Review of Serverless Frameworks

Kyriakos Kritikos | ICS-FORTH

Pawel Skrzypek | 7bulls.com
Outline

• Introduction and motivation
• Review Analysis
  • Goal
  • Scope
  • Search Process
  • Criteria
  • Results
• Challenges
Introduction

Melodic
Big data cloud

A single, universal platform for optimized deployment and management of applications in the cross-cloud environment.

Serverless extension to Melodic platform enabling the management of serverless components.
Motivation

• To unlock the serverless vendor lock-in
• To optimize serverless component deployment
• To make serverless usage more efficient and devops-friendly

Complete multicloud deployment platform - MELODIC
Architecture idea
Functionizer - initial architecture flow

1. Application deployment on Melodic (CAMEL)
   - Camel model enhancement - serverless

2. Setting parameters of application (CAMEL)
   - Extension to support serverless components

3. Calculation of initial deployment (CP Generator Solvers)
   - Extension to profiling and optimization

4. Deployment orchestration to selected Cloud Providers (Adapter)
   - Extension to orchestrated deployment of serverless components and adaptation

5. Deployment execution and metric collection (Cloudiator)
   - Serverless deployment execution and monitoring

6. Metrics collection & Aggregation
   - Extension to monitoring of serverless components

Legend:
- Current functionality on Melodic
- Extension delivered within the Functionizer

Should deployment be optimized?
Introduction

• Serverless computing attains momentum
  – Multiple advantages:
    • Zero administration
    • Infinite elasticity
    • Minimal cost
    • Capability to handle unanticipated workloads
  – Multiple applications:
    • Image processing
    • Video processing
    • Scientific computing
    • Edge computing
Introduction

• Traditional big cloud providers came into play
  – Offer serverless platforms
    • Mostly in beta version with known limitations
    – Plus added-value services to lock-in customers
      • e.g., trigger-oriented or state-handling
  • Lock-in issue can be addressed via the use of serverless platforms which
    – Abstract away from technical specificities
    – Make the life of the devops easier via the supply of serverless component development & deployment CLIs
Introduction

• Serverless frameworks differ wrt:
  – The level they abstract from
  – The level of support to the serverless application lifecycle

• Main question for devops:
  – Which serverless framework to choose based on the devops needs?
Review Goal

• We provide an answer to this question via a review on serverless frameworks based on carefully designed set of criteria spanning the serverless application lifecycle

• We view a serverless framework as: a software middleware that abstracts away from serverless platform specificities and eases the deployment and provisioning of multi-cloud serverless applications
Review Scope

• Two kinds of serverless frameworks reviewed:
  – Abstraction frameworks (e.g., serverless.com)
  – Provisioning frameworks (e.g., Fission)
    • Enable to operate mini-serverless platforms over existing clouds
• We have not reviewed frameworks which abstract from just one serverless platform
• We have not also reviewed proprietary frameworks
• Both latter are filtering criteria in search process
Review Search Process

• Multi-source search process utilising
  – Search engines (e.g., Google)
  – Scholarly repositories (e.g., Web of Science)

• Findings:
  – CNCF pointer to multiple frameworks:
    https://landscape.cncf.io/grouping=landscape&landscape=serverless
  – Numerous articles pointing or proposing such frameworks
Review Criteria

- Based on the (serverless) application lifecycle

Nomenclature:

- Normal flow
- Runtime adaptation flow
- Adaptation flow
Review Criteria

• Design
  – Composition: composition flow description
  – FaaSification: process to produce functions out of existing code

• Development:
  – Language: support for multiple languages
  – Function Development Kits (FDKs)
  – Integration: wrt other frameworks and platforms
Review Criteria

• Deployment
  – CI/CD
  – Versioning

• Testing: support to different types of testing

• Execution
  – Event coverage
  – Execution support: via a CLI or UI or both
Review Criteria

• Monitoring & Adaptation
  – Logging: level of logging supported
  – Metric support: richness of metric set used to monitor serverless components
  – Monitoring UI

