Deliverable D3.5

Infrastructural code generation – v2

Editor(s):	Lorenzo Blasi
Responsible Partner:	HPE
Status-Version:	Final–v1.0
Date:	01.12.2022
Distribution level (CO, PU):	PU

Project Number:	101000162
Project Title:	PIACERE

Title of Deliverable:	Infrastructural code generation – v2
Due Date of Delivery to the EC	30.11.2022

Workpackage responsible for the Deliverable:	WP3 - Plan and create Infrastructure as Code
Editor(s):	Lorenzo Blasi (HPE)
Contributor(s):	Laurentiu Niculut (HPE CDS), Debora Benedetto (HPE CDS), Lorenzo Blasi (HPE)
Reviewer(s):	Radosław Piliszek (7BULLS)
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP4, WP5

Abstract[1]:	These deliverable presents the advancements of Task T3.4 made in year 2. It comprises both an updated version of the software prototype [KR3] and a Technical Specification Report. The document includes the technical design of the current version of the ICG, installation instructions and user manual.
Keyword List:	Code generation, Infrastructure as Code
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

[1] This is the same deliverable description provided in the DoA

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	30.09.2022	Definition of the ToC	Lorenzo Blasi, HPE
v0.2	26.10.2022	Updated ToC according to the new common template	Benedetto Debora, HPECDS
v0.3	28.10.2022	Added first draft	Benedetto Debora, HPECDS
v0.4	28.10.2022	Added chapter integration	Niculut Laurentiu, HPECDS
v0.5	16.11.2022	Updated sections on scenarios, user manual and template library	Benedetto Debora, Niculut Laurentiu, HPECDS, Lorenzo Blasi, HPE
v0.6	17.11.2022	Version ready for review	Benedetto Debora, Niculut Laurentiu, HPECDS, Lorenzo Blasi, HPE
v0.7	28.11.2022	Corrections done as required from review	Benedetto Debora, Niculut Laurentiu, HPECDS
v1.0	01.12.2022	Ready for submission	Juncal Alonso, TECNALIA

Table of contents

List of tables

List of figures

Terms and abbreviations

AWS	Amazon Web Services
DevOps	Development and Operation
DoA	Description of Action
DOML	DevOps Modelling Language
GA	Grant Agreement to the project
IaC	Infrastructure as Code
ICG	Infrastructural Code Generator
IDE	Integrated Development Environment
IEM	IaC Executor Manager
KR	Key Result
MC	Model Checker
IOP	IaC Optimization
IR	Intermediate Representation
JSON	JavaScript Object Notation
VT	Verification Tool
PRC	PIACERE Runtime Controller

Executive Summary

This deliverable describes the second iteration of the PIACERE Infrastructural Code Generator (ICG), developed in Task 3.4. The document reports updates to the ICG component developed in the second year of the project, its functional and technical description, and how it can be installed and used.

This release of ICG greatly improves over the first version by implementing the Parser for DOML models, by packaging the component into a container image offering a REST interface and by tightly integrating with the other design-time PIACERE components, especially with the IDE. The deliverable reports the functional and technical updates to the ICG component, and describes the output provided by the Code Generator, with details on the configuration files that drive the execution of the generated IaC code. Another improvement to the ICG is in its library of templates, which in this release has been expanded and better structured. This version of the ICG is compatible with v2 of DOML and it was aligned with each update of DOML during the last year. A final important novelty in this release is that the ICG source code has been release in open source, with the Apache 2.0 license.

The next version of the ICG will further enhance the template library to offer more complete code generation capabilities, both for improving the support of DOML elements and possibly for supporting other target platforms, depending on Use Case requirements.

1 Introduction

1.1 About this deliverable

This deliverable is a second version of the deliverable about the Infrastructural Code Generator (ICG) component of PIACERE. The document reports about updates to the ICG component developed in the second year of the project, how the new version fits into the overall PIACERE framework, and how it can be installed and used. This release of ICG greatly improves over the first version (D3.4 [1]) by implementing the Parser for DOML models, by packaging the component into a container image offering a REST interface and by tightly integrating with the other design-time PIACERE components, especially with the IDE.

1.2 Document structure

Section 2 of the document explains what has changed in the second release of ICG, describes its architecture and functionalities and provides some high-level usage scenarios; this section also offers a complete technical description of the component, including a description of each subcomponent and details about the structure and purpose of the produced output files.

The D3.5 deliverable is a software deliverable, therefore this document also provides details about the released software in section 3: how it is structured, how it can be obtained, installed and used, plus licensing information. In section 4 there are the conclusions and some indications about the next steps.

2 Implementation

2.1 Changes in v2

The second version of the ICG has been released in M24 and it contains several improvements with respect to ICG v1 released in M12 (D3.4 [1]). The ICG is now fully functional, all the components now work as required, the component that has seen more improvements being the ICG Parser that was entirely developed during this year. The component is now packaged into a Docker image and supports both command line execution and http-based Rest API calls using the Open API interface definition. It is integrated in the PIACERE Framework, it correctly parses DOML v2, is called by the PIACERE IDE and when required also by the PRC and it generates IaC code compatible with the IEM requirements for execution. The template library was reorganized and extended to support new providers, such as Openstack, and resources like security groups or applications like Nginx. The ICG has also been released in Open Source with the Apache 2.0 license. Thus, the following sections of this document have been accordingly updated to reflect those changes in the functional and technical description of the ICG (sections 2.2 and 2.3) as well as in the delivery and usage of this release.

Requirements in Table 1 have been updated and most of them fulfilled: REQ96 is achieved thanks to the implementation of the ICG DOML Parser, REQ29 has been discarded and replaced by REQ100 which is achieved and will be furthermore extended in the next ICG version, REQ41 has been discarded and replaced by REQ110.

