most likely next token at each step for straightforward output.
      - **Sample:** Randomly selects from probable next tokens, resulting in more varied and creative responses.

   g. Use the **Repetition Penalty** slider to control how much the model discourages repeating the same tokens in its output:

For example: A higher value, for example, 1.5, makes more different responses. A lower value, for example, 1.0, allows natural repetition of words.

h. Specify the minimum number of tokens in the **Min Token** for the response.

i. Specify the maximum number of tokens in the **Max Token** for the response.

j. Click **Save**.

**Results**

You have configured the IBM watsonx integration with DevOps Loop.

## Configuring Ollama integration

You can configure the Ollama integration in HCL DevOps Loop to connect to the provider and use Loop Genie services.

**Before you begin**

You must have completed the following tasks:

- Ensured that you have administrative privileges in DevOps Loop.
- Completed the steps to prepare your Ollama account for integration with DevOps Loop. See Requirements for Ollama integration on page 67.

1. Click **Settings > Integrations**.
   The **Integrations** page is displayed.
2. Click 

   .
   The **New Integration** pane is displayed with the **Overview** tab.
3. Enter a name to identify the integration in **Name**.
4. Select **Ollama** from the **Provider** list.
5. Click **Next**.
   The **Configure** tab is displayed.
6. Enter the URL of the AI provider in **API Endpoint**.
7. Select **Chat** in **Mode** to set the mode of communication.
8. Select an LLM to be used from the **Model** list.
9. Click **Save**.

**Results**

You have configured Ollama integration with DevOps Loop.

# DevOps Loop MCP server

The DevOps Loop MCP (Model Context Protocol) server provides a standardized interface that allows external MCP clients to communicate with a loop. Through this server, MCP-compatible clients such as Visual Studio Code (via MCP

extensions) and Claude Desktop can query, retrieve, and modify loop resources from within their development or AI-assisted environments.

The MCP server enables secure, controlled access to DevOps Loop resources without requiring users to switch interfaces. The MCP server enforces a 30-minute idle timeout. After 30 minutes of inactivity, your session expires and you might need to sign in again to continue using MCP tools. After the initial setup, MCP client configurations are stored persistently, allowing external tools to reconnect automatically in future sessions.

By offering well-defined MCP tools, the server allows clients to run queries, fetch loop details, and invoke supported operations from external MCP clients while fully adhering to your DevOps Loop access permissions.

### MCP tools available in DevOps Loop

After an MCP client is connected to DevOps Loop, it can access a set of tools offered by DevOps Loop. These tools provide controlled access to loop resources across loop server, Plan, tenant server, and analytics services. Each tool specifies its required inputs and expected output.

**Table 1. Loop-related tools**

| MCP Tools | Description | Inputs | Output |
|---|---|---|---|
| **loop_get_all_loops** | Fetch all loops in a teamspace. | teamspace_id | List of loops that you are a member of |
| **loop_get_all_loop_users** | Fetch all users in a loop. | loop_id, teamspace_id | List of users who are part of the loop |
| **loop_add_user_to_loop** | Add a user to a loop. | loop_id, teamspace_id, user (email) | Confirmation that the user was added to the loop |
| **loop_add_test_project_to_loop** | Add a test project to a loop. | teamspace_id, loop_id, project_name, project_description | Details of the test project created in the loop |
| **loop_add_control_repo_to_loop** | Add a control repository to a loop. | teamspace_id, loop_id, repo_name | Details of the control repository added to the loop |

**Table 2. Plan-related tools**

| Tools | Description | Inputs | Output |
|---|---|---|---|
| **plan_get_projects** | List all projects in DevOps Plan for a loop. | teamspace_id, loop_name | List of projects in the loop |
| **plan_get_components** | List components in DevOps Plan for a loop. | teamspace_id, loop_name | List of components in the loop |

**Table 2. Plan-related tools (continued)**

| Tools | Description | Inputs | Output |
|---|---|---|---|
| **plan_get_workitem_types** | Retrieve available work item types for a loop. | teamspace_id, loop_name | List of available work item types |
| **plan_get_work_items** | Retrieve work items with optional filters. | teamspace_id, loop_name, project_name, filters | List of work items matching the filters |
| **plan_get_applications** | List applications in a teamspace. | teamspace_id, loop_name | List of applications in the teamspace |
| **plan_create_work_item** | Create a new work item. | teamspace_id, loop_name, title, project_name, optional fields | Details of the created work item |
| **plan_delete_work_item** | Delete a work item by ID. | teamspace_id, loop_name | Confirmation that the work item was deleted |

**Table 3. Loop tenant-related tools**

| Tools | Description | Inputs | Output |
|---|---|---|---|
| **loop_get_all_tenants** | Fetch all tenants. | None | List of all tenants |
| **loop_get_all_teamspace_users** | Fetch all users in a teamspace. | teamspace_id | List of users in the teamspace |
| **loop_get_all_teamspaces** | Fetch all teamspaces. | None | List of all teamspaces |
| **loop_invite_user_to_teamspace** | Invite a user to a teamspace. | teamspace_id, user (email) | Confirmation that the user was invited to the teamspace |

**Table 4. Deploy-related tools**

| Tools | Description | Inputs | Output |
|---|---|---|---|
| **list_applications** | Fetch all applications. | None | Lists all applications. |
| **list_application_processes** | Fetch all processes defined in a specified application. | Application_id, application name | Lists all processes defined in a specified application. |
| **list_application_environments** | Fetch all environments defined in a specified application. | Application_id or application name | Lists all environments defined in a specified application |
| **list_application_snapshots** | Fetch all snapshots defined in a specified application. | Application_id or application name | Lists all snapshots defined in a specified application. |
| **get_deployment_status** | Fetch all the deployment status of an application | None | Returns the deployment status of an application |
| **deploy_snapshot_to_environment** | Deploy an application snapshot to a specified environment. | Application_id, snapshot_id, process_id, and environment_id | Deploys an application snapshot to a specified environment. |

**Table 5. Loop analytics-related tools**

| Command | Description | Inputs | Output |
|---|---|---|---|
| **analytics_search_work_items** | General search for work items with filtering. | teamspace_id, loop_id, optional filters | List of work items matching the filters |
| **analytics_get_work_item_status** | Retrieve status for work items. | teamspace_id, loop_id, filters | List of work item statuses |
| **analytics_count_work_items** | Count work items based on filters. | teamspace_id, loop_id, filters | Count of work items matching the filters |
| **analytics_get_story_points** | Retrieve story point analytics. | teamspace_id, loop_id, owner, date filters | Story point analytics data |
| **analytics_get_recent_and_old_work_items** | Retrieve recent or oldest work items. | teamspace_id, loop_id, optional filters | List of work items ordered by time |
| **analytics_search_commits** | Search commits by creator, date, repo, etc. | teamspace_id, loop_id, filters | List of commits matching the filters |
| **analytics_search_pull_requests** | Attribute-based PR search. | teamspace_id, loop_id, filters | List of pull requests matching the filters |
| **analytics_count_pull_requests** | Count PRs by creator, repo, status, or month. | teamspace_id, loop_id, filters | Count of pull requests matching the filters |
| **analytics_get_recent_and_old_pull_requests** | Retrieve time-ordered PRs. | teamspace_id, loop_id, filters | List of pull requests ordered by time |
| **analytics_search_builds** | General search for builds. | teamspace_id, loop_id, filters | List of builds matching the filters |
| **analytics_count_builds** | Count builds by creator, status, or month. | teamspace_id, loop_id, filters | Count of builds matching the filters |
| **analytics_get_recent_and_old_builds** | Retrieve recent or oldest builds. | teamspace_id, loop_id, filters | List of builds ordered by time |
| **analytics_search_deployments** | Search deployments with filters. | teamspace_id, loop_id, filters | List of deployments matching the filters |

**Table 5. Loop analytics-related tools (continued)**

| Command | Description | Inputs | Output |
|---|---|---|---|
| **analytics_count_deployments** | Count deployments by status, creator, or month. | teamspace_id, loop_id, filters | Count of deployments matching the filters |
| **analytics_get_recent_and_old_deployments** | Retrieve recent or oldest deployments. | teamspace_id, loop_id, filters | List of deployments ordered by time |

## VS Code connection for the DevOps Loop MCP server

You can connect **Visual Studio Code** (VS Code) with GitHub Copilot to the DevOps Loop MCP server to run MCP-based queries that retrieve and update loop resources directly from VS Code. After you connect the VS Code, the MCP server connection is saved and remains available across VS Code sessions.

You can connect VS Code to the DevOps Loop MCP server by using one of the following methods:

- By using the command palette. See Connecting VS Code to the DevOps Loop MCP server by using the command palette on page 78.
- By editing the MCP configuration file. See Connecting VS Code to the DevOps Loop MCP server by manually editing the MCP configuration file on page 79.

## Connecting VS Code to the DevOps Loop MCP server by using the command palette

You can use the command palette to connect VS Code to the DevOps Loop MCP server.

**Before you begin**

You must have completed the following tasks:

- Ensured that you have installed VS Code and the GitHub Copilot extension.
- Obtained the MCP Server URL. For example: `https://hostname/loop/mcp`

Perform the following steps by using the command palette:

1. Open **VS Code**.
2. Open the **Command Palette** (Ctrl+Shift+P or Cmd+Shift+P).
3. Search and select **Copilot: Add Server**.
4. Choose **HTTP** as the server type, when prompted for the transport type.
   The DevOps Loop MCP Server uses HTTP-based transport.
5. Enter the **MCP Server URL**, in the **URL** field.
   VS Code automatically generates the required `mcp.json` configuration.
6. Enter a name for the server. For example: Loop MCP Server.
7. Select **Workspace**, when VS Code prompts you to choose where to save this configuration.

- **Global** – used for all VS Code workspaces
- **Workspace** – used only in the current folder/project

**Note:** Selecting **Workspace** automatically creates a `.vscode` folder and generates a `mcp.json` file inside it.

8. Click the **Extensions** icon  and navigate to **MCP-servers installed**.
9. Click the **Settings** icon  next to Loop MCP server.
10. Select the **Start server** option.

    VS Code displays a notification stating that the MCP server requires authentication.
11. Click **Allow** to proceed.
    A second notification is displayed asking whether you want to open an external website.
12. Click **Open**.

    Your default browser opens and displays the DevOps Loop login page.
13. Sign in with your DevOps Loop credentials.
    A consent screen is displayed requesting access to MCP tools, loop APIs, and basic profile information.
14. Click **Yes** to approve and complete the connection.
    After successful authentication, VS Code displays the list of available MCP tools. See MCP tools available in DevOps Loop on page 75.

**Note:** The MCP server times out after 30 minutes of inactivity. If no tool calls occur during this period, your session will expire, and you are prompted to re-authenticate.

**Results**

You have successfully connected VS Code to the DevOps Loop MCP server. The MCP server is displayed under **Copilot → Servers**.

## Connecting VS Code to the DevOps Loop MCP server by manually editing the MCP configuration file

You can manually connect VS Code to the DevOps Loop MCP server by creating or editing the `mcp.json` configuration file in your workspace.

**Before you begin**

You must have completed the following tasks:

- Ensured that you have installed VS Code and the GitHub Copilot extension.
- Obtained the MCP Server URL. For example: `https://hostname/loop/mcp`

Perform the following steps to manually modify the MCP configuration file:

1. Open **VS Code**.
2. Open any empty directory or project in VS Code.
3. Create a new folder named `.vscode` inside that directory, if not present.
4. Create a file named `mcp.json` inside the `.vscode` folder.
5. Paste the MCP configuration JSON. For example:

   ```
   {
     "servers": {
       "loop-mcp": {
         "uri": "https://hostname/loop/mcp",
         "type": "http"
       }
     },
     "inputs": []
   }
   ```

   Replace the `hostname` with your actual DevOps Loop server hostname.
6. **Save** the file.
7. Click **Extensions** icon  and navigate to **MCP-servers installed**.
8. Click the **Settings** icon  next to Loop MCP Server.
9. Select the **Start server** option.

   VS Code displays a notification stating that the MCP server requires authentication.
10. Click **Allow**.

    A second notification is displayed asking whether you want to open an external website.
11. Click **Open**.
    Your default browser opens and displays the DevOps Loop login page.
12. Sign in with your DevOps Loop credentials.

    A consent screen is displayed requesting access to MCP tools, loop APIs, and basic profile information.
13. Click **Yes** to approve and complete the connection.

    After successful authentication, VS Code displays the list of available MCP tools. See MCP tools available in DevOps Loop on page 75.

    **Note:** The MCP server times out after 30 minutes of inactivity. If no tool calls occur during this period, your session will expire and you are prompted to re-authenticate.

**Results**

You have successfully connected VS Code to the DevOps Loop MCP server. The MCP server is displayed under **Copilot → Servers**.

## Connecting Claude Desktop to the DevOps Loop MCP server

You can connect Claude Desktop to the DevOps Loop MCP Server and use it as a connector to retrieve and update loop resources.

**Before you begin**

You must have completed the following tasks:

- Ensured that Claude Desktop is installed and you have an active **Claude Pro subscription** (minimum).
- Obtained the MCP Server URL. For example: `https://hostname/loop/mcp`

1. Open the **Claude Desktop** application.
2. Navigate to **Settings > Connectors**.
3. Click **Add custom connector**.
   The **Add custom connector** window is displayed.
4. Enter the name of the connector. For example: Loop MCP server
5. Enter the MCP server URL in the **Remote MCP server URL** field.
6. Click **Add**.
   The MCP server is now added and listed on the **Connectors** page.
7. Click the **Connect** button next to the newly added Loop MCP server entry.

   Claude Desktop redirects you to the **DevOps Loop** login page.
8. Sign in with your DevOps Loop credentials.

   A consent screen is displayed requesting access to MCP tools, loop APIs (used internally for tool execution), and basic profile and role information.
9. Click **Yes** to approve and complete the connection.

   After successful authentication, Claude Desktop displays the list of available MCP tools provided by the DevOps Loop MCP Server. See MCP tools available in DevOps Loop on page 75.

   **Note:** The MCP server times out after 30 minutes of inactivity. If no tool calls occur during this period, your session will expire and you will be prompted to re-authenticate.

**Results**

You have successfully connected Claude Desktop to the DevOps Loop MCP Server.

# Teamspace management

A teamspace in HCL DevOps Loop is a secure, dedicated environment designed for multiple teams to work independently within a single shared DevOps Loop installation.

When you create a teamspace in DevOps Loop, the platform automatically provisions corresponding components for all integrated applications. Each component is provisioned with a corresponding name that reflects the Teamspace, ensuring consistency and seamless integration across all the applications.

The following table lists the configuration of each integrated application within the teamspace provisioning:

| Application | Description |
|---|---|
| Plan | A new teamspace is created. |
| Control | An organization is created. |
| Code | A teamspace is created. |
| Deploy | A team is created. |
| Test | A teamspace is created. |
| Measure | A teamspace and a default team are created. |

You can manage and switch between multiple teamspaces by clicking the **Switch Teamspace** icon  . The following options are available to help you create and manage team spaces in DevOps Loop:

| Action | Icon | Description |
|---|---|---|
| Create a teamspace |  | Select **Create TeamSpace** to create a team space within DevOps Loop. |
| Invite members to a teamspace |  | Select **Edit Teamspace** to add members to the teamspace. |

These configurations provide your team with a fully integrated DevOps environment, streamlining collaboration, automation, and traceability across the software delivery lifecycle.

Related information

Creating a teamspace on page 82

Adding or inviting members to a teamspace on page 84

Joining a teamspace as an invited member on page 84

## Creating a teamspace

You can create a teamspace in HCL DevOps Loop to provide your team with a dedicated, secure workspace that automatically spans across all the capabilities like Plan, Control, Code, Build, Test, Deploy, and Measure.

**Before you begin**

Ensure that the following prerequisites are completed:

- You are added to the **User** group in **Keycloak**.
- Ensure that you have administrative privileges in DevOps Loop.

1. Log in to DevOps Loop.
2. Click the **Create TeamSpace** button.

   Alternatively, select the **Switch Teamspace** icon  and click **+ Create TeamSpace**.

   **Result**

   The Teamspace creation page is displayed along with the **Define Teamspace** and **Invite Members** tabs.
3. Perform the following steps in the **Define Teamspace** tab.
   a. Provide a name in **Enter Teamspace Name**. The maximum character limit is 20.
   b. **Optional**: Provide a brief description of the teamspace in **Description**.
4. Click **Next**.

   **Result**

   The **Invite Members** tab is displayed.
5. Perform the following steps in the **Invite Members** tab.
   a. Search members with a valid email address or select them from the dropdown list and click **Add**.
   b. Type the email address of the members who are not in the Keycloak user directory and click **Add**.
6. Click **Create**.

   The Teamspace setup progress page is displayed, showing each phase of the setup. DevOps Loop begins provisioning the TeamSpace across all integrated capabilities.

   **Note:** A Control repository is added to Build, and a webhook is set up in Control to synchronize data between Control and Build.

   - If the members already exist in the Keycloak user directory, they are directly added as members in the teamspace.
   - If the members do not exist in the Keycloak user directory, an invitation email is automatically triggered after the teamspace is created.

     **Note:** You can add multiple members to your invite list.

   If an error occurs during teamspace creation, an error message is displayed with options to **Retry** or **Cancel and retry later**. Click **Retry** to attempt the operation again. If the problem persists, check the tenant server pod logs and the logs for the associated product components to obtain additional information.

   **Note:** The error badge  indicates that the teamspace setup is incomplete or has configuration issues.

   For a visual walkthrough of creating a teamspace watch the following video:
   ../videos/creating%20a%20teamspace.mp4

**Results**

You have created a teamspace, which is provisioned across all the capabilities of DevOps Loop. Additionally, the loop creation is enabled.

---

Related information

Adding or inviting members to a teamspace on page 84

Joining a teamspace as an invited member on page 84

Removing a member from a teamspace on page 85

## Adding or inviting members to a teamspace

You can add existing members or invite new members to join a teamspace in DevOps Loop by using the **Edit Teamspace** option.

**Before you begin**

- You must be a teamspace owner.

1. Log in to DevOps Loop.
2. Select the **Switch TeamSpace** icon  and select the teamspace in which you want to add members.
3. Click **Edit Teamspace**.
   **Result**
   The **Add members** pane is displayed.
4. Click **Add members**.
5. Perform the following steps in the **Invite Members** tab.
   a. Search members with a valid email address or select them from the dropdown list and click **Add**.
   b. Type the email address of the members who are not in the Keycloak user directory and click **Add**.
6. Click **Save**.
   If the members already exist in the Keycloak user directory, they are directly added to the corresponding teamspaces across all integrated applications.
   If the members do not exist in the Keycloak user directory, an invitation email is automatically triggered to join the teamspace.

**Results**

You have added members to your teamspace. If the members are not part of DevOps Loop, an invitation email is sent to them to join the teamspace.

## Joining a teamspace as an invited member

You can join a teamspace in DevOps Loop when you receive an invitation from a teamspace owner. The invitation email includes a link to update your account and access the DevOps Loop.

**Before you begin**

You must have completed the following tasks:

- Confirmed that you have received an invitation email from DevOps Loop.
- Ensured that you are not signed in with another account in the same browser session.

1. Open the invitation email sent by DevOps Loop.
2. Click the **Link to account update** link in the email.
   The account update page is displayed in a new browser tab.

   **Note:** If you are already signed in with another DevOps Loop account in your default browser, open the link in a different browser or in an incognito or private window in your default browser.

3. Click **Click here to proceed**.
4. Type **New password** and **Confirm password** and click **Submit**.
5. Type the **Email id**, **First name**, and **Last name** on the account setup page.
6. Click **Submit**.
   You are added to the invited teamspace.

---

Related information

Teamspace management on page 81

## Removing a member from a teamspace

You can remove a member from a teamspace when they no longer need access.

**Before you begin**

You must have completed the following tasks:

- Ensured that you have administrative privileges in DevOps Loop.
- Ensured that the member is removed from all associated loops before removing from the teamspace.

1. Log in to DevOps Loop.
2. Click the **Switch TeamSpace** icon and select the teamspace from which you want to remove a member.
3. Click **Edit Teamspace**.

   The **Teamspace Members** pane is displayed.

4. Click the Delete icon next to the member you want to remove in the Members List section.
5. Click **Save**.

**Results**

The member is removed from the teamspace.

Related information

Adding or inviting members to a teamspace on page 84

# Uninstalling DevOps Loop

You can safely uninstall DevOps Loop from your environment, ensuring that all resources and data are properly removed.

**Note:** For demo setups and air-gapped environment, the follow the same Kubernetes service uninstall procedures used for your platform.

## Uninstalling DevOps Loop from IBM Cloud Kubernetes Service (IKS)

To reinstall HCL DevOps Loop when an ongoing installation fails, you can uninstall DevOps Loop and its components from theIBM Cloud Kubernetes Service (IKS) cluster.

**Before you begin**

You must have completed the following tasks:

- Installed DevOps Loop.
- Closed DevOps Loop, any open web browsers, and all other applications that are enabled by DevOps Loop.
- **Optional**: Backed up data from the previous version of DevOps Loop.

1. Navigate to **Cluster Management > Clusters > Overview > Actions > Connect via CLI** in your IBM Cloud account and switch context to your cluster.
2. Run the following command to uninstall DevOps Loop:

   ```
   helm uninstall $HELM_NAME -n $NAMESPACE
   ```

   Where,

   ```
   NAMESPACE=devops-loop
   HELM_NAME=devops-loop
   ```

   The PersistentVolumeClaims and PersistentVolumes that were created during the installation are not deleted automatically. If you reinstall DevOps Loop, the user data is reused unless you specifically delete those volumes.
3. Run the following command to delete everything, including the user data contained in claims and persistent volumes:

   ```
   kubectl delete namespace $NAMESPACE
   ```

**Results**

You have uninstalled DevOps Loop from the IKS cluster.

## Uninstalling DevOps Loop from Kubernetes Service (K8S)

To reinstall HCL DevOps Loop when an ongoing installation fails, you can uninstall DevOps Loop and its components from the Kubernetes Service (K8S) cluster.

**Before you begin**

You must have completed the following tasks:

- Installed DevOps Loop.
- Closed DevOps Loop, any open web browsers, and all other applications that are enabled by DevOps Loop.
- **Optional**: Backed up data from the previous version of DevOps Loop.

