© 2025–2026 Yiguedan Philippe Andy Yaro



Cloud Resume Challenge: Built Like Production

Yiguedan Philippe Andy Yaro

View GitHub Repository: github.com/andyyaro/CloudResumeChallenge

Let's connect! Email: yyaro@gmu.edu

LinkedIn: linkedin.com/in/andy-yaro



I built my résumé site for the Cloud Resume Challenge, Forrest Brazeal's project: a static résumé with a visitor counter. I ran it like production, on four AWS (Amazon Web Services) accounts.

S3 stores the page; CloudFront serves it over HTTPS; Route 53 answers for the domain; API Gateway, Lambda and DynamoDB run the counter, an API (an interface the page calls) built in Terraform.

Built in about two weeks in June 2025; live at portfolio.andyyaro.com. The repository and its launch film are private; access on request.

This deck checks it as it stands on 2 November 2026, including where my README and write-up claim more than the code. I am Yiguedan Philippe Andy Yaro.

A. Why It Exists



The Cloud Resume Challenge asks for one web page and a visitor counter: a résumé in cloud storage, an API in front of a database, infrastructure as code and automated deploys.

I used it to practise production habits: separate accounts with their own blast radius, a signed and private path to the page, and checks anyone can repeat.

The path to the page is visible from outside, so this deck checks it from outside; the rest it checks in the code, or marks as my own account.

1. The Base Challenge, and What This Build Adds

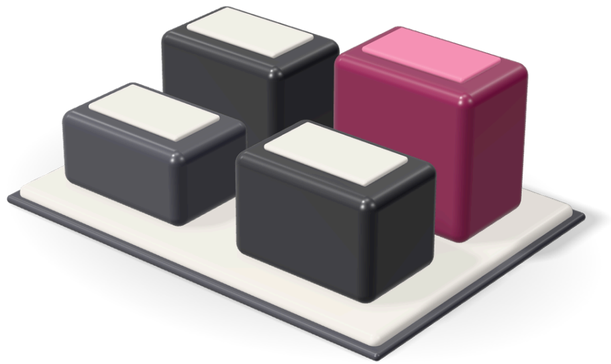
What this build has	Challenge step, or added	Where it lives	Checked on 2 November 2026
Four AWS accounts in one Organization: Management, Dev, Test, Prod	added	my diagram and blog	not visible from outside: my description
Sign-in through IAM Identity Center with MFA; a cost alarm in AWS Budgets	added	my blog	not visible from outside
A custom domain in Route 53, signed with DNSSEC, the key in AWS KMS	domain: step 6; DNSSEC: added	Management, my description	signatures validated by Cloudflare's and Google's resolvers
The HTML on S3, served over HTTPS by CloudFront; plain HTTP redirected	steps 2, 4 and 5	Prod, my description	the certificate chains to a public root; HTTP gets a 301
The bucket private and in another account, reached through an S3 Access Point	added: step 4 asks for an S3 website	Test, my description	direct requests get 403; the HTML arrives from S3
Visitor counter: API Gateway, Lambda in Python, and DynamoDB, in Terraform	steps 7 to 10 and 12; the page logs the count instead of showing it	IaC/main.tf, lambda_function.py	Terraform validates; the handler was run against a mocked AWS
A Playwright smoke test of the live API; two chained GitHub Actions workflows	steps 11 and 14; no unit tests of the Python	.github/workflows, tests/	read and traced; the recent runs on record are red
A 67-second launch film, with a source for every claim	added	media/launch-film	read; its fact sheet is used in this deck

MFA = multi-factor authentication
IAM = Identity and Access Management
DNS = Domain Name System
DNSSEC = DNS Security Extensions
KMS = Key Management Service
HTTP 301 = moved permanently: use HTTPS
smoke test = a quick check that it answers

Steps: the 16 in the challenge's AWS list at cloudresumechallenge.dev, read on 2 November 2026.

Step 7 asks the page to show the count. I chose to log it to the browser's console instead, as my blog says.