2.2 Functional description

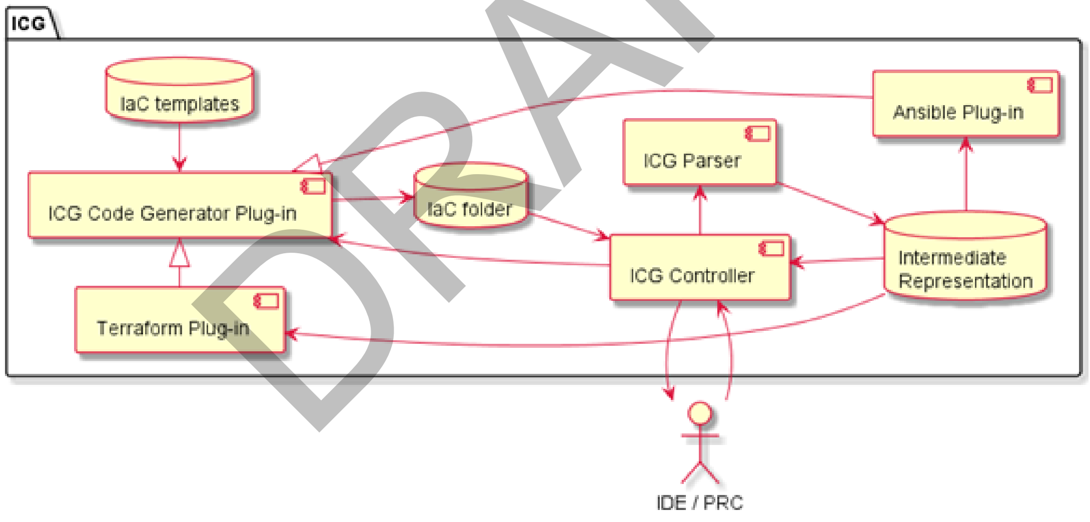


Figure 1: ICG component representation

The ICG version 2 evolved from a command line application into a microservice. Thus, the tool is aligned with the PIACERE microservices framework architecture: it is invoked through the REST API and takes as input the DOML model in the XML format for the generation of the Infrastructure as Code (IaC) which is returned in an archive (zip) format.

Figure 1 shows the internal ICG architecture, which has the same components declared in the D3.4 [1] section 2.1. It is slightly revised due to the adoption of the new microservice architecture. Below, we report the updates affecting each internal component.

ICG Controller is the main component and in this new version it is invoked by the PIACERE IDE or PIACERE PRC through its REST API. It reads the parameters and controls the internal flow between the other components of the ICG.

ICG DOML Parser is now implemented. It is activated by the Controller. It parses the input DOML model to produce an Intermediate Representation. The Parser reads the input model in XML format, a DOML representation that is called DOMLx. The current version of the Parser is compatible with DOML v2.

The **Intermediate Representation** is not changed, see D3.4 [1] section 2.1.

ICG Code Generator is not changed, see D3.4 [1] section 2.1.

The **Templates Library** has been extended to support more resources from more cloud providers. The list of the new resources can be found in Section 2.3.2.5.

The **IaC folder** is a new ephemeral storage to host the IaC files and configuration produced during the generation process. This folder will be the output of the ICG and will be described in Section 2.3.3.

The ICG main functionalities are the same as in D3.4 [1], and we recap them here:

- F1. Read the input DOML model to extract all the needed information.
- F2. Generate executable code for selected IaC languages.
- F3. Provide enough extensibility to support the DOML extension mechanism [KR4]
- F4. Provide enough extensibility to generate code for new IaC languages
- F5. Generate IaC code that supports different cloud platforms

In this second release, the functionality F1 is implemented, F2 and F5 have been extended and the others are still partially implemented.

Below, Table 1 is reporting the updates about the coverage of each ICG requirements indicated in deliverable D2.2 in relation to the functionalities.

Table 1 KR3 - ICG Requirements

Funct.	Req ID	Description	Status	Requirement Coverage at M24
F1	REQ96	ICG must be able to read DOML language.	Achieved	ICG can parse the DOML v2.
F2	REQ31	ICG should provide verifiable and executable IaC generated from DOML for selected IaC languages (e.g., TOSCA/Ansible/Terraform).	Achieved	Already implemented in v1, see previous deliverable (D3.4 [1])
F2	REQ77	ICG may generate IaC code for different supported/target tools according to the required DevOps activity (as listed in REQ76).	Partially Achieved	There are no changes from v1 (D3.4 [1]), deployment and configuration are available, orchestrations is not yet implemented
F3	REQ41	The IDE should be extensible through the plugin mechanism. Not only to	Discarded	This requirement should be fulfilled by the KR2.

		support PIACERE assets (ICG, VT) but also for third party collaborators.		The new REQ110 assigned to KR3 replaces this requirement.
F3	REQ110	ICG should provide enough extensibility to: comply with the DOML extension mechanism; be capable of integrating new IaC languages	Partially Achieved	The ICG is setup to be extendable, the full extension mechanism though is yet to be implemented.
F4	REQ110	ICG should provide enough extensibility to: comply with the DOML extension mechanism; be capable of integrating new IaC languages	Partially Achieved	Regarding this functionality there are no changes in comparison with v1 (D3.4 [1])
F5	REQ29	DOML should support the modelling of VM provisioning for different platforms such as (OpenStack, AWS) for canary and production environments.	Discarded	This requirement should be fulfilled by the KR1. The new REQ100 assigned to KR3 replaces this requirement.
F5	REQ100	ICG should generate IaC code that supports different cloud platforms.	Achieved	ICG can generate IaC code (Virtual Machine, Network, Security Group, Ssh Keys) for AWS, Azure and OpenStack platforms.

2.2.1 ICG Scenarios

To better define the functionalities and requirements of the ICG a few end user scenarios have been defined. In these scenarios the process of using the ICG will be explained using the Cucumber/Gherkin notation [2]. The scenarios below are complementary to the ones from the D2.2 deliverable, going more in detail on the available options.

Table 2 KR3 – ICG Scenarios

Phase	Title	Scenario	Description	Requirements
design-time	Generation of infrastructure provisioning code	Given a verified DOML model containing the infrastructure definition When a user navigates to the DOML document And right-clicks on it And selects "PIACERE" And selects "Generate IaC code" Then a compressed folder containing the infrastructural IaC code is generated	From IDE GUI the user can generate IaC files	REQ96, REQ31, REQ77, REQ110

design-time	Generation of infrastructure provisioning code for multiple providers	Given a verified DOML model containing the infrastructure definition And two different providers between the supported ones And a user selects one of the two as active When a user navigates to the DOML document And right-clicks on it And selects "PIACERE" And selects "Generate IaC code" Then a compressed folder containing the infrastructural IaC code for the active provider is generated	From IDE GUI the user can generate Infrastructural code for different providers	REQ96, REQ31, REQ77, REQ100, REQ110
design-time	Generation of bundled provisioning and configuration code	Given a verified DOML model containing the infrastructure and coherent application definition When a user navigates to the DOML document And right-clicks on it And selects "PIACERE" And selects "Generate IaC code" Then a compressed folder containing the infrastructural and subsequent application configuration IaC code is generated	From IDE GUI the user can generate code that can in a single deploy provision the infrastructure and configure the related application	REQ96, REQ31, REQ77, REQ110
design-time	Generation of monitoring and security agents configuration code	Given a verified DOML model containing the infrastructure definition And at least a virtual machine is defined When a user navigates to the DOML document And right-clicks on it And selects "PIACERE" And selects "Generate IaC code" Then a compressed folder is generated that	The user can be provided with monitoring and/or security agents configuration codes	REQ96, REQ31, REQ77, REQ110

		contains also the monitoring and security agents configuration IaC code		

2.2.2 Fitting into overall PIACERE Architecture

The role of the ICG inside the PIACERE framework is the same described in D3.4 [1]. During this second iteration, the main effort was to the integrate the component with the other PIACERE tools.

