

SERVERLESS + CONTAINERS = MODERN CLOUD APPLICATIONS

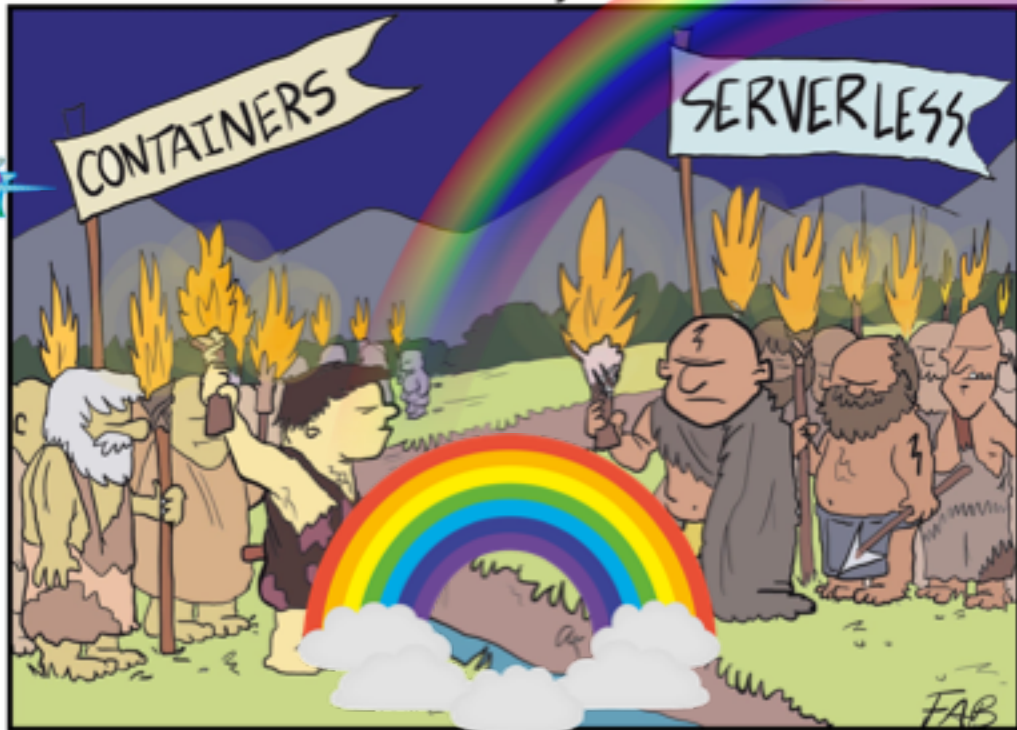
Donna Malayeri

Product Manager, Pulumi

@PulumiCorp

@lindydonna

FaaS and Furious by Forrest Brazeal



The two tribes regarded each other suspiciously in the glow of their brightly blazing production environments.



SERVERLESS AND CONTAINERS

- Tradeoff between control and productivity
- Containers give you full control over your compute workloads
- Serverless scales instantly and is cheaper to own and operate
- Modern applications need both compute models

