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 3
2. Pre-requisites 3
3. Add Existing TKGI K8s Cluster to GitLab project 4
4. Install and configure GitLab Runner using Helm chart, associate it with project 8
5. Configure and Execute CI/CD Pipeline for Building, Tagging and Publishing Application Image into Harbor Registry 14
6. Configure and Execute CI/CD Pipeline for Deployment of Application from Registry to K8s Cluster 21
7. Conclusion 29

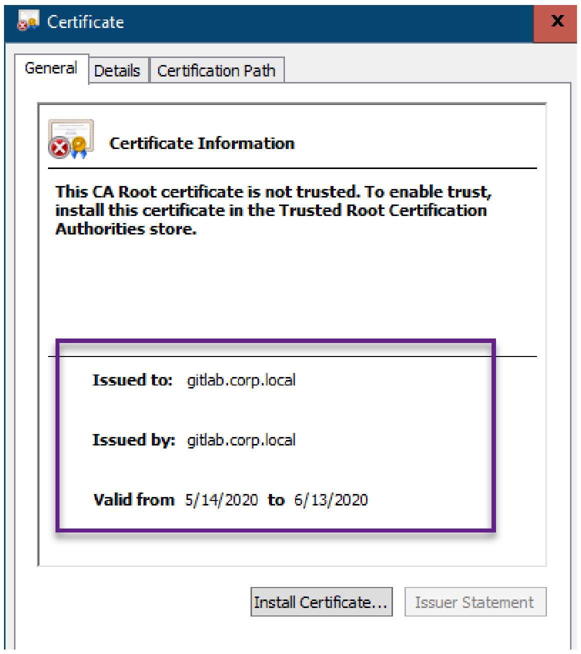
1. Introduction

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

General | Details | Certification Path

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

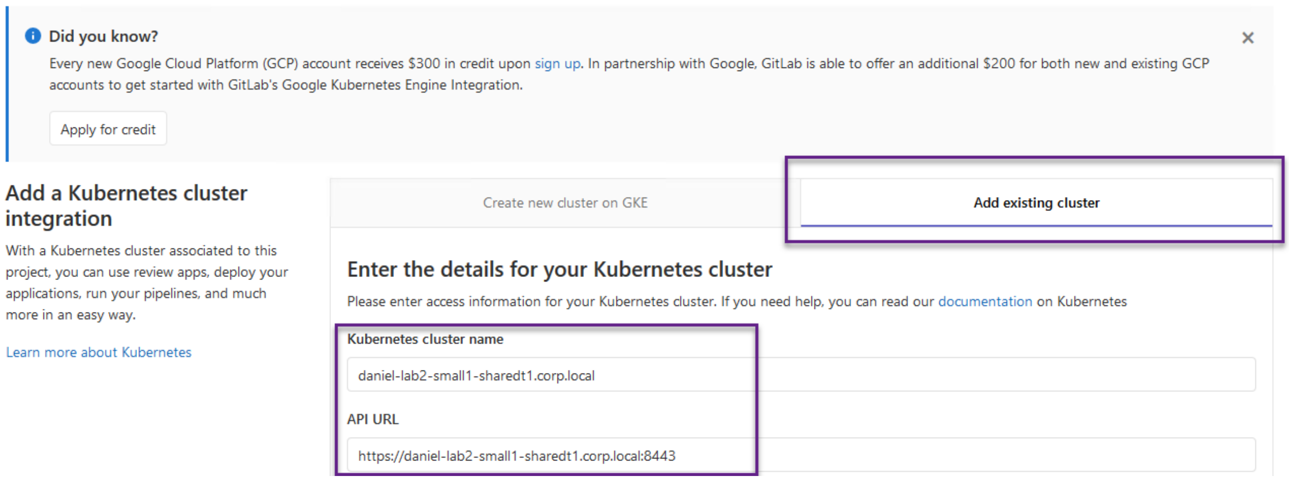
Valid from 5/14/2020 to 6/13/2020

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
```

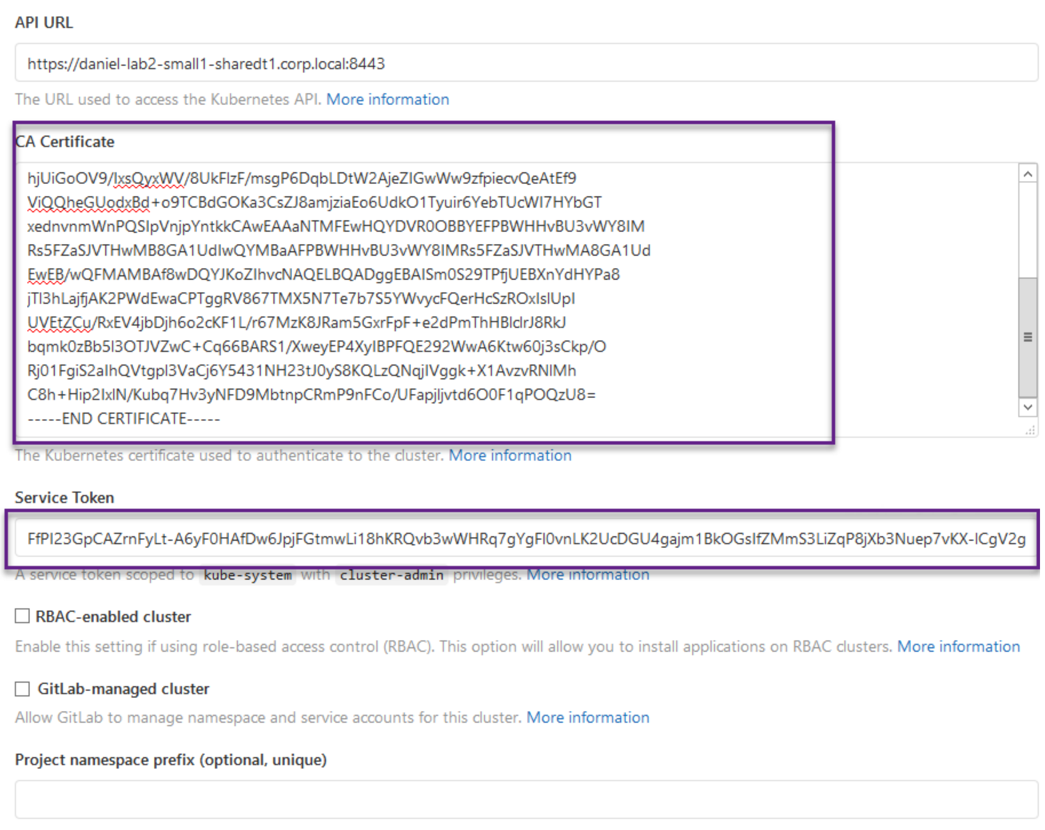
Create Service account and Cluster Role Binding in the target cluster:

