MSG-168 Lecture Series on
Modelling and Simulation as a Service (MSaaS)

13. MSaaS Execution in the Cloud

Tom van den Berg
TNO Applied Physics Laboratory
The Netherlands

tom.vandenberg@tno.nl

ABSTRACT

This paper covers the topic of "MSaaS Execution in the Cloud" within the Allied Framework for MSaaS. The paper starts with a description of the Simulation Operator use case (see Technical Reference Architecture) and some of the abilities that an MSaaS Capability should have w.r.t. the deployment and execution of services. The use case is followed by a discussion on the need for orchestration and an overview of available technology that supports the implementation of an MSaaS Capability. Several options for technology platforms are available. Kubernetes is discussed in more detail as the open source technology platform of choice for working with containerized workloads. The paper continues with an example service for Friendly Force Tracking, and concludes with a summary.

1.0 MSAAS DEPLOYMENT AND ORCHESTRATION

1.1 Introduction

NATO and the nations use distributed simulation environments for various purposes, such as training, mission rehearsal and decision support in acquisition processes. Consequently, Modeling and Simulation (M&S) has become a critical technology for the coalition and its nations. However, achieving interoperability between participating simulation systems and ensuring credibility of results still requires large expenditures with regards to time, personnel and budget.

Recent technical development in cloud computing technology and service-oriented architecture (SOA) offers opportunities to utilize M&S capabilities better in order to satisfy NATO critical needs. A new concept that includes service orientation and the provision of M&S applications via the as-a-service model of cloud computing may enable composable simulation environments that can be deployed rapidly and on-demand. This new concept is known as M&S as a Service (MSaaS).

NATO MSG-164 investigates MSaaS with the aim of providing the technical and organizational foundations for a future permanent service-based M&S ecosystem within NATO and partner nations, called the Allied Framework for MSaaS.

This paper discusses the topic of service deployment and orchestration, enabled via the MSaaS Portal Applications as defined in the MSaaS Technical Reference Architecture.

1.2 Use case: start simulation and access results

The Simulation Operator is the main actor involved in the deployment and orchestration of services. In the

"MSaaS Composition and Technical Approach" [1] the Integrator has prepared composition and deployment descriptions for the Simulation Operator to use. The use case for starting a simulation and accessing results is shown in Figure 1. The actor called "User" is included for completeness, as a consumer of the simulation started by the Simulation Operator. The User may be an actual person, or a physical system that consumes a provided service.

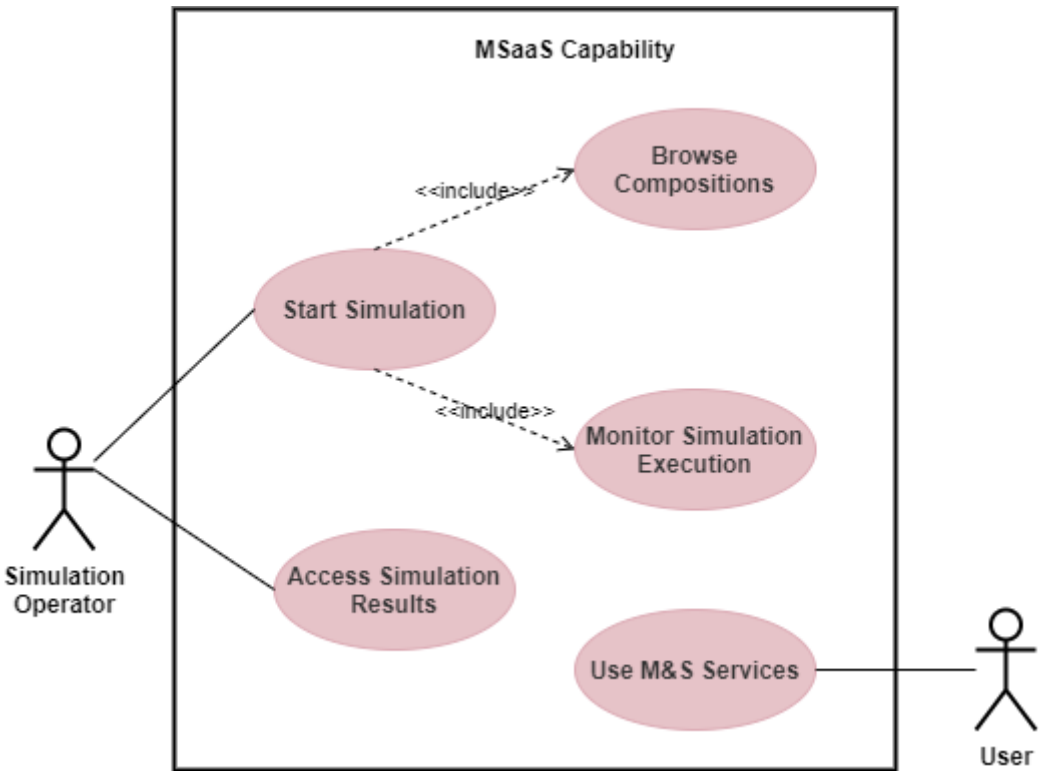

Figure 1: Simulation Operator use case.

The Simulation Operator is responsible for a correctly functioning simulation deployment with the MSaaS Capability. This includes (but is not limited to) the verification that the various (simulation) services function properly, that is, with compliance to agreed service interfaces and contracts; the planning for and allocation of computing resources for a simulation deployment; the start and termination of a simulation execution; the monitoring and control of a deployment, the configuration of simulation scenarios.

This Simulation Operator should be able to view or search available compositions, deployments, and scenarios that can be executed. The Simulation Operator should also be able to make decisions about how particular service will be deployed, e.g., on what computing resource and at which service port. The Simulation Operator is knowledgeable in both the simulated environment as well as for the reasons that the simulation environment is used (analysis, experimentation, testing, training, etc.).

1.2.1 Start Simulation

The "start simulation" use case includes several other use cases, as shown in Figure 1. The use case interactions of the Simulation Operator include the ability to review existing composition descriptions and associated deployment descriptions, define a simulation event, select the composition and deployment description for the event, select from available simulation scenarios for the event, and provide configuration parameters. Depending on the objectives and needs of the simulation event, a composition or deployment description may need to be adapted or tailored by the Integrator. In this case the Integrator use cases are

invoked, see [1].

Once the Simulation Operator is ready to execute a simulation, the interactions with the MSaaS Capability include the ability to provide information about the simulation execution and provide input on how the composition should be deployed, accessed, and executed. The simulation execution should be repeatable thus must be provided to capture all the information and choices. The choices to influence the execution include amongst others deployment options, access controls, number of executions (runs), and date and time of the event.

1.2.2 Access Simulation Results

Once the simulation event has concluded, the interactions with the MSaaS Capability include the ability to access the results of the simulation. The types of results vary depending on the composition. Results also include general service management and control data, such as metrics and log data.

