Showdown of the serverless cloud, orchestrating functions

choose your alliance, go to: https://presentain.com/showdown
Kenny Baas-Schwegler
Software Consultant - Domain-Driven Design
@kenny_baas
Baasie.com
xebia.com/blog/author/kbaas/

Marc Duiker
Azure Consultant - I ❤ Serverless
@marcduiker
blog.marcduiker.nl
Why do I like serverless?
Why do I like serverless?
Let's look at the specs!
### Comparing Cost*

<table>
<thead>
<tr>
<th>Type</th>
<th>AWS Lambda</th>
<th>Azure Functions</th>
<th>Google Cloud Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price 1M Executions ($)</td>
<td>0.20</td>
<td>0.20</td>
<td>0.40</td>
</tr>
<tr>
<td>Price GB-s ($)</td>
<td>0.00001667</td>
<td>0.000016</td>
<td>0.0000025</td>
</tr>
<tr>
<td>Price GHz-s ($)</td>
<td>-</td>
<td>-</td>
<td>0.0000100</td>
</tr>
<tr>
<td>Free executions/month</td>
<td>1M</td>
<td>1M</td>
<td>2M</td>
</tr>
<tr>
<td>Free GB-s/month</td>
<td>400K</td>
<td>400K</td>
<td>400K</td>
</tr>
<tr>
<td>Free GHz-s/month</td>
<td>-</td>
<td>-</td>
<td>200K</td>
</tr>
<tr>
<td>Total cost/month ($)</td>
<td>78.48</td>
<td>75.40</td>
<td>92.70</td>
</tr>
</tbody>
</table>

* For 10M function executions of 1 sec each using a 512MB / 800MHz machine.
## Comparing Runtimes

<table>
<thead>
<tr>
<th>Runtimes</th>
<th>AWS Lambda</th>
<th>Azure Functions</th>
<th>Google Cloud Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node.js</td>
<td>v6 &amp; v8</td>
<td>v6, v8, v10</td>
<td>v6 &amp; v8 (beta)</td>
</tr>
<tr>
<td>.NET Framework</td>
<td>-</td>
<td>v4.7</td>
<td>-</td>
</tr>
<tr>
<td>.NET Core</td>
<td>v1, v2.x</td>
<td>v2.x</td>
<td>-</td>
</tr>
<tr>
<td>Python</td>
<td>v.2.7, v3.6 &amp; v3.7</td>
<td>v3.6 (preview)</td>
<td>v3.7.1 (beta)</td>
</tr>
<tr>
<td>Java</td>
<td>v8</td>
<td>v8 (preview)</td>
<td>-</td>
</tr>
<tr>
<td>Ruby</td>
<td>v2.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Go</td>
<td>v1.x</td>
<td>-</td>
<td>v1.11 (beta)</td>
</tr>
</tbody>
</table>
Comparing Cold Start

Source: https://mikhail.io/2018/08/serverless-cold-start-war/
You get a FAAS
Everybody gets a FAAS
Flying Spaghetti Functions

https://clearlens.org/do-you-believe-in-a-flying-spaghetti-monster/
Domain-Driven Design
Request Car Entry

License Plate Repository

- Find Matching License Plate Registration
- * License plate registration must be matched
- No License Plate registered

Call office support

- Appointment License Plate registered
- Employee License Plate registered

Confirm Appointment

Confirm Employee

- Occupy Parking Space

Parking Garage
- Parking spot needs to be free

Reserve Parking Space

Parking Garage
- Parking spot needs to be free

Available parking spots

Parking Garage Occupied

Notify contact Policy

Parking Space Occupied

Open Gate Policy

Gate openend

Message: Welcome

Parking Garage Occupied

Notice: Parking space found

Parking Space Available

Message: No parking space available

Parking Space Reserved
**Question**: How do we split systems into functions?

**Heuristic**: Within a bounded context modules are a perfect candidate for splitting up into functions.

**Heuristic**: Use one source control repository per bounded context.
“Everything we hear is an \textit{opinion}, \textit{not a fact}. Everything we see is a \textit{perspective}, \textit{not the truth}.”

– Marcus Aurelius, Meditations
What is AWS Lambda?

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time you consume - there is no charge when your code is not running. With AWS Lambda, you can run code for virtually any type of application or backend service - all with zero administration. AWS Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. All you need to do is supply your code in one of the languages that AWS Lambda supports (currently Node.js, Java, C#, Go and Python).