B. Architecture

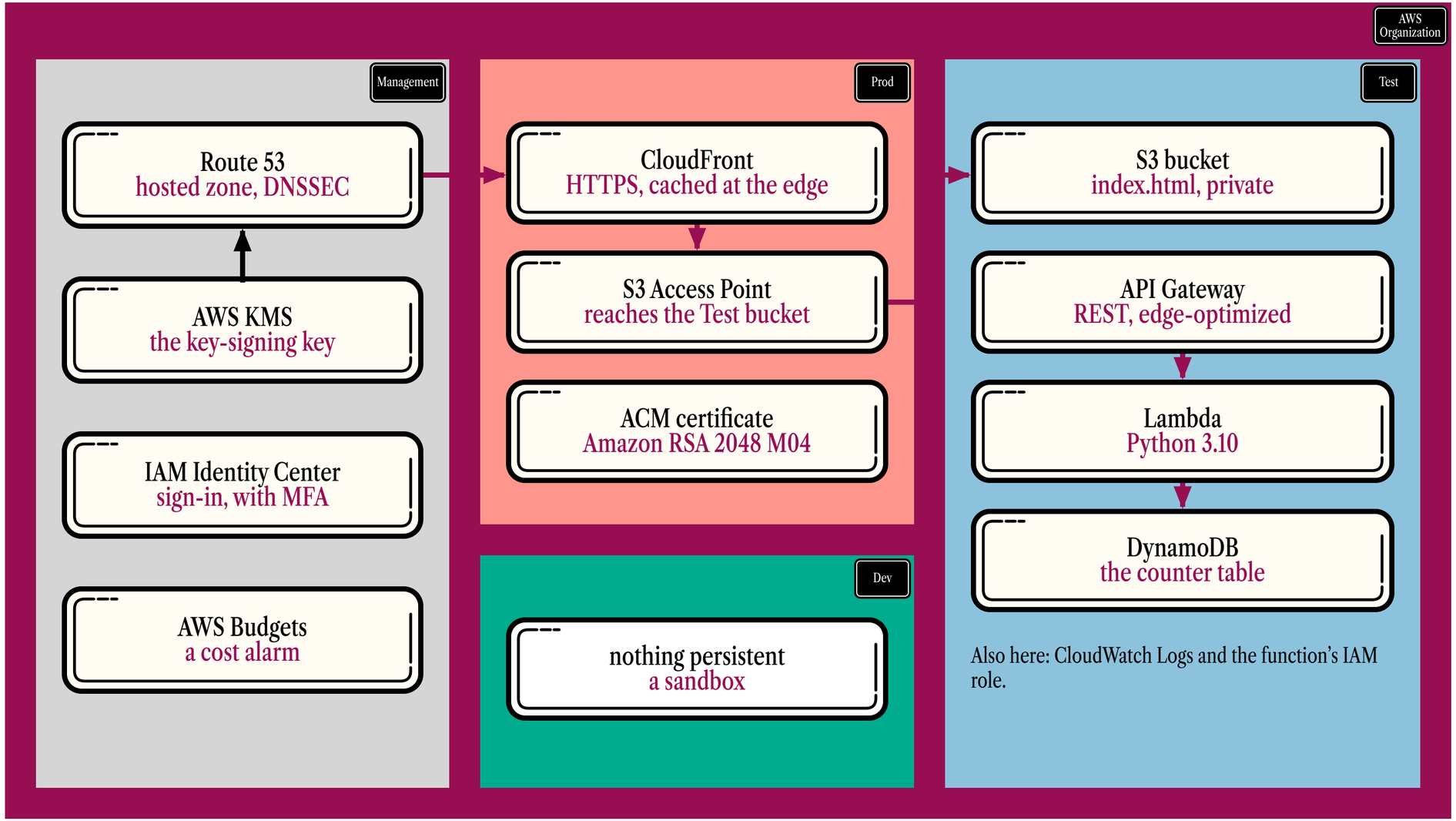


Four AWS accounts in one Organization. Management holds DNS, sign-in and the budget; Prod serves the page; Test holds the bucket and the counter; Dev is a sandbox that keeps nothing.

Production gets its own blast radius, instead of sharing an account with experiments.

The Terraform in this repository builds the counter API. The accounts, DNS, CloudFront and the bucket were set up in the console, as my blog post says.

1. Four Accounts, One Organization



DNSSEC = DNS Security Extensions
KMS = Key Management Service
MFA = multi-factor authentication
ACM = AWS Certificate Manager
IAM = Identity and Access Management
REST = an API of resources and methods
edge-optimized = reached through CloudFront

Redrawn from the architecture diagram in my write-up. Which account holds what is my own description; from outside, only the page's path can be checked.

Route 53 answers in Management, CloudFront serves from Prod, and the HTML comes from the bucket in Test.

: Organization : Management : Prod : Test : Dev : Requests : Signing key

2. Two Paths on Every Visit

The page



the address typed

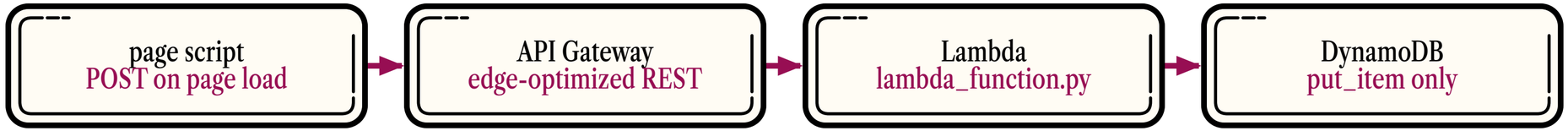
Cloudflare and Google: validated

HTTP gets a 301; Amazon's certificate, valid to April 2027

not visible from outside

direct requests: 403; objects encrypted

The counter



the reply goes to the console; no count is shown

CORS allows one origin, portfolio.andyyaro.com

keeps the count in its own environment

written on every visit, never read

DNSSEC = DNS Security Extensions
HTTP 301 = moved permanently: go to HTTPS
REST = an API of resources and methods
CORS = cross-origin resource sharing

Both start in the browser. The page's path runs none of my code; the counter is the only part that does, and the only part this repository's Terraform builds.

This is the path of the HTML. The page's scripts, images and fonts load from Framer's and Google's servers, outside it, and are not part of these checks.

