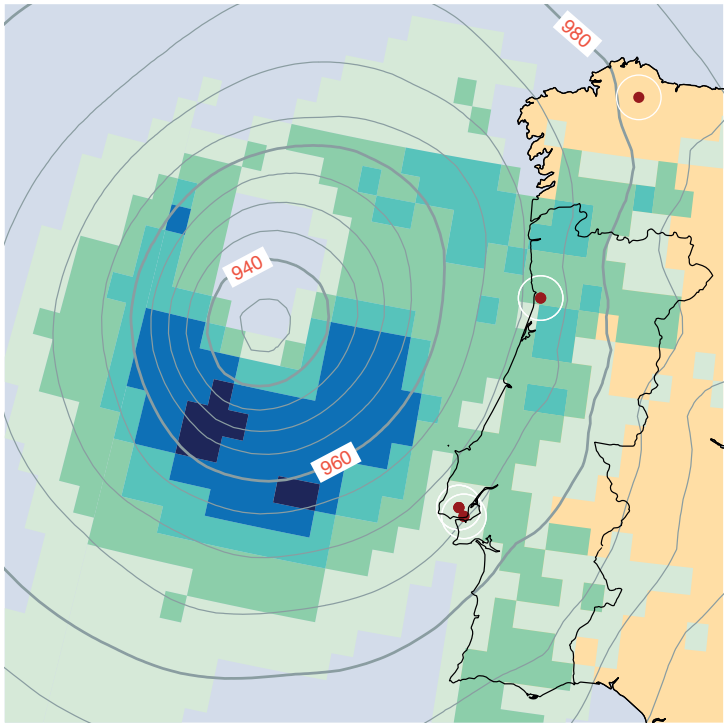
ECMWF Feature article

from Newsletter Number 175 – Spring 2023

COMPUTING

Automated weather data services deployed in Bologna



www.ecmwf.int/en/about/media-centre/media-resources

doi: 10.21957/sd32f71jlp

This article appeared in the Computing section of ECMWF Newsletter No. 175 – Spring 2023, pp. 38–43.

Automated weather data services deployed in Bologna

Sébastien Denvil, Simon Smart, Manuel Fuentes, Baudouin Raoult

In 2022, ECMWF migrated its 450+ PiB data archive (700 PiB when counting backup copies) from Reading, UK, to its new data centre in Bologna, Italy. It did so without interrupting or delaying its normal operations. We briefly describe ECMWF's Meteorological Archival and Retrieval System (MARS), the Fields DataBase (FDB) and ECMWF's File Storage system (ECFS), and we present the transition periods of this migration. We then discuss the challenges and opportunities the migration offered for the evolution and consolidation of our operational practices.

To do so, we will need to define what software deployment automation, Infrastructure-as-Code and DevOps are, and to explain why these concepts are important for modern software development and operations. Doing this, we will clarify concepts that are often confused with each other. The article will finally reaffirm what the keys elements are that enable a rapid and reliable transition from service development to operational release.

MARS, FDB and ECFS services

MARS and FDB are domain-specific object stores developed in-house for storing, indexing, and retrieving weather-related data in the GRIB format. Each GRIB message is stored as a field and indexed according to semantically and scientifically meaningful metadata (including physical variables, such as temperature, pressure, ...). A set of fields can be retrieved specifying a request using a specific language developed for accessing the MARS archive or FDB content. ECFS on the other hand, is ECMWF's unstructured archive, providing users with a logical view of a seemingly very large file system. It is used for data not suitable for storing in MARS. UNIX-like commands enable users to copy whole files to and from any of ECMWF's computing platforms. ECFS uses the storage hierarchy of disks and tapes within the high-performance storage system (HPSS) to store the files and a dedicated database for their associated metadata (file ownership, directory structure, etc.).

MARS is a scalable system as it decouples the physical organisation of the data from its logical organisation. The system is split in two parts. The first part, the MARS Server, contains the semantic knowledge of the data. It knows what a meteorological field is and what a forecast is. The second part, the Data Server, has a physical knowledge of the data. It knows if a piece of data is on tape, on disk or cached. Figure 1 shows the architecture of MARS and Figure 2 is an example of an object archived into MARS.

