How to create blue-green deployments using Kubernetes.

Blue-Green Deployment:

1. First we create one deployment and service.
2. Now we need to update the deployment. First we create new deployment without service.
3. If the new deployment works fine. Now we just map the service used in first deployment to the second deployment. This process is blue-green deployment.

Step 1:

First we create EKS cluster.

git clone https://github.com/ksnithya/blue-green.git

cd blue-green

terraform init

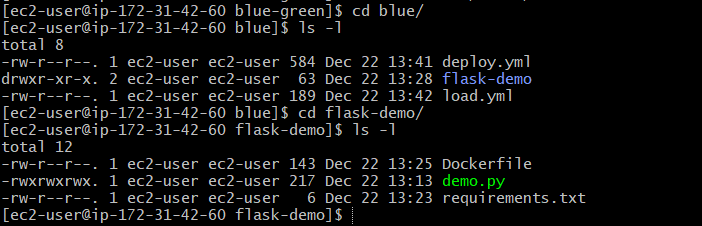
terraform plan

terraform apply

aws eks --region ap-south-1 update-kubeconfig --name eks\_cluster\_demo

Step 2:

We have sample python-flask app deployment. We create docker image and push it to dockerhub. Code available in same repo under “blue” directory.



Step 3:

Now we create the docker image and push to dockerhub.

A screen shot of a computer

Description automatically generated

Step 4:

Now we create first deployment. Code is available in same repo under “blue” directory.

Kubectl create -f .

A computer screen shot of a black screen

Description automatically generated

We can see the deployment and service created.

A screen shot of a computer

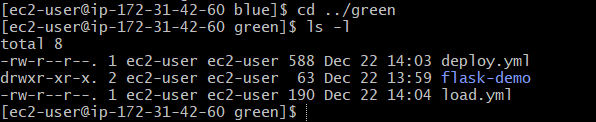
Description automatically generated

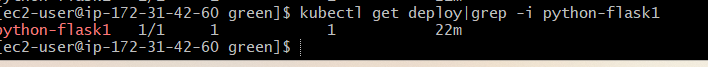
A screenshot of a computer

Description automatically generated

Step 5:

Now we create the new deployment same as above. Code is available under “Green” directory. We create docker image and push to docker hub as above. First we create deployment alone first and see if it is working fine.





Now our second deployment is created fine.

Step 6:

Now we map the already exist service to point to our new app and recreate the service

A black background with a black border

Description automatically generated with medium confidence

Kubectl replace -f load.yml

A computer screen with text on it

Description automatically generated

Now our service will point to new app.

A screenshot of a computer

Description automatically generated

How to setup blue-green deployment using Gitlab CI.

1. We need to setup docker executor.
2. Need to setup Kubernetes cluster connector
3. We can create job for deployment.
4. How to setup docker executor.

Step 1:

First we need to setup one server to run our gitlab runner and docker executor.

We can create on EC2 min t2.medium instance.

Then we install git and docker.

Then install docker, aws cli, git

Follow steps in below link to install docker in Amazon Linux.

<https://www.cyberciti.biz/faq/how-to-install-docker-on-amazon-linux-2/>

To install git:

Sudo yum install git -y

To install aws cli:

Step 2:

Login to gitlab.com -> go the repo where our code exist -> Settings -> CI/CD -> Runner->Expand

A screenshot of a computer

Description automatically generated

Then click on “New Project runner”

A screenshot of a computer

Description automatically generated

Then select OS of your server, enable “run untagged job”

