VMware Advanced Customer Engagements (ACE) Team

Omnibus GitLab Integration with VMware TKGI and Harbor

How-to guide for Integration and Configuration of CI/CD with examples

MAY 2020

Table of Contents

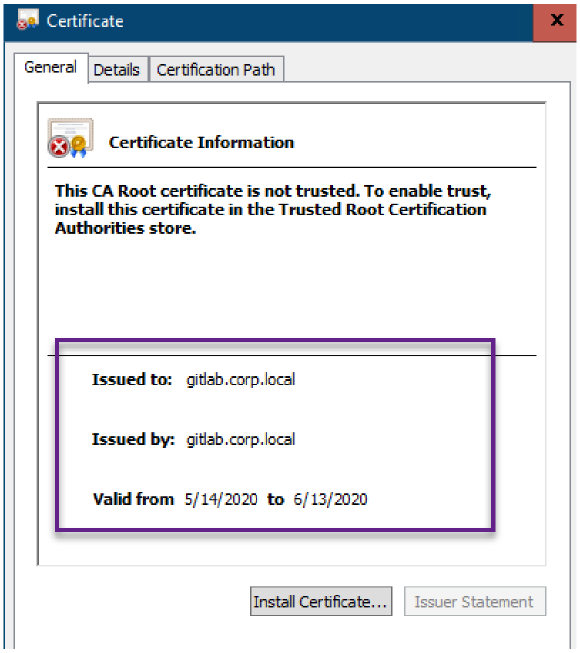
1. *Introduction*

In this document, we provide an overview of integration of OmniBus GitLab (Enterprise Edition) software change management (SCM) platform with VMware Tanzu Kubernetes Grid Integrated (TKGI, formerly known as VMware Enterprise PKS) Kubernetes clusters and Harbor container image registry platform for automated software build tasks. GitLab is a popular DevOps platform since it is compatible with Git file versioning, project directory structure and client software.

We highlight configuration steps to enable integration between GitLab EE and Kubernetes clusters provisioned with TKGI platform and Harbor container image registry to enable CI/CD process automation using GitLab tools.

2. *Pre-requisites:*

- The following software should be installed and accessible from
  - VMware TKGI (formerly 'Enterprise PKS', v 1.6.1 or 1.7)
  - Kubernetes cluster provisioned via TKGI environment that is accessible via kubectl CLI
  - OmniBus GitLab Enterprise Edition (v 11.2 or later, v 12.10.5-ee used in a Lab for this paper) installed and configured, accessible via URL like https://gitlab.corp.local via administrator level account
    - A GitLab project that contains software artifacts that can be built into container images. We are using the following example from GitHub: https://github.com/riazvm/dockersample cloned into a local GitLab project
  - "CLI VM" – typically a Linux VM that is used for Command Line access to Kubernetes clusters and runs other tools (Docker, Helm etc.) for configuration of integrations and intermediate validation of build process stages.
- There should be no networking issues (firewalls, blocked ports, DNS resolution etc.) between GitLab VM, TKGI K8s clusters and Harbor VM
- Main GitLab URL (such as **gitlab.corp.local**) should have a valid CA certificate (typically generated by GitLab installer script such as shown below:



Certificate Information

This CA Root certificate is not trusted. To enable trust, install this certificate in the Trusted Root Certification Authorities store.

Issued to: gitlab.corp.local

Issued by: gitlab.corp.local

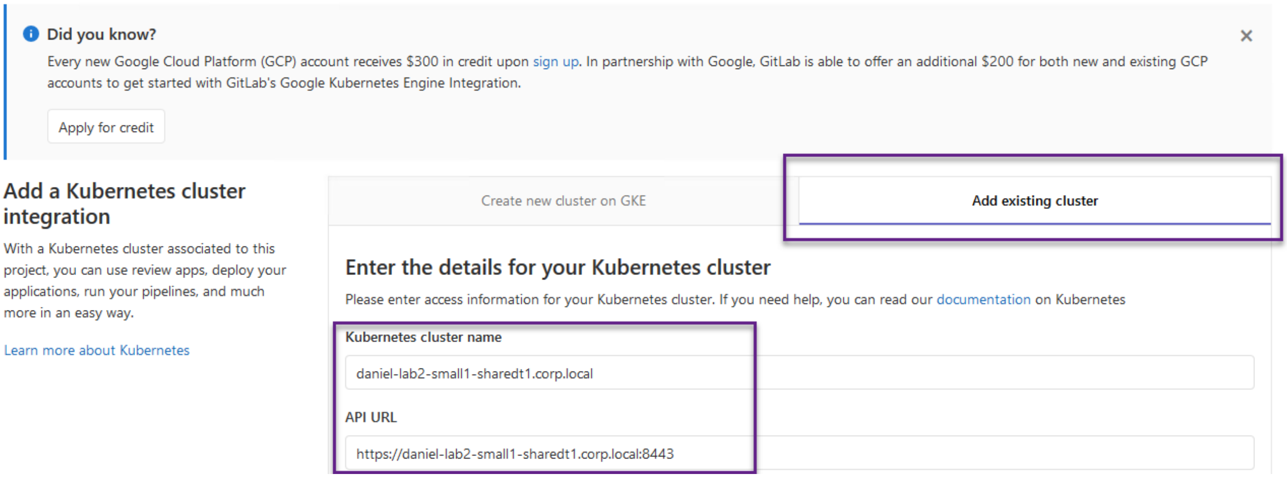
Valid from 5/14/2020 to 6/13/2020

3. *Add Existing TKGI K8s Cluster to GitLab project*

We need to add K8s cluster to our GitLab project as a target for CI/CD deployments of containerized applications and for deploying Runner components that execute pipeline tasks/scripts. See GitLab documentation for more information on Runners

- Start with “Add Existing Cluster” Tab in the “Operations – Kubernetes” menu for a project:

Specify name and FDQN (create DNS record if doesn’t exist yet) based URL of API Server/Master node(s)



Did you know?

Every new Google Cloud Platform (GCP) account receives $300 in credit upon sign up. In partnership with Google, GitLab is able to offer an additional $200 for both new and existing GCP accounts to get started with GitLab's Google Kubernetes Engine Integration.

Add a Kubernetes cluster integration

With a Kubernetes cluster associated to this project, you can use review apps, deploy your applications, run your pipelines, and much more in an easy way.

Enter the details for your Kubernetes cluster

Please enter access information for your Kubernetes cluster. If you need help, you can read our documentation on Kubernetes

Kubernetes cluster name: daniel-lab2-small1-sharedt1.corp.local

API URL: https://daniel-lab2-small1-sharedt1.corp.local:8443

- Follow “More Information” links for each field to be filled in, as specified in documentation https://gitlab.acelab.local/help/user/project/clusters/add_remove_clusters.md#add-existing-cluster with the following fields:
  Obtain CA certificate from the K8s cluster using command like

  ```
  kubectl get secret <secret name> -o jsonpath =
  "{['data']['ca\.crt']}" | base64 –decode

  E.G. kubectl get secret <default-token-p6br2> -o jsonpath =
  "{['data']['ca\.crt']}" | base64 -d
  ```

  Note: If the command returns the entire certificate chain, copy the *root ca* certificate value at the bottom of the chain:

  -----BEGIN CERTIFICATE-----
  MIIC+zCCAeOgAwIBAgIUBEYdVQpHO7z4r608A+8wNRLmhbkwDQYJKoZIhvcNAQEL
  BQAwDTELMAkGA1UEAxMCY2EwHhcNMjAwNDIyMDQ1NDM1WhcNMjQwNDIyMDQ1NDM1
  WjANMQswCQYDVQQDEwJjYTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
  …….
  …….
  C8h+Hip2IxlN/Kubq7Hv3yNFD9MbtnpCRmP9nFCo/UFapjljvtd6O0F1qPOQzU8=
  -----END CERTIFICATE-----

  Copy the above certificate string value for use in the following steps
- Obtain Authentication Token for GitLab authentication against K8s