Types of results include:

1. Simulation data includes:
   - Raw data files.
   - Processed output files.
   - Imagery.

2. Scenario data includes:
   - Scenario-based outputs:
     - Battle-related items such as kills, casualties, shots fired, phase line times, detections, etc.
     - Movement of entities over time.
   - Playback – ability to view the execution at any point in time, at variable rates of playback times, pausing, etc.

3. Service management and control data includes:
   - Computing performance logs (e.g. CPU and memory utilization) in order to easily see if there may have been a problem.
   - Networking performance to determine if the network was too saturated for ideal performance or if there were any connection problems during the event.
   - Error logs.
   - Simulation management application logs.
   - Metrics, such as number of service requests and responses.

Simulation data and scenario data are assumed to be specific to the composed simulation service. For instance, capabilities to record and post-process simulation data must be included in the composition itself, rather than expected to be provided as part of the Enabling Services of the MSaaS Capability, e.g. Data Recording Services.

Service management and control data is more general and the required functionality is provided by the Service Management and Control (SMC) capability within the MSaaS Capability. SMC is a crucial capability that allows the Simulation Operator to orchestrate compositions, obtain metrics and log data, and monitor services and compositions with the MSaaS Capability. We'll discuss service management and

control in more detail in the next chapter.

2.0 EVERYTHING AS CODE OR NEED FOR ORCHESTRATION

This chapter introduces the concept of "orchestration", a programmatic approach to the management and control of services and compositions. Several technologies are available that can be used to realize Service Management and Control within an MSaaS Capability, and this chapter provides a brief overview of these technologies. The next chapter elaborates on Kubernetes, one of the technologies.

2.1 Orchestration as solution to complexity Growth

Services provide users with "what is needed", i.e., enable access to certain capabilities using some defined interface. In the Information and Communications Technology (ICT) context, services may provide resources from the whole stack, ranging from the 'low level' (e.g., storage) to 'high level' (e.g., database). Services frequently consume what is provided by the other services (e.g., database needs a physical storage), creating structures (hierarchies, meshes etc.).

With the growing level of complexity and interdependencies, manual deployment and management of the services can make the process slow and difficult and/or can lead to the non-reproducible or even erroneous results. The problems can occur both across repetitive deployments or usage of the same service (e.g., when performing multiple runs of the simulation) or when porting a service from one environment to another (e.g., from testing to production). Some of the contributing factors are:

- Inconsistency in the infrastructure elements such as different versions of the operating systems, modules (e.g., software switches), applications and/or libraries they depends on;
- Inconsistent application configuration parameters;
- Differences is network topology, addressing or naming schemas;
- Variations is storage backend used, e.g. Hard Disk Drive (HDD) vs Solid State Drive (SDD).

Orchestration or *Everything as Code* (EaC) is one of the solutions to the aforementioned problem. It assumes a programmatic ("software-defined") approach to i) provision and manage the infrastructure to run the services ii) operate the services themselves. It exposes Application Programming Interfaces (APIs) which allow to automate tasks like deploying the servers (bare metal or virtual), installing operating systems, creating network fabric, providing storage options and deploying, configuring and managing the lifecycle of the applications.

The 'code' can be understood both as a piece of software written in (high-level) programming language (e.g. Python) as well as a text descriptor (e.g. a "yaml" file) that describes the infrastructure and the services, following a certain information model.

Since the 'code' is human-readable, this approach allows for easy versioning, tracking changes and sharing using standard tools (e.g., Git) and using Continuous integration/continuous deployment (CI/CD) pipelines for development, testing and deployment of services within the described infrastructure.

Typical steps for deployment and operations of the services which can be coded/automated are (mostly cited from [2]):

- Configuration management is the process of deploying and managing at runtime all the required

services (e.g. Tomcat, MySQL, Hadoop, Cassandra). It consumes reusable recipes – deployment descriptions - that (often declaratively) describe how to manage and configure the service across lifecycle phases.

- Server provisioning entails acquiring (e.g. from a public provider of Infrastructure-as-a-Service), configuring and running the required VMs or containers upon which services can run.

- Application deployment is the phase in which user's applications are executed on the resulting infrastructure.

- Monitoring, metering and logging activity of the running components (VMs, services, middleware and applications) is a basic enabler for almost any kind of runtime management activity.

- Self-adaptation is about applying a set of methods (e.g. VMs autoscaling, faults recovery, etc) to keep the status conforming with service-level agreements and the quality of service (QoS) variables therein (e.g., security, network safety, etc.).

Several technologies, both open source and commercial, address some or all of the steps listed above. Some of these technologies use industry standards (for example YANG – Yet Another Next Generation [3] as a data modelling language), while some define their own and due to popularity can be considered as de facto or community standards. The next section provides a brief overview of available technology for orchestration.

2.2 Technology for orchestration

In this section we provide a brief overview of available technology for orchestration, focusing on open source solutions. The technology is categorised using the deployment automation technology categorization described in [4].

General purpose technology

General purpose technologies support many deployment features and mechanisms. These technologies support single-, hybrid-, and multi-cloud deployments as well as different kinds of cloud services (XaaS). General purpose technology includes: Ansible, Chef, Terraform, JuJu, OSM, and ONAP. Several other technologies exist such as OpenStack Heat, SaltStack, Cloudify. They are, however, either less popular or – like Heat – or focus on a specific backend. It is out of the scope of this paper to discusss and compare these technologies in detail. The reader is referred to “The essential deployment metamodel: a systematic review of deployment automation technologies” [4] for a brief description of these.

Platform specific technology

These deployment technologies support multiple cloud providers, the creation of reusable entities, and influencing a component’s lifecycle (deployment features and mechanisms). In contrast to general purpose technology, these are restricted in terms of the cloud delivery model and the use of specific platform bundles for realizing components. However, these technologies offer definite advantages over the general purpose technologies when standardizing on such technology. Kubernetes is the best known platform specific technology and is used for container orchestration. However, there are other (and sometimes lesser-known) container orchestration environments that fall in this category. For example, Docker, CoreOs, Apache Mesos. The reader is referred to “Container orchestration environments for M&S” [5] for an overview. Kubernetes is the topic of the next chapter.

3.0 ORCHESTRATION USING KUBERNETES

Kubernetes (k8s) is an orchestration environment solely intended for containerized workloads (i.e., cannot orchestrate virtual machines nor physical devices) and is a current de facto standard regarding this technology. The original code base has its roots in Borg – a production-grade orchestration system used by Google and there is currently a large and lively community behind the Open Source development and associated ecosystem of Kubernetes [6].

