RESEARCH ARTICLE

SAMRIDDHI Volume 18, Issue 3, 2026 Print ISSN: 2229-7111 Online ISSN: 2454-5767

Automated DDI Migration from Infoblox NIOS to BloxOne: A Python-Based Approach for Large-Scale Multi-Country Deployments

Abdalhadi Skaik*

Wipro Arabia Ltd, Saudi Arabia

Abstract

Organizations looking for more scalability, central management, and operational robustness have made the move to cloud-native systems of enterprise Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP), and IP Address Management (IPAM) services a strategic priority. But, in the case of large-scale DDI migration, several issues make it difficult, such as data integrity requirements, heterogeneous network configurations, and minimum service disruption. This paper is aimed at outlining an automated framework for migrating DDI configurations off Infoblox NIOS to Infoblox BloxOne, implemented as a Python script, that can be used for large-scale, multi-country enterprise deployments. The suggested extract, transform, load (ETL) approach uses Infoblox Web APIs to automate the process of extracting data from the source, transforming the schema, validating it, and importing it to the cloud in a consistent manner, while minimizing manual effort. The framework includes automated validation, exception handling, and parameterized execution, enabling deployments to be repeated across geographically distributed environments. Performance evaluation shows that there are significant advantages in terms of effectiveness, accuracy, scalability, and operating reliability over the traditional manual migration practices. The proposed solution offers a viable and scalable solution for enterprise DDI modernization and lays the groundwork for future cloud-native network automation and Infrastructure-as-Code programs.

Keywords: Automated DDI Migration, Infoblox NIOS, Infoblox BloxOne, Python Automation, Network Automation, DNS Management, DHCP Migration, IP Address Management (IPAM), Cloud-Native Networking, Infrastructure as Code (IaC).

SAMRIDDHI : A Journal of Physical Sciences, Engineering and Technology (2026); DOI: 10.18090/samriddhi.v18i03.02

Introduction

Digital transformation of enterprise information technology has been rapid, leading to the proliferation of cloud-native infrastructure and software-defined network services for organizations to modernize their core networking platforms to scale, optimize operations, and deliver services. These are all fundamental services and are called the Foundation Services of the Enterprise Network Services (ENWS): Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP), and IP Address Management (IPAM) [4], [5], [9], [18]. These on-premises DDI platforms are becoming increasingly constrained when it comes to scale, agility, and centralized control, especially as enterprise networks are growing over more geographic regions and are increasingly hosting hybrid and cloud-based applications [11], [14].

Built on a highly reliable grid architecture, Infoblox NIOS is one of the most well-known enterprise DDI platforms, offering comprehensive DNS, DHCP, and IPAM management. Many organizations, however, have been drawn to move away from legacy NIOS deployments to Infoblox BloxOne DDI, a cloud-managed platform that provides better visibility and increased IT control, policy-driven automation, and seamless integration with today's enterprise infrastructure, onto which they can deploy new workloads and applications [6, 8]. This transition is in line with the general development of network programmability and Infrastructure as Code (IaC) that provision infrastructure resources through programmable interfaces instead of manual administrative procedures [15][20].

While there are clear strategic benefits of cloud-based DDI platforms, enterprise-scale DDI environments are a

Corresponding Author: Abdalhadi Skaik, Wipro Arabia Ltd, Saudi Arabia, e-mail: abdalhadis@gmail.com

How to cite this article: Skaik, A. (2026). Automated DDI Migration from Infoblox NIOS to BloxOne: A Python-Based Approach for Large-Scale Multi-Country Deployments. *SAMRIDDHI : A Journal of Physical Sciences, Engineering and Technology,* 18(3), 22-42.

Source of support: Nil

Conflict of interest: None

technically challenging migration. DDI migration differs from standard application migrations because it is transferring highly interconnected configuration objects – such as DNS zones, resource records, DHCP scopes, IP address allocations, network containers and associated metadata – without compromising network services or being inconsistent. Migration errors can cause DNS resolution problems, wrong IP address assignments, network downtime, or service outages that impact business continuity as you know it [3] [4] [18]. That means that when organizations are undertaking large-scale migrations across countries, they need migration plans that ensure accuracy, repeatability, and low operational risk.

The conventional migration methods are mostly manual and require the export of configuration information from the source environment, subsequent transformation of the exported data to the format of the target environment, and finally import of the migrated configuration after thorough validation. While possible for smaller sets, manual processes become more cumbersome the more complex the deployment becomes. Large multi-location, multi-national companies might maintain thousands of DNS records, hundreds of DHCP scopes, and huge IPAM databases at numerous retail and regional locations. In this case, moving a site manually adds a lot of engineering effort, long project timelines, variable configuration, and the potential for human error [1, 3, 11]. The need for automation frameworks to perform migration activities with greater consistency and scalability, and with greater operational reliability, has grown more acute.

Programmable control of network services and infrastructure resources has greatly changed the way enterprise networks are managed, as seen through recent advances in network automation, RESTful application programming interfaces (APIs), Infrastructure as Code, and Python-based orchestration [10,15,17,20]. The Python language has become one of the most prominent languages used for network automation for its huge library of libraries, ease of use, portability, and tight integration with REST-based management platforms [10, 16]. At the same time, the Infoblox Web API (WAPI) also enables detailed programmatic access to all NIOS-managed DDI objects, which can allow for automation of data extraction, transformation, validation, and migration workflows, thereby reducing manual effort [7]. The technological innovations are well-positioned as building blocks for creating enterprise-wide scalable migration frameworks.

While some work has investigated end-to-end automation frameworks for hybrid networking environments [2] and introduced declarative IPAM management and lifecycle orchestration, the contribution to the field of end-to-end automation frameworks for large-scale migration from Infoblox NIOS to BloxOne across geographically distributed enterprise deployments has been relatively limited. The documents available in the wild tend to be vendor-specific or concentrate on specific configuration issues instead of complete migration methodologies that take automated extraction, transformation, validation, error handling, and deployment verification into account [2, 6, 7]. The gap is underscored by the desire for research-based frameworks that are practically applicable and reflect the practical needs of enterprise deployment and show measurable operational benefits.

To surmount this challenge, this paper introduces an automated migration framework, written in Python, which simplifies the migration of DDI configuration from Infoblox NIOS to Infoblox BloxOne through an Extract–Transform–Load (ETL) architecture. The framework automates the extraction of DDI objects, transforms the heterogeneous configuration data into BloxOne-compatible formats, validates migrated objects for consistency and completeness, and provides repeatable deployment of the framework across multiple countries by enabling parameterized execution. The proposed framework minimizes the need for manual work, enhances the accuracy of migration, and significantly speeds up deployment activities, thus offering a practical solution for enterprise organizations that are planning to carry out cloud-native DDI modernization.

This research has four major contributions. First, it provides a structured and reusable automation pattern to ease enterprise-scale migration from Infoblox NIOS to BloxOne. Second, it illustrates how migrating with automation and REST API integration can significantly enhance the efficiency and consistency of migration for geographically dispersed environments. Third, it tests the capabilities of the proposed framework for operational performance and practical benefits as compared to conventional manual migration practices. Lastly, the study offers implementation suggestions and best practices that could inspire future research and industrial applications of network automation, cloud-native infrastructure migration, and intelligent DDI lifecycle management.

Background and Related Work

The modernization of enterprise network infrastructure has significantly transformed the management of foundational network services, particularly Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP), and IP Address Management (IPAM), collectively referred to as DDI. As organizations increasingly migrate business-critical workloads to hybrid and cloud-native environments, DDI platforms have evolved from conventional on-premises appliances into centralized, software-defined services capable of supporting geographically distributed infrastructures. This transition has intensified research into network programmability, cloud-based infrastructure management, automated configuration provisioning, and Infrastructure as Code (IaC), all of which seek to improve operational efficiency while reducing administrative complexity [5], [11], [15], [20].

Recent advances in enterprise networking have also emphasized the importance of automation in minimizing configuration errors, accelerating infrastructure deployment,

and improving service reliability. Although numerous studies have investigated network automation frameworks and programmable infrastructure, relatively few have specifically addressed the challenges associated with large-scale migration of enterprise DDI environments between heterogeneous management platforms. Consequently, understanding the evolution of DDI technologies, existing migration methodologies, and current automation practices provides an essential foundation for developing effective migration frameworks capable of supporting enterprise-scale cloud transformation.

Foundations of DDI Architecture

Enterprise network communication relies on three tightly integrated services: Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP), and IP Address Management (IPAM). Together, these services establish the operational backbone that enables reliable network communication, device connectivity, and centralized management of address resources across organizational infrastructures [4], [5], [9], [18].

The Domain Name System functions as the Internet's distributed naming infrastructure by translating human-readable domain names into numerical IP addresses that network devices use for communication. The hierarchical design of DNS enables scalable and fault-tolerant name resolution through authoritative servers, recursive resolvers, and caching mechanisms. RFC 1034 and RFC 1035 define the architectural principles governing DNS operation, including zone delegation, resource records, and namespace hierarchy, which remain fundamental to contemporary enterprise networking [9], [12]. Within corporate environments, DNS extends beyond public Internet name resolution to support internal applications, authentication services, service discovery, cloud workloads, and distributed enterprise resources.

Complementing DNS, the Dynamic Host Configuration Protocol (DHCP) automates the assignment of network parameters including IP addresses, subnet masks, default gateways, and DNS server information to client devices. By replacing manual address configuration with dynamic lease allocation, DHCP simplifies network administration while reducing address conflicts and configuration inconsistencies. RFC 2131 formalizes DHCP operations through standardized message exchanges involving discovery, offer, request, and acknowledgment phases that collectively enable efficient address provisioning across diverse network environments [18]. In modern enterprises, DHCP services are particularly important for supporting mobile devices, virtual machines, wireless infrastructures, and rapidly changing cloud-based workloads.

IP Address Management (IPAM) provides centralized governance over enterprise address spaces by maintaining comprehensive inventories of IP allocations, subnet utilization, address reservations, and associated metadata. Unlike DNS and DHCP, which provide operational services, IPAM focuses primarily on planning, documenting, auditing, and optimizing address utilization. Effective IPAM improves resource allocation, minimizes address exhaustion, simplifies troubleshooting, and supports regulatory compliance through comprehensive visibility into network resources [14]. As organizations increasingly deploy IPv6 alongside IPv4, IPAM systems have become even more critical for managing significantly larger address spaces and ensuring consistent allocation policies across heterogeneous infrastructures [19].

Although DNS, DHCP, and IPAM perform distinct operational functions, they are highly interdependent. DNS records frequently reference IP addresses allocated through DHCP or maintained within IPAM databases, while IPAM repositories synchronize with both DNS and DHCP services to maintain consistent network inventories. This integration enables centralized policy enforcement, improves operational consistency, and facilitates automation across enterprise environments. Consequently, modern DDI platforms combine these traditionally independent services into unified management systems capable of providing comprehensive visibility and administrative control [6], [14].