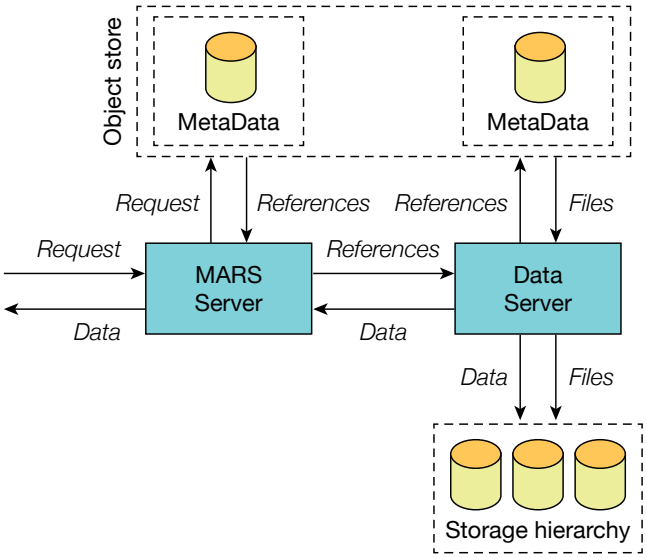


FIGURE 1 MARS architecture, from requests, to references, files, and data. The MARS Server does not handle files but data references. When a user request is processed, the MARS Server translates it into a list of data references, which are passed to the Data Server. The Data Server translates data references into actual files and returns the data.

doi: 10.21957/sd32f71jlp

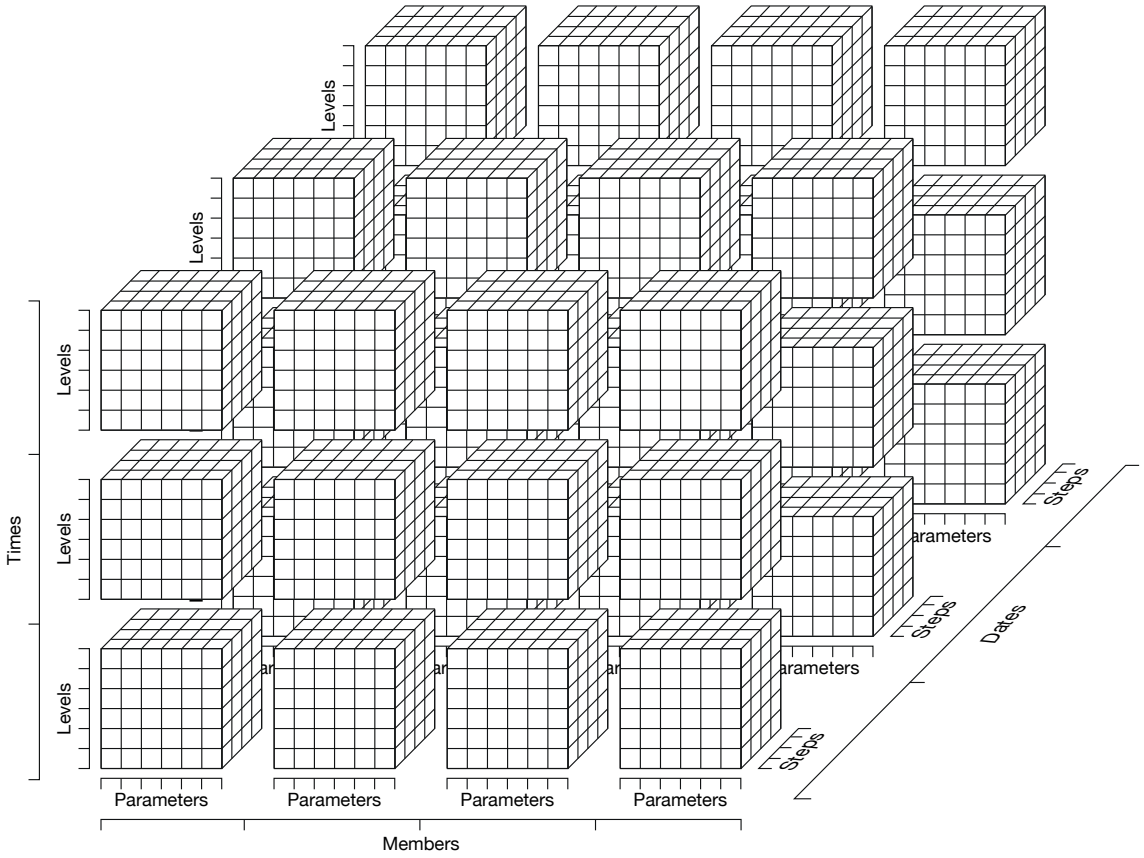


Figure 2 Example of an archived object in MARS, stored as hypercubes. In this example, each individual cube corresponds to a particular time, forecast date and ensemble member, and it is subdivided according to parameters, levels, and time steps.

