École Hassania des Travaux Publics

Department MIG

Major GIS

Final Studies Project Report

Realized by:

Yassine DBAICHI

Expand Infrastructure as Code Support in Graal CI

JSON-Driven Automation System with Pulumi

for Oracle Cloud Infrastructure

Host Organization

Oracle Corporation – GraalVM RISQ Team

Under the Supervision of:

Prof. Hatim LECHGAR – Academic Supervisor, EHTP

Dir. Mohamed EZ-ZARGHILI – Dotted Line Manager, Oracle

Mr. Hamza GHAISSI – Mentor, Oracle

Academic Year: 2024/2025

Dedications

To my everything: my incredibly dear parents,

No amount of eloquent words can truly express my gratitude and appreciation for everything you have done for me. Your tireless efforts in ensuring my education, safety, and well-being have shaped me into the person I am today. Your boundless encouragement, patience, and understanding have been the unwavering support you have consistently provided me.

I promise to always strive to make you proud and never disappoint you.

May Allah, the Almighty, protect you and grant you happiness and good health.

To my brothers

Thank you for your love and encouragement. I wish you boundless success and courage to chase your dreams fearlessly. May my journey inspire you to pursue your own achievements.

To my friends,

Thank you for making these past years unforgettable. Your companionship has added depth and joy to every moment of our adventure together.

To myself,

Thank you for never giving up and always believing.

Yassine Dbaichi

Acknowledgments

I would like to express my sincere gratitude to **Mr. Mohamed Ez-Zarghili**, Software Director and my dotted-line manager, for giving me the opportunity to be part of this valuable experience and for his continued trust and guidance.

My heartfelt thanks go to **Mr. Hamza Ghaissi**, my mentor, for his constant support, encouragement, and constructive feedback throughout this internship. His insights and patience have been instrumental to my growth.

I would also like to extend my appreciation to the **GraalVM RISQ team** for their warm welcome and for providing an enriching and dynamic technical environment that enabled me to learn and contribute effectively.

I am deeply thankful to my academic supervisor, **Prof. Hatim Lechgar** (EHTP), for his guidance, availability, and trust throughout this project.

My sincere appreciation also extends to the esteemed members of the **jury** for kindly agreeing to evaluate this work. I am grateful for the time they dedicated and the valuable insights they provided.

Finally, I would like to express my deepest gratitude to my cherished **family and friends**. Your unwavering support, encouragement, and unconditional love have been a constant source of strength. Your faith in me has fueled my determination, and this accomplishment is as much yours as it is mine.

Each of you has contributed in a meaningful way to the success of this project, and for that, I remain truly and deeply thankful.

Résumé

Ce rapport présente mon stage de fin d'études au sein de l'équipe Oracle GraalVM RISQ. J'y ai conçu une solution d'Infrastructure as Code (IaC) avancée, pilotée entièrement par fichiers JSON distincts par environnement ("stack-based").

L'architecture permet de décrire, déployer et maintenir l'infrastructure OCI (réseaux, machines, stockage, NAT Gateway, clusters OKE, NodePools, etc.) via un unique moteur Python basé sur Pulumi. La création des ressources est dynamique grâce à un Factory Pattern central, une gestion intelligente des dépendances (références par nom ou OCID), et un chargeur de configuration qui sélectionne le bon JSON selon l'environnement (production, staging, etc.).

L'ensemble est intégré à un pipeline CI/CD GitLab pour l'automatisation des déploiements, la détection de dérive et la gestion sécurisée des secrets.

Ce projet démontre une approche modulaire, réutilisable et extensible, adaptée aux besoins modernes du cloud.

Mots Clés : Infrastructure as Code, Pulumi, OCI, JSON, Stack, Python, OKE, CI/CD.

Abstract

This report presents my final-year internship conducted with the Oracle GraalVM RISQ Team, focused on designing a modular, production-ready Infrastructure as Code (IaC) platform for Oracle Cloud Infrastructure (OCI). The core innovation of this project is a fully JSON-driven engine that enables dynamic, environment-specific provisioning and management of cloud resources—networks, compute instances, storage buckets, gateways, Kubernetes clusters (OKE), and more.

The system leverages a Python orchestrator built around the Factory Pattern and dependency injection, ensuring resources are created in the correct order and can reference each other by logical names or OCIDs. Each environment ("stack") is described by its own JSON file, making the infrastructure easily reproducible, adaptable, and version-controlled. A dynamic configuration loader selects the appropriate settings based on the Pulumi stack in use.

A critical aspect of this work was designing the DependencyResolver and ResourceFactory classes, which manage resource relationships and allow seamless extension of new resource types—without changes to the core engine. The platform is deeply integrated with GitLab CI/CD, providing automated drift detection, preview, and deployment pipelines, as well as secure management of sensitive credentials via environment variables.

Through this project, I implemented a scalable, maintainable, and extensible IaC framework that follows modern DevOps best practices. Automated workflows, strict separation of configuration from code, and advanced error handling ensure reliability and operational excellence for cloud deployments.

Keywords: Infrastructure as Code, Oracle Cloud Infrastructure, Pulumi, Python, JSON, Factory Pattern, CI/CD, Stack, Automation, DevOps.

Abbreviations

Abbreviation	**Definition**
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Deployment
DevOps	Development and Operations
IaC	Infrastructure as Code
JSON	JavaScript Object Notation
OCI	Oracle Cloud Infrastructure
OCID	Oracle Cloud Identifier
OKE	Oracle Kubernetes Engine
Pulumi	Pulumi Infrastructure-as-Code Platform
SDK	Software Development Kit
CLI	Command-Line Interface
VCN	Virtual Cloud Network
NAT	Network Address Translation (Gateway)
S3	Amazon Simple Storage Service (S3)
IAM	Identity and Access Management

Contents

Dedications i

Acknowledgments ii

Résumé iii

Abstract iv

Abbreviations v

General Introduction 1

1 General Project Context 4

Introduction . . . 4

1.1 Host Organization . . . 4

1.1.1 Presentation of the Host Company . . . 4

1.1.2 Business Area of Activity . . . 5

1.1.3 Oracle MADC . . . 6

1.1.4 Oracle Labs . . . 7

1.1.5 Organizational Structure . . . 8

1.1.6 Organizational culture . . . 9

1.1.7 Oracle Morocco Projects . . . 9

1.2 Project Overview . . . 10

1.2.1 The GraalVM RISQ Team . . . 10

1.2.2 Infrastructure as Code and Oracle Cloud Infrastructure . . . 11

1.2.3 Context and Problem Statement . . . 11

1.2.4 Proposed Solution: JSON-Driven IaC Framework . . . 12

Conclusion . . . 13

2 Project Management **14**

Introduction . . . 15

2.1 Management Approach . . . 15

2.1.1 Team Meetings and Collaboration . . . 15

2.1.2 Mentor Check-Ins . . . 16

2.1.3 Task Management . . . 17

2.1.4 Documentation Tools . . . 17

2.1.5 Source Code Management . . . 18

2.1.6 Communication Tools . . . 19

2.2 On-boarding and integration process . . . 19

2.3 Mandatory Compliance Training . . . 19

2.4 Generic & Technical Training . . . 20

2.5 Development Workflow . . . 22

2.5.1 Task Lifecycle . . . 22

Conclusion . . . 23

3 Current State Analysis and Solution Specification **24**

Introduction . . . 24

3.1 Current State Analysis . . . 24

3.1.1 Manual Infrastructure Management . . . 24

3.2 Identified Issues . . . 25

3.2.1 Operational Problems . . . 25

3.2.2 Quality and Consistency Problems . . . 26

3.3 Requirements . . . 26

3.3.1 Functional Requirements . . . 26

3.3.2 Non-Functional Requirements . . . 27

3.4 Solution Evaluation . . . 27

3.4.1 Technology Comparison . . . 27

3.4.2 Selection decision . . . 27

3.5 Selected Solution: Pulumi . . . 28

3.5.1 Pulumi Overview . . . 28

3.5.2 Operational Model . . . 28

3.5.3 Architecture Design . . . 30

Conclusion . . . 32

4 Design and Implementation of the Solution 33

Introduction . . . 33

4.1 Solution Design Approach . . . 33

4.1.1 Use Case Analysis . . . 33

4.1.2 Design Principles . . . 35

4.2 System Architecture . . . 35

4.2.1 High-Level Architecture Overview . . . 35

4.2.2 Component Architecture . . . 36

4.3 Technical Design Patterns . . . 37

4.3.1 Factory Pattern Implementation . . . 37

4.3.2 Stateless Architecture Design . . . 39

4.3.3 Dependency Resolution Strategy . . . 39

4.3.4 Class Hierarchy and Inheritance . . . 40

4.4 Infrastructure Provisioning Workflow . . . 40

4.4.1 End-to-End Workflow Design . . . 40

4.4.2 GitLab CI/CD Pipeline Integration . . . 41

4.4.3 Stack-Based Configuration Management . . . 42

4.5 Implementation Architecture . . . 43

4.5.1 Core Component Implementation . . . 43

4.5.2 JSON Configuration Structure . . . 43

4.5.3 Custom Docker Image Integration . . . 44

4.6 State Management and Deployment . . . 44

4.6.1 Pulumi State Storage Architecture . . . 44

4.6.2 Stack Selection Mechanism . . . 44

4.6.3 Error Handling and Recovery . . . 45

Conclusion . . . 45

5 Results and Validation 46

Introduction . . . 46

5.1 Implementation Results . . . 46

5.1.1 Delivered Features . . . 46

5.1.2 Graal CI Integration Achievements . . . 47
5.2 Testing and Validation Results . . . 47
5.2.1 Infrastructure Deployment Testing . . . 47
5.2.2 CI/CD Pipeline Validation . . . 48
5.3 Performance, Reliability, and Impact . . . 49
5.3.1 Deployment Efficiency . . . 49
5.3.2 Team Adoption and Experience . . . 49
5.4 Framework Extensibility Demonstration . . . 49
5.4.1 Adding New Resource Types . . . 49
5.4.2 Current Limitations and Future Work . . . 50
5.5 Project Impact Assessment . . . 50
5.5.1 Workflow Comparison . . . 50
5.5.2 Quantifiable Benefits . . . 50
Conclusion . . . 51

General Conclusion **52**

List of Figures

1.1 Oracle logo 4
1.2 Oracle Customer Successes 6
1.3 Oracle MADC Location 6
1.4 Oracle Labs Logo [8]. 7
1.5 Executive Leadership Board [**?**] 8
1.6 GraalVM logo 10

2.1 Biweekly team meeting 16
2.2 Mentor check-in workflow 17
2.3 Jira backlog and task view 17
2.4 Confluence workspace 18
2.5 Git version control 18
2.6 GitLab CI/CD pipeline 18
2.7 Communication tools used for daily collaboration 19
2.8 New hire onboarding Tickets 20
2.9 Training platforms and modules 21
2.10 Task lifecycle from creation to completion 22

3.1 Pulumi Overview 28
3.2 Pulumi Deployment Steps 29
3.3 Pulumi Deployment engine 29
3.4 Architecture design. 31
3.5 workflow design. 31

4.1 Use Case Diagram 34
4.2 Architecture Layers 35
4.3 Project Structure. 36

4.4 Resource Factory - Extensible Registration Mechanism 38
4.5 dependancy resolution workflow 39
4.6 dependancy resolution workflow 40
4.7 End-to-End Workflow Design. 41
4.8 GitLab Pipeline Stages 42

5.1 Instance Deployed Successfully. 48
5.2 Drift Detection Report 48

List of Tables

1.1 Oracle Corporation Key Figures 5

3.1 Terraform [3] vs Pulumi [9] comparison 27

5.1 Validation Scenarios and Outcomes 47

5.2 Development Workflow Improvements 50

General Introduction

In today's rapidly evolving technological landscape, modern software development teams face unprecedented challenges in managing complex cloud infrastructure. Traditional manual approaches to infrastructure provisioning and management have become significant bottlenecks, hindering development velocity and introducing inconsistencies across environments. As organizations increasingly adopt cloud-native architectures and DevOps practices, the need for automated, reliable, and scalable infrastructure management solutions has become critical.

Oracle Cloud Infrastructure (OCI) provides a comprehensive suite of cloud services designed to support enterprise-grade applications and development workflows. However, manual infrastructure provisioning through web consoles remains time-consuming, error-prone, and inconsistent. Development teams often spend hours configuring network components, compute instances, and container orchestration platforms, leading to delayed project timelines and frustration among team members. Furthermore, the lack of version control and reproducibility in manual processes creates significant challenges for maintaining consistent environments across development, testing, and production stages.

Infrastructure as Code (IaC) has emerged as a transformative approach to address these challenges by treating infrastructure configuration as software code. This paradigm shift enables development teams to leverage familiar software engineering practices such as version control, code review, automated testing, and continuous integration for infrastructure management. By defining infrastructure through declarative configuration files, organizations can achieve consistent, repeatable, and auditable infrastructure deployments while significantly reducing manual overhead.