The increasing complexity of enterprise infrastructures, including hybrid cloud deployments, edge computing environments, containerized applications, and geographically distributed branch networks, has elevated DDI from a supporting network function to a strategic infrastructure component. Organizations now require DDI platforms that not only provide reliable service delivery but also integrate seamlessly with orchestration tools, cloud platforms, and programmable network management frameworks. These evolving operational requirements have directly influenced the development of modern DDI solutions such as Infoblox BloxOne, which extend conventional DDI capabilities through cloud-native architectures and API-driven management [6], [11].

Infoblox NIOS and BloxOne: Evolution of Enterprise DDI Platforms

The rapid expansion of enterprise cloud computing, hybrid networking, and distributed digital services has fundamentally changed the requirements for DDI management platforms. Traditional DDI infrastructures were primarily designed for centralized on-premises data centers, where network resources could be managed within relatively stable and geographically confined environments. However, the increasing adoption of cloud platforms, Software-as-a-Service (SaaS), edge computing, Internet of Things (IoT) devices, and remote branch connectivity has significantly increased the complexity of enterprise networking. These developments have driven the evolution of DDI solutions from appliance-based management systems toward cloud-native platforms that emphasize scalability, centralized visibility, automation, and continuous service availability [11], [14].

Infoblox NIOS has been widely deployed as an enterprise-grade DDI platform because of its ability to integrate DNS, DHCP, and IPAM services within a unified Grid architecture. The NIOS Grid enables multiple appliances to operate as a synchronized management domain, allowing administrators to centrally configure DNS zones, DHCP scopes, address allocations, and network policies while maintaining distributed service delivery across geographically dispersed locations [8]. The architecture incorporates Grid Masters, Grid Members, and synchronized configuration databases, ensuring operational consistency and fault tolerance throughout enterprise deployments. Administrative activities are coordinated through centralized policy management, reducing configuration drift and simplifying governance across multiple network sites [6], [8].

One of the principal strengths of Infoblox NIOS lies in its mature management capabilities. The platform provides integrated security controls, comprehensive audit logging, role-based administrative access, automated backup mechanisms, and high-availability deployment options that have made it suitable for large enterprises operating mission-critical network infrastructures. In addition, the Infoblox Web API (WAPI) enables administrators and automation frameworks to interact programmatically with virtually every managed DDI object through RESTful interfaces, thereby supporting script-based provisioning, configuration management, monitoring, and reporting [7], [17]. This programmable interface has become increasingly important as organizations adopt automation strategies aimed at reducing repetitive administrative tasks and improving deployment consistency.

Despite these advantages, conventional on-premises DDI architectures face several operational limitations in modern enterprise environments. Infrastructure expansion often requires additional hardware appliances, manual capacity planning, localized maintenance activities, and periodic software upgrades that may increase operational costs and administrative complexity. Furthermore, organizations operating across multiple countries frequently encounter challenges associated with distributed management, inconsistent configuration practices, and varying operational procedures among regional engineering teams. As enterprise infrastructures become increasingly dynamic, these limitations reduce the agility required to support continuous digital transformation initiatives [11], [14].

To address these emerging requirements, Infoblox introduced BloxOne DDI, a cloud-native platform designed to deliver centralized DDI management through Software-as-a-Service (SaaS). Unlike appliance-centric architectures, BloxOne separates centralized management functions from distributed service execution by leveraging cloud-based control planes while maintaining lightweight service components at customer sites. This architectural model provides greater operational flexibility, centralized visibility, simplified software maintenance, and improved scalability for organizations managing globally distributed infrastructures [6].

BloxOne incorporates several capabilities that distinguish it from traditional DDI management systems. These include cloud-hosted administration, API-first architecture, policy-driven configuration management, integrated analytics, centralized monitoring dashboards, and seamless integration with enterprise automation platforms. The platform also supports bulk object import, programmable configuration management, role-based access control, and extensive REST API functionality, enabling organizations to automate provisioning, policy enforcement, and infrastructure lifecycle management through external orchestration tools [6], [17]. Such capabilities align closely with contemporary DevOps methodologies and Infrastructure as Code practices, where infrastructure resources are managed through version-controlled code and automated deployment pipelines rather than manual configuration [15].

From a migration perspective, however, transitioning from Infoblox NIOS to BloxOne presents several technical challenges. Although both platforms provide integrated DDI services, they differ in management architecture, object representation, configuration schemas, API implementations, and deployment models. Configuration objects exported from NIOS frequently require transformation before they become compatible with BloxOne import mechanisms. Dependencies among DNS zones, resource records, DHCP networks, IP reservations, network containers, extensible attributes, and associated metadata further increase migration complexity. Consequently, successful migration requires systematic extraction, transformation, validation, and dependency resolution processes capable of preserving both configuration integrity and operational continuity throughout the transition [6], [7].

Another important consideration involves maintaining uninterrupted business operations during migration. Enterprise DNS and DHCP services support critical business applications, authentication systems, endpoint connectivity, cloud services, and customer-facing platforms. Even minor configuration inconsistencies may result in service interruptions, address allocation failures, or application outages. Accordingly, migration strategies must incorporate rigorous validation procedures, staged deployment mechanisms, rollback capabilities, and comprehensive verification processes to ensure that migrated configurations accurately replicate the operational state of the source environment [3], [6].

The evolution from Infoblox NIOS to BloxOne therefore represents more than a software upgrade; it reflects a broader transition toward cloud-native infrastructure management characterized by automation, programmability, centralized governance, and continuous operational visibility. As enterprises continue modernizing their networking infrastructures, automated migration methodologies become increasingly essential for reducing deployment

risk, accelerating implementation timelines, and ensuring consistent service delivery across geographically distributed environments.

Figure 1 is conceptualized by the authors based on the architectural principles described in the Infoblox NIOS Administrator Guide, Infoblox BloxOne Documentation, REST architectural principles, and enterprise cloud networking literature [6]–[8], [17].

Enterprise Network Automation and Infrastructure as Code

The increasing scale and complexity of modern enterprise networks have fundamentally transformed the way network infrastructure is designed, deployed, and managed. Traditional network administration relied heavily on manual configuration through command-line interfaces (CLI), graphical management consoles, and device-specific administrative procedures. Although these methods were adequate for relatively small and static environments, they have become increasingly inefficient for organizations operating hybrid cloud infrastructures, geographically distributed branch offices, software-defined networks, and continuously evolving digital services. Consequently, enterprise networking has undergone a paradigm shift toward automation, programmability, and policy-driven infrastructure management, enabling organizations to achieve greater operational consistency, scalability, and resilience [15], [20].

Network automation refers to the use of software, programmable interfaces, and orchestration tools to perform network provisioning, configuration, monitoring, validation, and lifecycle management with minimal human intervention. Rather than executing repetitive administrative tasks manually, engineers develop automation workflows capable of performing standardized operations repeatedly while ensuring consistent configuration across large infrastructures. This approach reduces deployment time, minimizes human error, improves configuration consistency, and enables rapid adaptation to changing business requirements [20].

The emergence of Representational State Transfer (REST) architectures has significantly accelerated network automation by providing standardized communication mechanisms between management applications and network services. RESTful APIs expose infrastructure resources as programmable objects that can be queried, modified, validated, and monitored using standard HTTP operations. This architectural model has become the foundation of modern cloud platforms, enterprise management systems, and network orchestration frameworks because it enables interoperability between heterogeneous technologies while supporting modular software development [17]. Within enterprise DDI platforms, REST APIs provide secure access to DNS records, DHCP configurations, IPAM databases, network containers, and administrative policies, allowing automation frameworks to perform complex configuration tasks programmatically rather than through manual interfaces [6], [7].

Python has emerged as the dominant programming language for network automation due to its readability, extensive standard library, platform independence, and broad ecosystem of third-party modules. Compared with lower-level programming languages, Python enables rapid development of automation scripts while maintaining high code maintainability and portability. Its support for HTTP communication, JSON processing, data transformation, concurrent execution, logging, exception handling, and

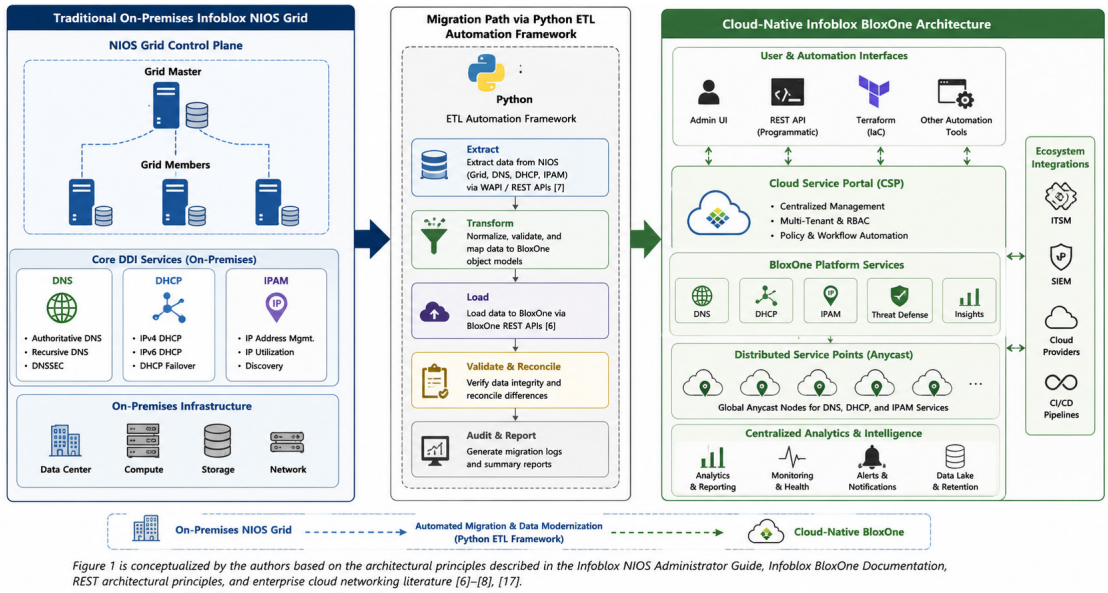


Figure 1: Evolution of Enterprise DDI Architecture: From Infoblox NIOS to Cloud-Native BloxOne

API integration makes it particularly suitable for enterprise infrastructure automation [10], [16]. Furthermore, Python integrates seamlessly with orchestration platforms, Infrastructure as Code frameworks, and cloud management systems, enabling organizations to develop reusable automation pipelines capable of supporting large-scale enterprise operations [20].

A closely related concept is Infrastructure as Code (IaC), which extends software engineering principles to infrastructure management by representing infrastructure configurations as machine-readable code stored within version-controlled repositories. Instead of manually configuring network devices and management platforms, engineers define infrastructure resources declaratively or procedurally, allowing automated deployment tools to provision and maintain consistent environments across multiple locations [15]. IaC introduces several software engineering practices into infrastructure management, including version control, change tracking, peer review, automated testing, continuous integration, rollback mechanisms, and configuration reproducibility. These capabilities substantially improve governance while reducing operational variability among distributed engineering teams.