```
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
jTI3hLajfjAK2PWdEwaCPTggRV867TMX5N7Te7b7S5YWvycFQerHcSzROxIslUpI
UVEtZCu/RxEV4jbDjh6o2cKF1L/r67MzK8JRam5GxrFpF+e2dPmThHBlclrJ8RkJ
bqmk0zBb5l3OTJVZwC+Cq66BARS1/XweyEP4XyIBPFQE292WwA6Ktw60j3sCkp/O
Rj01FgiS2aIhQVtgpl3VaCj6Y5431NH23tJ0yS8KQLzQNqjIVggk+X1AvzvRNIMh
C8h+Hip2IxlN/Kubq7Hv3yNFD9MbtnpCRmP9nFCo/UFapjljvtd6O0F1qPOQzU8=
-----END CERTIFICATE-----
```

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

Service Token

FfPI23GpCAZrnFyLt-A6yF0HAfDw6JpjFGtmwLi18hKRQvb3wWHRq7gYgFI0vnLK2UcDGU4gajm1BkOGsIfZMmS3LiZqP8jXb3Nuep7vKX-ICgV2g

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

☐ RBAC-enabled cluster

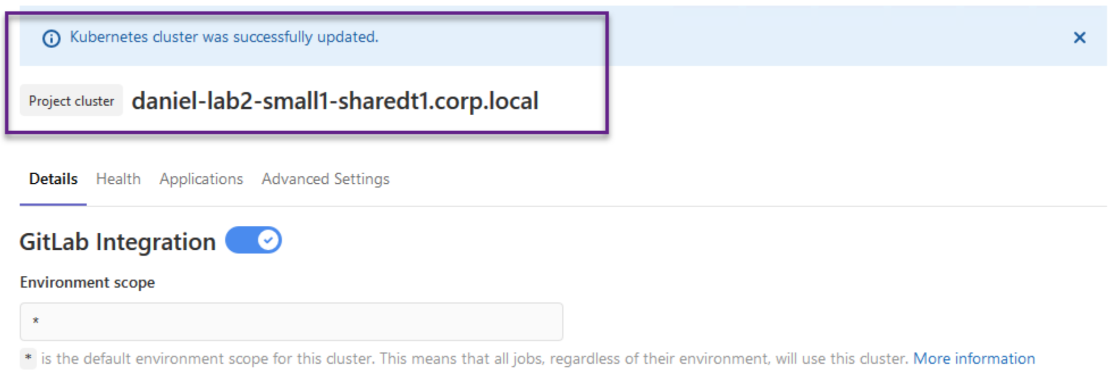
Enable this setting if using role-based access control (RBAC). This option will allow you to install applications on RBAC clusters. More information

☐ 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 update on GitLab UI**:



Kubernetes cluster was successfully updated.

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

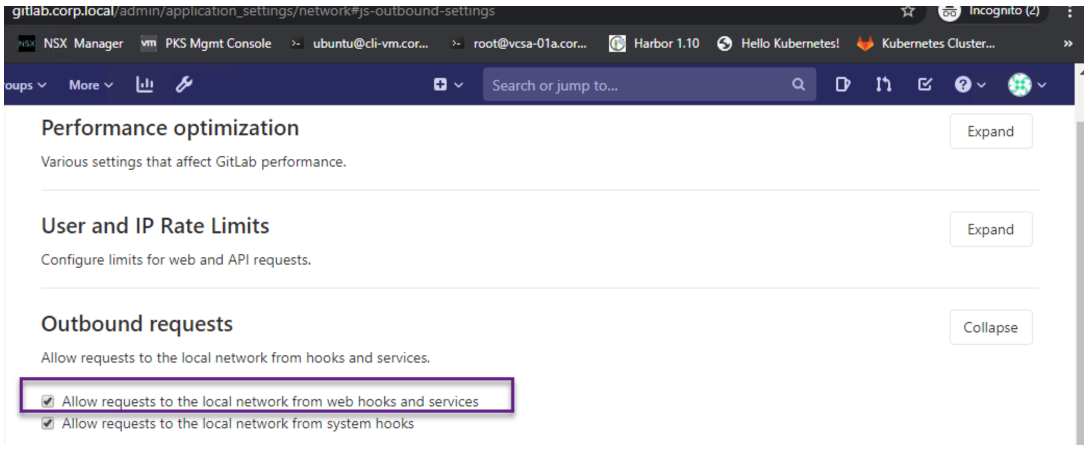
GitLab Integration

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 addresses are on the same network), we may need to explicitly allow requests to local networks from Web Hooks and Services from GitLab Admin UI, 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)

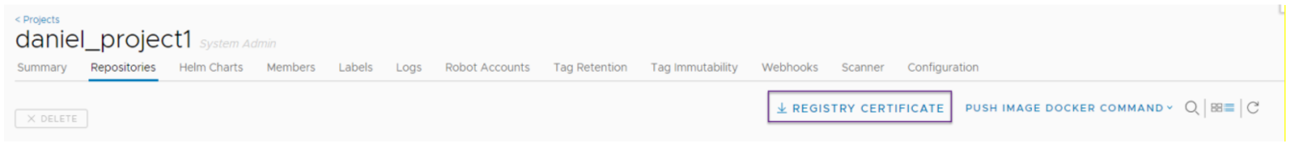
- Install Helm client/server following documentation , validate that it can reach general repositories containing **gitlab-runner** charts

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 (e.g. 'gitlabrunner' or any other valid name)

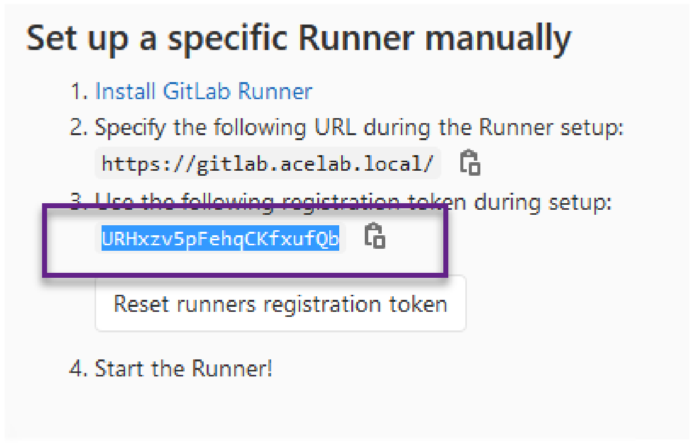
```
kubectl create ns gitlabrunner
```

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 deployment configuration 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**

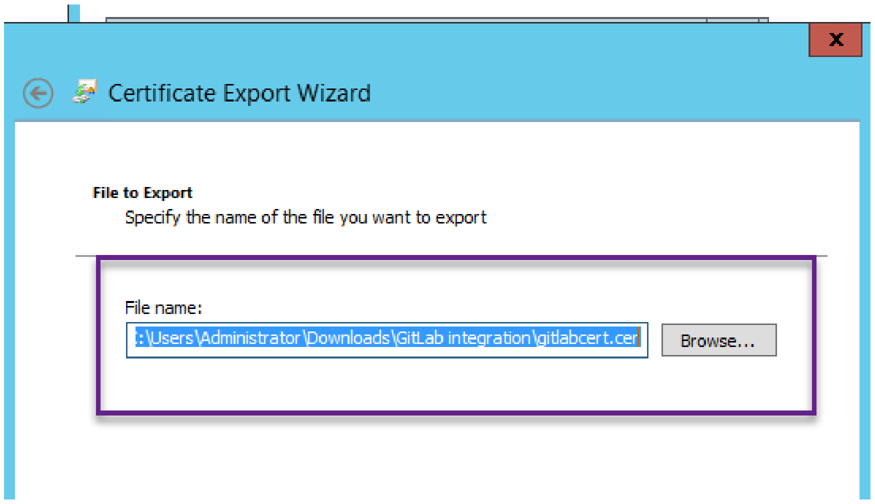
```
## For RBAC support:
rbac:
  create: true
  ## Define specific rbac permissions.
  # resources: ["pods", "pods/exec", "secrets"]
  # verbs: ["get", "list", "watch", "create", "patch", "delete"]
..
```

  (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:


Certificate Export Wizard

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

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

Notes:

- ✓ 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 that the secret got created:

```
$ kubectl get secrets

NAME                TYPE                                DATA   AGE
default-token-ckt9m   k11ubernetes.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 of GitLab VM file system:

```
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 (PSP) 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 this 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 GitLab 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 from the repository

An output of 'helm install' command should look like:

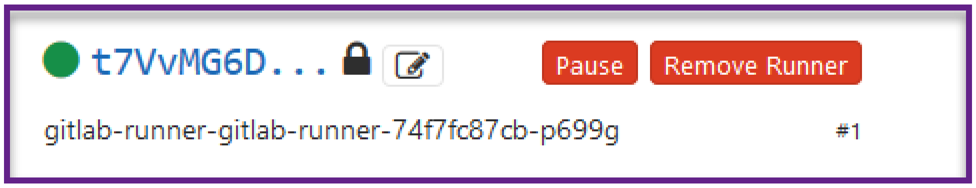
```
NAME: gitlab-runner
LAST DEPLOYED: Sun May  3 05:46:50 2020
NAMESPACE: gitlabrunner
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Your GitLab Runner should now be registered against the GitLab instance
reachable at: https://gitlab.acelab.local
```

- (Optional) validate that Runner deployment/pod 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 ➔ Runners** of the Project and check whether Runner shows as active (green color):


● t7VvMG6D...

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

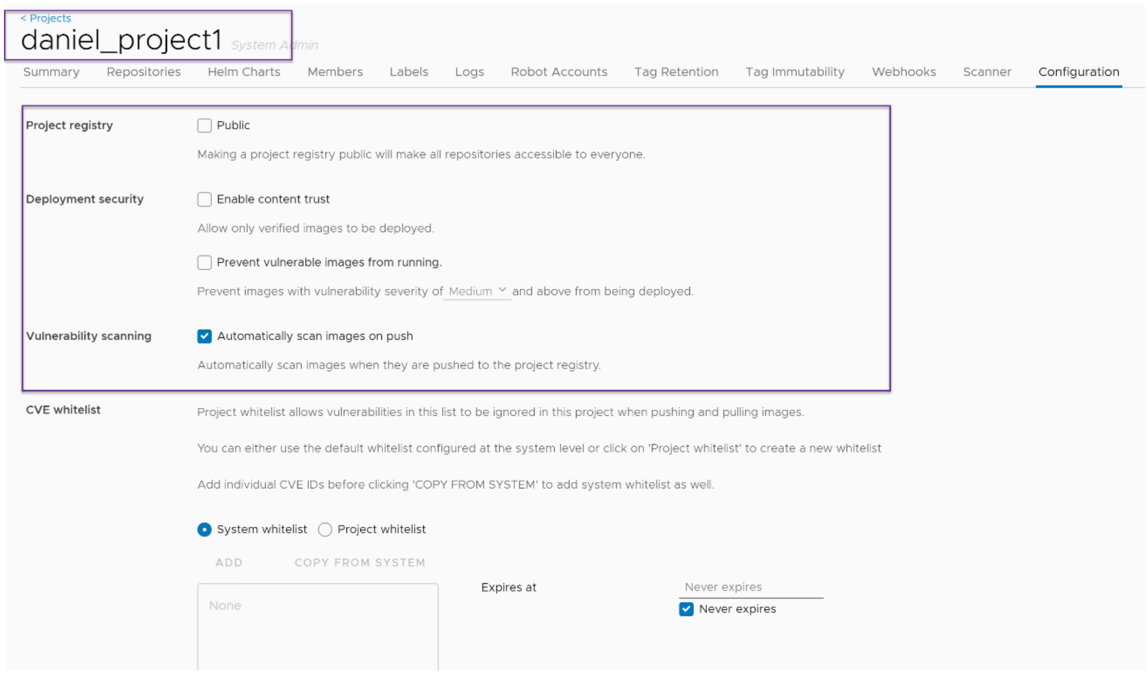
Notes:

- ✓ Usually, Runner configured on a project level will show up in 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 of 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 CI/CD Pipeline for Building, Tagging and Publishing Application Image into Harbor Registry*

We will start by building "Continuous Integration" part of our CI/CD pipeline. It compiles a simple SpringBoot based microservice from its Java source code using Maven, continues to build a container image using provided Dockerfile and pushes built/tagged image into designated project in the Harbor 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
- [x] 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

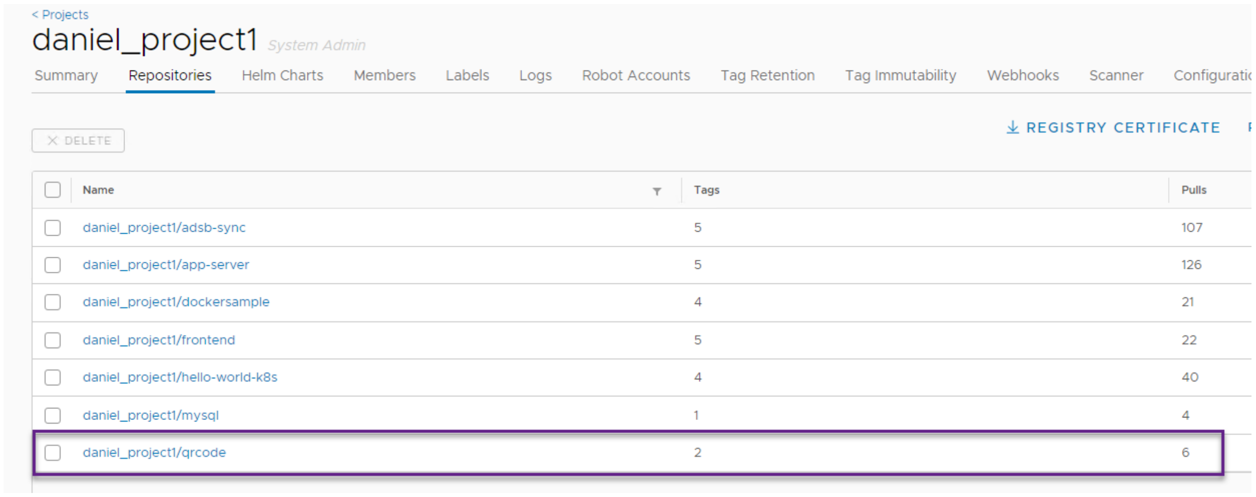
Add individual CVE IDs before clicking 'COPY FROM SYSTEM' to add system whitelist as well.

[✓ System whitelist] Project whitelist

None

Expires at: Never expires
- [x] Never expires

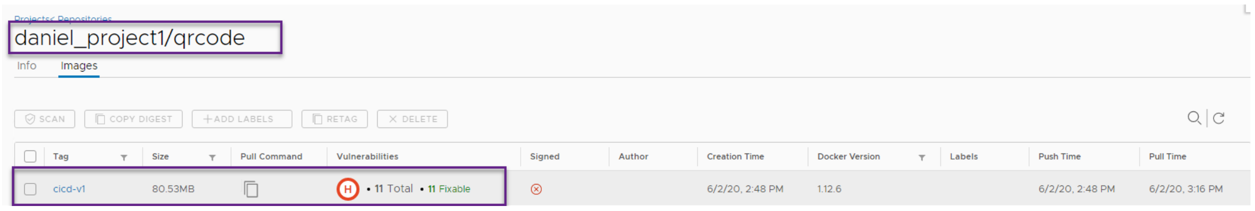
- There are other image repositories configured for that project, shown below:


< Projects

daniel_project1 System Admin

Name	Tags	Pulls
daniel_project1/adsb-sync	5	107
daniel_project1/app-server	5	126
daniel_project1/dockersample	4	21
daniel_project1/frontend	5	22
daniel_project1/hello-world-k8s	4	40
daniel_project1/mysql	1	4
daniel_project1/qrcode	2	6

Our GitLab pipeline will be building and pushing images into ‘**daniel_project1/qrcode’** repository that may contain previously uploaded images:

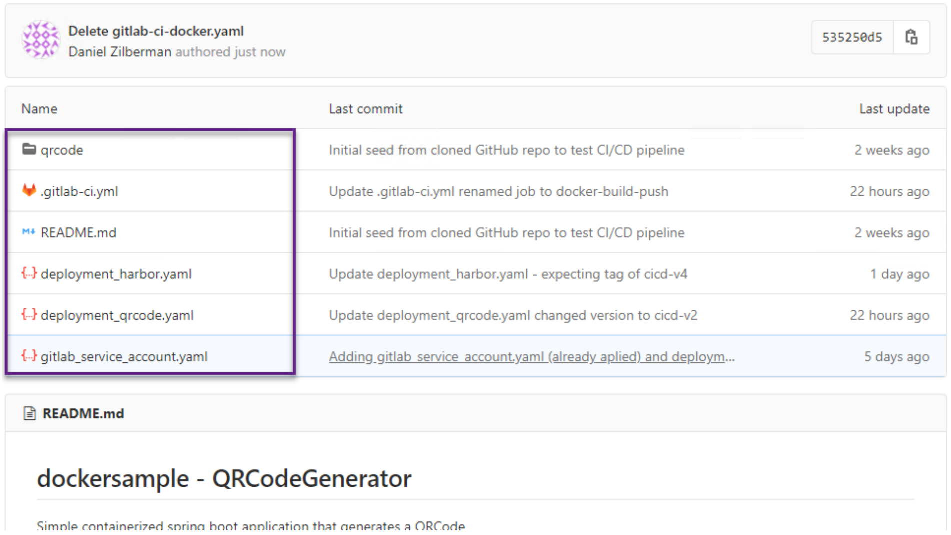

Projects< Repositories

daniel_project1/qrcode

Tag	Size	Pull Command	Vulnerabilities	Signed	Author	Creation Time	Docker Version	Labels	Push Time	Pull Time
cicd-v1	80.53MB		H • 11 Total • 11 Fixable	⊗		6/2/20, 2:48 PM	1.12.6		6/2/20, 2:48 PM	6/2/20, 3:16 PM

There is an existing version of an image with ‘cicd-v1’ tag

- File structure of the example GitLab project called **dockersample** is shown below:



Delete gitlab-ci-docker.yaml
Daniel Zilberman authored just now
535250d5

Name	Last commit	Last update
qrcode	Initial seed from cloned GitHub repo to test CI/CD pipeline	2 weeks ago
.gitlab-ci.yml	Update .gitlab-ci.yml renamed job to docker-build-push	22 hours ago
README.md	Initial seed from cloned GitHub repo to test CI/CD pipeline	2 weeks ago
deployment_harbor.yaml	Update deployment_harbor.yaml - expecting tag of cicd-v4	1 day ago
deployment_qrcode.yaml	Update deployment_qrcode.yaml changed version to cicd-v2	22 hours ago
gitlab_service_account.yaml	Adding gitlab_service_account.yaml (already aplied) and deploym...	5 days ago

README.md

dockersample - QRCodeGenerator

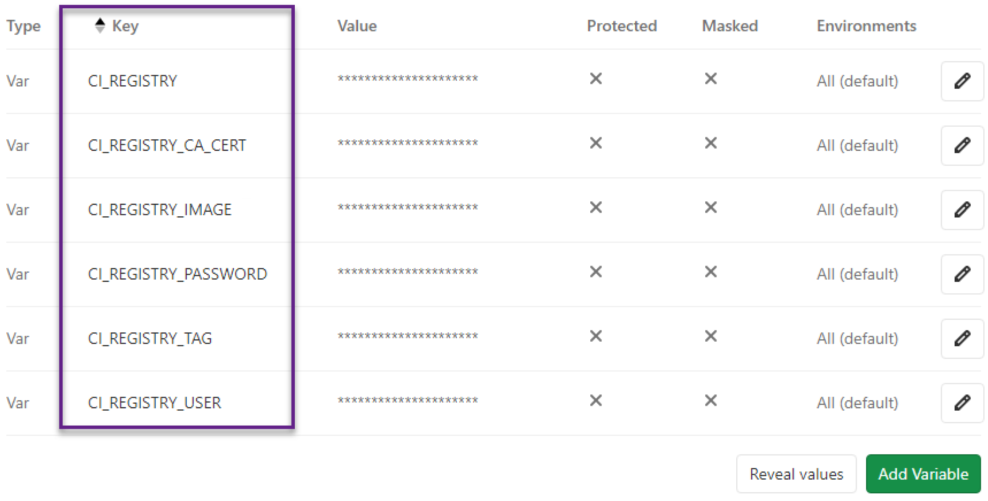
Simple containerized spring boot application that generates a QRCode

- 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):

NOTES:

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

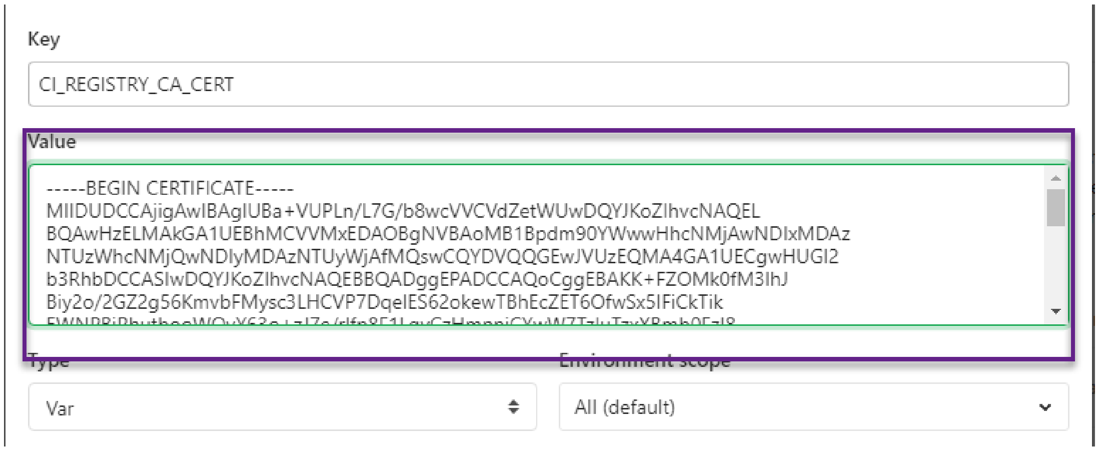
✓ Environment variables values referenced in pipeline scripts (CI_REGISTRY, CI_REGISTRY_USER etc.) should be defned via "Settings ➔ CI/CD ➔ Variables" section of the project:



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 (such as for passwords etc.) are available via GitLab settings.

✓ For variables that contain values of SSL 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
```

