
SUNFISH Platform Documentation Documentation

Release 0.9

SUNFISH Consortium

Nov 06, 2017

1	Federation-as-a-Service	3
1.1	Operational Phases	4
2	SUNFISH Platform in a nutshell	5
3	Setting-up a SUNFISH Cloud Federation	7
3.1	Data Security Enforcement Infrastructure	7
4	SUNFISH Use Case Demonstrator	13
5	API	15
5.1	SUNFISH Policy Administration Point (PAP) API	15
5.2	SUNFISH Policy Decision Point (PDP) API	17
5.3	SUNFISH Policy Enforcement Point (PEP) API	19
5.4	SUNFISH Policy Information Point (PIP) API	22
5.5	SUNFISH Policy Retrieval Point (PRP) API	23
5.6	SUNFISH Intelligent Workload Manager (IWM) API	26
6	Registry Interface	97
6.1	Instructions for Registry Interface Deployment and Development	97
7	Registry	101
7.1	Instructions for deploying chaincode	101
8	FRM	103
8.1	Instructions for deploying FRM	103
9	IWM	107
9.1	Overview of Intelligent Workload Manager	107
9.2	Screenshots	107
9.3	Instructions for deploying IWM	110
10	Registry Interface	113
11	Registry	115

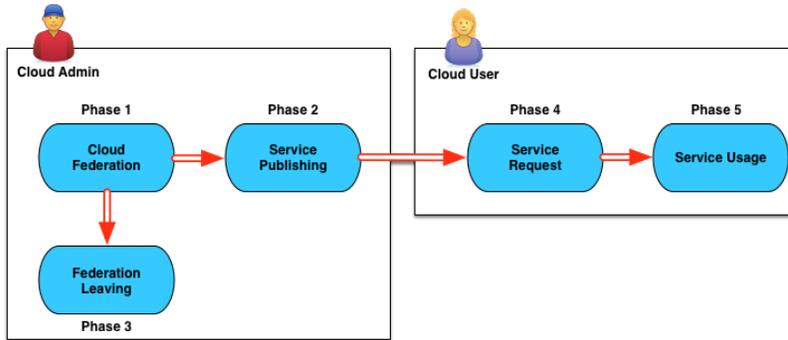
The SUNFISH Platform is software platform enabling Federation-as-a-Service (FaaS), a new and innovative Cloud Federation solution conceived and designed by the EU SUNFISH Project.

CHAPTER 1

Federation-as-a-Service

This is a page for FaaS

1.1 Operational Phases



CHAPTER 2

SUNFISH Platform in a nutshell

This is a page for the SUNFISH Platform

Setting-up a SUNFISH Cloud Federation

3.1 Data Security Enforcement Infrastructure

The SUNFISH data security enforcement infrastructure is responsible for regulating and securing access to services in a federated cloud environment.

It consists of the following components:

- The **Policy Enforcement Gateway** (PEG) responsible for enforcing decision regarding whether access to a resource is granted or not and which obligations need to be observed (if any). This component serves as the main entry point to a service protected by the SUNFISH DS enforcement infrastructure.
- An accompanying **Proxy** enabling the non-SUNFISH-aware applications to utilise the benefits of the SUNFISH platform.
- A **Policy Decision Point** (PDP) evaluating a decision request, indicating whether access to a service should be granted or not. In addition, Obligations such as *data masking*, for example can be part of the decision.
- **Policy Information Points** (PIPs) delivering information to the PEG and the PDP to enhance decision requests.
- A **Policy Administration Point** (PAP) providing an interface for administration data security policies.
- The **Registry Interface** responsible for storing, managing and delivering policies to the PDP for evaluation. Setup an operation of the RI is described separately, since it is operated independently of the other components.
- A **Masking Service** providing *data masking* capabilities to the PEG. Like the RI, the masking service is operated independently of the other components and therefore also discussed separately.

Typically, the enforcement infrastructure will be deployed among different *tenants*. The following setup instructions are based on a minimal example consisting of one *infrastructure tenant* and a *service tenant*.

The infrastructure tenant will typically house the PDP, any number of PIPs and the PRP. The service tenant hosts the actual service to be protected by the enforcement infrastructure as well as the PEG, any number of PIPs and the proxy to maintain backwards compatibility to non-SUNFISH-aware clients. As outlined initially, the PEG located at the service tenant serves as the main entry point, responding to incoming requests, which can either be submitted directly

to the PEG, or through the proxy. In case the requests was directed to the proxy, responses are also interpreted by the proxy and reduced in such a way that non-SUNFISH-aware applications are able to interpret it correctly (albeit losing expressibility in the process).

In-depth descriptions on how to set up a service tenant and an infrastructure tenant are available. These include step-by-step instructions to deploy the enforcement infrastructure on existing Java application servers. In addition, a streamlined, deployment-script-based setup as well as an automated, easy-to-use, self-contained, two-step, docker-based setup is provided for jump-starting a SUNFISH deployment. The referred scripts and configuration files are located at <https://github.com/sunfish-prj/Data-Security/tree/master/ds/doc/install>. The sub-folder `service` contains necessary files for the service tenant deployment, the `infrastructure` folder for the infrastructure tenant deployment respectively. The `docker` folder again contains the same structure, but for the dockerized setup.

3.1.1 Setting-Up a Service Tenant

It is assumed that a service is already running in the service tenant.

Step-By-Step Setup

Although not recommended, the SUNFISH data security enforcement infrastructure can be deployed following the succeeding steps. However, depending on the deployment use case, additional steps or adaptations to either configuration or system components may still be necessary. For demonstration purposes a two tenant setup is assumed. The sample configuration ships with precompiled Tomcat applications, which can be found in the respective `webapps` directory of either tenant. Additionally, a sample configuration for the service tenant can be found in the respective `conf` directory. To deploy the service tenant follow these steps:

- Copy the content of the provided `./tomcat/webapps` directory to `CATALINA_HOME/webapps` directory
- Copy the content of the provided `./tomcat/conf` directory to `CATALINA_HOME/conf` directory
- Copy the content of the provided `./proxy/` directory to any desired directory (referred to as `PROXY_HOME`)

In a divergent deployment scenario, the respective configurations of the SUNFISH components and the SUNFISH proxy need to be adapted individually. To start the SUNFISH data security enforcement infrastructure simply start your local Tomcat instance and execute the `start.sh` script, located in your `PROXY_HOME` directory.

Using the Deployment Script

The attached deployment script is an easy way to automatically setup a service tenant. For this, the following two steps are necessary:

- Adapt the configuration if necessary (`config.sh`)
- Execute the deployment script (`./deploy.sh`)

The deployment script will automatically create all necessary resources and copy them to their designated destination. No further steps are necessary. To start the SUNFISH data security enforcement infrastructure simply start your local Tomcat instance and execute the `start.sh` script, located in your `PROXY_HOME` directory.

Configuration Directives

The infrastructure tenant features several configuration options before installation. The following parameters are available:

- `TOMCAT_PORT`: Defines the port of the local Tomcat instance

- **CATALINA_HOME**: Defines the home directory of the local Tomcat instance (e.g. (/usr/local/tomcat/))
- **PEP_URL_PDP**: Defines the URL of the designated *PDP* for the *PEP*
- **PEP_URLS_PIPS**: Defines the possible *PIPs* available to the *PEP*. Multiple URLs can be specified, separated by a comma
- **PEP_ZONE**: Defines the tenant name the *PEP* is located in
- **PEP_URL_DM**: Specifies the URL to the data masking service
- **PEP_URL_ANON**: Specifies the URL to the anonymisation service
- **PIP_DATABASE**: Defines possible database values for the *PIP*. Each setting consists of a key and a value. In general three entries are necessary in order to setup a new service inside the service tenant:
 - **Host for ID**: Assign a hostname to a specific service. The key must be in the format `host.<service_id>`. The value represents a single URL to the designated service.
 - **Tenant for ID**: Assign a service to a specific tenant. The key must be in the format `zone.<service_id>`. The value defines the tenant the service is located at.
 - **PEP for Tenant**: Assign a *PEP* to a specific tenant. The key must be in the format `pep.<tenant name>`. The value represents a single URL to the designated *PEP*.
- **PROXY_HOME**: Defines the home directory of the SUNFISH proxy (e.g. (/usr/local/proxy/))
- **PROXY_IP**: Defines the IP address the SUNFISH Proxy will run on
- **PROXY_PORT**: Defines the port the SUNFISH Proxy will listen to
- **PROXY_PEP[<service_id>]**: Defines the URL of the *PEP* guarding the service `<service_id>` for the SUNFISH Proxy. Multiple services can be defined; should match the service IDs in the PIP database.

Dockerised Setup

The docker-based deployment also features a configuration file containing essentially the same (at this point mostly self-explanatory) directives and a deployment script. This script has to be invoked after editing the configuration file just as it is the case for the regular deployment-script-based setup.

To actually deploy the docker container, once the configuration file has been adapted, the following steps need to be performed:

- Download the service docker container (`tenant.tar`) from the *Releases* tab in the GitHub repository and copy it to `install/docker/tenant/`
- The preconfigured docker container `tenant.tar` needs to be loaded: `docker load -i tenant.tar`
- The deployment script has to be executed (`./deploy.sh`)

This should start a docker container, inside which the proxy is running on `PROXY_PORT` and the PEG and the PIP are running as web applications on a Tomcat server on `TOMCAT_PORT`. Both ports are mapped to their respective counterparts on the host machine.

Setting-Up a Service

To add a new service to the SUNFISH data security enforcement infrastructure, the following steps are necessary:

- Add a *host* for the *service id* to the configuration file `config.sh` or, if the SUNFISH tenant has already been setup, to the configuration file located in `CATALINA_HOME/conf/sunfish/pip/database/pip_database.config`

- Add a *tenant* for the *service id* to the configuration file `config.sh` or, if the SUNFISH tenant has already been setup, to the configuration file located in `CATALINA_HOME/conf/sunfish/pip/database/pip_database.config`. It is important to note that this step needs to be performed for all operational tenants, as long as the PIP database containing the service configuration is not replicated between all tenants.
- Add a *pep* for the *tenant* of the *service* to the configuration file `config.sh` or, if the SUNFISH tenant has already been setup, to the configuration file located in `CATALINA_HOME/conf/sunfish/pip/database/pip_database.config`. It is important to note that this step needs to be performed for all operational tenants, as long as the PIP database containing the service configuration is not replicated between all tenants.
- Restart your local Service Tenant Tomcat in order to apply the changes

Adding Policies

By default, any deployed service requires dedicated policies in order for the SUNFISH data security enforcement infrastructure to work. Policies can be added via the *PAP* and the defined **API** (see also Chapter *SUNFISH Policy Administration Point (PAP) API*). A sample policy, allowing access to a defined service is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:ns2="urn:sunfish
↳ " PolicyId="urn:sunfish:policy:demo-proxy-https" Version="1.0" RuleCombiningAlgId=
↳ "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
  <Description>Demo Permit-All Policy </Description>
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string
↳ ">129.27.142.49</AttributeValue>
          <AttributeDesignator Category="urn:sunfish:attribute-
↳ category:service" AttributeId="urn:sunfish:attribute:id" DataType="http://www.w3.
↳ org/2001/XMLSchema#string" MustBePresent="true"/>
        </Match>
        <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:string-starts-
↳ with">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string
↳ ">/demo-app/demo/</AttributeValue>
          <AttributeDesignator Category="urn:sunfish:attribute-
↳ category:response" AttributeId="urn:sunfish:attribute:request:path" DataType="http://
↳ www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string
↳ ">129.27.142.49</AttributeValue>
    <AttributeDesignator Category="urn:sunfish:attribute-
↳ category:service" AttributeId="urn:sunfish:attribute:id" DataType="http://www.w3.
↳ org/2001/XMLSchema#string" MustBePresent="true"/>
  </Match>
  <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:string-starts-
↳ with">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string
↳ ">/demo-app/demo/</AttributeValue>
    <AttributeDesignator Category="urn:sunfish:attribute-
↳ category:request" AttributeId="urn:sunfish:attribute:request:path" DataType="http://
↳ www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
  </Match>
</Policy>
```

