Continuous Integration from Git within MDCMS

Midrange Dynamics

From Version 8.6.1

Published May 12, 2024

Overview

What is Continuous Integration?

Continuous Integration (or CI) is the ability to automatically push modifications into the deployment cycle to target environments.

The other side of CI is CD, or Continuous Deployment, which is already fully handled by MDCMS itself, so this tutorial focuses on CI.

In the case of CI with Git, it is possible to configure the Git server to notify MDCMS whenever a Commit is pushed to the server for a specific repository.

Git then invokes the MDCMS webhook to trigger the CI processing. MDCMS inspects the files that were committed and if they reside in a branch and path that have been defined to be relevant, then MDCMS will retrieve the files from the Git server and place them on Object Requests within MDCMS for deployment to a target application level.

Prerequisites for using CI with MDCMS from Git

- An active MDCMS and MDOpen license (v8.2+) on the IBM i partition used to configure and connect with the Git server. A minimum MDCMS version of 8.6 is recommended.
- The MDOpen extension for VS Code or the MDOpen plug-in for Eclipse for performing the CI configuration.
- The type of Git server must be Azure DevOps, Bitbucket, GitHub or GitLab. These cover the most commonly used Git servers. If you use something else, let us know and we may be able to provide a Webhook for it.
- An active MDWorkflow Base license and a running MDCMS API http server must exist on the IBM i partition in order to enable the Webhook REST APIs. More information on MDREST API configuration can be found here.
- If you would like MDCMS to automatically generate Pull Requests once a deployment is complete to a particular target level, a MDWorkflow Pipeline license is also required.
- The Git server must be reachable within the network by the IBM i partition
- The MDCMS HTTP server must be reachable within the network by the Git server

Continuous Integration from Custom tools

The ability to push Object Requests from other tools that you have control over is possible by using the MDADDREQ command, the Object Request REST API or the Pipeline Request Trigger REST API.

Configure the Interface in MDOpen

Define the Git Credentials

Most Git Servers require authentication to communicate with a Git repository. If the repositories you need to connect allow anonymous access, you can skip to the next step.

Within the MDOpen option tree, click on Settings→DevOps Settings→Git Credentials

Within the Git Credentials view, click option Add

General Repository Settings

Credential ID	A unique identifier of up to 10 characters for a specific authentication configuration
Description	A description of the credentials
Git Authentication	SSH - A private/public key pair are used for authentication User/Password - a user and password are used for authentication

SSH Authentication Settings

Register new Private/Public Key Pair	enter or browse for the private or public key on your pc that is registered to be used on the Git server. The private key and public key must be in the same directory and have the same name, except for a .pub file type on the public key. For example the private key could be called githubkey and the public key could be called githubkey.pub
Set New Passphrase	If the private key is protected by a passphrase, enter it here

NOTE: MDCMS no longer supports SHA1 keys as they have been largely deprecated by the SSH community.

If you do not yet have a SHA2 public/private keypair available on your PC, here are steps to create them:

1. Go into a command prompt and execute the following command:
   ssh-keygen -t rsa-sha2-256 -b 2048 -C "MDCMS SSH Key"

2. You will then be prompted for the name and location of the keys. Change/Note that name
3. You will then be prompted for a passphrase – this is recommended to protect the keys
4. The keys will then be generated
5. Open the public key in an editor and copy the entire string
6. Register the string as a new SSH key within the settings of the Git service that you use

User/Password Authentication Settings

Repo User	The user id of the user registered on the Git server
Set New Password	The password of the user on the server

Once the values are entered, click Save.

Define the API Server

If your Git repositories are hosted by Azure DevOps, Bitbucket, GitHub or GitLab, and you have a MDWorkflow Pipeline license, you can define the host servers as API servers to communicate with them using REST. This provides the following 2 advantages:

1. You can very easily generate the Webhook definitions from within MDOpen
2. You can automatically generate Pull Requests once MDCMS has completed deployment to a specified target level.

If this does not apply to your situation, you can skip to the next step.