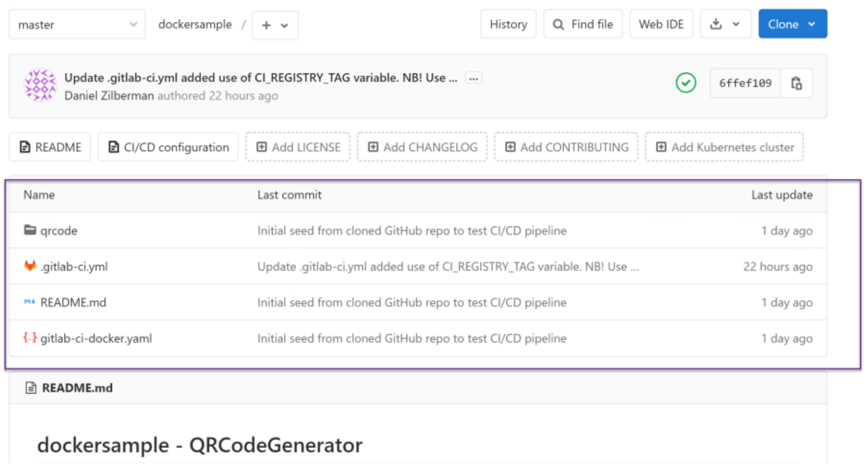
Type: Var

Environment scope: All (default)

(reason being that pipeline script 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:



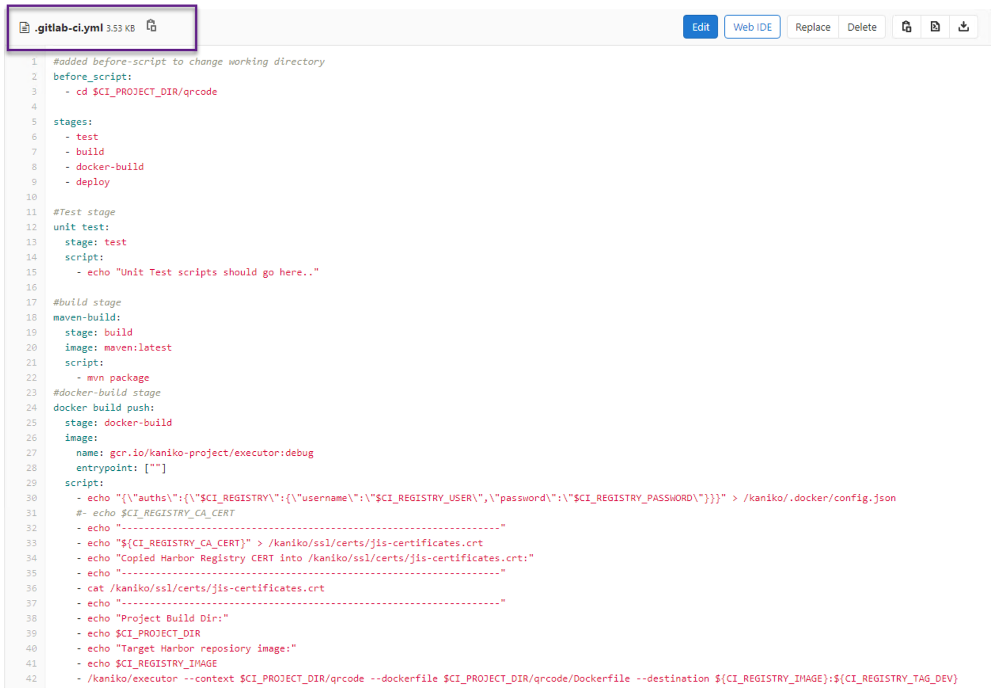
Update .gitlab-ci.yml added use of CI_REGISTRY_TAG variable. NB! Use ...
Daniel Zilberman authored 22 hours ago — 6ffef109

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

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



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

stages:
  - test
  - build
  - docker-build
  - deploy

#Test stage
unit test:
  stage: test
  script:
    - echo "Unit Test scripts should go here.."

#build stage
maven-build:
  stage: build
  image: maven:latest
  script:
    - mvn package
#docker-build stage
docker build push:
  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_DEV}
```

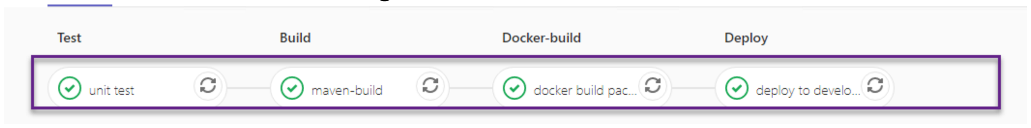
NOTES:

✓ Correctly formatted section of above **.gitlab-ci.yml** file can be found below:

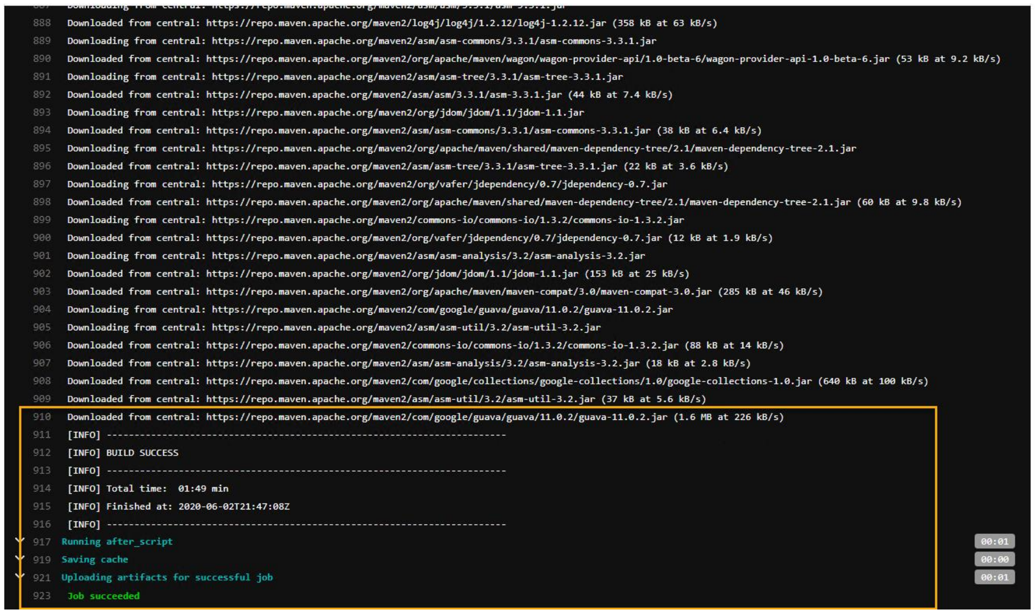
```
#added before-script to change working directory
before_script:
  - cd $CI_PROJECT_DIR/qrcode

stages:
  - test
  - build
  - docker-build
  - deploy

#Test stage
unit test:
  stage: test
  script:
    - echo "Unit Test scripts should go here.."