Kubernetes provides many capabilities out of the box for Service Management and Control as discussed in the MSaaS Techical Reference Architecture [7]. This includes amongst others container monitoring, metering and logging services; workload deployment services; load-balancing services; and service discovery services.

In this chapter we will not cover the technical components of Kubernetes. A good source of information is the Kubernetes site itself [6], or Wikipedia [8]. Some of the basic concepts associated with Kubernetes are discussed in the next section. The remaining sections that follow discuss the use of Kubernetes as an MSaaS Capability.

3.1 Kubernetes concepts in a nutshell

This section provides a brief overview of some of the Kubernetes concepts. Some of the most relevant concepts and their relationships are illustrated in Figure 2.

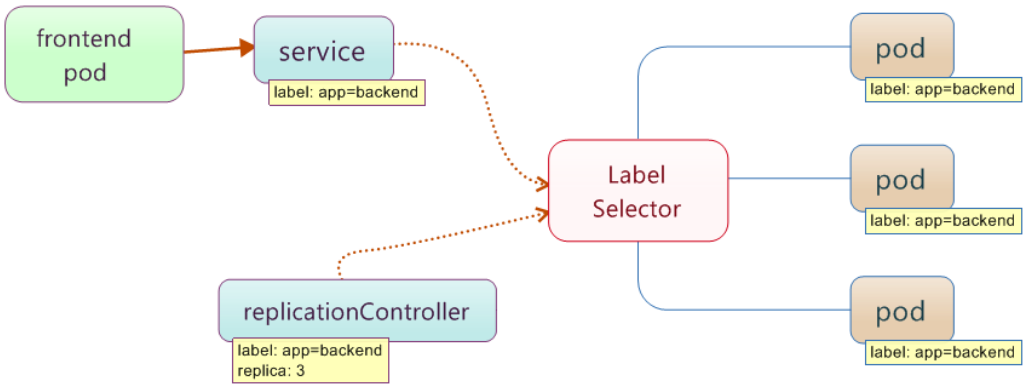

Figure 2: Kubernetes concepts: Service, Pod, and Replicas.

A *Pod* is the basic building block of Kubernetes and represents a group of containers and volumes, scheduled together on one node in a Kubernetes cluster. A pod has a unique IP address that is shared by every container in the Pod. Containers inside a Pod communicate with each other using the address “localhost”. Data volumes are also shared between the containers in a Pod. A Pod typically represents an application that provides a Service.

A *Service* is a network endpoint that refers to a set of Pods in the cluster. A Service has an IP address, a port, and a Label Selector. The set of Pods targeted by the Service is determined by the Label Selector. A Service is accessible from any node, and from external clients depending on the network endpoint configuration.

A *Label* is a key/value pair that can be attached to, for example, a Pod. A Label Selector is an expression that matches labels to filter certain resources, in this case Pods. With a Label Selector, a Service can identify the set of Pods that provides the service. A service request is subsequently load-balanced and routed to one of

the Pods.

Multiple copies of a Pod are called *replicas*. A *ReplicationController* ensures that a specified number of Pod "replicas" are running at any one time. If there are too many Pods, it will kill some. If there are too few, the ReplicationController will start more.

A *cluster* is a collection of Kubernetes nodes, consisting generally of Master and Worker nodes, see Figure 3. Nodes can have different roles. For fault tolerance purposes functionality and data may be replicated across nodes. Worker nodes generally execute the user Pods and Services. Important to note is that all Pods are attached to a single *overlay network*, throughout the cluster. Several implementation for overlay networks are available and may be used. Containers within a Pod communicate via their own "localhost" network.

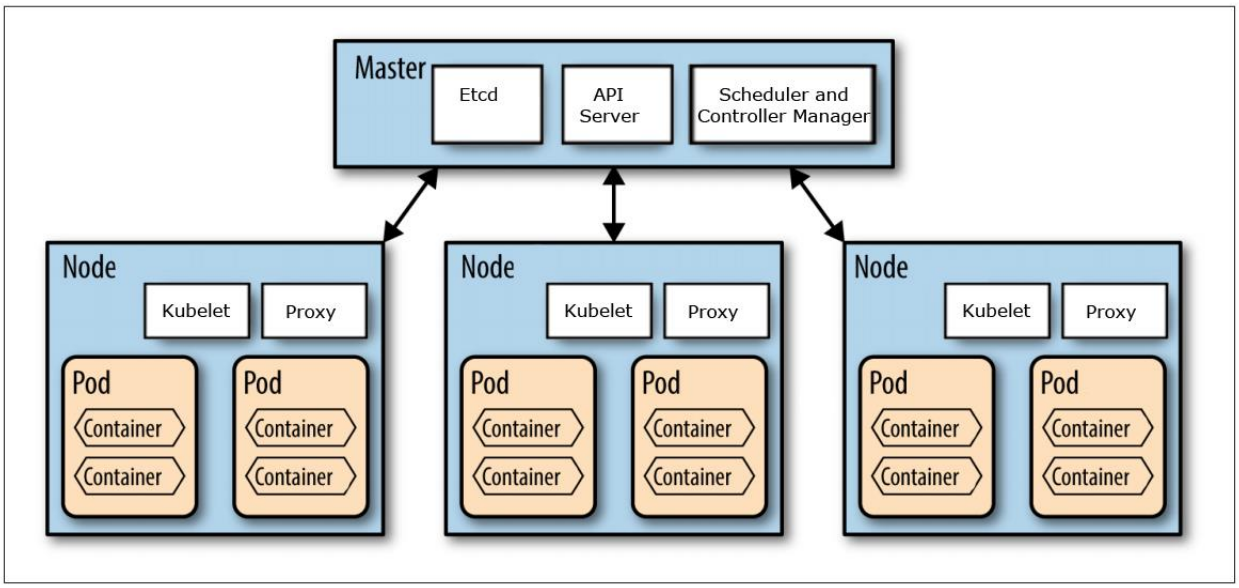

Figure 3: Kubernetes nodes (from [9]).

A Helm Chart (i.e. a *deployment description*) is the de facto standard for describing a Kubernetes application, and for deploying a Kubernetes application in a Kubernetes cluster. It provides information about the Pods, Services, etc, to be deployed. A Helm Chart is a so called *declarative* deployment description (see [1]) and describes the desired end-state of the deployment.

3.2 MSaaS in a Box

Kubernetes is an orchestration environment that can be used to realize a low-cost and feature rich MSaaS Capability, focussed on containerized workloads. Figure 4 illustrates Kubernetes as an MSaaS Capability implementation, from which a coherent set of simulation services can be offered to Users.