Within the MDOpen option tree, click on Settings→DevOps Settings→API Servers

Within the API Servers view, click option Add

General API Server Settings

Server ID	A unique identifier of up to 10 characters for a specific server
Description	A description of the server
Server Type	Select the type of Git Server that you will be connecting to: Azure DevOps Bitbucket GitHub GitLab

API Server URL	The URL of the server. For example: https://dev.azure.com

Azure DevOps API Server Settings

User	The user ID on the server – usually an email address
Set New Token	A valid Personal access token with rights to repositories, webhooks and pull requests. This can be set within Azure DevOps by clicking on User settings at the top right and selecting option Personal Access Tokens
Organization	The name of the organization that the project containing the repositories belongs to
Project	The name of the ADO project containing the repositories

Bitbucket Cloud API Server Settings

Set New Token	A valid Access Token with rights to repositories, webhooks and pull requests. This can be set for a workspace or for a specific repository
Workspace	The ID of the workspace that contains a collection of Git repositories. The value can be seen in URL when viewing the Repositories list

Bitbucket Data Center (or Server) API Server Settings

User	The user ID on the server
Set New Token	A valid Personal access token with rights to repositories, webhooks and pull requests. This can be set within Bitbucket by clicking on Manage User at the top right and selecting option Personal Access Tokens
Project	The key of the Project containing the repositories

GitHub API Server Settings

User	The user ID on the server
Set New Token	A valid Personal access token with rights to repositories, webhooks and pull requests. This can be set within GitHub by clicking on User settings at the top right and selecting option Settings.

	Then click option Developer settings. Then click option Personal access Tokens
Organization	The name of the organization that the repositories belong to, if applicable

GitLab API Server Settings

User	The user ID on the server
Set New Token	A valid Personal access token with rights to repositories, webhooks and pull requests. This can be set within GitLab by clicking on User settings at the top right and selecting option Preferences. Then click option Access Tokens. Set scope to api.

Define the Git Repository in MDOpen

MDCMS needs to know the URL of each Git Repository that will push artifacts to MDCMS.

Within the MDOpen option tree, click on Settings→DevOps Settings→Git Repositories

If the Repository is not yet defined, select option Add

Repository Settings

Repository ID	A unique identifier of up to 10 characters for the repository
URL	The network address (including http(s):// or git@) for the repository. The correct URL can usually be copied from the Git Server administration screen. However, if it is in http format and contains //user@host, then remove the user@ portion of the URL.
Git Type	Select the type of Git Server that you will be connecting to: Azure DevOps Bitbucket GitHub GitLab Other If it is one of the first 4, MDCMS has a specific Webhook defined for them and can be directly triggered for CI. When it is Other, you will need to have a pipeline tool such as Jenkins in between and then Jenkins can trigger the CI using MDCMS API /git/checkout. If it is one of the first 4, but you prefer that MDCMS is not involved until later in the DevOps process, you can also set the type to Other and use your master pipeline tool to orchestrate the CI.
Description	A description of the repository

Credential ID	If credentials are required, enter the Credential ID or use content-assist to select the ID of the credentials that provide access to the repository.
API Server ID	For simplified registration of webhooks and automated generation of Pull Requests, the repository needs to be tied to a API Server. Enter the API Server ID or use content-assist to select the ID of the server that provides REST API access to the Git server.
URL Value in Webhook	When the MDCMS webhook is triggered after the push of a commit, MDCMS attempts to map the URL in the webhook request body to a repository ID. If the URL in the request body is different than the URL used for MDCMS to clone/sync with the origin repository, the value in the webhook can be entered here. The value can best be obtained from the Git Webhook Deliveries entry after the webhook is triggered.
Main Branch	The name of the primary branch for the repository. This is typically main or master, but depending on the usage preferences, may be a different one. The Main Branch will be what is used for cross-referencing and pull-request merge purposes.

After these parameters are set, click Save. The MDGit service on the IBM i will then try to connect to the Git server using the credentials of the Credential ID (when not anonymous).

If this is not successful, click the Connection Logs icon for the Repository ID and open today's log to view the details of the exception. If a log is not available, then the MDGIT service job did not auto-start. In this case review the MDGIT service job settings such as job queue and auto-start.

If the connection is successful, MDCMS will automatically clone the main branch in the IFS under the /MDCMS/EXTREF folder. A re-clone is possible at any time by selecting option Re-Clone Main Branch for the Repository ID.

Map Git Users to MDCMS Users in MDOpen

If the developers that will commit changes to the Git server have different user IDs on the Git server than they have in MDCMS, their IDs can be mapped so that the correct user owns the object requests that are automatically created by the CI process.