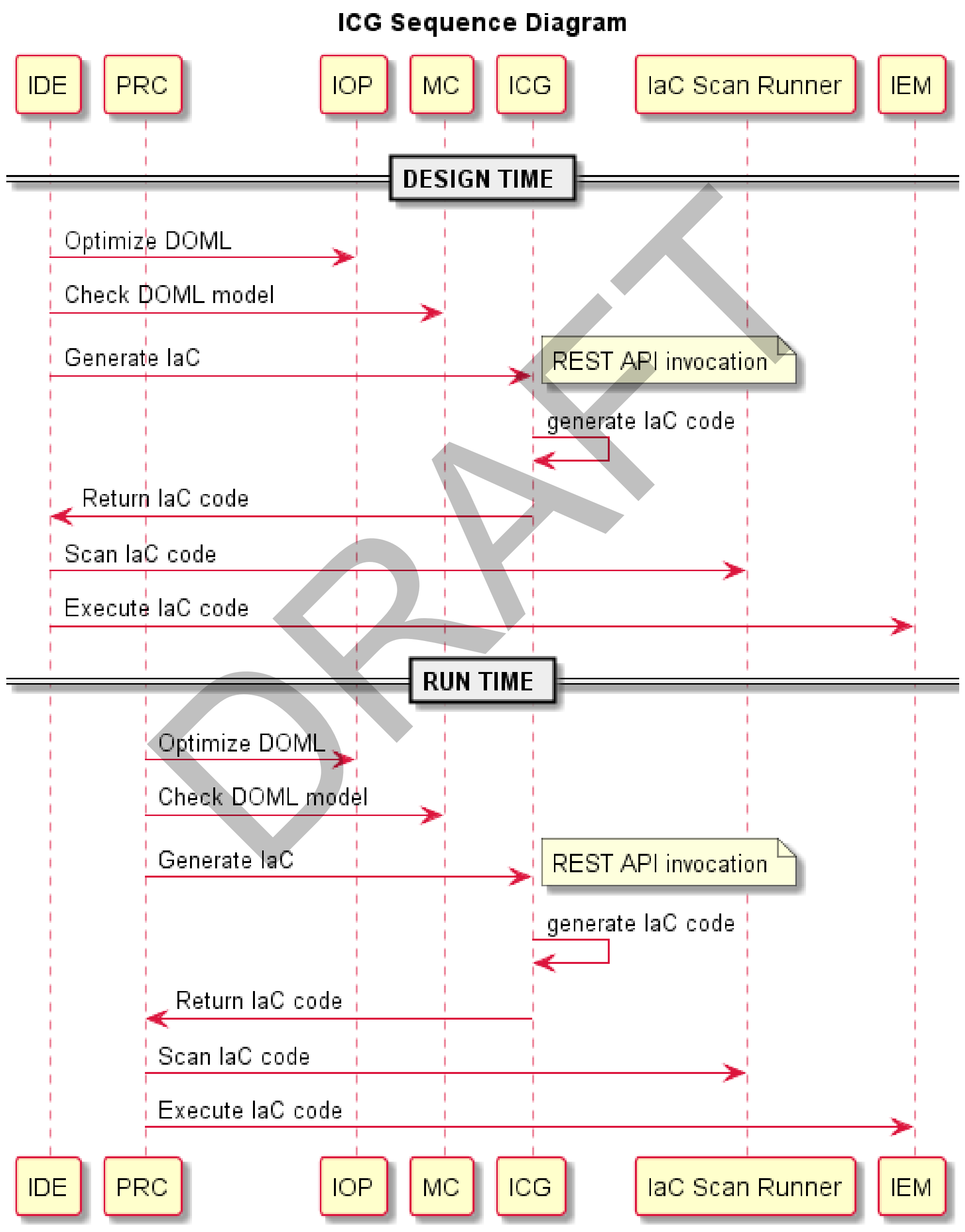
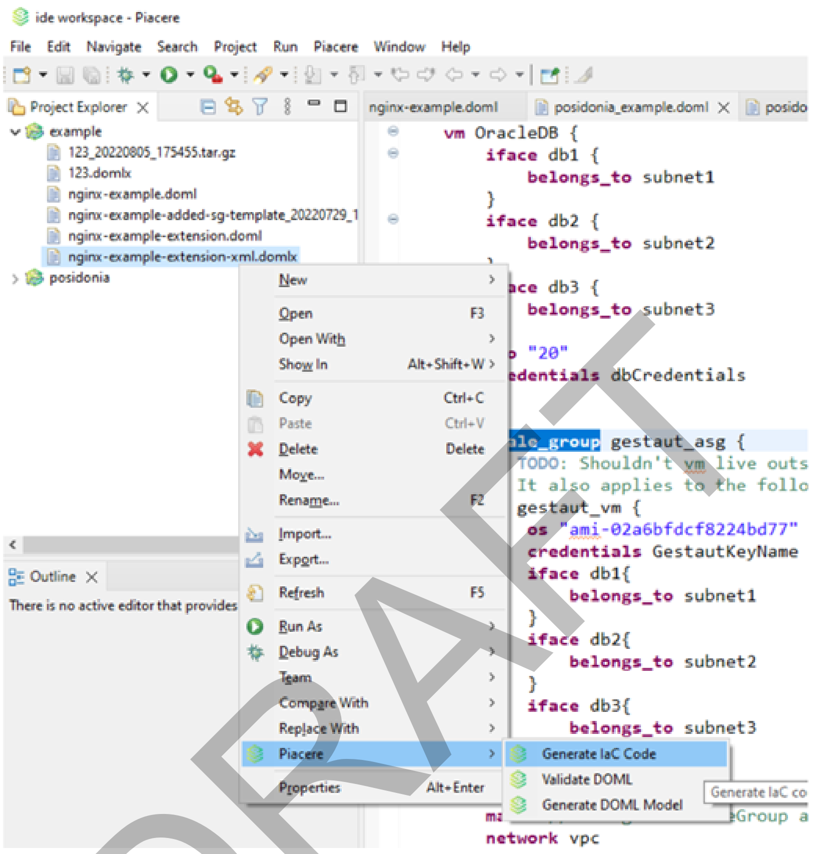


Figure 2: ICG sequence diagram overview

Figure 2 shows the ICG sequence diagram respect to the other PIACERE components. The IDE and PIACERE Runtime Controller (PRC) request to the ICG the generation of the IaC code, the IaC

Executor Manager (IEM) and IaC Scan Runner (previously named "Verification Tools") consume the IaC code generated.

