⛈ Architecture as Text

@brianleroux

Cofounder/CTO @begin

TKTK who are you? past companies? past projects?

2

ÐŸ‡¨ÐŸ‡¦

The chief cause of problems is solutions. - Eric Sevareid

Physical servers

Virtual machines

Containers

Cloud functions

- Physical servers break down so we moved to commodity hardware. - Virtual machines are slow so we removed state and load balanced them. - Containers are hard to coordinate so they became cloud functions.

5

Cloud functions are different: virtual machines and containers are a metaphor for a physical server

We knew Cloud Functions would bring significant advantages in an incumbant market.

We can't solve problems by using the same kind of thinking we used when we created them. - Albert Einstein

You have to manually provision and setup your deployment pipeline with your own assumptions about environment isolation and reproducability.

- Configuration and infrastructure can drift, leaving systems in difficult to repeat/reproduce and thus scale
- Hard to configure and thus automate
- Which means getting consistent, repeatable and reproducable builds is tough to get right

9

Compute
EC2
EC2 Container Service
Lightsail
Elastic Beanstalk
Lambda
Batch

Storage
S3
EFS
Glacier
Storage Gateway

Database
RDS
DynamoDB
ElastiCache
Redshift

Networking & Content Delivery
VPC
CloudFront
Direct Connect
Route 53

Migration
Application Discovery Service
DMS
Server Migration
Snowball

Developer Tools
CodeStar
CodeCommit
CodeBuild
CodeDeploy
CodePipeline
X-Ray

Management Tools
CloudWatch
CloudFormation
CloudTrail
Config
OpsWorks
Service Catalog
Trusted Advisor
Managed Services

Security, Identity & Compliance
IAM
Inspector
Certificate Manager
Directory Service
WAF & Shield
Artifact

Analytics
Athena
EMR
CloudSearch
Elasticsearch Service
Kinesis

Internet of Things
AWS IoT
AWS Greengrass

Contact Center
Amazon Connect

Game Development
Amazon GameLift

Mobile Services
Mobile Hub
Cognito
Device Farm
Mobile Analytics
Pinpoint

Application Services
Step Functions
SWF
API Gateway
Elastic Transcoder

Messaging
Simple Queue Service
Simple Notification Service
SES

Business Productivity
WorkDocs
WorkMail

Infrastructure as Code

1. Manifest checked into your revision control system so you can version your infra beside your code
2. Tooling for managing the manifest based infra (usually global CLI binary)

Terraform

Serverless

AWS SAM

None of these things existed when we started building begin.com; as they matured and the thing that would become .arc emmerged we still liked our approach

13

Terraform HCL

Terraform HCL

```
resource "aws_iam_role" "iam_for_lambda" {
  name = "iam_for_lambda"
  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [{
      "Action": "sts:AssumeRole",
      "Principal": {
      "Service": "lambda.amazonaws.com"
    },
    "Effect": "Allow",
    "Sid": ""
  }]
}
EOF
```

Provisions a Lambda function on AWS.

Serverless YAML

Serverless YAML

```
# serverless.yml

service: users

provider:
  name: aws
  runtime: nodejs6.10
  stage: dev # Set the default stage used. Default is dev
  region: us-east-1 # Overwrite the default region used. Defau
  profile: production # The default profile to use with this s
  memorySize: 512 # Overwrite the default memory size. Default
  deploymentBucket:
    name: com.serverless.${self:provider.region}.deploys # Ove
    serverSideEncryption: AES256 # when using server-side encr
  versionFunctions: false # Optional function versioning
```

Wires a couple of Lambdas and a DynamoDB Table

AWS SAM

AWS SAM

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Simple CRUD webservice. State is stored in a Simp
Resources:
GetFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.get
    Runtime: nodejs4.3
    CodeUri: s3:///api_backend.zip
    Policies: AmazonDynamoDBReadOnlyAccess
    Environment:
      Variables:
        TABLE_NAME: !Ref Table
    Events:
```

Wires a couple of Lambdas and a DynamoDB Table

1. Deep proprietary knowledge is required to configure and maintain common infrastructure primatives
2. Painful manifest files; JSON and YAML are not ideal
3. Tooling was designed for the last generation of metaphors

- Tooling lags behind AWS releases
- Setup requires expertise in AWS Primatives which might as well be AWS Frameworks
- JSON: is difficult to read, has no comments, and unforgiving to edit
- YAML isn't much better and especially worse with deeply nested statements

20

We are committing AWS infra config arcana into our revision control systems

Infrastructure as Code

~~Infrastructure as Code~~

Architecture as Text

`.arc` is a plaintext manifest file

Designed to solve the specific problems above by treating infra as a build artifacts

.arc

