Project – Build Your Website in the AWS cloud

Sample projects: https://meijuan-long.click/, https://sdbappi.com/

Benefits of the project

1. You will build out your own website that impresses people and recruiters. You will outstand from other candidates during the job search.
2. Gain hands-on experience with modern cloud technologies. I assure the technologies you learned in this class help you shine during the job interview and at work.

Goals

1. Build a fault-tolerant, highly scalable, production-ready API on the cloud.
2. Create a front-end app that utilizes the API – It will help you understand how back-end and front-end developers build a whole application.

Tasks

1. Build the front-end using React, Angular, or any front-end libraries and framework. Or you can use templates online. Deploy it in S3. Put CloudFront in front of the S3 bucket.
2. Build the back-end API using Lambda, DynamoDB, API Gateway, SNS, S3, and SQS.
3. Put the **CloudFront** in front of the S3 that hosts your front-end. Buy a domain name for yourself and configure that in **Route53**. You can get free certificate in **ACM** (Amazon Certificate Manager).
4. Research a service in AWS that we didn't learn in class such as CodeDeploy, ElasticBeanstalk, Glue/Athena, Systems manager etc. Write a blog on your website about it. You can use AWS Academy. Serverless services cost less whereas services that require instances cost much. The blog content must all be in your own words. The content will be similar to a lesson that I teach. It must have what that service is? What it does do? Why do you need it? When to use it? the service features etc. There must a hands-on as well. https://meijuan-long.click/blog-details.html and https://medium.com/awesome-cloud/aws-transit-gateway-overview-introduction-to-aws-transit-gateway-tgw-getting-started-guide-9990fd0aaebb and https://medium.com/@taimurcloud123/amazon-guardduty-why-it-is-a-must-for-aws-security-b5ece190c9c5 Topics could be OpenSearch, CodeWhisperer, ECS, one of business applications services.

ALWAYS FOLLOW THE LEAST PRIVILEGES PRINCIPLE.

API requirement

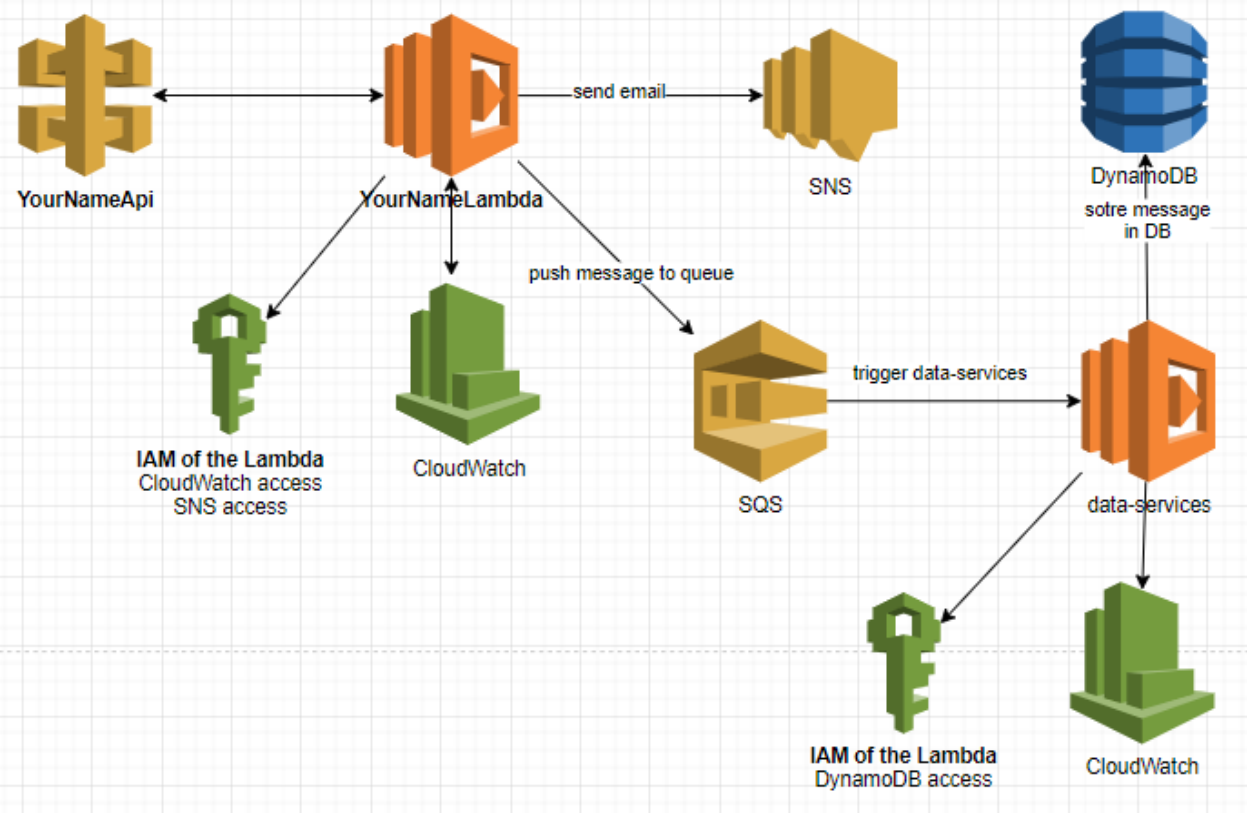
Feel free to come up with your own back-end app or stack.

