Trinetric Nightingale

Server Song

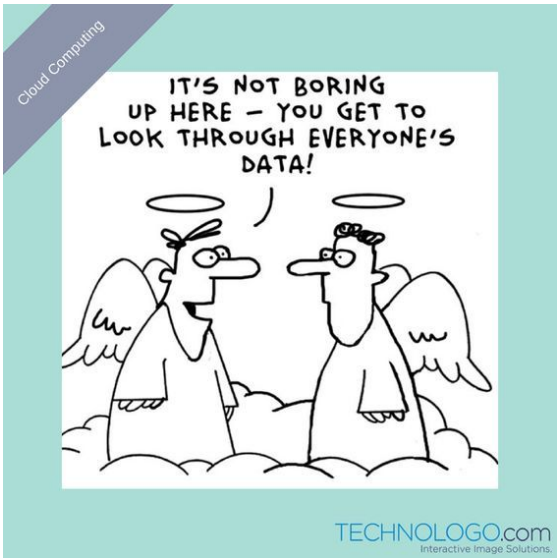
In this quest, you will build your own version of the Server Song web app we talked about in class. Those who didn't attend class (especially the online-only students) - please make sure to watch the class recordings and screencasts we're posting to our YouTube channel (nonlinearmedia). I think Larry is producing a detailed walk through of the first part of this quest (server-based version). Feel free to see it as many times as you want.

AFTER implementing the server-based version that looks at and translates your *sensitive system log file*, you will shift all the computation over the cloud from your server to your clients (in JS) and thereby move your entire application into a FAR cheaper and safer S3 bucket. You will also listen to and translate a safer data source from noaa.gov (cosmic rays).

Overview

Here is the app you'll be building in the first part of this quest.

http://ec2-3-19-219-218.us-east-2.compute.amazonaws.com



Play with it a bit, try to view the source html, css and js. I'll separately share the php file we built in class. I'll leave this server up at least until the end of the quarter.

To have a complete working implementation, you clearly need an EC2 instance (because this is a server-based version). I hope you've completed your first quest already, because you're gonna have to do it all again. At least up to the part that sets me up with a login.

For obvious reasons, this instance should be completely isolated from any sensitive data you wish to protect. Simply firing up a brand new free-tier ec2 instance and creating a new custom key for it ought to be sufficient.

As always, exercise good judgement and caution with everything you do in the cloud. Refer to the comic at the top of our Canvas page from time to time if you need to.

For good measure, before you start this quest, have your budget and budget alarms set correctly. Install the AWS mobile app on your device so you can kill the instance at the tap of a button if you ever suspect it's been hacked (even though it's unlikely). Ask us in the forums if you're unsure how to do any of this.

You may watch the class video and the many different screen caps as many times as you want, and ask for clarifications in the forums whenever you're stuck.

Now let's make some music. Here are the details of Server Song.

Miniquest 1 - Wherefore art thy songs?

Yeah, I think it's funny too. This is the last thing you have to do. Sometimes working out the right order in which to complete these miniquests is itself a miniquest!

When your server is ready, create a public facing website in an S3 bucket. Create a file called `herein-be-my-songs.txt` and put it in the root of your bucket right next to your index.html.

This file should contain a single line, and this line should be the public DNS address of the EC2 instance you have set up as your song server.

Now this URL - of the public facing static website in your S3 bucket (NOTE - not of your EC2 instance) - is the one you submit into the questing site.[1]

You can put anything you want in your `index.html`. I'm not going to look at it. I only need to get your other file.

It's round-about, but it's more fun.

Anyway, at least, you don't have to mess with access policies for now. It's ok to make this file publicly accessible. No need to set me up with exclusive access to it.

[1] Note that all URLs you submit either directly (entering in text boxes) or indirectly (via special files you place in special locations) should be the public facing endpoints that Amazon gives you. Not other domains you may have aliased to them.

Miniquest 2 - Hoo Bee Yoo

This is everything you did in the tadpole quest up to the point of letting me login and read the `~anand/.cs55a-secret` file. But this time, you don't have to set me up to be able to serve my own `public_html`.

Of course, it's not critical that you put your own student id the secret file. But only the student id I find in there (if valid) gets credit recorded for the quest.

