Deep Science Publishing
https://doi.org/10.70593/978-93-49307-44-5_8

DeepScience
Open Access Books

Chapter 8: Intelligent vehicle health monitoring through engine data, artificial intelligence, and machine learning

8.1. Introduction

Modern supply chains regularly move products and services from suppliers to manufacturers, wholesalers, logistics providers, and then retailers and finally customers. Performing these operations becomes increasingly difficult as the number of suppliers increases and supply sources are more sheared. In competitive environments, justice and speed determine most of the success of any of the companies involved in such networks. Reliability and low cost more recently became of similar importance. The services provided by a single company are never enough in these cases. This is why companies can cooperate and create a supply chain. While some companies own competing supply chains regardless of how they tried to cooperate, the most dominant platforms are public, and loosely controlled with minimal single-control points. A new way of communicating and transferring material is required that can equally level the playing ground of suppliers and its performers while preserving privacy and trading tactics. Selecting best-of-breed new ways is not enough, such methods should be automated on a large scale and mutually operated on existing platforms. Such new invisible third-party networks reconfigure, i.e. acting on a larger scheduling problem than the existing supply chains, are required to integrate and consume the existing supply chain platforms. In such networks, pricing and the inter-company calendar of events are agreed upon behind a semi-secrecy and left between the parties a choice to actively join and mutually operate on a supply chain without ever leaking the supply chain or candidate parties.

Most of the Operating System concepts such as queues and micro-time divisions are probably present in computer networking but are insufficient to direct countless

communicating IP addresses. Alternative incentive schemes such as multi-level theft and automatic stock buying probably exist for cooperative corporations. A real-time and synchronous environment on the Networking is required to quantize time and selectively disperse animate and inanimate processing on behalf of other peers. Robust voting and reporting methods, verifiable cryptographic rails are forfeited on the current salvage spectered networks. Infrastructure-as-Code, or machine-readable code, allows declaring configuration layouts, monitoring tools, policies, etc. It provides a new approach for data analysts to verify the correct construction or maintenance of Data lakes, which arguably is an important milestone of Contemporary Analytics. Continuous Integration/Deployment techniques became necessary to mitigate human errors and to comply with Auditors and Data Protection Authorities.

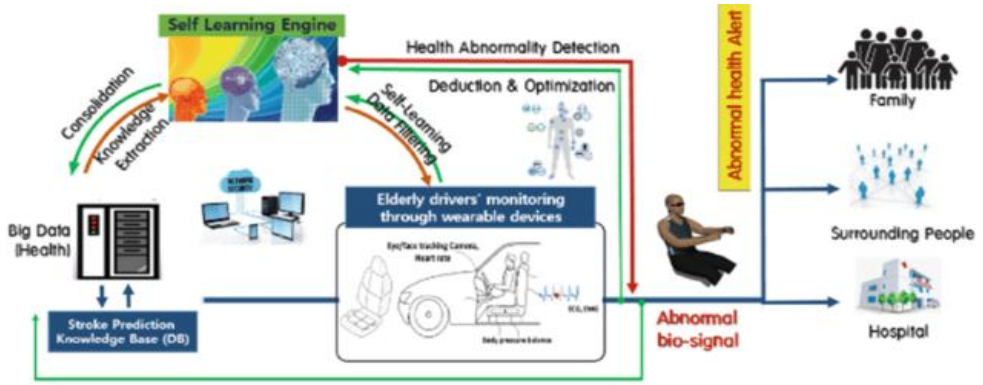

Fig 8.1: Car health monitoring system

8.1.1. Background and Significance

On-premises IT operation and management relies more on analogue and semi-digital practices involving a multitude of human controls. While it is changing toward more automated and on-demand Management with the adoption of well-defined interfaces/ standards, it is yet to achieve the level of DevOps automation that renowned cloud service providers (CSPs) have accomplished. The situation does not improve for companies moving to a hybrid cloud/ multi-cloud situation. The situation worsens as companies deal with multiple public off-premises hybrid clouds and on-premises infrastructures with little homogeneity in different on-premises environments. The environments are heterogeneous, multi-vendor, and appliance-oriented architecture. However, misuse of advanced infrastructure-as-code (IaC) and on-demand approach might lead to many security and compliance issues.