Within the MDOpen Git Repositories view, select option User Mapping

Mapping Settings

Repository User	The User ID of a User on the Git Server. This value is case sensitive
MDCMS User	The User ID of a User registered in MDCMS (MDSEC) to be mapped to

Repo ID	*ANY - any repository containing the Repository User should map to the given MDCMS User Otherwise, specify the Repository ID or use content-assist to select from the list

Define the Continuous Integration Settings in MDOpen

Once the repository is defined, the repository, or certain paths within the repository, can be configured for CI.

Within the MDOpen option tree, click on Settings→DevOps Settings→Continuous Integrations

If the integration is not yet defined, select option Add. Multiple CI definitions can be created for the same repository when various granular selections are required.

CI Settings

Repo ID	The ID of the Repository. Use content-assist to select from the list of defined Git repositories.
Branch Opt	Defines which branch within the repository triggers CI processing in MDCMS. Main Branch – when a commit is pushed to the branch designated as Main Branch in the MDOpen Repository definition Not Main Branch – when a commit is pushed to any branch except the Main Branch Specific Branch – when a commit is pushed to the branch selected in Specific Branch.
Specific Branch	When Branch Opt = Specific Branch, enter the name of the branch or use content-assist to select it from a list
Path	if blank, then anything committed to the repository will be pushed to MDCMS. Otherwise, specify the path to limit the pushed items to anything located in the path or within a child directory of the path. If multiple paths should be considered for the same repository, copy the setting record for each additional distinct parent path. If the repository has already been cloned to MDCMS, content-assist can be used to select the path.
File Naming Pattern	If blank, then any files in the path will be requested (unless blocked in the Attribute Mapping). Otherwise, state the naming pattern to filter which files to request. For example, *.jar for any files of type jar
Application	The target MDCMS application to push objects to *APPL - the application will be determined by the Application field for the task that the object request will be assigned to specific application - enter the id of a defined application or select using content-assist

Level	The target MDCMS application level to push objects to. The level must allow checkouts, unless the parameter "Level is Target of a Merge" is set to true. If Application is set to *APPL, and a specific level number is defined for the CI definition, then the Level field for the task that the object request will be assigned to is used. If level is 0, MDCMS will retrieve the lowest level number allowing checkout for the application. When parameter "Level is Target of a Merge" is set to true, a specific level number must be defined.
Level is Target of a Merge	If the push from the branch will target a level that doesn't allow checkouts (such as QA or production), set this parameter to true. This typically occurs after a Pull Request is completed that merges changes into QA or production branch.
Checkout	Diffs - only push the differences (adds/changes/deletes) based on comparing the working tree for the branch before and after the commit Contents - push the complete contents of the path
Request IFS/Remote Objects	If true, the contents of the path are directories and files that will be deployed to the IFS or a remote Linux or Windows server
Request Source for IBMi Objects	If true, the contents of the path are source files of system objects on the IBM i NOTE: a specific CI definition can only be for IFS/Remote Objects or for Source for IBMi Objects - not for both at the same time. If both exist on the repository, then create a CI definition for each.
Default MD Attribute	The MDCMS Attribute to assign to the pushed objects, if attribute mapping doesn't locate an attribute. This is recommended for IFS/Remote Objects and can be of type *IFS, *PIPE or *REMOTE. If for Source, it can be used, if mapping, history and MDXREF doesn't find a match.
Default User	The default user to assign the object requests to, if the user isn't found in the mapping table for the Repository
Relative Path	The relative path for the object requests. From Repo Root - the relative path is the entire path from the repository itself From Repo Path - the relative path is only any folders located below the path defined in the Path parameter No Relative Path - all files should be deployed directly to the target folder for the attribute rather than a relative path under that folder.
Map to IFS Path	If requesting source for IBM i objects, the correct MDCMS attribute to use for source that is new to MDCMS can be more accurately selected if the source for the target level is stored in the IFS and the folder structure and naming is identical to the working tree of the repository.

