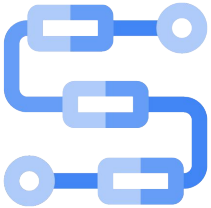
Choreography vs **Orchestration** in serverless microservices

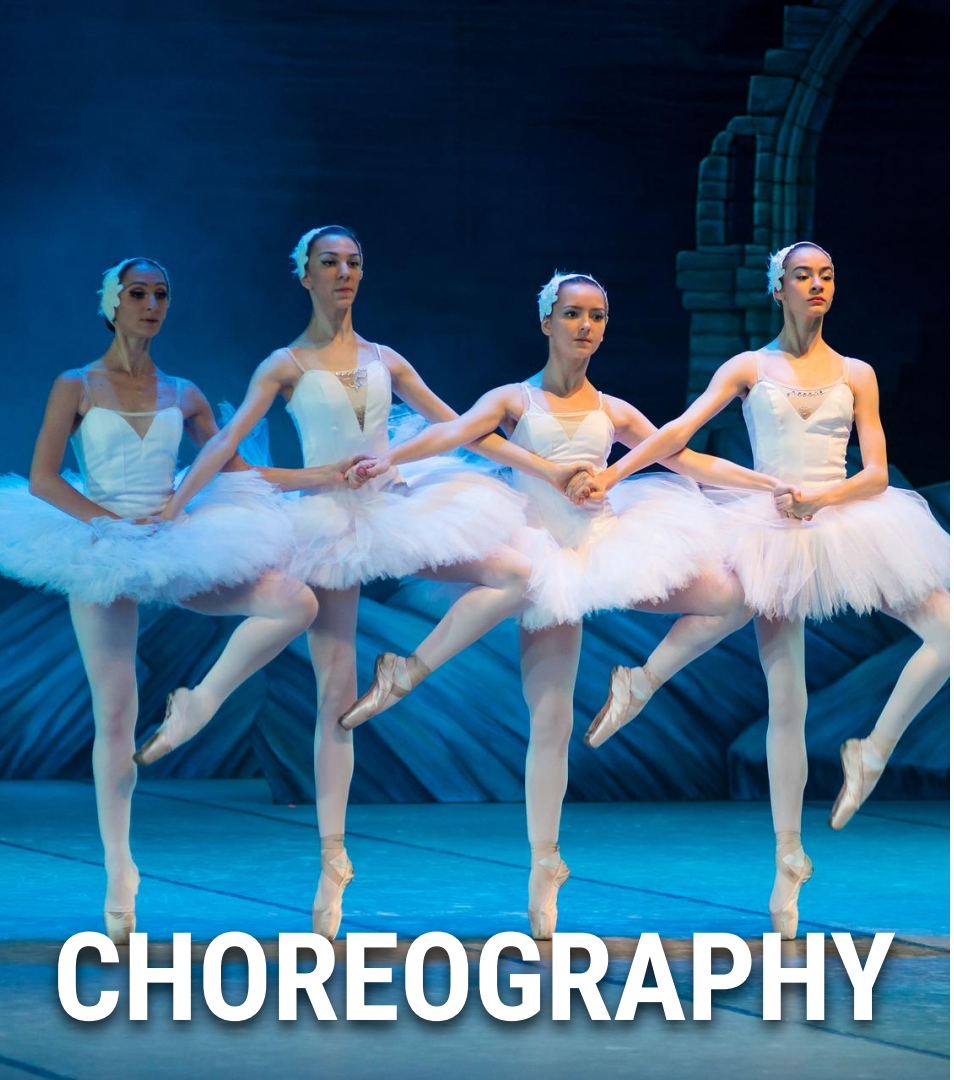
Mete Atamel
Developer Advocate at Google
@meteatamel
atamel.dev
speakerdeck.com/meteatamel

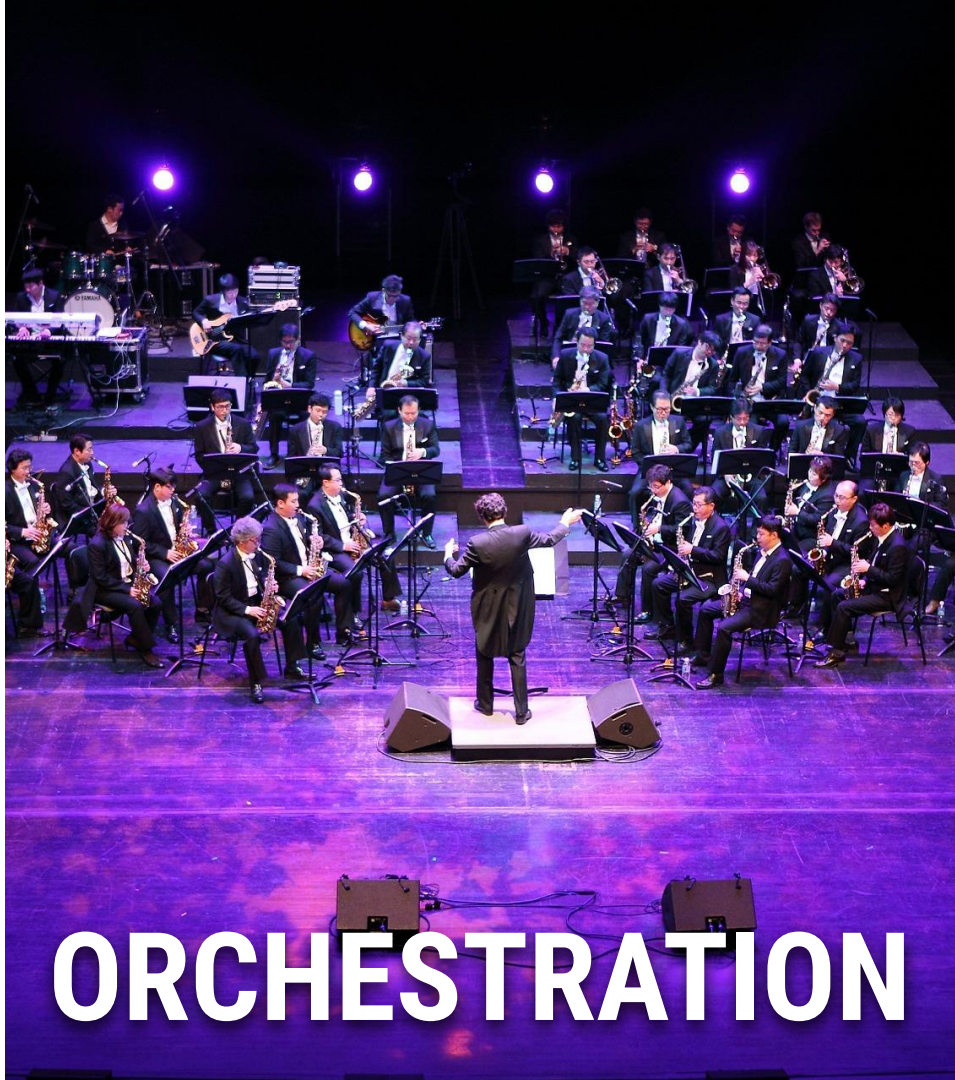
Guillaume Laforge
Developer Advocate at Google
@glaforge
glaforge.appspot.com