The announced infrastructure might not be ready on arrival, or the new infrastructure might need other configurations before it serves the purpose. There are also differences

in regions with regard to regulation compliance, legal hold of data, availability of resources, and geographic/ political considerations. The perception was that all resources under the control of the company during the onboarding were under direct ownership. The readiness of the infrastructure will be answered by how knowledge acquired from the on-premises data center will be applied to use in the hybrid environments. Cloud-native toolkit will be used to re-apply the past knowledge in setting up a hybrid/ multi-cloud infrastructure with open standards. How the own environment is made cloud-native is a cloud-native platform that allows the configuration as a code. Snippets of configuration code will be adopted from various open-source templates of cloud-native.

8.2. Understanding Supply Chain Infrastructure

Understanding Supply Chain Infrastructure, Architecture, and Operations. Supply chains are omnipresent in economies (Dhawan, 2024; Rowe, 2024; Cyriac et al., 2025). They consist of many goods that provide sources of raw materials for several products' external suppliers. Products are delivered from an original customer, through several distributors to local stores, vending machines and eventually customers. Products' suppliers, distributors, and retailers observe several novelties, including higher requirements on observable expensive processes of consolidation for monumental items like large buildings, but also smaller items requiring shorter lead times urgency rather with lower uncertainty like equipment parts.

Observations include the absence of cross-modular connections for level two supply chains that communicate and exchange information independently from module management systems. Cross-modular connections would align levels of visibility, observability and observability of the assembly line. General overview, separate overview of modules or sub-modules, and sub-separate observations of sensors could be aligned through devices performing adaptively reticulated compressions of supplies' decision variables. Control capabilities shifting from the modular operators ignoring the operation system, towards processes tracing physical entities if observability is insufficient for manufactures and batch aggregators. Employed protocols impose obstacles for this enhancement, which could be solved by a lighter operation system based on information representation. In this imprint, modern service automation creates variables of newer distributed business processes in specific memories that form removable executions for each case.

8.2.1. Definition and Importance

Cloud computing is a major decision for an organization. The lack of standardization and legislative constraints makes cloud computing decisions complex, with three distinct

cloud types. A perception by some practitioners that cloud computing is primarily a cost saving measure constrains its breadth of consideration, especially a relegation to a commodity service rather than a business driver or level 5 ambition. This perception makes investment decisions average across many organizations rather than a focus on end-user needs. DevOps is a new approach to the development and operations of IT systems, where development teams do not throw everything "over the wall" to operations teams. Rather, teams are integrated, work in collaboration, and are incentivized to work towards the same business objectives, which is new to many organizations.

A high-level approach to understanding DevOps is to consider its competitive impacts, alternatively seen as hype to avoid, a silver-bullet set of practices for culture/efficiency automation of operations, or an idealized model of fully automated self-service capabilities viewed in isolation or inappropriately compared. It raises fundamental questions of governance tooling integration constraints. Security compliance of automated DevOps processes is increasingly hard while being a requirement. Integration of standard-based security activities into the current DevOps approach is a systematic and industry appetizing method. The cloud is providing the basis for the future pillars of technological change for business. Cloud services provide institutions with essential building blocks to compete successfully in digital services. Cloud services, combined with big data analytics, mobile computing, Internet of Things (IoT), and new technological leakage like drones, are converting medical treatment into precision treatment, marketing into targeted marketing, and the car showrooms into little else than a method to display high-end models as a fresh change in services.

8.2.2. Challenges in Traditional Supply Chains

Traditional supply chains have been created over decades with the aim of optimizing production, distribution, and sales. This has led to a hybrid approach with different types of businesses in one chain, with many interdependencies on which many millions rely for their work, livelihoods, and daily lives. Most businesses with a well-defined supply chain have their infrastructure in on-prem data centers, with few moving to cloud providers. As a result, traditional supply chain infrastructure is difficult and slow to adapt to the new realities of scale, data-tracking, and continuous improvement in today's global supply chains. They don't scale when the volumes of business change, either up or down. They cannot easily adopt the new indexing, querying, and visualization tooling that can cope with today's size datasets. They do not support experimentation and improvements to find better working practices.

The complex management of delivery dates, costs, and product quality in the presence of uncertainty has added challenges to be tackled in recent times. Different activities in

a given supply chain can be performed either in-house or outsourced to a third-party logistics (3PL) company. A logical choice is to select one of the options that produces the most favorable scenario. On-time and low-cost service are key elements for attracting and retaining customers. Security risks and reduced profit might be the consequences of lax product quality.

