Automated Deployment and Configuration of Key Management Systems in QKD Networks

Tommy van Duijn, Simon Rommel
Department of Electrical Engineering
Eindhoven University of Technology
Eindhoven, Netherlands
{t.c.j.v.duijn,s.rommel}@tue.nl

***Abstract*—Quantum key distribution (QKD) enables information-theoretically secure key exchange, but its practical deployment at scale depends on secure and reliable key management systems (KMSs). As QKD networks grow in size and complexity, manual configuration and operation of KMSs could become a major source of operational overhead and risk of misconfiguration. We demonstrate the automated deployment and configuration of KMSs for QKD networks using a workflow orchestrator, achieving successful deployment and full configuration of KMSs and key acquisition and delivery services within $4\,\mathrm{min}$.**

***Index Terms*—quantum key distribution, key management system, workflow orchestration**

I. Introduction

The development of quantum computers poses a serious threat to the security of widely-used key-exchange algorithms due to Shor's algorithm [1]. Quantum key distribution (QKD) enables information-theoretically secure key exchange and offers protection against present and future advances in computing and cryptanalysis [2]. However, the practical deployment of QKD networks presents significant challenges, particularly in scaling beyond point-to-point links towards operational, multi-node infrastructures [3].

In the absence of quantum repeaters, key relaying with trusted nodes is the primary solution to overcome fundamental rate-distance limitations and enable QKD to move from point-to-point links to a full network [4], [5]. A crucial component in this ecosystem is the Key Management System (KMS), which is responsible for key relaying and must ensure that keys are securely delivered to cryptographic applications anywhere in the network [6].

Currently, deploying and configuring KMS instances within a QKD network may require substantial manual intervention, involving complex configuration of interfaces, policies, and inter-system dependencies. This includes (but is not limited to) creating and installing certificates, establishing links between KMSs, and registering cryptographic applications. The standardization of QKD network orchestration is still ongoing, and the exact procedures and terminology differ per QKD vendor, but the process is error-prone and time-consuming. As QKD networks evolve toward real-world deployment, the need for automation in KMS provisioning and QKD network orchestration becomes increasingly important [7].

In this paper, we present workflows for the automated deployment and configuration of our in-house developed KMS for QKD networks. The workflows are implemented using an existing workflow orchestration framework. The main contribution of this work is the design and implementation of orchestration workflows that enable automated provisioning, configuration, and lifecycle management of containerized KMS instances using established cloud-native technologies. We show and discuss the results of systematic benchmarks of the designed and implemented workflows.

II. Related work

Early demonstrations of QKD networks, such as SECOQC, were largely based on ad-hoc experimental testbeds [8]. These efforts focused primarily on the feasibility and integration of the physical-layer, with limited attention to operational scalability or automated service provisioning and management. More recent work focused on the integration of QKD systems into existing classical telecommunications infrastructures [9], [10]. In this context, interoperability with optical transport networks, network management systems, and service platforms has become important.

Since software-defined networking (SDN) has been widely adopted in telecom networks, the application of software-defined networking principles to QKD networks and its integration with existing SDN controllers are also being investigated and standardized [7]. SDN techniques have been demonstrated in QKD networks for service provisioning [11], relay path computation [7], and quality assurance [12]. The automated provisioning of KMSs is not addressed by these or similar works.

In parallel, classical telecom and cloud infrastructures have increasingly adopted automated provisioning and orchestration frameworks to manage complex service lifecycles, including virtual network functions and security services [13]. Tools such as Kubernetes, Ansible, and network orchestrators are commonly used to manage containerized network functions and microservices in cloud-native architectures. Despite the existence of these technologies and the introduction of SDN in

This work was supported by the the Dutch Ministry of Economic Affairs and Climate Policy (EZK) as part of the Quantum Delta NL programme and the EU's Digital Europe Programme under project QCINed (grant no. 101091656) within the EuroQCI initiative.

