**VIETNAM NATIONAL UNIVERSITY HCM UNIVERSITY OF INFORMATION TECHNOLOGY**

**FACULTY OF COMPUTER NETWORKS AND COMMUNICATIONS**



**Ảnh có chứa màu trắng, thiết kế

Mô tả được tạo tự độngẢnh có chứa hình mẫu, Đồ họa, biểu tượng, Decal ô tô

Mô tả được tạo tự động**

**BÁO CÁO ĐỒ ÁN**

**NGHIÊN CỨU, TRIỂN KHAI APACHE OPENWHISK**

**GVHD: ThS. BÙI THANH BÌNH**

**Lớp: NT533.N21**

Nhóm 10

Diệc Thành – 20521909

Mai Phước Sang – 20520735

Trần Triệu Thiên – 20521954

*Thành phố Hồ Chí Minh, ngày 05 tháng 05 năm 2023*

# NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

*……., ngày……...tháng……năm 2023*

**Người nhận xét**

*(Ký tên và ghi rõ họ tên****)***

NỘI DUNG

[NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN 1](#_Toc136436557)

[A. DANH MỤC CÁC BẢNG, HÌNH VÀ BIỂU ĐỒ 3](#_Toc136436558)

[B. CƠ SỞ LÝ THUYẾT 3](#_Toc136436559)

[I) Khái niệm căn bản 3](#_Toc136436560)

[1) FaaS 3](#_Toc136436561)

[2) Apache Openwhisk 4](#_Toc136436562)

[II) Tổng quan về Openwhisk 5](#_Toc136436563)

[1) Ngôn ngữ hỗ trợ 5](#_Toc136436564)

[2) Ứng dụng 5](#_Toc136436565)

[3) Các thành phần trong Openwhisk 7](#_Toc136436566)

[III) Cơ chế và mô hình 9](#_Toc136436567)

[1) Cơ chế 9](#_Toc136436568)

[2) Mô hình 10](#_Toc136436569)

[3) Thực thi 11](#_Toc136436570)

[4) Giới hạn của hệ thống 14](#_Toc136436571)

[C. THAM KHẢO, TRÍCH DẪN 15](#_Toc136436572)

# DANH MỤC CÁC BẢNG, HÌNH VÀ BIỂU ĐỒ

|  |  |  |
| --- | --- | --- |
| **Hình** | **Tên bảng / biểu đồ / hình** | **Trang** |
| 1 | Sơ đồ cơ chế của Apache OpenWhisk | 5 |
| 2 | Mô hình của Apache OpenWhisk | 6 |
| 3 | Dịch vụ cần thiết của Apache OpenWhisk | 8 |

# CƠ SỞ LÝ THUYẾT

## Khái niệm căn bản

### FaaS

#### Định nghĩa

* Function as a Service, hay còn được gọi là FaaS, là một cách serverless để chạy function trong môi trường đám mây. Với sự giúp đỡ của FaaS, các developer có thể tập trung function mà không cần quan tâm đến phía Server như deploy, duy trì cơ sở hạ tầng, hệ điều hành hay bảo mật. Thay vào đó, nền tảng serverless sẽ tự động quản lý các tài nguyên phía máy chủ, chỉ chạy function khi được gọi và dừng khi hoàn thành. FaaS được sử dụng chủ yếu trong bối cảnh tính toán sự kiện, trong đó các hàm được kích hoạt bởi một sự kiện cụ thể như message queue, các HTTP request, vv.
* Nhiều nền tảng hỗ trợ FaaS đã xuất hiện trên thị trường, ví dụ như AWS Lambda, Azure Functions, Google Cloud Functions và nhiều hơn nữa. Những nền tảng này hỗ trợ nhiều ngôn ngữ lập trình, cung cấp các tiện ích để phát triển, thử nghiệm và triển khai các function.[1]

#### Ưu điểm

* Tốc độ – Improved developer velocity: Với FaaS, lập trình viên có thể có thêm thời gian để viết về logic của ứng dụng, bớt thời gian lo lắng về servers và deploys. Điều đó cũng đồng nghĩa với việc development nhanh hơn.
* Khả năng mở rộng – Scability: Dev không còn phải lo vấn đề về quá tải khi traffic cao/ Server sẽ cung cấp và handle tất cả các vấn đề liên quan với scaling.
* Giá cả – Cost efficiency: Khách hàng chỉ trả tiền cho thời gian mà họ sử dụng, không phải phung phí tiền cho các tài nguyên cloud khác.[2]