You can use AWS Lambda to run your code in response to events, such as changes to data in an Amazon S3 bucket or an Amazon DynamoDB table; to run your code in response to HTTP requests using Amazon API Gateway; or invoke your code using API calls made using AWS SDKs. With these capabilities, you can use Lambda to easily build data processing triggers for AWS services like Amazon S3 and Amazon DynamoDB, process streaming data stored in Kinesis, or create your own back end that operates at AWS scale, performance, and security.

You can also build serverless applications composed of functions that are triggered by events and automatically deploy them using AWS CodePipeline and AWS CodeBuild. For more information, see AWS Lambda Applications.

For more information about the AWS Lambda execution environment, see Lambda Execution Environment and Available Libraries. For information about how AWS Lambda determines compute resources required to execute your code, see Basic AWS Lambda Function Configuration.

When Should I Use AWS Lambda?

AWS Lambda is an ideal compute platform for many application scenarios, provided that you can write your application code in languages supported by AWS Lambda (that is, Node.js, Java, Go and C# and Python), and run within the AWS Lambda standard runtime environment and resources provided by Lambda.

When using AWS Lambda, you are responsible only for your code. AWS Lambda manages the compute fleet that offers a balance of memory, CPU, network, and other resources. This is in exchange for flexibility, which means you cannot log in to compute instances, or customize the operating system or language runtime. These constraints enable AWS Lambda to perform operational and administrative activities on your behalf, including provisioning capacity, monitoring fleet health, applying security patches, deploying your code, and monitoring and logging your Lambda functions.

If you need to manage your own compute resources, Amazon Web Services also offers other compute services to meet your needs.
A WALL OF TEXT CRITS FOR

OVER 9000!!!
Create your first function using Visual Studio

10/17/2018 • 5 minutes to read • Contributors all

Azure Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

In this article, you learn how to use the Visual Studio 2017 tools for Azure Functions to locally create and test a "hello world" function. You then publish the function code to Azure. These tools are available as part of the Azure development workload in Visual Studio 2017.

```
https://mydemofunctionapp20180106.azurewebsites.net/api/Function1?name=Bill
```

This topic includes a video that demonstrates the same basic steps.

Prerequisites
Google Cloud Functions documentation

Google Cloud Functions is a lightweight compute solution for developers to create single-purpose, stand-alone functions that respond to Cloud events without the need to manage a server or runtime environment.

- **Quickstart**
  - Create & deploy Cloud Functions

- **How-to guides**
  - Perform specific tasks

- **APIs & reference**
  - REST, RPC, and gcloud reference

- **Concepts**
  - Develop a deep understanding of Cloud Functions

- **Tutorials**
  - Walkthroughs of common applications

- **Support**
  - Get assistance with Google Cloud Functions issues

Was deze pagina nuttig? Laat ons weten hoe goed we u hebben geholpen:
## Documentation

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>• Extensive, detailed documentation.</td>
<td>• Extensive wall of text crits you</td>
</tr>
<tr>
<td></td>
<td>• Everything you need!</td>
<td></td>
</tr>
<tr>
<td>Azure Functions</td>
<td>• Quickstarts for novices</td>
<td>• Not 100% complete across all languages.</td>
</tr>
<tr>
<td></td>
<td>• Detailed reference material for the more experienced.</td>
<td></td>
</tr>
<tr>
<td>GCP Functions</td>
<td>• Quickstarts for novices.</td>
<td>• Hard to find specific information</td>
</tr>
</tbody>
</table>
export function handle(event: APIGatewayEvent, context: Context, callback: Callback) {

    let licensePlate: LicensePlate;
    try {
        licensePlate = LicensePlate.fromString(JSON.parse(event.body as string));
    } catch (error) {
        console.log(`error: "+error");
        return callback( error: null, Response.BAD_REQUEST(error));
    }

    return callback( error: null, Response.OK(licensePlate.number));
}

https://www.npmjs.com/package/@types/aws-lambda
export class Response {

    public static OK = (body: string) => new Response( statusCode: 200, body);
    public static BAD_REQUEST = (body: string) => new Response( statusCode: 400, body);
    public static CONFLICT = (body: string) => new Response( statusCode: 409, body);
    public static INTERNAL_SERVER_ERROR = (body: string) => new Response( statusCode: 500, body);