978-3-903176-78-2

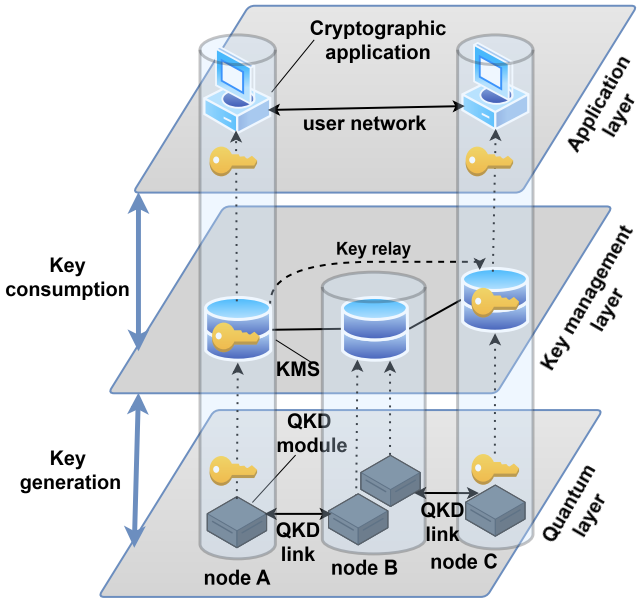

Fig. 1: Model of QKD network with trusted nodes. QKD keys are established between adjacent nodes (connected by QKD link), on the quantum layer. Keys can be exchanged between non-adjacent nodes through key relaying on the key management layer. The keys are used by cryptographic applications to encrypt data in the user network. The cryptographic application in the middle node (B) is omitted for legibility.

QKD networks, the automated provisioning of KMSs remains an open challenge.

Pedone et al. described a software stack for QKD networks [14]. While the proposed software stack demonstrates the feasibility of integrating QKD services in containerized environments, the evaluation is limited to simulated deployments and does not address multi-hop key relaying through trusted nodes. In this paper, we show that widely-used technologies can be used in practice to automate the provisioning, configuration, and lifecycle management of containerized KMS instances for QKD networks via workflow orchestration, Docker and REST APIs compliant with ETSI standards.

III. QKD Network Model and KMS

A. QKD network model

Figure 1 shows the QKD network model based on the ITU-T recommendations [15]. The model consists of three layers: the quantum layer, key management layer, and application layer. On the quantum layer, secret bytes are established between adjacent nodes. Pairs of QKD modules connected by a QKD link continuously run a QKD protocol and deliver the output to the key management layer, where the secret bytes are stored and managed by the KMS. These bytes are either delivered to cryptographic applications on the application layer when requested, or used to perform key relaying on the key management layer, enabling secure key exchange between non-adjacent nodes. Cryptographic applications use QKD-established keys to encrypt data in the user network.

B. KMS architecture

Each KMS must have a high-level abstract overview of the QKD modules within its node and certificates to retrieve keys from each of them. Every KMS periodically retrieves keys from QKD modules within its node using the ETSI GS QKD 014 API. In each pair of adjacent nodes, one KMS takes initiative to retrieve keys from the appropriate QKD module within its node and sends the IDs of the retrieved keys to the other KMS. The responder requests the corresponding keys from the QKD module within its own node upon receiving the key IDs. The key IDs are exchanged over a classical channel called “key acquisition link”.

The KMS securely buffers keys in memory for later use, maintaining independent key pools per remote QKD node. Bits from key pools are used to serve key requests from authorized cryptographic applications or to perform hop-by-hop key relaying as described in ITU-T Y.3803 [15]. By abstracting away from QKD hardware, the KMS can facilitate vendor-agnostic key relaying across heterogeneous QKD links.

Key pools can also be established between non-adjacent nodes through proactive key relaying, reducing the latency experienced by cryptographic applications while exchanging keys between non-adjacent nodes [5]. Therefore, there may be a logical key association between any pair of nodes, regardless of whether they are connected by a QKD link or not. Key relaying and key pool synchronization are performed over a classical channel called “key management link” [16]. Although the key management link and the key acquisition link are distinct logical channels, in practice, all communication between two KMS instances can be implemented over the same physical link. We use Envoy as a forward and reverse proxy to secure communications on behalf of the KMS.

The KMS delivers keys to authorized applications through the ETSI GS QKD 014 interface. When a KMS receives a key request, it must identify the remote KMS associated with the target application specified in the key request. In case the remote KMS is not a direct neighbor, the KMS must be able to find a suitable relay path. To ensure that every KMS can select the appropriate endpoint for key delivery, it is assumed that each application is registered on a specific node. The KMS supports both static routing and forwarding routing requests to an external router to find suitable relay paths [17].

