Tooling in the age of serverless computing

Donna Malayeri
Product and Community Manager, Pulumi
@lindydonna

In 2008, AWS had 4 services

Products

Infrastructure Services

- **Amazon Elastic Compute Cloud (Amazon EC2)**
  Amazon Elastic Compute Cloud delivers scalable, pay-as-you-go compute capacity in the cloud.

- **Amazon SimpleDB**
  Amazon SimpleDB works in conjunction with Amazon S3 and Amazon EC2 to run queries on structured data in real time.

- **Amazon Simple Storage Service (Amazon S3)**
  Amazon Simple Storage Service provides a fully redundant data storage infrastructure for storing and retrieving any amount of data, at any time, from anywhere on the Web.

- **Amazon Simple Queue Service (Amazon SQS)**
  Amazon Simple Queue Service provides a hosted queue for storing messages as they travel between computers, making it easy to build automated workflow between Web services.

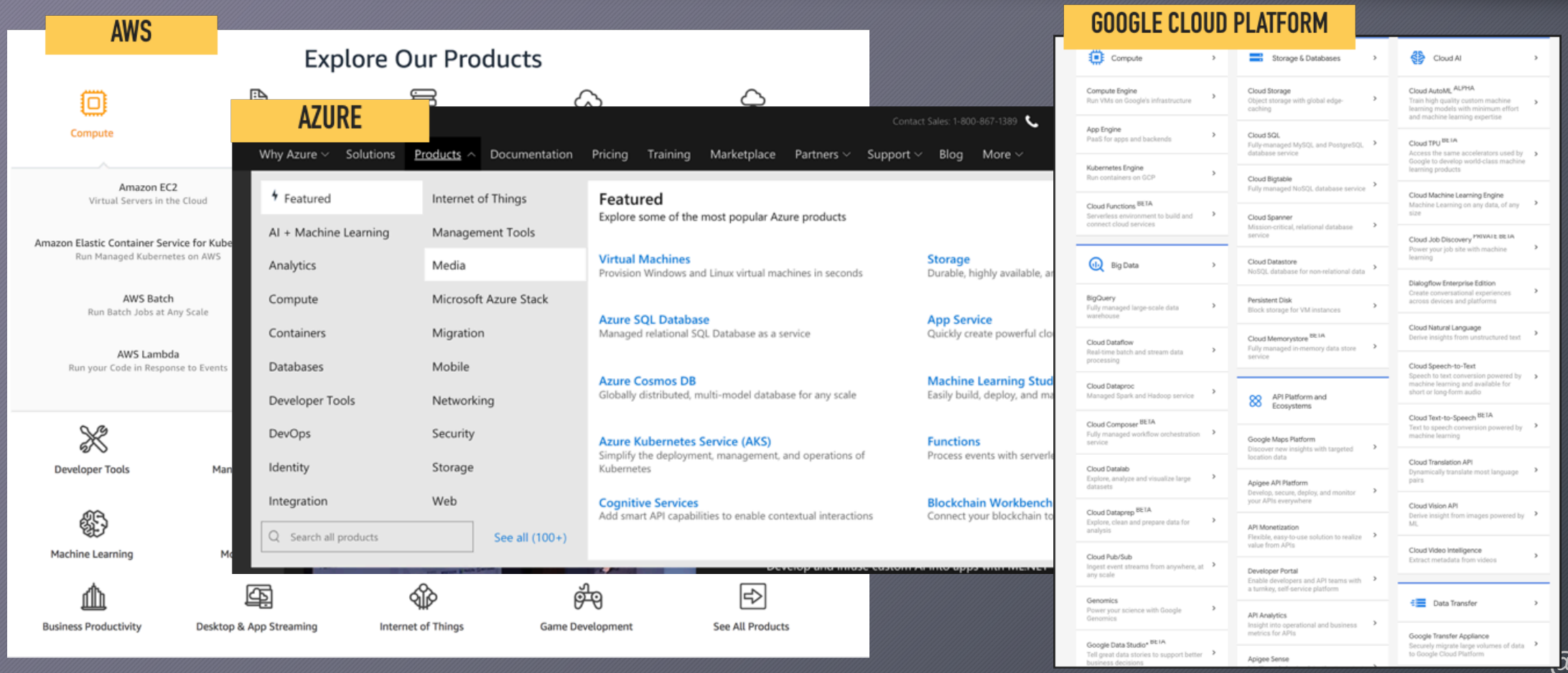
Virtual Machines

Database

Storage

Queues

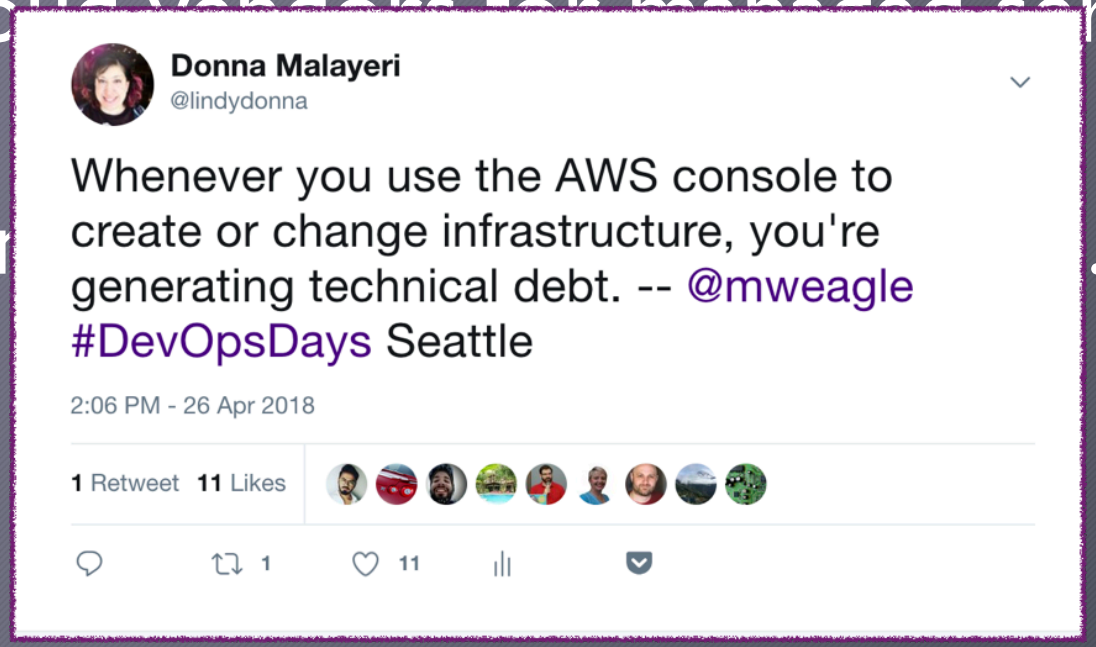
Today, each vendor has over a hundred services



#serverless? Or #servicefull?

Now, we have a new problem

- Relying on clo[illegible]rvices == net good
- Having to mar[illegible]..



Option A: use vendor's technology

AWS CLOUDFORMATION

```
    Policies:
    - PolicyDocument:
        Statement:
        - Action:
          - logs:CreateLogGroup
          - logs:CreateLogStream
          - logs:PutLogEvents
          Effect: Allow
          Resource: '*'
          Sid: AllowLogging
        Version: '2012-10-17'
      PolicyName: LambdaWriteCWLogs
    - PolicyDocument:
        Statement:
        - Action:
          - s3:Get*
          Effect: Allow
