

Everything We Forgot To Tell You

Agenda

1. Comparison of IaC Frameworks
2. Common Security Attack Vectors
3. CS40 Cloud Infrastructure
4. Four Miscellaneous Things
5. Case Studies of Designing Scalable Systems
6. Choosing Between AWS Services

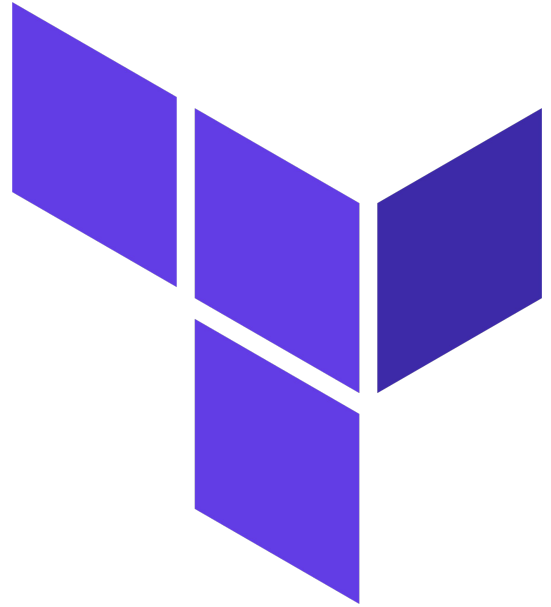
Comparing IaC frameworks



POV: CDK/CloudFormation
State Management

Recall: Terraform

- Written in HCL or JSON
- Purely declarative
- Contains a **provider** (AWS, Azure, GCP, Proxmox, etc) followed by a list of **resources**



Example of (non-AWS) Terraform

```
resource "cloudflare_pages_project" "sadsingles" {
  account_id      = var.cloudflare_account_id
  name            = "sadsingles"
  production_branch = "main"

  source {
    type = "github"
    config {
      owner              = "saligrama"
      repo_name          = "sadsingles"
      production_branch  = "main"
      deployments_enabled = true
      production_deployment_enabled = true
    }
  }
}
```

Why use Terraform vs AWS CDK?

Terraform Pros

- Support for *any* provider, even niche ones (extensible)
- **Significantly** faster deploys
- State management is portable: local JSON file → file in S3 bucket → cloud SaaS
- Lots of ecosystem tooling (Terraform Cloud, but also Terragrunt, Atlantis, Spacelift, Scalr, Env0, etc...)
- Faster (yes) support for new AWS features

AWS CDK Pros

- Write IaC in a language you're already familiar with
- L2 and L3 constructs make deployment of common patterns easier (e.g. `ApplicationLoadBalancedFargateService`)
- Easier to logically organize resources in a way AWS is aware of (stacks)

Some other contenders: CDKTF and Pulumi

- More takes on the “write IaC in a language where constructs can be generated from Typescript” concept
 - CDKTF itself synthesizes to Terraform
 - Pulumi allows you to use AWS CDK constructs, including L2 and L3 constructs

- Still have the same footguns associated with IaC in an imperative language (e.g., loops)



 Cloud Development Kit for



HashiCorp

Terraform

*CS 40 will likely switch to
Terraform in 2024-2025.*

CDKs generally work better in Typescript

- All CDKs are written natively for Typescript, and dynamically “translated” to create polyglot libraries for other languages
 - JSII: open-source AWS-written package for such translation
- Try to use what you’re familiar with – if that’s Python, consider using Python – but probably default to Typescript
 - First-class support for JSON
 - Better dependency and third-party ecosystem for CDK



Example: CDK in Go

```
lambda := awslambda.NewDockerImageFunction(  
    stack,  
    jsii.String("GradingLambda"),  
    &awslambda.DockerImageFunctionProps{  
        Code: awslambda.DockerImageCode_FromImageAsset(  
            jsii.String(path.Join(".", "../synthesizer")),  
            &awslambda.AssetImageCodeProps{}},  
        ),  
        Architecture: awslambda.Architecture_ARM_64(),  
        Tracing:       awslambda.Tracing_ACTIVE,  
        Timeout:      awscdk.Duration_Minutes(jsii.Number(5)),  
        MemorySize:   jsii.Number(2048),  
    })
```

