

Functions-as-a-Service & Workflows

Building a server-less application using FaaS Workflows

SPEAKER: DIEGO MARTIN

INDEX OF CONTENTS

- Stage 0
 - Functions-as-a-Services, a brief introduction
 - FaaS Workflows
 - Working with the technology: providers
- Stage 1
 - Building a ToDo-App from scratch



FUNCTIONS-AS-A-SERVICE

A BRIEF INTRO

Functions-as-a-Service, a brief introduction

Function-as-a-Service (aka FaaS) is a specialized kind of Platform-as-a-Service (aka PaaS) in which everything, besides the functional code itself, is abstracted and handled by the provider



It can be modeled like the image above in which both sides (the input and the output of the function) can be anything the provider allows you: databases, HTTP triggers, events, files, devices, etc



FaaS WORKFLOWS

FaaS Workflows

FaaS Workflows arise from the need to connect two (or more) functions in a logic and function-agnostic way



This connections can be as simple as a sequence or chain, or evolve into a more complex relation of functions that can result in a complete state machine (which could be considered as States-as-a-Service)



WORKING WITH THE TECHNOLOGY

PROVIDERS



Working with the technology

Until today, there's a few providers offering this technology:

- AWS Step Functions
- IBM Composer
- Platform9 Fission Workflows
- Microsoft Azure Logic Apps
- Oracle Fn Project (Fn Flow)

Working with the technology

AWS Setp Functions

```
{
  "Comment": "A simple minimal example of the States language",
  "StartAt": "Hello World",
  "States": {
    "Hello World": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloWorld",
      "End": true
    }
  }
}
```

Fission Workflows

```
apiVersion: 1
output: WhaleWithFortune
tasks:
  GenerateFortune:
    run: fortune
    inputs: "${$.Invocation.Inputs.default}"

  WhaleWithFortune:
    run: whalesay
    inputs: "${$.Tasks.GenerateFortune.Output}"
    requires:
      - GenerateFortune
```

IBM Composer

```
composer.try(
  composer.sequence(
    'myWatsonTranslator/languageId',
    composer.if(
      p => p.language !== 'en',
      composer.sequence(
        p => ({translateFrom: p.language, translateTo: 'en', payload: p.payload}),
        'myWatsonTranslator/translator'
      ),
      composer.sequence(
        p => ({text: p.payload}),
        'en2shakespeare'
      )
    )
  ),
  err => ({payload: 'Sorry we cannot translate your text!'})
)
```

Working with the technology

AWS Step Functions

The screenshot displays the AWS Step Functions console. On the left, a workflow graph is shown with a legend: Success (green), Failed (red), Needs retry (yellow), and In progress (blue). The graph starts at a yellow 'Start' node, followed by a green 'FetchAnOrder' node, then a green 'RegionChoice' node. From 'RegionChoice', the flow branches into three paths: 'CreateOrderA' (dashed box), 'CreateOrderB' (green box), and 'UnreservedRegion' (dashed box). 'CreateOrderA' leads to a green 'OrderOK' node, which then leads to a green 'ProcessOrder' node. 'CreateOrderB' leads to a dashed 'DatabaseError' node, which leads to a dashed 'NoOrderPossible' node. 'UnreservedRegion' leads to a dashed 'NoOrderPossible' node. All paths converge at a yellow 'End' node.

On the right, the 'Execution Details' panel is shown. It has tabs for 'Info', 'Input', and 'Output'. The 'Info' tab is active, showing the following details:

- Execution Status: **Succeeded**
- Started: Nov 20, 2016 9:58:28 AM
- Closed: Nov 20, 2016 9:58:32 AM

Below this is the 'Step Details' section, which contains a table with the following data:

ID	Type	Timestamp
▶ 1	ExecutionStarted	Nov 20, 2016 9:58:28 AM
▶ 2	TaskStateEntered	Nov 20, 2016 9:58:28 AM
▶ 3	LambdaFunctionScheduled	Nov 20, 2016 9:58:28 AM

IBM Composer

The screenshot shows the IBM Composer interface for an application named 'myApp' (version v0.0.1). It indicates that the user is in edit mode, viewing the currently deployed version. Below this, there is a code editor with the following code:

```
1 // try typing "composer." to begin your composition
2 composer.sequence(args => ({msg: `hello ${args.name}!`}));
```

Below the code editor is a workflow diagram. It starts with an 'Entry' node, followed by a blue action node containing the code: `args => ({msg: `hello ${args.name}!`})`. This is followed by an 'Exit' node.

At the bottom of the interface, there are buttons for 'Deploy' and 'Revert', along with some utility icons.

