UNIT V – DEPLOYMENT AND CLOUD BASICS [9 hours]

How Software is Deployed in Real Life ,Cloud Platforms Overview (AWS/GCP/Azure), Hosting a Web App on Cloud (AWS EC2/Heroku), Docker Compose Basics, Introduction to Monitoring Tools (like Grafana)

HOSTING A WEB APP ON CLOUD HEROKU

Heroku is a platform-as-a-service (PaaS) for deploying, scaling, and managing applications. It supports various programming languages such as Ruby, Java, Node.js, Scala, Clojure, Python, PHP, and Go. Being cloud-based, Heroku removes the need for server and hardware management, enabling developers to concentrate on coding.

To deploy on Heroku, you upload your application's code to the Heroku platform. This can be done using Git, containerizing your app with Docker, or leveraging Heroku's integrations with third-party systems. After deployment, Heroku runs your application in a dyno, which is a lightweight, containerized Linux environment. An application can utilize multiple dynos to scale up for increased traffic or scale down when less capacity is required.

Before deploying your codebase to Heroku, ensure it adheres to a few conventions to ensure a successful deployment. Below are the steps to prepare your codebase:

1. Track your codebase in a Git repository

Heroku's deployment system uses Git, so your codebase must be committed to a Git repository. If you don't have Git installed, install it and set it up. If you are using another version control system, export your codebase to Git.

```
git init
git add .
git commit -m "My first commit"
```

2. Add a Heroku Git remote

Every Heroku application has its own Heroku-hosted Git repository. To deploy your app, push your code changes to this repository. Add the Heroku remote to your local Git repository:

```
heroku git:remote -a my-application
```

3. Add a Procfile

A Procfile is a text file in the root directory of your application that specifies the commands Heroku should run to start your app. Create a Procfile with the following content if you're using a Node.js application:

```
web: node app.js
```

For a Rails application, it might look like:

```
web: bundle exec rails server -p $PORT
```

worker: bundle exec rake jobs:work

4. Listen on the correct port

On Heroku, your application must listen on a port specified by the PORT environment variable. Modify your code to use this variable. For instance, in a Node.js app using Express:

```
let port = process.env.PORT || 8000;
app.listen(port, () => {
 console.log('Server is running on port ${port}');
});
```

5. Use a database or object storage instead of writing to the local filesystem

Heroku dynos have an ephemeral filesystem, so any data written to the local filesystem will be lost when the dyno cycles. Use a managed database service like Heroku Postgres or object storage like Amazon S3 for persistent storage.

1. Deploying to Heroku with Git

Deploying an application to Heroku using Git involves several steps, which include initializing a Git repository, creating a Heroku remote, and pushing your code to Heroku. Below is a detailed guide.

Prerequisites:

Ensure that both Git and the Heroku CLI are installed on your system.

Step 1: Create a Heroku remote Next, create a new Heroku application, which automatically sets up an empty Git repository hosted on Heroku. Use the heroku create command:

```
heroku create -a my-app
```

This command creates a new application named my-app and sets up the Heroku remote repository. You can verify the remote setup using:

```
git remote -v
```

The output should list the Heroku remote URLs for fetching and pushing:

```
heroku  https://git.heroku.com/my-app.git (fetch)
heroku  https://git.heroku.com/my-app.git (push)
```

Step 2: Deploy your code To deploy your code to Heroku, push your local repository's main branch to the Heroku remote:

```
git push heroku main
```

Heroku will receive the code, build the application, and deploy it to the specified app.

If you need to deploy code from a branch other than main, use the following syntax to push it to the main branch on Heroku:

```
git push heroku my-branch-name:main
```

This command tells Git to push the specified branch to the main branch on the Heroku remote.

2. Deploying Docker Images to Heroku

Deploying Docker images to Heroku involves several steps, leveraging Heroku's Container Registry. Here's a detailed guide to deploying your Docker images.

Prerequisites

Ensure that you have Docker installed and that you are logged into Heroku using the Heroku CLI.

1. **Log in to Heroku container registry:** First, log in to Heroku's container registry. This can be done using the Heroku CLI:
   heroku container:login
2. **Clone a sample application:** For demonstration purposes, you can clone a sample Python application based on Alpine Linux:
   git clone https://github.com/heroku/alpinehelloworld.git
   Navigate to the application directory:
   cd alpinehelloworld
3. **Create a Heroku application**
   Create a new Heroku application that uses the container stack:
   heroku create --stack container
   This command will create a new Heroku application and set up a Git repository for it. The output will include the app's URL and Git repository.
4. **Build and push the Docker image**
   Build the Docker image and push it to Heroku's container registry:
   heroku container:push web
5. **Release the Docker image**
   After pushing the image, release it to your Heroku application:
   heroku container:release web
6. **Open the application**
   Finally, open your application in the browser:
   heroku open