#### Nhược điểm

* Giới hạn chức năng: Các function trong FaaS là các đoạn mã nhỏ được sử dụng để thực hiện các nhiệm vụ đơn giản (Function do only one thing). Nếu bạn muốn viết các hàm phức tạp, FaaS có thể không phải là lựa chọn tốt nhất của bạn.
* Khó khăn trong quản lý chức năng: Vì bạn sẽ phát triển các đoạn mã chức năng nhỏ, việc quản lý một số lượng lớn các chức năng có thể gặp khó khăn khi doanh nghiệp của bạn tiếp tục phát triển.
* Hiệu năng thấp: vì mỗi request cần phải có thời gian để start.[2]

### Apache Openwhisk

* OpenWhisk là 1 nền tảng Serverless phân tán mã nguồn mở, thực thi các chức năng phản hồi các Event ở mọi quy mô, được sử dụng bởi IBM Cloud.
* OpenWhisk quản lý cơ sở hạ tầng, các máy chủ và mở rộng quy mô bằng các Docker container để người dùng có thể tập trung xây dựng các ứng dụng hiệu quả.
* Nền tảng OpenWhisk hỗ trợ 1 mô hình lập trình trong đó các nhà phát triển viết logic mang chức năng (được gọi là Actions), bằng bất kỳ ngôn ngữ lập trình nào được hỗ trợ, có thể được lên lịch động và chạy để phản hồi các Event (thông qua các Trigger) từ các nguồn bên ngoài (Feeds) hoặc từ các yêu cầu HTTP.
* Một dự án bao gồm một Giao diện dòng lệnh (CLI – Command Line Interface) dựa trên REST API cùng với các công cụ khác hỗ trợ đóng gói, các dịch vụ danh mục và nhiều tùy chọn triển khai container phổ biến.[3]

## Tổng quan về Openwhisk

### Ngôn ngữ hỗ trợ

* Hiện tại, OpenWhisk hỗ trợ khá nhiều ngôn ngữ lập trình thông dụng, cung cấp cho người dùng sự linh hoạt trong việc lựa chọn ngôn ngữ lập trình để viết các hàm cho ứng dụng serverless của mình.
  + JavaScript / Node.js
  + Python
  + Swift
  + Java
  + PHP
  + Go
  + Ruby
  + .NET Core
* Các ngôn ngữ này dược hỗ trợ thông qua các Docker runtime container giúp người dùng có thể sử dụng ngôn ngữ để phát triển và triển khai ứng dụng serverless nhanh chóng và dễ dàng hơn.

### Ứng dụng

* Apache OpenWhisk là một nền tảng serverless-computing mạnh mẽ, linh hoạt và có thể mở rộng. Nó cung cấp một mô hình thực thi dựa trên sự kiện (event-driven) cho phép triển khai và quản lý các ứng dụng mà không cần lo lắng về việc quản lý hạ tầng. Dưới đây là một số ví dụ về các ứng dụng phổ biến của OpenWhisk:
* Microservices: OpenWhisk hỗ trợ việc triển khai và quản lý các microservices. Với tính chất mô-đun và khả năng tự mở rộng của OpenWhisk, người dùng có thể triển khai các đoạn mã độc lập nhau để xử lý các chức năng cụ thể. Việc triển khai và quản lý các microservices trở nên dễ dàng hơn và cho phép phát triển ứng dụng theo cách linh hoạt.
* Ứng dụng web: OpenWhisk không chỉ hỗ trợ các ứng dụng dựa trên sự kiện, mà còn có thể xây dựng các ứng dụng web. Bằng cách kết hợp OpenWhisk với các công cụ nhỏ như Node.js, bạn có thể xây dựng các ứng dụng web dễ dàng và tiết kiệm chi phí so với việc chạy một quy trình máy chủ trên nền tảng PaaS.
* IoT: OpenWhisk cung cấp một giải pháp tuyệt vời cho việc triển khai ứng dụng IoT. Với khả năng mở rộng linh hoạt và khả năng xử lý tải lớn, OpenWhisk có thể xử lý dữ liệu đến từ các thiết bị IoT và thực hiện các xử lý tùy chỉnh trên dữ liệu đó. Việc triển khai các ứng dụng IoT trở nên dễ dàng và linh hoạt hơn nhờ sự linh hoạt của OpenWhisk.

