For Immutable Infrastructure, Real Code beats DSLs

Donna Malayeri
Product Manager, Pulumi
@lindydonna

aws.amazon.com in 2008

Products

Infrastructure Services

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

Virtual Machines

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

Database

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.

Storage

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.

Queues

Each vendor has over a hundred services

AWS

AZURE

GOOGLE CLOUD PLATFORM

Explore Our Pro

Why Azure | Solutions | Products | Documentation | Pricin

Featured
AI + Machine Learning
Analytics
Compute
Containers
Databases
Developer Tools
DevOps
Identity
Integration
Internet of Things
Management Tools
Media
Microsoft Azure Stack
Migration
Mobile
Networking
Security
Storage
Web
Search all products
See all (100+)

Machine Learning
Mobile Services
AR & VR
Business Productivity
Desktop & App Streaming
Internet of Things

Compute
Compute Engine — Run VMs on Google's infrastructure
App Engine — PaaS for apps and backends
Kubernetes Engine — Run containers on GCP
Cloud Functions BETA — Serverless environment to build and connect cloud services

Big Data
BigQuery — Fully managed large-scale data warehouse
Cloud Dataflow — Real-time batch and stream data processing
Cloud Dataproc — Managed Spark and Hadoop service
Cloud Composer BETA — Fully managed workflow orchestration service
Cloud Datalab — Explore, analyze and visualize large datasets
Cloud Dataprep BETA — Explore, clean and prepare data for analysis
Cloud Pub/Sub — Ingest event streams from anywhere, at any scale
Genomics — Power your science with Google Genomics
Google Data Studio* BETA — Tell great data stories to support better business decisions

Storage & Databases
Cloud Storage — Object storage with global edge-caching
Cloud SQL — Fully-managed MySQL and PostgreSQL database service
Cloud Bigtable — Fully managed NoSQL database service
Cloud Spanner — Mission-critical, relational database service
Cloud Datastore — NoSQL database for non-relational data
Persistent Disk — Block storage for VM instances
Cloud Memorystore BETA — Fully managed in-memory data store service

API Platform and Ecosystems
Google Maps Platform — Discover new insights with targeted location data
Apigee API Platform — Develop, secure, deploy, and monitor your APIs everywhere
API Monetization — Flexible, easy-to-use solution to realize value from APIs
Developer Portal — Enable developers and API teams with a turnkey, self-service platform
API Analytics — Insight into operational and business metrics for APIs
Apigee Sense

Cloud AI
Cloud AutoML ALPHA — Train high quality custom machine learning models with minimum effort and machine learning expertise
Cloud TPU BETA — Access the same accelerators used by Google to develop world-class machine learning products
Cloud Machine Learning Engine — Machine Learning on any data, of any size
Cloud Job Discovery PRIVATE BETA — Power your job site with machine learning
Dialogflow Enterprise Edition — Create conversational experiences across devices and platforms
Cloud Natural Language — Derive insights from unstructured text
Cloud Speech-to-Text — Speech to text conversion powered by machine learning and available for short or long-form audio
Cloud Text-to-Speech BETA — Text to speech conversion powered by machine learning
Cloud Translation API — Dynamically translate most language pairs
Cloud Vision API — Derive insight from images powered by ML
Cloud Video Intelligence — Extract metadata from videos

Data Transfer
Google Transfer Appliance — Securely migrate large volumes of data to Google Cloud Platform

To use the cloud most effectively, use managed services

To move quickly, application devs need to understand the cloud

- To have the fastest time-to-market, must eliminate silos between dev and ops team
- To meet business goals, application developers need to deeply understand cloud services

More managed services: yay! Now we have to manage them… 😐

Create a Simple Lambda Function

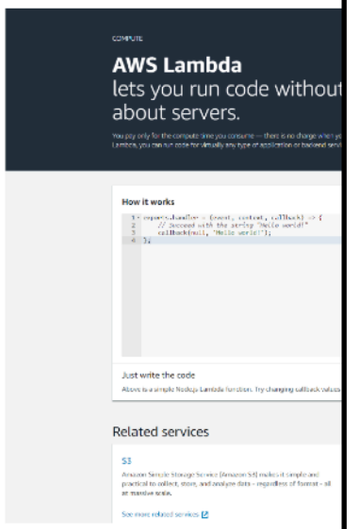
Invoke the Lambda Function

Follow the steps in this section to create a simple Lambda function.

