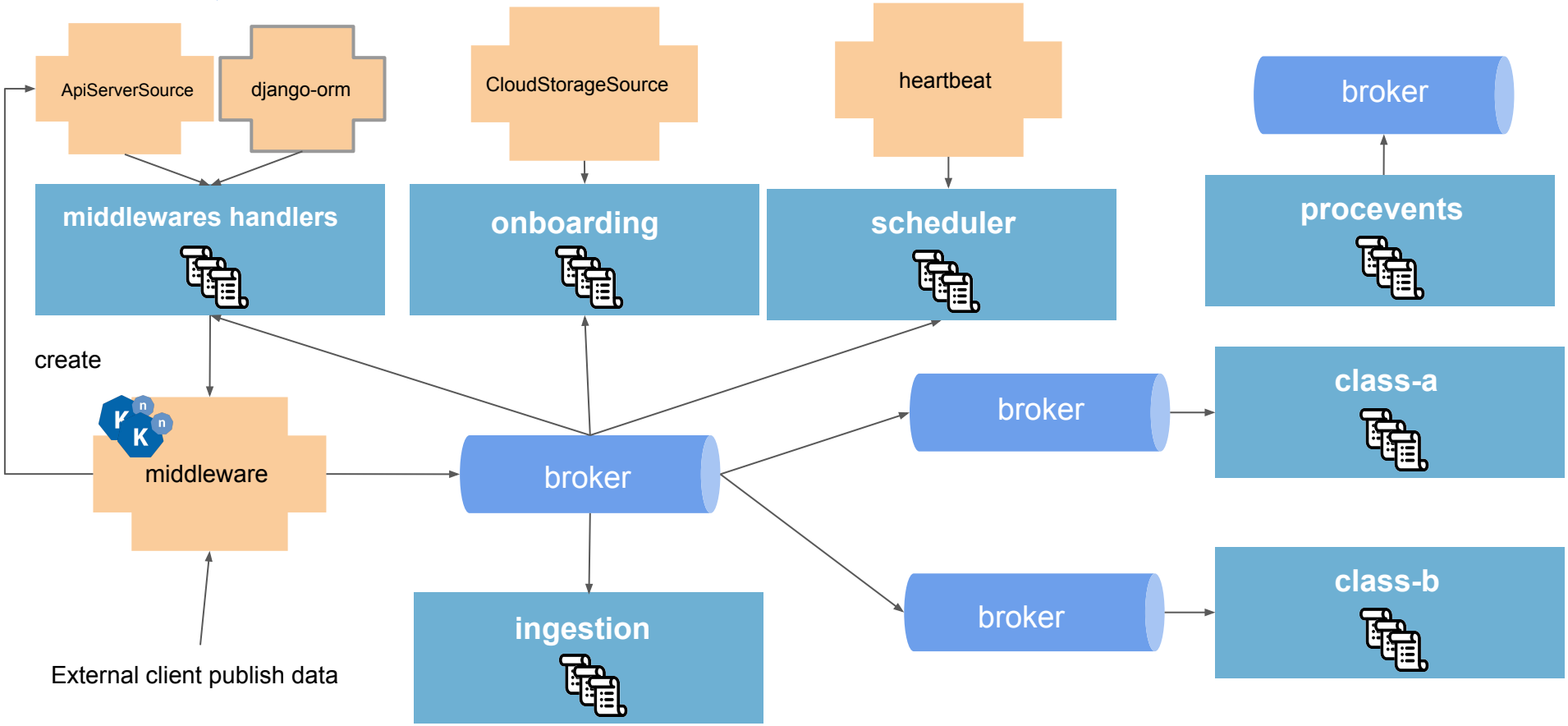
IoT Demo

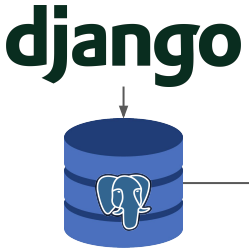
Airspot
cloud native development

sources brokers and ruleset groups

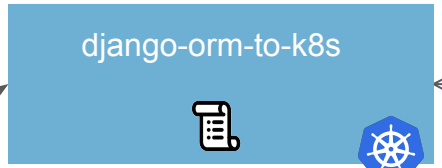


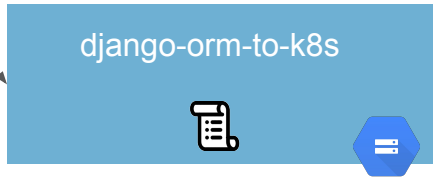
Airspot
cloud native development

Middleware: model creation



broker **default**

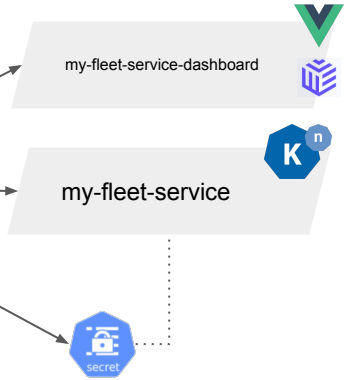




Attributes,
  type: **django.orm.post_save**
  subject: **DCE:device_manager.fleet/my-fleet-service**
Extensions,
  djangoapp: device_manager
  djangomodel: fleet
Data,
 {
   "data": {
     "name": "**my-fleet-service**",
     "apikey": "**d069ee84-4636-4193-9de2-53247333091**",
     ...
   },
   "signal_kwargs": {
     "created": **true/false**,
     ...
   },
 }