          Resource:
            Fn::Sub: arn:aws:s3:::${PhotoRepoS3Bucket}/*
          Sid: ReadFromPhotoRepoS3Bucket
        Version: '2012-10-17'
      PolicyName: ReadFromPhotoRepoS3Bucket
    - PolicyDocument:
        Statement:
        - Action:
          - s3:PutObject
          Effect: Allow
          Resource:
            Fn::Sub: arn:aws:s3:::${PhotoRepoS3Bucket}/*
          Sid: WriteToPhotoRepoS3Bucket
        Version: '2012-10-17'
      PolicyName: WriteToPhotoRepoS3Bucket
    - PolicyDocument:
        Statement:
        - Action:
          - dynamodb:UpdateItem
          - dynamodb:PutItem
          Effect: Allow
          Resource:
            Fn::Sub: arn:aws:dynamodb:${AWS::Region}:${AWS::AccountId}:table/${ImageMetadataDDBTab
          Sid: WriteToImageMetadataDDBTable
        Version: '2012-10-17'
      PolicyName: WriteToImageMetadataDDBTable
    - PolicyDocument:
        Statement:
        - Action:
          - rekognition:DetectLabels
          Effect: Allow
          Resource: '*'
          Sid: RekognitionDetectLabels
        Version: '2012-10-17'
      PolicyName: RekognitionDetectLabels
    - PolicyDocument:
        Statement:
        - Action:
          - states:StartExecution
          Effect: Allow
          Resource: '*'
          Sid: StepFunctionStartExecution
        Version: '2012-10-17'
      PolicyName: StepFunctionStartExecution
  Type: AWS::IAM::Role
CreateS3EventTriggerFunction:
  Properties:
    CodeUri: s3://media-sharing-refarch/8c7e2179b5bc3480407509396c78b95e
    Description: Used with CloudFormation as a custom resource helper to enable
      S3 event trigger to invoke the start step function Lambda function.
```

