Adrien Samuel Marie Lejeune

BUILDING A LOCAL RENDERING CLUSTER AT W COMPANY

Graduation Project presented to the Polytechnic School of the University of São Paulo for the obtention of the Production Engineering degree.

São Paulo
2025

Adrien Samuel Marie Lejeune

BUILDING A LOCAL RENDERING CLUSTER AT W COMPANY

Graduation Project presented to the Polytechnic School of the University of São Paulo for the obtention of the Production Engineering degree.

Advisor: Prof. André Leme Fleury

São Paulo
2025

CATALOGUE CARD

Lejeune, Adrien Samuel Marie

Building a local rendering cluster at W company / Adrien Samuel Marie Lejeune, São Paulo, 2025.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Produção.

1.Engenharia de Produção 2.Start-up 3.Cluster de Computadores 4.Renderização 3D I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Produção II. t.

To public education

ACKNOWLEDGEMENTS

I would like to thank my professor André Leme Fleury for his guidance and support throughout the realization of this work. His availability, trust, and constructive feedback were essential for the completion of this thesis.

I am also grateful to the Classe Préparatoire PC* of Lycée Joffre, in Montpellier, to the Arts et Métiers Institute of Technology, Aix-en-Provence campus, and to the Escola Politécnica of the University of São Paulo for the rigorous scientific and engineering education that formed the basis of my trajectory, developed my professional and technical skills, and opened concrete perspectives for my future career as an engineer.

ABSTRACT

This work examines how a small creative studio can replace ad hoc cloud rendering with a managed local render cluster. Focusing on W company, a Brazilian firm active in architectural visualisation and immersive 3D experiences, the study analyses existing rendering practices, designs a local infrastructure based on Deadline, and documents its implementation on standard on premises hardware. The results indicate that structured render management improves the predictability of rendering, increases utilisation of existing resources, and reduces operational disruption for artists. More broadly, the thesis argues that a local render cluster is a viable and economically attractive option for studios seeking greater control over performance, costs, and confidentiality.

Keywords: distributed computing, local render cluster, resource optimization, render management, Deadline, server infrastructure.

LIST OF ILLUSTRATIONS

LIST OF ABBREVIATIONS AND ACRONYMS

AI - Artificial intelligence based assistant used as a technical support tool
AWS - Amazon Web Services
CLUSTER - Local on premises render cluster built from internal machines
CORONA - Rendering engine by Chaos used with 3ds Max
CPU - Central processing unit
DHCP - Dynamic Host Configuration Protocol for automatic IP address assignment
GPU - Graphics processing unit
HDD - Hard disk drive used for bulk project storage
LAN - Local area network
LAUNCHER - Local Deadline component that starts and supervises the Worker service
MONITOR - Graphical Deadline interface used to submit, monitor and control jobs
NAS - Network attached storage server providing shared volumes
NODE - Machine that participates in the cluster as a render node
PATH MAPPING - Automatic mechanism that rewrites file paths so that local assets resolve correctly on shared storage
RCS - Remote Connection Server that exposes Deadline services to client machines
REPOSITORY - Central Deadline storage for configuration, plugins, scripts and logs
SMB - Server Message Block file sharing protocol used by Windows and Samba
SSD - Solid state drive used for system, database and high speed storage
TCP - Transmission Control Protocol carrying Deadline and file sharing traffic
TLS - Transport Layer Security used to protect connections and client certificates
UI - User interface
UNC - Universal Naming Convention style network path (for example \server\share)
V-RAY - Rendering engine by Chaos used with SketchUp
VM - Virtual machine running on the cluster host
WORKER - Background Deadline process that executes render tasks on each node
ZFS - Copy on write file system and volume manager used on the NAS server

TABLE OF CONTENTS

1. INTRODUCTION

1.1 CONTEXT: W COMPANY, FROM ARCHITECTURAL FOUNDATIONS TO DIGITAL

W company is a Brazilian architecture and real-estate technology firm (proptech) headquartered in São Paulo. Founded in September 2020 by a French architect, the company emerged in the wake of the Covid-19 pandemic with the stated mission of integrating architectural excellence and digital innovation to design agile, human-centred, and sustainable workplaces. Public materials position W company at the intersection of corporate architecture and immersive technologies, and trace its conceptual development to design work conducted in São Paulo since 2013.

1.1.A Foundational trajectory and entrepreneurial motivation

Before setting up the current organisation, the founder had already built a significant track record by directing the Brazilian office of a European corporate-architecture firm. That practice operated in several major cities and delivered large office projects for multinational clients, which gave the founder direct exposure to corporate real-estate strategies, complex stakeholder environments, and high specification fit out work. Industry publications and contemporary records document both the firm's activity in Brazil and the founder's leadership role during that phase.

The internal testimonies collected for this thesis introduce a more intimate dimension to this history. They recount how the former partnership came to an end during the Covid-19 pandemic, after the death of one of its co-founders, and how this event prompted a moment of reassessment. From this perspective, the current organisation can be seen as a new beginning: a conscious attempt to rebuild after personal and organisational disruption, while at the same time positioning the studio in relation to structural shifts in office markets, such as the move towards hybrid work patterns and experience driven workplaces. This storyline is based on internal accounts and is therefore treated as contextual background rather than as a verifiable external fact, but the wider market dynamics it refers to are aligned with contemporary analyses of the office sector and the ongoing hybridisation of workplace practices.

1.1.B Integrated value proposition and dual operating model

From its inception, the organisation articulated an integrated value proposition structured around three complementary pillars: bespoke corporate interior design, high-fidelity three-dimensional visualisation aimed at reducing uncertainty and de-risking decisions prior to construction, and immersive and interactive experiences designed to accelerate stakeholder alignment and support commercial storytelling. In practice, this means that the same team that designs physical workspaces also produces deterministic, presentation-grade imagery and interactive content that help clients understand spatial options, compare alternatives and communicate design intent internally.

This value proposition is reflected in the organisation's operating model, which is structured around two complementary divisions, as illustrated in Figure 1: operating model of W company. The Architecture Division, corresponding to the classical practice, delivers corporate interiors, renovations and competition work using a SketchUp and V-Ray pipeline for modelling and rendering. In parallel, the Digital Innovation Division develops immersive practice, relying on 3ds Max with Corona Renderer for hyper-real stills and Unreal Engine for virtual tours and real-time walkthroughs. Together, these two divisions provide a coherent framework that links architectural design, advanced visualisation and client-experience consultancy within a single studio.

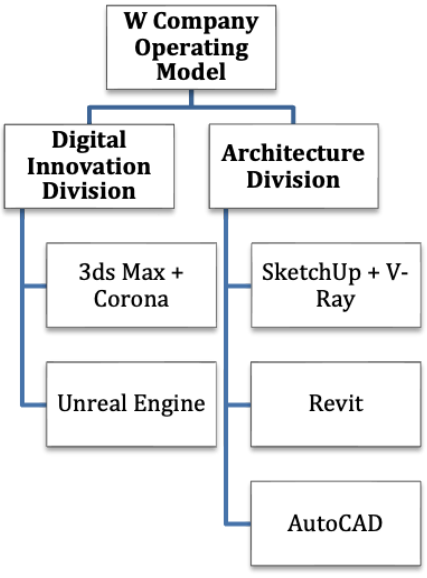

Figure 1: Operating model of W company

1.1.C Portfolio signals, technology stack and production implications

The organisation's portfolio highlights collaborations with major corporate clients and consolidates both conceptual and project work across corporate interiors and architectural visualisation. Publicly accessible project pages illustrate this diversity, presenting examples that combine traditional architectural expression with digitally augmented experiences. More broadly, the collection of showcased projects signals a mixed pipeline comprising built environments, high-fidelity visualisation deliverables, and immersive digital assets. Taken together, these references confirm the organisation's dual orientation, integrating architectural practice with immersive technologies, and demonstrate the strategic use of imagery and interactive experiences as key differentiators in client communication and pre-construction decision-making.

The toolchain underpinning these operations relies on modelling environments, offline rendering engines such as V-Ray and Corona, and real-time visualisation platforms such as Unreal Engine, a combination that is computationally intensive and highly sensitive to delivery deadlines, particularly under iterative client cycles typical of corporate fit-outs and branding-driven projects. These characteristics expose a production-engineering challenge centred on the need to scale rendering capacity while controlling cost and preserving both speed and visual fidelity. In the subsequent sections, this challenge motivates the evaluation of render-management systems and the design of a local computing cluster aimed at reducing dependence on external cloud execution, while the technical properties and operational implications of this toolchain, including its effects on throughput, cost structure and workflow stability, are analysed in detail throughout the thesis.

Figure 2 presents an example of a render produced at W company, illustrating the level of photorealism that characterises the organisation's visual output. This image reflects the integrated workflow described earlier, where architectural modelling and advanced rendering techniques converge to support client communication and design decision making. The clarity of materials, the controlled lighting, and the immersive spatial composition visible in this example are representative of the standards that guide the studio's deliverables. More broadly, it also illustrates the computational intensity and iterative refinement that define the company's production model, providing a concrete reference point for the technical and operational challenges examined later in this thesis.

Figure 2: Examples of a render produced at W company

1.2 PROBLEM DEFINITION: CLOUD RENDERING AND OPERATIONAL CONSTRAINTS

This section formalises the production engineering problem that motivates the remainder of the thesis. Building on the description of W company's dual production model and toolchain, with one division focused on architecture and another on digital visualisation relying on V-Ray, Corona, and Unreal Engine, it examines how reliance on cloud rendering emerged as a pragmatic response to local capacity constraints, and how time based pricing models interact with iterative visual workflows to generate cost volatility. Together, these operational and economic pressures define the constraints that the subsequent design of a local, queue based render cluster is intended to address.

1.2.A Cloud rendering as a temporary response to local production constraints

The operating model, with one division focused on architectural production and another on digital visualisation, generates large volumes of photorealistic imagery under tight deadlines, with iterative client feedback acting as a central quality driver. In this context, the organisation currently relies on a small pool of high performance workstations that serve simultaneously as designers' primary machines. Rendering tasks are typically launched manually on these personal workstations, and no internal render queue or local farm is available to enable unattended, overnight or parallel execution. In practical terms, daytime productivity directly competes with rendering operations, since a workstation engaged in rendering cannot be used for modelling, lighting work or client revisions, and artists must actively monitor jobs rather than delegating them to an automated pipeline.

Given these local constraints, the organisation has temporarily relied on cloud based rendering solutions as a way to absorb peaks in demand and to process multiple images concurrently without blocking local workstations. Commercial render farms provide elastic and on demand computational capacity, with straightforward submission workflows and native integration with widely used rendering engines such as V-Ray, developed by Chaos Group, and Corona. They support remote batching, queuing and parallel execution of scenes, so that rendering tasks can be executed off site while designers continue working on their primary machines. In effect, the cloud platform functions as an external queue and execution layer, compensating for the absence of a local render farm and allowing the studio to maintain creative throughput during intensive delivery cycles, at the cost of shifting the bottleneck into a consumption based external service.

1.2.B Time based pricing, iterative workflows and cost volatility

Most commercial render farms charge for compute time rather than for a finished image. Provider documentation makes this explicit by tying cost to the actual duration consumed on farm nodes, often measured to the second, while online calculators are positioned as approximate estimates prior to submission. For a still image pipeline of the type operated by the organisation, this pricing structure has two immediate consequences. First, the price of a single render is inherently variable, because it depends on scene complexity, resolution, denoising parameters and lighting choices rather than on a fixed per image tariff. Second, any client driven iteration, such as a modification of material properties, a background change or a camera adjustment, triggers new compute time and therefore new expenditure, even when the visual difference is relatively modest. Technical explanations from rendering engine vendors reinforce this logic by explicitly linking both render time and cost to scene complexity and to the characteristics of the hardware executing the job.

Internal project records illustrate how these mechanics translate into unit economics. Under standard provider settings, a typical cloud render for catalogue style imagery has generally been priced around US$ 2 to 3 per image. For brand accurate delivery, however, it is common for each image to require two or three successive iterations, so that the effective cost per final output can approach approximately US$ 10. While these amounts may appear modest when considered in isolation, the cumulative effect across many deliverables and rounds of revision becomes material, especially for a studio whose differentiation rests on visual fidelity and rapid iteration and that therefore deliberately invites feedback loops with clients and external collaborators. Under time based cloud billing, each additional loop multiplies compute minutes, turning robust creative practice into budget volatility. Combined with the structural local limitations of a workstation based setup without a render queue, this dependence on consumption based cloud rendering defines the core production engineering problem that motivates the design of a fully local, queue based render cluster intended to internalise capacity and stabilise costs.

As a synthesis of these intertwined constraints, Figure 3 presents an overview of the rendering workflow, showing how local workstation overload, reliance on cloud platforms as an external execution layer, and time based pricing mechanisms combine to create the operational and economic pressures examined in this section.