The MARS Server does not handle files but rather data references. When a user request is processed, the MARS Server translates it into a list of data references that are passed to the Data Server. The Data Server translates data references into concrete locations and returns the corresponding data. By using this design, we have a system that is independent from the underlying hardware (disks, tape libraries, etc.) and from the underlying software (such as HPSS). The data can be physically reorganised without any impact on the system. Data files are split or joined and moved from disk to tape without involvement of the MARS Server. We have learnt from experience that the fewer files we have, the more manageable the system is. With this architecture, we can reduce the number of files by merging fields into larger files.

On an average day, the system handles requests for more than 13,000 tape mounts, the archive grows by about 287 TB, and 215 TB is retrieved. MARS data represents about 75% of the volume of data stored, but only about 4% of the number of files. ECFS data represents almost all the remaining 25% of the data, corresponding to 96% of the files.

Between 8 September and 11 November 2022, the complete data archive was moved from Reading to Bologna (see the Box for further information). At the beginning of the BOND (Bologna Our New Datacentre) programme, ECMWF's Technical Design Authority (TDA) was set up to manage the overarching technical governance for ECMWF in the context of BOND. An automation work stream was defined that helped shape the work that has been done and that is presented in this article. The corresponding automation journey for FDB/MARS/ECFS, together with the challenges inherent in software-defined infrastructure and services, is discussed below.

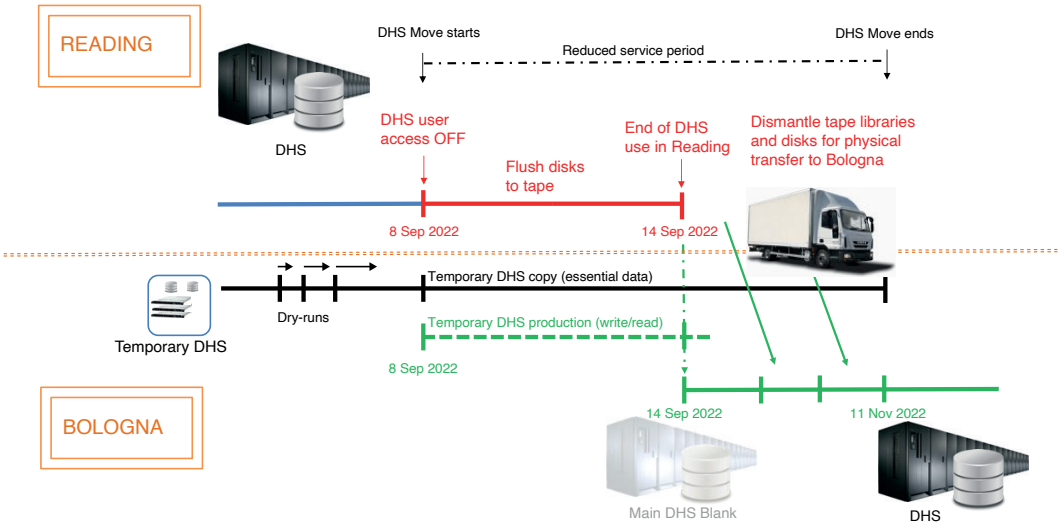
Moving the data archive to Bologna

A

Through the migration of the data archive, ECMWF had to preserve its forecast, product generation and archival services, maintaining appropriate levels of service for its users. The planning and testing of a continuity plan of operations for operational forecasts, time-critical suites belonging to Member and Co-operating States, the services provided by the EU's Copernicus services implemented by ECMWF, and research activities was established. As it was both practically and financially unviable to maintain access to the full 450 PiB archive through the migration, the plan relied on exhaustive identification and traceability of the data flows involved in time-critical operations.

The figure shows the transition plan from the Data Handling System (DHS) being operated from Reading to being operated from Bologna. The plan was successfully trialled on three occasions (labelled dry-runs in the figure): on 5 April 2022 for ten hours, from 26 to 28 of April 2022 for 48 hours, and from 28 June to 5 July 2022 for seven days. Together with key requirements being met in the Bologna data centre, the success of these trials gave the green light to the real move, which started on 8 September 2022 at 9:30 UTC.

At that point, two temporary systems in Bologna took over from the production systems in Reading. On 15 September, the final systems in Bologna, which were mostly empty at the time, were reconfigured to take over from the temporary systems. The dismantled tape libraries, disks, and servers from Reading were gradually transferred, and reassembled in Bologna. Half of the archive was accessible by 28 October, 75% was accessible by 3 November, and the complete archive was accessible by 11 November.



