NATO Federated Coalition Cloud with Kubernetes: A National Prototype Perspective

Hallvard M. Andersen, Borgar F. Flytør, Emil Onsøyen
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

Even T. Holmen, Niklas Sølvberg
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

Thomas Haugan, Oskar Gabrielsen
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

Frank T. Johnsen
Norwegian Defence Research Establishment (FFI)
Kjeller, Norway

***Abstract*—In NATO, the IST-168 research task group (RTG) "Adaptive Information Processing and Distribution To Support Command and Control" aims to investigate cloud computing at the tactical edge from a coalition force perspective. From recent developments in cloud computing and DevSecOps stemming from civilian use, but also with increasing adoption for military applications, virtualization of infrastructure and containerization of software is currently a preferred approach to cloud computing. Containers provide a lightweight approach to packaging and orchestrating software components and services. Kubernetes is a tool for orchestrating and managing deployment, scaling and migration of such containers.**

In the NATO IST-168 RTG, the emphasis is on interoperability and interconnecting different nations' Kubernetes clusters. The work described in this paper presents Norway's contribution to the RTG.

I. Introduction

The term *cloud computing* is not necessarily well defined, and so different people may attribute different meanings to the concept of "the cloud". For the sake of this paper, we use the definition of cloud computing from the National Institute of Standards and Technology (NIST), which describes cloud computing as follows [1]:

> A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

In other words, cloud computing can be seen as a modern and effective approach to deploying, governing, using, and maintaining Communication and Information Systems (CIS) resources.

The cloud computing approach has been gaining momentum for civilian applications for several years now. Likewise, NATO is aware of cloud computing benefits and looking to exploit it operationally. Given that the benefits of cloud can be applied also to military applications, this could potentially enable more effective operations in the future. For NATO, the prospect of more stable, more effective CIS for coalition forces could be an enabler for more agile operations in the future, especially with respect to interoperability and rapid deployment.

The nature of military operations introduce new challenges to cloud computing that are usually not seen in civilian applications, notably the notion of CIS services, and the *vision of cloud services, at the tactical edge* [2]. Simply put, "the tactical edge is where users operate in certain environments that are constrained by such things as limited communications connectivity and limited storage availability [3]". In NATO, the IST-168 research task group (RTG) "Adaptive Information Processing and Distribution To Support Command and Control" (IST-168) aims to investigate Kubernetes from a coalition force perspective, with emphasis on interoperability and interconnecting different nations' Kubernetes clusters at the tactical edge [4]. IST-168 anticipates Kubernetes to be a prominent building block of future NATO CIS in coalition operations, and so it is an important research question to investigate if it is possible to achieve interoperability across clusters owned by different nations.

The work described in this paper has been performed as part of the Norwegian contribution to the IST-168 work. Here, we provide our perspective on setting up Kubernetes, and taking part in IST-168 forming a multi-national cluster, as seen from the viewpoint of one participating nation in this experimentally-oriented RTG.

II. Related work

The vision of network-centric operations is to increase operational capabilities through networked collaboration. Services are essential in *service-oriented architecture*[1] and, more

[1]SOA, or service-oriented architecture, defines a way to make software components reusable and interoperable via service interfaces. Services use common interface standards and an architectural pattern so they can be rapidly incorporated into new applications. This removes tasks from the application developer who previously redeveloped or duplicated existing functionality or had to know how to connect or provide interoperability with existing functions [5].

recently, *cloud-based architecture*[2], where CIS services can be accessed at all times by very large ranges of consumers independently of where the consumers are [7].

Kubernetes is an open source system for automating application deployment, scaling, and management [8]. Kubernetes can be used to ensure resilience and scalability for services and end-systems, properties that are essential to both civilian and military systems. It is often used to deploy services when building modern CIS, using agile *DevSecOps*[3] methodology. A prominent example of this effect is the US Department of Defense (DOD) Enterprise DevSecOps Initiative, where Kubernetes is an important component [10]. In NATO, IST-168 aims to investigate Kubernetes from a coalition force perspective, with emphasis on interoperability and interconnecting different nations' Kubernetes clusters at the tactical edge [4]. This approach is referred to as *federated cloud architecture* in IST-168, a concept involving Kubernetes and defining interoperable services and open, well-defined Application Programming Interfaces (APIs). Think about this a military refinement of the concept of a civilian *Kubnernetes federation* [11] where all clusters participate equally and share all resources — in IST-168 we have the addition of nationally sovereign clusters, and fine-grained control of which services and resources to share with partners.

The motivation behind investigating a federated cloud architecture for military systems is because it is crucial for mission partners to have sovereignty over the infrastructure they provide and over the data and services they host [12]. This is due to strong security and classification policies. It includes individual policies on how their clouds expose services and capabilities to other nations. As well, it is each nation's own choice as to the type of hardware and supplier to use. Hence, the architecture must ensure functional interoperability and compatibility of partner clouds. Therefore, mainstream cloud container technology is used. An industry standard for containers was adopted, e.g., Open Container Initiative (OCI) [13], and uses Kubernetes along with its large, rapidly growing ecosystem [14].

Furthermore, the discovery of other nations' cloud should be peer-to-peer based, as this eliminates the need for a central authority that all nations need to trust. Discovery in this respect is not a part of Kubernetes, and so IST-168 has been working on defining a service discovery API, which allows for rapid experimentation and deployment, supporting a multi-national mission context like, e.g., in Federated Mission Networking (FMN) [15].

Finally, the federated cloud infrastructure must be able to operate in mission contexts with disadvantaged networks, including the interactions and communications for inter-cloud connectivity. To provide functionality for deploying services to edge devices and other compute clusters in mobile platforms (such as a military vehicle) and managing the desired state in these varied heterogeneous cluster environments, orchestration over multiple platforms is required [16]. Thus part of the IST-168 research examines whether, and to what extent, individual clouds may span multiple nodes (e.g., vehicles or even dismounted soldiers) in which case the disadvantaged network condition also holds for intra-cloud connectivity [17].

So, in summary, the overall technical goal of IST-168 is to have several different nations install and operate their own Kubernetes clusters, that can be interconnected to discover and share resources in a federation of systems. This paper provides the authors' perspective on participating in building and testing such a multi-national federated system.

