ENVELOPE

Evaluation and validation of connected mobility in real open systems beyond 5GS

Experimentation as a Service

User Manual

HORIZON JU Innovation Actions | 101139048 |
ENVELOPE - HORIZON-JU-SNS-2023

Co-funded by the European Union

6GSNS

Legal Disclaimer

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Smart Networks and Services Joint Undertaking (SNS JU). Neither the European Union nor SNS JU can be held responsible for them.

Table of contents

1 Introduction

The experimentation in the ENVELOPE project is managed by the EaaS module. The user interacts with the ENVELOPE Portal to perform any operations (e.g., launch experiment, retrieve information, load applications). It is the only point of access for exploiting the features offered by the ENVELOPE Platform. The ENVELOPE Portal consists of the Graphical User Interface (GUI) and interfaces to the back-end components of the EaaS module. The user must be authenticated before being able to operate on the ENVELOPE Portal.

The user will essentially perform three main actions:

- Onboarding an application (see Section 2)
- Onboarding an experiment (see Section 3)
- Interacting with the Portal to manage an experiment instance

In the first two sections, we will describe the building blocks required to define an application and an experiment. Finally, we will explain how to interact with the Portal to initiate and manage an experiment in Section 4.

2 Application Onboarding

The applications are used to compose a vertical service (i.e., a chained set of applications) to experiment in the ENVELOPE Platform.

Application Onboarding consists of uploading a single ZIP file representing the application and all related information. The contents of this ZIP file are referred to as the Application Package. The user can provide an arbitrary number of files (see example), and these files may eventually be used by the orchestrator to deploy the application.

An Application is the object instantiated from an Application Package. The Application Descriptor and its constituent Virtual Deployment Units (VDUs) are the descriptor objects used to instantiate the Application (see Section 3.3).

When defining an Application, the user must specify the name of the application, it will referred to as: <application-name>

2.1 Application Package

An Application Package is the smallest object able to be onboarded. It contains all the information related to a single application in terms of artifacts and descriptor. Two specific files are mandatory and must be named <application-name>.mf and <application-name>.yaml. These represent, respectively, the manifest of the package and the descriptor of the application.

```
!--------- NginxApplication.mf
!--------- NginxApplication.yaml
!--------- Artifacts
          !--------- MCIOPs
                     !--------- app-nginx-0.1.0.tgz
```

2.2 Application Manifest

An Application Manifest (<application-name>.mf) provides metadata about the application, including its version, compatibility information, and a list of files that make up the package. It is the first file to be parsed when processing the package and is considered the entry point that enables the system to interpret the contents of the package.

It consist of two block, one is the metadata fields where the application package metadata are listed, the second block consist a list of all constituent file you want to be onboarded.

2.2.1 Metadata Fields

All manifest metadata lives under the top-level metadata: key.

Name	Description	Example

appd_id	A unique identifier for this application descriptor.	nginx-appd-001
app_product_name	A human-readable name for your application. <application-name>	NginxApplication
app_provider_id	The UUID of the organization or provider publishing this descriptor.	123e4567-e89b-12d3-a456-426614174000
app_software_version	The version tag of the application image or code itself.	1.0.0
app_package_version	The version of this descriptor package (so you can bump it independently of the software).	1.0
app_release_date_time	The exact release timestamp of this package, in ISO-8601 format with timezone.	2017-01-01T10:00:00+03:00
appm_info	Application-Management (APPM) metadata string(s), often including the APPM schema version and a custom tag.	etsiappm:v2.3.1,0:myGreatAppm-1
compatible_specification_versions	A comma-separated list of Application-Descriptor specification versions that this package supports.	2.7.1,3.1.1

Tip: Always update app_release_date_time when you publish a new package, and append any new compatible spec versions to compatible_specification_versions.

```
metadata:
appd_id: nginx-appd-001
app_product_name: NginxApplication
app_provider_id: 123e4567-e89b-12d3-a456-426614174000
app_software_version: 1.0.0
app_package_version: 1.0
app_release_date_time: 2017-01-01T10:00:00+03:00
appm_info: etsiappm:v2.3.1,0:myGreatAppm-1
compatible_specification_versions: 2.7.1,3.1.1
```

2.2.2 Sources

List every file that must accompany this manifest. Order is not strict, but including the manifest first is conventional.

Sources	Description	Example

Source: <application-name>.mf	The manifest itself.	Source: NginxApplication.mf
Source: <application-name>.yaml	The main Application Descriptor.	Source: NginxApplication.yaml
Source: <file-location>	Any artifacts or other bundles in this package.	Source: Artifacts/MCIOPs/app-nginx-0.1.0.tgz

Tip: If your descriptor relies on multiple artifact packages (scripts, Helm charts, etc.), list each under its own Source: line.

```
Source: NginxApplication.mf
Source: NginxApplication.yaml
Source: Artifacts/MCIOPs/app-nginx-0.1.0.tgz
```

2.3 Application Descriptor

An Application Descriptor is a YAML template that lets you define all the metadata, deployment parameters, and interfaces for an application you want to deploy. This manual will help you:

Understand each top-level section of the descriptor.

Know which fields are mandatory vs. optional.

Write your own descriptor by modifying the example.

2.3.1 Metadata

Identifies the application uniquely and provides versioning info.

Name	Description	Example
appdId	A unique identifier for this application descriptor.	nginx-appd-001
appdExtInvariantId		nginx-appd-extinv-001
appProvider	The UUID of the organization or provider publishing this descriptor.	123e4567-e89b-12d3-a456-426614174000
appProductName	A human-readable name for your application. <application-name>	NginxApplication
appSoftwareVersion	The version tag of the application image or code itself.	1.0.0
appdVersion	The version of this descriptor package (so you can bump it	1.0

	independently of the software).	
appmInfo	Application-Management (APPM) metadata string(s), often including the APPM schema version and a custom tag.	Generic-ApplicationM

Tip: Choose appdId and appdExtInvariantId using your organization's naming conventions.

```yaml
appdId: nginx-appd-001
appdExtInvariantId: nginx-appd-extinv-001
appProvider: 123e4567-e89b-12d3-a456-426614174000
appProductName: NginxApplication
appSoftwareVersion: 1.0.0
appdVersion: "1.0"
appmInfo:
  - Generic-ApplicationM
```

2.3.2 Software Image(s) (swImageDesc)

Defines one or more container or VM images.

Each entry defines one image your application uses.

Name	Description	Example
id	The identifier of this software image	nginx_image_id
name	The name of this software image.	nginx
version	The version of this software image.	latest
swImage	This is a reference to the actual software image. The reference can be relative to the root of the Application Package or can be a URL.	nginx
supportedVirtualisationEnvironment	Specifies the virtualisation environments (e.g. hypervisor) compatible with this software image.	docker

Tip:

Add multiple images if your app has frontend/backend containers.

Ensure each id is unique.

2.3.3 OS Container Descriptor(s) (osContainerDesc)

Specifies resource requirements and links to software images. Describes resource needs for containers or VMs.

Name	Description	Example
osContainerDescId	Unique identifier of this OsContainerDesc in this descriptor.	nginx_container_desc_id
name	Human readable name of this OS container.	Nginx Desc
description	Human readable description of this OS container.	Nginx Container Descriptor
requestedCpuResource	Number of CPU resources requested for the container	1
requestedMemoryResource	Amount of memory resources requested for the container	586
cpuResourceLimit	Number of CPU resources the container can maximally use	1
memoryResourceLimit	Amount of memory resources the container can maximally use	1024
swImageDesc	Describes the software image realizing this OS container.	nginx_image_id

To customize:

Adjust CPU/memory to your app's footprint.

