



Software Defined AppLication Infrastructures management and Engineering

SODALITE Experiments User Manual

The SODALITE Team
31/01/2022



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825480.



Table of contents

1. Introduction	2
2. Exercise to be accomplished	2
3 Reference material	3
4. The application to be deployed	3
5. Configuration of the SODALITE platform	6
User registration	6
Installation and configuration of the IDE	7
1- Requirements	7
2- Configure Sodalite backend connection	7
3 - Configuration for public testbed:	9
6. Experiments	9
Modeling phase	9
Exercise A - modelling an abstract application deployment model	10
Deployment phase and error handling	14
Exercise B - testing error handling.	15
Execution phase	17
Exercise C - checking that the machine learning application is working properly	17
Visualize the monitoring dashboard (Yosu)	19
Undeploy the ML application	19
7. Exercise finalization and conclusion	19



1. Introduction

This document aims at guiding end users in the usage of the SODALITE platform in a simple but meaningful case.

SODALITE (<https://sodalite.eu/>) is an open-source project that provides support to the development, deployment, operation, and execution of complex applications on heterogeneous cloud/HPC resources, with a particular focus on performance, quality, manageability, and reliability. In particular, it offers the following assets:

- *Smart IDE - Application Deployment Modelling*: it supports the deployment modelling activity and the subsequent generation of executable TOSCA and Ansible IaC. The models developed with the Smart IDE concern the structure of the application to be deployed, its mapping on specific resources, the definition of the operations to configure resources and application components, QoS objectives and technological constraints to be used as part of performance optimization and monitoring.
- *PESTO (dePloyment, orChEstraTion & prOvisioning)*: it is able to automate all phases that lead to the proper operation of an application on top of the defined infrastructure.
- *FindIaC Bug - Verification and Bug Prediction in the IaC code*: it analyses TOSCA and Ansible code to identify potential smells that are then submitted to the users together with proposals for code improvement.
- *RIDER (RunTIme DEployment Refactoring)*: When the application is running in its execution environment, RIDER monitors its performance to check whether the defined QoS objectives are met. Two types of reconfiguration mechanisms are supported. On-the-fly vertical scaling operations can be undertaken when clusters including CPUs and GPUs are used. When less specialised resources are exploited, the predictive refactoring can identify new possible configurations for the system that can lead to an improvement.
- *POET (PerfOrmance opTimization)*: In case an application deployment requires the usage of HPC clusters, POET can optimise the application code or the corresponding execution containers based on the QoS objectives and technological requirements expressed by the user.

Following the guidelines contained in this document, the reader will exploit the Smart IDE to modify an existing deployment model, the features provided by FindIaC Bug to identify and fix problems, the deployment, orchestration and provisioning mechanisms offered by PESTO, as well as the monitoring features made available by RIDER. More advanced features offered by RIDER itself and by POET are not proposed here for the sake of simplicity.

The following of this document is organized as follows: Section 2 presents an overview of the exercise we propose to the readers; Section 3 lists the reference training material; Section 4 presents the structure of the application to be deployed. Section 5 provides guidelines on how to configure the SODALITE SaaS testbed and the IDE that will be executed as a desktop application; Section 6 provides the details on the steps to be accomplished and, finally, Section 7 concludes the document.

2. Exercise to be accomplished

The exercise we propose aims at providing to the reader a short introduction to the basic features offered by SODALITE. For a complete overview of SODALITE, the reader should refer to the general documentation and demonstration examples available on the SODALITE website.

The exercise consists of the following steps:

- Short training: the reader can refer to the video material listed in Section 3 to get to know about how to use SODALITE.
- Exercise A - Inspection and modification of a deployment model for a Machine Learning application: this task aims at highlighting a small set of the features offered by the SODALITE Smart IDE to support the modeling activities.
- Exercise B - Solution of deployment issues: the SODALITE framework helps the user in generating IaC (Infrastructure as Code) from Abstract Application Deployment Model (AADM). In this step the user will take advantage of the offered help dialog to inspect and solve the problems.
- Exercise C - Execution of the Machine Learning application: in this phase the user will verify that the application has been deployed properly and works as expected.
- Exercise D - Undeployment and clean up.
- Finalization: we will ask you to fill in a form to give us some feedback on your experience.

3 Reference material

Before following the exercise described in this document, we suggest the reader to watch the following videos:

- [SODALITE IDE AADM Edition Video](#)
- [SODALITE IDE AADM Deployment Video](#)

An important source in case of difficulties is the [SODALITE IDE user's manual](#).