AZURE RESOURCE MANAGER

```
"variables": {
  "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'sacustmdata')]",
  "imagePublisher": "Canonical",
  "imageOffer": "UbuntuServer",
  "nicName": "networkInterface1",
  "vmName": "vm1",
  "virtualNetworkName": "virtualNetwork1",
  "publicIPAddressName": "publicIp1",
  "addressPrefix": "10.0.0.0/16",
  "subnet1Name": "Subnet-1",
  "subnet1Prefix": "10.0.0.0/24",
  "vmStorageAccountContainerName": "vhds",
  "publicIPAddressType": "Dynamic",
  "storageAccountType": "Standard_LRS",
  "vnetID": "[resourceId('Microsoft.Network/virtualNetworks',variables('virtualNetworkName'))]",
  "subnet1Ref": "[concat(variables('vnetID'),'/subnets/',variables('subnet1Name'))]",
  "apiVersion": "2015-06-15"
},
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[variables('storageAccountName')]",
    "apiVersion": "2015-06-15",
    "location": "[resourceGroup().location]",
    "properties": {
      "accountType": "[variables('storageAccountType')]"
    }
  },
  {
    "apiVersion": "2015-06-15",
    "type": "Microsoft.Network/publicIPAddresses",
    "name": "[variables('publicIPAddressName')]",
    "location": "[resourceGroup().location]",
    "properties": {
      "publicIPAllocationMethod": "[variables('publicIPAddressType')]",
      "dnsSettings": {
        "domainNameLabel": "[parameters('dnsLabelPrefix')]"
      }
    }
  },
  {
    "apiVersion": "2015-06-15",
    "type": "Microsoft.Network/virtualNetworks",
    "name": "[variables('virtualNetworkName')]",
    "location": "[resourceGroup().location]",
    "properties": {
      "addressSpace": {
        "addressPrefixes": [
          "[variables('addressPrefix')]"
        ]
      },
      "subnets": [
        {
          "name": "[variables('subnet1Name')]",
          "properties": {
            "addressPrefix": "[variables('subnet1Prefix')]"
          }
        }
      ]
    }
  },
  {
    "apiVersion": "2015-06-15",
    "type": "Microsoft.Network/networkInterfaces",
    "name": "[variables('nicName')]",
```

GOOGLE CLOUD DEPLOYMENT MANAGER

```
imports:
  - path: path/to/my_vm_template.jinja
    name: my_renamed_template.jinja
  - path: special_vm.py
```

If your template uses other templates as dependencies, import the dependent templates in your configuration as well:

```
imports:
  - path: path/to/my_vm_template.jinja
  - path: special_vm.py
  - path: base_vm.jinja
```

You can also import text files in order to inline the content. For example, if you create a file named resource_type.txt with the following string:

```
compute.v1.instance
```

Import it into your configuration and provide the content inline like so:

```
imports:
- path: resource_type.txt

resources:
- name: my-vm
  type: {{ imports["resource_type.txt"] }} # Resolves to "compute.v1.instance"
  properties:
    zone: us-central1-a
    machineType: zones/us-central1-a/machineTypes/f1-micro
    disks:
    - deviceName: boot
      type: PERSISTENT
      boot: true
      autoDelete: true
      initializeParams:
        sourceImage: projects/debian-cloud/global/images/family/debian-8
    networkInterfaces:
    - network: global/networks/default
      accessConfigs:
      - name: External NAT
        type: ONE_TO_ONE_NAT
```

Option B: use a deployment tool

When should you use a tool?