Let IaC manage infrastructure resources, not data

- IaC should track the state of the infrastructure *only*
- If IaC tracks data, state management becomes difficult
 - e.g. database contents, secret values
 - IaC has no way of knowing the state of the data: this is the domain of application logic
- IaC should create the infrastructure resources to hold the data, but no more
 - Delete/rollback protection can ensure IaC doesn't delete data resources

Common Security Abuse Vectors

Instance Metadata Service

- For some compute instances, AWS exposes a *metadata service* (HTTP) that code running on the instance can access to get instance info + credentials
 - EC2: 169.254.169.254
 - ECS: 169.254.170.2
- Example usage: what IP address in the VPC is assigned to the ECS task?

```
curl http://169.254.170.2/v2/metadata | \  
jq .Containers[].Networks[].IPv4Addresses
```

Instance Metadata SSRF

- Instance Metadata Service contains sensitive information, since IAM role credentials are issued this way
- *Proxies* that don't restrict access to metadata endpoint can give attackers a foothold into your AWS environment
 - This is known as a **Server-Side Request Forgery** (SSRF) vulnerability

Example: SSRF → Metadata Service

Request

```
1 GET /api/dev/[REDACTED]/http://169.254.169.254/latest/meta-data/identity-credentials/ec2/
  security-credentials/ec2-instance HTTP/2
2 Host: [REDACTED]
3 Cookie: session=
  eyJlb
  NKOS5
  VzFoS
  4tdVhnMHZhdHZwbVJaS3NyN2sILCJ1c2VyaWQ10jJBZGl0eWEgU2FsaWdyYWInIn0=; session.sig=
  5z1pkd1Bk3yjXGexyuuc1oByd6A
4 Sec-Ch-UA: "Chromium";v="121", "Not A(Brand";v="99"
5 Sec-Ch-UA-Platform: "macOS"
6 Sec-Ch-UA-Mobile: ?0
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/121.0.6167.160 Safari/537.36
8 Content-Type: application/json
9 Accept: */*
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: [REDACTED]
14 Accept-Encoding: gzip, deflate, br
15 Accept-Language: en-US,en;q=0.9
16 Priority: u=1, i
17
18
```

Response

```
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 Date: Thu, 07 Mar 2024 23:54:39 GMT
4 Server: nginx/1.18.0 (Ubuntu)
5 Etag: W/"65a-GV7/gSuYpQnzHr0/4Rca6zuzkXY"
6 X-Powered-By: Express
7 X-Cache: Miss from cloudfront
8 Via: 1.1 b7621cdee138918b674c9cb957a70492.cloudfront.net (CloudFront)
9 X-Amz-Cf-Pop: SF053-P6
10 Alt-Svc: h3=":443"; ma=86400
11 X-Amz-Cf-Id: unfEqRfLIQ84UsEqB_TdGpgNXFPbVJSgWYroY5VAMaV0E6SVoSrFzg==
12
13 {
14   "bodytext":
15     "{\n  \"Code\" : \"Success\",\n  \"LastUpdated\" : \"2024-03-07T23:04:17Z\",\n  \"Type\" : \"AWS-HMAC
  \",\n  \"AccessKeyId\" : \"ASIAQZTX7IWH3CT7QJFG\",\n  \"SecretAccessKey\" : \"[REDACTED]
  NKrH3do56qNsF8NYi\",\n  \"Token\" : \"I0qJb3JpZ2luX2VlEM//////////wEaCXVzLXdlc3Q0MiJHMEUCIE6rST0WmC1
  5LU41KEd+dRTv/aDaoCQF9l0VcIH1vbzvWfPQ==\",\n  \"Expiration\" : \"2024-03-08T05:10:21Z\"}\n}"
15 }
```

March 7, 2024

AWS Credential Leakage

- The `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables are how most third-party services auth to AWS
 - These keys are often *long-lived* and don't expire
 - If an attacker gains access to these keys, they now have a foothold into your AWS account
- These keys are commonly exposed on:
 - Source code management: GitHub, GitLab, etc.
 - CI/CD platforms: CircleCI, Travis CI, Jenkins, etc.
 - Package repositories: NPM, PyPi, etc.
 - Container registries: Docker Hub, ECR, etc.
 - Frontends
 - Developer laptops (which leads to malware concerns)
- Best practice: use encrypted secrets, or use ephemeral authentication methods (e.g., OIDC for GitHub Actions, IAM Identity Center SSO)

Infosys leaked FullAdminAccess AWS keys on PyPi for over a year

15 Nov 2022

security

You can check out [their website for a lot of buzzwords](#), but it's clear from all the stock photos that they take security Very Seriously Indeed™.

