AWS MAKERSPACE

2023 CAPSTONE PROJECT

MEET THE TEAM

Sumanth Pandiri

OctoPrint Team

CJ Boni

OctoPrint Team

Nayha Hussain

Data Team

Susan Li

Data Team

Sol Lesesne

Data Team

Lucas Boyer

Data Team

INTRODUCTION

AWS has partnered with CU Makerspace, Clemson's hub and community for hands-on projects, to streamline processes to enhance the student and intern experience.

The Makerspace currently has:

1. Built-in management system to track user visits
2. Data visualization dashboards
3. Manually collected printer data logs

Our goal is to employ AWS services to:

1. Create a unified user data management system
2. Incorporate live machine information into the centralized database

3D printers in the Watt Makerspace

Laser cutters and interns in Watt Makerspace

CLIENT

CLEMSON UNIVERSITY MAKERSPACE

Makerspace Student Staff

Demographics and Role: Comprised of 19 student staff from varying degrees of technical prowess. We worked closely with Thomas, the president of CU Makerspace; Gavin, the data lead; and Steven, the webmaster. They oversee safety training quizzes and machine management

Insights and Behavior: The staff requires centralized user data. The existing system of Google Forms and Sheets doesn't allow an efficient connection between user data, safety training quiz scores, and machine usage. In other words, there is no relational database. The current tools don't provide adequate documentation, resulting in difficulties in troubleshooting and onboarding new or non-technical staff. Relational user data is important because CU Makerspace receives funding from the Student Funding Board and other sources that care about demographic data.

Student Users

Demographics and Role: Around 1000 students per semester, coming from all eight Clemson colleges, with a strong lean from CECAS students.

Insights and Behavior: The space serves as a hub for social and collaborative experiences where students can share their hands-on projects with each other. Students utilize equipment like 3D printers, laser cutters, CNC mills, and others to pursue their projects. Some students are unaware of the required forms to use each equipment which makes the creating process more timely. Other students have expressed issues with the lengthiness of the sign-in and printing forms, and with the lack of an intuitive printing process

Faculty

Demographics: Dr. Todd, the faculty advisor for the Makerspace.

Insights and Behavior: Dr. Todd prioritizes low operational costs while still using current technology to run the CU Makerspace smoothly. He was involved in the high level overview of our project and projected outcomes. Dr. Todd encouraged the team to document their design decisions to ensure the ample alternatives were considered before moving forward with proposed solution.

PARTNER & STAKEHOLDERS

PARTNER → AWS

Amazon Web Services (AWS), is a comprehensive and widely used cloud computing platform provided by Amazon.com.

AWS offers a variety of cloud services, including computing power, storage, databases, machine learning etc. to help businesses and individuals build and manage their applications and infrastructure in a scalable and cost-effective manner.

AWS has services to help organizations, such as CU Makerspace, build sophisticated applications with increased flexibility, scalability, and reliability.

Goals//Vision: Amazon Web Services' mission statement is to empower customers and drive innovation through cloud technology.

STAKEHOLDER → CU MAKERSPACE

The successfull implementation of AWS cloud services will allow seamless relational databases for the organization which will increase their efficiency and ability to be a collaborative hub for hands-on projects. Failure to complete project at hand might cause the organization funding or inability to operate due to complicated technology involve and lack of documentation.

STAKEHOLDER → CARRIE RUSSELL

The successful implementation of AWS cloud services to benefit the CU Makerspace will reflect greatly on Professor Carrie Russell and the entire Capstone program because it highlights Clemson senior computer students ability to work on real world projects and deliver stunning results. Failure to complete the project at hand might cause our stakeholders/partners to break our partnership and move on to different vendors.

PROJECT DESCRIPTION

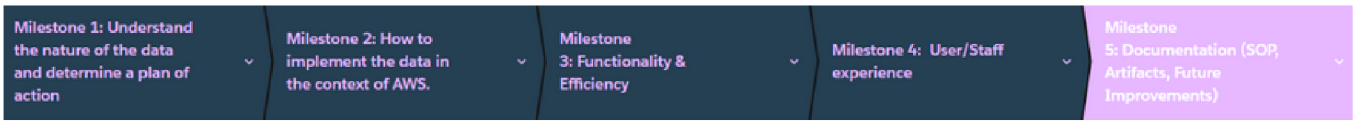
CURRENT PROBLEM/OPPORTUNITY

a. Disjointed data sources make it difficult for student staff to retrieve student user information efficiently.
b. The absence of a system that allows staff to easily check a student's safety quiz status.
c. Manual data logging required when a student uses a machine.

SOLUTION

a. Proposed System: A centralized data hub leveraging AWS alongside Google Sites, becoming a repository for Google form data, user information, safety quiz scores, and machine usage data.
b. Octoprint: OctoPrint offers cost-effective, open-source 3D printing management, seamlessly integrating with Makerspace's existing tools while enabling real-time monitoring and centralized control.
c. Alternative Considered: Platforms like Canvas for data consolidation and 3D Printer OS for 3D printer tracking were considered but were found to be inefficient in terms of cost and functionality.
d. Justification for Selection: Given the constraints of cost (a concern of the faculty) and the need for a unified data hub (a concern of the student staff), the AWS-backed system emerged as the optimal choice. For 3D printing, Octoprint, being free, aligns with the cost concerns and can be integrated with Cura, which Makerspace already uses.