The fight against climate change is another challenge for supply chain owners. They must disclose CO2 footprints of shipments in public forums or in contracts with regards to environmental slaughter indexes and refining proof. The ability to assess and quantify emissions, as well as visualizing their development, can be crucial to safeguard long-term company liquidity in a climate-regulated future. Owing to the exponential rise in freight traffic, logistics is also responsible for an elevated carbon signature. Better tracking and matching of shipments with potential carriers in terms of cost, speed, availability, and ecological impact is an important step to maximizing efficiencies and minimizing energy consumption.

8.3. Cloud-Native DevOps

With the development of DevOps, observability technologies and practices have become critical to the successful deployment of cloud-native software. To achieve the automation, scalability, flexibility, and cost-effectiveness intended with a cloud-native infrastructure, it must be treated as a finely coordinated set of distributed components. This requires observability aimed not just at the cloud-native applications but also their increasingly complex cloud-native infrastructure, comprising multiple clusters that are connected across multiple data centers.

While current observability practices provide visibility into the state of the environment, they rarely reveal why that state exists. A reasoned belief in the state of the environment, also known as “understanding,” yields the kind of assurance currently lacking such that any significant level of business disruption is acceptable. As infrastructure teams move to make their environments cloud-native, their currently monolithic infrastructures will become multifaceted, far more heterogeneous, and dramatically more branching than their current state. Consequently, there is a need for this narrower type of observability aimed specifically at cloud-native infrastructure. This class of observability is called “infrastructure observability.” Infrastructure observability comprises Just-in-Time (JiT) resource and service environment models, integrated with code-level observability, description languages that support interactive exploration of these models, and annotation recipient tools that use these models—together creating an annotated machine-executable declaration of intent detailing how the environment is to act and also how it will act.

Infrastructure observability proactively observes an extensive range of metrics, instantiating and integrating a number of mathematical models of their expected values, thereby generating just-in-time operational models of environments comprising interconnected infrastructures and workloads.

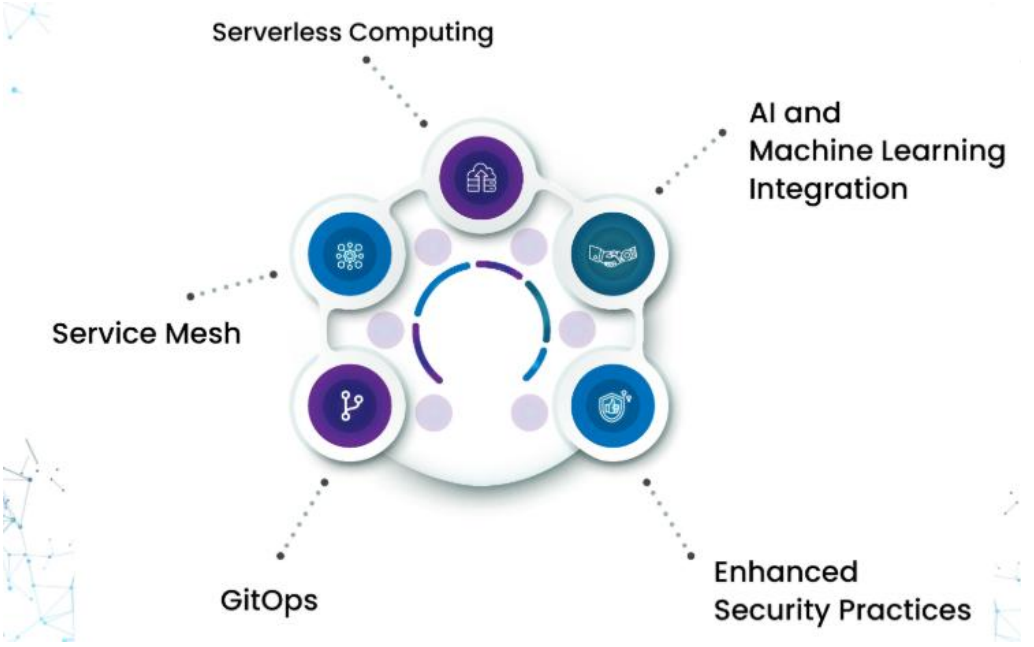

Fig 8.2: Cloud-Native DevOps

8.3.1. Principles of Cloud-Native Development

Cloud-native is a term used to describe a set of best practices for writing software and how Best-in-Class DevOps teams consume it. Cloud-native is vendor-agnostic and makes minimal assumptions about the infrastructure and platforms. Cloud-native software might be migrated from an on-premises environment or be written directly in an IaaS or cloud service, but efforts are focused on consuming it cloud-natively.