Link to the correct swImageDesc.id.

```
swImageDesc:
  - id: nginx_image_id
    name: nginx
    version: latest
    swImage: nginx
    supportedVirtualisationEnviroment:
      - docker
osContainerDesc:
  - osContainerDescId: nginx_container_desc_id
    name: Nginx Desc
    description: Nginx Container Descriptor
    requestedCpuResources: 1
    requestedMemoryResource: 586 #e.g in MB
    cpuResourceLimit: 1
    memoryResourceLimit: 1024
    swImageDesc: nginx_image_id
```

Virtual Deployment Unit(s) (vdu)

Groups containers/VMs into logical units.

Name	Description	Example
vduId	Unique identifier of this Vdu in this descriptor.	VDU_nginx
name	Human readable name of the Vdu.	Nginx Desc
description	Human readable description of the Vdu.	Nginx Container Descriptor
mcioIdentificationData	Name and type of the MCIO that realizes this VDU.	helm
osContainerDesc	Describes CPU, memory requirements and limits, and software images of the OS Containers realizing this Vdu.	nginx_container_desc_id

```yaml
vdu:
  - vduId: VDU_nginx
    name: Nginx-VDU
    description: "Default VDU hosting Nginx web server"
    mcioIdentificationData: helm
    osContainerDesc:
      - nginx_container_desc_id
```

Deployment Flavours (deploymentFlavour)

Describes a specific deployment version of a Application in terms of sizing/performance profiles and lifecycle parameters. Defines “flavours” (e.g., small, medium, large) with scaling and lifecycle settings.

Name	Description	Example
flavourId	Identifier of this DF within this descriptor.	df_nginx
description	Human readable description of the DF.	Default Deployment flavour for Nginx
vduProfile	Describes additional instantiation data for the VDUs used in this flavour.	See below
instantiationLevel	Describes the various levels of resources that can be used to instantiate the Application using this flavour	See below

To customize:

Define multiple flavours for different load scenarios.

Add or remove parameters under appLcmOperationsConfiguration.

Vdu Profile

Name	Description	Example
vduId	Uniquely references a VDU.	VDU_nginx
minNumberOfInstance	Minimum number of instances based on this VDU	1
maxNumberOfInstance	Maximum number of instances based on this VDU	4

Instantiation Level

Name	Description	Example
levelId	Uniquely identifies a level within the DF.	default_level
description	Human readable description of the level.	Default instanton level
vduLevel	Indicates the number of instances of this VDU to deploy for this level.	See below

MciopProfile

Name	Description	Example
mciopId	Identifies the MCIOP in the Application package.	Artifacts/MCIOPs/app-nginx-0.1.0.tgz
associatedVdu	List of VDUs which are associated to this MCIOP and which are deployed using this MCIOP.	VDU_nginx

```
deploymentFlavour:
  - flavourId: df_nginx
    description: Default Deployment flavour for Nginx
    vduProfile:
      - vduId: VDU_nginx
        minNumberOfInstances: 1
        maxNumberOfInstances: 4

    istantiationLevel:
      - levelId: default_level
        description: Default istantiation level
        vduLevel:
          - vduId: VDU_nginx
            numberOfInstances: 1

    mciopProfile:
      - mciopId: Artifacts/MCIOPs/app-nginx-0.1.0.tgz
        associatedVdu:
          - VDU_nginx
```

External Interfaces (appExtCpd)

Name	Description	Example
cpdId	dentifier of this Cpd information element.	ext

virtualCpd		Nginx80_connection_point

Lists connection points exposed externally. Exposes connection points to the outside world.

Virtual Connection Points (virtualCpd)

Defines network interfaces within the descriptor. Defines network interfaces that map to containers/VDUs.

Name	Description	Example
cpdId	Identifier of this Cpd information element.	nginx80_connection_point
layerProtocol	Specifies which protocol the CP uses for connectivity purposes.	IPV4 IPV6
description	Provides human-readable information on the purpose of the CP	Connection point to nginx service on port 80
vdu	References the VDU(s) which implement this service.	VDU_nginx
additionalServiceData	Additional service identification data of the VirtualCp exposed	portData: (seeBelow)

To customize:

Use portConfigurable: true if you want users to pick a different port.

Add multiple portData entries for multi-port services.

AdditionalServiceData

Name	Description	Example
name	The name of the port exposed by the CP.	nginx_port
protocol	The L4 protocol for this port exposed by the CP.	TCP
port	The L4 port number exposed by the CP.	80
portConfiguration	Specifies whether the port attribute value is allowed to be configurable.	false

```
appExtCpd:
  - cpdId: ext
    virtualCpd: nginx80_connection_point

virtualCpd:
  - cpdId: nginx80_connection_point
    layerProtocol:
      - IPV4
      - IPV6
    description: connection point to nginx service on port 80
    vdu:
      - VDU_nginx
    additionalServiceData:
      - portData:
        - name: nginx_port
          protocol: TCP
          port: 80
          portConfigurable: false
```

3 Experiment Onboarding

4.1 Experiment Package

An Experiment Package contains all the information related to a single experiment in terms of artifacts and descriptor.

4.2 Experiment Manifest

The Experiment Descriptor Manifest (.mf) is a lightweight YAML-style file that describes:

- Who created the experiment descriptor.
- Which version of the descriptor schema it follows.
- When it was released.
- Which other files (e.g., the main YAML descriptor) are part of this experiment package.
- Which specification versions it's compatible with.

This manifest makes it easy for tooling and users to check compatibility before attempting to deploy or process the descriptor. And is the file used by the module to parse your package

All manifest metadata lives under the top-level metadata: key. In your MyExperiment.mf, it looks like:

Field Details

Name	Description	Example
experimentd_designer	Who authored or maintains this descriptor (e.g., a person, team, or organization).	ExampleDesigner
experimentd_invariant_id	A stable machine-friendly ID that never changes, even if you bump the manifest version.	my_experiment
experimentd_name	A human-readable title shown in UIs or logs	MyExperiment
experimentd_file_structure_version	The version of the manifest schema itself. Increment this if you ever change the manifest format.	1.0
experimentd_release_date_time	When this descriptor version was released, in ISO-8601 format (including timezone).	2018-04-08T10:00+08:00
compatible_specification_versioexperiment	Which experiment-descriptor specification versions this	2.7.1,3.1.1,4.3.1

	manifest supports, comma-separated.	

Tip: Always update experimentd_release_date_time when you publish a new descriptor version, and add any new compatible spec versions here.

Sources

Below the metadata block, list each file that constitutes this experiment package. In your example:

Sources	Description	Example
Source: <experiment-name>.mf	The manifest itself.	Source: MyExperiment.mf
Source: <experiment-name>.yaml	The main Experiment Descriptor.	Source: MyExperiment.yaml

Each Source: line names a file that must accompany the manifest.

Ordering is not critical, but including the manifest itself first helps humans and tools verify integrity before loading the descriptor.

Tip: If your experiment descriptor splits across multiple files (e.g., separate network and compute fragments), list each here.

```
metadata:
experimentd_designer: ExampleDesigner
experimentd_invariant_id: my_experiment
experimentd_name: My Experiment
experimentd_file_structure_version: 1.0
experimentd_release_date_time: 2018-04-08T10:00+08:00
compatible_specification_versioexperiment: 2.7.1,3.1.1,4.3.1

Source: MyExperiment.mf
Source: MyExperiment.yaml
```

4.3 Experiment Descriptor

An Experiment Descriptor is a YAML template that lets you define a multi-application experiment, including which application descriptors to include, how many instances to run, and security settings. This manual will help you:

Understand each top-level section.

