
SUNFISH Platform Documentation Documentation

Release 0.9

SUNFISH Consortium

Mar 13, 2018

1	Cloud Computing and the Public Sector	3
1.1	Adoption of Cloud Computing: report and challenges	3
1.2	Benefits for the Public Sector	5
2	The SUNFISH approach	7
2.1	Federation-as-a-Service	7
2.2	Service Ledger	8
2.3	The SUNFISH Platform	9
3	Platform components	11
3.1	Identity Management (IDM)	12
3.2	Data Security (DS)	12
3.3	Federated Administration and Monitoring (FAM)	12
3.4	Intelligent Workload Management (IWM)	12
3.5	Data Masking (DM)	12
3.6	Anonymization (ANM)	13
3.7	Federated Runtime Monitoring (FRM)	13
3.8	Federated Security Audit (FSA)	13
3.9	Secure Multi-party Computation (SMC)	13
3.10	Service Ledger (SL)	13
4	Use Cases	15
4.1	Cross-cloud payslip calculation	15
4.2	Private-Public Cloud integration to underpin tax calculation	16
4.3	Secure cross-cloud data sharing	17
5	Anonymisation (ANM)	19
5.1	Functionality	19
5.2	Anonymisation Interface (ANI)	19
6	Data Masking (DM)	23
6.1	Functionality	23
7	Data Security (DS)	25
8	Federated Administration and Monitoring (FAM)	27
8.1	Service Level Agreement Manager (SLAM)	29

8.2	Configurator & Deployment Manager	30
9	Federated Runtime Monitoring (FRM)	31
9.1	Proxy	31
9.2	Chaincode	32
9.3	Policy Violation Engine	33
10	Federated Security Audit (FSA)	35
10.1	Functionality	35
11	Intelligent Workload Manager (IWM)	37
11.1	Azure Integration	38
12	Secure Multiparty Computation (SMC)	39
12.1	Intro to SMC	39
12.2	Sharemind	40
13	Service Ledger (SL)	43
13.1	Architecture	43
14	Platform API	47
14.1	Anonymisation (ANM)	47
14.2	Anonymisation Interface (ANI)	51
14.3	Data Masking (DM)	52
14.4	Policy Administration Point (PAP)	55
14.5	Policy Decision Point (PDP)	57
14.6	Policy Enforcement Point (PEP)	59
14.7	Policy Information Point (PIP)	60
14.8	Policy Retrieval Point (PRP)	62
14.9	Federated Administration Monitoring (FAM)	65
14.10	Federated Runtime Monitoring (FRM)	66
14.11	Intelligent Workload Manager (IWM)	67
14.12	Secure Multi-party Computation (SMC)	141
14.13	Service Ledger (SL)	144
14.14	Service Ledger Interface (SLI)	145
15	Networking Infrastructure	157
15.1	Network Component	157
16	Federated Administration Monitoring (FAM)	165
16.1	Dependencies	165
16.2	Administration Manager set-up	165
16.3	SLA Manager set-up	166
17	Configurator	167
17.1	Installation Steps	167
17.2	Installation	171
18	Data Security (DS)	173
18.1	Setting-Up a Service Tenant	173
18.2	Setting-Up an Infrastructure Tenant	177
19	Intelligent Workload Manager (IWM)	179
19.1	Deployment instruction	179
19.2	Screenshots	181

20	Anonymisation (ANM)	185
20.1	Anonymisation Interface (ANI)	185
21	Data Masking (DM)	187
22	Federated Runtime Monitoring (FRM)	189
22.1	Proxy	189
22.2	Chaincode	192
23	Federated Security Audit (FSA)	195
24	Secure Multiparty Computation (SMC)	197
24.1	Sharemind MPC Application Server	197
24.2	Sharemind Web Application Gateway	200
25	Service Ledger (SL)	201
25.1	Dependency	201
25.2	Service Ledger Interface	202
25.3	Service Ledger	202
25.4	Usage Guide	202
26	UC-1: Cross-cloud payslip calculation	203
27	UC-2: Private and Public Clouds for Tax Calculation	205
28	UC-3: Federation-based Intelligent Shared Index	207
28.1	Deployment Instructions	207

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 H2020 SUNFISH Project.

Note: The SUNFISH project has been supported by the H2020 Programme under the grant agreement N. 644666.

Cloud Computing and the Public Sector

Cloud computing is drawing wide attention in the *Public Sector*. Nowadays main bodies rely on their own private clouds, leading to a multitude of secluded, not-interoperable cloud centres. The lack of reliable cross-cloud infrastructure hinders effective and practicable exploitation of clouds in the Public sector.

The *SUNFISH Project* has built upon this need by providing *a software platform that via the principled usage of a blockchain infrastructure offers decentralised, democratic and secure federation of private clouds*.

1.1 Adoption of Cloud Computing: report and challenges

Cloud computing has been part of the computing landscape for more than 10 years and it is lately increasing its deployment within businesses and individual costumers. It provides substantial benefits in particular by offering:

1. **economic growth** by providing an IT environment where technology is located in the most efficient way;
2. **more choice and lower cost**, increasing competition among providers.

While in a rapidly evolving environment, cloud services are one of the cheapest means to secure a large part of e-Government services. Cloud computing overcomes barriers typical to the public sector by improving features of effectiveness, efficiency, transparency, participation, data sharing, cooperation, interoperability and security. **Cloud computing solutions are now among the most innovative tools** and their adoption within European public sector organisations would allow them to take thrilling advantages from their adoption.

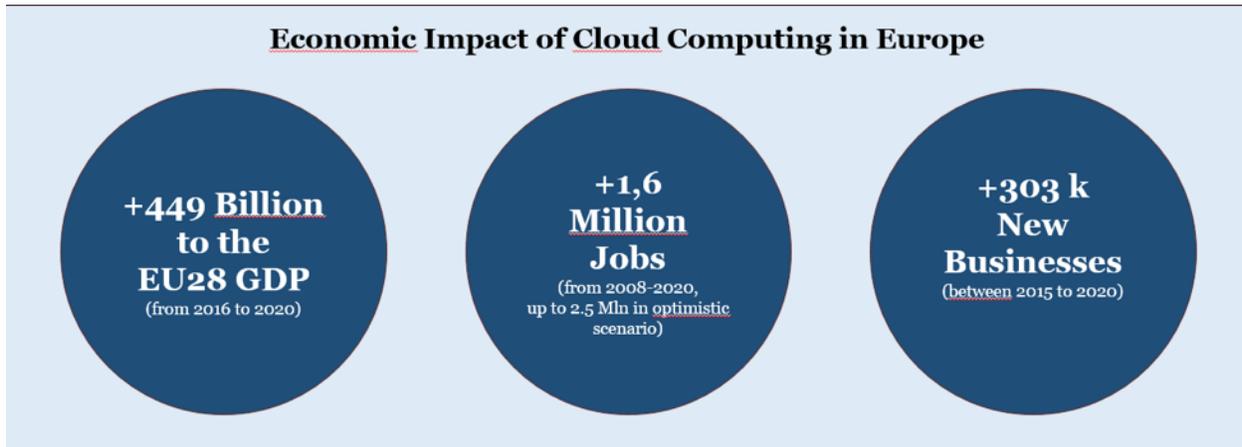
The interest in adoption of cloud computing solutions also for delivering Public Administration services has been emerging as a key target in the design of next generation public services. This process requires the selection of the most suitable solutions in order to fill into the current public sector technological gap, and to be able to face the challenges of the “EU Digital Market” for the next years. Cloud computing is surely a key enabling technology in order to improve efficiency and cost effectiveness while deploying new public services.

Cloud infrastructure is capable of introducing in the public sector mechanisms fulfilling citizens’ demands and it is particularly interesting when applied to support the provision of governmental applications provided to citizens by public authorities.

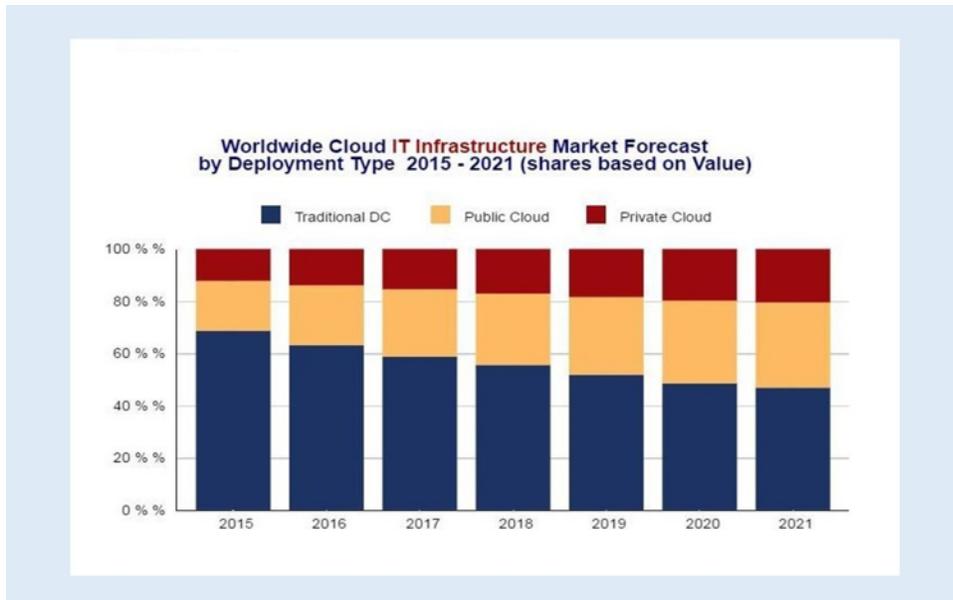
The European Commission study - “*Measuring the economic impact of cloud computing in Europe, 2016*” - estimated that in the period 2016-2020, cloud computing could add a cumulative total revenue of EUR 449 billion to the EU28

GDP (including in the Public Sector). Of these EUR 103,2 billion would be net new GPD generated in the year 2020, representing a share of 0,71% of total EU GDP.

According to this study, the **cumulative impact on employment is expected to reach 1,6 million jobs created up to 2020** (ranging from 2,5 million according to the optimistic scenario and slightly over 1 million in the pessimistic scenario). In terms of business creation, approximately 303.000 new businesses, in particular SMEs, could be created between 2015 and 2020 through the development and deployment of cloud computing.



According to IDC's Worldwide Quarterly Cloud IT Infrastructure (2016), traditional datacenters in 2017 weigh nearly 60% of IT infrastructure, while the remaining 40% are on clouds (with about 25% of public clouds). In 2021, the situation will reverse: traditional data centers will weigh about 45%, while the cloud for nearly 55%, of which about 35% will be public cloud while 20% private. However, according to Gartner (2016) by 2020 hybrid cloud will be the most common use of the cloud.



Clear examples of how governments are embracing this strategy are provided by the EU Regulation 2016/679, with the European Cloud Strategy, and national strategies and plans such as

1. the Italian AGID's [three-year Plan](#) for 2017-2019 and the [Cloudify program](#) of NoiPA;
2. the British government's cloud computing plan (*G-cloud* [_](https://www.gov.uk/guidance/the-g-cloud-framework-on-the-digital-marketplace));

3. the French Guide sur le Cloud Computing et les Datacenters à l'attention des collectivités locales;
4. the Spanish Líneas estratégicas del plan de Administración Electrónica del Gobierno.

1.1.1 Challenges

However, the migration process is not always a smooth procedure given several issues can emerge when the transition towards Cloud solutions is applied to the Public Sector. Main concerns in the adoption process for public organisations are the following:

- governance and control of ICT systems also across different Public Administration bodies;
- application of the concept of Quality of Experience (QoE) also to cloud services;
- ownership and asset liability;
- security, privacy and trustworthiness;
- resilience of infrastructures and services;
- interoperability and standards;
- dependencies with vendors;
- National and Supranational regulation.

The migration process towards Cloud solutions consists mainly of four pillars:

1. selection of applications/services that have to be migrated;
2. technical and process challenges;
3. backward compatibility with legacy applications;
4. operational cloud setup.

Given such a revolutionary context characterising new public services delivered via digital means, the SUNFISH project consortium partners have been contributing to this challenge by focusing on cloud development for the public sector.

1.2 Benefits for the Public Sector

According to up-to-date studies, the main common benefits of the adoption of Cloud computing solutions for a public sector organisation are:

1. **Cost effectiveness:** the use of Cloud computing solutions do not have all the maintenance costs that physical data centres do have. Especially in the public sector there is the need to rationalize public expenses and cost savings are a significant key factor;
2. **Sustainability & Green saving:** most data centres are environmentally and economically unsustainable due to their scarce energy consumption efficiency. The adoption of Cloud computing solutions, would allow on one hand to cut CO2 emissions, through a reduction of hardware use, and energy consumption thanks to the use of more efficient cooling systems; on the other hand public sector organisations would be able to pay the Cloud solution less than a data centre one, as there is a huge saving in energy consumption even for the providers.
3. **Ease of Implementation:** public sector organisations can deploy cloud computing rapidly as there is no need to purchase hardware, software licenses, or implementation services;

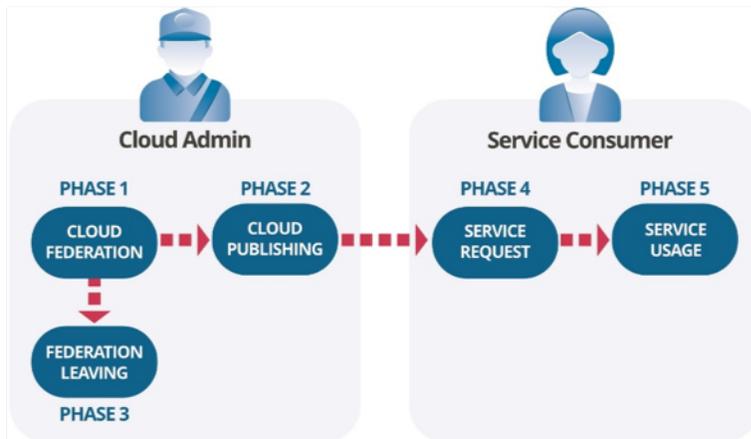
4. **Flexibility:** cloud computing solutions offer more in matching ICT resources to business functions than past computing methods. It can also increase staff mobility by enabling access to business information and applications from a wider range of locations and devices, enabling public sector employees to easily access data even if they are out of office through any kind of device;
5. **Innovation:** innovation represents a deep need of public sector organisations, as most of the time they lack innovation processes. Once the Cloud is adopted, its architecture would facilitate services across systems and organisational borders, such as the exchange of data among different administrations of the same public sector organisation;
6. **Scalability:** public sector organisations adopting cloud computing solutions don't need to procure any additional hardware and software when users' loads are increasing, but can instead simply add and subtract capacity to the Cloud when and if needed. In this way, resources are used only when needed;
7. **Redeployment of IT personnel due to Cloud efficiency:** by reducing or eliminating constant server updates and other computing issues, and consequentially cutting expenditures of time and money, public sector organisations can relocate ICT personnel on higher-value tasks;
8. **Focusing on Core Competencies:** the ability to run data centres and to develop and manage software applications is not necessarily a core competency of most public sector organisations. Indeed, the adoption of Cloud computing solutions can make it much easier to reduce these functions, enabling public sector organisations to concentrate on critical issues such as the development of policy and the delivery of public services;

The SUNFISH approach

This page outline the overall SUNFISH's approach to Cloud Computing solution for the Public Sector.

2.1 Federation-as-a-Service

The SUNFISH project coined **Federation-as-a-Service (FaaS)** a secure-by-design Cloud federation solution that enables public sector organisations to federate their clouds in a distributed and democratic manner, thanks to an underlying blockchain infrastructure.



FaaS creates a **homogenous goal-oriented aggregation of cloud systems**, which allows sharing of data and services. All participating nodes are peers: they enjoy the same duties and authorities.

The corner stone of FaaS is its **democratic and decentralised federation governance**. Generally speaking, it offers the these key features:

- **Dynamic Federation of Clouds** and their services with service level agreement policy and optimal workload strategies;

- **Cloud Federation Governance** supporting trust-less coalitions where participating clouds are governed by a federation contract agreed with a distributed consensus;
- **Privacy-Preserving Services** enforcing an advanced and innovative access control and monitoring.

Note: Blockchain technology

It has appeared on the market in recent years, firstly used as public ledger for the Bitcoin cryptocurrency. It mainly consists of consecutive chained blocks containing records that are replicated and stored by nodes of a peer-2-peer network. The records witness transactions occurred between the nodes of the network. Transactions may feature a cryptocurrency like, e.g., the Bitcoin, or other kinds of assets. The collection of transactions and their enclosing in chain blocks is carried out in a decentralised fashion by distinguished nodes of the network, called miners.

Besides cryptocurrency, blockchain offers so-called **smart contract**, immutable program deployed and executed autonomously upon a blockchain.

FaaS uses blockchain technology to offer a decentralised computation infrastructure at hand that alleviates the need for a trusted-third-party and reduces systemic risk of disputes and frauds.

More specifically, FaaS offers functionality to both administrator and service consumers. The phases concern both administrator-side operations, e.g. the creation of a cloud federation, and service consumer- side operations, e.g. the request of a service.

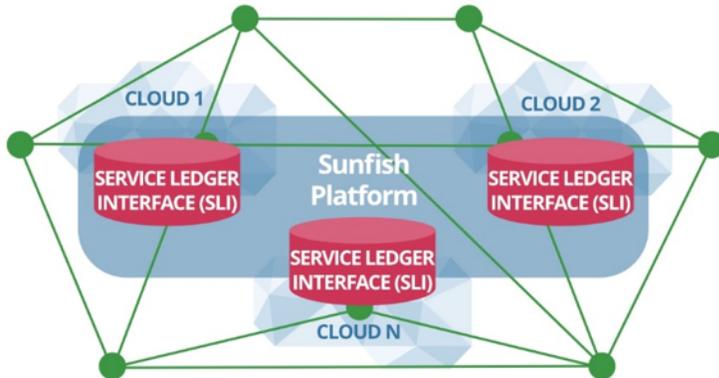
Following the previous figure, these functionalities can be grouped according to the following operating phases:

1. *Cloud Federation*: it refers to the functionalities that permit creating a cloud federation and enabling the joining of new members to an already created federation.
2. *Service Publishing*: it refers to the functionalities that permit: (i) registering a service offered by a federation member; (ii) making available the registered service to the other federation members.
3. *Federation Leaving*: it refers to the functionalities that permit a federation member to leave the federation. Leaving can be imposed when the business contract underlying the federation has been violated.
4. *Service Request*: a service consumer willing to use a service offered by a FaaS federation (i.e., any service from IaaS to SaaS) has first to submit a service request. This preparation phase has a twofold objective. On the one hand, it permits controlling the authentication and authorisation of the consumer and, on the other hand, it permits selecting the optimal service provider for such a request.
5. *Service Usage*: it amounts to the actual usage of the already requested, hence set up, service. During this phase, the provisioning of the service has to ensure that the corresponding access control policies are correctly enforced.

2.2 Service Ledger

The Service Ledger is a blockchain-empowered software layer which **underpins the whole FaaS federation** offering a decentralised (hence without any centralised point-of-failure) platform upon which basing highly trusted services such as the *governance of the federation* or *cross-cloud piece of computation*.

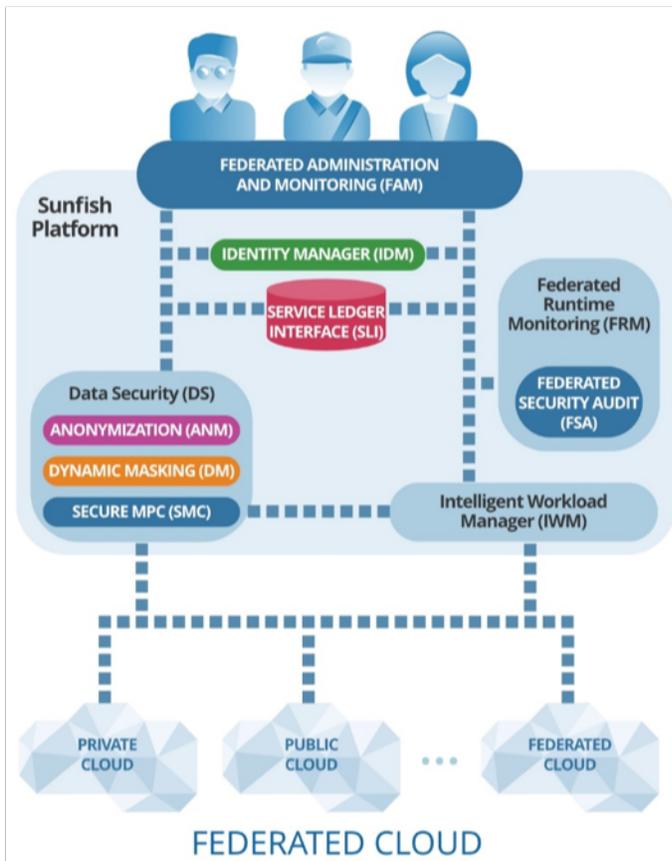
Faas Federation BLOCKCHAIN-BASED SERVICE LEDGER



On the fact of it, FaaS and the SUNFISH platform appear to be the **first blockchain-based cloud federation architecture of its denomination**.

2.3 The SUNFISH Platform

The SUNFISH Platform is a modular software solution that enables the dynamic and secure creation of cloud federations and their management.



