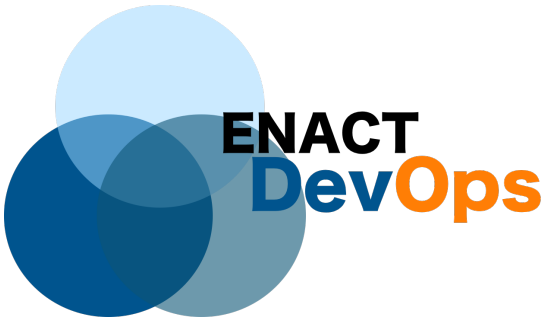


Title:	*ENACT DevOps Framework - First version*
Authors:	*Alexander Palm (UDE), Jacek Dominiak (BAW), Nicolas Ferry (SINTEF), Hui Song (SINTEF), Franck Dechavanne (CNRS), Stéphane Lavirotte (CNRS), Jean-Yves Tigli (CNRS), Diego Rivera (Montimage), Anne Gallon (Evidian), Christophe Guionneau (Evidian), Samuel Mamuye (Evidian), Jean-Christophe Durieu (Evidian), Elena González (BAW), Victor Muntés (BAW), Daniel Rudzinski (BAW), Eider Iturbe (Tecnalia), Saturnino Martinez (Tecnalia), Angel Rego (Tecnalia) and Borja Urkizu (Tecnalia)*
Editor:	*Jacek Dominiak (BAW)*
Reviewers:	*Modris Greitans (EDI), Ugis Grinbergs (BOSC)*
Identifier:	*Deliverable # D5.3*
Nature:	*Report*
Date:	*17 Octobre 2019*
Status:	*v2.0*
Diss. level:	*Public*

Executive Summary

The primary scope of the deliverable is to accompaniment the first releases of the ENACT enablers. Therefore, it includes a set of user guides for each of the project enablers, showcases the way we expose the API's and reasons about the integration with non-enact tools.

The research leading to these results has received funding from the European Community's H2020 Programme under grant agreement n° 780351 (ENACT).

Members of the ENACT consortium:

SINTEF AS	Norway
BEAWRE DIGITAL SL	Spain
EVIDIAN SA	France
INDRA Sistemas SA	Spain
Fundacion Tecnalia Research & Innovation	Spain
TellU AS	Norway
Centre National de la Recherche Scientifique	France
Universitaet Duisburg-Essen	Germany
MONTIMAGE	France
Istituto per Servizi di Ricovero e Assistenza agli Anziani	Italy
Baltic Open Solution Center	Latvia
Elektronikas un Datorzinatnu Instituts	Latvia

Revision history

Date	Version	Author	Comments
08/08/2019	V0.1	Jacek Dominiak (BAW)	Table of contents
17/09/2019	V1.0	Jacek Dominiak (BAW)	First version release
14/10/2019	V2.0	Jacek Dominiak (BAW)	Comments of the reviewers addressed.

Contents

1 Contents

1. Introduction

1.1 Context and objectives

The objective of this deliverable is to support the release of first version of the ENACT enablers. All of the open-source enablers are released under Enact Gitlab repository group available under: https://gitlab.com/enact. The repository group is publicly available as of the time of this writing.
Based on the initial architecture design described in D5.1 and D5.2 this deliverable showcases how the tools could be used in a real environment scenario. This deliverable also showcases the effort of the project to connect with external tools and standards as well as the effort of ENACT enablers to be as easy to adopt into the DevOps work circle.

1.2 Structure of the document

The remainder of the document is structured as follows. After a brief introduction of the integration with non-enact tools in Section 2, we introduce the way all of the open source enablers of the project are exposing the API in Section 3. Sections 4 to 14 showcases the user guides of the tools with typical use case example. Section 15 concludes the document.

2. Integration with non-enact tools

ENACT Consortium recognizes the importance of widely adopted IoT and DevOps platforms or software solutions and their ability to ease user adoption, due to less traction of embedding the ENACT proposed tools in the existing workflows. Using commonly adopted tools to back up and integrate with ENACT enablers allows to reach the additional communities of users and help to exploit and carry the project outcomes beyond the project timeline. Being a project that focuses on DevOps or SecDevOps and its processes, we do want to integrate with the usual workflow of DevOps teams.

Our integration points are listed in a live document which can be found under https://www.enact-project.eu/Integration.htm. The list includes such current market standards as Docker, Jira, Jenkins and OpenAI.

3. Unified Enact API experience

Novel approaches are often underused because of poor usability. Usability is often overlooked by research projects, but it is a must for the customer facing market leading tools. From the very start of the project, we aimed at providing tools whose usability will not be lacking against what is/will be commercially available. For that reason, the Consortium agreed on the Unified way of producing enablers, from the look'n'feel to the way the data is exchanged.

One of the points which does help the potential users of the ENACT enablers, is the ability to see how the tools respond to different inputs and what kind of API integration points are available. As per D5.1, we have set up to use Swagger, which is one of the two leading standards for API exposure. As of the delivery of this document, all of the open-source enablers of the project, do expose Swagger UI for the users. In connection with the ability of using Docker for pulling the example of the enablers, it allows for unbeatable experience to test and explore the integration points of each enabler. It is paired with an online testing tools where the user is free to express the data models needed for the API point and also execute it, to see the system response. In Figure 1 below we show the example of Swagger API from the Risk Management enabler.

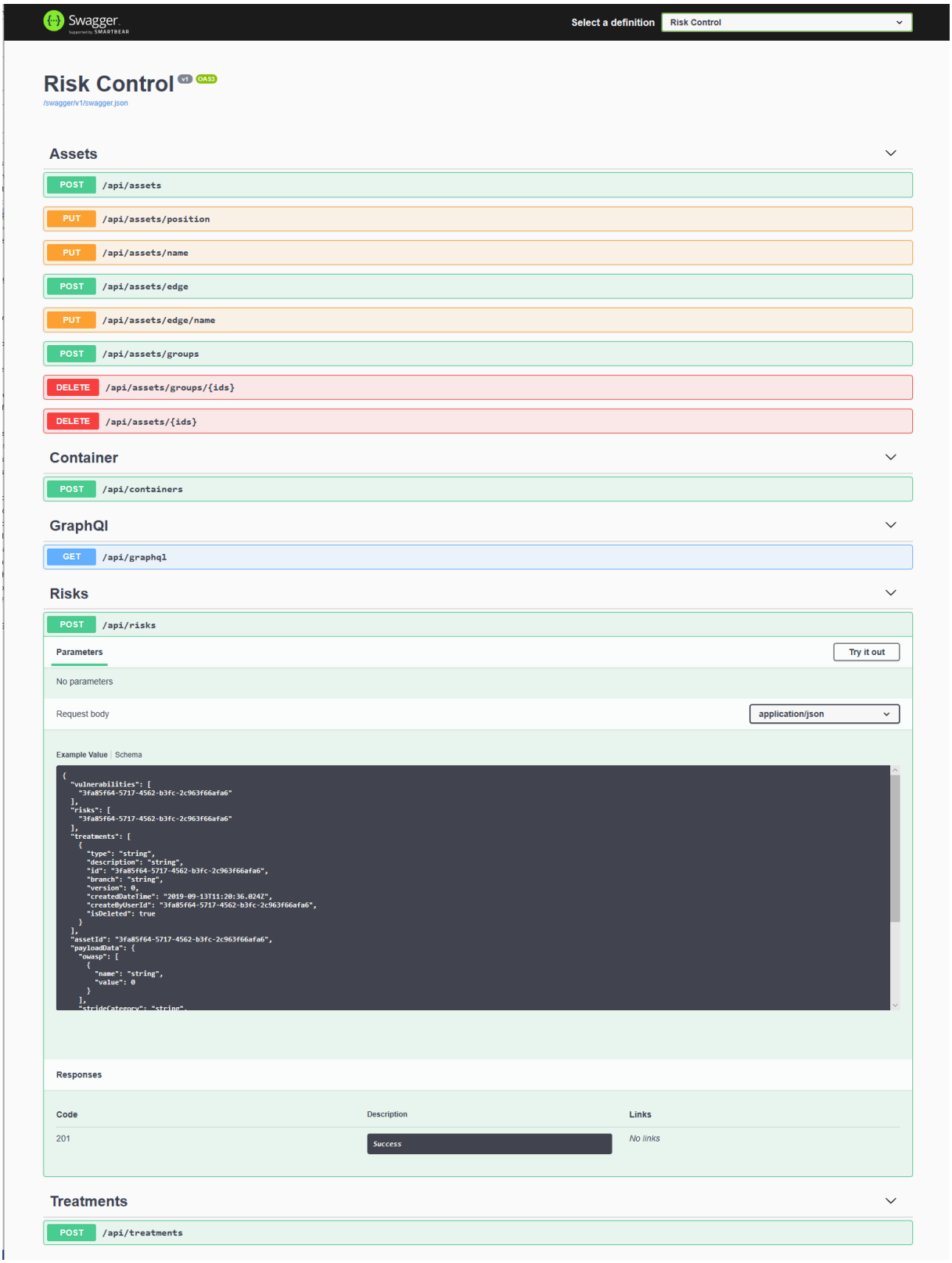


Figure 1. Example of the Unified ENACT API Swagger dashboard

The exact reasoning, benefits and drawbacks of providing the Unified ENACT API standards was described in the deliverable D5.1, section 4.2.

In the next sections we will showcase the set of examples of user guides for all of the ENACT enablers. Please note that the specified user guide is just an example to illustrate the usual workflow of the tool and more tutorials and user guides can be found within the repositories of the tools under **doc/** subfolder.

4. Risk Management User Guide

The Risk Management tool provides concepts and tools for the agile context-aware and risk-driven decision support and mechanisms for application developers and operators to support the continuous delivery of trustworthy smart IoT systems. The approach is an evolution of the MUSA Risks management tool that is focused on security for cloud-based systems. The extension comes with the ability to consume any types of risks, defined within a catalogue or by the user. It allows the creation of non-functional risks and evaluations of such risks. The Risk Management tool integrates with the DevOps cycle in order to continuously monitor the risk mitigation status though evidence collectors.

The tool allows for a number of possibilities of the risk control on the technical and non-technical level. Detailed scenario of the risk management process along with examples can be found on the ENACT Project Youtube channel with title “ENACT Risk Management – User guide – initial” or under direct link.

5. Online Learning User Guide

The Online learning enabler offers continuous improvement/reparameterization of a self-adaptive software system.
The improvement is done by adjusting a parameter provided by the self-adaptive software system enabling to trade off two contrasting quality requirements against each other. Based on a reward-function the enabler is able to compute rewards for each parameter-setting and find the optimal parameter-setting.
Core of the enabler is a policy-based Reinforcement Learning method updating a policy-network by means of Proximal Policy Optimization (PPO). A graphical user interface offers means to configure the enabler according to the self-adaptive software system that needs to be enriched by online-learning. Additionally, the GUI offers a dashboard to get insight into the internal behaviour of the underlying algorithm and the adaptation of the trade-off-parameter of the system to adapt.

5.1 Setup

5.1.1 Pre-requisite

- Docker

5.1.2 Installation using git:

- Clone the Repository:

```
git clone https://gitlab.com/enact/online-learning-enabler.git
```

- Run **start-ol.sh**:

```
sudo ./start-ol.sh
```

    - This script will:
        - remove previous images of the online_learning_enabler (if existing)
        - build a new image
        - run the new image

5.2 Building and running the OLE

To build and run the OLE, execute start-ol.sh

```
sudo ./start-ol.sh
```

5.3 Accessing the bash of the docker container

To execute commands inside the docker container, execute the following command (olc is the name of the container):

```
docker exec -it olc bash
```

5.4 Starting the learning-process of the OLE

To start the Online_Learning_Enabler (OLE), there are two options:

Start OLE by executing start_ol.py (located at `/var/www/REST-API/rest-api/start_ol.py`)

- This has to be executed in the bash of the Docker container
- This option allows you to see all the output that OLE will print.

Start OLE by using the REST-API (have a look at this Swagger-UI tutorial and this GET/POST tutorial)

Make sure that the controlled system is available when you start the learning process, as the OLE will try to connect to the system. Also ensure that the OLE is fully configured when you start the learning process. For a tutorial about how to configure the tool via SwaggerUI or GET/POST-commands have a look at the repository.

5.5 Meaning of the configuration-parameters

For an example configuration have a look at the example folder.