To create a Lambda function

1. Sign in to the AWS Management Console and open the AWS Lambda console.
2. Note that AWS Lambda offers a simple `Hello World` function upon introduction under the **How it works** label and includes a **Run** option, allowing you to invoke the function as a general introduction. This tutorial introduces additonal options you have to create, test ar… the Lambda console and provides links to ea…

   Choose **Create a function** under the **Get St…**



Note

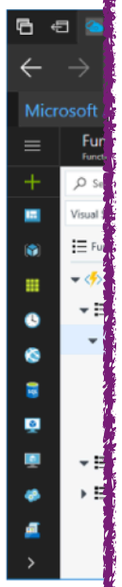
The console shows the **Get Started** pa… have created functions already, you wi… **Create a function** to go to the **Create…**

3. On the **Create function** page, you are prese…
   - **Author from scratch**
   - **Blueprints**
   - **Serverless Application Repository**

Create your first function in the Azure portal

03/28/2018 · 4 minutes to read · Contributors all

Azure Funct… web applica…



If you don't…

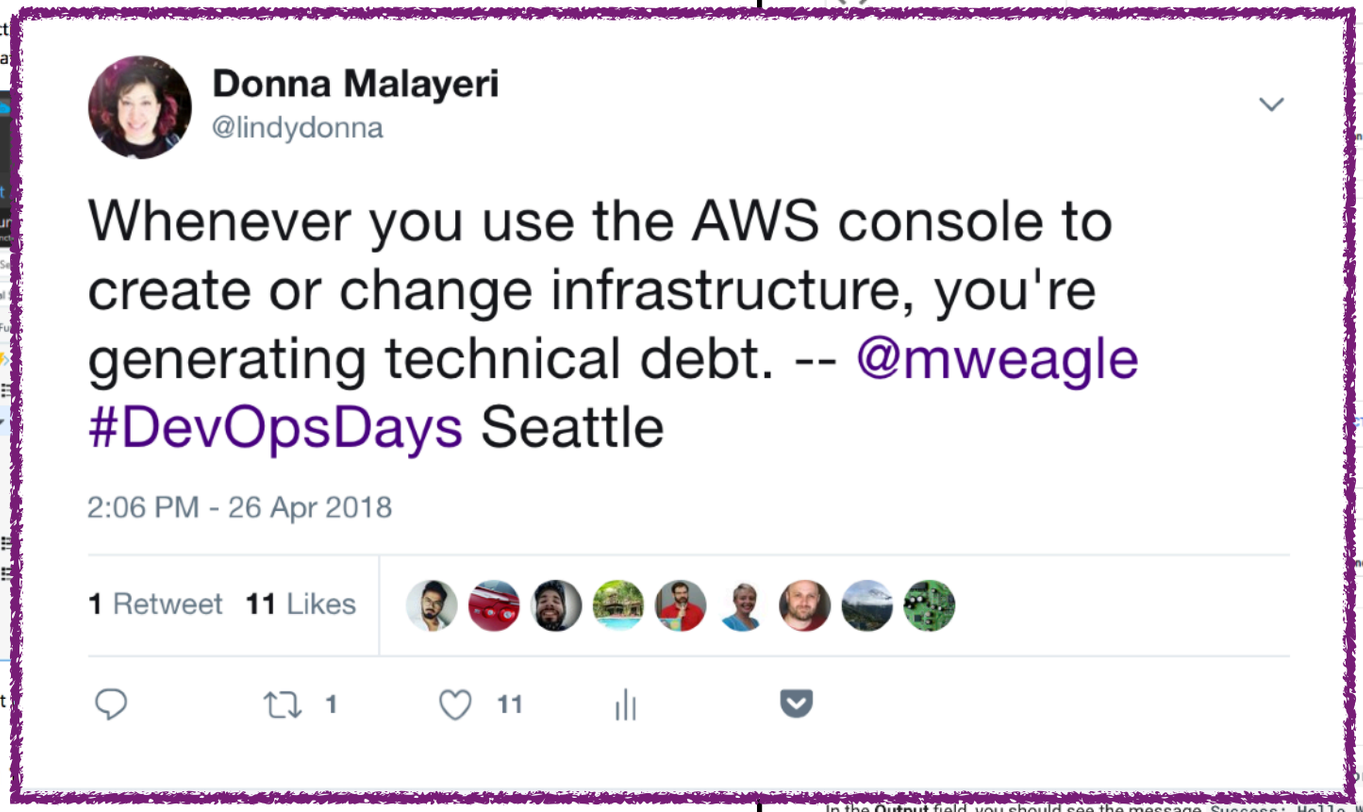
Log in …

Sign in to the Azure portal at http://portal.azure.com with your Azure account.

Create a function app

You must have a function app to host the execution of your functions. A function app lets you group fun… unit for easier management, deployment, and sharing of resources.

1. Select the **New** button found on the upper left-hand corner of the Azure portal, then select **Compu… App**.



Deploy a function

Deploy the `helloWorld` function:

1. At the bottom of the page, click **Create**.
2. After clicking **Create**, GCP Console redirects to the Cloud Functions Overview page.

   While the function is being deployed, the icon next to it is a small spinner. After it finishes deploying, the spinner turns to a green check mark:

…rld!"}, and click **Test the function**.

In the **Output** field, you should see the message `Success: Hello World!`.

In the **Logs** field, a status code of 200 indicates success.

Cloud configuration: YAML and JSON

```yaml
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}:
      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:
```

AWS CLOU

