Conquering Serverless

MANAGING THE HOLES IN THE SERVERLESS DEVELOPMENT LIFECYCLE

Chase Douglas, CTO, stackery.io

chase@stackery.io

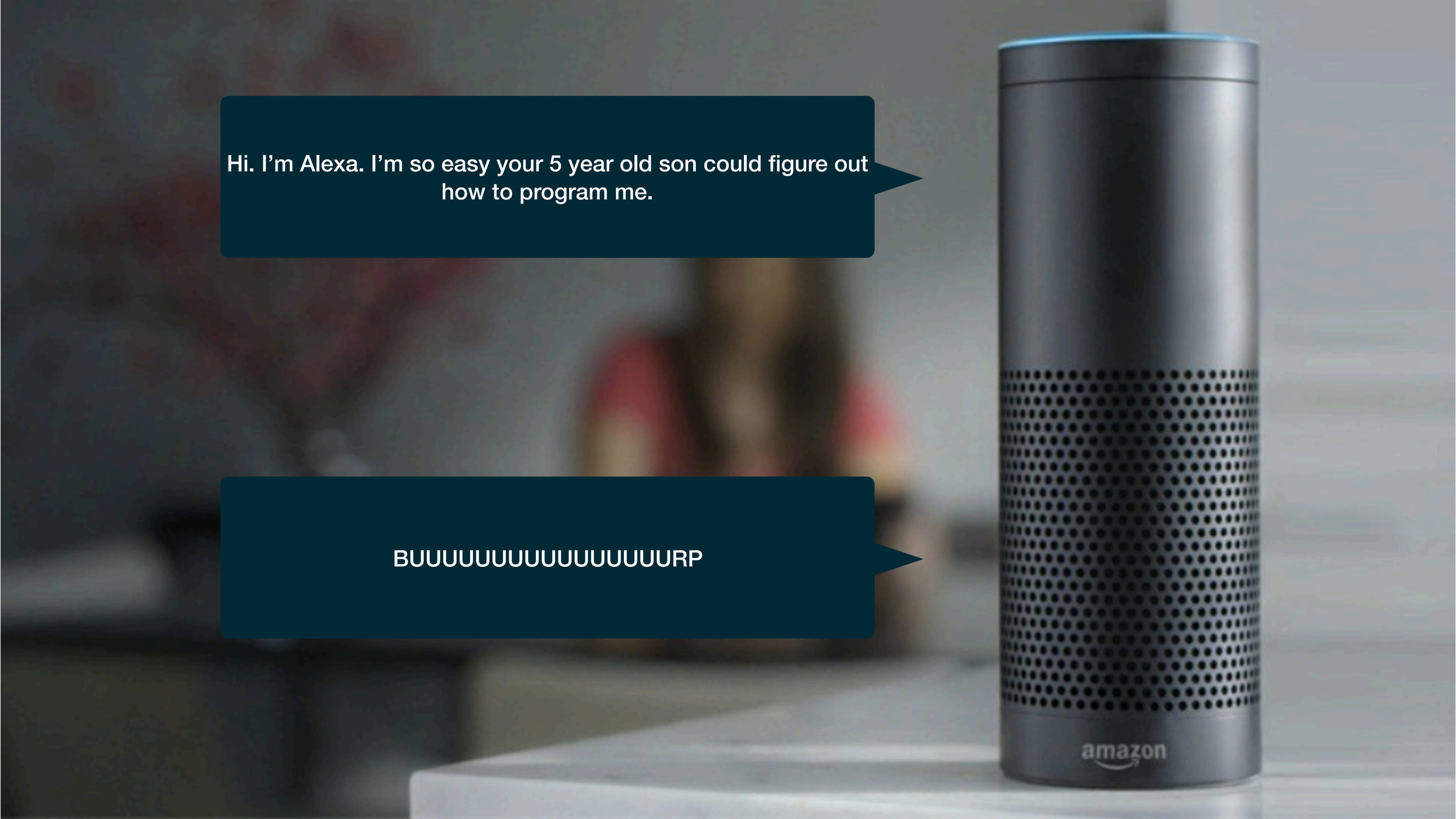
@txase

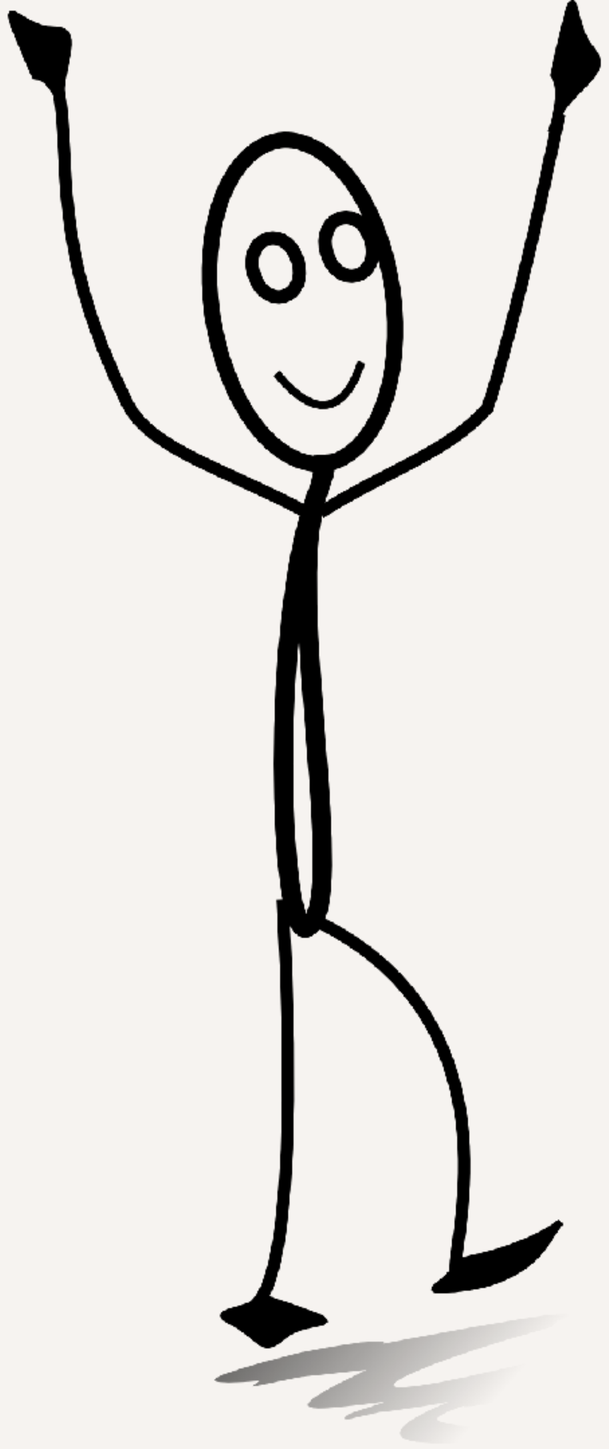