However, from what I've found recently, it seems that Infosys use the following Comprehensive Management-Endorsed Proficiently Driven Cybersecurity Strategy and Framework items:

- Don't use AWS roles or temporary credentials for your developers
- Instead, use IAM user keys and give them all `FullAdminAccess` permissions
- Never rotate these keys and store them as plaintext in git
- Use these keys to protect what appears to be medical data about COVID patients
- Have someone publish those keys and the code in a public package to pypi
- Keep those keys active for days after leakage
- Make nonsensical pull requests to try and remove all references to the leak

What infrastructure did CS40 use?

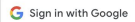
Provisioning Infrastructure

- Theme: anything that needed Stanford auth, but otherwise short, locally stateless requests
- What:
 - DNS A records
 - DNS NS records
 - AWS credits
 - Datadog flags
- Solution: Cloudflare Pages Functions + Canvas API + Google OAuth

[provisiondns.infracourse.cloud/ns](#)

DNS Provisioner

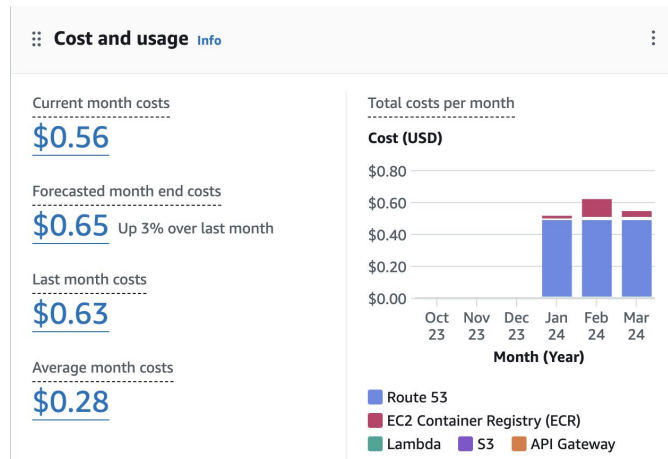
First, log in using your Stanford Google account. Then, enter your AWS nameservers to delegate your SUNet DNS zone to your AWS account.



Enter nameservers, one per line...

Autograding: Static Checks

- Problems:
 - `cdk synth` requires AWS account auth, but Gradescope can't securely auth to anything (no environment variable/secret support)
 - `cdk synth` executes student code, but student code on Gradescope can easily hijack the autograder and assign itself an arbitrary grade
- Solution: delegate `cdk synth` capability to a Lambda function
 - Gradescope zips your submission and POSTs it to the Lambda
 - Lambda runs `cdk synth` and returns the resulting CloudFormation JSON
 - We run OPA static checks on Gradescope – which doesn't execute your code



Autograding: Runtime Checks

- Goal: Make sure the website is usable by an actual human
- Backend: Use Python `requests` to directly call the backend API, ensure functionality by validating responses
- Frontend:
 - Install Chrome, Chrome Web Driver, and `selenium` in a Docker image
 - Take a screenshot of your Yoctogram login page
 - Use a perceptual hashing library to verify the frontend looks correct

Four Miscellaneous Things

Speculative Decoding

- Idea: Use a small LLM to approximate the behavior of a large LLM for a fraction of the cost
- Two LLMs, one large LLM and one small LLM
- Steps:
 - Given a prompt, have the small LLM generate a sequence of tokens
 - Estimate the probability p that the large LLM would generate this same sequence
 - Have some threshold t of acceptable responses
 - If $p > t$, then return the output from the small LLM
 - If $p < t$, then query for the expensive LLM

Spot Instances

- Spot instances: revocable instances that cost less than standard instances
 - Can be revoked by AWS with only two minutes notice
 - Useful interruptible workloads

- Way for Amazon to manage unused capacity by providing incentives to customers to pay for this unused capacity
 - Accounting for the possibility of burst workloads requiring these spot instances to be reclaimed