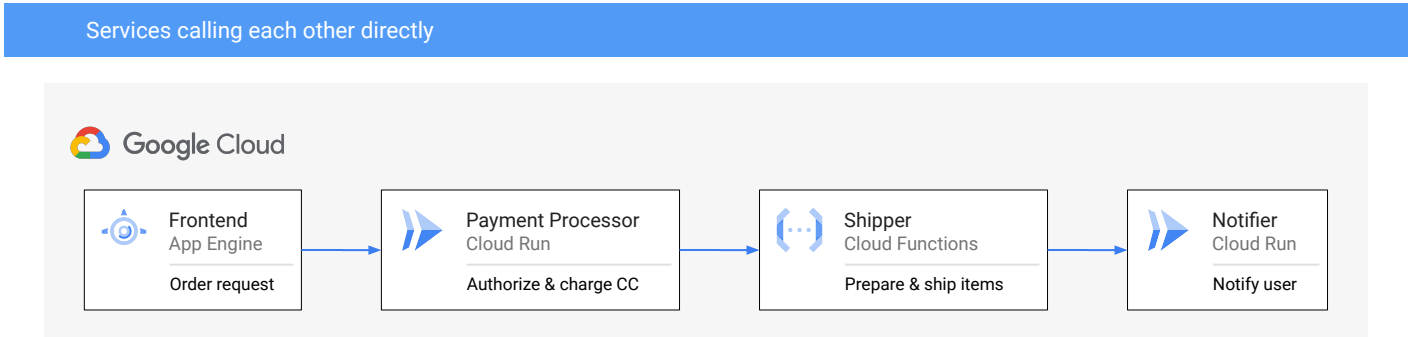
Choreography vs Orchestration

01


CHOREOGRAPHY


ORCHESTRATION

Imagine a simple e-commerce transaction



Simple REST: Pros and Cons

Pros

➕ Better than a single monolith

➕ Easy to implement: Services simply call each other

Cons

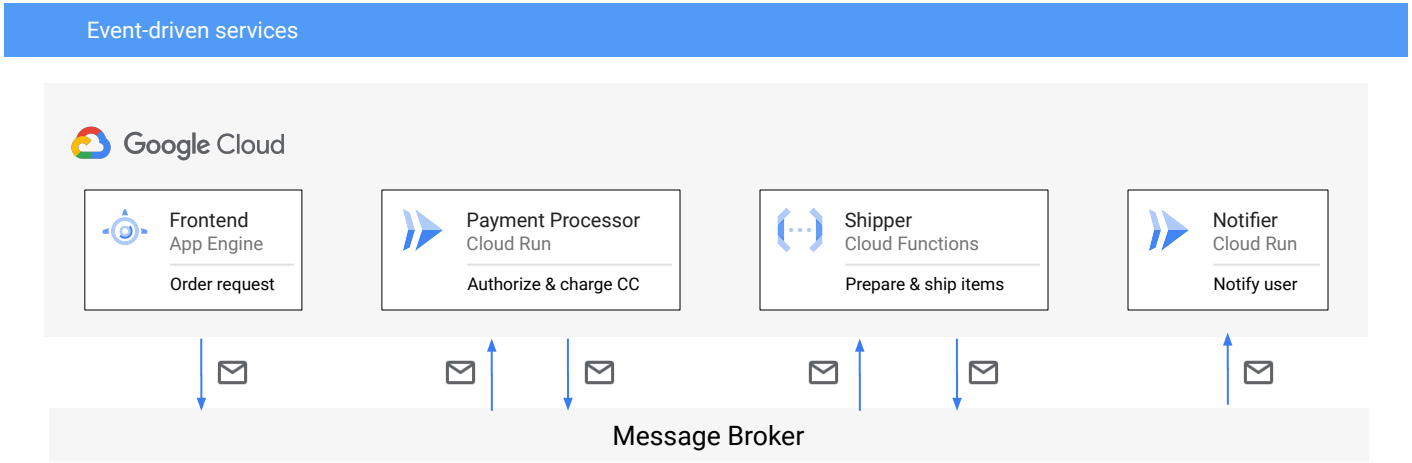
➖ Too much coupling between all the services