- Your team does not have expertise in the vendor technology
  - Or, some team members don't have this expertise
- You're comfortable taking a new dependency, if it buys productivity
- You have a small team and can't afford to specialize

A framework for evaluating tools

Serverless deployment tools

Option B: use a deployment tool

Option B: use a deployment tool

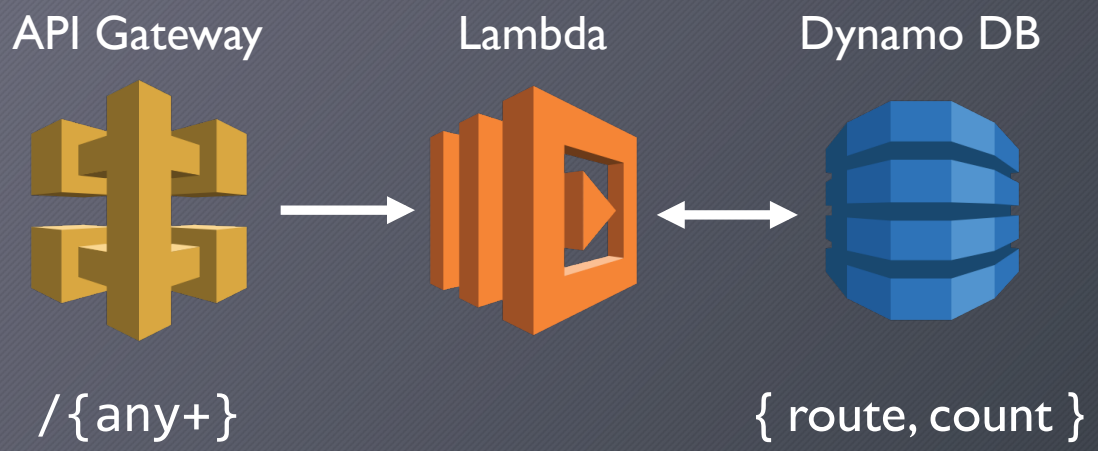
More at github.com/lindydonna/velocity-examples

Count the number of times a route has been hit

```
$ curl $URL/foo
{"id":"foo","count":1}

$ curl $URL/foo
{"id":"foo","count":2}

$ curl $URL/bar
{"id":"bar","count":1}
```



Serverless Framework

- *Serverless is your toolkit for deploying and operating serverless architectures. Focus on your application, not your infrastructure.*

- First class experience for serverless functions on AWS, Azure, GCP, and more
- Manages code *and* infrastructure
- Can author functions in JavaScript, Python, Java, and more
- Rich ecosystem of community plugins

Serverless Framework example

45 LOC for config

30 LOC for Lambda

```
RoutesDynamoDbTable:
  Type: 'AWS::DynamoDB::Table'
  DeletionPolicy:
  Properties:
    AttributeDefin
      -
        AttributeNa
        AttributeTyp
    KeySchema:
      -
        AttributeNa
        KeyType: H
    ProvisionedTh
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
    TableName: ${self:provider.environment.DYNAMODB_TABLE}
```

```
const AWS = require('aws-sdk’);

const dynamoDb = new AWS.DynamoDB.DocumentClient();

module.exports.handler = (event, context, callback) => {
  ...
}
```