PHB: We need a new service to process messages from our gizmos

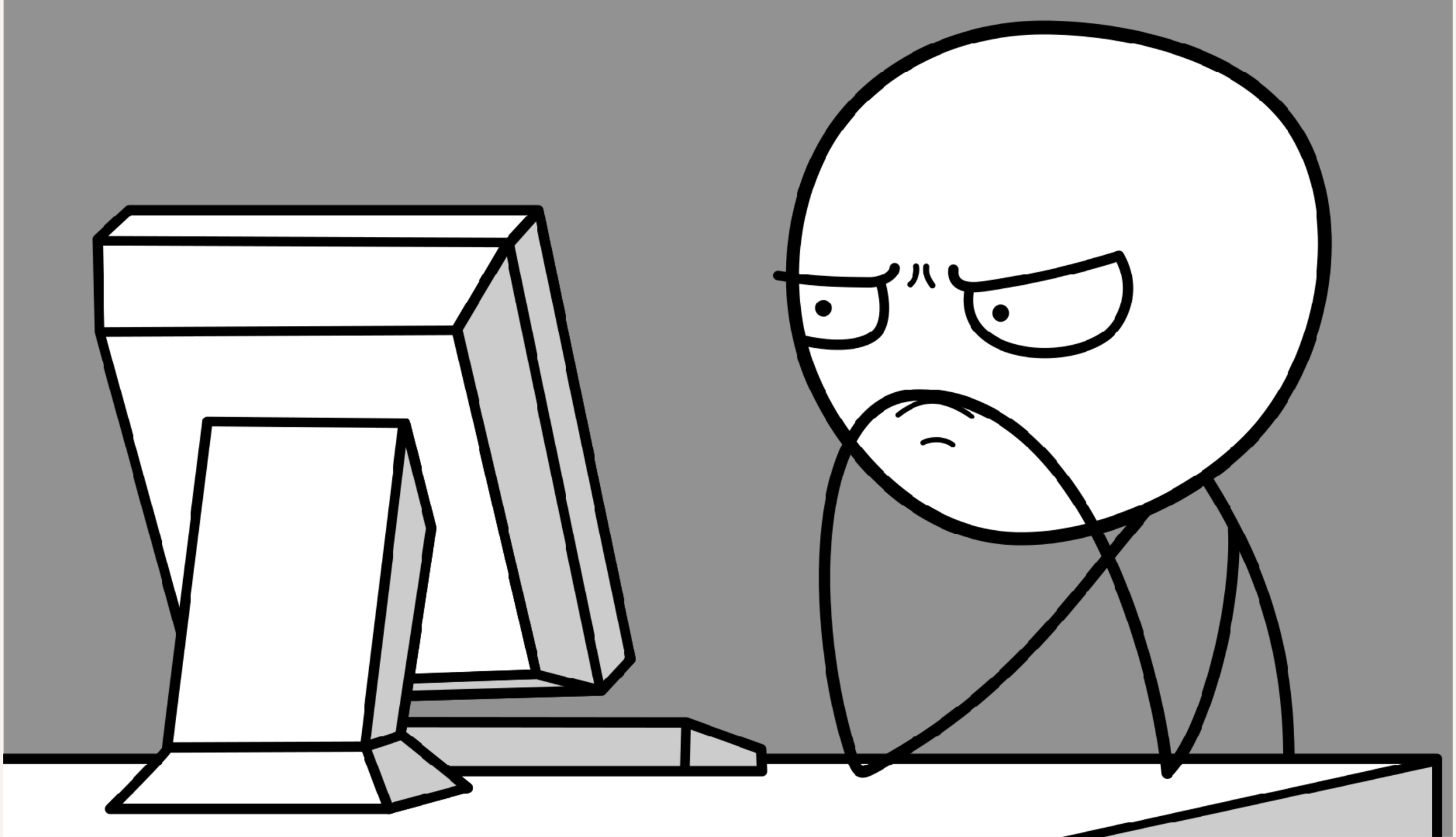
You: (Oh man oh man, I think I can do this with serverless, it’s the new hotness!)