PROJECT SCOPE



AWS

Main goals:

- Create a centralized data hub using AWS
- Be able to access that data with a website or as a file (*or as a visualization***)
- Create documentation and cleanup

Data will come from:

- User information
- Google forms safety quizzes
- User visits
- 3d printer logs

Data needed as file:

- User information
- User visits

Data needed from website:

- User quiz data

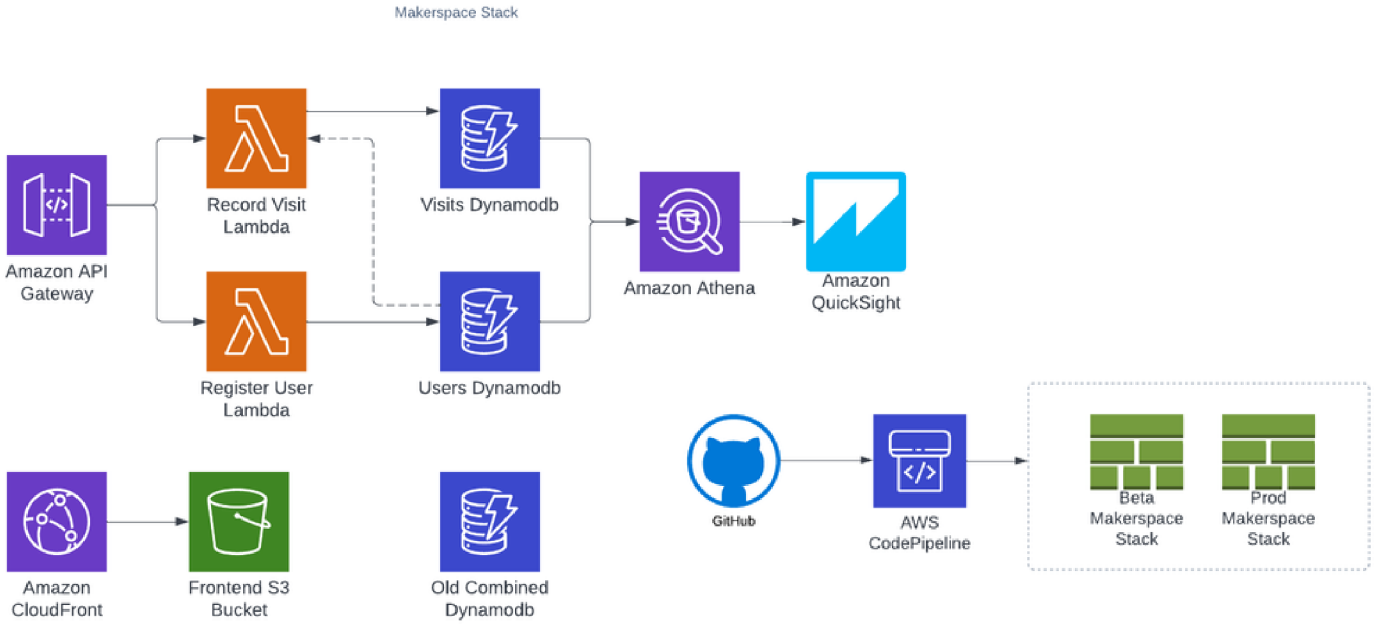
OCTOPRINT

Main goals:

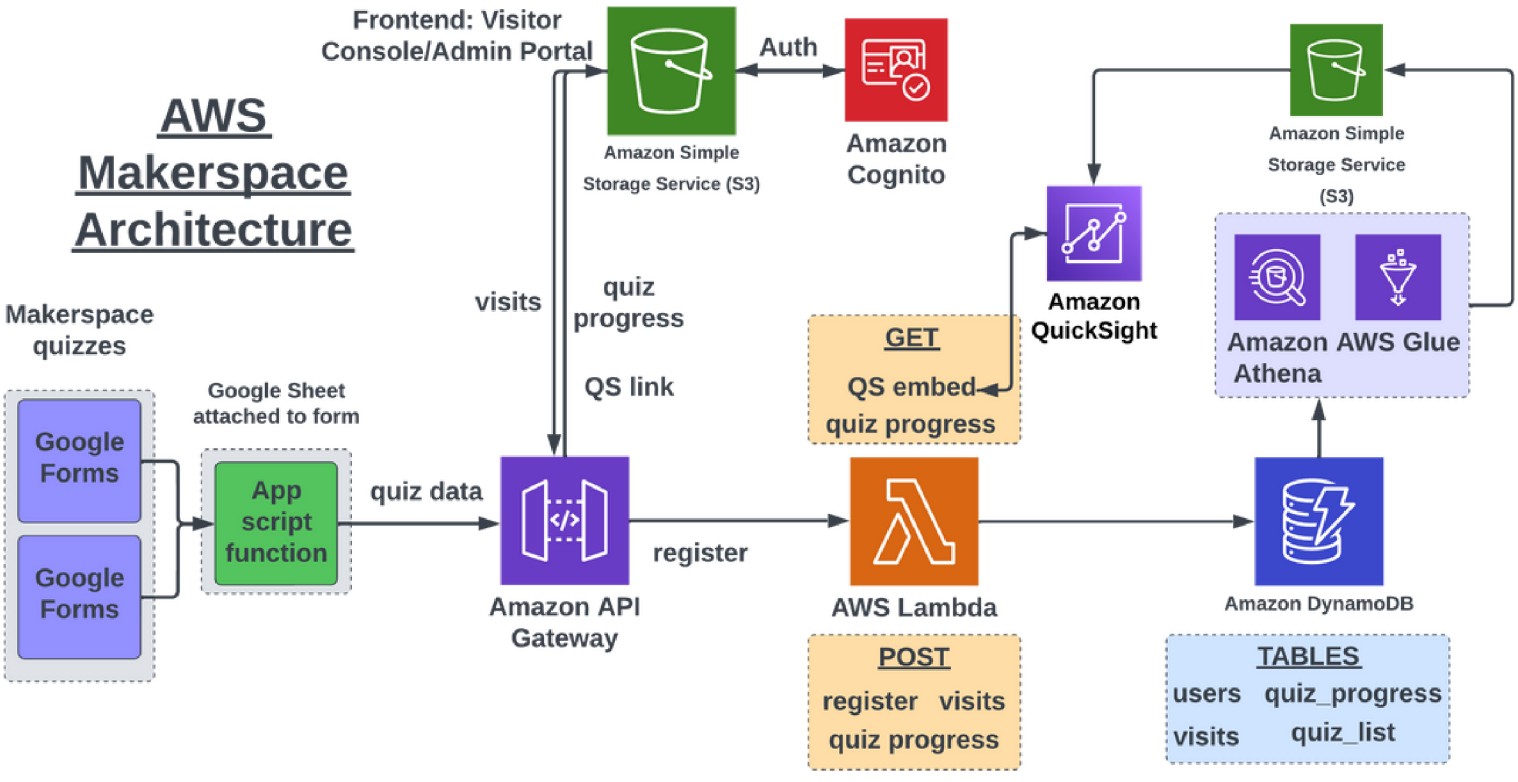
- Incorporate a centralized user interface for 3D printing with multiple printers
- Develop plug-ins for print slicing and printer log data collection
- Create documentation and maintain ease of use

