Grant Agreement No.: 731677
Call: H2020-ICT-2016-2017
Topic: ICT-13-2016
Type of action: RIA

D3.11: FLAME Technology Roadmap V3

Sebastian Robitzsch, Dirk Trossen (InterDigital Europe)

11 October 2019

This report is the third technology roadmap for a ground-breaking media service delivery platform being developed within the FLAME project. The report is positioned in-between the major release cycles of FLAME platform beta and final candidate releases (Release Candidate). It therefore focuses on the areas of platform deployment and replication, and complements the technology features of the core platform itself described in previous deliverables. Furthermore, this report outlines the FLAME consortium approach to managing the agile development lifecycle of the platform, which drives platform releases before the planned milestone releases outlined in D3.7.

Co-funded by the Horizon 2020
Framework Programme of the European Union

Work package	WP3
Task	Task 3.11
Due date	30/04/2019
Submission date	11/10/2019
Deliverable lead	InterDigital
Version	1.0
Authors	Dirk Trossen (InterDigital), Sebastian Robitzsch (InterDigital)
Reviewers	Manuel Braunschweiler (ETH), Gino Carrozzo (NXW)
Keywords	Agile lifecycle, release schedule, replication

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	06/08/2019	First outline for comments	Dirk Trossen
V0.2	30/08/19	ARDENT added	Sebastian Robitzsch
V0.3	2/09/19	Best Current Practises, Introduction	Sebastian Robitzsch, Dirk Trossen
V0.4	11/09/19	Section 2 completed, revised positioning to other deliverables	Sebastian Robitzsch, Dirk Trossen
V0.5	12/09/19	Review Version	Sebastian Robitzsch, Dirk Trossen
V1.0	27/09/2019	Final Version	Sebastian Robitzsch
V1.1	11/10/2019	Final Quality Assurance	Michael Boniface

Disclaimer

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731677.

This document reflects only the authors' views and the Commission is not responsible for any use that may be made of the information it contains.

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:	**R**	
Dissemination Level		
PU	Public, fully open, e.g. web	✓
CL	Classified, information as referred to in Commission Decision 2001/844/EC	

Co-funded by the Horizon 2020 Framework Programme of the European Union

CO	Confidential to FLAME project and Commission Services	

Co-funded by the Horizon 2020
Framework Programme of the European Union

EXECUTIVE SUMMARY

This report is deliverable D3.11 FLAME Technology Roadmap V3, and it is positioned in-between the major release cycles of FLAME platform beta and candidate releases (Release Candidates, RC). The document focuses on the areas of platform and technology development that complements the technology features of the FLAME core platform described in previous deliverables issued by workpackage 3.

Due to this main scope, the report presents also the agile development lifecycle of the platform while also outlining the replication tools and processes that are being developed in the course of the replication to the different sites, including the Open Call 3 replicators. The outcome is a tool chain for accelerating the replication of the FLAME platform at deployment sites with a significant reduction in time and cost. ***The deliverable originally due by April 2019, has been delayed to incorporate first insights and integration outcomes from Open Call 2 replication into our tools and process.***

The development lifecycle approach described in this document has been developed in conjunction with the experimentation and replication at sites in Bristol and Barcelona. The document describes how the lifecycle management is realized in collaboration with both experimenters and replicators, leading to intermediary releases outside the main release milestones outlined in D3.7, i.e., alpha, beta and Release Candidate releases.

The replication process described in this document was developed alongside the platform development and deployment in experimentation sites, specifically Bristol and Barcelona. This replication process and the developed tools to support replication are described, while a more detailed replication documentation is planned for a later release in the form of an online repository throughout fall of 2019.

This report complements the previously delivered report D3.7 [FLAME-D3.7] and focuses on the technology development and process aspects more than the specific technology features of the FLAME platform itself. With this, the document serves as a stand-alone document.

Co-funded by the Horizon 2020
Framework Programme of the European Union

TABLE OF CONTENTS

Co-funded by the Horizon 2020
Framework Programme of the European Union

LIST OF FIGURES

Co-funded by the Horizon 2020
Framework Programme of the European Union

LIST OF TABLES

Co-funded by the Horizon 2020
Framework Programme of the European Union

ABBREVIATIONS

API	Application Programming Interface
ARDENT	Automated platfoRm DEploymeNt Toolchain
BCP	Best Current Practise
CI	Continuous Integration
CLI	Command Line Interface
CLMC	Cross-Layer Management and Control
CIDR	Classless Inter-Domain Routing
DC	Data Centre
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Server
FiaB	FLAME-in-a-Box
FQDN	Fully Qualified Domain Name
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol.
HOT	HEAT Orchestration Template
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
KPI	Key Performance Indicator
LEARNT	FLIPS Debugging and Profiling Testbed
MGMT	Management
MPLS	Multi-protocol Label Switching
MTU	Maximum Transmission Unit
NFV	Network Function Virtualisation
PCE	Path Computational Element

Co-funded by the Horizon 2020
Framework Programme of the European Union

PoA	Point of Attachment
PS	Platform Service
QoS	Quality of Service
RAN	Radio Access Network
RC	Release Candidate
SDN	Software Defined Network
SIA	Secure Inbound Access
SF	Service Function
SFC	Service Function Chain
SFE	Service Function Endpoint
SFEMC	SF Endpoint Management and Control
SFR	Service Function Routing
SR	Service Router
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
TOSCA	Topology and Orchestration Specification for Cloud Applications
UE	User Equipment
VLAN	Virtual Local Area Network
VM	Virtual Machine
WiFi	Wireless Fidelity
YML	Yet another Markup Language

Co-funded by the Horizon 2020
Framework Programme of the European Union

1 INTRODUCTION

This document describes the technical roadmap for replication and infrastructure integration of the FLAME platform in replicators, including a description of the agile development approach that leads to the planned, stable platform releases. With this, the deliverable is positioned as a process description document that complements the planned platform releases with additional processes and tools for said replication and agile development.

The FLAME consortium has planned three major releases of the FLAME offering within the lifetime of the project, as outlined in [FLAME-D3.7]. The timing of major releases is aligned with the timescales of trials. Each major release will include significant feature enhancements within the overall offering across infrastructure, platform and media services. A release at the project level indicates the launch of a "FLAME Service" for trials in contrast to the release of specific software products that the FLAME Service depends on.

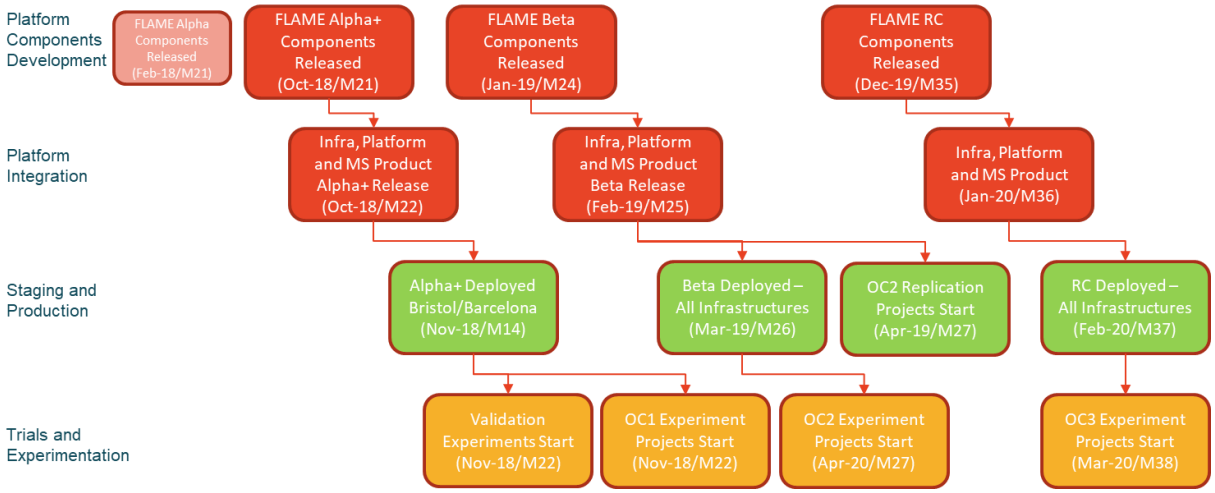

Figure 1: Platform releases in relation to project milestones

After the initial releases in Feb-18 (for the initially planned alpha release for internal testing), updated releases were planned for Oct-18, Jan-19 and Dec-19 with the working names of Alpha, Alpha+, Beta and Release Candidate (see Figure 1), respectively. These major releases correspond to milestones for FLAME feature implementation, with said feature mapping onto the releases outlined in [FLAME-D3.7].

The project implements DevOps processes to offer greater agility in the implementation of release and features. As such, minor releases are delivered in between major milestones to incorporate new features when they are available and hot bug fixes when they are critical to service operations.

With the final release candidate planned for December 2019, this deliverable addresses the roadmap on replication documentation and tools, placed in the cycle of Figure 1 in-between the beta and RC release of the overall platform. For this, we utilize the ongoing replication of the FLAME platform to not only the internal sites (Barcelona and Bristol for the test beds, and London as well as Southampton for the developer toolchain, as outlined in [FLAME-D3.7]) but also to the Open Call 2 accepted replication projects.

Co-funded by the Horizon 2020 Framework Programme of the European Union

2 PLATFORM DEVELOPMENT LIFECYCLE

As mentioned in the introduction, the major platform releases in FLAME are planned according to the timeline presented in D3.7 [FLAME-D3.7], shown in Figure 1. The following section presents our approach to working in conjunction with replications and trials to establish an agile development lifecycle.

2.1 PROCESS OVERVIEW

Through Open Call 2, additional external replication sites are starting to bring FLAME into their infrastructures. The process of deployment, instantiation at specific infrastructures and experimentation generates insights which help to consolidate and refine the FLAME release contents in sake of stabilisation. Insights from Open Call 1 and 2 validation experiments as well as FLAME trials in Bristol and Barcelona are being utilised in improving the platform reliability across the existing sites as well as the Southampton Sandpit and FLAME-in-a-Box (FiaB).

To ease and efficiently manage this process, a more agile code base maintenance processes must be implemented in order to integrate new features and bug fixes into the code base and bring it to all replication sites. Figure 2 illustrates the experimental-driven approach applicable to all replication sites at high level.

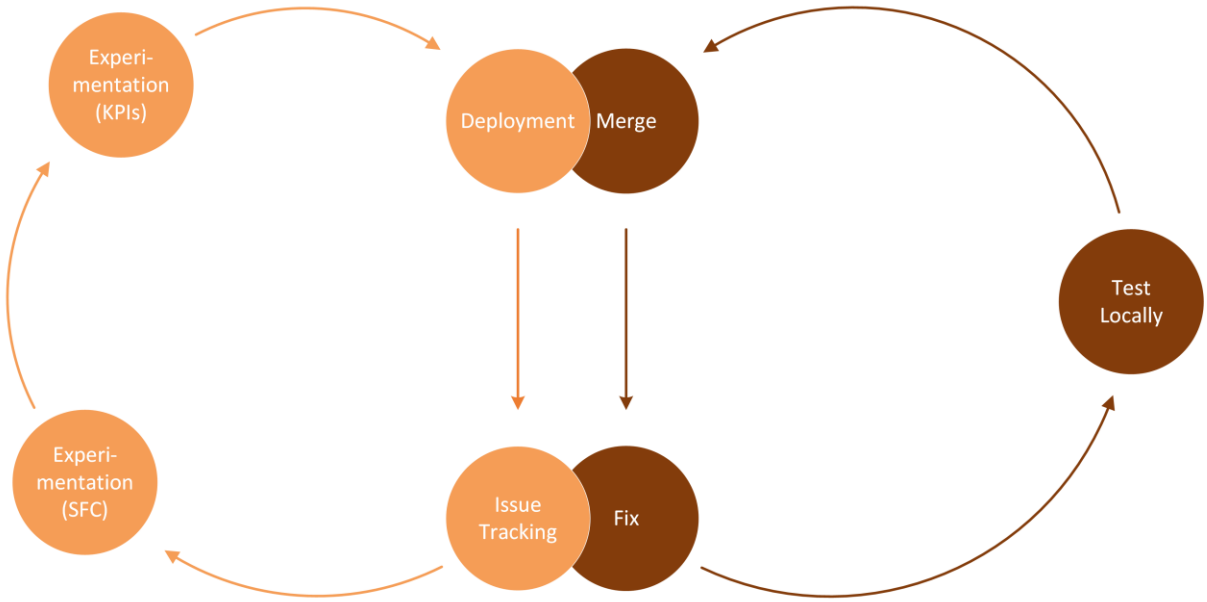

Figure 2: Agile experiment-driven replication workflow

On the left side, the experiments are carried out at a particular replication site[1] focussing on testing and experimenting with the service function chain itself (circle at the bottom left in Figure 2); later the experimentation runs against pre-defined objectives and KPIs ensuring the readiness of the entire communication stack for a trial (circle at the top left in Figure 2). Once an issue has been found, it is tracked, fixed, and tested locally in a separate process outside the replication site, as illustrated in the

[1] This includes the sandpit and FLAME-in-a-Box (FiaB) with FiaB focusing on experimentation of the SFC only.

