AWS Cloud Project Bootcamp Outline

Organized by: AWS Ontario Virtual User Group

*Registration is **CLOSED***
*To join the bootcamp you need to register via the **Offical Registration Form***

Related Resources:

- **AWS Cloud Project Bootcamp Outline**
- **AWS Cloud Project Bootcamp FAQs**
- **AWS Cloud Project Bootcamp Grading Rubric**
- **AWS Cloud Project Bootcamp Surveys**
- **AWS Cloud Project Bootcamp Codes of Conduct**
- **AWS Cloud Project Bootcamp Sponsorship Package**
- **AWS Ontario Virtual User Group Meetup Page**

Before asking, READ THE FAQS.

Please ***don't comment spelling or grammar mistakes***,
I will run this document through Grammarly, and have it proofread.

Bootcamp Official Schedule

This is the official schedule for all events in the bootcamp

Type	Name	Datetime
Lab (Live)	Week 0 — Billing and Architecture **Maragret & Chris**	Feb 11 12PM (Noon) ET
Lab (Live)	Week 1 — App Containerization **Edi & James**	Feb 18 12PM (Noon) ET
Lab (Live)	Week 2 — Distributed Tracing **Jessica**	Feb 25 12PM (Noon) ET
Lab (Live)	Week 3 — Decentralized Authentication	Mar 4 12PM (Noon) ET
Lab (Live)	Week 4 — Postgres and RDS	Mar 11 12PM (Noon) ET
Lab (Live)	Week 5 — DynamoDB and Serverless Caching **Kirk and Alex**	Mar 18 12PM (Noon) ET

Lab (Live)	Week 6 — Deploying “Serverless” Containers	Mar 25 12PM (Noon) ET
Lab (Live)	Week 7 — Solving CORS with a Load Balancer and Custom Domain	Apr 1 12PM (Noon) ET
Lab (Live)	Week 8 — Serverless Image Processing **Kristi**	Apr 8 12PM (Noon) ET
LabLab (Live) (Live)	Week 9 — CI/CD with CodePipeline, CodeBuild and CodeDeploy **Du’an**	Apr 15 12PM (Noon) ET
Lab (Live)	Week 10 — CloudFormation Part 1 **Rohini**	Apr 22 12PM (Noon) ET
Lab (Live)	Week 11 — CloudFormation Part 2	Apr 29 12PM (Noon) ET
Lab (Live)	Week 12 — Modern APIs **Michael**	May 6 12PM (Noon) ET
Lab (Live)	Week 13	

Bootcamp Goal

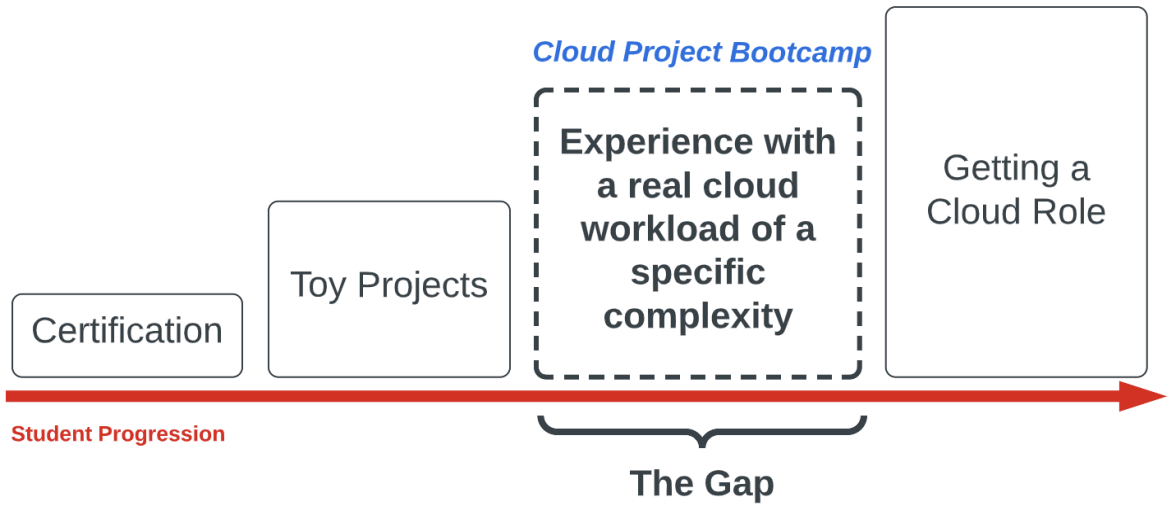
Target Audience
The goal of this bootcamp is to help students who have acquired associate-level knowledge and are at a point where they realize they need a cloud project on their resume, to progress their career goals.

The Problem
The target audience is those who are not sure how to:

- build a cloud project with enough complexity to merit worthiness on their resume
- combine multiple cloud services to emulate a real-world production workload

This bootcamp is rated as **Level 250** based on the following difficulty scale:

- Level **100** Foundational
- Level **200** Intermediate
    - Level **250** [AWS Cloud Project Bootcamp]
- Level **300** Advanced
- Level **400** Expert



Project Scenario

A startup company has decided to build their own micro-blogging platform and has hired you to be its first cloud engineer.

The company paid a web-development firm to translate their wireframe designs into a mock web-application for the purpose of demoing to raise capital.

After a successful round of funding, you [the cloud engineer] have been tasked with taking the mock web-application and making it production ready at scale.

The startup company consulted a fractional CTO to help choose some of the technical requirements to place the company on a good technical roadmap:

- The frontend application should be written in Javascript using React (functional components).
- The backend application should be written in Python using Flask
    - Since we plan to be API only
    - Since we want to choose a popular framework API only framework
    - Since we want a micro-framework because we are offloading as much to the cloud as possible to avoid be a monolithic application
    - Since we don't want an ORM because the CTO considers it an anti-pattern
    - Since Python is the most popular language being learned for cloud right now
- That an API specification be defined detailing the exact endpoints required.
- The web application:
    - Shall be deployed to AWS.
    - Takes advantage of modern-applications cloud services.

