How much does a few minutes cost?

Instance Server

Subtitle

Cisco Dynamic RevProxy
Senior Design
2018

Will Rooney
Matthew King
Cecil Hutchings

Table of Contents

Summary

The instance server simulation is designed to simulate a content manager for Amazaon Web Services. When AWS launches an application, it allocates resources at endpoints on the cloud. Our client wanted a list of each endpoint on the server so that they could reach resources that they created and deployed on the server.

By setting up a Cloud Watch lambda function on AWS, we receive a notification each time a project is launched or destroyed. This simulation shows what happens when our system was to receive one of these notifications. It creates and maintains a list of applications that have been launched on AWS. It does so automatically, without the need for human intervention.

Use

To use the simulation, travel to the index of the simulation site. You will be served the user interface of the simulation. It will automatically list any instances that it has. The simulation updates once every five seconds.

Because this project is a simulation, it does not save an instance list. If the server is shutdown, the data on the server will be lost.

Deployment

File Structure:

```
instanceserver/
    build/
    endpoints/
    node_modules/
    pictures/
    public/
    src/
        App.css
        App.js
        App.test.js
        index.css
        index.js
        logo.svg
        registerServiceWorker.js
    package.json
    package-lock.json
    procfile
    Readme.md
    Readme.txt
    server.js
```

Important Folder and File descriptions

a) App.js

Contains the code for the Reactjs user interface. Contains the variable UPDATETIME that specifies the resolution of the updates.

b) Endpoints

Contains the html code that allows the simulated instances to be served by the server

c) Index.css

This is the file with the style sheet for the user interface.

d) Pictures

Pictures used to simulate an aws instance.

e) Public

The public folder contains the html file that is partially used to build the GUI frontend. However, most of the interface is actually build by App.js.

f) Server.js

An express.js web server that manages the site content and serves the application.

Endpoints

/

This endpoints servers the client UI.

/dataurl

This is a POST endpoint. It takes a json representation of an AWS RunInstance. A correctly formatted run instances example can be found in readme.txt. When calling this command, the next time the client UI updates it will add a new instance with a name, specified by the tag 'Name', an ID from the tag 'id' and a URI from 'ApplicationName'.

Our simulation recognizes the endpoints 'bastion', 'consul', 'huron' and 'kibana'. See configuration for how to create a new endpoint. The server will send a 404 response if you attempt to travel to an unhosted endpoint.

/deleteurl

This is a DELETE endpoint. It takes a json representation of an AWS TerminateInstance. A correctly formatted terminate instance json data can be found in readme.txt.

/getml

A GET endpoint. Returns in alphabetical order a list of hosted applications on the server.

/sanity

A local only GET endpoint. Returns a random number. Used as a sanity check that the server is sending data to the user interface, since it updates every time that the server does.

/end/*

Any endpoint here redirects to a hosted endpoint, if the server recognizes a running application. Otherwise gives a 404 error.

Configuration

Limited configuration can be done on the server. The src/index.css file contains the style information for the GUI. The public/index.html serves hosts the GUI and may be built upon as needed, however keep in mind that most of the work can be done with App.js

In App.js around line 5 there is a constant UPDATE time that specifies the update time in milliseconds. This handles the most basic configuration.

To add an endpoint, create an html file and put it into the endpoint folder. Our current system hosts pictures, which we store in the pictures folder of our website.

When reconfiguring the application you will need to redeploy it. See the next section for more details.

Deployment

Deploying the system requires familiarity with git, nodejs. The files for this system are located in the repository:

https://github.com/CoffeeCecil/csserver.git

Once you have the file, run 'npm install' in the InstanceServer directory. This completes local deployment.

Running 'npm start' will start the project running on 'localhost:3002'. Be aware that the command 'npm run build' will recreate the projects cached css and optimized javascript files. **This command needs to be run every time a change is made to a file in the /src/ directory.** The server only uses optimized css and javascript when it hosts the gui. It is not used in the initial deployment of the project.

To create a server deployment on heroku, create an online repository on Heroku that syncs to a git file in the InstanceServer directory. The full instructions can be found here:

https://devcenter.heroku.com/articles/git

To make a new heroku repository for the project commands needed should be:

```
‘git init’
‘git add .’
‘git commit . -m “Commit Message”’
‘heroku create’
‘git push heroku master’
```

After you have a git repository that is syncing into the heroku platform, redeployment is as simple as:

```
‘git add .’
‘git commit . -m “My message”’
‘git push heroku master’
```

If a file in the src directory is changed, the sequence is:

```
‘npm run build’
‘git add .’
‘git commit . -m “Commit msg”’
‘git push heroku master’
```