: Request order

3. What This Repository Holds

Three Branches

Branch	What it holds
main	Terraform, the Lambda handler, both deploy workflows, the smoke test, the launch film; the default branch
Frontend	the site's index.html and the workflow that deploys it
Backend	an earlier state of main; on 1 November it lacked only the logo commit

This upload is main. The Frontend branch, and with it the page itself, is not part of it.

So the claim to check is narrower than "the whole stack in code": the counter API is code; the path to the page was built by hand.

IaC = infrastructure as code
CI = continuous integration

IaC/main.tf
18 Terraform resources

IaC/.terraform.lock.hcl
provider versions

IaC/api_url.txt
empty; CI passes the URL on

lambda_function.py
the counter handler

tests/test_smoke.py
one Playwright test

.github/workflows/
infra.yaml, deploy-lambda.yaml

media/launch-film/
the film's source and FACTS.md

docs/assets/brand/
logo, icons, colours

Not in Terraform: the Organization and its accounts, Route 53 and DNSSEC, CloudFront, the certificate, the bucket and the Access Point. My README says the whole stack is managed as code, and my blog's summary says "Everything is codified in Terraform"; its build section says the Terraform codifies "everything in the Backend."

C. The Path to the Page



Most hops of the page's path can be checked from outside, and I checked them on 2 November 2026.

DNS answers are signed and validated; the certificate is Amazon's and chains to a public root; plain HTTP is redirected; the bucket refuses direct requests.

One thing my write-up claims is missing: none of the usual security headers come back.

1. The Path, Checked From Outside

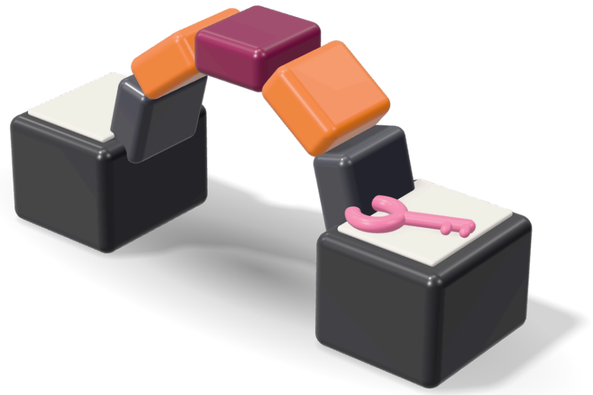
```
$ curl -sI https://portfolio.andyyaro.com | grep -iE \
    '^(HTTP/2|server|x-amz-server-side|etag|x-cache|via):?'
HTTP/2 200
server: AmazonS3
x-amz-server-side-encryption: AES256
etag: "b387c66fac63ca61d9c6a52552122cdc"
x-cache: Hit from cloudfront
via: 1.1 e3c842dc909e196086e730c7cd1792fe.cloudfront.net (CloudFront)
$ curl -sI https://portfolio.andyyaro.com | grep -ciE \
    '^(strict-transport-security|content-security-policy|x-frame-options|x-content-type-options):'
0
$ curl -s https://portfolio.andyyaro.com | md5sum
b387c66fac63ca61d9c6a52552122cdc  -
$ curl -sI http://portfolio.andyyaro.com | grep -iE '^(HTTP|location|server):?'
HTTP/1.1 301 Moved Permanently
server: CloudFront
location: https://portfolio.andyyaro.com/
$ curl -s -o /dev/null -w '%{http_code}\n' https://<bucket>.s3.amazonaws.com/index.html
403
$ openssl verify -CAfile public-roots.pem -untrusted served-intermediates.pem served-leaf.pem
served-leaf.pem: OK
$ openssl x509 -in served-leaf.pem -noout -subject -issuer -enddate
subject=CN = portfolio.andyyaro.com
issuer=C = US, O = Amazon, CN = Amazon RSA 2048 M04
notAfter=Apr 15 23:59:59 2027 GMT
```

```
$ python tools/dns_check.py
cloudflare portfolio.andyyaro.com  A   AD=true  A 3.170.19.12 | A 3.170.19.39 | RRSIG A 13 3 60
cloudflare andyyaro.com            NS  AD=true  NS ns-417.awsdns-52.com. | NS ns-668.awsdns-19.net. | RRSIG NS 13 2 172800
cloudflare andyyaro.com            DS  AD=true  DS 38231 13 2 d416ef69a96b… | RRSIG DS 13 2 86400
google     portfolio.andyyaro.com  A   AD=true  A 3.170.19.45 | A 3.170.19.39 | RRSIG a 13 3 60
```

1. The HTML comes from S3 through CloudFront, and S3 encrypts it at rest.
2. None of HSTS (always use HTTPS), CSP (limits what the page may load), X-Frame-Options or X-Content-Type-Options comes back, though my write-up says I added a security header policy.
3. The body's MD5 hash equals its ETag, S3's fingerprint of the file, and the film's fact sheet (1 November) records the same hash for the Frontend branch's index.html: the live HTML is that file.
4. Plain HTTP gets a 301 to HTTPS from CloudFront. A direct request to the bucket gets 403: it is private.
5. The certificate verifies against public roots alone; Amazon issued it, valid to 15 April 2027.
6. Cloudflare and Google both validate the DNSSEC signatures (AD=true). Records are signed with algorithm 13, ECDSA P-256.