➖ Each service can be a SPOF

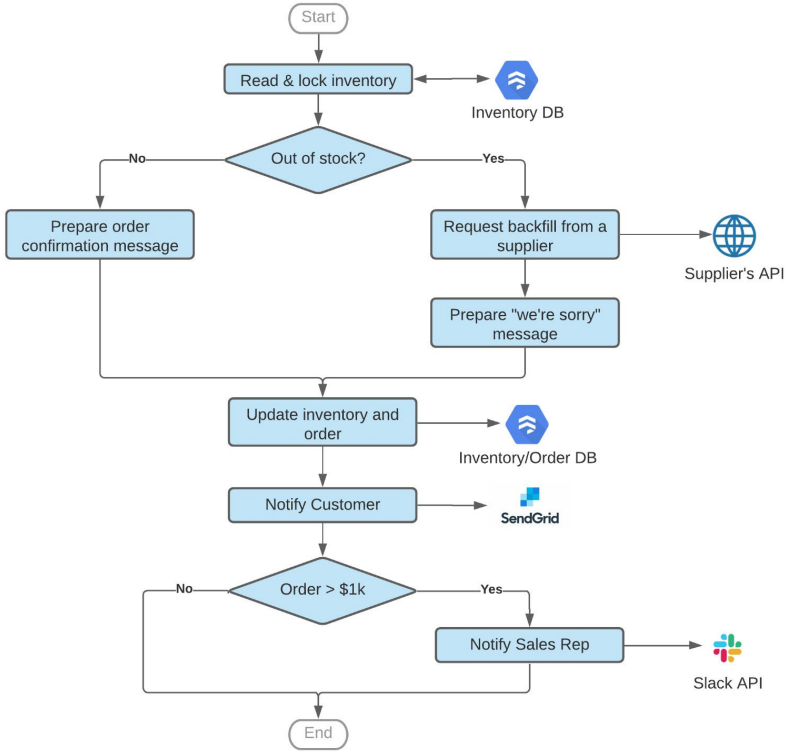
➖ Each service needs its own error / retry / timeout logic

➖ Who ensures the whole transaction is successful?

Choreography (event-driven)



Imagine a more complex transaction



Choreography: Pros and Cons

Pros

➕ Services are loosely coupled,

➕ Services can be changed independently

➕ Services can be scaled independently

➕ No single point of failure

➕ Events are useful to extend the system beyond the current domain

Cons

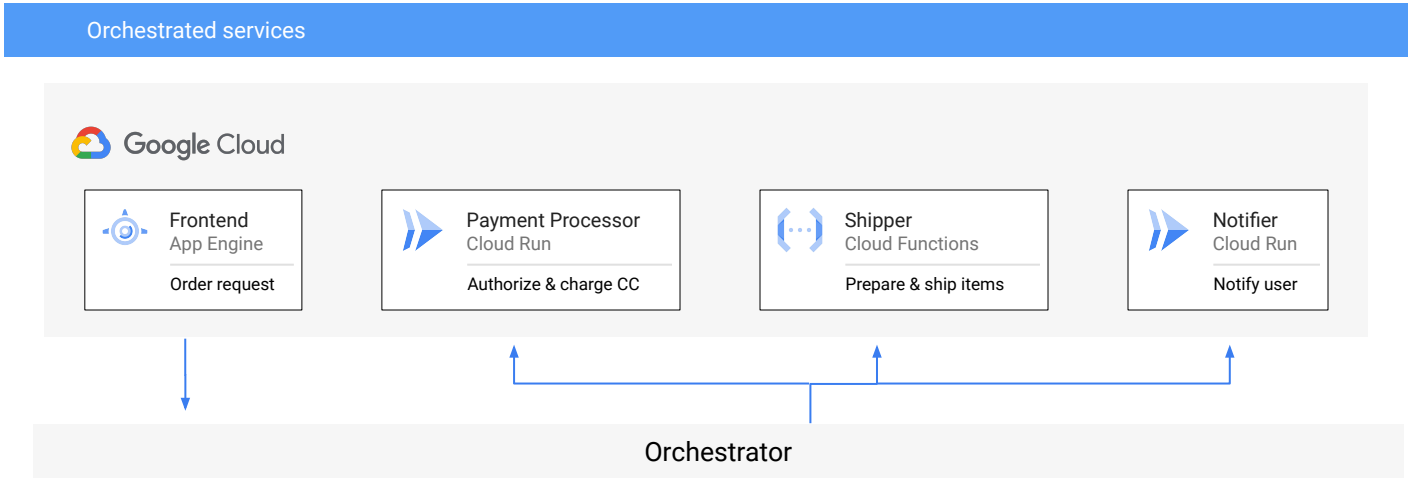
➖ Difficult to monitor the whole system

➖ Errors / retries / timeouts are problematic

➖ The business flow is not captured explicitly

➖ Who ensures the whole transaction is successful?

Orchestration



Orchestration: Pros and Cons

Pros

➕ Business flow captured centrally and source controlled

➕ Each step can be monitored

➕ Errors / retries / timeouts are centralized

➕ Use simple REST, no need for events

➕ Services are independent

Cons

➖ A new orchestrator service to worry about

➖ Orchestrator could be a single point of failure

➖ Reliance on REST means more tight-coupling

Which one is better?

Choreography or
Orchestration?

It depends...