4. The application to be deployed

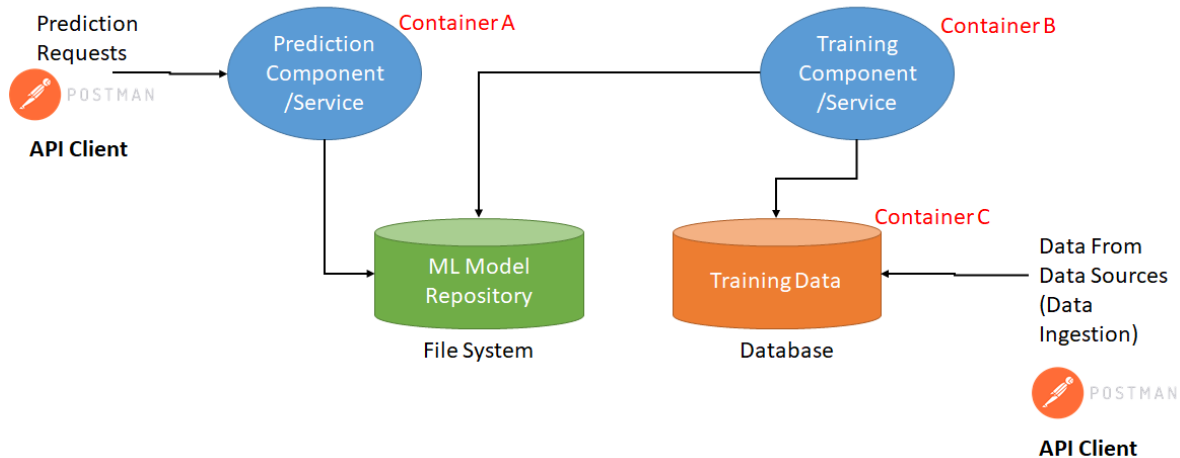


Figure 1. Distributed ML Application

Figure 1 shows the ML application. It consists of 1) a component that stores training data in a database (SQLite), 2) a component that trains a machine learning model 3) a repository that stores trained machine learning models, 4) a component that makes predictions/inferences based on the trained models. Figure 2 illustrates the deployment model of this application.

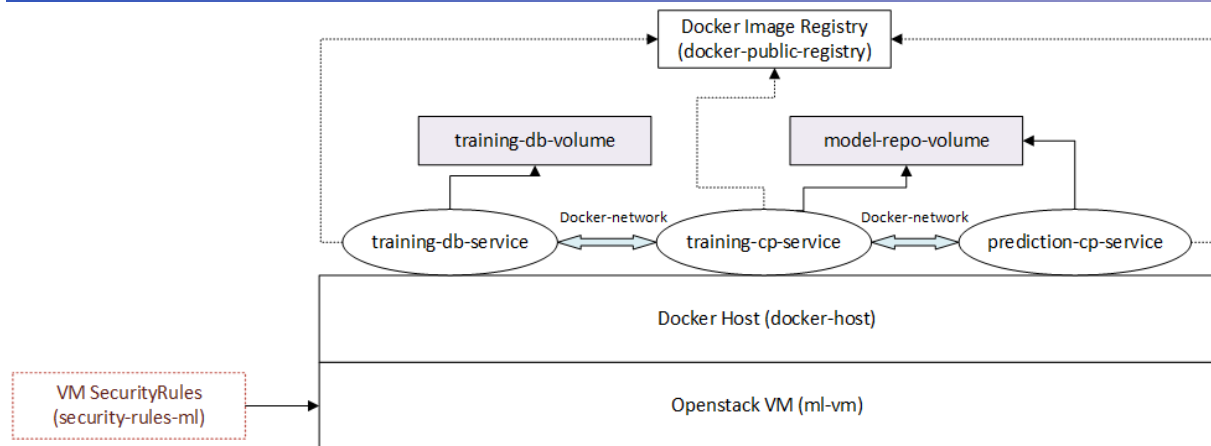


Figure 2. Deployment model of the ML Application

All three services (prediction-cp-service, training-cp-service, and training-db-service) are Dockerized components (i.e., Docker containers). They are executed and managed by a Docker engine (docker-host) hosted in an Openstack VM (ml-vm). The VM is protected by the security configuration (security-rules-ml). The Docker containers are connected via Docker Network (docker-network). The Docker images are hosted in Docker Hub (docker-public-registry). These images are pulled and executed to create the container instances. Docker volumes are used to store training data (training-db-volume) and machine learning models (model-repo-volume).

The following are the configuration parameters for the application components in the above deployment model.

Docker Container: training-db-service

type	docker/sodalite.nodes.DockerizedComponent
properties	<ul style="list-style-type: none">• image_name: 'indikakumara/trainingdbapi:0.0.1'• registry_url: "registry.hub.docker.com"• restart_policy: "always"• ports: ['5000:5000']• alias: 'trainingdb_service'• docker_network_name: get_property (docker-network.name)• env (environment variables) DB_URL: 'sqlite:///features.db'
requirements	<i>host as docker-host</i> <i>network as docker-network</i> <i>registry as docker-public-registry</i>

Docker Container: training-cp-service



type	docker/sodalite.nodes.DockerizedComponent
properties	<ul style="list-style-type: none">• image_name: 'indikakumara/trainingcpapi:0.0.1'• registry_url: "registry.hub.docker.com"• restart_policy: "always"• ports: ['5001:5000']• alias: 'training_service'• docker_network_name: get_property (docker-network.name)• volumes model-repo: '/usr/src/myapp/models'• env (environment variables) MODEL_REPO: '/usr/src/myapp/models' TRAIN_DB_API: 'http://trainingdb_service:5000/training-db/diabetes''
requirements	<i>host as docker-host network as docker-network registry as docker-public-registry dependency with training-db-service dependency with model-repo-volume</i>

Docker Container: prediction-cp-service

type	docker/sodalite.nodes.DockerizedComponent
properties	<ul style="list-style-type: none">• image_name: 'indikakumara/predictionapi:0.0.1'• registry_url: "registry.hub.docker.com"• restart_policy: "always"• ports: ['5002:5000']• alias: 'prediction_service'• docker_network_name: get_property (docker-network.name)• volumes model-repo: '/usr/src/myapp/models'• env (environment variables) MODEL_REPO: '/usr/src/myapp/models'
requirements	<ul style="list-style-type: none">• <i>host as docker-host</i>• <i>network as docker-network</i>• <i>registry as docker-public-registry</i>• <i>dependency with training-service</i>• <i>dependency with model-repo-volume</i>

5. Configuration of the SODALITE platform

User registration

To create a new account in SODALITE platform follow the link

<https://keycloak.public-testbed.sodalite.eu/auth/realms/SODALITE/account/#/>

and click the **Sign In** button on the right. Press **Register** at the bottom of the form ([Figure 1](#)).

