

# Git Integration for CI/CD Pipelines – Webhooks

## Objectives

- Apply configuration changes to Red Hat Ansible Automation Platform based on files stored in a Git repository by triggering job execution with webhooks.

## Using Webhooks in Red Hat Ansible Automation Platform

A *webhook* is a user-defined HTTP callback, performed as a HTTP POST request, that allows applications to communicate and share information with each other. Most frequently, webhooks are used to notify applications about events. The message sent by the POST request is usually formatted as JSON or XML content or form data that the application can interpret. Webhook calls are usually authenticated to ensure that unauthorized requests are rejected.

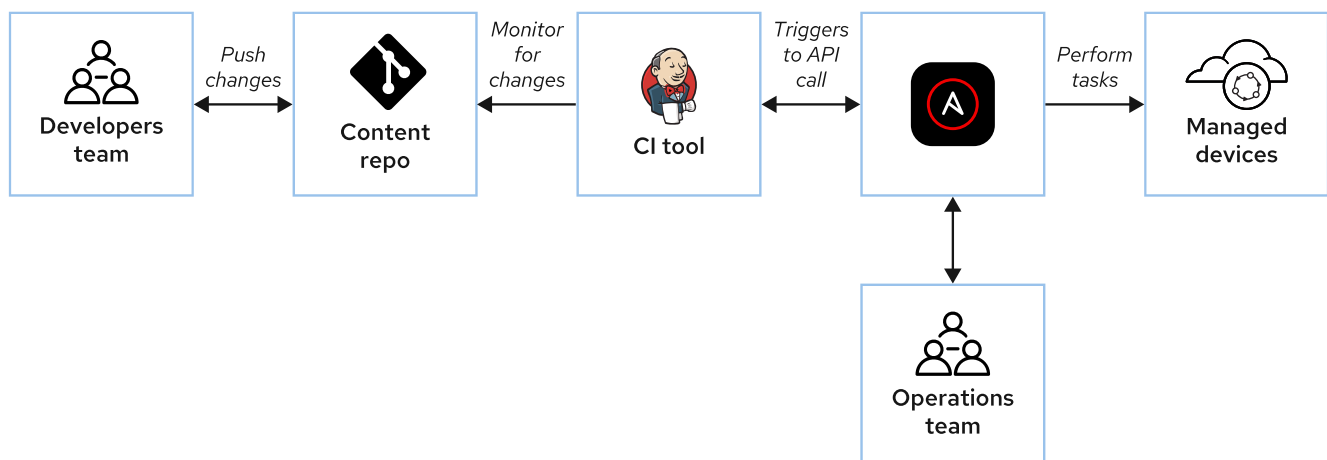
You can set up your automation controller to use webhooks so that it is notified when events occur for a Git repository, such as new commits being added to a particular branch.

Automation controller can then perform specific tasks when it receives certain events. For example, a new commit to a Git repository could cause automation controller to automatically update a related project and launch a job template that uses it.

