❖ install php & extensions:

- sudo apt update && sudo apt install php php-cli php-mbstring unzip php-curl php-xmlreader php8.2-mysql php-redis
- Note: I used Ubuntu VMs and php 8.2-Apache and latest versions of other packages to perform this task.

❖ Install Composer:

- sudo php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
- sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer

❖ Install Laravel:

- install the extension : sudo apt install php-xml
- add this lines in php.ini
  Extension=xmlreader.so
- sudo composer create-project --prefer-dist laravel/laravel saber
- cd saber #this will be working directory which contain the project and binaries

❖ Install & configure MySQL (easy and supported):

- sudo apt install mysql-server
- sudo mysql_secure_installation
- sudo mysql -u root –p #to create this :
  - ✓ CREATE DATABASE myweb;
  - ✓ CREATE USER 'test'@'localhost' IDENTIFIED BY 'Password';
  - ✓ GRANT ALL PRIVILEGES ON test.* TO 'test'@'localhost';
  - ✓ FLUSH PRIVILEGES;
  - ✓ EXIT;
- cp .env.example .env
- vim .env
  - ✓ DB_CONNECTION=myweb
  - ✓ DB_HOST=127.0.0.1
  - ✓ DB_PORT=3306
  - ✓ DB_DATABASE=myweb
  - ✓ DB_USERNAME=test
  - ✓ DB_PASSWORD=Password
  - ➢ Note, we must set the needed permissions to the root user and another user in mysql , also change the ownership of all files or ;Laravel and the Project to owned by current user
  - ➢ We need to create a table then insert data to retrieve it while test the DB and CACHE,:

```
mysql> use myweb;
mysql> CREATE TABLE student ( student_id INT AUTO_INCREMENT PRIMARY KEY,
first_name VARCHAR(50), last_name  VARCHAR(50) );
mysql> INSERT INTO student (first_name, last_name) VALUES ('ahmed', 'mohamed');
mysql> INSERT INTO student (first_name, last_name) VALUES ('maged', 'ali');
mysql> INSERT INTO student (first_name, last_name) VALUES ('hassan', 'hossam');
```

❖ migrate the db to the project to use it :

- php artisan migrate

❖ Install & configure Redis as a caching service for Mysql :

- sudo apt install redis-server #after add its repo and so on , then enable and start the server
- then add this line in .env file : CACHE_DRIVER=redis
- Generate Encryption Key: This command will generate a key and update your .env file with the new key. >>> php artisan key:generate
- Check .env File:
- We have to ensure that your .env file contains the necessary configuration, including database settings and the generated application key. It should look something like this:

  APP_KEY=theKeywillappearhere

- After making changes to the .env file, it's a good idea to clear the configuration cache using the following Artisan command:

  php artisan config:cache

- we have to ensure that the Laravel application has the necessary file permissions, especially for storage and bootstrap/cache directories. You can recursively set the correct permissions using: chmod -R 775 storage bootstrap/cache

❖ PHP Code “Backend”:

- Create a controller by this command: php artisan make:controller NewTestController
- Edit the controller which is app/Http/Controllers/NewTestController.php and add this :

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class NewTestController extends Controller
{
    /**
     * Display the welcome page.
     *
     * @return \Illuminate\View\View
     */
    public function redis()
    {
        return view('welcome', [
            'message' => 'This is a message about Redis.'
        ]);
    }
}
```

❖ Create a rout to use the controller:

- In the file , routes/web.php we will add the new rout:
  use App\Http\Controllers\NewTestController;

  Route::get('/', [NewTestController::class, 'redis']);

❖ Test Cache and Database connectivity:

- redis-cli ping # the output will be Pong , in the next file , there is a function to test caching , in the second load of the web page , it will be faster than the first time , in the first time it will query from the database direct.
  - we will update the view to check the application in this file :

```
<html>
  <head>
    <title>Using Redis Server with PHP and MySQL</title>
  </head>
  <body>
    <h1 align = 'center'>Students' Register</h1>

    <table align = 'center' border = '2'>

    <?php
        try {

            $data_source = '';

            $redis = new Redis();
            $redis->connect('127.0.0.1', 6379);

            $sql = 'select
                    student_id,
                    first_name,
                    last_name
                    from student
                    ';

            $cache_key = md5($sql);

            if ($redis->exists($cache_key)) {

                $data_source = "Data from Redis Server";
                $data = unserialize($redis->get($cache_key));

            } else {

                $data_source = 'Data from MySQL Database';

                $db_name     = 'testdb';
                $db_user     = 'testuser';
                $db_password = 'password';
                $db_host     = 'localhost';

                $pdo = new PDO('mysql:host=' . $db_host . '; dbname=' . $db_name, $db_user,
$db_password);
                $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

```
            $stmt = $pdo->prepare($sql);
            $stmt->execute();
            $data = [];

            while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
               $data[] = $row;
            }

            $redis->set($cache_key, serialize($data));
            $redis->expire($cache_key, 10);
        }

        echo "<tr><td colspan = '3' align = 'center'><h2>$data_source</h2></td></tr>";
        echo "<tr><th>Student Id</th><th>First Name</th><th>Last Name</th></tr>";

        foreach ($data as $record) {
           echo '<tr>';
           echo '<td>' . $record['student_id'] . '</td>';
           echo '<td>' . $record['first_name'] . '</td>';
           echo '<td>' . $record['last_name']  . '</td>';
           echo '</tr>';
        }


    } catch (PDOException $e) {
        echo 'Database error. ' . $e->getMessage();
    }
  ?>

</table>
  </body>
</html>
```

❖ final test:

- Starting and expose the service:
  php artisan serve --host=0.0.0.0 # after start the Laravel
- Open http://localhost:8000 in the browser # or replace localhost with Vm IP

❖ Create a Dockerfile for the application:

- After clone the first VM , which contain the project and setup the Docker on it , the first step is cd to the app dire then create Dockerfile as the following , each part is explained as a comment.

```
# Use the official PHP image with Apache
FROM php:8.2-apache
# Install necessary extensions
RUN docker-php-ext-install pdo_mysql mbstring xml
# Enable Apache modules
RUN a2enmod rewrite
# Set the working directory
WORKDIR /var/www/html
# Copy application files to the container
COPY . .
# Install Composer dependencies
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --
filename=composer
RUN composer install --no-dev
# Set directory permissions
RUN chown -R www-data:www-data /var/www/html/storage
/var/www/html/bootstrap/cache
# Expose port 80
EXPOSE 80
# Start Apache
CMD ["apache2-foreground"]
```

- Create a .dockerignore and add this lines :

```
.dockerignore
.git
node_modules
vendor
```

- Run : docker build -t my-laravel-app .

❖ Create a docker compose for redis and mysql & create a migration file:

- This is to define services, networks, and volumes for the application permanent:
- Vim docker-compose.yml in the root dir contain the following , the values must match the .env file :

```
version: '3'

services:
  web:
    image: my-laravel-app
    ports:
      - "8000:80"
    volumes:
      - .:/var/www/html
    depends_on:
      - db
      - redis

  db:
    image: mysql:8.0
    environment:
      MYSQL_DATABASE: myweb
      MYSQL_USER: test
      MYSQL_PASSWORD: Password
      MYSQL_ROOT_PASSWORD: root_password
    ports:
      - "3306:3306"

  redis:
    image: "redis:alpine"
    ports:
      - "6379:6379"
```

- Run :docker-compose up –d
- Run: docker-compose exec web php artisan migrate.

❖ Kubernetes needed configuration

- Create Kubernetes Deployment and Service files by create deployment.yaml and service.yaml files in the root directory of your project.
- Note: I'm using a preconfigured K8s cluster deployed on 2 VMs master node and worker node , to expose the pods of App externally , this needs nginx ingress , which is not applicable on the Vms and only we can use it easily on the cloud as a managed service.

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-laravel-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-laravel-app
  template:
    metadata:
      labels:
        app: my-laravel-app
    spec:
      containers:
      - name: registry.gitlab.com/username/my-laravel-app:latest
        image: my-laravel-app
        ports:
        - containerPort: 80
# service.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-laravel-app
spec:
  selector:
    app: my-laravel-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: LoadBalancer
```

- Then: kubectl apply -f deployment.yaml && kubectl apply -f service.yaml

❖ Packaging the application using Helm Chart

- Create Helm Chart by running the following commands:

```
helm create my-laravel-app-chart
```

- Configure Helm Values by modify the values.yaml file in the my-laravel-app-chart directory to match your application's settings.
- Install Helm Chart: helm install my-laravel-app-release my-laravel-app-chart , it will also generate the files as the following :

1. values.yaml: This file contains default configuration values for your Helm chart. You can customize these values to match your application's settings.

```
replicaCount: 1

image:
  repository: my-laravel-app
  tag: latest
  pullPolicy: IfNotPresent

service:
  name: my-laravel-app
  type: ClusterIP
  port: 80
```

2. templates/deployment.yaml: This file defines the Kubernetes Deployment for your application. It specifies the container image, environment variables, and other deployment-related settings.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "my-laravel-app-chart.fullname" . }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: {{ include "my-laravel-app-chart.name" . }}
      release: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app: {{ include "my-laravel-app-chart.name" . }}
        release: {{ .Release.Name }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
        ports:
        - containerPort: 80
```

3. templates/service.yaml: This file defines the Kubernetes Service for your application.

```
apiVersion: v1
kind: Service
metadata:
  name: {{ include "my-laravel-app-chart.fullname" . }}
spec:
  selector:
    app: {{ include "my-laravel-app-chart.name" . }}
    release: {{ .Release.Name }}
  ports:
  - protocol: TCP
    port: {{ .Values.service.port }}
    targetPort: 80
  type: {{ .Values.service.type }}
```

4. Chart.yaml: This file contains metadata about the Helm chart.

```
apiVersion: v2
name: my-laravel-app-chart
description: A Helm chart for deploying my Laravel app
version: 0.1.0
appVersion: 1.0.0
```

❖ GitLab CI Configuration

- Create .gitlab-ci.yml file in the root directory of your project.

```
stages:
  - build
  - deploy

variables:
  CONTAINER_IMAGE: registry.gitlab.com/username/my-laravel-app

build:
  stage: build
  script:
    - docker build -t $CONTAINER_IMAGE .
    - docker push $CONTAINER_IMAGE

deploy:
  stage: deploy
  script:
    - helm upgrade --install my-laravel-app-release my-laravel-app-chart
```

❖ Configure GitLab CI/CD:

- In your GitLab project, go to Settings > CI/CD.
- Expand the "Variables" section and add a variable named DOCKER_REGISTRY_PASSWORD with your GitLab Container Registry password.
- Notes :for triggering the changes when the developer push the new version of code to the branch we can use:
  - **Webhooks:** we can configure webhooks in the GitLab project to trigger a pipeline on events like code pushes, merge requests, etc.
  - **Scheduled Jobs:** this is another option; schedule pipeline runs at specific intervals using the GitLab CI/CD schedules.
- Note: here , I used the preinstalled gitlab server as a VM , but we can use it as a K8s pod