A Reusable Approach to Network Service Orchestration

Roman Lapacz, Sonja Filiposka, Frédéric Loui, Tomasz Szewczyk, Marcin Adamski, Konstantinos Georgilakis

A Reusable Approach to Network Service Orchestration

Roman Łapacz
Poznan Supercomputing and Networking Center - PSNC
ul. Jana Pawła II 10
61-139 Poznan, Poland

Sonja Filiposka
Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University
Rugjer Boshkovikj 16
1020 Skopje, North Macedonia

Frederic Loui
RENATER
23-25, rue Daviel
75 013 Paris, France

Tomasz Szewczyk
Poznan Supercomputing and Networking Center - PSNC
ul. Jana Pawła II 10
61-139 Poznan, Poland

Marcin Adamski
Poznan Supercomputing and Networking Center - PSNC
ul. Jana Pawła II 10
61-139 Poznan, Poland

Konstantinos Georgilakis
Greek National Technology and Research Infrastructure Network - GRNET
7 Kifisias Avenue
115 23 Athens, Greece

Abstract

The Network Infrastructure as Code paradigm empowers network engineers to create a repository of configuration templates and automation scripts, facilitating the automated configuration, management, and monitoring of the network infrastructure and services. Moreover, applying standardised approaches to high-level network processes, utilising a common data model for network description, and employing standardised open APIs for components, can offer reusability of the implementation of automated and orchestrated network solutions. In its recent activities, the GÉANT Global Platform for Labs (GP4L) group is actively collaborating with the Polish national research and education network (NREN) PIONIER focusing on developing a generalised, reusable approach to orchestrating service provisioning. By abstracting the process to define common steps in service provisioning, the goal is to provide a collaborative blueprint that other NRENs can adopt. The modular nature of the approach allows the creation of a repository of processes, sub-processes, and elementary tasks that can be reused across the community. This article presents the concept of developing generalised reusable processes and discuss challenges and decisions while working on the PIONIER use case. Aiming for reusability and sharing lessons learned, it builds towards a community of network automation and orchestration activities.

Keywords

Automation, orchestration, standardisation, workflows, service provisioning, network management

1 Introduction

By integrating advanced digital technologies into organisational processes, digital transformation [1] has led to significant changes in various sectors, including network management. The adoption of technologies such as cloud computing, artificial intelligence, and big data has introduced new demands for agility, flexibility, scalability, and real-time responsiveness in networks. These evolving requirements are fundamentally altering the landscape of network management, necessitating a shift away from traditional manual methods toward automated, efficient approaches.

The NRENs, operating in complex and diverse research and education environments, face unique challenges in planning and managing their networks. The need to support a wide range of services, often with stringent performance requirements, coupled with the rapid pace of technological advancements, demand a high degree of adaptability. The increasing complexity and criticality of networks in research and education (R&E) further underscore the need for moving towards automation and orchestration, essential enablers of streamlined network operations. Thus, in the past few years, the NRENs have turned their attention to taking advantage of the potential of network automation and orchestration.

Furthermore, the adoption of Infrastructure as Code (IaC) [2] and NetDevOps [3] principles can significantly accelerate the transition towards automated and orchestrated network management. IaC enables the definition and management of network infrastructure using code, promoting consistency, reproducibility, and version control. NetDevOps, a collaborative approach combining network engineering, development, and operations, fosters a culture of continuous integration, continuous delivery, and automation. By adopting these practices, NRENs can accelerate the deployment of network services, improve operational efficiency, and enhance overall resilience.

The growing availability of open-source tools and frameworks has spurred a wave of experimentation and small-scale projects within the R&E network community. However, it has been recognised that to advance more effectively by reducing the learning curve, and fostering a shared understanding of best practices in network orchestration, there is a strong need to leverage existing knowledge and promote the adoption of common, preferably standardised, approaches. This more generalised strategy will enable faster progress and greater collaboration across NRENs and the broader R&E networking community.

