



Secure and Extensible Platform Architecture for Enterprise Network Orchestration

Manevannan Ramasamy

Software Engineering Senior Technical Leader, Milpitas, California, United States of America

manevannan3@gmail.com

ABSTRACT: There is a growing demand to have enterprise network orchestration platforms to be extensible in the sense that they can be able to integrate different services, automation tools, external systems and independently developed capabilities without affecting the architectural stability and operational control. The paper presents a safe and scalable platform architecture that puts emphasis on controlled extension plane which isolates orchestration invariants of the core and optional capabilities. Typical used architecture is versioned capability contracts, service catalog, event-based subscriptions, isolated execution environment, delegated authorization, resource quotas, conformance testing and governance of lifecycle. It can engage in the orchestration ecosystem by having explicit ownership, compatibility requirements, data classes, resource budgets and failure-handling policies. Security is integrated during the registration, discovery, execution, upgrade, monitoring and retirement of extensions as opposed to being considered only as a runtime issue. The framework also provides compatibility management, policy enforcement, observability and faulty or nonconforming extensions recovery mechanisms. Framework assessment includes effort on integration, preservation of compatibility, isolation of execution, enforcement of authorization, observability of operations, and effectiveness of recovery. Work offered architecture not only offers independent teams and external systems to develop capabilities, but also offers standardized interfaces and governance boundaries and minimizes dependency on internal data stores and release processes. The resulting model is able to maintain platform development particularly in regards to orchestration core that is maintained, audited, and governed by policy and remains stable platforms.

KEYWORDS: platform architecture, secure extensibility, network orchestration, capability contracts, API governance, event-driven integration, lifecycle management

I. INTRODUCTION

Enterprise networks are being transformed into a more distributed, programmable and service-based systems, comprising data centers, cloud, branch, and environments, integrating software-defined infrastructure and heterogeneous network devices. These types of environments are hard to operate using manually coordinated configuration processes that become complicated to scale, achieve consistency, have observability, and react to operational changes. To solve these problems, the modern orchestration platforms open up programmable interfaces through which automation services and applications can learn about the network resources, provision infrastructure, learn about operational state and orchestrate network services. Enterprise-controllers such as Cisco Catalyst Center, which provide API access to network-management features, are a pointer to the growing popularity of programmable orchestration interfaces in large-scale network environments [1]. But being able to expose APIs is not a full-fledged architectural solution to integrate the capabilities developed independently. The potential extensions may require different release patterns, authorization, access patterns of data, and/or computational facilities and policies of operation which attract significant architectural and security concerns.

The rise of external services, fragments of automation and analytics or artificial intelligence applications, context-specific to the organization all contribute to the need to have a scalable orchestration platform. Special monitoring, policy analysis, configuration validation, event processing, compliance assessment, optimization, or automated remediation may be performed by an extension. By wrapping these capabilities next to the orchestration core, dependencies complexity can be increased by any extensions, as well as release coordination requirements and failure impact. Conversely, the infinite access to extensions may disclose sensitive network state, privileged APIs, credentials and infrastructure activity to components that may not be entitled to such privileges. Due to this architecture, extensibility ought to be an architectural feature that is controlled, with flexibility of integration compromised to security, isolation, compatibility and governance.



Security requirements are of particular significance when the orchestration platforms cut across organizational and infrastructure boundaries. The concept of zero-trust architecture is that there should be no implicit trust in access based on network location and ongoing authorization should be judged based on explicit policies [2]. This philosophy can be directly applied to extensible orchestration since an extension ought to be viewed as having precisely the capabilities it needs to perform its designated purpose. Rather than offering full-scale access to the orchestration platform, it is possible to grant access with narrow scopes, capabilities, or context policies and resource restrictions. This can minimize potential danger of having improperly set up, bugged, or unsecured extensions whose crashing or hacking can have no likely effects on the broader platform backbone, and makes the compatibility controlled with the core.

Secure extensibility is another valuable basis of workload identity. The distributed orchestration platform is likely to be executing services in containers, virtual machines, clusters and clouds and familiar identity mechanisms are not available to provide consistent identification of different workloads. Through SPIFFE, identities are assigned to workloads in a standardized way and there is a system which establishes trust between distributed services[3]. Within an extension architecture, extension instances may be given definable identities by the identity of the workloads, and concrete ownership context. This identity can then be used by authorization policies, audit mechanism, service-to-service communication controls and lifecycle management processes. Consequently, the actual workload can be linked to extension access as opposed to making use of network location or the use of fixed credentials only.

