

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.



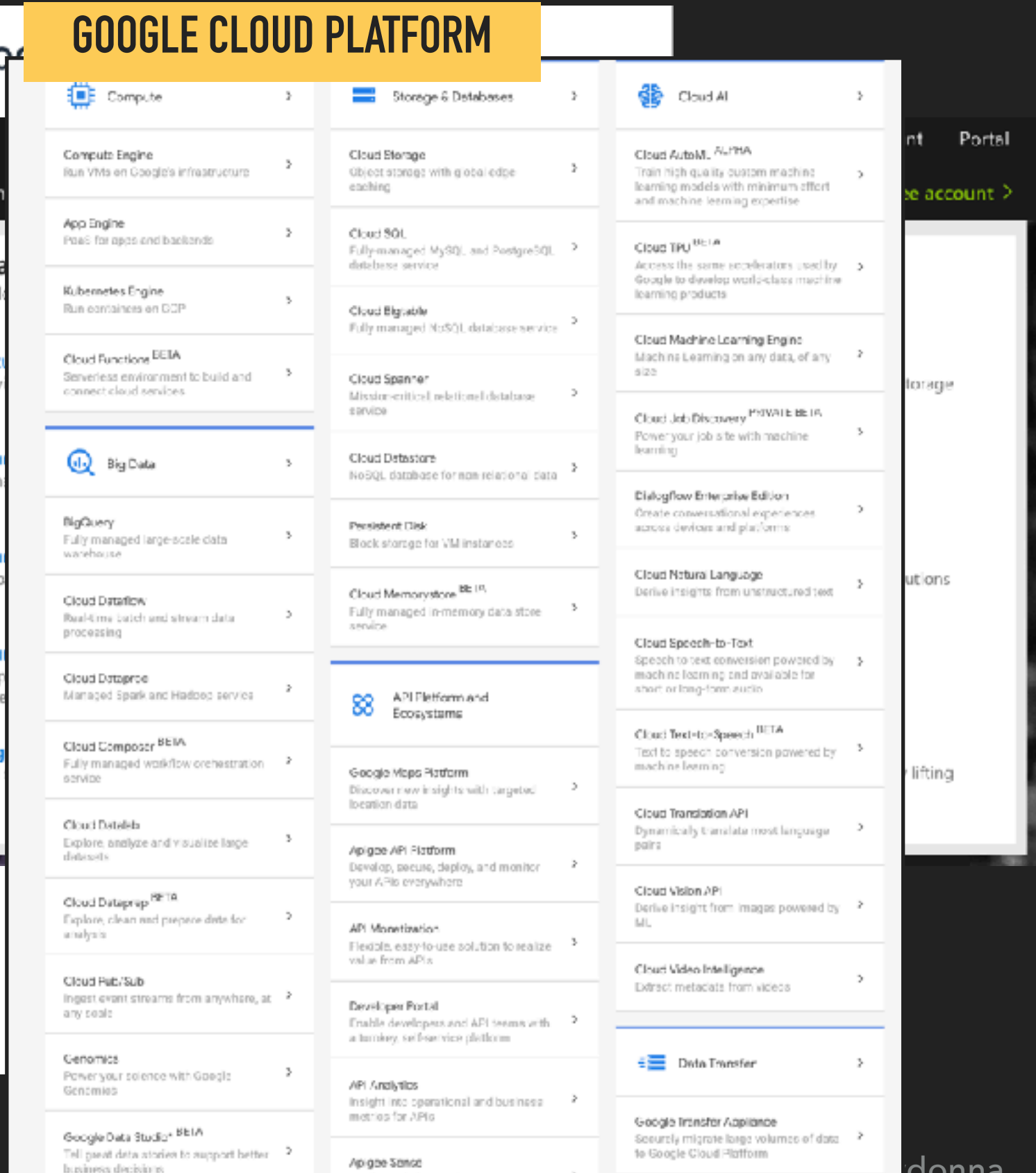
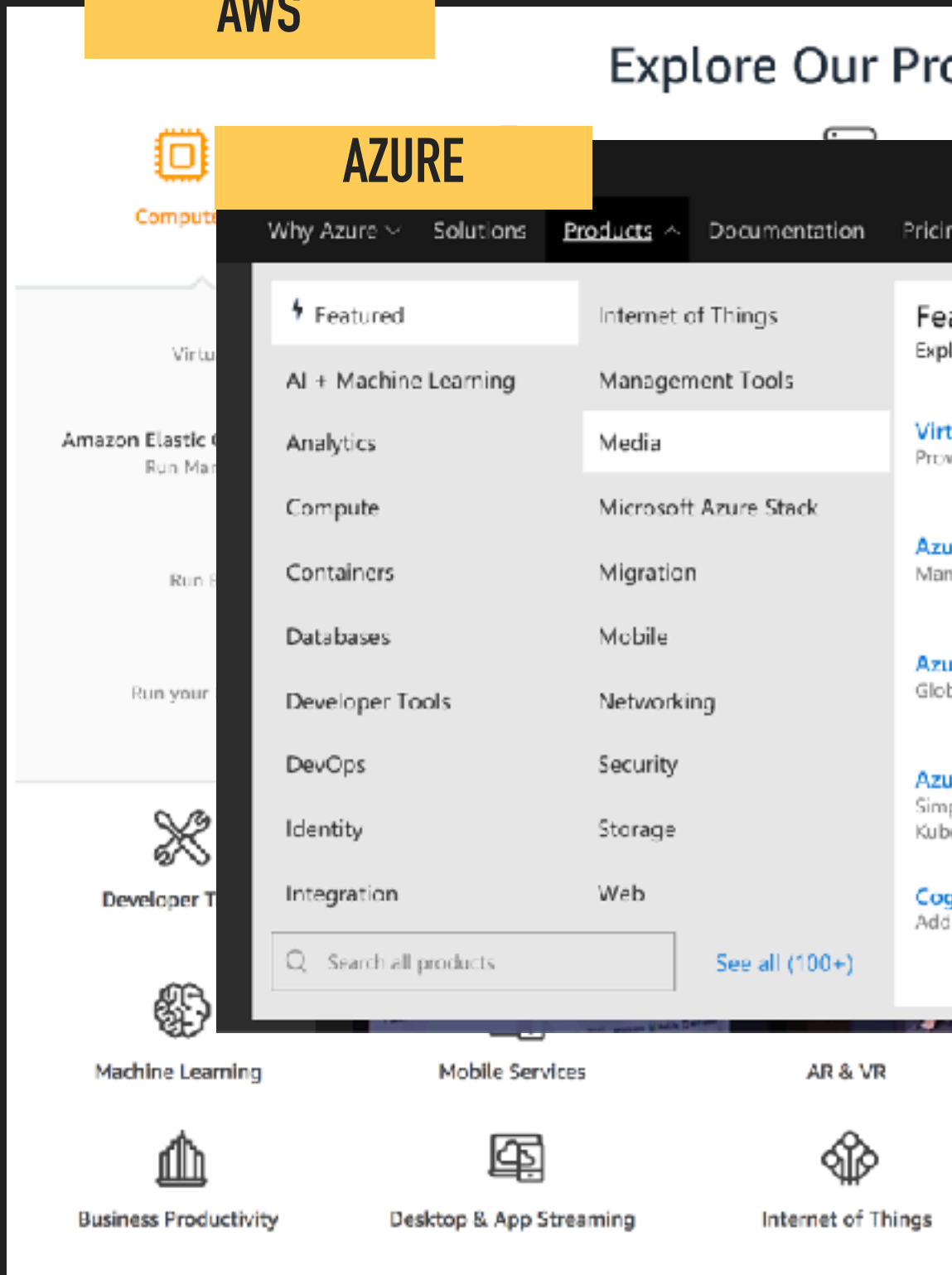
Queues

