Data Centric C2-Services Deployment: an Experiment on Fleets of Military Vehicles

Geert Pingen, Johan van der Geest, Merle Beaujon, Jeroen Voogd and Reinout Pieneman

August 17, 2021

Data-centric C2-Services Deployment: an Experiment on Fleets of Military Vehicles

Geert Pingen*, Johan van der Geest*, Merle Beaujon*, Jeroen Voogd†, and Reinout Pieneman‡
*Monitoring & Control Services, TNO {geert.pingen, johan.vandergeest, merle.beaujon}@tno.nl
†Modelling, Simulation and Gaming, TNO {jeroen.voogd}@tno.nl
‡Human Behaviour & Organisational Innovations, TNO {reinout.pieneman}@tno.nl

***Abstract*—Defense operational and tactical activities are increasingly shifting towards joint, inter-agency, multinational and public (JIMP) partners. In addition, C2 is based on data being generated at an increasing rate by an increasing number of sources in the physical, human and information landscape. Future C2 systems need to cope with this varying number of heterogeneous, distributed knowledge bases using an information services architecture, as well as operational circumstances such as varying connectivity. In this concept paper we therefore propose that a data-centric approach for the enabling infrastructure may provide a better and future-proof basis than a traditional application-centric architecture, which has been a major focal point for IT-design and development over the last decades. Using an experimental data-centric architecture, we aim to understand how a vehicle fleet data processing infrastructure can offer 1) quicker access to the data(sets) users require for developing their services by automated data synchronization and discovery; and 2) provide better insights into service performance by automatically gathering telemetry to generate recommendations for fleet managers.**

***Index Terms*—Data-centric, Multi-domain C2, Federated Cloud, Service Orchestration, Data Synchronization, Kubernetes, KubeFed, Fleet Management**

I. Introduction

Intelligence is of utmost importance to increase the effectiveness of military actions. However, there are three related problems that limit flexibility and efficiency in information provision to a military user.

First, the intel collection process is organized reactively, with the military user mostly looking for the information himself. However, the amount of information that is becoming available is constantly increasing, making it impossible for humans to fully process. The search for information itself can be augmented by services that apply to a more proactive automated push of tailor-made, pertinent information to the military user which is tailored to the operational context [1]. Since it is not known beforehand which information is relevant to which user in a mission, the available data should be available to all authorized users, instead of being locked up in one system or application, as is currently often the case.

Second, the current development and deployment cycle of C2 services is limiting operational effectiveness. For example, new technologies such as AI-powered data processing, take a long time before they are used by existing applications. This sometimes results in slow distribution of important insights gained during a mission to other mission partners. Moreover, developing, deploying, and rolling out updates for C2 services using these insights on the fly is currently not possible within the existing C2 services architectures as these are typically pre-configured, static, monolithic and have a large size.

Third, current C2 systems are not able to cope with a varying number of heterogeneous, distributed information sources to which joint, inter-agency, multinational and public (JIMP) partners can provide information. Future C2 systems therefore should be able to cope with the systems of the coalition partners as well as available other sources of knowledge. The requirement to handle multiple, heterogeneous, distributed data sources in a JIMP context leads to an architecture that allows newly developed C2 systems as well as existing sources to exchange data.

Fourth, future C2 services will need to function in a mission with interrupted network connectivity. These interruptions can come from an operational need to limit electromagnetic signals, but also because radio connectivity can be interrupted by jamming, physical obstacles in a mission area etc.

This concept paper has the following structure: in Section II, we start by describing the technical developments on microservice and containerisation as a starting point for our analysis. Second, Section III describes the specific objectives and experiment approach in building and testing a microservice architecture. We end in Section IV with conclusions and proposed follow up work in Section V.

II. Solution Direction

The proposed solution addresses the four abovementioned problems by making use of these technical trends: data-centricity, microservices, container orchestration, and federated/multi-cluster environments.

