CS 346 (Spring 2025): Cloud Computing

Projects #7,8

Project 7 due at 11:59pm, Tue Apr 22, 2025
Project 8 due at 11:59pm, **Wed** May 7, 2025

REMEMBER: Javascript is **BANNED** in this project - and all following projects - until/unless I say otherwise.

1 Overview

This spec describes two projects, where the second builds on the first. In the first part, you will build a simple web interface, which allows you to create new cloud VM instances, automatically install some software on them, and then do very basic monitoring of their state. This first version will not be designed to be accessed by a user; instead, it will be a set of fairly low-level CGI scripts, more designed for developer (or computer) use.

In the second project, you will add additional features: some trivial cookie support, the ability to terminate an instance, and a human-readable interface.

1.1 Video Requirement

The TAs are finding it difficult to hand-grade our projects. And since our work is based on the cloud, it's obviously impossible for you to leave your servers up for them to test. (Even if it was possible, it might be a bad idea, since your code can create multiple cloud instances!)

To aid them in grading, you will submit a video which serves as a demo of your website; you should walk through each of the required features, and demonstrate that your code works. While the TAs will also look at your code, they will use this video as the primary mechanism to make sure that your code works as required.

Please upload your video to a video sharing site (an unlisted video on YouTube would be fine), and include, in your Gradescope submission, a README file which tells your TA how to find the video.

1.2 What These Servers Will Run

The cloud VM instances that you create must run some sort of code, and provide an online service. This service must be something that is accessible to the world outside; that is, a computer outside the specific cloud environment you choose will connect to the instance, and get some sort of useful service.

I **strongly prefer** that you set up some sort of application that takes at least a little bit of CPU power (such as a game server); otherwise, the ability to "add another instance" isn't particularly interesting. However, since not all people have access to the same game clients (making it hard to test things), I'm giving you wide flexibility. At a bare minimum, you can view each new cloud VM instance as a web server - all of them sharing your project 6 website. Hopefully, most of you will do something more interesting - setting up a server for Minecraft, Factorio, OpenRA, or something else.

1.3 Cloud Providers and REST APIs

From a cursory glance at the Internet, it looks like many cloud providers provide a Python library that you can call directly from your CGI scripts. As a heads up, you will likely need to generate an API token for this process. Make sure you don't share or post this API token publicly, as it will allow anyone possessing it to perform operations on your cloud environment - **treat it as a password!** Additionally, submitting this API token as part of your project code will result in a **significant grade deduction**.

For this specific project, feel free to share tips and tricks for the APIs of your chosen cloud service with your classmates. However, please don't share code or API calls for specific sections of this project. Piazza may be an excellent place for these sorts of posts.

Additionally, be very careful about not exceeding the free tier of your chosen cloud provider. Cloud costs can unexpectedly add up very quickly, especially when you have a large number of instances! For this reason, in your testing, please only create at most two instances at a time and make sure you delete any failed deployments to avoid leaving them running in the background. Additionally, please make sure to delete your cloud instances after you submit the project!

1.3.1 Be Polite in the Public Space

If you want to host a game server (hooray!), please think carefully about how your server will interact with the wider world. Does the game automatically connect servers to a central matchmaking server? If so, you may want to disable that feature.

Hopefully, in most cases, you won't have to worry about this. But please think about it before you start creating (and then shortly destroying) a bunch of nodes.

2 Database and REST API

You will have a tiny database. Of course, in a more realistic system, you would have a number of tables - with users, lots of information about the nodes, etc. But in Project 7, you only need a single database; name it `servers`. The `servers` database should have:

- An auto-incremented primary key
- An "owner" (must be non-NULL), which gives the name of the person who created the server
- A "description field" (can be NULL), which gives a description of what the server is for.
- Some form of instance ID to track the specific ID of the cloud VM instance you are using. Depending upon the cloud provider you choose, this may take the form of one or multiple fields.
- A "ready" field that you maintain. This field must be set to 0 when you create a new node, and a 1 once the node has been configured and is ready to use.

In Project 8, you will implement a trivial version of the CSC346 Standard Model for cookies. For that, you will need to add a `sessions` table. The `sessions` table will **not** be visible through REST.

2.1 REST

Part of your website will be a **tiny** REST API. For simplicity, I'm not going to require that you support `POST, PUT, DELETE` for any URL on this REST API; instead, it will be `GET` operations only - so it will only support **reading** state, not changing it.