GitLab authenticates against K8s using service tokens, which are scoped to a namespace. The token used should belong to a service account with ‘cluster-admin’ privileges.
Follow GitLab documentation to create a Service Account and Cluster Role Binding with “cluster-admin” privileges using sample *gitlab-admin-service-account.yaml* K8s deployment descriptor provided as an example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: gitlab-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: gitlab-admin
  namespace: kube-system
```

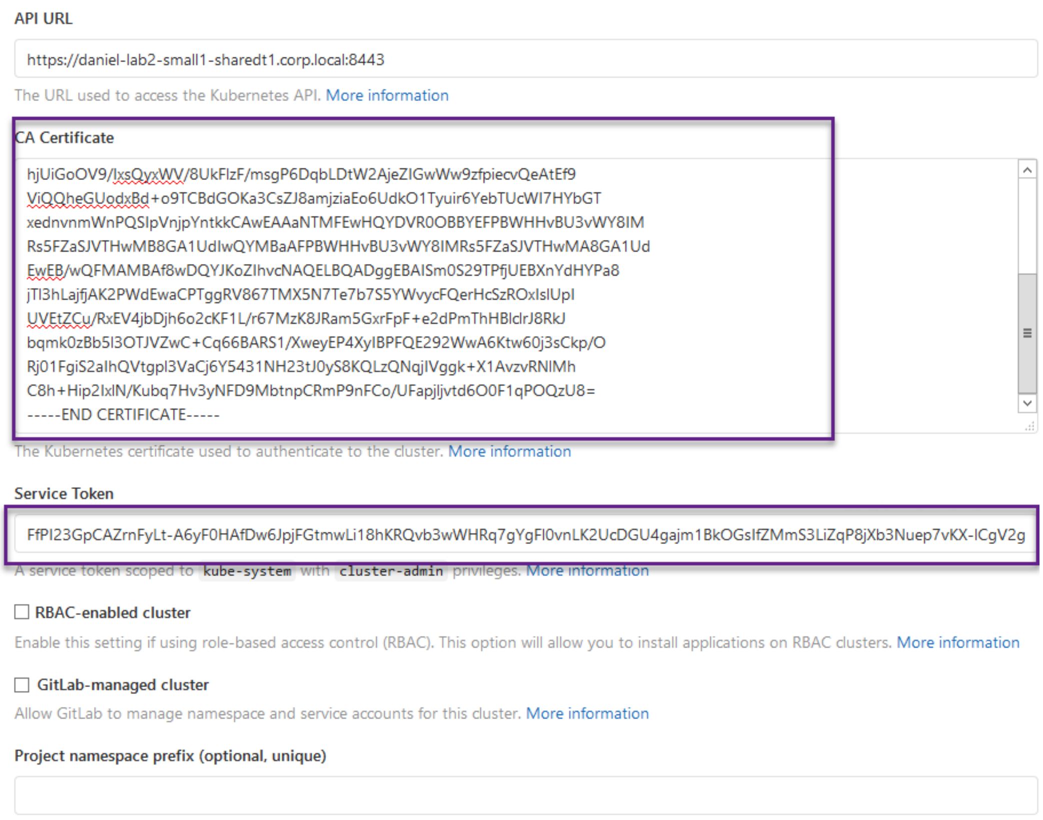
Create Service account and Cluster Role Binding in the target cluster:

```
  kubectl apply -f gitlab-admin-service-account.yaml
serviceaccount/gitlab-admin created
clusterrolebinding.rbac.authorization.k8s.io/gitlab-admin created
```

- **Retrieve token for the *gitlab-admin* Service Account:**

```
    kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep gitlab-admin | awk '{print $1}')
Name:       gitlab-admin-token-rf6fr
Namespace:   kube-system
Labels:      <none>
Annotations:  kubernetes.io/service-account.name: gitlab-admin
              kubernetes.io/service-account.uid: dd65123d-1a2c-47e7-8a5e-97adf871c27d

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1094 bytes
namespace:  11 bytes
token:  < authentication-token>
```

Paste value of ***authentication-token*** into the 'Service-Token' filed in the "Add Existing Cluster" GitLab screen:



API URL

https://daniel-lab2-small1-sharedt1.corp.local:8443

The URL used to access the Kubernetes API. More information

CA Certificate

```
hjUiGoOV9/IxsQyxWV/8UkFlzF/msgP6DqbLDtW2AjeZIGwWw9zfpiecvQeAtEf9
ViQQheGUodxBd+o9TCBdGOKa3CsZJ8amjziaEo6UdkO1Tyuir6YebTUcWI7HYbGT
xednvnmWnPQSIpVnjpYntkkCAwEAAaNTMFEwHQYDVR0OBBYEFPBWHHvBU3vWY8IM
Rs5FZaSJVTHwMB8GA1UdIwQYMBaAFPBWHHvBU3vWY8IMRs5FZaSJVTHwMA8GA1Ud
EwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAISm0S29TPfjUEBXnYdHYPa8
jTl3hLajfjAK2PWdEwaCPTggRV867TMX5N7Te7b7S5YWvycFQerHcSzROxIslUpI
UVEtZCu/RxEV4jbDjh6o2cKF1L/r67MzK8JRam5GxrFpF+e2dPmThHBlclrJ8RkJ
bqmk0zBb5l3OTJVZwC+Cq66BARS1/XweyEP4XyIBPFQE292WwA6Ktw60j3sCkp/O
Rj01FgiS2alhQVtgpl3VaCj6Y5431NH23tJ0yS8KQLzQNqjIVggk+X1AvzvRNIMh
C8h+Hip2IxlN/Kubq7Hv3yNFD9MbtnpCRmP9nFCo/UFapjljvtd6O0F1qPOQzU8=
-----END CERTIFICATE-----
```

The Kubernetes certificate used to authenticate to the cluster. More information

Service Token

FfPI23GpCAZrnFyLt-A6yF0HAfDw6JpjFGtmwLi18hKRQvb3wWHRq7gYgFl0vnLK2UcDGU4gajm1BkOGsIfZMmS3LiZqP8jXb3Nuep7vKX-ICgV2g

A service token scoped to `kube-system` with `cluster-admin` privileges. More information

☐ RBAC-enabled cluster

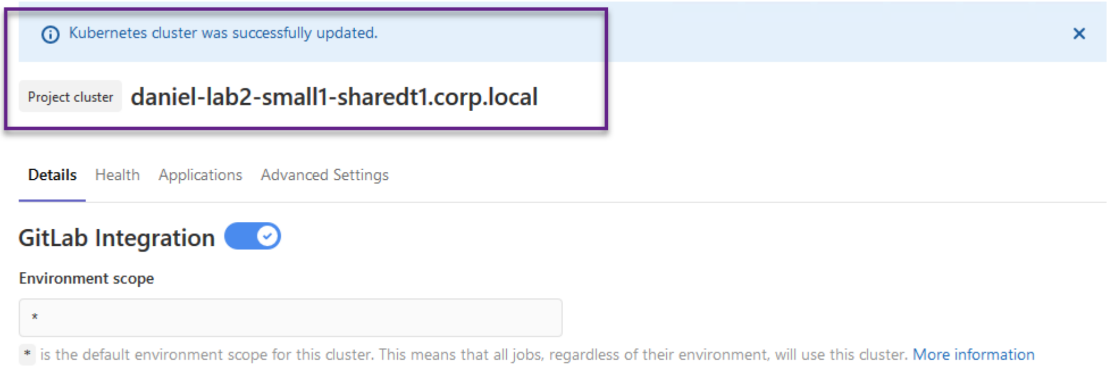
Enable this setting if using role-based access control (RBAC). This option will allow you to install applications on RBAC clusters. More information

☐ GitLab-managed cluster

Allow GitLab to manage namespace and service accounts for this cluster. More information

Project namespace prefix (optional, unique)

- **Click "Add Cluster" button – should get a confirmation of successful result on GitLab UI**:



ⓘ Kubernetes cluster was successfully updated.

Project cluster daniel-lab2-small1-sharedt1.corp.local

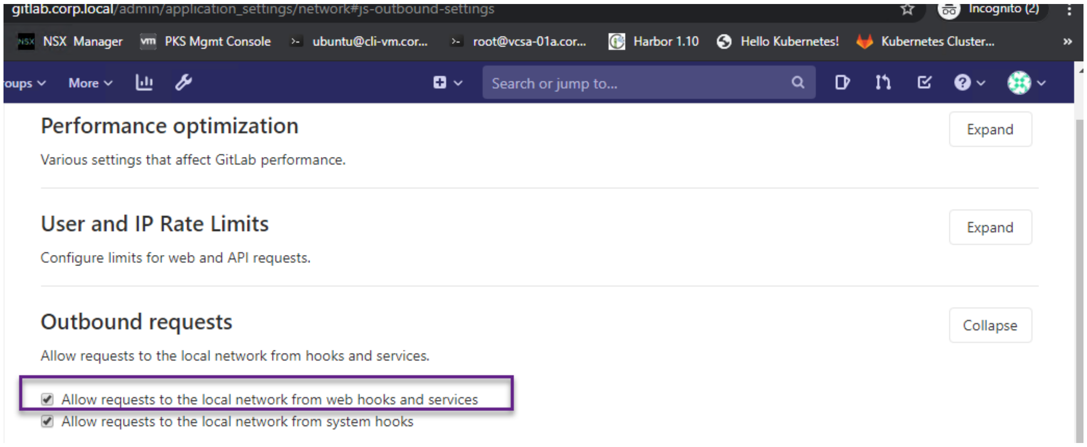
GitLab Integration ✓

Environment scope

*

* is the default environment scope for this cluster. This means that all jobs, regardless of their environment, will use this cluster. More information

NOTE: in case if a warning about blocked requests to local networks is displayed (when GitLab VM and K8s cluster API are on the same network), we may need to explicitly allow requests to local networks from Web Hooks and Services, following KB Article: https://gitlab.com/gitlab-org/gitlab-foss/-/issues/57948



Performance optimization

Various settings that affect GitLab performance.

User and IP Rate Limits

Configure limits for web and API requests.

Outbound requests

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

4. *Install and configure GitLab Runner using Helm chart, associate it with project*

In GitLab CI/CD, Runners run the code defined in the **.gitlab-ci.yml** pipeline definition file. They can be dedicated virtual machines or dedicated Kubernetes Pods that pick up build jobs through the coordinator API of GitLab CI/CD. A Runner can be specific to a certain project or serve any project in GitLab CI/CD, the latter is called a Shared Runner.

Below are the steps to prepare for installation of GitLab Runners as in-cluster K8s resource via Helm chart, performed on CLI VM.(generally, follows the documentation https://docs.gitlab.com/runner/install/kubernetes.html)

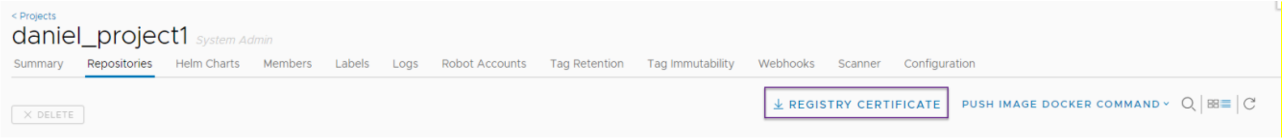
- Install Helm client/server following documentation , validate that it can reach general repositories containing **gitlab-runner** charts

E.G. `helm search hub gitlab-runner`

