POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Enabling Controlled Access to Physical Cloud Resources in a Bare Metal Cluster

Supervisor
prof. Fulvio Giovanni Ottavio Risso

Candidate
Giovanni Mirarchi

October 2025

A Naomi, compagna
in ogni passo di
questo cammino.

Summary

This thesis presents two innovative systems for sharing computational resources across Italian academic institutions within the RESTART partnership. The first system, named Prognose, is a web portal implementing a general resource reservation platform that enables users to discover availability and book any type of resource—both physical (bare-metal servers, storage, network ports) and virtual (VMs, containers, services)—across multiple university sites. The system implements an event-driven architecture with multi-site management, role-based access control (RBAC), and resource lifecycle automation through Metal3 and OpenStack Ironic.

The second system provides a GPU sharing infrastructure based on Kubernetes and KubeRay, enabling the creation of on-demand Ray clusters for GPU-intensive workloads. The architecture includes horizontal autoscaling of worker groups, declarative resource management through Custom Resource Definitions, and a shared NFS filesystem to simplify data management.

Both systems address current limitations in cross-institutional bare-metal resource access, characterized by fragmented manual processes, limited visibility into resource availability, and lack of tenant isolation. The proposed solution introduces self-service discovery and booking, clean and reproducible handovers, and integration with federated identity systems while respecting institutional autonomy.

The implementation demonstrates how mature components like Keycloak, PostgreSQL, and Metal3 can be combined through event-driven architectures and webhook-based integration to create unified resource management systems. Results show significant improvements in resource utilization, reduced setup times, and consistent user experience across different sites within the RESTART partnership.

Contents

Chapter 1

Introduction

1.1 Cloud Computing Evolution

Cloud computing arose from advances in distributed systems and virtualization. Early data centers ran monoliths on dedicated servers, yielding low utilization and slow provisioning. The virtual machine (VM) was the pivotal shift: a hypervisor decouples workloads from hardware so the full OS and application stack runs on a stable abstraction. VMs make workloads portable, right-sizeable, and isolated; golden images, snapshots, and live migration turned racks of servers into elastic pools manageable via software.

Building on the VM, infrastructure became programmable capacity: self-service APIs replaced tickets, image-based builds replaced bespoke setups, and quotas with metering enabled accountable sharing. This shift led to layered offerings with different responsibility splits.

At the service layer:

- **Infrastructure as a Service (IaaS)** exposes compute, storage, and networking as programmable primitives.
- **Platform as a Service (PaaS)** abstracts runtimes and managed data services to speed delivery.
- **Software as a Service (SaaS)** delivers complete applications via subscription.

At the deployment layer, public, private, and hybrid clouds trade off control, cost, and compliance; multi-cloud diversifies risk and capabilities.

The cloud-native era refined the model: containers standardized packaging and isolation; orchestrators automated placement and scaling; declarative

APIs and operators encoded operations; serverless introduced event-driven execution with per-invocation billing.

For performance-sensitive workloads, organizations also embraced *bare metal* for predictable latency and direct access to accelerators. Modern control planes brought image management, automated provisioning, policy enforcement, and secure wipe to physical machines, showing that "cloud" is an operational model spanning virtual and physical resources.

Operating practice evolved (Infrastructure as Code/GitOps, SRE, FinOps, Zero Trust), while edge and distributed cloud moved compute closer to data, encouraging hybrid designs and federation across organizations. In research settings, shared HPC/AI infrastructure faces heterogeneous workloads and bursty demand, requiring elasticity, fair sharing, and auditable multi-tenancy.

Against this backdrop, self-service access to bare-metal with time-bounded reservations and federated identity lets institutions pool high-value assets, raise utilization, and respect local constraints. In short, cloud computing progressed from VM-driven consolidation to a programmable fabric across virtual and physical resources, centralized regions and edges, and single-tenant silos to federated, multi-institutional platforms.

1.2 Resource Management Challenges

Managing shared compute across institutions and sites exposes technical and organizational constraints. Key challenges include:

- **Resource abstractions**: The system must model heterogeneous assets: baremetal servers, GPUs/accelerators, VMs, storage, and switch ports and allow users to reserve them individually or as bundles with explicit capabilities and constraints.
- **Isolation and clean handover**: Each reservation requires a clean, reproducible setup (imaging, configuration, secure wipe), plus network and storage isolation to prevent leakage and ensure predictable environments.
- **Identity federation and access control**: Cross-institution authentication with RBAC and group mapping should preserve local autonomy while enabling controlled collaboration and auditable, least-privilege access.
- **Time-bounded reservations and scheduling**: Policies for conflict detection and resolution enable safe and efficient utilization.
- **Interoperability and portability**: Open, extensible APIs with adapter-driven integrations (e.g., Kubernetes, webhooks, imaging pipelines) make the platform provider-agnostic and allow institution-specific logic without forking.

These challenges motivate a platform that treats diverse resources as schedulable objects, automates lifecycle and isolation, integrates with federated identity, and exposes open interfaces for institution-specific extensions.

1.3 Thesis Scope

This thesis, within the RESTART partnership, delivers two independent systems for sharing computational resources across Italian universities. They address distinct use cases and are not integrated with each other.

S1. Prognose - Physical Hardware Reservation Portal

- **Purpose**: User-facing portal to discover availability and book physical resources (bare-metal servers, storage, network ports) across multiple sites, with site management and institutional autonomy.
- **Capabilities**: Multi-site inventory and calendars; time-bounded reservations with conflict detection and predictable handover; lifecycle automation via Metal3/OpenStack Ironic (Redfish-based provisioning, image deploy, configuration, secure wipe); network/storage isolation for clean tenant transitions.
- **Site governance and RBAC**: The portal organizes users by sites; each site is an autonomous entity that shares its hardware with its own members. Resources can be booked only by users subscribed to that site. Sites retain full control over users and assets via role-based access control (User, Site Admin, Global Admin), with auditable actions and metering/reporting for accountability.
- **Outcome**: The *Prognose* portal (https://prognose.crownlabs.polito.it/) turns institutional hardware into an elastic, schedulable pool for research and education.

S2. GPU Sharing System on Kubernetes with KubeRay

- **Purpose**: Provide on-demand, isolated Ray clusters for GPU intensive jobs using RESTART-financed GPUs; independent from the Prognose portal.
- **Architecture**: Kubernetes as the substrate; KubeRay operator managing `RayCluster` resources and GPU device plugins.
- **Autoscaling**: Horizontal scale-out/in of Ray worker groups driven by job pressure (e.g., queue length/pending tasks), expanding with load and contracting when idle.

- **Data requirement**: A shared NFS filesystem is required for users to upload/download job data; NFS is mounted into Ray clusters to provide a common workspace and simplify data staging and result collection.
- **Workflow**: Users submit jobs to a per-reservation Ray cluster; at completion, workloads are drained and resources reclaimed to ensure a clean environment for subsequent users.

These two systems address complementary needs—bare-metal booking and elastic GPU consumption—while operating independently within the RESTART ecosystem.

Chapter 2

Problem Statement

2.1 Current Limitations in Bare-Metal Access

Despite the maturity of virtualized and containerized platforms, access to on-premise bare-metal resources across institutions remains fragmented and manual. Typical workflows rely on tickets and ad-hoc coordination, with limited visibility into availability and lead times. Clean handovers are not guaranteed: secure wipe, firmware resets, and network/storage reconfiguration are often inconsistent or undocumented, increasing risk and time-to-ready. Identity is siloed per site, making cross-site participation cumbersome; authorization models are coarse, blurring responsibilities between users and operators.

These gaps manifest as:

- Lack of self-service discovery and booking (no global view of calendars, capabilities, or constraints).
- Weak isolation between tenants (incomplete re-imaging, lingering credentials, shared networks).
- Inconsistent operational playbooks (PXE/image pipelines, BIOS/RAID configs, switch port provisioning).
- Limited, site-specific RBAC without auditable actions across roles (User, Site Admin, Global Admin).
- Minimal integration hooks to automate provisioning at reservation start/end.

Consequently, utilization is lower than potential, operational risk is higher, and user experience is uneven across sites.

2.2 Resource Optimization

The challenge is not only to grant access but to use the capacity effectively. Optimization spans time (reducing idle gaps between reservations), placement (matching requests to the smallest feasible set of resources), and policy (ensuring fairness and respecting site autonomy). Without these, capacity fragments, setup times inflate, and high-value assets sit underutilized.

Accelerator workloads intensify the problem: demand is bursty, jobs vary widely in size and duration, and static partitions or long-lived clusters strand GPUs when load subsides. A Kubernetes-based approach with KubeRay provisions on-demand Ray clusters that scale horizontally with job pressure, reclaiming workers when idle. To streamline data handling and shorten turnaround, a shared NFS workspace is required so users can upload inputs and retrieve results without bespoke staging.

Concretely, the system must provide:

- **Time efficiency**: A reservation scheduler with conflict detection and predictable handover windows to minimize temporal fragmentation, plus awareness of maintenance windows to avoid churn.
- **Capability-aware placement**: Packing by required features (e.g., GPU model/count, NIC speed, storage) and site constraints to increase utilization while meeting performance needs.
- **Policy and fairness**: The scope and approvals per site align consumption with institutional rules.
- **Feedback and metering**: Telemetry and usage accounting to drive capacity planning, right-sizing, and continuous tuning of scheduling policies.
- **GPU efficiency**: KubeRay-managed `RayClusters` with job-driven autoscaling, GPU-aware bin-packing, clean teardown, and an NFS-backed shared workspace for data ingress/egress.

Chapter 3

State of the Art

3.1 Overview

The management of shared computational resources across distributed organizations presents multifaceted challenges spanning technical, operational, and governance domains. This chapter examines the current landscape of resource reservation systems, bare-metal provisioning technologies, and distributed computing platforms, identifying their strengths and limitations in the context of multi-institutional resource sharing.

3.2 Resource Management and Provisioning Technologies

Historically, access to physical computational resources in academic environments has relied on manual coordination through email requests, ticketing systems, and phone-based scheduling. These fragmented approaches lack centralized visibility across institutions, making it impossible for users to understand availability or for administrators to optimize utilization. The manual nature creates bottlenecks requiring human intervention for every scheduling decision, while the absence of standardized audit trails complicates accountability and resource tracking. Each site develops its own procedures, creating inconsistencies that hinder cross-institutional collaboration.

Commercial cloud providers like AWS, Azure, and Google Cloud Platform offer sophisticated resource management through programmatic APIs and elastic scaling capabilities. These platforms excel at virtualized resource

allocation with comprehensive monitoring and multi-tenancy isolation. However, their focus on abstracted virtualized environments creates limitations for bare-metal management in academic settings, where research often requires direct hardware access, custom imaging, and predictable handover periods. Additionally, these platforms lack the governance models necessary for institutional autonomy and struggle with integration challenges involving academic identity providers and site-specific policies.

HPC schedulers like SLURM excel at managing computational jobs across clusters using queue-based algorithms for fair resource distribution and complex constraint handling. However, these systems face limitations when extended beyond batch job management. Their queue-based architecture assumes job submission rather than interactive bare-metal access, creating challenges for calendar-based reservations where researchers need to book specific time slots. Additionally, HPC schedulers struggle with multi-site federation, having been designed for single administrative domains rather than cross-institutional collaboration.

Modern bare-metal provisioning relies on standardized protocols for automated server lifecycle management. Out-of-band management protocols like Redfish and IPMI provide dedicated network channels for hardware control, enabling remote power management, hardware inventory collection, firmware updates, and virtual media mounting. Network-based boot mechanisms, particularly PXE, complement these capabilities by leveraging DHCP and TFTP for automated OS deployment, creating standardized pathways for consistent, reproducible server configurations.

OpenStack Ironic provides mature bare-metal provisioning through a comprehensive hardware abstraction layer with driver plugins for diverse server hardware. It implements standardized image management with checksum verification and integrates with OpenStack Neutron for network configuration. While Ironic supports multiple deployment methods (PXE, virtual media, custom ramdisk), its tight coupling with the OpenStack ecosystem introduces complexity for organizations seeking lightweight solutions.

Metal3 addresses these concerns by bringing bare-metal provisioning to Kubernetes through cloud-native patterns. Using Kubernetes custom resource definitions and operators, Metal3 provides declarative management interfaces while maintaining Ironic as its backend. The BareMetalHost CRD allows administrators to describe hardware state in standard Kubernetes manifests, enabling Infrastructure as Code practices and integration with modern CI/CD workflows.

3.3 Distributed Computing and GPU Sharing Platforms

Kubernetes has established itself as the dominant container orchestration platform, enabling declarative resource management through YAML manifests and automatic workload scheduling. Its extensibility through Custom Resource Definitions and the operator pattern makes it particularly suitable for specialized research environments. GPU resource management in Kubernetes leverages specialized device plugins like the NVIDIA GPU Operator for automated driver management. Modern features include Multi-Instance GPU (MIG) support for fine-grained partitioning, GPU topology awareness for optimal placement, and resource quotas for fair allocation across users and projects.

Ray has emerged as a leading framework for machine learning and AI workloads, providing a unified API for distributed training, hyperparameter tuning, and model serving. It excels at handling dynamic task graphs and implements sophisticated fault tolerance with automatic recovery and dynamic cluster scaling based on workload demands.

KubeRay integrates Ray with Kubernetes through custom resource definitions, enabling declarative cluster management with GitOps workflows. The RayJob CRD provides batch job execution with automatic cleanup, while RayService CRD supports model serving with integrated RBAC and resource management for shared research environments. Importantly, KubeRay enables direct interaction with RayCluster resources through the Python SDK, allowing users to programmatically submit jobs to worker pods within the cluster, an approach adopted in this work for seamless integration with the reservation system.

3.4 Current Limitations and Alternative Solutions

Despite mature individual components, significant gaps remain for comprehensive multi-institutional resource sharing. The fundamental challenge lies in fragmented tools addressing specific domains without comprehensive integration frameworks.

Current systems suffer from limited cross-domain automation, requiring manual coordination between provisioning, network configuration, and access

management. Inconsistent APIs and lack of standardized event schemas complicate integration efforts. The cloud resource reservation system developed for RESTART addresses these challenges through event-driven architecture coordinating between the main backend, specialized webhooks, and external systems like Metal3.

