LAB 4.1: CREATE A PERSONAL GROUP

Keyboard Time: 1 mins, **Automation Wait Time**: 0 mins

Scenarios: Instructor-Led, Self-Paced

> ⚠️ **Warning**   ❯
>
> Skip this lab if you already have a personal group created by a previous lab.

1. While in 'classgroup', near the top right of the page, *Click* **New subgroup** (button)

2. Name the group with the mask `firstname_lastname` 📋 so that it will be unique, easy to remember and easy for others to identify. (For example if your gitlab user id is @supercoolcoder and your avatar URL is https://gitlab.com/supercoolcoder, name your subgroup 'supercoolcoder'). From here on in the exericses this will be referred to as 'yourpersonalgroup'

3. *Click* **Create Group**.

4. On the left hand navigation *Click* **Settings**

5. Next to "General", *Click* **Expand**

6. For "Visibility Level", *Check* **Public**.

> ⚠️ **Must Be Public**   ❯
>
> Projects that are used by the GitLab Agent must be public when the agent registration is done in a project other than the one the deployment happens from and when the image being sourced is not using a stored docker login secret.

7. **Record or remember** 'yourpersonalgroup' =

_____

> ⚠️ **IMPORTANT** ›
>
> Throughout the remaining exercises you will replace the text
> `yourpersonalgroup` with this actual group name.

Workshop Version: v1.3.3

LAB 4.2: PREPARE THE APPLICATION PROJECT

Keyboard Time: 20 mins, **Automation Wait Time**: 2 mins

Scenarios: Instructor-Led, Self-Paced

🎋 GitOps Convention ❯

A common GitOps convention is to seperate Application Build repositories from Environment Deployment repositories. This lab sets up the Environment Deployment repository to follow this convention.

☰ Target Outcomes ❯

1. Create a GitOps Application Build project from a template.

2. Create and publish (in a group CI/CD variable) a token that allows the Environment Deployment project to read the container images in the Application Build project.

❯ Click Here To Expand a Visual Overview of The GitLab Application Project Pipeline

⚠️ Warning ❯

Before continuning make sure to use DNSChecker.com to check if both `the Load Balancer DNS Name` 📋 and `Load Balancer IP` 📋 **.nip.io** have propagated through global DNS and wait (or troubleshoot) if they have not.

💡 Tip ❯

This project auto-increments images with a simple semantic

version (prereleases not supported). You can also force a specific version number and tell it which part of the version number to auto-increment.

1. While in 'yourpersonalgroup' (created in a prior lab) *Click* **New project** (button) and then *Click* **Import project**

2. On the 'Import project' page, *Click* **Repository by URL**

3. On the next page, for 'Git repository URL' *Paste* **https://gitlab.com/guided-explorations/gl-k8s-agent/gitops/apps/hello-world.git**

4. In 'Project name' *Type* **Hello World** (may already be defaulted to this)

5. *Scroll down* to 'Visibility Level'

6. *Click* **Public**.

> ⚠ Must Be Public   ›
>
> Projects that are used by the GitLab Agent must be public when the agent registration is done in a project other than the one the deployment happens from and when the image being sourced is not using a stored docker login secret.

7. Near the bottom of the page *Click* **Create project** (button)

8. On the left navigation bar, *Click* **CI/CD => Pipelines**

9. In the upper right of the page, *Click* **Run pipeline** (button)

10. On the 'Run pipeline page', leave the defaults and in the lower left of the page *Click* **Run pipeline** (button)

> The code is designed to increment an image version from the container registry of your project. If no image is found, it starts at version 0.0.1.

11. **[Automation wait: ~1 min]** Wait for the pipeline to complete successfully.

12. On the left navigation panel, *Click* **Packages & Registries => Container Registry**

13. On the Container Registry page click the item ending in "/main"

> You should see a version tag, a git short sha tag and a latest-prod tag that all have the same value for "Digest"

14. On the left navigation bar, *Click* **CI/CD => Pipelines**

15. In the upper right of the page, *Click* **Run pipeline** (button)

16. On the 'Run pipeline page', Change the value of the variable "NEXTVERSION" to **1.0.0** (replace the text 'increment-existing-image-version')

17. *Click* **Run pipeline** (button)

18. **[Automation wait: ~1 min]** Wait for the pipeline to complete successfully.

19. On the left navigation bar, *Click* **Packages & Registries => Container Registry**

20. On the Container Registry page **click the [line item ending in "/main"]**

> Among the tags you should see a 0.0.1 version tag and a 1.0.0 version tag.
>
> Note: the next time the pipeline runs it will increment 1.0.0 to 1.0.1 automatically because it will read the current version from the latest image.

💡 Created To Be A Template   ❯

This source project followed several specific principles that makes it this easy to use as a template:

1. Soft codes most of the paths to be self referential to the

project path name.

2. When looking in the project container registry for the last image version, it handles an error in reading the container image as an indicator this is the first run and then forces the version number to 0.0.0 before it is incremented and a new container is generated. Subsequent builds then find the latest-prod image.

3. Avoids use of the 'latest' tag as it is very common and might be created by other unknown build processes that are integrated later.