```

        </AllOf>
    </AnyOf>
</Target>
<Rule RuleId="urn:sunfish:rule:permit" Effect="Permit">
    <Target/>
</Rule>
</Policy>

```

3.1.2 Setting-Up an Infrastructure Tenant

Step-By-Step Setup

Although not recommended, the SUNFISH data security enforcement infrastructure can be deployed following the succeeding steps. However, depending on the deployment use case, additional steps or adaptations to either configuration or system components may still be necessary. For demonstration purposes a two tenant setup is assumed. The sample configuration ships with precompiled Tomcat applications, which can be found in the respective `webapps` directory of either tenant. Additionally, a sample configuration for the infrastructure tenant can be found in the respective `conf` directory. To deploy the service tenant follow these steps:

- Copy the content of the provided `webapps` directory to `CATALINA_HOME/webapps` directory
- Copy the content of the provided `conf` directory to `CATALINA_HOME/conf` directory

In a divergent deployment scenario, the respective configurations of the SUNFISH components need to be adapted individually. To start the SUNFISH data security enforcement infrastructure simply start your local Tomcat instance.

Using the Deployment Script

The attached deployment script is an easy way to automatically setup an infrastructure tenant. For this, the following two steps are necessary:

- Adapt the configuration if necessary (`config.sh`)
- Execute the deployment script (`./deploy.sh`)

The deployment script will automatically create all necessary resources and copy them to their designated destination. No further steps are necessary. To start the SUNFISH data security enforcement infrastructure simply start your local Tomcat instance.

Configuration Directives

The infrastructure tenant features several configuration options before installation. The following parameters are available:

- `TOMCAT_PORT`: Defines the port of the local Tomcat instance
- `CATALINA_HOME`: Defines the home directory of the local Tomcat instance (e.g. `/usr/local/tomcat/`)
- `PAP_URL_RI`: Defines the URL of the designated **Registry Interface** for the *PAP*
- `PDP_URLS_PRPS`: Defines the possible *PRPs* available to the *PDP*. Multiple URLs can be specified, separated by a comma
- `PDP_URLS_PIPS`: Defines the possible *PIPs* available to the *PDP*. Multiple URLs can be specified, separated by a comma
- `PRP_URL_RI`: Defines the URL of the designated **Registry Interface** for the *PRP*

- `PIP_DATABASE`: Defines possible database values for the *PIP*. Each setting consists of a key and a value. In general, no additional values are necessary for the *PIP* in the infrastructure tenant.

Dockerised Setup

The docker-based deployment also features a configuration file containing essentially the same (at this point mostly self-explanatory) directives and a deployment script. This script has to be invoked after editing the configuration file just as it is the case for the regular deployment-script-based setup.

To actually deploy the docker container, once the configuration file has been adapted, the following steps need to be performed:

- Download the infrastructure docker container (`infrastructure.tar`) from the *Releases* tab in the GitHub repository and copy it to `install/docker/infrastructure/`
- The preconfigured docker container `infrastructure.tar` needs to be loaded: `docker load -i infrastructure.tar`
- The deployment script has to be executed (`./deploy.sh`)

This should start a docker container, inside which the PDP, the PRP and the PIP are running as web applications on a Tomcat server on `TOMCAT_PORT` which is mapped to the same port on the host machine.

CHAPTER 4

SUNFISH Use Case Demonstrator

This is the page for Use Case Demonstrator

This is a page for API

5.1 SUNFISH Policy Administration Point (PAP) API

The PAP interface follows a straight forward REST interface, as it requires bare access to the policy storage.

Version: 1.0.0

Contact information:

Alexander Marsalek

alexander.marsalek@a-sit.at

5.1.1 /v1/policies

GET

Summary: This endpoint is used by entities interfacing with the PAP to retrieve policies

Description:

Parameters

Name	Lo- cated in	Description	Re- quired	Schema
SUNFISH- issuer	header	References the entity that issued the request. This field may include the data that confirms the authentication of source entity and its authentication level.	Yes	string

Responses

Code	Description	Schema
200	The body of the response contains the requested policies according to the schema defined in Listing 3. The response result set only contains a certain amount of entries. Pagination is done using the Web Linking approach according to RFC5988. A Link header is included in the response pointing to the next resultset: Link: ;rel=next">https://%3Chost/pap/api/v1/policies/?page=2>;rel=next The possible “rel” values are “next” pointing to the next result-set. Pagination URLs are not allowed to be constructed manually.	string
400	Invalid request	
403	The requestor is not allowed to perform this operation	
404	No policies matching the specified request were found	

POST

Summary: This endpoint is used by entities interfacing with the PAP to add a policy

Description:

Parameters

Name	Lo- cated in	Description	Re- quired	Schema
body	body	The body of the request contains a to be added policy according to the schema in Listing 1.	Yes	string
SUNFISH- issuer	header	References the entity that issued the request. This field may include the data that confirms the authentication of source entity and its authentication level.	Yes	string

Responses

Code	Description	Schema
200	Created successful	string
400	Invalid request	
403	The requestor is not allowed to perform this operation	
409	The policy exists already	

5.1.2 /v1/policies/{id}/{version}

DELETE

Summary: This endpoint is used by entities to remove policies

Description:

Parameters

Name	Lo- cated in	Description	Re- quired	Schema
id	path	Id of the policy to delete	Yes	string
version	path	Specifies the version of the policy to be deleted	Yes	string
SUNFISH- issuer	header	References the entity that issued the request. This field may include the data that confirms the authentication of source entity and its authentication level.	Yes	string

Responses

Code	Description	Schema
200	Deleted successful	string
400	Invalid request	
403	The requestor is not allowed to perform this operation	
404	Policy not found	

5.2 SUNFISH Policy Decision Point (PDP) API

This API is primarily used by adjacent PEPs to issue authorization requests for intra-zone and cross-zone interactions. In this specification we partially rely on the REST profile suggested by the OASIS XACML Standard

Version: 1.0.0

Contact information:

Bernd Prünster
bernd.pruenster@a-sit.at

5.2.1 /v1

GET

Summary: API entry point. This point is used to identify functionality and endpoints provided by PDP.

Description:

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	The response contains a resource with link relation http://docs.oasis-open.org/ns/xacml/relation/pdp and a valid URL.

5.2.2 /v1/verifyServicePolicy

POST

Summary: Verify a service policy

Description:

Parameters

Name	Located in	Description	Required	Schema
SUNFISH-signature	header	This field is used to provide integrity and authenticity of messages.	No	string
body	body	Contains XACML-format ted policy for PDP to perform verification.	Yes	string

Responses

Code	Description	Schema
200	Contains information about the verification result.	<i>VerifyPolicyResult</i>
400	Invalid request	
404	The requestor is not allowed	

5.2.3 /v1/verifyServicePolicySet**POST**

Summary: Verify a service policy set

Description:

Parameters

Name	Located in	Description	Required	Schema
SUNFISH-signature	header	This field is used to provide integrity and authenticity of messages.	No	string
body	body	Contains XACML-format ted policy set for PDP to perform verification .	Yes	string

Responses

Code	Description	Schema
200	Contains information about the verification result.	<i>VerifyPolicyResult</i>
400	Invalid request	
404	The requestor is not allowed	

5.2.4 /v1/authorization**POST**

Summary: This endpoint is used by PEPs to issue authorization decision requests to PDP. These requests are sent using POST method. Inputs to this endpoint are parameters that describe access requests initiated by entities interacting through the calling PEP. Additionally, this request contains other contextual parameters that can be used by PDP to evaluate request.

Description:

Parameters

Name	Located in	Description	Required	Schema
SUNFISH-signature	header	This field is used to provide integrity and authenticity of messages.	No	string
body	body	Contains XACML-format ted (or other) request with all relevant data and attributes necessary for PDP to perform authorization decision.	Yes	string

Responses

Code	Description	Schema
200	Contains complete XACML-format ted answer. Body can include additional answer that deals with activity context, if requested.	string
400	Invalid XACML request	
404	Requestor is not allowed to perform the request	

5.2.5 Models

VerifyPolicyResult

Name	Type	Description	Required
status	string	Indicates the status of the verification operation.	No
description	string	Description, containing detailed information about the requested operation.	No
statusCode	integer	Status code of the operation.	No

5.3 SUNFISH Policy Enforcement Point (PEP) API

The interactions executed inside one zone are checked by and enforced in the scope of a PEP assigned for that zone. The approach is similar for the zones that consist of geographically dispersed locations: each PEP (or sub-PEP) is responsible for its geographical unit or layer. Being the single point of contact of a zone, the PEP is primarily responsible for checking incoming and outgoing requests. In the second instance, depending on security settings and application requirements, PEP might serve as an inter-zone communication gateway, as well.

Version: 1.0.0

Contact information:

Dominik Ziegler
dominik.ziegler@a-sit.at

5.3.1 /v1/request

POST

Summary: This endpoint is used by PEPs to POST new requests to other PEPs. Inputs to this endpoint are contextual parameters that establish the request, application and target specific settings. The response of this action is the data record that contains request id and data structure describing status parameters or other PEP requirements.

Description:

Parameters

Responses

Code	Description	Schema
200	Body of the original request	[byte]

5.3.2 /v1/app-request

POST

Summary: Applications can POST new requests to this endpoint. Inputs to this endpoint are contextual parameters that establish the request, application and target specific settings. For this specification, the applications rely on common SUNFISH functionalities and components. The response of this action is the original response of the target service (synchronous use case).

Description:

Parameters

Name	Located in	Description	Required	Schema
body	body	Body of the original request	No	[byte]
SUNFISH-issuer	header	References the application that issued the request. This field may include the data required to perform application authentication, in the form of authentication token.	No	string
SUNFISH-service	header	Machine-readable description of endpoint including at least an identifier of the service. With the service id, the PEP can resolve other required attributes.	Yes	string
SUNFISH-request	header	Machine-readable description of the target endpoint and request data. The PEP at least requires the parameters method, port, path and protocol. If additional attributes are registered in the SUNFISH federation, the PEP can retrieve these attributes from a corresponding PIP. Furthermore, this field may include validity constraints on a request (not-valid-before, not-valid-after).	Yes	string
SUNFISH-request-parameters	header	The parameters related to the request, including its priority, SLA requirements, callback URI. This field includes other request meta-data that may extend or override the definitions provided in centralized administrative console. These include API request type, application-specific policies or obligations to be	No	string
5.3. SUNFISH Policy Enforcement Point (PEP)				21

Responses

Code	Description	Schema
200	The same response as provided by the target service	[byte]

5.4 SUNFISH Policy Information Point (PIP) API

The PIP is generally defined as “the system entity that acts as source of attribute values

Version: 1.0.0

Contact information:

Dominik Ziegler
dominik.ziegler@a-sit.at

5.4.1 /v1/collect

GET

Summary: This endpoint is used to retrieve collection of all available attribute ids

Description:**Parameters**

Name	Lo- cated in	Description	Re- quired	Schema
SUNFISH- issuer	header	References the entity that issued the request. This field includes the data that confirms the authentication of source entity and its authentication level.	Yes	string

Responses

Code	Description	Schema
200	Contains collection of attribute designators ids according to the attribute designator set schema.	string
400	Invalid request	
403	The requestor is not allowed	

5.4.2 /v1/request

POST

Summary: This endpoint is used to retrieve additional attributes

Description:**Parameters**

Name	Lo-cated in	Description	Re-quired	Schema
body	body	Contains the requested attributes and the request context as issued by the PEP. If multiple PIPs are involved, the PIP always receive the most recent request context.	No	string
SUNFISH-header issuer		References the entity that issued the request. This field includes the data that confirms the authenticati on of source entity and its authenticati on level.	Yes	string

Responses

Code	Description	Schema
200	The request context was enhanced with all or some of the requested attributes.	string
400	Invalid request	
403	The requestor is not allowed	
404	This PIP does not provide any of the requested attributes.	

5.5 SUNFISH Policy Retrieval Point (PRP) API

The PRP is not included in the OASIS XACML standard, but provides another abstraction level of the PAP

Version: 1.0.0

Contact information:

Alexander Marsalek
alexander.marsalek@a-sit.at

5.5.1 /v1/collect

POST

Summary: This endpoint is used by PDPs to retrieve collection of policies for specified decision request.

Description:

Parameters

Name	Lo-cated in	Description	Re-quired	Schema
body	body	Contains the request formatted according to the XACML decision request language with all relevant data and attributes necessary for the PRP to identify the relevant policies.	Yes	string

Responses

Code	Description	Schema
200	Contains single policy set where all policies are contained or references according to the XACML policy set schema.	string
400	Invalid request	
404	No policies matching the specified request were found	

5.5.2 /v1/policyset/{id}/{version}

GET

Summary: This endpoint is used to retrieve policy by id. Optionally version can be specified.

Description:

Parameters

Name	Located in	Description	Required	Schema
root-PolicySet	query	true	false Defines if root policy-set or re-usable policies- set should be returned.	Yes
id	path	Specifies the id of the policy set to be returned in the response.	Yes	string
version	path	Specifies the version of the policy set to be returned in the response. If no version is specified the newest policy set will be returned.	Yes	string

Responses

Code	Description	Schema
200	Contains the requested policy set	string
400	Invalid request	
403	The requestor is not allowed to retrieve this policy	
404	The policy set with the specified id was not found	

5.5.3 /v1/policy/{id}

GET

Summary: This endpoint is used to retrieve policy by id. Optionally version can be specified.

Description:

Parameters

Name	Located in	Description	Required	Schema
rootPolicy	query	true	false Defines if root policy or re-usable policy should be returned.	Yes
id	path	Specifies the id of the policy to be returned in the response.	Yes	string

Responses

Code	Description	Schema
200	Contains the requested policy set	string
400	Invalid request	
403	The requestor is not allowed to retrieve this policy	
404	The policy set with the specified id was not found	

5.5.4 /v1/policyset/{id}

GET

Summary: This endpoint is used to retrieve policy by id. Optionally version can be specified.

Description:

Parameters

Name	Located in	Description	Required	Schema
root-Policy-Set	query	true	false Defines if root policy-set or re-usable policies- set should be returned.	Yes
id	path	Specifies the id of the policy set to be returned in the response.	Yes	string

Responses

Code	Description	Schema
200	Contains the requested policy set	string
400	Invalid request	
403	The requestor is not allowed to retrieve this policy	
404	The policy set with the specified id was not found	

5.5.5 /v1/policy/{id}/{version}

GET

Summary: This endpoint is used to retrieve policy by id. Optionally version can be specified.

Description:

Parameters

Name	Located in	Description	Required	Schema
rootPolicy	query	true	false Defines if root policy or re-usable policy should be returned.	Yes
id	path	Specifies the id of the policy to be returned in the response.	Yes	string
version	path	Specifies the version of the policy to be returned in the response. If no version is specified the newest policy will be returned.	Yes	string

Responses

Code	Description	Schema
200	Contains the requested policy set	string
400	Invalid request	
403	The requestor is not allowed to retrieve this policy	
404	The policy set with the specified id was not found	

5.6 SUNFISH Intelligent Workload Manager (IWM) API

IWM provides lifecycle management for virtual resources in a multi-cloud multi-tenant environment. It also provides optimized planner for the target infrastructure (costs, tags, etc). Functionality is implemented on top of the Waldur hybrid cloud broker.

Version: 1.0.0

Contact information:

Ilja Livenson
ilja.livenson@gmail.com

5.6.1

PUT /api/openstacktenant-snapshots/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/openstacktenant-snapshots/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/openstacktenant-snapshots/{uuid}/

- **Consumes:** [u'application/json']
-

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/openstacktenant-snapshots/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

200 -

GET /api/customers/{uuid}/users/

A list of users connected to the customer

- **Description:** A list of users connected to the customer

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/project-permissions/

- Projects are connected to customers, whereas the project may belong to one customer only,

- **Description:** - Projects are connected to customers, whereas the project may belong to one customer only, and the customer may have multiple projects. - Projects are connected to services, whereas the project may contain multiple services, and the service may belong to multiple projects. - Staff members can list all available projects of any customer and create new projects. - Customer owners can list all projects that belong to any of the customers they own. Customer owners can also create projects for the customers they own. - Project administrators can list all the projects they are administrators in. - Project managers can list all the projects they are managers in.

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/project-permissions/

Project permissions expresses connection of user to a project.

- **Description:** Project permissions expresses connection of user to a project. User may have either project manager or system administrator permission in the project. Use `*/api/project-permissions/*` endpoint to maintain project permissions.

Note that project permissions can be viewed and modified only by customer owners and staff users.

To list all visible permissions, run a `**GET**` query against a list. Response will contain a list of project users and their brief data.

To add a new user to the project, `**POST**` a new relationship to `*/api/project-permissions/*` endpoint specifying project, user and the role of the user ('admin' or 'manager');

.. code-block:: http