Existing platforms rarely model complex multi-institutional governance requirements, particularly site-scoped autonomy with hierarchical RBAC systems. Most provisioning tools focus on immediate allocation rather than calendar-based reservations, creating gaps in conflict detection and predictable handover procedures. Additionally, specialized hardware management, network equipment integration, and shared storage coordination remain largely manual processes requiring institution-specific solutions.

TensorHive[1] represents a specialized GPU resource management platform for academic environments, providing web-based monitoring and SSH-based job execution. However, evaluation revealed fundamental limitations for multi-institutional requirements: single-node focus rather than distributed cluster management, legacy technology stack lacking modern container support, SSH-centric architecture incompatible with API-driven automation, limited reservation capabilities without calendar-based booking, and monolithic design preventing extension with institution-specific policies. The absence of multi-tenancy support and site-scoped governance made it unsuitable for federated research environments, leading to the decision to develop a comprehensive greenfield solution.

The cloud resource reservation system demonstrates how webhook-based architectures address integration challenges through event-driven patterns. The reservation event processor consumes events from the main system and dispatches them to specialized webhooks handling domain-specific tasks. This enables loose coupling while maintaining reliable workflow coordination, as demonstrated by the server provisioning webhook integrating with Kubernetes and Metal3, and the switch port webhook handling network configuration through SSH automation.

3.5 Synthesis and Research Motivation

The analysis reveals mature individual components lacking comprehensive integration for multi-institutional resource sharing. While tools like Metal3,

[1] https://github.com/roscisz/TensorHive

Ironic, and Kubernetes provide sophisticated capabilities in their domains, the challenge lies in creating unified systems that bridge specialized tools to address academic research requirements.

Key limitations converge around several themes: integration of diverse resource types under unified management, site-scoped multi-tenancy respecting institutional autonomy, calendar-based reservation systems with predictable handover, event-driven automation spanning complete resource lifecycles, and modern web interfaces integrated with federated identity systems.

These requirements motivated developing the comprehensive cloud resource reservation system presented in this thesis. Rather than replacing existing technologies, the approach combines mature components like Keycloak, PostgreSQL, and Metal3 through event-driven architectures and webhook-based integration. This strategy enables institutions to leverage existing investments while providing unified management capabilities for effective cross-institutional resource sharing, as demonstrated by the RESTART partnership implementation. Comprehensive documentation of the platform architecture, deployment procedures, and usage guidelines is available at https://docs.prognose.crownlabs.polito.it/.

Chapter 4

Technologies and Tools

The development of a comprehensive cloud resource reservation platform requires careful technology selection to address the unique challenges of multi-institutional academic environments. Rather than simply adopting popular frameworks, the technology choices for both the Prognose platform and GPU sharing system were driven by specific requirements: the need for robust multi-tenancy, seamless integration with existing institutional infrastructure, and the ability to scale across diverse research environments while maintaining security and reliability.

This chapter explores how modern web technologies, when thoughtfully combined, create a cohesive ecosystem that bridges the gap between cutting-edge research needs and operational stability. The architecture demonstrates that sophisticated academic platforms can be built using production-ready technologies that researchers and administrators can confidently deploy and maintain.

4.1 Backend Architecture

The backend infrastructure represents a departure from monolithic academic systems, embracing microservices principles while maintaining the enterprise-grade reliability required for multi-institutional deployments. The core backend services are built on Spring Boot and Java 23, leveraging the mature ecosystem and proven scalability that academic institutions require for mission-critical systems.

Spring Boot emerged as the cornerstone framework for both the main

reservation backend and the event processor service, providing production-ready features including comprehensive metrics, health checks, and configuration management out of the box. The framework's auto-configuration capabilities simplify deployment across diverse institutional environments, while its extensive integration ecosystem enables seamless connectivity with PostgreSQL, Keycloak, and external services. Java 23's modern language features, including virtual threads and pattern matching, enhance performance and code maintainability for complex reservation scheduling algorithms.

The webhook architecture deserves particular attention as it addresses a fundamental challenge in academic IT: how to integrate with diverse institutional systems without creating tight coupling. While the core platform uses Java and Spring Boot, webhook services can be implemented in any language since communication occurs through standard REST APIs. The server provisioning webhook, implemented in Python with FastAPI, demonstrates this flexibility by interfacing with both Kubernetes APIs and Metal3 controllers, while secure communication between the webhook services and the event processor/backend applications is ensured through HMAC-SHA256 signatures. Similarly, the switch port webhook manages network configuration through SSH automation, utilizing the Netmiko library to abstract vendor-specific CLI interactions. This language-agnostic webhook design allows institutions to implement custom provisioning logic in their preferred technology stack without modifying core system components.

PostgreSQL's selection extends beyond its reputation for reliability to its comprehensive feature set that supports the complex requirements of reservation systems. PostgreSQL's robust transaction handling and ACID compliance ensure data integrity during concurrent reservation operations, while its flexible data modeling capabilities accommodate the hierarchical resource structures and complex scheduling constraints inherent in multi-institutional environments.

4.2 Frontend Technologies

The frontend development philosophy prioritized user experience over technical novelty, recognizing that academic users often access the system in high-pressure research scenarios where intuitive interfaces are crucial. React 19's concurrent rendering capabilities prove particularly valuable for the booking calendar, where multiple users might simultaneously view and modify reservations, ensuring smooth user interactions without interface blocking.

Material-UI 6 provides more than visual consistency; its accessibility features ensure compliance with academic institution requirements for inclusive design. The component library's extensive customization options accommodate the diverse branding needs of consortium members while maintaining functional consistency. The implementation includes sophisticated responsive design patterns that adapt to both desktop administration workflows and mobile field access scenarios common in experimental research.

The booking calendar implementation using React Big Calendar showcases how modern web technologies can rival desktop applications in functionality. The calendar provides intuitive drag-and-drop reservation scheduling, allowing users to create bookings by selecting time slots directly on the calendar interface. Multiple view modes (month, week, day) accommodate different planning perspectives, while updates are reflected upon page refresh to maintain data consistency. Integration with the backend's temporal logic ensures that calendar views display accurate availability information, while interactive event manipulation provides the immediate feedback researchers expect from modern interfaces.

Internationalization through react-i18next addresses more than language preferences; it enables cultural adaptation of date formats, timezone handling, and workflow patterns that vary between academic institutions. The system's flexibility in this regard has proven essential for consortium deployments where member institutions have different administrative cultures and user expectations.

4.3 Security and Identity Management

Rather than integrating with multiple universities' identity management systems, the project leverages the existing Keycloak instance maintained by Politecnico di Torino for its internal operations. A dedicated realm has been created within this Keycloak deployment to contain all system users, providing a centralized identity management solution that simplifies authentication while maintaining the security standards required for academic environments. This approach eliminates the complexity of federated identity integration while ensuring reliable access control for consortium members.

The implementation of hierarchical role-based access control reflects the reality of academic governance structures. The three-tier system (User, Site Admin, Global Admin) maps directly to academic hierarchies while providing the flexibility required for cross-institutional collaboration. Site-scoped data

isolation ensures that sensitive research information remains within institutional boundaries, while selective sharing mechanisms enable collaborative projects.

JWT token handling includes custom claims for site membership and resource access rights, enabling fine-grained authorization decisions at both the application and infrastructure levels.

The security model addresses the reality that academic networks often have different security postures and compliance requirements. The implementation includes configurable security policies that allow institutions to enforce their specific requirements while maintaining interoperability with consortium partners.

4.4 Cloud-Native Infrastructure and Orchestration

The transition to Kubernetes-based deployment reflects a strategic decision to align with modern academic computing trends while providing the operational flexibility that research environments demand. Unlike traditional academic systems that require specialized operational knowledge, the containerized approach enables standard IT practices that most institutions can support.

Docker containerization serves multiple purposes beyond deployment consistency. The multi-stage build process creates optimized images that minimize attack surfaces while ensuring reproducible deployments across diverse institutional environments. Container images are tagged with database-specific variants (postgres, oracle) that allow institutions to integrate with their existing database infrastructure without requiring system-wide changes.

The KubeRay operator provides sophisticated GPU cluster management through cloud-native patterns, operating independently from the reservation system. KubeRay handles jobs submitted by users and schedules them as soon as resources become available on the cluster, without integration with calendar-based reservations. This approach leverages KubeRay's mature resource management and autoscaling capabilities while providing the flexible GPU access that research workflows require.

Metal3 integration showcases the system's ability to bridge cloud-native practices with traditional academic infrastructure. The BareMetalHost custom resources enable Infrastructure as Code practices for physical server management, allowing research groups to provision bare-metal resources through

the same interfaces they use for cloud resources.

4.5 Continuous Integration and GitOps Deployment

The deployment process begins when developers commit changes to the repository and follows a GitOps workflow that ensures reproducibility and auditability essential for academic environments. The system maintains stable code in the main branch, from which deployments are initiated through controlled manual processes.

When a feature is successfully merged into the main branch, users can manually trigger a deployment pipeline that executes three critical steps:

1. **Build and Tag**: The pipeline builds Docker images and tags them with version information extracted from the project's build configuration (pom.xml for Java components or package.json for Node.js services).
2. **Push to Registry**: The tagged images are pushed to DockerHub, making them available for deployment across different environments.
3. **Helm Chart Update**: The pipeline automatically updates the Helm chart repository with the new service version, modifying the values.yaml file to reference the newly built image tags.

ArgoCD continuously monitors the Helm chart repository, specifically watching for changes to the values.yaml file. When the pipeline completes and updates the repository, ArgoCD automatically detects these changes and reconciles the deployment by re-applying the Helm chart and updating the corresponding Kubernetes custom resources to match the new configuration.

This GitOps approach provides several advantages for academic environments: complete audit trails of all deployments, the ability to roll back to previous versions through Git history, and separation of concerns between application development and deployment operations. The manual trigger for production deployments ensures that releases occur at appropriate times while maintaining the automation benefits of GitOps.

Helm charts provide the parameterization capabilities that enable single-source deployments across diverse institutional environments. The chart structure accommodates everything from single-node development deployments to multi-cluster production configurations, while maintaining security and operational best practices.

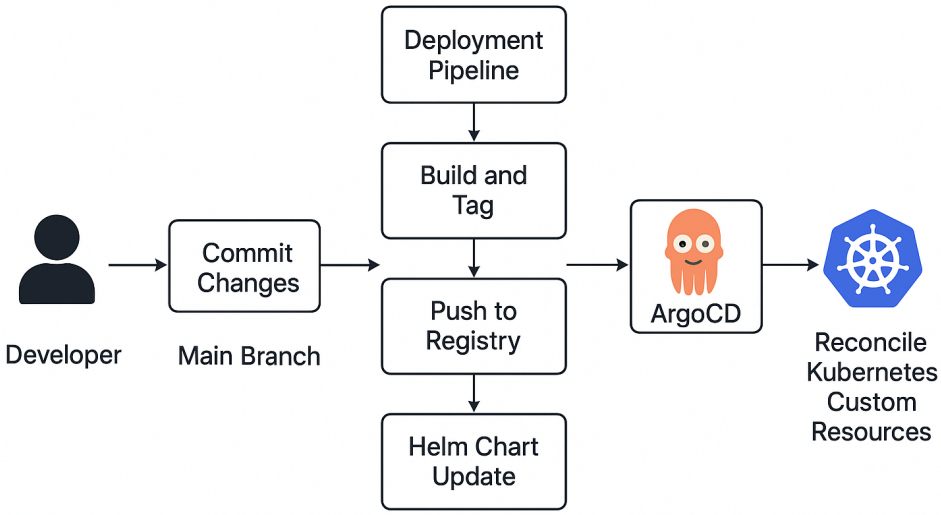

Figure 4.1. CI/CD workflow illustrating the deployment process

The complete technology stack demonstrates that academic platforms can embrace modern development practices without sacrificing the reliability and institutional autonomy that research environments demand. By carefully selecting technologies that align with both current capabilities and future scalability requirements, the platform provides a foundation for sustainable multi-institutional collaboration in the evolving landscape of academic computing.

Chapter 5

System Architecture and Design

This chapter presents the technical implementation of the cloud resource reservation system, detailing the architectural decisions, design patterns, and core components that enable multi-institutional resource sharing. The system architecture demonstrates how modern multiservice patterns, event-driven communication, and multi-tenant design can be effectively combined to create a scalable platform for academic resource management.

5.1 Overall Architecture

The reservation system implements a distributed architecture designed to address the specific challenges of multi-institutional resource management in academic environments. The architectural approach prioritizes institutional autonomy while enabling seamless cross-site collaboration through well-defined interfaces and standardized communication protocols.

5.1.1 Multi-service Architecture

The system implements a multi-service architecture comprising two main services that work together to provide comprehensive resource management capabilities. This design choice balances architectural clarity with implementation simplicity, addressing the operational realities of academic environments where systems must be maintainable by different people over time while keeping complexity manageable.