	For example, program source and module source each use the same file types, so MDCMS can't know which attribute to apply based only on the file type, but if module source is kept in a different folder than program source on both the repository and in the IFS on the IBM i, then MDCMS will know the intent.
Project	The project to assign to the Object Requests. If blank, the project can be assigned prior to submitting the RFP *BRANCH – the repository branch name is either the reference code for an Azure Workitem, Jira Issue or ServiceNow task, or it is a MDCMS Project/Task/Subtask. See those special values for instructions, whereby MDCMS will parse the branch name instead of the commit message and a : ending delimiter isn't expected. If the branch naming convention will be more descriptive than a Reference ID or MDCMS Project/Task/Subtask ID, then do the following: 1) create a Custom Project/Task field to contain the branch name. It should be defined as a String and may be up to 160 characters in length. 2) Add the Custom Field to the appropriate Task Types 3) If using Azure DevOps, Jira or ServiceNow to manage tasks, add the custom field to the screen for those tasks and map the field to the MDCMS Custom field. 4) Enter the value of the new branch into the task field prior to committing changes to the branch. *AZUR - The project/task/subtask will be based on the Azure Workitem number in the commit message. For this to work, the developer must enter the Azure Workitem number at the very beginning of the commit message and the workitem must have already been exported into MDCMS. *JIRA - The project/task/subtask will be based on the JIRA Issue Key in the commit message. For this to work, the developer must enter the JIRA Issue Key at the very beginning of the commit message and the issue must have already been exported into MDCMS. For example, if the commit message was MDSD-3891 - demo of CI, then MDCMS will check if issue key MDSD-3891 maps to an MDCMS task or subtask and if it does, that will be assigned to the object requests. *KACE - The project/task/subtask will be based on the KACE Service Desk number in the commit message. For this to work, the developer must enter the KACE number at the very beginning of the commit message and the workitem must have already been exported into MDCMS. *SNOW - The project/task/subtask will be based on the ServiceNow Task ID in the commit message. For this to work, the developer must enter the ServiceNow Task ID at the very beginning of the commit message and the task must have already been exported into MDCMS. *LAST - the Project/Task/Subtask for the most recently installed object of the same name and attribute will be used. Recommended when for the target level of a Merge (completion of a Pull-Request) The use case for this is if requests for higher MDCMS environments are triggered by pull-requests into higher branches for a Git Repository. *MD - The project/task/subtask will be based on the MDCMS Project, Task and Subtask values. For this to work, the developer must enter the information at the very beginning of

	the commit message. The format is PROJECT-TASK.SUBTASK: The : is the delimiter for the end of the information to parse. MDCMS then takes the value until the : or final - for the project id. If the project is valid, MDCMS takes the values between the final - and the : or final . for the Task number. If the task is valid, MDCMS takes the values between the final . and the : for the subtask number. The Task Number and Subtask Number are generally optional.
Task	The task to assign to the Object Requests. Leave blank if directly for a project, or if the project is blank or has a special value.
Subtask	The subtask to assign the Object Requests. Leave blank if directly for a project or task, or if the project is blank or has a special value.
Generate RFP	True - the object requests will be assigned to a new RFP (if not already checked out) False - the object requests will not be assigned to an RFP If the object request already exists from the same branch, it will be automatically updated to the newly committed version.
RFP Description	If assigned to an RFP, what the description of the RFP should be. *COMMITMSG - use the Git commit message *TASK – use the task description
Auto-Request Dependencies	If requesting source for IBM i objects, MDCMS can automatically request any dependencies for recompile. This includes dependencies impacted by file changes, module changes or copybook changes. The source used for the recompile must exist in the source migration path, based-on path or source search template on the target partition.
Auto-Create in Dev Lib	If requesting source for IBM i objects, MDCMS can automatically attempt to compile the requested objects as well as the dependencies into the development library. If the target of the source is a source member, the stream file will be copied to the source file of the same name in the developer’s work library.
Auto-Submit RFP	If true, MDCMS will auto-submit the RFP once the objects have been requested, even if the Auto-Submit parameter on the level is set to false
Pipeline Attribute	If an attribute of type *PIPE is entered in this parameter (content-assist can be used), an object request will be automatically generated for that attribute for the same RFP as for the committed objects. This is typically used to trigger build, test, deploy or approval jobs on pipeline servers.
Pull-Request Location / Level	MDCMS can automatically generate a Pull Request from the branch that commits were pushed to, for a merge into a target branch. For this to occur, the repository definition must include the API Server ID and the Partition Location ID and Level number must be entered

	here. When MDCMS has completed the deployment of the objects into the defined target level on the defined target location, the pull-request will be generated, if a pull-request from the same branch isn't already pending.
Pull-Request Target Branch	The name of the branch that the changes should be merged into. If blank, the designated Main branch of the repository will be used.