Choreography

Services are not closely related

Services can exist in different bounded contexts

Multiple bounded contexts communicating via events

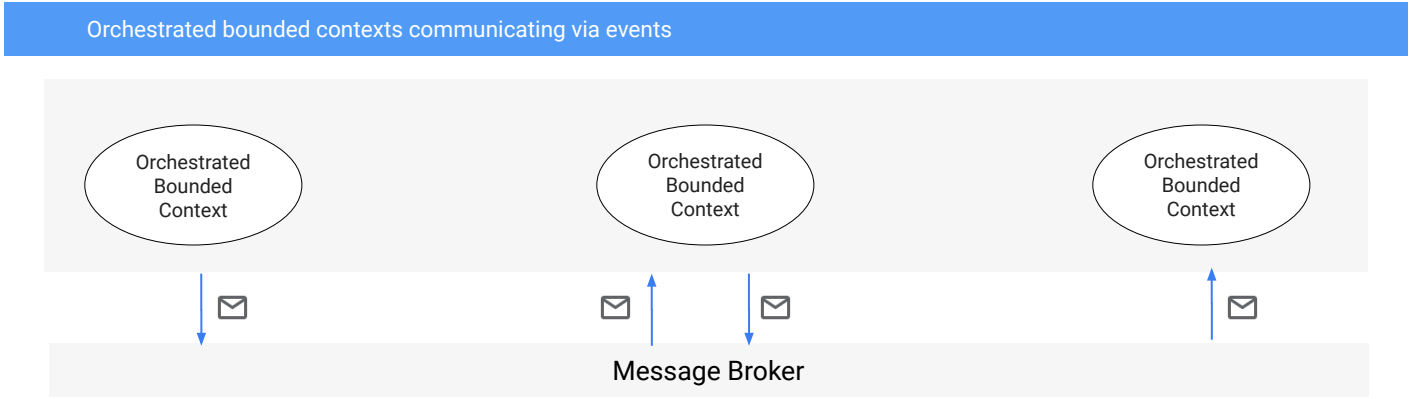
Orchestration

Can you describe the business logic in a flow chart?

Are services closely related in a bounded context?

Do you want to stay in REST?

Hybrid approach



Landscape

02

Google Cloud

Choreography (event-driven)

AWS: SQS, SNS, EventBridge

Azure: Event Grid, Event Hubs, Service Bus

Google Cloud: Pub/Sub, Eventarc

Other: Kafka, Pulsar, Solace PubSub+, RabbitMQ, NATS...

Orchestration

AWS: Step Functions

Azure: Logic Functions

Google Cloud: Workflows, Cloud Composer

Other: Apache Airflow

Serverless Workflow Specification

serverlessworkflow.io

A sandbox-level project at CNCF for a specification

Defines a declarative and domain-specific workflow language for orchestrating events and services

Serverless Workflow Specification

serverlessworkflow.io

Workflow projects need to implement & support the spec

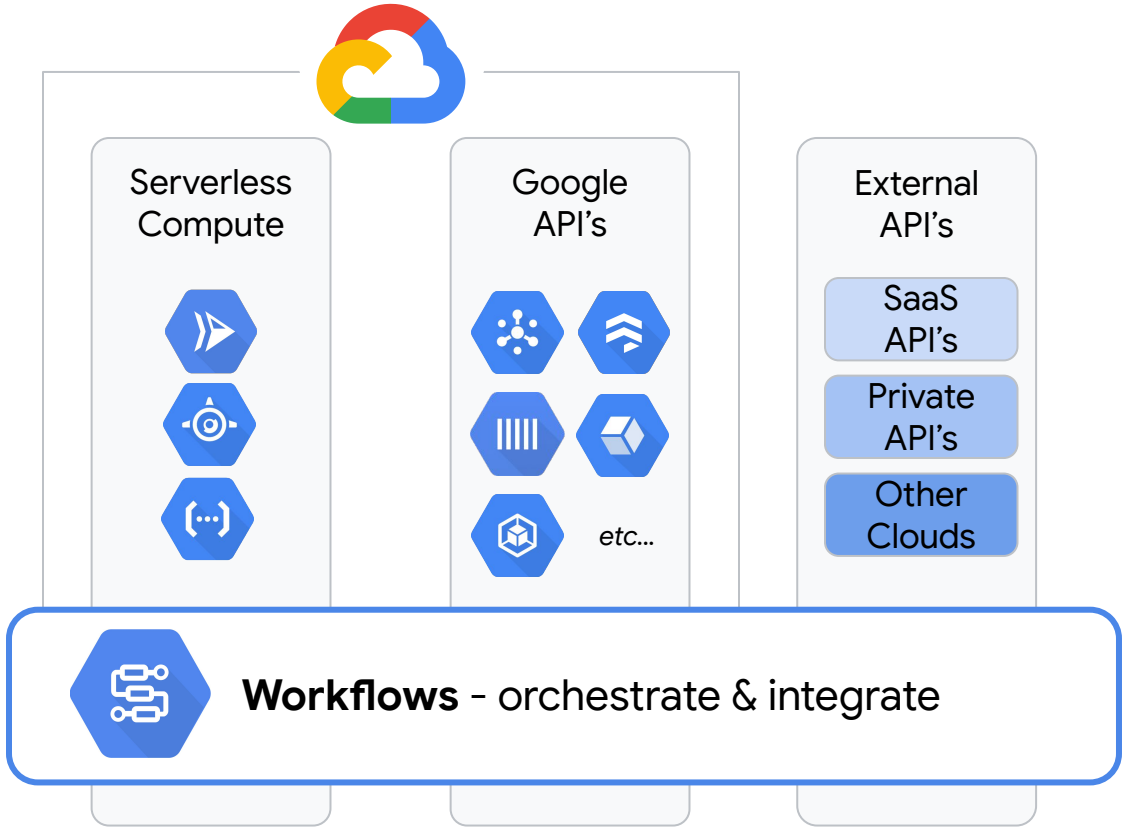
Spec doesn't necessarily cover all aspects of a product and not all products cover the whole specification

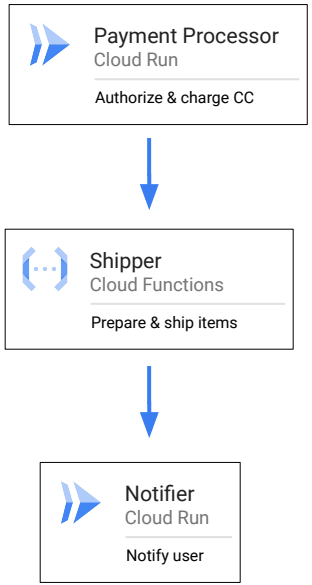
Services need to be described with OpenAPI