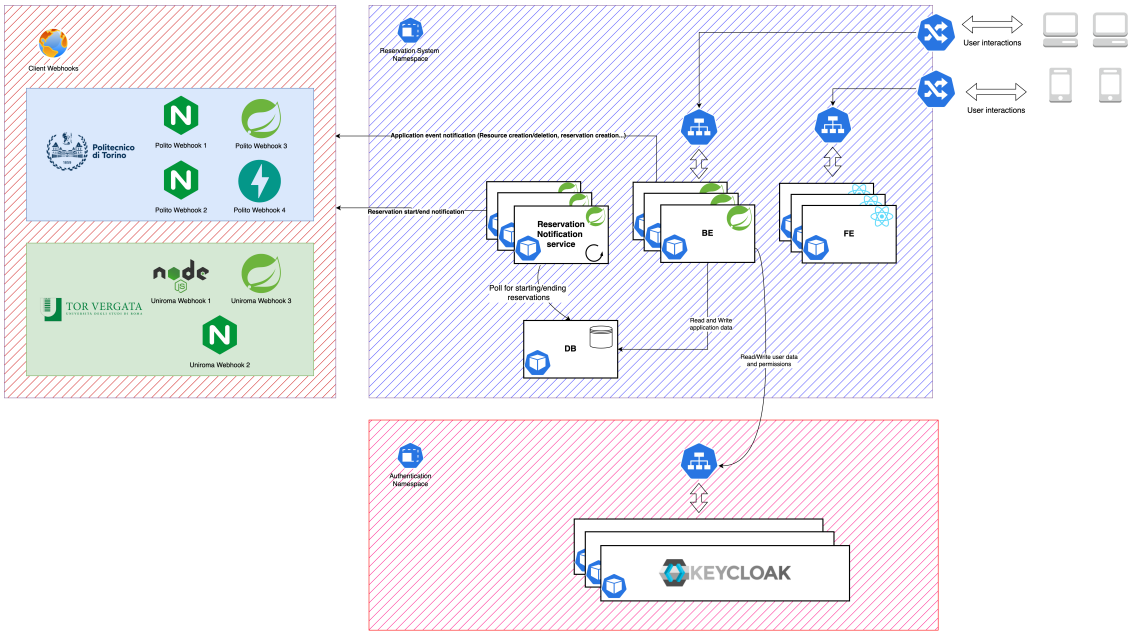

Figure 5.1. High-level System Architecture Overview showing the main components of the cloud resource reservation system and their interactions.

The primary service is the reservation backend implemented in Spring Boot with Java 23, which consolidates all REST API functionality including resource management, booking operations, user management, and administrative functions. While this monolithic backend could theoretically be decomposed into independent microservices for different functional domains, the decision was made to maintain a unified service to reduce operational complexity and facilitate maintenance in academic environments where staff turnover is common.

The second service is the Event Processor, which operates independently to handle asynchronous operations. This service monitors database events for new reservation starts and other lifecycle changes, then notifies registered webhooks accordingly. This separation ensures that user-facing operations in the main backend remain responsive while background processing occurs independently, preventing webhook delivery delays from affecting the user experience.

The two services communicate through the shared PostgreSQL database using scheduled polling, where the Event Processor periodically queries for reservation events that need processing. This approach creates loose coupling while maintaining data consistency through database transactions. Both services are containerized using Docker and can be deployed independently, allowing institutions to scale the event processing separately from the main application based on webhook notification volume.

The API design follows RESTful principles with comprehensive OpenAPI documentation, facilitating integration with institutional systems and third-party tools. Authentication and authorization are externalized to Keycloak, ensuring consistent security policies across all services while allowing institutional customization of authentication mechanisms. All API requests to the main backend are authenticated using JWT (JSON Web Token) tokens issued by Keycloak, which contain user identity, role assignments, and site membership information. The Event Processor service, being an internal component that processes database events and sends webhooks, does not validate JWT tokens since it is never directly contacted by users and instead uses Keycloak's Admin Client with service credentials to fetch user information when needed.

5.1.2 Multi-tenant Design

The system implements a sophisticated multi-tenant architecture that respects institutional boundaries while enabling resource sharing. The design

uses a site-based tenancy model where each participating institution represents a distinct tenant with its own resource inventory, user base, and administrative hierarchy.

Site-scoped resource management ensures that institutions maintain full control over their resources while participating in the broader resource sharing ecosystem. Each site has dedicated administrators who can configure resource types, define booking policies, and manage local user access. Resources are tagged with site identifiers, ensuring that queries and operations respect institutional boundaries.

The multi-tenant implementation extends to the user interface, where site selection controls the visible resource inventory and available administrative functions. Users can be associated with multiple sites, reflecting the reality of collaborative research projects that span institutional boundaries. Role-based access control (RBAC) operates within site contexts, ensuring that administrative permissions remain scoped to appropriate institutional boundaries.

Database design supports multi-tenancy through site-scoped foreign keys and query filters that automatically limit data access to appropriate tenants. This approach ensures data isolation while maintaining query performance and simplifying application logic.

5.1.3 Event-driven Architecture

The system employs event-driven patterns to achieve loose coupling between components and enable responsiveness to booking lifecycle events. The Event Processor uses scheduled polling to monitor database changes, providing reliable event processing with minimal infrastructure complexity.

Event processing follows a publish-subscribe pattern where booking operations generate events that are consumed by multiple subscribers. The primary event types include booking creation, modification, deletion, and status changes. Each event carries comprehensive payload information, including resource details, user information, and timing data.

The event processor service monitors database events and triggers appropriate responses, including webhook notifications to external systems, email notifications to users, and audit log entries. This asynchronous processing ensures that user operations complete quickly while background systems receive timely updates about state changes.

Webhook integration enables external systems to receive real-time notifications about booking events through a flexible and configurable system. Since

different resource types may require specialized provisioning logic, the system provides an administrative webhook management interface where administrators can configure custom endpoints for specific event types and resource combinations. Each webhook configuration specifies the target URL, the triggering event types (reservation start, end, modification), and the specific resources or resource types that should trigger notifications.

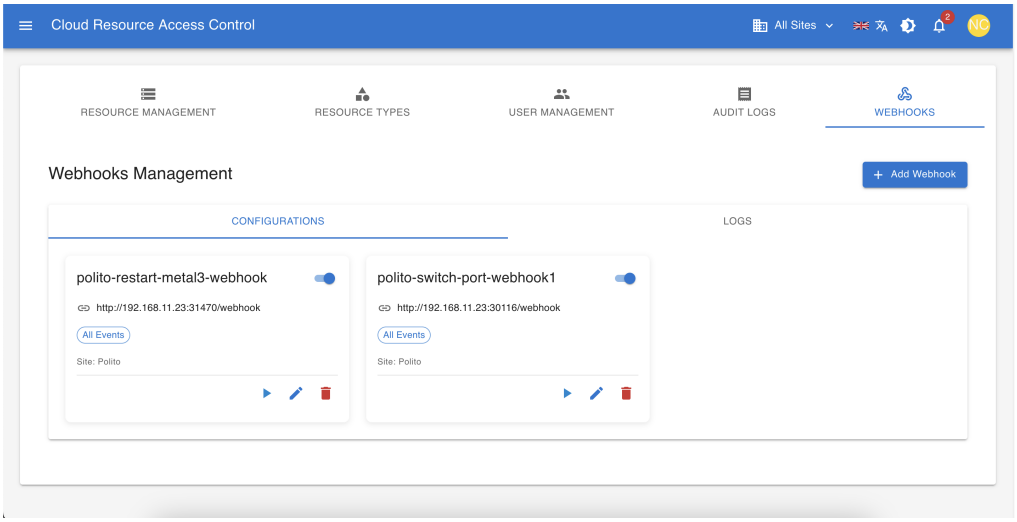

Figure 5.2. Administrative Webhook Management Interface

The webhook system implements bidirectional HMAC-SHA256 signature verification for secure communication. When the Event Processor sends notifications to external webhook endpoints, it includes an HMAC signature generated using a shared secret configured through the administrative interface. External webhook systems must validate this signature to ensure the authenticity of incoming requests and prevent unauthorized provisioning triggers from malicious sources.

Conversely, when webhooks need to send status updates back to the main system (such as provisioning started, completed, or failed notifications), they must include their own HMAC signature using the same shared secret. The main backend validates these signatures before accepting webhook responses, ensuring that only authorized external systems can update reservation statuses and notify users about provisioning outcomes. All webhook requests include comprehensive payload information about the reservation, user details, and resource specifications, enabling external systems to implement custom provisioning workflows. The system includes sophisticated retry logic with

exponential backoff for reliable delivery, maintaining communication integrity even when external systems experience temporary unavailability.

5.1.4 Hierarchical Resource Model

The system implements a sophisticated hierarchical resource model that enables complex parent-child relationships between resources, providing automatic conflict resolution and dependency management crucial for academic computing environments.

Resource Creation Flow

Resource creation follows a two-step process that ensures consistency and proper classification:

1. **Resource Type Creation**: Administrators first define resource types using the administrative API endpoints. Each resource type specifies:
   - Basic properties (name, color for UI visualization, site association)
   - Custom parameter definitions stored as JSON in the `customParameters` field
   - Site-scoped access control ensuring institutional autonomy
2. **Resource Instance Creation**: Individual resources are then created as instances of these types through dedicated API endpoints. Resources inherit the type's custom parameters but maintain instance-specific properties including hierarchical relationships through `parentId` and `subResources` fields.

The backend maintains referential integrity through JPA relationships: resources reference their parent via `@ManyToOne` mapping and maintain bidirectional relationships with children through `@OneToMany` collections.

Parent-Child Resource Relationships

The hierarchical model addresses common academic scenarios where booking one resource should automatically affect related resources. The implementation uses database-level parent-child relationships:

- **Parent Resources**: When reserved, automatically block all child resources for the same time period
- **Child Resources**: Cannot be independently reserved if their parent is already booked
- **Automatic Blocking**: The system prevents conflicts by checking hierarchical dependencies during booking validation

For example, booking a complete server rack (parent) automatically makes individual servers within that rack (children) unavailable, preventing double-booking scenarios while ensuring resource isolation.

The frontend provides an intuitive drag-and-drop interface that allows administrators to establish parent-child relationships by dragging resources onto potential parents. The system includes circular reference detection to prevent invalid hierarchies and provides visual indicators for hierarchical relationships in calendar views.

Custom Parameter System

The resource type system includes a flexible custom parameter mechanism that enables institution-specific data collection during booking:

- **Parameter Definition**: Resource types store custom parameter schemas as JSON in the `customParameters` field, defining field labels and whether they are required
- **Booking Integration**: The booking interface dynamically renders these custom fields based on the selected resource's type
- **Validation**: Both frontend and backend enforce required parameter validation, ensuring complete data collection
- **Storage**: Parameter values are serialized as JSON and stored in the booking's `customParameters` field

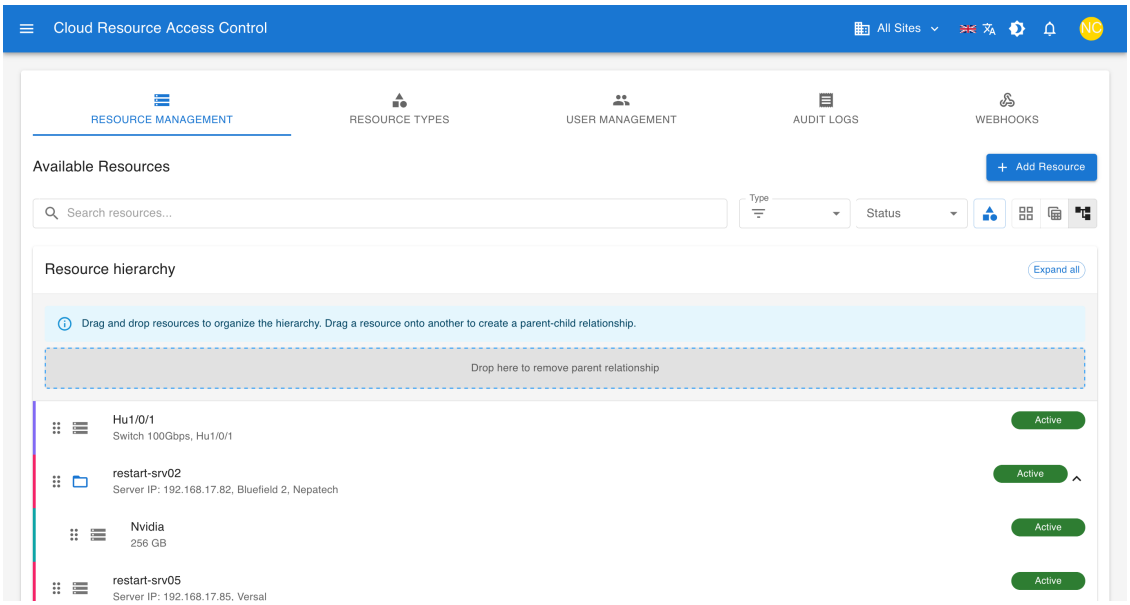

Figure 5.3. Resource Hierarchy Management Interface

This approach allows institutions to collect resource-specific information such as software requirements, access credentials, or project codes without modifying core system schemas. The administrative interface provides comprehensive tools for managing these custom parameters with real-time validation.

Resource availability calculations consider hierarchical relationships at multiple levels, enabling efficient conflict detection and capacity planning. The booking engine evaluates constraints across the entire resource hierarchy, ensuring that reservations respect parent-child dependencies, institution-specific policies, and custom parameter requirements. This comprehensive approach provides the flexibility needed for diverse academic computing environments while maintaining system consistency and preventing resource conflicts.

Resource Status Management

The system implements a comprehensive resource status management mechanism that provides administrators with granular control over resource availability and enables proper maintenance workflows. Each resource maintains a status field that determines its booking eligibility and operational state.

The system defines three primary resource statuses through the ResourceStatus enumeration:

- **ACTIVE**: Resource is fully operational and available for booking. Only resources in ACTIVE status can receive new reservations
- **MAINTENANCE**: Resource is temporarily unavailable due to scheduled maintenance, repairs, or updates. Existing bookings remain valid, but no new reservations can be created
- **UNAVAILABLE**: Resource is permanently or indefinitely unavailable for booking due to decommissioning, major failures, or policy restrictions

The backend provides dedicated endpoints for status management, including a PATCH /resources/{id}/status operation that allows administrators to update resource status independently from other resource properties. This design enables rapid status changes during operational incidents without requiring full resource updates.

Status changes trigger comprehensive notification workflows managed by the resource management service. When a resource transitions from ACTIVE to MAINTENANCE or UNAVAILABLE, the system automatically identifies all users with future bookings for that resource and sends warning notifications about potential service disruptions. Conversely, when resources return to ACTIVE status, users with upcoming reservations receive success notifications confirming resource availability.

The booking validation system enforces status-based constraints at multiple levels. The event management service includes mandatory status checks that prevent booking creation or modification for non-ACTIVE resources. These validations occur alongside conflict detection and permission verification, ensuring comprehensive booking integrity.

