



Aras DevOps

Release 1.14

Last Modified: 05/28/2026

Copyright Information

Copyright © 2026 Aras Corporation. All Rights Reserved.

Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Phone: 978-691-8900

Notice of Rights

Copyright © 2026 by Aras Corporation and/or its affiliates. All rights reserved.

This document is protected by U.S. and international copyright laws and conventions. No copyright may be obscured or removed from this document. This document may not be modified or altered, or reproduced or transmitted in any form, without the explicit permission of the copyright holder.

Aras Innovator, Aras, and the Aras Corp "A" logo are registered trademarks of Aras Corporation in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

Notice of Liability

This document is provided for informational purposes only, and the contents hereof are subject to change without notice. The information contained in this document is distributed on an "as is" basis, without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement. Aras shall have no liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this document or by the software or hardware products described herein.

Table of contents

Release Notes	7
Platform support matrix	8
Fixed Issues and Known Issues	9
Key Enhancements	14
User Guide	16
Introduction	17
Aras DevOps Overview	18
Customization	20
Standard Development Environment (SDE)	21
Azure DevOps SignOn	23
SDE Navigation	24
Local Development Environment (LDE)	33
Connecting LDE to SDE	34
Obtaining Initial Baseline	36
Create Fork and Clone Repository	40
Review Working Directory	43
Local Environment Variables Set Up	45
Build and Deploy Locally	48
Partial Local Deployment	50
Continuous Integration and Continuous Delivery (CI/CD)	53
Testing	55
Contributor Process	57
The Contribution Process Overview	58
Adding Remote Reference	60
Making Changes in Local Repo	62
Add or Modify file in Code Tree	63



Exporting Packages	65
Make Required Changes in Aras Innovator Instance	66
Export Package After the Changes	67
Copying the Export Utility's Output to the Local Repo	69
Staging Modified Files	70
Continuous Integration Script	71
Test the Deployment Locally	72
Pushing Changes to Fork	74
Fetching Changes/Rebasing	75
Pushing Changes to Fork	76
Creating a Pull Request	77
Trigger, Build, and Test	79
Reviewing a Pull Request	80
Merging the Pull Request	83
Preparing the Project's Initial Baseline	85
Baseline Management	86
Pipelines	88
Deploy to System Integration Testing (SIT) Environment	89
Release Innovator pipeline	92
Use Case one: Standard release delivery (SIT → UAT → STG → PROD)	93
Use Case two: Fast release delivery (UAT → PROD)	99
Generate New Baseline	105
Running the Baseline Pipeline	107
Download Baseline Pipeline	110
Delete Aras Innovator from SIT Environment	116
Delta deployment to production and non-production environments	120
Redeploy UAT deployment on Existing Database	121
Pipeline Retention Policy	127



Using Transformations	128
Type of Transformation	129
The Purpose of Transformation	130
Utilizing Transformation	131
Example 1: XML Document Transformation (XDT)	132
Example 2: JSON Document Transformation (JDT)	133
Ignore Configuration Files Transformation	135
Environment Specific Configuration	136
Example: Define an environment-specific login screen	137
Default Variable Groups	142
Public Variables list	143
Adding a Secret	146
Variables Naming and Scope	147
Restrictions	148
External authentication	149
Aras Innovator External Authentication using SAML 2.0 Authentication	150
Example: Setup of Aras Innovator SAML 2.0 Authentication with Azure as Identity Provider	151
Configuring Secure Files	162
Upload a certificate as a secure file	163
Configure File Permissions for a Pipeline	165
Add a Transformation with Link to Certificate	169
SAML2 Authentication Plugin Configuration	173
Base Configuration	174
Configuring Identity Provider Metadata	176
Configuring Service Provider Metadata	180
Configuring Certificates	184
Additional Authentication Options Configuration	187
Aras Innovator User Setup	189



Generic User Mapper	191
GenericUserMapper Plugin Configuration	192
Packaging	198
Summary of Modeling	199
Review of Packaging Scenarios	201
Case 2	202
Case 3	203
Packaging Tools and Methods	204
Create and Manage New Application	206
Creating a Package Definition	207
Export Package and Update the Imports Manifest File	209
Confirming Manifest Changes in Version Control System	210
Update Repository to Use Single Package	211
Update Repository to New BDS Version	215
Updating from BDS 1.11 to higher Version	216
Updating from older versions to BDS 1.11 or higher	217
Change Management and Implementation	220
Production Countdown Sequence	222
Branding Customization	225
Splash Screen	226
Change Banner	228
Appendix I: Local Development Environment Setup	231
Installing Windows Powershell	232
Installing Chocolatey using Windows PowerShell	233
Installing Git	234
Installing Azure CLI	235
Required Specifications	236
Appendix II: Standard Solution Packaging Tools	237



Export.exe	238
Import.exe	239
Consoleupgrade.exe	240
Appendix III: Adding Applications to a Project	241
Appendix IV: Using a Shared Repository and Merging Conflicts	243
Appendix V: Transformations	247
Appendix VI: Vault Replication	250
Appendix VII: Configure CORS Settings in SDE with Aras.DevOpsFramework.SaaS3 package	252
Appendix VIII: Continuous Integration Pipeline Configuration	254



Release Notes



Platform support matrix

For the software required for the Aras DevOps Release 1.14:

Component	Release
Aras Innovator MSI Deployment	Release 21 Release 23 Release 24 Release 25 Release 26 Release 27 Release 28 Release 29 Release 30 Release 31 Release 32 Release 33 Release 34 Release 35 Release 36 Release 37 Release 38 Release 39
Aras Innovator AICD Deployment	Release 27 Release 28 Release 29 Release 30 Release 31 Release 32 Release 33 Release 34 Release 35 Release 36 Release 37 Release 38 Release 39

This contains release information about the current Build Number:

Release	Build Number
Build and Deployment Services 1.14	1.14.0



Fixed Issues and Known Issues

Fixed Issues in Aras DevOps Release 1.14

Issue #	Description
IH-10961	Aras DevOps has been extended to support deploying Aras Innovator releases, which are running on .NET 10.
IH-10618	Introduced the Release Innovator pipeline that provides a standardized and automated way to deliver updates to Aras Innovator SaaS environments while keeping release operations simple, predictable, and governed.
IH-9215	Baseline updates are now picked up automatically after being uploaded to the SDE feed, allowing CI/CD pipelines to run without additional manual steps or support involvement.
IH-9214	Automated the downloading of baseline artifacts during <code>repo-init.ps1</code> script when they are not found locally by the path specified in <code>Machine.Specific.include</code> file.
IH-8436	SaaS infrastructure has been migrated to Traefik to align with the modern SaaS platform architecture. Deploy pipeline running with update strategy in SaaS environment is no longer fail when
IH-10973	deployments are provisioned with multiple regions. Delta deployments now succeed consistently regardless of how many times they are applied to an existing multi-region deployment.
IH-11616	GenerateBaseline pipeline is no longer failing with an out-of-memory error when processing baselines with a large number of files.
IH-11616	Baseline generation now succeeds consistently, regardless of the number or size of archived files.
IH-11210	<code>New-ArasDevOpsRepository</code> cmdlet is no longer failing due to the deprecated Nuget list command. Repository creation now succeeds consistently without errors.
IH-10956	Continuous integration pipeline in SaaS environments is no longer failing on the Disable High Availability step.



The High Availability disable operation now completes successfully without unnecessary retries or delays.

Known Issues in Aras DevOps Release 1.14

Issue #	Description	Workaround
I-050998	<p>When a user executes The Get-ArasDevopsPackagesList of the Powershell module, only the latest versions are displayed for IOM and InnovatorConfigs. It should display this package's latest version of the current Innovator version.</p>	<p>The commandlet was originally created due to nuget.exe failure to display upstream package versions when listing. Standard Find-Package is capable of properly listing packages and this commandlet is not needed anymore. Get-ArasDevopsPackagesList command is therefore obsolete and will be deprecated.</p>
I-058395	<p>In the Local Development Environment, the BuildAndDeploy.ps1 fails on the install step of Visual Studio 2022 Build Tools when there is a pre-existing installation of Visual Studio Build Tools.</p>	<ol style="list-style-type: none"> 1. Uninstall Visual Studio Installer and all build tools with their dependencies. <pre>> choco uninstall visualstudio2019-workload-webbuildtools</pre> <pre>> choco uninstall visualstudio2019-workload-databuildtools</pre> <pre>> choco uninstall visualstudio2019buildtools</pre> <pre>> choco uninstall visualstudio2022buildtools</pre> <pre>> choco uninstall visualstudio-installer</pre> 2. Reset Powershell session. 3. Run the script again and let the tool install everything it needs: <pre>Import-Module Aras.DevOps</pre> <pre>Install-BuildMachinePrerequisites -InnovatorVersion <version></pre> 4. In case the problem is not resolved by following the steps above, or the choco packages are not installed in the first



place, download visual studio installer (if not already installed) and perform the following steps:

- Open Visual Studio Installer, for both 2022 and 2019 versions, export the configuration separately to dedicated files. The export option is under the More button. These files will be modified and re-imported.
- Open a Powershell terminal and run> explorer (join-Path (Get-Module Aras.DevOps).Path "\.resources")
- For reference, open vsbuildtools2019.vsconfig file in your text editor. DO NOT Modify the file. Edit the 2019 configuration exported in step 4 a, to remove all components listed in the vsconfig file.
- Import 2019 configuration back into the Visual Studio Installer, by clicking on "More" and selecting "Import configuration."
- Perform steps 3 & 4 for the 2022 version of visual studio, using



		<p>vsbuildtools2022.vsconfig file.</p> <ul style="list-style-type: none"> Note: If only one of the two versions above exists, perform the steps only for that version.
I-058883	<p>In the Local Development Environment, ContinuousIntegration.ps1 fails if using Aras Innovator Container Deployment in Aras Enterprise and displays the following error: <The sp_configure value "contained database authentication" must be set to 1 in order to restore a contained database. You may need to "RECONFIGURE" to set the value_in_use. RESTORE DATABASE is terminating abnormally></p>	<p>In the SQL Server on the local machine, enable the contained database authentication option. The option can be enabled by two ways:</p> <ol style="list-style-type: none"> Follow steps in: https://learn.microsoft.com/en-us/sql/database-engine/configure-windows/contained-database-authentication-server-configuration-option?view=sql-server-ver16 Configure the option using SQL Server Management Studio properties
I-059096	<p>When deploying on a local machine, users may see an error message directly from SQL Server that the password validation failed. The error message may look like:"System.Data.SqlClient.SqlException: Password validation failed. The password does not meet the operating system policy requirements because it's too short"</p>	<p>Deployment of Aras Innovator on a user's machine relies on local SQL Server that defines policies for SQL user passwords. Aras uses 2 SQL users named "innovator" and "innovator_regular". Passwords used during deployment are defined in Machine.Settings.include Make sure that both users can log into SQL Server using defined password before launching new deployment.</p>
I-059899	<p>When an existing project moves to a container-based deployment, it is possible that the initial repository contains</p>	<p>A customer repository should not contain any configuration files.</p>



	some modified configuration files. In this case, the deployment will not be successful.	The modifications to configuration files that might be required in the context of a project needs be described using Config Transformations feature available with the service.
I-072044	<p>Chocolatey Version Update</p> <p>In Aras DevOps 1.10, an updated version of Chocolatey is now used to meet enhanced security requirements. Since two versions of Chocolatey cannot run simultaneously on the same machine, users must manually downgrade Chocolatey if they need to work on projects that reference an older version of Aras DevOps.</p>	Manually downgrade the version of Chocolatey.
I-073506	The Generate Baseline pipeline may intermittently fail when used with unpatched Aras Innovator versions lower than Release 32.	Retry the Generate Baseline pipeline execution.
IH-10423	<p>After upgrading to Aras DevOps 1.13, the Selenium Tests pipeline may fail (pipeline status turns yellow) when used on Software Development Environments (SDEs) running Aras Innovator 14.0.6 or earlier.</p> <p>This occurs because updated test logic in BDS 1.13 is no longer compatible with end-of-life Aras Innovator versions. Upgrade Aras Innovator to a supported version to restore compatibility.</p>	



Key Enhancements

Manage SaaS deployments using the Release Innovator pipeline

The new Release Innovator pipeline provides a standardized and automated way to deliver fixes and updates to Aras Innovator SaaS environments while keeping release operations simple, predictable, and governed.

New Behavior:

- A single Release Innovator pipeline is now used to manage deployments across SIT, UAT, STG and Prod environments with embedded approvals.
- Stage names and execution order are standardized, making the release process easier to understand, operate and audit.
- Deploy pipelines are split by environments.
- Deploy pipelines logic is moved out of the repository

Benefit:

- Simplifies SaaS release operations by replacing multiple manual steps with a single, clearly structured workflow.
- Enables development teams to manage releases more confidently while maintaining governance and control over Production deployments.

Automated Baseline Artifact Acquisition During Repository Initialization

The `repo-init.ps1` script now includes logic to verify the presence of necessary baseline in the configured local path. If this artifact is missing, the script automatically downloads it from the project's universal feed.

Automated Baseline Artifact Handling

Baseline handling has been simplified and automated. Baseline updates are now picked up automatically after being uploaded to the SDE feed, allowing pipelines to run without additional manual steps or support involvement.

Migration to Traefik Ingress



SaaS infrastructure has been migrated to Traefik to align with the modern SaaS platform architecture.

This change standardizes ingress configuration, improves maintainability, and provides a more scalable foundation for platform services. Users will benefit from continued security compliance and infrastructure stability without requiring any manual reconfiguration of current routing rules.

Support for Aras Innovator running on .NET 10

Aras DevOps has been extended to support deploying Aras Innovator 39, which runs on .NET 10. This ensures Aras Innovator running on .NET 10 can be deployed and managed in SaaS and Aras DevOps environments.



User Guide



Introduction

Purpose

This user guide provides detailed information for contributors utilizing the Aras DevOps service to manage and customize their locally deployed Aras Innovator and application instances.

Scope

The scope of this user guide provides instructions to define, manage, customize, and validate locally deployed customizations, as well as outline the following:

- Contributor Process
- Branding
- Baselines
- Pipelines
- Transformations
- Packaging
- Change Management and Implementation
- Appendices which provide tool information and instructions

This user guide provides good practices to ensure proper configuration management of a solution for business-critical operations.

Contributors' experience with these processes and procedures will determine what tools they already use and prefer. The tools mentioned in this user guide are suggestions for consistency and practical embodiment of the concepts, not endorsements or mandates.

Target audience

This document is intended for contributors responsible for following its instructions (customizers, PLM developers, and stakeholders involved in software development and project management).

Contributors working on implementing solutions using the Aras Innovator platform are responsible for adhering to the information and steps outlined in this user guide.



Aras DevOps Overview

Overview

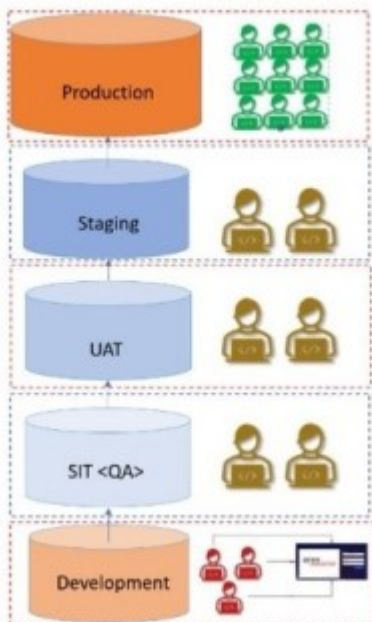
Aras DevOps is a subscription service that provides a cloud-based set of tools, scripts, and processes to manage Aras Innovator customizations for an Aras Innovator implementation project.

Aras DevOps is inherently part of the Aras Enterprise subscription (SaaS) but can also be purchased separately as an Aras DevOps subscription to support customer-hosted Aras Innovator environments.

Centralized Development Model vs. Distributed Development Model (Aras DevOps)

In the Centralized Development Model, developers collaborate and implement changes within shared installations of Aras Innovator, such as development, testing, and production instances. Developers export and import Packages between the deployments and keep track of changes.

Centralized Development Model

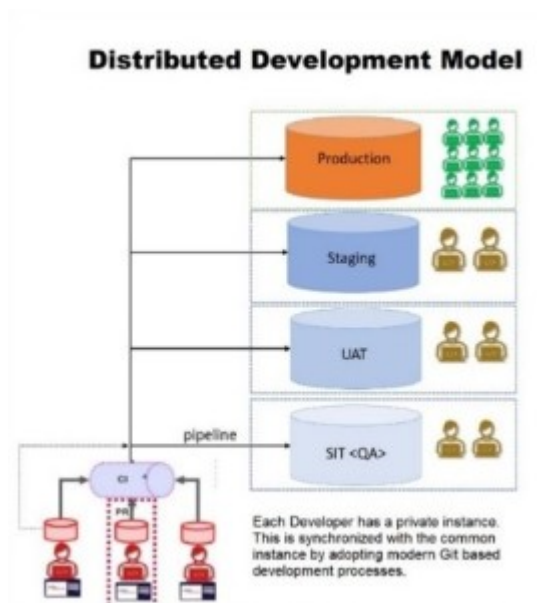


Aras DevOps enables developers to build and deploy an individual instance of Aras Innovator with a Distributed Development Model. The system is modified using Git (a Version Control System), which

allows changes to be exported and stored.

Modifications are extracted from Git to build the system. Aras DevOps is equipped to execute automated tests written by the development team and specifically designed for the Aras Test Automation Framework (TAF) to verify the system's validity.

Multiple developers can contribute to the system, and Aras DevOps enables developers to identify conflicts and merge all their contributions into a single Build. This enables early detection and resolution of conflicts, improving collaboration and development efficiency.



Important

The link to the production environment is not included for On-Premises customers who purchased Aras DevOps as a stand-alone service. This link is included for Aras Enterprise subscription customers.

Customization

System customization consists of configurations per customer usability. The following outlines the general contributor configurations/customizations:

- Workflow
- Reporting
- Interfacing
- Configuring lifecycles and other preferences
- Enhancing
- Forms

Contributors must set up a Local Development Environment (LDE) for local project configuration and access to the Standard Development Environment (SDE) for contributions.



Standard Development Environment (SDE)

SDE Overview

Aras DevOps includes the Standard Development Environment (SDE), enabling contributors to streamline their customized Aras solution in a cloud environment. It consists of the SIT and Build environment, as outlined in this section.

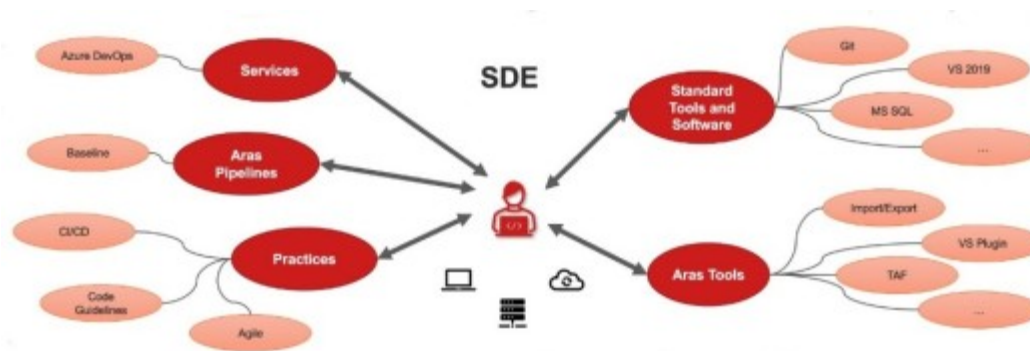
SDE consists of tools and procedures that support contributors in implementing standard Continuous Integration/Continuous Deployment (CI/CD) practices. This environment is designed to streamline software development by facilitating efficient collaboration, version control, code testing, and seamless deployment.

The following tools are provided by Aras:

- **Aras Visual Studio Plugin:** Allows seamless integration between Aras and Visual Studio.
- **Import/Export:** Tools that facilitate the easy movement of data in and out of the system.
- **Test Automation Framework (TAF):** Helps validate the system's functionality.

To supplement these resources, the SDE uses Azure DevOps services, which provide a development environment where contributors can commit their changes, build the applications, and test the Aras customizations. This ensures a standardized, repeatable process, reducing the risk of deployment errors.

The SDE incorporates Aras **Pipelines**, which are workflows that automate steps in the software delivery process, such as build, test, and deployment. By integrating these different components, the SDE provides a comprehensive suite of tools for managing the entire software development lifecycle.



Once access is granted by Aras, a link to the SDE implementation project is emailed to the requester (e.g., <https://dev.azure.com/{organization}/{project}>). This link navigates to the dedicated space within Aras DevOps. All developers should have access to this link.

The Azure DevOps environment is only available to customers who have acquired either the Aras DevOps Subscription or the Aras Enterprise Subscription.

The Local Development Environment configuration includes developer machine requirements, which include creating a Fork, **Baseline** setup, Aras Repository Clone, and environment setup.

Refer to the appendices at the end of this user guide for instructions related to the tools Aras suggests.



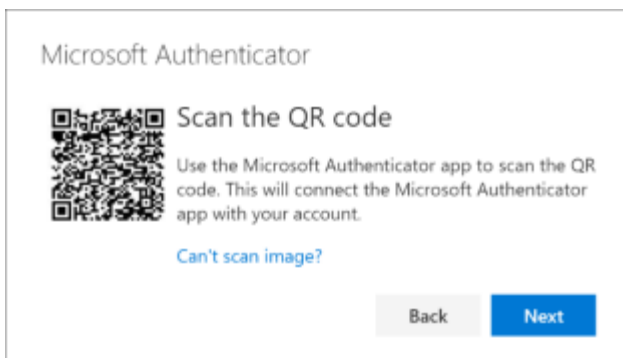
Azure DevOps SignOn

The following steps outline the process of signing into Azure DevOps:

1. Click the `https://dev.azure.com/{organization}/{project}` link.
2. Enter the registered email address used for access.
3. Install the **Microsoft Authenticator** application on a mobile device.
4. Click **Next** on the **Microsoft Authenticator** dialog box.



5. Open the **Microsoft Authenticator** application on the mobile device and click the + icon.
6. Select the account and scan the QR code on the computer.
7. Click **Next**.

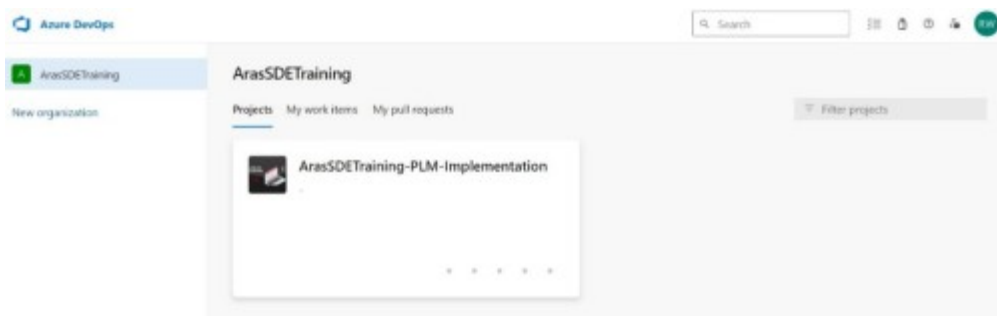


SDE Navigation

The URL to the SDE environment is accessible only after the invitation process, which requires setting up multi-factor authentication, is completed.

When connecting to the SDE URL, the following screens appear:

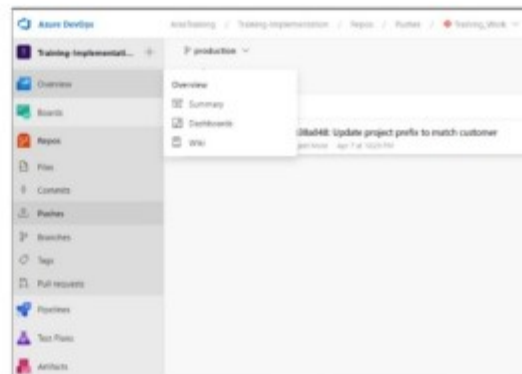
- **Azure DevOps landing page:** The visible implementation project represents where the project team customizes Aras Innovator for subscribers.



- **Azure DevOps Structure:** Displays required information.

Azure DevOps Structure

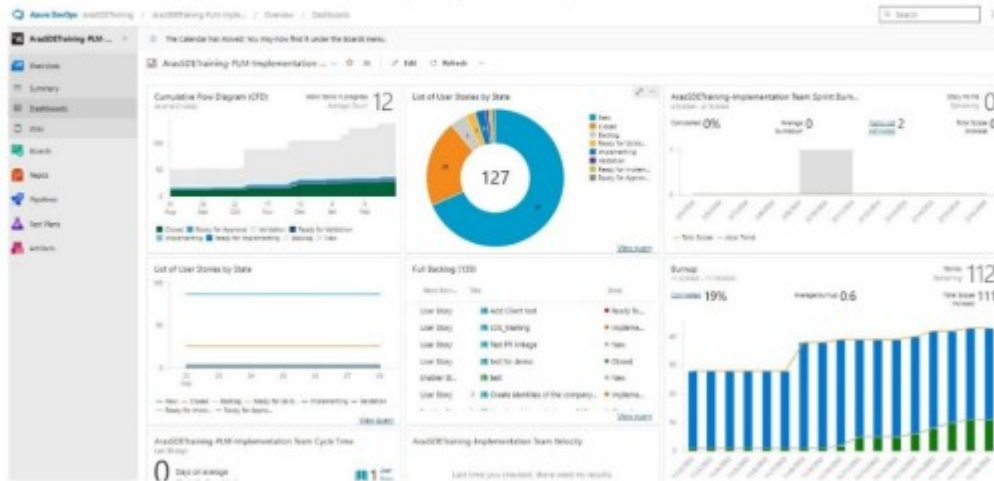
- Overview – dashboards (scrum Masters)
- Boards – backlog, queries, etc.
- Repos – team vs. Individual
- Pipelines – ContinuousIntegration, DeployToSIT, etc.
- Test Plans – Test Plans, Runs
- Artifacts – Baselines, Feeds



- **Dashboards:** Scrum masters use these to track project progress for team members' Views. A link is provided for traceability between change management and implementation.

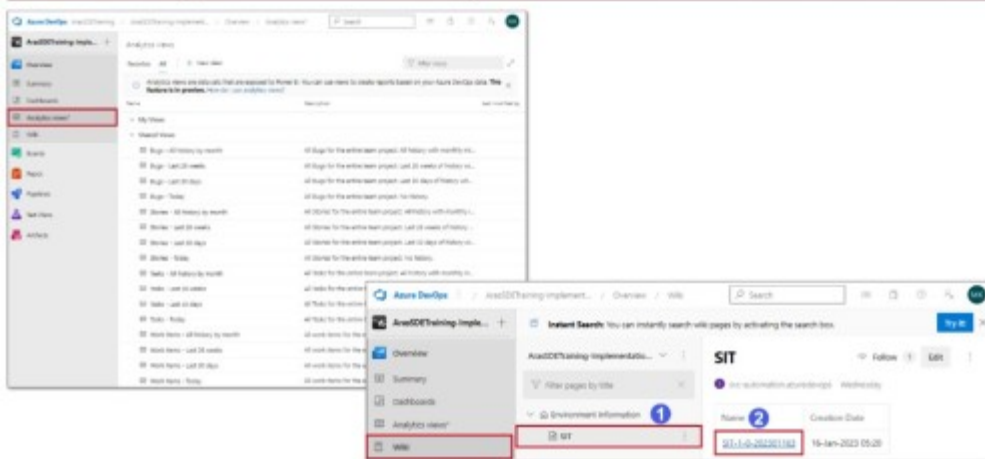
Azure DevOps Overview – Dashboards

Customizable, highly configurable dashboards provide you and your teams with the flexibility to share information, monitor progress and trends, and improve your workflow processes.



- Analytics Views and Wiki Page:** The team project **Wiki** is a platform for sharing information with other team members. Provisioning a **Wiki** from scratch initiates a new Git Repository that stores Markdown files, images, attachments, and a sequence of pages. The **Wiki** can support collaborative editing of both its content and structure. Aras uses **Wiki** to share the URLs of SIT test instances.

Azure DevOps Overview – Analytics views & Wiki



- Work Items:** These Items are required for approved changes. This may encompass other tasks necessitated by the project or the tracking of requests made to Aras.



Azure DevOps Boards – Work Items

- Work item types (WITs) represent the core of the Azure DevOps tracking system and can be a bug, a requirement, a general to-do, etc.
- Each work item has a unique ID to keep track of its references from its creation to its implementation as a piece of executable software.

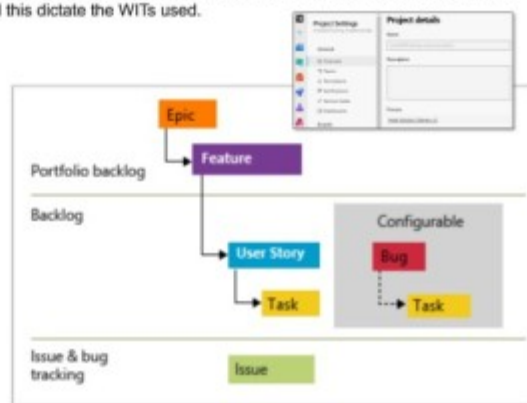
ID	Title	Assigned To	State	Area Path	Tag	Comments	Activity Date
360	Add Design Request P01	Unassigned	New	ArasDevOps\PSM\implem...			2/16/2024 2:58:00 PM
342	Design Request Management P05	Unassigned	Formal	ArasDevOps\PSM\implem...			2/16/2024 2:40:00 PM
360	Change Management P05	Unassigned	New	ArasDevOps\PSM\implem...			2/16/2024 2:48:47 PM
202	Add Design Request C03	Unassigned	New	ArasDevOps\PSM\implem...			2/16/2024 1:31:34 PM
307	Add Design Request C04	Done-Marko	New	ArasDevOps\PSM\implem...			2/16/2024 1:31:19 PM
354	Add Design Request C08	Unassigned	New	ArasDevOps\PSM\implem...			2/16/2024 1:30:45 PM
340	Add Design Request C10	Unassigned	New	ArasDevOps\PSM\implem...			2/16/2024 1:27:39 PM
309	Design Request Management C10	Unassigned	Formal	ArasDevOps\PSM\implem...			2/16/2024 2:40:19 PM
322	Change management C04	Done-Marko	New	ArasDevOps\PSM\implem...			2/16/2024 2:40:57 PM
328	Design Request management C06	Done-Marko	Formal	ArasDevOps\PSM\implem...			2/16/2024 2:40:46 PM
310	Design Request Management C12	Philip Ischell	Formal	ArasDevOps\PSM\implem...			2/16/2024 2:42:34 PM
358	Change Management C10	Unassigned	New	ArasDevOps\PSM\implem...			2/16/2024 2:40:28 PM
358	Install package C10	Justin Desrosier	New	ArasDevOps\PSM\implem...			2/16/2024 2:39:44 PM
329	Change management C10	Justin Desrosier	New	ArasDevOps\PSM\implem...			2/16/2024 2:39:27 PM
358	Add Design Request C16	Tom DeWent	New	ArasDevOps\PSM\implem...			2/16/2024 2:37:39 PM
358	Add Design Request C07	Unassigned	New	ArasDevOps\PSM\implem...			2/16/2024 2:37:46 PM
307	Design Request Management C16	Tom DeWent	Formal	ArasDevOps\PSM\implem...			2/16/2024 2:37:11 PM
328	Design Request Management C07	Unassigned	Formal	ArasDevOps\PSM\implem...			2/16/2024 2:37:02 PM

- **Work Item Structure:** Azure DevOps manages the process according to the definition set by Aras Global Cloud Services (GCS). GCS continually updates the process to address subscriber input and feedback. The current Agile Solution Delivery version is 4.1.

Azure DevOps Boards – Agile Process Model

- Supports Agile planning methods, including Scrum, and tracks development and test activities separately. This process works great if you want to track user stories and (optionally) bugs on the Kanban board, or track bugs and tasks on the taskboard. The process type is based on **Agile Solution Delivery v3**, and this dictate the WITs used.

- Hierarchy
 - Epics
 - Features
 - Stories: User Story and Enabler Story, with Acceptance Criteria
 - Areas: Functional, capability attached/assigned to teams
 - Sprints (Iterations): Used for refinement (grooming)
- Set definitions of Ready and Done
- Epics and features reflect the business focus, user stories and tasks are related to the development
- Example:
 - Area: Certification
 - Epic: Change Management
 - Feature: Design Request Management
 - User Story: Add Design Request



- **Boards:** The **Boards** can track various **Work Items**, including features, user stories, tasks, bugs, etc. They support both Scrum and Kanban methodologies. Additionally, the **Boards**



include capabilities for planning **Sprint**, managing **Backlogs**, and generating reports on work progress.

Azure DevOps Boards – Agile Process Model

• Supports Agile planning methods, including Scrum, and tracks development and test activities separately. This process works great if you want to track user stories and (optionally) bugs on the Kanban board, or track bugs and tasks on the taskboard. The process type is based on **Agile Solution Delivery v3**, and this dictate the WITs used.

• **Hierarchy**

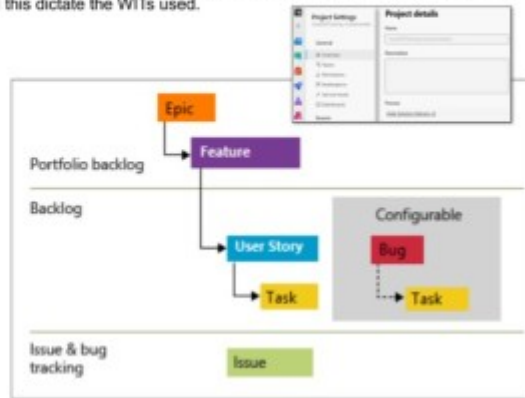
- Epics
- Features
- Stories: User Story and Enabler Story, with Acceptance Criteria
- Areas: Functional, capability attached/assigned to teams
- Sprints (Iterations): Used for refinement /grooming

• **Set definitions of Ready and Done**

• Epics and features reflect the business focus, user stories and tasks are related to the development

• **Example:**

- Area: Certification
- Epic: Change Management
- Feature: Design Request Management
- User Story: Add Design Request



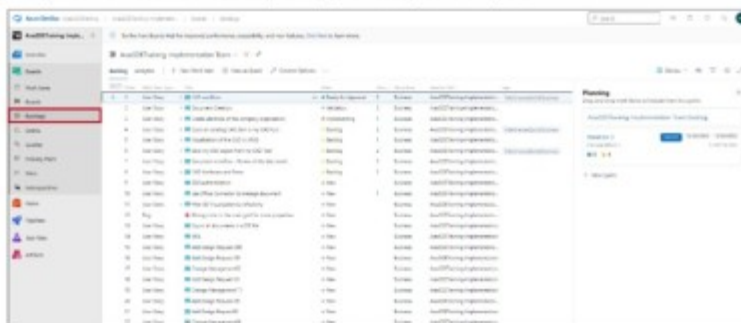
- **Backlogs:** Backlogs in Azure DevOps are used to manage and prioritize **Work Items** in a queue. They provide an ordered list of **Work Items**, such as user stories, features, or bugs that the team needs to work on.

Azure DevOps Boards – Backlogs

• Backlogs display work items as a list and boards display them as cards. With list backlogs, you can quickly develop your project plan, group and prioritize work, and make bulk updates on selected work items.

• There are three classes of backlogs

- **Portfolio backlogs** typically track high-level features, scenarios, or epics; they help you organize your product backlog into a hierarchy of elements.
- The **product backlog** contains a prioritized list of user stories, deliverables, or works you plan to build or fix.
- **Sprint backlogs** contain just those items that each team is working on during a scheduled sprint or iteration period.

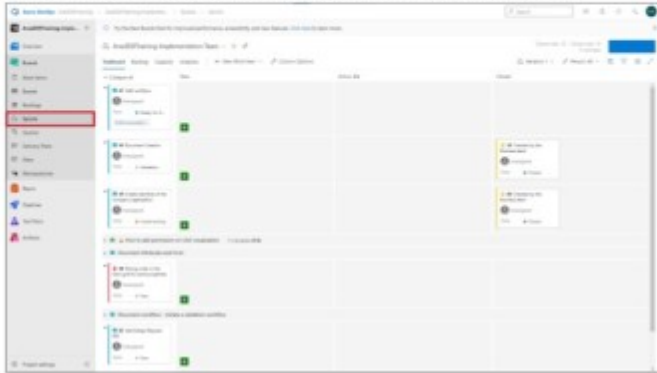


- **Sprints:** Sprints in Azure DevOps represent time-boxed iterations where a set amount of work is completed.



Azure DevOps Boards – Sprints

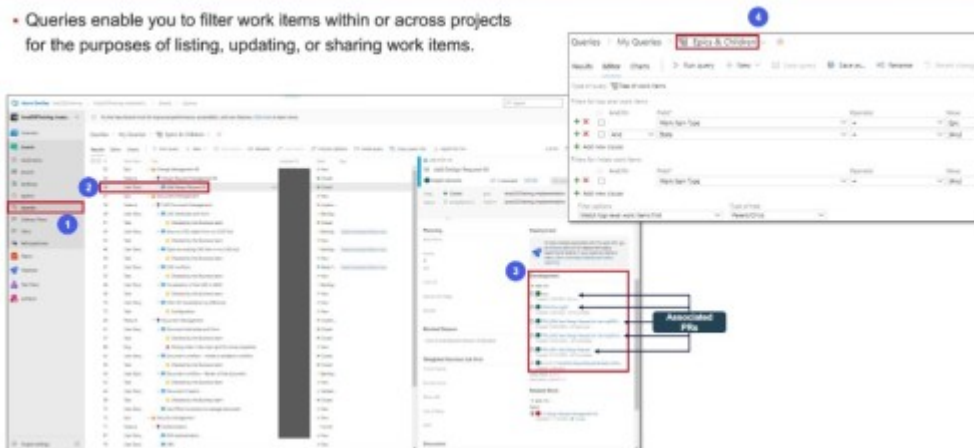
- Sprints, specified by iteration Paths, are defined for a project and then selected by teams. A sprint cadence can vary between one week to four weeks or longer.
 - You assign work to sprints that teams commit to deliver at the end of the sprint.
 - Azure Boards tools rely on sprint assignments to a team Sprint backlogs, Taskboard, and Delivery plans.



- **Queries:** Queries enable users to filter **Work Items** within or across projects for listing, updating, or sharing **Work Items**.

Azure DevOps Boards – Queries

- Queries enable you to filter work items within or across projects for the purposes of listing, updating, or sharing work items.



- **Repos:** There is only one main configuration **Repo** per project for the Standard Development Environment. Notice the orange and white backgrounds of the Git logo. The white background repositories are Forks.



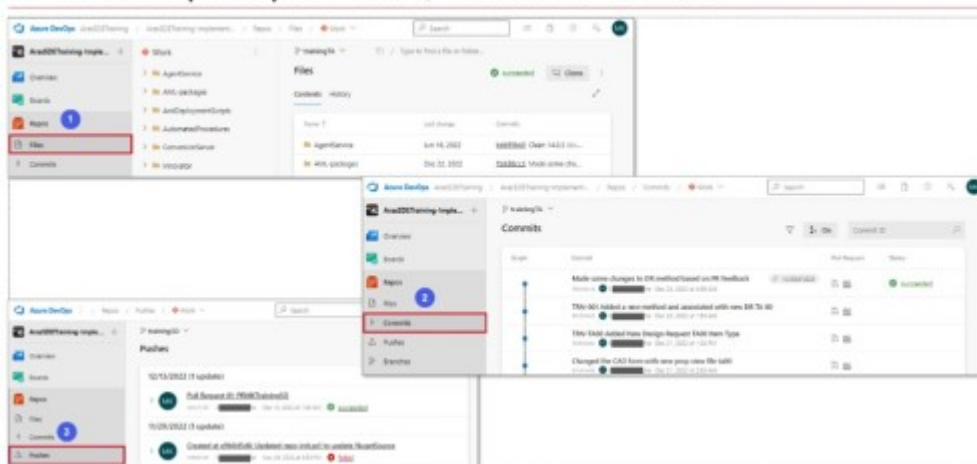
Azure DevOps Repos – Repos

- Azure Repos is a set of version control tools that you can use to manage your code.
- Before we start the development or implementation, based on Innovator version required, an Azure DevOps repository (CRT) will be created by GCS and access will be given to customer project team
- This Azure DevOps Repo will be based on CRT, and all customizations and configurations should be committed to this Azure DevOps repository.
- Building custom packages and deployments will happen from this repository.



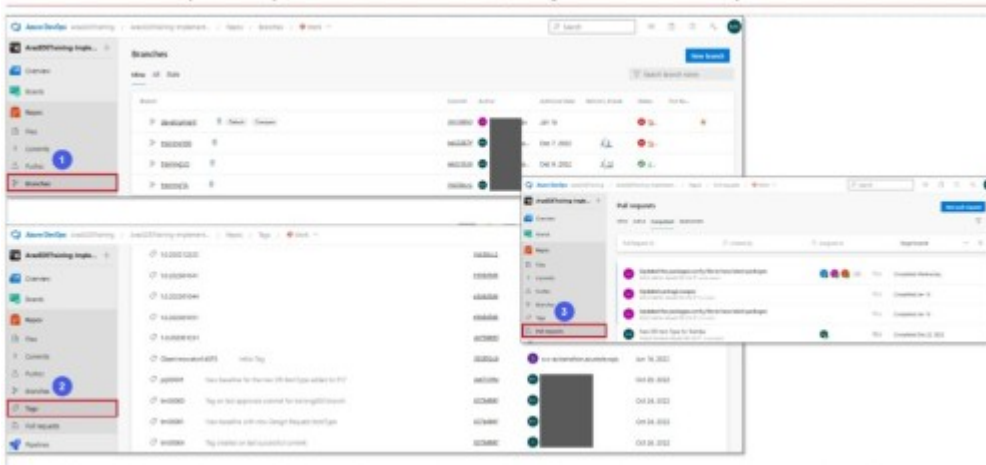
- **Files, Commits, and Pushes:** In Azure DevOps, **Files** are the individual project components, **Commits** are snapshots of changes made to those files, and **Pushes** are the action of uploading these **Commits** to a Remote Repository for team access and collaboration.

Azure DevOps Repos – Files, Commits & Pushes



- **Branches, Tags, and Pull Request:** In Azure DevOps, **Branches** are separate versions of the codebase for isolated development, **Tags** are reference points to specific versions of the code, and a **Pull request** is a mechanism for developers to propose, review, and merge changes from one **Branch** to another.

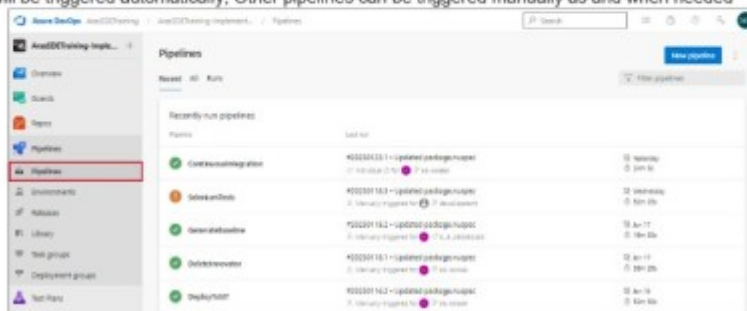
Azure DevOps Repos – Branches, Tags & Pull Requests



- **Pipelines:** Pipelines in Azure DevOps are automated workflows for Continuous Integration and Delivery, enabling code build, test, and deployment processes.

Azure DevOps Pipelines – Pipelines

- Azure Pipelines automatically builds and tests code projects to make them available to others.
- Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to test and build your code and ship it to any target (SIT, and other environments).
- These pipelines are configured for your repository based on the subscription model
- CI Pipeline will be triggered automatically; Other pipelines can be triggered manually as and when needed



- **Test Plans:** Test Plans in Azure DevOps provide a structured approach for defining, tracking, and managing testing activities to ensure software quality.



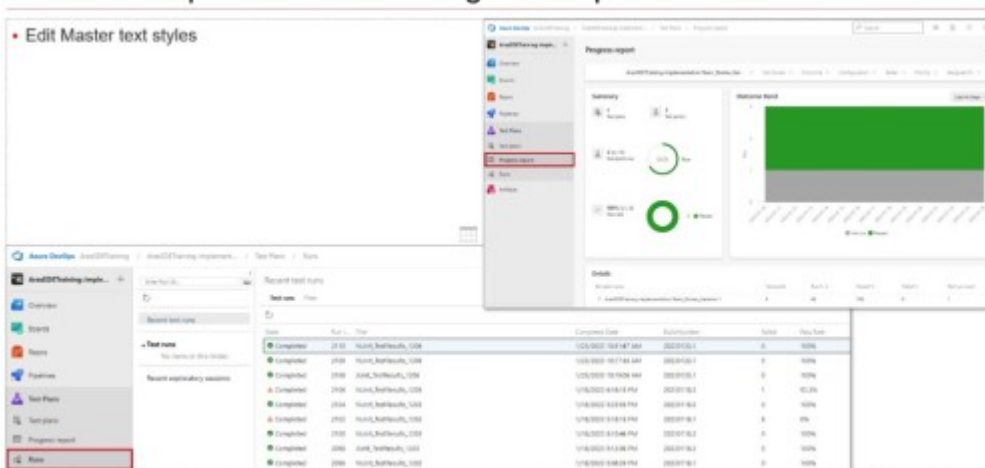
Azure DevOps Test Plans – Test Plans

- Azure Test Plans is an easy-to-use, browser-based test management solution that provides all the capabilities required for planned manual testing, user acceptance testing, exploratory testing, and gathering feedback from stakeholders.
- This is an addition to the standard SaaS environment (additional cost).