## What Are the Benefits of Webhooks?

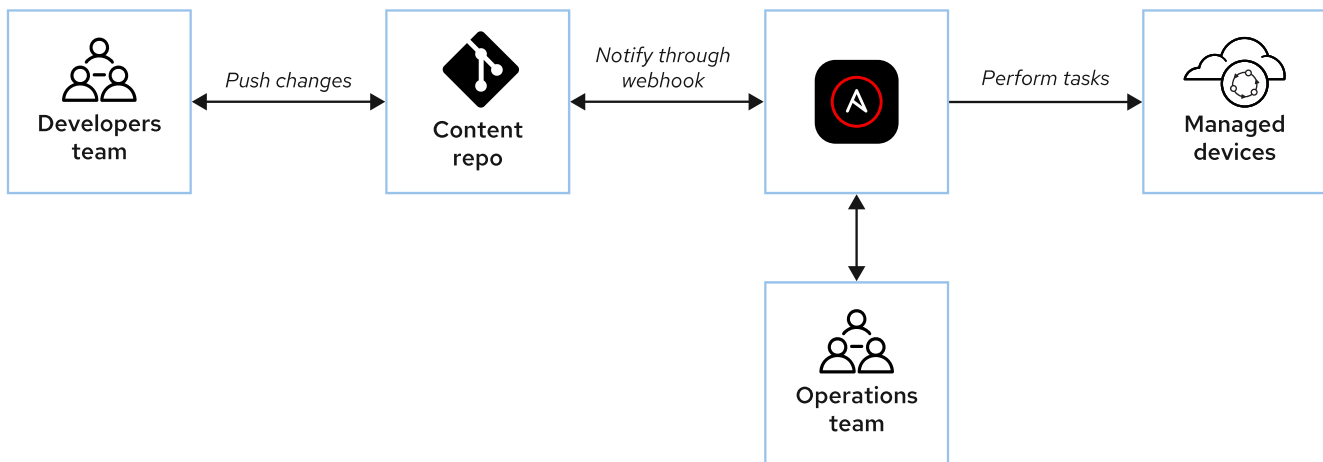
Webhooks help you automate workflows that involve multiple applications and components.

As an example of one way that webhooks can help automate workflows, the following diagram illustrates a common Continuous Integration (CI) workflow that might be used to perform tasks for the devices managed by Ansible.



In this example, the operations team is in charge of providing automation content that performs tasks on the Ansible-managed hosts. When the developers' team pushes a change to the Git content repository, the Jenkins CI tool captures this event. The event signals Jenkins to use the API on automation controller to automatically launch a job template that updates its copy of the project and runs on the managed hosts.

The following diagram shows another approach that uses webhooks:



By using webhooks, you simplify the workflow. In this case, when a developer pushes something to the Git content repository, automation controller gets notified directly by a webhook. Automation controller then launches a job template that updates its copy of the project and runs on the managed hosts.

Therefore, one major benefit is that you can use webhooks to trigger actions on automation controller without the need to maintain and manage an additional Continuous Integration (CI) tool such as Jenkins.

## Configuring Webhooks

To set up webhooks, you need to prepare both your Git repository server so that it triggers webhooks, and your automation controller to listen for them. You also need to configure both sides so that the Git repository can authenticate its messages to automation controller.

### Important

In the following example, GitLab is used as the Git repository server because it matches the classroom environment used by this course for hands-on exercises. In practice, you can use other Git repository servers and services such as GitHub

instead. They can be configured using a procedure similar to the one illustrated for GitLab by this course.

## Configuring Automation Controller to Listen for Webhooks

To configure a project to update a job template to launch automatically when automation controller is notified by a webhook, you must configure the project and job template to meet certain requirements:

- For the project, you must enable the Update Revision on Launch option. This ensures that automation controller pulls the latest revision from the Git repository when it launches the job template because the Git repository has changed.
- Because you cannot interact with the job template, you must disable *all* the Prompt on launch settings in any job template that webhook notifications might launch.
- After you have met the previous requirements, then you have to configure automation controller to listen for webhook notifications.

The following example outlines one way to do this for notifications from a GitLab repository. It enables a webhook listener for an existing job template that is already configured to run without prompting and that uses a project that is configured with Update Revision on Launch selected.

1. Log in to the automation controller web UI as the **admin** user.
2. Navigate to Resources → Templates, and click the Edit Template icon for the job template that you want to edit.
3. Select the Enable Webhook checkbox, and choose GitLab from the Webhook Service field.
4. Click Save and then go to Details tab to see the webhook URL and webhook key needed to configure the webhook in GitLab. GitLab needs to send the webhook notification to the webhook URL and use the webhook key to authenticate its message to automation controller.