ARCHITECTURE

BEFORE



AFTER



END-TO-END CUSTOMER EXPERIENCE

Example Scenario: Gavin, a staff member, can now easily verify if a user has filled out the necessary forms for using a particular machine, thanks to the centralized data system. First, he will enter makerspace staff credentials. Then, he will be able to use a person's username to find their quiz scores to verify they have taken the necessary precautions to utilize equipment and passed the quiz

TESTING & ITERATION

Definition of Success:

- Successful linkage of quiz scores with user data.
- Successful demo of a working ecosystem using Octoprint
- Centralized access to student user information without needing multiple AWS accounts

.

Measuring Success:

- Monitoring Raspberry pi & octoprint functionality for machine usage tracking.
- Gathering and evaluating feedback from users and staff on system efficiency

DEVELOPER GUIDE

OCTOPRINT TEAM


Recreating our work

Raspberry Pi Setup

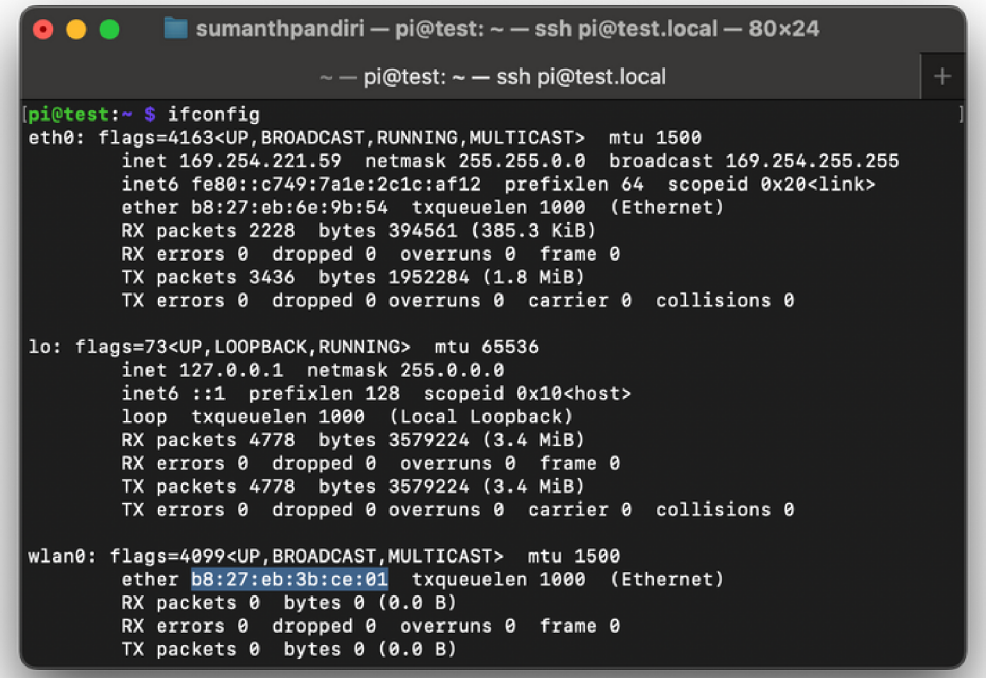
- Image a Raspberry Pi with the OctoPi
  - This requires you to install the Raspbery Pi Imager. Go into “Other specific-purpose OS” -> “3D Printing” -> “OctoPi” -> “OctoPi (stable)”
  - In the settings, set the hostname you prefer (usually the printer name), enable SSH with password authentication and set a username and password.
  - Make sure the SD card is in the Raspberry Pi and selected as storage.
- Connect the Raspberry Pi to power via mini-USB, connect it to the printer via USB, and connect it to the local router via ethernet.
- Connect your computer to the router via ethernet.

Accessing OctoPrint

- To SSH: Open your terminal and SSH via the command:
  - ssh your_username@your_hostname.local
- To access the OctoPrint UI in any web browser, use “http://your_hostname.local”

Connecting Raspberry Pi to the Internet

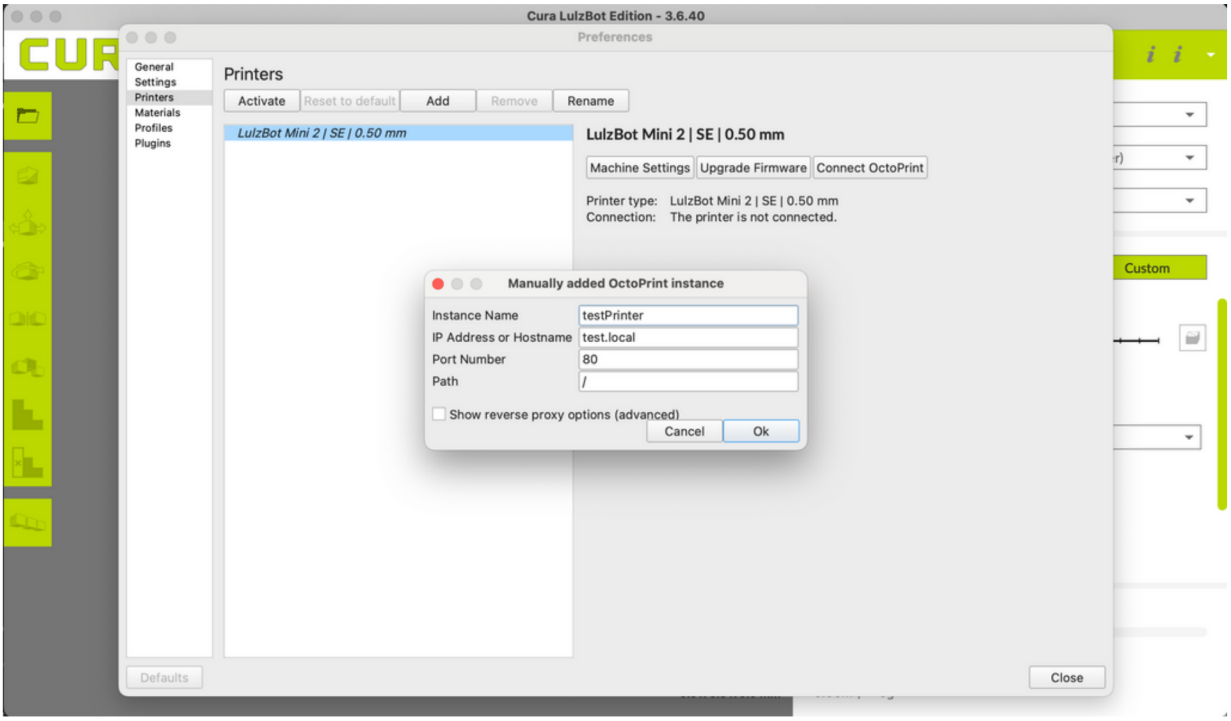
- In the Terminal, SSH using the instructions above, and type “ifconfig”.
- Copy the MAC address - see below:



- Go to http://netreg.clemson.edu
- Log in with your Clemson username and password
- Select Add Registration on the left of the screen
- Enter the wireless MAC Address of your device in the format XX:XX:XX:XX:XX:XX in the box labeled Hardware Address.
- Enter the hostname or printer name as the a Device Description
- Click I Agree at the bottom of the screen
- Back in the terminal, type in the command “**sudo raspi-config**”
- Use the arrow keys to navigate to the network interface setting and press enter
- Enter “resmedianet” into the SSID field and “tigerpaw” into the password field
- Use the arrow keys to navigate to the confirm button and hit enter
- The Raspberry Pi should successfully connect to the wifi

Setting up Cura

- To set up Cura, an API key has to be generated in OctoPrint.
  - In OctoPrint, go to the “User Settings” in the top right.
  - Select “Application Keys”
  - In the “Application Identifier”, type “Cura”
  - Generate the key, and copy it onto your clipboard
- In Cura, go into “Settings” -> “Printer” -> “Manage Printer” -> “Connect to Octoprint” -> “Add”
- Fill in the Instance name with the printer name, and the hostname with the hostname you set up for the Raspberry Pi, as such:



- Click on the printer instance, and paste the API key into the field to the right and hit “Connect”.

Setting up Webhooks Plugin

- To send data to AWS, via the API Gateway, we have to install the Webhooks plugin.
- Open the OctoPrint UI and go to the OctoPrint Settings (wrench symbol at the top).
- Click “Plugin Manager” on the left.
- Click “Add More”, search “Webhooks”, and install the plugin.
- Restart the OctoPrint UI, and go back into the OctoPrint Settings and scroll down to the “Webhooks” section
- Paste the URL from the API Gateway into the URL field and save it.
- In the Advanced section, under DATA, paste this and save:

```
{
  "deviceIdentifier":"@deviceIdentifier",
  "apiSecret":"@apiSecret",
  "topic":"@topic",
  "message":"@message",
  "extra":"@extra",
  "state": "@state",
  "job": "@job",
  "progress": "@progress",
  "currentZ": "@currentZ",
  "offsets": "@offsets",
  "meta": "@meta",
  "currentTime": "@currentTime"
}
```

Setting up AWS

Step 1: Creating the API Gateway:

- Navigate to the API Gateway page within your AWS AdminAccess Account
- Click “Create API” in the top right corner
- Find “REST API” and click build
- Choose “New API”, give it a name, and then click “Create API”

Step 2: Creating the Lambda Function:

- Navigate to the Lambda page within your AWS AdminAccess Account
- Click “Create Function” in the top right corner
- Give it a name, change the “Runtime” dropdown to Python, and then click “Create Function”
- Paste the following code into the code box for the created function and then click deploy:

```
import json
import boto3

def lambda_handler(event, context):
    dynamodbresource = boto3.resource('dynamodb')
    octoTable = dynamodbresource.Table('Octo_Table')

    job = event['job']

    item = {
        'ID' : job['file']['name'],
        'estimatedPrintTime' : job['estimatedPrintTime'],
        'user' : job['user']
    }
    for key in item:
        item[key] = str(item[key])

    response = octoTable.put_item(Item=item)

    responseCode = response['ResponseMetadata']['HTTPStatusCode']

    HEADERS = {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        }

    return {
                'headers': HEADERS,
                'statusCode': responseCode
            }
```

Step 3: Creating the DynamoDB:

- Navigate to the DynamoDB page within your AWS AdminAccess Account
- Click “Create Table” in the top right corner
- Give it a name, enter “ID” in the Partition Key field, change the Read/Write capacity settings to “On-demand” and then click “Create Table”

