



MLOps On-prem Without Kubernetes

A Faster Path to AI Inference in Production

White Paper within the project Data-Driven Organisations / SAIL
Version: 1.0 | Date: 2025-12-16 | Author: Aixia AB and Region Halland



Aixia AB | +46 31 762 02 40



Region Halland | +46 35 13 48 00

Table of contents

1. Executive Summary	3
2. Background	3
3. The Challenge: From Experiment to Production	4
4. Kubernetes: Powerful But Complex.....	4
5. A Simplified MLOps Strategy for Small Teams	5
6. Case study: Region Halland and DDO Project.....	7
6.1. Server Architecture	7
6.2. Core Components	7
6.3. Workflow: From Training to Production	7
6.4. Early Lessons from the Implementation	8
6.5. Current Focus and Future Direction.....	9
7. When is it time for Kubernetes?.....	9
8. References	11
Appendix A – The project “Data-driven organisations – Best practices for operationalisation of AI in Sweden	12
SAIL (Sustainable AI Infrastructure Lifecycle)	13
Appendix B – Glossary.....	14



1. Executive Summary

This white paper presents a pragmatic approach for organisations seeking to implement MLOps on-premises without the complexity of Kubernetes. The strategy begins with Docker-based containerisation, version control, and basic automation to enable a fast path to production. By adhering to the Twelve-Factor App principles, the architecture remains future-proof and can seamlessly migrate to Kubernetes as organisational needs evolve.

Our collaboration with Region Halland illustrates the challenges many organisations face when adopting AI. Together, we are developing an MLOps foundation designed to support multiple machine learning initiatives within the organisation. The project demonstrates that an effective MLOps stack for smaller-scale deployments can be established with relatively simple means.

The expected outcome is faster time-to-value, lower barriers to AI deployment, and preserved scalability for future infrastructure growth as organisational demands increase.

2. Background

Many organisations are drawn to the transformative potential of Artificial Intelligence (AI) and Machine Learning (ML) yet moving from a successful lab experiment to a deployed model in production often proves challenging. At Aixia AB, we support both large and small organisations on their AI journey. A recurring pattern we observe is a strong desire to adopt AI, while a lack of experience and resources often hinders progress.

Large technology companies such as Google and Amazon maintain full-time engineering teams and advanced platforms dedicated to MLOps. If smaller organisations attempt to replicate these complex environments – by, for example, adopting Kubernetes-based container orchestration systems – they may discover that they lack the time and expertise required to operationalise them effectively.

Region Halland, a Swedish regional authority with healthcare as its primary responsibility, is one such organisation. They have several promising ML projects but currently lack the resources to build and maintain a full-scale Kubernetes infrastructure. Through the DDO project, Aixia and Region Halland are collaborating to address this challenge. Their situation is far from unique and serves as the basis for the practical recommendations in this white paper.

MLOps aims to accelerate the path from model development to production by introducing methods from traditional DevOps into the ML domain: version control, continuous integration, continuous delivery (CI/CD), continuous training (CT) of models, monitoring, and rapid iteration [1]. However, implementing MLOps does not mean adopting every possible tool and process from day one. The key for smaller ML teams – such as the one Region Halland has – is to identify which components deliver the most value early on, and which can wait. As we will demonstrate, it is possible to realize most of the benefits of MLOps with a limited initial investment by starting simple.

This white paper proposes an alternative path for organisations seeking to deploy AI in on-premises or hybrid environments: begin with a lightweight MLOps setup without Kubernetes. By keeping the solution simple and flexible at the outset – and introducing complexity and automation only as needs evolve – organisations can move from AI research to stable production deployment in a significantly shorter period.

We have evaluated and analysed how Region Halland can shorten its time-to-production for AI solutions. The same principles can be applied to other organisations with similar conditions. Using



modern tools such as Docker containerisation and sound architectural practices ensures that a future transition to Kubernetes or more automated operations will be smooth when the organisation is ready.

This white paper is intended for IT leaders and technical practitioners in organisations with limited resources, demonstrating how robust MLOps can be achieved without unnecessary overhead.