Run on 3 November 2026 at 01:01 UTC (2 November, New York) with tools/web_check.sh and tools/dns_check.py, written for this deck; read-only requests. The bucket's name is left out.

D. Five Tries Across Accounts



CloudFront lives in the Prod account and the bucket lives in Test. Getting one to read the other took five attempts.

The fifth, an S3 Access Point made in Prod and pointing at the Test bucket, worked. Then the page still failed: first it needed a default root object, then a setting left over from attempt 3 blocked it.

This is my own account, from my blog post: it cannot be checked from outside.

1. Five Tries to Reach a Bucket in Another Account



A role made in Management and assumed from Test: I could create CloudFront and DNS records, but lost my way to the Test bucket.

It likely worked, but the S3 console lists only the current account's buckets, so I read the empty list as failure.

Enabling ACLs took the bucket out of Bucket owner enforced, and CloudFront still could not read it.

It granted access from the CLI, but would have been complex to wire into CloudFront.

Made in Prod, pointing at the Test bucket; both policies allow it, and the distribution reads through it.

Then It Still Failed

First, the root address needed a default root object, index.html.

Then the custom domain returned Access Denied. The ACLs left on since attempt 3 were the cause: setting the bucket back to Bucket owner enforced fixed it.

The distribution also needs an Origin Access Control.

What I Took From It

An assumed role does not add up two accounts' rights: only its own account counts.

An empty console list proved nothing; a real request would have.

A leftover setting can break a policy that is right, and simplifying can be the fix.

From my blog post, Phase 3, June 2025. From outside only the end state shows: the bucket is private and the page is served.

i

ACL = access control list
CLI = command-line interface
Bucket owner enforced = ACLs off; the owner owns every object
Origin Access Control = lets CloudFront sign its requests to S3

——— : The next attempt

E. The Visitor Counter

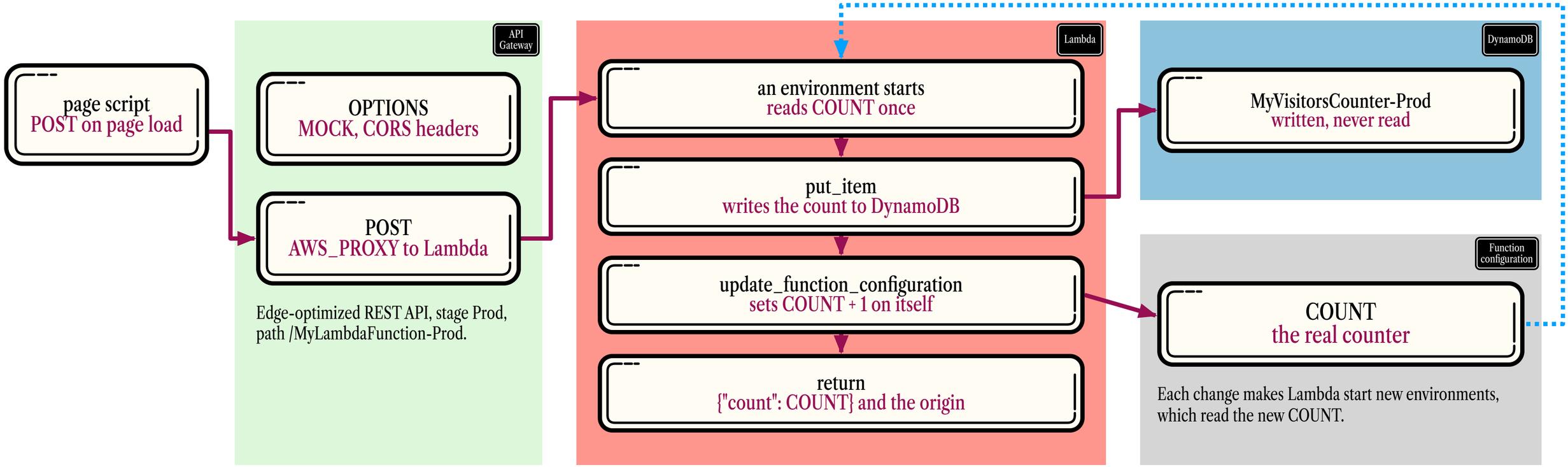


Each page view sends one POST to an API Gateway endpoint, and a Python Lambda function answers with the count.

The count does not live in DynamoDB. The function keeps it in its own environment variable and rewrites its own configuration on every visit.

Running the handler against a mocked AWS shows what that does.

1. How the Count Moves



What the Function's Role May Do

Create log groups anywhere in the region; write only to its own.

PutItem, GetItem and UpdateItem on the counter table; only PutItem is used.

UpdateFunctionConfiguration and GetFunctionConfiguration on itself: it may rewrite its own settings; only the update is used.

The page's request sends no custom headers, so browsers skip the preflight; the reply is readable because the function adds the allowed origin itself.