Identify mandatory vs. optional fields.

Create your own descriptor by modifying the provided example.

Identity & Designer

Use your own naming convention for experimentdIdentifier and experimentdInvariantId to ensure uniqueness.

Included Applications (appdid)

List the Application Descriptor IDs that this experiment will deploy:

Each entry must match a valid appdId from your Application list.

You can include as many applications as needed.

Name	Description	Example
experimentdIdentifier	Identifier of this descriptor information element.	my_experiment_1
designer	Specifies the designer of this experiment.	ExampleDesigner
experimentdName	Provides the human readable name of the experiment	My Experiment
experimentInvariantId	Identifies a descriptor in a version independent manner.	my_experiment
appdid	References the application descriptors of this experiment.	- nginx-appd-001 - busybox-appd-001

Experiment Deployment Flavours (experimentDf)

Defines one or more Deployment Flavours—similar to “profiles”—that group application profiles under a flavour key.

Name	Description	Example
experimentDfId	unique within this descriptor.	df_my_experiment
flavourKey	arbitrary tag (e.g., “small”, “test”).	string
appProfile	List of app profile	See below

AppProfile

Name	Descriptionr	Example
appProfileId	local reference for the profile.	experiment_nginx_profile_1
appdId	link to the original application descriptor.	nginx-appd-001
appdExtInvariantId	link to the original application descriptor.	nginx-appd-extinv-001
flavourId	refer to the Application Descriptor own flavours.	df_nginx
instantiationLevel	refer to the Application Descriptor own level.	default_level

minNumberOfInstances	control scaling bounds.	1
maxNumberOfInstance	control scaling bounds.	1

Tip: Add additional experimentDf blocks if you need multiple experiment-wide flavours.

Experiment Instantiation Levels (experimentInstantionLevel)

Maps profiles to actual instance counts for a given experiment level:

Name	Description	Example
nsLevelId	Identifier of this NsLevel information element. It uniquely identifies an NS level within the DF.	default_experiment_level
descriptiorn	Human readable description of the Experiment level	Simple experiment level
appToLevelMapping	Specifies the profile of the Applications involved in this Experiment level and, for each of them, the required number of instances.	See below

nsLevelId: e.g., “default”, “scale-out”, etc.

appToLevelMapping: ties each appProfileId to an exact replica count.

Name	Descriptiorn	Example
appProfileId	References the profile to be used for a VNF involved in an NS level.	experiment_nginx_profile_1
numberOfInstance	Specifies the number of VNF instances required for an NS level.	1

Tip: Define multiple levels (e.g., “stress_test_level”) to switch between configurations.

Security

Provides a signature section to prevent tampering

Name	Description	Example
signature	cryptographic signature of the descriptor.	<string>
algorithm		SHA256
certificate	reference to a certificate (optional).	

```yaml
experimentdIdentifier: my_experiment_1
designer: ExampleDesigner
experimentdName: My Experiment
experimentdInvariantId: my_experiment
appdid:
  - nginx-appd-001
  - busybox-appd-001
experimentDf:
  - experimentDfId: df_my_experiment
    flavourKey: string
    appProfile:
      - appProfileId: experiment_nginx_profile_1
        appdId: nginx-appd-001
        appdExtInvariantId: nginx-appd-extinv-001
        flavourId: df_nginx
        instantiationLevel: default_level
        minNumberOfInstances: 1
        maxNumberOfInstance: 1
      - appProfileId: experiment_busybox_profile_1
        appdId: busybox-appd-001
        appdExtInvariantId: busybox-appd-extinv-001
        flavourId: df_busybox
        instantiationLevel: default_level
        minNumberOfInstances: 1
        maxNumberOfInstance: 1
    experimentInstantionLevel:
      - nsLevelId: default_experiment_level
        description: simple experiment level
        appToLevelMapping:
          - appProfileId: experiment_nginx_profile_1
            numberOfInstances: 1
          - appProfileId: experiment_busybox_profile_1
            numberOfInstances: 1
security:
  signature: string
  algorithm: string
  certificate: not_specified_0_to_1
```

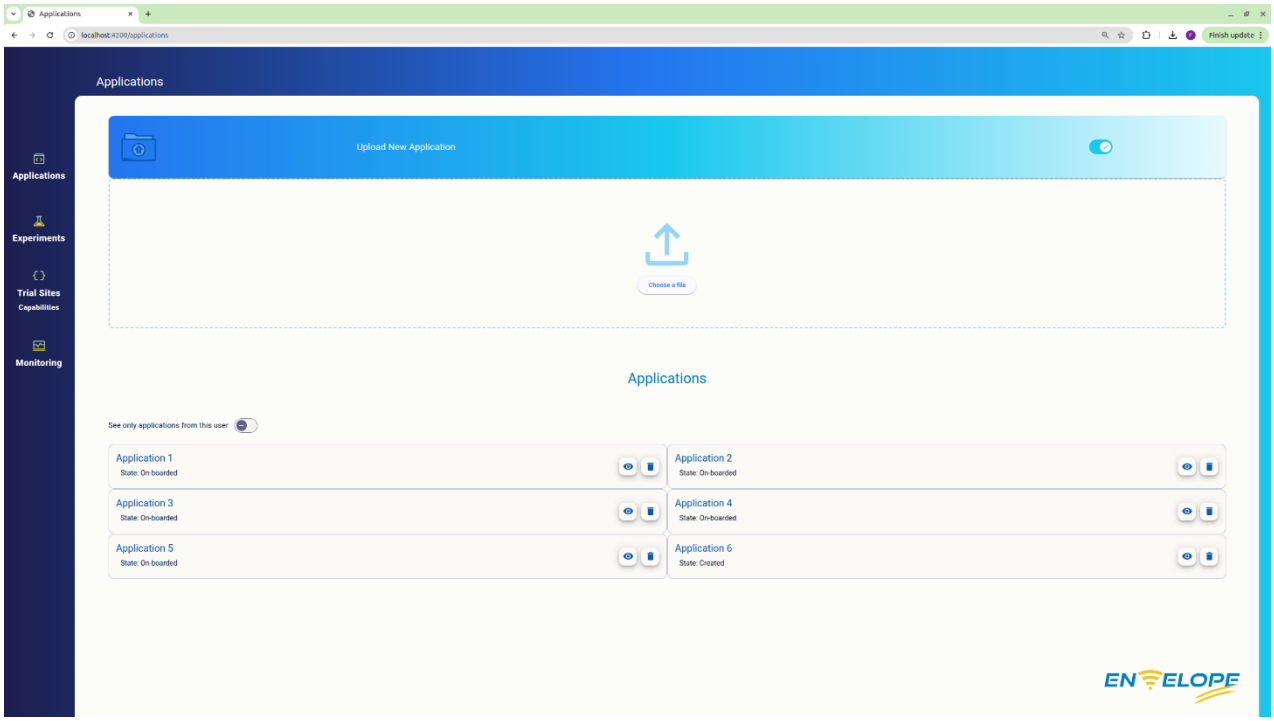
4 Graphical User Interface

This section provides an overview about the steps that an experimenter has to perform for interacting with the ENVELOPE Portal (i.e., the EaaS front-end).