1. Switch to your K8S cluster by using the following command:

   ```
   kubectl config use-context <context-name>
   ```

2. Run the following command to uninstall DevOps Loop:

   ```
   helm uninstall $HELM_NAME -n $NAMESPACE
   ```

   Where,

   ```
   NAMESPACE=devops-loop
   HELM_NAME=devops-loop
   ```

   The PersistentVolumeClaims and PersistentVolumes that were created during the installation are not deleted automatically. If you reinstall DevOps Loop, the user data is reused unless you specifically delete those volumes.

3. Run the following command to delete everything, including the user data contained in claims and persistent volumes:

   ```
   kubectl delete namespace $NAMESPACE
   ```

**Results**

You have uninstalled DevOps Loop from the K8S cluster.

# Chapter 5. Working with Loops

A loop is a workflow that connects DevOps activities of your team from planning and development through build, deployment, testing, and release phases.

You can find information about creating and managing loops, learning loop, managing members, and using dashboards to monitor progress across all capabilities of DevOps Loop.

## Loop management

The loop consists of several interconnected phases that ensure continuous software development, testing, deployment, and monitoring.

Loop creation ensures seamless integration between development and operations, leading to faster and more reliable DevOps workflow. The following changes take place as a result of loop creation:

- Provisions your planning, source control, testing, deployment, and metrics tools for a ready-to-use project environment.
- Connects all capabilities by installing and configuring essential plugins, webhooks, repositories, and integrations for real-time data sharing and visibility.
- Provides access to a unified dashboard and links for direct access to each application.
- Adds users in each capability and sets specific access rights across all applications.

Overall, loop creation reduces the time and manual effort required to configure and maintain your development toolchain so that you can focus on innovation.

You can explore loop capabilities by enabling Learning Loop option. This allows you to learn and practice workflows by working with sample projects, work items, and configurations without starting from an empty setup. See Learning Loop overview on page 97.

**Managing loops**

You can use the following options on the **Home** page:

| Actions | Description |
|---|---|
| View all the loops | The **All loops** tab lists all the loops within your teamspace. |
| View the recently added loops | The **Recently Added** tab lists the recently added loops within your teamspace. |
| View My favorites | The **My Favorites** tab lists the loops that are marked as favorite within your teamspace. |
| View Disabled loops | The **Disabled** tab lists the loops that are disabled within your teamspace. |
| View the incomplete loops | The **Incomplete** tab lists all the incomplete loops within your teamspace. You can click **Retry** to complete the creation of the loop. |

| Actions | Description |
|---|---|
| Search loops | You can search for a loop by clicking the  icon and entering the name in the **Search Loops** field when the list of loops is lengthy. |
| Create a loop | You can create a new loop by clicking the **Create Loop** button. |
| Edit a loop | You can modify the list of team members who can access the loop by clicking the **Settings** icon  . |
| Mark as favorite | You can mark a loop as a favorite to make it easier to find by clicking the **Add to favorites** icon  . |
| Open Loop Genie | You can interact with Loop Genie and send your queries related to the loop by clicking the  button. |
| View dashboards | You can view the overview and detailed dashboards for the loop by clicking |
| View loop details | You can view loop details such as configuration details and its linked resources, such as test projects and control repositories, by clicking the **Loop Details** option on the loop. |

## Creating a loop

You can create a loop to build a pipeline of tasks across the integrated capabilities in HCL DevOps Loop.

**Before you begin**

Ensure that the following prerequisites are completed:

- You are added to the **User** group in **Keycloak**.
- Created a teamspace or a member of an existing teamspace.
- Read and understood the information in About user roles and access permissions on page 57.

**About this task**

When you create a loop, DevOps Loop creates specific resources in each of the applications and also, installs and configures the required plugins. During the loop creation, when the plugins are installed and configured, personal access tokens are created in **Plan**, **Control**, and **Test**. On loop creation, the respective users are added to the specific resources in each of the capabilities.

1. Click **Teamspace > <teamspace_name>** to navigate to your teamspace.
2. Click **Create Loop** on the home page of your teamspace.
   The **Define Loop** tab is displayed.
3. Perform the following actions in the **Define Loop** tab:

a. Enter a loop name.

> **Note:** The first five characters for the loop name must always be unique, and they should not be reused while creating another loop.

b. **Optional**: Enter details about the loop in **Description**.
c. **Optional**: Enable the **Learning Loop** option to prepopulate the loop with demo data for exploring workflows and practicing tasks.

4. Click **Next**.
   The **Invite Members** tab is displayed.
5. Perform the following actions in **Invite Members** tab.
   a. Enter the email ID of the team member to be added to the loop in **Search members**.
   b. Click **Add**.
6. Click **Create**.
   The loop setup process begins. A progress view displays the status of each capability being initialized. Once all the capabilities are fully initialized, the loop is successfully created.
   For details about the capabilities and resources configured during loop creation, see Capabilities and resources configured during loop creation on page 91.

   **What to do next**: After the loop is created, DevOps Loop automatically provisions the required resources and integrations across all capabilities. You can begin using these resources based on your workflow needs.

   The following examples show common actions you may perform after creating a loop. These are **optional** and depend on the tasks you want to complete within the loop:
   - View the Dashboard to see summaries and statuses of the activities through visualizations. See Dashboards and insights on page 99.
   - Create components and applications in DevOps Deploy. See Creating components in Deploy for a loop and Creating applications in Deploy for a loop.
   - For Control integration, create a file named `input.txt` under the repository in Control.

   For a visual walkthrough of creating a loop watch the following video:
   ../videos/creating%20a%20loop.mp4

---

Related reference

REST commands for Deploy on page 148

Related information

Loop creation failure at the Measure stage

## Capabilities and resources configured during loop creation

When you create a loop, the platform automatically provisions resources, installs and configures plugins, generates tokens, establishes integrations, and assigns appropriate permissions to the loop creator and invited members across multiple capabilities.

### Capability-specific resources and configurations

| Capability or Plugin | Configuration during loop creation |
|---|---|
| Plan | • Creates an application in the same teamspace where the loop is created.<br>• Sets up email notifications for work item changes.<br>• Names the application using the first five characters of the loop name. |
| Control | • Creates a repository with the same name as the loop in the organization associated with the teamspace. |
| Code | • No resources are created at this stage. |
| Build | • Creates a team with the following convention: `<TeamspaceName~LoopName>`.<br>• Creates the following templates:<br>  ◦ Project<br>  ◦ Process<br>  ◦ Source<br>  ◦ Jobs<br>• Creates a job template with basic CI/CD steps.<br>• Creates the **Deploy** and **Control** integrations.<br>• Creates the main project with the naming convention **TeamSpace~Loop:LoopName** (The UI displays this convention as **TeamSpace~Loop**) and configures it using the Process and Source templates.<br>  **Note:** A project is associated with only one loop. |
| Test | Creates a project with the same name as the loop within the teamspace. |

| Capability or Plugin | Configuration during loop creation |
|---|---|
| Deploy | • Creates an application with the same name as the loop and a team named <TeamspaceName~LoopName>.<br>• Creates three empty environments: Development, QA, and Production.<br>• Creates a template pipeline and an empty application process. |
| Plan Plugin | • Installs and configures the Plan plugin in Measure.<br>• Installation occurs once; configuration occurs with each loop.<br>• Generates a personal access token in Plan which is used to configure the Plan Plugin for Plan-Measure connection.<br>• Auto-synchronizes every 5 minutes.<br>• Default version: 1.2.3 |
| Build Plugin | • Installs and configures the Build plugin in Measure.<br>• Installation occurs once; configuration occurs with each loop.<br>• Auto-synchronizes every 5 minutes.<br>• Default version: 1.1.5 |
| Control Plugin | • Installs and configures the Control plugin in Measure.<br>• Installation occurs once; configuration occurs with each loop.<br>• Generates a personal access token in Control which is used to configure the Control Plugin for Control-Measure connection.<br>• Auto-synchronizes every 5 minutes.<br>• Default version: 1.1.1 |
| Deploy Plugin | • Installs and configures the Deploy plugin in Measure during the first loop creation.<br>• Generates a personal access token in Deploy which is used to configure the Deploy Plugin for Deploy-Measure connection.<br>• Auto-synchronizes every 5 minutes.<br>• Default version: 5.1.1 |
| Test Plugin | • Installs and configures the Test plugin in Measure.<br>• Installation occurs once; configuration occurs with each loop. |

| Capability or Plugin | Configuration during loop creation |
|---|---|
| | • Generates an offline user token in Test which is used to configure the Test Plugin for Test-Measure connection.<br>• No auto-synchronization.<br>• Default version: 1.0.32 |
| Control–Plan Integration | • Links repository changes (pull requests, commits, branch creation) in Control to work items in Plan.<br>• Pull request titles and branch names should start with the work item ID for proper syncing.<br>• Changes appear in the SCM Event section of the work item in Plan.<br>• A personal access token in Plan configures a webhook in Control. |
| Test–Plan Integration | • Creates work items in Plan from Test Hub via integration.<br>• A personal access token in Plan is used for setup.<br>• Verify by opening your Test Hub project and navigating to Manage Integration > Change Management. |
| Test–Control Integration | • Maps the repository created in Control to the Test Hub project associated with the loop.<br>• View it in Test Hub under Manage Configuration > Repositories.<br>• Enables Test Hub to pull test assets from Control. |
| Measure | • Creates a team <TeamspaceName~LoopName Team> and adds the creator/admin with full permissions.<br>• Creates a value stream and pipeline associated with the team, using application data from Deploy by using the YAML plugin in Measure. |
| Analytics | • Creates dashboards in Opensearch for the newly created loop. |

### Access and Permissions

The loop creator is granted the highest permissions for all resources in the integrated capabilities. Invited users are given only basic permissions by default when they join a loop. They can manually elevate their own permissions through the UI for any resource as needed.

## Viewing loop details

You can use the **Loop Details** page to view a complete overview of a loop, including its key configuration and resource information.

When you want to view a loop's configuration details and its linked resources, such as test projects and control repositories, select the **Loop Details** option on the loop. Each resource includes metadata such as its name, URL, and creation date, making it easy to understand how the loop is structured and which assets it relies on.

From the **Loop Details** page, you can add additional test projects and control repositories as needed, so that you can expand or update the loop's configuration without leaving the page.

The page also includes a team members section. Here, you can view all the members who are part of the loop, along with their role (Admin or Team Member), their email ID, and the other loops they belong to. You can search through the list of members or make updates using the available edit capabilities, helping you keep membership aligned with your project requirements.

---

Related information

Adding members to a loop on page 94

## Adding members to a loop

You can add members to a loop who are already part of your teamspace so they can collaborate and stay updated within it.

**Before you begin**

You must have completed the following tasks:

- Created a loop.
- Ensured that you have administrative privileges in DevOps Loop.
- Ensured that the member is already added to the teamspace.

1. Log in to DevOps Loop.
2. Click **Teamspace > <teamspace_name>** to navigate to your teamspace.
3. Select the loop in which you want to add members.
4. Click the Settings icon and select **Edit**.
   **Result**
   The **Loop Members** pane is displayed.
5. Click **Add Members**.
   The list of members in your teamspace is displayed.
6. Type the email address in the **Search** bar or select the member from the list and click **Add**.
7. Click **Save**.

**Results**

The member is added to the loop and its capabilities.

Related information

Removing a member from a loop on page 95

## Removing a member from a loop

You can remove a member from a loop when they no longer need access.

**Before you begin**

You must have completed the following tasks:

- Created a loop.
- Ensured that you have administrative privileges in DevOps Loop.

**About this task**

When you remove a member from a loop, the member is also removed from all connected capabilities within that loop.

1. Log in to DevOps Loop.
2. Select the loop from which you want to remove the member on the **All Loops** tab.
3. Click the Settings icon  and select **Edit**.

   The **Loop Members** pane is displayed.
4. Select the member in the **Members List** section, and click the **Delete** icon  .
5. Click **Save**.

   The member is removed from the loop and its capabilities.

Related information

Adding members to a loop on page 94

## Disabling a loop

You can disable a loop to pause its activities and stop updates. You can enable it again from the **Disabled** tab whenever you need it.

**Before you begin**

You must have completed the following tasks:

- Created a loop.
- Ensured that you have administrative privileges in DevOps Loop.

**About this task**

When you disable a loop, the following connected capabilities within that loop are impacted:

- **Plan**: Emails, alerts, and search functions for work item updates are disabled.
- **Control**: The Plan-Control webhook is deactivated and all the information in the events (such as pull request, branch creation etc) stop flowing from Control to Plan.
- **Measure**: Data flow from Control, Plan, Test and Build is interrupted and therefore value streams and dashboards no longer receive updates.

1. Log in to DevOps Loop.
2. Select the loop that you want to disable from the **All Loops** tab.
3. Click the Settings icon and select **Disable**.

   The loop is disabled and moved to the **Disabled** tab.

---

Related information

Enabling a loop on page 96

## Enabling a loop

You can enable a loop to restore its activities and allow updates to flow again.

**Before you begin**

You must have completed the following tasks:

- Created a loop.
- Disabled the loop you want to enable.
- Ensured that you have administrative privileges in DevOps Loop.

**About this task**

When you enable a loop, the following connected capabilities are restored:

- **Plan** – Emails, alerts, and search functions for work item updates are re-enabled.
- **Control** – The webhook is activated and all the events such pull request, branch creation etc starts flowing from Control to Plan.
- **Measure** – Data flow from **Control**, **Plan**, **Test** and **Build** resumes, and therefore value streams and dashboards receive updates.

1. Log in to DevOps Loop.
2. Select the Loop that you want to enable from the **Disabled** tab.
3. Click the Settings icon  and select **Enable**.

   The loop is enabled and appears in the **All Loops** tab.

---

Related information

Disabling a loop on page 95

# Learning Loop overview

Learning loop provides preloaded sample datasets across multiple DevOps Loop capabilities. It helps you learn and practice workflows by exploring sample projects, work items, and configurations without starting from an empty setup.

The learning loop, with its preloaded sample data, helps you to explore and understand DevOps Loop features:

- Understand how events flow between the capabilities. For example, a commit in Control triggers a build in Build and a deployment in Deploy. The updates automatically appear on the loop's hosted webpage.
- View a consistent end-to-end view of DevOps Loop capabilities for demonstrations, evaluations, and training.
- Experiment with features and configurations in a non-production environment.

Learning Loops include an automated deployment to web experience. Code changes flow through the full pipeline **code change → build → deployment → updated webpage** and each loop includes its own uniquely hosted URL. A dedicated NgineX web server pod serves static Learning Loop content and ensures isolation between loops.

### Scope

The sample datasets currently spans the following capabilities of DevOps Loop:

- **Plan**: Creates a project with releases, sprints, and work items.
- **Control**: Adds a sample file to a repository.
- **Build**: Requires a manual step to configure an active agent.
- **Deploy**: Maps an agent to a team, sets the license mode to floating, and creates components with predefined configurations. Deployments automatically publish updates to the loop’s hosted webpage through the NgineX pod.

The learning loop has the following key characteristics:

- Built for learning and exploration, not for use in production environments.
- Some manual steps, such as configuring build agents is required.
- Once configured, the end-to-end workflow is automated across Control, Build and Deploy. Code changes in Control trigger a build in Build, and a deployment in Deploy and the deployed content is automatically published to the loop's hosted webpage.
- After the sample data is loaded, you can edit or extend it to fit your own configurations.

  **Note:** Sample applications created by learning loop are now automatically hosted through the NgineX pod.

- Each learning loop is assigned a unique URL to ensure isolated hosting of static content.
- Supports more complex deployments in future releases, including multi-component segments.

## Preloading sample data into a learning loop

When you want to explore DevOps Loop capabilities and learn how workflows operate across Plan, Control, Build, Deploy, and Test, you can use the Learning Loop option to preload sample data with preconfigured projects, work items, and settings.

**Before you begin**

You must have administrative privileges in DevOps Loop.

1. Log in to DevOps Loop.
2. Click **Create Loop**.
3. Enter the required loop details, such as the loop name and description.
4. Enable the **Learning Loop** option.
   This option automatically loads sample projects, work items, and configurations across Plan, Control, Build, Deploy and Test once the loop is created.

   Enabling learning loop automatically sets up a hosted environment:

   - A dedicated nginX web server pod is created to host Learning Loop static content.
   - A unique URL is generated for the loop.
   - Deployments will automatically update content at this URL.
   - The test `SmokeTest.dtx.yaml` file is automatically created under the project in Test.
5. Perform the following steps to set the license type for the automatically installed deploy agent:
   a. Navigate to **Resources > Agents**
   b. Click **Actions** menu for the agent.
   c. Select the **Set License Type** from the following options:
      - **Authorized** or
      - **Floating**
   d. Click **Save**.

6. Navigate to **Test** and execute the test manually.
   The test result is displayed in **Execution** tab.

**Results**

You have successfully created a learning loop with preloaded sample data.

# Dashboards and insights

The dashboard consolidates data from all capabilities, such as Plan, Code, Control, Build, Test, Release, and Deploy, by offering a centralized view of the status. Dashboards are available with both summary and detailed perspectives, and they filter information specific to each loop.

Upon loop creation, two dashboards for each subscription tier are automatically provisioned and linked directly to the workspace of each loop. These dashboards are universally visible, irrespective of the user's licensing tier. You might see tier-specific differences in the dashboard that might arise from the depth and variety of data sources through integrated applications for a specific tier. The default dashboards are preconfigured and available for immediate use.

For advanced analytics, DevOps Loop includes a milestone risk assessment feature, which requires an optional plug-in.

### Data authorization

All the dashboard data is displayed based on your loop membership and access rights. You will only see information for loops of which you are a member, ensuring sensitive data remains secure and collaboration across teams is controlled.

## Viewing dashboards

You can view the overview and detailed dashboards for the loops in your teamspace. The dashboards display key metrics such as issues, workloads, builds, deployments, milestones, lead time, and cycle time. Data is shown only for loops of which you are a member, based on your access rights.

**Before you begin**

You are a member of the loop.

**About this task**

When you want to view dashboards for loops in your teamspace, you can access both **Overview** and **Detailed** dashboards. The Overview dashboard provides a high-level summary of loop activity, including issues, team workload, test results, builds, deployments, and milestone risk.

The Detailed dashboard provides more granular insights, such as open issues, pull requests, build and deployment details, test metrics, and timeline performance.

**Note:** You can view data only for the loops you are a member of, and the information displayed is restricted based on your access rights.

1. Click the **Dashboards** tab in the navigation panel.
   The **Dashboards** page is displayed, with **Overview** and **Detailed** dashboard for every loop.
2. Select the dashboard from the **Dashboard name** column.

**Results**

The selected dashboard with metrics and data is displayed.

Related information

Editing dashboards on page 100

## Editing dashboards

You can customize the dashboard view by applying filters and selecting specific metrics such as test results, build history, or deployment status to focus on the data most relevant to your loop.

**Before you begin**

You must have completed the following tasks:

- Created a loop.
- Ensured you have administrative privileges.

1. Click the **Dashboards** tab.
   **Result**
   The **Dashboards** page is displayed, with an **Overview** and a **Detailed** dashboard for every loop.
2. Click the **Edit** icon  .
3. Use any of the following options to edit the dashboard:

| Option | Description |
|---|---|
| Full Screen | Expands the dashboard to full-screen mode for easier viewing. |
| Share | Shares the dashboard view with other users in the following format<br>• Embed code<br>• Permlinks |

| Option | Description |
|---|---|
| Clone | Creates a copy of the selected widget within the dashboard layout. |
| Swap | Allows for swapping the position of widgets within the dashboard layout. |
| Add | Adds new panels or widgets to the dashboard. |
| Cancel | Cancels the current edit operation. |
| User Margin Between Panels | Adjusts spacing between panels in the dashboard. |
| Show Panel Titles | Toggles panel titles on or off. |
| Add Filters | Applies a filter to the data to be displayed in the dashboard. |

4. Click **Save** to save the changes.
5. Click **Refresh** to update the dashboard view with the latest data.

**Results**

You have customized the dashboard to focus on the metrics and information that are most important to your loop.

## AI-powered search

When you want to search work items, records, and related data in DevOps Loop, you can use the search bar to locate information across multiple capabilities. You can search across a wide set of artifact types, including issues, builds, tool requests, commits, and deployments, and you can filter results by loop or by category to quickly narrow your search.

You can search for epics, defects, and stories across Plan, and the results are presented as interactive cards that include metadata. From these cards, you can directly access associated URLs for pull requests (PRs), commits, and builds, making it easy to move from search results to action.

You can generate AI summaries of search results to quickly understand the context of work items, issues, and related data. These summaries are generated for the top five results on each page, helping accelerate decision-making by providing quick insights as you navigate through your results.

**Note:** AI summaries are available only if the OpenAI or watsonx integration is enabled in DevOps Loop. If neither integration is enabled, DevOps Loop will not generate summaries.

Search results are automatically filtered based on your access rights. You will only see documents and records associated with loops of which you are a member, ensuring that sensitive data remains secure and collaboration across teams is

controlled. This ensures that results are fully scoped to the current user, preventing visibility into team spaces you do not belong to.

# Chapter 6. Capabilities of DevOps Loop

HCL DevOps Loop provides an integrated suite of capabilities such as Plan, Code, Control, Build, Test, Release, Deploy and Measure to manage every stage of the software development lifecycle within a connected, continuous loop.

Each capability is designed for a specific purpose, but together they form a continuous feedback cycle, helping teams plan, build, deploy, and measure value efficiently.

## Plan

Plan is a change management software application within the DevOps Loop that supports seamless ticket creation, tracking, and comprehensive issue management. With customizable workflow automation, it helps teams to streamline and adapt processes to meet their unique business requirements while preparing work for the next stages of the loop.

For more details refer to the Plan Documentation.

## Code

Code is a cloud-based IDE that is part of HCL DevOps Loop, allowing developers to write, compile, build, and debug code directly in the browser without requiring local setup.

Built on Visual Studio Code, Code offers pre-configured tools, extensions, and libraries, along with support for custom extensions. It includes a file system, terminal, and tools for debugging and testing while integrating with remote source control repositories for secure version control.

