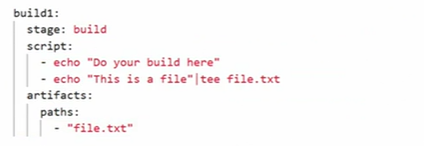
**GITLAB CICD**

Continuous Integration (CI)

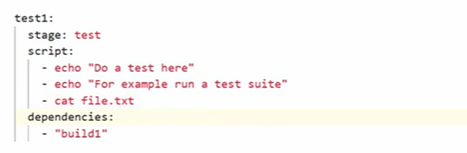
* Runners: Gitlab Runner is an application that works with Gitlab CI/CD to run jobs in pipeline

1. Shared Runners: its available to all groups and projects in a GitLab instance
2. Group Runners: Available to all projects and subgroups in a group
3. Specific runners: Associated with specific projects. Typically specific runners are used for one project at a time.

* Echo “This is a file “|tee file.txt : command for add contents to file.txt



* Above commands is to create an artifact from the file.txt. The ***artifacts*** keyword helps on this, We need to mention the source file for creating artifacts inside ***artifacts*** keyword.



* With help of ***dependencies,*** We can pass the build stage to testing stage and this helps to test our artifacts and file



* Method to set variables inside script



* Method to call variable inside our jobs.



* This is the output we can see inside our job details

Continuous Deployment (CD)

* Environments: The place where we deploy our Code .egg : Dev ,qa ,staging .Production.
* Environment path in Gitlab : Gitlab portal ->Deployments -> Environments

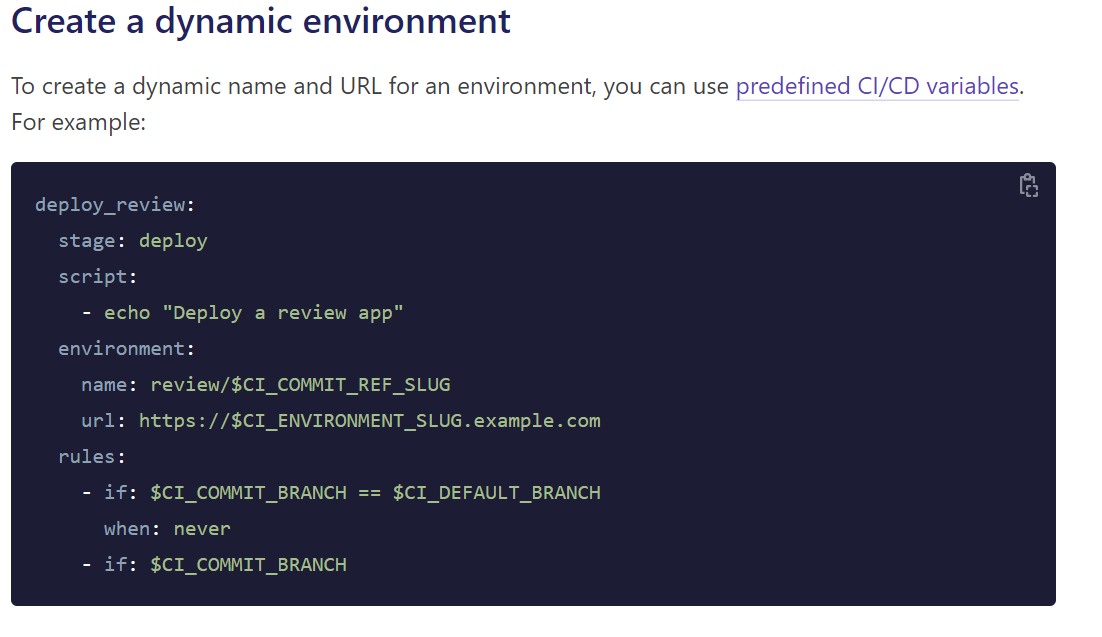
In your.gitlab-ci.yml file:

1. Specify a name for the environment and optionally, a URL, which determines the deployment URL. For example:



2.Trigger a deployment. (For example, by creating and pushing a commit.)

When the job runs, the environment and deployment are created.



* The name is review/$CI\_COMMIT\_REF\_SLUG. Because the [environment name](https://docs.gitlab.com/ee/ci/yaml/index.html#environmentname) can contain slashes (/), you can use this pattern to distinguish between dynamic and static environments.
* For the url, you could use $CI\_COMMIT\_REF\_SLUG, but because this value may contain a / or other characters that would not be valid in a domain name or URL, use $CI\_ENVIRONMENT\_SLUG instead. The $CI\_ENVIRONMENT\_SLUG variable is guaranteed to be unique.