Within Oracle's GraalVM RISQ team, the need to expand Infrastructure as Code support in Graal CI pipelines became apparent as development workflows increasingly required rapid provisioning of complex testing and development environments. The existing manual processes were creating bottlenecks in the continuous integration workflow, preventing the team from achieving optimal development velocity and consistent environment management.

This project addresses these challenges through the development of a comprehensive JSON-driven

Infrastructure as Code framework specifically designed to integrate with and extend existing Graal CI pipelines. The solution leverages Pulumi's programmatic approach combined with Python's flexibility to create a factory pattern-based system that enables developers to provision complex Oracle Cloud Infrastructure environments through simple JSON configuration files.

The core innovation lies in the framework's "Update JSON, get infrastructure" philosophy, which abstracts the complexity of infrastructure provisioning behind an intuitive configuration interface. Developers can define their infrastructure requirements in human-readable JSON files, and the framework automatically handles dependency resolution, resource creation ordering, and integration with GitLab CI/CD pipelines. This approach democratizes infrastructure management within the team, allowing developers to focus on their core responsibilities while maintaining full control over their infrastructure requirements.

The technical architecture employs several key design patterns to ensure extensibility, maintainability, and reliability. The factory pattern enables easy addition of new resource types without modifying core framework components, while static class-based implementation ensures optimal performance and predictable behavior. The dependency resolution system automatically converts logical resource names to Oracle Cloud Identifiers (OCIDs) at runtime, supporting both managed resources and integration with existing infrastructure.

Integration with GitLab CI/CD pipelines provides seamless workflow enhancement, introducing infrastructure provisioning capabilities directly into existing development processes. The framework includes preview and apply workflows with appropriate approval gates, ensuring that infrastructure changes undergo proper review and validation before deployment. State management through OCI Object Storage enables team collaboration while maintaining proper access controls and audit trails.

The solution encompasses comprehensive support for Oracle Cloud Infrastructure services essential for modern application development, including Virtual Cloud Networks (VCNs), compute instances, Oracle Kubernetes Engine (OKE) clusters, and storage services. Automatic Oracle Linux image discovery ensures that provisioned instances always use the latest security patches, while network isolation and security group configuration follow best practices for enterprise environments.

This project represents a significant advancement in the team's development capabilities, providing a scalable foundation for infrastructure automation that can evolve with changing requirements. The framework's extensible architecture ensures that new Oracle Cloud Infrastructure services can be easily integrated as they become available, while the JSON-driven configuration approach maintains accessibility for team members regardless of their infrastructure expertise level.

The following sections of this report provide detailed examination of the project's context, methodology, implementation, and results. The document structure encompasses:

- **Chapter 1** establishes the organizational context within Oracle's GraalVM RISQ team.
- **Chapter 2** details the project management methodologies employed throughout the development process.
- **Chapter 3** presents comprehensive analysis of existing infrastructure management challenges and solution requirements.
- **Chapter 4** examines the technical architecture designed to address these requirements through innovative Infrastructure as Code solutions.
- **Chapter 5** provides detailed implementation insights, validation results, and assessment of the framework's impact on team development workflows and infrastructure management practices.
- **General Conclusion** evaluates the project's achievements, lessons learned, and future development directions for expanding Infrastructure as Code capabilities within the organization.

Through this comprehensive Infrastructure as Code implementation, the project establishes a robust foundation for automated infrastructure management that enhances development team productivity, ensures consistent environment provisioning, and provides the scalability necessary to support the organization's evolving technical requirements. The framework represents a significant step toward fully automated development workflows while maintaining the flexibility and control essential for enterprise-grade software development practices.

Chapter 1

General Project Context

Introduction

This chapter provides an overview of the project's context, both organizational and technical, by introducing the host organization and its services, as well as by presenting the project's problems and objectives.

1.1 Host Organization

My internship was hosted by Oracle Corporation, specifically at the Oracle Morocco Research and Development Center. This section introduces both entities, offering insights into their history and services.

1.1.1 Presentation of the Host Company

Figure 1.1: Oracle logo

Oracle Corporation is a technology company headquartered in Austin, Texas. It was founded in 1977 by Larry Ellison, Bob Miner, and Ed Oates under the name Software Development Laboratories (SDL).

Initially, the company focused on creating the Oracle Database, a relational database management system that became its flagship product.

Today, Oracle offers a broad portfolio that includes enterprise software, cloud computing solutions, hardware systems, and consulting services. A major milestone in its history was the acquisition of Sun Microsystems in 2010, which brought key technologies such as Java and the Solaris operating system.

Oracle is now a global leader in cloud-based IT infrastructure and enterprise software. It enables organizations to scale operations, improve efficiency, and enhance performance. Among its innovations is the Oracle Autonomous Database — the world's first autonomous database — as well as Oracle Cloud [7].

A summary of Oracle's current key figures is presented in the table below:

Table 1.1: Oracle Corporation Key Figures

Country	U.S.
CEO	Safra A. Catz
Website	oracle.com
Employees	132,000
Revenue ($B)	40.479
Profit ($B)	13.746
Market Value ($B)	220.736.6

1.1.2 Business Area of Activity

Oracle provides a variety of goods and services to its clients [**?**], including:

- Software development tools
- Cloud services
- Training, consultancy, and certifications
- Integrated solutions such as SaaS, PaaS, IaaS, and DaaS

Oracle provides a variety of goods and services to its clients. Software development tools, cloud services, training, consultancy, and credentials are just a few of the goods and services offered here. One of the brands Oracle offers is Oracle Database, which has held the top spot since 1979 and is supported by cloud artificial intelligence. SaaS (Software as a Service), PaaS (Platform as a Service), IaaS (Infrastructure as a Service), and DaaS (Data as a Service) are just a few of the integrated solutions offered by Oracle Cloud Infrastructure for business, IT, and development needs.

Oracle Middleware is also a platform that is integrated for creating and running intelligent, agile applications while increasing technical efficiency using contemporary software designs. Oracle Services, which include Oracle Advanced Customer Support Services, Oracle Premium Support, Oracle Consulting, Oracle Financing, and Oracle Managed Cloud Services [**?**].

Figure 1.2: Oracle Customer Successes

1.1.3 Oracle MADC

Oracle has a well-established presence in the North Africa region, with offices not only in Morocco but also in Algeria. Oracle has chosen Casablanca as the location for the growth of its international R&D program. The Moroccan office was founded in 1995 and has since grown to encompass two entities with a total of 269 staff members. These entities are Oracle North Africa, which includes 49 headcounts, and Oracle Research and Development Morocco, which as of February 1st, 2024, consists of 220 staff members, including 80 interns.

Figure 1.3: Oracle MADC Location

The Oracle Morocco facility forms part of Oracle's global R&D innovation network. In these centres, researchers apply novel approaches and methodologies using cloud-based technologies to create meaningful solutions. This work includes research, consulting, and product incubation with many projects involving university collaborations and interns from over 50 universities worldwide. Oracle Morocco operates from several strategic locations within the Casablanca Nearshore business area:

- **Shore 14, Second Floor**
- **Shore 23, First Floor**
- **Shore 23, Ground Floor**
- **Shore 32, New Building (7 Floors)**

1.1.4 Oracle Labs

The research and development division of Oracle is called Oracle Labs [8], which focuses primarily on creating technologies that will keep Oracle at the forefront of the IT industry. Researchers at Oracle Labs pursue innovative methodologies and approaches, frequently taking on challenges or projects that would be difficult to complete within a product development organization [8].

Figure 1.4: Oracle Labs Logo [8].

Real-world applications are the focus of Oracle Labs research, and the researchers are working to create technologies that will eventually have a significant impact on how society and technology evolve. For instance, work done in Oracle Labs led to the development of chip multithreading and the Java programming language. Oracle's quest for research and development is greatly supported by Oracle

Labs. Oracle can recognize challenges, find creative solutions, and create new products that can spur growth and innovation in the future because of its well-rounded research portfolio [8].

1.1.5 Organizational Structure

Oracle Corporation has a hierarchical organizational structure with three main business divisions: Hardware, Services, Cloud and License. Each segment is divided into smaller business units focused on goods, services, or geographical areas. The Board of Directors oversees the company's performance and strategic direction, while the Executive Leadership Team [**?**] manages daily operations. Functional divisions, including sales, marketing, finance, human resources, legal, and IT, fall under the Executive Leadership Team. Business units are arranged by region, product, and service, with a senior VP or general manager overseeing each unit's operations. The employees are subordinated to both a functional manager and a business unit manager in the matrix-based organizational structure of the business units.

Figure 1.5: Executive Leadership Board [**?**]

In general, Oracle's organizational structure is built to encourage teamwork, creativity, and client attention. While the matrix form enables people to work across many functions and business divisions to achieve shared goals, the hierarchical structure promotes effective decision-making and communication.

1.1.6 Organizational culture

Oracle has a robust corporate culture that places a premium on innovation, client centricity, and excellence. The company's history of innovation, emphasis on customer fulfilment, and commitment to nurturing a welcoming and collaborative workplace all have an impact on its culture. Oracle's culture places a strong emphasis on the following:

- **Innovation:**A long history of developing innovative products and technologies is fostered by a commitment to research and development and by providing employees with the resources and encouragement to test new ideas and develop new products. [**?**].
- **Customer Satisfaction:** Oracle's customer-centric approach reflects its commitment to providing goods and services that satisfy client needs. Employees are encouraged to collaborate closely with clients to understand their demands and craft appropriate solutions. [**?**].
- **Quality:** Oracle maintains a strong reputation for quality, reliability, and performance. Employees are supported with training and resources to pursue excellence. [**?**].
- **Collaboration and Teamwork:** Effective cooperation is crucial. The organisation encourages cross-functional collaboration and provides tools to support communication and coordination.

Overall, Oracle's organizational culture is characterized by a commitment to innovation, customer focus, excellence, collaboration, and teamwork. These values are reflected in the company's products, services, and operations, and are embraced by its employees at all levels of the organization [**?**].

1.1.7 Oracle Morocco Projects

Oracle Morocco is renowned for strong research and development capabilities, but its contributions extend far beyond its labs. The organization encompasses several lines of business (LOBs), which allow it to significantly contribute to the broader goals of Oracle Corporation. This diversity fosters innovation and supports various business functions across the region. New lines of business at Oracle Morocco include:

1. **Java Platform Group**
2. **Digital Government**
3. **NetSuite Cloud**

4. **JFusion ERP**
5. **Oracle Analytics**
6. **Oracle Cloud Infrastructure**

These are just some of the teams at Oracle Morocco. While there are many more initiatives, this list provides a glimpse into the dynamic and evolving nature of Oracle Morocco.

Internship placement: My internship took place within the **GraalVM RISQ team** at Oracle Labs (hosted at Oracle Morocco Advanced Development Center). During this internship I worked on an Infrastructure as Code (IaC) [5] automation framework to support Graal CI workflows and the team's infrastructure needs.

1.2 Project Overview

1.2.1 The GraalVM RISQ Team

Figure 1.6: GraalVM logo

The GraalVM RISQ team at Oracle Labs is responsible for the Release, Infrastructure, Sustaining, Security, Support, and Quality of the GraalVM [6] project. This team serves as the operational backbone for GraalVM, managing both open-source and commercial releases, automating the testing and delivery pipelines, and ensuring the reliability and security of the platform across multiple architectures.

Team Mission and Organization:

- **Release and QA:** Managing the full release process, from CI builds and artifact signing to extensive automated testing and validation.
- **Infrastructure Management:** Provisioning, automating, and maintaining the OCI (Oracle Cloud Infrastructure) [7] resources needed for continuous integration, nightly testing, and on-demand experimental environments.
- **Security and Sustaining:** Coordinating patching, long-term support, and security compliance with both Oracle internal and external open-source stakeholders.

- **Tooling and Automation:** Building custom Python, Java, and shell tooling to automate repetitive workflows and increase team efficiency.

The team operates in a highly collaborative environment, using agile practices, peer reviews, and continuous integration to deliver rapid, high-quality improvements to GraalVM. My internship within this team focused on designing and implementing a JSON-driven Infrastructure as Code (IaC) automation framework to modernize and standardize how cloud environments are provisioned and maintained for Graal CI.

1.2.2 Infrastructure as Code and Oracle Cloud Infrastructure

Infrastructure as Code (IaC) [5] is a modern approach to managing and provisioning computing resources through machine-readable definition files, rather than physical hardware or interactive configuration tools. This paradigm allows teams to treat infrastructure just like application code—versioned, tested, reviewed, and deployed using automation pipelines [4].

Oracle Cloud Infrastructure (OCI) [7] is Oracle's public cloud platform, providing compute, storage, networking, container, and security services suitable for enterprise and R&D workloads. OCI exposes a robust API and native SDKs that make it an ideal target for IaC and DevOps automation. Key OCI resources for the RISQ team include:

- Virtual Cloud Networks (VCNs) and subnets for network isolation and routing,
- Compute instances for build, test, and deployment,
- Kubernetes clusters (OKE) for scalable CI workloads,
- Object storage for artifacts and state management.