3. The Challenge: From Experiment to Production

Moving an ML model from a data scientist's experimental environment to a reliable production system presents several challenges. Region Halland is an organisation at the forefront of innovation and thus have several research projects in pilot phase. Models are often developed in sandboxed environments – such as Jupyter notebooks or custom-built setups – focused primarily on maximizing accuracy. However, when the same model is to be integrated into the production environment, a range of practical considerations must be addressed: application environments may need to be made reproducible, data pipelines automated, models exposed through APIs or services, and real-time performance and error handling ensured [2].

Without a clear deployment plan, many ML projects around the world become trapped in what is commonly referred to as “Proof-of-Concept limbo” – the model works perfectly in the lab but never reaches production. Gartner research indicates that only 48% of AI projects make it from prototype to production [3].

Most organisations – including Region Halland – lack the luxury of large, specialised MLOps teams. Instead, they operate with limited expertise, budgets, and data volumes compared to larger enterprises. This means they must be deliberate in how they invest their time and resources. Attempting to build a fully featured MLOps infrastructure from the start – complete with scalable cloud services, Kubernetes clusters, microservice architectures, and full CI/CD automation – can easily exceed the capabilities of a small team. Excessive complexity too early often delays progress and inflates both development time and operational costs without bringing the solution closer to production readiness.

Region Halland and many other organisations share a strong incentive to get ML solutions into production faster. In smaller organisations, a compact ML project that moves from idea to value creation in just a few months can make a significant impact. This could mean automating an internal process, enhancing user experience, or enabling better decision-making through predictive models. Having to wait a year or more not only deprives the organisation of these benefits but also risks the model becoming obsolete before deployment, rendering much of the prior effort a waste of resources.

4. Kubernetes: Powerful But Complex

Kubernetes has become the de facto standard for orchestrating containerized applications [4]. The platform is designed to manage thousands of containers across multiple nodes, offering advanced capabilities such as auto-scaling, load balancing, and self-healing. There is no doubt that these features are invaluable for large-scale systems with high traffic and dynamic workloads.

However, our experience at Aixia shows that for smaller applications, Kubernetes often introduces more burden than benefit. The learning curve and operational complexity are steep – it requires deep technical understanding to set up, maintain, and troubleshoot effectively. Running a Kubernetes cluster also entails significant resource and maintenance costs, which can be difficult to justify at smaller scales. A production-grade setup typically requires multiple nodes for redundancy and a full ecosystem of supporting components to remain operational. For Region Halland and other



organisations beginning to implement AI with only a handful of ML models, this can result in a disproportionate share of server resources being consumed just to keep the infrastructure running, rather than executing the actual ML workloads.

In addition to the resource overhead, there is also an administrative burden. With limited DevOps capacity, the additional abstraction layer that Kubernetes introduces can quickly become a challenge. Teams must continuously monitor and manage the Kubernetes environment – time that could otherwise be spent developing and improving ML functionality.

This does not mean that Kubernetes lacks value; rather, organisations should weigh the administrative cost against the operational benefit. For teams like Region Halland ones that are managing only a few ML services, with moderate and predictable workloads, there are often simpler and more efficient paths to production. Before investing in Kubernetes, it is worth asking: Do we really need all these advanced capabilities right now? If the answer is no, this may be a good reason to start with a lighter, more pragmatic solution.

The following section outlines how such an approach can be structured and why it often delivers faster, more sustainable results.

5. A Simplified MLOps Strategy for Small Teams

Rather than immediately adopting a full-scale, Kubernetes-based MLOps environment, small teams can benefit significantly from applying “just enough MLOps” – a minimal set of processes and tools that delivers maximum early impact. The idea is to build a Minimum Viable Pipeline – an MLOps equivalent of the Minimum Viable Product concept [5]. This means implementing only the essential components required to move from experimentation to production, and nothing more. With such a strategy, organisations can achieve much of the value of MLOps with a fraction of the effort.