Reserved Instances

- Opposite of a spot instance, reserve an instance ahead of time for some period (usually ~3 years) to get a discount
 - Can be negotiated with AWS if you're a big customer
- Good if you know you're going to need an instance for a long time
- Be careful! May be stuck paying for an instance you don't really want

Billing Limits

- Almost every service can have a billing limit set
 - Service turns itself off once billing limit is reached rather than continuing to charge
- Difficult to manage at scale due to difficulties in modeling costs across many AWS services
- Very useful for small projects with limited budgets (i.e., CS 40 autograder)

Designing Scalable Systems

(Easy) Example: URL Shortener

Task: Design a scalable URL shortener like TinyURL

- Clarifying Questions:
 - Does the user need to be able to specify their own abbreviations?
 - Maximum length of URL?
 - Use cases: How many reads compared to writes? Number of concurrent accesses? How fast/scalable does it really need to be?

- Answer: A really thin wrapper over DynamoDB
 - Scalable: it's DynamoDB; AWS will infinitely scale without you having to think about what it abstracts over
 - Cost effective: only pay for what you use

(Medium) Example: Social Media Site

Task: Design a social media platform like Facebook

- Requirements:
 - Low latency
 - High availability
 - Extremely scalable

- Clarifying Questions:
 - What types of data are we supporting? What features (e.g., chat, video calls)?
 - How many concurrent users do we expect? How much data will these users generate?

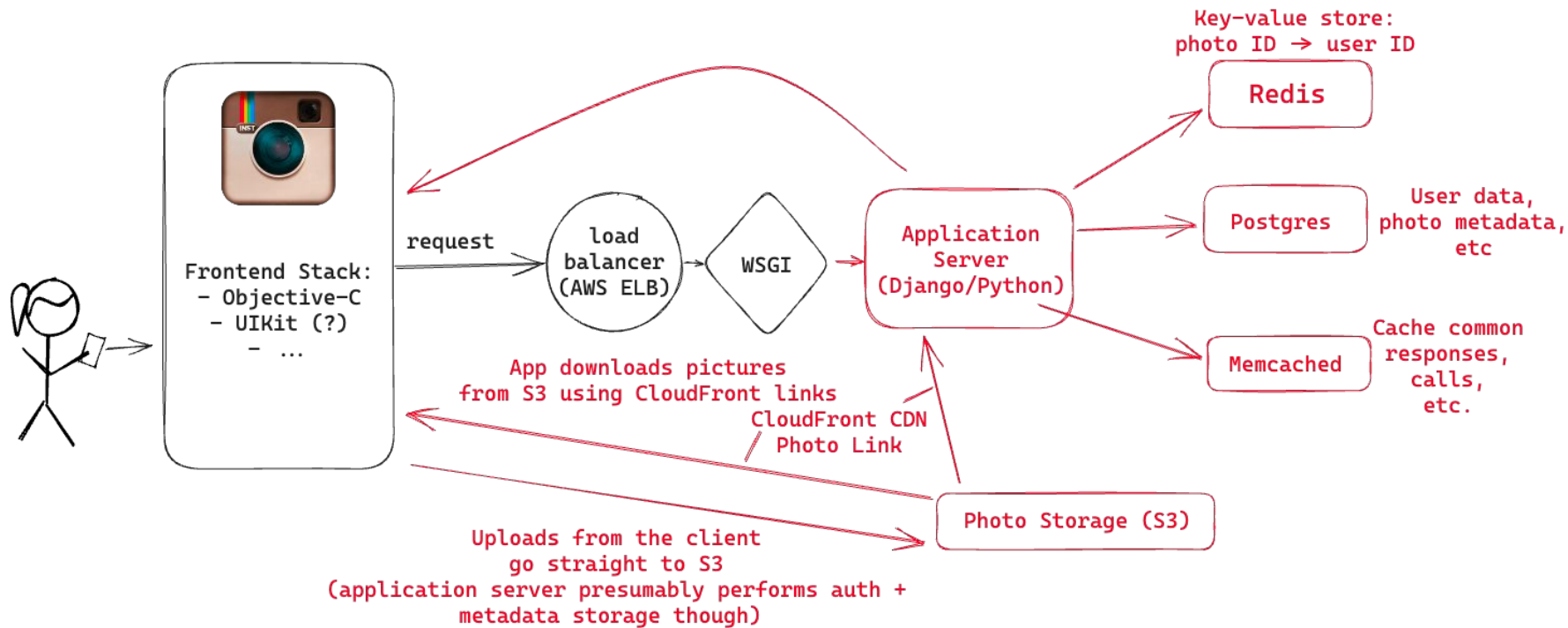
Example: Social Media Site

Task: Design a social media platform like Facebook

- Understanding the problem
 - Low latency:
 - Some sort of CDN
 - Redis caching
 - High availability:
 - Microservice architecture
 - Redundant services
 - Fault tolerant infrastructure
 - Extremely scalable
 - Elastic compute, ability to quickly grow clusters
 - Sharded DB

