IT&C 101 - Cloud Computing Lab

Overview

Before you start building your website you will need an HTML page with CSS and Javascript. You will have written this page the HTML lab where you made an html page with inline CSS and Javascript.

With basic HTML, CSS, and JavaScript, you can just open the files in your browser by dragging the HTML page into an open browser window.

However, you want other people to see your website, so you'll want a better solution because you can't just open those files in a browser.

Functionality

- Setup production environment in AWS
- Serve a web page
- Make accounts

Concepts

- Cloud Environments
- Configurations with web servers
- Working with Remote Servers

Technologies

- Docker
- HTML
- Linux
- Apache

Step 1: Creating a GitHub Repository in VS Code

This guide walks you through the steps to create and publish a new GitHub repository directly from Visual Studio Code (VS Code). We'll cover both using the VS Code Git UI and the integrated terminal.

Prerequisites

1. **VS Code Installed**

   Make sure you have Visual Studio Code installed on your machine. Download from https://code.visualstudio.com/ (https://code.visualstudio.com/) if needed.

2. **Git Installed**

   Verify that Git is installed and configured on your system. You can check by running:

   ```
   git --version
   ```

   If you don't have Git, install it from https://git-scm.com/ (https://git-scm.com/).

3. **GitHub Account**

   You must have a GitHub account. Sign up at https://github.com/ (https://github.com/) if you don't already have one.

4. **GitHub Authentication in VS Code**

   - Open VS Code.
   - In the bottom status bar, look for the account icon (▤) or the "Sign in to GitHub" prompt.
   - Click it and follow the OAuth popup to authenticate with GitHub.
   - Once signed in, you'll see your GitHub username in the Source Control panel.

If you dont know what Git or Github is:

Git is the tool you'll use on your local machine to track every change you make to your files. Whenever you "commit," Git snapshots your project's state, so you can revisit, compare, or undo past work. You can also create branches to experiment with new features without touching your main code—then merge them back in when they're ready. Because Git is distributed, every student's computer holds the full history, letting you work offline, resolve conflicts when pulling or pushing changes, and maintain a clean, organized timeline of who changed what and why.

GitHub builds on Git by providing a cloud-based home for your repositories. Once you've committed locally, you "push" to GitHub to back up your work and share it with classmates or instructors. On GitHub you can open pull requests to propose changes, review code together, track issues, and even automate testing with GitHub Actions. By authenticating your GitHub account in VS Code, you get seamless sign-in, one-click cloning of remote repos, and built-in support for creating and reviewing pull requests—all without leaving your editor.

This is a great video on how GIT and Github works: https://www.youtube.com/watch?v=i_23KUAEtUM&ab_channel=VisualStudioCode (https://www.youtube.com/watch?v=i_23KUAEtUM&ab_channel=VisualStudioCode)

Step 1.1: Create a New Folder / Project

1. Open VS Code.
2. Select **File** → **Open Folder…** (or **Open…** on macOS).
3. Create or select an empty folder where your new project will live.
4. In the Explorer pane (left sidebar), you should see your folder structure.

Step 1.2: Initialize a Local Git Repository

You have two main options: using the VS Code Git UI or the integrated terminal. Both do the same Git initialization.

Option A: Using VS Code Git UI

1. Click the **Source Control** icon in the Activity Bar (□).
2. You'll see a button that says **Initialize Repository**.
3. Click **Initialize Repository**.
    - VS Code runs `git init` in your folder and creates a `.git` folder.
4. Once initialized, the Source Control panel will show a list of **Untracked Changes** (your project files).

Option B: Using the Integrated Terminal

1. Open the integrated terminal with **Ctrl+`** (backtick) or via **Terminal** → **New Terminal**.
2. Make sure you're in the project folder (check with `pwd` / `cd`).
3. Run:

```
git init
```

4. Git will create a `.git` directory. You've now got a local repository.

Step 1.3: Copy Your Files into the Folder

Copy your html file that you made in the HTML lab earier in this class into this folder. You can either drag and drop or use the command line. You will need to have completed the HTML lab to be able to complete this lab.

