3

Backplane API Gateway

3.1 Objectives

The Backplane gateway system is the building block in charge of offering to all participants and marketplaces access to the Backplane system. The goal of the Backplane API is therefore twofold: on the one hand, it serves an integrated API endpoint for all the i3-MARKET services offered by i3-MARKET and implemented in the respective building blocks. On the other hand, it provides secure mechanisms for preventing not-allowed accesses.

In terms of internal connections with other i3-MARKET building blocks, Backplane gateway system has secure communication with the rest of subsystems to integrate their services into the Backplane API, in order to provide secure access to authorized clients.

The Backplane API is the set of endpoints exposed by the gateway. It comprises all the publicly available endpoints of the subsystems integrated with the Backplane, as well as a few other endpoints, belonging to the Backplane itself, used in the authentication/authorization flows.

The API follows the OpenApi Specification 3.0[1]. Furthermore, the endpoints corresponding to each subsystem are generated automatically based on the subsystem's own OpenApi specification, using the *service integrator engine*, written in Dart.

In Figure 3.1, there is an overview of the overall Backplane gateway architecture. It shows how the Backplane router incorporates all subsystem endpoints; so it can redirect each query to the corresponding subsystem, applying an authentication layer above to avoid unauthorized requests. Users can access to the Backplane gateway via the Backplane API, which publishes all available subsystems together with their endpoints, being totally agnostic of its implementation and how to access the subsystem directly.

[1]https://swagger.io/specification/

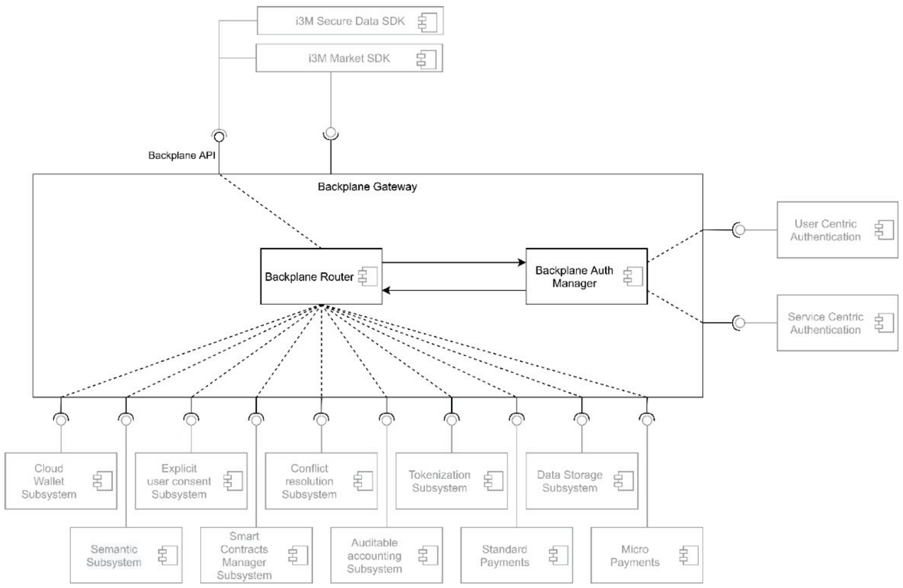

Figure 3.1 Backplane gateway architecture.

The Backplane gateway exposes all subsystem endpoints through a single Backplane API. This simplifies the user interaction with the system; furthermore, it provides an auto-generated documentation that follows the OpenApi specification (OAS).

3.2 Solution Design/Blocks

3.2.1 Authentication and authorization

In the current Backplane API gateway implementation, OAuth 2.0[2] authentication flow is used. Combined together with OpenID Connect (OIDC)[3], that provides a simple identity layer on top. Using OAuth Authorization Code flow (see Figure 3.2), a JWT token is generated at the end of the login flow, which, later, can be used in subsequent queries to authenticate clients against subsystem endpoints, using the Backplane API as gateway.

[2]https://oauth.net/2/
[3]https://openid.net/connect/

3.2.1.1 Authentication

Clients are expected to request their JWT token through a given login endpoint, to further request secured endpoints using those credentials.

Thanks to the OpenID Connect identity layer, scopes and claims can be used. Each endpoint can declare a set of scopes, which will be later used to ensure that the requesting user has enough privileges, in a claim-based authorization fashion.