By starting with a minimal yet modern pipeline, leveraging open-source tools, automating the right steps, and designing for transparency and observability, small teams can bring a functional ML system into production quickly. In fact, research on software project teams suggests that small teams often outperform large ones – completing comparable projects at a fraction of the cost with only marginal schedule differences [6]. Agility and flexibility are the strengths of smaller teams – and they should be used to their advantage.

A key element of our recommended approach is that even though Kubernetes is not adopted initially, the solution should still be architected for future compatibility. This ensures that when the organisation’s needs evolve, migration to Kubernetes or other orchestration frameworks can be done smoothly and incrementally.

Below are several architectural considerations to prepare for a potential future transition to Kubernetes:

Consistent Use of Containers

Containerisation with Docker is arguably the single most important foundation for future portability [7]. Kubernetes is designed to operate with containers, meaning that the same Docker images can often be reused directly during a migration – the only change lies in how they are orchestrated. By containerising from the start, organisations achieve independence from specific machines or operating systems, reducing the risk of unexpected issues when moving environments.



Microservices and Twelve-Factor Principles

Follow the best practices of “12-Factor Apps” [8] when building services. This methodology outlines twelve principles for creating modern, scalable, and maintainable applications. Some particularly relevant aspects for MLOps include:

- Keep application configuration – such as API keys and environment-specific settings – outside the source code, using environment variables or configuration files instead.
- Avoid storing critical data solely within a service’s memory or local filesystem. Use external data stores, such as databases or object storage.
- Design services to be stateless and replaceable, as Kubernetes assumes this behaviour by default. Applications adhering to these principles will integrate more smoothly in a clustered environment.

A microservices mindset, where each service has a clearly defined responsibility and communicates through well-specified interfaces, makes it easier to distribute workloads across multiple nodes when the need arises.

Standardized Logging and Monitoring

Adopt standardized log formats and expose metrics through accessible endpoints. This allows existing tools such as Prometheus to be integrated directly in the event of a Kubernetes migration. Also include consistent health-check endpoints – Kubernetes relies on these to determine whether a pod is healthy. By following such standards from the outset, future rewrites of logging and metrics collection pipelines can be avoided.

Documentation and Configuration as Code

Store all infrastructure-related configuration as code or scripts. This may include docker-compose files for container orchestration or bash scripts for environment setup. This Infrastructure as Code mindset simplifies restoration, modification, and replication of on-prem environments while providing a solid foundation for translation into Kubernetes manifests.

Kubernetes operates on a declarative model, where the desired state of the system is described – rather than issuing imperative commands. YAML files define how applications, networks, and resources should look, and Kubernetes continuously works to enforce and maintain that state. This contrasts with traditional scripting approaches, where each action is manually orchestrated. The declarative model enables self-healing systems, simpler version control of infrastructure, and predictable environment replication.

By structuring on-prem infrastructure according to similar principles – where configuration files define the entire environment – organisations prepare for a smoother transition in the future. Tools exist that can even convert Docker Compose files into Kubernetes YAML files with reasonable accuracy. While the output is rarely perfect, it provides a strong starting point.

In summary, by adhering to these architectural principles, organisations ensure that the path to Kubernetes remains open. This approach delivers the best of both worlds: speed and simplicity today, without compromising future scalability.



6. Case study: Region Halland and DDO Project

To demonstrate how these principles can work in practice, we present Region Halland's ongoing implementation of an MLOps pipeline for healthcare AI. This case study shows how an ML workflow using Docker, MLflow, and incremental automation can be implemented without the overhead of Kubernetes.

6.1. Server Architecture

The pipeline is deployed across four servers, each with a specific role:

Server	Role	Key Components
Training Server	Model development and fine-tuning	Azure DevOps Agent, Training Scripts
MLflow Server	Experiment tracking and model registry	MLflow, PostgreSQL, MinIO, Prometheus, Grafana
Staging Server	Pre-production testing	Triton, Node Exporter
Production Server	Live model serving	Triton, Node Exporter

All servers communicate over a dedicated VLAN, ensuring network isolation and security for healthcare data.

6.2. Core Components

MLflow is the central hub for experiment tracking and model versioning [9]. Training runs are logged with parameters, metrics, and artifacts. The model registry maintains versions with aliases – challenger for staging candidates and champion for production-approved models. MLflow is backed by PostgreSQL for metadata and MinIO for artifact storage, all running as Docker containers.