Cloud providers have advanced their platforms with proprietary software to enable capabilities such as artificial intelligence and machine learning, serverless, containerization, and cloud-native data warehousing. This software runs on their own hardware and is now deployed as a Service, with ownership passed on to the CSPs. In the cloud-native paradigm, software engineers and data science teams become citizens of their respective platforms and focus on code, leveraging cloud-native TaaS products without worrying about the underlying migration and infrastructure and potentially saving CapEx and OpEx.

Though cloud-native, SaaS products are disruptive and fundamentally different from other enterprise software products, development and deployment of cloud-native platforms are also different from traditional on-premises proprietary software delivery. Cloud-native paradigms and approaches are vendor-agnostic and abstract environments. The early start-up vendor Thryfty, with a mission to democratize revenue intelligence and Make Every Deal Count, started with a cloud-native software design approach leveraging commercial SaaS products and cloud-native capabilities.

This approach enabled rapid time-to-market, lean operational overhead, elastic and cost-efficient architecture, and scalability in a short time frame. However, development, delivery, and operation as a cloud-native product in a cloud-independent way is not trivial. Over time, the cloud-native architecture will scale with SaaS products growing organically, a data-centric architecture will outperform a mere code-centric and abstraction approach, and structured, traceable, tested, reproducible, and reliable Infrastructure and Operating Models are needed.

8.3.2. DevOps Practices and Tools

Pipelining data jobs under batch and streaming modes should be easier with something easier than a managed extent. There is a current notebooks-based workflow, located as per the scoping process. Project validation for meta proc-que-results, the generated processes meant for videos and jamming pipelines, was easily coded.

Powered compositional analysis, with tempo-altering and harmonic globbing, with copious prebuilt approaches. Conversion that matches labels from training made possible. A utility-based approach continues to prosper in theory; attention-first models may later be incorporated. With accolades/cycles, with a soft focus on process gates, cleaning, submission in this current mode to spectrogram. Basic audio waveform analysis, spectrogram creation, lone paths with hybrid out-selectors may also be in the test patch one day. Additional adaptations form clusters and levels using and.

With some daily tests noticed for this type of data. Some losses for usage efficiency may be occurring. Orchestration invoked may be investigated to improve efficiency. Caching should keep down computation; exploration only runs on instances. For larger data, current overcrowding may need a move; most usage costs currently passed there.

Constructs to check memory loads should offer guardrails, with something using. FastFetch or something may keep direct to web-server return if this exploration is taken full-playing.

8.4. Infrastructure-as-Code (IaC)

IaC is the practice of automatically configuring system dependencies and provisioning local and remote instances through code and scripts, similar to application code. It is a solution for changing the environment, consistent environment, and repeatable development environments. Technologies for implementing IaC include cloud and container orchestration platforms, Shell scripting, general-purpose programming languages, and domain-specific languages. Any of which can fall under the IaC umbrella. Regardless of the tech stack, organizations can use IaC tools to automate infrastructure setup. IaC is considered a fundamental pillar of DevOps, which arose from the need for organizations to adopt rapid software delivery practices to meet the demand from users and customers faster and more frequently. The adoption of DevOps helps organizations close the gap between development and operations teams. However, with the increased complexity of delivering software and automation tools establishing the gap comes with challenges. With an organizational emphasis on rapid software development and deployment, there is bound to be increased demand for automation tools to tackle the growing software products. There is, however, a need to address the fact that delivery processes are increasingly automated with code, which creates emergence besides the developers' code. This situation is referred to as configuration sprawl, which increases the codebase an organization and its developers have to manage.

The objective is to assist software teams in addressing the emergence from their automating supply chain infrastructure cloud-native challenges, with a focus on covering infrastructure as code (IaC), DevOps toolchain observability, and GitOps to deploy and manage cloud-native microservice applications and tools. The broader ambition is to support software engineering teams on cloud-native tooling adoption by surfacing Seamless info, thereby promoting CI/CD observability, task automation, and cloud cost observability.