Each vendor has over a hundred services

AWS

AZURE

GOOGLE CLOUD PLATFORM



donna

~~#serverless~~... #servicefull?

To use the cloud most effectively, use managed services



Patrick Debois

@patrickdebois



I think [#serverless](#) is just a manifestation of a bigger trend that is [#servicefull](#)

[6:57 AM - May 24, 2016](#)

♡ 14 💬 16 people are talking about this



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

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 additional options you have to create, test, and deploy the Lambda **console** and provides links to explore.

Choose **Create a function** under the **Get Started** section.



Note

The **console** shows the **Get Started** page. If you have created functions already, you will see the **Create a function** button.

3. On the **Create function** page, you are presented with the following options:

- Author from scratch
- Blueprints
- Serverless Application Repository

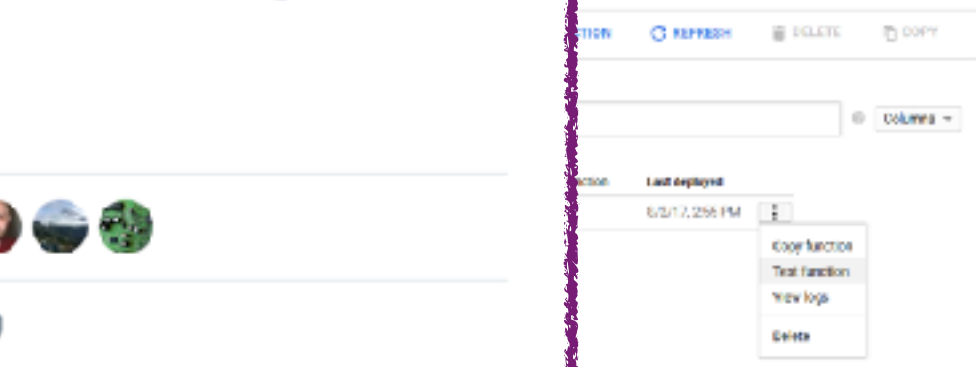
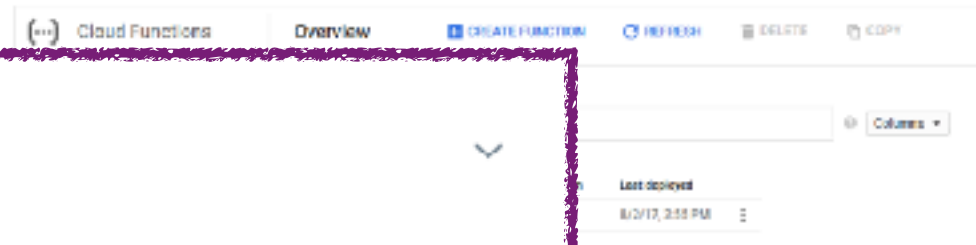
Invoke the Lambda Function

Deploy a function

Deploy the **helloWorld** function:

1. At the bottom of the page, click **Create**.
2. After clicking **Create**, the **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:



Click the **Test** button (next to the **helloWorld** function), and click **Test the function**.

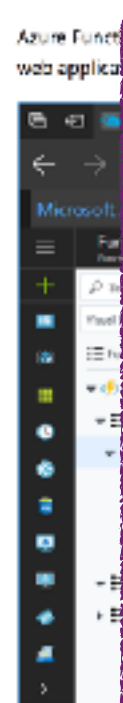
In the **Output** field, you should see the message: **Success: Hello World!**.

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



Create your first function in the Azure portal

By 10/26/2018 · 4 minutes to read · Contributors



If you don't

Log in

Sign in to the Azure **portal** at <https://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 functions 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 **Compute** > **Function App**.