III. Operational perspective and scenario

In IST-168, the member nations looked into cloud computing from a NATO perspective, i.e., a federation of stand-alone national installations, and investigating how cloud computing resources hosted by such different nations may be leveraged in operational contexts. The background is that improved sensing and mobile computing capabilities may lead to ever more information and processing power being available in mission contexts. In modern military operations, the new information sources like military use of the Internet of Things (IoT) [18] and overall improved sensing capabilities lead to a lot of data that needs to be processed. Ideally, this data should be processed near where it originated to rapidly benefit the decision maker, so as not to have to transport it to a centralized data center somewhere for processing. Ideally, we want to be able to leverage mobile computing capabilities, which again may lead to more information becoming available more rapidly. To achieve this, an important aspect of *information superiority*[4] is having agile (including such properties as robust, adaptable, capable) CIS systems. We think that a federated cloud architecture is needed to maintain information superiority in future NATO coalition operations.

Naturally, the use of cloud approaches should not be limited only to coalition operations. As our national military CIS systems evolve from old stove-piped monoliths towards microservices built using modern DevSecOps best practices, we are rapidly moving towards a future where we can anticipate that cloud-based architecture and cloud-native CIS systems become the norm. The increased complexity of modern CIS systems due to microservices and possibly decentralization across multiple physical locations leads to a potentially less *efficient*[5] implementation than the old stove-pipes. But, the gains in system resilience and the other desirable aspects of

[2]Microservices (or microservices architecture) are a cloud native architectural approach in which a single application is composed of many loosely coupled and independently deployable smaller components, or services [6]. Simply put, microservices can be seen as the evolution of SOA, towards smaller autonomous services that lend themselves well to independent deployment and scaling using cloud computing.

[3]DevSecOps is the industry best practice for rapid, secure software development. DevSecOps is an organizational software engineering culture and practice that aims at unifying software development (Dev), security (Sec) and operations (Ops) [9].

[4]Information Superiority is the operational advantage derived from the ability to collect, process, and disseminate an uninterrupted flow of information while exploiting or denying an adversary's ability to do the same [19].

[5]Efficient (adj.) — Performing or functioning in the best possible manner with the least waste of time and effort.

cloud computing, means that this is a more *effective*[6] way of building, deploying, and maintaining CIS capabilities than previous architectural approaches.

Cloud computing is not limited to any specific operational scenario, and so we think it is equally suited to supporting all types of military CIS systems and needs. Specifically, the IST-168 group has been looking at both land-based and maritime scenarios [12]. Cloud computing encompasses a collection of technologies, offering unprecedented flexibility and scalability, and a need for new business models to take advantage of these new opportunities. Hence, the expected benefits of cloud services include cost savings, improved scalability, improved security and resilience to system failure, when combined with modern CIS using cloud-based architecture and built and maintained through DevSecOps.

We anticipate being able to reap similar benefits for military CIS and operations. Potential benefits to CIS include enabling more balanced and efficient utilization of resources by off-loading computations to more resource-capable nodes. This in turn may lead to lowered demands on the disadvantaged tactical network, by performing data processing close to the data source. Finally, what we think perhaps the single most important property of cloud computing: Improved resilience through adaptive cloud deployment, allowing service redundancy and geographic distribution for increased CIS resilience.

Benefits to military operations include increased availability of information to all echelons. Through better use of available resources, we make the most of resource-constrained and communication-constrained nodes. Further, this may lead to a reduction in data overload on the soldiers, by converting voluminous raw data into smaller information updates. Also, bringing cloud computing capabilities into the field and nearer to the soldiers allows more rapid processing and use of available information.

For the sake of this paper, we will concentrate on a land-based scenario, which is a subset of the *Anglova Scenario* [20], [21] as developed by IST-124 "Heterogeneous Tactical Networks Improving Connectivity and Network Efficiency" and released to the public domain. For those who don't know the Anglova Scenario, it consists of three vignettes:

1) Intelligence preparation of the battlefield
2) Deployment of coalition forces and surveillance
3) Urban operation, including insurgent defeat and MEDEVAC.

Figure 1 gives an overview of the scenario. In particular, we use vignette three — urban operation — as the frame for the work in this paper. In short, this vignette provides a set of sensors within the city, a number of dismounted soldiers (resource constrained), vehicles with more capable resources (yet still resource constrained), and finally the operations center (plenty of resources). Overall, communications are limited due to the urban environment and node movement.

[6]Effective (adj.) — Adequate to accomplish a purpose; producing the intended or expected result.

Figure 1: Anglova scenario overview.

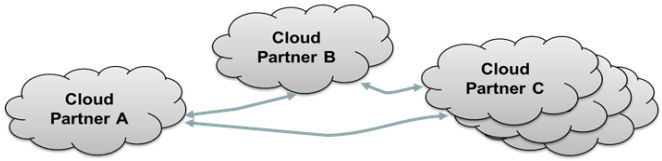

Figure 2: High-level overview: Federated cloud architecture.

For the sake of this paper, we consider a NATO coalition operation, involving multiple partner nations, Norway included. Each partner has its own cloud, consisting of one or more Kubernetes clusters, where the partner clouds are interconnected in a federation. Realizing the federated cloud architecture, the idea is that there should be no central instance (so no single point of failure). In the federation, each partner hosts its own services and exposes a subset of its services to other partners, furthermore, it can allow other partners to use its resources.

The idea is that with standard approaches and standard APIs, we can achieve such inter-cloud interoperability. Figure 2 illustrates three partner clouds, autonomous on their own, but with the capability to interconnect. Recall that how each partner realizes their particular cloud (and handles intra-cloud communication) is left to each sovereign nation to decide and handle. The need for agreement and standardization is limited to the inter-cloud communication, illustrated by the arrows between the clouds.

IV. Technology and Building the National Cluster

Kubernetes takes the so-called Cloud Native Security approach towards its security policies [22]. At its core, Cloud Native Security describes the 4 C's: Cloud, Cluster, Container, and Code. Each describes a layer on which security measures

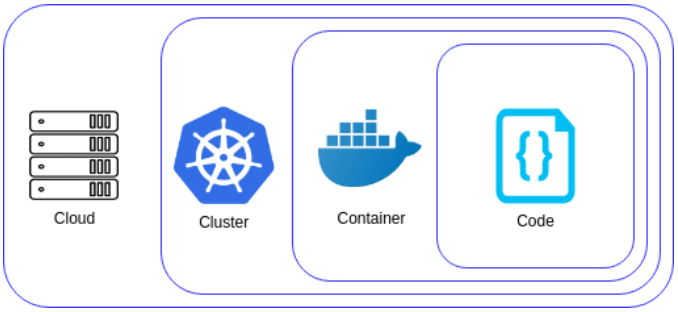

Figure 3: The 4 C's of Cloud Native Security (adapted from [22]).

are taken and which the next layer of security builds on, shown in Figure 3.