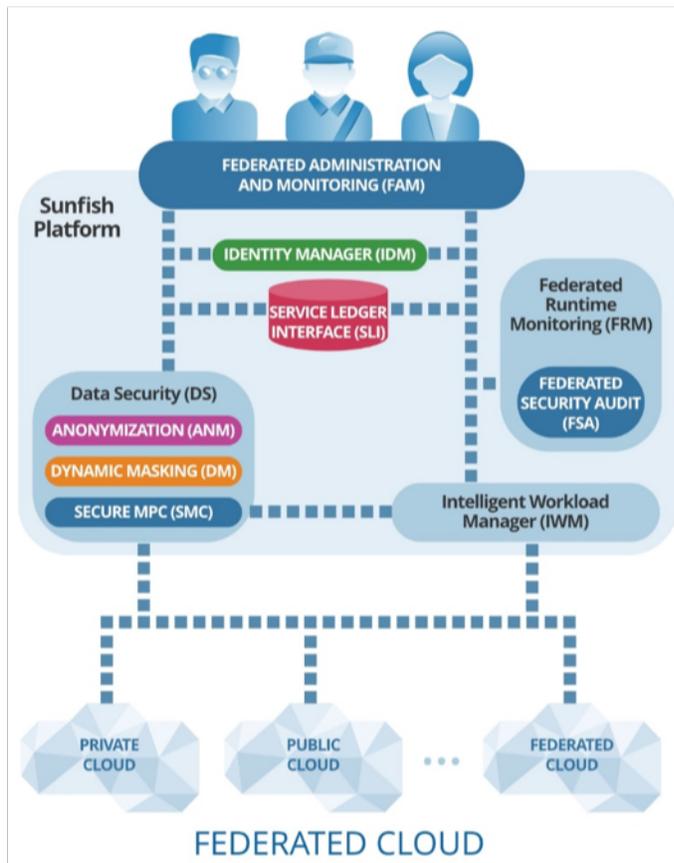
Its main features are:

- **Dynamic cloud federation management.** A dynamic federation of clouds and their related services, with optimal service level and workload; the service offered is based on “core components” (IDM, DS, FAM, IWM) that are necessary to assure the creation and essential federation management.
- **Democratic governance.** An innovative cloud federation governance supporting trust-less coalitions, as none of the federated organizations rules on the others, thanks to the “Service Ledger” which - based on the blockchain technology - offers decentralised and democratic enforcement of governance rules in the federation. Such governance is then ruled according to a federation contract negotiated among partners.
- **Data security.** Advanced, innovative privacy-preserving services ensuring high security of provisioned services and managed data. The backbone of data security is a distributed access control infrastructure transparently enforcing cross-cloud access policies and privacy-preserving services. Data masking (DM) ensures that sensitive data can be securely stored protecting sensitive data of interest. Data anonymisation (ANM) ensures that datasets can be released (both in a micro and macro fashion) without leaking sensitive data. Secure multi-parties computation (SMC) offers privacy-preserving computation of sensitive data: any of the party involved in a computation can learn anything on the data itself.
- **Brokerage of Federated services.** Services of single clouds can be dynamically federated and brokered according to security and Service Level Agreement (SLA) policy. Due to an intermediate layer of API, any type of service, ranging from infrastructure to software and data, can be federated and provided.
- **Federation Monitoring.** Cross-cloud integrations and distributed nature of federations require advanced monitoring facilities that can ensure the integrity and correctness of the provisioned services. FRM and FSA offer runtime and offline monitoring respectively to protect from security violations such illegitimate accesses and privilege escalations.

This set of functionalities is implemented via state-of-the-art technology and take advantage of the blockchain infrastructure underlying SUNFISH federations to strengthen security assurance of provisioned services and security controls.

Platform components

The SUNFISH Platform is a modular software solution that enables **dynamic and secure creation of cloud federations and their management**. Its components interact between them to establish the FaaS approach but their use is not limited to this as all of them can be deployed independently.



3.1 Identity Management (IDM)

Software providing a set of services to authenticate the access to and within a FaaS federation. It supports the authentication of all the entities part of the federation, varying from users and administrators to service providers and platform components. The added value is that by abstracting the existing IDM solutions to pre-agreed roles in the federation, it enables a flexible definition of data access and data usage policies. This enables the setup of a SUNFISH federation, that uses the pre-deployed IDM solutions. Furthermore, the platform is integrated with eIDAS and it enables a pan-European user authentication.

3.2 Data Security (DS)

Software aiming to enforce the access control policies associated with the federated services. Its main role is to decide whether to allow access requests concerning service requests and provisioning. Existing approaches focus on access control or data-usage control. The added value of this component is provided by combining these approaches and rely on developed advanced data protection mechanisms. Extensions are based on well known access control languages. Furthermore, this component intercepts the main communication channel and applies the defined policies, based on data as well as endpoint characteristics.

3.3 Federated Administration and Monitoring (FAM)

Software representing the logical entry-points for the management of a FaaS federation, hence, for the interaction with the SUNFISH platform components. It provides a front-end for the administrators and service consumers of the federation based on a graphical web interface. It permits the administration of member clouds (i.e., entering and leaving a federation), tenants (i.e., creation and deploying of tenants), service publishing (i.e., registering a service to the federation), and service provisioning (i.e., management of Service Level Agreement and access control policies).

3.4 Intelligent Workload Management (IWM)

Software performing service brokerage of the federated services offered by the member clouds. In particular, once a service consumer requests a service, it provides an optimal federation-based workload deployment target to satisfy such a service request. To actually deploy the workload, the IWM interacts directly with the clouds to create / delete virtual machines running on the federated clouds. The IWM can provide different workload management strategies optimised according to different parameters. It thus solves an optimisation problem based on the current state of the federation and the requested service. IWM offers a comprehensive set of services as a stand-alone component, with added value being: accent on end-user, pushing more IT governance to the edges, reducing operator's load, integration with blockchain Service Ledger offering higher guarantees against malicious data manipulation.

3.5 Data Masking (DM)

Software providing a generic service for masking in a selective way personal and/or sensitive information. This service is called masking service. The service, given a policy and payload (e.g. text, JSON, XML), results with a masked payload. The masked payload itself is identical in format and structure to the original payload except for the personal or sensitive information that is masked. The added value provided by the masking component is its combined support for selecting the sensitive elements and the actions performed (redaction, tokenization, encryption). Moreover, both the selection and action are highly configurable using a flexible policy.

3.6 Anonymization (ANM)

Software providing Micro data and Macro data anonymization services. Micro data anonymization: a data set is released with the k-anonymity guaranty. This process ensures the protection of sensitive information against linkage attacks using other open data sets. Macro data anonymization: statistical data are released with differential privacy guarantees. This process adds noise to the summary statistic such that the probability to identify if a single person is added or removed is extremely small. The added value of this component is its flexible support for different privacy guarantees when releasing a data set. Specifically the component allows the user to select the required privacy guarantee that best fits the use case: data perturbation, k-anonymity and crowd blending.

3.7 Federated Runtime Monitoring (FRM)

Software providing a distributed infrastructure to intercept (via transparent plug-in proxies) and monitor every access control request received and possibly authorised by the Data Security (DS). The added value lays in the usage of smart-contracts (programs that facilitate, verify, or respect the negotiation or execution of a contract).

3.8 Federated Security Audit (FSA)

Software providing an automatic detection service against vulnerabilities in the distributed access control system and against security breaches possibly occurred within the federation. The added value of this component is its use of Role Mining techniques to identify real needs from users' behaviour. This allows to identify vulnerabilities and security breaches with much more confidence.

3.9 Secure Multi-party Computation (SMC)

Software enabling privacy-preserving computation of sensitive data. It can be thus used for computing tasks on confidential data. The added value of this component is that it is integrated with the governmental backbone technology UXP (X-Road) already deployed in Estonia, Finland, Namibia, Haiti, Azerbaijan and ongoing deployment in Ukraine. This is a novel way of securely processing administrative data that enables to use private or public cloud resources.

3.10 Service Ledger (SL)

Blockchain-based infrastructure managing the storage and evaluation of governance data. Its seamless integration with the SUNFISH platform via the Service Ledger Interface component permits realising the FaaS “democratic” governance. The Service Ledger offers a set of APIs used by authorised components to invoke smart contracts deployed on the Service Ledger.

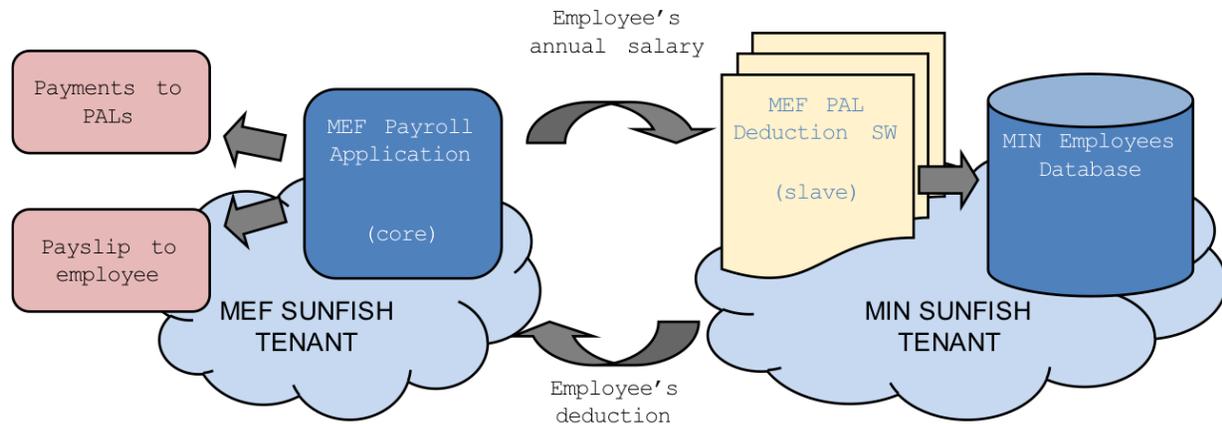
This is a page for the SUNFISH use cases, reporting the business rationale behind the use case.

4.1 Cross-cloud payslip calculation

The General Administration, Personnel and Services Department (DAG) of the Italian Ministry of Economy and Finance (MEF) is in charge of the management of payroll functions for approximately 2.1 million Italian public sector employees. Such service is provided through a unique payroll function, NoiPA - which currently manages annually more than €51 billion in payments. Starting in 2015, the compulsory entrance in NoiPA of Italian police and military personnel generated an increase of around 25% of the monthly payslips managed by the system.

The Italian legal framework forces the Ministry of Interior (MIN), in charge of Police Forces, to be the exclusive controller of sensitive data of its employees. The main problem generated by the entrance in NoiPA of MIN's employees was overcoming segregation of Public Bodies data among Clouds for calculating payslips.

In particular, the MEF must compute local taxes on actual residence, which is however sealed for data classification purposes within the MIN. The MEF and the MIN had therefore to balance two contrasting needs: on one side, the MEF's need to have certified computation of sensitive data, on the other side the MIN's need to keep sensitive data within its perimeter. This created a problem for the overall calculation of taxes and to overcome it the MEF and the MIN were forced to an intricate cooperation keeping low level of efficiency and elevated costs.



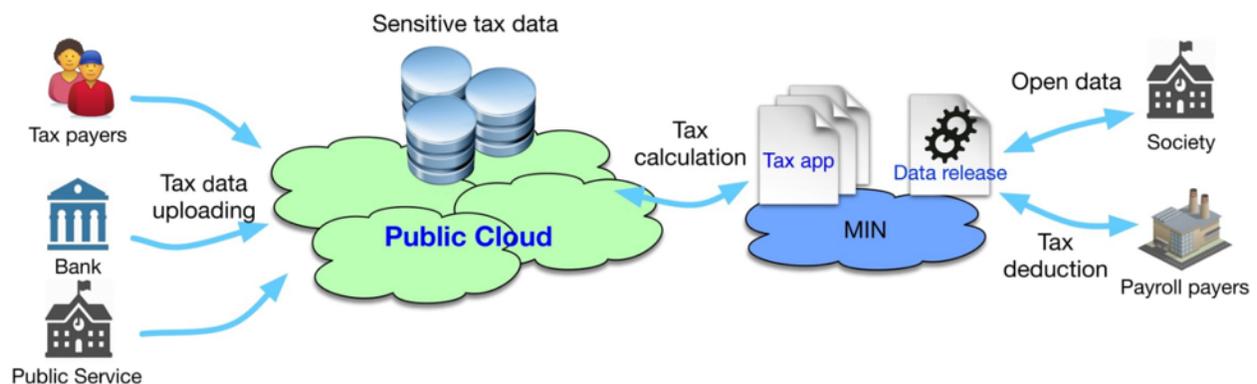
The potential conundrum was overcome via a Federation-as-a-Service platform and its blockchain-empowered Service Ledger infrastructure. This system, put in place by SUNFISH, allows the democratic governance of cloud federations: none of the federated clouds rules on the other, but each of them shares the same duty and authority

4.2 Private-Public Cloud integration to underpin tax calculation

The Maltese Ministry for Finance has been leading the innovation of the Country's Public Sector with the goal of easing citizens' interaction with the Government. This has been achieved by adopting a once-only principle and by facilitating the re-use of public data. Pushed by the growing awareness of central public administrations' need to promote a different role across Europe, the Ministry's interest for innovation led it to look for proposals in the field of cloud computing for the public sector.

Such kind of development is a great opportunity in particular within the Taxation Department, which requires taxpayers, employers, banks and SMEs, to submit information to the Office of the Commissioner for Revenue. This information relates to Payroll, Financial Statements, information related to payments that qualify for deduction from chargeable income, and receipts of payments that need to be included in taxable income, trading records and accounting records that maybe subject to audit checks.

Large enterprises can lean on their financial capability to submit payroll data and financial statements via the Department's website and await for the end of the year to receive their tax deduction back. Small businesses, on the other hand, might struggle because of their financial means.



Where applicable, the Department requires data from Employers and other third parties for the calculation of tax statements and eventual issuance of refunds. To provide a holistic solution, the Department can make use of public cloud services to host Software-as-a-Service (SaaS) solutions and to federate these with its own private cloud.

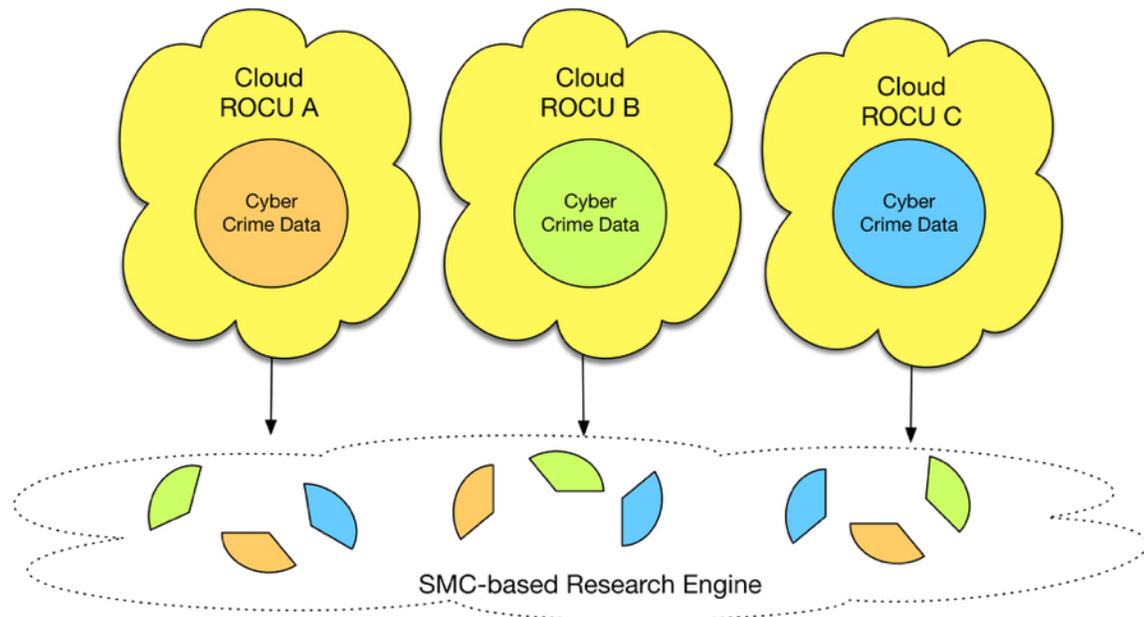
This use case enables the use of public cloud Platform-as-a-Service (PaaS) offerings to deploy applications which collect data and perform required calculations and validations while ensuring compliance with the secrecy, privacy and data protection legislations and regulations. It also allows for the use of federated systems between the MFIN application on the public cloud PaaS and other commercial SaaS solutions providing services such as payroll etc.

4.3 Secure cross-cloud data sharing

South East Regional Cyber Crime Unit (SEROCU) is one of the nine Regional Cyber Crime Unit (ROCU) operating across the UK. Besides its regional role, it collaborates on a national level with major crime units, all ROCUs and the National Cyber Crime Unit, to prosecute offenders based in Europe and beyond.

SEROCU is responsible for the investigation of offences categorised under the Misuse of Computer's Act 1990 and other offences where a digital aspect is believed to be involved. Its powers include the seizure and forensic examination of digital data and electronic devices, as well as live network investigations. Part of SEROCU's mandate is to store securely large quantities of cyber-crime evidences and highly sensitive data, such as: high-level corporate information, data produced from network servers and personal digital storage devices. Its investigations generate evidences with different security classifications, each of which, depending on Governmental guidance, comes with its own strict handling conditions.

The storage of such data must be localised on the Unit's premises but at the same time, each unit must ensure access, in a regulated manner and with different levels of accessibility, to all other ROCUs while investigations are in process. The sharing of such information among ROCUs not only encountered difficulties brought about the different interfaces implemented, but it was convoluted and hardly automated. Moreover, due to changing reporting procedures around cyber-crime issues, it is impossible to predict with certainty the future demand for the unit and, therefore, data capacity and processing requirements. There is a current need to ensure the efficient and secure reception, supply, and storage of intelligence/data between the regional units, local policing forces, and governmental departments.



Clouds have the ability to help overcoming concealed ROCUs data storage systems by fostering cross-Cloud regulated sharing of information. This would allow reaping the cost, usability and connectivity benefits of the cloud, whilst sharing the infrastructure safely and reliably between many different Government and Policing agencies.

Anonymisation (ANM)

The anonymization service provides the means to anonymize data. In particular it support Macro Anonymization - adding noise to queries - as well as Micro Anonymization which generalizes values for the release of personal records in data sets. The service provides REST API's that allow a user to configure the service, upload data, anonymize the data and finally download the anonymized data.

5.1 Functionality

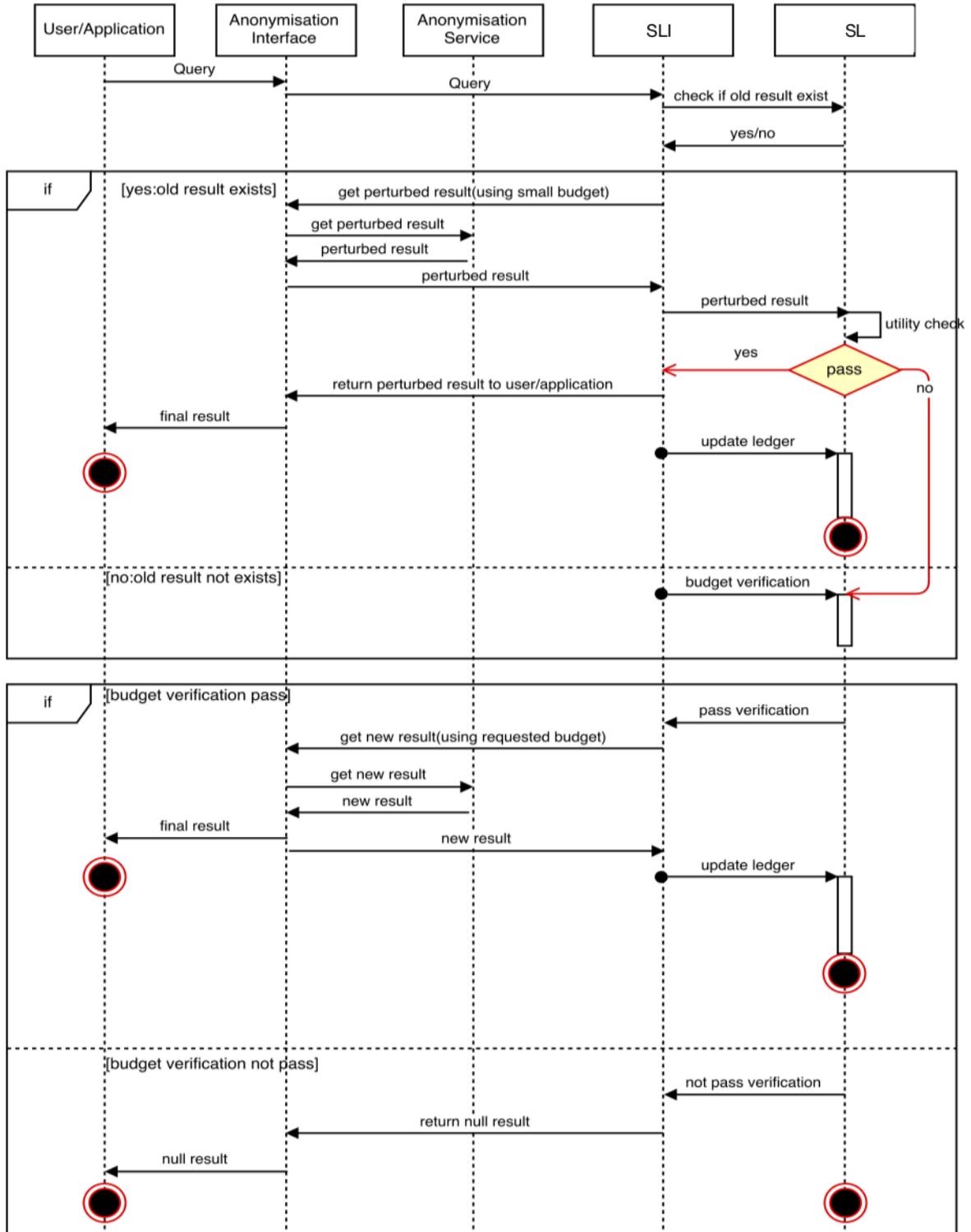
The anonymization service provided the necessary functionality to anonymize (micro, macro) a data set. It allows a privacy expert to configure and then perform the anonymization. The Anonymization flow is very simple:

1. Upload configuration (which contains necessary anonymization parameters as well as data format and data types)
2. Upload Data
3. Perform anonymization
4. Download anonymized data (for micro only)

All the functionalities are invoked via RESTful calls. Examples are provided in the deployment guide.

5.2 Anonymisation Interface (ANI)

Related to Macro anonymization, the statistical data API of ANM is also integrated with the Service Ledger blockchain to provide a greater level of control for privacy protection of data sets. This integration is realised via the Anonymisation Interface (ANI), whose high-level behaviour is here detailed



This blockchain-based anonymization approach relies on two phases:

1. Setup, to put in place privacy and data utility requirements, and
2. Runtime, to dynamically control privacy budgets and tune the data sharing process.

