Implementing cluster backup solution to build resilient cloud architecture

Dang Minh Bui

2020 Laurea

Laurea University of Applied Sciences

Implementing cluster backup solution to build resilient cloud architecture

Author: Dang Minh Bui

Business Information Technology

Bachelor’s Thesis

May, 2020

Laurea University of Applied Sciences **Abstract**

Degree Programme in Business Information Technology

Bachelor's Thesis

Dang Minh Bui

Implement cluster backup solution to build resilient cloud architecture

Year 2020 Number of pages 24

This thesis project aimed to implement a backup solution in the cloud architecture of the client company - Solibri. The company provides out-of-the-box tools for model validation, compliance control, design process coordination, design review, analysis and rules checking. With Solibri products, every party involve in a construction project can build, view and check errors of the building models blueprint.

The company is in the progress of migrating their desktop application to become completely cloud native, and in the long-term to a Software-as-a-Service product. To build a stable infrastructure that is resilient against disaster scenarios, the author of this report oversaw the implementation of a back-up solution called Velero into the running cluster. The criteria to implement Velero was found by careful studying documentation from Velero website along with learning about the company's current cloud infrastructure. One criteria was a persistent storage location is needed for backups to reside. Such a resource was partitioned from the cloud provider Amazon Web Service. Afterwards, the backup solution was configured to connected to a partitioned storage location, deployed into the cluster, and created schedule backups for several resources.

Using Kubernetes as a cluster orchestrator, the information about the deployed Velero solution can be viewed from the command line. Next, the author checked to see if the backups are in place. The scheduled backups can be viewed from AWS console along with retrieved from the command line. The results from these tests showed that the backup solution was integrated into the cluster and creating schedule backups as needed. Since this project is limited to only integrating a backup solution, further steps are still needed to complete the disaster recovery pipeline which should be addressed in future projects. This report can act as documentation on the procedure and knowledge needed to implement this backup tool.

Keywords: Backup solution, resilient infrastructure, cloud native

Contents

1 Introduction

Construction spending is an economic measure that calculates how much money is spent on new construction. According to a survey conducted "Statista.com" through the period of 2014 to 2017, construction spending on a global scale has reached 11.4 trillion dollars in 2018. This number is projected to reach 14 trillion by 2025. This includes the public and private sector, which involves residential and non-residential construction. With such amount of capital invested in construction, it is important that projects are carried out smoothly and effectively. Thus, quality assurance throughout every steps of the project plays a key aspect in a successful project.

One of the essential tools to keep every parties involved on the project on track is the Building Information Modelling (BIM) tool. NBS (2016) stated that is a process for creating and managing information on a construction project across the project lifecycle. One of the key outputs of this process is the Building Information Model, the digital description of every aspect of the built asset. This model draws on information assembled collaboratively and updated at key stages of a project. Creating a digital Building Information Model enables those who interact with the building to optimize their actions, resulting in a greater whole life value for the asset. Following the "NBS BIM Object Standard", a BIM model can be viewed as a combination of many different elements. The model must have necessary information content to define the product. These data include product properties, geometry representation, etc. The model must also contain visualization data for the object to have a recognizable appearance for users. With such information, the BIM models can provide every parties, within the project, with a visual outcome of project construction along with needed information for development.

1.1 About the company

In 1996, Solibri was found with the purpose of helping construction specialists to eliminate any obstacle by detecting problems early on. After 20 years of constant development and innovation, Solibri has become the leader in BIM Quality Assurance and Quality Control. The company provides out-of-the-box tools for BIM validation, compliance control, design process coordination, design review, analysis and code checking.

With the company's product: "Solibri Office", users can not only build and view the project's model but also use it as a collaborative platform throughout the entire project life cycle. Every needed information about every component of a building is brought together. With Solibri Office, the BIM data can be used to illustrate the entire building lifecycle, from start to completion, from inception and design to demolition and materials reuse. Spaces, systems, products and sequences can be shown in relative scale to each other and, in turn, relative to the entire project. The product also comes with predefined rules which every building must comply. By signalling conflict detection, Solibri Office prevents errors to go unfixed at the various stages of the construction development. After an architecture designer has created a BIM model, the software will automatically run checks on the model against those rules. The software comes with other checks and measurements for construction aspects such as material cost optimization and building regulations according to specific countries and regions.

1.2 About the project

Currently, the company's products are only available as desktops application which are only available on operating systems such as Microsoft Windows and MacOS. These products are working well and bring value to customers. However, there are certain drawbacks with desktop applications. Firstly, any improvements to the product can only be published through annual release versions. Although, these releases contain many fixes to bugs and new feature, they can take up to 2 to 3 months of development. Within such period of times, customers must find a way to work around any bugs that have not been fixed. Secondly, fixing bugs and errors are quite tedious. Since errors which appear in production are only found by users, customers are the only party who can produce error reports. With the users' desktop configuration relative unknown to developers, it takes up a lot of time to verify that the programs had the errors or the users' environment caused it. The company wanted to tackle the by utilizing functionality of cloud computing in service develop.

Some initial research into the capability of cloud computing services showed that the issue can be fixed. This research found that using the partitioned resources, the application can be run from partitioned computers in multiple geological locations. Using this approach, customers would not have to install the desktop application anymore. Solibri's services could be accessed through a good internet connection from any geological location. Furthermore, the software would only be running on the partition machines so any environment ambiguity would be eliminating. With these findings, developers had tried to migrate the entire code base of the desktop application to provisioned cloud resources. However, the project resulted in a complete failure. Next, development teams have decided to completely build the application from the very beginning. Every function of the application will be re-developed. With this approach, the company can enter an entirely new way of development. The current desktop application can be completely transformed into cloud-native application which customers can used as a Software-as-a-Service product.

However, the company does not have any cloud infrastructure to host the application. So, the first steps for the migration plan are to build a reliable cloud infrastructure. The DevOps team oversaw developing the needed infrastructure. Most of the components needed are either in development or scheduled. However, it is vital that a cloud infrastructure provide enough resiliency against disasters which this infrastructure did not have. Therefore, the author of this project oversaw implementing a cloud backup solution. The purpose of this project is to eliminate human errors accidentally cause by developers. The scenario which this project aims to aid is considered the "worst case scenario". Since the current infrastructure is managed using Kubernetes and every DevOps engineer have access to the tool and entire cluster, they can easily terminate the entire cluster. Without any backup to restore the cluster from, the infrastructure would need to be rebuild from the beginning. For any company, this scenario could be devastating to its open SLAs, revenue and reputation. Before the beginning of this project, Velero was chosen as the most suitable solution. The author's responsibility in this project is to successful integrate Velero into the current system and backup current system's state. It is important to note that this project only addresses part of a complete solution to the disaster scenario. The project scope is only to implement a backup solution to restore the cluster. The process of implementing steps to recover the cluster will be implemented in later projects.

2 Research & organization frameworks

Establishing a good research approach can aid developers greatly in any project. In 2009, Dawson stated that research is the expansion of knowledge with series of systematic activities that have not been executed. The author also mentioned that theory plays a valuable factor in every research. Throughout this project, the author applied several theories as support to the development of the final product.

2.1 Software development life cycle (SDLC)

Within Solibri DevOps team, the infrastructure which the team oversees is completely cloud based. This meant that every resource that the team utilized is accessed and controlled remotely through coding. "Infrastructure as code" is the term used for this approach (Dadgar, 2018). Unlike Networking engineers, who deals with hardware infrastructures, Solibri's DevOps engineers handles infrastructure by partitioning cloud resources and manage them remotely through running scripts and commands. Every configuration, modification, feature of the infrastructure is treated as software. Therefore, the "Software development life cycle"

(SDLC) is a suitable procedural guideline for development in this project. The SDLC procedure comprises of 5 main distinctive sections shown in figure 1. Each has its own role within a project, and each support one another.

Firstly, developers need to analysis and collect information about the requirements of the final product. The outcome of this step should be a list of conditions that the software needs to satisfy. The result of this phase will be used as the fundamental grounds for development in later steps.

Next, developers and engineers work together to come up with a basic design of the final application. This includes its functionalities, features and operation within the company. In this phase, the procedures to build the software are also drawn up. Based on all the features of the application, steps to complete such functions are created so whoever take up the tasks can easily follow them without ambiguity.

Thirdly, the engineers in charge of making the application, which is the author of this report, will start coding at this phase. Using the predefined procedure and requirements listed as guides, the develop will build the application. Furthermore, any unknown knowledge needed to carry out this task will also be gathered.

Once, the application is developed, it needs to go through testing to ensure the product meets conditions stated in the first phase. These tests also use the requirements as guides to be developed.

Finally, the product is deployed to production environment and maintained. The product will be monitored for any errors which will be fixed with later releases. New features will also be added if they are deemed necessary. Once the product no longer provides customer with value, it will be discarded, and its life cycle ends.

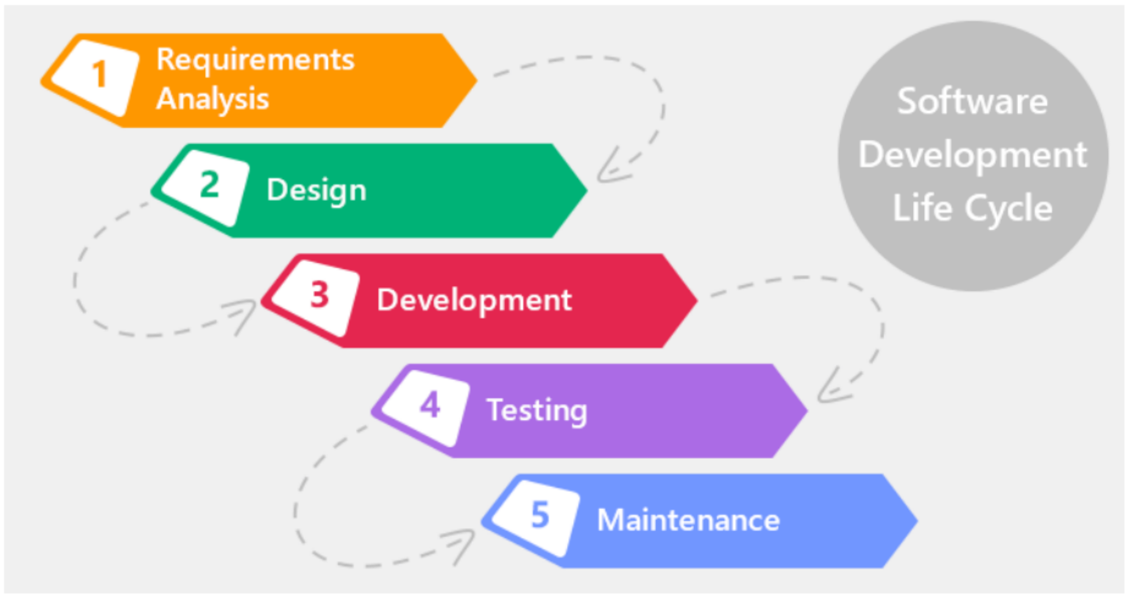


Figure 1 Software Developement Life Cycle phases

2.2 DevOps organization framework

Although, the SDLC is a good procedural guideline for linear software development, the DevOps development framework also needs to be mentioned. Many aspects of project follow steps within this framework therefore it should be discussed.