#build stage
maven-build:
  stage: build
  image: maven:latest
  script:
    - mvn package
#docker-build stage
docker build push:
  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 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_DEV}
    - echo "CI/CD job completed, APP image should be available at:"
    - echo ${CI_REGISTRY_IMAGE}:${CI_REGISTRY_TAG_DEV}
  environment:
    name: development
  #build docker app only when a pipeline on master branch is run
  only:
  - master
```

NOTES:

- ✓ This CI/CD script contains the following stages:
  - **test** to perform app unit testing (not implemented fully)
  - **build** to perform Maven build of SpringBoot service from Java source code
  - **docker-build** to perform Docker build/tag/push operations to publish images to container registry
  - **deploy** to perform deployment of containerized application from container registry to target K8s cluster (covered in detail in Section 6)
- ✓ It is using **maven:latest** container image for **build** stage jobs execution .
- ✓ Then **gcr.io/kaniko-project/executor:debug** container image used for **docker-build** stage jobs execution to build an 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_DEV
- ✓ For now, we only want to deploy the application when pipeline on **master** branch is run hence the 'only -master' condition at the end of stage script
- • Pipeline execution progress and results (success/failure) for each stage can be monitored in real time as shown for build stage below:

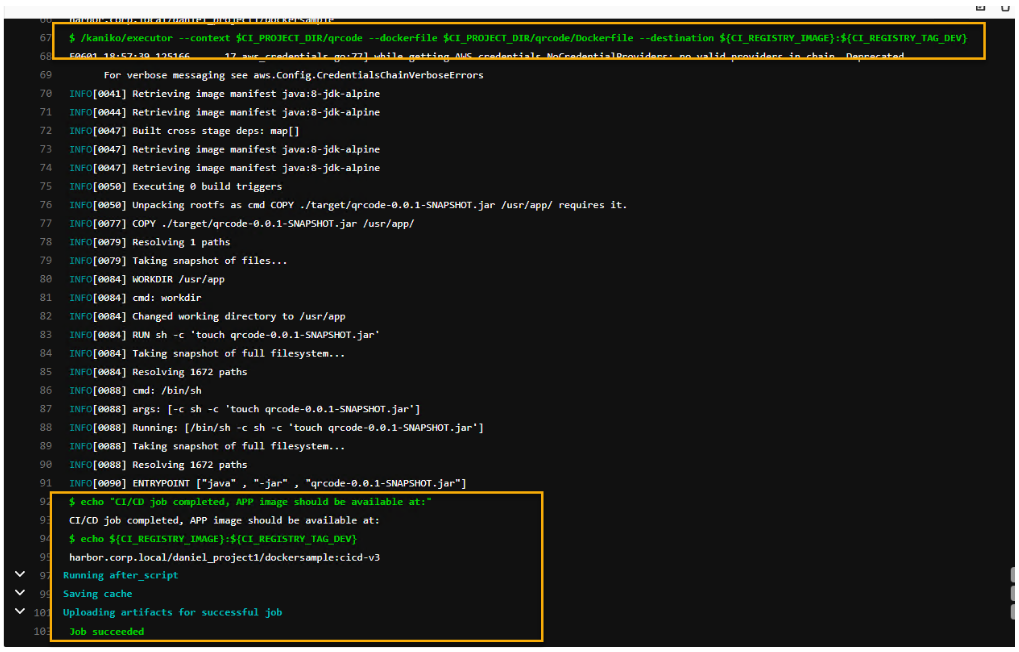


and reviewed in detail during or after its completion via GitLab CI/CD UI, as shown below for **build** stage:



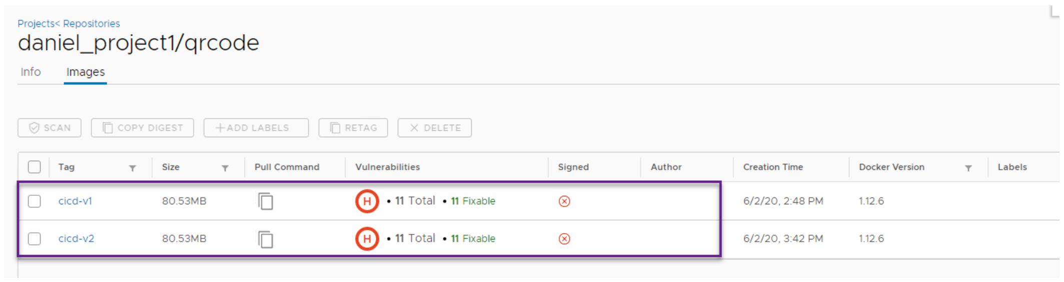
```
Downloaded from central: https://repo.maven.apache.org/maven2/log4j/log4j/1.2.12/log4j-1.2.12.jar (358 kB at 63 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/asm/asm-commons/3.3.1/asm-commons-3.3.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/wagon/wagon-provider-api/1.0-beta-6/wagon-provider-api-1.0-beta-6.jar (53 kB at 9.2 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/asm/asm-tree/3.3.1/asm-tree-3.3.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/asm/asm/3.3.1/asm-3.3.1.jar (44 kB at 7.4 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/jdom/jdom/1.1/jdom-1.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/asm/asm-commons/3.3.1/asm-commons-3.3.1.jar (38 kB at 6.4 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-dependency-tree/2.1/maven-dependency-tree-2.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/asm/asm-tree/3.3.1/asm-tree-3.3.1.jar (22 kB at 3.6 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/vafer/jdependency/0.7/jdependency-0.7.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-dependency-tree/2.1/maven-dependency-tree-2.1.jar (60 kB at 9.8 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/commons-io/commons-io/1.3.2/commons-io-1.3.2.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/vafer/jdependency/0.7/jdependency-0.7.jar (12 kB at 1.9 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/asm/asm-analysis/3.2/asm-analysis-3.2.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/jdom/jdom/1.1/jdom-1.1.jar (153 kB at 25 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-compat/3.0/maven-compat-3.0.jar (285 kB at 46 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/com/google/guava/guava/11.0.2/guava-11.0.2.jar
Downloading from central: https://repo.maven.apache.org/maven2/asm/asm-util/3.2/asm-util-3.2.jar
Downloaded from central: https://repo.maven.apache.org/maven2/commons-io/commons-io/1.3.2/commons-io-1.3.2.jar (88 kB at 14 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/asm/asm-analysis/3.2/asm-analysis-3.2.jar (18 kB at 2.8 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/com/google/collections/google-collections/1.0/google-collections-1.0.jar (640 kB at 100 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/asm/asm-util/3.2/asm-util-3.2.jar (37 kB at 5.6 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/com/google/guava/guava/11.0.2/guava-11.0.2.jar (1.6 MB at 226 kB/s)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:49 min
[INFO] Finished at: 2020-06-02T21:47:08Z
[INFO] ------------------------------------------------------------------------
Running after_script                                                                00:01
Saving cache                                                                        00:00
Uploading artifacts for successful job                                              00:01
 Job succeeded
```

And for **docker-build** stage