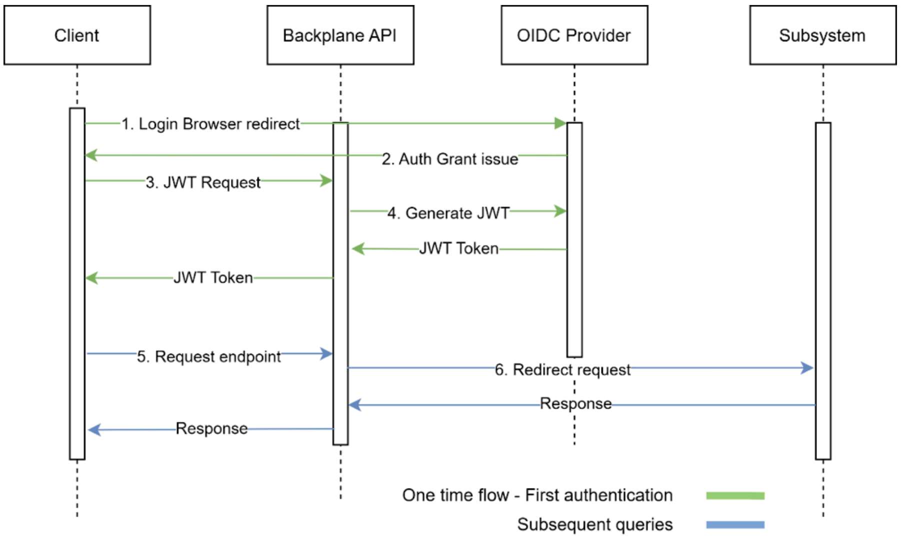

Figure 3.2 Backplane authentication flow overview.

There is a description of each connection considered during the authentication flow described in Figure 3.2:

1. **Login browser redirect:** When a user requests a Backplane authenticated endpoint without providing the required credentials, it is redirected to the identity provider authorization page (OIDC provider).
2. **Auth grant issue:** In case login succeeds, an authorization grant is issued and provided to the client.
3. **JWT request:** The client requests an access token, providing the Auth grant code.
4. **Generate JWT:** Now, the Backplane generates an access token JWT, adding the user claims that are requested to our identity provider.

5. **Request endpoint:** The client uses the previously generated JWT to authenticate their requests to the Backplane.
6. **Redirect request:** In case the user has enough privileges to access the requested endpoint, checking the endpoint scope and user claims, the Backplane will redirect the query to the corresponding subsystem endpoint.

3.2.1.2 Authorization

After performing the whole authentication flow, clients will end up with two JWT tokens:

- **access_token:** Contains the subject id, together with the scope.
- **id_token:** Contains information about the user itself, including the Verifiable Credentials associated with the corresponding claims, based on the user profile.

Clients are expecting to provide those tokens in the header part when querying a secured endpoint. Figure 3.3 illustrates the authorization flow.

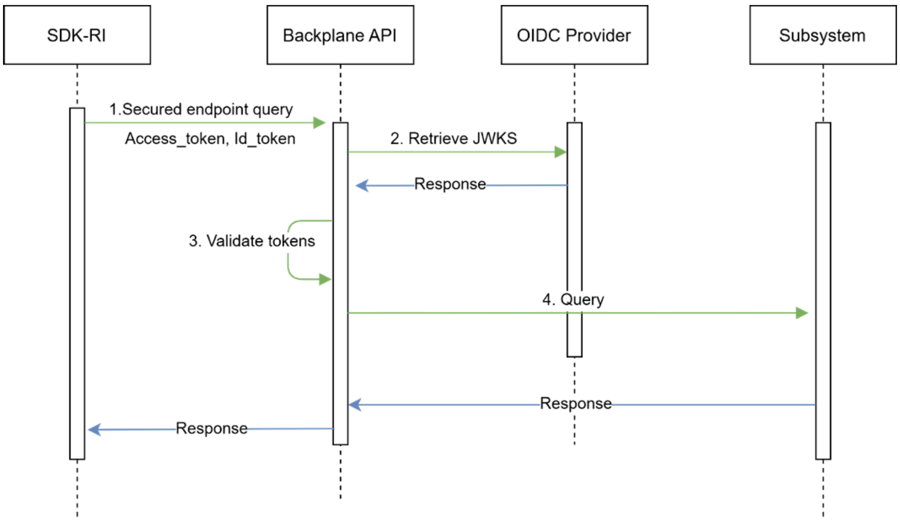

Figure 3.3 Backplane authorization flow overview.

1. **Secured endpoint query:** Clients are expected to include the access_token and id_token headers when requesting a Backplane authenticated endpoint.

2. **Retrieve JWKS[4]:** The OIDC uses token asynchronous validation; so the Backplane just needs to retrieve the JWKS, an array of public cryptographic keys, in order to validate each token in offline mode using EdDSA[5], a public-key cryptography signature algorithm.
3. **Validate tokens:** The Backplane internally validates the tokens' signature and verifies that the user has the required claims to access the endpoint.
4. **Query:** The query is redirected to the subsystem, together with the id_token header, containing a JWT token that describes the requester.

3.2.2 Subsystem implementation

While subsystems do not need to worry about authentication, they need to indicate in their OAS specification which of their endpoints are protected and which are not. To mark an endpoint as protected, it must include:

- **JWT security reference:** The endpoint specification must show that JWT is used as a means of authentication. This is done by adding de JWT schema to the security field, specifying if needed the claims required to access the endpoint.

```
“security”: [
  {
    “jwt”: [“consumer”]
  }
]
```

Then, clients must define the security schema as an ApiKey, expected to be presented in the header *id_token*:

- **JWT security schema:** Add the following security schema to the subsystem OpenApi specification (OAS):

```
"securitySchemes":{
"jwt":{
      "type": "apiKey",
      "in": "header",
      "name": "id_token"
}
},
```

[4]JSON Web Key Sets (https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-key-sets).