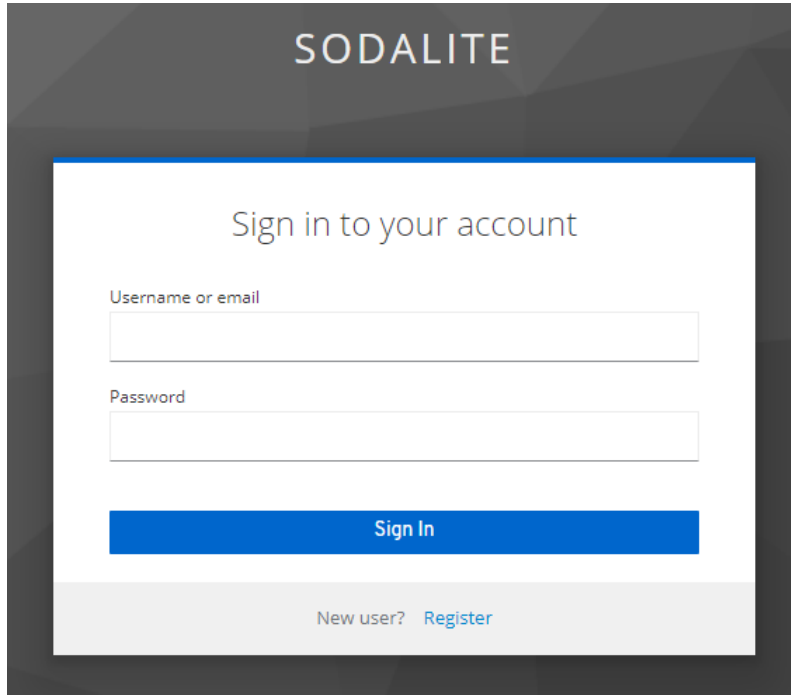
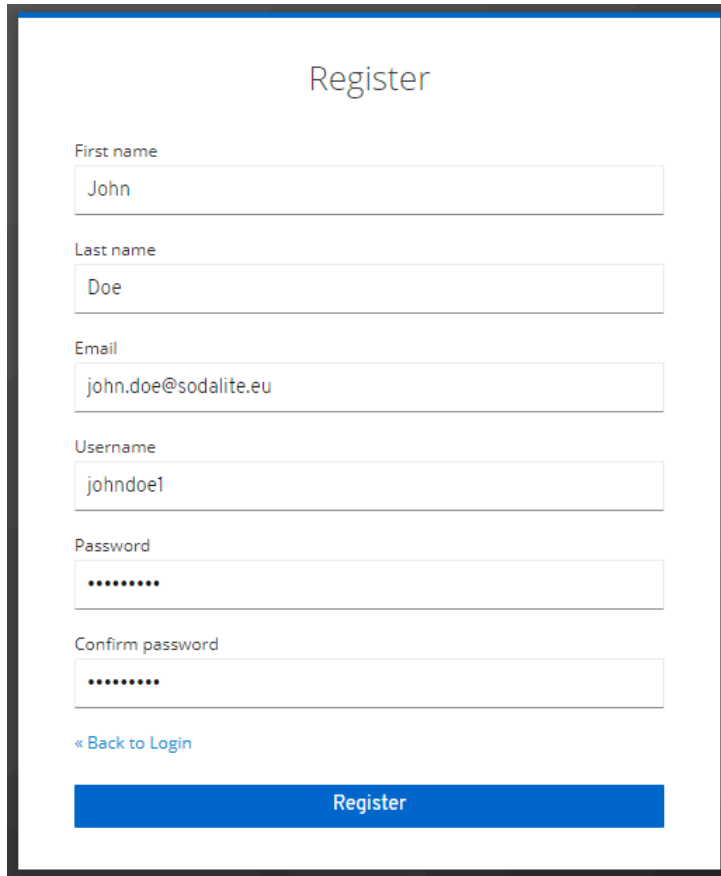
The image shows a web interface for the SODALITE platform. At the top, the word "SODALITE" is displayed in a large, white, sans-serif font against a dark, geometric background. Below this, a white rectangular box contains the text "Sign in to your account" in a medium-sized font. Underneath, there are two input fields: the first is labeled "Username or email" and the second is labeled "Password". Both fields are empty and have a light gray border. Below the password field is a prominent blue button with the text "Sign In" in white. At the bottom of the white box, there is a link that says "New user? Register", where "Register" is in blue and "New user?" is in gray.

Figure 1

In the registration form ([Figure 2](#)) please fill in your personal data, email, desired login and password. For login please use alphanumeric characters (only letters and numbers). Currently email verification is disabled in SODALITE, so no confirmation email would be sent to the provided address and your account would be enabled after the form is submitted.



The screenshot shows a web form titled "Register". It contains several input fields: "First name" with the value "John", "Last name" with the value "Doe", "Email" with the value "john.doe@sodalite.eu", "Username" with the value "johndoe1", "Password" with masked characters "*****", and "Confirm password" with masked characters "*****". Below the fields is a link "« Back to Login" and a blue "Register" button.

Figure 2

Use the username and password you provided for IDE configuration. Your username would be also used as a private module name used in the modelling phase to store private RMs and AADMs.