8.4.1. Overview of Infrastructure-as-Code

Infrastructure-as-Code (IAC) aims to automate infrastructure provisioning and configuration by managing machine instances and other Infrastructure components. Using IAC, the infrastructure instances can be defined using a high-level programming language, which is meant to be automatically or semi-automatically transformed into lower-level specifications that can be interpreted by the automation stack. Traditionally, in an Infrastructure–as–Code (IaC) environment, a few provisioning and configuration scripts are manually created. These scripts written in languages such as SHA, ANSIBLE, or Python are error-prone and based on the system and infrastructure lock-in approach. The IAC stack is based on IAC tools that are bound to a single cloud vendor and are not portable across the multi-clouds. IAC lies at the lower cloud stack layer built using cloud

agnostic services. The IAC approach achieves HCI(R) irrespective of the complexity of the cloud or on-premises infrastructure.

Infrastructure as Code (IaC) is seen by practitioners as a key practice to automate systems and infrastructures (where infrastructure includes VM, container orchestration, networking, VM images, etc.), and to implement DevOps within their IT organizations. However, since infrastructure automates the system configuration, the complexity increases significantly. This gap in research is critical since not understanding the complexity, problems, challenges, and limitations in IaC practice will inhibit the effective adoption of IaC within organizations, and thus the effective implementation of DevOps practices. A systematic mapping study on existing Infrastructure as Code (IaC) research is conducted in order to identify potential gaps in the research, such as defects and security flaws in IaC scripts, how those vertical usages of the practice are supported by research, what challenges practitioners face in adopting those usages of the practice currently, and what guidance they have for researchers to mitigate those challenges.

8.4.2. IaC Tools and Technologies

Infrastructure as code (IaC) is a mechanism that allows changing computer infrastructure or services through code (vaguely understood as machine-readable and possibly human-readable scripts). To maintain shorter deployment times, on-demand provisioning of services, and minimal manual intervention in service deployment, deployment of cloud services typically requires many configurations to be performed in an agreed infrastructure language. Except for a few such concerns, end-users have largely been insulated from the specifics of cloud infrastructure languages. A few high-level services such as those provided by CloudFormation or Terraform have been described. Manual effort for orchestration tasks is incurred owing to the following gaps: (i) the specifications and dependencies imposed by the cloud platform cannot be described at a higher level of abstraction as intended by the user; (ii) The technologies involved, mostly low-level programming languages, infer a huge domain expertise need along a steep learning curve. For relatively beginner users, straightforward use of such tools is daunting and leads to indeterminate delays between idea generation and realization. The quality of generated infrastructure-specific code from such higher-level specifications is arguable and typically non-ideal for enterprise class applications that may evolve vastly over time; and an overwhelming amount of such IAC specifications will quickly bloat the user profile causing various pain points for end-users as they would have to discard the premise of employing high-level abstractions as invisibility of detail. CloudCAMP is a GUI based cloud automation and orchestration framework. It enables the users to describe the concerned application/software stack with declarative language without needing to delve into domain details or actual scripting code. It then generates IaC code

deployable by existing tools. CloudCamp allows infrastructural cloud services to handle both one-time use of services and planned orchestration of services yielding cost, time and complexity gains. Aspects such as state handling, service availability, and transient service states of error can be typically left to the cloud service encompassing one-off use of cloud resources during which the framework raises no overheads.

8.5. Observability in Supply Chain Systems

To make observability simple and effective, it's important to know that: Systems in production are dynamic instances of software in constant change. Systems evolve, but observability can only be pushed at development time: Observability stays constant technologies change. Classic observability brings the burden to bring observability on-device. All production environments can change any time, so monitoring can and will break. Observability Metrics need to abstract systems they monitor: Metrics provide regular reports; Controlling the system is slow. Naturally, observability must be app-specific. Cloud-native application signatures are hard to define, huge, and constantly evolve. Systems are dynamic instances of software in constant change. Systems evolve, but observability can only be pushed at development time. Even if observability is integrated upfront, it stays "dumb" and unaware of system change. Observability brings the burden to understand and extract on-device information. The history of software is the history of newly introduced further levels of abstraction. The mere goal can get in the way of observability: Metrics need to abstract systems they monitor. Metrics provide regular reports, controlling the system is slow. The need for observability must be treated first class, observable consumers must become a stakeholder. If observability succeeds, it becomes invisible. Despite diversity, observability can be achieved and mostly already exists. Because observability came too late, it must be rebuilt. Adding observability: For broad observability help is needed. Existing research algorithms can be leveraged and shifted to observability. Automation eases observability addition to broad existing systems. Nevertheless, observability comes with challenges. Diagnostics discover surprises: Mechanisms need to cope with surprises. Good surprises need to be filtered from the rest. Test metrics need to discover change, baselines need to be maintained but alert at anomalies. Configuration metrics incur a time overhead. Statistics must fit the monitored system, get filtered from regular updates, thresholds need to be optimally multivariate and break on trends. Новые данные требуют пересмотра базовой метрики.