A picture containing text, screenshot, software, multimedia

Description automatically generated

* Backend API: OpenWhisk hỗ trợ tự động tạo ra REST API cho các action, giúp người dùng nhanh chóng xây dựng các API mà không cần lo lắng về việc quản lý máy chủ. Bạn có thể kết nối các công cụ quản lý API như IBM API Connect với OpenWhisk để quản lý API một cách dễ dàng.
* Mobile backend: OpenWhisk cung cấp một giải pháp tuyệt vời cho việc triển khai logic phía máy chủ cho ứng dụng di động. Điều này cho phép các nhà phát triển tập trung vào phát triển ứng dụng di động mà không cần lo lắng về việc quản lý máy chủ. OpenWhisk cũng hỗ trợ Swift, giúp người dùng tái sử dụng kỹ năng lập trình iOS hiện có.
* Xử lý dữ liệu: OpenWhisk hỗ trợ xử lý dữ liệu đa dạng, bao gồm xử lý dữ liệu cấu trúc và phi cấu trúc. Người dùng có thể cấu hình OpenWhisk để phản ứng với thay đổi dữ liệu và thực hiện các xử lý tùy chỉnh trên dữ liệu đó. Việc này giúp xây dựng các ứng dụng linh hoạt và dễ dàng thích ứng với yêu cầu thay đổi.
* Công nghệ nhận thức (Cognitive): OpenWhisk có thể kết hợp với các công nghệ nhận thức để tạo ra các ứng dụng mạnh mẽ. Ví dụ, bạn có thể sử dụng các dịch vụ như IBM Alchemy API và Watson Visual Recognition để trích xuất thông tin hữu ích từ video mà không cần xem video thực tế.

### Các thành phần trong Openwhisk

1. *Function và Event*

* Trong OpenWhisk, các function được sử dụng để hoàn thành các tác vụ. Một hàm thường là một đoạn mã nhận một số input và phản hồi lại với output. Điều quan trọng cần lưu ý ở đây là sự stateless của function.
* Nếu thiết kế 1 fuction là statefull, nó giới hạn khả năng scaling của hệ thống vì sẽ cần 1 storage cho CSDL. Quan trọng hơn, hệ thống sẽ cần một công cụ để đồng bộ trạng thái giữa các lần kích hoạt làm chi phí đắt đỏ hơn.
* Môi trường OpenWhisk quản lý cơ sở hạ tầng,và waiting cho 1 event xảy ra, khi đó function sẽ được gọi. Xử lý event là hoạt động quan trọng nhất mà môi trường serverless quản lý. Các function được viết để xử lý và phản hồi đúng khi event xảy ra - ví dụ, yêu cầu từ người dùng hoặc sự xuất hiện của dữ liệu mới.

[5]

1. *Action và cấu tạo của Action*

* Một action là một đoạn mã, được viết bằng một trong các ngôn ngữ lập trình được hỗ trợ. Khi gọi , action sẽ nhận một số parameter dưới dạng input.
* Để tiêu chuẩn hoá việc truyền parameter giữa nhiều ngôn ngữ lập trình, OpenWhisk sử dụng định dạng JavaScript Object Notation (JSON) vì nó khá đơn giản và có thư viện để mã hóa và giải mã định dạng này có sẵn cho hầu hết các ngôn ngữ lập trình.
* Các parameter được truyền cho các action trong các đối tượng JSON ,được xem như 1 string sẽ được action nhận vào khi bắt đầu và xử lý. Kết thúc quá trình xử lý, mỗi action phải trả vềmột kết quả, dưới dạng giá trị JSON.

[5]

1. *Dịch vụ cần thiết*

* OpenWhisk được xây dựng trên nền tảng các dự án mã nguồn mở phổ biến:
  + - * Nginx: một proxy reverse và là một web server hiệu suất cao.
      * CouchDB: cơ sở dữ liệu NoSQL document-oriented có khả năng mở rộng dễ dàng.
      * Kafka: hệ thống message publish/subscribe phân tán, hiệu suất cao.
* Ngoài ra, OpenWhisk còn bao gồm các thành phần chính sau:
  + - * Controller: quản lý các thực thể, xử lý các chuỗi trigger và định hướng các action.
      * Invoker: khởi chạy các container để thực thi các action.
      * Action Containers: thực hiện các action.