You: Sure, we can build that in half a day.


We Can Do It!
POST FEB. 15 TO FEB. 20
WAR PRODUCTION CO-ORDINATING COMMITTEE


Hi. I’m Alexa. I’m so easy your 5 year old son could figure out how to program me.
BUUUUUUUUUUUUUUURP
amazon





The Promise:

AWS Lambda invokes your code only when needed and automatically scales to support the rate of incoming requests without requiring you to configure anything. There is no limit to the number of requests your code can handle.

AWS | LAMBDA FEATURES PAGE

The Reality:

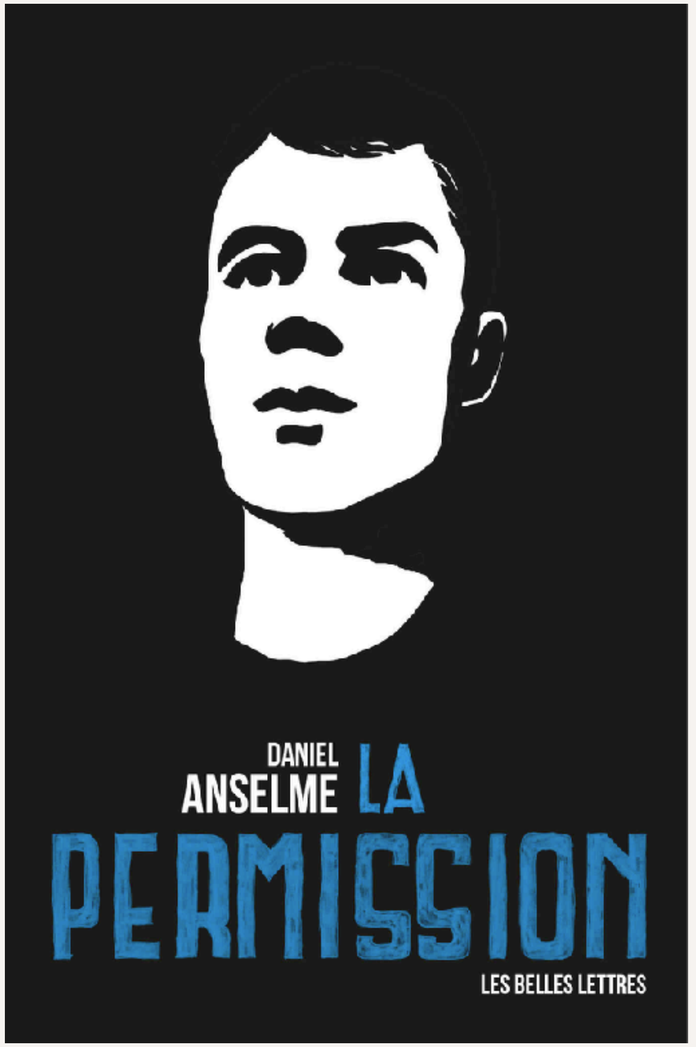
AWS Lambda invokes your code ~~only~~ sometimes when needed and ~~automatically~~ can scales to support the ~~rate~~ certains of incoming requests ~~without~~ but requiring you to ~~configure anything~~ properly. There ~~is no~~ are limits to the number of requests your ~~code~~ architecture can handle.

AWS | LAMBDA ~~FEATURES PAGE~~ (suggested edits)

Serverless Development Lifecycle Gaps

- Access And Permission Management
- Collaboration Mechanisms
- Testing
- Monitoring And Instrumentation