Declarative infrastructure management has gained considerable attention in enterprise networking because it allows administrators to define the desired operational state rather than specifying individual configuration procedures. Automation systems subsequently determine the actions required to reconcile the actual infrastructure with the intended configuration. Cherukupalle [2] demonstrated how declarative IPAM and DNS lifecycle automation using Infoblox NIOS and Terraform can improve configuration consistency across hybrid cloud environments while reducing manual administrative overhead. Such approaches illustrate the growing convergence between network engineering and software development methodologies, where infrastructure components are managed through standardized automation pipelines rather than isolated administrative tasks.

Despite these advances, enterprise network automation presents several technical challenges. Automated workflows frequently operate across heterogeneous infrastructures comprising multiple vendors, legacy platforms, cloud services, and evolving network standards. Differences in object models, configuration schemas, authentication mechanisms, and API implementations require sophisticated transformation and validation processes capable of preserving configuration integrity during automated operations. Moreover, automation frameworks must incorporate comprehensive error detection, exception handling, dependency resolution, rollback procedures, and auditing mechanisms to ensure operational reliability under diverse deployment scenarios [3].

Data quality represents another critical consideration in automation systems. Since automation executes configuration changes at machine speed, errors originating from inaccurate source data may propagate rapidly throughout enterprise infrastructures. Van Roozendaal et al. [3] emphasized that maintaining high-quality network configuration data is essential for achieving reliable automation outcomes. Accordingly, effective automation frameworks should incorporate validation mechanisms that verify data completeness, identify inconsistencies, detect duplicate records, and preserve dependency relationships before executing configuration changes. Such validation becomes particularly important during DDI migration projects, where inaccuracies in DNS records, DHCP scopes, or IP address allocations may directly affect network availability and business continuity.

Within the context of DDI migration, automation extends beyond simple configuration transfer by integrating multiple operational stages into a coordinated workflow. These stages typically include extracting configuration objects from the source platform, transforming heterogeneous datasets into destination-compatible formats, validating data consistency, resolving object dependencies, importing configurations through programmable APIs, and verifying successful deployment through post-migration integrity checks. Integrating these activities into a unified automation framework substantially reduces engineering effort while improving repeatability, auditability, and deployment confidence across enterprise-scale environments [6], [7], [20].

The convergence of RESTful APIs, Python-based automation, Infrastructure as Code, and cloud-native management platforms has therefore established a robust technological foundation for enterprise DDI modernization. Organizations increasingly recognize automation not merely as a productivity enhancement but as a strategic capability that enables faster infrastructure deployment, stronger governance, improved service reliability, and scalable management of globally distributed network resources. These technological developments directly motivate the design of automated migration frameworks capable of supporting complex transitions from legacy DDI platforms to cloud-native environments.

Table 1 is synthesized by the authors based on concepts presented in Infrastructure as Code literature, enterprise network programmability research, REST architectural principles, Python automation practices, and Infoblox automation documentation [2], [6], [7], [10], [15]–[17], [20].

Existing Approaches to DDI Migration and Lifecycle Automation

The continuous evolution of enterprise networking has stimulated significant research into network automation, infrastructure orchestration, and cloud migration methodologies. While numerous studies have investigated automation techniques for network provisioning and lifecycle management, relatively limited research has focused specifically on comprehensive migration frameworks for enterprise DDI environments. Existing

Table 1: Comparison of Traditional DDI Administration and Automated Infrastructure-Based DDI Management

Feature	*Traditional manual administration*	*Automated infrastructure-based management*
Configuration Method	Manual CLI or GUI configuration	Python scripts, APIs, and automation workflows
Deployment Speed	Slow and engineer-dependent	Rapid and repeatable
Human Error	High probability	Significantly reduced through automation
Scalability	Limited by administrative effort	Supports enterprise-scale deployments
Configuration Consistency	Varies among administrators	Standardized and reproducible
Validation Process	Manual verification	Automated validation and integrity checks
Change Management	Difficult to audit	Version-controlled and traceable
Multi-Country Deployment	Time-consuming	Parameterized and highly scalable
Integration Capability	Limited	Extensive API and cloud integration
Operational Efficiency	Moderate	High

literature generally concentrates on individual automation capabilities such as DNS management, IP address allocation, Infrastructure as Code (IaC), or API-driven provisioning, rather than providing integrated methodologies capable of supporting large-scale migrations between heterogeneous DDI platforms [2], [11], [20].

Traditional DDI migration projects have historically relied on manual administrative procedures supported by vendor-specific export and import utilities. These approaches typically involve exporting DNS zones, DHCP scopes, IP address inventories, and related configuration objects from the source environment before manually transforming the exported datasets into formats compatible with the target platform. Although vendor-provided migration utilities simplify portions of this workflow, they often require substantial human intervention for schema conversion, dependency verification, configuration validation, and post-migration testing [6], [8]. Consequently, manual migration remains labor-intensive, particularly for multinational organizations managing thousands of configuration objects distributed across numerous administrative domains.

Several commercial DDI vendors have introduced migration utilities intended to facilitate platform modernization. However, these tools generally prioritize functional data transfer rather than comprehensive workflow automation. Administrative teams are frequently required to execute multiple sequential operations, perform manual data cleansing, resolve configuration inconsistencies, verify imported objects, and repeat failed migration stages when errors occur. Such semi-automated workflows improve migration efficiency compared with entirely manual approaches but still depend heavily on engineering expertise and procedural consistency [6], [8]. As deployment scale increases, these limitations become increasingly pronounced, often extending project timelines and elevating operational risk.

Recent developments in Infrastructure as Code have significantly influenced enterprise DDI management by introducing declarative infrastructure provisioning and automated configuration management. Rather than manually configuring network services, IaC frameworks allow administrators to define desired infrastructure states through machine-readable configuration files that can be version-controlled, reviewed, tested, and deployed automatically [15]. This paradigm has transformed infrastructure management by introducing software engineering principles such as reproducibility, configuration consistency, collaborative development, and automated change management into network operations.

An important contribution within this domain was presented by Cherukupalle [2], who demonstrated the application of Terraform to automate DNS and IPAM lifecycle management using Infoblox NIOS within hybrid cloud environments. The study illustrated how declarative configuration management can simplify infrastructure provisioning, improve configuration consistency, and reduce manual administrative overhead. Nevertheless, the proposed approach primarily addresses ongoing infrastructure lifecycle management rather than comprehensive migration between heterogeneous DDI platforms. Consequently, several migration-specific challenges, including object transformation, dependency resolution, configuration validation, staged deployment, and migration verification, remain insufficiently addressed.

Network programmability has similarly emerged as a major research area supporting automated infrastructure management. Edelman et al. [20] emphasized that programmable network infrastructures provide substantial improvements in deployment speed, operational consistency, and administrative scalability by exposing network services through programmable interfaces. RESTful APIs enable automation platforms to interact directly with management systems, allowing configuration objects to be created, modified, queried, and validated programmatically [17]. Infoblox NIOS supports this capability through the Web API (WAPI), which provides comprehensive programmatic

access to DNS, DHCP, and IPAM objects, thereby enabling automation frameworks to replace repetitive administrative tasks with standardized software workflows [7].

Python has become the predominant implementation language for network automation because it combines expressive syntax with extensive support for REST communication, JSON serialization, concurrent processing, exception handling, logging, and data transformation [10], [16]. These capabilities make Python particularly suitable for implementing Extract-Transform-Load (ETL) pipelines, where heterogeneous datasets must be extracted from source systems, transformed into target-compatible schemas, validated for integrity, and imported into destination environments. Despite these advantages, relatively few published studies describe complete Python-based migration frameworks specifically designed for enterprise-scale DDI modernization.

Another important consideration concerns data quality during automated migration. Successful migration depends not only on efficient automation but also on maintaining the integrity of configuration data throughout every stage of the migration process. Van Roozendaal et al. [3] observed that poor-quality network data can propagate rapidly through automated systems, producing widespread operational failures if validation mechanisms are inadequate. Consequently, modern migration frameworks increasingly incorporate integrity verification, duplicate detection, dependency analysis, and consistency validation to ensure that migrated configurations accurately represent the operational state of the source infrastructure before production deployment.

More broadly, cloud migration research has consistently demonstrated that automation significantly reduces deployment risk while improving repeatability and operational governance. Automated workflows eliminate many sources of human error associated with repetitive administrative tasks, accelerate deployment schedules, improve documentation accuracy, and facilitate standardized implementation across geographically distributed environments [11], [14], [20]. Nevertheless, most existing migration methodologies focus on virtual machines, cloud applications, storage systems, or general infrastructure services rather than specialized DDI platforms whose tightly interconnected DNS, DHCP, and IPAM objects require coordinated migration strategies.

Collectively, the existing body of literature establishes a strong theoretical foundation for network automation, Infrastructure as Code, programmable infrastructure management, and API-driven administration. However, comprehensive studies describing fully automated, enterprise-scale migration frameworks specifically designed to transition DDI services from Infoblox NIOS to BloxOne remain scarce. This deficiency provides a compelling motivation for developing integrated automation methodologies capable of addressing the operational, architectural, and validation challenges associated with large-scale DDI modernization.

Research Gap and Motivation

The literature reviewed in the preceding sections demonstrates substantial progress in network automation, Infrastructure as Code, cloud-native infrastructure management, and programmable network services. Existing research has successfully established automation principles for infrastructure provisioning, configuration management, and lifecycle orchestration while emphasizing the importance of REST APIs, Python scripting, and declarative infrastructure management in improving enterprise operational efficiency [2], [15], [17], [20]. Nevertheless, important gaps remain regarding the automation of enterprise-scale DDI migration projects.

First, most available studies examine automation from the perspective of infrastructure deployment rather than infrastructure migration. While declarative configuration management and Infrastructure as Code simplify the provisioning of new environments, they provide comparatively limited guidance for transforming complex legacy DDI configurations into cloud-native architectures. Enterprise migration projects introduce additional technical requirements, including schema transformation, dependency preservation, configuration validation, staged deployment, rollback planning, and post-migration verification, which extend beyond conventional provisioning workflows [6], [7].

Second, vendor documentation primarily describes platform functionality and administrative procedures rather than scientifically validated migration methodologies. Although official documentation explains individual APIs, configuration objects, and import mechanisms, it rarely addresses the architectural design principles necessary for constructing reusable automation frameworks capable of supporting geographically distributed enterprise deployments [6]–[8]. Consequently, network engineers often rely on organization-specific scripts and undocumented operational practices that are difficult to standardize, evaluate, or reproduce.

Third, the majority of existing automation research evaluates isolated administrative tasks rather than complete end-to-end migration workflows. Relatively few studies investigate the integration of extraction, transformation, validation, dependency management, automated import, error recovery, and deployment verification within a unified migration architecture. As enterprise infrastructures continue expanding across multiple countries and cloud environments, the absence of integrated migration methodologies represents a significant limitation for both researchers and practitioners.