The level of security of the software artifact is also dependent on the integrity of the introduced software artifact into the orchestration environment. New software is usually dependent on third-party software packages, container images, build pipelines and other third party aspects, that create supply-chain risks with potential to leak into running platforms. SLSA provides a paradigm to improve the integrity of software supply chains, which have mechanisms associated with artifact provenance, build procedures and incremental security measures [4]. Using these principles in extensions of orchestration allows platforms to check the provenance of extension artifacts prior to deployment. Registration by extension can thus include provenance, version number, ownership, compatibility specification, and integrity witnessing instead of considering deployment as a mere install procedure.

Operational observability is important, as well, because a secure extensibility implies that the platform can learn the behavior of extensions after its deployment. Distributed extensions have the capability of producing asynchronous events, calling orchestration APIs, consuming computing resources and communicating with various services. Without standardized telemetry, administrators might not be able to tell whether an extension is operating as intended, it is using too much resources, has an unusual behavior or is part of an operational incident. OpenTelemetry provides a standard form of system-collected, and cross-system correlating systems of telemetry such as traces, metrics, and logs [5]. This observability, made an integral part of the plane of extensions, can guarantee coherent operation evidence to monitor, audit, troubleshoot, implement policies and make lifecycle decisions.

Even though such single technologies and architectural ideas are applicable the gap that remains in applying the technologies and architectural ideas to network-orchestration extensibility in business is big. The current practices are inclined to address API integration, authorization, workload identity, software integrity, or observability individually. These issues need to work in harmony in a coherent lifecycle in an extensible orchestration platform. The registration of extensions should be through explicit contracts, located in a controlled catalog, verified to be compatible, granted with extensive capabilities, executed with no more than an appropriate level of isolation, monitored with standardized telemetry, and be retired through managed lifecycle mechanisms.

This paper hence suggests a stable and scalable platform design that revolves around an exclusive extension plane. The architecture contains cores orchestrations invariants and optional capabilities and connects all extensions with clearly owned ownership, compatibility of versions, data classification, resource constraints, authority environments, and failure policies. The model under consideration also aims at providing the capabilities of the autonomous teams and external systems to add new orchestration without the need to be directly linked to internal databases and have an unlimited access to the control core. The architecture is expected to offer coherent and auditable underpinning on which network-orchestration ecosystems in businesses develop, via managed extensibility, and identity, authorization, artifact integrity, isolation, observability, and lifecycle management.



II. RELATED WORK

Closely connected management systems have evolved into programmable, service-oriented, intent-driven and increasingly policy-regulatory architectures, which are now commonly known as enterprise network orchestration. Modern systems have made APIs and automation interfaces available so that external applications can control devices, access information on the functioning of the network and bind network services. Cisco Catalyst Center API offers a representative enterprise information orchestration interface which applications can programmatically communicate with network-management offerings [1]. These interfaces give a meaningful foundation on extensibility even though API availability is not a resolution to the proprietorship of extensions, versioning extensions in capabilities, execution isolation, delegated authorization, resource quotas or lifecycle management. These limitations push architectural designs in which extensibility is a progressively-managed platform attribute, but not just a simple API access.

Security-oriented architecture research mechanisms are others that can be used in extensible orchestration. Zero-trust architecture is based on the principles of authorization that is always present, formally defined policies, access in the least possible privilege, and protection of resources irrespective of its location on the network [2]. These rules can especially be applied to extensions developed by third parties or independently and those that can communicate with orchestration services. The identity specifications, adopted by the workloads such as SPIFFE provide identity assignment of workloads and identity-based trust of distributed spaces [3]. Similarly, SLSA resolves software supply-chain integrity issues with provenance and stepwise security, which provide backgrounds that can be utilized in reassessing the provenance and integrity of extension artifacts before deployment [4]. The distributed services and automation workflows observability [5] is offered by OpenTelemetry which offers the standardised trace, metrics, and logs collection mechanisms. Guidelines on microservice security also emphasize APIs protection, identity of services, segmentation, secure communication and centralized policies to compute distributed application architectures [6]. All these help in the definition of important security and operational primitives, but is not a lifecycle model specific to network-orchestration extensions.

The task of automation has also been enlarged with the assistance of the study on the automation of zero-touch networking that connects the sphere of service checking and the activity of autonomy. The end-to-end service monitoring provides tools, which check network services and repair automated operations choices [7]. This type of monitoring is particularly pertinent to an extensible orchestration platform since extensions created independently need to have observable performance of status of execution, failures, usage of resources, and service-level performance. Monitoring mechanisms generally consider the operational visibility rather than considering the lifecycle of registration, discovery, compatibility validation, controlled execution, upgrading, and retirement holistically.

Another significant architectural direction is intent-based networking. Intent methods map operational objectives on higher levels into network policies and automated operations without the need of relying on manual configurations [8], [9]. Under intent-based slicing the evolution of the policies whereby the resources across the different areas of network can be managed and service administration could be automated could be retained [8]. The conceptual relationship between the intent to business or operation, policies translation, a current network state, and automatic enforcement, is set within the broader intent-based networking studies [9]. These approaches apply to orchestration platforms because extensions can provide custom intent-processing, policy-analysis, assurance or remediation to assist. Nonetheless, intent-based architectures concentrate more on mapping intent to network behavior, rather than putting distinctly developed extensions under the scrutiny of a formalised platform element, be it controlled or versioned.