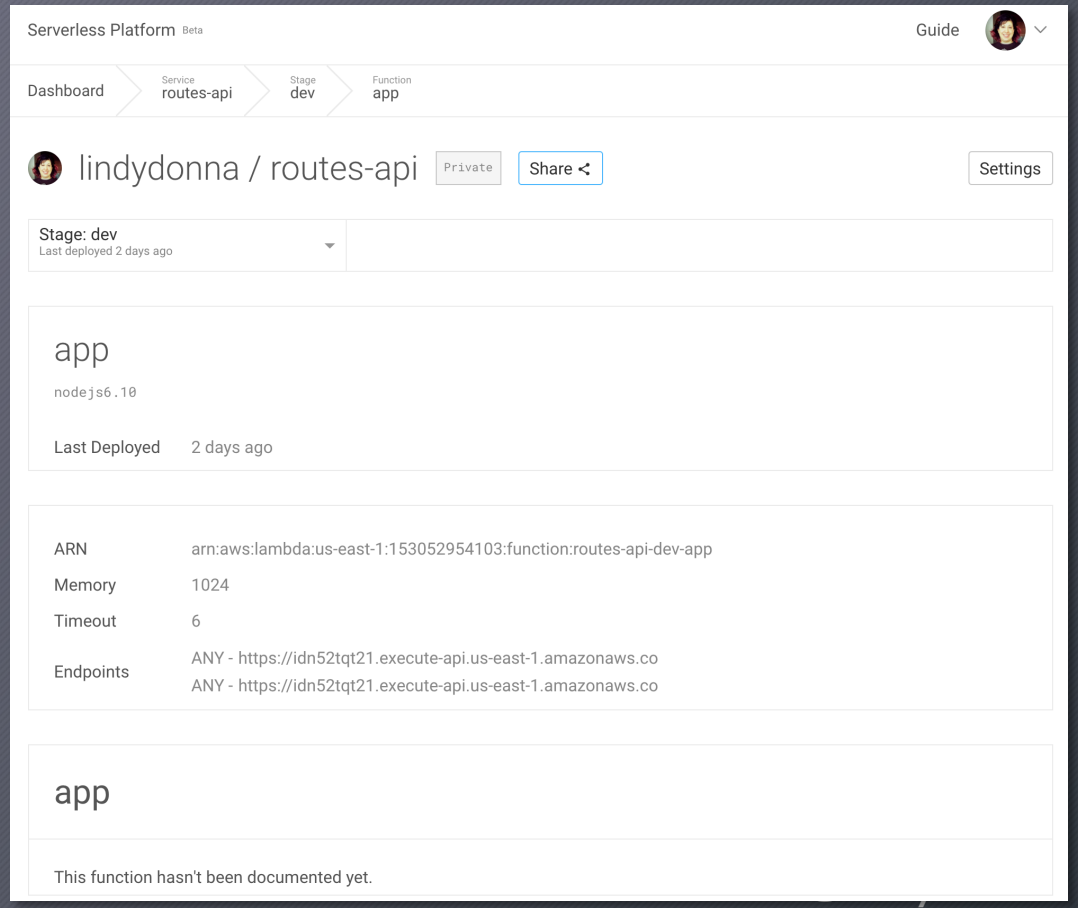
Deploying with Serverless Framework

`sls deploy` - Uses vendor's deployment service

- Such as CloudFormation on AWS

After deployment

- `sls logs` - View logs for a specific function
- CloudFormation console



Serverless Framework

How do I describe infrastructure?

- Lambda and APIG are first-class
- For everything else, use CloudFormation

How do I write my app?

- No change to how Lambda is authored

How do I deploy?

- sls deploy, uses CloudFormation

How do I learn?

- An active community
- Lots of plugins and tutorials

Terraform

- *HashiCorp Terraform enables you to safely and predictably create, change, and improve infrastructure. Define infrastructure as code to increase operator productivity and transparency.*

- Defines a custom language for describing deployments to AWS, Azure, etc
- No special support for serverless scenarios

Terraform example

```
data "archive_file" "lambda" {
  type        = "zip"
  source_file = "app.js"
  output_path = "lambda.zip"
}

resource "aws_lambda_function" "example" {
  filename      = "${data.archive_file.lambda.output_path}"
  function_name = "ServerlessExample"

  handler = "app.handler"
  runtime = "nodejs8.10"
  role             = "${aws_iam_role.lambda_exec.arn}"
  source_code_hash = "${base64sha256(file("${data.archive_file.lambda.output_path}"))}"
  publish          = true
  environment {
    variables = {
      DYNAMODB_TABLE = "${aws_dynamodb_table.dynamo-table.id}"
    }
  }
}
```

150 LOC for config

30 LOC for Lambda

Deploying with Terraform

terraform apply

- Uses vendor APIs directly, does not use services like CloudFormation
  - Has a checkpoint file to store state of last deployment
  - In free version, you manage this yourself