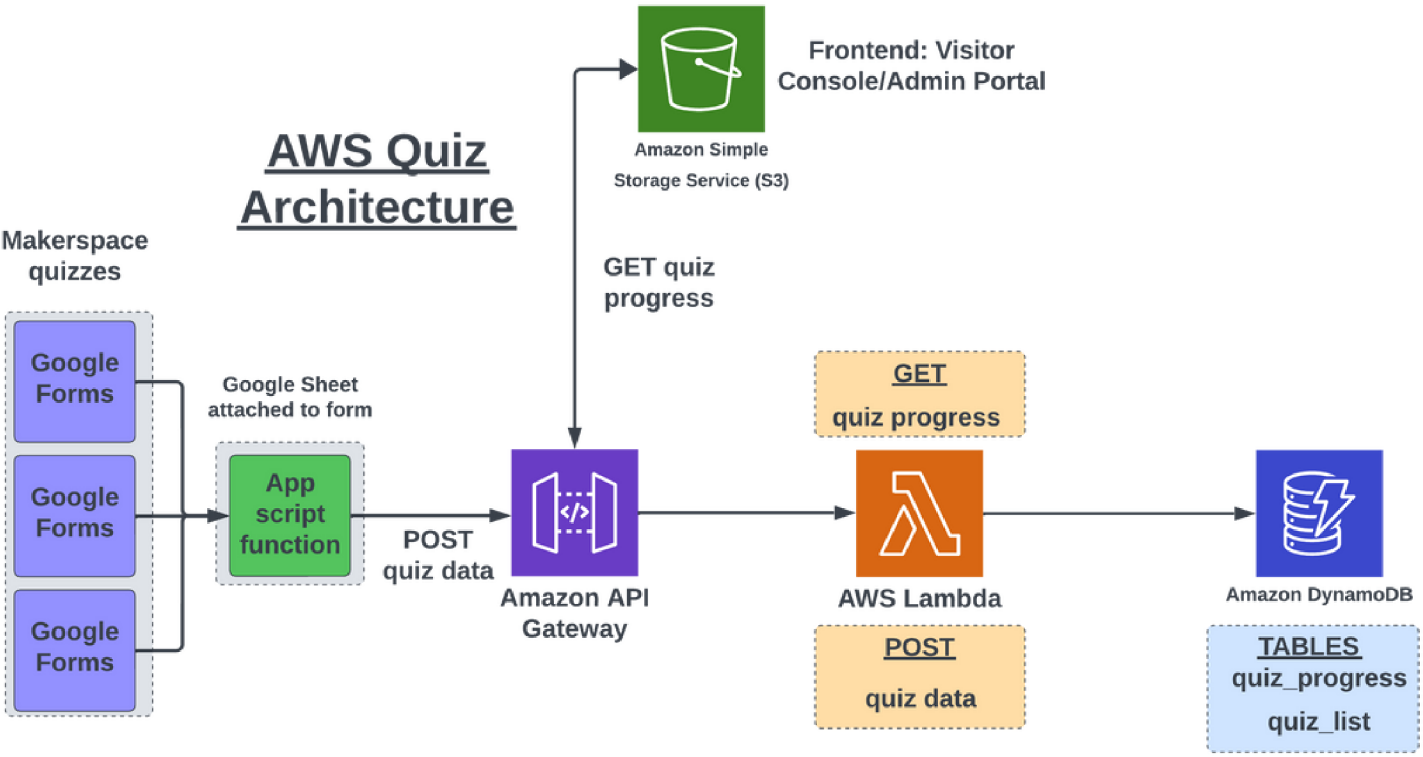
Step 4: Connecting the three:

- Navigate back to the created Lambda function.
- Within the dynamodbresource.Table() function, replace the string with the name of the table you just created and deploy it again
- Click on the configuration tab then on the permissions tab within
- Click on the linked name under “Role Name”
- Click “Add permissions” then “Create in-line policy”
- Choose dynamodb in the service dropdown
- Choose all actions and all resources then click “Next”
- Give it a meaningful name and then create it
- Navigate back to the API Gateway that you created
- Click “Create Method”
- Under “Method Type”, select POST
- Select Lambda, and then the Lambda function you created where it says to choose it
- Click “Create Method” and everything should be in working order

DEVELOPER GUIDE

DATA TEAM

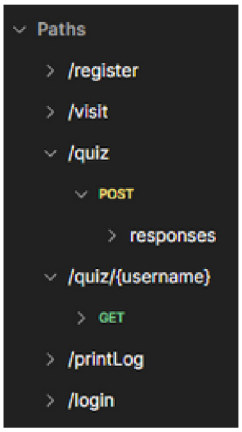




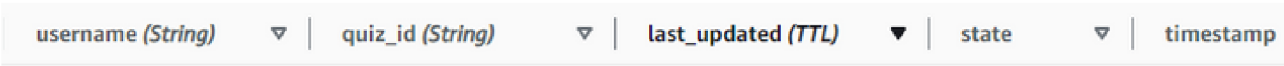
Training Quizzes Technology

- The quizzes are found in the Makerspace's Google Drive as Google Forms and they are all linked to a single Google Sheet.
- The Google Sheet has an App Script which will call the AWS API Gateway to trigger a lambda function. This will put the quiz data into the DynamoDb tables
- There is also the frontend hosted in S3 which makes a GET request which will go through API Gateway -> Lambda -> DyanamoDb then return the quiz progress.

API/Lambda



- We added 2 new API endpoints **/quiz** and **/quiz/:username.** /quiz has a POST request to insert quiz data and /quiz/:username is used for GET a specific users quiz progress
- Both of these endpoints trigger the same lambda function (**QuizFunction**) which handles both request. QuizFunction formats, inserts, and fetches data from our DynamoDb tables.



DynamoDb

- We added 2 new tables **quiz_list** and **quiz_progress**. quiz_list has PK quiz_id and stores all of the safety quizzes the Makerspace uses. quiz_progress (image above) holds all users quiz submissions and score is stored as state. State can be 1 or 0 (pass or fail).
- When we fetch a users quiz progression, we fetch all quizzes in quiz_list then look for any entires in quiz_progress for that user. If a user has no entires for a quiz_id we give that quiz a state of -1 (not attempted).

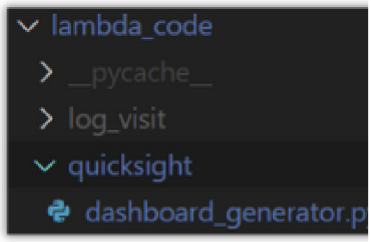
Testing

- There is automated testing for merging push requests (PR requests) on github with pytest
- This testing is also used for our AWS code pipeline, which has beta (additional testing) and prod (prod is the stage that code changes actually get pushed out and visible to the users)
- Note: the testing_script.py file contains all the test cases without function headers and is used by AWS pipeline to run test cases
- implemented test cases for get and post request for the lambda functions that pertains to quiz progression API request in the test_quiz.py file.

DEVELOPER GUIDE

DATA TEAM

Quicksight Dashboard Embedding (Part 1)



dashboard_generator.py

- Purpose: To generate a QuickSight dashboard embed URL.
- Environment: AWS Lambda.
- Key Services: AWS QuickSight, AWS Lambda, Boto3

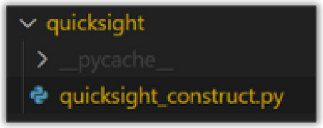
Key Components

1. Imports:
   - boto3: AWS SDK for Python, used to interact with AWS services.
   - os: Standard Python library to interact with the operating system, used here to access environment variables.
2. Constants:
   - HEADERS: A dictionary defining headers for the response, allowing cross-origin requests.
3. Lambda Handler Function (lambda_handler):
   - This is the main function that AWS Lambda invokes when the script is executed.

