Complete Dynamic Multi-cloud Application Management

Project no. 644925

Innovation Action

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

Call identifier: H2020-ICT-2014-1

Topic: ICT-07-2014 – Advanced Cloud Infrastructures and Services

Start date of project: January 1st, 2015 (36 months duration)

Deliverable D6.4

Summary of Provided Brokering, Deployment, and Management Features

Due date: 30/09/2017

Submission date: 27/10/2017

Deliverable leader: TUB

Editors list: Dirk Thatmann (TUB), C. Loomis (SixSq)

Dissemination Level

☒	PU:	Public
☐	PP:	Restricted to other programme participants (including the Commission Services)
☐	RE:	Restricted to a group specified by the consortium (including the Commission Services)
☐	CO:	Confidential, only for members of the consortium (including the Commission Services)

List of Contributors

Participant	Short Name	Contributor
Interoute S.P.A.	IRT	Domenico Gallico
SixSq Sàrl	SIXSQ	C. Loomis, K. Skaburskas, L. Schaub, K. Basbous
QSC AG	QSC	
Technische Universitaet Berlin	TUB	Dirk Thatmann, Mathias Slawik
Fundacio Privada I2CAT, Internet I Innovacio Digital A Catalunya	I2CAT	
Universiteit Van Amsterdam	UVA	Alexéy Ilyushkin
Centre National De La Recherche Scientifique	CNRS	

Change history

Version	Date	Partners	Description/Comments
0.0	13/09/2017	SixSq	Table of contents
0.5	19/10/2017	SixSq	Initial draft for use case input and feedback.
0.6	23/10/2017	TUB	Feedback on initial draft.
0.9	24/10/2017	SixSq	Final draft for internal review.
1.0	27/10/2017	SixSq	Final document.

Table of Contents

List of Tables

List of Figures

Executive Summary

At the beginning of the CYCLONE project, SlipStream was already capable of managing the basic lifecycle of cloud applications. With input and feedback from the application developers of the targeted use cases (WP3), potential improvements were identified and requirements defined.

Over the last three years, the project has significantly enhanced the brokering, deployment and management features for multi-cloud applications by extending SlipStream. The use cases from WP3 have provided guidance on the features to be implemented and have validated those features with Nuvla, SixSq's commercial SlipStream service that is a component of CYCLONE's testbed. The fact that Nuvla was updated fortnightly with CYCLONE updates is further evidence that the developed features are of production quality and useful to users from diverse domains.

This document summarizes the brokering, deployment, and management features that are available from SlipStream, the core service within the CYCLONE project for cloud application management. The implemented features include:

- **Application curation**: Portability, Automated Deployment, Component Coordination, Application Parameterization, Shared Applications, and Stock Components.
- **Authentication and authorization**: Authentication by Username/Password, API Key/Secret, GitHub, eduGAIN, Elixir AAI; Authorization by User or by Role/Group; and Role/Group definition.
- **Deployment and management**: Offer-Based Provisioning, Multi-Cloud Deployments, Horizontal Scaling, Vertical Scaling, Ranking by Cost, and Policy Constraints.
- **Monitoring**: Benchmarking, Current Usage, Historical Usage, and Quota.
- **Scalability, reliability, and usability**: Python API, Libcloud Driver, and Clojure(Script) API in addition to a general move towards micro-services to improve the scalability and reliability.

Overall, there were 56 requirements of which 33 (59%) were fully implemented, 11 (20%) remain on the short-term roadmap, 12 (21%) will not be implemented for various reasons.

As SlipStream is a commercial solution at the core of SixSq's portfolio, the evolution of SlipStream will continue after the end of CYCLONE. As mentioned above, 11 of the requirements remain on the short-term SlipStream roadmap and will likely be implemented within the next six months. These include:

- **Application curation**: Services, Hierarchical Applications, Module Isolation, and GitHub Integration.
- **Authentication and authorization**: Completion of CIMI Migration and Resource Segmentation.
- **Deployment and management**: Extended Pre-Filtering, User-Specified Ranking, Scaling with Offers, Scaling through the UI, and Dynamic Application Topology.
- **Monitoring**: Extended Resource Coverage and User-Defined Quotas.
- **Scalability, reliability, and usability**: Continued Migration to Micro-Services.

One large area that remains to be covered is the inclusion of data management. Experiments have been conducted with the European Space Agency to see how data resources can be defined in the Service Catalog and with HNSciCloud to understand how to integrate scientific data management services, in this case Onedata from the Indigo Data Grid project.

1. Introduction

At the beginning of the CYCLONE project, SlipStream was already capable of managing the basic lifecycle of cloud applications. With input and feedback from the application developers of the targeted use cases (WP3), a number of improvements were identified. Previous documents have described the requirements associated with those improvements and plans for implementation.

The features are organized into the following categories:

- Application curation,
- Authentication and authorization:
- Deployment and management,
- Monitoring, and
- Scalability, reliability, and usability.

A short summary of the features is provided at the end of the document. The appendix contains detailed commentary on individual requirements defined in the previous three deliverables of WP6.

The feedback from the use cases (WP3) and developments from the other CYCLONE work packages have been instrumental in defining the direction of the CYCLONE developments, prioritizing changes, and validating those that have been implemented. Details can be found in the WP3 documents. Although some requested features have not yet been implemented, they are, in most cases, part of the SlipStream roadmap that will drive the evolution of SlipStream through the remainder of the project and afterwards.

2. Application Curation

Running applications on a cloud system requires management of virtual machine images. In many cloud management systems, users generate these machine image files and then upload them to their cloud service provider (CSP). The management overhead associated with the transport, conversion, and evolution of these images discourages the use of multiple CSPs.

SlipStream takes a different approach: users specify the resource requirements, placement constraints, and the software installation and configuration procedures. SlipStream then uses this information to transform existing, minimal images (optimized for each cloud provider) into the customized VM requested by the user. This has two advantages: 1) the image descriptions are portable and can be used for any cloud supported by SlipStream and 2) all knowledge about the application is captured and managed.

On clouds that support customized user images, binary image files can be produced ("built") by SlipStream to reduce the startup latency. Users must explicitly request the build of these binary image files, but once produced, SlipStream will use them automatically. This maintains cloud portability while allowing users to shorten start up latencies for particular CSPs.

SlipStream provides a "workspace" in which users manage their application and component descriptions. These descriptions can be shared with other users. In addition, system administrators can publish vetted applications into an "App Store" to make them visible to all SlipStream users. Applications in the App Store can be found easily and launched with a "single click".

Within the component definitions, the resource requirement and placement constraints directly affect the CSPs that are chosen. The placement constraints can include security, location, availability, and other non-functional requirements to support the definition and enforcement of SLAs.

The following subsections highlight features that are commonly used by CYCLONE project participants and within the ported use cases. They also identify points where the platform could be further improved; this feedback helps define the roadmap for SlipStream during the remainder of the project and afterwards.

Implemented features commonly used for CYCLONE platform components and use cases:

- **Portability**. All the CYCLONE components have been designed for a multi-cloud (hybrid cloud) context. Consequently, all these components make use of the cloud portability features of SlipStream. The use cases similarly take advantage of cloud portability, with most use cases demonstrated on multiple cloud infrastructures.
- **Automated Deployment**. Users can easily define complex applications containing any number of components. SlipStream fully automates the deployment of such applications, allowing users to concentrate on managing the application as a whole, rather than the numerous individual machines. This feature saves time by freeing people from the tedious details of a complex deployment but also by avoiding mistakes that occur with manual deployments.
- **Component Coordination**. In any complex application, there are dependencies between the various components. For example, in a typical 3-tiered web application the business logic cannot start before the underlying database is ready, and of course, the business logic must know the endpoints of the database. SlipStream provides a simple mechanism for passing information

between the components and critically, for waiting on required information to be published. This feature is used for all component and application definitions.