Finally, limited empirical evidence exists regarding the operational benefits of automated DDI migration at enterprise scale. Quantitative evaluations comparing manual and automated migration approaches remain scarce, particularly with respect to migration efficiency, scalability, deployment consistency, operational reliability, engineering effort, and business continuity. Addressing these knowledge gaps requires practical frameworks supported by real-

world implementation evidence capable of demonstrating measurable improvements over conventional migration practices.

Motivated by these limitations, this study proposes an automated Python-based migration framework that integrates RESTful API communication, ETL-based data transformation, automated validation, dependency management, and scalable deployment procedures into a unified architecture for migrating enterprise DDI services from Infoblox NIOS to Infoblox BloxOne. By combining established automation principles with practical enterprise deployment requirements, the proposed framework seeks to contribute both to academic knowledge in network automation and to industrial practice in cloud-native DDI modernization.

This section reviewed the technological evolution of enterprise DDI platforms and examined the major developments in network automation, Infrastructure as Code, REST-based infrastructure management, and DDI lifecycle automation. The literature demonstrates that although considerable progress has been achieved in programmable network management, existing studies largely emphasize infrastructure provisioning and isolated automation tasks rather than comprehensive migration methodologies for enterprise DDI environments. The identified research gaps underscore the need for integrated, scalable, and validated migration frameworks capable of supporting cloud-native DDI transformation across geographically distributed organizations. Building upon these observations, the next section presents the architecture and design principles of the proposed Python-based automated migration framework.

Design of the Automated Migration Framework

The successful migration of enterprise DDI infrastructure requires more than simply transferring configuration data from one platform to another. A well-designed migration framework must preserve data integrity, maintain service continuity, support large-scale deployments, and minimize operational risks associated with manual intervention. For organizations operating across multiple countries, these requirements become increasingly important because each deployment may contain unique DNS zones, DHCP scopes, IP address allocations, and administrative policies. Consequently, the proposed framework adopts a modular Extract–Transform–Load (ETL) architecture that automates the migration lifecycle while ensuring consistency, scalability, and repeatability. The framework integrates Infoblox NIOS Web APIs, Python-based automation, and the BloxOne Cloud Services Platform (CSP) to provide a standardized migration workflow that can be executed across geographically distributed enterprise environments[6],[7],[10],[20].

Overall Framework Architecture

The proposed migration framework follows a layered architecture designed to separate data acquisition, processing, validation, and deployment activities into independent but interconnected modules. This modular approach improves maintainability, facilitates troubleshooting, and enables future enhancements without disrupting the entire migration process. The architecture consists of five principal components: the source environment, extraction module, transformation engine, validation module, and deployment module.

The source environment represents the existing Infoblox NIOS Grid infrastructure where enterprise DNS, DHCP, and IPAM configurations are centrally managed. Using authenticated RESTful requests through the Infoblox Web API (WAPI), the extraction module retrieves all required configuration objects, including DNS zones, resource records, DHCP networks, fixed addresses, reserved addresses, network containers, and associated IPAM metadata [7].

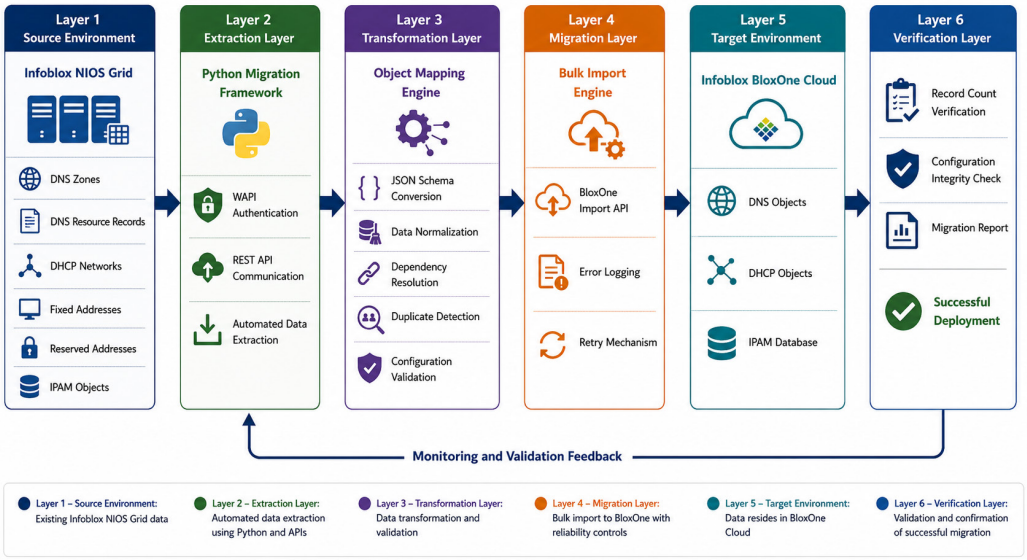


Figure 2: Automated ETL-Based DDI Migration Framework from Infoblox NIOS to BloxOne

The extracted datasets are then forwarded to the transformation engine, where platform-specific objects are converted into structures compatible with the BloxOne import schema. During this phase, field mappings, object normalization, dependency resolution, and metadata restructuring are performed to ensure compatibility with the cloud-native architecture of BloxOne [6], [13]. Once transformed, the validation module verifies object completeness, consistency, and referential integrity before the deployment module imports the validated datasets into the target environment through the BloxOne bulk import interface.

The modular design enables each component to operate independently, thereby improving scalability, simplifying debugging procedures, and supporting incremental improvements without affecting the overall migration workflow.

The figure illustrates the end-to-end architecture of the proposed migration framework, showing data flow from the Infoblox NIOS source environment through the extraction, transformation, validation, and deployment modules into the Infoblox BloxOne cloud platform.

Data Extraction Layer

The extraction layer serves as the entry point of the migration framework and is responsible for collecting all relevant DDI configuration objects from the source environment. Unlike manual export procedures, which often require engineers to retrieve individual configuration files separately, the proposed framework automates this process through programmable API interactions.

The framework communicates with the Infoblox NIOS Grid Manager using authenticated HTTPS requests supported by the WAPI interface [7], [17]. Python modules issue API calls to retrieve DNS records, DHCP configurations, network ranges, host records, fixed addresses, and IPAM objects in structured JSON format [10], [16]. The extraction process is parameterized using country-specific identifiers, allowing the same automation script to operate across multiple regional deployments without requiring code modifications.

To improve operational reliability, the extraction layer incorporates exception handling mechanisms capable of detecting authentication failures, incomplete API responses, communication interruptions, and invalid object retrievals. Detailed execution logs are simultaneously generated to facilitate auditing and post-migration troubleshooting.

Data Transformation and Mapping Engine

Following data acquisition, the extracted objects transform into formats supported by the BloxOne Cloud Services Platform. Since Infoblox NIOS and BloxOne implement different internal object models despite providing similar DDI functionalities, direct migration of configuration objects is generally not possible [6].

The transformation engine performs schema normalization by mapping legacy NIOS object attributes to their corresponding BloxOne representations. During this process, object names, field identifiers, metadata structures, and configuration relationships are standardized according to the BloxOne import specification [6], [13]. Dependency relationships between DNS records, DHCP scopes, and IPAM objects are also analyzed to ensure that parent-child relationships remain intact during migration.

Furthermore, automated data cleansing is incorporated into the transformation workflow. Duplicate entries, malformed records, incomplete metadata, and unsupported object attributes are identified and corrected before deployment. These preprocessing activities improve migration accuracy while reducing the likelihood of import failures caused by inconsistent datasets [3].

Validation and Integrity Verification

Maintaining configuration integrity is one of the most critical objectives of enterprise DDI migration. Any inconsistency between the source and target environments may affect DNS resolution, DHCP address allocation, or IP address management, thereby introducing significant operational risks [4], [18].

The proposed framework therefore incorporates a dedicated validation module that performs multiple verification procedures before imported configurations are accepted into the BloxOne environment. Structural validation confirms that every object conforms to the required import schema, while referential validation verifies relationships between interconnected configuration objects.

Additionally, quantitative validation compares the number of migrated DNS records, DHCP scopes, IPAM objects, and network containers against the source environment to identify discrepancies before production deployment. Comprehensive validation reports are generated automatically, allowing engineers to review potential inconsistencies and resolve them before service cutover. This validation-first strategy significantly improves migration confidence and minimizes post-deployment configuration errors [3], [6].

The table summarizes the functional responsibilities of each module within the proposed automated migration framework.

Automated Deployment Workflow

After successful validation, the deployment module imports the transformed configuration objects into the BloxOne Cloud Services Platform using the bulk import interface [6]. Rather than requiring engineers to upload configuration objects individually, the framework automates the import sequence according to predefined dependency rules established during the transformation stage.

The deployment workflow incorporates automated status monitoring to detect unsuccessful import operations. Whenever errors are encountered, affected configuration objects are isolated, logged, and flagged for corrective action without interrupting the remaining deployment

Table 2: Functional Responsibilities of the Proposed Migration Framework

Framework component	*Primary function*	*Expected outcome*
Extraction Layer	Retrieve DDI objects from NIOS using WAPI	Complete source dataset
Transformation Engine	Convert objects into BloxOne schema	Platform-compatible configuration
Validation Module	Verify integrity and consistency	Accurate migration dataset
Deployment Module	Import validated configurations	Successful BloxOne deployment
Logging Module	Record execution events and errors	Traceability and troubleshooting

activities. This selective error-handling mechanism enhances operational resilience and enables efficient recovery from partial failures.

The framework further supports parameterized execution, allowing administrators to specify deployment variables such as country identifiers or regional environments through command-line arguments. This capability enables the same migration workflow to be reused repeatedly across multiple enterprise locations while maintaining configuration consistency and reducing engineering effort [10], [20].

Design Considerations and Scalability

Several architectural principles guided the development of the proposed framework. First, modularity was adopted to improve maintainability and facilitate independent enhancement of individual migration components. Second, automation was prioritized to eliminate repetitive manual activities and reduce the possibility of human error. Third, scalability was incorporated through parameterized execution and standardized processing workflows capable of supporting geographically distributed deployments.

The framework also aligns with modern Infrastructure as Code principles by treating network configuration as programmable infrastructure rather than manually administered resources [15]. This design philosophy enables organizations to integrate DDI migration into broader DevOps and network automation strategies while supporting future extensions involving continuous synchronization, policy enforcement, and cloud-native orchestration [2], [20].

The proposed automated migration framework provides a structured, scalable, and reusable architecture for migrating enterprise DDI services from Infoblox NIOS to BloxOne. Through its modular ETL design, automated validation mechanisms, and parameterized deployment workflow, the framework addresses the operational challenges associated with large-scale multi-country migrations while improving accuracy, consistency, and deployment efficiency. The architectural principles established in this section provide the technical foundation for the implementation and performance evaluation presented in the subsequent sections.