```
URL                                              CHART VERSION   APP VERSION     DESCRIPTION
https://hub.helm.sh/charts/choerodon/gitlab-runner      0.2.4          0.2.4          gitlab-runner for
Choerodon
https://hub.helm.sh/charts/pnnl-miscscripts/git...     0.1.3          0.1.2-1        A Helm chart for
Kubernetes
https://hub.helm.sh/charts/camptocamp/gitlab-ru...     0.12.6         12.6.0         GitLab Runner
https://hub.helm.sh/charts/gitlab/gitlab-runner       0.16.0         12.10.1        GitLab Runner
```

- Download Harbor Registry certificate from its UI

  Login to Harbor UI, navigate to the Project where plan to host built container images click on “Registry certificate” to download the certificate file:



- Create namespace in the K8s cluster where GitLab Runner Pod will be running

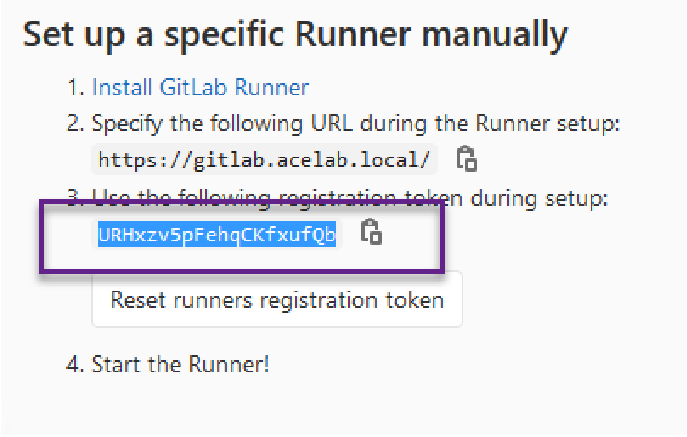
```
kubectl create ns gitlabrunner
```

NOTE: Here the Runner namespace is called **gitlabrunner** but it can be other valid namespace name

Set that namespace as current context:

```
kubectl config set-context --current --namespace=gitlabrunner
```

- Navigate in GitLab UI to “Settings ➔ CI/CD ➔ Runners”:



Set up a specific Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup:
   `https://gitlab.acelab.local/`
3. Use the following registration token during setup:
   `URHxzv5pFehqCKfxufQb`

4. Start the Runner!

and copy values for GitLab URL and Runner Registration token from the screen above.

- Use GitLab URL and registration token values obtained in the previous step in the **values.yaml** Helm chart configuration file (full example available for Runner Helm chart installation in the GitHub repository: https://gitlab.com/gitlab-org/charts/gitlab-runner/-/blob/master/values.yaml)

```
## GitLab Runner Image
##
## By default it's using gitlab/gitlab-runner:alpine-v{VERSION}
## where {VERSION} is taken from Chart.yaml from appVersion field
##
## ref: https://hub.docker.com/r/gitlab/gitlab-runner/tags/
```

```
##
##image: gitlab/gitlab-runner:alpine-v11.6.0
gitlabUrl: https://gitlab.acelab.local
runnerRegistrationToken: 'URHxzv5pFehqCKfxufQb'
```

NOTE: please see GitLab documentation https://docs.gitlab.com/runner/install/kubernetes.html for recommended values of additional fields in values.yaml file for GitLab Runner Helm chart.

- Another important field is RBAC support for a Runner service account. To have the chart create new Service account during installation, set **rbac.create** to **true**

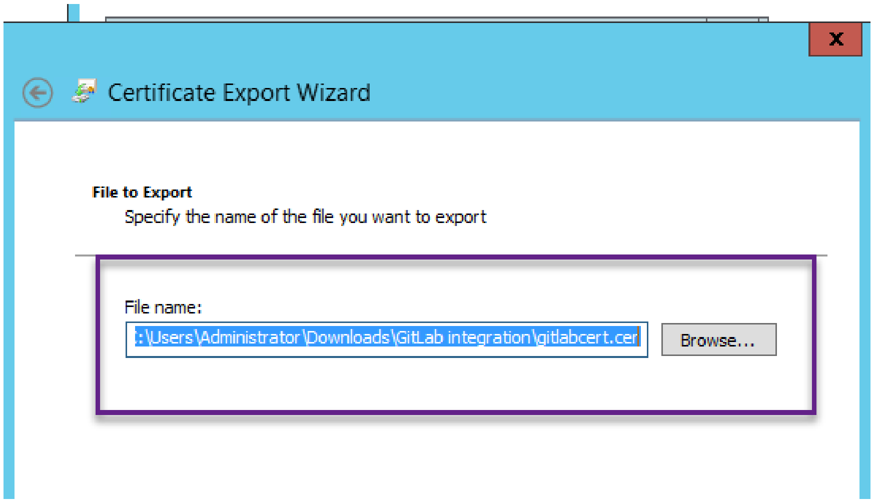
```
## For RBAC support:
rbac:
  create: true
  ## Define specific rbac permissions.
  # resources: ["pods", "pods/exec", "secrets"]
  # verbs: ["get", "list", "watch", "create", "patch", "delete"]
..
```

(Otherwise, set to **rbac create** to **false** and specify existing Service Account)

- An important setting is Max number of concurrent jobs to run which is controlled by **concurrent** filed value. Set it based on projected size of build jobs and related resource utilization:

```
..
## Configure the maximum number of concurrent jobs
## ref: https://docs.gitlab.com/runner/configuration/advanced-configuration.html#the-global-section
##
concurrent: 10
..
```

- To allow containers activated on Runner to make API calls against GitLab Secure API, we need to export SSL certificate from GitLab server in a CER (BASE 64) format into a file:



File to Export
Specify the name of the file you want to export

File name: :\Users\Administrator\Downloads\GitLab integration\gitlabcert.cer

- ✓ The certificate file name used should be in the format **<gitlab.hostname.domain.crt>**, for example **gitlab.corp.local.crt**.
- ✓ Any intermediate certificates need to be concatenated to your server certificate in the same file.
- ✓ The hostname used should be the one the certificate is registered for.

- Generate K8s secret from GitLab CA certificate file (saved as **gitlab.corp.local.crt** in previous step) that complies with above conditions in the K8s namespace where Runner will be deployed using command like:

  ```
  $ kubectl create secret generic gitlabca  --from-
  file=gitlab.corp.local.crt -n gitlabrunner
  secret/gitlabca created
  ```

  Validate the secret got created

  ```
  $ kubectl get secrets

  NAME              TYPE                               DATA   AGE
  default-token-ckt9m   k10ubernetes.io/service-account-token   3      30m
  gitlabca          Opaque                             1      9s
  ```

- Use that K8 Secret name in the **certsSecretName** section of the **values.yaml** file as shown below:

  ```
  ..
  ## Set the certsSecretName in order to pass custom certificates for GitLab Runner to use
  ## Provide resource name for a Kubernetes Secret Object in the same namespace,
  ## this is used to populate the /home/gitlab-runner/.gitlab-runner/certs/ directory
  ```

```
## ref: https://docs.gitlab.com/runner/configuration/tls-self-signed.html#supported-options-for-self-signed-certificates
## secret name
certsSecretName: gitlabca
```

- Configure environment variables that will be present when Runner registration command runs in the following section of values.yaml file:

```
## This provides further control over the registration process and the config.toml file
## ref: `gitlab-runner register --help`
## ref: https://docs.gitlab.com/runner/configuration/advanced-configuration.html
##
envVars:
  - name: RUNNER_ENV
    value: "DOCKER_TLS_CERTDIR="
  - name: CI_SERVER_TLS_CA_FILE
    value: /home/gitlab-runner/.gitlab-runner/certs/gitlab.acelab.local.crt
```

NOTE: certificate file name in the value for **CI_SERVER_TLS_CA_FILE** variable should be same as file name used to generate K8s secret above (**gitlabca)** used in the ***certsSecretName*** section

- If CI/CD task will require using “executor” images running containers in ‘privileged’ mode (such as when using popular DIND “docker in docker”, per GitLab documentation), perform the following optional configuration steps:

a. Update “privileged” parameter flag in the **values.yaml** file:

```
  ## Run all containers with the privileged flag enabled
  ## This will allow the docker:dind image to run if you need to run Docker
  ## commands. Please read the docivi s before turning this on:
  ## ref: https://docs.gitlab.com/runner/executors/kubernetes.html#using-docker-dind
  ##
  privileged: true
.....
```

b. Copy previously downloaded Harbor certificate file to **/etc/gitlab/trusted-certs** and **/etc/gitlab/ssl** folders in GitLab VM:

```
ls /etc/gitlab/ssl
ca_harbor.crt
gitlab.acelab.local.crt
....
ls /etc/gitlab/trusted-certs
ca_harbor.crt
```

gitlab.acelab.local.crt

c. Target TKGI K8s cluster should be deployed with Pod Security Policies set to “privileged” mode, per documentation

NOTE: in our CI/CD example below we will be using an executor container image based on Google Project Kaniko which **does not require running in privileged mode**, please see GitLab documentation for details. Therefore, the configuration steps in this section are not required for running that example.

- Add a Helm repository containing the chart for Runner deployment:

```
helm repo add stable https://kubernetes-charts.storage.googleapis.com
```

Verify that repository has been added and is available:

```
helm repo list
NAME   URL
gitlab  https://charts.gitlab.io
stable  https://kubernetes-charts.storage.googleapis.com
```

- Perform a Runner installation using Helm chart, from the directory where **values.yaml** file is located:

```
helm install gitlab-runner -f ./values.yaml  gitlab/gitlab-runner -n gitlabrunner
```

NOTES:

- ✓ IMPORTANT: if running Helm command from another directory that doesn’t contain **values.yaml** file, provide full path to that file to customize
- ✓ ‘gitlab-runner’ is the name of Runner chart deployment, chosen arbitrary
- ✓ gitlab/gitlab-runner is the name of Helm chart

An output of ‘helm install’ command should look like:

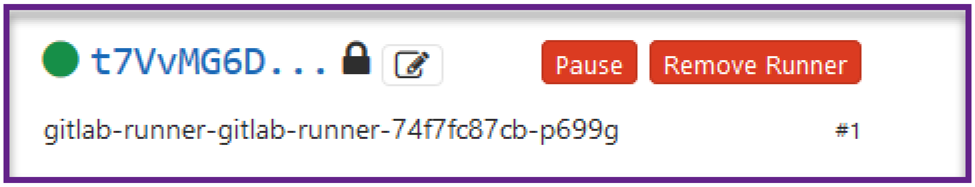
```
NAME: gitlab-runner
LAST DEPLOYED: Sun May  3 05:46:50 2020
NAMESPACE: gitlabrunner
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Your GitLab Runner should now be registered against the GitLab instance
reachable at: https://gitlab.acelab.local
```

- (Optional) validate that Runner deployments/pods are running in the designated K8s namespace:

```
kubectl get deploy, po -n gitlabrunner
NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/gitlab-runner-gitlab-runner   1/1        1              1         33s
NAME                                        READY     STATUS    RESTARTS     AGE
pod/gitlab-runner-gitlab-runner-74f7fc87cb-p699g   1/1    Running        0           33s
```

- Verify that newly installed Runner is configured for GitLab project(s):

Navigate to **Settings ➔ CI/CD** for the Project and check whether Runner shows as active:



gitlab-runner-gitlab-runner-74f7fc87cb-p699g #1

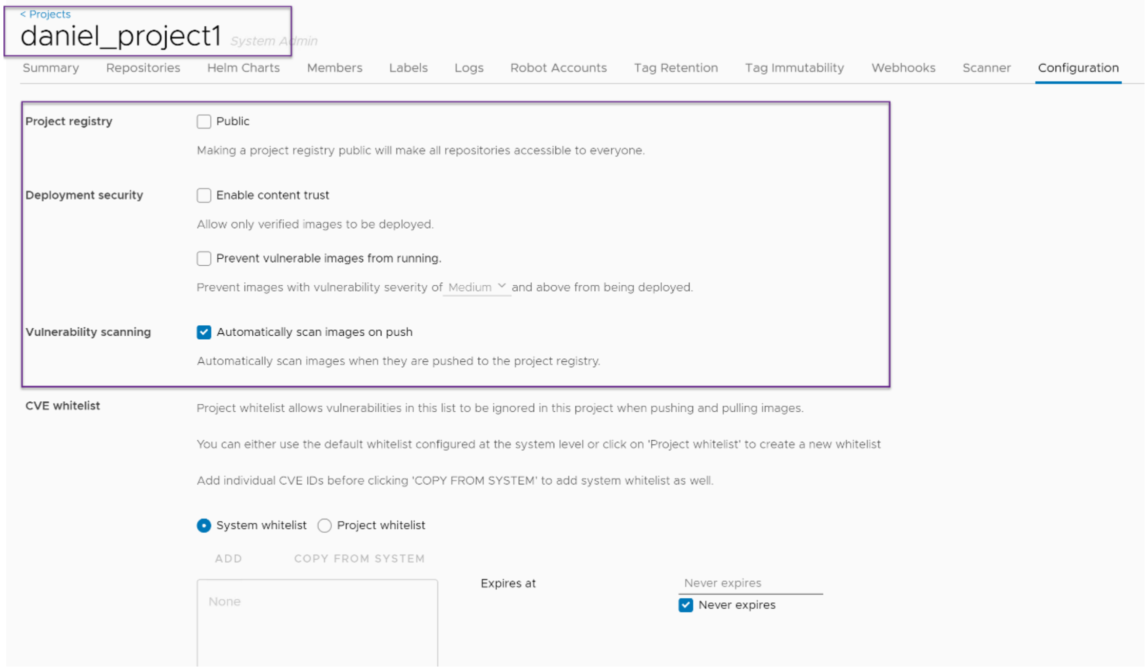
Notes:

- ✓ Usually, Runner configured on a project level will show up in its Settings ➔ CI/CD automatically, when deployed to integrated cluster. In other cases they may be additional steps needed to make Runner available for a GitLab project, per documentation
- ✓ Same instance GitLab Runner can be optionally shared among multiple projects, that can be configured in GitLab "CI/CD Settings ➔ Shared Runners", per documentation

5. *Configure and Execute Project CI/CD Pipeline on Runner*

Below is an example of a CI/CD pipeline that builds a simple SpringBoot microservice from its Java source code using Maven, continues to build a container image using Dockerfile residing in a subdirectory and finally pushes built image into designated project in the Harbor container image repository.

- Properties of target Harbor project (top level construct for images hosting) are shown below. It is not 'Public' and therefore requires authorized user login with at least "Developer" access level, per Harbor documentation. It has image vulnerability scanning on 'push': any time an image is added to a registry via 'docker push…' command, it will be scanned for vulnerabilities automatically.



< Projects

daniel_project1 *System Admin*

Project registry □ Public
Making a project registry public will make all repositories accessible to everyone.

Deployment security □ Enable content trust
Allow only verified images to be deployed.

□ Prevent vulnerable images from running.
Prevent images with vulnerability severity of Medium ⌄ and above from being deployed.

Vulnerability scanning ☑ Automatically scan images on push
Automatically scan images when they are pushed to the project registry.

CVE whitelist Project whitelist allows vulnerabilities in this list to be ignored in this project when pushing and pulling images.

You can either use the default whitelist configured at the system level or click on 'Project whitelist' to create a new whitelist

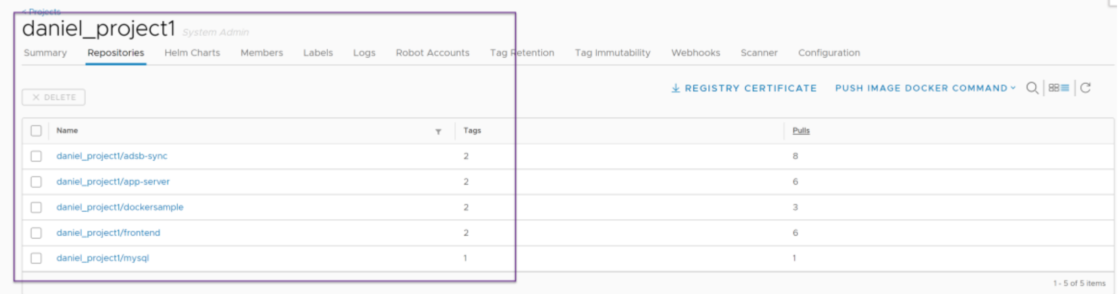
Add individual CVE IDs before clicking 'COPY FROM SYSTEM' to add system whitelist as well.

● System whitelist ○ Project whitelist

None

Expires at: Never expires
☑ Never expires

- There are existing image repositories configured for that project, shown below:



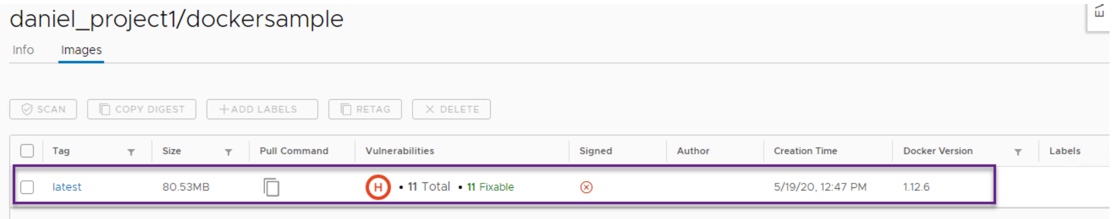
< Projects

daniel_project1 *System Admin*

	Name	Tags	Pulls
□	daniel_project1/adsb-sync	2	8
□	daniel_project1/app-server	2	6
□	daniel_project1/dockersample	2	3
□	daniel_project1/frontend	2	6
□	daniel_project1/mysql	1	1

1 - 5 of 5 items

Our GitLab pipeline will be building and pushing images into ‘**daniel_project1/dockersample’** repository:

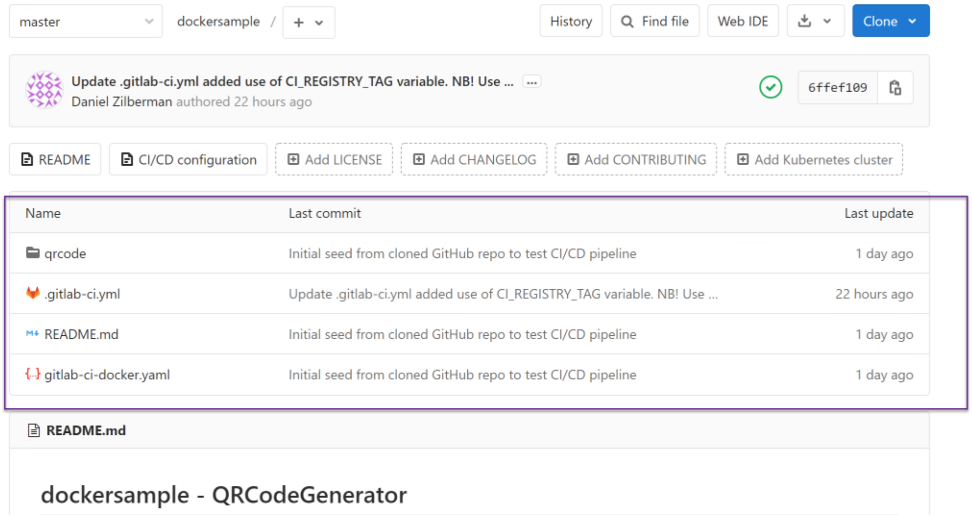


daniel_project1/dockersample

	Tag	Size	Pull Command	Vulnerabilities	Signed	Author	Creation Time	Docker Version	Labels
□	latest	80.53MB		H • 11 Total • 11 Fixable	⊗		5/19/20, 12:47 PM	1.12.6	

There is an existing version of an image

- Structure of the example project **dockersample** is shown below:



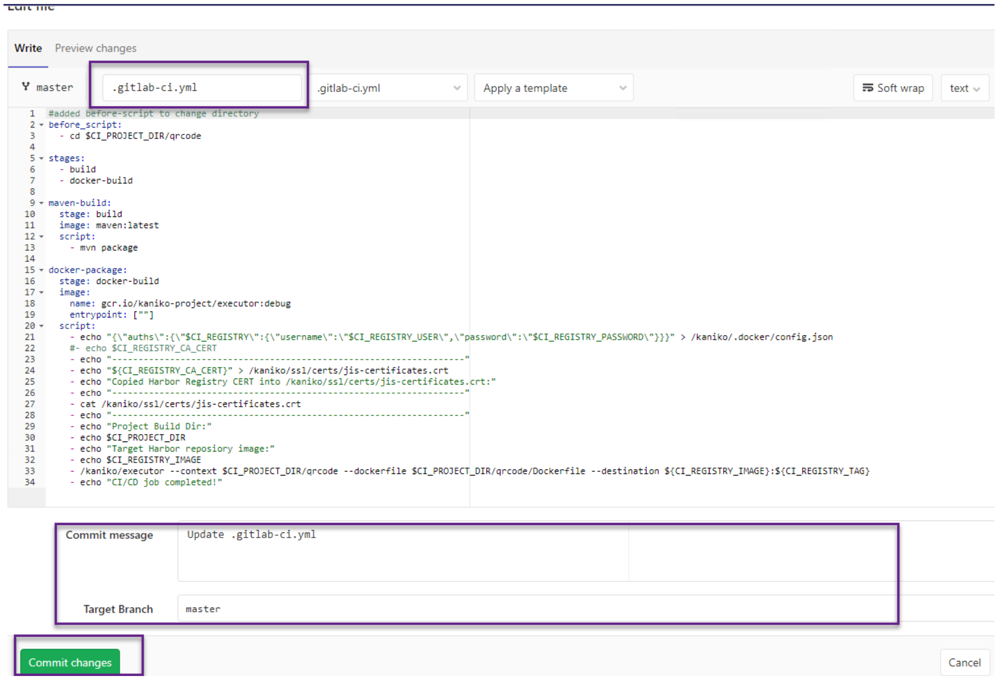
Update .gitlab-ci.yml added use of CI_REGISTRY_TAG variable. NB! Use ...
Daniel Zilberman authored 22 hours ago
6ffef109

Name	Last commit	Last update
qrcode	Initial seed from cloned GitHub repo to test CI/CD pipeline	1 day ago
.gitlab-ci.yml	Update .gitlab-ci.yml added use of CI_REGISTRY_TAG variable. NB! Use ...	22 hours ago
README.md	Initial seed from cloned GitHub repo to test CI/CD pipeline	1 day ago
gitlab-ci-docker.yaml	Initial seed from cloned GitHub repo to test CI/CD pipeline	1 day ago

README.md

dockersample - QRCodeGenerator

- Add default CI/CD pipeline script file (**.gitlab-ci.yaml**) at the root level of the Project using "CI/CD Configuration" option
  NOTE: .**gitlab-ci.yaml** CI/CD script syntax should comply with structure and stages defined in accordance with documentation: https://docs.gitlab.com/ee/ci/yaml/README.html
- Edit contents of that script file from GitLab IDE (or outside of it and use Git client to commit changes to project repository):