Blockchain smart contracts are used to store, evaluate and keep track of historical queries and privacy budget. These functionalities empower untrusted privacy services of a Cloud federation.

5.2.1 Setup

At the Setup phase, data owners provide their privacy and data utility requirements which are then stored in the smart contract. The privacy requirement is represented by the privacy budget, which represents the maximum amount of budget allowed on sharing data. According to data owner preferences, the budget can be associated to one or many datasets, or even to single columns of a single dataset. Data utility requirement is represented by a numerical variable, representing the maximum amount of noise allowed on the actual query result, thus to maintain adequate data utility.

5.2.2 Runtime

At the Runtime phase, data queries are managed returning anonymized results, when allowed by the privacy budget and requirements. Indeed, our approach M consists of an unbounded sequence of mechanisms M_1, M_2, \dots , where M_i operates when the i -th query is received. Logically, it can be decomposed into three main test activities:

1. Query matching: it aims at determining whether a newly received query has been executed before. Smart contract checks the sharing history stored on the blockchain;
2. Utility-based approximation: it aims at checking whether a previous released result can approximate the result to return for the current query;
3. Budget verification: this test is triggered if there has been no same query executed (i.e., the query matching test failed), or the query result cannot be approximated (i.e., the approximation test failed). Thus, a new result has to be computed, as long as the remaining privacy budget is enough.

The masking service provides the means by which sensitive data is replaced, possibly in a reversible and format preserving manner, with data that is unintelligible to receiver. The service supports parsing and classifying sensitive/confidential data in payloads, mask them in different ways and supports flexible configuration of the process.

6.1 Functionality

The masking service provided the necessary functionality to mask different data payloads (e.g. xml, json etc). The service enables to user to upload a policy which configures where the data items can be found in the payload (e.g. xpath, css etc.) and how to mask them (e.g. redaction, format preserving encryption). The Masking flow is very simple:

1. Upload policy
2. Create context
3. Process payload (mask/unmask)

All the functionalities are invoked via RESTful calls. Examples are provided in the deployment guide.

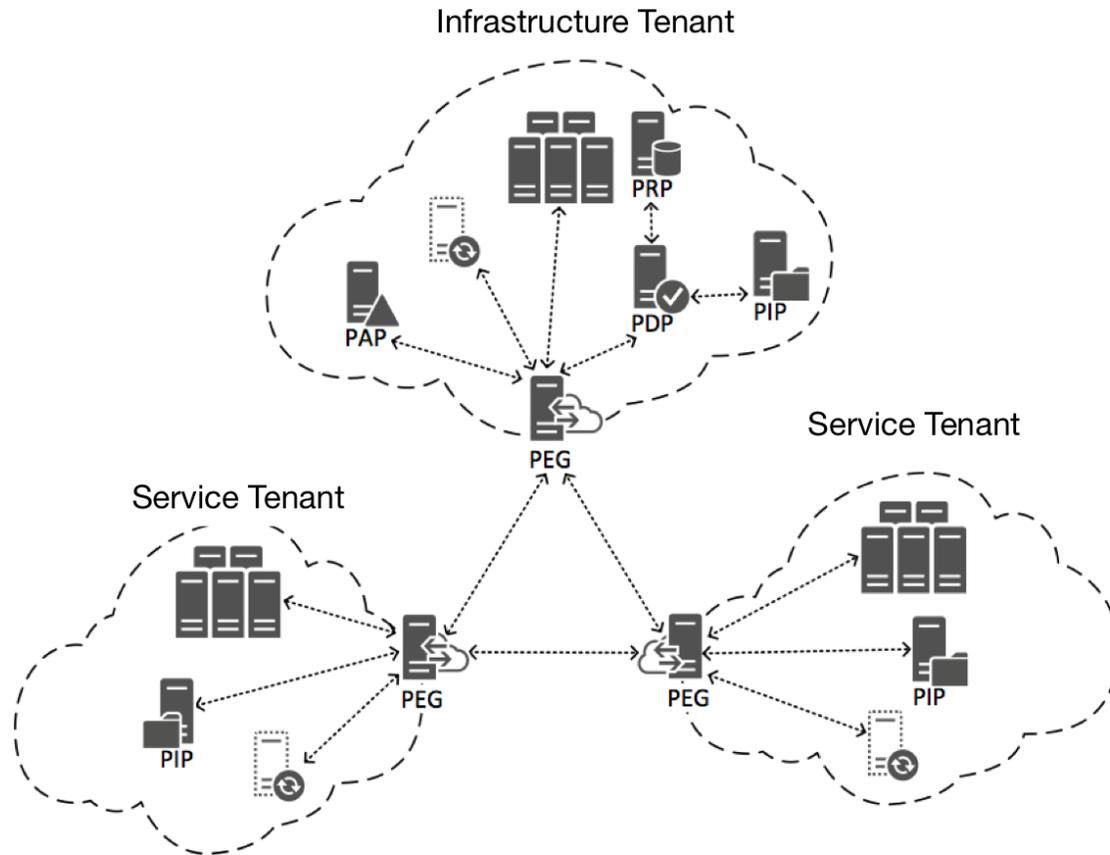
Data Security (DS)

The SUNFISH Data Security (DS) enforcement infrastructure is responsible for regulating and securing access to services in a federated cloud environment.

The infrastructure 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 **Service Ledger Interface** responsible for storing, managing and delivering policies to the PDP for evaluation. Setup an operation of the SLI is described separately, since it is operated independently of the other components.
- A **Masking Service** providing *data masking* capabilities to the PEG. Like the SLI, 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*. A minimal example consists of one *infrastructure tenant* and a *service tenant*. A typical cross-cloud set-up will be as follows, with highlighted typical inter-tenant interactions.



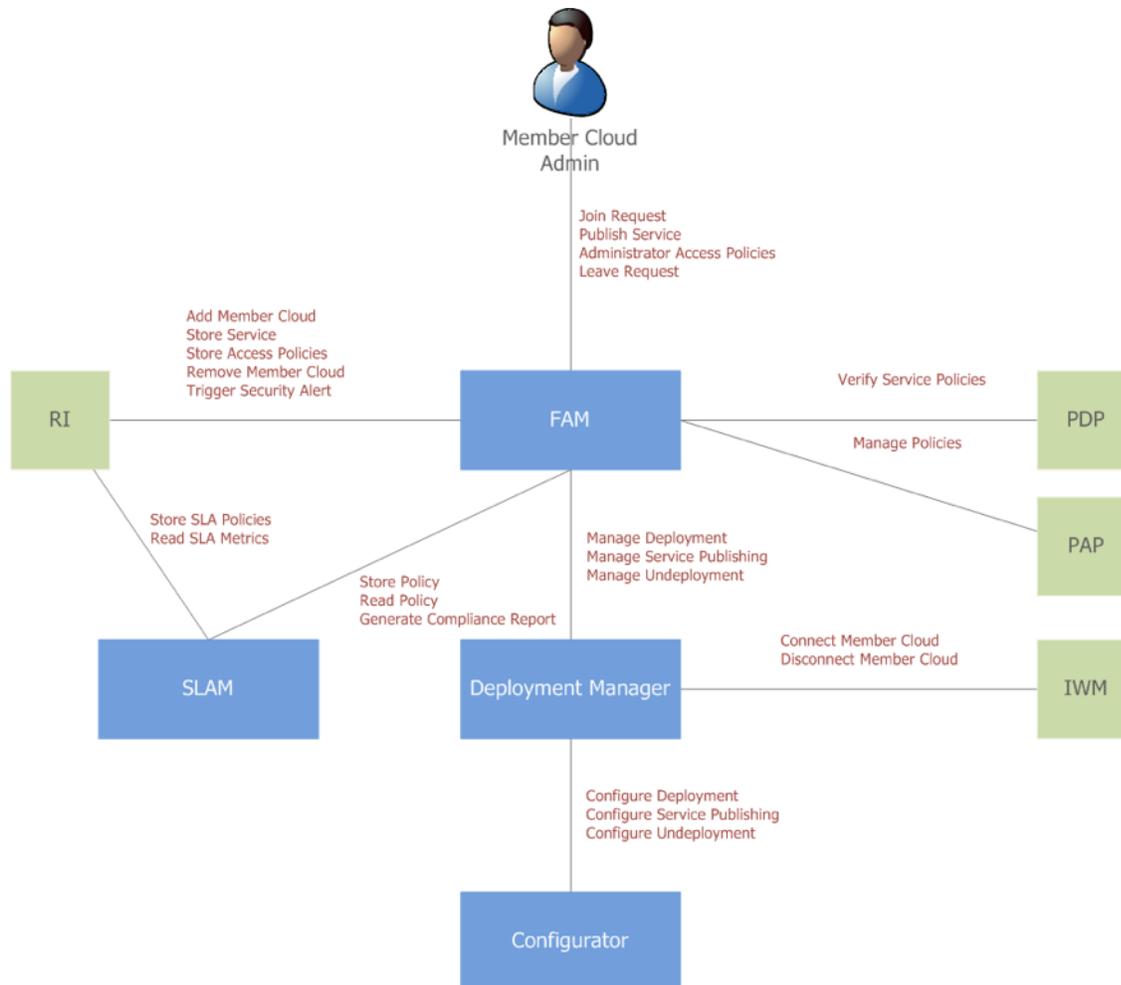
The *infrastructure tenant* will house the PDP, any number of PIPs and the PRP. The *service tenant* hosts the actual service to be protected by the enforcement infrastructure as well as the PEG, any number of PIPs and the proxy to maintain backwards compatibility to non-SUNFISH-aware clients.

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).

Federated Administration and Monitoring (FAM)

The Federated Administration and Monitoring (FAM) component is the management plane of the SUNFISH FaaS federation. It provides federation administrators with a graphical interface to configure, manage and monitor the services that are made available to the federation. The FAM captures the information submitted by the administrators, validates it, and dispatches that information to the respective low-level sub-components.

The FAM provides administrators with a set of functions such as to join and leave the federation, publish services and administrator access policies. It then relies on other SUNFISH components to execute the low-level instructions. The FAM has 3 sub-components being the Deployment Manager, Configurator and SLA Manager as depicted below.



The high-level interactions orchestrated by the FAM to federate a new member cloud (Cloud Federation phase, see *Federation-as-a-Service*) and to make an already federated member cloud leave the federation (Federation Leaving phase, see *Federation-as-a-Service*). The logic to coordinate these operations is encapsulated in the Deployment Manager component, which instructs the Configurator and consequently the IWM to execute the required actions.

Thus, the flow of interactions for the Cloud Federation phase can be summarised as follows

- The FAM asks the Deployment Manager to federate a new member cloud.
- The Deployment Manager instructs the IWM Adapter to establish a connection to such member cloud.
- The IWM interacts with this member cloud at the infrastructure level to install the software required to make it configurable for the next step.
- The Configurator deploys required SUNFISH components over the resources provided by the new member cloud, according to a deployment plan decided by the Deployment Manager.

Similarly, for the Federation Leaving phase, the high-level sequence of interactions is as follows

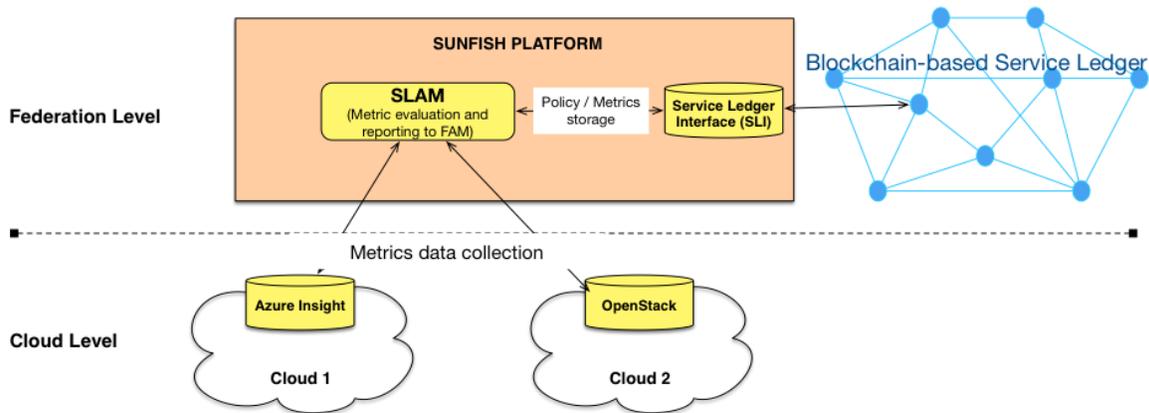
- The FAM asks the Deployment Manager to make an already federated member cloud leave the federation.
- The Deployment Manager commands the Configurator to un-deploy all the SUNFISH components currently placed on the resources provided by the member cloud.
- The Configurator applies such un-deployment and thus removes all the SUNFISH software currently installed on the member cloud.

- Then the Deployment Manager asks the IWM Adapter to remove all the software that were installed during the Cloud Federation phase to establish the initial connection.

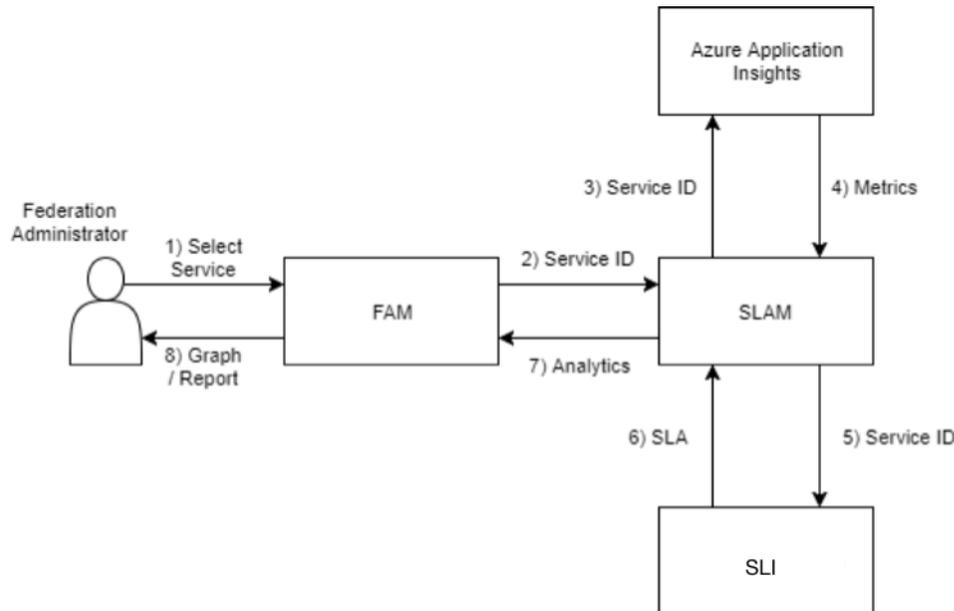
8.1 Service Level Agreement Manager (SLAM)

The Service Level Agreement Manager (SLAM) unit provides management of service level agreements (SLAs) and metrics for services within the federation. Administrators are able to monitor these services on the FAM graphical user interface via data representation graphs composed from the analytics provided by the SLAM. The use of SLAs identifies which services are not fulfilling the set requirements and notifies administrators about such services.

To realise the SLAM, we rely on products already available to realise SLA monitoring and adopt the SLALOM metrics specialised to the context of the Public Sector. The realised component is integrated with the SLA service Insight of Microsoft Azure (see the code [here](#)).



SLAs are used for analytics reports on services registered within the federation. Below is shown a detailed workflow between the FAM, SLAM and SLI components to construct and view analytical reports to federation administrators



The reported workflow can be described as follows

1. Federation administrator selects service via graphical user interface on the FAM.

2. The FAM sends the service ID along with query parameters to the SLAM and requests data analytics of the service.
3. The SLAM sends the service ID to Azure Application Insights and requests metrics for the service.
4. Azure Application Insights returns all available metrics for the given criteria.
5. The SLAM sends the service ID to the Service Ledger Interface and requests the service's SLA.
6. The Service Ledger Interface returns the SLA for the respective service.
7. Data analytics are sent back to the FAM.
8. The FAM builds the analytics into visual data representation.

8.2 Configurator & Deployment Manager

The Configurator component consists in a set of Infrastructure Services actuating commands and controls relative to the deployment and configurations of other Infrastructure Services.

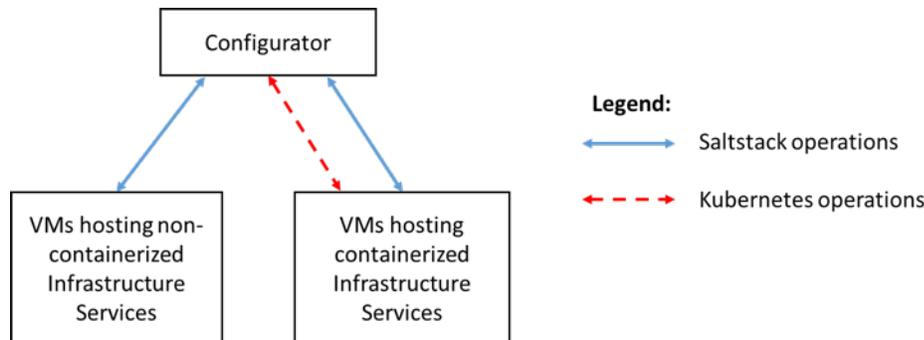
While the Configurator has not a decision-making role on the Deployment State of a federation, it provides APIs for its actuation, and for the retrieving of information sufficient for its determination. The Configurator provides the following functionalities:

- Automatic configuration of the VMs hosting non-containerised Infrastructure Services
- Automatic deployment and managing of the containerised Infrastructure Services among Nodes of a cluster

The Configurator relies on the following technologies:

- Configuration Management Engine: a software responsible of ensuring that a remote operating system is configured in terms of installed packages, presence of files and services. The solution utilised is Saltstack.
- Container: it represents an isolated, resource-controlled, portable operating environment where an application can be deployed. The solution utilised is Docker. Container Cluster Manager: it represents a software capable of orchestrating Containers among a set of nodes. The solution utilised is Kubernetes.

The functionality is therefore carried out according to the following logical connection



The Deployment Manager, properly instructed by the FAM, is utilised to properly instruct the configurator. Therefore, according to the current global state the Deployment Manager takes a decision on how to modify and set up a specific service via the Configurator.

Federated Runtime Monitoring (FRM)

FRM (Federated Runtime Monitoring) consists of the following components:

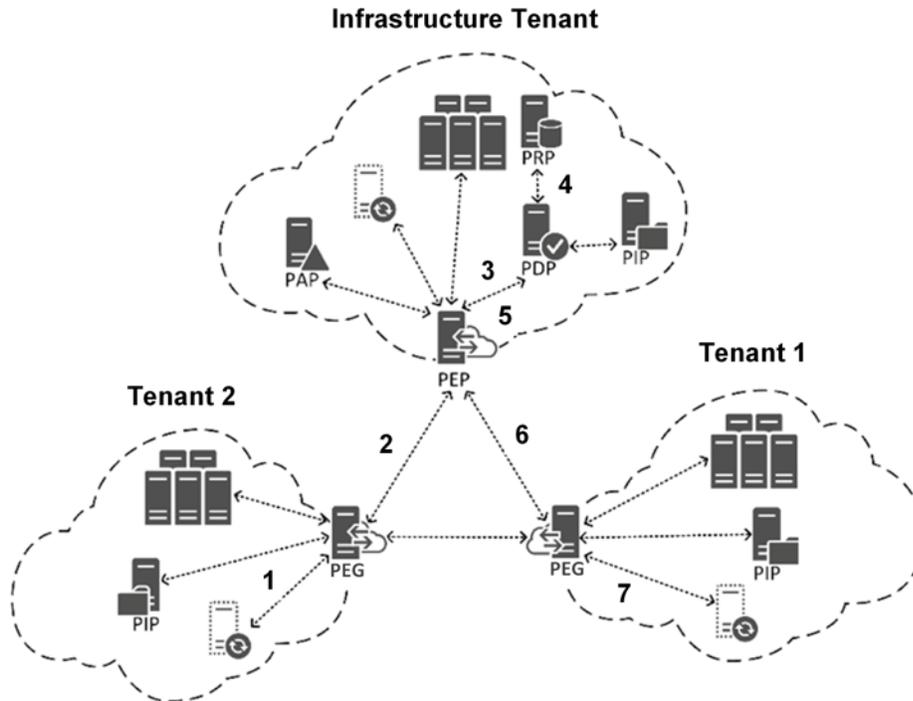
- *Proxy*: it monitors the interactions among the DS components, providing the access log upon which the monitoring service is built;
- *Chaincode*: it is the blockchain smart contract checking that no access decision has been subverted due to hijacking of intermediate DS components (aka PEGs and PIPs see *Data Security (DS)*);
- *Policy Violation Engine*: it is the component based on the formal language **FACPL** checking that the PDP (see *Data Security (DS)*) has not been compromised and hence access control policy circumvented.

The functionality of the components are discussed below.

9.1 Proxy

The DS components architecture (*Data Security (DS)*) is based on a Tomcat server solution (*Data Security (DS)*), where Policy Enforcement Points (PEP) mediate access among tenants and services. Actually, PEPs enforce decision calculated by the Policy Decision Point (PDP) which relies (via other DS components here not listed for the sake of presentation) on the Service Ledger to retrieve the policy in force in the federation for a given access request.