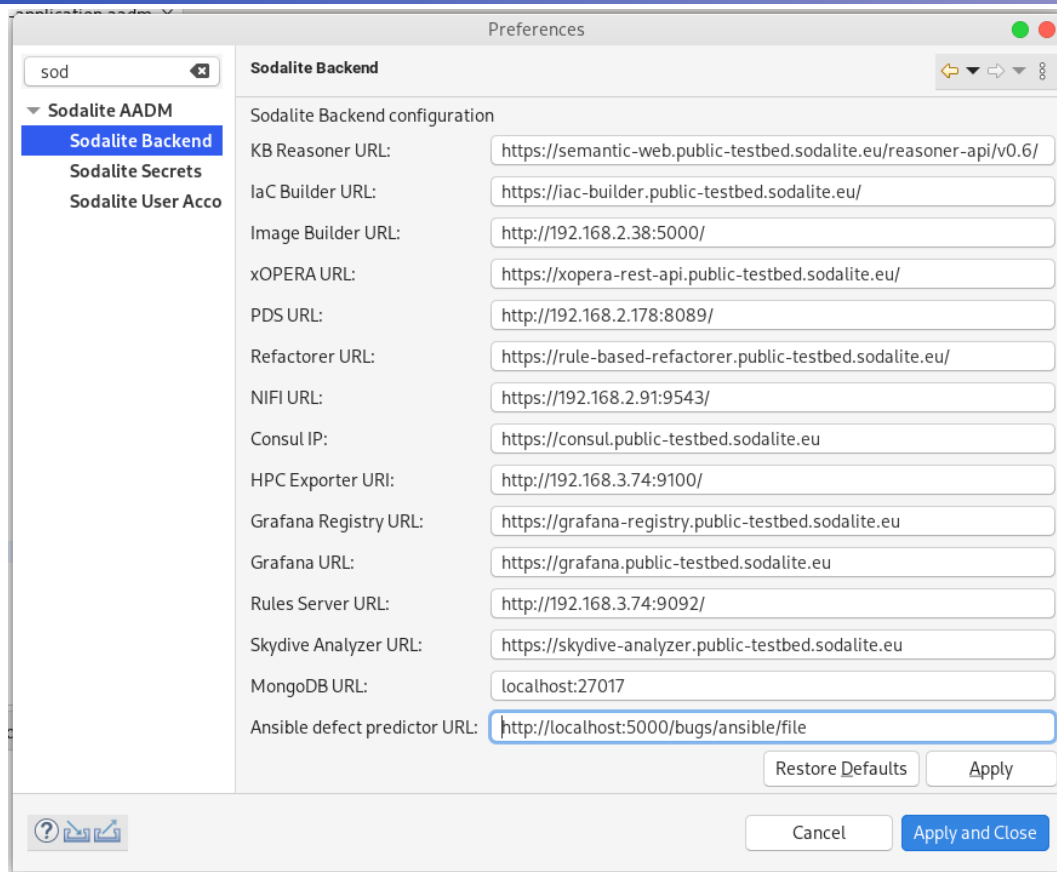
Installation and configuration of the IDE

1- Requirements

Install Sodalite IDE into an Eclipse 2021-09 following the instructions contained in:
<https://github.com/SODALITE-EU/ide/blob/master/README.md>

2- Configure Sodalite backend connection

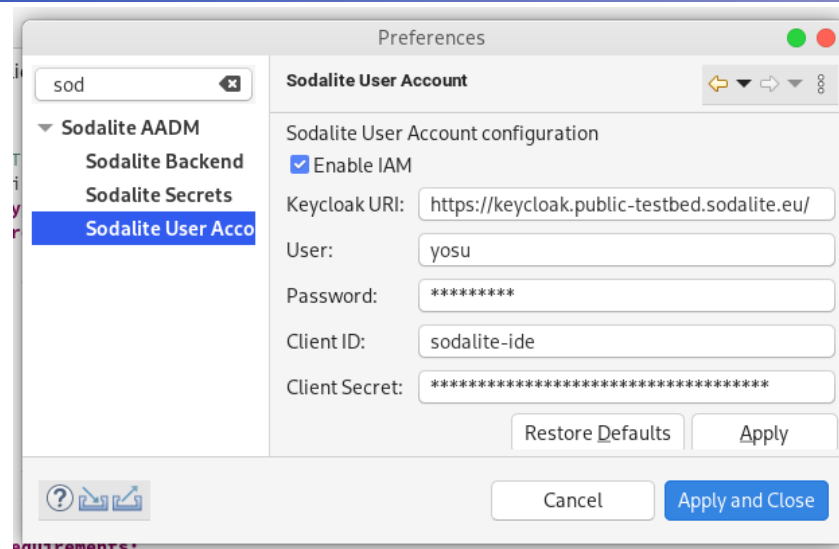
Open IDE preference page: menu **Window/Preferences** (**Eclipse/Preferences** for Mac users). Search for Sodalite in search text. Click on Sodalite Backend



Edit the URLs for each of the following public Sodalite Backend services (see the list below):
KB Reasoner URL, IaC Builder URL, xOpera URL, Refactorer URL, Consul IP, Grafana Registry URL, Grafana URL.

IDE is connected to the public SODALITE platform that requires IAM communication. To configure it, open the *Sodalite User Account* preference page, enable IAM, and provide the following information (contact the SODALITE administrator for it):

- Keycloak URI: <https://keycloak.public-testbed.sodalite.eu/>
- User/Password: <use your Keycloak credentials>
- Client ID: sodalite-ide
- Client secret: 6a2d840b-9e72-47af-97b4-13226855111a



If AIM is required and not properly set in this page, during the usage of the IDE you will get notified with errors anytime you use content-assistance for model edition or when you try to browse models in the KB or try to save or deploy them.

3 - Configuration for public testbed:

Keycloak URI: <https://keycloak.public-testbed.sodalite.eu>

Client Secret: 6a2d840b-9e72-47af-97b4-13226855111a

KB Reasoner URL: <https://semantic-web.public-testbed.sodalite.eu/reasoner-api/v0.6/>

laC Builder URL: <https://iac-builder.public-testbed.sodalite.eu/>

Image Builder URL: N/A

xOpera URL: <https://xopera-rest-api.public-testbed.sodalite.eu/>

PDS URL: N/A

Refactorer URL: <https://rule-based-refactorer.public-testbed.sodalite.eu/>

NIFI URL: N/A

Consul IP: <https://consul.public-testbed.sodalite.eu>

Grafana Registry URL: <https://grafana-registry.public-testbed.sodalite.eu>

Grafana URL: <https://grafana.public-testbed.sodalite.eu>

Rules Server URL: N/A

Skydive Analyzer URL: <https://skydive-analyzer.public-testbed.sodalite.eu>

HPC Exporter URL: <http://hpc-exporter>

MongoDB URL: N/A

Ansible defect predictor URL: N/A

6. Experiments

Modeling phase

Read section 1 of the [IDE user's manual](#) for more information about the AADM Modeling with the IDE.