- **Application Parameterization**. The same feature that allows coordination between application components also allows the application to take values for input parameters from the person deploying the application, promoting generalization of applications and reuse. This feature is also used by application developers to feed information (e.g. randomized passwords or service endpoints) back to the application owner. All the CYCLONE components use this parameterization to provide reusable components. Most of the use cases also use this feature to allow for customized deployments, for example, varying the input data for a scientific analysis.
- **Sharing Applications**. Having the ability to reuse component definitions and parameterize applications is of limited use unless they can also be shared with others. On the developer's side, this means sharing the work of building and maintaining an application between team members. On the user's side, this means being able to publish applications for use by other people on the platform. The SlipStream access control allows the developer to define the visibility of an application by others. Vetted applications can also be published in the App Store. All the CYCLONE components are visible to all users of the platform. The applications for use cases are commonly shared between a team of developers and a targeted set of users.
- **Stock Components**. Developers can take advantage of a large number of stock components that are maintained by SixSq and published in the App Store. These range from minimal operating system images to scalable container infrastructures such as Kubernetes or Docker Swarm. CYCLONE has taken the same approach with most of the networking, security, and AAI components available as "bricks" to incorporate more complex applications.

Features identified through the CYCLONE activities that would further enhance the utility of the cloud application management platform:

- **Services**. Although many applications have a single lifecycle and benefit directly from SlipStream's automated deployments, many applications have subsystems with independent lifecycles. For example, operators of a 3-tiered web application may want to operate the database separately from the business logic. Nonetheless, they would still like to benefit from SlipStream's coordination infrastructure to allow information (e.g. endpoints or access credentials) to be passed between the different "service" deployments.
- **Hierarchical Applications**. Allowing an operator to decide at deployment time whether to deploy an application in its entirety or to deploy it as a set of cooperating services would provide further flexibility. To support this, it would be useful to be able to define SlipStream applications hierarchically, allowing application definitions to incorporate other applications directly. (Currently applications can only incorporate single machine components.)
- **Module Isolation**. The SlipStream workspace has a root that is shared by all users of the system. Although this makes sharing a bit easier, it raises the possibility of conflicts with top-level project names and "pollutes" a user's workspace with the visible projects of others. Removing this common root while providing the ability to search for shared components would improve the usability of the platform.
- **GitHub Integration**. SlipStream allows component and application definitions to be managed and versioned directly within the platform. However, many developers would prefer instead to be able to manage their SlipStream recipes next to their source code. Allowing SlipStream to integrate directly with GitHub (or any other popular source code management system) would improve the usability of the platform and appeal to a larger group of developers.

3. Authentication and Authorization

When CYCLONE started, the Authentication and Authorization Infrastructure (AAI) for SlipStream was basic. Only authentication with usernames and passwords defined within the internal database was supported. SlipStream access control was limited to Unix-like permissions, allowing rights to be defined for owner, group (as an explicit list of users), and others.

The SlipStream AAI has improved considerably over the lifetime of the project, expanding the supported authentication sources based on work from the security work package (WP4) and implementing a more flexible resource authorization based on a generic Access Control List (ACL).

3.1. Authentication Methods

To allow new authentication methods to be added more easily, the SlipStream authentication code was significantly refactored. Adding direct support for new authentication methods now only requires adding a couple resources compatible with the Cloud Infrastructure Management Interface (CIMI) from DMTF [CIMI16] and a function to handle the interactions with the (external or internal) service that provide authentication information. Figure 1 shows the internal and external authentication methods that are now supported by SlipStream (Nuvla).

The refactored authentication code contains “plugins” for handling the GitHub (OAuth2) [OAUTH] and OpenID Connect (OIDC) [OIDC] protocols for external authentication. The project’s Keycloak server, allowing access to the eduGAIN identity federation through the German NREN DFN, is accessed through the OIDC protocol by SlipStream. As Keycloak supports a wide variety of identity providers, sources such as LDAP, Twitter, Facebook, etc. can be supported via straightforward configuration changes.

SixSq has deployed and certified a second Keycloak server to support the Helix Nebula Science Cloud (HNSciCloud) project. This second Keycloak server allows access to the eduGAIN [EDUGAIN] identity federation through SWITCH, the Swiss NREN. In addition, access to the identity federation for Elixir [ELIXIR], a European flagship project for bioinformatics, has been configured. This deployment is an excellent validation of the work done on Keycloak within WP4, showing the generality of the service and its utility for large scientific collaborations.

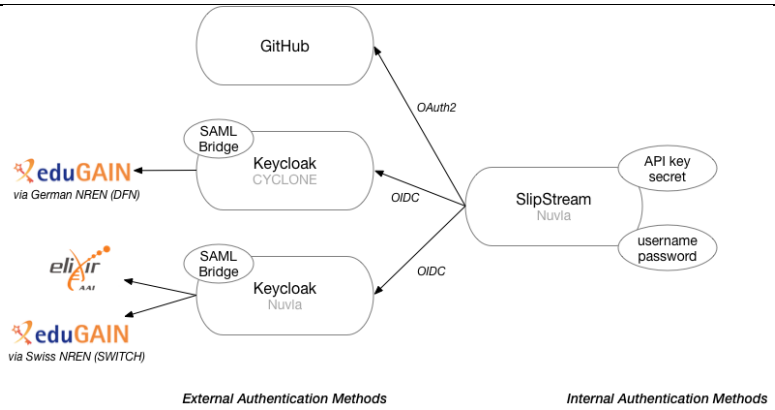

Figure 1: SlipStream (Nuvla) Authentication Services

SlipStream still supports authentication via usernames and passwords stored in SlipStream's internal database. In addition, SlipStream also allows users to generate separate API key/secret pairs. The API key/secret pairs can optionally be created with an expiration date; an API key/secret pair can be revoked at any time. A different API key/secret pair can be generated for each client, allowing more fine-grained control over authorization for clients accessing SlipStream via the command line or API.

In summary, the authentication methods that are supported are:

- **Username/Password**. Simple credentials tied to a user account and stored in SlipStream's internal database.
- **API Key/Secret**. Credentials that can be generated by users to allow clients to access SlipStream via the command line or API. The pairs can be revoked independently of one another; they can also be created with an expiration date.
- **GitHub (OAuth2)**. Direct support for authentication via the OAuth2 protocol, configured on Nuvla for GitHub.
- **eduGAIN (OIDC)**. Support for the eduGAIN identity federation via the CYCLONE and SixSq Keycloak servers. This allows authentication through the identity providers of most academic institutions in Europe.
- **Elixir (SAML2)**. Support for the Elixir AAI federation via the SixSq Keycloak server.
- **Group/Role Definition**. When using external authentication methods through the Keycloak server, it is possible to define groups and roles and assign these to users. This allows collaborations that span multiple institutes to define common attributes independently of each institute's identity provider.

The above features cover completely the defined use cases and there are no authentication enhancements on the current SlipStream roadmap. However, there have been some tentative requests for direct LDAP support to allow COmanage (from the Internet2 initiative) to be used instead of Keycloak.

3.2. Access Control

As described in previous deliverables, the SlipStream code base is migrating from one that provides a custom API to one based on the CIMI standard. The authorization models for the two cases differ significantly.