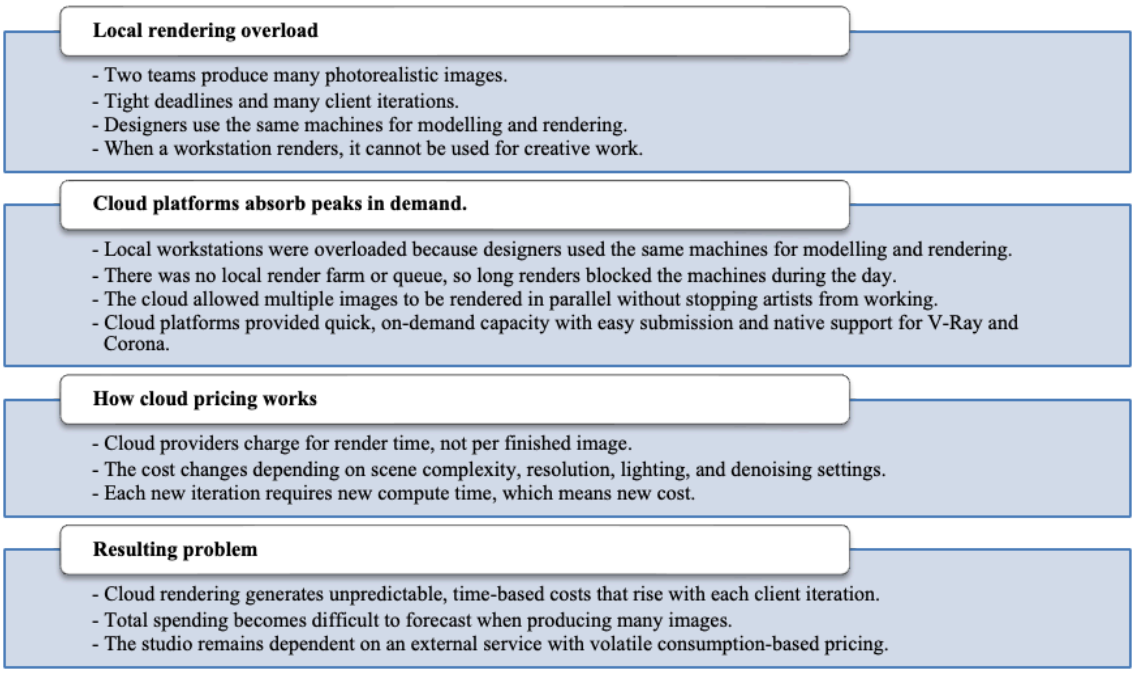

Figure 3: Rendering workflow constraints and cost structure

1.3 OBJECTIVE: A LOCAL CLUSTER

Building on the earlier description of W company's operating model and its reliance on cloud rendering to compensate for limited local capacity, this section specifies the target state that motivates the remainder of the thesis. The objective is not only to move away from ad hoc dependence on external render farms, but to redesign rendering as a managed production process executed on a local computing cluster. The following subsections translate the problem into a strategic objective, describe the intended local architecture and operating model, and outline the main outcomes against which the initiative will later be evaluated.

1.3.A Strategic objective and problem translation

The central strategic objective is to eliminate manual, workstation bound rendering on designers' personal machines and to replace it with a disciplined, queue based system executed entirely within the company's own infrastructure. The organisation currently operates with two intertwined fronts, architectural production and digital visualisation, both of which rely on rapid cycles of photorealistic imagery and deliberate client feedback. Under the previous arrangement, the same high performance computers were used both for creative work and for rendering, so that every heavy render interrupted modelling or lighting activities and encouraged recourse to cloud services priced by compute time.

In this context, the objective can be reformulated as the internalisation of the useful properties of cloud platforms without their exposure to consumption based billing. The local cluster should provide parallel submission, unattended execution and predictable turnaround while allowing designers to continue working on their primary tasks. A queue based render manager, understood here as a software tool that centralises jobs and assigns them to available machines according to explicit rules, becomes the backbone of this transformation. Instead of artists launching and supervising renders one by one, jobs are placed in an ordered list and executed according to a chosen queue discipline, that is, a set of policies that determine which tasks run first based on criteria such as project, deadline or client importance. Rendering is thus reframed as an orchestrated background service that supports, rather than competes with, daytime creative work.

1.3.B Target local architecture and operating model

The target local architecture centres on the deployment of a render management tool that acts as an orchestration layer across the organisation's existing machines. This tool maintains a shared Repository, defined as a central storage area for configuration, plugins and logs, and exposes a common job queue through which any authorised workstation can submit, monitor and control renders. Each participating machine runs a Worker, that is, a lightweight execution service responsible for pulling tasks from the queue and running them with the specified settings. In practice, the cluster is built first by pooling the most capable internal workstations as render nodes, transforming them from isolated personal assets into a coordinated computing resource that can be scheduled as a whole.

Compatibility with the current production pipelines is a non negotiable constraint of this design. On the architectural side, SketchUp scenes rendered with V-Ray must be exportable in a form that can be reliably executed on other machines, either through dedicated submission plugins or through standardised scene exports. On the digital visualisation side, 3ds Max with Corona and, where applicable, Unreal Engine must be able to submit stills or animations to the same queue without fragmenting storage or configuration. All inputs and outputs are expected to reside on clearly defined folders under a single network location, with stable naming conventions that minimise missing textures and duplicated assets and that prepare the ground for later use of Path Mapping, the automatic mechanism that rewrites file paths so that assets saved locally appear consistently on shared storage.

The operating model associated with this architecture formalises how work flows through the cluster in everyday practice. Designers remain responsible for their own submissions, including selecting appropriate presets, choosing output locations and verifying that assets are correctly referenced before dispatching jobs. A light governance structure designates a "queue steward" role, rotating if necessary, to review priorities at the end of the day, align the queue with delivery commitments and prepare off hours execution. Agreed time windows, typically evenings and weekends, are reserved for long or heavy batches that would otherwise degrade workstation responsiveness during client facing hours, while very short preview renders can remain local and interactive. Over time, these routines constitute an operational cadence in which rendering follows shared rules rather than individual habits, making the cluster predictable and manageable with limited overhead.

1.3.C Expected outcomes and evaluation

The local cluster is expected to deliver a set of outcomes that respond directly to the constraints described earlier in the chapter. On the productivity side, shifting heavy rendering to unattended windows and distributing jobs across multiple nodes should keep designers' primary machines responsive for modelling, lighting and client sessions during the day. Batches of images that previously ran sequentially on a single workstation are expected to progress in parallel, reducing lead times for catalogue style deliveries or intensive campaign work. On the economic side, routine iterations that once multiplied cloud expenditure will execute on hardware already owned by the company, so that the marginal cost of exploring an additional creative option becomes almost negligible in accounting terms. This shift from variable, usage based expenditure to predictable internal costs is intended to stabilise project budgets and improve the accuracy of commercial proposals.

Beyond schedule and cost, the cluster is also intended to improve reliability, reproducibility and scalability of rendering operations. Consistent presets, disciplined storage paths and visible job states in the render management tool should reduce failed renders and missing assets, lowering rework and frustration for artists. The governance framework is designed so that additional internal machines can be added as nodes with minimal friction when project loads require extra capacity, allowing the cluster to grow incrementally rather than through abrupt step changes. Subsequent chapters will evaluate these expectations using both quantitative indicators, such as render times and workstation availability, and qualitative feedback from users on workflow fit and perceived stability, thereby connecting this objective statement to the concrete results observed in practice.

1.4 JUSTIFICATION: REDUCING COST BY ELIMINATING CLOUD DEPENDENCE

Replacing routine cloud rendering with a fully local, queue based cluster at W company is not a purely technical choice. It is a response to a concrete combination of financial volatility and day to day friction in production. When high value artist workstations are used both for modelling and for rendering, long jobs block creative activity and often push work into nights and weekends. When the same jobs are offloaded to external farms, invoices arrive in a fragmented and unpredictable way, indexed to how many iterations the client ultimately demands rather than to a stable production plan.

The proposed architecture addresses these issues by turning external, per minute billing into predictable internal capacity and by decoupling creative work from long running renders. It also gives the company closer control over the rendering pipeline and creates room for systematic learning about how rendering behaves in practice. For clarity, the argument is structured in two parts. Section 1.4.A concentrates on economic and operational effects, while the following subsection focuses on technical control and organisational learning, so that both financial and non financial benefits can be evaluated in a coherent way.

1.4.A Economic and operational rationale for local execution

The organisation’s production model is intrinsically iterative. Visual assets evolve through successive feedback rounds until they reach the required level of quality, and each new version typically requires at least one fresh render. In commercial render farms, pricing is tied directly to elapsed compute time, so every correction starts a new billing interval. Internal records indicate that typical renders have been charged around US$ 2 to 3 per image and, with two to three further iterations for brand accurate delivery, often approach roughly US$ 10 per final deliverable. This consumption logic converts creative rigour into budget volatility, since every additional refinement increases spend, and it complicates forecasting across image libraries and campaigns, because the number of client feedback loops cannot be known in advance.

A fully local architecture reverses this relationship. Once the cluster hardware has been procured and amortised, the marginal cost of an additional preview or re render is close to zero in accounting terms. Unit economics are no longer driven by opaque external tariffs, but by internal capacity planning decisions, including how many nodes are available, which queue discipline is used to decide which jobs run first, and which time windows are reserved for heavy computation. In practical terms, expenditure shifts from uncertain operating costs to predictable in house costs. This

stabilises budget governance, improves cash flow planning, and makes client quotations more accurate, while also eliminating ancillary cloud charges such as data egress, priority surcharges, and hidden transaction costs linked to uploads, retries, or failed submissions.

On the operational side, installing a render management layer such as Deadline and pooling high specification machines into a local cluster directly addresses the constraints of manual, workstation bound rendering. Instead of each designer launching and supervising renders individually, Deadline maintains a central queue of jobs and dispatches them to available machines according to priority and eligibility rules. The team can submit long job lists in one step, coordinate priorities across projects, and rely on unattended execution, so rendering becomes a background production flow rather than an interruptive task tied to a single workstation.

Batch scheduling then moves heavy computation to evenings and weekends, freeing workstations for modelling, lighting, and client sessions during office hours, while parallel execution across all available nodes converts idle local capacity into throughput without external outsourcing. For high volume contexts such as catalogue drops or marketing pushes, the queue can saturate local resources overnight and recover the next morning with dozens or hundreds of frames advanced, without degrading daytime responsiveness. Standardised presets, which distinguish preview from final passes and catalogue stills from interiors, act as predefined submission templates that reduce friction and misconfiguration. Simple telemetry on job duration, failure rate, and queue waiting time surfaces bottlenecks early and supports incremental improvement of operational practices as the cluster becomes part of everyday work.

1.4.B Operational control and organisational capability

A local first design gives W company direct operational control over the entire rendering workflow, from scene preparation to final output, by keeping inputs, assets, and results inside the company network. Scene files and textures remain on internal shared volumes used jointly by the architecture and visualisation teams, which reduces relinking errors, avoids large uploads and downloads of high resolution imagery, and removes dependence on external network conditions. The core Deadline components are also deployed on premises. The Repository, understood as the central storage for configuration, plug ins, and logs, the Remote Connection Server (RCS), which brokers secure communication between user interfaces and the render farm, and the Worker services, which are the agents executing render tasks on each node, all operate under the same internal policies. This uniformity supports reproducibility, because the same software versions, presets, and network paths apply

across all nodes, which simplifies incident diagnosis and makes behaviour easier to audit.

Failure handling becomes more predictable under this local control. The queue can automatically retry failed tasks on another node, temporarily isolate problematic jobs, and record detailed error conditions for later review, which reduces rework and unplanned downtime. At the same time, local execution limits dependency on external providers. The company is less exposed to vendor lock in, price changes, API deprecations, or quota constraints in external services, and confidentiality for client material is managed through internal rules on role based access, versioned storage, and auditable paths. In practice, these technical choices translate into smoother day to day operation, particularly during peak periods when delivery deadlines and rendering demands coincide.

Beyond the infrastructure, the move to a local cluster supports the development of a shared organisational capability around rendering. Removing the cloud as a routine option encourages the adoption of a standard way of rendering inside W company. Instead of each artist managing a personal workflow on an individual workstation, shared folder structures, stable naming conventions, and concise submission presets become part of everyday practice, which reduces hand off friction between teams and shortens onboarding for new staff and recurring collaborators. A lightweight operating cadence, understood as the regular rhythm of activities such as end of day queue reviews, weekly node health checks, and a brief incident log, keeps the system reliable without excessive bureaucracy and provides clear touchpoints where issues can be raised and resolved. Over time, technicians learn to operate the queue, maintain nodes, and refine presets, and designers learn to frame submissions with shared quality gates such as preview, pre final, and final. These routines accumulate into organisational knowledge, including checklists, templates, and playbooks, that raise baseline reliability, reduce the likelihood of costly last minute fixes, and transform rendering from a fragile individual practice into a robust collective capability.

1.5 Structure of the document

The remainder of this document is structured to guide the reader from context and problem definition to theoretical framing, methodological design, concrete implementation and observed results, before closing with a global synthesis. Chapter 1, in which this subsection appears, has an explicitly introductory role. Section 1.1 presents W company and its evolution, explaining how an architectural practice progressively incorporated digital visualisation and real estate technology into a single studio. Section 1.2 then formulates the production and cost problem created by reliance on cloud based rendering in an iterative, image intensive workflow. Section 1.3 states the objective of deploying a local, queue based render cluster and clarifies the intended operating model. Section 1.4 justifies this objective from economic, operational, technical and organisational perspectives, arguing that rendering should be treated as a managed, software intensive process rather than a set of isolated actions on individual workstations.

Building on this foundation, Chapter 2 develops the literature review that provides the analytical lens for the rest of the thesis. Section 2.1 revisits the foundations of software engineering, including how projects are organised into processes and life cycle phases, how requirements and stakeholders are identified, and how quality attributes such as performance or reliability are used to shape design choices. Section 2.2 introduces core concepts in software architecture and distributed systems that will later be used to describe the cluster as a software intensive system, with attention to architectural views and patterns relevant for render farms. Section 2.3 examines testing, validation and monitoring in distributed environments, extending the usual testing perspective to long running, asynchronous workloads. Section 2.4 focuses on organisational change and governance, discussing how roles, routines and user engagement influence the success or failure of technical initiatives. Chapter 3 then translates this body of theory into a four phase method for implementing a local cluster at W company. It describes how the initial diagnostic and requirements were established, how they informed the architectural design, how this design was turned into a concrete network and storage topology and an ordered sequence of implementation and testing steps, and how governance, roles and operating cadence were defined so that the solution could be sustained in everyday work.

Chapter 4 constitutes the empirical counterpart of this method. It documents the results of applying the four phases in the specific context of W company. One section is devoted to the implemented architecture of the cluster, detailing the control plane, the characteristics of the server host, and the integration of the network and shared storage into production. A subsequent section follows the operation in daily work, tracing how jobs move from designers' workstations through submission and queuing to execution on Workers. Another section reports operational results and user experience, combining quantitative indicators such as execution times and queue behaviour with qualitative feedback from designers and operators. The chapter also describes the supporting artefacts, including documentation, training activities and onboarding materials, that enable the internal team to operate and evolve the cluster. It closes with a section on limitations and future improvements, addressing both technical and competence related aspects. Finally, the conclusion revisits the initial objectives in light of the results, summarises the contributions for W company and for the broader discussion on local render clusters, and suggests avenues for further work and generalisation, followed by the bibliographic references.