Create a Token To Read The Container Registry
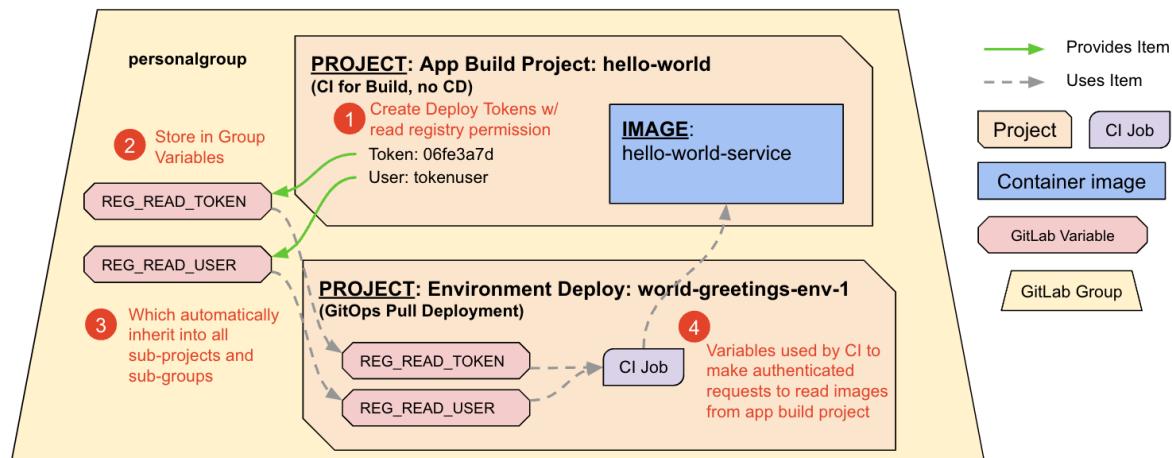
💡 Group or Project Access Tokens For High Trust Group Hierarchies ❯

For the next steps you will create a Project Deployment Token so that the Environment Deployment project can read the container image of this Application Build project. If you have a paid GitLab license, a project or group level 'Access Token' can give the same access to all container registries in a group heirarchy. This works well if there is high trust between all Application Build and Environment Deployment projects as there are fewer credentials granted at an appropriate level.

Registry Read Token Overview

Real World: In the real world one App Build project may be referenced by 10's or 100's of Environment Deploy projects - so envision many Environment Deployments to see why the steps of creating project level variables for the token and user would scale nicely.

Least Privileged by Design: a "least privileged" security model is reflected in the above choices of what level to create the credentials at (personal access token, project, parent group, higher level group) and what level to create the CI/CD Variables that contain the credentials at (project, enclosing group, higher level group).

1. While in 'yourpersonalgroup/hello-world' in the left navigation, *Click* **Settings => Repository** (button)

2. Next to 'Deploy tokens' *Click* **Expand**

3. Under 'New deploy token', for Name, *Type* **ReadContainerRegistry**

4. Under 'Scopes (select at least one)', *Select* **read_registry** (**DO NOT SELECT** read_repository)

5. *Click* **Create deploy token**

> ⚠️ Page Reloaded With Update - Don't Close ❯
>
> Notice the same page reloads, but at the top of the screen now has a grey box containing the token information.
>
> **IMPORTANT** - Do not navigate to another page in this browser as this is the only time you can see the token. You will have to create a new token if you leave the page.

6. Use copy and paste to record the following in a temporary document (do not hand type tokens):

    ○ READ_REG_USER = **[user id from token generation UI]**

◦ READ_REG_TOKEN = [token from token generation UI]

7. **In a NEW browser tab**, open 'yourpersonalgroup' (the group level - not the hello-world project)

☠ Verify Your Location ❯

It is very easy to accidentally create these at the project level. The **token** is created in the **project**, but the variables MUST be at the **GROUP** level for them to be visible to the Environment Deployment project you will create in the next lab.

8. On the left navigation, *Click* **Settings => CI/CD**

9. To the right of 'Variables', *Click* **Expand**

10. *Click* **Add variable**

11. For Key, *Type* **READ_REG_USER** (it is usually best to copy this name from this exercise rather than type it)

12. Copy the Value for READ_REG_USER from your temporary document, *Paste* **[the Clipboard contents]**

13. Under Flags, *Deselect* **Protect variable**

14. *Click* **Add variable** (button)

15. *Click* **Add variable** (button on page - this is not a duplicate instruction)

16. For Key, *Type* **READ_REG_TOKEN** (it is usually best to copy this name from this exercise rather than type it)

17. Copy the Value for READ_REG_TOKEN from your temporary document, *Paste* **[the Clipboard contents]**

18. Under Flags, *Deselect* **Protect variable**

19. Under Flags, *Select* **Mask variable**

20. *Click* **Add variable**

You should now have two variables in 'yourpersonalgroup'

that contains READ_REG_USER and READ_REG_TOKEN with the values from the Deploy Token creation.

> ✅ **Accomplished Outcomes** ❯
>
> 1. Create a GitOps Application Build project from a template.
>
> 2. Create and publish (in a group CI/CD variable) a token that allows the Environment Deployment project to read the container images in the Application Build project.