```
POST /api/project-permissions/ HTTP/1.1 Accept: application/json Authorization: Token
95a688962bf68678fd4c8cec4d138ddd9493c93b Host: example.com
```

```
{ "project": "http://example.com/api/projects/6c9b01c251c24174a6691a1f894fae31/", "role": "manager",
"user": "http://example.com/api/users/82cec6c8e0484e0ab1429412fe4194b7/" }
```

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
role	query		string
user	query		string
user_url	query		string
username	query		string
full_name	query		string
native_name	query		string
o	query		string
customer	query		string
project	query		string
project_url	query		string

Responses

200 -

POST /api/openstack-tenants/{uuid}/pull_floating_ips/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

201 -

PUT /api/hooks-email/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/hooks-email/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/hooks-email/{uuid}/

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/hooks-email/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/openstacktenant-snapshots/

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstacktenant-snapshots/**Parameters**

Name	Position	Description	Type
page	query		string
page_size	query		string
customer	query		string
customer_uuid	query		string
customer_name	query		string
customer_native_name	query		string
customer_abbreviation	query		string
project	query		string
project_uuid	query		string
project_name	query		string
service_uuid	query		string
service_name	query		string
service_settings_name	query		string
service_settings_uuid	query		string
name	query		string
description	query		string
state	query		string
uuid	query		string
tag	query		string
rtag	query		string
o	query		string
source_volume_uuid	query		string
source_volume	query		string
backup_uuid	query		string
backup	query		string

Responses

200 -

PUT /api/openstacktenant-service-project-link/{id}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path		string
data	body		

Responses

200 -

DELETE /api/openstacktenant-service-project-link/{id}/**Parameters**

Name	Position	Description	Type
id	path		string

Responses

204 -

PATCH /api/openstacktenant-service-project-link/{id}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path		string
data	body		

Responses

200 -

GET /api/openstacktenant-service-project-link/{id}/

To remove a link, issue **DELETE** to URL of the corresponding connection as staff user or customer owner.

- **Description:** To remove a link, issue **DELETE** to URL of the corresponding connection as staff user or customer owner.

Parameters

Name	Position	Description	Type
id	path		string

Responses

200 -

POST /api/openstack-tenants/{uuid}/create_floating_ip/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

201 -

PUT /api/project-permissions/{id}/

- Projects are connected to customers, whereas the project may belong to one customer only,

- **Description:** - Projects are connected to customers, whereas the project may belong to one customer only, and the customer may have multiple projects. - Projects are connected to services, whereas the project may contain multiple services, and the service may belong to multiple projects. - Staff members can list all available projects of any customer and create new projects. - Customer owners can list all projects that belong to any of the customers they own. Customer owners can also create projects for the customers they own. - Project administrators can list all the projects they are administrators in. - Project managers can list all the projects they are managers in.

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path		string
data	body		

Responses

200 -

DELETE /api/project-permissions/{id}/

To remove a user from a project, delete corresponding connection (**url** field). Successful deletion

- **Description:** To remove a user from a project, delete corresponding connection (**url** field). Successful deletion will return status code 204.

.. code-block:: http

```
DELETE /api/project-permissions/42/ HTTP/1.1 Authorization: Token
95a688962bf68678fd4c8cec4d138ddd9493c93b Host: example.com
```

Parameters

Name	Position	Description	Type
id	path		string

Responses

204 -

PATCH /api/project-permissions/{id}/

- Projects are connected to customers, whereas the project may belong to one customer only,

- **Description:** - Projects are connected to customers, whereas the project may belong to one customer only, and the customer may have multiple projects. - Projects are connected to services, whereas the project may contain multiple services, and the service may belong to multiple projects. - Staff members can list all available projects of any customer and create new projects. - Customer owners can list all projects that belong to any of the customers they own. Customer owners can also create projects for the customers they own. - Project administrators can list all the projects they are administrators in. - Project managers can list all the projects they are managers in.

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path		string
data	body		

Responses

200 -

GET /api/project-permissions/{id}/

- Projects are connected to customers, whereas the project may belong to one customer only,

- **Description:** - Projects are connected to customers, whereas the project may belong to one customer only, and the customer may have multiple projects. - Projects are connected to services, whereas the project may contain multiple services, and the service may belong to multiple projects. - Staff members can list all available projects of any customer and create new projects. - Customer owners can list all projects that belong to any of the customers they own. Customer owners can also create projects for the customers they own. - Project administrators can list all the projects they are administrators in. - Project managers can list all the projects they are managers in.

Parameters

Name	Position	Description	Type
id	path		string

Responses

200 -

GET /api/events/event_groups/

Returns a list of groups with event types.

- **Description:** Returns a list of groups with event types. Group is used in exclude_features query param.

Parameters

Name	Position	Description	Type

Responses

200 -

PUT /api/hooks-push/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/hooks-push/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/hooks-push/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/hooks-push/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

200 -

GET /api/events/scope_types/

Returns a list of scope types acceptable by events filter.

- **Description:** Returns a list of scope types acceptable by events filter.

Parameters

Name	Position	Description	Type
------	----------	-------------	------

Responses

200 -

PUT /api/openstack-floating-ips/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

200 -

DELETE /api/openstack-floating-ips/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/openstack-floating-ips/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses200 -

GET /api/openstack-floating-ips/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses200 -

POST /api/openstacktenant-volumes/{uuid}/extend/

Increase volume size

- **Description:** Increase volume size
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses201 -

PUT /api/openstack-subnets/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses200 -

DELETE /api/openstack-subnets/{uuid}/


```
{ "timestamp": "2015-07-14T12:12:56.000000", "message": "Customer ABC LLC has been updated.",
  "type": "customer_update_succeeded", "context": { "user_native_name": "Walter Lebrowski", "customer_contact_details": "", "user_username": "Walter", "user_uuid": "1c3323fc4ae44120b57ec40dea1be6e6", "customer_uuid": "4633bbbb0b3a4b91bffc0e18f853de85", "ip_address": "8.8.8.8", "user_full_name": "Walter Lebrowski", "customer_abbreviation": "ABC LLC", "customer_name": "ABC LLC" }, "levelname": "INFO" }
```

Note that context depends on event type.

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/hooks-web/

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
user	query		string
is_active	query		string
last_published	query		string
destination_url	query		string
content_type	query		string
author_uuid	query		string

Responses

200 -

POST /api/openstack-tenants/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstack-tenants/

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
customer	query		string
customer_uuid	query		string
customer_name	query		string
customer_native_name	query		string
customer_abbreviation	query		string
project	query		string
project_uuid	query		string
project_name	query		string
service_uuid	query		string
service_name	query		string
service_settings_name	query		string
service_settings_uuid	query		string
name	query		string
description	query		string
state	query		string
uuid	query		string
tag	query		string
rtag	query		string
o	query		string

Responses

200 -

POST /api/openstack-floating-ips/

Parameters

Name	Position	Description	Type
------	----------	-------------	------

Responses

201 -

GET /api/openstack-floating-ips/

To get a list of all available floating IPs, issue **GET** against `*/api/floating-ips/*`.

- **Description:** To get a list of all available floating IPs, issue **GET** against `*/api/floating-ips/*`. Floating IPs are read only. Each floating IP has fields: 'address', 'status'.

Status `*DOWN*` means that floating IP is not linked to a VM, status `*ACTIVE*` means that it is in use.

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
customer	query		string
customer_uuid	query		string
customer_name	query		string
customer_native_name	query		string
customer_abbreviation	query		string
project	query		string
project_uuid	query		string
project_name	query		string
service_uuid	query		string
service_name	query		string
service_settings_name	query		string
service_settings_uuid	query		string
name	query		string
description	query		string
state	query		string
uuid	query		string
tag	query		string
rtag	query		string
runtime_state	query		string
o	query		string
tenant_uuid	query		string
tenant	query		string

Responses

200 -

POST /api/openstack-subnets/

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstack-subnets/

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
customer	query		string
customer_uuid	query		string
customer_name	query		string
customer_native_name	query		string
customer_abbreviation	query		string
project	query		string
project_uuid	query		string
project_name	query		string
service_uuid	query		string
service_name	query		string
service_settings_name	query		string
service_settings_uuid	query		string
name	query		string
description	query		string
state	query		string
uuid	query		string
tag	query		string
rtag	query		string
o	query		string
tenant_uuid	query		string
tenant	query		string
network_uuid	query		string
network	query		string

Responses

200 -

POST /api/openstack/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstack/

To create a service, issue a **POST** to `*/api/openstack/*` as a customer owner.

- **Description:** To create a service, issue a **POST** to `*/api/openstack/*` as a customer owner.

You can create service based on shared service settings. Example:

.. code-block:: http

```
POST /api/openstack/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```


Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

POST /api/openstack-networks/{uuid}/pull/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

POST /api/projects/

A new project can be created by users with staff privilege (is_staff=True) or customer owners.