- timesteps
  - number of steps (actions) the ole will take before termination
- amount_of_data
  - amount of data the ole receives from the controlled system
- state_size
  - amount of data which ole has to learn with
- selected_socket_data_indexes
  - Array with integers to select the corresponding data out of the received data
  - sets the structure of the state which ole has to lern with
  - size of this Array should be equal to state_size
- ip
  - ip-address of the controlled system
- port
  - port of the controlled system
- if_statements
  - used to define a custom value based on conditions
  - values can be later be used inside the reward_function
- reward_function
  - defines the function to calculate the reward

5.6 Using placeholders inside if_statements and reward_function

For a better understanding we will use the following config:

```
{
  "timesteps": "50000",
```

```
  "amount_of_data": "5",
  "state_size": "1",

 "selected_socket_data_indexes": [
    0
  ],
  "ip": "172.17.0.3",
  "port": "11211",
  "if_statements": "if(#value_2#>0.03){0}else{1}",
  "reward_function": "#value_5# * 1 * #if_statement_1# * 1"
}
```

Inside this config, `#value_2#`, `#value_5#` and `#if_statement_1#` are used as placeholders. Placeholders consists of a name and a number and are wrapped in `#`. The number counts from one upwards. The basic structure looks like this:

```
#placeholdername_number#
```

The placeholdername is not customizable. In the current state of OLE, only `value` and `if_statement` are supported placeholdernames.

- The placeholder `#value_2#` represents the second entry in the received data
- The placeholder `#value_5#` represents the firth entry in the received data
- The placeholder `#if_statement_1#` represents the value of the first if_statement

5.7 Save-Location of the configurations

The configurations are stored in different files due to concurrency issues. In the Docker container the files are stored at `/var/www/REST-API/restapi/configs/`. To set the default configuration, the files can be changed at `online_learning_enabler/www/REST-API/restapi/ole_config` (before building the Docker image). Also have a look at the Directory-Overview.

6. GeneSIS User Guide

GeneSIS aims to facilitate the engineering and continuous deployment of smart IoT systems, allowing decentralized processing across heterogeneous IoT, edge and cloud space. GeneSIS includes: (i) a domain specific modelling language to model the orchestration and deployment of smart IoT systems across the IoT, edge and cloud spaces; and (ii) an execution engine that will support the orchestration of IoT, edge and cloud services as well as their automatic deployment over IoT, edge and cloud resources.

The following Sections detail how GeneSIS can be started and used on a Node-Red container via Docker on the machine running GeneSIS. More advanced tutorials can be found here: https://gitlab.com/enact/GeneSIS/tree/master/docs/tutorial

In the rest of the Section it is assumed that (i) a Docker engine is running on the machine running GeneSIS and the Docker Remote engine is accessible, and (ii) GeneSIS is properly installed on the machine.

Instructions on how to make the Docker remote engine accessible can be found here.

6.1 Start GeneSIS:

First, start GeneSIS by using the following command in the src folder of GeneSIS:

```
npm start
```

The following message should appear:

```
> GeneSIS@0.2.0 start /Users/ferrynico/Documents/Code/GeneSIS-
gitlab/GeneSIS
> concurrently "nodemon ./app.js" "webpack-dev-server "

[0] [nodemon] 1.19.0
[0] [nodemon] to restart at any time, enter `rs`
[0] [nodemon] watching: *.*
[0] [nodemon] starting `node ./app.js`
[0] 2019-09-19T09:13:13.469Z - [info]: Engine started!
[0] 2019-09-19T09:13:13.567Z - [info]: PlantUML diagram generator started
on port: 8081
[0] 2019-09-19T09:13:13.584Z - [info]: MQTT websocket server listening on
:9001
[0] 2019-09-19T09:13:13.585Z - [info]: GeneSIS Engine API started on 8000
[1] ℹ ｢wds｣: Project is running at http://0.0.0.0:8880/
[1] ℹ ｢wds｣: webpack output is served from http://0.0.0.0:8880/dist/
[1] ℹ ｢wds｣: Content not from webpack is served from
/Users/ferrynico/Documents/Code/GeneSIS-gitlab/GeneSIS/public/
```

GeneSIS is started once the following line 'Compilation completed' is shown. The tool is then accessible at the following address: `http://127.0.0.1:8880`

6.2 Specifying the deployment model

At this stage the GeneSIS editor can be used to specify a deployment model. Usually this process starts first by defining the high-level architecture of the deployment before digging into the configuration of the different components that form the deployment model.

Since GeneSIS Modelling language is inspired by component-based approaches. Deployment models can thus be regarded as assemblies of components. In this example, the deployment model will be composed of two components:

- a *SoftwareComponent* (i.e., the Node-RED container that will be deployed by GeneSIS) and more precisely an *InternalComponent* as its deployment lifecycle will be managed by GeneSIS.
- an *InfrastructureComponent* (i.e., the host on top of which we will deploy the *SoftwareComponent*, in this case a Docker Engine).

First, start by creating the *InternalComponent* by clicking on `'Edit > Software Component > Internal Component > Node-RED'` as depicted in Figure 2.

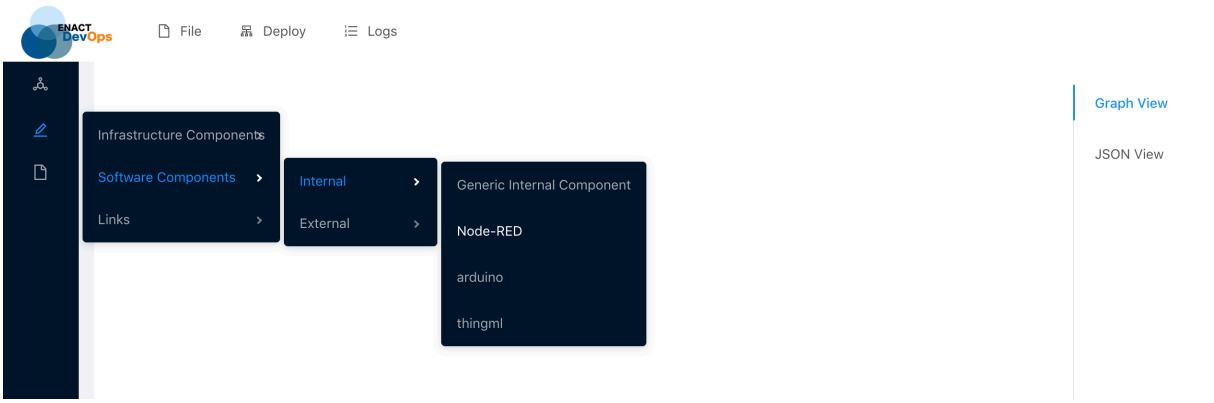


Figure 2. GeneSIS, creating an Internal Component

As a first step specify its 'name' and click on the `'OK'` button to actually add the component to the deployment model. A circle should appear in the editor. Please note that it is possible to (i) zoom in/out by using the mouse wheel, (ii) move a component by drag and drop, and (iii) edit the properties of a component with a right click on it.

Create the *InfrastructureComponent* by clicking on `'Edit > InfrastructureComponent > Docker Host'.` Specify its 'name' and click on 'OK'.

To specify that the *InternalComponent* will be deployed on the *InfrastructureComponent* (i.e., Node-RED on Docker) it is required (i) to specify the execution ports of our components and (ii) to create a containment relationship between the two components. To specify the provided execution port of the docker host (i.e., my_machine). Right-click on the Docker Host and change the name of the *'provided execution port'* property (e.g., offerDocker). Similarly, right-click on the Software component (i.e., nodered) and change the name of the required execution port (e.g., demandDocker).

Then the containment relationship can be added by clicking on `'Edit > Link > Add Containment'`. Select the proper nodes and click on 'add'. The circle should now be contained by the rectangle as depicted in Figure 3.

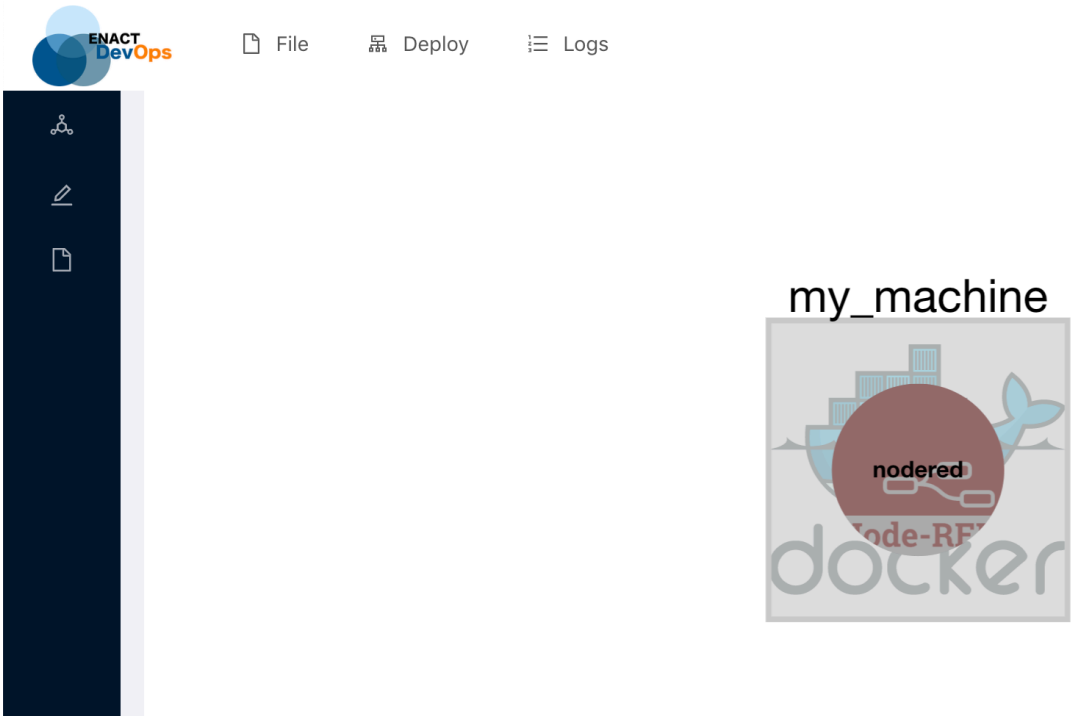


Figure 3. GeneSIS, creation of a containment

The next step consists in configuring the two components. For the *InfrastructureComponent* (aka., Docker Host), to make sure that (i) the *'IP'* property of the component is set to *'127.0.0.1'* and (ii) that the 'port' property is set to *"['2376']"*. This property of the component can be edited by right-clicking on it and modifying the 'IP' field as depicted in Figure 4.

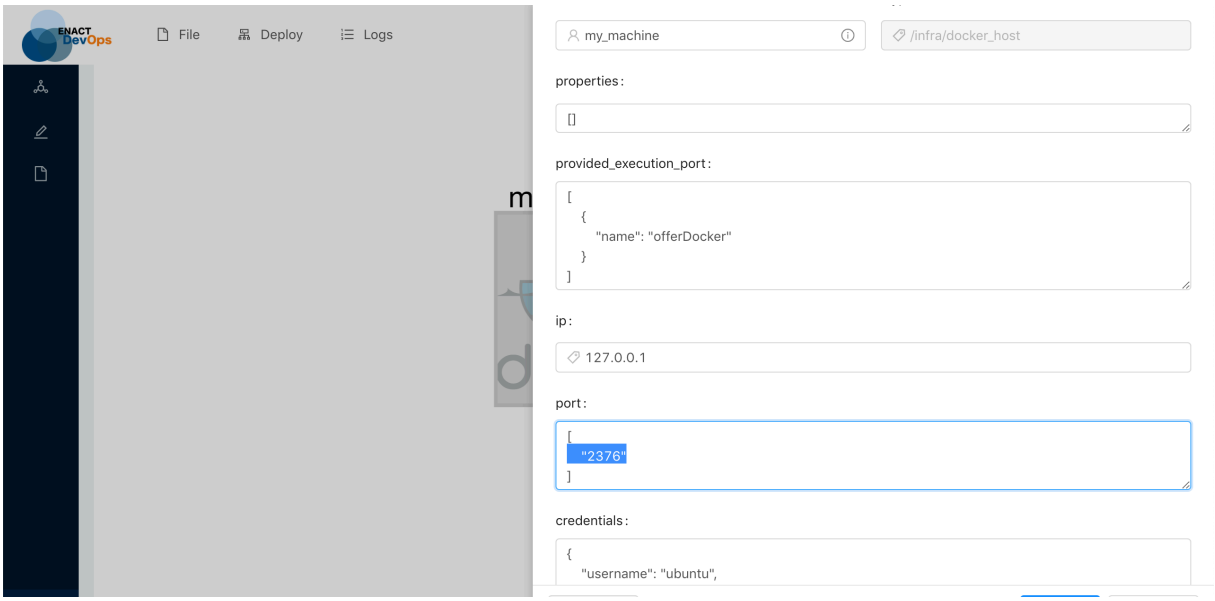


Figure 4. GeneSIS, component configuration.

For editing the *InternalComponent* (aka., Node-RED), the GeneSIS JSON editor can be used. The JSON editor view can be opened by clicking on the top right button named `'JSONEditor'`. To specify the Docker port binding (i.e., how the port of the service running in the Docker container will be accessible from outside), edit the JSON as depicted in the figure below. Typically, Node-RED is exposed using port 1880.

Figure 5. GeneSIS, component configuration with the JSON editor.