The startup company has spent the majority of their funding on hiring you for the next 6 months (but mostly spent the money on marketing and buying a really cool domain) and so you also need to ensure you keep the cloud provider costs as low as possible.

Good Luck! 😂

Learning Outcomes

- To learn how to containerize a frontend and backend web application
    - ***eg. Docker***
- To learn how to work with multiple serverless containers
    - ***eg. AWS Fargate, Amazon ECR***
- To learn how to abstract API call with GraphQL
    - ***eg. AWS AppSync***
- To learn basic data modeling for NoSQL databases
    - ***eg. Amazon DynamoDB***
- To learn basic data modeling for SQL databases
    - ***eg. Amazon RDS***
- To learn aside-caching to improve database performance
    - ***eg. Momento***
- To learn how to set up a CI/CD pipeline
    - **eg. CodeBuild, CodeDeploy, CodePipeline**
- To learn how to use Infrastructure as Code (IaC)
    - **eg. CloudFormation**
- To learn how to offload background jobs to serverless functions
- To learn how to setup hosted zones
    - **Route53**

Architecture Description

This is a description of the AWS architecture in bullet form, it does not describe every AWS resource as its implied in some cases they are being used, we want to generate out code for the AWS CLI in the previous code format as a single file.

- Resources generally will run in a preferred region based on the student's choice, however some services will run in other AWS regions based on AWS's limitations or restrictions.
- A custom VPC with three public subnets that route on to the internet, all AWS resources attempt to use this custom VPC
- An ECS Fargate cluster running a single service for the backend flask application
- The backend flask application container image is hosted in a private ECR repo
- The target services are using ECS Connect
- A custom domain name that is being managed by Route53
- An internet facing application load balancer is serving the backend flask application a subnet for the custom domain of api. The backend flask application runs on port 4567
- The target group for the backend flask application is performing health checks
- A CodePipeline for deploying code changes that is sourcing from Github
- The CodePipeline has a build step using CodeBuild to build a docker image and push it our private ECR repo for the backend-flask application container image

- An RDS Postgres server that is internet available in public subnets which is used by the backend flask application
- The CodePipline deploy steps uses ECS deployment not CodeDeploy
- A DynamoDB table that has a DynamoDB stream which triggers a lambda to write updates to the same DynamoDB table
- A Cognito User Pool used for authentication in our web-application, we have a post-configuration lambda webhook that inserts a new user into our RDS Postgres database
- We have a frontend react static website hosted on in an S3 bucket which is only accessible via a CloudFront distribution which redirects to HTTPS.
- The CloudFront distribution for the static website hosting is for both the www. And naked domain
- The application load balancer, and cloudfront distributions all use HTTPS and is using a public certificate generated by Amazon Certification Manager
- Our application has profile photos which are uploaded client side, we do this by having an HTTP API Gateway endpoint which will point to a lambda that will generate out a presigned URL to bucket that will raw assets
- Our HTTP API Gateway uses a Lambda authorization for our Cognito User Pool
- In our HTTP API Gateway we use a proxy to a custom lambda to handle the CORS preflight check
- When new files are uploaded to the bucket holding raw assets this will trigger via S3 Event Notifications a lambda which will process the images into thumbnails and it will output the the images into another bucket which will called the assets buckets
- This assets bucket is only accessible via a cloudfront distribution that will server assets from the assets. subdomain
- Route53 is pointing the naked domain and www to the cloudfront distribution that servers that static frontend react application, where the api. Is serving the backend flask application where as the assets. Subdomain is serving the cloudfront distribution for the graphical assets

Individual Project Requirements

All Students (even if you're in a team) must Create a new repository from this template

- https://github.com/ExamProCo/aws-bootcamp-cruddur-2023

The repo must exactly match the following name

- aws-bootcamp-cruddur-2023

The repo must be set to public

You need to provide the Github URL to the Individual Project in the student portal

Why do I need an individual repo if I'm working in a team?

The purpose of teams is to pool your efforts to complete homework, not the baseline instructional content.

Team Project Requirements [Scraped]

Students participating in a team must Create a new repository from this template

- https://github.com/ExamProCo/aws-bootcamp-cruddur-2023

The repo must exactly match the following name:

- aws-bootcamp-cruddur-2023

The repo should exist in a new Public Github Organization will all team members added.
The repo must be set to public
You need to provide the Github URL to the Team Project in the student portal

Student Evaluation and Final Grading

A grading rubric has been created along with the digital badges which could be earned.

- **AWS Cloud Project Bootcamp Grading Rubric**

These are the following components:

Instructional Completion

Homework Completion

Cloud Career Quiz

Pricing Quiz

We'll have weekly prerecorded sessions covering pricing for the last week of class.
You'll watch the video and complete a quiz with a passing grade on the first attempt.

Security Quiz

Cloud Technical Essay

Write a long-form essay published to LinkedIn Article, DEV article, Medium article or your own self-hosted website that demonstrates the depth of knowledge gained and to show good documentation and communication skills.

A technical article could be:

- Expanding on a homework assignment
- Applied technology outside the scope of the bootcamp to the project
- Writing about your challenges of upskilling on a specific cloud service
- Expanding on the theoretical architecture by solution architecting systems that don't exist

Project Description

Pitching Cruddur to Customers
Introducing a new micro-blogging platform that emphasizes privacy and the present moment.

Our platform allows users to post updates, thoughts, and photos that automatically expire after a period of time, ensuring that your personal information and conversations stay relevant and in the moment.

Perfect for busy professionals, students and anyone who wants to stay connected without the pressure of maintaining a permanent online presence.

Sign up now and experience the *freedom* of ephemeral social media.