- **Description:** A new project can be created by users with staff privilege (is_staff=True) or customer owners. Project resource quota is optional. Example of a valid request:

.. code-block:: http

```
POST /api/projects/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "name": "Project A", "customer": "http://example.com/api/customers/6c9b01c251c24174a6691a1f894fae31/", }
```

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/projects/

To get a list of projects, run ****GET**** against `*/api/projects/*` as authenticated user.

- **Description:** To get a list of projects, run ****GET**** against `*/api/projects/*` as authenticated user. Here you can also check actual value for project quotas and project usage

Note that a user can only see connected projects:

- projects that the user owns as a customer
- projects where user has any role

Supported logic filters:

- ?can_manage - return a list of projects where current user is manager or a customer owner; - ?can_admin - return a list of projects where current user is admin;

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
name	query		string
customer	query		string
customer_name	query		string
customer_native_name	query		string
customer_abbreviation	query		string
description	query		string
created	query		string
o	query		string

Responses

200 -

POST /api/openstacktenant-instances/

OpenStack instance permissions

- **Description:** OpenStack instance permissions ^^^
 - Staff members can list all available VM instances in any service. - Customer owners can list all VM instances in all the services that belong to any of the customers they own. - Project administrators can list all VM instances, create new instances and start/stop/restart instances in all the services that are connected to any of the projects they are administrators in. - Project managers can list all VM instances in all the services that are connected to any of the projects they are managers in.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstacktenant-instances/

OpenStack instance permissions

- **Description:** OpenStack instance permissions ^^^
 - Staff members can list all available VM instances in any service. - Customer owners can list all VM instances in all the services that belong to any of the customers they own. - Project administrators can list all VM instances, create new instances and start/stop/restart instances in all the services that are connected to any of the projects they are administrators in. - Project managers can list all VM instances in all the services that are connected to any of the projects they are managers in.

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
customer	query		string
customer_uuid	query		string
customer_name	query		string
customer_native_name	query		string
customer_abbreviation	query		string
project	query		string
project_uuid	query		string
project_name	query		string
service_uuid	query		string
service_name	query		string
service_settings_name	query		string
service_settings_uuid	query		string
name	query		string
description	query		string
state	query		string
uuid	query		string
tag	query		string
rtag	query		string
o	query		string
tenant_uuid	query		string

Responses

200 -

GET /api/openstacktenant-security-groups/

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
name	query		string
settings_uuid	query		string
settings	query		string

Responses

200 -

POST /api/openstack-networks/

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstack-networks/**Parameters**

Name	Position	Description	Type
page	query		string
page_size	query		string
customer	query		string
customer_uuid	query		string
customer_name	query		string
customer_native_name	query		string
customer_abbreviation	query		string
project	query		string
project_uuid	query		string
project_name	query		string
service_uuid	query		string
service_name	query		string
service_settings_name	query		string
service_settings_uuid	query		string
name	query		string
description	query		string
state	query		string
uuid	query		string
tag	query		string
rtag	query		string
o	query		string
tenant_uuid	query		string
tenant	query		string

Responses

200 -

POST /api/openstack-tenants/{uuid}/pull/

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

GET /api/openstacktenant-security-groups/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

200 -

PUT /api/openstack/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses200 -

DELETE /api/openstack/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses204 -

PATCH /api/openstack/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses200 -

GET /api/openstack/{uuid}/

To update OpenStack service issue **PUT** or **PATCH** against `*/api/openstack/<service_uuid>/*`

- **Description:** To update OpenStack service issue **PUT** or **PATCH** against `*/api/openstack/<service_uuid>/*` as a customer owner. You can update service's 'name' and 'available_for_all' fields.

Example of a request:

```
.. code-block:: http
```

```
PUT /api/openstack/c6526bac12b343a9a65c4cd6710666ee/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "name": "My OpenStack2" }
```

To remove OpenStack service, issue ****DELETE**** against `*/api/openstack/<service_uuid>/*` as staff user or customer owner.

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/openstack-security-groups/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstack-security-groups/**Parameters**

Name	Position	Description	Type
page	query		string
page_size	query		string
description	query		string
name	query		string
error_message	query		string
backend_id	query		string
start_time	query		string
service_project_link	query		string
tenant	query		string
customer	query		string
customer_uuid	query		string
customer_name	query		string
customer_native_name	query		string
customer_abbreviation	query		string
project	query		string
project_uuid	query		string
project_name	query		string
service_uuid	query		string
service_name	query		string
service_settings_uuid	query		string
service_settings_name	query		string
state	query		string
uuid	query		string
tag	query		string
rtag	query		string
o	query		string
tenant_uuid	query		string

Responses

200 -

GET /api/hooks/

Use */api/hooks/* to get a list of all the hooks of any type that a user can see.

- **Description:** Use */api/hooks/* to get a list of all the hooks of any type that a user can see.

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string

Responses

200 -

POST /api/users/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/users/

User list is available to all authenticated users. To get a list,

- **Description:** User list is available to all authenticated users. To get a list, issue authenticated ****GET**** request against `*/api/users/*`.

User list supports several filters. All filters are set in HTTP query section. Field filters are listed below. All of the filters apart from `?organization` are using case insensitive partial matching.

Several custom filters are supported:

- `?current` - filters out user making a request. Useful for getting information about a currently logged in user.
- `?civil_number=XXX` - filters out users with a specified civil number
- `?is_active=True/False` - show only active (non-active) users
- `?potential` - shows users that have common connections to the customers and are potential collaborators. Exclude staff users. Staff users can see all the customers.
- `?potential_customer=<Customer UUID>` - optionally filter potential users by customer UUID
- `?potential_organization=<organization name>` - optionally filter potential unconnected users by their organization name (deprecated, use `'organization plugin <http://nodeconductor-organization.readthedocs.org/en/stable/>'` instead)
- `?organization_claimed` - show only users with a non-empty organization (deprecated, use `'organization plugin <http://nodeconductor-organization.readthedocs.org/en/stable/>'` instead)

The user can be created either through automated process on login with SAML token, or through a REST call by a user with staff privilege.

Example of a creation request is below.

.. code-block:: http

```
POST /api/users/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "username": "sample-user", "full_name": "full name", "native_name": "taisnimi", "job_title": "senior cleaning manager", "email": "example@example.com", "civil_number": "12121212", "phone_number": "", "description": "", "organization": "" }
```

NB! Username field is case-insensitive. So "John" and "john" will be treated as the same user.

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
full_name	query		string
native_name	query		string
organization	query		string
organization_approved	query		string
email	query		string
phone_number	query		string
description	query		string
job_title	query		string

Continued on next page

Table 5.1 – continued from previous page

Name	Position	Description	Type
username	query		string
civil_number	query		string
is_active	query		string
registration_method	query		string
o	query		string
full_name	query		string
native_name	query		string
organization	query		string
organization_approved	query		string
email	query		string
phone_number	query		string
description	query		string
job_title	query		string
username	query		string
civil_number	query		string
is_active	query		string
registration_method	query		string
o	query		string
full_name	query		string
native_name	query		string
organization	query		string
organization_approved	query		string
email	query		string
phone_number	query		string
description	query		string
job_title	query		string
username	query		string
civil_number	query		string
is_active	query		string
registration_method	query		string
o	query		string

Responses

200 -

POST /api/openstacktenant-volumes/{uuid}/attach/

Attach volume to instance

- **Description:** Attach volume to instance
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

GET /api/hooks-email/

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
user	query		string
is_active	query		string
last_published	query		string
email	query		string
author_uuid	query		string

Responses

200 -

POST /api/openstacktenant/{uuid}/unlink/

Unlink all related resources, service project link and service itself.

- **Description:** Unlink all related resources, service project link and service itself.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

PUT /api/customer-permissions/{id}/

- Customers are connected to users through roles, whereas user may have role “customer owner”.

- **Description:** - Customers are connected to users through roles, whereas user may have role “customer owner”.
- Each customer may have multiple owners, and each user may own multiple customers. - Staff members can list all available customers and create new customers. - Customer owners can list all customers they own. Customer owners can also create new customers. - Project administrators can list all the customers that own any of the projects they are administrators in. - Project managers can list all the customers that own any of the projects they are managers in.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path		string
data	body		

Responses

200 -

DELETE /api/customer-permissions/{id}/

- Customers are connected to users through roles, whereas user may have role “customer owner”.

- **Description:** - Customers are connected to users through roles, whereas user may have role “customer owner”.
 - Each customer may have multiple owners, and each user may own multiple customers. - Staff members can list all available customers and create new customers. - Customer owners can list all customers they own. Customer owners can also create new customers. - Project administrators can list all the customers that own any of the projects they are administrators in. - Project managers can list all the customers that own any of the projects they are managers in.

Parameters

Name	Position	Description	Type
id	path		string

Responses

204 -

PATCH /api/customer-permissions/{id}/

- Customers are connected to users through roles, whereas user may have role “customer owner”.

- **Description:** - Customers are connected to users through roles, whereas user may have role “customer owner”.
 - Each customer may have multiple owners, and each user may own multiple customers. - Staff members can list all available customers and create new customers. - Customer owners can list all customers they own. Customer owners can also create new customers. - Project administrators can list all the customers that own any of the projects they are administrators in. - Project managers can list all the customers that own any of the projects they are managers in.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path		string
data	body		

Responses

200 -

GET /api/customer-permissions/{id}/

To remove a user from a customer owner group, delete corresponding connection (**url** field).

- **Description:** To remove a user from a customer owner group, delete corresponding connection (**url** field). Successful deletion will return status code 204.

.. code-block:: http

```
DELETE /api/customer-permissions/71/ HTTP/1.1 Authorization: Token
95a688962bf68678fd4c8cec4d138ddd9493c93b Host: example.com
```

Parameters

Name	Position	Description	Type
id	path		string

Responses

200 -

POST /api/openstack-tenants/{uuid}/create_network/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

PUT /api/openstack-service-project-link/{id}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path		string
data	body		

Responses

200 -

DELETE /api/openstack-service-project-link/{id}/

Parameters

Name	Position	Description	Type
id	path		string

Responses

204 -

PATCH /api/openstack-service-project-link/{id}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path		string
data	body		

Responses

200 -

GET /api/openstack-service-project-link/{id}/

To remove a link, issue **DELETE** to URL of the corresponding connection as staff user or customer owner.

- **Description:** To remove a link, issue **DELETE** to URL of the corresponding connection as staff user or customer owner.

Parameters

Name	Position	Description	Type
id	path		string

Responses

200 -

POST /api/users/{uuid}/password/

To change a user password, submit a **POST** request to the user's RPC URL, specifying new password

- **Description:** To change a user password, submit a **POST** request to the user's RPC URL, specifying new password by staff user or account owner.

Password is expected to be at least 7 symbols long and contain at least one number and at least one lower or upper case.

Example of a valid request:

.. code-block:: http

```
POST /api/users/e0c058d06864441fb4f1c40dee5dd4fd/password/ HTTP/1.1 Content-Type: application/json
Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "password": "nQvqHzeP123", }
```

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

GET /api/openstacktenant-floating-ips/**Parameters**

Name	Position	Description	Type
page	query		string
page_size	query		string
name	query		string
settings_uuid	query		string
settings	query		string
runtime_state	query		string

Responses

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
user	query		string
is_active	query		string
last_published	query		string
type	query		string
device_id	query		string
device_manufacturer	query		string
device_model	query		string
token	query		string
author_uuid	query		string

Responses

200 -

POST /api/events/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/events/

To get a list of events - run **GET** against `*/api/events/*` as authenticated user. Note that a user can

- **Description:** To get a list of events - run **GET** against `*/api/events/*` as authenticated user. Note that a user can only see events connected to objects she is allowed to see.

Sorting is supported in ascending and descending order by specifying a field to an `o=` parameter. By default events are sorted by `@timestamp` in descending order.