The Global Platform for Labs (GP4L) team within the GÉANT GN5-1 project aims to support the community in developing and adopting effective network automation and orchestration solutions. By investigating and piloting a generalised approach to network service orchestration, GP4L seeks to contribute to the collective knowledge base and promote a common framework for building and deploying such solutions. In this way, the proposed approach will not only benefit individual organisations but also foster interoperability and collaboration within the broader community. To achieve this goal, GP4L has collaborated with the network engineers of the Polish R&E network PIONIER. This paper presents the ongoing work on a pilot implementation. The solution is designed to be easily adopted and reused by other network operators, leveraging open-source tools

and a reusability-by-design approach. By sharing our experiences and lessons learned, we hope to inspire and guide others in their journey towards effective network automation and orchestration.

The rest of this paper is structured as follows: In the second section we introduce the concept of generalised network service orchestration by defining the necessary generalised workflows and related data model. We then continue with discussing the tools selected for implementation of the orchestration architecture and the structured approach to storing and using the developed code. In this section we also provide the details of the pilot as an implementation example. Finally we summarise the main point of the paper and discuss future work.

2 Generalised network service orchestration

Network service orchestration [4] aims to streamline and automate the provisioning and lifecycle management of network services. It encompasses both business and operational activities, ensuring that network services are delivered efficiently and aligned with procedures and business objectives.

In the design stage the balance between service implementation approaches and service-agnostic approaches should be considered. Service implementation approaches involve defining specific workflows for each type of service, providing tailored orchestration capabilities. On the other hand, service-agnostic approaches focus on abstracting common patterns and processes across different services, promoting reusability and flexibility. The high-level actions for service orchestration workflows may be service-agnostic, the lower-level tasks and configurations often require service-specific details, thus making a hybrid approach that combines elements of both approaches particularly effective.

2.1 Orchestration workflows

A generalised network service orchestration framework aims to abstract common patterns and processes to promote reusability across different network services such as IP connectivity, Point-to-Point, VPN, etc. By focusing on generalised concepts, network service providers can reduce the need for custom development for each new service. Additionally, this can foster collaboration and knowledge sharing within the network community, as organisations can benefit from the collective experience and best practices.

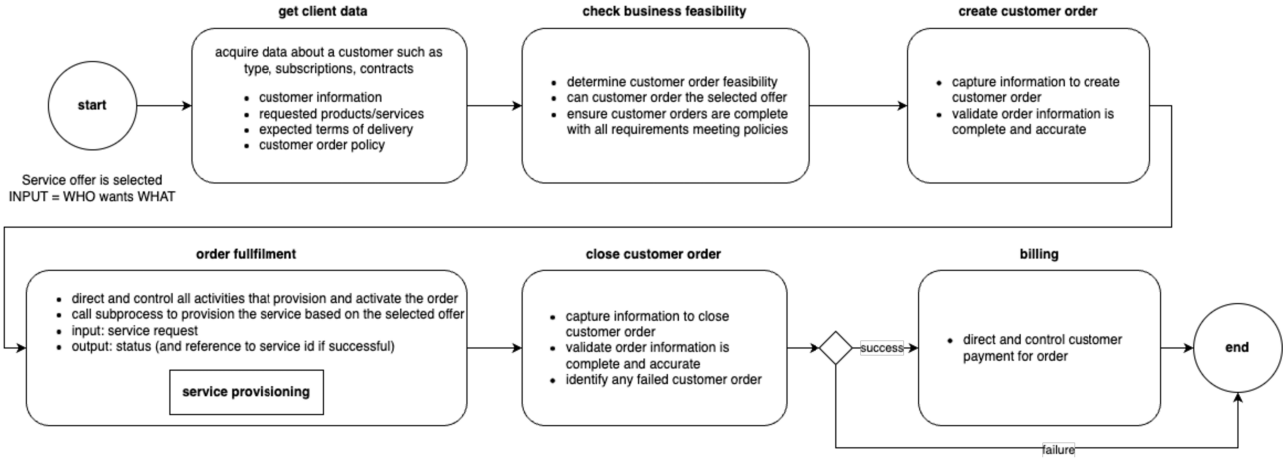

Figure 1 – Adaptation of the order-to-payment process for generalised network orchestration

To facilitate the orchestration of network services, a well-defined set of workflows is essential. The TMF Business Framework [5] provides a valuable starting point for developing these workflows. Building upon this framework, we can define generalised workflows that capture the common steps involved in orchestrating various network services.