Here's the link to my public key, to save you a lookup:

https://quests.nonlinearmedia.org/keys/cs55a-lab-key.id_rsa.pub

Miniquest 3 - Empower your cat

Think the unthinkable. Do the undoable. Only because you know this is an instance you can reimage fearlessly.

In this miniquest, you must create a super cat, a cat imbued with root power, to be able to read from secret system log files that only root has access to. Yeah, I know this part has invisible red DON'T DOs written all over it. But what the heck? What's life without taking a few risks?

To implement this crude hack, you're going to create the directory `/var/www/bin`, and copy your `/usr/bin/cat` command into it.

Use the command `md5sum` just like in the last quest to generate an md5 checksum from your student id again. It must be the same as what you got before. Rename your copy of cat to this checksum. (Use the command "`mv cat` *`123456789abcdef12`*" or something like that. The number will be replaced by your checksum, of course.)

Finally use `sudo` to change the owner of this file to `root` and use `chmod 4755` to give it `root` permissions when executed.

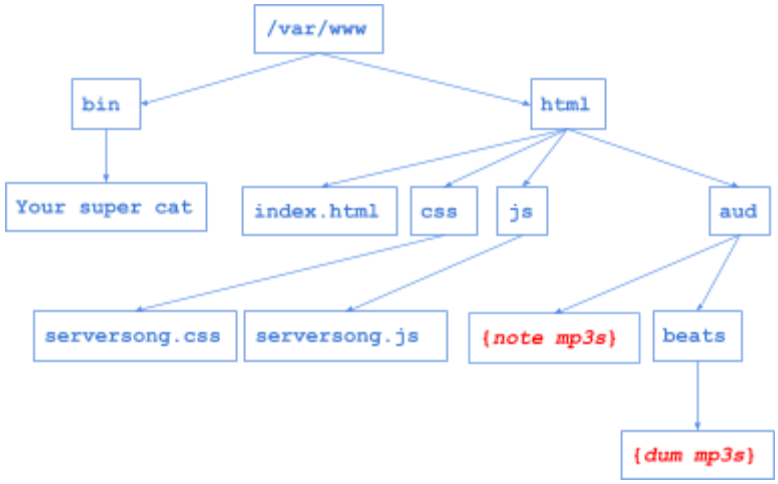
Make sure that the file is executable by any random user who happens to login to your machine as long as they know the full pathname of the file. For example, I should be able to run it from my home directory because I would know the full pathname of this file: /var/www/bin/{md5 of your student id}.

I guess part of doing this dangerous experiment in an isolated virtual machine is to find out just how much power a random hacker into your machine might have if they found out what this command could do. It's better to play with fire in a controlled environment than to be burned later when you don't expect it, right?

If you don't know how to generate the name of this secret executable, review the spec for the previous quest.

Miniquest 4 - Set up shop

You are, of course, free to organize your files and assets any way that suits you. But for this quest, the organization has to mirror the one you saw in the videos on our channel. TLDR? The following picture shows you the gist of it.



Dir structure to set up for Server Song

I won't look at your exact sound file names. This miniquest depends on your ability to reproduce the exact directory structure above around your document root as shown. Note that file and directory permissions matter. You need to think and determine what they ought to be.

You're free to use your own sound files. If you want mine, you can get them from:

https://quests.nonlinearmedia.org/share/aud.zip

Miniquest 5 - Find it in translation

Write your own, or download adapt if necessary, and install the php script. This script would read your system log file (`/var/log/messages`) and respond to POST requests from browsers with a json_encoded vector of numbers. These numbers would be the indices of the first patterns matched in a list of patterns the user would have sent you via POST.

Feel free to discuss any aspect of this script (your own or downloaded) in the forums.

You can get a copy of the php script we hacked up in class:

https://quests.nonlinearmedia.org/share/serversong/serversong.php.txt

After this script has been installed correctly (make sure the filename ends in .php, not .txt), you should be able to run it on the command line without any errors (if POST data is missing it uses a default set of patterns to match with the log lines).