On Fleet creation create related folder base structure on GCS Bucket

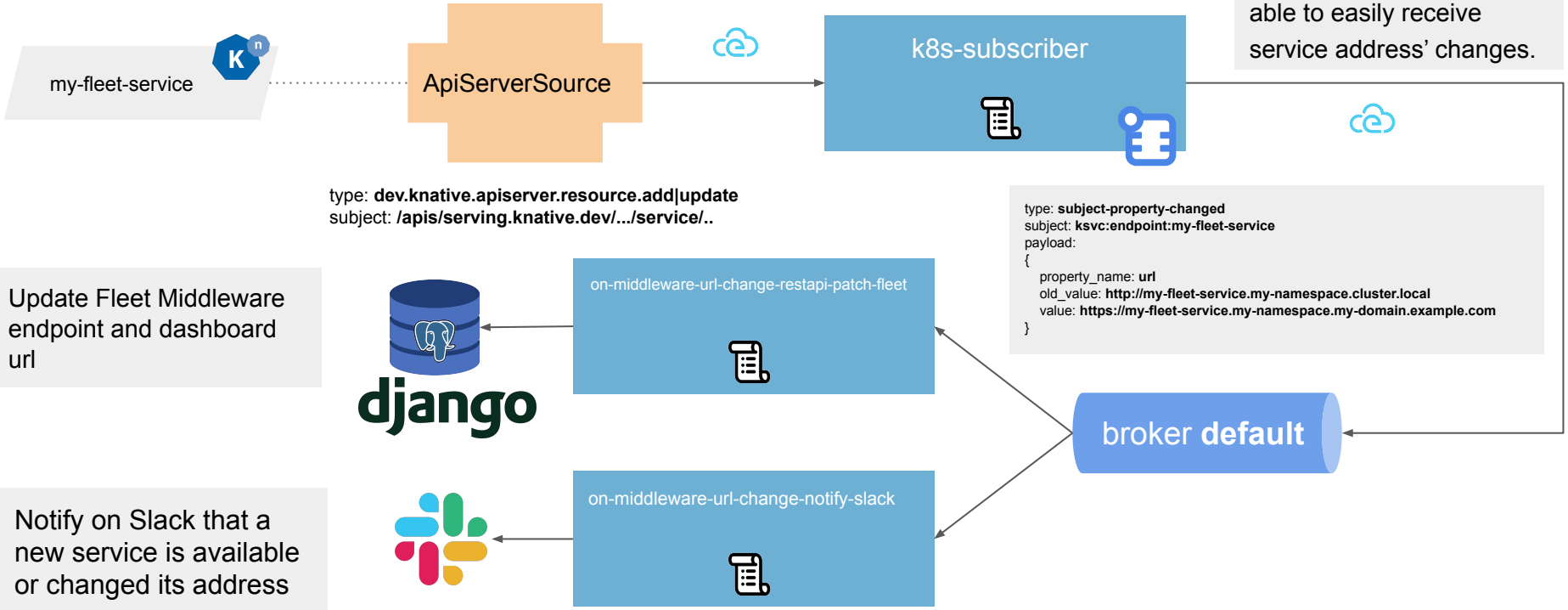
my-fleet middleware



On Fleet creation create **service** and **secret**, containing API key.
On each Fleet update, create a new **configuration** that originates a new **revision** consuming its own secret.
Each service will be coupled with a **dashboard**, which will receive service data in real time (Pusher websocket).

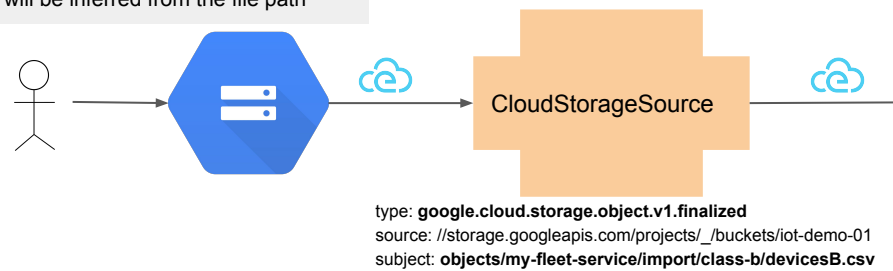
Middleware: on knative services create/update

url has been stored in a subject property on a subject acting as a twin of the service. System is now able to easily receive service address' changes.