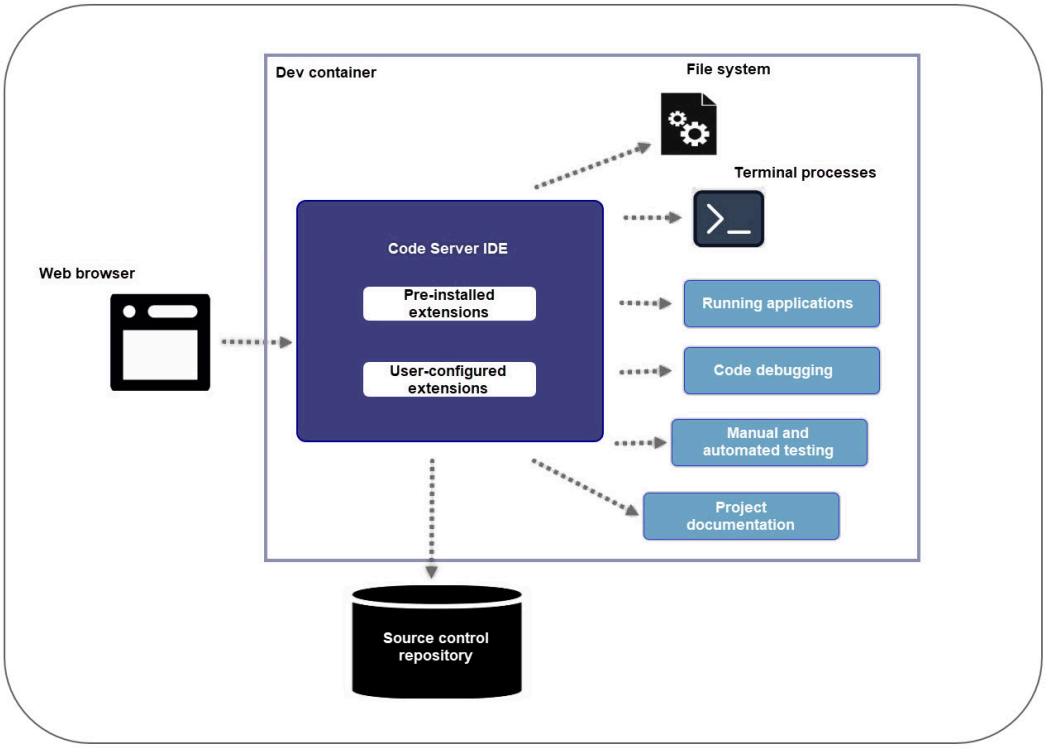
Continue reading to learn how Code enhances your development experience and integrates with DevOps Loop.

### Code overview

Code is a cloud-based integrated development environment on HCL DevOps Loop, with which you can write, compile, build, and debug software applications in a so-called dev container, accessible directly from a web browser. The editor provides a completely configured development environment with preinstalled extensions, tools, and libraries. You can start coding without the need to set up the environment on your local machine.

#### Architecture

The general architecture of the Code platform is shown in the following image.




Code provides personal dev containers that are configured for members of a teamspace. By using each dev container, you can write, build, and debug your code in an integrated environment without affecting the work of other users in the team. Each container is isolated for a specific user to prevent conflicts with different project setups. Additionally, you can also customize the individual dev container to meet your specific needs, by ensuring that all the necessary extensions and libraries are available for use.

The dev container integrates essential tools and processes, with the web browser serving as an interface for accessing the browser IDE, supporting coding, debugging, version control, and more.

### Visual Studio Code-based environment

The browser development environment is based on Visual Studio Code, which means that almost all features the latter provides are also available in Code. For more information, refer to Visual Studio Code documentation.

The development environment includes preinstalled extensions on page 130. However, you can also add custom extensions to suit project needs. It includes a file system for file management, a terminal for command-line operations, and tools for application execution, debugging, testing, and maintaining documentation.

### Source control integration

Code integrates with remote source control repositories, ensuring secure, accessible version control and project continuity across sessions. Code offers built-in support for Git repositories (for example, repositories stored in https://help.hcl-

software.com/devops/plan/3.0.3/oxy_ex-1/control/content/index.en-us.html). Other types of source control repositories are supported by means of extensions that you can install into the dev container.

### Accessing Code

To learn how to access the Code feature, see DevOps Loop user interface on page 21.

---

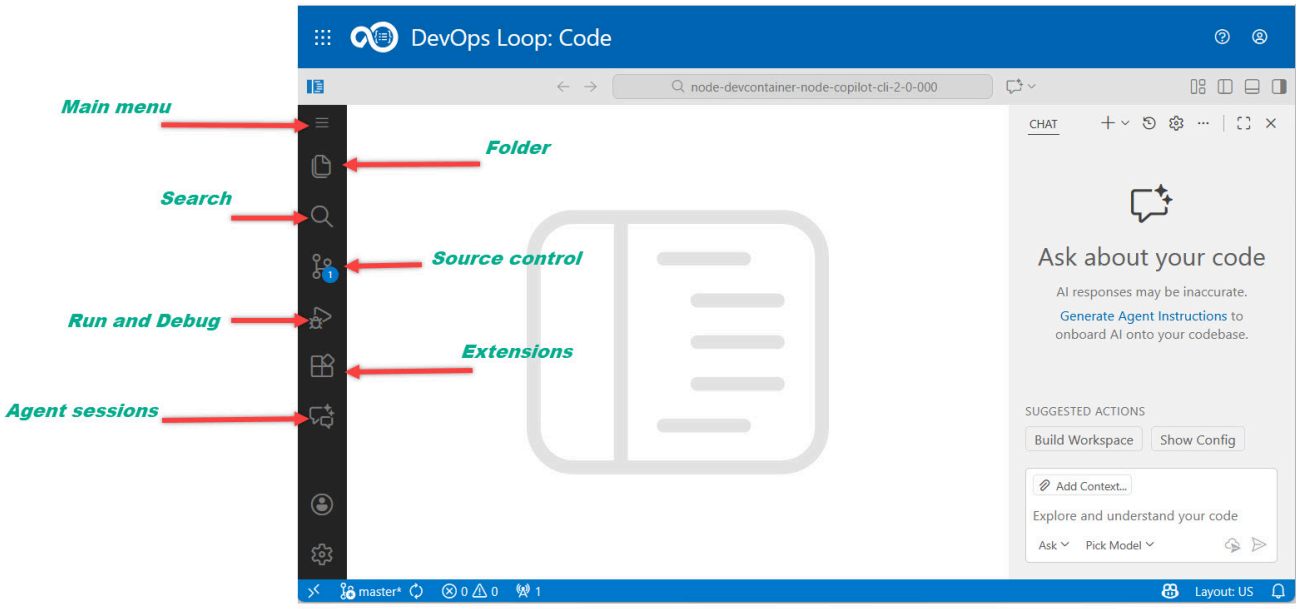
Related information

Git authentication for dev containers on page 114

## User interface

The Code user interface offers key features that are designed to facilitate navigation, file management, source control, debugging, and extension management within the browser IDE.

The following image shows the user interface with a vertical toolbar on the left-hand side of the Code IDE.




The following table lists the main highlights of the Code user interface:

| Menu items | Description |
|---|---|
| **Main (hamburger menu)** | Opens a collapsible side menu in a dropdown, and includes general menu items such as File, Edit, Selection, View, Go, Terminal, and Help. These menus are mostly the same as those that are present in the main toolbar of Visual Studio Code. |
| **Buttons** | |

| Menu items | Description |
|---|---|
| Folder | Opens the Explorer panel that provides tree-like navigation of the workspace structure. |
| Search | Provides the search and replace functionality for files in the workspace. |
| Source control | Provides the source control panel from which you can perform commands on Git repositories that are used in your workspace. |
| Run and Debug | Provides common actions for running and debugging applications. |
| Extensions | Provides a menu to manage extensions. |
| Agent sessions | Provides a view to manage and monitor ongoing AI-powered autonomous tasks, allowing you to interact with AI agents. |

For more information about the UI of the browser IDE, refer to the documentation of Visual Studio Code.

## Switching to your teamspace

You can switch to your teamspace to access team-specific development containers and their pre-configured settings.

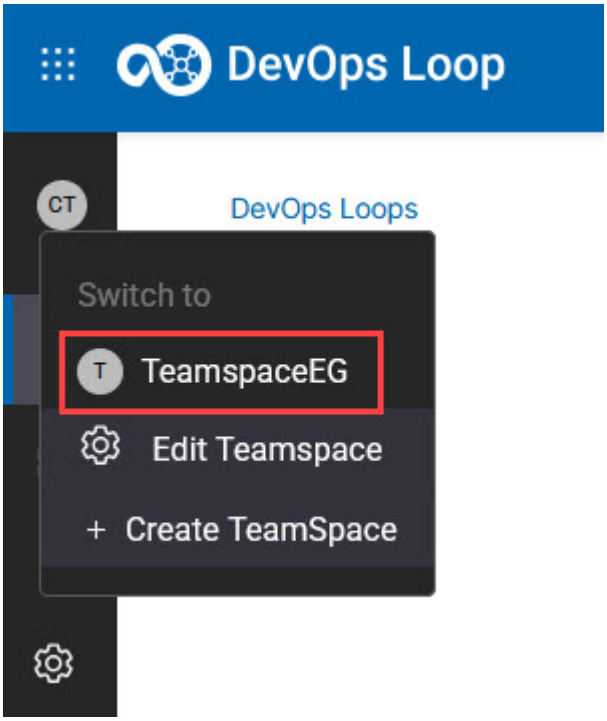
**About this task**

Upon login, you can navigate using two tabs: **Running Dev Containers** and **Other Dev Containers**.

If you have an assigned teamspace, you can launch preconfigured dev containers - such as Starter, Java, Python, or C++ - directly from the **Other Dev Containers** tab. Once launched, you can manage these running dev containers in the **Running Dev Containers** tab. You can find full details in the Working with dev containers on page 107 guide.

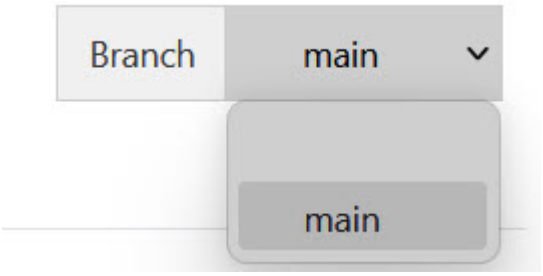
To access your team-specific dev containers:

1. Click the switch teamspace button in the top-left corner.




2. Select the appropriate teamspace created by your team admin.
   **Note**: If you do not see any teamspace listed, it means you have not been added yet. Contact your team admin to request access to a teamspace.

   You can also switch to a different branch (other than main) using the Branch menu in the upper right corner of the IDE. The branch switch menu is visible only in **Other Dev Containers** tab. This allows for versioning dev containers, which lets a development environment evolve together with the application it creates.

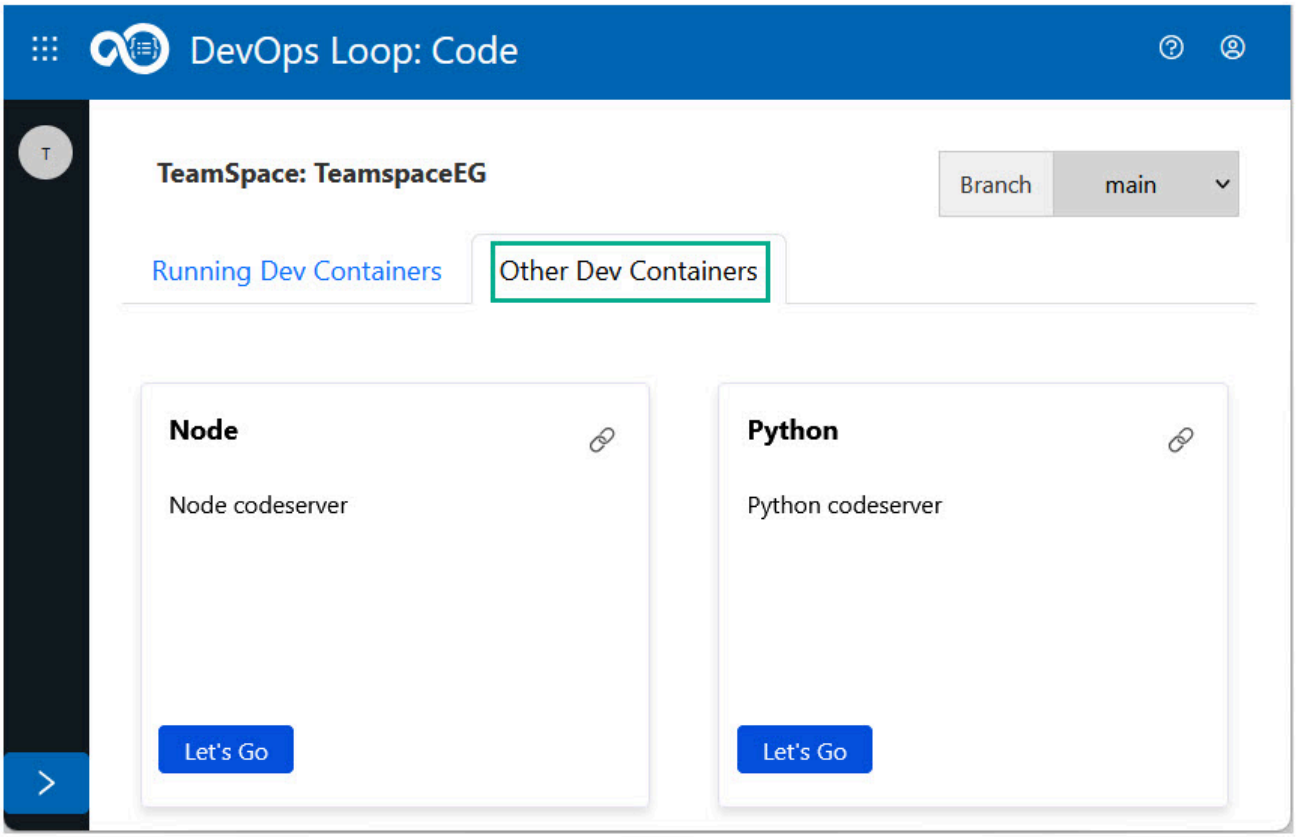



## Working with dev containers

You can launch a dev container to load your specific development environment and begin active development tasks.

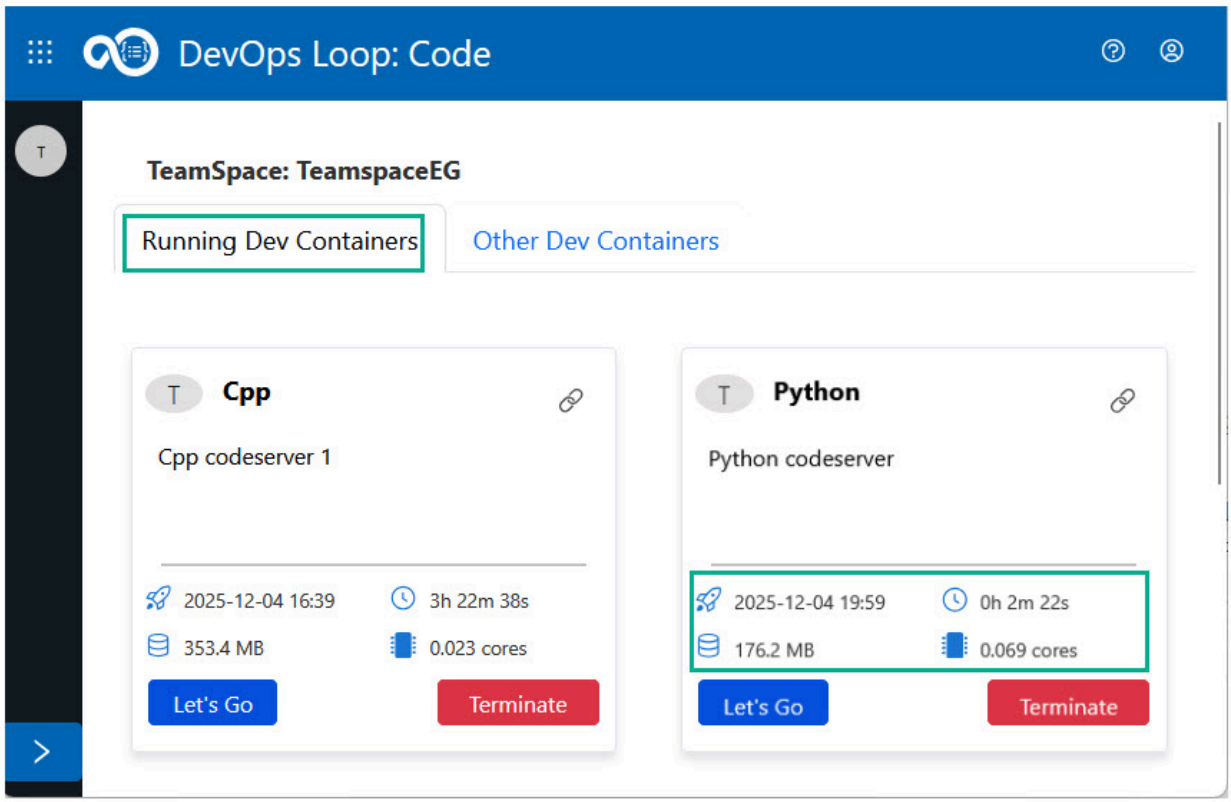
**About this task**

When you are added to a teamspace, new preconfigured dev containers appear on your landing page under the **Other Dev Containers** tab. Since you have just accessed the workspace for the first time, the **Running Dev Containers** tab will initially be empty. Launching a container moves it to the **Running Dev Containers** tab, where all active environments are managed.




To start working with a dev container:

1. Click the **Let's Go** button on a dev container.
2. The dev container will launch and begin running - this may take some time initially.
3. Once running, the container will move to the **Running Dev Containers** tab.




To optimize performance, Code provides real-time visibility into the resource metrics of your dev containers, such as:

- Launch time and date: The specific date and time the container was started.
- Duration: The total uptime of the container since launch.
- Memory consumption: The real-time RAM usage, measured in Megabytes (MB).
- CPU cores: The number of CPU cores currently utilized by the container.

4. Go to **File** > **Open Folder** to get started, after the browser-based IDE loads.
5. Click **Yes** when prompted with "**Do you trust the authors of the files in this folder?**".
6. Navigate to the workspace directory.
7. You can now start working in your workspace. For example, you can clone a Git repository to populate your workspace with the source code you want to work with.
   **Note**: This is just a quickstart example. As the environment runs on a VS Code Server, you can install extensions, debug applications with breakpoints and live logs, use terminals and Git integration, and fully customize your dev container with the dependencies you need. Use this as a starting point to explore the full development workflow.

   A video version of the above steps is also available for reference. It demonstrates the full workflow - from accessing Code to running your application in the browser IDE. Watching the video can help you follow along more easily. Learn more about dev containers here on page 110.

   ../videos/new-qs.mp4

## Dev containers

You can use this information to understand why and when to choose between available dev containers. You should choose the Starter container when you need basic tasks, or a preconfigured container when your project requires a customized environment (like Node.js or Java) assigned by an administrator.

**Starter dev container**

The **Starter** dev container provides a basic environment designed for essential tasks like text editing and cloning Git repositories. It includes the **DevOps Code** extension, which offers commands for seamless integration with tools like DevOps Plan on page 131.

This container is available to all users, regardless of teamspace membership. After logging into Code, you will find **Starter** under the **Other Dev Containers** tab. Once launched, it moves to the **Running Dev Containers** tab.

The following video demonstrates two scenarios: one for users without a teamspace, and one for users within a teamspace.

../videos/starter-userspace-not-userspace.mp4

To launch the Starter container, go to the Code landing page and click the **Let's Go** button on the **Starter** container tile. Alternatively, to copy a direct link to the container, click the copy link button on the tile. You can then paste the link into a web browser address bar to open the container.

**Preconfigured dev container**

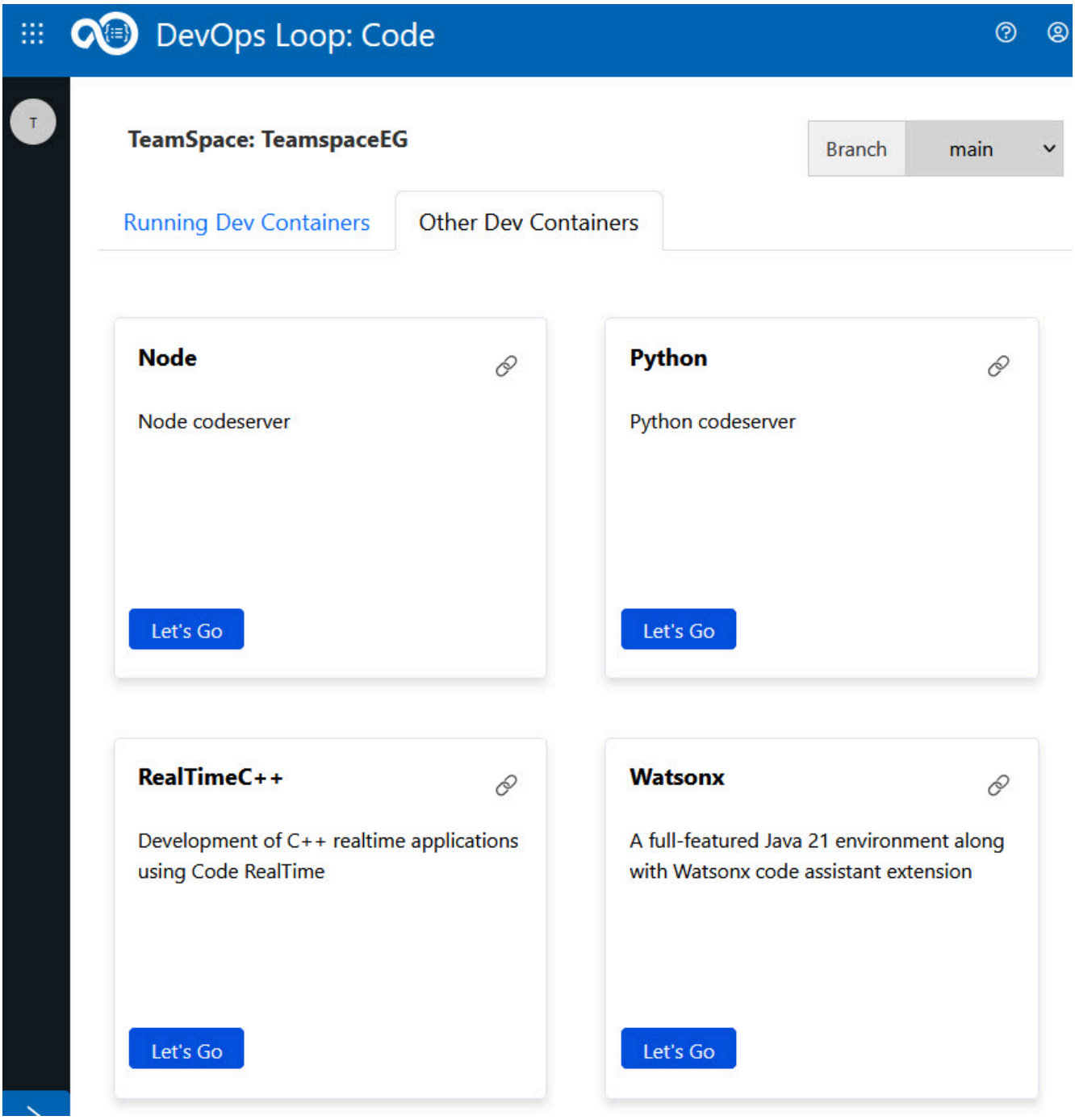
A preconfigured dev container provides a customized development environment tailored to specific project requirements, such as C++, Java, Node, Python, RealTimeC++, Watsonx, etc. A preconfigured dev container is added via a teamspace created and managed by an administrator on page 120.