    private headers: { [name: string]: string } = {};

    constructor(public readonly statusCode: number, public readonly body: string) {
    }
}
```javascript
public static async saveParkingGarage(parkingGarage: ParkingGarage): Promise<ParkingGarage> {
    const tableName = await resolveTableName();
    const params: PutItemInput = {
        TableName: tableName,
        Item: parkingGarage.toJson() as PutItemInputAttributeMap,
    };
    return documentClient.put(params).promise().then(
        onfulfilled: data => Promise.resolve(parkingGarage),
        onrejected: error => Promise.reject(error)
    );
}
```
public static async findParkingGarage(id: string): Promise<ParkingGarage> {
    const tableName = await resolveTableName();

    const params = {
        TableName: tableName,
        KeyConditionExpression: '#id = :id',
        ExpressionAttributeNames: {
            '#id': 'id',
        },
        ExpressionAttributeValues: {
            ':id': id,
        }
    };

    let data = await documentClient.query(params).promise();
    if (data.Items) {
        return Promise.resolve(ParkingGarage.fromJson(data.Items.pop() as any));
    }
    return Promise.reject(reason: 'no parking garage found');
}
public static class Function1
{
    [FunctionName("Function1")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string name = req.Query["name"];

        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
        name = name ?? data?.name;

        return name != null
            ? (ActionResult)new OkObjectResult($"Hello, {name}"")
            : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
    }
}
public async Task<ParkingGarage> GetByIdAsync(string id)
{
    ParkingGarage result;
    var documentUri = UriFactory.CreateDocumentUri(DatabaseId, CollectionId, id);
    var requestOptions = new RequestOptions{ PartitionKey = new PartitionKey(PartitionKeyValue) };
    try
    {
        var document = await _documentClient.ReadDocumentAsync<ParkingGarage>(documentUri, requestOptions);
        result = document.Document;
    }
    catch (Exception e)
    {
        throw RepositoryExceptionBuilder.CreateExceptionForDocumentRead(documentUri, e);
    }
    return result;
}
```javascript
import {LicensePlate} from './entities/license-plate';
import {Request, Response} from 'express';

export function handle(request: Request, response: Response) {

  let licensePlate: LicensePlate;

  try {
    licensePlate = LicensePlate.fromJson(request.body);
  } catch (error) {
    console.log("error: " + error);
    return response.status(code: 400).send(error);
  }

  return response.send(licensePlate.number);
}
```
export class LicensePlateRepository {

  public static async findLicensePlate(license: string): Promise<LicensePlateRegistration> {
    const query = datastore.createQuery(kind: 'License').filter(property: "license", operator: "="+, license);

    const [licensePlateRegistrations] = await datastore.runQuery(query);
    if (licensePlateRegistrations.length > 0) {
      return Promise.resolve(LicensePlateRegistration.fromJson(licensePlateRegistrations.pop() as any));
    }
    return Promise.resolve(new LicensePlateRegistration(license, LicensePlateType.UNKNOWN));
  }
}

https://www.npmjs.com/package/@types/google-cloud__datastore
SDK - GCP

Request

The req object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on. In this documentation and by convention, the object is always referred to as req (and the HTTP response is res) but its actual name is determined by the parameters to the callback function in which you’re working.

For example:

```javascript
app.get('/user/:id', function(req, res) {
    res.send('user ' + req.params.id);
});
```

But you could just as well have:

```javascript
app.get('/user/:id', function(request, response) {
    response.send('user ' + request.params.id);
});
```

The req object is an enhanced version of Node’s own request object and supports all built-in fields and methods.

Properties

In Express 4, req.files is no longer available on the req object by default. To access uploaded files on the req.files object, use multipart-handling middleware like busboy, multer, formidable, multiparty, connect-multiparty, or pair.

- req.app

This property holds a reference to the instance of the Express application that is using the middleware.

