International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Cloud Migration Made Easy: A 5-Step Automation Framework

Prof. Manisha Patil[1], Supriya Kamthekar[2], Shon Gaikwad[3], Yash Lonkar[4], Akshaya Survase[5]

[1,2,3,4,5] *Computer Department, Trinity College of Engineering and Research, Pune, India*

---***---

Abstract - In the evolving landscape of software systems, the demand for seamless migration of applications from traditional local servers to cloud environments is rapidly increasing. However, this transition often involves a series of complex, manual, and repetitive tasks that can hinder scalability and productivity. This paper presents a streamlined approach aimed at minimizing the human effort involved in the migration of server-based applications to cloud infrastructure. The proposed system introduces an intelligent interface that allows users to initiate and manage the migration process through a minimal-input design. By leveraging automated scripting and remote execution, the system enables local environments to communicate with cloud-based virtual machines, facilitating the end-to-end setup, configuration, and hosting of applications with little to no manual intervention. The architecture focuses on reducing the operational overhead typically associated with cloud onboarding, offering a lightweight, user-centric model adaptable to various application types. This approach contributes to faster deployment cycles, improved efficiency, and an overall reduction in the complexity of cloud adoption processes.

Key Words: Cloud Migration, Infrastructure Automation, Remote Scripting, Application Hosting, Virtual Machines, User Interface Automation, Lightweight Framework, Deployment Simplification, CI/CD Pipelines, DevOps, IaaS (Infrastructure as a Service), Cloud-Native Architecture, Platform Abstraction, Virtualized Environments, SSH Automation, Script-Driven Deployment, Remote Infrastructure Management, Server Provisioning, Cloud Scalability, Web Application Hosting.

1. INTRODUCTION

Cloud computing has revolutionized the way applications are built, deployed, and managed, prompting a widespread shift from traditional server-based setups to flexible, scalable cloud infrastructures. Organizations now seek the benefits of reduced operational costs, greater agility, and enhanced global reach. However, migrating legacy or on-premise systems to the cloud continues to pose significant challenges. This difficulty arises from the disjointed nature of current migration workflows, the vast range of technologies involved, and the specialized knowledge required to handle cloud architecture and infrastructure automation effectively.

For many businesses especially small to mid-sized enterprises lacking dedicated DevOps teams the migration process is daunting. It typically involves multiple sequential steps, including setting up the environment, mapping infrastructure, packaging workloads, deployment, and rigorous testing. Each step demands careful manual configuration and validation, often resulting in delays, errors, or abandoned migration efforts due to the complexity and costs involved. This study aims to simplify this intricate process by proposing an intelligent and user-friendly migration framework. The objective is to ease the transition to cloud environments by reducing technical overhead, streamlining operations, and enabling broader access to cloud technologies through guided, efficient solutions.

2. RELATED WORKS

Cloud migration has become increasingly vital due to the scalability and cost-efficiency of cloud platforms, yet traditional methods remain heavily manual, involving multiple intricate phases like infrastructure setup and deployment configuration. While tools such as Terraform and Ansible offer automation, they often require significant expertise and are tailored for DevOps professionals, making them less accessible to general developers or small teams. Existing research, including model-based tools like CloudMIG and template-driven platforms like Stratus Cloud, have introduced automation but still demand complex configurations. Conversely, low-code and no-code platforms provide simplicity but often limit flexibility and control. This work proposes a middle-ground solution—a script-assisted, user-friendly automation framework that streamlines cloud migration by reducing manual effort while maintaining sufficient customization, targeting developers and project teams seeking efficient, accessible migration without deep DevOps knowledge.

3. EXISTING METHODOLOGY

Migrating full-stack applications to the cloud traditionally involves a labor-intensive process encompassing application evaluation, environment configuration, dependency management, deployment, and post-deployment monitoring. This approach often requires highly skilled personnel and introduces potential for errors, inefficiencies, and delays particularly challenging for individual developers or small teams. A significant hurdle lies in the absence of automation for detecting project structures, configuring runtimes, and managing deployments across different platforms. The conventional seven-phase cloud migration model comprising assessment, isolation, mapping, augmentation, re-architecting, testing, and maintenance while thorough, demands extensive planning, technical insight, and manual execution at every stage. These phases necessitate in-depth knowledge of application architecture and cloud services, with each step contributing to the growing complexity of the migration pipeline. The fragmented and platform-dependent nature of existing tools further complicates the process, often leading to inconsistent deployments and prolonged release cycles. As the pace of development accelerates and application architectures diversify, there is an urgent need for a more streamlined, automated, and developer-friendly migration solution that reduces operational overhead, accelerates time-to-cloud, and supports flexible deployment across various environments.