Donna Malayeri

@lindydonna

Whenever you use the AWS console to create or change infrastructure, you're generating technical debt. -- @mweagle #DevOpsDays Seattle

2:06 PM - 26 Apr 2018

1 Retweet 11 Likes



Cloud configuration: YAML and JSON

AWS CLOU

AZURE RESOURCE MANAGER

GOOGLE CLOUD DEPLOYMENT MANAGER

```

Policies:
- PolicyDocument:
  Statement:
  - Action:
    - logs:CreateLogGroup
    - logs:CreateLogStream
    - logs:PutLogEvents
    Effect: Allow
    Resource: '*'
    Sid: AllowLogging
  Version: '2012-10-17'
PolicyName: LambdaWriteCWLops
- 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/${ImageMetadata}
    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:

```

```

variables: {}
'storageAccountName': '[concat(uniquestring(resourceGroup().id), 'storageacct')]',
'imagePublisher': 'Canonical',
'imageOffer': 'UbuntuServer',
'nicName': 'networkInterface',
'vmName': 'vm1',
'virtualNetworkName': 'virtualNetwork1',
'publicIPAddressName': 'publicIp1',
'addressPrefix': '10.0.0.0/16',
'subnetName': 'Subnet-1',
'subnetPrefix': '10.0.0.0/24',
'vmStorageAccountContainerName': 'vms',
'publicIPAddressType': 'Dynamic',
'storageAccountType': 'Standard_LRS',
'vnetID': '[resourceId('Microsoft.Network/virtualNetworks', variables('virtualNetworkName'))]',
'subnetID': '[concat(variables('vnetID'), '/subnets/', variables('subnetName'))]',
'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('publicIPAllocationMethod')]',
    'dnsSettings': {
      'domainNameLabel': '[parameters('domainNameLabel')]'
    }
  }
},
{
  'apiVersion': '2015-06-15',
  'type': 'Microsoft.Network/virtualNetworks',
  'name': '[variables('virtualNetworkName')]',
  'location': '[resourceGroup().location]',
  'properties': {
    'addressSpace': {
      'addressPrefixes': [
        '[variables('addressPrefix')]'
      ]
    },
    'subnets': [
      {
        'name': '[variables('subnetName')]',
        'properties': {
          'addressPrefix': '[variables('subnetPrefix')]'
        }
      ]
    ]
  }
},
{
  'apiVersion': '2015-06-15',
  'type': 'Microsoft.Network/virtualMachines',
  'name': '[variables('vmName')]',
  'location': '[resourceGroup().location]',
  'properties': {
    'hardwareProfile': {
      'vmReference': '[concat('Microsoft.Compute/images/', variables('imagePublisher'), '/', variables('imageOffer'), '/', variables('imageSKU'))]'
    },
    'storageProfile': {
      'imageReference': {
        'id': '[variables('vnetID')]'
      },
      'osDisk': {
        'createOption': 'FromImage',
        'lun': 0,
        'storageAccountType': '[variables('storageAccountType')]'
      },
      'dataDisks': [
        {
          'lun': 1,
          'storageAccountType': '[variables('storageAccountType')]'
        }
      ]
    },
    'networkProfile': {
      'networkInterfaces': [
        {
          'id': '[resourceId('Microsoft.Network/networkInterfaces', variables('nicName'))]'
        }
      ]
    },
    'diagnosticsProfile': {
      'bootDiagnostics': {
        'enabled': true,
        'storageUri': '[concat('https://', variables('storageAccountName'), '.blob.core.windows.net/', variables('diagnosticsContainerName'))]'
      }
    }
  }
}
]
}

```

Azure Resource Group Template

Google Cloud

If your template uses other templates:

```

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

```

You can also import text files in order to inline the following string:

```

compute.v1.instance

```

Import it into your configuration and provide the following string:

```

imports:
- path: resource_type.txt

resources:
- name: my-vm
  type: {{ imports('resource_type.txt') }}
  properties:
    zone: us-central1-a
    machineType: zones/us-central1-a/n1-standard-1
    disks:
    - deviceName: boot
      type: PERSISTENT
      boot: true
      autoDelete: true
      initializeParams:
        sourceImage: projects/debian-cloud/global/images/family/debian-9
    networkInterfaces:
    - network: global/networks/default
      accessConfigs:
      - name: External NAT
        type: ONE_TO_ONE_NAT