The authorization model associated with resources inside the "custom API" uses Unix-like semantics. Users may assign a limited set of rights to the "owner", "group" and "others". The group is defined as an explicit

list of users. This model only recognizes one group and does not allow the rights to be assigned to externally defined groups or roles. Although there is some limited inheritance of groups, group definitions cannot be effectively reused. Overall, this rigid model is too limited to provide effective authorization for multi-institute collaborations.

The authorization model for the CIMI-based resources is much more flexible. The Access Control List (ACL) for a resource consists of an owner (which may be a user, group, or role) that always has full control of the resources and an optional list of rules. A rule consists of a principal (user, group, or role) and an associated right. This is a significant improvement over the previous authorization model because rights can be associated with external group and role information. One intentional limitation of the implemented ACL is that it allows only positive definition of rights, that is rules that deny certain rights are not supported. This limitation allows fast evaluation of the ACL and allows pre-filtering within the database, ensuring good performance and scalability of the SlipStream service.

The implementation of the ACLs is completely generic and is automatically added to a resource when it migrates to the CIMI-based framework. Consequently, the features of the new authorization model are visible to users as soon as a resource has been migrated. The resources that remain to be migrated are the user, deployment, and module resources. The user resource is currently being migrated and the migration of the deployment resource is on the short-term roadmap.

The authorization features that are available with the CIMI-based framework are:

- **Authorization by User**. Allows the rights for a resource to be assigned based on the identity of a user. Different rights can be assigned to different users.
- **Authorization by Role/Group**. Allows rights for a resource to be associated based on a group or role, which is assigned independently of the ACL.

The following features related to authorization appear on the SlipStream roadmap:

- **Complete Migration**. Complete the migration of the remaining resources (user, deployment, and module) so that they benefit from the new authorization model.
- **Resource Segmentation**. The ability to segment the defined resources into metadata, data, and ACL and control the authorization separately for each segment.

The resource segmentation will be implemented as a richer set of rights that can be assigned to users. In fact, this richer set of rights has been implemented but is not yet applied separately to the three segments of a resource.

4. Cloud Application Deployment and Management

In the initial SlipStream deployment model, application developers had to specify the virtual machine size ("flavor") for every cloud and users had to specify the target cloud directly. This deployment model works well for small numbers of clouds but quickly becomes tedious as the number of clouds grows. Moreover, this model does not allow for "special offers", like reserved instances or spot instances, to be selected.

Because of these limitations, a deployment model based on "Service Offers" was designed near the beginning of the project. The implementation consists of a Service Catalog and an enhanced Deployment Engine. In addition, the application scaling process has been enhanced.

4.1. Service Catalog

The Service Catalog is a collection of resources to allow people to understand the offers available from cloud providers and to select offers appropriate for a given cloud applications. The Service Catalog resources are:

- **Service Offer**: The primary resource containing information about a specific offer from a cloud provider. For instance, this may be an offer for a particular flavor of a virtual machine with an associated price and/or quality of service.
- **Service Benchmark**: A secondary resource that allows any user to post performance or reliability information about cloud resources. This is primarily intended for pre-filtering and ranking of cloud infrastructures.
- **Service Attribute**: A human-readable description of a particular attribute that can appear in a Service Offer. This primarily documents the semantics of an attribute.
- **Service Attribute Namespace**: An association between an attribute prefix and URI. This is a technical requirement to ensure uniqueness of attributes.

Together these resources provide a "database" that can be queried by humans and machine alike, to select appropriate cloud resources.

4.2. Deployment Engine

The deployment procedure consists of the following steps:

1. **Pre-filtering**. Selection of the cloud infrastructures to be considered for the deployment. Remove from consideration offers from ineligible clouds.
2. **Offer Filtering**. Based on resource requirements and policy constraints (e.g. jurisdiction) for the application, filter those that do not meet the application requirements.
3. **Ranking**. Based on a ranking algorithm, order the eligible offers from most to least attractive.
4. **Selection**. Choose the service offer to use for each component of an application. This can either be done manually by the user or automatically based on the ranking.

5. **Provisioning**. Allocate the selected resources and then run the application's "recipes" to bring the application into a running state.
6. **Scaling**. As the load on the application changes over time, scale the application by changing the resources allocated to the application. When adding resources, the same resource selection procedure should be used as for the initial startup.
7. **Termination**. Stop the application and free all allocated resources.

Note that the provisioning is done per-component, so that a multi-component application can easily use resources from multiple clouds in the same application deployment.

This new deployment model has been progressively implemented over the course of the project. Currently the SlipStream deployment engine is entirely based on service offers, although there are some limitations at the various stages listed above. These limitations include:

- Pre-filtering is limited to the list of configured clouds for a given user. The abilities to pre-filter on cloud performance metrics or the location of required data are planned, but not yet implemented.
- The ranking algorithm is limited to cost. The ability to allow users to define ranking algorithms based on other criteria is planned.
- The scaling of an application uses resources from the same cloud provider(s) that are currently used. The scaling process will eventually re-evaluate the offers for each new resource allocation.

These limitations will be removed as part of the ongoing SlipStream development roadmap.

4.3. Application Scalability

The load of a cloud application will vary over time. To maintain performance, the resources allocated to the application must rise and fall in tandem with the load. SlipStream provides a mechanism to automate the scaling process.

SlipStream provides both horizontal and vertical scaling through the API. The primary limitations are that the topology of an application cannot be changed at run time and that the scaling actions cannot be triggered through the browser interface.

SixSq provides documentation for how to trigger scaling through the SlipStream API. SixSq also provides an example application that demonstrates "auto-scaling" based on application-defined metrics. UvA has also worked to show that the mechanisms are general enough to be adapted to different scaling algorithms.

4.3.1. Riemann Scaling

The example auto-scaling application from SixSq bases its implementation on the Riemann plugin of collectd and a custom publisher written in Python conforming to the Riemann client library API. The full details have been described in Sections 4.7 and 4.8 of a previous deliverable [D6.3] and are not repeated here.

The example auto-scaling application contains the following components:

- **webapp**: a stateless web application that takes requests, synchronously performs a moderately intensive computation (calculating π up to 100 digits), and returns the result,
- **nginx**: a load balancer based on the Nginx [NGIN16] web server that distributes client requests to the set of stateless web servers,
- **client**: a test client based on Locust [LOC16] that simulates a varying number of clients, and
- **autoscaler**: Standard SlipStream autoscaler component that makes scaling decisions.

The application can be found in the AppStore on Nuvla; its source code is in the "client-nginx-webapp" module in the GitHub repository. The application changes the number of "webapp" machines based on the observed response time for requests; the "autoscaler" component interacts with SlipStream to trigger the addition or removal of "webapp" machines.

4.3.2. Pegasus Scaling

To ensure that the scaling mechanisms in SlipStream are independent of technologies chosen in the example autoscaling application, UvA implemented an elastic deployment of the Pegasus workflow management system [PEG17] in SlipStream, called ElasticPegasus. The provided setup allows to execute complex scientific workflows in an elastic environment equipped with an autoscaler, which permits adjusting the number of allocated resources (thus, horizontal scaling) to meet SLAs automatically.

The Pegasus workflow management system is based on HTCondor [CON17] workload management system for compute-intensive jobs. Pegasus facilitates the execution of complex (scientific) workflows in distributed computing environments. There are many popular workflow types which are used in many different fields from astronomy to bioinformatics [EDL17]. Pegasus uses HTCondor to transfer the files using either a shared file system or a so-called condorio mode in which HTCondor is responsible for file transfers between the tasks/computing nodes. In ElasticPegasus, the condorio mode is used because it simplifies the setup, avoiding the installation and configuration of a network file system.