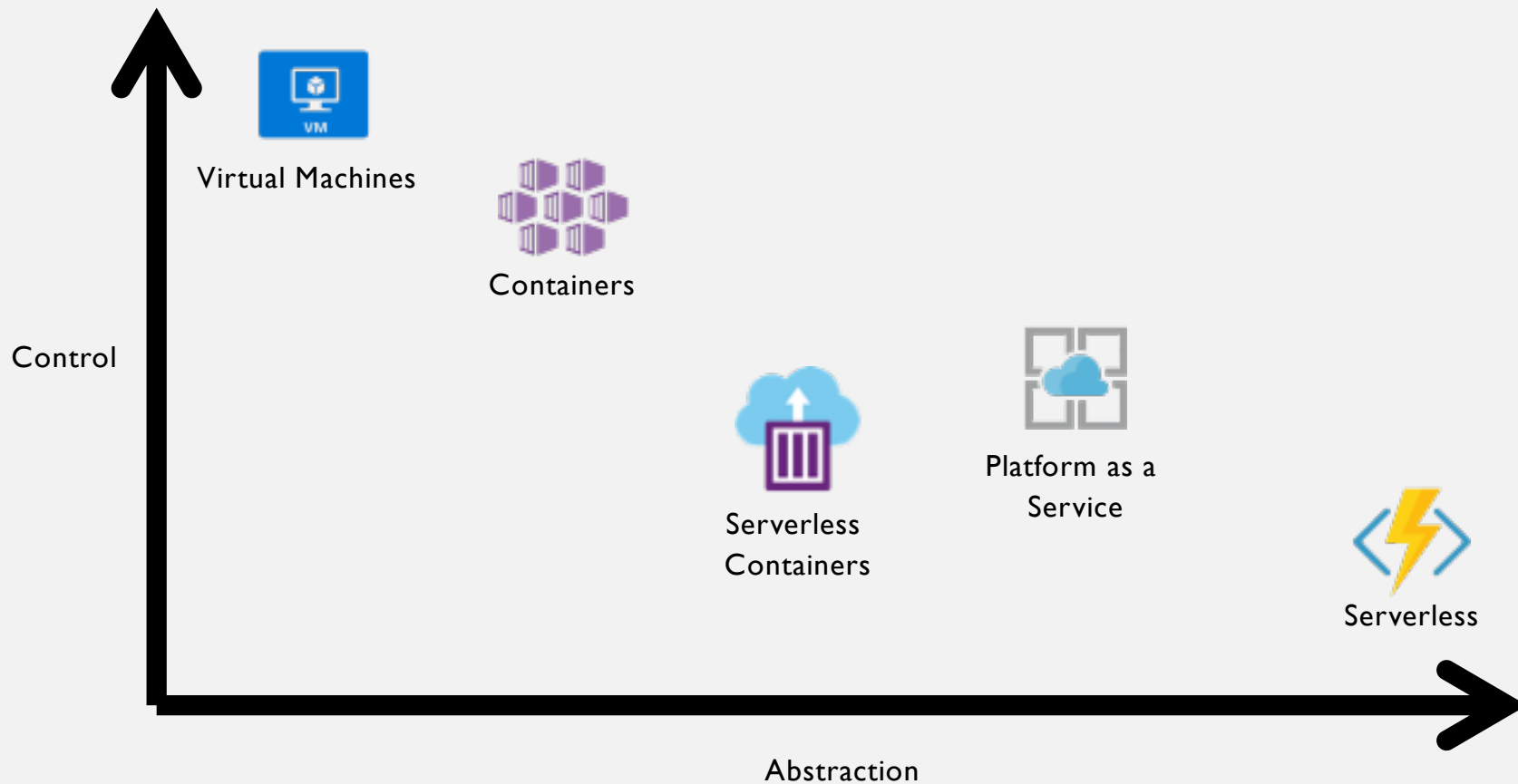
PROGRAMMING IS ABOUT ABSTRACTION

- JavaScript
- Go
- Python
- Ruby
- C#
- Java
- C/C++
- Assembly

If I have seen further it is only by standing on the shoulders of giants.

-- Isaac Newton

The cloud landscape



IN THE EARLY DAYS OF CLOUD, THERE WERE ONLY VIRTUAL MACHINES

- How often should I *patch* my server?
 - How *do* I patch?
- How do I deploy *code*?
- How *many* servers do I need?
- How can I *scale* my app?



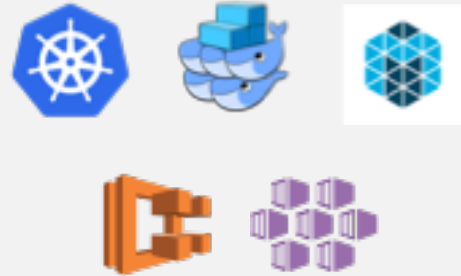
CONTAINERS REDUCE COMPLEXITY



Dockerfile



Docker image

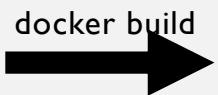


Container orchestrator

CONTAINERS



Dockerfile



Docker image



Container registry



```
FROM jrottenberg/ffmpeg
RUN apt-get update && \
    apt-get install python-dev python-pip -y && \
    apt-get clean
RUN pip install awscli
WORKDIR /tmp/workdir
ENTRYPOINT
```

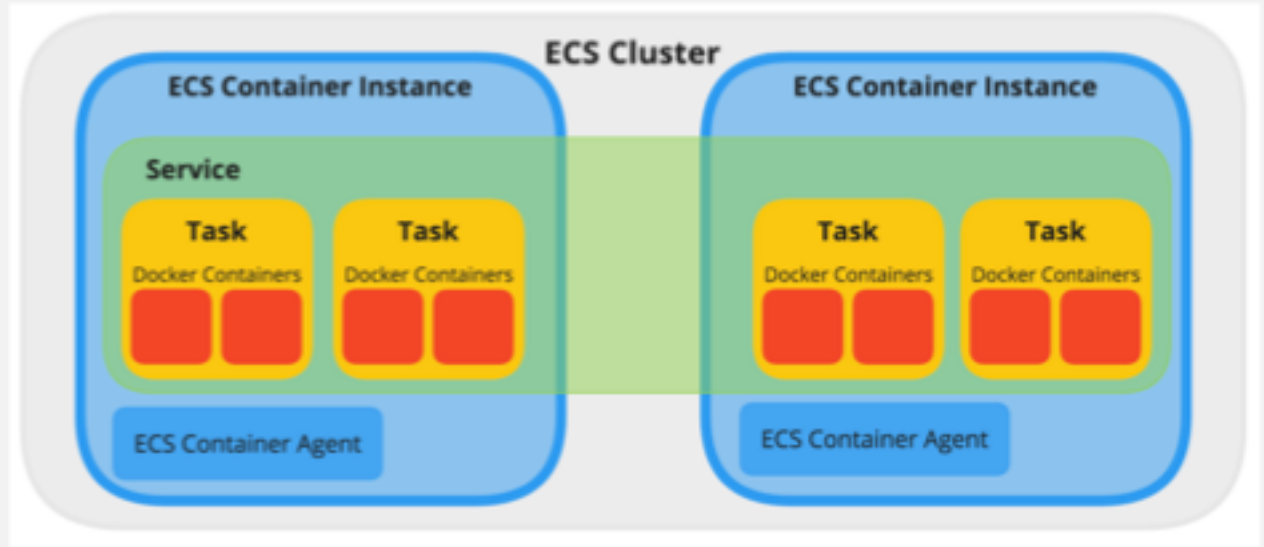
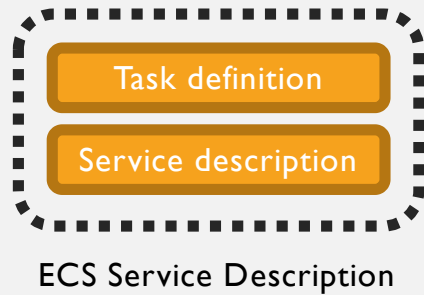


AWS Elastic Container Service

CONTAINER BENEFITS

- Abstraction for compute: containers instead of VMs
- Useful package format
- Full control over application environment
- Full control over task placement
- Control over compute resources

CONTAINERS AT RUNTIME



CONTAINERS: THINGS TO MANAGE

- How often should I *update* my Dockerfile dependencies?
- How do I *build* my container images?
- How do I get my containers in *production*?
- How *many* servers do I need?
- How can I *scale* my app?

SERVERLESS: JUST PROVIDE YOUR CODE



Trigger
definition

Code
zipfile



Cloud platform



SERVERLESS

- Event-driven compute with near-instant scale
- Managed, ephemeral compute
- Never pay for idle

(Btw, there are actually servers)



AWS Lambda



Azure Functions



Google Cloud Functions

WHY SERVERLESS?