A screenshot of a computer

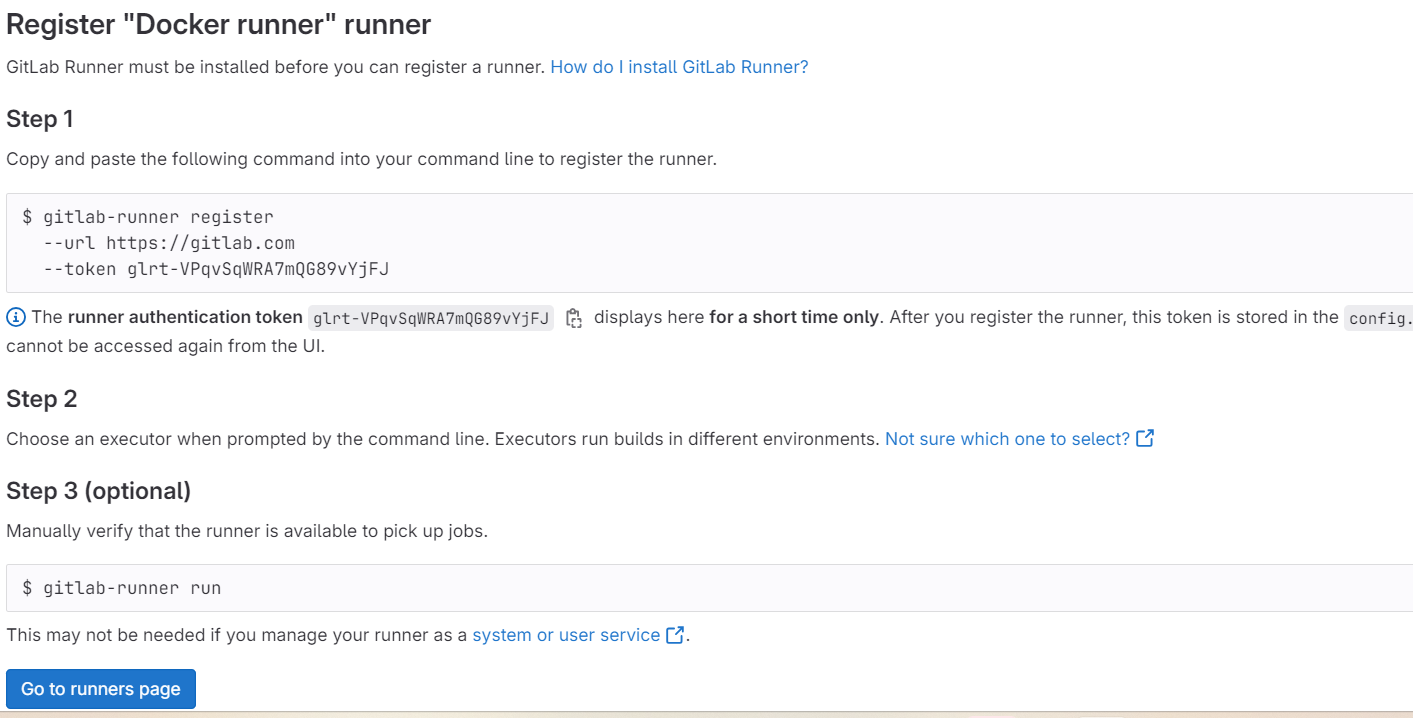
Description automatically generated

Then click on “create runner”.

A screenshot of a computer

Description automatically generated

Now we need to register runner using below steps.



We need to install gitlab runner in our server to run gitlab-runner command.

# Download the binary for your system

sudo curl -L --output /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64

# Give it permission to execute

sudo chmod +x /usr/local/bin/gitlab-runner

# Create a GitLab Runner user

sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash

# Install and run as a service

sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner

sudo gitlab-runner start

[ec2-user@ip-172-31-42-60 ~]$ **gitlab-runner register --url https://gitlab.com --token glrt-VPqvSqWRA7mQG89vYjFJ**

Runtime platform arch=amd64 os=linux pid=6503 revision=102c81ba version=16.7.0

WARNING: Running in user-mode.

WARNING: The user-mode requires you to manually start builds processing:

WARNING: $ gitlab-runner run

WARNING: Use sudo for system-mode:

WARNING: $ sudo gitlab-runner...

Created missing unique system ID system\_id=s\_d33763b02e5f

Enter the GitLab instance URL (for example, https://gitlab.com/):

[https://gitlab.com]: https://gitlab.com

Verifying runner... is valid runner=VPqvSqWRA

Enter a name for the runner. This is stored only in the local config.toml file:

[ip-172-31-42-60.ap-south-1.compute.internal]: docker-runner

Enter an executor: docker, shell, ssh, virtualbox, docker-autoscaler, docker+machine, custom, docker-windows, parallels, instance, kubernetes:

Docker **– Need to select what executor we are going to use.**

Enter the default Docker image (for example, ruby:2.7):

Alpine **– We can give any image name. Default is rubby.**

Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!

Configuration (with the authentication token) was saved in "/home/ec2-user/.gitlab-runner/config.toml"

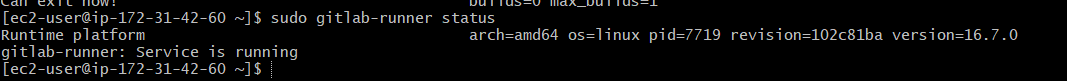
[ec2-user@ip-172-31-42-60 ~]$

A computer screen with text on it

Description automatically generated

We can check the status of gitlab-runner.

Sudo gitlab-runner status



We can also see status in gitlab.com

Project -> setting -> CI/CD -> Runner. We can see all available runner.

A screenshot of a computer

Description automatically generated

[ec2-user@ip-172-31-9-225 ~]$ id gitlab-runner

uid=1001(gitlab-runner) gid=1001(gitlab-runner) groups=1001(gitlab-runner)

[ec2-user@ip-172-31-9-225 ~]$ sudo usermod -a -G docker gitlab-runner

[ec2-user@ip-172-31-9-225 ~]$ newgrp docker

[ec2-user@ip-172-31-9-225 ~]$

1. Setup kubernetes cluster setup.

First we create one project for k8s connection.

New Project -> create project - > give project name, select public -> click on create project.

A screenshot of a computer

Description automatically generated

Step 2.1:

Now we setup connection between K8s cluster and gitlab.

To connect a Kubernetes cluster to GitLab, you must first [install an agent in your cluster](https://docs.gitlab.com/ee/user/clusters/agent/install/index.html).

