Các ngôn ngữ được 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

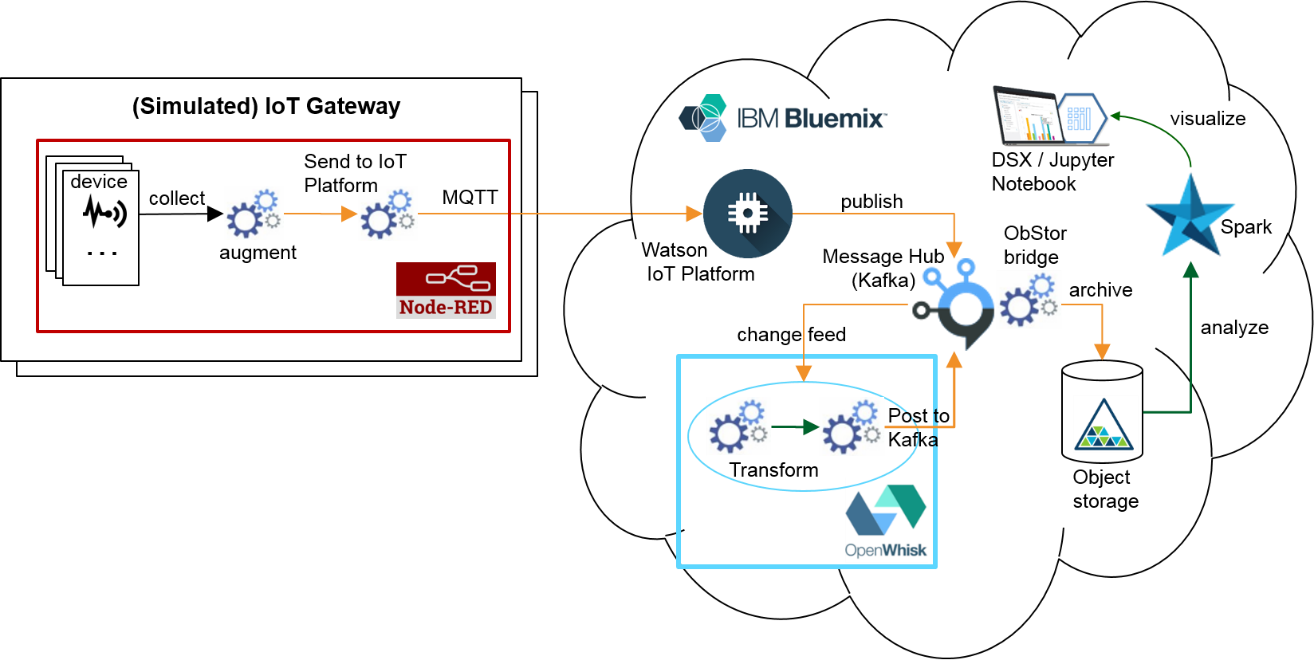
https://github.com/apache/openwhisk/blob/master/docs/use\_cases.md

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.



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.

Bổ sung về Tổng quan về hệ thống

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 |

Sự tương tác của các thành phần

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.

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 actin: 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.

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.

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 qua 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 |