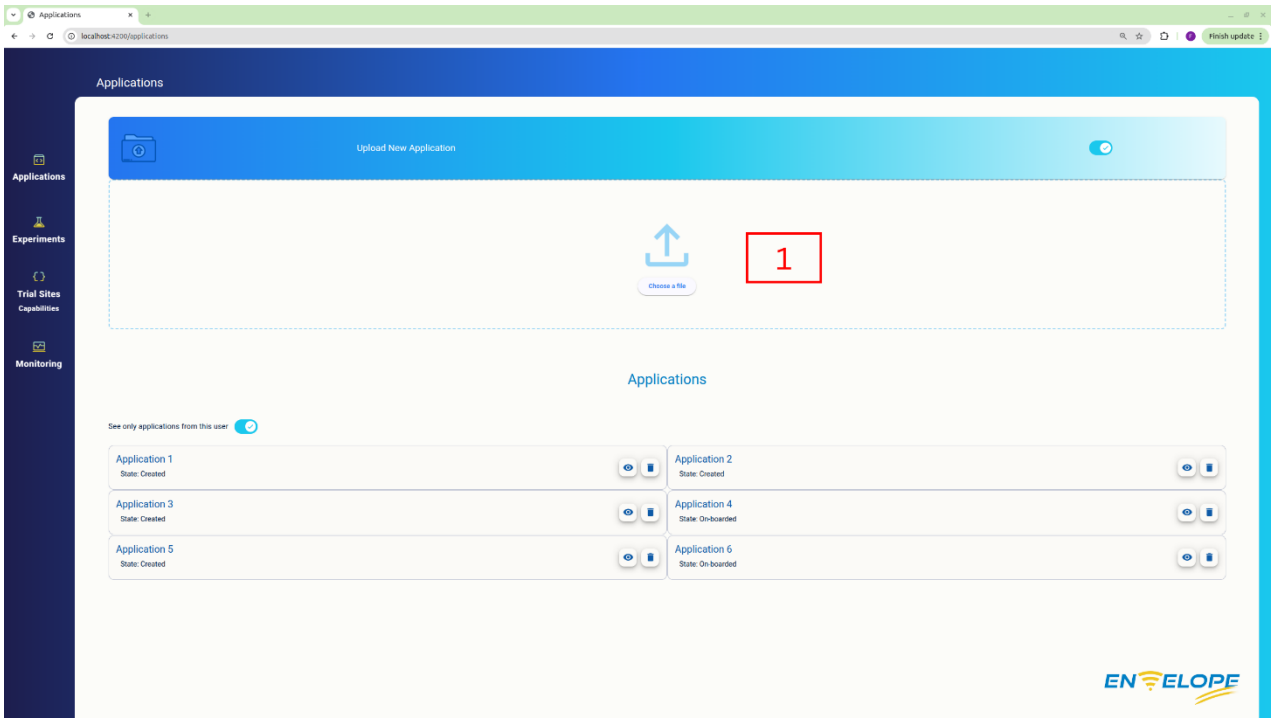
4.1 Applications

The following steps illustrates the procedure for managing the applications that are part of an experiment.

The screen is horizontally divided between an upper part for uploading and onboarding, and a lower area for retrieving and visualizing existing created or on-boarded application packages.



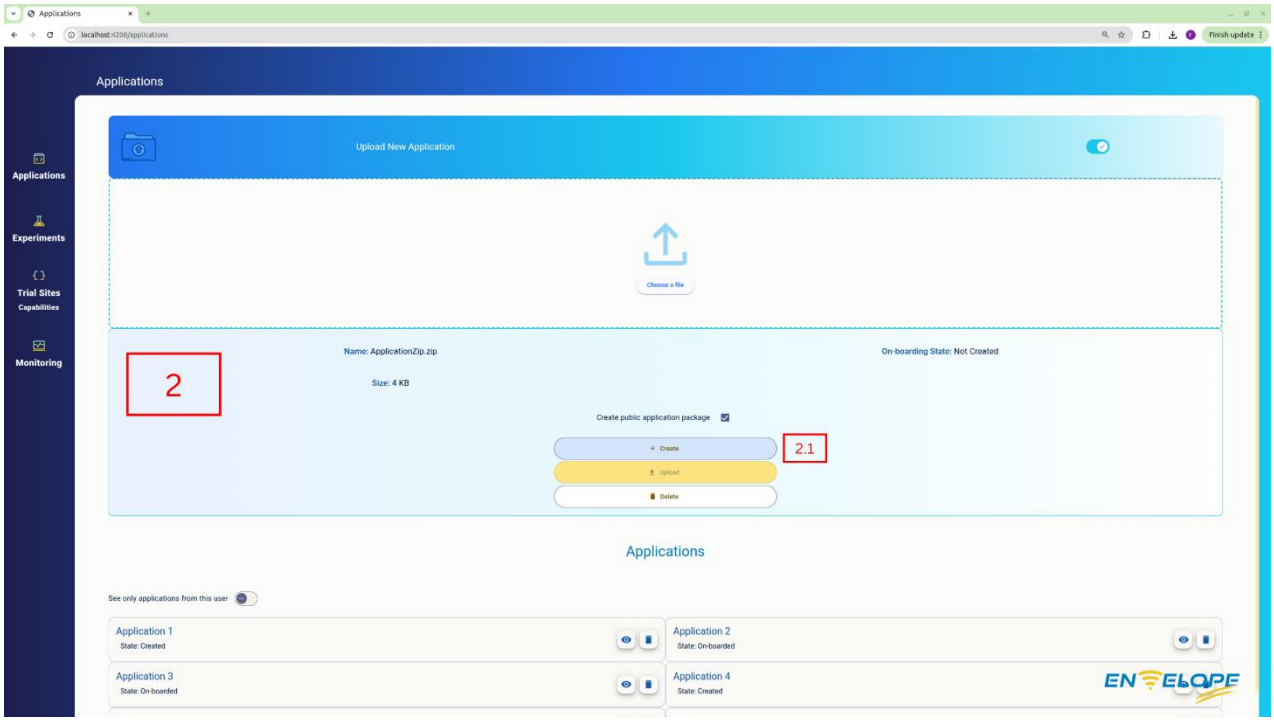
1. Upload local zip file: click “Choose a file” and select from your filesystem the zip file for your application. Alternatively, drag & drop it into the light blue dotted border area.



2. On-boarding an application package: a new UI dashboard shows up to manage the task.

2.1. Click “Create” to request the creation of a new application package.

Note: this Step does not involve the zip file uploaded in Step 1. It just initializes an empty application package, whose content may be uploaded later.