CORS = cross-origin resource sharing
MOCK = API Gateway answers by itself
AWS_PROXY = the whole request goes to Lambda
preflight = the browser's permission check

: API Gateway : Lambda : DynamoDB : Configuration : Each visit : The next environment

2. The Counter, Run and Probed

```
$ python tools/handler_sim.py .   # the unmodified handler, AWS mocked with moto
one warm environment, three visits:
  visit 1: returns {"count": 0}, stored count 0, COUNT now 1
  visit 2: returns {"count": 0}, stored count 0, COUNT now 1
  visit 3: returns {"count": 0}, stored count 0, COUNT now 1
a new environment after the configuration change:
  visit 4: returns {"count": 1}, stored count 1, COUNT now 2
two environments started from the same configuration, one visit each:
  visit 5: returns {"count": 2}; visit 6: returns {"count": 2}; stored count 2, COUNT now 3
[exit 0]

$ API_URL=https://example.invalid pytest --collect-only -q tests/test_smoke.py
tests/test_smoke.py::test_api_endpoint

1 test collected in 0.00s
```

1. The handler never adds 1 to its own copy, so one environment returns 0 every time; only a new one, reading the updated COUNT, moves on. The mock applies each update at once; on Lambda, a repeat visit during the update would get a 409 error instead.

2. Two environments started from the same configuration hand out the same number: under concurrent visits, counts repeat.

```
$ curl -si -X OPTIONS -H 'Origin: https://portfolio.andyyaro.com' -H \
    'Access-Control-Request-Method: POST' \
    https://<api-id>.execute-api.us-east-1.amazonaws.com/Prod/MyLambdaFunction-Prod | grep \
    -iE '^(HTTP/2|access-control|x-cache|via)|^Empty'
HTTP/2 200
access-control-allow-methods: POST,OPTIONS
access-control-allow-origin: https://portfolio.andyyaro.com
access-control-allow-headers: Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token
x-cache: Miss from cloudfront
via: 1.1 8f12d98e16716f721c4d4dbbc67ed8a8.cloudfront.net (CloudFront)
Empty
```

3. Live, the preflight returns exactly the headers in main.tf. CORS decides which pages can read the reply; it does not stop anyone else calling the API.

4. The preflight body is the word Empty: main.tf sets the response template to that string.

Handler: lambda_function.py, unmodified, with DynamoDB and Lambda mocked by moto 5.2.5 (tools/handler_sim.py, written for this deck). Preflight: answered by API Gateway's MOCK integration, which runs no code and writes nothing. No POST was sent: it would write to the live counter.

3. What This Design Costs, and a Fix

What This Design Does

Counts repeat when two environments start from the same configuration, as concurrent visits make them do.

While a configuration update is in progress, AWS says “you can’t modify the function, but you can still invoke it.” A visit in that window would fail at update_function_configuration with a conflict, and nothing catches it: that visitor gets an error.

DynamoDB is written but never read. The real count is an environment variable that main.tf sets to "0", so any successful apply sets the live count back to 0.

The function may rewrite its own configuration: a permission a counter does not need.

Python 3.10 reached its Lambda deprecation date on 31 October 2026, so AWS may no longer patch it. AWS forecasts blocking code and configuration updates from 31 August 2027; then every visit’s self-update would fail.

A Smaller, Safer Counter

One DynamoDB UpdateItem that adds 1 to the count and returns the new value: atomic, so no repeats, and no configuration writes.

Trim the role to UpdateItem on the table and its own logs, and drop the COUNT variable from main.tf.

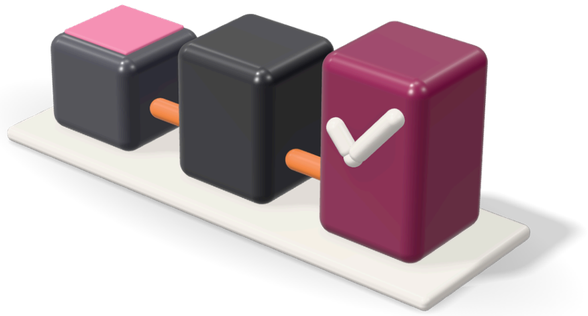
Move to a supported runtime, such as python3.13 on Amazon Linux 2023.

My blog says throttling is on; it is not in main.tf. A stage throttle in code would let scripts inflate the count only slowly.

Sources: the Lambda API reference for UpdateFunctionConfiguration, and AWS’s Lambda runtimes page; both read on 2 November 2026. Its dates are forecasts and can change.

One atomic update in DynamoDB removes the repeats, the conflicts, the reset on every apply and the self-update permission at once. The count is the part of this build to rewrite first.

F. Infrastructure and CI



18 Terraform resources build the counter API, pinned to Terraform 1.5.7 and AWS provider 5.99.1.

Two GitHub Actions workflows chain infrastructure, code and a smoke test: after a Terraform run, the code deploy waits for it to succeed.

Terraform validates today. The records of recent runs show where the chain breaks.