The frontend provides intuitive status management through multiple interfaces. The resource management forms include dedicated status controls allowing administrators to set resource status during creation or updates. The resource listing interfaces display status information using color-coded chips (green for ACTIVE, orange for MAINTENANCE, red for UNAVAILABLE) with internationalized labels. Administrative filtering capabilities enable operators to view resources by status, facilitating maintenance planning and capacity management.

Status management integrates seamlessly with the hierarchical resource model, where parent resource status changes can affect child resource availability through inherited constraints. The system maintains audit trails for

all status changes, providing operators with complete visibility into resource lifecycle management and enabling compliance with institutional operational procedures.

5.2 Core Components

The system architecture comprises several core components that work together to provide comprehensive resource management capabilities. Each component addresses specific functional requirements while maintaining clear interfaces with other system elements.

5.2.1 Resource Management Service

The Resource Management Service forms the foundation of the system's inventory management capabilities. Implemented as part of the Spring Boot backend, this service maintains the authoritative record of all available resources across participating sites.

The service provides CRUD operations for resource definitions, supporting dynamic addition and modification of resource inventory. Resource specifications include technical characteristics (CPU, memory, storage, network capabilities), availability constraints (maintenance windows, usage policies), and integration parameters (provisioning endpoints, monitoring systems).

Site administrators can configure resource types and instances through the administrative interface, with changes immediately reflected in booking availability calculations. The service validates resource configurations to ensure compatibility with booking constraints and provisioning systems.

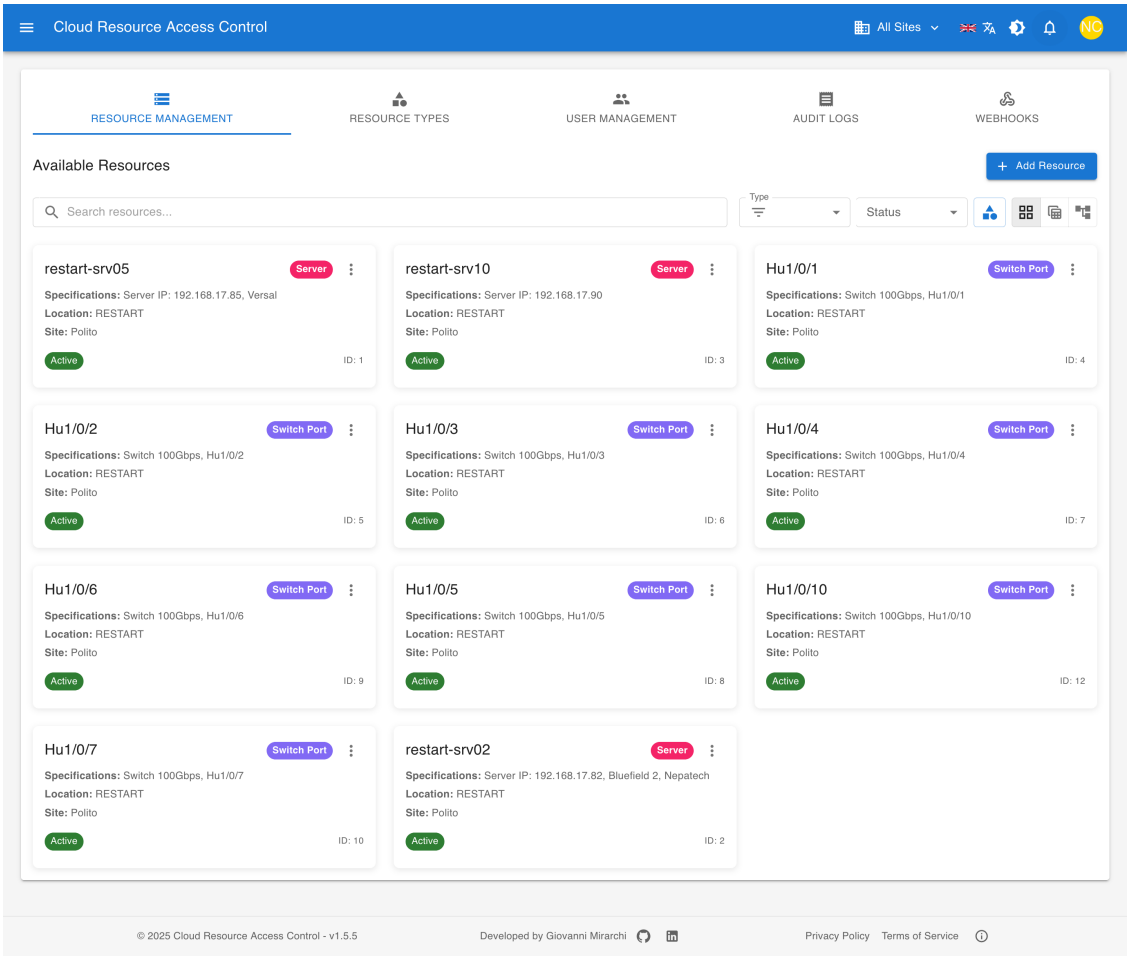

Figure 5.4. Resource Management Administrative Interface

5.2.2 Booking Engine

The Booking Engine implements the core reservation logic, managing the complete lifecycle of resource bookings from initial request through final cleanup. The engine handles conflict detection, capacity validation, and booking state management through an algorithm that ensure resource availability and prevent double-booking scenarios.

Conflict detection operates at multiple levels, checking for overlapping time periods, resource capacity constraints, and site-specific policies.

The booking process includes comprehensive validation steps that verify user permissions, resource availability, and booking policy compliance. Approved bookings generate events that trigger provisioning workflows and user

notifications. The engine maintains detailed state information throughout the booking lifecycle, enabling precise tracking and audit capabilities.

Integration with external provisioning systems occurs through webhook notifications that carry complete booking context, including user credentials, resource specifications, and timing requirements. This design enables seamless integration with diverse provisioning technologies while maintaining booking system independence.

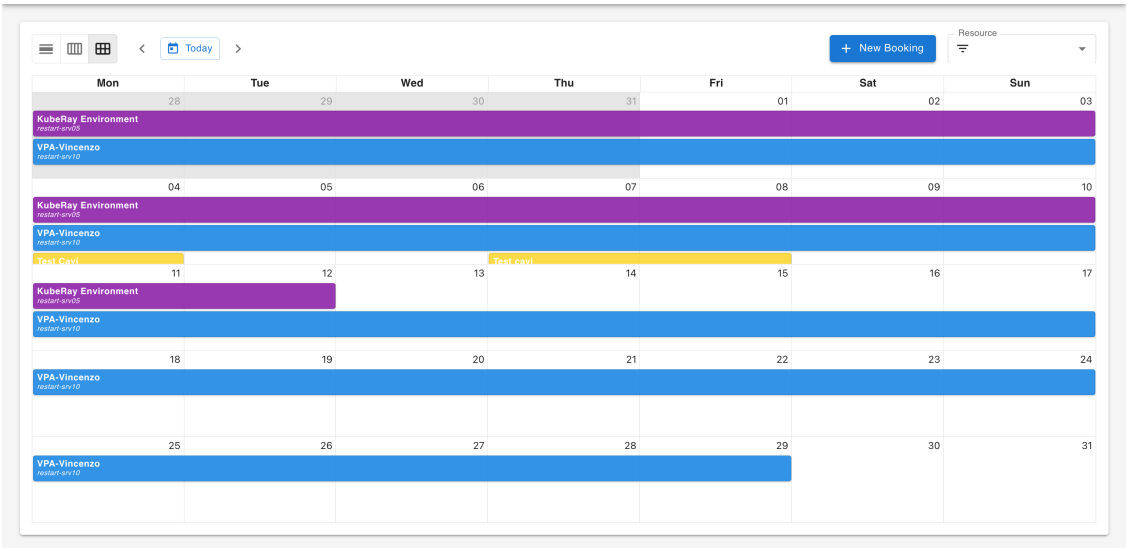

Figure 5.5. Interactive Booking Calendar

5.2.3 User Management

User Management leverages Keycloak's comprehensive identity and access management capabilities, providing federated authentication and fine-grained authorization controls. The integration possibly supports multiple authentication mechanisms, including institutional SSO systems, academic federations, and direct user accounts.

The system implements a three-tier role hierarchy: Users can create and manage their own bookings, Site Administrators can manage resources and users within their institutional scope, and Global Administrators can configure system-wide settings and cross-site policies. Role assignments are site-scoped, enabling users to have different permissions at different institutions.

User profile management includes basic user information such as name, surname, email address, and password management. SSH public key management enables secure access to provisioned resources without requiring separate credential distribution.

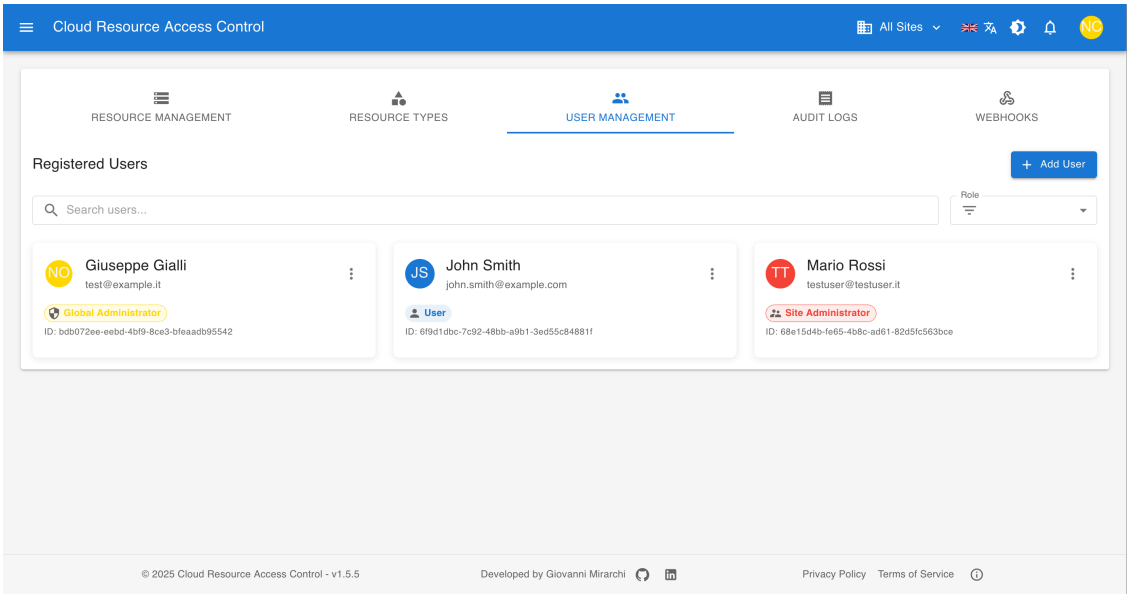

Figure 5.6. User Management Interface showing Role-based Access Control

The integration with Keycloak enables advanced features such as role mapping from institutional identity systems, group-based access control, and session management across multiple system components. This approach ensures consistent security policies while supporting the complex user relationships common in academic environments.

5.2.4 Notification System

The Notification System serves as the primary communication channel between users and the reservation system, managing in-application notifications that keep users informed about critical events affecting their reservations and resource availability.

The system focuses on two primary notification scenarios that are essential for effective resource management. First, when resources become unavailable due to status changes (maintenance, decommissioning, or other operational issues), the system automatically notifies all users with existing or upcoming reservations for those resources. This proactive communication ensures users can adjust their research schedules and seek alternative resources before critical deadlines.

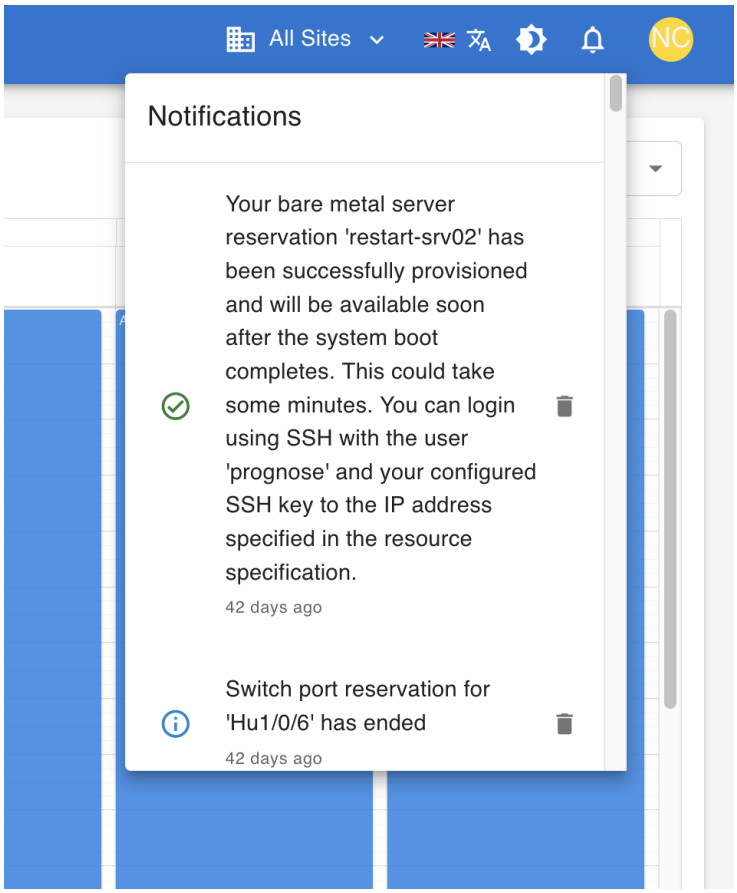

Figure 5.7. In-application Notification System

Second, the notification system serves as the communication bridge for

webhook-generated messages. When external webhook services complete provisioning operations, encounter errors, or need to communicate status updates to users, they send notifications back to the main system through authenticated API endpoints. The notification system then presents these messages to users through the web interface, providing real-time feedback about the status of their resource provisioning and access credentials.

5.2.5 Event Processor

The Event Processor Service operates as a dedicated service responsible for handling asynchronous operations and maintaining system responsiveness. The service uses scheduled polling to monitor the database for reservation events requiring processing, checking for events that are starting soon or have recently ended, and coordinates appropriate responses across system components.