The cloud layer describes measures that target the actual infrastructure the cluster runs on, where the underlying hardware and bare-metal hypervisors are the defining factors of what constitutes best security practices. Drawing the parallel to a federated cloud architecture, this part is left to each nation to choose as they see fit.

The cluster layer is concerned with two primary objectives; securing the cluster components that are configurable, and securing the applications that run on the cluster. Here, underlying hardware and future applications will vary, both in terms of vendor, provider, and developer. For the sake of IST-168 federated cloud architecture, Kubernetes was chosen as the Cluster technology so that we had a common agreed-upon approach that could feature interoperability across clouds.

Finally, the container and code layers describe application run time and implementation, respectively. These layers largely depend on the type of containers being used and the implementation by the developer. Probably the most used container technology today is Docker, so we have chosen that to package code in our cluster. The container/code layers are typically the two layers that can be considered part of the DevSecOps methodology, whereas the cloud and cluster layers can be considered infrastructure components.

A. Docker

Docker is a platform for building container images containing various pieces of applications and services [23]. The platform offers multiple ways for creating the images, either by using automatic processes built into Docker or by building it yourself.

A container is a sandboxed environment for running self-sustained applications and services. The Docker images are built using the containers. Similarly to containers, images are self-sustained sandboxed applications or services. After they have been built, they can run on any platform that supports either Docker or Kubernetes. The self-sustainability also means that the images are fully portable between different operating systems and across multiple clouds.

The Docker runtime environment consists of various parts, the main component being the Docker Engine, a software component that hosts the containers. The Docker Daemon is the component that takes requests from the Docker API and manages all images and containers. It also keeps control of networks and volumes. When using Docker on a personal computer (i.e., Windows or Mac), the Docker Desktop application contains all the parts of Docker and gives a one-in-all experience. For a machine running Linux, Docker Desktop is unavailable. Instead, images run directly on the Docker Engine.

B. Kubernetes

Recall that Kubernetes, also known as K8s, is an open source system for automating deployment, scaling, and management of containerized applications on a cluster.

Kubernetes is used for managing machines in a cluster by providing a framework for organizing and deploying various applications and services, requiring a wide variety of different resources [24]. It can run any type of application and service, due to its extensive usage of sandboxes for virtualization and containerization. A Kubernetes cluster can be organized into multiple levels of hierarchy. At the top of the hierarchy sits the cluster itself, which is organized into one or more nodes.

When multiple nodes are configured in the cluster, they can be split into one master node and several worker nodes. Kubernetes nodes function like virtual machines and control the resources used by the various applications and services running on each node. The master node also referred to as the control plane, is the component in charge of deploying the various applications and services. It also defines, manages and distributes the workload across the worker nodes. In order to run applications and services in nodes, the applications and services will be packed into pods running on each node.

Kubernetes pods are *"the smallest deployable units of computing that you can create and manage in Kubernetes"* [25]. In the Kubernetes cluster, the pods are used for storing and running the applications and services in containers. While pods work as self-sustained environments and are generally isolated from the rest of the cluster, they maintain the ability to communicate and exchange data with other pods. A pod can consist of one or more containers, but it is limited by node size and may not be fragmented between multiple nodes.

Kubernetes ships with a variety of built-in tools for automatic deployment, scaling, and management of containers. Docker is the most common platform for creating containers for a Kubernetes environment.

C. Kubernetes and Docker architecture

The bottom software layer of the architecture consists of a Kubernetes cluster [26]. The Kubernetes cluster is situated on top of the operating system and is used to support the various applications and services that were needed for the project. We had five equal physical machines available for this project. These five, all running Ubuntu 18.04.5 LTS, were named Ubuntu 1 through 5, called U1 ... U5 for short. Specifications are as follows:

- HP ELITEDESK 800 G3 SFF

- 2x HDD of 1TB each (one for operating system and runtime libraries and other software, the other for storage)
- 16GB RAM / 2GB SWAP
- Intel i7-6700 CPU @ 3.40GHz (8 CPU cores)

For this project, the Kubernetes cluster has been set up with three nodes, one master node and two worker nodes, each residing on a physical machine deployed at the Norwegian Defence Research Establishment (FFI). The remaining two computers were used as a backup in case anything would happen to the other nodes.

The nodes were created by connecting remotely into the Linux environment at FFI, and building the cluster using Bash and YAML. Each node was built to the same standards using the same script.

The architecture of Docker consists of the containers and Docker images that were built for the project.

Figure 4 shows the Kubernetes setup of all IST-168 clusters.

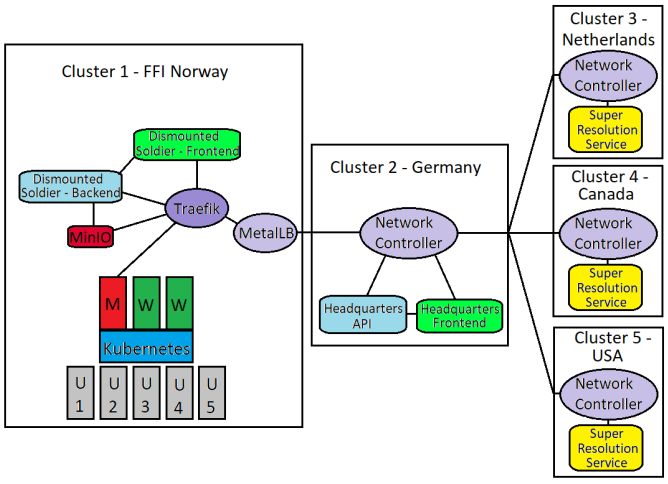

Figure 4: Overview of the collaborative cluster architecture.

D. Network connectivity

Network traffic targeting the Kubernetes cluster must be directed to its correct endpoint. Services deployed in Kubernetes are commonly exposed to the outside world by relying on a DNS server, hosted by the organization providing the underlying hardware and network infrastructure. Thus, configuring the cluster with proper network capabilities is the first step to creating an interface that allows plug-and-play between collaborating Kubernetes clusters.