Miniquest 6 - REST up and throw your voice

Now fetch and install the `index.html` and the `serversong.js` files. I'm not going to give you the URL of the JS file here. Instead you can simply select view-source on the demo application and navigate from there to the JS file. If you don't know how to do this, all you have to do is ask and one of us will point you in the right direction. Make sure you pick up the file from my serversong server, not the starsong server (below).

No reward has ever been easier to get, I think.

Miniquest 7 - Don't let anyone kick your assets

Protect your sound files from snoopers. Actually there's no real reason to do it seeing as they're just sound files. But you're going to do it anyway just to show that you can now protect any directory you want from snoopers.

It takes a great deal of effort to thwart determined hackers. So just provide the bare minimum deterrent (hiding the dir contents from displaying in apache) like you did in your previous quest.

Miniquest 8 - Serverless Star Songs

Okay. Brace yourself... for S3 on S3.

'Cuz this is the biggle. A simplee, but a biggee all the samee.

What you've built so far is just something cool for an author or webmaster to listen to in bed. Most normal people (except nosy hackers) don't care two bits what your server is doing. I do, of course. So don't count me.

We might as well give people some (safer) information they're more likely to be interested in...

... like the rate at which we're being bombarded by cosmic rays - Celestial protons. Whoo! it sounds cool even to say it. So that's what we're gonnadoo.

Now, if you consider serving your entire app off S3, you'll remember that you'll have to do without compute resources. Thus php is off-bounds. So you would have to make up for it by creating a real-time stream of numbers tied to some actual phenomena directly from your js.

We depended on a log file in our instance for these numbers.

Now, instead, we have to fetch the data from a (near) real-time public service and do the necessary processing in the client's javascript.

In this miniquest, you must create a version of your serversong app on S3. But this time, call it *starsongs*, not *serversong*. This app will play notes derived from the changing intensity of cosmic rays hitting the earth's atmosphere.

If you don't fancy hacking your own js, you can get mine by browsing to my implementation:[2]

https://listen.starsongs.name

Yeah. I saw the domain on Route 53 for about 10 bucks and so I said to myself "Cool! Let me lease that for a year."

Once you've installed your javascript and made the following changes, you'll find that the app will work purely off S3 without needing an ec2 instance any more.

1. Change all occurrences of serversong into starsongs - this would be in all file names, comments within files, and references to old file names within other files. You gotta find 'em all and change them. (Well, there's only 3 files to look at)
2. Completely remove the php directory. Your starsong app on S3 should only have an `index.html`, a `js/starsongs.js`, a `css/starsongs.css` and the mp3 files under `aud`.

Extra rewards await those who participate in interesting forum discussions on the potential savings per month (if that's all the EC2 instance was doing).

To make sure that I can browse the S3 bucket from where you're serving these files, you must set it up the same way as before, with public read access to all its contents.

Create another neighbor for the index.html in your submitted bucket. This file should be called `herein-be-my-starsongs.txt` and should contain a single line with the public facing website of your S3-based starsong app. Note that you must submit the amazon assigned public endpoint here also.

This miniquest is so easy that you will have to be flying blind most of the time. You won't get any feedback messages unless something goes horribly wrong. Of course, you can skip it silently if you want (not recommended).

[2] In case you're interested in the details, I set up the domain as an alias to the S3 bucket and created a cloudfront distribution which sources from this bucket. Then I got ACM to generate a free SSL cert and attached it to this domain on cloudfront. I'm happy to show you how if you want (and we have time).

If you're interested…

This is what the new javascript (`starsongs.js`) does.

Instead of invoking your php script via a POST request, it issues a GET request to a data server at the National Oceanic and Atmospheric Administration service.

https://services.swpc.noaa.gov/text/goes-magnetospheric-particle-flux-ts1-primary.txt

This cloud service, offered by noaa.gov, provides a near real-time feed of various interesting phenomena. The data in the above feed looks like this:

```
# Prepared by the U.S. Dept. of Commerce, NOAA, Space Weather Prediction Center
# Please send comments and suggestions to SWPC.Webmaster@noaa.gov
#
. . .

# Label: P1 = Flux from 95 keV Protons  units #/cm2-s-sr-keV
. . .

#    1-minute Average GOES Energetic Particle Flux Channels (Telescope 1)
. . .

# YR MO DA  HHMM    Day     Day      P1         P2         P3        . . .
2019 10 15  2252   58771   82320  2.53e+03   3.57e+02   7.97e+01      . . .
. . .
```

As you can see in the description above, the P1 column stands for a measurement of the average protonic energy (cosmic protons hitting our magnetosphere, I think), sampled every minute. Each line stands for the values recorded in the measurements taken during that minute.