[5]https://www.rfc-editor.org/rfc/rfc8032

Note: There is no need to define the *access_token* explained before, as it is only being used by the Backplane itself; so, subsystems can ignore it.

With the above-stated OAS modifications, the service integrator engine will add the required authorization mechanism to each endpoint, automatically, during Backplane deployment pipeline, as described in Section 3.2.5.

3.2.3 Data flows

When a service is integrated into the Backplane, it means that its resources can be accessed through the Backplane itself. So, when a client application accesses to a resource into the Backplane, it will redirect the request to the final resource path, specified in the resource provider OAS file.

Thanks to this approach, the client is agnostic of the final location of the required service, being all handled by the Backplane.

The Backplane establishes a communication using JWT authentication between the Backplane and the service to ensure data protection. This communication can also be easily secured using certificates HTTPs/TLS.

3.2.4 Service Integration Manager

The service integration manager is one of the key components of the i3-MARKET Backplane. It ensures the easy integration of any subsystem service to the i3-MARKET Backplane, using OpenAPI specification as bridge.

The Manager is written in Dart[6] and is the one responsible for external service integration to the Backplane API; so it is capable of acting as a gateway for this new service. In Figure 3.4, there is an overview of how the service integration manager works, proceeding with the following steps:

1. **Generate resources:** Given a new service OpenAPI specification, it runs the Loopback CLI OpenAPI generator command[7], which generates the specified *controllers* and *data sources* that later will be integrated into the final Backplane API Docker image.
2. **Integrate + Build:** As the Loopback CLI just provides a set of skeletons, some modifications need to be performed to the previously generated sources, customizing them for our use case. Then, it can be integrated to the Backplane API base code, building the final Backplane Docker image, ready to be used for deployment.

[6]https://dart.dev/

[7]https://loopback.io/doc/en/lb4/OpenAPI-generator.html

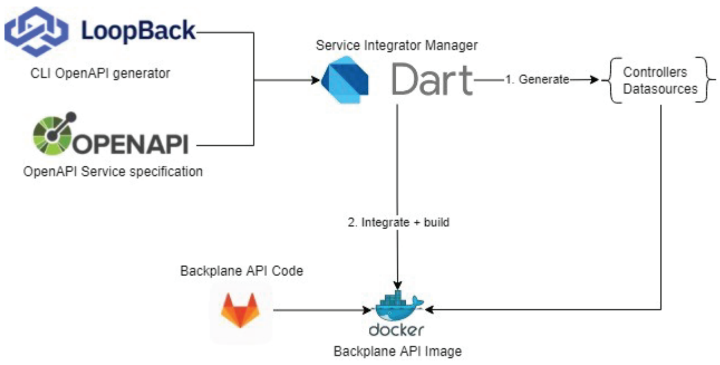

Figure 3.4 Service integrator process overview.

3.2.5 Automatic integration mechanism

In order to provide an easy onboarding experience, it is mandatory to build mechanisms to achieve easy and automated marketplaces and service integration. In order to achieve these goals, the consortium decided to use GitLab CI pipelines[8] together with Ansible playbooks[9], being GitLab responsible of artifact generation and Ansible of the deployment to i3-MARKET nodes.

3.2.6 Subsystem OAS repository

The integration process begins when an i3-MARKET maintainer validates a given subsystem OAS (OpenApi specification) and, hence, merges a pull request into the master branch adding or modifying a definition.

The lack of validation proofs hinders the i3-MARKET maintainer job, causing sometimes the approval of OAS files with errors or incompatibilities, which in the end break the Backplane. At this point, we found the need of implementing a CI/CD pipeline with a job responsible for validating the files, together with the correct integration within the Backplane base code, as described in Figure 3.5, performing the following steps in order:

1. **Validate the OAS file:** All the OAS files are collected and the API definition of each one is validated, using the npm swagger-cli[10] utility.

[8]https://docs.gitlab.com/ee/ci/pipelines/
[9]https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html
[10]https://www.npmjs.com/package/swagger-cli

2. **Clone Backplane repository:** In this step, we are cloning the Backplane repository. This is a needed step in order to verify the OAS files are compatible with the integrator and the Backplane itself.
3. **Integrate OAS:** In this step, using the latest integrator engine available, we are integrating all the OAS files into the base Backplane code. In case some error or incompatibly is reported, the whole pipeline fails and notifies the i3-MARKET maintainer.
4. **Integration test:** This step starts a Backplane instance only accessible locally. Then, using a tool called schemathesis[11], we are testing all the endpoints of the Greeter subsystem[12], making sure none of them return an error 5XX. Note the tool is not testing all the subsystem endpoints, given the fact that we cannot assume the status of all of them. We found out that scanning a single known subsystem is enough to detect common failures.
5. **Release new version:** At this moment, we could say the OAS files are safe to be deployed; so, a new tag is being created and pushed into the Backplane repository. Triggering the Backplane automatic integration pipeline is explained in the next section.