Interconnecting clouds or additional external services relies on the various network communication protocols that make up the Internet. Thus, the cluster constructed during this project is principally able to exchange data with external Kubernetes clusters and services. This was a major technical aspect to explore, i.e., the feasibility of the federated cloud architecture as envisioned by IST-168.

In our setup, due to security policy reasons, no DNS services were available, and the standard HTTP/HTTPS ports were not permitted. Moreover, the cluster was running on plain Ubuntu 18.04.5 LTS computers without any supporting commercial cloud platform (AWS, Azure, GCP...). These restrictions presented some additional configuration work to be conducted.

User traffic entering the Kubernetes system was processed by a network load-balancer that routed the incoming query to a specific service. Kubernetes on bare metal clusters are quite rare thus the provided network balancers will remain pending without support from a cloud provider. This was resolved by using MetalLB [27], which provides the necessary network load balancing. MetalLB is provided as a Docker image but must be customized according to the environment it is used. The most important features to specify is the protocol and pool of available IP addresses.

Any Kubernetes internal load balancer requesting an IP address will be assigned one of the addresses listed in the MetalLB pool. Figure 5 illustrates how the Kubernetes ingress controller manages the cluster IP addresses.

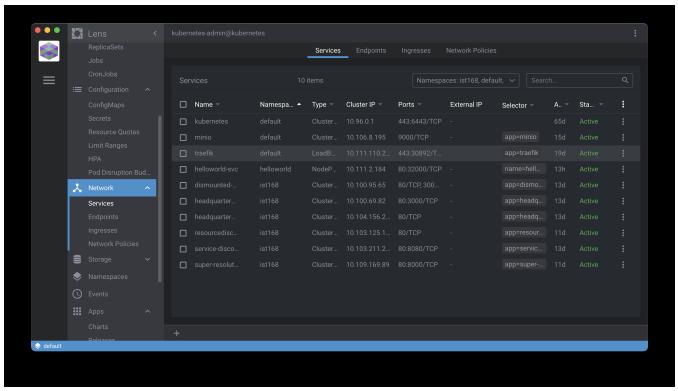

Figure 5: MetalLB manages cluster IP addresses in Lens.

E. Ingress control

In Kubernetes, a deployed microservice is referred to as a pod, and operational pods communicate via a virtual pod network managed automatically by Kubernetes. While pods operate on an internal network that is not accessible by the outside world, this is solved by linking a service to the deployment. In Kubernetes, a service is an abstract way of exposing applications by defining logical sets of pods and policies on how to reach these. The most common service types are:

- ClusterIP: This is the default ServiceType and exposes the service onto a cluster-internal IP address, which makes the service only reachable from within the cluster.
- NodePort: Exposes the service on each Node's IP at a static port number. A ClusterIP service, to which the NodePort service routes, is automatically created. The NodePort service is reachable from outside the cluster, by requesting <NodeIP>:<NodePort>. The NodePort service is a good option to use during the development stage and provided that sufficient amounts of ports are available.
- LoadBalancer: Exposes the service externally using the cloud provider's load balancer. NodePort and ClusterIP

services, to which the external load balancer routes, are automatically created.

In production grade systems hosting a great number of services, it is preferred to use an ingress controller to manage the network traffic between the outside world and the internal services. An ingress controller acts as the entry point for the cluster and consolidates routing rules into a single resource as it can expose multiple services under the same IP address. The ingress controller is not part of the Kube-controller-manager binary, meaning unlike many other built-in mandatory Kubernetes controllers, it must be supplied and configured explicitly by the user. Because Kubernetes clusters can be situated in a wide variety of infrastructures and run-time environments, a great number of third-party ingress controllers are supported in it. Each implementation has its advantages and disadvantages and must be carefully selected according to the given requirements. Subsequently, ingress controllers are versatile and match well with the dynamic nature of Kubernetes clusters, but can be demanding to configure. An ingress resource provides a set of traffic rules defining how exposed HTTP and HTTPS routes from outside the Kubernetes cluster shall be mapped to services within the cluster. An ingress can be configured to give services externally-reachable URLs, load balance inbound network traffic, rewrite target URL and terminate SSL/TLS connections. Every rule is formulated in a standard pattern:

- Hostname (optional). If a host is listed the rules apply to that host. When no host is defined the rule applies to all inbound HTTP traffic through the IP address specified.
- List of subpaths, each with associated backend defined with a service name, port name and port number. Host and path must match the URL of an incoming request before the load balancer directs traffic to the referenced service.
- Backend is a combination of service and port names. HTTP/HTTPS requests to the ingress that match the rule are routed to the listed backend. The matching rules can be formulated into complex expressions enabling very specific network routing.

An ingress controller is responsible for executing any ingress submitted to the Kubernetes API, usually in tandem with a load balancer. As previously mentioned, this project utilized a bare-metal Kubernetes cluster, thus adding the MetalLB load balancer was required to connect the cluster towards the FFI edge router. When deploying the ingress controller it will automatically be assigned an IP address by MetalLB.

For this project, *Trœfik*[7] was selected as the ingress controller as it was already being used with great success by the international partners.

F. Service and Resource Discovery APIs

The Service and Resource Discoveries APIs are developed and shared internally in IST-168. They are needed to obtain

[7]Træfik (aka Traefik) is a popular feature-rich open source Edge Router that is natively compliant with every major cluster technology [28].

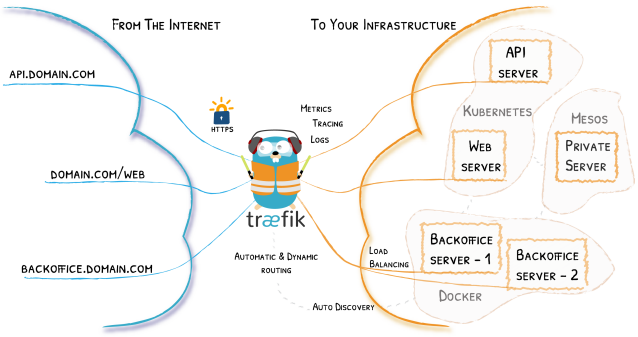

Figure 6: Architecture of the Træfik ingress controller [28].