```
"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/storageAccou
    "name": "[variables('storageAccountName
    "apiVersion": "2015-06-15",
    "location": "[resourceGroup().location]
    "properties": {
      "accountType": "[variables('storageAc
    }
  },
  {
    "apiVersion": "2015-06-15",
    "type": "Microsoft.Network/publicIPAddr
    "name": "[variables('publicIPAddressNam
    "location": "[resourceGroup().location]
    "properties": {
      "publicIPAllocationMethod": "[variabl
      "dnsSettings": {
        "domainNameLabel": "[parameters('dns
      }
    }
  },
  {
    "apiVersion": "2015-06-15",
    "type": "Microsoft.Network/virtualNetwo
    "name": "[variables('virtualNetworkName
    "location": "[resourceGroup().location]
    "properties": {
      "addressSpace": {
        "addressPrefixes": [
          "[variables('addressPrefix')]"
        ]
      },
      "subnets": [
        {
          "name": "[variables('subnet1Name'
          "properties": {
            "addressPrefix": "[variables('su
          }
        }
      ]
    }
  },
  {
    "apiVersion": "2015-06-15",
    "type": "Microsoft.Network/networkInter
```

AZURE RESOURCE MANAGER

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

GOOGLE CLOUD DEPLOYMENT MANAGER

If your template uses other templates a

```
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
```

Configuration languages are hard to use

- In practice: teams specialize, where some people know how to deploy infrastructure
- This makes it harder for dev and DevOps teams to collaborate

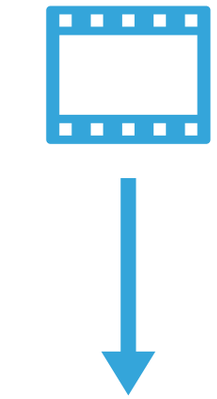
Configuration language == no reusable libraries

- Using a configuration language: copy-and-paste
- Using JavaScript: let me find an npm package that does this for me

Introducing Pulumi

- Define infrastructure in JavaScript or Python
  - Code is transformed to a declarative plan
- No extra learning curve, get all the tooling benefits of regular languages
- Can create reusable libraries
- Common language for both DevOps team and application dev team (don't have to throw code the wall!)