Possible Answer: Social Media Site

- Microservice based architecture:
 - Image service, Comment service, Like service, Follow service, etc.
- Data model:
 - Every microservice contains its own (sharded) DB
 - Postgres for transactional DB
 - DynamoDB for KV DB
 - Image service has some sort of blob storage (S3 on AWS), caches presigned URLs in Redis
 - Everything gets a UUID, when a resource is accessed, use UUID to query all relevant services for information
- Hosted on ECS, some sort of dynamic scaling
- CDN for caching popular images closer to users



(Hard) Example: Analytics Reporting Service

Task: Design an analytics reporting service that can create a report of user activity over a certain interval. Notify the user when the report is completed

- Context:
 - Assume your website has some analytics which stores information in some DB
 - Assume the it takes a long time to generate the report
 - Assume multiple reports may be requested at the same time by different parties
- Clarifying questions
 - What kind of analytics data are we working with? What volume? How is it stored?
 - How scalable does this service need to be? How many potential concurrent requests?
 - If not explicitly given, anything that determines the context specified above

Example: Analytics Reporting Service

- Understanding the problem
 - Assume your website has some analytics which stores information in some DB
 - Need to query DB to retrieve relevant information
 - Choose what to report and how to report it (answer dependent on the nature of the interview and what they told you was in the DB)
 - Assume the it takes a long time to generate the report
 - Don't burn money waiting for an HTTP request to return
 - Notify requester when done
 - Assume multiple reports may be requested at the same time by different parties:
 - Have some way of managing the current report requests

Possible Answer: Analytics Reporting Service

- Standard async pipeline using Lambdas and SQS
- When user initiates report, have one Lambda add an event to the queue
- Presence of messages in queue creates an event that calls a second Lambda
- Second Lambda generates the analytics report, notifies user when report is generated
 - In an interview: talk about the actual analytics report here (context dependent)
- Advantages of using SQS:
 - Predictable access patterns
 - Async handling is usually more efficient (can allow batching)
 - Reliability: lambda failures can be handled

Choosing the Right Service

Data Storage

- S3: Basically impossible not to use, use anytime you need to store files
- Databases: use whichever managed DB your app is written for
 - Postgres, MongoDB, etc.
 - DynamoDB is well regarded
 - Highly dependent on the data model of your application (recall 1/24 guest lecture)
- In general, there is not that much choice paralysis
 - Use whatever database your app supports that you know how to use

Should I Use a CDN?

- Consider a CDN if
 - Your workload is latency sensitive
 - You are serving media with asymmetric access patterns
 - You are dealing with large amounts of traffic to your application server

- Most applications could benefit from some sort of CDN
 - Usually cuts down on your S3 access costs
 - Offers some implicit DDoS protection

Compute

EC2 (possibly with Auto Scaling)	ECS (possibly with Auto Scaling or Fargate)
<ul style="list-style-type: none">● Based on AMIs (region specific)● Full control of OS, networking, storage● Isolation provided by hypervisor● Usually used for enterprise workloads or when you need access to the underlying machine	<ul style="list-style-type: none">● Based on Docker Images (global)● Underlying hardware abstracted away● Isolation provided by Docker runtime + Firecracker VMM (for Fargate)● Works well with microservices● Usually used for application logic

In general, we recommend using ECS for most workloads

Everything Else

- Step 1: Ask yourself if your workload can benefit from X service
- Step 2: Ask yourself if your workload can *actually* benefit from X service
- Step 3: Ask yourself if the benefits justify the cost of X service
- Step 4: Only now consider X service
 - The defaults of EC2/ECS, S3, Route 53, Lambda and some DB are enough for most use cases



**Good luck with final projects
and the rest of your quarter!**