The MARS and ECFS service transition between Reading, UK, and Bologna, Italy, took place between 8 September and 11 November 2022.

The role of automation

Regardless of the complexity of the environment, an operations automation strategy will help improve existing processes. Automation can save time, increase quality, and reduce costs. Centre-wide approaches can help to realise the full value of automation for modern, digital operations. Centre-wide automation enables an organisation to manage complex IT environments more readily and integrate new technology and processes more effectively. ECMWF, being both a research institute and a 24/7 operational service, must position itself very carefully in this context to ensure both reliability of service and a fast movement from service development to production cycle. The move to Bologna has been used to make a significant step change in the way we automate the deployment of central services.

Automation uses repeatable instructions to replace manual work in the field of IT. Imperative and declarative forms of automation come up frequently. The distinction between those two forms of automation is important. Both terms refer to how to provide direction to the automation software. With an imperative tool, you define the steps to execute to reach the desired solution. With a declarative tool, you define the desired state of your system, and the automation software determines how to achieve that state. In a software engineering context, declarative programming means writing code to describe what the program should do

doi: 10.21957/sd32f71jlp

as opposed to how it should do it. An example of a declarative form would be for you to state that a service should be in a state of running. The imperative alternative is to simply start the service.

Both have their benefits and drawbacks. Imperative languages are more focused on giving specific instructions to a computer to solve a problem. They can handle more complicated tasks and tend to be faster. Declarative languages, on the other hand, are focused on describing the end goal without worrying about the specific steps to get there. This makes it easier to ensure that the system is always in the desired state, even if the process is repeated.

Idempotency is a key concept in automation that refers to the property of a process or task of being able to be executed multiple times without producing different results. In other words, when a process is idempotent, running it once has the same effect as running it multiple times. In the context of automation, idempotency is important because it helps to ensure that the desired state of a system is always achieved, regardless of how many times the automation tool is run. For example, if a task is designed to ensure that a particular configuration is applied to a set of servers, running it multiple times should not produce any unexpected changes if the desired configuration is already in place.

By ensuring that a process is idempotent, automation tools can reduce the risk of errors and unintended consequences, while also increasing the efficiency of the automation process. It also makes it easier to manage large-scale environments with many interconnected systems, as the same task can be run repeatedly without worrying about unexpected changes. With a declarative language having the idempotency property, you always end up in the same place, no matter where you start. In contrast, imperative language defines a series of steps that must be followed to complete a task, which can lead to different results depending on the starting point.

Ansible and Puppet are popular automation engines that have a strong track record as declarative systems, amongst many other players. Both were evaluated and used during the preparation of the move to the Bologna data centre. They are usually classified as best serving different use cases. Puppet is more of a configuration management tool, whereas Ansible is more of a provisioning, configuration, and deployment tool, which is closer to what we are after across the variety of services ECMWF is managing.

Ansible is the automation engine now in use at ECMWF. It has been used before, during and since the completion of the move to Bologna. Ansible is an open-source, command-line IT automation software application written in Python.

Ansible has the concept of a control node and a managed node. The control node is where Ansible is executed from. Managed nodes are the devices being automated, for example a MARS mover server (see Figure 3).

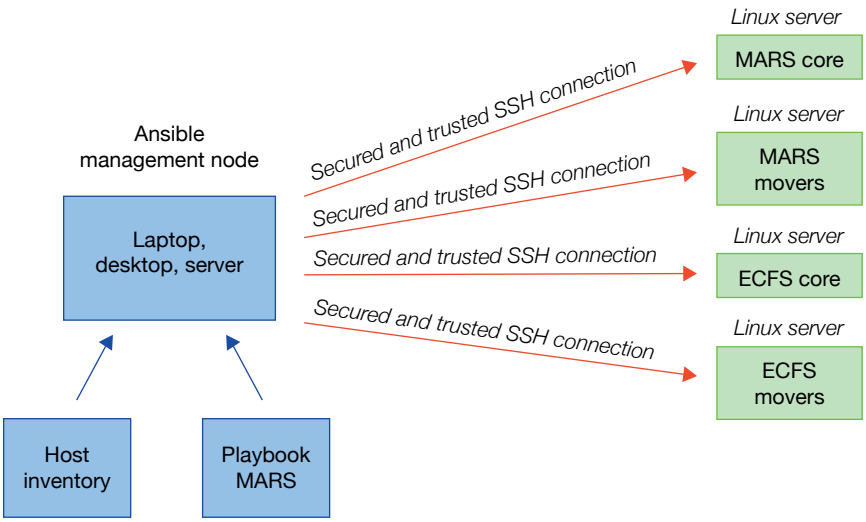


Figure 3 A view of the Ansible control node and managed nodes. The management node in blue is where Ansible is executed from. Managed nodes in green are the devices being automated, which are listed in the inventory, for example a MARS mover server.