2. LITERATURE REVIEW

The purpose of this chapter is to assemble the theoretical concepts that support the design and deployment of a local render management infrastructure at W company. Instead of starting from generic life cycle models, the discussion adopts a pattern based view of software and systems engineering, in which architectures are built around recurring solutions to recurring problems and applications are treated as structured compositions of components and services. Martin Fowler's work on enterprise application patterns is central to this perspective, since it emphasises how stable architectural structures can guide the organisation of logic, data and deployment topologies in complex systems, while Thomas Erl's contributions on service oriented and cloud based architectures provide the vocabulary to compare cloud platforms and on premise services in a consistent way (FOWLER, 2002; ERL, 2013).

Building on these technical foundations, the chapter then turns to literature on deployment, operations and DevOps, which is directly relevant to turning a conceptual architecture into a working computing cluster that behaves reliably in production. Jez Humble's work on Continuous Delivery describes how disciplined build, test and deployment automation can make software releases predictable and reversible, while Gene Kim's studies of high performing IT organisations show how operational practices and culture jointly determine the stability and throughput of technology platforms (HUMBLE, 2010; KIM, 2016). In the context of W company, these contributions underpin the discussion of how a central repository, worker processes, monitoring interfaces and local agent services should be deployed, monitored and evolved as an integrated service instead of a one off installation.

Finally, the chapter incorporates an organisational perspective, since implementing a local cluster affects not only servers and network diagrams but also roles, responsibilities and everyday routines. John P. Kotter's model of leading change highlights the importance of creating urgency, forming a guiding coalition, articulating a clear vision and anchoring new practices in organisational culture, which are all directly relevant when a studio moves from ad hoc cloud rendering to a centrally managed internal queue (KOTTER, 1996). The sections that follow apply these combined insights from Fowler, Erl, Humble, Kim and Kotter to frame the technical foundations of a local render infrastructure, the architecture and operations of the computing cluster, and the governance and change mechanisms required for its sustained adoption at W company.

2.1 TECHNICAL FOUNDATIONS

2.1.A Technical view on software and systems engineering

In this thesis, software and systems engineering are understood through the lens of architecture and patterns rather than as a purely document driven process. In his work on enterprise application patterns, Martin Fowler presents large software systems as structured compositions of components that solve recurring problems in standardised ways, and argues that the central task of software engineering is to organise code, data and infrastructure so that these recurring problems are treated systematically instead of ad hoc. This perspective emphasises that even when the focus is on deploying and operating existing software rather than writing all of it from scratch, engineers still make architectural decisions about how responsibilities are divided, how components interact and how the system will evolve over time (FOWLER, 2002).

Within this view, software architecture becomes the set of structural decisions that shape how a system behaves and how costly it will be to change. Fowler describes how typical enterprise systems can be organised into layers for presentation, application logic and data access, and shows that these layers can be combined with patterns for handling transactions, integrating external services and managing state, so that complex behaviour emerges from a clear structural backbone rather than from scattered implementation choices. Choices about layering, deployment boundaries, integration mechanisms and the responsibilities of major components are treated as architectural because they are difficult to reverse and because they determine how easily the system can be adapted, monitored and operated in the future (FOWLER, 2002).

From this pattern based perspective, infrastructure projects are included within software engineering to the extent that they involve designing and maintaining such structures. Decisions about how to organise control services and worker processes, how to define shared storage conventions or how to expose internal capabilities as services are examples of architectural choices in Fowler’s sense, even when the underlying software is largely off the shelf. Treating these decisions as part of software and systems engineering, rather than as informal configuration, supports a more rigorous analysis of local computing infrastructures, including render clusters and other specialised platforms, and provides the conceptual basis for the later chapters of this thesis (FOWLER, 2002).

2.1.B Service oriented and cloud perspectives for local infrastructure

Thomas Erl's work on cloud computing provides a detailed foundation for understanding how computing resources can be organised as services rather than as collections of isolated machines. In his treatment of cloud environments, he describes clouds as distributed technology platforms that expose computing, storage and networking capabilities through standardised service interfaces, with clearly defined contracts, quality properties and management mechanisms (ERL, 2013).

This perspective emphasises that what distinguishes cloud systems is not only the physical location of servers, but the way resources are made available as reusable services, for example infrastructure as a service or storage as a service, with predictable behaviours and formalised responsibilities between providers and consumers (ERL, 2013).

Within the same body of literature, service orientation is presented as a general design paradigm in which capabilities are encapsulated into services with explicit contracts, loose coupling and clear boundaries. Erl explains that services are designed to be autonomous units that can participate in larger solutions through composition, governed by principles such as statelessness, standardised interfaces and policy driven behaviour, so that different services can be combined without tight dependencies. This approach provides a vocabulary and a set of patterns for structuring complex systems as networks of services instead of monolithic applications, linking technical mechanisms and business level concerns through service definitions that can be reused across multiple solutions (ERL, 2013).

A further contribution of this literature is the articulation of cloud delivery and deployment models, which show how service orientation interacts with economic and governance considerations. Erl distinguishes between models such as software as a service, platform as a service and infrastructure as a service, and between public, private and hybrid deployment options, arguing that each combination implies a specific distribution of control, responsibility and risk between consumers and providers (ERL, 2013).

From this point of view, a local computing infrastructure can be designed to behave in a cloud like manner for its internal users by applying the same service oriented principles that underpin public clouds. Internal capabilities such as compute capacity for specialised workloads, shared storage or scheduling mechanisms can be encapsulated into services with clear contracts, access interfaces and quality expectations, even when all hardware remains on premises and under direct organisational control. The literature therefore suggests that the contrast between external cloud platforms and local infrastructures is largely a matter of governance

scope and service design, and that the conceptual tools developed for cloud computing can be used to reason about how internal clusters and other local resources are structured, exposed and managed as shared services inside an organisation (ERL, 2013).

2.2 ARCHITECTURE, DEPLOYMENT AND OPERATIONS OF THE COMPUTING CLUSTER

2.2.A Architectural structure in enterprise systems

In the literature on enterprise applications, software architecture is described as the way a system is organised into components that collaborate through well defined responsibilities and interactions. Martin Fowler characterises enterprise applications as systems that display, manipulate and store large volumes of complex data in order to support business processes, and he argues that their recurring design problems are best addressed through a catalogue of architectural patterns rather than isolated solutions (FOWLER, 2002).

A central theme in this work is the use of layered architecture to separate concerns within enterprise systems. Fowler describes how typical applications can be organised into layers for presentation, application logic and data access, each layer depending on the ones below but remaining independent from the layers above, which helps limit the impact of change and reduces the risk of uncontrolled coupling across the system (FOWLER, 2002).

Building on this perspective, Fowler emphasises that enterprise architectures usually distinguish user facing interfaces, domain logic that contains the rules and calculations of the business, and infrastructure components that handle storage, communication and integration with external services. These elements are combined through patterns that clarify which part of the system is responsible for which function and where modifications should be made when requirements evolve, so that complex behaviour emerges from well understood structural decisions instead of ad hoc code scattered across the application (FOWLER, 2002).

Finally, the literature stresses that these structural choices have technical and economic consequences, because they influence how easily a system can be adapted, monitored and operated over time. An architecture in which responsibilities are clearly separated, deployment boundaries are explicit and interactions between components follow known patterns tends to reduce the cost and risk of future changes, whereas a structure with blurred boundaries and implicit dependencies makes each modification more uncertain and expensive, especially in large scale enterprise environments (FOWLER, 2002).

2.2.B Deployment pipelines and DevOps practices

The literature on Continuous Delivery describes deployment as a disciplined, automated flow rather than a series of manual, ad hoc steps. Jez Humble presents the deployment pipeline as a central pattern in which every change to the code base passes through a sequence of automated stages, from build and test to deployment in production like environments, with the aim of making releases frequent, predictable and low risk. Automation of build, test and deployment is treated not only as an efficiency gain, but as a way to reduce human error and to ensure that the same process is followed every time a change is promoted, so that failures can be traced and corrected systematically instead of being attributed to unique release events (HUMBLE, 2010).

Within this framework, a deployment pipeline also embodies a specific attitude toward system evolution. Rather than accumulating large batches of changes and releasing them infrequently, Humble argues that small, incremental updates that flow continuously through the pipeline tend to be safer and easier to diagnose, because any defect can be associated with a limited set of modifications and rolled back quickly if necessary. This approach implies that the infrastructure needed for an application, such as configuration, scripts and environment definitions, should be treated as versioned artefacts that move through the same pipeline as the application code, so that the entire system, including its runtime environment, can be reproduced and deployed in a controlled and automated way (HUMBLE, 2010).

Gene Kim extends this technical view by situating deployment practices inside a broader DevOps perspective that links engineering methods to organisational performance. In his work on high performing technology organisations, DevOps is presented as a set of principles and practices that integrate development and operations through fast feedback loops, shared responsibility for reliability and a culture of continual learning. Empirical studies reported in this literature associate practices such as automated testing, continuous integration, small batch changes and proactive monitoring with shorter lead times for changes, higher deployment frequency and lower change failure rates, suggesting that disciplined deployment pipelines are central to both agility and stability in modern IT environments (KIM, 2016).

Taken together, the contributions of Humble and Kim converge on a view of deployment and operations in which infrastructure is managed through code, changes are propagated via repeatable pipelines and teams are organised to treat reliability as a continuous, measurable outcome rather than as a separate phase at the end of a project. The deployment pipeline becomes not only a technical mechanism for releasing software, but also a coordination device that structures how work flows across roles and environments, from initial commit to production, and how

information about failures and performance is fed back into design and implementation decisions (HUMBLE, 2010; KIM, 2016). This combined perspective provides the conceptual background for later discussions of how computing clusters and supporting services can be deployed, updated and operated using the same principles of automation, small changes and shared ownership that characterise Continuous Delivery and DevOps.

2.2.C Cloud versus local deployment decisions

In the cloud computing literature, deployment choices are framed as architectural decisions that combine technical mechanisms with business and governance considerations. Cloud platforms are described as distributed environments in which computing, storage and networking resources are exposed as services with standardised interfaces and explicit quality properties, and where a rich set of concepts and models helps compare different service and deployment options, such as public, private or hybrid cloud arrangements (ERL, 2013).

Within this body of work, service orientation is presented as a design paradigm that can be applied independently of whether the underlying infrastructure is hosted externally or remains on premises. Capabilities are encapsulated into services with explicit contracts, loose coupling and clear boundaries, and an internal environment can be organised as a private cloud like setting in which resources are provided as shared services to multiple consumers, even though the hardware is fully controlled by the organisation itself (ERL, 2013).

The literature on Continuous Delivery and DevOps complements this view by emphasising that cloud based and local deployments face similar requirements for automation, repeatability and operational feedback. Deployment pipelines, automated configuration and proactive monitoring are treated as general mechanisms for achieving frequent, low risk releases and stable operations, regardless of where the servers are physically located, and high performing organisations are characterised by their ability to apply these practices consistently across all environments rather than by a particular choice of hosting model (HUMBLE, 2010; KIM, 2016).

2.3 GOVERNANCE, OPERATIONAL CADENCE, AND ORGANISATIONAL CHANGE

2.3.A DevOps culture, feedback loops and operational cadence

In the literature on Continuous Delivery, deployment is described not as a final step at the end of a project, but as a repeatable flow that is integrated into everyday work. Jez Humble defines Continuous Delivery as the ability to keep software in a state where it can be released to users safely and quickly at any time, using automated build, test and deployment processes to make releases routine rather than exceptional events. Within this perspective, the deployment pipeline becomes the central organising pattern, since every change must pass through the same sequence of stages, which provides visibility into the status of each modification and reduces the risk that local, manual practices will undermine system stability (HUMBLE, 2010).

Gene Kim extends this technical view by presenting DevOps as an approach that links such automation to organisational culture and performance. In his work on high performing technology organisations, DevOps is described as a set of principles and practices that integrate development and operations through shared goals, fast feedback and continual learning, with the explicit objective of improving both deployment speed and reliability. Drawing on longitudinal survey data, this literature reports that organisations which adopt practices such as frequent deployments, automated testing and comprehensive monitoring tend to achieve shorter lead times for changes, lower change failure rates and faster recovery from incidents than those that rely on manual, siloed processes (KIM, 2016).

A recurring element in these studies is the role of feedback loops and flow metrics in shaping operational cadence. Humble argues that small, incremental changes that move continuously through the deployment pipeline are easier to diagnose and to roll back than large, infrequent releases, because any problem can be associated with a narrow set of modifications and corrected quickly without destabilising the entire system (HUMBLE, 2010). Kim similarly highlights four key indicators of delivery performance, deployment frequency, lead time for changes, mean time to recover and change failure rate, and treats them as practical measures of how effectively feedback is being captured and used to adjust processes and architectures over time (KIM, 2016).

Taken together, these contributions portray DevOps as more than a set of tools, it is a culture built around automated delivery pipelines, shared responsibility for reliability and explicit feedback mechanisms that determine the tempo at which systems can safely evolve. In this view, governance is enacted through day to day

routines such as how often deployments occur, how incidents are reviewed and how information from monitoring is translated into design changes, rather than only through formal policies or occasional transformation projects (HUMBLE, 2010; KIM, 2016). This combined perspective provides the conceptual background for analysing how technical practices and organisational routines must align when new computing infrastructures are introduced and maintained in production environments.

2.3.B Organisational change and adoption of new practices

In the change management literature, John P. Kotter's work is one of the most influential references for understanding how organisations adopt new ways of working. In his book Leading Change, he argues that most transformation efforts fail because managers underestimate the difficulty of changing habits, structures and culture, and because they treat transformation as a single event instead of a multi stage process. His model presents organisational change as a sequence of phases that must build on one another, rather than as a set of isolated actions, and stresses that skipping stages or declaring victory too early tends to undermine the entire effort (KOTTER, 1996).