1.2.3 Context and Problem Statement

Prior to this project, the RISQ team managed OCI resources primarily via manual console operations, ad-hoc scripts, and out-of-date documentation. This approach led to multiple challenges:

- **Inconsistent Environments:** Manual setup resulted in subtle differences between development, staging, and production environments, causing unexpected CI failures or "works on my machine" issues.
- **Slow Provisioning:** Creating or updating infrastructure required significant manual effort, slowing down the ability to launch new test scenarios or onboard new team members.

- **No Change Tracking:** Without version control, it was difficult to know who changed what, or to roll back broken environments.
- **Bottlenecked Expertise:** Only a few team members had deep OCI knowledge, concentrating risk and blocking self-service.
- **Security and Compliance Gaps:** Manual steps made it easy to overlook access control, network rules, or audit requirements.

As the complexity and scale of GraalVM's CI and release workflows increased, these problems became a barrier to agility and reliability.

1.2.4 Proposed Solution: JSON-Driven IaC Framework

To address these issues, the internship focused on developing a JSON-driven Infrastructure as Code framework using Pulumi [9] and Python, designed specifically for the RISQ team's needs:

- **Declarative Configuration:** All OCI resources (VCNs, subnets, compute, OKE, buckets, etc.) are defined in JSON files—versioned in Git and reviewed by the team.
- **Automated Provisioning:** A factory pattern [1] maps JSON sections to Python classes, enabling dynamic creation and destruction of resources via Pulumi [9].
- **Dependency Resolution:** The system automatically resolves dependencies between resources, ensuring correct creation order and minimizing human error.
- **GitLab CI/CD Integration:** The framework integrates with GitLab [2] pipelines, supporting preview plans, manual approvals, and drift detection as part of the team's standard workflow.
- **Multi-Environment Support:** Separate configurations for development, staging, and production, with stack isolation for safe, reproducible changes.
- **Self-Service and Security:** Any team member can request or propose changes via pull request, with all changes reviewed, audited, and deployed automatically.

This solution provided the RISQ team with a robust, extensible, and maintainable foundation for managing cloud infrastructure—dramatically improving agility, consistency, and team productivity.

Conclusion

This chapter introduced Oracle Corporation, Oracle Labs, and the GraalVM RISQ team's infrastructure challenges. We identified the core problem: manual infrastructure management causing configuration drift and operational inefficiencies. The proposed JSON-driven Pulumi framework addresses these issues through automation and declarative configuration.

The next chapter examines the project management methodology, including organizational structure, communication workflows, collaboration tools, and development practices used during this internship.

Chapter 2

Project Management

Introduction

Effective project management plays a vital role in ensuring structured development, smooth collaboration, and timely delivery of project goals. While the project did not strictly follow an Agile methodology, it was managed using a flexible and adaptive approach. Meetings were held on a weekly, biweekly, or monthly basis depending on progress and availability, and tasks were driven by evolving objectives and exploratory development cycles. I worked exclusively within the GraalVM RISQ team.

2.1 Management Approach

The project followed a semiformal management style, tailored to the exploratory and research-driven nature of the GraalVM ecosystem. Rather than strict Scrum sprints, we used a lightweight **Kanban** method to manage work. Tasks were tracked in Slack project tracker with four main columns: **To Do**, **In Progress**, **Blocked**, and **Done**. Daily short check-ins with my mentor and biweekly team meetings provided synchronization and review points, while milestone-based planning guided the main deliverables.

2.1.1 Team Meetings and Collaboration

Beginning in April, I was invited to participate in the GraalVM RISQ team's biweekly meetings. These comprehensive sessions brought together members from various subteams and served as a vital touchpoint for maintaining cohesion and transparency across the project. The meetings typically included:

- **Progress Updates:** Each participant shared updates on recent activities, technical developments, and any issues encountered.
- **Planning and Prioritization:** We discussed upcoming tasks, refined timelines, and adjusted priorities based on the current status of the project.
- **Collaborative Problem Solving:** The meetings provided a platform for team-wide brainstorming to address blockers and exchange solutions.

These meetings not only kept me informed of cross-functional developments but also offered valuable exposure to real-world engineering practices and collaborative workflows.

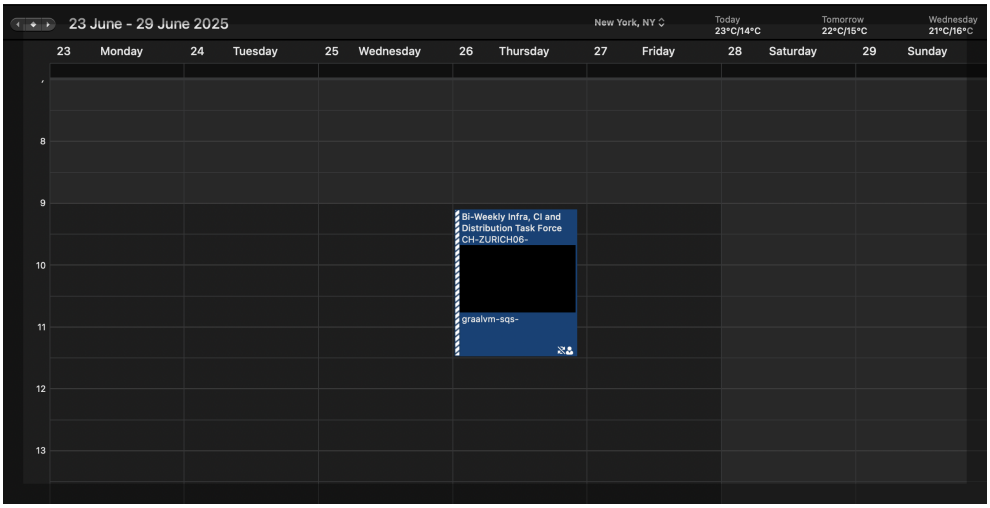

Figure 2.1: Biweekly team meeting

2.1.2 Mentor Check-Ins

In addition to team meetings, I had recurring one-on-one check-ins with my assigned mentor. These sessions occurred at flexible intervals, typically weekly, and were critical in ensuring continuous guidance and support throughout the internship. Typical topics discussed included:

- **Task Review:** Evaluating progress on recently completed tasks and discussing the underlying approaches taken.
- **Current Focus:** Aligning on ongoing work and ensuring it met both short-term goals and long-term project objectives.
- **Planning Ahead:** Setting priorities for upcoming work, identifying learning opportunities, and defining strategic next steps.
- **Issue Resolution:** Identifying any blockers or technical challenges and collaboratively developing strategies to overcome them.

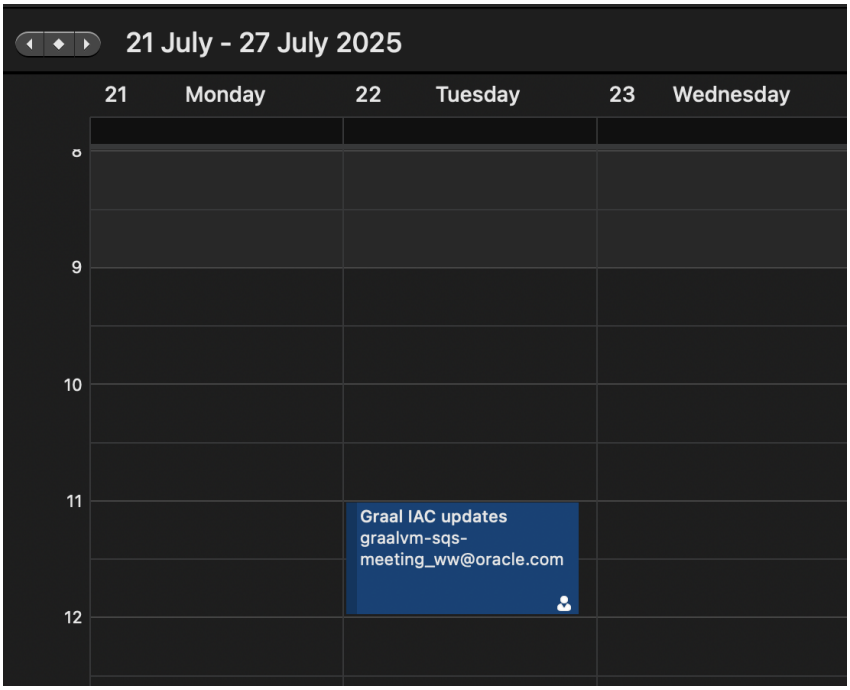

Figure 2.2: Mentor check-in workflow

2.1.3 Task Management

- **Jira:** An agile project management tool by Atlassian used for issue tracking, task decomposition, backlog management and progress monitoring. Each ticket included a short description, acceptance criteria and status to make priorities clear.

Jira Software

Figure 2.3: Jira backlog and task view

2.1.4 Documentation Tools

- **Confluence:** A web-based documentation and knowledge management platform by Atlassian. It stores design documents, meeting notes, runbooks and onboarding pages.

Figure 2.4: Confluence workspace

2.1.5 Source Code Management

- **Git:** Distributed version control used for collaborative development, enabling branching, merging and safe rollback.

Figure 2.5: Git version control

- **GitLab:** Hosted repositories, merge request reviews, and CI/CD pipelines [2]. GitLab pipelines ran automated tests, linting, and Pulumi preview/apply jobs for infrastructure changes.

Figure 2.6: GitLab CI/CD pipeline

2.1.6 Communication Tools

- **Slack:** Real-time messaging and small-group channels for feature work and incident response.
- **Zoom:** Video conferencing for design reviews, pair programming and demos.
- **Outlook:** Email and calendar coordination for formal scheduling and official communications.

Slack Zoom Outlook

Figure 2.7: Communication tools used for daily collaboration

2.2 On-boarding and integration process

Oracle MADC's onboarding process is designed to help new hires quickly adapt and contribute. Interns begin with immersive sessions in their first two days, followed by access to tools and support for smooth team integration. Using Jira and Slack, training, documentation, and system access are clearly organised and tracked, ensuring efficiency and reducing confusion. Slack also serves as a key channel for timely IT support.

Beyond administrative setup, the onboarding phase also introduced interns to Oracle's learning culture, through a blend of compliance training, hands-on technical tutorials, and access to world-class educational platforms. This comprehensive process laid a strong foundation for adapting to the fast-paced, collaborative engineering environment.

2.3 Mandatory Compliance Training

Compliance training was one of the first critical requirements, ensuring that all new interns aligned with Oracle's corporate policies and ethical standards. Key topics included:

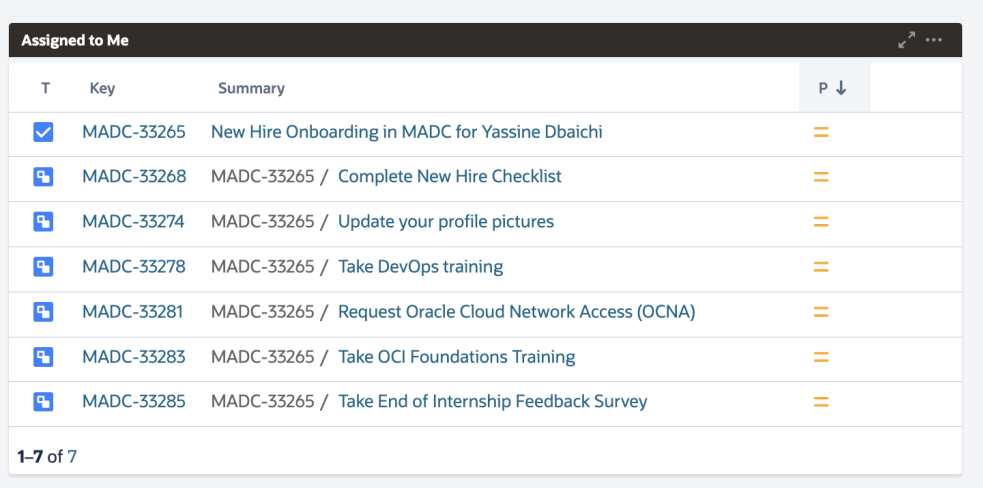

Figure 2.8: New hire onboarding Tickets

- Oracle's Code of Conduct and workplace ethics;
- Security policies for safeguarding sensitive information;
- Global data protection and privacy regulations;
- Responsible business practices and cybersecurity protocols.

This training ensured that every intern was prepared to act professionally and securely within Oracle's regulated environment.

2.4 Generic & Technical Training

A standout aspect of the onboarding experience was access to a rich suite of learning platforms, including:

- Oracle My Learn;
- LinkedIn Learning;
- Degreed;
- O'Reilly;
- Harvard ManageMentor.

