UNECE Blockchain API

Serverless API to manage materials supply chain according to the UNECE standard

Endpoints

To enable data entry without using the GUI provided by the pilot project, REST APIs were created to enable the interaction directly through applications via HTTP calls.
The requirements follow the specifications given at the following link:
https://jargon.sh/redoc.html?url=/user/unece/traceabilityEvents/v/working/artefacts/openapi/render.json .

The exposed endpoints for which POST, GET, and PUT methods were implemented, are explained in detail below.

ObjectEvent

- POST -> */objectEvents* : it stores on chain a list of certifications starting from the information contained in the request body, formatted in the "ObjectEvent" JSON object, which is then saved into an IPFS.

  Since multiple certifications are obtained from a single event, an event id is created and associated with them. The latter is then returned as a response to be saved and used to view or modify the entities.

- GET -> */objectEvents/{eventID}* : it returns the ObjectEvent object with "*eventID*" as identification field.
- PUT -> */objectEvents/{eventID}* : all the certifications associated with "*eventID*" are retrieved and for each of them it is checked if it already exists (via the *"certificateID"* field) and if so an update takes place, otherwise a new certification is created with the information specified in the body.

  The content of ObjectEvent is saved into an IPFS and referenced to the created or updated entities and the old saved content is removed.

TransactionEvent

- POST -> */transactionEvents* : it stores on chain a trade and a list of relative materials, starting from the information contained in the request body, formatted in the "TransactionEvent" JSON object, which is then saved into an IPFS.

  The return value of the response is an identification of the created event, which can be used to view or modify the entities.

  IMPORTANT: a trade cannot be built if there are no materials specified (it requires at least one).

- GET -> */transactionEvents/{eventID}* : it returns the TransactionEvent object with "*eventID*" as identification field.
- PUT -> */transactionEvents/{eventID}* : the materials related to the trade are updated or created, if new ones are specified in the JSON object of the body, and the old trade is retrieved by the "eventID" parameter and its fields are updated.

  The content of TransactionEvent is saved into an IPFS and referenced to the created or updated entities and the old saved content is removed.

TransformationEvent

- POST -> */transformationEvents* : it stores on chain a transformation and a list of relative input and output materials, starting from the information contained in the request body, formatted in the "TransformationEvent" JSON object.

  The return value of the response is an identification of the created event, which can be used to view or modify the entities.