What's interesting about this REST API is that not all of the information comes from the database (and some of the database fields are hidden). What you will show the user is the following:

- The "server id" - that is, the primary key from the table
- The owner and description
- The public IP address field - which must be blank if "ready" is 0. If "ready" is 1, get this information from your chosen cloud provider's API or SDK.

2.2 REST - Required URLs

You are only required to support two URL patterns for this website. As noted above, you only need to support GET operations for each one. (And to make things even simpler, I won't even require you to send 405 status if somebody uses the wrong method.)

- `api/servers` - The collection of `server` objects.
- `api/servers/<ID>` - A single server

3 User Experience: Main Page (Project 8 only)

In Project 7, you will drive changes using `curl`, and you will check the status by reading the REST API. But in Project 8, there will be a human-readable interface.

The main page of the website will check to see if you have a cookie defined (and create one if you do not). It will also see if that cookie represents a session that has "logged in." If not, then the only action that the user can perform is to "log in."

Since we don't have encryption on our web servers, there's no point in trying to do anything **real** with logging in - so we're just going to pretend. To "log in", a user simply enters their username on a form; this form will send them to a CGI script, which will fill in the "username" field in the `sessions` table in the database. There are **no passwords,** and no security or enforcement of any kind - we're just practicing with cookies.

Once the user is logged in, they should be presented with CGI pages (generating HTML) which allow them to:

- View the servers that they have created (they should not see servers created by other users)
- Terminate one of the servers that they created
- Create one more server

You may either put this all into one big HTML file, with lots of buttons on it (remember, a page may have as many `<form>...</form>` blocks as you want), or you can break it into many pages.

4 User Experience: Creating a New Node

A user requests a new node by running the `create_server` CGI script. In Project 7, you will drive this manually, using `curl`; in Project 8, you will do it using a form.

The `create_server` should complete very quickly (even though creating the new node will take many minutes); it should immediately redirect the user to a status page. For Project 7, it will redirect to the newly-created node's page in the REST API; for Project 8, it will redirect to an HTML page that gives much the same information, but which is designed for human interaction.

Of course, it takes a while for the new node to be created. The user can reload the status page as often as they like - eventually, the node will become "ready," and the IP address will be shown. As soon as this is true, the user can connect to their new server, and use it.

4.1 Details

The `create_server` script uses URL-encoded variables (passed as either GET or POST, you aren't required to check). The user must provide the following variables:

- `user` - (Project 7 only) The name of the user who is creating the node. Must not be empty.

  In Project 8, this information is read from the `sessions` table, not a URL-encoded variable.
- `desc` - The description of the server that is being created. It's OK if this is empty.

Confirm that the user passed the correct variables; report a 400 error if they did not. If the variables are OK, then this script must:

- Create a new node, and start the script which will monitor it for completion (see below for details).
- Add a new entry in the database, with the name, description (if any), and the VM instance ID. (Of course, set the "ready" field to 0.)
- Redirect to the status page for this node - either the REST API for Project 7, or a human-readable page for Project 8.

4.2 Running Things in the Background

Depending upon how the cloud platform you chose works, you will likely need to monitor the server creation in order to know when it's safe to set the "ready" flag for your new node in the DB. Until then, the user won't be able to even **attempt** to connect to the new node, since they won't know its address.)

For this reason, you likely need a background process, running on your main server, which is monitoring the process of creating the new node.

The easiest way to do this is to break `create_server` into two pieces: the CGI script, which runs quickly and immediately gives status, and the background script, which will run for a few minutes before it completes the last of its work. I presume that your CGI script will be written in Python (although that isn't required; `bash` scripts are also possible). Your long-running background script could be written in Python, `bash`, or any other language - you choose.

To kick off a background process from a Python CGI script, you can use the `os.system()` call. In Python, `os.system()` runs a command as if you had typed it on the shell. Additionally, on Unix-like systems, you can run a command in the background by adding an ampersand at the end, like so:

```
cmd &
```

However, in order for Apache to know that your script has completed, you have to do one more thing: you must disconnect the background-script from `stdout` and `stderr`. The easiest way to do this is to use **redirection** - and simply send both streams to `/dev/null`, which is a special file in UNIX, which throws away all of the data that it's sent.

