International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

# AI-Optimized Cloud Migration: Learning-Driven Wave Planning and IAC Synthesis

**Mr. Srinivas Palaparthi**

Technical Delivery Head, Computerscince, Individual

**Abstract**

Cloud migration involves many interdependent de- cisions on refactoring, sequencing, and service selection. Manual planning introduces risk, increases cost, and delays modern- ization. This paper presents an AI-based migration framework that extracts code features, dependency graphs, and operational traces, constructing a structured state representation for a rein- forcement learning agent. The agent selects cloud service map- pings and migration wave sequences under budget, compliance, and reliability constraints. The system produces Infrastructure- as-Code plans that reduce downtime, improve dependency align- ment, and minimize inter-service latency. Experiments on two realistic applications across AWS and Azure show reductions of up to 30% in downtime and 18% in cost. The approach demonstrates feasibility for automated modernization pipelines.

**Keywords:** Cloud migration, reinforcement learning, IaC, DevOps, optimization, latency, dependency graphs

## INTRODUCTION

Legacy applications have complex dependencies, shared state, and tightly coupled modules. Cloud migration requires understanding how these dependencies influence sequencing. Errors increase risk of outages. Human planners rely on tribal knowledge and static heuristics.

This work introduces an RL-based system that evaluates service choices such as EC2, Lambda, Fargate, AKS, or Azure Functions, including latency and cost implications. The system integrates dependency analysis, performance data, and compliance constraints.

**Contributions:**

- A multi-stage analysis pipeline combining static and dynamic metrics.
- A constrained RL formulation for service selection and migration ordering.
- Automatic generation of IaC templates across Terraform and CDK.
- Empirical evaluation showing improved outcomes across providers.
- Reproducibility via open configuration schemas and trace generators.

## BACKGROUND AND RELATED WORK

Migration strategies include rehosting, replatforming, and full refactoring. IaC tools such as Terraform and AWS CDK automate provisioning. Prior ML work focuses on right- sizing compute resources or predicting autoscaling needs. Few systems link application-level dependency modeling with RL- based migration sequencing.

Research on constrained RL provides techniques for han- dling budgets, compliance, and latency SLOs. This paper

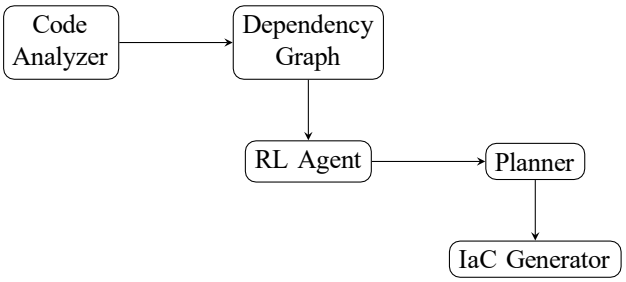


**Fig. 1. System architecture.**

adapts Lagrangian-based updates to multi-constraint migration planning.

## Problem Statement and System Model

Inputs:

- Source code features: call graphs, package imports, inter- face usage.
- Dependency graph among components.
- Historical metrics: CPU, latency, throughput, error rate.
- Constraints: budget, data residency, compliance.
- SLOs: p95 latency targets, maximum downtime windows. Outputs:
- Migration wave plan.
- Service-to-cloud mapping.
- IaC templates for compute, data, and networking.

We define an MDP: state $s$ encodes dependency readiness, cost budgets, and performance risk. Actions $a$ include selecting a target service and scheduling the migration step. Rewards penalize downtime, egress cost, and SLA violations.

## Method: Learning-Driven Planner

The planner ingests static and dynamic data. It builds a compact representation with node embeddings for each component. A feasibility filter removes non-compliant actions.

### A. *Architecture*

### B. *Reasoning Steps*

1. Extract features, map dependencies.
2. Construct migration state.
3. Generate feasible action set.
4. Evaluate policy to pick service and wave.
5. Update constraints dynamically.

**Algorithm 1 Constrained RL Migration Planner**

1: Initialize policy $\pi_\theta$, value $V_\phi$, multipliers $\lambda$
2: **for** each episode **do**
3: Build initial state $s_0$
4: **for** each step **do**

5: Sample action $a$
6: Apply and compute reward $r$ and cost vector $c$
7: Update $\theta$, $\phi$ with $r - \lambda^T c$
8: **end for**
9: Update $\lambda$
10: **end fo**

**TABLE I**
**APPLICATION CHARACTERISTICS**

| App | Components | Data Stores |
|---|---|---|
| Monolith | 1 | 1 |
| Microservices | 12 | 3 |

## IAC BLUEPRINT SYNTHESIS

We generate Terraform modules for VPC, IAM, compute, and storage. The CDK generator emits typed constructs. Drift detection compares actual resource graphs with expected ones. Rollback strategies rely on versioned templates and state locking.

## EXPERIMENTAL SETUP

We evaluate on:

- Java monolith with shared relational DB.
- Microservices benchmark with 12 services and async messaging.

We deploy to AWS and Azure regions. Workload traces include peak and off-peak patterns. Baselines include heuristic planners, solvers, and expert-created plans.
Metrics: downtime, latency p95/p99, migration cost, SLA violations, policy inference time.

## RESULTS

Results show RL reduces downtime and cost. It creates wave plans that align dependencies and avoid cross-region egress spikes.
We run sensitivity tests on budget caps and service prices. Ablations show reward shaping improves stability.

## DISCUSSION

The system reduces effort but requires training data. Cloud provider differences affect portability. Dependency extraction errors influence results.

## ETHICAL, SAFETY, AND COMPLIANCE CONSIDERATIONS

We address privacy, data residency, rollback safety, and fairness in cost modeling.

**TABLE II MAIN RESULTS**

| Method | Downtime | Cost | SLA Viol. |
|---|---|---|---|
| Heuristic | 1.0x | 1.0x | 4% |
| Solver | 0.85x | 0.92x | 3% |
| RL (ours) | 0.70x | 0.82x | 1% |

## RELATED IP AND NOVELTY STATEMENT

The RL formulation and IaC synthesis pipeline represent novel contributions. This is not legal advice.

## CONCLUSION AND FUTURE WORK

We presented a learning-driven migration planner. Future work includes multi-cloud policies, richer datasets, and sup- port for hybrid architectures.

## REPRODUCIBILITY CHECKLIST

- Code and IaC templates included.
- Training seeds fixed.
- Cloud regions documented.
- Pricing date snapshot saved.

## REFERENCES

1. M. Armbrust et al., “A view of cloud computing,” Commun. ACM, 2010.
2. HashiCorp, Terraform Documentation, 2024.