The KMS can be configured through a REST API based on ETSI GS QKD 015.

IV. Workflow Orchestrator Application Architecture

We implemented our provisioning workflows within the framework of an existing open-source workflow orchestrator (WFO) that provides both a workflow engine and a graphical user interface [18]. It is designed to execute sequences of automated operations in a structured manner. At its foundation is the orchestrator-core component, a Python-based backend built on the FastAPI web framework and Pydantic for data modeling and validation. The core defines the ruleset for product modeling and workflows.

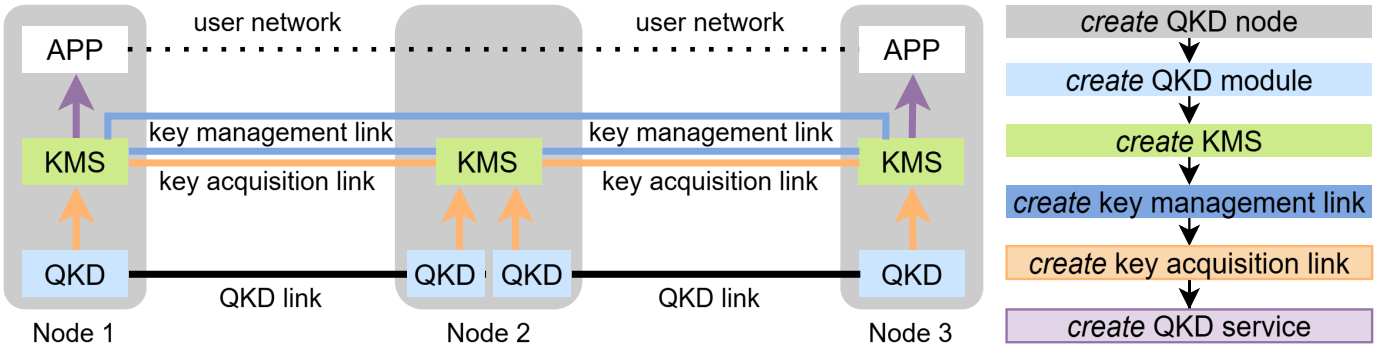

Fig. 2: Mapping of QKD network elements (left) to workflows for their deployment and configuration (right). The color of each element corresponds with the associated workflow. The "APP" in the middle node is omitted for legibility.

In the context of this paper, a workflow is a list of ordered steps, each corresponding to a Python function that performs a specific task in the provisioning or configuration process. The WFO executes these steps in sequence, stores intermediate states in its database (core-db), and tracks the lifecycle of each workflow instance. The steps are designed to be atomic, which means that no changes are committed to the database if a step fails. Workflows should ensure that any changes made to external systems are rolled back in case a step fails. Failed steps can be retried, starting from the state of the last successful step.

Some steps may require input information. The WFO uses NetBox as the source of truth, which means that certain information required to execute workflows, such as IP addresses, is stored in and retrieved from NetBox. If input is required for a workflow step, the WFO loads possible options from NetBox and lets the user/administrator select the appropriate option in the GUI. Such input forms can be automatically generated by a workflow. The WFO also writes data to NetBox, representing logical connections resulting from executed workflows.

Although NetBox is used as much as possible for information storage, it is not suitable for all kinds of information, and information that is specific to the WFO is stored in the core-db. In addition to the states of workflows, the core-database contains products that represent the services offered and managed by the WFO. Workflows can create, modify, and terminate product instances, so-called subscriptions, that belong to a specific customer.

We modeled the KMS, logical links, and services as products and added these products to the WFO. We added *create* workflows to deploy KMSs and configure key acquisition and delivery services, and *delete* workflows to terminate KMS instances and services; corresponding *modify* workflows allow modification/update of existing subscriptions.

V. Workflow Descriptions

The workflows and the order of their execution for the deployment of a QKD network are shown in Fig. 2 and explained below.

QKD node

create: Generates a random node ID (UUID) and adds it to a custom field of the selected site in NetBox. This custom field is used to logically group devices in NetBox. The generated ID is also used as the unique identity of the KMS in this node. This unique identity is important for KMS-to-KMS communication, authentication, and relay path selection.

delete: Removes the selected QKD node ID from NetBox. Before executing this workflow, all subscriptions related to this node must be terminated.