Workshop Version: v1.3.3

❮ ❯

LAB 4.3: PREPARE THE ENVIRONMENT DEPLOYMENT PROJECT

Keyboard Time: 25 mins, **Automation Wait Time**: 5 mins

Scenarios: Instructor-Led, Self-Paced

⤴ GitOps Conventions ❯

1. A common GitOps convention is to seperate Application Build repositories from Environment Deployment repositories. This lab sets up the Environment Deployment repository to follow this convention.
2. Completely constructed manifests are stored in a repository for easy human reading, visibility and source control managed state. This project creates such manifests under two conditions:
   1. when the source code of this project is altered,
   2. when a new version of the Application Build container is detected or specified.

☰ Target Outcomes ❯

1. Create a GitOps Environment Deployment project from a template.

2. Configure it to monitor the Application Build project for new images.

3. Create a token so that the CI job can write back the constructed manifests back to it's own project.

4. Do a dry run to see if the manifests update as expected.

› Click Here To Expand a Visual Overview of The GitLab Environment Deployment Project Pipeline

1. While in 'yourpersonalgroup' (created in a prior lab) *Click* **New project** (button) and then *Click* **Import project**

2. On the 'Import project' page, *Click* **Repository by URL**

3. On the next page, for 'Git repository URL' *Paste* **https://gitlab.com/guided-explorations/gl-k8s-agent/gitops/envs/world-greetings-env-1.git**

4. In 'Project name' *Type* **World Greetings Env 1** (likely already be defaulted to this)

5. *Scroll down* to 'Visibility Level'

6. *Click* **Public**.

   > ⚠️ Must Be Public   ›
   >
   > Projects that are used by the GitLab Agent must be public when the agent registration is done in a project other than the one the deployment happens from and when the image being sourced is not using a stored docker login secret.

7. Near the bottom of the page *Click* **Create project** (button)

8. When the import is complete, you will be placed in the default landing page of the project.
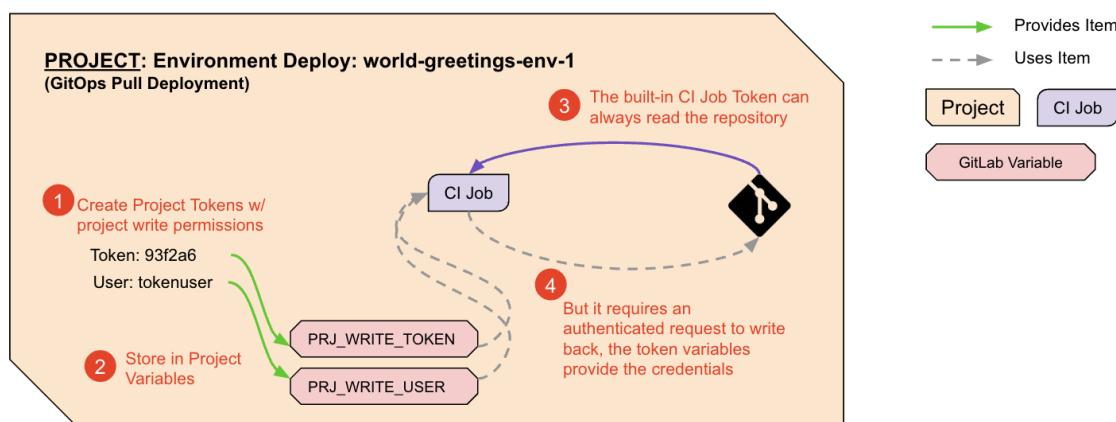
   > 💡 Group or Project Access Tokens For High Trust Group Hierarchies   ›
   >
   > For the next steps you will create a Token so that the Environment Deployment project can read the container image of this Application Build project. If you have a paid GitLab license, a group level 'Access Token' can give the same access to all container registries in a group heirarchy. This works well if there is high trust between all Application Build and Environment Deployment projects as there are

fewer credentials granted at an appropriate level.

1. **Instructor-Led Classroom:** Please ask the instructor whether to use **Token Option 1: Using a Project Access Token (Paid Licenses Only Feature)** or **Token Option 2: Using a Personal Access Token (PAT)**

2. **Self-paced:** Try **Token Option 1: Using a Project Access Token (Paid Licenses Only Feature)** first. If your instance does not have a specific GitLab paid license feature you will be directed to **Token Option 2: Using a Personal Access Token (PAT)**.

Project Write Token Overview

Real World: For push gitops our "deploy" is simply to write kubernetes manifests to a git repository and then the agent polls for changes. Hence our "deploy jobs" have to write updated manifests back to their own repository so they can be pulled.

Least Privileged by Design: a "least privileged" security model is reflected in the above choices of what level to create the credentials at (personal access token, project, parent group, higher level group) and what level to create the CI/CD Variables that contain the credentials at (project, enclosing group, higher level group).

IMPORTANT: Requires a paid GitLab license, even an ultimate trial will not have the Project Level Access Token Feature. The project menu choice **Settings => Access Tokens** will not exist. You will need to use "Token Option 2" below.

1. While in 'yourpersonalgroup/world-greetings-1' (created in a prior lab), on the left navigation, *Click* **Settings => Access Tokens**

   > This menu option will not exist if you do not have a paid GitLab license. You will need to use "Token Option 2" below.