## Installation steps

To install the agent in your cluster:

1. [Create an agent configuration file](https://docs.gitlab.com/ee/user/clusters/agent/install/index.html#create-an-agent-configuration-file).
2. [Register the agent with GitLab](https://docs.gitlab.com/ee/user/clusters/agent/install/index.html#register-the-agent-with-gitlab).
3. [Install the agent in your cluster](https://docs.gitlab.com/ee/user/clusters/agent/install/index.html#install-the-agent-in-the-cluster).
4. Create an agent config file.

Go inside k8s-connection project we have created in above step and create a “.gitlab/agents/<agent-name>/config.yaml” file.

A screenshot of a computer

Description automatically generated

Give the file name and click on create. You can leave the file blank for now, and [configure it](https://docs.gitlab.com/ee/user/clusters/agent/install/index.html#configure-your-agent) later.

“.gitlab/agents/k8s-connections/config.yaml

A yellow and black label

Description automatically generated

1. Now we can register agent with gitlab.

You must register an agent before you can install the agent in your cluster. To register an agent:

1. On the left sidebar, select **Search or go to** and find your project. If you have an [agent configuration file](https://docs.gitlab.com/ee/user/clusters/agent/install/index.html#create-an-agent-configuration-file), it must be in this project. Your cluster manifest files should also be in this project.
2. Select **Operate > Kubernetes clusters**.
3. Select **Connect a cluster (agent)**.
   * If you want to create a configuration with CI/CD defaults, type a name.
   * If you already have an [agent configuration file](https://docs.gitlab.com/ee/user/clusters/agent/install/index.html#create-an-agent-configuration-file), select it from the list.
4. Select **Register an agent**.
5. GitLab generates an access token for the agent. You need this token to install the agent in your cluster.
6. Copy the command under **Recommended installation method**. You need it when you use the one-liner installation method to install the agent in your cluster.

Go to operator -> Kubernetes cluster -> connect a cluster

A screenshot of a computer

Description automatically generated

From dropdown select the repo we have created and click on register.

A screenshot of a computer

Description automatically generated

We need to follow the below steps in our cluster server.

A screenshot of a computer

Description automatically generated

Login to our Kubernetes cluster server and install the connections.

A screen shot of a computer

Description automatically generated

After installing the connection we can see the connection is active.

A screenshot of a computer

Description automatically generated

1. Now we can create the blue-green deployment pipeline.

First we setup EKS cluster. Code is available in <https://gitlab.com/ksnithya/blue-green.git>

Login to our server where we want to setup eks cluster.

Then clone the repo.

Git clone <https://gitlab.com/ksnithya/blue-green.git>

Cd blue-green

Terraform init

Terraform plan

Terraform apply

aws eks --region ap-south-1 update-kubeconfig --name eks\_cluster\_demo

Now we can write pipeline for deployment.

variables:

KUBE\_CONTEXT: ksnithya/k8s-connections:k8s-connections

stages:

- build\_blue

- package\_blue

- build\_green

- package\_green

build\_blue\_image:

stage: build\_blue

image: docker:25.0.0-beta.2-cli

services:

- docker:25.0.0-beta.2-dind

variables:

DOCKER\_TLS\_CERTDIR: "/certs"

before\_script:

- docker login -u $CI\_REGISTRY\_USER -p $CI\_REGISTRY\_PASSWORD $CI\_REGISTRY

script:

- cd blue/flask-demo

- ls -ltr

- docker build -t $CI\_REGISTRY/ksnithya/blue-green/nithya-image-blue:latest .

- docker push $CI\_REGISTRY/ksnithya/blue-green/nithya-image-blue:latest

- echo "image build"

deploy\_blue\_image:

stage: package\_blue

image:

name: bitnami/kubectl:latest

entrypoint: ['']

script:

- kubectl config use-context $KUBE\_CONTEXT

- kubectl get pods

- kubectl get nodes -o wide

- echo "Deploying nithya-resume-blue app"

- ls $CI\_PROJECT\_DIR/blue

- kubectl apply -f $CI\_PROJECT\_DIR/blue/.

- kubectl get deploy

- kubectl get svc

build\_green\_image:

stage: build\_green

image: docker:25.0.0-beta.2-cli

services:

- docker:25.0.0-beta.2-dind

variables:

DOCKER\_TLS\_CERTDIR: "/certs"

before\_script:

- docker login -u $CI\_REGISTRY\_USER -p $CI\_REGISTRY\_PASSWORD $CI\_REGISTRY

script:

- cd blue/flask-demo

- ls -ltr

- docker build -t $CI\_REGISTRY/ksnithya/blue-green/nithya-image-green:latest .

- docker push $CI\_REGISTRY/ksnithya/blue-green/nithya-image-green:latest

- echo "image build"

deploy\_green\_image:

stage: package\_green

image:

name: bitnami/kubectl:latest

entrypoint: ['']

script:

- kubectl config use-context $KUBE\_CONTEXT

- kubectl get pods

- kubectl get nodes -o wide

- echo "Deploying nithya-resume-green app"

- ls $CI\_PROJECT\_DIR/green

- kubectl apply -f $CI\_PROJECT\_DIR/green/deploy.yml

- kubectl replace -f $CI\_PROJECT\_DIR/green/load.yml

- kubectl get deploy

- kubectl get svc