OpenWhisk Compositions

Olivier Tardieu
IBM Research, NY

Goals

- Compose OpenWhisk functions
  - polyglot
    - functions & compositions
  - no double billing
    - cost ~= cost of composed functions
  - substitution
    - any function can be replaced by a composition (transparent to the caller)
    - a function does not need to know it is part of a composition when invoked
  - structured programming
    - sequences, conditionals, loops
    - hierarchical compositions: structured error handling, nested compositions
    - *parallelism and concurrency*
  - minimal runtime extension
    - minimal orchestration layer is part of the serverless runtime
    - programming model is not part of runtime extension
  - *orchestrate cloud services using functions for bridging gaps between services*
  - *reactive compositions (~ multiple events, correlation, streaming...)*

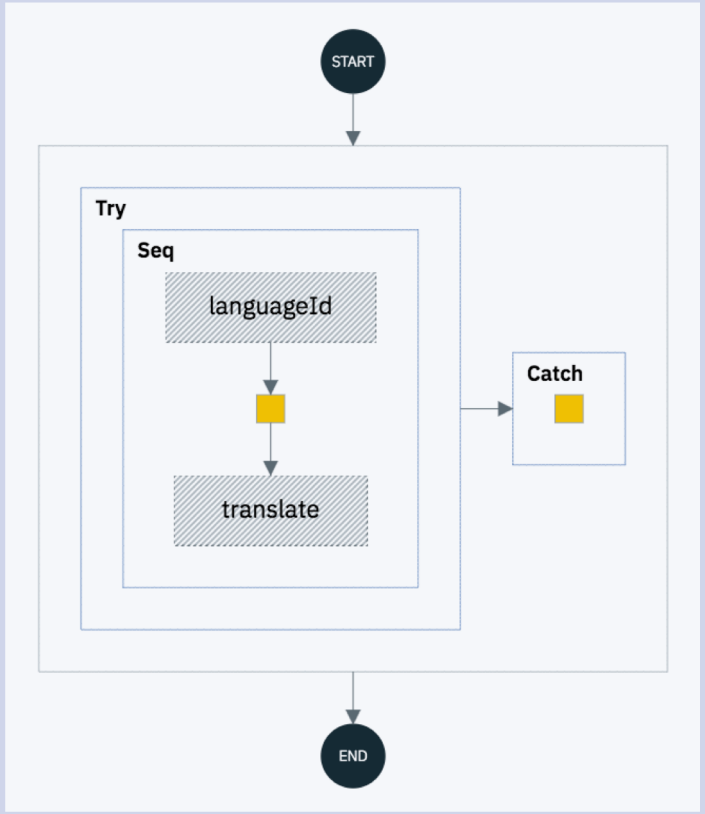
Compile and Run

- Compile: composer
  - https://github.com/ibm-functions/composer/
  - open source Node.js library (npm package) to program functions compositions
    - typical control-flow constructs: sequence, if, try, while... + functions

- Run: conductor actions
  - https://github.com/apache/incubator-openwhisk/blob/master/docs/conductors.md
  - extension of the OpenWhisk controller (PRs 3202, 3298, 3328)
  - a conductor action orchestrates a dynamic series of function invocations
    - unlike sequence actions, the functions to invoke are decided at run time
    - composition state is preserved in-between component invocations

Composer Example

```
composer.try(
   composer.seq(
      "languageId",
      p => ({
         translateFrom: p.language,
         translateTo: "en",
         payload: p.payload }),
      "translator"),
   _ => ({
      payload: "Cannot identify language"
   }))
```



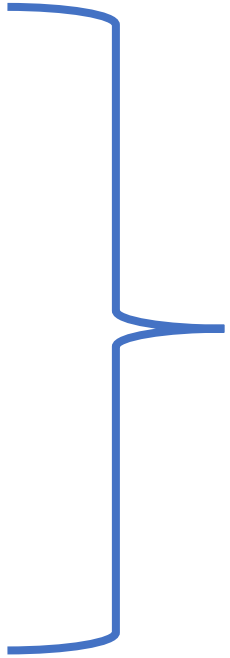
Composer's Constructs

Method	Description	Example
	named function	`"languageId"`
	inline function	`({ x, y }) => ({ product: x * y })`
seq	sequence	`composer.seq("languageId", "translator")`
if	conditional	`composer.if(condition, consequent, alternate)`
while	loop	`composer.while(condition, body)`
try	error handling	`composer.try(body, handler)`
retry	error recovery	`composer.retry(n, body)`
let	variable declaration	`composer.let({ x: 42 }, body)`
async	non-blocking invocation	`composer.async(body)`

Data Flow and State

- Data flow
  - the output of one function is the input of the next function

- State
  - position
    - program counter
  - execution context = stack
    - registered exception handlers
    - variable declarations
  - optional custom state
    - session id
    - callback
    - ...



could be externalized
- only track the session id

must externalized if too big

Conductor Actions

- A conductor action acts as a scheduler for a composition
  - must return a triplet { action, params, state }
    - if action is defined
      - invokes action on params producing result
      - reinvokes itself after action
        - input = result + state
    - if action is undefined
      - returns params

```
function main(params) {
  let step = params.$step || 0
  delete params.$step
  switch (step) {
    case 0: return { action: 'triple', params, state: { $step: 1 } }
    case 1: return { action: 'increment', params, state: { $step: 2 } }
    case 2: return { params }
  }
}
```

From Composer to Conductor Actions

- Composer generates a JSON representation for the composition
  - self-contained Abstract Syntax Tree
- Composer stitches
  - JSON composition
  - generic conductor action code (same for all composer compositions)
- Conductor action code
  - compiles the Abstract Syntax Tree to a Finite State Machine
  - interprets the finite state machine

JSON Format

```
{
    "type": "try",
    "body": {
        "type": "seq",
        "components": [
            {
                "type": "action",
                "name": "/_/languageId"
            },
            {
                "type": "function",
                "function": {
                    "exec": {
                        "kind": "nodejs:default",
                        "code": "p => ({\n translateFrom: p.language,\n translateTo: \"en\",\n payload: p.payload })"
                    }
                }
            },
            {
                "type": "action",
                "name": "/_/translator"
            }
        ]
    },
    "handler": {
        "type": "function",
        "function": {
            "exec": {
                "kind": "nodejs:default",
                "code": "_ => ({\n payload: \"Cannot identify language\"\n })"
            }
        }
    }
}
```

Goals

- Compose OpenWhisk functions
  - polyglot
    - functions & compositions
  - no double billing
    - cost ~= cost of composed functions
  - substitution
    - any function can be replaced by a composition (transparent to the caller)
    - a function does not need to know it is part of a composition when invoked
  - structured programming
    - sequences, conditionals, loops
    - hierarchical compositions: structured error handling, nested compositions
    - *parallelism and concurrency*
  - minimal runtime extension
    - orchestration is part of the serverless runtime
    - programming model is not part of runtime extension
  - *orchestrate cloud services using functions for bridging gaps between services*
  - *reactive compositions (~ events, correlation, streaming...)*