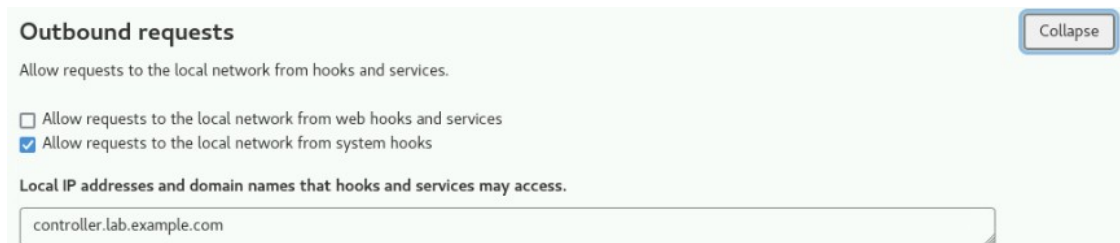
Webhook Service	GitLab	Webhook URL	https://controller.lab.example.com/api/v2/job_templates/9/gitlab/	Webhook Key	OXsYs72odGZAa75hB VeMsByqQG5GL3qaN zui9t0UTvrwTAA4BP
-----------------	--------	-------------	---	-------------	--

## Creating the Webhook for the Repository in GitLab

Before any user can configure webhooks on your GitLab server, an administrator of the server must allow them to be used.

If this has not been done, you need to log in to your GitLab server as a user that has an `admin` role. Navigate to Admin Area → Settings → Network → Outbound requests. Make sure that the automation controller to which you want to send webhook notifications is listed under Local IP addresses and domain names that hooks and services may access.

The result should appear similar to the following:



**Outbound requests** Collapse

Allow requests to the local network from hooks and services.

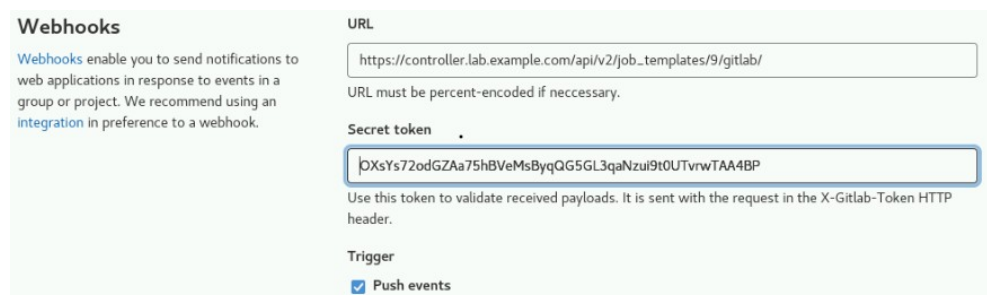
☐ Allow requests to the local network from web hooks and services  
☒ Allow requests to the local network from system hooks

Local IP addresses and domain names that hooks and services may access.

controller.lab.example.com

If this requirement is correctly configured, a regular user can create webhooks for their Git projects.

1. Log in to the GitLab web UI as a regular user.
2. Navigate to your Git project (one of the repositories under Projects → Your projects), and then go to Settings → Webhooks.
3. On the Webhooks configuration page, create the webhook by adding the information you obtained from automation controller, putting the webhook URL into the URL field and the webhook key into the Secret token field.
4. In the Trigger section, select which events trigger the webhook notification, and click Add webhook.



**Webhooks**

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.

**URL**

https://controller.lab.example.com/api/v2/job\_templates/9/gitlab/

URL must be percent-encoded if necessary.

**Secret token**

DXsYs72odGZAa75hBVeMsByqQG5GL3qaNzui9t0UTvrwTAA4BP

Use this token to validate received payloads. It is sent with the request in the X-Gitlab-Token HTTP header.

**Trigger**

☒ Push events

5. You can test the webhook by clicking Test → Push events. You get the message `Hook executed successfully` if you configured it correctly.

## Use Cases for Webhooks

You can use webhooks with automation controller in a variety of ways. The subsections that follow explore two of them.

### Triggering Different Job Templates Using Branches

For this use case, you use a webhook for a job template specifying a branch from the Source Control Branch field. When you push new content for the given branch, the job template starts performing Ansible tasks on the inventory that you set up for that job template.

