LAB 5.1: ADD ENVIRONMENTS AND SECURITY SCANNING TO THE APPLICATION PROJECT

Keyboard Time: 20 mins, **Automation Wait Time**: 10 mins

Scenarios: Instructor-Led, Self-Paced

GitLab Security Scanning and Automated Review Environments

GitLab’s robust Security Scanning can be configured for GitOps Pull Deployment projects. This adds Security scan results to developer MRs and to GitLab Security Dashboards. Adding Dynamic Review Environments not only enables specific types of security scanning that require an active version of the application - it also enable developers to visually verify application changes with no impact to production clusters.

Target Outcomes

Add security scanning and dynamic review environments. Per-branch dynamic environments are required for all scanning types that need an active copy of the application - DAST, IAST, Fuzzing and Accessibility Auditing all fall in this category. Dynamic environments can also be used for human review and testing.

1. Use the Pipeline Editor and its Lint feature.
2. Update the Application Build project to add Auto DevOps (Runner Push Deployment) with the GitLab Agent cluster connection method.

3. Track the progress of the build through security scanning, staging environment and ‘production’ environment. (Production in this case means that the Application container is production ready.)
4. Through both environments of the Environment Deployment project using the background page color.

Tip

This configuration also works for any kind of **GitLab Runner Push CD** to the cluster using Helm and kubectl commands, not only Auto DevOps.

In this Lab you will update the application project to have a review application and perform security scanning.

1. Open ‘yourpersonalgroup/hello-world’
2. In the left navigation, *Click* **CI/CD => Editor**

   > You will be editing YAML - be careful that tabbing is properly aligned. Only removing the comment character (“#”) should result in proper tabbing.

3. Under `include:` uncomment `- template: Auto-DevOps.gitlab-ci.yml` which should make the section look like this.:

```
include:
  - local: .gitlab/ci_includes/increment_semver.gitlabci.yml
  - template: Auto-DevOps.gitlab-ci.yml
```

4. Under `variables:` uncomment the variables indicated below at the top of the section. There will be additional variables in the section - starting with `NEXTVERSION:` - that must be left as-is

```
variables:
  BUILD_DISABLED: 'true'
  TEST_DISABLED: 'true'
  POSTGRES_ENABLED: 'false'
  CODE_QUALITY_DISABLED: 'true'
  MANUAL_PROMOTE: 'true'
  STAGING_ENABLED: '' #empty disables staging
  DAST_DISABLED: 'true'
  BROWSER_PERFORMANCE_DISABLED: 'true'

  NEXTVERSION: 'read-from-registry'
   ...
```

5. Immediately above `determine-version:`, uncomment the following segment. The keyword `container_scanning` should start completely to the left edge and each level indented 2 spaces.

```
container_scanning:
  variables:
    GIT_STRATEGY: fetch
```

6. Above the editing area, *Click* the word **Validate**
7. Near the bottom center of the page, *Click* **Validate pipeline** (button)

> You should have a green banner that says “Syntax is correct”

Observe

You will now purposely insert a syntax error to see the linter catch it. The GitLab CI Yaml linter only exists in this CI Editor.

8. In the same top navigation area, *Click* **Edit**
9. On a **blank** line *Type* **this is an error**
10. Above the editing area, *Click* the word **Validate**

11. Near the bottom center of the page, *Click* **Validate pipeline** (button)

    > You should have a red banner that says “Syntax is incorrect”

12. Make a mental note of the line number and column noted in the grey box.
13. In the same navigation area, *Click* **Edit**
14. Find the line number and *Delete* **this is an error**
15. Above the editing area, *Click* the word **Validate**
16. Near the bottom center of the page, *Click* **Validate pipeline** (button)

    > You should have a green banner that says “Syntax is correct”

17. In the same navigation area, *Click* **Edit**
18. *Click* **Commit changes**

Observe

These simple steps enable Auto DevOps Dynamic Review Environments and Security Scanning.