Ansible's basic concepts are the following:

- **Inventory** lists and groups **hosts**
- **Playbook** contains **plays** and can run against hosts or groups of hosts
- **Plays** contain **tasks**
- **Tasks** call **modules** (Files modules, System modules, Storage modules ...)
- **Tasks** run sequentially on a given host
- **Tasks** can run on multiple hosts in parallel
- **Handlers** are triggered by tasks and run once, at the end of **plays (stop, start, restart)**
- **Roles** automatically load **tasks** and **handlers**. Grouping content by roles also allows **easy sharing** of roles with **other users**.

MARS, FDB and ECFS share many design and deployment concepts, and hence large fractions of their codebases that are core reusable building blocks. This is reflected in the deployment and configuration structure, with the ability of Ansible to define role dependencies and with the Ansible variable inheritance and scope mechanism. Role dependencies and variable inheritance are essential to avoid redundancy of information.

Ansible uses variables to control how roles and tasks behave. If multiple variables with the same name are defined in different places in the code hierarchy, they override each other in a specific order. Default values can be set for included or dependent roles, and these will have the lowest priority of any variables available. They can be easily overridden by any other variable, including those supplied in the inventory or on the command line.

Figure 4 shows Ansible roles forming the building blocks of MARS/FDB/ECFS automation and their dependencies. For example, 'service-common', 'mars-common' and 'ecfs-common' roles are the foundational roles on top of which we specialise deployments and configurations. This is paramount to avoid defining variable or file content at multiple places and to ensure consistency across our deployed services.

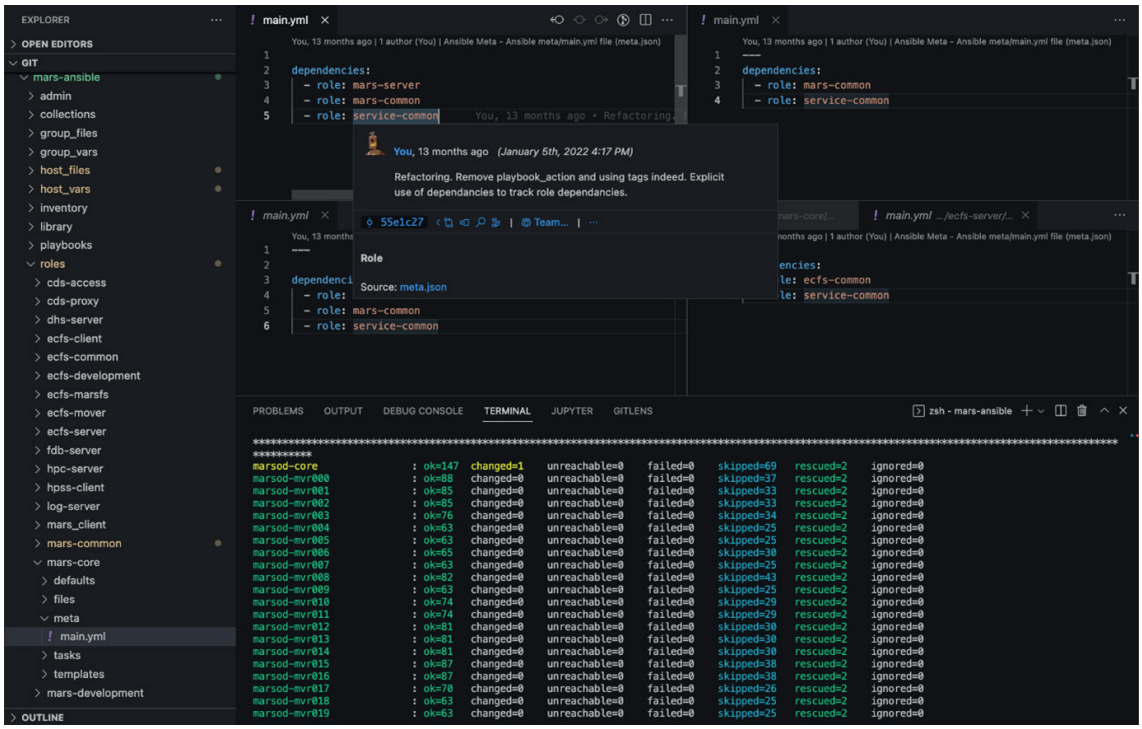


Figure 4 This computer screenshot shows on the left-hand side the available roles composing MARS/FDB/ECFS services and on the right-hand side some examples of the structure of MARS/FDB/ECFS automation and role dependencies.