A. *Data-centricity*

Over the last few years, a switch from application-centric to a data-centric approach in software development has taken place. Previously, the application-centric architecture has been the major focal point of the IT industry, in which the applications were the primary structural element, supported by middleware solutions to enable inter-connectivity. The disadvantage of the application-centric approach is that data is ‘locked’ in by the application and cannot be easily reused by other applications. The data-centric paradigm puts the data central and an application is just an entity that performs an

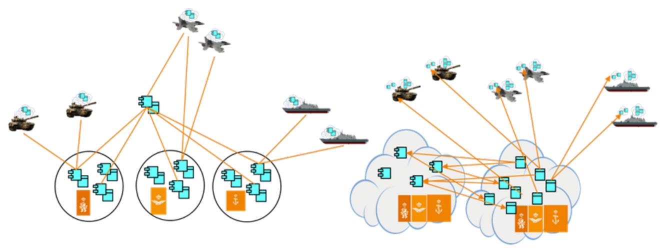
Fig. 1: Comparison between an application-centric (left) and data-centric (right) approach.

operation on some data and produces some new data. All data is stored in data stores accessible via a small number of API's, which make the data re-usable for multiple API's. In addition, data can remain unchanged (e.g. for later analysis) while the API's can update, upgrade or be changed completely. This requires solutions that help to manage microservices in for example schema management, read/write access when using a set of data stores. The accompanying paper [2] describes one way of how microservices can be managed through a service directory and smart connector.

This switch from an application-centric to a data-centric approach is illustrated in Figure 1. On the left side of the figure the Army, Navy and Airforce of a single nation have their own applications, and each application holds its own data. This is aggravated in a JIMP coalition. On the right side of the figure data is, within security limits, available for all applications from the Army, Navy and Airforce.

B. *Microservices*

As described above, the new operational reality forces a change from monolithic applications that keep data hidden from other applications, towards an architecture where data is central and intelligent modules use this data to make transformations towards higher level data suited for human operators. These intelligent modules are not large pieces of software, but instead envisioned as many small modules that have single purposes. Larger functionality is built upon the combination of several smaller modules. The appropriate architectural principle to implement these modules is as microservices. A microservice architecture enforces loosely coupled components in the design of a software application. Each microservice implements a set of narrowly, related functions [3]. In [2] the needed technical infrastructure to build a larger application from microservices is examined.

Using microservices has several advantages over using monolithic applications. First, individual services can (more) easily, and with quicker pace, be extended, updated or replaced. Separate development teams can work on the microservices simultaneously. This is because interfacing with a relatively low number of standardized data stores provides simplicity over interfacing with many application-specific API's. Second, as the number of API's becomes more manageable, it is easier to share data between different parties. Third, testing and deploying the software becomes easier [4]. Fourth, In the Mobile-Dismounted-on-foot conditions it is easier to determine where the data should reside in order to have it available when necessary. This means that some data processing can be performed near the acquisition devices, thus saving precious bandwidth [5]. Note that while use of microservice architecture has benefits for feature development and deployment, it is well known to add complexity and overhead in terms of communication. As microservices are a relatively new phenomenon, it is not yet known under which conditions a data-centric implementation of C2 services using microservices holds the most added value [6].

For example, decentralized acquisition of data is done by a camera mounted on a military vehicle. A microservice running on the IT infrastructure of the vehicle has sufficient computing power for number plate recognition, but not for full image analysis. This microservice thus allows for a quick alarm on the spot. Full analysis of the camera feed to search for related information can be postponed, when a vehicle returns to base where more extensive camera feed processing power is present using a regular video processing application. In practice therefore, we expect that a hybrid architecture with several monolithic applications combined with specific microservices will be the most practical way forward. How and when microservices hold added value for this problem context, is the focus of this research paper.

C. *Container orchestration*

For military users to make full use of the new possibilities of microservices, the management of thousands of services of many vehicles needs to be supported by using container orchestration. More specifically, microservices can be updated instantaneously as new releases become available with bugfixes or feature additions, while military vehicles usually suffer from limited bandwidth or no connection at all during a mission due to enemy jamming. Moreover, in a fleet of vehicles it is very time-consuming to manually check and update each unit individually. Some approach has to be in place to facilitate updating services.

Containerized service deployment technology allows services to be rapidly deployable, scalable and updated frequently. In a network with full connectivity, this allows code to be shipped and deployed quickly when needed on a specific location and/or computing platform. In environments with frequent interruptions in network connectivity this approach helps to enable a faster development and deployment cycle of C2 services in three ways:

First, a containerized microservice deployment approach will help to leverage important insights gained during missions in the field, by enabling the rollout of improved versions of services in a fleet of vehicles based on new mission data. Currently, software technology in military vehicles is at best updated every few years.