Orchestration:
Google Cloud Workflows

03

Google Cloud


Serverless Compute
Google API's
etc...
External API's
SaaS API's
Private API's
Other Clouds
Workflows - orchestrate & integrate

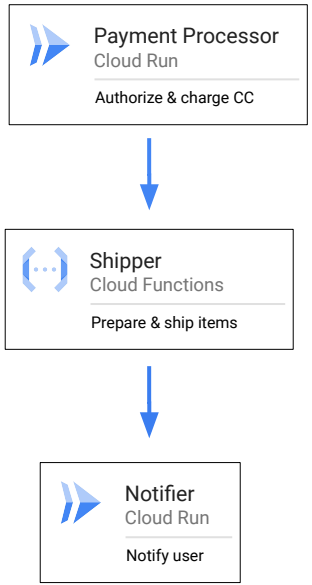


YAML or JSON syntax

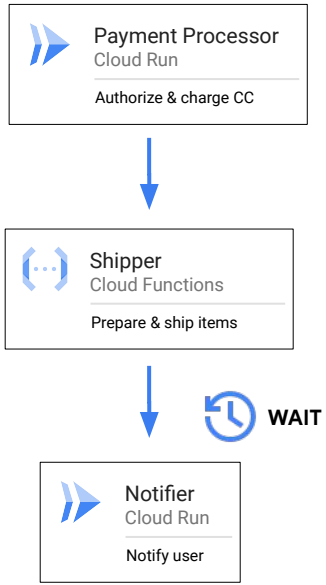
```
- processPayment:
    params: [paymentDetails]
    call: http.post
    args:
        url: https://payment-processor.run.app/...
        body:
          input: ${paymentDetails}
    result: processResult
- shipItems:
    call: http.post
    args:
        url: https://.../cloudfunctions.net/ship
        body:
            input: ${processResult.body}
    result: shipResult
- notifyUser:
    call: http.post
    ...
```