Access And Permission Management

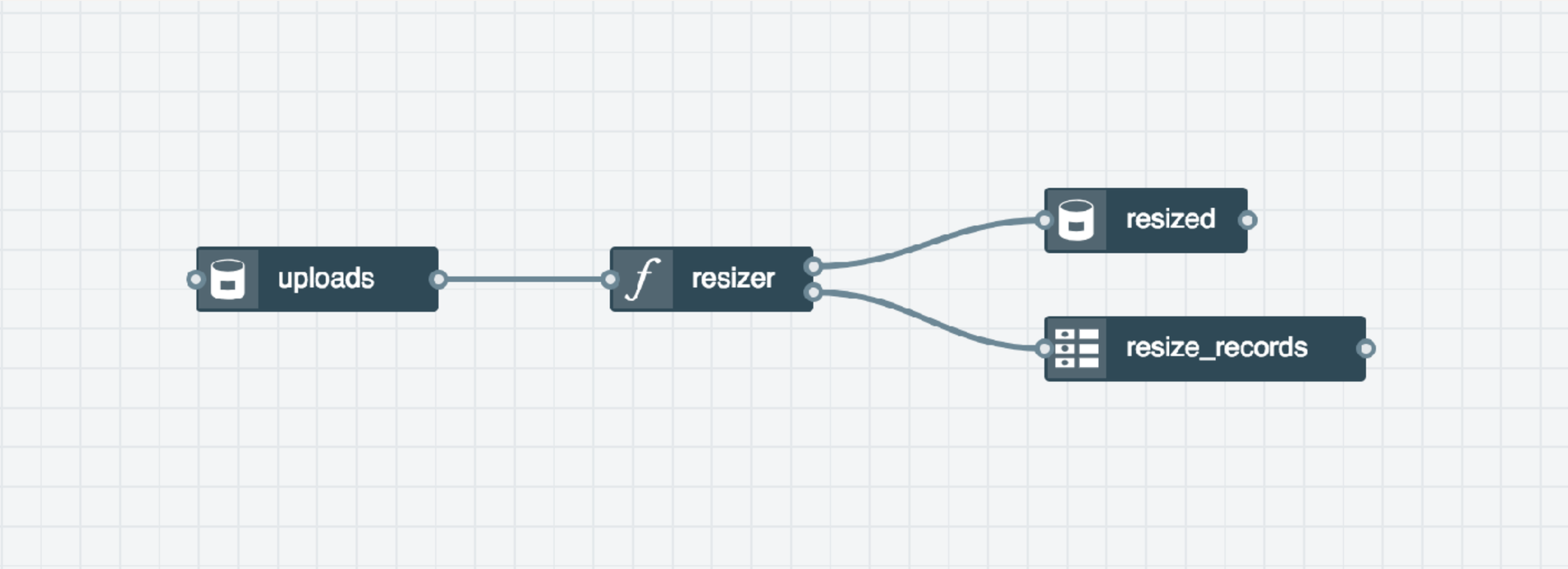


Access And Permission Management

Scenario: A serverless function that

1. Is triggered by an uploaded image to S3 Bucket “uploads”
2. Resizes the image
3. Saves the image to S3 Bucket “resized”
4. Updates a record in DynamoDB table “resize_records”

Access And Permission Management



Access And Permission Management

Shouldn’t this just work?

Nope.

(And that’s a good thing)

serverless permissions

Google Search I'm Feeling Lucky

Access And Permission Management

```yaml
service: upload-to-s3-and-postprocess

frameworkVersion: ">=1.1.0"

custom:
  bucket: <your-bucket-name>

provider:
  name: aws
  runtime: nodejs4.3
  iamRoleStatements:
    - Effect: Allow
      Action:
        - s3:*
      Resource: "*"

functions:
  postprocess:
    handler: handler.postprocess
    events:
      - s3:
          bucket: ${self:custom.bucket}
          event: s3:ObjectCreated:*
          rules:
            - suffix: .png
```


Access To Do Anything In Every S3 Bucket In AWS Account!

HTTPS://GITHUB.COM/SERVERLESS/EXAMPLES/BLOB/MASTER/AWS-NODE-UPLOAD-TO-S3-AND-POSTPROCESS/SERVERLESS.YML

Access And Permission Management

Need To Scope Access To Specific Actions

Need To Scope Access To Specific Resources

Access And Permission Management

```
- Effect: Allow
- Action:
  - s3:GetObject
  - s3:PutObject
- Resource:
  - arn:aws:s3:::uploads
  - arn:aws:s3:::resized
```