Step 1.4: Stage and Commit Your Changes

Option A: Using VS Code Git UI

1. In the **Source Control** panel, hover over each file and click the □ (plus) icon to stage it, or click **Stage All Changes** at the top.
2. In the **Message** box, type a commit message (e.g., "Initial commit").
3. Click the checkmark (✔) button to commit.

Option B: Using the Integrated Terminal

1. To stage all files:

```
git add .
```

2. To commit:

```
git commit -m "Initial commit"
```

Step 1.5: Create a New GitHub Repository (Remote)

You can either create the remote repo on GitHub's website and then link it, or use the VS Code GitHub extension.

Option A: Via GitHub Website

1. Go to https://github.com/new (https://github.com/new).
2. Enter a **Repository name**, optionally add a description, choose Public or Private, and **uncheck** "Initialize with a README" (this is a file that is usually used to describe the project but you won't need one for this).
3. Click **Create repository**.
4. After creation, GitHub provides instructions under "…or push an existing repository from the command line." Copy the lines that look like:

```
git remote add origin https://github.com/YourUsername/YourRepoName.git
git branch -M main
git push -u origin main
```

5. Paste those commands into your VS Code terminal and run them.

Option B: Using VS Code's "Publish to GitHub" Button

1. In the **Source Control** panel, click the three-dot menu (⋯) → **Publish to GitHub**.
2. If prompted, select or confirm your GitHub account/organization.
3. Enter the **Repository name** and (optionally) a description and visibility (Public/Private).
4. Click **Publish**.
    - VS Code will run the equivalent `git remote add origin ...` and `git push -u origin main` for you.

Step 1.6: Verify on GitHub

1. Open your browser and navigate to `https://github.com/YourUsername/YourRepoName`.
2. You should see your `README.md` and any other files you committed.
3. From here, you can manage issues, pull requests, and more.

Step 1.7: Subsequent Changes

1. Make changes to files in VS Code.
2. Stage and commit via the **Source Control** UI or terminal:

```
git add .
git commit -m "Describe your changes"
```

3. Push to GitHub:

```
git push
```

   - If you used `main` as your default branch, Git will know to push to `origin/main`.

Quick Summary

1. **Initialize local repo**
   - `git init` (or click **Initialize Repository** in VS Code)
2. **Create/update files** (e.g., `README.md`)
3. **Stage & commit**
   - `git add .`
   - `git commit -m "Initial commit"`
4. **Create remote repo** on GitHub (via website or **Publish to GitHub** in VS Code)
5. **Link & push**
   - `git remote add origin <remote-URL>`
   - `git branch -M main`
   - `git push -u origin main`
6. **Verify** on GitHub

That's it! You now have a GitHub repository created and linked with VS Code. Happy coding!

Step 2: Make sure everything is Commited and Pushed to Github

Right now you have one place (a folder on your computer) where your code lives. Eventually, there will be 3:

1. Your Development Environment (your personal computer or a lab computer)
2. GitHub (history of all changes and a place to collaborate with team members)
3. Your Production Environment (a server in the cloud)

You created the repo on GitHub already (the BYU-ITC-101 repo or whatever you named it), and should see the status and any changes you made in the Source Control tab on the left edge of the VSCode window.

VSCode knows how to keep track of changes because of a `.git` directory inside your project root folder. If you can't see the `.git` folder, it may be because by default most operating systems hide files and folders with names that begin with a period.

Step 3: Set up your Production Environment

Settingup the Live Server on AWS

Your Production Environments (Live Servers) will be hosted on Amazon Web Services(AWS) a popular online cloud hosting platform.

Using AWS Academy

1. You should have already received an email to join an AWS Academy classroom. If you have not received an email reach out to an IT101 TA.
2. Follow the instructions in the AWS Academy Learner Lab Student Guide PDF (AWS-Academy-Learner-Lab.pdf)

> *Note: If you do not have a Canvas account you will need to create one. Your BYU Canvas account will not work.*

Setting Up Your EC2 Instance

1. On the AWS Management Console, go to the top left `Services` drop down and select `Compute` > `EC2`.
2. On the left-hand sidebar under `Instances`, select `Instances`. This dashboard is where you can start, stop, and monitor your live server.
3. Click the orange `Launch instances` button in the top right corner.
4. On the `Quick Start` tab of the `Application and OS Images` section, select `Ubuntu`. The correct AMI should be selected by default, but make sure it is `Ubuntu Server 22.04 LTS` and says `Free tier eligible`. You should not need to edit any other configurations in this section.
5. For the Instance Type make sure `t2.micro` is selected. Again, this should be selected by default.
6. The `Key pair (login)` section contains information about SSH keys. **SSH** means **S**ecure **Sh**ell, which is a way to access a remote server via any terminal/command prompt as long as you have the key. An SSH key is a simple unique text document that you will store on your computer to access your EC2 instance. Treat your ssh key like a password. Do not lose your key file, do not publish your key file online anywhere.