Steps

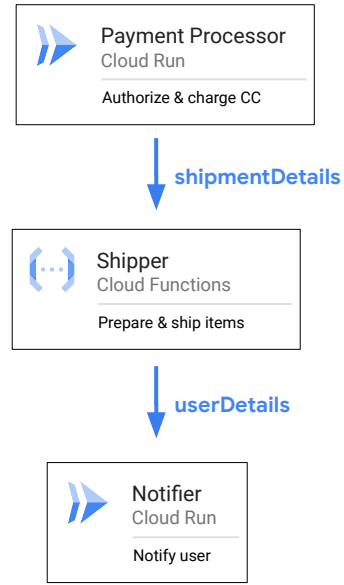
Step Sequencing



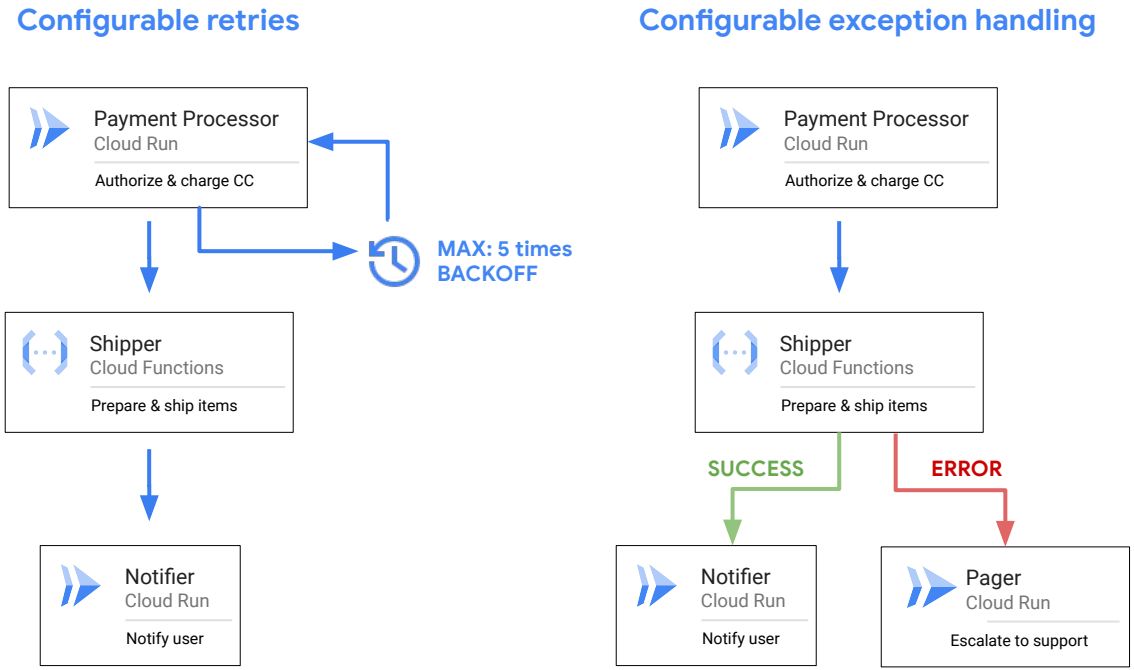
Serverless Pause



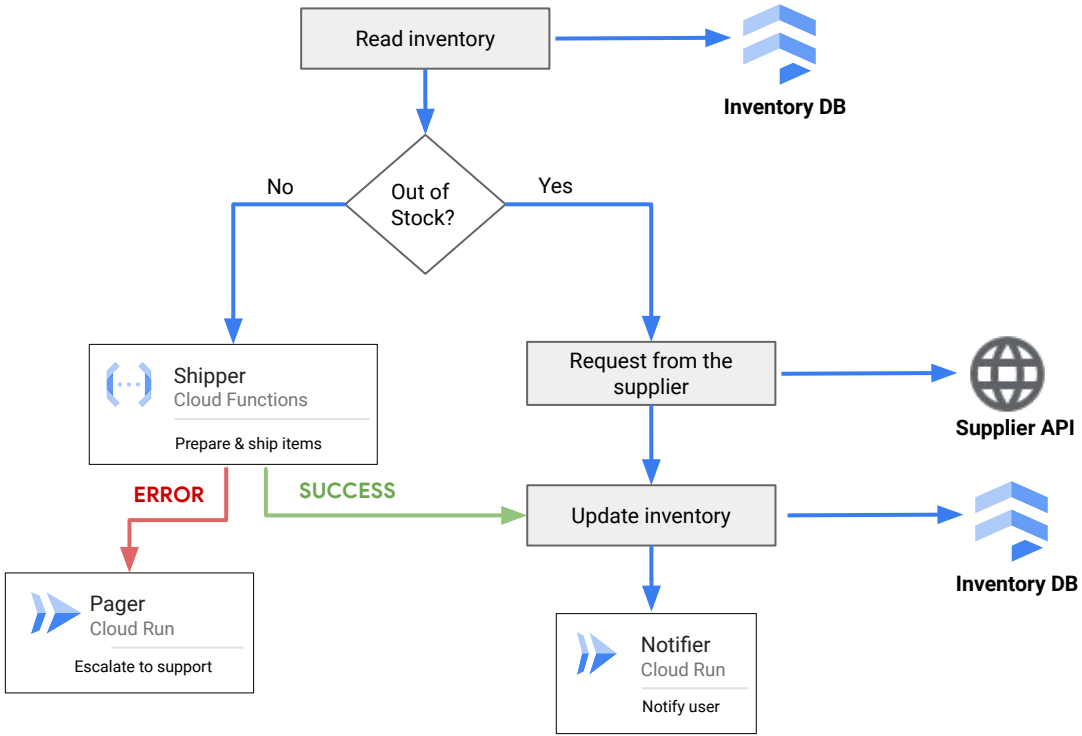
Variable passing
JSON Parsing



Errors and retries



Conditionals and 3rd party calls



Other useful features

Subworkflows

to encapsulate common reusable flows

Connectors ßeta

to connect to other Google Cloud APIs

Deploy, execute, manage workflows

```
# Deploy a workflow

gcloud beta workflows deploy my-workflow --source=workflow.yaml

# Execute a workflow

gcloud beta workflows execute my-workflow

# See the result

gcloud beta workflows executions
    describe <your-execution-id>
    --workflow my-workflow
```

Google Cloud Platform | workflows-atamel | Search products and resources

Workflows BETA | Edit Workflow | SYNTAX REFERENCE

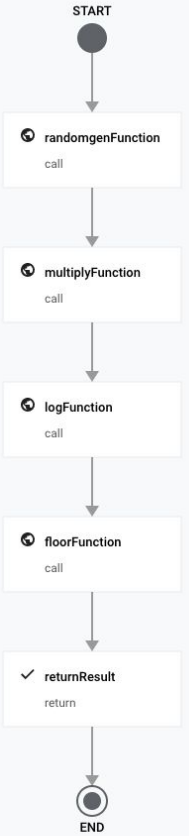
Configure — 2 **Define workflow**

Use Workflows syntax to define a workflow. Please refer to the syntax reference and sample workflows. DISMISS

```
# Copyright 2020 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
- randomgenFunction:
    call: http.get
    args:
        url: https://us-central1-workflows-atamel.cloudfunctions.net/randomgen
    result: randomgenResult
- multiplyFunction:
    call: http.post
    args:
        url: https://us-central1-workflows-atamel.cloudfunctions.net/multiply
        body:
            input: ${randomgenResult.body.random}
    result: multiplyResult
- logFunction:
    call: http.get
    args:
        url: https://api.mathjs.org/v4/
        query:
            expr: ${"log(" + string(multiplyResult.body.multiplied) + ")"}
    result: logResult
- floorFunction:
    call: http.post
    args:
        url: https://floor-wvdg6hhtla-ew.a.run.app
        auth:
            type: OIDC
        body:
            input: ${logResult.body}
    result: floorResult
- returnResult:
    return: ${floorResult}
```