These platforms offered curated content tailored to interns, covering technical, professional, and domain-specific topics. Key areas of study included:

- Oracle Cloud Infrastructure (OCI) [7];
- Git and Jira fundamentals;
- Linux, Bash, and Python scripting [11];
- DevOps principles [4];

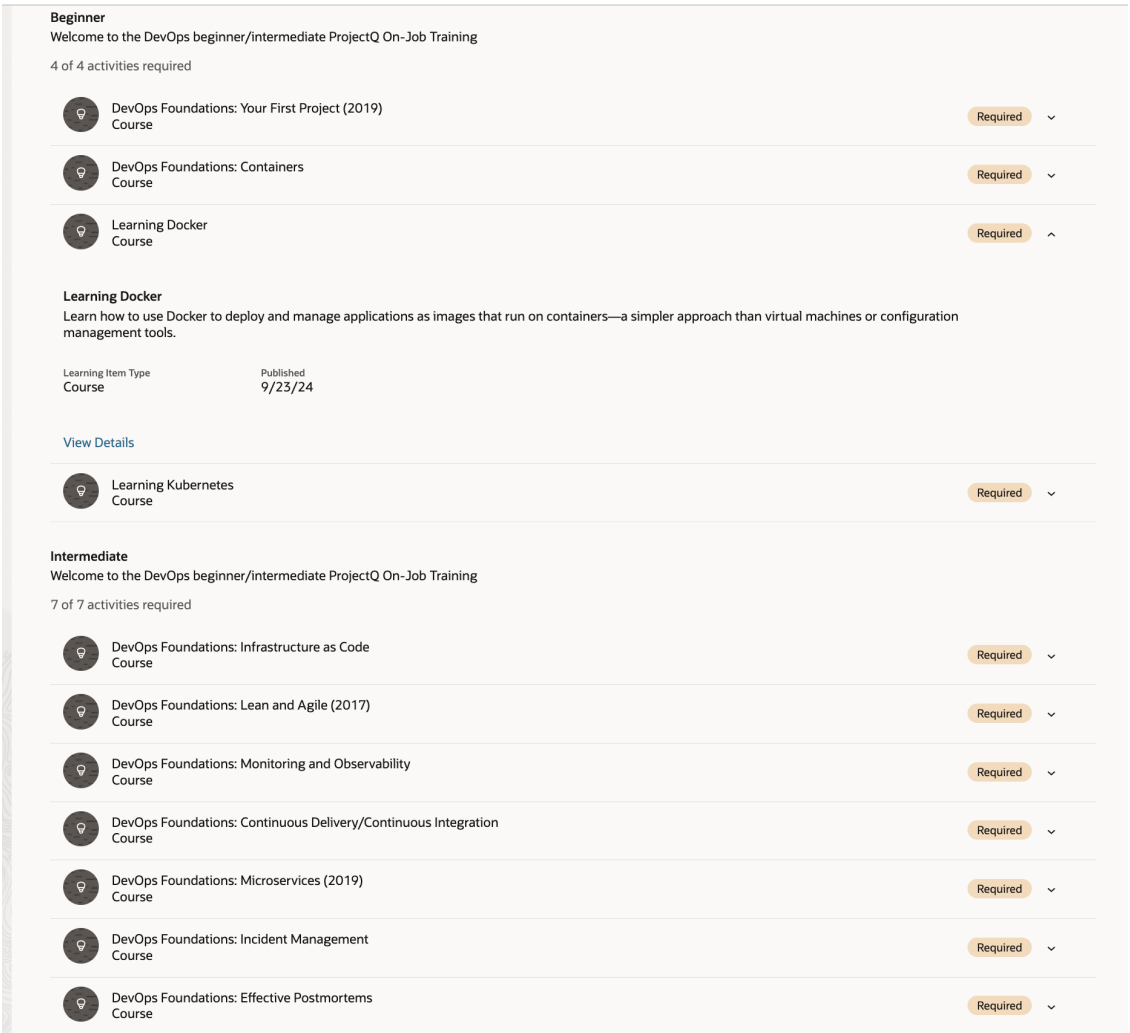

Figure 2.9: Training platforms and modules

Each course typically concluded with an assessment or certificate, fostering a sense of progress and readiness. The modular format enabled interns to build both foundational and advanced skills over time.

In addition to the generic technical training, project-specific courses were also assigned based on the intern's role and project focus. In my case, I received additional training in topics relevant to infrastructure automation and cloud provisioning [5], which prepared me for the IaC work described in later chapters. This included specific training on Pulumi [9] and its OCI provider integration [10].

2.5 Development Workflow

2.5.1 Task Lifecycle

Typical workflow for a development task was:

1. Create a ticket in Jira with a clear description and acceptance criteria.
2. Implement changes in a feature branch using Git.
3. Submit a merge request in GitLab [2] and request peer review.
4. Merge after approvals.
5. Close the Jira ticket and update documentation in Confluence.

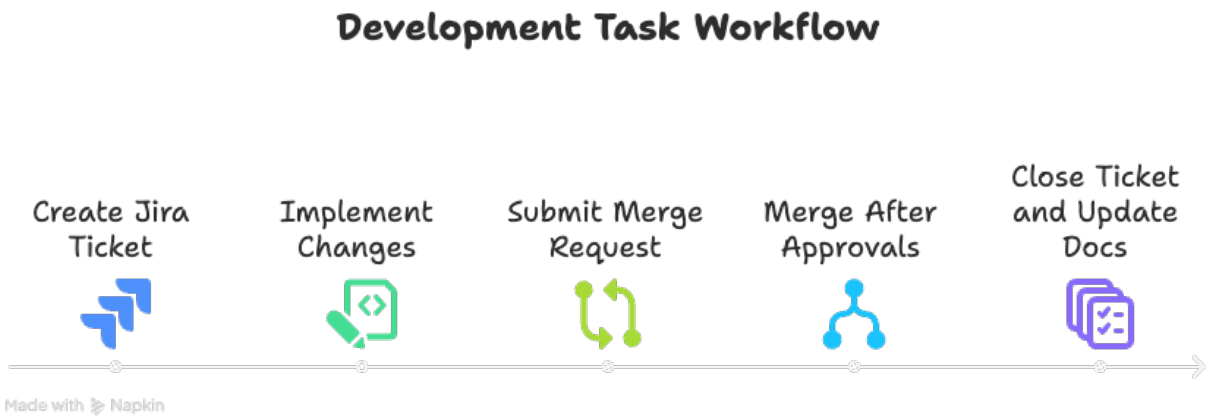

Figure 2.10: Task lifecycle from creation to completion

Conclusion

This chapter presented the project management approach: mentor oversight, regular meetings, collaboration tools (GitLab, Confluence, Jira, Slack), training on OCI and Pulumi, and iterative development practices. This structured methodology ensured quality and alignment throughout the project.

The next chapter analyzes the technical problems with manual infrastructure management, defines solution requirements, and justifies the selection of Pulumi through comparative evaluation of IaC tools.

Chapter 3

Current State Analysis and Solution Specification

Introduction

This chapter analyzes existing infrastructure management practices within the GraalVM RISQ team, identifies operational limitations, evaluates Infrastructure as Code solutions, and specifies the selected technical approach.

3.1 Current State Analysis

3.1.1 Manual Infrastructure Management

The GraalVM RISQ team manages Oracle Cloud Infrastructure (OCI) [7] resources through manual console operations:

Provisioning Process:

- Resources created through OCI Console web interface
- Sequential configuration of VCNs, subnets, gateways, and compute instances
- Configuration parameters manually copied between console sections
- Resource dependencies tracked manually
- Environment documentation stored in files

Resource Scope:

- Virtual Cloud Networks (VCNs) and subnets
- Internet and NAT gateways
- Security lists and route tables
- Compute instances for CI/CD workloads
- Oracle Kubernetes Engine (OKE) clusters and node pools
- Object storage buckets

CI/CD Integration: GitLab [2] pipelines assume infrastructure exists and is properly configured, with no automated validation or coordination between infrastructure and application deployment.

3.2 Identified Issues

The manual infrastructure management approach has exposed critical operational deficiencies that impact team productivity and system reliability. These issues can be categorized into two primary domains: operational inefficiencies and quality assurance challenges.

3.2.1 Operational Problems

Efficiency Issues:

- Environment provisioning requires a lot of time per stack
- Significant developer time consumed by repetitive tasks
- No parallel environment creation capability
- Infrastructure expertise concentrated among few team members

Process Issues:

- No version control for infrastructure changes
- Limited audit trail for modifications
- Complex onboarding process for new team members

3.2.2 Quality and Consistency Problems

Environment Drift:

- Development, staging, and production environments diverge over time
- Manual configuration introduces errors and inconsistencies
- Inconsistent resource naming and security configurations

Pipeline Limitations:

- Fragile pipelines due to infrastructure state assumptions
- No automated rollback capabilities
- Limited scalability for parallel development
- Manual resource cleanup leads to cost inefficiencies

3.3 Requirements

Based on the analysis of current limitations and team needs, the infrastructure automation solution must meet the following requirements, organized into functional and non-functional categories:

3.3.1 Functional Requirements

- JSON-based infrastructure definitions accessible to non-technical users
- Stack-based configuration system for multiple environments
- Version control integration for all specifications
- Automated resource creation, updates, and deletion
- Dynamic dependency resolution between resources
- GitLab CI/CD [2] pipeline integration with preview and approval workflows

3.3.2 Non-Functional Requirements

- Infrastructure provisioning in short time
- Modular architecture supporting new resource types
- Secure credential handling and access controls
- Comprehensive error handling and audit capabilities

3.4 Solution Evaluation

3.4.1 Technology Comparison

Aspect	**Terraform**	**Pulumi**
Language	HCL (domain-specific)	Python (team expertise)
JSON Support	Variable files only	Native configuration
CI/CD Integration	Manual setup required	Built-in GitLab support
Extensibility	HCL module system	Programming language flexibility
Learning Curve	Medium (new syntax)	Minimal (familiar concepts)
OCI Provider	Mature, comprehensive	Full-featured with Python SDK

Table 3.1: Terraform [3] vs Pulumi [9] comparison

3.4.2 Selection decision

Pulumi [9] with Python [11] was selected based on:

- Team's existing Python expertise enables immediate productivity
- Native JSON configuration meets accessibility requirements
- Good GitLab CI/CD integration capabilities [2]
- Flexible architecture supporting factory patterns [1] and dependency resoulution
- Strong OCI provider [10] support with active development

3.5 Selected Solution: Pulumi

3.5.1 Pulumi Overview

Pulumi [9] is an Infrastructure as Code [5] platform enabling cloud resource management using general-purpose programming languages instead of domain-specific configuration languages.

Core Concepts:

- **Programs:** Infrastructure defined as executable code in Python [11]
- **Providers:** Plugins interfacing with cloud APIs (OCI, AWS, Azure)
- **Stacks:** Isolated deployments for different environments
- **State:** Backend storage tracking infrastructure state and changes

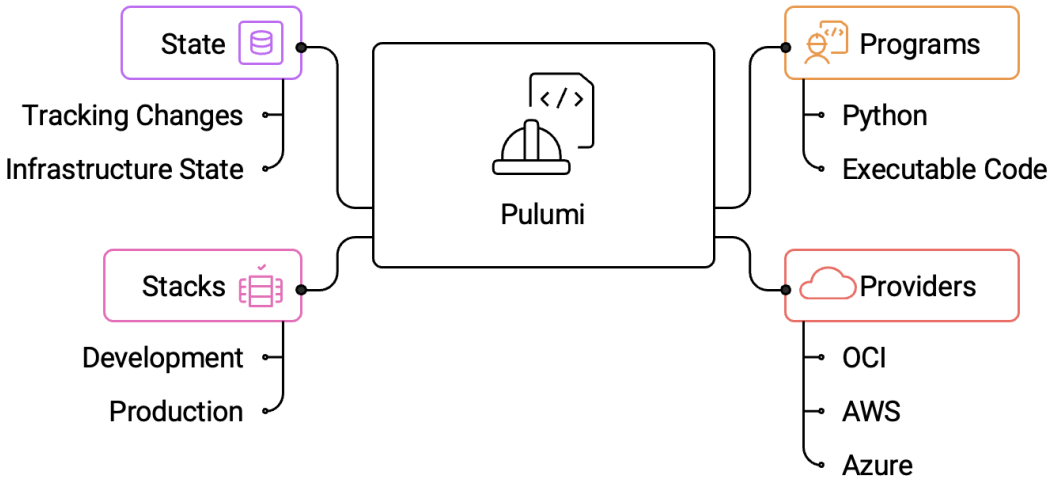

Figure 3.1: Pulumi Overview

3.5.2 Operational Model

Deployment Process:

1. Program execution constructs resource dependency graph
2. Plan generation identifies required changes
3. Resource provisioning through API calls with dependency ordering