Kotter's eight step model begins with creating a sense of urgency around the need for change and forming a guiding coalition that has enough credibility and authority to drive the process. These initial steps are followed by the development of a clear vision and strategy, and by systematic communication of this vision throughout the organisation so that employees understand the direction and rationale of the change. The model then highlights the importance of removing obstacles that block the new behaviours, empowering people to act, and generating short term wins that demonstrate tangible progress and help maintain commitment over time (KOTTER, 1996).

The final stages of the model focus on consolidating gains and anchoring the new approaches in the organisational culture. Kotter emphasises that early successes should be used to drive further change, rather than as signals to stop, and that new practices only become durable when they are linked to visible improvements in performance and are reinforced through recruitment, promotion and everyday management routines. From this perspective, change is considered complete only when the new behaviours and structures have become the normal way of working, and when regression to previous patterns is unlikely because the underlying assumptions and values of the organisation have also evolved (KOTTER, 1996).

This literature therefore frames organisational adoption of new practices as a leadership task that combines symbolic and practical elements, from articulating a compelling vision and building coalitions to adjusting systems and recognising early wins. Kotter's model suggests that technical initiatives, such as the introduction of

new tools or infrastructures, require explicit attention to urgency, coalition building, communication and cultural anchoring if they are to be sustained beyond initial deployment, which provides a structured lens for later analysis of how technology related changes are introduced and maintained in practice (KOTTER, 1996).

2.4 SYNTHESIS OF THE LITERATURE

The literature reviewed in this chapter presents software and systems engineering as an architectural discipline, organised around patterns and services rather than isolated implementations. Fowler's work frames enterprise applications as layered structures whose long term cost and adaptability depend on clear separations of responsibility and well chosen interaction patterns, while Erl extends this reasoning to cloud inspired and service oriented environments in which computing and storage are exposed through explicit contracts and governance models (FOWLER, 2002; ERL, 2013).

Continuous Delivery and DevOps studies add an operational lens, showing how automated pipelines, small batch changes and feedback driven routines enable frequent, reliable evolution of technical platforms, and how these practices correlate with improved deployment speed and stability in high performing organisations (HUMBLE, 2010; KIM, 2016). Kotter's model of organisational change complements these insights by emphasising that durable adoption of new practices requires urgency, coalitions, vision, short term wins and cultural anchoring, rather than technical changes alone (KOTTER, 1996).

Together, these contributions provide a compact framework for the remainder of the thesis: Fowler and Erl for the structural design of a local render infrastructure, Humble and Kim for its deployment and operational cadence, and Kotter for the organisational conditions under which such an infrastructure can be introduced and sustained at W company (FOWLER, 2002; ERL, 2013; HUMBLE, 2010; KIM, 2016; KOTTER, 1996).

3. METHOD: IMPLEMENTING A CLUSTER

This chapter explains how W company moved from the earlier problem, blocked workstations and volatile cloud costs, to a concrete local render cluster based on AWS Thinkbox Deadline. The project is treated as a software intensive initiative that requires aligned choices on process, architecture, distributed systems, testing, and governance, rather than a simple software installation.

The overall approach is structured into four phases, summarised in Figure 4. Phase 1, Diagnostic, Stakeholders, and Requirements, analyses current workflows and cloud usage, identifies the main stakeholders, and consolidates functional and non functional requirements. Phase 2, Architectural Design of the Cluster, defines the system architecture, maps components to network and file sharing resources, and makes explicit the main distributed system patterns and quality attribute trade offs. Phase 3, Topology, Implementation, and Testing, covers the installation and configuration of the cluster, together with basic functional and performance tests and the introduction of simple monitoring and logging mechanisms. Phase 4, Governance, Roles, and Organisational Embedding, specifies roles and operating cadence, usage policies, documentation practices, and feedback loops that support the sustained use of the cluster in everyday work. The remainder of this chapter follows these four phases in turn and explains, for each one, both the concrete activities carried out at W company and the way in which the chosen method draws on the software engineering literature.

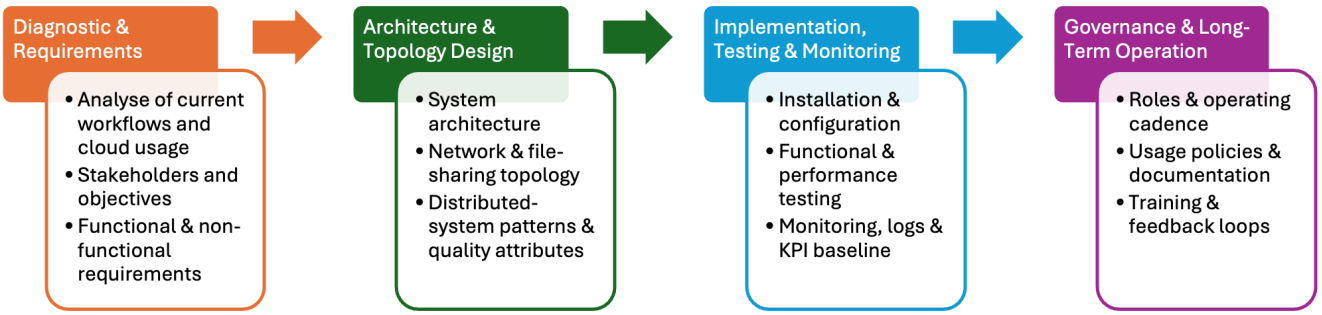

Figure 4: Methodological roadmap for the implementation of the local cluster

3.1 METHODOLOGICAL ROADMAP AND FOUR-PHASE STRUCTURE

The method adopted in this thesis translates the initial problem of expensive and unpredictable cloud rendering into a sequence of concrete, traceable steps that lead to a fully local render cluster at W company. Instead of treating the implementation as a single technical jump, the work is structured into four successive phases that move from understanding the existing situation, to designing an appropriate architecture, to installing and testing the cluster, and finally to embedding it in everyday work through governance and training. These four phases are summarised in Figure 4, which presents the methodological roadmap for the implementation of the local cluster.

The first phase, Diagnostic and Requirements, focuses on understanding how rendering was actually performed before the project and on consolidating the expectations of the people involved. This includes mapping current workflows across the architecture and digital innovation fronts, identifying where and when render jobs accumulate, and documenting the practical problems that arise, such as blocked workstations during core hours or last minute resort to cloud services. On this factual basis, the phase identifies the main stakeholders, such as architects, visualisation specialists and technical support, and captures both functional requirements, for example the need for unattended overnight rendering, and quality attributes, for example predictability of turnaround time and stability of costs. In other words, Phase 1 answers what needs to change, for whom, and why, using the language of requirements and quality attributes introduced earlier in the literature review.

The second phase, Architecture and Topology Design, treats the future cluster as a software intensive system that must be explicitly defined before it is built. Here, the method specifies the system boundary of the render-management solution, the main components on the control plane and on the client side, and the interactions between them. This includes defining how Deadline’s Repository, Database, Remote Connection Server and Worker applications will be arranged, how they depend on the local area network and on shared storage, and how they relate to existing tools such as SketchUp with V-Ray, 3ds Max with Corona and Unreal Engine. At the same time, the phase selects distributed-system patterns and trade offs that are appropriate for a small studio, for example concentrating services on a single host for simplicity, while still enforcing a clear separation between control-plane services and shared file storage. The output is a set of architectural views and a network and storage topology that can be implemented without ambiguity.

The third phase, Implementation, Testing and Monitoring, translates the design into a working cluster and verifies that it behaves as intended. Concretely, this involves installing the chosen operating system on the control-plane server, deploying the Deadline components, configuring shared storage, and enrolling artist workstations as Workers and Monitors. Once the basic installation is complete, a set of functional and performance tests is executed, using representative scenes from W company's pipelines to check that jobs can be submitted, queued and processed correctly, that Path Mapping resolves file paths as expected, and that render times and queue behaviour are coherent with the design assumptions. Simple monitoring and logging mechanisms are also introduced at this stage, so that job status, failures and basic performance indicators can be observed without manual inspection of each node. Together, these activities close the gap between the abstract architecture and a concrete, observable system.

The fourth phase, Governance and Long term Operation, recognises that a render cluster only delivers value if it is used consistently and maintained over time. In this phase the method specifies who operates the system, which roles are responsible for queue supervision, maintenance and support, and how decisions about priorities or configuration changes are made. It also defines simple operating routines, such as end of day reviews of the queue, weekly checks of Worker status and storage usage, and periodic review of job presets. Usage policies and short internal documents are prepared to clarify how artists should submit jobs, where outputs must be stored, and how to escalate issues. Training sessions and feedback loops are included so that early experiences can be incorporated into adjustments of presets, policies or even architecture if necessary. In practice, Phase 4 ensures that the cluster becomes part of the studio's normal production system, instead of remaining a one off technical experiment.

3.2 PROJECT THROUGH THE 5W1H QUESTIONS

This section summarises the project using the classic 5W1H framework. The aim is to clarify, in simple and direct terms, why W company is implementing a local render cluster, who is involved, what is being built, where it operates, when it is deployed, and how success is assessed. This framing provides a concise reference for the remainder of the chapter and ensures that the project is understood consistently by both technical and managerial readers.

Why this project exists is straightforward. W company needs to stop blocking artist workstations during office hours and to reduce the volatility of cloud rendering costs. In the current situation, the same high performance machines are used for modelling and for rendering, so long jobs freeze creative work and often spill into nights and weekends. When the team pushes work to external farms instead, invoices arrive late, are hard to allocate to projects, and are difficult to forecast. This project is

therefore aimed at a concrete goal, to move rendering to a managed, queue based local cluster that runs mostly off hours, keeps artists productive during the day, and replaces unpredictable cloud expenses with a stable in house capacity that can be monitored and tuned.

Who is involved in this project at W company is deliberately limited so that decisions stay clear. At the strategic level, a member of management sponsors the investment in the cluster and links it to client delivery and financial discipline. At the operational level, a project supervisor arbitrates priorities between architecture and digital visualisation work and approves maintenance windows. A technical lead installs, configures and operates the Deadline based render system and becomes the single technical owner. Around this core, a small group of primary users in both divisions prepare scenes, submit jobs through standardised presets, and provide feedback on image quality and turnaround times, so that the project can be adjusted to real production needs rather than abstract assumptions.

The project implements a local, queue based render infrastructure built around AWS Thinkbox Deadline and tied directly into W company's existing production tools. At its core is a Deadline control plane composed of a central Repository that holds configuration, plug ins, scripts and logs, a MongoDB database that tracks jobs and Worker state, and a Remote Connection Server that mediates all client communication over HTTP or HTTPS. Artist workstations run the Deadline Worker service, which executes renders, and the Deadline Monitor, which provides the interface for submitting and supervising jobs. Submissions use native plug ins, such as Submit Max To Deadline for 3ds Max with Corona and the V Ray Standalone plug in for .vrscene exports, and are supported by Path Mapping rules that systematically rewrite file paths so that scenes, textures and outputs are always accessed on shared storage instead of local disks. In parallel with this technical stack, the project defines governance elements such as presets, naming conventions and simple operating routines, so that the cluster is delivered not only as a functioning system but as a managed service embedded in everyday production.

Figure 5 presents how this Deadline based infrastructure is embedded in W company's local network. The existing Windows Server 2019 host remains the central file and network service, exposing the shared drives G, H and P to artist workstations while also acting as firewall, gateway and DHCP server. A separate Ubuntu Server 22.04 machine with IP address 192.168.0.100 runs all Deadline control plane services, combining the Repository and Remote Connection Server, the MongoDB database for job state, and the DeadlineStorage export that workstations mount as drive X. Artist machines reach these resources over the local area network, use SMB to read and write assets and rendered outputs on the shared drives, and contact the Remote Connection Server over HTTPS on port 8080 for job management. Both servers and workstations obtain internet access through the router

and the gateway server, so that the cluster remains fully inside the office network while still being able to connect to external services when necessary.