Heterogeneous environment and distributed resource management is increasing in popularity in recent studies on orchestration. The concept of multiobjective network-slicing approaches tackles these multiple targets of achieving simultaneous optimisation of several goals on space-air-ground integrated networks, and shows the significance of a coordinated allocation of resources as well as distributed decision processes [10]. Similarly, smart computation-offloading methodologies employ smart learning-based structures to detect dynamically the resource that it is succeeded by in the dynamically varied network conditions [11]. The earlier studies of the sphere of AI-based edge intelligence also provide the examples of how the distributed intelligence can be integrated with network resources as well as new kinds of service architectures [12]. The realization of next-generation network machine-learning studies also emphasizes the need to adaptively harness the use of resources and service-provisioning automation in the face of the challenging performance requirements [13]. These studies prove the worth of intelligent orchestration, however, their main focus is on the measure of resource optimization and service performance as opposed to secure platform extensiveness.



The scheduling-based studies can be used to give complementary information about the orchestration of resources given dynamical resource constraints. It has been demonstrated [14] that delay-sensitive task scheduling using deep learning reinforcement algorithms in heterogeneous network settings with an adaptive distribution of the computational and communication resources. The powerful scheduling methods also address the uncertainty and delay specifications in the coordination of distributed resources [15]. The ample design plans of communication-and computing resources reduce to identifying and identifying optimization, learning based, and shared resource provision as main paths in organizing different networks [16]. These studies contribute to the literature on dynamically orchestrating as much as they can, but in most instances, orchestration capabilities are viewed as extensions of an existing system rather than as deployable extensions, with explicit platform contracts.

III. GOVERNED EXTENSION FABRIC: SECURE PLANES, CAPABILITY CONTRACTS, AND LIFECYCLE CONTROL

3.1 Architectural Overview

A suggested architecture brings in a controlled extension fabric separating the stable orchestration center and autonomously evolved abilities. The principle behind the architectural design here is that one should not disrupt the main principles of identity, authorization, resource ownership, the workflow integrity, auditability, and authoritative network state by the extensibility. The architecture develops a specific limit of capabilities that extensions draw on and give validated returns, rather than precautions of extensions to the interior databases or special control devices.