 doi: 10.21957/sd32f71jlp

Automation deployment structure and the underlying software guiding design principles and architecture turn out to be two sides of the same coin. The close collaboration between software developers and the operation team has been an essential component in this endeavour during the move to Bologna.

Going beyond automation

Automation alone is not sufficient to significantly accelerate the development to production release cycle. DevOps is a methodology that emphasises collaboration and communication between software developers and operations teams to streamline the software development process and improve the delivery of software like MARS, FDB and ECFS. It is a combination of the words ‘development’ and ‘operations’.

Automation and Infrastructure-as-Code (IaC) are both pillars of DevOps. DevOps teams use automation tools to simplify and streamline the software development process, reducing the time and effort required to deploy, test, and release software. IaC is the practice of managing infrastructure using code, rather than manual processes. It involves defining infrastructure elements, such as servers, networks, or storage, as code and then using automation tools to provision and manage those elements. IaC allows DevOps teams to manage infrastructure in a repeatable and scalable way, reducing the risk of configuration errors and increasing the consistency of infrastructure across different environments.

Together, automation and IaC enable DevOps teams to manage software development and infrastructure in a streamlined and efficient way. DevOps seeks to break down silos between development and operations teams and encourages them to work together to ensure that software products are delivered efficiently and effectively. The goal of DevOps is to increase the speed of software development, improve the quality of software, and reduce the risk of errors and downtime.

DevOps also emphasises the use of continuous integration and continuous delivery (CI/CD) practices, which rely heavily on automation. CI/CD pipelines automate the process of building, testing, and deploying software, enabling DevOps teams to release software quickly and reliably.

In advance of the MARS, FDB and ECFS migration to Bologna, ECMWF has been intensifying its DevOps approach, culture of collaboration, automation, and continuous improvement, so that those services could be developed and delivered more quickly, reliably, and securely. Many new developments were necessary to complete the Bologna migration, and a whole new infrastructure was available for us to use. We brought together the traditionally separate functions of software development and operations into a single, integrated approach, working daily together.

Environmental drift, which refers to the configuration of a software environment changing over time, thus deviating from the desired or documented state, can be a major challenge that we wanted to avoid imperatively. This can happen due to various reasons, including manual changes made to the environment, differences in software versions or dependencies, or configuration errors in any slice of the underlying infrastructure or software dependency. In a context where numerous software code changes were made, and where a new data centre was coming to life, avoiding any environmental drift has been paramount to success. By having the environmental drift under control, we protected ourselves against inconsistent testing results, which could lead to wrong conclusions. We typically deployed MARS and ECFS software to different environments (such as development, testing, pre-production, and production environments). Each environment serves a different purpose in the software development lifecycle and has its own configuration and dependencies. Knowing exactly what changes have been made that can explain the behaviour you are observing was key to success. The DevOps approach was essential to achieving that.

The way forward

In preparation of the ECMWF move to the Bologna data centre, significant progress had been made about automation and IaC practices for the development and deployments of MARS/FDB/ECFS services. This modernisation played an important role in ECMWF being able to migrate its 450+ PiB primary data archive without interrupting or delaying its normal operations. This endeavour needs to be sustained and expanded.

We have seen that automation alone is not sufficient to provide a fast development to production cycle capability. Automation must be done using declarative language, in an idempotent (having the same effect being run once or multiple times) and reproducible way. This way it can be a foundation for further practices in which infrastructure is managed and provisioned through version-controlled code, rather than manual processes. Automation alone being the silver bullet of modernised IT practices is a myth.

doi: 10.21957/sd32f71jlp

Future work includes focusing on a seamless approach to MARS, FDB and ECFS deployment by further integrating and orchestrating the different slices of the data infrastructure and their dependencies: provisioning, network, servers' configuration, storage, HPSS. This will consist of a better integration of various classes of existing stress test suites, and of the integration and orchestration in an automation platform (like Ansible Tower) of the steps of our deployment workflow, to get the most out of our CI/CD pipeline.

 doi: 10.21957/sd32f71jlp