Read section 1.5 on [IDE user's manual](#) to learn how to model Abstract Application Deployment Models (AADM). Read section 1.3 to learn how to create a project in your IDE workspace. The



following instructions assume you are familiarised with Eclipse IDE usage. If not, learn Eclipse usage from the Internet.

Exercise A - modelling an abstract application deployment model

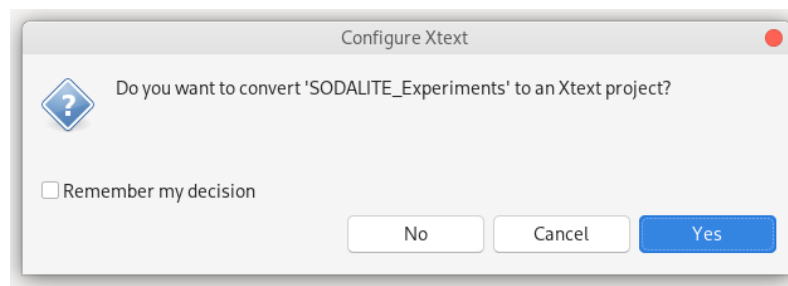
We will create an AADM for our ml-application (see section *The application to be deployed*). But instead of creating this model from scratch, let's start from one that already contains some ancillary application components. Download the ml-application2_template.aadm from:

https://raw.githubusercontent.com/SODALITE-EU/exercises/main/ml-application2_template.aadm

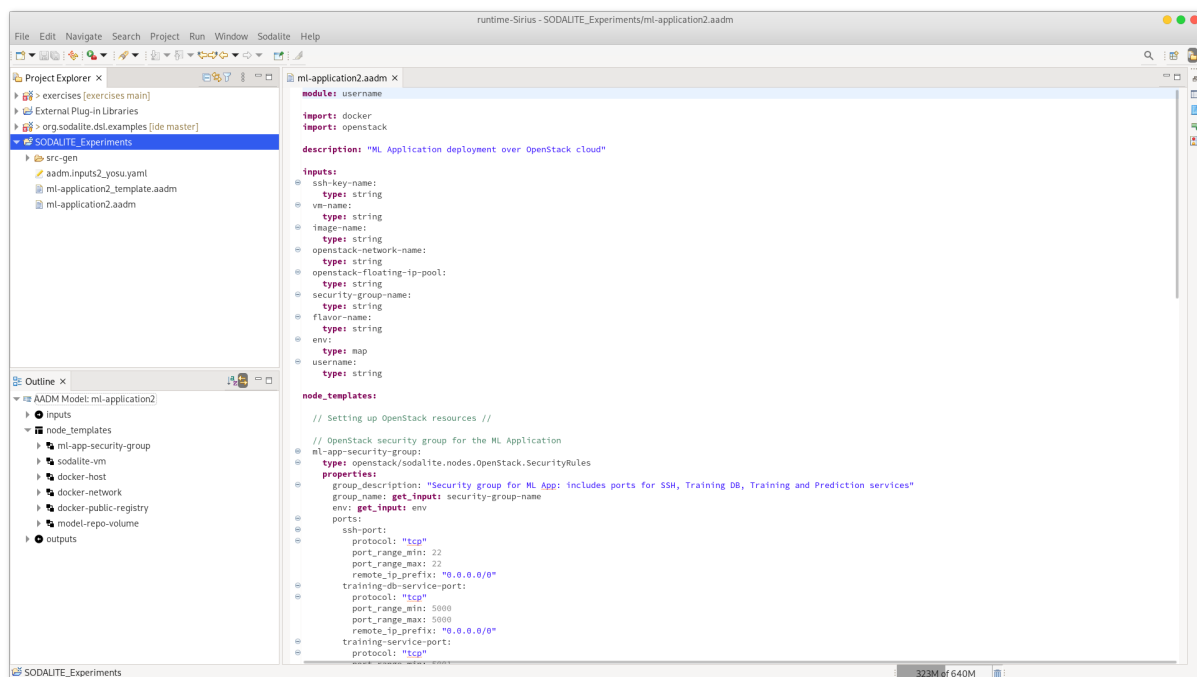
and save it into the project you have recently created in your IDE workspace. Rename it or copy it in the same folder as *ml-application2.aadm*.



To open the project, double click on it. If prompted to convert your project to Xtext, select “yes”



The model will appear on the AADM editor

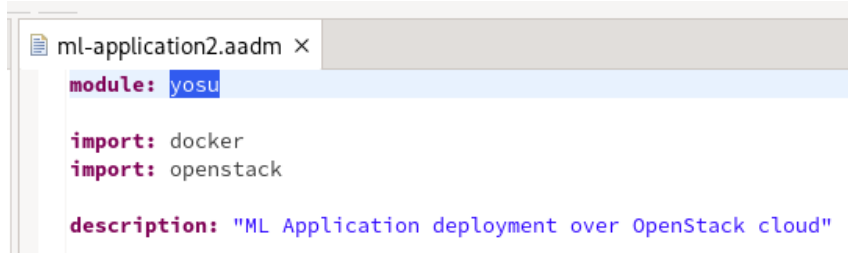


We propose that you complete this ml-application AADM. You have to create node templates for the following m-application components:

- training-db-service
- training-service
- prediction-service

using the information collected in the tables of section *The application to be deployed*.

