Cloud Models

Motivation

- Main Models
  - IaaS (basic)
  - SaaS (motivation)
  - PaaS (utility)
  - Serverless/FaaS

IaaS

- Foundational
  - Makes *utility computing* possible
    - Pay-as-you-go
      - □ Just like power/water/gas: utilities
- Fully depends on virtualization techniques
  - Versatility in resource management
- Fully depends on providers
  - On premise
    - Own CPD
    - Tailored CPD
  - Off-premise
  - Hybrid schemes
    - On-premise-then-off-premise

SaaS

- SaaS <u>KEEPS STATE </u>through its lifetime
  - This is a service
  - The state is the history of event executions
    - It is a sort of DataBase
    - Each event is like a transaction that atomically modifies the state of the service.
- Thus
  - SaaS requires 24/7 activation
    - Supposedly is always active
    - Can be relaxed
  - And complex scaling logic
    - To meet QoS parameters
  - Same request at different times…
    - …may produce different results

SaaS: is it always needed?

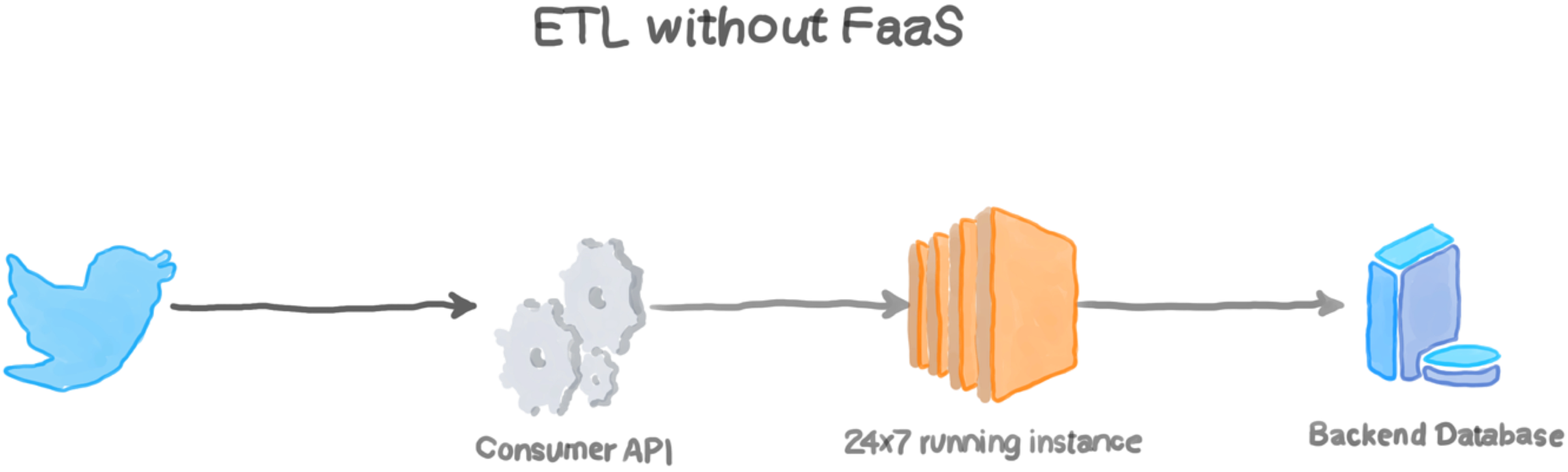
- Are there loads that do not require to store state?
  - I.e., purely functional
    - Fire and forget
- If so, do they require 24/7 activation?
  - On demand only
    - Called like a **F**unction (**a**s **a** **S**ervice, of course)
      - No scaling issues, per se
        - Scaling is automatic
      - No state
        - Indeed we DO NOT WANT ANY STATE
          - Danger to break isolation
      - But possible latency issues when activating function

Serverless

- Aka FaaS: Function as a Service
  - Logic are small pieces of code
  - Respond to events
    - Event-driven
    - Functional
    - Logical activation when an even occurs
      - □ E.g. availability of a message on a message queue
      - □ Variety of triggers
- No local state store
  - But may use external persistent stores
    - i.e. SaaS (which keep state)
- Main advantage: ELASTIC BY DESIGN
  - If latencies are manageable