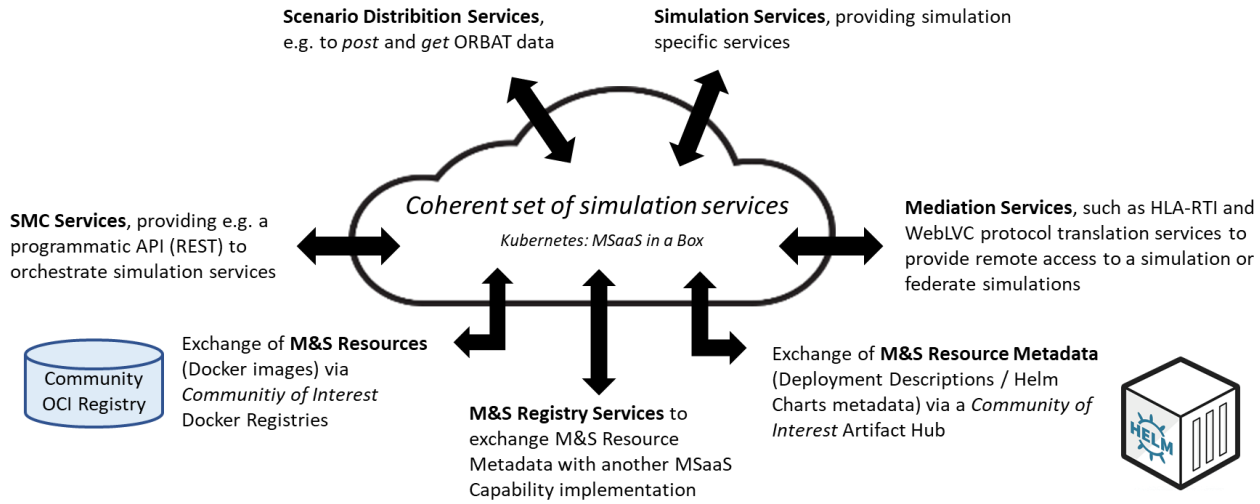

Figure 4: Kubernetes: MSaaS in a Box.

The MSaaS in a Box includes a Docker Registry to store container images, a Git repository to store Helm Charts, and a service catalog to browse for and start services. M&S Enabling Services in the catalog include (implementations of) Scenario Distribution Services (e.g. an MSDL ORBAT Server), Mediation Services (e.g. WebLVC Server), Message Oriented Middleware Services (e.g. Pitch or VTMaK HLA-RTI, Gateways, and WAN connectors such as Pitch Booster), and Logging and Metering Services (e.g. Prometheus). These services enable (other implementations of) Simulation Services to function within an MSaaS Capability, serve users, and provide the means to participate in a federated simulation.

The MSaaS Capability also provides the means to be federated in a larger MSaaS ecosystem, with the aim to share M&S Resource Metadata and M&S Resources, and to share SMC data. In the figure these means are provided for by SMC Services, and by the ability to share cloud artifacts through *Community of Interest* Open Container Initiative (OCI) Registries and an Artifact Hub. A Community of Interest OCI Registry and an Community of Interest Artifact Hub should typically be serviced by one of the MSaaS Capability implementations in the ecosystem, providing general M&S Resources to the ecosystem. Much like the role of the Docker Hub and the Artifact Hub on the public internet, where the whole world is the ecosystem.

The sevice catalog and the Artifact Hub are discussed in more detail in the following sections.

3.2.1 Service catalog

Kubernetes provides a UI to manage and control workloads in a Kubernes cluster, and provides the ability to deploy a catalog such as Kubeapps [10], from which workloads (services, compositions) can be browsed and started. The data that is shown in the catalog is retrieved from Helm Chart repositories. These are typically hosted in a private (company) or a public Git repository.

An example of a catalog is provided in Figure 5. This catalog is served by Rancher [11], an open source product that packages and delivers Kubernetes as a service. From this catalog the Simulation Operator can start a service or a composition with just a few clicks. After selecting the Pitch RTI (CRC), the Simulation Operator can provide deployment-specific configuration options as shown in Figure 6, before deploying the application.

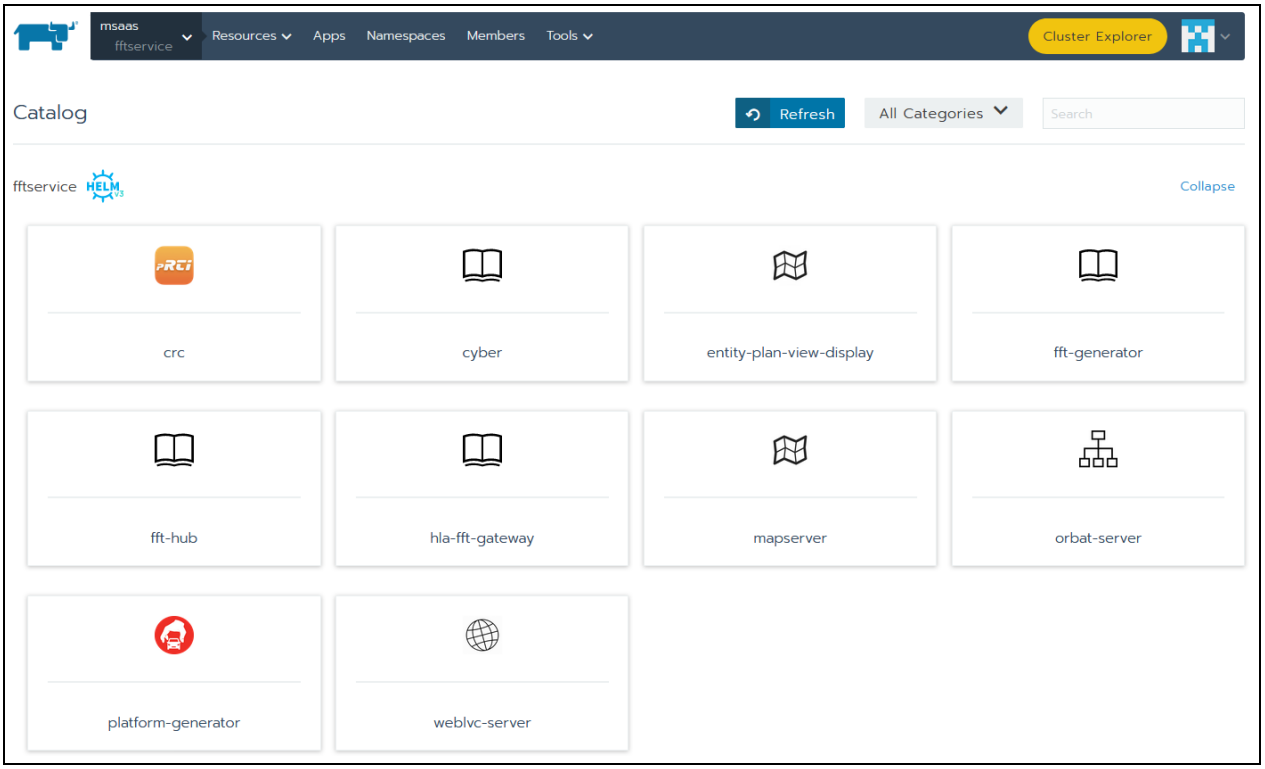

Figure 5: Rancher Helm Chart catalog.