For ElasticPegasus, an elastic SlipStream application with a single head node and a set of worker nodes is used. The whole setup runs an additional single orchestrator VM, automatically instantiated by SlipStream during the deployment of the application. The orchestrator is needed to make the deployment scalable and this is a requirement of the SlipStream platform.

The head node acts as an HTCondor Central Manager, which maintains the HTCondor pool and it is responsible for job submissions. The worker VMs only execute HTCondor jobs. Using the component parameter mapping, SlipStream guarantees that the head VM boots before all the other VMs and that the IP address of the head VM is provided to all the worker VMs. Worker VMs automatically join the HTCondor pool using the provided IP address of the head VM.

Pegasus is only installed on the head VM, where it provides commands to generate workflow Directed Acyclic Graphs (DAGs), plan the workflow execution, and to submit new workflows. It also allows monitoring the execution of workflows, collects results, and runs a web interface to monitor the progress of the submitted workflows.

The head VM also runs a small provisioning controller daemon and a RESTful autoscaling service. The provisioning controller periodically (by default every 30 seconds) invokes the autoscaler. The RESTful interface requires two input arguments: current demand and service rate, and returns as a prediction a number of VMs to provision or release. The prediction is used by the provisioning controller to start and stop VMs. We specify current demand as the number of currently eligible (with satisfied precedence constraints) and running workflow tasks of all the workflows in the system. To calculate the service rate parameter, the provisioning controller, measures how many workflow tasks are processed per autoscaling interval by a single VM. The provisioning controller is also responsible for issuing provisioning and release commands to SlipStream. For that, it keeps track on the VM load information and decides which VMs should be stopped. It only stops idle VMs, starting with VMs that have been idle the longest.

ElasticPegasus has two autoscaling policies: React and Adapt [ILY17]. Both policies are workflow-agnostic, which means that they can also be used to control other application types.

- **React**: Chieu et al. [CHI09] presented a dynamic scaling algorithm for automated provisioning of VM resources based on the number of concurrent users, the number of active connections, the

number of requests per second, and the average response time per request. The algorithm first determines the current web application instances with active sessions above or below a given utilization. If the number of overloaded instances is greater than a predefined threshold, new web application instances are provisioned, started, and then added to the front-end load-balancer. If two instances are underutilized with at least one instance having no active session, the idle instance is removed from the load-balancer and shutdown from the system. In each case, the technique reacts to the workload change.

- **Adapt**: Ali-Eldin et al. [ALI12] propose an autonomous elasticity controller that changes the number of VMs allocated to a service based on both monitored load changes and predictions of future load. The predictions are based on the rate of change of the request arrival rate, i.e., the slope of the workload, and aims at detecting the envelope of the workload. The designed controller adapts to sudden load changes and prevents premature release of resources, reducing oscillations in the resource provisioning. Adapt tries to improve the performance in terms of number of delayed requests, and the average number of queued requests, at the cost of some resource over-provisioning.

Figure 2 shows the primary components of the ElasticPegasus application. The application can be selected and deployed through the SlipStream browser interface or API. Running this application with varying loads show that the autoscaling mechanisms work as expected. These autoscalers showed good performance when used for the autoscaling of workflow applications.

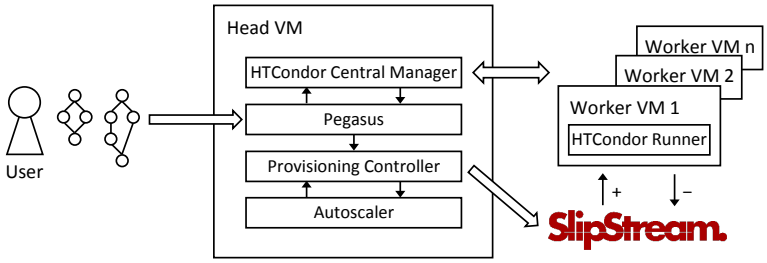

Figure 2: Pegasus Autoscaling with SlipStream

4.4. Multi-Cloud Support

The CYCLONE platform easily integrates multiple cloud service providers to create a uniform interface to a hybrid (and usually heterogeneous) cloud. The broker's "connectors" act as an abstraction layer between the broker (SlipStream) and the various APIs of the cloud service providers. In this way, users see a uniform interface for provisioning and managing cloud resources independently of the underlying cloud API. SlipStream provides connectors for popular open-source and commercial cloud APIs. Table 1 lists the available connectors and the required SlipStream Edition. The Community Edition is available under the Apache 2 license; the Enterprise Edition requires purchase of a commercial license.

SixSq's commercial SlipStream service, called Nuvla, is the broker used to bind cloud infrastructures together to create the CYCLONE testbed. Table 2 shows the cloud infrastructures that are accessible from Nuvla. Cloud infrastructures operated by Interoute, CNRS-LAL, and QSC contribute officially to the CYCLONE testbed. Project participants have also used clouds operated by Amazon, Exoscale, and IFB to run various applications and to validate components of the CYCLONE platform.

Table 1: SlipStream Cloud Connectors

Cloud API	SlipStream Edition	
	Community	Enterprise
Amazon EC2		✔
Azure		✔
Cloudstack	✔	✔
Exoscale		✔
Openstack	✔	✔
NuvlaBox		✔
Open Telekom Cloud (OTC)		✔
OpenNebula	✔	✔
SoftLayer		✔
StratusLab	✔	✔

Table 2: Nuvla Cloud Service Providers

Cloud Service Provider	Cloud API	Countries	Regions	CYCLONE Testbed
Advania	Openstack	SE	1	
Amazon Web Services (AWS)	EC2	EU+	10	≈
Atos ITER	StratusLab	ES	1	
CESNET	OpenNebula		1	
CloudFerro	Openstack		1	
CNRS-LAL	Openstack	FR	1	✔
EBI Embassy	Openstack		1	
Exoscale	Exoscale	CH	2	≈
IBM SoftLayer	SoftLayer	IT	1	
Institut Français Bioinformatique	Openstack	FR	6	≈
Interoute (IRT)	Openstack	IT	1	✔
Microsoft Azure	Azure	NL	1	
NuvlaBox	NuvlaBox	CH, FR, IT	18	
Open Telekom Cloud (OTC)	OTC	DE	1	
QSC	Openstack	DE	1	✔
SCISSOR (H2020)	OpenNebula	FR, IT	3	
Tiede HPC	OpenNebula	ES	1	

4.5. Features

The features that have been implemented are:

- **Offer-Based Provisioning**. The ability to provision cloud resources based on specific service offers, including characteristics like available resources, location, and quality of service.
- **Multi-Cloud Deployments**. Developers can create cloud applications that simultaneously use resources from different clouds.
- **Scaling**. Allowing an application to scale horizontally (more or fewer machines) or vertically (more local resources) to accommodate a change in load.
- **Horizontal Scaling**. Adding or removing virtual machines from a running cloud application.

- **Vertical Scaling**. Adding more resources (CPU, RAM, disk) to a single virtual machine.
- **Ranking by Cost**. The ability to rank appropriate service offers based on the approximate cost of those resources.
- **Policy Constraints**. The possibility for application developers to define policy constraints for an application, for example, to limit where an application can be run.
- **Benchmarking**. Allowing all users of the platform to publish benchmark information about service offers within the Service Catalog, to allow users to make better choices concerning the available cloud resources.

The full provisioning chain is available to users, but there are some limitations:

- **Extended Pre-filtering**. Allow the cloud pre-filtering to include benchmarking criteria and other constraints, like the location of required data sets.
- **User-Specified Ranking**. Allow the users to specify their own ranking algorithms to identify the most attractive offers from the Service Catalog.
- **Scaling with Offers**. Update the scaling process inside of the deployment engine to use offers from the Service Catalog.
- **Scaling through UI**. Extend the browser-based UI to allow scaling actions to be triggered.
- **Dynamic Topology**. Application developers must describe how many clouds will be used in an application in its definition. The distribution of components over clouds cannot currently be changed dynamically.