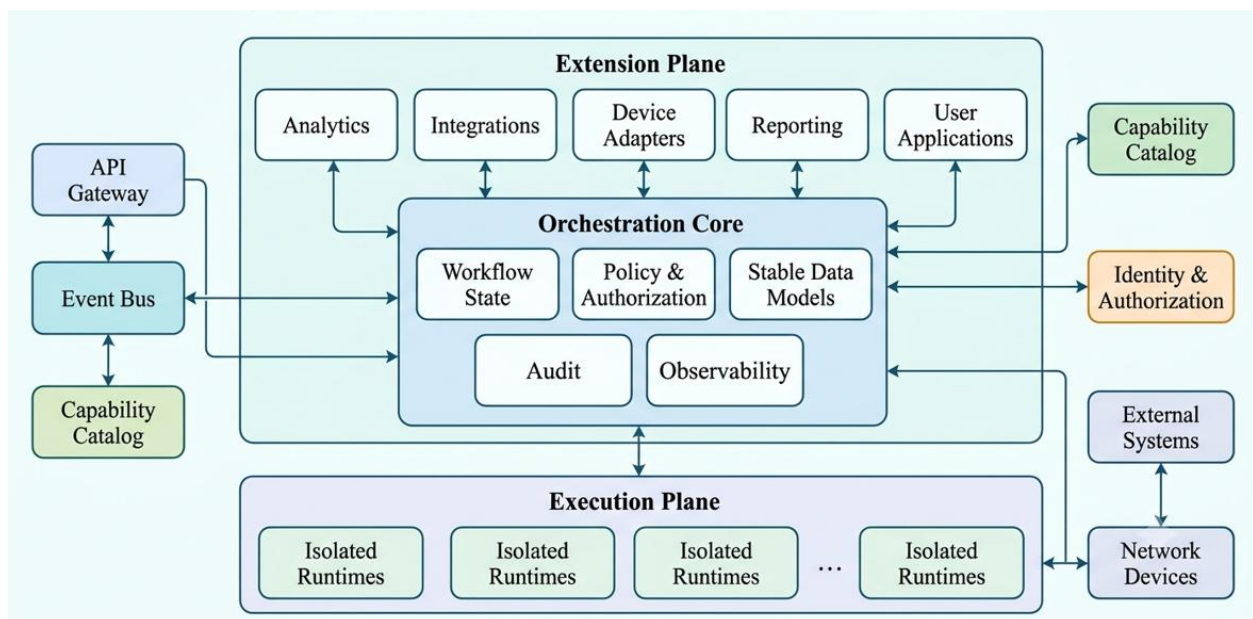


Figure 1- Secure and Extensible Orchestration Architecture

The architecture is made up of three closely related planes; the core plane, the extension plane and the execution plane. Platform state of authority and fixed contracts are anchored on the core plane. The extension water provides a managed condition of integrations, analytics, applications, policy pro-packs and device-specific hastenings. The execution plane is a perspective which provides isolated vistas of extensions which require the execution of special or more authoritative functions. This isolation allows optional functionality to be implemented without any effect on the unrelated core processes in case of failure and unavailability of an extension.

3.2 Core-Extension Plane Separation

Platform invariants, such as identity, authorization, accepted intent, resource ownership, workflow state, audit records and schemas of stable managed entities are owned by the core plane. Core services expose public contracts, rather than being implementation specific database structures. The extensions, therefore, communicate with authoritative resources by approved commands, queries and events as opposed to manipulating internal storage.

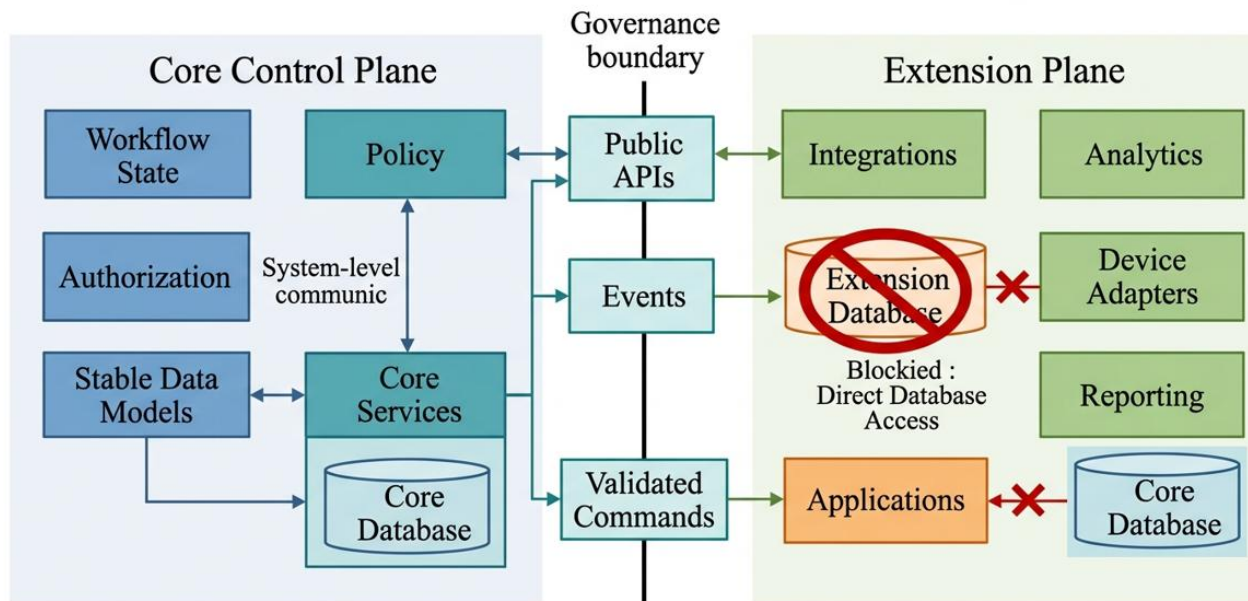


Figure 2- Core-Extension Plane Separation

This is a major difference in maintainability long-term. Internal databases and the persistence mechanisms and implementation technologies can evolve; without all the extensions being rearchitected. Better said, failure due to extensions is limited to a locality. Should an optional analytics service not be able to break down, the core will be capable of continuing to admit, reconcile and administer other unrelated resources. This isolates failure core control to a failure-isolated core where optional functionality boosts orchestration, not becoming a condition of essential platform functionality.

Integrations, analytics engines, user applications, device adapters, visualization services and policy packages can be found on the extension plane. The extensions are engaged with the platform through public APIs, registered capabilities and event subscriptions. They do not have access to core databases and they are unable to authoritative state. Any change of state execution must pass via fundamental commands, which are validated and consequently it can execute inherently authorization, schema authentication, ownership checks, policy checks and audit logging.

Environments that require to be individualized or grant operations that are privileged are provided by the execution plane. Data created in the platform might be exposed to device adapters and similar elements, but it is not included in the trusted core. This platform provides signed descriptions of the work, narrowly scoped credentials, network policies and resource limits. Adapters then manufacture results and supporting evidence in a formal format and the core consumes this information, potentially determining whether the reported outcome can be incorporated into workflow state that is authoritative.

