Apache OpenWhisk

Distributed Software Systems, 2024/25

Davide De Rosa

Introduction to Serverless Computing

Introduction to Serverless Computing

Serverless Computing has transformed how we develop and deploy cloud applications. Traditional cloud computing required developers to manage virtual machines or containers, while **Serverless** eliminates this burden.
This model operates through **small**, **stateless functions** triggered by **events**, with the **cloud provider** managing backend infrastructure. Key benefits include:

- **Autoscaling** to handle varying loads.
- A **pay-as-you-go** model, where costs align with actual function usage.

However, Serverless isn't without its challenges. Cold starts, limited runtime languages, and debugging complexities still hinder its adoption.
Despite this, serverless computing has become integral to modern architectures, thanks to its simplicity and cost efficiency.

Introduction to Serverless Computing

Over the past decade, cloud platform hosting has evolved significantly:

- Initially, organizations purchased or rented physical servers to run applications, incurring costs for both the hardware and its maintenance.
- This shifted with the adoption of **virtualization**, which allowed a single physical server to function as multiple software-defined Virtual Machines (VMs), enhancing flexibility and resource utilization. **Containerization** emerged as a further refinement, combining aspects of virtualization with configuration management.
- The introduction of **Platform-as-a-Service** (**PaaS**) took abstraction further, freeing users from managing servers and deployment processes.
- The latest advancement, serverless computing, builds on *PaaS* by enabling deployment of small code fragments that can autonomously scale, supporting the creation of self- scaling applications.

Serverless Architecture

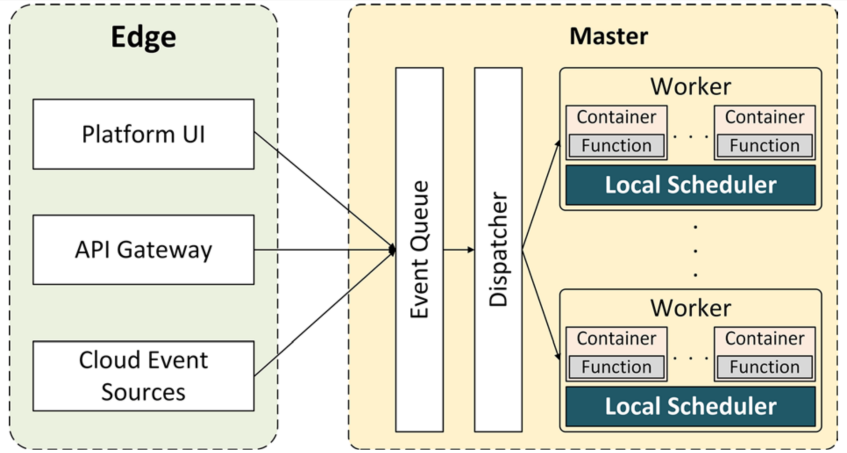
Serverless Architecture

At its core, Serverless Computing revolves around simplicity and efficiency. Functions run in isolated containers, with each triggered by specific events like HTTP requests or database updates. The process begins when an event triggers the platform:

- An **event queue** receives the trigger and informs a dispatcher.
- The **dispatcher** assigns the function to a **worker node**.
- If a **worker** is free, it executes the function. Otherwise, the request is queued.

A load balancer ensures smooth operations, though challenges like resource contention and cold starts remain. To address this, providers pre-warm containers, minimizing latency during new invocations.

Serverless Architecture



Apache Openwhisk

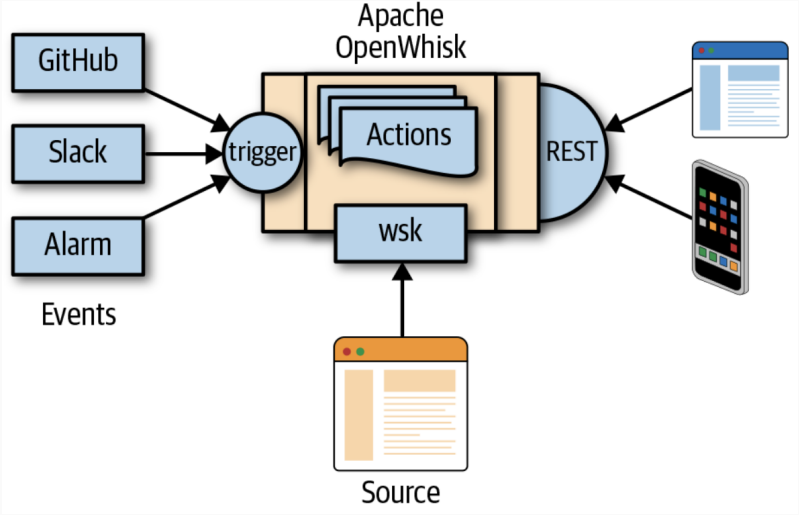
Apache Openwhisk

Apache Openwhisk is **IBM**'s **open-source** serverless platform. It competes with giants like *AWS Lambda* and *Google Cloud Functions* by offering flexibility and openness. Developers use Openwhisk to build applications requiring *real-time processing*, *IoT data handling*, or *serverless APIs*.
Its design is **event-driven**, and its programming model is built around three elements:

- **Actions**: Stateless functions that process events.
- **Triggers**: Events that activate functions.
- **Rules**: Bindings connecting triggers to actions.

What sets Openwhisk apart is its adaptability. It supports deployment on platforms like *Kubernetes* and *OpenShift*, and its CLI tool makes it accessible across operating systems.

Apache Openwhisk



Apache Openwhisk Key Features

- **Deploys Anywhere**: with its container-based architecture, Openwhisk offers versatile deployment options, supporting local setups and cloud infrastructures.
- **Supports Any Programming Language**: Openwhisk is compatible with a wide range of programming languages. For unsupported platforms or languages, users can easily create and customize executables using Docker.
- **Integration Support**: Openwhisk enables easy integration of developed Actions with popular services through pre-built packages.
- **Rich Function Composition**: functions written in multiple programming languages can be packaged with Docker for flexible invocation options, including synchronous, asynchronous, or scheduled execution.
- **Scalability and Resource Optimization**: Openwhisk allows Actions to scale instantly. Resources scale automatically to match demand, pausing when idle, so users only pay for actual usage with no costs for unused resources.

Apache Openwhisk
Main Architectural Drivers

Apache Openwhisk Main Architectural Drivers

ciao

Apache Openwhisk Architecture

Apache Openwhisk Architecture

ciao

Competing solutions

Competing solutions

ciao

Implementation

Implementation

ciao

References

References

- Etas: predictive scheduling of functions on worker nodes of apache openwhisk platform (Banaei, Ali and Sharifi, Mohsen), 2022
- Evaluating apache openwhisk-faas, Quevedo (Sebastián and Merchán, Freddy and Rivadeneira, Rafael and Dominguez, Federico X), 2019
- Crypto currencies prices tracking microservices using apache OpenWhisk (Huy, Lam Phuoc and Saifullah, Saifullah and Sahillioglu, Marcel and Baun, Christian), 2021
- Open-source serverless architectures: an evaluation of apache openwhisk (Djemame, Karim and Parker, Matthew and Datsev, Daniel), 2020
- LEARNING APACHE OPENWHISK: developing open serverless solutions (Sciabarrà, Michele), 2019