Web Based Pipeline Editor

The pipeline editor runs on the GitLab server which gives it the advantages of having the production linter and the ability to create a merged YAML view that shows the final YAML with all includes processed. Other code editors do not have these capabilities because they are not integrated with the GitLab instance. The pipeline editor does not yet allow editing of includes that are in the same project.

15. In the left navigation *Click* **Hello World** (The project name banner)

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

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

18. **[Automation wait: ~7 min]** While you wait for the GitLab deployment to staging to complete, notice these things:

> Observe
>
> 19. There are multiple security tests under the "Test" stage.
> 20. A special environment is deployed by the "Review" stage for DAST testing by the "Dast" stage.
> 21. The "Cleanup" stage will run to tear down the dast environment even before you approve the production deployment
> 22. The "promote-image-to-latest-prod" job no longer runs automatically, but will require a manual play button activation.

19. *Click* **Deployments => Environments**

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

> If SSL is not yet resolving, click the "advanced" option and open the site anyway.

21. In the browser tabs, *Click* **[the tab with environments]**

22. Return to the same pipeline (Browser Back Control may take you there)

23. For the job 'production_manual', *Click* **[the Play button]**

> Observe
>
> Removing or commenting the STAGING_ENABLED variable will eliminate the staging environment step if it is unneeded for your

process.

Info

The environments in the Application Build Project are built using CD Push through the GitLab Agent and are only for testing and qualifying the Application Container. The environment called 'Production' simply means that the container is "production ready" if it passes all tests and gets into that review environment. The 'promote-image-to-latest' job is what actually signals downstream Environment Deployment Projects that a new version has been published. The DevOps methodology and processes you use should determine how much testing and qualification is done in this Application Build Project and how much is done in Environment Deployment Projects. There could be a lot of testing in both if downstream environments want to double check or have additional checks before deploying to actual production endpoints.

24. **[Automation wait: ~3 min]** Wait for the browser_performance job to complete successfully and the Play button to appear on the 'promote-image-to-latest' job.
25. **IMPORTANT:** For the job 'promote-image-to-latest' *Click* **[the Play button]**
26. *Click* **Packages & Registries => Container Registry**
27. *Click* **[the line ending in '/main']**
28. Search for the latest-prod tag

> It should have been built in the last 15 minutes. There should also be a new version tag with the same value for 'Digest'

Warning

This last step is what updates the image tags so that the version that was just tested is marked as ready for Environment

Deployment projects to consume.

Accomplished Outcomes

1. Use the Pipeline Editor and its Lint feature.
2. Update the Application Build project to add Auto DevOps (Runner Push Deployment) with the GitLab Agent cluster connection method.
3. Track the progress of the build through security scanning, staging environment and ‘production’ environment. (Production in this case means that the Application container is production ready.)
4. Through both environments of the Environment Deployment project using the background page color.

Workshop Version: v1.3.2

LAB 5.2: MERGE REQUEST AND REVIEW APP

> **Keyboard Time**: 10 mins, **Automation Wait Time**: 5 mins
>
> **Scenarios:** Instructor-Led, Self-Paced

Merge Request: Developer's Single Pane of Glass

GitLab's MR view of vunerability findings and code quality changes are unique among security tooling because:

- They are only for the code the developer has just changed on their branch.
- They are presented during coding iterations before merging with shared branches - when developers are much more willing to change implementations and upgrade dependencies to eliminate vulnerabilities.
- They are directly associated with the Developer who created them because the authorship of the branch and commits is automatically tracked.
- They allow developers to collaborate around findings with the full power of GitLab collaborative issues.

Target Outcomes

1. Perform security scanning on the application.
2. Explore the Merge Request view of vulnerability findings.

3. Open 'yourpersonalgroup/hello-world'

4. In the left navigation *Click* **Repository => Files**
5. Near the top right of the page **Click** ***Web IDE*** (button)
6. In the left file navigation *Navigate to* the file **src/microwebserver.py**
7. Around line 11, *locate* the text `<BODY style="background:`
8. Change the color after the word `background:` to `beige`

   Result: `<BODY style="background:beige">`

   > Other available color values are listed here

9. In the left file navigation *Click* **Dockerfile**
10. Change the first line starting with FROM to `FROM python:3.6-slim-bullseye`
11. In the left file navigation *Click* **.gitlab-ci.yml**
12. *Delete* or *Comment* the line **STAGING_ENABLED: true**

    > This will speed our labs up - keep in mind the production environment is only published the container - so a review environment and production environment for integration may be enough environments for your development process.