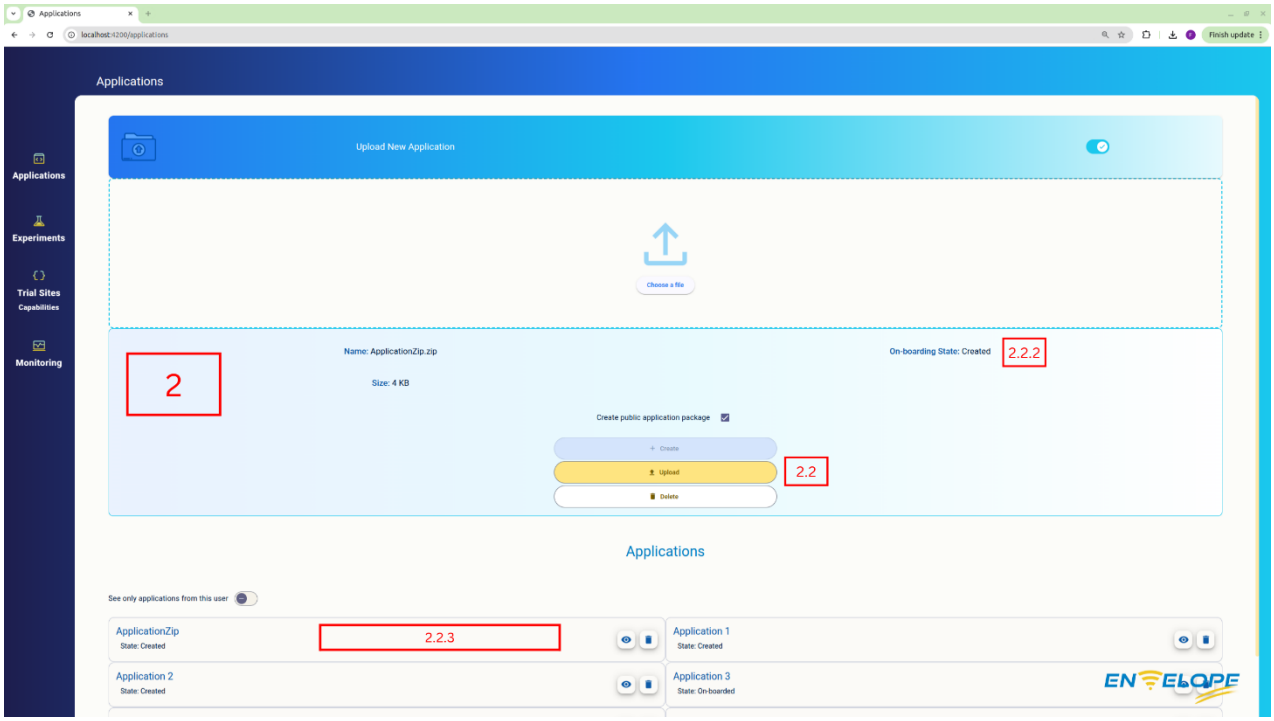


2.2. After 2.1 is completed:

2.2.1. “Upload” button becomes enabled, “Create” button becomes disabled.
2.2.2. “On-boarding State” info tab switches to “Created”.
2.2.3. A newly created package appears in the list of stored application packages.
2.2.4. Click the “Upload” button to upload the application package created in 2.1.

Note: The content of the zip file uploaded in Step 1 will be used for uploading the content of the application package created in Step 2.1.

2.3 It's possible to "Delete” the zip file from client and its associated application package.

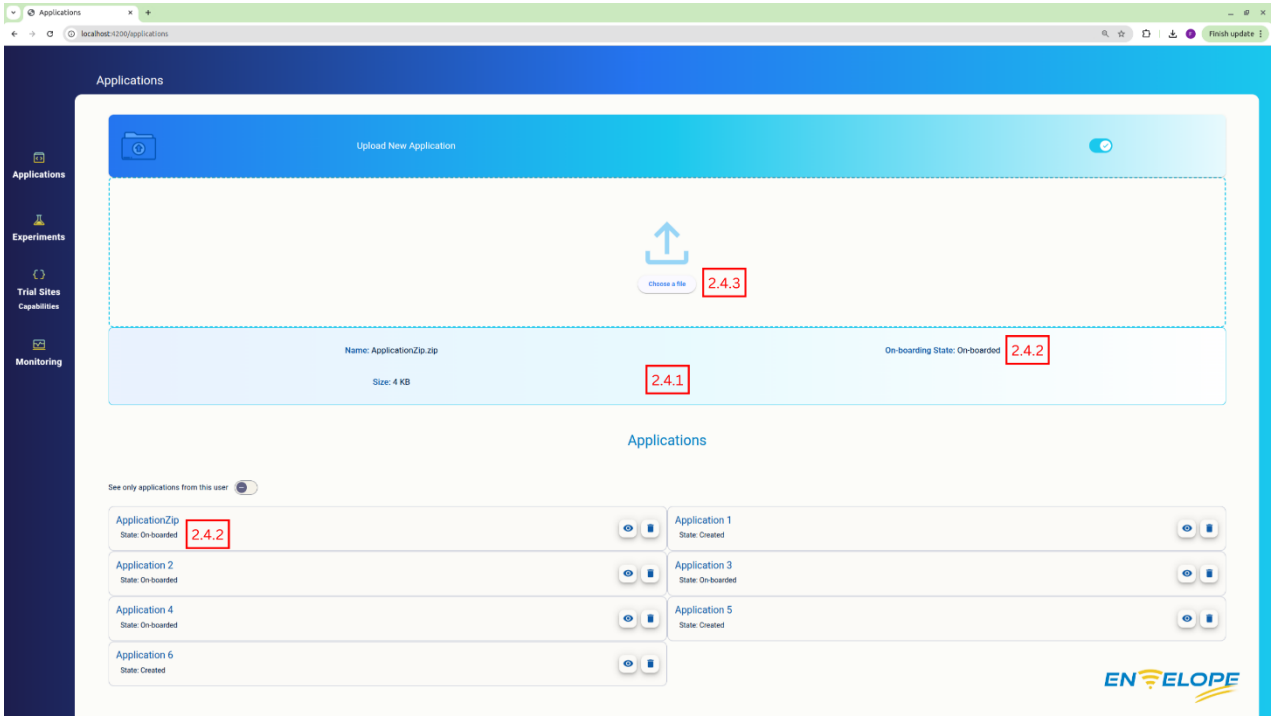


2.4. After 2.2.4 is completed:

2.4.1. “Upload” and “Create” buttons disappear.
2.4.2. “On-boarding State” info tab switches to “On-boarded”.

Note: the system will take a few seconds for on-boarding the package. Right after 2.2.4, the package will transition through the following states - “Created” → “Uploading” → “Processing” - before finally settling on “On-boarded.” This final state appears immediately after the system completes the on-boarding process.

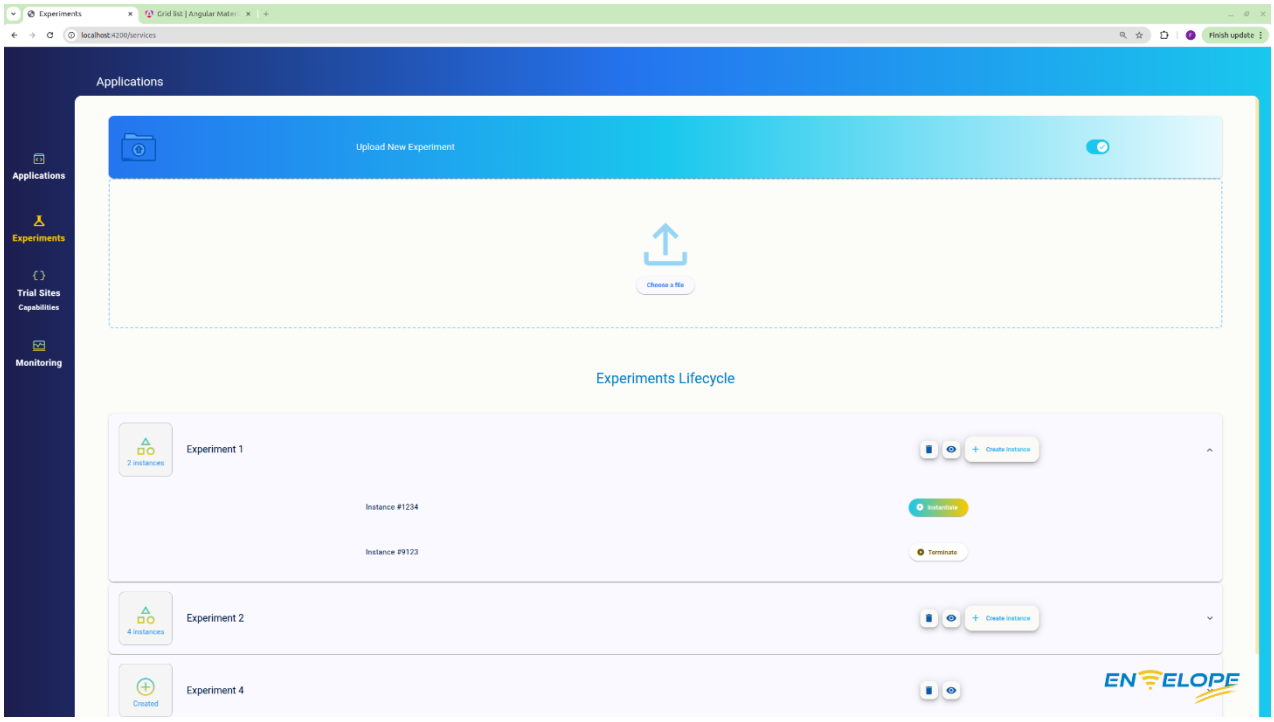
2.4.3 Can start again from Step 1.



