

1.2. Continuous Integration and Continuous Deployment (CI/CD)

Continuous Integration (CI) is the practice of merging all the developers' working code to a single shared repository to build and test for automating the integration of code changes from multiple developers.



Continuous Deployment (CD) is the practice of automatically passing the integrated software features that passed the automated testing phase into the production environment, making changes that are visible to the users. CI/CD is one of the important practices for the **DevOps process**.

Study guide

- Create a local repository and a central repository, and connect them together.
- Practice the process of committing and publishing the committed code to the central repository.
- Practice code refactoring and continuous integration.
- Practice continuous deployment through automatic deployment to a real production environments, e.g., AWS, Heroku, etc.
- Understand the entire pipeline of the CI and CD process.

How can I save my project files so that I can continue working on my project anywhere or/and collaborating with my team members?

Create a project repository using Git and GitHub and connect it with Visual Studio Code

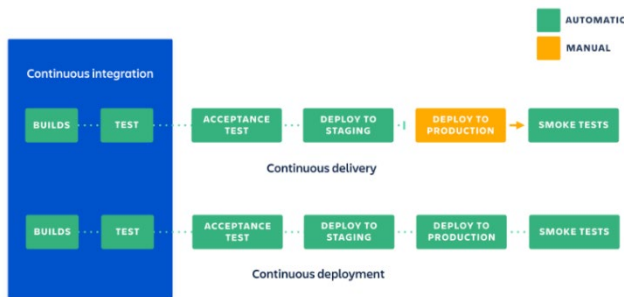
- **Create a project folder** for your workspace, e.g., 'myapp' and copy the project files, e.g., 'hello.js' to this directory.
- **Download Git** and install it. **Git is a version control system** that allows you to manage and keep track of your source codes.
- **Enable Git in VS Code:** File->Preferences->Settings, type 'Git: Enabled' in the search box, and make sure that the box for 'Git: Enabled' is checked.
- **Create a Git repository** (Git repo) for your project from  **'Source Control'** on the left menu bar of VS Code (or View->Source Control) by clicking 'Initialize Repository'. You can also initialize git by **Git commands** 'git init' in your project directory.
- **Verify** if the Git repository (.git folder) was created in your project folder 'myapp'. This repository tracks all changes made to files in your project. So if you delete the .git/ folder, then you delete your project's history.
- **Create an account at GitHub.com and create a repository for your project.** **GitHub is a cloud-based hosting service** that lets you manage Git repositories in a centralized location.
- **Connect** your local Git repository on your computer with GitHub in VS Code: View->Command Palette->Git: Add remote..., enter the remote repository URL (from GitHub, '<>Code' -> Code (green color) -> copy the URL). Or create a repository on GitHub and **clone** it in your local directory.
- **To verify the connection, create a simple HTML file**, e.g., 'hello.html', click  for Source Control, click + to stage changes, enter a commit message, click ✓ to commit, click **'Sync Changes'** (or **Publish Branch** if you have not published this branch on GitHub before). If connected properly, now you can push the source codes of your project in a local repository to the GitHub repository, allowing you to access it and collaborate with your team members anywhere.
- **Open the repository page on your GitHub account to verify** if the file was pushed correctly and click <>Code. If you find the file ('hello.html') in the GitHub repository, everything works fine. For a detailed demo, [watch this tutorial](#).

- **Make sure you add .gitignore file** to let **Git** ignore tracking the changes of certain files or folders. You can use a .gitignore extension or manually create it. For an **auto-generated .gitignore** file (View->Command Palette (Ctrl+Shift+P), type 'Add gitignore', choose JavaScript, add VS code extension, and edit it to add or remove ignore items as needed (e.g., add a line '.vscode/'). You can also **right-click a file** (in Source Control) to add it to .gitignore list.

Creating a project folder, a local project repository, and a repository on GitHub and connecting all these repositories and various servers related to the project (e.g., **development server**, **staging server**, and **production server**) is **one of the important steps** for **Continuous Integration**, **Continuous Delivery**, and **Continuous Deployment or Delivery (CI/CD)** practice.

What is Continuous Integration, Continuous Delivery, Continuous Deployment, or Delivery? Why are they important?

Continuous Integration (CI) is the practice of integrating any new module developed and testing each change done to your codebase automatically and as early as possible. **Continuous Delivery (CD)** is an extension of continuous integration since it automatically deploys all code changes to a testing and/or product environment after the build stage. With the automated release process, you can deploy your application at any time by clicking a button. **Continuous Deployment (CD)** goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers without human intervention.



CI/CD is to accelerate the feedback loop with your customers and take pressure off the team. CI/CD is a frequently used industry best practice and one of the fundamental **DevOps best practices**. Many **toolchains** are available to support this practice and automation.