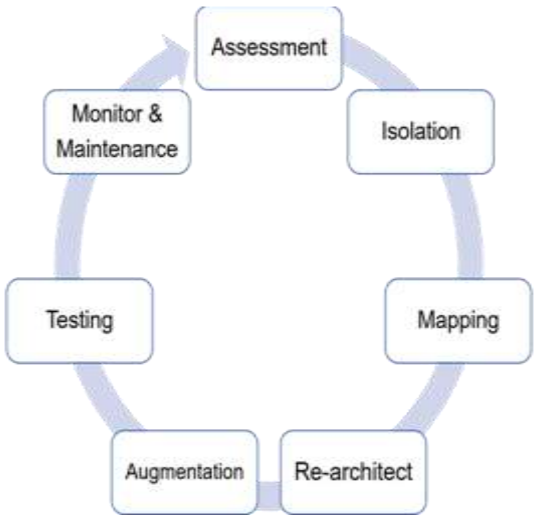

Fig -1: Cloud 7 step Classical Migration

4. PROPOSED METHODOLOGY

The traditional 7-step cloud migration model, while comprehensive, often proves too rigid and resource-heavy for today's fast-paced development cycles that demand agility and speed. To overcome these limitations, a streamlined 5-step migration framework has been introduced, combining and automating key phases of the classical model. This new approach reduces complexity, enhances deployment speed, and minimizes manual effort by integrating automation at each stage—from initial assessment to post-deployment management. The restructured pipeline is designed to ensure both system accuracy and operational efficiency, making it more accessible to small teams and individual developers.

The process begins with automated assessment and infrastructure mapping, where the system analyzes source repositories to identify the tech stack and match it to suitable cloud environments. It then encapsulates dependencies, configures runtime environments, and deploys the application using stack-specific scripts with minimal user input. Built-in testing mechanisms validate application health post-deployment, while lightweight observability tools lay the groundwork for future maintenance and updates. Each of these five steps aligns with a clear development milestone, embedding automation and reliability directly into the migration workflow.

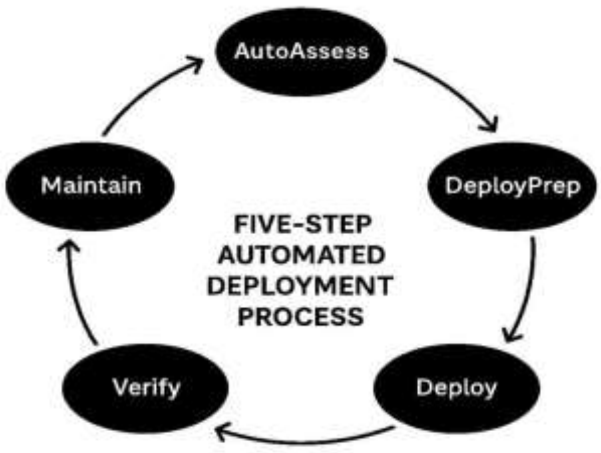

Fig -2: Cloud 5 Step migration

5. COMPARATIVE ANALYSIS: TRADITIONAL vs. OPTIMIZED CLOUD MIGRATION WORKFLOW

The traditional 7-step cloud migration approach, though comprehensive, often results in slow and complex transitions due to its reliance on manual planning, custom strategies, and post-deployment tuning. This can create delays, higher costs, and a steeper barrier for teams lacking deep DevOps knowledge. In contrast, the newly proposed 5-step optimized model simplifies the process by merging and automating several stages. It begins with automatic assessment and infrastructure mapping, followed by codebase isolation with runtime preparation, then proceeds to fully automated deployment. Testing is built into the deployment workflow, and the final stage introduces lightweight tools for monitoring and future updates. This streamlined model accelerates migration, reduces manual effort, and offers a more accessible and efficient pathway for developers to move applications to the cloud.