Second, this approach helps to share workloads during a mission immediately with other JIMP partners in the network and/or information or data analysts that reside on the military compound, by providing scalability and isolation, and a reproducible runtime environment. For example, when group of vehicles does not have the processing power to analyze a larger video feed, the computation can be containerized and shared with other vehicles of a JIMP partner nearby to speed up the analysis by distributing the workload.

Third, a containerized microservice deployment approach helps to add and remove new C2 services and functions more quickly when the information need of users changes. Currently, C2 systems in military vehicles are monolithic, preconfigured and static; it is not possible to swap out functionality easily when users require new information because of changes in a mission.

D. Federated and multi-cluster environments

To provide functionality for deploying services to edge devices and other compute clusters in mobile platforms (such as a military vehicle) and managing desired state in these varied heterogeneous cluster environments, orchestration over multiple platforms is required. Container orchestration platforms such as Kubernetes enable robustly running and managing isolated, containerized workloads in a common environment with ease.

The wish to communicate and apply control over multiple (geo-distributed) clusters is one that springs to mind quickly. While federated, and multi-cluster environments have been considered and applied for a number of years already, having a standard interface popularized by the ecosystem around Kubernetes allow operators to set up such environments with much more ease. This in turn created a need for new solutions in the multi-cluster space, to solve challenges with e.g. multi-cluster visibility, service discovery, and workload placement, which are the topic of ongoing research. Different approaches to multi-cluster environments can be applied which pose their own challenges, such as centrally managed federated clusters (which is applied in this work), or those that are decentralized, an example of which is proposed in [4].

E. Research questions

Current time-to-act with data gathered in the field can be decreased significantly. In this concept paper two aspects of a data-centric, containerized and knowledge driven architecture are examined to see which technological implementations are suitable.

The first aspect deals with how data can be managed in order to be ready for processing when the end-user has a need for that particular piece of information. The second aspect deals with how a fleet manager can best determine when to update, deploy, delete or upgrade these services over multiple platforms. The implementation of both aspects must take operational circumstances such as limited bandwidth and the loss of connectivity into account. More specifically, we focus on two research questions in particular:

- How to quickly distribute data and or insights obtained during a mission as soon as connection has been established (possibly with limited bandwidth)?
- How to make sure that the services deployed on mobile platforms remain up to date in a situation where services can be updated or added at any moment?

On a technical level these can be translated into a number of more detailed research questions:

- What technology is suited to speed up the development and deployment cycle of C2 services in a large-scale heterogeneous environment of mobile platforms?
- What technology is suited to increase adaptivity of C2 services by faster deployment, easier migration and less overhead?
- What technology is required to have data gathered on missions quickly available as input to other C2 services, and to serve as insight for intelligence officers and developers of C2 services?
- What technology is suited to keep an overview of the state of such a system of systems, and to manage interactions with the system?
- What technology is suited to give insight into performance of such systems in varying conditions?

III. Technology architecture implementation

To this end we have investigated and realised a technical implementation of a data-centric federated architecture as proposed by [1]. We first describe the main technical components, after which we will highlight, and discuss in more detail, the components that specifically enable the two characteristics noted in Section II-E above.

A use case is used to evaluate prototype implementations of the components.

A. Use case

The following scenario is used to outline the proposed solution.

- A vehicle returns to compound after completing its current operation, having collected variety of data (object detections, service metrics, etc.). This is stored in a message bus and in local databases on the vehicle.
- On board, it is running a set of services (e.g. Vehicle Recognition Service (VRS)) previously deployed by the fleet manager.
- Once back at the compound, the vehicle is automatically connected and registered to the compound cluster. Data, services, and telemetry from various sources are synchronized (such that they can serve as a permanent Knowledge Base [2]), allowing data scientists and automated services to identify and develop improvements to the VRS given the current mission profile.
- Additionally, this data is immediately available to military analysts.
- The fleet manager decides the new improved version of the VRS should be deployed over a subset of vehicles and schedules the rollout on his dashboard.
- Reconciliation commences to synchronize the specified vehicles and upgrade their software, readying them for their next task.