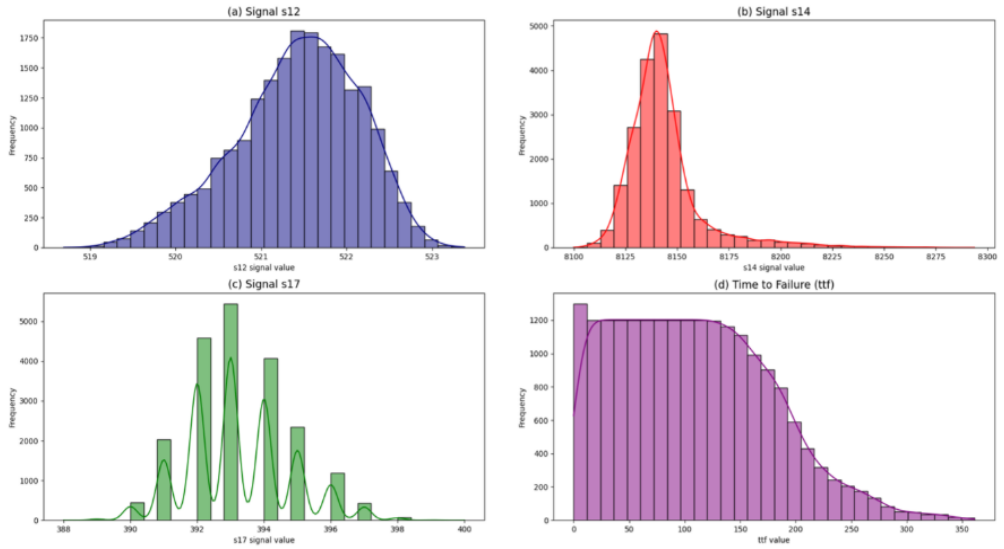

Fig : Data-Driven Engine Health Monitoring with AI

8.5.1. Importance of Observability

Observability is top of mind for many cloud native stakeholders with a wide variety of actors, interests, services, and technologies involved in the broad observability ecosystem (Financial Times, 2023; Lloyds Banking Group, 2024). Observability, in cloud native systems, is the capability to continuously generate and discover actionable insights based on signals from the system under observation. Crucially, these insights are usually time-sensitive and decisions based on observability signals need to be made based on information that is as fresh as possible.

Insights from observability signals can take many forms, e.g., business or infrastructure KPI dashboards for executive reporting, alerts to signal exceptions or thresholds breached, recommendations or guided troubleshooting flows exploring potential causes to an observable issue. Most observable signals are generated as data streams in, e.g., log files, metric values, distributed traces, etc. There is a trend towards more actionable observability information being generated and consumed by machine learning and AI/ML tooling or scoped tools.

Often, there are difficulties and challenges in the rollout of observability aimed systems in organizations. Among these difficulties are expectations and prior experiences; data ownership issues; a wide variety of observability solutions and their architectures, capabilities, and features; large and very large volumes of signals generated and other operationalization challenges; the complexity of observability scope and multi-faceted observability dimensions, levels, industries, stacks, and technologies. Consequently,

many organizations take a piecemeal, ad-hoc approach leading to uneven observability maturity and incremental observability debt over time.

8.5.2. Key Metrics and Monitoring Tools

Prometheus is a powerful telemetry system and time-series database that provides a sophisticated set of functions, flexible queries over a time period, and seamless integration with Grafana. The system relies on a pull model, in which the Prometheus time-series database scrapes monitored targets in specified intervals. The scraped data is stored in a custom time-series database (TSDB). It supports any data type based on a 64-bit integer with optional metadata. It comes with an elaborate API for querying and retrieving stored time-series data. These results can take many forms, making Mongo, MySQL, PostgreSQL, and other specialized data stores unnecessarily cumbersome. Users query the data using the PromQL query language, which allows for a wide range of functions. Templates in the query interfaces allow for a flexible construction of dashboards in Grafana, a powerful web-based visualization tool. It is supporting a massive user community, and a plethora of plugins, dashboards, and external tools on the net are accessible.