For the work presented here we have chosen one of the developed general workflows: the order-to-payment process. This process outlines the key steps involved in provisioning a network service, from receiving a service order to completing the payment.

As presented in Figure 1, we aim to define generalised workflows that capture the common high-level steps involved in orchestrating network services. This modular approach balances the service-agnostic and service-specific elements.

2.2 Data modelling

Data modelling is essential for the successful implementation of generalised network service orchestration. A well-structured data model provides a common language for representing network services, their components, and their relationships. By defining clear data structures and relationships, we can facilitate the exchange of information between different components of the orchestration system and ensure data consistency.

To support the generalised nature of network service orchestration, a well-defined data model is required. It should be agnostic to specific service implementations. In this way the need for custom data structures for each type of service can be avoided, thus promoting reusability and flexibility. Additionally, it can facilitate the integration of different network components and systems, and it enables defining the inputs and outputs of the high-level workflow steps in a service-agnostic manner.

Using JavaScript Object Notation (JSON) objects to transfer data through the workflow tasks (subprocesses) is a structured and efficient way for managing complex workflows, particularly in distributed systems or environments where multiple tasks or services interact. In our pilot example the order-to-payment process begins with a high-level order request, which contains minimal information necessary to initiate the workflow. This initial JSON object acts as a skeleton or blueprint, outlining the fundamental parameters or identifiers needed to start the process.

As the workflow progresses, different tasks or components interact with the initial JSON object, gradually enriching it with more detailed information. Each task adds or updates specific fields within the JSON object as it completes its function providing more detailed information about the service being created.

For instance, the initial JSON object might only include basic details such as a request ID, customer ID, and a high-level task description. As individual tasks are executed — such as validating the request, checking an inventory, or processing service elements — additional fields are populated within the JSON object. For example, the inventory check task might add port availability data, while the payment task would add transaction details.

This approach offers several advantages:

1. Modularity: each task can focus on processing only the information it requires and updating only the relevant parts of the JSON object.
2. Scalability: the gradual construction of the data object reduces the need for complete information upfront, allowing systems to scale by deferring detailed computations to later stages.
3. Consistency: using a single JSON object throughout the process ensures data consistency across all tasks, reducing errors caused by mismatched or incomplete data being passed between components.
4. Flexibility: this method allows tasks to be decoupled, meaning new tasks can be added or existing ones modified without needing to redesign the entire data structure or workflow.

To maintain data consistency and ensure that the orchestration system has access to the most up-to-date information, it is essential to interact with an inventory that plays the role of the source of truth. It can be a centralised database, configuration management system, or other authoritative source of network data.

By integrating with a source of truth, the orchestration system can retrieve the necessary data to execute workflows and update the source of truth with the results of those workflows. This helps to prevent inconsistencies and ensure that the network configuration reflects the desired state.

3 Orchestration components

The implementation of a generalised network service orchestration solution typically requires at least the following components:

- Orchestrator Engine to define, schedule, execute and monitor workflows.
- Configuration Management tool to automate the configuration of network resources and services.
- Source of Truth in the format of a resource and service inventory to store and maintain the desired network state.

Based on the requirements and evaluation criteria defined for the PIONIER network, we decided to work with the following tools:

- Apache Airflow [6]: an advanced Orchestration Engine with a rich set of functionalities and powerful Application Programming Interface (API) which are discussed in more details later in the text.
- Ansible [7] as a Configuration Management tool with a simple, agent-less architecture, and powerful automation capabilities. It uses the concept of playbooks that define the tasks, such as creating interfaces or configuring protocols, to be executed on network devices in YAML format. To execute the tasks, Ansible provides different modules that can interact with various network devices using their Command Line Interface (CLI) or API. The connection to the device is done via Secure Shell (SSH).

– Maat [8] as a network Single Source of Truth offering flexible data modelling and ability to store and manage network data. It serves as a service and resource inventory that describes devices and services and their relationships in a structured and consistent vendor-agnostic manner using an extensible model. Developed as a free, open source solution, Maat exposes powerful TM Forum[1] compliant APIs for service and resource management, respectively, making it an excellent candidate for automation and orchestration implementations.