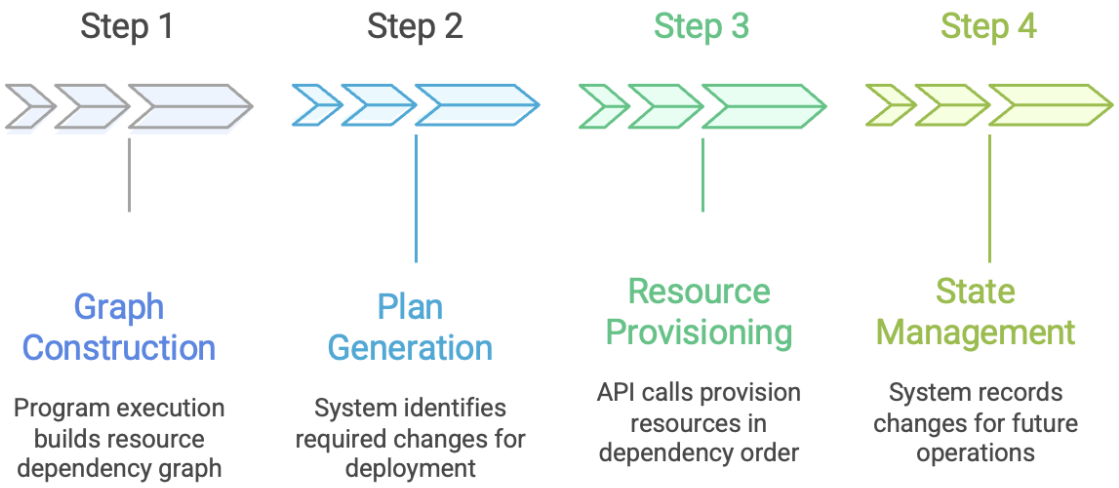

Figure 3.2: Pulumi Deployment Steps

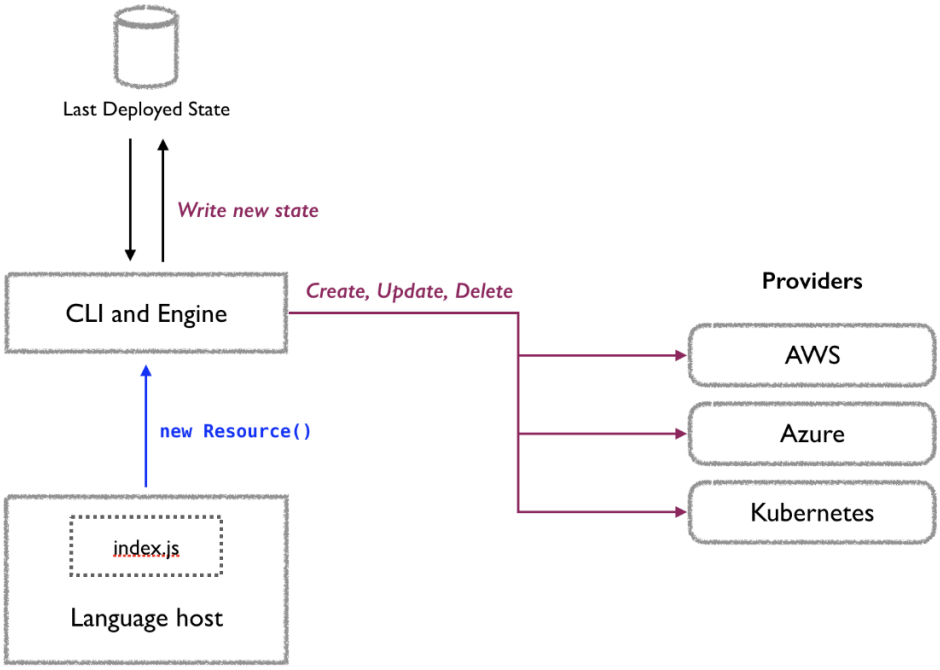

Figure 3.3: Pulumi Deployment engine

4. State management records changes for future operations

3.5.3 Architecture Design

Key Components:

- **State Backend:** Pulumi state is kept in OCI Object Storage [7] (S3-compatible) to store stack state remotely.
- **JSON Configuration:** Stack-specific JSON files hold the desired resources and parameters. Changes to these files start the provisioning flow.
- **Resource Factory:** A factory [1] maps JSON sections to resource handlers. This lets the system create resources from configuration.
- **Dependency Resolver:** Converts logical names in JSON to OCI identifiers and enforces the right creation order.
- **GitLab Integration:** GitLab CI [2] jobs run Pulumi commands (for example: preview and apply) and produce preview output for review.
- **Drift Detection:** A scheduled refresh job checks the real state against the JSON and creates drift html reports and sends them to the team when drift is found.
- **Import / Legacy Onboarding:** A discovery and import workflow maps existing OCI resources and helps add them to Pulumi state safely.

Provisioning workflow

1. **Initialize stack (one-time)** — Create the new stack with Pulumi, for example: `pulumi stack init <stack-name>`.
2. **Write config** — Create or edit `configs/<stack>.json` with logical names and resource definitions.
3. **Load config** — The Config Loader reads that JSON and gives the data to the engine.
4. **Resolve dependencies** — The Dependency Resolver finds references in the JSON and builds the correct creation order (topological order).
5. **Materialise resources** — The Resource Factory maps each JSON section to a resource handler and prepares arguments (using resolved IDs where needed).

6. **Execute with Pulumi** — The Execution Engine calls Pulumi to run the plan (for example: `pulumi preview` then `pulumi up`) and applies changes in OCI following the resolved order.

7. **Update state** — Pulumi writes the current stack state to the remote backend so the system knows what exists now.

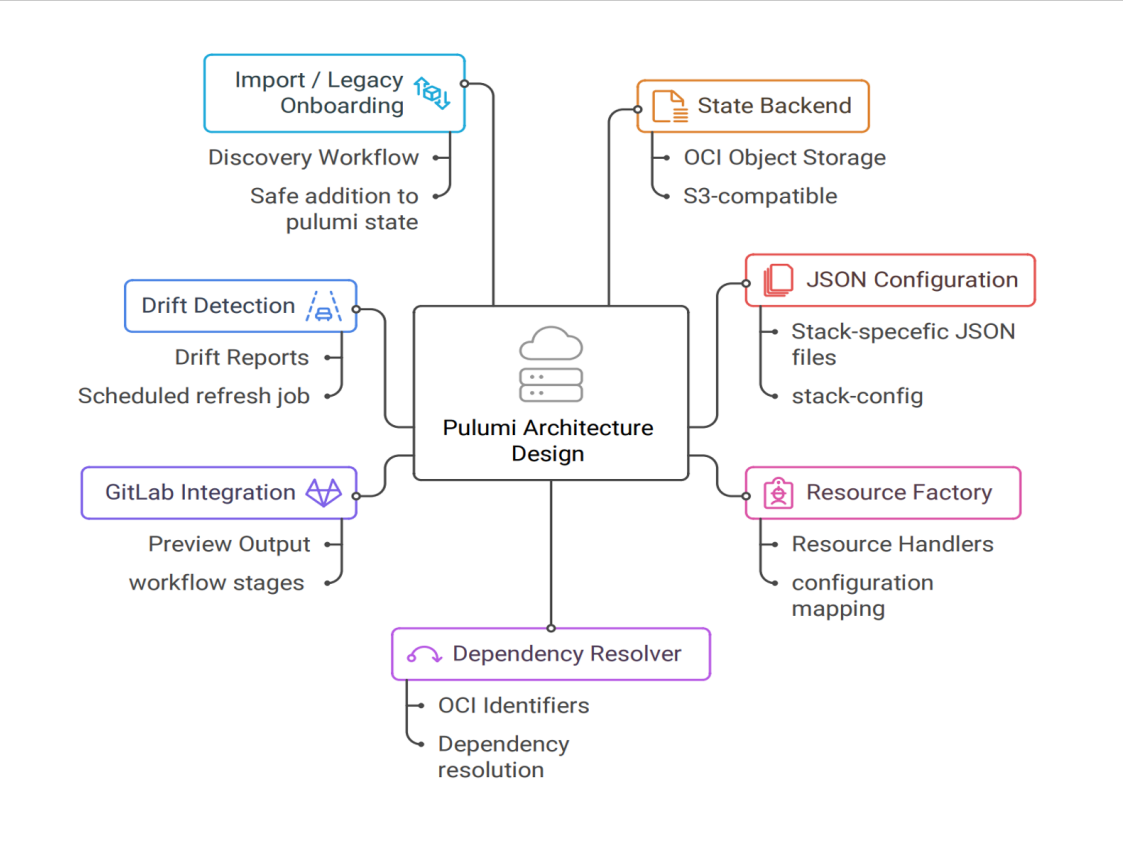

Figure 3.4: Architecture design.

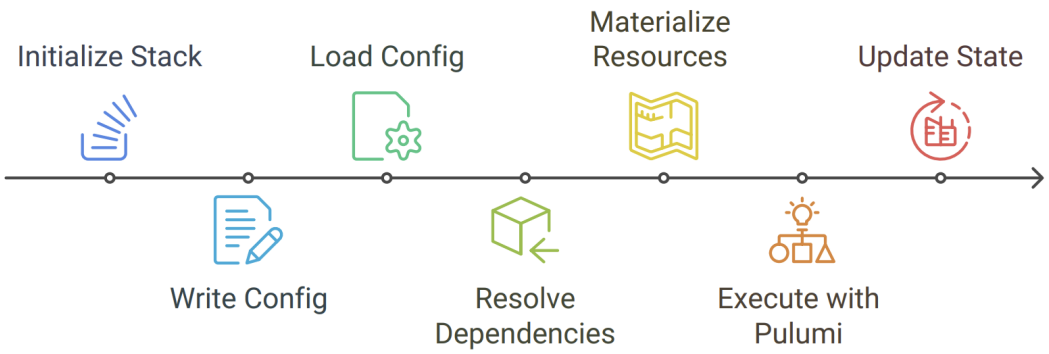

Figure 3.5: workflow design.

Conclusion

This chapter identified critical infrastructure management problems: time-consuming manual workflows, configuration inconsistencies, and lack of reproducibility. We defined functional and non-functional requirements, then selected Pulumi over Terraform, based on Python integration, OCI support, and CI/CD compatibility.

The next chapter presents the framework's architecture: Factory pattern implementation, dependency resolution, component structure, and the complete deployment workflow through GitLab CI/CD pipelines.

Chapter 4

Design and Implementation of the Solution

Introduction

This chapter presents the design and implementation approach for the JSON-driven Infrastructure as Code framework. It covers the solution design methodology, system architecture, technical design patterns, workflow implementation, and deployment architecture used to address the infrastructure automation needs identified in the previous chapter.

4.1 Solution Design Approach

4.1.1 Use Case Analysis

The framework supports three primary actors and their interactions with the infrastructure provisioning system:

System Actors:

- **Developer:** Team members who need infrastructure resources for development and testing
- **Infrastructure Administrator:** Team members responsible for reviewing and approving infrastructure changes
- **CI/CD System:** GitLab pipelines that automate the provisioning and management processes

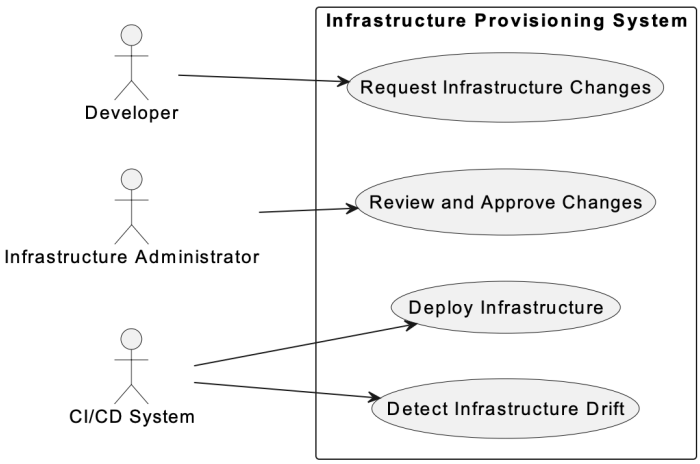

Figure 4.1: Use Case Diagram .

Primary Use Cases:

UC1 - Developer Infrastructure Request:

- Developer creates feature branch for infrastructure changes
- Edits appropriate `configs/{stack}.json` configuration file
- Submits pull request with infrastructure modifications
- System automatically generates preview of planned changes

UC2 - Infrastructure Review and Approval:

- Infrastructure administrator receives pull request notification
- Reviews proposed changes and examines preview artifacts
- Evaluates changes for security and compliance requirements
- Approves or rejects infrastructure modifications

UC3 - Automated Infrastructure Deployment:

- CI/CD system detects approved merge to protected branch
- Validates stack selection and configuration integrity
- Executes infrastructure provisioning through Pulumi
- Updates remote state and provides deployment feedback

UC4 - Infrastructure Drift Detection:

- Scheduled job compares actual infrastructure state with desired configuration
- Identifies changes between managed and actual resources
- Generates drift reports for team review

4.1.2 Design Principles

The framework implementation follows core architectural principles that ensure maintainability and extensibility:

Stateless Factory Pattern: All resource management operates through static class methods without object instantiation, reducing memory overhead and simplifying the execution model.