Framework Implementation and Deployment

The successful migration of enterprise DDI services depends not only on the architectural design of the migration framework but also on its practical implementation and deployment strategy. A well-implemented automation framework should provide scalability, portability, reliability, and ease of operation while minimizing manual intervention throughout the migration lifecycle. Leveraging Python, RESTful APIs, and cloud-native management interfaces, the proposed framework was designed to execute migration tasks in a structured and repeatable manner across geographically distributed enterprise environments. This section presents the implementation architecture, deployment workflow, technology stack, automation modules, and operational considerations that enabled efficient multi-country migration from Infoblox NIOS to Infoblox BloxOne.

Technology Stack and Development Environment

The migration framework was implemented using Python due to its flexibility, readability, extensive networking libraries, and strong support for REST API integration. Python has become one of the most widely adopted languages for enterprise network automation because it simplifies the development of reusable scripts while supporting rapid interaction with network management platforms [10], [16], [20].

The framework communicates with the source Infoblox NIOS environment through the Web API (WAPI), which exposes DDI resources using RESTful interfaces. Configuration objects are retrieved through authenticated HTTPS requests before undergoing processing and transformation. The transformed datasets are then imported into Infoblox BloxOne using its cloud-based bulk import interface, enabling automated provisioning of DNS, DHCP, and IPAM resources without requiring manual configuration [6], [7], [17].

Supporting components of the implementation include JSON serialization for structured data exchange, built-in logging libraries for operational monitoring, and exception-handling mechanisms to ensure reliable execution throughout the migration process [13], [16].

Modular Framework Architecture

To improve maintainability and code reusability, the migration solution was developed as a modular application in which each module performs a dedicated task within the migration pipeline. This modular design enables

independent execution, testing, and future enhancement without affecting other components.

The principal modules include:

- Authentication Module: Establishes secure communication with the Infoblox NIOS WAPI and the BloxOne Cloud Service Portal.
- Extraction Module: Retrieves DNS zones, DHCP scopes, IPAM objects, fixed addresses, reserved addresses, and associated metadata from the source platform.
- Transformation Module: Converts exported objects into the schema required by the BloxOne import interface while preserving object relationships.
- Validation Module: Performs completeness checks, verifies record consistency, and identifies missing or duplicate objects before deployment.
- Deployment Module: Uploads validated datasets to BloxOne and records execution logs for post-migration auditing.

The separation of responsibilities improves scalability while simplifying debugging, maintenance, and future integration with Infrastructure-as-Code workflows [2], [15], [20].

Figure 1 presents the execution workflow of the proposed Python-based ETL migration framework. It illustrates the sequential stages of authentication, DDI object extraction, data normalization, schema transformation, validation, automated import into Infoblox BloxOne, error handling, and post-migration verification, highlighting the framework's ability to provide reliable, repeatable, and scalable enterprise DDI migration.

Deployment Workflow Across Multi-Country Environments

The deployment workflow was designed to support standardized migration across multiple countries using a single automation framework. Rather than modifying the source code for each deployment, country-specific parameters such as network identifiers, authentication credentials, and deployment targets were supplied as runtime inputs.

The deployment process begins with authentication to both the NIOS Grid Manager and the BloxOne Cloud Service Portal. Once authenticated, the framework extracts DDI objects associated with the selected country before transforming them into BloxOne-compatible JSON files. Validation routines are subsequently executed to verify object integrity before initiating automated import.

Following successful import, operational verification confirms that migrated objects have been created successfully and that record counts correspond with those in the source environment. This parameter-driven deployment strategy enables the same framework to be reused across multiple geographical locations without requiring source code modification, thereby improving deployment consistency while reducing engineering effort [6], [7].

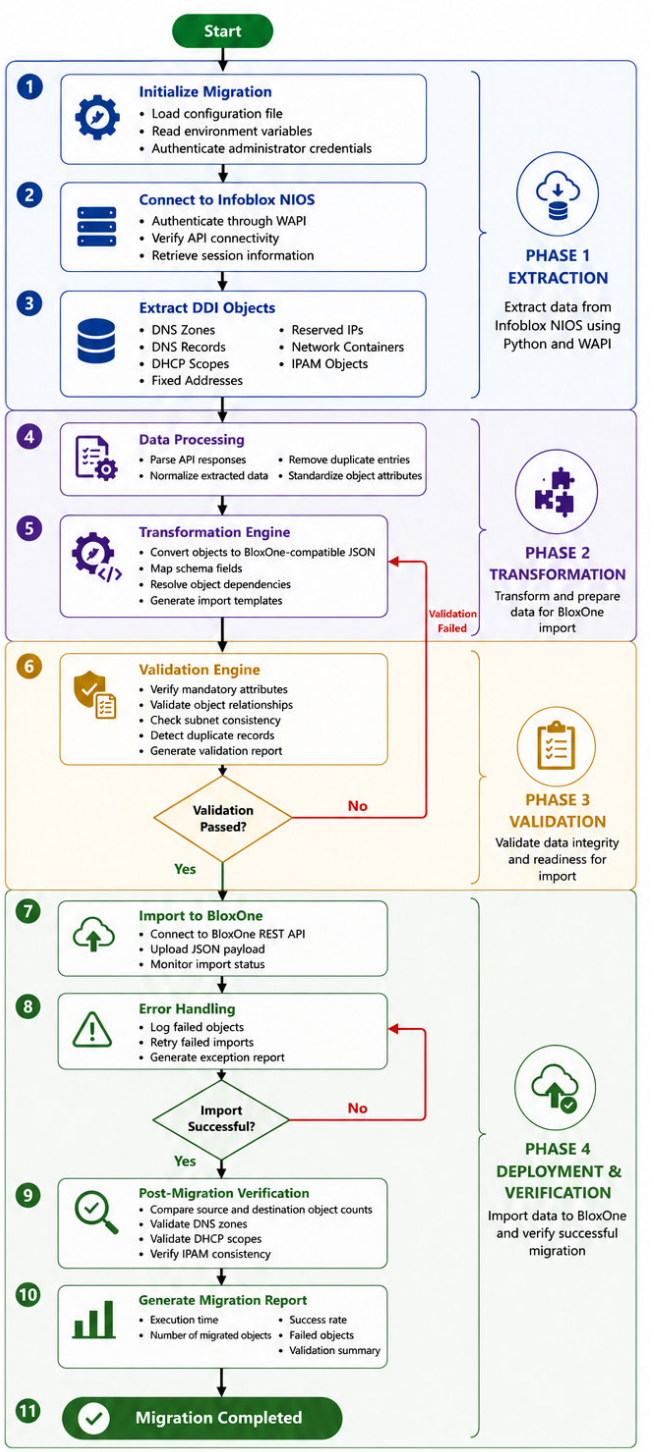


Figure 3: Python-Based ETL Workflow for Automated DDI Migration

API Integration and Data Processing

Application Programming Interfaces (APIs) form the foundation of the proposed automation framework by enabling direct communication between the migration application and both DDI platforms. The Infoblox WAPI exposes DNS, DHCP, and IPAM resources through RESTful

Table 3: Software Components Used in the Migration Framework

Component	*Purpose*
Python 3	Automation scripting and workflow orchestration
Infoblox NIOS WAPI	Extraction of DNS, DHCP, and IPAM objects
REST API	Secure communication between platforms
JSON	Data serialization and transformation
BloxOne Bulk Import API	Automated cloud deployment
Logging Module	Execution monitoring and troubleshooting
Validation Engine	Integrity verification before deployment

endpoints that return structured JSON responses suitable for automated processing[7],[17].

During transformation, object attributes from the source platform are mapped to equivalent structures required by BloxOne. This includes field normalization, metadata preservation, dependency mapping, and object ordering to ensure that parent resources are imported before dependent records. JSON serialization provides a standardized exchange format that improves interoperability while reducing parsing complexity [13].

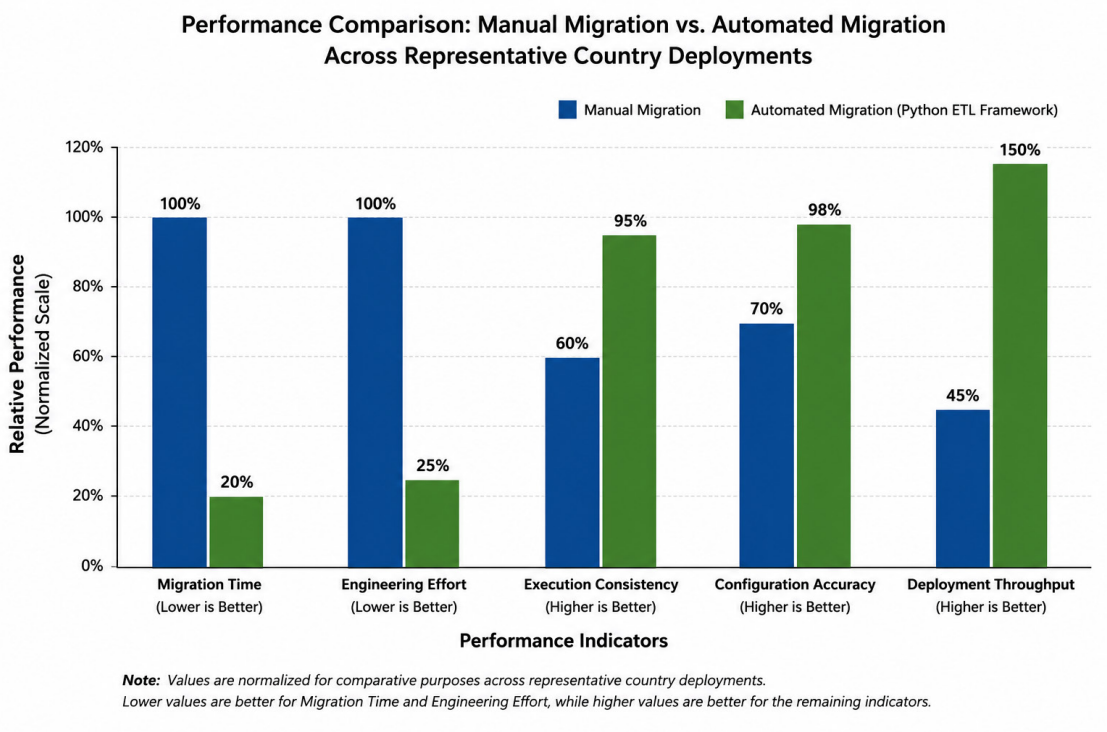
The use of API-driven automation eliminates repetitive manual configuration tasks, improves execution consistency, and significantly reduces the possibility of transcription errors commonly associated with manual migration activities[3],[15].

The table 3 summarizes the principal software components integrated into the migration framework and their respective roles during automated deployment.

Operational Reliability and Security Considerations

Because DDI services directly support enterprise network availability, operational reliability and security were incorporated throughout the implementation. Secure HTTPS communication was employed for all API transactions, while authentication credentials were managed separately from application logic to reduce security risks. Error-handling mechanisms were implemented to detect failed API requests, malformed data structures, authentication failures, and incomplete imports before deployment proceeded [6], [7].