```

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

If your template uses other templates:

```
imports:
- path: path/to/my_vn_template.jinja
- path: special_vn.py
- path: base_vn.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

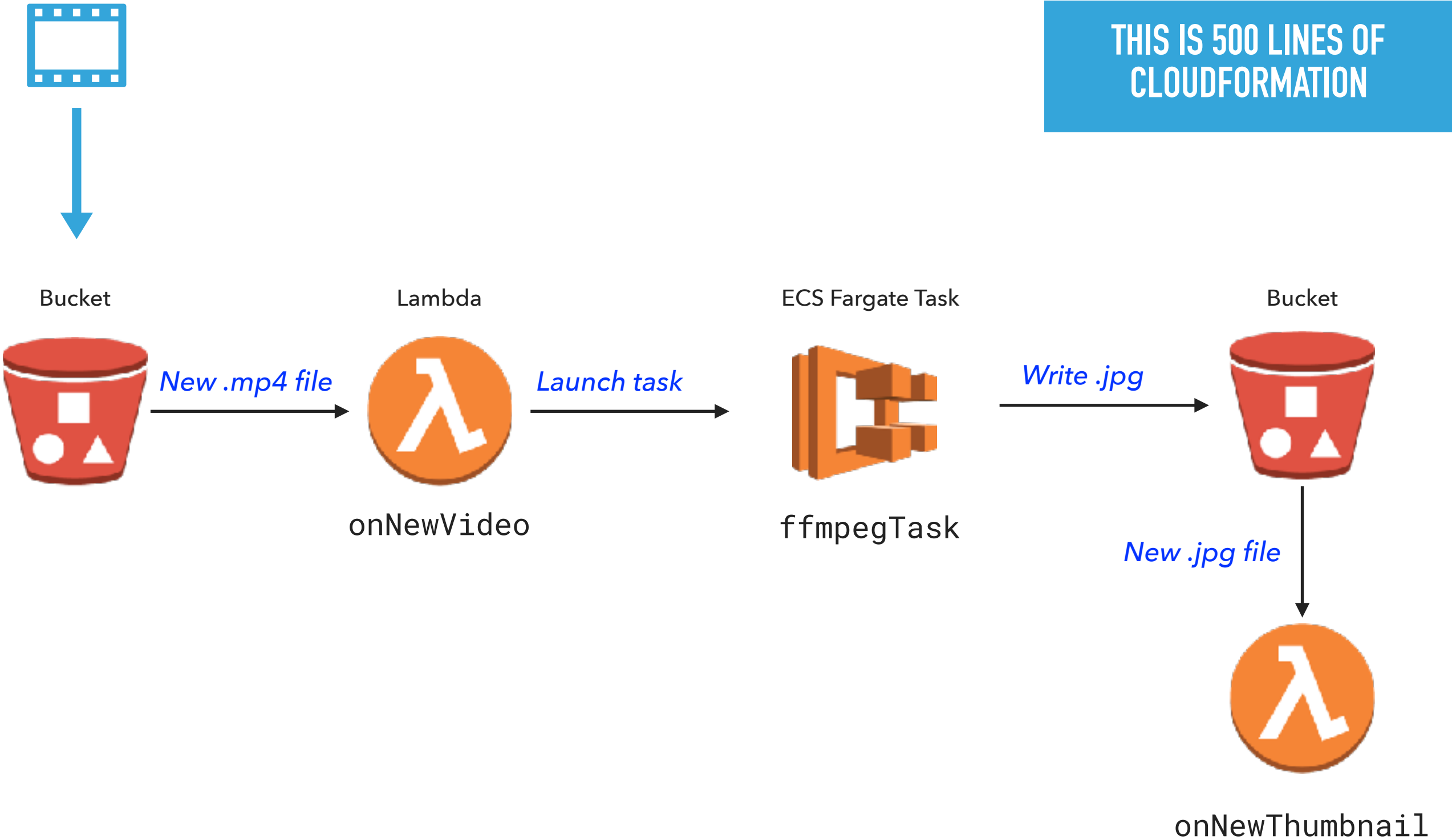
Configuration language == no reusable libraries