Co-funded by the Horizon 2020
Framework Programme of the European Union

circles at the center and the right in Figure 2. If issues are of rather complex nature, a replication of the scenario is conducted in a local testbed where debug versions of the deployed platform are being used to investigate the issue in a more efficient manner. Upon the fix being addressed, the merged solution is then re-deployed sequentially across all sites based on their availability to cause the least disruption to any experimenter.

If the merged solution relates to the interaction with the infrastructure (SDN rule construction, new logical topologies or even a change of hardware) a set of tests are performed to validate the readiness of the platform at a particular site, as described in detail in Section 2.3.

2.2 LIFECYCLE IMPLEMENTATION VIA GITLAB

The maintenance and coordination of the high-level process described in Section 2.1 is described in this section in detail and has been fine-tuned during the last year after a change of leadership in WP3 and the responsible partner for integration. A dedicated "flame-platform" GitLab repository exists since day 1 of the FLAME project and comprises the scripts and documentation to prepare, deploy and maintain a deployed FLAME platform at any replication site (operating on top of OpenStack). As the development of the platform components themselves are hosted outside of this repository, the programming languages used in flame-platform are Shell, Awk, Makefiles and Python only, as illustrated in the figure below[2]. The reason for that is that the flame-platform code are wrapper functions around the OpenStack CLI to build images and deploy/maintain and delete the platform from an OpenStack cloud via the CLI.

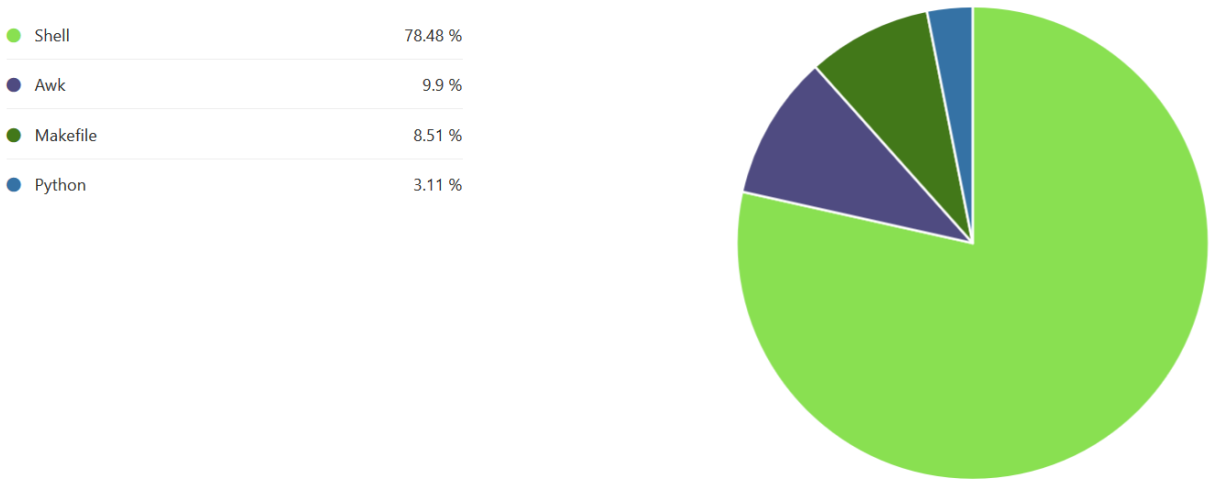

Figure 3: Programming languages used in the flame-platform repository

Furthermore, the development of the service-function routing solution by InterDigital is hosted and conducted outside of FLAME shared repositories: in fact, only binaries are being shared with every partner who has signed an evaluation license with InterDigital in order to deploy the FLAME platform themselves (such as FLAME-in-a-Box).

The *flame-platform* repository has two permanent branches: *master* and *integration*. The tasks around the two branches are illustrated in Figure 4 with master on the left and integration on the right. As depicted, the master branch only receives updates if a milestone (MS) has been closed and is ready to

[2] Obtained from https://gitlab.it-innovation.soton.ac.uk/FLAME/consortium/3rdparties/flame-platform/graphs/master/charts. Note, the repository is private and requires authentication.

Co-funded by the Horizon 2020 Framework Programme of the European Union

be merged into master. In that case, a merge request is conducted, and the update HEAD of master is tagged with the internal release numbers. As a last step, the new flame-platform release is officially released to the project by upgrading the platforms deployed across all sites using the master branch.

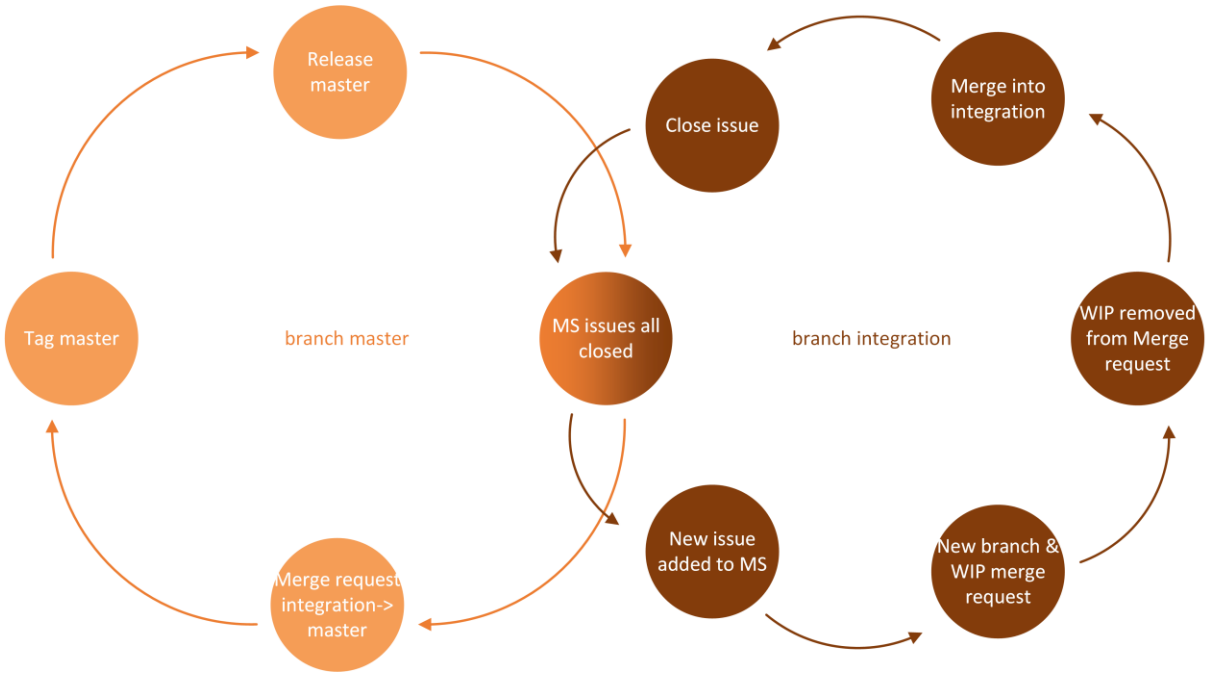

Figure 4: Release workflow and integration cycle

If an issue has been found in a platform a new issue is opened on GitLab and added to the currently active milestone. From there, a new branch is created if the code of flame-platform is affected including a merge request which is called WIP as long as it has not been fixed. Upon the resolution of the issue, the related branch is merged into integration and the issue is closed.

2.2.1 Issue Tracking

The procedures for tracking issues in the flame-platform repository has been developed over time by the Consortium and is the result of various attempts to optimise the level of details and process required to deliver stable and complete releases. Thus, it has been decided to relate GitLab issues to complications experimenters have with the platform or the platform has with the infrastructure. When a new issue is created templates are available for each replication site which allow to select one or more categories from a list along with fields to describe the problem someone is facing. Along with the issue template the platform and infrastructure owners are tagged as well as which version of the platform is being used and which infrastructure it is. An example is given below for the sandpit in Southampton:

```
# Summary

(Summarize the bug encountered concisely)

Please select the platform component you have issues with
- [ ] Service function routing (SFR)
- [ ] Platform services (DNS, DHCP, Internet)
- [ ] Cross layer management and control (CLMC)
- [ ] Orchestrator
- [ ] Service function endpoint management and control (SFEMC)
- [ ] ARDENT
```


Co-funded by the Horizon 2020 Framework Programme of the European Union

```
Please fill out the table below too if SFR was ticked in the list above:

| OSI Layer | Protocol |
| --- | --- |
| Internet | IP or ICMP |
| Transport | UDP or TCP |
| Application | HTTP, HTTPS, SSH, DHCP, FTP, TLS/SSL, other |

# Steps to Reproduce

(How one can reproduce the issue - this is very important)


# What is the current behaviour?

(What actually happens)


# What is the expected correct behaviour?

(What you should see instead)


# Relevant logs and/or screenshots

(Paste any relevant logs - please use code blocks (```) to format console
output,
logs, and code as it's very hard to read otherwise.)


# For Admin Use Only

/label ~sandpit ~"2.3.0"
/cc @sebastian.robitzsch @mjb
```

If an issue is acknowledged, the person working on it assigns it to himself on GitLab. If an issue requires discussions or is about reporting results, there is no new branch created to change any of the code hosted in this repository. If the code is affected a new branch and merge request is created. An issue is being closed by the project owner in corporation with the people involved in the discussion.

2.2.2 Merge Requests

While merge requests are created when a new issue receives its own branch to work on the code, they are not always linked to an issue. Sometimes a new merge request is being created to work directly in GitLab's web IDE on the scripts. These changes are marginal and do not require a new issue itself. In such case the merge request is added to the milestone and it can be retrieved as part of the milestone on GitLab as a new, active or closed issue.

2.2.3 Milestones

In order to track progress, milestones are used in the flame-platform repository, which have the name of the upcoming internal release number. As mentioned before, each issue is added to a milestone when being created and once all issues of a milestone have been resolved, the merge into master can be conducted. Thereby, the tagging of the master branch using the milestone version numbering approach allows to refer directly to what has been changed without writing additional release notes.

The list of completed milestones at the time of writing this deliverable are listed in the figure below.

Co-funded by the Horizon 2020
Framework Programme of the European Union

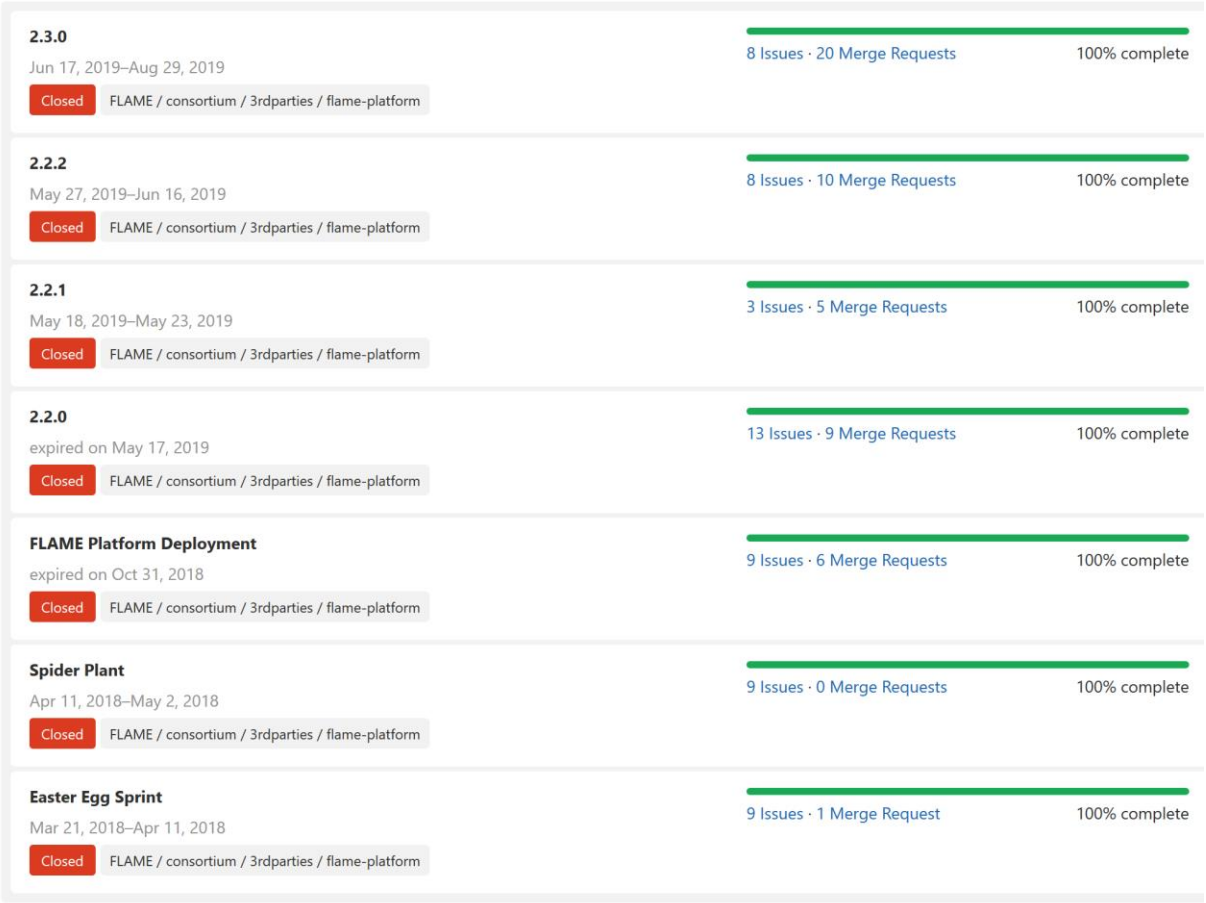

Figure 5: Completed milestones of the flame-platform repository[3]

2.2.4 Tags

Tags are solely used to refer to a particular platform release in the master branch. As indicated before, the release version is used as a tag which maps directory to a milestone where new component versions will be mentioned, and completed issues and merge requests are tracked. Also, if required older platform versions can be easily deployed by checking out a specific tag number from the flame-platform repository. In Figure 6 all existing tags in master are illustrated with the milestone (of format %1.2.3) as hyperlinks to check the changes.