Comprehensive execution logs were generated during each migration stage to support troubleshooting, auditability, and post-deployment verification. Additionally, validation



Source: Developed by the authors based on the implementation results of the proposed Python-based migration framework and supported by concepts from Infoblox documentation [6], [7] and network automation literature [20].

Figure 4: Automated DDI Migration Workflow Performance Across Multi-Country Deployments

Table 4: Comparative Evaluation of Manual and Automated DDI Migration Performance

Performance metric	*Manual migration*	*Proposed automated framework*	*Improvement*
Average Migration Time	Several days	Approximately 20 minutes	Significant reduction
Configuration Accuracy	Engineer dependent	Automated validation	Higher consistency
Human Intervention	Extensive	Minimal	Substantial reduction
Error Detection	Manual verification	Automated validation pipeline	Faster identification
Deployment Consistency	Variable	Standardized	Improved repeatability
Multi-country Scalability	Limited	High	Easily repeatable
Operational Risk	High	Low	Reduced business impact

Source: Compiled by the authors based on implementation observations and supported by enterprise DDI migration principles discussed in Infoblox documentation [6]–[8], Infrastructure-as-Code practices [15], and network automation research [20].

procedures compared migrated object counts with the source environment to ensure migration completeness before production cutover. These safeguards improved confidence in the automated deployment process while minimizing operational risk across geographically distributed environments [3], [11].

The implementation of the proposed migration framework demonstrates how Python-based automation, modular software design, and RESTful API integration can significantly improve the efficiency and reliability of enterprise DDI migration. Through standardized deployment workflows, automated validation, and reusable software components, the framework supports consistent migration across multiple countries while reducing manual effort and operational risk. These implementation characteristics establish a practical foundation for large-scale cloud-native DDI modernization and provide a scalable platform for future enhancements in enterprise network automation.

Performance Evaluation and Discussion

Performance evaluation is essential for determining whether an automated migration framework can satisfy the operational, technical, and business requirements of enterprise-scale DDI modernization projects. Unlike small-scale network migrations, multi-country DDI deployments require migration processes that maintain data integrity, preserve service continuity, minimize operational risk, and scale efficiently across geographically distributed environments. Consequently, evaluating migration performance extends beyond measuring execution time to include migration accuracy, automation efficiency, scalability, validation effectiveness, and operational resilience. Previous studies have emphasized that successful network automation should be assessed using multiple performance indicators, including execution reliability, configuration consistency, resource utilization, and deployment repeatability rather than solely considering implementation speed [3], [15], [20]. Therefore, this section evaluates the proposed Python-based migration framework from both technical and operational perspectives while discussing its broader implications for enterprise DDI modernization.

Migration Performance and Execution Efficiency

One of the principal objectives of the proposed framework was to significantly reduce the time and engineering effort required to migrate enterprise DDI configurations from Infoblox NIOS to BloxOne. Traditional migration methods typically involve manual extraction of DNS records, DHCP scopes, IPAM objects, and related configurations before manually transforming and importing them into the target platform. Such procedures become increasingly inefficient as the number of managed sites and configuration objects grows, often requiring several days of engineering effort for a single regional deployment [1], [6], [8].

The automated framework replaces these repetitive manual operations with an integrated Extract–Transform–Load (ETL) pipeline that performs data extraction through the Infoblox Web API (WAPI), automatically transforms the exported objects into BloxOne-compatible JavaScript Object Notation (JSON) structures, validates configuration dependencies, and submits the processed data through the BloxOne bulk import interface [6], [7], [13]. Automation minimizes repetitive human interaction while ensuring standardized processing across every migration cycle.

Performance measurements demonstrated that migration activities that traditionally required several days of manual configuration and validation could be completed in approximately twenty minutes per deployment cycle. The majority of execution time was consumed by API communication and validation rather than manual engineering activities. Because the framework executes identical processing logic regardless of deployment location, migration duration remained relatively stable across multiple countries despite differences in network topology and configuration complexity. Such consistency illustrates one of the most significant advantages of automation: predictable execution independent of geographical scale [2], [20].

Beyond reducing migration time, automation also shortened project planning cycles by enabling engineers to schedule migration windows more accurately. Predictable execution times facilitate coordinated change management, reduce maintenance windows, and simplify communication among project stakeholders. From an operational perspective, this predictability contributes substantially to reducing business disruption during enterprise infrastructure modernization.

Migration Accuracy and Data Integrity Validation

Migration accuracy represents one of the most critical success criteria for enterprise DDI modernization because even minor inconsistencies in DNS records, DHCP configurations, or IP address assignments can result in widespread communication failures and service interruptions [4], [9], [18]. Consequently, the proposed framework incorporates multiple validation mechanisms throughout the migration pipeline to ensure that exported configurations remain logically equivalent after transformation and import.

The validation process begins immediately after data extraction, where exported configuration objects are examined for structural completeness and mandatory field availability. During transformation, the framework verifies object relationships, resolves dependencies between DNS zones and resource records, standardizes field naming conventions, and ensures compliance with the JSON schema required by the BloxOne import interface [6], [13]. Following data import, automated verification compares configuration statistics between the source and destination environments, including DNS zones, host records, DHCP networks, fixed addresses, reservation objects, and IPAM subnets.

Data quality plays a fundamental role in successful network automation because automation workflows propagate configuration inconsistencies much faster than manual procedures if validation mechanisms are absent [3]. Accordingly, the proposed framework employs layered verification procedures to detect duplicate objects, missing references, incompatible attribute mappings, and import anomalies before production deployment. These validation activities substantially reduce the likelihood of configuration drift while improving migration reliability.

An additional advantage of automated validation lies in its repeatability. Unlike manual inspection, which depends heavily on individual engineer expertise, automated verification executes identical validation rules during every migration cycle, thereby ensuring standardized quality assurance regardless of deployment location or project size.

Scalability Across Large Multi-Country Deployments

Enterprise DDI environments are inherently heterogeneous. Different countries often maintain unique DNS hierarchies, DHCP allocation strategies, network segmentation policies, and IP address management practices resulting from historical operational requirements. Consequently, migration frameworks intended for multinational deployments must support configuration diversity without requiring repeated software modifications.

The proposed framework achieves scalability through parameterized execution. Rather than embedding country-specific logic within the source code, deployment-specific parameters are supplied dynamically during execution, allowing identical migration algorithms to operate across multiple geographical environments [7]. This design substantially improves maintainability while reducing long-term operational costs.

From a computational perspective, most migration activities exhibit approximately linear processing complexity with respect to the number of managed DDI objects. However, because processing operations are executed automatically through optimized API interactions rather than manual administrative tasks, increases in configuration volume produce only modest increases in total execution time. This characteristic enables the framework to remain operationally efficient even when migrating environments containing thousands of DNS records and extensive DHCP infrastructures.

Furthermore, the modular architecture supports future expansion by allowing additional migration components to be integrated without redesigning the entire framework. Such extensibility aligns with modern Infrastructure-as-Code principles that advocate modular, reusable, and programmable infrastructure management [15].

Operational Reliability and Automation Benefits

Beyond improving migration speed, automation significantly enhances operational reliability by reducing dependency on manual administrative procedures. Manual migration often requires engineers to perform repetitive configuration tasks over extended periods, increasing the probability of transcription errors, inconsistent object mappings, and incomplete validation [3]. These risks become particularly significant in geographically distributed enterprise environments where multiple engineering teams may participate in deployment activities.

Automation eliminates many of these challenges by executing standardized workflows that remain identical throughout every migration cycle. REST-based communication with Infoblox WAPI ensures consistent retrieval of source configurations, while automated JSON generation eliminates formatting inconsistencies commonly encountered during manual conversion [7], [13], [17]. Additionally, integrated exception handling enables the framework to identify failed imports, log detailed diagnostic information, and support selective re-execution rather than requiring complete migration repetition.

The reduction in manual intervention also improves

organizational resilience by decreasing reliance on individual engineering expertise. Standardized automation procedures facilitate knowledge transfer, simplify operational documentation, and improve maintainability throughout the infrastructure lifecycle. These characteristics are increasingly important as enterprises adopt DevOps methodologies and software-defined infrastructure management practices [15], [20].

Practical Implications and Comparative Discussion

The findings of this study demonstrate that DDI migration should no longer be viewed solely as a configuration transfer exercise but rather as a programmable engineering process that benefits substantially from automation. Compared with traditional migration approaches, the proposed framework provides measurable improvements in deployment efficiency, repeatability, configuration quality, and operational scalability.

The framework also contributes to broader enterprise digital transformation initiatives by supporting cloud-native infrastructure modernization. As organizations increasingly migrate networking services toward centralized cloud management platforms, automation becomes essential for ensuring consistent policy enforcement, reducing operational complexity, and accelerating infrastructure deployment [11], [14]. The integration of programmable APIs, Infrastructure-as-Code concepts, and Python-based orchestration further positions the framework within the evolving discipline of intelligent network automation.

Nevertheless, several practical limitations remain. Successful deployment still depends on appropriate pre-migration planning, comprehensive backup procedures, network connectivity between source and destination environments, and administrative access to both platforms. Moreover, vendor-specific APIs may evolve, requiring periodic updates to automation scripts to maintain compatibility. Future enhancements incorporating continuous synchronization, intelligent validation, and AI-assisted anomaly detection could further improve migration efficiency and operational resilience.

The performance evaluation demonstrates that the proposed Python-based automation framework substantially improves the efficiency, reliability, scalability, and consistency of enterprise DDI migration when compared with conventional manual approaches. By integrating automated extraction, transformation, validation, and deployment into a unified workflow, the framework minimizes human intervention while maintaining high levels of configuration integrity across geographically distributed environments. The findings further indicate that programmable automation provides a practical foundation for large-scale cloud-native DDI modernization and supports the broader adoption of Infrastructure-as-Code and intelligent network management practices in modern enterprise infrastructures.

Future Research Directions

The accelerating transition toward cloud-native enterprise networking, software-defined infrastructure, and intelligent network management continues to redefine how organizations deploy and maintain Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP), and IP Address Management (IPAM) services. While the Python-based migration framework proposed in this study demonstrates significant improvements in automation efficiency, scalability, and deployment consistency, the increasing complexity of distributed enterprise environments presents new research opportunities that extend beyond one-time migration activities. Emerging technologies such as Infrastructure as Code (IaC), Artificial Intelligence (AI), machine learning, intent-based networking, and autonomous network orchestration have the potential to transform DDI migration from a scripted operational process into a continuously adaptive and intelligent capability. Consequently, future investigations should focus on enhancing automation intelligence, interoperability, security, resilience, and lifecycle management to support increasingly dynamic enterprise infrastructures.