Pitching Cruddur to Investors

What are the largest hurdles for long-term success and largest liability for social media platforms? **Trust and Safety Issues**

A micro-blogging platform with expiring posts can help reduce trust and safety issues that are common on most social media platforms in several ways:

- Reduces the amount of personal information available online: By having posts automatically expire, users are less likely to share sensitive personal information that could be used for identity theft or other malicious purposes.
- Decreases the potential for cyberbullying and harassment: With ephemeral content, there is less of a chance for negative or harmful posts to be saved and shared, reducing the potential for bullying and harassment.
- Improves the overall user experience: By allowing users to focus on the present moment, rather than worrying about the long-term impact of their posts, the platform can provide a more positive and enjoyable experience for users.
- Increases the sense of community: By focusing on the present, the platform's users are more likely to engage in conversations and build relationships, rather than just passively scrolling through old content.
- Increases trust and safety: With less personal information and harmful content available, users will feel more secure in using the platform and be more likely to trust the platform with their personal information.

In summary, a micro-blogging platform with expiring posts can:

- reduce trust and safety issues by limiting the amount of personal information online,
- decreasing the potential for cyberbullying and harassment,
- and improving the overall user experience.

Additionally, it will increase sense of community and increase trust and safety among users.

What is the hardest challenge for micro-blogging platforms? **Monetizing their platform** for long terms sustainability and a strong exit strategy.

A micro-blogging platform with expiring posts can be great for monetization in several ways:

- Increases user engagement: With expiring posts, users are more likely to be active and engaged on the platform, as they have a sense of urgency to view and interact with content before it disappears. This increased engagement can lead to more opportunities for monetization, such as through advertising or sponsored content.
- Encourages user-generated content: By limiting the lifespan of posts, the platform can create an environment where users are more likely to create and share new content, rather than just scrolling through old posts. This user-generated content can be monetized through advertising or sponsored content.
- Creates a sense of exclusivity: Expiring posts can create a sense of exclusivity and scarcity, making users more likely to engage with the platform and view sponsored content.
- Enhances the platform's ability to target ads: An ephemeral platform can enhance the platform's ability to target ads, as the platform can gather data on users' interests, preferences, and conversations that are currently relevant. This allows for more effective and relevant targeting of ads, which can increase the value of the platform for advertisers.
- Provides opportunities for in-platform transactions: An ephemeral platform allows for the platform to enable in-platform transactions, such as purchasing access to certain content or exclusive feature, which can be monetized.

In summary, a micro-blogging platform with expiring posts can be great for monetization as it:

- increases user engagement,
- encourages user-generated content,
- creates a sense of exclusivity,
- enhances the platform's ability to target ads,
- and provides opportunities for in-platform transactions.

All of these features can increase the value of the platform for advertisers and lead to more revenue opportunities.

How is Cruddur different from to existing competitors:

- Snapchat has ephemeral features but is focused on being private 1-to-1 or group chat. Snapchat lost trust with users because their private conversations turned

out to not be ephemeral. Cruddur is focused on being a public-facing micro blogging platform similar to Twitter

- Twitter is a micro-blogging platform but ephemeral posts have to be implemented through third-party applications and it not how Twitter is used. Twitter had Twittter Stories but removed the feature since it was a bolted on feature as opposed to the whole platform being ephemeral-first.
- Instagram is a platform focused around photo-sharing and while they do have Instagram Stories, an ephemeral feature for posts to expire, its not the primary way of interacting on the platform.

Cruddur will succeed by being a **true ephemeral-first** micro-blogging platform.

Time Commitment

Let’s fully detail the time a student will need to commit to fully experience this bootcamp over the 10 weeks:

Task	Estimated Time
Prerequisite Knowledge This is strongly recommended knowledge you will want to obtain prior to the course. The time commitment can greatly vary, so we’ll provide an average amount of time committed	10 hours
Prerequisite Technologies You need to register specific cloud service accounts. This needs to be performed before the course starts	1-2 hour
Classroom time Each class is time-blocked for 2 hours. Let’s also assume you might want to watch back the video	2-3 hours per week 13 weeks = 26-39 hours
Homework time Homework is not necessary to complete, you will be provided multiple challenges to perform on your own. It's your decision to decide how much time you wish to commit to homework.	2-10 hours per week 13 weeks = 26-130 hours
Student Discussion In the Discord and email, we’ll send simple polls. We’ll have office hours where you can optionally attend.	1 hour per week 13 weeks = 13 hours
***Per Month Commitment**	25-64 hours
Total Time Commitment	76-194 hours

Prerequisite Knowledge

Expectations
The following should be completed before you attend the course.

Complete as many of the below steps as possible to make the bootcamp process smoother, that way you can spend less time worrying about your set-up, and more time doing actual learning.

[1] Cloud Knowledge
The student is expected to have either obtained the knowledge of the Certified Cloud Practitioner or general cloud knowledge from another cloud service provider.

The student is expected to be comfortable navigating around the AWS Console, eg.

- Changing regions
- Using the search bar to find a service

If you do not have this knowledge, please watch: AWS CLF-C01 (Youtube)

[2] Programming Knowledge
The student is expected to be able to perform basic programming tasks…
eg. functions, variables, loops, classes, conditionals, imports, some complex data types

The student is not expected to have deep knowledge of web-frameworks

The programming languages that will be used:

- Python 3
- Javascript ECMAScript 6 (ES6)
- YAML and JSON files

If you do not have this knowledge or wish to prepare further, please watch:

- *Python for Beginners (Youtube)*
- *Javascript Full Course for Beginners (Youtube)*

[3] CodeEditor Knowledge
The student is expected to be comfortable working in the Visual Studio Code Editor (VSCode) with common editor actions eg.

- Committing code to a linked git repository
- Working with indentation level
- Installing VSCode extensions

- Using find, search or replace
- Copying and pasting code
- Basic usage of the bash terminal

The student does not require Gitpod knowledge prior, but there is a linked free course if you want to increase your knowledge prior to the bootcamp.

If you do not have this knowledge, please watch:

- *VSCode Crash Course (Youtube)*
- *Gitpod Crash Course*
    - *This one is a crash course and enough for the bootcamp.*
- *Gitpod Certification Course (Youtube)*
    - *This one is overkill, but covers everything…*

[4] Git and Online Version Control Systems (VCS) Knowledge

The student is expected to be comfortable using git via the terminal and also through the VSCode Source Control panel to commit, push and pull code.