- **Progress Reports and Runs:** Progress Reports in Azure DevOps provide insights into the development lifecycle and project milestones, while Runs represent individual executions of tests, builds, or deployments.

Azure DevOps Test Plans – Progress Report & Runs

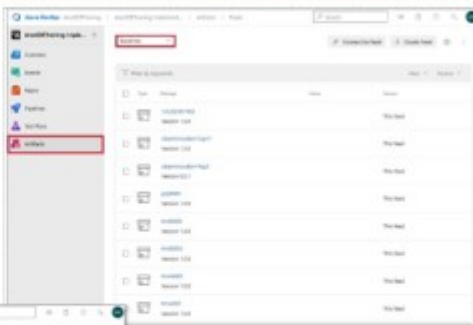


- **Baselines and Artifacts:** Baselines in Azure DevOps represent specific versions of the project for comparison or recovery, while Artifacts are the output files generated from build and release Pipelines.



Azure DevOps Artifacts – Baselines & Artifacts

- With Azure Artifacts, you can create and share NuGet and npm packages from private and public sources with teams in Azure DevOps. These packages can be used in source code and can be made available to the CI/CD pipelines. With Azure Artifacts, you can create multiple feeds that you can use to organize and control access to the packages.
- Feeds
 - **Baselines**
 - Provides the GCS initial baselines that can be used by customers
 - Home for future baselines created during the customer development cycles
 - **Artifacts (non-SaaS customers)**
 - Provides diverse CRT and System related packages



Local Development Environment (LDE)

A Local Development Environment (LDE) is a software and tools configuration on a developer's computer that enables developers to write, debug, and test software. This environment often mimics the production setting as closely as possible, encompassing programming languages, code editors, Version Control Systems, and possibly virtual machines or containers. The LDE allows developers to make and test changes without affecting the live application or production data.

Depending on company policies, an IT department may need to perform the below steps for users. For more details, refer to the Setting Up Local Development Environment Appendix.

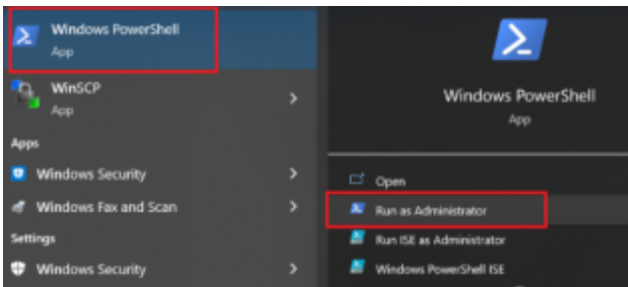


Connecting LDE to SDE

When working in the LDE, open Windows PowerShell as an administrator to connect to SDE.

The following steps outline the process for connecting to an SDE:

1. Run Windows PowerShell as an administrator.



2. Run the command: **az login**

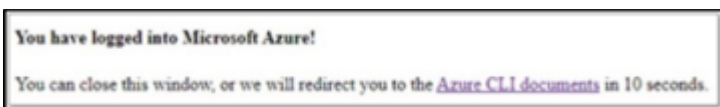
If tenant-level access is missing, run the command:

az login --allow-no-subscriptions

The Microsoft Azure dialogue box appears.



3. Select the Microsoft Azure account to launch Azure. The following message should appear:



Using **az login** might not consistently grant access to the specific SDE. If the **az login** does not work, using the **Personal Access Token (PAT)** method is recommended.

The current **Baseline** can be obtained and stored with the LDE successfully linked to the SDE.



Obtaining Initial Baseline

Important

Starting with Aras DevOps 1.14 version, the baseline is downloaded automatically from the corresponding feed to the Local Development Environment when you run the BuildAndDeploy.ps1 command.

Follow the steps in this chapter only if the automatic download does not work and you need to download the baseline manually.

Aras uses a **Baseline** as a database and Code Tree backup. When the project starts, Aras initializes the SDE with the current release of Aras Innovator and provides the corresponding **Baseline** in the **Baseline** feeds in the **Artifacts**.

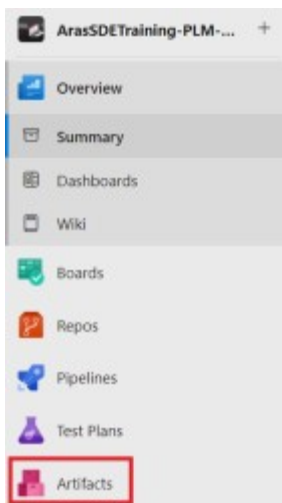
Projects periodically generate new **Baselines**, which may be created at the start of a project to add applications and language packs.

The following steps outline the process to obtain the initial **Baseline**:

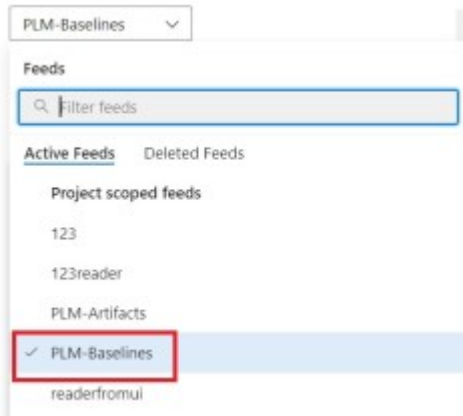
1. Create a folder to save the **Baseline** in the local machine. For example,

```
C:\ArasProjects\Baselines\Cleaninnovatorxxspyy.
```

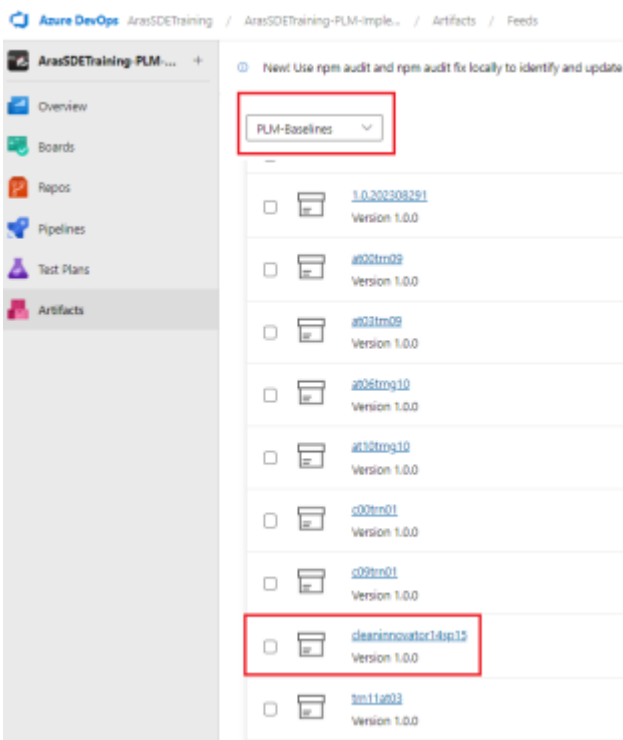
2. From Azure DevOps, select **Artifacts**.



3. Select **Baselines** from the drop-down menu.



4. Select the available **Baseline(s)** for the Aras Innovator release.



Aras provides the **CleanInnovatorxxSPyy Baseline** for every new SDE. When selecting the **Baseline(s)**, the command line for downloading appears.



Overview Versions Upstream Versions

Get this package with the Azure DevOps Extension for Azure CLI

```
az artifacts universal download \
--organization "https://dev.azure.com/ArasSDETraini
ng/" \
--project "8f93ba30-0191-455b-8230-b1ab4fa6d662" \
--scope project \
--feed "PLM-Baselines" \
--name "CleanInnovator12sp17" \
--version "1.0.0" \
--path .
```

Connect to Feed then

Provenance

Publisher
Clifford Yu using VSservices/16.105.31303.2 (NetStandard; Microsoft Windows 10.0.19042) ArtifactTool/02.190.15124052 © Jul 17, 2021 (7/17/2021 12:54:18 AM)

Development

Source
This feed

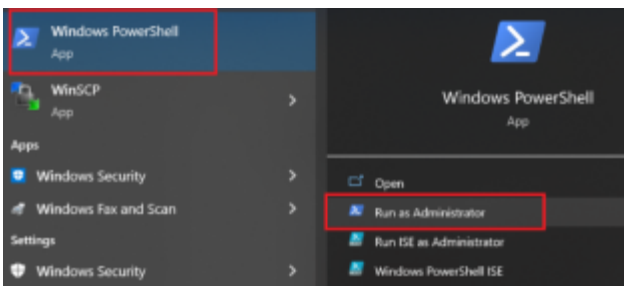
Stats

- 106 downloads of this version
- 106 total downloads
- 57 users of this version
- 57 total users

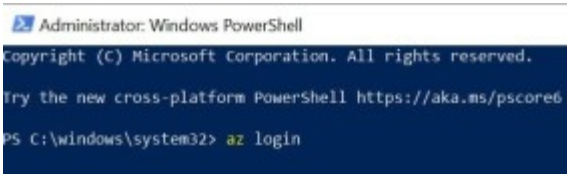
Description

Out of the box Innovator version 12SP17

5. Run Windows PowerShell as an administrator.



6. Run command: **az login**



7. Run the command to access the newly created folder in step 1:

```
cd C:\ArasProjects\Baselines\Cleaninnovator14sp3
```

8. Copy the command from Azure DevOps to download the **Baseline** and run the command in Windows PowerShell.



Overview Versions Upstream Versions

Get this package with the Azure DevOps Extension for Azure CLI

```
az artifacts universal download \
--organization "https://dev.azure.com/ArasSDETraining/" \
--project "8f93ba30-0191-455b-8230-b1ab4fa6d662" \
--scope project \
--feed "PLM-Baselines" \
--name "cleaninnovator14sp3" \
--version "1.0.0" \
--path .
```

Connect to Feed then

Provenance

Publisher
Clifford Yu using VSServices/16.185.31303.2 (NetStandard)
Microsoft Windows 10.0.19042
ArtifactTool/0.2.190.15124052
© Jul 17, 2021 (7/17/2021 12:54:18 AM)

Development

Source
This feed

Administrator: Windows PowerShell

```
Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\windows\system32> az devops login --organization https://dev.azure.com/ArasSDETraining/
Token:
PS C:\windows\system32> cd C:\ArasProjects\Baselines\Cleaninnovator14sp3
PS C:\ArasProjects\Baselines\Cleaninnovator14sp3> az artifacts universal download --organization "https://dev.azure.com/ArasSDETraining/" --project "8f93ba30-0191-455b-8230-b1ab4fa6d662" --scope project --feed "PLM-Baselines" --name "Cleaninnovator14sp3" --version "1.0.0" --path .
Downloading Universal Packages tooling (ArtifactTool_win10-x64_0.2.267): 100.00% ..
```

The following files are visible in the newly created folder when the download is complete.

This PC > Windows (C:) > ArasProjects > Baselines > Cleaninnovator14sp3

Name	Type	Size
CodeTree	Compressed (zipped)...	393,665 KB
DB.bacpac	BACPAC File	4,172 KB
DB.bak	BAK File	14,926 KB



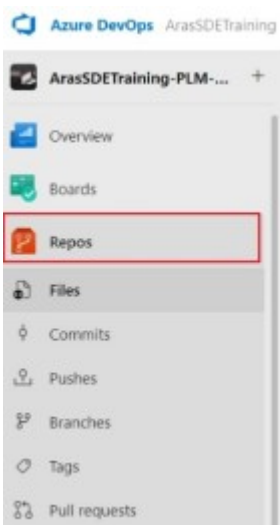
Create Fork and Clone Repository

Creating Forks

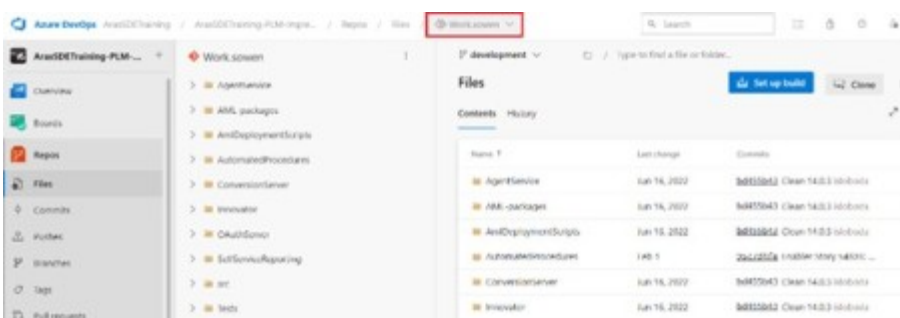
A Fork refers to creating a copy of a Repository within the same organization or project. Forking a Repository generates a new copy of the original Repository, including all its code, **Branches**, and **Commit History**.

The following steps outline the process to create Forks:

1. From Azure DevOps, select **Repos**.

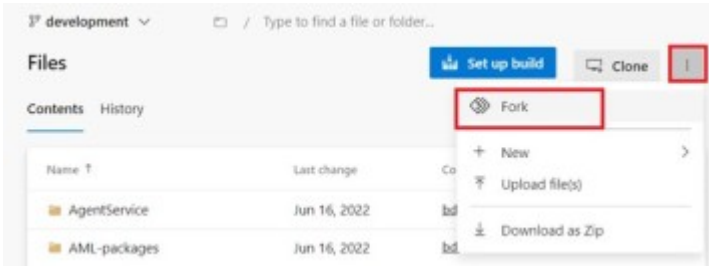


2. Locate and click the **Repository to Fork**.



3. Click **More actions** and select **Fork**.





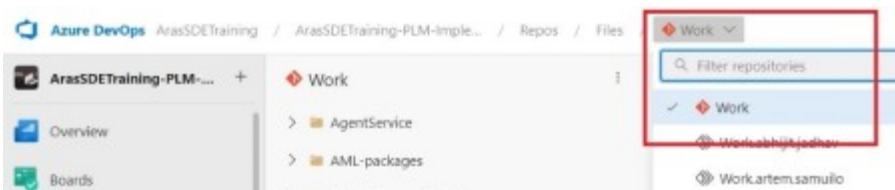
4. Select **All branches** and click **Fork**. The **Repository** name and **Project** is auto populated. Change the **Repository** name if needed. Azure DevOps creates the Forked Repository and redirects the user to its page once the process is complete. Users clone the Forked Repository to the local machine for making changes and push them back to the Forked Repository.

Cloning Repo to Local Working Directory

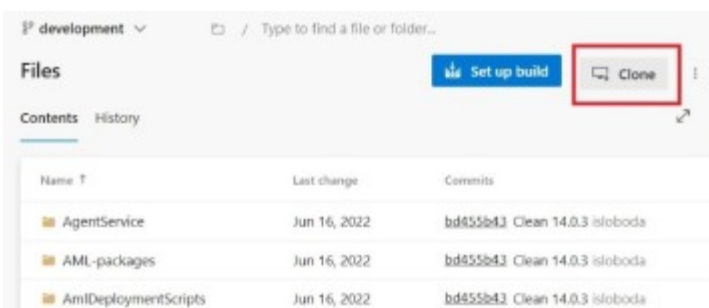
When cloning the Repository, a complete copy of the project's codebase, **Commit** History, and related files is created on a local machine.

The following steps outline the process of cloning a Repository to a local directory:

1. Select the Fork to be cloned in Azure DevOps from the **Work** drop-down menu.

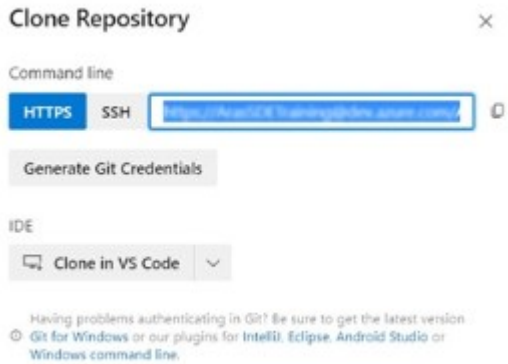


2. Click the **Clone** button.



The **Clone Repository** dialog box appears with the Repository's clone URL.

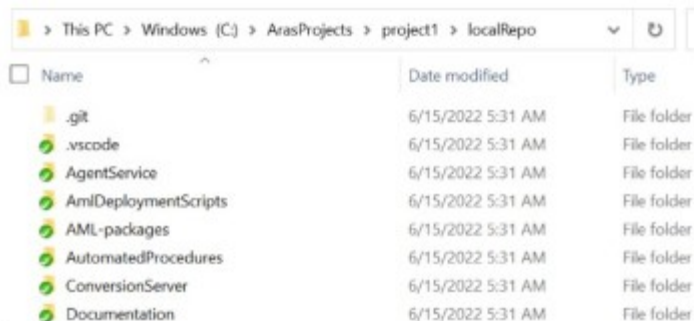




3. Copy the clone URL (HTTPS or SSH). An example URL is `https://dev.azure.com/{organization}/{project}/git/{repository}`. Use any Version Control Tools required to clone the Repository.
4. In the Version Control Tool's interface, find the option to clone or create a new Repository.
5. Paste the clone URL copied from the Azure DevOps Project.
6. Browse to the destination directory to clone the Repository.

Optional: Depending on the tool used, additional configuration options are available during the cloning process. These could include selecting **Branches**, specifying authentication credentials, or choosing the desired clone depth.

7. Click the **Clone** button within the Version Control Tool.
8. When the cloning is completed, confirm the contents of the localRepo folder against the contents of the Fork in Azure DevOps.



Review Working Directory

To continue with this section, you must ensure that the workstation (either a laptop or a virtual machine) has been configured to operate within the LDE. For more details, see the Setting up Local Development Environment Appendix.

The following steps outline the process to set up the working directory:

1. Open a Windows PowerShell session as administrator and navigate to the Repository. For example, `C:\ArasProjects\project1\localRepo`
2. To check the directory, execute the command: **dir**

```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\windows\system32> cd C:\ArasProjects\Project1\LocalRepo\MyFork
PS C:\ArasProjects\Project1\LocalRepo\MyFork> dir

Directory: C:\ArasProjects\Project1\LocalRepo\MyFork

Mode                LastWriteTime         Length Name
----                -
d-----          5/23/2023   3:20 PM             AgentService
d-----          5/23/2023   3:20 PM             AML-packages
d-----          5/23/2023   3:20 PM             AmlDeploymentScripts
d-----          5/23/2023   3:20 PM             AutomatedProcedures
d-----          5/23/2023   3:20 PM             ConversionServer
d-----          5/23/2023   3:20 PM             Innovator
d-----          5/23/2023   3:20 PM             OAuthServer
d-----          5/23/2023   3:20 PM             SelfServiceReporting
d-----          5/23/2023   3:21 PM             src
d-----          5/23/2023   3:20 PM             Tests
d-----          5/23/2023   3:21 PM             TransformationsOfConfigFiles
d-----          5/23/2023   3:21 PM             VaultServer
-a-----          5/23/2023   3:20 PM             106 .aras.binaries.ignore
-a-----          5/23/2023   3:20 PM          11877 .encoding.ignore
-a-----          5/23/2023   3:20 PM          2569 .eslintignore
-a-----          5/23/2023   3:20 PM          1393 .eslinttrc.json
-a-----          5/23/2023   3:20 PM              7 .gitattributes
-a-----          5/23/2023   3:20 PM              23 .gitconfig
-a-----          5/23/2023   3:20 PM          5727 .gitignore
-a-----          5/23/2023   3:20 PM          5128 .jshintignore
-a-----          5/23/2023   3:20 PM              74 .jshinttrc
-a-----          5/23/2023   3:20 PM          441 .luGet.config
-a-----          5/23/2023   3:20 PM          4347 README.md
-a-----          5/23/2023   3:20 PM              46 README.txt
-a-----          5/23/2023   3:21 PM          9003 repo-init.ps1

PS C:\ArasProjects\Project1\LocalRepo\MyFork>

```

3. Run the following script: **repo-init.ps1**

A few more files are added after the **repo-init.ps1** script is run.



Administrator: Windows PowerShell

```

Directory: C:\ArasProjects\Project1\LocalRepo\MyFork

Node                LastWriteTime           Length Name
-----
d-----            5/23/2023  8:09 PM                .vscode
d-----            5/23/2023  8:09 PM            AgentService
d-----            5/23/2023  3:20 PM            AML-packages
d-----            5/23/2023  3:20 PM            AmlDeploymentScripts
d-----            5/23/2023  8:09 PM            AutomatedProcedures
d-----            5/23/2023  8:09 PM            ConversionServer
d-----            5/23/2023  8:06 PM            Documentation
d-----            5/23/2023  3:20 PM            Innovator
d-----            5/23/2023  8:09 PM            OAuthServer
d-----            5/23/2023  8:06 PM            Resources
d-----            5/23/2023  8:06 PM            Sample Data
d-----            5/23/2023  8:09 PM            SelfServiceReporting
d-----            5/23/2023  8:06 PM            src
d-----            5/23/2023  3:20 PM            Tests
d-----            5/23/2023  3:21 PM            TransformationsOfConfigFiles
d-----            5/23/2023  8:09 PM            VaultServer
-a-----            5/23/2023  3:20 PM             106 .aras.binaries.ignore
-a-----            5/23/2023  3:20 PM          11877 .encoding.ignore
-a-----            5/23/2023  3:20 PM          2569 .eslintignore
-a-----           12/28/2022  9:13 PM          1393 .eslintrc.json
-a-----            5/23/2023  3:20 PM              7 .gitattributes
-a-----            5/23/2023  3:20 PM             23 .gitconfig
-a-----            5/23/2023  8:09 PM          5727 .gitignore
-a-----            5/23/2023  3:20 PM          5128 .jshintignore
-a-----            5/23/2023  3:20 PM             74 .jshintrc
-a-----           12/28/2022  9:13 PM          582 ActivateInnovatorClientWatcher.ps1
-a-----           12/28/2022  9:13 PM          7645 build.ps1
-a-----           12/28/2022  9:13 PM          879 BuildAndDeploy.ps1
-a-----           12/28/2022  9:13 PM         3144 BuildDeploymentArtifact.ps1
-a-----           12/28/2022  9:13 PM         1155 CleanUp.ps1
-a-----           12/28/2022  9:13 PM         1483 ContinuousIntegration.ps1
-a-----            3/17/2022  2:33 PM          699 ConversionServerConfig.xml
-a-----           12/28/2022  9:13 PM         1810 DeployArtifact.ps1
-a-----            3/17/2022  2:33 PM          903 InnovatorServerConfig.xml
-a-----            5/23/2023  3:20 PM          441 NuGet.config
-a-----           12/28/2022  9:13 PM         5783 package.config
-a-----            5/23/2023  3:20 PM         4347 Readme.md
-a-----            5/23/2023  3:20 PM           46 Readme.txt
-a-----           12/28/2022  9:13 PM         9003 repo-init.ps1
-a-----           12/28/2022  9:13 PM          735 RunIntegrationTests.ps1
-a-----           12/28/2022  9:13 PM          732 RunSeleniumTests.ps1
-a-----            3/17/2022  2:33 PM          240 SelfServiceReportConfig.xml

PS C:\ArasProjects\Project1\LocalRepo\MyFork>

```



Local Environment Variables Set Up

A local environment is now established and connected to the Standard SDE.

The build and deployment scripts running in the SDE possess local equivalents that require specific details related to the given environment and the associated Git **Branch**. Property settings provide these details.

The property values have the following precedence. Based on this precedence, they can be set or overridden. The highest is evaluated last, as shown.

- **Default.Settings.include:** This file is located in ...\\AutomatedProcedures\\
- **Machine.Settings.include:** This file is located in C:\\ or an alternate root directory.
- **<ProjectPrefix>-<git-branch>.Settings.include:** The project Prefix is set in Default.Settings.include the Git **Branch** must match the **Branch** the user checks out.

The BuildAndDeploy.ps1 file is used to deploy Aras Innovator and the customization from the current directory.

The following steps outline the process to run the **BuildAndDeploy.ps1** script in a local environment:

1. Open Windows PowerShell as administrator and navigate to the working directory. For example, C:\\ArasProjects\\project1\\localRepo
2. Run the command: **.\BuildAndDeploy.ps1**

When the **BuildAndDeploy.ps1** script is run for the first time, it does several environment verifications and creates the following file: C:_machine_specific_includes\\Machine.Settings.include

```
You are about to get path to Machine specific Includes directory...
Do you want to create directory 'C:\\_machine_specific_includes'? [Y] Yes or [N] No (default is 'N'): Y
```



The Machine.Settings.include file contains a series of key/value pairs that are used by the build process to build Aras Innovator on the local machine.

The project prefix and a reference to the last **Commit** are used from an Include file in the following Local Repository: C:\ArasProjects\project1\localRepo\AutomatedProcedures\Default.Settings.include

The Default.Settings.include includes all the default properties required for a project.

When executing a script, a Variable is first loaded from Default.Settings.include. If the same Variable is defined in the Machine.Setting.include, the value is overwritten. Finally, the value is overwritten if the Variable is defined in a **Branch**-specific file.

3. Update the Machine.Setting.include file with the following:

```
<?xml version="1.0" encoding="utf-8" ?>
<project name="Default.Settings">
  <property name="Path.To.Baselines.Dir" overwrite="true" value="C:\ArasProjects\Baselines" />
  <property name="Path.To.DB.Bak" overwrite="true" value="C:\ArasProjects\Baselines\ArasSSETraining\CleanInnovator14SP3\DB.bak" />
  <property name="Path.To.CodeTree.Zip" overwrite="true" value="C:\ArasProjects\Baselines\ArasSSETraining\CleanInnovator14sp3\CodeTree.zip" />
  <property name="MSGQ.Server" overwrite="true" value="" />
  <property name="MSGQ.SA.Password" overwrite="true" value="" />
  <property name="MSGQ.Innovator.Password" overwrite="true" value="" />
  <property name="MSGQ.Innovator.Regular.Password" overwrite="true" value="" />

  <!-- Note: If you want to set licenses here, please make sure they have been removed from 'BranchSpecificSettingsFile' to avoid overwriting. -->
  <choose>
    <when test="$([string]:starts-with(Version.Of.Installed.Innovator, 11))">
      <!-- Here you can set all the properties specific to Innovator 11, in particular, the licenses -->
      <property name="Innovator.License.Type" overwrite="true" value="" />
      <property name="Innovator.License.Company" overwrite="true" value="" />
      <property name="Innovator.License.Key" overwrite="true" value="" />
      <property name="Innovator.Activation.Key" overwrite="true" value="" />
      <property name="Feature.License.Strings.List" overwrite="true" value="" />
    </when>
    <when test="$([string]:starts-with(Version.Of.Installed.Innovator, 12))">
      <!-- Here you can set all the properties specific to Innovator 12, in particular, the licenses -->
      <property name="Innovator.License.Type" overwrite="true" value="" />
      <property name="Innovator.License.Company" overwrite="true" value="" />
      <property name="Innovator.License.Key" overwrite="true" value="" />
      <property name="Innovator.Activation.Key" overwrite="true" value="" />
      <property name="Feature.License.Strings.List" overwrite="true" value="" />
    </when>
    <when test="$([string]:starts-with(Version.Of.Installed.Innovator, 14))">
      <!-- Here you can set all the properties specific to Innovator 14, in particular, the licenses -->
      <property name="Innovator.License.Type" overwrite="true" value="Unlimited" />
      <property name="Innovator.License.Company" overwrite="true" value="Aras Corporation" />
      <property name="Innovator.License.Key" overwrite="true" value="" />
      <property name="Innovator.Activation.Key" overwrite="true" value="" />
      <property name="Feature.License.Strings.List" overwrite="true" value="" />
    </when>
  </choose>
</project>
```

Project-Specific Settings:

- **Path to baseline directory:** For example, C:\ArasProjects\Baselines
- **Path.To.DB.Bak:** File path to a “clean” copy of the Aras Innovator solutions database. For example, C:\ArasProjects\project1\Baselines\CleanInnovatorxxSPyy\DB.bak
- **Path.To.CodeTree.Zip:** A file path to the CodeTree.zip archive containing the code tree of the production Aras Innovator instance. For example, C:\ArasProjects\project1\Baselines\CleanInnovatorxxSPyy\CodeTree.zip

Machine-Specific Properties:

- **Innovator.License.Type:** This is typically Unlimited, Version, or Verified, depending on the key being used for this project.
- **Innovator.License.Company:** The licensed company name.



- **Innovator.License.Key:** A license key used for the installation can be obtained from <http://www.aras.com/support/LicenseKeyService/> .
- **Innovator.Activation.Key:** An activation key for features that have been licensed for this project.
- **MSSQL.Server:** Name of the SQL Server instance (default is the local machine).
- **MSSQL.SA.Password:** The password for the sa (system administrator) login.
- **MSSQL.Innovator.Password:** Password for the Aras admin login (default is innovator).
- **Feature.License.Strings.List:** This is an optional property. It is required only when the user needs to set values for features such as TAF (Test Automation Framework) which is required as part of CI/CD.

Important

In SaaS, ensure that MSSQL.Innovator.Password and MSSQL.Innovator.Regular.Password meet Microsoft SQL Server complexity requirements. Passwords must comply with [SQL Server Password Policy](#) to be used with contained databases.

Failure to use complex passwords will result in BuildAndDeploy.ps1 errors during database setup.



Build and Deploy Locally

The cloned Repository on the local machine contains the actual definition of the customization project. The next step is to deploy it on the local machine.

The following steps outline the process of building and deploying Aras Innovator locally:

1. Open a new session of Windows PowerShell as administrator and navigate to the working directory. For example, C:\ArasProjects\project1\localRepo
2. Run the command: **.\BuildAndDeploy.ps1**

If any errors occur, make corrections in the Machine.Settings.include file and execute the same command again.

Once the process is complete, a new instance of Aras Innovator is deployed (the output from the batch file will indicate the URL). A new database is created based on the values provided in the settings file, and Aras Innovator is ready to run.

http://localhost/[MachineName]-[Prefix]-[branch]

SQL Server DB is restored:

[MachineName]-[Prefix]-[branch]

```
[exec] reporting feature licenses to the sql database component.
[exec]
[exec] The "root" user login is enabled for "database" database.
[exec]
[exec] Print url of installed Innovator:
[exec] -----
[exec]
[exec] URL of configured Innovator is: http://localhost/1813-18F-884406-ArasSDETraining-Training
[exec] -----
[exec]
[exec] SUCCEEDED.
[exec]
[exec] Press any key to continue . . .
[exec] Starting "powershell.exe (-command "subst t: /d")" in "C:\ArasProjects\Project1\localRepo\Myfork\AutomatedProcedures\Targets"

BUILD SUCCEEDED
Total time: 21.3 seconds.

AdaptInnovatorForDeveloperEnvironment:
[exec] C:\ArasProjects\Project1\localRepo\Myfork\AutomatedProcedures\Targets\AdaptInnovatorForDeveloperEnvironment.xml
Buildfile: file:///C:/ArasProjects/Project1/localRepo/Myfork/AutomatedProcedures/Targets/AdaptInnovatorForDeveloperEnvironment.xml
Target framework: Microsoft .NET Framework 4.0
Target(s) specified: _AdaptInnovatorForDeveloperEnvironment

_AdaptInnovatorForDeveloperEnvironment:

BUILD SUCCEEDED
Total time: 0 seconds.

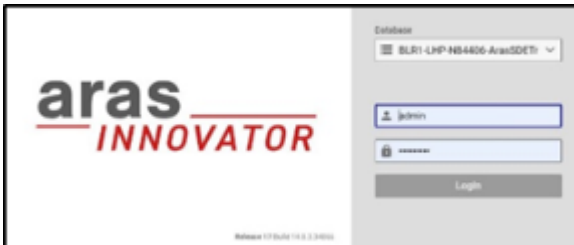
BUILD SUCCEEDED
Total time: 593.8 seconds.
SUCCESS!!!
PS C:\ArasProjects\Project1\localRepo\Myfork >
```



3. Copy the URL and paste it into the browser.
4. Log in to Aras Innovator as an administrator with the following credentials:

Username: admin

Password: innovator



At this point, contributors have a basic work environment set up and are ready to contribute as members of a customization project team.

As mentioned earlier, a central focus of DevOps is to instill a culture and discipline to ensure proper management of the solution development process and the transition of a well-managed solution configuration into business-critical operations.

The following two sections introduce two essential aspects of the DevOps culture and lean agile development in general.

Partial Local Deployment

If you already have a deployed instance of Aras Innovator with data (see section 3.2.6 Build and Deploy Locally), and you want to iteratively update it during development, you can perform a partial deployment to speed up the process and avoid unnecessary steps.

By default, `.\BuildAndDeploy.ps1` execution without any parameters will perform the following actions:

1. Validate and install build machine prerequisites
2. Remove installed Aras Innovator instance
3. Install new Aras Innovator instance from baseline
4. Build `CustomSolutions.sln`
5. Build new deployment package with all kinds of customizations such as AML changes, Code Tree changes, and Configuration transformation scripts
6. Deploy the package on top of the new Aras Innovator instance.

As a result, each time the `BuildAndDeploy.ps1` script is executed without parameters, a fresh instance is deployed from scratch — even for minor changes like updating the login page image.

To accelerate development and avoid re-running unnecessary steps, you can use delta deployment option parameters of `BuildAndDeploy.ps1` for partial deployments.

Important

A full deployment must be executed at least once before using any delta deployment options. These options update only an existing instance and will fail if no instance is found.

Important

If `.\BuildAndDeploy.ps1` with delta deployment options failed or you are not sure that update can be deployed on top of existing Aras Innovator instance, please use `.\BuildAndDeploy.ps1` without any parameters.

Deployment Option Parameters

`-UpdateInstance`



Deploys all types of customizations (AML changes, CodeTree changes, and Configuration transformation scripts) on top of the existing Aras Innovator instance without removing it.

```
BuildAndDeploy.ps1 -UpdateInstance  
-UpdateAMLOnly
```

Deploys only AML customizations to the database of the existing Aras Innovator instance.

```
BuildAndDeploy.ps1 -UpdateAMLOnly  
-UpdateCodeTreeOnly
```

Deploys only the CodeTree customizations (e.g., server-side and client-side code changes) without affecting AML or configuration files.

```
BuildAndDeploy.ps1 -UpdateCodeTreeOnly  
-UpdateConfigurationOnly
```

Deploys only configuration transformation customizations (e.g., changes in config files in TransformationsOfConfigFiles folder) on top of the existing instance.

```
BuildAndDeploy.ps1 -UpdateConfigurationOnly
```

Important

Deployment package that is built during local delta deployment is stored in AutomatedProceduresOutput/UpdatePackage. Do not use this deployment package outside of the existing instance as far as it is partial deployment package and can't be reused for other deployments. To determine what kind of customizations are in the update deployment package, you can look at "PackageInfo.md" from the package content.

Skip Prerequisite Installation

When running a full `.\BuildAndDeploy.ps1`, the script validates and installs build machine prerequisites and restores packages. This step is also executed by default when running delta deployments, which can take time even if everything is already set up.



If you are sure that the Aras.DevOps packages are already restored and prerequisites are installed, you can skip this step using the `-SkipPrerequisites` parameter:

```
BuildAndDeploy.ps1 -SkipPrerequisites
```

Important

Do not skip prerequisites when updating to a new version of the Aras.DevOps packages. Run the full `BuildAndDeploy.ps1` script first to ensure everything is validated and restored properly. After that, you may add `-SkipPrerequisites` to reduce execution time for subsequent deployments.

Example: Fastest Way to Deploy Code Customization

If you have an already configured environment and an existing deployed instance, and you only want to update the CodeTree (e.g., UI or server code), use the following command for the fastest deployment:

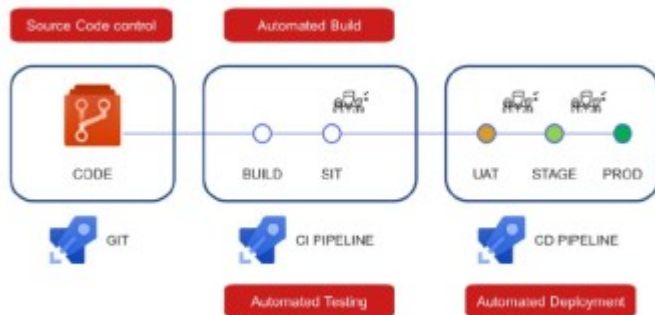
```
BuildAndDeploy.ps1 -UpdateCodeTreeOnly -SkipPrerequisites
```

This will apply CodeTree customizations directly to the existing instance without re-installing or validating prerequisites.



Continuous Integration and Continuous Delivery (CI/CD)

Aras DevOps is based on Continuous Integration (CI) and Continuous Delivery (CD) practices.



The above illustration shows the basic flow of the CI and CD process of merging code changes from multiple contributors to a single Repository.

- Code equals source code control. Source code control includes items such as AML Packages, configuration files, and settings. It includes the Code Tree modification and the various libraries that constitute the solution.
- **Commits** are means to manage changes that take solutions from one configuration to the next.
- The CI **Pipeline** supports Continuous Integration, where various contributors use **Pull requests (PRs)** to submit their contributions to the integrated solution. The system automatically builds and runs any available automated tests written by developers. If automated tests fail, the **PR** will fail. The developer will need to fix their code to successfully submit their **PR**. Reviewers check the work before accepting it.
- The resulting **Artifacts** with successful **Pipelines** are candidates for deployment to a System Integration Testing (SIT) instance for manual testing.

Continuous Delivery is an approach where teams release quality products frequently and predictably from a source code Repository to production in an automated fashion. Once code has been tested and built as part of the CI process, Continuous Delivery takes over during the final stages to ensure it can be deployed as packaged, with everything it needs to deploy to any environment at any time.

Continuous Delivery can cover everything from provisioning the infrastructure to deploying the application to the testing or production environment.

1. **Important**

When Aras DevOps is purchased separately (as opposed to within an Aras Enterprise subscription), the customer independently hosts the UAT, staging, and production environments on their own infrastructure. The creation of these **Pipelines** falls completely outside the scope of Aras DevOps.



Testing

Testing is an integral part of agile and lean methodology:

1. **Continuous Improvement:** Regular testing provides feedback that aids in constant product refinement, catching bugs early, and improving quality.
2. **Customer Satisfaction:** Testing ensures the product meets customer requirements and functions as expected, enhancing user experience.
3. **Risk Mitigation:** Testing identifies potential issues early, reducing the risk of major problems and saving time and resources.
4. **Collaboration:** Testing fosters better communication and understanding between developers and testers.
5. **Rapid Delivery:** Continuous testing supports Agile and Lean's emphasis on frequent, incremental software delivery.
6. **Built-In Quality:** In Lean, quality is embedded in the development process, so every piece of code is tested as it's developed.
7. **Adaptability:** Testing ensures changes made during the project don't negatively impact the system.

The Aras Test Automation Framework (TAF) is a framework for writing and executing automated tests for the Aras Innovator Platform included within Aras DevOps. It is a set of APIs that abstracts the complexity of the underlying testing frameworks used: Selenium, NUnit and TestRunner. TAF greatly reduces the brittleness of automated tests as the underlying browser and application technologies change. The goal of TAF is to absorb the changes made by, say, Google Chrome or Aras Innovator, such that tests written on one version for a piece of functionality continue to work on the next version of Aras Innovator for the same functionality.

TAF includes APIs for automated tests to cover the following:

- Web UI (Selenium)
- Integration (AML)

To support the demand for Continuous Integration and Continuous Delivery (CI/CD), automated testing increases an organization's ability to thoroughly test new functionality and applications as fast as they are developed. This framework is built to work with the Aras Innovator Platform to improve the development process's speed, cost, and quality.



TAF has a framework for writing web-functional and end-to-end tests that use the Aras Innovator Web Client like an end-user.

The following section details the steps for an individual contributor to make changes within the Local Development Environment and submit it for integration into the SDE.



Contributor Process

The Local Development Environment (LDE) section introduced the LDE and explained the configuration process and procedures for building and deploying a local instance of Aras Innovator.

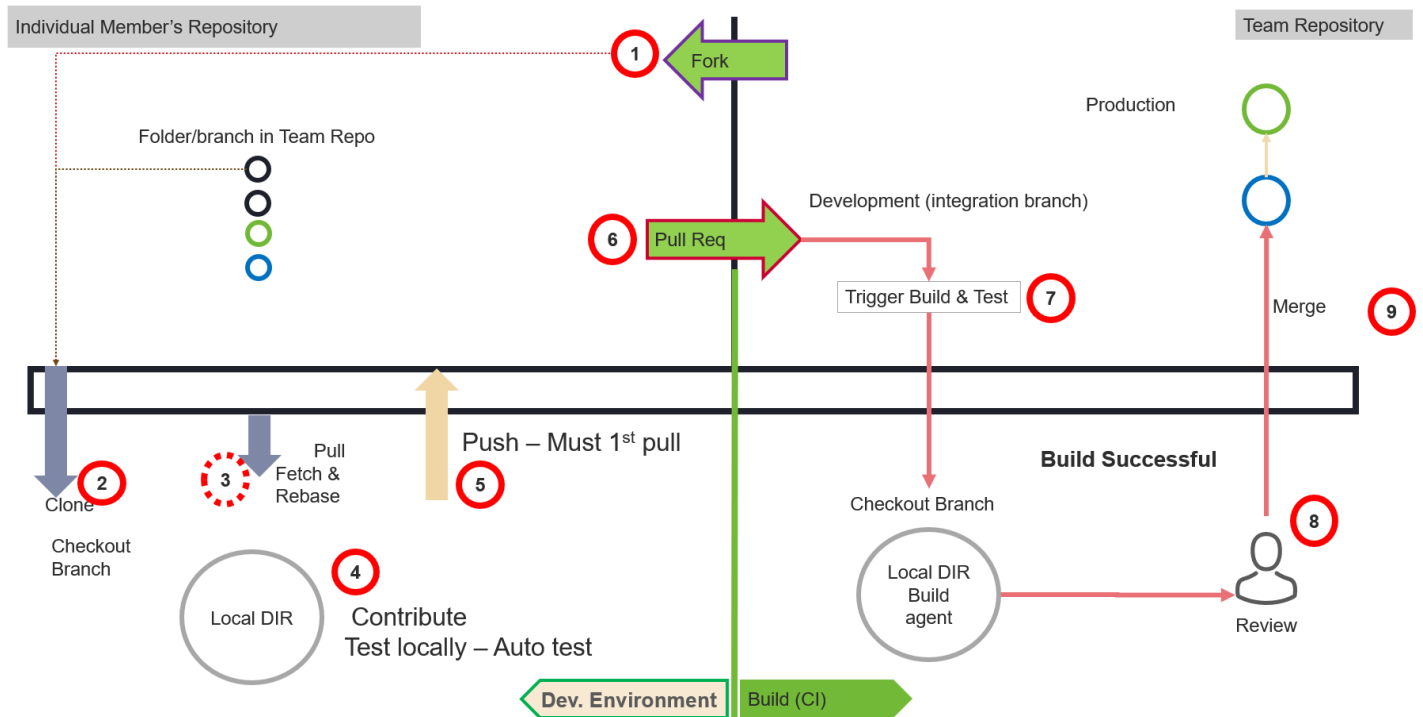
This section outlines how to enable a contributor to make contributions.



The Contribution Process Overview

Developers, test/QA engineers, and other team stakeholders are considered Contributors. Contributors who may contribute source files (configuration files, source code C#, etc.) must create a Fork to manage their work in the environment.

The Contribution Process



1. Create a Fork (copy of the Shared Team Repository).
2. Clone the Fork to the local environment and check out the **Branch** to work on.
3. Add a Remote Reference to the Team **Repo** to fetch and rebase its current work.
4. Make changes in the Repository and test locally by running **RunIntegrateTest**, **ContinuousIntegration**, and other scripts.
5. Commit the changes locally and then push.
6. Create the **Pull request**.
7. Updates to the Source **Branch** after the **Pull request** is created and the initial creation triggers a **ContinuousIntegrationPipeline** based on **Branch** policy.
8. When the **ContinuousIntegrationPipeline** has successfully built the **Artifact**, it will run a test and report a green status or not. Based on that status, reviewers may approve the **PR**. They



may also reject it, even with a green Build, if some other project practice is violated.