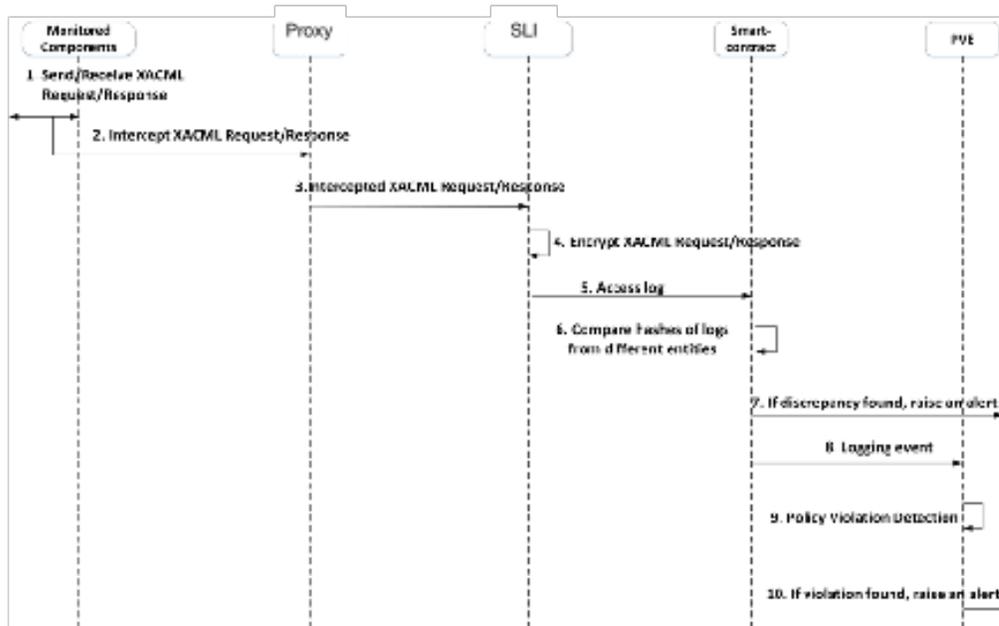
The monitoring algorithm implemented via the chaincode requires the requests transmitted across tenants—hence exchanged between distributed PEPs—must be controlled to avoid Man-in-the-Middle attacks and most of all subverted PEP gateways that maliciously alter the request contents to obtain unauthorised access. The following figure graphically reports the protocol in place among the DS components and hence what interactions the proxy must sense.



Indeed, it is worth noticing that the Attribute-based Access Control (ABAC) featured by the DS component does not ensure monotonicity of evaluation. Namely, by adding a new attribute to a request there is no guarantee on the ordering of the calculated decision, that is a *PERMIT* decision for a *n-attribute request* can become *DENY* introducing an additional one, or vice versa. Therefore, it is of paramount importance that while a request is in transit is not maliciously altered by compromised components

9.2 Chaincode

The interaction among the proxy and monitoring engine is detailed as per the following figure



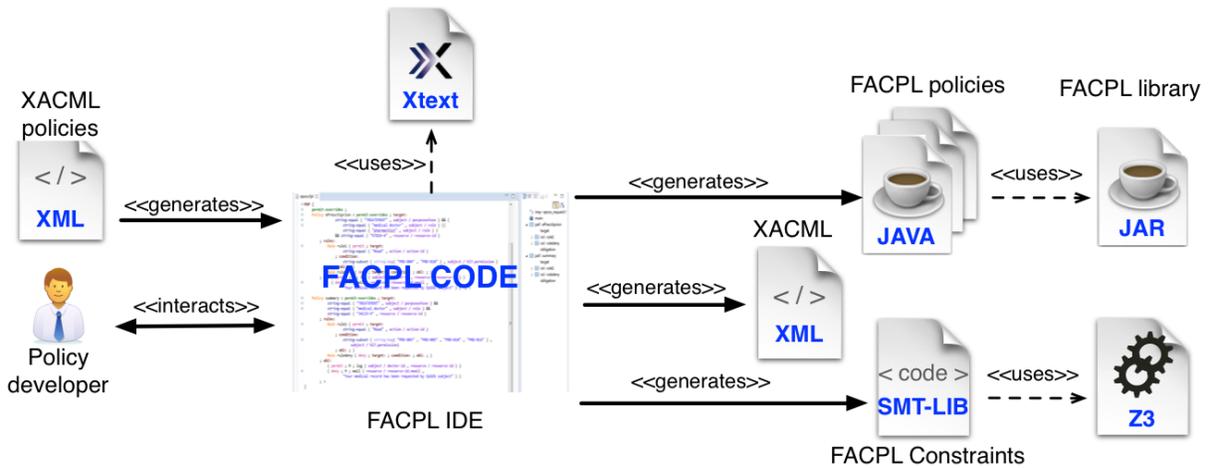
The chaincode mainly supports two main functionalities:

- *Storage of access logs*: this is realised via the Key-Value store of the Hyperledger Fabric. This gives a significant added value of relying on the history ledger of the keys: via a mechanism called CompositeKey the requests/responses generated in a session can be grouped and consequently evaluated.
- *Comparison of exchange requests*: this realises the monitoring check previously mentioned about ensuring that end-point components are not compromised as well as the communication means.

To invoke this functionality, we rely on the two-level invocation structure of the Service Ledger Interface - Service Ledger (*Service Ledger (SL)*). Therefore, via a REST invocation the Service Ledger Interface is invoked and in its own turn invokes the *Invoke* API of the Service Ledger to transparently invoke the chaincode (*Service Ledger (SL)*).

9.3 Policy Violation Engine

The Policy Violation Engine (PVE) is realised via the formal-based access control language fully interoperable with XACML named FACPL.



The toolchain provides via the Xtext framework an Eclipse-based IDE in the form of a plugin for content-assisted development of access control policies with automated translation to XACML. At the same time, it offers standalone translator library to both generate SMT-LIB code ready to be analysed via the SMT constraint solver Z3, and Java source code to potentially enforce the policy.

The analysis functionalities provided by the FACPL framework enable static verification of two main groups of properties of FACPL policies:

- *Authorisation properties* permit reasoning on the evaluation of a policy with respect to a specific request, by also considering additional attributes that can be possibly introduced in the request at run-time and that might lead to unexpected authorisations.
- *Structural properties* permit reasoning on the whole set of evaluations of policies and can be exploited, e.g., to implement maintenance and change-impact analysis techniques. Therefore, how a changing in an access control policy can affect the considered system.

Thus, the FACPL language, consequently the PVE, can be used both as supporting tool for the development of the access control policies (via the Eclipse plugin) and as implementation of the timely analysis of the monitored accesses. Examples of FACPL policies and analysis can be found [here](#).

Federated Security Audit (FSA)

The FSA exploits machine learning techniques to figure out the high-level activities occurred in the federation and, consequently, to identify vulnerabilities and security breaches.

When vulnerabilities or security breaches are detected, the FSA raises the corresponding alert to the FRM. The FSA consists of a single module that, given in input the access control logs provided by the FRM, detects vulnerabilities and security breaches.

10.1 Functionality

The FSA uses the following techniques:

- *Role Mining*: the goal here is to learn roles from actual usage information and to employ the learned roles for the purpose of vulnerabilities identification.
- *Anomaly detection*: these techniques deal with learning what is normal behaviour (profile) from historical data and comparison of profile with actual behaviour. Any significant deviation from the learned behaviour could indicate an attempt for attacks.

FSA works on a knowledge model of accesses according to the previous techniques. Specifically, it is based on three operations:

1. *CreateModel*: it is used to learn real roles and normal users' behaviour from the log file and save the results internally. This method will be invoked iteratively (i.e. every week/month) each time with a new log file.
2. *IdentifySuspiciousActivities*: it is used to compare users' behaviour demonstrated in the input log file against learned models and alert on suspicious activities. This operation allows to check quickly (and frequently, i.e. every hour) observed users' behaviour with the learned model.
3. *GetEntitlementVulnerabilites*: is used to extract entitlement information from the learned models, and will compare existing entitlement against learned models and alert on vulnerabilities.

It is worth mentioning that the proposed operations are not exposed in terms of API. Indeed, due to the size of exchanged data, we prefer to use an approach based on file exchanges.

Therefore, the functionalities of the FSA can be summarised as follows:

1. Detect vulnerabilities in existing access control mechanisms by analysing access logs.
2. Detect security breaches by analysing access logs.
3. Rise alerts to the FRM to signal vulnerabilities and/or security breaches.

Intelligent Workload Manager (IWM)

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 optimised 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)**.

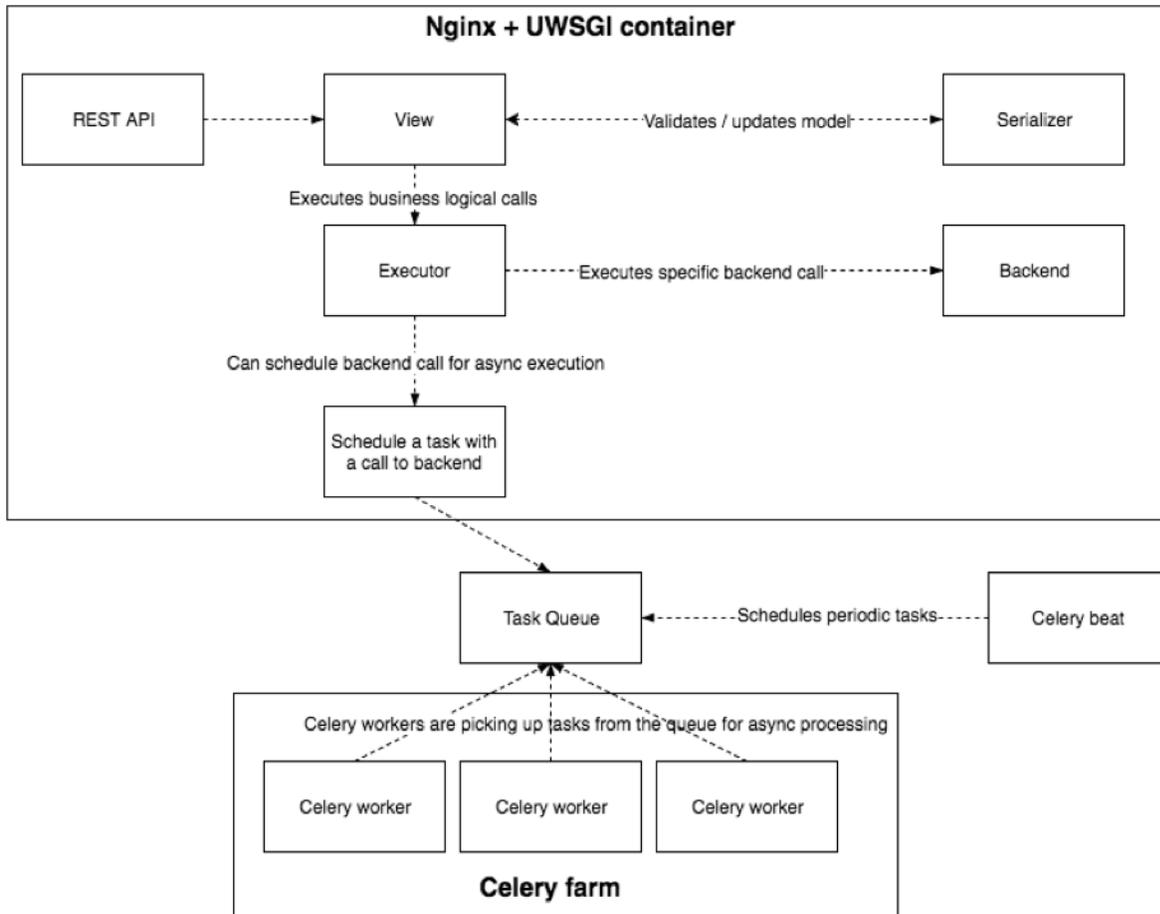
Optimisation model applicable to the scenario of Service provisioning by Service Consumer is offering an improvement over local scheduling while imposing as little as possible of additional overhead on the definition of the workload requirements. Improvement means achieving a better outcome regarding 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.

Technically, IWM is composed of the following primary components as from the figure below

- Nginx and uwsgi container hosting IWM - exposing API over HTTPS.
- Task Queue (Redis) for storing asynchronous tasks.
- Celery worker pool - processors of the stored tasks.
- Celery beat - a process for scheduling regular tasks (similar to Cron on Linux).



11.1 Azure Integration

IWM supports provisioning and management of compute resources on private and public clouds. Within SUNFISH, Azure support has been added to IWM for management of the Azure Classic VMs. The pricing and configuration of VM types has been integrated with the optimizer component to allow simplified comparison and finding optimal match also with Azure, if federation has a registered account.

Secure Multiparty Computation (SMC)

This is a page for Secure Multi-party Computation (SMC) component of the SUNFISH platform. SUNFISH platform uses [Sharemind MPC](#) as a practical implementation of SMC.

We first introduce the concept of SMC, then the Sharemind approach.

12.1 Intro to SMC

Secure multi-party computation (SMC, also abbreviated as MPC) is a technique for evaluating a function with multiple peers so that each of them learns the output value but not each other's inputs. There are various ways for implementing secure MPC with different number of peers and security guarantees. Here, we concentrate on systems based on secret sharing.

Share computing systems use the concept of secret sharing introduced by Blakley (Blakley, 1979) and Shamir (Shamir, 1979). In secret sharing, a secret value s is split into any number of shares s_1, s_2, \dots, s_n that are distributed among the peers (computing nodes). Depending on the type of scheme used, the original value can be reconstructed only by knowing all or a predefined number (threshold t) of these shares. Any group of t or more peers can combine their shares to reconstruct the original value. However, the result of combining less than t shares provides no information about the value they represent.

Secure multi-party computation protocols can be used to process secret-shared data. These protocols take secret-shared values as inputs and output a secret-shared result that can be used in further computations. For example, let us have values u and v that are both secret-shared and distributed among all the peers so that each computation node C_i gets the shares u_i and v_i . To evaluate $w = u \oplus v$ for some binary function \oplus , the computation nodes engage in a share computing protocol and output w in a shared form (node C_i holds w_i). During the computation, no computation node is able to recover the original values u or v nor learn anything about the output value w .

The fact that most SMC protocols are composable and the computation result is also secret-shared, allows one to use the output of one computation as an input for the next one. Using this property, one can combine primitive functions like multiplication or comparison into algorithms (e.g. sorting) and such algorithms into applications that implement the necessary business logic in privacy-preserving fashion.

Multi-party computation protocols can be secure in either passive or active corruption models. In the passive model, an adversary can read all the information available to the corrupted peer, but it cannot modify it. In this case, the corrupted

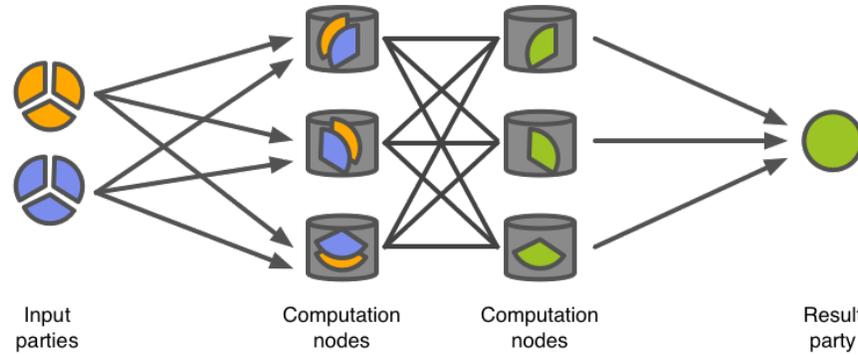


Fig. 12.1: An example of secret sharing two values (orange and blue) among three computation nodes and an SMC protocol that results in a secret-shared result (green).

peer still follows the predefined protocol, but it tries to deduce the original data values based on the information available to that peer. This is also known as *honest-but-curious* model. In the active model, an adversary has full control over the corrupted peer. For more properties of SMC protocols, see Cramer *et al.*, 2004.

12.2 Sharemind

Sharemind MPC is a practical implementation of secure multi-party computation technology with the emphasis on performance and ease of use. Sharemind MPC supports several different SMC schemes called *protection domains*, but the SUNFISH platform uses the *shared3p* protection domain, which stands for 3-out-of-3 secret sharing with passive security. This protection domain uses additive secret sharing scheme, where a secret value s is secret shared as follows:

$$\begin{aligned} s_1 &\leftarrow \text{random}(), \\ s_2 &\leftarrow \text{random}(), \\ s_3 &\leftarrow s - s_1 - s_2, \end{aligned}$$

such that $s = s_1 + s_2 + s_3$. All these computations are done modulo the corresponding data type size, e.g. modulo 2^{64} for 64-bit (unsigned) integers. Note that this modulo computation happens automatically for primitive data types like `(u)int8`, `(u)int16`, `(u)int32` and `(u)int64`. More complex data types (e.g. floating point numbers) use structures of primitive data types.

12.2.1 Architecture

Sharemind MPC deployment consists of two types of components – application servers and client applications. Sharemind Application Server implements the SMC computation node role, it talks to other Application Servers during SMC protocols and to client applications for user input and output. It also has a local persistent storage for saving input shares and computation results between computations. A typical Sharemind MPC deployment supporting the *shared3p* protection domain has three application servers and any number of client applications.

On Sharemind MPC platform, privacy-preserving applications are developed using the open source SecreC programming language. SecreC is a domain specific language that separates private and public data flows. By marking user (and other sensitive) input as private, an application developer can be confident that all computations involving these values are executed in the secure SMC environment. At the same time, the developer does not have to know the underlying SMC protocol details.

An example SecreC program, counting the number of occurrences of a secret value in a vector of secret values:

```

import shared3p;
import stdlib;

// All variables prefixed with `pd_shared3p` are secret-shared
domain pd_shared3p shared3p;

void main() {
  pd_shared3p uint64[[1]] haystack = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  // In reality, this comes from database:
  // pd_shared3p uint64[[1]] haystack = loadFromDB(...);
  pd_shared3p uint64 needle = 5;
  // In reality, this comes from user-supplied argument, for example:
  // pd_shared3p uint64 needle = argument("needle");

  // Because of private operands, the equality operation
  // invokes a corresponding SMC protocol
  pd_shared3p bool[[1]] match = (needle == haystack);

  // Publish orders each computation node to send its
  // corresponding share back to the client application
  publish("count", sum(match));
}

```

SecreC programs are deployed on Application Servers and invoked by authorised client applications by their name (think of remote procedure calls or stored procedures in database management systems). This happens in parallel in all three Application Servers.

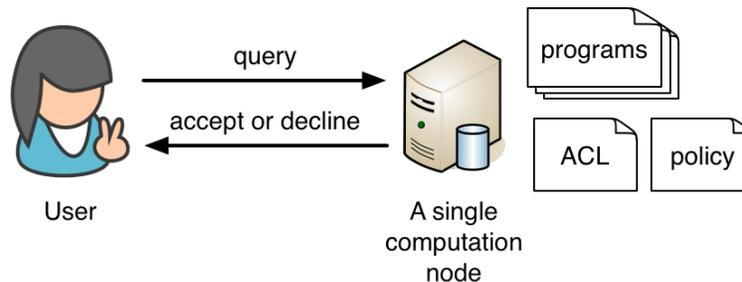


Fig. 12.2: In Sharemind MPC, each Application Server is independent in validating the user query against its access control list (ACL) and the data usage policy.

It is important to notice that each Application Server is independent in deciding a) whether the user is authorised to run a given SecreC program; and b) if the requested SecreC program correctly implements the data usage policy. An SMC protocol cannot be executed if the Application Servers do not reach consensus in these questions. Consequently, a user can only run a predetermined set of programs and a single server or a pair of servers cannot allow potentially malicious queries without the consent of the third server. This provides cryptographic enforcement of data usage policies.

12.2.2 Requirements and Privacy Guarantees

Deploying Sharemind MPC in practice requires that the three Application Servers (computation nodes) are hosted by independent parties who do not collude. Good candidates are government organisations from different jurisdictions or peers that are themselves interested in the correct outcome of the computation.

With the non-collusion requirement holding, secure multi-party computation technology and Sharemind MPC guarantee the confidentiality of private values, except the ones that are explicitly published by all three servers (either to the user or the servers themselves).

Contact information:

Riivo Talviste

riivo.talviste@cyber.ee

Service Ledger (SL)

The main rationale behind the architecture of the Service Ledger is **flexibility, modularity** and **interoperability**.

Service Ledger offers to the SUNFISH platform components straightforward interaction with complex computing infrastructure, such as **blockchain smart contract**, alleviating any technicality burdens of setting up proper communication and invocation mechanisms.

- It features a modular architecture with respect to the tenant organisation fostered by FaaS (i.e., infrastructural, operational and segregated) to ease its deployment according to the needs.
- It empowers the FaaS governance with democratic and distributed features by transparently integrating blockchain with all FaaS activities so to inject distinguishing characteristics like **democratic control of data computation** and **data integrity throughout the federation**.

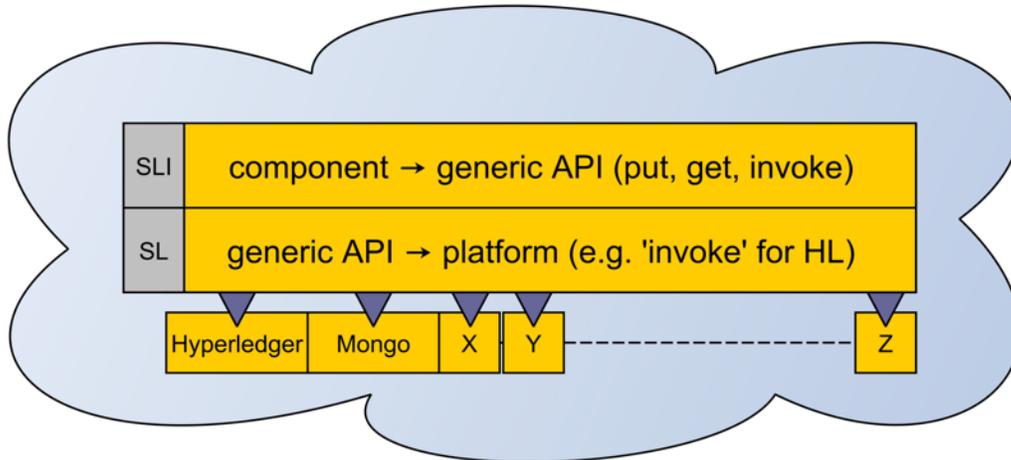
Service Ledger has also been designed to be pluggable: any new technology can be easily integrated without changing any high-level interaction with platform components. This crucially enhances the **sustainability over time of the SUNFISH Cloud federation solution** and, most of all, of one of its key contributions: underpinning federation with blockchain-empowered storage and computation.

