FaaS Shell
Multi-Cloud Portable Serverless Function Workflow

Naohiro Tamura
Professional Engineer
Fujitsu Limited
About me

Recent work
- Apr. 2017 – Open Source PaaS, FaaS, Cloud Native Development (FaaS Shell)
- Sep. 2014 Proprietary System and Resource Management Software Development

Expertise
- System and Resource Management Software Development

Personal Interest
- Functional and Logic Programming

FaaS Shell is written in Prolog for agility, and also for joyful Programming 😊
What is Function Workflow?

- **Serverless Function**
  - Small for doing single task
  - Stateless and Immutable
  - Event Driven and Reactive
  - Auto scale
  - Pay per use

- **Functional Programming Language (FPL) Function**
  - Small for doing single task
  - Stateless and Immutable
  - Good at Event Driven and Reactive
  - Good at Concurrent and Parallel

- **FPL main theme is how to compose function**
  - sequential, conditional, repetition, parallel, exceptional handling, retry

- **Serverless Function requires Function Composition as well as FPL**
  ⇒ That is Serverless Function Workflow
Serverless Function Workflow

- Consists of 3 elements, for example AWS case
  - Serverless Function : AWS Lambda
  - Serverless Function Workflow : Step Functions
  - Workflow Language : Amazon States Language

- Workflow = Composition
  - Sequential
  - Conditional
  - Repetition
  - Parallel
  - Exceptional handling
  - Retry
Amazon States Language (ASL)

- State Machine
  - Pass
  - Task
  - Choice
  - Wait
  - Succeed
  - Fail
  - Parallel

```json
{
"Comment": "A Hello World example of the Amazon States Language using a Task state",
"StartAt": "HelloWorld",
"States": {
  "HelloWorld": {
    "Type": "Task",
    "End": true
  }
}
```
What is Function Workflow?

What is FaaS Shell and Why?

Demo

How does FaaS Shell work?

Summary

Q&A
What is FaaS Shell?

- It’s a shell which enables to exploit Serverless Functions across Multiple Clouds
- Covers major 4 FaaS Providers as well as reusable Functions
- Works as abstraction layer with keeping agility for future changes

**Diagram:**

- **Amazon State Language (ASL)**
- **Other Workflow Language**
- **FaaS Shell**
- **Abstraction Layer**

**Cloud Providers:**

- AWS Step Functions
- AWS Lambda
- Azure Logic Apps
- Google Cloud Composer
- IBM Functions Composer
- IBM Cloud Functions
- Apache OpenWhisk
- Fission Workflow
- Fission
- IFTTT
- Twilio
- StackStorm
Why FaaS Shell?

Serverless Function Workflow Landscape - Serveless Silo

- Each provider or platform has its own workflow service
- There is no interoperability among providers
- This situation will continue along with competing each other
Benefit of using FaaS Shell

FaaS Shell helps us shift Single Strategy to Multi Cloud Strategy

1\textsuperscript{st} Stage: Single Cloud Strategy
- Commit to one cloud vendor after thorough investigation
- Staying vendor lock-in state is nothing wrong if you and your customer are happy

2\textsuperscript{nd} Stage: Multi Cloud Strategy
- Serverless Function doesn’t cost at all unless called
- Each Cloud has strengths and weaknesses, own characteristics, especially in AI area
- Reusable FaaS such as IFTTT (if-this-then-that), Twilio, StackStorm, Node-RED, etc are available

Why not exploit the best part of each cloud and integrate them?
- In fact, we actually have the fortune of selecting the most attractive features from each provider, to enable a multi-cloud strategy.
What is Function Workflow?

What is FaaS Shell and Why?

Demo

How does FaaS Shell work?