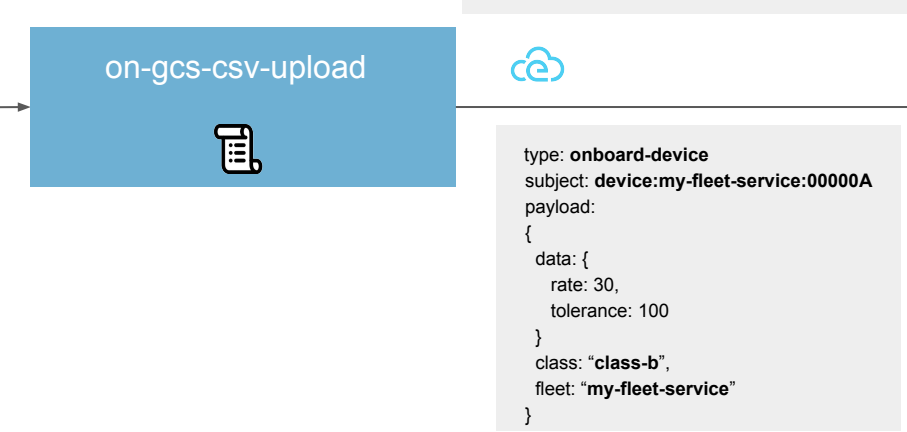


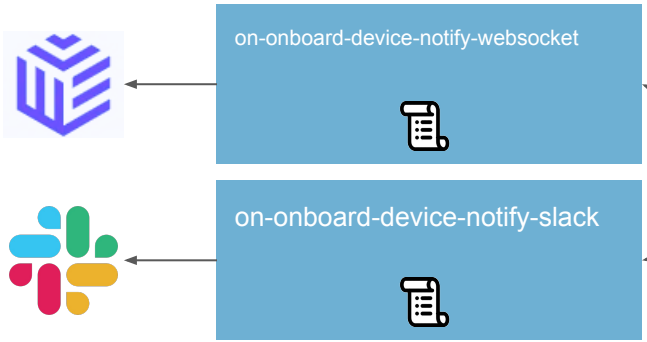
Onboarding: csv upload

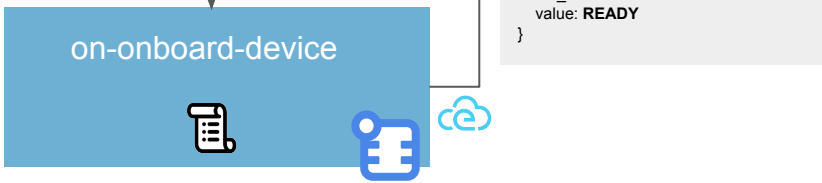
User uploads device list on gcs bucket as a csv file. **Device class** will be inferred from the file path



Read csv content and produce an **onboard-device** event for each line of it, creating a subject for each device







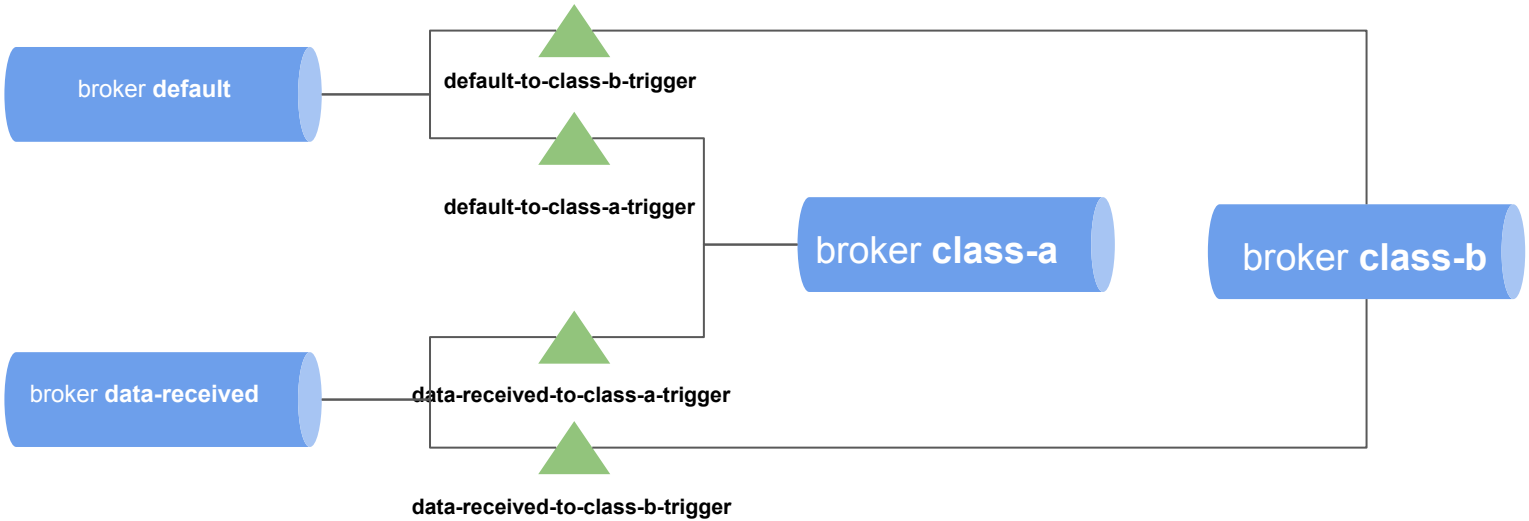
Set device subject initial status **READY** and base properties using **payload.data** and add **deviceclass** and **fleet** to subject **extended properties** enriching its future events' metadata