The main role of a software developers ends when the final product has been built. The developers then wait for any errors report or features request before continuing. However, this is not so for the role of a DevOps engineer. The term “DevOps” can be viewed as a combination of “development” and “operation”. This also means that the engineer is responsible for both the building of the infrastructure, monitoring of application and maintenance of the infrastructure. Illustrated in Figure 2, the DevOps development lifecycle is a continuous cycle and not as linear step as the SDLC. In Figure 2, encompassing the entire cycle is the term: “Real-time communication”. This illustration pointed out the key aspect for successful DevOps approach is the need for team member to continuously communicate. Any problem that the engineer encounters should be relayed and reported to be fixed. Since the team oversees the entire application infrastructure, any errors that gets pass could potentially result in the company’s service becoming unavailable. At the centre of the illustration, two key aspect of DevOps development are shown, which are “continuous integration” and “continuous feedback”. With the team communicating frequently and effectively, issues can be addressed early on and fixed. This leads to the team repeatedly developing and integrating solutions to the infrastructure.

To effectively find and detect errors, DevOps engineer needs to rely heavily on tools as production environment can have numerous disasters scenario. Furthermore, DevOps engineers are also overseeing development of an automated, continuous pipeline for service deployment. With fierce competition, any new feature and errors fixes which developers have created needs to reach the market as soon as possible. With so much to monitor and develop, it is best practice DevOps engineers to utilize automated tools as much as possible. In 2018, a blog post from AltexSoft stated that existing DevOps tools cover almost all phases of continuous delivery, starting from continuous integration environments and ending with containerization and deployment. While today some of the processes are still automated with custom scripts, mostly DevOps engineers use various open source products. However, this heavy utilization of tools also has one drawback. With new technology, a learning curve is necessary. Most engineers who are unfamiliar with a technology will need time to do research on how to use the tools and its compatibility with current system. Therefore, learning about modern technology plays a vital part in day-to-day operation of a DevOps engineer.

The Solibri DevOps team also takes this principle to heart and constantly utilized modern tools to build a more robust and effective cloud-native infrastructure. Team members emphasis on the practice of learning about new technology and plan to integrate them into our workflow. The author of this report also shares the same principle and utilize numerous tools to complete the project. These tools will be discussed in the “Knowledge base” section of the report.

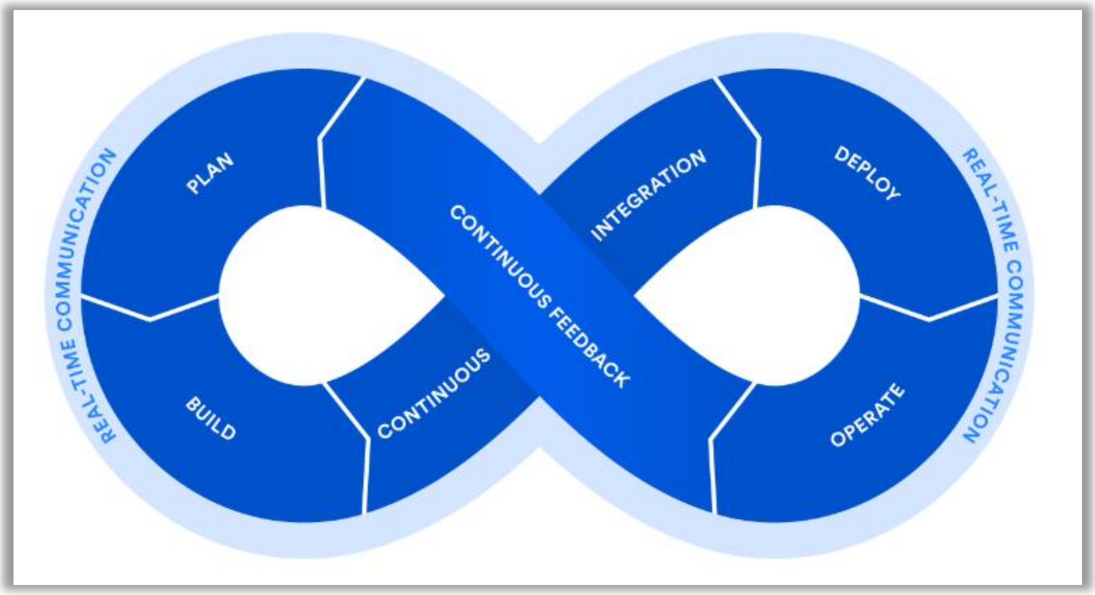


Figure 2 DevOps developement lifecycle

2.3 Agile project management methodology

Agile methodology is a project management methodology following the Lean development principles. This principle was created to help developers concentrate on issue and activities which contributes to the service enhancement (Fair, 2012). In Fair's opinion, Lean principle main purpose is to eliminate the "waste" and focus developer's attention on more important aspects such using customer feedback as knowledge to build products and add business value. The "waste" which this principle tries to discard included unnecessary meetings with managers to decide which products or features to be built. Planning large product development period can be considered "waste" as this step can take a large amount of time and yield plans that are subjected to changes.

Agile method represents a flexible software development method that divides the process into small, manageable iterations called "Sprints", illustrated in figure 3. The duration for each sprint usually last for 2 weeks. Before each sprint begins, the team while host a meeting to determine what should be the focus of development and divide the role of each developer. The context of the meeting usually includes the projected outcome, tasks needed to be completed, knowledge supporting each task and final product of the sprint. Engineers also takes this time to discuss what could be possible drawback and impediments to be eliminated first. Once the meeting is over, engineers carry out the written plan for that sprint. When each sprint is completed, the work can be reviewed by the project team. Once the expected results are not satisfied, the work is revised and modified into new sprint planning.

An obvious benefit gained from this method is fast product time to market. At the end of every sprint, new features and bug fixes are created. DevOps team can deploy those new changes through the continuous pipeline and keep development speed fast. With team members constant communication throughout the process, they can see a more realistic progress of the project, from planning to review sessions. Agile deliver this transparency due to developers are focused on the same goal as one another. This project management method also provides improvement in quality assurance and adapting to new changes. By testing and fixing defects so quickly in each cycle, Agile increases the overall quality of the final product. Furthermore, another advantage of using Agile, which can be considered as its "signature", is the ability to familiarize with new modifications. With sprint being such short period of times, engineers will be available soon after each sprint and ready to deal with new issues. Additionally, agile allows for changes to be made in any stage of development.

There are some disadvantages when applying Agile methodology that should be mentioned. For example, a project using Agile requires constant communication between team members. During a sprint, member can still come into impediments and requires assistance. A sprint will only be considered successful if all tasks are completed. With members not receiving needed help, the task can be delayed along with the sprint goal and deadline. Therefore, Agile methodology requires a high involvement and collaboration from team members.

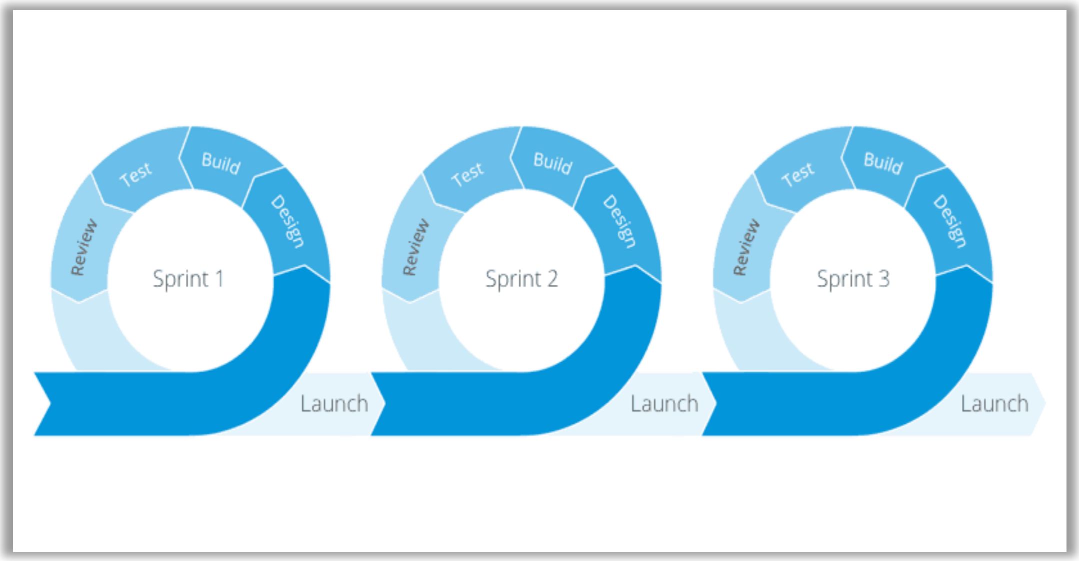


Figure 3 Agile sprints iteration

2.4 Applying research method within organization framework

As mentioned above, the Solibri DevOps team build the current infrastructure following the principle “infrastructure as code”. Therefore, the final product was treated as a software and the SDLC guideline was used. However, it is important to note that this project main purpose is not to develop new software but to integrate an existing solution. The SDLC only acted as initial instruction to assist in planning of tasks needed to be carried out. Throughout the project, the author used mainly the practices and principle of DevOps development lifecycle and Agile project management framework.