[3] The latest list of completed milestones can be obtained from https://gitlab.it-innovation.soton.ac.uk/FLAME/consortium/3rdparties/flame-platform/milestones?sort=due_date_desc&state=closed. Note, in order to access the link you must have access to FLAME's GitLab.

Co-funded by the Horizon 2020
Framework Programme of the European Union

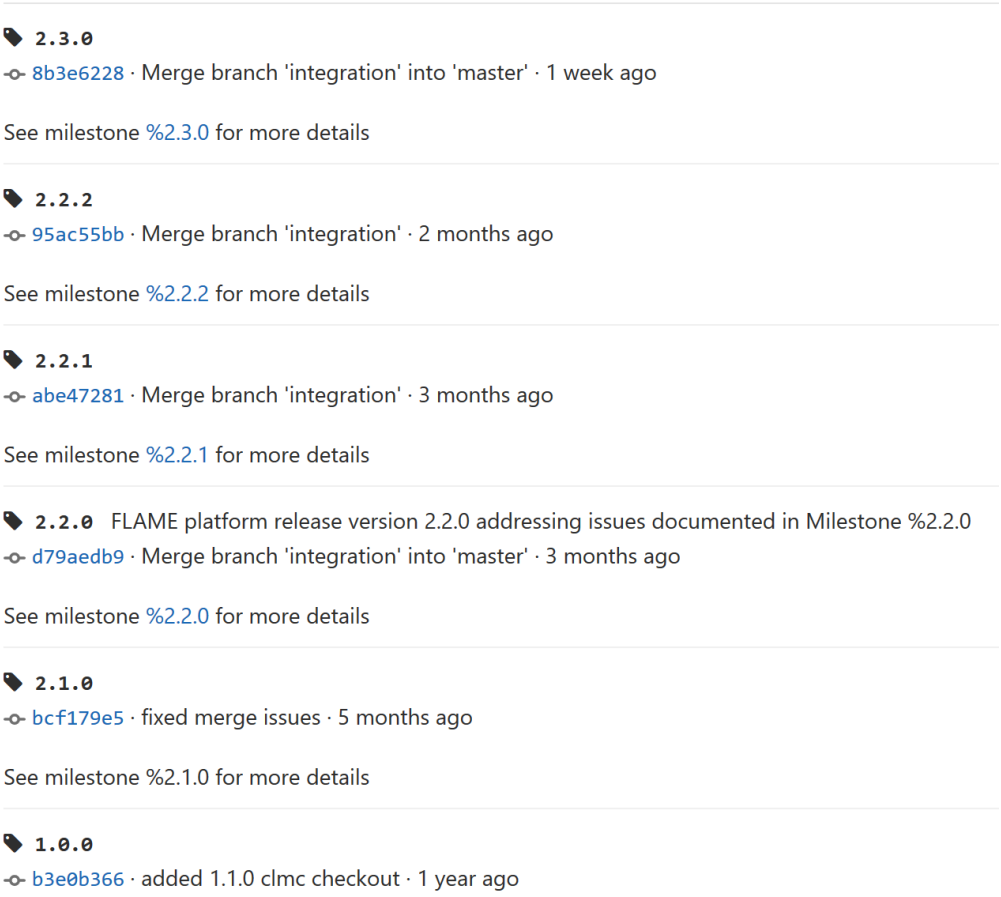

Figure 6: Master tags on GitLab with links to their merged milestones[4]

2.2.5 Labels

In addition to aggregate issues into milestones, labels are used for each issue allowing to further filter new, active and closed issues. It became apparent that some issues relate only to a specific replication site or component. Therefore, the following labels were introduced in order to get a different perspective of issues:

- *Releases*: the currently active release this issue is based on. If an issue cannot be fixed in an upcoming release the label will indicate when it occurred.

- *Replication sites*: each replication site receives a unique label that is added to an issue once it is related to only this particular deployment or the replication site's infrastructure.

2.3 PLATFORM TESTING AND VALIDATION

The FLAME platform consists of various components and APIs (of type service endpoint as well as endpoint), as defined in the platform specification outlined in [FLAME-D3.10]. In order to ensure that

[4] The tags can be reached via https://gitlab.it-innovation.soton.ac.uk/FLAME/consortium/3rdparties/flame-platform/tags. Note that in order to access the link you must have access to FLAME's GitLab.

Co-funded by the Horizon 2020
Framework Programme of the European Union

any update does not affect any other functionality or readiness of the platform a series of tests is conducted.

While Deliverable D4.2 [FLAME-D4.2] has listed the areas each platform component is tested against, this section focuses on the lifecycle of those tests and how the testing and validation work is organised to ensure the readiness of the platform across multiple sites.

2.3.1 Service Function Routing

The testing and validation lifecycle for service function routing (SFR) is illustrated in Figure 7. While the hardening of SFR is in the foreground after most features have been completed, a dedicated testbed has been created at InterDigital's offices in London which allows to debug and profile InterDigital's FLIPS platform, the prototype that implements the SFR component in the FLAME platform. In addition to that, a staging testbed is available for benchmarking the platform over a hardware-based SDN switching fabric interconnecting a three-tier edge compute infrastructure. The two testbeds are described in more detail later in this subsection. First, we focus on the lifecycle to put the testbeds into perspective.

When a new SFR version has been created, it is tested in the **fLips dEbugging And pRofiliNg Testbed** (LEARNT) against memory leaks and profiled against inefficiencies in the code, while comparing it to the benchmark results obtained from the SFR version before. The system tests for this purpose are fully automated procedures and cover the SFR feature list (see [FLAME-D4.2] for more details). Once this step is successfully completed, the updated SFR binaries are integrated into the image build process of the platform including any changes to the process itself or to how FLIPS must be configured when being deployed. The image building and platform deployment is then tested in the staging testbed of InterDigital, also described in more detail later in this section. To ensure the updated platform stack is operated flawlessly, the multi-layer **PeRfOrmance TEst SuiTe** (PROTEST) is being used, which performs iperf-based UDP and TCP, ICMP, HTTP progressive download tests. Upon its successful completion, the updated platform is being released and deployed into the various sites. As part of the updating process, PROTEST is used again to ensure the SFR update has not caused any unexpected behaviour at a specific site.

Co-funded by the Horizon 2020
Framework Programme of the European Union

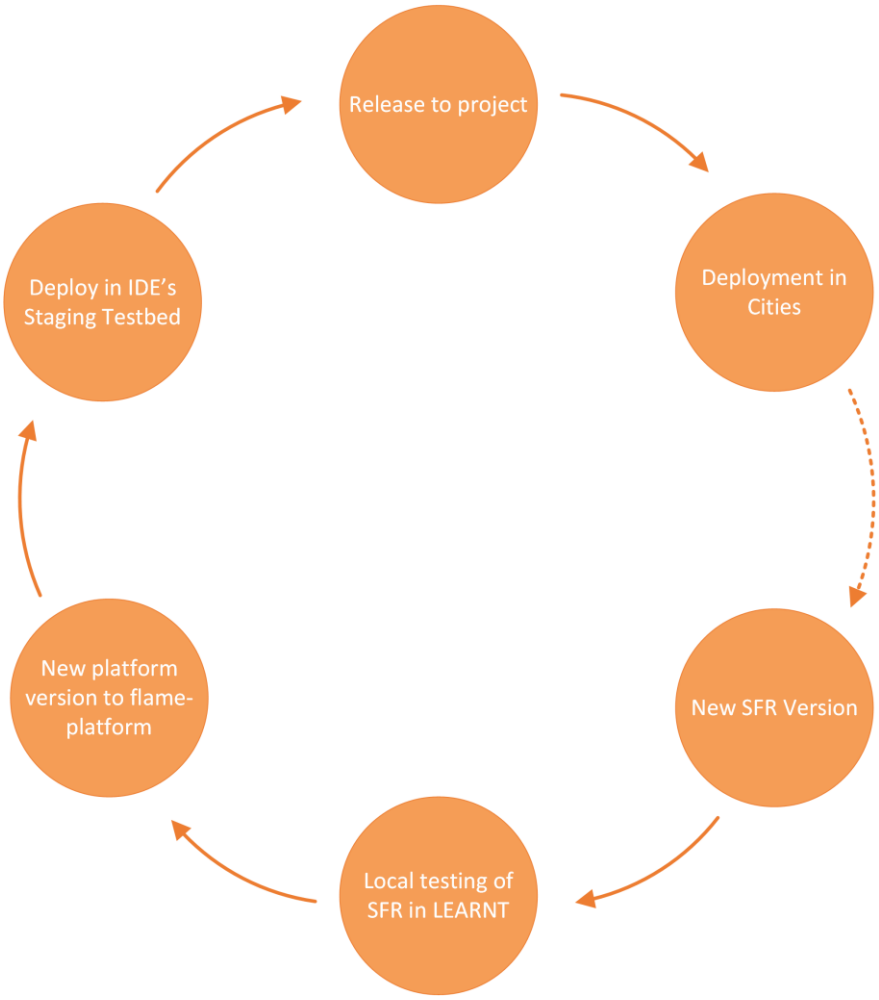

Figure 7: Testing and validation process for new SFR releases

2.3.1.1 FLIPS Debugging and Profiling Testbed

In order to fix issues, LEARNT allows to run FLIPS in different level of logging levels with additional debugging tools ready. LEARNT is also used to profile FLIPS in order to understand where improvements could be made to getting close to an all-IP set-up. The environment is illustrated in the figure below. All nodes at the top are KVM-based virtual instances on a single 8 core machine called *aslan* with one vCPU configured for each instance. The two nodes outside *aslan* are dedicated APU2-based set-ups to not further overload the available cores of *aslan*. One important thing to note is that the UE node *apu46-ue* is connected to the SR, *sr-ue*, via a dedicated 1G port to ensure a stable and predictable connection into the deployed SFR testbed.

Co-funded by the Horizon 2020
Framework Programme of the European Union

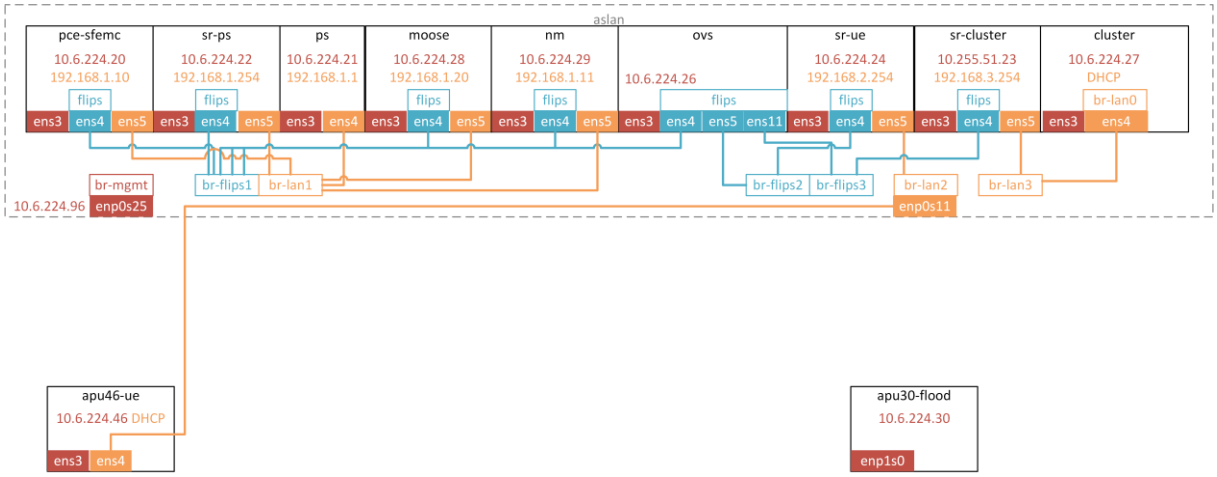

Figure 8: Physical Topology of SFR debugging and profiling testbed at InterDigital

The resulting logical topology is illustrated below. The idea behind this topology is to have a dedicated SDN software switch in the topology that interconnects the SR serving the UE and the SR serving the cluster as well as a dedicated port for traffic towards the other instances.