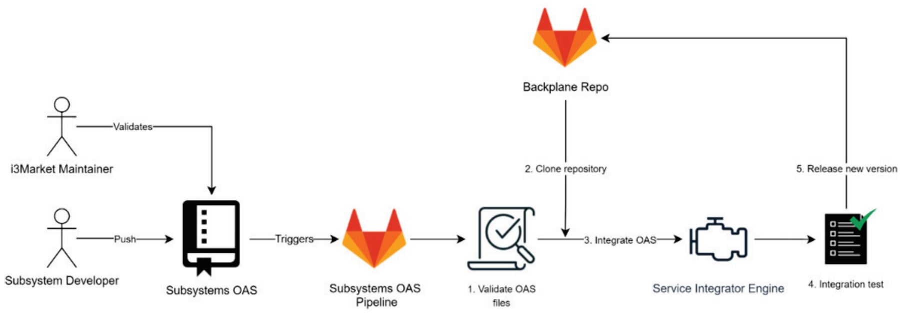

Figure 3.5 Subsystem OAS automatic integration mechanism overview.

3.2.7 Backplane repository

Validated updates on the subsystem OAS repository trigger the Backplane automatic integration mechanism, described in Figure 3.6, performing the following steps in order:

1. **Run the service integrator engine:** The engine artifact is collected from the corresponding code repository, and the code components that

[11]https://github.com/schemathesis/schemathesis
[12]Mockup of an OAS subsystem, created as an example for the rest of partners.

later will be integrated to the final Backplane artifact are generated. The functionality of the service integrator is fully explained in the previous section.

2. **Check vulnerabilities:** In this phase, a vulnerability check using Trivy[13] is performed, a vulnerability scanner developed by AquaSecurity[14]. This step scans NPM and OS libraries, marking the pipeline as failed in case any critical vulnerability is found.
3. **Integration test:** This step verifies the functionality of the fully integrated Backplane, as explained in the section before (subsystem OAS repository).
4. **Build image:** Using the code stored in the Backplane repository, together with the output of the service integrator, a new Docker image for production deployment is generated and uploaded to the project registry; so future deployment can easily be performed using Docker.
5. **Deploy:** The pipeline triggers the deployment Ansible playbooks, which deploy the Backplane API using the Docker image built previously, along with the i3-MARKET SDK Docker image.
6. **Update the developer portal:** In parallel to this process, because a new OAS has been uploaded to the project, the developer portal must be updated, triggering the documentation repository pipeline. It generates a new developer portal artifact and deploys it using GitLab Pages[15].

3.2.7.1 Remote images

All production-ready images can be found in the private and public repositories managed by the consortium (GitLab and Nexus). Currently, we are providing two different image flavours:

- **Major.minor.patch:** Base Backplane image, which includes the latest subsystem OAS available at the build instant.
- **Major.minor.patch-with-integrator:** Built from the base image, although it also includes the integrator binary under */integrator* path. This image provides a custom entry point that will check the existence of custom OAS files under */home/node/app/specs*. If affirmed, the integrator will integrate those specs into the base Backplane image before

[13]https://github.com/aquasecurity/trivy
[14]https://www.aquasec.com/
[15]https://docs.gitlab.com/ee/user/project/pages/

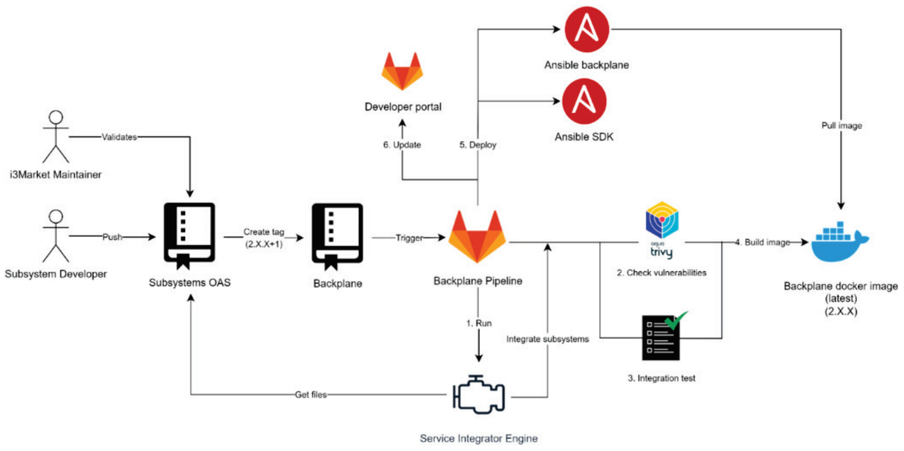

Figure 3.6 Backplane automatic integration mechanism overview.

running the Backplane; otherwise, the integration phase will be skipped, and the Backplane will be executed using the latest OAS definitions at the image compilation instant.

Both image flavours can be pulled using the described versioning format (*major.minor.patch*) or the *latest* tag to get the most recent version.

3.2.8 Final deployment

Final deployment phase, described in Figure 3.7, is orchestrated using a single Ansible playbook triggered by the GitLab CI pipeline described before. Actually, during this testing phase, four i3-MARKET nodes are being considered, each one performing the following tasks:

1. **Get config files:** Queries against i3-MARKET nexus repository are being executed in order to obtain the required configuration files for each node.
2. **Get Backplane Docker image:** The latest Backplane image is retrieved from the GitLab Docker image registry used in the artifact construction phase.
3. **Start Backplane container:** Now, the running container is replaced, launching a new one with the latest image, configuring the volumes and environment variables required.