#### Important

The Source Control Branch field is only available if you set up a Git project with Allow Branch Override selected.

For example, suppose you want to test a Git repository for a development environment before going to production. In that case, you can create a development branch for the repository, configure the job template to use this branch, and associate the inventory with the development hosts. When you add a commit to the development branch, the webhook launches the job template for the development hosts.

After you verify that the development branch of your automation is working correctly, you can merge your changes to the production branch, and even use the same webhook mechanism to automatically launch a job template that runs for managed hosts in your production inventory.

### Configuration as Code

Configuration as code is a technique that uses a version control repository to describe the desired state of your configuration.

You can use configuration as code methods in conjunction with Ansible Content Collections and Ansible Playbooks to manage the configuration of your Red Hat Ansible Automation Platform systems in a version control system. This is similar in principle to the concept of Infrastructure as Code, and has similar benefits.

For example, you could implement configuration as code to automatically reconfigure and help you maintain your Ansible Automation Platform servers. You set up files and Ansible Playbooks in a Git repository that define and apply the desired configuration for your automation infrastructure.

You can accomplish this by combining webhooks with playbooks that use the `ansible.controller` Ansible Content Collection. Using this Ansible Content Collection, you can configure most of the objects in automation controller (such as job templates, projects, inventories, and so on), using Git as a single source of truth.

To summarize some benefits of storing and managing your Ansible Automation Platform configuration as code:

- You can standardize your configuration in several operating environments.
- You can easily track the changes in your configuration by using the version control features and see who made what changes and when, which can help you troubleshoot and resolve issues.
- You can easily replicate and deploy additional Ansible Automation Platform deployments with the same configuration.

## Custom Exercise for DO417:

### Git Integration for CI/CD Pipelines – Webhooks

Use webhooks to reconfigure automation controller when you commit changes to content that is stored in a Git repository.

#### Outcomes

- Configure a job template that runs when a Git server triggers a webhook to notify your automation controller that a new commit has been applied to the repository for the job template's project.

As the `student` user on the `desktop` machine, open a VS Code terminal by clicking **View** → **Terminal**. Use the `ssh` command to connect to the `workstation` machine.

In the SSH session to the `workstation` machine, use the commands below to prepare your system for this exercise. These commands prepare your environment and ensure that all required resources are available.

```
PS C:\Users\student> ssh student@workstation

[student@workstation ~]$ lab start playbooks-run

[student@workstation ~]$ mkdir git-repos && cd git-repos

[student@workstation git-repos]$ git clone \
https://github.com/jessedscott/aap_demo

[student@workstation git-repos]$ cd aap_demo/update_controller

[student@workstation update_controller]$ ansible-galaxy collection install \
-r collections/requirements.yml

[student@workstation update_controller]$ ansible-playbook resources.yml
```

#### Instructions

1. As the `root` user on your GitLab server, configure settings to allow requests from webhooks on the local network.
  - 1.1. Navigate to `https://git.lab.example.com` and log in as the `root` user using `R3Dh4tR00t` as the password.

- 1.2. Use the search bar in the left panel to navigate to the **Admin Area**. Note that the left panel may not be present until you move your mouse pointer to the left side of the page.
- 1.3. Navigate to **Settings** → **Network** in the left panel. Select **Outbound Requests** and check the box to **Allow requests to the local network from webhooks and integrations**. Click on **Save changes**.
2. On your automation controller, configure an existing job template named **Install IIS** to use a webhook.
  - 2.1. Navigate to `https://controller.lab.example.com` and log in as the student user with `StuD3nt123!@` as the password.
  - 2.2. Navigate to **Resources** → **Templates** and click the **Edit Template** icon for the **Install IIS** job template.
  - 2.3. Scroll down to the **Options** section and select **Enable Webhook**.
  - 2.4. In the **Webhook details** section, choose **GitLab** from the Webhook Service list.
  - 2.5. Click **Save**.
  - 2.6. On the **Details** tab, take note of the **Webhook URL** and **Webhook Key** values needed to configure the webhook in GitLab in the next step.