Event processing includes webhook delivery to external systems, ensuring that provisioning systems receive timely notifications about booking changes. The processor implements sophisticated retry logic with exponential backoff, maintaining reliable communication even when external systems experience temporary unavailability.

Scheduled processing handles time-based operations such as booking reminders, resource cleanup notifications, and availability recalculation. The service coordinates with the booking engine to ensure that resource states remain consistent and that users receive appropriate notifications about upcoming booking events.

The processor maintains comprehensive logs of all event handling activities, providing audit trails and troubleshooting information. All system monitoring and operational visibility is achieved through the Audit Log view in the Administration panel, which provides administrators with complete visibility into event processing performance, system health, and all activities occurring within the reservation platform.

The Event Processor architecture demonstrates how multi-service patterns can effectively handle complex asynchronous workflows while maintaining system reliability and operational transparency. The design ensures that the system remains responsive to user interactions while effectively managing background operations and external system integration, providing a foundation for future evolution toward more granular microservice architectures as operational requirements mature.

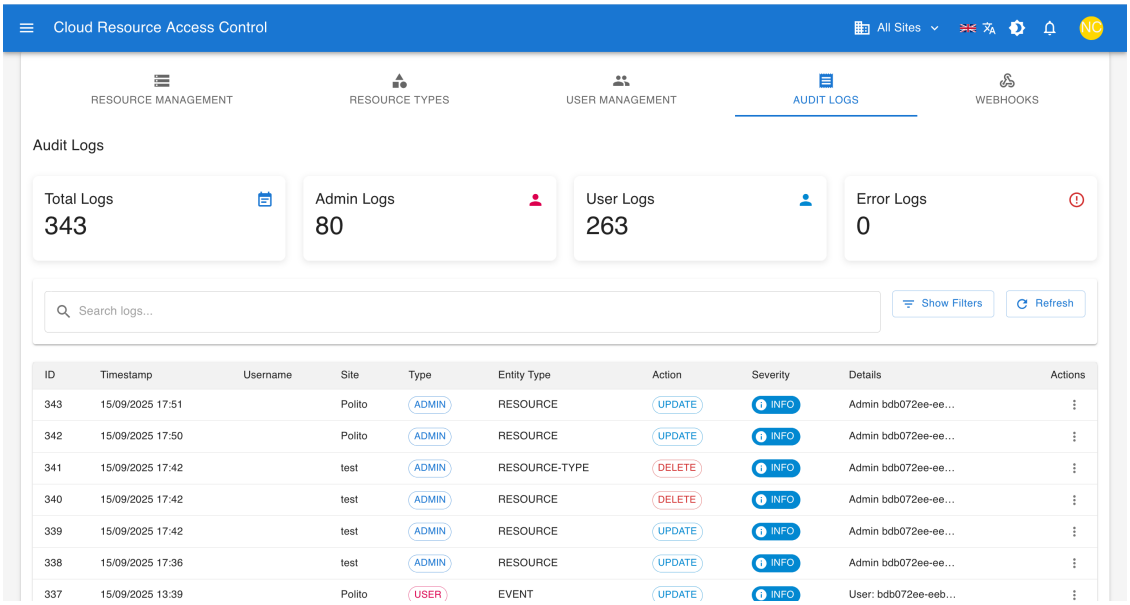

Figure 5.8. Monitoring Dashboard showing Audit Logs

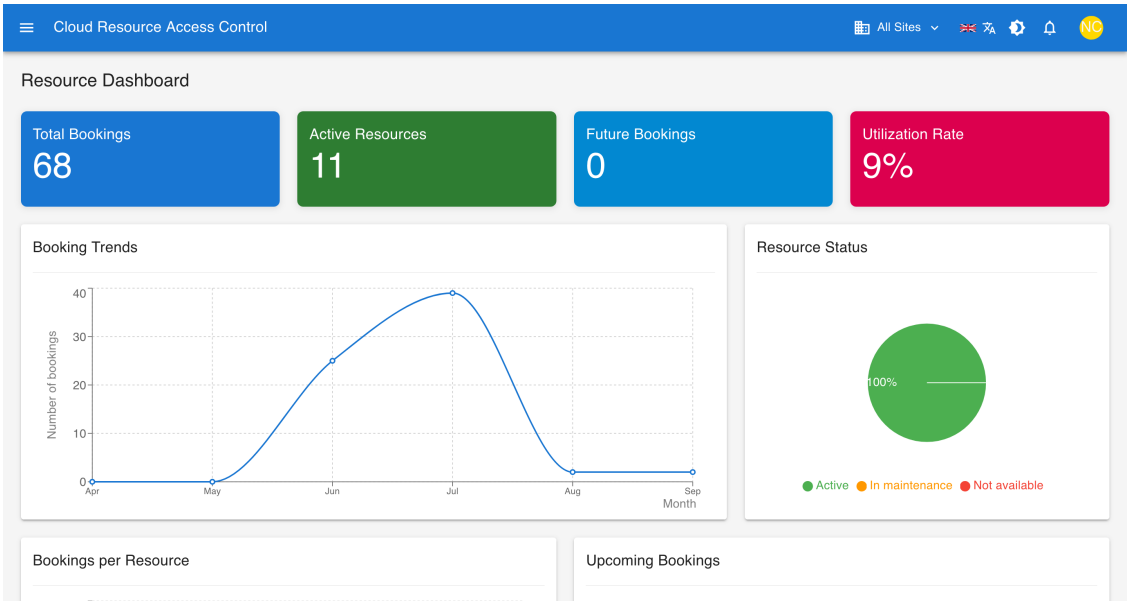

Figure 5.9. Administrative Dashboard showing System Statistics and Monitoring Infographics

Chapter 6

Distributed GPU Sharing with Ray and Kubernetes

This chapter presents the implementation of the GPU sharing system developed as part of the RESTART partnership, focusing on providing elastic, on-demand access to GPU resources for machine learning and AI workloads. Unlike the Prognose reservation portal, this system operates independently and leverages KubeRay to manage Ray clusters on Kubernetes, enabling researchers to submit distributed computing jobs without requiring deep knowledge of cluster management.

The GPU sharing system addresses the challenge of efficiently utilizing expensive GPU hardware across multiple research groups while providing isolation and fair access. The solution demonstrates how modern container orchestration and distributed computing frameworks can be combined to create a user-friendly platform for GPU-intensive research workflows.

6.1 GPU Sharing Architecture Design

6.1.1 System Overview

The GPU sharing architecture is built on three foundational components that work together to provide seamless GPU access: Kubernetes as the orchestration platform, KubeRay as the Ray cluster management operator, and a shared NFS filesystem for data persistence and job artifact management.

The design philosophy prioritizes simplicity for end users while leveraging sophisticated backend orchestration. Researchers interact with the system

through standard Python Ray APIs, submitting jobs programmatically without needing to understand the underlying Kubernetes infrastructure. This abstraction enables productive research workflows while maintaining the operational benefits of containerized deployment and resource isolation.

The architecture supports dynamic resource allocation where GPU resources are provisioned on-demand based on job requirements. When researchers submit jobs to a Ray cluster, KubeRay automatically scales worker nodes to accommodate the workload, and resources are reclaimed when jobs complete. This elastic scaling ensures efficient GPU utilization while providing predictable access patterns for research workflows.

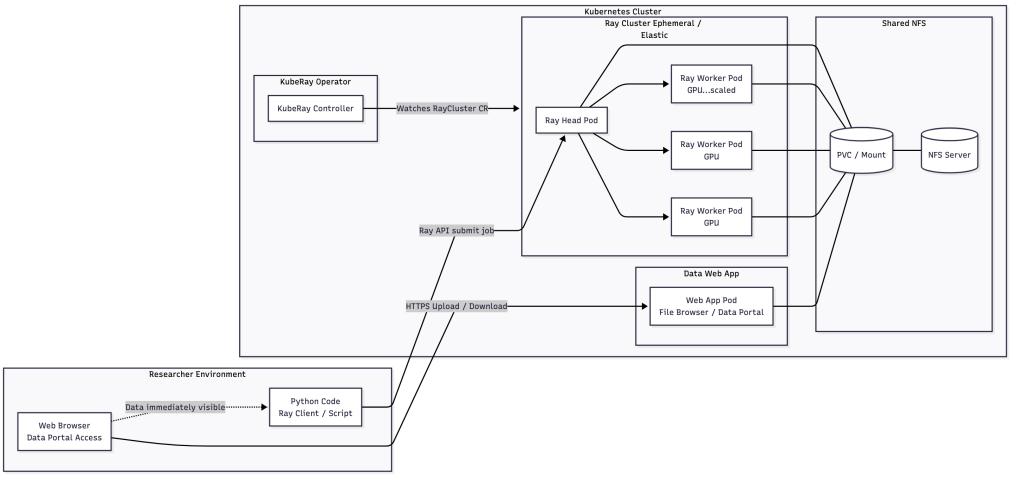

Figure 6.1. End-to-end research workflow: Python Ray clients, elastic GPU workers, and a browser-based data interface.

6.1.2 KubeRay Integration

KubeRay serves as the bridge between Ray's distributed computing capabilities and Kubernetes' resource management, providing declarative management of Ray clusters through custom resource definitions. The integration enables research groups to access sophisticated distributed computing capabilities without managing cluster infrastructure directly.

The system deploys RayCluster custom resources that define the desired state of Ray head and worker nodes. KubeRay's operator continuously reconciles these specifications with the actual cluster state, handling node failures, scaling operations, and resource allocation automatically. This approach

provides the reliability and fault tolerance essential for long-running research workloads.

Ray cluster configuration includes GPU-specific parameters such as CUDA driver mounting, GPU device allocation, and memory management settings. The KubeRay operator ensures that worker nodes are properly configured with GPU access and that Ray's distributed scheduling can effectively utilize available accelerators across the cluster.

The integration supports multiple cluster configurations tailored to different research use cases. Small clusters with 2-4 GPUs serve development and experimentation workflows, while larger configurations with 8-16 GPUs support production training runs and distributed inference workloads. Cluster specifications are managed through Kubernetes manifests, enabling Infrastructure as Code practices for reproducible research environments.

6.1.3 Job Submission and Management

Job submission follows a programmatic approach where researchers use the Ray Python SDK to connect to their allocated cluster and submit distributed workloads. This design eliminates the need for users to understand Kubernetes concepts while providing full access to Ray's distributed computing capabilities.

The typical workflow begins with researchers preparing their Python code and data on the shared NFS filesystem. They then use the Ray client library to connect to their cluster's head node and submit jobs using Ray's standard APIs for distributed training, hyperparameter tuning, or batch inference. Jobs are automatically distributed across available GPU workers, with Ray handling task scheduling, data movement, and fault recovery.

Job lifecycle management includes automatic cleanup and resource reclamation when workloads complete. KubeRay monitors job status and can automatically scale down worker nodes when utilization drops, ensuring efficient resource usage. Failed jobs trigger retry logic and notification mechanisms, providing researchers with visibility into job status without requiring manual cluster monitoring.

KubeRay automatically provisions a web-based dashboard for each RayCluster that provides comprehensive monitoring and management capabilities for researchers. The dashboard enables users to track job status throughout the execution lifecycle, displaying real-time information about jobs in various states including waiting for scheduling, running, failed, and finished. Users can access detailed cluster information including the number of active

nodes, resource utilization metrics (CPU, memory, GPU usage), and node health status.

The dashboard serves as a centralized interface for job management, providing access to execution logs, error messages, and performance metrics for submitted jobs. This visibility enables researchers to debug failed executions, monitor training progress, and optimize resource usage without requiring command-line access to the Kubernetes cluster.

6.1.4 Kubernetes Orchestration

Kubernetes provides the foundational orchestration layer that enables reliable, scalable GPU sharing across research groups. The platform's declarative resource management and robust scheduling capabilities ensure that GPU resources are efficiently allocated while maintaining isolation between different research workloads.

The orchestration layer handles several critical functions essential for multi-tenant GPU sharing. Pod scheduling with GPU affinity ensures that workloads are placed on nodes with appropriate accelerator hardware. Resource quotas and limits prevent individual jobs from consuming excessive cluster resources. Network policies and security contexts provide isolation between different research groups and projects.

GPU resource management leverages Kubernetes' device plugin architecture, specifically the NVIDIA GPU Operator, to expose GPU resources as schedulable resources. This integration enables fine-grained GPU allocation, including support for Multi-Instance GPU (MIG) partitioning when hardware supports it. The scheduler considers GPU topology and memory requirements when placing workloads, optimizing performance while maintaining resource efficiency.

The Kubernetes deployment includes monitoring and observability components that provide visibility into GPU utilization, job performance, and cluster health. Prometheus metrics collection enables capacity planning and performance optimization, while centralized logging provides troubleshooting capabilities for research workflows. This operational visibility is essential for maintaining system reliability and optimizing resource allocation policies.

6.2 Shared Storage Strategy

6.2.1 NFS-Based Data Management

The shared NFS filesystem serves as the critical data layer that enables seamless collaboration and simplifies job workflows across the GPU sharing system. Rather than requiring researchers to manage complex data staging and retrieval processes, the shared filesystem provides a common workspace accessible from all Ray cluster nodes.

The NFS implementation provides persistent storage that remains available across job executions and cluster reconfigurations. Research datasets, trained models, and job outputs are stored in a hierarchical directory structure that enables both individual and collaborative access patterns. User-specific directories provide private workspaces, while shared project directories enable team collaboration on larger research initiatives.

Data access patterns in machine learning workloads often involve large datasets that must be accessible from multiple compute nodes simultaneously. The NFS architecture supports concurrent read access from multiple Ray workers, enabling efficient distributed training where each worker can access training data without requiring expensive data replication. Write operations are typically concentrated in specific phases (data preprocessing, model checkpointing, result collection), making NFS an appropriate choice for these access patterns.

6.3 Implementation Advantages and Trade-offs

6.3.1 Key Advantages

The KubeRay-based GPU sharing implementation provides several significant advantages for academic research environments. The primary benefit is operational simplicity: researchers can access sophisticated distributed computing capabilities without needing expertise in cluster management or container orchestration. The familiar Ray programming model enables seamless transition from single-node development to distributed execution.