To access a preconfigured dev container, switch to the teamspace assigned to you by your administrator. Then, select the relevant dev container from the available options. For example, to work in a Node.js environment, click the **Let's Go** button on the Node dev container tile.

**Note:** If no teamspace appears, it likely means you are not added to one yet. Contact your team admin to request access.

**For example only**: The image below shows how dev containers appear after switching to an admin-assigned teamspace, such as TeamspaceEG. In this case, you can see preconfigured dev containers alongside the Starter container.

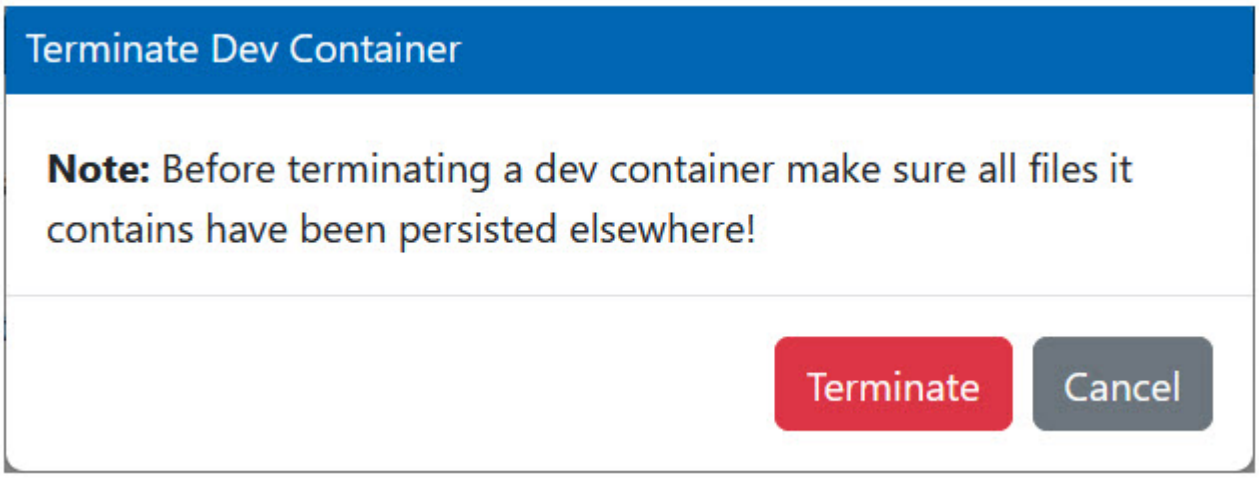
This setup is just an example. The actual teamspace name and available containers might vary.




**Note:** Administrators can define standard dev containers to ensure consistency across team environments. For more information on configuring dev containers, see the Administrator Guide on page 120.

**Terminating a dev container**

When a dev container starts running, you can see a **Terminate button** on the dev container tile. This action opens the **Terminate Dev Container** dialog.




**Important:** When you terminate a dev container, all files and unsaved work are deleted. Therefore, you must push your changes to a remote source control repository, such as a Git repository, before termination.

## File system

Understanding the file system structure in a dev container helps you organize your work effectively and maintain compatibility across different development environments.

**File system tree**

You can store your files in or below the /usr/code root directory. This works like a shared directory across all dev containers, ensuring your files remain persistent and accessible.

```
/usr/code/                                    [ROOT DIRECTORY - Shared files
accessible from all dev containers]
│
├── dev-container-1/
│   ├── (your project files)
│   ├── data directory/           [RESERVED - DO NOT MODIFY]
│   └── extension/                [RESERVED - DO NOT MODIFY]
│
├── dev-container-2/
│   ├── (your project files)
│   ├── data directory/           [RESERVED - DO NOT MODIFY]
│   └── extension/                [RESERVED - DO NOT MODIFY]
│
├── dev-container-3/
│   ├── (your project files)
│   ├── data directory/           [RESERVED - DO NOT MODIFY]
│   └── extension/                [RESERVED - DO NOT MODIFY]
│
├── ...
│
└── (additional shared files/repositories)
```

**Container-specific directories**

You will find each dev container has its own subdirectory under /usr/code where you can create and organize your project files, source code, configurations, and cloned repositories.

You will see two system-managed directories in each container:

- data directory: Contains system-managed data
- extension: Contains system-managed extension files

**Note:** You must not modify these directories as changes may cause your dev container to malfunction.

**Shared files**

You can place files directly under /usr/code to make them accessible from any dev container. This is useful for common configurations, shared libraries, or cross-environment projects.

## Git authentication for dev containers

You can use Git within a dev container to ensure version control and collaboration while keeping your environment isolated.

There are many authentication methods specific to configuring Git in dev containers, including UI feature, personal access tokens and device code authentication.


- DevOps Code UI (Recommended and Easier) on page 114
- Personal Access Token (PAT) on page 117
- Device Code (GPG) on page 118


**Note:** For information on configuring Git credentials using Control Repositories, see Managing Git authentication for developers.

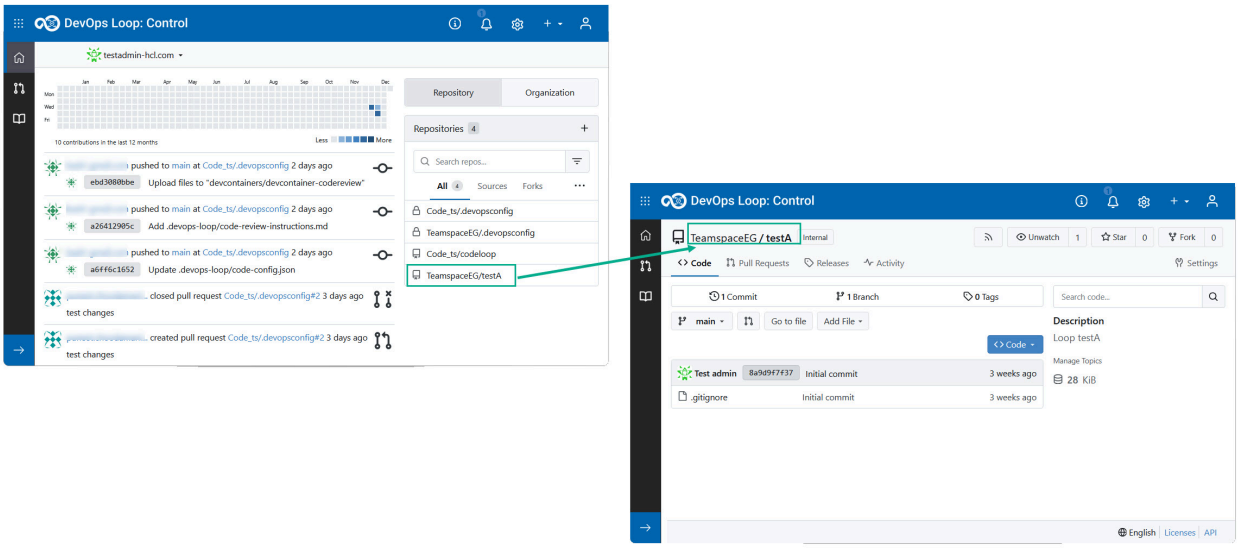
## Authenticating with DevOps Code UI (Recommended)

You can create a personal access token in Control, and then use that as your password when authenticating from Code. This type of authentication is based on SSH keys, which are automatically generated for your dev container and registered with Control.

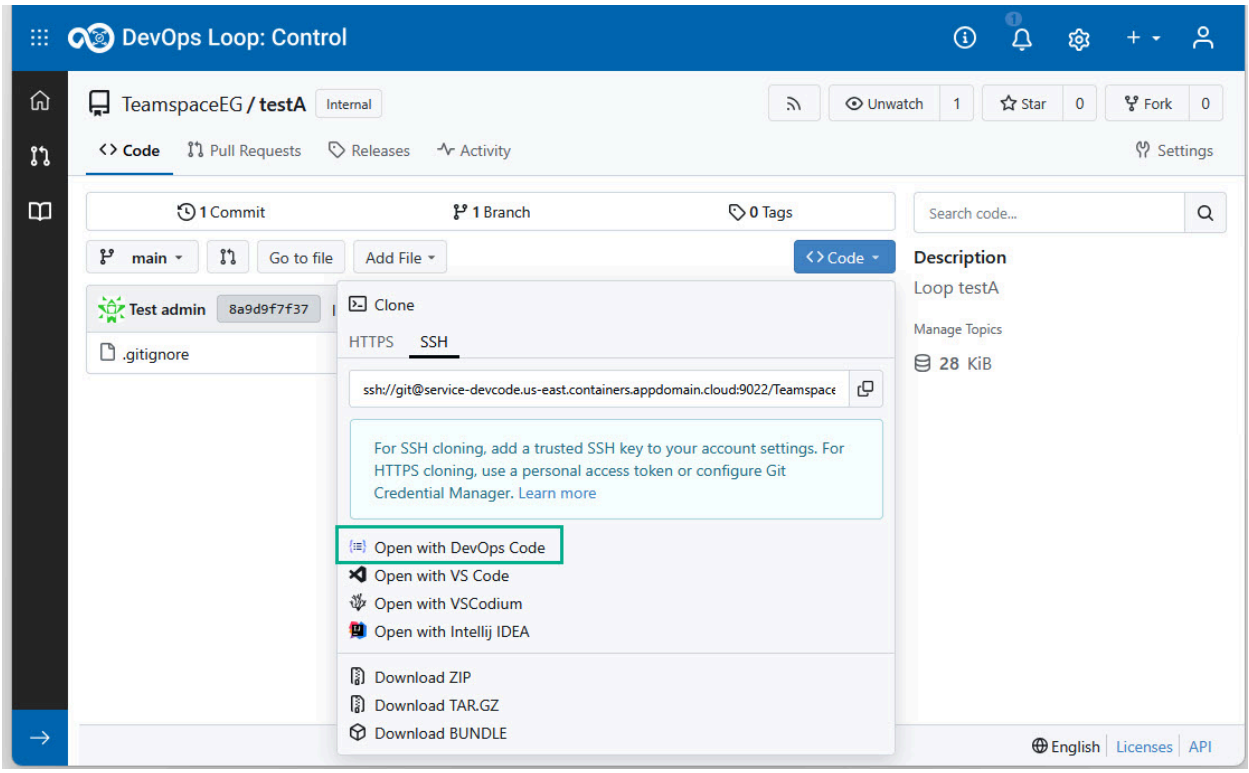
### About this task

You can use this procedure to clone a **Control** repository and push changes using **Code**.

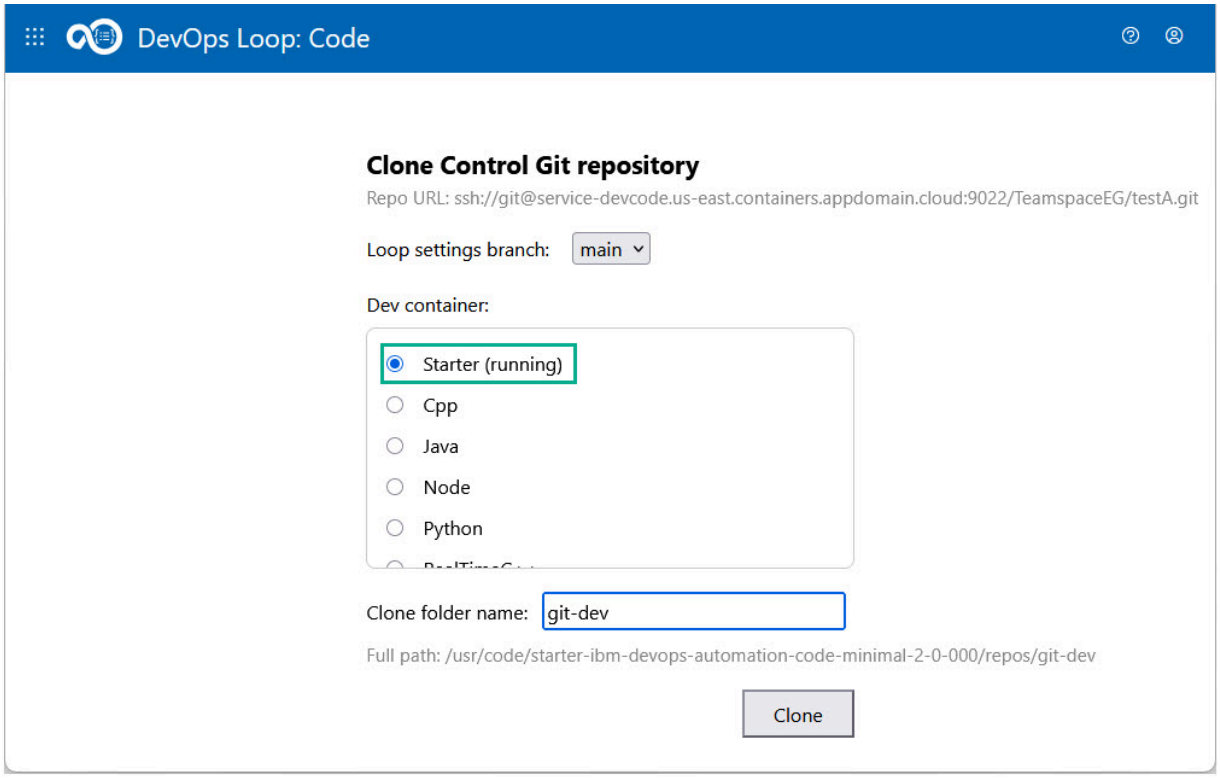
1. Navigate to the Control repository you wish to clone:




2. Click **Open with DevOps Code**.




3. You will be redirected to the **Clone Control Git repository** page.



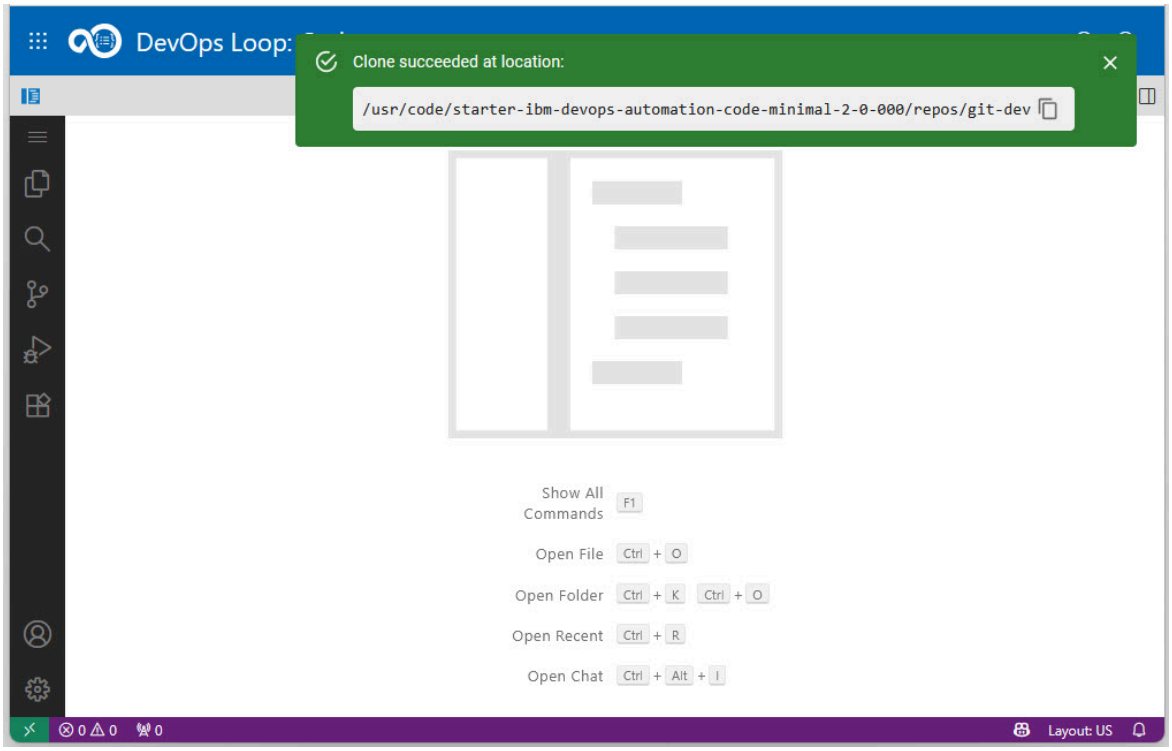

Configure the repository settings:

a. Select the **Loop settings** branch. This is the branch from where you can fetch available dev containers that you choose to use.
b. Select **Dev container**. The procedure will be faster if you select a dev container that is already running. Otherwise, the dev container will first be launched before beginning a cloning.
c. Enter a valid folder name in the **Clone folder name** field.

> **Note:** The **Full path** field will automatically update on page 112 to reflect the folder name you enter.

d. Click **Clone**. The Code IDE instance will open, and the repository will be cloned.

4. Copy the clone location:
   a. Wait for the IDE page to load completely.
   b. Locate the **Clone succeeded at location** notification popup.




   c. Copy the file path displayed in the message.
   d. Open something from the cloned repository in the Code IDE. Choose one of the following alternatives:
      - Open the repository root folder (which you copied above), or one of its subfolders, as a workspace folder: **File > Open Folder**
      - Add the repository root folder (which you copied above), or one of its subfolders, to a workspace that is already open: **File > Add Folder to Workspace**
      - Open a .code-workspace file from the cloned repository: **File > Open Workspace from File**

5. Commit and push your changes:
   Use the built-in Git source control tools within the IDE to commit your updates and push them to the desired branch.

## Authenticating Git with personal access token

You can securely clone a Git repository from your Control system. You can use it as a secure alternative to other Git authentication methods for accessing files in your dev container.

**Before you begin**

You must have performed the following task:

- Read File system in the dev container on page 112.
- Read How to generate and use Personal Access Token in Control.

**About this task**

The following method outlines how to clone a Git repository in Code using a Personal Access Token (PAT).

1. **Generate a Personal Access Token**
   a. Navigate to your **Settings** in Control to generate a PAT.
   b. To generate a personal access token, click the **Access Tokens** tab in Settings.
   c. Add the Token Name, and set all permissions to Read and Write.
2. **Construct the clone URL for repository owned by username or organization ID**
   a. The URL for cloning is structured with your username and the PAT. For example, assume your username is code@example.com, DevOps Loop is deployed at mydomain.com and your Control repository has the name repo-name, then the cloning URL will look like this:

      https://code-example.com:PAT@mydomain.com/control/code-example.com/repo-name

   b. Note that the @ in the username is replaced with a hyphen (-) in the URL.
   c. Replace PAT with the personal access token you generated in Control.

      **Note:** When creating a repository in Control, the **Owner** tab gives you two options: username or teamspace/organization ID. This choice dictates the correct structure for the clone URL.

      For a teamspace-owned repository, the teamspace ID must be used as the organization_identifier (the second-to-last part of the URL path) for the clone operation to succeed.

      https://code-example.com:a1b39ac1e546af4dc664ba36f4b12e77f8dad2da@mydomain.com/control/teamspace ID/repo-name

3. **Clone the Repository**
   You can clone the repository either via the terminal or UI:
   **Using Terminal:**
   a. Run the full git clone command with the constructed URL

      git clone https://code-example.com:<PAT>@mydomain.com/control/code-example.com/repo-name

   **Using UI**:

a. Go to the Source Control page in the IDE.
b. Click the Clone Repository button in the UI.
c. Paste the constructed clone URL into the input box.
d. Choose the target folder (e.g., /usr/code/repos).
e. Click Clone to start cloning.

**Results**

The Git repository is successfully cloned to your target folder. You can now begin working with the cloned files in your dev container.

## Authenticating Git with device code

You can set up a GPG key and password store within your dev container to perform Git operations securely and establish a robust authentication layer to clone, commit, and push changes to your repository.

**Before you begin**

You must have performed the following task:

- Read File system in the dev container on page 112.

**About this task**

The process involves creating a GPG key, initializing a password store, and then using a device code or passphrase to perform Git operations. https://docs.github.com/en/authentication/managing-commit-signature-verification/generating-a-new-gpg-key

1. **Launch the dev container**
   a. From the launcher in DevOps Loop, click the **Code** button.
   b. Run the dev container to start the Code IDE instance.
   c. After the container is running, open a new terminal by clicking the **Explorer** button in the right vertical toolbar and selecting Terminal.