Summary

Q&A
### AI Services Evaluation Demo Background

Which provider’s service is suitable for a specific app? **We need evaluation!**

<table>
<thead>
<tr>
<th>Natural Language Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>aws</strong> Amazon Comprehend, Amazon Translate, Amazon Transcribe, Amazon Lex, Natural Language API, Translation API, Watson Language Translator, Watson Language Classifier</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Speech Recognition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>aws</strong> Amazon Polly, Amazon Speech, Amazon Speech API, Watson Speech to Text, Watson Text to Speech</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image Recognition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>aws</strong> Amazon Rekognition, Computer Vision, Vision API, IBM Visual Recognition</td>
</tr>
</tbody>
</table>

Dataset and Algorism are different each other.
Evaluate Translation using an Ambiguous Sentence
- very basic Natural Language Processing example
- it may be that the professor is lecturing with the cat, or that the student has the cat.

See how each service translates it into other language and back to English

[1] “Structure and Interpretation of Computer Programs”
Translation Evaluation Demo Workflow Graph

- Total 26 States and 4 Parallel Transitions
Translation Evaluation Demo Deployment

Text

Translation Eval. Workflow

Faas Shell

Authentication and Authorization code are required

Input

Output

Serverless Function

Translation Service

AWS Lambda

Amazon Translate

Azure Functions

Azure Translator Text

Google Cloud Functions

Google Translation API

Watson Language Translator

IBM Cloud Functions

RBAC setting only, No auth coding

FaaS Shell
Demo Steps through REST Interface

- Register Statemachine

- Execute Statemachine
  - English to Arabic (ar), and back to English
  - English to Chinese (zh), and back to English
  - English to Japanese (ja), and back to English

- Generate Statemachine Graph (Graphviz DOT)
TOC

- What is Function Workflow?
- What is FaaS Shell and Why?
- Demo
- How does FaaS Shell work?
- Summary
- Q&A
### REST API

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/</code></td>
<td>Get version number</td>
</tr>
<tr>
<td><code>/executions/{execution_id}</code></td>
<td>Get background statemachine information</td>
</tr>
<tr>
<td><code>/activity/{activity_task}</code></td>
<td>Get/Put/Heartbeat activity</td>
</tr>
<tr>
<td><code>/trigger/{event_state}</code></td>
<td>Send Event</td>
</tr>
<tr>
<td><code>/faas/{function}</code></td>
<td>List callable function information in each FaaS</td>
</tr>
<tr>
<td><code>/statemachine/{statemachine}</code></td>
<td>Register/Execute/Delete/Graph statemachine</td>
</tr>
<tr>
<td><code>/shell/{dsl}</code></td>
<td>Register/Execute/Delete Workflow DSL</td>
</tr>
</tbody>
</table>
FaaS Shell Components

- GUI Editor
- Amazon State Language (ASL) (JSON)
- ASL Compiler
- Flow DSL
- Deploy
- API Gateway
- FaaS Shell API End Point
- Invoke (REST)
- Message Queue (Kafka/built-in)
- DOT Generator
- Fasshell REPL (Interactive Shell)
- NoSQL DB (CouchDB/Coudant)
- AWS Lambda
- Azure Functions
- GCP Cloud Functions
- IBM Cloud Functions (Apache OpenWhisk)
- IFTTT (IF-This-Then-That)
- K5, Fission, etc
- Flow DSL Interpreter
- Cloud Provider Plugin
- Message Queue Plugin
- FaaS Shell Container
- program
- library
- definition
- packaging

※FaaS Shell
What is Function Workflow?

What is FaaS Shell and Why?

Demo

How does FaaS Shell?

Summary

Q&A
What is FaaS Shell?
- FaaS Shell is a shell which enables to exploit Serverless Functions across Multiple Clouds
- FaaS Shell helps us shift Single Strategy to Multi Cloud Strategy

Future Plan
- Azure LogicApp Workflow Definition Language

Visit github “https://github.com/NaohiroTamura/faasshell” and Try demos
- Especially if you are interested in Functional and Logic Programming