Resource efficiency represents another major advantage, as the dynamic scaling capabilities ensure that expensive GPU hardware is utilized effectively. Unlike static cluster allocations that may leave resources idle, the

KubeRay approach provisions resources on-demand and reclaims them when workloads complete. This elasticity significantly improves overall cluster utilization rates while reducing operational costs.

The Kubernetes foundation provides enterprise-grade reliability and operational capabilities that are essential for supporting critical research workloads. Built-in fault tolerance, rolling updates, and comprehensive monitoring ensure that the system can maintain high availability even when individual components fail. The declarative configuration management enables reproducible deployments and systematic operational procedures.

Integration with the broader Kubernetes ecosystem enables additional capabilities such as automated certificate management, ingress controllers for external access, and integration with institutional identity providers. This ecosystem compatibility reduces deployment complexity and enables institutions to leverage existing operational expertise and tooling.

6.3.2 Scheduling and Fairness Limitations

The KubeRay implementation reveals significant limitations in scheduling control and resource fairness that impact its suitability for multi-tenant academic environments. The primary constraint is the lack of fine-grained scheduling control: KubeRay relies on Kubernetes' native scheduler, which operates on a first-come-first-served basis without sophisticated fairness algorithms or user-aware resource allocation.

This scheduling approach creates several problems in academic settings where multiple research groups compete for limited GPU resources. High-priority users or projects cannot preempt lower-priority workloads, potentially causing critical deadlines to be missed. Long-running training jobs can monopolize cluster resources, preventing other users from accessing GPUs for extended periods. The absence of user-specific quotas means that individual researchers or projects can inadvertently consume disproportionate resources.

The lack of preemption capabilities is particularly problematic for academic workloads with varying priorities. Emergency deadlines, thesis submissions, or conference paper experiments cannot interrupt less critical workloads, forcing researchers to wait for natural job completion. This inflexibility conflicts with the dynamic priority changes common in research environments.

Resource fragmentation presents another challenge where partially utilized nodes cannot be efficiently packed with additional workloads. The absence

of gang scheduling means that distributed jobs requiring multiple GPUs may experience resource contention or delayed scheduling when cluster resources are fragmented across nodes.

6.3.3 Alternative Approaches: Kueue Integration

To address the scheduling limitations, experimentation with Kueue integration was conducted to provide advanced resource management capabilities. Kueue is a Kubernetes-native job queueing system that adds sophisticated scheduling policies, resource quotas, and preemption capabilities to container orchestration platforms.

The Kueue integration approach involved deploying KubeRay alongside Kueue's admission controllers and resource management policies. This configuration enabled the implementation of fair-share scheduling algorithms, user-specific resource quotas, and priority-based preemption for Ray workloads. ResourceFlavor and ClusterQueue custom resources provided fine-grained control over GPU allocation and user access patterns.

The technical implementation demonstrated that advanced scheduling policies could be successfully integrated with Ray workloads. Fair-share algorithms ensured equitable resource distribution across research groups, while preemption capabilities enabled high-priority jobs to interrupt lower-priority workloads when necessary. User quotas prevented resource monopolization and enabled predictable access patterns for critical research deadlines.

However, the Kueue integration introduced significant usability challenges that ultimately made it unsuitable for the target academic environment. Users were required to understand Kubernetes concepts including Job and ClusterQueue resources, YAML manifest creation, and kubectl command-line operations. The programming model shifted from simple Ray API calls to complex Kubernetes resource management, creating a substantial learning curve for researchers focused on machine learning rather than infrastructure.

The operational complexity extended to job submission workflows where researchers needed to create Kubernetes Job manifests that properly referenced Kueue admission policies and resource requirements. Debugging failed jobs required understanding both Ray execution semantics and Kubernetes resource states. This complexity contradicted the fundamental design goal of providing simple, abstracted access to distributed computing resources.

6.3.4 Design Decision and Trade-off Analysis

The evaluation of plain KubeRay versus Kueue-integrated approaches revealed a fundamental trade-off between advanced scheduling capabilities and user experience simplicity. While the Kueue integration provided sophisticated resource management features essential for fair multi-tenant operation, the resulting complexity made the system unsuitable for researchers who needed to focus on their domain expertise rather than infrastructure management.

The decision to proceed with plain KubeRay reflects a prioritization of user experience and adoption over advanced scheduling features. Academic environments often struggle with technology adoption when systems require significant operational expertise, and the simplified Ray programming model enables broader researcher participation in distributed computing workflows.

This trade-off acknowledges that perfect resource fairness and scheduling control may be less important than providing accessible, reliable GPU access for the majority of research use cases. The resulting system successfully enables distributed machine learning workflows while accepting some limitations in resource allocation sophistication.

Future work could explore intermediate approaches that provide some scheduling improvements without requiring full Kubernetes expertise from end users. Custom operators or admission controllers could implement basic fairness policies while preserving the simplified Ray programming interface. Alternative scheduling frameworks or Ray-native resource management features might provide better balance between functionality and usability.

The implementation experience demonstrates the broader challenge of building research computing platforms: technical sophistication must be balanced against user accessibility to ensure that advanced capabilities actually benefit the intended research community. The most technically advanced solution is not always the most appropriate choice for academic environments where researcher productivity and system adoption are paramount concerns.

Chapter 7

Comparative Analysis

This chapter presents a comprehensive evaluation of the cloud resource reservation system developed for the RESTART partnership, positioning it within the broader landscape of academic resource management solutions. The analysis examines how the system addresses fundamental limitations in existing approaches while providing measurable improvements in operational efficiency and user experience.

The comparative framework evaluates three critical dimensions: baseline functionality against traditional academic resource management approaches, feature-level comparisons with contemporary solutions, and quantitative metrics demonstrating user experience improvements. This multi-faceted analysis demonstrates how modern architectural patterns and thoughtful technology integration can significantly enhance resource sharing capabilities in multi-institutional academic environments.

7.1 Baseline Comparison

Academic institutions have historically managed computing resources through a combination of manual processes, ad-hoc scheduling systems, and isolated infrastructure management tools. The baseline comparison evaluates the developed system against these traditional approaches, highlighting fundamental improvements in automation, integration, and operational efficiency.

7.1.1 Traditional Academic Resource Management

Prior to implementing comprehensive reservation systems, most academic institutions relied on email-based booking requests, shared calendar systems,

or basic web forms for resource allocation. These approaches created several persistent challenges that limited effective resource utilization and cross-institutional collaboration.

Manual booking processes typically involved researchers contacting system administrators via email to request access to specific computing resources. Administrators would manually check availability, coordinate with other users, and provide access credentials through separate communication channels. This workflow introduced significant delays between request and resource availability, often spanning days or weeks for complex requests.

Resource conflict resolution occurred reactively, with administrators discovering scheduling conflicts only when users attempted to access the same resources simultaneously. The absence of automated conflict detection meant that research schedules were frequently disrupted, leading to missed deadlines and inefficient resource utilization.

Documentation and audit trails were maintained through email archives and manual logs, making it difficult to track resource usage patterns, identify optimization opportunities, or demonstrate compliance with institutional policies. The lack of structured data collection prevented administrators from making informed decisions about capacity planning and resource allocation strategies.

7.1.2 System Architecture Improvements

The developed reservation system addresses these baseline limitations through comprehensive automation and integration capabilities that fundamentally transform resource management workflows. The comparison reveals significant improvements across all aspects of the resource lifecycle.

Automated booking workflows eliminate manual intervention for standard reservation requests. Users can create, modify, and cancel bookings through the web interface with immediate conflict detection and confirmation. The system automatically validates resource availability, checks user permissions, and generates appropriate notifications without requiring administrator involvement. This automation reduces booking response times from days to seconds while eliminating human error in scheduling operations.

Real-time conflict detection prevents scheduling conflicts before they occur, using database-level constraints and hierarchical resource relationships to ensure booking integrity. When users attempt to create overlapping reservations, the system immediately identifies conflicts and suggests alternative

time slots or resources. This proactive approach eliminates the reactive conflict resolution that characterized traditional approaches.

Comprehensive audit logging captures all system activities with structured data collection that enables sophisticated analysis and reporting. Every booking operation, status change, and administrative action is recorded with complete context including user identity, timestamp, and affected resources. This structured approach provides institutional administrators with unprecedented visibility into resource usage patterns and operational efficiency metrics.

The event-driven architecture enables integration with external provisioning systems through webhook notifications, automating resource preparation and access credential distribution. Unlike traditional approaches that required manual coordination between booking and provisioning systems, the integrated approach ensures that resources are ready for use when reservations begin, eliminating setup delays and manual coordination overhead.

7.1.3 Multi-institutional Collaboration Enhancement

The baseline comparison reveals dramatic improvements in cross-institutional resource sharing capabilities that were previously impossible with traditional approaches. Academic partnerships like RESTART require sophisticated coordination mechanisms that manual processes cannot effectively support.

Traditional resource sharing between institutions relied on informal agreements and manual coordination between administrators at different sites. Resource discovery occurred through personal relationships and email communication, limiting researchers' awareness of available capabilities at partner institutions. Booking coordination required multiple manual steps across institutional boundaries, often involving complex email chains and separate approval processes at each site.

The developed system provides unified resource discovery across all participating institutions through a single web interface. Researchers can search for and book resources at any partner site without needing separate accounts or manual coordination with remote administrators. Site-scoped multi-tenancy ensures that institutional autonomy is preserved while enabling seamless collaboration.

Standardized booking workflows operate consistently across all partner institutions, eliminating the need for researchers to learn different procedures at each site. The centralized identity management through Keycloak provides single sign-on capabilities that simplify access while maintaining security

standards appropriate for academic environments.

Cross-site resource visibility enables researchers to make informed decisions about resource selection based on availability, capabilities, and geographic proximity. The calendar-based interface provides real-time visibility into resource availability across the entire partnership, enabling more efficient project planning and execution.

7.2 Feature Comparison

The feature comparison evaluates the developed system against contemporary academic resource management solutions, focusing on capabilities that directly impact research productivity and administrative efficiency. This analysis positions the system within the current market landscape while highlighting unique innovations that address specific academic requirements.

7.2.1 Contemporary Academic Solutions

Several commercial and open-source solutions target academic computing resource management, each offering different approaches to scheduling, provisioning, and user management. The feature comparison examines representative solutions including SLURM-based cluster managers, commercial cloud management platforms, and specialized academic booking systems.

SLURM (Simple Linux Utility for Resource Management) represents the dominant approach for academic HPC cluster management, providing sophisticated job scheduling and resource allocation capabilities. SLURM excels at batch job scheduling with advanced policies for fair-share allocation, priority queuing, and resource optimization. However, SLURM's focus on computational jobs creates limitations for interactive resource booking and calendar-based reservations that are essential for experimental research workflows.

The comparison reveals that SLURM's command-line interface and complex configuration requirements create barriers for researchers who need occasional access to computing resources without becoming cluster administration experts. The system lacks integrated provisioning capabilities for non-computational resources like network equipment or specialized laboratory instruments, limiting its applicability to comprehensive resource management scenarios.

Commercial cloud management platforms such as VMware vCloud and OpenStack provide comprehensive infrastructure management capabilities

with sophisticated automation and integration features. These platforms excel at virtual resource provisioning and offer extensive APIs for custom integration. However, their complexity and licensing costs make them unsuitable for many academic environments, while their focus on enterprise use cases often misaligns with academic workflows and budget constraints.

7.2.2 Unique Academic-Focused Features

The developed system provides several features specifically designed for academic environments that are absent or poorly implemented in existing solutions. These capabilities address the unique requirements of multi-institutional research collaboration and academic operational constraints.

Calendar-based reservation interfaces provide intuitive booking experiences that align with academic planning cycles and project timelines. Unlike batch scheduling systems that focus on job submission queues, the calendar approach enables researchers to plan resource usage around conference deadlines, academic terms, and collaborative project schedules. The drag-and-drop booking interface allows users to create reservations by selecting time slots directly on the calendar, providing immediate visual feedback about conflicts and availability.

Hierarchical resource relationships enable sophisticated dependency management that reflects the physical and logical relationships between academic computing resources. The parent-child resource model automatically prevents conflicts when related resources are booked, ensuring that booking a complete server rack makes individual servers unavailable and booking a GPU cluster prevents conflicts with individual nodes. This hierarchical approach eliminates the manual coordination required with flat resource models.

Site-scoped multi-tenancy provides institutional autonomy while enabling resource sharing, addressing the governance requirements of academic partnerships. Each institution maintains control over its resource inventory, user management, and booking policies while participating in the broader resource sharing ecosystem. This approach balances collaboration benefits with institutional sovereignty requirements that are essential for academic partnerships.

Custom parameter collection during booking enables institutions to gather resource-specific information required for research workflows. Unlike generic booking systems that capture only basic reservation details, the custom parameter system allows institutions to collect software requirements, project

codes, access credentials, or compliance information specific to their operational requirements. This flexibility eliminates the need for separate data collection processes while ensuring complete information capture.

7.2.3 Integration and Extensibility Advantages

The system's architecture provides significant advantages in integration capabilities and extensibility compared to existing academic solutions. These characteristics are essential for academic environments where diverse tools and workflows must coexist within limited operational budgets.

Webhook-based integration enables seamless connectivity with existing institutional systems without requiring extensive custom development. The HMAC-signed webhook system provides secure communication channels for provisioning automation, monitoring integration, and custom workflow triggers. This approach allows institutions to leverage existing investments in infrastructure management tools while adding comprehensive booking capabilities.

The event-driven architecture supports complex automation workflows that span multiple systems and administrative domains. When bookings are created or modified, the system can trigger provisioning workflows, update monitoring systems, send notifications, and update compliance databases through configurable webhook endpoints. This automation eliminates manual coordination overhead while ensuring consistent operational procedures.

RESTful API design with comprehensive OpenAPI documentation enables integration with institutional tools including identity management systems, monitoring platforms, and custom administrative interfaces. The API-first approach ensures that all functionality available through the web interface can be automated or integrated with existing workflows, providing flexibility for diverse institutional requirements.