Airspot
cloud native development

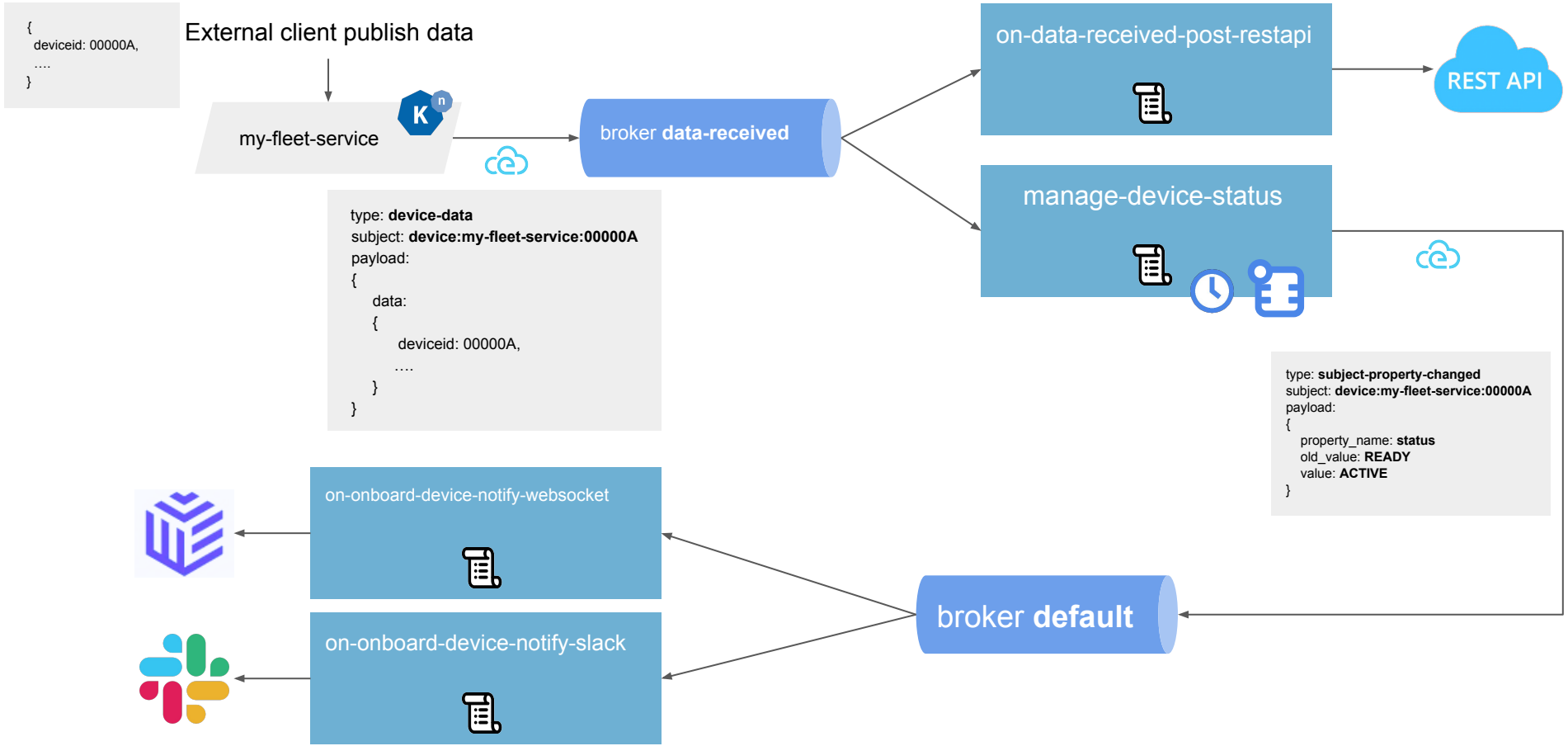
Onboarding: different classes of device

Starting from the csv file path a class device attribute is inferred becoming an **extended property** of device subject.
This information then is gathered as a cloudevent **extended attribute** allowing to define triggers to split events' traffic on totally dedicated broker, facilitating implementation of specific logic for each device class simply subscribing the appropriate broker.