Workflow

1. Setting up QuickSight Client:
   - Initializes a QuickSight client using boto3.client('quicksight').
2. Retrieving Configuration:
   - Retrieves AWS Account ID, Dashboard ID, and User ARN from environment variables.
3. Generating Embed URL:
   - Calls quicksight_client.get_dashboard_embed_url() with the necessary parameters (account ID, dashboard ID, user ARN, etc.).
   - Configures session settings like SessionLifetimeInMinutes, UndoRedoDisabled, etc.
4. Response Handling:
   - Success: Returns a 200 status code with the dashboard embed URL.
   - QuickSightUserNotFoundException: Returns a 404 status code if the QuickSight user is not found.
   - Other Exceptions: Catches and returns a 500 status code for any other exceptions.

Quicksight Dashboard Embedding (Part 2)



quicksight_construct.py

- Purpose: To set up infrastructure for generating QuickSight dashboard embed URLs using AWS Lambda and API Gateway.
- Key AWS Services: AWS Lambda, AWS IAM, Amazon API Gateway, AWS QuickSight.

Key Components

1. QuickSightEmbedConstruct Class:
   - A construct class that defines the necessary AWS resources.

Workflow

1. IAM Role for Lambda:
   - Creates an IAM role (DashboardGeneratorRole) for the Lambda function.
   - Grants full access to AWS Lambda and specific permissions for AWS QuickSight actions.
2. Lambda Function Definition:
   - Defines the QuickSightDashboardGenerator Lambda function.
   - Specifies the runtime, handler, code location, associated role, and environment variables.
3. API Gateway Setup:
   - Defines the shared_api_gateway for generating QuickSight dashboard embed URLs.
4. Lambda Integration:
   - Integrates the Lambda function with the API Gateway.
5. Outputs:
   - Outputs the URL of the API Gateway.

Deployment Steps

1. Prepare the CDK Environment:
   - Make sure AWS CDK is installed and configured correctly.
2. Deploy the Construct:
   - Deploy this construct within an AWS CDK stack to create the resources in your AWS account.
3. Configure Environment Variables:
   - Set aws_account_id, dashboard_id, and quicksight_user_arn appropriately.

Best Practices

- Security: Ensure that the IAM role has the least privilege necessary for operation.
- CORS Configuration: Modify the CORS settings according to your specific domain to enhance security.

Future Considerations

- API Gateway Settings: Fine-tune API Gateway settings (like throttling and stages) as per the application requirements.

Quicksight Dashboard Embedding (Part 3)

quicksight_setup In api_gatway/_init_.py

- Purpose: To integrate AWS QuickSight dashboard embedding functionality into the Makerspace AWS CDK application.

Key Components of quicksight_setup

1. Integration of QuickSightEmbedConstruct:
   - The method instantiates the QuickSightEmbedConstruct defined in quicksight_construct.py.
   - This construct sets up the necessary resources for QuickSight dashboard embedding.
2. Parameters:
   - aws_account_id: Automatically filled with the AWS account ID.
   - dashboard_id: A hard-coded dashboard ID for QuickSight.
   - quicksight_user_arn: ARN for the QuickSight user, formatted with the AWS region and account ID.
   - shared_api_gateway: API Gateway resource
   - api_resource_name: Name of the resource to add as a root to the API Gateway.

```
def route_quicksight(self):
    resource_name = 'dashboard'

    self.quicksight = QuickSightEmbedConstruct(
        self,
        "QuickSightEmbedSetup",
        aws_account_id=core.Aws.ACCOUNT_ID,
        dashboard_id="b153dda9-d2f2-4829-9f5d-df80dadddda2d",
        quicksight_user_arn=f"arn:aws:quicksight:{core.Aws.REGION}:
        shared_api_gateway=self.api,
        api_resource_name=resource_name
    )
```

Workflow

1. Initialization:
   - The quicksight_setup method is a part of the MakerspaceStack.
2. QuickSight Embed Setup:
   - A new instance of QuickSightEmbedConstruct is created, passing relevant parameters like AWS account ID, dashboard ID, and QuickSight user ARN.

Quicksight Dashboard Embedding (Part 4)

handleDashboardClick in Admin.tsx

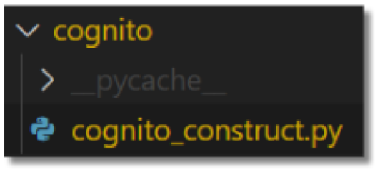
Overview

- Purpose: To fetch a QuickSight dashboard URL from an API Gateway endpoint and open it in a new tab.

Function Breakdown: handleButtonClick

1. Function Signature:
   - const handleButtonClick = async () => { ... };
   - Asynchronous function, implying it handles promises or awaits asynchronous operations.
2. Authentication Check:
   - Uses fetchAuthSession from AWS Amplify to check if the user is authenticated.
   - If the user is not authenticated, it displays an alert requiring sign-in.
3. API Request:
   - Makes a GET request to the API Gateway endpoint. The URL is hardcoded in this instance.
   - The Accept header is set to "text/plain", indicating the expected response type.
4. Response Handling:
   - Check if the HTTP response is OK (status 200).
   - If OK, alerts the user that the dashboard is being generated.
   - Extracts the response text (expected to be a URL) and checks if it includes "https".
5. Dashboard URL Opening:
   - If the response contains a valid URL (with "https"), it opens the URL in a new browser tab.
   - Handles any errors or non-URL responses by logging them to the console.

Cognito Authentication (Part 1)



CognitoConstruct

- **Purpose**: To define and set up an Amazon Cognito User Pool with specific configurations using AWS CDK.