1. 18 Resources, 15 Types

Group	Count	Resources
API Gateway	11	rest_api (edge), resource, method ×2 (OPTIONS, POST), integration ×2 (MOCK, AWS_PROXY), method_response ×2, integration_response, deployment, stage Prod
Lambda	2	function (python3.10, TABLE_NAME, COUNT = "0"); permission for API Gateway to call POST on this path only
IAM	3	role, policy, policy_attachment: create log groups region-wide and write its own; three DynamoDB actions on the table; two configuration actions on itself
DynamoDB	1	MyVisitorsCounter-Prod: on-demand billing, hash key visits/refreshes
random	1	random_string: an 8-character suffix for the IAM names

Pinned and Checked

Terraform 1.5.7, from the workflow; aws 5.99.1, archive 2.7.1 and random 3.7.2, from the lock file.

terraform validate: valid.

terraform fmt -check: one file to reformat, whitespace only.

No backend block: state stays local, and each CI run's copy is thrown away with the runner.

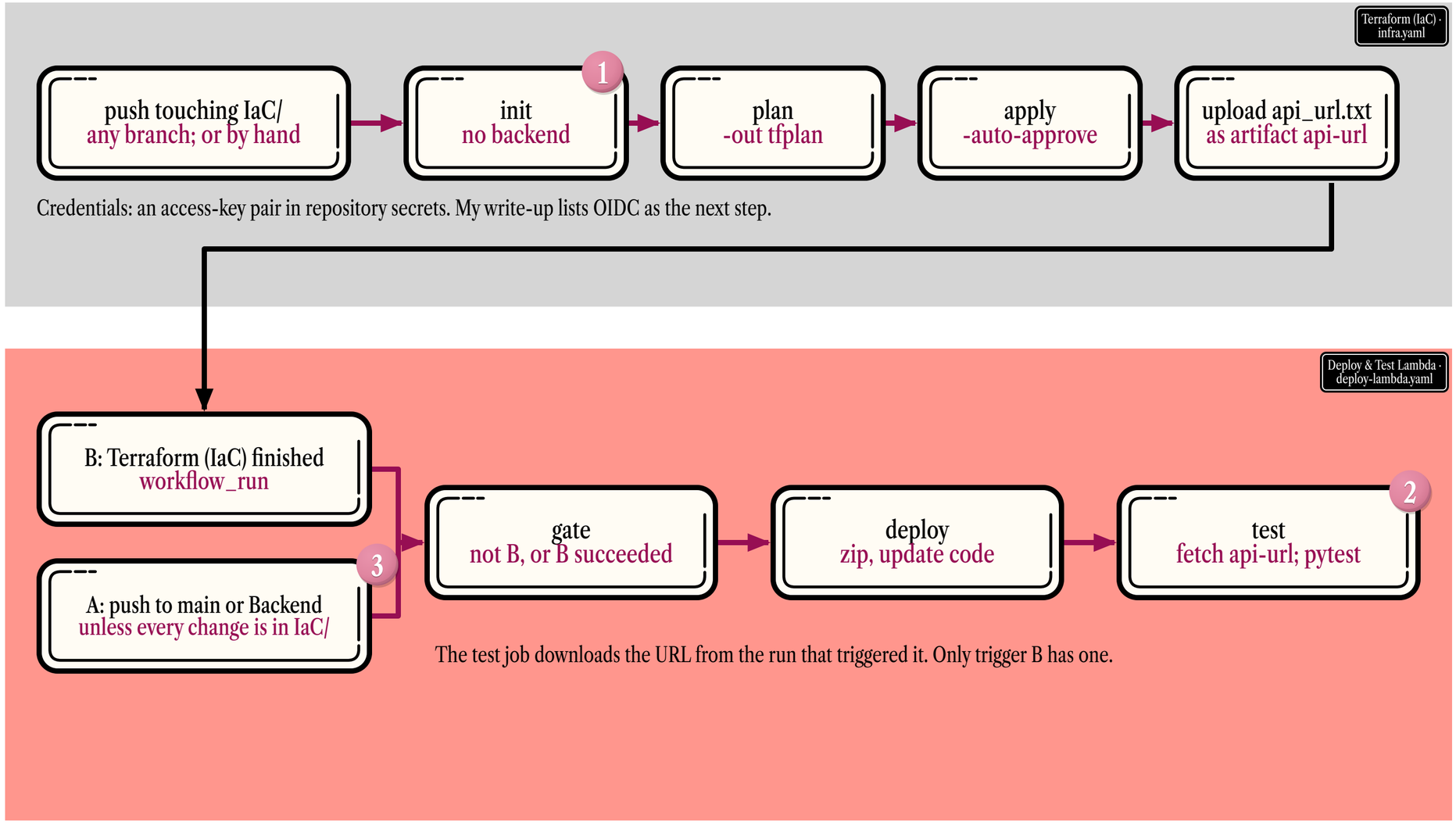
Details Worth Knowing

My README says 13 types; there are 15. The random provider is used but not declared in required_providers; Terraform finds it anyway.

aws_iam_policy_attachment is exclusive across the account: fine here, since only this role uses the policy.

The API deployment has no redeploy trigger, so later API changes need a new deployment to reach the stage.

2. Two Workflows, Chained, and Where They Break



1. No backend: each runner starts with empty state, so apply tries to create what already exists. On record, 19 August 2026: “Function already exist”, “Table already exists”.

2. On trigger A there is no triggering run, so the download fails. On record, 19 August and 1 November 2026: “Not Found”.