```
#added before-script to change directory
before_script:
  - cd $CI_PROJECT_DIR/qrcode

stages:
  - build
  - docker-build

maven-build:
  stage: build
  image: maven:latest
  script:
    - mvn package

docker-package:
  stage: docker-build
  image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""]
  script:
    - echo "{\"auths\":{\"$CI_REGISTRY\":{\"username\":\"$CI_REGISTRY_USER\",\"password\":\"$CI_REGISTRY_PASSWORD\"}}}" > /kaniko/.docker/config.json
    #- echo $CI_REGISTRY_CA_CERT
    - echo "----------------------------------------------------------------"
    - echo "${CI_REGISTRY_CA_CERT}" > /kaniko/ssl/certs/jis-certificates.crt
    - echo "Copied Harbor Registry CERT into /kaniko/ssl/certs/jis-certificates.crt:"
    - echo "----------------------------------------------------------------"
    - cat /kaniko/ssl/certs/jis-certificates.crt
    - echo "----------------------------------------------------------------"
    - echo "Project Build Dir:"
    - echo $CI_PROJECT_DIR
    - echo "Target Harbor reposiory image:"
    - echo $CI_REGISTRY_IMAGE
    - /kaniko/executor --context $CI_PROJECT_DIR/qrcode --dockerfile $CI_PROJECT_DIR/qrcode/Dockerfile --destination ${CI_REGISTRY_IMAGE}:${CI_REGISTRY_TAG}
    - echo "CI/CD job completed!"
```

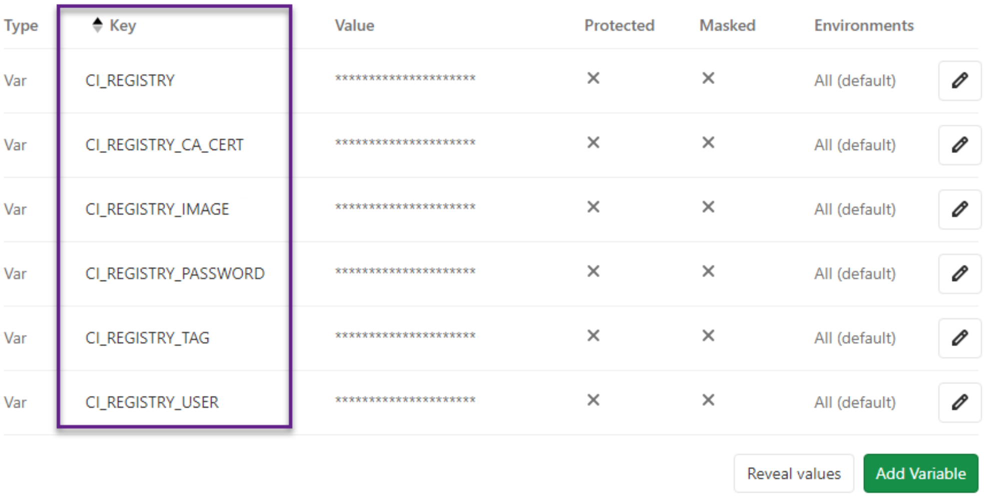
Commit message: Update .gitlab-ci.yml

Target Branch: master

NOTES:

✓ **.gitlab-ci.yaml** is a default CI/CD pipeline file name for any project, other pipeline definitions can be invoked from it

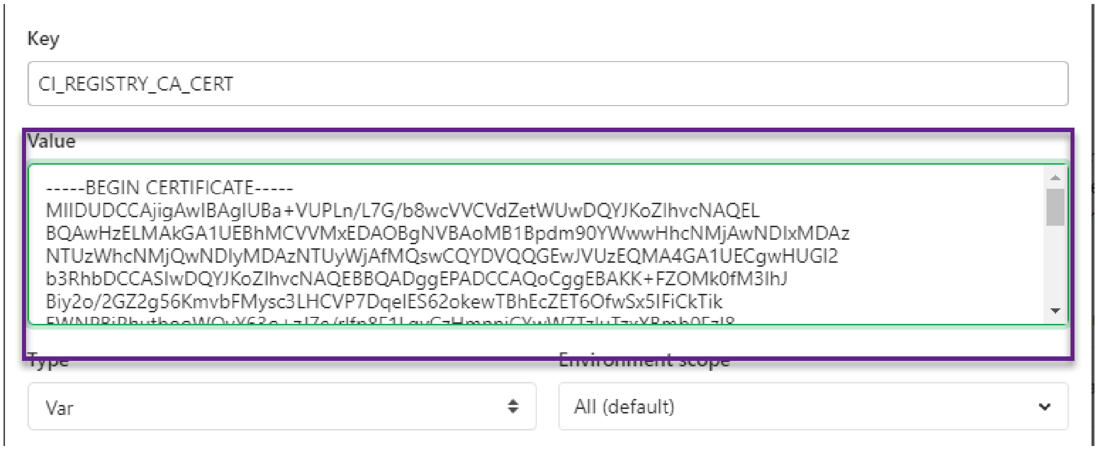
- ✓ Environment variables values referenced in pipeline scripts (CI_REGISTRY, CI_REGISTRY_USER etc.) should be set via “Settings ➔ CI/CD ➔ Variables” section of the project:



Type	Key	Value	Protected	Masked	Environments	
Var	CI_REGISTRY	********************	×	×	All (default)	✎
Var	CI_REGISTRY_CA_CERT	********************	×	×	All (default)	✎
Var	CI_REGISTRY_IMAGE	********************	×	×	All (default)	✎
Var	CI_REGISTRY_PASSWORD	********************	×	×	All (default)	✎
Var	CI_REGISTRY_TAG	********************	×	×	All (default)	✎
Var	CI_REGISTRY_USER	********************	×	×	All (default)	✎

Scope of those variables should be normally set as ‘Environment scope’, additional options to protect their values, if required, are available via GitLab settings.

- ✓ For variables that contain values of Container Image registry (Harbor) certificate, make sure it keeps its original format, as shown below:



Key: CI_REGISTRY_CA_CERT

Value:

```
-----BEGIN CERTIFICATE-----
MIIDUDCCAjigAwIBAgIUBa+VUPLn/L7G/b8wcVVCVdZetWUwDQYJKoZIhvcNAQEL
BQAwHzELMAkGA1UEBhMCVVMxEDAOBgNVBAoMB1Bpdm90YWwwHhcNMjAwNDIxMDAz
NTUzWhcNMjQwNDIyMDAzNTUyWjAfMQswCQYDVQQGEwJVUzEQMA4GA1UECgwHUGI2
b3RhbDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKK+FZOMk0fM3IhJ
Biy2o/2GZ2g56KmvbFMysc3LHCVP7DqeIES62okewTBhEcZET6OfwSx5IFiCkTik
```

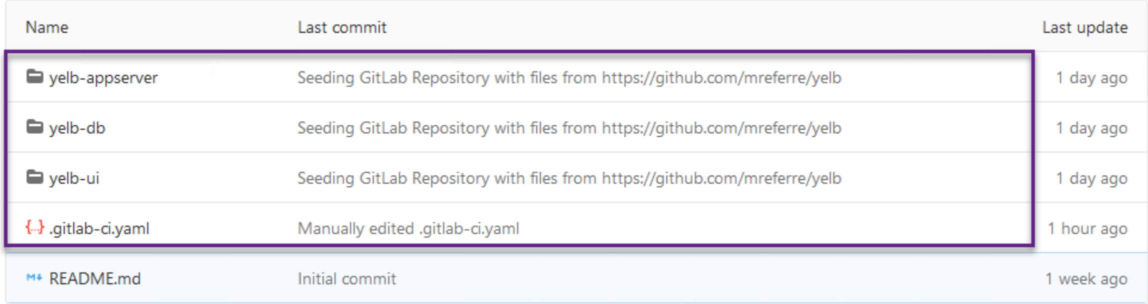
Type: Var

Environment scope: All (default)

(reason being that we will basically automate commands like: “**docker login ${CI_REGISTRY} -u ${CI_REGISTRY_USER} -p ${CI_REGISTRY_PASSWORD}**” to run as API call from CI/CD script)

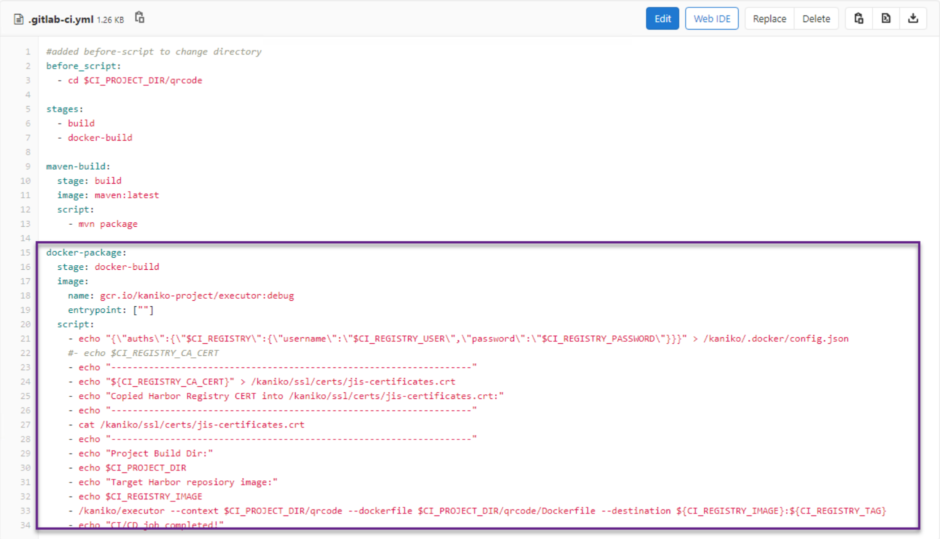
- To bypass a need for executor containers to have privileged access to Docker daemon to run “docker build” commands, we can use unprivileged access via Google Kaniko execution environment: https://docs.gitlab.com/ee/ci/ (OK for running builds, not for running container images) See details in: https://docs.gitlab.com/ee/ci/docker/using_kaniko.html

- Edit default CI/CD pipeline script (**.gitlab-ci.yaml**) at the root of GitLab project directory:



Name	Last commit	Last update
yelb-appserver	Seeding GitLab Repository with files from https://github.com/mreferre/yelb	1 day ago
yelb-db	Seeding GitLab Repository with files from https://github.com/mreferre/yelb	1 day ago
yelb-ui	Seeding GitLab Repository with files from https://github.com/mreferre/yelb	1 day ago
.gitlab-ci.yaml	Manually edited .gitlab-ci.yaml	1 hour ago
README.md	Initial commit	1 week ago

  - Example of a working version of above script that builds and pushes Docker container images from GitLab project sub-folders is shown below:



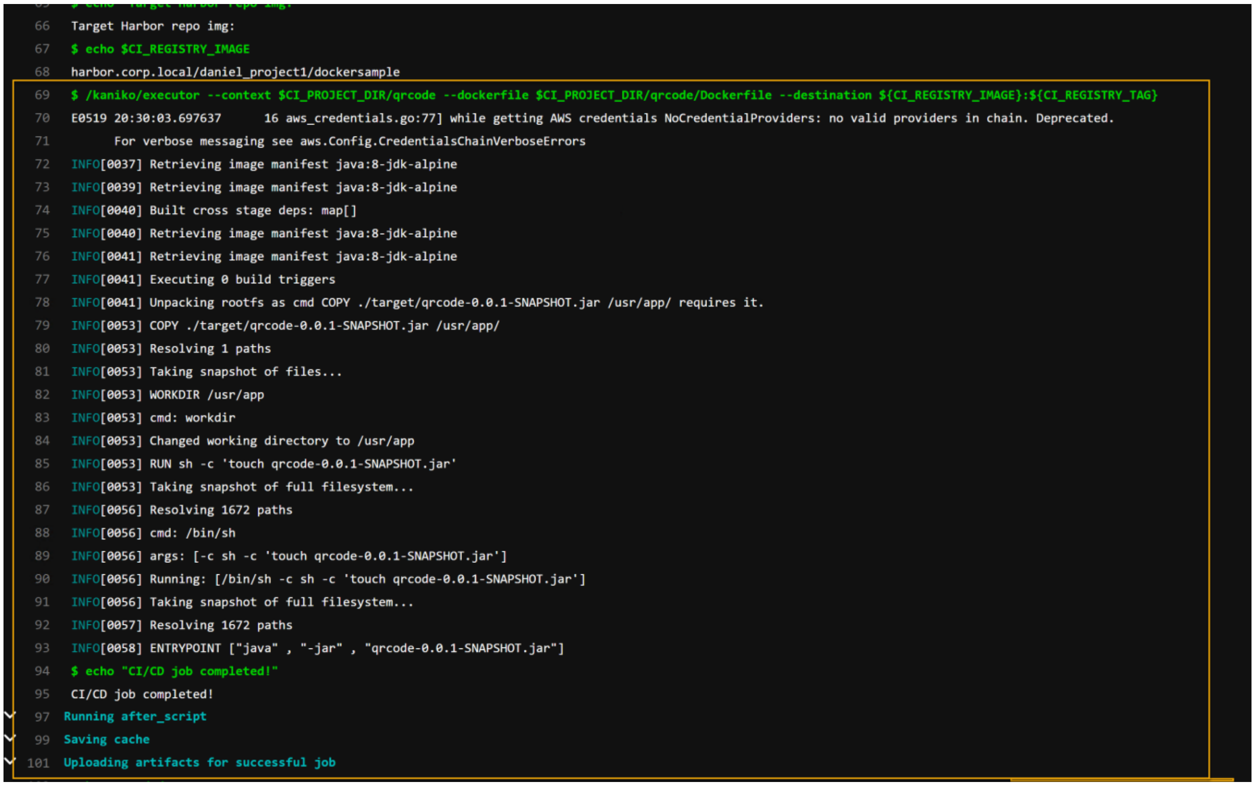
```
#added before-script to change directory
before_script:
  - cd $CI_PROJECT_DIR/qrcode