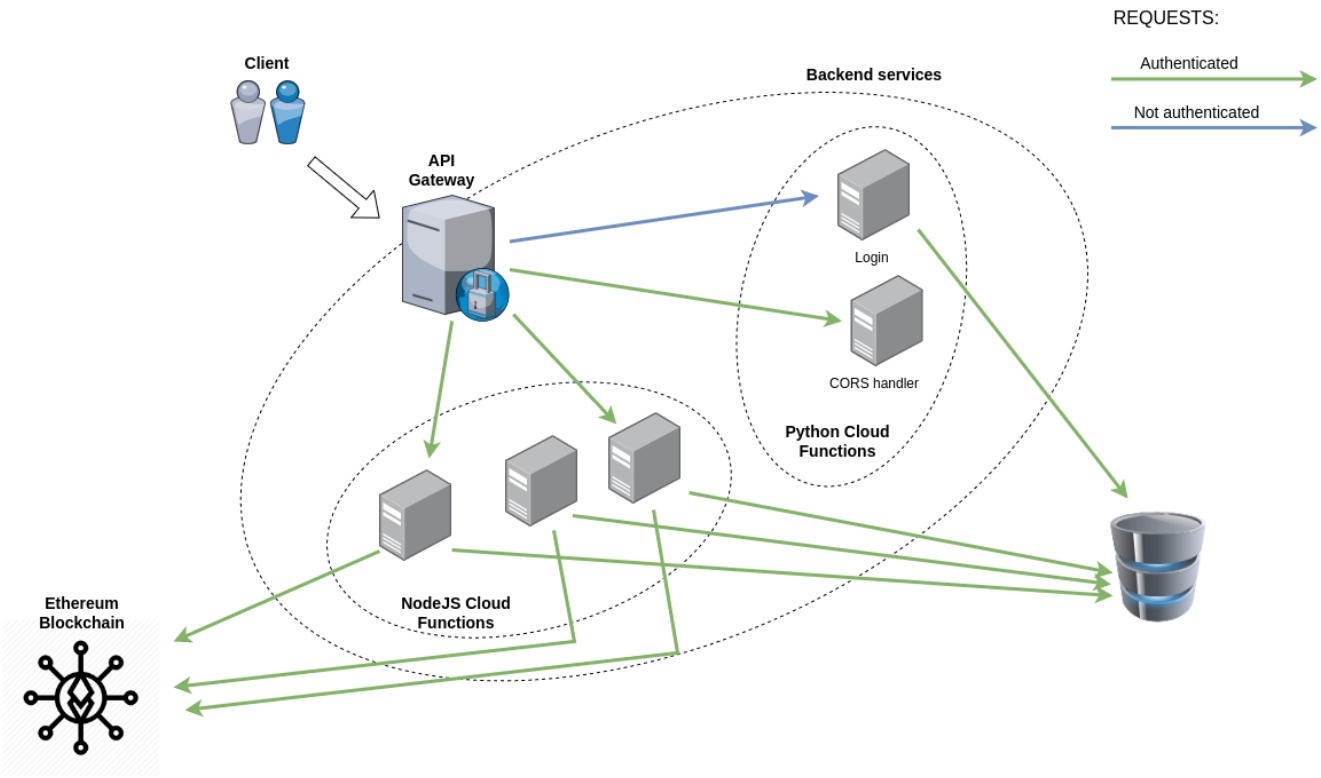
  IMPORTANT: a transformation cannot be built if there isn't at least one input material and one output material.

- GET -> */transformationEvents/{eventID}* : it returns the TransformationEvent object with "*eventID*" as identification field.
- PUT -> */transformationEvents/{eventID}* : the input and output materials related to the transformation are updated or created, if new ones are specified in the JSON object of the body, and the old trade is retrieved by the "eventID" parameter and its fields are updated.

  The content of TransactionEvent is saved into an IPFS and referenced to the created or updated entities and the old saved content is removed.

Technology stack

Endpoints are exposed through a serverless gateway offered by GCP (Google Cloud Platform) which invokes, depending on the call it receives, the dedicated cloud function.



As seen in the figure, the infrastructure consists of a distributed backend made up of a constellation of independent cloud functions that communicate with a remote MySql database and an Ethereum blockchain. The database currently resides within an always-on GCP virtual machine.

The backend is exposed by a gateway, which will be the only access point for the client.

API Gateway

API Gateway is a fully managed service to create, deploy and manage APIs in order to enable secure access to the backend services through a well-defined REST API that is consistent across all of the services. With a single entry point the developer can expose and connect multiple heterogeneous backend services, and protect their access using different kind of methods (for example by using JWT authorization token).

Each gateway deployed in a region contains an integrated load balancer to manage client requests to the API deployed to the gateway. Every cloud function is independent and is exposed by GCP with its own HTTPS address. In order to create a single ingress point for the client, a gateway was defined that, depending on the path that is passed in the URL, redirects traffic to the correct cloud function via internal endpoints.

Companies using the API only need to know the URL of the gateway, followed by the path defined by the endpoints created in the "Endpoints" chapter, in order to interface.