13. *Click* **Create commit...**
14. *Select* **Create a new branch** (Should be the default)

Info

The action of creating a new branch is what creates a per-feature branch review app. Commiting to the main branch would avoid the creation of a feature branch review environment - but this also prevents the developer's Single Pane of Glass (SPOG) assessment of their changes in the Merge Request view and prevents multiple developers working on changes to the same codebase.

11. Leave the default branch name.
12. *Check* **Start a new merge request** (should be defaulted to checked)
13. *Click* **Commit**
14. On the 'New merge request' page, for Title, *Type* **Updating color**
15. *Click* **Create merge request**

    > You will be put within the context of the new merge request

16. On the Merge Request page locate the tab bar containing Overview, Commits, Pipelines and Changes.
17. *Click* **Pipelines**

    > Merge Request pipelines are specifically associated with a Merge Request as well as the underlying branch.

18. To open the pipeline page, *Click* **[the Status badge]** or **[the pipeline #]**

    > The Review stage is where GitLab is creating a temporary copy of the running application so that Dynamic Application Security Testing (DAST) can be performed or any other type of QA requiring a running copy of the application.

19. **[Automation wait: ~5 min]** Wait for the Review stage to complete.

Observe

The jobs are the same as when we enabled Auto DevOps and deployed to staging - but now we are deploying to a completely isolated branch.

20. On the left navigation, *Click* **Deployments => Environments**
21. Next to 'review/<your_mr_branch_name>' *Click* **Open**.
22. This review environment should have the new background color.
23. In the browser tabs, *Click* **Environments**
24. Next to 'production' *Click* **Open**
25. The production environment should have the original background color.
26. In the browser tabs, *Click* **Environments**
27. In the left navigation, *Click* **Merge Requests**
28. *Click* **[your merge request ("Updating color")]**
29. In the main page body, next to 'Security scanning detected...', *Click* **Expand**
30. In the left navigation, *Click* **[one of the findings links]**

Observe

The vulnerability record you are viewing allows action to be initiated immediately with controls such as Dismiss vulnerability, Resolve with merge request, Download patch to resolve and Create issue.

31. *Click* **[the small X in the upper right corner]**
32. Take a mental note of the total number of Critical findings.
33. In the left navigation, *Click* **Security & Compliance => Vulnerability report**
34. Hover over the description for one of the vulnerabilities and

notice the branch reference. They are all from the main branch.

Observe: The Right Information at the Right Time with Known Problem ownership

The Merge Request gives excellent motivational context for the formation of an active vulnerability remediation habit because:

1. All the vulnerabilities in the dashboard are from the default branch “main”, so the ones in our Merge Request are unique to the code we just changed (the old container we introduced and) we have an opportunity to resolve them.
2. An attempt to merge without resolving the findings will cause the whole development team and the security department to pay attention to the new vulnerabilities.
3. Since we are still making decisions about what container would be best to use, it’s a great time to choose a newer one and take a little time to fix up any of our code that might be affected by the new version.
4. Our other unit and code tests should flush out if updating the container version causes functional code problems in the next push to GitLab.

Accomplished Outcomes

1. Perform security scanning on the application.
2. Explore the Merge Request view of vulnerability findings.

Workshop Version: v1.3.2

LAB 5.3: MERGE MR TO PUBLISH CONTAINER AND DEPLOY TO ENVIRONMENT

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

Scenarios: Instructor-Led, Self-Paced

Target Outcomes

1. Merge the Application Build project Merge Request to build and publish a new version of the Application Container.
2. Disable staging environment (for speed).
3. Optionally run the Environment Deployment project to push the application to production.