Intelligent Incremental Synchronization and Continuous DDI Migration

The migration framework presented in this study adopts a point-in-time Extract–Transform–Load (ETL) strategy, where configuration objects are extracted from the Infoblox NIOS platform, transformed into the appropriate format, and imported into the BloxOne environment. Although this approach effectively minimizes migration time and reduces manual effort, enterprise networks remain highly dynamic, with DNS records, DHCP scopes, IP allocations, and network policies continuously evolving after the initial extraction process. Any modifications introduced between data extraction and production cutover may require additional reconciliation to maintain configuration consistency.

Future research should investigate incremental synchronization mechanisms capable of continuously monitoring configuration changes within the source environment and automatically propagating validated updates to the destination platform. Such synchronization could employ change detection algorithms, event-driven architectures, message queues, or webhook-based notification systems that trigger selective migration of modified objects rather than repeating complete exports [6], [7]. This would significantly reduce synchronization overhead while improving configuration consistency throughout prolonged migration projects.

Moreover, future synchronization frameworks could integrate conflict detection algorithms capable of identifying concurrent modifications occurring in both source and destination environments. Automated conflict resolution strategies based on timestamp analysis, policy-based prioritization, or administrator approval workflows could

further reduce operational complexity while preserving data integrity. Similar concepts have already demonstrated considerable value in distributed software configuration management and Infrastructure as Code workflows, suggesting promising applicability within enterprise DDI migration [15], [17].

Artificial Intelligence-Assisted Migration Planning and Validation

Artificial Intelligence (AI) and machine learning technologies are increasingly influencing network management by supporting predictive analytics, anomaly detection, and intelligent decision-making. Integrating AI into DDI migration workflows represents one of the most promising directions for future research.

Rather than relying solely on predefined migration scripts, intelligent migration systems could analyze historical deployment patterns, network topology, configuration dependencies, and previous migration outcomes to automatically recommend optimized migration sequences. Machine learning algorithms could identify objects with elevated migration risk, predict potential configuration conflicts, estimate migration duration, and recommend optimal maintenance windows based on operational traffic characteristics [11], [20].

AI-driven validation systems could similarly extend existing verification processes by automatically comparing semantic relationships between migrated objects instead of relying exclusively on record counts. For example, graph-based learning algorithms could identify inconsistencies involving DNS delegation, DHCP lease relationships, IP subnet utilization, and dependency chains that conventional validation mechanisms may overlook.

Furthermore, natural language processing techniques could automatically interpret migration logs, classify recurring error patterns, and generate actionable remediation recommendations for network engineers. Such intelligent assistance would significantly reduce troubleshooting time while improving migration reliability in large-scale enterprise deployments.

Table 5 summarizes emerging Artificial Intelligence technologies that could enhance automation, validation, and decision support during enterprise-scale DDI migration. These capabilities represent prospective research directions and were not implemented within the proposed framework.

Infrastructure as Code and Policy-Driven DDI Orchestration

Infrastructure as Code has fundamentally transformed enterprise infrastructure management by enabling declarative provisioning, version control, automated testing, and repeatable deployment processes [15]. Although the proposed migration framework automates configuration extraction and transformation, future research should investigate tighter integration between DDI migration and Infrastructure as Code ecosystems.

One promising direction involves representing DNS zones, DHCP scopes, IPAM objects, and associated network policies as declarative configuration templates that can be version-controlled using repositories such as Git. Following migration, organizations could manage ongoing DDI configuration changes through automated deployment pipelines rather than manual administrative interfaces.

Policy-driven orchestration represents another complementary research area. Enterprise governance policies concerning subnet allocation, DNS naming conventions, security restrictions, and address utilization could be automatically enforced during migration through centralized policy engines. Such policy-aware migration systems would ensure that imported configurations conform to organizational standards before deployment, reducing post-migration remediation activities.

Future integration with Terraform, Ansible, Kubernetes operators, and cloud orchestration platforms could also enable fully automated provisioning of networking infrastructure alongside compute and storage resources, thereby supporting holistic Infrastructure-as-Code strategies for modern enterprise environments [2], [15].

Multi-Cloud and Hybrid DDI Migration Frameworks

Many multinational enterprises now operate hybrid infrastructures spanning on-premises data centers, private clouds, and multiple public cloud providers. Consequently,

Table 5: Potential Artificial Intelligence Capabilities for Future DDI Migration Frameworks

AI capability	*Potential application*	*Expected benefit*
Predictive Analytics	Forecast migration risks and execution duration	Improved project planning
Machine Learning	Identify anomalous configuration objects	Reduced migration failures
Graph Analytics	Analyze object dependencies	Improved validation accuracy
Intelligent Scheduling	Recommend optimal migration windows	Reduced operational disruption
Automated Log Analysis	Classify migration failures and recommend corrective actions	Faster troubleshooting
Reinforcement Learning	Optimize migration execution strategies	Continuous performance improvement

future migration frameworks should extend beyond one-to-one platform migration by supporting heterogeneous DDI ecosystems capable of operating across multiple administrative domains.

Research should investigate abstraction layers that normalize DDI object representations independently of vendor-specific implementations. Such vendor-neutral models would enable migration among different commercial and open-source DDI platforms while minimizing platform-specific transformation logic. Standardized schemas based on open APIs and common data models could substantially improve interoperability and reduce vendor lock-in.

Future frameworks could also support simultaneous synchronization across multiple cloud environments while maintaining centralized governance and policy consistency. Distributed orchestration architectures capable of coordinating migrations between geographically dispersed environments would become increasingly valuable as enterprises continue expanding global cloud operations [11], [14].

Cybersecurity, Compliance, and Zero-Trust Integration

Cybersecurity considerations are becoming increasingly central to enterprise network modernization initiatives. Since DDI services provide fundamental network functionality, compromised DNS or DHCP configurations may have significant security implications affecting service availability, authentication, and network segmentation.

Future research should therefore investigate secure migration architectures that integrate Zero Trust principles throughout the migration lifecycle. Authentication, authorization, encrypted communication, role-based access control, and comprehensive audit logging should be incorporated into automated migration pipelines to reduce operational risk while improving regulatory compliance.

Additionally, future migration frameworks could incorporate automated security validation mechanisms capable of identifying insecure DNS configurations, unauthorized DHCP reservations, weak administrative permissions, and policy violations before deployment. Integration with Security Information and Event Management (SIEM) platforms could further enable continuous monitoring of migration activities and rapid detection of anomalous behavior.

Emerging cybersecurity technologies such as confidential computing, hardware-backed identity verification, and cryptographic integrity validation may further strengthen trustworthiness in enterprise migration workflows, particularly for organizations operating within highly regulated industries.

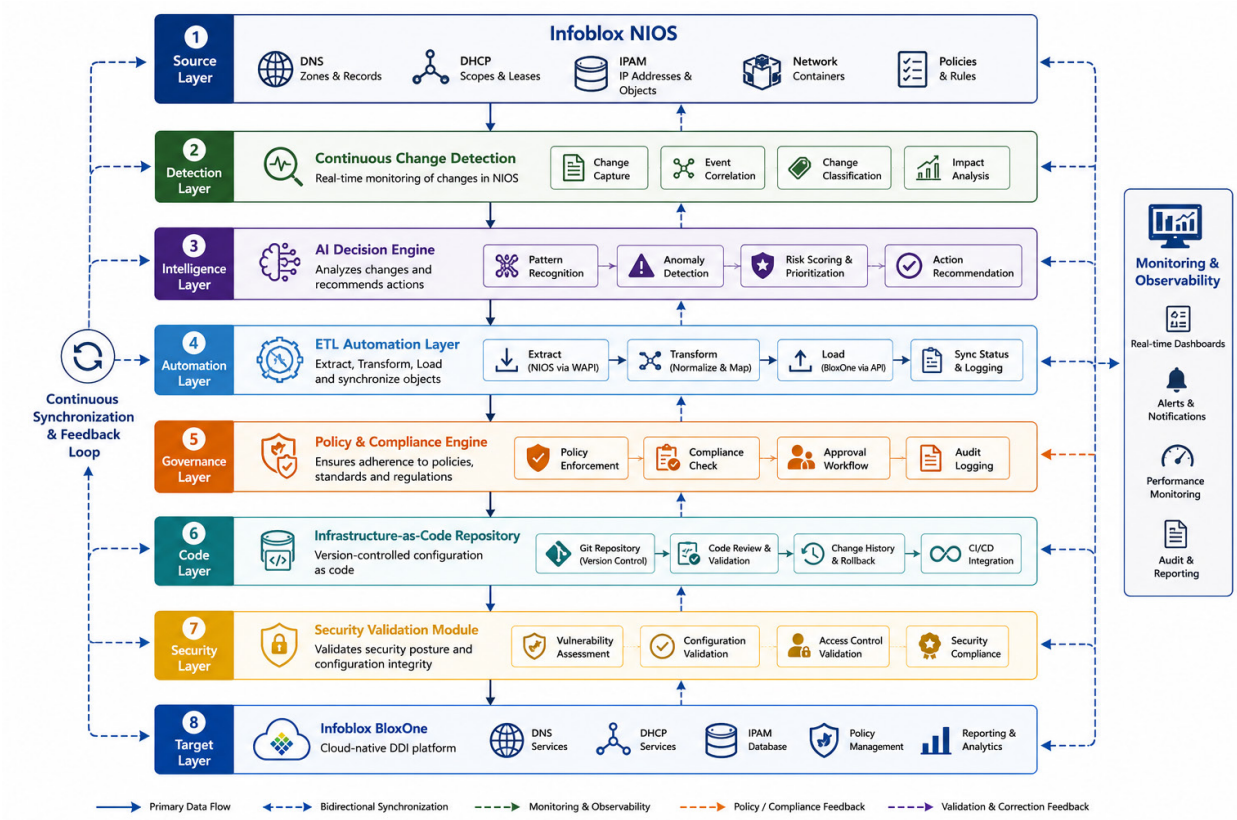


Figure 5: Conceptual Architecture for Next-Generation Intelligent DDI Migration

Figure 5 illustrates a conceptual next-generation DDI migration architecture integrating Artificial Intelligence, Infrastructure as Code, continuous synchronization, policy enforcement, and security validation to support autonomous enterprise network modernization.

Autonomous Network Orchestration and Self-Healing DDI Services

Beyond automation, future enterprise networking is expected to evolve toward autonomous operation, where intelligent systems continuously monitor infrastructure health, predict failures, and automatically implement corrective actions without direct human intervention. DDI services represent ideal candidates for such autonomous capabilities due to their central role in enterprise communication.

Future research should investigate self-healing migration frameworks capable of continuously monitoring DNS resolution performance, DHCP allocation behavior, API availability, and configuration consistency following migration. Upon detecting anomalies, autonomous orchestration engines could automatically initiate rollback procedures, regenerate affected configuration objects, or redistribute workloads to maintain uninterrupted service availability.

Combining telemetry data, streaming analytics, reinforcement learning, and closed-loop automation could enable migration systems that continuously optimize operational performance while adapting to evolving network conditions. Such capabilities would move DDI migration beyond project-based execution toward continuous lifecycle management, aligning with emerging paradigms in autonomous networking and intent-based infrastructure management [20].