3.3 Capability Contract and Service-Catalog Layer

The capability contract layer is one of the notable architectures. In any published machine-readable manifest, that contains its identity, ownership, version, compatible platform API range, its capabilities, required permissions, events subscriptions, generated schema, data classification, resource requirements, health endpoints and upgrade behavior.

Registration has therefore more than just installing software. The manifest is a definite statement of what the extension can be done and what it will need. RegistrationEnsures that the functions listed meet the policies on the platform, and constructs a subject to examine authorization grant.

The different types of capabilities are calculated with respect to the outcomes, rather than implementation destinations. As an example, an extension can provide telemetry collection, device-model visualization, condition diagnoses, exporting incidence, or showing a particular dataset. A service catalog documents such capabilities in such a way that planners and users can learn about the capabilities that are available without being aware of the deployment location or implementation details of a particular extension.

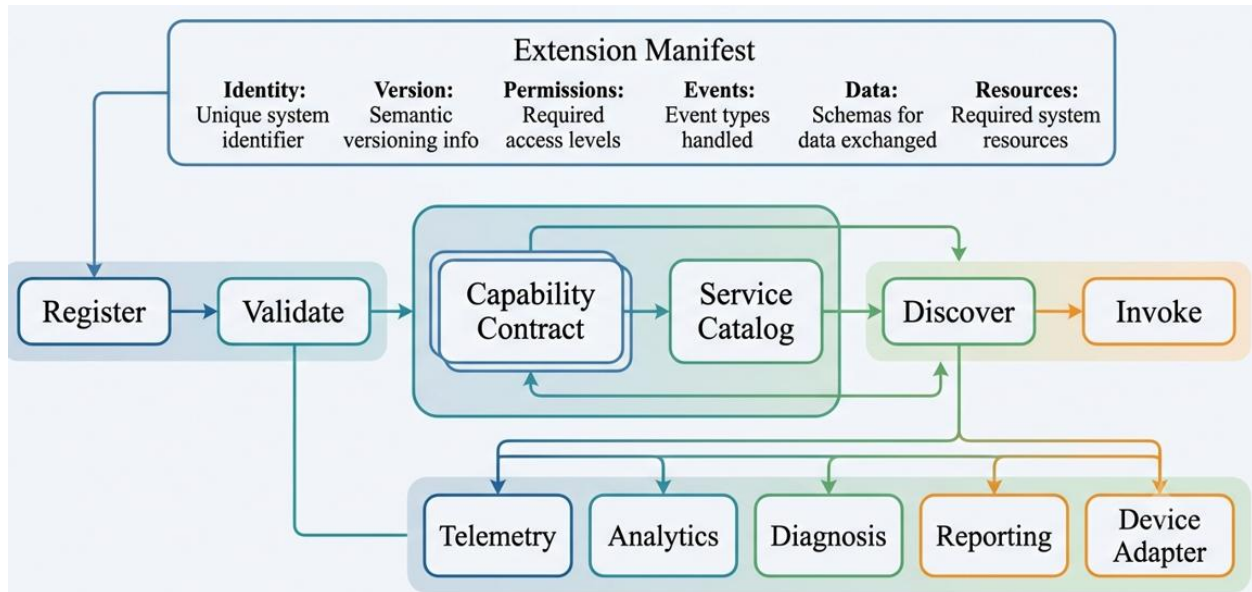


Figure 3- Capability Contract and Service-Catalog Model

The operational semantics (success and failure states, retry behavior, idempotency, timeout limits and pagination) is also defined in capability contracts. The continuing operations will give an identity of the task and not the need to have a synchronous request in operation. Transaction support, rollback limits and evidence requirements are also determined with device adapters. These definitions disallow the orchestration core to take unsafe choices about the conduct of extensions.

3.4 Versioned APIs and Event Evolution

Long-lived orchestration platforms need to explicitly support API and event evolution management. The architecture proposed is thus based on versioned contract with published compatibility policies. When appropriate, preferably additive changes are employed; however, in cases where semantic changes require a new version of contract, even though the underlying structure of the contract is the same, the current version of the contract is not compatible with a new version of the underlying contract, so the contract must be updated to support the version of the underlying contract, respectively.

Consumer testing entails testing representative requests and responses on supported versions, and allows an extension to be advertised. This will reduce the risk of a platform upgrade that would break extending programs without any know-upfront. The published depreciation periods enable an extension owner time to migrate and are not mutually exclusive with the core platform developing and never having to sustain old interfaces indefinitely.

Events are not commands, but are assumed to be facts that are promised. Event identifier, occurrence time, version of schema, and a retry context are all part of the events. Subscribers are able to sort out events by domain, severity, scope and sensitivity. The consumers have to use idempotent processing since events can be re-executed or re-played.