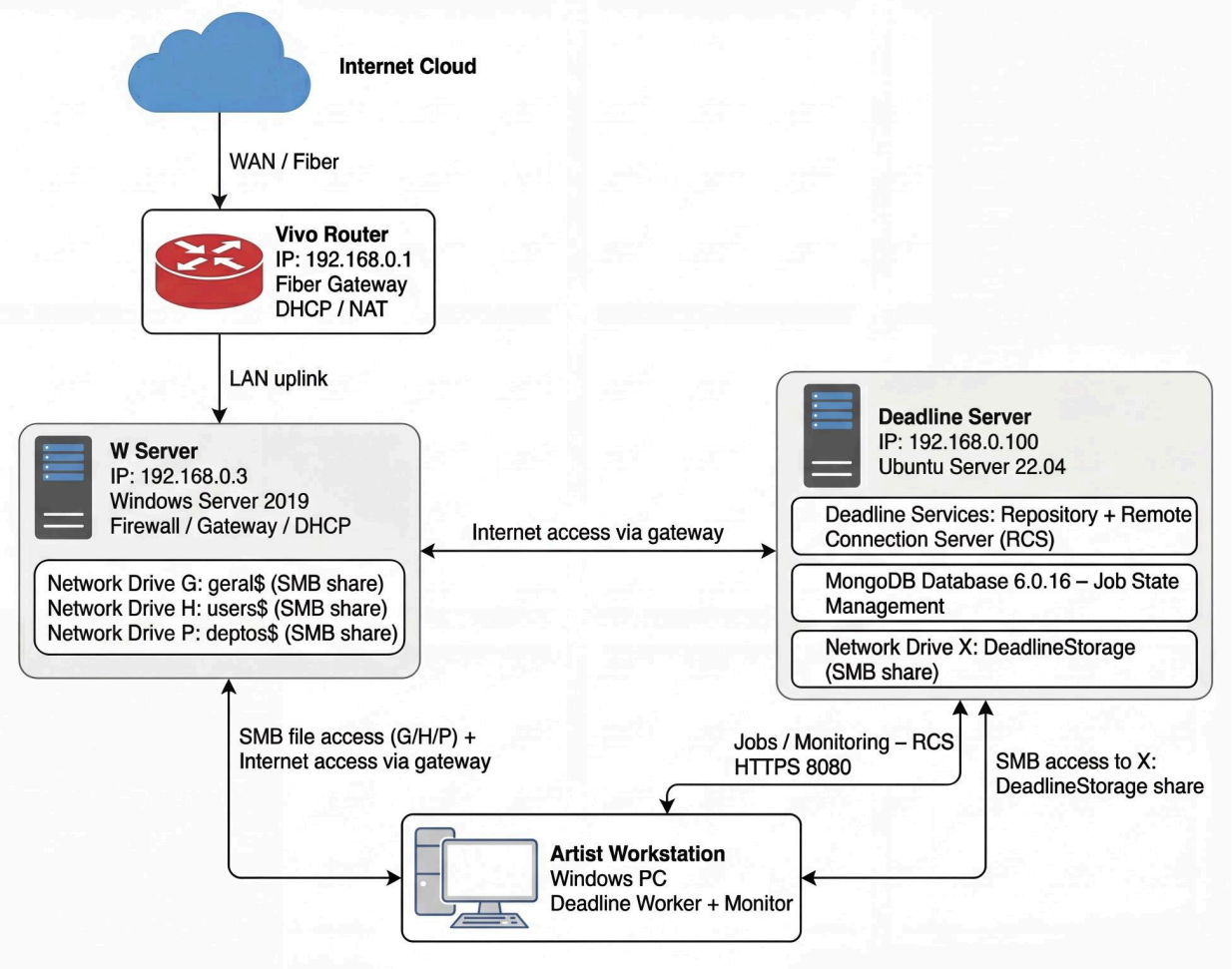

Figure 5: Network and storage topology of the Deadline based render cluster

Where the project is implemented is strictly limited to W company's on premises environment. All core Deadline services run on a dedicated Linux server on the office local area network, and all render jobs read and write data through a small number of defined network shares. The Repository and Database stay on fast local storage inside this server, while project scenes, textures and outputs live on a separate share that every Worker can access with the same path. No external infrastructure is required, and client data never leaves the company's network perimeter. This local scope shapes many design choices, for example the decision to use Deadline's Remote Connection Server to avoid exposing the database directly, and the emphasis on simple, static IP addressing so that a small team can manage the system without specialised network staff.

When the project is executed follows a staged logic aligned with the timeframe of the thesis and the rhythm of the studio. First, a short diagnostic and preparation

period is used to baseline current rendering practices, estimate weekly throughput and off hours usage, and capture the main pain points. Next, the control plane is installed, a small set of Workers is enrolled, and a bounded pilot is run in Deadline License Free Mode, using up to ten nodes without licence cost to verify that the cluster behaves as intended in real projects. During this pilot, the project team already measures simple indicators, weekly completed images, queue wait times, job success ratio, off hours share, and the mix between Worker based and workstation based execution. Only after these measurements show stable improvement does the project move to a broader deployment phase, during which the cluster is used in normal production and acceptance criteria are checked over several weeks instead of in isolated tests.

How success is evaluated and how the project is steered combines practical operations with clear quantitative targets. The cluster is considered successful if it meets a compact set of thresholds that W company can verify directly in Deadline Monitor, for example roughly doubling weekly image throughput compared to the baseline, keeping queue wait time for standard jobs within a few minutes under normal load, executing at least about 80 percent of jobs on Worker nodes rather than on submitter machines, achieving a job success ratio in the low to mid ninety percent range after stabilisation, and shifting about 70 to 80 percent of total render time to outside core office hours. Evidence is collected by exporting Deadline statistics and logs, by running a small acceptance suite of reference scenes for both divisions after each significant change, and by recording configuration updates and incidents in simple internal documents. In this way, the 5W1H framing is not just descriptive, it provides a precise backbone for the project, linking a clear motivation, defined actors, a bounded technical scope, a local deployment context, a realistic calendar, and measurable rules for judging whether the implementation of Deadline at W company has actually delivered the intended benefits.

3.3 CONCLUSION

The architecture defined in this section provides W company with a clear and workable foundation for a local render cluster. Deadline’s three core elements, the Repository for configuration, the Database for job state, and the Remote Connection Server for client access, are grouped on a single controlled server, while all Workers and artist machines interact with this core through shared storage and consistent Path Mapping. This keeps the system simple to operate, predictable in behaviour, and aligned with the constraints of a small studio.By separating configuration from runtime data, enforcing a single storage namespace, and funnelling all control traffic through one gateway, the design remains both robust today and ready for incremental expansion if needed. It turns the motivations of the project, unblocking workstations and stabilising rendering throughput, into a concrete technical baseline.

4. RESULTS

This chapter presents the concrete outcomes of the Deadline implementation at W company. It moves from design intentions and methodological choices to what was actually installed on site, how this installation behaves in day to day work, and which effects can be observed on costs, workstation usage and team practices. For readers who are less interested in technical theory and more concerned with what changed in reality, this chapter can be read as a factual narrative of the project, from the first repurposed server to the current render cluster integrated into the company routine.

The first part of the chapter describes what was physically and logically built. Section 4.1 details the implemented architecture of the local render cluster, starting from the control plane and server host and then showing how existing machines were reused as render nodes and how the cluster fits into the office network and shared storage. Section 4.2 follows the same system during a normal working week, explaining how artists submit jobs through Deadline Monitor, how the queue organises work across Workstations as Workers, and how storage conventions and path handling ensure that files can be found and rendered reliably. Together, these sections answer two basic questions that matter for both managers and engineers, namely what the cluster looks like in production and how it is used in practice by the two main pipelines of W company.

The second part of the chapter focuses on impact, sustainability and learning. Section 4.3 discusses operational results and user experience, combining simple performance indicators with qualitative feedback on how the cluster changed everyday work and on which pain points remain. Section 4.4 describes the documentation, training materials and operating routines that keep the system understandable and reliable despite staff rotation. Section 4.5 then presents the final state of the cluster and its current limitations at the time of writing, including the migration to the new company server. Finally, Section 4.6 summarises the main lessons learned from the project, both technical and organisational, and shows how an initially modest render cluster became the starting point for a broader reflection on server infrastructure at W company.

4.1 IMPLEMENTED ARCHITECTURE

This section presents the actual architecture of the local render cluster that was deployed at W company, focusing on the concrete infrastructure rather than on the idealised options discussed in Chapter 3. It offers a single, coherent “snapshot” of what exists in production: a repurposed desktop computer promoted to the role of central server, the way this host is connected to the existing office network, and the organisation of storage volumes used to hold both configuration data and render files. Building on the design principles, the cluster is deliberately compact, with all core Deadline services hosted on this control-plane machine. The Repository, which stores configuration files, plug ins, scripts and Path Mapping rules, the Database, which records job state and queue information, and the Remote Connection Server, which exposes Deadline to clients over standard network ports, are all co located on this single server. Around this control plane, ordinary workstations and Worker machines connect via the local area network to submit and process jobs, while shared storage volumes provide a common location for scene files, asset libraries and render outputs. The following subsections describe this implemented architecture in more detail, from the characteristics of the server host to its effective network and storage footprint.

4.1.A Delivered control plane and server host

The Deadline control plane at W company is hosted on a repurposed desktop tower that has been reinstalled with Ubuntu Server 22.04 LTS. Using a long term support Linux distribution ensures regular security updates and a stable package base, which is appropriate for a machine that is expected to operate continuously and largely unattended in a small studio. On this single host, all server side Deadline components are co located. The Repository stores configuration files, plug ins, Path Mapping rules, job presets, scripts and log files, in other words everything that defines how submissions are interpreted and how paths are normalised across machines. A MongoDB 6.0.16 instance records the dynamic state of the farm, including the job queue, per job metadata and global settings, while the Remote Connection Server (RCS) exposes Deadline over HTTP and HTTPS so that workstations can communicate through well known ports without directly mounting the Repository file system.

The same server also concentrates the storage functions required by the cluster. Internally it combines a solid state drive (SSD) and a traditional hard disk drive (HDD). The SSD hosts the operating system, the Repository and the MongoDB data directory, so that configuration reads, job lookups and state updates remain responsive even when many Workers are polling the queue. The HDD provides higher capacity magnetic storage for heavy project data and is exported as a Samba

share called DeadlineStorage over SMB on TCP port 445. Through Samba, the Linux server presents this volume as a Windows compatible network drive, so that both artists' workstations and Worker nodes access scene files, shared asset libraries and render outputs using identical paths, which simplifies Path Mapping and reduces relinking errors.

Figure 6 focuses on the physical aspect of this design. It shows the repurposed desktop tower that concentrates all Deadline control plane services and storage volumes in production, making visible the decision to collapse server roles onto a single, clearly identified machine. This visual emphasis on a compact, self contained host underlines the project's objective of minimising hardware costs and operational complexity while still providing a robust foundation for a managed render service.

Figure 6: Repurposed desktop tower hosting all Deadline control-plane services

Figure 7 complements this with a logical view of the same host. The diagram groups the Repository on SSD storage, the MongoDB 6.0.16 database and its state data, and the HDD based DeadlineStorage share under the Samba service, and indicates how the Remote Connection Server and local Deadline Monitor interact with these layers through HTTP or HTTPS for configuration and through SMB for file access. In doing so, it clarifies how configuration, state and project assets are separated yet tightly integrated inside the control plane server, and it makes explicit the points where future evolution, such as moving the database or the file share to dedicated machines, could be introduced without changing the overall architecture.

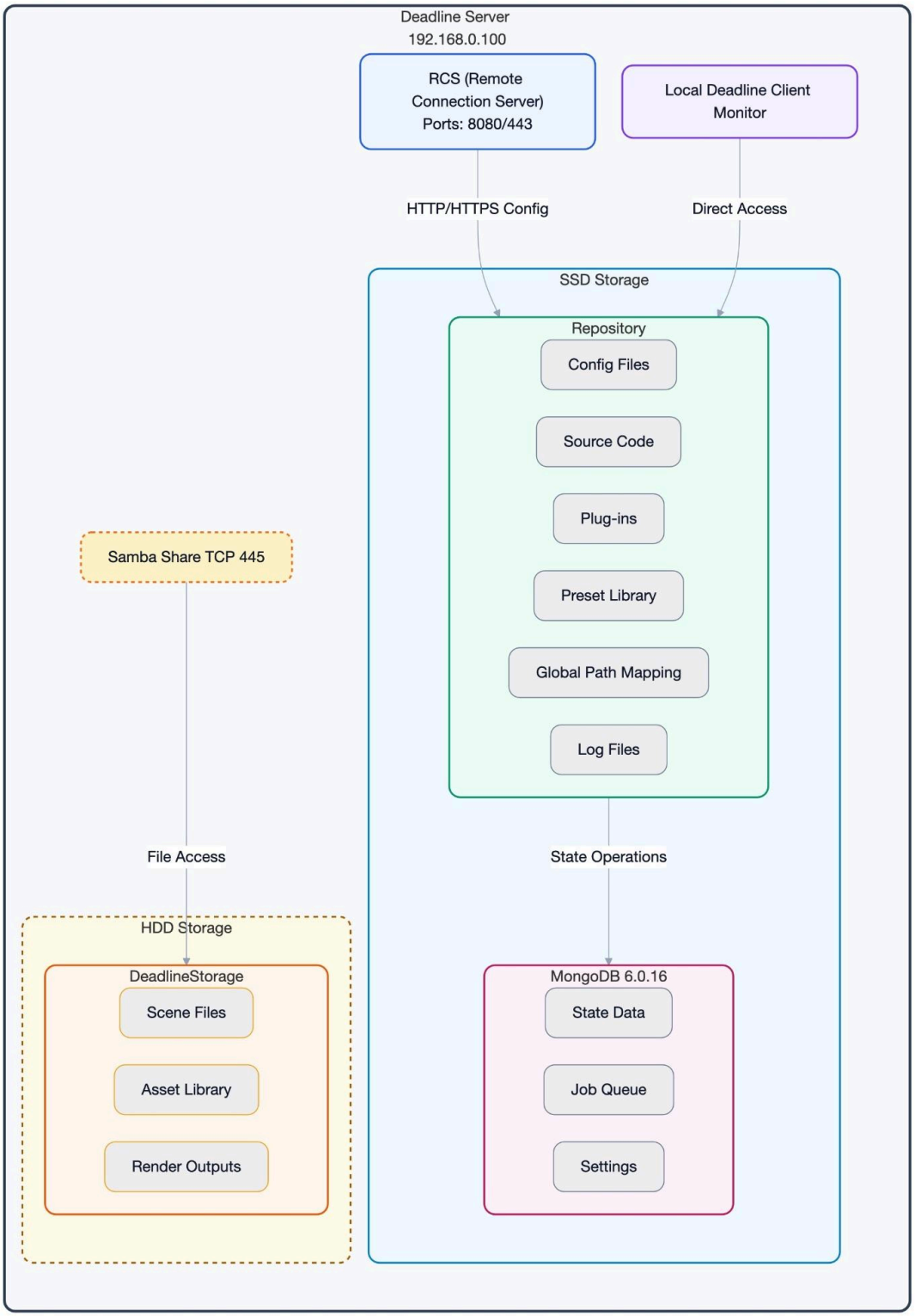

Figure 7: Logical Architecture of the Deadline Control-Plane Server and Storage

4.1.B Network and storage footprint in production

In production, the local render cluster is inserted into the existing office network without changing the global perimeter. Internet traffic still enters through the commercial router and is then passed to the legacy office server, which continues to act as gateway, firewall and DHCP provider. The Deadline control plane is added as a dedicated host at the fixed address 192.168.0.100 on the same local area network, so no new routing rules or external openings are required. A local area network, or LAN, is the internal segment that connects the company’s machines inside the office, typically over Ethernet or Wi-Fi.

Within this layout, all render related control traffic from Monitors and Workers is directed to the single address of the Deadline server. The Remote Connection Server (RCS) listens on TCP port 8080 for HTTP and can be configured to provide encrypted HTTPS on port 443, so users and render nodes always contact one stable endpoint for queue operations. The MongoDB database remains bound to the localhost interface on the Deadline server and is never exposed directly on the network, which is consistent with the hardening principles adopted for this project.