All these limitations appear in the SlipStream development roadmap and will be implemented in accordance with user requirements.

5. Monitoring

The SlipStream monitoring system is intended to provide information about both current and historical resource utilization. The existing implementation (see Figure 1) at the start of the project was extremely limited. It could not be extended to resources other than virtual machines and even for virtual machines it was not possible to associate the usage with groups and roles. At the technical level, the implementation would also not scale to a large number of users.

For these reasons, the system was redesigned and re-implemented. The system now uses a separate service to maintain the current global state of resource utilization across all clouds. To allow for historical views, a snapshot of the current global state is taken every minute. With this new design, users can obtain detailed reports of usage over any time period and filter the usage by group, role, application, or any other attribute of the metered resource. Because the cost is added during the snapshot, approximate billing can also be provided.

The quota enforcement mechanism has also been re-implemented to rely on the current global state. Because of the generality of the solution, quotas can be placed on any resource which appears in the global state. These quotas can also be applied by user, group, role, or other attributes.

The new implementation has just recently been incorporated into the production version of SlipStream, so there is not widespread use of these new features:

- **Current Usage**. The ability to see a "near" real time view of a user's (or group's, etc.) resource utilization.
- **Historical Usage**. The ability to see a user's (or group's, etc.) resource utilization over a given time period with cost information.
- **Quota**. The ability to limit resource utilization to a predefined value and to prevent further resource allocations that would violate this limit.

All of these features are available through the REST API, but have not been integrated into SlipStream's browser interface. A command like:

```
curl -X PUT -H 'content-type:application/x-www-form-urlencoded' \
     https://nuv.la/api/metering \
     --data-urlencode '$filter=deployment/user/href="user/'${user_id}'"' \
     -d '$filter=snapshot-time>="2017-10-01T00:00:00.000Z"' \
     -d '$filter=snapshot-time<="2017-10-08T00:00:00.000Z"' \
     -d '$last=0' \
     -d '$aggregation=count:id' \
     -d '$aggregation=sum:price' \
     -d '$aggregation=sum:serviceOffer/resource:vcpu' \
     -d '$aggregation=sum:serviceOffer/resource:ram'
```

Can be used to recover the resources used over a period of time. This command would provide the total number of machines running, the cost, total CPU · minute, and RAM · minute used over the period for a given user.

The foreseen extensions of the monitoring features include:

- **Extended Resource Coverage**. Currently only virtual machines are monitored. The monitoring needs to be extended to storage, networking, and other resources.

- **User-Defined Quotas**. Only the SlipStream administrator can define quotas. Individual users should be able to define quotas for themselves and group managers should be able to do the same for the group and for members of the group.

As for the other defined limitations, work to remove the limitations appears in the SlipStream roadmap.

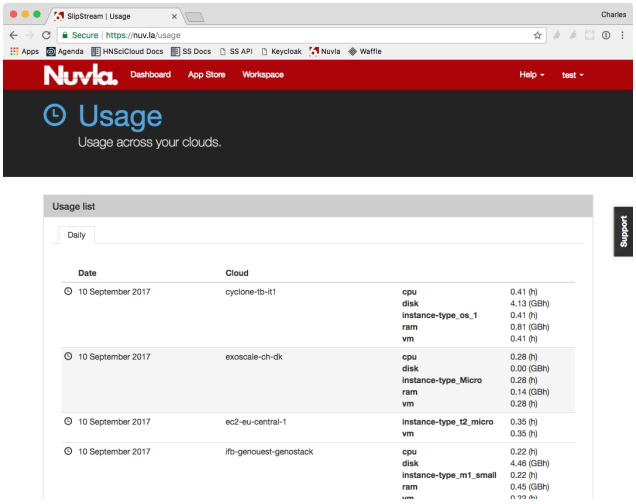

Figure 3: Old Interface for Cloud Usage Information

6. Scalability, Reliability, and Usability

The scalability, reliability, and usability are critical characteristics for any production service. There have been a number of improvements to SlipStream in these areas over the course of the project.

To improve the scalability and reliability of the SlipStream platform, important changes have been made to the operations model:

- **Micro-services**. The SlipStream service deployment has shifted towards micro-services to improve its performance and resilience. A SlipStream deployment now consists of the custom API server, the CIMI API server, the "Pricing and Ranking Service", the metering service, and the "collector" (for maintaining global state). This allows a better view of resource consumption within the system and more control over the individual processes.
- **Redundancy**. Newer micro-services within the SlipStream ecosystem have been designed to be "stateless". This allows for easy replication and scaling of the individual services, which in turn improves the overall reliability of the platform.
- **Distributed Database**. On the backend, a distributed Elasticsearch deployment is used to provide a reliable database for the majority of the SlipStream platform. A single HSQLDB database is still used for the deployment and module resources, but it will disappear as those resources are migrated towards CIMI.
- **Reduced Polling**. Foundational changes have been made that will allow the use of Server Sent Events (SSE) when interacting with the SlipStream services. This provides a straightforward mechanism to avoid polling by clients, reducing the load on both the client and server.

Changes have also been made to improve the usability of the platform from the browser and programming interfaces:

- **Refactored Python API**. The command line interface (CLI) for SlipStream initially had an internal API for SlipStream that was not exposed to end users. This internal API was separated from the CLI and improved so that Python programmers now have a clean, native interface to the SlipStream platform [SSPY].
- **Libcloud Driver**. Many Python programmers prefer to use the Libcloud API [LIBCLOUD] to control cloud-based workloads. In addition to the native SlipStream Python API, a Libcloud driver is also available [SSLC].
- **Clojure(Script) API**. Newer services on the SlipStream platform are written in Clojure, a LISP dialect running over the JVM. A complete Clojure API is available. This API can be used (with minimal wrapping) from Java. Moreover, this API is compatible with ClojureScript, allowing its use from the JavaScript ecosystem. [SSCLJ]

In addition to the above changes, significant work has been done to replace the browser interface. The newer design relies heavily on mature JavaScript technologies to provide an improved and more responsive experience through the browser. As this interface becomes more mature, it will replace the older interface that relies on a mixture of client and server side rendering.

7. Summary

This work package has over the last three years, significantly enhanced the brokering, deployment and management features for multi-cloud applications by extending SlipStream. The use cases from WP3 have provided guidance on the features to be implemented and validated those features with Nuvla, SixSq's production SlipStream server that is part of CYCLONE's testbed. The fact that Nuvla was updated fortnightly with CYCLONE updates is further evidence that the developed features are of production quality and useful to a wide diversity of users.

The project has made significant improvements in the following areas: application curation, AAI, deployment & management, monitoring, and scalability, reliability, & usability. Table 3 lists the features that were implemented in each category. In addition, Appendix A contains the full list of identified requirements, their states, and detailed commentary. Overall, there were 56 requirements of which 33 (59%) were fully implemented, 11 (20%) remain on the short-term roadmap, 12 (21%) will not be implemented for various reasons.

As SlipStream is a commercial solution at the core of SixSq's portfolio, the evolution of SlipStream will continue after the end of CYCLONE. As mentioned above, 11 of the requirements remain on the short-term SlipStream roadmap and will likely be implemented in the next six months. Table 3 also lists the primary CYCLONE features (derived from the requirements) that remain on the roadmap.