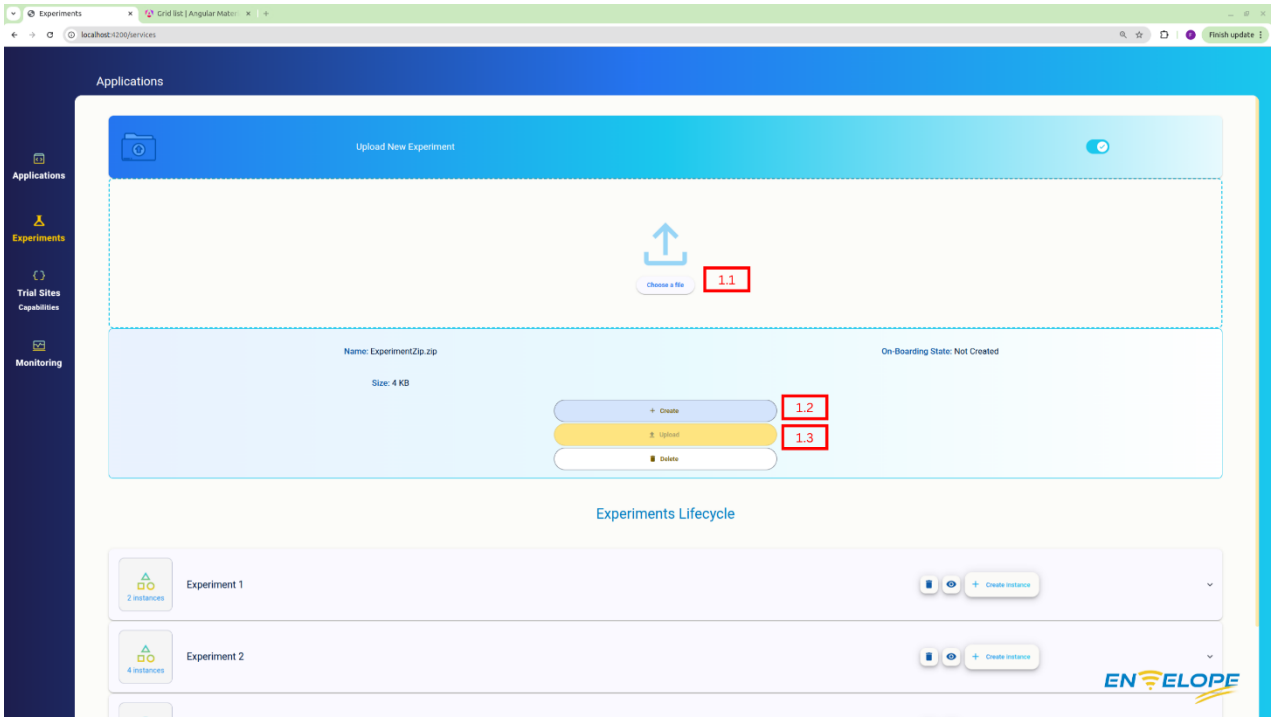
4.2 Experiments

The screen is horizontally divided between an upper part for uploading and onboarding an experiment descriptor, and a lower area for retrieving and visualizing existing created, on-boarded and/or instantiated experiments (lifecycle).

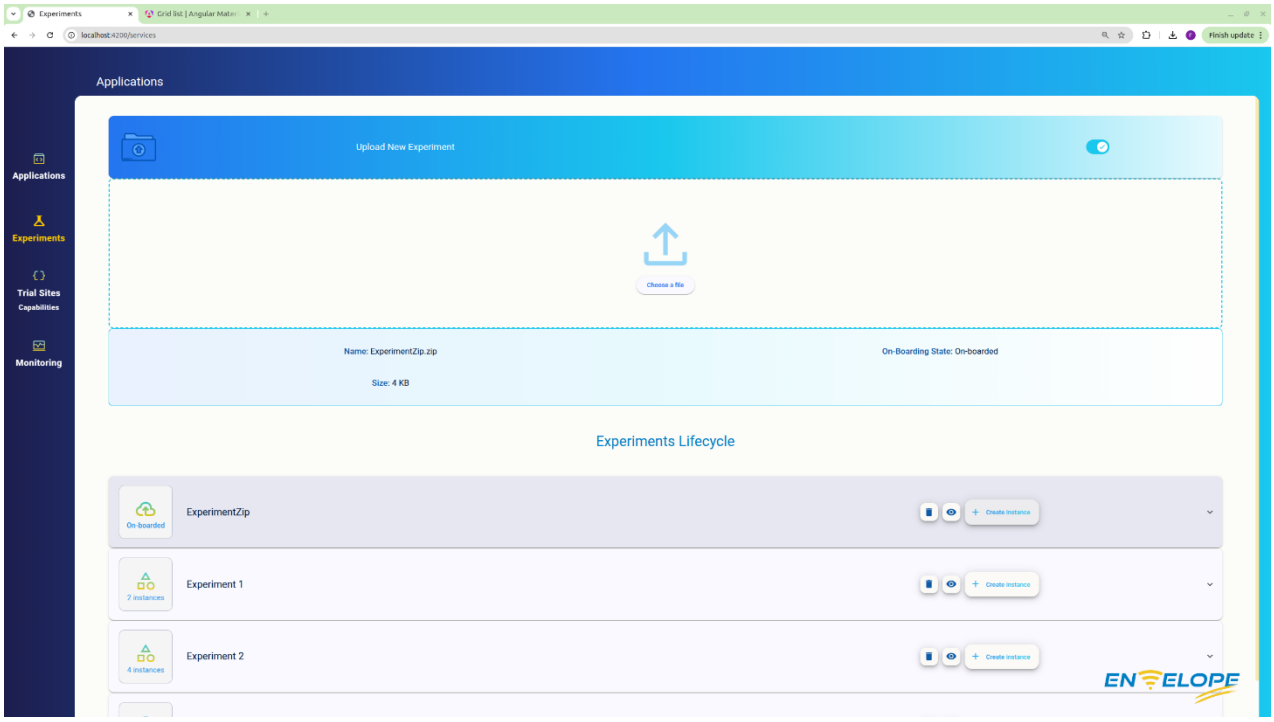
In the lower area, each experiment descriptor is represented as an expansible card; once expanded, it shows its created instances, with buttons to terminate each of those that are instantiated, or to instantiate those which are not.



1. The creation and on-boarding process is the same as illustrated in Paragraph 2.1 Applications. Following steps must be performed sequentially:
    1.1. Locally upload a zip file containing the desired experiment to onboard.
    1.2. Tap “Create” to create an Experiment Descriptor.
    1.3. Tap “Upload” to upload Experiment Descriptor Archive with the content of the zip file.