When new video is uploaded, extract a thumbnail



THIS IS 500 LINES OF CLOUDFORMATION

Bucket

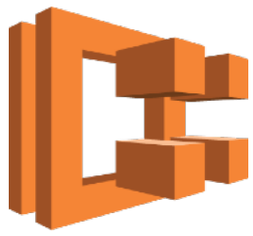
New .mp4 file

Lambda

onNewVideo

Launch task

ECS Fargate Task



ffmpegTask

Write .jpg

Bucket

New .jpg file

onNewThumbnail

Configuring Fargate manually is a lot of work

Getting Started with Amazon ECS using Fargate

Let's get started with Amazon Elastic Container Service (Amazon ECS) by creating a task definition that uses the Fargate launch type, scheduling tasks, and configuring a cluster in the Amazon ECS console.

The Amazon ECS first run wizard will guide you through the process to get started. The wizard gives you the option of creating a cluster and launching our sample web application,

Impo

Befor
Amaz

Step 1: Cre

A task defin
for containe

1. Open
2. From
3. Config

For **Co**
contai

For m

4. For **Ta**
resou

Task d
conso

For m

5. Choos

Step 2: Configure service

In this section of the wizard, you select how you would like to configure the Amazon ECS service that is created from your task definition. A service launches and maintains a specified number of copies of the task definition in your cluster. The **Amazon ECS sample** application is a web-based "Hello World" style application that is meant to run indefinitely, so by running it as a service, it will restart if the task becomes unhealthy or unexpectedly stops.

The first run w
or review and

1. In the **Se**
2. In the **Nu**
3. In the **Se**
4. (Optiona
is launch
Introduct

Imp

App
Loa

Complete

a. In t
con

b. In t

5. Review y

Step 3: Config

In this section of the wizard, you name your cluster, and then Amazon ECS take cares of the networking and IAM configuration for you.

1. In the **Cluster name** field, choose a name for your cluster.
2. Click **Next** to proceed.

Step 4: Review

1. Review your task definition, task configuration, and cluster configurations and click **Create** to finish. You are directed to a **Launch Status** page that shows the status of your launch and describes each step of the process (this can take a few minutes to complete while your Auto Scaling group is created and populated).
2. After the launch is complete, choose **View service** to view your service in the Amazon ECS console.

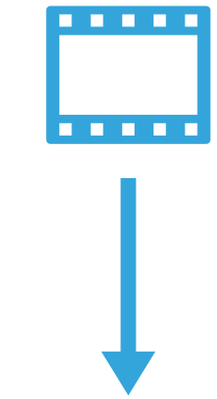
Step 5: (Optional) View your service's containers

If your service is a web-based application, such as the **Amazon ECS sample** application, you can view its containers with a web browser.

1. On the **Service:** ***service-name*** page, choose the **Tasks** tab.
2. Choose a task from the list of tasks in your service.
3. In the **Network** section, choose the **ENI Id** for your task. This will take you to the EC2 console where you can view the details of the network interface associated with your task, including it's **IPv4 Public IP** address.
4. Enter the **IPv4 Public IP** address in your web browser and you should see a web page that displays the **Amazon ECS sample** application.