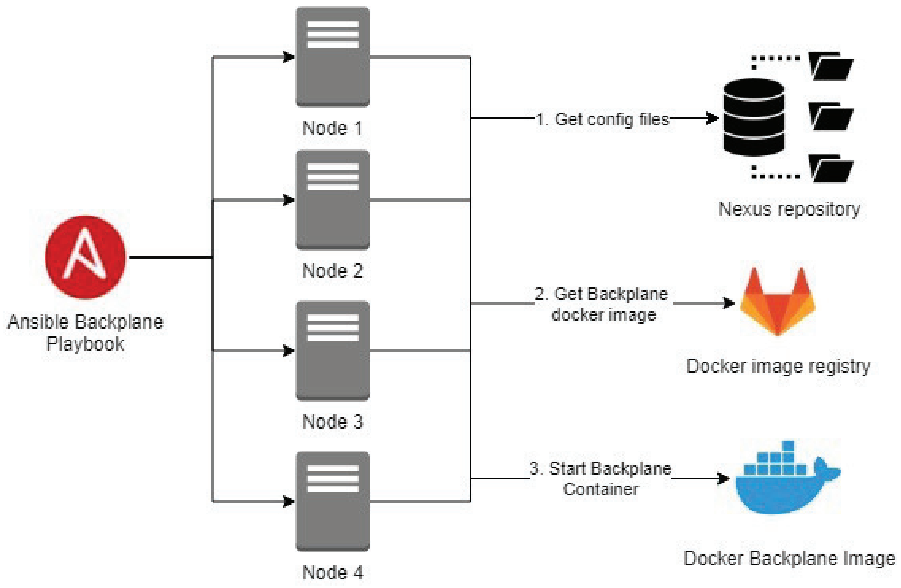

Figure 3.7 Ansible playbook run overview.

3.2.9 Multiple environments support

One of the limitations found in the current Backplane was the lack of support for multiple environment deployments. Specific OAS files had to be written for each environment, identical, except for the servers' annotation, that might differ based on the environment characteristics. Instead, we found out a way to support this requirement without having to duplicate OAS definitions.

Right now, we are using the Open API "servers" specification to indicate all servers providing the stated service, using custom tags to identify the ones to be used in each environment. For example:

```
{
      "servers": [
            {
              "url": "http://conflict-resolver-service:3000/",
              "x-tags": ["docker-compose"]
            },
            {
              "url": "http://node1.i3-MARKET.com:8888/",
              "x-tags": ["nodes"]
            },
            {
              "url": "http://node2.i3-MARKET.com:8888/",
              "x-tags": ["nodes"]
            }
      ],
}
```

In the previous definition, there are three different nodes providing the same service. Using the “x-tags,” we can tag each server in order to choose at start-up time which set of servers has to use the Backplane to redirect the queries for each service.

The Backplane can filter and choose the most convenient server based on the *SERVER_FILTER_TAGS* environment variable definition, a comma separated list of tags to indicate the servers to use.

Figure 3.8 shows one server that can be used to redirect queries; hence, in case the previous selector gets multiple server options, a DNS resolution probe is executed for each hostname to choose the first available option. Given the heterogeneity of subsystems, the Backplane cannot assure availability of each server, as it lacks any liveness endpoint definition to test; furthermore, the Backplane is agnostic of the service functionality that it provides and its behaviour. Below, there is an example considering only the nodes tagged with “node,” where node2 is being selected because node1 failed the DNS resolution.

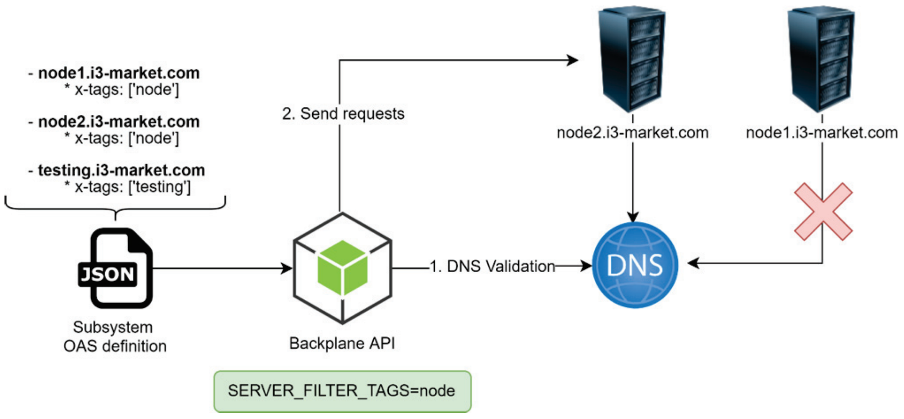

Figure 3.8 Server election process example.

We are aware that this approach is quite naïve, as host DNS availability does not prove there is a current API working in the server. However, it solves common issues of multiple environment deployments.