9. When the **PR** is approved and merged, the **Branch** typically has a policy of rerunning the **ContinuousIntegrationPipeline**. This ensures that all **PRs** are still in sync. The goal is to ensure that the Common **RepoBranch** is always buildable.

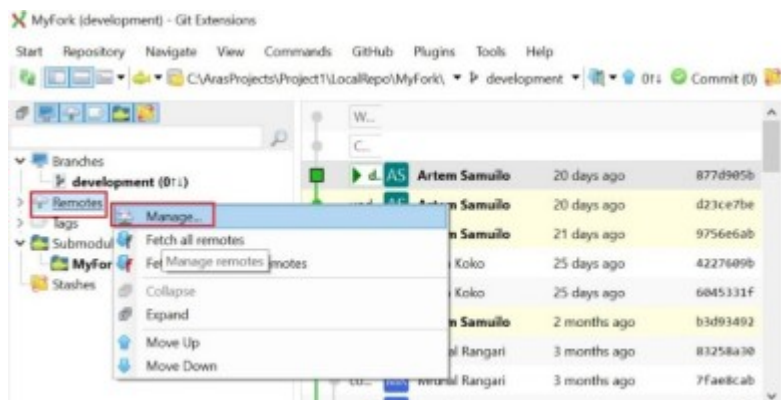


Adding Remote Reference

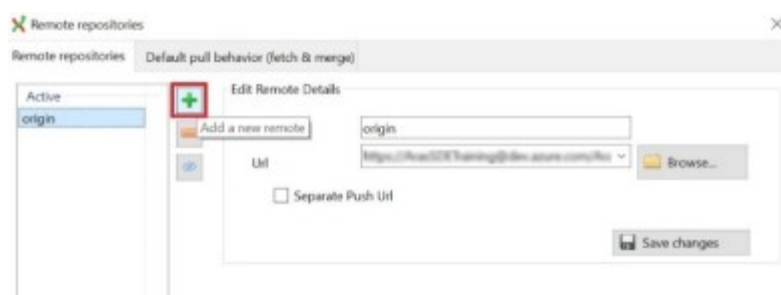
The contributor needs to establish a Remote Reference to incorporate the current work with the Team Repository and keep it up to date. By adding this Reference, the contributor connects the local and Shared Team Repositories. This allows them to fetch the latest changes made by the team and rebase the contributor's work on top of them. Cloning and other preliminary steps have already been completed; this Reference addition is specifically meant to facilitate contributions.

The following steps outline the process of creating additional Remotes:

1. Launch the chosen Version Control Tool on the local machine.
2. Use the Version Control Tool to navigate to the Local Repository where the user wants to add the Remote Reference.
3. Find the option or command within the Version Control Tool to add a Remote Reference or Remote Repository.

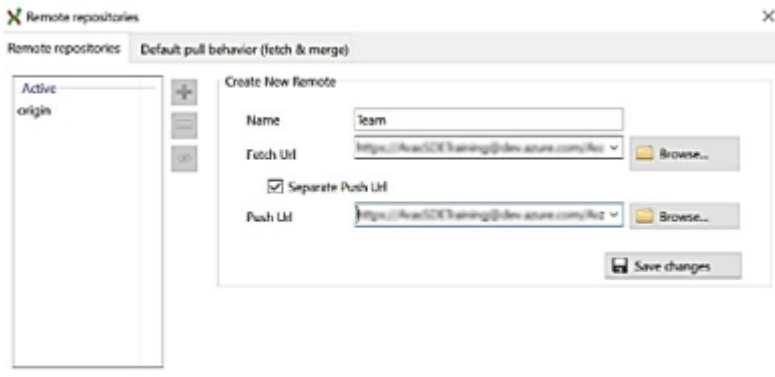


4. Click the + icon to **Add a new remote**.

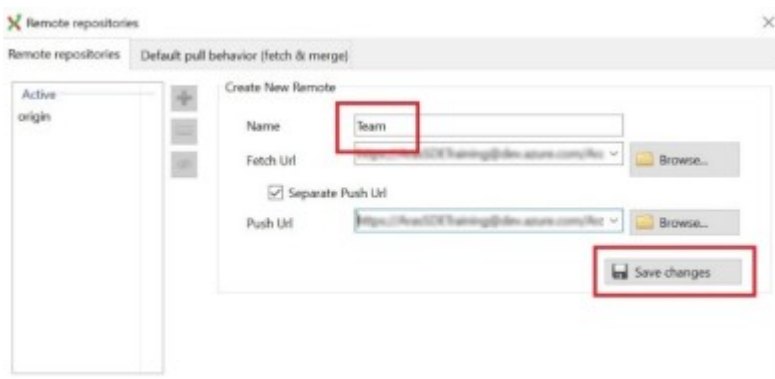


5. Enter the Remote Repository URL into the appropriate field or prompt within the Version Control Tool.

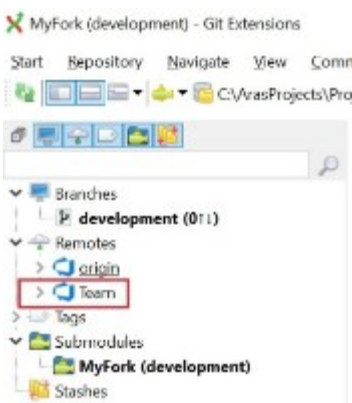




6. Optional: Assign a name to the Remote Reference to make it easier to reference in the future.



7. Once the necessary details are entered and any optional settings are configured, save and add the Remote Reference.



Making Changes in Local Repo

As an individual contributor, make changes in the Local **Repo** and test them locally before submitting them to merge into the team's work. Modifying the Local Repository of Aras Innovator involves changing the files and configurations stored on the local machine. These changes can include customizations to **ItemTypes**, **Forms**, **Workflows**, **Reports**, and other Items of Aras Innovator.



Add or Modify file in Code Tree

If the Nuget Package Aras.Crt.Core.1.1.XXX is present in the project's package.config file in the Project Repository, only new or changed files related to the project customization should be committed to the Git Repository. Users must commit any customized file into the Code Tree folder in the Project Repository.

The following steps outline the process of adding or modifying files in the Code Tree:

1. Identify the file that needs to be modified in the Project Repository.

For example, a user may want to modify the file Inbasket.html (Innovator\Client\scripts\InBasket\InBasket.aspx) to fulfill UI requirements of the implementation.

2. To customize a service, create a folder with a service's name under the Code Tree folder in the Project Repository.

For example, the following structure must be recreated: CodeTree\Innovator\Client\scripts\InBasket in the Git Repository under the Code Tree folder.



3. Commit the non-modified files before making any changes to those files.

1. Important

Configuration files should not be modified and committed in the Repository. Transformation Config functionality must be used to make changes in configuration files.

4. Make the required modifications to the files.
5. Commit the modified files.



1. **Important**

To modify an existing file in the deployed instance, it is recommended to first make a clean **Commit** and then perform the modification in a separate **Commit** to preserve the history of the change for future developers. However, this is not required to deploy the modified file.



Exporting Packages

Once the changes made in the Package Definition are complete, the Package can be exported using the **Export** Tool. This utility is a separate executable file named Export.exe that is available on the Aras Innovator CD image or can be downloaded from <https://www.aras.com/en/support/downloads> .

The **Export** Tool selects a Package Definition from the database and creates a Package folder structure in the file system. Each Package Group (**ItemType**, **Form**, etc.) becomes a separate subfolder in the file. Each exported Item is represented as an AML file with the same name as the exported Item within each subfolder.

The following steps outline the process of running the **Export** Tool:

1. Download the **Export** Tool from <https://www.aras.com/en/support/downloads> .
2. Run **Export.exe** as administrator.



Make Required Changes in Aras Innovator Instance

Login into the Aras Innovator Instance and make customizations per the project requirements.



Export Package After the Changes

Once all the changes are made to the Aras Innovator instance, the next step is to export those changes.

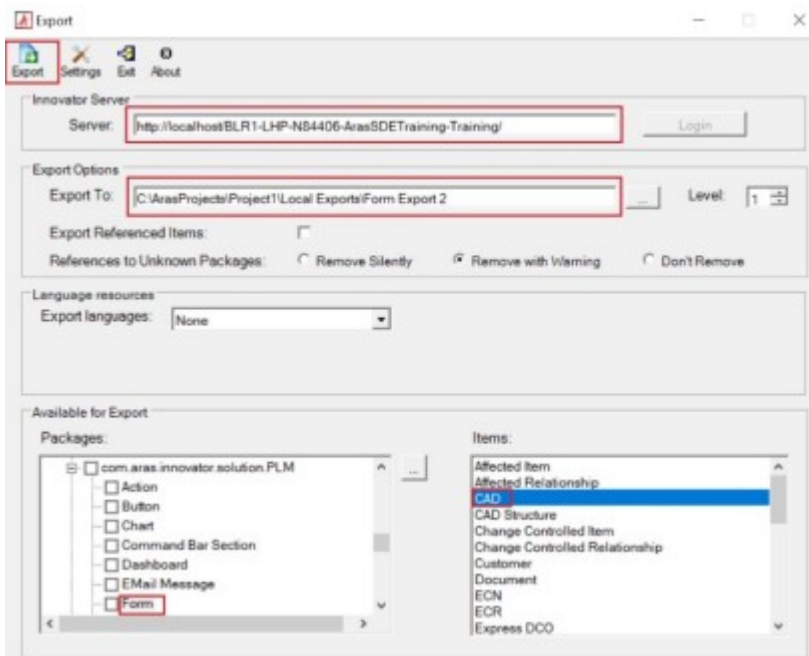
The following steps outline the process of exporting the Packages after the changes:

1. Create a folder to export the changes made. For example,

C:\ArasProjects\Project1\Local Exports\Form Export 2

All the exported data will be stored in the folder created in the first step.

2. Using the **Export Tool**, perform the following steps:



- **Server:** Enter the Aras Innovator server URL.
- **Login:** Click the **Login** button then fill in the **Username** admin and its **Password**.
- **Export To:** Set the new destination of the Export. All the exported files will be stored in that directory. For example,

C:\ArasProjects\Project1\Local Exports\Form Export 2

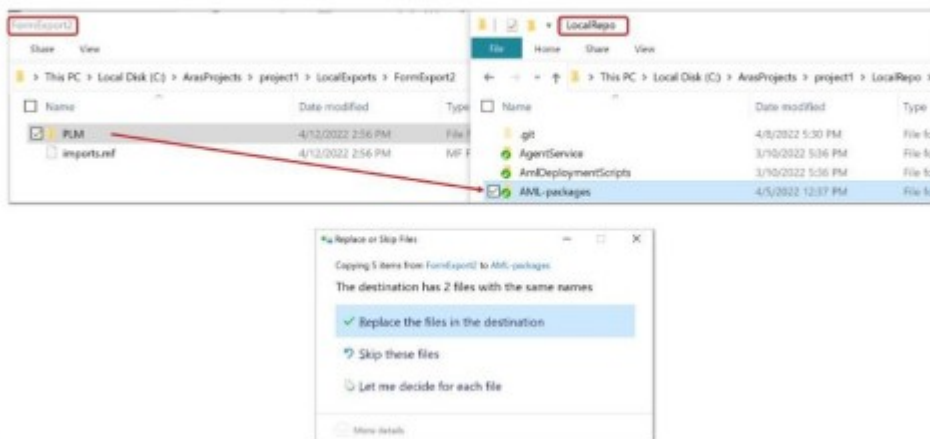
- **Packages:** Click the ellipsis (...) button, locate the changed **Package Definition**, and select the Items.
3. Click the **Export** button.



Copying the Export Utility's Output to the Local Repo

Once the changes are made and the files are exported, copy the top common folder and paste it into the local working directory of the Repository (**AML-Packages**). During this process, be sure to accept any file replacement warnings that may appear.

Copy the Export Utility's Output to the Local Repo



Staging Modified Files

Once the necessary changes have been made to the file, it is important to stage it before committing the changes to the Version Control System (Git). By staging the modified file, the user prepares it to be included in the next **Commit**, ensuring that the changes will be properly recorded and tracked within the Version Control System (Git).

The following steps outline the process of staging the modified file:

1. In the terminal (Terminal, Git Bash, or Windows Command Prompt), navigate to the **Repo** to which the newly exported file is copied, for example,

```
C:\ArasProjects\project1\LocalRepo\AML-packages\PLM.
```

2. Verify the status of the Repository by running the appropriate command.
3. Stage a file.
4. Verify now the new status of the Repository.
5. Commit the file.
6. Confirm the changes in the Version Control System.



Continuous Integration Script

An automated utility script is provided as part of each customer Repository to perform final validation and verification that a Build is successful.

Developers or system integrators can manually run this script to determine whether the Build passes or fails. Automation tools are also available to schedule executions of this script on a dedicated CI server.

The CI script runs all the unit and integration tests created for a project by installing and building a new instance of Aras Innovator, applying the project code and configuration, and ensuring that all tests are successful.

If issues need to be resolved, a report indicates success or failure with a running log. The script then deletes the running instance (and database).



Test the Deployment Locally

Aras Innovator should be redeployed using the script **./BuildAndDeploy.ps1**. Upon successful execution, a green SUCCESS message will appear.

```

Administrator: Windows PowerShell

[exec] Importing feature licenses to the all database components.
[exec]
[exec] The 'root' user logon is enabled for 'Database' database.
[exec]
[exec]
[exec] Print.url.of.Installed.Innovator:
[exec] -----
[exec]
[exec] *****
[exec] URL of configured Innovator is: http://localhost/BLR1-UHP-RB4406-ArasSDETTraining-Training
[exec] *****
[exec]
[exec] SUCCEEDED.
[exec]
[exec] Press any key to continue . . .
[exec] Starting 'powershell.exe {-Command "subst t: /D"}' in 'C:\ArasProjects\Project1\LocalRepo\MyFork\AutomatedProcedures\Targets'

BUILD SUCCEEDED

Total time: 21.3 seconds.

AdaptInnovatorForDeveloperEnvironment:

[nant] C:\ArasProjects\Project1\LocalRepo\MyFork\AutomatedProcedures\Targets\AdaptInnovatorForDeveloperEnvironment.xml
buildfile: file:///C:/ArasProjects/Project1/LocalRepo/MyFork/AutomatedProcedures/Targets/AdaptInnovatorForDeveloperEnvironment.xml
Target framework: Microsoft .NET Framework 4.0
Target(s) specified: _AdaptInnovatorForDeveloperEnvironment

_AdaptInnovatorForDeveloperEnvironment:

BUILD SUCCEEDED

Total time: 0 seconds.

BUILD SUCCEEDED

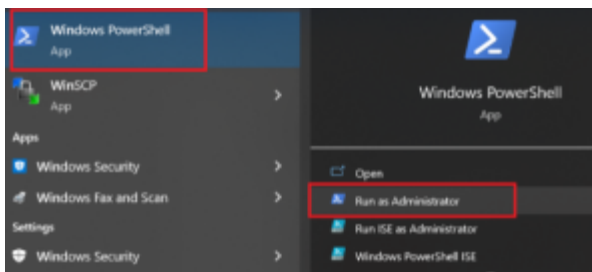
Total time: 593.8 seconds.

SUCCESS!!!
PS C:\ArasProjects\Project1\LocalRepo\MyFork>

```

The following steps outline the process of Rebuilding the Aras Innovator:

1. Open Windows PowerShell and run as Administrator.



2. Access the Local Repository: C:\ArasProjects\project1\LocalRepo
3. Run the **./BuildAndDeploy.ps1** script.



```
Administrator: Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\windows\system32> cd C:\ArasProjects\Project1\LocalRepo\MyFork\
PS C:\ArasProjects\Project1\LocalRepo\MyFork> ./BuildAndDeploy.ps1
```

4. Review the changes made in the newly rebuilt Aras Innovator instance.



Pushing Changes to Fork

Before creating a **Pull request** to share the work with the rest of the team, the developer must push the changes to the Fork. By pushing the changes, a developer makes them available for others to review, collaborate on, and merge into the original Repository if necessary.

A Fork in Git refers to a copy of a Repository that is created in a developer's Local Repository. Forking allows users to contribute to an open-source project or collaborate with others without affecting the original Repository.



Fetching Changes/Rebasing

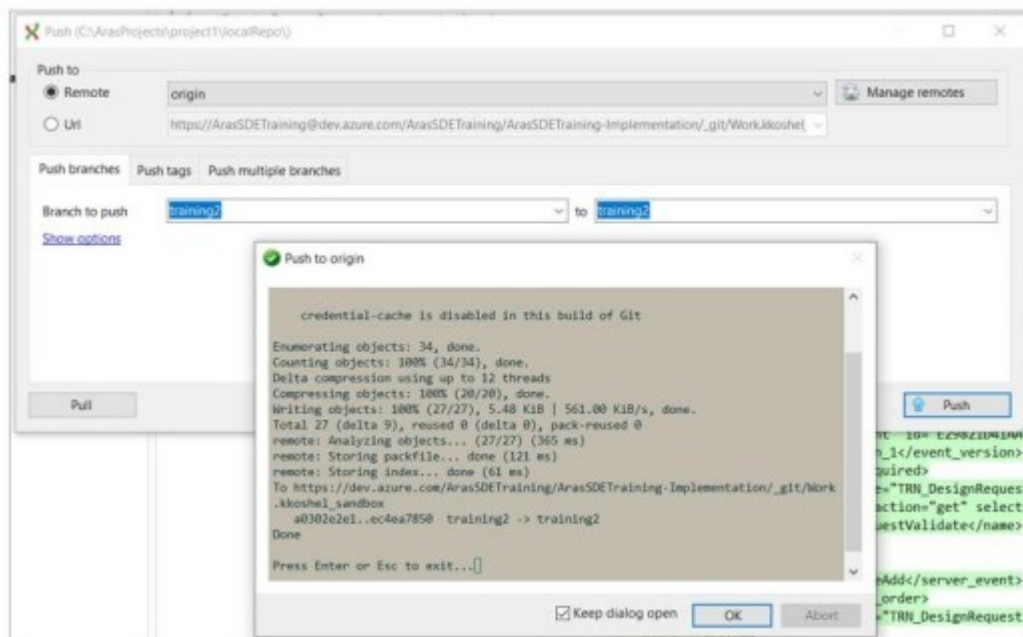
While the implementation was in progress, modifications or updates may have been made and pushed to the Team Repository. Hence, it is important to fetch the latest state of the Remote Repository and rebase our changes on top of it.

Fetch the changes using the desired Version Control System. The screenshot provided below serves as a visual representation, illustrating an example of the fetching process performed using the Git Extension.



Pushing Changes to Fork

Before creating a **Pull request** to share the work with the rest of the team, it is important to push the changes to the Fork. The screenshot below is a visual representation, illustrating an example of pushing changes to the Fork performed using the Git Extension.



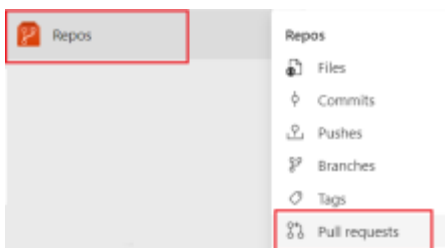
Creating a Pull Request

A **Pull request (PR)** is a way for developers to propose changes to a codebase and collaborate with others to review and merge those changes. Some of the key features of a **Pull request** include:

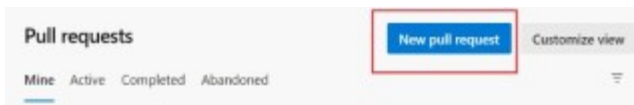
- **Code Changes:** A **Pull request** includes the changes made to the codebase by the developer. Other developers can view and review the changes, and any feedback or comments can be added directly to the **Pull request**.
- **Discussion and Feedback:** **Pull requests** provide a platform for developers to discuss and give feedback on the proposed changes. This allows for collaborative review and discussion of the code, ensuring that any issues are caught and resolved before the changes are merged into the codebase.
- **Automated Tests:** **Pull requests** can be configured to run automated tests, ensuring that the changes don't break existing functionality. This helps to catch any issues early on before the code is merged into the codebase.
- **Reviewers:** **Pull requests** can be assigned to specific reviewers who are responsible for reviewing the proposed changes. Reviewers can add comments and suggestions and approve or reject the changes before merging.
- **Status Checks:** **Pull requests** can be configured to include status checks, verifying that the changes meet certain criteria before merging. For example, a status check might ensure that all automated tests pass or that the code meets certain coding standards.
- **Mergeability:** **Pull requests** are merged into the codebase once reviewed and approved. This ensures that the changes are properly integrated into the codebase and that any conflicts with other changes are resolved.

The following steps outline the process of creating a **Pull request**:

1. Navigate to <https://dev.azure.com/{organization}/{project}>.
2. Click **Repos** and select **Pull requests**.



3. Click the **New pull request** button.



The **New pull request** Form appears:

The screenshot shows the 'New pull request' form. It includes the following fields and elements, each highlighted with a red box and labeled with a letter:

- a**: Source repository and branch selection (Work.kkoshel_sandbox into training2).
- b**: Target repository and branch selection (Work into training2).
- c**: Overview, Files (7), and Commits (2) tabs.
- d**: Title and Description text areas. The title is 'TRN-0001 Add Design Request' and the description is also 'TRN-0001 Add Design Request'. There is an 'Add commit messages' button.
- e**: Reviewers section with 'Add required reviewers' button and a selected reviewer 'kkoshel'.
- f**: Work items to link section with a search box 'Search work items by ID or title'.
- g**: Tags text area.

A 'Create' button is located at the bottom right of the form.

Enter the details as follows:

1. **Source Repo** or **Branch**: User's Fork.
2. **Target Repo** or **Branch**: Team **Repo**.
3. Review the **Files** and **Commits** to be included.
4. **Title** and **Description**: User Story name.
5. **Select** Reviewers.
6. Add **Work items to link**: Identify **Work Items** in the Azure DevOps as applicable.
7. Add **Tags** if needed.
8. Click the **Create** button.



Trigger, Build, and Test

The contributor's contribution is built and validated through the CI **Pipeline**. The **Pull request** triggers the CI **Pipeline**, which runs tests and checks against the changes.

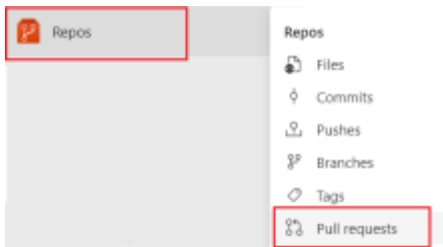


Reviewing a Pull Request

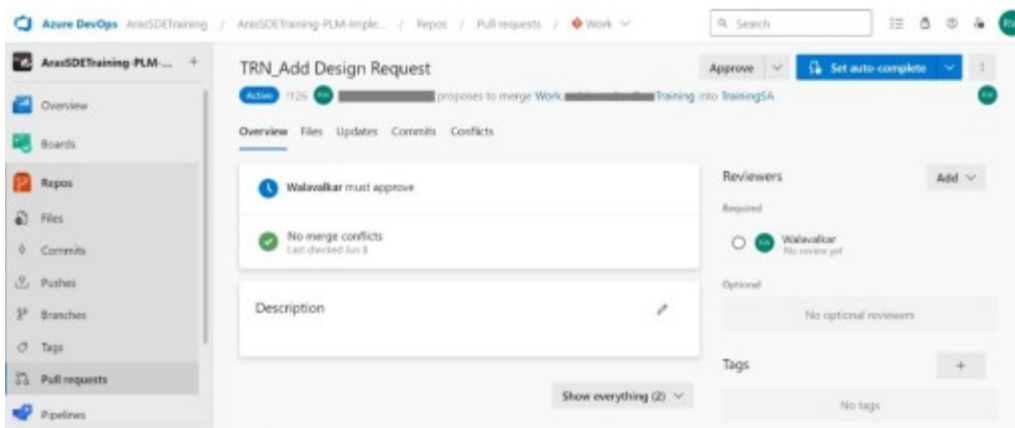
Each developer does not have **Push** (or write) access to the Central Remote Repository. They must issue a **Pull request** so code reviewers can inspect their work. The code reviewer then examines the developer's work, approves (or rejects) it, and applies changes to the Central Repository or communicates with the developer to make corrections.

The following steps outline the process of Reviewing a **Pull request**:

1. Navigate to <https://dev.azure.com/{organization}/{project}>.
2. Click **Repos** and select **Pull requests**.



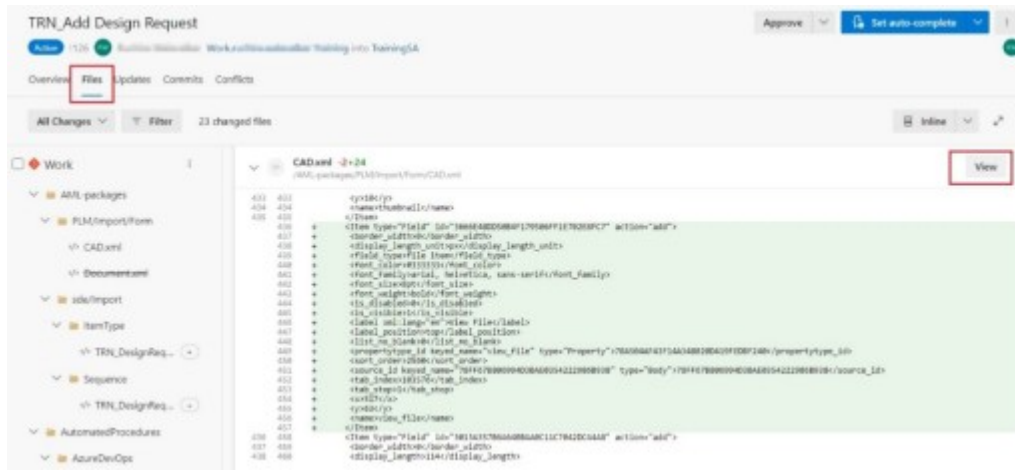
3. In the **Active** Tab, select the required **Pull request**.
4. In the **Overview** Tab of a **PR**, see the **Title**, **Description**, **Reviewers**, **Linked Worked Items**, **History**, **Status**, and **Comments**. Read the **PRDescription** to see the proposed changes. View the **Comments** to understand the issues raised by other reviewers.



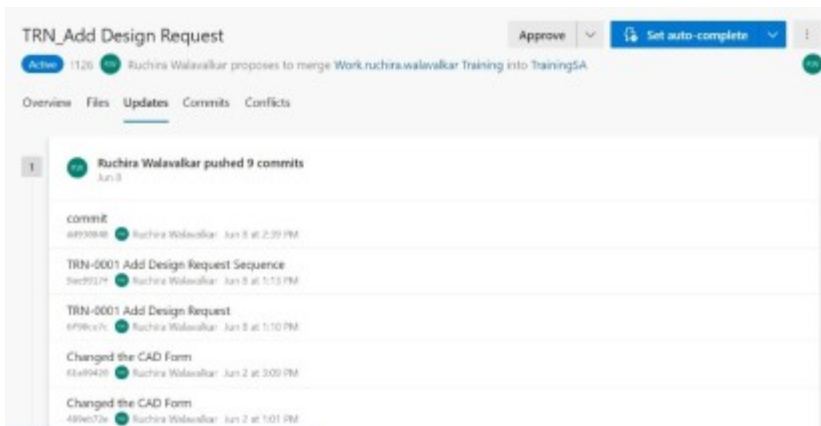
5. Select the **Files** Tab to review all **PR's** Source **Branch** content changes. The initial view shows a summary view of all file changes. Choose the **View** button next to a file to view only that file's



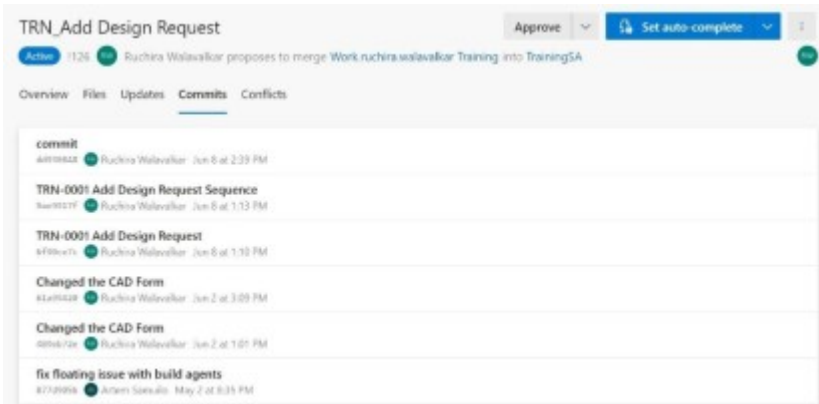
changes. The **View** button opens a Diff View if the file was modified or a content pane if the file was added or deleted.



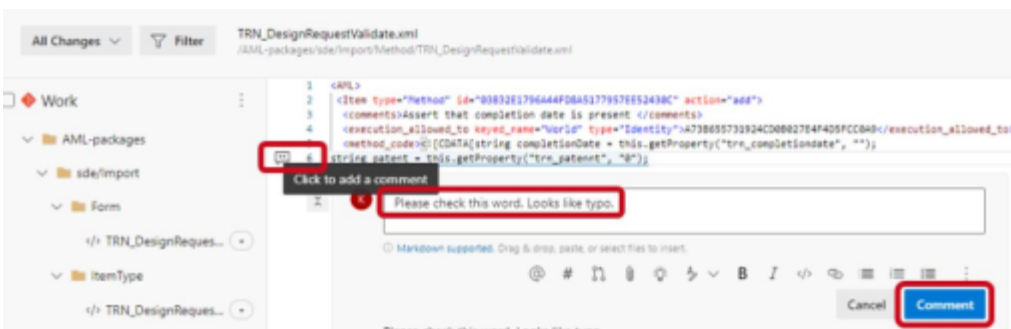
- In a Diff View for a file, select either a Side-by-side or Inline Diff layout.
- To review the **Changeset** introduced by specific **Pushes** to the Source **Branch**, select one or more **Changesets** from the **Changes** drop-down list. When one or more **Changesets** is selected, the Diff View updates to show only the changes from the selected **Changesets**. This feature is useful when changes have been pushed to the **PR** since the last review and a user just wants to see the new changes. The **Changes** dropdown list names each **Changeset** with the **Commit** Message from the final **Commit** in each **Push** operation.
- Choose the **Updates** Tab to view all pushed **Changesets** to ensure any Source **Branch** changes are not missed. The **Changesets** are numbered, and the most recent **Changeset** appears at the top of the list. Each **Changeset** shows the **Commits** that were pushed in that **Push** operation. A force-pushed **Changeset** won't overwrite the **Changeset** History and will show up in the **Changeset** list like any other **Changeset**.



- Choose the **Commits** Tab to view the **Commit** History of the Source **Branch** after it diverges from the Target **Branch**. The **Commit** History in the **Commits** Tab will be overwritten if the **PR** author force-pushes a different **Commit** History, so the **Commits** shown in the **Commits** Tab might differ from those shown in the **Updates** Tab.



- If there are any issues or questions about the changes provided by the developer, the reviewer can leave a comment for the developer. In the left panel, select the Method file, hover over the line to comment on, and select the **Comment** button to open an inline **Comment** box. The user can also select multiple lines and then select the **Comment** button that appears when hovering over those lines.



- As the **PR** author, the developer must resolve the Comment by taking the appropriate action (e.g., making the changes in their Local Repository) and then submitting the change for approval through the **PR** request workflow.



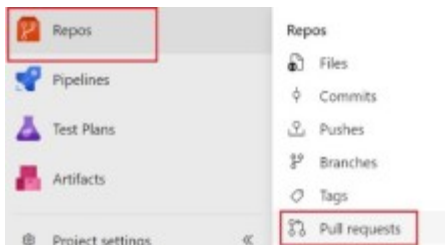
Merging the Pull Request

If the reviewer agrees with the proposed changes, the **Merge** operation combines changes into the Central Repository.

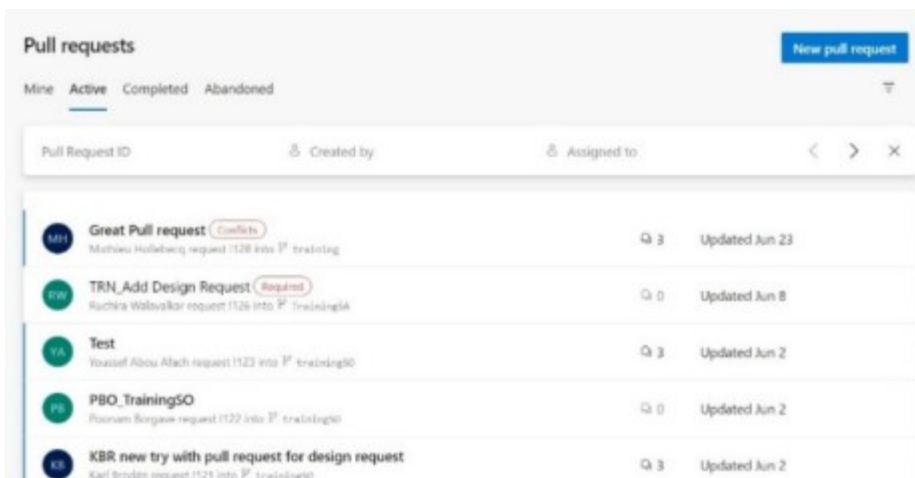
When the reviewer clicks the **Merge** button, the developers' proposed **Commit(s)** are merged into the Central Repository (Team **Repo**).

The following steps outline the process of **Merging the Pull Request**:

1. Navigate to <https://dev.azure.com/{organization}/{project}>.
2. Click **Repos** and select **Pull requests**.

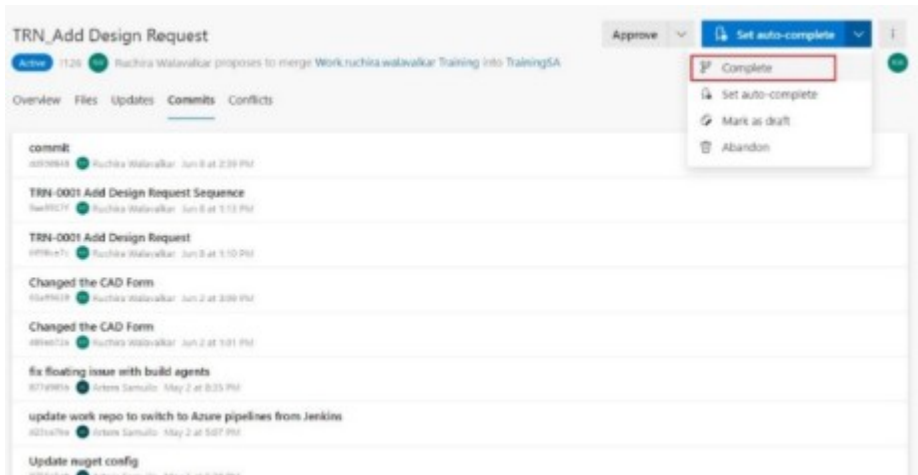


3. In the **Active** Tab select the required **Pull request**.

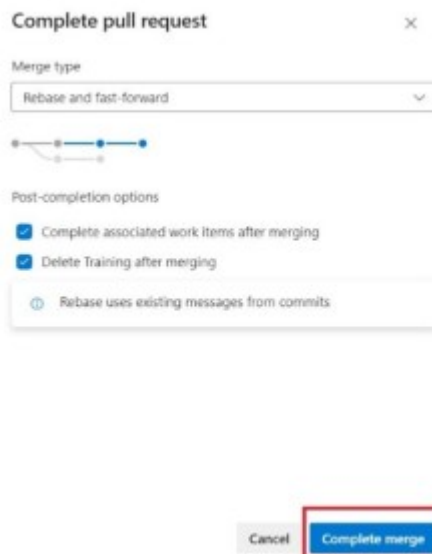


4. Select **Complete** on the upper right to complete the **PR**.





5. In the **Complete pull request** pane, under **Merge type**, select **Rebase and fast-forward** and click on **Complete merge**.



The completion of the **PR** triggers a new Build.

6. Verify that the Build is green (successful) by clicking **Pipelines** in the left menu, selecting **Pipelines** in the right menu, and selecting the **Recent** Tab.

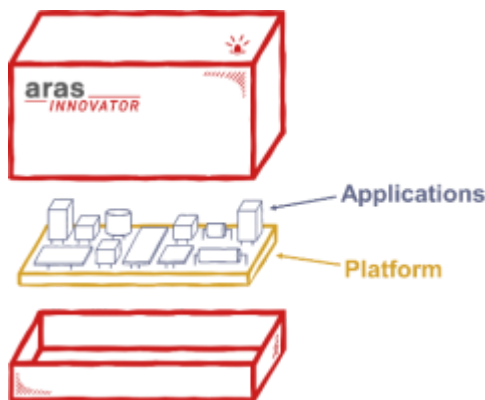


Preparing the Project's Initial Baseline

A Clean **Baseline** refers to the original or initial plan at the beginning of a project, which is free from any changes. It represents the unaltered version of the project plan, typically provided by the project management team or stakeholders at the beginning of the project.

A **Baseline** serves as a reference point and a point of comparison throughout the project's life cycle. It helps track and manage project progress, identify deviations from the original plan, and assess the impact of changes and risks as the project evolves.

The diagram below illustrates Aras Innovator's structure, comprising a platform and various applications. When the SDE is delivered, it includes the platform, and the specific project can choose the necessary applications and components. Making these decisions early on and creating a new project **Baseline** is usually advantageous. This **Baseline** acts as the starting point for further solution development. Refer to the Baseline Management section below to understand how to build a **Baseline**.



Baseline Management

When a project team receives the SDE from Aras, a **Baseline** is established, which acts as the starting point for all future changes to the software. The execution of the setup script (**BuildAndDeploy.ps1**) installs an initial database and sets up all required files in the code tree directory.

The established **Baseline** becomes the foundation for layering changes each time the setup scripts are run.

As these changes accumulate over time, they may grow significantly larger. Therefore, when the solution enters production, a fresh **Baseline** incorporating all customizations can be set. This new **Baseline** will be the starting point for the setup scripts.

After Aras has provided an initial **Baseline**, the project team can make modifications as needed. This may include:

- **Add new applications to the platform:** Adding applications to a platform involves enhancing its functionality and broadening the system's capabilities. The Adding Applications to a Project Appendix demonstrates an example of integrating an application into Aras Innovator.
- **Add Language packs:** Adding language packs to software is crucial in making applications accessible and user-friendly to a diverse global audience.
- **Establish new Baselines:** A new **Baseline** provides a snapshot of the project's status, including what has been achieved and the resources expended to reach this point. Once established, this new **Baseline** serves as the starting point for subsequent phases or steps in the project. For more information, refer to the Generate New Baseline section.
- **Deploy Build to SIT (for QA):** SIT involves testing the system as a whole in an environment that closely mirrors production to ensure that all integrated components work together as expected. This includes ensuring new applications function correctly with the existing system and language packs work as intended. For more details, refer to the Deploy to System Integration Testing (SIT) Environment section.

When an SDE from Aras is received and a **Baseline** is established, it is not a final process. Instead, it is an ongoing effort, and modifications to the software are carried out over time as the project



requirements evolve. These modifications might include integrating new applications, adding new features, fixing bugs, and improving system performance, among other things.

However, this process should be handled appropriately. Software changes must align with the project's goals and should not introduce new issues or conflicts. Therefore, comprehensive testing should be performed after each modification to verify that the changes are working as expected.

Reducing build time is another important aspect of this process. When modifications are organized and managed properly, the time required to build the software can be significantly reduced. This efficiency can lead to quicker deployments and a shorter overall time to market, which can be a significant advantage for the project.

Finally, all these activities must be done as part of a deployment policy. A deployment policy outlines the standards and procedures for making changes to the software, testing those changes, and deploying the software to the production environment. This policy helps ensure that all changes are carried out in a controlled and consistent manner, which can contribute to the overall quality and success of the project.



Pipelines

A **Pipeline** is a set of automated processes that allow developers to consistently and reliably build, test, and deploy their code.

Aras provides the following set of **Pipelines**:

1. **ContinuousIntegration**: This **Pipeline** runs a Build given a **Commit** in the Git **Repo**. The system triggers it in one of the following ways:
 - Validation for a **Branch**.
 - Validation for a **Pull request**.
 - Request by a DevOps user.Typically, the Continuous Integration (CI) **Pipeline** is not manually initiated; it is automatically triggered by actions such as **Pull requests (PRs)** and mergers governed by **Branch** policies.
2. **DeployToSIT**: This **Pipeline** creates a Deployment **Package** from a Build, deploys it to the SIT environment, and stores a copy in the **Artifacts** storage.
3. **DeleteInnovator**: This **Pipeline** deletes a given test instance in the SIT environment.
4. **GenerateBaseline**: Generating **Baselines** establishes a starting point and facilitates tracking project changes. This **Pipeline** generates a new **Baseline** and stores the **Artifact** in the **Artifact** storage.

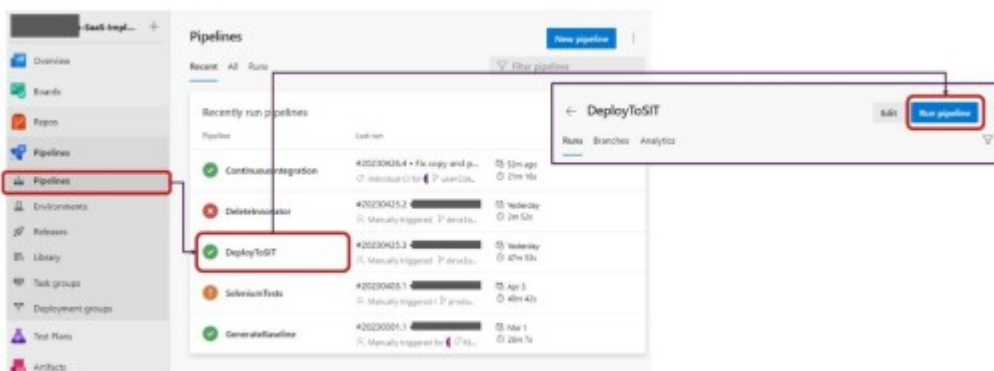


Deploy to System Integration Testing (SIT) Environment

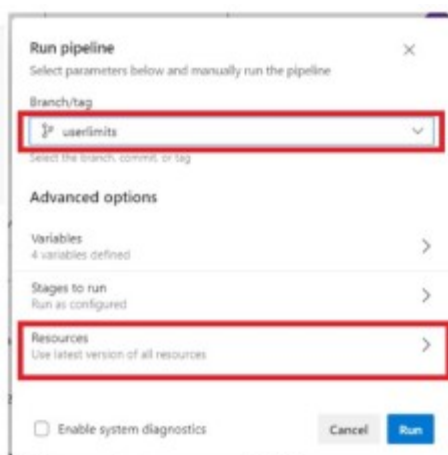
Deploying on SIT allows developers and testers to assess the Aras Innovator's behavior and performance in a simulated production-like setting before it is deployed to the actual production environment.

The following steps outline the process of deploying Aras Innovator to SIT:

1. Navigate to <https://dev.azure.com/{organization}/{project}>.
2. Identify the Build to deploy.
3. Select the **DeployToSITPipeline** and click **Run Pipeline**.

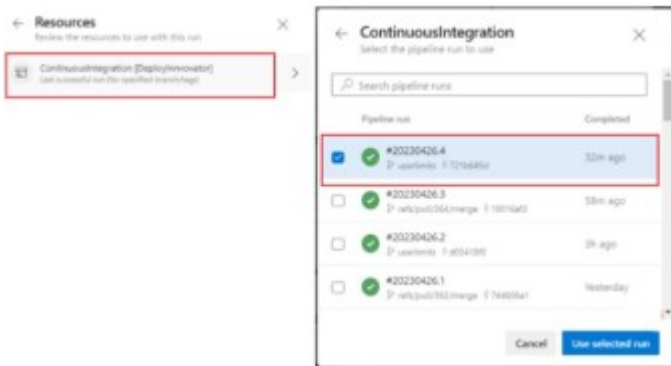


4. Select the **Branch** in the Work Repo and click **Resources**.



5. Click **ContinuousIntegration** (last successful **Run**) and select the required **Run**.





6. Click the **Use selected run** button.
7. Navigate back to **Run Pipeline** dialog box and click **Run**.

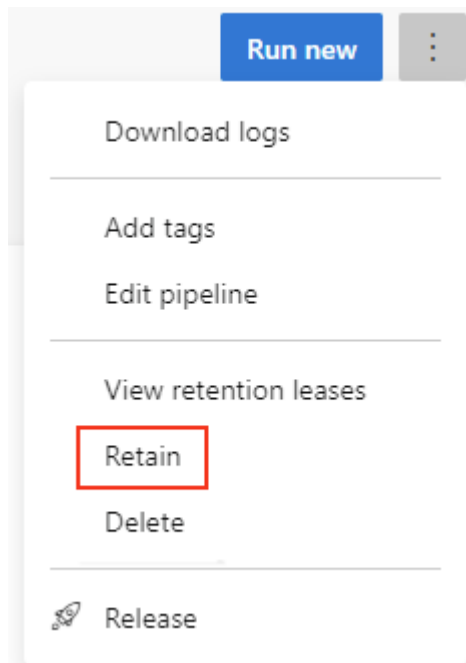


8. Optionally, click the **Deploy** link to watch the progress.
9. When the **Pipeline** is completed, a link to the new instance should be available on the **Wiki** page for testing.