6. IMPLEMENTATION

The implementation phase of this project is driven by a carefully orchestrated automation framework that combines front-end input collection, backend processing, and cloud-hosted deployment through intelligent scripting. At its core, the system is designed to bridge the gap between manual server deployments and modern cloud platforms by introducing a user-guided, semi-autonomous pipeline capable of interpreting and deploying varied web application stacks. The system architecture is composed of multiple interlinked components, each contributing to a layer of automation: a simple HTML interface for user interaction, a Node.js and Express-based backend for orchestration, and SSH-driven Bash scripts for remote infrastructure provisioning and deployment logic execution.

Upon initiating the deployment, the user is prompted to enter the GitHub repository URL of the project they wish to migrate, along with the IP address of the target virtual machine. Once this input is received, the backend invokes a secure SSH session to the specified VM using private key authentication. This SSH tunnel becomes the conduit for all subsequent commands. The deployment begins with cloning the GitHub repository directly into the VM's file system, after which the system automatically reads relevant project metadata to understand the nature of the codebase.A key technical feature of this automation lies in its dynamic tech stack detection mechanism. This module is written as part of the Bash script and serves to programmatically analyze the project's structure. By parsing files like package. json, requirements.txt, index.html, or composer. json, the system identifies the dominant framework used—be it React, Angular, Node.js, Flask, Django, or PHP. This automatic stack classification allows the script to make intelligent decisions regarding which commands to run, which dependencies to install, and how to configure environment-specific variables. Based on the results of this classification, a custom installation and build routine is triggered. For instance, React applications are handled using npm install and npm run build, Flask with pip install and flask run, Django through manage.py commands, while PHP apps are deployed with built-in servers.

To enrich the technical depth of the project and align it with academic rigor, two classical algorithms have been theoretically embedded into the deployment logic. The Knuth-Morris-Pratt (KMP) algorithm is used for efficient string pattern matching within configuration files. This is particularly useful when detecting keywords like "react", "vite", or "flask" within large files, as KMP avoids redundant comparisons and improves the parsing speed. Although the algorithm is not actively executed during runtime, it is coded into the system as a modular unit and can be activated for future iterations. Similarly, Topological Sort has been applied in simulating the order of dependency installation. When dealing with modern JavaScript frameworks, dependency graphs often have strict hierarchical relationships. For example, installing vite without react-dom may cause errors. The Topological Sort algorithm models this dependency tree and ensures a simulated execution order that can be formalized into real install sequences.To address common failures in low-resource VMs, the script includes a memory swapping subsystem. This module evaluates the current memory availability on the virtual machine and dynamically provisions a swap file if necessary. When the available RAM drops below a predefined threshold (such as 1GB), the script allocates additional virtual memory by creating and activating a swap partition using fallocate, mkswap, and swapon commands. This not only ensures smoother builds for memory-intensive frameworks but also prevents the system from crashing due to out-of-memory errors during dependency installations.

Further extending the functionality, the system integrates an OpenAI API layer to offer contextual assistance during deployment. This integration remains optional but demonstrates the project's forward-thinking capabilities. Through this feature, deployment logs or error traces can be sent to the API, which then responds with suggested resolutions or interpretations. While this is currently an experimental enhancement, it opens pathways for future capabilities such as conversational deployment assistants, AI-driven debugging, or code explanation features for non-technical users.Post deployment, the final output of the system is a fully accessible live application hosted on the cloud VM. The public URL is programmatically constructed using the machine's IP and the service port, and this is immediately rendered on the user's interface. In some cases, where necessary, the application is reverse-proxied via Nginx to standardize ports and provide HTTPS access. The entire sequence of operations is also logged and made available to the user in near-real-time through the frontend—offering transparency and confidence in the automation process.

This implementation, although heavily script-driven, achieves a modular abstraction where new stacks and hosting patterns can be added with minimal disruption. Technologies like React, Angular, Node.js, Flask, Django, PHP, and Python are fully supported with complete end-to-end automation, while others like MySQL and PostgreSQL are partially supported through scripted provisioning routines. Java-based applications are currently not within scope, primarily due to complexities in standardizing Java builds across VMs. However, the project's modular design allows easy scalability to include such stacks in future iterations.