NVIDIA Triton Inference Server handles high-performance model serving. For teams still learning the workflow, a simpler **FastAPI**-based serving option is also available.

Model Synchronisation is handled by a dedicated service that polls MLflow for models tagged with the appropriate alias (challenger or champion) and automatically deploys them to the Triton model repository.

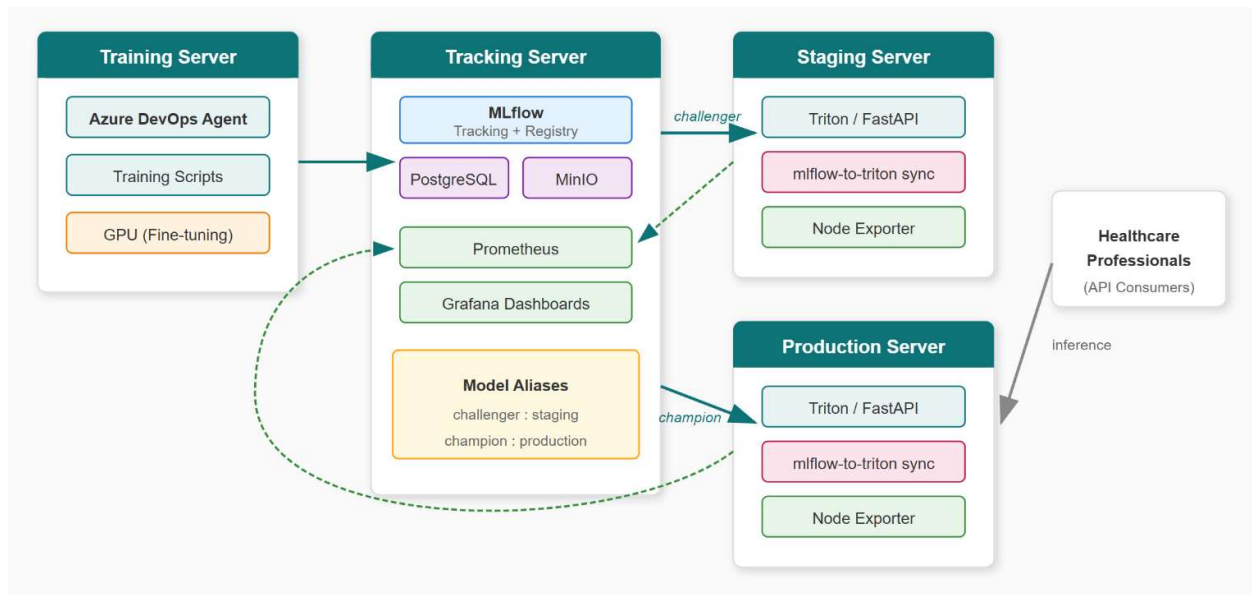
Monitoring is achieved through Prometheus collecting metrics from both Triton and Node Exporter on each serving node. Grafana provides visualisation dashboards for inference latency, throughput, and system health.

6.3. Workflow: From Training to Production

The pipeline follows a clear progression:



1. **Training:** Data scientists run training jobs on the GPU server. The training script logs experiments to MLflow, including model artifacts, hyperparameters, and metrics.
2. **Registration:** Upon successful training, the model is registered in MLflow's model registry. The training script automatically applies the challenger alias to new versions.
3. **Staging Deployment:** The staging server's sync service detects the new challenger model and automatically downloads and deploys it to Triton. Teams can validate model behaviour through the inference API.
4. **Promotion to Production:** After validation, an ML engineer promotes the model by applying the champion alias in MLflow. The production server's sync service then deploys the approved model.
5. **Monitoring:** Prometheus continuously scrapes metrics from both Triton and the underlying infrastructure. Grafana dashboards enable teams to monitor inference performance and detect anomalies.