Visualization

MAIN



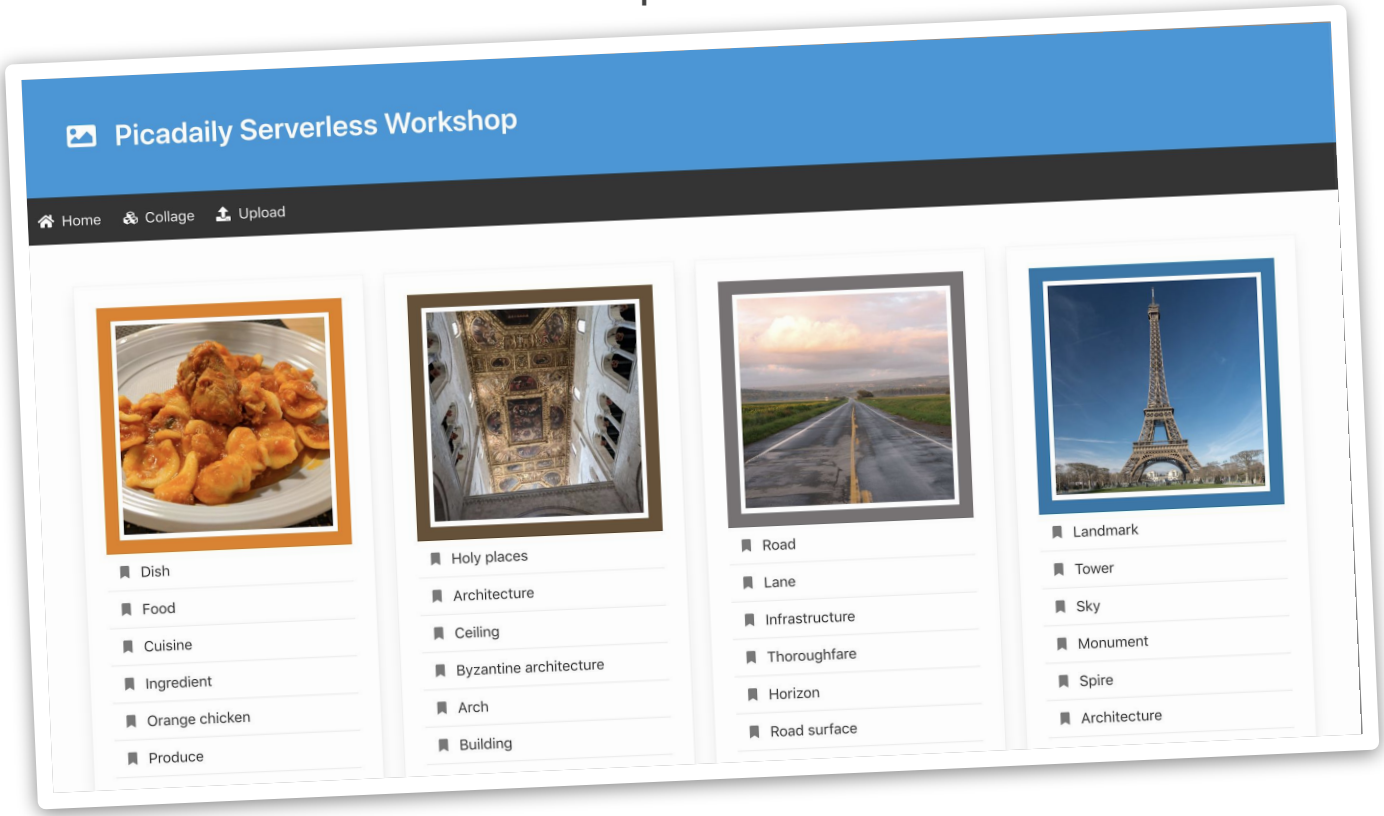
Case study: Pic-a-daily, A microservice-based picture sharing application