GitOps Practices

In this scenario we have a branch review environment and production environment for the Application Build Project. If the Application Build Project is used only by the same environment types which are also directly managed by the same DevOps team as the Application Build Project, then it may make sense to ensure there is enough testing in the Application Build Project that Environment Deployment Projects can simply ship the application to production (disable staging).

However, if the artifacts (container in this case) of the Application Build Project are used by many different types of environments and/or by Ops teams that have seperation of security or duties from the Application Build Project team, then

> there could be a lot of testing in downstream Environment Deployment Projects as well.

1. Open ‘yourpersonalgroup/hello-world’
2. On the left navigation, *Click* **Merge Requests**
3. *Click* **[your merge request (“Updating color”)]**

   > The information in the main body of the merge request is what is contstantly updated. Approvals also have to be given when they are configured.

4. We are going to pretend you’ve gotten your code running, resolved all CI checks, addressed all MR feedback and gotten the sufficient approvals to merge.

   > If you are NOT in an instructor-led class, you could revert the container version changes and after the CI runs, see that you no longer have vulnerabilities in your MR view.

5. In the main page body, Click* **Merge** (button)
6. On the left navigation, *Click* **CI/CD => Pipelines**

   > One pipeline starts with your merge request name - when a branch is merged, the previous pipeline’s “stop_review” job is run to destroy the review environment and prevent test environment sprawl.
   >
   > The other pipeline starts with ‘Merge branch’ and is for the merge into the default branch and it will push the changes through both a staging and production environment.

7. **[Automation wait: ~1 min]** Wait for the shorter pipeline to complete.
8. On the left navigation, *Click* **Deployments => Environments**

9. Notice the ‘review/<your_mr_branch_name>’ review app is gone - it was removed by the MR pipeline running “stop_review” when you merged to main.
10. To return to the pipeline list, *Click* **[the browser back button]**
11. To open the “Merge branch…” pipeline page, *Click* **[the Status badge]** or **[the pipeline #]**
12. **[Automation wait: ~2 min]** Wait until the pipeline starting with “Merge branch…” has a status of ‘blocked’ (meaning pressing play is required to continue) or in the pipeline details page you can see that the ‘production’ job is complete
13. On the left navigation, *Click* **Deployments => Environments**
14. Next to ‘production’ *Click* **Open**
15. The production environment should have the new background color.
16. In the browser tabs, *Click* **Environments**
17. On the left navigation, *Click* **CI/CD => Pipelines**
18. To open the blocked pipeline starting with ‘Merge branch…’ *Click* **[the Status badge]** or **[the pipeline #]**
19. **[Automation wait: ~5 min]** Wait for [the play icon] to appear on the ‘promote-image-to-latest’ job. Hitting the browser refresh button helps ensure your status display is accurate.

    > You may have to wait for the ‘browser_performance’ CI test job to complete before [the Play button] appears on the ‘promote-image-to-latest’ job.

20. **IMPORTANT:** Once it is available on the job ‘promote-image-to-latest’ *Click* **[the Play button]**

Observe: Abandoning a Release Candidate

The way to abandon a release candidate is to never push play on 'promote-image-to-latest'. The CI pipeline should also be completely deleted so that no one can accidently push the button.

Info

The environments in the Application Build Project are only for testing and qualifying the build - even the one called 'Production'. The 'promote-image-to-latest' is what actually signals downstream Environment Deployment Projects that a new version has been published. The DevOps methodology and processes you use should determine how much testing and qualification is done in this Application Build Project and how much is done in Environment Deployment Projects. There could be a lot of testing in both if downstream environments want to double check or have additional checks before deploying to actual production endpoints.

Part B: GitOps CD Pull Changes to Production (can be skipped)

> **Keyboard Time**: 10 mins, **Automation Wait Time**: 5 mins
> **Scenarios:** Instructor-Led, Self-Paced

Info

While these lab steps completely close the loop with an end-to-end change with review environments and security scanning, since you have seen this flow already, it is optional to push this particular change all the way to production..

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

   > You will now take the change in the World Greeting Environment Deployment Project.