If you follow the pattern in which you create a module that just exports a middleware function and require() it in your main file, then the middleware can access the Express instance via req.app

For example:

```javascript
//index.js
app.get('/viewdirectory', require('./mymiddleware.js'))
```
## SDK

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS Lambda</strong></td>
<td>● Flexible SDK, extensive.</td>
<td>● Quirks in the gateway body as string. Need to do validation yourself, also typed.</td>
</tr>
<tr>
<td></td>
<td>● Can also create your own runtime.</td>
<td></td>
</tr>
<tr>
<td><strong>Azure Functions</strong></td>
<td>● Easy to use SDK for C#.</td>
<td>● Limited usage of certain NuGet package versions the framework is using (e.g. Newtonsoft.Json).</td>
</tr>
<tr>
<td><strong>GCP Functions</strong></td>
<td>● Using standard 3rd party libraries.</td>
<td></td>
</tr>
</tbody>
</table>
Orchestration - AWS

https://github.com/binxio/aws-cfn-update
Orchestration - AWS

Execution ARN
arn:aws:states:eu-west-1:761563646002:execution:serverless-showdown-state-machine:7642ab7a-7b92-6464-638c-b1be2e273f5b

Input

End Time
Feb 5, 2019 06:03:05.219 AM

Output

Visual workflow

Success  Failed  Cancelled  In Progress

Code

Step details

Select a step to view its details.
Orchestration - AWS

Step details (ConfirmEmployee)

Status
Failed

Resource
arn:aws:lambda:eu-west-1:761563646002:function:confirmemployee-kb-local | CloudWatch logs

Input
{
    "number": "178",
    "type": "EmployeeLicensePlateMatched"
}

Output

Exception

Error
Lambda,Unknown

Cause
The cause could not be determined because Lambda did not return an error type.
public static class ParkingGarageCarEntryOrchestration
{
    [FunctionName(nameof(ParkingGarageCarEntryOrchestration))]
    public static async Task<ParkingOrchestrationResponse> Run(
        [OrchestrationTrigger] DurableOrchestrationContextBase context,
        ILogger logger)
    {
        if (!context.IsReplaying)
        {
            logger.LogInformation($"Started {nameof(ParkingGarageCarEntryOrchestration)} with InstanceId: {context.InstanceId}. ");
        }

        var request = context.GetInput<ParkingOrchestrationRequest>();

        var licensePlateResult = await context.CallActivityAsync<LicensePlateRegistration>(
            nameof(GetLicensePlateRegistration),
            request.LicensePlateNumber);

        var confirmParkingRequest = ConfirmParkingRequestBuilder.Build(request.ParkingGarageName, licensePlateResult);
        var confirmParkingResponse = await ConfirmParking(confirmParkingRequest, licensePlateResult, context);

        if (confirmParkingResponse.IsSuccess)
        {
            await context.CallActivityAsync(
                nameof(OpenGate),
                confirmParkingResponse.ParkingGarageName);
        }
        else
        {
        }
    }
}
public static class GetLicensePlateRegistration
{
    private static readonly ILicensePlateRegistrationService Service = new LicensePlateRegistrationService();

    [FunctionName(nameof(GetLicensePlateRegistration))]
    public static async Task<Domain.LicensePlateRegistration> Run(
        [ActivityTrigger] string licensePlateNumber,
        ILogger logger)
    {
        logger.LogInformation($"Started {nameof(GetLicensePlateRegistration)} with {licensePlateNumber}");

        var licensePlate = await Service.GetAsync(licensePlateNumber);
        return licensePlate;
    }
}
Orchestration - GCP

Not supported OOTB :(
<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS Lambda</strong></td>
<td>● Visualisation in the portal</td>
<td>● Uses JSON/YAML configuration.</td>
</tr>
<tr>
<td><strong>Azure Functions</strong></td>
<td>● Flexibility because the orchestration is in code.</td>
<td>● The orchestration needs to be deterministic, so don’t use code which is not (e.g. new GUIDs, DateTime, new threads).</td>
</tr>
<tr>
<td><strong>GCP Functions</strong></td>
<td>● N/A</td>
<td>● N/A</td>
</tr>
</tbody>
</table>
Deployment Cycle - AWS

```json
FindMatchingLicensePlate:
  Type: 'AWS::Serverless::Function'

Properties:
  FunctionName: !Sub "pg-findmatchinglicenseplate-$Stage"
  Handler: find-matching-license-plate.handle
  CodeUri: dist/
  Description: FaaS handler for requesting a car entry
  MemorySize: 128
  ReservedConcurrentExecutions: 20
  Environment:
    Variables:
      LICENSE_PLATE_TABLE_NAME: !Ref 'LicensePlateRepository'
  Policies:
    - Version: "2012-10-17"
    Statement:
      - Effect: "Allow"
        Action:
          - "dynamodb:Query"
        Resource:
          !Sub 'arn:aws:dynamodb:${AWS::Region}:${AWS::AccountId}:table/LicensePlateRepository*'
```
Deployment Cycle - AWS

https://xebia.com/blog/building-an-elixir-runtime-for-aws-lambda/
## Deployment Cycle - AWS