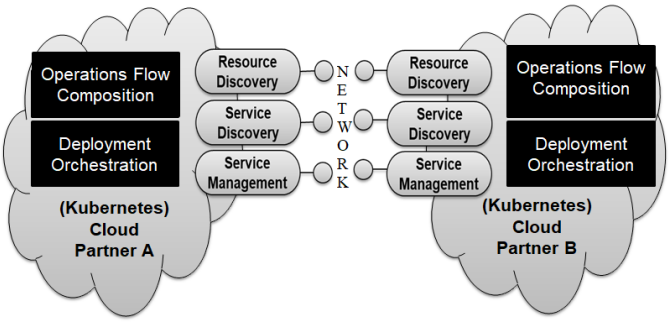

Figure 7: Detailed overview: Federated cloud architecture showing two partner clouds (adapted from [12]).

a cooperative and coordinated federated cloud, because no such standardized APIs currently exist. The Kubernetes cluster running the Deployment Orchestrator queries the other Kubernetes clusters, it searches and maintains an overview of the available services and resources provided in the federation.

The Resource and Service Discovery APIs respond by returning metadata in JSON format back to the central unit. The Resource Discovery analyses the hardware capabilities of the cluster, e.g., CPU, GPU, RAM and storage as well as the inbound/outbound network connectivity to other Kubernetes clusters. Equally important is the Service Discovery in which each cluster can report its available services. Each service is identified with a universally unique identifier (UUID) along with additional metadata that describe the service.

Recall the high-level architecture shown in Figure 2, where the arrows allude to the need for standardized APIs. Following the APIs discussed above, the more elaborate architectural overview becomes as shown in Figure 7.

G. Deployment Orchestrator

The Deployment Orchestrator was provided by IST-168 international partners. It collects and maintains an overview of the federated cloud. By querying the Service and Resource Discovery APIs of each contributing cluster, it can use this accumulated metadata to compute metrics, health checking, orchestration of services and perform custom ranking and optimization algorithms based on factors like computational power, storage capacity or quality of network connectivity.

The Deployment Orchestrator is one of the key elements to manage the interconnection of several Kubernetes clusters.

V. Functional Services

Recall that we are basing our work on the Anglova Scenario vignette 3: Urban operation. Specifically, the case we're using to show the feasibility of the federated cloud architecture, is as follows:

Dismounted soldiers on patrol want to report a suspicious incident. A soldier makes a photo of a car speeding away and reports it to the HQ for further analysis. Receiving the information, the HQ decides upon follow-up actions requiring processing intensive image analysis: The image needs to be enhanced, and so a Super Resolution Service (SRS) needs to be employed to be able to discern the vehicle's license plate. Different nations taking part in the operation offer different services, and so there is a need to employ the federated cloud architecture to complete the information analysis. All partners (for this test: NOR, DEU, NLD, USA, CAN) have deployed the following common services:

- Resource Discovery API
- Service Discovery API
- Object Storage Service

In addition, different partners offer the following specific capabilities in their clouds:

- NOR: Dismounted soldier system
- DEU: HQ C2-system, Deployment Orchestrator
- NLD, USA, CAN: SRS

With the aforementioned urban tactical edge scenario in mind, applications were made for the dismounted soldiers and the headquarters. The soldier application allows the dismounted soldier to create a SPOT report containing an image or a video, and upload it to the HQ. The HQ application is able to view the SPOT report and invoke the SRS which enhances the image/video sent by the soldier. The steps involved are further shown in Figure 8.

A. Dismounted Soldier

The Dismounted Soldier application consists of a simple frontend and a database instance. The frontend is a webpage with file upload and data inputs which makes the SPOT report. Figure 9 shows the frontend GUI. The SPOT report generated on this webpage includes the activity, the location, and the time and date from when the image or video was captured. After a SPOT report has successfully been created, it can be transferred to the Headquarters application. The Dismounted Soldier required a database to store the image or video files uploaded from the frontend. A MinIO database was recommended for use by the IST-168 group, as a file storage system was the only database needed for this project. This database was created as a Docker image from a prebuilt image supplied by MinIO, this database realized the object storage service [29]. Both the MinIO storage instance and the frontend are hosted on the Norwegian cluster.

Figure 8: Scenario case: Step by step.

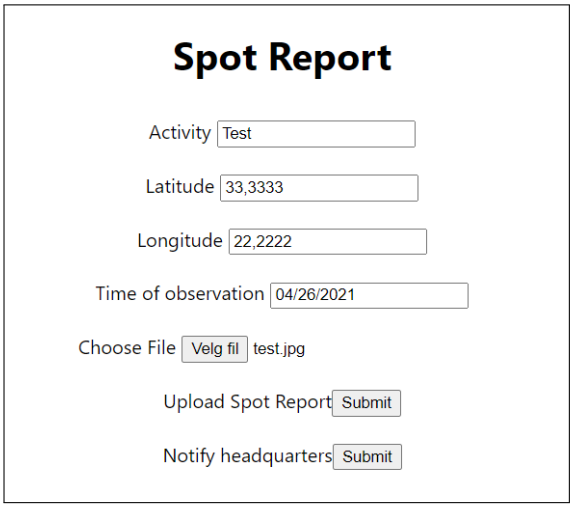

Figure 9: The Dismounted Soldier frontend.

B. Headquarters - Frontend

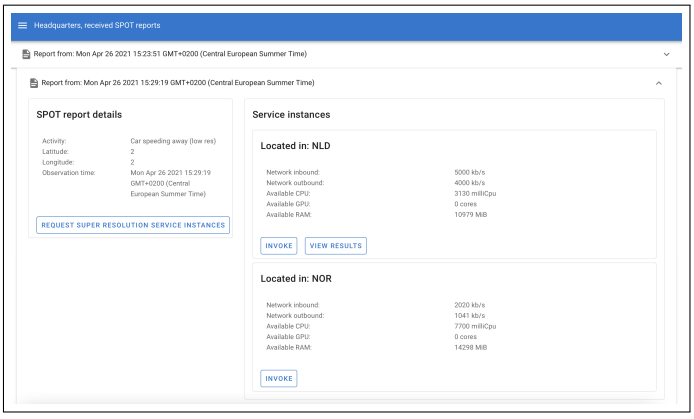

Figure 10: The Headquarters frontend running on the German cluster.

