CloudEvents Practical Usage (CloudEvents-16)

Alicja Niewiadomska
Kinga Sąkól
Jakub Nowakowski
Michał Skorek

Środowiska Udostępniania Usług | AGH 2023

Introduction

CloudEvents [1] is an open standard for describing event data in a common way, allowing for interoperability between different cloud systems and services. As more and more organizations move towards cloud-based solutions, there is a growing need for standards that can help facilitate communication and data exchange between different systems.

In this project, we will explore a simple demo of CloudEvents. We will examine how to create and send CloudEvents, as well as how to receive and process them.

Theoretical background and technology stack

CloudEvents is based on the principles of event-driven architecture (EDA), which is a software architecture pattern that emphasizes the use of events to communicate and trigger actions within a system. In an event-driven architecture, events are first-class citizens, and the system is designed to be loosely coupled, distributed, and reactive. EDA is often used in modern, cloud-native architectures to enable agility, scalability, and resilience.

Events are messages that represent some occurrence or state change in the system, such as a user creating a new account, a sensor detecting a change in temperature, or a purchase being made. Events can be produced by any system or component within the architecture and can be consumed by any other system or component. In an event-driven architecture, systems communicate by sending and receiving events asynchronously, without waiting for a response.

CloudEvents specification defines a common format for representing event data and metadata, making it easier for developers to work with events across different systems and platforms. The specification is designed to be extensible, meaning that it can be adapted to work with different types of events and systems.

CloudEvents is a technology-agnostic specification, which means that it can work with a wide range of systems and platforms. However, it does require some underlying technologies to function properly. These technologies include:

1. Transport Protocols: CloudEvents supports various transport protocols, including HTTP, MQTT, and Kafka. These protocols are used to transmit events between systems.
2. Event Formats: CloudEvents supports different event formats, such as JSON and binary. These formats determine how event data is structured and encoded.
3. Event Brokers: Event brokers are responsible for receiving, storing, and delivering events. CloudEvents can be integrated with various event brokers, such as Apache Kafka, RabbitMQ, and AWS EventBridge.
4. Event Processors: Event processors are responsible for consuming and processing events. CloudEvents can be used with various event processors, such as Apache Flink, AWS Lambda, and Google Cloud Functions.
5. Event Management: Event management tools can be used to monitor, analyze, and manage events. CloudEvents can be integrated with various event management tools, such as Elasticsearch, Kibana, and Grafana.

Case study concept description

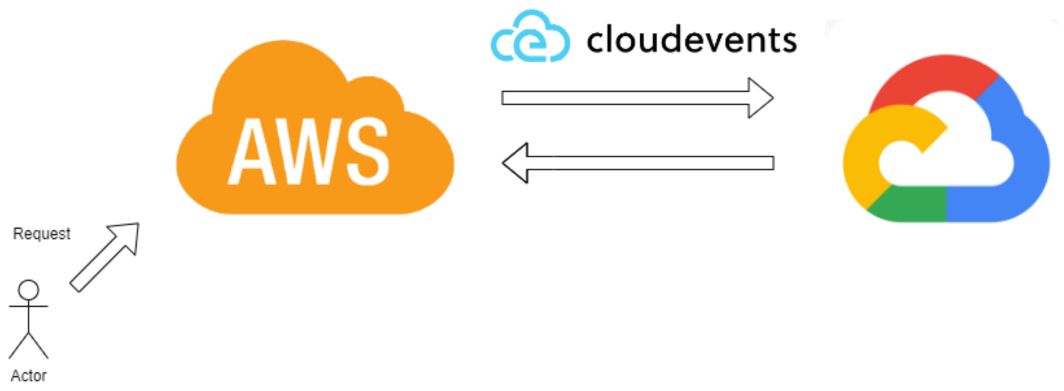

Diagram 1: high-level overview of the proposed demo

The objective of our demo will be to show how the use of cloud events can help integrate two independent cloud environments. We plan to realize an architecture based on two separate clouds, the first of which will be tasked with receiving files from the user and returning the results to them. We have decided to use AWS to realize this. The second will be Google Cloud, which will be responsible for processing the file sent by the user.