   All subsequent steps must be run as commands from this terminal.
2. **Configure a GPG Key**
   a. In the terminal, run the following command to create a GPG key with a passphrase

      ```
      gpg --full-generate-key
      ```

   b. When prompted, select "RSA and RSA (default)", which is the first option, as the key type.
   c. Specify the desired key size and how long the key should be valid.
   d. Once the key longevity is provided, you will get a prompt showing the key's expiration date. You need to confirm if this date is correct.
   e. The key identification process will then ask for a User ID, which includes your real name, email address, and an optional comment. During this step, you may be instructed to perform actions like moving your mouse or typing on the keyboard to generate enough "entropy".
   f. To protect your new key, you will be prompted to enter a passphrase.

g. After the key is generated, you will see a line that says

   gpg: key <GPG Key> marked as ultimately trusted

   This confirms the key's creation.

h. If the key is not printed properly, use this command:

   gpg --list-secret-keys --keyid-format LONG

3. **Initialize the Password Store**
   - Use the following command to initialize the password store with your key.

     pass init <GPG Key>

     A message similar to the following confirms that the password store has been initialized and a directory (e.g., /usr/code/starter-ibm-devops-automation-code-minimal-1-0-300/.password-store/) has been created:

     mkdir: created directory '/usr/code/starter-ibm-devops-automation-code-minimal-1-0-300/.password-store/'

     Password store initialized for <GPG Key ID>

4. **Export an environment variable**
   You have to export an environment variable each time you open a new terminal.
   export GPG_TTY=$(tty)
5. **Perform Git Operations via Terminal**
   After the setup is complete, you can use a device code to perform Git operations.The device code is only valid as long as the GPG key is valid. If the key expires (for example, in one day), you will need to repeat the entire setup process the next day. When a session expires, you can use the passphrase to perform further Git operations.
   Example: Cloning, adding, committing, and pushing changes to the Control repository
   a. Copy the repository link from Control Repository, and run the following command to clone it:

      git clone <control-repo-link>

      **Note:** During the clone operation, you are prompted to authenticate by visiting a link provided in the terminal, entering the provided device code, and then granting access to the Git credential manager by clicking Yes.

   b. After cloning, you can add a file, and then commit and push the changes to the repository. This process is similar to working with a standard GitHub repository.

      When you run git push, you are prompted to provide your passphrase to complete the operation.

6. **Alternatively, using the Git UI**
   a. From the left toolbar, click File > Open folder and select the root directory.
   b. Click the Clone Repository button.
   c. Paste the Control repository link when prompted.

d. Authenticate by visiting the provided link and entering the device code, then approve the Git credential manager prompt.
e. Once cloned, use the UI to add, commit, and push changes.
f. When pushing, enter your passphrase when prompted to complete the operation.

## Configuring dev containers

As a teamspace admin in Code, you can configure the dev containers so that they are available to the team members.

**Before you begin**

You must have performed the following tasks:

- Installed Docker and ensured that it is running on your machine.
- Installed Node.js and npm (for Dev Container CLI).
- Created a DockerHub account and enabled it for pushing images.
- Gained access to Control with permission to create repositories.

**About this task**

You can set up and preconfigure dev containers for users in a teamspace while using Code. You can create and configure the container environment, build an image, publish it to a registry, and make it accessible within the teamspace's configuration repository. These configurations follow the Dev Containers specification and are versioned by pushing them to a Control Git repository.

1. Install the **Dev Container CLI** tool:
   a. Clone the GitHub repository by running the following command:
      ```
      git clone https://github.com/devcontainers/cli.git
      ```
   b. Navigate to the cloned directory and install the following Dev Container CLI:
      ```
      npm install -g @devcontainers/cli
      ```
2. Create the **devcontainer.json** file:
   You can create the devcontainer.json file by defining the metadata such as name, description, build or image source, and runtime Visual Studio Code extensions.
   Here's a sample devcontainer.json:
   ```
   {
     "name": "Go Dev Container",
     "description": "A preconfigured Go development environment with code-server support and Go tooling.",
     "build": {
       "dockerfile": "Dockerfile"
     },
     "customizations": {
       "vscode": {
         "extensions": [
           "secure-dev-ops.devops-code",
           "golang.go"
         ]
       }
   ```

```
  }
}
```

**Note:** In the example above, extensions are specified by their IDs, and will then be downloaded from the Open VSX registry (that is, <publisher.name>  format. However, administrators may need to pre-bundle .vsix extension files directly within the dev container image. In such scenarios, especially within offline environments, you can specify the local .vsix files using a relative path. See Loading extensions from local .vsix files on page 128.

You can also download a sample file here: devcontainer.json

Refer to the Dev Container metadata reference guide to include JSON metadata related to general settings, Docker or Docker Compose-specific options, and tool-specific properties for configuring a devcontainer.js file.

3. Create the **Dockerfile** for custom environment setup:

   You can add a Dockerfile to define the environment and behavior of your development container by specifying the base image, installing tools, setting environment variables, and configuring the container runtime.

   Example Dockerfile:

```
FROM ubuntu:20.04
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && \
    apt-get install -y \
        sudo \
        curl \
        git \
        ca-certificates \
        build-essential && \
    apt-get clean

# Install Go
ENV GO_VERSION=1.22.3
RUN curl -fsSL https://go.dev/dl/go${GO_VERSION}.linux-amd64.tar.gz | tar -C /usr/local -xz
ENV PATH="/usr/local/go/bin:${PATH}"
ENV GOPATH="/home/1001/go"
ENV PATH="${GOPATH}/bin:${PATH}"

# Install code-server
RUN curl -fsSL https://code-server.dev/install.sh | sh -s -- --version=4.98.0

# Safe for Ubuntu + RHEL images
RUN chmod -R ug+w /etc/pki/ca-trust/source /etc/pki/ca-trust/extracted || true

RUN useradd -m -u 1001 devuser
USER 1001
WORKDIR /home/1001
CMD ["tail", "-f", "/dev/null"]
```

Key Commands required for the Dockerfile:

- FROM: Set the base image.
- ENV: Set the environment variables.
- RUN: Install dependencies and clean cache to keep the image lean.
- USER: Run the container as a non-root user (1001) for security.
- CMD: Keep the container running by tailing a null file.

To create or customize a Dockerfile for your development container, refer to the official Dockerfile reference. This guide provides detailed explanations and examples of all supported commands. You can also explore dev container documentation for guidance on integrating your Dockerfile into a .devcontainer setup.

**Note**: If DevOps Loop runs using self-signed certificates or private Certificate Authorities (CAs), the admin must add the line RUN chmod -R ug+w /etc/pki/ca-trust/source /etc/pki/ca-trust/extracted in the Dockerfile before the line USER 1001.

You can download a sample Dockerfile here: Dockerfile

4. Build the **dev container** image:
   Navigate to the root **.devcontainer** directory and run the following command to build the dev container image.

```
devcontainer build --workspace-folder . --image-name sarasonia/go-container:1.0
```

   This command reads the devcontainer.json file and executes the Dockerfile. The syntax should be in the following format:

```
devcontainer build --workspace-folder . --image-name <username>/<container-name>:<tag>
```

   where,
   - **<username>**: Docker Hub username
   - **<container-name>**: your container (repository) name
   - **<tag>**: version (for example, 1.0)
5. Update the **devcontainer.json** file to use the image key:

```
{
  "name": "Go Dev Container",
  "description": "A preconfigured Go development environment with code-server support and Go tooling.",
  "build": {
    "dockerfile": "Dockerfile"
  },
  "image": "sarasonia/go-container:1.0",
  "customizations": {
    "vscode": {
      "extensions": [
        "secure-dev-ops.devops-code",
        "golang.go"
      ]
    }
  }
}
```

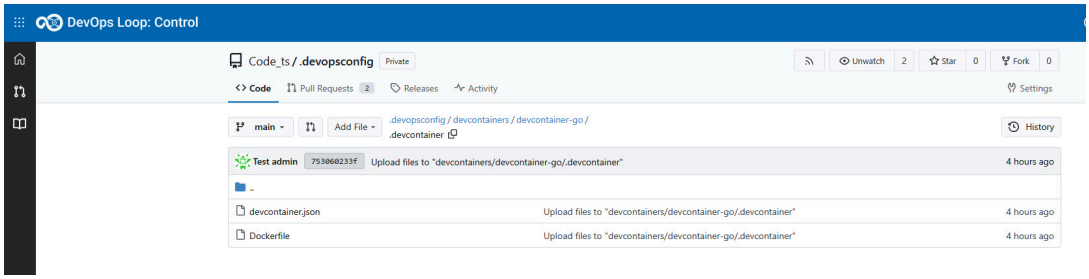
6. Push the image to **Docker Hub** registry:
   Login and push.

```
docker login
docker push sarasonia/go-container:1.0
```

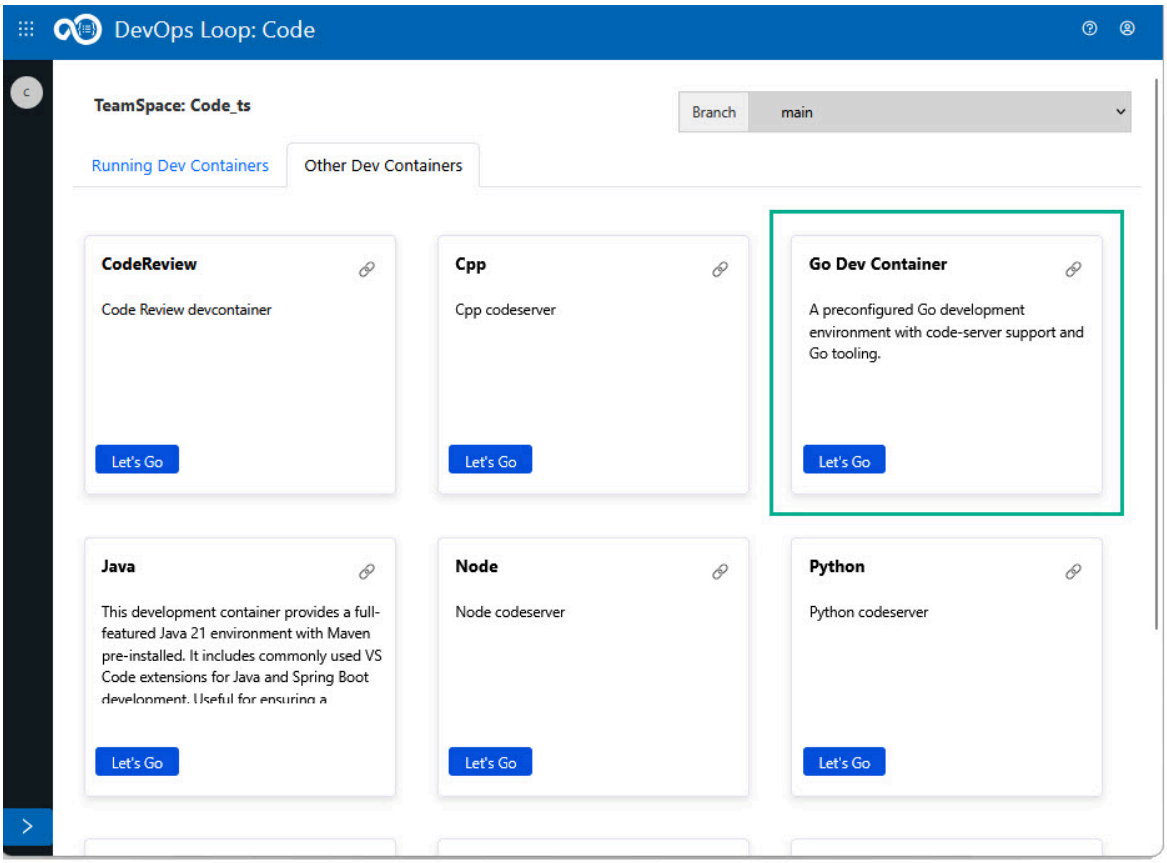
After it is pushed, it will be available at: docker.io/devuser/java-container.

7. Access **Control**:
    a. Sign in to HCL DevOps Loop. Navigate to your teamspace.
    b. Go to the **Control** page. You'll see a repository named .devopsconfig under your teamspace (e.g., TeamspaceEG/.devopsconfig).
8. Upload **devcontainer.json** and **Dockerfile** using Control:
    a. Inside the **.devcontainer** folder of the repo, upload devcontainer.json and Dockerfile.
    b. Commit and push the changes to the main branch.

**Note:** You can also commit the changes to a different branch other than the main branch. In such a case, you will see different branches on the Branch menu.




9. Start the sample **Go container**:
    a. Go to the Code landing page.
    b. Click the **teamspace** button and switch to your teamspace on page 106. Under **Other Dev Containers**, you will see a new dev container:

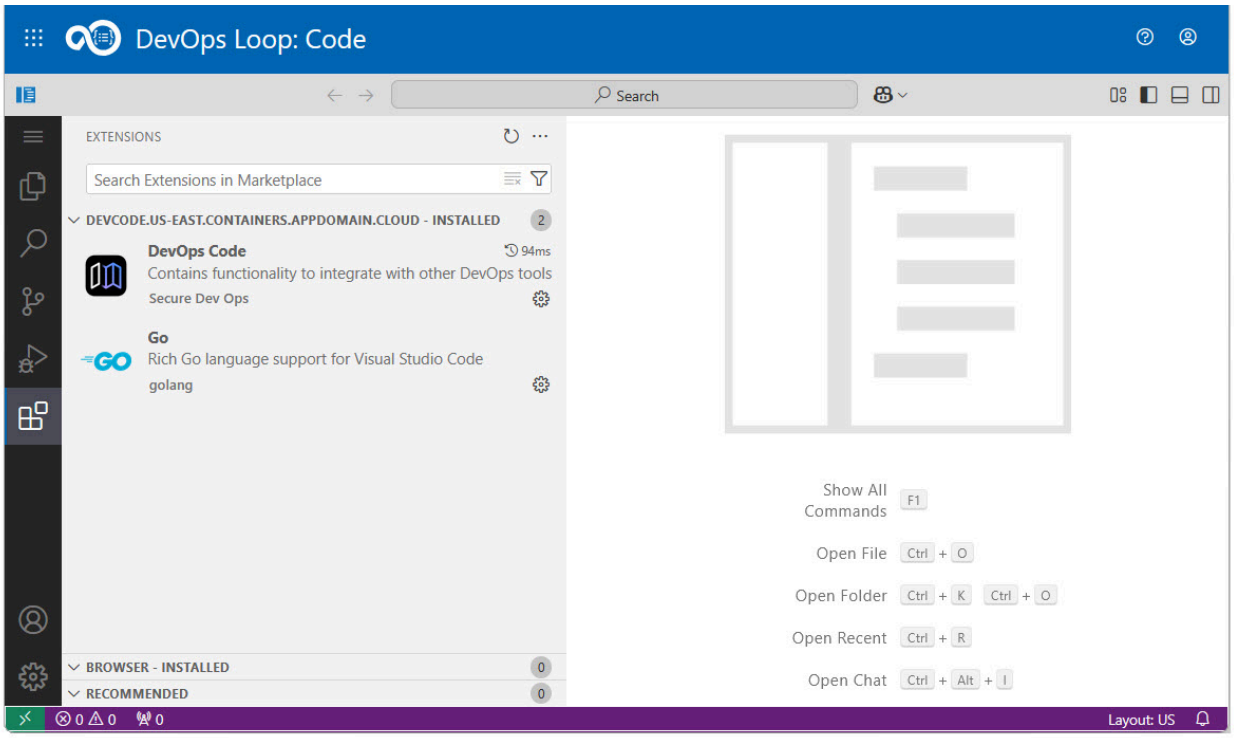



c. Click **Let's Go** to launch the container.

**Note**: The first-time launch may take a few minutes.

10. View **Visual Studio Code** Extensions:

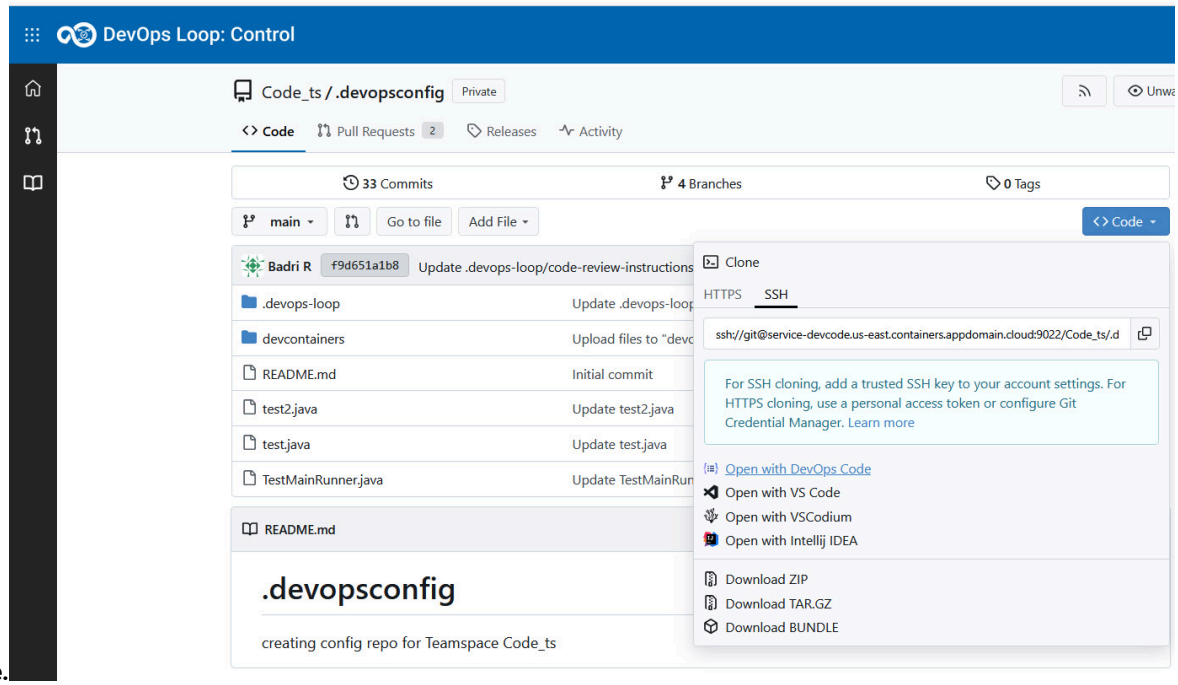
    In the Code IDE instance, click the **Extensions** button to see and verify the extensions you had configured in **devcontainer.json** file.



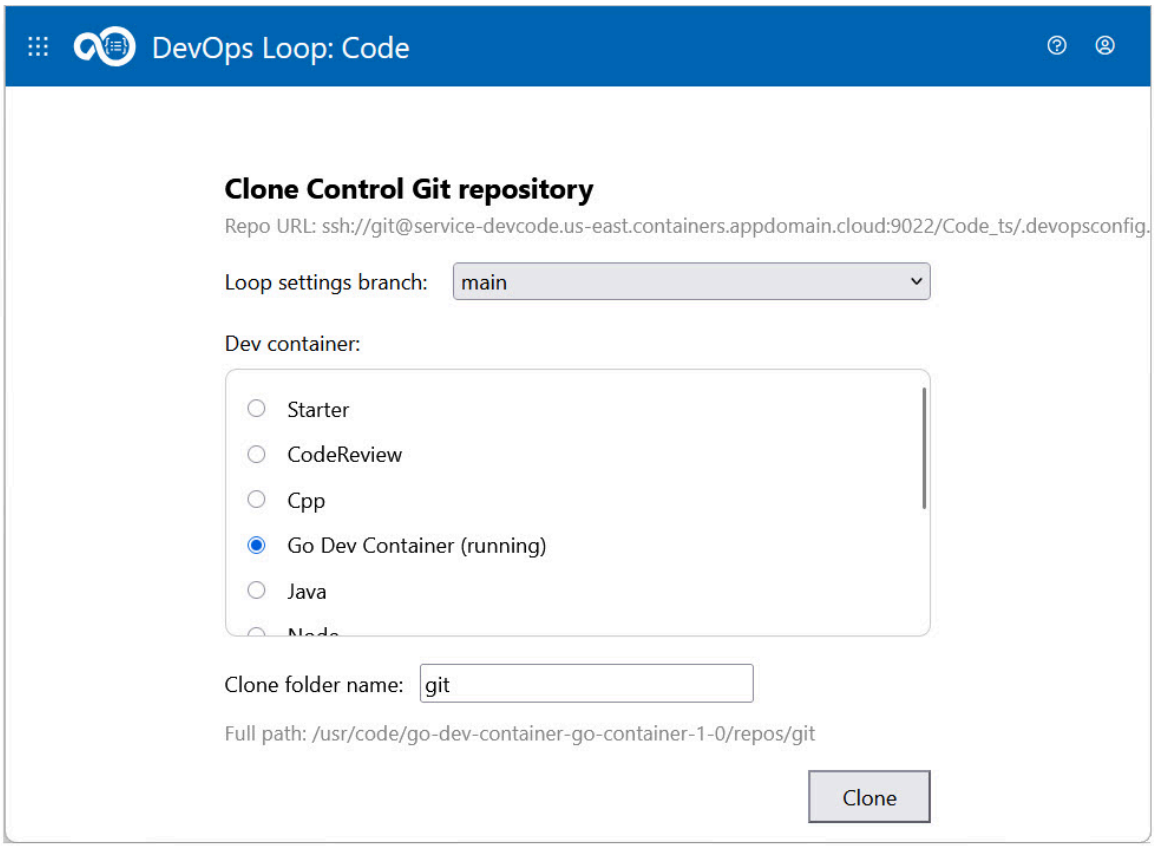

11. Cloning .devopsconfig repo using **DevOps Code**:
    After navigating to the Control page and locating the .devopsconfig repository, you can update your configuration files using the recommended workflow

    This method is recommended for making multiple changes, as it provides an easier user experience by leveraging the Git functionality within Code.

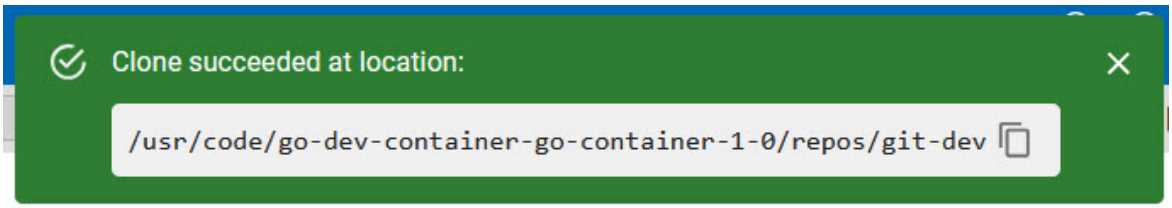
a. Click the **Code** dropdown button next to the repository name, and select **Open with DevOps Code.**




b. This directs to the **Clone Control git repository** page. Here, add the **Loop settings branch**, **Dev container** and a suitable **folder name** for cloning. You can see how the full path changes according to the name you updated in the **Clone folder name** textbox.




c. Click **Clone** to clone the repository. This directs to Code IDE instance. After successful loading of the IDE page, you get a **Clone succeeded at location** message popup. Copy the location.




d. In the Code IDE instance, go to the hamburger menu, select **File** and click **Open Folder**, and paste the location path in the **Go to file** dropdown textbox, and click the **OK** button.
e. Navigate to the devcontainer folder in the left Explorer pane, add/update your devcontainer.json and Dockerfile files, and use the built-in Git tools within **Code** to commit and push the changes to the desired branch of **Control**.