```
# this is an .arc file
@app
helloapp

@html
get /
```

Things to understand about `.arc`

1. Comments start with #
2. Sections start with `@`
3. **Everything after a section becomes instructions for generating AWS infrastructure**

`.arc` lives in the root of a Node project

```
# this is an .arc file
@app
helloapp

@html
get /
```

`.arc` file always lives beside `package.json`

`@app` defines a namespace

```
# this is an .arc file
@app
helloapp

@html
get /
```

`@app` namespace scopes all generated infrastructure within AWS.

`@html` defines `text/html` routes

```
# this is an .arc file
@app
helloapp

@html
get /
```

(Other HTTP handlers supported: `@json`)

`.arc` tooling is *local* to the project using `npm scripts`

npm run create

```
#   /
#   |-src
#   | '-html
#   |   '-get-index
#   |     |-index.js
#   |     '-package.json
#   |-.arc
#   '-package.json
```

A Lambda function wired to API Gateway handles requests to `/`.

`.arc` is expressive

```
# this is an .arc file
@app
helloapp

@html
get /
get /about
get /contact
post /contact

@json
get /api
```

npm run create

```
#   /
#   |-src
#   | |-html
#   | | |-get-index/
#   | | |-get-about/
#   | | |-get-contact/
#   | | '-post-contact/
#   | '-json
#   |   '-get-api/
#   |-.arc
#   '-package.json
```

The file system layout mirrors the `.arc` file.

a complete `.arc` example

```
@app
hello

@html
get /
post /likes

@json
get /likes

@events
hit-counter

@scheduled
daily-affirmation rate(1 day)
```

Lets look at the generated cloud function code

src/html/get-index/index.js

```
var arc = require('@architect/functions')

function index(req, res) {
  console.log(JSON.stringify(req, null, 2))
  res({
    html: 'hello world'
  })
}

exports.handler = arc.html.get(index)
```

src/html/post-likes/index.js

```
var arc = require('@architect/functions')

function likes(req, res) {
  // send an email or something here
  res({
    location: '/'
  })
}

exports.handler = arc.html.post(likes)
```

src/events/hit-counter/index.js

```
var arc = require('@architect/functions')

function count(payload, callback) {
  // do a db thing here probs
  console.log(JSON.stringify(payload, null, 2))
  callback()
}

exports.handler = arc.events.subscribe(count)
```

src/scheduled/daily-affirmation/index.js

```
var arc = require('@architect/functions')

function daily(payload, callback) {
  // idk, send an sms here maybe?
  console.log(JSON.stringify(payload, null, 2))
  callback()
}

exports.handler = arc.scheduled(daily)
```

src/tables/likes-update/index.js

```
var arc = require('@architect/functions')

function like(record, callback) {
  // anything you want!
  console.log(JSON.stringify(payload, null, 2))
  callback()
}

exports.handler = arc.tables.update(like)
```

`.arc` has three facets

1. Declaritively define architecture with a high level primitives in plain text in `.arc`
2. Workflows to generate local code, configure, provision and deploy infrastructure
3. Cloud function code itself

Other stuff you should know

- `npm start` runs code completely offline and in memory
- `npm run deploy` ships to `staging`
- `ARC_DEPLOY=production npm run deploy` promotes code to `production`
- Safely use the console tactically to access and admin deployed infra
- Format, parser and tooling are completely open to extension

Before `.arc`

1. Super long dev/depoy cycles
2. Inconsistent infra leading to hard to trace bugs
3. Frustration, demotivation, unhappiness and a sense of doom

After `.arc`

1. Dev cycles are immediate, deployments run in secondss
2. Infra is consistent, bugs are easy to trace (and isolated!)
3. Green builds and happy devs

arc.codes