One large area that remains to be covered is the inclusion of data management. Experiments have been conducted with the European Space Agency to see how data resources can be defined in the Service Catalog and with HNSciCloud to understand how to integrate scientific data management services, in this case Onedata from the Indigo Data Grid project.

Table 3: Summary of Available and Planned Features

	Implemented Features	Roadmap
Application Curation	Portability Automated Deployment Component Coordination App. Parameterization Sharing Applications Stock Components	Services Hierarchical Applications Module Isolation GitHub Integration
AAI	Username/Password API Key/Secret GitHub eduGAIN Elixir AAI Group/Role Definition Authorization by User Authz. by Role/Group	Complete CIMI Migration Resource Segmentation
Deployment & Management	Offer-Based Provisioning Multi-Cloud Deployments Horizontal Scaling Vertical Scaling Ranking by Cost Policy Constraints	Extended Pre-filtering User-Specified Ranking Scaling with Offers Scaling through UI Dynamic Topology
Monitoring	Benchmarking Current Usage Historical Usage Quota	Extended Resource Coverage User-Defined Quotas
Scalability, Reliability & Usability	Python API Libcloud Driver Clojure(Script) API	Continued Migration to Micro-Services

References

[D6.1]	Complex Application Description Specification http://www.cyclone-project.eu/assets/images/deliverables/Complex%20Application%20Description%20Specification.pdf
[D6.2]	Specification of Interfaces for Brokering, Deployment, and Management http://www.cyclone-project.eu/assets/images/deliverables/Specification%20of%20Interfaces%20for%20Brokering,%20Deployment,%20and%20Management.pdf
[D6.3]	Solutions for Non-functional Aspects of Cloud Computing http://www.cyclone-project.eu/assets/images/deliverables/Solutions%20for%20Non-functional%20Aspects%20of%20Cloud%20Computing.pdf
[ALI12]	Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. 2012. An Adaptive Hybrid Elasticity Controller for Cloud Infrastructures. In IEEE NOMS.
[CHI09]	T.C. Chieu and others. 2009. Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. In IEEE ICEBE.
[CIMI16]	Distributed Management Task Force, Inc., Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol, 2016. https://www.dmtf.org/sites/default/files/standards/documents/DSP0263_2.0.0.pdf
[COMANAGE]	COmanage https://www.internet2.edu/products-services/trust-identity/comanage/
[DFN]	DFN-Verein (German NREN) https://www.dfn.de/en/
[EDUGAIN]	eduGAIN https://www.geant.org/Services/Trust_identity_and_security/Pages/eduGAIN.aspx
[ELIXIR]	Elixir AAI. https://www.elixir-europe.org/services/compute/aai
[EDUGAIN]	eduGAIN https://www.geant.org/Services/Trust_identity_and_security/Pages/eduGAIN.aspx
[ELIXIR]	Elixir AAI. https://www.elixir-europe.org/services/compute/aai
[ILY]	A. Ilyushkin, et al. An Experimental Performance Evaluation of Autoscaling Policies for Complex Workflows. Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, 2017.
[LIBCLOUD]	Apache Libcloud https://libcloud.apache.org/
[LOC16]	Locust, Locust, 2016. http://locust.io/
[NGI16]	Nginx, Nginx, 2016. https://www.nginx.com/
[OAUTH]	OAuth 2.0 https://oauth.net/2/
[OIDC]	OpenID Connect. https://openid.net/connect/

[PEG]	Pegasus Workflow Management System, 2017. https://pegasus.isi.edu/
[RIES16]	Riemann, Riemann, 2016. http://riemann.io
[SSLC]	SlipStream Libcloud driver documentation. https://slipstream.github.io/slipstream-libcloud-driver/
[SSCLJ]	SlipStream Clojure(Script) documentation. https://slipstream.github.com/SlipStreamClojureAPI
[SSPY]	SlipStream Python API documentation. https://slipstream.github.io/SlipStreamPythonAPI/
[SSDOC]	SlipStream documentation. http://ssdocs.sixsq.com
[SSAPI]	SlipStream API documentation. http://ssapi.sixsq.com
[SWITCH]	SWITCH (Swiss NREN) https://www.switch.ch/

Glossary

AAI	Authentication and Authorization Infrastructure
ACL	Access Control List
API	Application Programming Interface
CIMI	Cloud Infrastructure Management Interface
CLI	Command Line Interface
CSP	Cloud Service Provider
DAG	Directed Acyclic Graph
IaaS	Infrastructure-as-a-Service
JVM	Java Virtual Machine
NREN	National Research and Education Networking organization
OIDC	OpenID Connect
PaaS	Platform-as-a-Service
SaaS	Software-as-a-Service
SAML	Security Assertion Markup Language
SP	Service Provider
SSE	Server Sent Events
WP	Work Package

Appendix A Requirements

This appendix collects the requirements defined in the previous WP6 deliverables and provides a summary of the current implementation status. Those requirements marked with a checkmark (✔) are fully implemented; those with approximately equal sign (≈) indicate requirements that are planned to be implemented in the short term or have been implemented in an alternate way; and those with a cross (✘) indicate those that have not been implemented. Detailed comments on the status are provided for all of the requirements. A summary of the overall implementation status is provided in main text of this document.

Table 4: Implementation Status of Requirements

	No. of Requirements	Percentage of Requirements
Fully implemented (✔)	33	59%
Planned or alternate implementation (≈)	12	21%
Not implemented (✘)	11	20%
TOTAL	56	100%

Table 5: Collected Requirements

Doc.	ID		Title	Level	Description	Comment
D6.1	1	✔	Machine readability	MUST	The application descriptions will be processed primarily by the CYCLONE components. The chosen format MUST be easily readable by machine from a wide variety of different programming languages.	The application descriptions are not yet managed via the CIMI interface and consequently are still exposed programmatically in XML. The XML is machine readable with a consistent schema. Because these resources are used nearly exclusively by the micro-services and clients within the SlipStream ecosystem, the portability of this application is not a major concern.
D6.1	2	✘	Human readability	SHOULD	Application developers may need to read or to create the descriptions directly for debugging, testing, or development of tooling. The chosen format SHOULD be easily readable by humans.	As the application description is still in XML, it is not friendly for human processing. When these descriptions migrate to CIMI, the JSON representation will be much more accessible to humans. Migrating these resources is in the SlipStream roadmap but they will be the last to be migrated.
D6.1	3	✘	Easily understandable	SHOULD	The application descriptions SHOULD be easily understandable by humans, meaning that the schema should have limited complexity and be well documented (e.g. with tutorials, examples, APIs, specifications, etc.).	As the application description is still in XML, it is not easy for humans to understand. Once these resources are migrated to CIMI (and JSON format), they will be easier for humans to work with directly. Migrating these resources is in the SlipStream roadmap but they will be the last to be migrated.
D6.1	4	≈	Extensible	MUST	The application deployment format MUST be extensible to allow for characteristics or features specific to CYCLONE components.	The format is extensible but requires supporting modifications within the SlipStream servers and clients for each change. Some CYCLONE enhancements have been added to the descriptions over the course of the project. The extensibility will be greatly improved once the full migration to CIMI is complete.
D6.1	5	✔	Maturity	SHOULD	The application deployment format SHOULD have production adoption within the wider IT community, demonstrating that it is appropriate for general adoption.	The mature SlipStream application description format has been in production use for more than 3 years. The CIMI translation of the application description will contain the same information but be easier to process and for human readability (being based on JSON rather than XML).
D6.1	6	✔	Application summary	MUST	The description format MUST allow the application developer to provide a detailed, human-readable summary of the application's features, characteristics, and limitations.	The application description format allows the developer to provide detailed metadata concerning the component or application.