Do something interesting and extra. Such as writing front-end from scratch yourself, dynamic blogging, visitor name registration in DB. For example, https://meijuan-long.click/latestblog/latest.html

- **YournameLambda** – Your back-end code. That logs out info and error to CloudWatch. Sends the message from a guest to SNS and SQS. The SNS sends you an email about what the guest messaged.
- **SQS –** receive a message from YournameLambda and the DataserviceLambda will pull the message.

- **DataserviceLambda –** Gets a message from the queue and stores it in DynamoDB. It gets triggered by the SQS queue.
- **API Gateway** – It is your API. Behind it, your back-end application code (lambda) runs. A custom domain can be created here if you configure a domain name for your web.
    - Resource name: contact
    - Method: POST
- **CloudWatch** – Stores log data
- **DynamoDB** – NoSQL database for your app.
    - MessageTitle [partition_key]
    - Message
    - Email
    - GuestName
    - Phone [optional]
- **SNS** – Sends email to you.
    - Send the email to you. It includes all data above.
- **S3** – Hosts your front-end
- Utilize **CloudFront** on top of the React front-end website hosted on S3.
- Utilize **Route53** and **ACM.** – Have your own domain (i.e. yourname.com). Run the API and S3 app behind your domain name.

Decoupled application with AWS SQS. There is a Lambda named “DataServices” for storing data in the database. Real-life applications utilize this architecture as it is much faster by providing async data storage and fault-tolerant by storing messages in a queue.



It will cost less than $3 a month to host your website using S3, API Gateway, Lambda, DynamoDB, and SNS. Delete the DynamoDB functionality after the course which will reduce the price. Just pass the message to your email via SNS.

Front-end requirements

Web sections:

- Intro. One statement about yourself.
- About. More detailed. 2 or 3 paragraphs
- Projects you did at MIU or at work.
- Skills. List technologies you know.
- Education
- Testimonial
- Blog. At least one blog about a service that you learn by researching yourself and is not covered in this class.
- Contact me section. It will call the back-end API.

Acceptance criteria:

- The blog about the service you explored.
- Show the architecture diagram on your website.
- It should print a success or error message when sending a message.
- Hide the button after successful submission or add a captcha.
- Phone number is optional. There should be client-side validation.

- Don’t show your DOB or age.

Your front-end app will have a form that has 5 fields:

- MessageTitle
- Message
- Email
- GuestName
- Phone [Optional]

Other info:

- You find front-end templates online. Or learn React or write from scratch I will consider as extra.
- Use the Axios library to call the back-end API.
- When deploying the React app to S3, don’t forget to define **index.html** as **Error Document**.