The SDLC model main components were used as main goals of each Agile Sprints iteration. These iterations were created for the SDLC phases: “Requirements”, “Knowledge gathering & Design”, “Implementation and testing”. Although the guideline comprises of 5 main phases, some changes were made to be more suitable to the project.

Within the First phase, the requirements for the final product is not the only focus point. As a DevOps engineers, the author must intergrade new technology into the current system. Therefore, requirements on such subject also needed to be discussed. The second phase had an additional purpose. As mentioned, the DevOps development principle have a heavy learning curve with new technology before being implemented. Thus, after having all primary requirements have been established, the author needed to gather enough knowledge to design a plan for development. The purpose of this extra work is to discard any possible ambiguity during development. Thirdly, “implementation” and “testing” are combined as a single step. This is caused since the solution needed to be deployed to the infrastructure. This deployment is considered a test to see if the solution is integrated successfully. If it fails, the engineer needed to investigate the error and redeploy as many times as needed. The final change made to the SDLC model is the discard of the “maintenance” phase. This project only focusses on implementing a solution to Solibri infrastructure. Therefore, this phase is not mentioned. Although, it is continued beyond the time of completion of this project.

3 Knowledge base

Within this project, the author had to use several theoretical concepts to implement the backup solution. The following section explains the concepts of resilience infrastructure, containerized application architecture, and the DevOps tools used through out the project.

3.1 Resilience Architecture

For an application to be considered resilient. It must withstand multiple harsh factors of production environment and any disaster that comes with it. For example, the application must work flawlessly under extremely high workload. The Kubernetes must be setup to automatically response to such situation. A common configuration is to create more nodes to handle the workload.

According to an article by Google in 2020, a resilient application is one that continues to function despite failures of system components. This resiliency requires planning at all levels of the underlying architecture. It influences how engineers lay out infrastructure and network, and how data storage and services were designed. Building and operating resilient application infrastructure is considered difficult. This is especially true for distributed apps, which might contain multiple layers of infrastructure, networks, and services. Mistakes and outages happen and improving the resilience of the service is a crucial continuous process.

Another type of disaster is misconfiguration of the infrastructure. Unlike other scenario which are cause by parties outside the company. This type of disaster is caused by developers. Although this seems rare, this scenario does happen. Any enough privilege can make changes to the running infrastructure, either with or without harmful intention, the developer can misconfigure the entire cluster to behave in undesired way. This can greatly affect how the application functions on clients' side. Cassel, in 2017, had written an article on an incident caused by a Junior Developer. The developer accidental clears the company's production database completely. Although, this was an accident, legal measure was needed as the data loss was extremely severe. Human errors are an existing hindrance to any infrastructure. Therefore, when building any sort of architecture, this type of issue also needs to be considered and remedied.

3.2 Containerized application architecture

This method of application architecture design is used by the Solibri's DevOps team. The current architecture is built following this method. Therefore, it is important to understand this concept before implementing or configuring and part of the current infrastructure.

3.2.1 Application containerization technology

According to Pablo in 2017, application ccontainerization (container-based virtualization) is a method for deploying and running applications and services. This method allows services to be released for customers without having to launch Virtual Machines for each application. The illustration in figure 4 shows the comparison between virtual machines application and containerized application.

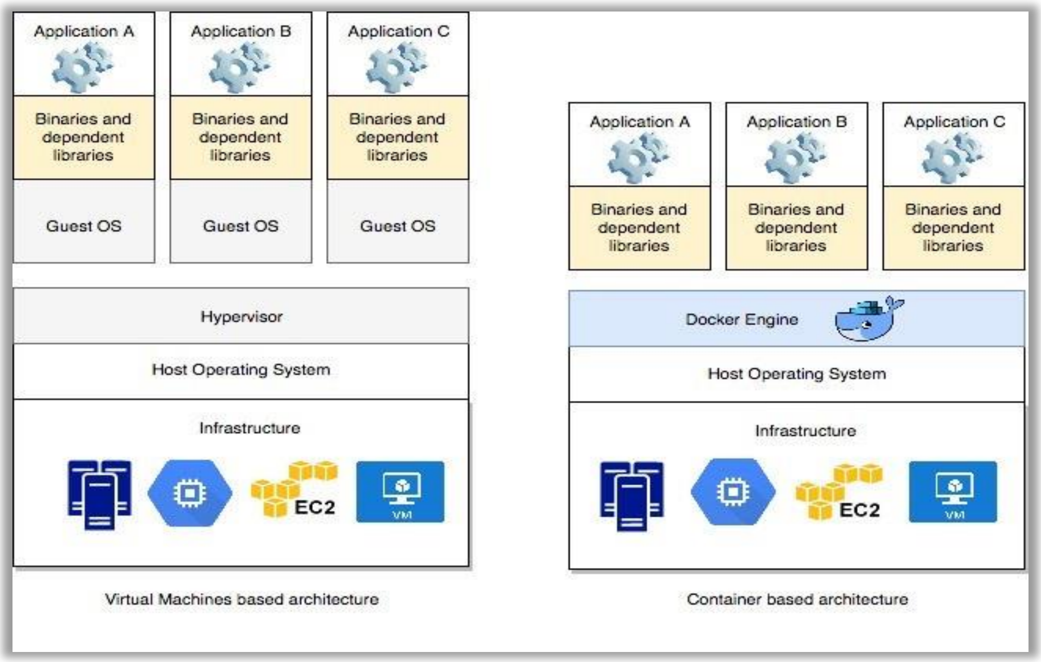


Figure 4 Virtual machines vs container based architectures

Currently, the most well-known method and tool for application containerization is Docker. This tool utilizes resources isolation aspect of the Linux kernel, such as “kernel namespaces”. With such implementation, independent containerized applications or “containers” can be run without the need for virtual machine. Ultimately, overheads from starting and managing resources for each virtual machine is discarded. Figure 5 illustrates the client-server architecture used by Docker. The client sends commands form users to the Docker daemon, which consists of all the main containerizing function such as building, running and deploying containers. Both the client and the daemon could be running on the same environment or from remote system. With remote systems, the two sides communicate through network interfaces.

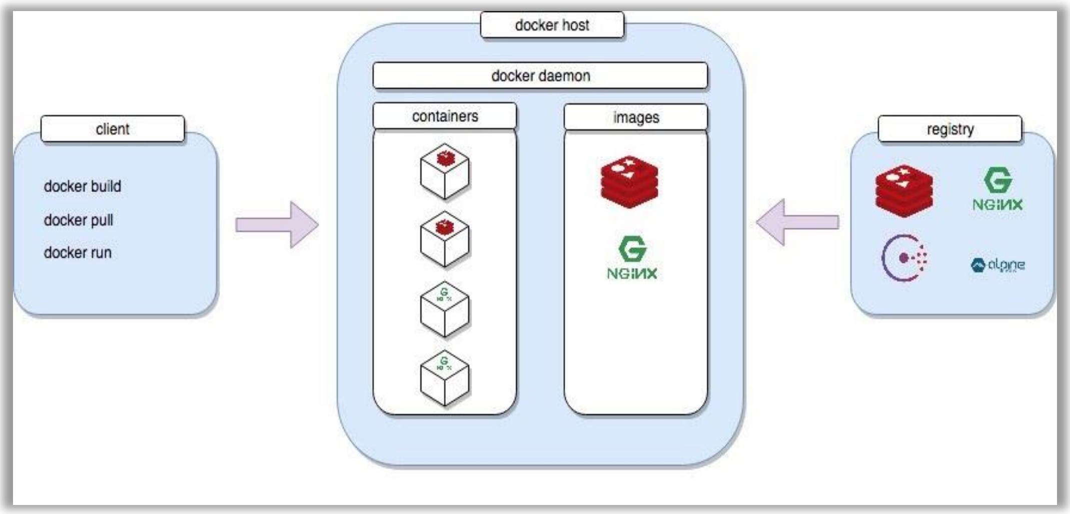


Figure 5 Docker architecture overview

In 2017, Iorio has mentioned that there are numerous advantages to using containerized application. This approach to development help developers in every stage of the service lifecycle include development, quality assurance and product maintenance. For example, “Homogeneous environments” is one of the best advantages of this technology. Developers can focus on building the solution and once it containerized, it will work on any predefined environment with docker engine. This ensure eliminate maintenance overhead of separate development and production environment. Furthermore, with a well-defined continuous integration pipeline, any changes to the container can trigger a new build and testing process. Once all the testing pass, the new changes will be automatically deployed. This saves developers valuable time.

3.2.2 Cluster orchestration

Once an application has been built and containerized, it will reach customers by being deployed to provisioned machined called “nodes”. These nodes are usually partitioned through utilizing available resources from cloud providers such as Amazon, Microsoft and Google. With such approach, the application can reach customers across geological location. With a large audience of users, more and more nodes will be deployed to meet customers’ demand. The numbers of nodes could be thousands. It is practically impossible to manually manage such large-scale infrastructure. Therefore, cluster (multiple nodes) needs to be managed through Orchestration tools such as Kubernetes which is currently utilized within Solibri DevOps team.

A Kubernetes cluster is a set of node machines for running containerized applications. At a minimum, a cluster contains five worker nodes and three master nodes illustrated in figure 6. The master node is responsible for maintaining the desired state of the cluster, such as which applications are running and which container images they use. Worker nodes execute the process needed to run the applications and workloads. A desired state is defined by configuration files made up of “manifests”, which are JSON or YAML files that declare the type of application to run and how many replicas are required to run a healthy system. The cluster’s desired state is defined with the Kubernetes API. This can be done from the command line (using the kubectl tool) or by using the API to interact with the cluster to set or modify the desired state. Kubernetes will automatically manage the cluster to match the desired state. As a simple example, an application with a desired state of "3" was deployed. This means 3 workers nodes with the container of the application should be running. If a single container crashes, Kubernetes will see that only 2 replicas are running, so it will add 1 more to satisfy the desired state. Within different scenarios, Kubernetes will responses differently. However, this cluster orchestration tool will always maintain the desired state of the cluster.