Key Components

1. **Cognito User Pool**:
   - **self.user_pool**: Defines a new Cognito User Pool within the AWS environment.
   - Configuration options include:
     - **user_pool_name**: Custom name for the User Pool.
     - **self_sign_up_enabled**: Determines whether users can sign themselves up.
     - **sign_in_aliases**: Allows users to sign in using a username or email.
     - **auto_verify**: Automatically verifies users' email addresses.
     - **password_policy**: Sets requirements for user passwords.
     - **mfa**: Configures multi-factor authentication (MFA) as optional.
     - **mfa_second_factor**: Specifies the MFA methods available (SMS and OTP).
     - **account_recovery**: Sets the account recovery method to email.
2. **Cognito User Pool Client**:
   - **self.user_pool_client**: Creates a new client associated with the user pool, enabling applications to interact with the user pool.
3. **Outputs**:
   - Generates CloudFormation outputs for the User Pool ID and User Pool Client ID, making these IDs accessible after deployment.

Workflow

1. **Initialization**:
   - The constructor takes **scope**, **id**, and **user_pool_name** as parameters, along with additional optional keyword arguments.
2. **User Pool Creation**:
   - Instantiates a **cognito.UserPool** with various configurations based on the provided arguments and default settings.
3. **User Pool Client Creation**:
   - Adds a client to the user pool for application interactions.
4. **Outputs**:
   - Outputs the IDs for the created User Pool and User Pool Client for reference and use in other parts of the CDK application.

Cognito Authentication (Part 2)

cognito_setup In makerspace.py

- **Purpose**: To configure and deploy an Amazon Cognito User Pool for the Makerspace application.
- **Environment**: AWS CDK.
- **Key AWS Service**: Amazon Cognito.

```
def cognito_setup(self):

    self.cognito = CognitoConstruct(
        self,
        "CognitoTest",
        user_pool_name="MakerspaceTest"
    )
```

cognito_setup Method Breakdown

1. **Method Signature**:
   - **def cognito_setup(self):**
   - A method of the **MakerspaceStack** class, which is a subclass of **core.Stack**.
2. **Cognito Construct Initialization**:
   - Instantiates the **CognitoConstruct**, a custom construct for creating a Cognito User Pool.
   - **self.cognito = CognitoConstruct(...)**: The construct is initialized with the current stack (**self**), an identifier ("**CognitoTest**"), and the name of the user pool ("**MakerspaceAuth**").

Workflow

1. **Stack Initialization**:
   - Within the **MakerspaceStack** class, various components of the application are initialized, including database, API Gateway, DNS records, and others.
2. **Cognito User Pool Creation**:
   - **cognito_setup** is invoked to set up the Cognito User Pool using the **CognitoConstruct**.
   - The user pool is named "**MakerspaceAuth**", which can be customized based on application requirements.
   - The creation of users within the user pool can be done via AWS CLI or the AWS Cognito Console.

Cognito Authentication (Part 3)

Amplify Cognito Setup in App.tsx

- Purpose: To configure AWS Amplify to integrate with an Amazon Cognito User Pool for user authentication in a React application.
- Key Technologies: AWS Amplify, Amazon Cognito, React.

Configuration Breakdown

1. Amplify Configuration:
   - Amplify.configure({...}): This function call configures Amplify with the provided settings.
   - The configuration object contains the Auth key, which is specifically for authentication settings.
2. Cognito Settings:
   - userPoolId: Specifies the Amazon Cognito User Pool ID. It should match the ID of the User Pool created in the AWS environment.
   - userPoolClientId: Refers to the Web Client ID associated with the Cognito User Pool. It's used for client-side interactions with Cognito.

Amplify and Cognito Integration in the App

1. Import Statements:
   - import { Amplify } from 'aws-amplify';: Imports the Amplify library to use in the application.
2. Application Routing:
   - The app uses BrowserRouter for client-side routing. Various routes are defined for different components like LocationSelection, Registration, Admin, etc.
3. Use of Amplify.configure:
   - Called at the top-level of the application file, ensuring that the Cognito configuration is set before any component uses authentication features.

```
Amplify.configure({
  Auth: {
    Cognito: {
      //  Amazon Cognito User Pool ID
      userPoolId: "us-east-1_oqeB8UF90",
      // Amazon Cognito Web Client ID
      userPoolClientId: "12aipeh8hrte44saghfbbacn72",
    },
  },
});
```

USER GUIDE

OCTOPRINT TEAM


Octoprint-Cura Usage

- The built-in octoprint dashboard should take care of most of the work for you
- When starting a print:
  - Upload the STL or other 3D file type to Cura
  - Press the slice button in the bottom right corner
  - After the slice completes, click ‘Send to Octoprint
  - The file should upload through Octoprint and to the printer and begin printing
- If for any reason the print needs to be stopped:
  - Click cancel in the bottom right and the print will stop
  - The dashboard available from cura will allow you to move the extruder as you see fit
- With multiple printers set up and connected:
  - At the top right, click on the bar with the printer name and swap to a different printer
  - All connected printers will be available to view the current status of and to send prints to

Octoprint-Cura Maintenence

- Issues may arise when trying to access octoprint or when trying to send a print from Cura via Octoprint
- If Octoprint cannot be accessed or Cura cannot connect to Octoprint:
  - Check that the Raspberry Pi for the associated printer is connected to both a power source and the router being used.
  - Ensure the Octoprint is properly connected to Cura via an API Key
  - Ensure there is no active print then try removing the Raspberry Pi from power and reconnecting it after a short period of time
  - If none of this works, attempt to reimage the Rasperry Pi following the instructions listed in the developer guide
  - If this does not work, the SD Card in the Raspberry Pi or the Raspberry Pi itself might need to be replaced