In order to improve the server election mechanism, we would need to enforce a liveness/readiness endpoint in marketplace definitions, which could also lead to including algorithms to failback to an alternative server in case the main one fails.

3.3 Interfaces

Backplane API for the i3-MARKET project:

3.3.1 Developers

GET `/OpenIDConnectProvider/release2/developers/login`

Obtain a valid initial_access_token for registering a new client

POST `/OpenIDConnectProvider/release2/oidc/reg`

Registering a new client

3.3.2 OIDC discovery

GET `/OpenIDConnectProvider/release2/oidc/.well-known/openid-configuration`

Get OpenID Provider configuration information

3.3.3 OIDC core

GET `/OpenIDConnectProvider/release2/oidc/auth`

Request authorization code

GET `/OpenIDConnectProvider/release2/oidc/jwks`

Get JSON Web Key Set

POST `/OpenIDConnectProvider/release2/oidc/token`

Request access token and id token with authorization code or refresh token

3.3.4 RegistryBlockchainController

POST `/auditableAccounting/calculateMerkleRoot`

GET `/auditableAccounting/getCurrentRoot`

POST `/auditableAccounting/updateRegistries`

3.3.5 RegistryController

GET `/auditableAccounting/registries/count`

PUT `/auditableAccounting/registries/{id}`

PATCH `/auditableAccounting/registries/{id}`

GET `/auditableAccounting/registries/{id}`

DELETE `/auditableAccounting/registries/{id}`

POST `/auditableAccounting/registries`

PATCH `/auditableAccounting/registries`

GET `/auditableAccounting/registries`

3.3.6 AuthController

GET `/auth/openid/callback`

GET `/auth/openid/login`

GET `/auth/whoAmI`

3.3.7 Conflict-resolver service

POST `/conflictResolverService/dispute`

Initiates a dispute claiming that a cipherblock cannot be decrypted and thus that the data exchange is invalid

POST `/conflictResolverService/verification`

Verification request of completeness of non-repudiation protocol regarding a data exchange

3.3.8 FarewellController

POST `/greeter/farewell/body`

GET `/greeter/farewell/headerParams`

GET `/greeter/farewell/pathParams/{name}/{age}`

GET `/greeter/farewell/queryParams`

3.3.9 HelloController

GET/greeter/hello/authenticated

GET/greeter/hello/consumer

GET/greeter/hello/provider

GET/greeter/hello/unauthenticated/{name}

3.3.10 OpenApiController

GET/notification-manager-oas/api/v1/health

Version

GET/notification-manager-oas/api/v1/version

Version

3.3.11 Notifications

POST/notification-manager-oas/api/v1/notification/service

Notification service

GET/notification-manager-oas/api/v1/notification/unread

Get unread notifications

GET/notification-manager-oas/api/v1/notification/user/{user_id}/unread

Get unread notifications by id

GET/notification-manager-oas/api/v1/notification/user/{user_id}

Get notification by Userid

PATCH/notification-manager-oas/api/v1/notification/{notification_id}/read

Modify notification

PATCH/notification-manager-oas/api/v1/notification/{notification_id}/unread

Modify notification

GET/notification-manager-oas/api/v1/notification/{notification_id}

Get notification

DELETE/notification-manager-oas/api/v1/notification/{notification_id}

Delete notification

POST/notification-manager-oas/api/v1/notification

Notification user

GET/notification-manager-oas/api/v1/notification

Get notifications

3.3.12 Queues

PATCH/notification-manager-oas/api/v1/services/{service_id}/queues/{queue_id}/activate

Switch status queue

PATCH/notification-manager-oas/api/v1/services/{service_id}/queues/{queue_id}/deactivate

Switch status queue

GET/notification-manager-oas/api/v1/services/{service_id}/queues/{queue_id}

Get queues by id

DELETE/notification-manager-oas/api/v1/services/{service_id}/queues/{queue_id}

Delete queue

POST/notification-manager-oas/api/v1/services/{service_id}/queues

Post queues

GET/notification-manager-oas/api/v1/services/{service_id}/queues

Get queues

GET/notification-manager-oas/api/v1/services/{service_id}

Get services by id

DELETE/notification-manager-oas/api/v1/services/{service_id}

Delete service

POST/notification-manager-oas/api/v1/services

Create service

GET/notification-manager-oas/api/v1/services

Get services

3.3.13 Subscriptions

GET/notification-manager-oas/api/v1/users/subscriptions/{category}

Returns a Json containing a list of users subscribed to that category

GET/notification-manager-oas/api/v1/users/subscriptions

Get all user subscriptions

PATCH/notification-manager-oas/api/v1/users/{user_id}/subscriptions/{subscription_id}/activate

Activate or deactivate user subscription

PATCH/notification-manager-oas/api/v1/users/{user_id}/subscriptions/{subscription_id}/deactivate

Activate or deactivate user subscription

GET/notification-manager-oas/api/v1/users/{user_id}/subscriptions/{subscription_id}

Get user subscription by user_id and subscription_id

DELETE `/notification-manager-oas/api/v1/users/{user_id}/subscriptions/{subscription_id}`