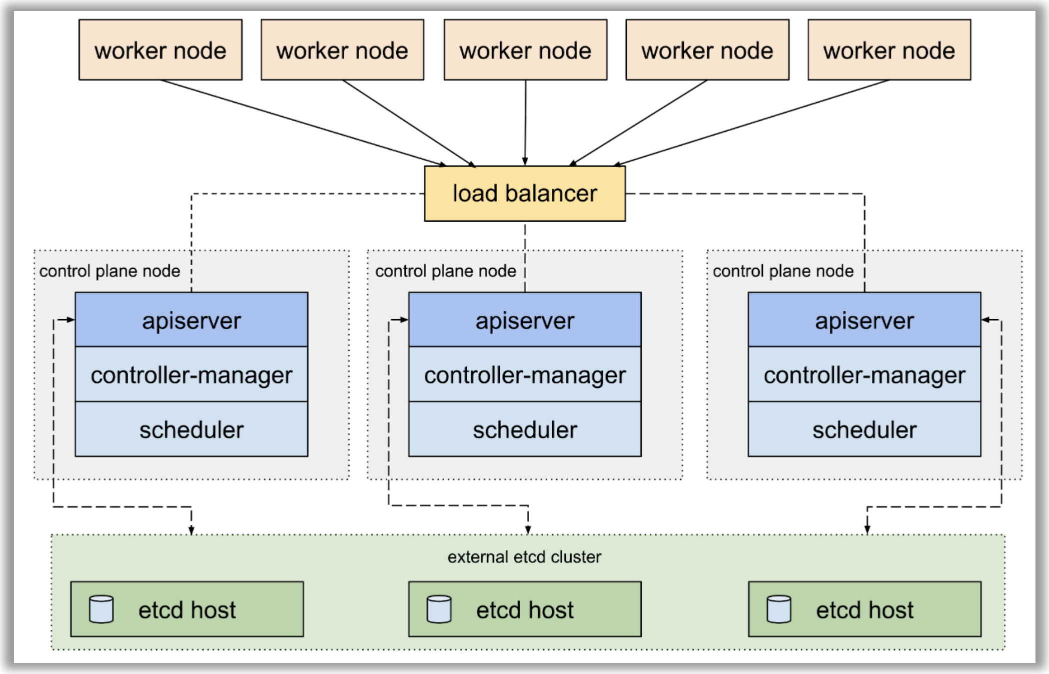


Figure 6 Minimum Kubernetes cluster illustration

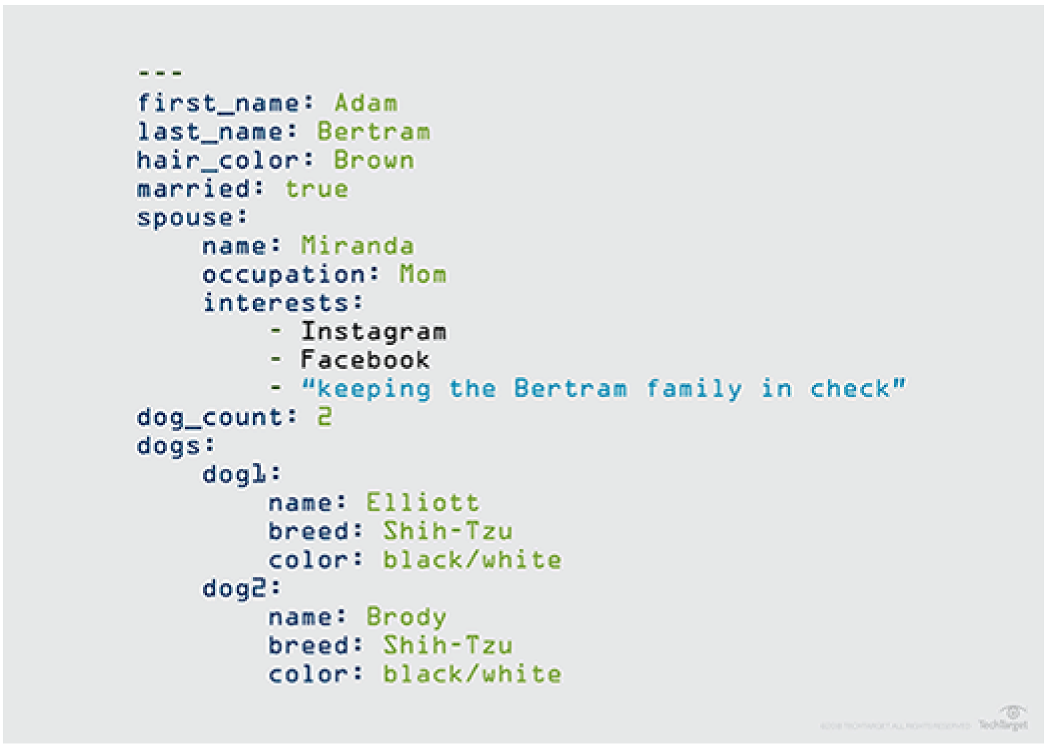
3.3 DevOps Tools

As mentioned above, within the DevOps organization framework, multiple tools are used in day-to-day operation. Therefore, it is vital that engineers understand the tools needed to complete any given task. The following tools are

3.3.1 Yaml file format

This file format was mentioned as it differs from normal format used for coding. The author of this report had to learn and become familiarized with this format before implementation can begin. Throughout this project, every tool used this file format. Each of the DevOps tools also has its own syntax and parameter in YAML format. To understand how to use them, these fields also needed to be study before development.

According to Wikipedia contributors (2020), YAML (a recursive acronym for "YAML Ain't Markup Language") is a human-readable data-serialization language as exampled in figure 7. This language is commonly used in configurations files with the purpose of storing data to be transmitted. YAML main advantage is its simple syntax which allows anyone to read and understand the program. Additionally, YAML syntax follows key/value pair. Every variable will have a value, this helps transmit data easily. Another great advantage of YAML is its usage of Python-style indentation and nesting. This helps engineers create lists and arrays and for loops with YAML file. In this format, square brackets “[]” are used for lists and curly brackets “{}” for maps.



```
---
first_name: Adam
last_name: Bertram
hair_color: Brown
married: true
spouse:
    name: Miranda
    occupation: Mom
    interests:
        - Instagram
        - Facebook
        - "keeping the Bertram family in check"
dog_count: 2
dogs:
    dog1:
        name: Elliott
        breed: Shih-Tzu
        color: black/white
    dog2:
        name: Brody
        breed: Shih-Tzu
        color: black/white
```

Figure 7 YAML file format example

3.3.2 Ansible

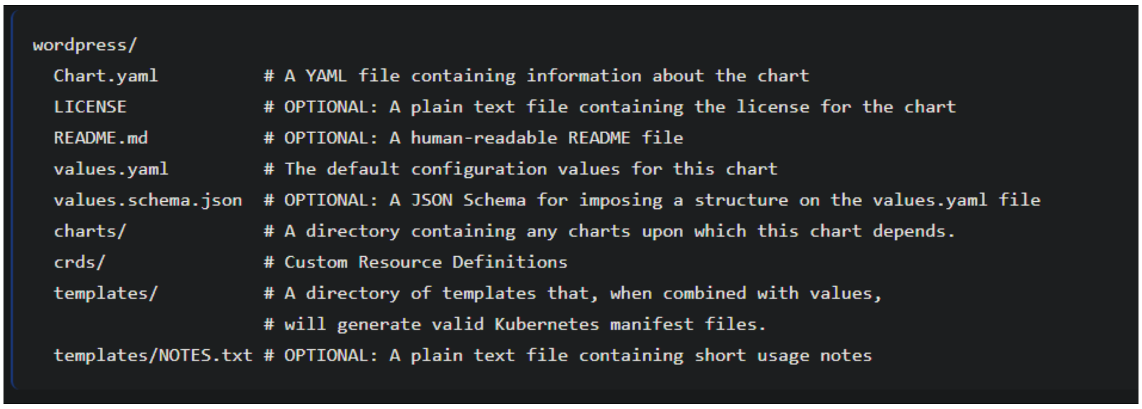
Kubernetes main functionality is to ensure the application health and availability. This tool manages how the infrastructure is scaled and how the nodes functions. However, the state of the software, nodes and cluster must be declared to the orchestration tool. The Ansible tool was made for such purpose. In this project, Ansible is considered the cluster configuration tool.

According to Red Hat official documentation, Ansible is an information technology automation tool. This tool can be used to configure systems, deploy software, and orchestrate more advanced tasks. Ansible works by establishing OpenSSH connection to currently running nodes and deploy small applications called "Ansible modules". These programs resource models of the desired state of the cluster. Once the tools have finished executing every single module on all required nodes, these modules will be removed. The main purpose of using this tool in development is to simplify the cluster configuration tasks for engineers. By using a centralized tool for configuration and each configuration is written as file, engineers can easily mange and see what has been configured to the cluster.

To fully understand how to work with this tool, a few concepts must be mentioned. Firstly, Ansible need to know exactly what nodes needs modification. The lists of managed nodes are called the inventory filer or “hostfile”. Using these files, developers can segment to infrastructure and make changes to specific segment using Ansible. The second concept that engineers should be aware of is “modules”. Ansible comes will prebuild function which are called “modules”. Each module has a particular use, from administering users on a specific type of database to managing VLAN interfaces on a specific type of network device. Developers can invoke a single module with a task or invoke several different modules in a playbook. A “task” is a command or a call for action from users. An Ansible “playbook” is a list of such tasks. Playbooks are written in YAML and are easy to read, write, share and understand. When executing a playbook, the tasks will be carried out sequentially. Within these files, some specification can be declared including targeted hosts, SSH tunnelling port, users’ privilege, credential, destination folder, etc.