……………………………………………………………………………………………………………………………………….

* Rules Keyword in the below example helps us to run deployment jobs manually



* Rules Keyword in the below example helps us to run deployment jobs automatically.



* In the staging job , it will run only if there is any tag added .otherwise it wont run .We use rules as below for this. $CI\_COMMIT\_TAG is Predefined variable in Gitlab

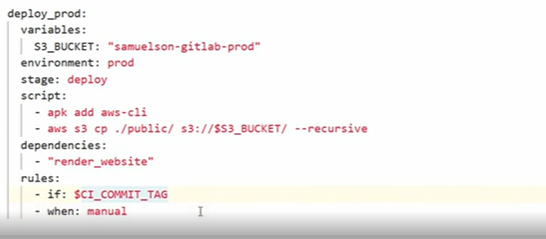


* In the deploy\_prod job, it will run only if there is any tag added .otherwise it wont run .But for this prod job we have added manual deployment .

We use rules as below for this. $CI\_COMMIT\_TAG is Predefined variable in Gitlab also use When: Manual in addition to $COMMIT\_TAG.

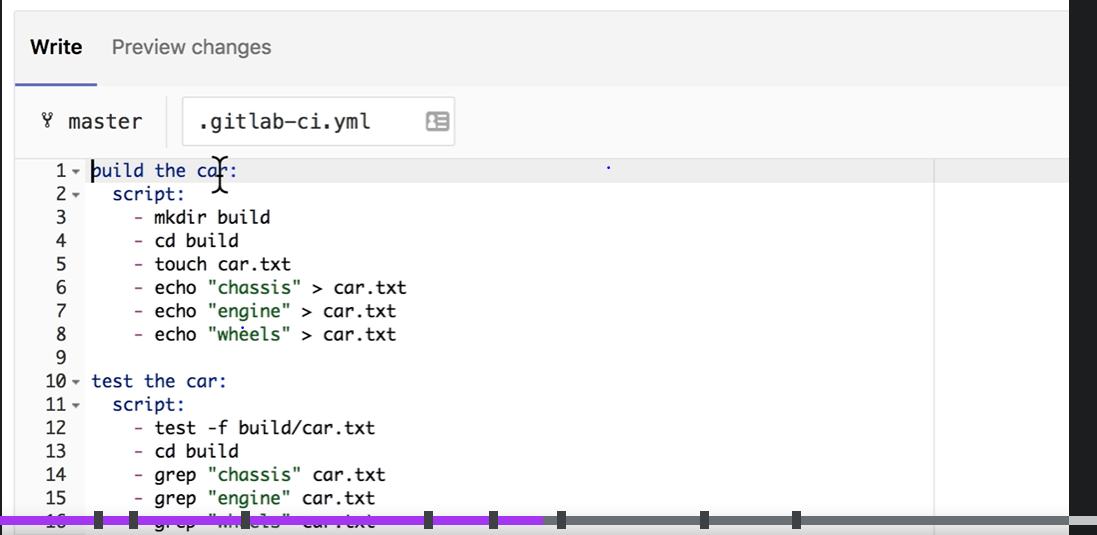
* If there is any bug found in after any deployment .We can roll back it to previous commit.

Path to roll back : gitlab portal ->Environments -> previous commit->Rollback



**Gitlab notes from UDEMY Training**

* A simple .gitlab-ci.yml file (artifacts query is not added, need to add it)



* Curl: Curl is a command line tool and library for transferring data with URL’s. it supports many protocols like http.
* Below is an example for CI/CD pipeline script

…………………………………………………………………………………………………………………………………………….

image: node

stages:

- build

- test

- deploy

build website:

stage: build

script:

- npm install

- npm install -g gatsby-cli

- gatsby build

artifacts:

paths:

- ./public

test artifact:

image: alpine

stage: test

script:

- grep -q "Gatsby" ./public/index.html

test website:

stage: test

script:

- npm install

- npm install -g gatsby-cli

- gatsby serve &

- sleep 3

- curl "http://localhost:9000" | tac | tac | grep -q "Gatsby"

deploy to surge:

stage: deploy

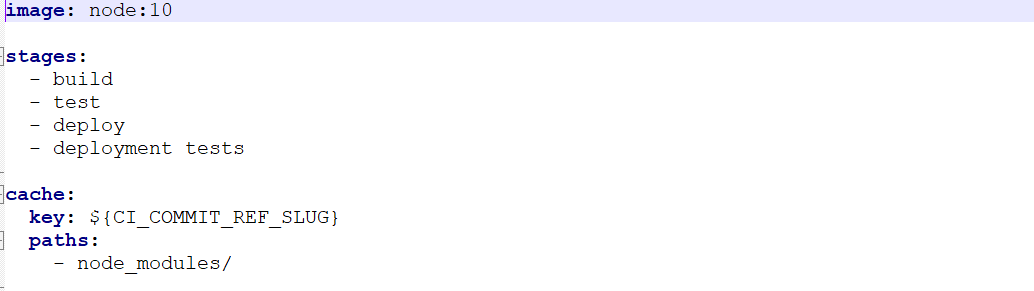
script:

- npm install --global surge

- surge --project ./public --domain instazone.surge.sh

……………………………………………………………………………………………………………………………………………

* Adding cache setup to the previous Script.



* Caching is a mechanism that can be used to save time when your jobs are running .Caching is about speeding the time a job is executed by reusing the same content of previous job

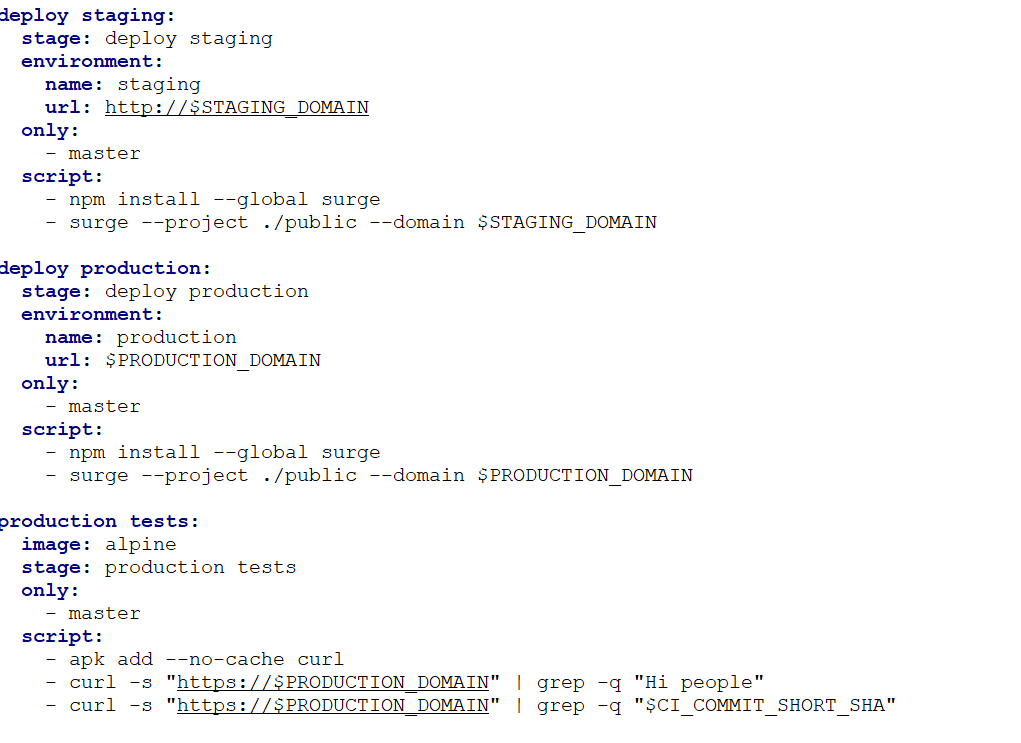
……………………………………………………………………………………………………………………………………………

Allow-failure:

Use allow\_failure to determine whether a pipeline should continue running when a job fails.

* To let the pipeline continue running subsequent jobs, use allow\_failure: true.
* To stop the pipeline from running subsequent jobs, use allow\_failure: false.





* Above is the example of using **only** command .It will help us to run our job in particular branch.

……………………………………………………………………………………………………………………………………………….

* Only:
  + Merge\_requests

The above example is help to run the job inly if there is any Merge request.

……………………………………………………………………………………………………………………………………………….

* before\_script and after\_script

before\_Script Used to defined commands/scripts that should run before the normal script block and it can be used locally or globally inside job.

* For disabling any jobs in our script just put a dot(.) Infront of job name.

Eg: .Deploy

* We can list all environment variables to a script with export command

Eg: job\_name:

Script:

-export