2. Under 'Add a project access token', for Token name, *Type* **WriteRepository**

3. Under 'Select a role', *Select* **Maintainer**

4. Under 'Select scopes'

   1. *Select* **read_repository** (**DO NOT SELECT** read_registry)
   2. *Select* **write_repository**(**DO NOT SELECT** write_registry)

5. *Click* **Create project access token** (button)

> ⚠️ Page Reloaded With Update - Don't Close ＞
>
> Notice the same page reloads, but at the top of the screen now has a grey box containing the token information. **IMPORTANT** - Do not navigate to another page in this browser as this is the only time you can see the token. You will have to create a new token if you leave the page.

6. Use copy and paste to record the following in a temporary document (do not hand type tokens):

   - PROJECT_COMMIT_TOKEN = **[project access token from UI]**

7. On the left navigation, *Click* **Project Information => Members**

8. In the search prompt *Type* **WriteRepository**

> The user list should return one entry

9. In the listing, under "WriteRepository", copy the user name that starts with "project_" and ends with "_bot" - do not include the @ sign.

10. In the previous temporary document, record:

   - PROJECT_COMMIT_USER = **[the user id you just copied]**

❓ Token Option 2: Using a Personal Access Token (PAT) ›

☠ In Shared class groups Other Participants Have Access to This PAT ›

If you are in a shared classgroup environment (including an asynchonous and long lived ones where other students work at other times than yourself), this PAT will be visible to everyone who has access to classgroup. If you are using a production GitLab user id - this step will give repository read and write access to every repository your user id has access to.

1. In the upper right of the page *Click* **[your Avatar icon]** and then *Click* **Edit profile**

2. On the left naviagion, *Click* **Access Tokens**

3. Under 'Add a personal access token', for Token name, *Type* **WriteRepository**

4. Under 'Select scopes'

   1. *Select* **read_repository**
   2. *Select* **write_repository**

5. *Click* **Create personal access token** (button)

⚠ Page Reloaded With Update - Don't Close ›

Notice the same page reloads, but at the top of the screen

now has a grey box containing the token information. **IMPORTANT** - Do not navigate to another page in this browser as this is the only time you can see the token. You will have to create a new token if you leave the page.

6. Record the following in a temporary document:
   - PROJECT_COMMIT_TOKEN = **[personal access token from UI]**
   - PROJECT_COMMIT_USER = **[your gitlab user id without the '@' and without the path]**

1. **In a NEW browser tab**, open the project 'yourpersonalgroup/world-greetings-env-1' again (this time we are at the PROJECT level).

2. On the left navigation, *Click* **Settings => CI/CD**

3. To the right of 'Variables', *Click* **Expand**

4. *Click* **Add variable**

5. For Key, *Copy and Paste* **PROJECT_COMMIT_TOKEN**

6. In the Value field *Copy and Paste* **[the temporary document value for PROJECT_COMMIT_TOKEN]**

7. Under Flags, *Deselect* **Protect variable**

8. Under Flags, *Select* **Mask variable**

9. *Click* **Add variable**

10. To add another variable, *Click* **Add variable**

11. For Key, *Copy and Paste* **PROJECT_COMMIT_USER**

12. In the Value field *Copy and Paste* **[the temporary document value for PROJECT_COMMIT_USER]**

13. Under Flags, *Deselect* **Protect variable**

14. *Click* **Add variable**

Among the existing variables in the 'yourpersonalgroup/world-greetings-env-1' project, you should have the two new variables PROJECT_COMMIT_TOKEN and PROJECT_COMMIT_USER.

These permissions are least privilege, in part, because the CI/CD Variables are only published at the project level.

27. In a new browser tab, open your 'yourpersonalgroup/hello-world' Project. (**IMPORTANT: not the same project you are in now**)

28. On the left navigation panel, *Click* **Packages & Registries => Container Registry**

29. Next to the line item ending in "/main", *Click* **[the Clipboard icon]**

30. **IMPORTANT:** Switch back to 'yourpersonalgroup/world-greetings-env-1' Project

31. In the left navigation, *Click* **Repository => Files**

32. On the upper right of the Project page, *Click* **Web IDE**

33. In the files list, *Click* **.gitlab-ci.yml**

34. Under 'variables:' *Find* **IMAGE_NAME_TO_MONITOR**

35. In the quoted value, *Remove* **the existing value**

36. Paste your copied image path

> The result should be something like
> **IMAGE_NAME_TO_MONITOR: "registry.gitlab.com/somegroups/classgroup/yourpersonalgroup/hel world/main"**

37. *Click* **Create commit...**

38. *Select* **Commit to main branch** (change from "Create a new branch")

39. *Click* **Commit**

> 💡 **Wait for it** ❯
>
> The pipeline id link may take up to 30 seconds to appear as the CI job has to kick off before it displays.