One of the reasons for choosing Airflow as the orchestrator engine is also the possibility to define the workflows in external dedicated Git Repositories. In this way, one can easily implement advanced version control capabilities, which enable collaborative development and tracking of changes (GitOps). In addition, Airflow's user management system allows for granular control over user permissions, enabling effective access management to sensitive network data and workflows.

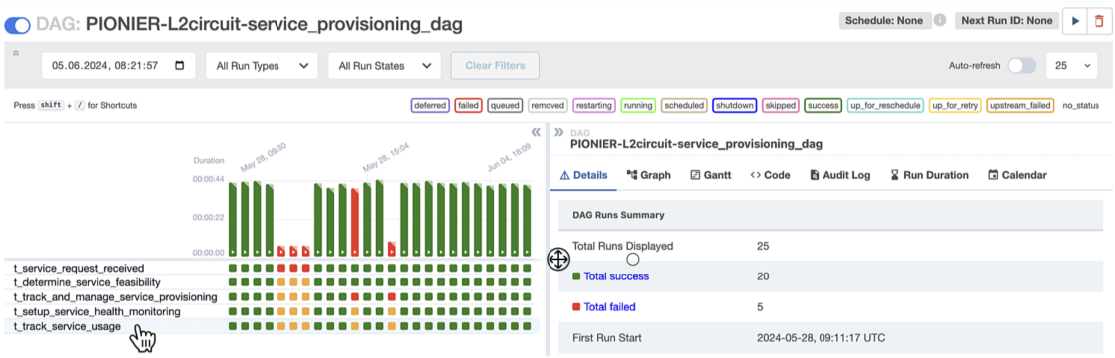

Figure 2 – Visualisation of one implemented DAG in Airflow

Directed Acyclic Graphs (DAGs) are the fundamental building blocks of Airflow workflows. They represent a workflow with series of tasks that are connected to form a directed acyclic graph. The DAG's structure ensures that tasks are executed in the correct order, preventing bottlenecks and ensuring efficient workflow execution. Airflow's intuitive dashboard allows for easy monitoring, troubleshooting, and management of the orchestration process.

DAGs are composed of individual tasks, which can be simple Python functions or more complex operators that interact with external systems. Tasks can be connected with dependencies, specifying the order in which they should be executed. DAGs can include branching logic to execute different tasks based on conditions, and sub-DAGs can be used to modularise complex workflows. DAGs can also be scheduled to run periodically or triggered by external events. In addition, Airflow supports retrying failed tasks and backfilling historical data to ensure data integrity. By effectively utilising DAGs and their features, one can create powerful workflows to automate various tasks, such as network service provisioning in our case.

3.1 Reusable workflow implementation

A generalised approach to network service orchestration offers significant advantages, particularly in terms of modularity and reusability. By decomposing workflows into smaller, reusable components, development time is reduced, and the overall system becomes easier to maintain. This modular approach allows for the augmentation of existing workflows with additional tasks or functionalities in a flexible manner. In particular, parts of tasks can be reused or adapted, allowing

1 https://www.tmforum.org/

for more granular control over the orchestration process. For example, a validation task could be reused in multiple workflows, reducing the need to write new code for each instance. Furthermore, different parts of a task, such as input validation, data transformation, or error handling, can be isolated and reused independently across workflows.

Workflow enhancements can be achieved through several strategies:

- Creating New Tasks: Custom tasks can be written in Python or any other programming language, as Airflow allows for the integration of tasks implemented in other languages, such as Bash, Java, or Go. This flexibility enables seamless integration with legacy systems.
- Modifying Existing Tasks: Current tasks can be customised or extended to meet evolving requirements. For instance, a task performing data validation could be updated to include new rules without rewriting the entire workflow. Alternatively, existing tasks can be augmented by adding support for more protocols or data formats, depending on the needs of the network service.
- Combining Tasks: Multiple tasks can be grouped into a single, more complex task, enabling streamlined orchestration of intricate processes such as service validation, monitoring, and reporting. This allows for greater efficiency when handling multifaceted network operations.