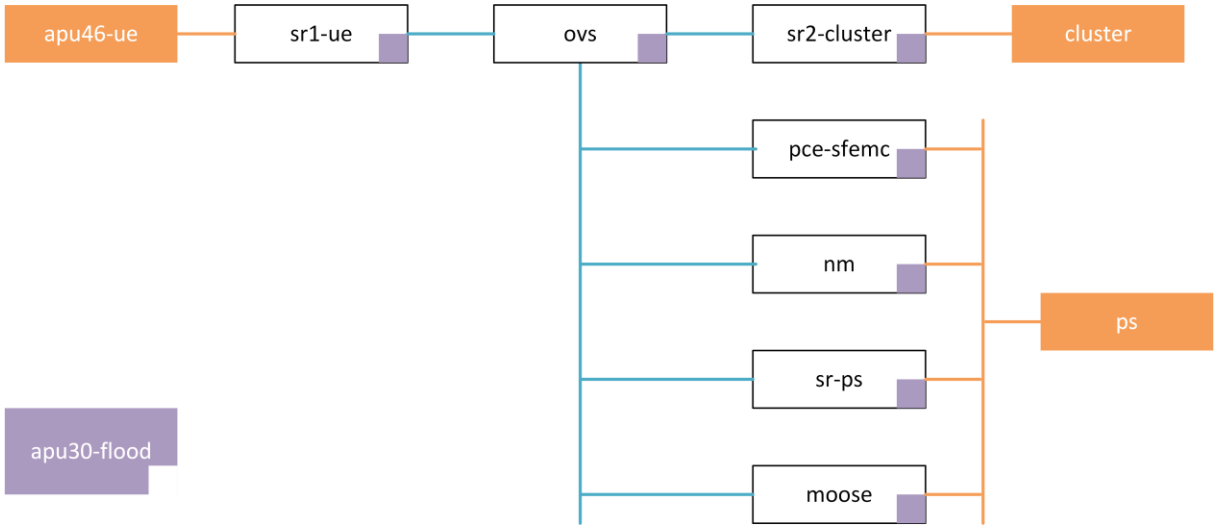

Figure 9: Logical topology of SFR debugging and profiling testbed at InterDigital

On this testbed, system tests have been implemented covering all SFR features listed before and can be run fully automated to ensure they all pass and no SFR update has any impact on the feature. Furthermore, SFR has assertions build into the code to fail on them if an unexpected behaviour occurs which stops the operation of the component gracefully.

2.3.1.2 Staging Testbed

The staging testbed is a three-tier edge compute infrastructure that is interconnected with hardware-based SDN switches as the core 10G ring and a local breakout point with a WiFi access point for connecting any sort of device. The SDN hardware switches are labelled Pica8-n in Figure 10 and interconnect the data centre compute nodes *os-data-centre-1* and *os-data-centre-1*, the edge compute nodes *os-edge-1*, *os-edge-2* and *os-edge-3* as well as the physical breakout point at *Pica8-2* of platform instances that are not orchestrated via OpenStack. The local breakout, also labelled as the

Co-funded by the Horizon 2020
Framework Programme of the European Union

far edge, offers resources to deploy SFs as well. In the figure below, each coloured box inside a compute node represents a virtual instance deployed via OpenStack and is part of the platform stack.

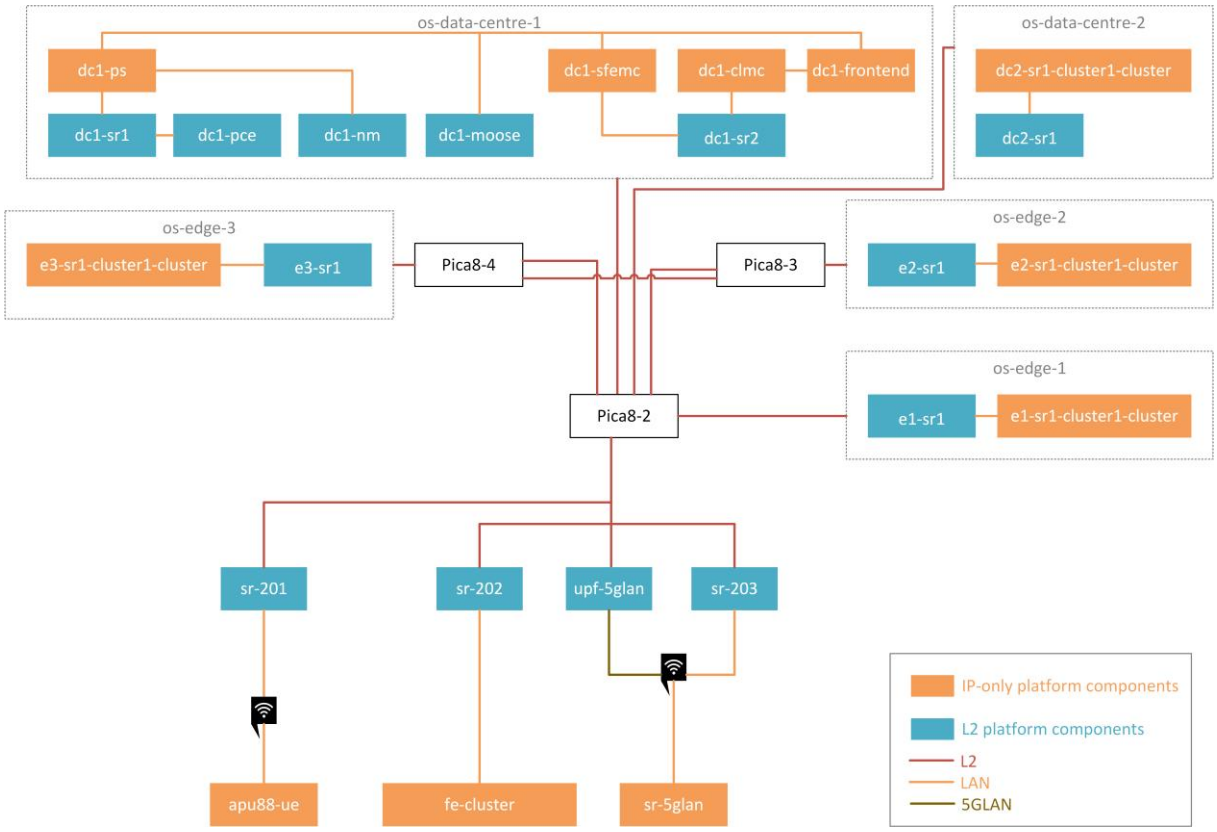

Figure 10: InterDigital's staging testbed

To test the platform, PROTEST is installed on the *apu88-ue* node (bottom left corner in the figure above) which is connected via WiFi to the platform.

2.3.2 Cross-layer Management and Control

The FLAME CLMC is tested and validated at IT Innovation using an extensive set of tests covering different aspects of the component, i.e. set-up, unit-, stress- and system-tests (more information in [FLAME-D4.2]). The tests are run through continuous integration processes coordinated through GitHub and executed on separate containerised testing platforms. With the successful completion of the tests, the CLMC component is merged into a new platform release and tested inside InterDigital's staging testbed by deploying a PROTEST-based SFC to test data points such as CPU load and reported number of HTTP requests. In addition to that, an alert descriptor is given to CLMC to ensure the CLMC <> SFEMC API for monitoring and alerting is working as expected.

2.3.3 Orchestrator and SFEMC

While the orchestrator and SFEMC are being used in LEARNT to deploy, maintain and delete the SFC to conduct the necessary debugging and profiling tasks, their readiness is validated in the staging testbed via dedicated system tests (see [FLAME-D4.2] for more details) that are run in a loop for several hours. This also includes a dummy endpoint issuing triggers to SFEMC mimicking CLMC.

Co-funded by the Horizon 2020
Framework Programme of the European Union

2.3.4 Northbound Service Endpoints of Infrastructure Provider

The two northbound APIs used by the platform are OpenStack and the SDN controller to deploy the platform and to configure the SDN switching fabric, respectively. While OpenStack is more of interest to ARDENT (which is not a platform component), any change in infrastructure and/or logical platform topology requires thorough testing in environments where hardware switches are being used to ensure the rules injected via the SDN controller are translated correctly (see 3.1.2 for more information). The tests piggyback on PROTEST's availability and the addition of emulated UEs deployed as a 2nd stack in OpenStack alongside the platform. Testing various traffic patterns across all possible UE <> SFs connections ensures the rules and the platform in a particular infrastructure are functional.

2.4 PLATFORM REPOSITORY

The platform repository flame-platform has the four main folders:

- **ardent -** The proof-of-concept implementation of ARDENT which follows the work-flow described in 3.2 but not the APIs. This initial implementation does not auto-generate HOTs either.

- **deploy -** This directory comprises the HEAT orchestration templates for each component as well as the site-specific HOT which calls the component HOTs in order to create a single OpenStack stack for the entire platform.

- **Infrastructures -** Inside this directory each infrastructure provider added their scripts to configure their switches or OpenStack in case the step has to be repeated. The infrastructure descriptor for the proof-of-concept implementation of ARDENT is located there too.

- **Src -** This folder contains the scripts to build the images for each platform component.

Documentation is gradually moved to the wiki[5] of the flame-platform, as it would otherwise require a commit and merge request to update any new documentation. The wiki follows a hierarchical structure:

- Infrastructures
    - Bristol
    - Barcelona
    - London (Open Call 2 winner KCL)
    - Sicily (Open Call 2 winner Level7)
    - Sandpit (Southampton)
- Platform
    - Component properties

[5] The wiki can be accessed via https://gitlab.it-innovation.soton.ac.uk/FLAME/consortium/3rdparties/flame-platform/wikis/home. Note, the GitLab wiki is private and requires authentication.

Co-funded by the Horizon 2020
Framework Programme of the European Union

- Deployed versions across infrastructures
- Deployment instructions

Co-funded by the Horizon 2020
Framework Programme of the European Union

3 PLATFORM REPLICATION PROCESS & TOOLS

The FLAME platform being deployed at four sites at the moment: Bristol and Barcelona being the main trial sites, complemented by Southampton for the sandpit-based testing and the InterDigital London deployment for platform testing; two more replicators follow as part of Open Call 2 activities.

The replication process, tooling and its derived best current practices are going to be presented in this section. As part of this replication process, FLAME has designed and implement ARDENT, a toolchain that automates such deployment into a fully programmable infrastructure.

3.1 BEST CURRENT PRACTICES

This section describes the best current practices on how to deploy the FLAME platform into a programmable infrastructure with OpenStack and OpenFlow as the primary APIs towards it. It is worthwhile mentioning that the presented content has been solely developed in a trial-and-error manner, as there is no standard approach at the time of writing. Across all replication sites (including the two Open Call 2 winners), the existing infrastructures are differently managed, serve different purposes and have been used in different contexts, which explains some of the practises being explicitly listed and explained here.

3.1.1 Compute Node Locations

Compared to platforms of telco vendors (RAN solutions) the key difference of the FLAME deployment is the requirement of FLAME platform components to be deployed in particular compute nodes and having compute nodes distributed across the infrastructure, such that they are literally located where they are needed. This can be explained by the fact that a FLAME service router (SR)[6] needs to terminate IP traffic and acts as a TCP proxy for any TCP-based traffic; any communication between SRs is then based on path-based forwarding (see [FLAME-D3.10] for more details). Therefore, in order to leverage the potential of path-based forwarding (service-based routing for HTTP traffic), the placement of SR functionality should be where the traffic is initiated, i.e., at the very edge of the network. Placing those SR compute nodes deeper inside the network would reduce efficiency of traffic forwarding.

The FLAME platform separates service and infrastructure providers from each other by providing its own service orchestration. From a technical side, this is enabled through instances deployed inside the infrastructure's compute nodes, into which service providers are offered to deploy their services into. Those instances are called **clusters** inside the FLAME platform. Based on the location of a compute node and its available resources, it is labelled as either a *data centre*, *edge*, *far edge* or *mist* compute node. ARDENT uses this type of compute tiers to determine which platform components are instantiated on a compute node. More information about this can be found in Section 3.2.3.2.

3.1.2 SDN Switching Fabric

The FLAME platform demands an SDN-enabled switching fabric inside the infrastructure's network in order to program the forwarding rules via OpenFlow (Version 1.3 or above required). While software-based SDN switches[7] do not have any hardware constraints, SDN hardware switches are more limited

[6] The SR implements part of the SFR component of the FLAME platform [FLAME-D3.10]. Said SR is called Service Communication Proxy in the 3GPP SA2 notation [3GPP 23501] for service-based architecture (SBA), while the work in the IETF SFC [SFC2019] defines the SR as the name-based Service Function Forwarder (nSFF)

[7] Open source implementation version 2.10.1 and above from https://www.openvswitch.org

Co-funded by the Horizon 2020
Framework Programme of the European Union

in what and how they support the OpenFlow specification and therefore how well they support the forwarding rules of the FLAME platform.

Specifically, hardware switches translate rules that have been received via OpenFlow into an internal ternary content addressable memory (TCAM) table, which in turn is used by the chipset to match incoming packets against rules for Layer 2 through 4. The TCAM is limited in its size to guarantee the link speed advertised by the vendor, while larger TCAM tables will increase the overall SDN price point exponentially. To support all sorts of tenants, an SDN switch must support switching protocols such as MPLS and VLAN as well as OpenFlow-based rules which results in a hybrid confirmation that essentially splits the TCAM table into SDN and non-SDN areas. Given that TCAMs do offer several thousand entries for rules, it must also be understood that, for example, IPv6 requires higher memory inside the TCAM than IPv4 which further limits the number of rules a TCAM-based SDN switch accepts.

As OpenFlow only supports known header offsets of the IP protocol suite, FLAME semantically overloads the IP (version 6 or 4) header with its rules, which are arbitrary bitmasks instead of longest prefix. These rules, however, when hardware switches are being used, are getting translated into longest prefix rules again, as this is the main semantic of operation that TCAMs and their chipsets have been designed to do in SDN switches. As the arbitrary bitmask allows to have a single incoming packet being matched against multiple output ports, the TCAM table must receive all possible combinations of the arbitrary bitmask being translated into longest prefix again, inflating a single bit position matching rule in many TCAM entries. This action could lead to limitations at hardware switches in the infrastructure that have more than four or five output ports.