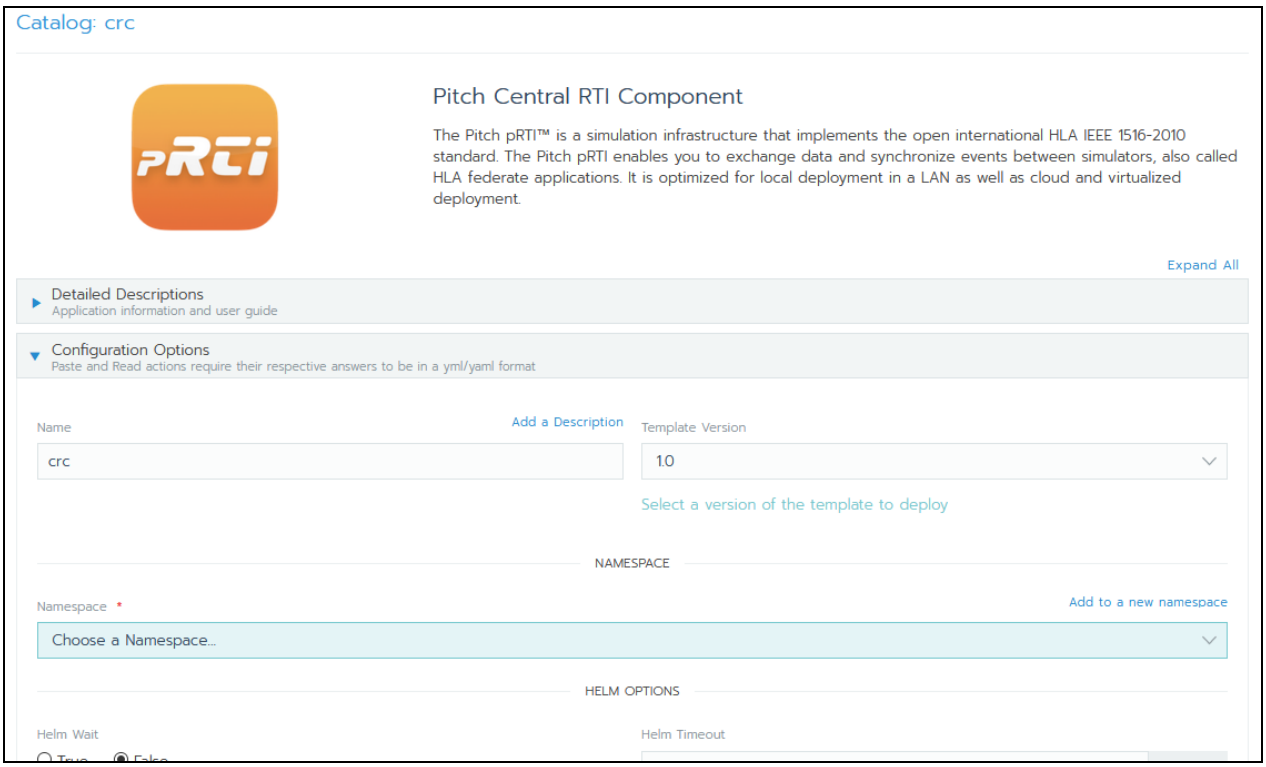

Figure 6: Rancher Helm Chart configuration options.

The service catalog is just one way of starting services. Services may also be started or stopped via a

command line tool called Helm, or via a Kubernetes REST API. This API enables the implementation of a more specific catalog and the integration with for example metadata repositories and composition tools, thus supporting the *Everything as Code* approach.

3.2.2 Helm Charts and Artifact Hub

A Helm Chart (i.e. a *deployment description*) is a collection of text (YAML) files to describe, and to deploy and upgrade a Kubernetes application with a tool called Helm [12]. Helm itself is a commandline tool, but a product like Rancher provides a graphical front-end to deploy, upgrade or de-install a chart, i.e. the Helm Chart catalog described in the previous section. Helm Charts may also be served from a dedictated Helm Chart repository server such as ChartMuseum [13]. The use of such a repository server is comparable to how package management for Linux works.

A recent development is that of the Artifact Hub [14], as a central place where various cloud-related artifacts such as Helm Charts can be published and searched. The Hub does not store the artifacts itself, but harvasts the metadata from the provided sources and makes the metadata available, to be searched in one place. The Artifact Hub is an open source project and the software can be deployed in an on-premises Kubernetes cluster, obviously using Helm. The opening page of the public Artifact Hub is shown in Figure 7.

A search for “crc” yields in this example only one cloud-artifact, as shown in Figure 8. Clicking on the discovered artifact provides more information such as type of artifact, version information, where to get it; see Figure 9. Note that this information is all meta-data. The artifact itself (a Helm Chart in this example) is located in the supplier’s repository.