Dependency Injection: Resources reference each other through logical names, with the system handling automatic resolution to Oracle Cloud Infrastructure identifiers at runtime.

Stack-Based Isolation: Environment-specific configurations maintain strict separation between development, staging, and production deployments.

4.2 System Architecture

4.2.1 High-Level Architecture Overview

The Infrastructure as Code framework implements a layered architecture that separates configuration management, resource orchestration, and cloud provisioning into distinct components.

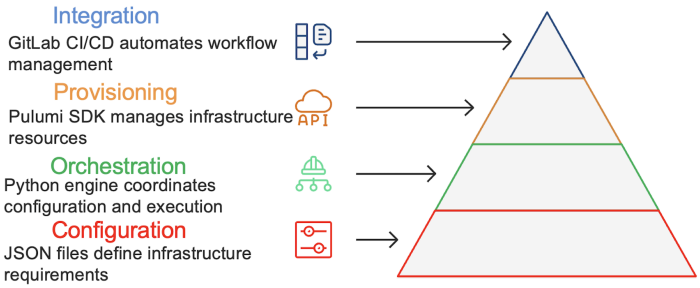

Figure 4.2: Architecture Layers .

Architecture Layers:

Configuration Layer: Stack-specific JSON files define infrastructure requirements using human-readable resource names and direct parameter mapping to Oracle Cloud Infrastructure services.

Orchestration Layer: Python-based engine coordinates configuration loading, dependency resolution, resource factory operations, and execution management through a single-loop processing model.

Provisioning Layer: Pulumi SDK interfaces with Oracle Cloud Infrastructure provider APIs to create, update, and manage infrastructure resources according to the processed configuration.

Integration Layer: GitLab CI/CD pipelines provide automated workflow management, including preview generation, approval gates, and deployment execution using a custom Docker image.

4.2.2 Component Architecture

The project consists of four primary components that work together to transform JSON configurations into deployed Oracle Cloud Infrastructure resources:

```
Initial-Pulumi-Setup/
├── Pulumi.yaml              # Pulumi project and stack metadata
├── Pulumi.dev.yaml          # Dev stack settings (config...)
├── configs/                 # Stack-specific configuration files
│   ├── dev.json            # Development stack resources
│   ├── prod.json           # Production stack resources
│   └── test.json           # Test stack resources
├── requirements.txt         # Python dependencies
├── venv/                    # Python virtual environment
├── .env.example             # Template for environment variables
├── .env.local               # Your personal environment variables (create this)
├── core/                    # Base_resource, factory, resolver
├── loaders/                 # JSON loader and .env reader
├── resources/               # Python classes for each OCI resource type
└── main.py                  # Entrypoint: one-loop provisioning logic
```

Figure 4.3: Project Structure.

ConfigLoader:

- Loads stack-specific JSON configuration files based on current Pulumi stack context
- Performs initial validation and error handling for configuration integrity
- Provides unified interface for configuration access across the system
- Maps stack names directly to configuration file paths

ResourceFactory:

- Maintains dynamic registry mapping JSON sections to Python resource implementation classes
- Enables resource creation without hardcoded dependencies on specific resource types
- Supports automatic registration of new resource types during module import
- Provides consistent creation interface across all supported Oracle Cloud Infrastructure services

DependencyResolver:

- Converts logical resource names to Oracle Cloud Infrastructure identifiers at runtime
- Handles both internal resource references and external Oracle Cloud Infrastructure identifiers
- Maintains registry of created resources for dependency lookup during provisioning

ExecutionEngine:

- Orchestrates single-loop resource creation in proper dependency order
- Coordinates interactions between configuration loading, factory operations, and dependency resolution
- Generates pipeline outputs and exports for integration with CI/CD workflows

4.3 Technical Design Patterns

4.3.1 Factory Pattern Implementation

The factory pattern enables dynamic resource creation without hardcoding specific resource types in the main execution logic, supporting extensibility and maintainability.

Registration Mechanism: Resource types register themselves with the factory during module import in the main execution file, creating dynamic mapping between JSON configuration sections and Python implementation classes. This approach enables adding new resource types without modifying the core framework logic.

Implementation Benefits:

- New Oracle Cloud Infrastructure resource types added without core framework modifications
- Consistent creation interface across all resource implementations
- Simplified testing and debugging of individual resource types
- Runtime extensibility supporting future Oracle Cloud Infrastructure services

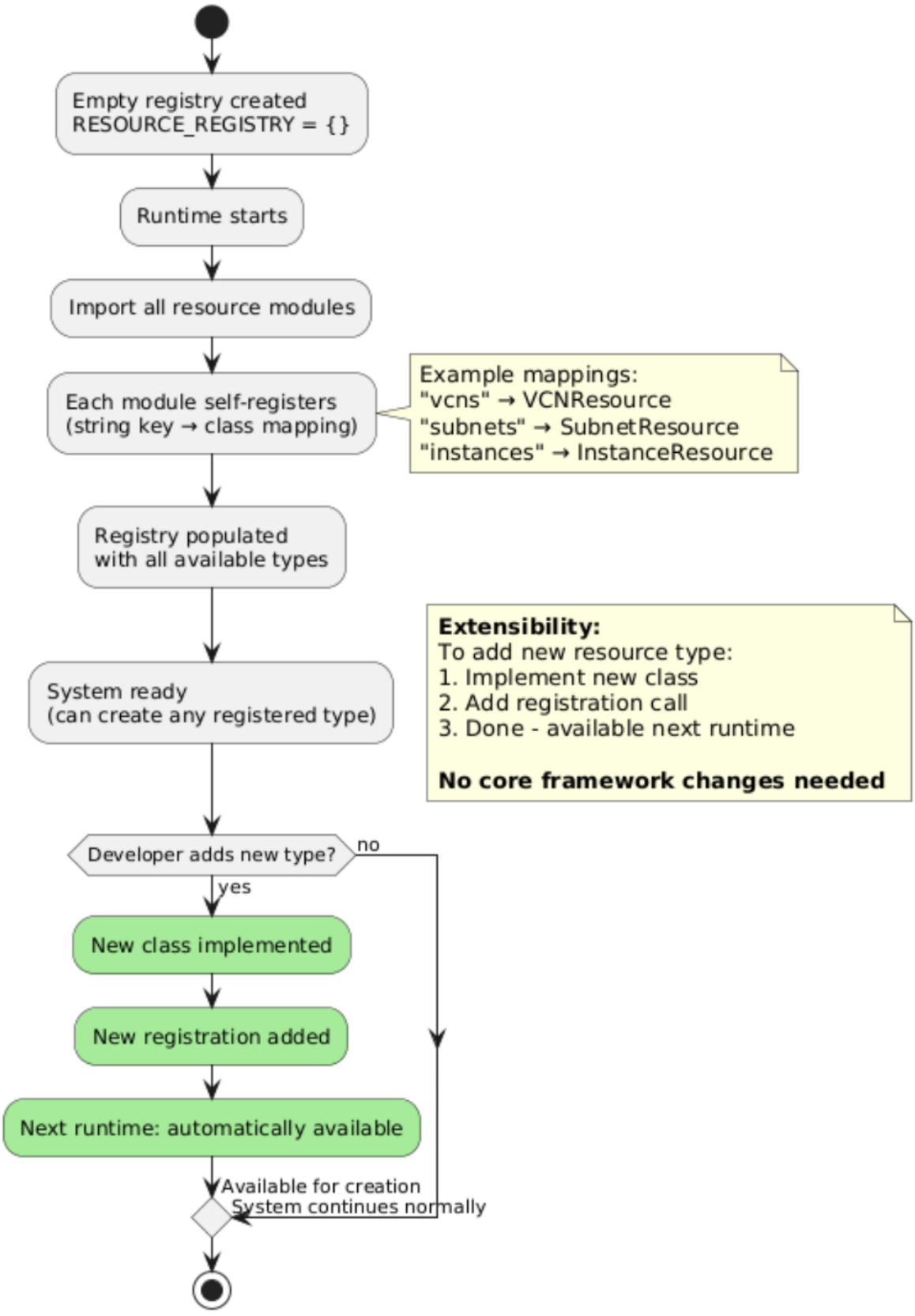

Figure 4.4: Resource Factory - Extensible Registration Mechanism .

4.3.2 Stateless Architecture Design

All resource classes implement static methods exclusively, eliminating object instantiation throughout the system and creating a stateless execution environment.

Design Advantages:

- Reduced memory overhead during resource creation processes
- Elimination of object lifecycle management complexity
- Simplified unit testing through direct class method invocation
- Consistent implementation patterns across all resource types

4.3.3 Dependency Resolution Strategy

The dependency resolver handles both logical resource names and direct Oracle Cloud Infrastructure identifiers, providing flexibility for hybrid infrastructure scenarios where some resources exist externally.

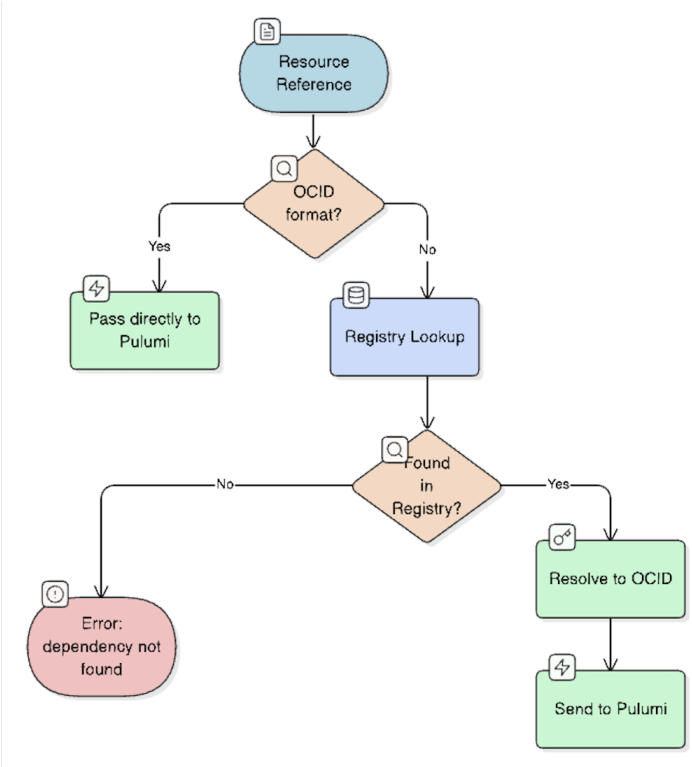

Figure 4.5: dependancy resolution workflow .

Resolution Logic:

- Oracle Cloud Infrastructure identifiers beginning with "ocid1." pass through directly for external resources

- Logical names resolve through internal resource registry maintained during execution
- Missing dependencies trigger clear error messages with specific resource identification

4.3.4 Class Hierarchy and Inheritance

The framework implements a template method pattern with an abstract base class and specialized implementations for each Oracle Cloud Infrastructure resource type.

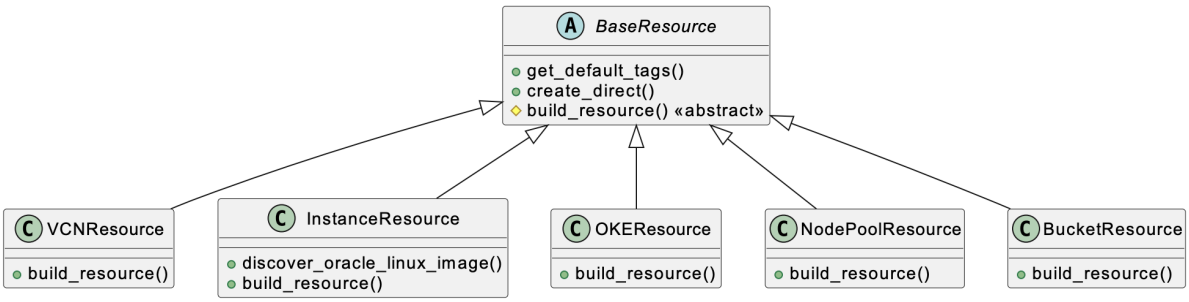

Figure 4.6: dependancy resolution workflow .

Abstract Base Class Design:

- Defines common resource creation interface through template method pattern
- Provides shared functionality including resource tagging and validation
- Enforces consistent implementation patterns across all resource types
- Enables polymorphic resource handling within the factory system

4.4 Infrastructure Provisioning Workflow

4.4.1 End-to-End Workflow Design

The infrastructure change management process integrates with standard development workflows through GitLab merge requests, providing controlled and auditable infrastructure modifications.

Workflow Stages:

Configuration Stage: Developer creates feature branch and modifies appropriate stack-specific JSON configuration file with new or updated infrastructure requirements, then submits pull request for review.

Preview Stage: GitLab CI automatically triggers preview job using custom Docker image, loading stack-specific configuration and generating Pulumi deployment plan without making infrastructure changes.