6.3 Deploy

In order to trigger a deployment, click on: `'Deploy > All'`.

Once the deployment has started, deployment logs are available in the console of the GeneSIS tool. The deployment process will consist in (i) checking if the Docker engine is reachable (if so the rectangle will turn green), (ii) pulling the Node-RED docker image (whilst this operation is being performed the component will be yellow), and (iii) start the Docker container (At this point the circle should also be green as depicted in Figure 6).

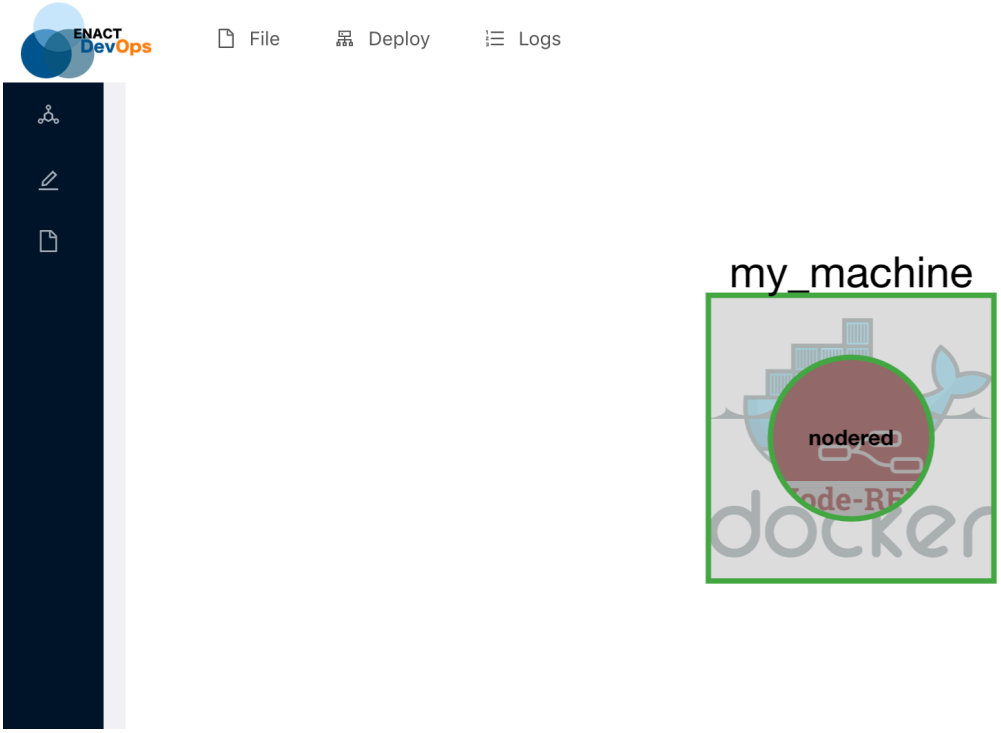


Figure 6. GeneSIS, deployment status.

Finally, the Node-RED can be accessed by right-clicking on the component in the graph view of the editor and by clicking on the "`Go To`" button.

6.4 Deploy on a remote Host

So far, Node-RED has been deployed on the machine running GeneSIS. But actually, the same process can be used to deploy Node-RED on a remote host. To do so, it is only required to change the IP parameter of the *InfrastructureComponent* (aka., the host) to the IP address of the host where Node-RED should be deployed.

6.5 Deploy Node-RED and initialize it with a flow

When deploying an instance of the Node-RED runtime the property `nr-flow` can be used to specify that a Node-RED flow should be deployed once the Node-RED runtime is up and running. In this tutorial the following flow can be used:

https://gitlab.com/enact/GeneSIS/blob/master/docs/examples/a_node-red_flow.json.

This flow leverages a `worldmap` node that needs to be installed in the Node-RED runtime before the flow can be executed. In GeneSIS, this can be specified via the `packages` property. To install the `worldmap` node, this property should be assigned with the following value:

```
["node-red-contrib-web-worldmap"]
```

Figure 7 below depicts these two properties.

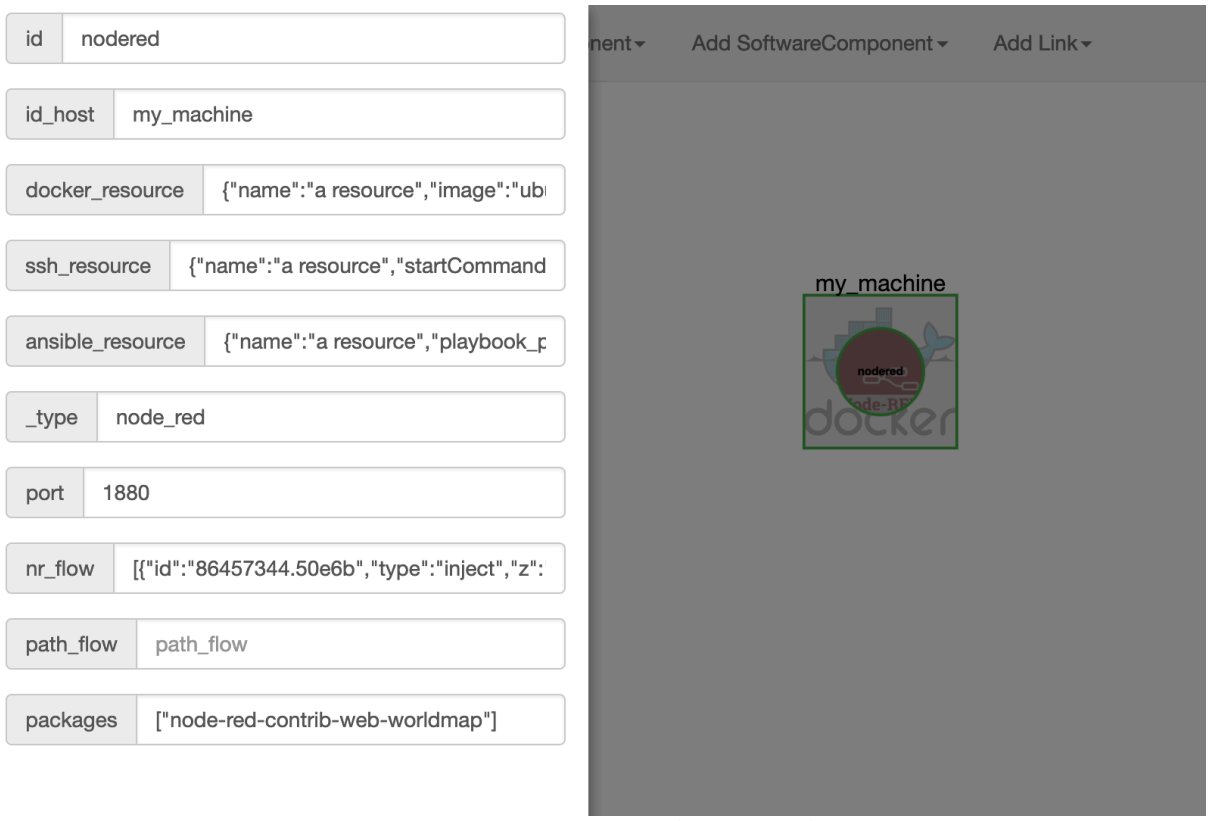


Figure 7. GeneSIS, deployment of a Node-Red flow.

After deployment, the following flow in shown in the Node-RED editor.

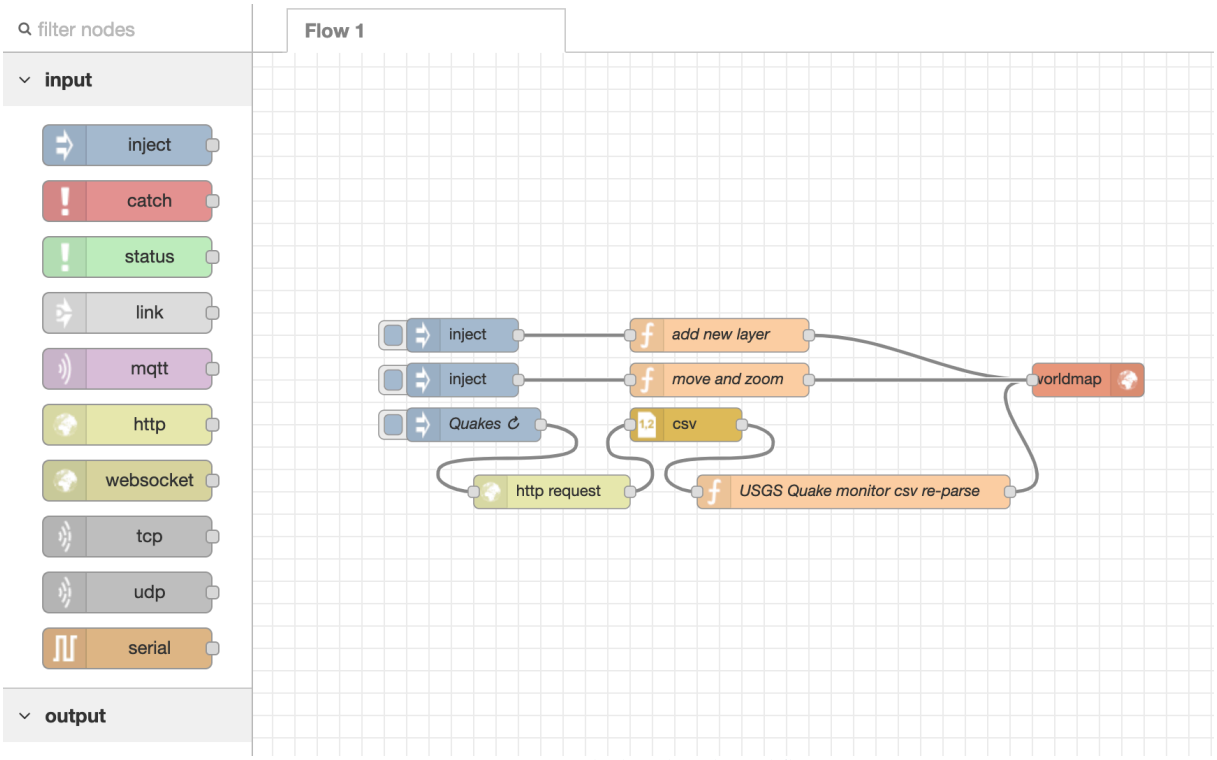


Figure 8. GeneSIS, deployed Node-Red flow.

7. Actuation Conflict Manager User Guide

In this example we will use the ENACT Actuation Conflict Manager (ACM) in order to identify and solve actuation conflicts in a GeneSIS deployment model. The GeneSIS deployment model specifies how to deploy two instances of Node-RED, each of them executing a simple flow interacting with a single actuator.

7.1 Prerequisites

- A GeneSIS instance is able to deploy Docker containers and Node-RED
  - A recommended read would be the GeneSIS tutorials, especially Node-RED localhost and Two Node-REDs on localhost.
- Running ACM instance and instructions on getting ACM up can be found here.