3.3.3 Helm chart

With numbers of applications and tools deployed to the cluster, it is important to have means of configuring and updating such programs when needed. Helm is a Kubernetes package and operations manager. According to Foster in 2019, Helm-Charts are used to deploy an application, or one component of a larger application. A Helm repository is an http server with an index.yaml file containing names and metadata for the repository’s charts. This means that the server location is extremely versatile. Developers can use any online repository of the application such as GitHub or GitLab. Only the packages are stored in this repository. The charts themselves are version controlled, usually with the application that they support. A Chart is organized as a collection of files inside of a directory. The directory name is the name of the chart (without versioning information). All information about the package are defined within this repository. A typical chart file structure would look like the example of figure 8. In this figure, the example chart describes a software called “WordPress”. Engineers will specify any configuration needed to the software using the Chart files. Afterwards, Helm tool is used to deploy this updated software to the cluster.



```
wordpress/
  Chart.yaml          # A YAML file containing information about the chart
  LICENSE             # OPTIONAL: A plain text file containing the license for the chart
  README.md           # OPTIONAL: A human-readable README file
  values.yaml         # The default configuration values for this chart
  values.schema.json  # OPTIONAL: A JSON Schema for imposing a structure on the values.yaml file
  charts/             # A directory containing any charts upon which this chart depends.
  crds/               # Custom Resource Definitions
  templates/          # A directory of templates that, when combined with values,
                      # will generate valid Kubernetes manifest files.
  templates/NOTES.txt # OPTIONAL: A plain text file containing short usage notes
```

Figure 8 Chart file structure example with explanation

3.4 Disaster recovery solution - Velero

Velero was designed to be an out-of-the-box cloud-native backup solation. The software allows backing up and restoring deployed Kubernetes cluster resources and persistent volumes. DevOps engineers can run Velero with a cloud provider or on-premises data storage. Main function of this tools includes taking backup of the cluster manifests along with application state, migrating and replicating clusters to other clusters. Each operation, including on-demand backup, scheduled backup and restore, is a custom Velero resource, defined with a Kubernetes Custom Resource Definition (CRD) and stored in etcd. This backup tool also includes controllers that process the custom resources to perform backups, restores, and all related operations. With such features, Velero can be seamlessly integrated into any Kubernetes clusters.

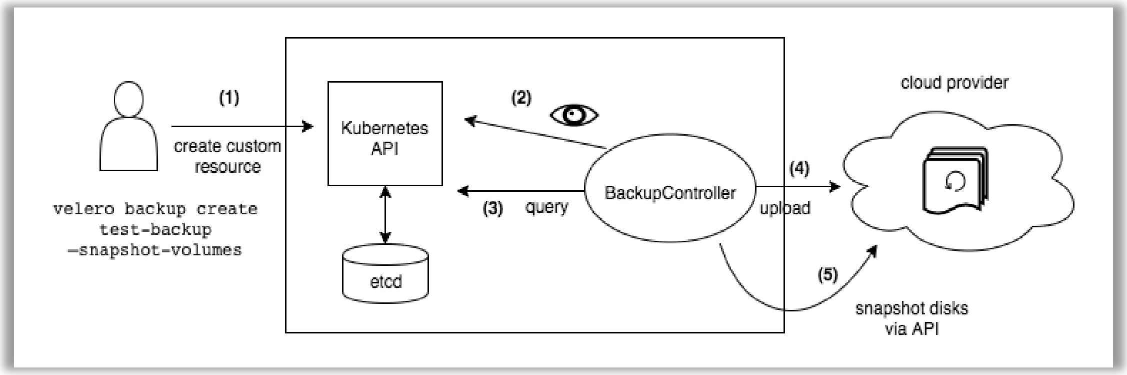


Figure 9 Velero backup flow

Illustrated in figure 9, a backup process has 4 distinctive operation. Firstly, from the command line, users can create a backup of any desired resources. This will create a call to the Kubernetes API server to create a backup object. Next, the BackupController is alerted of the new Backup object and performs validation. Afterwards, the same controller will execute the backup process by collecting data through a query to the Kubernetes API for resources. Finally, the backup is uploaded to the desired destination which could be cloud providers' persistent volumes.

4 Implementation & testing

This section documents the procedure used to integrate Velero into the running architecture.

4.1 Sprint 1: Requirement Analysis

Firstly, in the kick-off meeting, the team discussed and specified some requirements for the final product. This backup solution needs to help the DevOps engineers build a more resilient infrastructure. This includes enhancing the architecture to deal with the "worst case scenario" of complete cluster termination. Therefore, the solution needs to able to constantly make backup of current system. Additionally, the backups need to be run automatically. Furthermore, the backups need to be easily restored in case such disaster occurs. Finally, security measure to ensure only personnel with credential can access these backups files.

The requirements for Velero integration was not discussed in the meeting. This task was delegated to the author of this report. After some research, some initial configuration requirement could be seen. The backup tool needs a persistent volume for the backup to be stored. Since the company is using resources from AWS (Amazon Web Service), Amazon S3 bucket is the most suitable product to used for this project. However, AWS has IAM (See Appendices) security measures in place where no services can access resources without proper credential. Therefore, Velero needs credentials to use the S3 bucket. Furthermore, the current system has implemented protocols to deploy new application through Ansible and Helm Chart. Thus, to deploy a backup solution, this software also needs to follow the team's protocol.

4.2 Sprint 2: Knowledge gathering & design

As mentioned, the DevOps principle focus heavily on learning about new technology. The author has gathered a list of subjects that was unfamiliar but needed to complete the project. The result of this process is the "knowledge base" section within this report. However, the information within that section is not enough to complete the project. An overall understanding of deployment protocol is necessary. The protocol utilized of the time of this project have conditions on how to use each DevOps tools.

Kubernetes communicated with the deployed application through namespaces. Therefore, each component needs a separate namespace. Furthermore, Kubernetes secretes for each software's needs to be stored in the "k8s_secrets" directory. These secrets are credential to access AWS resources. They must be hashed with Ansible-vault built-in function.

Ansible is used as the deployment tools and each deployment must be executed through the playbook "k8s-cluster.yml". Every necessary configuration must be in place before the playbook is ran. Furthermore, every task must be written in playbooks and not execute as ad-hoc commands. This ensure that the team can review what have been deployed to the cluster.

Helm is utilized as the package manager for the cluster. This tool has a built-in function to compare changes in local helm charts with charts from designated repository. By default, the Helm will prepare the application according to the upstream repository. However, if there are any changes detected in the comparison, Helm will modify the software to match such changes. Therefore, in this project, developers only needs to specify needed change in the "values.yaml" of the Velero helm chart since this file is responsible for the configuration.

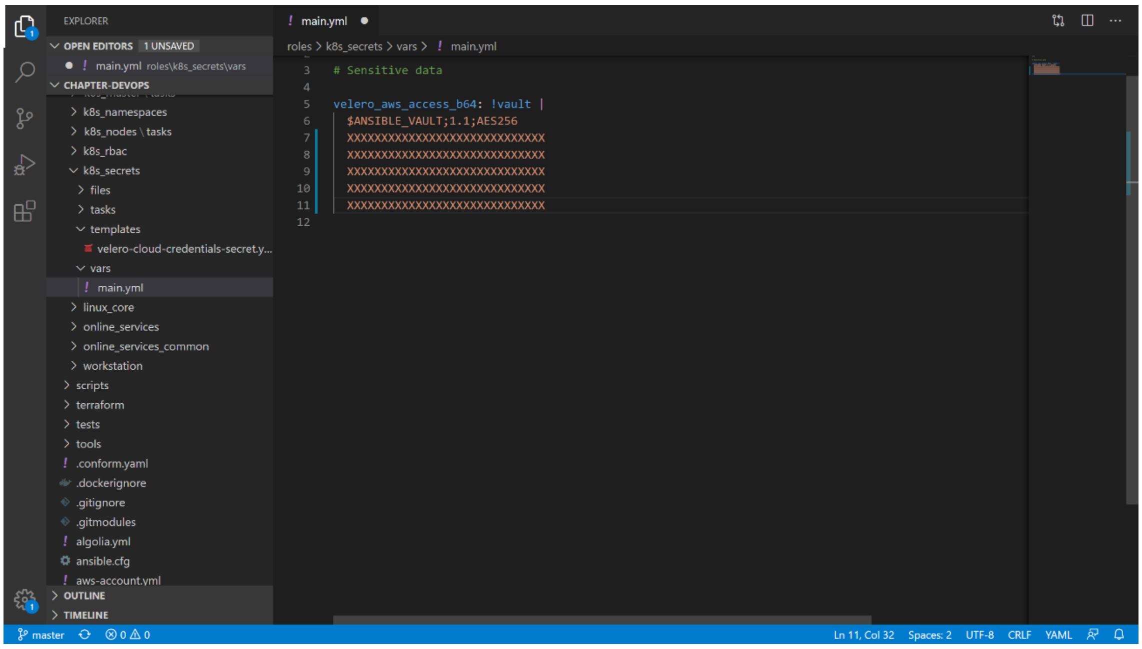
With the requirements and gathered knowledge, a design of procedural steps to complete the project can be drawn up. Firstly, AWS resources and security measures needs to be in place before further action can be taken. Next, the namespace for the backup solution needs to be created. After the above steps are complete, the configuration file of the solution needs to be created. Finally, the solution can be deployed and used to make backup.

4.3 Sprint 3: Deployment & Testing

This section document the deployment and testing results of integrating Velero into the running infrastructure.

4.3.1 Deployment

First, a persistent volume storage service (S3 bucket) was partitioned from AWS cloud provider. This resource was named "velero-backup-1504-io" for later configuration. Once the volume is set up, a user account was created for Velero. This account only has enough privilege to access the bucket. Next, the ansible-vault tool was used to hash the credentials. The ansible pipeline will automatically decrypt the credential when the service makes a request to AWS resources. Furthermore, the hash key can only be accessed by team members to ensure security integrity. The resulting encrypted credential is stored in the directory "k8s_secrets" following team's protocol as shown in figure 10.