Steps by step instructions

1. Create an SQS topic that sends you an email. Create a subscriber which is your email.
2. Create a DynamoDB. Make sure that your Max Read/Write Capacity Unit is 2. So it will cost less.
3. Create Lambda. Log out of the **event.** Create environment variables for Topic ARN and DynamoDB table names. Declare global variables for AWS SDK, SNS, DynamoDB, TableName, and TopicArn. Refer to the sample code below.
4. Update Lambda Execution Role for DynamoDB and SNS access.
5. Create an API on the API gateway. Create a resource **contact** and a **POST** method under it. Associate the method with the Lambda
6. Copy the **event** in the log and create a **test** event in Lambda. That will help you debug your code.
7. Write code.
    a. Error handling for system error (500), invalid requests (400) such as invalid HTTP endpoint, and resources etc.
    b. Publish a message to SNS.
    c. Put an item in DynamoDB.
    d. Send CORS headers in the response.
8. Test code with postman. (End to End test)
9. Create the front-end app that has 5 fields.
    a. MessageTitle [partition_key]
    b. Message
    c. Email
    d. GuestName
    e. Phone [optional]
10. Call the back-end API from the front-end react app via Axios.
11. Build your front-end app.
12. Create an AWS S3 bucket with name i.e., yourname.com. And upload your build files there. Refer the S3 Lab if need.

Domain name

- Buy your domain.

- Create a hosted zone on AWS Route 53. For example, yourname.com
- Copy the NS record values and create an NS record on the domain name providers website. You can contact them to configure that. Or maybe they will contact you to confirm. What you are doing is that, that domain name will be managed on AWS Route 53 rather than the domain name provider. So that you can manage all your DNS records on AWS, create Alias records for your website etc.
- Create certificate for your website with Amazon Certificate Manager. You will use that when creating a CloudFront distribution in the next step. Refer: https://docs.aws.amazon.com/acm/latest/userguide/gs-acm-request-public.html
    - Make sure you selected DNS validation. Once you created the certificate on AWS ACM, it will generate a CNAME record. That you will register in your hosted zone.
- Create an AWS CloudFront distribution for the S3 bucket. That will distribute your code all over the world. Refer: https://aws.amazon.com/cloudfront/getting-started/S3/.
    - Select the issued certificate when creating the CloudFront distribution.
- Create a DNS record for the CloudFront distribution on Route 53. Refer: https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-to-cloudfront-distribution.html
- [Optional] – Now your website is accessible from both S3 and CloudFront. The best practice is access only through CloudFront. Refer this: https://aws.amazon.com/premiumsupport/knowledge-center/cloudfront-access-to-amazon-s3/

Sample Back-end code

```
const AWS = require("aws-sdk");
const sns = new AWS.SNS({apiVersion: '2010-03-31'});
const dynamodb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });
const tableName = process.env.COURSE_TABLE;
const topicArn = process.env.TOPIC_ARN;

exports.handler = async (event) => {
    console.log("New message from a visitor: " + JSON.stringify(event));

    const invalidReq = {
           statusCode: 400,
        headers: {
            "Access-Control-Allow-Origin" : "*",
            "Access-Control-Allow-Credentials" : true
        },
           body: JSON.stringify('Invalid request')
       };

    if (event.httpMethod === "POST") {
        if (event.path === "/contact") {
            try {
                const body = JSON.parse(event.body);

                if (!body || !body.Email || !body.GuestName || !body.Message) {
                    console.log("Invalid request - Required params are missing.");
                    return invalidReq;
```

```
}

let emailSubject = `${body.GuestName} messaged you on unubold.com!`;

const snsParams = {
    Message: JSON.stringify({
        subject: body.MessageTitle,
        email: body.Email,
        phone: body.Phone,
        message: body.Message
    }),
    Subject: emailSubject,
    TopicArn: topicArn
};

const res = await sns.publish(snsParams).promise();

console.log("Successfully sent email: " + JSON.stringify(res));

// Saving a message in the DB
const saveParams = {
    TableName: tableName,
    Item: {
        "MessageTitle": {
            S: body.MessageTitle
        },
        "Email": {
            S: body.Email
        },
        "Phone": {
            S: body.Phone
        },
        "Message": {
            S: body.Message
        },
        "GuestName": {
            S: body.GuestName
        }
    }
};

await dynamodb.putItem(saveParams).promise();

return {
    statusCode: 200,
    headers: {
        "Access-Control-Allow-Origin" : "*",
        "Access-Control-Allow-Credentials" : true
    },
    body: JSON.stringify('Success')
};
```

```
            } catch (err) {
                const msg = "System error while sending SNS or saving DynamoDB";

                console.log(msg + ": " + JSON.stringify(err));

                return {
                    statusCode: 500,
                    headers: {
                        "Access-Control-Allow-Origin" : "*",
                        "Access-Control-Allow-Credentials" : true
                    },
                    body: JSON.stringify(msg)
                };
            }

        } else {
            console.log("Invalid request - Wrong endpoint.");
            return invalidReq;
        }

    } else {
        console.log("Invalid request - Wrong http method.");
        return invalidReq;
    }
};
```

