Serverless computing

Mathieu Bacou
mathieu.bacou@telecom-sudparis.eu

2022 – 2023

CSC5004 — CLOUD COMPUTING ARCHITECTURES

Actually, containers are hard

- An environment is required
  - Overhead of building containers and pods
- A management layer is required
  - Overhead of configuring service availability
- Backends are required
  - Overhead of management of non-core features
    - Database servers, monitoring…
  - Always running servers
    - Can scale down to 0, but then latency overhead on next request

Introducing: serverless

- Real cloud-native applications: only provide code for the business core features
- All management and execution provided by the cloud platform
  - From execution environment to service availability

Serverless {

Function-as-a-Service

+

Backend-as-a-Service

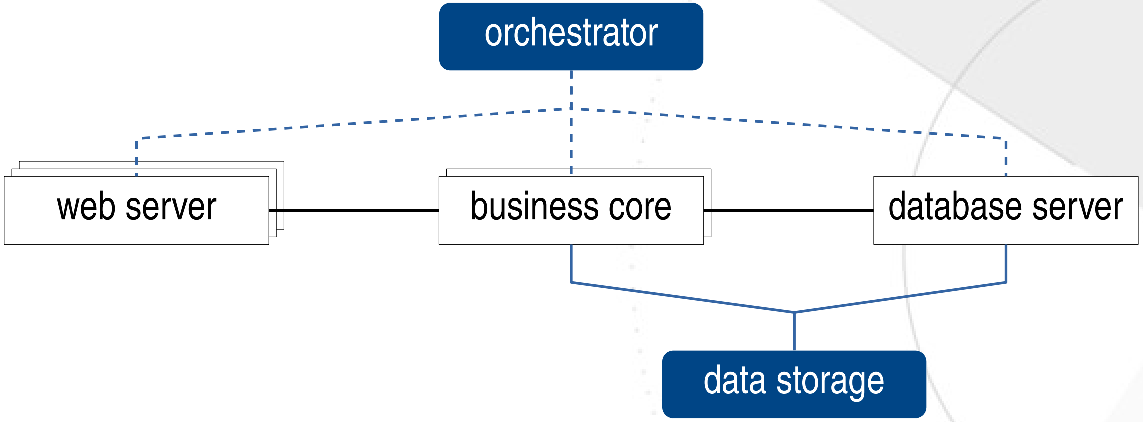
Backend-as-a-Service

- Common backend components in application architectures
  - Database servers, message queues, (object) storage…
- Better served by the cloud provider
  - Mutualized, no overhead for the user, available
  - Provides an ecosystem of components
    - Beware vendor lock-in!
- Elasticity requirement: scale quickly, up and down to zero, with the FaaS workload

Function-as-a-Service

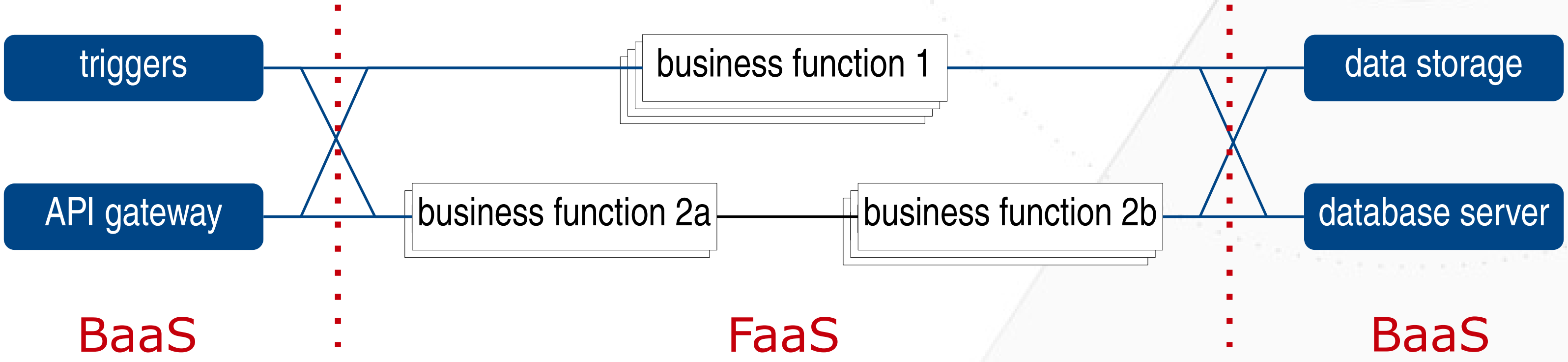
- Run backend code without long-lived servers
  - Execution environments are spawned on-demand
  - All managed by the cloud platform
- The unit of execution is a code block: the function
  - Applications are mostly event-driven
  - Parallelism at the cloud function level
  - Technically, also concurrency inside the cloud function
- Central feature of serverless