04

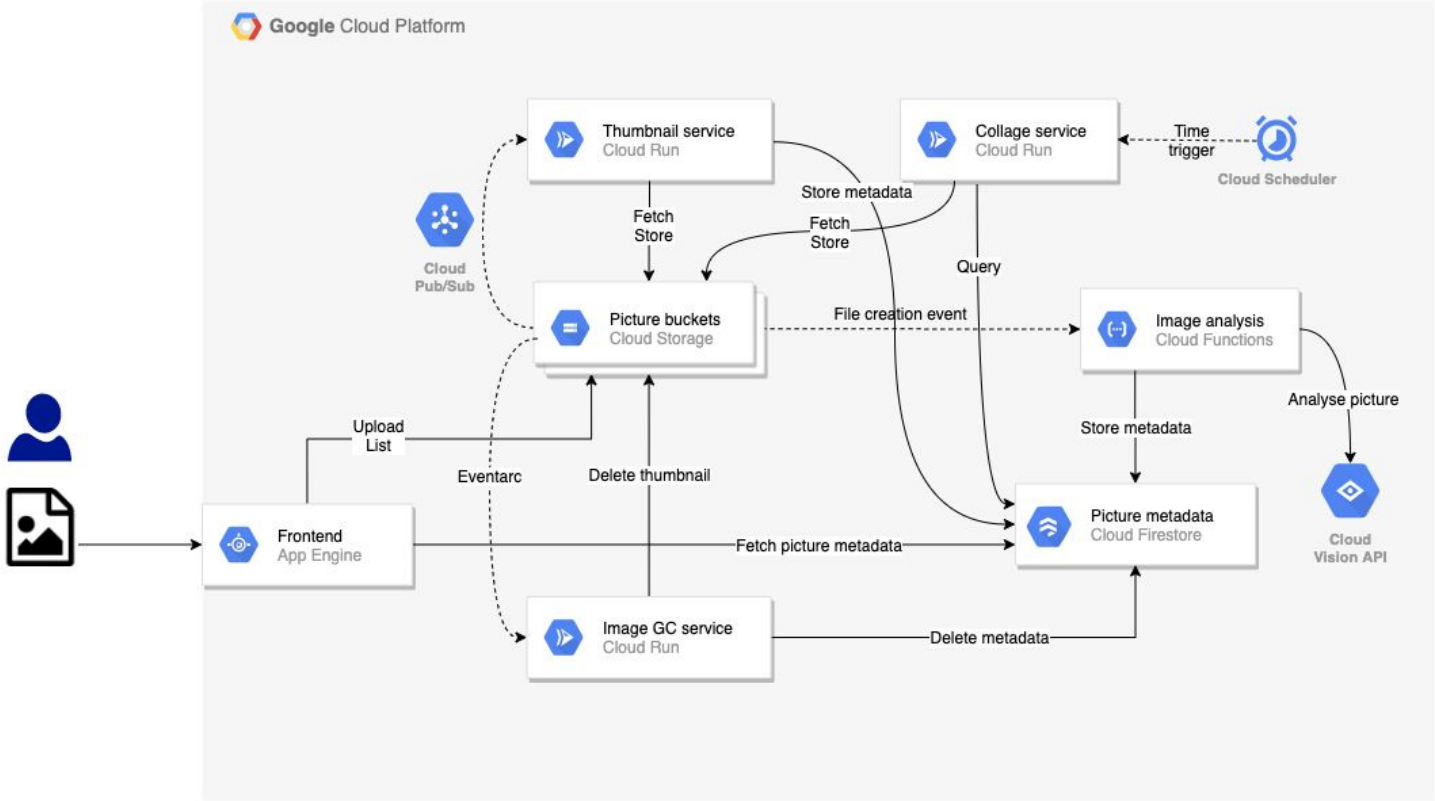
Google Cloud

Pic-a-daily: A photo sharing application

g.co/codelabs/serverless-workshop



Choreographed (event-driven) architecture



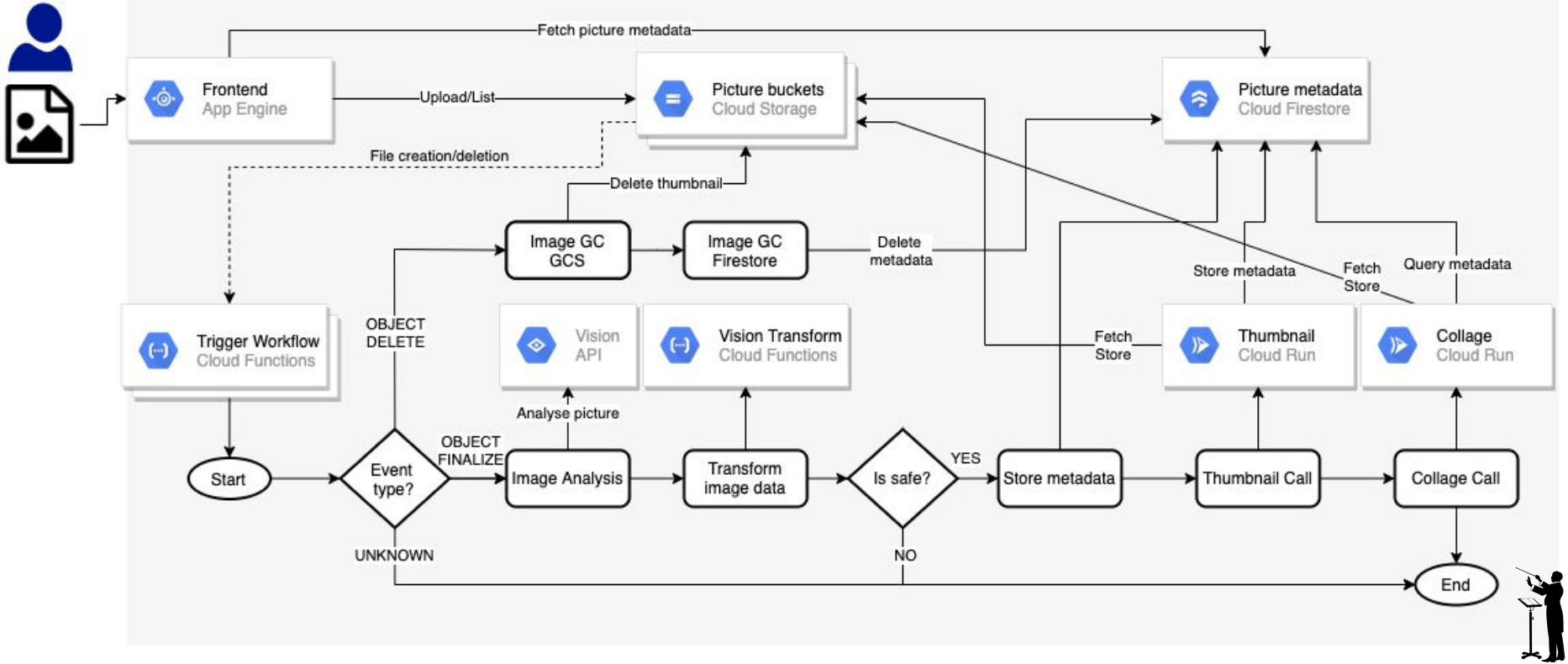
3 different event handling approaches

```
exports.vision_analysis = async (event, context) => {
    const filename = event.name;
    const filebucket = event.bucket;
```

```
app.post('/', async (req, res) => {
        const pubSubMessage = req.body;
        const eventType = pubSubMessage.message.attributes.eventType;
        const fileEvent = JSON.parse(Buffer.from(pubSubMessage.message.data, 'base64').toString().trim());
```

```
app.post('/', async (req, res) => {
        const cloudEvent = HTTP.toEvent({ headers: req.headers, body: req.body });
        const logEntryData = toLogEntryData(cloudEvent.data);
        const tokens = logEntryData.protoPayload.resourceName.split('/');
        const bucket = tokens[3];
        const objectName = tokens[5];
```

Orchestrated architecture



Lessons learned

Google Cloud

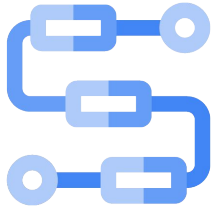
Lessons Learned

+ Simple REST was refreshing (vs. 3 eventing formats)
+ Less code (eg. no event parsing, no Image Analysis & Garbage Collector functions)
+ Less setup (eg. no Pub/Sub, no Scheduler, no Eventarc)
+ Easier error handling (eg. the whole chain stops on error)

Lessons Learned

- New service to learn with its quirks and limited docs
- Code vs. YAML, in a single YAML file
  (code is easier to write and test than YAML!)
- Debugging / testing / logging is not mature, no IDE support
- Lost parallelism
- Loss of eventing flexibility

Thank you



Cloud Workflows
cloud.google.com/workflows

Cloud Workflows tips
bit.ly/gcw-tips

Quickstart
cloud.google.com/workflows/docs/quickstarts

Codelab: Intro to serverless orchestration with Workflows
codelabs.developers.google.com/codelabs/cloud-workflows-intro

Mete Atamel
@meteatamel
atamel.dev
speakerdeck.com/meteatamel

Guillaume Laforge
@glaforge
glaforge.appspot.com