Review Stage: Infrastructure administrator examines proposed changes through preview artifacts, evaluating modifications for security, compliance, and operational impact before approval.

Deployment Stage: Approved merge triggers automated deployment pipeline with stack validation, resource provisioning through Pulumi orchestration, and remote state updates in Oracle Cloud Infrastructure Object Storage.

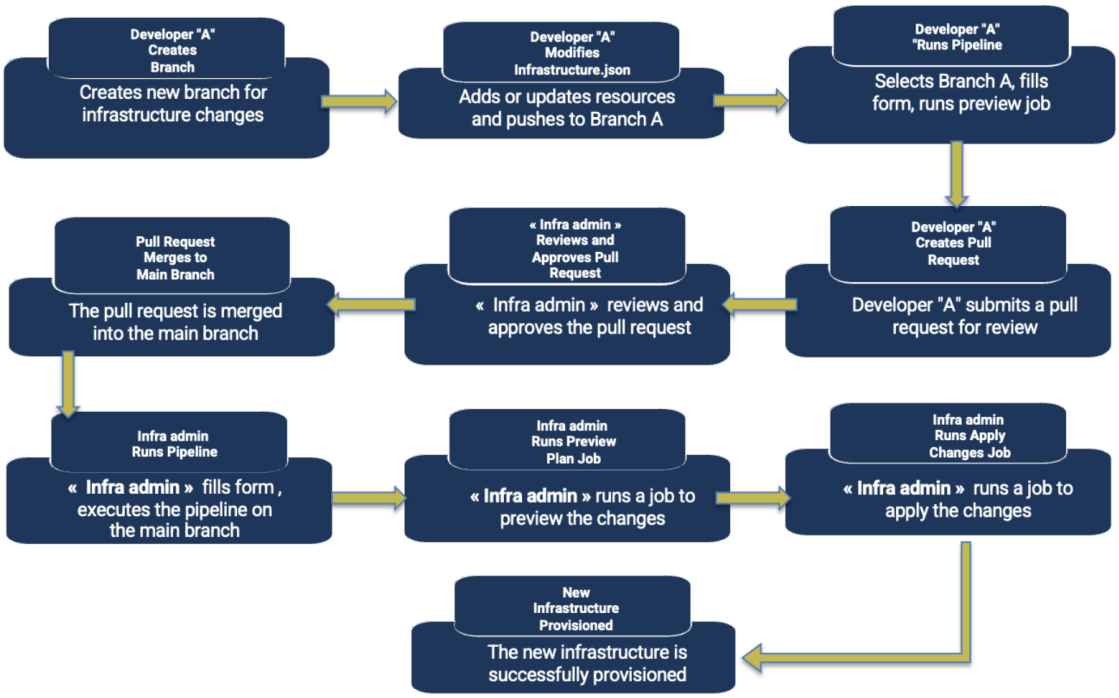

Figure 4.7: End-to-End Workflow Design.

4.4.2 GitLab CI/CD Pipeline Integration

The framework integrates with GitLab CI/CD through specialized pipeline stages that handle preview generation, approval gates, and deployment execution using a custom Docker image containing all required dependencies.

Custom Docker Image: The team uses a custom Docker image containing Pulumi CLI, Python runtime, Oracle Cloud Infrastructure CLI, and the complete framework implementation. This image ensures consistent execution environments across all GitLab CI jobs and simplifies dependency management.

Pipeline Stage Design:

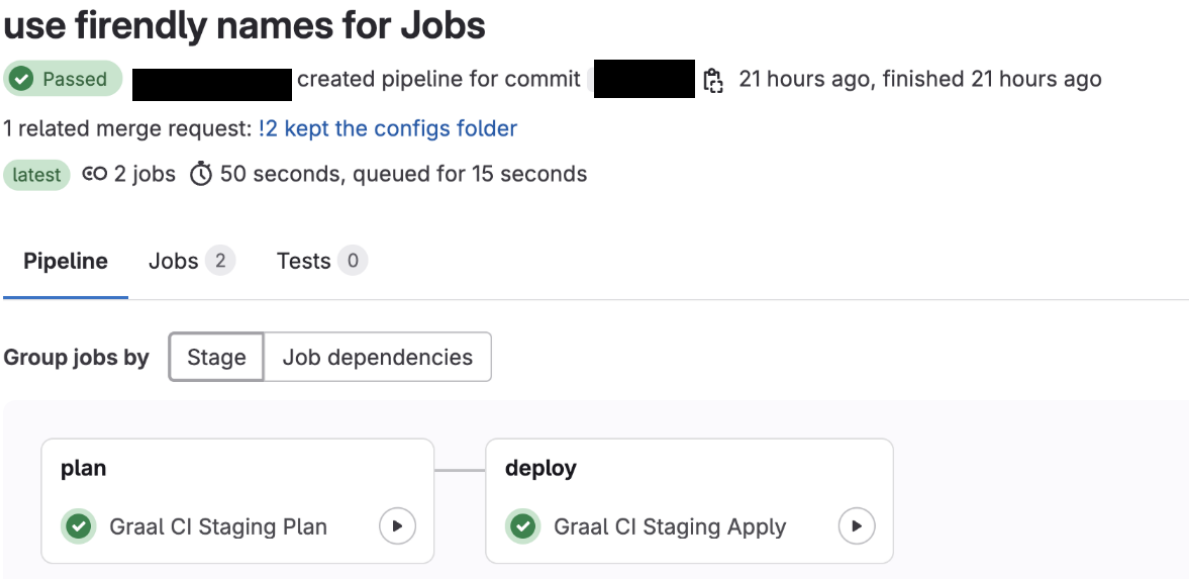

Figure 4.8: GitLab Pipeline Stages .

- **Preview Stage:** Automatically triggered on pull request creation, performs stack selection based on branch naming conventions, validates configuration, and generates preview artifacts
- **Apply Stage:** Requires manual approval for protected stacks, performs stack verification before provisioning, executes comprehensive logging, and synchronizes with remote state backend

4.4.3 Stack-Based Configuration Management

The framework implements environment isolation through Pulumi's stack concept, with each environment corresponding to separate configuration files and state management.

Stack Structure Implementation:

- `configs/dev.json` - Development environment resource definitions
- `configs/staging.json` - Staging environment resource definitions
- `configs/prod.json` - Production environment resource definitions

Stack Selection Logic: GitLab CI pipeline determines stack selection based on branch naming: main branch targets production stack, while feature branches target staging stack. The ConfigLoader reads the appropriate configuration file based on the selected Pulumi stack context.

4.5 Implementation Architecture

4.5.1 Core Component Implementation

The four core components implement specific responsibilities within the single-loop execution model, coordinating to transform JSON configuration into deployed Oracle Cloud Infrastructure resources.

Single-Loop Execution Model: The ExecutionEngine processes resources in hardcoded dependency order to ensure proper provisioning sequence: Virtual Cloud Networks, Internet Gateways, Security Lists, Route Tables, Subnets, Compute Instances, Oracle Kubernetes Engine clusters, and Object Storage buckets.

Configuration Processing: ConfigLoader determines appropriate configuration file based on Pulumi stack context, performing validation and providing error handling for configuration integrity issues.

Dynamic Resource Creation: ResourceFactory maps JSON sections to implementation classes through automatic registration during module import, enabling extensible resource support without core framework modifications.

Dependency Management: DependencyResolver converts logical resource names to Oracle Cloud Infrastructure identifiers, maintains registry of created resources, and handles both internal references and external identifier passthroughs.

4.5.2 JSON Configuration Structure

The system uses structured JSON format with logical name references enabling loose coupling between resources while maintaining direct parameter mapping to Pulumi Oracle Cloud Infrastructure provider specifications.

Configuration Features:

- Direct mapping to Pulumi Oracle Cloud Infrastructure provider parameters
- Human-readable resource definitions accessible to non-technical team members
- Logical name references enabling flexible resource relationships
- Environment-specific configurations supporting multiple deployment targets

Example Configuration Structure: The JSON configuration directly maps to Oracle Cloud Infrastructure resource parameters, with logical names like "production-vcn" resolved automatically to actual Oracle Cloud Infrastructure identifiers during provisioning.

4.5.3 Custom Docker Image Integration

The implementation uses a custom Docker image hosted in Oracle's internal registry, containing all required dependencies for GitLab CI job execution.

Image Composition:

- Pulumi CLI with Oracle Cloud Infrastructure provider support
- Python runtime with required framework dependencies
- Oracle Cloud Infrastructure CLI for authentication and state management

GitLab Integration: All GitLab CI jobs use the custom Docker image, ensuring consistent execution environments and eliminating dependency management issues across different pipeline stages.

4.6 State Management and Deployment

4.6.1 Pulumi State Storage Architecture

The framework stores Pulumi state in Oracle Cloud Infrastructure Object Storage using S3-compatible API, enabling team collaboration and providing centralized state management.

State Storage Implementation:

- Remote state files stored in dedicated Oracle Cloud Infrastructure Object Storage bucket
- S3-compatible API access using Oracle Cloud Infrastructure credentials
- Stack-specific state files maintaining environment isolation
- Centralized storage supporting team collaboration and audit requirements

State Management Benefits: Centralized storage enables team collaboration, provides version history for rollback capabilities, maintains secure storage within Oracle Cloud Infrastructure, and supports audit trails for compliance requirements.

4.6.2 Stack Selection Mechanism

The system implements stack selection through GitLab CI environment variables, with branch-based logic determining appropriate target environments.

GitLab CI Stack Selection: GitLab CI pipeline sets PULUMI_STACK environment variable based on branch name, executes pulumi stack select command, and ConfigLoader uses pulumi.get_stack() to determine appropriate configuration file.

Local Development Support: Local testing supports manual stack selection, with ConfigLoader automatically reading corresponding configuration files based on selected Pulumi stack context.

4.6.3 Error Handling and Recovery

The framework implements error handling for common failure scenarios, providing clear feedback for troubleshooting and resolution.

Dependency Resolution Errors: When logical name lookup fails, the system raises Python KeyError with specific resource identification, enabling clear troubleshooting and resolution guidance.

Configuration Validation: Initial JSON parsing and validation provide early error detection, preventing deployment failures due to configuration syntax or structure issues.

Resource Creation Failures: Pulumi error handling provides detailed logging for resource provisioning failures, with state management ensuring consistent recovery and rollback capabilities.

Conclusion

This chapter detailed the framework's design: stateless factory architecture, dependency resolution converting names to OCIDs, strict resource creation ordering, and component architecture (BaseResource, ResourceFactory, DependencyResolver, ConfigLoader, EnvironmentManager). GitLab CI/CD integration enables automated deployments with remote state management in OCI Object Storage.

The next chapter presents validation results: testing methodology, performance improvements, operational impact, and identified limitations for future enhancement.

Chapter 5

Results and Validation

Introduction

This chapter presents the results obtained from implementing the Infrastructure as Code (IaC) framework in the Graal CI environment. It covers the delivered features, validation testing, team adoption, and quantifiable improvements. The section also analyzes framework extensibility and discusses current limitations and future perspectives.

5.1 Implementation Results

5.1.1 Delivered Features

The following milestones were achieved during the project:

- **JSON-Driven Automation**: Infrastructure fully declarative via per-stack JSON files.
- **Dynamic Factory Pattern**: Modular codebase allows adding resource types without modifying core logic.
- **Full CI/CD Integration**: Infrastructure provisioning, preview, and drift detection jobs seamlessly integrated into GitLab pipelines.
- **Multi-Environment Support**: Isolated development, staging, and production stacks with shared framework logic.
- **Resource Coverage**: Support for VCNs, subnets, compute instances, security lists, gateways, object storage buckets, and OKE clusters.

- **Remote State Management**: Pulumi state files securely managed in OCI Object Storage, supporting collaborative workflows.

5.1.2 Graal CI Integration Achievements

- **Seamless Adoption**: Pipelines extended with infrastructure jobs, requiring no changes for application developers.
- **Approval Gates**: Infrastructure changes only applied after manual review.
- **Artifact Management**: State and outputs stored for traceability and reproducibility.
- **Team Documentation**: Readme and onboarding guides provided for new users.

5.2 Testing and Validation Results

5.2.1 Infrastructure Deployment Testing

The solution was validated with real deployments and CI pipeline runs:

- **Network Creation**: VCNs with defined CIDRs successfully provisioned in test/production compartments.
- **Instance Deployment**: Compute instances launched in correct subnets, with public IPs and SSH access verified.
- **Resource Linking**: Gateways and route tables correctly associated with respective VCNs.
- **Storage and Buckets**: Object storage buckets created and accessible for CI artifacts.