Across most deployments the very first dialog between the FLAME platform provider and the infrastructure provider is how to design the “wholesale infrastructure slice” inside which the FLAME platform operates as a tenant to the infrastructure. Without any exception across any replicator, the slice is created using VLAN identifiers. Before diving into the challenges with SDN switches and 802.1a, it must be understood that the FLAME platform does not and must not be aware of any VLAN identifiers used to create the infrastructure slice for the FLAME tenant. The Path Computation Element (PCE) of the FLAME platform calculates paths among SRs and retrieves the topology from the SDN controller. As the PCE expects no switch rules to be inserted on the bridges of the infrastructure’s SDN switches, it first flushes all previously installed ones (from previous FLAME deployments) and inserts the rules needed to enable the path-based forwarding, which essentially is one rule per port[8]. In order to have SDN hardware switches to present a bridge to the SDN controller with access (untagged) ports, the infrastructure provider must untag all packets on a dedicated bridge on the switch first, which is then wired up to the ports for the infrastructure tenant. In SDN mode, the switch does not allow to present access ports to the SDN bridge presented to the tenant (i.e. the PCE).

3.1.3 OpenStack Configuration

From an OpenStack perspective, it is important to understand that the FLAME platform performs the crucial routing of service requests within the platform, which is implemented through the aforementioned SR functionality of the SFR component. With this, the routing of service provider traffic between SRs is located outside of OpenStack, while the platform is being deployed into the infrastructure via OpenStack. However, how to route, security configurations of networks and VM access, is all expected to be programmable through the OpenStack CLI. In comparison to typical Telco-deployments that use OpenStack, this results in some crucial differences regarding potential

[8] In Open vSwitch and when monitoring the OpenFlow protocol FLAME inserts one rule per outport. As states in the same section, the TCAM is implemented for longest prefix only and does translate the rules into a larger set of rules.

Co-funded by the Horizon 2020
Framework Programme of the European Union

performance implications, how Telcos assume security of their VNFs is being configured and how much of what OpenStack provides is disabled and lifted up into the VNF again.

One of the first changes all infrastructure providers have to make is to allow the FLAME tenant to deploy into particular compute nodes, especially when they are all configured into the same availability zone, since it is the FLAME platform that manages and controls the placement of service provider functionality throughout the deployment infrastructure. In a traditional cloud environment this capability is not required as OpenStack determines the most suitable compute node into which the instance should be placed (and reserves the right to change that).

The FLAME platform deployment itself is orchestrated via HEAT templates, which must be enabled as well as the deployment from outside of the controller itself. As the FLAME platform manages the firewall configurations for each provider network it manages, port security must be enabled inside OpenStack as well as the ability to use security groups.

Lastly, the FLAME platform and all clients that attach to SRs act in a single /16 subnet, thus behaving like if they are all part of the same LAN. In order to configure subnets for each SR's LAN provider, network OpenStack must allow the configuration of multiple /24 subnets of the same /16 CIDR.

3.2 ARDENT TOOL

In recent years, the acceptance and adoption of cloud principles in telecommunication systems has significantly increased, with companies such as Suse, RedHat and Canonical offering the required knowledge and solutions (as-a-service) to operators and infrastructure providers. The shift towards VNFs and cloud-native deployments replicates the successful model of cloud companies and how they can scale their services into a business sector which traditionally preferred physical boxes and functions with a clear one-to-one relationship in execution and state handling. This opens the floor for fully software-based approaches that can operate on commodity hardware. While this shift in itself demands a significant change of how telecom solutions are architecturally designed and built, the FLAME platform and its internal components are a perfect example of how customer service deployment but also the routing of service requests can be softwarised to offer a truly innovative platform. With this in mind, the deployment of such a platform into a cloud-based infrastructure is challenging when enforcing and demanding a truly programmable infrastructure, which includes the programmability of security, switches, and hybrid cloud resources.

In order to automate the actual deployment process of a platform that requires particular components at specific (edge) cloud locations, an **Automated platfoRm DEploymeNt Toolchain** (ARDENT) has been designed, specified and implemented as a proof-of-concept for the deployments of the FLAME platform in Barcelona and Bristol. Additionally, ARDENT is also in use for the deployment of the FLAME platform into the sandpit in Southampton and for the deployment of the FLAME platform into InterDigital's testing lab infrastructure.

Initially, a proof-of-concept was developed using Bash and AWK. Once ARDENT moved more into a flexible toolkit to deploy (and replicate) the platform across sites or accommodating changes in the infrastructure, ARDENT was fully re-engineered from scratch using Golang and MySQL to provide a standalone application with RESTful service endpoints to offer a more service-oriented approach to the deployment of a 5G platform.

Co-funded by the Horizon 2020
Framework Programme of the European Union

3.2.1 Purpose and Workflow

The figure below illustrates the ARDENT workflow to deploy the FLAME platform. The four circles represent the major steps to achieve this goal and the responsibility is split between the infrastructure and platform providers, as indicated.

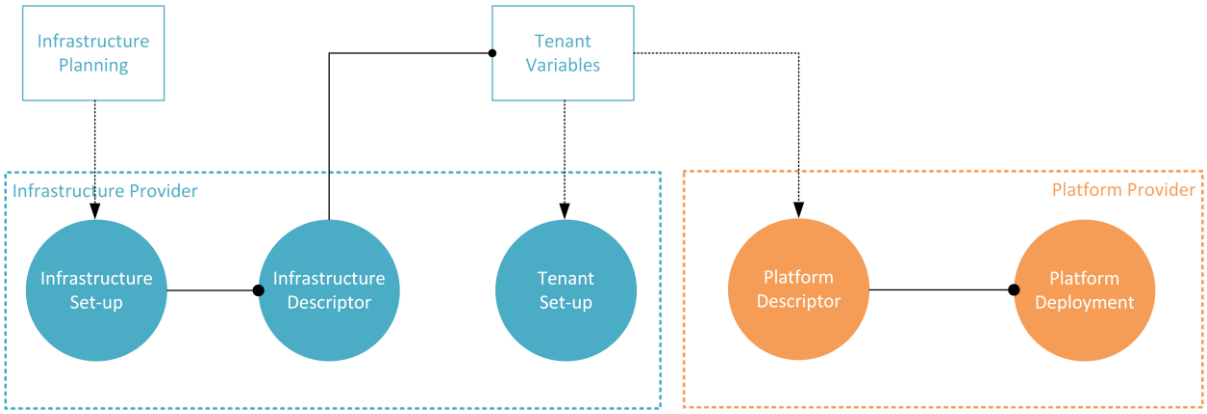

Figure 11: Workflow for deployment of platform using infrastructure and platform descriptors

The very first step for an infrastructure provider is to **plan** the resources that should be given to the infrastructure tenant, i.e., compute, storage and networking. The creation of the various networks for data plane as well as management plane is part of this procedure and results in the **infrastructure set-up**.

The **infrastructure descriptor** is a YML-based descriptor communicating the components of the infrastructure topology (compute nodes, infrastructure services, SDN switching fabric) as well as their physical connection and the names of the provider networks.

From the infrastructure descriptor the **tenant variables** are derived and populated in a knowledge base which is shared with the next three tasks, i.e. setting up the tenant, creating the platform descriptor and eventually deploying the platform.

The **tenant set-up** is a collection of bash scripts with several variable files which comprise infrastructure provider-independent and infrastructure provider-specific values. The scripts allow the infrastructure provider to a) set up the FLAME tenant using an automated procedure and b) to share all necessary values that describe the NFV platform and allows the platform provider to write the platform descriptor and to deploy the platform eventually.

Given that OpenStack is the chosen NFV realisation the **platform descriptor** is a HEAT-based YAML file describing where certain platform instances are deployed and which networks they attach to. The information required to write the platform descriptor is taken from the infrastructure descriptor and the variable files written during the tenant set-up activity.

The last step is **platform deployment** where the platform descriptor is given to OpenStack via its CLI.

3.2.2 Networks, Subnets and Security Groups

Figure 12 illustrates the provider networks including their purpose on the infrastructure level.

Co-funded by the Horizon 2020
Framework Programme of the European Union

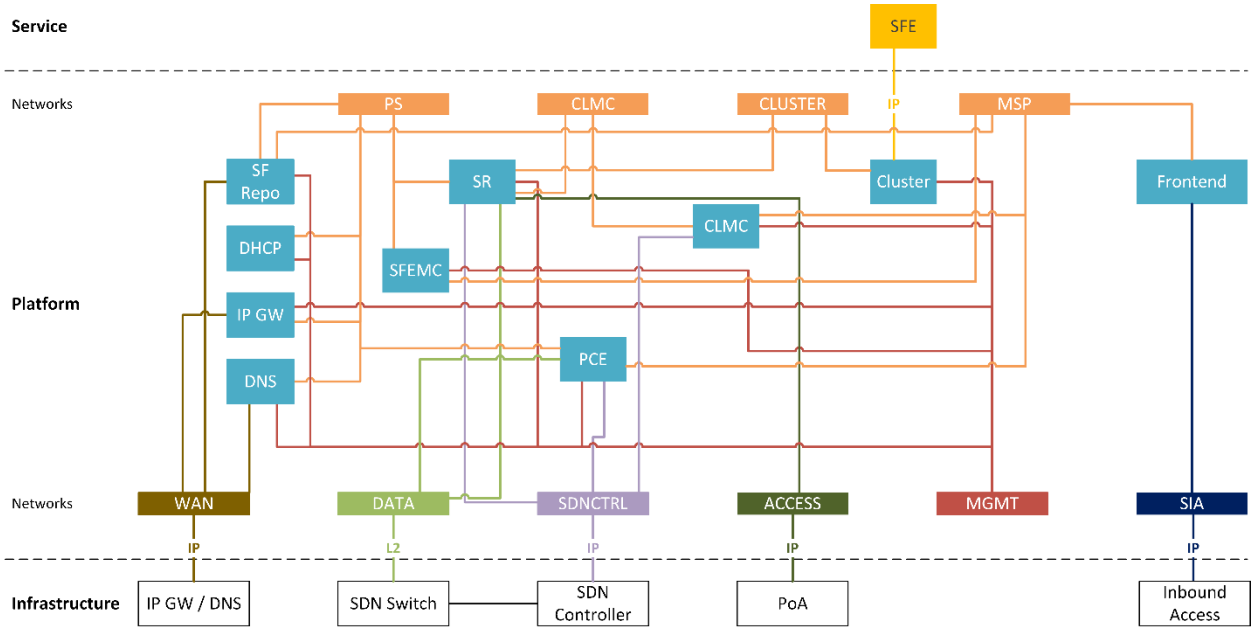

Figure 12: Infrastructure Tenant Networks and their Purpose

The platform networks allowing access to infrastructure resources are:

- **WAN**: The network to access the internet through one or more IP gateways of the infrastructure provider and a dedicated DNS (if provided). If the infrastructure provider does not maintain their own DNS, a public DNS is used (e.g., OpenDNS or Google). This network must be configured with DHCP, a gateway and a DNS server
- **SDNCTRL**: The network to reach the SDN controller from any deployed platform instance. This network must be configured with DHCP but no gateway or DNS server
- **DATA**: The network for interconnecting the compute nodes via the underlying SDN switching fabric of the infrastructure provider. Note, most infrastructure providers create dedicated DATA networks between a SDN switch port and a compute node often using VLANs. Therefore, the number of DATA networks very often equals the number of SDN ports configured for the platform. This network is treated by the platform as an L2 link and does not require any IP configuration in the NFV platform. If the infrastructure provider does not own its own SDN switching fabric, a single DATA network must be created to allow platform instances to communicate with each other.
- **ACCESS**: The network which allows the platform to handle traffic from end devices connected to the infrastructure via point of attachments (WiFi, cellular, cable). Similar to the DATA network, the ACCESS network is most likely also configured using VLANs and one network per Point-of-Attachment (PoA) is therefore created to connect the PoA and specific compute node. The IP assignment of IP endpoints attached to this network is done via a platform service rather than an infrastructure one. The platform provider refers to this network as the LAN.
- **SIA**: This network provides a *secured inbound access* (SIA) towards the FLAME frontend platform instance, which allows service providers to deploy and maintain their service function chains.

For managing the platform instances a MGMT network is required which allows the platform provider to manage and maintain its deployed platform instances.

Furthermore, the following platform networks are required:

- **CLMC**: The network that allows to let the CLMC and its SR to communicate via IP.

Co-funded by the Horizon 2020
Framework Programme of the European Union

- **CLUSTER**: The network that allows a cluster instance to communicate via IP with its SR.
- **PS**: The network that allows the PCE/SFEMC instance and all platform service instances to communicate with their SR.
- **MSP**: The network that allows a media service provider to access the orchestrator, CLMC and the service function repository to create, maintain and configure their deployed service function chains.

The table below lists the various networks, subnets and security groups described above and summarises their properties and when they are created by whom during the workflow, as described in Section 3.2.1. If a security group is not configured for ports on a provider network the port security must be disabled. Note, the "created when" column refers to the steps illustrated in Figure 11.