Any events that cannot be successfully delivered enter into a limited dead-letter process, this is noticeable to the extension owner in charge. This approach will prevent the inexorable consumer failures in the establishment of interminable queues as well as holding sufficient evidence to mishandle and recuperate in a controlled condition.

Public controller interfaces demonstrate the feasibility of northbound APIs and event-driven combination to automate an enterprise; the recommended structure is an expansion of the proposed model with the ability to register capabilities, authority delimitation, compatibility management, and lifecycle controls.

3.5 Isolation and Delegated Authority

Isolation of extension is accomplished using either separate processes, Containers or by using sandboxed execution environments. The budgets of CPU, memory, storage, queue and of network-connection are well defined. Network



communication is based upon a default-deny model, where any destination and protocol is only authorized when it is needed by an authorized capability.

host-level interfaces are also restricted as well as filesystem interfaces. As a result, an intensive analytics extension will end up eating up the worker capacity to be used by core reconciliation. Isolation converts the possible extension failures which can also lead to failure of the whole platform to scope failures.

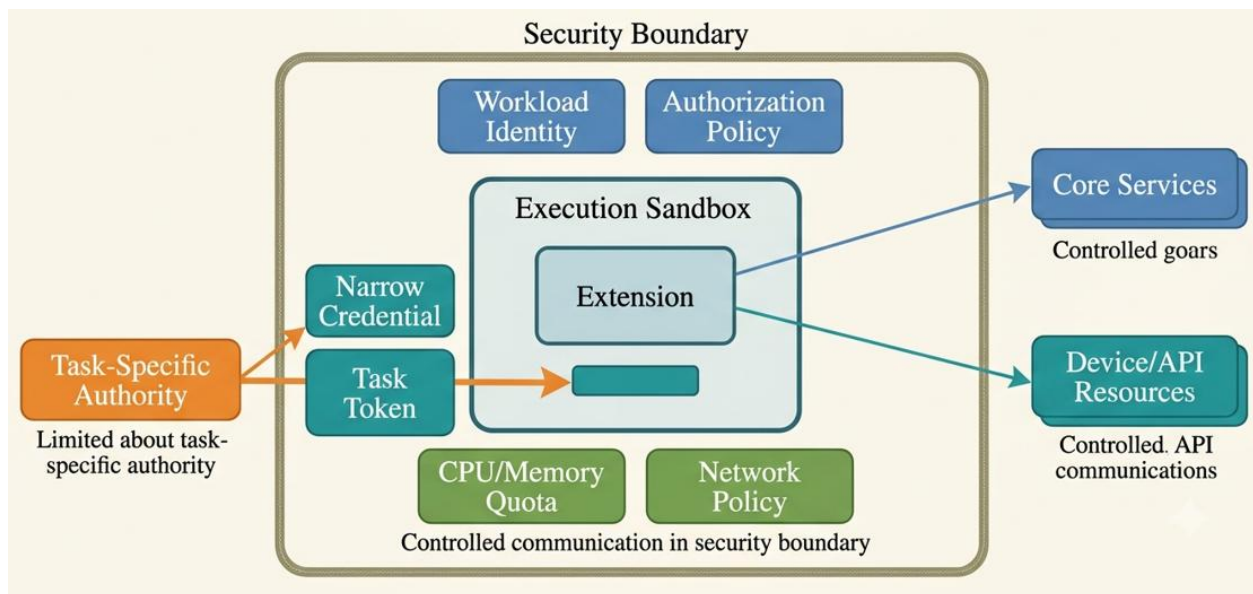


Figure 4- Isolated Extension Execution and Delegated Authorization

Delegated ability-specific authority is used to authenticate. The extensions get an identity as workload and permissions as per their registered capabilities. Zero-trust principles are more favorable to resource-based authorization over presumed-trust basing on network proximity [2] and workload identity frameworks also provide the benefit of producing verifiable workload identities [3].

Delegation is delimited. When an integration wants a workflow to be executed, the resulting execution authority is less restrictive than what the integration permissions are generally, and lasts only as long as the task is executed or its authorization lifetime. The original external principal exists in audit records and where such exists, the extension identity is retained. This creates the traceability at different levels of authorization.

3.6 Extension Lifecycle Governance

The initiation of extension lifecycle is admission control. Registration can encompass schema validation, static analysis, artifact-signature validation, provenance validation, vulnerability-policy validation, contract validation and resource-policy validation. Principles of software supply-chain like the ones embodied by SLISA may offer a basis to create provenance and integrity of artifacts [4].

Once it has been admitted, the deployment process occurs in a series of strictly controlled phases that encompass development, testing, limited scope deployment and general availability. Canary deployment may be restricted to a laboratory, location, tenant or tiny resource populace. The extension health is determined using process readiness, dependency availability, event latency, and error rates, along with the capability specific indicators.