<b>Webhook URL</b>	<code>https://controller.lab.example.com/api/v2/job_templates/11/gitlab/</code>	<b>Webhook Key</b>	<code>vJygx4F7PaCGD6g4D57y3hruzYmxqJWmvSMCMvS7COrIKQD7Zp</code>
--------------------	---	--------------------	---

3. As the student user on your GitLab server, create a webhook for the **playbooks** Git repository that notifies your automation controller about push events.
  - 3.1. Navigate to `https://git.lab.example.com` and log out by selecting the profile icon and then **Sign out** in the left panel. Log in again as the student user using `StuD3nt123!@` as the password.
  - 3.2. Navigate to **Projects** → **Yours** → **Student User/playbooks**, and then navigate to **Settings** → **Webhooks** in the left panel. Note that the left panel may not be present until you move your mouse cursor to the left side of the page.
  - 3.3. On the webhook configuration page, select **Add new webhook** and create the webhook by adding the **WebhookURL** and **Webhook Key** values, obtained from



automation controller in the previous step, to the **URL** and **Secret token** fields, respectively.

#### URL

`https://controller.lab.example.com/api/v2/job_templates/11/gitlab/`

URL must be percent-encoded if neccessary.

#### Secret token

`vJygx4F7PaCGD6g4D57y3hruzYmxqJWmvSMCMvS7COrIKQD7Zp`

Use this token to validate received payloads. It is sent with the request in the X-Gitlab-Token HTTP header.

#### Trigger

☒ Push events

### Important

The preceding screen capture is only an illustration. Use the URL and Secret token values for your own environment. Use the **Webhook URL** and **Webhook Key** values that you generated in your automation controller UI for your URL and Secret token fields.

3.4. Select **Push events** and clear **Enable SSL verification**.

### Note

Normally, you would not disable SSL verification. You would instead ensure that both automation controller and GitLab could validate each other's TLS/SSL certificates.

3.5. Click **Add webhook** to save the webhook.

3.6. Scroll down to the bottom of the page. In the **Project Hooks** section, notice that you now have one webhook configured. Click **Test** and choose **Push events** to test the webhook. You should see the **Hook executed successfully** message.

4. Test the webhook by making a change to the `code-collection` Git repository.

4.1. As the student user on the workstation machine, create the `/home/student/git-repos` directory if it does not already exist, and then change into it.

```
[student@workstation update_controller]$ cd ~/git-repos/
```

4.2. Clone the `playbooks.git` repository and then change into the cloned repository:

```
[student@workstation git-repos]$ git clone \
https://git.lab.example.com/student/playbooks.git
Cloning into 'playbooks'...
...output omitted...

[student@workstation git-repos]$ cd playbooks
```

4.3. Edit the `install-iis.yml` file and change the “Hello World!” string to “Content changed!”. Leave all other automation code as it is. The content of the changed task should contain the following before saving the file.

```
[student@workstation playbooks]$ vi install-iis.yml

...content omitted...
- name: Website index.html created
  ansible.windows.win_copy:
    content: "Content changed!"
    dest: C:\Inetpub\wwwroot\index.html
```

4.4. Commit and push the changes to the remote Git repository.

```
[student@workstation playbooks]$ git add install-iis.yml

[student@workstation playbooks]$ git commit -m "Changed web content"
[main d164229] Changed web content
1 file changed, 1 insertion(+), 1 deletion(-)

[student@workstation playbooks]$ git push -u origin main
Password for 'https://student@git.lab.example.com': StuD3nt123!@
...output omitted...
branch 'main' set up to track 'origin/main'.
```

4.5. In the automation controller web UI, navigate to **Views** → **Jobs**.

4.6. Your `git push` command launched the `Install IIS` job template and started the jobs `Source Control Update` and `Playbook Run`.

4.7. After the `Playbook Run` job is successful, navigate to `http://win1.example.com` to see that the web content has changed.

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish playbooks-run
```