10. **Important**

If a pipeline run is going to be deployed to production, UAT, or staging, you also need to retain it—along with the source CI pipeline run (see the “Pipeline retention policy” section). To do this, open the pipeline run and click **Retain** in the menu.





Release Innovator pipeline

The **Release Innovator pipeline** provides a standardized and automated way to deliver fixes and updates to Aras Innovator SaaS environments while keeping release operations simple, predictable, and governed. The pipeline uses a single, consistent workflow to manage deployments to **SIT**, **UAT**, **STG**, and **Production**, with built-in approvals for controlled promotion.

The following steps outline the high-level deployment process for the Release Innovator:

1. **Define the Release Name:** Identify a unique release identifier used to track the deployment across all environments and distinguish it from other releases.
2. **Identify the initial CI pipeline Build Number:** Determine the initial CI build used for SIT deployment and potential promotion to UAT. CI pipelines for STG and PROD environments are handled separately and require manual tagging.
3. **Trigger the Pipeline:** Initiate the Release Innovator pipeline.
4. **Approve Environment Promotion:** Provide the necessary approvals to advance the update through the SIT, UAT, STG, and PROD environments.
5. **Obtain Production Approval:** Request final approval from Aras before deploying to the Production environment.



Use Case one: Standard release delivery (SIT → UAT → STG → PROD)

This section provides an overview of the basic scenario for the Release Innovator pipeline usage to promote builds across environments in a controlled and repeatable way.

The pipeline enables consistent delivery by using build tags to identify and deploy the same release through different stages, from validation environments to production.

Define the Release Name

Define a unique release identifier used to track the deployment across all environments and distinguish it from other releases. Use any meaningful name (for example, a sprint identifier). This value is used as the “**Release name**” parameter in the Release Innovator pipeline.

Identify the initial CI pipeline Build Number

Determine the initial CI build to be used for SIT deployment and potential promotion to UAT environment.

This CI pipeline Build is the source of customizations for SIT deployment during release initiation. The same build can be used for UAT environment update, or it can be adjusted before approving the stage.

CI builds for STG and PROD are handled separately and require manual tagging.

Trigger the Pipeline

The following steps outline the process of running the Release Innovator pipeline:

1. Navigate to the Release Innovator pipeline.
2. Select **Run Pipeline**.
3. Select the appropriate **Branch/Tag**.



Run pipeline



Select parameters below and manually run the pipeline

Pipeline version

Select pipeline version by branch/tag

Select the pipeline to run by branch, commit, or tag

Parameters

Release name (CI build tag suffix)

Required

Skip "Deploy SIT" stage

Put tags on selected CI build for SIT and UAT environments

Optional variable group name: (Default one is used if none provided)

4. Set **Release name** defined previously.
5. To skip deploying to SIT environment and start with UAT environment update, check **Skip "Deploy SIT" stage** option.
6. To tag identified initial CI pipeline for SIT and UAT check **"Put tags on selected CI build for SIT and UAT environments"** option, otherwise you will need to setup tags manually to promote the changes. It affects only **"Tag CI build"** stage.

Full concept of Release Innovator pipeline is based on environments and pipeline tags.



The appropriate pipeline stage automatically finds the CI build tagged with “**SIT-ReleaseName**”, “**UAT-ReleaseName**”, “**STG-ReleaseName**” or “**PROD-ReleaseName**” and uses its artifact to deploy or update the Aras Innovator.

- In Pipeline artifacts, select the CI pipeline build number you want to deploy to SIT or UAT. If “**Put tags on selected CI build for SIT and UAT environments**” option is not checked, the selected CI pipeline build should be tagged with “**SIT-ReleaseName**” manually in advance.

Pipeline artifacts



- Click the **Run** button.

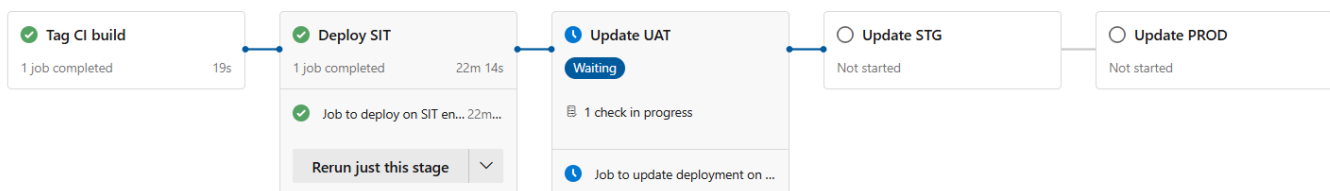
As a result, Release Innovator Pipeline will start. The “**Tag CI build**” and “**Deploy SIT**” stages will pass automatically, if the appropriate options were selected.

Release Innovator pipeline will deploy new instance to SIT environment and deliver updates on top of existing instance in case of UAT, STG and PROD environments.

Environment Promotion Approval

When the pipeline reaches Update UAT stage, it will wait for approval.

Stages Jobs



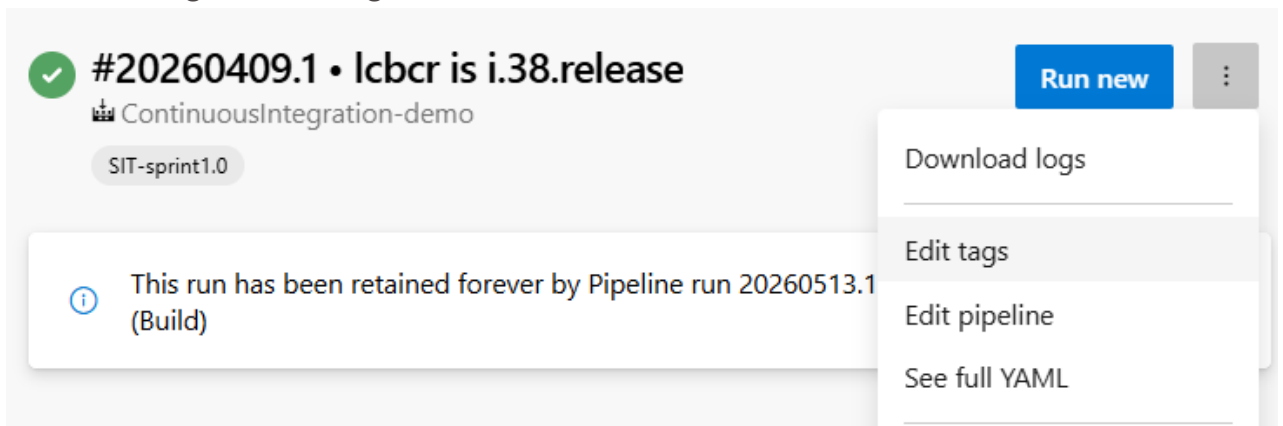
- Before approving Update UAT stage please make sure that correct CI pipeline is tagged. If you want to deploy to UAT from specific UAT branch, make sure that CI pipeline from UAT branch has “**UAT-ReleaseName**” tag.
To review or update tags follow the steps:



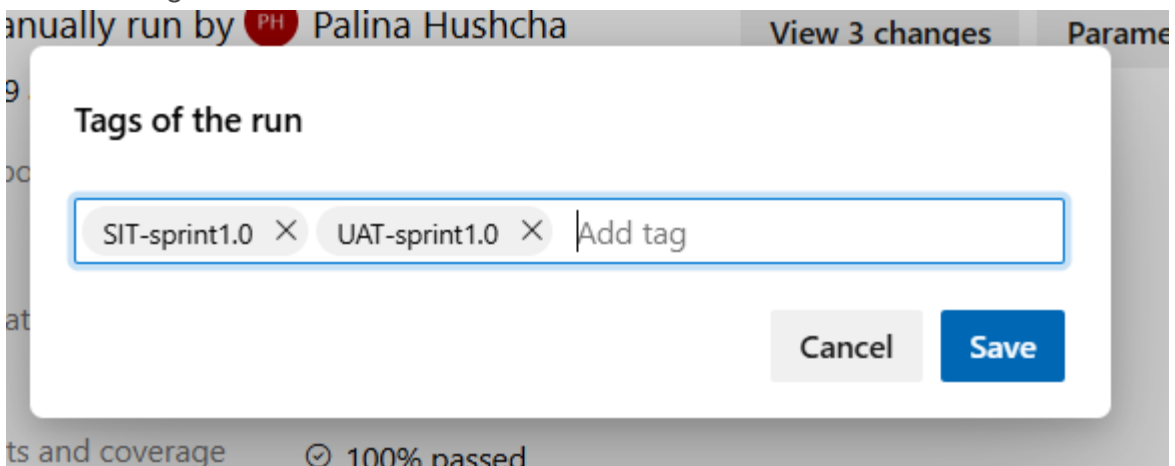
- Open the necessary CI **Pipeline**
- Review tags



- Click **Add tags** or **Edit tags**



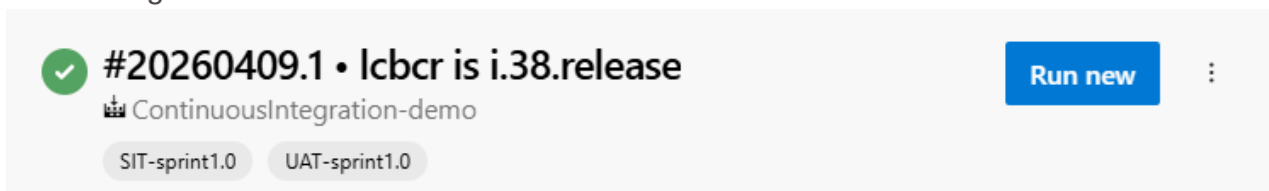
- Add new tag



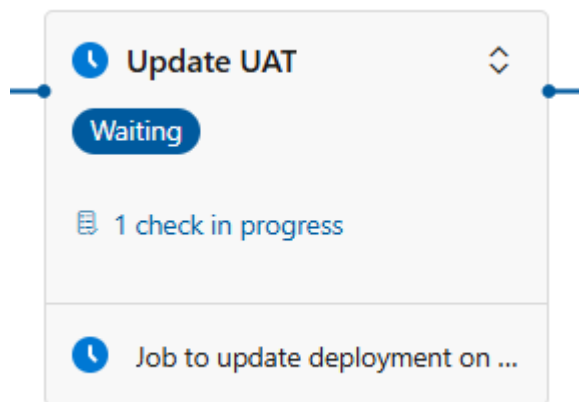
- Click **Save** button



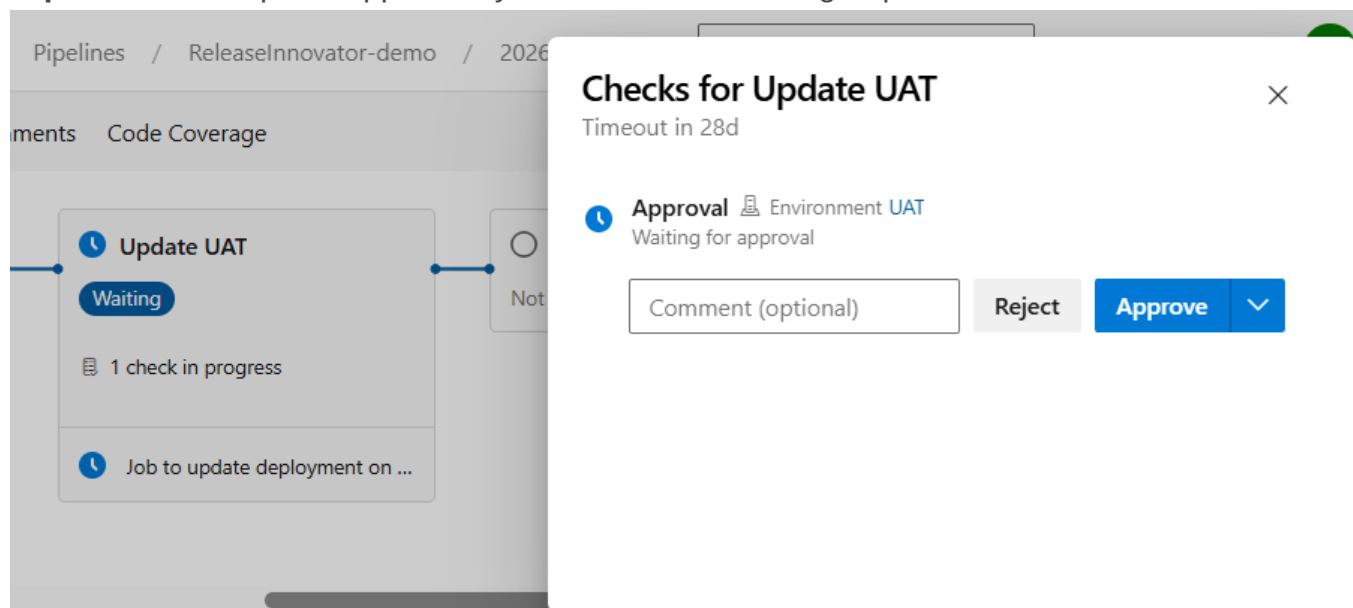
- o Review tags



2. If you are running Release Innovator Pipeline for the first time, make sure that the innovator release name is specified in **ReleasePipelineConfigValues** group.
3. If CI pipeline is tagged correctly, click on “**1 check in progress**”.



4. Approve the stage.
“Update UAT” requires approval by “Business Owners” group.



Release Innovator pipeline will trigger the UAT update. You can find link to UAT pipeline inside **Queue DeployInnovator on (UAT) and wait for completion** step.

The process of updating STG environment is similar to UAT update. Before running Update STG, ensure the corresponding “**STG-ReleaseName**“ tag exists on the intended CI build (this is not applied automatically by the pipeline). “**Update STG**” requires approval by “**Business Owners**” group.

Obtain Production Approval

The process of updating PROD environment is similar to UAT and STG update. Before running “**Update PROD**”, ensure the corresponding “**PROD-ReleaseName**“ tag exists on the intended CI build (this is not applied automatically by the pipeline).

“**Update PROD**” stage requires approvals by “**Business Owners**” and “**GCS Operators**” groups.

After approval, the customization is promoted to production with full traceability through release naming, tags, and approval history.



Use Case two: Fast release delivery (UAT → PROD)

This section describes the fast release delivery scenario for the Release Innovator pipeline, used when a customization must be promoted to UAT and PROD with minimal delay skipping Deploy SIT and Update STG stages while preserving approval controls and release traceability.

The pipeline enables consistent delivery by using build tags to identify and deploy the same release through different stages, from validation environments to production.

Define the Release Name

Define a unique release identifier used to track the deployment across all environments and distinguish it from other releases. Use any meaningful name (for example, a sprint identifier). This value is used as the “**Release name**” parameter in the Release Innovator pipeline.

Identify the initial CI pipeline Build Number

Determine the initial CI build to be used for UAT update.

This CI pipeline Build is the source of customizations for UAT deployment during release initiation.

CI build for PROD is handled separately and require manual tagging.

Trigger the Pipeline

The following steps outline the process of running the Release Innovator pipeline for a fast release delivery:

1. Navigate to the Release Innovator pipeline.
2. Select **Run Pipeline**.
3. Select the appropriate **Branch/Tag**.



Run pipeline



Select parameters below and manually run the pipeline

Pipeline version

Select pipeline version by branch/tag

Select the pipeline to run by branch, commit, or tag

Parameters

Release name (CI build tag suffix)

Required

Skip "Deploy SIT" stage

Put tags on selected CI build for SIT and UAT environments

Optional variable group name: (Default one is used if none provided)

4. Set **Release name** defined previously.
5. Select the **Skip "Deploy SIT" stage** option to begin the process from UAT.
6. To tag identified initial CI pipeline for UAT check "**Put tags on selected CI build for SIT and UAT environments**" option, otherwise you will need to setup tags manually to promote the changes. It affects only "**Tag CI build**" stage.

Full concept of Release Innovator pipeline is based on environments and pipeline tags.



The appropriate pipeline stage automatically finds the CI build tagged with "**UAT-ReleaseName**" or "**PROD-ReleaseName**" and uses its artifact to deploy or update the Aras Innovator.

- In Pipeline artifacts, select the CI pipeline build number you want to deploy to UAT.

Pipeline artifacts



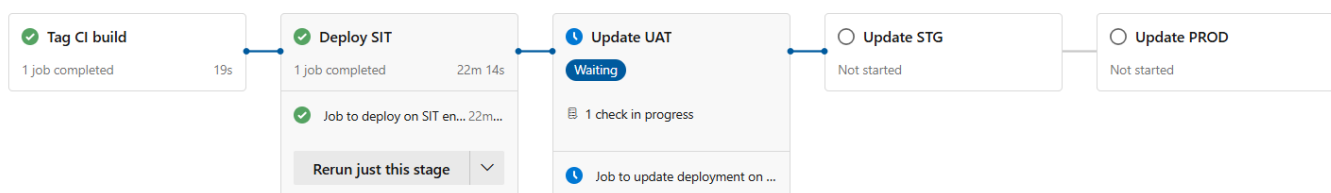
- Click the **Run** button.

As a result, the Release Innovator pipeline will start, pass "**Tag CI build**" stage and skip the Deploy SIT stage. The selected CI build is prepared for UAT update and tracked under the defined release identifier.

Environment Promotion Approval

When the pipeline reaches Update UAT stage, it will wait for approval.

Stages Jobs

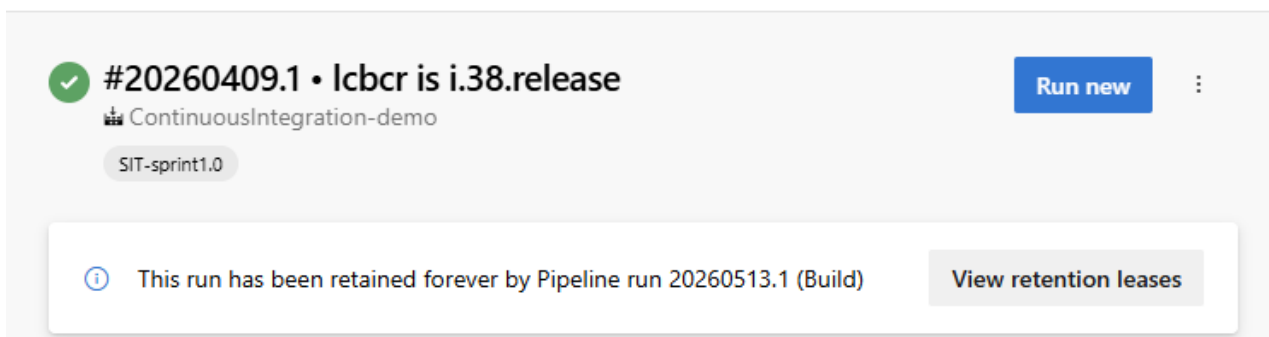


- Before approving Update UAT stage please make sure that correct CI pipeline is tagged. If you want to deploy to UAT from specific UAT branch, make sure that CI pipeline from UAT branch has "**UAT-ReleaseName**" tag.

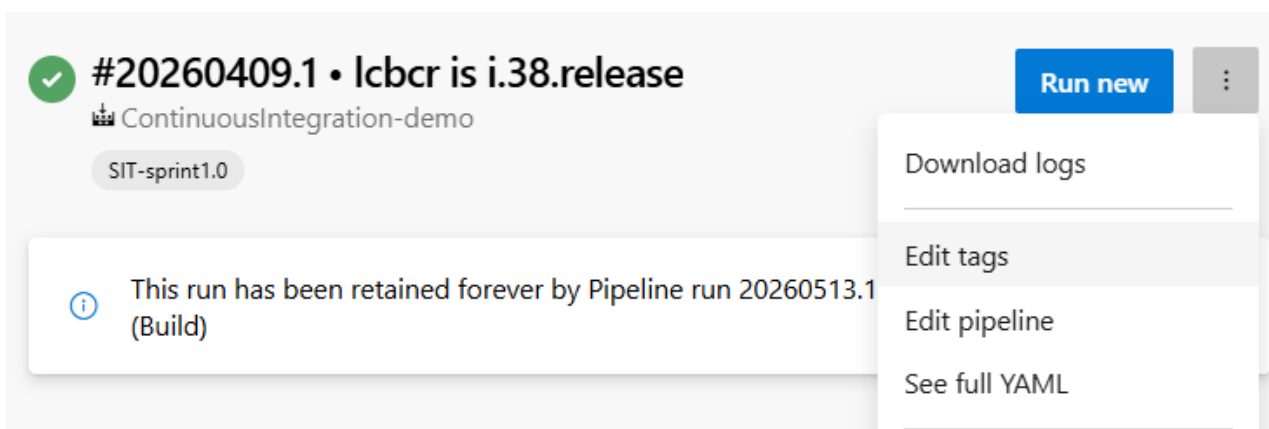
To review or update tags follow the steps:



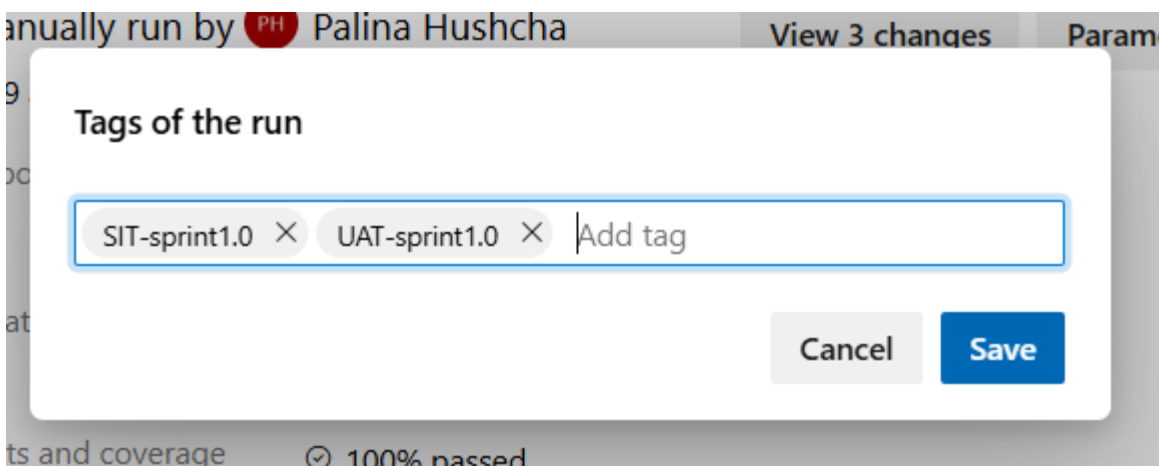
- Open the necessary CI **Pipeline**
- Review tags



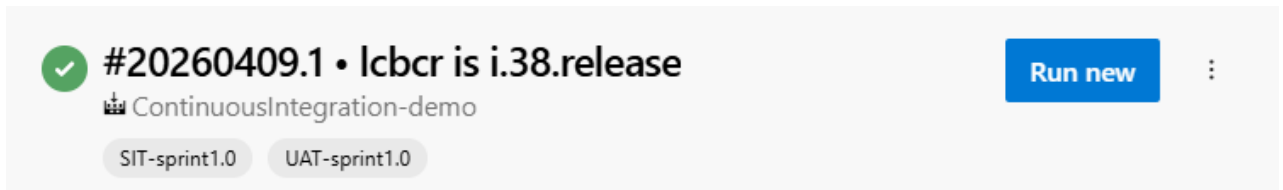
- Click **Add tags** or **Edit tags**



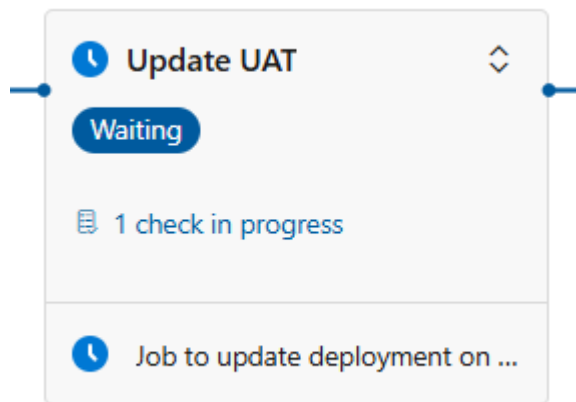
- Add new tag



- Click **Save** button
- Review tags

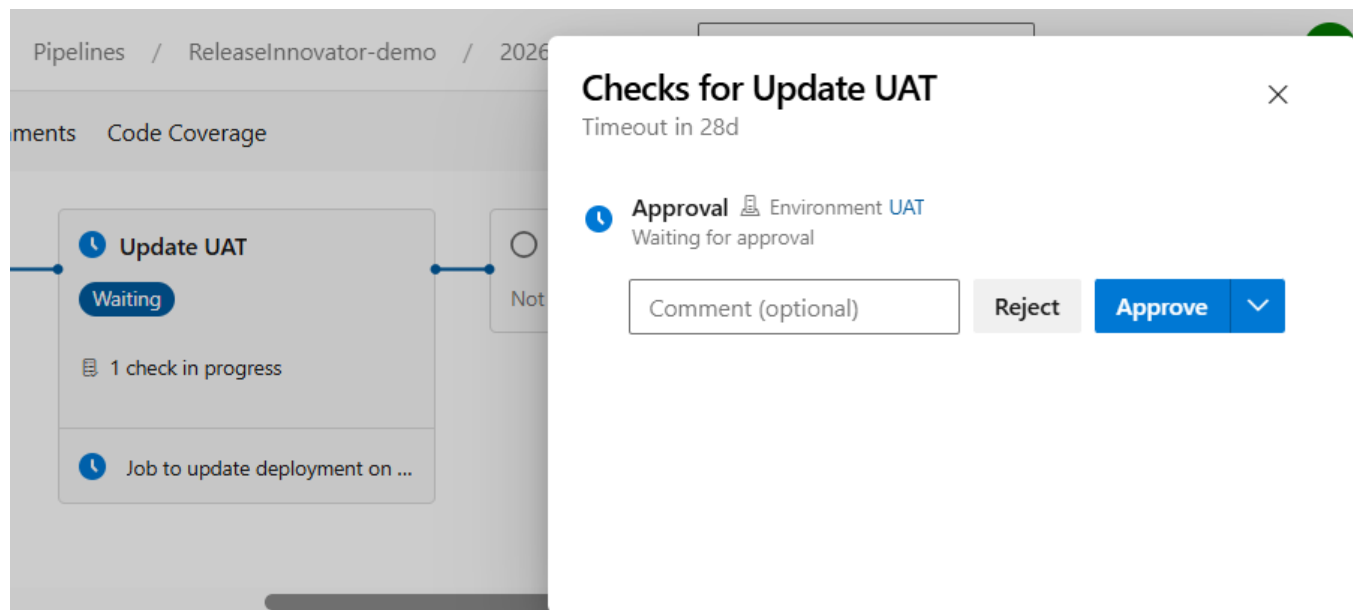


2. If you are running Release Innovator Pipeline for the first time, make sure that the innovator release name is specified in **ReleasePipelineConfigValues** group.
3. If CI pipeline is tagged correctly, click on “**1 check in progress**”.



4. Approve the stage.

“Update UAT” requires approval by “Business Owners” group.



Release Innovator pipeline will trigger the UAT update. You can find link to UAT pipeline inside **Queue DeployInnovator on (UAT) and wait for completion** step.

5. To skip **Update STG** stage make sure that the innovator release name for STG instance is not specified in **ReleasePipelineConfigValues** group.

Obtain Production Approval

The process of updating PROD environment is similar to UAT update. Before running Update PROD, ensure the corresponding “**PROD-ReleaseName**” tag exists on the intended CI build (this is not applied automatically by the pipeline).

“**Update PROD**” stage requires approvals by “**Business Owners**” and “**GCS Operators**” groups.

After approval, the customization is promoted to production with full traceability through release naming, tags, and approval history.



Generate New Baseline

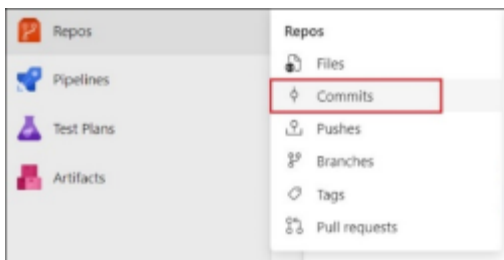
Creating Tag on Last Approved Commit

When the Aras Innovator is installed for the first time, a Git **Tag** marks the starting point. It uses the following format: CleanInnovatorxxSPyy, where xx is the version and yy is the Service Pack number of the base platform.

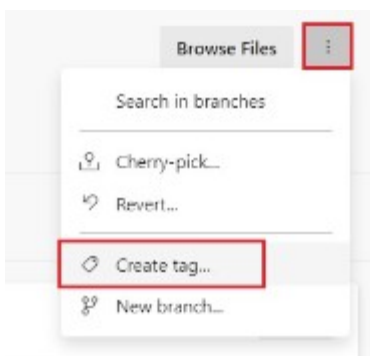
Define the Git **Tag** on the last approved **Commit**. The **Pull request (PR)** process must be completed to get the tested **Commit** that is proposed as the new **Baseline** to the Destination (Central **Repo**) **Branch**. The **Commit** must be tagged.

The following steps outline the process of creating a **Tag** on the last approved **Commit**:

1. Navigate to <https://dev.azure.com/{organization}/{project}>.
2. Click **Repos** and select **Commits**. Select the corresponding successful **Commit**.



3. Click the **More options** menu and select **Create tag ...**



4. Enter the following details in the **Create a tag** dialog:
 - o **Name:** Name of the **Tag**.
 - o **Based on:** **Commit** above.

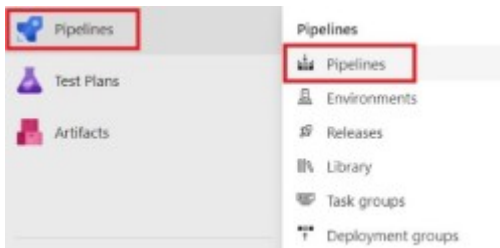
- **Description: Tag** Description.
 - Select an appropriate **Baseline** naming convention. For most projects without features, using **prj** (Project), **bl** (Baseline), and numbers is sufficient. For example, **prjbl001** for a user's first **Baseline**.
 - After each product release, it is also recommended that the user must have a **prdbl001** Production **Baseline**.
 - Some projects may have several workstreams (features) with different go-live dates. For such projects, the team may include feature designations, such as **mbsebl01**: **mbse** (Model-Based Systems Engineering), **bl** (Baseline), and **01** (chosen identifying number).
5. Click the **Create** button.



Running the Baseline Pipeline

The following steps outline the process of running the **Baseline Pipeline**:

1. Navigate to <https://dev.azure.com/{organization}/{project}>.
2. Click **Pipelines** in the left menu and select **Pipelines**.



3. Click **GenerateBaselinePipeline**.

Pipeline	Last run
SeleniumTests	#20230623.1 • added SDE09 with new item and corresponding artifacts ↳ Manually triggered for development Jun 23 52m 31s
ContinuousIntegration	#20230623.9 • added SDE09 with new item and corresponding artifacts ↳ Scheduled for development Jun 23 22m 35s
DeployToSIT	#20230623.1 • added SDE09 with new item and corresponding artifacts ↳ Manually triggered for attn00 Jun 23 32m 7s
GenerateBaseline	#20230623.2 • Changed method as a result of review process ↳ Manually triggered for attn00 Jun 23 21m 30s
DeleteInnovator	#20230504.1 • fix floating issue with build agents ↳ Manually triggered for development May 4 22m 55s

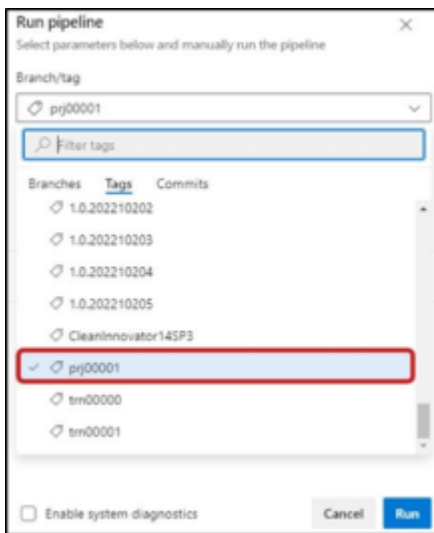
4. Click the **Run Pipeline** button in the top-right corner.

Description	Stages
#20230623.2 • Changed method as a result of review process ↳ Manually triggered for attn00 31bee7a	
#20230623.1 • added SDE09 with new item and corresponding arti... ↳ Manually triggered for attn00 3f2f396	
#20230504.1 • fix floating issue with build agents ↳ Manually triggered for 1,0,20230504.1 8770902b	

5. In the **Run Pipeline** dialog, enter the following details:
 - **Branch/Tag**: Select the **Branch/Tag** created in the above section.



- **Advance options:** Leave as default settings.
- **Enable system diagnostics:** Leave unchecked.



Important

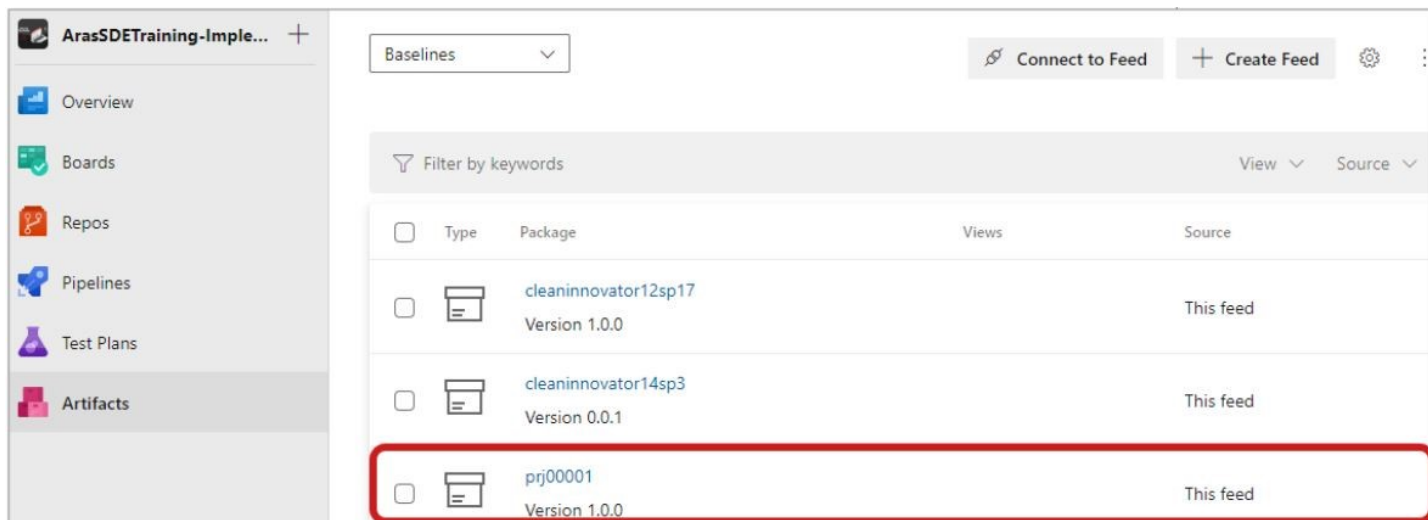
The **Pipeline** must be executed by selecting **Tags** only.

Important

If SDE is referencing the Aras.SaaS.DevOpsFramework.AICD package, it is possible to specify the following parameters:

- **Pipeline timeout in minutes:** Defines the maximum duration the baseline generation process is allowed to run before it is automatically stopped.
- **Optional variable group name:** Overrides the default variable group if a custom name is provided; otherwise, the default group is used.

6. Click the **Run** button.
7. The system queues the **Pipeline**, and progress can be monitored by clicking on the **Stages and Jobs** Tabs on the **Pipeline Run** page.
8. The new **Baseline** will be uploaded to the storage account and **BaselinesArtifact** feed (**Baselines** drop-down menu).



Once the pipeline finishes successfully, the new Baseline artifact is uploaded to the **SDE Baselines feed**.

The updated baseline is then automatically used by CI/CD pipelines and downloaded locally by `BuildAndDeploy.ps1` once `Last.Commit.Before.Current.Release` value in the `Default.Settings.include` file is updated to reference the appropriate tag.



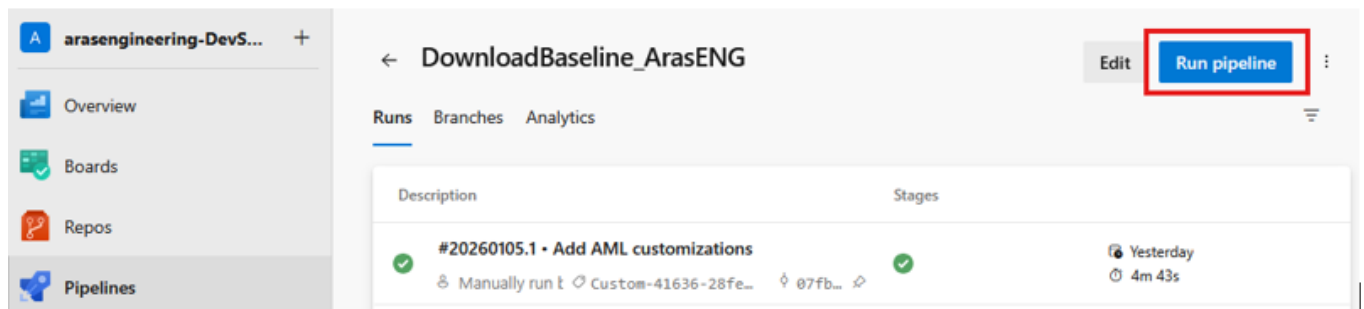
Download Baseline Pipeline

This chapter is specific to SDE, that is referencing **Aras.DevOpsFramework.SaaS3** package, and enables you to establish a new baseline for Local development environment.

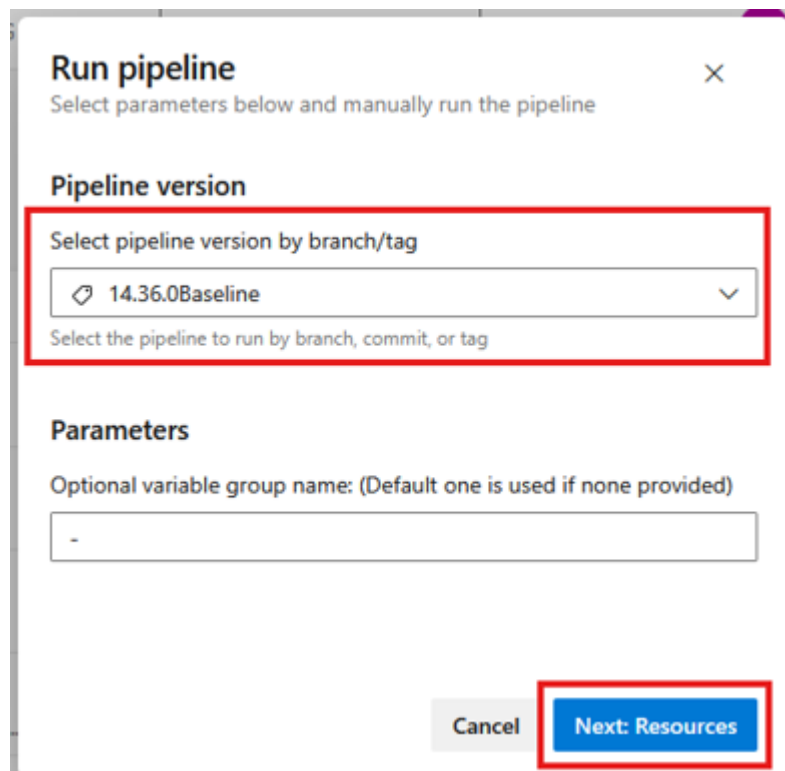
The baseline must be downloaded using the **Download Baseline** pipeline, which provides the CI-generated baseline. The downloaded **.bacpac** file should be used for your local development environment.

Follow these steps to download a baseline using the **Download Baseline** pipeline:

1. **Run the pipeline.** In your Azure DevOps project, go to **Pipelines**, locate the **Download Baseline** pipeline, and click **Run pipeline**.



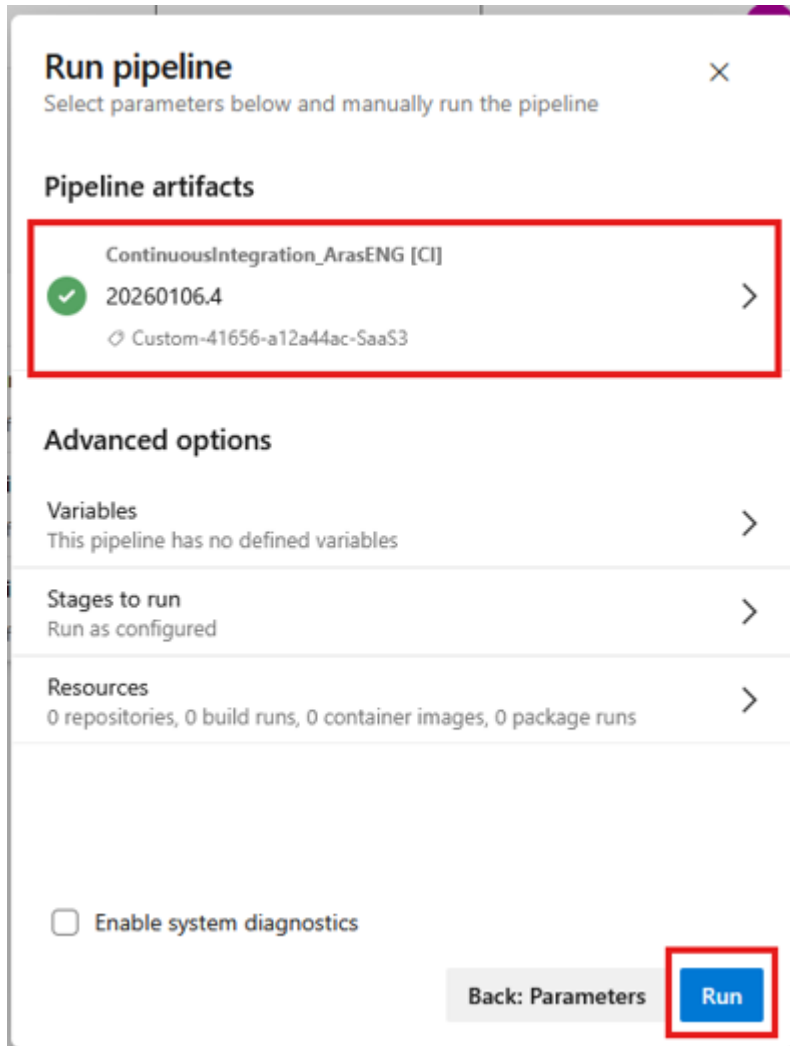
2. Select the **Branch** in the Work **Repo** and specify **Optional variable group name** if necessary, then click **Next: Resources**.



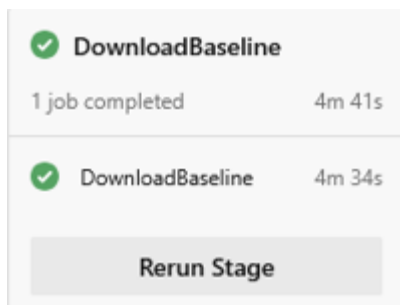
The image shows a 'Run pipeline' dialog box with a close button (X) in the top right corner. Below the title, it says 'Select parameters below and manually run the pipeline'. The 'Pipeline version' section is highlighted with a red box and contains a dropdown menu with '14.36.0Baseline' selected. Below the dropdown is the text 'Select the pipeline to run by branch, commit, or tag'. The 'Parameters' section has a text input field with a hyphen '-' inside. At the bottom, there are two buttons: 'Cancel' and 'Next: Resources', with the latter highlighted by a red box.

3. **Select the CI run.** Choose the CI pipeline run, which provides the required generated baseline, then click **Run**.