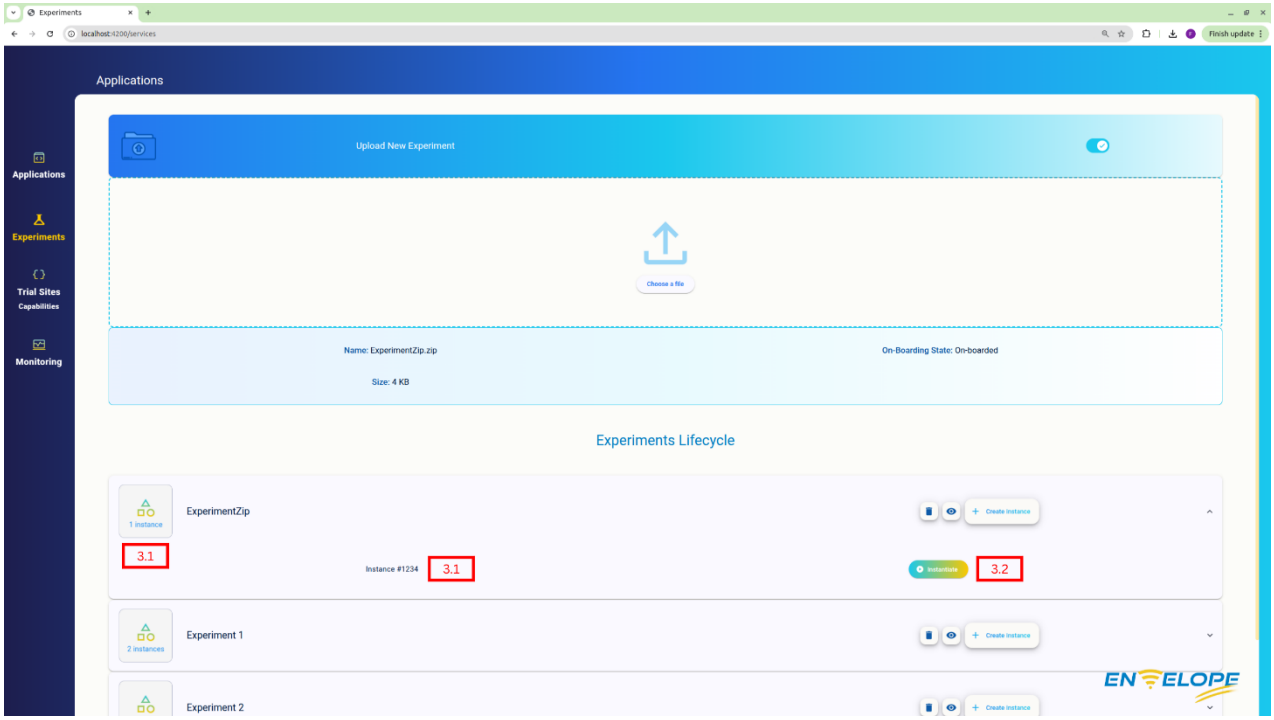
2. The experiment now is shown listed in the lower screen area.
    2.1. Click “Create instance” to create an Experiment identifier.



3. After 2.1:

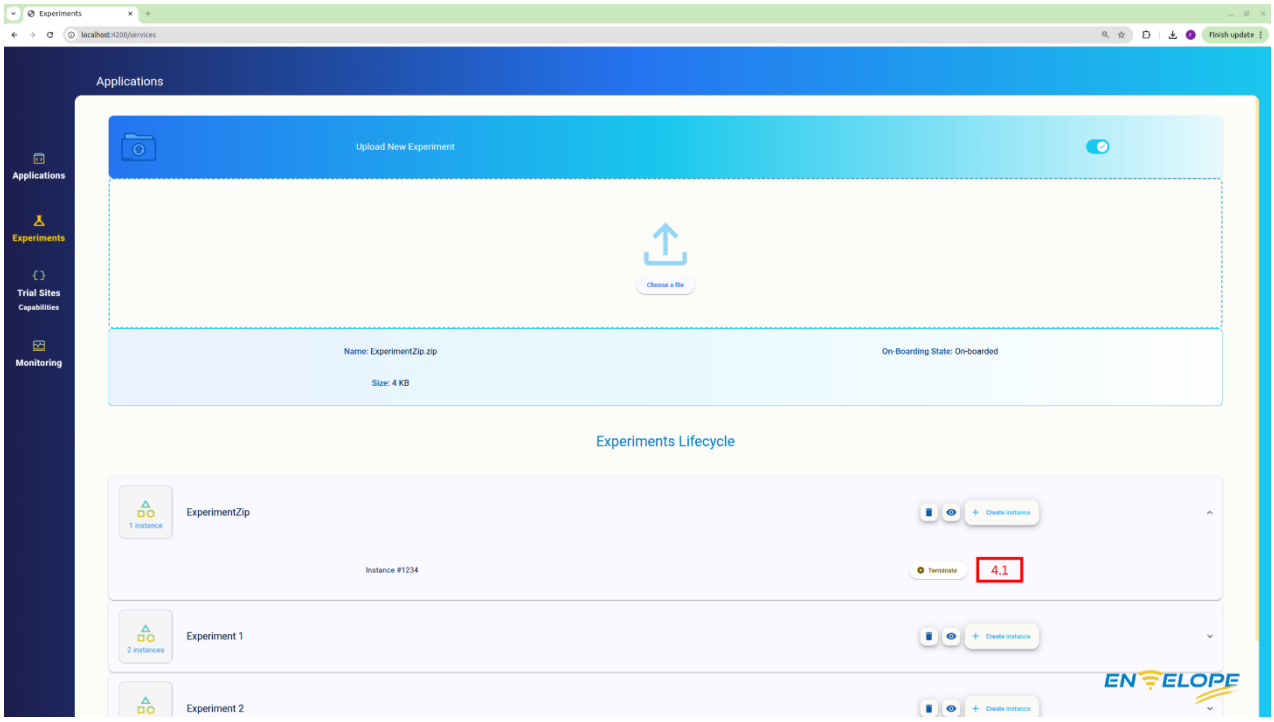
    3.1. New experiment instance is created and visible

    3.2. Click “Instantiate” button to run the experiment instance



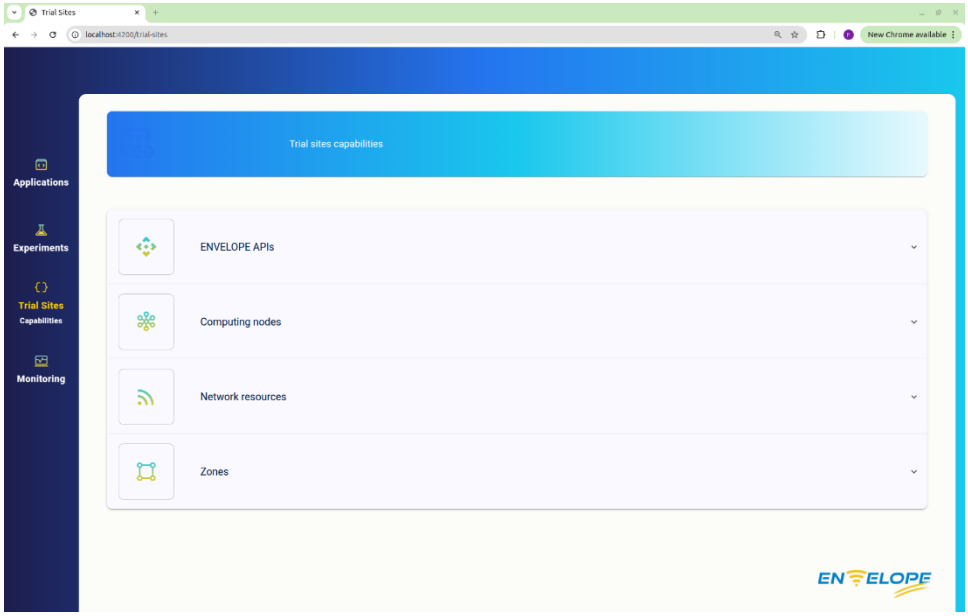
4. After 3.2:

4.1. “Terminate” button is available. Click to stop the instance.