7.2 Building the GeneSIS deployment model

- Open the GeneSIS interface in the browser: http://localhost:8880/
- Add a Docker Engine (InfrastructureComponent) and configure it as such
  - name: `docker_local`
  - id: `docker_local`
  - ip: IP address of Docker Engine
  - port: `["2376"]`
- Add two Node-RED components (SoftwareComponent/Internal)
  - Configure the first as such:
    - name: `node_red_local_0`
    - id: `node_red_local_0`
    - nr_flow:
      - contents of nr_flow_0.json

```
[{
    "id": "534e8391.16269c",
    "type": "inject",
    "z": "68bf7da6.0a29f4",
    "name": "Application 1",
    "topic": "",
    "payload": "true",
    "payloadType": "bool",
    "repeat": "",
    "crontab": "",
    "once": false,
    "onceDelay": 0.1,
    "x": 550,
    "y": 220,
    "wires": [["9b11eba.111b618"]]
}, {
    "id": "9b11eba.111b618",
    "type": "function",
    "z": "68bf7da6.0a29f4",
    "name": "Application 1",
    "func": "\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "x": 820,
```

```
            "y": 220,
            "wires": [["68b39a65.2fa0c4"]]
        }, {
            "id": "68b39a65.2fa0c4",
            "type": "debug",
            "z": "68bf7da6.0a29f4",
            "name": "Actuator 1",
            "active": true,
            "tosidebar": true,
            "console": false,
            "tostatus": false,
            "complete": "payload",
            "targetType": "msg",
            "x": 1090,
            "y": 220,
            "wires": []
        }]
```


- And the second:
  - name: `node_red_local_1`
  - id: `node_red_local_1`
  - docker_resource:


```
{"name":"a
resource","image":"nicolasferry/multiarch-node-red-
thingml:latest","command":"","port_bindings":{"1881"
:"1880"},"devices":{"PathOnHost":"","PathInContainer
":"","CgroupPermissions":"rwm"}}
```


  - nr_flow:
    - contents of nr_flow_1.json


```
"id": "9e1ac4.38bee54",
"type": "inject",
"z": "68bf7da6.0a29f4",
"name": "Application 2",
"topic": "",
"payload": "true",
"payloadType": "bool",
"repeat": "",
"crontab": "",
"once": false,
"onceDelay": 0.1,
"x": 470,
"y": 220,
"wires": [["ede3df0b.aef39"]]
}, {
    "id": "ede3df0b.aef39",
    "type": "function",
    "z": "68bf7da6.0a29f4",
    "name": "Application 2",
    "func": "\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "x": 750,
```

```
        "y": 220,
        "wires": [["845bded2.c87f5"]]
    }, {
        "id": "845bded2.c87f5",
        "type": "debug",
        "z": "68bf7da6.0a29f4",
        "name": "Actuator 2",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "payload",
        "targetType": "msg",
        "x": 1010,
        "y": 220,
        "wires": []
    }]
```


- Add Containment links between node red instances and the docker container
- The resulting deployment model should look like this in the GeneSIS editor

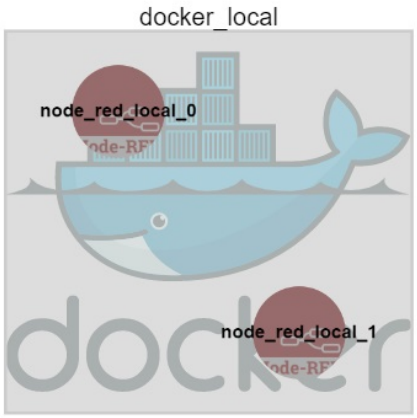


Figure 9: GeneSIS deployment for ACM sample

- Press Deploy/All and wait for GeneSIS to finish deploying all the Node-REDs
  - A first Node-RED instance is available at http://<docker_ip>:1880
    - This instance contains Application 1 flow
  - A second Node-RED instance is available at http://<docker_ip>:1881
    - This instance contains Application 2 flow
- Finally, export this deployment model using the File menu
  - For reference, *here is the deployment model* that will be used for the rest of the tutorial. It should be identical to the one just created

7.3 Load the flow in ACM for analysis

- Open the ACM interface in the browser: http://localhost:3333/
- Click `"Load model"` in the menu at the top.
- Input the GeneSIS server URL, set model type to GeneSIS
- After loading the GeneSIS model into ACM, click on `Environment model > Load environment model`
- In the dialog, input the environment model data from this file:

  - The environment model is composed of two collections, loosely resembling extra Node-RED nodes
    - `physical_processes`: contains a list of defined physical processes
    - `links`: contains a list of links, linking an actuator node (through its Node-RED id) to a physical process

```
{
  "physical_processes": [
    {
      "id": "9b11eba.0a29f4",
      "name": "physicalProcess0",
      "x": 200,
      "y": 200
    }
  ],
  "links": [
    {
      "from_id": "845bded2.c87f5",
      "to_id": "9b11eba.0a29f4"
    },
    {
      "from_id": "68b39a65.2fa0c4",
      "to_id": "9b11eba.0a29f4"
    }
  ]
}
```

- Press `Done`
- Then click on `Find conflicts` to let ACM identify the issues
- ACM shows a view of both flows, inside boxes representing the two Node-REDs. ACMs are created and put in their own box representing the two nodes in each flow are linked.

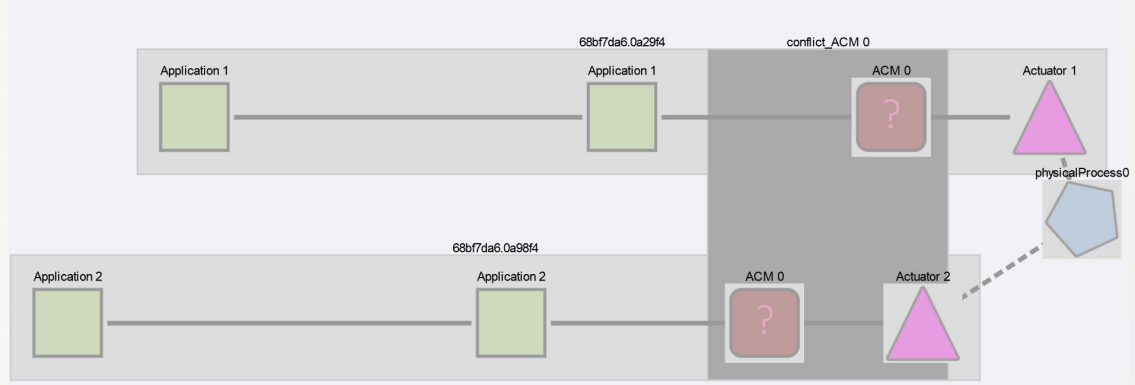


Figure 10: Environment model added to application model and conflicts identification

- Two extra ACM nodes appear in front of both actuators
  - The nodes communicate with each other and act as a single ACM
  - The reason multiple nodes are created is to support indirect actuation conflicts across several Node-RED flows

7.4 Solving the conflict

- In the main interface, click on one of the ACM nodes in red.

- A dialog opens with the available strategies from the palette

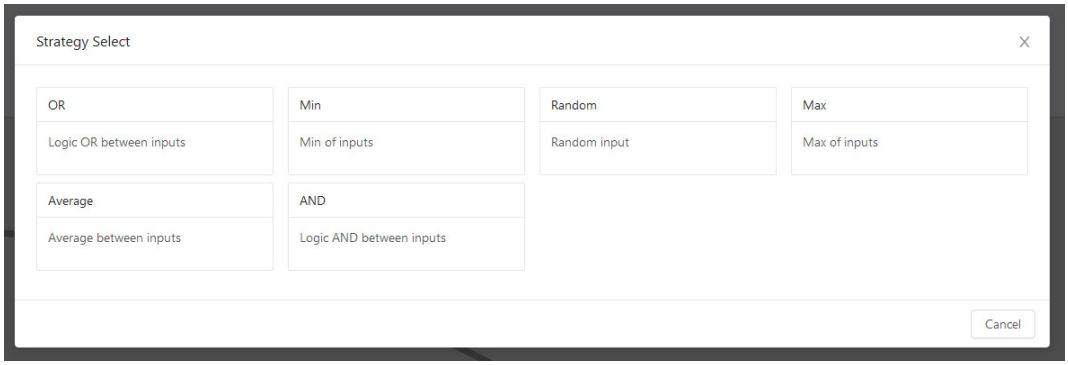


Figure 11: Off the shelf ACM nodes selection

- Select the desired one
  - Currently, the ECA rule engine is not implemented but the node can be configured
- If the node has configuration, like the ECA rules node, an extra dialog appears allowing input of extra configuration data
- When the configuration, if any, is completed both linked ACM's icon change to signal they are configured and ready to be deployed

8. Diversifier User Guide

- Create an instance of Device Provisioning Service (DPS) and link it with an existing IoT Hub as described here
- Create and save an individual enrolment with the following parameters:
  - Mechanism: Symmetric Key
  - Auto-generate keys: Yes
  - Registration ID: {registration_id}
  - IoT Hub Device ID (optional): {device_id}
  - Select how devices will be assigned to hubs: Static configuration
  - Select the IoT hubs this device can be assigned to: {an_existing_iot_hub}
  - In the Initial Device Twin State it is possible to include target conditions using tags, so that the provisioned device gets immediately affected by a deployment. For example:
  - ```
    {
      "tags": {
       "environment": "test"
      },
      "properties": {
       "desired": {}
      }
    }
    ```

- Keep note of device registration ID `registration_id`, primary key `symmetric_key`, and DPS ID Scope `scope_id`, which will be used below.
- SSH into the edge device and open `config.yaml`: `sudo nano /etc/iotedge/config.yaml`
- Modify and save `config.yaml` as follows using previously noted values (pay attention to spaces in yaml):
    ```

```
# provisioning:
#  source: "manual"
#  device_connection_string: "{connection_string}"

# DPS symmetric key provisioning configuration
provisioning:
 source: "dps"
 global_endpoint: "https://global.azure-devices-provisioning.net"
 scope_id: "{scope_id}"
 attestation:
  method: "symmetric_key"
  registration_id: "{registration_id}"
  symmetric_key: "{symmetric_key}"
```

- Restart the IoT Edge service: `systemctl restart iotedge`
- After some time, your device will be registered in your IoT Hub (using either `registration_id` or `device_id`).
- If specified tags match one of the deployments target conditions, the device will also be assigned with that deployment.

9. Test & Simulation User Guide

Test and simulation tool provides concepts and tools for running application scenarios against the set of programmed circumstances. The toolset is aiming to provide a baseline for performance, resilience testing as well as risk management testing. It does replicate the behavior of previously observed devices and is able to play back the sensors data against the programmed scenarios.

Due to one of the partners decision to leave the project, this decision direct impact and the subsequent change of the partners delivering the tool, the tool release has been postponed and it is planned to reach its initial release in M25 of the project.

10. Context aware access control User Guide

The objective of the Context-aware Access Control (CAAC) is to provide mechanisms for controlling the security, privacy and trustworthiness behaviour of smart IoT systems. The Context-aware Access Control tool is a feature provided by Evidian Web Access Manager (WAM[1]). It is an evolution of the authentication and authorization mechanisms provided by WAM intended for the Internet of Things.

The Access Control tool leverages on the OAuth 2.0 Device Flow protocol. The only requirements to use this flow are that the device is connected to the Internet, and able to make outbound HTTPS requests, be able to display or otherwise communicate a URI and code sequence to the user, and that the user (device owner) has a secondary device (e.g., personal computer or smartphone) from which to process the request. There is no requirement for two-way communication between the OAuth client (i.e. the connected device) and the end user's user-agent, enabling a broad range of use-cases.

10.1 Authorization Request synopsis

The synopsis of the requests to be implemented by a device to take benefit of the Context-aware Access Control tool is described as follows:

[1] EVIDIAN Proprietary software.

1. Enrolment phase
    a. Request a device code.
    b. Receive the device code.
    c. Transmit the code to the device owner, so he will be able to give his consent to the data scope the device asks to access.
    d. Poll the Access Control tool to get the access token.
    e. Receive the access token once the device owner has given his consent.
2. Access protected resource with token.

10.2 Authorization Request description

10.2.1 Request a device code

The connected object initiates the flow by requesting a set of verification codes from the Access Control tool by making an HTTP "POST" request to the Access Control tool's endpoint. The connected object constructs the request with the following parameters, encoded with the "application/x-www-form-urlencoded" content type:

```
client_id
      REQUIRED. The object identifier.

scope
      OPTIONAL. The scope of the access request.
```

Warning: Additional data will be required in later versions. In particular, the serial number and/or technical information of the object, will be required. This information will be used for filtering and searching for objects in the authorized object management user interface.

Example:

```
curl -i -H 'Content-Type: application/x-www-form-urlencoded' \
-X POST 'http://stephane.test-pxp.frec.bull.fr:8081/form/oidc/devicecode' \
--data 'client_id=3a84c51e-0e53-499d-bd70-ce3688981749' \
--data 'scope=email+openid'
```

10.2.2 Receive the device code

In response, the Access Control tool generates a device verification code and an end-user code that are valid for a limited time and includes them in the HTTP response body using the "application/json" format with a 200 (OK) status code. The response contains the following parameters:

```
device_code
      REQUIRED. The device verification code.

user_code
      REQUIRED. The end-user verification code.

verification_uri
      REQUIRED. The end-user verification URI on the Access Control tool.

verification_uri_complete
      OPTIONAL. A verification URI that includes the "user_code" (or other
      information with the same function as the "user_code"), designed for
      non-textual transmission.

expires_in
      OPTIONAL. The lifetime in seconds of the "device_code" and "user_code".
```

```
interval
      OPTIONAL. The minimum amount of time in seconds that the object SHOULD
      wait between polling requests to the Access Control tool.
```

Example:

```
{
 "user_code": "UCAAUJ",
 "device_code": "28b8e2ad-985f-401f-aa05-13542edfb37e",
 "verification_uri": "http://stephane.test-pxp.frec.bull.fr:8081/form/oidc/device"
 "expires_in": 1800
}
```

10.2.3 Request the device access token

After displaying instructions to the user, the connected object makes an Access Token Request to the Access Control tool with a "grant_type" of "urn:ietf:params:oauth:grant-type:device_code", with the following parameters:

```
grant_type
      REQUIRED. Value MUST be set to
      "urn:ietf:params:oauth:granttype:device_code".

device_code
      REQUIRED. The device verification code, "device_code" from the Device
      Authorization Response, defined in section 10.2.2.

client_id
      REQUIRED. The object identifier.

client_secret
      REQUIRED. The object secret.
```

Example:

```
curl -i -H 'Content-Type: application/x-www-form-urlencoded' \
      -X POST 'http://stephane.test-pxp.frec.bull.fr:8081/form/oidc/token' \
      --data 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Adevice_code' \
      --data "device_code=28b8e2ad-985f-401f-aa05-13542edfb37e" \
      --data 'client_id=3a84c51e-0e53-499d-bd70-ce3688981749' \
      --data         'client_secret=FCUgy08dvYAzidLLSE2Or0l1RWC6dqndADKXPG8h5EqNue-
EFq3NUYnF6SF4N_qaSzxoJHIiY3x1_TrJTQJNww'
```

The response to this request is defined in section 10.2.4. It is expected for the client to try the Access Token Request repeatedly in a polling fashion, based on the error code in the response. The interval at which the object polls MUST NOT be more frequent than the "interval" parameter returned in the Device Code Response (see section 10.2.2). If no interval was provided, the client MUST use a reasonable default polling interval.

10.2.4 Receive the device access token

If the user (device owner) has approved the grant, the Access Control tool responds with a success response; otherwise it responds with an error. The response is a JSON containing the *Access Token.*

Example:

```
{
    "access_token":
"eyJraWQiOiJyc2EyNTYiLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJld29pYzNWaUlpQTZJQ0p6YldsMGFFQ
kNkV2xzZEMxcGJpQlZjMlZ5SjNNZ1JHbHlaV04wYjNKNUlpd0tJbTVoYldVaUlEb2dJa3B2YUc0Z1UyMXBk
R2dpTEFvaVoybDJaVzVmYm1GdFpTSWdPaUFpU205b2JpSXNDaUptWVcxcGJIbGZibUZ0WlNJZ09pQWlVMjF
wZEdnaUxBb2ljSEpsWm1WeWNtVmtYM1Z6WlhKdVlXMWxJaUE2SUNKemJXbDBhQ0lzQ2lKblpXNWtaWElpSU
RvZ0ltMWhiR1VpTEFvaWFYTmZiMmxrWTE5aFpHMXBibWx6ZEhKaGRHOXlJaUE2SUdaaGJITmxDbjA9IiwiY
XpwIjoiMjZkZDZjYzQtOWE2MC00OGM1LThmNmMtZTQ2MGExNjExNjFiIiwiaXNzIjoiaHR0cDpcL1wvc3Rl
```

```
cGhhbmUudGVzdC1weHAuZnJlYy5idWxsLmZyOjgwODFcL29pZGMtcHJvdmlkZXJcLyIsImV4cCI6MTU1MzA
4MTI4OCwiaWF0IjoxNTUzMDc3Njg4LCJqdGkiOiJlMDg3NmYwNC04NDVhLTQyNTUtYjJiMC03YjY5ZDZhNz
gxNzkifQ.9DAG2R6z__LY9t130EZXRgUeXmJo0Te554R37dywd3xkcL1XiWZnLj7h_LQTzIhvWGoe9nl3Dd
XhyD8qtDBgIus5uy8NJ5AkSmVQCcXwngwdTiNeVZ_UWqpAlyi93nuzMNPBtwYOC_D6mju_ZHZHT2Qc8Wsww
SGCcFaFHK5o-
LbWm5h3y_4fMlhZqKYlwnZrNsAudJ1E_piyWS482V29UM3btvKIYCTC5ETi4evfouP1unTrd97gvUXeqo-
MUyDPyCVMN1F8TLIyWF73VQ7I3JuV2bhhXfSOXKRwhk2NZXsWM5gzjOxA2FNEEI76QuD99K8ZIIBZSIUlgN
z0olNtRQ",
    "token_type": "Bearer",
    "expires_in": 3599,
    "scope": "openid email",
    "id_token":
"eyJraWQiOiJyc2EyNTYiLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJzbWl0aEBCdWlsdC1pbiBVc2VyJ3MgR
GlyZWN0b3J5IiwiYXVkIjoiMjZkZDZjYzQtOWE2MC00OGM1LThmNmMtZTQ2MGExNjExNjFiIiwia2lkIjoi
cnNhMjU2IiwiaXNzIjoiaHR0cDpcL1wvc3RlcGhhbmUudGVzdC1weHAuZnJlYy5idWxsLmZyOjgwODFcL2Z
vcm1cL29pZGNcLyIsImV4cCI6MTU1MzA3ODI4OCwiaWF0IjoxNTUzMDc3Njg4LCJqdGkiOiIwYWZhNzkxZC
03NGVhLTQ0YzQtODhjZC0wYjIxYzUzNjhlZmEifQ.2OHuTjnIaqOpByv0RXf655zWd-
zhsuYf6lhojZo2NDlWwwwp02sUVP6fMssM3DXuI67ljR3qugmWeHaW7roNuaDikxwkI3hoWkbogxvEg1gtc
lzlyDYGmIWlBmw_LtQaYRXB5Ht1-3V1841iSRhnmuRscP2gZKEdvWF2Mqzq1vwVy3JyDYArnHPu1WtVQ-
tgFegEF0LslLT_CFo2u8b8uuxv_NXmp-R2RjwQDpyV2cm1sYMaFI5eLe7QG-
KtcU36e0nEsgtrKwIpKj1xiK0i57YzODD6ZUU8hWuYuKXNQVWQUzT3McMbh6nONlVkI0OQgG7vERYR1dGct
7P-ErGNWw"
}
```


10.2.5 Access protected resource with token

In the example beow, the device sends temperature, humidity and light information to the back-end server.

```
String data = (String)"temperature="
                + temperature + "&humidity=" + humidity + "&light=" + light;
  WIFI_CLIENT.println("POST /add HTTP/1.1");
  WIFI_CLIENT.println("Host: stephane.test-pxp.frec.bull.fr:10002");
  WIFI_CLIENT.println("Authorization: Bearer " + OAUTH_ACCESS_TOKEN);
  WIFI_CLIENT.println("Content-Type: application/x-www-form-urlencoded");
  WIFI_CLIENT.println("Content-Length: " + String(data.length()));
  WIFI_CLIENT.println("");
  WIFI_CLIENT.println(data);

  delay(30000);
```

In this example, OAUTH_ACCESS_TOKEN is the access token obtained during the enrolment phase (cf. section 10.2.4).

10.3 Context-awareness capacities

The access authorizations provided by the Access Control tool can be adapted according to contextual information. *Context* may be, for instance, the date and time an access authorization is requested, or the geolocation of this request; it may be also composed of a set of information about the status of the underlying infrastructure, the physical system status, Security Information and Event Management (SIEM) alerts, for example to make certain information more widely available in the case when an alarm has been triggered.

The Access Control Tool is composed of an Authorization server associated to a Post authorization plugin. This Post authorization plugin is aimed to add dynamic controls during the authorization phase. The Post authorization plugin extends the basic authorization phase and is entirely customizable. Any operation can be executed during the authorization phase, including calling external programs.

In particular, the Post authorization plugin communicates with a Context Server. The Context Server exposes a REST API that provides contextual data on the user and his devices. These data are dynamic attributes and come from other external sources (sensors, other applications, etc.). In this way, each time

a device asks for a resource access, the Access Control Tool can get the dynamic context information of the user associated to the device that performs the request.

From the provided contextual information, a Risk value is computed either statically, depending on a defined configuration, or dynamically by using a REST API to dialog with an external decision engine. This risk value is then used to make dynamic access controls based on the context information during the authorization phase. For example, if the access token is valid while the risk value does not respect the authorization policy, either an access request can be rejected, or its accessible scopes can be limited. The authorization policy is a set of rules that define whether a user or device must be permitted or denied accessing to a resource. An administrator can control this adjustment and create special authorization rules based on risk levels computed from the context data provided.

This risk value can be used also to apply context-aware dynamic scopes. The Post authorization plugin can create injection variables that can be reused and injected in the initial request sent to the backend server.

11. Context monitoring & Behavioural drift analysis User Guide

In this example, we will simply create and deploy a Behavioural Drift Computation model using GeneSIS.

11.1 Start Behavioural Drift Computation model editor:

First, let's start Behavioural Drift Computation (BDC) model editor by using the following command:

```
cd bda-app
npm start
```

The following message should be displayed:

```
> bda-app@0.1.0 start
D:\Dev\contrats\ENACT\behavioural_drift_analysis\bda-app
> concurrently "react-scripts start" "nodemon server/index.js"