3. A commit touching IaC/ and the handler starts both workflows at once: paths-ignore skips a run only when every changed file matches.

i

OIDC = OpenID Connect

: infra.yaml : deploy-lambda.yaml : Steps : Trigger B

3. Terraform, Checked Today; CI, as Recorded

```
$ terraform version | head -1
Terraform v1.5.7
$ terraform init -input=false | grep -E '^- Installed|^Initializing the backend'
Initializing the backend...
- Installed hashicorp/random v3.7.2 (signed by HashiCorp)
- Installed hashicorp/archive v2.7.1 (signed by HashiCorp)
- Installed hashicorp/aws v5.99.1 (signed by HashiCorp)
$ terraform validate
Success! The configuration is valid.                                  1
$ terraform fmt -check; echo exit $?
main.tf                                                               2
exit 3
$ grep -c '^resource' main.tf
18                                                                    3
$ grep '^resource' main.tf | awk '{print $2}' | sort -u | wc -l
15
$ checkov -d . --framework terraform --compact | grep 'Passed checks'
Passed checks: 25, Failed checks: 19, Skipped checks: 0               4
```

1 Valid at the pinned versions. 2 One file to reformat. 3 18 resources, 15 types. 4 checkov, a generic static scanner: most failures are enterprise defaults (a private network or VPC, customer-managed keys, client certificates, a web application firewall or WAF); the ones that matter here are no concurrency limit on the function and no access logs.

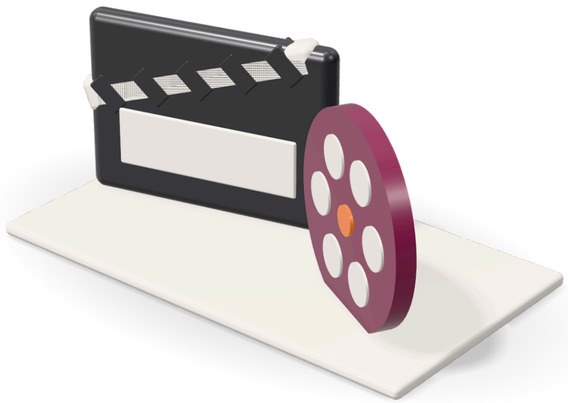
Runs on Record

When	Run, as recorded
19 Aug 2026	Terraform (IaC): apply failed, “Function already exist”, “Table already exists”
19 Aug 2026	Deploy index.html (Frontend): success in 7 s; the live page’s last-modified is 19:22:15 GMT that day
19 Aug, 1 Nov 2026	Deploy & Test Lambda, test job: failed at download-artifact, “Not Found”
1 Nov 2026	Deploy & Test Lambda, deploy step: succeeded

From the launch film’s FACTS.md, read from GitHub’s API on 1 November 2026. Run history is not in the upload, so these are as recorded there.

Terraform and checkov run on 2 November 2026 from the uploaded copy, with no AWS credentials: nothing was planned or applied. Terraform’s download was checked against HashiCorp’s SHA-256 sums.

G. The Launch Film



My README opens with a 67-second launch film: the live site plays in a browser window while the path behind it builds up beside it.

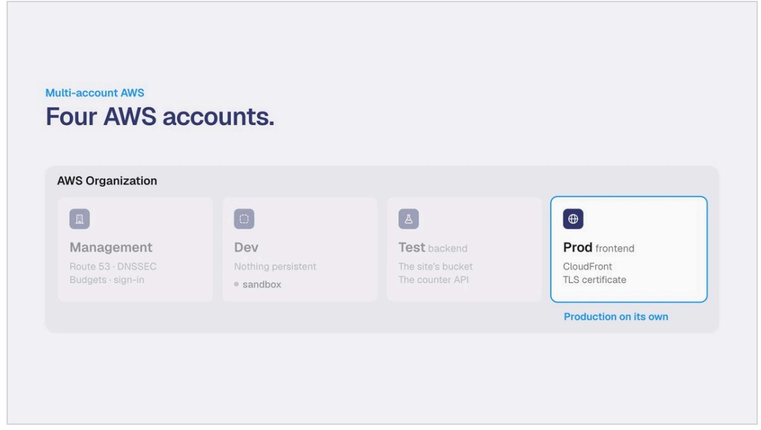
Every spoken line and on-screen claim has a source in FACTS.md, and the fact sheet lists what the film must never show: account IDs, the bucket name, any visitor count.

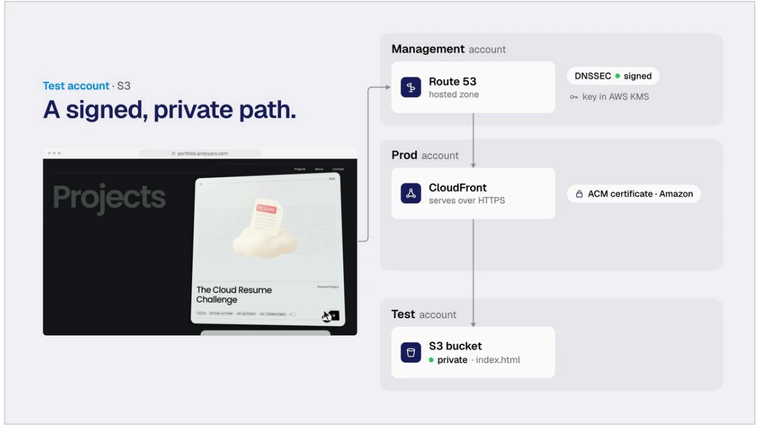
It was recorded from the live site, read only, with the counter API blocked.