The student is expected to be comfortable forking a GitHub repository, and navigating their codebase within GitHub.

If you do not have this knowledge, please watch:

- *Git and GitHub for Beginners Crash Course (Youtube)*

[5] Container Knowledge

The student is expected to understand the basic concepts of virtualization and containers.

The student is not expected to have working knowledge Docker prior to this bootcamp but it is recommended the student have general knowledge of what docker is and its components.

Please watch this quick introduction to docker:

- Docker in 05 minutes (Edith Puclla)

Prerequisite Technologies

The student is required to register an account with the following online cloud services:

[1] GitHub Account

GitHub will be used to store our application code.

We will be working from an existing minimal application web-application that is divided into a backend and frontend.

A student will be required to fork a copy of the following two repos:

- Backend Python Flask Application (TBA)
- Frontend JavaScript React Application (TBA)

[2] Gitpod Account

Gitpod will be used as our Cloud Developer Environment (CDE).

We will be working in Gitpod to write code as we integrate cloud services into the provided web applications.

Primary instruction will be conducted within Gitpod

Gitpod has a generous free-tier, which we'll be using.

[3] Github Codespaces Account

Github Codespaces will be used as our alternative Cloud Developer Environment (CDE).

The reason we have two CDE's listed is in case user's use up their Gitpod free-tier, they can then use Github Codespaces.

Github Codespaces has a generous free-tier, which we'll be using.

Why not AWS Cloud9?

The reason Gitpod and Github Codespaces was chosen over Cloud9 is to:

- Help reduce free-tier spend on AWS
- Gitpod has a VSCode experience
    - *(developers are generally more comfortable with VSCode)*
- Gitpod can be easily configured to stage the environment with needed development tools.

[4] AWS Account

Amazon Web Services (AWS) will be our Cloud Service Provider (CSPs)

AWS has a free-tier, but requires a credit card to activate your account.

This is a Bring-Your-Own-Account (BYOA) cloud project bootcamp.

There is a risk of spending while we are performing this bootcamp.
You will find these potential costs under each respective week.

We will attempt to stay with free-tier services, and provide guidance to watch out for unexpected spending.

[5] Momento Account

Momento offers a serverless cache with a generous free tier.
To build in more scalability and keep our costs down, we will cache our DynamoDB requests using Momento Serverless Cache service.

Registering a Momento account happens through a CLI. So in order to register, you must write some code in a bash terminal.

If you find this too challenging, just skip this step and we'll perform registration on the DynamoDB day. If you can register ahead of time that will expedite the process.

Instructions on how to use Momento and Register an Account

- https://www.freecodecamp.org/news/serverless-caching-for-your-web-applications/
- https://docs.momentohq.com/getting-started#obtain-an-auth-token
-

[6] Custom Domain Name

A domain name is required to complete this project in order for the backend to talk to the frontend via the web browser.

You need to either:
- purchase a domain name (recommend through Route53)
- or you can obtain a free domain name through **freenom**
    - Consider that this free domain name may expire after a period of time.

It's possible to find domains that are under a single dollar for the first year.
https://tld-list.com/

You can register any domain extension, to avoid the large renewal fee you might want to set your domain to not auto-renew.

Ensure the domain provider allows you to update the nameservers, since we will pointing the domain to AWS Namespaces to allow Amazon Route 53 to manage teh domain.

> You do not need to point your domain to Route53 until Week 7, Amazon Route53 Hosted Zones have a spend of $0.50 per month.

[6] Lucid Charts

Lucid Charts is a diagramming application to create various kinds of charts.
Lucid Charts has a bundled AWS Icon set within the application.

We will be using Lucid Charts to build our Architecture diagram

You'll need to create a free-tier account which allows us to build up to three diagrams.

[7] HoneyComb.io

We are using HoneyComb for distributed tracing.

You need to create a free-tier account.

[8] Rollbar Account

We are using Rollbar is a bug tracking tool that can capture application crashes, uncaught errors and slow response to make it easily debug production applications.

Rollbar has a generous free tier.

Project Course Outline

Week 0 — Bootcamp Overview and Introduction to Cloud Spend

Technical Tasks
In the class, we are going to lay out the foundation for the entire bootcamp by:

- Discussing the format of the bootcamp
- Going over the business use-case of our project
- Looking at an architectural diagram of what we plan to build
    - Showcase how to use **Lucid Charts** to build architectures
    - Talk about **C4 Models**
- Running through the cloud services we will utilize
- Testing that we can access our AWS accounts
- Settings up AWS free-tier and understand how to track spend in AWS
    - eg . AWS Budgets, AWS Cost Explorer, Billing Alarms
- Understanding how to look at monthly billing reports
- Launching **AWS CloudShell** and looking at AWS CLI
- Generating AWS credentials

Business Scenario
Your company has asked to put together a technical presentation on the proposed architecture that will be implemented so it can be reviewed by the fractional CTO.

Your presentation must include a technical architectural diagram and breakdown of possible services used along with their justification.

The company also wants to generally know what spend we expect to encounter and how we will ensure we keep our spending low.

Weekly Outcome

- *Gain confidence when working meter-billing with a Cloud Service Provider (CSP)*
- *To understand how to build useful architecture diagrams*
- *To gain a general idea of the cost of common cloud services*
- *To ensure we have a working AWS account*

Possible Spend Considerations

- You need a credit card to activate your AWS Account
- If your AWS account is older than a year, you will not be eligible for some free-tier services.