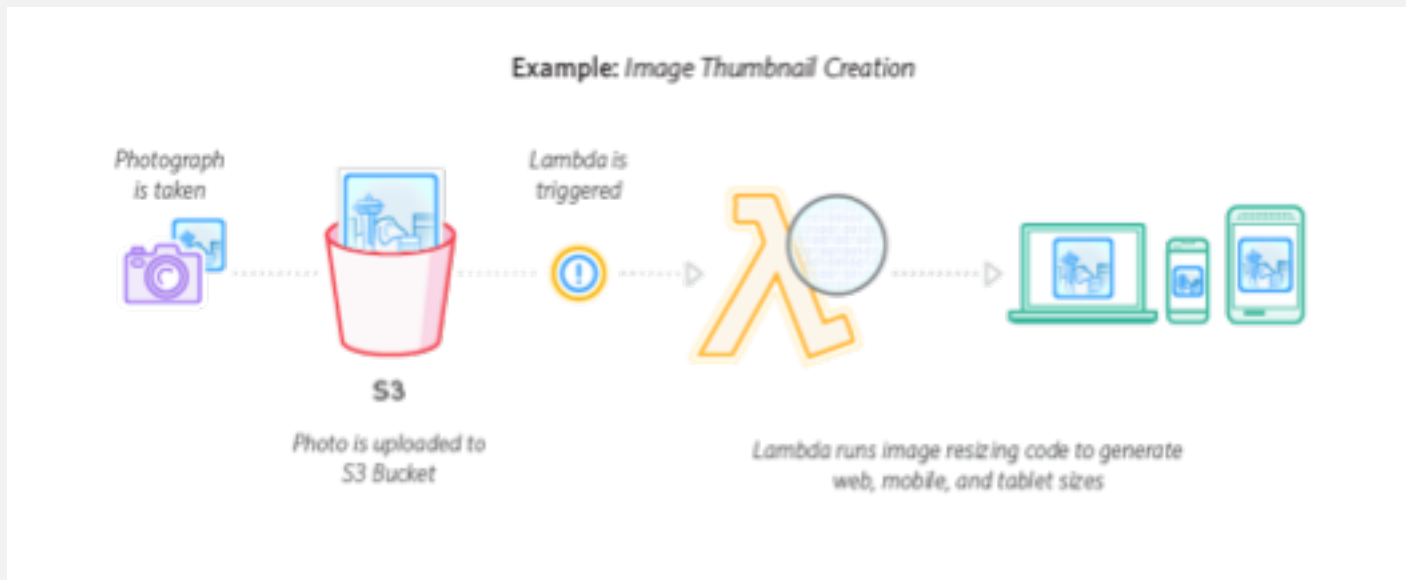
- Reduce operational overhead
- Faster time to market
- Focus on business value

The Serverless Spectrum <https://read.acloud.guru/the-serverless-spectrum-147b02cb2292>

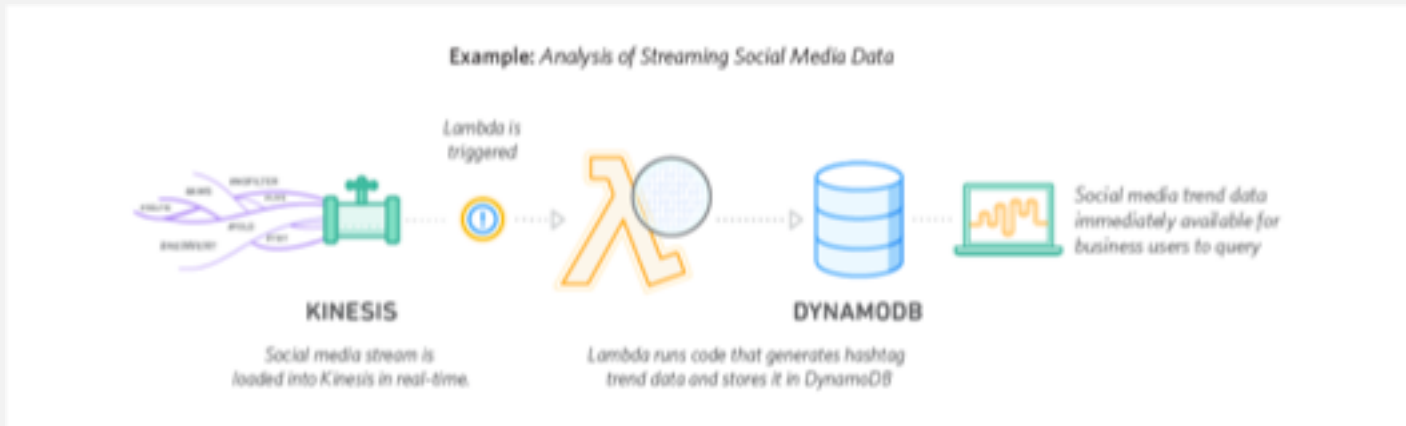
SCHEDULED TASKS



CREATE IMAGE THUMBNAIL



ANALYZE SOCIAL MEDIA STREAM



SERVERLESS CAVEATS

- Works best for event-based workloads
- Cloud vendor supports specific languages and runtimes
- Can't customize execution environment
- Not well-suited for long-running tasks