The storage footprint follows a clear separation between control plane data and production payloads. All configuration files, plug ins, scripts and log files reside on the server’s solid state drive inside the Deadline Repository, which is not accessed directly by end users. Project scenes, textures and final frames are stored on a hard disk based network share exported from the same host as the UNC path \\192.168.0.100\DeadlineStorage, so that all machines reference the same location in a uniform way. The DeadlineStorage share is implemented with Samba, the Linux service that speaks the same file sharing protocol as Windows, SMB, over TCP port 445. Workstations map this share as a normal network drive and Workers stream assets and write outputs directly to it, without needing local copies. In daily operation, artists export render ready packages into structured folders on DeadlineStorage and submit jobs that point to these UNC paths; Workers then read the scenes from the share and write finished frames back to the same volume. This arrangement keeps heavy production traffic away from the SSD that hosts the Repository, enforces a single canonical location for all render inputs and outputs, and simplifies troubleshooting, since every job can be traced to a well defined folder tree on a single server.

Figure 8 presents the render-farm network topology at W company. It summarises how the router, the legacy office server and the new Deadline server are connected, and shows how control traffic through the RCS and file traffic through DeadlineStorage fit into the existing office network.

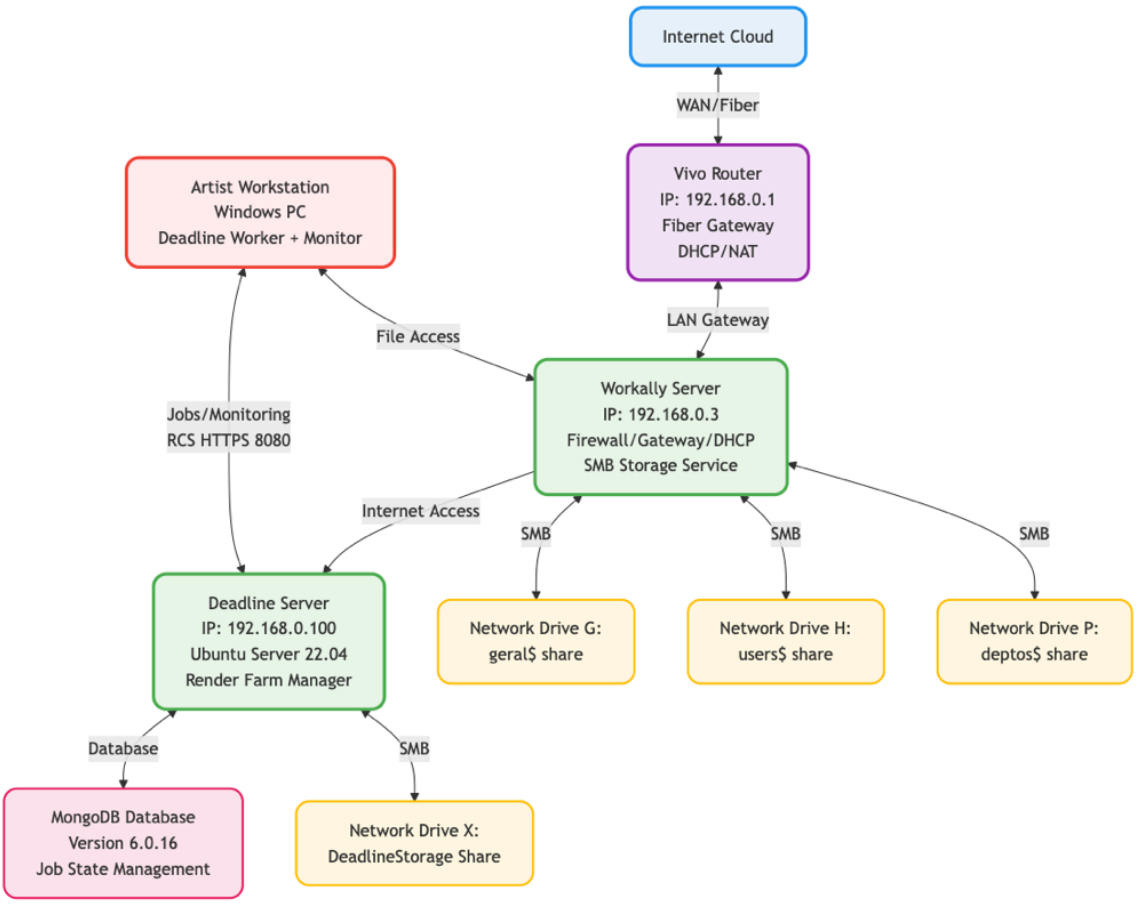

Figure 8: Network and Storage Footprint of the Local Render Cluster in Production

4.1.C Render node population and alignment with the planned topology

Building on the server footprint described in Section 4.1.A and the network layout presented in Section 4.1.B, this final subsection details how W company's existing machines were effectively turned into render nodes and how closely the resulting footprint matches the small on-prem topology. In practice, the render farm is composed of a mixed population of high-specification designer workstations and a small number of secondary desktop machines that are no longer required for day-to-day modelling work but remain fully usable for background computation. Each eligible machine receives the Deadline Launcher and Worker components, which together transform a regular Windows workstation into a node capable of accepting jobs from the central queue. As shown schematically in figure 7, this results in a compact but heterogeneous cluster in which all nodes connect to the same

control-plane host and shared storage, without introducing any additional network tier or specialised appliance beyond the single Deadline server.

From an operational perspective, nodes join and leave the farm according to simple, transparent rules based on off-hours policies and logical groupings. Deadline “pools” and “groups” are used as lightweight classification mechanisms: pools indicate the type of workloads that a node may receive (for instance, architectural stills versus immersive content), while groups reflect hardware and availability characteristics such as GPU-capable machines, always-on secondary desktops, or laptops that should only be used exceptionally. In everyday terms, this means that a designer’s primary machine only accepts jobs from the appropriate pool and only during defined windows, typically evenings and weekends, when the user is logged off and the workstation would otherwise be idle. Secondary machines, by contrast, are configured as permanent nodes, remaining attached to the farm for most of the week and providing a stable baseline of capacity. The resulting configuration ensures that rendering resources are harvested opportunistically, without disrupting interactive work or requiring staff to modify their core tools and routines.

When compared to the target topology, the realised render-node population closely follows the intended pattern of “one modest server plus a small constellation of internal machines”, while documenting a few deliberate compromises. The cluster remains anchored on a single control-plane host, which simplifies administration and cost but concentrates risk on a single physical box; similarly, overall capacity is bounded by the limited number of workstations that can be safely enrolled without jeopardising daytime responsiveness. These constraints were explicitly accepted in the design, since they still satisfy the operational objectives and acceptance criteria, notably the ability to process overnight batches locally and to eliminate routine cloud usage for still-image projects. At the same time, the alignment between logical groupings, shared storage and network reachability creates a clear path to incremental scale-out: additional repurposed desktops can be added to existing pools with minimal configuration effort, and, if future demand justifies it, dedicated render nodes or a second control-plane host can be introduced without revisiting the fundamental architecture established in this chapter.

4.2 OPERATION IN DAILY WORK

4.2.A Submission workflows across the main pipelines

In everyday work, the two main production divisions at W company, the Architecture division and the Digital Visualisation division, interact with the render farm through a small number of standardised submission paths. In both cases, the render cluster is accessed through Deadline Monitor, the graphical interface used to submit and track jobs, and through the dedicated submission plug ins installed on each workstation, so artists do not have to interact with the control plane directly. Scenes are prepared in the familiar authoring tools, then exported together with their associated textures and auxiliary data into project specific folder trees on the shared DeadlineStorage volume, the central network share used for render scenes and outputs. This reliance on common UNC paths, that is, network locations written in the form \server\share, translates the integration objectives into concrete practice: regardless of the pipeline, jobs enter the farm through a predictable combination of local tools, submitter windows and centrally managed paths.

Within the architectural pipeline, designers work primarily in SketchUp with V-Ray as their rendering engine. When a scene is ready for high quality output, it is exported to the shared storage and submitted to Deadline using the dedicated V-Ray submission window instead of manually creating a job in Monitor. This window allows the artist to define the main job parameters and to select which Workstations-as-Workers, that is, workstations that can temporarily act as render nodes, are eligible to execute the render by moving machines between unselected and selected lists. Once the form is confirmed, Deadline returns a concise summary of the operation, including the Repository address, the network path used for assets and the identifier of the created job, confirming that the scene has been accepted into the queue. In practice, architects dispatch renders directly from their usual V-Ray interface while still benefiting from the central queue and from the storage discipline imposed by DeadlineStorage.

The Digital Visualisation division follows a closely related pattern for 3ds Max and Corona workloads, with an additional layer of automation for batch processing. For individual scenes, artists rely on the Submit Max To Deadline plug in, which packages the current .max file and its render settings and forwards them to the central queue using the same shared path conventions as in the architectural pipeline. For larger batches, such as catalogues of views or families of camera angles derived from a common base model, the studio uses custom 3ds Max scripts written in MaxScript, the native scripting language of 3ds Max. These scripts traverse a designated project directory, identify all relevant .max files, perform an automatic search for missing assets across the studio file servers, repair broken file paths when the corresponding

textures or proxies are found, and then submit a separate Corona job for each corrected scene. Building on the same integration mechanisms, this scripted workflow allows dozens of scenes to be cleaned and dispatched to Deadline in a single operation, reducing manual effort, lowering the risk of missing assets at render time and ensuring that even complex visualisation batches enter the cluster through a predictable, repeatable submission process.

Figures 9-10 present this interaction from the point of view of the architectural pipeline. They show the V-Ray job submission window and the subsequent confirmation dialog in Deadline Monitor, highlighting how artists select Workers, define basic parameters and receive a clear acknowledgment that the job has been registered in the central queue.

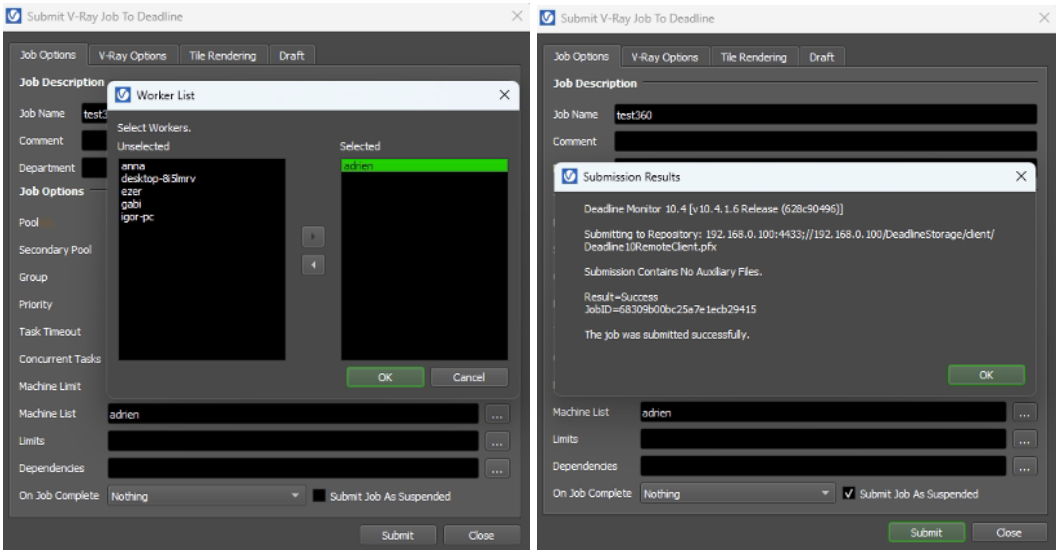

Figure 9-10: V-Ray Job Submission and Confirmation in Deadline Monitor

4.2.B Workstations as Workers, queue discipline, and off-hours utilisation

In the implemented cluster, most of the effective rendering capacity comes from the same Windows workstations that architects and visualisation artists use during the day. Each eligible machine runs the Deadline Launcher and Worker services locally, which together transform an ordinary desktop into a render node capable of polling the central queue and executing tasks without additional user intervention. Pools and groups are used as lightweight classification mechanisms so that the farm encodes who can render what and where. Pools indicate the kind of workloads a node may receive, for instance architectural stills or Corona based visualisation jobs, while groups reflect more stable properties such as hardware configuration or whether a machine is permanently available. Building on the operational objectives, designers' primary machines are configured to act as Workers only during agreed off hours

windows, typically evenings and nights, when interactive modelling and client sessions are no longer taking place. In everyday terms, an architect can submit a series of renders at the end of the day and log off; once the workstation becomes idle, the local Worker service accepts jobs in the relevant pools and processes them overnight, so that results are available the next morning without having blocked daytime work.

This same configuration also supports a simple but effective form of queue discipline, in line with the governance principles. Rather than relying on informal negotiations, a designated Champion or technically inclined Primary User performs a brief review of the queue at the end of each working day using Deadline Monitor. During this review, they verify that urgent deliveries have appropriate priority, that jobs are assigned to suitable pools and groups, and that heavy exploratory renders are not scheduled on machines that must remain responsive early in the following morning. The practical rules agreed with management therefore become encoded in Deadline’s scheduling attributes instead of remaining implicit, which makes the behaviour of the farm more predictable and easier to explain to occasional users. In this way, the governance model proposed in Chapter 3 is translated into a regular, lightweight routine that shapes how the local cluster is actually used in production.

When problems arise, such as failed jobs, missing assets or misconfigured scenes, they are detected and handled through a combination of Deadline’s built in observability and a concise incident logging practice. Error summaries and per job logs in Monitor allow the Champion to identify whether a failure stems from path mapping issues, insufficient resources on a particular node or errors in scene preparation. Recurrent issues are recorded in a simple incident log, which notes the project, the failure mode and the corrective action taken. This log clarifies who is expected to intervene in each type of situation, whether it is a matter for the technical lead, a Champion or the original scene author. Over time, these modest practices of queue review and incident tracking help to maintain the reliability of the cluster without requiring a full time operator, and they anchor Deadline as a managed shared resource rather than a collection of opaque background processes running on individual workstations.