- ▶ 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



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. [View the Amazon ECS first run wizard](#) to learn more about the steps to get started with Amazon ECS.

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.

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 its **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.

Amazon ECS Sample App

Congratulations!

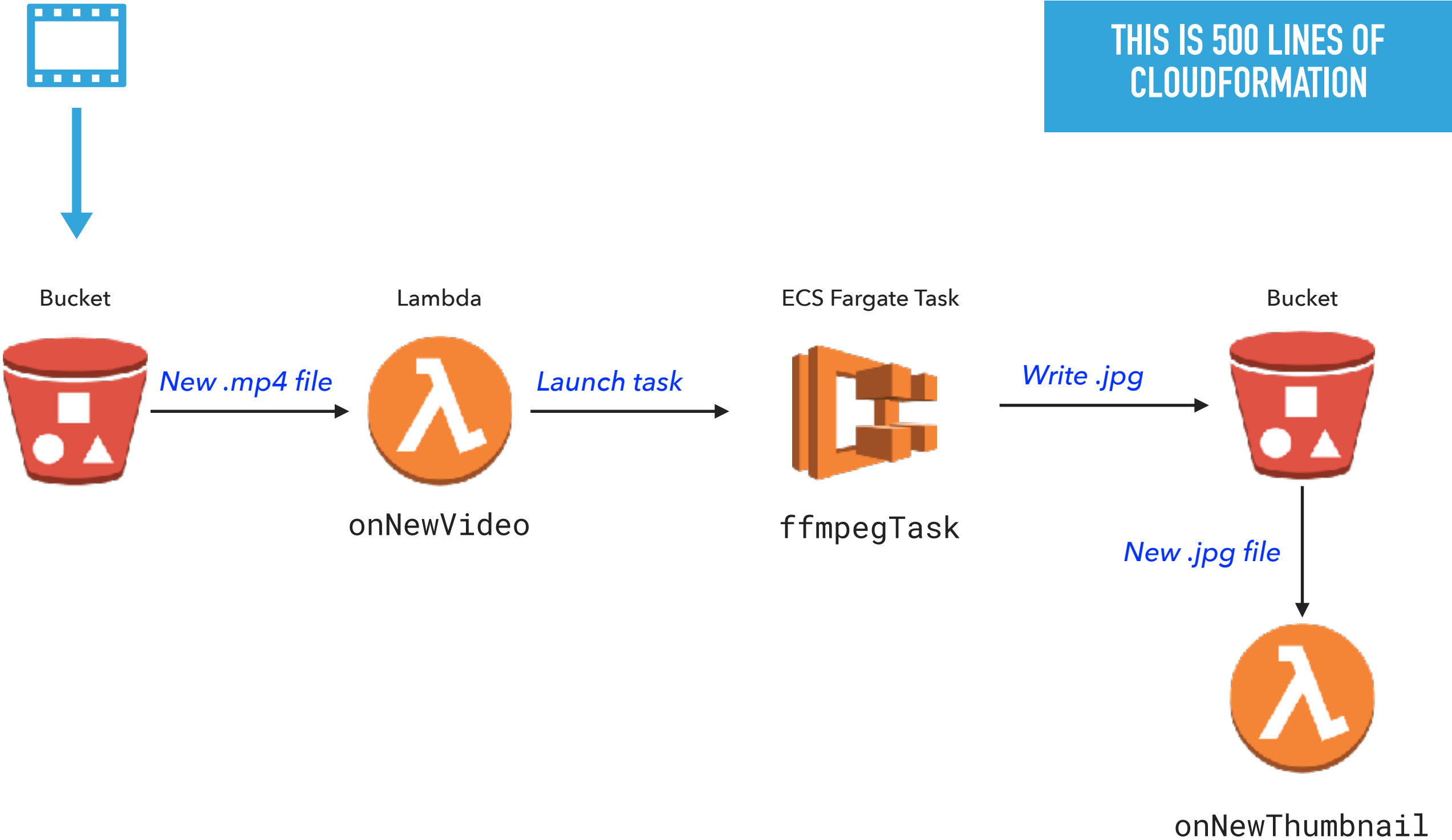
Your application is now running on a container in Amazon ECS.

Step 3: Configure cluster

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