Apache Airflow has the ability to create so-called custom operators, which is a powerful feature allowing the interaction with a wide range of network components, including devices, APIs, and databases. By writing custom operators, developers can extend Airflow's capabilities to interact with specific technologies and systems that may not be natively supported. For instance, custom operators can implement advanced communication protocols or interface with specialised network equipment, ensuring seamless orchestration across various network environments.

The modularity of Airflow also allows for various approaches to network management tasks such as validation, monitoring, and configuration. Tasks can be designed to handle these functions in different ways, such as validating by doing a configuration dry run versus using a digital twin [9], i.e. a virtual copy of the network infrastructure, depending on the requirements of the network or the complexity of the service being orchestrated. Monitoring tasks can similarly be tailored to use different data sources, protocols, or frameworks depending on the network environment. For example, tasks can incorporate monitoring libraries such as Prometheus or Zabbix to collect and analyse network metrics. By decoupling monitoring from other tasks, workflows can be adapted to monitor different aspects of the network without affecting the underlying orchestration logic.

The use of modular tasks and reusable operators also allows for enhanced flexibility in higher-level orchestration. This means that entire workflows can be modified or extended by adding or removing tasks without impacting other parts of the system. For instance, a monitoring task could be added to a workflow to check network health during specific stages, without altering other validation or deployment steps. For an example, in our pilot implementation we developed an operator to interact with Maat. This enables the possibility for others to reuse the workflows as they are and only change the operator to be able to use a different Source of Truth.

Airflow supports clear separation of DAG definitions and their configuration parameters, which further enhances flexibility and reusability of the implementation. DAG definitions, which define

the overall workflow structure, can be stored in Python files, while configuration parameters, which control task execution, can be maintained in separate configuration files or passed as runtime arguments. This separation allows for easier customisation of workflows for different environments, network components, or operational scenarios. For instance, the same DAG can be used for different network services by simply changing the configuration parameters, eliminating the need for redundant code.

Efficient management and version control of DAGs, tasks, and operators are essential for large-scale orchestration systems. A hierarchical tree structure within a Git repository can be used to organise the sources of DAGs, tasks, custom operators, and configuration files logically. This structured approach allows for better code organisation, logical grouping of related components, and easier navigation through the codebase. Using a clear separation between different parts of the system such as workflows, component-specific operators, and configuration files, development teams can more easily collaborate, maintain version control, and ensure long-term maintainability.

A proposed organisation of the files in a Git repository that we are aiming to follow in our development of the PIONIER pilot is presented below.

```
├── dags/
│   ├── general/                    # General service-agnostic DAG definition
│   │   ├── tasks/                  # High-level reusable tasks
│   │   ├── sub_dags/               # Reusable sub-DAGs for common logic
│   └── service_specific/
│       ├── service_A/              # Service A specific DAG definition
│       │   ├── tasks/              # Service A specific tasks
│       │   ├── sub_dags/           # Service A specific sub-DAGs
├── operators/
│   ├── service_agnostic/           # Service-agnostic custom operators
│   └── service_specific/
│       ├── service_A/              # Service A custom operators
├── configs/
│   ├── service_agnostic/           # Global configuration for service-agnostic DAGs
│   └── service_specific/           # Configuration for Service A
├── utils/                          # Utility functions used across DAGs
└── docs/                           # Documentation for service A and service-agnostic workflows
```

3.2 PIONIER use case implementation

To demonstrate the application of our generalised network service orchestration approach, we have implemented a pilot for L2 circuit service provisioning in the PIONIER network.

After defining the general order-to-payment workflow, the team focused on one particular subprocess which is the service provisioning workflow presented in Figure 3. The main goal of the team was to define the tasks in such a way that it minimises the service specific parts. At this stage the track and manage service provisioning sub-workflow is fully functional wherein a digital twin (virtual representation) of the PIONIER network is used for validation purposes. As already mentioned, a digital twin is a virtual representation of a physical network (in our case, the PIONIER network) that allows the user to test configurations, troubleshoot issues, or simulate different scenarios before implementing them in the live environment. To build and manage the PIONIER network digital twin we decided to use containerlab [10]. This tool streamlines the creation and

management of the network twin within a containerised environment. It leverages container technology to run network devices such as routers and switches within isolated containers, loading the appropriate network operating system image. The user can define the network topology using a simple YAML configuration file. The virtual connections between containers in containerlab, simulate real-world network connections. Containerlab starts the containers and initializes the network devices, allowing SSH or similar access.