AWS region

When new video is uploaded, extract a thumbnail



THIS IS 500 LINES OF CLOUDFORMATION

Bucket

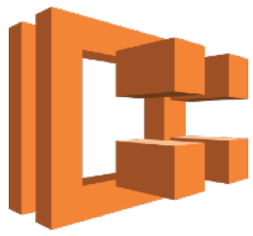
New .mp4 file

Lambda

onNewVideo

Launch task

ECS Fargate Task



ffmpegTask

Write .jpg

Bucket

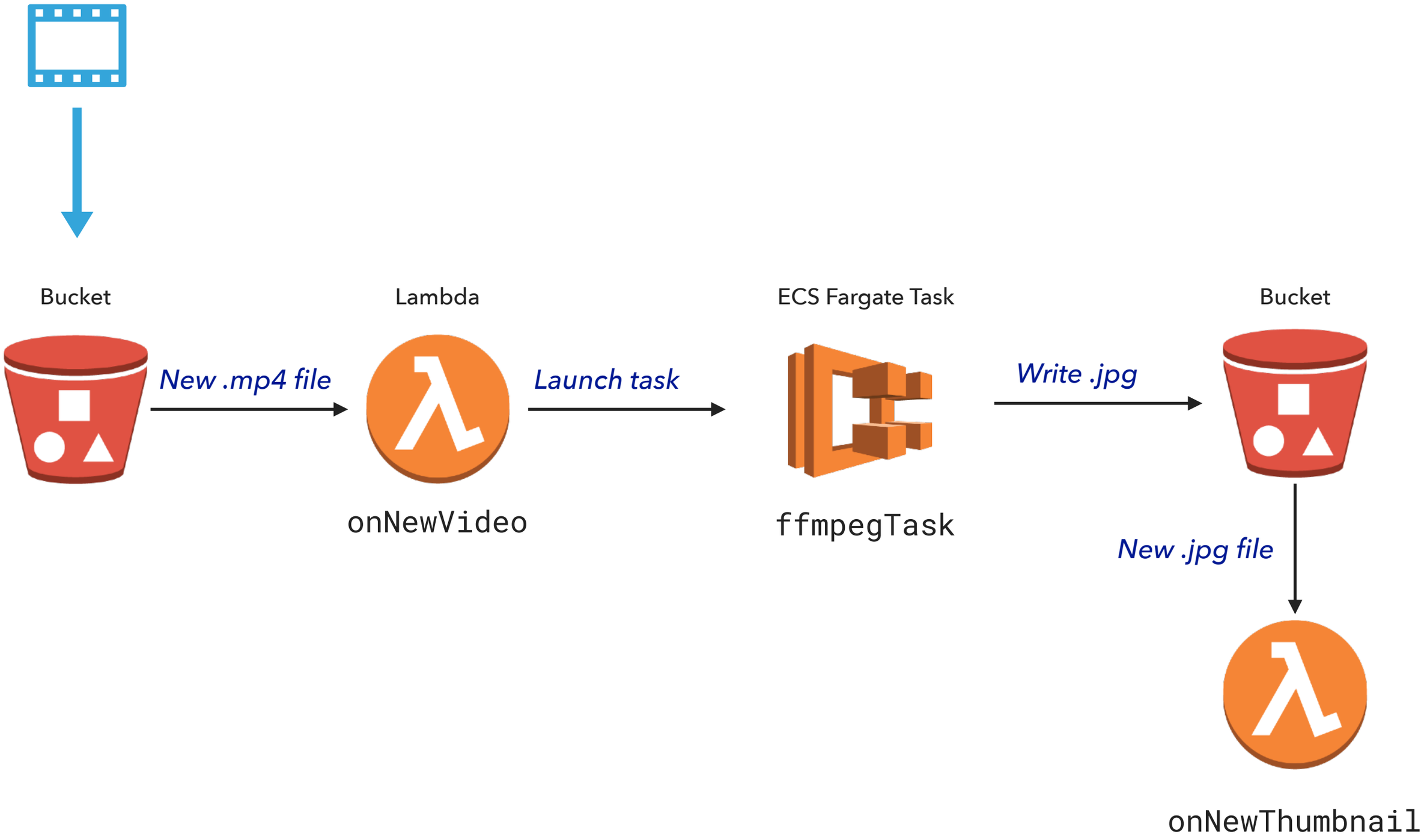
New .jpg file

onNewThumbnail

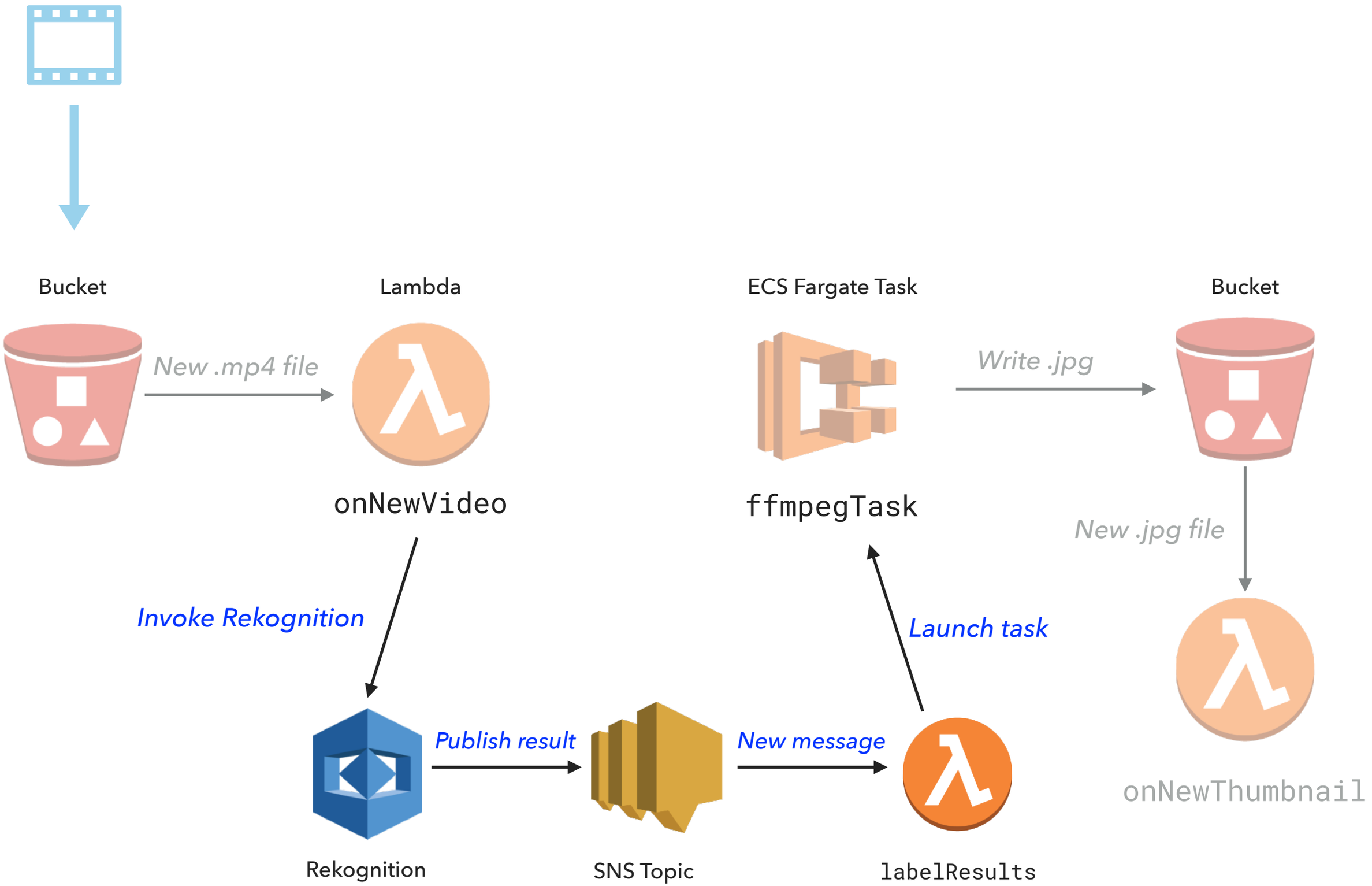
DEMO

THUMBNAILER IN PULUMI

When new video is uploaded, extract a thumbnail



Add machine learning



DEMO
ADD REKOGNITION

Pulumi: the easiest way to compose cloud applications

- Define infrastructure and (app code!) in JavaScript or Python
- Get all the tooling benefits of regular languages
  - Testing, refactoring, IDE completion, reusable libraries
- Easily compose multiple services #service-full
- Pulumi supports AWS, Azure, GCP, and Kubernetes

Sign up for the Pulumi private beta!

- Go to pulumi.com and enter your email
- Email donna@pulumi.com to fast-track your access