Alternatives and Considerations

- …

Security Considerations

- …

Homework Challenges

- Destroy your root account credentials, Set MFA, IAM role
- Use EventBridge to hookup Health Dashboard to SNS and send notification when there is a service health issue.
- Review all the questions of each pillars in the Well Architected Tool (No specialized lens)
- Create an architectural diagram (to the best of your ability) the CI/CD logical pipeline in Lucid Charts
- Research the technical and service limits of specific services and how they could impact the technical path for technical flexibility.
- Open a support ticket and request a service limit

Week 1 — Docker and App Containerization

Technical Tasks

In this class, we are going to:

- Create a new GitHub repo
- Launch the repo within a Gitpod workspace
- Configure Gitpod.yml configuration, eg. VSCode Extensions
- Clone the frontend and backend repo
- Explore the codebases
- Ensure we can get the apps running locally
- Write a Dockerfile for each app
- Ensure we get the apps running via individual container
- Create a docker-compose file
- Ensure we can orchestrate multiple containers to run side by side
- Mount directories so we can make changes while we code

Business Scenario

Your company has received the code repositories for the demo application from the contracted web-development firm. The company wants you to investigate the codebases, and ensure you can get them running.

The fractional CTO has suggested we first begin containerizing the applications for both developer and production use, and their reasons stayed why:

- *To avoid lack of documentation of application and OS configuration*

- *To ensure the effort of application and OS configuration is factored in before investing lots of feature development*
- *If we decide to hire our technical team members we can quickly replicate these environments in any preferred choice.*

The fractional CTO has asked that everything be developed in Gitpod, (a Cloud Developer Environment). This will allow the CTO at a press of a button launch the developer environment in a clean state to help with any tricky or emergency implementations, and ensure developer accountability.

Gitpod was since it supports multiple Version Control Services (VCS).. The company has invested considerable effort already in Atlassian JIRA working with the web-dev firm, and repo's highly likely might be moved to Atlassian BitBucket to take advantage of better integrations within the Atlassian's ecosystem.

Weekly Outcome

- *Gain practical knowledge working with common docker command and running container images for the purpose of local development*
- *Gain practical knowledge of working within a Cloud Development environment*
- *Be able to navigate a backend and front web-application and generally understand how they work*

Possible Spend Considerations

- *Detail GitHub free-tier*
- *Detail Gitpod free-tier*

Alternatives and Considerations

- …

Security Considerations

- …

Homework Challenges

- Run the dockerfile CMD as an external script
- Push and tag a image to DockerHub (they have a free tier)
- Use multi-stage building for a Dockerfile build
- Implement a healthcheck in the V3 Docker compose file
- Research best practices of Dockerfiles and attempt to implement it in your Dockerfile

- Learn how to install Docker on your localmachine and get the same containers running outside of Gitpod / Codespaces
- Launch an EC2 instance that has docker installed, and pull a container to demonstrate you can run your own docker processes.

Week 2 — Observability

Technical Tasks

- Instrument our backend flask application to use Open Telemetry (OTEL) with Honeycomb.io as the provider
- Run queries to explore traces within Honeycomb.io
- Instrument AWS X-Ray into backend flask application
- Configure and provision X-Ray daemon within docker-compose and send data back to X-Ray API
- Observe X-Ray traces within the AWS Console
- Integrate Rollbar for Error Logging
- Trigger an error an observe an error with Rollbar
- Install WatchTower and write a custom logger to send application log data to CloudWatch Log group

Business Scenario

- …

Weekly Outcome

- The fractional CTO has suggested that we implement distributed tracing first so because as we begin to add cloud services it will become difficult to pinpoint issue and we want to keep pace with the (probably unrealistic) development timeline

Possible Spend Considerations

- …

Alternatives and Considerations

- …

Security Considerations

- …

Homework Challenges

- Instrument Honeycomb for the frontend-application to observe network latency between frontend and backend[HARD]
- Add custom instrumentation to Honeycomb to add more attributes eg. UserId, Add a custom span

- Run custom queries in Honeycomb and save them later eg. Latency by UserID, Recent Traces

Week 3 — Decentralized Authentication

Technical Tasks

- Provision via ClickOps a Amazon Cognito User Pool
- Install and configure Amplify client-side library for Amazon Congito
- Implement API calls to Amazon Coginto for custom login, signup, recovery and forgot password page
- Show conditional elements and data based on logged in or logged out
- Verify JWT Token server side to serve authenticated API endpoints in Flask Application

Business Scenario

- The fractional CTO has suggested that authentication be solved before implementing any other business logic in the application and to ensure that we use a decentralized authentication service and specifically Amazon Cognito.

Weekly Outcome

- Practical knowledge of implementing a decentralized authentication service into a web-application with custom login and signup pages in a react application.

Possible Spend Considerations

- …

Alternatives and Considerations

- We could have used Auth0 which is a popular decentralized authentication service which has a free-tier. Since we are building a social media website we have to consider the cost of Monthly Active Users (MAUs)
    - In practice a social media platform would likely roll its own decentralized authentication service
- AuthN would have put on a good technical path to roll own decentralized authentication service however it requires Postgres and Redis so it would be too many extra moving parts and costs considerations for the scope of this bootcamp
- Azure AD B2C is another possible solution. Its low cost has support for many Identity Providers (IpDs)

Security Considerations

- …

Homework Challenges

- [Medium] Decouple the JWT verify from the application code by writing a Flask Middleware
- [Hard] Decouple the JWT verify by implementing a Container Sidecar pattern using AWS's official Aws-jwt-verify.js library
- [Hard] Decouple the JWT verify process by using Envoy as a sidecar https://www.envoyproxy.io/
- [Hard] Implement a IdP login eg. Login with Amazon or Facebook or Apple.
- [Easy] Implement MFA that send an SMS (text message), warning this has spend, investigate spend before considering, text messages are not eligible for AWS Credits
-

Week 4 — SQL Database

Technical Tasks

In this class, we are going to:

- Have a lecture about data modelling in (3rd Normal Form) 3NF for SQL
- Launch Postgres locally via a container
- Seed our Postgres Database table with data
- Write a Postgres adapter
- Write a DDL (for creating schema)
- Write an SQL read query
- Write an SQL write query
- Provision an **RDS Postgres instance**
- Configure **VPC Security Groups**
- Configure local backend application to use production connection URL
- Add a caching layer using Momento Serverless Cache
- Propagate metrics from DDB to an RDS metrics table

Business Scenario

The CEO met with the primary investor and demoed the web-application. The CEO feels the investor was impressed with our rapid development progress thus far.