40. In the very bottom left, immediately after the text 'Pipeline' *Click* **[the pipeline number which is preceeded with a #]** (Or on the left navigation *Click* **CI/CD => Pipelines** and *Click* **[the status badge]** or [pipeline #] for the latest running pipeline)

41. Expand the Downstream pipeline with the great than arrow (>).

> ⚠️ **Some Possible CI Errors** ❯
>
> Possible error messages (not an exhaustive list):
>
> ```
> level=fatal msg="authenticating creds for
> \"registry.gitlab.com\": Requesting bear token: invalid
> status code from registry 403 (Forbidden)"
> .
>
> level=fatal msg="error logging into
> \"registry.gitlab.com\": invalid username/password"
> ```
>
> can be caused by:
>
> - a badly formed value in IMAGE_TO_MONITOR in world-greetings-env-1/.gitlab-ci.yml
> - the variables READ_REG_USER and READ_REG_TOKEN
>   - being incorrectly named,
>   - at the wrong group level or

- having invalid values or accidental swapping of the values (e.g. User Id in READ_REG_TOKEN)

- READ_REG_USER starting with `@` (this should be left off)

- having incorrect permissions in the token (should be "read_registry", not "read_repository")

The error message:

remote: GitLab: You are not allowed to push code to protected branches on this project. ! [remote rejected] main -> main (pre-receive hook declined)

error: failed to push some refs to

can be caused by:

- You did not select 'Maintainer' for the token role when setting up the repository write token above.

42. **[Automation wait: ~3 min]** Watch the pipeline complete through the 'update-staging-manifests' job.

43. The update-staging-manifests job should complete successfully.

44. To get back to the Web IDE, *Click* **[the browser back button]**

45. *Click* **[the browser refresh button]**

46. In the files list on the left *Click* **manifests > hello-world.staging.yaml**

47. *Search* for **- image:**

48. The image reference should be the registry pointer to your Application Project, followed by the latest-prod image version ("1.0.0" if you only built the Application Project twice, maybe higher if you did more builds)

🔖 **Only Staging Manifest Has Changed So Far** ❯

In the next steps you will observe that the production manifest has not change yet because you have not approved the deployment to production yet.

51. In the files list on the left *Click* **manifests > hello-world.production.yaml**

52. *Search* for **- image:**

53. The image version tag does not match staging. (If all labs were done as described it should say
`registry.gitlab.com/_replace-with-hello-world-service-container-regi`
📋)

54. **In a NEW browser tab**, open 'yourpersonalgroup/world-greetings-env-1' again.

    > Shortcut - right click the project heading in the left navigation and *Click* **Open Link in New Tab**)

55. In the left navigation *Click* **CI/CD => Pipelines**

56. Find the last non-skipped pipeline and *Click* it's **[Status badge]** or **[Pipeline #]** to open the pipeline.

57. Expand the Downstream pipeline with the great than arrow
    (>).

    > NOTE: Depending on your screen width, you may need to use the horizontal scroll bar under the pipeline to find the update-production-manifests job.

58. Next to the update-production-manifests job, *Click* **[the play button]**

59. **[Automation wait: ~1 min]** Wait until the update-production-manifests job has a green check next to it.

60. In the browser tabs, *Switch* back to **[the Web IDE tab]**

61. *Click* **[the browser refresh button]**

62. In the files list on the left *Click* **manifests > hello-world.production.yaml**

63. *Search* for **- image:**

64. The image reference and version tag should match the staging manifest (hello-world.staging.yaml) which should be the latest-prod tagged image in the Hello World Application Build project.

> 💡 **Tip**   ❯
>
> The manifests are not yet monitored by the GitLab Agent, but once they are, the action of updating them in the project is all that is necessary for the GitLab Agent to find them and update the Kubernetes Cluster to match the manifest.

> 💡 **Created To Be A Template**   ❯
>
> This source project followed several specific principles that makes it this easy to use as a template:
>
> 1. Soft codes most of the paths to be self referential to the project path name.
> 2. When looking in the project container registry for the last image version, it relies on the exact named variables for the authentication token that were setup as part of the Application Build project.
> 3. It relies on several of the variables that were configured at the classgroup level for the Kubernetes Agent integration.

> ✅ **Accomplished Outcomes**   ❯
>
> 1. Create a GitOps Environment Deployment project from a template.

2. Configure it to monitor the Application Build project for new images.

3. Create a token so that the CI job can write back the constructed manifests back to it's own project.

4. Do a dry run to see if the manifests update as expected.

Workshop Version: v1.3.3

LAB 4.4: LINK AND TEST PROJECTS

Keyboard Time: 5 mins, **Automation Wait Time**: 3 mins

Scenarios: Instructor-Led, Self-Paced

:≡ Target Outcomes ›

1. Link the projects using a scheduled pipeline.

💡 Tip ›

These methods of linking the projects are loosely coupled. The benefits of this approach are described in
Loose Project Coupling

Scheduled Pipeline Model

1. In 'yourpersonalgroup/world-greetings-env-1' *Click* **CI/CD => Schedules**

2. On the upper right of the page, *Click* **New schedule** (button)

3. Under Description *Type* **CheckForNewContainerVersion**

4. Leave all other items at their defaults.

5. Near the bottom left of the page, *Click* **Save pipeline schedule**

    > Normally you would take time to create one or more schedules specific to your desired frequency.