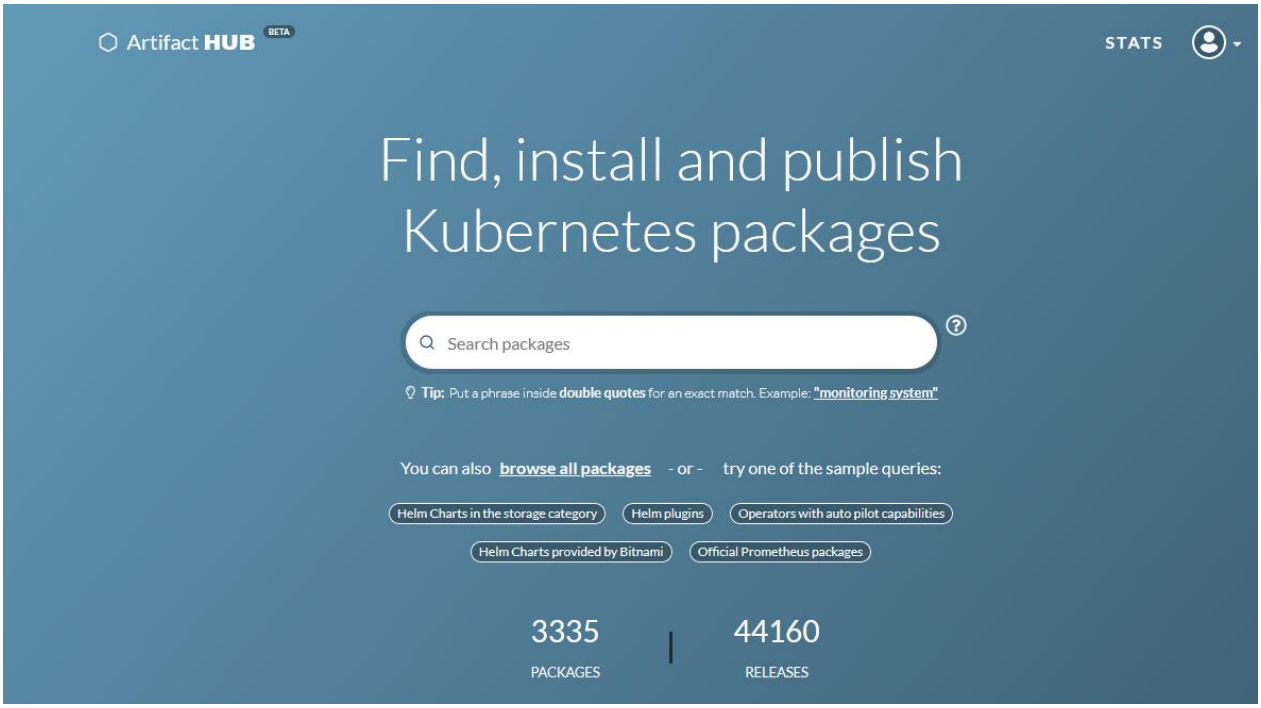

Figure 7: Artifact Hub to share and search cloud-related artifacts such as Helm Charts.

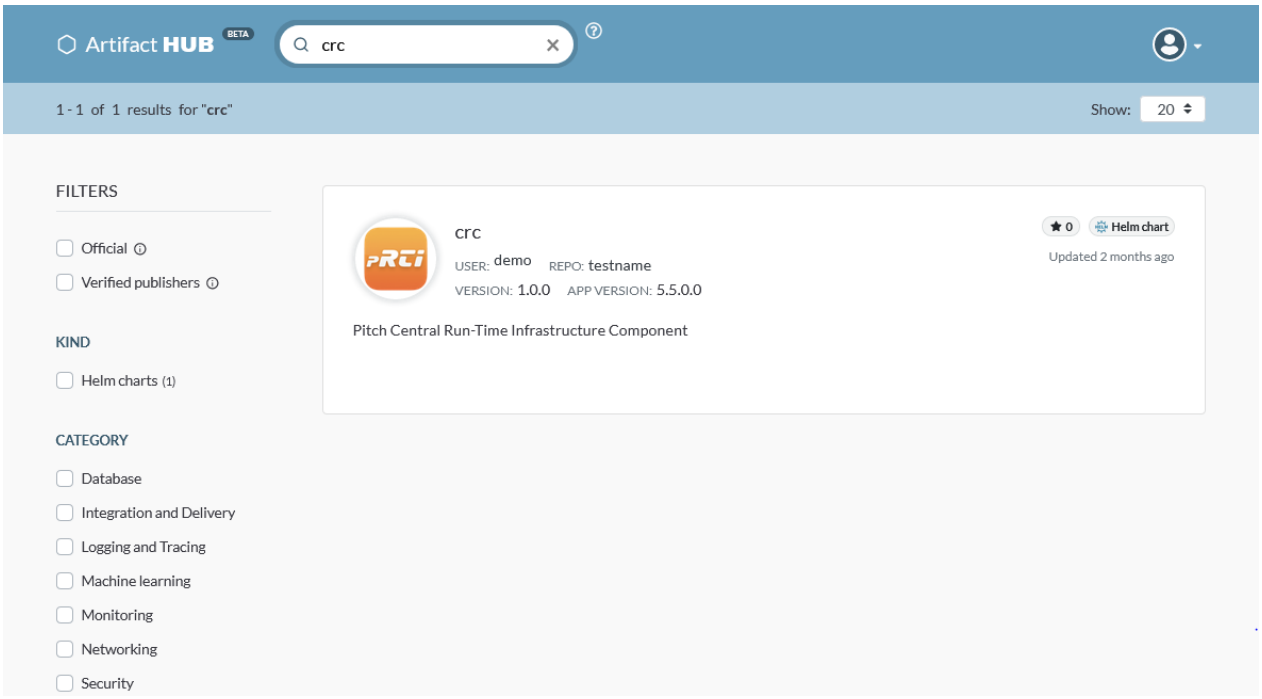

Figure 8: Discovered cloud-artifact for crc.

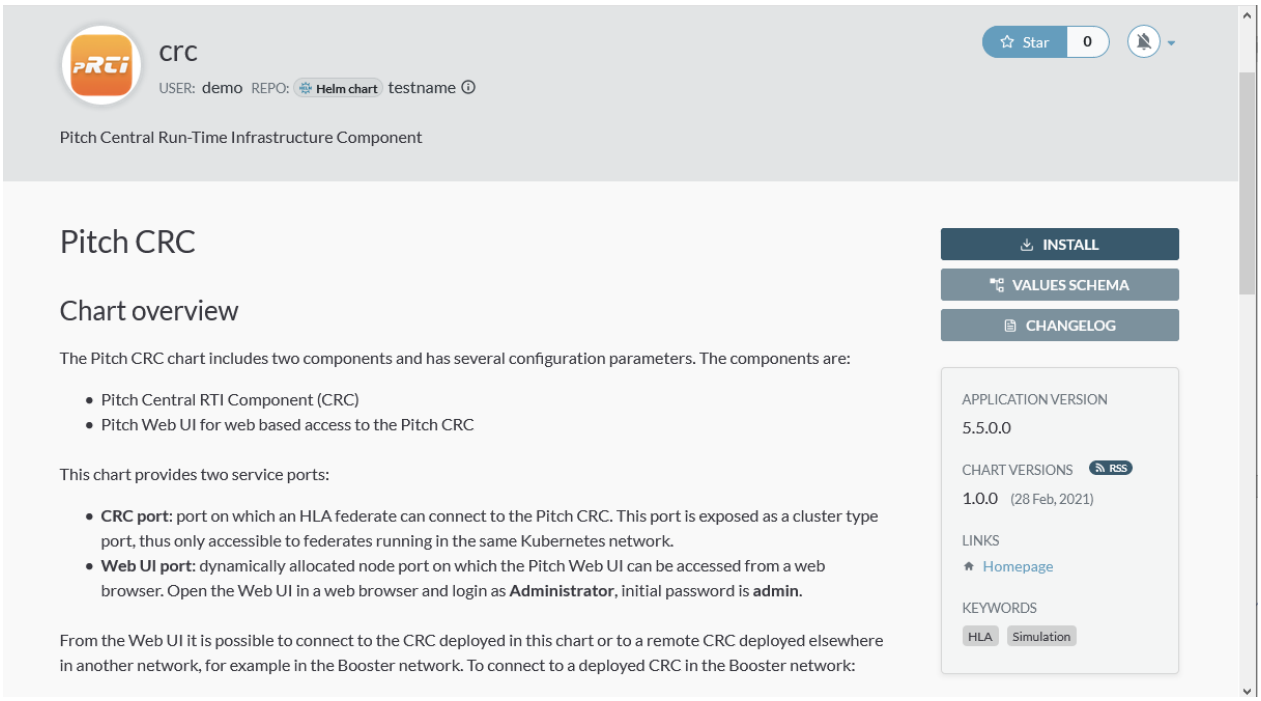

Figure 9: Cloud-artifact information.