2. *Click* **CI/CD => Schedules**
3. On the right of CheckForNewContainerVersion *Click* **[the play button]**

   > If there are no scheduled pipelines, just navigate to **CI/C => Pipelines** and *Click* **Run pipeline** and *Click* **Run pipeline**

4. On the left navigation, *Click* **CI/CD => Pipelines**
5. On the latest pipeline *Click* **[the Status badge]** or **[the pipeline #]**
6. Wait until the Child deploy pipeline has a status of ‘blocked’ indicated by the gear icon (meaning manual action is needed)
7. Expand the Downstream pipeline with the great than arrow (`>`).
8. **[Automation wait: ~3 min]** Wait until the update-staging-manifests job has a green check next to it
9. On the left navigation, *Click* **Deployments => Environments**
10. Next to ‘staging’ *Click* **Open**
11. The staging environment should have the new background color.
12. In the browser tabs, *Click* **Environments**
13. Next to ‘production’ *Click* **Open**
14. The production environment should still have the original background color.
15. In the browser tabs, *Click* **Environments**
16. On the left navigation, *Click* **CI/CD => Pipelines**
17. To open the blocked pipeline, *Click* **[the Status badge]** or **[the pipeline #]**

18. Expand the Downstream pipeline with the great than arrow (`>`).
19. Next to the 'update-production-manifests' job, *Click* **[the play button]**
20. **[Automation wait: ~5 min]** Wait for the GitLab deployment to production to complete successfully and wait ~ 3 more minutes for the GitLab Agent to perform a GitOps CD Pull.
21. On the left navigation, *Click* **Deployments => Environments**
22. Next to 'production' *Click* **Open**
23. The production environment should have the new background color.

    > This is your change in the actual production environment. Keep in mind there could be many Environment Deployment Projects that consume the Application Build Project 'hello-world' container image. They could be using schedules, manual triggering by running pipelines or merge requests. For the supported methods in these working example projects you can read more here.

Accomplished Outcomes

1. Merge the Application Build project Merge Request to build and publish a new version of the Application Container.
2. Disable staging environment (for speed).
3. Optionally run the Environment Deployment project to push the application to production.

Workshop Version: v1.3.2

LAB 5.4: ADD KUBERNETES MANIFEST SECURITY SCANNING

> **Keyboard Time**: 15 mins, **Automation Wait Time**: 8 mins
>
> **Scenarios:** Instructor-Led, Self-Paced

Kubernetes Manifest SAST Security Scanning

GitLab can do kubernetes manifest scanning using Kubesec.

Target Outcomes

1. configures the security scanning of fully rendered Kubernetes manifests for security problems.

CI Runner Required

While this project only uses the GitLab Agent Pull CD mode for deployment, it does need a runner if we want to do additional CI tasks in this project. In this case we want to add manifest scanning, which can only be done in the Environment Deployment project where the full manifests are constructed.

1. Open 'yourpersonalgroup/world-greetings-env-1'
2. On the upper right of the Project page, *Click* **Web IDE**
3. In the left side file browser, *Click* **update-manifests.gitlab-ci.yml**

   > You will be editing YAML - be careful that tabbing is properly aligned. Only removing the comment character ("#") should result in proper tabbing.

4. Under `include:` uncomment `- template: Security/SAST-IaC.latest.gitlab-ci.yml` which should make the section look like this.:

```
include:
  - local: .gitlab/ci_templates/git-push.yaml
  - template: Security/SAST-IaC.latest.gitlab-ci.yml
```

5. Under `variables:` uncomment the `variables:` heading and the two variable which should make the section look like this:

```
variables:
  SCAN_KUBERNETES_MANIFESTS: "true"
  KUBESEC_HELM_CHARTS_PATH: $CI_PROJECT_DIR/constructed-manifests/
```

6. In the left side file browser, Click packages/hello-world/base/deployment.yaml

7. At the bottom of the file edit the `securityContext:` section to look like this (be sure to keep the same indentation starting by not moving the existing keyword `securityContext` and indenting sub levels by two spaces):

```
      securityContext:
        capabilities:
          add:
            - SYS_ADMIN
```

8. *Click* **Create commit...**

9. *Select* **Commit to main branch** (this is not the default)