6.4. Early Lessons from the Implementation

Although the workflow has only recently been established, several clear advantages of the chosen approach have already emerged:

Faster time-to-value: The threshold to get started was significantly lower compared to a Kubernetes-based approach. Several of Region Halland's team members were already familiar with Docker but lacked Kubernetes expertise. By leveraging existing skills and providing both simple (FastAPI) and advanced (Triton) deployment options, team members could quickly begin experimenting – training models and observing their behaviour during inference – without first having to master complex orchestration tooling.

Maintained data security: All sensitive information remains on-premises, which is an absolute requirement for handling healthcare data under Swedish regulations.

Gradual complexity: The dual-path approach (FastAPI for learning, Triton for scale) allows teams to start simple and add complexity only when justified by actual requirements.

Future readiness: The consistent use of Docker containers, externalized configuration, and standardized APIs ensures that a future migration to Kubernetes can be performed smoothly. The same Docker images can be reused, only the orchestration layer changes.

Operational visibility: Integrated monitoring with Prometheus and Grafana provides the observability needed for production operations, enabling proactive identification of performance issues.

6.5. Current Focus and Future Direction

At this stage of the project, the primary purpose is for Region Halland's staff to practice and become familiar with working with MLflow, Triton and other ML tools in a realistic but low-risk environment. We intentionally lean towards more manual steps at this point – for example, manually triggering training runs and explicitly promoting models between staging and production. This deliberate approach ensures that the team gains hands-on understanding of each component before automation abstracts away the underlying mechanics.

As the team's confidence grows and operational patterns become clearer, the plan is to incrementally introduce additional elements: automated CI/CD pipelines triggered by code commits, automated model validation gates, redundancy for critical services, and horizontal scaling of inference servers during peak demand. The guiding principle remains the same – add complexity only when the need is demonstrated and the team is ready to maintain it.

7. When is it time for Kubernetes?

So far, we have argued for starting without Kubernetes, but there may come a point when adopting Kubernetes – or a similar large-scale orchestration framework – becomes a natural next step. Below are several indicators that an organisation is approaching that stage:

Multiple services and complex dependencies

As the ML solution grows and the number of containers increases, manual management can become more challenging. Separate teams may begin handling different parts of the system – data collection, feature engineering, inference, and dashboards – adding to the overall complexity. Kubernetes excels at managing large numbers of services and providing unified control. When it becomes difficult to maintain an overview of dependencies or orchestrate components efficiently, Kubernetes' declarative configuration can bring much-needed structure.

Need for automatic scaling and high availability

When workloads fluctuate significantly or downtime would have serious consequences, Kubernetes' built-in features can be highly beneficial. Auto-scaling allows new pods to be spun up during peak demand, while the cluster's self-healing capabilities automatically replace failed instances. Lightweight orchestration tools such as Docker Swarm can be worth exploring as an early-stage alternative. However, for large-scale setups with strict automation and availability requirements, Kubernetes remains the superior choice.



Expanded team and DevOps capacity

As the organisation grows and gains access to dedicated DevOps or platform specialists, the likelihood of a successful Kubernetes adoption increases. Small teams without such expertise often struggle to both develop ML models and operate a complex platform. With more hands available, responsibilities can be divided – perhaps through a dedicated platform team or a centralized IT operations function. At that point, migrating to Kubernetes can enable greater standardisation and consolidation, serving as a common platform not only for ML but for a wide range of organisational applications – creating operational synergies.

Need for advanced MLOps frameworks

Some powerful MLOps frameworks are designed specifically to run on Kubernetes. That said, it is important to note that most functionality can be achieved using Kubernetes-independent tools. Frameworks such as MLflow, Apache Airflow, and DVC work perfectly well without Kubernetes. Only when the need arises to orchestrate hundreds of parallel experiments or deploy dozens of model APIs with advanced release strategies do Kubernetes-specific tools become truly relevant. This represents a high level of maturity – one that many smaller organisations may never need to reach, opting for simpler, more lightweight solutions.

If one or more of the above criteria are met, it may be time to start planning for a transition. By following the recommendations outlined in this white paper, much of the groundwork will already be in place: applications are containerized, stateless, and built on open standards, and the infrastructure is defined as code. This enables a gradual migration, starting with less critical components and operating in a hybrid mode for a period.