Comparison with micro-services



Micro-services architecture in the cloud

■ Provided by cloud platform



Serverless architecture in the cloud

Benefits of FaaS

- Elasticity: granularity of the request handler
  - Quick scaling, down to zero
- Deployment: just write code and upload
  - Quick experimentation, update
- Cost: pay only the compute time you need
  - No request = no function running = no resource = no cost
  - Roughly: Cost = Compute Time x Reserved Memory

Demo: Apache OpenWhisk

- Create new function
- Manually invoke function
- Use API gateway
- Use triggers
- Warm and cold starts

APACHE
OpenWhisk™

FaaS application architecture

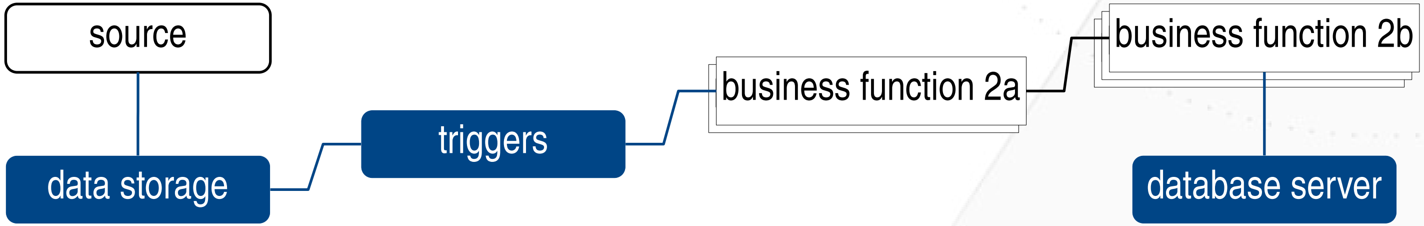
- Extract-Transform-Load (ETL) paradigm: get data, process data, output data

Extract Transform Load

  - Event-driven
    - Execution when a request arrives or a trigger is fired
  - Stateless functions: no side-effects
    - Use BaaS services to store business data
    - Rely on API gateway or client to keep request state

Side-effect: modifying global state

In the cloud: modifying the DB, touching storage... while processing data



Example of the Extract-Transform-Load paradigm in Serverless

FaaS application architecture

- Extract-Transform-Load (ETL) paradigm: get data, process data, output data

Extract

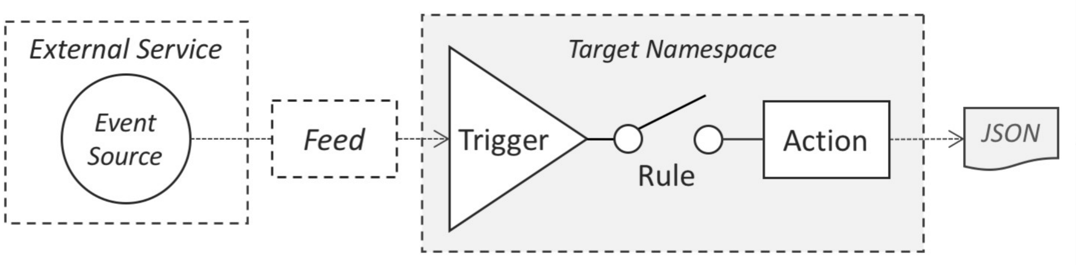
Transform

Load

  - Event-driven
    - Execution when a request arrives or a trigger is fired
  - Stateless functions: no side-effects
    - Use BaaS services to store business data
    - Rely on API gateway or client to keep request state