6. On the right of CheckForNewContainerVersion *Click* **[the play button]**

7. On the left navigation, *Click* **CI/CD => Pipelines**

8. On the latest pipeline *Click* **[the Status badge]** or **[the pipeline #]**

9. Wait for the pipeline to complete.

10. If you did not perform any extra builds on the Application Project, the "deploy" job will have a failed status and the pipeline will have a status of "Passed"

> ⚠️ Failed 'deploy' job is OK ❯
>
> The deploy job status is 'failed' because no child pipeline jobs are scheduled because there has not been a new container published since the last run (and no changes were made to the Environment Deployment Project package manifests). GitLab considers it a failure when a parent pipeline fails to create a child pipeline, but we've marked this job "allowed_to_fail" which gives the Pipeline status of "Passed" (because it is the most efficient way to only run the manifest builds when there is actually a change we care about.)

> ✅ Accomplished Outcomes ❯
>
> 1. Link the projects using a scheduled pipeline.

> ⚠️ Instructor Led Classroom ❯
>
> If you are in an Instructor-Led course, do not do **[Extra Credit]** exercises after this point.

[Extra Credit] Pipeline Subscription Model

> ⚠️ Instructor Led Classroom ❯
>
> If you are in an Instructor-Led course, do not do **[Extra Credit]** exercises including this one.

> Pipeline subscriptions allow an Environment Deployment Project to trigger nearly immediately after the Application Project completes a build.

1. Open 'yourpersonalgroup/hello-world'

2. In the browser URL bar, copy the URL path without the domain. For example if the browser url is https://gitlab.com/group1/myuser/hello-world you would copy 'group1/myuser/hello-world '

3. **In a NEW browser tab**, open 'yourpersonalgroup/world-greetings-env-1'

4. *Click* **Settings => CI/CD**

5. Next to Pipeline subscriptions, *Click* **Expand**

6. Under Project path, *Paste* **[the path copied from the hello-world project]**

7. *Click* **Subscribe**

8. Next to Pipeline subscriptions, *Click* **Expand**

[Extra Credit] Run Pipeline With NEXTVERSIONTOUSE Variable To Specify Version

> ⚠️ Warning   ❯
>
> **If you are in an instructor-led workshop, please ask the instructor before performing this lab** as it could affect workshop timing or the stability of additional assigned labs. If used in production this method would not be paired with any auto-update mechanism above because that mechanism would dynamically install the latest.

> This method can also be used to roll back an environment.

1. Open 'yourpersonalgroup/world-greetings-env-1'
2. On the left navigation, *Click* **CI/CD => Pipelines**
3. In the upper right of the page, *Click* **Run pipeline** (button)
4. Under Variables, *Type* **NEXTVERSIONTOUSE** over 'Input variable key'
5. On the same line, *Type* **0.0.1** over 'Input variable value'
6. In the lower left of the page *Click* **Run pipeline** (button)
7. Wait for the pipeline to complete successfully.
8. **In a NEW browser tab**, open 'yourpersonalgroup/world-greetings-env-1' in the Web IDE.
9. In the files list on the left *Click* **manifests > hello-world.staging.yaml**
10. *Search* for **- image:**
11. The image reference should point to the version "0.0.1"
12. In the left navigation *Click* **CI/CD => Pipelines**
13. Find the last non-skipped pipeline and *Click* it's **[Status badge]** or **[Pipeline #]** to open the pipeline.
14. Expand the Downstream pipeline with the great than arrow (**>**).
15. Next to the update-production-manifests job, *Click* **[the play button]**
16. Wait until the update-production-manifests job has a green check next to it.
17. *Switch* back to **[the Web IDE tab]**
18. *Click* **[the browser refresh button]**
19. In the files list on the left *Click* **manifests > hello-world.production.yaml**
20. *Search* for **- image:**
21. The image reference and version tag should match the staging manifest ("0.0.1")

[Extra Credit] Create an MR with NEXTVERSIONTOUSE File To Specify Version

> ⚠️ Warning   ⟩
>
> **If you are in an instructor-led workshop, please ask the instructor before performing this lab** as it could affect workshop timing or the stability of additional assigned labs. This section is just to let you know that you can create a Merge Request that creates or updates a file called NEXTVERSIONTOUSE that only contains the desired version on the first and only line in the file. This enables MR review by as many people as necessary to gather approvals before environment deployments are performed. If you have previously

environment deployments are performed. If you have previously

done MRs in GitLab, you could do this procedure to experience an MR approval based workflow in an Environment Deployment Project.

Workshop Version: v1.3.3

LAB 4.5: SETUP THE GITOPS CD PULL AGENT

Keyboard Time: 10 mins, **Automation Wait Time**: 5 mins

Scenarios: Instructor-Led, Self-Paced

⑂ GitOps Conventions ❯

1. Monitoring manifests by an agent running in a Kubernetes cluster. This lab configures the GitLab Agent to monitor the manifests in this repository.

☰ Target Outcomes ❯

1. Configure the Kubernetes Agent to monitor the CI constructed kubernetes manifests.

2. Observe the initial deployment of staging and production via tailing the Kubernetes Agent log and the appearance of the target environments.

⚠ Warning ❯

For instructor-led classes, this portion will be done by the instructor.

If you are not in an instructor-led course, perform the lab as described.

