Serverless Composition of Serverless Functions

Olivier Tardieu
IBM Research
Serverless Computing (Wikipedia)

- **Serverless computing** is a cloud-computing execution mode in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity.

- **Serverless computing** can simplify the process of deploying code into production. Scaling, capacity planning and maintenance operations may be hidden from the developer or operator.

Serverless Computing
From Cloud to Serverless Cloud

- Why?
  - no resources to manage
  - automatic elastic scaling
  - pay for use not capacity
  - scale down to zero

- For what?
  - compute
  - storage
  - events

- How?
  - large numbers
  - keep it simple
  - pragmatic limits
Serverless Conf NYC 2017 Sponsors
Serverless Functions

Principles
Example
Limitations
Functions-as-a-Service

• Serverless platform for distributed event processing
  • run functions in response to events (or on demand)

• Serverless promises
  • no resources to manage
  • automatic elastic scaling
  • pay for use not capacity
  • scale down to zero

• Key metrics
  • thousand-way parallelism
  • 100ms billing granularity
  • $0.000017 per GB-s
  • avg. latency << container init.

• Constraints
  • no persistent state, no concurrency controls
Warm Latency (Node.js functions)

Overhead is defined as the time from request to response without the actual time the function took.

### Average

<table>
<thead>
<tr>
<th>Platform</th>
<th>Time (ms)</th>
<th>Data Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>73ms avg</td>
<td>(90,900)</td>
</tr>
<tr>
<td>Google Cloud Functions</td>
<td>770ms avg</td>
<td>(90,900)</td>
</tr>
<tr>
<td>IBM Cloud Functions</td>
<td>159ms avg</td>
<td>(41,856)</td>
</tr>
<tr>
<td>Azure Functions</td>
<td>791ms avg</td>
<td>(90,900)</td>
</tr>
<tr>
<td>Cloudflare Workers</td>
<td>103ms avg</td>
<td>(90,900)</td>
</tr>
</tbody>
</table>

### Percentiles

<table>
<thead>
<tr>
<th>Platform</th>
<th>Time (ms)</th>
<th>Data Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>67ms median</td>
<td>1089ms max</td>
</tr>
<tr>
<td>Google Cloud Functions</td>
<td>750ms median</td>
<td>14695ms max</td>
</tr>
<tr>
<td>IBM Cloud Functions</td>
<td>122ms median</td>
<td>25986ms max</td>
</tr>
<tr>
<td>Azure Functions</td>
<td>713ms median</td>
<td>17910ms max</td>
</tr>
<tr>
<td>Cloudflare Workers</td>
<td>91ms median</td>
<td>9715ms max</td>
</tr>
</tbody>
</table>

λ Serverless Benchmark

https://serverless-benchmark.com
Use Cases

• Asynchronous, concurrent, easy to parallelize into independent units of work

• Infrequent or has sporadic demand, with large, unpredictable variance in scaling requirements

• Stateless, ephemeral, without a major need for instantaneous cold start time

• Highly dynamic in terms of changing business requirements that drive a need for accelerated developer velocity
Apache OpenWhisk (Incubating)

• Open-source FaaS platform

OpenWhisk Supporters
https://openwhisk.apache.org/community.html
Programming Model
Demo

```javascript
// hello.js
function main ({ name }) {
    return { message: 'Hi ' + name }
}
```

$ wsk action create myAction hello.js

$ wsk action invoke myAction --result --param name John
{ message: 'Hi John' }

$ wsk trigger create myTrigger --feed /whisk.system/alarms/alarm
    --param cron "*/2 * * * *" 
    --param trigger_payload "{"name": "John"}"

$ wsk rule create myRule myTrigger myAction
Serverless Deployment

Function = code + runtime id

• Benefits for the developer
  • agility

• Benefits for the platform
  • less storage
  • less bandwidth
  • runtime reuse
  • stem cells
Function Container Life Cycle

- create
- /init
- prealloc
- /run
- suspend
- resume
- destroy

“cold”
“pre-warmed”
“hot”
“warm”

Maintain pool of “stem cell” containers for heavily used language runtimes

- grace period (50ms)
- grace period expires
- > idle limit or LRU eviction

< grace period (50ms)
Applications?

Did we go too far?

Abstracting the Back-end with FaaS, John McKim
https://serverless.zone/abstracting-the-back-end-with-faaS-e5e80e837362
Event Processing?