### Stacks (2)

<table>
<thead>
<tr>
<th>Stack name</th>
<th>Status</th>
<th>Created time</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverless-showdown-kb-local</td>
<td>UPDATE_COMPLETE</td>
<td>Mon, 04 Feb 2019 14:23:40 GMT</td>
</tr>
<tr>
<td>serverless-showdown-kb-local</td>
<td>UPDATE_COMPLETE</td>
<td>Mon, 04 Feb 2019 14:23:40 GMT</td>
</tr>
</tbody>
</table>

### Logs

1. **UPDATE_ROLLBACK_COMPLETE**
   - 16:46:51 UTC-0500
   - Status: AWS:CloudFormation:Stack
   - serverless-showdown-kb-local

2. **UPDATE_ROLLBACK_COMPLETE_CLEANUP_IN_PROGRESS**
   - 16:46:49 UTC-0500
   - Status: AWS:CloudFormation:Stack
   - serverless-showdown-kb-local

3. **UPDATE_COMPLETE**
   - 16:46:49 UTC-0500
   - Status: AWS:StepFunctions:StateMachine
   - ParkingAccessStateMachine

4. **UPDATE_COMPLETE**
   - 16:46:46 UTC-0500
   - Status: AWS:Lambda:Function
   - GarageGateway

5. **UPDATE_COMPLETE**
   - 16:46:46 UTC-0500
   - Status: AWS:Lambda:Function
   - ConfirmEmployee

6. **UPDATE_COMPLETE**
   - 16:46:46 UTC-0500
   - Status: AWS:Lambda:Function
   - FindMatchingLicensePlate

7. **UPDATE_COMPLETE**
   - 16:46:46 UTC-0500
   - Status: AWS:Lambda:Function
   - ConfirmAppointment

8. **UPDATE_COMPLETE**
   - 16:46:46 UTC-0500
   - Status: AWS:Lambda:Function
   - RequestCarEntry

9. **UPDATE_IN_PROGRESS**
   - 16:46:46 UTC-0500
   - Status: AWS:Lambda:Function
   - GarageGateway

10. **UPDATE_IN_PROGRESS**
    - 16:46:46 UTC-0500
    - Status: AWS:Lambda:Function
    - ConfirmAppointment

11. **UPDATE_IN_PROGRESS**
    - 16:46:46 UTC-0500
    - Status: AWS:Lambda:Function
    - ConfirmEmployee

12. **UPDATE_IN_PROGRESS**
    - 16:46:46 UTC-0500
    - Status: AWS:Lambda:Function
    - FindMatchingLicensePlate

13. **UPDATE_IN_PROGRESS**
    - 16:46:46 UTC-0500
    - Status: AWS:Lambda:Function
    - RequestCarEntry

14. **UPDATE_IN_PROGRESS**
    - 16:46:46 UTC-0500
    - Status: AWS:Lambda:Function
    - serverless-showdown-kb-local

15. **UPDATE_IN_PROGRESS**
    - 16:46:13 UTC-0500
    - Status: AWS:CloudFormation:Stack
    - serverless-showdown-kb-local

16. **UPDATE_FAILED**
    - 16:40:12 UTC-0500
    - Status: AWS:StepFunctions:StateMachine
    - ParkingAccessStateMachine

17. **UPDATE_IN_PROGRESS**
    - 16:40:10 UTC-0500
    - Status: AWS:StepFunctions:StateMachine
    - ParkingAccessStateMachine