The ICG generates IaC code both for the user and the PIACERE internal components, thus it handles the generation of the IaC code for the deployment of the PIACERE monitoring and security agents. This version only integrates the monitoring agents, the security agents will be added during the next iteration.



```
vm OracleDB {
    iface db1 {
        belongs_to subnet1
    }
    iface db2 {
        belongs_to subnet2
    }
ace db3 {
    belongs_to subnet3

o "20"
edentials dbCredentials

ale_group gestaut_asg {
TODO: Shouldn't vm live outs
It also applies to the follo
gestaut_vm {
 os "ami-02a6bfdcf8224bd77"
 credentials GestautKeyName
 iface db1{
     belongs_to subnet1
 }
 iface db2{
     belongs_to subnet2
 }
 iface db3{
     belongs_to subnet3

ma                   eGroup a
network vpc
```

Figure 3: IDE-ICG integration

Figure 3 shows the usage of the ICG through the IDE: the user describes their infrastructure in the DOML language and converts it into the DOMLx file, finally he/she requests the generation of the IaC code using menu selection. After that, the IDE requests through the ICG POST API the generation of the IaC code and obtains a compressed folder with the execution files. This folder will be used by the IEM and the IaC Scan Runner.

2.3 Technical description

This section describes the technical details of the implemented software for the M24 release.

2.3.1 Prototype architecture

The current Internal ICG sequence diagram can be seen inFigure 4.

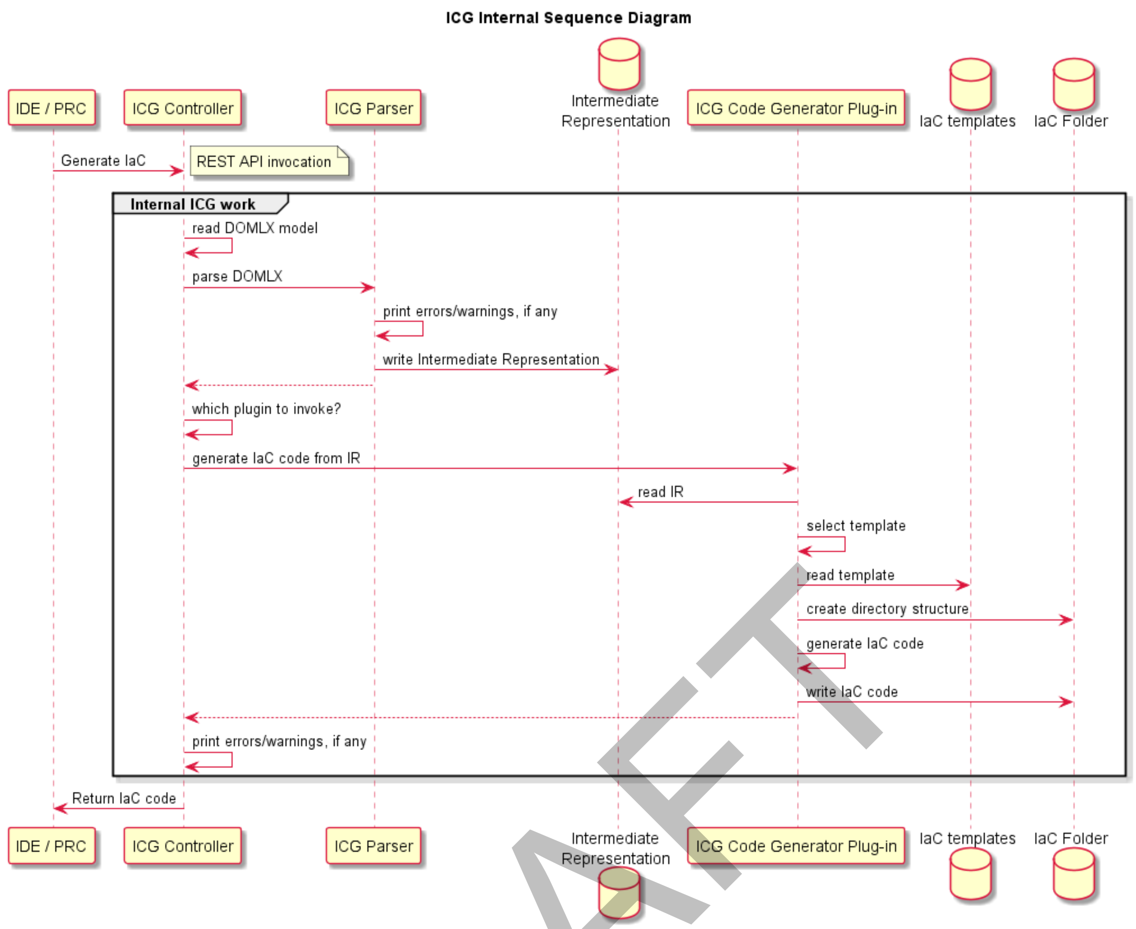


Figure 4: ICG Internal Sequence Diagram

Since the previous prototype in M12, there were a few minor changes to the architecture.
The ICG Controller had internal updates, mainly due to the implementation of the Parser that required updates to the controller to handle the interactions to the Parser. Regarding its functionalities, the ICG Controller did not have any updates.
The ICG Parser was implemented for this prototype. The solution selected for the implementation of the ICG Parser was Python and PyEcore for the DOML metamodel interpretation.
In the same way as the Controller, the Intermediate Representation, the ICG Code Generator and the ICG Plugins functionalities were not updated, the updates were in the internal optimization and evolution of this components.
Lastly, the ICG was updated from a command line compiler to a microservice called through the REST API, this also changed how the IaC code files are managed to be returned as output.

2.3.2 Components' description

Most of the ICG components did not have functional changes from the last prototype so the next chapters will focus mainly on the evolution of these components.

2.3.2.1 ICG Controller

In the current prototype of the ICG, the Controller is complete and all the required functionalities were implemented.

First, the ICG controller code was updated with the addition of a REST API interface which receives the input from the other components, allowing the ICG to receive the DOML model as input (the old prototype used the intermediate representation as input).

The other major improvement was in the definition of all the methods related to the newly developed Parser. After the Controller is called by the IDE or the PRC, it takes the DOMLx received as input and passes it to the Parser that returns the derived Intermediate Representation. Once the Intermediate Representation is generated, the remaining functionalities are as described in the previous deliverable.