```
$ /kaniko/executor --context $CI_PROJECT_DIR/qrcode --dockerfile $CI_PROJECT_DIR/qrcode/Dockerfile --destination ${CI_REGISTRY_IMAGE}:${CI_REGISTRY_TAG_DEV}
E0601 18:57:39.125166      17 aws_credentials.go:77] while getting AWS credentials NoCredentialProviders: no valid providers in chain. Deprecated.
	For verbose messaging see aws.Config.CredentialsChainVerboseErrors
INFO[0041] Retrieving image manifest java:8-jdk-alpine
INFO[0044] Retrieving image manifest java:8-jdk-alpine
INFO[0047] Built cross stage deps: map[]
INFO[0047] Retrieving image manifest java:8-jdk-alpine
INFO[0047] Retrieving image manifest java:8-jdk-alpine
INFO[0050] Executing 0 build triggers
INFO[0050] Unpacking rootfs as cmd COPY ./target/qrcode-0.0.1-SNAPSHOT.jar /usr/app/ requires it.
INFO[0077] COPY ./target/qrcode-0.0.1-SNAPSHOT.jar /usr/app/
INFO[0079] Resolving 1 paths
INFO[0079] Taking snapshot of files...
INFO[0084] WORKDIR /usr/app
INFO[0084] cmd: workdir
INFO[0084] Changed working directory to /usr/app
INFO[0084] RUN sh -c 'touch qrcode-0.0.1-SNAPSHOT.jar'
INFO[0084] Taking snapshot of full filesystem...
INFO[0084] Resolving 1672 paths
INFO[0088] cmd: /bin/sh
INFO[0088] args: [-c sh -c 'touch qrcode-0.0.1-SNAPSHOT.jar']
INFO[0088] Running: [/bin/sh -c sh -c 'touch qrcode-0.0.1-SNAPSHOT.jar']
INFO[0088] Taking snapshot of full filesystem...
INFO[0088] Resolving 1672 paths
INFO[0090] ENTRYPOINT ["java" , "-jar" , "qrcode-0.0.1-SNAPSHOT.jar"]
$ echo "CI/CD job completed, APP image should be available at:"
CI/CD job completed, APP image should be available at:
$ echo ${CI_REGISTRY_IMAGE}:${CI_REGISTRY_TAG_DEV}
harbor.corp.local/daniel_project1/dockersample:cicd-v3
Running after_script
Saving cache
Uploading artifacts for successful job
 Job succeeded
```

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



daniel_project1/qrcode

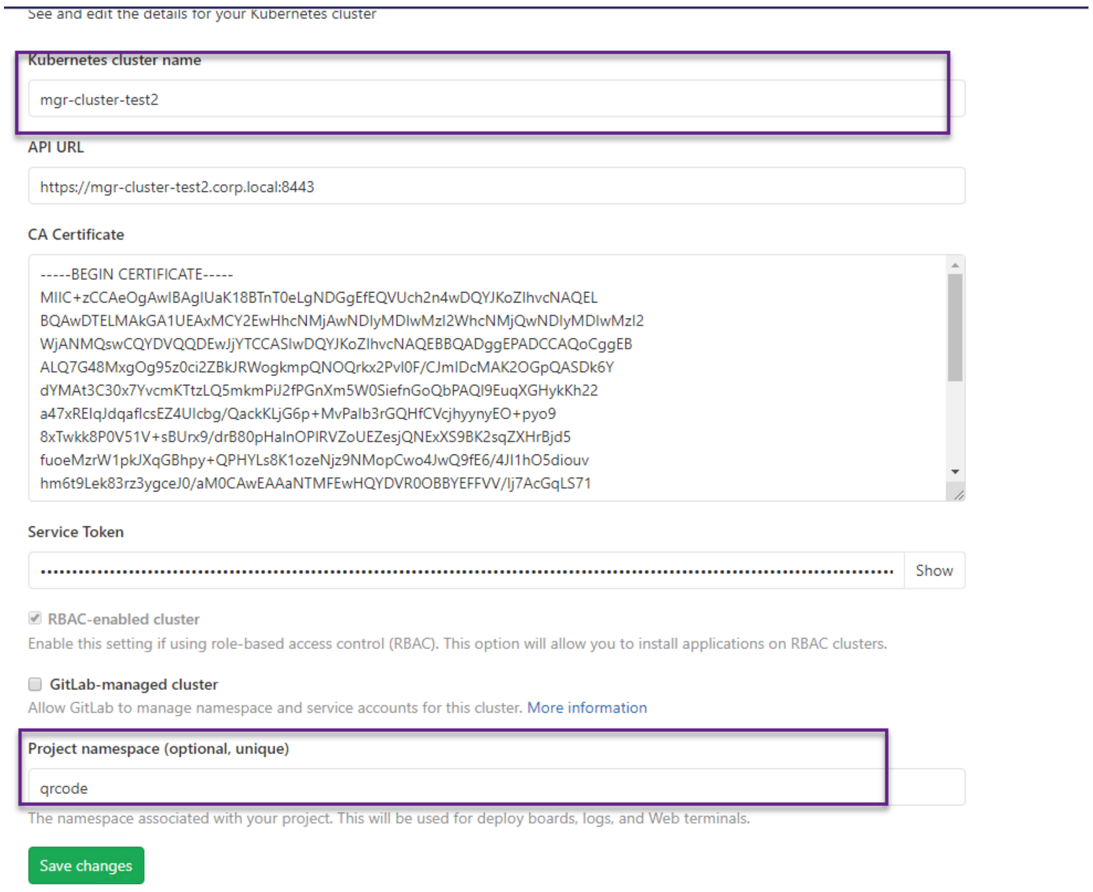
Tag	Size	Pull Command	Vulnerabilities	Signed	Author	Creation Time	Docker Version	Labels
cicd-v1	80.53MB		H • 11 Total • 11 Fixable	⊗		6/2/20, 2:48 PM	1.12.6	
cicd-v2	80.53MB		H • 11 Total • 11 Fixable	⊗		6/2/20, 3:42 PM	1.12.6	

- Notes:
  - This part of pipeline essentially implements its "CI" use case by automating source code and docker container build and upload to a centralized image repository for further use in CD processes
  - Other 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 or run additional pipelines defined via **.gitlab-ci.yaml** as in example above.
  - If developers need to create their own customized pipelines, please see GitLab documentation: https://gitlab.acelab.local/help/ci/pipelines/settings#custom-ci-configuration-path

6. *Configure and Execute CI/CD Pipeline for Deployment of Application from Registry to K8s Cluster*

In this section we will continue building our end-to-end CI/CD pipeline stage that implements automated deployment of container images from Harbor registry (selected by tags) into TKGI K8s cluster integrated with our project, thus completing "Continuous Deployment" part of CI/CD.

- Similar to how it was done in Section 3 (for deployment of GitLab Runner into the **daniel-lab2-small1-sharedt1** K8s cluster), we will integrate a K8s cluster **mgr-cluster-test2** for deployment of an application from our GitLab project. Below are properties of an integrated K8s cluster for our project, **dockersample**:



See and edit the details for your Kubernetes cluster

Kubernetes cluster name

mgr-cluster-test2

API URL

https://mgr-cluster-test2.corp.local:8443

CA Certificate

```
-----BEGIN CERTIFICATE-----
MIIC+zCCAeOgAwIBAgIUaK18BTnT0eLgNDGgEfEQVUch2n4wDQYJKoZIhvcNAQEL
BQAwDTELMAkGA1UEAxMCY2EwHhcNMjAwNDIyMDIwMzI2WhcNMjQwNDIyMDIwMzI2
WjANMQswCQYDVQQDEwJjYTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
ALQ7G48MxgOg95z0ci2ZBkJRWogkmpQNOQrkx2Pvl0F/CJmIDcMAK2OGpQASDk6Y
dYMAt3C30x7YvcmKTtzLQ5mkmPiJ2fPGnXm5W0SiefnGoQbPAQI9EuqXGHykKh22
a47xREIqJdqaflcsEZ4UIcbg/QackKLjG6p+MvPalb3rGQHfCVcjhyynyEO+pyo9
8xTwkk8P0V51V+sBUrx9/drB80pHalnOPIRVZoUEZesjQNExXS9BK2sqZXHrBjd5
fuoeMzrW1pkJXqGBhpy+QPHYLs8K1ozeNjz9NMopCwo4JwQ9fE6/4JI1hO5diouv
hm6t9Lek83rz3ygceJ0/aM0CAwEAAaNTMFEwHQYDVR0OBBYEFFVV/lj7AcGqLS71
```

Service Token

••• Show

☑ RBAC-enabled cluster

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

☐ GitLab-managed cluster

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

Project namespace (optional, unique)

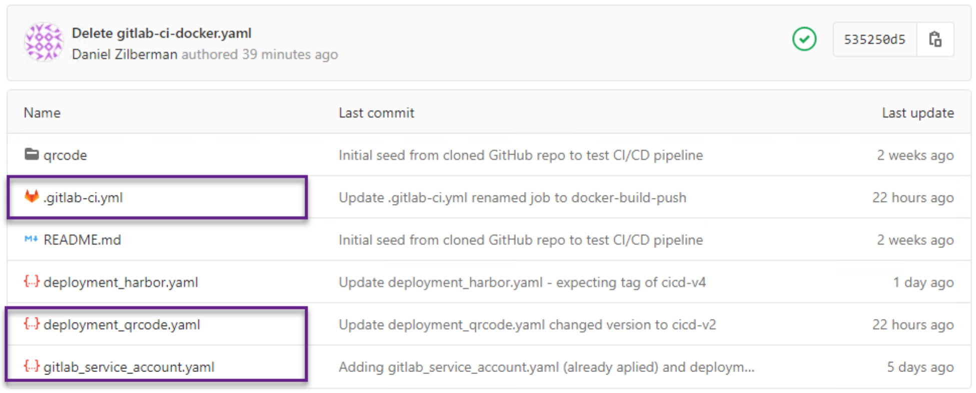
qrcode

The namespace associated with your project. This will be used for deploy boards, logs, and Web terminals.

Save changes

NOTE: Optional Project namespace property will map into K8s namespace that would be set as a target namespace for applications deployment. In our example, that namespace is “qrcode” and it will need to exist in the K8s cluster when application will be deployed by CI/CD pipeline

- We will add 2 Kubernetes related files to our GitLab project repository:
  - Deployment descriptor for GitLab Service Account used to create service account for integration with K8s cluster above (gitlab-service-account.yaml)
  - Deployment descriptor for deployment of application containers into K8s cluster (deployment_qrcode.yaml)



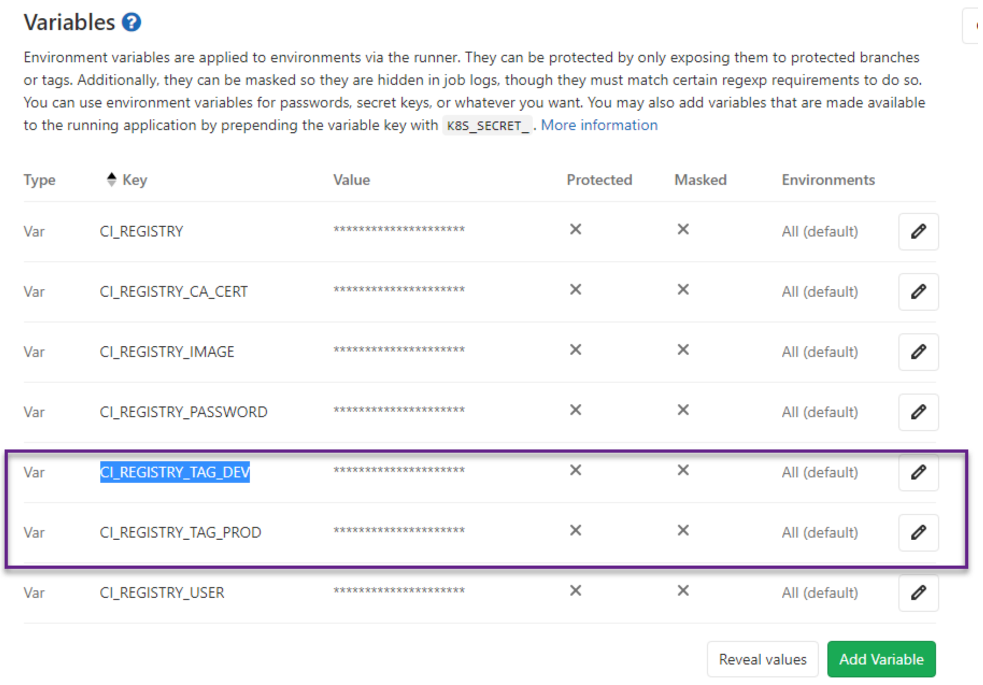
Delete gitlab-ci-docker.yaml
Daniel Zilberman authored 39 minutes ago
535250d5

Name	Last commit	Last update
qrcode	Initial seed from cloned GitHub repo to test CI/CD pipeline	2 weeks ago
.gitlab-ci.yml	Update .gitlab-ci.yml renamed job to docker-build-push	22 hours ago
README.md	Initial seed from cloned GitHub repo to test CI/CD pipeline	2 weeks ago
deployment_harbor.yaml	Update deployment_harbor.yaml - expecting tag of cicd-v4	1 day ago
deployment_qrcode.yaml	Update deployment_qrcode.yaml changed version to cicd-v2	22 hours ago
gitlab_service_account.yaml	Adding gitlab_service_account.yaml (already aplied) and deploym...	5 days ago

- Below is content of **deployment_qrcode.yaml** file – a pretty standard K8s Deployment descriptor that specifies image name, labels and number of replicas to be created in ReplicaSet as well as K8s secret with Harbor login credentials that would need to exist in the namespace where application will be deployed:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: qrcode-java
  #namespace: hello-world
  labels:
    app: qrcode
spec:
  replicas: 2
  selector:
    matchLabels:
      app: qrcode
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 33%
  template:
    metadata:
      labels:
        app: qrcode
    spec:
      imagePullSecrets:
        - name: regcred
      containers:
        - name: qrcode
          #NOTE: cicd-v3 tag is for deploymet branch, will
have to use another one for production
          image:
harbor.corp.local/daniel_project1/qrcode:cicd-v3
          ports:
            - containerPort: 8080
```

