Apache Openwhisk

Distributed Software Systems, 2024/25

Davide De Rosa

1 Description of Apache Openwhisk

What it is: Briefly introduce Apache OpenWhisk as an open-source, serverless cloud platform designed to execute functions in response to events.
Core concept: Describe its Function-as-a-Service (FaaS) nature and event-driven approach.
Key features: Outline the system's capabilities, such as dynamic scaling, event-triggered function execution, and integration with other cloud services and APIs.

2 Context

Main use cases: List some typical scenarios where OpenWhisk is commonly applied:
Event-driven processing, such as responding to webhooks or changes in a database.
Real-time data processing, like stream processing from IoT devices.
Serverless backend for applications, where each API endpoint triggers a different function.
Context Diagram: Include a high-level context diagram showing interactions between OpenWhisk, event sources (like APIs, databases, or user actions), and other services.
Example scenario: Describe a scenario, e.g., a photo-sharing app where OpenWhisk triggers a function to generate thumbnails whenever a user uploads an image.
History and Invention: Explain that OpenWhisk was created by IBM to fill the need for scalable, event-driven computing on the cloud, with contributions from the open-source community leading to its incubation and adoption by the Apache Software Foundation.

3 Main Architectural Drivers

Scalability: Explain that OpenWhisk is built to handle variable loads, adjusting function instances as needed without requiring pre-allocated server resources.
Low Latency: Address OpenWhisk's design for quick response times, especially in real-time processing scenarios.
Fault Tolerance: Describe mechanisms for error handling and retry policies, which ensure robustness in the face of transient failures.
Vendor-Neutral: Highlight OpenWhisk's open-source nature, allowing it to run on multiple cloud platforms or on-premises infrastructure, making it vendor-neutral.
Flexibility in Integration: Describe how OpenWhisk's modular design allows it to connect with various services, enabling it to support diverse event sources and destinations.

4 Structure

Patterns: Discuss patterns OpenWhisk uses, like microservices and event-driven processing.
Components and Connectors:
Controller: Responsible for routing requests, managing APIs, and authentication.
Invoker: Executes functions in response to triggers, managing container lifecycles.
Messaging System: Connects event sources to functions.
Database: Stores metadata about functions, triggers, and actions (often uses CouchDB or similar).
Connector components: Enable integration with third-party services and external event sources.

5 Behavior

How OpenWhisk works:
Trigger-Action Model: Describe the core event-driven behavior, where functions (actions) are invoked by specific triggers.
Execution Flow: Explain the flow when an event occurs:
An event source triggers an action.
The Controller routes the request to an Invoker.
The Invoker pulls an image from a container registry and runs the function.
The system processes the output and sends it back to the client or next service.
Concurrency and Isolation: Describe how OpenWhisk manages concurrency using containers to isolate function executions.

6 Rationale

Why OpenWhisk has this architecture: Explain the design choices for the following reasons:
Containerized functions: Containers are lightweight and isolated, providing a scalable and secure way to handle function executions.
Event-driven model: Emphasizes resource efficiency and aligns with many real-time and sporadic-use cases common in serverless applications.
Microservices Architecture: Allows modular, flexible, and resilient system design, making it easier to add or modify individual components.

7 Similar or Competing Systems/Middleware

AWS Lambda: Compare OpenWhisk with Lambda, discussing differences like deployment options (AWS-specific vs. multi-cloud) and the ecosystem of integrated services.
Google Cloud Functions: Highlight differences in implementation, integration with Google's ecosystem, and scaling approaches.
Microsoft Azure Functions: Describe how Azure Functions differs in pricing, integration, and scaling compared to OpenWhisk.

Kubeless or Knative: Mention these Kubernetes-based serverless platforms as alternatives, especially for organizations invested in Kubernetes.

8 Simple Implementation Experiment

Experiment Idea: Outline a basic OpenWhisk implementation on a cloud provider (IBM Cloud or a local installation with Kubernetes). A simple experiment might include:
Setting up OpenWhisk and creating a basic function, like a "Hello World" HTTP endpoint.
Configuring a trigger, e.g., making the function respond to HTTP requests.
Deploying and testing the function to ensure it executes correctly.
Reporting Results: Document any performance measurements, such as response time and scalability observations, or any challenges faced during deployment.

References

[1] Author, A. (Year). *Title of the paper*. Journal Name, Volume(Issue), pages.