**Results**

The custom container image has been successfully built and published to the Docker registry. The image is now ready to be configured for use within your teamspace.

**What to do next**

After you configure dev containers for teamspaces, members in a team can launch any dev container directly from their Code landing page.

## Loading extensions from local `.vsix` files

Using Code, you can install and activate extensions in an offline or air-gapped environment by loading them from local .vsix files specified in the dev container configuration.

**About this task**

You can load extensions from a local .vsix file by specifying a relative path. This approach allows offline or air-gapped environments to activate extensions and is also necessary for proprietary extensions that should not be published publicly, without requiring access to the Open VSX registry.

1. Place your .vsix extension files into a local folder, such as extensions/ inside a cloned .devopsconfig repository: This allows offline environments to access the extensions without connecting to the Open VSX registry.
2. Reference the local .vsix files in your dev container configuration file (devcontainer.json):

   **devcontainer.json example:**

   ```
   {
     "name": "Node",
     "description": "Node codeserver environment with offline extensions",
     "build": {
       "dockerfile": "./Dockerfile"
     },
     "customizations": {
       "vscode": {
         "extensions": [
           "IBM.wca-core-1.6.2.vsix",
           "ms-python.python-2025.14.0.vsix"
         ]
       }
     }
   }
   ```

   You can specify the exact version of each extension you want to install and control updates centrally.
3. Update your Dockerfile to copy the extensions folder into the container during build:

   **Dockerfile example:**

   ```
   FROM mcr.microsoft.com/devcontainers/javascript-node:1-22-bookworm

   ENV DEBIAN_FRONTEND=noninteractive

   RUN apt-get update && \
       apt-get install -y sudo curl git gnupg pass pinentry-tty && \
       apt-get clean

   RUN curl -fsSL
    https://github.com/git-ecosystem/git-credential-manager/releases/download/v2.6.1/gcm-linux_amd64.2.6.1.
   tar.gz -o gcm.tar.gz \
       && tar -xvf gcm.tar.gz -C /usr/local/bin \
       && rm -rf gcm.tar.gz \
   ```

```
    && git-credential-manager configure

RUN curl -fsSL https://code-server.dev/install.sh | sh -s -- --version=4.103.2

COPY extensions /opt/extensions/

USER 1001

CMD ["tail", "-f", "/dev/null"]
```

This ensures the extensions are installed automatically when the container is set up.

4. Build and run the dev container:
   The extension manager will detect the local .vsix files and install them during container startup.

**Results**

You have successfully installed and activated extensions in your dev container using local .vsix files. The extensions are now immediately available for use.

## Installing GitHub Copilot in dev containers

As an admin, you can install the GitHub Copilot extension to enable AI-powered code suggestions, and chat assistance within your dev container.

**Before you begin**
You must have performed the following tasks:

- Gained access to edit the .devopsconfig control repository.
- Have a valid GitHub account with an active GitHub Copilot subscription.

**About this task**

This task describes how you can configure a dev container to support GitHub Copilot, and install and activate the extension within the running environment.

1. Configure the dev container image:
   a. Navigate to the .devopsconfig Control repository.
   b. Edit the existing devcontainer.json file (for example, within the Java dev container configuration), and update the image property to define the specific container version:

      ```
      "image": "baravich/devcontainer-java-copilot:2.0.000"
      ```

   c. Commit and push the changes to the repository.
2. Launch the dev container:
   a. In Code, locate the appropriate tile for the dev container configured in Step 1.
   b. Click the tile to launch the dev container.
3. Install the Copilot Extensions:

a. You must run a script to install the necessary binaries once the environment is running. Open a new Terminal in the dev container environment.
b. Copy and paste the following command to download and run the installation script:

```
curl -fsSL
 https://raw.githubusercontent.com/sunpix/howto-install-copilot-in-
code-server/refs/heads/main/install-copilot.sh
```

This script automatically installs both the GitHub Copilot and Copilot Chat extensions.

4. Authenticate and access the Copilot:
   a. Locate the GitHub Copilot icon in the activity bar or status bar, once the installation is complete.
   b. Select the icon and follow the prompts to sign in to your GitHub account.
   c. Authorize the extension to access your subscription.

**Results**

The GitHub Copilot and Chat extensions are now installed and active. You can now use AI assistance for code generation and technical questions directly within your development environment.

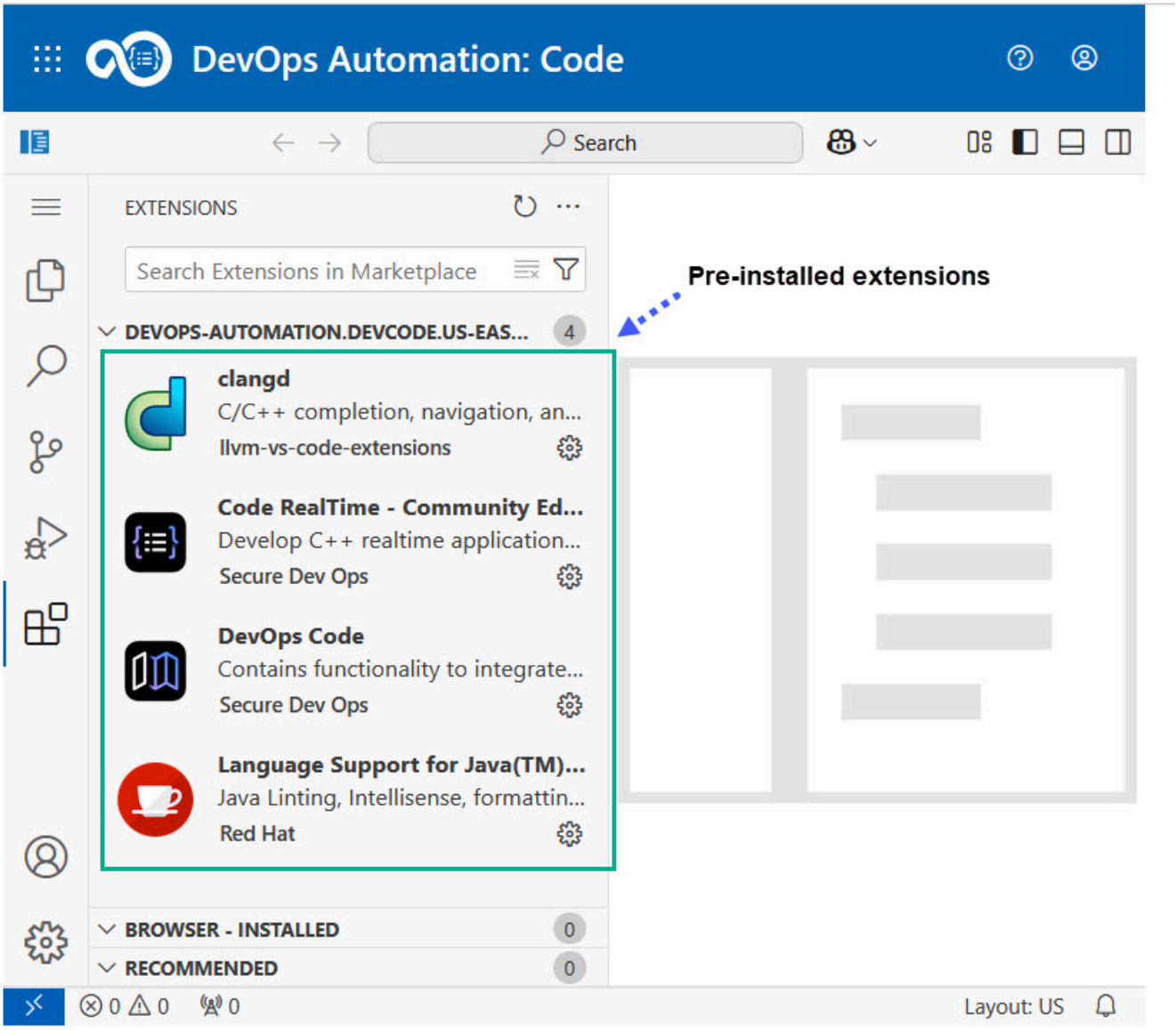
## Extensions

You can use extensions to specialize the Code IDE, integrating tools for coding, debugging, version control, and more. You can deploy them during initial project setups or to ensure workflow consistency in containerized and offline environments.

**Preinstalled extensions**

Your administrator can configure dev containers with preinstalled extensions during setup. These extensions are defined in the devcontainer.json file and sourced from the Open VSX Registry. For details on how to configure these in the devcontainer.js, refer to the Administrator Guide on page 120.

The following image shows a simple example of preinstalled extensions available in Code when you launch a dev container.




While working in your browser-based IDE, you can also perform the following tasks:

- Install additional extensions directly from the Open VSX Registry from the Extensions button in the left toolbar of the IDE.
- Install local VSIX files if needed.

**Note**: Any extensions you manually install is removed when the dev container is terminated. For more information, see Using dev containers on page 110.

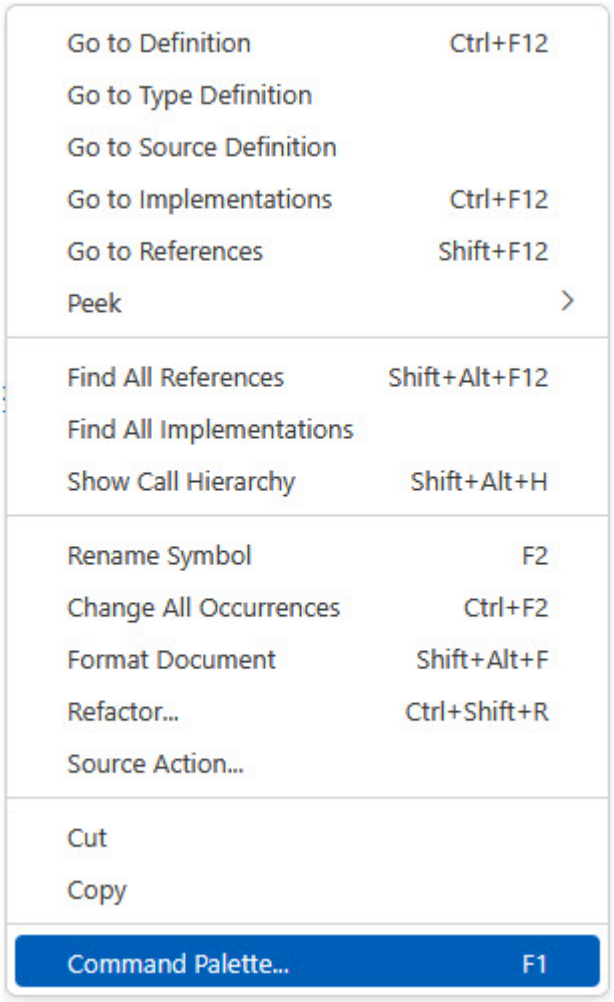
## Integrating Plan with Code

The DevOps Code extension allows integration with HCL DevOps Plan (Plan), enabling you to create and manage work items directly from your development environment in the browser IDE.

**About this task**

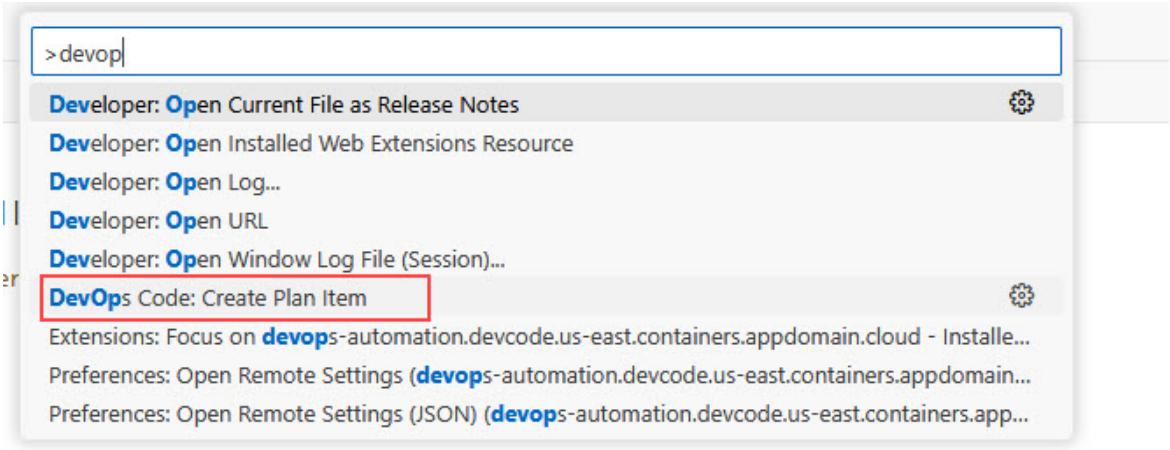
You can follow these steps to configure DevOps Code extension in the Code browser IDE, and create work items in HCL DevOps Plan (Plan).

1. Open the Command Palette.
   Right-click to open the context menu and select **Command Palette** in the editor.

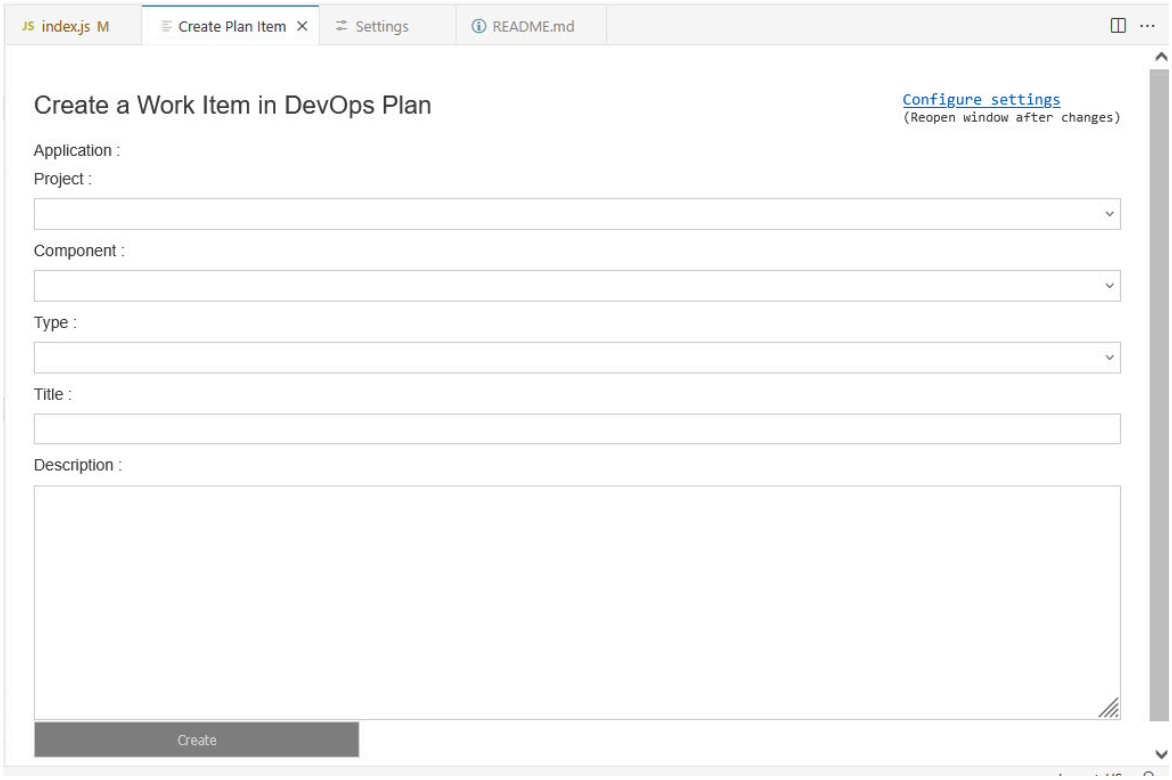



2. Access the DevOps Code Command.

a. Choose the Code command from the list of available commands.




The **Create Plan Item** window is displayed.



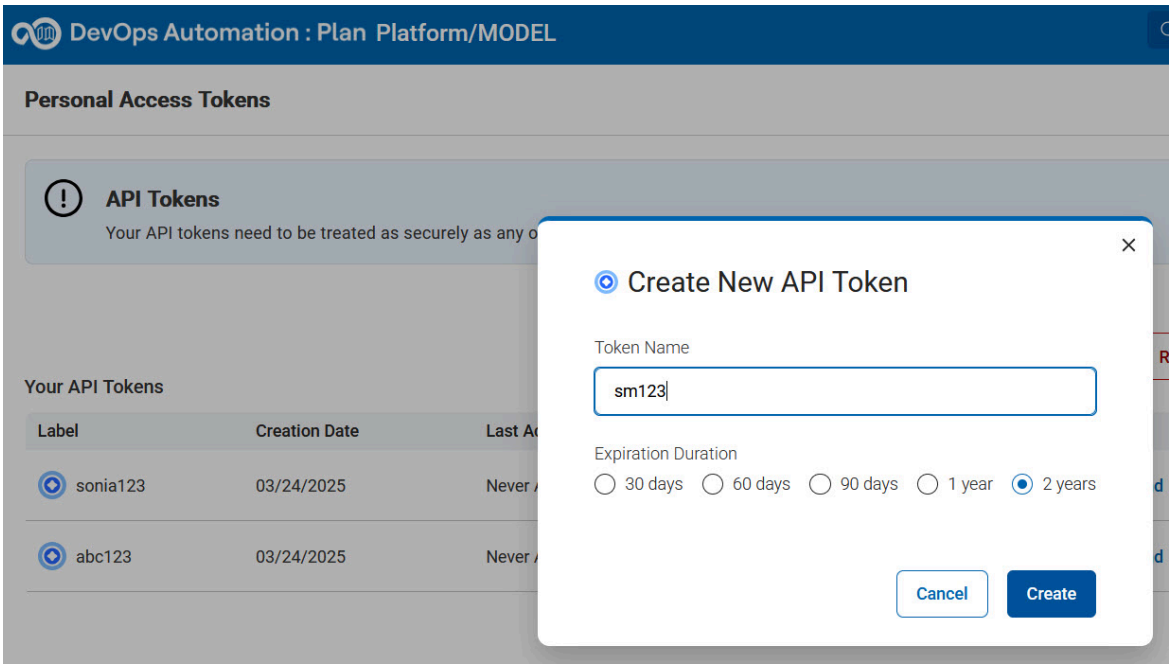

b. Click the **Configure settings** button to integrate DevOps Code with Plan for the first time.

The **Settings** window is displayed to add the Personal Access Token, Plan Server URL, and Team Space ID.

You can add these fields by navigating to the Plan platform, which is detailed in the next step.

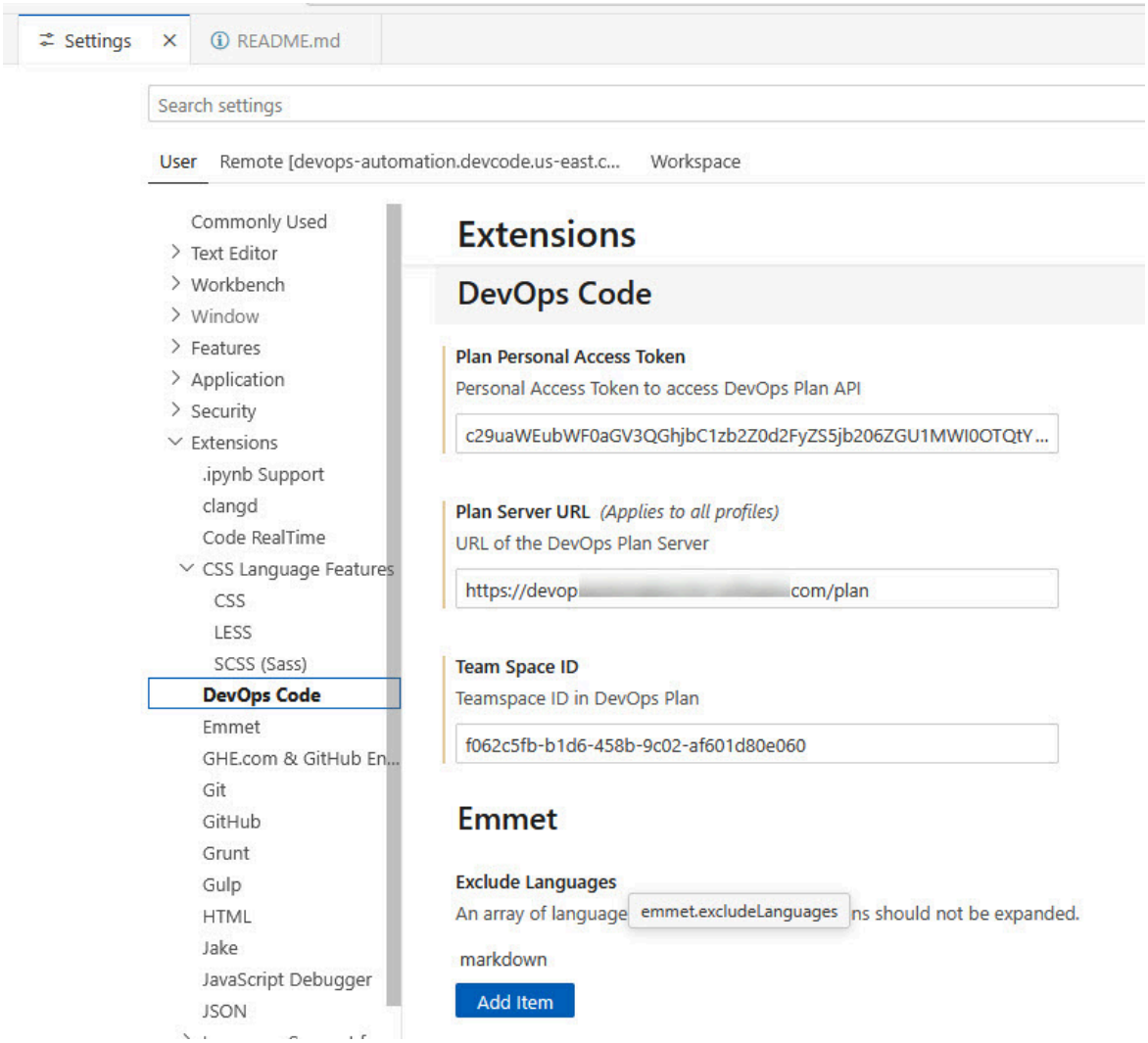
3. Perform the following steps to get configuration details from Plan:

a. Click the **Plan** button in the HCL DevOps Loop switcher to navigate to the Plan page.
b. Go to the **Settings** section of Plan, and enter the **Personal Access Token** (or API tokens) in the **Create New API Token** dialog.