• Security: support for both authentication & authorisation to regulate the controlled access to functions
<table>
<thead>
<tr>
<th>Phase</th>
<th>Criterion</th>
<th>Fission</th>
<th>Kubeless</th>
<th>Iron Functions</th>
<th>Sparta</th>
<th>Fn</th>
<th>Snafu</th>
<th>Serverless</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Compos.</td>
<td>Workflows</td>
<td></td>
<td></td>
<td>AWS Step</td>
<td>Flow</td>
<td></td>
<td>AWS Step</td>
</tr>
<tr>
<td></td>
<td>FaaSific.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Dev.</td>
<td>Language</td>
<td>NodeJS, Python, Ruby, Go, PHP, Bash, any linux exec.</td>
<td>NodeJS, Python, Ruby, Go, PHP, Ballerina</td>
<td>Go, .NET, Javascript, Java, Lambda, Python, Ruby, Rust</td>
<td>Go</td>
<td>Any</td>
<td>Python, Java</td>
<td>Javascript, C#, F#, Scala, Python, Java, Goland, Groovy, Kotlin, PHP, Swift</td>
</tr>
<tr>
<td></td>
<td>FDKs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
## Review Results

<table>
<thead>
<tr>
<th>Phase</th>
<th>Criterion</th>
<th>Fission</th>
<th>Kubeless</th>
<th>Iron Functions</th>
<th>Sparta</th>
<th>Fn</th>
<th>Snafu</th>
<th>Serverless</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploy.</td>
<td>CI/CD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unit</td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td>Event Cov.</td>
<td>HTTP, Cron, MQ</td>
<td>HTTP, Cron, MQ, Stream</td>
<td>HTTP, MQ, Alarm</td>
<td>HTTP, MQ, Stream</td>
<td>HTTP, MQ</td>
<td>HTTP, MQ, FS, Cron</td>
<td>HTTP, Cron, MQ, Stream</td>
</tr>
<tr>
<td>Support</td>
<td></td>
<td>CLI</td>
<td>Both</td>
<td>Both</td>
<td>CLI</td>
<td>Both</td>
<td>CLI</td>
<td>CLI</td>
</tr>
<tr>
<td>Metrics</td>
<td></td>
<td>Resource</td>
<td>(Succ.) Call Num, Exec. Time</td>
<td>Not Def.</td>
<td>Custom</td>
<td>Exec. Time</td>
<td>CloudWatch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UI</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Challenges

• Overall Vision: abstraction framework supporting the adaptive provisioning of mixed applications

• Two main directions to support this:
  – Integration of serverless frameworks with multi-cloud application management frameworks
  – Improvement of serverless frameworks wrt the application lifecycle
Challenges

- **Design**
  - **C1**: Novel design methods & techniques for mixed applications
  - **C2**: FaaSification of applications
    - FaaS-readiness tools
    - Improve FaaSification tools by also covering other languages
  - **C3**: Serverless component composition:
    - Reuse vast knowledge & experience in workflow modelling & scientific computing
    - Better integration with different types of events
Challenges

• Development
  – **C4**: integration of serverless frameworks as plugins in development frameworks
  – **C5**: FDK improvement
    • Better & more uniform coverage of progr. languages
    • FDK extensions over: (a) enhanced error handling; (b) proper data binding & capabilities to extend it; (c) arbitrary calls to any kind of function/component
Challenges

- **C6**: Deployment reasoning for mixed applications
  - C6.1: matching component requirements with cloud/platform capabilities
  - C6.2: appropriate formulation and solving of resp. optimisation problem

- **C7**: Modelling to support matching & reasoning:
  - C7.1: mixed application modelling covering all possible aspects
  - C7.2: cloud/platform offering modelling

- **C8**: Automatic (custom) serverless platform reconfiguration based on app. requirements and configuration patterns / details
Challenges

• Testing
  – C9: Unit testing spanning additional languages
  – C10: Development and/or extending existing integration methods for mixed application integration testing

• Execution
  – C11: Realisation of Event Gateways based on right abstraction methods & concepts from event programming
Challenges

• Monitoring & Adaptation
  • C12: Advanced monitoring & evaluation capabilities
    – Support for custom metrics
    – Metric aggregation
    – Mechanisms for event pattern detection
  • C13: Cross-level adaptation of mixed applications along with
    – The ability to sense the “problematic” situations
    – The ability to semi-automatically generate the right adaptation rules
Functionizer & Melodic

Melodic website:

www.melodic.cloud

Download and develop:

https://melodic.cloud/download.html