Run POST against `*/api/events/*` to create an event. Only users with staff privileges can create events. New event will be emitted with 'custom_notification' event type. Request should contain following fields:

- level: the level of current event. Following levels are supported: debug, info, warning, error - message: string representation of event message - scope: optional URL, which points to the loggable instance

Request example:

```
.. code-block:: javascript
```

```
POST /api/events/ Accept: application/json Content-Type: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "level": "info", "message": "message#1", "scope": "http://example.com/api/customers/9cd869201e1b4158a285427fcd790c1c/" }
```

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string

Responses

200 -

GET /api/customer-permissions-log/{id}/

Parameters

Name	Position	Description	Type
id	path		string

Responses

200 -

POST /api/openstack-tenants/{uuid}/create_security_group/

Example of a request:

- **Description:** Example of a request:

.. code-block:: http

```
{ "name": "Security group name", "description": "description", "rules": [ { "protocol": "tcp", "from_port": 1, "to_port": 10, "cidr": "10.1.1.0/24" }, { "protocol": "udp", "from_port": 10, "to_port": 8000, "cidr": "10.1.1.0/24" } ] }
```

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

POST /api/openstacktenant-volumes/{uuid}/pull/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

GET /api/customer-permissions-log/

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
role	query		string
user	query		string
user_url	query		string
username	query		string
full_name	query		string
native_name	query		string
o	query		string
customer	query		string
customer_url	query		string

Responses

200 -

GET /api/openstacktenant-flavors/

VM instance flavor is a pre-defined set of virtual hardware parameters that the instance will use:

- **Description:** VM instance flavor is a pre-defined set of virtual hardware parameters that the instance will use: CPU, memory, disk size etc. VM instance flavor is not to be confused with VM template – flavor is a set of virtual hardware parameters whereas template is a definition of a system to be installed on this instance.

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
ram	query		string
ram_gte	query		string
ram_lte	query		string
name	query		string
settings	query		string
cores	query		string
cores_gte	query		string
cores_lte	query		string
disk	query		string
disk_gte	query		string
disk_lte	query		string
settings_uuid	query		string
o	query		string

Responses

200 -

POST /api/openstack-ip-mappings/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstack-ip-mappings/

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
project	query		string
private_ip	query		string
public_ip	query		string

Responses

200 -

GET /api/openstacktenant-flavors/{uuid}/

VM instance flavor is a pre-defined set of virtual hardware parameters that the instance will use:

- **Description:** VM instance flavor is a pre-defined set of virtual hardware parameters that the instance will use: CPU, memory, disk size etc. VM instance flavor is not to be confused with VM template – flavor is a set of virtual hardware parameters whereas template is a definition of a system to be installed on this instance.

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

PUT /api/openstack-packages/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

DELETE /api/openstack-packages/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/openstack-packages/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

200 -

GET /api/openstack-packages/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/keys/

SSH public keys are injected to VM instances during creation, so that holder of corresponding SSH private key can

- **Description:** SSH public keys are injected to VM instances during creation, so that holder of corresponding SSH private key can log in to that instance. SSH public keys are connected to user accounts, whereas the key may belong to one user only, and the user may have multiple SSH keys. Users can only access SSH keys connected to their accounts. Staff users can see all the accounts. Project administrators can select what SSH key will be injected into VM instance during instance provisioning.

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/keys/

To get a list of SSH keys, run ****GET**** against `*/api/keys/*` as authenticated user.

- **Description:** To get a list of SSH keys, run ****GET**** against `*/api/keys/*` as authenticated user.

A new SSH key can be created by any active users. Example of a valid request:

```
.. code-block:: http
```

```
POST /api/keys/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "name": "ssh_public_key1", "public_key": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDDURXDP5YhOQUYoD
TtLm5yBDRLKAERqtlbH2gkrQ3US58gd2r8H9jAmQOydfvgwauXuJUE4eDpaMWupqquMYsYLB5f+vVGhdZbbzfc6DTQ2rY
dknWoMoArlG7MvRMA/xQ0ye1muTv+mYMipnd7Z+WH0uVArYI9QBpqC/gpZRRiouQ4VIQIVWGoT6M4Kat5ZBXEa9yP-
D2C05GX3gumoSAVyAcDHn/xgej9pYRXGha4I+LKKFdGwAoXdV1z79EG1+9ns7wXuqMJFHM2KDpxAizV0GkZcojISvDw
vEAFdOJcqjyyH4FOGYa8usP1 jhon@example.com", }
```

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
name	query		string
fingerprint	query		string
uuid	query		string
user_uuid	query		string
o	query		string

Responses

200 -

GET /api/customers/{uuid}/counters/

Count number of entities related to customer

- **Description:** Count number of entities related to customer

.. code-block:: javascript

```
{ "alerts": 12, "services": 1, "projects": 1, "users": 3 }
```

Parameters

Name	Position	Description	Type
uuid	path		string
page	query		string
page_size	query		string

Responses

200 -

GET /api/openstack-images/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

PUT /api/projects/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/projects/{uuid}/

Deletion of a project is done through sending a **DELETE** request to the project instance URI.

- **Description:** Deletion of a project is done through sending a **DELETE** request to the project instance URI. Please note, that if a project has connected instances, deletion request will fail with 409 response code.

Valid request example (token is user specific):

.. code-block:: http

```
DELETE /api/projects/6c9b01c251c24174a6691a1f894fae31/ HTTP/1.1 Authorization: Token
c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/projects/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/projects/{uuid}/

Optional 'field' query parameter (can be list) allows to limit what fields are returned.

- **Description:** Optional 'field' query parameter (can be list) allows to limit what fields are returned. For example, given request /api/projects/<uuid>/?field=uuid&field=name you get response like this:

.. code-block:: javascript

```
{ "uuid": "90bcfe38b0124c9bbdadd617b5d739f5", "name": "Default" }
```

Parameters

200 -

GET /api/openstack-flavors/{uuid}/

VM instance flavor is a pre-defined set of virtual hardware parameters that the instance will use:

- **Description:** VM instance flavor is a pre-defined set of virtual hardware parameters that the instance will use: CPU, memory, disk size etc. VM instance flavor is not to be confused with VM template – flavor is a set of virtual hardware parameters whereas template is a definition of a system to be installed on this instance.

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/openstack-packages/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstack-packages/**Parameters**

Name	Position	Description	Type
page	query		string
page_size	query		string
name	query		string
customer	query		string
project	query		string
tenant	query		string

Responses

200 -

GET /api/openstacktenant-floating-ips/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

DELETE /api/keys/{uuid}/

SSH public keys are injected to VM instances during creation, so that holder of corresponding SSH private key can

- **Description:** SSH public keys are injected to VM instances during creation, so that holder of corresponding SSH private key can log in to that instance. SSH public keys are connected to user accounts, whereas the key may belong to one user only, and the user may have multiple SSH keys. Users can only access SSH keys connected to their accounts. Staff users can see all the accounts. Project administrators can select what SSH key will be injected into VM instance during instance provisioning.

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

GET /api/keys/{uuid}/

SSH public keys are injected to VM instances during creation, so that holder of corresponding SSH private key can

- **Description:** SSH public keys are injected to VM instances during creation, so that holder of corresponding SSH private key can log in to that instance. SSH public keys are connected to user accounts, whereas the key may belong to one user only, and the user may have multiple SSH keys. Users can only access SSH keys connected to their accounts. Staff users can see all the accounts. Project administrators can select what SSH key will be injected into VM instance during instance provisioning.

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

GET /api/service-metadata/

To get a list of supported service types, run ****GET**** against `*/api/service-metadata/*` as an authenticated user.

- **Description:** To get a list of supported service types, run ****GET**** against `*/api/service-metadata/*` as an authenticated user. Use an endpoint from the returned list in order to create new service.

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string

Responses

200 -

POST /api/openstacktenant-volumes/

- **Consumes:** [u'application/json']
-

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstacktenant-volumes/

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
customer	query		string
customer_uuid	query		string
customer_name	query		string
customer_native_name	query		string
customer_abbreviation	query		string
project	query		string
project_uuid	query		string
project_name	query		string
service_uuid	query		string
service_name	query		string
service_settings_name	query		string
service_settings_uuid	query		string
name	query		string
description	query		string
state	query		string
uuid	query		string
tag	query		string
rtag	query		string
instance	query		string
instance_uuid	query		string
o	query		string

Responses

200 -

POST /api/openstack-tenants/{uuid}/create_service/

Create non-admin service with credentials from the tenant

- **Description:** Create non-admin service with credentials from the tenant
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

In all cases all currently running resources are returned, if SLA for the given period is not known or not present, it will be shown as ****null**** in the response.

Resources may have monitoring items attached to it. Example rendering of monitoring items:

```
.. code-block:: javascript
“monitoring_items”: { “application_state”: 1 }
```

You may filter or order resources by monitoring item.

- Example query for filtering list of resources by installation state:

```
/api/<resource_endpoint>/?monitoring__installation_state=1
```

- Warning! If resource does not have monitoring item attached to it, it is not included in ordered response. Example query for ordering list of resources by installation state:

```
/api/<resource_endpoint>/?o=monitoring__installation_state
```

Filtering by tags ^^^^^^^^^^^^^^^^^^^^^

Resource may have tags attached to it. Example of tags rendering:

```
.. code-block:: javascript
“tags”: [ “license-os:centos7”, “os-family:linux”, “license-application:postgresql”, “support:premium” ]
```

Tags filtering:

- ?tag=IaaS - filter by full tag name, using method OR. Can be list. - ?rtag=os-family:linux - filter by full tag name, using AND method. Can be list. - ?tag__license-os=centos7 - filter by tags with particular prefix.

Tags ordering:

- ?o=tag__license-os - order by tag with particular prefix. Instances without given tag will not be returned.

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string

Responses

200 -

POST /api-auth/password/

Api view loosely based on DRF’s default ObtainAuthToken,

- **Description:** Api view loosely based on DRF’s default ObtainAuthToken, but with the responses formats and status codes aligned with BasicAuthentication behavior.

Valid request example:

```
.. code-block:: http
POST /api-auth/password/ HTTP/1.1
```

Parameters

Name	Position	Description	Type
------	----------	-------------	------

Responses

201 -

PUT /api/openstacktenant/{uuid}/

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/openstacktenant/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/openstacktenant/{uuid}/

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/openstacktenant/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/openstacktenant-instances/{uuid}/assign_floating_ip/

To assign floating IP to the instance, make ****POST**** request to

- **Description:** To assign floating IP to the instance, make ****POST**** request to `*/api/openstacktenant-instances/<uuid>/assign_floating_ip/*` with link to the floating IP. Make empty POST request to allocate new floating IP and assign it to instance. Note that instance should be in stable state, service project link of the instance should be in stable state and have external network.

Example of a valid request:

.. code-block:: http

```
POST /api/openstacktenant-instances/6c9b01c251c24174a6691a1f894fae31/assign_floating_ip/
HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token
c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "floating_ip": "http://example.com/api/floating-ips/5e7d93955f114d88981dea4f32ab673d" }
```

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

PUT /api/openstack-networks/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/openstack-networks/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/openstack-networks/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/openstack-networks/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/openstacktenant-instances/{uuid}/backup/

OpenStack instance permissions

- **Description:** OpenStack instance permissions ^^^
 - Staff members can list all available VM instances in any service. - Customer owners can list all VM instances in all the services that belong to any of the customers they own. - Project administrators can list all VM instances, create new instances and start/stop/restart instances in all the services that are connected to any of the projects they are administrators in. - Project managers can list all VM instances in all the services that are connected to any of the projects they are managers in.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

PUT /api/users/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/users/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/users/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/users/{uuid}/

User fields can be updated by account owner or user with staff privilege (is_staff=True).

- **Description:** User fields can be updated by account owner or user with staff privilege (is_staff=True). Following user fields can be updated:

- organization (deprecated, use 'organization plugin <[Can be done by **PUT**ing a new data to the user URI, i.e. `*/api/users/<UUID>/*` by staff user or account owner. Valid request example \(token is user specific\):](http://nodeconductor-organization.readthedocs.org/en/stable/>'_ instead) - full_name - native_name - job_title - phone_number - email</p></div><div data-bbox=)

.. code-block:: http

```
PUT /api/users/e0c058d06864441fb4f1c40dee5dd4fd/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6bac41c706593e66f567a7fa4 Host: example.com
```

```
{ "email": "example@example.com", "organization": "Bells organization", }
```

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/openstack-floating-ips/{uuid}/pull/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

201 -

POST /api/openstack-service-project-link/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstack-service-project-link/

In order to be able to provision OpenStack resources, it must first be linked to a project. To do that,

- **Description:** In order to be able to provision OpenStack resources, it must first be linked to a project. To do that, ****POST**** a connection between project and a service to `*/api/openstack-service-project-link/*` as stuff user or customer owner.

Example of a request:

.. code-block:: http