Welcome your feedback!
- naohirot@jp.fujitsu.com
Q & A


Translation Service Catalog Spec among Multiple Clouds

Surprisingly, AWS is not the leader in this area at this moment

<table>
<thead>
<tr>
<th>Cloud</th>
<th>Service Name</th>
<th>Current Supported Languages (more than 6,000 languages on the planet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>Amazon Translate</td>
<td>6 languages to and from English for Preview: (Arabic, Simplified Chinese, French, German, Spanish, and Portuguese) 6 additional languages will be supported soon, (Japanese, Russian, Italian, Traditional Chinese, Turkish, and Czech)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not yet</td>
</tr>
<tr>
<td>Azure</td>
<td>Translator Text</td>
<td>supports more than 60 languages</td>
</tr>
<tr>
<td>GCP</td>
<td>Translation API</td>
<td>supports more than 100 different languages</td>
</tr>
<tr>
<td>IBM</td>
<td>Language Translator</td>
<td>Support 14 languages Arabic, Chinese (Simplified &amp; Traditional), Dutch*, English, French, German, Italian, Japanese, Korean, Polish*, Portuguese (Brazil), Russian*, Spanish, and Turkish*  *These languages are supported with the early access preview</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Doesn’t work</td>
</tr>
</tbody>
</table>
Comparison

**Difference between Serverless Framework and FaaS Shell**

- Serverless Framework: Portable Solution for Serverless Function
- FaaS Shell: Portable Solution for Serverless Function Workflow

**Related work**

- Serverless Inc. OSS Event Gateway: Similar but different focus and approach
FaaS Shell Compiler and Workflow DSL

- Compiler supported Amazon State Language at first
  - Generate Workflow DSL

- `fsm()`
  - `pass()`
  - `task()`
  - `choice()`
  - `wait()`
  - `succeed()`
  - `fail()`
  - `parallel()`

```json
{
    "Comment": "A Hello World example using a Task state",
    "StartAt": "HelloWorld",
    "States": {
        "HelloWorld": {
            "Type": "Task",
            "End": true
        }
    }
}
```

```javascript
fsm([task('HelloWorld','arn:aws:lambda:us-east-2:410388484666:function:hello',[])])
```
FaaS Shell Runtime

- Execute Workflow DSL FSM (Finite State Machine)

- Implemented as Meta Interpreter
  - Very concise code in language such as Lisp or Prolog
  - This is important to keep agility for future changes

```prolog
%%% reduce(+Dsl, +In, -Out, +EnvIn, -EnvOut)
%%% reduce(fsm(Dsl), I, O, EI, EO) :- !,
%%%     reduce(Dsl, I, O, EI, EO).
reduce([], O, O, E, E) :- !.
reduce([A|B], I, O, EI, EO) :- !,
%%%     reduce(A, I, M, EI, EM),
     reduce(B, M, O, EM, EO),
%%%     reduce(A, I, O, EI, EO) :-
%%%     call(A, I, O, EI, EO).
```

- Entry
- Exit
- Recursive
- Call State
FaaS Shell Runtime Executes Workflow in Batch Mode

$ curl -ksX POST ${FAASSHELL_APIHOST}/statemachine/hello_world_task.json?blocking=true ¥ -H 'Content-Type: application/json' -d '{"input": {"name": "Curl"}}' -u $DEMO
{
 "asl": {
 "Comment":"A Hello World example of the Amazon States Language using a Task state",
 "StartAt":"HelloWorld",
 "States": {
 "HelloWorld": {
 "End":true,
 "Type":"Task"
 }
 }
 },
 "dsl":"fsm([task('HelloWorld',"arn:aws:lambda:us-east-2:410388484666:function:hello",[])]),"input": {"name":"Curl"},
 "name":"hello_world_task.json",
 "namespace":"demo",
 "output": {"payload":"Hello, Curl!"}
}
How to resolve Function Input and Output Mismatch?

- State Optional Parameter
  InputPath, ResultPath, OutputPath

**HelloWorld task output**

```
{"payload": "Hello, IFTTT!
```}

**SaveResult task input**

```
{"value1": "Hello, IFTTT!
```}

```
"InputPath": "$.payload",
"ResultPath": "$result.ifttt.value1",
"OutputPath": "$result.ifttt",
```

```
"Comment": "IFTTT as FaaS Demo",
"StartAt": "HelloWorld",
"States": {
  "HelloWorld": {
    "Type": "Task",
    "Resource": "frn:wsk:functions:::function:hello",
    "Next": "UpdateArg"
  },
  "UpdateArg": {
    "Type": "Pass",
    "InputPath": "$payload",
    "ResultPath": "$result.ifttt.value1",
    "OutputPath": "$result.ifttt",
    "Next": "SaveResult"
  },
  "SaveResult": {
    "Type": "Task",
    "Resource": "frn:ifttt:webhooks:::function:save_result",
    "End": true
  }
}
```
FaaS Shell REPL Executes Workflow Interactively