13.1 Architecture

The infrastructure of the Service Ledger is composed by three main logical subsystems:

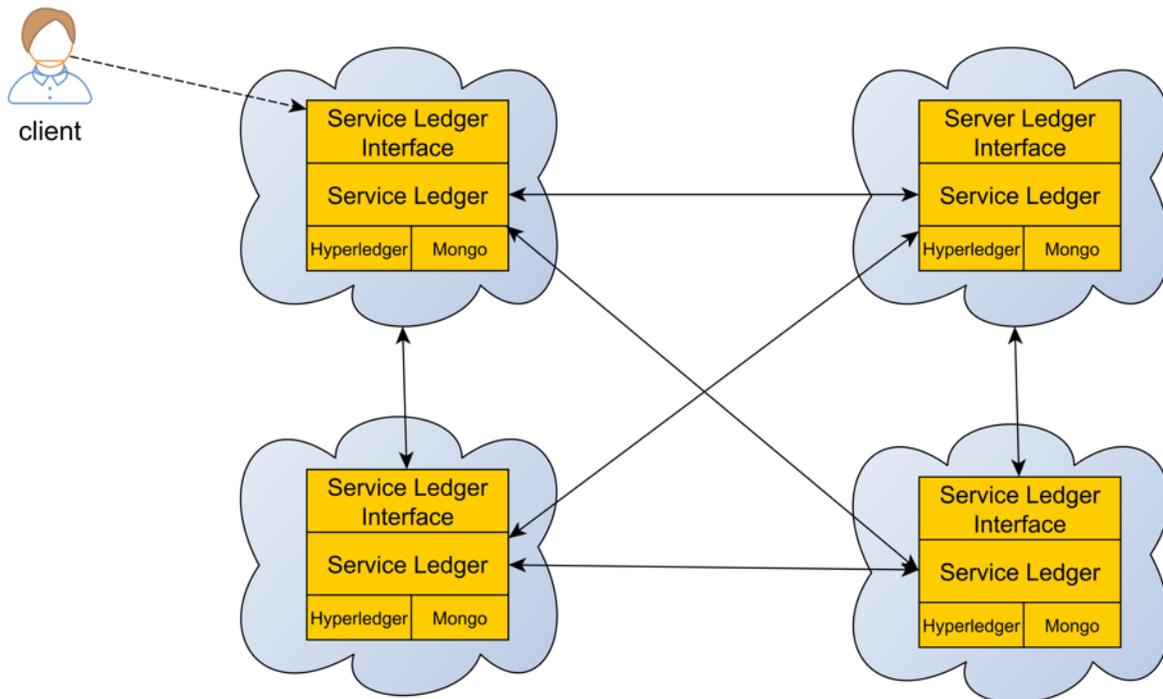
- *Computing/Storage Platform*: the low-level blockchain (or other solution) platform used to carry out decentralised computation/storage;
- *Service Ledger*: low-level API offering a common interface of the underlying platform;
- *Service Ledger Interface*: component-level API acting as point of contact for the platform components.

Logically, we have then the following setting

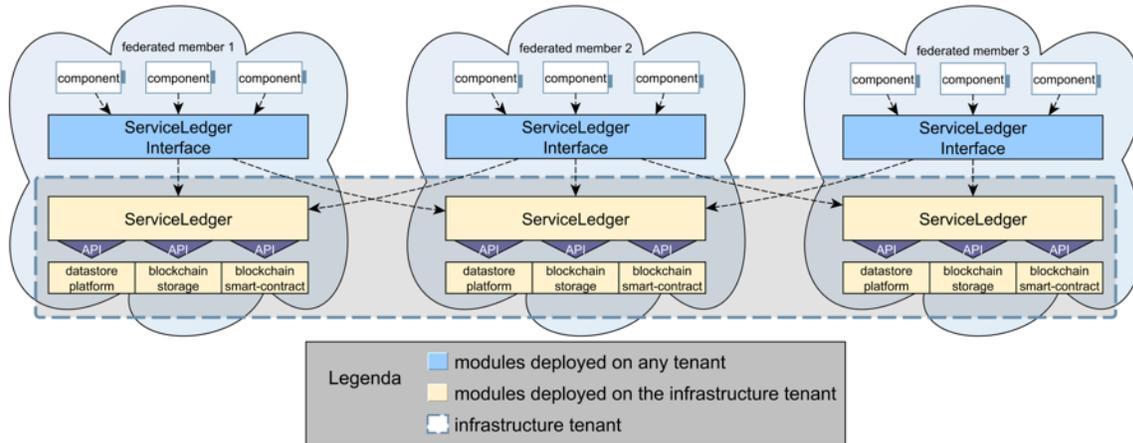


hence with a two layers of API to transparently interact with the underlying platform. For the FaaS deployment, we developed full integration with two underlying platforms: MongoDB (used for testing and integration purposes) and the smart contract blockchain platform Hyperledger Fabric.

Moving to a distributed deployment of the components, the scenario is as follows



Specifically to a FaaS federation, the high-level architecture of the Service Ledger deployment boils down to the following scenario



The *infrastructure tenant*, created across all the member clouds, features one or more SL instances (yellow blocks below) to connect with different underlying platforms; we consider three potential platforms: a data-store, a blockchain and a smart contract enabled blockchain. It comes without saying that in case of distributed infrastructure, say blockchain, such infrastructure has to be the same. The SL can be invoked by an instance of the SLI (the blue blocks below) from any tenant, that in turn can be invoked by any component.

13.1.1 Computing Platform

The Computing/Storage Platform is the low-level underlying infrastructure to execute computations and store/retrieve values. It is assumed to be distributed on each federation member composing the infrastructure tenant and it can be implemented either with just a data-store or a blockchain system.

Computations in a decentralised fashion are anyway possible only by employing a **smart contract enabled blockchain**. A data-store or a pure blockchain platform (aka à la Bitcoin) can instead just store values.

13.1.2 Service Ledger

The Service Ledger (SL) is a distributed stateless REST API server operating as logical interface of the Computing/Storage Platform. As the underlying computing/storage platform, it is deployed on each federation member of the infrastructure tenant. The Service Ledger enables the communication with the Computing/Storage Platform.

It exposes three main operations:

- *get(k)*
- *put(k,v)*
- *invoke(args)*

The operations *get(k)* and *post(k,v)* can be used with both a data-store and a blockchain as long as the memory model is based on a key-value store (KVS). Thus, NoSQL-like data-store can be employed as well as most common blockchain as they directly rely on a KVS (with versioning). The input parameters are a key *k* and an associated value *v*. **The *get(k)* operation returns the associated value *v*, while the *put(k,v)* returns confirmation of an insert of the pair key *k* and value *v*.**

In case of smart contract-based blockchain as underlying platform, the ***invoke(args)* operation can be used to invoke a computation on a smart contract and store the results on the blockchain.**

13.1.3 Service Ledger Interface

The Service Ledger Interface (SLI) is a stateless web app used as entry point to issue computations towards the SL outside the infrastructure tenant. Each component of the federation runs an instance of the SLI which as many operations as deployed services (or smart contract) running on the underlying Computation/Storage Platform. The SLI converts the received input in a couple key-value or in an 'args' format and, consequently, invokes the corresponding API of the SL.

Note: By splitting the overall Federated Service Ledger in the just presented three modules permits:

- increasing flexibility, as multiple underlying infrastructure can be easily plugged-in: they just need to implement the three API put, get and invoke;
- increasing modularity, as interaction between components and the low-level infrastructure occurs via two levels of API

Therefore, platform components, being within or outside the infrastructure tenant, can access all Service Ledger service via the exposed API considering the underlying infrastructure as a black box. Moreover, being multiple SLI/SL acting as entry points for the platform, the availability of the platform itself is strengthened.

The Service Ledger infrastructure offers computational means not just for the federation governance, but also to empower cross-Cloud services. In fact, via a high-level SLI API the SL API invoke can be used to move part of the computation on, e.g., a blockchain smart contract. This gives the benefit of decentralised, immutable and non-repudiable computation. By way of example, if two member clouds need to share a decentralised computation they can use an application-oriented smart contract deployed on the blockchain via such high-level API. This is indeed the case of UC1 where part of the computation is moved to blockchain.

This page reports the API definitions of the SUNFISH Platform components. All the OpenAPI specification in *.json/.yaml* files are available in [this GitHub repository](#).

14.1 Anonymisation (ANM)

14.1.1

PUT /micro/configuration/{id}

- **Description:** Update a 'MicroConfiguration' object.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path	MicroConfiguration id	string
body	body	MicroConfiguration	

Responses

200 - Success. The MicroConfiguration was updated

404 - The MicroConfiguration with the specified id was not found

400 - Invalid request (invalid MicroConfiguration id)

DELETE /micro/configuration/{id}

- **Description:** Delete a 'MicroConfiguration' object.
-

Parameters

Name	Position	Description	Type
id	path	MicroConfiguration id	string

Responses

200 - Success. The MicroConfiguration was deleted

404 - The MicroConfiguration with the specified id was not found

400 - Invalid request (invalid MicroConfiguration id)

GET /micro/configuration/{id}

- **Produces:** [u'application/json']
- **Description:** Get a 'MicroConfiguration' object.

Parameters

Name	Position	Description	Type
id	path	MicroConfiguration id	string

Responses

200 - Success. The body contains the requested MicroConfiguration

404 - The MicroConfiguration with the specified id was not found

400 - Invalid request (invalid MicroConfiguration id)

POST /macro/configuration

- **Produces:** [u'application/json']
- **Description:** Creates a new 'MacroConfiguration' object.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	MacroConfiguration	

Responses

200 - Success. The body contains the requested MacroConfiguration id

400 - Invalid request (invalid MacroConfiguration).

POST /micro

- **Produces:** [u'application/json']
 - **Description:** Anonymize data
-

Parameters

Name	Position	Description	Type
configuration	query	MicroConfiguration id	string
file	query	file id	string

Responses

200 - Success. The body contains output file id.

404 - The MicroConfiguration id or file id were not found

PUT /macro/configuration/{id}

- **Description:** Update a 'MacroConfiguration' object.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path	MacroConfiguration id	string
body	body	MacroConfiguration	

Responses

200 - Success. The MacroConfiguration was updated

404 - The MacroConfiguration with the specified id was not found

400 - Invalid request (invalid MacroConfiguration id)

DELETE /macro/configuration/{id}

- **Description:** Delete a 'MacroConfiguration' object.

Parameters

Name	Position	Description	Type
id	path	MacroConfiguration id	string

Responses

200 - Success. The MacroConfiguration was deleted

404 - The MacroConfiguration with the specified id was not found

400 - Invalid request (invalid MacroConfiguration id)

GET /macro/configuration/{id}

- **Produces:** [u'application/json']
- **Description:** Get a 'MacroConfiguration' object.

Parameters

Name	Position	Description	Type
id	path	MacroConfiguration id	string

Responses

200 - Success. The body contains the requested MacroConfiguration

404 - The MacroConfiguration with the specified id was not found

400 - Invalid request (invalid MacroConfiguration id)

DELETE /file/{id}

- **Description:** Delete a file.

Parameters

Name	Position	Description	Type
id	path	file id	string

Responses

200 - Success. The file was deleted

404 - The file with the specified id was not found

400 - Invalid request (invalid file id)

GET /file/{id}

- **Produces:** [u'multipart/form-data']
- **Description:** Download a file.

Parameters

Name	Position	Description	Type
id	path	File id	string

Responses

200 - Success. The body contains the requested file

404 - The file with the specified id was not found

500 - Internal server error.

POST /file

- **Produces:** [u'application/json']
 - **Description:** Uploads a new file.
 - **Consumes:** [u'multipart/form-data']
-

Parameters

Name	Position	Description	Type
file	formData	the file to upload	file

Responses

200 - Success. The body contains the uploaded file id

500 - Internal server error

400 - Invalid request

POST /micro/configuration

- **Produces:** [u'application/json']
- **Description:** Creates a new 'MicroConfiguration' object.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	MicroConfiguration	

Responses

200 - Success. The body contains the requested MicroConfiguration id

400 - Invalid request (invalid MicroConfiguration).

POST /macro

- **Produces:** [u'application/json']
- **Description:** Execute queries with differential privacy

Parameters

Name	Position	Description	Type
configuration	query	Macroonfiguration id	string
file	query	file id	string

Responses

200 - Success. The body contains outpu file id.

404 - The MacroConfiguration id or file id were not found

14.2 Anonymisation Interface (ANI)

ANI provides anonymisation service for federated clouds via differential privacy guarantee.

Version: 1.0.0

Contact information:

Andrea Margheri

a.margheri@soton.ac.uk

14.2.1

POST /interface/register

- **Description:** This endpoint is used to register a new data-sharing event.

Parameters

Name	Position	Description	Type
body	body	JSON body of the register information	

Responses

200 - *The response body for successful response*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.)*

400 - *Invalid request, required parameters missing.*

POST /interface/queryFromUser

- **Description:** This endpoint is used to query the anonymised statistical the DataId and requested budget

Parameters

Name	Position	Description	Type
body	body	JSON body of the query input	

Responses

200 - *The response body for a successful response*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.)*

400 - *Invalid request, required parameters missing*

14.3 Data Masking (DM)

14.3.1

DELETE /contexts/{id}

- **Description:** Delete a ‘Context’ object.

Parameters

Name	Position	Description	Type
id	path	Context id	string

Responses

200 - Success. The context was deleted

404 - The context with the specified id was not found

400 - Invalid request (invalid context id)

POST /policies

- **Description:** Creates a new ‘Policy’ object.
- **Consumes:** [u’application/json’]

Parameters

Name	Position	Description	Type
body	body	Policy	

Responses

200 - Success. The body contains the requested policy id

400 - Invalid request (invalid policy).

POST /process

- **Description:** process a payload
- **Consumes:** [u’text/plain’, u’text/xml’, u’application/json’, u’application/xml’]

Parameters

Name	Position	Description	Type
policy	query	policy id	string
context	query	context id	string
X-DM-encryption-key	header	encryption key to use	string
X-DM-encryption-iv	header	encryption initial vector to use	string
body	body	payload	

Responses

200 - Success. The body contains the processed payload

404 - The policy id or context id were not found

400 - Invalid request (invalid policy id, context id, missing headers).

POST /contexts

- **Description:** Creates a new 'Context' object.

Responses

200 - Success. The body contains the requested policy id

PUT /policies/{id}

- **Description:** Update a 'Policy' object.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path	Policy id	string
body	body	Policy	

Responses

200 - Success. The policy was updated

404 - The policy with the specified id was not found

400 - Invalid request (invalid policy id)

DELETE /policies/{id}

- **Description:** Delete a 'Policy' object.

Parameters

Name	Position	Description	Type
id	path	Policy id	string

Responses

200 - Success. The policy was deleted

404 - The policy with the specified id was not found

400 - Invalid request (invalid policy id)

GET /policies/{id}

- **Produces:** [u'application/json']
- **Description:** Get a 'Policy' object.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
id	path	Policy id	string

Responses

200 - Success. The body contains the requested policy

404 - The policy with the specified id was not found

400 - Invalid request (invalid policy id)

14.4 Policy Administration Point (PAP)

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

14.4.1 /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://%3C host/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 authenticati on of source entity and its authenticati on 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	

14.4.2 /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 authenticati on of source entity and its authenticati on 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	

14.5 Policy Decision Point (PDP)

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

14.5.1 /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.

14.5.2 /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	

14.5.3 /v1/verifyServicePolicySet

POST

Summary: Verify a service policy set

Description:

Parameters

Name	Located in	Description	Re-quired	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	

14.5.4 /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	Lo-cated in	Description	Re-quired	Schem a
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	

14.5.5 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

14.6 Policy Enforcement Point (PEP)

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

14.6.1 /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]

14.6.2 /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	Lo- cated in	Description	Re- quired	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-read able 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-read able 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 correspondin 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 , call-back URI. This field includes other request meta-data that may extend or override the definitions provided in centralized administrative console. These include request type, application- specific policies or obligations to be applied beyond the ones defined in the central console, or parameters related to data-masking policies. The scope of applicable and allowed definitions provided in this variable depends on an extent of delegation policies, as determined in centralized console.	No	string
SUNFISH- request- data	Header	This field encapsulates the original header data and the original query string as issued by the application.	Yes	string
SUNFISH- signature	Header	This parameter is used to ensure integrity and authenticity of the source message for applications which require a higher degree of security. It contains signed request and fields, according to predefined schema	No	string

Responses

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

14.7 Policy Information Point (PIP)

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

14.7.1 /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 authenticati on of source entity and its authenticati on 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	

14.7.2 /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-issuer	header	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.	

14.8 Policy Retrieval Point (PRP)

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

14.8.1 /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	

14.8.2 /v1/policyset/{id}/{version}

GET

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

Description:

Parameters

Name	Lo- cated in	Description	Required	Schema
root-Poli- cySet	query	true	false Defines if root policy-se t 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
ver- sion	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	

14.8.3 /v1/policy/{id}

GET

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

Description:

Parameters

Name	Lo- cated in	Description	Required	Schema
root- Policy	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 re- turned 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	

14.8.4 /v1/policyset/{id}

GET

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

Description:

Parameters

Name	Lo- cated in	Description	Required	Schema
root- Policy- Set	query	true	false Defines if root policy-se t 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	

14.8.5 /v1/policy/{id}/{version}

GET

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

Description:

Parameters

Name	Lo- cated in	Description	Required	Schema
root- Pol- icy	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
ver- sion	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	

14.9 Federated Administration Monitoring (FAM)

Here reported the API offered by the FAM for alert management. Notice that the other functionalities are offered via a UI, not via API

14.9.1

POST /triggerAlert

- **Produces:** [u'application/json']
- **Description:** This endpoint is used to receive security alerts.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - The response body for a successful response

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.).

400 - Invalid request, required parameter(s) missing.

14.10 Federated Runtime Monitoring (FRM)

14.10.1 Proxy

POST /agent

- **Produces:** [u'application/json']
- **Description:** This endpoint is used to send an alert from FSA to a SMART Agent.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

14.10.2 Policy Validation Engine

POST /pve

- **Produces:** [u'application/json']
- **Description:** This endpoint is used to indicate new logs have been stored in the blockchain.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

14.11 Intelligent Workload Manager (IWM)

IWM provides lifecycle management for virtual resources in a multi-cloud multi-tenant environment. It also provides optimised 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

14.11.1

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 stuff user or customer owner.

- **Description:** To remove a link, issue **DELETE** to URL of the corresponding connection as stuff 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

Responses

200 -

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

Name	Position	Description	Type
uuid	path		string

Responses

200 -

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		

Responses

201 -

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

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

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

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

.. code-block:: http

```
POST /api/hooks-web/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6bac41c706593e66f567a7fa4 Host: example.com
```

```
{ "event_types": ["resource_start_succeeded"], "event_groups": ["users"], "destination_url": "http://example.com/" }
```

When hook is activated, **POST** request is issued against destination URL with the following data:

.. code-block:: javascript

```
{ "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": "4633bbb0b3a4b91bffc0e18f853de85", "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:

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/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 -

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		

Responses

200 -

DELETE /api/openstack/{uuid}/

Parameters

Name	Position	Description	Type
uuid	path		string

Responses

204 -

PATCH /api/openstack/{uuid}/

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

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

Continued on next page

Table 14.1 – continued from previous page

Name	Position	Description	Type
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
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

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

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/hooks-push/

To create new push hook issue **POST** against `*/api/hooks-push/*` as an authenticated user.

- **Description:** To create new push hook issue **POST** against `*/api/hooks-push/*` as an authenticated user. You should specify list of `event_types` or `event_groups`.

Example of a request:

.. code-block:: http

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

```
{ "event_types": ["resource_start_succeeded"], "event_groups": ["users"], "type": "Android" }
```

You may temporarily disable hook without deleting it by issuing following **PATCH** request against hook URL:

.. code-block:: javascript

```
{ "is_active": "false" }
```

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

Parameters

Name	Position	Description	Type
data	body		

Responses

201 -

GET /api/hooks-push/**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/xgej9pYRXGha4l+LkKfDgwAoXdV1z79EG1+9ns7wXuqMJFHM2KDpxAizV0GkZcojISvDw
vEAFdOJcqjyH4FOGYa8usP1 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']

GET /api/service-settings/

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

- **Description:** To get a list of service settings, run **GET** against `*/api/service-settings/*` as an authenticated user. Only settings owned by this user or shared settings will be listed.

Supported filters are:

- `?name=<text>` - partial matching used for searching
- `?type=<type>` - choices: OpenStack, DigitalOcean, Amazon, JIRA, GitLab, Oracle
- `?state=<state>` - choices: New, Creation Scheduled, Creating, Sync Scheduled, Syncing, In Sync, Erred
- `?shared=<bool>` - allows to filter shared service settings

Parameters

Name	Position	Description	Type
page	query		string
page_size	query		string
name	query		string
type	query		string
state	query		string
shared	query		string
name	query		string
type	query		string
state	query		string
shared	query		string

Responses

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

200 -

GET /api/openstack-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

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

- ?period=YYYY-MM - return a list with SLAs for a given month - ?period=YYYY - return a list with SLAs for a given year

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 <<http://nodeconductor-organization.readthedocs.org/en/stable/>>' instead) - full_name - native_name - job_title - phone_number - email

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):

.. code-block:: http

```
PUT /api/users/e0c058d06864441fb4f1c40dee5dd4fd/ HTTP/1.1 Content-Type: application/json Accept: application/json Authorization: Token c84d653b9ec92c6cbac41c706593e66f567a7fa4 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 staff 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