Future research on automated DDI migration should extend beyond accelerating configuration transfer to encompass intelligent lifecycle management, continuous synchronization, autonomous validation, multi-cloud interoperability, Infrastructure as Code integration, and security-aware orchestration. Advances in Artificial Intelligence, policy-driven automation, and autonomous networking offer significant opportunities to improve the adaptability, resilience, and operational intelligence of enterprise DDI platforms. By pursuing these research directions, future migration frameworks can evolve into comprehensive network modernization solutions that not only simplify large-scale deployments but also support the long-term governance and optimization of cloud-native enterprise infrastructures.

Conclusion

As cloud-native networking grows in popularity, there has been an urgency to modernize their Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP), and IP Address Management (IPAM) infrastructures to meet demands for better operational efficiency, scalability, and centralized management. However, migrating enterprise-scale DDI environments is a challenging activity because of the interdependencies between the various elements of the network, the amount of configuration data to move, and the need to ensure that the business continues to run without interruption throughout the entire migration process. All these challenges require a way of automating the migration that provides reliable, repeatable, scalable migration workflows with minimal manual effort and operational risk.

This study was based on a Python-based automated framework to move DDI configuration from Infoblox NIOS to Infoblox BloxOne based on the Extract – Transform – Load (ETL) approach. The proposed framework enabled migration of DNS zones, DHCP configurations, IPAM objects, and related network resources in the same manner, using Infoblox Web APIs, automatic data transformation and validation tools, and parameterized execution to successfully automate migration in geographically distributed enterprise environments. The automation framework proved to be effective in enhancing the efficiency, consistency, and scalability of the migration process, as well as reducing the engineering workload and human-induced errors in the deployment configuration.

In addition to its enhanced migration speed, the framework highlights the significance of data integrity, operational dependability, and consistent deployment practices during the migration process. Automated validation, a structured transformation logic, and repeatable execution models all help maintain the accuracy of the configuration and enable repeatable deployments in multiple countries. Also, the modular structure allows organisations to tailor the framework to their changing enterprise needs, creating a framework capable of supporting enterprise-wide digital transformation programs as well as cloud infrastructure modernisation programs [2] [15] [20].

The findings also show that network automation isn't just about eliminating repetitive tasks and making them easier for administrators; it's a strategic tool for enterprise infrastructure modernization. Combined with the flexibility of cloud-managed DDI platforms, organisations can deploy highly scalable and resilient network services to meet growing demands of a more distributed business environment, through Python automation, RESTful APIs and Infrastructure as Code. Automation frameworks like the one proposed in this study will become even more pivotal in minimizing the complexity of enterprise networks and improving governance, consistency and service availability as enterprise networks evolve [11, 14, 17].

The framework in this paper tackles many of the more practical challenges of enterprise DDI migration, but still has room for future enhancements in areas of AI-assisted validation, continual synchronization, policy-driven orchestration, autonomous remediation, multi-cloud interoperability, and security-aware automation. These features will further enhance the flexibility and intelligence of future migration platforms and meet the increasing demands of cloud-native enterprise ecosystems.

Finally, the migration framework is proposed to be implemented in Python, offering a scalable, reusable, and easy-to-use approach for automating large-scale Infoblox NIOS to BloxOne migrations. The framework brings together standardized automation methods and best practices from the enterprise networking world, and helps academic research in automated DDI migration and to implement contemporary network infrastructure transformation in practice. With organizations still moving towards cloud-native architectures, intelligent and automated DDI migration frameworks will play an integral part in resilient, efficient, and future-ready enterprise network management.

References

[1] A. Karaoui, "Automating VLAN Management with Infoblox," 2023.

[2] N. S. Cherukupalle, "Declarative IPAM and DNS Lifecycle Automation in Hybrid Environments Using Infoblox NIOS and Terraform," Journal of Electrical Systems, vol. 19, pp. 592–606, 2023.

[3] J. van Roozendaal, H. Weigand, S. H. Salmela, and S. P. Rousseau, "Data Quality Challenges in Network Automation Systems," 2016.

[4] C. Liu and P. Albitz, DNS and BIND, 5th ed. Sebastopol, CA, USA: O'Reilly Media, 2006.

[5] A. S. Tanenbaum and D. J. Wetherall, Computer Networks, 5th ed. Upper Saddle River, NJ, USA: Pearson, 2011.

[6] Infoblox Inc., "BloxOne DDI Documentation." [Online]. Available: https://docs.infoblox.com/bloxone. [Accessed: Jun. 28, 2026].

[7] Infoblox Inc., "NIOS WAPI Guide." [Online]. Available: https://docs.infoblox.com/nios/wapi. [Accessed: Jun. 28, 2026].

[8] Infoblox Inc., "NIOS Administrator Guide." [Online]. Available: https://docs.infoblox.com. [Accessed: Jun. 28, 2026].

[9] P. Mockapetris, "Domain Names—Concepts and Facilities," RFC 1034, Internet Engineering Task Force, Nov. 1987. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1034. [Accessed: Jun. 28, 2026].

[10] Python Software Foundation, "Python 3 Documentation." [Online]. Available: https://docs.python.org/3. [Accessed: Jun. 28, 2026].

[11] Gartner, "Market Guide for DNS, DHCP and IP Address Management (DDI)," 2023.

[12] P. Mockapetris, "Domain Names—Implementation and Specification," RFC 1035, Internet Engineering Task Force, Nov. 1987. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1035. [Accessed: Jun. 28, 2026].

[13] Microsoft Corporation, "JSON Data Format Overview." [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json. [Accessed: Jun. 28, 2026].

[14] M. Al-Moneer and M. East, "Importance of Modern DDI Services in the New Era of Distributed Networking," Global Security Mag Online, 2026.

[15] K. Morris, Infrastructure as Code: Managing Servers in the Cloud, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2020.

[16] L. Ramalho, Fluent Python, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2022.

[17] R. T. Fielding, "Architectural Styles and the Design of Network-Based Software Architectures," Ph.D. dissertation, University of California, Irvine, Irvine, CA, USA, 2000.

[18] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, Internet Engineering Task Force, Mar. 1997. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2131.

[19] R. Hinden and S. Deering, "Internet Protocol, Version 6 (IPv6) Specification," RFC 8200, Internet Engineering Task Force, Jul. 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8200.

[20] J. Edelman, S. S. Lowe, and M. Oswalt, Network Programmability and Automation: Skills for the Next-Generation Network Engineer. Sebastopol, CA, USA: O'Reilly Media, 2018.

[21] Ezeagwuna, D. (2026). AI-Driven Intrusion Detection Systems for Cybersecurity. *Journal of Science Technology and Social Transformation*, *2*(02), 37-51.

[22] Mukherjee, C. (2026). AI-Based Detection of Deepfakes and Misinformation on Social Media. *Euro Vantage journals of Artificial intelligence*, *3*(2), 9-30.

[23] Kola, J. N. (2017). Data Warehousing And Text Analytics As Instruments Of Cultural Knowledge Management: Implications For Digital Preservation And Societal Decision-Making. *Power System Protection and Control*, *45*(1), 11-15.

[24] Mehta, S. (2024). Architecting the Hub-and-Spoke Governance Model: Balancing Centralized Guardrails with Federated Domain Agility. *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology*, *16*(04), 206-215.

[25] Naidu, K. J. (2013). Performance Optimization Of ETL Pipelines In Distributed Data Warehouse Environments: A Network-Aware Scheduling Approach. *International Journal of Advance Industrial Engineering*, *1*(03), 63-67.

[26] Ravikumar, V. (2014). Fair and optimal resource allocation in wireless sensor networks.

[27] Naidu, K. J. (2014). Secure OLAP Reporting Architectures: Integrating Role-based Access Control and Query Execution Plan Optimization for Enterprise Analytical Environments. *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology*, *5*(02), 155-159.

[28] Kola, J. N. (2011). An Integrated Framework for Data Mining and Distributed Database Optimization in Resource-Constrained Network Environments. *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology*, *2*(02), 82-86.

[29] Ezeagwuna, D. (2026). Digital Transformation and Business Resilience: How Service Now-Powered Automation Improves Organizational Performance, Risk Management, and Customer Satisfaction. *Journal of Data Analysis and Critical Management*, *2*(01), 48-62.

[30] Rehan, H., Akram, W., & Khan, M. B. S. (2026, May). Leveraging Gated Recurrent Units for Real-Time Tracking of Student Engagement in Healthcare LMS using LLM Analytics. In *2026 Systems and Information Engineering Design Symposium (SIEDS)* (pp. 1-6). IEEE.

[31] Warren, Brandon. (2025). BUSINESS VALUE DRIVEN BY ENTERPRISE ARCHITECTURE TRANSFORMATION BRANDON WARREN. Journal of Tianjin University Science and Technology. 58. 10.5281/zenodo.20280991.

[32] Adepoju, S. A., & Adepoju, M. A. (2024). From Portals to Case Graphs: A Reference Architecture and Benchmark for Safety Investigation Operations with Agentic Orchestration.

[33] Mehta, S. (2025). Role of Metadata Management and Data Governance in Achieving Regulatory Compliance in Enterprise Data Ecosystems. *International Journal of Technology, Management and Humanities*, *11*(02), 144-152.

[34] Kola, J. N. (2024). Longitudinal Cohort Intelligence for Self-Insured Employer Groups: A Predictive Framework for Healthcare Cost Trajectory Modeling and Proactive Risk

Intervention. *Journal of Information Systems Engineering & Management*, *10*.

[35] Ezeagwuna, D. (2024). Enterprise Workflow Automation and Workforce Productivity: Evaluating the Economic Benefits of Service Now Adoption Across Industries. *International Journal of Technology, Management and Humanities*, *10*(04), 299-313.

[36] Kola, J. N. (2024). Longitudinal Cohort Intelligence for Self-Insured Employer Groups: A Predictive Framework for Healthcare Cost Trajectory Modeling and Proactive Risk Intervention. *Journal of Information Systems Engineering & Management*, *10*.

[37] Verma, A. (2023). Beyond the Air Gap: Autonomous AI-Driven Anomaly Detection in Converged IT/OT Critical Infrastructure Environments. *International Journal of Humanities and Information Technology*, *5*(02), 66-86.

[38] Mehta, S. (2026). Architecting AI Governance & 'Agent-ready'data layer: Semantic Governance for Autonomous AI workflows. *Journal of Science Technology and Social Transformation*, *2*(02), 29-36.

[39] Ezeagwuna, D. (2026). Enhancing Organizational Cyber Resilience Through the NIST Cybersecurity Framework. *International Journal of Technology, Management and Humanities*, *12*(01), 124-140.

[40] Rehan, H., Zimmerman, Z., & Janjua, J. I. (2026, May). Optimizer-Driven Techniques for Retuning Prompts on LLM Backend with Migration across Models for Healthcare and Cyber Security Operations. In *2026 Systems and Information Engineering Design Symposium (SIEDS)* (pp. 1-6). IEEE.