B. Technology

Two computing clusters were set up on premise. A main cluster, acting as a central control node, represents the compound in our use-case. A second, smaller edge cluster represents the mobile platform. Kubernetes [7] is selected in this work as container orchestration solution. Originally developed in-house by Google, it is now open-source, maintained by the Cloud Native Computing Foundation [8], and dominates the container orchestration space. As it has the most mature ecosystem and supports the vast number of different workloads that we might encounter in a military environment, it is preferred in this work to alternative solutions such as Docker Swarm [9] or Apache Mesos [10]. Kubernetes follows a master/worker architecture, the full details of which are available at [4]. Moreover, support for edge, multi-cluster and federated architectures are present or well-supported by third party providers, which is a notable requirement of the concept we propose.

In resource-constrained edge environments it is important to be able to deliver orchestration capacities in a lightweight manner, and since we envision mobile platforms hosting their own edge clusters, we require a way to enable federated cluster communication and control. For this purpose, we investigated Kubernetes Cluster Federation (KubeFed), the official cluster federation API-extension [11] for federated resource orchestration, and Rancher's K3s [12] as lightweight Kubernetes edge distribution. Considered alternatives include Rancher Fleet [13] (still in very early stage of development) and KubeEdge [14]. KubeFed takes a more Kubernetes-native approach – being able to federate any Kubernetes resource to a federated kind - and while KubeEdge has (at Kubernetes v1.17) constraints on cluster size, KubeFed theoretically does not have a cap on the number of clusters it supports, which is in line with our use-case. One limiting factor are the etcd storage limits, when objects with references to each cluster (one can imagine CA certs) grow too large for the default maximum etcd object size. Figure 2 gives a high-level overview of this setup.

The edge cluster on the mobile platform consists of a set of base components to provide functionality for connecting to the compound cluster (vehicle daemon), obtaining service metrics (metrics service), and storing service images (OCI registry). It also holds a message bus, modelled after the NGVA [1] concept [15]. Two user applications, the camera- and vehicle recognition service record data and publish them to the message bus. This data is persisted locally on disk. The compound cluster holds components that provide a prototype technical solution to answer our research questions. Most notably, the fleet management services, and the data sync service. We will discuss these components and their evaluation in further detail below. Kubernetes Cluster Federation handles

[1]NGVA is a NATO Standardisation Agreement (STANAG 4754) based on open standards to design and integrate multiple electronic sub-systems onto military vehicles which are controllable from a multi-function crew display and control unit.

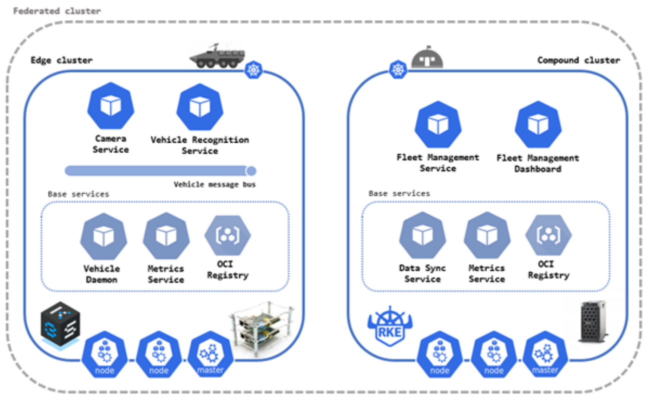

Fig. 2: High level overview of cluster setup with vehicles (edge clusters) and a compound cluster. Listed are the most relevant services introduced in this work. Basic Kubernetes resources such as etcd nodes, Ingress, and DNS, including service-specific resources such as Deployments, PV's, etc. are left out for brevity and clarity.

joining multiple clusters together, allowing for tailored service deployment to individual (vehicle) clusters.

C. Fleet Management

Fleet management encompasses a number of operations we aim to support. A fleet manager should be able to see what vehicles are currently joined to the compound cluster and ready to synchronize; see what versions of C2 services are currently available on the vehicles; be able to manually schedule data synchronization; and schedule new deployments of C2 services to a set of vehicles. This has been implemented as a frontend, together with a backend that exposes an API.