The following resource(s) failed to update: ParkingAccessStateMachine.
Invalid State Machine Definition: "MISSING_TRANSITION_TARGET: Missing 'Next' target: Choice at /States/FindMatchingLicensePlate/Next, MISSING_TRANSITION_TARGET: State 'HandleParking' is not reachable at /States/HandleParking" (Service: AWSStepFunctions; Status Code: 400; Error Code: InvalidDefinition; Request ID: 49eb359e-2bcb-11ea-83bf-e683c9ef1b83)
Deployment Cycle - Azure
Deployment Cycle - Azure
Deployment Cycle - GCP

```hcl
-/- google_storage_bucket_object.archive (new resource required)
  id: "showdown-serverless-deploy-http_trigger.zip" => <computed> (forces new resource)
  bucket: "showdown-serverless-deploy" => "showdown-serverless-deploy"
  content_type: "application/zip" => <computed>
  crc32c: "0ZDR0B4Q=" => <computed>
  detect_md5hash: "hVC76p+XKpeD6zFpFETA.=" => "different hash" (forces new resource)
  md5hash: "hVC76p+XKpeD6zFpFETA.=" => <computed>
  name: "http_trigger.zip" => "http_trigger.zip"
  source: ".//dist/index.zip" => "./dist/index.zip"
  storage_class: "STANDARD" => <computed>

Plan: 1 to add, 0 to change, 1 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

google_storage_bucket_object.archive: Destroying... (ID: showdown-serverless-deploy-http_trigger.zip)
google_storage_bucket_object.archive: Destruction complete after 0s

google_storage_bucket_object.archive: Creating...
  bucket: "" => "showdown-serverless-deploy"
  content_type: "" => "computed="
  crc32c: "" => "computed="
  detect_md5hash: "" => "different hash"
  md5hash: "" => "computed="
  name: "" => "http_trigger.zip"
  source: "" => ".//dist/index.zip"
  storage_class: "" => "computed="

google_storage_bucket_object.archive: Creation complete after 0s (ID: showdown-serverless-deploy-http_trigger.zip)

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.

The state of your infrastructure has been saved to the path below. This state is required to modify and destroy your infrastructure, so keep it safe. To inspect the complete state use the 'terraform show' command.

State path: ./state/terraform.tfstate

baaste@baaste-Precision-5520:~$ gitrepos/prive/serverless-showdown/gcloud/infra$ terraform destroy
```
## Deployment Cycle

<table>
<thead>
<tr>
<th>AWS Lambda</th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Stateful Cloudformation.</td>
<td></td>
<td>● Sometimes manual interference when config is false.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Azure Functions</th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Multiple ways to deploy your functions to the cloud.</td>
<td></td>
<td>● Watch out for breaking changes in orchestrations.</td>
</tr>
<tr>
<td>● Prefer CLI over ARM.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GCP Functions</th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Terraform to the rescue!</td>
<td></td>
<td>● Google Cloud deployment configuration counter intuitive.</td>
</tr>
</tbody>
</table>
Portal - AWS

AWS Management Console

AWS services

Find services

Recently visited services

All services

Build a solution

Launch a virtual machine

Build a web app

Launch using virtual server

Create an IoT device

Start a development project

Register a domain

Deployment serverless microservice

Create a backend for your mobile app

Learn to build

Website and Web Apps

Storage

Database

DevOps

Machine Learning

Big Data

Access resources on the go

Explore AWS

AWS Marketplace

Amazon RDS

Solutions, Backup & Restore with Amazon S3

More on Resource Containers with AWS Fargate

Home feedback?
## Portal

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS Lambda</strong></td>
<td>● Most parts are simple and easy to discover.</td>
<td>● No real boundaries of projects.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Some parts are hard to navigate.</td>
</tr>
<tr>
<td><strong>Azure Functions</strong></td>
<td>● Resource Groups are useful containers for multiple services.</td>
<td>● It is very slow to navigate &amp; use.</td>
</tr>
<tr>
<td><strong>GCP Functions</strong></td>
<td>● Projects is good concept for grouping services.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Portal is quick to use.</td>
<td></td>
</tr>
</tbody>
</table>
Testing - AWS