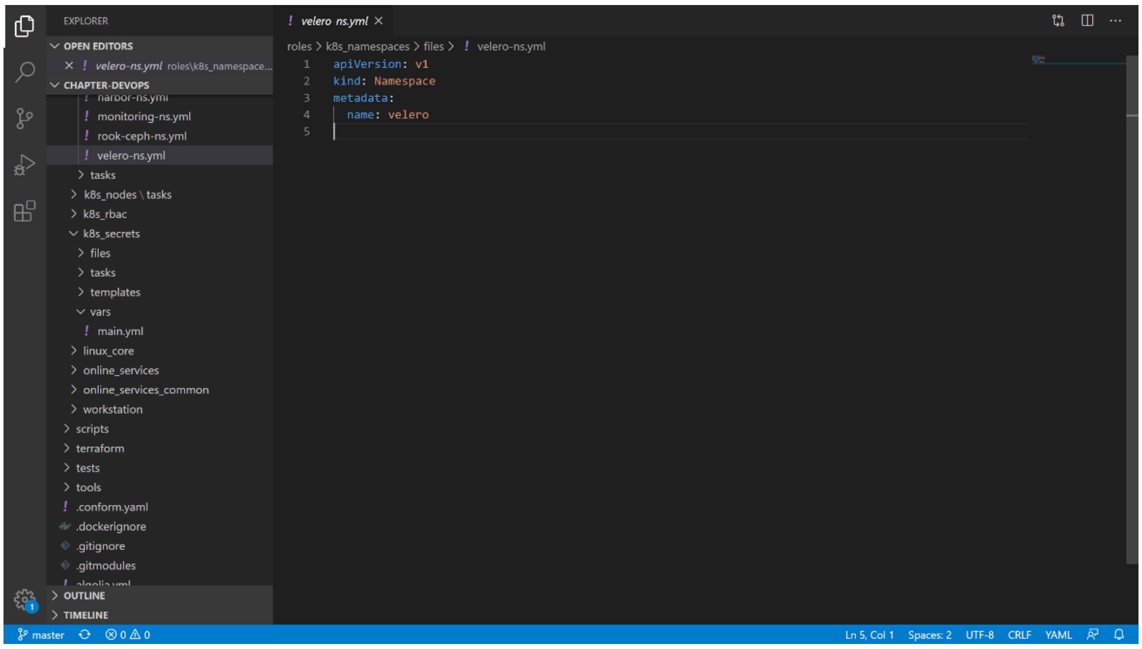


roles > k8s_secrets > vars > main.yml

```yaml
# Sensitive data

velero_aws_access_b64: !vault |
  $ANSIBLE_VAULT;1.1;AES256
  XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Figure 10 Hashed credential for Velero

Next, a namespace for the backup solution is created. The team also had a playbook to automatically create new namespace when necessary. The author added a file in the directory “k8s_namepsace” which specify that a namespace called “velero” is needed (figure 11).

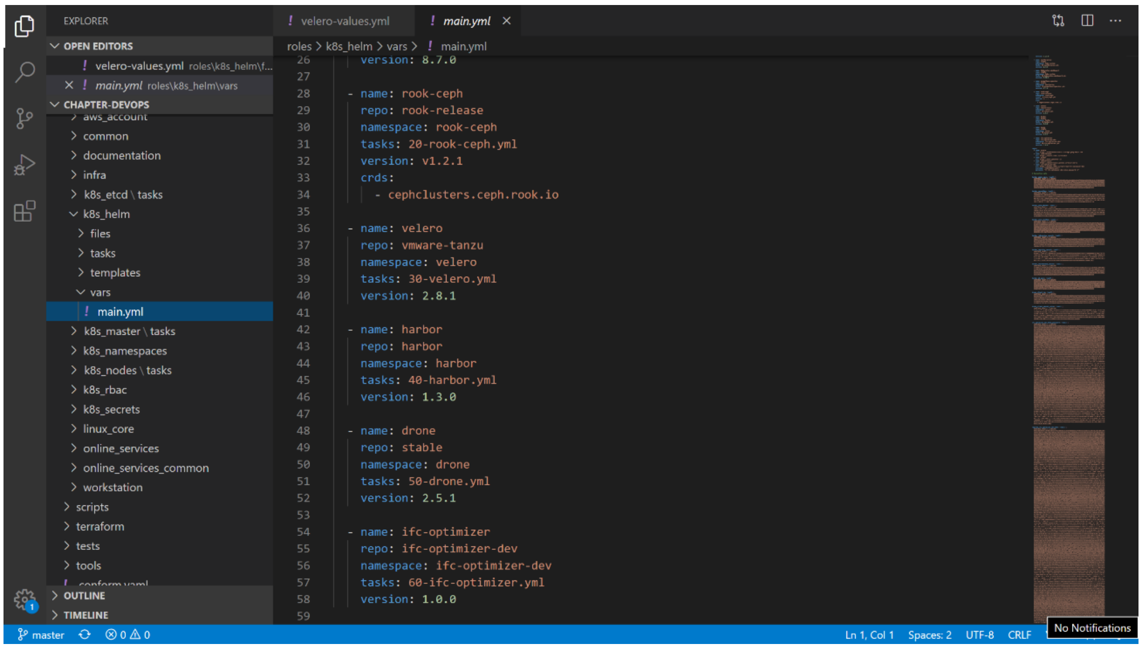


roles > k8s_namespaces > files > velero-ns.yml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: velero

```

Figure 11 Velero namespace file

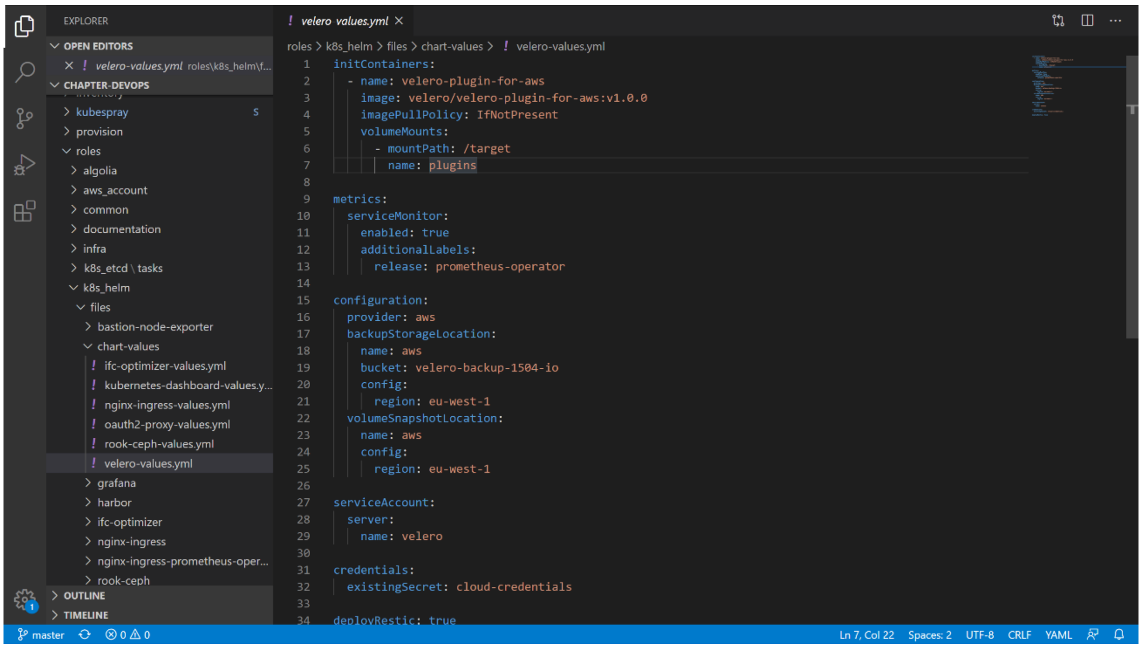
Finally, the Velero configurations needs to be modified so the solution can be deployed an integrated. A new task to install new application across nodes needs to be added to tasks file in the “k8s_helm” repository, (figure 12). The team’s protocol stated that configuration or values files must remain under the repository “k8s_helm/files/chart_values”. Furthermore, the values file must follow a common naming convention. Once the task has been added, the values file was created as shown in figure 13. This configuration file stated several aspects of the desired final product. First, the initial container needs to be created if it is not present. The container needs to be built with an image of the Velero plugin for AWS. The backup storage location is the S3 bucket with the name “velero-backup-1504-io” within the eu-west-1 region. Kubernetes will communicate with this application through the namespace “velero” as stated by the serviceAccount configuration. Although, the YAML file format make the file seems simple. In this project, the syntax was followed strictly from Velero documentation as the tool has its own syntax for configuration file.



roles > k8s_helm > vars > main.yml

```
    version: 8.7.0

  - name: rook-ceph
    repo: rook-release
    namespace: rook-ceph
    tasks: 20-rook-ceph.yml
    version: v1.2.1
    crds:
      - cephclusters.ceph.rook.io

  - name: velero
    repo: vmware-tanzu
    namespace: velero
    tasks: 30-velero.yml
    version: 2.8.1

  - name: harbor
    repo: harbor
    namespace: harbor
    tasks: 40-harbor.yml
    version: 1.3.0

  - name: drone
    repo: stable
    namespace: drone
    tasks: 50-drone.yml
    version: 2.5.1

  - name: ifc-optimizer
    repo: ifc-optimizer-dev
    namespace: ifc-optimizer-dev
    tasks: 60-ifc-optimizer.yml
    version: 1.0.0
```

