

Intro: Side Project. Use AWS CDK in TypeScript, Lambda code and web scraping in Python, and UI tooling in Java to webscrape Amazon and allow caching and tracking of Amazon pricing information

Project Structure

Main Internal APIs - Just Lambdas

void updateProductPrices()

- updates products tracked by DynamoDB#1 if last updated over an hour ago
- Python script that webscrapes Amazon with Selenium/BeautifulSoup

void sendScheduledMessage()

- Scheduled to run weekly
- Checks DynamoDB#2 and sends out emails to all users attached to SQS
- Calls updateProductPrices then getTrackedPrices (internal version)

Main External APIs - APIs as well

void updateUserDB(username=, password=, *add_track=, *remove_track=, *add_email=, *remove_email=)

- Update DynamoDB#2 for added and removed tracking links and emails
 - If emails updated, update SQS subscriptions
- Add new username/password hash to DynamoDB#2 if needed
- PUT Request

table getTrackedPrices(username=, password=)

- Pull info from DynamoDB#2 for what is being tracked
- Pull info for prices from DynamoDB#1
 - If anything tracked not refreshed recently enough within hour, calls updateProductPrices()
- Make a version that accepts just hash value as well as an optional hidden value
- GET Request

AWS Infrastructure

- AWS/DynamoDB
 - DynamoDB #1
 - Only need to update if last updated over hour ago
 - Master Product Tracking Database
 - Key isProduct Link

- Product Name
 - Original Price
 - Current Price
 - Historic Low
 - Last Updated (Web Cache)
- DynamoDB #2
 - User Product Tracking Database
 - Key = Hash(Hash(User) | Hash>Password))
 - Links User Is Tracking
 - Emails To Send Tracked Info To (Also keep encrypted in table)
- *AWS/IAM-API Gateway-Lambda-SES*
 - APIs
 - Role to call external APIs via HTTP Queries
 - Role to call all internal APIs, limited perms
 - SES for emails
- *AWS/S3-CloudFormation-CDK*
 - Setup/Bootstrap for all the AWS infrastructure in TypeScript
- *AWS/CloudWatch*
 - Logs for Lambdas

User Interface - Java

- Java JFrame for UI
 - Allow Selecting Products To Track via URL
 - Allow Deselecting Products via Entering Just A Number
 - Allow Email Selecting/Deselecting
 - Pricing Information On Selected Products
- Login Information input to cache what we're tracking into DynamoDB #2
Upon Program Start and Program End
 - Parameters On Program Start
 - Will Ask Users If They Want To Save Info To Password File
 - Password File If Available
 - Will Ask Users If They Want To Use Info From There
- Save DynamoDB #1 calls for app version in Local Cache
 - Only need to update with DynamoDB if last updated over hour ago
 - Local Cache

- Product Link
 - Product Name
 - Original Price
 - Current Price
 - Historic Low
 - Last Updated
- Refresh values every 5 minutes to account for web crawling ban :(and add refresh button