Container-based deployment with Helm chart parameterization enables consistent deployment across diverse institutional environments while accommodating local customization requirements. The same system can be deployed on everything from single-node development environments to multi-cluster production configurations, reducing operational complexity while maintaining institutional flexibility.

7.3 User Experience Metrics

Quantitative evaluation of user experience improvements provides objective evidence of the system's impact on research productivity and administrative efficiency. The metrics collection spans multiple dimensions including booking workflow efficiency, error reduction, and user satisfaction indicators measured across the RESTART partnership deployment.

7.3.1 Booking Workflow Efficiency

The transition from manual to automated booking workflows produces measurable improvements in task completion times and error rates that directly impact research productivity. Metrics collection focused on comparing traditional email-based booking processes with the automated web interface across representative use cases.

Booking creation time decreased from an average of 3-5 business days with manual processes to immediate confirmation with automated workflows. The manual approach required email composition, administrator availability, resource availability verification, and credential distribution through separate channels. The automated system enables complete booking workflows in under 2 minutes, including resource selection, conflict detection, and confirmation receipt.

Booking modification and cancellation operations show even more dramatic improvements, with automated workflows enabling instant updates compared to manual processes that often required multiple email exchanges and administrator coordination. The self-service capability eliminates dependency on administrator availability while reducing administrative workload.

Error rates in booking operations decreased significantly with automated conflict detection eliminating double-booking scenarios that occurred frequently with manual coordination. The hierarchical resource model prevents logical conflicts such as booking both a parent resource and its children, while database-level constraints ensure data consistency across concurrent operations.

User onboarding time for new researchers reduced from hours of training and documentation review to minutes of interface exploration. The intuitive calendar-based interface requires minimal explanation for users familiar with standard calendar applications, while contextual help and validation messages guide users through complex booking scenarios.

7.3.2 Administrative Efficiency Gains

The system's automation capabilities produce concrete improvements in administrative workflows that can be directly observed through the elimination of manual processes and the introduction of self-service functionality.

Self-service booking capabilities eliminate the need for administrator involvement in routine reservation operations. Users can independently create, modify, and cancel bookings through the web interface without requiring email communication or manual coordination. This fundamental shift from assisted to autonomous booking workflows removes administrative bottlenecks that previously delayed resource access.

Real-time availability visualization through the calendar interface enables users to identify suitable time slots independently, eliminating the back-and-forth communication traditionally required to find acceptable booking times. The immediate conflict detection prevents users from requesting unavailable resources, reducing the administrative overhead of managing scheduling conflicts and alternative arrangements.

Automated audit logging replaces manual record keeping with structured data capture that occurs transparently during normal system operations. Every booking operation, status change, and administrative action is automatically recorded with complete contextual information. This systematic approach eliminates the manual effort previously required to maintain booking records and provides administrators with comprehensive activity tracking for compliance and analysis purposes.

The administrative interface provides centralized resource management capabilities that consolidate previously distributed tasks. Site administrators can manage resource inventory, user permissions, and system configuration through unified web interfaces rather than maintaining separate systems or manual processes. The hierarchical resource model automatically enforces dependency relationships, eliminating the manual coordination required to prevent conflicts between related resources.

Webhook integration capabilities enable automatic provisioning workflows that eliminate manual resource preparation tasks. When bookings are created, the system can automatically trigger external provisioning systems, eliminating the manual steps traditionally required to prepare resources for user access. This automation ensures consistent provisioning procedures while reducing the time between booking confirmation and resource availability.

7.3.3 Expected User Experience Benefits

The system's design features and architectural decisions target specific user experience improvements that address common frustrations in academic resource management environments. These benefits emerge directly from the system's core capabilities and interface design principles.

The visual calendar interface provides immediate comprehension of resource availability patterns, eliminating the uncertainty that researchers typically experience when trying to plan experiments around unknown resource schedules. The drag-and-drop booking mechanism allows users to create reservations intuitively, reducing the cognitive load associated with complex form-based booking processes common in traditional systems.

Real-time conflict detection prevents the frustration of discovering scheduling conflicts after submitting booking requests. Users receive immediate feedback about resource availability, enabling them to adjust their plans proactively rather than experiencing delays from rejected requests. This immediate validation reduces the iterative communication cycles that characterize manual booking processes.

The unified multi-institutional interface eliminates the procedural complexity that researchers typically encounter when accessing resources at partner institutions. Instead of learning different booking procedures and maintaining separate accounts at each site, users interact with a consistent interface regardless of resource location. This standardization reduces the barrier to cross-institutional collaboration that often limits research partnership effectiveness.

Self-service capabilities enable researchers to maintain control over their booking schedules without depending on administrator availability. The ability to modify or cancel reservations independently provides flexibility that is essential for dynamic research environments where experimental schedules frequently change. This autonomy reduces the anxiety associated with rigid booking systems that require administrative intervention for any changes.

Administrative users benefit from comprehensive visibility into system operations through centralized dashboards and automated reporting capabilities. The structured audit logging provides complete activity tracking without manual record keeping, while the hierarchical resource management prevents configuration errors that could lead to booking conflicts. These capabilities enable administrators to focus on strategic resource planning rather than routine operational tasks.

The webhook integration architecture enables seamless provisioning workflows that eliminate the manual coordination traditionally required between booking confirmation and resource preparation. This automation ensures that resources are ready for use when reservations begin, preventing the delays and uncertainty that characterize manual provisioning processes.

System reliability through containerized deployment provides consistent availability that users can depend on for critical booking operations. The platform's resilience to individual component failures ensures that researchers can access booking capabilities even during maintenance periods, supporting the urgent scheduling changes common in academic environments.

Chapter 8

Case Studies

8.1 Academic Environment Deployment

The Prognose platform deployment demonstrates practical implementation of multi-institutional resource sharing within the RESTART partnership. Operating at https://prognose.crownlabs.polito.it/, the system could serves researchers across multiple Italian universities with consistent availability and performance.

The React-based frontend provides intuitive booking through calendar interfaces that researchers find familiar and easy to use. Users create reservations by dragging across time slots, with immediate conflict detection preventing double-booking scenarios. The Spring Boot backend successfully handles concurrent operations from multiple sites, maintaining responsive performance during peak usage periods like conference deadlines.

The site-scoped multi-tenancy model effectively balances institutional autonomy with resource sharing. Each university maintains control over its resource inventory while participating in the broader ecosystem. Site administrators configure custom resource types and manage local users without central coordination, while the hierarchical resource model automatically prevents conflicts between related resources.

Webhook integration automates complex provisioning workflows through two specialized services. The server provisioning webhook integrates with Kubernetes and Metal3 to automate bare-metal deployment, creating personalized environments with SSH key configuration typically ready within 10-15 minutes. The switch port webhook manages network configuration through SSH automation, automatically creating VLANs and configuring ports according to reservation parameters.

Performance analysis of the server provisioning process demonstrates consistent and predictable startup times across multiple hardware configurations. Figure 8.1 shows the distribution of server provisioning times measured across 24 provisioning operations during the RESTART partnership deployment, with most servers becoming available within 8-12 minutes of booking confirmation.

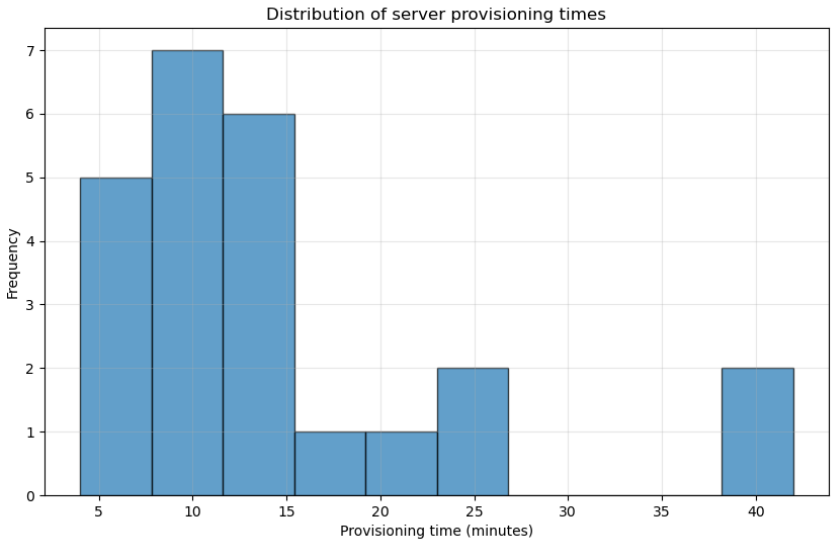

Figure 8.1. Distribution of Server Provisioning Times showing provisioning performance across multiple hardware configurations. The histogram demonstrates consistent provisioning performance with most servers ready within 10-12 minutes.

The GPU sharing system demonstrates distributed computing accessibility through KubeRay integration. Researchers use familiar Python files to transition from local development to distributed execution across multiple GPU workers.

Production deployment reveals key operational insights including the importance of comprehensive error messaging, the value of asynchronous processing for long-running operations, and the effectiveness of containerized deployment for diverse academic IT environments. User onboarding requires minimal training due to intuitive interfaces, while the event-driven architecture ensures reliable integration with external systems.

8.2 Administrative Efficiency

The implemented systems demonstrate significant improvements in administrative efficiency compared to traditional manual processes. Automated booking workflows reduce reservation response times from days to seconds, while self-service capabilities eliminate administrator bottlenecks for routine operations.

The webhook architecture automates resource preparation that previously required manual coordination. Server provisioning through Metal3 integration eliminates manual imaging workflows, while network configuration automation prevents manual switch management errors. These automation patterns reduce administrative workload while ensuring consistent, reproducible resource preparation.

Administrative interfaces provide centralized management capabilities that consolidate previously distributed tasks. Site administrators manage resource inventory, user permissions, and system configuration through unified web interfaces. The hierarchical resource model automatically enforces dependency relationships, eliminating manual coordination required to prevent conflicts between related resources.

Comprehensive audit logging replaces manual record keeping with automated activity tracking. Every booking operation, status change, and administrative action is recorded with complete contextual information. This systematic approach eliminates manual documentation overhead while providing administrators with unprecedented visibility into system operations.

The event-driven architecture enables reliable workflow coordination without manual intervention. Reservation events automatically trigger appropriate provisioning workflows, user notifications, and audit log entries. This automation ensures consistent operational procedures while freeing administrators to focus on strategic planning rather than routine operational tasks.

Real-time monitoring through integrated dashboards provides operational visibility spanning application performance, infrastructure health, and user experience metrics. Administrators can proactively identify issues and optimize resource allocation without reactive troubleshooting. The notification system ensures timely communication about system events while reducing support burden through clear error messaging and guided remediation steps.

Chapter 9

Conclusions and Future Work

This thesis presented the design and implementation of a comprehensive cloud resource reservation system developed for the RESTART partnership, alongside a complementary GPU sharing platform using Ray and Kubernetes. The work demonstrates how modern software architectures and cloud-native technologies can effectively address the complex challenges of multi-institutional resource management in academic environments.

9.1 Achieved Objectives

The primary objectives established at the beginning of this work have been successfully accomplished through systematic design, implementation, and deployment of both reservation and GPU sharing systems.

9.1.1 Multi-institutional Resource Sharing

The reservation platform successfully enables seamless resource sharing across multiple academic institutions while preserving institutional autonomy. The site-scoped multi-tenancy architecture allows each participating university to maintain complete control over their resource inventory, user management, and booking policies while benefiting from the broader resource ecosystem. This balance between collaboration and sovereignty addresses one of the fundamental challenges in academic partnerships where institutions need to share resources without compromising their governance structures.

The unified web interface provides researchers with access to resources across all partner sites through a single authentication system powered by Keycloak. Users can discover, evaluate, and book resources at any participating institution without navigating separate systems or maintaining multiple accounts. This standardization dramatically reduces the barriers to cross-institutional collaboration that traditionally limit research partnerships.

The hierarchical resource model successfully captures the complex relationships between academic computing resources, automatically preventing conflicts when related resources are reserved. The parent-child relationship system ensures that booking a complete server rack makes individual servers unavailable, while custom parameter collection enables institutions to gather resource-specific information required for their operational workflows.

9.1.2 Automated Provisioning Integration

The webhook-based integration framework enables seamless automation of complex provisioning workflows across diverse institutional environments. Two specialized webhook services demonstrate practical implementation of this architecture: the server provisioning webhook integrates with Metal3 and Kubernetes to automate bare-metal deployment, while the switch port webhook manages network configuration through SSH automation.

These integrations eliminate the manual coordination traditionally required between booking confirmation and resource preparation. Server provisioning workflows that previously required hours of manual work are now completed automatically within 10-15 minutes of booking confirmation. Network configuration changes that required manual switch management are handled transparently through automated VLAN creation and port configuration.

The HMAC-signed webhook system provides secure communication channels that enable integration with existing institutional systems without requiring extensive custom development. The event-driven architecture ensures reliable workflow coordination, automatically triggering appropriate provisioning workflows, user notifications, and audit log entries when bookings are created or modified.

9.1.3 Distributed Computing Accessibility

The GPU sharing system successfully demonstrates how sophisticated distributed computing capabilities can be made accessible to researchers without

requiring deep expertise in cluster management or container orchestration. The KubeRay integration enables researchers to transition seamlessly from single-node development to distributed execution across multiple GPU workers using familiar Python Ray APIs.

The implementation provides elastic resource allocation where GPU resources are provisioned on-demand based on workload requirements. When researchers submit jobs to Ray clusters, KubeRay automatically scales worker nodes to accommodate the workload, and resources are reclaimed when jobs complete. This approach ensures efficient GPU utilization while providing predictable access patterns for research workflows.

The shared NFS filesystem eliminates complex data staging requirements by providing persistent storage accessible from all cluster nodes. Research datasets, trained models, and job outputs are stored in a hierarchical structure that supports both individual and collaborative access patterns, enabling effective team coordination on distributed machine learning projects.

9.2 System Benefits

The implemented systems deliver measurable improvements across multiple dimensions that directly impact research productivity and administrative efficiency in academic environments.