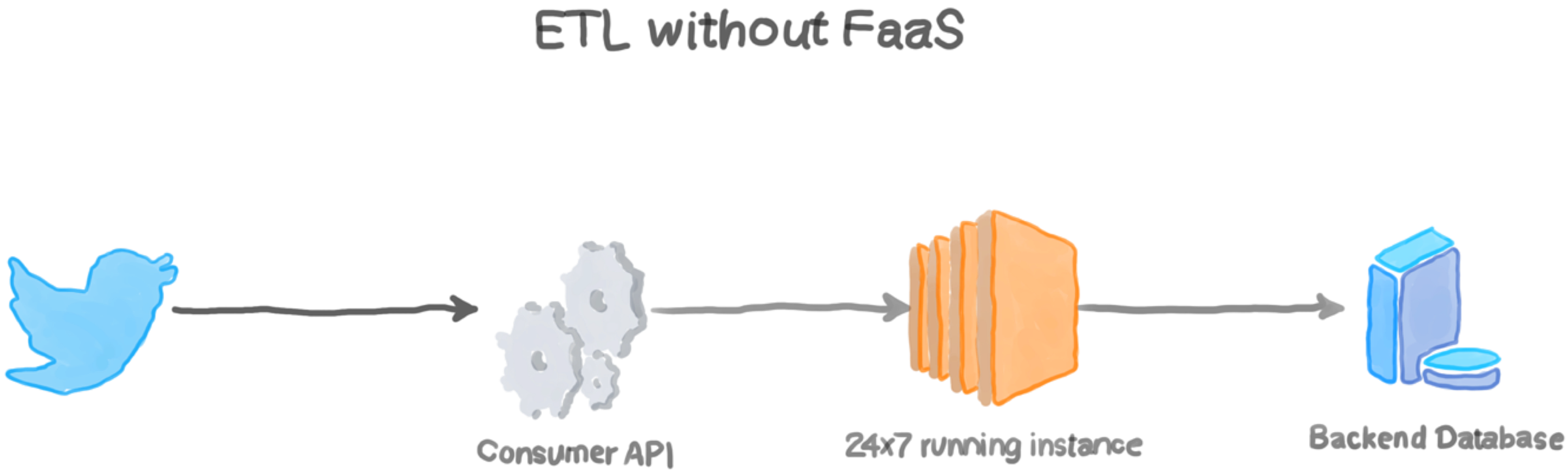
Load Example: ETL

- Extract, Transform, and Load
  - Basic process in Data Analysis tasks
  - When data arrives, it is conditioned/preprocessed
    - Then saved to a database



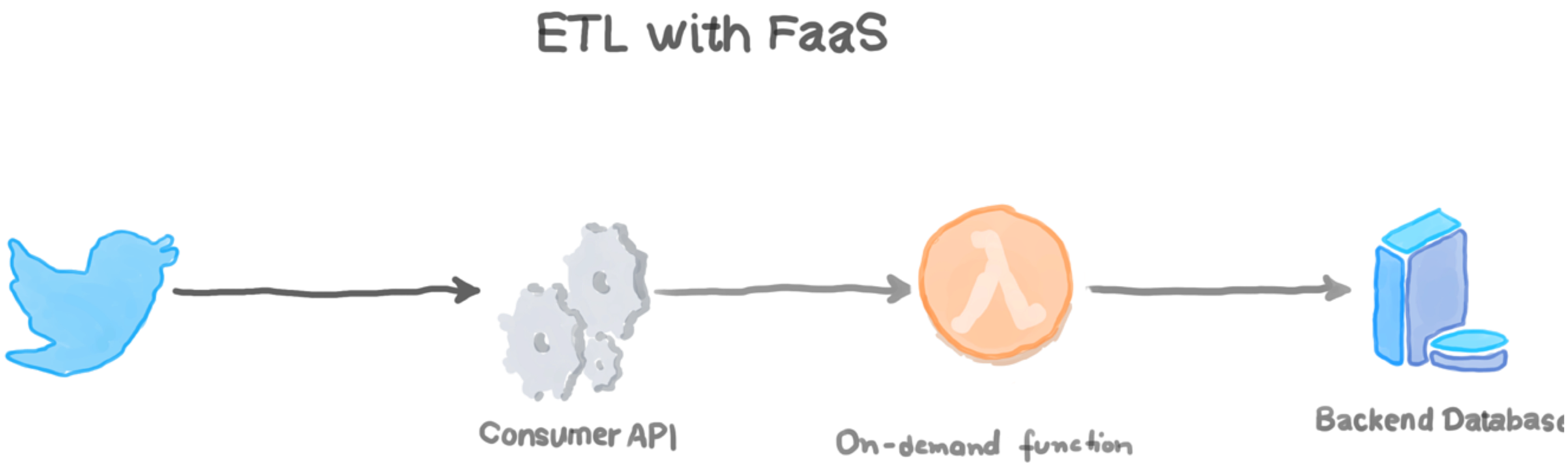
Load Example: ETL, with SaaS

- Allocate an instance for 24/7 operation
  - Needed for when data arrives
  - Constant cost
  - Need to scale if lots of data arriving
- What happens if Data arrival is irregular?
  - Instance idle often
  - If suddenly lots of data arrive, potential scaling issues