Delete subscription by user_id and subscription_id

POST `/notification-manager-oas/api/v1/users/{user_id}/subscriptions`

Create subscription to category

GET `/notification-manager-oas/api/v1/users/{user_id}/subscriptions`

Get Subscriptions by Userid

3.3.14 PingController

GET `/ping`

GET `/pingConsumer`

GET `/pingProvider`

GET `/pingUser`

3.3.15 Cost-controller

GET `/pricingManager/fee/getfee`

Get I3M fee

PUT `/pricingManager/fee/setfee`

Set I3M fee

3.3.16 Price-controller

GET `/pricingManager/price/checkformulaconfiguration`

Check formula and parameter consistency

GET `/pricingManager/price/getformulajsonconfiguration`

Get configuration using Json format

GET `/pricingManager/price/getprice`

Get the price of data

PUT `/pricingManager/price/setformulaconstant`

Set formula constant

PUT `/pricingManager/price/setformulajsonconfiguration`

Set configuration using Json format

PUT `/pricingManager/price/setformulaparameter`

Set formula parameter

PUT `/pricingManager/price/setformulawithdefaultconfiguration`

Set formula with default values for constants and parameters

3.3.17 RatingService

GET`/rating/api/agreements/{id}/isRated`

Check if an agreement is rated

GET`/rating/api/agreements/{id}/rating`

Get the rating object of a specified agreement

GET`/rating/api/consumers/{pk}/agreements`

Get the terminated agreements of the consumer

GET`/rating/api/consumers/{did}/ratings`

Get the ratings of the consumer

GET`/rating/api/providers/{pk}/agreements`

Get the terminated agreements of the provider

GET`/rating/api/providers/{did}/ratings`

Get the ratings of the provider

GET`/rating/api/providers/{did}/totalRating`

Get the average rating of the provider

GET`/rating/api/questions`

Get all the questions

POST`/rating/api/ratings/{id}/respond`

Respond to a rating object

PUT`/rating/api/ratings/{id}`

Edit an existing Rating

GET`/rating/api/ratings/{id}`

Get a single rating.

DELETE`/rating/api/ratings/{id}`

Delete a single rating.

POST`/rating/api/ratings`

Create a new rating

GET`/rating/api/ratings`

Get all the ratings

3.3.18 Agreement

GET`/sc-manager-oas/check_active_agreements`

Check active agreements

POST`/sc-manager-oas/check_agreements_by_consumer`

Check agreements by consumer

GET`/sc-manager-oas/check_agreements_by_data_offering/{offering_id}`

Check agreements by data offering

POST`/sc-manager-oas/check_agreements_by_provider`

Check agreements by provider

POST/sc-manager-oas/create_agreement_raw_transaction/{sender_address}

Create agreement

POST/sc-manager-oas/deploy_signed_transaction

Deploy signed transaction

PUT/sc-manager-oas/enforce_penalty

Enforce penalty

POST/sc-manager-oas/evaluate_signed_resolution

Verify a signed resolution

GET/sc-manager-oas/get_agreement/{agreement_id}

Get agreement

GET/sc-manager-oas/get_pricing_model/{agreement_id}

Get agreement's pricing model

GET/sc-manager-oas/get_state/{agreement_id}

Get the state of the agreement

POST/sc-manager-oas/propose_penalty

Choose penalty

GET/sc-manager-oas/retrieve_agreements/{consumer_public_key}

Retrieve the active agreements, which start date is reached, based on consumer public key

GET/sc-manager-oas/template/{offering_id}

Request template with static and dynamic parameters

PUT/sc-manager-oas/terminate

Terminate agreement

3.3.19 Explicit user consent

GET/sc-manager-oas/check_consent_status/{dataOfferingId}

Check consent status

POST/sc-manager-oas/deploy_consent_signed_transaction

Deploy consent signed transaction

POST/sc-manager-oas/give_consent

Give consent

PUT/sc-manager-oas/revoke_consent

Revoke consent

3.3.20 Registration-offering

GET/semantic-engine/api/registration/ActiveOfferingByCategory/{category}

Get a registered active data offerings by category

GET/semantic-engine/api/registration/ActiveOfferingByProvider/{id}/providerId

Get a registered active data offering by provider

GET/semantic-engine/api/registration/categories-list

Get a list of all categories

GET/semantic-engine/api/registration/contract-parameter/{offeringId}/offeringId

Get contract parameters by offering id

POST/semantic-engine/api/registration/data-offering

Register a data offering

DELETE/semantic-engine/api/registration/delete-offering/{id}

Delete a data offering

GET/semantic-engine/api/registration/federated-activeOffering/{id}/providerId

Get a registered active data offering by provider

GET/semantic-engine/api/registration/federated-activeOffering/{category}

Get a registered active federated data offering by category

GET/semantic-engine/api/registration/federated-contract-parameter/{id}/offeringId

Get contract parameters by offering id in federated search

GET/semantic-engine/api/registration/federated-offering/getActiveOfferingByText/{text}/text