Access And Permission Management

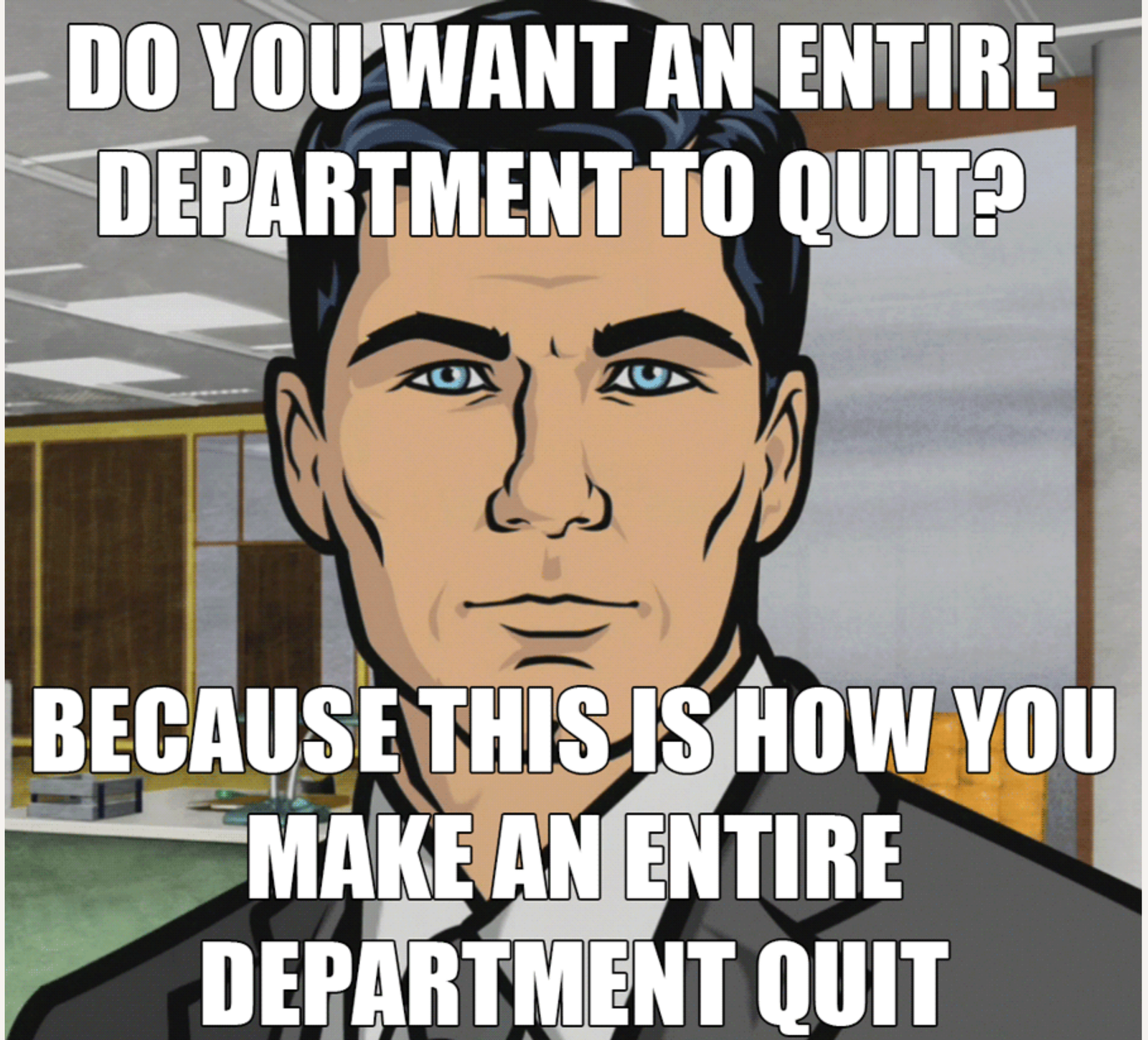
I Have To Do This For Every Function And Resource?

How?

Access And Permission Management

Option A: Manual Generation And Provision

1. Developer Hand-Codes IAM Policies
2. Principal Architect Reviews Policies
3. DevOps Deploys Policies
4. You Can Finally Use Your Policy