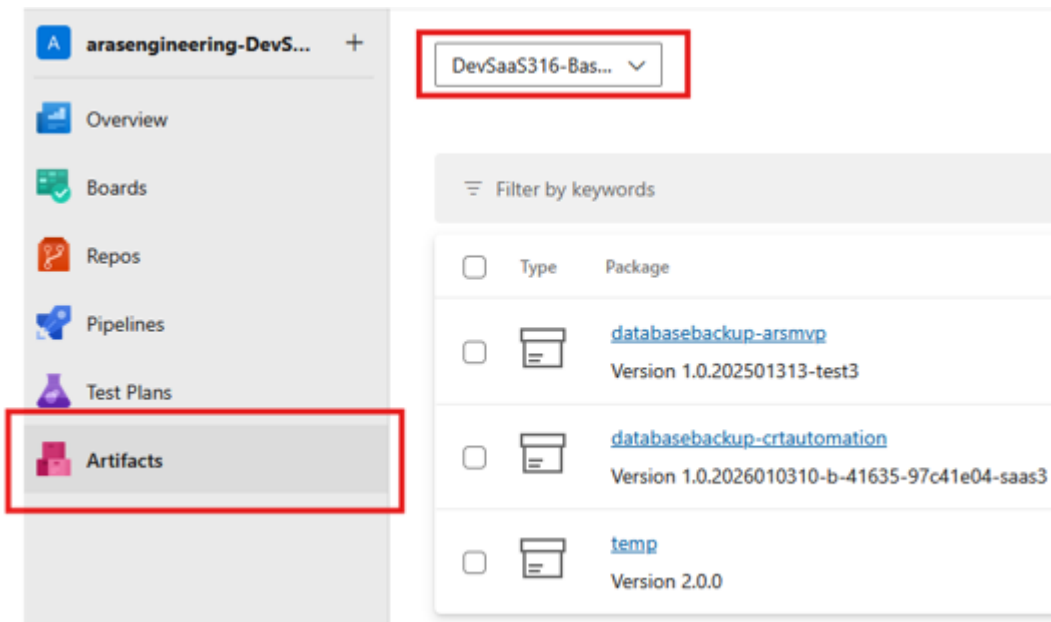
4. **Wait for completion.** Wait until the pipeline finishes successfully (status is green).



5. Open **Publish DB backup** step of green **Download Baseline** pipeline and note down the package name and version.

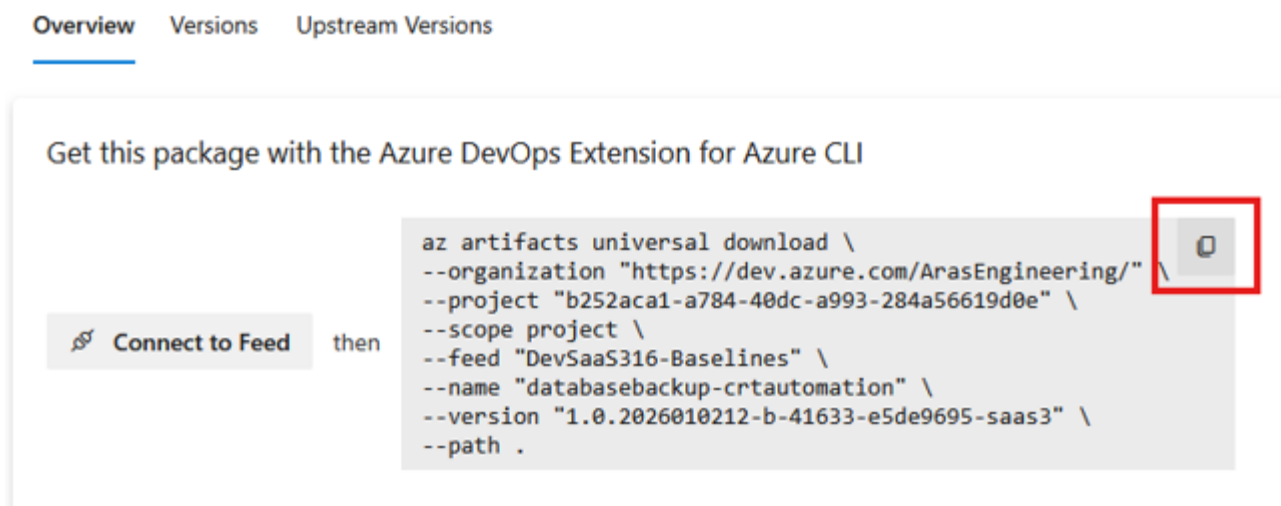


6. **Open the Baselines feed.** Go to **Artifacts** in your Azure DevOps project and open the **Baselines Feed**.



7. **Select the required version.** Open **Versions** and click the version you want to download.

8. **Copy the download command.** Copy the PowerShell command provided for downloading the artifact.



9. **Download the baseline locally.** Open **PowerShell** on your local machine, paste the copied command, and press **Enter** to start the download.

If everything is successful, you will see output similar to the following, and the file **DatabaseBackup.bacpac** will be downloaded to the current PowerShell directory.

```
PS C:\> az artifacts universal download --organization "https://dev.azure.com/ArasEngineering/" --project "b252acal-a784-40dc-a993-284a56619d0e" --scope project --feed "DevSaaS316-Baselines" --name "databasebackup-crtautomation" --version "1.0.2026010212-b-41633-e5de9695-saas3" --path .
C:\Users\sazarau\.azure\cliextensions\azure-devops\azext_devops\dev\__init__.py:5: UserWarning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html. The pkg_resources package is slated for removal as early as 2025-11-30. Refrain from using this package or pin to Setuptools<81.
import pkg_resources
{- Downloading ..
  "Description": "DB backup",
  "ManifestId": "794A63199342803CC65036137BC0AB91650DE5D96D813859F7A0DAAC9126034E01",
  "PackageSize": 4932205,
  "SuperRootId": "D32D4F3397CE6DC253103F5469CC145F284D838F61453543FB09A09D182C804E02",
  "Version": "1.0.2026010212-b-41633-e5de9695-saas3"
}
```

Important

If you encounter an authentication error, run a login and follow the prompts. After that, re-run the download command.

```
Failed to update Universal Packages tooling.
Before you can run Azure DevOps commands, you need to run the login command(az login if using AAD/MSA identity else az devops login if using PAT token) to setup credentials. Please see https://aka.ms/azure-devops-cli-auth for more information.
```

10. Make sure that last commit before current release is specified in **Default.Settings.include** and corresponds to downloaded baseline.

```
Default.Settings.include
245 <!--
246 Last.Commit.Before.Current.Release - ID of last commit before current release. This is commit which is used as a baseline to
247 figure out what files were changed during current release. These files define code tree and AML packages changes, that should
248 be deployed to target server.
249
250 Comments for the default value:
251 CleanInnovator12SP12 - a tag in a version control system that points to the 'baseline' commit
252 -->
253 <property name="Last.Commit.Before.Current.Release" overwrite="false" value="b-41582-35419bf1-SaaS3" />
```

11. Make sure that code tree.zip of clean released Aras Innovator (without customizations) is added to **Machine.Settings**. include

Example:



```
<property name="Path.To.CodeTree.Zip" overwrite="true" value="C:\path\to\CodeTree.zip" />
```

12. Make sure that Path.To.DB.Bacpac environment variable is set to the downloaded bacpac filepath.

Example of PowerShell command to setPath.To.DB.Bacpac:

```
[Environment]::SetEnvironmentVariable('Path.To.DB.Bacpac', 'C:\path\to\DatabaseBackup.bacpac')
```



Delete Aras Innovator from SIT Environment

The SIT environment provides resources for a maximum of ten Aras Innovator test instances. Aras recommends keeping the number of test instances at about three to five. The project team should retain the latest test instances while removing older ones to prevent potential performance degradation. The **DeleteInnovator Pipeline** deletes test instances by providing the instance name (lowercase) and identifying the Build.

Important

It is possible to delete also from UAT, STG, and PROD environments. Deleting from other environments follows the same user experience. Pipeline names may vary depending on the single package in use.

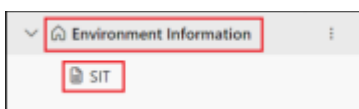
After approximately thirty days, the system will automatically delete a Build.

The following steps outline the process of deleting the Aras Innovator from SIT Environment:

1. Navigate to <https://dev.azure.com/{organization}/{project}>.
2. Click **Overview** and select **Wiki**.



3. Expand **Environment Information** and select **SIT**.



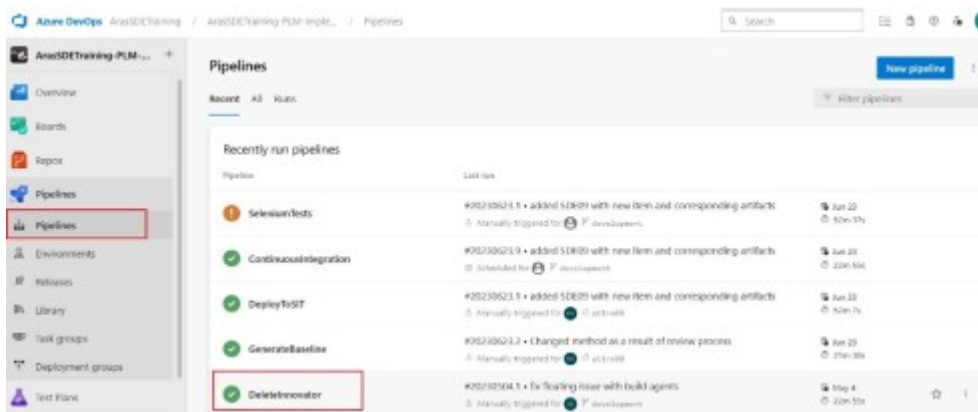
The list of available instances appears.

SIT

Witahh Vitahh Monday

Name	Creation Date
SIT-1-0-202207054	05-Jul-2022 11:43
SIT-1-0-202207062	06-Jul-2022 10:13
SIT-1-0-202207064	06-Jul-2022 01:18
SIT-1-0-202207081	08-Jul-2022 08:44
SIT-1-0-202207112	11-Jul-2022 11:13
SIT-1-0-202207116	11-Jul-2022 03:34

- Copy the required instance.
- Navigate to **Pipelines** and select the **DeleteInnovatorPipeline**.



- Click the **Run Pipeline** button.
- In the Aras Innovator instance **Name to Delete** field, paste the SIT instance which was copied in step 4.
- Select **Resources** and click **ContinuousIntegration [Delete Innovator]**.

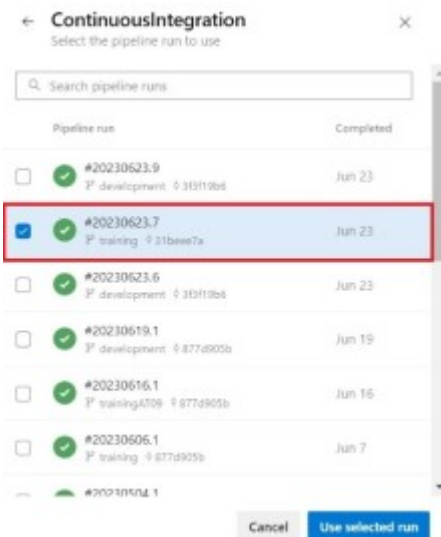


- Select the correct Build. The Build Name ends with a timestamp `yyyymmdd##`. In the corresponding Build, a dot separates the Build Number from the Date.

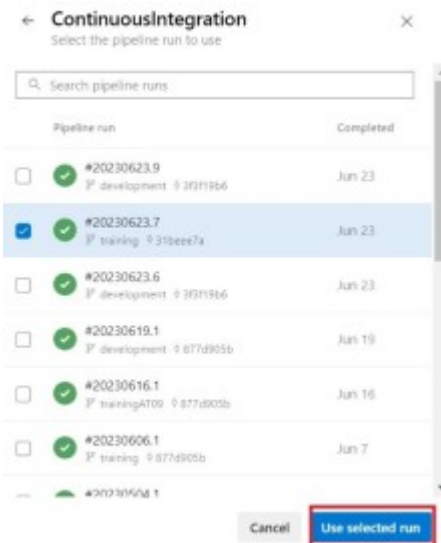
Important



The timestamp of SIT instance should match the Build's timestamp.



10. Click the **Use selected run** button.



11. Select **Variables** and enter the value in the **Update Variable** dialog box. Set the Variable in lowercase "sit" and click **Update**.



← Update variable ×

Name Required

Value

Cancel Update

12. Navigate back to **Run Pipeline** pane and click the **Run** button.

Run pipeline ×

Select parameters below and manually run the pipeline

Branch/tag

Select the branch, commit, or tag

Innovator instance name to delete Required

Advanced options

Variables
This pipeline has no defined variables >

Stages to run
Run as configured >

Resources
0 repositories, 1 pipeline run, 0 build runs, 0 container images, 0 packag... >

Enable system diagnostics

Cancel Run

13. If the Build has expired, select the oldest green Build.



Delta deployment to production and non-production environments

Delta Deployment functionality is an Aras DevOps feature for Aras Enterprise customers working with containerized deployments.

Delta deployment allows deploying changes from the Work Repository on top of an existing Aras Innovator deployment. The database of the target Aras Innovator deployment will not be dropped and restored. The customization changes will be deployed on top of the existing database.

Users have the capability to utilize this feature to update the existing instance in SIT (System Integration Testing), UAT (User Acceptance Testing), Staging, or Production environments.

The following steps outline the high-level process of delta deployment:

1. Determine the Aras Innovator Release Name the user wants to update.
2. Determine the build number of the **CI pipeline** run that includes the customization user wants to deploy to the previously deployed Aras Innovator.

Important

The CI pipeline run **must be retained** manually ahead of time (see the “Pipeline retention policy” section).

3. Run the **Deploy Pipeline** with Update strategy.



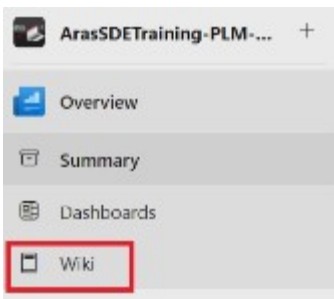
Redeploy UAT deployment on Existing Database

The following sections outline redeploying UAT Deployments on top of the existing database.

Determine the Innovator Deployment Name

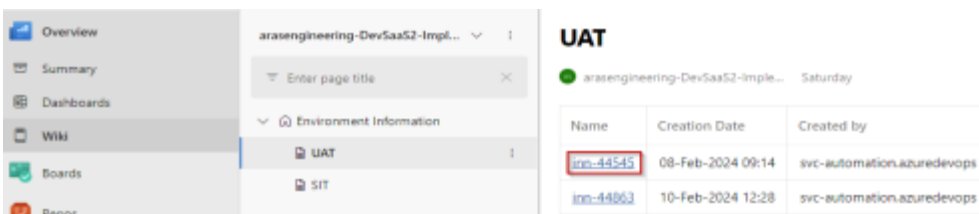
The following steps outline the process of determining the Aras Innovator Release Name for the deployed Aras Innovator Instance that the user wants to update:

1. On Azure DevOps, click **Wiki**.



2. Under the **Environment Information** section, select **UAT**.

Copy the release name under the **Name** Tab in **UAT** section.



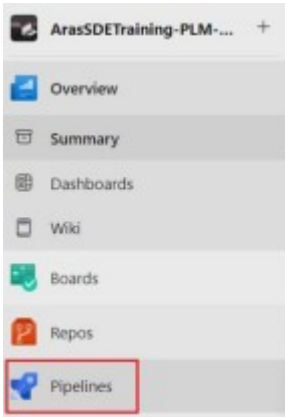
Copy the name; it will be useful during Deploy **Pipeline**.

Determine Deployment Package

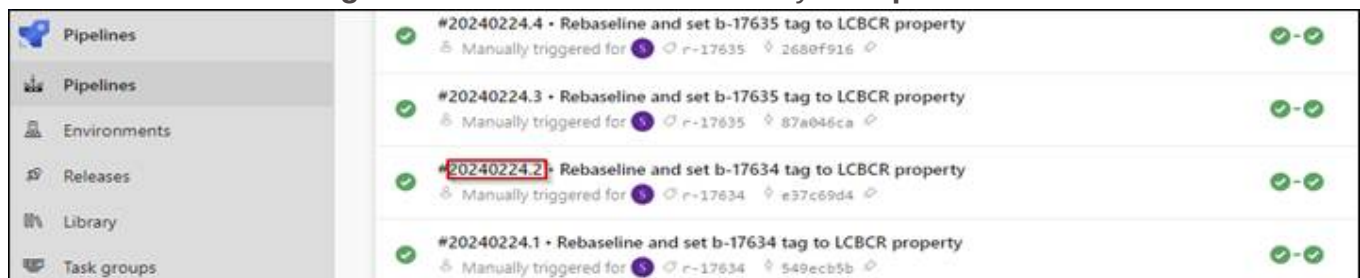
Determine the Deployment **Package** Version with the customization the user wants to deploy to the previously deployed Aras Innovator. This can be done via obtaining the Build Number of the CI **Pipeline** corresponding to the version of the customization the user wants to deploy:

1. In Azure DevOps, go to **Pipelines**.





2. Select **ContinuousIntegration** and select the necessary CI **Pipeline**.



3. Copy the Build Number; it will be useful during Deploy **Pipeline**.

Run Deploy Pipeline with Update strategy

The following steps outline the process of running the Deploy **Pipeline** with Update Strategy:

1. Navigate to the **Deploy Innovator Pipeline**.
2. Select **Run Pipeline**.
3. Select the appropriate **Branch/Tag**.
4. Choose **Deployment type**.



Run pipeline ✕

Select parameters below and manually run the pipeline

Branch/tag
 ▼
Select the branch, commit, or tag

Deployment type:
 ▼

Deploy timeout in minutes

Optional variable group name: (Default one is used if none provided)

Custom config variable group name: (Default one is used if none provided)

Advanced options

Variables
 8 variables defined >

Stages to run
 Run as configured >

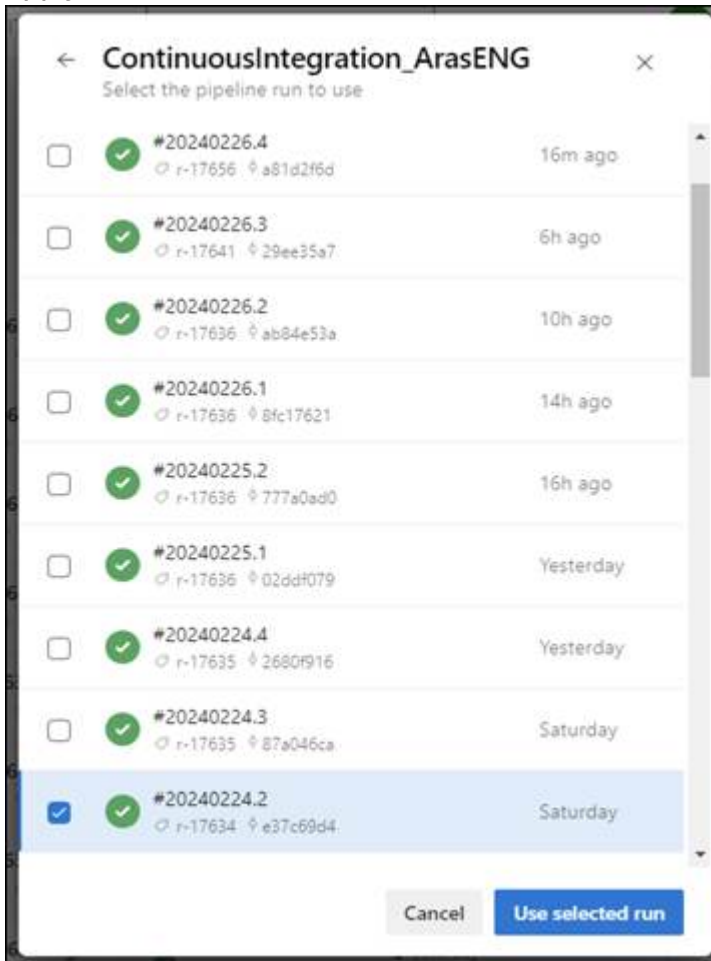
Resources
 Use latest version of all resources >

Enable system diagnostics

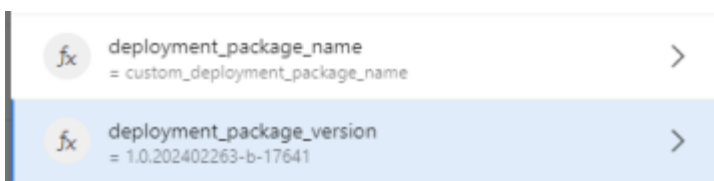
5. To choose the required Deployment **Package**, use one of the options below:
 - Click Resources in **Advanced** options. Search for the correct CI **Pipeline** with the code changes that need to be applied to Aras Innovator, which is obtained in the section **Determine Deployment Package**, and then click the **Use selected to run** button to



apply.



- o Click **Variables** in **Advanced Options** and specify **deployment_package_name** and **deployment_package_version** with the values obtained in the section **Determine Deployment Package**.



6. Click **Variables** in **Advanced options**.



Run pipeline ✕

Select parameters below and manually run the pipeline

Branch/tag

Select the branch, commit, or tag

Advanced options

- Variables**
This pipeline has no defined variables >
- Stages to run
Run as configured >
- Resources
Use latest version of all resources >

7. Add the following Variables:

Name	Value
personal_access_token	Enter the PAT token in case of deployment to UAT, Staging, or PROD.
innovator_release_name	Enter the Release Name of the currently deployed Aras Innovator that needs to be updated.
setup_instance_strategy	Enter Update.
cluster_index	Enter a cluster identification value for deployment. This is especially helpful when deploying multiple Vaults across different geographical clusters. Typically, 01 suffices as the default value.
install_conversion_server	The default value is true.

Variables ✕

ArsEmptyVarFix-DI

- cluster_index = 01 >
- deployment_package_name = custom_deployment_package_name >
- deployment_package_version = 1.0.202402263-b-17641 >
- innovator_release_name = inn-44345 >
- install_conversion_server = true >
- personal_access_token = ***** >
- setup_instance_strategy = Update >

[Add variable](#)



The **setup_instance_strategy** can have the following values:

- **Deploy**: Enables the creation of a new Deployment. This is the default value of this Variable for Deployments to SIT environment.
- **Update**: Enables updating an existing Deployment. This is the default value of this Variable for Deployments to UAT, Staging, and PROD environments.
- **RollingUpdate**: This update type should be used when only Code Tree changes should be applied. The Pipeline will fail if there are any changes in AML Packages.

8. Click the **Run** button.

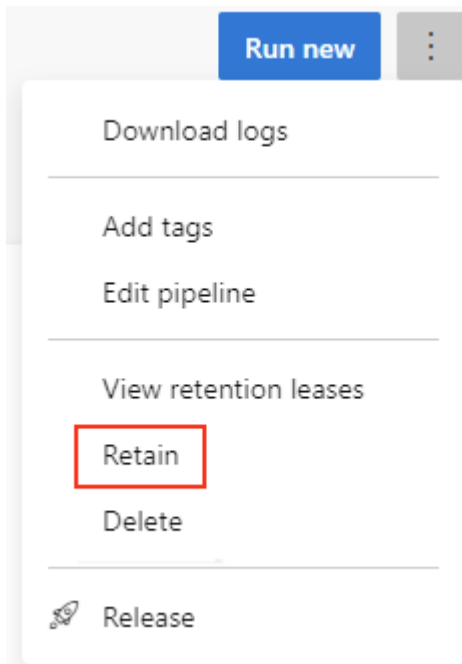
The outcome of this operation will deploy the new changes onto the existing database and Aras Innovator instance, matching the **innovator_release_name** with **setup_instance_strategy** set to **Update**.



Pipeline Retention Policy

The pipeline retention policy defines how long Azure DevOps keeps pipeline files before deleting them. To fully recreate a deployed Aras Innovator instance, the user needs to retain both the CI and DeployInnovator pipeline runs. By default, Azure Pipelines keeps only the **three most recent** runs for the development, SIT, UAT, staging, and production branches.

Any runs beyond these defaults must be manually marked for retention by the user:



Using Transformations

[Transformation Overview](#)

A Transformation is a mechanism for updating configuration files such as Aras Innovator's XML or JSON files using a special syntax. Since all config files are XML or JSON, the XDT or JDT transformation is used, respectively.

This procedure is intended to be idempotent, implying that repeated transformation application to a specific configuration file (like those of Aras Innovator) should consistently result in the same state achieved immediately after the initial transformation application.

Idempotence ensures that regardless of the number of times a transformation is applied to a specific configuration file, the outcome remains consistent and predictable, eliminating the need to manage delta changes.



Type of Transformation

There are two following types of transformation:

1. **XMI Document Transformation (XDT):** Enables transforming XML files.
2. **JSON Document Transforms (JDT):** Enables transforming JSON files.



The Purpose of Transformation

Only the specific configuration files are updated rather than a complete overwrite of the content.

This approach facilitates modifications only in the required sections of the configuration, ensuring that all other settings remain unaffected.

The developer is responsible for creating such a Transformation, which might be applied to the config file often. It will give the same result as after the first application of the Transformation.

The standard Aras Innovator platform Deployment only contains the information below in the conversion server configuration file.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
<configSections>
<section name="oauth" type="Aras.OAuth.Configuration.OAuthSection, Aras.OAuth.Configuration" />
</configSections>
<oauth configSource="OAuth.config" />
</configuration>
```

The project team must provide information in the Transformation file to activate conversion according to the project's requirements and entitlements.



Utilizing Transformation

If any update is required in the configuration file, a file with an identical name should be created within the TransformationsOfConfigFiles directory using the relative path of the file.

Idempotence isn't provided out-of-the-box; utilization of specific Transformation actions is required.

In the XDT framework, these operations are signified by the suffix IfMissing (such as InsertIfMissing). Conversely, in JDT, the **Merge** action is the most suitable for this purpose. Therefore, it's up to the developer to ensure their Transformations are idempotent.



Example 1: XML Document Transformation (XDT)

Consider a scenario where a user needs to modify the file OAuthServer\Web.config and wants to add an attribute to the oauth **Tag** in which the value of the configSource attribute is equal to OAuth.config.

1. Create a file OAuthServer\Web.config in the TransformationsofConfigFiles folder according to XDT rules.
2. Fill it in according to the XDT rules.
3. Commit the changes.
4. The Transformation is reflected in OAuthServer\Web.config after the next Deployment. Next time, the config file should give the same result as after the first application of the Transformation.
5. Sample XML Transformation:

```
``xml  
  
<?xml version="1.0"?>  
  
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">  
  
    <oauth configSource="OAuth.config" yourNewAttribute="value" xdt:Transform="SetAttributes" xdt:Locator="id" />  
  
</configuration>
```

- For XDT Documentation, visit [https://learn.microsoft.com/en-us/previous-versions/aspnet/dd465326\(v=vs.110\)](https://learn.microsoft.com/en-us/previous-versions/aspnet/dd465326(v=vs.110)).
- For Web.config Transformation Syntax for Web Project Deployment Using Visual Studio, visit <https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/transform-webconfig?view=aspnetcore-5.0>



Example 2: JSON Document Transformation (JDT)

Consider a scenario where a user needs to add a new plugin to OAuthServer\OAuthServer.Plugins.json.

1. Create a file named OAuthServer\OAuthServer.Plugins.json in the directory TransformationsOfConfigFiles according to JDT rules.
2. Fill it in according to the JDT rules.
3. Commit the changes.

The Transformation will reflect in OAuthServer\Web.config after the next Deployment. Next time, the config file should give the same result as after the first application of the transformation.

Sample JDT Transformation:

```
```json
{
 "@jdt.merge": {
 "@jdt.path": "$['OAuthServer.Plugins']",
 "@jdt.value": [
 {
 "Name": "New.Aras.Plugin",
 "Enabled": true
 }
]
 }
}
```
```



- For JDT documentation, visit <https://github.com/microsoft/json-document-transforms/wiki> .



Ignore Configuration Files Transformation

This need usually arises when the user does not require Transformation or if a validation error occurs during implementation.

To ignore Transformation add a line with file location (relative path) in the following file:
TransformationsOfConfigFiles\.transformations.ignore.

For more details and Transformation syntax, please refer to the file in the cloned Repository:
TransformationsOfConfigFiles\Readme.md.



Environment Specific Configuration

Environment-specific configuration functionality is an Aras DevOps feature available for Aras Enterprise customers working with containerized deployments.

This functionality enables customers to deploy Aras Innovator instances from one single Git **Branch** to different environments (System Integration Testing, User Acceptance Testing, Staging, and Production) and introduce variability in the configuration. The configuration Transformations use Variables instead of fixed values. Customizable Azure DevOps Variable Groups define Variable Values.

Aras Enterprise customers use this feature to manage variability in Aras Innovator configuration files on different environments without having to manage separate **Branches** for each environment.

The following steps outline the high-level process of environment-specific configuration:

1. Add a Transformation that adds a custom Variable to a configuration file.
2. Add the custom Variable to the environment-specific Variable Group.
3. Run CI **Pipeline** to add the transformation to the Deployment **Package**.
4. Run Deploy **Pipeline** with the appropriate Variable Group.



Example: Define an environment-specific login screen

This section illustrates how to create two SIT Deployments with different configurations. For each Deployment, different login screen background images will be created.

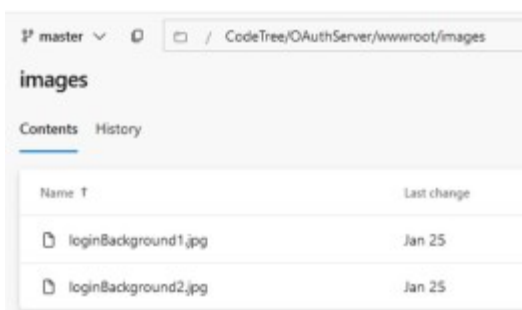
The following steps outline the process of creating different login screen background images:

1. Add a Transformation that adds custom Variable to a configuration file. The following is an example of a Transformation that sets the `$(imagePath)` value in the `BackgroundImageUrl` property. This should be added to the `TransformationsOfConfigFiles/OAuthServer/OAuthServer.config` folder:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
  <appSettings>
    <add key="BackgroundImageUrl" xdt:Locator="Match(key)" xdt:Transform="SetAttributes(value)" value="
  </appSettings>
</configuration>
```



New background images should be added to the `CodeTree/OAuthServer/wwwroot/images` folder.



2. Commit the Transformation and images and push them to Work Repository in Azure DevOps.



3. Next, set the values for Variables for each Environment. By default, the project has the following five Variable Groups defined for custom configurations:

Pipelines	fx CustomConfigValues-CI
Pipelines	fx CustomConfigValues-PROD
Environments	fx CustomConfigValues-SIT
Releases	fx CustomConfigValues-STG
Library	fx CustomConfigValues-UAT

4. Select **CustomConfigValues-SIT** Variable Group.

Pipelines	fx CustomConfigValues-CI
Pipelines	fx CustomConfigValues-PROD
Environments	fx CustomConfigValues-SIT
Releases	fx CustomConfigValues-STG
Library	fx CustomConfigValues-UAT

5. Add a Variable with the Name **imagePath** and the Value `~/images/loginBackground1.jpg`

Library > CustomConfigValues-SIT

Variable group | Save | Clone | Security | Pipeline permissions | Approvals and checks | Help

Properties

Variable group name

Description

Link secrets from an Azure key vault as variables ⓘ

Variables

Name ↑	Value
imagePath	~/images/loginBackground1.jpg

+ Add

6. Run **ContinuousIntegration Pipeline**.

7. Run **Deploy Pipeline** with default parameters using the Continuous Integration from step 6 as a Resource.



This **Pipeline** creates a new Deployment using the **CustomConfigValues-SIT** Variable Group for custom configuration. The **BackgroundImageUrl** Value will be set to the Value of **imagePath** from the Variable Group as defined in the Transformation.

Environment-specific Variables from the **CustomConfigValues-SIT** Variable Group are always applied by Deploy Aras Innovator Pipelines during initial Deployment or Redeployment.

8. Visit the Deployed Aras Innovator to verify the change in the background image on the Login page.

To create another Deployment using another configuration of the image path, modify the default Variable Group (**CustomConfigValues-SIT**) or create a new Variable Group for the second Deployment.

The following is the example of deployment with a new Variable Group with **my-custom-group** Name:

- In Azure DevOps, select **Library** and click **+ Variablegroup** to create a new Variable Group.



- Set the name to **my-custom-group**.
- Add the **imagePath** Variable and set the path to the second image as the Value (**~/images/loginBackground2.jpg**).

Library > my-custom-group

Variable group | Save | Clone | Security | Pipeline permissions | Approvals and checks | Help

Properties

Variable group name
my-custom-group

Description

Link secrets from an Azure key vault as variables ⓘ

Variables

Name ↑	Value
imagePath	~/images/loginBackground2.jpg

+ Add

- Run **Deploy Pipeline** again with the new **Variable group name** set as the parameter.

Run pipeline ✕

Select parameters below and manually run the pipeline

Branch/tag

Select the branch, commit, or tag

Deployment type:

Deploy timeout in minutes

Optional variable group name: (Default one is used if none provided)

Custom config variable group name: (Default one is used if none provided)

Advanced options

Variables
8 variables defined >

Stages to run
Run as configured >

Resources
Use latest version of all resources >

Enable system diagnostics



- Go to the deployed Aras Innovator to check that the background image on the Login page has changed.



Default Variable Groups

By default, there are five auto-generated Variable Groups:

- CustomConfigValues-CI
- CustomConfigValues-SIT
- CustomConfigValues-UAT
- CustomConfigValues-STG
- CustomConfigValues-PROD

The first group, **CustomConfigValues-CI**, is the default group for the CI pipeline. The rest are used by default when running the Deploy Aras Innovator Pipeline to SIT, UAT, Staging, or Production.

Additionally, custom Variable Groups can be created and passed as parameters.

Custom config variable group name: (Default one is used if none provided)



Public Variables list

The environment-specific configuration functionality available for Aras Enterprise customers working with containerized deployments supports the list of Public Variables. These public variables can be used in transformations without adding a variable to Variable Groups as their values are automatically added during deployment. See the list of supported public variables below:

- ARAS_AGENT_REPLICATION_STATUS
- ARAS_AGENT_REPLICATION_INTERVAL
- ARAS_AGENT_REPLICATION_MAX_BATCH
- ARAS_AGENT_REPLICATION_MAX_PENDING
- ARAS_AGENT_REPLICATION_TIMEOUT
- ARAS_AGENT_REPLICATION_USER
- ARAS_AGENT_LoggerConfiguration__MinimumLevel
- ARAS_AGENT_AgentService__InnovatorServerUrl
- ARAS_INNOVATOR_INSTANCE_NAME
- ARAS_INNOVATOR_CLIENT_ORIGIN
- ARAS_INNOVATOR_CLIENT
- ARAS_INNOVATOR_CLIENT_INTERNAL
- ARAS_INNOVATOR_CLIENT_SERVER
- ARAS_INNOVATOR_DEPLOYMENT_NAME
- ARAS_INNOVATOR_SERVER
- ARAS_INNOVATOR_SERVER_INTERNAL
- ARAS_INNOVATOR_VAULT_SERVER_INTERNAL
- ARAS_INNOVATOR_CONVERSION_SERVER
- ARAS_INNOVATOR_VAULT_SERVER
- ARAS_INNOVATOR_OAUTH_SERVER
- ARAS_INNOVATOR_OAUTH_SERVER_INTERNAL
- ARAS_INNOVATOR_AGENT_SERVICE
- ARAS_INNOVATOR_TRANSIENT_ERROR_NUMBERS
- ARAS_INNOVATOR_SERVER_METHOD_ASSEMBLY_CACHE_EXPIRATION
- ARAS_CLIENT
- ARAS_CLIENT_REDIS_CONNECTION_STRING
- ARAS_CLIENT_REDIS_KEY_PREFIX
- ARAS_CLIENT_DATA_PROTECTION_KEY_LIFETIME



- ARAS_CLIENT_LoggerConfiguration__MinimumLevel
- ARAS_OAUTH_CERTIFICATE_PASSWORD
- ARAS_OAUTH_BASE_PATH
- ARAS_OAUTH_LoggerConfiguration__MinimumLevel
- ARAS_OAUTH_DATA_PROTECTION_KEY_LIFETIME
- ARAS_OAUTH_REDIS_CONNECTION_STRING
- ARAS_OAUTH_REDIS_KeyPrefix
- ARAS_SERVER_MAIL_DELIVERYMETHOD
- ARAS_SERVER_SMTP_SERVER
- ARAS_SERVER_SMTP_PORT
- ARAS_SERVER_SMTP_SSL
- ARAS_SERVER_SMTP_USERNAME
- ARAS_SERVER_SMTP_PASSWORD
- ARAS_SERVER_DATABASES
- ARAS_SERVER_ACCESSCONTROLMAXAGE
- ARAS_SERVER_ACCESSCONTROLEXPOSEHEADERS
- ARAS_SERVER_ACCESSCONTROLALLOWORIGIN
- ARAS_SERVER_INITIALIZE_DATABASES_ON_STARTUP
- ARAS_SERVER_LoggerConfiguration__MinimumLevel
- ARAS_SERVER_Tracing__Instrumentation__SqlClient__Enabled
- ARAS_SERVER_VAULT_REQUEST_TIMEOUT
- ARAS_SERVER_REDIS_CONNECTION_STRING
- ARAS_SERVER_REDIS_KEY_PREFIX
- ARAS_SERVER_REDIS_RETRY_BACKOFF_MILLISECONDS
- ARAS_SERVER_REDIS_PREVENT_THREAD_THEFT
- ARAS_SERVER_DATA_PROTECTION_KEY_LIFETIME
- ARAS_SERVER_INITIALIZE_DATABASES_ON_STARTUP_TIMEOUT_SECONDS
- ARAS_SERVER_REPORT_SERVER
- ARAS_SERVER_MSGRAPHAPICLIENT_TENANTID
- ARAS_SERVER_MSGRAPHAPICLIENT_CLIENTID
- ARAS_SERVER_MSGRAPHAPICLIENT_CLIENTSECRET
- ARAS_VAULT_INNOVATORSERVERURL
- ARAS_VAULT_ACCESSCONTROLALLOWORIGIN
- ARAS_VAULT_ISPLUGINSTORAGE
- ARAS_VAULT_CLOUDSTORAGEPLUGINNAME



- ARAS_VAULT_LoggerConfiguration__MinimumLevel
- ARAS_VAULT_RedisDataStorage__Enabled
- ARAS_VAULT_RedisDataStorage__ConnectionString
- ARAS_VAULT_RedisDataStorage__KeyPrefix
- ARAS_DB_NAME
- ARAS_CONVERSION_USER
- ARAS_CONVERSION_LoggerConfiguration__MinimumLevel



Adding a Secret

To use Secret as an environment-specific Variable, you should open the Variable Group and mark the Variable as Secret.

The following steps outline the process of marking a Variable as Secret:

1. From Azure DevOps, go to **Library**.
2. Select a Variable Group.
3. In the **Variables** section, mark a Variable as Secret by clicking **Change variable type to plain text**.



As Secrets are encrypted Variables with a higher protection level, users must declare them in the `env_vars` section of the `TransformationsOfConfigFiles/environment-specific-secrets.yaml` file.

This will allow reading Secrets and passing them, along with other environment-specific Variables, to Deployments during the Deploy Aras Innovator or Continuous Integration Pipelines.

If the Secret is not defined in the `environment-specific-secrets.yaml` file, it will be read as an empty Variable, so use caution.



Variables Naming and Scope

By default, a Variable, such as **imagePath**, can have any name and will be common to all servers.

A Variable for one specific server can be specified. Suppose the user wants to ensure the Variable will be used on one specific server (OAuth server or conversion server). In that case, the Variable Name should use a prefix, for example, **oauthimagePath**. The prefixes are **oauth**, **client**, **server**, **conversion**, **agent**, and **vault**.



Restrictions

The environment-specific configuration functionality has the following restrictions:

1. The length of a Variable Value in Azure DevOps is limited to 4096 symbols.
2. The total length of the Variables file created by converting Variables and Values stored in the Variable Group must be less than 25600 characters to fit the key vault Secret length. This limitation is because each Deployment Pipeline creates a Variables file based on the Variable Group and saves this file into the Azure Key Vault to provide the possibility of auditing history and rolling back values.



External authentication

External authentication functionality is an Aras DevOps feature available for Aras Enterprise customers working with containerized deployments.

This functionality enables customers to configure Aras Innovator instances to allow single sign-on using an external identity provider.

Each external authentication consists of two steps:

1. External authentication in the external identity provider.
2. Process the result of external identity provider logins by mapping an external user to the Aras Innovator user. Each external authentication has its own user format, so it is important that the user mapper can handle any user format. The Generic User Mapper plugin is flexible in mapping an external user to an Aras Innovator user for multiple authentication types.

Configuring authentication plugins for each step is necessary for single sign-on using external authentication to Aras Innovator.

The following steps outline the high-level process of external authentication configuration:

1. Configure the external identity provider (e.g., add application registry).
2. Add Transformation for external authentication plugin (e.g., `Aras.OAuth.Server.Plugins.Saml2Authentication` plugin) and for user mapper plugin (e.g., using `Aras.OAuth.Server.Plugins.GenericUserMapper` plugin).
3. Run the CI Pipeline and deploy it.
4. Create a user that corresponds to mapping.
5. Configure access to the external identity provider (e.g., DNS settings).



Aras Innovator External Authentication using SAML 2.0 Authentication

Aras Innovator allows administrators to control the maintenance of user logins. One method is the SAML 2.0 Authentication Plugin, which provides a way to log into Aras Innovator using the SAML 2.0 authentication protocol.

To view the documentation of the SAML 2.0 protocol, visit <https://www.oasis-open.org/standards/#samlv2.0>.

SAML 2.0 is a protocol for exchanging authentication and authorization data between security domains. The XML-based protocol uses security tokens containing assertions to pass information about a principal (usually an end user) between a SAML authority, known as an Identity Provider, and a SAML consumer, known as a Service Provider. SAML 2.0 enables web-based, cross-domain single sign-on (SSO), which helps to reduce the administrative overhead of distributing multiple authentication tokens to a user.

The Aras Innovator logon may be customized using the SAML 2.0 Authentication Plugin described in this section. This plugin provides a way to use external identity providers by the SAML 2.0 protocol for Aras Innovator. The customization requires changes in the OAuth server configuration to enable the SAML 2.0 authentication plugin.

The SAML 2.0 protocol includes the following:

- An identity provider authenticates users and gives the service provider an authentication assertion if successful.
- A service provider relies on the identity provider to authenticate users. The OAuth server acts as a service provider in the system.

The following steps outline the high-level process of SAML2 authentication configuration:

1. Configure external identity provider (e.g., add application registry).
2. Add Transformations for the `Aras.OAuth.Server.Plugins.Saml2Authentication` plugin and the `GenericUserMapper` plugin.
3. Run **CI Pipeline** and **Deploy Pipeline**.
4. Create a user that corresponds to the mapping.
5. Configure access to the external identity provider (e.g., DNS settings).



Example: Setup of Aras Innovator SAML 2.0 Authentication with Azure as Identity Provider

This section provides an example of configuring a deployment to log in to Aras Innovator using an external user and the SAML2 protocol. For this Deployment, logging in using a Google account will be configured.

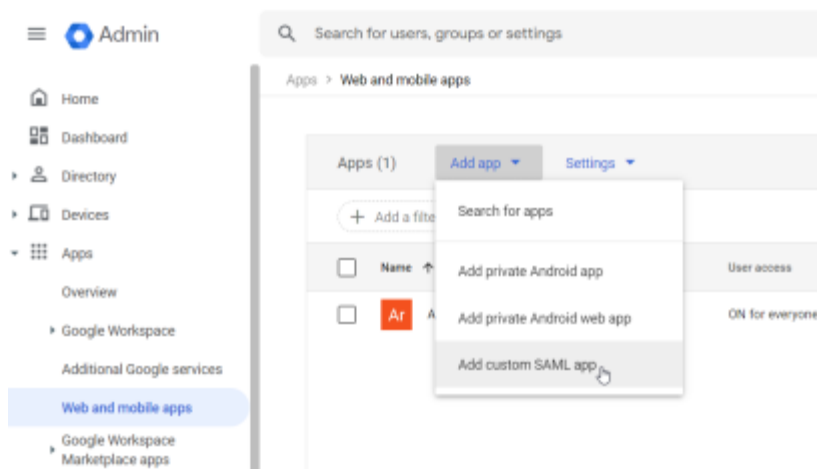
The following sections outline the process of configuring external authentication.

Configure external identity provider (e.g. add app registry)

For more details, refer to the manual: <https://support.google.com/a/answer/6087519> .

The following steps outline the process of configuring the external identity provider:

1. Ensure there is a Google admin account on admin.google.com . The e-mail should be of the domain which will be used as the Service provider (Aras Innovator). For example, if the instance is https://domain.com/instance/, the e-mail should be username@domain.com.
2. Log into <https://admin.google.com/> .
3. From the left pane, Go to **Apps** and click **Web and mobile apps**.
4. Click **Add app**, then select **Add custom SAML Apps**.



5. Fill in the **App** details.



× Add custom SAML app

To configure single sign-on (SSO) for SAML apps, follow your service provider's instructions. [Learn more](#)

Option 1: Download SP metadata

[DOWNLOAD METADATA](#)

OR

Option 2: Copy the SSO URL, entity ID, and certificate

SSO URL

Entity ID

Certificate

SHA-256 fingerprint

BACK CANCEL CONTINUE

6. Click the **Continue** button.
7. Download metadata on the computer and copy **EntityId** (download the signing certificate, if required).