2.3.2.2 ICG DOML Parser

In the last prototype the ICG Parser wasn't yet developed so this component is the one that experienced the most prominent evolution. First of all, contrary to the studies made for the previous prototype, the implementation choice was to have this component integrated inside the ICG and not inside the IDE. To better fit in the ICG architecture, the ICG Parser was developed in Python and integrates the PyEcore library to read and navigate the DOML metamodel.

The Parser is called by the Controller, receives as input the DOMLx model and returns as output the Intermediate Representation. DOMLx is the machine-readable version of the DOML model and is generated by the IDE at design time. The ICG Parser extrapolates from the DOMLx all the relevant information necessary to generate the IaC code.

In the table below, a small example of DOML and the related DOMLx representation are presented. It is important to note that the example provided is of v2 of DOML which is the current supported version DOML by the ICG.

Table 3 DOML and DOMLx example

```
doml mysql
application app {
    software_component mysql {
    }
}
infrastructure infra {
    key_pair ssh_key {
        keyfile "local path to ssh key"
    }
    vm vm1 {
        os "ubuntu-20.04.3"
        credentials ssh_key
        iface i1 {
            belongs_to net1
        }
    }
    net net1 {
        address "10.10.10.0/24"
        protocol "tcp/ip"
    }
}
deployment config {
    mysql -> vm1
}
active deployment config
concretizations {
    concrete_infrastructure con_os_infra {
        provider openstack {
            properties {}
            vm concrete_vm1 {
                properties {
                    vm_flavor = "small-centos";
                }
                maps vm1
            }
            net concrete_net {
                properties {}
                maps net1
            }
        }
    }
    active con_os_infra
}
```

```
<?xml version="1.0" encoding="ASCII"?>
<commons:DOMLModel xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:app="http://www.piacere-project.eu/doml/application"
xmlns:commons="http://www.piacere-project.eu/doml/commons"
xmlns:infra="http://www.piacere-project.eu/doml/infrastructure"
name="mysql" activeConfiguration="//@configurations.0"
activeInfrastructure="//@concretizations.0">
  <application name="app">
    <components xsi:type="app:SoftwareComponent" name="mysql"/>
  </application>
  <infrastructure name="infra">
    <nodes xsi:type="infra:VirtualMachine" name="vm1" os="ubuntu-20.04.3"
credentials="//@infrastructure/@credentials.0">
      <ifaces name="i1" belongsTo="//@infrastructure/@networks.0"/>
    </nodes>
    <networks name="net1" protocol="tcp/ip" addressRange="10.10.10.0/24"
connectedIfaces="//@infrastructure/@nodes.0/@ifaces.0"/>
    <credentials xsi:type="infra:KeyPair" name="ssh_key" keyfile="local path to
ssh key"/>
  </infrastructure>
  <concretizations name="con_os_infra">
    <providers name="openstack">
      <vms name="concrete_vm1" maps="//@infrastructure/@nodes.0">
        <annotations xsi:type="commons:SProperty" key="vm_flavor"
value="small-centos"/>
      </vms>
      <networks name="concrete_net" maps="//@infrastructure/@networks.0"/>
    </providers>
  </concretizations>
  <configurations name="config">
    <deployments component="//@application/@components.0"
node="//@infrastructure/@nodes.0"/>
  </configurations>
</commons:DOMLModel>
```

From the example we can observe that the DOML has different layers. The ICG takes care of the concretization, deployment and application layers, navigates, reorganizes and saves this information in a JSON format, the Intermediate Representation.

The Intermediate Representation generated is compatible with the one that was constructed by hand for the previous prototype, allowing for an easy integration with the other ICG components.

2.3.2.3 Intermediate Representation

The Intermediate Representation has not been updated from the last prototype, see D3.4 [1] section 2.2.2.3 for further details.

2.3.2.4 ICG Code Generator

The ICG Code Generator for this prototype did get some refactoring done and due to the changes of the overall ICG structure there were also some updates to the Code Generator, but in the same way as the other ICG component the functionalities of the Code Generator did not change.

The first change to be observed is due to the containerization of the ICG. In the previous prototype, the Plug-ins, after generating the code, wrote it on the filesystem where the other components could retrieve it or the code could simply be viewed by the user. With the new approach, the code is only temporarily stationed on the container's ephemeral volume to be packaged and returned as body of the REST API call.

From the previous prototype there were also relevant changes in the folder structure the Code Generator provides for the output code. This update was made to allow for a more readable and organized output code.

In the same line, to also allow for the proper integration with the IaC Executor Manager (IEM), the new folder structure was accompanied by new configuration files. These files are generated by the Plug-ins and indicate to the IEM the required properties and information to execute the code.

Lastly, to integrate the monitoring agents, the Code Generator for this prototype provides the IaC required to configure them on the defined virtual machines.

2.3.2.4.1 Terraform plug-in

The Terraform plug-in did not receive updates that would change its functionalities or inner workings. Its changes are strictly related to the changes of all ICG Code Generator. The main update was the general refactoring of the code that made the ICG faster and more robust.

Aside from that, the plug-ins were not impacted by any other addition to the providers or resources available in DOML. This was possible due the extensibility features present in the ICG, mainly the capability to add or update the template library as a way to easily add new resources.

2.3.2.4.2 Ansible plug-in

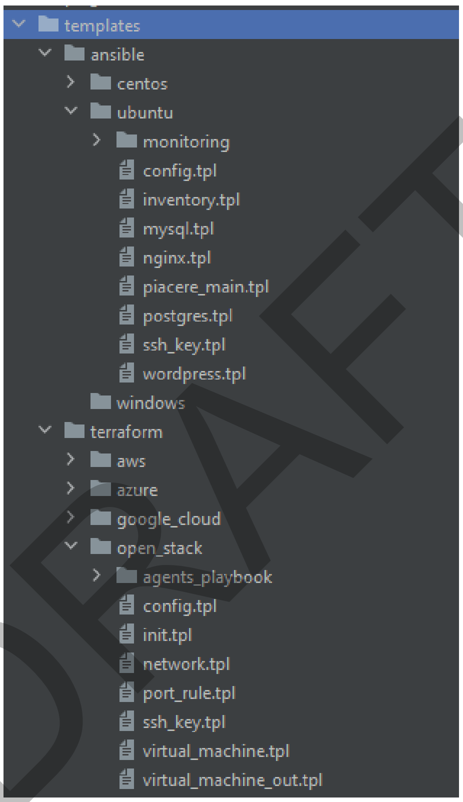
Same as for the Terraform plug-in the Ansible plug-in main update was the refactoring of the code, which increased the robustness and performance of the code.