NOTE: in this version, an image tagged as **harbor.corp.local/daniel_project1/qrcode:cicd-v3** is expected to be present in the Harbor repository at the time of deployment to K8s (automated in the previous stages of CI/CD pipeline). That tag value is contained in the CA_REGISTRY_TAG_DEV variable below



Variables

Environment variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. Additionally, they can be masked so they are hidden in job logs, though they must match certain regexp requirements to do so. You can use environment variables for passwords, secret keys, or whatever you want. You may also add variables that are made available to the running application by prepending the variable key with `K8S_SECRET_`. More information

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_DEV	********************	✕	✕	All (default)
Var	CI_REGISTRY_TAG_PROD	********************	✕	✕	All (default)
Var	CI_REGISTRY_USER	********************	✕	✕	All (default)

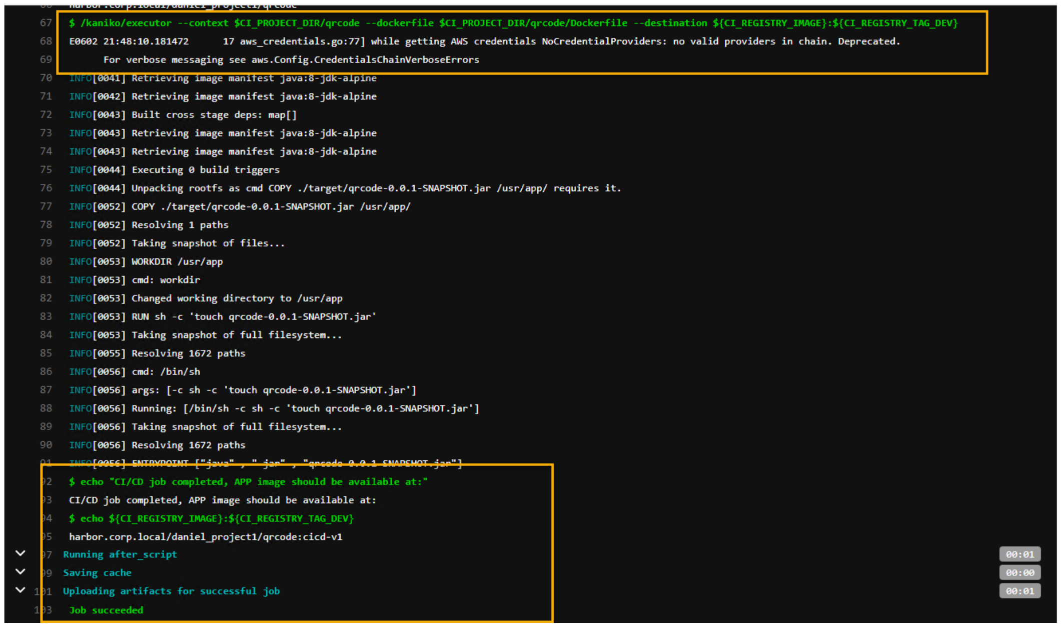
- Now we need to implement **deploy** stage of our CI/CD pipeline script (same .gitlab-ci.yaml file in our example) that will take care of establishing connection to target K8s cluster and performing all necessary steps to deploy our application using deployment descriptor above. For now, we only want to deploy the application only when pipeline on **master** branch is run hence the 'only …' condition

```
…..
#deploy stage
deploy to development:
  stage: deploy
  image:  "registry.gitlab.com/gitlab-org/cluster-
integration/auto-deploy-image:v0.15.0"
  script:
    - echo "Runner in Environment:"
    - echo $CI_ENVIRONMENT_SLUG
    - echo "WILL USE BUILT IN K8s VARS PER ENVIRONENT:"
    - echo "KUBE_URL:"
    - echo $KUBE_URL
    - echo "KUBE_NAMESPACE:"
    - echo $KUBE_NAMESPACE
    - echo "Path to kubeconfig:"
    - echo $KUBECONFIG
    - echo "=================================="
    - echo "Trying to 'get nodes' using default
kubeconfig setting..."
    # will use syntax like kubectl config --
kubeconfig=config-demo set-cluster
    - kubectl --kubeconfig="${KUBECONFIG}" get nodes
    - echo "=================================="
    - echo "CHECK K8s current context set to:"
    - kubectl config current-context
    - echo "Getting Nodes info w/o using context:"
    - kubectl get nodes
    #- echo "-----------------------------------------
-----"
    # TODO: add error handling in case if NS exists or
just delete proactively
    - kubectl delete ns $KUBE_NAMESPACE
    - sleep 60
    # now create it again
    - kubectl create ns $KUBE_NAMESPACE
    - echo "Creating regcred Docker login secret
'regcred'.."
    - kubectl create secret docker-registry regcred --
docker-server=$CI_REGISTRY --docker-
username=$CI_REGISTRY_USER --docker-
password=$CI_REGISTRY_PASSWORD --docker-
email=admin_harbor@acelcab.local
    - kubectl get secrets
    - echo "Will deploy QR Code container app to target
cluster/namespace:"
    - kubectl apply -f
$CI_PROJECT_DIR/deployment_qrcode.yaml
    - echo "QR Code container App deployed to K8s
cluster, checking Pod status:"

    - kubectl get po -n $KUBE_NAMESPACE
  environment:
    name: development
  #deploy app to K8s only when a pipeline on master
branch is run
  only:
  - master
```

NOTES:

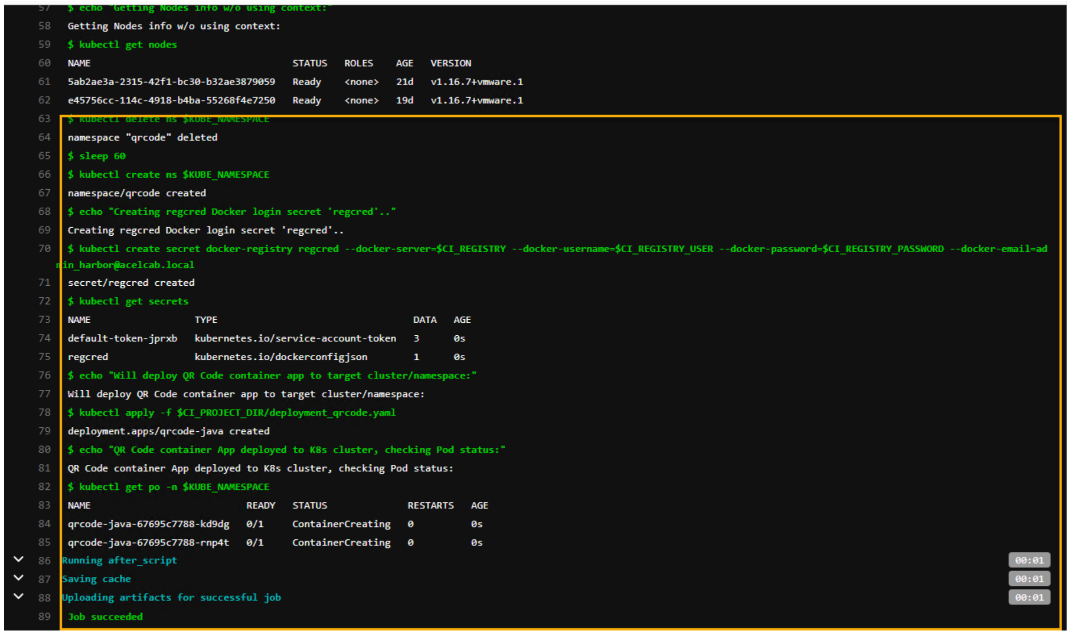
- We are using a different container image to run jobs for **deploy** stage, "registry.gitlab.com/gitlab-org/cluster-integration/auto-deploy-image:v0.15.0" that has kubectl CLI utility pre-installed
- $KUBE_URL, $KUBE_TOKEN, $KUBE_NAMESPACE and $KUBECONFIG are K8s environment variables that become available to CI/CD scripts once an **environment** is defined, as in the last 2 lines above (see details in documentation: https://docs.gitlab.com/ee/user/project/clusters/#deploying-to-a-kubernetes-cluster)
- $KUBECONFIG variable contains path to K8s configuration file placed on the 'builder' image that can be used by 'kubectl' CLI utility - pretty much the only parameter needed for access to target cluster
- Namespace contained in the $KUBE_NAMESPACE needs to exist in the target K8s cluster, so it is first deleted by `kubectl delete ns $KUBE_NAMESPACE` to make sure there is no error and then getting created by `kubectl create ns $KUBE_NAMESPACE` command
- A secret containing Harbor registry credentials for deployment of an image needs to exist in that namespace and gets created by `kubectl create secret docker-registry regcred …` command.
- Actual application deployment is done by running `kubectl apply -f $CI_PROJECT_DIR/deployment_qrcode.yaml` command that is using deployment descriptor above
- As mentioned above, current version of pipeline deploys the application into **development** environment only when pipeline on **master** branch is run
- Multiple '- echo ' operators are placed into the script for debugging purposes and can be removed