QKD module

create: Logically adds a QKD module to a QKD node by writing the selected node ID to a custom field of the selected QKD module in NetBox. This marks from which QKD modules KMSs can retrieve keys. The QKD module must be located on the same location/site as the QKD node.

delete: Logically removes the selected QKD module from the node by resetting the custom field of the device in Netbox.

KMS

create: First generates private keys and certificate sign requests (CSRs) on the target host on which the KMS will be deployed. After generating the CSRs, the WFO copies them and prompts the administrator to sign the corresponding certificates. The signed certificates are then provided as input for the next step, during which the WFO installs them on the target host. If an automated public key infrastructure (PKI) solution (e.g., Dogtag or similar) is available, this step can be fully automated. Note that the time for certificate signing is excluded from the performance measurements, time for installing the signed certificates is included. Next, two Docker containers are deployed on the target host using the Python Docker API. One container is for the Envoy proxy and the other container is for the KMS itself. The proxy has interfaces for key requests, KMS-to-KMS communication, management, and monitoring. The port number must be specified for each interface. The IP address is loaded from NetBox (must be manually selected if KMS has multiple IP addresses). The WFO connects to the target host using SSH. It is assumed that SSH has been configured so that the WFO has access to the target host without manual intervention.

delete: Terminates a KMS instance by stopping and removing

the Docker containers using the Docker API. All links and services related to this KMS must be terminated before executing this workflow.

modify: Redeploys the Docker containers using the Docker API. This allows for changing the IP address and/or port numbers, renewal of certificates, and software updates. In case IP addresses or port numbers are changed, existing key management links have to be updated.

key management link

create: Connects two KMS instances by creating a key management link in each instance using the KMS REST API. The type of this link can be either "direct" for links between adjacent nodes or "virtual" for links between non-adjacent nodes. In both cases, this workflow creates a logical key association and allows key relaying between the two KMS instances. The required IP addresses, port numbers, and IDs are obtained from NetBox.

delete: Deletes the selected key management link from a pair of KMSs using the REST API. After this workflow has been executed, the pair of KMSs can no longer communicate with each other and the shared key pool is erased. Key acquisition links and QKD services related to this key management link must be terminated before executing this workflow.

key acquisition link

create: Creates a key acquisition link between a pair of KMSs and informs each KMS about the appropriate QKD module within its node using the REST API. The IP address and port number of the ETSI014 key interface on the QKD module are loaded from NetBox. The Secure Application Entity (SAE) IDs and file names of the client certificates for the communication between the KMS and QKD module must be specified manually. Since the manner of specifying the SAE ID in the client certificate is not yet standardized and may differ per QKD vendor, it is assumed that the required certificates are created manually. The certificates and corresponding private keys are also manually installed to avoid that the WFO must copy private keys over the network. The proxy configuration is automatically updated to enable communication with the appropriate QKD module. After creating the link using the KMS REST API, the KMS schedules a job to periodically retrieve keys from the appropriate QKD module.

delete: Deletes the selected key acquisition link from a pair of KMSs using the REST API. After this workflow has been executed, the pair of KMSs will no longer acquire keys from the related QKD modules. Bits remaining in existing key pools may still be used.

modify: Updates the selected key acquisition link using the KMS REST API. The IP addresses, port numbers, certificate file names, and SAE IDs for key acquisition may be updated.

QKD service

create: Registers a pair of SAE IDs in the selected KMSs in each KMS using the REST API. This allows the associated cryptographic applications to request keys from the KMS where they are registered. The registered SAE ID must match the ID in the certificate presented by the client. A relay path (route) must be specified in case the applications are registered on non-adjacent nodes.

delete: Deletes the selected service from a pair of KMSs using the REST API. After this workflow has been executed, the pair of applications can no longer exchange keys.

modify: Updates the selected QKD service using the KMS REST API. The SAE IDs and/or relay path associated with this QKD service may be updated.

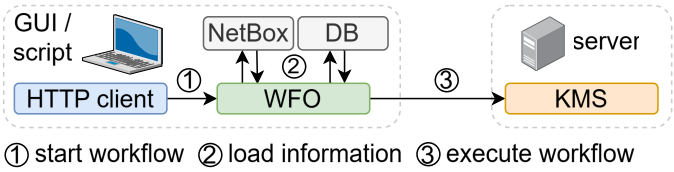

Fig. 3: Experimental setup. An HTTP request triggers workflows. The workflow orchestrator loads required information from NetBox and interacts with a remote server to deploy/configure the KMS.