1. How the Film Was Made

capture/ the live site → narration/ Kokoro-82M → sound/ voice, music, effects → compositions/ HyperFrames, GSAP → FACTS.md a source per claim → film-v1 release MP4, 11.0 MB

1. A virtual clock puts every frame exactly 1/30 s apart; the recorder blocks the counter API and analytics.
2. Voice af_heart. The best take of each line, with word times that drive the captions and the cues.
3. 70 effects, timed to the voice's words and the music's cues; music and effects from Mixkit, each listed with its licence.
4. One file per scene, eight scenes; GSAP fetched at a pinned version and checked by SHA-256.
5. Each line's source and checked value, and a list of what never appears.
6. With captions and a poster, kept out of git history as release assets; the repository is private.





Two frames from the README's preview of the film (docs/assets/readme).

Held Back on Purpose

My write-up's tiles of 16 services, about $3 a month and 3 CI runs: none was confirmed.

The write-up's claim of DynamoDB transactions: not in the code.

Any visitor count: reading one would write to the counter.

Account IDs, ARNs, the API ID and the bucket name.

ARN = Amazon Resource Name
API ID = the API's address prefix

: From recording to release

H. Proof and Limits



What could be checked was checked on 2 November 2026: live requests to the site and the API, Terraform at its pinned version, a static scan, and the handler run against a mocked AWS.

Not run: the smoke test, which writes to the live counter. The run history and the Frontend file's hash come from the film's fact sheet, read on 1 November.

The last slides set what my README and write-up claim against what the code does, and list what to fix next.

1. Where My README and Write-Up Say More Than the Code

What it says	Where	What the code or the live site shows
“DynamoDB transactions to prevent race conditions”	write-up	put_item only; the count lives in an environment variable and repeats under concurrency
“added a security header policy”	write-up	no HSTS, CSP, X-Frame-Options or X-Content-Type-Options on the live page
“streams JSON logs to CloudWatch”	write-up	the handler prints one plain line, the table’s name, per environment
“least-privilege IAM”	write-up	the function may rewrite its own configuration; GetItem, UpdateItem and GetFunctionConfiguration are granted, never used
“the whole stack managed as Infrastructure as Code”; “Everything is codified in Terraform”	README; blog	Terraform builds the counter API; the accounts, DNS, CloudFront and bucket were made in the console (my blog)
“18 resources across 13 types”	README	18 resources, 15 types
a smoke test “after every deploy”; tests “on every push”	README, blog	it runs only after Terraform-triggered deploys; on code pushes it cannot find the URL; the recent runs on record are red (FACTS.md)
“IaC/api_url.txt holds the deployed endpoint”	README	the committed file is empty; CI passes the URL as an artifact
“enabled throttling”	blog	not in main.tf; checking it live would mean flooding the counter
“enabled CloudFront Functions to force HTTPS”	write-up	the live 301 is marked “Redirect from cloudfront”, CloudFront’s own HTTP-to-HTTPS setting; a function’s reply is marked FunctionGeneratedResponse

Write-up: portfolio.andyyaro.com/the-cloud-resume-challenge, read on 2 November 2026. Each “shows” cell rests on the code in this upload, a live check in this deck, or, where marked, my blog and FACTS.md. CloudFront’s labels: its logging reference and its CloudFront Functions tutorial.

2. What to Fix Next

1. **Remote state**
   An S3 backend with locking, so CI applies change what exists instead of colliding with it.

2. **An atomic counter**
   One DynamoDB UpdateItem; remove the self-configuration permission and the COUNT variable.

3. **A supported runtime**
   Move off python3.10 before AWS blocks updates, forecast for 31 August 2027.

4. **No stored keys**
   An OIDC role for GitHub Actions in place of the access-key pair, as my write-up plans.

5. **Test every deploy**
   Read the API URL from Terraform's output or the stage, so code pushes are tested too.

6. **Security headers**
   A CloudFront response headers policy: HSTS, CSP, frame and content-type options.

7. **One path to production**
   Apply only from main, and make a mixed commit deploy code after Terraform, not alongside it.

8. **Accurate docs**
   Run terraform fmt; fix the type count and the claims in the table before.

My next steps, in order of what each protects: the live stack first, then the count, then the record of it.

i

OIDC = OpenID Connect

3. What It Shows, and What It Does Not

It Shows

A multi-account design, with production in its own account.

A signed, encrypted and private path for the page's HTML that anyone can verify from outside.

A counter API built entirely in Terraform, with CORS set explicitly.

After a Terraform change, infrastructure, code and a smoke test wired to run in order.

It Does Not Show

An accurate count: counts can repeat, and DynamoDB is never read.

A working pipeline today: the runs on record are red.

The front end as code: that part was built by hand in the console.

Proof of the account layout: it rests on my diagram and blog; from outside, only the path shows.

The live path is the strongest part, and anyone can check it today. The counter and the pipeline are where the next work goes.