AZURE Logic Apps

The screenshot shows the Azure Logic Apps designer interface. The workflow starts with a trigger: 'When a feed item is published'. This is followed by a 'Condition' step. The condition is configured with the following logic:

- And (dropdown)
- Feed su... (selected) contains Microsoft
- Buttons: '+ Add', 'Add dynamic content'

Below the condition, there are two branches:

- If true** (green background): Contains an action 'Office 365 Outlook - Send an email'.
- If false** (red background): Empty.

At the bottom of each branch, there are buttons for 'Add an action' and 'More'.



The Hands-on!

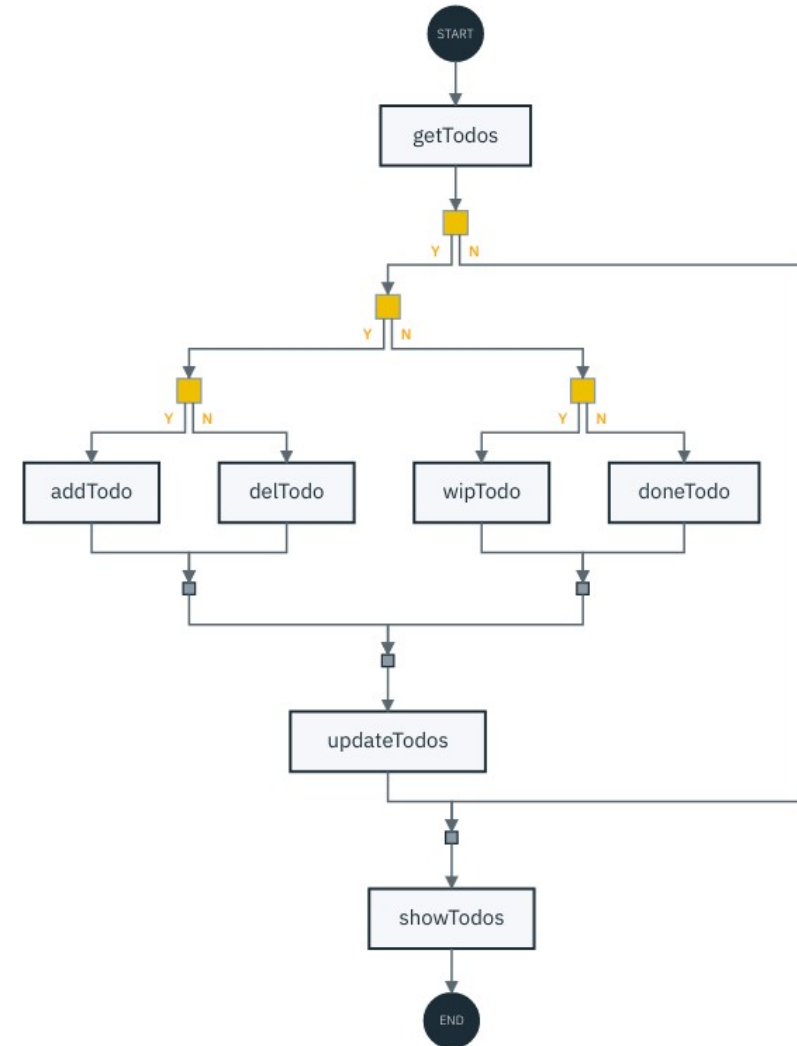
The composition

7 Simple functions

2 kind of connectors:

- sequence
- if-then-else

```
composer.sequence('FWTt/getTodos',  
  composer.if(params => (!!params.action) && (params.action.constructor === Object),  
    composer.sequence(  
      composer.if(params => params.action.add || params.action.del,  
        composer.if(params => params.action.add,  
          'FWTt/addTodo',  
          'FWTt/delTodo'  
        ),  
        composer.if(params => params.action.wip,  
          'FWTt/wipTodo',  
          'FWTt/doneTodo'  
        )  
      ),  
      'FWTt/updateTodos'  
    ),  
    composer.empty()  
  ),  
  'FWTt/showTodos'  
);
```



The image features a white background with decorative teal lines. On the left side, there are three parallel lines forming a corner shape. On the bottom right side, there are three parallel lines forming a diagonal shape.

The Hands-on!

The image features a white background with decorative teal lines in the corners. In the top-left corner, there are three parallel lines forming an L-shape. In the bottom-left corner, there are three parallel lines forming an L-shape. In the bottom-right corner, there are three parallel lines forming a diagonal shape. The word "THANKS!" is centered in the middle of the page.

THANKS!