The Headquarters application, shown in Figure 10, contains a list of SPOT reports which can be selected for a detailed view. The HQ can invoke a SRS on any selected report image or video. By requesting the SRS, the service discovery API displays available instances of the SRS and their available

resources. The SRS upscales the resolution of the image as shown in Figure 11, and adds this to the SPOT report. The Headquarters frontend and API is hosted on the German cluster.

Figure 11: Using SRS to increase image resolution. Original photo (left) and enhanced version (right).

C. Implementing ingress controller and ancillary services

A fully working IST-168 cluster requires a minimum of ancillary services provided either by third-party entities, as well as a number of pre-existing custom micro-services developed and provided by the IST-168 software developers.

The software components are readily available from Docker image registries hosted by the corresponding parties and can be deployed onto a Kubernetes cluster by running the associated YAML files.

Many of the YAML files typically required a great deal of customization as each Kubernetes cluster often have its unique properties in terms of networking and run time environment.

The MetalLB load balancer and Træfik ingress controller are third-party software and were definitely the components requiring the most customization, thus quite some time was needed for trial and error.

The core of the SRS providing the artificial intelligence-based image enhancement was implemented by IBM [30], as shown in Figure 11, while an outer software layer was added to offer an API compliant with the Kubernetes clusters.

VI. Testing the cluster and Testing Strategy

Testing the cluster has been done by first testing applications and containers on virtual machines running on personal devices before deploying them at the hardware stationed at FFI. This ensures one proper layer of testing where deployment errors can be addressed before exposing the hardware.

Manually testing the nodes can be done by checking their status with the following command:

```
$ sudo kubectl get nodes
```

The results are shown in Figure 12.

```
sigsoa@ubuntu2:~/Documents/test-deployment$ sudo kubectl get nodes
NAME      STATUS   ROLES                  AGE     VERSION
ubuntu2   Ready    control-plane,master   7d18h   v1.20.4
ubuntu3   Ready    <none>                 7d18h   v1.20.4
ubuntu4   Ready    <none>                 7d18h   v1.20.4
```

Figure 12: Results of get nodes command.

Similarly, manually testing the pods can be done by checking their status with the following command:

```
$ sudo kubectl get pods
```

The results are shown in Figure 13.

```
sigsoa@ubuntu2:~/Documents/test-deployment$ sudo kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
demo   1/1     Running   0          69m
```

Figure 13: Result of get pods command.

The first test deployed on the cluster was a simple application which pings 8.8.8.8 and prints the feedback to the terminal. The following command was used to run the test scenario where “demo” represents the name of the pod and the result in presented in Figure 14:

```
$ sudo kubectl logs demo
```

```
sigsoa@ubuntu2:~/Documents/test-deployment$ sudo kubectl logs demo
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=112 time=9.091 ms
64 bytes from 8.8.8.8: seq=1 ttl=112 time=9.148 ms
64 bytes from 8.8.8.8: seq=2 ttl=112 time=9.079 ms
64 bytes from 8.8.8.8: seq=3 ttl=112 time=15.828 ms
```

Figure 14: The first deployment on the cluster.

A. Service Discovery and Resource Discovery

The Service and Resource Discovery cannot be tested in the traditional sense. Instead, they are tested by calling upon the different services and resources on the cluster.

Figures 15 and 16 show the Service Discovery and the Resource Discovery, respectively.

Figure 15: Service Discovery in browser.

Figure 16: Resource Discovery in browser.

B. Dismounted Soldier

Figure 17 shows that the upload functionality works for both images and videos. The test is executed by uploading an image and a video from the Dismounted Soldier frontend shown in Figure 9.

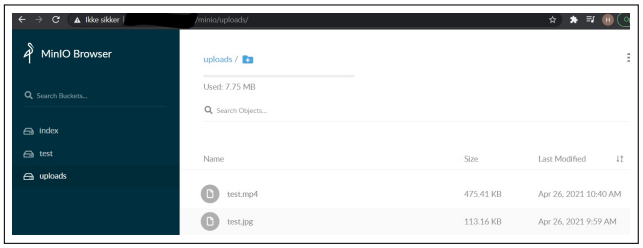

Figure 17: The MinIO storage server.

C. Ingress controller

It is not possible to show the ingress controller in action. All services are routed through a single port and are assigned different URLs by the ingress controller. Therefore, the ingress controller is tested indirectly since the routing works for all services.

Figure 18 gives an overview of an ingress controller.

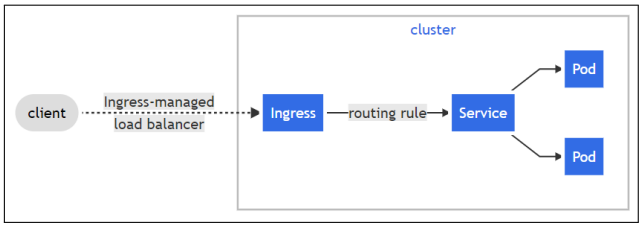

Figure 18: Example of an ingress controller process [31].

VII. Discussion

A. Technology Perspective

Cloud services are a major part of everyday life, that come with both advantages and disadvantages. A major point of cloud services is their availability and reliability. Cloud services can be accessed from anywhere, at anytime. While this benefits the average user, it makes the cloud more exposed to malicious conduct. Additionally, from a single user's perspective, the availability of a cloud service could be less than that of a locally run service, e.g., if the user loses the internet connection.

All the infrastructure linked to cloud services are hosted around the world by various different entities. This gives the average user an easy to maintain cloud, but gives less control of the actual hardware, and the users are at the mercy of the providers.

The cloud is easily scalable, which means it is capable of dealing with situations where there is a need for more resources, as opposed to a locally run service, which would require the purchase of additional hardware.

Kubernetes is a framework for cloud deployment, which comes with its own advantages and disadvantages. One of the major advantages of Kubernetes is its reliability and scalability. Kubernetes comes built in with multiple tools for creating, managing, and distributing nodes and containers. It is easy to add new worker nodes and balance the load between the nodes in the cluster. One of the disadvantages of Kubernetes is that it can be complicated to set up. The installation- and configuration process is fairly complex, and it is recommended to have multiple computers for creating a Kubernetes cluster. It is possible to install Kubernetes on a single computer, but then naturally foregoing many of the cloud benefits (e.g., resilience to partial failure, load balancing). Single-computer installations are mostly used for testing and developing services, whereas production systems use actual clusters consisting of multiple computers.