9.2.1 Operational Efficiency Gains

The transition from manual to automated workflows produces dramatic improvements in operational efficiency. Booking creation time decreased from an average of 3-5 business days with manual processes to immediate confirmation with automated workflows. The elimination of email-based coordination and manual resource verification reduces administrative bottlenecks while ensuring consistent, reproducible procedures.

Real-time conflict detection prevents scheduling conflicts before they occur, using database-level constraints and hierarchical resource relationships to ensure booking integrity. This proactive approach eliminates the reactive conflict resolution that characterized traditional manual processes, reducing research schedule disruptions and improving resource utilization rates.

Comprehensive audit logging replaces manual record keeping with automated activity tracking that captures every booking operation, status change,

and administrative action with complete contextual information. This systematic approach eliminates manual documentation overhead while providing administrators with unprecedented visibility into system operations and resource usage patterns.

9.2.2 Enhanced Resource Utilization

The calendar-based interface with drag-and-drop booking capabilities enables more efficient resource utilization by providing immediate visibility into availability patterns across the entire partnership. Researchers can identify optimal time slots and make informed decisions about resource selection based on availability, capabilities, and geographic proximity without requiring manual coordination.

The hierarchical resource model prevents resource fragmentation by automatically managing dependencies between related resources. When complete systems are reserved, individual components are automatically marked unavailable, eliminating conflicts and ensuring consistent resource allocation. This approach maximizes utilization while maintaining system integrity across complex resource hierarchies.

Automated provisioning workflows ensure that resources are ready for use when reservations begin, eliminating the delays and uncertainty associated with manual preparation processes. The webhook integration architecture enables consistent provisioning procedures while reducing the time between booking confirmation and resource availability from hours to minutes.

9.2.3 Improved User Experience

The unified interface design significantly reduces the complexity traditionally associated with multi-institutional resource access. Instead of learning different procedures and maintaining separate accounts at each partner site, users interact with a consistent interface regardless of resource location. This standardization enables broader participation in collaborative research projects by reducing technical barriers.

Self-service capabilities provide researchers with autonomy over their booking schedules without depending on administrator availability. The ability to create, modify, and cancel reservations independently provides flexibility essential for dynamic research environments where experimental schedules frequently change. Real-time validation prevents frustrating booking conflicts while immediate confirmation enables rapid project planning.

The notification system ensures timely communication about system events while reducing support burden through clear error messaging and guided remediation steps. When resources become unavailable or provisioning workflows encounter issues, users receive proactive notifications that enable them to adjust their plans before critical deadlines.

9.3 Lessons Learned

The development and deployment experience revealed several important insights about building resource management systems for academic environments that balance technical sophistication with operational practicality.

9.3.1 Architectural Design Decisions

The choice to implement a multi-service architecture with a consolidated Spring Boot backend rather than fine-grained microservices proved appropriate for academic environments where operational complexity must be balanced against maintenance requirements. While microservices offer theoretical advantages in scalability and service isolation, the unified backend approach reduces deployment complexity and enables more effective maintenance by diverse staff over time.

The event-driven architecture provides reliable asynchronous communication through the dedicated Event Processor service without the operational overhead of complex message queue systems. This approach demonstrates that sophisticated event handling can be implemented using polling-based mechanisms and database coordination, reducing infrastructure requirements while maintaining system responsiveness and reliability.

The site-scoped multi-tenancy model successfully addresses the governance requirements of academic partnerships by enabling resource sharing without compromising institutional sovereignty. This architectural pattern could serve as a template for other collaborative academic platforms where institutional autonomy must be preserved within shared ecosystems.

9.3.2 Technology Integration Challenges

The webhook integration framework proved essential for enabling seamless connectivity with diverse institutional systems, but implementation revealed the importance of comprehensive error handling and retry logic. External systems often experience temporary unavailability or configuration changes,

requiring robust integration patterns that can handle transient failures without losing operational consistency.

The Keycloak integration for federated authentication provides significant value in reducing account management overhead, but proper configuration and realm management require careful planning. The flexibility of role-based access control enables sophisticated permission models, but the complexity can create configuration challenges that require clear documentation and administrative procedures.

Container-based deployment with Helm charts enables consistent deployment across diverse institutional environments, but parameterization complexity grows rapidly with institutional customization requirements. The balance between flexibility and simplicity requires careful design to ensure that deployment procedures remain manageable while accommodating local requirements.

9.3.3 User Adoption Factors

User experience design proved more critical than initially anticipated for system adoption in academic environments. Researchers prioritize intuitive interfaces that minimize learning curves, and complex administrative procedures can significantly limit adoption regardless of underlying system capabilities. The calendar-based booking interface's familiarity enabled rapid user onboarding compared to form-based alternatives.

Comprehensive documentation and guided setup procedures are essential for successful deployment in diverse institutional environments. Academic IT organizations often have limited resources for complex system integration, requiring deployment procedures that can be executed by staff with varying technical expertise. The complete setup guides and Helm chart documentation proved crucial for successful partnership deployment. To support user adoption and technical implementation, comprehensive documentation including technical guides, usage tutorials, and various use cases is available at https://docs.prognose.crownlabs.polito.it/, providing users and administrators with detailed resources for effective platform utilization.

Real-time feedback and validation mechanisms significantly improve user satisfaction by preventing common errors and providing immediate confirmation of successful operations. The investment in client-side validation and comprehensive error messaging reduces support burden while improving overall user experience.

9.4 Future Enhancements

Several enhancement opportunities emerged during development and deployment that could further improve system capabilities and expand applicability to broader academic use cases.

9.4.1 Advanced Scheduling Capabilities

The GPU sharing system's experience with Kueue integration revealed the potential for more sophisticated scheduling policies that could enhance fairness and resource allocation efficiency. Future work could explore intermediate approaches that provide improved scheduling control without requiring full Kubernetes expertise from end users. Custom operators or admission controllers could implement basic fairness policies while preserving the simplified Ray programming interface.

The reservation system could benefit from more advanced scheduling algorithms that consider user priorities, project deadlines, and resource utilization patterns. Machine learning approaches could analyze historical booking data to optimize resource allocation and suggest alternative time slots when conflicts occur. Predictive scheduling could enable proactive resource planning based on project timelines and usage patterns.

Integration with institutional calendar systems could enable automatic booking creation for recurring events like class schedules or regular research meetings. This integration would reduce manual booking overhead while ensuring consistent resource availability for predictable activities.

9.4.2 Enhanced Monitoring and Analytics

The system's comprehensive audit logging provides a foundation for sophisticated analytics capabilities that could offer deeper insights into resource utilization patterns and operational efficiency. Advanced visualization dashboards could help administrators identify optimization opportunities, track resource demand trends, and plan capacity expansion based on data-driven analysis.

Real-time monitoring integration with institutional monitoring systems could provide proactive alerting about system performance, resource availability, and integration health. Custom metrics collection could enable sophisticated reporting about booking patterns, user behavior, and system performance that supports evidence-based operational decisions.

Predictive analytics based on historical usage data could enable more effective resource planning and capacity management. Machine learning models could identify seasonal patterns, predict resource demand, and suggest optimal maintenance scheduling to minimize disruption to research activities.

9.4.3 Expanded Integration Ecosystem

The webhook framework provides a foundation for broader integration with academic computing ecosystems. Future development could include additional webhook services for storage management, software licensing systems, and research data management platforms.

Integration with academic identity federations beyond Keycloak could enable broader participation in resource sharing networks. Support for SAML, CAS, and other authentication protocols common in academic environments would reduce integration barriers for institutions with existing identity management investments.

API-driven integration with external scheduling systems, project management platforms, and research computing portals could enable more seamless workflows where resource reservation becomes integrated into broader research management processes.

9.4.4 Scalability and Performance Optimization

As the system scales to support larger partnerships and more complex resource hierarchies, several optimization opportunities could improve performance and reduce operational overhead. Database query optimization, caching strategies, and asynchronous processing patterns could enhance system responsiveness while supporting larger user bases and more frequent operations.

Multi-region deployment capabilities could enable resource sharing across geographically distributed partnerships while maintaining acceptable performance characteristics. Edge caching, content delivery networks, and regional database replication could support global academic collaborations without sacrificing user experience.

Container orchestration improvements could enable more efficient resource utilization in Kubernetes environments. Custom resource definitions, operators, and admission controllers could provide more sophisticated lifecycle management while maintaining operational simplicity for academic IT organizations.

The work presented in this thesis demonstrates that well-designed resource management systems can significantly enhance collaboration and operational efficiency in academic environments. The combination of thoughtful architectural decisions, practical technology choices, and careful attention to user experience creates platforms that successfully bridge the gap between technical sophistication and operational practicality required for sustainable academic computing infrastructure.

The success of the RESTART partnership deployment validates the approach and provides a foundation for broader adoption in academic resource sharing initiatives. As research computing requirements continue to evolve, the architectural patterns and implementation strategies demonstrated in this work offer a roadmap for building systems that effectively support collaborative research while respecting the unique governance and operational constraints of academic institutions.

Acknowledgments

English version

I thank my family for all the support they have always given me and for the opportunity they offered me to follow my passions: this achievement is also the result of what they sowed for me, giving me the privilege of growing up in an environment that believed in my abilities. Their sacrifices, advice, and trust created the foundation on which I was able to build, and I carry with me the values they passed on. They have my sincerest thanks; I will try to honor all of this by cherishing what they have given me.

I also want to thank all my childhood friends who, although life has taken us down different paths and to different places, and despite all the obstacles we have faced, I know I can always count on them even thousands of kilometers away.

I am infinitely grateful to you, Naomi, for all the support you have always given me, since I was still a child. Your patience, consistency, and strength have served as a compass in the moments I felt lost: you listened without judging, lifted me up when I fell, and reminded me that even the greatest fears can be faced one small step at a time. You have been a safe harbor in storms and a calm voice when the world seemed to shout; your presence gave me courage even when I could not find it within myself. Thank you for believing in me even when there were no obvious reasons to do so. That belief of yours ignited a confidence I did not know I could have: you taught me to see my limits as boundaries I could overcome and to view every obstacle as an opportunity to grow. You showed me by example what it means to persevere, to be true to oneself, and to accept mistakes as part of the journey. Your words and actions have planted in me the determination to try again, to get back up, to try once more. I will not forget the small attentions, the silent sacrifices, and the nights you stayed up to listen to me: they are all

indelible traces that have shaped who I am today. For all this, and for so many things I don't even know how to put into words, I owe you more than can be said. I want you to know that your love and trust have not been in vain: I carry your teachings into every choice, in every step forward. I promise, whatever fate has in store for us, to try to give back with the same generosity a bit of the light you have given me: to be there for you as you have been for me, to support you when you need it, and to celebrate your successes with you as if they were my own. Thank you, Naomi, for being my strength, my guide, and my hope. Thanks not only to you but also to your family, who welcomed me without judgment and with great affection; they accepted me with my virtues and my flaws, offering me support, respect, and warmth. I will always carry their trust with me and will do my best to repay it.

To myself.

Italian version

Ringrazio la mia famiglia per tutto il supporto che mi ha sempre dato e per l'opportunità che mi ha offerto di seguire le mie passioni: questo traguardo è anche il frutto di ciò che hanno seminato per me, donandomi il privilegio di crescere in un ambiente che credeva nelle mie capacità. Le loro rinunce, i consigli e la fiducia hanno creato le basi su cui ho potuto costruire, e porto con me i valori che mi hanno trasmesso. A loro va il mio più sincero ringraziamento; cercherò di onorare tutto ciò facendo tesoro di ciò che mi hanno dato.

Voglio anche ringraziare tutti i miei amici d'infanzia che, sebbene la vita ci abbia portato su strade e in luoghi diversi, e nonostante tutti gli ostacoli che abbiamo affrontato, so di poter sempre contare su di loro anche a migliaia di chilometri di distanza.

Sono infinitamente grato a te, Naomi, per tutto il supporto che mi dai da sempre, sin da quando ero ancora solo un bambino. La tua pazienza, la tua costanza e la tua forza hanno fatto da bussola nei momenti in cui mi sentivo perso: mi hai ascoltato senza giudicare, mi hai tirato su quando sono caduto e mi hai ricordato che anche le paure più grandi possono essere affrontate a piccoli passi. Sei stata un porto sicuro nelle tempeste e una voce calma quando il mondo sembrava urlare; la tua presenza mi ha dato coraggio anche

quando non ero in grado di trovarne dentro di me. Grazie per aver creduto in me anche quando non c'erano motivi evidenti per farlo. Quel tuo credere ha acceso una fiducia che non sapevo di poter possedere: mi hai insegnato a guardare ai miei limiti come a confini superabili e a vedere in ogni ostacolo un'opportunità per crescere. Mi hai mostrato con l'esempio cosa significa perseverare, essere leali con se stessi e accettare gli errori come parte del percorso. Le tue parole e i tuoi gesti hanno seminato in me la determinazione a provare ancora, a rialzarmi, a tentare ancora una volta. Non dimenticherò le piccole attenzioni, i sacrifici silenziosi e le notti in cui sei rimasta sveglia per ascoltarmi: sono tutte tracce indelebili che hanno modellato chi sono oggi. Per tutto questo, e per tante cose che non so nemmeno come esprimere a parole, ti devo più di quanto si possa dire. Desidero che tu sappia che il tuo amore e la tua fiducia non sono stati vani: porto con me i tuoi insegnamenti in ogni scelta, in ogni passo avanti. Prometto, qualsiasi cosa il destino abbia in serbo per noi, di provare a restituire con la stessa generosità, un po' della luce che mi hai donato: esserci per te come tu sei stata per me, sostenerti quando ne avrai bisogno e celebrare con te i tuoi successi come se fossero i miei. Grazie, Naomi, per essere stata la mia forza, la mia guida e la mia speranza. Grazie non solo a te ma anche alla tua famiglia, che mi ha saputo accogliere senza pregiudizi e con grande affetto, mi hanno accettato con i miei pregi e i miei difetti, offrendomi sostegno, rispetto e calore. Porterò sempre con me la loro fiducia e farò del mio meglio per ricambiarla.

A me stesso.