Jest

```javascript
import {doesJournalEntryExist, localDate, putBMICalculatedEvent} from '../src/port.adapter.dynamodb/document-client';
import {BASIC_BMI_CALCULATED_EVENT} from '../../domain/bmi-eater-events.fixtures';
import {PersonInfo} from '../../src/domain/plastic-categories';

const mockResolvableTableName = jest.fn(implementation () => Promise.resolve({value: 'table'}));
jest.mock(moduleName: '../../src/support/properties', factory () => ({
  resolvePlasticTable(tableName: string) => mockResolvableTableName()
}));

let mockCount = 1;
const mockQueryResult = jest.fn(implementation () => Promise.resolve({
  Count: mockCount
}));
const mockPutResult = jest.fn(implementation () => Promise.resolve({
  }));
jest.mock(moduleName: 'aws-sdk', factory () => (
  DynamoDB: { 
    DocumentClient: class { 
      query() { 
        return { 
          promise: mockQueryResult
        };
      }
      put() { 
        return { 
          promise: mockPutResult
        };
      }
    }
  }
});
```
Testing - AWS

- Python Versions support
  - Python 2.7
  - Python 3.6
  - Python 3.7
- Supported AWS Lambda Runtimes
  - nodejs
  - nodejs4.3
  - nodejs6.10
  - nodejs8.10
  - java
  - python2.7
  - python3.6
  - python3.7
  - go1.x
    - dotnetcore1.0
    - dotnetcore2.0
    - dotnetcore2.1
  - ruby2.5
  - Provided
- AWS credential support
- Debugging support
- Inline Swagger support within SAM templates
- Validating SAM templates locally
- Generating boilerplate templates
  - nodejs
  - nodejs4.3
  - nodejs6.10
  - nodejs8.10
  - java
  - python2.7
  - python3.6
  - python3.7
  - go1.x
    - dotnetcore1.0
    - dotnetcore2.0
    - dotnetcore2.1
  - ruby2.5
  - Provided
private DurableOrchestrationContextBase CreateFakeContextForUnknownLicensePlate()
{
    var context = A.Fake<DurableOrchestrationContextBase>();
    // Configure input
    A.CallTo(() => context.GetInput<ParkingOrchestrationRequest>())
        .Returns(new ParkingOrchestrationRequest
        {
            ParkingGarageName = "Parking Garage 1",
            LicensePlateNumber = "ABC-123"
        });

    // Configure GetLicensePlateRegistration activity
    A.CallTo(() => context.CallActivityAsync<LicensePlateRegistration>(
        nameof(GetLicensePlateRegistration),
        A<string>.Returns("")))
        .Returns(Task.FromResult(new LicensePlateRegistration {Type = LicensePlateType.Unknown}));

    // Configure DisplayMessage activity
    A.CallTo(() => context.CallActivityAsync(
        nameof(DisplayMessage),
        A<DisplayMessageRequest>.Returns("")))
        .Returns(Task.CompletedTask);

    return context;
}
public async Task GivenLicensePlateIsUnknown_WhenOrchestrationIsStarted_ThenGateOpenedShouldBeFalse()
{
    // Arrange
    var context = CreateFakeContextForUnknownLicensePlate();
    var logger = A.Fake<ILogger>();

    // Act
    var result = await ParkingGarageCarEntryOrchestration.Run(context, logger);

    // Assert
    result.GateOpened.ShouldBeFalse();
}
Testing - Azure

```
[2/4/2019 15:43:55] Found the following Functions:
[2/4/2019 15:43:57] Content root path: C:\dev\git\personal\serverless-showdown\azure\src\ServerlessParking.Application\bin\Debug\netcoreapp2.0
[2/4/2019 15:43:57] Now listening on: http://0.0.0.0:7071
```

**Http Functions:**

```
OrchestrationStarter: [POST] http://localhost:7071/api/orchestration/{functionName}
```

```
[2/4/2019 15:44:02] Host lock lease acquired by instance ID '000000000000000000000081C73C16'.
```
Testing - Azure

Orchestration starter

POST http://localhost:7071/api/orchestration/ParkingGARageCarEntryOrchestration

Params | Authorization | Headers | Body | Pre-request Script | Tests | Cookies | Code | Comments