Service provider details
To configure single sign on, add service provider details such as ACS URL and entity ID. [Learn more](#)

ACS URL
https://{OAuthServerURL}/Saml2-AzureAD/Acs

Entity ID
https://{OAuthServerURL}/Saml2-AzureAD/

Start URL (optional)

Signed response

Name ID
Defines the naming format supported by the identity provider. [Learn more](#)

Name ID format
EMAIL

Name ID
Basic Information > Primary email

BACK CANCEL CONTINUE

11. Click the **Continue** button and save.
12. Click **Apps**, click on **Web and mobile apps**, and check if user access is **ON for everyone**.

SAML

Te TestApp

User access ▼

To make the managed app available to select users, choose a group or organizational unit. [Learn more](#)

[View details](#)

ON for everyone

Service provider details ▼

Certificate	ACS URL	Entity ID
Google_2025-5-5-33622_SAML2_0 (Expires May 5, 2025)	https://devsaas214-nprd-01.gcs.arasqa.com/instance/OAuthServer/Saml2-AzureAD/ACS	https://devsaas214-nprd-01.gcs.arasqa.com/instance/OAuthServer/Saml2-AzureAD/

SAML attribute mapping

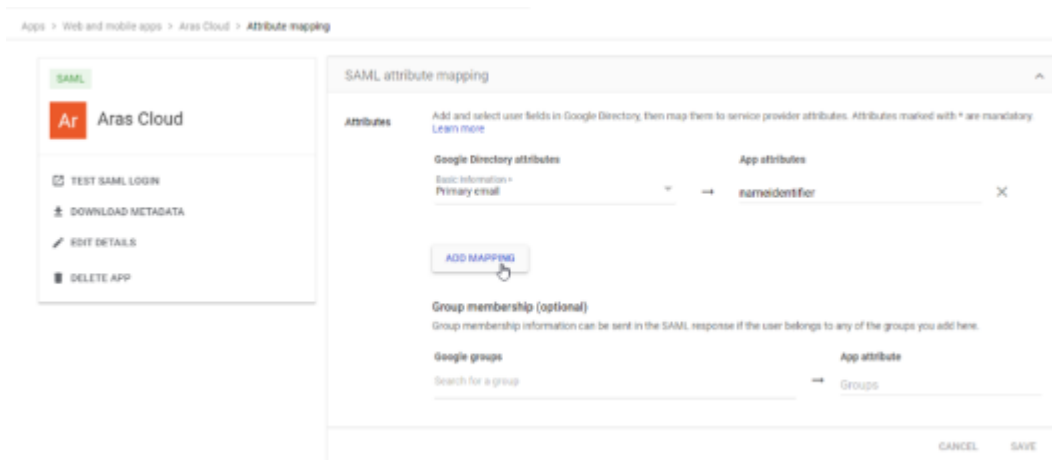
SAML attribute mapping isn't configured

Map Google directory user profile fields to SAML service provider attributes.

[Configure SAML attribute mapping](#)

13. Click **Add Mapping** and change mapping options to receive the correct claim after logging into Google.





14. Go to the **Home** Tab, and in the **Users** section, assign users to log in.

Refer to the following Microsoft link about configuring Azure as an identity provider:

<https://learn.microsoft.com/en-us/entra/identity/enterprise-apps/add-application-portal-setup-ss0> .

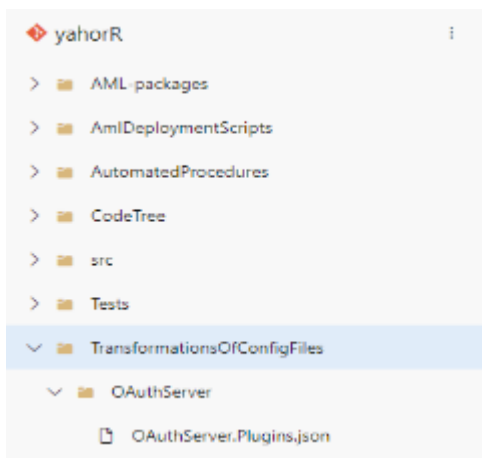
Add transformation for Aras.OAuth.Server.Plugins.Saml2Authentication plugin and for Aras.OAuth.Server.Plugins.GenericUserMapper plugin.

To turn on Aras Innovator SAML 2.0 Authentication, update the **OAuthServer.Plugins.json** settings with the Transformation mechanism using JDT Transformation.

The readme file in the TransformationsOfConfigFiles folder of the Work Repository in Azure DevOps contains more information.

The values of the required parameters can be obtained from the Configure external identity provider section.

The OAuthServer.Plugins.json file must be added to TransformationsOfConfigFiles/OAuthServer of the Repository.



The file should contain the JDT Transformation of the configuration of **Aras.OAuth.Server.Plugins.Saml2Authentication** and **Aras.OAuth.Server.Plugins.GenericUserMapper** plugins.

The following is an example of JDT Transformation using OAuthServer.Plugins.json file:



```
{
  "@jdt.merge": {
    "@jdt.path": "$,['OAuthServer.Plugins']",
    "@jdt.value": [
      {
        "Name": "Aras.OAuth.Server.Plugins.Saml2Authentication",
        "Enabled": true,
        "Options": [
          {
            "AuthenticationType": "Saml2-AzureAD",
            "DisplayName": "Saml2 Google",
            "ServiceProviderOptions": {
              "EntityId": "https://{OAuthServerURL}/Saml2-AzureAD/"
            },
            "IdentityProviderOptions": {
              "EntityId": "https://accounts.google.com/o/saml2?idpid=C02c0u5vx",
              "MetadataSource": "MetadataLocation",
              "MetadataLocation": "/secure_files/sit_GoogleIDPMetadata.xml"
            }
          }
        ]
      }
    ],
  },
}
```



```
{
  "Name": "Aras.OAuth.Server.Plugins.GenericUserMapper",
  "Enabled": true,
  "Options": [
    {
      "AuthenticationType": "Saml2-AzureAD",
      "InnovatorUserNameFormat": "{username}",
      "ClaimActions": [
        {
          "ActionName": "CreateFrom",
          "ActionOptions": {
            "ClaimType": "username",
            "SourceClaimType": "http://schemas.xmlsoap.org/ws/2005/05/identity/claims",
            "ReplacePattern": "@.+",
            "Replacement": "",
            "PatternOptions": [ "IgnoreCase", "Singleline" ]
          }
        },
        {
          "ActionName": "Validate",
          "ActionOptions": {
            "ClaimType": "username",
```



Since the username field in Aras Innovator is limited by the number of symbols, mapping should consider this limitation.

Refer to the Generic User Mapper section for more information about **Generic User Mapper** configuration.

The above example demonstrates one of the basic SAML2 flows using a file configuration of EntityId and Metadata.

Following SAML2 flows are also supported:

- Configuration with URLs to an external provider.
- Configuration with external files, such as certificates and metadata files.

For more details about configuration with external files, refer to the Configuring Secure Files section.

Commit Changes and Run Pipeline

Once all the changes are complete, commit the changes, run the CI Pipeline, and deploy the Pipeline.

Create a User for Corresponding Mapping

Login to the Aras Innovator instance as an administrator and create a user according to the mapping strategy. For example:

The Login Name will be the username of the e-mail address, which is the prefix of @.

The screenshot shows a user creation form in Aras Innovator. At the top, there is a header 'User Name' with a user icon, a star, and a flag. Below the header is a toolbar with icons for Edit, Refresh, Undo, Redo, Add, Delete, and a menu icon. The form itself is titled 'User' and contains several input fields arranged in a grid:

Login Name username	First Name User	Last Name Name	Company Name
Password ***	Email username@domain.com	Telephone	Mail Stop
Confirm Password ***	Fax	Home Phone	Picture
Logon Enabled <input checked="" type="checkbox"/>	Cell	Pager	
Starting Page	Employee #	Manager	
Default Vault Default	Working Directory		



Configure access to the external identity provider (e.g., DNS settings)

If Google requests domain verification, please follow the provided instructions.

The following steps outline the process of configuring the communication between Google and the domain:

1. Open domain settings in admin.google.com and fill in the required properties. For more details, see the [Configure External Identity Provider](#) section.
2. Open the Azure portal DNS settings and follow the Google instructions about adding MX records for the domain to make the domain verified. Two records were added to DNS settings.

devsaas214-nprd-01 ...
gcs.arasqa.com

Save Discard Delete Users Metadata

devsaas214-nprd-01.gcs.arasqa.com

Type
MX

TTL * 1 TTL unit Hours

Preference	Mail exchange
1	SMTP.GOOGLE.COM ***
15	VNGALAG4QJPKSCZERROPRTJMABYEWK424ZVBRKMNHL7EKO6A53ZQ.MX-VERIFICATIO... ***

mail.contoso.com

3. Continue verification in admin.google.com .

After following all the above steps, the login should be successful. For more details, refer to the [SAML2 Authentication Plugin Configuration](#) section.



Configuring Secure Files

The following steps outline the high-level process of using the certificates during deployment:

1. Upload a certificate to secure file storage.
2. Configure file permissions for Pipeline.
3. Add a Transformation with a link to the certificate.

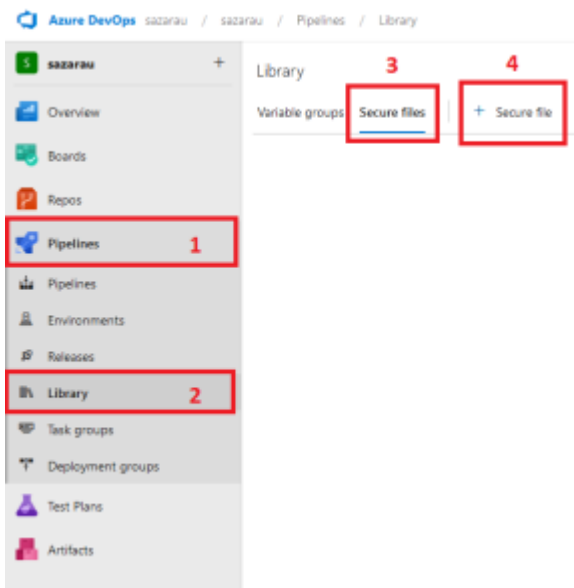
File names must begin with one of the following CI prefixes to specify the environment in the Continuous Integration Pipeline where the files are used: **SIT**, **UAT**, **STG**, or **PROD**. For example, **SIT-SAML.pem**. File names not starting with one of these designated prefixes will be ignored.



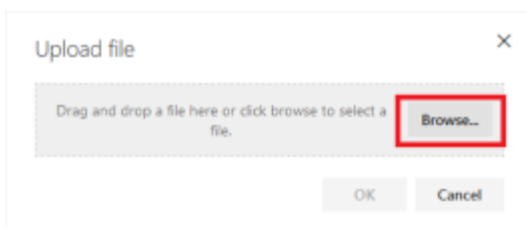
Upload a certificate as a secure file

The following steps outline the process of uploading a certificate as a secure file:

1. After uploading a certificate as a secure file in Azure DevOps, expand **Pipelines**.
2. Select **Library**.
3. Select the **Secure files** Tab.
4. Click the **+ Secure file** button.

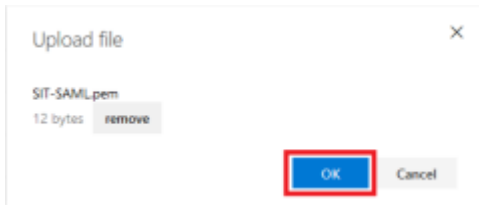


5. Click **Browse...** button and select the secure file.



6. Click **Ok**.



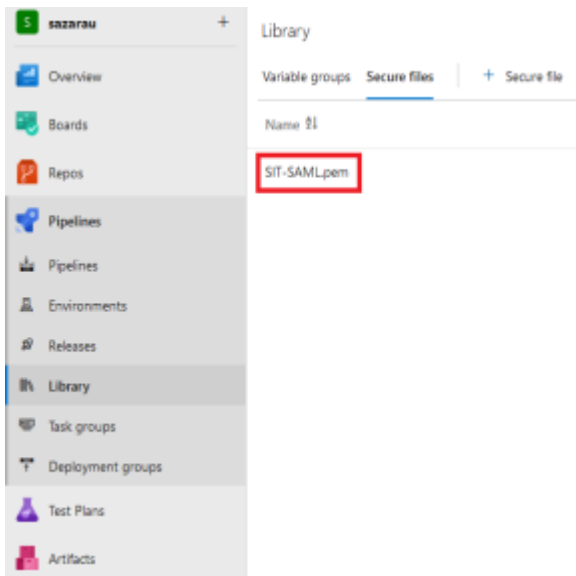


The file will be available in the list of secure files. If needed, click + **Secure file** again to add another file.

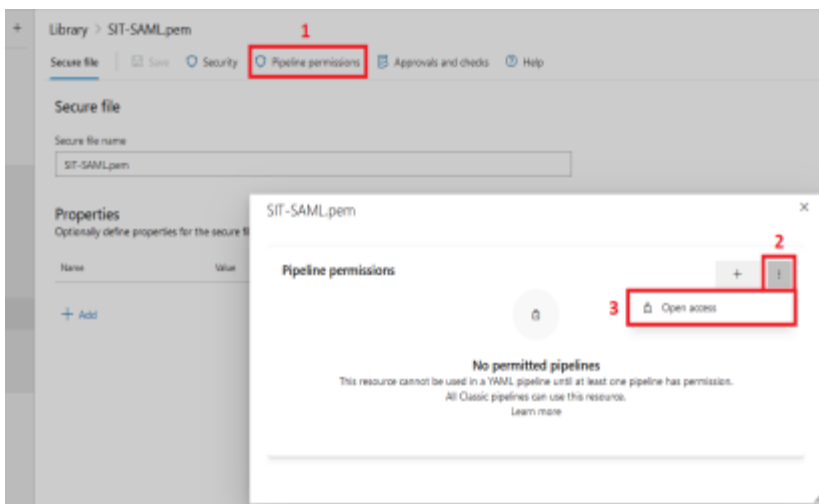
Configure File Permissions for a Pipeline

The following steps outline the process of configuring file permissions for a Pipeline:

1. Click on the file name to open file properties.

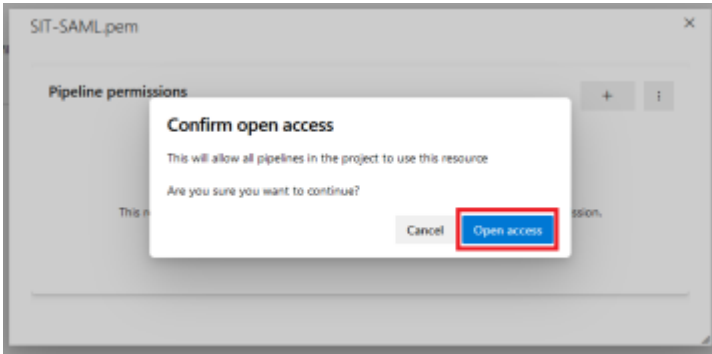


2. Click the **Pipeline permissions** Tab, the **ellipses** button, and select **Open access** from the menu.

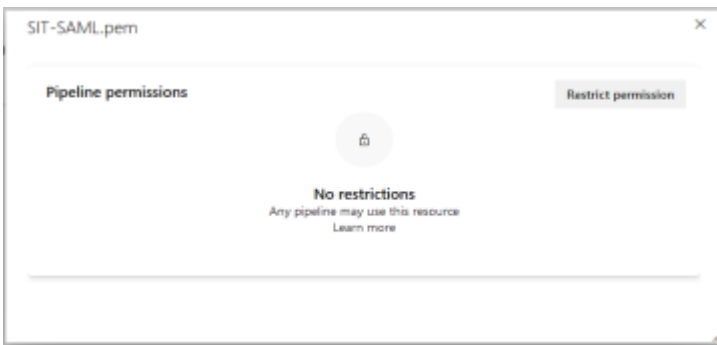


3. Click the **Open access** button to allow Pipelines to download the file.

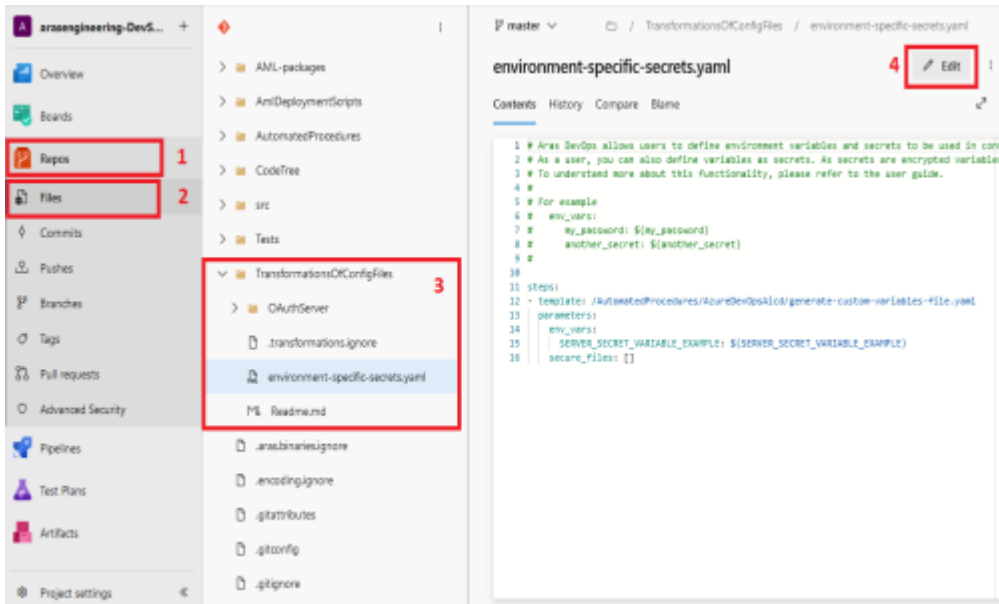




Once the permission has configured successfully, close the window.



4. Add the certificate file name to a configuration file. This allows the file to be used in Pipelines. Click **Repos** and open the TransformationsOfConfigFiles/environment-specific-secrets.yaml file.
5. Click the **Edit** button.



6. If the `secure_files` section is missing, add the file name to it. To have a valid YAML file, all indents should be the same as in the screenshots below.

```

1 # Aras DevOps allows users to define environment variables and secrets to be used in
2 # As a user, you can also define variables as secrets. As secrets are encrypted vari
3 # To understand more about this functionality, please refer to the user guide.
4 #
5 # For example
6 #   env_vars:
7 #     my_password: $(my_password)
8 #     another_secret: $(another_secret)
9 #
10
11 steps:
12 - template: /AutomatedProcedures/AzureDevOpsA1cd/generate-custom-variables-file.yaml
13   parameters:
14     env_vars:
15       SERVER_SECRET_VARIABLE_EXAMPLE: $(SERVER_SECRET_VARIABLE_EXAMPLE)
16   secure_files:
17     - SIT-SAML.pem
18     - SIT-SAML-123.pem
19     - UAT-SAML.pem
20

```

7. Click the **Commit** button.



```

1 # Aras DevOps allows users to define environment variables and secrets to be used in config transformations.
2 # As a user, you can also define variables as secrets. As secrets are encrypted variables with higher protection
3 # To understand more about this functionality, please refer to the user guide.
4 #
5 # For example
6 # env_vars:
7 #   my_password: $(my_password)
8 #   another_secret: $(another_secret)
9 #
10
11 steps:
12 - template: /AutomatedProcedures/AzureDevOpsAics/generate-custom-variables-file.yaml
13 parameters:
14   env_vars:
15     SERVER_SECRET_VARIABLE_EXAMPLE: $(SERVER_SECRET_VARIABLE_EXAMPLE)
16   secure_files:
17     - SIT-SAML.pem
18
  
```

8. Set the Comment to **Commit** (use the default **Commit** Message) and click the **Commit** button.



Add a Transformation with Link to Certificate

The following steps outline the procedure for adding a Transformation with a link to the certificate:

1. Add the Transformation file for the following file to the Repository:
TransformationsOfConfigFiles/OAuthServer/OAuthServer.Plugins.json
For more information about setting up SSO, follow the steps in the Example: Setup of Aras Innovator SAML 2.0 Authentication with Azure as Identity Provider section.
2. Set the content to the Transformation file with links to secure files. An example follows:



```

{
  "@jdt.merge": {
    "@jdt.path": "$.[ 'OAuthServer.Plugins' ]",
    "@jdt.value": [
      {
        "Name": "Aras.OAuth.Server.Plugins.Saml2Authentication",
        "Enabled": true,
        "Options": [
          {
            "AuthenticationType": "Saml2-AzureAD2",
            "DisplayName": "Saml2 Google with signing",
            "ServiceProviderOptions": {
              "EntityId": "https://{OAuthServerURL}/Saml2-AzureAD/"
            },
            "IdentityProviderOptions": {
              "EntityId": "https://accounts.google.com/o/saml2?idpid=C014h6zu2",
              "MetadataSource": "MetadataOptions",
              "Metadata": {
                "SingleSignOnService": {
                  "Location": "https://accounts.google.com/o/saml2/idp?idpid=C014h6zu2",
                  "Binding": "HttpRedirect"
                }
              }
            }
          }
        ]
      }
    ]
  }
}

```



```
    "WantAuthnRequestsSigned": false,  
    "SigningCertificate": {  
      "SourceType": "File",  
      "FilePath": "/secure_files/stg-Google_2029-5-7-91113_SAML2_0.pem"  
    }  
  }  
}  
}  
}  
]  
,  
{  
  "Name": "Aras.OAuth.Server.Plugins.GenericUserMapper",  
  "Enabled": true,  
  "Options": [  
    {  
      "AuthenticationType": "Saml2",  
      "InnovatorUserNameFormat": "{uid}",  
      "ClaimActions": [  
        {  
          // Validate uid (default Innovator users are denied).  
          "ActionName": "Validate",  
          "ActionOptions": {
```



SAML2 Authentication Plugin Configuration

The Aras Innovator login may be customized using the SAML 2.0 Authentication Plugin described in this section. This plugin provides a way to use external identity providers by the SAML 2.0 protocol for Aras Innovator. The customization requires changes in the OAuth server configuration to enable the SAML 2.0 authentication plugin.

Changes in the configuration should be made using a transformation mechanism. For more information about full flow, refer to the [Example: Setup of Aras Innovator SAML 2.0 Authentication with Azure as Identity Provider](#) section.



Base Configuration

Metadata for the identity and service providers must be configured to set up the base SAML 2.0 plugin configuration. The metadata contains all necessary information for communication between the providers.

The following is an example of the base SAML 2.0 plugin configuration:

```
{
  "Name": "Aras.OAuth.Server.Plugins.Saml2Authentication",
  "Enabled": true,
  "Options": [{
    "AuthenticationType": "<AuthenticationType>",
    "DisplayName": "<DisplayName>",
    "ServiceProviderOptions": {
      "EntityId": "<ServiceProviderEntityId>"
    },
    "IdentityProviderOptions": {
      "EntityId": "https://idp.example.com"
    }
  ]
}
```

1. Important

Ensure that the JSON is valid. The ',' symbol should appear between configuration sections. Also, ensure that Options is a JSON array with at least one object.

This configuration example allows to specify the following parameters:

- **AuthenticationType:** Describes the name of the authentication scheme added to the OAuth server.
- **DisplayName:** The label that appears in the Login with dropdown on the Aras Innovator login page.
- **ServiceProviderOptions:** The options for configuring the service provider.
- **EntityId:** The unique identifier of the service provider (required).
- **IdentityProviderOptions:** The options for configuring the identity provider.



- **EntityId:** The unique identifier of the identity provider (required). If EntityId is a URL, it can be used by the service provider to load the identity provider metadata. If the metadata is not located by the URL, refer to the Configuring Identity Provider Metadata section to configure the identity provider metadata.

To make this base configuration work, it is necessary to configure service provider metadata in an identity provider management system after configuring the identity provider metadata (see the description of the EntityId property). For instructions, refer to the Configuring Service Provider Metadata section.

1. **Important**

IIS must be restarted after installing the SAML 2.0 authentication plugin.



Configuring Identity Provider Metadata

To use SAML 2.0 authentication, the service provider should know all the relevant information for communication with the identity provider.

When the identity provider does not provide its metadata by URL to the EntityId property, the location for the metadata can be specified, or the metadata can be configured manually.

Configuring of Metadata Location

Metadata can be retrieved from the identity provider using a URL or an XML file.

The following example shows the configuration of the SAML 2.0 plugin:

```
{
  "Name": "Aras.OAuth.Server.Plugins.Saml2Authentication",
  "Enabled": true,
  "Options": [{
    "AuthenticationType": "<AuthenticationType>",
    "DisplayName": "<DisplayName>",
    "ServiceProviderOptions": {
      "EntityId": "<ServiceProviderEntityId>"
    },
    "IdentityProviderOptions": {
      "EntityId": "https://idp.example.com",
      "MetadataSource": "MetadataLocation",
      "MetadataLocation": "https://idp.com/metadata"
    }
  ]
}
```

This configuration example specifies the following parameters:

- **IdentityProviderOptions:** The options for configuring the identity provider.
- **EntityId:** The unique identifier of the identity provider (required). It must be the same as the identifier in the metadata.
- **MetadataSource:** The type of source from which metadata will be loaded.
- **MetadataLocation:** The location from which metadata will be loaded (required when MetadataSource has MetadataLocation value). It can be a URL, an absolute path to a local file,



or an app-relative path.

Configuring Metadata Manually

The SAML 2.0 authentication plugin enables specifying metadata options for the identity provider, if necessary.

The following is an example of a SAML 2.0 plugin configuration with identity provider metadata options:

```
{
  "Name": "Aras.OAuth.Server.Plugins.Saml2Authentication",
  "Enabled": true,
  "Options": [{
    "AuthenticationType": "<AuthenticationType>",
    "DisplayName": "<DisplayName>",
    "ServiceProviderOptions": {
      "EntityId": "<ServiceProviderEntityId>"
    },
    "IdentityProviderOptions": {
      "EntityId": "https://idp.example.com",
      "MetadataSource": "MetadataOptions",
      "Metadata": {
        "SingleSignOnService": {
          "Location": "https://idp.example.com/sso",
          "Binding": "HttpRedirect"
        },
        "ArtifactResolutionServices": [
          {
            "Index": "0",
            "Location": "https://idp.example.com/ars"
          }
        ],
        "WantAuthnRequestsSigned": false,
        "SigningCertificate": {
          "SourceType": "File",
          "FilePath": ".\\secure_files\\SigningCertificate.cer"
        }
      }
    }
  ]
}
```



This configuration example specifies the following parameters:

- **IdentityProviderOptions:** The options for configuring the identity provider (optional, like all child properties).
- **MetadataSource:** The source type from which metadata is loaded.
- **Metadata:** The configuration of the identity provider metadata (required if the MetadataSource has a MetadataOptions value).
- **SingleSignOnService:** Describes the authentication request protocol endpoint to which the user agent delivers the authentication request message or **Artifact** representing it (required if Metadata is configured).
- **Location:** The URL where the identity provider listens for incoming sign-on requests (required if **SingleSignOnService** is configured). The URL must be written in a way that the client understands since the client's web browser will be redirected to the URL. Specifically, this means that using a host name-only URL or a host name that only resolves on the server's network will not work.
- **Binding:** The SAML binding supported by the endpoint (defaults to HttpRedirect). Other possible values are HttpPost and Artifact.

1. **Important**

Single sign out (SingleLogoutService in terms of SAML 2.0 specification) is currently not supported.

- **ArtifactResolutionServices:** Zero or more elements that describe indexed endpoints used for dereferencing a SAML **Artifact** into a corresponding protocol message.
- **Index:** The non-negative integer that is used to distinguish the possible endpoints.
- **Location:** The URL to which a requester, having received an **Artifact**, sends a request for **Artifact** resolution (required if ArtifactResolutionServices is configured).
- **WantAuthnRequestsSigned:** A value (true or false) that indicates whether the identity provider wants the authentication request messages to be signed (default is false to support authentication flow without certificates).

1. **Important**



The `WantAuthnRequestsSigned` value is used together with the `ServiceProviderOptions`. The `SigningBehavior` option is only considered if `SigningBehavior` has `IfIdpWantAuthnRequestsSigned` or `Always` values.

- **SigningCertificate:** The identity provider's certificate to sign its messages. Refer to the identity provider certificate configuration description in the [Configuring Identity Provider Metadata](#) section.

Configuring Service Provider Metadata

The identity provider should have all the information to communicate with the service provider using SAML 2.0 authentication.

Base configuration of service provider metadata

A complex service provider metadata configuration is not needed. Only two options must be configured in an identity provider management system: the Assertion Consumer Service URL and the Service Provider Entity ID:

1. **Assertion Consumer Service URL:** The URL where SAML assertions are sent after a user has been authenticated. The URL is composed of the OAuth server URL, authentication type, and /Acs postfix, e.g., `https://server.com/instance/OAuthServer/Saml2-AzureAD/Acs`. Suppose internal and external clients connect to the OAuth server using different URLs. The Assertion Consumer Service URL should be configured in a used identity provider management system according to each OAuth server URL.
2. **Service Provider Entity ID:** The unique identifier most often used as an audience of SAML assertion.

Important

Case sensitivity is important while configuring the Assertion Consumer Service URL. Make sure that it is in the same registry as the path where the OAuth server authentication cookie is stored (it is the OAuth server path, e.g., `Innovator/OAuthServer` from the Assertion Consumer Service URL example above).

Some identity providers make configuring SAML 2.0 authentication possible using service provider metadata. This metadata is accessible via the Metadata Endpoint URL, which is composed of the OAuth server URL and the authentication type, such as `https://server.com/instance/OAuthServer/Saml2-AzureAD`. The next section provides information about configuring service provider metadata.

Configuring Metadata Options

The following is an example of configuring service provider metadata:



```
{
  "Name": "Aras.OAuth.Server.Plugins.Saml2Authentication",
  "Enabled": true,
  "Options": [{
    "AuthenticationType": "<AuthenticationType>",
    "DisplayName": "<DisplayName>",
    "ServiceProviderOptions": {
      "EntityId": "<ServiceProviderEntityId>",
    }
  ]
  "Metadata": {
    "CacheDuration": "01:00:00",
    "WantAssertionsSigned": true,
    "Organization": {
      "Names": [
        "Aras Corp"
      ],
      "DisplayNames": [
        "Aras"
      ],
      "Urls": [
        "https://www.aras.com"
      ],
      "Language": "en"
    },
    "Contacts": [
      {
        "Type": "Support",
        "Company": "Aras Corp",
        "GivenName": "John",
        "Surname": "Smith",
        "PhoneNumbers": [
          "978-806-9400"
        ],
        "Emails": [
          "info@aras.com"
        ]
      }
    ]
  }
}

  },
  "IdentityProviderOptions": {
    "EntityId": "http://idp.example.com"
  }
}
```



```
}]
}
```

This configuration example specifies the following parameters:

- **ServiceProviderOptions:** The options for configuring the service provider.
- **Metadata:** The configuration of service provider metadata.

1. Important

All Metadata configuration properties are optional.

- **CacheDuration:** The time interval during which anyone should cache the metadata presented by the service provider before trying to fetch a new copy (the default is one hour, 01:00:00).
- **WantAssertionsSigned:** The value (true or false) indicating whether the service provider wants assertions provided by the identity provider signed (the default is true).
- **Organization:** The basic information about an organization responsible for a SAML entity.
- **Names:** One or more language-qualified names that may or may not be suitable for human consumption (required if the Organization is configured; at least one item is required).
- **DisplayNames:** One or more language-qualified names suitable for human consumption (required if the Organization is configured; at least one item is required).
- **URLs:** One or more language-qualified URLs that specify a location to direct a user to additional information. The language qualifier refers to the material's content at the specified location (required if the Organization is configured; at least one item is required).
- **Language:** The language **Tag** in the xml:lang XML attribute for all Names, DisplayNames, and URLs (the default language is English, en language **Tag**).

1. Important

Examples of language **Tags** are ja (Japanese), de (German), and fr (French). The **Tags** for Identifying Languages specification has more information at <https://tools.ietf.org/html/rfc5646>.

- **Contacts:** The basic contact information for a person responsible for a SAML entity.



- **Type:** The type of contact (the default is Unspecified). Other possible values are Technical, Support, Administrative, Billing, and Other.
- **Company:** The name of the company for the contact person.
- **GivenName:** The first name of the contact person.
- **Surname:** The surname of the contact person.
- **PhoneNumbers:** Zero or more string elements specifying a telephone number for the contact person.
- **Emails:** Zero or more string elements specifying the e-mail address of the contact person.

1. Important

All Contacts configuration properties are optional.

Loading Metadata to the Identity Provider

The service provider's metadata endpoint is inaccessible to the identity provider because it is located on localhost. In this case, service provider metadata (configured in the Configuring metadata options section) can be downloaded from the Metadata Endpoint URL, saved as an XML file, and imported to the identity provider's side.

If the service provider's metadata endpoint URL is accessible from the identity provider's side, the URL can be configured in an identity provider management system.

The base plugin configuration is completed after the service provider metadata is loaded in the identity provider.



Configuring Certificates

The service and identity providers can use certificates to make communication more secure.

Configuring Identity Provider Signing Certificates

The identity provider can use a certificate to sign its messages. The certificate can either be loaded from a file or the Certificate Store.

The following is an example of configuring signing certificate loading from a file:

```
"IdentityProviderOptions": {  
  ...  
  "MetadataSource": "MetadataOptions",  
    "Metadata": {  
    "SigningCertificate": {  
      "SourceType": "File",  
      "FilePath":  
        ".\\secure_files\\SigningCertificate.cer"  
    }  
  }  
}
```

This configuration example specifies the following parameters:

- **SourceType:** The source type for certificate loading. It can be either File or CertificateStore.
- **FilePath:** The path to load the certificate from. The path is relative to the execution path of the application.

The following is an example of configuring signing certificate loading from the Certificate Store:



```

"IdentityProviderOptions": {
...
"MetadataSource": "MetadataOptions",
  "Metadata": {
"SigningCertificate": {
  "SourceType": "CertificateStore",
  "StoreLocation": "LocalMachine",
  "StoreName": "My",
  "FindType": "FindBySubjectDistinguishedName",
  "FindValue": "CN=CertificateSubject",
  "ValidOnly": true
}
}
}

```

This configuration example specifies the following parameters:

- **SourceType:** The source type for certificate loading can be either File or CertificateStore (required if SigningCertificate is configured).
- **StoreLocation:** The location of the store to search for the certificate (required if SourceType has CertificateStore value). There is no default value for the property. Possible values from the System.Security.Cryptography.X509Certificates.StoreLocation enumeration are CurrentUser and LocalMachine.
- **StoreName:** The name of the certificate store to search for the certificate (required if SourceType has CertificateStore value). There is no default value for the property.

Possible values from the System.Security.Cryptography.X509Certificates.StoreName enumeration are AddressBook, AuthRoot, CertificateAuthority, Disallowed, My, Root, TrustedPeople, and TrustedPublisher.

Important

It is recommended that the identity provider's certificate be kept in the "Other People" store, which is specified by the AddressBook enumeration value.

- **FindType:** The value type from the findValue property that will be used to find the certificate (required if SourceType has CertificateStore value). There is no default value for the property. The following values from the System.Security.Cryptography.X509Certificates.X509FindType



enumeration are supported: FindByThumbprint, FindBySubjectName, FindBySubjectDistinguishedName, FindByIssuerName, FindByIssuerDistinguishedName, FindBySerialNumber, FindByTemplateName, FindByApplicationPolicy, FindByCertificatePolicy, FindByExtension, FindByKeyUsage, and FindBySubjectKeyIdentifier.

Important

For security, the FindBySerialNumber enumeration value is recommended.

- **FindValue:** The search term (string) to find the certificate (required if SourceType has CertificateStore value). There is no default value for the property.
- **ValidOnly:** Value (true or false) indicating that the certificate that will be loaded must be valid (the default is false).

Important

The certificate is validated on expiration, correct signature, trusted root certificate, and other parameters from the base certificates chain policy. Refer to base policy errors: https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/ns-wincrypt-cert_chain_policy_status#members.



Additional Authentication Options Configuration

The SAML 2.0 authentication plugin allows configuring the authentication process by specifying additional options (not related to metadata and certificates).

The following is an example of a SAML 2.0 plugin configuration:

```
{
  "Name": "Aras.OAuth.Server.Plugins.Saml2Authentication",
  "Enabled": true,
  "Options": [{
    "AuthenticationType": "<AuthenticationType>",
    "DisplayName": "<DisplayName>",
    "ServiceProviderOptions": {
      "EntityId": "<ServiceProviderEntityId>"
    }
  },
  "OutboundSigningAlgorithm": "Sha256",
  "SigningBehavior": "IfIdpWantAuthnRequestsSigned",
  "NameIdPolicy": {
    "AllowCreate": true,
    "Format": "Persistent"
  },
  "RequestedAuthnContext": {
    "ClassRef": "urn:oasis:names:tc:SAML:2.0:ac:classes:Password",
    "Comparison": "Minimum"
  }
},
  "IdentityProviderOptions": {
    "EntityId": "https://idp.example.com",
  }
}]
}
```

This configuration example specifies the following parameters:

- **OutboundSigningAlgorithm:** The signing algorithm for metadata and outbound messages (the default is Sha256). The algorithm is used for both service and identity providers. Other possible values are Sha514, Sha384 and Sha1 (case-insensitive; full algorithms signatures are also supported).
- **SigningBehavior:** The signing behavior of generated authentication requests (the default is IfIdpWantAuthnRequestsSigned, sign authentication requests if the identity provider is



configured for it using its `WantAuthnRequestsSigned` property). Other possible values are: `Always` (always sign all authentication requests) and `Never` (never sign any authentication requests).

- **NameIdPolicy:** Controls the generation of the `NameIDPolicy` element in authentication requests to manage the name identifier returned in the subjects of assertions. The name identifier can be used to map an external user to an Aras Innovator user.
- **AllowCreate:** A nullable value (true or false) indicates whether the identity provider is allowed while fulfilling the request to create a new identifier to represent the principal (the default is null, meaning that the attribute is not included in generated authentication requests). When false, the requester constrains the identity provider to only issue an assertion to it if an acceptable identifier for the principal has already been established.
- **Format:** The requested format of `NameIDPolicy` for generated authentication requests (the default is `Transient`). Other possible values are: `NotConfigured`, `Unspecified`, `EmailAddress`, `X509SubjectName`, `WindowsDomainQualifiedName`, `KerberosPrincipalName`, `EntityIdentifier`, and `Persistent`.

Important

Setting this parameter does not guarantee returning the `NameID` in a specified format. Some identity providers may ignore the format of `NameIdPolicy`. If `Transient` format is specified, it is not permitted to specify `AllowCreate` according to the SAML 2.0 specification.

- **RequestedAuthnContext:** Specifies the authentication context requirements of authentication statements returned in response to a request or **Query**.
ClassRef: The URL reference identifying authentication context classes or declarations. It can be a full URL or a single word if one of the predefined classes is used in the SAML 2.0 Authentication context specification.

Important

Refer to section 3.4 of the SAML 2.0 Authentication context specification for predefined classes: <http://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf>.

- **Comparison:** The comparison method used to evaluate the requested context classes or statements (the default is `Exact`). Other possible values are `Minimum`, `Maximum`, and `Better`.



Aras Innovator User Setup

A user Item with the required login_name must exist in the Aras Innovator database with logon_enabled set to true to use an authentication plugin with a mapper plugin. The user's login_name must match the username received from the identity provider implemented in SAML 2.0.

Log in as an Authenticated User

To log in as a SAML 2.0 authenticated user, a user must select the name of the configured SAML 2.0 identity provider from the **Login with** drop-down menu, which presents this **Login** screen:



Press the **Continue** button to be redirected to the identity provider's login page.

Log in as a Standard User

To log in as a standard user, a user must select local authentication from the **Login with** drop-down menu (the default display name is "Aras Innovator"), which presents this **Login** screen:



Switching Between Logon Types

For end-user convenience, the Aras Innovator login screen caches the login method (SAML 2.0 or Standard) using cookies. To return to the logon mode selection dialog, add the prompt query parameter with a select_account value to the application URL.

Another option is to clear **Aras.OAuth.Preferences.AuthenticationType** and **Aras.OAuth.Preferences.Database** cookies.



Generic User Mapper

Aras Innovator has the flexibility to provide many options to administrators when controlling the maintenance of user logins by providing multiple external authentication options. A User Mapper must be created for each external authentication that maps an external user to an Aras Innovator user. Each external authentication has its own user format, so it is important that the User Mapper can handle any user format. The Generic User Mapper can flexibly configure mapping an external user to an Aras Innovator user for multiple authentication types.

The Generic User Mapper customizes the user mapping process during external login. This plugin provides a flexible configuration that allows it to be used with any authentication type.

A Transformation mechanism can be used to make changes to the configuration. For more information, refer to the [Example: Setup of Aras Innovator SAML 2.0 Authentication with Azure as Identity Provider](#) section.



GenericUserMapper Plugin Configuration

The GenericUserMapper plugin can be configured for multiple authentication types. Each authentication type mapping should be configured in a separate options object.

Below is an example of the GenericUserMapper plugin configuration for multiple authentication types:

```
{
  "Name": "Aras.OAuth.Server.Plugins.GenericUserMapper",
  "Enabled": true,
  "Options": [
    {
      "AuthenticationType": "<AuthenticationType1>",
      "InnovatorUserNameFormat": "{<Claim1>}"
    },
    {
      "AuthenticationType": "<AuthenticationType2>",
      "InnovatorUserNameFormat": "{<Claim2>}"
    }
  ]
}
```

Important

Ensure the Options is a JSON array with at least one object.

The plugin allows users to specify the following parameters for the options object:

- **AuthenticationType:** The name of the authentication scheme registered in the OAuth server for which these mapping options should be applied (this value should correspond to the AuthenticationType value of the authentication plugin).
- **InnovatorUserNameFormat:** A username format string that consists of fixed text intermixed with named placeholders. A placeholder is a claim type that appears enclosed in braces. The result is a string where the corresponding claim value replaces each placeholder. An example of a username format string is:

```
"Prefix_{<Claim1>}_{<Claim2>}_Postfix"
```



Important

The maximum length of Aras Innovator username is 32 characters, so be aware of this when configuring `InnovatorUserNameFormat`.

- **ClaimActions:** Actions that should be performed on Claims before generating a username based on `InnovatorUserNameFormat`.

Claim Actions Configuration

Claim Action Configuration objects contain the following settings:

- **ActionName:** Name of action.
- **ActionOptions:** Action configuration.

The `GenericUserMapper` plugin supports the following types of actions:

- **CreateFrom:** Creates new claim from a value.
- **Validate:** Allows or denies claim values.

CreateFrom Action

The **CreateFrom** action allows users to get a value from one claim, edit it using a regular expression, and save it in a new claim.

The **CreateFrom** action configuration contains the following options:

- **ClaimType:** Type of new claim where a new value is set.
- **SourceClaimType:** Type of claim where a value should be used.
- **ReplacePattern:** The regular expression pattern to match.
- **Replacement:** The string to replace the match.
- **PatternOptions:** Options for matching.

Important

Supported `PatternOptions` values are described at <https://docs.microsoft.com/en-us/dotnet/api/system.text.regularexpressions.regexoptions#fields>.



The **CreateFrom** action uses the following algorithm:

1. Get the value from SourceClaimType claim.
2. Apply the regular expression from ReplacePattern with matching options from PatternOptions.
3. Replace the matched value with Replacement.
4. Save the new value in a new ClaimType claim.

The following is an example of configuring a **CreateFrom** action, which takes the value from the claim `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`, replaces all text after `@` inclusive with an empty string, and saves the new value in claim `username`:

```
{
  "ActionName": "CreateFrom",
  "ActionOptions": {
    "ClaimType": "username",
    "SourceClaimType": "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress",
    "ReplacePattern": "@.+",
    "Replacement": "",
    "PatternOptions": [ "IgnoreCase", "Singleline" ]
  }
}
```

Important

Only the full format of claim types is supported. To validate regular expressions with different values, it is recommended to use regex tools which are available online.

Validate Action

The **Validate** action enables users to verify a claim value by checking it against specified allow and deny regular expression patterns.

The **Validate** action configuration contains the following options:

- **ClaimType:** The type of claim value to be used.
- **AllowPattern:** The regular expression pattern to match allowed values. This option might not be presented if DenyPattern is set.



- **DenyPattern:** The regular expression pattern to match denied values. This option might not be presented if AllowPattern is set.
- **PatternOptions:** The list of regular expression options used to find a match.

Important

Supported PatternOptions values are described at <https://docs.microsoft.com/en-us/dotnet/api/system.text.regularexpressions.regexoptions#fields>.