Table 1: Networks, Subnets and Security Groups for FLAME Platform

Network Type	Created By	Subnet		DHCP	IP GW	DNS	Security Group	
		Configured	Created by				Configured	Created by
WAN	Infrastructure Provider	Yes	Infrastructure Provider	Yes	Yes	Yes	Yes	Infrastructure Provider
SDNCTRL	Infrastructure Provider	Yes	Infrastructure Provider	Yes	No	No	Yes	Infrastructure Provider
DATA	Infrastructure Provider	No	---	---	---	---	No	Infrastructure Provider
SIA	Infrastructure Provider	Yes	Infrastructure Provider	Yes	No	No	Yes	Infrastructure Provider
ACCESS	Infrastructure Provider	Yes	ARDENT	Yes	No	No	No	Infrastructure Provider
MGMT	Infrastructure Provider	Yes	Infrastructure Provider	Yes	No	No	Yes	Infrastructure Provider
CLUSTER	Infrastructure Provider	No	ARDENT	Yes	Yes	Yes	No	Infrastructure Provider
PS	Infrastructure Provider	No	ARDENT	Yes	Yes	Yes	No	Infrastructure Provider
MSP	Infrastructure Provider	No	ARDENT	Yes	No	No	Yes	Infrastructure Provider

3.2.3 Automations Performed by ARDENT

In Telco-oriented infrastructures various types of edges exist which are categorised into data centres, edge and far edge. Most likely the following compute node characteristics for each type of tier applies:

- *Data Centres*: Compute nodes of this type are far away from any point of attachment and are high in compute resources.
- *Edge*: Compute nodes of this type are usually near infrastructure switches where the underlying network branches off into different directions to reach areas of PoAs.
- *Far Edge*: Compute nodes of this type are one or two hops away from a PoA to keep latency and jitters at a bare minimum, supporting VR-type of services.
- *Mist*: Compute nodes of this type have very constraint cloud resources and are located at the very front of the network.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Once the infrastructure descriptor has been communicated to ARDENT there are several automations taking place, eventually resulting in a single HEAT template describing the deployment of the platform. In order to achieve such automation, there are certain criteria which help to maximise the constraints, such that the set of options becomes limited.

Certain set of platform components are grouped together and are deployed on the same compute node. Those groups are:

- SR, PS and PCE
- SR and CLMC
- SR and Cluster
- Monitoring Server (MOOSE)
- Frontend
- SR_{PoA} (providing access to UEs)

The aforementioned groups can be templated as HEAT orchestration templates (HOTs) which are called by an overarching HEAT orchestrator template, which feeds missing values for various types of variables into it. Given the requirement of usage for various platform components, their templated HOTs can be directory mapped to a tier, as shown in the table below

Table 2: Platform components per compute node tier level

Tier	Platform Components
Data Centre	- SR, PS, NM and PCE - SR, SFEMC and CLMC - SR and Cluster - MOOSE - Frontend
Edge	- SR and Cluster
Far Edge	- SR and Cluster - SR_{PoA}
Mist	- SR_{PoA}

3.2.3.1 CIDR and Routing Prefixes for SRs

The infrastructure provider communicates the CIDR to the platform provider via the infrastructure descriptor. The CIDR for the platform must be a /16 or larger one, as each SR requires a /24 subnet. Consequently, a `a.b::16` CIDR allows ARDENT to generate 254 SRs with routing prefixes of format `a.b.n::24` where n={1,254}.

While it does not quite matter which SR receives which routing prefix, it has become a common practice to have the SR serving the internet to receive n=1, the SR serving the SFEMC/CLMC n=2, followed by the SRs for all clusters and then the SRs for PoA/access networks.

All SR and cluster OpenStack instances perform DHCP to receive their IP addresses. Those IP addresses are assigned by OpenStack's DHCP service. However, for deployed service function endpoints (as LXD- or KVM-based instances inside clusters) or UEs attached to access networks, the IP addresses are handled by the platform's DHCP server inside the PS node and it must be ensured that the two IP addressing spaces (in OpenStack and in DHCP server) do not clash. Therefore, all SRs and clusters have a subnet configured of range a.b.c.0/29 with Class D values of .1 and .2 reserved for OpenStack; consequently, the range configured is .3 through .7.

Co-funded by the Horizon 2020
Framework Programme of the European Union

3.2.3.2 Cluster Flavours and Compute Node Categories

As mentioned in the introduction of this section, all flavours except the ones for clusters are fixed. The reason for that is to max out the wholesale slice configured by the infrastructure provider. In the following, we present those compute node categories in more detail.

Data Centre As listed in Table 2, all control functions of the platform are placed into compute node(s) of type data centre such as PCE, PS, CLMC, SFEMC and frontend. The flavour properties for each aforementioned type is listed below.

If only one data centre compute node is available, then all control functions of the platform are placed into the single node. If the RAM is not sufficient enough to host the memory intense instance CLMC, its RAM requirement can be reduced to half of its size if necessary. Then, the data centre cluster can be calculated by adding an additional SR for serving the cluster and give all the remaining vCPUs, RAM and disk to the cluster. If the resulting cluster flavour is less than 2 cores, 2GB RAM or 2GB disk the cluster is not created by ARDENT.

If the data centre is composed of more than one compute node, then all control functions (and their SRs) are placed on one compute node and each other compute node receives an SR with a cluster which maxes out the resources on each.

Unless the WAN provider network has not been made available to all data centre computer nodes, ARDENT uses the first data centre compute node as the one which hosts the control functions (PS in particular). If the WAN provider network has been made available at a particular compute node only, then ARDENT uses that one for all control functions.

Note, SRs and their IP endpoints (clusters and PSs in particular) must never be placed on different compute nodes to avoid performance bottlenecks of the link (network card) between the SR and its IP endpoint.

Table 3: Flavour properties for control functions of the platform

Node Type	vCPUs	RAM [MB]	Disk [GB]
PCE	1	1536	15
PS	1	1024	100
SFEMC	1	1024	15
CLMC	4	32768	100
MOOSE	1	1536	15
Frontend	1	1024	5
SR	1	1024	10

Edge Compute nodes of this type host SR and cluster bundles only, as these compute nodes do not have access networks attached to them due to their physical placement in the infrastructure's network. Hence, each edge compute node receives a single SR and a cluster that maxes out the cloud resources.

Far Edge This compute node has access networks attached to it but it is still large enough to host a SR+cluster bundle. As not all far edge compute nodes are physically located at the PoA for UEs, multiple access networks could be configured at a particular far edge compute node. In order to offer a cluster per PoA in a far edge scenario, the following methodology must be applied to determine the number of clusters and SRs on a far edge node: first each access network receives a SR. The remaining cloud resources are then divided by the number of access networks to be given to a SR+cluster bundle.

Co-funded by the Horizon 2020
Framework Programme of the European Union

As the specification for a different compute node of the same category can be different, ARDENT creates a cluster flavour per data centre/far edge compute node.

Mist This compute node has access networks only attached and therefore hosts SRs only. Taking the cloud resource requirements for SRs into account, ARDENT creates one SR_{PoA} per access network on a compute node of tier mist. If the resources are insufficient ARDENT will not deploy the stack.

3.2.3.3 Fixed IP Addresses

The platform nodes PS, frontend, SFEMC, CLMC, and MOOSE have some ports with fixed IP addresses assigned to them for various purposes. Those IPs are derived from the routing prefix the SR has been configured to:

Table 4: Fixed IP addresses

Node	Port Type	IP Address
PS	LAN	a.b.n.1
PS	MSP	c.d.e.<MIN>
SFEMC	MSP	c.d.e.<MIN+1>
CLMC	MSP	c.d.e.<MIN+2>
MOOSE	MSP	c.d.e.<MIN+3>
Frontend	MSP	c.d.e.<MIN+4>

with ‘e’ being the highest class C value possible from the CIDR provided by the infrastructure provider, i.e. 255 for a /16 CIDR. As the MSP subnet is a /24 again, ARDENT determines <MIN> (Class D value of the IP address) as .3 leaving .1 for the IP GW service in OpenStack for this subnet and .2 for the DHCP service.

3.2.4 Architecture

The figure below illustrates the architecture of ARDENT with components and interfaces. Following the workflow described in Section 3.2.1, ARDENT has three main groups of interfaces:

- Infrastructure provider to submit the infrastructure descriptor
- OpenStack CLI to configure the admin and tenant environment
- Interface into ARDENT to customise platform stack

These components and interfaces are described in Section 3.2.4.1 and 3.2.4.2, respectively.

Co-funded by the Horizon 2020
Framework Programme of the European Union

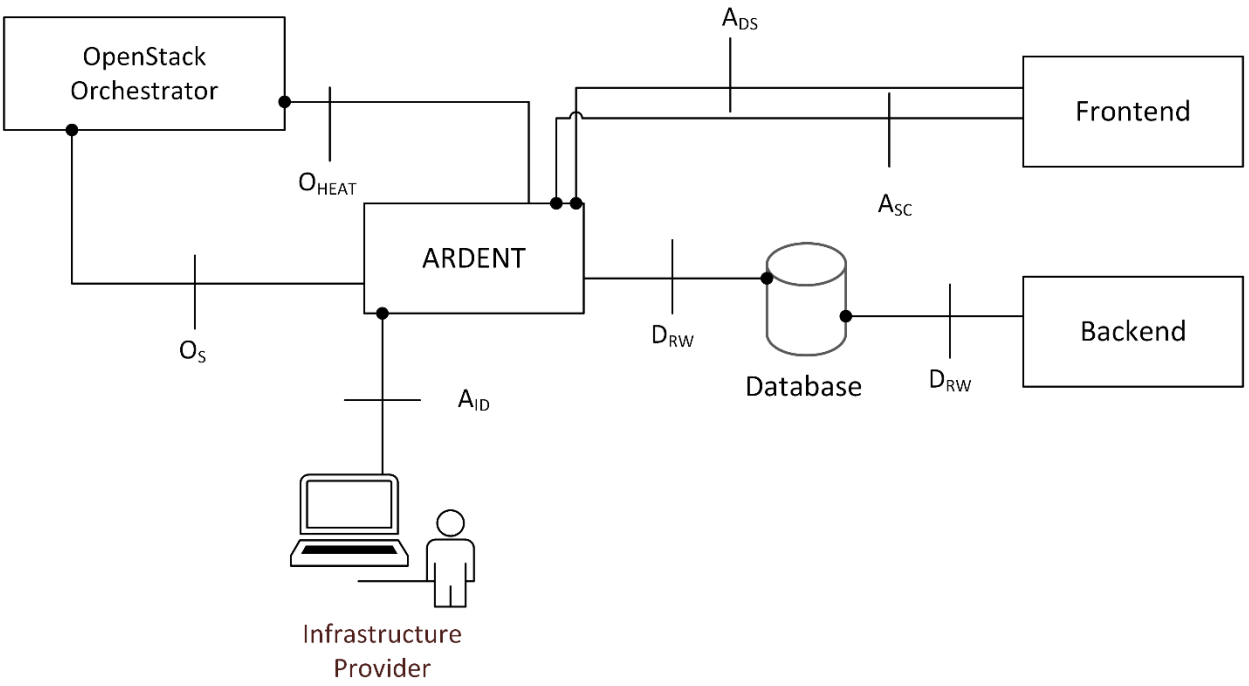

Figure 13: ARDENT Architecture

3.2.4.1 Components

This section describes the components of the ARDENT architecture including their purpose and scope.

Infrastructure Provider

Instead of a pure software component this element is a person or a group of people. They write the infrastructure descriptor, which communicates various aspects of the infrastructure that an OpenStack tenant cannot reverse engineer when using the OpenStack CLI only. The information includes:

- Compute node names and their tier level
- Network names and their type
- Subnet names and their type
- Other variables such as infrastructure services, CIDR, MTU, tenant identifier and IP address of the OpenStack controller

ARDENT Application

This component represents the actual process that offers the various service APIs and executes the changes an ARDENT user configures via the GUI.

Database

The database holds all states the ARDENT application obtains and/or changes and allows the GUI component to obtain the required information to visualise the stack across all tiers.