[5]

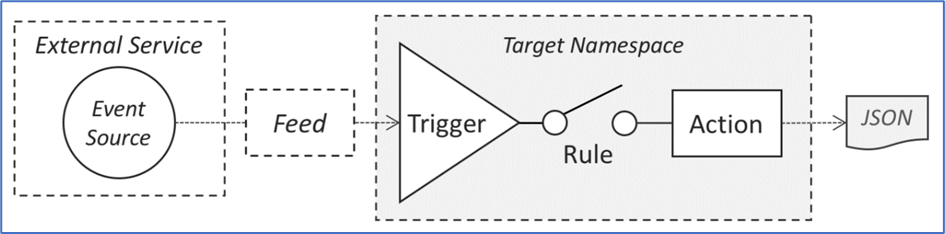
1. *Thực thể*

* Các action, trigger và rule của OpenWhisk thuộc về một namespace hoặc một gói.
* Tên đầy đủ của một thực thể có định dạng / namespaceName[/packageName] / entityName. Phương pháp đặt tên này sẽ được sử dụng chung với OpenWhisk CLI .

| **Tên đầy đủ** | **Alias** | **Namespace** | **Package** | **Name** |
| --- | --- | --- | --- | --- |
| /whisk.system/cloudant/read |  | /whisk.system | cloudant | read |
| /myOrg/video/transcode | video/transcode | /myOrg | video | transcode |
| /myOrg/filter | filter | /myOrg |  | filter |

## Cơ chế và mô hình

### Cơ chế



Hình 1. Sơ đồ cơ chế của Apache OpenWhisk

* Event Source: các sự kiện được tạo ra từ những dịch vụ thường chỉ ra những thay đổi trong dữ liệu hoặc tự mang dữ liệu. 1 số ví dụ về nguồn sự kiện phổ biến:
  + Tin nhắn đến trên hàng đợi tin nhắn
  + Thay đổi trong cơ sở dữ liệu
  + Thay đổi trong Document Store
  + Tương tác trang web hoặc ứng dụng web
  + Các dịch vụ API đang được gọi
  + Các framework IoT chuyển tiếp dữ liệu cảm biến thiết bị
* Actions: là các chức năng phi trạng thái (đoạn mã) chạy trên nền tảng OpenWhisk. Các Action đóng gói logic ứng dụng để được thực thi đáp ứng các Event. Các Action có thể được gọi thủ công bằng OpenWhisk REST API, OpenWhisk CLI, các API đơn giản và do người dùng tạo hoặc được tạo tự động thông qua các Trigger.
* Triggers: là các kênh được đặt tên cho các lớp hoặc loại sự kiện được gửi từ Nguồn sự kiện (Event Source).
* Rules: được sử dụng để liên kết 1 trigger với 1 action. Sau khi loại liên kết này được tạo, mỗi khi 1 event kích hoạt thì hành động tương ứng sẽ được gọi. [4]
* Cơ chế hoạt động: OpenWhisk dựa trên kiến trúc hướng sự kiện trong đó hầu hết các Action được thực thi khi các event diễn ra. Bản thân Trigger được kích hoạt với 1 từ điển các cặp khóa-giá trị (các tham số) đến từ Nguồn sự kiện và cho phép định cấu hình các giá trị mặc định tùy chọn, nhờ đó giúp đảm bảo dữ liệu tương thích với mọi Action được liên kết. Các Rule cho phép cùng 1 Trigger được liên kết với nhiều Action, cũng như cho phép bật hoặc tắt tự động hóa cụ thể 1 cách linh hoạt mà không phá hủy định nghĩa Trigger.

### Mô hình

A picture containing text, screenshot, diagram, design

Description automatically generated

Hình 2. Mô hình của Apache OpenWhisk

* Apache OpenWhisk, là một nền tảng serverless cloud mã nguồn mở. Nó hoạt động bằng cách thực thi các function (hay action) đáp lại các event. Các event có thể xuất phát từ nhiều nguồn khác nhau, bao gồm hẹn giờ, cơ sở dữ liệu, message queue hoặc các trang web như Slack hoặc GitHub.
* OpenWhisk chấp nhận source code làm đầu vào để thực thi một lệnh đơn với giao diện dòng lệnh (CLI), sau đó vận chuyển các dịch vụ thông qua web tới nhiều consumer, chẳng hạn như các trang web khác, các ứng dụng di động hoặc các dịch vụ dựa trên REST APIs. [5]