The **Validate** action uses the following algorithm:

1. Get the value from the ClaimType claim.
2. Apply a regular expression from AllowPattern with the matching option from PatternOptions. If there is no match, an error is returned.
3. Apply a regular expression from DenyPattern with a matching option from PatternOptions. If there is a match, an error is returned.

The following is an example of configuring a **Validate** action that allows all values and denies a value if it is equal to any of the standard Aras Innovator administrators:

```
{
  "ActionName": "Validate",
  "ActionOptions": {
    "ClaimType": "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name",
    "AllowPattern": ".*",
    "DenyPattern": "^admin$|^root$|^vadmin$|^authadmin$|^esadmin$",
    "PatternOptions": [ "IgnoreCase", "Singleline" ]
  }
}
```

Important

Only the full format for claim types is supported.

Generic User Mapper Execution



Claim actions are executed in the same order as defined in a configuration. After executing all actions, the **Generic User Mapper** creates a username using the InnovatorUserNameFormat option.

An example follows:

```
{
  "Name": "Aras.OAuth.Server.Plugins.GenericUserMapper",
  "Enabled": true,
  "Options": [
    {
      "AuthenticationType": "Saml2-AzureAD",
      "InnovatorUserNameFormat": "{username}",
      "ClaimActions": [
        {
          "ActionName": "CreateFrom",
          "ActionOptions": {
            "ClaimType": "username",
            "SourceClaimType": "http://schemas.xmlsoap.org/ws/2005/05/identity/claims",
            "ReplacePattern": "@.+",
            "Replacement": "",
            "PatternOptions": [ "IgnoreCase", "Singleline" ]
          }
        },
        {
          "ActionName": "Validate",
          "ActionOptions": {
            "ClaimType": "username",
            "AllowPattern": ".+",
            "DenyPattern": "^admin$|^root$|^vadmin$|^authadmin$|^esadmin$",
            "PatternOptions": [ "IgnoreCase", "Singleline" ]
          }
        }
      ]
    }
  ]
}
```

The following is an example of **GenericUserMapper** plugin configuration for SAML2 authentication:



```
{
  "Name": "Aras.OAuth.Server.Plugins.GenericUserMapper",
  "Enabled": true,
  "Options": [
    {
      "AuthenticationType": "Saml2",
      "InnovatorUserNameFormat": "{uid}",
      "ClaimActions": [
        {
          // Validate uid (default Innovator users are denied).
          "ActionName": "Validate",
          "ActionOptions": {
            "ClaimType": "uid",
            "AllowPattern": ".+",
            "DenyPattern": "^admin$|^root$|^vadmin$|^authadmin$|^esadmin$",
            "PatternOptions": [ "IgnoreCase", "Singleline" ]
          }
        }
      ]
    }
  ]
}
```



Packaging

Aras Innovator is a low-code platform, which means users can add very little code to achieve outstanding results rapidly.

Users can also use configuration to express business rules, such as a **LifeCycle Map**.

When working directly with an instance of Aras Innovator, these changes are stored anonymously within the system and can reference any other items already in the system and vice versa.

Making such changes directly in a business-critical system serving users is not a good practice. As mentioned earlier, a central focus of DevOps is instilling the discipline of well-managed solution configurations, including change management and implementation.

The following sections explain how to export these changes into named **Packages**, define explicit dependencies, and commit the changes for proper configuration and version control.

This ensures that the Build System can replicate the swiftly accomplished interactive tasks, introducing configuration and version control discipline to the low-code product.



Summary of Modeling

The process of building a new application can be divided into two primary aspects:

- Data modeling (schema)
- Interaction (business rules, UX)

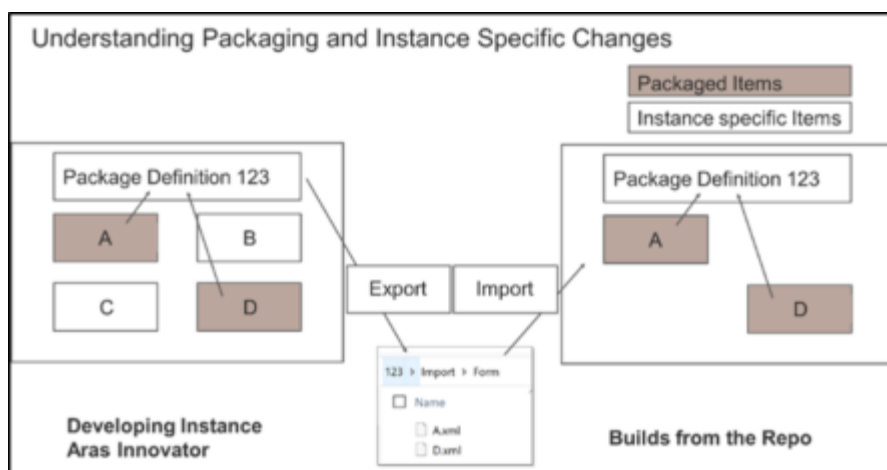
This is done on an existing system with standard products from Aras, Modules from 3rd Parties, or earlier work by the user's company for various reasons.

The new applications must consider all the existing work when modularizing, packaging, and building.

The diagrams below illustrate that the new application can be simple or ambitious.



The diagram below summarizes the impact of anonymous items in an instance and the effect of defining, exporting, and providing **Packages** to the Build System.



On the left, the user has four items (**A**, **B**, **C**, and **D**). **A** and **D** represent new or modified items properly packaged, exported, copied, and assigned to the change control system Git. **B** and **C** represent Innovator instance-specific items not intended for use in the next Build and, therefore, are not packaged.

The Build System then produces the instance on the right. It's important to observe that the Aras Innovator instance on the right excludes items **B** and **C**, showcasing the capability to dictate what DevOps builds. This capacity to specify what DevOps creates is a foundational element of utilizing DevOps.



Review of Packaging Scenarios

Case 1

Case 1 represents a straightforward addition of properties to standard classes for which the user must provide forms. Notice the specifications for the **Delta Extraction Tool**. By default, it is false.

Packaging Scenarios – Case 1

When your project needs to add properties to **existing** standard production components.
e.g., Document and Part

Package Definition PLM

Part

[get property]

Document

[get property]

- Keep items in their original AML package, this means, too, avoid moving items from Standard Aras Packages into custom packages
- Avoid modifications of Core packages

Enable the Delta Extraction tool by setting

- Use.Delta.Extraction.Tool parameter to true
- in AutomatedProcedures\Default.Settings.include

```

<!--
  Use.Delta.Extraction.Tool - flag to enable usage of Delta Extraction tool
  for AML-Packages. See description in Documentation folder.
-->
<property name="Use.Delta.Extraction.Tool" overwrite="false" value="false" />

```

```

<package name="com.myproject.PLM" path="myproject/pin/import">
  <dependson name="com.aras.innovator.solution.PLM" />
</package>

```

Adding properties and hence modifying core package could not be avoided or was more efficient

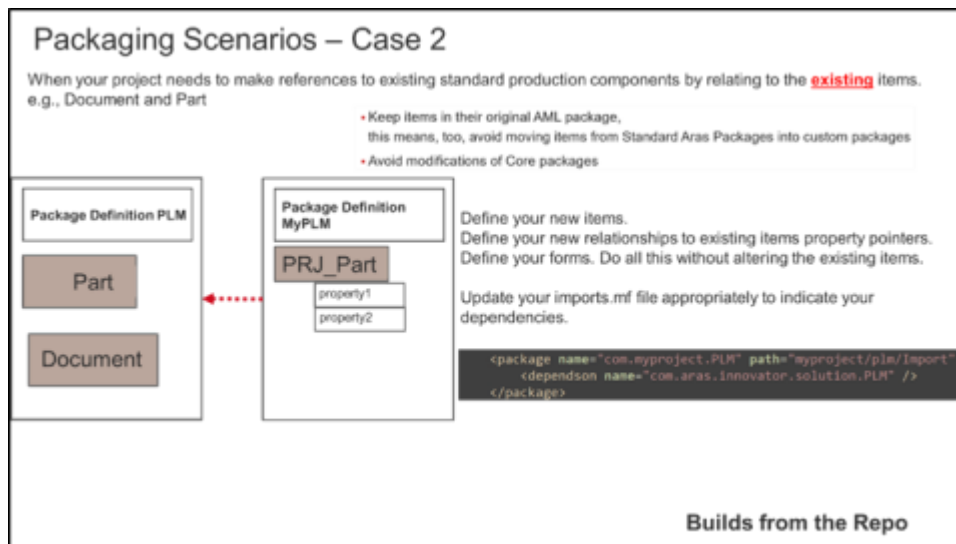
- Enable the delta extraction tool
- Make the change without moving items from the existing packages.

Builds from the Repo



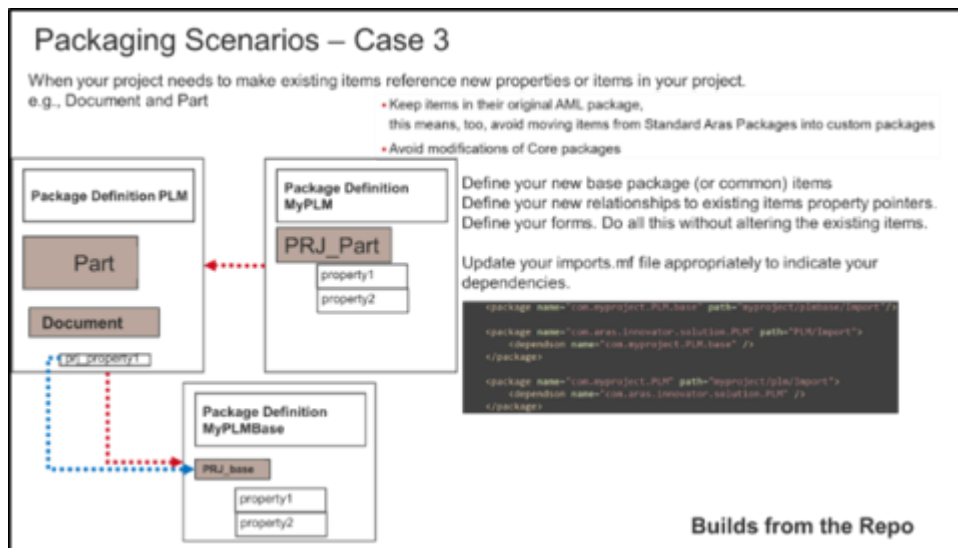
Case 2

Case 2 represents a situation in which the user needs a relationship between the new Item Types and the existing Item Types. The user must now define relationships and specify dependencies.



Case 3

In case 3, if the user has effectively introduced a circular dependency, it will not be recognized in the Aras Innovator instance utilized for development. Aras Innovator automatically resolves all the dependencies since all Items are already present. If packaging and modularization are not explicitly specified, the loading sequence can be misaligned and fail when using a Build System. To avoid such failures, explicitly specify **Packages** and their dependencies.



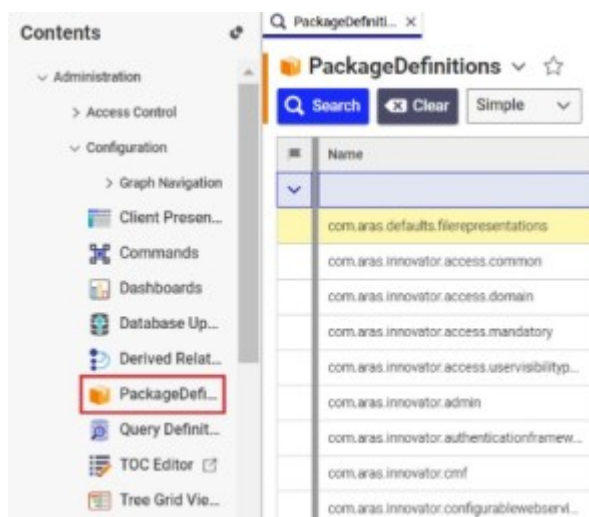
Packaging Tools and Methods

The Aras Innovator architecture is designed to customize standard Aras Solutions/Apps and build custom Solutions.

Solution Packaging is the mechanism that allows Solution Developers to register customizations in **Packages** so they can be extracted and transported to other Aras Innovator installations, such as from a development environment to a test environment.

The Items are organized into **Packages**. A **Package** element (identified by its GUID) cannot be added to multiple **Packages**. To facilitate identification, folder names should be aligned with the **Packages** they contain.

It is recommended to use packaging when creating items, lists, properties, etc., in an Aras Innovator instance since the packaging is mandatory for use in CI/CD. **Packages** can be exported and imported.



A newly installed Aras Innovator database contains **Package Definitions** of two types:

- **Core Packages:** These **Packages** define the basic structure of every Aras Innovator database, regardless of what solutions are used. Core **Packages** are not meant to be overwritten or customized.
- **Solution Packages:** These **Packages** define the elements that comprise the definition and functional rules of different custom data models created in the project context.

To learn about Standard Solution Packaging Tools, refer to the Standard Solution Packaging Tools Appendix.

The following section provides an illustration related to the topic discussed above.



Create and Manage New Application

Customization in Aras Innovator can consist of modifying existing applications and creating new ones to solve custom business needs. This section describes the main steps that will need to be taken.

A new application normally results in a new **AML Package** with its own **Package Definition**. The **Package Definition** contains all changes made to instances of Aras Innovator.

The process of creating a new **Package** requires several steps:

1. Create a new **Package**.
2. Export the **Package** to the file system, locate the results properly in the Repository directories, and update the manifest file to include not only the existing **Packages** but also the newly added ones.
3. Assign the new and modified files to the Git Repository with **Commit** or **Stage**, including the manifest file.
4. If a new **Package** is created, it is recommended to use Java naming conventions for **Packages** (lowercase and dots indicating a hierarchy) and align folder names with the **Packages** they contain.

Important

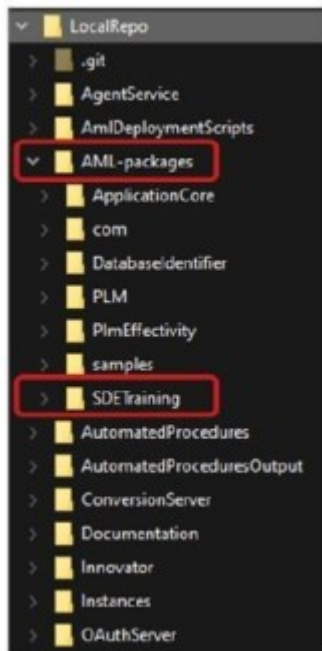
When a user updates any Repository sections, the new or modified files must be staged or committed to become part of the next Build.



Creating a Package Definition

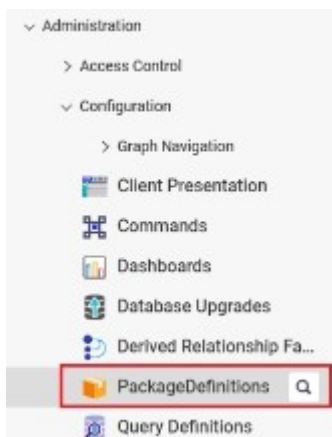
To set up a new application in Aras Innovator, the user must create a **Package** by creating a **Package Definition** with its dependencies and then exporting the **Package**.

In this case, the manifest file for the import must be adapted and needs to be committed or staged to the Version Control System.

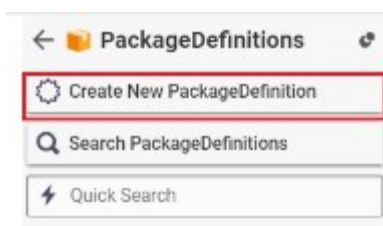


The following steps outline the process of creating a **Package Definition**:

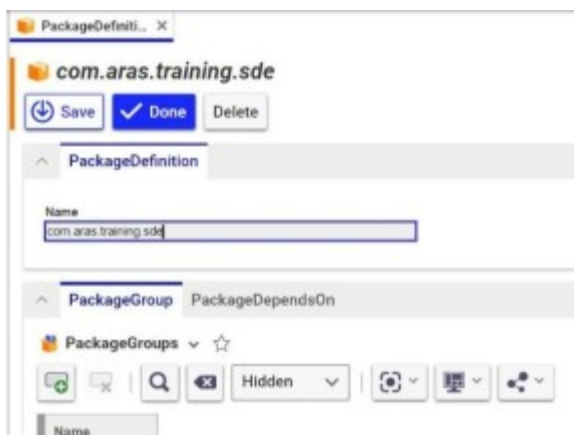
1. Login into the Aras Innovator instance.
2. From the **Table of Contents**, expand **Administration** and **Configuration**, then select **PackageDefinitions**.



3. Click **Create New PackageDefinition**.



4. On the **PackageDefinition** Form, enter the name of the **PackageDefinition**. For example, `com.aras.training.sde`.



5. Create a test Item and add it to the new **Package**.



Export Package and Update the Imports Manifest File

The Manifest file describes the list of **Packages** that will be imported during solution deployment and the dependencies between them. A Custom **Package** must be included in the manifest file when it is created and imported during project deployment.

The following steps outline the process of exporting and updating the Imports Manifest File:

1. Export the new **Package**.
2. Open the resulting manifest file. Copy the **Package Name** from the import.mf file. For example:
 1. `<package name="com.aras.training.sde" path="sde\Import" />`
3. Open the manifest file from the Local Repository: `C:\{Working Directory}\AML-packages`.
4. Paste the code copied in step 2 inside the import.mf file in the Local Repository.

For example:

```
<imports>
<package name="com.aras.training.sde" path="sde\Import">
<dependson name="com.aras.innovator.solution.ApplicationCore" />
</package>
</imports>
```

5. For cleanup, delete the test item, the only item in the **Package** `com.aras.training.sde`.



Confirming Manifest Changes in Version Control System

Verify the modifications made to the manifest file before committing them to the Repository. This process ensures the changes are accurate and aligned with the contributor's intentions.

It typically involves checking the status of the Version Control System (VCS), reviewing the modifications made to the manifest file using the Diff command specific to the VCS, and committing the changes if they are satisfactory. The exact steps and commands may vary depending on the VCS which is used.

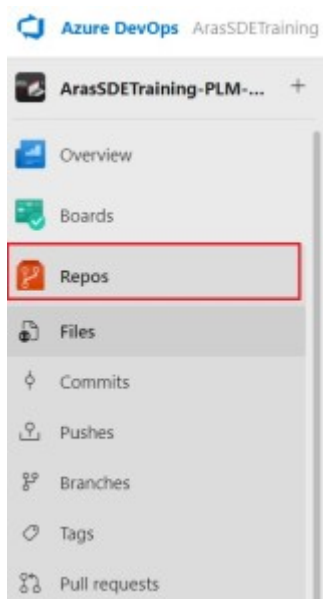


Update Repository to Use Single Package

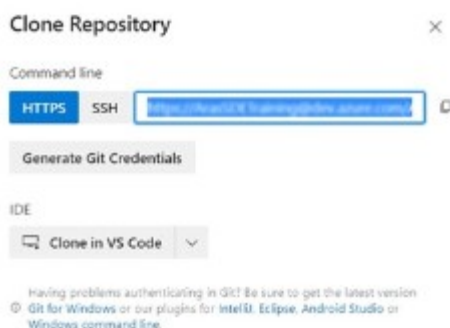
The latest release of Aras DevOps offers all essential Core **Packages** into a single NuGet **Package**. This simplifies the process of creating repositories and ensures that the Core **Packages** work seamlessly together.

The following steps outline the process of updating the Work Repository to use the Single **Packages**:

1. On **Azure DevOps**, select **Repos**.



2. Click **Clone** to clone the Repository that needs to be updated. The **Clone Repository** dialog box appears with the Repository's clone URL.



3. Copy the clone URL (HTTPS or SSH). An example URL:
https://dev.azure.com/{organization}/{project}/git/{repository}. Use any Version Control Tools required to clone the Repository.
4. Find the option to clone or create a new Repository in the Version Control Tool's interface.
5. Paste the clone URL copied from the Azure DevOps project.

Browse to the destination directory to clone the Repository.

Optional: Depending on the tool used, additional configuration options are available during the cloning process. This could include selecting **Branches**, specifying authentication credentials, or choosing the desired clone depth.

6. Click **Clone** within the Version Control Tool.
7. Open **Windows PowerShell** as Administrator.
8. Run **./repo-init.ps1** command for initializing the Repository with Initial **Packages**.
9. To identify the source name of the **Package** that points to the NuGet feed containing the Single **Package**, run the **Get-PackageSource** command.

```
PS C:\Projects\NewRepo\Work\Work.ruchira.walavalkar\Work> Get-PackageSource
```

Name	ProviderName	IsTrusted	Location
nuget.org	NuGet	False	https://api.nuget.org/v3/index.json
azure.artifacts.aras.com	NuGet	True	https://pkgs.dev.azure.com/ArasSDET/...
PSGallery	PowerShellGet	False	https://www.powershellgallery.com/api/v2
azure.artifacts.aras.com	PowerShellGet	True	https://pkgs.dev.azure.com/ArasSDET/...

For the correct update, a list of Single **Package Dependencies** is required. The list will be compared to the individual Packages specified in the AutomatedProcedures/tools/packages config file.

10. Remove-Module -Name "Aras.Devops" -Force
11. Install-Module -Name "Aras.Devops" -RequiredVersion <new version>
12. Import-Module -Name "Aras.Devops" -RequiredVersion <new version>
13. To get dependencies of the NuGet Package, run Find-Package -Name "Aras.SaaS.DevOpsFramework.Msi" -RequiredVersion "<single_package_version>" -Source <package_source_name> -IncludeDependencies | Select Name



```
PS C:\> Find-Package
>> -Name "Aras.Saas.DevOpsFramework.Msi"
>> -RequiredVersion "1.2.0.13823"
>> -Source azure.artifacts.aras.com
>> -IncludeDependencies | Select Name
Name
----
Aras.Saas.DevOpsFramework.Msi
Aras.DeltaExtraction.CommandLine
Aras.Deployment.Tool
Aras.Update.Cmd
Aras.ConsoleUpgrade
Aras.LanguageTool
Aras.Crt.Core
NAnt
NAnt.Contrib.Portable
MSBuild.Microsoft.VisualStudio...
Microsoft.Experimental.IO
Microsoft.Extensions.FileSyst...
Microsoft.Web.Xdt
Aras.Nant.Shim
Microsoft.PowerShell.5.Refere...
Aras.Crt.SeleniumTests
Aras.Crt.IntegrationTests
Aras.Crt.AzurePipeline
```

14. Navigate to the Local Repository and select the AutomatedProcedures file.
15. Click **Tools** and open packages.config file.
16. Locate the dependencies identified in step 10 and remove them from the packages.config file.

The following screenshot presents an example of the packages.config file with all Single **Package Dependencies**:

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Aras.IOM" version="14.0.15.38102" />
  <package id="Aras.Crt.InnovatorConfigs" version="14.0.15.38102" />
  <package id="Aras.DeltaExtraction.CommandLine" version="1.0.0.20" />
  <package id="Aras.Deployment.Tool" version="1.2.0.221" />
  <package id="Aras.Update.Cmd" version="1.22.1328" />
  <package id="Aras.ConsoleUpgrade" version="14.0.17.38577" />
  <package id="Aras.LanguageTool" version="14.0.17.38577" />
  <package id="Aras.Crt.Core" version="1.2.0.13732" />
  <package id="Aras.Nant.Shim" version="1.2.0.13732" />
</packages>
```

The following screenshot demonstrates the example of the packages.config file after the dependencies are removed:

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Aras.IOM" version="14.0.15.38102" />
  <package id="Aras.Crt.InnovatorConfigs" version="14.0.15.38102" />
  <package id="Aras.Saas.DevOpsFramework.Msi" version="1.2.0.13823" />
</packages>
```

17. To restore the new **Packages** list with dependencies from NuGet, open Windows PowerShell as Administration and run the following command: Restore-ArasDevopsPackages - WorkRepository "<path_to_work_repository>"

Add the Work Repository path as a parameter. Ensure that the command is executed successfully and no errors appear in the console.



1. Important

The **init.ps1**, **update.ps1** and **cleanup.ps1** scripts are executed (if present) only for **Packages** with names starting with Aras*.

18. Verify any changes to NuGet **Packages** by checking the output of **Restore-ArasDevopsPackages -WorkRepository "<path_to_work_repository>"** command on Window PowerShell console.

This display indicates which **Packages** have been added, updated, or deleted in the Work Repository:

```
NuGet packages added to the work repository as a result of 'Restore-ArasDevopsPackages' call:
Name                                     Version
----                                     -
Aras.Crt.AzurePipeline                  1.2.0.13823
Aras.Crt.IntegrationTests                1.37.3838
Aras.Crt.SeleniumTests                  1.37.3838
Aras.SaaS.DevOpsFramework.Msi           1.2.0.13823

NuGet packages updated in the work repository as a result of 'Restore-ArasDevopsPackages' call:
Name           NewVersion OldVersion
----           -
Aras.Crt.Core  1.2.0.13823 1.2.0.13732
Aras.Nant.Shim 1.2.0.13823 1.2.0.13732
```

19. Commit the changes using the required Version Control System.



Update Repository to New BDS Version

The update process depends on your current BDS version. Follow the flow that matches your scenario:

1. Updating from BDS 1.11 to higher versions.
2. Updating from older versions to BDS 1.11 or higher.



Updating from BDS 1.11 to higher Version

Local Development Environment initialization

1. Clone the repository and check out the branch (See [Cloning Repo to Local Working Directory](#))
2. Open Windows PowerShell as Administrator.
3. Run ./repo-init.ps1 command for initializing the Repository with initial Packages.

Run Update-ArasDevops Framework Version cmdlet

1. **Open** Windows PowerShell as **Administration** and run the following command:

```
Update-ArasDevopsFrameworkVersion -DevopsFrameworkReleaseVersion "{BDS simple version}" -WorkRepositoryPath {Work Repository path}
```

{Work Repository path} - should be replaced with the Work Repository path.

{BDS simple version} - should be replaced with the new BDS version you want to use, for example:

```
Update-ArasDevopsFrameworkVersion -DevopsFrameworkReleaseVersion "1.11" -WorkRepositoryPath {Work Repository path}
```

2. **Verify Success:**

Ensure the command is executed successfully and no errors appear in the console.

Confirm a new commit is added, containing at least:

- An **updated single package** version in AutomatedProcedures\tools\packages.config
- An **updated DevOps.Framework.Version** in AutomatedProcedures\Default.Settings.include



Updating from older versions to BDS 1.11 or higher

The following steps outline the process of updating the Work Repository to use the new version of DevOps package:

1. Local Development Environment initialization.
2. Determine the version of Aras DevOps to install.
3. Update version of the Aras DevOps PowerShell module.
4. Run Update-ArasDevOpsFrameworkVersion cmdlet.

Local Development Environment initialization

1. Clone the repository and check out the branch (See [Cloning Repo to Local Working Directory](#))
2. Open **Windows PowerShell** as Administrator.
3. Run **./repo-init.ps1** command for initializing the Repository with initial **Packages**

Determine the version of Aras DevOps to install

1. To identify the source name of the DevOps package that points to the NuGet feed containing the DevOps package, run the Get-PackageSource command.

```
PS C:\Projects\NewRepo\Work\Work.ruchira.walavalkar\Work> Get-PackageSource
```

Name	ProviderName	IsTrusted	Location
nuget.org	NuGet	False	https://api.nuget.org/v3/index.json
azure.artifacts.aras.com	NuGet	True	https://pkgs.dev.azure.com/ArasSDETr
PSGallery	PowerShellGet	False	https://www.powershellgallery.com/ap
azure.artifacts.aras.com	PowerShellGet	True	https://pkgs.dev.azure.com/ArasSDETr

2. Run of the following commands to find last released version of DevOps package:

```
Find-Package -Source azure.artifacts.aras.com -Name "{package_name}" -Provi
```

where "{package_name}" can be Aras.SaaS.DevOpsFramework.Msi or Aras.SaaS.DevOpsFramework.Aicd or Aras.DevOpsFramework.SaaS3 based on current name of the package used in AutomatedProcedures\Tools\packages.config



As a result, command will return the latest version:

```
PS C:\Users\vignatyuk> Find-Package -Source azure.artifacts.aras.com -Name "Aras.SaaS.DevOpsFramework.Msi" -ProviderName
NuGet -MinimumVersion 1 -MaximumVersion 9999

Name                               Version          Source            Summary
----                               -
Aras.SaaS.DevOpsFramework.Msi     1.3.0.15043     azure.artifacts.aras.com  This package contains references to...
Aras.SaaS.DevOpsFramework.Msi     1.7.0.20441     azure.artifacts.aras.com  This package contains references to...
```

Note down received version and use it during update instead of "{new version}".

Update version of the Aras DevOps PowerShell module

1. **Remove the current installed Aras DevOps Module:** Run the following command to remove old module in the Windows PowerShell :

```
Remove-Module -Name "Aras.Devops" -Force
```

2. **Install new version of Aras DevOps Module:** It is very important to use the relevant version of Aras DevOps module as there might be changes inside commands that can be critical for update, so run the following command to install the new module in the Windows PowerShell:

```
Install-Module -Name "Aras.Devops" -RequiredVersion "{new version}"
```

where "{new version}" is version received during **Determine the version of Aras DevOps to install** step of this chapter.

3. **Import new version of Aras DevOps Module:** Run the following command to import new module in the Windows PowerShell:

```
Import-Module -Name "Aras.Devops" -RequiredVersion "{new version}"
```

where "{new version}" is version received during **Determine the version of Aras DevOps to install** step of this chapter.

Run Update-ArasDevOpsFrameworkVersion cmdlet

1. Open Windows PowerShell as Administration and run the following command:



```
Update-ArasDevopsFrameworkVersion -DevopsFrameworkReleaseVersion "{BDS simple version}"
```

{Work Repository path} - should be replaced with the Work Repository path.

{BDS simple version} - should be replaced with the new BDS version you want to use, for example:

```
20. Update-ArasDevopsFrameworkVersion -DevopsFrameworkReleaseVersion "1.11"
```

2. Verify Success:

Ensure the command is executed successfully and no errors appear in the console.

Confirm a new commit is added, containing at least:

- An **updated single package** version in AutomatedProcedures\tools\packages.config
- An **updated DevOps.Framework.Version** in AutomatedProcedures\Default.Settings.include



Change Management and Implementation

Aras enforces a strict Solution Configuration Management discipline. The solution includes:

- **Standard Aras Innovator Platform:** The release is created regularly by Aras Engineering and tracked with versioning (such as Aras Innovator Release 32; refer to the support matrix in the Subscriber Portal).
- **Standard Aras Innovator Applications:** Users can include them with the base Aras Innovator platform.
- **Configuration and Customizations:** Specified by the customer and implemented by the customer and or customer's agents such as Aras Solution Delivery Services and 3rd Party Systems Integrators. These include:
 - Workflow configurations
 - Reports
 - Configurations such as adding new properties to Items
 - Forms
 - Integrations to various other systems
 - Enhancements (customizations for specific purposes)

The project manages all the above to ensure comprehensive Solution Configuration Management. Aras Cloud policy requires a project to provide the following approvals to get a specific solution configuration into production or modify the solution:

- **Staging Environment:** The Staging Environment allows users to complete final preparations and checks on a thoroughly tested deployment before it goes into production. It closely mirrors the production environment and is the last phase before deployment. The project team initiates requests for provisioning and de-provisioning staging environments for production.
- **System Qualification Approval:** The system satisfies requirements as tested in the SIT environment.
- **Functional Qualification Approval:** The system configuration has been completed and is acceptable to the user community as tested in the UAT environment.
- **Data Qualification Approval:** The system contains the required initial data.
- **Production Qualification Approval:** This is approval to go live and an invitation to the user community to use the system. Testing for this approval is conducted in the preproduction environment. Solution configuration management pertains to the initial implementation and



subsequent improvements, bug fixes, and any modifications that change the system's configuration.



Production Countdown Sequence

Aras understands that the need for flexibility during development will interfere with the project's chosen practices.

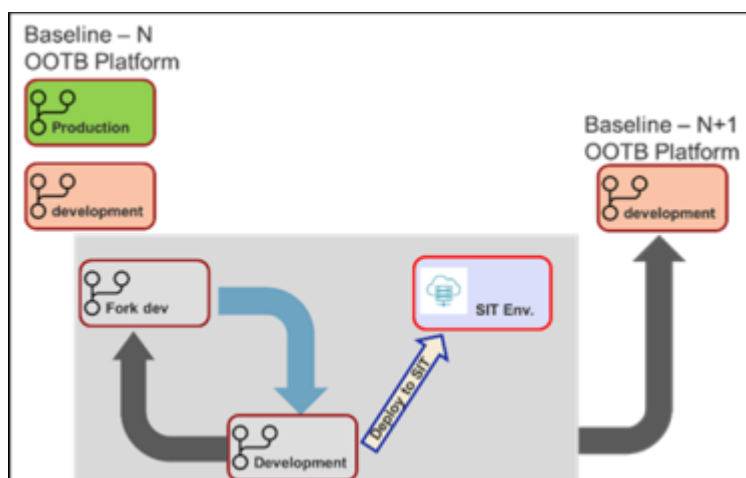
However, during the countdown sequence to production deployment, Aras requires a protocol to ensure a smooth transition and secure the necessary approvals.

Aras focuses on the following **Branches**:

- Development
- SIT
- UAT
- Staging
- Production

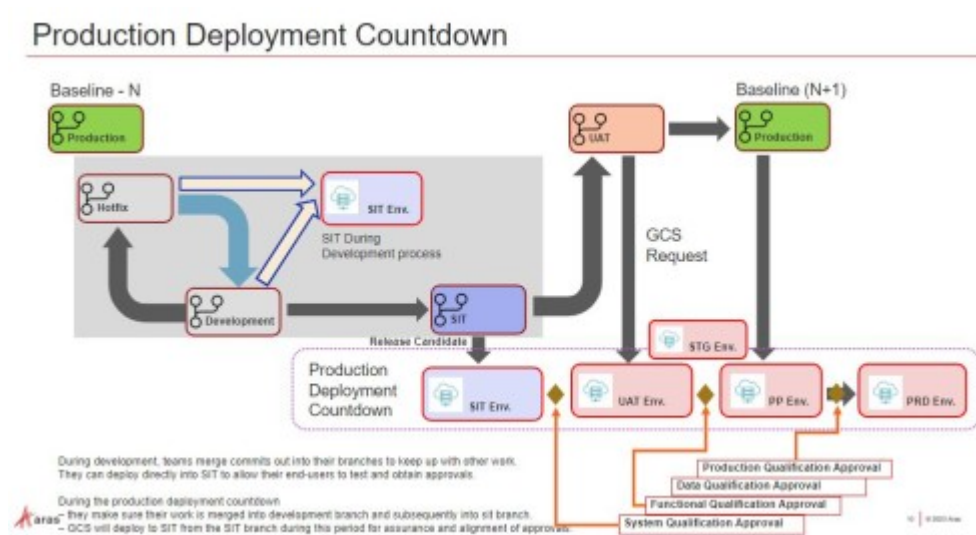
As part of the production deployment sequence, Aras mandates that the project team integrate their release candidate from the Development **Branch** into the “sit” **Branch** before deploying it in the SIT environment.

During deployment, the project team can deploy from any Work (Team) Repository **Branch** to the SIT environment to support ongoing testing.



During the countdown sequence, the change implementation policy requires the project team to deploy to the SIT environment only from the “sit” **Branch**. The practice of deploying to SIT only from

the "sit" **Branch** during the countdown sequence helps ensure alignment. The customer is required to identify the Build for which they approve for System Qualification.



Aras mandates that the project team obtains a System Qualification Approval (SQA) from the client to conclude SIT testing. This approval signifies that the client is content with the entire solution. All necessary integrations, SSO connections, CAD, and Office connectors have been established as needed.

Note that the project team may have identified a release candidate, deployed it to SIT, collected feedback, and performed remediation to obtain SQA. Aras does not dictate the number of cycles; the customer just provides SQA before deployment to UAT.

Once the SQA is obtained, the project team immediately asks Aras to initiate the suitable Build deployment in the UAT environment. The project team should prepare and deploy to the UAT environment from the "uat" **Branch**. In collaboration with the client, the project team facilitates a system review by the end-users, leading towards the Function Qualification Approval (FQA). The client must provide FQA before Production Qualification Approval (PQA).

In projects involving data migration, the project team must secure Data Qualification Approval (DQA). To obtain DQA, the project team must request production environment provisioning and necessary endpoints to run the data import. Once the project team has imported the data, the team must ask the customer to review the data and approve data quality.



The project team must request the UAT, staging, and production environments with the customers' approval. The staging environment is used for importing data from other systems and as pre-production to perform final testing to secure PQA.



Branding Customization

Branding customizations enable contributors to use their own logos and banners for Aras Innovator.



Splash Screen

The following steps outline the process to change the Aras Innovator splash screen:

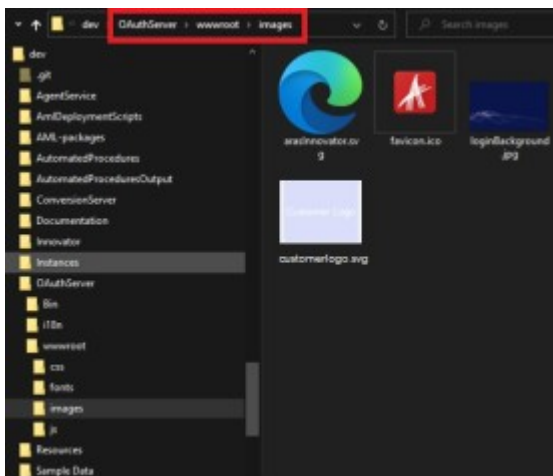
1. Select a background Image and obtain the customer's logo to be applied to the splash screen.

Important

Contributors need a tool to manipulate SVG images. Use any required tool.

2. In the following directory, add the Background image and Customer logo obtained from step 1:

```
C:\{working directory}\Instances\dev\OAuthServer\wwwroot\images
```



3. From C:\{working directory}\TransformationsOfConfigFiles\ directory, modify the OAuthServer.config file. Notice the location of the file to edit. It is important to do this in the **Transforms** folder, as shown in the screenshot below:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <configuration xmlns:xsi="http://schemas.microsoft.com/XML-Document-Transform" >
3
4   <oauth configSource="Oauth.config"></oauth>
5
6   <configuration>
7     <section name="oauth" type="Aras.OAuth.Configuration.OAuthSection, Aras.OAuth.Configuration"></section>
8     <sectionGroup name="Aras">
9       <sectionGroup name="Web">
10        <section name="RequestProvider" type="
11          Aras.Web.Configuration.RequestProviderConfigurationSection, Aras.Web"></section>
12        </sectionGroup>
13      </sectionGroup>
14    </configuration>
15
16    <add key="InnovatorServerUrl" value="$ADT_INNOVATOR_URL/Server/innovatorServer.aspx"
17      xsi:transform="Replace" </add>
18
19    <!--
20     'AuthenticationAdmin' is a name of user to access InnovatorServer.
21    -->
22    <add key="AuthenticationAdmin" value="authadmin"</add>
23
24    <!--
25     UI Customization
26    -->
27    <add key="ProductName" value="Aras Innovator"</add>
28    <add key="OauthInstallationLogoImage" value="Aras.Innovator"</add>
29    <add key="LogoUrl" value="/images/ArasInnovator.png"</add>
30    <add key="BackgroundImageUrl" value="/images/loginBackground.jpg"</add>
31
32  </configuration>
33 </oauth>
34
35   <add key="RequestProvider"
36     </add>
37
38 </configuration>
39 </configuration>
40
41 <add key="LogoUrl" value="$ADT_INNOVATOR_INSTALLATIONPATH/OAuthServer/wwwroot/images/customerLogo.svg" xsi:transform="Replace" </add>
42 <add key="BackgroundImageUrl" value="$ADT_OAUTH_INSTALLATIONPATH/images/customerLoginBackground.jpg" xsi:transform="Replace" </add>

```

Important

It is best to use the OAuth installation path, ADT_OAUTH_INSTALLATIONPATH, directly as it may be installed differently.

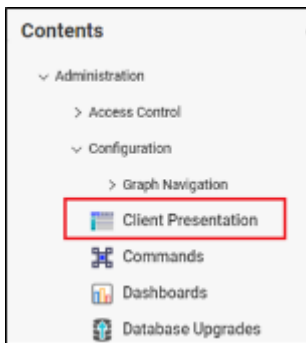
4. Stage the changes before running the Build scripts.
5. Run the **./BuildAndDeploy.ps1** script to rebuild Aras Innovator.
6. Notice the changes to the login screen.



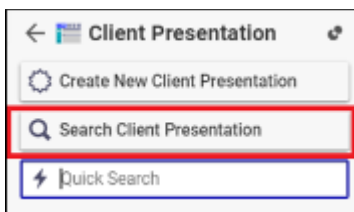
Change Banner

The following steps outline the process of updating the banner logo:

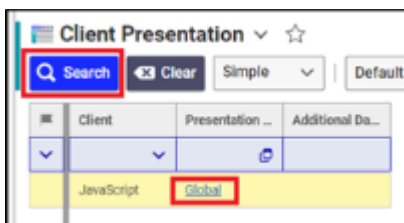
1. Navigate to the existing image.
2. Login into the Aras Innovator instance.
3. From the **Table of Contents**, expand **Administration** and **Configuration**, then select **Client Presentation**.



4. From **Client Presentation** Table of Contents, click **Search Client Presentation**.



5. On **Client Presentation** Form, click the **Search** button then click the **Global** link.



6. In **Command Bar Section** Tab, select the row with the **Main Window Header** label.

Global ☆ □

Command Bar Section cui_PresentConfigWinSection Presentation Command

CommandBarSection ☆

#	Classification	Name	Builder Meth...	Label	Description	Additional De...	Location [...]	sort_Lander	For Identity [...]	For Classifa...
	Data Model	com.aras.inno...	extendedManipViewerToolBar			{'buttonstyle'...	extendedMdr...	5376	World	
	Method	com.aras.inno...	CuiMdrView...				PopUpMenu1...	5760	World	
	Method	com.aras.inno...	CuiMdrView...				PopUpMenu1...	6144	World	
	Method	com.aras.inno...	CuiMdrView...				PopUpMenu1...	6028	World	
	Data Model	com.aras.inno...	Default TOV Toolbar				TOV_Toolbar	6656	World	
	Data Model	com.aras.inno...	TOV Context Menu Default				TOV_Context	6764	World	
	Data Model	com.aras.inno...	Main Window Header				MainWindow...	6912	World	
	Data Model	com.aras.inno...	Effectivity Expression Item Grid T...				eff_Levexpre...	6912	World	
	Data Model	com.aras.inno...					MainWindow...	7040	World	
	Method	com.aras.inno...	cui_default...				TOV	7168	World	
	Data Model	itemviewer.ite...	Item View Default Command Bar				CommandBar...	7296	World	

7. Select `com.aras.innovator.cui_default.mwh_header_logo`.

Client Presenta... X Global X com.aras.innov... X com.aras.innov... X Client Presenta... X

com.aras.innovator.cui_default.mwh_header ☆ □

Location: MainWindowHeader

Classification: Data Model

Label: Main Window Header

Description:

Command Bar Item Command Bar Section Dependency

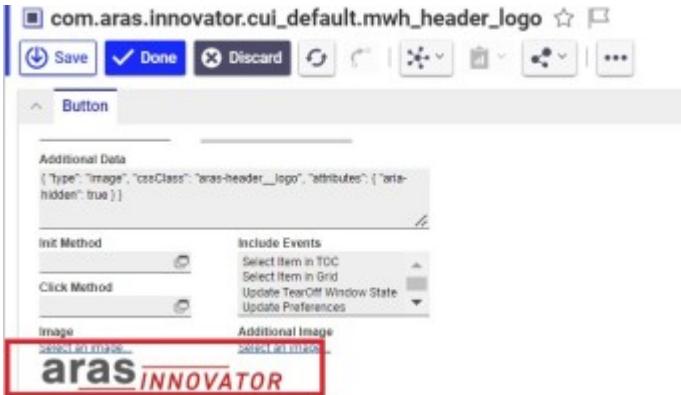
CommandBarItem ☆

#	ItemType	Name	Additional ...	Init Method [...]	Sort Order	Action	Alternate [...]	For Identity [...]	For Classifa...
	Button	com.aras.innovator.cui_default.mwh_header_resizeH...	{'clickEventT...		18	Add		World	
	Button	com.aras.innovator.cui_default.mwh_header_logo	{'type': 'img...		32	Add		World	
	Button	com.aras.innovator.cui_default.mwh_header_resizeV...	{'right': true, '...	cui_default.r...	64	Add		World	
	Button	com.aras.innovator.cui_default.mwh_header_resizeH...	{'right': true}	cui_default.r...	86	Add		World	
	Button	com.aras.innovator.cui_default.mwh_header_notify...	{'right': true, '...	cui_default.r...	128	Add		World	

< Prev Next > Page: 1 of 1 | 5 Results

8. Notice the existing image. The height is significant for adjusting the customer's logo to fit while maintaining conformity with the customer's branding guidelines.





9. On the local machine, add the required Customer Logo in the following directory:
C:\{working directory}\Instances\dev\OAuthServer\wwwroot\images
The image should be in SVG format.
10. Navigate to the following folder: C:\{Working Directory}\AML-packages\com\aras\innovator\cui_default\CommandBarButton
11. Open the com.aras.innovator.cui_default.mwh_header_logo file.
12. In the com.aras.innovator.cui_default.mwh_header_logo file, do the following:
 - o Change the action attribute from add to edit.
 - o Add the CustomerLogo.svg in the image **Tag**. Ensure that the path is correct.