The primary investor asked us how we plan to track platform growth. The CEO said that we had already begun implementation on basic tracking growth metrics within a backend dashboard of our platform:

- How many users are on the platform
- How many posts are on the platform
- How many new users per day for the last week
- How many new posts per day for the last week

The CEO is going to meet again with the primary investor next week, and the CEO wants us to deliver on their spur-of-the-moment promise. So we need to rapidly deliver a simple dashboard with platform metrics.

To keep up with the pace of feature development, the CTO has suggested we use Postgres as a relational database. Since our backend does not require the same scale as the customer-facing part of our application.

Weekly Outcome

- *Be able to data model using 3rd normal forms*
- *Practical working knowledge of utilizing a Postgres database*
- *Basic knowledge of working with an Online Analytical Processing (OLAP)*

Possible Spend Considerations

- *RDS instance*
- *RDS Snapshots*
- *Momento free-tier limit*

Alternatives and Considerations

- We require a Postgres service if we plan into AuthN which will rely on Postgres to store users for authentication.
- I want both an OnLine Transaction Processing (OLTP) and an OLAP in this project
- Redshift Serverless could be used as an OLAP
- DDB Streams to S3 and then Athena caching to Momento could have been possible as well for our OLAP like solution

Homework Challenges

- …

Week 5 — NoSQL Database

Technical Tasks

In this class, we are going to:

- Have a lecture about data modeling (Single Table Design) for NoSQL
- Launch **DynamoDB local**
- Seed our DynamoDB tables with data using **Faker**

- Write **AWS SDK code for DynamoDB** to query and scan put-item, for predefined endpoints
- Create a production DynamoDB table
- Update our backend app to use the production DynamoDB
- Add a caching layer using **Momento Severless Cache**

Business Scenario

The CEO has a very personal relationship with our primary investor, and the CEO is meeting up next week with the investor and wants to surprise them by showing that we already have a working application.

The fractional CTO has suggested that we hook up the most basic parts of the application with dummy data so that the CEO can use the Gitpod environment to launch the environment on their laptop and do a very limited demo with an emphasis that the data is backed by a real database.

The CEO wants to impress upon us that our implementation can also scale to millions of users, so the CTO has suggested that we use Amazon DynamoDB because it has a guarantee of performance at any scale based on setting a read and write capacity for a specific cost.

The CEO feels that the investor is going to ask how much this NoSQL database is going to cost at scale, and pressed the fractional CTO to crunch the numbers on the spot. The CEO was then shocked at the cost and asked for ways to curb this cost, and so that the fractional CTO can bring down repetitive costs by using a read-aside cache.

Even though the project does not require right not a read-aside cache, the fractional CTO said to implement the easiest, fastest and cost-effective implementation to appease the CEO request.

Weekly Outcome

- *Be able to data model using single table design*
- *Basic knowledge of working with a cloud SDK*
- *Basic implementation of read-aside cache in front of a database*
- *Interact with a production NoSQL database*
- *Basic knowledge of working with an OLTP*

Possible Spend Considerations

- *DynamoDB provisioned capacity*
- *CloudWatch Alarms generated by DynamoDB (DDB)*

- *Momento free-tier limit*

Alternatives and Considerations

- DynamoDB Accelerator (DAX) could have been used an alternative to Momento
- Amazon ElastiCache could have been used but decided against using ElastiCache due the time to provision, the limitations around having to be in the same VPC as our workload, and some of Redis configurations are prone to time-consuming troubleshooting with client libraries
- Momento was chosen because it requires no provisioning, we will use it likely for more than one use-case, (both SQL and NoSQL), I want to include a third-party service so we can talk about how we deal with non-aws services when using CFN, I want to teach the practice of caching, but spend tons of time dealing with caching services.
- Caching is not just for cost-savings, but often for improved performance (queries return faster) or avoid read and write contention (too many queries that make the database unresponsive)

Security Considerations

- …

Homework Challenges

- …

Week 6 — Deploying Serverless Containers (Part 1/2)

Technical Tasks

In this class, we will:

- Create an **Elastic Container Repository (ECR)**
- Push our container images to ECR
- Write an **ECS Task Definition** file for Fargate
- Launch our **Fargate services** via CLI
- Test that our services individually work
- Play around with Fargate desired capacity
- How to push new updates to your code update Fargate running tasks
- Test that we have a **Cross-origin Resource Sharing (CORS)** issue

Business Scenario

Our primary investor emailed our CEO asking when the web-application will be online to show to their extended private network.

The CEO asked if we could get the web-application up on a domain, and they agreed on a timeline of two weeks.

There are many options to deploy our containers to AWS, and ECS Fargate was chosen because its:

- Serverless containers (fewer moving parts to worry about, pay for value),
- We might want to migrate to (Kubernetes) K8s in the future (EKS can use Fargate as worker nodes)

The CTO wants us to deploy the frontend and backend as separate containers, to ensure we avoid building a monolithic application.

Weekly Outcome

- *Being able to push and tag container images to remote repository*
- *Practical knowledge of deploying, configuring and updating a serverless container*
- *Basic knowledge of working with a cloud CLI*

Possible Spend Considerations

- Elastic Container Repository spend
- Fargate Compute Spend
- CloudWatch Logs pricing
- Transfer charges (hidden costs)?

Alternatives and Considerations

- AWS Copilot CLI could have been used to deploy Fargate services, but it abstracts away a lot of the components, and it will take considerable learning opportunities. The AWS Copilot CLI generates will generate out CloudFormation code. While it can be customized, it's an additional layer we must work through.
- AWS AppRunner could have been used and poses similar issues that adopting AWS Copilot CLI in that it abstracts away a lot of our learning opportunities. For a startup, AWS AppRunner is an ideal choice but only for a small amount of containers.
- If we had time, I would have had multiple weeks to:
    - deploy to AppRunner
    - then migrate to using AWS Copilot CLI
    - then use Fargate with CFN (this is our bootcamp focus)
    - then a final migration to Fargate with Kubernetes
- The front-end application could have beenhosted on Amplify Hosting or hosted using a static S3 website with CloudFront as the CDN and in many cases this would be “ideal way” to host your frontend.