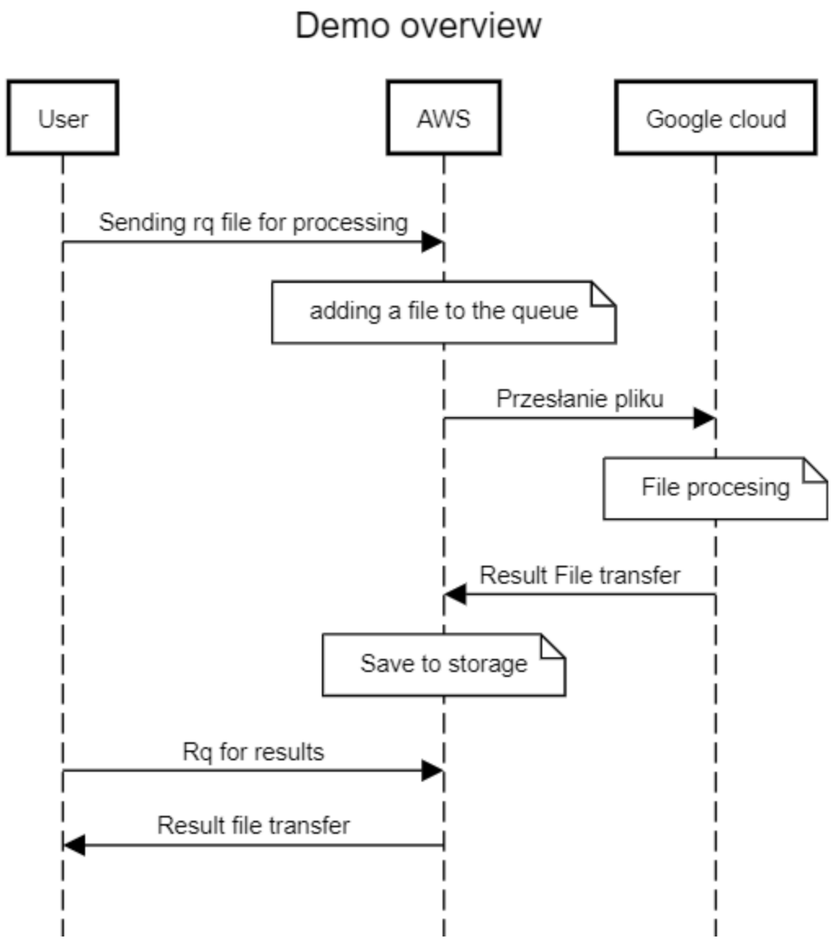

Diagram 2: flow diagram of proposed demo

Solution architecture

The system consists of two parts: on AWS (Amazon Web Services) and on GCP (Google Cloud Platform) which exchange information using CloudEvents.

AWS

For the part hosted on AWS, we will use following components:

- AWS Lambda function - triggered by the client
- S3 bucket - for storing processed files