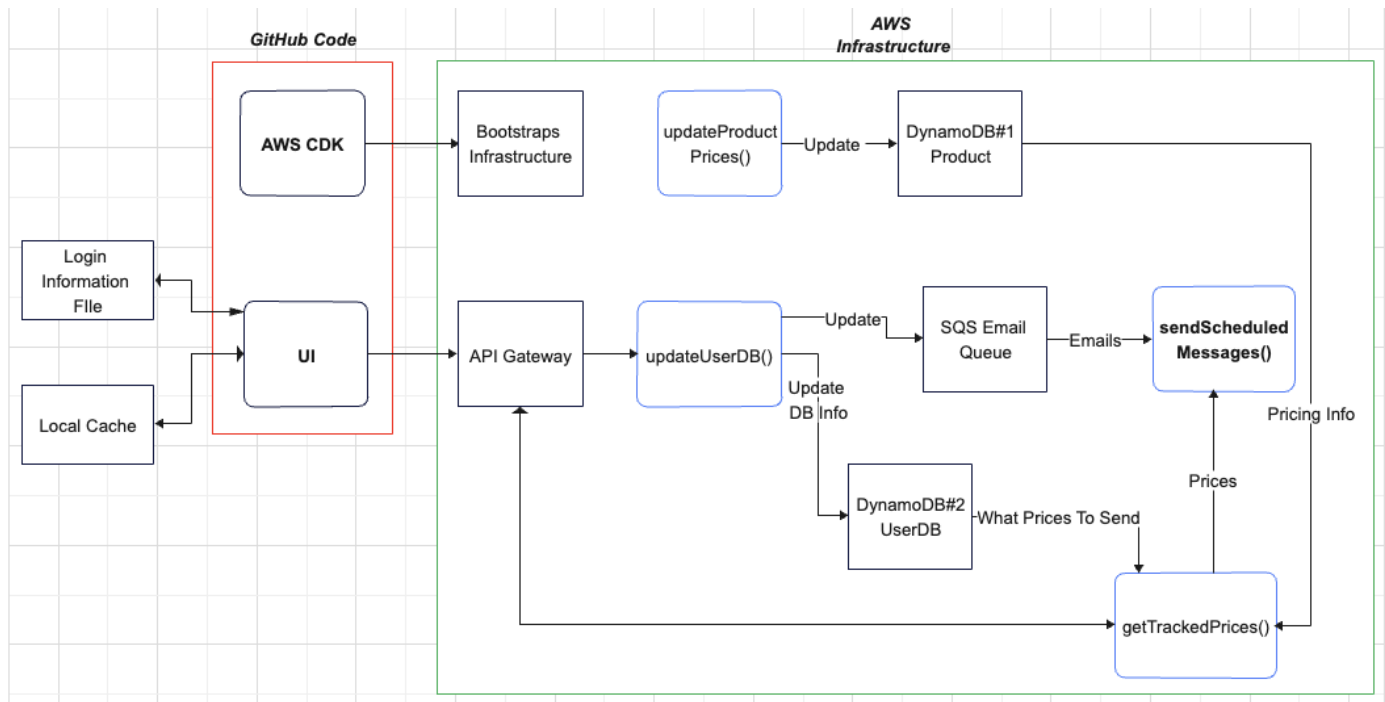
Testing

- TestNG and Spring Framework
- Proper Unit Tests for UI, Local Cache Functionality
- Proper Integration Tests for all APIs

Building

- Use Maven to build Java app
- Use CDK to build and deploy CDK app

Architecture Diagram



Bolded are architecture starting points. **Blue** are APIs/Lambdas.

UI Mock

Note:
Ctrl-R to Refresh Product Tracking Info Or
Will Refresh Every 5 Minutes

Amazon Price Tracker - Username: {}

#	Product Name	Orig Price	Curr Price	Hist Low	Del
1	Product Name	Price	Price	Price	
2	Product Link				
1u	6u	2u	2u	2u	<1u









Product:

Track

Email:

Update

Timeline

- 1) Create API Plan 
- 2) Create Architecture Diagrams 
- 3) Create GitHub Repo 
- 4) Web Scraping Script for Amazon Python 
- 5) Create AWS Resources (IAM, S3, CloudFormation, CloudWatch, Lambda, DynamoDB, SES, API Gateway) with AWS CDK - TypeScript
 - a) Bootstrap Stack (S3, CloudFormation) 
 - b) IAM Roles Stack (Internal Role and External Role) 
 - c) Database Stack (DynamoDB) 
 - d) API Stack (API Gateway, Lambda, SES)
 - i) Blank Lambdas set up 
 - ii) Setup actual lambda functions - **Oct 18**
 - (1) <https://dev.to/awscommunity-asean/creating-an-api-that-runs-selenium-via-aws-lambda-3ck3>
 - (a) Follow to update web scraping to use for AWS
 - (2) Logs And Metrics Emitted from Lambda Functions Through Print()
- 6) Create UI to interact with APIs - Java **Oct 22**
 - a) Unit Tests and Integration Tests