We'll use this data as a *real time* source of the intensity with which we're being bombarded by cosmic rays. We can thus give our users an opportunity to turn the variation in the intensity of these rays into music.

That is what the new javascript does. It polls the cloud data service at noaa. When this data arrives, it extracts the P1 measurement column and processes it line by line.[3] Starting at the second most recent line and working backwards, it emits one of the following words for each line of P1 data: LOW, LEAST, LESS, SAME, MORE, MOST, HIGH,

[3] Each line in the file is a measurement taken during that minute. There is one line per minute.

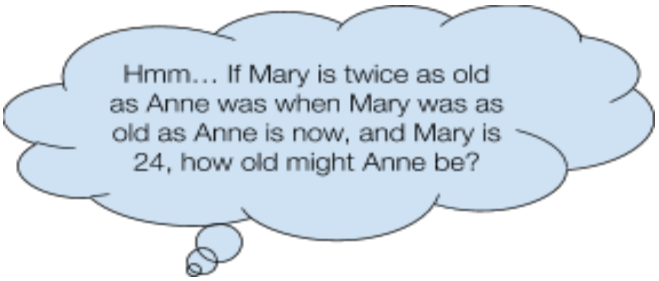
by comparing the measurement on that line with the corresponding measurement on the previous line, and with two extreme thresholds (for LOW and HIGH)

You can expect the output of the script to look something like:

```
SAME
LEAST
MORE
LEAST
MOST
MOST
LESS
MORE
LESS
LESS
LOW
...
```

I'm not printing the timestamps. But you can assume that the words pertain to per-minute P1 measurements in reverse chronological order (with the current minute at the top).

The above data means[4] that the cosmic ray intensity during the minute before the current one was the same as it is now. But the minute before that, it was more than it was during the minute before now, etc. Parse that at your leisure.



The user of your web app would try and match one or more of the above words in this feed. For example, suppose they had typed in:

```
low
least
less
same
more
most
high
```

[4]This assumes that the first line stands for the current moment, which is inaccurate. It is an approximation to the true value.

The new javascript would play the notes corresponding to the array `[ 3, 1, 4, 1, 5, 5, 2, 5, 2, 2, 1 ]`, which would be the third, first, fourth, first, fifth, fifth, ... notes in its chosen scale at the rate of one note per beat.

Here's an interesting question for you to discuss in the forums: Your serverless star song app is interactive, responsive, and works on near real-time data. It is served off of an S3 bucket, with no need for an EC2 instance.

Is this a lambda? Is it a special class of lambda functions that S3 can fulfil? What do you think (not what the documentation may say)?

EC2 (Extra credit too) rewards await thoughtful discussions of this question in the forums.

Miniquest 8 - Share the good stuff and get liked

Yeah! Go for it. Get a public domain on Route 53 or somewhere and attach it to your serverless star song app.

Share it on Canvas, and with your friends not on Canvas.

Hope they like it. Hack it until they do.

Testing your setup

Like I said, the best way to make sure you ace all the miniquests is to set it all up in such a way that your experimental user is able to do all the things I can.

And none of the things I can't.

Submission

When you think you're happy with your set up and it passes all your own tests, it is time to see if it will also pass mine.

1. Head over to https://quests.nonlinearmedia.org
1. Enter the secret password for this quest in the box.
2. Drag and drop (or copy and paste) the public domain name for your S3 bucket. **This should be the aws assigned name. Not another domain name you have attached to your IP**. I bolded it 'cuz some of you got confused in the tadpole quest.
3. Wait for me to complete my tests and report back (usually a minute or less).

Points and Extra Credit Opportunities

I monitor the discussion forums closely and award extra credit points for well-thought out and helpful discussions.

May the best cloudsters win. That may just be all of you.

Happy Questing,

&