ANALOGY: RENTING VS OWNING A BIKE



NEW CONTAINER EXECUTION MODELS

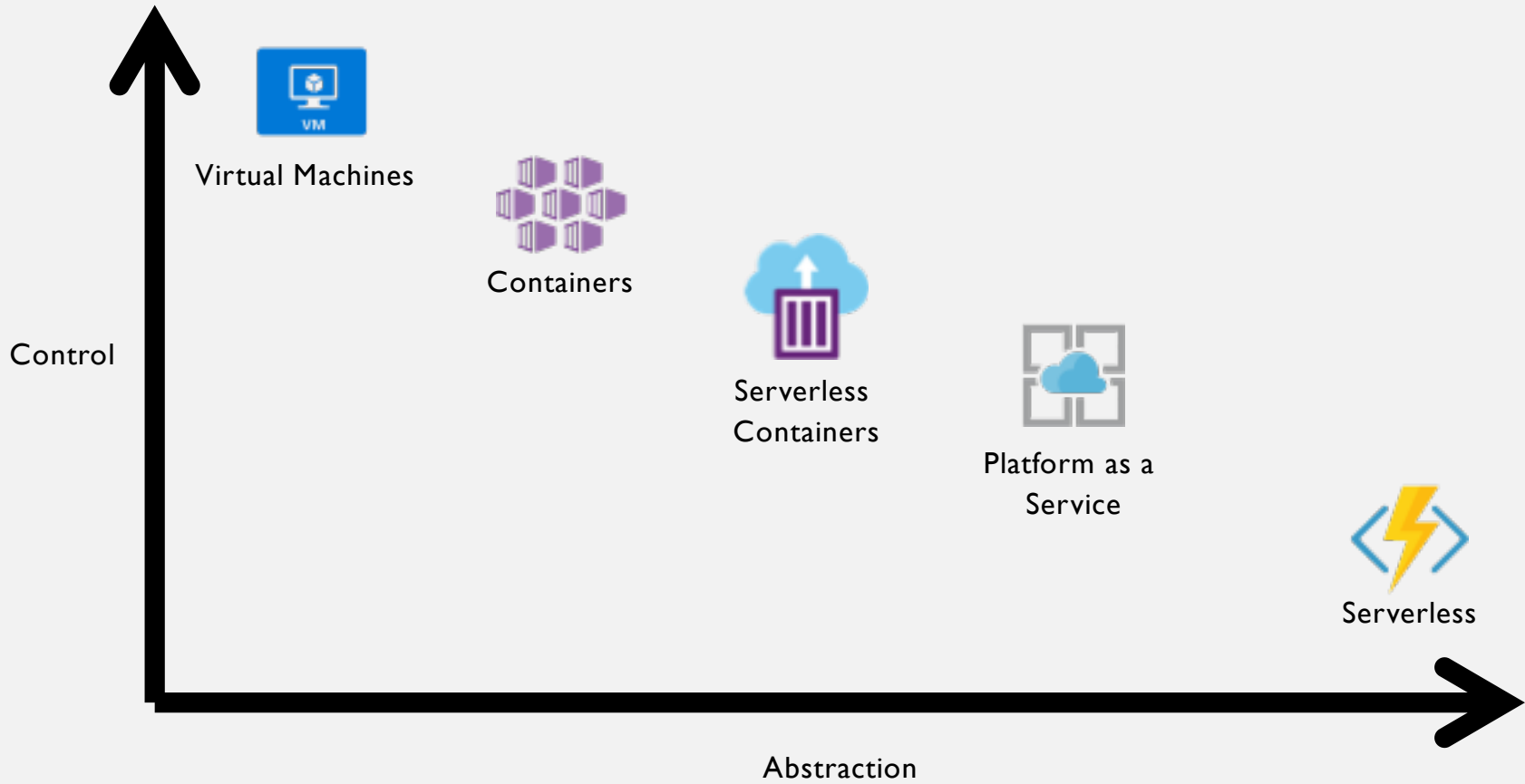
- Azure Container Instances
- AWS Fargate

- On-demand containers
- Don't have to manage underlying cluster

CONTAINERS AND SERVERLESS

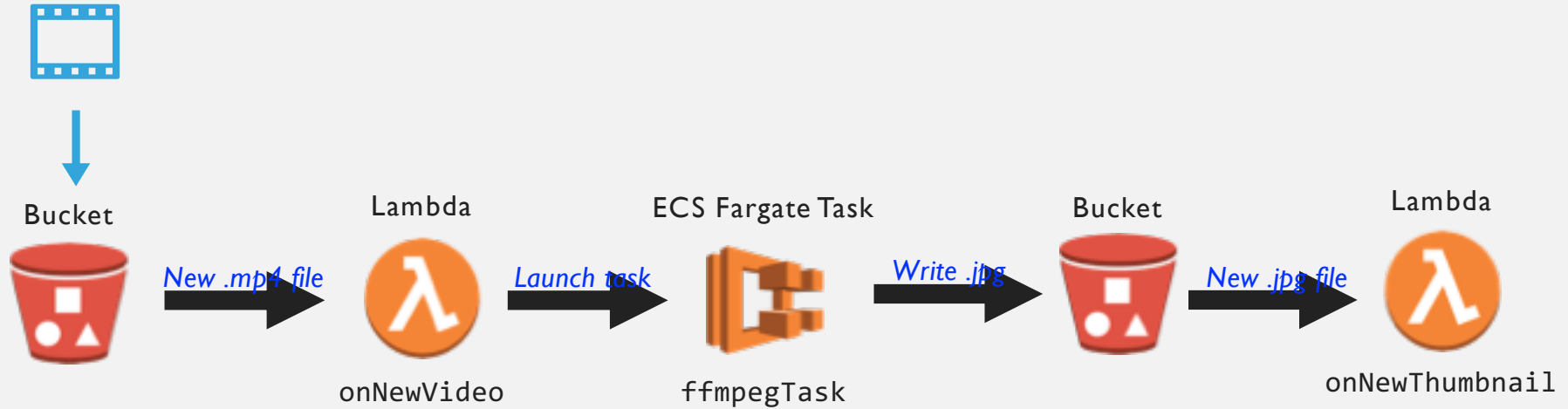
- Use containers for control over the execution environment
 - Customize software and physical servers
 - Great for long-running compute
- Use serverless for event-based compute that scales on demand
 - Less to manage
 - Less to configure