```
<XML>
<Item type="CommandBarButton" id="83E579F86F0848508927F070D099091E" action="add">
<additional_data { "type": "image", "cssClass": "aras-header_logo", "attributes": { "aria-hidden": true } }</additional_data>
<image>./images/CustomerLogo.svg</image>
<name>com.aras.innovator.cui_default.mwh_header_logo</name>
</Item>
</XML>
```

13. Run the **./BuildAndDeploy.ps1** script to rebuild the Aras Innovator instance. The Banner will now be the new image.



Appendix I: Local Development Environment Setup

For a contributor to make changes and test them locally, the environment must support the various deployment elements, such as a development database server.

This section outlines the essential requirements for setting up a local development environment to make changes. It covers the necessary software, tools, and configurations to create a productive and efficient development environment. By following the points below, users can ensure that their local development setup meets the prerequisites for seamless software development and testing.

The preparation of this environment is primarily automated. However, the following tools must be present before running the scripts:

- **Windows PowerShell:** The minimum required version is 5.1, but the latest version should be used.
- **Chocolatey:** <https://chocolatey.org/install> (version 0.11.0)
- **choco install gitextensions:** <https://community.chocolatey.org/packages/gitextensions>



Installing Windows Powershell

If Windows PowerShell is not already installed on the system, set up the most recent version.

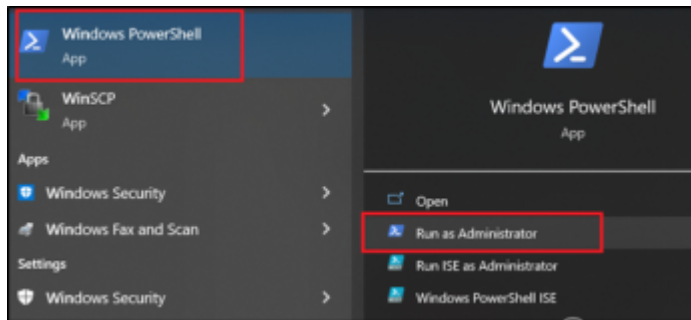
- To install, visit <https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows?view=powershell-7.3> .



Installing Chocolatey using Windows PowerShell

The following steps outline the process to install the Chocolatey tool using Windows PowerShell:

1. Open Windows PowerShell and run as Administrator.

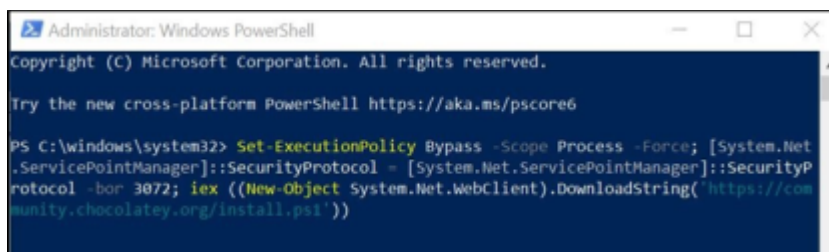


With PowerShell, please ensure that **Get-ExecutionPolicy** is not **Restricted**. Using **Bypass** is recommended to bypass the policy to get things installed, or **AllSigned** for more security.

Run **Get-ExecutionPolicy**. If it returns **Restricted**, then run **Set-ExecutionPolicy AllSigned** or **Set-ExecutionPolicy Bypass -Scope Process**.

2. Copy the following command and paste into PowerShell:

```
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```



3. Wait a few seconds for the command to complete.
4. For more information, please visit <https://chocolatey.org/install> .



Installing Git

The recommended minimum Git version is 2.23.0, but use the latest version of Git. Install Git using either of the two methods below:

- Install Git from the official site, <https://git-scm.com/downloads> .
- Using Chocolatey Package Manager. In PowerShell, type the following command:

choco upgrade -y git

```
PS C:\windows\system32> choco upgrade -y git
Chocolatey v1.3.1
Upgrading the following packages:
git
By upgrading, you accept licenses for the packages.
git is not installed. Installing...
Progress: Downloading git.install 2.40.0... 100%
Progress: Downloading git.install 2.40.0... 100%
Progress: Downloading chocolatey-core.extension 1.4.0... 100%
Progress: Downloading chocolatey-core.extension 1.4.0... 100%
Progress: Downloading chocolatey-compatibility.extension 1.0.0... 100%
Progress: Downloading chocolatey-compatibility.extension 1.0.0... 100%
Progress: Downloading git 2.40.0... 100%
Progress: Downloading git 2.40.0... 100%

chocolatey-compatibility.extension v1.0.0 [Approved]
chocolatey-compatibility.extension package files upgrade completed. Performing other installation steps.
Installed/updated chocolatey-compatibility extensions.
The upgrade of chocolatey-compatibility.extension was successful.
Software installed to 'C:\ProgramData\chocolatey\extensions\chocolatey-compatibility'

chocolatey-core.extension v1.4.0 [Approved]
chocolatey-core.extension package files upgrade completed. Performing other installation steps.
Installed/updated chocolatey-core extensions.
The upgrade of chocolatey-core.extension was successful.
Software installed to 'C:\ProgramData\chocolatey\extensions\chocolatey-core'

git.install v2.40.0 [Approved]
git.install package files upgrade completed. Performing other installation steps.
Using Git LFS
Installing 64-bit git.install...
git.install has been installed.
git.install installed to 'C:\Program Files\Git'
git.install can be automatically uninstalled.
Environment Vars (like PATH) have changed. Close/reopen your shell to
see the changes (or in powershell/cmd.exe just type 'refreshenv').
The upgrade of git.install was successful.
Software installed to 'C:\Program Files\Git\'

git v2.40.0 [Approved]
git package files upgrade completed. Performing other installation steps.
The upgrade of git was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\git'

Chocolatey upgraded 4/4 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
PS C:\windows\system32>
```



Installing Azure CLI

To install Azure CLI, visit <https://learn.microsoft.com/en-us/cli/azure/install-azure-cli> .

To install the Azure DevOps extension for Azure CLI, visit <https://learn.microsoft.com/en-us/azure/devops/cli/?view=azure-devops> .



Required Specifications

The following specifications are required for Local Development Environments (LDE):

- MSSQL Server 2019 or 2022
The contained database authentication option must be enabled as shown:

```
sp_configure 'contained database authentication', 1;  
GO  
RECONFIGURE;  
GO
```
- SSMS SQL Server Management Studio 2019 or 2022
The contained database authentication option can also be enabled in SSMS SQL Server Management Studio.
- MS IIS Server
- File diff/merge tool (e.g., Kdiff3)
- Git 2.23 or later (use the latest version).
- Git Extensions
- Visual Studio Community (Professional) Edition or above 2019 and 2022
- Visual Studio Code 1.77 or later



Appendix II: Standard Solution Packaging Tools

The **Package Import Export Utilities** are provided with every Aras Innovator release.



Export.exe

This tool is part of the **Package Import Export Utilities**. The **Export** Tool allows users to select **Package Elements** to export to the file system as XML. These **Package Elements** can be exported individually, as part of a **Package Group**, or as part of a **Package Definition**.



Import.exe

This tool is part of the **Package Import Export Utilities**. The **Import** Tool allows users to select predefined manifest files and import the corresponding **Package AMLs** into a database. In a CI/CD environment, users do not have to do any imports manually. These will be automated steps triggered by the relevant CI/CD automation.



Consoleupgrade.exe

This tool is part of the **Package Import Export Utilities**. The **Console Upgrade** Tool is a command-line version of both the **Export** Tool and **Import** Tool described above. The command-line parameters can be found by typing `/?` as the command-line parameter. In a CI/CD process, this Tool is part of the deployment automation and does not have to be used manually.



Appendix III: Adding Applications to a Project

When a project starts, it may wish to use several applications and potentially language Packages. This section explains how to add an application to the Project Repository. For reference, the following steps provide instructions for installing a Simulation Management (SM) application. The steps might differ depending on the user's application to integrate with Aras Innovator.

The following steps outline the process of adding an SM application:

1. Determine the required version for application installation by comparing the current version of the Aras Innovator with the specified name of the Application to be installed in the [Aras Support Matrix](#).

End of Life	Product Packaging	Program Management	Requirements Engineering	Systems Architecture	Simulation Management	Configuration Engineering	Technical Documentation	Quality Management System	Process Quality Working	Manufacturing Release Planning	Digital Pan-View	Other Capabilities	Enterprise Search	BI Analytics	Process Engine
Release 14 (Build 14.0.0.30340)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)
Release 15 (Build 14.0.1.32287)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)
Release 16 (Build 14.0.2.33840)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)
Release 17 (Build 14.0.2.35586)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)
Release 18 (Build 14.0.4.36640)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)
Release 19 (Build 14.0.4.38717)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)
Release 20 (Build 14.0.4.38717)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)	Release 14 (14.0.1)

Certified - QA and Acceptance Testing Completed
 Supported - Not Certified - Aras Provides Full Support
 End of Life - This product has reached end of life

2. Download the Simulation Management CD image from the Aras FTP site and unzip the file on the local computer.
3. Copy the Aras Innovator folder to the Repository overwriting the existing \Innovator folder and all its contents.
4. Copy the content from the Import folder and paste it into AML Packages folder.
5. Update the Import Manifest file. The following line is present in the Import.mf file:

```
<package name="com.aras.innovator.solution.SM" path="SM\Import" />
```

```
<?xml version="1.0" encoding="utf-8"?>
<imports>
  <package name="com.aras.innovator.solution.SM" path="SM\Import" />
</imports>
```

6. Commit the changes with an appropriate message.
7. Run the **.\BuildAndDeploy.ps1** script to ensure that updates build successfully.



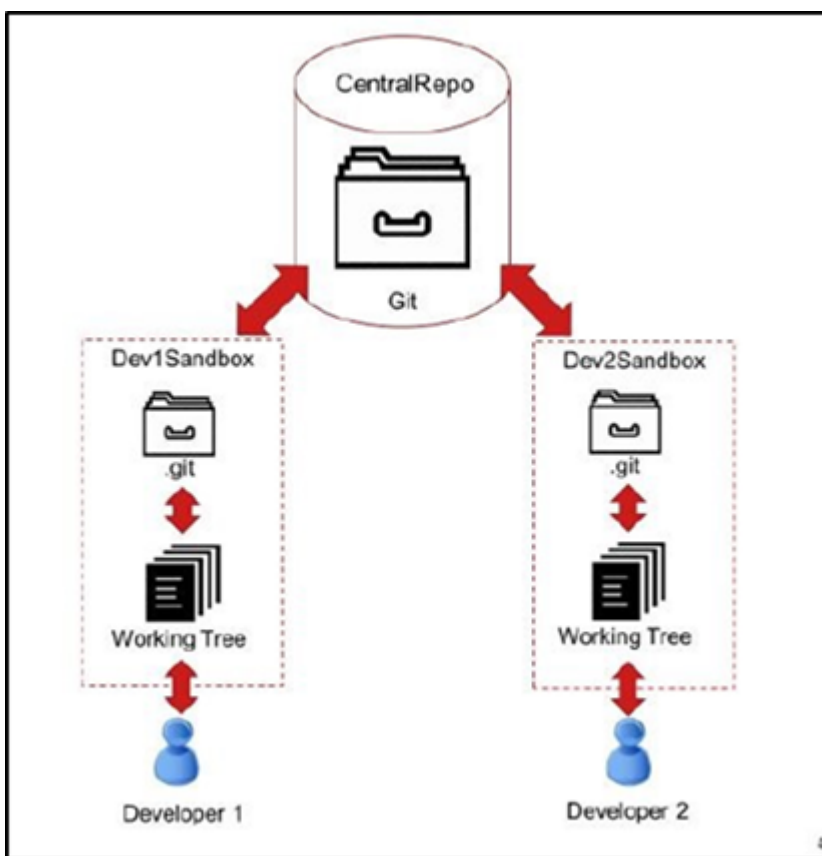
8. After successful execution of the **.\BuildAndDeploy.ps1** script, commit and push the changes.
9. Once changes are pushed, create a **Pull request (PR)** and merge them. For instructions on creating a **PR**, refer to the Creating a Pull Request section.
10. The Continuous Integration **Pipeline** will execute successfully after the merge.
11. Create a **Tag** on the latest **Commit** to generate a new **Baseline**. For instructions, refer to the Generating New Baseline section.



Appendix IV: Using a Shared Repository and Merging Conflicts

Use Shared Repository

Multiple developers use a Central (also called Remote) Repository, allowing each authorized developer to push and fetch changes from a single source. Although each developer maintains a local copy of the Repository on their machine, the Remote Repository collects the changes from everyone on the team. Each developer is responsible for updating or fetching changes from the Remote Repository regularly.



Connect to Shared Repository

Adding a Remote Reference allows developers to establish a connection between their Local Repository and a Remote Repository. It enables developers to push their changes to the Remote Repository, fetch updates made by others, and synchronize their work with the rest of the team.

Push Changes to Shared Repository

Pushing changes to a Remote Repository allows the developer to send the local **Commits** and updates to the Remote Repository, making them accessible to others working on the project.

Fetch Changes from Shared Repository

Fetching changes from the Shared Repository ensures that the developers have the most recent code updates, allowing them to incorporate the changes into the Local **Branch** and maintain a synchronized codebase.

Managing File Conflicts

A merge conflict occurs when two or more developers make conflicting changes to the same part of a file. For example, if Developer 1 modifies a function while Developer 2 modifies the same function, the Version Control System may be unable to determine which changes should take precedence automatically.

Resolving Merged Conflicts

When working with Git, conflicts can be encountered when merging two **Branches**. This occurs when Git cannot automatically merge changes made to the same lines of code in both **Branches**. The following steps outline the process to resolve merge conflicts:

1. Open the file(s) that have conflicts.
2. Look for the conflict markers in the file(s), which looks like below screenshot:

```
CSS

<<<<<<< HEAD
code from the current branch
=====
code from the branch being merged
>>>>>> branch-name
```

3. Edit the code to reflect the changes to be kept.
4. Remove the conflict markers from the file(s).



5. Save the changes to the file(s).
6. Add the modified file(s) to the staging area.
7. Commit the changes.

8. Push the changes to the Remote Repository.

If using a Merge tool like VS Code or SourceTree, it should have a graphical interface to help resolve conflicts more easily. Launch the tool and follow its instructions to resolve conflicts.

It's important to note that resolving merge conflicts can be a complex and time-consuming process, especially if there are many conflicts to resolve. It's always a good idea to carefully review and test changes after resolving conflicts to ensure they work as intended.

Sharing Changes with the Remote Repository

Once the changes are made to the Local Repository and any Merge conflicts resolved, developers need to share those changes with the Remote Repository. Here are a few Developer responsibilities:

- **Commit Changes:** It is essential to commit changes locally. Make frequent well-documented **Commits**.
- **Fetch the Latest Changes:** Fetching the latest changes from the Remote Repository is good practice. This ensures that users have the most up-to-date version of the codebase and reduces the chances of conflicts.
- **Rebase:** Use Rebase to update Local Repository with Remote Repository changes.
- **Push Changes:** Push Local Repository changes to the Remote Repository frequently.
- **Verify Changes:** After pushing the changes, it is essential to verify that they have been successfully shared with the Remote Repository.

Using Stash

Stash allows developers to store the modifications, including both staged and un-staged changes, in a safe place to switch to a clean working directory without losing their work. It provides temporary storage for their changes, enabling them to seamlessly move between tasks or **Branches**. Stashing is particularly useful when developers are not yet ready to commit their changes or when they want to work on a different task without the interference of their current modifications.

The following points will explain the process of using stash effectively:



- **Stashing Changes:** A common use case for stashing changes is when users make changes to the working directory that are not yet committed but need to fetch changes from another developer in the Remote Repository.
- **Viewing the Stash:** Users can view the contents of the stash at any time by using the git stash list command. Users can also view the stash graphically by reviewing the Revision Graph diagram.
- **Applying the Stash Changes:** When ready, users can restore the stash into the current staged snapshot and then commit the changes including the stashed changes.



Appendix V: Transformations

This section illustrates the transformation to add converters available to the project team. The platform includes the template shown below. The project team must complete the required changes idempotently.

The following template is provided:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- Common converter service configuration -->
    <section name="ConversionServer" type="Aras.ConversionFramework.ConversionServer.Configuration.Conver
    <sectionGroup name="ConverterSettings">
      <!-- Place here class configuration section definitions for converters -->
      <section name="ArasCadConverter" type="Aras.ConversionFramework.Converter.Hoops.Configuration.HoopsC
    </sectionGroup>
  </configSections>
  <ConversionServer>
    <InnovatorServer url="" />
    <Converters>
      <Converter name="Aras CAD to PDF Converter" type="Aras.ConversionFramework.Converter.Hoops.HoopsCon
    </Converters>
  </ConversionServer>
  <ConverterSettings>
    <!-- Place here configuration sections for converters -->
    <ArasCadConverter>
      <Application converterPath="{Path.To.Hoops.Converter.Dir}\bin\hoops_converter.exe"/>
      <Command arguments="--input_pdf_template_file '{Path.To.Hoops.Converter.Dir}\templates\Blank_Templ
    </ArasCadConverter>
  </ConverterSettings> </configuration>
```

The project team must provide information in the transformation file to activate conversion according to the project's requirements and entitlements.

Below is an illustration of the desired state.

Element InnovatorServer:

"\${ADT_INNOVATOR_URL}/Server/InnovatorServer.aspx" consider using the simpler \${ADT_DATABASE_INNOVATORSER

Important

It is important to utilize these parameters, as the location of various servers in the cloud cannot be hardcoded or predetermined.

3. For Section Group, add missing Converter elements to the Converters element.

```
<Converters>
  <Converter name="cmf_ExcelPublishingConverter" type="Aras.Cmf.Publishing.Excel.ExcelExportConverter, Aras.Cmf.Publishing" />
  <Converter name="cmf_XpsPrintingConverter" type="Aras.Cmf.Publishing.Xps.XpsPrintingConverter, Aras.Cmf.Publishing" />
  <Converter name="Aras CAD to PDF Converter" type="Aras.ConversionFramework.Converter.Hoops.HoopsConverter, ArasCadConverter" />
  <Converter name="Aras PRC to SCS Converter" type="Aras.ConversionFramework.Converter.Hoops.HoopsConverterPrc, ArasCadConverter" />
  <Converter name="tp_XmlPublishingConverter" type="Aras.Publishing.XmlPublishingConverter, Aras.TDF.PublishingConverter" />
  <Converter name="tp_PdfPublishingConverter" type="Aras.Publishing.PdfPublishingConverter, Aras.TDF.PublishingConverter" />
  <Converter name="tp_htmlPublishingConverter" type="Aras.Publishing.HtmlPublishingConverter, Aras.TDF.PublishingConverter" />
  <Converter name="PDF.Watermarking" type="Aras.PDF.Watermarking.PdfWatermarkConverter, Aras.PDF.Watermarking" />
</Converters>
```

4. Add converters if missing for the ConverterSettings element. Notice that these elements have child elements.

```
<converterSettings adt:transform="replace" adt:locator="XPath(/configuration/ConverterSettings)">
  <!-- Place here configuration sections for converters -->
  <ArasCadConverters>
    <Application converterPath=".\\VOPES Converter\\bin\\converter.exe" />
    <Command arguments="--sc_compute_bounding_boxes 'All' --input_pdf_template_file '.\\VOPES Converter\\templates\\Blank_Template_1.pdf' --output_pdf 'FilePath\\VOPES\\Filename6.pdf' />
    <Output>
      <UploadToVault>
        <file extension="prc" arguMarkers="--output_prc" />
        <file extension="scs" arguMarkers="--output_scs" />
        <file extension="pdf" arguMarkers="--output_pdf" />
        <file extension="png" arguMarkers="--output_png" />
        <file extension="sti" arguMarkers="--output_sti" />
        <file extension="xml" arguMarkers="--output_xml_assemblytree" />
      </UploadToVault>
    </Output>
    <AssemblyCommand dynamicEnabled="true" arguments="--sc_compute_bounding_boxes 'All' --input_pdf_template_file '.\\VOPES Converter\\templates\\Blank_Template_1.pdf' --output_g />
  </ArasCadConverters>
  <ArasCadConverterPrc>
    <Application converterPath=".\\VOPES Converter\\bin\\converter.exe" />
    <Command arguments="--output_scs 'FilePath\\VOPES\\Filename6.scs' --output_xml_assemblytree 'FilePath\\VOPES\\Filename6.xml' --output_logfile 'FilePath\\VOPES\\Filename6.log' />
    <Output>
      <UploadToVault>
        <file extension="prc" arguMarkers="--output_prc" />
        <file extension="scs" arguMarkers="--output_scs" />
        <file extension="pdf" arguMarkers="--output_pdf" />
        <file extension="png" arguMarkers="--output_png" />
        <file extension="sti" arguMarkers="--output_sti" />
        <file extension="xml" arguMarkers="--output_xml_assemblytree" />
      </UploadToVault>
    </Output>
  </ArasCadConverterPrc>
  <PDFPublishingConverter>
    <conversionTool path="%[adt:conversionTool_PATH%]\\VOPES\\bin\\prince.exe"/>
  </PDFPublishingConverter>
</ConverterSettings>
```



Appendix VI: Vault Replication

This section illustrates the process of enabling vault replication for the Aras Innovator instance. Enabling the vault replication for the Aras Innovator instance requires two following steps:

1. Creating a transformation file.
2. Running the pipelines.

Create Transformation File

The following steps outline the process of creating a transformation file:

1. Create a transformation configuration file in the /TransformationsOfConfigFiles/AgentService/ folder of the Work.git Repository and name it replication.config.
2. Add the following content to the file:

```
<replication xmlns:xdt= http://schemas.microsoft.com/XML-Document-Transform status="enabled" xdt:Transform="SetAttributes(status)"></replication>
```
3. Commit the changes.

Run Pipelines

The following **Pipelines** can process the Aras Innovator deployments with Vault Replication:

- continuous-integration
- deploy-innovator
- delete-innovator

The following steps outline the process of configuring a Pipeline to enable Vault Replication:

1. To configure the Pipeline to deploy Aras Innovator with Vault Replication, set the **cluster_index** Pipeline Variable to **00**.
2. To configure the Pipeline to deploy Aras Innovator without vault replication, set the **cluster_index** Pipeline Variable to **01**.

A value of **00** indicates that the Pipeline deploys the necessary components needed to enable Vault Replication on all clusters.



A value of **01** indicates that the Pipeline deploys only the primary Aras Innovator instance without Vault Replication.

Refer to the Aras Innovator – Configuring Vault Replication document for the steps on configuring the replication rules.



Appendix VII: Configure CORS Settings in SDE with Aras.DevOpsFramework.SaaS3 package

Cross-Origin Resource Sharing (CORS) allows web applications hosted on different domains to connect safely to instances deployed through SDE with the Aras.DevOpsFramework.SaaS3 package. This is useful when third-party web apps need to reach APIs or other resources.

To set up CORS, the user must follow these steps:

1. Open the correct default variable group for the environment (CI, SIT, UAT, STG, or PROD), or a custom group if needed:
 - CustomConfigValues-CI
 - CustomConfigValues-SIT
 - CustomConfigValues-UAT
 - CustomConfigValues-STG
 - CustomConfigValues-PROD
2. Add the CORS settings to the variable named corsAllowOrigins in the variable group used for “Environment Specific Configuration”. (See the “Environment Specific Configuration” chapter for more details.)

Properties

Variable group name

CustomConfigValues-UAT

Description

Link secrets from an Azure key vault as variables ⓘ

Variables

Name ↑	Value	
<u>corsAllowOrigins</u>	https://supplier.example.com, https://partner.example.com	🔒

The value must be a text string that lists domains separated by commas. Each entry should follow this format:



```
protocol://domain[:port]
```

For example:

“ <https://example.one.com> , <https://example.two.com> ”.

Important

* symbol is not allowed in values.

Important

If this setting is not needed, the user should remove the corsAllowOrigins variable from the corresponding variable group.

3. Run the CI and Deploy pipelines when needed.

If the user needs to update the corsAllowOrigins value, the user should change the setting and run the Deploy pipeline using the **Update** strategy.

Important

Do not configure CORS using transformations in case of SDE with Aras.DevOpsFramework.SaaS3 package.



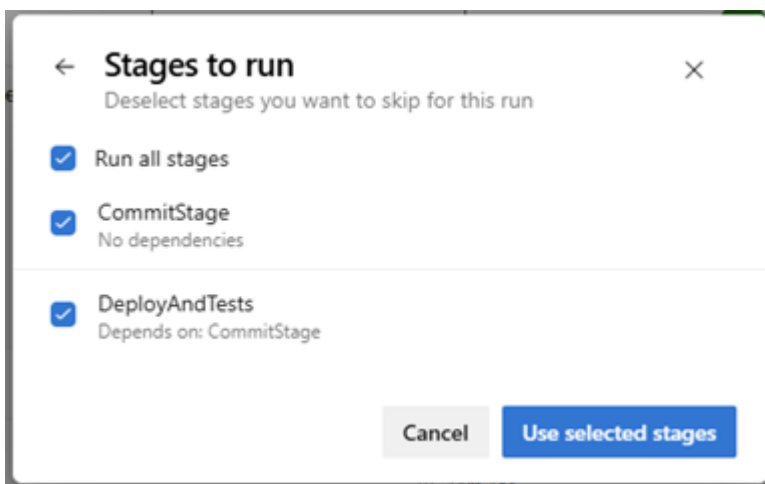
Appendix VIII: Continuous Integration Pipeline Configuration

By default, Continuous Integration pipeline is configured to execute the following logical steps:

1. Build artifacts with customizations
2. Deploy Aras Innovator with customizations.
3. Run integration tests
4. Delete Aras Innovator.

You can view and run these steps by opening the CI pipeline, clicking the Run Pipeline button, and expanding the Stages section. The default pipeline consists of the following stages:

- CommitStage – Builds artifacts with customizations.
- DeployAndTests – Deploys Aras Innovator, runs integration tests, and deletes the instance afterward.



Automatic Triggering

CI pipelines are automatically triggered when there are changes to the following branches: development, sit, uat, or prod, or when a Pull Request (PR) is targeted to any of them.

Important

CI pipelines will not trigger automatically for custom branches unless explicitly configured.

Use Case 1: Customize Build Validation Rules for Custom Branches

You may want to change the CI behavior for custom branches—for example, to trigger CI on PR creation or disable automatic CI runs.

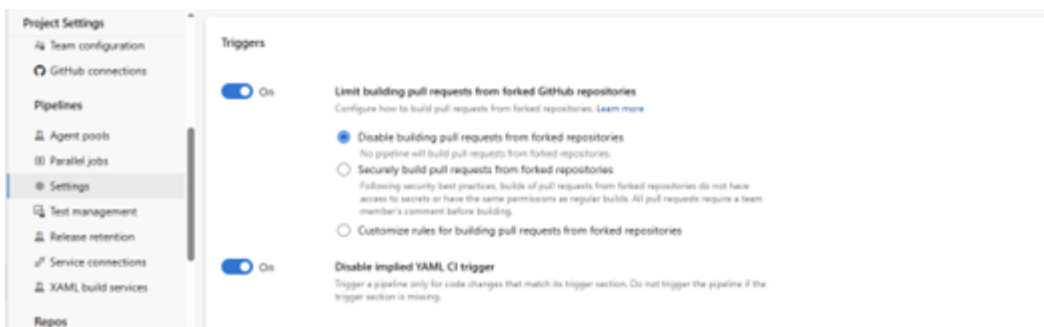
- **Disable Automatic CI Pipeline (Optional for Release 1.10+)**

Important

If you don't have the necessary permissions, please create a ticket in the support portal so that the support team can assist you.

To prevent the CI pipeline from running automatically:

1. Go to **Project Settings** → **Pipelines** → **Settings**
2. In the **Triggers** section, enable **Disable implied YAML CI trigger**



[Microsoft Docs: Prevent Unintended Pipeline Runs](#)

CI pipelines for development, sit, uat, and prod will still trigger automatically, but for custom branches it will not.

- **Enable CI Pipeline for Custom Branches on Pull Requests**

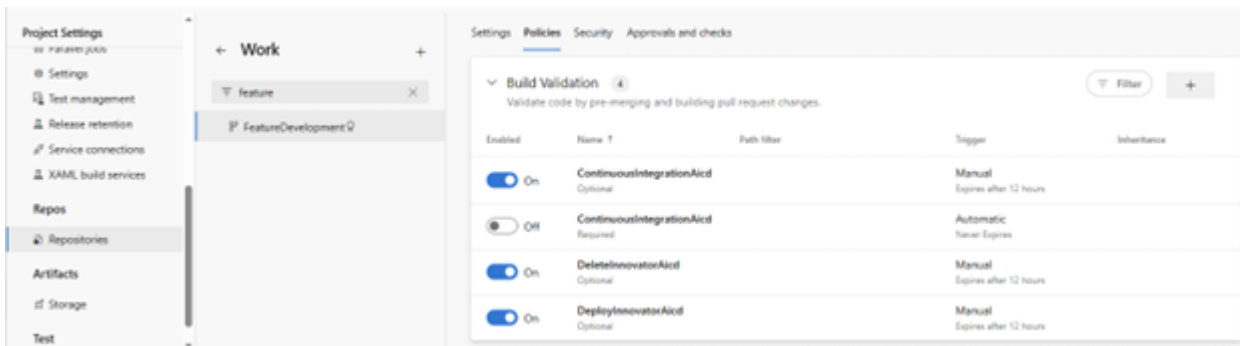
Important

If you don't have the necessary permissions, please create a ticket in the support portal so that the support team can assist you.

To configure CI for a specific custom branch:



1. Go to **Project Settings** → **Repositories** → **Your repository**
2. Select the branch (e.g., FeatureDevelopment)
3. Go to the **Policies** tab
4. Under **Build Validation**, add or edit the validation rules



[Microsoft Docs: Build Validation](#)

Use Case 2: Develop in Custom Branch Without CI Pull Request Validation

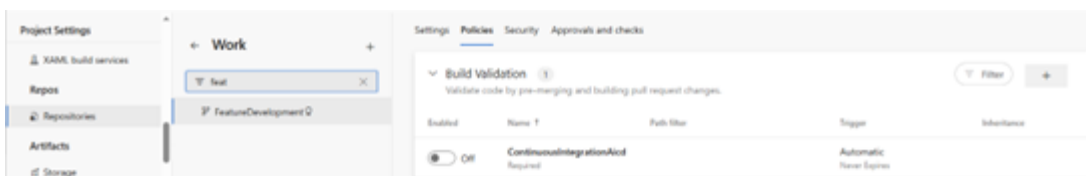
If you're developing in a custom branch and want to bypass CI validation during PRs, you can disable CI on PRs but still run it manually afterward.

- Disabling CI Pipeline automatic run for Pull Requests to custom branch
 1. Create a custom branch (e.g., FeatureDevelopments)
 2. Configure policies:

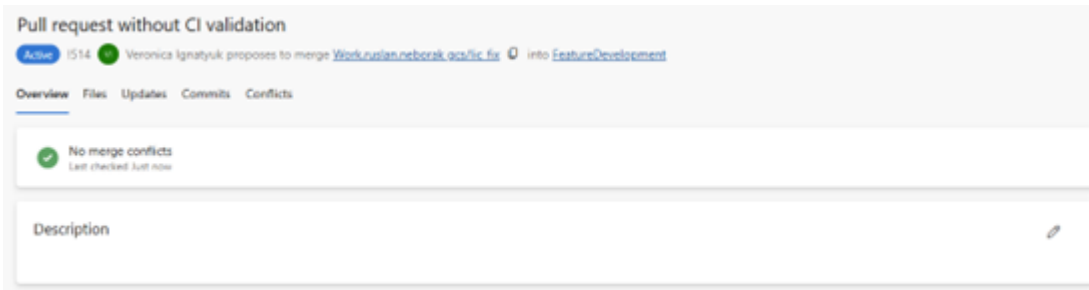
Important

If you don't have the necessary permissions to perform this step, please create a ticket in the support portal so that the support team can assist you.

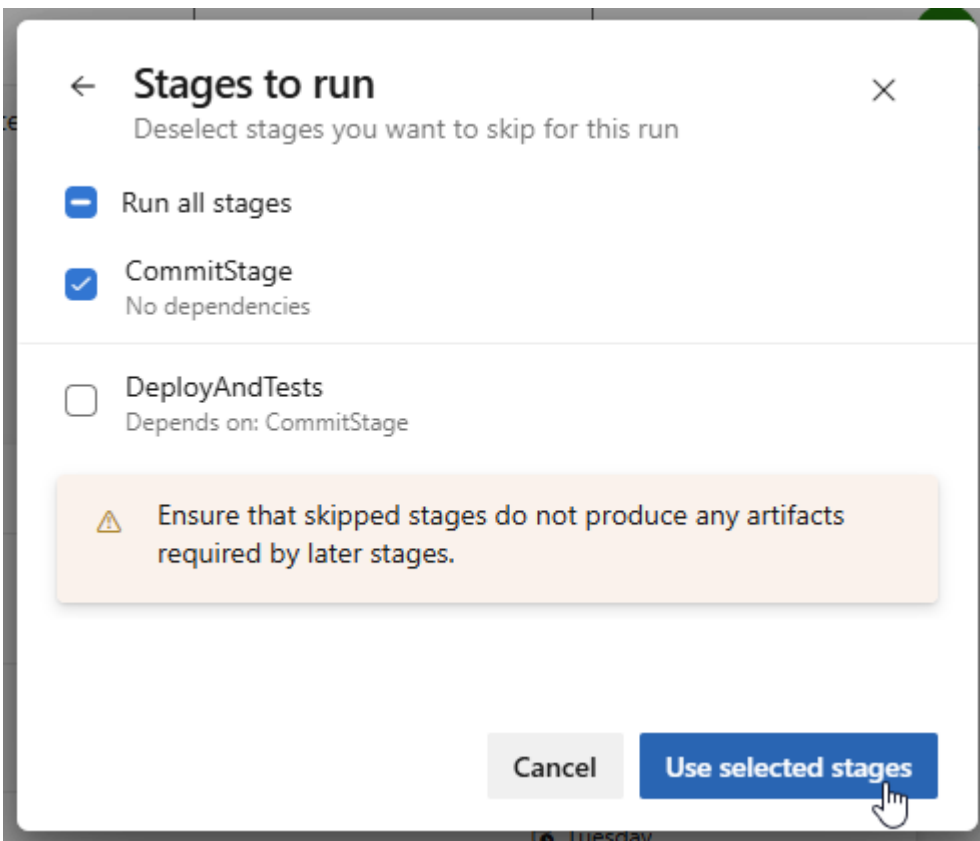
- Go to **Project Settings** → **Repositories** → **Your repository** → **Your branch**
- Under **Policies**, find **Build Validation**
- Either **remove** or **disable** the CI validation rule



This setup allows PRs to be merged without CI validation.

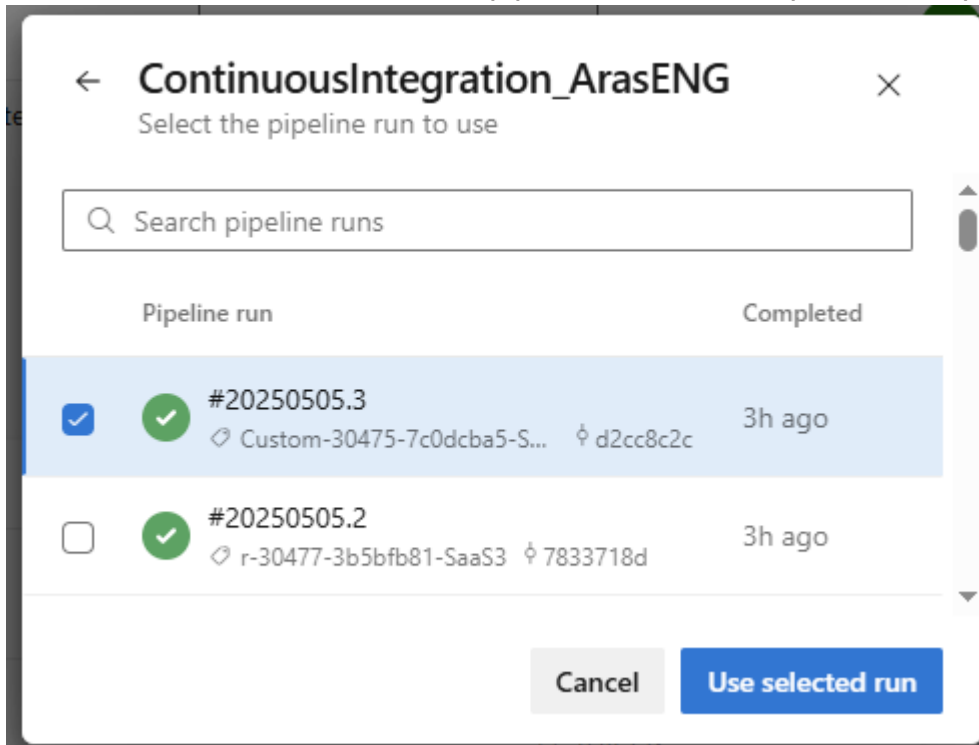


- **Run CI Pipeline with one stage manually after merge**
 1. Open the **CI pipeline**
 2. Click **Run Pipeline**
 3. Select your branch (e.g., FeatureDevelopments)
 4. Click **Stages to run**, select only CommitStage
 5. Click Use selected stages and Run pipeline



As soon as the pipeline passes it will be possible to deploy a new instance based on this CI pipeline run

- **Deploy using CI Artifacts with one stage**
 1. Open the **Deploy Innovator** pipeline
 2. Click **Run Pipeline**
 3. Select the same branch (e.g., FeatureDevelopments)
 4. Click **Resources** → Select the CI pipeline run from the previous step



5. Click **Use selected stages**, then **Run pipeline**

This allows rapid deployment based on the most recent merged changes.

Use Case 3: Fast Integration Without Repeating CI Validation

Each developer works in their own fork/branch and wants to integrate all branches with the development branch. You want to avoid running CI pipeline per each PR that can lead to multiple CI pipeline validation.

Steps:

1. Create custom branch for integration (e.g. IntegrationBranch)



2. Disable **Build Validation** rules for custom branch (see Use Case 2)
3. Target all Pull Requests to IntegrationBranch
4. Merge PRs without CI validation
5. Create a PR from IntegrationBranch to developmentbranch
6. Fix any issues until the CI pipeline passes ("green") before merging

Use Case 4: Validate Deployability Before Merging a PR to custom branch (Without Waiting for Integration Tests)

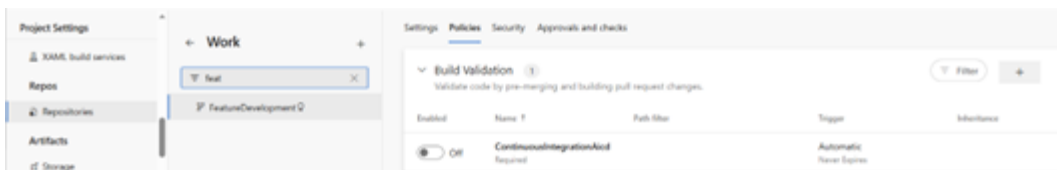
If your PR doesn't change any code covered by integration tests, you may want to validate deployability before merging it, skipping unnecessary test time.

Steps:

- Open Project settings -> Repositories -> Your work repository (e.g. "Work") -> Required branch (e.g. "FeatureDevelopments") -> Policies

Important

If you don't have the necessary permissions to perform this step, please create a ticket in the support portal so that the support team can assist you.



- Make sure that Continuous Integration pipeline is configured as optional for custom branch in "Build Validation" section.

Important

If you don't have the necessary permissions to perform this step, please create a ticket in the support portal so that the support team can assist you.



Edit build policy [X]

Enabled
 On

Build pipeline *
ContinuousIntegrationAicd [v]

Path filter (optional)
[] ⓘ

Trigger
 Automatic (whenever the source branch is updated)
 Manual

Policy requirement
 Required
Build must succeed in order to complete pull requests.
 Optional
Build failure will not block completion of pull requests.

Build expiration
 Immediately when ⁸⁹ FeatureDevelopment is updated
 After hours if ⁸⁹ FeatureDevelopment has been updated
 Never

Display name
[]

[Save] [Cancel]

- Create optional “Build Validation” policy for Deploy Innovator and Delete Innovator pipelines



Important

If you don't have the necessary permissions to perform this step, please create a ticket in the support portal so that the support team can assist you.

Enabled	Name	Path filter	Trigger	Inheritance
<input type="checkbox"/> Off	ContinuousIntegrationAicd Required		Automatic Never Expires	
<input checked="" type="checkbox"/> On	ContinuousIntegrationAicd Optional		Manual Expires after 12 hours	
<input checked="" type="checkbox"/> On	DeleteInnovatorAicd Optional		Manual Expires after 12 hours	
<input checked="" type="checkbox"/> On	DeployInnovatorAicd Optional		Manual Expires after 12 hours	

- Create Pull Request
- Click on "View 3 checks" link

Pull request without CI validation

Active 15:14 Veronica Ignatyuk proposes to merge [Worik.worik.neborak.oca/cic_fix](#) into [FeatureDevelopment](#)

Overview Files Updates Commits Conflicts

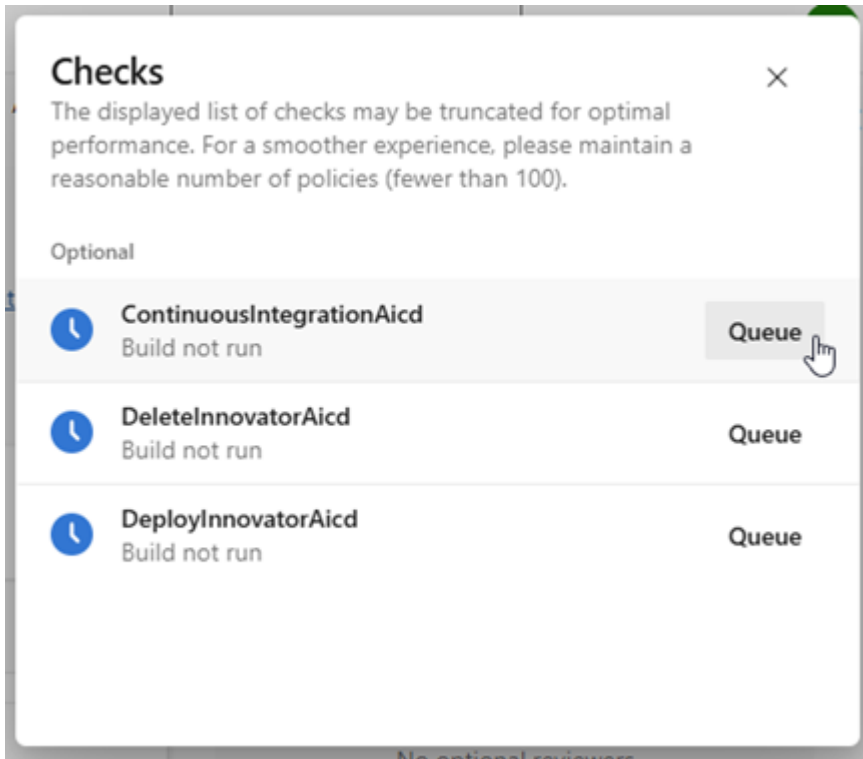
No required checks
3 optional checks not yet run

[View 3 checks](#)

No merge conflicts
Last checked: just now

- Click on "Queue" button for the CI pipeline





- If it is not necessary to run integration tests, then open triggered CI pipeline, cancel the triggered pipeline and click Run New button
- In the opened window make sure that pipeline will be triggered on Pull Request branch and configure other parameters of CI pipeline following the steps from Use Case 2

Branch/tag

Select the branch, commit, or tag

- Run pipeline
- When CI pipeline is finished, repeat steps 6 for Deploy Innovator pipeline, cancel the triggered pipeline and click Run New button
- In the opened window make sure that pipeline will be triggered on Pull Request branch

Branch/tag

Select the branch, commit, or tag



- Click on Resources and select Continuous Integration pipeline from step 6 as a source
- Configure other parameters as necessary
- Run pipeline
- After Deploy Innovator pipeline is finished, it is possible to do required validation.
- Repeat steps 10-14 for Delete Innovator pipeline to clean up.