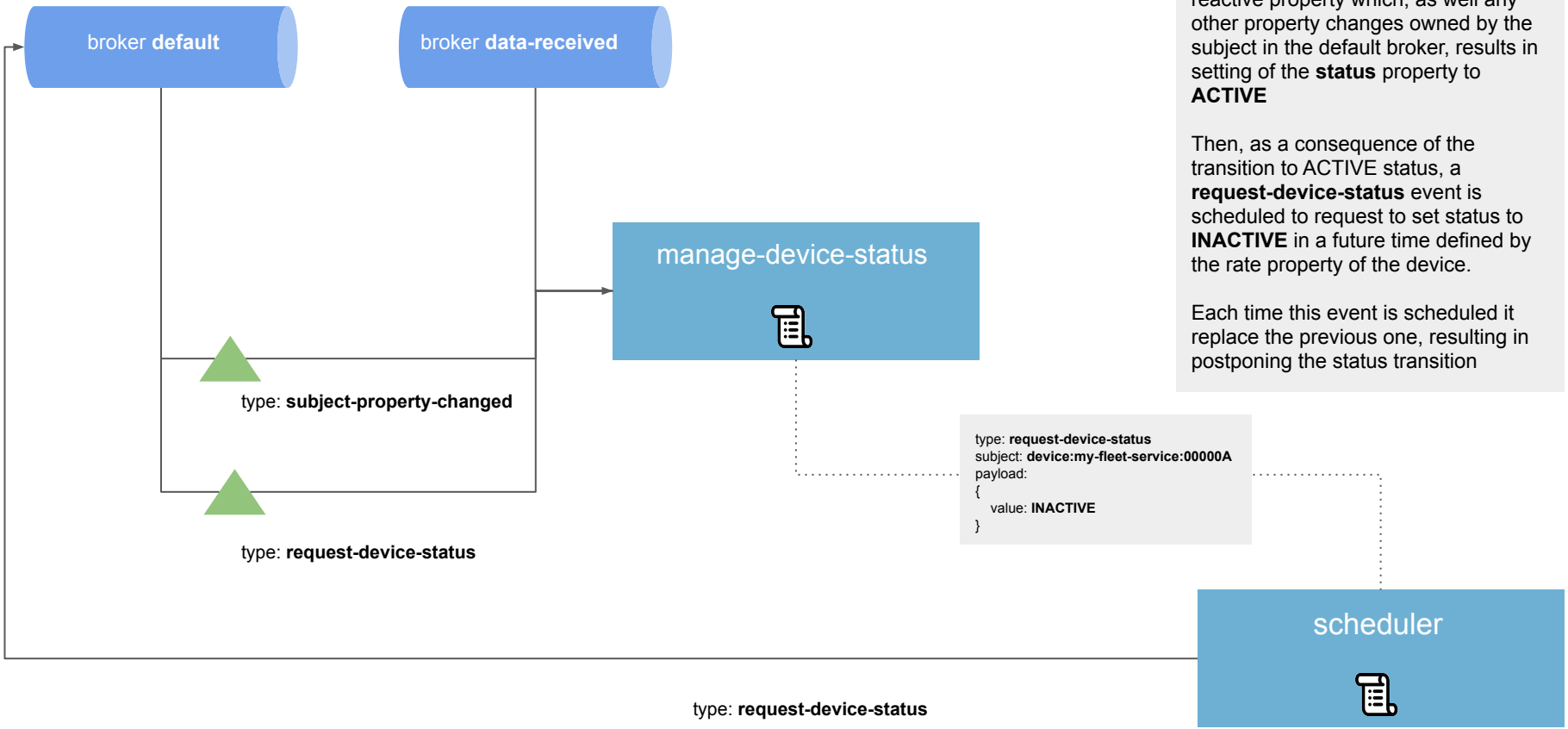
Thanks to knative eventing we potentially could have a different channel implementation for each broker which could allow us to simply define complex architectures, for example when we have an edge-to-cloud-to-edge scenario shaping routing logic dynamically inside rulesets.



Ingestion: on data received



Ingestion: device status management



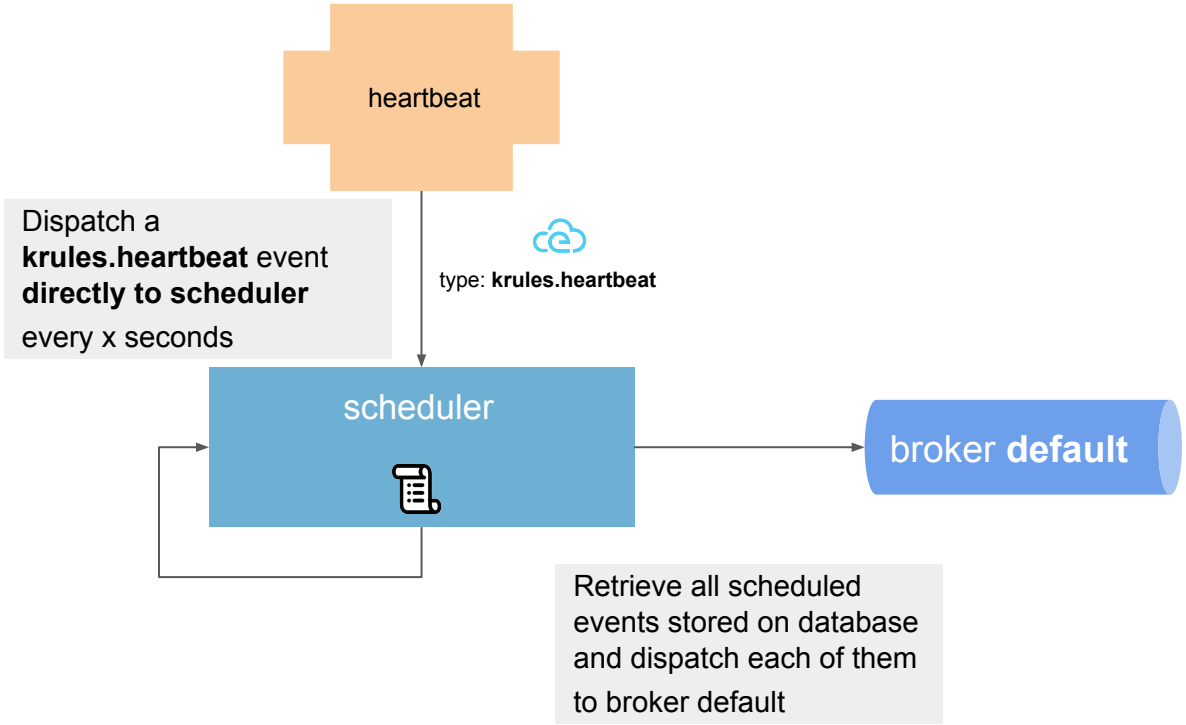
First of all, each event coming from data-received broker set the **lastSeen** reactive property which, as well any other property changes owned by the subject in the default broker, results in setting of the **status** property to **ACTIVE**

Then, as a consequence of the transition to ACTIVE status, a **request-device-status** event is scheduled to request to set status to **INACTIVE** in a future time defined by the rate property of the device.