In summary, Kubernetes should be viewed as a tool that becomes relevant when complexity and scale justify it, and when the organisation is ready to manage it effectively. By preparing the foundation early, organisations can choose the right moment for transition – without pressure or premature decisions.

8. References

- [1] Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). "Machine Learning Operations (MLOps): Overview, Definition, and Architecture." IEEE Access.
- [2] Sculley, D. et al. (2015). "Hidden Technical Debt in Machine Learning Systems." NeurIPS.
- [3] Gartner. (2024). "Gartner Survey Finds Generative AI is Now the Most Frequently Deployed AI Solution in Organizations." Gartner Press Release, May 7, 2024.
<https://www.gartner.com/en/newsroom/press-releases/2024-05-07-gartner-survey-finds-generative-ai-is-now-the-most-frequently-deployed-ai-solution-in-organizations>
- [4] Burns, B. et al. (2016). "Borg, Omega, and Kubernetes." ACM Queue. (Google's perspective on container orchestration evolution)
- [5] Ries, E. (2011). The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business.
- [6] Putnam, D. (1997). "Team Size Can Be the Key to a Successful Project." QSM.
<https://www.qsm.com/blog/2019/4-key-studies-team-size>
- [7] Merkel, D. (2014). "Docker: Lightweight Linux Containers for Consistent Development and Deployment." Linux Journal.
- [8] Wiggins, A. (2017). The Twelve-Factor App. <https://12factor.net/>
- [9] Zaharia, M. et al. (2018). "Accelerating the Machine Learning Lifecycle with MLflow." IEEE Data Engineering Bulletin.

Appendix A – The project “Data-driven organisations – Best practices for operationalisation of AI in Sweden

This material has been produced as part of the Vinnova-funded project Data-driven organisations – Best practices for operationalisation of AI in Sweden (DDO), a project lasting just under two years with twenty participants from private sector, public sector, and academia. Together, they tackled issues concerning large- and small-scale operation of AI solutions and how to enable and use AI broadly across an organisation.

The work focused on three specific use cases: Local sustainable operation of AI, legal and technical prerequisites for effective infrastructure, and how to create the best conditions for keeping thousands of AI models in operation.

A compilation of all material produced within the framework of DDO is available on AI Sweden's website – <https://www.ai.se/en/project/data-driven-organizations-best-practices-ai-operationalization-sweden>.

The organisations that participated in DDO were:

- Aixia <https://aixia.se>
- Hewlett Packard Enterprise <https://www.hpe.com>
- Hopsworks <https://www.hopsworks.ai>
- IBM <https://www.ibm.com>
- Linköpings Universitet <https://liu.se>
- NetApp <https://www.netapp.com>
- Predli <https://www.predli.com>
- Proact <https://www.proact.se>
- RISE <https://www.ri.se>
- RedHat <https://www.redhat.com>
- Region Halland <https://www.regionhalland.se>
- Sahlgrenska University Hospital <https://www.sahlgrenska.se>
- Statistics Sweden (Statistiska Centralbyrån) <https://www.scb.se>
- The Swedish Tax Agency (Skatteverket) <https://www.skatteverket.se>
- Stormgrid <https://www.stormgrid.ai>
- The Swedish Transport Administration (Trafikverket) <https://www.trafikverket.se>
- Volvo Parts <https://www.volvogroup.com>
- Region Västra Götaland <https://www.vgregion.se>
- Santa Anna <https://www.santa-anna.se>
- AI Sweden <https://www.ai.se/en>

The project was funded by the participating organisations and Vinnova. AI Sweden is in part financed by the EU.

SAIL (Sustainable AI Infrastructure Lifecycle)

This white paper is produced within the SAIL use case, which is part of the DDO project. SAIL focuses on building cost-efficient, environmentally sustainable, and operationally effective infrastructure that supports the entire AI lifecycle – from exploration and development to training, deployment, inference, and long-term operations.

The goal is to enable organisations to adopt and scale AI sustainably by reducing costs, minimizing environmental impact, and ensuring that AI systems can operate and evolve over time without unnecessary complexity.