    - The reason we didn't use CloudFront is because creating a distrubtion takes a long time to create and troubleshooting a distribution has a long wait time to update

Security Considerations

- …

Homework Challenges

- Try and host your frontend application on Static Website Hosting with CloudFront
    - This challenge could count for both Week 4 and 5 because you will have to deal with CORS as well.

Week 7 — Solving CORS with a Custom Domain and Load Balancing (Part 2/2)

Technical Tasks

In this class, we are going to:

- *Create a **Route53 hosted zone** to manage our domain
- Generate a public certificate via **AWS Certificate Manager (ACM)**
- Create an **Application Load Balancer (ALB)**
- Create ALB target group that points to our Fargate instances
- Update our application to handle CORS

*We need a domain name, so you need to either purchase a domain name or use a free domain name eg. freenom. Domain names take time to create and propagate, so you have mandatory homework, or we'll have an extra half-class just for domains between weeks 6 and 7.

Weekly Outcome

- *Working with DNS records and hosted zones*
- *Configuring TLS termination at the load balancer*
- *Deploying and configuring a load balancer for multiple subdomains*
- *Basic understanding of solving CORS issues and backend-to-frontend communication*

Possible Spend Considerations

- *ALB free-tier and monthly cost of ALB*
- *Avoiding private ACM*

Alternatives and Considerations

- …

Homework Challenges

- …

Week 8 — Serverless Image Processing

Technical Tasks

In this class, we are going to:

- Test our JavaScript code to use Sharp and process a thumbnail
- Write an AWS Lambda function
- Deploy our Lambda function
- Create an S3 Bucket
- Create an S3 Event Notification to process images upload to S3 and deposit them back in the bucket
- Implement basic file upload to S3 client-side

Business Scenario

The CEO shared our private beta web-application with our investor and shared it with their internal network, and the immediate feedback was, “how do I upload my own profile photo?”

The CEO has asked that we implement this immediately.

Weekly Outcome

- *Basic knowledge of writing, deploying and logging serverless functions*
- *Basic knowledge of working with serverless object storage*
- *Basic knowledge of working with event-bus actions*

Possible Spend Considerations

- *Detail Lambda free-tier cost*
- *Detail S3 Bucket free-tier cost*

Alternatives and Considerations

- …

Security Considerations

- …

Homework Challenges

- …

Week 9 — CI/CD

Technical Tasks

In this class, we are going to:

- Write a **buildspec.yml** to build new images from our GitHub repository
- Test **AWS CodeBuild** is building and tagging our images correctly
- Write an **appspec.yml** with multiple lambda for steps
- Manually trigger a deploy with **CodeDeploy** to Fargate
- Create a **CodePipeline** that will trigger CodeBuild and CodeDeploy when code changes are pushed to our GitHub repository

Possible Spend Considerations

- *Detail Codebuild costs*
- *Detail Codedeploy costs*
- *Detail CodePipeline cost*

Weekly Outcome

- *Configuring and running a build server to bake container images and push to private repo*
- *Configure deployment controller for server containers*
- *Implement Continuous Deployment for backend and frontend applications*

Business Scenario

The fractional CTO has had a talk with the CEO about the pace of feature development and the fact that its a manually and messy process to push changes since we have yet to automated, and we risk keeping up with feature development.

The CEO agrees with the CTO and has allowed us to focus on improving the automation around deployments.

The CTO has chosen that we use AWS CodePipeline simply due to leveraging more of the AWS ecosystem and the interoperability between services.

Alternatives and Considerations

- …

Homework Challenges

- Create a build server that will run the test suite for the backend code
    - Add the test server step to CodePipeline

Week 10 — CloudFormation (Part 1/2)

https://aws.amazon.com/cloudformation/resources/templates/

In this class, we are going to:

- Write a CFN template for our Cluster and Load Balancer
- Deploy CFN template
- Update CFN template

Business Scenario

The CEO went on a trip with our primary investor to Singapore to see how they can break into the APAC (Asian Pacific region).

The CTO has taken this indication that CEO might turn around at any moment and ask for us to prepare to launch in APAC and since there are no active feature requests by the company we take this opportunity to write our Infrastructure as Code (IaC)

CloudFormation has been chosen since nearly all of our infrastructure lives in AWS (with the exception of our GitHub Git Repositories and Momento Caching Layer),

We do not expect to be changing our IaC often due to our small team and since we are likely entering a long feature development cycle. So relying on yaml files requires zero future maintenance since CFN specification rarely changes.

Weekly Outcome

- *Write Declarative Infrastructure as Code for a three-tier architecture*
- *Deploy infrastructure via IaC service that manages the remote state*

Possible Spend Considerations

- *AWS CloudFormation is free, but the resources it can provision may not be. Review the overall cost of what we are provisioning*

Alternatives and Considerations

- In the business scenario we are suggesting the “right solution” is to implement Multi-region, there are other options that would have reduced complexity and improved performance such as us utilizing AWS Global Accelerator, Amazon CloudFront.

- Other reasons why we would want to use CloudFormation (CFN) is to be explicit about what we expect the state of our infrastructure should be, this is good for security, and determining misconfiguration. If we (and we should) commit our CFN to a repo we can keep track of different interactions of our infrastructure, and this allows us possibly to rollback to the previous IaC state.

Security Considerations

- …

Homework Challenges

- …

Week 11 — CloudFormation (Part 2)

Technical Tasks

In this class we are going to

- Write a CFN template for our CI/CD pipeline
- Implement Cross-reference stack implementation
- Deploy CFN template
- Update CFN template

Weekly Outcome

- *Write Declarative Infrastructure as Code for a CI/CD pipeline*

Possible Spend Considerations

- …

Alternatives and Considerations

- …

Security Considerations

- …

Homework Challenges

- …

Week 12 — Modern APIs

Technical Tasks

In this class, we will use GraphQL and **AppSync** to modernize our API:

- Implement Realtime Pub/Sub APIs
- Implementing WebSockets for front-end react application
- Custom Resolver for Lambda Authorizator with AuthN
- Use the Amplify Libraries for AppSync
- Design GraphQL Schema
- Custom Domain with AppSync

Business Scenario

- *Implement a GraphQL API utilizing a NoSQL and SQL data source*

The company is looking to hire a web-application developer to keep pace with our upcoming feature-developer cycle to work along your the Cloud Engineering team,

After reviewing hundreds of applicants it's been hard to find web-application developers that complement writing raw SQL and next to none who know the DynamoDB API.