```
POST /api/openstack-service-project-link/ HTTP/1.1 Content-Type: application/json Accept: application/json
Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{  "project":      "http://example.com/api/projects/e5f973af2eb14d2d8c38d62bcbaccb33/",  "service":
"http://example.com/api/openstack/b0e8a4cbd47c4f9ca01642b7ec033db4/" }
```

To remove a link, issue DELETE to URL of the corresponding connection as stuff user or customer owner.

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
project	query		string
service	query		string
service_uuid	query		string
customer_uuid	query		string
project_uuid	query		string

Responses

200 -

PUT /api/hooks-web/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/hooks-web/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/hooks-web/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/hooks-web/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

GET /api/openstacktenant/{uuid}/managed_resources/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/openstack-packages/extend/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

POST /api/openstacktenant-service-project-link/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstacktenant-service-project-link/

To get a list of connections between a project and an service, run ****GET**** against service_project_link_url

- **Description:** To get a list of connections between a project and an service, run ****GET**** against service_project_link_url as authenticated user. Note that a user can only see connections of a project where a user has a role.

If service has 'available_for_all' flag, project-service connections are created automatically. Otherwise, in order to be able to provision resources, service must first be linked to a project. To do that, ****POST**** a connection between project and a service to service_project_link_url as stuff user or customer owner.

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
project	query		string
service	query		string
service_uuid	query		string
customer_uuid	query		string
project_uuid	query		string

Responses

200 -

GET /api/services/

Filter services by type

- **Description:** Filter services by type ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

It is possible to filter services by their types. Example:

/api/services/?service_type=DigitalOcean&service_type=OpenStack

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string

Responses

200 -

POST /api/openstack-security-groups/{uuid}/set_rules/

WARNING! Auto-generated HTML form is wrong for this endpoint. List should be defined as input.

- **Description:** WARNING! Auto-generated HTML form is wrong for this endpoint. List should be defined as input.

Example: [{ "protocol": "tcp", "from_port": 1, "to_port": 10, "cidr": "10.1.1.0/24" }]

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

POST /api/openstacktenant-instances/{uuid}/start/

OpenStack instance permissions

- **Description:** OpenStack instance permissions ^^^
 - Staff members can list all available VM instances in any service. - Customer owners can list all VM instances in all the services that belong to any of the customers they own. - Project administrators can list all VM instances, create new instances and start/stop/restart instances in all the services that are connected to any of the projects they are administrators in. - Project managers can list all VM instances in all the services that are connected to any of the projects they are managers in.

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

POST /api/openstacktenant-volumes/{uuid}/detach/

Detach instance from volume

- **Description:** Detach instance from volume

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

POST /api/openstack-tenants/{uuid}/pull_security_groups/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

POST /api/openstack-networks/{uuid}/create_subnet/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

PUT /api/service-settings/{uuid}/

To update service settings, issue a **PUT** or **PATCH** to `*/api/service-settings/<uuid>/` as a customer owner.

- **Description:** To update service settings, issue a **PUT** or **PATCH** to `*/api/service-settings/<uuid>/` as a customer owner. You are allowed to change name and credentials only.

Example of a request:

.. code-block:: http

```
PATCH /api/service-settings/9079705c17d64e6aa0af2e619b0e0702/ HTTP/1.1 Content-Type: application/json
Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "username": "admin", "password": "new_secret" }
```

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

PATCH /api/service-settings/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/service-settings/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/openstack/{uuid}/link/

To get a list of resources available for import, run ****GET**** against `*/<service_endpoint>/link/*`

- **Description:** To get a list of resources available for import, run ****GET**** against `*/<service_endpoint>/link/*` as an authenticated user. Optionally `project_uuid` parameter can be supplied for services requiring it like OpenStack.

To import (link with NodeConductor) resource issue ****POST**** against the same endpoint with resource id.

.. code-block:: http

```
POST /api/openstack/08039f01c9794efc912f1689f4530cf0/link/ HTTP/1.1 Content-Type: application/json
Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{  "backend_id": "bd5ec24d-9164-440b-a9f2-1b3c807c5df3",  "project": "http://example.com/api/projects/e5f973af2eb14d2d8c38d62bcbaccb33/" }
```

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

GET /api/openstack/{uuid}/link/

To get a list of resources available for import, run ****GET**** against `*/<service_endpoint>/link/*`

- **Description:** To get a list of resources available for import, run ****GET**** against `*/<service_endpoint>/link/*` as an authenticated user. Optionally `project_uuid` parameter can be supplied for services requiring it like Open-Stack.

To import (link with NodeConductor) resource issue ****POST**** against the same endpoint with resource id.

.. code-block:: http

```
POST /api/openstack/08039f01c9794efc912f1689f4530cf0/link/ HTTP/1.1 Content-Type: application/json
Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{      "backend_id":          "bd5ec24d-9164-440b-a9f2-1b3c807c5df3",      "project":
"http://example.com/api/projects/e5f973af2eb14d2d8c38d62bcbaccb33/" }
```

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

PUT /api/openstack-ip-mappings/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/openstack-ip-mappings/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/openstack-ip-mappings/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses200 -

GET /api/openstack-ip-mappings/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses200 -

POST /api/openstack-tenants/{uuid}/set_quotas/

A quota can be set for a particular tenant. Only staff users can do that.

- **Description:** A quota can be set for a particular tenant. Only staff users can do that. In order to set quota submit ****POST**** request to `*/api/openstack-tenants/<uuid>/set_quotas/*`. The quota values are propagated to the backend.

The following quotas are supported. All values are expected to be integers:

- instances - maximal number of created instances.
- ram - maximal size of ram for allocation. In MiB_.
- storage - maximal size of storage for allocation. In MiB_.
- vcpu - maximal number of virtual cores for allocation.
- security_group_count - maximal number of created security groups.
- security_group_rule_count - maximal number of created security groups rules.
- volumes - maximal number of created volumes.
- snapshots - maximal number of created snapshots.

It is possible to update quotas by one or by submitting all the fields in one request. NodeConductor will attempt to update the provided quotas. Please note, that if provided quotas are conflicting with the backend (e.g. requested number of instances is below of the already existing ones), some quotas might not be applied.

.. _MiB: <http://en.wikipedia.org/wiki/Mebibyte> .. _settings: <http://nodeconductor.readthedocs.org/en/stable/guide/intro.html#id1>

Example of a valid request (token is user specific):

.. code-block:: http

```
POST /api/openstack-tenants/c84d653b9ec92c6cbac41c706593e66f567a7fa4/set_quotas/ HTTP/1.1 Content-Type: application/json Accept: application/json Host: example.com
```

```
{ "instances": 30, "ram": 100000, "storage": 1000000, "vcpu": 30, "security_group_count": 100, "security_group_rule_count": 100, "volumes": 10, "snapshots": 20 }
```

Response code of a successful request is ****202 ACCEPTED****. In case tenant is in a non-stable status, the response would be ****409 CONFLICT****. In this case REST client is advised to repeat the request after some time. On successful completion the task will synchronize quotas with the backend.

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

POST /api/openstack/{uuid}/unlink/

Unlink all related resources, service project link and service itself.

- **Description:** Unlink all related resources, service project link and service itself.

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

PUT /api/customers/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/customers/{uuid}/

Deletion of a customer is done through sending a ****DELETE**** request to the customer instance URI. Please note,

- **Description:** Deletion of a customer is done through sending a ****DELETE**** request to the customer instance URI. Please note, that if a customer has connected projects, deletion request will fail with 409 response code.

Valid request example (token is user specific):

```
.. code-block:: http
```

```
DELETE /api/customers/6c9b01c251c24174a6691a1f894fae31/ HTTP/1.1 Authorization: Token
c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/customers/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/customers/{uuid}/

Optional 'field' query parameter (can be list) allows to limit what fields are returned.

- **Description:** Optional 'field' query parameter (can be list) allows to limit what fields are returned. For example, given request /api/customers/<uuid>/?field=uuid&field=name you get response like this:

.. code-block:: javascript

```
{ "uuid": "90bcfe38b0124c9bbdadd617b5d739f5", "name": "Ministry of Bells" }
```

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/customers/

A new customer can only be created by users with staff privilege (is_staff=True).

- **Description:** A new customer can only be created by users with staff privilege (is_staff=True). Example of a valid request:

.. code-block:: http

```
POST /api/customers/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "name": "Customer A", "native_name": "Customer A", "abbreviation": "CA", "contact_details": "Luhamaa 28, 10128 Tallinn", }
```

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/customers/

To get a list of customers, run GET against */api/customers/* as authenticated user. Note that a user can

- **Description:** To get a list of customers, run GET against */api/customers/* as authenticated user. Note that a user can only see connected customers:

- customers that the user owns - customers that have a project where user has a role

Staff also can filter customers by user UUID, for example /api/customers/?user_uuid=<UUID>

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
name	query		string
abbreviation	query		string
contact_details	query		string
native_name	query		string
registration_code	query		string
o	query		string

Responses

200 -

POST /api/openstacktenant-snapshots/{uuid}/pull/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

PUT /api/customers/{uuid}/image/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/customers/{uuid}/image/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/customers/{uuid}/image/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/customers/{uuid}/image/

Parameters

Name	Position	Description	Type
uuid	path		string
page	query		string
page_size	query		string

Responses

200 -

GET /api/events/count/

To get a count of events - run **GET** against `*/api/events/count/*` as authenticated user.

- **Description:** To get a count of events - run **GET** against `*/api/events/count/*` as authenticated user. End-point support same filters as events list.

Response example:

```
.. code-block:: javascript
```

```
{“count”: 12321}
```

Parameters

Name	Position	Description	Type
------	----------	-------------	------

Responses

200 -

POST /api/openstacktenant-volumes/{uuid}/snapshot/

Create snapshot from volume

- **Description:** Create snapshot from volume
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

POST /api/openstack-security-groups/{uuid}/pull/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

POST /api/openstack-subnets/{uuid}/pull/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

201 -

GET /api/openstacktenant-images/**Parameters**

Name	Position	Description	Type
page	query		string
page_size	query		string
name	query		string
settings_uuid	query		string
settings	query		string

Responses

200 -

PUT /api/openstacktenant-volumes/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/openstacktenant-volumes/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/openstacktenant-volumes/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/openstacktenant-volumes/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

POST /api/openstacktenant/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/openstacktenant/

To list all services without regard to its type, run ****GET**** against `*/api/services/*` as an authenticated user.

- **Description:** To list all services without regard to its type, run ****GET**** against `*/api/services/*` as an authenticated user.

To list services of specific type issue ****GET**** to specific endpoint from a list above as a customer owner. Individual endpoint used for every service type.

To create a service, issue a ****POST**** to specific endpoint from a list above as a customer owner. Individual endpoint used for every service type.

You can create service based on shared service settings. Example:

```
.. code-block:: http
```

```
POST /api/digitalocean/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "name": "Common DigitalOcean", "customer": "http://example.com/api/customers/1040561ca9e046d2b74268600c7e1105/", "settings": "http://example.com/api/service-settings/93ba615d6111466ebe3f792669059cb4/" }
```

Or provide your own credentials. Example:

```
.. code-block:: http
```

```
POST /api/oracle/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 Host: example.com
```

```
{ "name": "My Oracle", "customer": "http://example.com/api/customers/1040561ca9e046d2b74268600c7e1105/", "backend_url": "https://oracle.example.com:7802/em", "username": "admin", "password": "secret" }
```

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
name	query		string
project_uuid	query		string
customer	query		string
project	query		string
settings	query		string
shared	query		string
type	query		string
tag	query		string
rtag	query		string

Responses

200 -

GET /api/openstack/{uuid}/managed_resources/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

200 -

GET /api/openstack-images/

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
name	query		string
settings_uuid	query		string
settings	query		string

Responses

200 -

PUT /api/openstack-security-groups/{uuid}/

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/openstack-security-groups/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses204 -

PATCH /api/openstack-security-groups/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses200 -

GET /api/openstack-security-groups/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses200 -

GET /api/openstacktenant-images/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses200 -

POST /api/customer-permissions/

- Customers are connected to users through roles, whereas user may have role “customer owner”.

- **Description:** - Customers are connected to users through roles, whereas user may have role “customer owner”.
 - Each customer may have multiple owners, and each user may own multiple customers. - Staff members can list all available customers and create new customers. - Customer owners can list all customers they own. Customer owners can also create new customers. - Project administrators can list all the customers that own any of the projects they are administrators in. - Project managers can list all the customers that own any of the projects they are managers in.