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: exam-
ple.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 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/e5f973af2eb14d2d8c38d62bcbacb33/" }
```

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		

Responses

200 -

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

Name	Position	Description	Type
uuid	path		string

Responses

200 -

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		

Responses201 -

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

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses200 -

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

Name	Position	Description	Type
uuid	path		string

Responses204 -

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		

Responses201 -

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

- Consumes: [u'application/json']

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses201 -

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

Responses200 -

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

Responses

204 -

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

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

Parameters

Name	Position	Description	Type
uuid	path		string
data	body		

Responses

200 -

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

Name	Position	Description	Type
uuid	path		string

Responses

200 -

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

Name	Position	Description	Type
uuid	path		string

Responses

200 -

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 -

14.12 Secure Multi-party Computation (SMC)

SUNFISH framework's SMC service consists of two components, SMC Proxy and computation node, both providing their own API. In each case, both the request and response body are JSON-encoded

Version: 0.1

Contact information:

Riivo Talviste
riivo.talviste@cyber.ee

14.12.1 SMC Node Service

SMC Node Service provides the interface to run privacy-preserving programs (i.e., SecreC programs) on SMC nodes. Each SMC node provides this service independently, the client must invoke this service in parallel at each SMC node.

POST /startProcess

- **Description:** Start the SecreC secure computation process on computation nodes.

Parameters

Name	Position	Description	Type
data	body		

Responses

200 - Successful response

Name	Description	Type
process_id		integer
vars	Values published by the SecreC program.	object
server	Server number	integer

POST /requestProcess

- **Description:** Initiate a new process request at computation nodes.

Parameters

Name	Position	Description	Type
data	body		

Responses

200 - Successful response

Name	Description	Type
process_id		integer
relay_data	Contains signed relay data from the computation node. This data is relayed to the other computation nodes by the next request.	
server	Server number	integer

POST /relayedProcessShare

- **Description:** Relay signed data obtained by ‘requestProcess’ to other computation nodes.

Parameters

Name	Position	Description	Type
data	body		

Responses

200 - *Successful response*

14.12.2 SMC Proxy Service

SMC Proxy Service is a client-side helper service for secret-sharing user input for the secure multi-party computation and reconstructing the results for further processing in the client application.

POST /secretShare

- **Description:** Secret share input data

Parameters

Name	Position	Description	Type
data	body		

Responses

200 - *Successful response*

POST /reconstruct

- **Description:** Reconstruct data from shares

Parameters

Name	Position	Description	Type
data	body		

Responses

200 - *Successful response*

Version: 1.0.0

Contact information:

Andrea Margheri
a.margheri@soton.ac.uk

14.13 Service Ledger (SL)

14.13.1 store

POST /get

- **Description:** Retrieving a value by its key

Parameters

Name	Position	Description	Type
getId	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

404 - *The requested key is not found.*

401 - *The operation is not allowed.*

400 - *Invalid request, required parameter(s) missing.*

POST /getKeys

- **Description:** Get all the key of a category
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /put

- **Description:** Storing a key-value pair

Parameters

Name	Position	Description	Type
putSpec	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed.*

400 - *Invalid request, required parameter(s) missing.*

POST /delete

- **Description:** Delete a stored key

Parameters

Name	Position	Description	Type
body	body		

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed.*

400 - *Invalid request, required parameter(s) missing.*

14.13.2 exec**POST /invoke**

- **Description:** invoke the functions in the chaincode

Parameters

Name	Position	Description	Type
invokeSpec	body		

Responses

200 - *The response body for a successful chaincode invoke*

401 - *The operation is not allowed*

400 - *Invalid request, required parameter(s) missing.*

14.14 Service Ledger Interface (SLI)**14.14.1 dm**

POST /dm/store

- **Description:** Storing encryption/tokenization key
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /dm/read

- **Description:** Retrieving a stored key

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing*

POST /dm/delete

- **Description:** Delete a stored key

Parameters

Name	Position	Description	Type
body	body		

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing*

14.14.2 monitoring

POST /monitoring/store

- **Produces:** [u'application/json']
- **Description:** This endpoint is used to store relevant monitoring data.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /monitoring/read

- **Produces:** [u'application/json']
- **Description:** This endpoint is used to read the relevant monitoring data.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

14.14.3 alert

POST /alert/store

- **Produces:** [u'application/json']
- **Description:** This endpoint is used to store alerts.
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /alert/read

- **Produces:** [u'application/json']
- **Description:** This endpoint is used to retrieve the stored alert using the index. The body contains the alert id received by the store api
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

14.14.4 state

POST /state/vm-store

- **Description:** Storing federated services
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /state/member-store

- **Description:** Storing federated services
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /state/tenant-create

- **Description:** Creating a tenant
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /state/member-read

- **Description:** Storing federated services
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON cotaining the service id	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /state/tenant-addMember

- **Description:** Creating a tenant
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /state/service-read

- **Description:** Storing federated services
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON cotaining the service id	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /state/getKeys

- **Description:** Get all the key of a category
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /state/tenant-read

- **Description:** Storing federated services
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON cotaining the service id	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /state/vm-read

- **Description:** Storing federated services
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON cotaining the vm id	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /state/service-store

- **Description:** Storing federated services
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /state/deleteKey

- **Description:** Remove the pair identified by the key
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - The response body for a successful response.

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.).

400 - Invalid request, required parameter(s) missing.

14.14.5 policy

POST /policy/delete

- **Description:** Deleting a policy by its id

Parameters

Name	Position	Description	Type
policyId	body	Body in JSON	

Responses

200 - The response body for a successful response.

404 - The respective policy is not found.

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.).

400 - Invalid request, required parameter(s) missing

POST /policy/store

- **Description:** Storing a new policy

Parameters

Name	Position	Description	Type
policySpec	body	Body in JSON	

Responses

200 - The response body for a successful response.

409 - The operation is not allowed as the policy already exists.

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.).

400 - Invalid request, required parameter(s) missing.

POST /policy/polService

- **Description:** Retrieving policies associated to a service

Parameters

Name	Position	Description	Type
serviceId	body	Body in JSON	

Responses

200 - The response body for a successful response.

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.).

400 - Invalid request, required parameter(s) missing

POST /policy/read

- **Description:** Retrieving a policy by its id

Parameters

Name	Position	Description	Type
policyId	body	Body in JSON	

Responses

200 - The response body for a successful response.

404 - The requested policy is not found.

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.).

400 - Invalid request, required parameter(s) missing

14.14.6 proposal

POST /proposal/getProposal

- **Description:** getting a proposal
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /proposal/voteProposal

- **Description:** vote for a submitted a proposal
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /proposal/submitProposal

- **Description:** Submitting a proposal
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - *The response body for a successful response.*

401 - *The operation is not allowed (unauthorised access, the token is invalid, etc.).*

400 - *Invalid request, required parameter(s) missing.*

POST /proposal/countVotes

- **Description:** Request a votes counting to validate a stored proposal
- **Consumes:** [u'application/json']

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - The response body for a successful response.

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.).

400 - Invalid request, required parameter(s) missing.

14.14.7 anonymisation

POST /anonymisation/register

- **Description:** This endpoint is used to register a data-sharing event.

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - The response body for a successful response.

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.).

400 - Invalid request, required parameter(s) missing.

POST /anonymisation/queryOldRes

- **Description:** This endpoint is used to query the anonymised statistical result in Registry given the DataId and requested budget.

Parameters

Name	Position	Description	Type
body	body	Body in JSON	

Responses

200 - The response body for a successful response.

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.)

400 - Invalid request, required parameter(s) missing

POST /anonymisation/updateLedger

- **Description:** this endpoint is used to update final result to Registry

Parameters

Name	Position	Description	Type
body	body	JSON body of the received result	

Responses

200 - Success

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.)

400 - Invalid request, required parameter(s) missing

POST /anonymisation/receiveAnonyRes

- **Description:** this endpoint is used to receive anonymised result from the anonymisation interface

Parameters

Name	Position	Description	Type
body	body	JSON body of the received result	

Responses

200 - Success

401 - The operation is not allowed (unauthorised access, the token is invalid, etc.)

400 - Invalid request, required parameter(s) missing

Networking Infrastructure

This is a page for helping the setup the network and access services needed for connecting different clouds.

15.1 Network Component

This text details the configuration of the SUNFISH Network service, which is the component realizing the Cross-Cloud Networking functionalities (see SUNFISH deliverable ID5). The *aggregation* of networks realizes Network OSI layer visibility among different *sections* (i.e., networks), which realizes a single SUNFISH tenant. This is achieved using site-to-site VPN channels, provided by multiple instances of the Network component VMs.

15.1.1 Planning phase

Before setting-up the network services, it is necessary to:

P1. - Plan the networks for the Sections, and the Sections for the SUNFISH Tenants:

1. plan the address space of the set of SUNFISH Sections, which has to form a SUNFISH Federation. This task depends on the maximum number of services to be deployed on each of the Section. As an example, for a /24 Network the maximum number of services which can be deployed (total usable hosts) in the Section is 253.
2. define what Sections are aggregated to form the SUNFISH Tenants.

P2. - **Plan the VPN channels topology among clouds.** Given a certain set of SUNFISH Tenants composed by Sections, there are in general multiple VPN topologies which can be used for their implementation. The criterias to be utilized when choosing a VPN topology are outside the scope of this document. The idea however is to try to minimize the number of VPN connections while avoiding throughput bottlenecks and single points of failture. In the simplest case a hub-and-spoke topology is considered. In this case, for aggregating a set of Sections, there are the following conditions:

1. (deployment) one instance of the Network service is deployed in each cloud, serving alle the sections belonging to the cloud

2. (configuration) a single instance of the Network service is configured as a VPN Server, while the remaining are configured as VPN Clients connected to the Server
 3. (routing) the VPN Clients and Server are configured to realize the Tenants.
- P3. - Decide other parameters related to networking aspects:
1. IP and ports the Network services, as seen from outside the federation. The IP and ports are those which are used by a VPN servers and clients externally to a cloud.
 2. IP and ports the Network services, as seen from inside the federation. A convention for the Network IP address defines a certain IP address in each Section to be reserved as gateway for inter-section communication (i.e., a default gateway for the federated services). As an example, the IP `..*.254` can be reserved as a convention in each section for the inter-tenant communication.
 3. what are the security parameters for the VPN connections which will be used. The security parameters determines the kind of security measurement adopted for the VPN channels.

15.1.2 Configuration phase

Based on the plan phases, there are the following task to the beformed:

T1. - Configuration at cloud level:

1. for each cloud, it has to be created a cloud Tenant (to not be confused with a SUNFISH tenant) reserved to SUNFISH. Then the virtual network(s) associated to sections (point P1) has to be created
2. for each cloud, the VMs dedicated to the instances of Network service (in the simplest case, just one) has to be instantiated. Given that a Network service serves a number n of sections, its VM has to have $n+1$ NICs: {LAN_1, LAN_2, ... LAN_n, WAN}, where the WAN interface is the one directly exposed to a static cloud's public IP
3. a proper IP address has to be assigned to the various LAN NICS, accordingly to the network convention of point P3
4. for each cloud, there is the need to configure a set of firewall rules at cloud level, allowing traffic from/to the cloud's tenant (specifically, from/to the IP forwarded to the WAN interface). This depends on the planned topology of point P2. The firewall rules has to consider the connection direction between VPN Clients and Servers, given the fact that each Client initiate a connection toward a Server. In particular the firewall rules must allows inbound traffic for a VPN server, and outbound traffic for a VPN client.
5. depending on the specific cloud implementation, there could be additional traffic rules to be enabled. As an example, in Microsoft Azure there is the need to enable the "forward between NICs", and in Openstack there is the need to enable "traffic from/to unknown networks".

T2. - Configuration of each of the instances of the Network service

1. configure the OpenVPN service of each of the Network service instances. This depends on the planning of point P2 (i.e. Server or Client role), and on the planning of point P3.
2. configure the firewall and routing tables for each of the Network service instance. Similarly to what done in point T1.4, the traffic must be allowed as inbound or outbound on WAN interface depending on the server/client role and the topology, but in this case at the level of the WAN network interface of the VM hosting the Network service. Moreover, there has to be a set of routing rules instructing each instance of the Network service to properly route traffic through the WAN interface (when having sections of the same SUNFISH Tenant in two different clouds), or between different LAN interfaces (when having sections of the same SUNFISH Tenant in the same clouds). As a result, for each SUNFISH Tenant, all the sections composing

the Tenant must have full Network OSI visibility, while different SUNFISH sections has to be isolated. An exception is given by the Infrastructure Tenant, which have to be visible to all the remaining SUNFISH sections (see SUNFISH Deliverable D3.11).

15.1.3 Configuration of a VPN: details

This sub-section further elaborates the point T2 through an example. In our prototype we used PFSense 2.4.1 for the instances of the Network service. The configuration of the services can be done by web interface, or by providing an XML file. In the following, we provide the relevant parts of such XML file for the configuration of an instance of a Network service to be configured as a VPN Client. We consider an example in which network N1 located in Cloud “A” has to be aggregated to networks N2, N3 located in Cloud “B”. In particular we have:

1. Cloud “A” have network N1 defined as 192.168.1.0/24, with an instance of PF-Sense “NS1” to be configured as server
2. Cloud “A” have networks N2,N3 defined as 192.168.8.0/24 and 192.168.9.0/24, with an instance of PF-Sense “NS2” to be configured as a client

Then for “NS2” there are the following relevant parts to be customized in its configuration:

Network interfaces:

There are defined the network interfaces to be used by PF-Sense.

```
<interfaces>
  <wan>
    <enable></enable>
    <if>hn0</if>
    <ipaddr>dhcp</ipaddr>
    <gateway></gateway>
    <blockbogons>on</blockbogons>
    <media></media>
    <mediaopt></mediaopt>
    <dhcp6-duid></dhcp6-duid>
    <dhcp6-ia-pd-len>0</dhcp6-ia-pd-len>
  </wan>
  <lan1>
    <descr><![CDATA[LAN1]]></descr>
    <if>hn1</if>
    <enable></enable>
    <spoofmac></spoofmac>
    <mtu>1446</mtu>
    <ipaddr>192.168.8.254</ipaddr>
    <subnet>24</subnet>
  </lan1>
  <lan2>
    <descr><![CDATA[LAN2]]></descr>
    <if>hn2</if>
    <enable></enable>
    <spoofmac></spoofmac>
    <mtu>1446</mtu>
    <ipaddr>192.168.9.254</ipaddr>
    <subnet>24</subnet>
  </lan2>
</interfaces>
```

Firewall rules:

It is allowed an administrator IP 1.5.5.5 to configure the PF-Sense trough web interface and ssh:

```
<filter>
  <rule>
    <id></id>
    <tracker>1511201327</tracker>
    <type>pass</type>
    <interface>wan</interface>
    <ipprotocol>inet</ipprotocol>
    <tag></tag>
    <tagged></tagged>
    <max></max>
    <max-src-nodes></max-src-nodes>
    <max-src-conn></max-src-conn>
    <max-src-states></max-src-states>
    <statetimeout></statetimeout>
    <statetype><![CDATA[keep state]]></statetype>
    <os></os>
    <protocol>tcp</protocol>
    <source>
      <address>1.5.5.5</address>
    </source>
    <destination>
      <network>wan</network>
      <port>22</port>
    </destination>
    <descr><![CDATA[To Configure PF-Sense. ONLY FOR TESTING. To be CLOSED
↳in real world scenario.]]></descr>
  </rule>
  <rule>
    <id></id>
    <tracker>1511201153</tracker>
    <type>pass</type>
    <interface>wan</interface>
    <ipprotocol>inet</ipprotocol>
    <tag></tag>
    <tagged></tagged>
    <max></max>
    <max-src-nodes></max-src-nodes>
    <max-src-conn></max-src-conn>
    <max-src-states></max-src-states>
    <statetimeout></statetimeout>
    <statetype><![CDATA[keep state]]></statetype>
    <os></os>
    <protocol>tcp</protocol>
    <source>
      <address>1.5.5.5</address>
    </source>
    <destination>
      <network>wan</network>
      <port>443</port>
    </destination>
    <descr><![CDATA[To Configure PF-Sense. ONLY FOR TESTING. To be CLOSED
↳in real world scenario.]]></descr>
  </rule>
```

It is allowed the PF-Sense client to connect to the PF-Sense Server at 1.2.3.4:8443 :

```

<rule>
  <id></id>
  <tracker>1510931486</tracker>
  <type>pass</type>
  <interface>wan</interface>
  <ipprotocol>inet</ipprotocol>
  <tag></tag>
  <tagged></tagged>
  <max></max>
  <max-src-nodes></max-src-nodes>
  <max-src-conn></max-src-conn>
  <max-src-states></max-src-states>
  <statetimeout></statetimeout>
  <statetype><![CDATA[keep state]]></statetype>
  <os></os>
  <protocol>tcp</protocol>
  <source>
    <network>wan</network>
  </source>
  <destination>
    <address>1.2.3.4</address>
    <port>8443</port>
  </destination>
  <descr><![CDATA[Allow to initiate VPN connection to the VPN Server Cloud]]></
↳descr>
</rule>

```

It is allowed traffic from/to the Sections:

```

<rule>
  <id></id>
  <tracker>1510931286</tracker>
  <type>pass</type>
  <interface>openvpn</interface>
  <ipprotocol>inet</ipprotocol>
  <tag></tag>
  <tagged></tagged>
  <max></max>
  <max-src-nodes></max-src-nodes>
  <max-src-conn></max-src-conn>
  <max-src-states></max-src-states>
  <statetimeout></statetimeout>
  <statetype><![CDATA[keep state]]></statetype>
  <os></os>
  <source>
    <any></any>
  </source>
  <destination>
    <any></any>
  </destination>
  <descr><![CDATA[Allow traffic trough virtual interface openVPN]]></
↳descr>
</rule>
<rule>
  <id></id>
  <tracker>1510935658</tracker>
  <type>pass</type>
  <interface>lan1</interface>

```

```

        <ipprotocol>inet</ipprotocol>
        <tag></tag>
        <tagged></tagged>
        <max></max>
        <max-src-nodes></max-src-nodes>
        <max-src-conn></max-src-conn>
        <max-src-states></max-src-states>
        <statetimeout></statetimeout>
        <statetype><![CDATA[keep state]]></statetype>
        <os></os>
        <source>
            <any></any>
        </source>
        <destination>
            <any></any>
        </destination>
        <descr><![CDATA[Allow traffic of lan1]]></descr>
    </rule>
    <rule>
        <id></id>
        <tracker>1512646679</tracker>
        <type>pass</type>
        <interface>lan2</interface>
        <ipprotocol>inet</ipprotocol>
        <tag></tag>
        <tagged></tagged>
        <max></max>
        <max-src-nodes></max-src-nodes>
        <max-src-conn></max-src-conn>
        <max-src-states></max-src-states>
        <statetimeout></statetimeout>
        <statetype><![CDATA[Allow traffic of lan2]]></statetype>
        <os></os>
        <protocol>tcp</protocol>
        <source>
            <any></any>
        </source>
        <destination>
            <any></any>
        </destination>
        <descr></descr>
    </rule>
</filter>

```

OpenVPN configuration

This block contains the configuration of the OpenVPN client. In particular it defined the address:port of the Server, the security parameters (i.e., we choose to use a AES-256-GCM shared key between the Server and the Client), the VPN modality (i.e. site-to-site) and a routing rule for distributing the traffic among the Sections.

```

<openvpn>
    <openvpn-client>
        <auth_user>admin</auth_user>
        <auth_pass>adminPassword</auth_pass>
        <vpnid>1</vpnid>
        <protocol>TCP4</protocol>

```

```

<dev_mode>tun</dev_mode>
<ipaddr></ipaddr>
<interface>wan</interface>
<local_port></local_port>
<server_addr>1.2.3.4</server_addr>
<server_port>8443</server_port>
<proxy_addr></proxy_addr>
<proxy_port></proxy_port>
<proxy_authtype>none</proxy_authtype>
<proxy_user></proxy_user>
<proxy_passwd></proxy_passwd>
<description></description>
<mode>p2p_shared_key</mode>
<topology>subnet</topology>
<custom_options>route 192.168.0.0 255.255.0.0</custom_options>
<shared_key>930C5...SHAREDKEYCONTINUES...</shared_key>
<crypto>AES-128-CBC</crypto>
<digest>SHA1</digest>
<engine>none</engine>
<tunnel_network>10.10.9.0/24</tunnel_network>
<tunnel_networkv6></tunnel_networkv6>
<remote_network>192.168.8.0/24, 192.168.9.0/24</remote_network>
<remote_networkv6></remote_networkv6>
<use_shaper></use_shaper>
<compression></compression>
<auth-retry-none></auth-retry-none>
<passtos></passtos>
<udp_fast_io></udp_fast_io>
<sndrcvbuf></sndrcvbuf>
<route_no_pull></route_no_pull>
<route_no_exec></route_no_exec>
<verbosity_level>1</verbosity_level>
<ncp-ciphers>AES-256-GCM,AES-128-GCM</ncp-ciphers>
<ncp_enable>disabled</ncp_enable>
</openvpn-client>
</openvpn>

```

Federated Administration Monitoring (FAM)

16.1 Dependencies

Install the dependencies

- OS:
 - Windows 7 and newer
 - Windows Server 2008 R2 and newer
- [DotNetCore 1.1.4 Windows Hosting](#)
- [Internet Information Services Manager](#)

To check that all the dependencies have been set up, execute

```
dotnet --version
-> Microsoft .NET Core Shared Framework Host

    Version : 1.1.0
    Build   : 928f77c4bc3f49d892459992fb6e1d5542cb5e86
```

16.2 Administration Manager set-up

To set the service, execute the following commands

```
git clone https://github.com/sunfish-prj/Administration-Manager.git
cd Administration-Manager/manager
dotnet SUNFISH.dll
```

The server is now running and listening on the port chosen in the *hosting.json* file (e.g. 80).

The Administration Manager is expected to interact with the: - [Service Level Agreements Manager](#) - [Service Ledger Interface](#)

The *urls* and *ports* for the above applications may be altered in the *appsettings.production.json* config file.

For the Administration Manager to make use of the mock up IDM, set the *UseIDM* variable in the *appsettings.production.json* to true.

- The current IDM url is *49.118.99.72* using port *44001*
- For the default username & password contact Alexander Tanti.

16.3 SLA Manager set-up

To set the service, execute the following commands

```
git clone https://github.com/sunfish-prj/SLA-Manager.git
cd SLA-Manager
dotnet Sunfish.SLAMService.dll
```

The server is now running and listening on the port chosen in the *hosting.json*. file (e.g. 80).

The SLA Manager is expected to interact with the:

- [Service Ledger Interface](#)

The *urls* and *ports* for the above application may be altered in the *appsettings.production.json* config file.

The codebase of the Configurator is available here [Configurator](#)

The overall structure of the codebase is organised in the following files and directories:

- /client/*
- /configurator/*
- /saltstack/*
- /Vagrantfile

17.1 Installation Steps

There are the following steps for creating a cluster of Kubernetes Nodes:

1. Instantiate the Virtual Machines (configure and execute Vagrantfile)
2. Configure the saltstack master and minions
3. Apply kubernetes states
4. Proceed with cluster operations

In this document we describe the setup of a Kubernetes cluster composed by two nodes.

17.1.1 Vagrant config file

The initial point for the setup process is the file *Vagrantfile*. This file contains all the information needed for the configuration of the Virtual Machines (VMs).

The minimum requirement for setting this infrastructure up is the initialization of four VMs: *Saltstack master*, *kubernetes master* and two minions. Vagrantfile provides the configuration details for each of these four virtual machines, referred respectively as *saltmaster*, *master*, *node1*, and *node2*.

Apart of the mandatory specifications that a common configuration of a VM machine requires (like operating system, hostname, ip), in this infrastructure the VMs should have some additional configurations.

Specifically, for the configuration of saltmaster machine are required:

Source code that saltmaster should run.

This includes two folders: salt and pillar

1. Path to a custom salt master config file
2. Path to a custom salt minion config file
3. Path to master key
4. Path to minion key

Since in Saltstack's perspective the other three VMs are minions, the requirement for their initial configuration are identical:

1. Path to a custom minion config file
2. Path to minion key

Note that to provision of the guest machines is used the Vagrant Salt provisioner, which allows the usage of the Salt states. Also, in the configuration of the saltmaster machine, the folders clients and configurator refer to the necessary libraries and source code for the Configurator API. Configurator API is included inside the saltmaster machine, but practically it can be located to a new VM.

To boot the environment and have all the virtual machines running: `vagrant up`

To access the shell of a running virtual machine: `vagrant ssh [name]`

At the end of this process should be running the saltstack master machine and three minions. At this point, the minions are identical to each other, and they are all connected to saltmaster machine. This can be verified by executing these shell commands:

Enter saltmaster machine: `vagrant ssh saltmaster`

Check the status of the connected minions: `salt-run manage.status`

To require the deployment of the kubernetes master in the minion *master*: `salt-run state.orchestrate k8s_orchestrate`

At this stage, the Kubernetes states may be applied to the remaining minions in order to obtain Kubernetes nodes.

17.1.2 Configuration of Saltstack master

- `/root/conf/*` contains configurations files of saltstack
- `/srv/salt/_states/*` contains custom state module for saltstack
- `/srv/salt/_modules/*` contains custom execution module for saltstack
- `/srv/salt/top.sls` provides mapping between states files and target minions
- `/srv/pillar/top.sls` provides mapping between pillar data and target minions

The Salt Master maintains States and Pillars, located respectively at `/srv/salt/*` and `/srv/pillar/*`.

Salt States (/srv/salt/*)

Salt is a directory hierarchy that contains state files. Each state file describes one or more states to be configured and enforced on the targeted machines. `top.sls` is the first file that will be processed, and it determines the SLS files to run on particular minions. Pillar Data (/srv/pillar/*).

Pillar is managed in a similar way as the Salt states. Pillar data is used to configure the specific data that should be distributed to minions. This directory contains the `top.sls` file and the pillar files. `Top.sls` provides a mapping between the minions and the pillar files.

All the pillar files provide the data targeted to a minion, described as *key:value*.

Keys

`/root/k8s_keys/*` contains the list of keys for the target minions.

Applying the states

From SaltStack Master can be applied the states for the minions machines. The execution of `state.orchestrate` on saltstack master provides a stateful management of the entire infrastructure by allowing control over the application of asynchronous states on different minions.