DO YOU WANT AN ENTIRE DEPARTMENT TO QUIT?
BECAUSE THIS IS HOW YOU MAKE AN ENTIRE DEPARTMENT QUIT

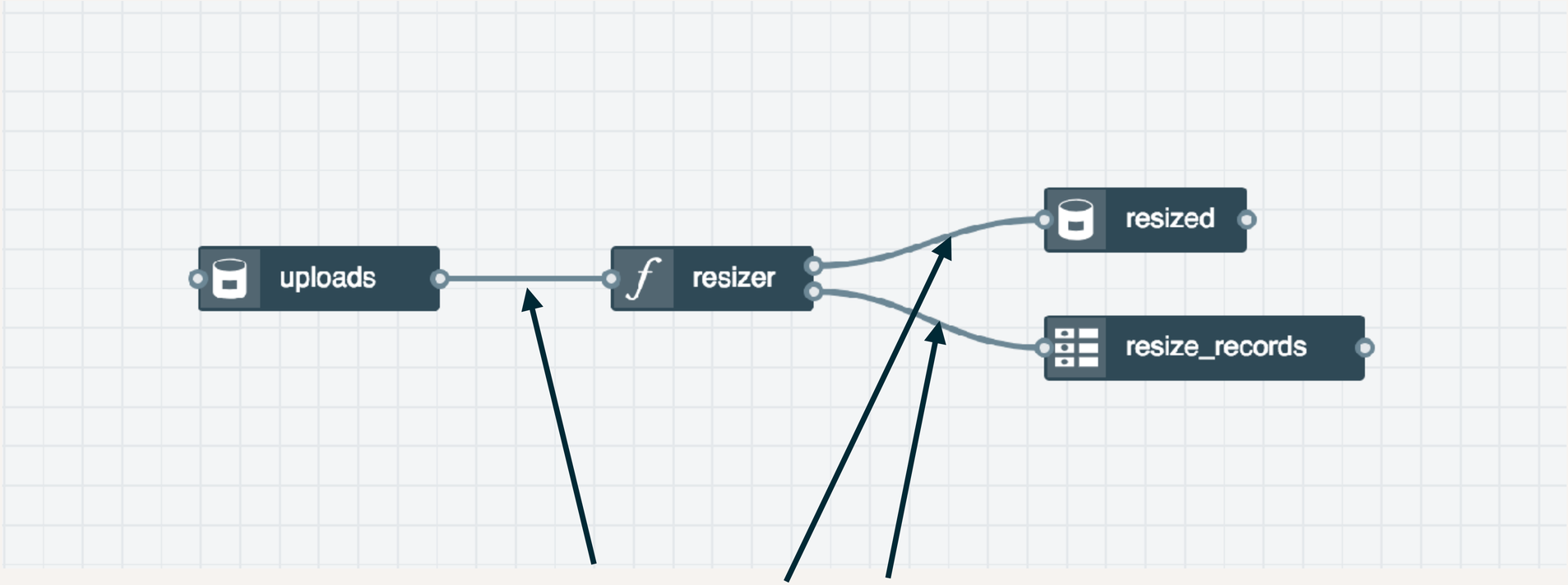
Access And Permission Management

Option B: Let Everyone Do Whatever They Want

Access And Permission Management

Option C: Use A Framework That Automatically Generates Permissions

Access And Permission Management



Automatically Generate Permissions At Deployment Time

Access And Permission Management

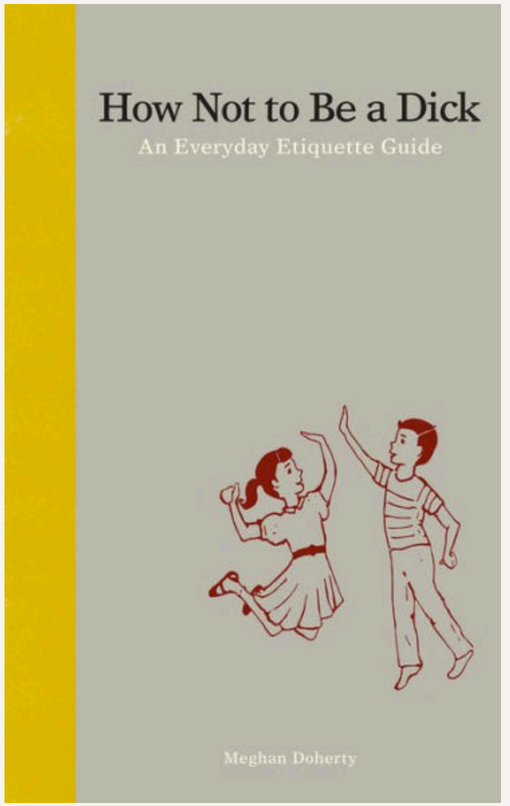
Framework-based permission management enables:

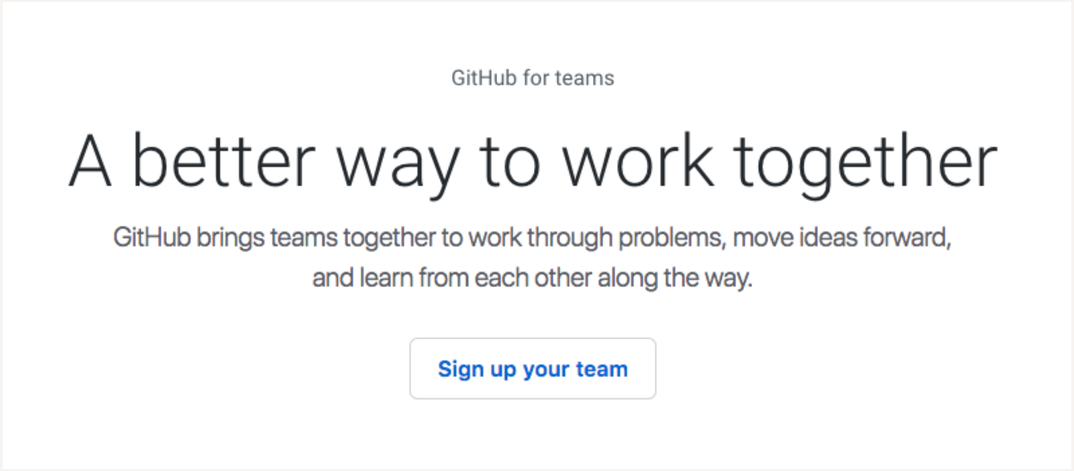
Faster development

Less errors

Compliance benefits for the organization

Collaboration Mechanisms





We’re Done Here, Right?

Collaboration Mechanisms

Serverless is cheap enough for every developer to have their own application instances

Serverless local development and testing is hard

I want all my developers to be able to provision into my team's shared AWS account

But resources require unique names

Collaboration Mechanisms

Solution: Namespace resource names

Collaboration Mechanisms

Option A: Namespace Resources Manually

```
service: new-service
provider: aws
functions:
  hello:
    name: ${opt:stage}-hello
    handler: handler.hello
  world:
    name: ${opt:stage}-world
    handler: handler.world
```

Collaboration Mechanisms

Option B: Framework Namespaces Automatically

Function Name: hello

+

Environment Name: dev

=

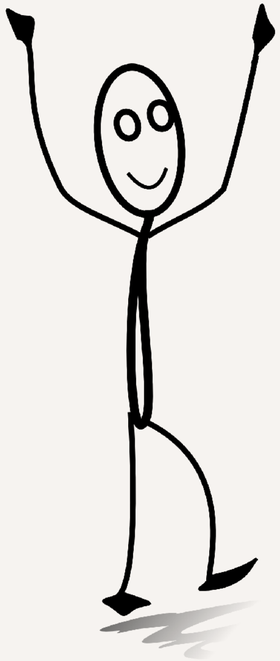
AWS Lambda Name: dev-hello

Collaboration Mechanisms

+

My Own Environment

=



Testing


"The Testing is a chilling and devious dystopian thriller that all fans of The Hunger Games will simply devour."
—JONATHAN MABERRY, New York Times best-selling author of Rot & Ruin and Flesh & Bone
THE TESTING
TIME IS ALMOST UP. YOUR TIME IS ALMOST UP. YOUR TIME IS
JOELLE CHARBONNEAU

Testing

Serverless Does Not Change Testing!