USER GUIDE

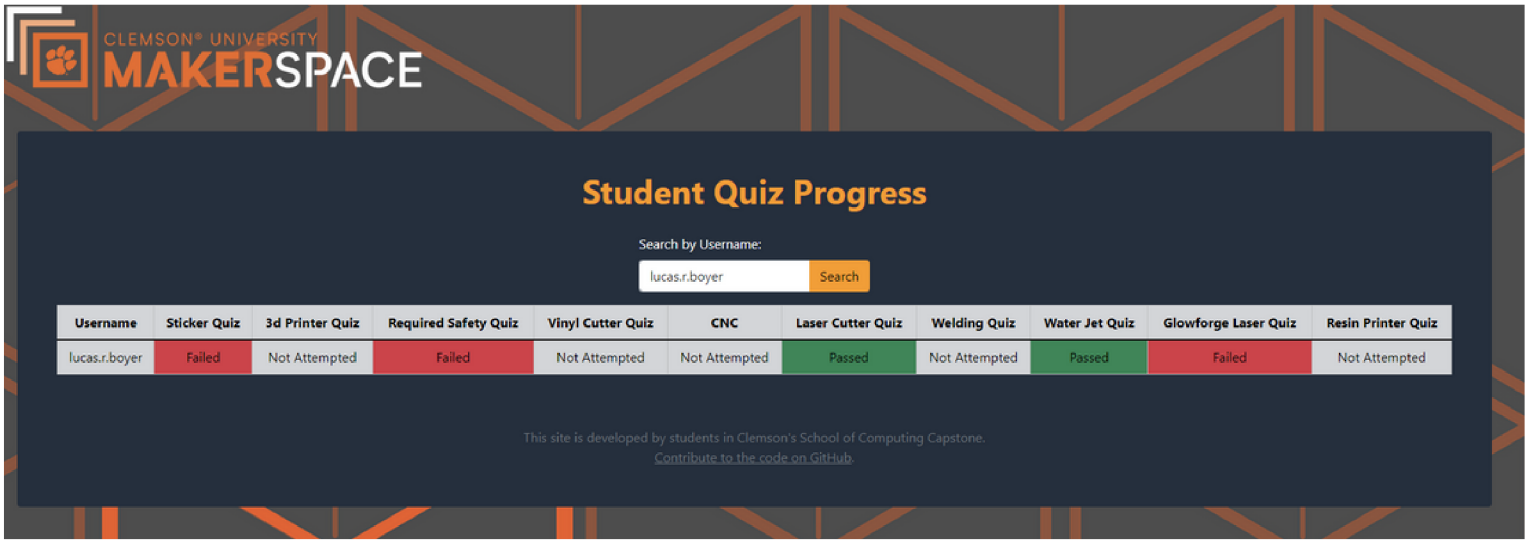
DATA TEAM



Training Quizzes Progress Access

- For Makerspace staff to access user quiz data:

1. Navigate to https://www.cumaker.space/
2. Click on the admin button
3. Use admin login credentials
4. Once authenticated, look up student by username (ex: nayhah@clemson.edu username would be nayhah)
5. You should see this



Training Quiz Student Access

- Students will access the Google Forms by

1. Going to https://www.cumaker.space
2. Hover over the equipment training
3. Choosing a tool and clicking on the quiz
4. Then filling out the Google Form and submitting

STEPS TO CREATE A NEW MAKERSPACE QUIZ AND INTEGRATE WITH AWS

1. CREATE A NEW GOOGLE FORM
 1. Create a new google form (In the Makerspace folder Equipment Quizzes)
 2. Make sure that the google form
  - is a quiz
  - all questions have a right answer
  - Form defaults -> Collect email addresses by default -> Responder input
  - Responses -> Collect email addresses -> Responder input

2. LINK IT TO GOOGLE SHEETS
 1. Click Responses -> Link to sheets -> Select existing spreadsheet -> Select
 2. Then select the "Makerspace User Trainings" Google Sheet
 3. At the bottom find "# From Responses" and rename the sheet to the name of the quiz

WHAT DOES THIS DO:
Now whenever a student fills out the google form and submits you should be able to find them by username in the Student Quiz Progress in the Admin Portal
You can see if a student has passed, failed, or not attempted all quizzes

DEBUG:
If a quiz doesn't work
- Change the title of the google form then change it back (this will update the metadata of the form)

If a quiz stops working to see errors
- Go to the "Makerspace User Trainings" Google Sheet
 - Then go to the top and click "Extensions" -> Apps Script -> Executions tab
- Or check AWS

USER GUIDE

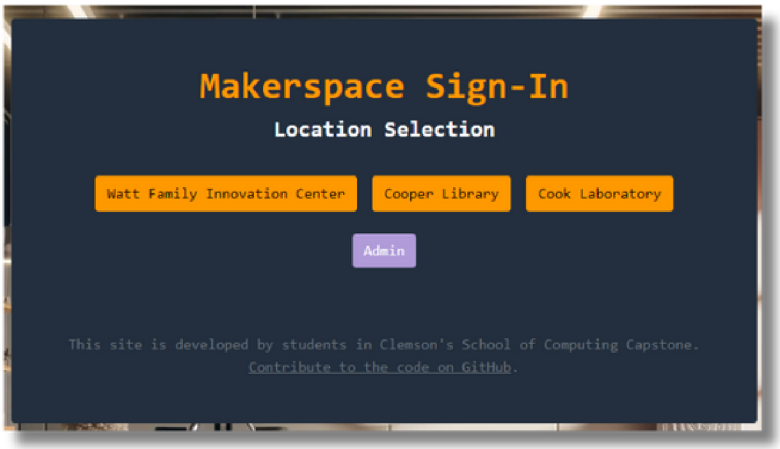
DATA TEAM

Quicksight Dashboard Embedding

- Access the website: visit.cumaker.space.
- Sign in using Cognito authentication.
- After successful sign-in, you'll be redirected to the admin analytics page.
- Click on the "Dashboard" button to generate a Quicksight dashboard link.
- The generated dashboard link will open in a new page, displaying an interactive Quicksight dashboard.

Cognito Authentication

- Access the website: visit.cumaker.space.
- Sign in using Cognito authentication via the admin button.



CONTACT

CAPSTONE TEAM

- Sol Lesesne - soll@clemson.edu
- CJ Boni- cjboni@g.clemson.edu
- Lucas Boyer- lrboyer@clemson.edu
- Nayha Hussain- nayhah@g.clemson.edu
- Susan Li- sli4@g.clemson.edu
- Sumanth Pandiri- spandir@g.clemson.edu

AWS

- Dr. Richard Weatherly- awsrmw@amazon.com
- Owen Phillips- oap@amazon.com
- Dillon Gorsen- dggorsen@amazon.com
- Joshua Little- jplittle@amazon.com

MAKERSPACE

- Dr. Todd Schweisinger- todds@clemson.edu
- Gavin Mcswain- grmcswa@clemson.edu