The IaC code used to configure the monitoring agents is Ansible-based, so it counts as a new addition to the Ansible template library. This update also did not directly impact the functionality of the Ansible plug-in.

2.3.2.5 *Template library*

The IaC templates collection is expanded and re-organized. Now, the ICG can create IaC code for new resources defined in the DOML language and can extend its set of templates in an easy way.

The main directory is the "template" directory, and it is organized with one folder per IaC language supported by the ICG. At the moment, these folders are "ansible" and "terraform", but in the future an expert user can make new ones dedicated to new IaC languages.
In these folders, there are the templates for the generation of the target IaC language. Thus in the "ansible" directory there are the Ansible IaC templates grouped by operating system and in the "terraform" directory there are the Terraform IaC templates grouped by the cloud provider (see Figure 5). Here again, an expert user can add his/her templates to allow the creation of new IaC code for that specific IaC language.



```
templates
  ansible
    centos
    ubuntu
      monitoring
      config.tpl
      inventory.tpl
      mysql.tpl
      nginx.tpl
      piacere_main.tpl
      postgres.tpl
      ssh_key.tpl
      wordpress.tpl
    windows
  terraform
    aws
    azure
    google_cloud
    open_stack
      agents_playbook
      config.tpl
      init.tpl
      network.tpl
      port_rule.tpl
      ssh_key.tpl
      virtual_machine.tpl
      virtual_machine_out.tpl
```

Figure 5: Templates library

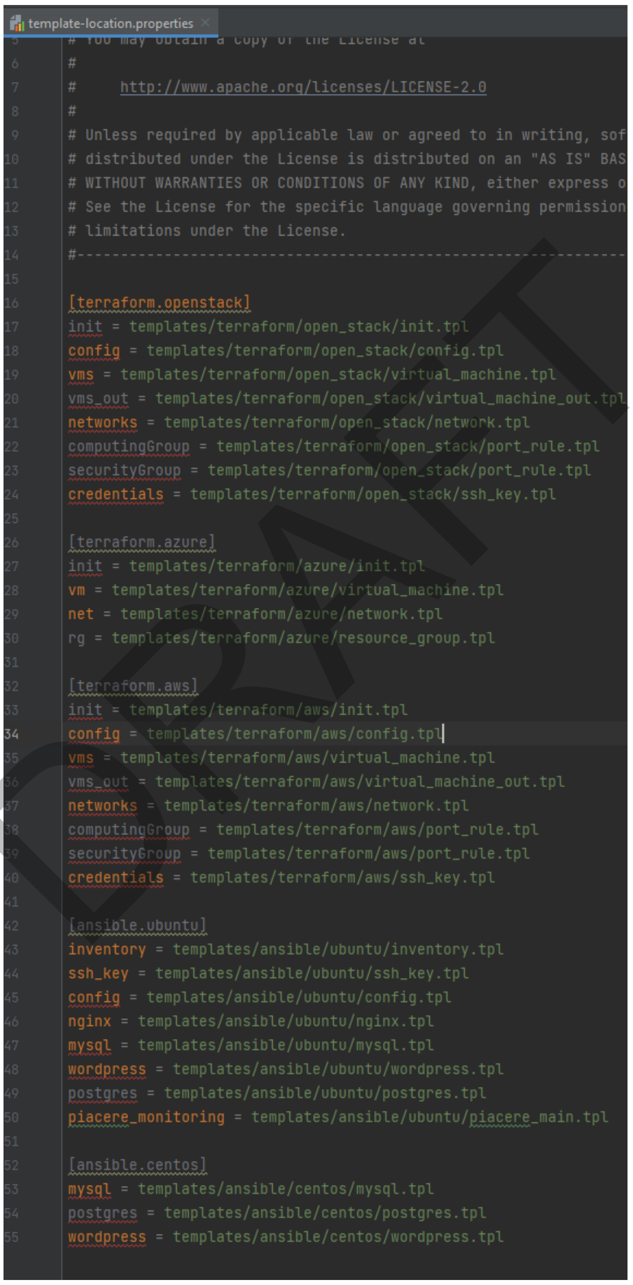
Regarding the Ansible templates, the ICG can now generate IaC code for the creation of the nginx application, the PIACERE monitoring agents, WordPress and the MySQL and Postgres databases.

Regarding the Terraform templates, ICG now supports AWS, Azure and OpenStack cloud providers and can generate Network, Subnet, Security Group, Virtual Machine and SSH Keys cloud resources.

The ICG plug-ins generate the IaC code choosing the right template from a properties file called "template-location.properties", shown in Figure 6. In this file, for each DOML resource there is the path to the template to be used. Thanks to this approach, an expert user can add the reference to his new templates adding the right path.

For instance, referring to the properties files presented in Figure 6, let's see what happens if the XML representation of DOML model presented in Table 3 is provided to the ICG. The first information the ICG obtains in this example is contained inside the concretizations layer and is the "provider". Once the provider is specified the second object it recognizes is the "vms" object, this is associated to a template in the properties file (see line 19 of properties file Figure 6) which is then selected. The same happens for the next object "networks" and so on until all the DOML model is completed. The selected templates will be used by the ICG to provide the required IaC.

Note: The elements present inside the XML format of the DOML are different from the ones inside the DOML model, for example the element referred as "vm" inside the DOML model becomes part of the "vms" element in the DOMLx representation.