Sample Front-end code

```
import React, { Component } from 'react';
import axios from 'axios';

class App extends Component {
      state = {
            data: {
                  email: '',
                  name: '',
                  phone: '',
                  subject: '',
                  message: ''
            },
    submitted: false
      };

      onChange = (e) => {
            return this.setState({
                  data: { ...this.state.data, [e.target.name]: e.target.value }
            });
      };

      onSubmit = () => {
            console.log(JSON.stringify(this.state.data));
    const correctData = {};
```

```
correctData.GuestName = this.state.data.name;
correctData.Email = this.state.data.email;
correctData.Phone = this.state.data.phone;
correctData.MessageTitle = this.state.data.subject;
correctData.Message = this.state.data.message;

axios.post(
  'https://yvzvk41grl.execute-api.us-east-1.amazonaws.com/v1/contact',
  correctData
).then(
  (response) => {
    console.log("Response: " + JSON.stringify(response));
  },
  (error) => {
    console.log("Error " + error);
  }
);
   };

   render() {
          return (
  <form id="contactForm">
    <input
      type="text"
      id="name"
      name="name"
      className="form-control"
      placeholder="Your Name*"
      onChange={this.onChange}
    />
    <input
      type="email"
      className="form-control"
      id="email"
      name="email"
      placeholder="Your Email*"
      onChange={this.onChange}
    />
    <input
      type="text"
      id="subject"
      name="subject"
      className="form-control"
      placeholder="Subject*"
      onChange={this.onChange}
    />
    <input
      type="text"
      className="form-control"
      id="phone"
      name="phone"
```

```
          placeholder="Phone"
          onChange={this.onChange}
        />
        <textarea
          name="message"
          id="message"
          className="form-control"
          rows="6"
          placeholder="Your Message* ..."
          onChange={this.onChange}
        ></textarea>
        {!this.state.submitted && (
          <button
            type="button"
            onClick={this.onSubmit}
            disabled={this.state.loading}
            className="btn send_btn theme_btn"
          >
            Send A Secure Message
          </button>
        )}
      </form>
    )
      }
}

export default App;
```

Breakdown

Project score breakdown:

- Domain name and certificate - 3 point
    - Buy a domain for example, GoDaddy.
    - Create a hosted zone for your domain in Route 53.
    - Copy 4 NS records and paste it in the domain name provider. So you can manage your domain name in AWS. If the domain name provider accepts 2 NS records, put 2 of them then.
    - Create an A record. Toggle Alias. Then select the CloudFront distribution.
    - Go to ACM (Amazon Certificate Manager) and hit the “request a certificate”.
    - Qualified domain names are
        - yourname.com
        - *. yourname.com
    - After request has been created, click on that and click on “create records in Route 53”.
- CloudFront – 2 points
    - Select the bucket in the origin domain.
    - Select OPTIONS and all other HTTP methods in Allowed HTTP methods.
    - [Optional] Enter domain name in Alternate domain name (CNAME).

    - [Optional] Select the certificate in Custom SSL certificate - If you have the certificate issued by ACM.
    - Enter index.html as Default root object.
- Client side validation and success/error messages - 2 points
- Front-end – 2 points
- Back-end – 4 points
- Blog – 4 points
- Extra – 3 points