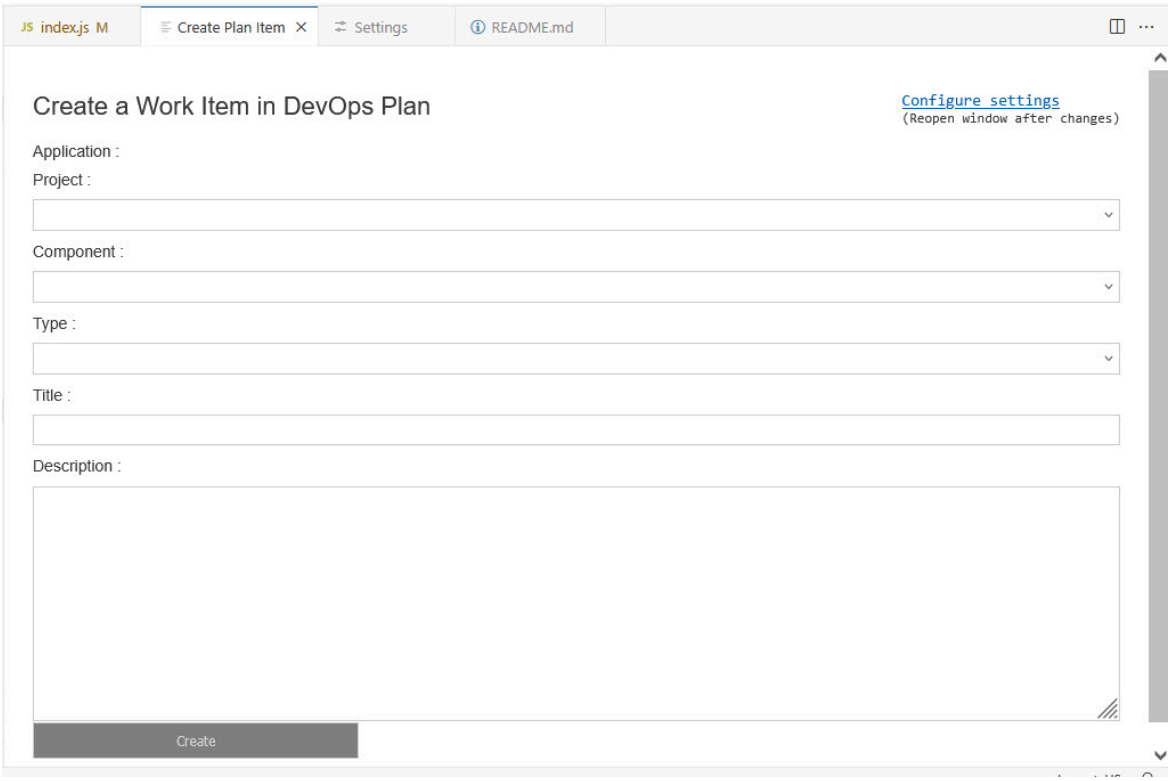

4. Perform the following steps to enter the configuration details:
   a. Navigate back to the Code application window from the switcher.
   b. Update the configuration details in the **Settings** window of Code.

**Note:** To find the team space ID to use, open a work item in Plan and look in the browser address bar for the text between the words "repos" and "databases". For example, 'f062c5fb-b1d6-458b-9c02-af601d80e060' in https://<DevOpsLoopURL>/plan/#/ccmweb/view/repos/f062c5fb-b1d6-458b-9c02-af601d80e060/databases/MODEL/records/WorkItem/MODEL00000179




5. Create a work item:
    a. Select the **DevOps Code** command from **Command Palette**.

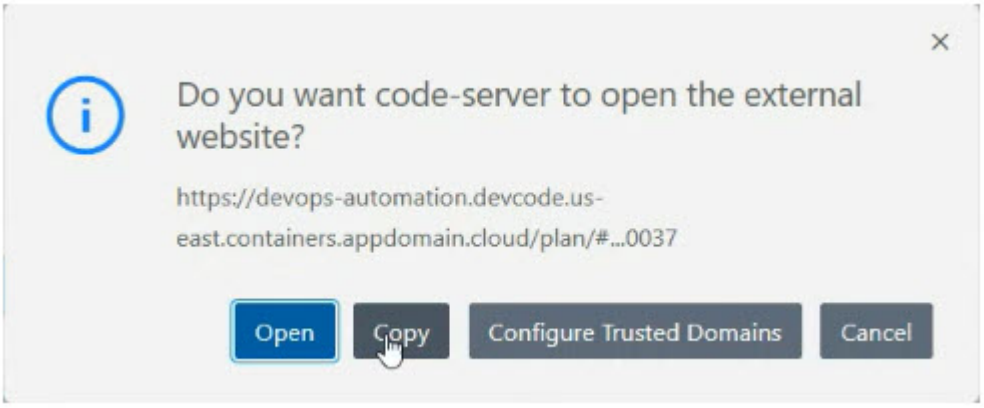
    The **Create Plan Item** window is displayed again.




b. Fill in the required fields such as Application Project, Component, Type, Title, and Description.
c. Click the **Create** button to create your work item in Plan.


A prompt is displayed at the end of the IDE to indicate that the work item has been created successfully.

6. Click **Open Work Item Link** to view the work item on the Plan page.
   A dialog is displayed to open the Plan web page.




7. Click **Open** to navigate to the work item page on the Plan web page.

**Results**

By following these steps, you can easily configure the Code extension and begin creating and managing work items directly within your development environment. This integration eliminates the need to switch between your Code IDE and Plan.

**Note:** Currently, the DevOps Code extension has the following limitations:

- At most one Plan application can be used. If you have multiple applications, the first one will be used.
- The Plan application must use the Agile business flow. Other flows are currently not supported.
- Your Plan project(s) must have at least one component.

## Access applications via automatic port forwarding

You can access applications running within your dev container from your local machine as Code automatically detects and forwards listening ports (7000–8000), to ensure uninterrupted interaction with containerized services.

**Note:** The dev container uses some ports internally and reserves the port range 7000 - 8000 for your application to use.

When Code detects that an application is running in the dev container and listening on a port, it automatically forwards the port, making it accessible outside the container. For example, you can connect to applications running within the dev container from your local machine.

../videos/auto-port-forwarding.mp4

## File management features

You can transfer files between your local computer and dev container, and simplify your development workflow.

The following section outlines the key features provided by Code for file management:

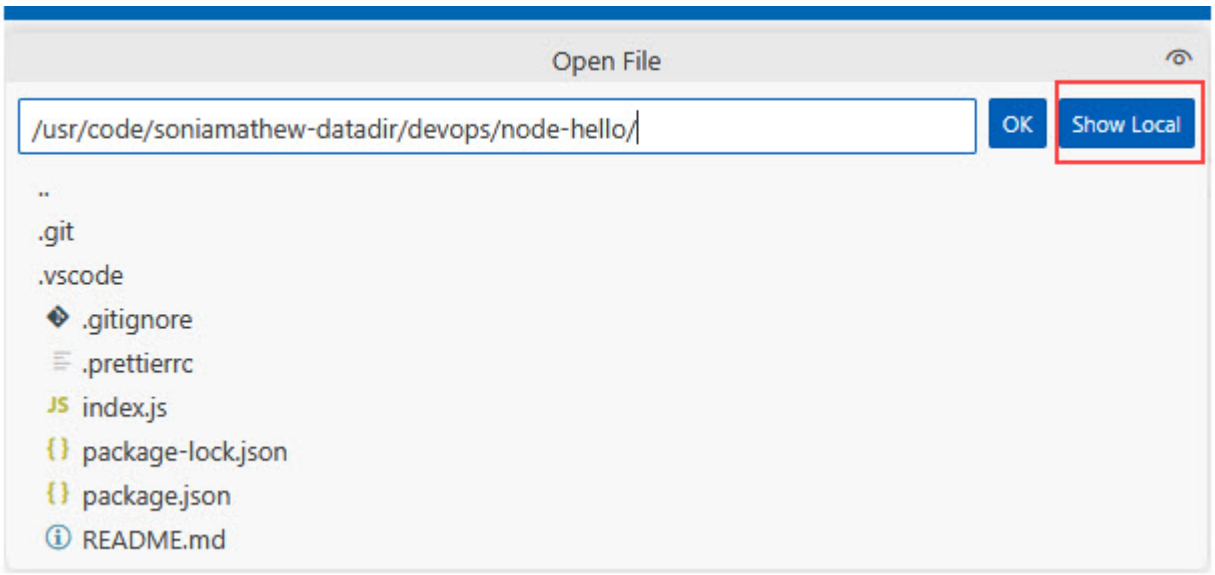
**Drag and Drop**

Code provides a drag-and-drop feature that simplifies file management, so that you can effortlessly transfer files between your local computer and the dev container. You can simply drag files from your local system and drop them into the Explorer view of the Code IDE to add them to a workspace folder.

../videos/drag-drop.mp4

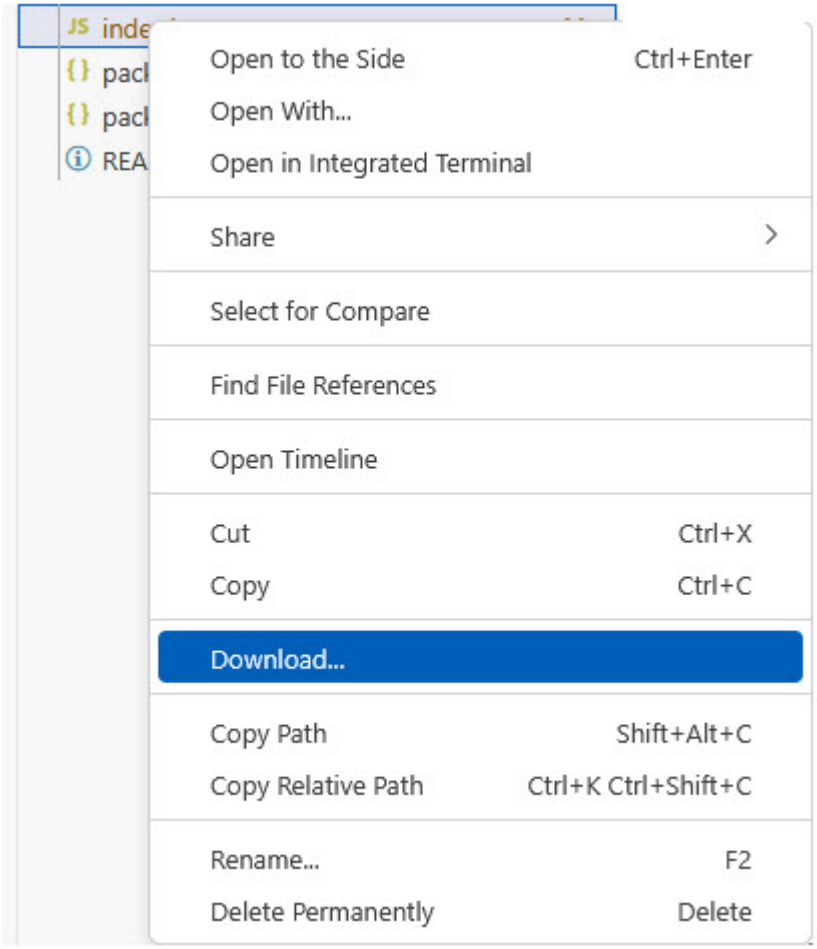
**Browse local files**

Code provides the **Show Local** button feature, which allows you to browse files from your local machine. This gives you the option to choose between using the file system of your local computer or that of the dev container.




**Download**

Code provides the **Download** option to easily download files and folders from the server IDE to your local system. Simply right-click on the required file or folder, select the **Download** option, and the file will be transferred to your local machine for backup.




# Control

Control manages source code within the DevOps Loop using GitHub compatible APIs that integrate with existing developer tools, workflows, and automation. It supports version management, collaboration, and code governance as development progresses.

For more details refer to the Control Documentation.

## Managing Git authentication for developers

In this topic, you will learn how to manage authentication when using Git on your system to access Control repositories within DevOps Loop deployment.

There are three types of credentials you can use with Control in a Loop deployment:

- **SSH keys**: For information, see the Using an SSH key (for Desktop) section in the Control Documentation.
- **Personal access token via HTTPS**: For information, see the Using HTTPS (not with OAuth2 support) section in the Control Documentation.

- **OAuth credentials with Git-Credential-Manager via HTTPS**: To configure your Git client in the development environment for HTTPS communications with Control in a DevOps Loop deployment, add settings similar to the following example using the Git command line. Replace *`yourhost.example.com`* with your Loop server's hostname:

```
git config --global credential.https://yourhost.example.com.provider generic
git config --global credential.https://yourhost.example.com.oauthClientId git-credential-manager
git config --global credential.https://yourhost.example.com.oauthTokenEndpoint /auth/realms/devops-automation/protocol/openid-connect/token
git config --global credential.https://yourhost.example.com.oauthDeviceEndpoint /auth/realms/devops-automation/protocol/openid-connect/auth/device
git config --global credential.https://yourhost.example.com.oauthAuthorizeEndpoint /auth/realms/devops-automation/protocol/openid-connect/auth
git config --global credential.https://yourhost.example.com.oauthAuthModes DeviceCode
```

This configuration ensures secure and efficient access to your repositories.

# Build

Build is a build management platform within the DevOps Loop that supports application development, continuous integration, and artifact storage. Its template-driven approach helps teams standardize processes, enforce policies, and integrate testing into their workflows.

For more details refer to the Build Documentation.

## Configuring an external agent for Build in DevOps Loop

You can configure an external Build agent to connect with DevOps Loop installed on Kubernetes (K8s), Azure Kubernetes Service (AKS), IBM Kubernetes Service (IKS), or OpenShift Container Platform (OCP).

**Before you begin**

Before you begin, ensure the following prerequisites are met:

- Installed DevOps Loop on Kubernetes (K8s), Azure Kubernetes Service (AKS), IKS, or OCP.
- Ensured that you have access to the Kubernetes or OpenShift cluster where DevOps Loop is running.
- Installed the DevOps Build agent on a supported external platform.
- Been granted permissions to edit services or create routes in the cluster.

1. Edit the Emissary ingress service to expose the Build agent port for Kubernetes (K8s), Azure Kubernetes Service (AKS), or IKS deployments.

    a. Run the following command:

    ```
    kubectl edit svc emissary-ingress -n emissary
    ```

    b. Under the *spec.ports* section, add the following configuration:

    ```
    - name: build-agent
    nodePort: 31123
    port: 7920
    ```

```
protocol: TCP
targetPort: 7920
```

2. Update the *installed.properties* file of the agent with the cluster domain and port.

```
locked/agent.brokerUrl=failover:(ah3://<service-domain>:31123)
locked/agent.jms.remote.host=<service-domain>
locked/agent.jms.remote.port=31123
```

**Result**

You must replace *<service-domain>* with the domain exposed in your cluster setup.

3. Start the external Build agent.

**Result**

The agent is displayed under the **Agents** section in the Build UI.

4. For IKS, configure the cluster service domain, if it is not configured.

Follow step 12 on page 33 in the **Installing DevOps Loop on IBM Cloud Kubernetes Service (IKS)** topic for setting up the service domain.

5. For OpenShift deployments, create a route to expose the Build agent service.

a. Apply the following route definition:

```
cat <<EOF | oc apply -n <namespace> -f -
apiVersion: route.openshift.io/v1
kind: Route
metadata:
name: devops-build-agent
spec:
to:
kind: Service
name: devops-build-server
port:
targetPort: agent
tls:
termination: passthrough
insecureEdgeTerminationPolicy: Redirect
EOF
```

b. Retrieve the route host by running the following command:

```
oc get route devops-build-agent
```

c. Update the *installed.properties* file of the agent as follows:

```
locked/agent.brokerUrl=failover:(ah3://<route-host>:443)
locked/agent.jms.remote.host=<route-host>
locked/agent.jms.remote.port=443
```

d. Start the external Build agent.

**Result**

The agent is displayed under the **Agents** section in the Build UI.

**Results**

You have configured the external Build agent and connected it to DevOps Loop.

**What to do next**

You can now run jobs on this agent.

## Integrating Build resources with existing loop and teamspaces

You can integrate the previously created Build resources, including teams, mappings, and agent pools to an existing teamspace and loop after you upgrade from DevOps Loop 1.0.2 to 1.0.3 or later.

**Before you begin**

You must have completed the following tasks:

- Ensured that you have the administrator privileges in DevOps Loop.
- Ensured that you have a valid bearer token.
- Ensured that you have a loop ID, teamspace ID, and loop name.

1. Create a team in Build by using the following API call:

```
curl -kX PUT "${LOOPDOMAIN}/build/rest2/team/loop/${loopId}" \
-H "Authorization: Bearer ${BEARER_TOKEN}" \
-H "Content-Type: application/json" \
-d '{
"name": "${LoopName}",
"description": "",
"type": "Loop",
"teamspaceId": "${teamspaceId}",
"roleMappings": [
{
"role": "Platform User",
"user": "${UserEmailAddress}",
"authenticationRealm": "Platform_build_OIDC"
}
]
}'
```

   Replace all variables with actual values:

   - **loopId**: Navigate to **Deploy > Settings > Teams**. Select the team created during loop initialization and loop ID can be found at the end of the URL.
   - **teamspaceId**: Navigate to **Deploy > Settings > Teams**. Select the team created during teamspace creation and teamspace ID can be found at the end of the URL
   - **LoopName**: Use the loop's name, without any prefix.
   - **BEARER_TOKEN**: Use a valid bearer token.

   You can assign additional members in the `roleMappings` array. Supported roles include `Platform User` and `Platform Admin`.

2. Perform the following steps to map resources to the newly created team in Build:

a. Click **Teams** in the left navigation panel.
   The **Teams** tab is displayed with the list of teams.

b. Select the newly created team.

c. Map the following resources in the **Team Objects Mappings** section:
   - Project
   - Project Process
   - Job
   - Repository
   - Template
   - Agent Pool

d. Select each resource from the drop-down, click **Add**, select the newly created team, and click **OK**.

3. Add the required users to the new team.
4. Configure the agent and assign it to the appropriate agent pool.
5. Click **Retry** for the failed teamspaces after upgrading from DevOps Loop 1.0.2 to DevOps Loop 1.0.3 or later to complete the teamspace creation.

**Results**

The Build environment is now successfully integrated into the previously created loops and teamspaces after upgrading from DevOps Loop 1.0.2 to DevOps Loop 1.0.3 or later.

---

Related information

Configuring Agents

Assigning users to teams

# Test

Test Hub centralizes test data, environments, test runs, and reports in a single web based interface within the DevOps Loop. It streamlines test management, execution, and tracking, providing greater efficiency and visibility to ensure quality throughout the loop.

For more details refer to the Test Hub Documentation.

## Integration of Test Hub with Measure

When you want to export the test results to Measure, you can use the Test Hub plugin to integrate Test Hub with Measure. After you complete the integration, you can run test assets, and then view the metrics on the Measure dashboard.

### Prerequisites

You must have completed the following tasks in Measure before proceeding with the tasks to be performed in Test Hub:

1. Added Test Hub integration in Measure. For more information, refer to the Test Hub plugin documentation.
2. Copied the URL that is displayed on the **Integrations** page in Measure and saved it for later use. Here is the URL sample: https://10.0.2.15.nip.io:9443/reporting-consumer/pluginEndpoint/67eaa3c12b5778a2d1332269/onetest/callback.

### Steps to perform in Test Hub

You can perform the following tasks when you want to export test results to your dashboard in Measure:

1. Configure a webhook for Measure on page 144.
2. Add additional parameters for test runs on page 146.

## Configuring a webhook for Measure

When you want to export test results to Measure after the test run, you can do so by configuring a webhook to receive the customized results.

#### Before you begin

You must have completed the following tasks:

- Read and fulfilled the prerequisites that are listed in Integration of Test Hub with Measure.
- Read and understood Configuration of a webhook.
- Read and understood the configuration of a webhook template. For more information, refer to Configuring a webhook template.
- Read and understood the configuration of a server webhook. For more information, refer to Configuring a server webhook.

#### About this task

You can set up webhooks to send notifications to various messaging apps such as Microsoft Teams, Slack, Measure, or to any application that can receive an HTTP POST request, whenever specific events happen on Test Hub. For instance, a webhook can be triggered when a test run fails. After the event, the payload that is received by the application contains the details of the event, including the additional parameters that you added. If you want the result events to be correlated when using systems such as Measure, you must add the properties that are present in the webhook template as additional parameters in the **Execute test asset** window.

1. Log in to Test Hub.
   The **Projects** page of the initial team space is displayed.
2. Click **My projects > *project_name*** to open the project that contains the test assets.
   The **Overview** page of the project is displayed.
3. Click **Manage > Webhooks**.
   The **Templates** tab is displayed.
4. Click **New Template**.
   The **New template** page is displayed.

5. Perform the following actions on the **DETAILS** tab.
    a. Enter a unique name for the template.
    b. Create a new channel type by clicking the **Create new channel type ...** option.

       The **New channel type** dialog is displayed.

    c. Enter a unique name for the channel type, and then enter a description for the channel type.
6. Click **Apply**.
   The channel type is created and added to the **Channel Type** list.
7. Click **This template is suitable for events**, and select only the following events from the **Execution Events** list:
    - Execution Stopped Manually
    - Execution Completed with Verdict Pass
    - Test and Suite Errors
8. Click the **Template** tab.
9. Click the **infinity** icon  to insert a Measure sample template in the **Template body**.
   A message is displayed about the action that inserts text in the template body.
10. Click **Ok**.
11. The text is inserted in the **Template body**.

    **Note:** Remove `#if( $commit || $buildId || $buildUrl)` from the webhook template if you are not passing either `commit`, `buildId` or `buildUrl` as additional parameters during test execution in Test. If you do not remove the webhook is not triggered.