```
vagrant ssh saltmaster
sudo salt-run state.orchestrate k8s_orchestrate
```

17.1.3 Configuration of Kubernetes

- `/srv/salt/k8s/*` contains the kubernetes formulas
- `/srv/salt/k8s_yaml/*` contains the files for kubernetes deployments
- `/srv/salt/k8s_orchestrate.sls` is the file that manages kubernetes installation
- `/srv/salt/k8s_orchestrate.sls` provides the configurations required to deal with the creation of the cluster. Specifically, this file defines the states that should be applied for:

Certificate Authority (CA)

In `/srv/pillar/k8s_common.sls` should be defined the list of nodes authorized to get a signed certificate by CA and the location of the public keys.

Kubernetes Master

In `/srv/pillar/k8s_master.sls` should be defined the kubernetes nodes that are going to be deployed, the labels applied to them, and the location of the yaml files for service deployment.

Kubernetes Nodes

and `/srv/pillar/k8s_node.sls` contains the configurations that should be assigned to target minions like the IP range of the cluster services.

Apply node to label

This action requires to define the name of the kubernetes node and the label in the pillar file `/srv/pillar/k8s_master.sls`. After this, should be applied the custom state: `salt "master" state.apply k8s.master.node_label`

Deploy a service

The file with the configuration of the service should be located at `/srv/salt/k8s_yaml/*` (e.g `new.yaml`). This file should be included in the configurations of `/srv/pillar/k8s_master.sls`. The new state should be applied: `salt "master" state.apply k8s.master.deploy_yaml`

Custom state module

`/srv/salt/_state/k8s_custom.py` contains the implementation of functions for different custom states of kubernetes:

- `label_node_present`
- `node_cordoned`
- `node_uncordoned`
- `node_drained`
- `node_absent`
- `yaml_applied`
- `node_labels`

Custom execution module functions

`/srv/salt/_modules/k8s_custom.py` contains the implementation of functions for different custom modules:

- `get_node_list` lists the information of all nodes in the cluster
- `get_pods_list` lists the information of all pods in the cluster
- `get_svc_list` lists the information of all services in the cluster
- `get_node` lists the information of a single node

Configurator

`/root/clients/*` saltstack client

- `/srv/salt/salt_rest_api.sls` provides the configuration of the saltstack client (such as keys, users, modules etc.)
- `/root/configurator/*` configurator API
- `/srv/salt/configurator.sls` provides the required packages for Configurator API

To set the configurator up, is required the installation of the saltstack client library on the saltstack master machine. This library is located at `/root/clients/*` (specified in Vagrantfile). In the same way, the Configurator API is located at `/root/configurator/*`. Note that, for simplicity, in Vagrantfile, is specified to install the Configurator API in SaltStack master machine, but this API can be installed in any deployed virtual machine.

17.2 Installation

- `sh /root/configurator/setup.sh`
- `sh /root/configurator/start.sh`

In the end of these steps, the minions of the saltstack, which are not configured as saltstack master or kubernetes master, will be stored as Virtual Machines resources.

Configurator API will be running at port 8443, and it can be accessed at `https://IP:8443/api/configurator/v1`

17.2.1 API Calls

- **/confVMS** Modify virtual machine with `vmID="node1"` `curl -ik https://localhost:8443/api/configurator/v1/confVMS/node1 -H "Content-Type: application/json" -X PUT -d '{"vmID":"node1","vmType":"containerized","confID":"1"}'`.
- **/confNodes** Configure a kubernetes node for the VM with `id="node1"` `curl -ik https://localhost:8443/api/configurator/v1/confNodes -H "Content-Type: application/json" -X POST -d '{"vmID":"node1","nodeID":"node1","labels":{"label_key":"123","label_key2":"qwerty"}}'`.
Remove the node with `nodeID="node1"` from kubernetes cluster `curl -ik https://localhost:8443/api/configurator/v1/confNodes/node1 -H "Content-Type: application/json" -X DELETE`
- **/jobs** List all the async operations added in a queue `curl -ik https://localhost:8443/api/configurator/v1/jobs -H "Content-Type: application/json" -X GET`

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.

18.1 Setting-Up a Service Tenant

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

18.1.1 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.

18.1.2 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*.

In addition, a special PIP with SLI interface can be used. This pip is automatically deployed on the infrastructure tenant as `/pip-sli` and retrieves host, tenant, and PEP for a service from the SLI. It requires `SLI` to be set to the endpoint of the SLI to use. To use it, set `USE_PIP_SLI` in the service's `config.sh` and add the endpoint of this PIP (running on the infrastructure tenant) to `PEP_URLS_PIPS`.

- `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 SUNISH 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. The proxy interprets the first part of any path as `service_id` and strips it from the request forwarded to the PEP declared for `<service_id>`.

18.1.3 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 and extract the ‘Docker Latest’ release from the *Releases* tab in the GitHub repository.
- The preconfigured docker containers *demotenant.tar* and *infra.tar* need to be loaded: `docker load -i demotenant.tar` `docker load -i infra.tar`
- The deployment script of the infrastructure has to be executed (`./deploy.sh`) inside the infrastructure directory
- The IP address of the docker container will be printed out. Copy it and press Enter.
- This address needs to be configured as infrastructure tenant IP address for the demotenant container in the file `tenant/config.sh`
- Execute the demo tenant by invoking `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 in the same container on `TOMCAT_PORT`. Both ports are mapped to their respective counterparts on the host machine.

The demo tenant includes a demo application, but no DS policies. A postman collection is also part of the ‘Docker Latest’ release. It contains a sample policy allowing access to `/demo-app/demo/ds/index.*` and denying everything else. It should work out-of-the-box for the default setup. The default configuration of the proxy assigns the demo application the identifier ‘demo’. Consequently, the demo application (and the sample DS policy) can be tested, by invoking `http://localhost:10000/demo/demo-app/demo/ds/index.html`. The DS component can also be bypassed for debugging purposes by connecting to `http://localhost:8081/demo-app/demo/ds/index.html`.

18.1.4 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

18.1.5 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>
    </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:request" AttributeId="urn:sunfish:attribute:request:path" DataType="http://
↳ www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
    </Match>
  </AllOf>
</AnyOf>
</Target>
<Rule RuleId="urn:sunfish:rule:permit" Effect="Permit">
  <Target/>
</Rule>
</Policy>
```

18.2 Setting-Up an Infrastructure Tenant

18.2.1 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.

18.2.2 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.

18.2.3 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.

Intelligent Workload Manager (IWM)

19.1 Deployment instruction

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
```

19.2 Screenshots

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

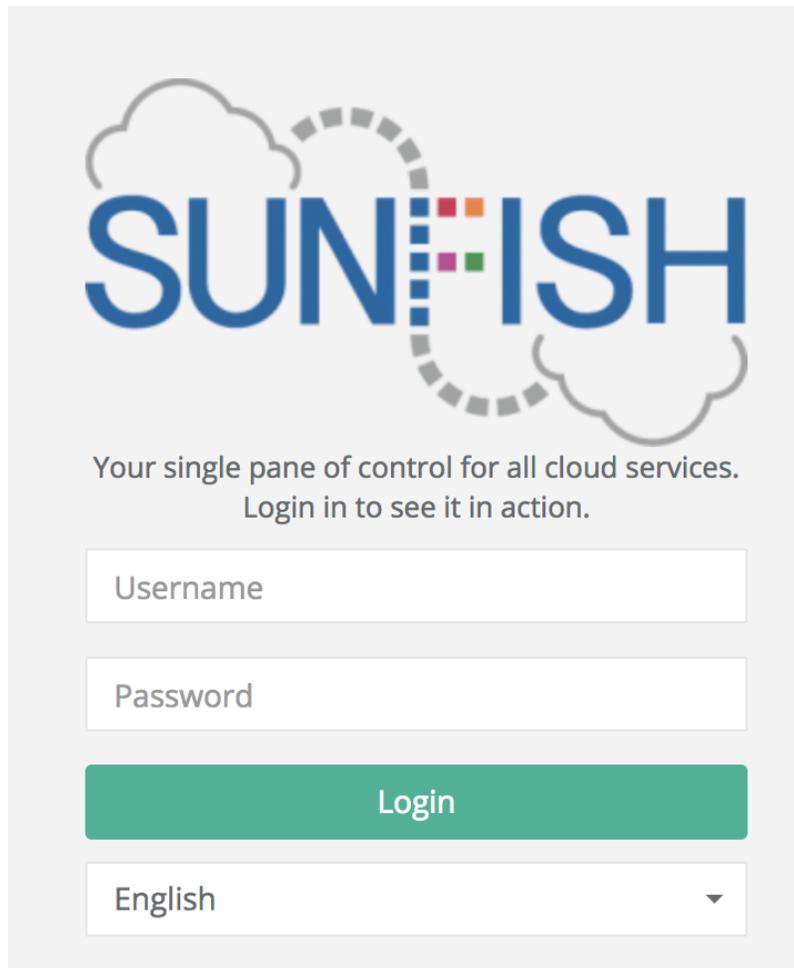


Fig. 19.1: Login view of IWM frontend, white-labelled to a concrete federation.

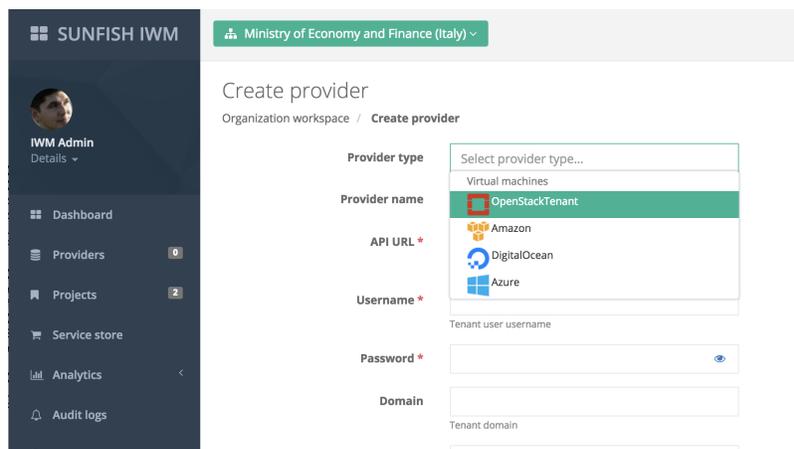


Fig. 19.2: Adding federation service providers to IWM.

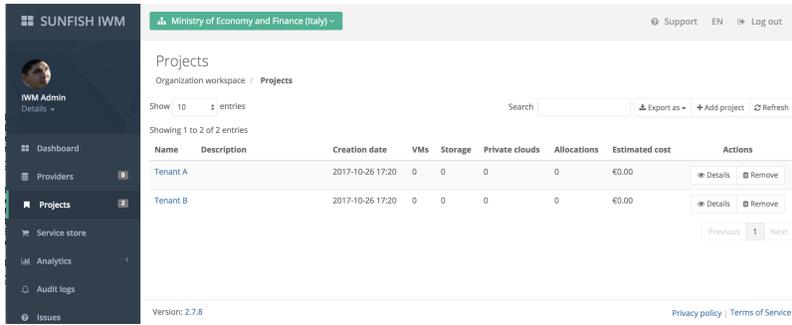


Fig. 19.3: Listing registered SUNFISH tenants within an IWM.

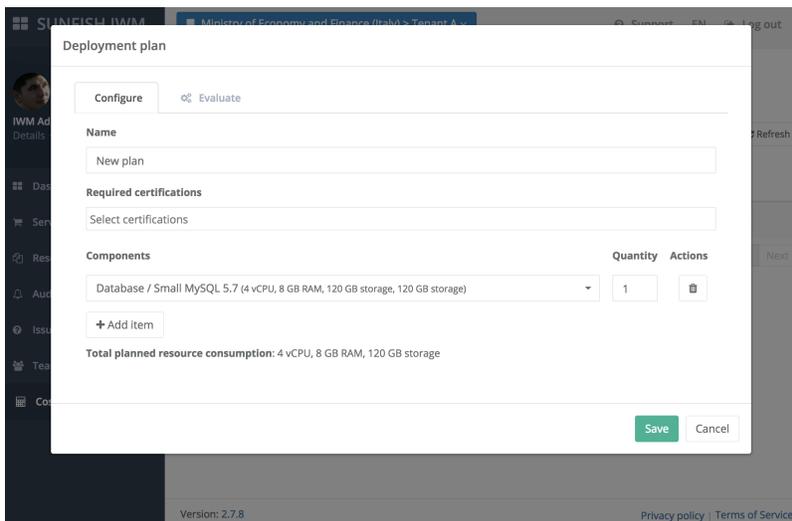


Fig. 19.4: Visual interface to optimisation API for finding the best option for a planned infrastructure.

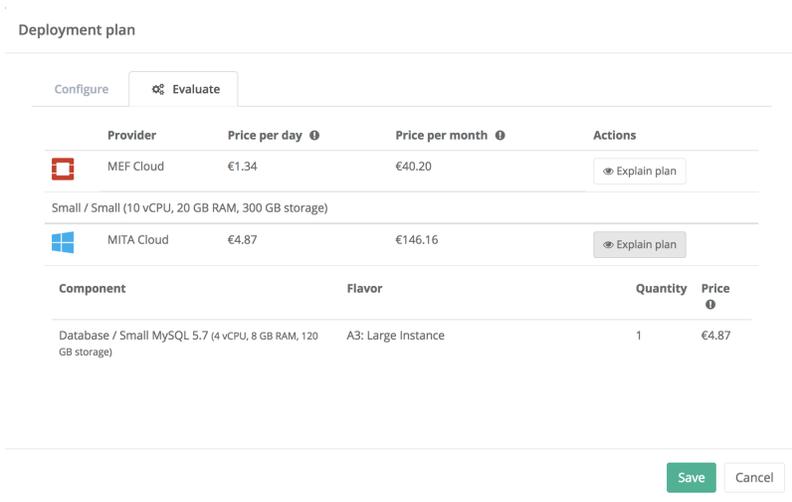


Fig. 19.5: Results of the optimisation with 2 service providers in the federation.

Anonymisation (ANM)

This is the installation for the anonymisation component.

The anonymization service has a single dependency, namely Java8 run time environment.

The service installation procedure is very simple:

1. Place AnonymizationService.jar in your target directory.
2. Place AnonymizationService_lib/ directory in the same target directory.
3. Create an 'application.properties' file and set the port number.

For example, the application.properties can contain the following line: *server.port=50001*

To run the service, execute the following command:

```
> java -jar AnonymizationService.jar
```

The service is now ready to accept REST calls.

20.1 Anonymisation Interface (ANI)

Install the dependencies

Nodejs v6.x Npm v3.x Releases and installation guides can be found on the official websites [here](#) and [here](#)

To check that all the dependencies have been set up, execute

```
$ node -v
-> v6.1.0
$ npm -v
-> 3.10.6
```

Anonymisation Interface set-up

To set the service, execute the following commands

```
$ git clone https://github.com/sunfish-prj/Anonymisation-Interface.git
$ npm start
```

The server is now running and listening on the port chosen in the `config/server_config.yaml` file (e.g. 50001).

The anonymisaiton interface is expected to interact with the anonymisation component in the Service Ledger Interface, whose url and port are defined in the configuration file `config/default.yaml`.

Data Masking (DM)

This is the installation for the data masking component.

The masking service has a single dependency, namely Java8 run time environment.

The service installation procedure is very simple:

1. Place MaskingService.jar in your target directory.
2. Create an 'application.properties' file and set the port number. For example, the application.properties can contain the following line: server.port=50002
3. Create/Modify the 'sunfish.properties' file (located within the jar) and set URI's for the Service Ledger. For example:

riStoreUri=http://localhost:60005/r/put
teUri=http://localhost:60005/r/delete

riReadUri=http://localhost:60005/r/get

riDele-

To run the service, execute the following command:

```
> java -jar MaskingService.jar
```

The service is now ready to accept REST calls

Federated Runtime Monitoring (FRM)

The installation of the FRM component is here listed.

22.1 Proxy

The proxy component has been developed as a servlet filter in order to be compatible with the DS servlets. It is released as a .jar dependency in such a way to be totally integrated in the Tomcat Server. The proxy 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 additional libraries supplied in the `src/main/webapp/WEB-INF/lib` directory that need to be properly added into the java path during the deployment.

22.1.1 Installation guide

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

N.B. The following instructions refer to the dockerised setup of the SUNFISH Data Security Enforcement Infrastructure. Follow the doc for the deployment.

1. Download the last release of the Proxy Infrastructure tenant from the *Releases* tab in the GitHub repository [release](#).
2. Once loaded the preconfigured docker containers *demotenant.tar* and *infra.tar* open the infrastructure folder and copy from the `Federation-Monitoring/install/ds-infrastructure/` folder the following elements:
 - `./params.json` this contains the configuration params for the ProxyFilter
 - `./web.xml` this contains a new version of the configuration for Tomcat. It enables the filter-mapping.

- `./deploy.sh` this is the modified script to launch the infrastructure tenant. It load all required dependencies and configurations in Tomcat.
3. Copy `ProxyFilter-1.0-release.jar` download at 1. in `Federation-Monitoring/install/ds-infrastructure/dependencies/`
 4. Copy the dependencies/ inside the infrastructure folder.

Once finished you should have inside the DS module infrastructure and the tenant folders. Your infrastructure folder should looks like this:

```
ubuntu@ubuntu:~/Data-Security/ds/doc/install/docker~/infrastructure$ ls -la
attributes
config.sh
dependencies
deploy.sh
infra.tar
params.json
web.xml
```

22.1.2 Usage guide

In `./params.json` must be defined the configuration directives. Configure the following paameters:

- **ServerIP** #It is the IP of the localhost where the dockers are running.
- **Path** #It is the path for calling the API of the monitoring service.

Once completed the configuration, run the `./deploy.sh` script to execute the infrastructure tenant with the Proxy-Filter.

By using the demo application the ProxyFilter intercept requests as the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:xsi=
↳ "http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
↳ "urn:oasis:names:tc:xacml:3.0:core:schema:wd-17 http://docs.oasis.org/
↳ xacml/3.0/xacml-core-v3-schema-wd-17.xsd" ReturnPolicyIdList="false"
↳ CombinedDecision="false">
    <Attributes Category="urn:sunfish:attribute-category:service
↳ ">
        <Attribute AttributeId=
↳ "urn:sunfish:attribute:id" IncludeInResult="false">
            <AttributeValue DataType=
↳ "http://www.w3.org/2001/XMLSchema#string">demo</AttributeValue>
        </Attribute>
        <Attribute AttributeId=
↳ "urn:sunfish:attribute:service:zone" IncludeInResult="false">
            <AttributeValue DataType=
↳ "http://www.w3.org/2001/XMLSchema#string">demozone</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="urn:sunfish:attribute-
↳ category:application">
        <Attribute AttributeId=
↳ "urn:sunfish:attribute:id" IncludeInResult="false">
            <AttributeValue DataType=
↳ "http://www.w3.org/2001/XMLSchema#string">TBD?!!</AttributeValue>
        </Attribute>
```

```

        <Attribute AttributeId=
↪"urn:sunfish:attribute:application:zone" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#string">demozone</AttributeValue>
        </Attribute>
        <Attribute AttributeId=
↪"urn:sunfish:attribute:application:host" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#string">TBD?!!</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="urn:sunfish:attribute-category:request
↪">
        <Attribute AttributeId=
↪"urn:sunfish:attribute:request:method" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#string">GET</AttributeValue>
        </Attribute>
        <Attribute AttributeId=
↪"urn:sunfish:attribute:request:path" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#string">/demo-app/demo/ds/index.html</
↪AttributeValue>
        </Attribute>
        <Attribute AttributeId=
↪"urn:sunfish:attribute:request:port" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#integer">80</AttributeValue>
        </Attribute>
        <Attribute AttributeId=
↪"urn:sunfish:attribute:request:protocol" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#string">http://</AttributeValue>
        </Attribute>
        <Attribute AttributeId=
↪"urn:sunfish:attribute:request:content-type" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#string">application/json</AttributeValue>
        </Attribute>
        <Attribute AttributeId=
↪"urn:sunfish:attribute:request:body-data" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#string">sfbd20812981</AttributeValue>
        </Attribute>
        <Attribute AttributeId=
↪"urn:sunfish:attribute:request:content-type" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#string">text/xml</AttributeValue>
        </Attribute>
        <Attribute AttributeId=
↪"urn:sunfish:attribute:request:header-parameter" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#string">sfhp021</AttributeValue>
        </Attribute>
        <Attribute AttributeId=
↪"urn:sunfish:attribute:request:header-parameter" IncludeInResult="false">
            <AttributeValue DataType=
↪"http://www.w3.org/2001/XMLSchema#string">sfhp101</AttributeValue>

```

```

                </Attribute>
            </Attributes>
        </Request>

```

and send to the Service Ledger Monitoring the following json:

```

{
    "timeStamp": "2017-12-13 17:47:21",
    "requestorID": "TODO",
    "data":
    ↪ "W0RvY3VtZW50iAgTm8gRE9DVF1QRSBkZWNSYXJhdGlvbiwgUm9vdCBpcyBbRWxlbWVudDogPFJlcXVlc3QgW05hbWVzZ
    ↪ ",
    "dataType": "REQUEST",
    "loggerID": "PDP",
    "token": "TODO",
    "monitoringID": "/demo-app/demo/ds/index.html"
}

```

22.2 Chaincode

The code to be deployed is available [here](#).