Key areas of focus include:

- Designing scalable and flexible AI infrastructure that grows with organisational needs
- Exploring hardware, cloud, and modular/shared platform options
- Creating practical guidelines for long-term AI operations
- Optimizing resources and reducing technical and administrative overhead

The outcome will be a validated model and actionable recommendations that help organisations of all sizes and maturity levels build, operate, and evolve AI solutions in a sustainable, efficient, and future-proof way.

In addition to this white paper, SAIL has also produced the following white papers:

The AI Implementation Spectrum – Strategies for Sustainable and Scalable Adoption

Introduces a framework for understanding different maturity levels of AI implementation and how organisations can build scalable and sustainable strategies across the full AI lifecycle.

Sustainable Image Inference in Practice: Investigates which hardware platforms deliver the best balance of performance, energy efficiency, and cost for running AI inference on radiology images in an on-premises healthcare environment.

Benchmarking Large Language Models for ICD-10 Code Generation

Evaluates different hardware and software configurations to identify the most efficient and sustainable setup for running large language model inference when generating ICD-10 codes from clinical notes.

Appendix B – Glossary

Term	Description
Apache Airflow	An open-source platform for creating, scheduling, and monitoring workflows and data pipelines.
API (Application Programming Interface)	An interface that allows different software systems to communicate with each other.
Artifact	In MLOps, any output from a training run that should be preserved – such as model files, metrics, or configuration.
Artificial Intelligence (AI)	Computer systems capable of performing tasks that normally require human intelligence, such as pattern recognition or decision-making.
Azure DevOps	Microsoft's cloud platform for software development collaboration, including version control, CI/CD pipelines, and project management.
Challenger/Champion Pattern	A model deployment strategy where new candidate models (challengers) are tested in staging before replacing the current production model (champion).
CI/CD (Continuous Integration / Continuous Delivery)	Practices for automatically testing and continuously deploying software instead of large, infrequent updates.
Container	A packaged software environment containing everything needed to run an application – like a moving box packed the same way regardless of where it goes.
Containerisation	The process of packaging software into containers to make it portable and easy to run anywhere.
CT (Continuous Training)	The practice of automatically retraining machine learning models as new data becomes available.
Database	An organized system for storing and managing large volumes of digital information.
DevOps	A set of practices and a culture that combine software development and IT operations to enable faster delivery
Docker	A popular tool for creating and running containers. The name comes from dock workers who load and manage shipping containers.
Docker Compose	A tool for defining and running multiple Docker containers together as a unified system.
Docker Swarm	Docker's built-in orchestration tool for managing clusters of containers – simpler than Kubernetes but with fewer features.
Docker Image	A blueprint or template used to create a container – like a recipe defining how the container should look and behave.
DVC (Data Version Control)	A version control system designed specifically for machine learning projects, handling large datasets and model files.
Endpoint	A specific address or entry point in an API where requests can be sent.
FastAPI	A modern Python web framework for building APIs quickly, often used for serving machine learning models.
Feature Engineering	The process of preparing and transforming raw data into useful input features for machine learning models.
Fine-tuning	The process of taking a pre-trained model and further training it on a specific dataset to adapt it to a particular task.
GPU (Graphics Processing Unit)	A specialized processor originally designed for graphics rendering, now widely used to accelerate machine learning computations.
Grafana	An open-source platform for creating dashboards and visualizing metrics data from sources like Prometheus.
Health Check	Automated tests that verify whether a system or service is operating correctly.