Doc.	ID		Title	Level	Description	Comment
D6.1	7	✔	Single machine apps.	MUST	Many applications and services can be contained within a single virtual machine. The description format MUST be able to describe simple, single-VM applications.	For SlipStream, a single VM application or service is called a "component". These components can be fully described by developers and easily deployed by operators or end users.
D6.1	8	✔	Multiple machine apps.	MUST	Many applications require a number of different services deployed on different virtual machines. The description format MUST be able to describe applications composed of multiple machines.	For SlipStream, a multi-machine cloud application is called an "application" and is composed of SlipStream "components". SlipStream handles the coordination between the constituent components and allows the full application to be managed as a whole.
D6.1	9	✘	Hierarchical composition	SHOULD	Complex applications are often built from reusable components that are themselves complex, multi-machine services. The description format SHOULD allow a hierarchical composition of components to foster reuse of descriptions, minimizing developer effort.	This is not currently supported but is an important component of the SlipStream roadmap. The implementation of this is pending because of a dependency on the move of the "deployment" resource to the SlipStream CIMI server. This "deployment" migration is expected to happen at the end of 2017 and will unblock the implementation of this feature.
D6.1	10	✔	Virtual machines	MUST	The description format MUST be able to describe the all of the parameters for the provisioning of virtual machines on a cloud infrastructure.	The description allows the developer to provide the resource requirements of a virtual machine, including CPU, RAM, and disk. With the shift to provisioning based on service offers, all characteristics of a resource (location, price, etc.) can be used for resource selection.
D6.1	11	≈	Containers	SHOULD	Use of (Linux) containers is becoming more widespread as an alternative to full virtual machines. The description format SHOULD be able to describe all of the parameters for the provisioning of a container.	A number of SlipStream components and applications have been provided in the App Store that allow use of containers, including Docker, Docker Compose, Docker Swarm, Kubernetes, and Mesos. Further integration of containers into SlipStream's cloud application management is planned but not yet available.
D6.1	12	✔	CPU specification	MUST	The description format MUST allow the application developer (or user) to define the characteristics of the CPU(s) required.	The minimum number of vCPUs for a component can be defined.
D6.1	13	✔	RAM specification	MUST	The description format MUST allow the application developer (or user) to define the amount of RAM required.	The minimum RAM required for a component can be defined.
D6.1	14	✔	Disk specification	MUST	The description format MUST allow the application developer (or user) to define the amount of disk space available on the application's virtual machines.	The minimum disk space required for a component can be defined, on the cloud platforms that support this. In addition, users can specify "extra disks" when the underlying cloud allows additional disks to be allocated.

Doc.	ID		Title	Level	Description	Comment
D6.1	15	≈	Persistent storage	MAY	The description format MAY allow the application developer (or user) to specify the characteristics of persistent storage (type, amount, location, etc.).	Management of storage resources is not integrated into SlipStream. However, Onedata services (from the Indigo Data Cloud project) are available from the App Store. This allows users to create their own data management platform and control it through SlipStream.
D6.1	16	✔	Multi-cloud	SHOULD	The description format SHOULD allow the application developer to indicate which parts of an application can be deployed on different cloud infrastructures.	The SlipStream application description format and provisioning system fully support multi-cloud applications.
D6.1	17	✔	Placement policies	MAY	The description format MAY allow the application developer (or user) to specify policies for the placement of application components.	Developers can specify placement policies on individual components that identify appropriate service offers. The policies are defined in the rich CIMI filtering language and can use any characteristics defined in the service offers.
D6.1	18	≈	Network connectivity	MUST	The description format MUST allow the application developer to define the network connectivity between application components and between the user and the application. This concerns the accessibility of ports on the application's VMs.	The CYCLONE networking features are delivered through the CNSMO component that is available from the SlipStream App Store. The connectivity (firewall) is defined through the CNSMO component parameters.
D6.1	19	≈	Network isolation	SHOULD	The description format SHOULD allow the application developer to specify the characteristics of isolated network(s) to be created for a given application. This network isolation MAY be extended to machines outside of the application, such as a user's workstation.	The CYCLONE networking features are delivered through the CNSMO component that is available from the SlipStream App Store. The network isolation (VPN) is defined through the CNSMO component parameters.
D6.1	20	✔	Parameterization	MAY	The description format MAY allow an application or application component to be parameterized, allowing information to be passed into or out of an application at deployment or runtime.	A SlipStream component can be parameterized with input and output parameters. Developers can wire parameters from different components together when defining a SlipStream application.
D6.1	21	✘	Metrics	SHOULD	The description format SHOULD allow the application developer (or user) to specify what standard metrics (e.g. CPU load or RAM utilization) should be collected along with their frequency.	Because the metrics and the performance trade-offs vary greatly between applications, metrics collection has not been integrated directly into the SlipStream deployment and application management processes. Instead examples have been provided to show how application developers can implement metric collection, analysis, and actions using SlipStream.

Doc.	ID		Title	Level	Description	Comment
D6.1	22	✘	Dynamic metrics	MAY	The description format MAY allow the application developer (or user) to modify the collected metrics while the application is running.	See comment for D6.1/21.
D6.1	23	✘	External services	SHOULD	The description format SHOULD allow the application developer (or user) specify outside services to be used in conjunction with a deployed application (for example, external authentication databases for single sign-on).	See comment for D6.1/21.
D6.1	24	✘	Actions	MAY	The description format MAY allow actions to be taken in response to the values of collected metrics or provided key performance indicators (KPIs) to be defined.	See comment for D6.1/21.
D6.1	25	✘	SLA	MAY	The description format MAY allow service-level agreements to be defined as constraints on service placement and/or triggers to defined actions.	See comment for D6.1/21.
D6.1	26	✔	Credentials	MUST	The description format MUST be able to provide credentials for configuring access to deployed application services (e.g. SSH public keys for remote shell access).	The description format can specify what SSH keys should be used for an application deployment. SlipStream now provides a general credential resource that supports SSH Keys and API keys/secrets and can be extended to other credentials in the future. The application description format and the user resource are being extended to include references to these credentials.
D6.1	27	✔	Tooling	SHOULD	Tooling (e.g. IDE) that facilitates the creation and use of the application descriptions WOULD be advantageous.	SlipStream itself provides good tooling for creating the application descriptions.
D6.1	28	✔	Lifecycle actions	MUST	The description format MUST allow the cloud application developer to define actions associated with state transitions in the application lifecycle.	Application developers can define actions for any of the cloud application lifecycle state transitions.
D6.1	29	✔	Execution environment	MUST	The application model and description MUST not impose constraints on the execution environment of the application, for example, by excluding the use of certain operating systems (e.g. Windows).	The tools used to control resources from SlipStream are written to be as lightweight and portable as possible. All Linux and Windows platforms are supported. Other platforms, if required, may require small adaptations of the SlipStream integration tools.