4.3 OPERATIONAL RESULTS AND USER EXPERIENCE.

4.3.A Functional validation and everyday operational behaviour

Before the local render cluster was introduced into routine production work, it underwent a phase of functional validation to ensure that the main architectural elements described in Sections 4.1 behaved as intended under representative conditions. A small but diverse set of test scenes was assembled from the Architecture and Digital Visualisation pipelines, including V-Ray based stills and Corona scenes with typical texture libraries and project structures. These jobs were submitted through the standard workflows detailed in Section 4.2, using Deadline Monitor and the dedicated submitters, and were executed on a subset of Workstations-as-Workers configured to respect off hours usage. The validation focused on basic but critical success criteria: jobs had to appear correctly in the queue, transition to the rendering state without manual intervention, complete without errors at the Deadline level, and produce outputs in the expected folders on the shared DeadlineStorage volume with paths that could be reopened directly in the authoring tools. Logs and error summaries in Monitor were used to confirm that Repository access, Path Mapping and Worker polling behaved consistently across machines. While no formal benchmarking campaign or performance time series were produced, this initial testing showed that the cluster could reliably process real project scenes and that minor configuration issues, such as occasional missing textures or incorrect network paths, could be resolved by tightening folder conventions and Repository settings.

Once these functional checks were in place, the cluster was progressively used on live projects, which provided a clearer picture of its everyday operational behaviour. During the observation period, designers submitted end of day batches that were processed overnight by workstations configured as Workers, with the Worker service started only when interactive work had ceased, so that daytime responsiveness was preserved. The queue absorbed these jobs without requiring constant supervision, and outputs were generally available on DeadlineStorage at the beginning of the following morning, replacing earlier practices in which individuals left renders running manually on their own machines. Importantly, for the categories of work routed through the farm, the studio stopped relying on cloud rendering and handled the relevant workloads entirely on the local cluster, in line with the economic and operational intent. In qualitative terms, the system demonstrated stable behaviour under the studio’s normal load patterns: jobs progressed predictably through the queue, Workstations-as-Workers provided sufficient overnight capacity for the tested projects, and no structural bottleneck or recurrent failure mode emerged during the period considered, even though the results are reported here as operational observations rather than as formal quantitative KPIs.

4.3.B User experience and qualitative alignment with original objectives

From the perspective of designers, Champions and managers, the introduction of Deadline has primarily been experienced as a change in how render time is organised rather than as the arrival of a new, visibly complex system. Once initial habits were in place, architects and visualisation artists reported a clear benefit in being able to continue modelling, preparing presentations or joining client meetings while their renders progress on other machines, instead of having their own workstation locked by a long sequence of frames. Deadline Monitor, which was initially perceived as technical, gradually became a useful reference point to check job status, identify which batches are running overnight and confirm that outputs have been produced on the shared DeadlineStorage volume. For management, the existence of a single queue, rather than a collection of personal rendering practices, provides reassurance that rendering is being handled as a shared service with at least minimal visibility and traceability. Importantly, for the categories of work that have been integrated into the cluster, cloud rendering is no longer used in routine production: jobs that would previously have been offloaded to cloud providers are now dispatched to internal Workers, which reduces budget uncertainty and reinforces the sense that the studio is operating its own controllable capacity rather than renting it per project.

This positive picture is tempered by a non-negligible learning curve, which shapes how far the cluster can advance the objectives. New or occasional users often require guidance to understand the submission windows, select the appropriate pools and groups, and follow the storage conventions associated with DeadlineStorage, and some still need help interpreting job states and error messages in Monitor. Nevertheless, when the operational objectives and acceptance criteria are revisited qualitatively, several appear clearly satisfied: daytime workstation availability has improved for those who use the system, overnight capacity is regularly exploited, and the reliability of the cluster is generally perceived as adequate, with most failures attributable to scene preparation rather than to infrastructure. Other objectives, such as systematic and effortless use of the queue by all potential users, are only partially met and depend on continued training, refinement of documentation and incremental simplification of submission presets and folder templates. Overall, the local render cluster can be considered a meaningful step towards the envisioned balance between cost control, creative freedom and operational predictability, even if its full potential will be realised gradually as practices stabilise and adoption broadens within W company.

4.4 SUPPORTING ARTEFACTS AND CHANGE MANAGEMENT

4.4.A Operator documentation, runbooks, and user guides

Beyond the deployed infrastructure described earlier in Section 4.1, which detailed the implemented architecture and network layout, the project deliberately produced a small but coherent suite of documentation so that the render cluster can be operated by staff who were not involved in its initial implementation. The central artefact is an internal Deadline manual tailored to W company and hosted as a Google Docs file on the studio's shared drive. As shown in Figure 11, this manual is available in both English and Portuguese and is structured into a concise set of sections that mirror the typical lifecycle of a client machine: Project Details, Installation Client, Deadline Overview, V-Ray, Corona and Resources. The Project Details section introduces Deadline in accessible, non specialist terms, presenting it as a render farm manager and outlining its main advantages and limitations for a studio of this size. The Installation Client and Deadline Overview sections then provide step by step guidance for mapping the DeadlineStorage network share, installing the Deadline Client from the shared installers, configuring the Remote Connection Server certificate and launching the Launcher, Monitor and Worker components with the correct Repository settings. The V-Ray and Corona sections walk through standard submission workflows based on scenes exported to shared storage, while the Resources section centralises links to submission plug ins, the client key, installer archives and upstream Thinkbox documentation, so that users can quickly find all necessary material in a single location.

These user facing materials are complemented by more operational documents, grouped informally into an Operator Pack that supports the technical lead or Champion responsible for the cluster. In practice, this pack includes installation and configuration checklists for new client machines, notes on the expected layout of shared folders and UNC paths, and simple runbooks for recurring administrative tasks such as verifying Repository connectivity, checking Worker health through Monitor, and revoking or reissuing the client TLS certificate stored alongside the installers on DeadlineStorage. While lightweight, these documents give concrete form to the implementation steps and operating practices defined during the implementation phase by specifying how to connect to the control plane, how to bring new nodes into the farm in a repeatable way, and how to perform basic incident response when jobs fail for infrastructure related reasons. Most of these artefacts are stored either on the central server, within the Repository tree or in clearly labelled subfolders of DeadlineStorage, or on the company's shared drive next to the manual itself, so that any authorised staff member can access the current version of the guides and runbooks.

Figure 11 presents an overview of this internal Deadline manual as it is delivered to W company. It shows the bilingual structure, the main sections that map to client installation and submission workflows, and the way links to resources and installers are grouped, making visible how the documentation layer supports everyday use of the cluster.

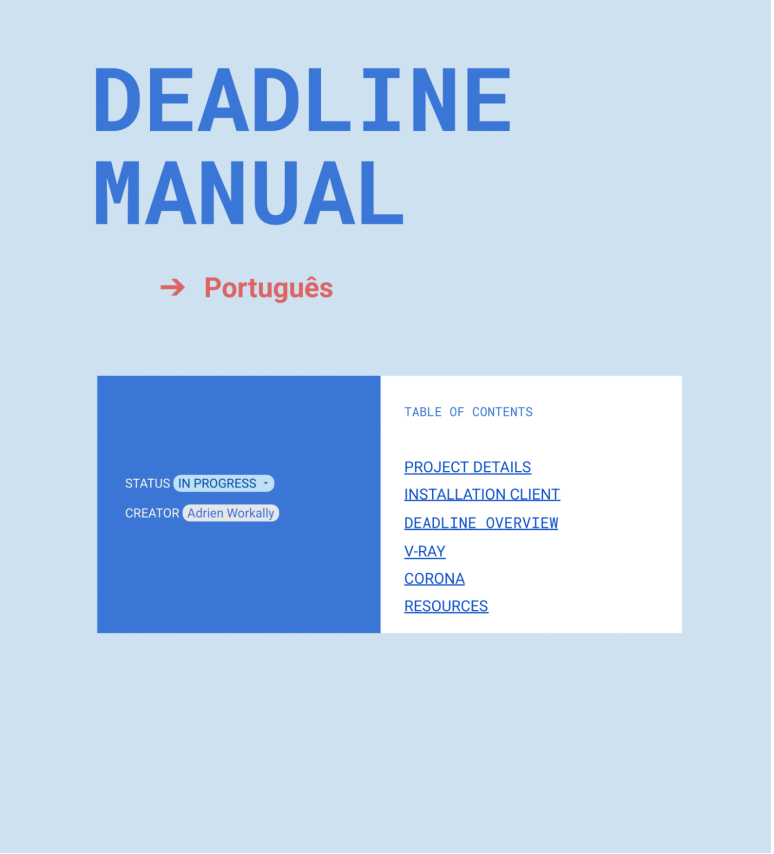

Figure 11: Overview of the Internal Deadline Manual Created for W Company

4.4.B Training, onboarding, and competency development

Training activities played a central role in ensuring that the render cluster could be adopted smoothly by designers and visualisation artists, whose daily routines had previously relied on workstation based or cloud based rendering with little exposure to render farm concepts. The onboarding process began with short demonstration sessions delivered to small groups, during which the overall architecture of the system was introduced in accessible terms and users were shown how to install the Deadline Client, map the shared DeadlineStorage volume and authenticate through the Remote Connection Server. These sessions focused on the concrete actions required during a typical workday, such as launching Deadline Monitor, locating the submission windows within their authoring tools and verifying that outputs appeared

correctly on the shared storage. By presenting the system through familiar workflows rather than abstract diagrams, this initial training tied directly into the operational conventions and prepared the ground for autonomous usage.

Following these introductory demonstrations, the project relied on targeted, hands-on clinics with the staff members identified as local reference points for Deadline. These users, who effectively play the role of Champions in the sense of Phase 4 of the method, acted as intermediaries between the technical lead and the wider team. They received more detailed explanations on topics such as the meaning of pools and groups, the expected behaviour of Workstations as Workers during off hours windows, and the interpretation of job states and error logs within Monitor. Reference scenes were used deliberately during this phase: small, well controlled examples allowed participants to practise the submission workflow end to end without the uncertainty and time pressure associated with full production projects. Each successful submission served both as a learning milestone for the individual and as evidence that the procedures could be executed in practice, thereby spreading operational know-how beyond the original implementer.

As the cluster entered routine use, competency development evolved into an ongoing, distributed process rather than a one-off training event, supported by both documentation and audiovisual material. Designers and visualisation artists continued to consult these local reference users when encountering configuration issues or unfamiliar error messages, and brief explanations were integrated naturally into live project work instead of being confined to formal sessions. In parallel, a short internal training video was recorded, hosted as an unlisted clip and summarised in figure 12, which walks through the key steps of connecting to Deadline, submitting a scene and interpreting basic job status indicators in Monitor. This video, together with the documentation layer described in Section 4.4.A, provided a stable reference point for reinstalling the client, checking Worker settings or following the expected folder structure on DeadlineStorage, thereby reducing the cognitive load on individual experts. Over time, this combination of structured onboarding, practical exercises with reference scenes, informal peer support and a reusable training video contributed to institutionalising a shared understanding of how to submit, monitor and interpret render jobs, and ensured that the cluster could be operated and supported by more than one person within W company.

Figure 12 presents a screenshot from the internal Deadline training video produced for W company. In roughly three minutes, this video walks a new user through the complete basic workflow for SketchUp and V-Ray: exporting a render ready .vrscene file from the SketchUp interface, placing it in the appropriate project folder on shared storage, opening Deadline Monitor, creating a new job that points to this exported file and finally sending the job into the pending queue so that it can be picked up by Workers during off hours. By condensing the full sequence from scene preparation to job submission into a short, visual demonstration, the video provides a concrete, easy to follow reference on how to place a V-Ray render on the Deadline farm.

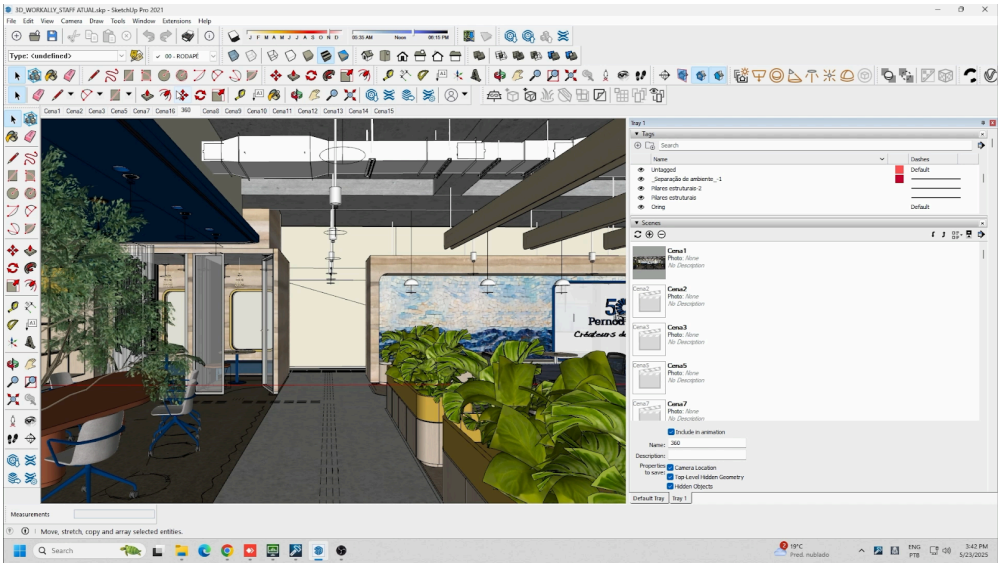

Figure 12: Screenshot from the Internal Deadline Training Video (https://www.youtube.com/watch?v=HyIQOgtLujo)

4.5 FINAL STATE AND CURRENT LIMITATIONS OF THE CLUSTER

4.5.A Consolidated architecture at the time of writing

Several months after the initial deployment of Deadline on the repurposed desktop tower, W company decided to give the render manager a more durable home and, at the same time, to expand its shared storage. To achieve this, a new physical server was specified, assembled and configured by the author of this thesis. The intention was not only to migrate Deadline away from a single workstation class machine, but to build a simple and modern platform that could host virtual machines and other infrastructure services. In parallel, the project aimed to increase internal storage capacity so that W company would rely less on ad hoc disks attached to individual workstations and more on centralised pools that could be managed and backed up consistently.

