Deployment

CS 485/698: AI-Assisted SE

Today’s Agenda

- Team meeting (~10 minutes)
  - Standup: is your project deployable yet? If not, what are the blockers that you need to deal with so that it is?
    - I will stop by during the team meeting to ask
- Lecture on serverless computing, infrastructure-as-code, etc
- In-class activity: deploying a backend on AWS Lambda

Motivation: Did Monday's Deploy Work?

- I observed that at least some groups successfully deployed the calculator's **frontend** on Monday using Amplify
  - But for several teams, *only* the frontend worked
  - Why?

Motivation: Did Monday's Deploy Work?

- I observed that at least some groups successfully deployed the calculator's **frontend** on Monday using Amplify
  - But for several teams, *only* the frontend worked
  - Why?
- Amplify is very limited - it's only actually designed for deploying your frontend code, and even then it's limited to "simple" cases
  - For a real application, we need to manage the backend ourselves

Backend Deployment: “Serverless Computing”

- An abstraction: no servers that you must manage
  - “You just deploy code”
  - Scales automatically
  - Pay-as-you-go
  - Runs your code on “trigger events”. E.g.:
    - timers
    - HTTP events
    - file uploads

Backend Deployment: “Serverless Computing”

- How does serverless computing actually work?

Backend Deployment: “Serverless Computing”

- How does serverless computing actually work?
  - Your code + dependencies are packaged together

Backend Deployment: “Serverless Computing”

- How does serverless computing actually work?
  - Your code + dependencies are packaged together
  - Runs inside a lightweight VM in an isolated environment

Backend Deployment: “Serverless Computing”

- How does serverless computing actually work?
  - Your code + dependencies are packaged together
  - Runs inside a lightweight VM in an isolated environment
  - Some other service (e.g., your frontend on Amplify) provides trigger events

Backend Deployment: “Serverless Computing”

- How does serverless computing actually work?
  - Your code + dependencies are packaged together
  - Runs inside a lightweight VM in an isolated environment
  - Some other service (e.g., your frontend on Amplify) provides trigger events
  - After executing, the VM stays around for a bit to be reused
    - Cold start is “slow”, warm start is very fast

Backend Deployment: “Serverless Computing”

- How does serverless computing actually work?
  - Your code + dependencies are packaged together
  - Runs inside a lightweight VM in an isolated environment
  - Some other service (e.g., your frontend on Amplify) provides trigger events
  - After executing, the VM stays around for a bit to be reused
    - Cold start is “slow”, warm start is very fast
  - Eventually, VM is removed

Backend Deployment: “Serverless Computing”

- How does serverless computing actually work?
  - Your code + dependencies are packaged together
  - Runs inside a lightweight VM in an isolated environment
  - Some other service (e.g., your frontend on Amplify) provides trigger events
  - After executing, the VM stays around for a bit to be reused
    - Cold start is “slow”, warm start is very fast
  - Eventually, VM is removed
  - VMs are independent, so many can be deployed in parallel

GUIs vs Infrastructure-as-Code

- It’s possible to use the AWS console to set up their serverless computing tool (“AWS Lambda”) as a backend for our calculator
  - Click some buttons, etc
  - What’s wrong with this?

GUIs vs Infrastructure-as-Code

- It’s possible to use the AWS console to set up their serverless computing tool (“AWS Lambda”) as a backend for our calculator
  - Click some buttons, etc
  - What’s wrong with this?
- Instead, it’s better to represent the ***infrastructure as code***

GUIs vs Infrastructure-as-Code

- It's possible to use the AWS console to set up their serverless computing tool ("AWS Lambda") as a backend for our calculator
  - Click some buttons, etc
  - What's wrong with this?
- Instead, it's better to represent the ***infrastructure as code***
  - Can version control it
  - Can repeat and automate our deployments
  - Critical for continuous delivery, like we discussed last class
  - Not limited to serverless computing!

GUIs vs Infrastructure-as-Code

- It's possible
  computing t
  - Click so
  - What's w
- Instead, it's better to represent the ***infrastructure as code***
  - Can version control it
  - Can repeat and automate our deployments
  - Critical for continuous delivery, like we discussed last class
  - Not limited to serverless computing!