3.3 On-demand simulation services

This section provides an example of how an MSaaS Capability based on Kubernetes can be used to on-

demand provide Friendly Force Tracking (FFT) as a service. The service can be used to stimulate a C2 system with ADatP-36 compliant Blue Force Tracks.

The MSaaS Capability provides the means to execute and manage (on-demand) a variety of simulation services in a cloud based environment. The FFT Service can be deployed on-demand, and replicated with different simulation scenarios to serve multiple C2 systems simultaneously. At the back-end the FFT Service is compliant with NATO AMSP-04 NETN [15] and several other SISO (Simulation Interoperability Standards Organization) simulation standards, providing interoperability between components and with simulations from partner nations. These simulations can connect to the service via e.g. Mediation Services. See Figure 10.

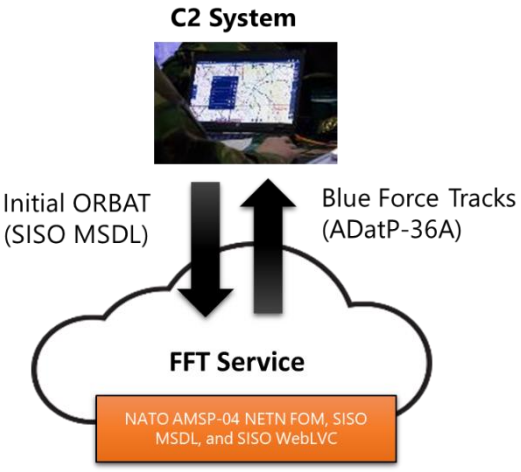

Figure 10: FFT Service.

The simulated tracks are generated by VR-Forces, one of the cloud-based components of the FFT Service. Other components in the FFT Service include an ORBAT Server to process the initial ORBAT data from the C2 System, an HLA-FFT Gateway to serve simulated ADatP-36 tracks, and an Entity Plan View Display to task simulated units. The latter provides a web-based user interface to the C2 user to task simulated units that are controlled by VR-Forces and potentially other simulation systems from partner nations.

The notional activity workflow to request and use the FFT Service is summarized in Figure 11, starting with a request for an FFT Service from the C2 (ELIAS) User (1).

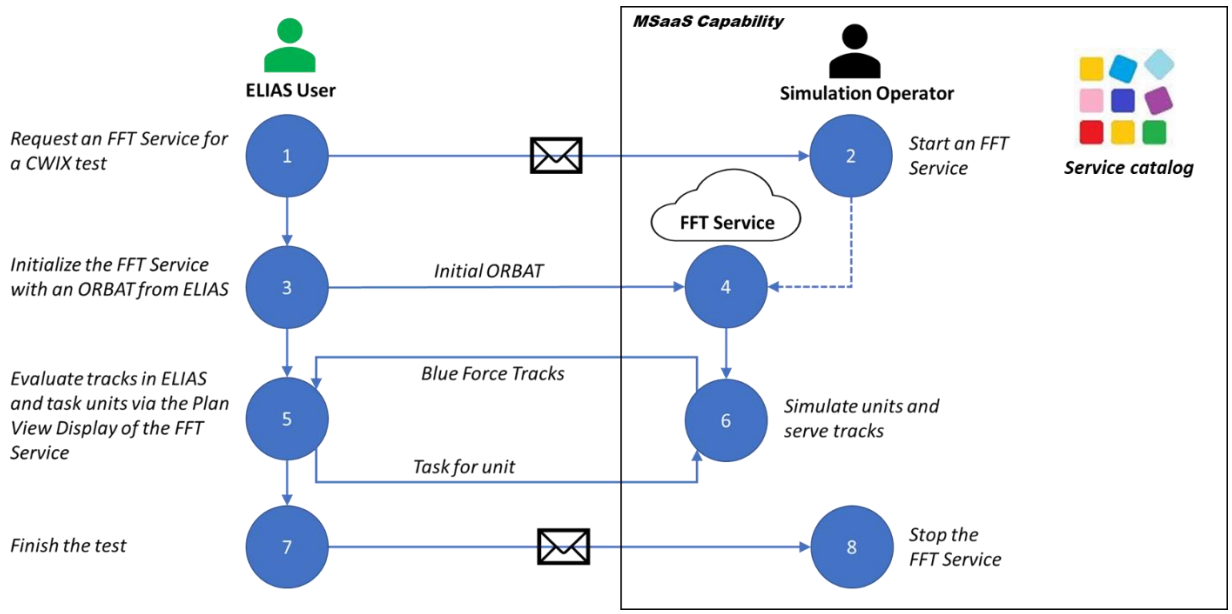

Figure 11: FFT Service request workflow.

The service catalog provides ready to deploy applications. In (2) the Simulation Operator selects the relevant services in the catalog, enters service configuration options (e.g. radio parameters, service access data) and presses the "Launch" button to deploy the application, see Figure 12. The C2 System can now connect to the FFT Service, although no tracks are reported yet.

The initial ORBAT data is exported from the C2 System in Military Scenario Definition Language (MSDL) format and posted to the FFT Service by the user in (3). The simulation initializes.

In (5) the C2 user tasks units from the Entity Plan View Display, see Figure 13. The resulting AdatP-36 tracks reported by the command unit CP-35 are reflected in the C2 System.

And finally in (7) the C2 user requests the FFT Service to be terminated.