IMPORTANT: Modify the *module* value and give it the *username* you registered in Keycloak. This sets that your model will be stored into the KB associated with your *username* namespace, for which you have granted read/write permissions, and avoids your model overrides other experimenters' models.



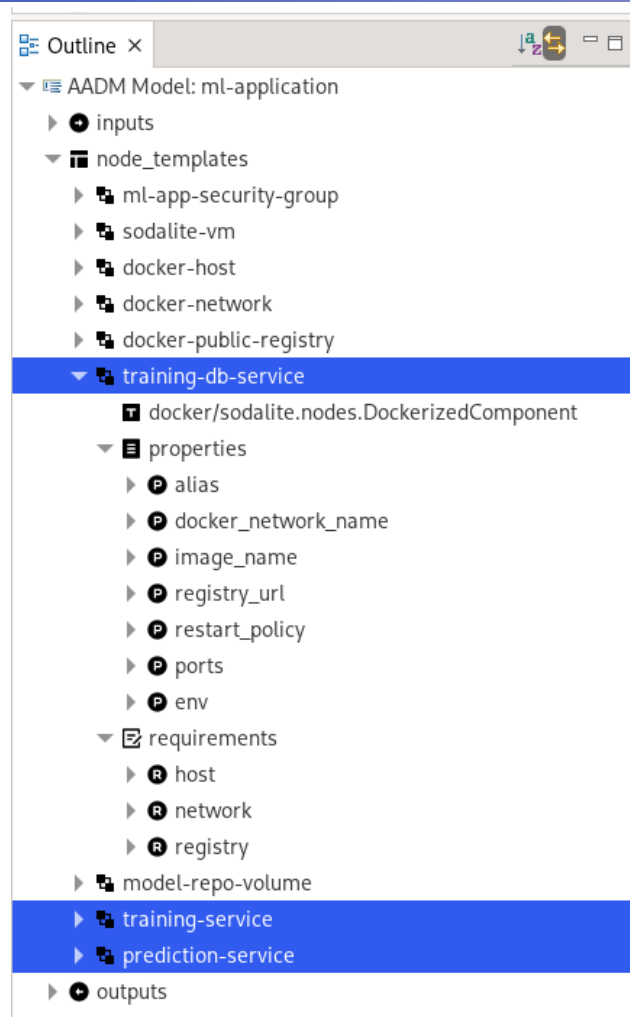
```
ml-application2.aadm x
module: yosu

import: docker
import: openstack

description: "ML Application deployment over OpenStack cloud"
```

IMPORTANT: Once you have declared a new module, all references within the model to entities (such as requirement nodes, or properties in the `get_property` function need to be escaped with that module name.

Once you are done, your AADM outline view should look like:



Once you are done, save your model into the KB and validate it from possible detected semantic errors. To learn how to do this, **read sections 7 and 8 in the [IDE user's manual](#)**.

Once your model have been successfully stored into the KB (this also implies it is free of semantic errors), you can deploy your model (**read section 9 in the [IDE user's manual](#)**)

Inputs:	Get secret
env	<input type="checkbox"/> <div>os_env: OS_USERNAME: demo OS_PASSWORD: sodalite-demo-2022 OS_AUTH_URL: https://fra1.citycloud.com:5000 OS_USER_DOMAIN_NAME: CCP_Domain_42461</div>
flavor-name	<input type="checkbox"/> 1C-4GB-20GB
image-name	<input type="checkbox"/> "Ubuntu 20.04 Focal Fossa 20200423"
openstack-floating-ip-pool	<input type="checkbox"/> ext-net
openstack-network-name	<input type="checkbox"/> sodalite-net
security-group-name	<input type="checkbox"/> ml-app-security
ssh-key-name	<input type="checkbox"/> xopera-keypair
username	<input type="checkbox"/> ubuntu
vm-name	<input type="checkbox"/> yosu

Give a deployment name and leave the other fields with default values (as in above image)
The inputs you need to feed into the wizard form are the following:

vm-name: **<user vm name>**
security-group-name: **<user security group name>**
security-groups: **<user security group name>**,default
ssh-key-name: xopera-keypair
image-name: "Ubuntu 20.04 Focal Fossa 20200423"
openstack-network-name: sodalite-net
openstack-floating-ip-pool: ext-net
flavor-name: 1C-4GB-20GB
username: ubuntu
env:
os_env:



OS_USERNAME: demo
OS_PASSWORD: sodalite-demo-2022
OS_AUTH_URL: <https://fra1.citycloud.com:5000>
OS_USER_DOMAIN_NAME: CCP_Domain_42461
OS_PROJECT_DOMAIN_NAME: CCP_Domain_42461
OS_REGION_NAME: Fra1
OS_PROJECT_NAME: "Default Project 42461"
OS_TENANT_NAME: "Default Project 42461"
OS_AUTH_VERSION: 3
OS_IDENTITY_API_VERSION: 3

As *vm-name* and *security-group-name* input values give a **unique value**. **Note:** *security-groups* input value should also contain the exact value of *security-group-name* input with the "default" security group separated by comma, e.g.:
security-group-name: **security-group-user123**
security-groups: **security-group-user123,default**