13541



The analysis considers the time to build, complexity of implementation, and reusability, complexity of application code required. Particular attention is given to the potential capability contracts and standardized API to reduce integration specific logic. Lower implementation energy and reliance on the implementation platform means higher extensiveness with no corruption of architectural boundaries.

5.2 Compatibility and Evolution Validation

Compatibility tests run extensions on a variety of supported platform versions and schema changes that are controlled. The assessment logs violations of the agreement, needed changes in the source-code, incompatibilities in the APIs and dependencies in the up-grades. Additive schema changes are evaluated in isolation of semantic changes to find out whether the versioning practice preserves existing integrations. It is also considered whether extensions can be supported to run during periods of transition where two or more versions of contract are running.

5.3 Isolation and Failure Containment

Isolation tests are deliberately made to cause failures in the level of extension like crash of process, unavailable dependencies, resource-quota exhaustion, malformed event generation and storms of retries. In this case, the core API latency, reconciling age, and emergency-work-capacity are measured. The argument is that a survivable structure is that in which the failure of extensions does not substantially affect, irrelatively core functionality and whose resource management limits a bad extension to run to shared resource capacity.

5.4 Security and Authority Enforcement

Security tests analyze whether extensions do not go outside of the reported capabilities and permissions. Typical examples of scenarios covered by tests include the unauthorized access to data, event subscription escalation, a set of capabilities that have been forged, re-evaluation of execution tasks, and un-confirmed software updates. The workload identity is checked, delegated authorization, artifact verification, and capability-scoped execution controls are checked with the help of the assessment. Security breaches should be rejected or recorded and not given an opportunity to expand the power base of the issue at.

5.5 Lifecycle Resilience and Auditability

Controlled deployment, canary release, rollback, state migration and capability retirement are tested using lifecycle tests. When in a canary deployment, a known buggy extension version is deployed to test rollback. The reversible and consistent nature of state migration are gauged, and a test on retirement conducted in the evaluation of whether there are dependencies before the removal of a capability.

Audit assessment builds up the entire history of the extension such as registration, approval, deployment, invocation, delegating authority, upgrades and retirement. The assessment will overall pass when the failure of extensions is constrained, when security is ensured, whereby lifecycle operation can be reversed where necessary and where an operator can discover the owner of the failure and perform necessary recovery measures without needing to access internal databases of the platform.

VI. DISCUSSION

Extensibility will provide huge business benefit to enterprise orchestration systems, but will be long term structural encumbrances. The compatibility surface of the platform includes all its public APIs, event schemas, capability contracts, and integration interfaces. Versioning, migration, and depreciation of versions should be enforced rather than be random after relying on external teams. Extensibility thus is to be managed as a controlled architectural pledge as opposed to a mere feature of integrations.

The proposed architecture tackles this challenge with respect to explicit capability contracts and centralized service catalog. The operators of the platform can see the ownership, supported versions, permissions, dependencies, health status, and lifecycle state in the catalog. This visibility erases the bafflement of extensions being built or broken and puts order in maintaining compatibility. It might also assist organizations to distinguish between integrations that are actively maintained and those which are not maintained and/or were developed long ago.

Special extensions such as device adapters and execution parts are of much interest as far as security considerations are concerned. These elements bring functionality to the platform, but increase barriers to trust and how to fail as well. Cell-scoped authorization, resource quotas, isolated execution and signed artifacts thus provide the complementary control over the impact of their operation.



The architecture also presumes extensibility should not lead to runaway growth in integration mechanisms. When many special-purpose paths are host in a platform, it can become prone to redundant interfaces, policy differences, and hard-to-maintain dependencies. Alternatively, a smaller number of expressive mechanisms (versioned APIs, event subscriptions, capability contracts and controlled execution interfaces) can provide broader integration coverage without compromising architectural consistency.

In general, sustainable extensibility is a matter of having a balance between innovation and control. Addition of new capabilities should be made easy but ownership and security boundaries, compatibility and retirement must be maintained as simple throughout the lifecycle.

VII. CONCLUSION

The coverage of a sustainable enterprise network organizing system who requires regular architecture core with controlled and extendable extension plane. The isolation enables the organisations to incorporate new integrations, analytics, device adaptors, reporting services and the operational features without rendering the central data stores vulnerable and liberalizing underlying control processes. The architecture proposes this boundary that is defined by clearly defined capability manifestations, versioned contracts, standard APIs, event-based integration, isolated execution environments, delegated authorization, resource quotas, and lifecycle governance.