To maintain the current state of the fleet, we decided to implement a heartbeat mechanism, in which the clients (a compact microservice running at the vehicles) repeatedly try to reach the server (the fleet manager running at the compound) by sending a unique token representing the vehicle. These messages will never be sent over the scarce radio bandwidth outside of the compound, but instead only using short-distance communication methods within the compound, like Wi-Fi in our experiment. The problem with a reverse method, where the server is responsible for verifying which vehicles are in the compound, is that the clients are often hidden behind NAT (Network Address Traversal) devices like routers and firewalls, and more often than not, unreachable from the server [16].

Once a vehicle returns to the compound and has network connectivity to the compound cluster, it will not only show up as 'Connected' in the dashboard, but also within the federated Kubernetes cluster. Kubernetes will try to get to a desired state again, by reading the current state of the services running on the vehicle, and then check if there are any changes that need to be applied to get to the desired state. Imagine that a data scientist releases an improved version of the Vehicle Recognition Service. After it has been tested, a fleet manager can decide that this new version is rolled out on a specific vehicle. This vehicle is currently outside the compound, but once it returns, the federated Kubernetes solution will automatically roll-out this new version.

D. Data Synchronization

Synchronizing data from vehicles that connect to the compound is managed by the data sync service. When a vehicle joins the federated cluster, initiated by the daemon running on the vehicle, a request is sent to the data sync API with information on the vehicle's identity. After identity validation, the data sync service queries the edge cluster for services to synchronize (it may not be required to synchronize all data) and generates security credentials. It then uses Kubernetes Cluster Federation to spawn a sync job at the edge cluster with these credentials and information regarding the data to be synchronized. This allows the sync job to locate the data and synchronize securely to the compound cluster. An overview of this process is given in Figure 3 below.

We opted for low-level Kubernetes-native solutions to these challenges, instead of wrapping built-in synchronization provided by e.g. well-known object stores, since we cannot rely on those being present on all mobile platforms. Data synchronization is implemented using Volume Cloning [17], which is available for dynamic storage provisioners (in theory nothing should prevent cloning non-dynamic Persistent Volume (PV) as long as both Persistent Volume Claims (PVCs) request the same storage class and regular dynamic provisioning works, but this has not been validated). This allows us to create new volumes from existing volumes, and enables the use of non-RWX (ReadWriteMany) PVs (something that is prevented by having the synchronization job claiming the actual PV used by the application to be synced), at the cost of temporary data duplication.

We ran functional tests to validate the synchronization mechanism and service components in a lab environment by running a database application on the edge cluster with PV mounted to a local disk. On sending a request to the data sync API, we observed credentials being generated; the synchronization job being spawned on the edge cluster; and the data being synchronized correctly to the compound cluster – in normal situations as well as during interrupts. We aim to validate this behaviour further in a setting with live vehicles in future work, given multiple data sources and in collaboration with SME's to provide feedback.

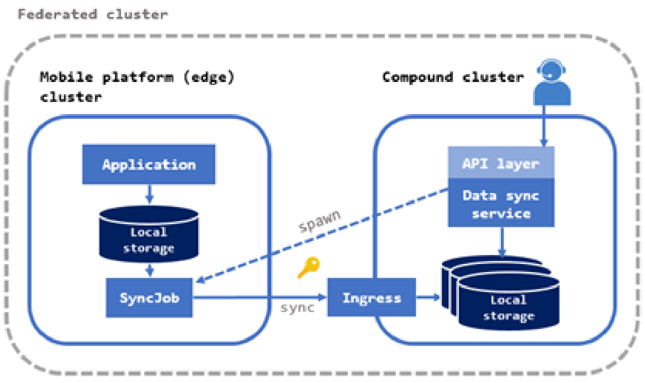

Fig. 3: Technical overview for data sync service in a federated/multi-cluster environment. A sync job is spawned on the edge cluster, either manually by the fleet manager or automatically when connection is established with the vehicle. This job contains the necessary information to locate relevant application data as well as security credentials to obtain restricted access to the compound cluster. On completion of the synchronization the environment is sanitized, and the fleet manager is notified of newly available insights.