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/customer-permissions/

Each customer is associated with a group of users that represent customer owners. The link is maintained

- **Description:** Each customer is associated with a group of users that represent customer owners. The link is maintained through `**api/customer-permissions/**` endpoint.

To list all visible links, run a `**GET**` query against a list. Response will contain a list of customer owners and their brief data.

To add a new user to the customer, `**POST**` a new relationship to `**customer-permissions**` endpoint:

.. code-block:: http

```
POST /api/customer-permissions/ HTTP/1.1 Accept: application/json Authorization: Token 95a688962bf68678fd4c8cec4d138ddd9493c93b Host: example.com
```

```
{ "customer": "http://example.com/api/customers/6c9b01c251c24174a6691a1f894fae31/", "role": "owner", "user": "http://example.com/api/users/82cec6c8e0484e0ab1429412fe4194b7/" }
```

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
role	query		string
user	query		string
user_url	query		string
username	query		string
full_name	query		string
native_name	query		string
o	query		string
customer	query		string
customer_url	query		string

Responses

200 -

PUT /api/openstack-tenants/{uuid}/

- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

DELETE /api/openstack-tenants/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/openstack-tenants/{uuid}/

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

GET /api/openstack-tenants/{uuid}/**Parameters**

Name	Position	Description	Type
uuid	path		string

Responses

200 -

6.1 Instructions for Registry Interface Deployment and Development

The Registry Interface is a key component of the SUNFISH architecture. It has two purposes. On one hand, it interacts with the fabric blockchain. On the other hand, it provides web interfaces for other SUNFISH components to enable the interactions between those components and the blockchain.

It has two components **RI** and **RI Infrastructure** which are presented below.

6.1.1 RI

RI is responsible for handling requests from other SUNFISH components in all tenants, except in an infrastructure tenant. Next, two instruction sets are presented. One describes how to deploy this component. The other presents the development paradigm to aid other developers working in future.

Deployment Guide

RI can be deployed by following the steps presented below. It has been integrated with the fabric blockchain by utilising the docker container provided by the fabric project. However, thanks to its modular architecture, other blockchain can be easily integrated in future. Follow the Development Guide below to understand how it can be done.

1. Prepare the hosting machine by following the instructions at: <http://hyperledger-fabric.readthedocs.io/en/latest/prereqs.html>
2. Setup your GOPATH environment variable as required.
3. Clone the Registry repository using the following command:

```
git clone https://github.com/sunfish-prj/Registry.git
```

4. cd into *Registry/chaincode* directory.
5. Copy the “github.com” directory from the *Registry/chaincode* directory to $\$GOPATH/src/$

6. Clone the Registry Interface repository using the following command:

```
git clone https://github.com/sunfish-prj/Registry-Interface.git
```

7. cd into *Registry-Interface/RI* directory.

8. Issue the following command to make shell scripts executable:

```
chmod a+x channel_test.sh deployAll.sh stop.sh
```

9. Issue the following command to install the required node packages:

```
npm install
```

10. Update the **goPath** config field in *Registry-Interface/RI/config.json* with the go path directory of the host (as printed by the **:bash:'echo \$GOPATH'** command)
11. Update the **dockerIP** config field in *Registry-Interface/RI/config.ini* with the IP address of the docker interface (for ubuntu, use the `ifconfig` command to get the IP address of the docker interface for the host machine)
12. Within the *Registry-Interface/RI* directory, issue the following command in a terminal. This creates the fabric blockchain and initiates and deploys the required entities for the particular blockchain.

```
docker-compose -f dockerCompose.yml up -d
```

13. In another terminal, use the command `docker exec -it cli bash` to connect to the cli container and then issue: `more results.txt`. Repeat `more results.txt` until the following outputs are printed. This ensures that all peers have joined the created channel.

```
SUCCESSFUL CHANNEL CREATION
SUCCESSFUL JOIN CHANNEL on PEER0
SUCCESSFUL JOIN CHANNEL on PEER1
SUCCESSFUL JOIN CHANNEL on PEER2
```

14. In different terminals, the following commands can be used to trace the logs of the orderer and peer0 (or peer1/peer2 by changing the respective value) respectively :

```
docker logs -f orderer
```

```
docker logs -f peer0
```

15. In another terminal, within the *Registry-Interface/RI* directory, the following command needs to be issued to deploy the required smart contracts (chaincode):

```
./deployAll.sh
```

16. Wait until the following output is printed. This confirms that the smart contract has been successfully deployed in the fabric blockchain. This output will be repeated all each chaincode.

```
The chaincode transaction has been successfully committed
```

17. In the same terminal (or in a different terminal), within the *Registry-Interface/RI* directory, the following command needs to be issued. This starts the node server for the registry interface, listening at port 8075.

```
node ri.js
```

18. Wait until the *server started* output is printed in the terminal. This indicates that the node server for RI has been successfully started.

19. Test the interface by registering, retrieving, updating and deleting some dummy data, use the test cases from the from the testCases file. For these test cases, *docker_IP* needs to be updated accordingly. The *in/index* field needs to be updated accordingly for reading from the interface.
20. To get the output of the smart-contract, the following command can be issued after a single data has been registered/stored. Here, "..." represents the corresponding container name.

```
docker logs -f peer0-peer0...
```

21. Once finished, issue the following command to stop and remove the fabric containers:

```
./stop.sh
```

22. Repeat the steps from step 10 to deploy the smart contracts and utilise the ri.
23. To enable the interactions between the RI and FRM/FAM, a separate instance of RI for any infrastructure tenant is required. This needs to be deployed following the instructions provided below.

Development Guide

Ri has been developed using node.js. The flow control in the registry interface is as follows:

```
SUNFISH Component ====> ri.js --> *API.js --> hyperledger/hyperledger*.js ====> ↵
↵ fabric ====> SUNFISH Component
```

The *ri.js* is the entry point of the registry interface. There are different *hyperledger*.js* files inside the *hyperledger*; each of which is responsible for interacting with a particular smart-contract. There are also different *API.js files* which are responsible for forwarding each request to the appropriate *hyperledger.js* file. Currently, these **API.js* files are configured to *hyperledger*. However, if needed, this configuration can be changed in the *config.ini* file and also by developing required **.js* files which interact with the other blockchain.

A SUNFISH component submits a request following the SUNFISH RI specification. Based on the request path, the request is forwarded internally to the appropriate *API.js file*. Then this file forwards the request to the corresponding *hyperledger.js* file where the request is handled.

6.1.2 RI Infrastructure

RI Infrastructure is responsible for handling requests from other SUNFISH components in an infrastructure tenant. Next, two instruction sets are presented. One describes how to deploy this component. The other presents the development paradigm to aid other developers working in future.

Deployment Guide

1. If not already cloned, clone the Registry Interface project using the following command:

```
git clone https://github.com/sunfish-prj/Registry-Interface.git
```

2. cd into *Registry-Interface/INF_RI* directory.
3. Configure the IP address of the hosting machine by changing the *frmIP* parameter in the *config.ini* file.
4. In a terminal, within the *Registry-Interface/INF_RI* directory, the following command needs to be issued. This starts the node server for the registry interface for the infrastructure tenant, listening at port 8076.

```
node infRI.js
```

5. Wait until the *server started* output is printed in the terminal. This indicates that the node server for Infrastructure RI has been successfully started.

Development Guide

This follows the same pattern described in the previous section.

This is a page for the Registry Architecture.

7.1 Instructions for deploying chaincode

The current iteration of SUNFISH Registry leverages the hyperledger fabric blockchain and the chaincode represents the smart-contract which are executed on the fabric blockchain.

Currently, the chaincode has been written in Go lang and is hosted at: <https://github.com/sunfish-prj/Registry>

Within the *chaincode/github.com* directory of the repository, there are several directories. In each directory, there is a chaincode for a particular functionality of the **RI**. For example, the *chaincode/github.com/alert* directory contains a chaincode called *alert.go* which is used by corresponding **RI** endpoint to store and retrieve an alert in the blockchain and so on. These directories also contain a file called **Dockerfile** which is used to deploy any particular chaincode in the corresponding container.

7.1.1 Deployment Guide

Follow steps are required to deploy any chaincode.

1. Prepare the hosting machine by following the instructions at: <http://hyperledger-fabric.readthedocs.io/en/latest/prereqs.html>
2. Setup your GOPATH environment variable as required.
3. Clone the Registry repository using the following command:

```
git clone https://github.com/sunfish-prj/Registry.git
```

4. cd into *Registry/chaincode* directory.
5. Copy the *github.com* directory from the *Registry/chaincode* directory to `$GOPATH/src/`

6. Once copied, no other additional step is required. The copied chaincode will be automatically deployed in the container by the deployment script of the RI.

This is a page for the FRM.

8.1 Instructions for deploying FRM

FRM (Federated Runtime Monitoring) consists of two directories having two components: i) Proxy and ii) Chaincode component. The development and deployment models for each of these components are discussed below.

8.1.1 Proxy

The proxy component represents the proxies that need to be attached to the DS components to enable the interception of access requests and responses.

Development model

The proxy component has been developed as a servlet filter in order to be compatible with the DS servlets. It consists of two Java source files, named ProxyFilter.java and CachedServletRequest.java under the sunfish.frm.proxy package.

The supplied pom.xml file contains the maven dependency code snippet for the required libraries.

The supplied web.xml file, located under the src/main/webapp/WEB-INF directory, contains the servlet mapping for the servlet filter.

A config.json file, located under the src/main/webapp/WEB-INF directory, contains configuration directives.

There are several additional libraries supplied in the src/main/webapp/WEB-INF/lib directory. which need to be properly added into the java path during the deployment.

Deployment guide

The following steps are required to deploy and/or integrate the proxy with each DS component.

1. **Setup the configuration directives:** 1.1 `hostingID`: denotes the DS component with which the proxy is being integrated. Currently, it takes the value of either **PDP** or **PEP**. Other values can be attached, however, the `ProxyFilter.java` source needs to be updated accordingly. 1.2 `loggerID`: the identifier of the hosting entity.
2. Add the dependency code snippet from the `pom.xml` file of the proxy to the `pom.xml` file of the corresponding DS component.
3. Add the code servlet mapping code snippet from the `web.xml` of the proxy to the `web.xml` file of the corresponding DS component.
4. Add the additional libraries from the `src/main/webapp/WEB-INF/lib` directory to the java path during the deployment.

TODO: For the integration, it will also require to update the `doFilter` method of `ProxyFilter.java` file so that it can capture the supplied parameters according to the APIs of the DS component and pass it to the corresponding API of the RI.

8.1.2 Chaincode

The chaincode components exposes the endpoints for the PVE (Policy Violation Engine) and the agent. The PVE is an integrated component of the FRM used to analyse the access logs. The agent endpoint is used by the FSA to forward alerts to the RI.

Development model

The current iteration of the chaincode component of FRM leverages the hyperledger fabric blockchain and the chaincode represents the smart-contract which are executed on the fabric blockchain.

This component has been developed using `node.js`. The flow control in the registry interface is as follows:

```
SUNFISH Component ==> frm.js --> *API.js --> hyperledger/hyperledger*.js ==> ↵  
↵ fabric ==> SUNFISH Component
```

The `frm.js` is the entry point of the chaincode component. There are different `hyperledger*.js` files inside the `hyperledger` directory; each of which is responsible for interacting with a particular smart-contract. There are also different `API.js` files which are responsible for forwarding each request to the appropriate `hyperledger.js` file. Currently, these `*API.js` files are configured to hyperledger. However, if needed, this configuration can be changed in the `config.ini` file and also by developing required `*.js` files which interact with the other blockchain.

A SUNFISH component submits a request following the SUNFISH RI specification. Based on the request path, the request is forwarded internally to the appropriate `API.js` file. Then this file forwards the request to the corresponding `hyperledger.js` file where the request is handled.

Deployment guide

Follow steps are required to deploy any chaincode.

1. Prepare the hosting machine by following the instructions at: <http://hyperledger-fabric.readthedocs.io/en/latest/prereqs.html>
2. Setup your `GOPATH` environment variable as required.
3. Clone the Registry repository using the following command:

```
https://github.com/sunfish-prj/Federation-Monitoring.git
```

4. `cd` into `Federation-Monitoring/chainComponent` directory.

3. Configure the IP address of the docker container and the id of the PVE in the config.ini file.
4. In a terminal, within the *Federation-Monitoring/chainComponent* directory, the following command needs to be issued. This starts the node server for the FRM, listening at port 8077.

```
node frm.js
```

5. Wait until the *server started* output is printed in the terminal. This indicates that the node server for Infrastructure RI has been successfully started.