stages:
  - build
  - docker-build

maven-build:
  stage: build
  image: maven:latest
  script:
    - mvn package

docker-package:
  stage: docker-build
  image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""]
  script:
    - echo "{\"auths\":{\"$CI_REGISTRY\":{\"username\":\"$CI_REGISTRY_USER\",\"password\":\"$CI_REGISTRY_PASSWORD\"}}}" > /kaniko/.docker/config.json
    #- echo $CI_REGISTRY_CA_CERT
    - echo "------------------------------------------------------------------"
    - echo "${CI_REGISTRY_CA_CERT}" > /kaniko/ssl/certs/jis-certificates.crt
    - echo "Copied Harbor Registry CERT into /kaniko/ssl/certs/jis-certificates.crt:"
    - echo "------------------------------------------------------------------"
    - cat /kaniko/ssl/certs/jis-certificates.crt
    - echo "------------------------------------------------------------------"
    - echo "Project Build Dir:"
    - echo $CI_PROJECT_DIR
    - echo "Target Harbor reposiory image:"
    - echo $CI_REGISTRY_IMAGE
    - /kaniko/executor --context $CI_PROJECT_DIR/qrcode --dockerfile $CI_PROJECT_DIR/qrcode/Dockerfile --destination ${CI_REGISTRY_IMAGE}:${CI_REGISTRY_TAG}
    - echo "CI/CD job completed!"
```

NOTES:

✓ Properly formatted (no TAB characters) snippet of above **.gitlab-ci.yml** file, including optional debug prints of variable values, can be found below:

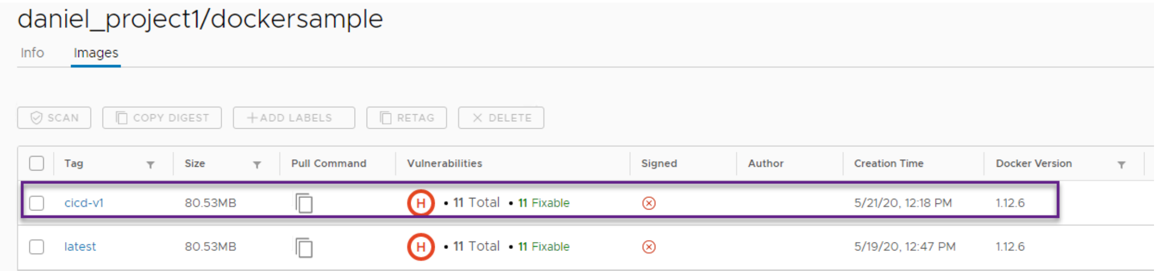
```
#added before-script to change working directory
before_script:
  - cd $CI_PROJECT_DIR/qrcode

stages:
  - build
  - docker-build

maven-build:
  stage: build
  image: maven:latest
  script:
    - mvn package

docker-package:
  stage: docker-build
  image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""]
  script:
    - echo
"{\"auths\":{\"$CI_REGISTRY\":{\"username\":\"$CI_REGISTRY_USER\",\"pas
sword\":\"$CI_REGISTRY_PASSWORD\"}}}" > /kaniko/.docker/config.json
    - echo $CI_REGISTRY_CA_CERT
    - echo "----------------------------------------------------------
--------"
    - echo "${CI_REGISTRY_CA_CERT}" > /kaniko/ssl/certs/jis-
certificates.crt
    - echo "Copied Harbor Registry CERT into /kaniko/ssl/certs/jis-
certificates.crt:"
    - echo "----------------------------------------------------------
--------"
    - cat /kaniko/ssl/certs/jis-certificates.crt
    - echo "----------------------------------------------------------
--------"
    - echo "Project Build Dir:"
    - echo $CI_PROJECT_DIR
    - echo "Target Harbor repo img:"
    - echo $CI_REGISTRY_IMAGE
    - /kaniko/executor --context $CI_PROJECT_DIR/qrcode --dockerfile
$CI_PROJECT_DIR/qrcode/Dockerfile --destination
${CI_REGISTRY_IMAGE}:${CI_REGISTRY_TAG}
    - echo "CI/CD job completed!"
```

- ✓ This CI/CD script example is using **gcr.io/kaniko-project/executor:debug** container image for build jobs execution. It compiles source code, builds image defined in Dockerfile and pushes it into registry defined by CI_REGISTRY_IMAGE environment variable with a tag defined in the CI_REGISTRY_TAG
- Pipeline execution progress can be monitored in real time and after its completion via GitLab CI/CD UI, as shown below:



```
Target Harbor repo img:
$ echo $CI_REGISTRY_IMAGE
harbor.corp.local/daniel_project1/dockersample
$ /kaniko/executor --context $CI_PROJECT_DIR/qrcode --dockerfile $CI_PROJECT_DIR/qrcode/Dockerfile --destination ${CI_REGISTRY_IMAGE}:${CI_REGISTRY_TAG}
E0519 20:30:03.697637      16 aws_credentials.go:77] while getting AWS credentials NoCredentialProviders: no valid providers in chain. Deprecated.
      For verbose messaging see aws.Config.CredentialsChainVerboseErrors
INFO[0037] Retrieving image manifest java:8-jdk-alpine
INFO[0039] Retrieving image manifest java:8-jdk-alpine
INFO[0040] Built cross stage deps: map[]
INFO[0040] Retrieving image manifest java:8-jdk-alpine
INFO[0041] Retrieving image manifest java:8-jdk-alpine
INFO[0041] Executing 0 build triggers
INFO[0041] Unpacking rootfs as cmd COPY ./target/qrcode-0.0.1-SNAPSHOT.jar /usr/app/ requires it.
INFO[0053] COPY ./target/qrcode-0.0.1-SNAPSHOT.jar /usr/app/
INFO[0053] Resolving 1 paths
INFO[0053] Taking snapshot of files...
INFO[0053] WORKDIR /usr/app
INFO[0053] cmd: workdir
INFO[0053] Changed working directory to /usr/app
INFO[0053] RUN sh -c 'touch qrcode-0.0.1-SNAPSHOT.jar'
INFO[0053] Taking snapshot of full filesystem...
INFO[0056] Resolving 1672 paths
INFO[0056] cmd: /bin/sh
INFO[0056] args: [-c sh -c 'touch qrcode-0.0.1-SNAPSHOT.jar']
INFO[0056] Running: [/bin/sh -c sh -c 'touch qrcode-0.0.1-SNAPSHOT.jar']
INFO[0056] Taking snapshot of full filesystem...
INFO[0057] Resolving 1672 paths
INFO[0058] ENTRYPOINT ["java" , "-jar" , "qrcode-0.0.1-SNAPSHOT.jar"]
$ echo "CI/CD job completed!"
CI/CD job completed!
Running after_script
Saving cache
Uploading artifacts for successful job
```

- If Pipeline execution is successful, in the target Harbor project/repository there should be a new built/pushed image(s) from GitLab project, tagged according to passed value of CI_REGISTRY_TAG variable and scanned for vulnerabilities, per project settings:



daniel_project1/dockersample

□	Tag	Size	Pull Command	Vulnerabilities	Signed	Author	Creation Time	Docker Version
□	cicd-v1	80.53MB		H • 11 Total • 11 Fixable	⊗		5/21/20, 12:18 PM	1.12.6
□	latest	80.53MB		H • 11 Total • 11 Fixable	⊗		5/19/20, 12:47 PM	1.12.6

- Notes on additional GitLab CI/CD resources and Best practices:
  - Examples of end-to-end Docker container build and deployment automation via GitLab CI/CD pipelines can be found in various blogs such as: https://sanderknape.com/2019/02/automated-deployments-kubernetes-gitlab/
  - Operations teams that just need to run stabilized builds may use “Auto DevOps” GitLab mode and run pipelines defined in **.gitlab-ci.yaml** as in example above.
  - Developers may need to create their own customized pipelines, please see GitLab documentation: https://gitlab.acelab.local/help/ci/pipelines/settings#custom-ci-configuration-path