After deployment

- No specific tooling for viewing resources or getting logs

Terraform

How do I describe infrastructure?	How do I write my app?	How do I deploy?	How do I learn?
• Use HCL to configure any cloud resource on any cloud • Can author your own provider	• No change to how Lambda is authored	• Uses vendor APIs directly • You have to manage a state file	• Lots of tutorials and examples for infrastructure • Less common for serverless

Claudia.js

- *Claudia makes it easy to deploy Node.js projects to AWS Lambda and API Gateway. It automates all the error-prone deployment and configuration tasks, and sets everything up the way JavaScript developers expect out of the box.*

- Supports JavaScript on AWS only
- Automatically sets up CORS, request/response format, etc
- With API builder, use API Gateway with regular JavaScript web server syntax
- Manages only API Gateway and Lambda resources

Claudia.js example

Create Dynamo table manually

```
aws dynamodb create-table --table-name RouteCounts ...
```

```
{
  "DYNAMODB_TABLE"
}
```

```
claudia crea
--api-module index --policies policy \
--set-env-from-json env.json
```

policy.json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
```

18 LOC for config

```
const ApiBuilder = require('claudia-api-builder');
var api = new ApiBuilder();

api.get('/{proxy+}', function (request) {
  ...
});
```

Deploying with Claudia.js

- Deploy new app with `claudia create`
  - Uses AWS APIs directly, creates a JSON file
- Deploy code with `claudia update`

After deployment

- No specific tooling for viewing resources or getting logs

Claudia.js

How do I describe infrastructure?	How do I write my app?	How do I deploy?	How do I learn?
• JSON for APIG and Lambda • Anything else, use an additional tool	• Convenient syntax for defining APIs	• Uses AWS APIs directly • You have to manage a json state file	• Lots of tutorials and examples • Upcoming book later this year

pulumi

Create containers, serverless functions, and cloud infrastructure, with all the benefits of immutable infrastructure, and real programming languages.

- For cloud configuration, use JavaScript, Python or Go
- For functions and containers, use any supported language
- Manage any resource in AWS, Azure, Kubernetes, or GCP
- Use low-level resources or higher-level framework

Pulumi example: low level

```
const aws = require("@pulumi/aws")

const policy = { ... };

let deployment = new aws.apigateway.Deployment("api-deployment", {
    restApi: restApi,
    stageName: "",
});

let stage = new aws.apigateway.Stage("api-stage", {
    restApi: restApi,
    deployment: deployment,
    stageName: stageName
});

let invokePermission = new aws.lambda.Permission("api-lambda-permission", {
    action: "lambda:invokeFunction",
    function: lambda,
    principal: "apigateway.amazonaws.com",
    sourceArn: deployment.executionArn.apply(arn => arn + "*/*"),
});
```

110 LOC for config

30 LOC for Lambda

Pulumi example: using @pulumi/aws-serverless

```
const aws = require("@pulumi/aws");
cons
    const policy = { ... };
cons    const role = new aws.iam.Role(...);
    const fullAccess = new aws.iam.RolePolicyAttachment(...);

    let lamb
        code
        role
        hand
        runt
});      environment: {
            variables: {
              "DYNAMODB_TABLE": table.name
            }
        },
    });
```

```
const api = new serverless.apigateway.API("myapi", {
  routes: [
    { method: "GET", path: "/{proxy+}", handler: lambda },
  ],
});
```

54 LOC for config

30 LOC for Lambda

Pulumi example: define Lambda *inline*

```
// The handler for GET /{route+}.
const getHandler = async (event) => {
  const awssdk = require("aws-sdk");
  const dynamo = new awssdk.DynamoDB.DocumentClient();