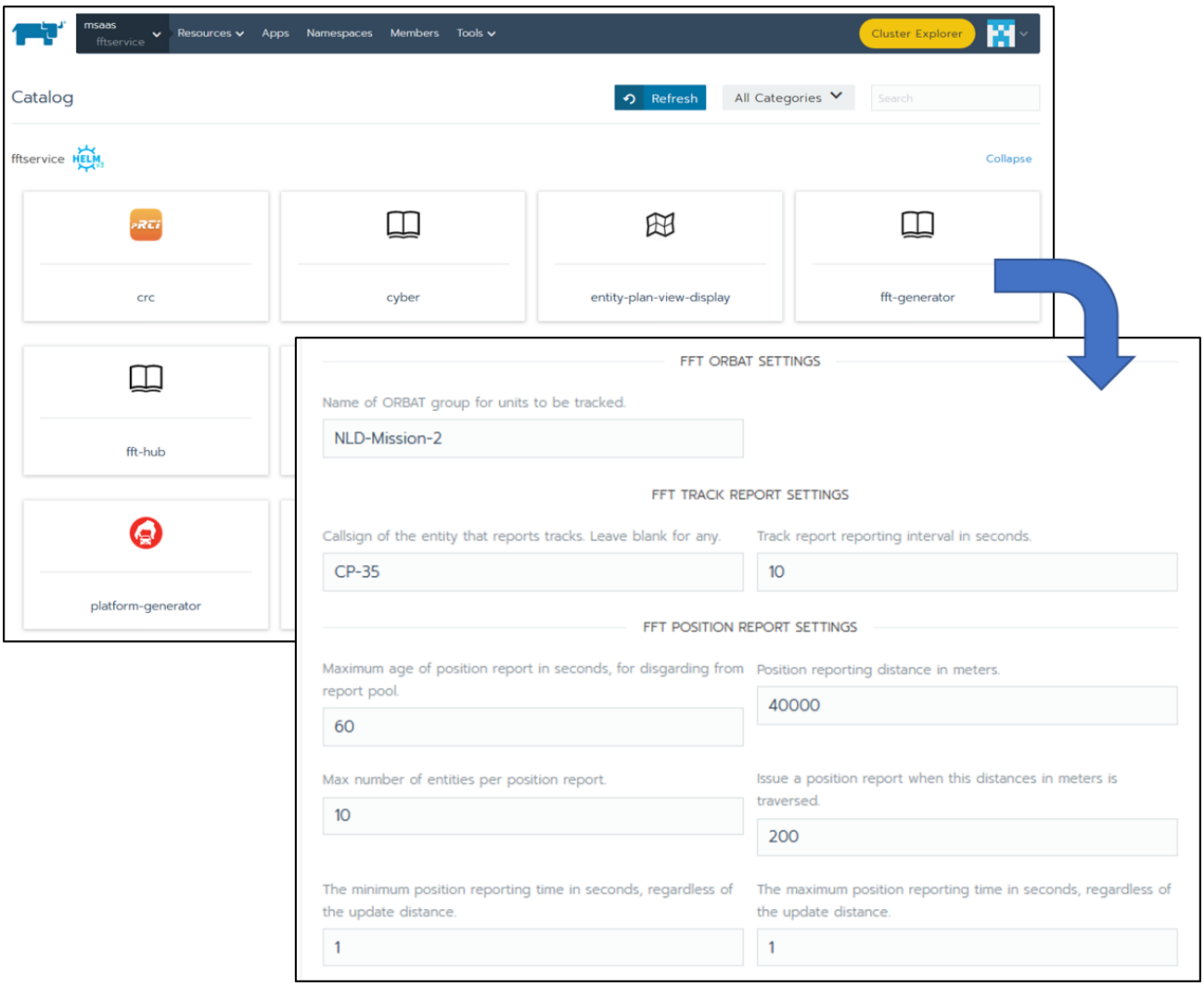

Figure 12: FFT Service deployment options can be provided via the catalog.

Figure 13: Entity Plan View Display to task units.

4.0 SUMMARY

As the complexity of the simulations systems grows, the need and benefit of automated orchestration becomes evident and there is a number of (open source) technology covering one or several aspects of this process. At the same time, there is always a cost associated: setting up an orchestrating environment, adjusting or re-packaging current simulation components to be compatible with the orchestration environment of choice, re-training personnel and possible altering the working habits, just to name a few.

The trend for newly created (micro) services is to deploy them as containerized applications, and in that case Kubernetes is a common choice. However, one must remember that it requires that the underlying infrastructure capable of hosting the Kubernetes cluster and the Kubernetes cluster itself are already available. Provisioning of the infrastructure requires therefore different tools. Furthermore, in practice, one can be faced with the situation that a (part) of a service is not containerized. In addition, there will always be a need for edge/physical services (e.g. sensors, gateways, data collectors) to connect live (and distributed) systems to a cloud environment. In these cases, Kubernetes cannot be the only solution and specific edge services or tools which can orchestrate multiple types of resources are required.

Finally, effort should be directed to adapting the “everything as a code” approach and use orchestrators such as Kubernetes to run simulation components. For new applications, containerized deployments orchestrated by Kubernetes are recommended, also leveraging the vast open source ecosystem that surrounds Kubernetes. For legacy applications a migration/modernization strategy should be developed, e.g. by initially running the application in a virtual machine as is. At the same time, hybrid solutions can be used e.g., using both containers and virtual machines, managed by orchestrators like OSM or OpenStack which can handle both types of workloads.

5.0 ACKNOWLEDGEMENTS

The author would like to acknowledge to contribution of MSG-164 in relation to the content of this paper; in particular members of the MSG-164 TEK team.

References

[1] Berg, "MSaaS Composition and Technical Approach," in *MSG-168 LS*, 2021.

[2] M. Artac, T. Borovšak, E. Di Nitto, M. Guerriero, P.-P. D and T. D, "Infrastructure-as-Code for Data-Intensive Architectures: A Model-Driven Development Approach," in *IEEE International Conference on Software Architecture (ICSA)*, Seattle, 2018.

[3] M. Björklund, "YANG -A Data Modeling Language for the Network Configuration Protocol (NETCONF); RFC6020," [Online]. Available: doi:10.17487/RFC6020.

[4] M. Wurster, U. Breitenbücher, M. Falkenthal and e. al., "The essential deployment metamodel: a systematic review of deployment automation technologies," *SICS Softw.-Inensiv. Cyber-Phys. Syst.*, vol. 35, 2020.

[5] Berg, "Container orchestration environments for M&S (SISO SIW 2017-17F-006)," in *SISO Fall SIW*, Orlando, 2017.

[6] "Kubernetes," 2021. [Online]. Available: https://kubernetes.io.

[7] Berg, "MSaaS Technical Reference Architecture," in *MSG-168 LS*, 2021.

[8] "Kubernetes," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Kubernetes.

[9] "Kubernetes, at a first glance," [Online]. Available: http://nishadikirielle.blogspot.com/2016/02/kubernetes-at-first-glance.html.

[10] "Kubeapps," GitHub, [Online]. Available: https://github.com/kubeapps/.

[11] "Rancher," Rancher, [Online]. Available: https://rancher.com.

[12] "Helm," [Online]. Available: https://helm.sh.

[13] "ChartMuseum," [Online]. Available: https://github.com/helm/chartmuseum.

[14] "Artifact Hub," [Online]. Available: https://github.com/artifacthub/.

[15] "NATO Distributed Simulation Federation Object Model (NETN FOM)," [Online]. Available: https://github.com/AMSP-04. [Accessed 2020].