**Haskell REPL**

ghic> **hello** = e -> return (e+1)
ghic> **update_arg** = e -> return (e*2)
ghic> **save_result** = e -> return (e^2)
ghic> **pure** 1 >>= **hello** >>= **update_arg** >>= **save_result** 16

FaaS Shell `startsm` corresponds Haskell `pure`

**FaaS Shell REPL**

faasshell> **Hello** = task('HelloWorld',"frn:wsk:functions:::function:hello",[]),
| **UpdateArg** = pass('UpdateArg',[result_path('$.ifttt.value1'),input_path('$.payload'),out_path('$.ifttt')]),
| **SaveResult** = task('SaveResult','frn:ifttt:webhooks:::function:save_result',[]),
| `startsm`(_{name:"Repl"}), **Hello**, **Pass**, **SaveResult**.

Output=Congratulations! You've fired the save_result event.

FaaS shell comma operator ',' corresponds to Haskell bind operator '>>='.
Devises for supporting Multiple Clouds

■ Function Name Resolution
  ■ Resource Naming (Direct Invocation)
  ■ Function Discovery (Indirect Invocation)

■ Event State
  ■ Supported Event State in Workflow DSL
  ■ CNCF WG-Serverless Whitepaper v1.0 [1] (P.32) pointed out:
    > AWS provides “step function” primitives (state machine based primitives) for the user to specify its workflow, but step function does not allow specification of what event/events triggering what functions in the workflow.
  ■ Amazon States Language can handle Event through Activity Task, but it’s not straightforward.

Resource Naming - Function Name Resolution (1)

Resource Naming in Amazon States Language Specification [1]
- A Task State MUST include a “Resource” field, whose value MUST be a URI that uniquely identifies the specific task to execute.

AWS Step Functions Implementation
- ARN (Amazon Resource Name)

FaaS Shell Implementation
- FRN (FaaS Resource Name)
  AWS  " Resource":"frn:aws: ..." or "arn:aws: ...
  Microsoft " Resource":"frn:azure: ..."
  Google " Resource":"frn:gcp: ...
  IBM/OpenWhisk  "Resource":"frn:wsk: ...
  IFTTT  " Resource":"frn:ifttt: ...
  Fujitsu " Resource":"frn:k5: ...

Function Discovery - Function Name Resolution (2)

- Discover Function Semantically
  - Make use of Open Linked Data technology, OWL (Web Ontology Language)

- FaaS Shell Implementation
  - "Resource": "https://naohiotamura.github.io/faasshell/ns/faas#hello"

- Define Triple (Subject – Predicate – Object)
  - User owns Function
  - FaaS assigns FRN
  - FRN invokes Function

```
https://naohiotamura.github.io/faasshell/ns/faas#hello
```
There are several weather service on Internet

- Assume reusable function “weather” open linked RDF data or SPARQL end point is available
- "Resource": "https://weather.org/ns/faas#weatherβ"

Querying the linked data, we will figured out that “frn02”, “frn03”, “frn04” are callable using PublicUser “u1” account.
Event State

① Input {“name”: “Event”}

② Event “frn::states:::event:test”

③ Function “hello”

④ Output {“payload”: “Hello, Event!”}
FaasS Shell Deployment

① Development

Docker/Kubernetes/OpenShift

Faas Shell → CouchDB

Faasshell namespace

② Horizontal Scale Out

Kubernetes/OpenShift

Message Queue (Kafka, Zookeeper)

Faas Shell 1 (statefulset)

Faas Shell 2 (statefulset)

Faas Shell 3 (statefulset)

Service End Point

NoSQL DB (CouchDB)

NoSQL DB (CouchDB)

Message Queue (Kafka, Zookeeper)