Ảnh có chứa văn bản, biểu đồ, Song song, hàng

Mô tả được tạo tự động

Hình 3. Dịch vụ cần thiết của Apache OpenWhisk

### Thực thi

Để thực thi một hành động trong OpenWhisk, các bước sau được thực hiện nội bộ:

* Người dùng gửi request đến OpenWhisk để kích hoạt một action. Action bao gồm tên action, các tham số và bất kỳ thông tin xác thực nào cần thiết.
* Đầu vào hệ thống:
  + nginx
* Lệnh được gửi thông qua CLI wsk thực chất là một HTTP request đối với hệ thống OpenWhisk. Lệnh được đưa đến điểm vào đầu tiên của hệ thống là nginx, một Proxy reverse Server. Nó chủ yếu được sử dụng chuyển tiếp HTTP phù hợp đến thành phần tiếp theo.
  + Controller
* Sau khi nhận được các HTTP request, nginx sẽ chuyển tiếp nó cho Controller - thành phần tiếp theo trong OpenWhisk. Nó là một phiên bản dựa trên Scala của REST API cung cấp như giao diện cho tất cả những gì mà user có thể làm, bao gồm các yêu cầu CRUD cho các thực thể trong OpenWhisk và invoke các action.
* Controller của OpenWhisk nhận request và xác thực thông tin xác thực của người dùng. Sau đó, nó kiểm tra cơ sở dữ liệu của hệ thống (CouchDB) để tìm action đã chỉ định.
* Sau khi Controller đã tìm thấy action, nó kiểm tra xem liệu nó có cần được khởi tạo không. action chưa được khởi tạo, Controller tạo một instance cho action và khởi tạo nó với các dependency cần thiết.
* Nếu action đã được khởi tạo, Controller gửi yêu cầu đến Invoker để thực thi action.
* Xác thực và phân quyền: CouchDB
  + Bây giờ Controller xác minh user là ai (Authentication) và xem liệu rằng user có đặc quyền để làm điều đó với thực thể đó hay không (Authorization). Các chứng chỉ trong request sẽ được xác minh qua bằng cơ sở dữ liệu các đối tượng trong CouchDB.
  + Trong trường hợp này, nó sẽ kiểm tra rằng người dùng tồn tại trong cơ sở dữ liệu OpenWhisk và rằng nó có quyền để invoke action nào đó hay không.
* Lấy hành động: CouchDB
  + Khi Controller đã chắc chắn rằng user được phép và có quyền để invoke action của mình, nó sẽ lấy action này từ cơ sở dữ liệu whisks trong CouchDB.
  + Hồ sơ của action chứa chủ yếu code thực thi với các tham số mặc định mà bạn muốn truyền vào action của mình, hợp nhất với các tham số bạn đã bao gồm trong invoke request. Cũng như nó chứa các ràng buộc tài nguyên đã được áp đặt trên thực thể sắp thực thi, chẳng hạn như bộ nhớ mà nó được phép tiêu thụ.
* Invoke action: Load Balancer
  + Load Balancer, là một phần của Controller, có tầm nhìn toàn cục về các bộ thực thi có sẵn trong hệ thống bằng cách kiểm tra trạng thái của chúng một cách liên tục. Những bộ thực thi đó được gọi là Invokers. Load Balancer, biết Invokers nào được sẵn có, chọn một trong số chúng để triệu hồi hành động được yêu cầu.
* Hàng đợi: Kafka
  + Có hai trường hợp xấu có thể xảy ra đối với invoke request :
* Hệ thống gặp sự cố, khiến cho invoke request bị mất
* Hệ thống bị quá tải, nên invoke request cần phải đợi cho đến khi hoàn tất các invoke request khác trước tiên.
  + Cách giải quyết cho cả hai trường hợp này đều là Kafka, "hệ thống message publish-subscribe thông lượng cao, xử lý phân tán". Từ đó, Controller và Invoker sẽ giao tiếp thông qua các tin nhắn được đệm và được lưu trữ bởi Kafka. Điều đó giúp giảm gánh nặng của việc đệm trong bộ nhớ, ra khỏi cả Controller và Invoker đồng thời đảm bảo rằng các tin nhắn không bị mất trong trường hợp hệ thống gặp sự cố, tránh khỏi rủi ro OutOfMemoryException.
  + Để invoke action, Controller truyền một tin nhắn đến Kafka, chứa thông tin về action để triệu hồi và các tham số cần truyền đến action đó. Tin nhắn này được đưa đến cho Invoker mà Controller đã chọn ở trên từ danh sách các invoker có sẵn.
  + Khi Kafka xác nhận rằng nó đã nhận được tin nhắn, HTTP request tới user sẽ được phản hồi với một ActivationId. User sẽ sử dụng nó sau này để truy cập cụ thể đến kết quả của invoke.
* Invoke mã thực thi : Invoker
  + Invoker là trái tim của OpenWhisk. Trách nhiệm của Invoker là invoke một action. Nó cũng được thực hiện bằng Scala. Và để thực hiện các action một cách cô lập và bảo mật, Docker sẽ được sử dụng.
  + Docker được sử dụng để thiết lập container cho mỗi hành động mà chúng ta triệu hồi một cách nhanh chóng, cô lập và kiểm soát. Đối với mỗi action invoke, một Docker container được bắt đầu, mã thực thi được đưa vào cùng với các tham số được cung cấp. Thực hiện xong thì kết quả được lấy ra, container sẽ được hủy.
* Action nhận yêu cầu và xử lý nó theo code đã được lập trình. Action đọc parameter được truyền dưới dạng JSON và tạo ra một response trong định dạng JSON.
* Sau khi hoàn tất việc thực thi, container trả lại kết quả cho Invoker.
* Lưu trữ kết quả: CouchDB
  + Khi Invoker nhận được kết quả, kết quả sẽ được lưu vào cơ sở dữ liệu activations qua ActivationId đã được đề cập ở trên trong CouchDB.
* Invoker trả về kết quả cho Controller.
* Controller trả về kết quả cho người dùng.
* Tổng thể, quá trình thực thi là sự kết hợp của nhiều lớp và thành phần hoạt động cùng nhau một cách trơn tru để cung cấp cho end user một trải nghiệm đơn giản.

