CINS 465 Lecture 16 Outline

- Lecture Focus: **Hosting Your Web Project on Google Cloud**
- **Cloud Computing**
    - Definition: the on-demand availability of computing resources as services over the internet. It eliminates the need for enterprises to procure, configure, or manage resources themselves, and they only pay for what they use.
    - In relation to this class, “hosted on the cloud” means that a user can visit your website without having to host it locally (meaning that I don’t have to type the runserver command and visit your site on localhost)
    - You may create a domain name from a web hosting service, but you do **not** have to – you can just give me an IP address
    - Amazon Web Services (AWS) and Google Cloud Platform (GCP) are two examples of cloud computing services that offer education credits
    - A Google Cloud Credit coupon is available for you (see retrieval link on Blackboard homepage)
- **Google Cloud Platform**
    - Visit https://cloud.google.com/ then click on **Go to console**
        - This will take you to a console for your default Google account
        - Does **not** need to be your @mail.csuchico account
    - We will use Google Compute Engine
        - For more info: https://cloud.google.com/compute
- **Apply GCP Credits**
    - https://console.cloud.google.com/education
    - Enter your name and the coupon code, and agree to the terms of service
    - After you Accept and Continue, you’ll be directed to your account’s billing page, and you should see your $50 credit
    - Select the Billing Account for Education
    - You may need to enable the Compute Engine API – this should direct you to the Compute Engine VM Instances page (or you can go to this page using the instructions below)

- **Set up a Compute Engine VM Instance**
    - From your console.cloud.google.com home page, click on the navigation menu (hamburger icon in upper left corner). In the `COMPUTE` section, go to: `Compute Engine -> VM Instances`
    - At this point, your billing should be set up (the Google Cloud credit – no need to enter credit card info)

        - If you do not have any existing VMs, there will just be a button to Create a VM instance
        - If you have an existing VM instance, there should be a link to CREATE INSTANCE
- **Create an Instance**
    - Note that there is a monthly estimate for the default configuration
        - You can change the configuration to make it more or less expensive (and more or less powerful)
        - The workload for your web project should not be that high, so no reason to go for the most expensive option
        - To save money, you may want to power off your instance when not in use
    - Give your instance a relevant **name**
        - e.g. cins465 (follow naming rules)
    - Select a **region** and **zone**
        - Default is fine
        - In certain cases, might want to host it near most of your users (can be faster for those users)
    - **Machine Configuration**
        - Fine to stick with some of the default settings, but may want to update some components:
            - **Machine Family**: General Purpose
                - For common workloads, optimized for cost and flexibility
            - **Series**: E2
                - CPU platform selection based on availability
            - **Machine type**: e2-medium (2 vCPU, 4 GB memory) (default)
                - Each virtual CPU (vCPU) is implemented as a single hardware hyper-thread on one of the available CPU Platforms.
                - Shared core: lower performance (could be throttled at some point), but affordable (good when just starting out or working on school projects)
                - In class, I'll use e2-standard-2 (2 vCPU, 8 GB memory), because it will be faster for setting things up during class, but will migrate to a smaller machine after initial install and set up – this is the max I would use for this class (you can probably get away with the smallest machine)
            - **Boot disk**:

    - Change from Debian (default) to Ubuntu (version Ubuntu 20.04 LTS)
    - Boot disk type: balanced persistent disk
    - Size (GB): increase to 40 GB (increases monthly cost, but helpful if you're using lots of images/media
- **Firewall**
    - Allow HTTP traffic and HTTPS traffic
    - (Default: all incoming traffic from outside network is blocked)
    - Allow both so we can easily use either port 80 (HTTP, the default for unencrypted web pages) or port 443 (HTTPS, encrypted, uses a TLS certificate)
- Click on the toggle for **Networking, Disks, Security, Management, Sole-Tenancy**
- **Networking => Network Interfaces**
    - Click on the toggle option to edit the default network interface – leave as default: **Network**, **Subnetwork**, and **Primary Internal IP** (should be Ephemeral (Automatic))
    - **Network Service Tier**
        - Select: **Standard** (us-central1)
        - This is adequate for your CINS465 project, but would probably want Premium for anything going to production
    - **External IP**: click on the toggle option and **Create IP Address** – give an appropriate name and description
        - This will generate a static IP (so it won't change every time you stop and restart the machine)
    - Click DONE (for Networking component)
- **Security**
    - Leave most defaults, but click on the MANAGE ACCESS toggle
    - **Add manually generated SSH keys**
        - Create an SSH key (in a terminal on your local computer) if you don't have one already – if you do have one, no need to create a new one, just copy and paste the public key where it says "Enter public SSH key"
        - E.g. copy everything printed to the terminal