Doc.	ID		Title	Level	Description	Comment
D6.2	1	✔	Machine Readable Service Description Format	MUST	CYCLONE consists of a set of interoperating tools that will produce and process the service descriptions. Consequently, the format MUST be a standard format (XML, JSON, etc.) that can be easily processed in a wide variety of programming languages.	The service descriptions ("service offers") are provided in JSON format. The documents can be consumed directly or via one of the supported programming APIs—Clojure (Java), ClojureScript (JavaScript), and Python. The syntax of the descriptions is essentially a key/value format, where the keys are namespaced to avoid collisions.
D6.2	2	✔	Human Readable Service Description Format	SHOULD	Humans will need to read the service descriptions to understand provider offerings and to prepare appropriate queries (policies). The service description format SHOULD be easily readable by humans.	The service offers are extensible key/value pairs provided in JSON format. In addition, supporting resources that provide information about key namespaces and semantic information about each key are provided. These also use a JSON format.
D6.2	3	≈	Endorsement Format	MUST	The format used for service endorsements (e.g. certifications, availability metrics, etc.) MUST be the same format as the service description format.	Separate endorsement resources have not been provided as these have not been required by the users to date. These will likely be added as part of another H2020 project (EU-SEC). However, a Benchmark resource that permits users to provide performance information about service offers has been added.
D6.2	4	✔	Machine Readable Service Query Format	MAY	The service query format (policy) may be created or altered by the CYCLONE tools. The policy format MAY be easily readable from a wide variety of programming languages, either it can be a well-supported standard or written according to a well-defined grammar.	The query language used for the service catalog (and indeed all CIMI-based resources) is the CIMI filtering language. The grammar is defined in the CIMI API specification.
D6.2	5	✔	Human Readable Service Query Format	MUST	Humans as well as tools will be creating the service queries (policies), so the format and grammar MUST be easily readable by humans.	The filtering language defined in the CIMI specification was designed to be easily used by humans. The syntax is quite natural for anyone who has programmed or used a database.
D6.2	6	✔	Flexible Service Description Schema	MUST	The characteristics of the underlying cloud services may vary from service to service (e.g. one based on virtual machines and another based on containers) and over time as new services appear. Consequently, the schema of the service description MUST be flexible.	The schema of the service offers (and benchmarks) is an open collection of key/value pairs. Any appropriate attributes can be added to the service offers. The only requirement is that the attribute namespace(s) must be registered with SlipStream to avoid collisions.

Doc.	ID		Title	Level	Description	Comment
D6.2	7	✔	Graceful Handling of Missing or Incorrect Information	MUST	For many reasons (e.g. a field isn't appropriate for a service, a tool/human creates an erroneous entry), the service descriptions may not contain information expected by a particular query or may have provided the information in an incompatible format (e.g. string instead of an integer). The matchmaking process MUST respond to malformed queries/descriptions in a well-defined manner and MUST accept them without crashing.	The implementation will ignore missing information or incorrect information. In addition, the implementation extends the CIMI filtering language to allow users to search for service offers that do not define a particular key (through support of a "null" value).
D6.2	8	≈	Join Service Descriptions and Endorsements	MUST	For matching on non-functional characteristics, the query language MUST be able to join information from service descriptions and from third-party endorsements.	The ability to perform a "join" between all documents within the service catalog was impossible to implement while maintaining good general query performance. Consequently, the join will be implemented as a staged selection process instead. A pre-filtering stage will eliminate offers that do not conform to performance, security, or other user-level requirements. This will then be followed by the filtering of offers based on application requirements.
D6.2	9	✔	Ordering of Results	SHOULD	A particular service query may return more than one that matches the given requirements. The query language SHOULD allow multiple matches to be ranked or ordered. This allows user preferences to be taken into account.	The results of any query can be ordered (ascending or descending) by the value(s) of any key(s) in the service offer.
D6.2	10	✔	Logical Operations	MUST	The query language MUST support logical AND and OR operations. Additional logical operations (e.g. XOR) MAY also be supported.	The query language supports both AND and OR logical operations.
D6.2	11	≈	Arithmetic Operations	MUST	The query language MUST support basic arithmetic operations (add, subtract, multiply, divide) for integer and floating values. Other arithmetic operations MAY also be supported.	The query language supports relational operations on integer, string, and date values. Extensions to the CIMI filtering language also allow aggregations (sums, averages, statistics, etc.) to be performed over a set of service offers. General arithmetic operations haven't been requested by the current users.
D6.2	12	✔	String Matching	MUST	The query language MUST support string equality and inequality operations.	Both equality and inequality of strings are supported. In addition, a "starts-with" (^=) operator has been provided.

Doc.	ID		Title	Level	Description	Comment
D6.2	13	✘	String Regular Expressions	SHOULD	The query language SHOULD support string matching based on regular expressions.	String matching based on “globbing” would be fairly straightforward to implement; full regular expression support would be difficult. In any case, this functionality has not been required by current users.
D6.2	14	≈	Array Operations	SHOULD	The query language SHOULD support array operations, for example, be able to define filters on the inclusion and exclusion of values within an array.	This feature has not been requested by current users and is not provided. However, all CIMI resources allow users to define a set of properties (key/value pairs) that can be used to tag resources. Searching for documents with certain properties is fully supported by the CIMI filtering syntax and the current SlipStream implementation.
D6.2	15	✘	Service Level Agreement	MUST	The brokering implementations must facilitate the exchange of Service Level Agreements in standard or customary formats.	The implementation does not directly support the import or export of SlipStream placement policies or offers as Service Level Agreements.
D6.3	1	✔	Scaling on Load	MUST	Triggering of scaling actions of an application based on application metrics using simple, predefined algorithms (e.g. adding node based on machine load).	The SlipStream API allows both horizontal (e.g. another database node) and vertical (e.g. more RAM) scaling actions. These actions can be triggered by users or by applications. An example auto-scaling application is provided in the App Store that demonstrates how this can be accomplished.
D6.3	2	✔	Scaling on Application Metric	MUST	Triggering of scaling actions of an application based on application metrics defined by the developer of the application.	See comment for D6.3/1.
D6.3	3	✔	Publishing Benchmarks	MUST	Ability to publish application-specific benchmarks of cloud providers into the Service Catalog or Open Service Compendium.	A Benchmark resource that permits users to provide performance information about service offers has been added.
D6.3	4	✔	Static Placement	MUST	Placement based on static characteristics (e.g. geographical location) of a cloud service provider.	Application developers can specify placement policies based on the characteristics in a service offer.
D6.3	5	≈	Dynamic Placement	SHOULD	Placement based on dynamic VM monitoring information from SlipStream itself.	See comment for D6.2/8. This dynamic information (e.g. availability) will eventually be provided as benchmarking (or similar) resources.
D6.3	6	✔	Placement with External Information	MUST	Placement based on external information pushed into the SlipStream Service Catalog or Open Service Compendium.	The information for service offers can be gathered from any source, including external ones. Once available in a service offer, the standard query/filtering mechanisms are available to users.
D6.3	7	≈	Joined Placement	SHOULD	Placement based on the join of all information associated with a given cloud service provider.	See comment for D6.2/8.

Doc.	ID		Title	Level	Description	Comment
D6.3	8	✔	Price Ranking	MUST	Ranking of selected cloud service providers based on predefined algorithms (e.g. price).	SlipStream already ranks selected service offers based on the calculated price (for service offers with price information).
D6.3	9	✘	User-Defined Ranking	SHOULD	Ranking based on algorithms provided by the application developer and/or the application operator.	This feature is not scheduled in the SlipStream roadmap.
D6.3	10	✘	Notifications	SHOULD	Ability to trigger notifications/alerts through SlipStream.	This feature is planned but considered low priority. This will probably be made available near the end of 2017.
D6.3	11	✔	Auto-scaling	MUST	Ability to trigger scaling actions from within the application.	An example auto-scaling application has been provided in the Nuvla App Store. This shows how to collect and analyze application-defined metrics and then to perform scaling actions based on those metrics.
D6.3	12	✔	Service Offer Search	MUST	Ability to search the Service Catalog and Open Service Compendium manually to see the results from various policies and to ideally then associate those policies with applications.	Searching the catalog of service offers is fully supported through the SlipStream CIMI API. In addition, a prototype web-based UI has been developed. The prototype is available on Nuvla, although not yet a fully-supported, official feature.