TABLE I: Hardware specifications of the laptop running the orchestrator during the experiments

Component	Specification
Processor (CPU)	Intel Core i7-11800H @ 2.30GHz
Memory (RAM)	16 GB DDR4-3200
Storage	NVMe SSD 1 TB
Operating System	Windows 11 24H2

VI. Experimental Setup

To assess the performance of our workflows and automation process, we deployed the WFO on a laptop and executed every workflow multiple times, as illustrated in Fig. 3. We deployed all Docker containers required to run the WFO, i.e., the WFO core, NetBox, and database, on the same machine. Table I shows the hardware specification of this machine.

Two Raspberry Pi 4B servers were used to run the KMSs deployed by the WFO. We deployed QKD modules in the field and manually created certificates for them prior to the experiments [10]. We also installed servers, manually entered the IP addresses of the servers and QKD modules into NetBox, and configured SSH access to the servers.

We used the WFO to automatically deploy and configure KMS instances, and measured the duration of each workflow, including container startup time and network latency. We used a Bash script to automatically start workflows through HTTP requests, eliminating fluctuations in duration of workflows arising from the GUI and manual interventions. Each *create* workflow was followed by a *delete* workflow to return the system to the previous state. This process was repeated ten times per workflow. We used `curl` to measure how long it took the orchestrator to complete each workflow.

VII. Results and Discussion

Table II presents the duration of each workflow. The first two workflows, which interact only with NetBox deployed on the same machine as the WFO, complete in less than 1 s, while

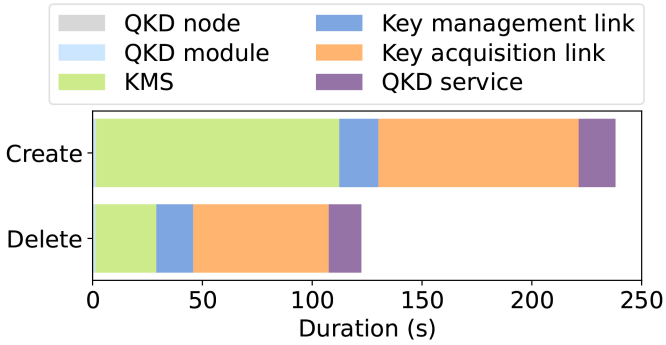

Fig. 4: Average duration of workflows.

TABLE II: Measured durations of *create* and *delete* workflows (mean and std dev).

Workflow Name	***create* (s)**	***delete* (s)**
QKD node	0.64 ± 0.05	0.54 ± 0.08
QKD module	0.88 ± 0.14	0.76 ± 0.32
KMS	110.79 ± 1.01	27.70 ± 0.71
key management link	17.82 ± 2.01	16.83 ± 0.79
key acquisition link	91.19 ± 2.02	61.68 ± 0.17
QKD service	16.79 ± 1.47	14.89 ± 0.31

workflows involving remote devices take significantly longer, up to nearly $2\,\mathrm{min}$.

Although certain workflows take some time to complete, independent workflows can be executed in parallel and user input for subsequent workflows can be prepared during execution. Additionally, workflows may be triggered automatically (e.g., via scripts), further reducing operational overhead.

The total average duration of all *create* workflows is $238.11\,\mathrm{s}$, as shown in Fig. 4, indicating that KMS instances and services can be deployed within $4\,\mathrm{min}$, assuming pre-installed hardware and available input parameters. The measured deployment time includes container startup time and network latency; time for manual steps is not included.

Certain steps, such as certificate signing and QKD module configuration, are currently performed manually, which may affect efficiency and reproducibility. While KMS certificate management could be automated using existing PKI solutions (e.g., Dogtag), automated configuration of QKD modules (e.g., through SDN agents) is currently not supported by the commercial QKD modules used in our testbed.

The experimental setup represents a lab-scale deployment, with the orchestrator running on a single workstation and several components co-located. This implementation should be regarded as an initial prototype that demonstrates the feasibility of automated provisioning, configuration, and lifecycle management of containerized KMS instances using established cloud-native technologies. The evaluation in this paper focuses primarily on execution time as a performance indicator; future work could consider additional evaluation metrics, including scalability and system load, as well as distributed deployments, automated certificate management, and concurrent workflow execution.