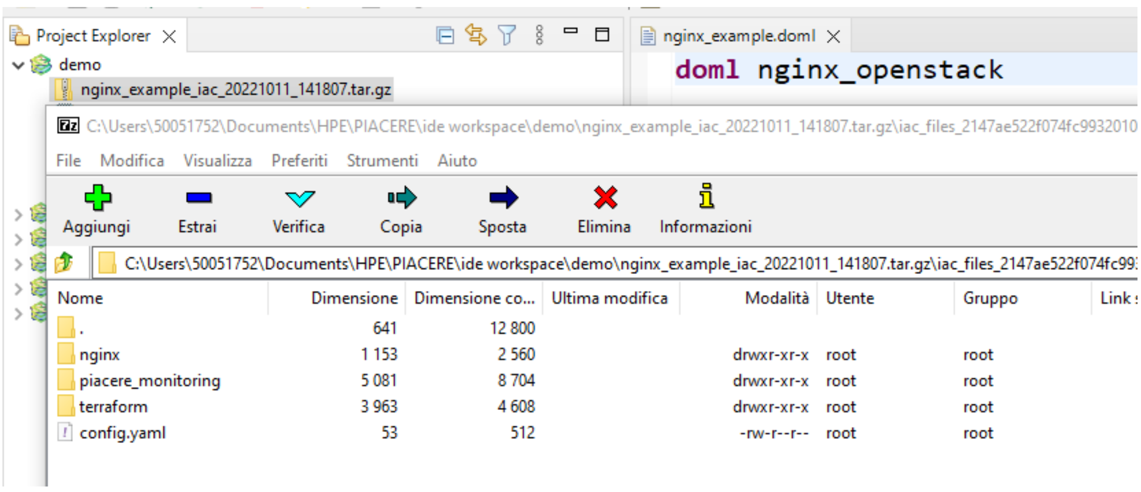

template-location.properties

```
5   # You may obtain a copy of the License at
6   #
7   #     http://www.apache.org/licenses/LICENSE-2.0
8   #
9   # Unless required by applicable law or agreed to in writing, sof
10  # distributed under the License is distributed on an "AS IS" BAS
11  # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express o
12  # See the License for the specific language governing permission
13  # limitations under the License.
14  #----------------------------------------------------------------
15
16  [terraform.openstack]
17  init = templates/terraform/open_stack/init.tpl
18  config = templates/terraform/open_stack/config.tpl
19  vms = templates/terraform/open_stack/virtual_machine.tpl
20  vms_out = templates/terraform/open_stack/virtual_machine_out.tpl
21  networks = templates/terraform/open_stack/network.tpl
22  computingGroup = templates/terraform/open_stack/port_rule.tpl
23  securityGroup = templates/terraform/open_stack/port_rule.tpl
24  credentials = templates/terraform/open_stack/ssh_key.tpl
25
26  [terraform.azure]
27  init = templates/terraform/azure/init.tpl
28  vm = templates/terraform/azure/virtual_machine.tpl
29  net = templates/terraform/azure/network.tpl
30  rg = templates/terraform/azure/resource_group.tpl
31
32  [terraform.aws]
33  init = templates/terraform/aws/init.tpl
34  config = templates/terraform/aws/config.tpl
35  vms = templates/terraform/aws/virtual_machine.tpl
36  vms_out = templates/terraform/aws/virtual_machine_out.tpl
37  networks = templates/terraform/aws/network.tpl
38  computingGroup = templates/terraform/aws/port_rule.tpl
39  securityGroup = templates/terraform/aws/port_rule.tpl
40  credentials = templates/terraform/aws/ssh_key.tpl
41
42  [ansible.ubuntu]
43  inventory = templates/ansible/ubuntu/inventory.tpl
44  ssh_key = templates/ansible/ubuntu/ssh_key.tpl
45  config = templates/ansible/ubuntu/config.tpl
46  nginx = templates/ansible/ubuntu/nginx.tpl
47  mysql = templates/ansible/ubuntu/mysql.tpl
48  wordpress = templates/ansible/ubuntu/wordpress.tpl
49  postgres = templates/ansible/ubuntu/postgres.tpl
50  piacere_monitoring = templates/ansible/ubuntu/piacere_main.tpl
51
52  [ansible.centos]
53  mysql = templates/ansible/centos/mysql.tpl
54  postgres = templates/ansible/centos/postgres.tpl
55  wordpress = templates/ansible/centos/wordpress.tpl
```

Figure 6: template-location.properties file

2.3.3 Output description

The ICG output is a compressed folder with the IaC files and the instructions about how to execute them.


Project Explorer
demo
nginx_example_iac_20221011_141807.tar.gz
nginx_example.doml
File Modifica Visualizza Preferiti Strumenti Aiuto
Aggiungi
Estrai
Verifica
Copia
Sposta
Elimina
Informazioni

```
doml nginx_openstack
```

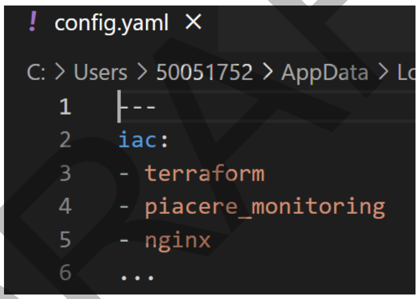
C:\Users\50051752\Documents\HPE\PIACERE\ide workspace\demo\nginx_example_iac_20221011_141807.tar.gz\iac_files_2147ae522f074fc9932010

C:\Users\50051752\Documents\HPE\PIACERE\ide workspace\demo\nginx_example_iac_20221011_141807.tar.gz\iac_files_2147ae522f074fc99

Nome	Dimensione	Dimensione co...	Ultima modifica	Modalità	Utente	Gruppo	Link
.	641	12 800					
nginx	1 153	2 560		drwxr-xr-x	root	root	
piacere_monitoring	5 081	8 704		drwxr-xr-x	root	root	
terraform	3 963	4 608		drwxr-xr-x	root	root	
config.yaml	53	512		-rw-r--r--	root	root	

Figure 7: ICG output compressed folder

Figure 7 shows the root of the output compressed folder. It contains one folder per each module to be executed: the "terraform" folder contains the files for the provisioning of the infrastructure, the "piacere_monitoring" folder is dedicated to the installation of the PIACERE monitoring agents, the "nginx" folder is for the installation of the nginx. The configuration file describes the order of execution of the folders, it is called "config" and it uses the YAML format (see Figure 8).



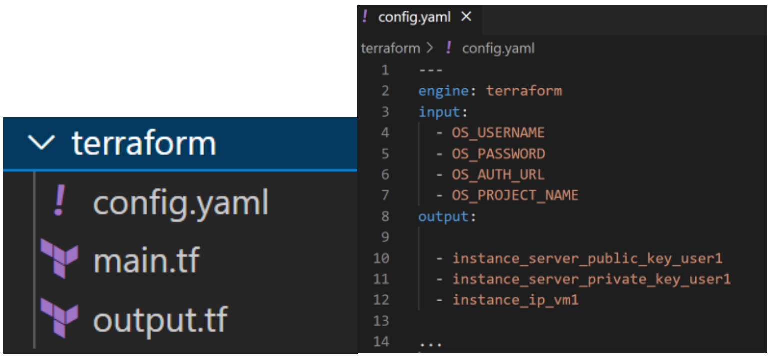
```yaml
---
iac:
- terraform
- piacere_monitoring
- nginx
...
```

Figure 8: ICG output config.yaml

Each folder in the root contains the IaC files and a config.yaml file. This time, the configuration file contains instructions about:

- "engine": it is the IaC language to be used
- "input": it is a list of input variables to be passed during the execution of the IaC files
- "output": it is a list of output variables coming from the execution of the IaC files.