Get a registered data offering by text/keyword

GET/semantic-engine/api/registration/federated-offering/textSearch/text/{text}

Get a registered data offering by text/keyword in federated search

GET/semantic-engine/api/registration/federated-offering/{id}/offeringId

Get a registered data offering by offering id

GET/semantic-engine/api/registration/federated-offering/{category}

Get a registered data offering by category

GET/semantic-engine/api/registration/federated-offerings-list/on-Active

Get a list of offerings for active in federated search

GET/semantic-engine/api/registration/federated-offerings-list/on-SharedNetwork

Get a list of offerings for shared status in federated search

GET/semantic-engine/api/registration/federated-offerings-list

Get a list of offerings

GET/semantic-engine/api/registration/federated-providers-list

Get a list of providers

GET`/semantic-engine/api/registration/getActiveOfferingByText/{text}/text`

Get a registered data offering by text/keyword

GET`/semantic-engine/api/registration/getOfferingByActiveAndShareDataWithThirdParty/{active}/{shareDataWithThirdParty}`

Get a registered data offering by active and sharedWithThirdParty status

GET`/semantic-engine/api/registration/getOfferingBySharedAndTransferableAndFreePrice/{shared}/{transfer}/{freePrice}`

Get a registered data offering by shared and transferable and FreePrice status

GET`/semantic-engine/api/registration/offering/ByTitleAndPricingModelName/{dataOfferingTitle}/{pricingModelName}`

Get a registered data offering by title and pricing model name

GET`/semantic-engine/api/registration/offering/offering-template`

Download offering template

GET`/semantic-engine/api/registration/offering/provider/{providerId}`

Get data provider by providerId

GET`/semantic-engine/api/registration/offering/{id}/offeringId`

Get a registered data offering by offering id

GET`/semantic-engine/api/registration/offering/{id}/providerId`

Get a registered data offering by provider id

GET`/semantic-engine/api/registration/offering/{category}`

Get a registered data offering by category

GET`/semantic-engine/api/registration/offerings`

Get total offering and its list

GET`/semantic-engine/api/registration/offerings-list/on-SharedNetwork`

Get a list of offerings for shared status

GET`/semantic-engine/api/registration/offerings-list/on-active`

Get a list of offerings for active

GET`/semantic-engine/api/registration/offerings-list`

Get a list of offerings

DELETE`/semantic-engine/api/registration/provider/{providerId}/delete`

Delete a data provider by providerId

GET`/semantic-engine/api/registration/providers/{category}/category`

Get a list of providers by category

GET`/semantic-engine/api/registration/providers-list`

Get a list of providers

GET`/semantic-engine/api/registration/textSearch/text/{text}`

Get a registered data offering by text/keyword

PUT`/semantic-engine/api/registration/update-offering`

Update already registered offering info

POST`/semantic-engine/api/registration`

Register provider info

3.3.21 TokenizerController

POST`/tokenization/api/v1/operations/clearing`

Retrieve the transaction object to start the marketplace clearing operation

POST`/tokenization/api/v1/operations/exchange-in`

Retrieve the transaction object to perform an exchangeIn.

POST`/tokenization/api/v1/operations/exchange-out`

Retrieve the transaction object to perform an exchangeOut

POST`/tokenization/api/v1/operations/fee-payment`

Generate the fee payment transaction object

POST`/tokenization/api/v1/operations/set-paid`

Generate the payment transaction object

GET`/tokenization/api/v1/operations`

Get list of operations

GET`/tokenization/api/v1/treasury/balances/{address}`

Get the balance for a specific account

POST`/tokenization/api/v1/treasury/community-wallet`

Alter the community wallet address and the related community fee

GET`/tokenization/api/v1/treasury/marketplaces/{address}`

Get the index of a registered marketplace

POST`/tokenization/api/v1/treasury/marketplaces`

Register a marketplace

GET`/tokenization/api/v1/treasury/token-transfers/{transferId}`

Get the token transfer given a TransferId

POST`/tokenization/api/v1/treasury/transactions/deploy-signed-transaction`

Deploy a signed transaction

GET`/tokenization/api/v1/treasury/transactions/{transactionHash}`

Get the receipt of a transaction given a TransactionHash

3.3.22 Credential

GET`/verifiableCredentials/release2/vc/credential/issue/{credential}/callbackUrl/{callbackUrl}`

Create a new credential with Veramo framework and store it in the wallet (full flow)

GET`/verifiableCredentials/release2/vc/credential/issue/{did}/{credential}`

Generate a new credential with Veramo framework for a specific DID

POST`/verifiableCredentials/release2/vc/credential/revoke`

Revoke a credential by JWT

POST`/verifiableCredentials/release2/vc/credential/verify`

Verify a credential by JWT

GET`/verifiableCredentials/release2/vc/credential`

Get the credential list

3.3.23 Issuer

GET`/verifiableCredentials/release2/vc/issuer/subscribe`

Subscribe this issuer in the i3-MARKET trusted issuers list

GET`/verifiableCredentials/release2/vc/issuer/unsubscribe`

Unsubscribe this issuer from the i3-MARKET trusted issuers list

GET`/verifiableCredentials/release2/vc/issuer/verify`

Verify the subscription status of the issuer