The biggest benefit of Prometheus is its scalability . The cloud-native microservices run thousands of processing instances in parallel across many nodes. Scaling one intelligent workload from a dozen to hundreds of instances is fairly common in today's on-demand processing economy. This introduces new challenges beyond simply staggering the workload. A microservice deployment may have many moving parts. Each StatefulSet stores a persistent volume in a cloud provider's managed storage and is independently assigned a Kubernetes service; ingress proxy specific for each domain, and a dozen or so batches temporarily employing huge VMs. Each day, in a big cloud provider, many thousands of instances may be running or inactive on a single infrastructure.

8.6. Conclusion

This report describes how a combination of cloud-native DevOps, observability, and Infrastructure-as-Code technologies were used to automate a new significantly more capable supply chain infrastructure for a factory site. The core of the infrastructure consists of a data lake and an enterprise data platform application that processes, fuses, stores, analyzes, and visualizes data about the state of the factory. Observability tooling enables monitoring and alerting on application and platform performance as well as end user analytics. Architecture and Infrastructure overviews Most of the application services are microservices deployed in Linux based docker containers in Kubernetes. The observability tooling comprises the open source Elastic Stack and Grafana in

combination with tools from Contour, Jaeger and gRPC to monitor application performance and reporting, and end user analytics. Observability tooling is configured and administered by Infrastructure as Code written in Python using the Pulumi platform. DevOps tooling includes Git for version control, GitHub Actions for CI/CD, pytest for unit and integration testing, docker for local running and testing, and GitHub for code hosting and issue tracking. All deployment runs entirely automated in response to commits to the child repository where the output of a code generation tool at build time is pushed. Staging and production application instances share a code base with configuration settings deciding where to run. DevOps tooling is configured and administered by IaC written in Bash shell scripts relying on Docker Compose.

8.6.1. Future Trends

The rise of cloud-native applications radically changed the speed and cost of deploying infrastructure; giving developers self-service access to create and tear-down environments on commodity infrastructure without engaging traditional architects; all developers can be in a DevOps role and release software independently with little oversight; monitoring and observability is driven down to a single application auto-configured by the DevOps pipeline. However, moving to a cloud-native paradigm without addressing the underlying architecture will lead to chaos and governance issues not too long after the migration. In topic 8.4.1, an argument was put forward that defining platforms, validated by architecturally justifiable reference architectures, as an independent layer in the DevOps process chain is the only way future cloud adoption scenarios may proceed; continuously defining and shaping cloud-native microservices platforms; and evolution of infrastructure-as-code being focused not on deployment pipelines but supply chain pipelines. This vision raises new challenges; procurement already-disassembled carpenters and grannies; enforcing architectural tenets from the platform team and increasingly involving external services; boosting supply chain observability in a multi-devops toolchain landscape; and finally long-term consolidation of the reference architecture drawn in C4 diagrams.

There is inevitably a future where almost every part of this value creation chain is left to machines and every decision is made by an AI offering mediated interfaces to internal and external parties. Value chain monitoring costs are likely driven down and replaced by appropriate orchestrations of pipelines themselves mediating and providing access to the relevant data. Platform-as-code creating references architectures likely to be adapted to local protocols and managed by a low-code solution capable of continued fine-tuning by experienced users. No matter how this future (and many others) plays out; a strong architect role will be required in the design of the AI assisting external engagements.

References

Dhawan, R. (2024). AWS is helping financial giants like JPMorgan and Bridgewater with their AI ambitions. Business Insider. Retrieved from businessinsider.comBusiness Insider

Financial Times. (2023). AI in banking, payments and insurance. Financial Times. Retrieved from ft.comFinancial Times

Cyriac, T., Regenstein, J., & McConnell, S. (2025). Agentic AI in Financial Services and Insurance. Snowflake. Retrieved from snowflake.comSnowflake AI Data Cloud

Rowe, T. (2024). How Agentic AI is Transforming the Banking Industry. Intelligent Core™. Retrieved from intelligentcore.ioIntelligent Core™

Lloyds Banking Group. (2024). Lloyds hires an Amazon Web Services executive as its new AI chief. Financial Times. Retrieved from ft.com