Docker was used in this project for creating containers in the Kubernetes cluster. It is easy to create Docker containers that can run on any system. Docker containers can be deployed to the Kubernetes cluster. This integration, however, can be difficult to implement depending on what container registry the cluster is using. Docker hub is supported out of the box, but other container registries may require additional configuration.

Important to note, Kubernetes has announced that support for Docker runtime will be deprecated after version 1.20 [32]. Instead, Kubernetes will deploy the Container Runtime Interface (CRI) created for Kubernetes. This means, that moving forward with Kubernetes will mean abandoning Docker in preference of the Kubernetes CRI for development of future services and CIS capabilities. This may or may not be a smart move from Kubernetes. On one hand, it may lead to increased support for their platform and their CRI. On the other hand, since Docker is by far the most used CRI presently, this may also lead to certain developers abandoning Kubernetes in favor of other approaches that continue supporting Docker. For example, *Docker Swarm* is such an alternative, providing much of the same functionality as Kubernetes [33].

B. Operational Perspective

In this work, we were able to show the federated cloud architecture, built using Kubernetes assisting with C2 information processing and distribution workflow in an urban operation context. We saw, that with different nations offering different capabilities, that the coordinated efforts leveraging distributed processing were essential for this coalition operation in the Anglova Scenario's vignette 3: Urban operation.

We can anticipate that with more complex operations, and even more sensors providing data to the tactical edge, these sensors likely don't have much computing resources (e.g., Military IoT [18]). And so, data must be processed to generate actionable information, of which such a federated cloud architecture, if deployed at the tactical edge, could help enable the generation of actionable information. So what is it that Kubernetes brings to the table, as opposed to prior CIS deployments? Kubernetes offers horizontal and vertical scaling of services, and so a new level of resilience to partial

network failure that we did not previously have. In the case that different nations also offer copies of the same services, one can also anticipate use in parallel for lower response times, and/or selecting services based on system load and back-haul link capacity.

And so, we can foresee several benefits to military operations, with increased availability of information to all echelons. With new services targeting analysis and timely reports and concise, to-the-point information products that are suited to the resource-constrained and communication-constrained nodes, we can expect to avoid or at least limit the potential for data overload on the soldiers. One example, is that the improved and more resilient computing capabilities offered by Kubernetes, it is possible to convert voluminous raw data into smaller information updates at the tactical edge. This, in turn, allows for more rapid feedback and so is also an important aspect in maintaining information superiority in high-paced operations.

Finally, concerning future possible use of Kubernetes in FMN, it would be of great importance that the APIs IST-168 developed become standardized within NATO, as they are foundational to realizing the federated cloud architecture that the RTG proposed. Lacking such standardization, the current state of the work is an interesting proposal and has been demonstrated to show promise in an operational setting, but it is not ready for operational deployment and use. At the time of writing, the IST-168 group is working on its final report, where the API specifications will be a major contribution of the work. Hence, we think this report, when published, could be of importance to potential adoption of Kubernetes in FMN in future spirals.

VIII. Conclusion

A fully functional Kubernetes cluster across three physical nodes was set up on the hardware stationed at FFI. The cluster runs on multiple machines with a master node and two worker nodes, with other machines available as backup. The cluster can connect to other Kubernetes clusters, including the cloud federation, for synchronizing services and information sharing.

Due to the nature of the technologies, most of the testing has been manual. As Kubernetes handles orchestration and deployment, it will fail if an error occurs. Therefore, if deployment is successful, it is also indirectly tested.

In this work, we framed our test in context of an urban coalition operation, as specified in the Anglova Scenario (vignette 3). We were able to show that the different nations participating (NOR, DEU, NLD, USA, CAN) successfully could interconnect and share data between their federated cloud-deployed CIS capabilities. Hence, Kubernetes, if deployed at the tactical edge together with a suitable set of services, could help enable the generation of actionable information in support of C2 in a coalition operation.

IX. Further work

Future work from a national perspective could be to set up a Kubernetes cluster on an ARM-based computer like the Raspberry Pi. The Raspberry Pi is a low-powered computer that can be deployed anywhere. The more energy-efficient ARM architecture could be used for making a mobile cluster. The K3s implementation is a lightweight version of Kubernetes that can be used with ARM-based computers [34]. K3s can be used to set up a mobile cloud with Kubernetes. Deploying K3s can be automated with a shell script setting up and installing all packages needed for the K3s cluster.

Some use cases for a mobile cluster like this could be natural disaster relief and humanitarian aid. For example, a cluster like this could be deployed in the aftermath of a natural disaster where an area's infrastructure has been destroyed. In a scenario like this, a mobile Kubernetes cluster could be deployed to provide services to either the victims of the disaster or to the rescue teams trying to find survivors. An example of a service could be a database keeping track of how many people have been found/missing, and the locations where these people were found. Another example could be a service keeping track of areas the rescue teams already have searched, and areas left to be searched.

Future work in context of IST-168 (and potentially follow-on groups) would be addressing how state of the art cloud technologies may be used to distribute processing tasks between mission partner clouds over low-bandwidth and unreliable tactical networks. In this paper we have shown the feasibility of a federated cloud architecture and distributed processing tasks, but further investigating services for adaptive C2 in disadvantaged tactical networks remains work to be done.

Acknowledgment

The NTNU authors performed the software development described in this paper in partial fulfillment of their bachelor degree, and would like to extend their thanks to the NTNU course staff, and especially Emmanouil Papagiannidis, for help and feedback during this project.

We acknowledge the cooperation with our international peers in the IST-168 research task group. In particular Research Scientists Bram Musters (Netherlands Organization for Applied Scientific Research, TNO) and Thomas Kudla (Fraunhofer FKIE, Germany), who provided valuable insights that helped us achieve our technical goals. Also, a big thanks to Harrie Bastiaansen (TNO) who, as chairman of IST-168, coordinated the international efforts towards success.

Last but not least, thanks to Ketil Lund for providing and configuring network resources at FFI.

References

[1] NIST, "The NIST Definition of Cloud Computing," Special Publication 800-145, September 2011.