[1] [nodemon] 1.19.1
[1] [nodemon] to restart at any time, enter `rs`
[1] [nodemon] watching: *.*
[1] [nodemon] starting `node server/index.js`
[1] BDA server listening on port 3003!
[0] [HPM] Proxy created: /bda-server  ->  http://localhost:3003/
[0] Starting the development server...
[0]
[0] Compiled successfully!
[0]
[0] You can now view bda-app in the browser.
[0]
[0]   Local:            http://localhost:3000/
[0]   On Your Network:  http://192.168.56.1:3000/
[0]
[0] Note that the development build is not optimized.
[0] To create a production build, use npm run build.
[0]
```

Once the BDC editor is up and running, it will open itself in your default browser at this address:

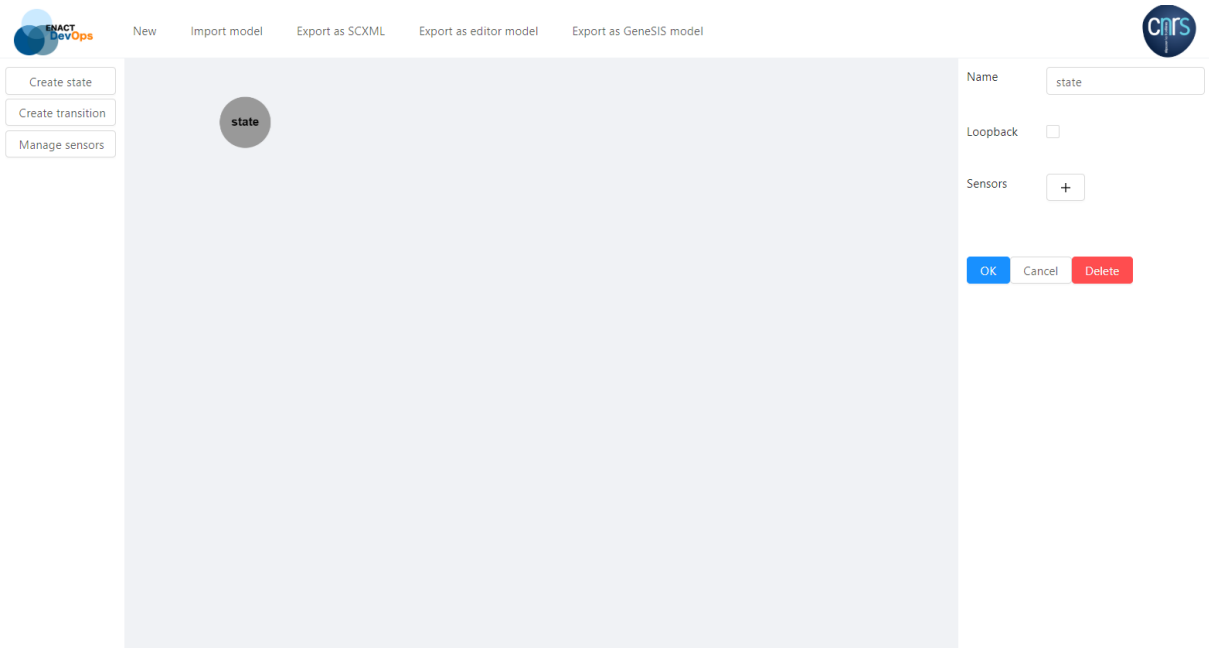
```
http://127.0.0.1:3000
```

11.2 Specifying the Behavioural Drift Computation model

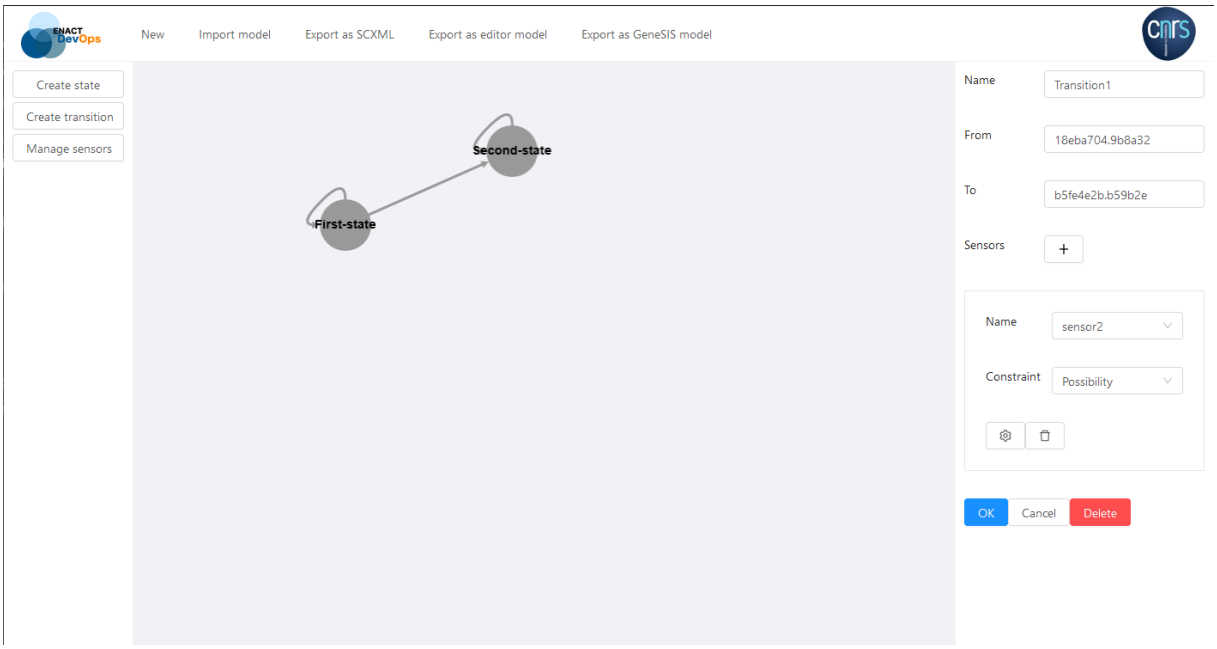
We can now start specifying our behavioural drift computation model. Doing so is like building a finite state machine with sensors information on transitions (inputs) and on states (outputs).
First, we start by defining the sensors involved in the behavioural drift computation model we are creating.

- Click on the 'Manage sensors' button to bring up the management dialog.
- From there select create new in the dropdown, press Next and give it a name.
    - Note: allowed characters are uppercase, lowercases, minus and underscore.

Create all sensors during this first step or create new ones when you need it. However, create a sensor using the management dialog before using it in a finite state machine's state or transition.
Then, specify the finite state machine by creating states and transitions.
Create a state by clicking on 'Create State' and clicking on the interface grey part to place it. To edit state properties, click on the state node. From the right side pane, you can change the state's name, choose if this state has a loopback transition and attach sensors to it by clicking on "+" button.



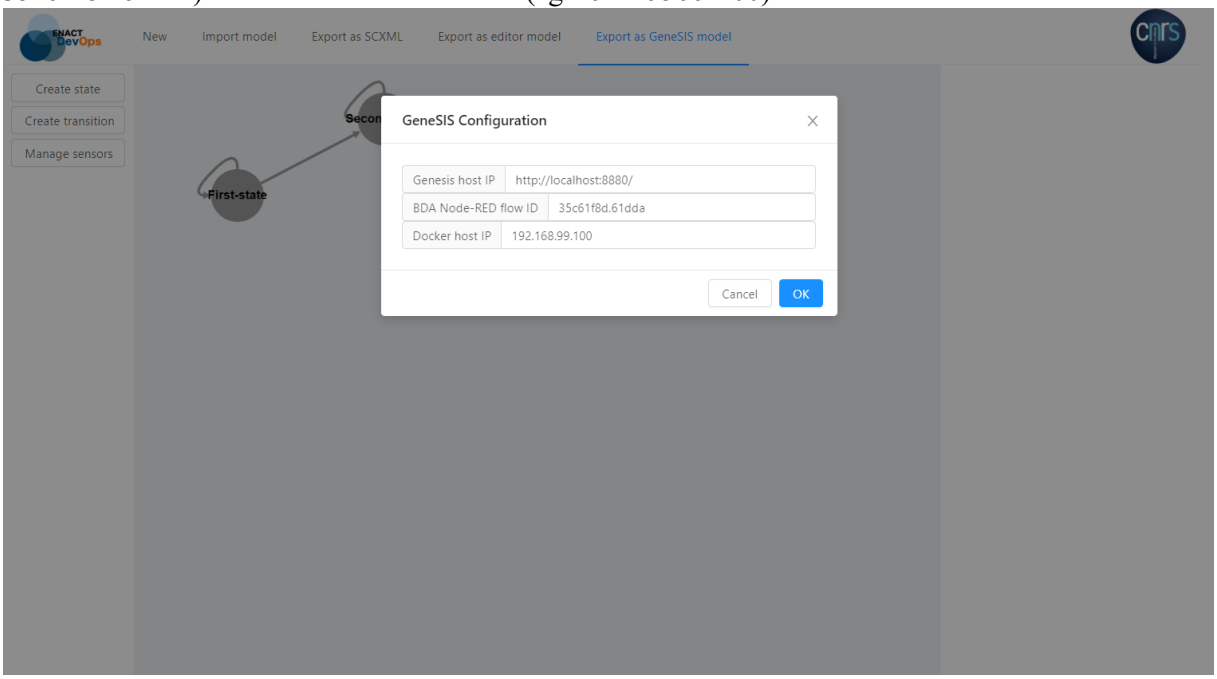
To create a transition between two states, first click on the `'Create transition'` button then click on the source state and then click on the target state. To edit transition properties, select transition by clicking on it and change its name and attached sensors.



Each sensor added to a state or transition must be configured, this is done by first setting the constraint field to the appropriate value, then clicking the ⚙ (cog) icon to open the parameter entry dialog and inputting the desired parameters.

11.3 Deploying the Behavioural Drift Computation model

To deploy the specified model, just select `'Export as GeneSIS' model` in the main menu. Then specify 3 parameters: the GeneSIS address (eg. http://localhost:8880/), the Node-RED flow ID (eg. 35c61f8d.61dda) and the Docker IP address (eg. 192.168.99.100).



This injects a new Docker instance running Node-RED and the BDC flow in the target GeneSIS application. However, the generated flow is not connected to the actual sensors, so some assembly is still required.

11.4 Editing the Behavioural Drift Computation model

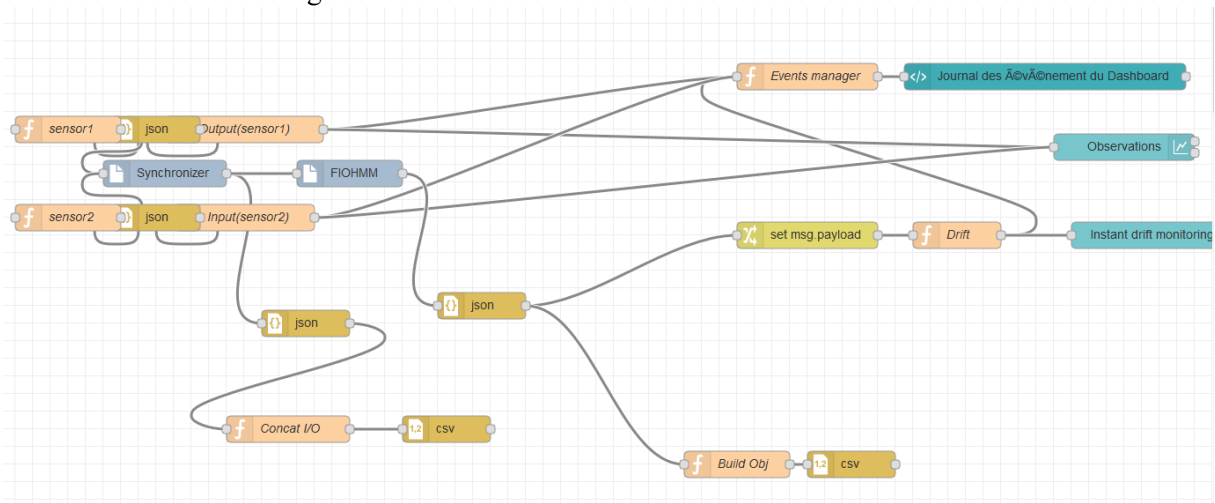
Edit the Node-RED flow to connect the real sensor data and publish the Behavioural Drift value. Access the Node-RED instance the flow is running on available at:

```
http://<docker-ip>:1880/
```

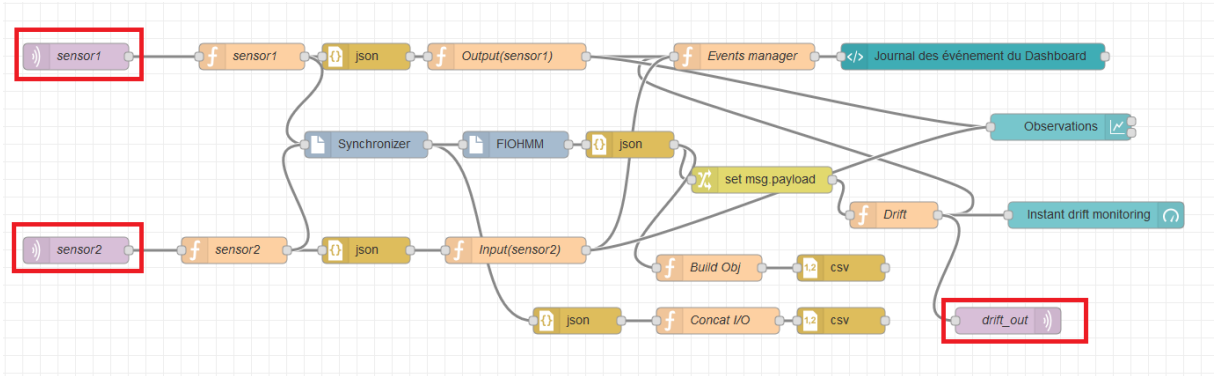
In our example, the instance will be available at:

```
http://192.168.99.100:1880/
```

A Node-RED flow looking like this:



Connect the sensor data and publish the behavioural drift value using the protocol of your choice. In this example, MQTT is used to subscribe to sensors' values and publish the behavioural drift value.



Deploy the flow and behavioural drift computations will begin. A default dashboard allows to view the computed values.

12. Security & Privacy Monitoring and Control User Guide

12.1 Security & Privacy Monitoring

12.1.1 Deployment and configuration

The Security & Privacy Monitoring Enabler is delivered as a docker based deployment. Docker needs to be installed in all machines used for deploying the Monitoring Enabler components. Docker installation is well explained in the documentation of Docker[2].

[2] About Docker CE installation documentation, https://docs.docker.com/install/

The Docker images of the Monitoring Enabler components will be generated and published in an artifact repository managed by Tecnalia: https://artifact.tecnalia.com/. They are available for authorised internal use of ENACT partners.

Network monitoring agents and Network Intrusion Detection System (NIDS)
Instances of the network monitoring agent and NIDS components need to be deployed in each of the networks to be monitored in the Smart IoT System. In order to monitor the network traffic that is not being sent to or from the monitoring machine, the network interface of the machine needs to be enabled in a promiscuous mode, and on a switched Ethernet network, it also needs to be set up the use of port mirroring.

System monitoring agents
Instances of the system monitoring agents need to be deployed in the machines of the SIS to be monitored.

Streaming bus, Data Capturing and Parser, IoT data storage, Monitoring dashboard (Server-side components of the Monitoring Enabler)
The server component parts of the Monitoring Enabler need to be deployed in a machine that at least fulfils the following hardware requirements of the base technologies:

- Streaming bus based on Kafka: HW requirements https://kafka.apache.org/documentation/#hwandos
- IoT data storage based on Elasticsearch: HW requirements https://www.elastic.co/guide/en/elasticsearch/guide/current/hardware.html

App monitoring agent
The app monitoring agent is particular of SMOOL IoT platform and its source code is published in ENACT gitlab: https://gitlab.com/enact/smool_enact.

The installation and usage guides are included there in the README.md file.

This agent execution is closely related to the execution of the SMOOL Platform. The usage guide of SMOOL can be found here: https://bitbucket.org/jasonjxm/smool/wiki/Home

12.1.2 Usage

The Security & Privacy Monitoring Enabler Dashboard will display all acquired data and generated events in a user-friendly manner in form of alerts, statistics and graphs.

First log in by an authorized user is required.

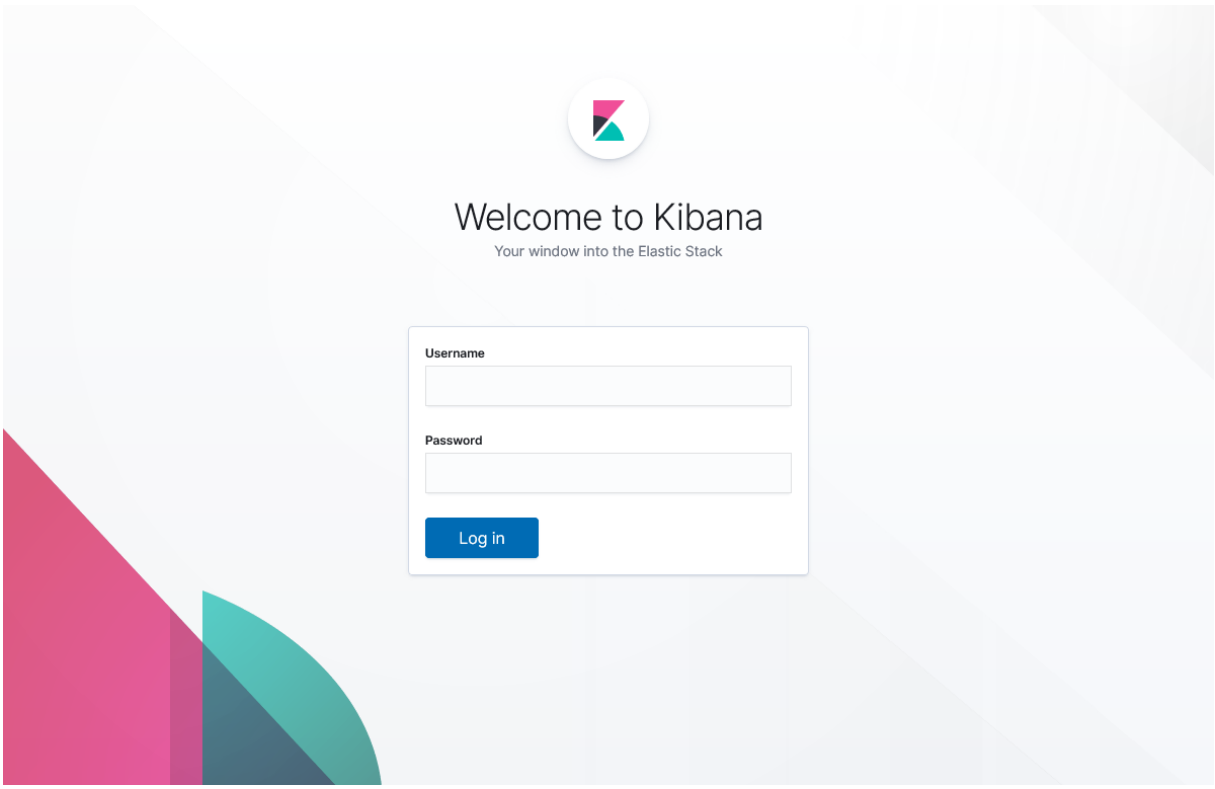


Figure 12. View of the Security & Privacy Monitoring Enabler dashboard – Log in

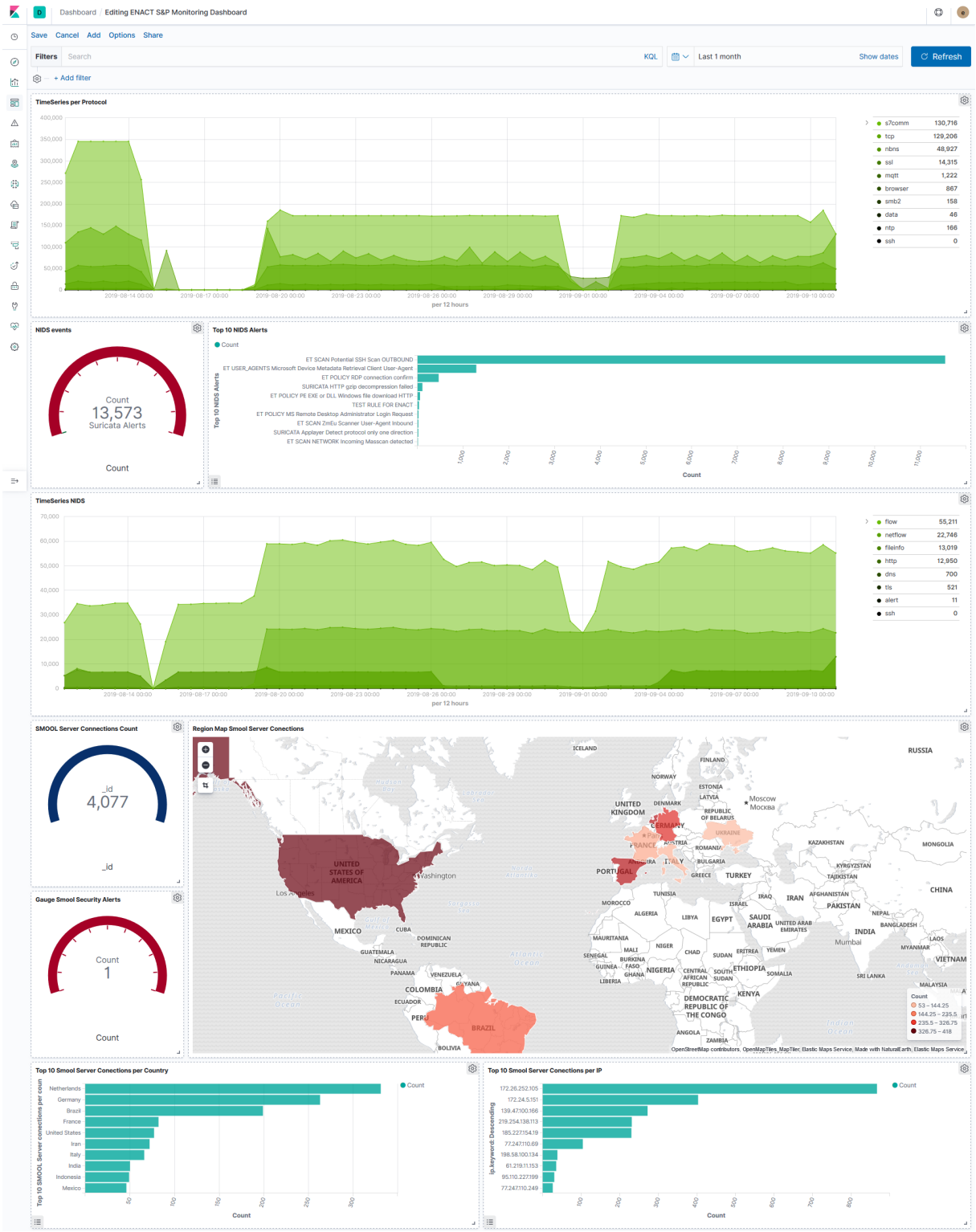


Figure 13. Overview of the Security & Privacy Monitoring Enabler dashboard

Multiple graphs such as time series of network traffic data and Network IDS events are shown in the dashboard. Figure 15 and Figure 15 show examples of the aforementioned graphs of time series.

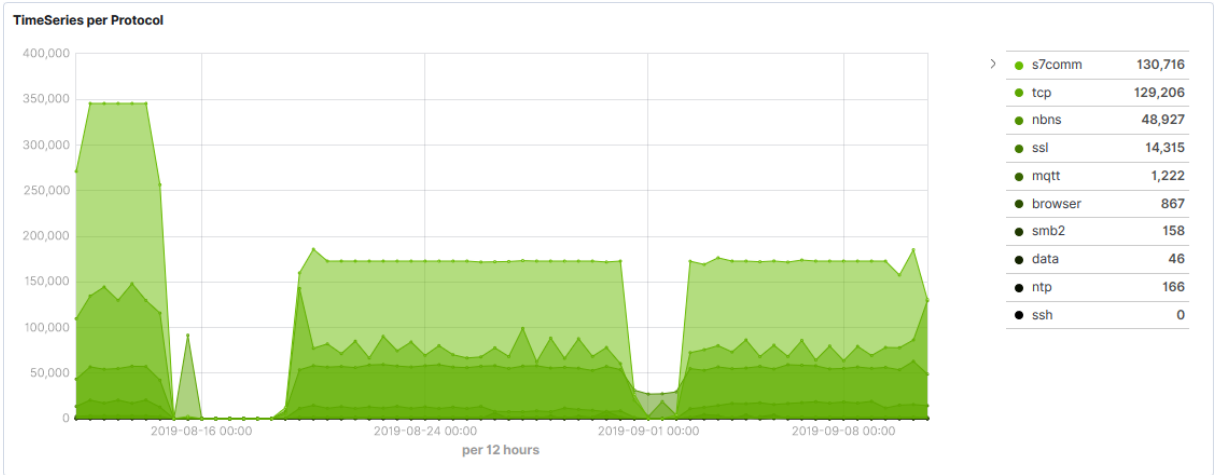


Figure 14. View of the Security & Privacy Monitoring Enabler dashboard – Network traffic time series graph

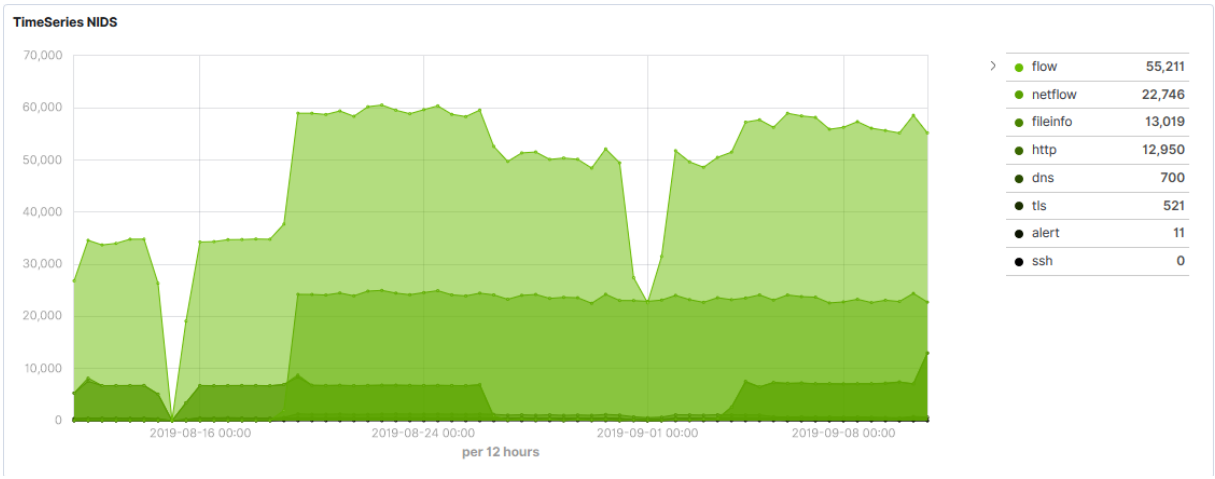


Figure 15. View of the Security & Privacy Monitoring Enabler dashboard – NIDS time series graph

More details related to NIDS events also are shown in other formats such as Top 10 of the alerts and number of events in a specific period.

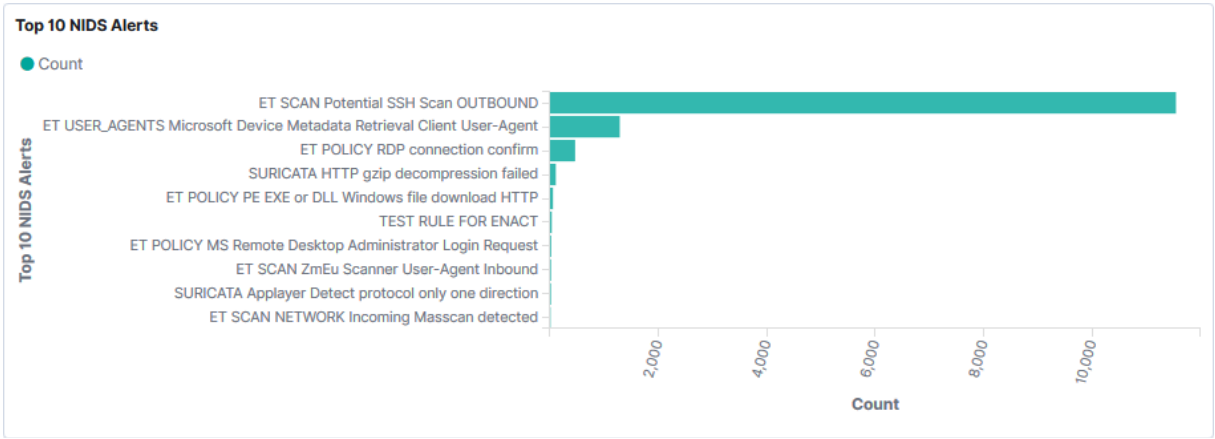


Figure 16. View of the Security & Privacy Monitoring Enabler dashboard – Top 10 NIDS alerts

Moreover, events related to SMOOL (SOFIA Platform) are also shown in the dashboard. For instance, Figure 17 shows the map of the connections received by SMOOL, and Figure 18 shows the number of security alerts identified by SMOOL.

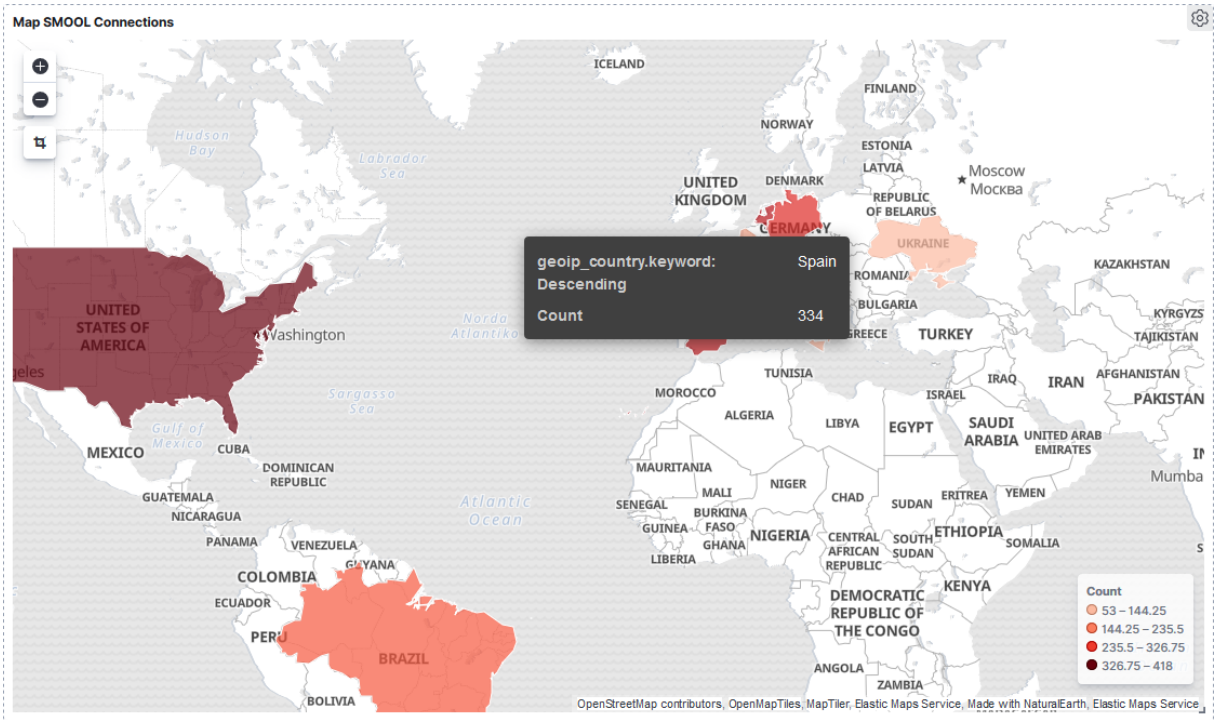