IaC helps **even more** with physical infrastructure:

- installing packages
- keeping servers up to date
- etc

Infrastructure-as-Code

- **Goal**: Create a system that, when run, can automatically bring physical or virtual machines to some configured state

Infrastructure-as-Code

- **Goal**: Create a system that, when run, can automatically bring physical or virtual machines to some configured state
- **Metaphor**: "***Recipes***" for configuring servers, organized into "***cookbooks***"

Infrastructure-as-Code

- **Goal**: Create a system that, when run, can automatically bring physical or virtual machines to some configured state
- **Metaphor**: "***Recipes***" for configuring servers, organized into "***cookbooks***"
    - Engineers define "healthy" states for infrastructure, then the system automatically provisions, validates, and (if needed) repairs deployed resources

Infrastructure-as-Code

- **Goal**: Create a system that, when run, can automatically bring physical or virtual machines to some configured state
- **Metaphor**: "***Recipes***" for configuring servers, organized into "***cookbooks***"
  - Engineers define "healthy" states for infrastructure, then the system automatically provisions, validates, and (if needed) repairs deployed resources
  - Recipes can be version controlled, code reviewed, etc

Infrastructure-as-Code

- **Goal**: Create a system that, when run, can automatically bring physical or virtual machines to some configured state
- **Metaphor**: "***Recipes***" for configuring servers, organized into "***cookbooks***"
  - Engineers define "healthy" states for infrastructure, then the system automatically provisions, validates, and (if needed) repairs deployed resources
  - Recipes can be version controlled, code reviewed, etc
- "Oh, this is how they do things at Amazon" - Inspiration for Chef
  - Other similar tools: Puppet, Ansible, Terraform, Palumi

In-class Activity: Deploying a Backend

- This is a pair programming activity. Find your partner.
- The overall goal today is to **deploy a real backend** for the calculator app from Monday's in-class activity.
  - You need a running calculator app frontend for this activity, so start by setting one up using Amplify (see last class' slides)
- Two parts to this activity:
  - **Part one**: deploy the calculator's backend on Lambda
  - **Part two**: wire the backend to the frontend via API Gateway
- In both parts, liberally consult documentation and your LLM!
- Challenge option: try to do as much as you can via IaC

In-class Activity: Part One: λ

- Install the AWS CDK (`npm install -g aws-cdk`)
  - https://docs.aws.amazon.com/cdk/v2/guide/hello-world.html
- Install AWS Toolkit extension into VS Code (or Cursor, or your IDE…)
- From the AWS Console, open up AWS Lambda
- Choose “Create Function”
  - Choose “Author from scratch”, enter calculate, and choose “runtime”
  - Choose “Create function”
- Copy the index.mjs file to your calculator backend codebase.
- Connect the handler to your calculate() function.
- Deploy your Lambda function with a .zip file archive
  - https://docs.aws.amazon.com/lambda/latest/dg/nodejs-package.html

In-class Activity: Part Two: Connect REST API

- Modify the Lambda code so it is invoked by HTTP POST requests to your calculate() function.
- Step 1: Open the AWS API Gateway console.
  - Create API
  - In the REST API box, choose Build
  - Under API details, enter Calculator
- Step 2: In the Resources page for your API, choose Create Resource
  - ResourceName is CalculatorManager
- Step 3: Create an HTTP POST method
  - In Resources, highlight CalculatorManager. Choose Create Method
    - Method Type: POST
    - Integration type: Lambda
    - Lambda: enter LambdaFunctionOverHttps
- Step 4: Deploy the API
  - In Resources, choose Deploy API
  - Stage: New stage: test
  - Copy the invoke URL into your calculator frontend.
- This tutorial may help: https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway-tutorial.html

Best practice: don’t commit binary files

- The history database records **changes**, not the entire file every time you commit

Best practice: don’t commit binary files

- The history database records **changes**, not the entire file every time you commit
- **Avoid** binary files for content (especially simultaneous editing)
  - Word .docx files, Excel .xlsx files, other **proprietary formats**

Best practice: don’t commit binary files

- The history database records **changes**, not the entire file every time you commit
- **Avoid** binary files for content (especially simultaneous editing)
  - Word .docx files, Excel .xlsx files, other **proprietary formats**
- Do not commit **generated files**, such as:
    - Binaries (e.g., .class files), etc.
    - IDE files (your teammates might use other tooling)

Best practice: don’t commit binary files

- The history database records **changes**, not the entire file every time you commit
- **Avoid** binary files for content (especially simultaneous editing)
  - Word .docx files, Excel .xlsx files, other **proprietary formats**
- Do not commit **generated files**, such as:
    - Binaries (e.g., .class files), etc.
    - IDE files (your teammates might use other tooling)
  - Wastes space in repository
  - Causes merge conflicts

Wrapup and Reminders

- You should have already signed up for A6
  - If you haven’t, do so ASAP
- Grad students: today is the deadline to sign up for A7
  - If you haven’t, do so ASAP
- Undergrads can also sign up for A7 (for extra credit)