Serverless Changes How You Run Tests

Testing

Unit Tests: Same As Always

System Tests: ???

Integration Tests: ???

Testing

System And Integration Tests: Two Schools Of Thought

A: Always Test In The Cloud

B: Fake Services For Local Testing

Testing

Integration Tests In The Cloud

Pros: Faithful representation, possible today

Cons: Slower, requires cloud access

Testing

Integration Tests Locally With Service Fakes

Pros: Faster, does not require cloud access

Cons: Skew in behavior vs cloud, not very well supported today

Upstream projects are trying to make this possible/ easier (e.g. AWS SAM Local)

Testing

Integration Tests: Advice

(For Today Only!)

If application is only API endpoints + Functions, do local tests

Otherwise, deploy into cloud and test

Testing

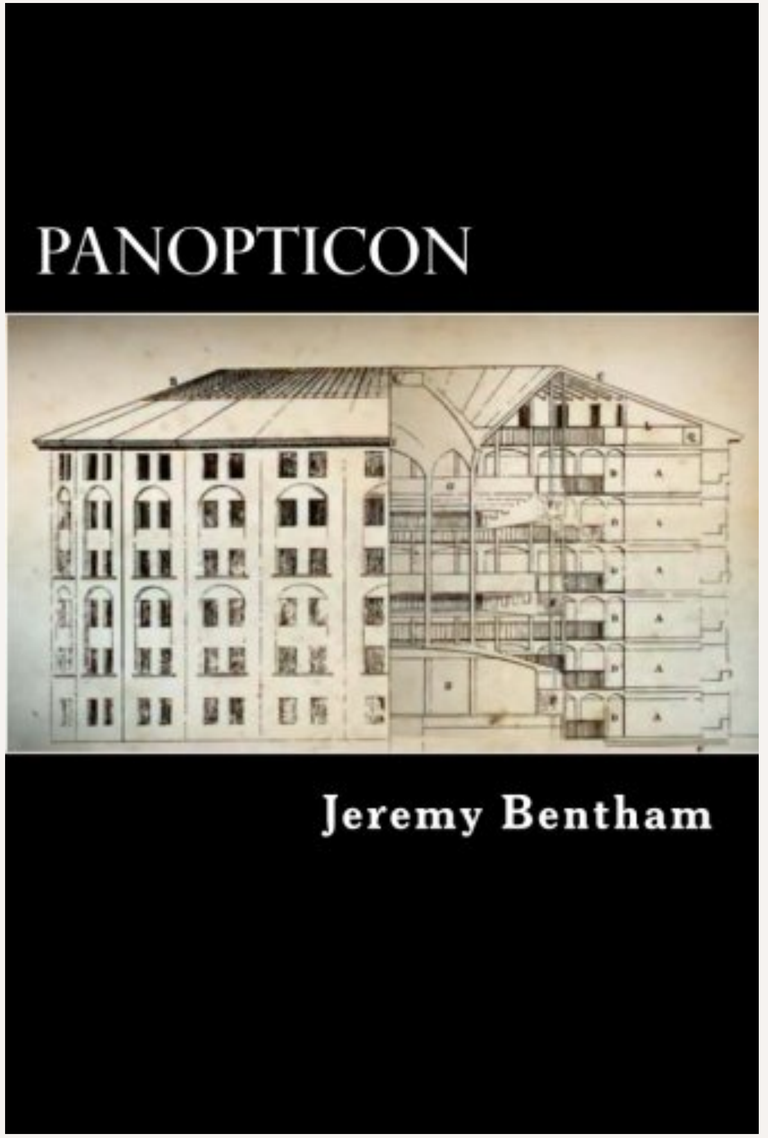
So How Do I Make A Test Environment In The Cloud?

We Solved This Already With Namespaced Resources!