        - `$ cat ~/.ssh/id_rsa.pub`
    - We're done – Click **CREATE** to create your VM instance

- **SSH into our VM**
    - Can access our VM:
        - (1) through a terminal or
        - (2) Connect SSH (open in a browser window)
    - SSH into the generated External IP address through a terminal. For example, if your external IP address is 35.209.22.150:
        - `$ ssh 35.209.22.150`
        - Should see something like this: `yourusername@vminstancename:~$`

- **Install Docker on our VM**
    - Follow these instructions: https://docs.docker.com/engine/install/ubuntu/
    - Install using the repository:
        - Complete the 3 steps to **Set up the repository**
        - Complete step 1 to **Install the Docker Engine**
            - You do **not** need to install a specific version
            - You may want to complete step 3 to verify the install
    - Scroll to the bottom of the page – under Next Steps, click on the link to **Continue to Post-installation steps for Linux**
        - https://docs.docker.com/engine/install/linux-postinstall/
        - Follow the steps to **Manage Docker as a non-root user**
            - Make sure to exit and then ssh back in to confirm you can run commands without sudo

- **Git Clone your web project to your remote cloud instance**
    - Check that you have git installed: `$ which git`
    - Need to create an ssh key for this machine:
        - `$ ssh-keygen -t rsa -b 8192`
        - Copy the key and add it to GitHub: go to your personal account => settings => SSH and GPG keys => add a New SSH Key (I named my key GCP Deploy Key) – it should have read/write access
    - Alternatively, you can follow GitHub's instructions to Generate a new SSH key and Add a new SSH key to your GitHub account
    - On your remote VM instance, run git clone with your web project repo:
        - `$ git clone git@github…`
    - Confirm that you have your code on the cloud instance

- **Install docker-compose on the VM instance**
  - https://docs.docker.com/compose/install/
  - Install compose on Linux systems:
    - (1) Run the command to download the current stable release of Docker Compose
    - (2) Apply executable permissions to the binary
    - (4) Test the installation (step 3 is optional)

- **SSH into our VM instance through VSCode (optional)**
  - Can use VSCode for remote editing (so it's just like remote editing, but saved on the virtual machine)
  - Inside VSCode, go to Extensions (in the left sidebar)
  - Install the **Remote Development** package (search for "remote ssh" if you don't see it right away)
  - The Open a Remote Window symbol will appear in the lower left corner
  - Select **Connect to Host…**
  - Enter: user@host (for your username and hostname)
  - Can find your hostname using the command: `$ ip a`
    - Look for inet IP address (should match VM instance external IP)
  - This also works with your local VM – just make sure to install the OpenSSH server: `$ sudo apt install openssh-server`

- **Set up nginx**
  - Nginx is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache
  - I recommend you use nginx for hosting your site (useful for web sockets and getting your website closer to production-ready) – avoids externally exposed port 8000 (use as a proxy connection)
  - (in a future class, will discuss Caddy (an alternative option if you want to host on a specific domain name), but using nginx is good for this project)
  - Go into your repo (on the VM), start the service named web, and run the migrate command:
    - `$ cd CINS465reponame`
    - `$ docker-compose run web /bin/bash`
    - `# cd mysite`
    - `# python manage.py migrate`
  - In settings.py, add our external IP address to the list of allowed hosts (this list includes localhost by default):
    - `ALLOWED_HOSTS = ['35.209.22.150']`

- Make sure you are working as yourusername@vminstancename (e.g. `shelleywong@cins465-s22`, not root or another user) to complete the next step – this will help ensure that you have read and write access to all newly created files
- Create an nginx folder in our root repo directory – within the nginx folder, create a Dockerfile and another subfolder called sites-enabled. Within the sites-enabled folder, create a file called default.conf:

```
- CINS465reponame/
    - nginx/
        - sites-enabled/
            - default.conf
        - Dockerfile
```