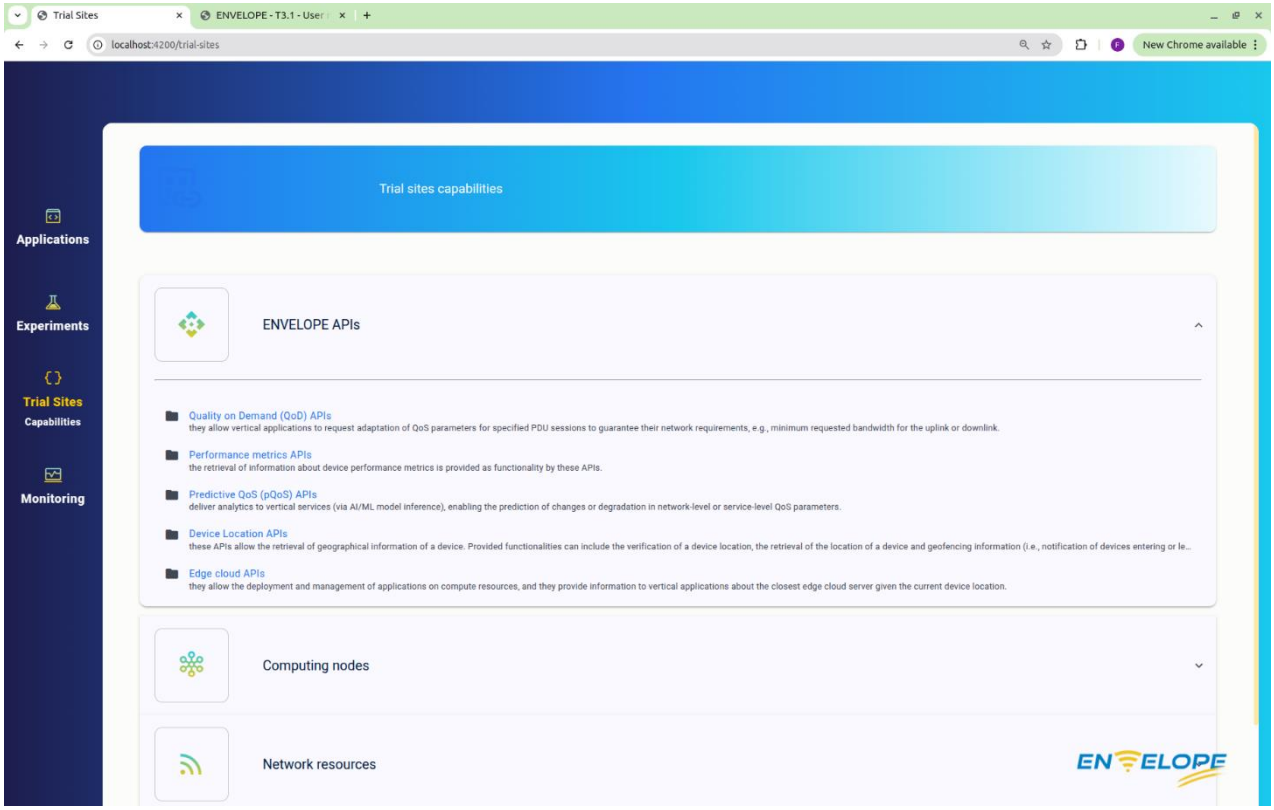


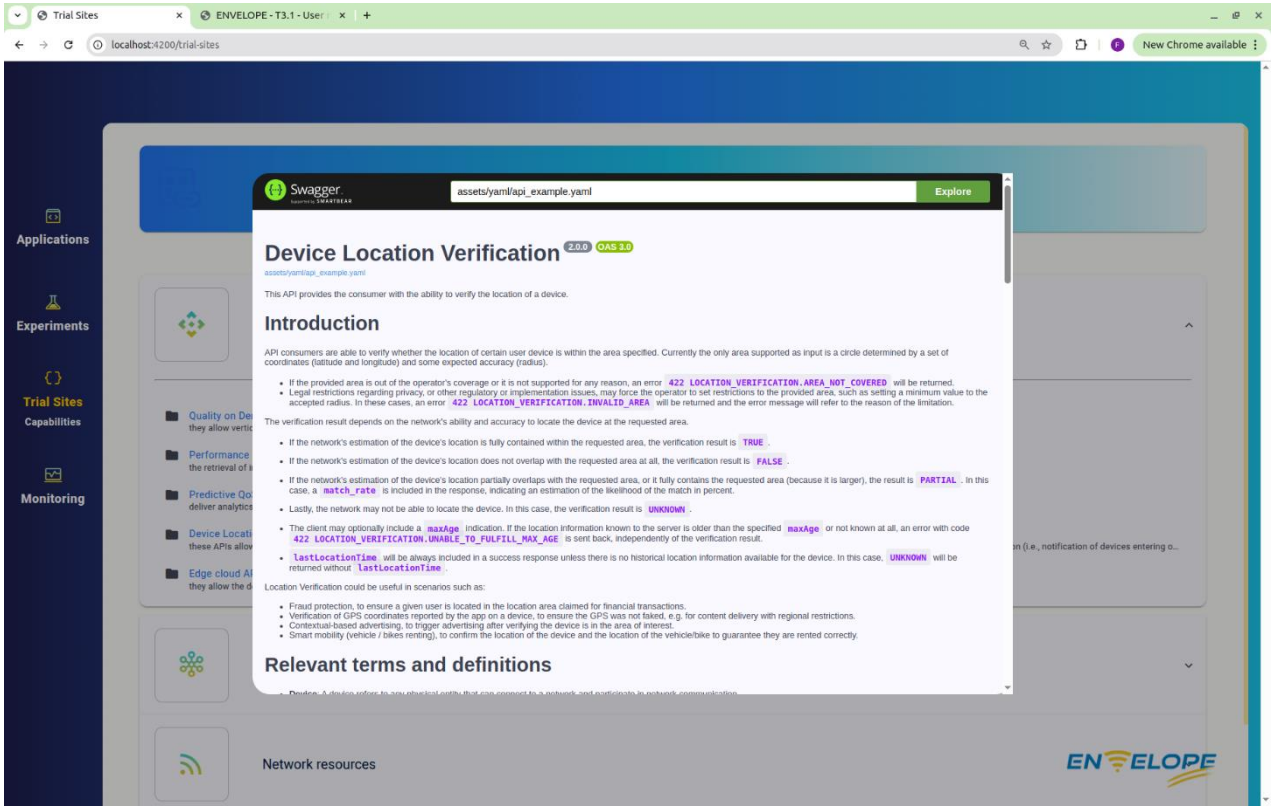
4.3 Trial Sites Capabilities

This section provides information about ENVELOPE APIs specification, Computing Nodes, Network Resources and Zones.



The ENVELOPE APIs specifications supported can be retrieved and visualized using Swagger.





4.4 Monitoring

The trial site's metrics and experimentation metrics are collected through Prometheus and they can be visualized in a Grafana dashboard.

An example of monitoring dashboard is shown below. In this case, the visualisation focuses on network metrics, providing panels for packet forwarding and drop rates, as well as session and IP pool utilisation, or system resource usage.