Testing

With The Right Approach, Serverless Is Just As Testable As Other Architectures

Monitoring And Instrumentation



Monitoring And Instrumentation

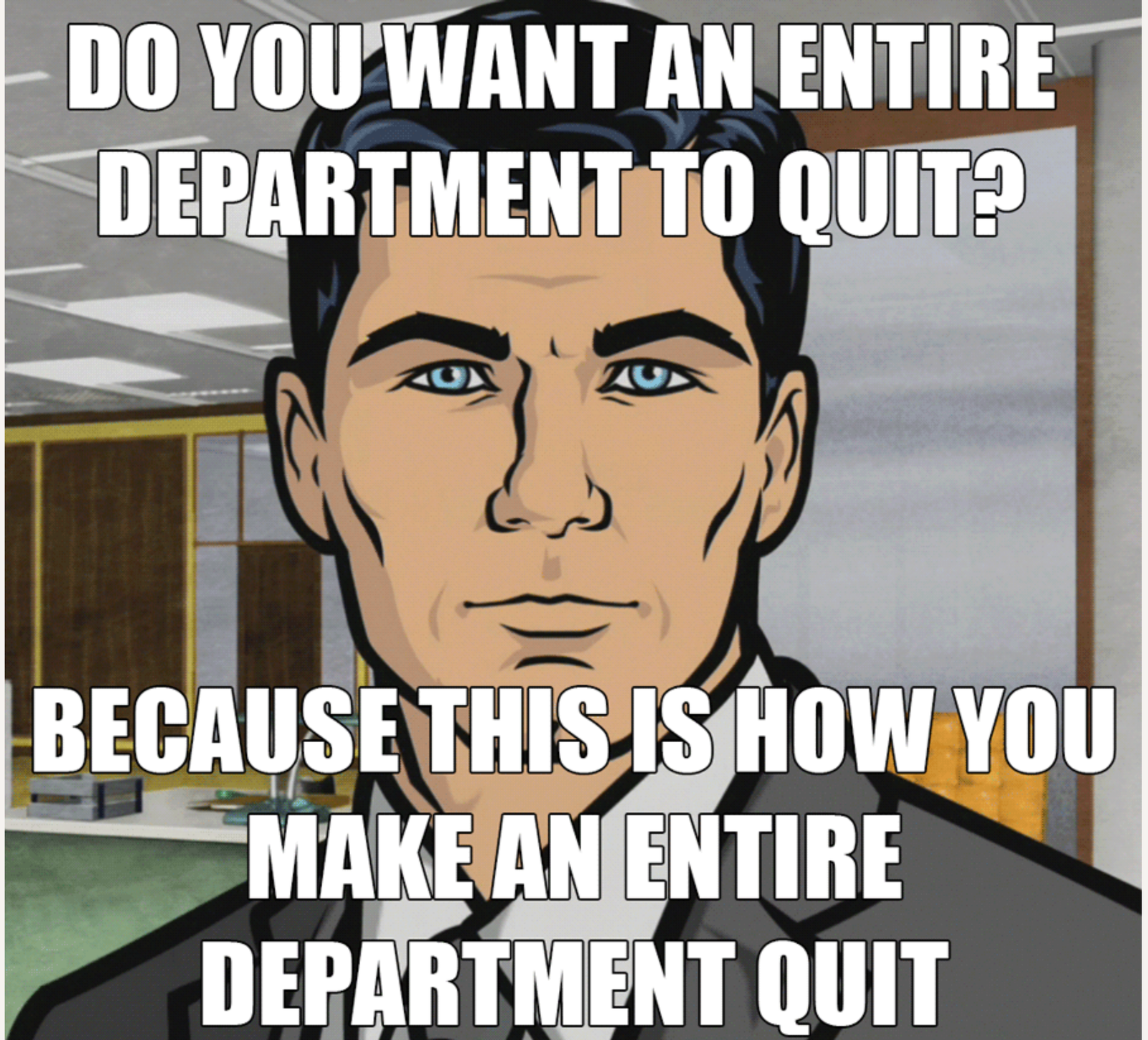
How We Do It Today

1. Organization picks a set of monitoring tools

2. Ask everyone to always instrument the same way

3. Pray

4. Draconian measures


DO YOU WANT AN ENTIRE DEPARTMENT TO QUIT?
BECAUSE THIS IS HOW YOU MAKE AN ENTIRE DEPARTMENT QUIT

Monitoring And Instrumentation

How We Should Do It

1. Pick a set of monitoring tools

2. Define instrumentation rules centrally

3. Framework auto-instruments every function

4. Cake

Monitoring And Instrumentation

How Can A Framework Auto-instrument?

Monitoring And Instrumentation

index.js Raw

```
1  module.exports.handler = event => {
2    return event.x + event.y;
3  };
```

Monitoring And Instrumentation

instrumented.js

Raw

```
1  const handler = require('./index').handler;
2
3  module.exports.handler = async event => {
4    try {
5      // Try to run original handler
6      return Promise.resolve(handler(event));
7    } catch (err) {
8      // If an error occurred, report it to Rollbar
9      const rollbar = require('rollbar');
10
11     rollbar.init(process.env.ROLLBAR_TOKEN);
12
13     // Report to Rollbar and wait for completion
14     await new Promise(resolve => rollbar.handleError(err, () => resolve()));
15
16     // Re-throw original error
17     throw err;
18   }
19 };
```

Monitoring And Instrumentation

Now Just Update The Handler:

index.handler => instrumented.handler

Monitoring And Instrumentation

Great Monitoring Solutions For Serverless

(Diatribe in person because this changes quickly over time and I don’t want to be called out for 2 year old slides)

Monitoring And Instrumentation

Metrics

Monitoring And Instrumentation

Logging

Monitoring And Instrumentation

Tracing

Monitoring And Instrumentation

Error Aggregation

Serverless Development Lifecycle Gaps

- Access And Permission Management
- Collaboration Mechanisms
- Testing
- Monitoring And Instrumentation

How Will You Manage The Gaps?

Build All The Things Yourself

Build All The Things Yourself

Use A Toolkit That Does It For You

STACKERY

STACKERY

Thank you!

Chase Douglas, CTO @ Stackery.io

chase@stackery.io

@txase