The cloud landscape

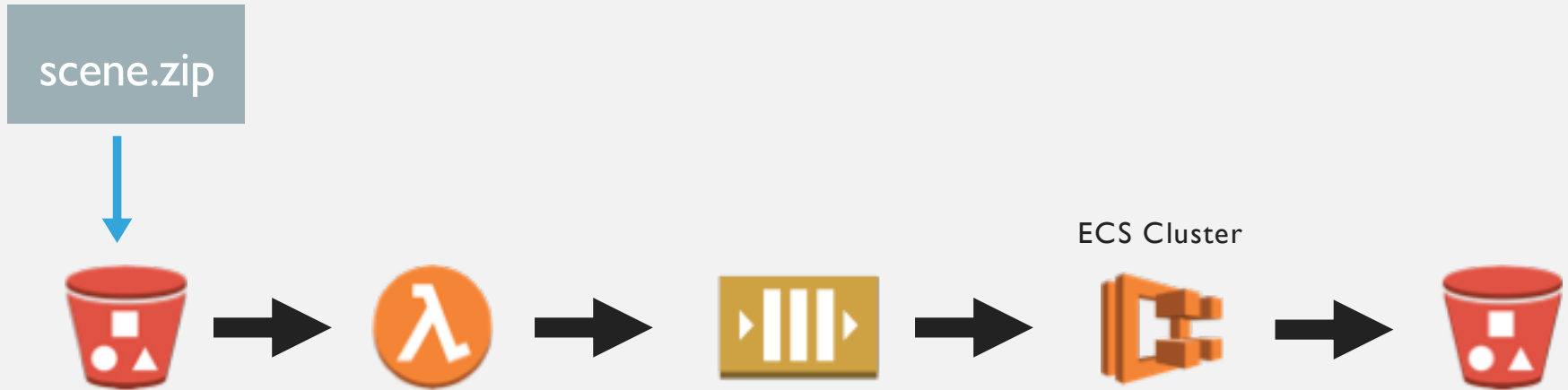


COMBINING THE TWO

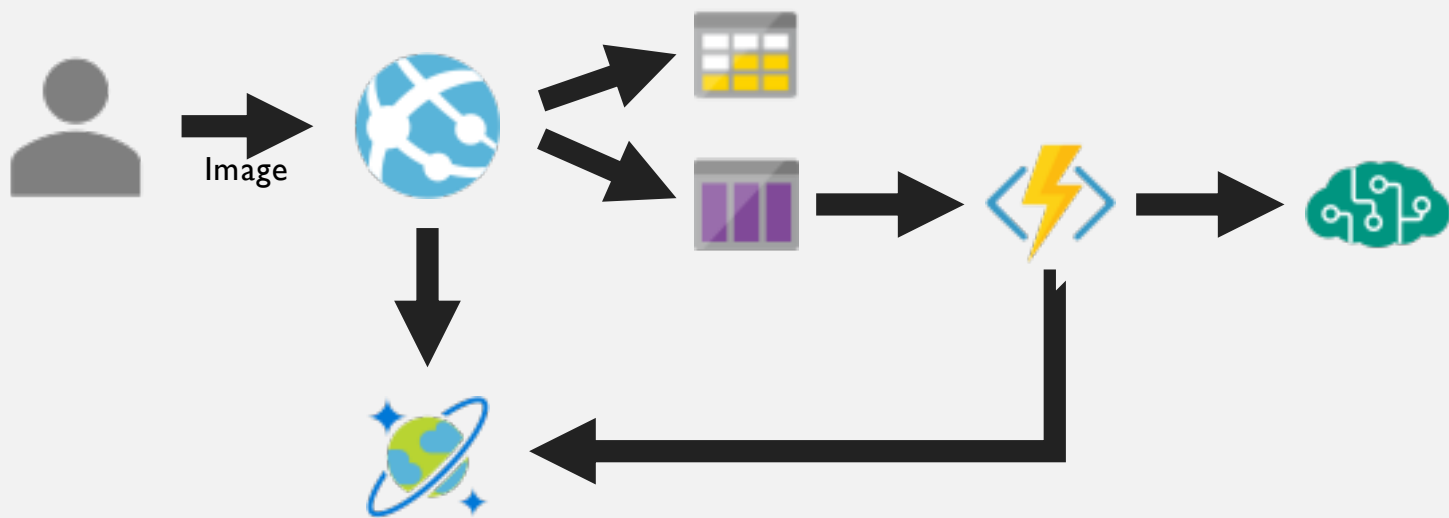
EXAMPLE: VIDEO THUMBNAILER



EXAMPLE: RAY TRACING



EXAMPLE: CONTENT MODERATION



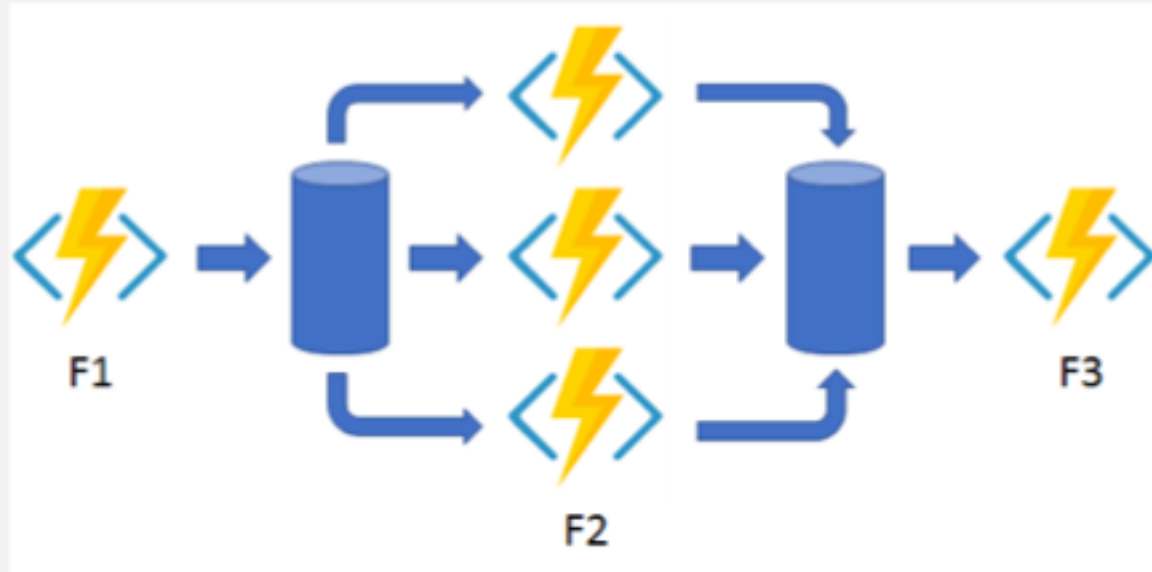
EXAMPLE: FUNCTION CHAINING



```
const df = require("durable-functions");

module.exports = df(function*(ctx) {
  const x = yield ctx.df.callActivityAsync("F1");
  const y = yield ctx.df.callActivityAsync("F2", x);
  const z = yield ctx.df.callActivityAsync("F3", y);
  return yield ctx.df.callActivityAsync("F4", z);
});
```

EXAMPLE: DURABLE FUNCTIONS



TOOLS

VENDOR DEPLOYMENT TOOLS

AWS CLOUDFORMATION

```

AWSTemplateFormatVersion: '2010-12-01'
Description: 'AWS CloudFormation Template'

Parameters:
- Name: MyParam
  Type: String
  Default: default

Resources:
  MyResource:
    Type: 'AWS::IAM::User'
    Properties:
      Path: '/'
      Name: MyResource
      PasswordPolicy:
        MinLength: 14
        RequireLowercase: true
        RequireUppercase: true
        RequireNumbers: true
        RequireSymbols: true
  
```

AZURE RESOURCE MANAGER

```

{
  "name": "Microsoft.Storage/storageAccounts",
  "type": "Microsoft.Storage/storageAccounts",
  "apiVersion": "2018-04-01",
  "location": "[resourceGroup().location]",
  "properties": {