• The good
  • sporadic, ephemeral, independent event processing

• The bad
  • regular, frequent event processing

• The ugly
  • very high frequency, very low latency
  • stream processing (stateful, in-order)
Composition of Serverless Functions
Programming 101

• Write **small, simple, stateless functions**
  • complex functions are hard to understand, debug, and maintain
  • separate code from data structures

and compose them
Example: Sequences

• Problem statement
  • first invoke function languageId on text
  • then invoke function translator with the result

• Possible implementations
  • client-side composition
  • server-side composition
  • fusion
  • event-driven composition
  • primitive sequence
// open a connection to openwhisk
let wsk = require('openwhisk')()

// helper function
function invoke (name, params) {
    return wsk.actions.invoke({ name,
        params, blocking: true, result: true })
}

invoke('languageId', params)
.then(result => invoke('translator', result))
let wsk
function invoke (name, params) {
  return wsk.actions.invoke({ name,
    params, blocking: true, result: true })
}

function main (params) {
  wsk = wsk || require('openwhisk')()
  return invoke('languageId', params)
    .then(result => invoke('translator', result))
}
function languageld (params) {
    ...
}

function translator (params) {
    ...
}

function main (params) {
    let result = languageld(params)
    return translator(result)
}
Event-Driven Composition

$ wsk trigger create link
$ wsk rule create mySequence link translator

function trigger (name, params) {
  return wsk.triggers.invoke({ name, params })
}

function main (params) {
  return languageId(params)
  .then(result => trigger('languageIdResult', result))
}
Serverless Trilemma

The serverless trilemma: function composition for serverless computing
https://dl.acm.org/citation.cfm?id=3133855
Primitive Sequence

- requires no change or knowledge of the composed functions
- requires no change to the client code
- does not inflate cost
- *built into the OpenWhisk runtime*

```bash
# create a sequence action
$ wsk action create seq --sequence languageld,translator

# invoke a sequence action
$ wsk action invoke seq -P params.json -r
```
Serverless Composition of Serverless Functions

Apache OpenWhisk Composer
Example: The Weather Underground Industry Survey
Apache OpenWhisk Composer

• Javascript library to program compositions
  • data-flow, imperative style
  • synthesizes conductor actions

```javascript
composer.try(
  composer.seq(
    'languageId',
    p => Object.assign(p, { translateTo: 'en' }),
    'translator'),
  _ => ({ text: 'Cannot translate' }))
```

https://github.com/apache/incubator-openwhisk-composer
## Combinators

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>named function</td>
<td>'languageId'</td>
<td></td>
</tr>
<tr>
<td>anonymous function</td>
<td>({ x, y }) =&gt; ({ product: x * y })</td>
<td></td>
</tr>
<tr>
<td>seq</td>
<td>sequence</td>
<td>composer.seq(first, second)</td>
</tr>
<tr>
<td>if</td>
<td>conditional</td>
<td>composer.if(condition, consequent, alternate)</td>
</tr>
<tr>
<td>while</td>
<td>loop</td>
<td>composer.while(condition, body)</td>
</tr>
<tr>
<td>try</td>
<td>error handling</td>
<td>composer.try(body, handler)</td>
</tr>
<tr>
<td>let</td>
<td>variable declaration</td>
<td>composer.let({ variable: value }, body)</td>
</tr>
<tr>
<td>async</td>
<td>non-blocking invocation</td>
<td>composer.async(body)</td>
</tr>
<tr>
<td>par</td>
<td>parallel execution</td>
<td>composer.par('searchGoogle', 'searchBing') (double billing)</td>
</tr>
</tbody>
</table>
Conductor Actions

• Implicit loop around action with persistent state

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

https://github.com/apache/incubator-openwhisk/blob/master/docs/conductors.md
Weather Underground Maps

http://s.w-x.co/staticmaps/wu/wxtype/none/usa/animate.png
Weather Underground Website

https://www.wunderground.com/maps
Weather Underground Composition

• Composition periodically updates all the animated GIFs

```javascript
composer.let(
  {items: [], thumbnails: [], start: -1, params: {}, batch: 30},
  // generate and upload json files and thumbnails by walking directories
  composer.try(
    'tools/intellicast-action',
    args => ({result: 'something went wrong'})),
  x => {items = x.body.result.animate, thumbnails = x.body.result.thumbnail, params = x.body.params},
  composer.while(() => ++start < items.length,
  composer.async(
    x => (Object.assign(params, {value: items.slice(start, start+batch)})), 'tools/intellicast-animate-action')))```

Jeff Lu, A serverless approach to Weather Underground
https://developer.ibm.com/articles/a-serverless-approach-to-weather-underground/
Kui Shell

Jeff Lu, A serverless approach to Weather Underground
https://developer.ibm.com/articles/a-serverless-approach-to-weather-underground/

Kui Shell
http://kui.tools
AWS Step Functions

AWS Step Functions – Build Distributed Applications Using Visual Workflows
http://amzn.to/2gd1xzy
Platform9's Fission Workflows

https://techcrunch.com/2017/10/03/platform9s-fission-workflows-makes-it-easier-to-write-complex-serverless-applications/
const df = require("durable-functions");

module.exports = df.orchestrator(function*(context) {
  const x = yield context.df.callActivity("F1");
  const y = yield context.df.callActivity("F2", x);
  const z = yield context.df.callActivity("F3", y);
  return yield context.df.callActivity("F4", z);
});
Conclusions

• Serverless is here to stay

• Plenty of FaaS platforms to choose form
  • open-source platforms
  • hosted, managed services

• Growing set of capabilities

• Events?