10. Under 'Commit Message', *Type* **[skip ci] Adding Manifest Security Scanning**

11. *Click* **Commit**
12. Below the Create commit… button, in the status bar, *Click* **[the pipeline #]**
13. Expand the Downstream pipeline with the great than arrow (`>`).
14. Under the new stage ‘Test‘, *Locate* the new job **kics-iac-sast**
15. *Click* **kics-iac-sast**
16. Near the bottom of the log, *Locate* **gl-sast-report.json : found 1 files and directories**
17. On the left navigation, *Click* **Security & Compliance => Vulnerability report**
18. If there are not any vulnerabilities listed, you can examine the page for `Last updated` followed by an elapsed time and a clickable pipeline id reference.

Accomplished Outcomes

1. configures the security scanning of fully rendered Kubernetes manifests for security problems.

Workshop Version: v1.3.2

LAB 5.5: ADD OPERATIONAL CONTAINER SECURITY SCANNING

> **Keyboard Time**: 5 mins, **Automation Wait Time**: 10 mins
>
> **Scenarios:** Instructor-Led, Self-Paced

Scanning Other Images In The Cluster

Operational Container Scanning is able to add security findings for images used in your cluster that are not a part of your development process and therefore do not get routinely scanned during CI of the application. This scanning currently focuses on the cluster’s images and not other aspects of cluster security. The other security scanning we’ve been configuring has explictly to do with development - this scanning is part of GitLab’s Protect stage - part of operational integrity that can be enabled via the GitLab Agent.

Target Outcomes

1. Configure Operational Container Scanning using the GitLab Agent for Kubernetes.
2. Examine Operational Container Scanning findings.

Troubleshooting: Operational Container Scanning

Troubleshooting Guide: Operational Container Scanning

Done By Instructor for Instructor-Led Courses

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

Temporary Fix

Until 361792 is resolved, you will need to run this command in the cluster:

```
kubectl create serviceaccount gitlab-agent -n gitlab-agent
```

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. Open ‘classgroup/cluster-management’
4. In the left navigation, *Click* **Repository => Files**
5. On the upper right of the Project page, *Click* **Web IDE**
6. Navigate to the file .gitlab/agents/spotazuseast2-agent/config.yml
7. Look at the minutes past the hour of the current time.
8. Add 5 minutes and insert the following snippet - substitute your minutes number for ‘55’ in the below:

   ```
   starboard:
     cadence: '55 * * * *' #Every hour at 55 minutes past the hour
   ```

   Do not set to a low frequency like every minute

   If the cluster scanning job launches multiple simultaneous instances, it is more likely to get in a bad state

> 9. *Click* **Create commit...**
>
> 10. *Select* **Commit to master branch**
>
> 11. Under ‘Commit Message’, *Type* **[skip ci] Adding Manfest Security Scanning**
>
> 12. *Click* **Commit**
>
> > The time can be updated to retrigger the agent if there are problems getting it to run.

12. 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

13. **[Automation Wait Time: ~5 mins]** Wait for the cluster to receive the new directive and perform a scan.

14. To see scanning results, while in ‘classgroup/cluster-management’

15. *Click* **Infrastructure => Kubernetes clusters => spot2azuseast2-agent1 => Security** (Tab)

16. Under ‘Status’ *Click* **[to expand the drop down]** and then *Click* **All statuses**

17. These findings are also visible in the standard security dashboards.

18. Open ‘classgroup’

19. On the left navigation, *Click* **Security & Compliance => Vulnerability Report**

20. In the tab bar under ‘Vulnerability report’, *Click* **Operational vulnerabilities**

21. Under 'Status' *Click* **[to expand the drop down]** and then *Click* **All statuses**

> Notice the list of vulnerabilities.

Accomplished Outcomes

1. Configure Operational Container Scanning using the GitLab Agent for Kubernetes.
2. Examine Operational Container Scanning findings.

Warning

To speed up the class results we set the cluster scanner to every minute. If this is a long lived cluster it would be prudent to update the starboard:cadence above to once a day or less.

Workshop Version: v1.3.2