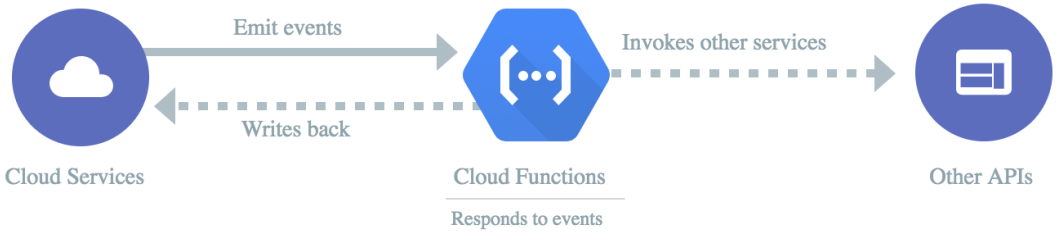
API Gateway URL: *https://api-gateway-8ot0fprs.ew.gateway.dev/api*

Cloud functions

Cloud functions are a FaaS (Function as a Service) system with pay-as-you-go, which can scale up according to load and at the same time does not require fixed active resources..

Their main advantages are:

- automatic scaling according to load
- monitoring and logging system (build and execution logs)
- integrated security at role level and function access based on the principle of minimum privilege
- no need to provision an infrastructure or worry about managing a server because infrastructure is managed directly by the provider
- functions can be written using a variety of languages such as Javascript, Java, Python, PHP, Ruby, Go and .Net.
- only the execution time of the function, calculated to the nearest 100 milliseconds, is invoiced
- there is no build-time quota even if the Cloud Build service has it own default quota
- configurable amount of available resources according to the estimated workload



Once deployed they can be exposed securely via https in order to be triggered and then they can interact with other APIs, such remote DB or other cloud functions, and finally write back a response to the service that has called them.

The logic that creates and manages the entities defined by the UNECE standard is fully managed in a serverless way through the GCP cloud functions. In this way, the platform can scale automatically in case of traffic peaks and provides significant cost savings since the computational logic becomes on-demand.

Each endpoint is resolved by a different function, which is invoked by the Gateway API. This generates an HTTP trigger that starts the instance which executes the function, passing it from the request, the original body and a header containing an authentication JWT token, which is previously created by the login phase. When the function has finished executing, the instance shuts down without consuming any more resources.

Cloud functions can be written heterogeneously using different programming languages. Below we see those used and for what purpose.

Python

The functions written with this language are, in this context, used as a support and utility. They solve the CORS problem and the initial authentication operation by verifying that the user (or application via API key) making the request is correctly present in the database and creating a JWT token that is injected into subsequent authenticated requests.

NodeJS

These functions are written using TypeScript, a typed language based on JavaScript, and they are engaged to execute the functionality exposed by the endpoint.

Each type of HTTP call (GET, POST, PUT) to the gateway, generates a trigger to dedicated cloud functions that resolve the request and make persistent the entities created on blockchain through a specifically developed library.

All context or private information are dynamically retrieved via Google Secret, allowing security while enabling rapid customization without functions re-deploying; i.e. to deploy functions to a new contract address or Ethereum network to connect to, simply update the related secrets directly from the GCP console without making changes to the codebase.

Storages

MySQL

The information needed to authenticate a user is contained in a MySQL database, which is currently containerized on a virtual machine provided by GCP.
The database will be invoked during the login phase and subsequently by every cloud function to retrieve the private key of the wallet related to the logged-in user, identified by his username contained in the JWT token.

Ethereum Blockchain

As mentioned above, the database is used as it holds the information of the user, company and relative wallet, while the Ethereum blockchain is meant to store the entities that are extracted from the body of the requests to the cloud functions.

A library, developed internally at the institute, enables the abstraction of authentication and communication between the function and blockchain via providers. It enables the actual storing of information and, with additional logic, also the retrieval and modification.

All the data needed to build the desired supply chain are written on chain.

Pinata IPFS

An IPFS (a distributed system for storing and accessing data) has been configured to save any type of blobs or “heavy” metadata, whose reference is associated with the entities stored on chain.

Again, the supporting library abstracts the communication and the secrets system allows the IPFS type to be changed dynamically without requiring a change in the codebase.