Warning

> *If you lose your key pair you will have to create a new server instance! 7. Click on `Create new key pair`. Name your key something useful like `IT101key`. Keep the default settings, then click `Create Key Pair` and store the key file somewhere safe on your computer. 8. In the `Network settings` section, check the `Allow HTTPs traffic from the internet` and `Allow HTTP traffic from the internet` boxes 9. Click the orange `Launch instance` button in the panel on the right side of the screen. 10. Once the instance has been successfully launched, you can see it by clicking `View all instances` in the bottom right corner.*

Connect to Your Instance

Windows Users Only

Move the SSH key from the location it was saved in the Windows file system to the Linux file system.

Open a Ubuntu terminal and run the following command. This assumes that you have downloaded the key to your downloads folder. If this is not the case, modify the file path to match where your key is saved. Make sure to replace `<your windows user name here>` with the user account name you are using.

```
cd /mnt/c/Users/<your windows user name here>/Downloads
```

```
mv <my ssh key file>.pem ~/<my ssh key file>.pem
```

```
cd ~
```

Windows, Linux & Mac Users

```
sudo chmod 400 <my ssh key file>.pem
```

1. On the `Instances` page, find your running instance in the table and right-click on it. In that pop-up menu, click `Connect`. Find the `SSH client` tab and follow those instructions on your computer's terminal to SSH into your AWS machine using the SSH key you downloaded.

> *Note: Windows users need to run the SSH command from the Linux command line. 2. Once you are connected to the server, you should be at a prompt that looks something like this:*

```
ubuntu@ip-<SOME IP ADDRESS>:~$
```

Install Apache2

To serve your website we will need to install apache2.

1. While in your SSH AWS prompt as done above, run these two commands:

```
sudo apt update -y && sudo apt upgrade -y
```

```
sudo apt install apache2
```

2. You can control your server's state with `sudo service apache2 <option>`, where `<option>` is one of `{start|stop|graceful-stop|restart|reload|force-reload}`. The usages of these are fairly easy to know just by the name.
3. Run these two commands to start apache2 and make sure it is running:

```
sudo service apache2 start
```

```
sudo service apache2 status
```

4. You can now see if apache is working by going back to your AWS Instances page selecting your instance and copying and pasting either the `Public IPv4 address` or the `Public IPv4 DNS` into a browser. The default Apache2 page has a red banner across the top that says "It Works!".

> *Note: Make sure to visit your site using HTTP not HTTPS because you do not have HTTPS set up.*

Clone your code

1. On GitHub.com, look at your repo and click the button that says "Clone or download" and then copy the URL in the box.

2. Navigate to the Apache WebRoot:

```
cd /var/www
```

3. Make yourself the owner of this directory:

```
sudo chown -R $USER . && sudo chgrp -R $USER .
```

Note: The next three instructions will be used in most of the labs to update the live server to the new website. Make note of them or remember where to find them.

4. Delete the current `html` folder that's found there:

```
rm -rf html
```

> *Note: Be VERY careful using this. You can break your server if you `rm` the wrong file.*

5. Clone your repo:

```
git clone <the URL you copied from github>
```

> *Unless you have saved credentials, it will prompt you for your username and password. For the username, use your GitHub username. For the password, use your GitHub **Personal Access Token** that you created in step 2.2. (Do not use your GitHub password.)*

6. Create a "Symbolic Link" called `html` that points to the `src` folder in the repo that you cloned:

```
ln -s <folder of your repo> html
```

> *Note: Remove the < and > symbols when replacing the folder name. This applies to anytime < and > are used.*

> *Explanation: This creates a "shortcut" to your src folder. When Apache tries to access `/var/www/html`, it will be pointed to the `src` folder in your project now.*

Remember this command!