    "accountType": "[variable('storageAccountType')]",
    "allowBlobPublicAccess": false
  }
},
{
  "name": "Microsoft.Network/virtualNetworks",
  "type": "Microsoft.Network/virtualNetworks",
  "apiVersion": "2018-04-01",
  "location": "[resourceGroup().location]",
  "properties": {
    "addressSpace": {
      "addressPrefixes": [
        "[variable('vnetAddressSpace')]"
      ]
    },
    "dhcpOptions": {
      "dhcpOptionsLabel": "[variable('dhcpOptionsLabel')]"
    }
  }
},
{
  "name": "Microsoft.Network/virtualNetworkGateways",
  "type": "Microsoft.Network/virtualNetworkGateways",
  "apiVersion": "2018-04-01",
  "location": "[resourceGroup().location]",
  "properties": {
    "sku": "Standard",
    "ipAddresses": [
        "[variable('vnetGatewayIP')]"
      ]
    }
  }
}
}

```

GOOGLE CLOUD DEPLOYMENT MANAGER

```

imports:
- path: path/to/My_vm_template.jinja
  name: My_rendered_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:

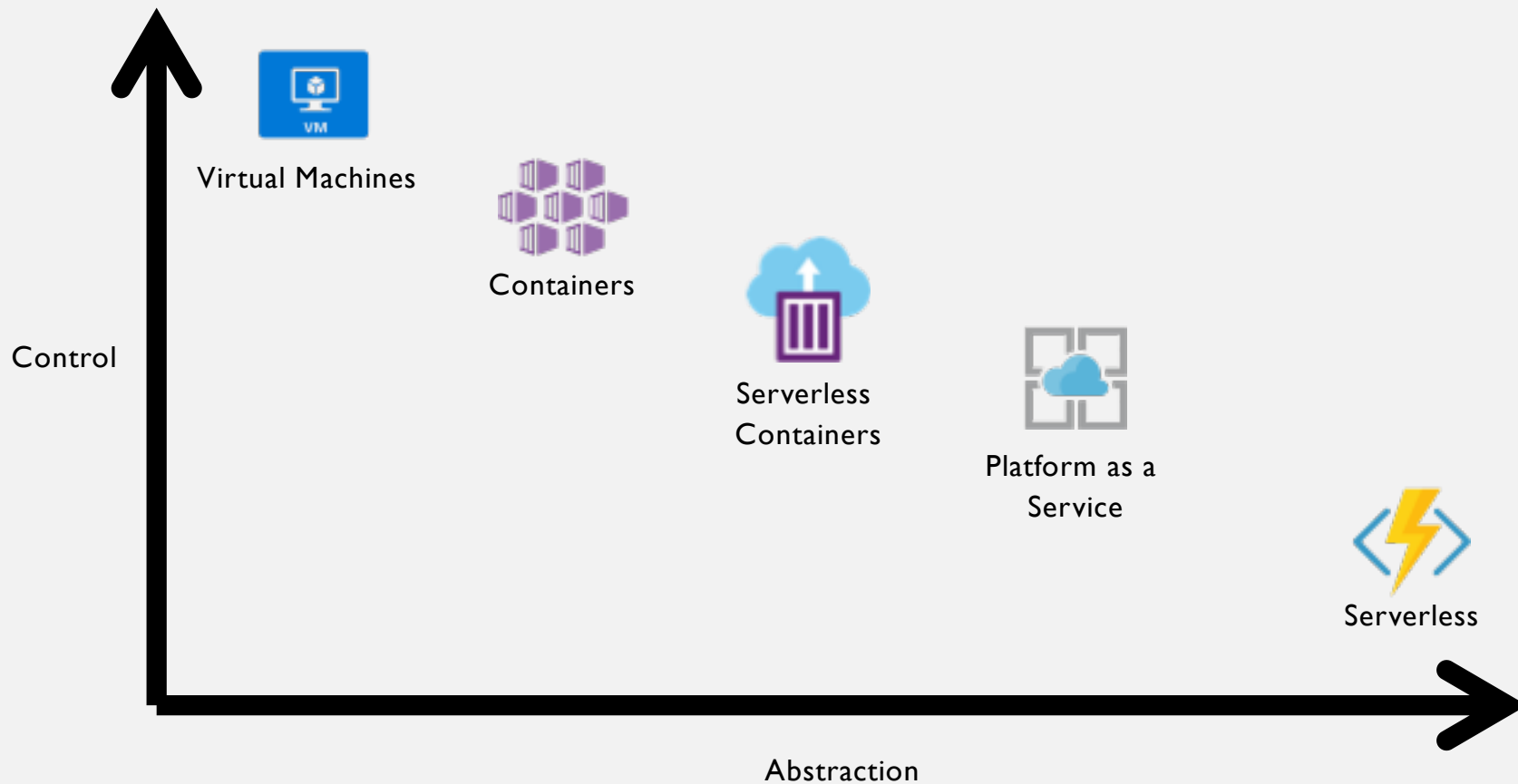
imports:
- path: resource_type.txt

resources:
- name: my-vm
  type: {{ imported('resource_type.txt') }} # Resolves to "compute.v1.instance"
  properties:
    zones: us-central1-a
    machineType: zones/us-central1-a/machineTypes/f1-micro
    disks:
    - deviceName: boot
      type: PERISTENT
      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
  