Advantages of CI/CD are:

- Increased productivity and quality as developers can focus on building software (increased productivity) and receive fast feedback from customers through a tighter feedback loop and collaboration that allows faster bug fixes (increases the quality).
- Faster time to market
- Reduced risk

How can I deploy my application based on continuous deployment (CD) practice?

In order to deploy a Web application, you need a server. In this example, we will use **Heroku** and **AWS** cloud for our servers.

Deploying an application to the Heroku cloud from the GitHub repository

1. Make sure you have a **repository created on GitHub** and all the program files pushed to this repository and ready to deploy to the production server.
2. In this example, we want to deploy a simple HTML page to **cloud platform Heroku** using a simple **PHP** file '**index.php**' with only one line of code `<?php include_once("hello.html");?>` that includes 'hello.html' file already created in the previous example as Heroku does not host a static website with

only HTML files. We used PHP **only for the purpose of deployment testing**, but **we will not use PHP in this lesson**. Push these two files to the GitHub repository.

3. **Open an account at Heroku for a free plan.** Heroku is a container-based cloud Platform as a Service (PaaS). The free plan offers many Heroku services for free.
4. **Create a new application at Heroku**, e.g., ‘myapp-heroku’ (Heroku requires an available name). You can add a pipeline (that lets you connect multiple apps together), but we will skip it now. You can add a [buildpack](#) at Heroku settings if necessary. Skip this step. We will try this later.
5. **Choose ‘GitHub’ as a deployment method** among Heroku Git, GitHub, and Container Registry. If you choose Heroku Git, you use Heroku CLI. For this, install the Heroku CLI, log in to your Heroku account by ‘heroku login’, and connect your local repository to the Heroku repository by ‘heroku git:remote -a myapp-heroku’, push the code to the Heroku repository to deploy it to Heroku.
6. Choose either Automatic deploys or Manual deploy. We will choose “Manual deploy” until we set up a Continuous Integration service configured.
7. **Connect Heroku to Github** by entering the GitHub repository name, ‘myreact-heroku’.
8. **Open the application** ‘Open app’ or ‘View’ or open a page ‘https://myapp-heroku.herokuapp.com/’ using your browser. If everything went well, you should be able to see the ‘hello.html’ page loaded by ‘index.php’.

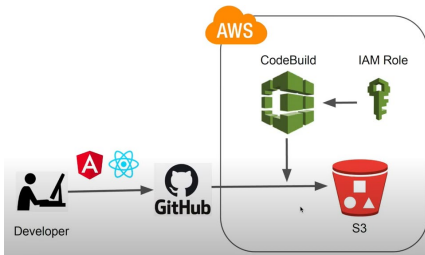
You can also **deploy from the Heroku repository** by choosing ‘Heroku Git’ instead of GitHub as the deployment method, following the steps below:

- Install Heroku CLI and verify it by ‘heroku -v’
- Log in to Heroku using Heroku CLI by ‘heroku login’
- Connect your local repository to Heroku Repository by ‘heroku git:remote -a myapp-heroku’.
- Commit and push your code to the Heroku repository.
- Open your application from <https://myapp-heroku.herokuapp.com/>.

Deploying a simple web application to AWS using Amazon S3

1. **Create an account at AWS.** You may want to try the [AWS free tier](#) that allows you to explore and try out AWS services free of charge up to specified limits for each service.
2. **Create a bucket in S3:** enter a bucket name, e.g., ‘myapp-bucket-aws’, -> uncheck ‘Block all public access, click ‘Create bucket’ button. If everything goes well, you should be able to see a list of buckets created on the Buckets page. A **bucket** is a container for objects stored in [Amazon S3](#) (Simple Storage Service) offered by **Amazon Web Services (AWS)** that provides object storage through a web service interface.
3. **Find the bucket** you just created and click it to upload files.
4. **Upload an HTML file**, e.g., ‘hello.html’ (In AWS, we don’t need to use ‘index.php’) and close the upload page to go back to the **bucket object page** ‘myapp-bucket-aws’.
5. **Allow the site for public access:** Click ‘Properties’ -> Edit ‘**Static website hosting**’ in the bottom of the page (for ‘hello.html’) -> Enable ‘Static website hosting’ -> Type ‘hello.html’ to specify the default page of the website (index.html or home.html are typical default page) -> Click ‘Save changes’ (or click ‘Permissions’ -> Edit and uncheck ‘Block Public Access’ -> check the box to accept the acknowledgement).
6. **Create a Bucket policy for the site:** Click ‘Permission’ on the bucket object page -> Edit ‘Bucket policy’. If you have not generated any policy, it will ask you to generate a policy, or it will show you a previously generated policy. **Bucket Policy** is a resource-based policy that you can use to grant access permissions to your bucket and the objects in it. Read [more information about the bucket policy](#). **Copy Bucket ARN** (Amazon Resource Name) for your application to generate a bucket policy. ARN uniquely identifies AWS resources.