In future labs, you will need to clone more repos to your live server and point apache to the right directory. This involves `rm`ing the old symbolic link and making a new one using this command. This is so you don't have to make another `sites-available` file (which we'll cover next) and make Apache reload.

Change Default Config

Apache comes with a default configuration file, and as of the time of writing this, it should be called `000-default.conf`. You'll create your own new site config.

1. Navigate to the `sites-available` directory for Apache:

```
cd /etc/apache2/sites-available
```

2. Copy the default config into a new config:

```
sudo cp 000-default.conf it101_lab.conf
```

3. Learn how to open your file with a text editor To open your file in an editor as root, using

```
sudoedit it101_lab.conf
```

The default editor is `nano`, but `vim`, `emacs`, or `ed` are also standard options. You can change which editor `sudoedit` uses by installing the editor and then running this command, following its prompts:

```
# Optional
sudo update-alternatives --config editor
```

4. Make sure a user can view your website without having to specify the `index.html` file in the URL.

5. Save the new document and then run the following commands to disable the default site and enable your new one:

```
sudo a2dissite 000-default.conf
sudo a2ensite it101_lab.conf
```

6. It says you need to reload Apache, so do it:

```
sudo service apache2 reload
```

Step 4: Set up Accounts

You are now going to set up multitple accounts on your Ec2 Instance. You are going to set up three different accounts, but first you need to allow remote ssh without a key. To do this you are going to edit two files.

Cloud File

To edit the first file type in

```
sudo nano /etc/ssh/sshd_config.d/60-cloudimg-settings.conf
```

You will change the line

```
PasswordAuthentication no
```

to

```
PasswordAuthentication yes
```

sshd_config

You are now going to edit the `sshd_config Enter `sudo nano /etc/ssh/sshd_config` Find and Change

```
 PasswordAuthentication no
```

to

```
PasswordAuthentication yes
```

Also ensure:

```
PermitRootLogin prohibit-password
```

Make sure none of the lines you changed have ## before them, that is how you comment something out. If you did it all correctly once you set up the other users you should be able to ssh into the EC2 instance without a key once you have made the other accounts/users.

If you did that all

Add Users and give user permissions

In this part we are going to let you figure some of it out. You will be required to make three new users

- Bob
- Alice
- Shannon

Bob and Shannon should be able to access the EC2 instance using ssh. Alice should not be able to use SSH only sftp.

For context SSH (Secure Shell) is a network protocol that establishes an encrypted, authenticated channel between a client and a remote server—typically over TCP port 22—enabling you to securely log in, execute commands, and tunnel other services. SFTP (SSH File Transfer Protocol) is a secure file-transfer subsystem built into SSH: it lets you upload, download, and manage files on the remote host over the same encrypted SSH connection, combining the safety of SSH with the functionality of traditional FTP.

Bob should have sudo privileges, Alice to only be able to upload/download files (and not use terminal commands). Shannon should only have access to the /var/www/ files.

Helpful Commands:

Make a new User:

```
sudo adduser newusername
```

Let users login with any type of SSH

```
sudo mkdir /home/username/.ssh
sudo cp /home/ubuntu/.ssh/authorized_keys /home/username/.ssh/
sudo chown -R username:username /home/username/.ssh
sudo chmod 700 /home/username/.ssh
sudo chmod 600 /home/username/.ssh/authorized_keys
```

Grant Sudo privileges:

```
sudo usermod -aG sudo newusername
```

Prevent shell access (SFTP-only and limited commands)

```
sudo usermod -s /usr/sbin/nologin username
```

Restrict read/write access to a certain directory

```
sudo usermod -d /restricted/path username
```

Tips

If you do not understand something or you aren't getting the results that you want, ***google it!!!***

Passoff Rubric

Screenshot of your working website.

Screenshot of all users permissions

```
awk -F: '{print $1 "\t" $6 "\t" $7}' /etc/passwd | while read user home shell; do
    echo "User: $user"
    echo "  Home:  $home"
    echo "  Shell: $shell"
    echo "  Groups: $(id -nG $user 2>/dev/null)"
    echo ""
done
```

This command will show you the users and their permissions. The output is what you should screenshot.