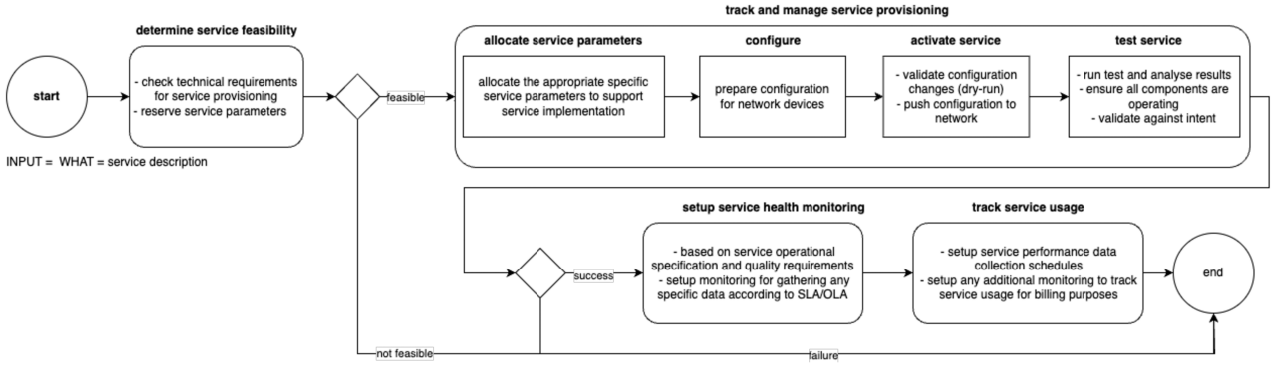

Figure 3 – The service provisioning workflow used for the PIONIER network

The whole workflow presented in Figure 3 can be activated programmatically or via a separately developed web interface that is used to collect the input parameters necessary to build the minimal JSON service description. At the end of this workflow the complete JSON service description for the L2 circuit is available in Maat crosslinked using references with all of its subelements such as service access points and related devices and interfaces as represented in Figure 4. In short, the L2 circuit instance points to two Service Access Points (SAPs) that represent the two ends of the circuit. Each of the SAPs also contains a relationship with the interface which maintains a resource relationship with the router it resides on. Each instance, no matter if service or resource type, has a specific set of parameters in the form of a list of characteristics relevant for that instance. It is worth mentioning that the L2 circuit data model follows the logic of service configuration in Juniper routers, but is fully device agnostic.

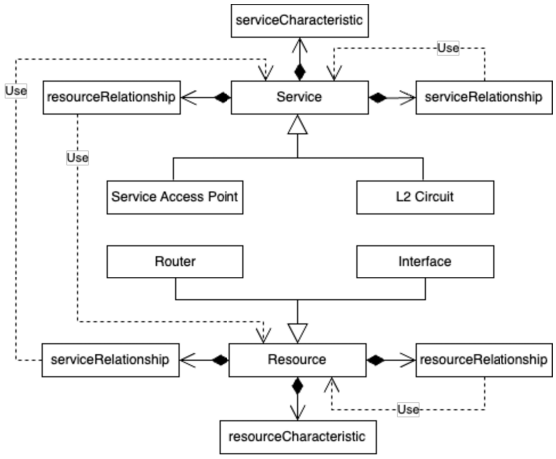

Figure 4 – Simplified UML representation of the L2 service data model used for the PIONIER network

A crucial step in the workflow that directly influences device configurations whether in a test topology within containerlab [10] or in the target production infrastructure — is the 'activate service' task. In this pilot implementation, Ansible playbooks are used to automate the configuration

of L2 circuit provisioning. To activate the service in the network, a task running in Airflow needs to activate playbooks in Ansible which resides on another host. However, Ansible does not natively support running playbooks from a remote host. Thus, we use an additional component, the GÉANT Lightweight Service Orchestrator (LSO) [11] application residing on the same host with Ansible. The LSO exposes a REST API allowing the Ansible playbooks to be executed by external workflow requests. All parameters required for Ansible to configure the network resources and services correctly are passed from the workflow through the LSO REST API.