- When our project CI/CD pipeline is executed end-to-end, it first runs a unit test stage (not implemented, has a placeholder), then rebuilds a Docker image from the source code and instructions contained in the Dockerfile, tags it with value $CI_REGISTRY_TAG_DEV and pushes it into corresponding repository in Harbor.
- An output of **docker build push** job section is shown below:



```
harbor.corp.local/daniel_project1/qrcode
$ /kaniko/executor --context $CI_PROJECT_DIR/qrcode --dockerfile $CI_PROJECT_DIR/qrcode/Dockerfile --destination ${CI_REGISTRY_IMAGE}:${CI_REGISTRY_TAG_DEV}
E0602 21:48:10.181472      17 aws_credentials.go:77] while getting AWS credentials NoCredentialProviders: no valid providers in chain. Deprecated.
        For verbose messaging see aws.Config.CredentialsChainVerboseErrors
INFO[0041] Retrieving image manifest java:8-jdk-alpine
INFO[0042] Retrieving image manifest java:8-jdk-alpine
INFO[0043] Built cross stage deps: map[]
INFO[0043] Retrieving image manifest java:8-jdk-alpine
INFO[0043] Retrieving image manifest java:8-jdk-alpine
INFO[0044] Executing 0 build triggers
INFO[0044] Unpacking rootfs as cmd COPY ./target/qrcode-0.0.1-SNAPSHOT.jar /usr/app/ requires it.
INFO[0052] COPY ./target/qrcode-0.0.1-SNAPSHOT.jar /usr/app/
INFO[0052] Resolving 1 paths
INFO[0052] 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[0055] 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[0056] Resolving 1672 paths
INFO[0056] ENTRYPOINT ["java" , "-jar" , "qrcode-0.0.1-SNAPSHOT.jar"]
$ echo "CI/CD job completed, APP image should be available at:"
CI/CD job completed, APP image should be available at:
$ echo ${CI_REGISTRY_IMAGE}:${CI_REGISTRY_TAG_DEV}
harbor.corp.local/daniel_project1/qrcode:cicd-v1
Running after_script                                                                 00:01
Saving cache                                                                         00:00
Uploading artifacts for successful job                                               00:01
 Job succeeded
```

Script in the **deploy** job section performs deployment of tagged container image from Harbor registry according to deployment descriptor into designated namespace in the integrated K8s cluster:



```
$ echo "Getting Nodes info w/o using context:"
Getting Nodes info w/o using context:
$ kubectl get nodes
NAME                                   STATUS   ROLES    AGE   VERSION
5ab2ae3a-2315-42f1-bc30-b32ae3879059   Ready    <none>   21d   v1.16.7+vmware.1
e45756cc-114c-4918-b4ba-55268f4e7250   Ready    <none>   19d   v1.16.7+vmware.1
$ kubectl delete ns $KUBE_NAMESPACE
namespace "qrcode" deleted
$ sleep 60
$ kubectl create ns $KUBE_NAMESPACE
namespace/qrcode created
$ echo "Creating regcred Docker login secret 'regcred'.."
Creating regcred Docker login secret 'regcred'..
$ kubectl create secret docker-registry regcred --docker-server=$CI_REGISTRY --docker-username=$CI_REGISTRY_USER --docker-password=$CI_REGISTRY_PASSWORD --docker-email=ad
min_harbor@acelcab.local
secret/regcred created
$ kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-jprxb   kubernetes.io/service-account-token   3      0s
regcred               kubernetes.io/dockerconfigjson        1      0s
$ echo "Will deploy QR Code container app to target cluster/namespace:"
Will deploy QR Code container app to target cluster/namespace:
$ kubectl apply -f $CI_PROJECT_DIR/deployment_qrcode.yaml
deployment.apps/qrcode-java created
$ echo "QR Code container App deployed to K8s cluster, checking Pod status:"
QR Code container App deployed to K8s cluster, checking Pod status:
$ kubectl get po -n $KUBE_NAMESPACE
NAME                           READY   STATUS              RESTARTS   AGE
qrcode-java-67695c7788-kd9dg   0/1     ContainerCreating   0          0s
qrcode-java-67695c7788-rnp4t   0/1     ContainerCreating   0          0s
Running after_script                                                                 00:01
Saving cache                                                                         00:01
Uploading artifacts for successful job                                               00:01
 Job succeeded
```

Checking directly in the **mgr-cluster-test2** integrated K8s cluster, we can see 2 replicas of **qrcode-java** pod running, as expected:

```
kubectl get po -n qrcode
kubectl get po -n qrcode
NAME                                        READY    STATUS     RESTARTS
AGE
```

```
qrcode-java-7f77794b99-gfz4c   1/1     Running   0
46m
qrcode-java-7f77794b99-n8dph   1/1     Running   0
46m
```

and Pod details

```
kubectl describe po qrcode-java-7f77794b99-gfz4c -n qrcode
Name:         qrcode-java-7f77794b99-gfz4c
Namespace:    qrcode
Priority:     0
Node:         e45756cc-114c-4918-b4ba-55268f4e7250/172.15.2.4
Start Time:   Wed, 03 Jun 2020 20:52:22 +0000
Labels:       app=qrcode
              pod-template-hash=7f77794b99
Annotations:  <none>
Status:       Running
IP:           172.16.26.3
IPs:
  IP:           172.16.26.3
Controlled By:  ReplicaSet/qrcode-java-7f77794b99
Containers:
  qrcode:
    Container ID:
docker://04a88bb3c480e45d6c9d0ac60184b09c2e345b637260c707c6ea2
ed4db4ed04d
    Image:
harbor.corp.local/daniel_project1/qrcode:cicd-v2
    Image ID:       docker-
pullable://harbor.corp.local/daniel_project1/qrcode@sha256:be8
668ca7dcf8e4f53fa527152548592e8d7b12d8b25206498c98ad9cad15aeb
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Wed, 03 Jun 2020 20:52:30 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from
default-token-lrjln (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-lrjln:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-lrjln
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for
300s
                 node.kubernetes.io/unreachable:NoExecute for
300s
```

Finally, using kube-proxy port forwarding from one of the pods, we can post a request and get a response (binary):

```
kubectl port-forward qrcode-java-7f77794b99-gfz4c  8090:8080 -n
qrcode
..

curl -d '{"merchantID":"123", "merchantName":"daniel"}' -H
"Content-Type: application/json" -X POST
http://localhost:8090/generateQRCode --output QRCode.out
  % Total    % Received % Xferd  Average Speed   Time    Time
Time  Current
                                 Dload  Upload   Total   Spent
Left  Speed
100   312  100   267  100    45   2321    391 --:--:-- --:--:-- -
-:--:--  2713
```

NOTES:

- In addition to **development** environment, other environments (e.g. **staging**, **production etc.**) can be defined in the CI/CD script sections to direct deployments and contain different instructions/parameters (such as Docker image tags). That's the purpose of having CI_REGISTRY_TAG_PROD variable. Please see documentation for using environments in CI/CD pipelines: https://docs.gitlab.com/ee/ci/pipelines/index.html
- Actual unit testing script can be defined in the **test** stage.
- As in our example, GitLab Runner containers that run scripts defined in pipelines and deployed Applications may run in different K8s clusters (in fact, that may be preferred). Those clusters just need to have networking/DNS access to each other as **daniel-lab2-small1-sharedt1** and **mgr-cluster-test2** do.

7. Conclusion

We hope this document was useful. As you try these configuration steps, please provide any feedback or questions in the comments section for this document on code.vmware.com. Also, please let us know if you have any suggestions or if you would like to see guidance on other topics.