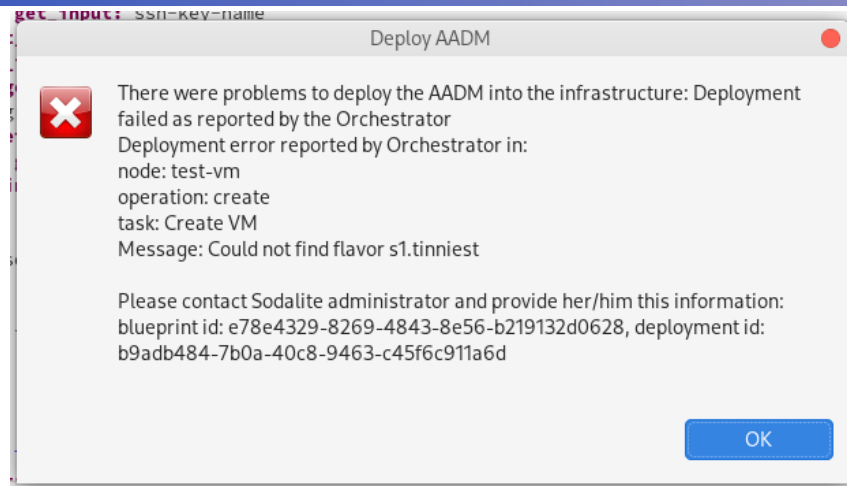
During the deployment process (and after it anytime at runtime) you can check the deployment status in the IDE Governance View (**see section 1.10 in the [IDE user's manual](#)**)

Important: The users must save/remember the outputs received from the IDE after the successful deployment of their applications. The output includes the **IP address** of the VM that the application was deployed on. In order to test the application, the IP address of the VM is needed.

Deployment phase and error handling

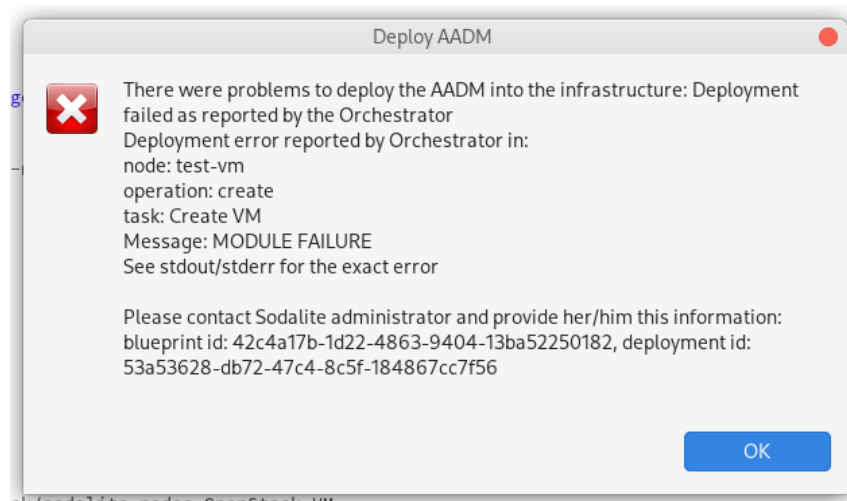
During the deployment, there is a possibility of a deployment failure, e.g. when a user specified incorrect input value that causes Ansible playbook to fail. In this case, if a problem executing Ansible playbooks occurs, IDE would provide a report indicating:

1. execution of which node of the deployed AADM resulted in an error
2. name of the operation that causes the error
3. name of the task in the Ansible playbook implementing the above mentioned operation
4. error message



This is a short description of the error. More detailed information on the problem can be obtained from the **stderr** field of the deployment status in the IDE Governance View.

In some cases deployment issue reports are indescriptive or are caused by service malfunctions and thus cannot be resolved by the user. In this situation please contact SODALITE support, providing your deployment id.

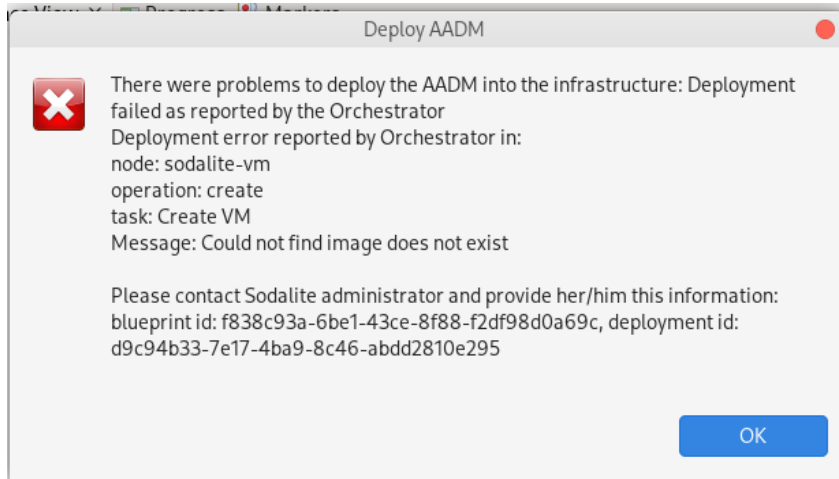


Exercise B - testing error handling.

SODALITE IDE provides the "Deployment Resume" functionality to recover some deployment failures caused by invalid user inputs. This functionality resumes the deployment from the failed component. As part of this exercise, you will be intentionally providing incorrect inputs, observe the errors, fix the inputs and try to resume the deployment from the failed component. For this exercise, the following steps should be executed:

1. Try deploying AADM, created before in the Modelling phase as described previously.
2. Fill in the input values provided in the Modelling phase.
3. In **image-name** input textbox fill in an erroneous value (e.g "does not exist")
4. Start the deployment.

- Wait for deployment to fail and observe the error report.