Table 5.1: Validation Scenarios and Outcomes

Test	**Scenario**	**Result**
VCN Creation	New VCN, IGW, security list	Success; all OCI resources present
Instance Deploy	Instance in subnet, SSH access	Success; SSH connection validated
Drift Detection	Manual change in OCI console	Success; Pulumi detected drift
CI/CD Preview	Add resource in JSON, PR preview	Success; pipeline output as expected

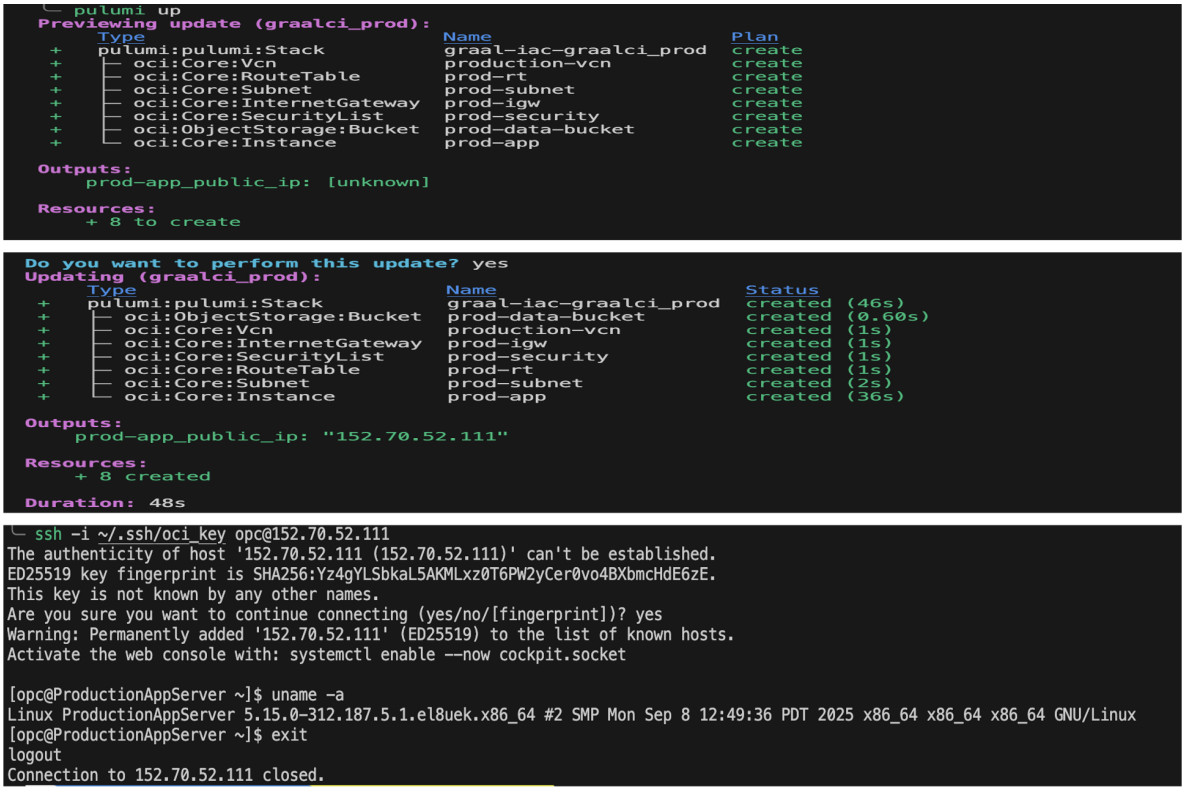

Figure 5.1: Instance Deployed Successfully.

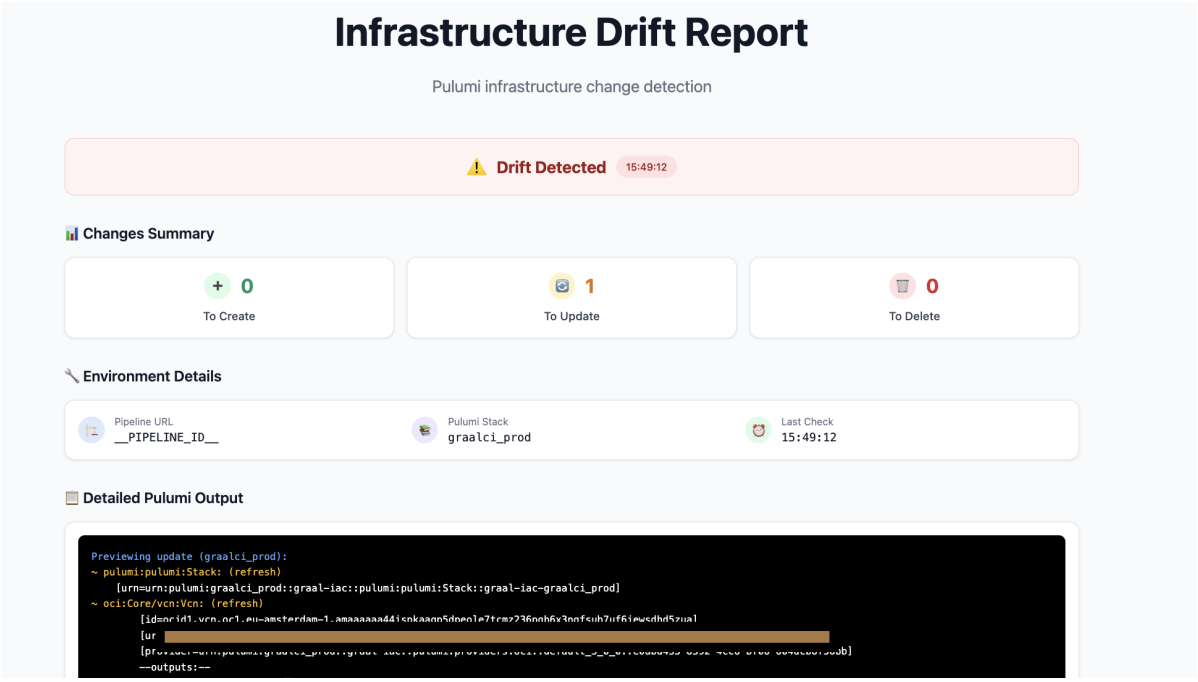

Figure 5.2: Drift Detection Report .

5.2.2 CI/CD Pipeline Validation

- **Preview Accuracy**: All planned changes shown before deployment, enabling effective code reviews.

- **Approval Workflow**: Infrastructure deployed only after team approval, preventing accidental changes.
- **Branch Targeting**: Main → production; feature branches → staging or dev.
- **Error Handling**: Failures surfaced in pipeline logs for immediate resolution.

5.3 Performance, Reliability, and Impact

5.3.1 Deployment Efficiency

- Average environment provisioning time reduced .
- Elimination of manual errors and configuration drift.
- Consistency across all environments (dev, staging, production).

5.3.2 Team Adoption and Experience

- **Developer Onboarding**: Reduced learning curve — edit JSON and submit PR to request infrastructure.
- **Self-Service**: Team members can provision environments without direct cloud console access.
- **Traceability**: All changes logged via Git history and pipeline state.
- **Rollback**: Environment rollbacks achieved via Git revert + pipeline rerun.

5.4 Framework Extensibility Demonstration

5.4.1 Adding New Resource Types

The factory pattern allowed rapid addition of new OCI resources:

1. Implement new Python class for the resource (e.g. Load Balancer).
2. Register in ResourceFactory.
3. Add corresponding JSON section.
4. No modification to core framework required.

5.4.2 Current Limitations and Future Work

- **JSON Schema Validation**: Currently, invalid config will cause runtime failure; schema-based pre-validation is a future goal.
- **Hardcoded Resource Order**: Dependency order fixed in code; can be made dynamic.
- **Additional Resources**: Adding advanced services (databases, monitoring, etc.) is straightforward but not yet implemented.
- **Advanced Testing**: More comprehensive unit/integration tests planned for pipeline robustness.

5.5 Project Impact Assessment

5.5.1 Workflow Comparison

Table 5.2: Development Workflow Improvements

Aspect	**Before (Manual)**	**After (Automated)**
Environment Setup	1 hour	15–20 min
Consistency	Varies by user	Identical every time
Change Tracking	No record	Complete Git history
Rollback	Manual recreation	Git revert + redeploy

5.5.2 Quantifiable Benefits

- Significant time savings per environment created.
- Reduced bottlenecks — in Infra team.
- Better collaboration — clear, auditable change history.

Conclusion

This chapter presented the validation approach for the framework. Testing included deployment and destruction cycles across different resource types (VCNs, subnets, instances, OKE clusters, buckets) to verify correct provisioning and dependency handling. The JSON-driven approach proved functional for personal testing scenarios, eliminating manual console errors and enabling reproducible deployments.

The general conclusion synthesizes the project achievements and outlines perspectives for future team adoption and framework enhancements.

General Conclusion

Project Achievements

This project achieved its main objective: enabling automated, scalable, and auditable cloud infrastructure provisioning for the GraalVM RISQ team through Infrastructure as Code (IaC) using Pulumi and Oracle Cloud Infrastructure (OCI).

Key outcomes include:

- **JSON-Driven Framework**: Designed and implemented a flexible, extensible IaC system, where all infrastructure is declaratively defined in JSON, enabling non-expert users to request changes via simple file edits.
- **Factory Pattern**: Established a modular resource factory supporting rapid addition of new resource types without modifying the core engine, improving long-term maintainability.
- **GitLab CI/CD Integration**: Seamlessly incorporated infrastructure management jobs (preview, apply, drift detection) into existing CI/CD pipelines, introducing code review, approval gates, and state management via OCI Object Storage.
- **Team Enablement**: Lowered the barrier for team members to request, review, and provision infrastructure, supporting agile, collaborative workflows and reducing bottlenecks related to manual provisioning.
- **Consistency and Traceability**: Achieved reproducible, version-controlled environments across all stages (development, staging, production), with complete audit trails for every change.

Technical Innovation

The project demonstrated several technical innovations:

- **Custom Dependency Resolution**: Logical name-to-OCID mapping allowed users to reference resources naturally in configuration files, while the engine handled all underlying ordering and relationships.
- **Environment Isolation**: Multi-stack setup enabled isolated, parallel environments for development, testing, and production, minimizing risk and supporting concurrent workflows.
- **CI/CD-First Approach**: All provisioning, changes, and rollbacks are performed via code and pipelines—no manual cloud console steps—maximizing automation and reducing errors.
- **Drift Detection**: Daily checks ensure infrastructure defined in code matches real-world OCI state, alerting the team to any out-of-band changes.

Organizational Impact

- **Faster Delivery**: Environment setup times reduced from hours or days to minutes, directly accelerating GraalVM development and testing cycles.
- **Improved Collaboration**: Team members collaborate through pull requests, with clear ownership, code review, and accountability for all infrastructure changes.
- **Knowledge Sharing**: Onboarding new developers is simplified through documentation and unified workflows; knowledge is embedded in code, not isolated with individual experts.
- **Operational Excellence**: Automated cleanup, rollback, and auditability contribute to more robust, cost-effective, and reliable operations.

Limitations and Future Perspectives

While the framework is production-ready, several areas can be improved:

- **Schema Validation**: Implementing JSON Schema checks will help catch configuration errors before pipeline runs.

- **Dynamic Dependency Graph**: Moving from a hardcoded creation order to a fully graph-based dependency resolver will allow even more flexibility and fewer manual steps.
- **Extended Resource Coverage**: Supporting additional resource types (databases, load balancers, advanced networking, monitoring) to match evolving team needs.
- **Advanced Security and Policy**: Integration with IAM policies, automated key management, and cost controls.
- **Cross-Region and Multi-Cloud**: Adapting the framework to support hybrid and multi-cloud environments, if required in the future.

Personal and Team Learning Outcomes

This project provided significant hands-on experience in:

- Infrastructure as Code best practices and real-world automation
- Pulumi, OCI, and Python development in an enterprise context
- CI/CD pipeline integration and DevOps workflows
- Software design patterns for extensibility and maintainability
- Agile collaboration, team onboarding, and technical documentation

Final Words

The successful delivery of this project demonstrates the power of Infrastructure as Code and collaborative DevOps practices to transform the speed, reliability, and traceability of cloud development. By automating infrastructure provisioning and integrating it tightly with the development lifecycle, the GraalVM RISQ team is now better equipped to scale, innovate, and deliver high-quality solutions at pace.

The next steps involve continuous improvement, incorporating feedback, and adapting the framework to future organizational needs. This work lays a solid foundation for sustainable, secure, and agile infrastructure operations in Oracle's cloud ecosystem.

Webography

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[2] GitLab Inc. Gitlab ci/cd documentation, 2025.

[3] HashiCorp. Terraform documentation, 2025.

[4] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010.

[5] Kief Morris. *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, 2016.

[6] Oracle Corporation. Graalvm documentation, 2025.

[7] Oracle Corporation. Oracle cloud infrastructure documentation, 2025.

[8] Oracle Corporation. Oracle labs, 2025.

[9] Pulumi Corporation. Pulumi documentation, 2025.

[10] Pulumi Corporation. Pulumi oracle cloud infrastructure provider, 2025.

[11] Python Software Foundation. Python 3 documentation, 2025.