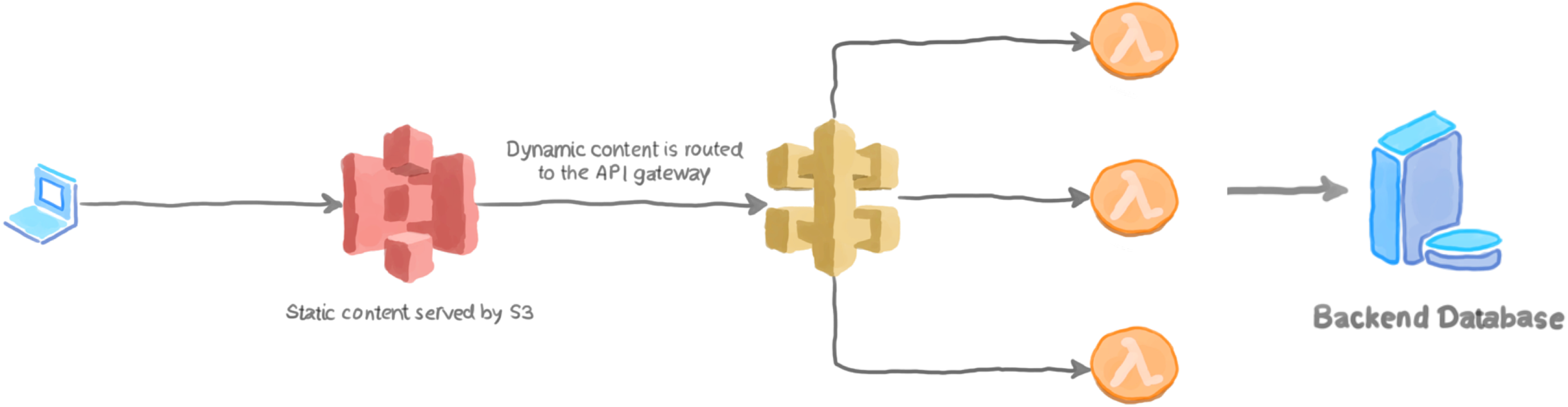
Load Example: ETL, with FaaS

- What happens if data arrives at irregular intervals?
  - Functions are activated only when there is data
    - Costs only when there is something to actually do
- If lots of data arrive suddenly
  - Multiple activations can be carried at the same time
  - Scaling is a non-issue for the function implementation
    - It impacts the FaaS implementation
    - It all depends on the framework



Serverless Programming and SPAs

- Much of the application function carried out within an SPA
  - Static code downloaded from a site
    - E.g., S3/Github/Gitlab,…
- Interact with a back-end to perform some function
  - On demand, at irregular times
  - State maintained possible on a database of some sort (if at all)


Dynamic content is routed to the API gateway
Static content served by S3
λ
λ
λ
Backend Database

Serverless

- Requires a framework
  - Configured to look for relevant events
    - And activate the code when an event arrives
  - Often provides a runtime for specific languages
    - Exceptions exist, and the pattern itself does not require it
- The framework itself must be stateful
  - Store the evnts
  - Keep track of the activations
  - Keep track of on who’s behalf an activation is proceeding
  - Keep track of where to find the functions
- Framework is a SaaS
- Backend store is often necessary
  - Another SaaS

PaaS: Platform as a Service

- Personas:
  - Developers
  - Integrators
  - Service managers
- Make life easy for all of them
  - Developers
    - Focus on the app, forget systems-related noise
  - Integrators
    - Use a high level spec to convey structure of the microservices
  - Service managers
    - Let the platform automate most of the life cycle management

PaaS

- Automations:
  - Fault-healing/high availability
  - Scaling (horizontal/vertical)
  - Disaster recovery
  - Upgrade paths
  - Security
- Approach
  - Constrain how service applications are expressed
    - Give the platform extra knowledge to automate

PaaS

- Benefits:
  - Developers do not have to think about system
  - Underlying OS is a non-issue
  - Automations to adapt to changes in demand volume/types
- Drawbacks
  - No control on underlying technology
  - Must fit the framework
    - Opinionated way of structuring things…
    - Specific API for managing aspects of the app
  - May be locked-in in some cases
    - Open source alternatives diminish this risk