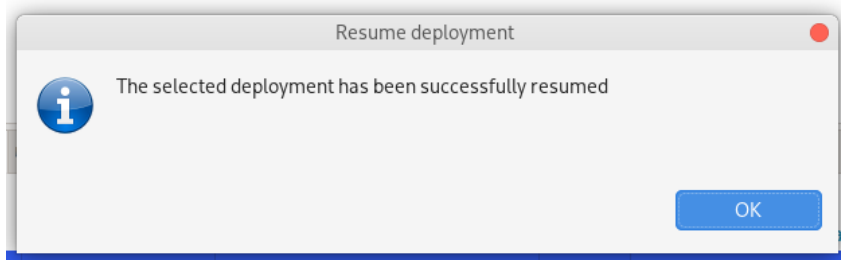
- Using IDE Governance View , select the failed deployment and resume it as described in the [IDE user's manual](#)

Id	Name	AADM id	Module	URL	Timestamp	State	Version id
f838c93a-6be1-43ce-8f88-f2df98d0a69c	ml-application.aadm	AADM_iutp14jtr12k15qq0kng66ucu5	yosu	https://gitlab.com/sodalite.xopera/gitDB_f838c93a-6be1	2022-02-11 09:14:06	-	v1.0
d9c94b33-7e17-4ba9-8c46-abdd2810e295	ml-app-yosu	-	-	-	2022-02-11 09:14:07	failed	-

- Put the correct value "Ubuntu 20.04 Focal Fossa 20200423" into the **image-name** input textbox.

- Resume the deployment.

9. Wait for deployment to succeed.



Execution phase

During the execution phase, the application is supposed to accomplish its specific tasks. To check whether our example application works properly you can go through the steps described below.

Exercise C - checking that the machine learning application is working properly

The users can use the API clients such as Postman, Insomnia, and cURL for testing services.

<https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdgghehccddcbncdddomop?hl=en>
<https://insomnia.rest/>

- **Create a Table at the train-db services**

```
curl --request POST \  
  --url http://IP_VM:5000/training-db/diabetes \  
  --header 'Content-Type: application/json' \  
  --data '{  
    "columns": [  
      "ntp",  
      "pgc",  
      "dbp",  
      "tsft",  
      "si",  
      "bmi",  
      "dpf",  
      "age",  
      "class"  
    ]  
  }'  
,
```

Note: **IP_VM** is '*public_ip*' attribute in the outputs you received from the SODALITE IDE after the successful deployment of your application.



```
POST http://86.107.243.229:5000/training-db/diabetes Send 200 OK 6 Hours Ago
JSON Auth Query Header Docs Preview Header
1 {
2   "columns": [
3     "ntp",
4     "pgc",
5     "dbp",
6     "tsft",
7     "si",
8     "bmi",
9     "dpf",
10    "age",
11    "class"
12  ]
13 }
14 }
```

- **Populate the table with Data**

Send a PUT request to the URL http://IP_VM:5000/training-db/diabetes with the data in the following JSON file as the message body

https://github.com/IndikaKuma/DE2021/blob/main/lab3/test_resources/training-db/train_data.json

```
PUT http://86.107.243.229:5000/training-db/diabetes Send 200 OK 6 Hours Ago
JSON Auth Query Header Docs Preview Header
1 [
2   {
3     "ntp": 6,
4     "pgc": 148,
5     "dbp": 72,
6     "tsft": 35,
7     "si": 0,
8     "bmi": 33.6,
9     "dpf": 0.627,
10    "age": 50,
11    "class": 1
12  }
13 ]
```

- **Train the ML model**

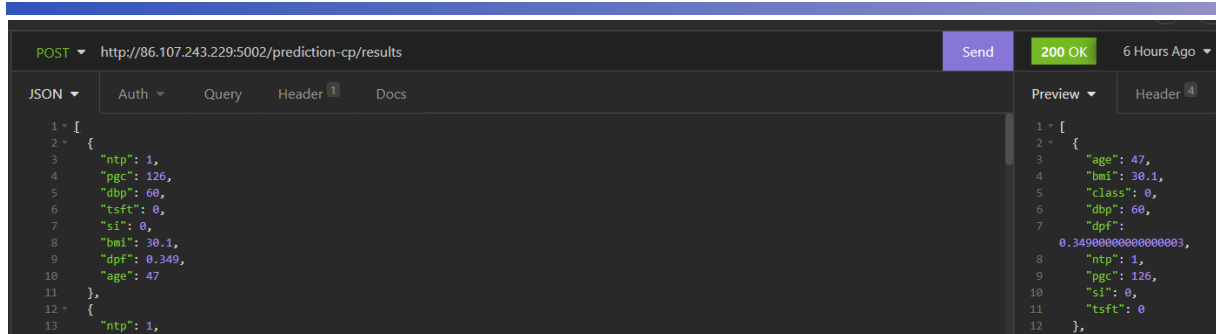
```
curl --request POST \
  --url http://IP_VM:5001/training-cp/model \
  --header 'Content-Type: application/json'
```

```
POST http://86.107.243.229:5001/training-cp/model Send 200 OK 6 Hours Ago
Body Auth Query Header Docs Preview Header
1 {
2   "accuracy":
3     0.7566433548927387,
4   "loss":
5     0.492754191160202
6 }
```

- **Make predictions**

Make a POST request to <http://VMIP:5002/prediction-cp/results> with the data in the following JSON file as the message body

https://github.com/IndikaKuma/DE2021/blob/main/lab3/test_resources/training-db/predict_data.json



Visualize the monitoring dashboard (Yosu)

(info at the VM level)

Undeploy the ML application

See instructions in section 1.10 of the [IDE user's manual](#) to undeploy your application and release the allocated resources.

7. Exercise finalization and conclusion

Thank you very much for getting to this point! We kindly ask you to fill in this form to give us some feedback on your experience <https://forms.office.com/r/MrZjR3t5t4>. Data is collected in a completely anonymous form and will be used only for the purpose of assessing the quality of our current results.

We do hope this experiment has triggered your interest in SODALITE. Please check our website (<http://sodalite.eu>) for further information and material.