VIII. Conclusions

We developed workflows for the automatic deployment and configuration of key management systems in QKD networks with trusted relay nodes. We demonstrated that we can automatically deploy and configure KMS instances, and key acquisition and delivery services, in $238\,\mathrm{s}$, given that the hardware is already installed and the required input information is readily available. Automation of this error-prone and time-consuming process is essential to enable and optimize successful real-world deployment of QKD networks.

References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484-1509, 1997.

[2] C. H. Bennett and G. Brassard, "Quantum Cryptography: Public Key Distribution and Coin Tossing," *Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing*, vol. 1, pp. 175-179, 1984.

[3] C. Elliott, "Building the quantum network," *New J. Phys.*, vol. 4, pp. 46.1-46.12, 2002.

[4] M. Mehic *et al.*, "Quantum Key Distribution: A Networking Perspective," *ACM Comput. Surv.*, vol. 53, no. 5, pp. 1-41, 2020.

[5] C. Stan, D. Verchere, J. J. Vegas Olmos, I. Tafur Monroy, and S. Rommel, "Dynamic-threshold-based pre-relaying for enhanced key allocation in quantum-secured networks," *J. Opt. Commun. Netw. 17(3)*, pp. 233-248, 2025.

[6] P. James, S. Laschet, S. Ramacher, and L. Torresetti, "Key Management Systems for Large-Scale Quantum Key Distribution Networks," *Proceedings of the 18th International Conference on Availability, Reliability and Security*, 2023.

[7] P. P. Tavares, A. González, J. Cunha and A. Costa, "6G QKD Network Controller for TeraflowSDN," *15th International Conference on Network of the Future*, pp. 81-85, 2024.

[8] M Peev *et al.*, "The SECOQC quantum key distribution network in Vienna," *New Journal of Physics*, vol. 11, no. 7, 2009.

[9] Martin *et al.*, "MadQCI: a heterogeneous and scalable SDN-QKD network deployed in production facilities," *npj Quantum Information*, vol. 10, no. 1, 2024.

[10] T. van Duijn, C. Stan, M. Alvarez Roa, I. Tafur Monroy, S. Rommel, "Quantum Key Distribution in the Field: Deployment of the Eindhoven Testbed," in *IEEE Photonics Conference*, 2025.

[11] Y. Cao *et al.*, "Experimental Demonstration of End-to-End Key on Demand Service Provisioning Over Quantum Key Distribution Networks with Software Defined Networking," in *Optical Fiber Communications Conference and Exhibition (OFC)*, 2019, pp. 1-3.

[12] Q. Zhu, X. Yu, Z. Wang, Y. Zhao and J. Zhang, "Demonstration of Autonomic End-to-End QoS Assurance over SDN-based QKD-Secured Optical Networks," *Opto-Electronics and Communications Conference (OECC)*, 2023, pp. 1-4.

[13] A. Aguado *et al.*, "Secure NFV Orchestration Over an SDN-Controlled Optical Network With Time-Shared Quantum Key Distribution Resources," *Journal of Lightwave Technology*, vol. 35, no. 8, pp. 1357-1362, 2017.

[14] I. Pedone, A. Atzeni, D. Canavese, and A. Lioy, "Toward a Complete Software Stack to Integrate Quantum Key Distribution in a Cloud Environment," *IEEE Access*, vol. 9, 2021.

[15] ITU-T Recommendation Y.3803, "Key management for quantum key distribution networks," 2020.

[16] T. van Duijn, S. R. Verschoor, and S. Rommel, "Synchronization Protocol for Quantum Key Distribution Networks," *Advanced Photonics Congress*, 2025.

[17] T. van Duijn, S. R. Verschoor, S. Rommel, I. Tafur Monroy, "Routing Strategies for Quantum Key Distribution Networks Based on Trusted Relay Nodes," in *28th International Conference on Optical Network Design and Modelling*, 2024.

[18] SURF, "Workflow Orchestrator," workfloworchestrator.org. Available: https://workfloworchestrator.org/orchestrator-core/. Accessed: 2 Oct. 2025.