Co-funded by the Horizon 2020
Framework Programme of the European Union

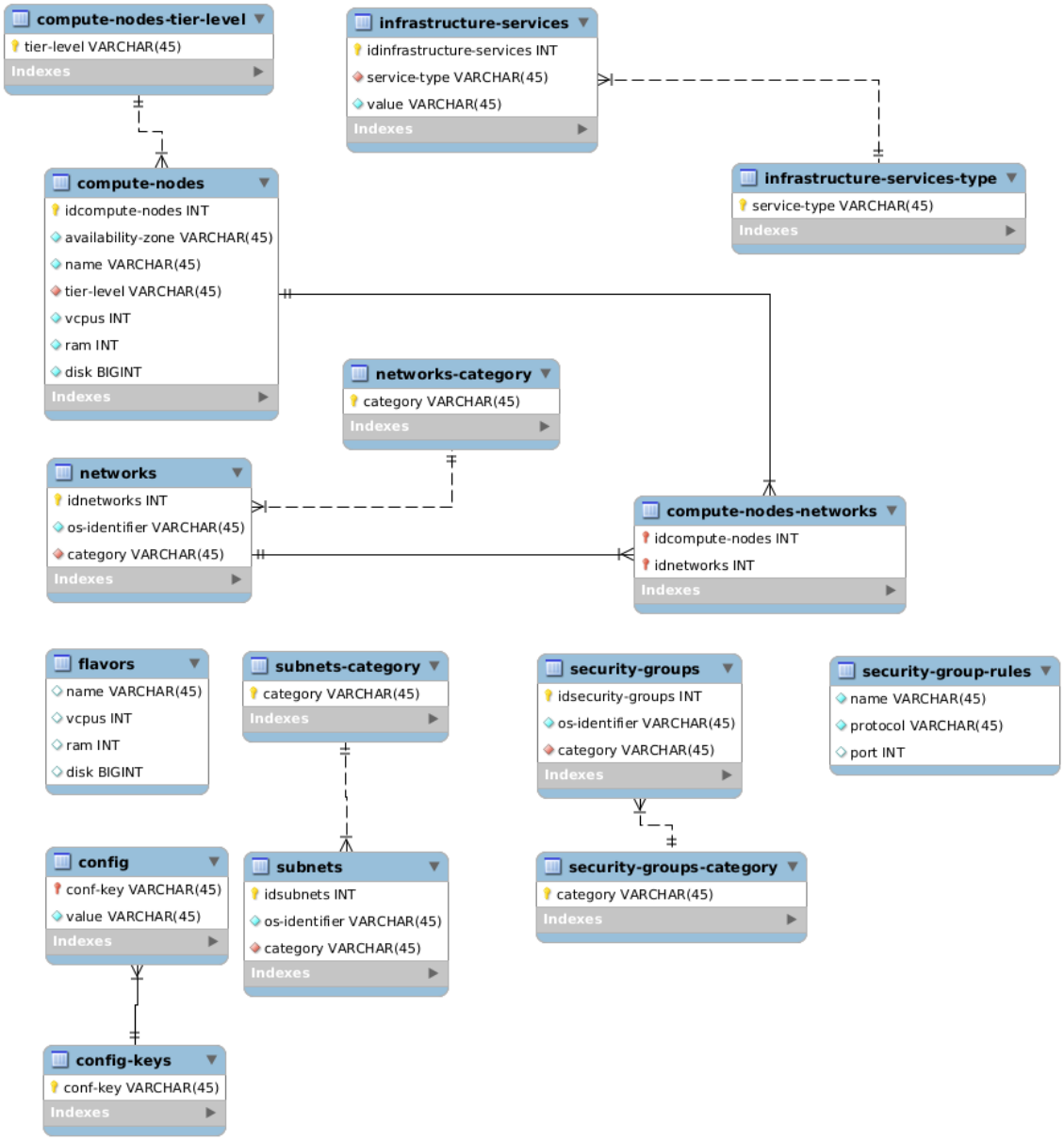

Figure 14: ARDENT database schema

Backend

This component provides the ability to interact with the database on behalf of the frontend. Given that there is no graphical frontend foreseen anytime soon the realisation of this component is embodied by phpMyAdmin.

Frontend

This component allows the platform provider to obtain information about the HEAT orchestration template, which was determined by ARDENT after having received an infrastructure descriptor. Furthermore, the frontend allows to conduct sanity checks and to create/delete stacks. For simplicity reasons this component provides a CLI only.

The graphical user interface of ARDENT allows the platform provider to get a visual representation of the (to be) deployed platform stack. The servers of the platform stack are categorised using the compute nodes and their tier levels (data centre, edge, far edge). Furthermore, the GUI illustrates the

Co-funded by the Horizon 2020
Framework Programme of the European Union

provider networks used by an OpenStack server, as illustrated in the example sketch of the GUI in Figure 15.

The GUI also allows to configure where to find the openrc files to authenticate as an infrastructure provider (optional) and/or platform provider (mandatory). If the infrastructure provider has not permitted access to the OpenStack CLI as admin, this option is disabled.

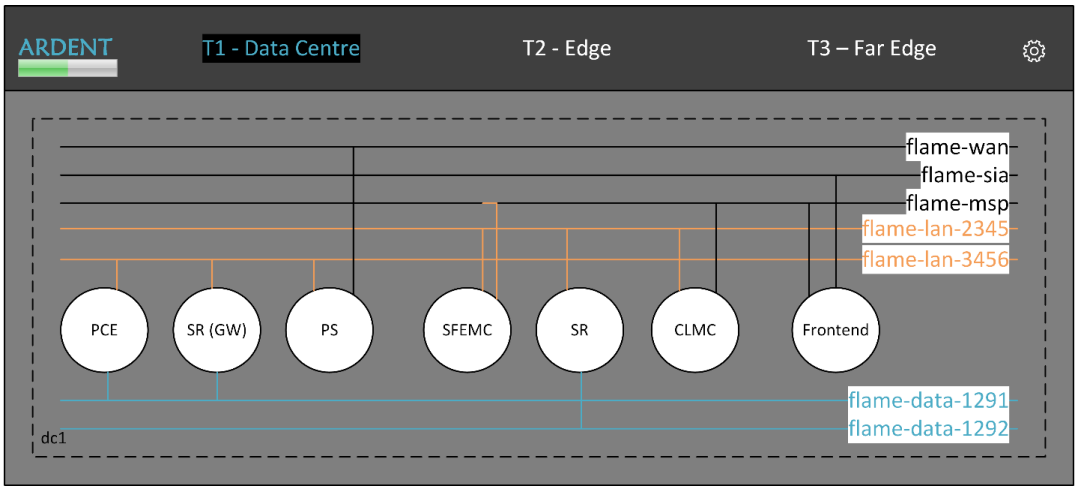

Figure 15: Example sketch of ARDENT's GUI

The GUI also allows to move servers into different compute nodes of the same tier level.

OpenStack Orchestrator

The OpenStack orchestrator provides a service API to read, set, configure and deploy the resources of the NFV environment and any deployed stack/server under the scope of the client's permissions.

3.2.4.2 Interfaces

This section defines the purpose of each interface and any technical detail required to independently implement any side of the service.

All RESTful service endpoints follow standard conventions of the usage of HTTP methods (POST, PUT, DELETE) and REST resources. The resources for ARDENT have been simplified to:

/<ENDPOINT_ACRONYM>/<ACTION_OR_ITEM>

With *<ENDPOINT_ACRONYM>* being either *infra*, *hot*, *sanity-check*, or *stack*. The *<ACTION_OR_ITEM>* field either identifies the action the service endpoint is acting upon (e.g. create or generate) or the item which should be retrieved or deleted. Note, as some service endpoints are proxying actions towards OpenStack the deletion action is a special use case and ARDENT follows the following convention: For items that are owned by ARDENT, such as the HEAT orchestration template or the infrastructure descriptor, the ARDENT service endpoint uses the HTTP method *DELETE* to remove them. For items that are proxied into OpenStack, such as the deletion of a stack, the HTTP method *POST* must be used.

A_{GH}

This interface provided by the ARDENT component allows to generate the HEAT orchestration template based on the information stored in the database. This step includes:

Co-funded by the Horizon 2020
Framework Programme of the European Union

- The generation of a random (non-dictionary) password for all platform instances of 12 characters

A_{DS}

This interface provided by the ARDENT component allows to orchestrate stacks as an OpenStack tenant. The interface allows to create and delete stacks but does not perform changes to existing stacks without deleting them first.

Type: RESTful web service
Port: 8181

Table 5: RESTful web service of A_{DS} interface

URI	HTTP Method	JSON Fields		
		Name	**Type**	**Description**
/stack/create	**POST**	*name*	*string*	The name of the stack that shall be deployed
/stack/delete	**POST**	*name*	*string*	The name of the stack that shall be deleted
/stack/status/<NAME>	**GET**	*status*	*string*	Obtaining the status of a stack as a JSON response

A_{IP}

This interface allows an infrastructure provider to upload their infrastructure descriptor as well as openrc runcom files to allow ARDENT to use the OpenStack CLI for the steps outlined in Section 3.2.1.

Type: HTML multi-form file upload service
Port: 8181

Table 6: RESTful web service of A_{ID} interface

URI	HTTP Method	Description
/infra/descriptor	**POST**	Allows to upload the infrastructure descriptor. The HTTP response comprises a single JSON field error with string values being NULL for no error (infrastructure descriptor accepted) or a string with an error message (descriptor rejected).
/infra/rc/admin	**POST**	Allows to upload the openrc to allow ARDENT to use the OpenStack CLI as an admin. The upload of this file is optional.
/infra/rc/tenant	**POST**	Allows to upload the openrc runcom to allow ARDENT to use the OpenStack CLI as a tenant. The upload of this is mandatory.

A_{SC}

This interface offered by the ARDENT application component allows the GUI component to invoke a sanity check of the infrastructure.

Type: RESTful web service
Port: 8181

Table 7: A_{SC} RESTful interface description

URI	HTTP Method	Description

Co-funded by the Horizon 2020 Framework Programme of the European Union

/sanity-check	**POST**	Allows to invoke a sanity check of the infrastructure
/sanity-check/status	**GET**	Allows to obtain the status of a sanity check. The response is a single JSON-formatted field *status* and a string value.
/sanity-check/results	**GET**	Allows to obtain the result of the last sanity check

The sanity check conducts the following checks

- Existence of networks at the compute nodes described in the infrastructure descriptor
- Port security settings for provider networks based on Table 1: Networks, Subnets and Security Groups for FLAME Platform
- Existence of security groups described in the infrastructure descriptor
- Existence of all security group rules
- Quotas and report on their configuration in case they are insufficiently configured, as only admin users are allowed to do so

The results can be queried separately via the dedicated REST resource /sanity-check/results.

D_{RW}

This interface offered by the database component allows the GUI component to read and write values into the database.

Database type: MySQL
Database Listening Port: 3306

O_S

This interface allows ARDENT to use the OpenStack CLI as an admin in order to:

- Read/change port security settings on existing networks
- Create/modify flavours
- Read/set tenant quotas

Note, this interface is only being used if the infrastructure provider agreed on the use of the OS CLI as admin by the platform provider. If not, the infrastructure provider must ensure the documented tenant requirements have been manually configured.

By default, OpenStack has all quotas for the tenant configured to rather low numbers which have to be adjusted based on the size of the platform. The following quotas are calculated by ARDENT based on the infrastructure descriptor and the internal automations to determine where which platform components are deployed.

Quota	Description
Cores	The number of vCPUs used by the platform. This is essentially the sum of all vCPUs across all compute nodes
RAM	The amount of RAM the platform will use. This is the sum of the memory across all compute nodes.
Instances	The number of instances the platform will be composed of.
Subnets	The number of subnets the platform will be creating. Note, the subnets for the SIA and the WAN provider network need to be included in this quota number even though they were created by the infrastructure provider.

Co-funded by the Horizon 2020
Framework Programme of the European Union

Ports	The number of ports the platform will be using. Each interface of an instance counts as a port independently of whether *network* or *port* is being used in the HEAT orchestration template.
Security Groups	The number of security groups the platform will need. Note, this includes the security groups for ports on the WAN and SIA provider networks.
Security Group Rules	The sum of security group rules ARDENT requires across all security groups. Note, this includes any rules added by the infrastructure provider for the provider networks they are maintaining, i.e. WAN and SIA.

While OpenStack tenants can read the current quotas, only admins are allowed to change them. Thus, if ARDENT did not receive the adminrc file through the A_{ID} interface it can only check if the quotas are sufficient enough. If not, the sanity check (see interface A_{SC}) will report upon such misconfiguration.

O_{HEAT}

This interface is used to create and delete stacks via HEAT orchestration templates (HOT). The interface is also being used to read the configuration of deployed servers.

3.2.4.3 Dependency Graph

The chart below illustrates the dependency of processes behind the RESTful service endpoints.