- In the **nginx/Dockerfile**, include the following:

```
FROM nginx:latest
# RUN rm /etc/nginx/conf.d/default.conf
COPY sites-enabled /etc/nginx/conf.d
```

  - This allows us to delete the default files and use the Docker configuration files from the host (https://www.nginx.com/blog/deploying-nginx-nginx-plus-docker)
  - **Note about the commented-out line**: Previously, I needed the line, "`RUN rm /etc/nginx/conf.d/default.conf`", but the most recent time I tried to build nginx, this caused an error (should run fine if you remove this line)
- Our sites-enabled/default.conf file is based off these instructions: https://nginx.org/en/docs/beginners_guide.html
- I'm starting with an example from a previous semester: https://github.com/CSUChico-CINS465/CINS465-F21-Examples
  - Go to the nginx/sites-enabled/default.conf file and look at the history: code added with the GCP basics commit
- In the **nginx/sites-enabled/default.conf** file (using default server_name):

```
server {
    listen 80;
    server_name www.example.org;
    charset utf-8;

#    location /static {
#        autoindex off;
#        alias /static/;
```

```
#    }

    location / {
        proxy_pass http://web:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
    }
}
```

- The commented-out section can be used if you are not using WebSockets (just serving static content): https://docs.nginx.com/nginx/admin-guide/web-server/serving-static-content/
- The `location` code is for setting up an nginx reverse proxy: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/

- **Build an nginx container**: update the **docker-compose.yml** file:
  - https://docs.docker.com/compose/compose-file/compose-file-v3/
  - (the following just shows updated lines of code)

```
services:
     web:
          ...
          restart: always
          expose:
               - 8000
          #  ports:
          #     - 8000:8000
          user: 1001:1002

     nginx:
          restart: always
          build: ./nginx/
          depends_on:
               - web
          ports:
               - 80:80
          links:
               - web:web
```

        - `restart: always` – Always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted.
        - `expose` – Expose ports without publishing them to the host machine - they'll only be accessible to linked services. Only the internal port can be specified.
        - `user` – specify the user id and user group (for linux). Find using the following command (grep yourusername):
            - `$ cat /etc/passwd | grep shelley`
            - (in my case, user id and user group – 1001:1002)
    - Note: you should have included db.sqlite3 file in the .gitignore (no reason to save the database to commit history) – thus, you should be starting fresh here (nothing in the database)
    - **Build the nginx service, start the server, and visit our website**:
        - (exit docker if already up and running)
        - `$ docker-compose build nginx`
        - `$ docker-compose up`
        - Go to our external ip address in the browser
            - Should be able to see our website (no database)
    - **Note**: I ran into an issue building the nginx container (a known issue for Docker, where you get into an unrecoverable container state and need to stop/remove all Docker containers)
        - To fix, use the the commands found here: https://coderwall.com/p/ewk0mq/stop-remove-all-docker-containers
        - Stop/remove all Docker containers:
            - `$ docker stop $(docker ps -a -q)`
            - `$ docker rm $(docker ps -a -q)`
        - Stop/remove all Docker images:
            - `$ docker rmi $(docker images -q)`
        - Then run the following:
            - `$ docker-compose run nginx /bin/bash`
            - `$ docker-compose up`
    - Create a superuser and test that I can login and add a question
        - `$ docker-compose run web /bin/bash`
        - (change to directory with manage.py file)
        - `# python manage.py migrate`
        - `# python manage.py createsuperuser`
        - (in separate terminal)
        - `$ docker-compose up`

        - Visit your website in the browser (go to your external IP)
            - Test that you can see everything you expect to see, login/register a user, submit forms, etc
        - **Note**: I ran into the error, **413 Request Entity Too Large** when trying to add an image file
            - https://www.keycdn.com/support/413-request-entity-too-large
            - Added a line to my nginx/sites-enabled/default.conf file:

```
server {
      …
      client_max_body_size 100M;
}
```

            - `$ docker-compose build nginx`
            - `$ docker-compose up`
            - After these steps, was able to go to the site and add an image

- **Stop the VM instance (if not in use)**
    - Go to your Compute Engine VM instances on console.cloud.google.com
    - Go to the three dots to the right of your VM instance (More Actions) and select **Stop**
    - Can also Start/Resume from here when you're ready to start developing again

- In future lectures, we will discuss more ways to use Google Cloud, but you mainly need to do what's in this lecture for your final project