The new machine runs a TrueNAS based environment that combines ZFS storage with a lightweight virtualisation layer. On the network, the server is attached to the office LAN through a bridge interface. This configuration means that each virtual machine can receive its own IP address on the same subnet as ordinary desktops, which simplifies routing and avoids complex network address translation rules. The physical host is identified as W company server with IP address 192.168.0.5. On top of it, several virtual machines are defined. Some of these are dedicated to infrastructure workloads such as Deadline, while others support more general uses, for example virtual desktop style environments for experimentation and future tools. All of them share the same underlying storage pools, which are also exposed as classic file shares for project data and common resources.

Within this environment, two Linux virtual machines are dedicated to the render cluster. The first VM, at IP address 192.168.0.6, runs Docker, understood here as a lightweight platform for running applications in containers that are isolated from the host system. Inside Docker, a MongoDB container hosts the Deadline database and stores the dynamic state of the farm, including job queues, pools, groups and configuration values that must persist across restarts. The second VM, at IP address 192.168.0.4, runs the main Deadline services. It exposes the Deadline Repository, which is the central file store for plug ins, scripts, Path Mapping definitions and log files, and it also hosts the Remote Connection Server, which is the network entry point for all clients. In daily operation, the services on 192.168.0.4 read and write job state to MongoDB on 192.168.0.6, while the Repository on the same VM provides configuration and plug ins to the Remote Connection Server and to the Workers.

From the perspective of the artists, this internal separation is largely invisible. Their workstations are configured with the Deadline Worker service, the background agent that executes render tasks, and the Deadline Monitor, the graphical interface used to submit and track jobs. These machines do not access the database directly and do not need to know that it runs inside a container. They simply connect to the Remote Connection Server on 192.168.0.4, which authenticates them, exposes the central queue and relays all requests to the services and database running inside the server. This keeps the cluster simple to use, since only a single address needs to be configured on the clients, while the data layer remains confined within the new hardware platform that was designed for storage and virtualisation.

Figure 13 presents this final server side layout of the Deadline control plane at W company. The diagram shows the W company server on IP address 192.168.0.5 with its two Linux virtual machines, one at 192.168.0.6 running Docker and the MongoDB container that stores the Deadline database, and one at 192.168.0.4 running the Deadline Repository and the Remote Connection Server. It also depicts how artist workstations connect only to the Remote Connection Server, which in turn uses the Repository and the MongoDB database to read and write job data, so that the figure visually summarises how the new server, its virtual machines and the desktops together form the current control plane of the render cluster.

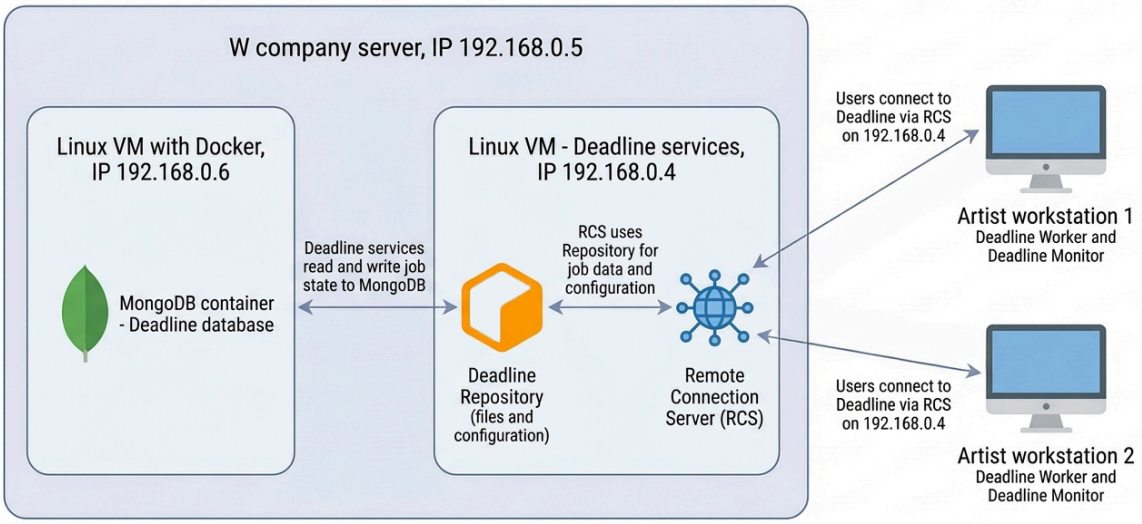

Figure 13: Final server based architecture of the Deadline control plane at W company

4.5.B Operational limitations, adoption and complementary tooling

The most visible limitation of the cluster at the time of writing does not come from hardware or software, but from people. W company has gone through a period of strong staff rotation, so several of the artists and technicians who were originally trained on Deadline and on the underlying render workflows are no longer in the organisation. Newcomers are often young and have little prior experience with render management tools, which makes their first contact with Deadline demanding in terms of concepts and vocabulary. In this context, the internal artefacts created during the project have become essential. The short videos that demonstrate the main workflows and the compact manual that explains how to install the client, connect to the Remote Connection Server and submit jobs now play a central role in onboarding. Without them, knowledge of the cluster would be much more fragile.

Despite this changing team, the technical platform itself is extremely stable. Since the migration to the dedicated server and its two Linux virtual machines, the Deadline services have run with very few interruptions and nightly queues have completed reliably.

In everyday work, the cluster functions as a safety valve for heavy batches of renders. When large volumes of images need to be produced, jobs are sent to the queue and processed overnight or during low activity periods, so that the impact on daytime interactive work remains limited. From the point of view of many artists, Deadline therefore appears as a black box that simply executes jobs when configured correctly. The difficulty is that configuration is not self explanatory. Understanding pools and groups, choosing appropriate priorities and diagnosing errors require a level of familiarity with the Monitor interface that new staff only acquire after several weeks of practice. Even with the manual and videos, users still rely heavily on informal support from colleagues who already know the tool, which concentrates operational knowledge in a small subset of the team and can create bottlenecks when those people are unavailable.

Alongside the main cluster, the author of this thesis also developed a smaller, independent application focused on V-Ray workloads. This auxiliary tool runs locally on a workstation, watches a predefined folder for V-Ray scene files and schedules them to be rendered during the night. Its scope is narrow, but its interface is intentionally simple and more modern than the standard Deadline Monitor, which makes it approachable for users who are reluctant to interact with a full render farm manager. Implemented in Python, it does not pursue aggressive optimisation or complex automation, yet in practice it has proved stable and has become a practical way to automate a subset of repetitive renders on a single machine.

As a result, W company now operates two complementary solutions for handling intensive rendering periods. The Deadline cluster on the server provides a powerful, highly reliable queue based system that distributes work across multiple machines, but its adoption depends on users investing time to understand its concepts and interface. The auxiliary V-Ray application is lighter, less general and only suitable for specific jobs, yet it offers an easier entry point to automation for artists who are still learning the main tool. The main limitation of the current situation is therefore not the absence of technical capability, but the challenge of embedding these solutions into a team that continues to change, which reinforces the importance of concise documentation, clear examples and incremental learning paths for new employees.

4.6 LESSONS LEARNED FROM THE IMPLEMENTATION PROJECT

This project first taught me how to design and operate servers as shared production resources for a team, rather than simply installing software on an individual machine. At the beginning I had almost no experience with networking or system administration, these were not my main areas of training. Starting from a single recycled office computer, I had to understand what it means to turn a workstation into a stable service unit, assign it a clear role in the internal network, keep it available continuously and ensure that other people can rely on it. This shifted my perspective from isolated applications to an integrated system, with interconnected machines, shared storage and services that must deliver predictable performance over time.

In parallel, the project gave me a concrete understanding of a render farm as a production system, and more broadly of what a shared technical service represents in a small organisation. Previously, rendering was a local operation, triggered from a 3D tool on the artist's own workstation. With Deadline and the cluster, I learned to think in terms of coordination and flow: a central node that plans work, resources that execute tasks, and information that must remain consistent and accessible across all points in the process. Delegating renders to a central system changes the production model of the studio, freeing workstations for interactive work and turning rendering into a background process rather than a bottleneck. It also made clear that technical design alone is not sufficient. The architecture has to match working practices, naming conventions, file structures and the way projects are prepared if it is to function as a real production system.

A significant part of the learning concerned the link between the technical system and the human system. The cluster creates value only if people understand what it provides and are able to use it correctly. Staff turnover at W company made this very visible. Several of the people trained at the beginning left, new colleagues arrived with no prior exposure to render management, and the system had to remain usable despite these shifts. To address this, I produced a concise manual, recorded short training videos and organised regular discussions to capture the problems that users encountered and to explain the most common error patterns. Acting as the internal reference person for Deadline required me to clarify my own reasoning, to translate relatively complex mechanisms into simple explanations, and to accept that project success is measured as much in stable adoption and routine use as in configuration details on the server.

The intensive use of artificial intelligence tools also changed the way I learned and worked throughout this project. As an engineer I already had a solid technical

background, but I did not master corporate networks, file servers or virtualisation environments. AI acted as a continuous technical coach. Instead of spending hours searching fragmented documentation, I could ask targeted questions, receive explanations in clear language and immediately test the proposed solution on W company's infrastructure. Work progressed through a series of small, controlled experiments with rapid feedback, rather than through long blocks of abstract study. This made learning less tedious and more operational, and allowed me to focus on higher value topics such as user experience, process clarity for artists and the long term sustainability of the system in a changing team.

In the end, this project moved me from someone who knew almost nothing about networks and servers to someone capable of designing and evolving an infrastructure that is actually used in production. Building on this first experience with the render cluster, I was able to design a more comprehensive server for W company, which now sits at the centre of internal storage and several core applications. This strengthened my overall understanding of how modern information systems operate inside an organisation, from physical resources to day to day user practices, and showed me how to combine new tools such as AI with human constraints and technical requirements to build a socio technical system that is both robust and understandable.

CONCLUSION

This work has examined the implementation of a local, Deadline based render cluster at W company, a Brazilian organisation operating at the intersection of architectural practice and digital visualisation. Starting from the production and cost challenges introduced in the first chapter, the thesis treated rendering as a software intensive process rather than as an isolated graphical task. The initial problem was characterised by a strong dependence on external cloud render farms, variable time indexed costs and a structural tension between interactive use of workstations during the day and heavy rendering workloads. In response, the project defined the objective of designing and deploying a fully local, queue based cluster capable of absorbing routine rendering demand on premises and turning rendering into a predictable, managed service aligned with the company's constraints and ways of working.

To support this objective, the thesis assembled a focused theoretical framework drawing on software engineering, software architecture, distributed systems, testing and organisational change. Concepts such as requirements and quality attributes, architectural views and patterns, distributed coordination, staged testing and monitoring, and governance and operating cadence were used to structure both the analysis and the proposed solution. These elements were then integrated into a four phase method that guided the implementation at W company: from diagnostic and requirements, through architectural design, network and storage topology and validation, to governance, roles and embedding in everyday practice. The method is explicit enough to be followed step by step, yet simple enough to be adapted by a small studio with limited specialised IT capacity.

Applying this method on the existing infrastructure led to the deployment of a control plane on a repurposed server and the enrolment of workstations and secondary machines as Workers executing jobs against shared storage. Submission workflows in the main production pipelines were restructured around Deadline as a central queue, so that designers and visualisation artists now send scenes to a common cluster rather than rendering locally or outsourcing by default to the cloud. Qualitative observations collected during the project indicate that, for the integrated workflows, daytime workstation availability has improved, overnight capacity is regularly exploited and recourse to cloud rendering for routine still images has largely been eliminated. Beyond the technical artefacts, the project also contributed to defining roles, routines and lightweight operating practices, together with internal documentation and training materials, so that the cluster is treated as a shared service that can be understood, operated and maintained by more than one individual.

At the same time, the analysis of limitations shows that the implemented solution remains a first iteration rather than a definitive end state. Organisationally, the studio still depends strongly on a small group of Champions to concentrate operational competence, in a labour market where staff turnover can be abrupt and formal handovers are not guaranteed. Without a more structured training pipeline, knowledge about pools and groups, Worker behaviour, troubleshooting procedures and expected folder structures risks eroding over time. Technically, the choice to concentrate all control plane services and shared storage on a single, ageing server keeps the solution economical and simple, but introduces a clear single point of failure and leaves the system exposed to local power cuts and hardware faults. Reasonable future improvements include appointing and training multiple Champions, defining a standardised onboarding path for new staff, scheduling periodic refresh sessions, adding an uninterruptible power supply and basic hardware health monitoring, and, in a later phase, gradually separating roles across two hosts or adding dedicated render nodes.

Beyond its immediate impact at W company, the thesis suggests a way of treating render management in small and medium sized studios as a software engineering and production engineering problem. By combining established concepts with a concrete, resource constrained implementation, it offers a method that can be reused or adapted by other organisations wishing to internalise render capacity using existing machines rather than relying systematically on the cloud. Possible extensions include developing more advanced monitoring and logging, exploring controlled hybrid configurations that occasionally burst to external capacity under clear rules, refining scheduling and quality presets based on observed usage and documenting comparative case studies across studios. In summary, the project has shown that it is possible, within the constraints of a small creative team, to design and implement a local render cluster that replaces routine cloud usage, improves control over rendering workflows and initiates the institutionalisation of rendering as a managed, observable and standardised service.

REFERÊNCIAS BIBLIOGRÁFICAS

FOWLER, M. Patterns of Enterprise Application Architecture. Boston, MA: Addison-Wesley Longman, 2002.

ERL, T. Cloud Computing: Concepts, Technology & Architecture. Harlow: Pearson Education, 2013.

HUMBLE, J.; FARLEY, D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston, MA: Addison-Wesley, 2010.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. Portland, OR: IT Revolution Press, 2016.

KOTTER, J. P. Leading Change. Boston, MA: Harvard Business School Press, 1996.