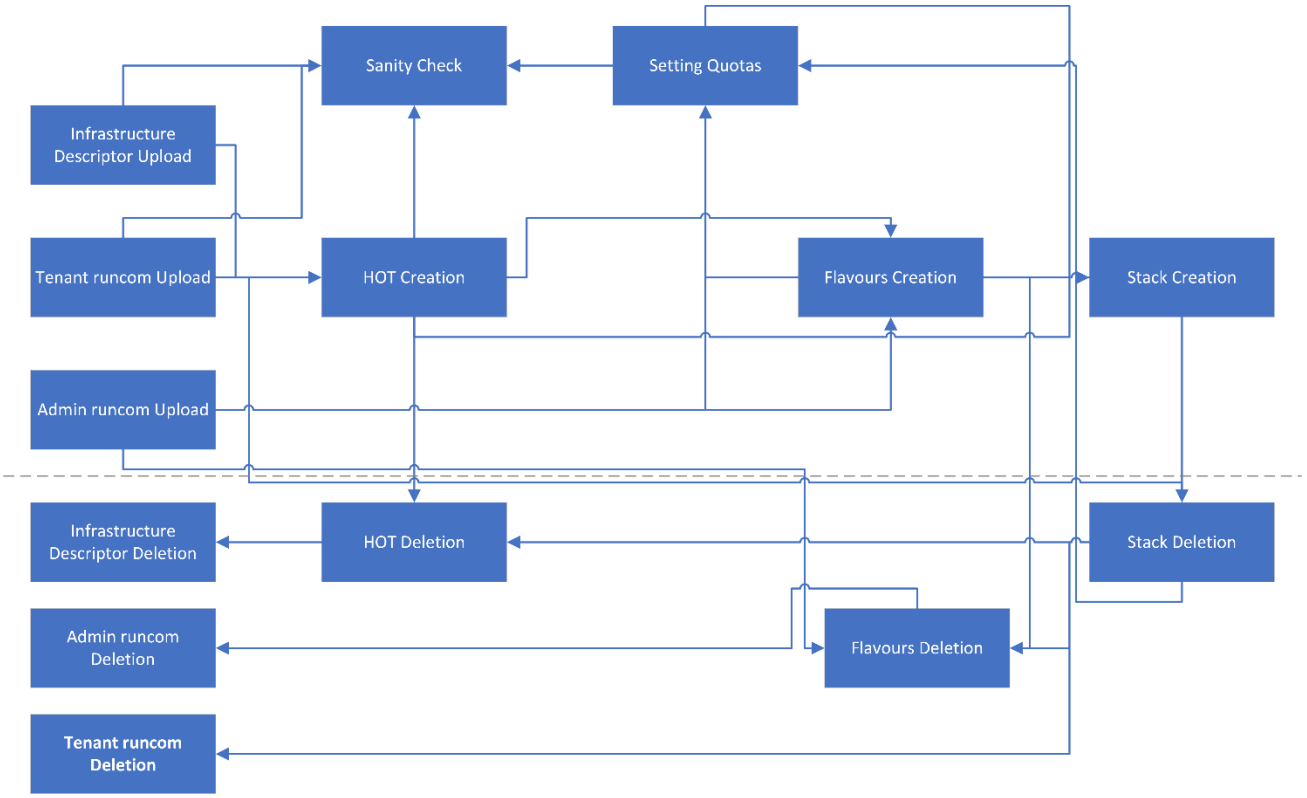

Figure 16: Process Dependencies

3.2.4.4 Infrastructure Set-up

The FLAME platform has certain requirements when it comes to the desired networks, as outlined above. However, the automation of the infrastructure tenant set-up and the deployment of the platform eventually cannot include the configuration of the infrastructure itself and the creation of certain resources inside the NFV environment due to the vast amount of possibilities and preferences

Co-funded by the Horizon 2020 Framework Programme of the European Union

by an IT department. Hence, the set-up of the infrastructure must be carried out using preferred methodologies of the infrastructure provider. This section briefly highlights the requirements of the FLAME tenant when it comes to networks and security groups created and configured in the NFV framework.

The following networks must exist before the infrastructure descriptor is written by the infrastructure provider:

- WAN
- SDNCTRL
- DATA
- ACCESS (only if there's an external access to the platform)
- MGMT
- SIA
- MSP
- CLUSTER
- PS
- CLMC

The configuration details required by FLAME can be found in Table 1 including the security groups.

Note, it is important that the WAN and MGMT networks are accessible from the deployment node provided to the platform provider.

3.2.4.5 Infrastructure Descriptor

The infrastructure descriptor allows infrastructure providers to describe their infrastructure so that ARDENT can build the HEAT orchestration templates accordingly.

The entire YML definitions file for the infrastructure descriptor is given in Appendix 5.1. Note, all YML nodes and keys are required.

Compute Nodes

The YML node *compute_nodes* is a stream of YML nodes with arbitrary but unique names with each of them comprising a list of keys that describe the compute node. The names can be the hostnames of the compute nodes (also reflected in the list of hypervisors). If another convention is chosen by the infrastructure provider, it would be helpful for debugging purposes to still use meaningful names.

The keys are listed in the table below.

Table 8: Keys for the YML node compute_node

Key	Type	Description
availability_zone	string	The availability zone for this OpenStack compute node.
name	string	The name of the OpenStack compute node.
vcpus	integer	The number of vCPUs that have been allocated to the tenant's wholesale slice.
ram	integer	The amount of RAM that has been allocated to the tenant's wholesale slice. Unit in MB.
disk	integer	The amount of disk that has been allocated to the tenant's wholesale slice. Unit in GB.

Co-funded by the Horizon 2020 Framework Programme of the European Union

networks	array	The list of networks this compute node has access to. Networks are identified by their identifier, not by their name.
tier	string	The tier this compute node represents. The values are data_centre, edge, far_edge, or mist. More information on the meaning of each tier can be found in Section 3.2.3.2.

Example:

```
compute_nodes:
    dc1:
        availability_zone: nova
        name: os-data-centre-1
        vcpus: 38
        ram: 29000
        disk: 800
        networks:
          - 1449337d-3c21-471d-aa9b-db857cf3653e
          - 058319d1-ec1a-46c1-bc97-3631f4fa1167
          - 2765f604-9f5a-4bf8-b859-cb571f55762f
          - 83d68f68-b36f-4bfc-9eb9-3fae3f9004e6
          - 948cc7ef-1728-4fb0-9b1f-d659b02527e2
          - e293a6c5-e9b9-4d9c-8ec2-c3f8083db56c
          - f643dbc2-42e3-47ec-9d1a-47253dd1212a
          - 716c64ee-0dae-4c54-931b-492e3bc81195
        tier: data_centre
    dc2:
        availability_zone: nova
        name: os-data-centre-2
        vcpus: 38
        ram: 29000
        disk: 800
        networks:
          - 1449337d-3c21-471d-aa9b-db857cf3653e
          - 2765f604-9f5a-4bf8-b859-cb571f55762f
          - e293a6c5-e9b9-4d9c-8ec2-c3f8083db56c
          - f643dbc2-42e3-47ec-9d1a-47253dd1212a
          - 645e4a7a-6d44-4588-acb6-96adaa993916
        tier: data_centre
```

Networks

The YML node network has a stream of arbitrary but unique network YML nodes with two keys, as listed in Table 9.

Table 9: Keys for the YML node network

Key	Type	Description
identifier	string	The OpenStack identifier of this network.
category	string	The network category of the network. Valid values are *access*, *clmc-sfemc*, *cluster*, *data*, *mgmt*, *msp*, *ps*, *sdnctrl*, *sia*, or *wan.*

Example:

```
networks:
    flame-mgmt:
        identifier: 1449337d-3c21-471d-aa9b-db857cf3653e
        category: mgmt
    flame-msp-dc1:
        identifier: 058319d1-ec1a-46c1-bc97-3631f4fa1167
```


Co-funded by the Horizon 2020
Framework Programme of the European Union

```
        category: msp
```

Subnets

The YML node *subnets* has a stream of arbitrary but unique subnet YML nodes with the keys listed in Table 10.

Table 10: Keys for the YML node subnet

Key	Type	Description
identifier	string	The OpenStack identifier of this subnet.
category	string	The category of this subnet. Valid values are *sia*.

Example:

```
subnets:
    flame-sia:
        identifier: 6be747bc-7914-46d2-b81b-892bb504cb9f
        category: sia
```

Security Groups

The YML node *security_groups* has a stream of arbitrary but unique security group YML nodes with the keys listed in Table 11.

Table 11: Keys for the YML node security_groups

Key	Type	Description
identifier	string	The OpenStack identifier of this security group.
category	string	The category of this security group. Valid values are *mgmt*, *msp*, *sdnctrl*, *sia* or *wan*.

Example:

```
security_groups:
    flame-mgmt:
        identifier: 8361221f-b55e-470f-85ae-3f1100430b50
        category: mgmt
    flame-msp:
        identifier: d7e362f0-b290-4797-8862-66380e407905
        category: msp
```

Infrastructure Services

The YML node *infrastructure_services* has a list of keys described in Table 12.

Table 12: Keys for the YML node infrastructure_services

Key	Type	Description
ip_gateway	string	IP address of the gateway of the infrastructure provider to access the internet.
dns	string	The IP address of the domain name server of the infrastructure provider to resolve FQDNs not served by the platform.
sdn_controller	string	The IP address of the SDN controller for the SDN switching fabric of the infrastructure provider.

Example:

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
infrastructre_services:
    ip_gateway: 10.6.224.1
    dns: 10.6.64.254
    sdn_controller: 172.50.231.137
```

Metadata

The YML node metadata has a list of keys described in Table 13.

Table 13: Keys for the YML node metadata

Key	Type	Description
tenant	string	The OpenStack tenant identifier that has been assigned to the platform provider.
cidr	string	The CIDR for the entire LAN the platform provider can use freely.
mtu	string	The MTU of the infrastructure for the ports of the platform on the DATA network.
sia-ip-frontend	string	The IP address the frontend node should be configured to on the SIA provider network

Example:

```
metadata:
    tenant: 73e73c407d0940fc9e7ef08a1c3f568b
    cidr: 192.168.0.0/16
    mtu: 1492
    sia-ip-frontend: 172.70.231.199
```


Co-funded by the Horizon 2020
Framework Programme of the European Union

4 CONCLUSIONS

This report has described the technical roadmap for complementary technology and processes developed within the FLAME project supporting accelerated replication. Those complement the FLAME platform in providing the necessary agile lifecycle development as well as replication processes and tools, stemming from our insights gained through the replication itself at the experimental and Open Call replicator sites.

We first outlined the agile development process, which improves on issues found and encountered in internal testing as well as experimentation with validation and Open Call partners. This process documents and produces minor release updates in-between the planned and documented major release roadmap that is described in D3.7 [FLAME-D3.7].

We furthermore outlined the current best practises and the utilized tools for replication of the FLAME platform in the various sites. For this, we delayed the submission of this deliverable to include the latest insights from the ongoing Open Call 2 replications in London and Sicily. Since we recognize that the lessons learned from our insights into replicating a programmable 5G service delivery platform are very useful for the wider community, the tools and processes described in this deliverable are continuously updated as part of the platform documentation with the plan of releasing an online version as a repository that goes alongside any platform deployment, serving as a best current practise repository with suitable tools included, such as ARDENT.

Co-funded by the Horizon 2020
Framework Programme of the European Union

5 APPENDIX

5.1 ARDENT INFRASTRUCTURE DESCRIPTOR DEFINITIONS

```
# Author: Sebastian Robitzsch <sebastian.robitzsch@interdigital.com>

compute_nodes:
    required: true
    compute_node:
        availability_zone:
            required: true
            type: string
            description: The availability zone for this OpenStack compute
             node
        name:
            required: true
            type: string
            description: The name of the OpenStack compute node
        vcpus:
            required: true
            type: integer
        ram:
            required: true
            type: integer
            description: Unit in MB
        disk:
            required: true
            type: integer
            description: Unit in GB
        networks:
            required: true
            type: array
            type_schema: string
            description: The list of provider networks this compute node
has access to
        tier:
            type: string
            description: The tier type of this compute node
            valid_values: [ "data_centre", "edge", "far_edge", "mist" ]

networks:
    required: true
    network:
        identifier:
            required: true
            type: string
        category:
            required: true
            type: string
            valid_values: [ "access", "clmc-sfemc", "cluster", "data",
"mgmt", "msp", "ps", "sdnctrl", "sia", "wan" ]

subnets:
    required: true
    subnet:
        identifier:
            required: true
            type: string
```


Co-funded by the Horizon 2020
Framework Programme of the European Union

```
        category:
            required: true
            type: string
            valid_values: [ "sia" ]
            description: The subnet category is directly mapped to the
network one

security_groups:
    required: true
    security_group:
        identifier:
            required: true
            type: string
        category:
            required: true
            type: string
            valid_values: [ "mgmt", "msp", "sdnctrl", "sia", "wan" ]


infrastructre_services:
    required: true
    ip_gateway:
        required: true
        type: string
        description: The IP address of the IP gateway of the
infrastructure
    dns:
        required: true
        type: string
        description: The IP address of the DNS of the infrastructure
    sdn_controller:
        required: true
        type: string
        description: The IP address of the SDN controller of the
infrastructure

metadata:
    required: true
    tenant:
        required: true
        type: string
        description: The OpenStack tenant identifier that has been
assigned to the platform provider
    cidr:
        required: true
        type: string
        description: The CIDR for the entire LAN the platform provider can
use
    mtu:
        required: true
        type: integer
        description: The MTU of the infrastructure for the ports of the
platform on the DATA network
    sia-ip-frontend:
        required: true
        type: string
        description: The IP address the frontend node should be configured
to on the SIA provider network
    ipv4-rules:
        required: false
        type: bool
```


Co-funded by the Horizon 2020
Framework Programme of the European Union

```
        description: Enable or disabled IPv4 header-based SDN arbitrary
        bitmask forwarding rules. If disabled, IPv6-based headers are
        being used.
```


Co-funded by the Horizon 2020
Framework Programme of the European Union

REFERENCES

[FLAME-D3.7] D3.7 FLAME Technology Roadmap v2, available at https://ict-flame.eu/wp-content/uploads/sites/3/2019/03/D3.7-FLAME-Technology-Roadmap-v1.0.pdf

[FLAME-D3.10] D3.10: FLAME Platform Architecture and Infrastructure Specification V2, available at https://ict-flame.eu/wp-content/uploads/sites/3/2019/03/D3.10-FLAME-Platform-Architecture-and-Infrastructure-Specification_v1.0.pdf

[FLAME-D4.2] D4.2: 2nd Report on Components for Adaptive Service Delivery of Interactive Media

[3GPP 23501] 3GPP, "System architecture for the 5G System (5GS)", V16.1.0, available at http://www.3gpp.org/ftp//Specs/archive/23_series/23.501/23501-g10.zip, June 2019

[SFC2019] Trossen et al., "Name-Based Service Function Forwarder (nSFF) component within SFC framework", IETF draft, available at https://tools.ietf.org/html/draft-trossen-sfc-name-based-sff-07, July 2019

Co-funded by the Horizon 2020
Framework Programme of the European Union