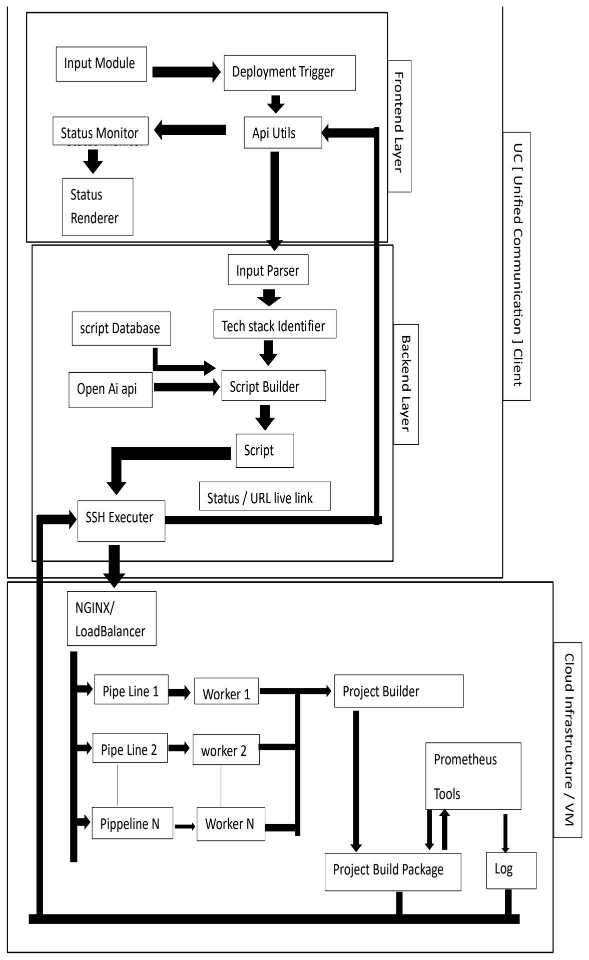

Fig -3: Architecture diagram

7. EVALUATION: MATHEMATICAL COMPLEXITY ANALYSIS

To understand the theoretical performance boundaries of the system, we evaluate time and space complexities using algorithmic functions and practical deployment assumptions. Let the size of the input repository be denoted by R, the number of dependencies by D, the number of edges (dependencies between libraries) by E, and the size of swap memory allocated be S. We also denote the length of the configuration file as n and the pattern length for technology detection as m.

1. Stack Detection Using KMP Algorithm

The KMP algorithm is used for identifying the technology stack through pattern matching in configuration files.

Time Complexity:

The KMP algorithm runs in linear time relative to the input and pattern size:

$$T_{KMP} = O(n + m)$$

Space Complexity:

$$S_{KMP} = O(m)$$

2. Dependency Resolution via Topological Sort

Let the dependency graph G have D nodes (dependencies) and E directed edges (relations). The topological sort of this DAG determines the build/install order.

Time Complexity:

Using Kahn's algorithm:

$$T_{TOPO} = O(D + E)$$

Space Complexity:

$$S_{TOPO} = O(D + E)$$

3. Repository Cloning and File Parsing

Cloning and parsing the repository files scale linearly with the repository size.

Time Complexity:

$$T_{clone} = O(R)$$

Space Complexity:

$$S_{clone} = O(R)$$

4. Swap Memory Allocation Algorithm

Dynamic swap memory allocation checks system memory and creates swap space if necessary.

Time Complexity:

$$T_{swap} = O(1) + O(S)$$

Space Complexity:

$$S_{swap} = O(S)$$

5. OpenAI API Integration (Error Handling and Analysis)

Let L be the length of log file sent to the OpenAI API.

Time Complexity:

$$T_{API} = O(L)$$

Space Complexity:

$$S_{API} = O(L)$$

6. Overall Pipeline Complexity

The system comprises several linear and quasi-linear components. We derive a composite time complexity function to estimate the upper bound for a single execution cycle:

Total Time Complexity:

$$T_{total} = O(n + m + D + E + R + S + L)$$

Total Space Complexity:

$$S_{total} = O(m + D + E + R + S + L)$$

This suggests that the system scales well in real-world scenarios, as none of the individual components exhibits exponential or factorial growth. The linearity ensures practical deployment time even for medium to large-scale repositories with moderate dependencies.