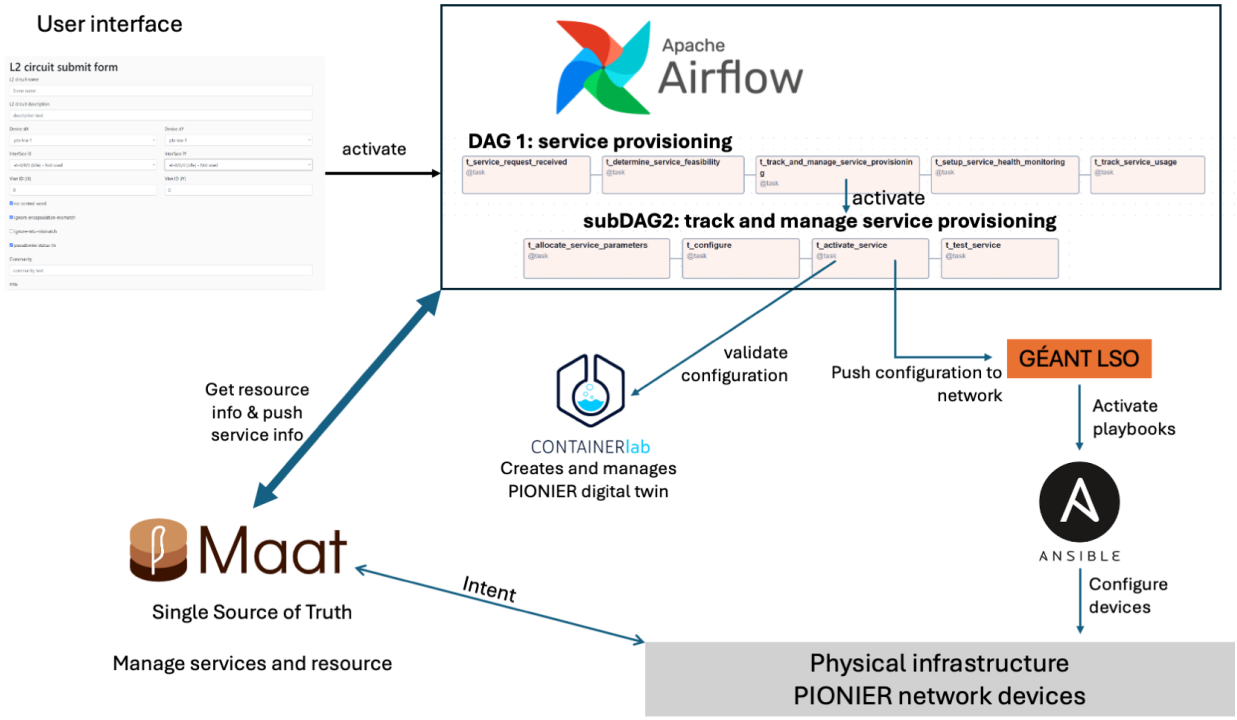

Figure 5 – Implementing L2 circuit service provisioning in the PIONIER network

Putting it all together, in Figure 5 we graphically present the implemented service orchestration processes for the L2 circuit provisioning in the PIONIER network. Once the network engineer completes the L2 circuit submit form with the necessary information, the main service provisioning workflow described in Figure 3 is activated in Airflow. In one of the tasks, it calls the sub-process 2 that is in charge of tracking and managing the service provisioning. This sub-process will allocate all service parameters, decide on the required configuration changes and then move into the activation task where it will first validate the new configuration in the digital twin created using containerlab, then, if everything is ok, attempt to push the configuration to the network by calling LSO. The LSO component will in turn process the request for new service implementation in the network and activate the corresponding playbooks in Ansible, which will configure the devices based on the tasks described in the playbooks. During this whole process, Maat is queried whenever information about the network resources or other relevant entities is needed, and a new service instance is created according to the specifications of the user representing our intent for the future network behaviour. Once the service is fully activated on the network devices, this new instance in Maat will have the active status.