22.2.1 Installation Guide

The chaincode has been implemented for the blockchain system Hyperledger Fabric v1.0.0. Its installation and deployment instruction can be found in the guide.

1. To install the chaincode named *monitoring* is as follow

```

peer chaincode install -n ex02 -v 1.0 -p github.com/hyperledger/fabric/sunfish/
↪chaincode/monitoring.go

```

We proceed now with the instantiation for the code for its actual running. For the sake of simplicity, we assume a blockchain network formed by two peers; the procedure can be extended for any number of peers.

2. Instantiate the chaincode on peer0 or peer2:

```

peer chaincode instantiate -o orderer0:7050 --tls $CORE_PEER_TLS_ENABLED --cafile
↪$ORDERER_CA -C mychannel -n ex02 -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -
↪P "OR ('Org0MSP.member','Org1MSP.member')"

```

As a result, a new docker container is now created to manage the chaincode, the following log is showed:

```

peer2      | 2017-06-20 18:32:40.189 UTC [dockercontroller] Start -> DEBU 3b6 Start_
↪container dev-peer2-ex02-1.0
peer2      | 2017-06-20 18:32:40.189 UTC [dockercontroller] getDockerHostConfig ->_
↪DEBU 3b7 docker container hostconfig NetworkMode: e2ecli_default
peer2      | 2017-06-20 18:32:40.190 UTC [dockercontroller] createContainer -> DEBU_
↪3b8 Create container: dev-peer2-ex02-1.0
peer2      | 2017-06-20 18:32:40.192 UTC [dockercontroller] Start -> DEBU 3b9 start-
↪could not find image ...attempt to recreate image no such image
peer2      | 2017-06-20 18:32:40.192 UTC [chaincode-platform] generateDockerfile ->_
↪DEBU 3ba
peer2      | FROM hyperledger/fabric-baseos:x86_64-0.3.0

```

```

peer2 | ADD binpackage.tar /usr/local/bin
peer2 | LABEL org.hyperledger.fabric.chaincode.id.name="monitoring" \
peer2 | org.hyperledger.fabric.chaincode.id.version="1.0" \
peer2 | org.hyperledger.fabric.chaincode.type="GOLANG" \
peer2 | org.hyperledger.fabric.version="1.0.0-snapshot-ecc29dd" \
peer2 | org.hyperledger.fabric.base.version="0.3.0"
peer2 | ENV CORE_CHAINCODE_BUILDLEVEL=1.0.0-snapshot-ecc29dd
peer2 | ENV CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/peer.crt
peer2 | COPY peer.crt /etc/hyperledger/fabric/peer.crt
peer2 | 2017-06-20 18:32:51.945 UTC [dockercontroller] deployImage -> DEBU 3bb_
↪Created image: dev-peer2-ex02-1.0
peer2 | 2017-06-20 18:32:51.945 UTC [dockercontroller] Start -> DEBU 3bc start-
↪recreated image successfully
peer2 | 2017-06-20 18:32:51.945 UTC [dockercontroller] createContainer -> DEBU_
↪3bd Create container: dev-peer2-ex02-1.0

```

Note: the `-o` flag indicates the orderer address:port who handle the channel; the `--tls` takes a boolean and indicates whether we are using TLS or not (it's true in our environment, look at the docker-compose file); the `--cafile` indicates the file with the certificate of the Orderer (ignore it for now); the `-c` flag indicates the args to give to the chaincode in a json format and finally `-P` is referred to the endorsement policy (we will see later in this course).

Now a new docker container should be up with the chaincode execution on the peer. Via the command `docker ps` the new docker can be shown. While via the command `docker attach <ID>` where `ID` is the CONTAINER ID discovered in the previous step with `docker ps`, the docker can be accessed to issue execution.

3. To query the chaincode on the initialised argument to check if they have been correctly stored

```
peer chaincode query -C mychannel -n ex02 -c '{"Args":["query","a"]}'
```

while to run an execution the following command can be executed

```
peer chaincode invoke -o orderer0:7050 --tls $CORE_PEER_TLS_ENABLED --cafile
↪$ORDERER_CA -C mychannel -n ex02 -c '{"Args":["invoke","a","b","10"]}'
```

Federated Security Audit (FSA)

This is the installation for the FSA component.

To install FSA extract the provided archive *FSA.tar.gz*. The extracted FSA folder has the following structure:

- bins/ - scripts and jars of the FSA application
- docs/ - documentation files with usage instructions
- rootFolder/ - containing input, output and intermediate files for FSA operations: CreateModel, IdentifySuspiciousActivities and GetEntitlementVulnerabilites
- activities_arch/ - IdentifySuspiciousActivities processed input files
- activities_in/ - IdentifySuspiciousActivities input files
- activities_out/ - IdentifySuspiciousActivities result files
- model_arch/ - CreateModel processed input files
- model_in/ - CreateModel input files
- model_out/ - CreateModel current model
- entitlement_arch/ - GetEntitlementVulnerabilites processed input files
- entitlement_in/ - GetEntitlementVulnerabilites input files
- entitlement_out/ - GetEntitlementVulnerabilites result files
- /spark - local installation of spark

To use FSA application follow the following steps:

1. Start FSA application by executing bins/FSA_start.sh
2. Add input files to rootFolder/XXX_in, select XXX according to the required operation (as described above).
3. Get result files from rootFolder/XXX_out, where XXX is the same as was selected in the previous step
4. Go to step 2 or stop FSA application

Secure Multiparty Computation (SMC)

24.1 Sharemind MPC Application Server

24.1.1 Requirements

- **Minimal hardware requirements:** 8 GB of RAM, 2 CPU cores. Optimal requirements depend on planned workloads.
- **Operating system:** Debian (Stretch), Ubuntu (16.04 or newer) or other APT-based Linux distribution.

24.1.2 Installation

Sharemind MPC is available from Cybernetica's private APT repository. To access it, add the repository location to your APT configuration and trust the Sharemind package signing GPG key.

```
gpg --keyserver subkeys.pgp.net --search sharemind-packaging@cyber.ee # Note the ↪
↪downloaded key ID
gpg -a --export <key ID> | sudo apt-key add - # Supply the key ID here

echo "deb https://repo.cyber.ee/sharemind/apt/debian stretch main" | sudo tee /etc/
↪apt/sources.list.d/sharemind-mpc.list
sudo apt-get update
```

Install Sharemind Application Server with the *shared3p* SMC module and HDF5 embedded storage backend:

```
sudo apt-get install sharemind-server sharemind-mod_shared3p sharemind-mod_tabledb-
↪hdf5
```

24.1.3 Key Generation and Exchange

Sharemind MPC uses Transport Layer Security (TLS) technology for secure, mutually authenticated and encrypted communication channels between computation nodes as well as between computation nodes and client applications

like CSV Importer. Therefore, each Sharemind component requires a personal asymmetric key pair for authentication and encryption.

As Sharemind MPC uses RSA keys in standard X.509 certificate format, already familiar tools like OpenSSL can be used. The process described below generates a new RSA key pair, where the private key is in file `my-private-key` and public part in `my-public-key`. The `-days` value should be at least the expected duration of the deployment.

```
openssl req -x509 -days 300 -nodes -newkey rsa:2048 -keyout my-private-key -out my-  
↪public-key -outform der
```

```
Generating a 2048 bit RSA private key  
.....+++  
...+++  
writing new private key to 'my-private-key'  
-----  
You are about to be asked to enter information that will be incorporated  
into your certificate request.  
What you are about to enter is what is called a Distinguished Name or a DN.  
There are quite a few fields but you can leave some blank  
For some fields there will be a default value,  
If you enter '.', the field will be left blank.  
-----  
Country Name (2 letter code) [AU]: *(Required)**  
State or Province Name (full name) [Some-State]: *(Required)**  
Locality Name (eg, city) []: *(Required)**  
Organization Name (eg, company) [Internet Widgits Pty Ltd]: *(Required)**  
Organizational Unit Name (eg, section) []:  
Common Name (e.g. server FQDN or YOUR name) []: *(Required)**  
Email Address []:
```

This `openssl` tool generates the private key by default in PEM format. However, for Sharemind MPC, it must be converted to DER format:

```
openssl rsa -in my-private-key -out my-private-key -outform der
```

The public key must be sent to each other participant in the SMC deployment that communicates directly with the Sharemind Application Server. This includes other (two) Sharemind Application Servers and client or proxy applications. The public key should be sent in a way that allows sender authentication, e.g. by digitally signing it.

24.1.4 Configuration

In every deployment, all three Sharemind Application Server hosts have to agree on unique server names and their order. Before continuing, make sure you have the following information about the deployment and other hosts:

- Deployment name
- **For each Sharemind Application Server:**
 - Node number (1, 2 or 3)
 - Name
 - Hostname or IP
 - Port number
 - Public key file

Sharemind Application Servers search for their main configuration file from the following locations (in order):

- Filename given by the `--conf` command line argument
- System-wide configuration file in `/etc/xdg/sharemind/server.conf` (XDG Basedir search path)
- System-wide configuration file in `/etc/sharemind/server.conf`

The configuration file is an INI-formatted file, where section names are between square brackets (`[Section]`) and configuration values are given with `key=value` pairs. A commented example server configuration file is available in `/usr/share/doc/sharemind/examples/server.conf`.

At minimum, the following changes to the example configuration are necessary:

- The value of `UuidNamespace` in section `[Server]` must be set to your deployment name.
- The value of `Name` in section `[Server]` must be set to your server's unique name.
- The value of `ListenInterfaces` in section `[Network]` must be set to an IP address and port number, where the Sharemind Application Server listens for incoming connections. If the server should listen only on a single network interface, insert its IP address. Otherwise, specify `0.0.0.0`. The port number can be chosen according to personal preference, keeping in mind that listening on low port numbers (up to 1023) requires root access.
- The values of `PublicKeyFile` and `PrivateKeyFile` in section `[Network]` should be the file names of your public and private keys, respectively. Following the example key generation procedure given above, these are `my-public-key` and `my-private-key`. File location can be given relative to the current configuration file with `%{CurrentFileDirectory}`, e.g. `%{CurrentFileDirectory}/keys/my-public-key`.
- Information about the other two servers is in sections `[Server <name>]`, where `<name>` is the unique agreed upon name of the respective server. For both servers, the following values should be changed: `Address`: server's IP address, `Port`: server's port number and `PublicIdentity`: file name of the corresponding server's public key file. File location can be given relative to the current configuration file with `%{CurrentFileDirectory}`.

Additionally the file `shared3p.conf` should be changed so that server names are assigned to the correct identifiers (node numbers). This configuration has to be identical for all three servers so they know their communication order in the secure multi-party protocols. The system-wide copy of this file is in `/etc/sharemind/shared3p.conf`.

Other parts of the configuration files should remain unchanged, as network and security parameters must be consistent for all servers and client applications.

Each client application or proxy owner also generates a key pair and sends its public key to each server host. The Sharemind Application Server only allows incoming connections from client applications whose public key is registered in its access control list, referenced by the `WhiteListFile` from the main configuration file (by default `%{CurrentFileDirectory}/server-whitelist.conf`). The format of this whitelist file is as follows:

```
# Format:
# path/to/public-key-filename: script-filename1[, script-filename2, ...] # Ignored
↪comment
# Example:
# key1: script1, script2 # Allow running only 'script1' and 'script2' with public
↪key 'key1'.
# key2: * # Allow 'key2' to run any script. NB! Should not be used
↪in production!
client-public-key: secrec-program.sb
```

24.1.5 Compiling SecreC Code

Each Sharemind Application Server host must audit and deploy necessary SecreC programs separately. SecreC code is compiled into bytecode with the `scc` program and by default, Sharemind Application Server looks for the bytecode

from `/var/lib/sharemind/scripts/` folder:

```
scc -o /var/lib/sharemind/scripts/program.sb /path/to/src/program.sc
```

24.1.6 Starting Sharemind Application Server

A system-wide installation of the Sharemind Application Server is controlled by the systemd unit file:

```
sudo systemctl start sharemind-server
sudo systemctl stop sharemind-server
```

24.2 Sharemind Web Application Gateway

Sharemind platform components communicate using a binary protocol. To support web-based client applications, a proxy service has to be deployed in front of each Sharemind Application Server that translates between HTTP and Sharemind’s binary protocol. Such proxy applications can be built with Sharemind Web Application Gateway add-on – a NodeJS library that must be installed separately on the server:

24.2.1 Installation

```
sudo apt-get install sharemind-web-gateway
```

Proxy applications written in NodeJS should then add “*sharemind-web-gateway*” as a dependency in their *package.json* file. Sharemind Web Application Gateway requires NodeJS version 6 or newer. If a suitable version is not provided directly by the Linux distribution, it can be obtained from [NodeJS site](#).

24.2.2 Configuration

A proxy application using Sharemind Web Application Gateway acts as a client application for the Sharemind Application Server. Thus, it also needs a key pair for TLS as described in *Key Generation and Exchange*.

An example client application configuration, also suitable for proxy applications, can be found in `/usr/share/doc/sharemind/examples/client.conf`. At minimum, the following changes to the example configuration are necessary:

- The value of `UuidNamespace` in section `[Controller]` must be set to your deployment name.
- The values of `PublicKeyFile` and `PrivateKeyFile` in section `[Network]` should be the file names of your public and private keys, respectively. Following the example key generation procedure given at *Key Generation and Exchange*, these are `my-public-key` and `my-private-key`. File location can be given relative to the current configuration file with `%{CurrentFileDirectory}`, e.g. `%{CurrentFileDirectory}/keys/my-public-key`.
- Proxy applications only connect to one Sharemind Application Server and thus have only one `[Server <name>]` section, where `<name>` is the unique agreed upon name of the respective server. The following values should be changed: `Address`: server’s IP address, `Port`: server’s port number and `PublicIdentity`: file name of the corresponding server’s public key file. File location can be given relative to the current configuration file with `%{CurrentFileDirectory}`.

25.1 Dependency

Install the dependencies

- *Node.js v6.x*
- *Npm v3.x*

Releases and installation guides can be found on the official web-sites [node](#) and [npm](#).

To check that all the dependencies have been set up, execute

```
$ node -v
-> v6.1.0
$ npm -v
-> 3.10.6
```

Note that you probably also need to have installed on the machine *make* and *gyp*; if not installed, execute also

```
$ apt-get install build-essential git
$ npm install -g node-gyp
```

Remember to execute the installation as superuser.

Additionally, to install the underlying platform we have two different options

1. *MongoDB* to be installed according to the used OS. To check installation outcomes

```
$ mongo --version
-> MongoDB shell version v3.4.6
```

2. *Hyperledger Fabric* blockchain to be installed according to the [guide](#).

25.2 Service Ledger Interface

To set the service, execute the following commands

```
$ git clone https://github.com/sunfish-prj/Service-Ledger-Interface.git
$ cd Service-Ledger-Interface/server
$ npm start
```

The server is now running and listening on the port chosen in the *config/default.yaml* file (e.g. 8089). You can use the *client-stub interface* <http://localhost:8089/docs>.

The Service-Ledger-Interface is expected to interact with the Service Ledger whose *url* and *port* are defined in the configuration file *config/default.yaml*.

25.3 Service Ledger

To set the service, execute the following commands

```
$ git clone https://github.com/sunfish-prj/Service-Ledger.git
$ cd Service-Ledger/server
```

To set up the ServiceLedger server, edit the file *config/default.yaml* properly. In this file it can be set to use MongoDB or Hyperledger Fabric.

In case the server is using MongoDB, you should also start it before ServiceLedger. See the corresponding command wrt your os [here](#). Similarly, if using Hyperledger Fabric, you need a running Fabric cluster to gather all information related to its docker containers.

Now you can start the ServiceLedger server:

```
$ npm start
```

The server is now running and listening on the port chosen in the *config/default.yaml* file (e.g. 8090). You can use the [client-stub interface](<http://localhost:8090/docs>).

25.4 Usage Guide

The Service Ledger Interface API is the expected entry-points for the SUNFISH platform components.

The Service Ledger API is a general-purpose invocation for blockchain underlying platform. The following parameters are expected:

- Chaincode: the name of the chaincode to invoke.
- Function: the name of the function of the chaincode to invoke.
- Argument: the arguments to give in input to the function.

Therefore, this API has been implemented to realise the monitoring functionality of the chaincode and, most of all, for the remaining number of chaincode part of the federation; namely anonymisation, key value store and the smart contract used for the Use Case 1.

UC-1: Cross-cloud payslip calculation

We refer to online videos on

[High-level video on the use case](#)

[Here](#) how the SUNFISH solution at work to realise the service

UC-2: Private and Public Clouds for Tax Calculation

We refer to online videos on

[High-level video on the use case](#)

[Here](#) how the SUNFISH solution at work to realise the service

UC-3: Federation-based Intelligent Shared Index

We refer to online videos on
[High-level video on the use case](#)
[Here how the SUNFISH solution at work to realise the service](#)

28.1 Deployment Instructions

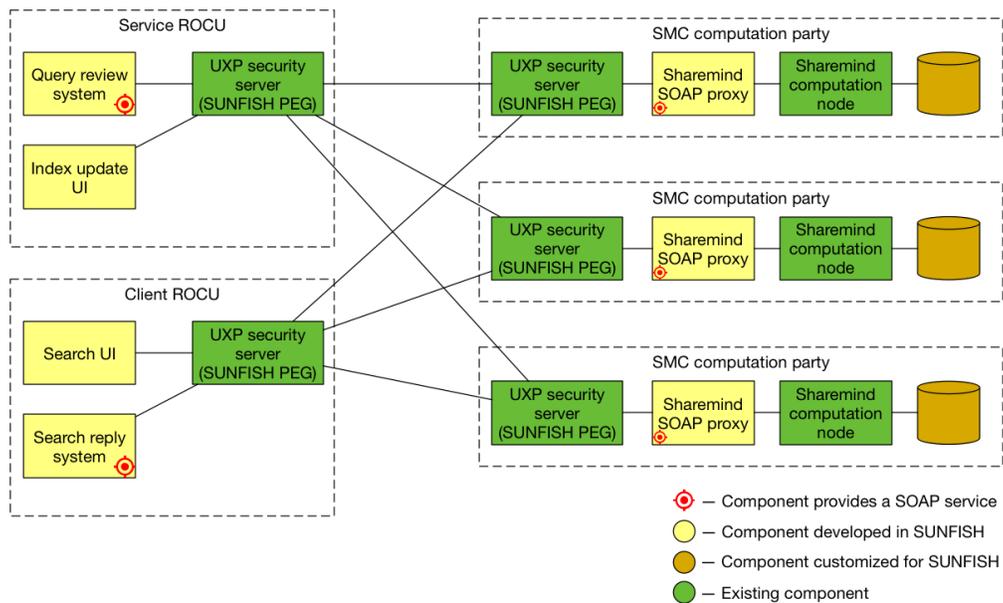


Fig. 28.1: SMC Architecture of the Intelligent Shared Index.

28.1.1 Index Service

SMC nodes together provide the shared intelligence index service. The following steps have to be completed on each SMC node. At each SMC node, the Sharemind Application Server is accompanied by a proxy application that implements a SOAP interface so the communication can be routed through the Unified eXchange Platform (UXP).

First, install and configure Sharemind Application Server as described in [Instructions for Deploying SMC](#). The SOAP proxy also requires a special version of Sharemind Web Application Proxy that uses SOAP for its transport layer.

Clone and run the SOAP proxy:

```
git clone https://github.com/sunfish-prj/Secure-Multiparty-Computation
cd Secure-Multiparty-Computation/usecase/index-service

# Install NodeJS dependencies
npm install

# Compile SecreC code
scc -o /var/lib/sharemind/scripts/add-document.sb secrec/add-document.sc
scc -o /var/lib/sharemind/scripts/add-owners.sb secrec/add-owners.sc
scc -o /var/lib/sharemind/scripts/search.sb secrec/search.sc

# Configure Sharemind Application Server names and ROCU Service addresses in gateway.
→ js

# Run SOAP proxy
node gateway.js <node number> <IP> <port> <configuration file>
```

28.1.2 ROCU Service

Because of the query review and oblivious notification systems, ROCU-s also act as SOAP services in addition to being SOAP clients. Node v6 or newer and npm are required to deploy ROCU Service. In addition, a copy of Sharemind MPC JavaScript Client library is required.

Clone and run the ROCU Service:

```
git clone https://github.com/sunfish-prj/Secure-Multiparty-Computation
cd Secure-Multiparty-Computation/usecase/rocu-service

# Point Sharemind MPC JavaScript Client library (sharemind-web-client) to a local_
→ copy in package.json

# Install NodeJS dependencies
npm install

# Configure Index Service addresses in rocu-service.js

# Run ROCU Service
node rocu-service.js <IP> <port>
```