Map Repository Files to MDCMS Attributes

When pushing files to be requested as IFS/Remote Objects, the CI processor does the following to determine the MD Attribute to use for a file: 1. look for a path/file name pattern match in the Attribute Mapping table for the Git Repository 2. use the Default MD Attribute defined for the CI definition

When pushing files to be requested as Source for System Objects, the CI processor does the following to determine the MD Attribute to use for a file: 1. look for a path/file name pattern match in the Attribute Mapping table for the Git Repository 2. if Map to IFS Path is true, check MDCMS Object History for most recent install of the file into the mapped path 3. check MDCMS Object History for most recent install of the file into the same application/level 4. check MDCMS Object History for most recent install of the file into the same application, regardless of level 5. check MDCMS Object History for most recent install of the file, regardless of application or level 6. in the MDCMS Attributes table, try to find a matching combination of Folder and Dft Source Type, or matching combination of Source Library, Source File and Dft Source Type to get the object type. If found, check for a reusable command for the object name and type and otherwise use the first matching MDCMS Attribute 7. use the Default MD Attribute defined for the CI definition

Define Attribute Mapping for the Git Repository

If certain path/file name exceptions exist that result in the wrong attribute being used or if certain files should be omitted, Attribute Mapping is used to define the results. Select option Attribute Mapping for the Git Repository.

Appl	The target application that the mapping relates to. A different mapping record can exist for each application targeted by CI definitions.
Path	A specific path from the root of the repository or a generic pattern for the path. The path value doesn't include the name of the file itself
File Name	A specific file name or a generic pattern for the file name
MD Attribute	The attribute that a match should map to *NONE - omit any files that match

Define the MDCMS WebHook on the Git Server

In order to trigger the push of objects from Git to MDCMS, a WebHook needs to be defined on the Git server.

Instructions for Azure DevOps, Bitbucket, GitHub or GitLab when an API Server ID is defined for the Repository

1. Within MDOpen, open the Git Repositories view
2. Right-Click on the Repository ID and select option Webhooks
3. Click the Create/Update Webhooks button

If an error occurs, check the API Server log for the given server ID

Instructions for Azure DevOps without an API Server ID

1. Log into the Azure server with a user that has administrative rights for the DevOps project containing the Repositories to trigger from.
2. Go into the Project Settings for the Project
3. Click on the Service hooks link
4. Click the + button to add a subscription
5. Select Web Hooks from the list and click Next
6. Select the Code pushed Trigger event from the list
7. Select the specific or (Any) Repository, Branch and group member and click Next
8. Enter the URL for the MDCMS Webhook. It will be the Endpoint defined in the MDCMS REST APIs settings + /<mdcms instance> + /azure/git/webhook
   For example, https://devsys.mycompany.com/mdcms/azure/git/webhook
9. The optional settings should be left at the default values
10. Click the Finish button

Instructions for Bitbucket

1. Log into the Bitbucket server with a user that has administrative rights.
2. Go into the Settings for the Repository
3. Click on the Webhooks link
4. Click the button Add webhook
5. Enter a Title (such as MDCMS)
6. Enter the URL for the MDCMS Webhook. It will be the Endpoint defined in the MDCMS REST APIs settings + /<mdcms instance> + /bitbucket/webhook
   For example, https://devsys.mycompany.com/mdcms/bitbucket/webhook

7. It is recommended to Enable request history collection so that Git-side troubleshooting can occur during the initial attempts
8. Ensure Repository push is selected for Triggers
9. Click the Save button

Instructions for GitHub without an API Server ID

1. Log into the GitHub server with a user that has administrative rights.
2. Click on the Settings tab for the Repository
3. Click on the Webhooks link under Options
4. Click the button Add webhook
5. Enter the URL for the MDCMS Webhook. It will be the Endpoint defined in the MDCMS REST APIs settings + /<mdcms instance> + /github/webhook
   For example, https://devsys.mycompany.com/mdcms/github/webhook
6. Ensure Just the push event is selected for Triggers
7. Click the button Add webhook