Figure 9 shows an example of the config.yaml file and files structure belonging to the "terraform" folder.



```yaml
---
engine: terraform
input:
  - OS_USERNAME
  - OS_PASSWORD
  - OS_AUTH_URL
  - OS_PROJECT_NAME
output:

  - instance_server_public_key_user1
  - instance_server_private_key_user1
  - instance_ip_vm1

...
```

Figure 9: output terraform folder

2.3.4 Technical specifications

The ICG components are still written with Python version 3.6 and use the Jinja2 Python library version 3.0.3. In this new version more libraries have been used:

- Fast API version 0.74.1 and Uvicorn version 0.17.5 for the REST API implementation
- PyEcore version 0.12.2 for the ICG Parser
- PyYaml version 6.0 for the creation of the configuration files given in the output compressed folder

3 Delivery and usage

3.1 Package information

During this second iteration a refactoring has been done to the ICG code.

The folders and files most useful to the user for the setup and management of the ICG are the following:

- **Templates** folder: contains the templates to be used for the IaC code generation grouped by the IaC language
- **Template-location.properties** file: this file lists the reference to the proper template to be used for the generation of a specific DOML resource
- **Input_file_generated** folder: contains the Intermediate Representation generated by the ICG
- **Dockerfile** and **requirements.txt** files: used for the Docker packaging of the application
- **Doc** folder: contains some example scenarios of usage of the ICG

The ICG components are organized in packages and folders containing these packages are the following:

- **Controller**: folder dedicated to the ICG Controller package
- **Icgparser**: folder dedicated to the ICG Parser package
- **Plugin**: folder containing the packages of the Terraform and the Ansible ICG plug-ins

3.2 Installation instructions

This new version of the ICG runs into Docker container and no Python nor Python library installation is needed.

The code is available on the public TECNALIA GitLab repository and can be downloaded through the following command:

```
git clone https://git.code.tecnalia.com/piacere/public/the-platform/icg.
```

Once the tool is downloaded, you can run it with docker executing these commands from the root of the project:

```
docker build -t icg:1.0.0 .
```

```
docker run --name icg -d -p 5000:5000 icg:1.0.0
```

and finally the API docs are available at: http://localhost:5000/docs.

The ICG command line is still available and the tool can be executed with Python 3.6 too as described in D3.4 [1], installing Jinja2 v3.0.3 library as in the previous version and including also PyYAML v6.0, FastAPI v0.74.1, Uvicorn v0.17.5 and PyEcore 0.12.2 libraries. The following command can be used to install all these libraries:

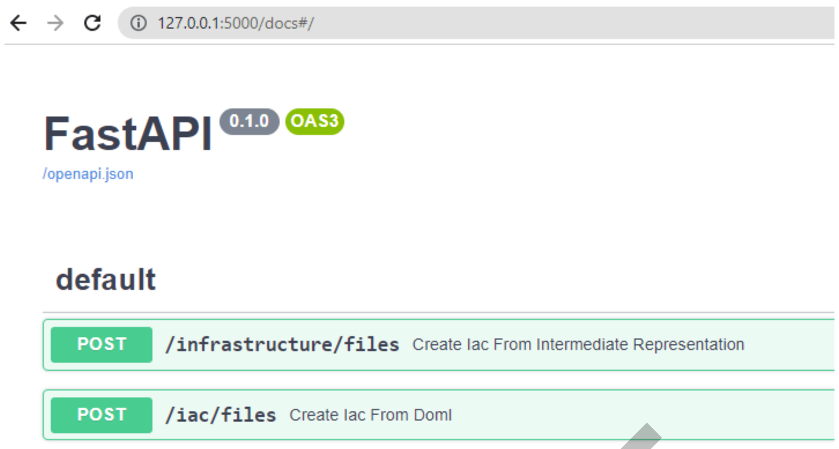
```
pip install -r requirements.txt
```

As previously introduced the ICG can be run as a command line tool, an example of how it can be done on Windows is the following:

```
py .\main.py -d icgparser/doml --single icgparser/doml/nginx-openstack_v2.domlx
```

3.3 User Manual

The initial step for the component installation is described in Section 213.2. After that, the ICG REST APIs are described at: http://localhost:5000/docs.


127.0.0.1:5000/docs#/

FastAPI 0.1.0 OAS3
/openapi.json

default

POST /infrastructure/files Create Iac From Intermediate Representation

POST /iac/files Create Iac From Doml

Figure 10: ICG REST APIs

There are two API endpoints:

- POST /infrastructure/files: takes as input the Intermediate Representation as JSON and produces in output the compressed folder. This API is used mostly for testing purposes.
- POST /iac/files: takes as input the XML representation of the DOML model and returns the compressed folder containing the IaC code and execution instructions.

The local ICG installation can be tested with the following command:

```
curl -X 'POST' \
  'https://localhost:5000/iac/files' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/xml' \
  -d "@model.domlx" \
  -o "output.tar.gz"
```

Where model.domlx is the XML representation of a given DOML model, see Table 3 for an example of a possible DOML model.

In section 2.2.1, there are also some scenarios that indicate how the ICG can be used through the IDE to fulfil different user stories.

3.4 Licensing information

The ICG component has been released in open source under the Apache 2.0 license.

3.5 Download

The ICG can be downloaded from the public TECNALIA GitLab repo at https://git.code.tecnalia.com/piacere/public/the-platform/icg or from the public HPE GitHub repo at: https://github.com/HewlettPackard/icg-iac-code-generator.

4 Conclusions

This document described the second release of the ICG component, implemented in the second year of the PIACERE project. The component is now fully operational, integrated with other design-time components and works with DOML v2. As the next version of DOML will be released the compatibility of the ICG with it will also be guaranteed.

The main functionalities of the ICG have been listed, along with its internal architecture, the relationship among the implemented functionalities and the updated requirements collected in deliverable D2.2, plus some advanced usage scenarios.

All internal ICG components have been described, along with their interactions. Furthermore, the deliverable described in detail the files included in the delivered package and documents how to install and use the released software.

The next version of the ICG will further enhance the template library to offer more complete code generation capabilities, both for improving the support of DOML elements and possibly for supporting other target platforms, depending on Use Case requirements.

In the final release, due at M30 (deliverable D3.6), we plan to provide guidelines for writing new templates, so that expert users will be able to develop their own templates or to modify existing ones, both for supporting new DOML concepts and for providing support for new IaC languages.

5 References

[1] «PIACERE Deliverable D3.4 - Infrastructural code generation – v1». [Online] https://zenodo.org/record/6821657#.Y0_Gr3ZBw2x

[2] «Gherkin Reference» [Online] https://cucumber.io/docs/gherkin/reference/.