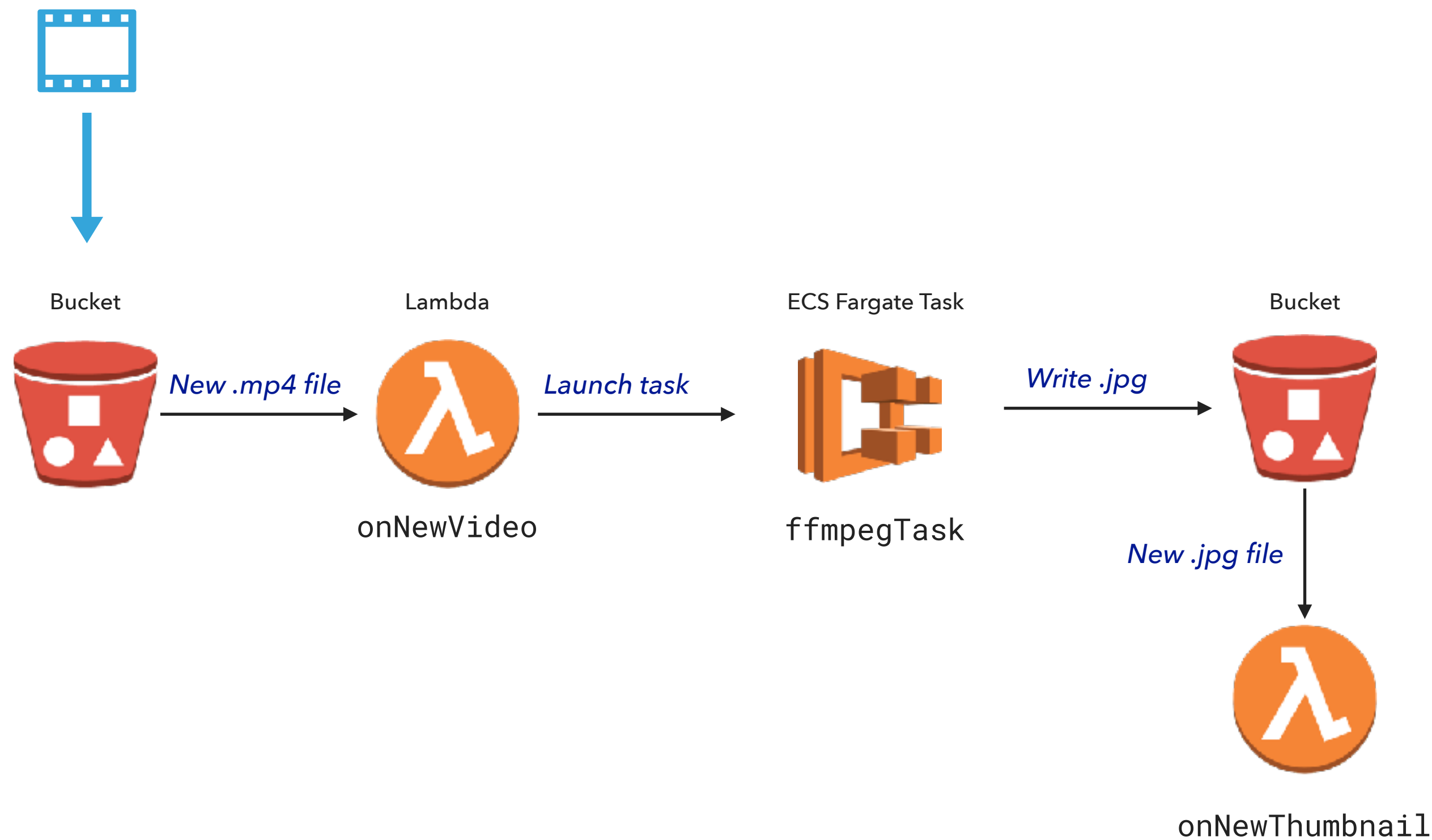
1. In the **Cluster name** field, choose a name for your cluster.
2. Click **Next** to proceed.

When new video is uploaded, extract a thumbnail

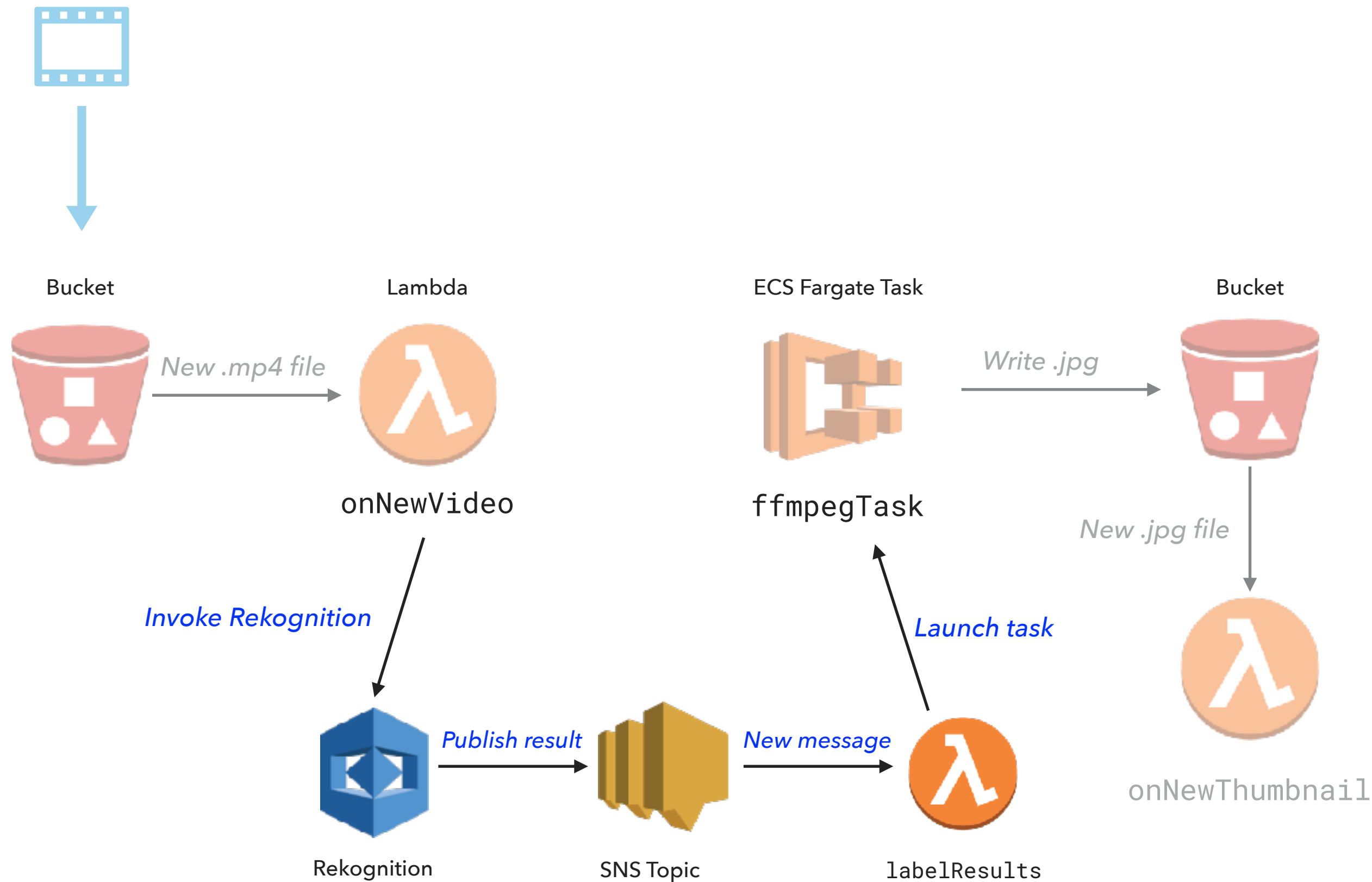


DEMO
THUMBNAILER IN PULUMI

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