Side-effect: modifying global state

In the cloud: modifying the DB, touching storage... while processing data

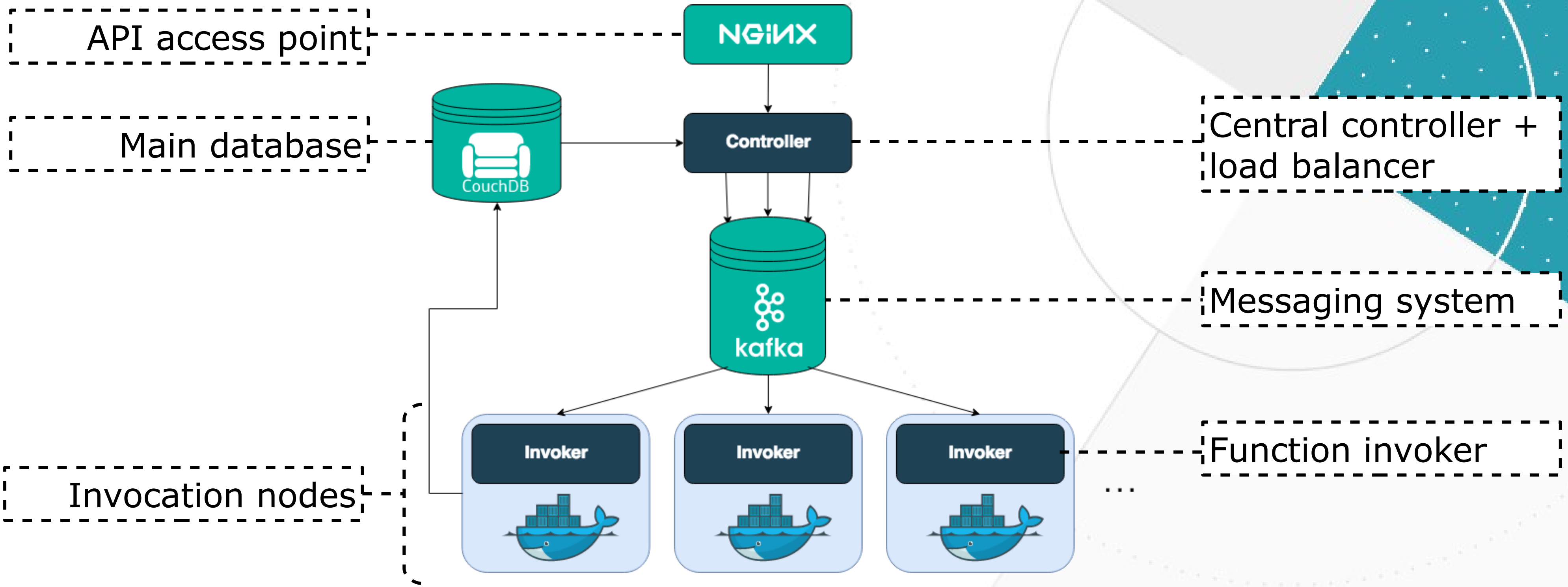


Programming model of Apache OpenWhisk

FaaS application architecture

- Platform-level parallelism
  - No need for multi-process management
- Chains and graphs of function dependencies
  - Chain function calls to implement more complex features while keeping fine granularity
  - Higher-level chaining: dependency graph (MapReduce…)
- In practice: collection of code pieces + platform-level configuration
  - Provided environments: NodeJS, Python, Java
  - Custom environments (Docker images)
  - Opaque binary executables

Internals of Apache OpenWhisk



API access point

Main database

Invocation nodes

Central controller + load balancer

Messaging system

Function invoker

Architecture of Apache OpenWhisk

Internals: function invocation

0)(after authentication and other tasks)

1)Spawn new Docker container with runtime

2)Inject action code

3)Execute action with parameters

4)Retrieve result

5)Destroy Docker container

OPTIMIZED

Function container management

- Very slow (for serving request) to spin up new container
  - Around 400ms
- Reuse existing containers!
  - Functions are stateless
- Cold starts and warm starts
  - No runtime container available: cold start
  - Available runtime container: warm start

40 times faster!

- Smart management of container pool
  - Pool of pre-warmed containers
  - Trade-off between occupied resources and execution latency
    - Containers kept warm use resources but are not billed to the user!

Limits of serverless

- Latency: cold starts
- Compatibility with serverful applications
  - What about stateful applications? (no local state)
  - What about massively parallel applications?
    - Isolation between functions: MPI is hard
  - FaaS is not fit for long-running computations
    - Will cost more while being less efficient
- Fresh, active area of research!

Serverless computing

- Function-as-a-Service for core business code and features
- Backend-as-a-Service to provide architectural services
- Most cloud-native paradigm
  - Fine-grained, elastic, pay-as-you-go
- Not suited to all applications
  - Yet?