- none
- form-data
- x-www-form-urlencoded
- raw
- binary

Body

```
{
  "licensePlateNumber": "123-ABC",
  "parkingGarageName": "Xebia Amsterdam"
}
```
public static class ParkingGarageCarEntryOrchestration
{
    [FunctionName(nameof(ParkingGarageCarEntryOrchestration))]
    public static async Task<ParkingOrchestrationResponse> Run(
        [OrchestrationTrigger] DurableOrchestrationContextBase context,  
        ILogger logger)  
        logger = [Logger]
    {
        if (!context.IsReplaying)
        {
            logger.LogInformation($"Started {nameof(ParkingGarageCarEntryOrchestration)} with InstanceId: {context.InstanceId}");
        }

        var request = context.GetInput<ParkingOrchestrationRequest>();
        var requestLicensePlateNumber = request.ParkingGarageName + "123-ABC"
        var requestLicensePlateRegistration = request.ParkingGarageName + "Xebia Amsterdam"
        var confirmParkingRequest = ConfirmParkingRequestBuilder.Build(request.ParkingGarageName, licensePlateResult);
        var confirmParkingResponse = await ConfirmParking(confirmParkingRequest, licensePlateResult, context);
        if (confirmParkingResponse.IsSuccess)
Testing - GCP

```
baasie@baasie-Precision-5520:~/.gitrepos/prive/serverless-showdown/gcloud$ functions start
Starting Google Cloud Functions Emulator...
Google Cloud Functions Emulator STARTED

<table>
<thead>
<tr>
<th>Status</th>
<th>Name</th>
<th>Trigger</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>READY</td>
<td>handle</td>
<td>HTTP</td>
<td><a href="http://localhost:8010/speeltuin-kenny-baas/us-central1/handle">http://localhost:8010/speeltuin-kenny-baas/us-central1/handle</a></td>
</tr>
</tbody>
</table>

baasie@baasie-Precision-5520:~/.gitrepos/prive/serverless-showdown/gcloud$ functions call handle --data='{"number":"test"}'
ExecutionId: 63ab6a3-4e7c-4eb5-97c2-84ebf8e648e
Result: test
baasie@baasie-Precision-5520:~/.gitrepos/prive/serverless-showdown/gcloud$ ```
## Testing

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>−</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS Lambda</strong></td>
<td>● Unit testing is easy</td>
<td>● SAM Local still in beta</td>
</tr>
<tr>
<td></td>
<td>● SAM Local seems promising</td>
<td>● Mocking dynamo takes some time to understand</td>
</tr>
<tr>
<td><strong>Azure Functions</strong></td>
<td>● Unit testing is easy</td>
<td>● Local runtime depends on a separate storage emulator (not all versions are compatible).</td>
</tr>
<tr>
<td></td>
<td>● Local runtime is available for running without cloud connection.</td>
<td></td>
</tr>
<tr>
<td><strong>GCP Functions</strong></td>
<td>● Node Emulator available for local testing.</td>
<td>● Emulator is in alpha version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Supports Node v6 only</td>
</tr>
</tbody>
</table>
Monitoring - AWS
Monitoring - Azure
## Monitoring

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>• CloudWatch is mature, gives the right amount of stats</td>
<td></td>
</tr>
<tr>
<td>Azure Functions</td>
<td>• Basic logging OOTB and easy integration with Application Insights.</td>
<td></td>
</tr>
<tr>
<td>GCP Functions</td>
<td>• Basic monitoring OOTB</td>
<td></td>
</tr>
</tbody>
</table>
Question: When do we use AWS?

Heuristic: When we want to use many different function runtimes.

Heuristic: When we want to use the latest runtimes.
Question: When do we use Azure?

Heuristic: When we’re familiar with the Microsoft ecosystem.

Heuristic: When we want to orchestrate functions.
Key takeaways

**Question:** When do we use Google Cloud?

**Heuristic:** When we want the best functions portal experience.
**Key takeaways**

**Question:** When should we use **Orchestrations**?

**Heuristic:** Use orchestrations within a bounded context.

**Heuristic:** Use orchestrations when having processes over multiple modules in your bounded context.

**Heuristic:** Use orchestrations for long running processes.
Sources

Pricing
https://aws.amazon.com/lambda/pricing/
https://cloud.google.com/functions/pricing

Comparison:
https://docs.google.com/spreadsheets/d/1Q6vIlVYe1CfHAK6MP6Uz6mTYPiU4DljwL8XN4E5FQdw/edit?usp=sharing

Runtimes & languages
https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html
https://docs.microsoft.com/en-us/azure/azure-functions/support/languages
https://cloud.google.com/functions/docs/writing/

Cold Start
https://mikhail.io/2018/08/serverless-cold-start-war/

Github Repository with our demo code
https://github.com/Baasie/serverless-showdown