This is a page for Intelligent Workload Manager (IWM).

9.1 Overview of Intelligent Workload Manager

SUNFISH Federation provides automated services for joining and leaving the federation, as well as an interface to the available Federation services for a Service Consumer with ability to request optimized service list of services matching Consumer requirements better. A common aspect of all use cases is requirement to be able to retrieve information about the Federation resources and optionally schedule execution of a workload on a particular service provider. Within SUNFISH, a component responsible for delivering such functionality is called **Intelligent Workload Manager (IWM)**.

Optimization model applicable to the scenario of Service provisioning by Service Consumer is offering an improvement over local scheduling while imposing as little as possible of additional overhead on the definition of the workload requirements. Improvement means achieving a better outcome regarding user-defined parameters (e.g. cost) while preserving the strict requirements for the job payload. The goal of the model is to offer an added value over local scope of resources by finding and managing a globally optimal target for the Service Consumer's planned workload.

Optimisation model is a logical component exposed to the user in form of an optional ordering and filtering capability used during provider lookup request.

IWM is based on open-source Waldur cloud brokerage platform. The latter is extended to include more fine-grained optimisation capability. The functionality developed within SUNFISH has been integrated with the upstream.

9.2 Screenshots

Screenshots below are taken from a demo deployment of IWM in a federation.

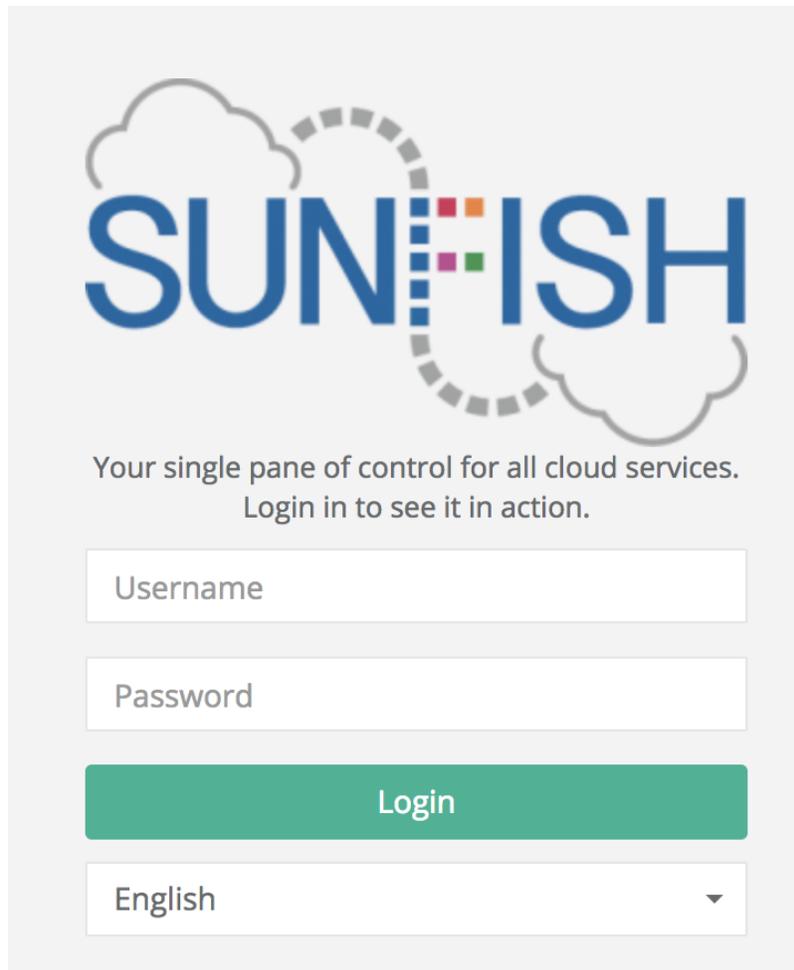


Fig. 9.1: Login view of IWM frontend, white-labelled to a concrete federation.

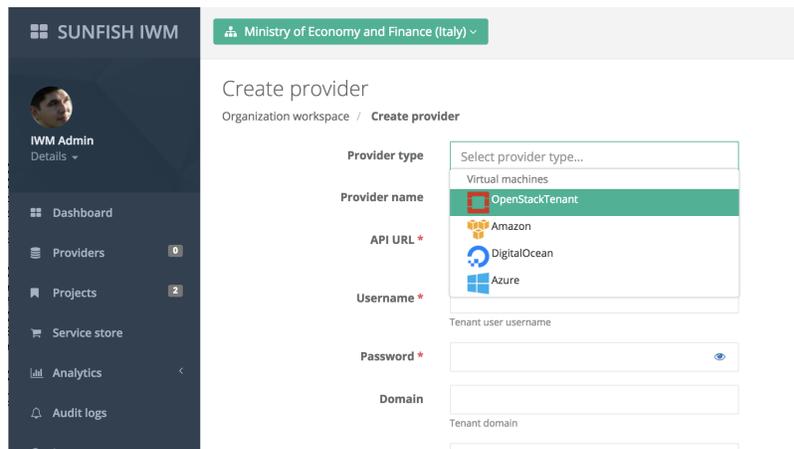


Fig. 9.2: Adding federation service providers to IWM.

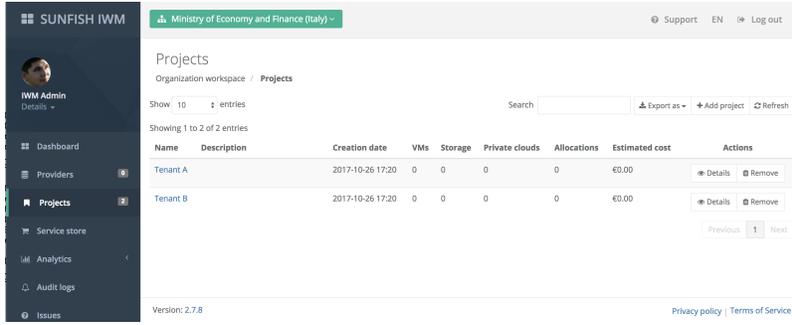


Fig. 9.3: Listing registered SUNFISH tenants within an IWM.

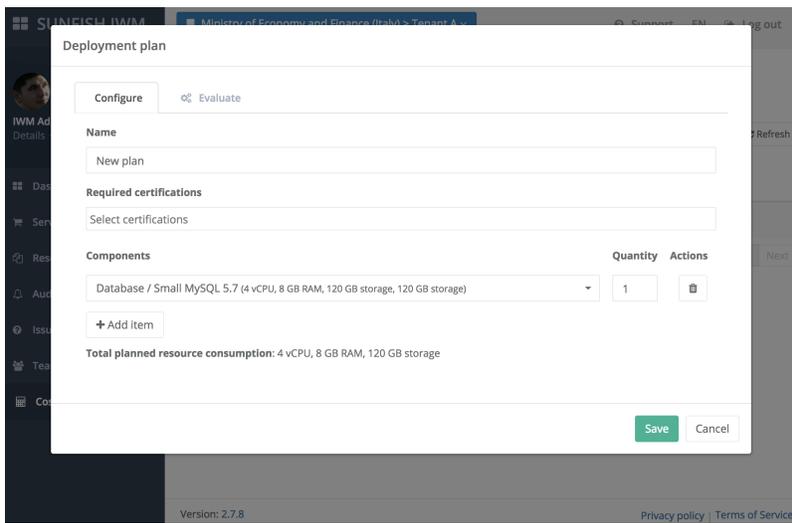


Fig. 9.4: Visual interface to optimisation API for finding the best option for a planned infrastructure.

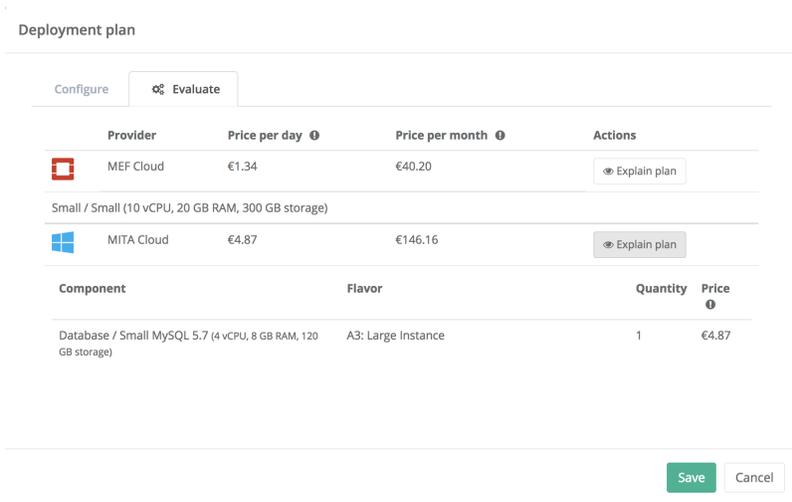


Fig. 9.5: Results of the optimisation with 2 service providers in the federation.

9.3 Instructions for deploying IWM

IWM functionality has been integrated into Waldur. As such, deployment of IWM is done in the same fashion as upstream. Installation script is below. Deployment requirements are:

- CentOS 7 or other RHEL7-compliant operating system
- At least 8GB of RAM, preferably 2 cores or more.

```
yum clean all
yum -y update

# Configure repositories
yum -y install epel-release
yum -y install https://download.postgresql.org/pub/repos/yum/9.5/redhat/rhel-7-x86_64/
↪pgdg-centos95-9.5-2.noarch.rpm
yum -y install https://opennodecloud.com/centos/7/elastic-release.rpm
yum -y install https://opennodecloud.com/centos/7/waldur-release.rpm

# Set up PostgreSQL
yum -y install postgresql95-server
/usr/pgsql-9.5/bin/postgresql95-setup initdb
systemctl start postgresql-9.5
systemctl enable postgresql-9.5

su - postgres -c "/usr/pgsql-9.5/bin/createdb -EUTF8 waldur"
su - postgres -c "/usr/pgsql-9.5/bin/createuser waldur"

# Set up Redis
yum -y install redis
systemctl start redis
systemctl enable redis

# Set up Elasticsearch
yum -y install elasticsearch java

systemctl start elasticsearch
systemctl enable elasticsearch

# Set up Logstash
yum -y install logstash

cat > /etc/logstash/conf.d/waldur-events.json <<EOF
input {
  tcp {
    codec => json
    port => 5959
    type => "waldur-event"
  }
}

filter {
  if [type] == "waldur-event" {
    json {
      source => "message"
    }
  }

  mutate {
```

```
    remove_field => [ "class", "file", "logger_name", "method", "path", "priority",
↪ "thread" ]
    }

    grok {
      match => { "host" => "%{IPORHOST:host}:%{POSINT}" }
      overwrite => [ "host" ]
    }
  }
}

output {
  elasticsearch { }
}
EOF

systemctl start logstash
systemctl enable logstash

# Set up Waldur Core
yum -y install waldur-core

su - waldur -c "waldur migrate --noinput"

systemctl start waldur-uwsgi
systemctl enable waldur-uwsgi

systemctl start waldur-celery
systemctl enable waldur-celery

systemctl start waldur-celerybeat
systemctl enable waldur-celerybeat

su - waldur -c "waldur createstaffuser -u admin -p admin"

# Set up Waldur MasterMind
yum -y install centos-release-openstack-pike
yum -y install waldur-mastermind

su - waldur -c "waldur migrate --noinput"

systemctl restart waldur-uwsgi
systemctl restart waldur-celery
systemctl restart waldur-celerybeat

# Set up Waldur HomePort
yum -y install waldur-homeport

# Set up Nginx
yum -y install nginx

systemctl start nginx
systemctl enable nginx
```


CHAPTER 10

Registry Interface

The Registry Interface (RI) is the logical component that manages the storing and retrieval operations directed to the blockchain based registry.

The RI offers a set of APIs that the different components, if authorised, can invoke to store or retrieve data from the blockchain-empowered registry. The specifications of the APIs can be found at: <https://github.com/sunfish-prj/SUNFISH-Platform-API>

RI, thus, is a central component of the SUNFISH infrastructure. Other respectice SUNFISH components need to leverage the RI to store and update corresponding data by invoking the defined endpoints of the APIs.

CHAPTER 11

Registry

Registy provides a blockchain-empowered storage and programming platform that is used to store and update relevant data to/from the blockchain via the Registry Interface (RI). It consists of several chaincode (smart-contract) which are programs executed within the blockchain platform. It consists of several chaincode with each one used for a specific purpose.