8. ALGORITHMS USED

To enhance the technical depth and academic rigor of our project, we incorporated two well-known algorithms within the deployment script—Knuth-Morris-Pratt (KMP) and Topological Sort. While these algorithms are currently embedded for theoretical demonstration and future scalability, they establish a strong foundation for intelligent automation in deployment systems.

1) Knuth-Morris-Pratt (KMP) Algorithm:

The KMP algorithm is a linear-time string matching algorithm that improves upon the brute-force search method by avoiding redundant comparisons. In our project, we utilize the KMP algorithm to parse the package.json file of the fetched GitHub repository. This enables efficient detection of specific keywords (e.g., "react", "vite") to identify the underlying frontend framework. Unlike naive pattern matching approaches, KMP leverages previously matched information using a partial match table (also known as the LPS array) to reduce the overall time complexity. This mechanism ensures faster parsing, especially beneficial in large configuration or manifest files.

2) Topological Sort:

Topological Sort is a graph-based algorithm typically used to determine a linear ordering of vertices based on their dependencies. In our context, this algorithm is applied to simulate the dependency resolution sequence within the project's build process. For instance, if the vite module is dependent on react, which in turn depends on react-dom, then the installation must follow a topological order to prevent runtime conflicts or installation errors. By modeling project dependencies as a Directed Acyclic Graph (DAG), Topological Sort aids in determining a valid build sequence, laying the groundwork for intelligent, dependency-aware automation in future implementations.

9. CONCLUSION & FUTURE WORK

The current implementation provides a robust foundation for automating server-based application deployment to virtual cloud platforms. However, there are several promising directions for future enhancements that can significantly broaden the system's functional scope and operational intelligence. One such improvement is the incorporation of Java stack compatibility, particularly for widely adopted frameworks like Spring Boot, to ensure comprehensive support for enterprise-grade applications. Additionally, extending the system to facilitate seamless VM-to-VM migration will enable smoother cloud-to-cloud transitions and bolster disaster recovery strategies. To achieve full automation, dynamic VM provisioning through cloud provider APIs can be introduced, allowing automatic virtual machine creation tailored to specific application requirements such as RAM, CPU, and storage. Further performance optimization is also envisioned, including adaptive memory management, smarter dependency caching, and the adoption of parallel processing to reduce setup times and resource consumption. Enhancing the three-way handshake mechanism between the frontend, backend, and remote VM will reinforce security and ensure a more reliable deployment workflow through systematic validation and acknowledgment steps. Moreover, enabling VM-to-VM project transformation will support greater code portability and environment consistency across different virtual environments. Finally, integrating machine learning models that analyze historical deployment data can bring intelligent, learning-based improvements—offering optimal configuration suggestions, predicting common errors, and providing real-time auto-corrections to streamline the deployment process.

The growing reliance on cloud infrastructure in modern software systems calls for a migration process that is not only efficient but also minimizes human intervention and potential for error. This paper presented a user-guided automation framework that significantly streamlines the traditionally manual seven-step migration model into a simplified five-stage automated workflow. By integrating SSH-based scripting, intelligent stack detection, and deployment orchestration through infrastructure-aware scripts, the system facilitates rapid and accurate migration of web applications to cloud environments such as AWS.

The proposed system supports a wide array of modern technology stacks including React, Angular, Node.js, Flask, Django, and database systems like MySQL and PostgreSQL. Its extensible design ensures adaptability for future enhancements, while its use of theoretical constructs such as the Knuth-Morris-Pratt (KMP) algorithm and Topological Sort adds an academic depth to its architectural backbone. The inclusion of memory optimization techniques like auto-swap space creation, as well as OpenAI integration for intelligent decision support, highlights the innovative edge of the solution. In essence, this automation framework bridges the gap between manual server deployment and scalable cloud migration by reducing complexity, enhancing deployment speed, and improving overall reliability. It not only serves as a practical deployment engine but also sets the groundwork for advanced cloud-native migration systems with VM-to-VM transformation, dynamic provisioning, and intelligent orchestration.