❞ Done By Instructor for Instructor-Led Courses ❯

1. Logon the cluster administration machine =>
   Instructions for SSM Session Manager for EKS

2. Run the following command to tail the kubernetes agent
   log while deployments are happening:

```
kubectl logs -f -l=app=gitlab-agent -n gitlab-agent
```

Leave this view open as you will be instructed to consult it
to see the deployment logging activity when the GItLab
Agent pulls and processes the kubernetes manifest.

3. In a web browser *Navigate to* **classgroup/cluster-
   management**

4. Near the upper right of the page, *Click* **Web IDE** (button)

5. Navigate to the file .gitlab/agents/spotazuseast2-
   agent/config.yml

6. Add the following to the file only once:

```
gitops:
   manifest_projects:
```

7. Under "gitops:manifest_projects:" add as below - replacing
   `_classgroup_` and `_yourpersonalgroup_` with the
   actual names for your project. Ensure indenting and
   "gitops:manifest_projects" should only appear once in the
   entire file.

For Instructors: add one of these sections per participant.
Ensure indentation is perserved.

```
  - id: _classgroup_/_yourpersonalgroup_/world-greetings-
  env-1
    default_namespace: default
    paths:
    - glob: '/manifests/**/*.yaml'
```

```
    reconcile_timeout: 3600s # 1 hour by default

    dry_run_strategy: none # 'none' by default
    prune: true # enabled by default
    prune_timeout: 360s # 1 hour by default
    prune_propagation_policy: foreground # 'foreground' by
default
    inventory_policy: must_match # 'must_match' by default
```

8. Final result should be something like this (including indentation - with repeating "id" sections for each participant if in a classroom):

```
gitops:
  manifest_projects:
  - id: _classgroup_/_yourpersonalgroup_/world-greetings-
env-1
    default_namespace: default
    paths:
    - glob: '/manifests/**/*.yaml'
    reconcile_timeout: 3600s # 1 hour by default
    dry_run_strategy: none # 'none' by default
    prune: true # enabled by default
    prune_timeout: 360s # 1 hour by default
    prune_propagation_policy: foreground # 'foreground' by
default
    inventory_policy: must_match # 'must_match' by default
```

9. *Click* **Create commit...**

10. *Select* **Commit to master branch** (change from "Create a new branch")

11. *Click* **Commit**

11. Watch the previously opened view of the GitLab Agent log for deployment activity.

    For Instructor-Led: the instructor may have this view displayed for everyone

12. To watch the progress, navigate to
    *classgroup/yourpersonalgroup/***world-greetings-env-1**

13. *Click* **Deployments => Environments**

14. **[Automation wait: ~3 min]** Keep refreshing until staging deployment activities complete.

> ⚠️ Warning   ›
>
> For all GitOps mode projects, when the deployment shows complete in the Environments page, it only means the manifests are completely setup, the Gitlab Agent for Kubernetes still has to find and deploy the changed manifests

15. **[Automation wait: ~3 min]** Wait after the status shows complete…

16. On the 'staging' line, to the right, *Click* **Open**

> You can see the staging deployed application.

17. In the browser tabs, *Click* **[the tab with the Environments page]**

18. On the 'production' line, to the right, *Click* **Open**

> You can see the production deployed application

19. If there an error indicating there is no site yet, keep refreshing the browser window until the site displays.

> ⚠️ Warning   ›
>
> For all GitOps mode projects, when the deployment shows complete in the Environments page, it only means the manifests are completely setup, the Gitlab Agent for Kubernetes still has to find and deploy the changed manifests. Also note that on the very first time the agent is configured to monitor your manifests - all environments are deployed. From this point forward the manifests will be updated sequentially and will require approval for production.

☠ **Critical Mindfulness: Only Pull Deployment In Environment Deployment Project** ❯

Subsequent labs will be adding many Runner Based jobs to enable security scanning and dynamic environments. However, the deployment of this application will always be accomplished by a Pull Deployment through the GitLab Agent as you have seen in this lab. You may consult the job log (to see a manifest commit only) and/or the Kubernetes Agent log to verify this.

✅ **Accomplished Outcomes** ❯

1. Configure the Kubernetes Agent to monitor the CI constructed kubernetes manifests.

2. Observe the initial deployment of staging and production via tailing the Kubernetes Agent log and the appearance of the target environments.

Workshop Version: v1.3.3

‹   ›

LAB 4.6: UPDATE THE APPLICATION BUILD PROJECT AND DEPLOY TO PRODUCTION

Keyboard Time: 20 mins, **Automation Wait Time**: 5 mins

Scenarios: Instructor-Led, Self-Paced

In this Lab you will update the background color of the application and track the progress of the automation through both repositories and both environments.