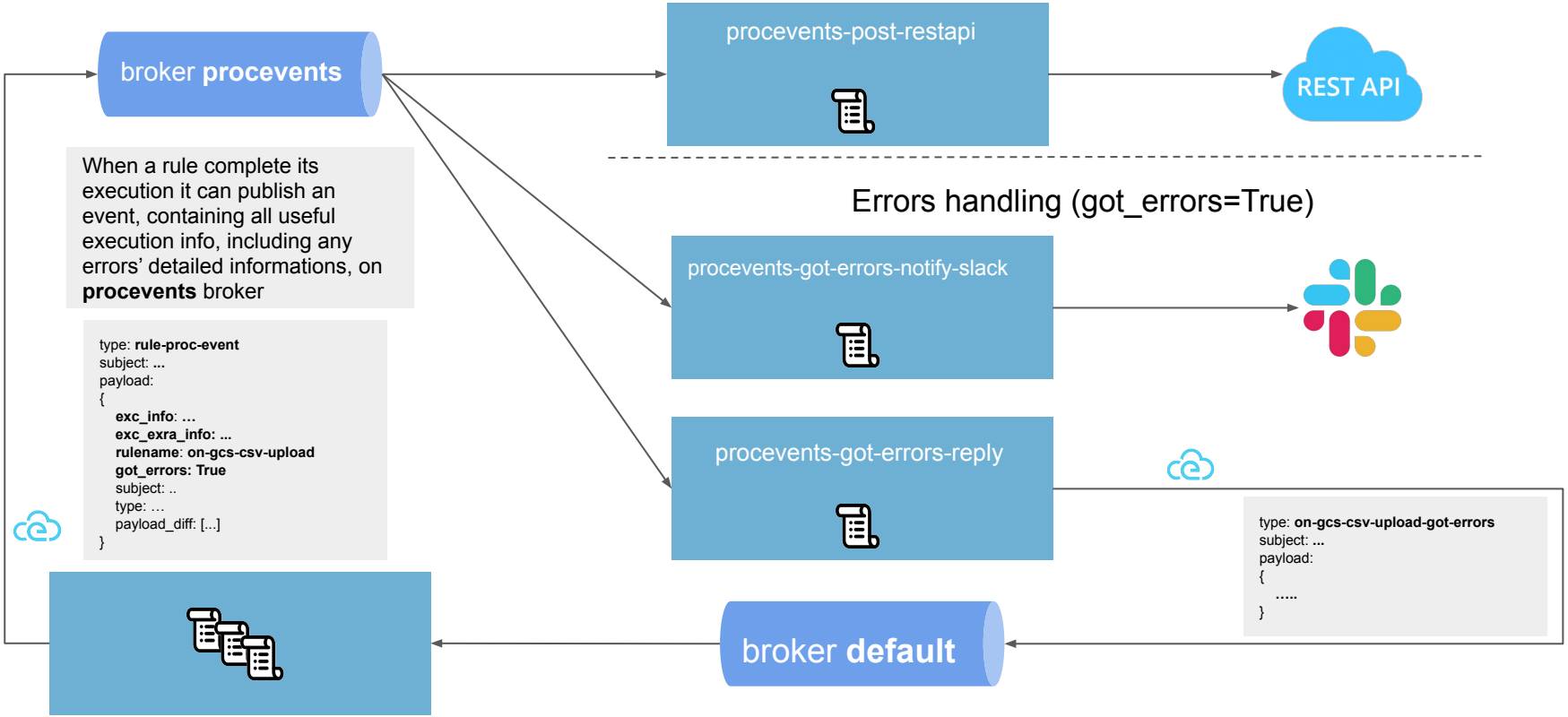
Each time this event is scheduled it replace the previous one, resulting in postponing the status transition