### Giới hạn của hệ thống

* OpenWhisk có một số giới hạn hệ thống, bao gồm bao nhiêu bộ nhớ mà một hành động có thể sử dụng và bao nhiêu lần gọi hành động được phép trong một phút.
* Dưới đây là bảng liệt kê các giới hạn mặc định cho các hành động :

| **Giới hạn** | **Mô tả** | **Cấu hình** | **Đơn vị** | **Mặc định** |
| --- | --- | --- | --- | --- |
| timeout | 1 container không thể chạy quá N ms | Mỗi action | Ms | 60000 |
| memory | 1 container không thể dược cấp phát hơn N MB bộ nhớ | Mỗi action | MB | 256 |
| logs | 1 container không thể ghi hơn N MB vào stout | Mỗi action | MB | 10 |
| instances | 1 action không thể có qua nhiều container so với giá trị này | Mỗi action | number | Theo giới hạn của namespace |
| codeSize | Kích thước tối đa của code | Có thể tùy biến, với mỗi action | MB | 48 |
| parameters | Kích thước tối đa của tham số có thể truyền vào | Mỗi action/package/trigger | MB | 1 |
| result | Kích thước tối đa của kết quả sau khi thực hiện action | Mỗi action | MB | 1 |

# THAM KHẢO, TRÍCH DẪN

[1] “Serverless vs. Function-as-a-Service (FaaS): Which One to Choose?” https://kubesphere.io/blogs/serverless-vs-faas/ (accessed May 12, 2023).

[2] “Tìm hiểu về Serverless, FaaS, deploy functions lên Firebase Cloud Functions,” Sep. 18, 2019. https://viblo.asia/p/tim-hieu-ve-serverless-faas-deploy-functions-len-firebase-cloud-functions-yMnKMw0El7P (accessed May 12, 2023).

[3] “Documentation.” https://openwhisk.apache.org/documentation.html (accessed May 12, 2023).

[4] “Build a serverless app with Node.js and OpenWhisk - LogRocket Blog.” https://blog.logrocket.com/build-serverless-app-node-js-openwhisk/ (accessed May 12, 2023).

[5] “1. Serverless and OpenWhisk Architecture - Learning Apache OpenWhisk [Book].” https://www.oreilly.com/library/view/learning-apache-openwhisk/9781492046158/ch01.html (accessed May 12, 2023).