[2] C. Suggs, "Technical Framework for Cloud Computing at the Tactical Edge," Program Executive OfficeCommand, Control, Communications, Computers and Intelligence (PEO C4I) presentation, https://www.actiac.org/system/files/Technical%20Framework%20for%20Cloud%20Computing%20at%20the%20Tactical%20Edge.pdf, 2013.

[3] MITRE, "Tactical edge characterization framework volume 1: Common vocabulary for tactical environments," MTR070331 MITRE Technical report, November 2007.

[4] H. Bastiaansen, C. van den Broek, T. Kudla, A. Isenor, S. Webb, N. Suri, A. Masini, C. Bilir, and M. Cocelli, "Adaptive Information Processing and Distribution to Support Command and Control in Situations of Disadvantaged Battlefield Network Connectivity," IEEE ICMCIS 2019, May 14-15, Budva, Montenegro.

[5] IBM education, "SOA (Service-Oriented Architecture)," https://www.ibm.com/cloud/learn/soa, published 2021-04-07, accessed 2021-05-21.

[6] ——, "Microservices," https://www.ibm.com/cloud/learn/microservices, published 2021-03-30, accessed 2021-05-21.

[7] J. Hannay and E. Gjørven, "Leveraging network-centric strategic goals in capabilities," Journal of Military Studies, 2021.

[8] Kubernetes, "Kubernetes," https://kubernetes.io/, accessed 2021-05-21.

[9] IBM education, "DevSecOps," https://www.ibm.com/cloud/learn/devsecops, published 2020-07-30, accessed 2021-05-21.

[10] N. Chaillan, "DOD Enterprise DevSecOps Initiative (Software Factory)," https://software.af.mil/wp-content/uploads/2019/12/DoD-Enterprise-DevSecOps-Initiative-Keynote-v1.7.pdf, accessed 2021-05-21.

[11] Platform9, "Kubernetes Federation: What it is and how to set it up," https://platform9.com/blog/kubernetes-federation-what-it-is-and-how-to-set-it-up/, accessed 2021-06-17.

[12] H. Bastiaansen, J. van der Geest, C. van den Broek, T. Kudla, A. Isenor, S. Webb, N. Suri, M. Fogli, B. Canessa, A. Masini, R. Goniacz, and J. Sliwa, "Federated Control of Distributed Multi-Partner Cloud Resources for Adaptive C2 in Disadvantaged Networks," IEEE Communications Magazine, vol. 58, no. 8, pp. 21-27, August 2020, doi: 10.1109/MCOM.001.2000246.

[13] The Linux Foundation Project, "The open container initiative," https://www.opencontainers.org/, accessed 2021-05-21.

[14] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," in IEEE Cloud Computing, vol. 1, no. 3, pp. 81-84, 2014. doi:10.1109/MCC.2014.51.

[15] NATO, "Federated Mission Networking (FMN)," https://www.act.nato.int/activities/fmn, accessed 2021-05-21.

[16] G. Pingen, J. van der Geest, M. Beaujon, J. Voogd, and R. Pieneman, "Data centric C2-services deployment: an experiment on fleets of military vehicles," 25th International Command and Control Research and Technology Symposium (ICCRTS), November 2020, Virtual conference.

[17] M. Fogli, T. Kudla, B. Musters, G. Pingen, C. van den Broek, H. Bastiaansen, N. Suri, and S. Webb, "Performance Evaluation of Kubernetes Distributions (K8s, K3s, KubeEdge) in an Adaptive and Federated Cloud Infrastructure for Disadvantaged Tactical Networks," IEEE ICMCIS 2021, 4-5 May 2021, Virtual conference.

[18] A. Raglin, S. Metu, S. Russell, and P. Budulas, "Implementing Internet of Things in a military command and control environment," Proc. SPIE 10207, Next-Generation Analyst V, 1020708 (3 May 2017); https://doi.org/10.1117/12.2265030.

[19] D. S. Alberts, J. J. Garstka, and F. P. Stein, "Network Centric Warfare — Developing and Leveraging Information Superiority," ISBN-10: 9385505777, ISBN-13: 978-9385505775, January 2016.

[20] N. Suri, A. Hansson, J. Nilsson, P. Lubkowski, K. Marcus, M. Hauge, and M. Peukuri, "A Realistic Military Scenario and Emulation Environment for Experimenting with Tactical Communications and Heterogeneous Networks," International Conference on Military Communications and Information Systems (ICMCIS) 2016. Brussels, Belgium.

[21] N. Suri, J. Nilsson, A. Hansson, U. Sterner, K. Marcus, L. Misirlioğlu, and M. Breedy, "The Angloval Tactical Military Scenario and Experimentation Environment," International Conference on Military Communications and Information Systems (ICMCIS) 2018. Warschau, Poland.

[22] Kubernetes, "Overview of Cloud Native security," https://kubernetes.io/docs/concepts/security/overview/, accessed 2021-06-01.

[23] Docker. What is a container? [Online]. Available: https://www.docker.com/resources/what-container

[24] B. Burns, J. Beda, and K. Hightower, "Kubernetes: Up and Running: Dive into the Future of Infrastructure," 2nd edition, ISBN-13: 978-1492046530, ISBN-10: 1492046531, October 2019.

[25] Kubernetes. Kubernetes pods. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/

[26] ——. What is kubernetes? [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[27] metalLB. Metallb website. [Online]. Available: https://metallb.universe.tf

[28] Træfik. Træfik. [Online]. Available: https://traefik.io/

[29] MinIO. Minio deployment quickstart guide. [Online]. Available: https://docs.min.io/docs/minio-deployment-quickstart-guide.html

[30] IBM. Ibm developer model asset exchange: Image resolution enhancer. [Online]. Available: https://github.com/IBM/MAX-Image-Resolution-Enhancer

[31] Kubernetes, "What is ingress?" [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/ingress/#what-is-ingress

[32] J. Castro, D. Cooley, K. Cosgrove, J. Garrison, N. Kantrowitz, B. Killen, R. Lejano, D. Papandrea, J. Sica, and D. Srinivas. Kubernetes is deprecating docker. https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/, accessed 2021-06-01.

[33] C. Rosen, "Docker Swarm vs. Kubernetes: A Comparison," https://www.ibm.com/cloud/blog/docker-swarm-vs-kubernetes-a-comparison, published 2021-04-03, accessed 2021-06-18.

[34] K3s, "Kubernetes k3s," https://k3s.io/, accessed 2021-05-22.