Airspot
cloud native development

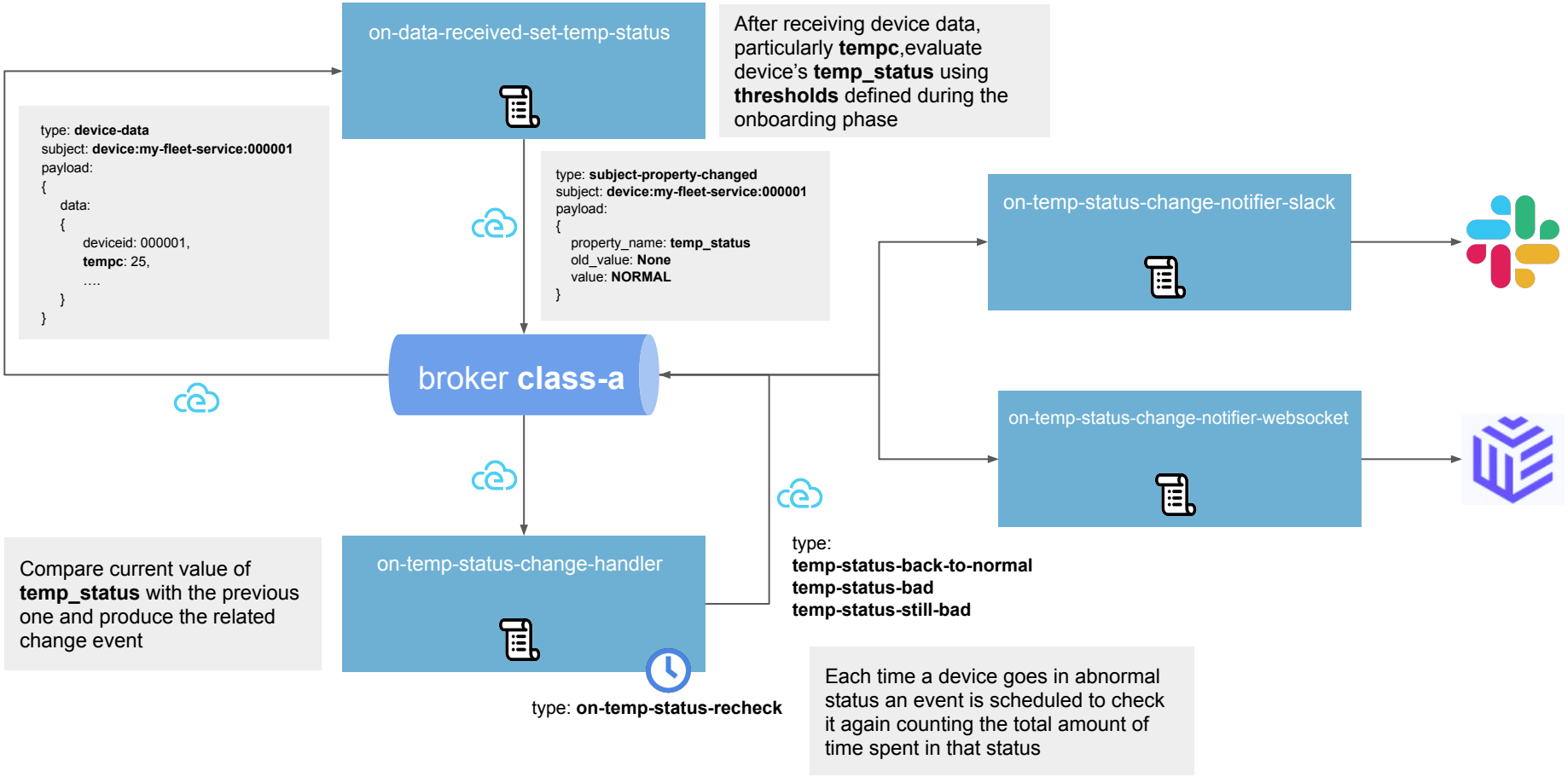
Scheduler



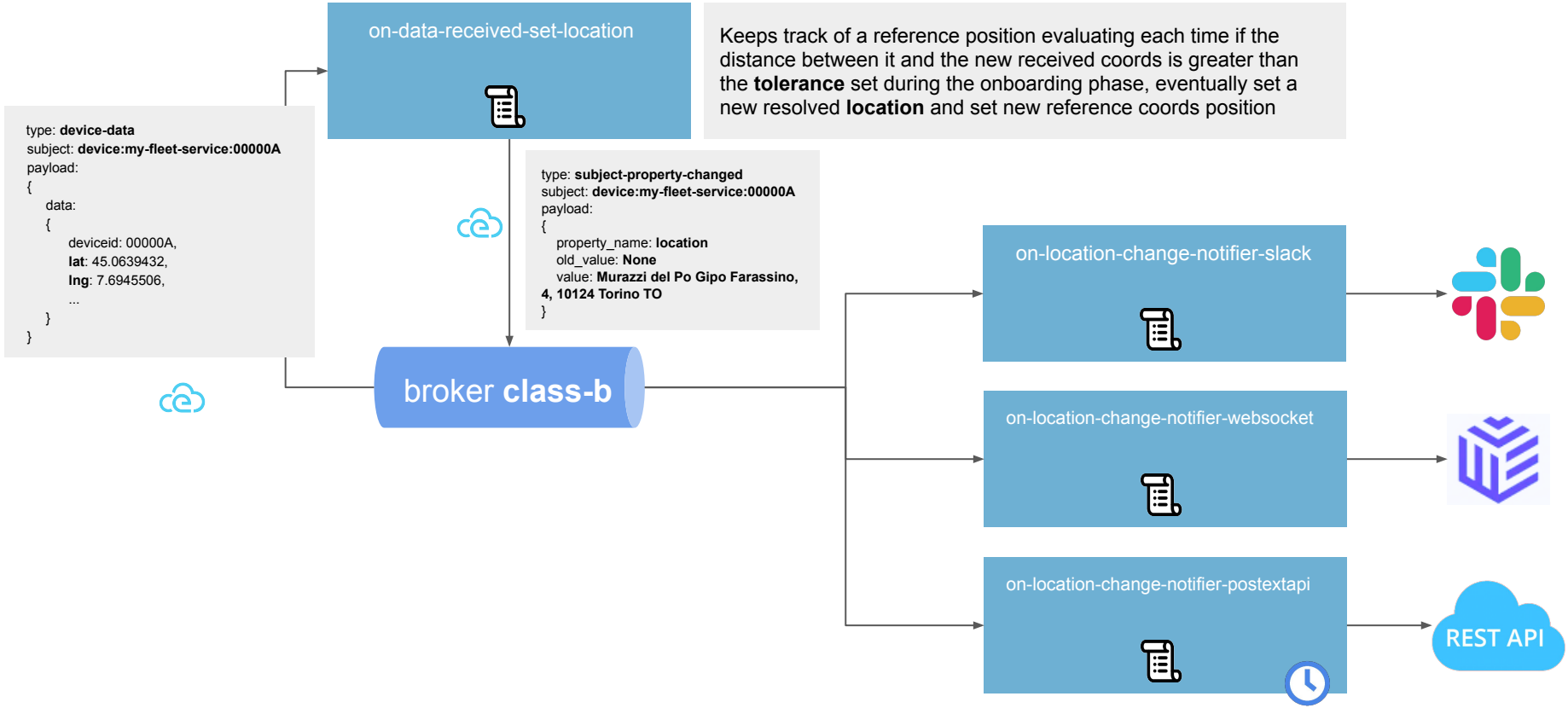
Procevents: error management



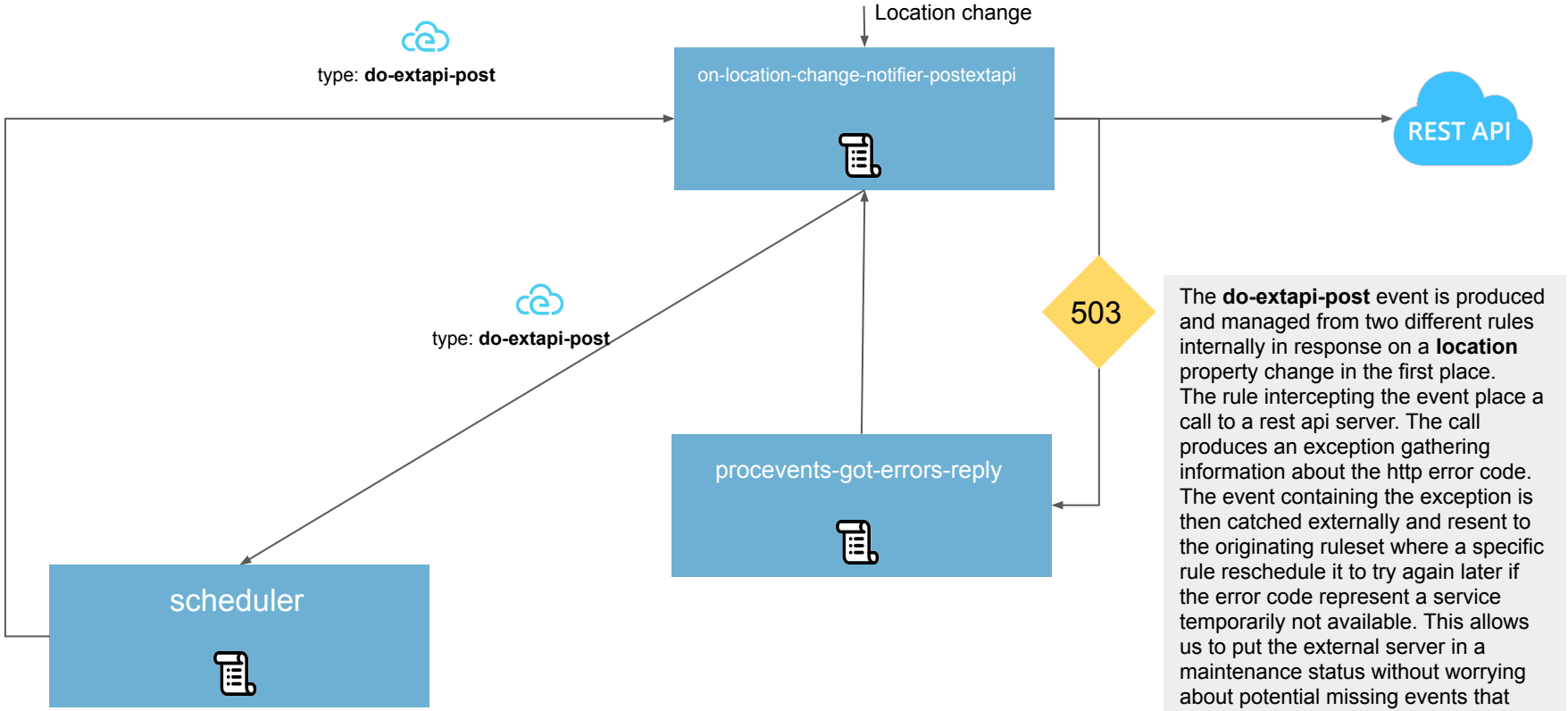
Class A: tracking temperature changes



Class B: tracking location changes



Class B: post location to an external API server subject to maintenance interruption



The **do-extapi-post** event is produced and managed from two different rules internally in response on a **location** property change in the first place.
The rule intercepting the event place a call to a rest api server. The call produces an exception gathering information about the http error code.
The event containing the exception is then catched externally and resent to the originating ruleset where a specific rule reschedule it to try again later if the error code represent a service temporarily not available. This allows us to put the external server in a maintenance status without worrying about potential missing events that they will be received all together when the service will come up again