To ensure the effectiveness of our generalised network service orchestration solution, we use continual evaluation of the implementation progress by the PIONIER network engineers that focuses on different aspects during their user acceptance testing such as functionality, performance, scalability, and reusability.

Some key lessons learned include the importance of adhering to standardised approaches and data models as crucial for ensuring interoperability and promoting reusability. Continuous improvement also emerged as essential, with ongoing evaluation and refinement needed to adapt the solution to evolving requirements and emerging challenges. Furthermore, collaboration proved valuable, as working closely with PIONIER network engineers combined with the experience from other NRENs that have representatives in the GP4L team fosters the sharing of knowledge, best practices, and resources, ultimately benefiting the entire community.

Once the pilot is fully completed, its detailed description together with reuse guide, links to the code repository and demo videos will be made available on the GP4L website [12]. In addition, to foster the concept of reusability, most of the components used in the orchestration process are available in the GÉANT nmaas [13] application catalogue. This provides the opportunity for interested parties to discover the full set of functionalities of Maat and/or Airflow in a containerised playground offered by nmaas.

4 Conclusion and future work

In this paper, we presented a generalised approach to network service orchestration that leverages open-source tools and a reusable framework. Our solution focuses on providing a flexible and efficient way to automate the provisioning and lifecycle management of network services. By adopting this solution, network service providers can improve operational efficiency, reduce costs, and enhance service quality.

Looking ahead, the GP4L team plans to explore new use cases by modelling and implementing additional services for the PIONIER network, integrate with emerging technologies, and collaborate with the wider network community to promote the adoption of best practices in network automation and orchestration.

Bibliography

[1] Vial G. *Understanding digital transformation: A review and a research agenda. Managing digital transformation*, 26:13-66. May 2021.

[2] Morris K. *Infrastructure as code*. O'Reilly Media; December 2020.

[3] Shah J. A., Dubaria D. ‘Netdevops: A new era towards networking & devops’, *IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference*, pp. 775-779 October 2019.

[4] De Sousa N. F., Perez D. A., Rosa R. V., Santos M. A., Rothenberg C.E. *Network service orchestration: A survey.* Computer Communications, 142:69-94. June 2019.

[5] TM Forum. *Business Process Framework (eTOM) Suite* v23.5, December 2023; https://www.tmforum.org/resources/suite/gb921-business-process-framework-etom-suite-v23-5/.

[6] Haines S. 'Workflow orchestration with Apache Airflow', *Modern Data Engineering with Apache Spark: A Hands-On Guide for Building Mission-Critical Streaming Applications*, pp. 255-295, Berkeley, CA: Apress, March 2022.

[7] Hochstein L., Moser R. *Ansible: Up and Running: Automating configuration management and deployment the easy way.* O'Reilly Media, Inc. July 2017.

[8] GN5-1 WP6T2. *Maat: Source of truth for automation and orchestration*. August 2024; https://geant-netdev.gitlab-pages.pcss.pl/MaatDocs/.

[9] Almasan P., Ferriol-Galmés M., Paillisse J., Suárez-Varela J., Perino D., López D., Perales A. A., Harvey P., Ciavaglia L., Wong L., Ram V. 'Network digital twin: Context, enabling technologies, and opportunities.', *IEEE Communications Magazine*, 60(11):22-27. June 2022.

[10] SRL labs, *containerlab*, September 2024; https://containerlab.dev/

[11] GÉANT Orchestration and Automation Team, *LSO*, September 2024; https://gitlab.software.geant.org/goat/gap/lso.

[12] GN5-1 WP6T2. *GP4L documentation website*, September 2024; https://geant-netdev.gitlab-pages.pcss.pl/gp4ldocs/.

[13] GN5-1 WP6T2. GÉANT nmaas service, October 2024, https://nmaas.eu/.