Term	Description
Hybrid Environment	An IT environment that combines on-premises servers with cloud-based services.
Hyperparameters	Configuration settings for a machine learning algorithm that are set before training begins, such as learning rate or batch size.
Inference	The process of using a trained machine learning model to make predictions on new data.
Infrastructure as Code (IaC)	Managing and defining IT infrastructure through code instead of manual processes
Kubernetes	An advanced system for automatically managing and orchestrating large numbers of containers – like a conductor directing a software orchestra.
Latency	The time delay between a request and its response – in inference, how long it takes for a model to return a prediction.
Load Balancing	Distributing workloads evenly across multiple servers to improve performance and reliability.
Logging	Systematically recording events and activities in a system for troubleshooting and monitoring.
LoRA (Low-Rank Adaptation)	A technique for efficiently fine-tuning large language models by training only a small number of additional parameters rather than the entire model.
Machine Learning (ML)	A type of AI where computers learn patterns from data rather than following explicitly programmed rules.
Metrics	Quantitative measurements used to monitor how a system performs.
Microservice	A small, self-contained software component responsible for a specific function and capable of running independently.
MinIO	An open-source object storage system compatible with Amazon S3, used for storing large files such as model artifacts.
MLflow	A tool for tracking machine learning experiments and managing different model versions.
MLOps (Machine Learning Operations)	Practices and tools for efficiently deploying, maintaining, and updating machine learning models in production.
Model	In machine learning, a trained system that makes predictions or decisions based on input data.
Model Registry	A centralized repository for storing, versioning, and managing machine learning models throughout their lifecycle.
Monitoring	Continuous observation of systems to detect issues and measure performance
Node	A single server or computer within a network or cluster.
Node Exporter	A Prometheus agent that collects hardware and operating system metrics from servers.
On-Prem (On-Premises)	IT systems hosted on an organisation’s own servers and infrastructure rather than in the cloud.
Orchestration	The automated coordination and management of multiple systems or containers to ensure they work together seamlessly.
Pipeline	A series of automated steps through which data or code flows – from development to deployment.
PoC (Proof of Concept)	An early test or prototype demonstrating that an idea works in practice.
Pod	In Kubernetes, the smallest deployable unit, typically containing one or more containers.
PostgreSQL	A powerful open-source relational database system, often used for storing metadata in MLOps platforms.



Term	Description
Prometheus	A tool for collecting and storing performance metrics for monitoring purposes.
Production	The live environment where users actually interact with the system, as opposed to a test or staging environment.
Reproducibility	The ability to recreate the same results or environment multiple times.
Sandbox	An isolated test environment where experiments can be conducted without affecting live systems.
Scaling	Adjusting system capacity up or down to handle varying workloads.
Staging Environment	A test environment that mirrors production, used for validation before deployment.
Stateless	A service that does not store data between requests, making it easier to scale and replace.
T5 (Text-to-Text Transfer Transformer)	A transformer-based language model developed by Google that frames all NLP tasks as text-to-text problems.
Throughput	The amount of work a system can process in a given time period – in inference, often measured as predictions per second.
Triton	A platform specialised in serving machine learning models in production.
Version Control	A system for tracking changes to code and files over time, enabling rollback to previous versions.
VM (Virtual Machine)	A simulated computer running inside a physical machine, with its own operating system.
VLAN (Virtual Local Area Network)	A virtual network that logically groups devices even if they are physically separated.
YAML	A human-readable file format commonly used for configuration files, including Docker Compose and Kubernetes definitions.

Project Context and Contributors

This white paper was developed as part of the Sustainable AI Infrastructure Lifecycle (SAIL) use case within the national project Data-Driven Organisations (DDO), coordinated by AI Sweden.

The purpose of the SAIL use case is to explore financially and environmentally sustainable approaches to AI infrastructure that support the entire AI lifecycle — from research and development to deployment, inference, and long-term operation.

Project Partners

The work has been carried out in collaboration between: Aixia AB, Region Halland, and AI Sweden, with additional insights shared through the broader DDO consortium including industry, academia, and public-sector partners.

Authors

- Olof Sandell, Aixia AB

Use Case SAIL

- Aixia AB: Cecilia Millheim, Ellen Reinhardt, Jonas Nordin, Klas Ludvigsson, Milena Miernik, Olof Sandell, Simon Janeck
- Region Halland: Georgios Bramis, Karin Westerberg, Lina Gårdemark, Stefan Bäckström, Torbjörn Olander

Aixia AB
Hälsingegatan 10
414 63 Göteborg

www.aixia.se

Region Halland
Box 517
301 80 Halmstad

www.regionhalland.se