```

TOOLS ALSO PROVIDE ABSTRACTION

- Use Terraform modules
- Use Serverless Framework plugins or components
- Use Pulumi components
- Examples: `github.com/lindyonna/velocity-examples`

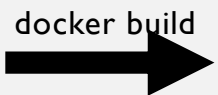
The cloud landscape



CONTAINERS



Dockerfile



Docker image



Container registry

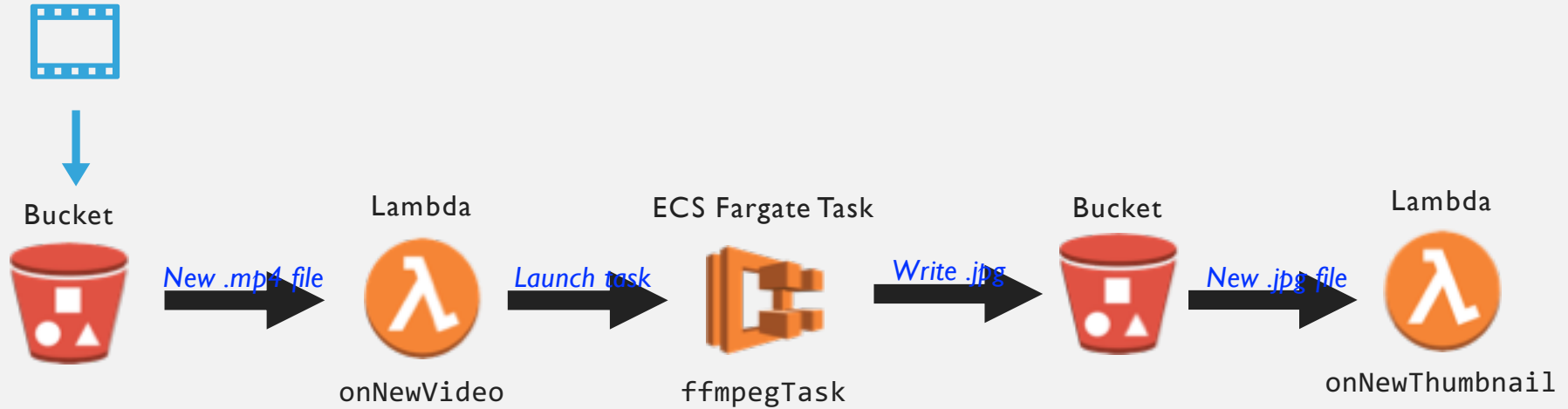


```
FROM jrottenberg/ffmpeg
RUN apt-get update && \
    apt-get install python-dev python-pip -y && \
    apt-get clean
RUN pip install awscli
WORKDIR /tmp/workdir
ENTRYPOINT
```



AWS Elastic Container Service

EXAMPLE: VIDEO THUMBNAILER



DEFINING THE APP IN PULUMI

Dockerfile

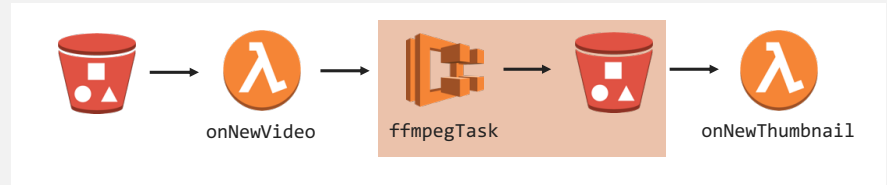
```
FROM jrottenberg/ffmpeg
```

```
RUN apt-get update && \  
    apt-get install python-dev python-pip -y && \  
    apt-get clean
```

```
RUN pip install awscli
```

```
WORKDIR /tmp/workdir
```

```
ENTRYPOINT \  
\  
aws s3 cp s3://${S3_BUCKET}/${INPUT_VIDEO} ./${INPUT_VIDEO} && \  
ffmpeg -i ./${INPUT_VIDEO} -ss ${TIME_OFFSET} -vframes 1 -f image2 -an -y ${OUTPUT_FILE} && \  
aws s3 cp ./${OUTPUT_FILE} s3://${S3_BUCKET}/${OUTPUT_FILE}
```



```
let bucket = new cloud.Bucket("bucket");

let ffmpegTask = new cloud.Task("ffmpegTask", {
  build: "./docker-folder",
  memoryReservation: 512,
});
```



```
bucket.onPut("onNewVideo", async (bucketArgs) => {
  const file = bucketArgs.key;
  const framePos = ... extract time offset from filename
});
```

```
await new cloud.ECSTask.run({
  imageRepository: "amazon/amazon-ecs-optimized-ubuntu",
  environment: [
    { "S3_BUCKET": bucket.id.get(),
      "INPUT_VIDEO": file,
      "TIME_OFFSET": framePos,
      "OUTPUT_FILE": file + '.jpg' },
  ],
});
```