  const tableName = table.name.get();
  ...
}

const api = new serverless.apigateway.API("api", {
  routes: [
    { method: "GET", path: "/{route+}", handler: getHandler }
  ]
});
```

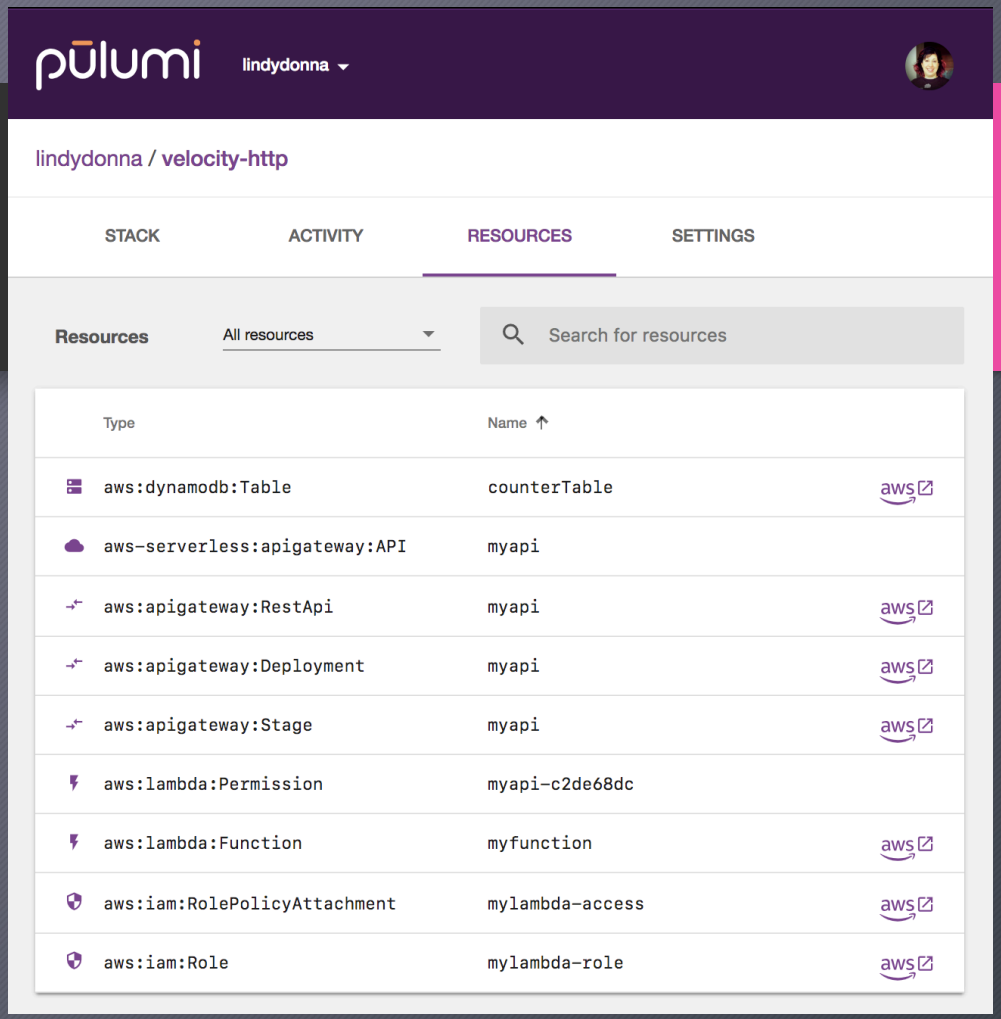
52 LOC for config *AND* Lambda

Reference table object directly

Deploying with Pulumi

pulumi update

- Uses vendor APIs directly
- Deployment state is stored on pulumi.com



After deployment

pulumi logs

```
$ pulumi logs -f
2018-05-25T12:57:30.705-07:00[            ] Running thumbnailer task.
2018-05-25T12:58:34.960-07:00[            ] Starting ffmpeg task...
```

Pulumi

How do I describe infrastructure?

- Use JavaScript, Python or Go
- First class support for containers and serverless
- Target AWS, Azure, GCP, k8s

How do I write my app?

- Can author as regular Lambda
- Or inline with infra code

How do I deploy?

- Uses vendor APIs directly
- State file is saved on pulumi.com

How do I learn?

- Still in beta, but lots of examples and tutorials

Summary

- For a more productive cloud experience, consider a deployment tool
- Answer these questions
  - *How do I describe infrastructure?*
  - *How do I write my app?*
  - *How do I deploy?*
  - *How do I learn?*

- All examples available at github.com/lindydonna/velocity-examples
- Try out Pulumi at www.pulumi.com