IV. Conclusion and Discussion

In this concept paper, we have described how current technological advancements in the cloud (native) domain can help alleviate many pains currently experienced within the realm of military vehicle fleets, and realized this in a technical, data-centric, federated solution architecture. It provides more flexibility and efficiency in future operations by providing information to military users in a timely manner, and ease of managing C2 services. More specifically, it can be stated that the individually implemented architectural elements work as intended within the limited scenario used. The used technology is largely based on open source (cloud native) solutions and provide a stable basis for further experimentation. While it is a relatively mature technology, it is not suggested that this exact implementation of the described technology should be used in later implementations. The developments described in this paper are used to derive requirements that need to be met by tailored solutions.

The fleet management and data synchronization components, together with our proposed federated architecture, provide solutions to our two main research questions in the context of C2 service management across a fleet of platforms: how can data be quickly distributed as soon as connection to a mobile platform has been established; and how to ensure that deployed services remain up to date. In particular, we described which - and provided an implementation of – technology that is suited to speed up the development and deployment cycle of C2 services and increase adaptivity; and developed solutions to augment the availability of mission data to analysts/developers, and to maintain an overview of the state of such a system of systems. While we tested our approach in a lab environment, in a next step we will validate the architecture and individual components in a larger, live experiment with SME's.

We observe that there exist limiting factors to operational effectiveness, flexibility, and efficiency in information provision to military users. These stem in part from the rigid C2 IT architecture and constrained data availability. In this work we present an alternative approach, that incorporates aspects of modern cloud native software development and open source cloud technology, to show the benefits of a highly-automated federated infrastructure in the C2 architecture space.

V. Future work and next steps

The research objective and experiment setup provide the opportunity for a large variety of other interesting research objectives. In future work we will assess the current technical implementation in a live experiment with multiple mobile platforms together with SME's. Other future research objectives include the assessment of security challenges in a federated multi-cluster environment; the investigation of cloud-native tools in the space of data exploration and manipulation to shorten the C2 service development cycle; the generation of deployment recommendations for fleet managers; connecting C2 service development to simulation services to determine fit-for-purpose; implementation in existing battle management systems; and the support of peer-to-peer service updates in an autonomous vehicle use-case.

References

[1] R. Pieneman, M. Schenk, B. Gerrits, and C. van den Broek, "Data centric information provision: an outline," *ICCRTS*, 2018.

[2] J. Verhoosel, B. Nouwt, and J. Voogd, "Data-driven, Service-Oriented, Knowledge-based Command and Control," *ICCRTS*, 2020.

[3] S. G. Du, J. W. Lee, and K. Kim, "Proposal of GRPC as a new northbound API for application layer communication efficiency in SDN," in *Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication*, 2018, pp. 1–6.

[4] J. van der Geest, C. van den Broek, H. Bastiaansen, and M. Schenk, "Enabling a Big Data and AI Infrastructure with a Data Centric and Microservice Approach: Challenges and Developments," in *NATO STB IST-160 Specialists' Meeting*, 2018.

[5] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of big data*, vol. 2, no. 1, pp. 1–21, 2015.

[6] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 524–531.

[7] Kubernetes. [Online]. Available: https://kubernetes.io

[8] Cloud Native Computing Foundation. [Online]. Available: https://www.cncf.io

[9] Docker Swarm. [Online]. Available: https://docs.docker.com/engine/swarm/

[10] Apache Mesos. [Online]. Available: http://mesos.apache.org

[11] KubeFed. [Online]. Available: https://github.com/kubernetes-sigs/kubefed

[12] Rancher. K3s - Lightweight Kubernetes. [Online]. Available: https://rancher.com/docs/k3s/latest/en/

[13] Rancher Fleet. [Online]. Available: https://github.com/rancher/fleet

[14] KubeEdge. [Online]. Available: https://kubeedge.io/en/

[15] STANAG, NATO, "4754: NATO Generic Vehicle Architecture (NGVA) for Land Systems."

[16] Infrastructures.org. Bootstrapping an Infrastructure. [Online]. Available: http://www.infrastructures.org/papers/bootstrap/bootstrap.html

[17] Kubernetes. Introducing Volume Cloning Alpha for Kubernetes. [Online]. Available: https://kubernetes.io/blog/2019/06/21/introducing-volume-cloning-alpha-for-kubernetes