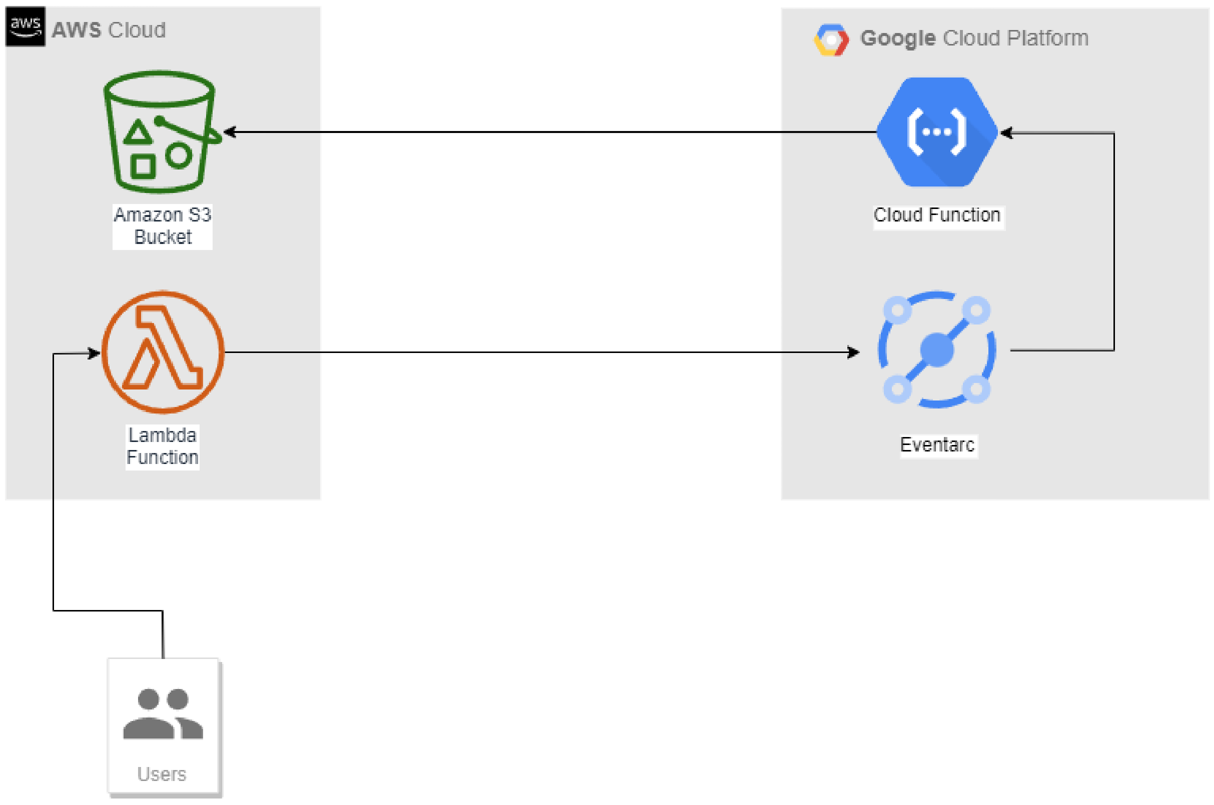

Diagram 3: solution architecture diagram

GCP

In Google Cloud platform, we will following components:

- Eventarc - will receive events from AWS on Pub/Sub topic
- Cloud Function - triggered when an event on a specific Eventarc topic is published. Function will process input and save on AWS

Environment configuration description

To present the demo, it will be necessary to configure two platforms - AWS and GCP. We have access to the first one through university resources. In the case of GCP, a free account has been set up where we have a certain amount of money at our disposal. It should suffice for presentation purposes.

Pub/Sub topic was selected as the transfer method with Eventarc. Cloud AWS will have an endpoint issued to which requests from the user can be directed. Communication will be from the user to the lambda function that will be provided by AWS. Then from the lambda function to Eventarc which will reside in GCP. Eventarc redirects events to Cloud Function, which will process the file and send it to S3 located on AWS.

Python was chosen as the SDK due to the simplicity of the language, ease of access to libraries and large community. Many issues have been covered in the forums, so the solution implementation should go smoothly.

Demo deployment steps

Configuration set-up

AWS

1. Clone repository: https://github.com/kingasakol/Cloud-Events-Processing
2. Create ECR repository [4]
3. Create S3 bucket [5]
4. Push image to ECR:
   a. Start docker daemon
   b. Run commands:

```
cd Cloud-Events-Processing
```

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS
--password-stdin <ECR REPOSITORY PATH>
```

```
docker build -t <ECR_REPOSITORY_PATH:TAG> -f aws/Dockerfile .
```

```
docker push <ECR_REPOSITORY_PATH_TAG>
```

5. Based on uploaded image create AWS lambda function [6]

GCP

1. Create new project on Google Cloud Platform and set it as current project
2. In search bar, find “Cloud Functions” and click on it
3. Click “Create function” button, then enable required APIs
4. Configure Cloud Function with “Environment: 2nd gen” option
5. Click “Add Eventarc trigger” and then enable other required APIs
6. Create new trigger with following configuration:

    - Trigger type: Google sources
    - Event provider: Cloud Pub/Sub
    - Event: google.cloud.pubsub.topic.v1.messagePublished
7. From the “Select a Cloud Pub/Sub topic” dropdown click “Create a topic” and create a new topic.
8. From “Service account" dropdown click “Create new service account” and create new service account
9. Go next on “Create function” page, enable other required APIs if popup shows
10. In the “Code” section you can copy code from the “gcp” folder in the cloned repository. Then click “Deploy”
11. In search bar find “Service accounts” section, then find service account in section 8, click on three dots icon and click “manage keys”
12. Click “Add key”, then “Create new key”, select JSON and finally “Create”. JSON key should start downloading
13. Import downloaded key to AWS

Execution procedure

In order to invoke the flow, run the following commands.

```
cd Cloud-Events-Processing
```

```
python run_client.py <path_to_file>
```

Results presentation

1. We can trace logs from lambda function invocation using CloudWatch.

```
START RequestId: 99863c3a-0b6f-439b-8089-10e63461b0b2 Version: $LATEST

END RequestId: 99863c3a-0b6f-439b-8089-10e63461b0b2

REPORT RequestId: 99863c3a-0b6f-439b-8089-10e63461b0b2     Duration:
610.71 ms   Billed Duration: 611 ms Memory Size: 128 MB     Max Memory
Used: 89 MB
```

Logs from CloudWatch

2. In the next step we can trace logs from GCP.

3. Finally we can see the processed image was uploaded to S3 bucket on AWS.

Summary – conclusions

In summary, the use of cloud events allowed us to achieve seamless communication between two different cloud architectures. This communication method can be considered modern and easy to use, serving as a foundation for building event-driven systems. The utilization of cloud events facilitates the proper sequencing of data processing regardless of the environment in which it takes place.

References

[1] https://cloudevents.io/
[2] https://aws.amazon.com/
[3] https://cloud.google.com/
[4] https://docs.aws.amazon.com/AmazonECR/latest/userguide/repository-create.html
[5] https://docs.aws.amazon.com/quickstarts/latest/s3backup/step-1-create-bucket.html
[6] https://docs.aws.amazon.com/lambda/latest/dg/images-create.html