Ability is expressed by ensuring equity, rights, relationships, compatibility requirements as well as business expectations through the commitment of extensions to the platform at the time of commitments being made to the platform making them visible before becoming incorporated into the platform. Confidence contracts and conformance testing offer ways to handle change and minimize accidental compatibility failures. Isolation of runtimes and authorization based on capabilities limits the effect of malfunctioning or harmed extensions, and resource quotas inhibit single elements taking up disoutatory platform bandwidth. Lifecycle controls also facilitate controlled deployment, canary validation, rollback, state migration and retirement.

The resultant architecture considers extensibility as a managed platform feature as opposed to a pile of independent integration pathways. This style will enable development teams to be more innovative, leave the integrity, availability and traceability of the orchestration core intact. In the operational lifetime of the extension, dependencies, delegated authority, extension health and lifecycle state, can be observed.

REFERENCES

- [1] Cisco Systems, "Introduction to Cisco Catalyst Center API." [Online]. Available: [Cisco Catalyst Center API documentation](#)
- [2] S. Rose et al., Zero Trust Architecture, NIST Special Publication 800-207, Aug. 2020, doi: 10.6028/NIST.SP.800-207.
- [3] SPIFFE Project, "Secure Production Identity Framework for Everyone (SPIFFE) Specification." [Online]. Available: [SPIFFE Specification](#)
- [4] SLSA Project, "Supply-chain Levels for Software Artifacts (SLSA) Specification." [Online]. Available: [SLSA Specification](#)
- [5] OpenTelemetry, "OpenTelemetry Specification." [Online]. Available: [OpenTelemetry Specification](#)
- [6] R. Chandramouli, Security Strategies for Microservices-Based Application Systems, NIST Special Publication 800-204, Aug. 2019, doi: 10.6028/NIST.SP.800-204.
- [7] N. F. S. De Sousa, D. L. Perez, C. E. Rothenberg, and P. H. Gomes, "End-to-end service monitoring for zero-touch networks," J. ICT Standardization, vol. 9, no. 2, pp. 91–112, May 2021.
- [8] K. Abbas, M. Afaq, T. A. Khan, A. Rafiq, and W.-C. Song, "Slicing the core network and radio access network domains through intent-based networking for 5G networks," Electronics, vol. 9, no. 10, Art. no. 1710, Oct. 2020, doi: 10.3390/electronics9101710.
- [9] L. Pang, C. Yang, D. Chen, Y. Song, and M. Guizani, "A survey on intent-driven networks," IEEE Access, vol. 8, pp. 22862–22873, 2020, doi: 10.1109/ACCESS.2020.2969208.
- [10] G. Zhou, L. Zhao, G. Zheng, S. Song, J. Zhang, and L. Hanzo, "Multiobjective optimization of space–air–ground-integrated network slicing relying on a pair of central and distributed learning algorithms," IEEE Internet Things J., vol. 11, no. 5, pp. 8327–8344, Mar. 2024, doi: 10.1109/JIOT.2023.3319130.



- [11] Y. Gao, Z. Ye, and H. Yu, "Cost-efficient computation offloading in SAGIN: A deep reinforcement learning and perception-aided approach," *IEEE J. Sel. Areas Commun.*, vol. 42, no. 12, pp. 3462–3476, Dec. 2024, doi: 10.1109/JSAC.2024.3459073.
- [12] C. Qiu, H. Yao, X. Wang, N. Zhang, F. R. Yu, and D. Niyato, "AI-chain: Blockchain energized edge intelligence for beyond 5G networks," *IEEE Netw.*, vol. 34, no. 6, pp. 62–69, Nov./Dec. 2020, doi: 10.1109/MNET.021.1900617.
- [13] J. Du, C. Jiang, J. Wang, Y. Ren, and M. Debbah, "Machine learning for 6G wireless networks: Carrying forward enhanced bandwidth, massive access, and ultrareliable/low-latency service," *IEEE Veh. Technol. Mag.*, vol. 15, no. 4, pp. 122–134, Dec. 2020, doi: 10.1109/MVT.2020.3019650.
- [14] C. Zhou, W. Wu, H. He, P. Yang, F. Lyu, N. Cheng, and X. Shen, "Deep reinforcement learning for delay-oriented IoT task scheduling in SAGIN," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 911–925, Feb. 2021, doi: 10.1109/TWC.2020.3029143.
- [15] Q. Chen, W. Meng, S. Han, C. Li, and H.-H. Chen, "Robust task scheduling for delay-aware IoT applications in civil aircraft-augmented SAGIN," *IEEE Trans. Commun.*, vol. 70, no. 8, pp. 5368–5385, Aug. 2022, doi: 10.1109/TCOMM.2022.3186997.
- [16] Q. Chen, Z. Guo, W. Meng, S. Han, C. Li, and T. Q. S. Quek, "A survey on resource management in joint communication and computing-embedded SAGIN," *IEEE Commun. Surveys Tuts.*, early access, Jul. 2024, doi: 10.1109/COMST.2024.3421523.