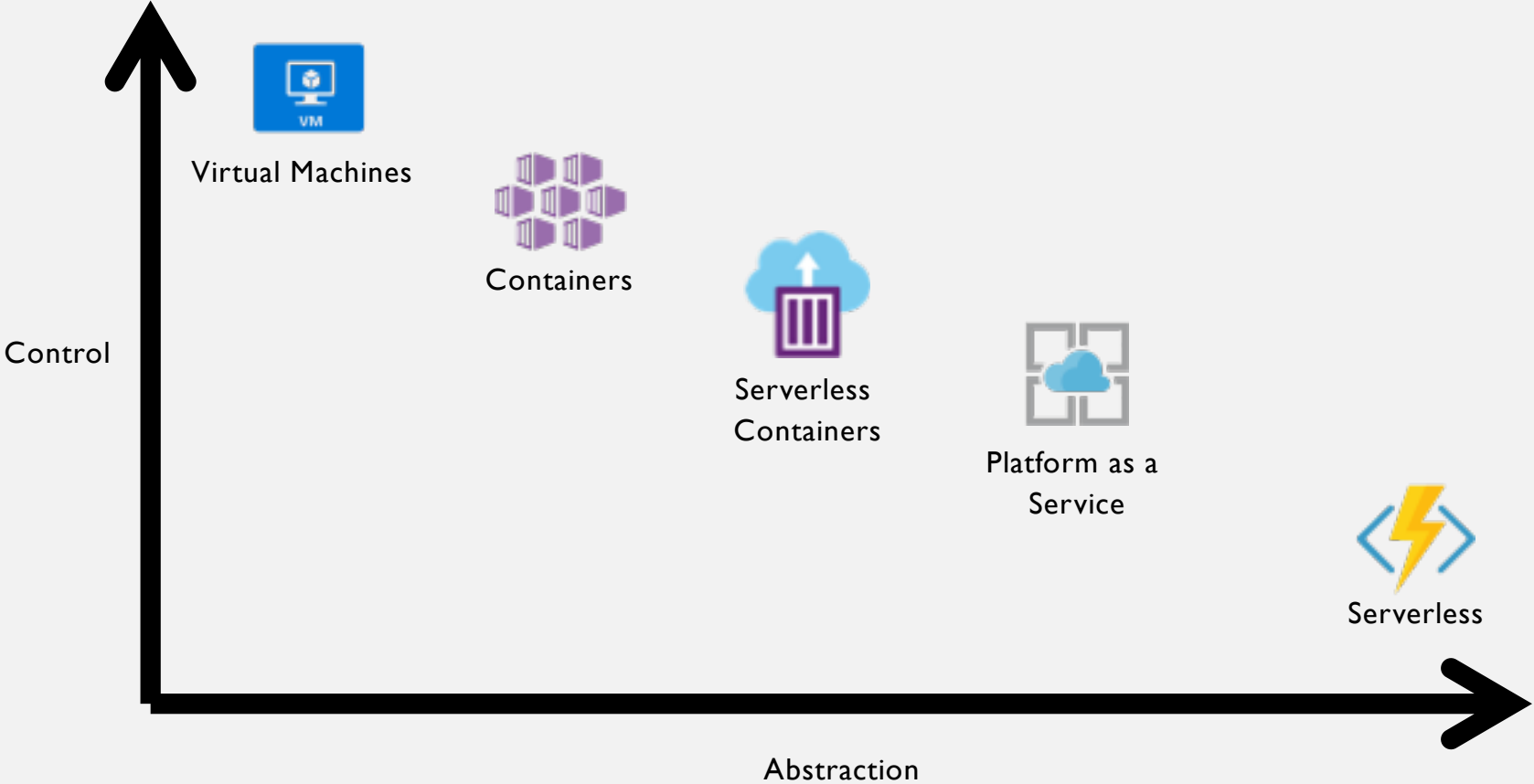
```
bucket.onPut("onNewThumbnail", async (bucketArgs) => {
  console.log(`*** New thumbnail: file ${bucketArgs.key}`);
}, { keySuffix: ".jpg" });
```

EXAMPLE: PROVISION QUEUES

```
function createQueue(name, deadLetter) {
  return new aws.sqs.Queue(`${common.prefix}-${name}`, { ... });
}

exports.certIssuer = {
  request:    createQueue("c-i-req", true),
  response:  createQueue("c-i-res", true),
  prepare:   createQueue("c-i-prep", true),
  initOrg:   createQueue("c-i-init-org", true),
  initOrgRes: createQueue("c-i-init-org-res", true),
  confirmTx: createQueue("confirm-tx"),
};
```

The cloud landscape



CONTAINERS WITH PULUMI

- How often should I *update* my Dockerfile dependencies?
- ~~How do I build my container images?~~
- ~~How do I get my containers in production?~~
- How *many* servers do I need?
- How can I *scale* my app?

SUMMARY

- Serverless and containers each have their place
- Use serverless for event-based code that needs to scale on demand
- Use containers for durable workloads, or to customize environment
- Define abstractions using infrastructure-as-code tooling

Learn more at pulumi.io

github.com/pulumi

@PulumiCorp

@lindyonna