Figure 17. View of the Security & Privacy Monitoring Enabler dashboard – Map of SMOOL connections

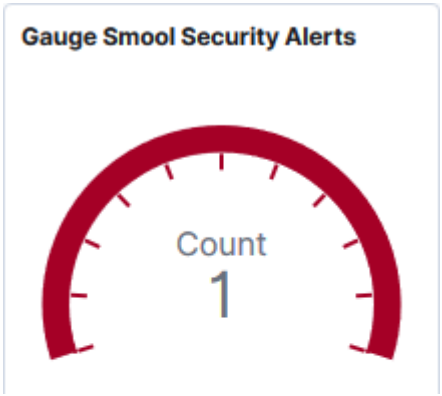


Figure 18. View of the Security & Privacy Monitoring Enabler dashboard – Number of security alerts related to SMOOL

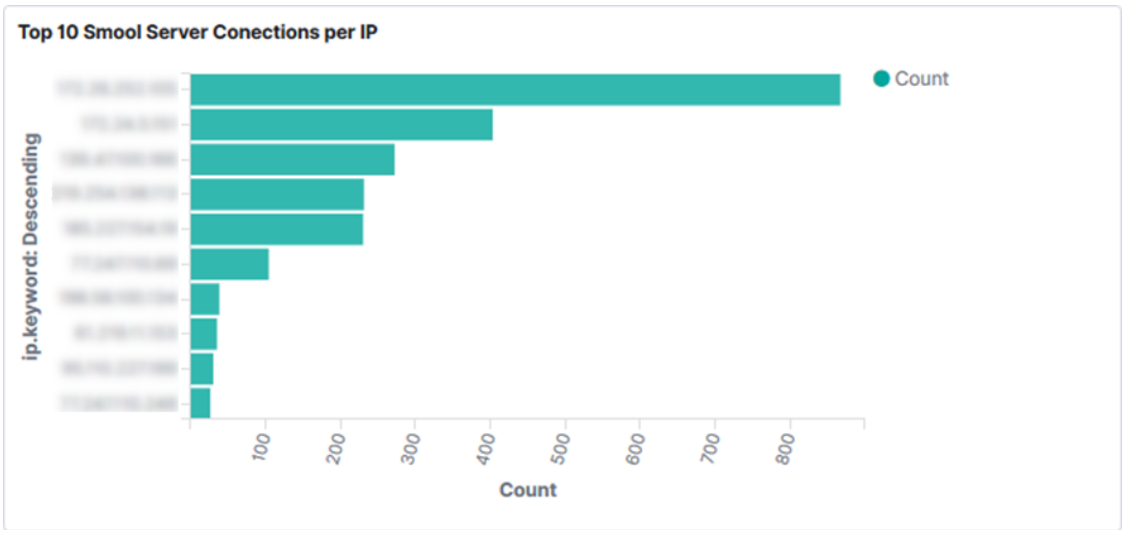


Figure 19. View of the Security & Privacy Monitoring Enabler dashboard –Top 10 of IP connections in SMOOL

At the top of the dashboard, the user can set the range of dates to visualize the data as shown in

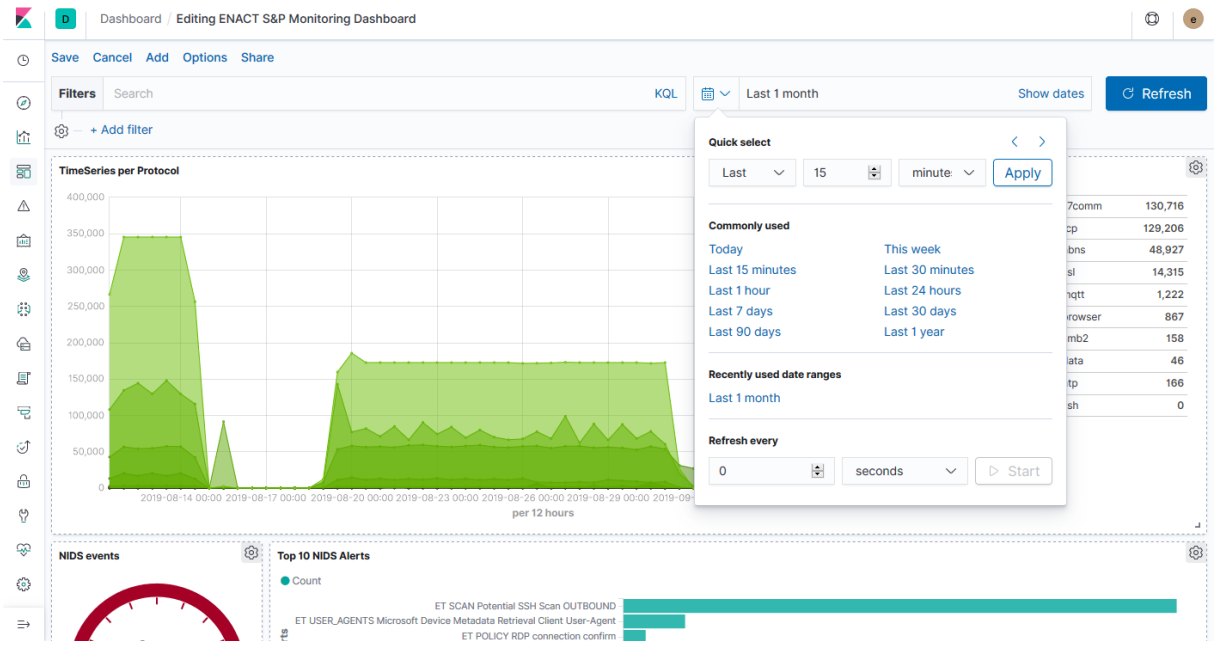


Figure 20. View of the Security & Privacy Monitoring Enabler dashboard – Set of date range

Additionally, the user can search for more details in the data stored in Elasticsearch (base technology of the Security & Privacy Monitoring Enabler dashboard) by clicking in "Discover" option of the navigation menu at the left. For that, the index desired by the user should be selected (network traffic data, NIDS event data, SMOOL related data, etc.) and the Kibana Query Language (KQL) should be use for specific queries and filters.

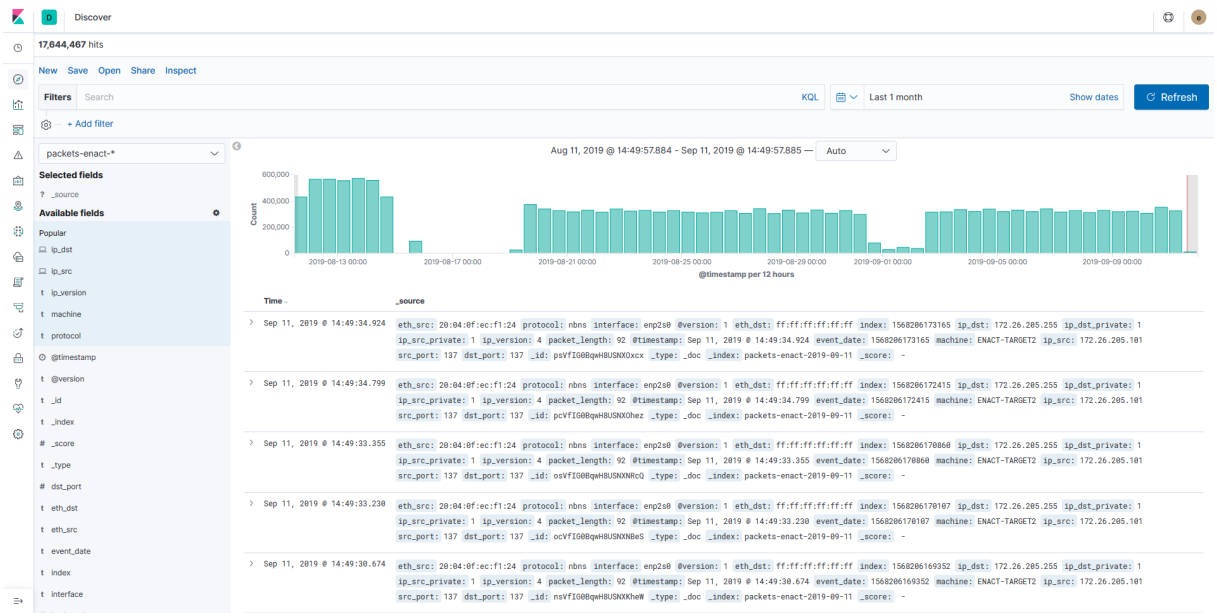


Figure 21. View of the Security & Privacy Monitoring Enabler dashboard – Set of date range

More details of the Security & Privacy Monitoring Enabler dashboard will be described in D4.3.

12.2 Security & Privacy Control in IoT platform

As part of the security and privacy controls developed in ENACT, the Control Manager will be able to manage the behaviour of the IoT Platform SMOOL (based on SOFIA technology) which is the platform used in the Smart building use case in ENACT. This will allow to enable the control of the communications between the "things" in the IoT environment by profiting from the intermediary nature of the IoT Platform in such communications.

To this end, SMOOL has been extended with a number of features oriented towards the full control of security and privacy of the data exchanged in the environment. Particularly, the features developed are described in deliverable D4.2 *Trustworthiness mechanisms for Smart IoT Systems - First version*.

There is no user interface to manage the SMOOL clients at runtime, as they worked in a automatic way. Next the deployment and configuration is explained.

12.2.1 Deployment and configuration

The Security& Privacy control mechanisms have been developed as extensions of the SMOOL clients of the SMOOL IoT platform and its source code is published in ENACT gitlab: https://gitlab.com/enact/smool_enact. The installation and usage guides are included there in the README.md file.

More information of the functionality provided by the SMOOL clients is described in D4.2 *Trustworthiness mechanisms for Smart IoT Systems - First version.*

The usage guide of SMOOL can be found here: https://bitbucket.org/jasonjxm/smool/wiki/Home

13. Root cause Analysis User Guide

The main objective of the Root Cause Analysis (RCA) module is to provide the ENACT platform with a reliable tool capable of detecting the origin of failures on the system. This engine relies on both instrumentation of the software (logs generation or specific instrumentation software) and monitoring of the devices and network. The RCA tool will use these data as the principal input to generate a graph of the system, which will be used to identify the scope of the detected anomalies. Later, the graph that represents the potential impact of the anomalies will be matched against a previously-assessed anomalies database, whose Root Cause is already known.

Due to one of the partners decision to leave the project, this decision direct impact and the subsequent change of the partners delivering the tool, the tool release has been postponed and it is planned to reach its initial release in M25 of the project.

14. Conclusion

In this deliverable we provided a set of the ENACT user guides on how to use ENACT enablers. For each tool we have used the most common use case. It should be added that most of the tools are at the first release level at this state and not all interactions are implemented according to the data formats and communication protocols defined in D5.2. However, this should not be the case in the final versions of the tools.