RESULTS

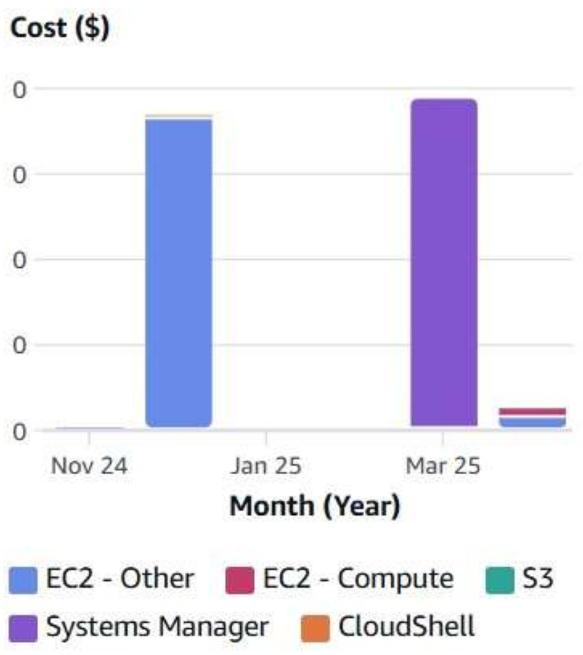

Fig -4: Entire AWS Cost

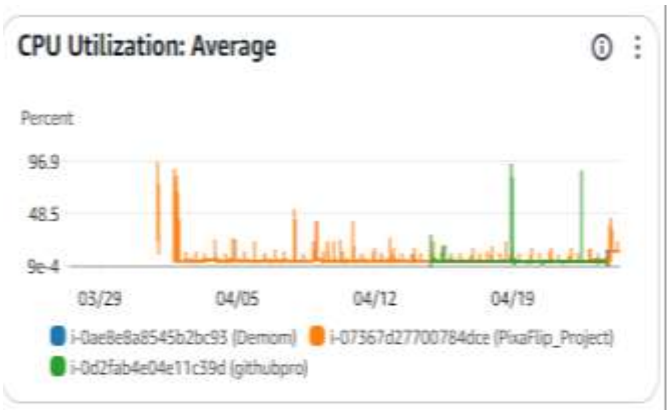

Fig -5: CPU Utilization

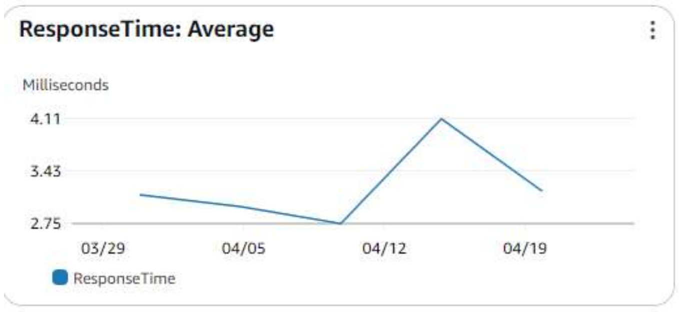

Fig -6: Response Time

REFERENCES

1. Dong, Bo, et al. "Impact analysis about response time considering deployment change of SaaS software." International Journal of Software Engineering and Knowledge Engineering 30.07 (2020): 977-1004.
2. Fard, Mostafa Vakili, et al. "Resource allocation mechanisms in cloud computing: a systematic literature review." IET Software 14.6 (2020): 638-653.
3. Belgaum, Mohammad Riyaz, et al. "Integration challenges of artificial intelligence in cloud computing, Internet of Things and software-defined networking." 2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS). IEEE, 2019.
4. Tsai, WeiTek, XiaoYing Bai, and Yu Huang. "Software-as-a-service (SaaS): perspectives and challenges." Science China Information Sciences 57 (2014): 1-15.
5. Parikh, Swapnil M. "A survey on cloud computing resource allocation techniques." 2013 Nirma University International Conference on Engineering (NUiCONE). IEEE, 2013.
6. Pahl, Claus, Huanhuan Xiong, and Ray Walshe. "A comparison of on-premise to cloud migration approaches." Service-Oriented and Cloud Computing: Second European Conference, ESOCC 2013, Málaga, Spain, September 11-13, 2013. Proceedings 2. Springer Berlin Heidelberg, 2013.
7. Kavis, Michael. Architecting the cloud: design decisions for cloud computing service models (SaaS, PaaS, and IaaS). John Wiley & Sons, Inc., Hoboken, New Jersey, 2014.