However, we notice many applicants know how to use React and also know how to use GraphQL. (Maybe there is a correlation between these open-source technologies built by Facebook)

The fractional CTO has asked you to investigate using AWS AppSync (a hosted version of GraphQL) and report back the effort to use maintain over a standard REST API.

Weekly Outcome

- *To be filled in…*

Possible Spend Considerations

- *AppSync has a good free-tier limit*
    - *$4M*

Alternatives and Considerations

- Instead of AppSync for realtime we could have API Gateway for Realtime, some would suggest that AppSync is easier to use
- Alternatives to AppSync could be using open-source Apollo. You'd have to run Apollo on your compute such as Lambda, Apollo does not (at the time of writing document) Realtime Subscriptions

Security Considerations

- You can set AWS WAF (OWASP 10)

- Authenication with Amazon Cognito
- Enable Logging for CloudWatch
    - This would track request to the endpoints
        - not obfuscated in the logs (just like a Lambda function)
- AppSync is Encypted by default
- Identity-based Policies through IAM
- API calls tracked through CloudTrail (GetGraphqlApi,CreateApikey)
    - This does not track requests to the endpoints

Homework Challenges

- Convert an existing DynamoDB REST API endpoint to use AppSync GraphQL
- Convert an existing Postgres SQL REST API endpoint to use AppSync GraphQL
- Replace AuthN with Cognito as authentication system with AppSync

Phase 2 Cloud Project Bootcamp

This section details how the bootcamp could be improved on the second iteration for the touline:

- Week X — Planning for expansion
- Week X — Operations DevOps
- Week X — Monitoring and Observability
- Week X — Networking
- Week X — Imperative IaC eg. CDK
- Week X — Security
- Week X — Identity
- Week X — Governance

Other improvement considerations:

- Sponsored grants to pay to guest instructors who are from an unrederpresented group
- Sandbox accounts for students who cannot gain access to valid AWS accounts
- Better pipeline for live-stream and VOD captioning for deaf and hard of hearing

Guest Instructors

Until the official timeline is confirmed guest instructors are pending confirmation

			
[Michael Liendo](#) 🦫 Modern APIs **AWS Senior Developer Advocate** (Week) Lead Instructor	[Lou Bichard](#) Cloud Career Advice **Open Up The Cloud** Cloud Career Mentor	[Ashish Rajan](#) 🔒 Cloud Security **Cloud Security Podcast** Security Instructor	[Kristi Perreault](#) Serverless **AWS Serverless Hero** (Week) Lead Instructor

			
[James Spurin](#) 🚢 Docker **Docker Captain** (Week) Support Instructor			

Contributors

This section details contributors who helped with the bootcamp

- [Kirk Kirkconnell](#) — Developer Advocate - Momento

    - Auditing bootcamp outline for grammatical spelling errors
    - Auditing bootcamp for cloud project outline completeness
    - Suggested outlining possible cloud spend
    - Suggested adding time participants will need to complete each week.
- **Lou Bichard** — OpenUpTheCloud
    - Audited project course outline's overall composition
    - Clarified prerequisite expectations
    - Suggested the addition of an SQL week
- **Sathyajith Bhat** — AWS Container Hero / SRE
    - Auditing bootcamp outline for grammatical spelling errors
    - Reworked Course Scenario to be more readable
    - Determined edge case of S3 bucket
- **Shaista Aman**
    - Suggested to add VSCode Extensions recommendations (added to Week1 Gitpod configuration for VSCode Extensions)
- **Christina Gorton** — Open Source BDM, CloudFormation - AWS
    - Upfront clarification of services used in Learning Outcomes
    - Evaluated cloud programming choices for learning outcome
    - Evaluated scope of CFN session and suggested dividing into two
    - Expanded classroom details for CFN sessions
    - Evaluated overall chunking/pacing of content with ideal classroom time blocks
    - Multiple corrections to services names
    - Suggested addition of direct links when referencing an AWS Services
- **Anurag Kale** — AWS Data Hero
    - Add OLTP to OLAP use case between DDB and RDS
    - Suggested Imperative IaC for Phase 2
    - Suggested focused on the the specific types of data modeling being used
    - Make clear the domain name requirements in the prerequisite technologies
- **Irina Geiman** — Director, Cloud Security Architecture — RBC
    - Add for each week an evolving scenario description to reinforce the purpose of each class.
    - Suggested linking the Backend and Frontend project template repos
      Suggested adding Weekly outcomes for each week
- **Vlad Ionescu** — AWS Container Hero
    - Clarified multiple technical challenges and solutions for ECS Fargate
    - Suggested the expansion of the cloud project description (what does it looks like, How does this micro-blogging platform work based on what already exists in the market? etc…)
    - Suggested clarifying how to achieve student success with the amount of content (overall and per week), the difficult content based on current experience
    - Suggested alternative technology choices, possible biases, or considerations for technical and business use-cases for weekly classes
    - Suggested weekly self Qualifier/PreAssessment help students set expectations of student outcome.

- **Markus Ostertag** — AWS Community Hero
    - Suggested possible solution for mapping subdomains to students' AWS accounts in case they cannot obtain a domain.
    - Suggested cautioning Multi-Region architectures due to complexity concerns and recommended possible alternatives.
- **Michael Liendo** — Senior Developer Advocate (AWS AppSync)
    - Developed Week 10 Modern APIs using AppSync
- **Jessica Kerr** — Manager of Developer Relations (Honeycomb.io)
    - Suggested implementing tracing and observability early, so we can debug as we go through the course.