Therefore, if we assume that your background script is called `monitor_new_node`, you could start it (from Python) with the following code:

```
os.system("/path/to/where/the/script/is/monitor_new_node 1>/dev/null 2>/dev/null &")
```

You may find it handy to pass one or more parameters to your background script; for instance, they may want to know the instance ID of the VM to monitor, or perhaps the private IP address of the node. You can pass those by simply adding them as parameters on the command:

```
os.system("background_command arg1 arg2 1>/dev/null 2>/dev/null &")
```

To read those arguments inside a `bash` script, use the variables `$1,$2,$3, ...` like this:

```
#! /bin/bash
arg1_I_was_sent=$1
arg2_I_was_sent=$2
```

5 Summary

(This section simply **summarizes** what you can find on the pages above.)

You must turn in:

- Your source code (**NOT** including any passwords!)

- A README file, which tells the TA how to find your video, where you demo'ed the features of your website

Your website must support:

- A REST API, reflecting a simple table in your database (but with some additional information, which will require you to make queries about nodes using the cloud provider API.
- A `create_server` script.
- (Project 8 only) A human-readable machine status page, which gives the same information as the REST API (for a single machine), but formats it as pretty HTML for humans to use.
- (Project 8 only) A main page (or set of pages, your choice), which requires users to login before they do anything; once logged in, they can see their own servers, create a new server, or destroy one of their own servers.

(see the last page for hints about how to run a Minecraft server)

6 Server Hints: Minecraft

Here's a brief rundown about running Minecraft servers in the cloud. Note that you are free to host any service (it doesn't even have to be a game), but if you choose something different, you will need to do your own research about how to set it up.

6.1 Java

Minecraft is a Java application, and the newest builds of Minecraft use very new versions of Java. You will need to install a JDK (Java Development Kit) on the servers you create in order for the Minecraft server to run.

One version of the JDK is available from `https://www.azul.com/downloads/?package=jdk#zulu`. Read anything you find about rights and restrictions - remember **you are responsible to make sure that you don't violate copyright** - but it appears that this version of Java is open to download and run, free of charge.

For your install script, you will not go to the URL above; instead, you will directly download some version of Java. Decide which one to use, for your own purposes, and hard-code the URL where it resides into your script. The `wget` tool will be useful for you here.

While I'm sure that it is possible to install Java on your VM instance, it's not necessary. It is sufficient to simply extract it into a folder. The file that you downloaded was probably a "tarball", which you can identify by the `.tar.gz` filename ending. To extract a tarball, go to the directory where you would like it to be extracted, and run the command

```
tar -xvzf TARBALL_FILENAME
```

If you extract the tarball, you will find the command `java` (which is what you will need to run Minecraft) in the `bin/` subdirectory.

6.2 Minecraft

Similarly, the Minecraft server is available free of charge (although you have to pay for the client). You can get information about it here: `https://www.minecraft.net/en-us/download/server`. As with Java, you should read the rules and restrictions, and then, when you're clear that you are not violating copyright, copy the current server URL and hard-code it into your init script.

The page that I've shown you will also include a command that will allow you to run the server in "nogui" mode - meaning that it won't try to display a graphical interface. That's an important feature if you're going to run on disconnected servers!

Also, note how much memory Minecraft requires. Investigate how large of a VM node you need, to run a minimal Minecraft server, while staying within the limits of the free tier of whichever cloud provider you have chosen.

6.3 First Run of Minecraft

Before you get too far into writing an init script, test out running Minecraft by hand. You will find a couple quirks - things that you will need to automate, so that your first automated run will go smoothly.

Similarly, you will notice that the very first run of the server (when Minecraft creates the initial world) takes a couple minutes. You may want to add `sleep` commands to your init script, to give it time to start up the world, before you let the users in.

6.4 Restarting After a Reboot

Your VM instances may go up and down from time to time. You want the server to automatically start up when the VM reboots.

I found the following page very helpful. It basically allows you to set up Minecraft as a standard "service" - that is, something which the system knows how to bring up and down. And once it's a service, you can ask the system to auto-start it, every time that the machine boots up. When installing packages, skip "openjdk-8-jre-headless" since we're already installing Java another way (and this is an extremely old version of Java that may not be available on current versions of Ubuntu or support modern versions of the Minecraft server). `https://minecraft.fandom.com/wiki/Tutorials/Server_startup_script#Systemd_Script`