Instructions for GitLab without an API Server ID

1. Log into the GitLab server with a user that has administrative rights.
2. For a given Project, hover the cursor over the Settings item in the sidebar and then Click on Webhooks
3. Enter the URL for the MDCMS Webhook. It will be the Endpoint defined in the MDCMS REST APIs settings + /<mdcms instance> + /gitlab/webhook
   For example, https://devsys.mycompany.com/mdcms/gitlab/webhook
4. Ensure the Push events is selected for Triggers
5. Click the button Add webhook

Troubleshooting

There is logging available at every step in the CI process to aid in problem resolution.

Git Webhook Deliveries

The first place to start troubleshooting or view occurred activity is to use option Git Webhook Deliveries in MDOpen. Each occurrence of the MDCMS git webhook invocation will be listed. For each row, any requested objects can be listed or the delivery can be easily repeated.

If an expected entry does not exist, then review the following locations:

Git Server

On the Azure, Bitbucket, GitHub and GitLab servers, the existing WebHook can be clicked on in order to view/repeat a recent request and response payload

MDCMS REST API Server

Under Settings→Logging within MDCMS:

- Log File MDCMS/LXERRLOG contains any errors in invoking a REST API
- Log File MDXREF/MDDFREP contains every transaction received from the MDCMS WebHook for Git along with how the URL and branch was mapped.

MDGIT service

The processing of the Git contents is passed from the WebHook job to the MDGIT service. The java log for this service can be reviewed for activity and exceptions.

Best Practices for using Git with MDCMS

1. The Git Repository should be structured so that it is possible for MDCMS to uniquely identify the correct attribute to use for every pushed file. Create Attribute Mapping entries to ensure the correct attributes will be assigned.
2. The Git Repository should contain a main branch reflecting the production level code.
3. The Git Repository should contain a QA branch reflecting the QA level code.
4. All code changes should be performed in a feature branch that is tied to an MDCMS Task. To minimize developer effort, the branch name should equal the PROJECT-TASK name (example: MYPROJ-123).
5. Publish the feature branch before making any changes.
6. Define the Developer Library Naming template for the checkout level to be a combination of the Project and Task.
7. Define the feature branch CI with the following values:
    - Branch Opt = Not Main Branch
    - Project = *BRANCH
    - Generate RFP = true
    - If for Source for IBM i Objects, set Auto-Request Dependencies to true
    - If for Source for IBM i Objects, set Auto-Create in Dev Lib to true
    - Pull-Request Location = local location
    - Pull-Request Level = target level
    - Pull-Request Target Branch = (name of QA branch)
8. Define the Developer Library Naming template for the QA level to be a static library and folder name.

9. Define the QA branch CI with the following values:
    - Branch Opt = Specific Branch
    - Specific Branch = (name of QA branch)
    - Set Level to level number of the copy of QA on the Dev partition for the Application
    - Level is Target of a Merge = true
    - Project = *LAST
    - Generate RFP = true
    - If for Source for IBM i Objects, set Auto-Request Dependencies to true
    - If for Source for IBM i Objects, set Auto-Create in Dev Lib to true
    - Pull-Request Location = local location
    - Pull-Request Level = target level
    - Pull-Request Target Branch = (name of main branch)
10. Define the Developer Library Naming template for the production level to be a static library and folder name.
11. Define the main branch CI with the following values:
    - Branch Opt = Main Branch
    - Set Level to level number of the copy of production on the Dev partition for the Application
    - Level is Target of a Merge = true
    - Project = *LAST
    - Generate RFP = true
    - If for Source for IBM i Objects, set Auto-Request Dependencies to true
    - If for Source for IBM i Objects, set Auto-Create in Dev Lib to true
12. Create a local Distribution Level from the checkout level to the QA level with the Local Distribution Reason set to N=Non-Git Requests. MDCMS will automatically place any object requests that didn't originate from Git in a Send RFP to then merge with the RFP that comes from Git after the Pull-Request is completed. Do the same from the QA level to the production level.
13. Ensure all Git user id's are mapped to MDSEC user id's so that the correct user is indicated on the requests and RFP.
14. A full copy of all source code should still be on the development partition for recompiling, debugging, analysis and emergency use.