Figure 12 Tasks file in k8s_helm repository



roles > k8s_helm > files > chart-values > velero-values.yml

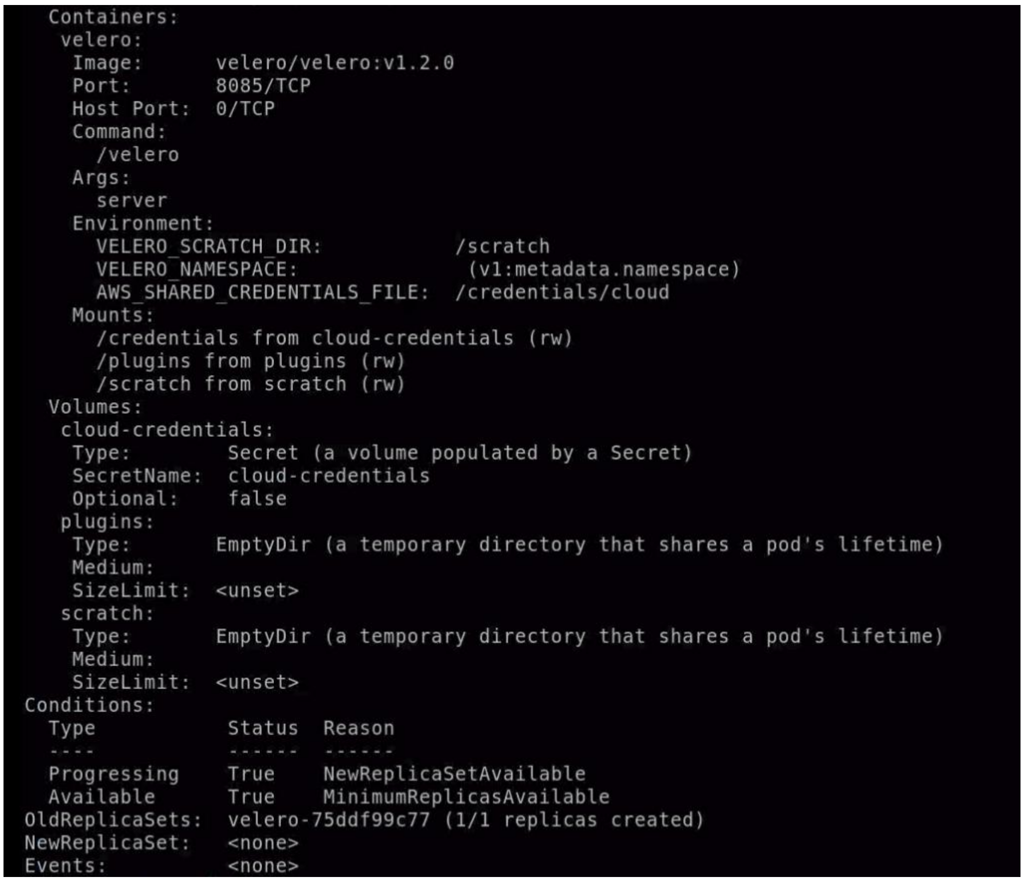
```
initContainers:
  - name: velero-plugin-for-aws
    image: velero/velero-plugin-for-aws:v1.0.0
    imagePullPolicy: IfNotPresent
    volumeMounts:
      - mountPath: /target
        name: plugins

metrics:
  serviceMonitor:
    enabled: true
    additionalLabels:
      release: prometheus-operator

configuration:
  provider: aws
  backupStorageLocation:
    name: aws
    bucket: velero-backup-1504-io
    config:
      region: eu-west-1
  volumeSnapshotLocation:
    name: aws
    config:
      region: eu-west-1

serviceAccount:
  server:
    name: velero

credentials:
  existingSecret: cloud-credentials

deployRestic: true
```

Figure 13 Velero configuration

4.3.2 Testing

Once every component was creating following the team’s protocol and the solution’s documentation. Velero was deployed to the cluster for testing using the flowing command: “ansible-playbook -i inventory/k8s.1504.io/hosts k8s-cluster.yml”

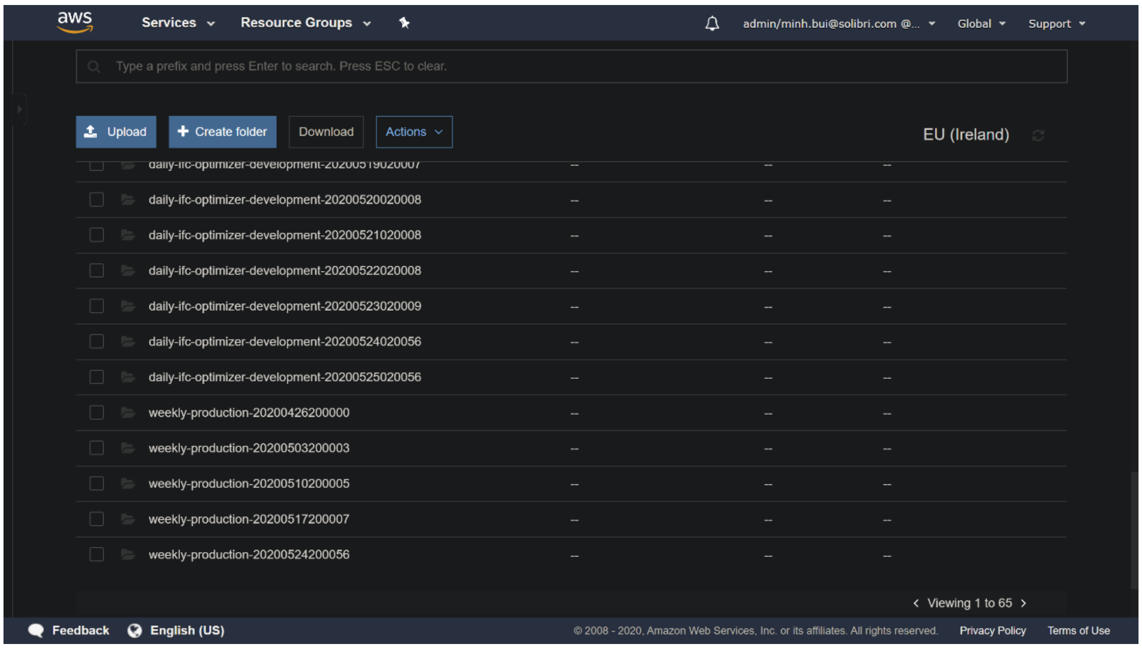
This command told Ansible to run the deployment playbook against every nodes listed in the inventory. Once the playbook has finish running, the deployed application information can be checked using the kubectl component in Kubernetes against the “velero” namespace. If the namespace was created unsuccessfully, an error would occur. On the other hand, successfully deploying the solution should be shown information about its container. The following commands was used to check for the needed information: “kubectl -n velero describe deploy velero”. Figure 14 shows the outcome of running the command through terminal. The figure also shows that a positive result. The application is running on a container with the correct image version.



```
   Containers:
    velero:
     Image:      velero/velero:v1.2.0
     Port:       8085/TCP
     Host Port:  0/TCP
     Command:
       /velero
     Args:
       server
     Environment:
       VELERO_SCRATCH_DIR:           /scratch
       VELERO_NAMESPACE:              (v1:metadata.namespace)
       AWS_SHARED_CREDENTIALS_FILE:  /credentials/cloud
     Mounts:
       /credentials from cloud-credentials (rw)
       /plugins from plugins (rw)
       /scratch from scratch (rw)
   Volumes:
    cloud-credentials:
     Type:        Secret (a volume populated by a Secret)
     SecretName:  cloud-credentials
     Optional:    false
    plugins:
     Type:       EmptyDir (a temporary directory that shares a pod's lifetime)
     Medium:
     SizeLimit:  <unset>
    scratch:
     Type:       EmptyDir (a temporary directory that shares a pod's lifetime)
     Medium:
     SizeLimit:  <unset>
 Conditions:
   Type           Status  Reason
   ----           ------  ------
   Progressing    True    NewReplicaSetAvailable
   Available      True    MinimumReplicasAvailable
 OldReplicaSets:  velero-75ddf99c77 (1/1 replicas created)
 NewReplicaSet:   <none>
 Events:          <none>
```

Figure 14 Kubectl command result

Next, Velero’s backup functionality needs to be tested and 2 schedule backups was created. The daily backup is for day-to-day development of a software running on the infrastructure. The weekly backup is used for storing the manifest of the entire cluster. After running the necessary commands to schedule these backups, the files can be view from AWS Web console as shown in figure 15. From screenshot, the schedule backups can clearly be seen. This indicates that Velero is triggering backups to be created correctly.

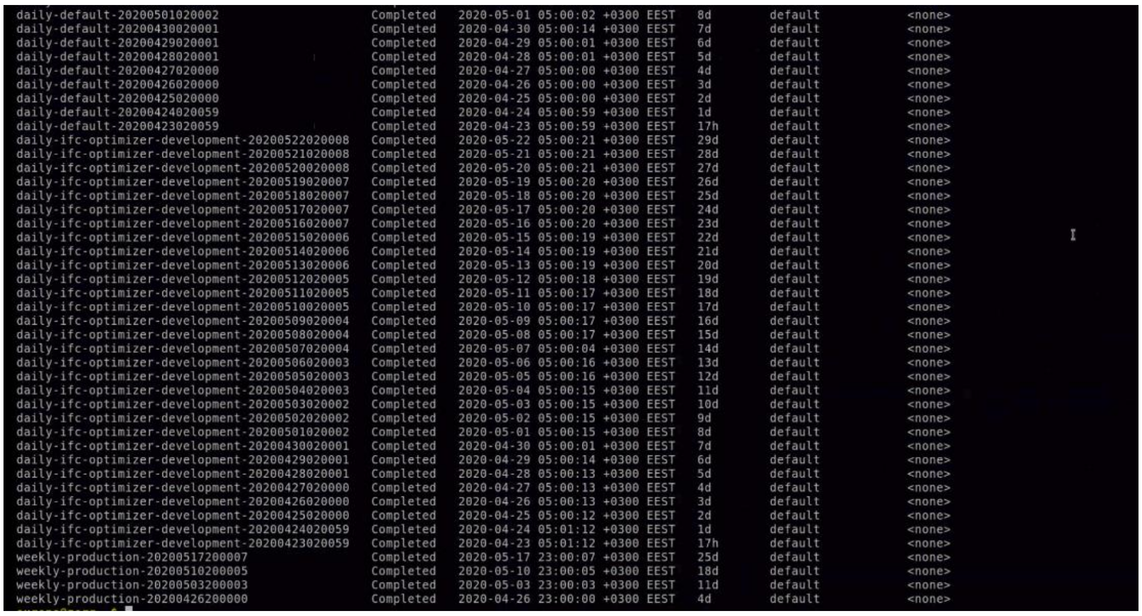


```
daily-ifc-optimizer-development-20200519020007      --      --      --
daily-ifc-optimizer-development-20200520020008      --      --      --
daily-ifc-optimizer-development-20200521020008      --      --      --
daily-ifc-optimizer-development-20200522020008      --      --      --
daily-ifc-optimizer-development-20200523020009      --      --      --
daily-ifc-optimizer-development-20200524020056      --      --      --
daily-ifc-optimizer-development-20200525020056      --      --      --
weekly-production-20200426200000                    --      --      --
weekly-production-20200503200003                    --      --      --
weekly-production-20200510200005                    --      --      --
weekly-production-20200517200007                    --      --      --
weekly-production-20200524200056                    --      --      --
```

© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Figure 15 AWS management console

Finally, the most important aspect of the solution needs to be tested. Engineers needs to be able to retrieve the backups from its storage. The built-in command “velero get backup” should retrieve the closet backups from the S3 buckets. After executing the command, results can see in the terminal screen shot in figure 16. The terminal interface shows that retrived files match the files in the AWS console. Therefore, this test has yield positive results in favour of Velero successful deployment.



```
daily-default-20200501020002                     Completed   2020-05-01 05:00:02 +0300 EEST   8d    default   <none>
daily-default-20200430020001                     Completed   2020-04-30 05:00:14 +0300 EEST   7d    default   <none>
daily-default-20200429020001                     Completed   2020-04-29 05:00:01 +0300 EEST   6d    default   <none>
daily-default-20200428020001                     Completed   2020-04-28 05:00:01 +0300 EEST   5d    default   <none>
daily-default-20200427020000                     Completed   2020-04-27 05:00:00 +0300 EEST   4d    default   <none>
daily-default-20200426020000                     Completed   2020-04-26 05:00:00 +0300 EEST   3d    default   <none>
daily-default-20200425020000                     Completed   2020-04-25 05:00:00 +0300 EEST   2d    default   <none>
daily-default-20200424020059                     Completed   2020-04-24 05:00:59 +0300 EEST   1d    default   <none>
daily-default-20200423020059                     Completed   2020-04-23 05:00:59 +0300 EEST   17h   default   <none>
daily-ifc-optimizer-development-20200522020008   Completed   2020-05-22 05:00:21 +0300 EEST   29d   default   <none>
daily-ifc-optimizer-development-20200521020008   Completed   2020-05-21 05:00:21 +0300 EEST   28d   default   <none>
daily-ifc-optimizer-development-20200520020008   Completed   2020-05-20 05:00:21 +0300 EEST   27d   default   <none>
daily-ifc-optimizer-development-20200519020007   Completed   2020-05-19 05:00:20 +0300 EEST   26d   default   <none>
daily-ifc-optimizer-development-20200518020007   Completed   2020-05-18 05:00:20 +0300 EEST   25d   default   <none>
daily-ifc-optimizer-development-20200517020007   Completed   2020-05-17 05:00:20 +0300 EEST   24d   default   <none>
daily-ifc-optimizer-development-20200516020007   Completed   2020-05-16 05:00:20 +0300 EEST   23d   default   <none>
daily-ifc-optimizer-development-20200515020006   Completed   2020-05-15 05:00:19 +0300 EEST   22d   default   <none>
daily-ifc-optimizer-development-20200514020006   Completed   2020-05-14 05:00:19 +0300 EEST   21d   default   <none>
daily-ifc-optimizer-development-20200513020006   Completed   2020-05-13 05:00:19 +0300 EEST   20d   default   <none>
daily-ifc-optimizer-development-20200512020005   Completed   2020-05-12 05:00:18 +0300 EEST   19d   default   <none>
daily-ifc-optimizer-development-20200511020005   Completed   2020-05-11 05:00:17 +0300 EEST   18d   default   <none>
daily-ifc-optimizer-development-20200510020005   Completed   2020-05-10 05:00:17 +0300 EEST   17d   default   <none>
daily-ifc-optimizer-development-20200509020004   Completed   2020-05-09 05:00:17 +0300 EEST   16d   default   <none>
daily-ifc-optimizer-development-20200508020004   Completed   2020-05-08 05:00:17 +0300 EEST   15d   default   <none>
daily-ifc-optimizer-development-20200507020004   Completed   2020-05-07 05:00:04 +0300 EEST   14d   default   <none>
daily-ifc-optimizer-development-20200506020003   Completed   2020-05-06 05:00:16 +0300 EEST   13d   default   <none>
daily-ifc-optimizer-development-20200505020003   Completed   2020-05-05 05:00:16 +0300 EEST   12d   default   <none>
daily-ifc-optimizer-development-20200504020003   Completed   2020-05-04 05:00:15 +0300 EEST   11d   default   <none>
daily-ifc-optimizer-development-20200503020002   Completed   2020-05-03 05:00:15 +0300 EEST   10d   default   <none>
daily-ifc-optimizer-development-20200502020002   Completed   2020-05-02 05:00:15 +0300 EEST   9d    default   <none>
daily-ifc-optimizer-development-20200501020002   Completed   2020-05-01 05:00:15 +0300 EEST   8d    default   <none>
daily-ifc-optimizer-development-20200430020001   Completed   2020-04-30 05:00:01 +0300 EEST   7d    default   <none>
daily-ifc-optimizer-development-20200429020001   Completed   2020-04-29 05:00:14 +0300 EEST   6d    default   <none>
daily-ifc-optimizer-development-20200428020001   Completed   2020-04-28 05:00:13 +0300 EEST   5d    default   <none>
daily-ifc-optimizer-development-20200427020000   Completed   2020-04-27 05:00:13 +0300 EEST   4d    default   <none>
daily-ifc-optimizer-development-20200426020000   Completed   2020-04-26 05:00:13 +0300 EEST   3d    default   <none>
daily-ifc-optimizer-development-20200425020000   Completed   2020-04-25 05:00:12 +0300 EEST   2d    default   <none>
daily-ifc-optimizer-development-20200424020059   Completed   2020-04-24 05:01:12 +0300 EEST   1d    default   <none>
daily-ifc-optimizer-development-20200423020059   Completed   2020-04-23 05:01:12 +0300 EEST   17h   default   <none>
weekly-production-20200517200007                 Completed   2020-05-17 23:00:07 +0300 EEST   25d   default   <none>
weekly-production-20200510200005                 Completed   2020-05-10 23:00:05 +0300 EEST   18d   default   <none>
weekly-production-20200503200003                 Completed   2020-05-03 23:00:03 +0300 EEST   11d   default   <none>
weekly-production-20200426200000                 Completed   2020-04-26 23:00:00 +0300 EEST   4d    default   <none>
```

Figure 16 Velero command result

5 Conclusion

As technology and customers’ demand grows. It is important for companies such as Solibri to integrate new solution to build better products. However, with development methods and infrastructure always brings new challenges. This thesis project was created to help the company built a more resilient architecture by implementing the cluster backup solution: Velero.

This solution is meant to help DevOps engineers to recover the cluster in case of entire architecture termination. Through working closely with the DevOps teams, following their protocol, learning and implementing modern tools, the author managed to implement the backup solution. Within this project, the author had used knowledge on several subjects. These concepts include containerized application architecture, what resilience infrastructure is and how to use different DevOps tools for different tasks. Each of these concepts plays a different but equally valuable part throughout the project. These subjects, especially the functionality of DevOps tools, where used to prepare, configure and deploy Velero into the cluster. Additionally, testing procedure were conducted through using this software as well. Using the Kubernetes built-in management tool (kubectl), the deployed container information could be seen from the command line. This showed that the application was deployed successfully. Using Velero, some backups were created to act as restore points in case of disaster. These backups can be seen through the AWS web console, which shows that the backups exist in the correct storage space. Finally, backups can also be retrieved through command line. These indicates that the backups are ready whenever needed. These testing results shows that Velero was integrated to the cluster and the backup solution is ready to be used. This project has provided means of backing up any needed data from the cluster. However, not every aspect of disaster recovery was dealt with. The pipeline in which the cluster can be restored is not in place. These subjects need to be addresses in later project to build a fully resilient infrastructure for Solibri's future development.

References

Printed

Dawson, C. 2009. Introduction to Research Methods: a practical guide for anyone undertaking a research project.

Electronic

Ansible documentation
https://docs.ansible.com/ansible/latest/index.html

Ansible VS Kubernetes
https://www.simplilearn.com/ansible-vs-kubernetes-article

Construction industry spending worldwide from 2014 to 2025 (in trillion U.S. dollars). Statista.com. Accessed March 2020:
https://www.statista.com/statistics/788128/construction-spending-worldwide/

Container based Architectures I/III: Technical advantages. Pablo I. 13th July 2017. Accessed March 2020:
https://medium.com/@pablo.iorio/container-based-architecture-i-iii-technical-advantages-7176195456c5

DevOps principle practices and DevOps engineer role. AltexSoft. 2020. Accessed March 2020:
https://www.altexsoft.com/blog/engineering/devops-principles-practices-and-devops-engineer-role/

Patterns for scalable and resilient apps. Google. Accessed March 2020:
https://cloud.google.com/solutions/scalable-and-resilient-apps

What is Infrastructure as code. Dadgar A. 20th August 2018. Accessed March 2020:
https://www.hashicorp.com/resources/what-is-infrastructure-as-code/

What is Kubernetes cluster. Red Hat Inc. 2020. Accessed March 2020:
https://www.redhat.com/en/topics/containers/what-is-a-kubernetes-cluster

What is Kubernetes. The Kubernetes Authors. 2020. Accessed March 2020:
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

What is a Helm Chart – A beginner's guide. Foster R. 6th August 2019. Accessed April 2020:
https://www.coveros.com/what-is-a-helm-chart-a-beginners-guide/

YAML Tutorial: Everything You Need to Get Started in Minutes. Goebelbecker E. 11th December 2018. Accessed March 2020:
https://rollout.io/blog/yaml-tutorial-everything-you-need-get-started/

YAML. Wikipedia contributors. Last updated 20th May 2020. Accessed March 2020:
https://en.wikipedia.org/wiki/YAML

Figures