7. **Generate a bucket policy statement:** Click 'Generate policy', select 'S3 Bucket Policy' for the type of policy, 'Allow' for Effect, '*' for Principal, 'GetObject' for Actions, copy and paste ARN (add /* to allow all files), click 'Add Statement', click 'Generate Policy' (that will show a policy statement in JSON format), **copy the bucket policy statement**, click 'Close, and paste it to Bucket policy editor, and choose 'Save changes.' It should show a success message.
8. **Test your website endpoint:** Buckets -> choose your bucket -> Properties -> At the bottom of the page, under 'Static website hosting', choose your Bucket website endpoint or click the URL to open your site.
9. For more information, refer to [this tutorial page](#). You will see the default page of your site.



Deploying a simple web application to AWS using Amazon S3 from GitHub for Continuous Deployment

1. Create a new bucket at AWS, e.g., 'myapp-bucket-aws-github'.
2. Upload an HTML file to the bucket.
3. Allow public access to the bucket.
4. Create a bucket policy for the bucket. Now you should be able to manually create a policy statement by directly typing it in the policy editor.
5. Open the page of the IAM dashboard (by searching it by keyword) to set up AWS IAM (Identity and Access Management). [AWS IAM](#) is a web service that helps you securely control access to AWS resources. You use IAM to control who is authenticated (signed in) and authorized (has permission) to use resources.
6. **Add an IAM user** by following the [steps here](#) (assuming you didn't create a user): User name (e.g., myapp-aws) -> check 'Programmatic access for AWS credential type and click Next: Permissions -> click Attach existing policies directly -> type s3full for Filter policies -> select AmazonS3FullAccess -> click Next:Tags -> click Next: Review (ignoring Tags in this example) -> click Create user (You should see a success message if this process went well) -> (optionally download .csv that contains a user name, access key ID, secret access key, console login link that will be used later) -> In the bottom of the page click 'Close'. You should be able to see a user created.
7. **Verify if ACLs are enabled:** Open your AWS bucket page for this application -> Permissions -> Edit on Object ownership -> select 'ACLs enabled' for the bucket and check the acknowledge statement -> Save changes.
8. **Open the repository on GitHub to deploy.**
9. **Add credentials for remote server** (AWS Access ID and key for a bucket): Click **Settings** -> **Secrets** (on the left menu bar) -> Actions -> New repository secret -> type YOUR_SECRET_NAME and value pair by 'AWS_ACCESS_KEY_ID' and ID value, 'AWS_SECRET_ACCESS_KEY' and key value (from the IAM user).

10. **Create a workflow at GitHub for automatic deployment:** In VS code, create a folder `‘.github/workflows’` and add a YAML file, e.g., `‘main.yml’`, and type the following text, save it, commit it, and push it to GitHub:

```
name: Upload Website
on:
  push:
    branches:
      - master
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@master
      - uses: jakejarvis/s3-sync-action@master
        with:
          args: --acl public-read --follow-symlinks --delete
    env:
      AWS_S3_BUCKET: ${ secrets.AWS_S3_BUCKET }
      AWS_ACCESS_KEY_ID: ${ secrets.AWS_ACCESS_KEY_ID }
      AWS_SECRET_ACCESS_KEY: ${ secrets.AWS_SECRET_ACCESS_KEY }
```

[YAML](#) is a data serialization language that is often used for writing configuration files. This YML file is based on the template from `‘S3 Sync’` available on the [GitHub marketplace](#). You can search this template by typing `‘sync s3’` on the GitHub Marketplace search box.

11. **Verify the deployment:** Click `‘Actions’` and see automatic deployment done correctly (green color). If failed, click it to see the error messages and correct it.
12. **Test it:** Modify the HTML file and open it from the link provided by AWS. If you see the modified page, you have now successfully set up the Continuous Deployment pipeline with GitHub for your application. If it doesn’t open the page you expected, check the Website endpoints, especially the region (the default region is `‘us-east-1’`) or default page name (`‘hello.html’` in this example).

Additional methods to deploy to AWS from GitHub

- AWS also provides a tool [AWS CodePipeline](#) for a continuous delivery service that enables you to model, visualize, and automate the steps required to release your software.
- [AWS Amplify](#) is a tool provided by AWS that helps mobile and web developers build and deploy full-stack applications on AWS. The tool includes authentication, data store, API, analytics, and push notification.