Target Outcomes

Observe an end-to-end application change:

1. Update the background color of the application

2. Track the progress of the automation through the Application Build project and

3. Through both environments of the Environment Deployment project using the background page color.

1. Open 'yourpersonalgroup/hello-world'

2. In the left navigation, *Click* **Repository => Files**

3. On the upper right of the Project page, *Click* **Web IDE**

4. *Navigate to* the file **src/microwebserver.py**

5. Around line 11, *locate* the text

```
<BODY style="background:lightsalmon"> 
```

6. Change the color after the word `background:` 📋 to `lightgreen` 📋

   Result: `<BODY style="background:lightgreen">` 📋

   > If you need a different color, other available color values
   > are listed here

7. *Click* **Create commit...**

8. *Select* **Commit to main branch** (not selected by default)

9. *Click* **Commit**

10. In the very bottom left, immediately after the text 'Pipeline'
    Click **[the pipeline number which is preceeded with a #]**

11. **[Automation wait: ~2 min]** Wait for the pipeline to complete.

12. *Click* **Packages & Registries => Container Registry**

13. *Click* **[the line ending in '/main']**

14. Scan for the latest-prod tag

    > It should have been built moments ago. There should also
    > be a new version tag with the same value for 'Digest'

    > ✅ Accomplished Outcome   ❯
    >
    > You just observed the automatic creation of a new
    > production ready container based on the normal
    > development activity of changing the application files.

15. Open 'yourpersonalgroup/world-greetings-env-1' project.

16. *Click* **CI/CD => Schedules**

17. On the right side of schedule called
    'CheckForNewContainerVersion', *Click* **[the play button]**

18. On the left navigation, *Click* **CI/CD => Pipelines**

19. Open the most recent non-skipped pipeline by clicking **[the pipeline Status badge]** or **[the pipeline #]**

20. Expand the Downstream pipeline - next to the deploy job, *Click* **[the small right arrow]**

21. **[Automation wait: ~3 min]** Wait for the 'update-staging-manifests' job to complete successfully.

22. In the pipeline, *Click* **update-staging-manifests**

23. Search the job log (manually or with your brower's 'in page search' feature) for the text "Changes to be committed" (near the bottom)

> ✅ Observation   ❯
>
> This job only did a commit back to the World Greetings Environment 1 project - it did not do any CD push operations. Since we are also using CI processes in this workshop it can be easy to mistakenly think this job pushed the changes, rather than the GitLab Agent in the cluster pulling them.

24. In the left navigation, *Click* **Repository => Files**

25. In the main page body, in the files and directories list, *Click* **manifests**

26. *Click* **hello-world.staging.yaml**

27. *Find* `- image:` 📋

28. Note the version number at the very end of the image string should match the image registry version you just saw. Keep this version in mind so you can compare to production in the

29. *Click* **[the browers back button]**

30. *Click* **hello-world.production.yaml**

31. *Find* `- image:` 📋
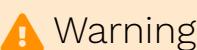
> 💡 Tip   ❯
>
> The version differences between the current state of these two manifests is what explains the results you will see when viewing the active environments in the next steps.

32. *Click* **Deployments => Environments**

33. **[Automation wait: ~3 min]** Keep refreshing until staging deployment activities complete.

> ⚠️ Warning   ❯
>
> For all GitOps mode projects, when the deployment shows complete in the Environments page, it only means the manifests are completely setup, the Gitlab Agent for Kubernetes still has to find and deploy the changed manifests.

> ⚠️ Warning   ❯
>
> **If you are in an instructor-led workshop, the instructor may need to access the cluster for you.** If you were to run into unusual deployment problems, you would need to login to the Kubernetes Cluster and run the below command. To do this, login to the EKS Bastion host the same was as was done in "Prep Lab 2.3: Use GitLab K8s Agent to Integrate The Cluster with GitLab" to install the GitLab Agent. Then run this command
>
> `kubectl logs -f -l=app=gitlab-agent -n gitlab-agent` 📋
>
> For common errors and more troubleshooting information visit Troubleshooting the GitLab agent for Kubernetes

34. On the 'staging' line, to the right, *Click* **Open**

> You should see that the staging environment is now the new color.

35. To approve the production deployment, in the left navigation, *Click* **CI/CD => Pipelines**

36. Open the most recent non-skipped pipeline by clicking **[the pipeline Status badge]** or **[the pipeline #]**

37. Next to the deploy job, *Click* **[the small right arrow]**

38. *Locate* the **update-production-manifests** job

> You may have to horizontally scroll right to see this final job.

40. *Click* **[the play button in a circle]**

41. **[Automation wait: ~3 min]** Keep refreshing until production deployment activities complete.

> ⚠️ Warning ❯
>
> For all GitOps mode projects, when the deployment shows complete in the Environments page, it only means the manifests are completely setup, the Gitlab Agent for Kubernetes still has to find and deploy the changed manifests.

42. *Click* **Deployments => Environments**

43. **[Automation wait: ~3 min]** Keep refreshing until staging deployment activities complete.

44. On the 'production" line, to the right, *Click* **Open**

> ✅ Observation ❯
>
> You should see that the production environment is now the new color.

💡 Tip ❯

While it is not necessarily easy to observe directly from GitLab, it is the GitLab Agent that is pulling the changes into the cluster. You can understand more about this flow by examining the box 'GitLab K8s Agent Channel' in the GitLab K8s Agent Connections and Flows diagram.

✓ **Accomplished Outcomes** >

1. Update the background color of the application

2. Track the progress of the automation through the Application Build project and

3. Through both environments of the Environment Deployment project using the background page color.

Workshop Version: v1.3.3

‹   ›