12. Click **Apply**.
    The template is created and added to the **Templates** tab.
13. Click the **Webhooks** tab, and then click **New Webhook**.
    The **New Webhook** page is displayed.
14. Enter a name for the webhook in the **Name** field.
15. Paste the URL that you copied earlier from the **Integrations** page of Measure in the **Webhook URL** field.
16. Click the**Channel Type**, and then select the channel type that you just created from the list.
17. Enter a description for the webhook in **Description**.
18. Select the template that you just created from the drop-down list.
19. Click **Apply**.
    The webhook is created and added to the **Webhooks** tab.

**Results**

You have configured a webhook to export the test results to Measure after the test run is complete.

**What to do next**

You can configure the test run and run the test to export the result summary to Measure. See Adding additional parameters for test runs on page 146.

## Adding additional parameters for test runs

When you want to add additional parameters to be included in test results, you can do so by defining the parameters.

**Before you begin**

You must have completed the following tasks:

- Read and fulfilled the prerequisites that are listed in Integration of Test Hub with Measure.
- Configured a webhook for Measure. See Configuring a webhook for Measure on page 144.

1. Click **Execution** in the navigation pane.
2. Select the test that you want to run and click  in the **Actions** column to open the **Execute test asset** window.
3. Click **Advanced settings**.
4. Enter the following parameters and values in **Additional Configuration Parameters**:

| Parameter name | Value |
|---|---|
| result.property.buildId | Enter the build ID for which you are running the test in the second edit field as the parameter value. For example, 1.13.4. |
| result.property.buildUrl | Enter the build URL for which you are running the test in the second edit field as the parameter value. For example, `https://10.10.110.112.nip.io/build/tasks/project/build_test123 .` |

5. Click **Add** for each parameter.
6. Click **Execute**.

**Results**

The additional parameter values are captured while your test runs, and the test results are exported to Measure after the test run is complete.

**What to do next**

You must perform the following tasks to view the test results in the dashboard of Measure:

1. Create a dashboard in the **Insights** section. For more information, refer to Creating dashboards.
2. Add a chart to the newly created dashboard, and when you click **Add charts**, you must select Functional Tests, Unit Tests, API Tests, and Performance Tests from the **Quality** section. For more information, refer to Adding a chart.

# Release

Release standardizes and automates the software release lifecycle with the DevOps Loop, moving applications smoothly from preproduction to production. It provides predictable scheduling, integrates changes from planning tools, and gives stakeholders clear visibility into plans and milestones.

For more details refer to the Release Documentation.

# Deploy

Deploy standardizes and simplifies the rollout of software components across environments withing the DevOps Loop. Its release automation tools improve deployment speed and reliability, providing visibility into multi-tiered deployments and orchestrating complex processes across environments and approval gates.

For more details refer to the Deploy Documentation.

## Installing an external agent for Deploy in DevOps Loop

You can install and connect an external Deploy agent to a DevOps Loop instance running on Kubernetes (K8s), and IBM Kubernetes Service (IKS).

**Before you begin**

Ensure that the following prerequisites are met:

- Installed DevOps Loop on Kubernetes (K8s), or IKS.
- You have access to the Kubernetes cluster where DevOps Loop is running.

1. Obtain the Deploy web agent communication URL for the Deploy server running in the DevOps Loop instance.
   In a Kubernetes cluster, the URL is indicated as: `wss://{{.Values.global.domain}}:7919`

   Here, the value for the `Values.global.domain` is the same URL used to access the DevOps Loop UI.
2. Install the Deploy agent using the Deploy web agent communication URL obtained in step 1.
   During the agent installation, you must specify any teams you want the Deploy agent to be included in. Refer to Installing agents for more information on Deploy agent installation.
3. Start the external Deploy agent.
   **Result**
   The agent is displayed under the **Agents** section in the Deploy UI.
   **Troubleshooting**
   If you encounter issues connecting the Deploy agent to the Deploy server running in DevOps Loop, verify that the Emissary Ingress service is configured to route traffic on port 7919 to the Deploy server WSS service:
   a. Run the following command to inspect the service:

   ```
   $ kubectl get svc -n emissary
   ```

   A correct output lists 7919 in PORT(S) column, like the following:

   ```
   NAME                    TYPE         CLUSTER-IP     EXTERNAL-IP PORT(S)
                           AGE
   emissary-ingress        LoadBalancer 172.21.20.31   xx.xx.xx.xx
    443:31734/TCP,80:32086/TCP,7919:31941/TCP,9022:31708/TCP 112d
   emissary-ingress-admin  ClusterIP    172.21.9.14    <none>      8877/TCP,8005/TCP
                           112d
   ```

   b. If the port 7919 is not listed for the emissary-ingress service, edit the service configuration:

   ```
   $ kubectl edit svc emissary-ingress -n emissary
   ```

c. Under the `spec.ports` section, add the following lines:

```
- name: deploy-wss
port: 7919
protocol: TCP
targetPort: 7919
```

After port 7919 is configured in the emissary-ingress service, your Deploy agent should successfully connect to the Deploy server running in DevOps Loop.

**Results**

You have installed and connected the external Deploy agent to DevOps Loop.

**What to do next**

You can now use this agent to run deployments.

## REST commands for Deploy

This set of reference topics provides additional information on using the command line interface for tasks related to teamspaces and loops in Deploy.

**Example**

### Creating applications in a loop from a JSON file

#### Prerequisite

To create applications in a loop, you must have any one of the following permissions:

- **Manage Team Spaces** permission.
- **Manage Loops** permission and are a member of the teamspace.

#### Request

```
PUT https://{hostname}:{port}
    /cli/application/create
Accept: {contentType}
```

**Table 6. Header parameters**

| Parameter | Type | Required | Description |
|---|---|---|---|
| **Accept** | `application/json` | true | |

This command takes a JSON request string or file. You must use the following template for the request:

**Example**

### Example JSON request with loop name

```
{
  "loop": {
    "name": "Loop 1",
    "teamSpace": "Team space 2"
  },
  "name": "My Application"
}
```

**Example**

### Example JSON request with loop ID

```
{
    "loop": "1967ea12-9651-128d-4216-0f9a23522dd0",
    "name": "My Application", // application name
}
```

**Example**

### Example command

```
curl -k -u username:passwd -X PUT https://{hostname}:{port}/deploy/cli/application/create -H 'Content-Type:
 application/json' --data-binary @create-application.json
```

## Get an application in a loop

### Request

```
GET https://{hostname}:{port}
    /cli/application
Accept: {contentType}
```

**Table 7. Header parameters**

| Parameter | Type | Required | Description |
|---|---|---|---|
| **Accept** | `application/json` | true | |

**Table 8. Required parameters**

| Parameter | Type | Required | Description |
|---|---|---|---|
| **application** | string | true | Name or ID of the application in the following format:<br><br>`teamspaceName~loopName:applicationName` |

**Example**

### Example command to get an application by using application ID

```
curl -k -u username:passwd -X GET https://{hostname}:{port}/deploy/cli/application
     -application "27d56999-91e1-4689-914a-e6c932a75234"
```

#### Example

### Example command to get an application by using application name in loop by ID

```
curl -k -u username:passwd -X GET https://{hostname}:{port}/deploy/cli/application
     -application "6d076690-8a79-4fbc-990a-4852fb857d25:applicationName"
```

**Note:** When using loop ID, you do not need to specify the teamspace name or ID. If you do, make sure that the teamspace name or ID is valid.

#### Example

### Example command to get an application by using application name in loop by name in teamspace by ID

```
curl -k -u username:passwd -X GET https://{hostname}:{port}/deploy/cli/application
     -application "6d076690-8a79-4fbc-990a-4852fb857d25~loopName:applicationName"
```

#### Example

### Example command to get an application by using application name in loop by name in teamspace by name

```
curl -k -u username:passwd -X GET https://{hostname}:{port}/deploy/cli/application
     -application "teamspaceName~loopName:applicationName"
```

#### Example

### Example response


```
[
    {
        "id": "1967ea12-9651-128d-4216-0f9a23522dd0",
        "securityResourceId": "1967ea12-9611-fde1-7dd3-fb62aef3d398",
        "loop": {
            "id": "1234bdb3-90a1-414d-88b3-1ff19e822ded",
            "name": "sampleLoop3",
            "teamSpace": {
                "id": "d0ff4b8a-ab83-4cca-9c81-78daab66db27",
                "name": "samplets3"
            }
        },
        "name": "My Application",
        "created": 1745881213285,
        "enforceCompleteSnapshots": false,
        "active": true,
        "tags": [],
        "deleted": false,
        "onlyChangedVersions": true,
        "useWizard": false,
        "user": "demo user (admin@admin.com)"
```

```
    }
]
```

## Creating components in a loop from a JSON file

### Prerequisite

To create components in a loop, you must have any one of the following permissions:

- **Manage Team Spaces** permission.
- **Manage Loops** permission and are a member of the teamspace.

### Request

```
PUT https://{hostname}:{port}
    /cli/component/create
Accept: {contentType}
```

**Table 9. Header parameters**

| Parameter | Type | Required | Description |
|---|---|---|---|
| **Accept** | `application/json` | true | |

This command takes a JSON request string or file. You must use the following template for the request:

**Example**

### Example JSON requests with loop name

```
{
    "loop": {
          "name" : "Loop 1",
          "teamSpace" : "Team space 2"
            },
    "name": "My Component",
}
```

**Example**

### Example JSON request with loop ID

```
{
    "loop": "1967ea12-9651-128d-4216-0f9a23522dd0",
    "name": "My Application",
}
```

**Example**

### Example command

```
curl -k -u username:passwd -X PUT https://{hostname}:{port}/deploy/cli/component/create -H 'Content-Type:
 application/json' --data-binary @create-component.json
```

## Get a component in a loop

### Request

```
GET https://{hostname}:{port}
    /cli/component
Accept: {contentType}
```

**Table 10. Header parameters**

| Parameter | Type | Required | Description |
|---|---|---|---|
| **Accept** | `component/json` | true | |

**Table 11. Required parameters**

| Parameter | Type | Required | Description |
|---|---|---|---|
| **component** | string | true | Name or ID of the component in the following format:<br><br>`teamspaceName~loopName:applicationName` |

**Example**

### Example command to get a component by using component ID

```
curl -k -u username:passwd -X GET https://{hostname}:{port}/deploy/cli/component
     -component "27d56999-91e1-4689-914a-e6c932a75234"
```

**Example**

### Example command to get a component by using component name in loop by ID

```
curl -k -u username:passwd -X GET https://{hostname}:{port}/deploy/cli/component
     -component "6d076690-8a79-4fbc-990a-4852fb857d25:componentName"
```

**Note:** When using loop ID, you do not need to specify the teamspace name or ID. If you do, make sure that the teamspace name or ID is valid.

**Example**

### Example command to get a component by using component name in loop by name in teamspace by ID

```
curl -k -u username:passwd -X GET https://{hostname}:{port}/deploy/cli/component
     -component "6d076690-8a79-4fbc-990a-4852fb857d25~loopName:componentName"
```

**Example**

### Example command to get a component by using component name in loop by name in teamspace by name

```
curl -k -u username:passwd -X GET https://{hostname}:{port}/deploy/cli/component
     -component "teamspaceName~loopName:componentName"
```

**Example**

### Example response


```
[
    {
        "id": "1967ea55-d829-fb58-d2d9-f5cc351c50a9",
        "securityResourceId": "1967ea55-d7e1-7303-7ae7-7fc579cd6d0f",
        "loop": {
            "id": "1234bdb3-90a1-414d-88b3-1ff19e822ded",
            "name": "sampleLoop3",
            "teamSpace": {
                "id": "d0ff4b8a-ab83-4cca-9c81-78daab66db27",
                "name": "samplets3"
            }
        },
        "name": "My Component",
        "description": "New component for command example",
        "created": 1745881488769,
        "componentType": "STANDARD",
        "ignoreQualifiers": 0,
        "importAutomatically": false,
        "useVfs": true,
        "active": true,
        "integrationFailed": false,
        "deleted": false,
        "defaultVersionType": "FULL",
        "cleanupDaysToKeep": 0,
        "cleanupCountToKeep": 0,
        "tags": [],
        "user": "demo user (admin@admin.com)"
    }
]
```


# Measure

Measure provides end-to-end visibility into the DevOps loop, giving teams to track how value is created throughout the work flow. It consolidates testing, security, and performance metrics to identify bottlenecks, team issues, and opportunities for improvement while strengthening governance across the organization.

For more details refer to the Measure Documentation.

# Chapter 7. Loop Genie - Tech Preview

**Disclaimer:**

This release contains access to the Loop Genie feature in HCL DevOps Loop as a Tech Preview. The Tech Preview is intended for you to view the capabilities of Loop Genie offered by HCL DevOps Loop, and to provide your feedback to the product team. You are permitted to use the information only for evaluation purposes and not for use in a production environment. HCL provides the information without obligation of support and "as is" without warranty of any kind.

Loop Genie is an AI-powered, multi-agent chatbot integrated into DevOps Loop. It performs tasks and retrieves information across multiple capabilities through a single conversational interface.

Loop Genie supports multi-step queries that can involve multiple tools in a single request. You can perform actions such as creating work items, retrieving work item counts, creating branches, listing repositories, and accessing branch commit history, all within one prompt. Loop Genie also produces formatted responses for better readability, replacing raw JSON outputs.

Loop Genie connects with built-in agents to handle prompts securely and efficiently. You can interact with Loop Genie through text or voice input, making it easier to manage teamspaces, add or remove users, create branches, edit files, and manage pull requests without navigating multiple interfaces.

Loop Genie's AI capabilities require or support integration with **OpenAI**, **IBM watsonx**, **Claude**, and **Gemini**. These integrations provide natural language understanding, context-aware responses, and advanced automation, enabling Loop Genie to perform tasks and retrieve information efficiently across your DevOps Loop environment.

Loop Genie supports multi-step queries that can use multiple capabilities and tools within a single request.

### Sample multi-step workflow execution in Loop Genie:

**Scenario:** You want to create a new work item in DevOps Loop, assign it to a user, and then fetch its status all in a single request.

**Prompt to Loop Genie:**

> "
>
> Create a new story in the `Payment API` project titled 'Implement new refund feature', assign it to `alice@example.com`, and then get the status of the newly created work item."
>
> "

**How Loop Genie handles the execution:**

1. **Step 1 – Create Work Item**
    - Calls `plan_create_work_item` with project name, title, and optional fields.
    - Returns the new work item ID.
2. **Step 2 – Assign User**
    - Calls `plan_update_work_item` (or equivalent MCP command) to assign the work item to `abc@example.com`.
3. **Step 3 – Retrieve Status**
    - Calls `analytics_get_work_item_status` with the work item ID to fetch the current status.

**Output returned by Loop Genie:**

| Field | Value |
|---|---|
| Work Item ID | 12345 |
| Title | Implement a new refund feature |
| Assigned To | abc |
| Status | Open |

# Interacting with Loop Genie

You can use Loop Genie, an AI assistant, to retrieve information or perform tasks across multiple capabilities and loops within DevOps Loop. The results or actions are displayed directly in the chat interface.

**Before you begin**

You must have completed the following tasks:

- Integrated an AI provider on the **Integrations** page. See Configuring OpenAI integration on page 68, Configuring IBM watsonx integration on page 72, Configuring Gemini integration on page 71, Configuring Claude Desktop integration on page 70.
- Created a loop and ensured that there are certain tasks and activities performed in the different solutions within the loop. See Creating a loop on page 89.

1. Click the 

button on the dashboard of a loop on the **My Loops** page.
**Result**
A chat interface is displayed with a greeting message: *"Hello test admin, How can I help you today?"*
2. Enter your prompt or query in the chat box.

Alternatively, you can click the  icon to enter your query through a voice search. You must allow the browser to use the microphone by accepting the notification when prompted.

If you have configured Loop Genie with **IBM watsonx** integration, use **# commands**. Using # commands ensures that Loop Genie correctly routes the request to the intended tool.

**Note:** Loop Genie allows a single prompt to execute multiple actions across various capabilities sequentially.

3. Select the **Focus** option to filter search results based on their scope.
   Available options include **On All**, **On Loop**, **On Control**, and **On Test**.
4. Click **Send**.
   **Result**
   Loop Genie processes your query and either displays the results or performs the requested action.

   **Note:** The output format might vary between AI providers, which can affect how responses are displayed in Loop Genie. The results from **OpenAI** and **IBM watsonx** may differ in structure or presentation.

5. **Optional:** Click the  icon to download the conversation in a `.json` format for later reference.
6. **Optional:** Click  to close the Loop Genie pane.

   **Note:** When you close Loop Genie, the chat session is terminated, and you cannot view the conversation history.

**Results**

You have interacted with Loop Genie to retrieve information or perform tasks across DevOps Loop capabilities.

---

Related reference

Prompt references for Loop Genie on page 156

# Prompt references for Loop Genie

You can use the following prompts to interact with Loop Genie, which will respond and perform the specified actions on your DevOps Loop platform.

**Example**

**Note:** These commands are for reference, and you can customize them based on your environment and version.

**Example**

Loop Genie supports multi-step workflows. A single prompt can involve multiple capabilities, and Loop Genie will execute them sequentially as part of a single request.

**Example**

The following table shows the scope and sample prompts for various modules in DevOps Loop:

| Module | Scope | Prompt examples |
|---|---|---|
| Loop | Manage loops, projects, users, test projects, and control repositories within a loop. | • List all projects under Loop “TeamAlpha”.<br>• Add user “jane.doe@example.com” to Loop “TeamAlpha”.<br>• Attach Test Project “QA-Release1” to Loop “TeamAlpha”.<br>• Add repository “devops-repo” to Loop “TeamAlpha”. |
| Loop Teamspaces | Manage teamspace, teamspace users, and invitations across teamspaces. | • Show all teamspaces in DevOps Loop.<br>• List all users in Teamspace “Engineering”.<br>• Invite “mark.smith@example.com” to Teamspace “Engineering”. |
| Loop (Plan) | Manage planning activities like components, work items, and applications. | • List all components for Plan ID “Plan123”.<br>• What work item types are supported in Plan “Plan123”?<br>• Show work items for Project “InventoryApp”.<br>• List applications under Plan “Release2.0”.<br>• Create a Bug work item “Fix login issue” in Project “InventoryApp”.<br>• Delete work item ID “4567” from Plan “Plan123”. |
| Loop (Opensearch) | Search and analyze work items, story points, and statuses. | • Search for work items containing “API Bug” in Sprint “Sprint7”.<br>• Show the status of Work Item ID “789”.<br>• How many open work items exist in Plan “Plan123”?<br>• Show total story points for Sprint “Sprint5”. |
| Loop (Testhub) | Manage and execute tests and test projects. | • List all Testhub projects for Loop “QA-Team”.<br>• List all tests in Project “ReleaseSmokeTests”.<br>• Run Test ID “T101” in Project “ReleaseSmokeTests”. |

| Module | Scope | Prompt examples |
|---|---|---|
| | | • Show results for Test ID “T101” in Project “ReleaseSmokeTests”.<br>• Get Testhub result for ID “5555”. |
| Loop (Control) | Manage repositories, branches, tags, releases, issues, pull requests, and searches. | • Show details of the authenticated user.<br>• List all organizations linked to my account.<br>• Fork repo “main-project” to my personal workspace.<br>• Show all repositories I own.<br>• Create branch “feature-123” in repo “backend-service”.<br>• List branches in repo “frontend-app”.<br>• Create release “v1.0” in repo “backend-service”.<br>• Get details of release “v1.0” in repo “frontend-app”.<br>• List all releases for repo “api-service”.<br>• Create tag “v2.0” in repo “frontend-app”.<br>• Get details of tag “v1.0-beta” in repo “backend-service”.<br>• List tags in repo “frontend-app”.<br>• Show commits in repo “backend-service”.<br>• Get content of “README.md” in repo “frontend-app”.<br>• Update file “config.yaml” in repo “api-service”.<br>• Get issue #12 from repo “frontend-app”.<br>• List all issues in repo “backend-service”.<br>• Create issue “Fix API bug” in repo “api-service”.<br>• Edit comment on issue #12 in repo “frontend-app”.<br>• Show comments for issue #34.<br>• List pull requests for repo “backend-service”.<br>• Create PR from branch “feature-123” to “main” in repo “api-service”.<br>• Search for team “DevOps” in Org “Enterprise”.<br>• Show current Gitea MCP Server version. |

# Chapter 8. Troubleshooting

You can contact HCL Support if you are unable to troubleshoot the problem. Gather all the required background information and provide the details to HCL Support for investigation. For more information, see HCL Customer Support.

# Security Considerations

This document describes the actions that you can take to ensure that your installation is secure, customize your security settings, and set up user access controls in HCL DevOps Loop.

You can find the following information about security considerations for the solution:

- Security considerations for Plan
- Security Considerations for Test
- Security Considerations for Deploy
- Security settings for Measure and Release and Considerations for GDPR readiness

# Notices

This document provides information about copyright, trademarks, terms and conditions for the product documentation.



This information was developed for products and services offered in the US.

HCL® may not offer the products, services, or features discussed in this document in other countries. Consult your local HCL® representative for information on the products and services currently available in your area. Any reference to an HCL® product, program, or service is not intended to state or imply that only that HCL® product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any HCL® intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-HCL® product, program, or service.

HCL® may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*HCL*
*330 Potrero Ave.*
*Sunnyvale, CA 94085*
*USA*
*Attention: Office of the General Counsel*

For license inquiries regarding double-byte character set (DBCS) information, contact the HCL® Intellectual Property Department in your country or send inquiries, in writing, to:

*HCL*
*330 Potrero Ave.*
*Sunnyvale, CA 94085*
*USA*
*Attention: Office of the General Counsel*

HCL TECHNOLOGIES LTD. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. HCL® may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-HCL® websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this HCL® product and use of those websites is at your own risk.

HCL® may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*HCL*
*330 Potrero Ave.*
*Sunnyvale, CA 94085*
*USA*
*Attention: Office of the General Counsel*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by HCL® under terms of the HCL® Customer Agreement, HCL® International Program License Agreement or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

Information concerning non-HCL® products was obtained from the suppliers of those products, their published announcements or other publicly available sources. HCL® has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-HCL® products. Questions on the capabilities of non-HCL® products should be addressed to the suppliers of those products.

Statements regarding the future direction or intent of HCL® are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to HCL®, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. HCL®, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. HCL® shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:
© (your company name) (year).
Portions of this code are derived from  HCL Ltd. Sample Programs.

# Trademarks



# Terms and conditions for product documentation

# Index