UPPSALA UNIVERSITET

UPTEC IT 20004
Examensarbete 30 hp
Februari 2020

Guidelines to Convert Legacy Applications into Docker Container Services

Albin Sundqvist

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Guidelines to Convert Legacy Applications into Docker Container Services

Albin Sundqvist

In the past people would buy a program on a disc and install it on their local computer or server. Today, cloud computation allows users to access their applications over the Internet while also providing them with greater computational speed and storage possibilities. But moving existing legacy application into a cloud environment can be difficult. The goals of this master thesis is to create guidelines that describe how the move should be executed, and evaluate the benefits of moving legacy applications into a containerized environment using Docker and Kubernetes.

The work for developing the guidelines was divided into iterations to incrementally build better understanding and allow step-by-step development of the method, discovering and focusing on a few new things each iteration and by doing research in that area. Two applications were used and apart from this, research about general practices also helped develop the method.

Research was conducted to find out what key aspects to consider, and what problems and risks to be aware of, from a general stand point when moving an application to a cloud environment. The results are general guidelines on how to move a legacy application into Kubernetes, the advantages it has and which parameters correlate to these advantages.

Handledare: Max Block
Ämnesgranskare: Salman Toor
Examinator: Lars-Åke Nordén
UPTEC IT 20004
Tryckt av: Reprocentralen ITC

Contents

1 Introduction **4**

2 Background **6**

2.1 Cloud Services 6

2.2 Cloud Deployment 7

2.3 Containerization vs Virtual Machines 8

2.4 Orchestration frameworks 9

2.5 Solutions to move legacy applications to the cloud 13

2.5.1 Redevelopment 14

2.5.2 Migration 14

3 Opportunities and Drawbacks with Cloud Technology **15**

3.1 Advantages of Cloud Technology 15

3.1.1 Scalable Applications 15

3.1.2 Updates 17

3.1.3 Manageability 18

3.2 Disadvantages of Cloud Technology 19

4 Development of method **20**

4.1 Service Desk Plus 20

4.2 EIdentificationAPI 21

5 Moving Legacy Applications to the Cloud **22**

5.1 Questions 22

5.2 Course of Action 24

6 Future work and Discussion **30**

7 Conclusion **31**

8 References **32**

1 Introduction

In the past people would buy a program on a disc, or download a program, and install it on their local computer or server. Today, cloud computation allows users to access their applications over the Internet while also providing them with greater computational speed and storage possibilities. N. Serrano, G. Gallardo, and J. Hernantes [31] defines cloud computing like this: "Cloud computing is 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." This is the US National Institute of Standards and Technology (NIST) definition of cloud computing[26].

Moving applications into a cloud environment has many benefits such as scalability, flexibility and low cost[17]. Today companies have cloud compatibility in mind when developing new applications[11], but when companies have legacy applications, problems with manageability can occur, for example when different programs have conflicting dependences, see Section 3.1.3.

Legacy application A legacy application is an application that is based on previous technology and the main development effort is over. Legacy applications often reflect the decisions made based on the tools that were available at the time and the programming language used for developing the application. The application is still working and in production.

Maintaining these outdated applications consumes more resources and time than any part of the software life cycle[35]. Moving legacy applications into the cloud can be hard, as you often need to rewrite them in order to support multi-tenancy[23]. Other changes may also be needed for the migration to a cloud. Containerization technology makes it is possible to support multi-tenancy, and with multi-tenancy you can host multiple applications on a single host.

Containerization With containerization technology it is possible to divide the application into smaller parts, called *microservices*[28], and also to run the applications isolated on the same host.

Being able to isolate the applications makes it possible for applications with conflicting dependencies to build and run on the same server. Many legacy applications have special conditions and complicated configurations that make it hard for other applications to run on the same machine. Being able to wrap these application inside a container containing all dependencies and executables, has many advantages. This is explained in more detail in Section 3.1.3. A container containing the application make the application portable, allowing it to be executed on any machine[33].

Goal The goals of this master thesis is to evaluate the benefits of moving legacy applications into a containerized environment using Docker, and to create guidelines that describe how the move should be executed. How containers are handled by *Kubernetes*[32], a container orchestration service, is also considered in the guidelines. The task also includes to find out what parameters are useful for the move to the cloud.

The work has been conducted in collaboration with *Data Ductus*, an IT consulting firm specializing in technically advanced solutions in Telecom, Enterprise IoT and Cloud Management[2].

2 Background

In this section the general cloud aspects are described more specific areas within cloud technology such as containerization, orchestration frameworks and solutions for moving an application to the cloud.

2.1 Cloud Services

The three major services in cloud computing that the providers use are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and lastly Software as a Service (SaaS)[12].

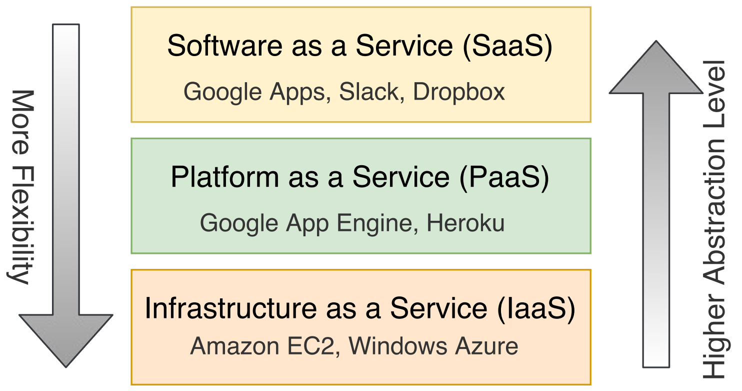
More Flexibility
Software as a Service (SaaS)
Google Apps, Slack, Dropbox
Platform as a Service (PaaS)
Google App Engine, Heroku
Infrastructure as a Service (IaaS)
Amazon EC2, Windows Azure
Higher Abstraction Level

Figure 1: Overview of different cloud services.

IaaS refers to the services for delivering resources such as processing, storage and network. IaaS is the most basic but the most flexible platform[14]. This enables the customers to run any software and operating system in the cloud. IaaS requires more managing than PaaS and SaaS because you are forced to manage the whole system from infrastructure and up to software, but having it in the cloud still allows for scaling, see Section 3.1.1.

PaaS is aimed to ease the developers frequent demand to push new updates and develop new features[10]. PaaS is a platform that includes programming language support and tools for developing an application. Just like SaaS, it is the cloud provider that supports and maintains everything so all the different tools are up-to-date and working.

SaaS is a model that gives the users access to the applications that would have previously been installed or hosted on their local device through the Internet. Users use a thin client interface, for example a web browser or a program interface, to access the service or application. The cloud provider is responsible for managing and updating the infrastructure, operating system (OS) and software so that the applications runs properly. This makes it possible for users with different environments to access and run their application anywhere just from an web browser and Internet connection. Every user that connects with the cloud service will use the same environment as everyone else on the cloud, thus facilitating maintenance and support.

2.2 Cloud Deployment

The cloud providers also have different deployment methods. Three of these methods are *public cloud*, *private cloud* and *hybrid cloud*, see Figure 2.

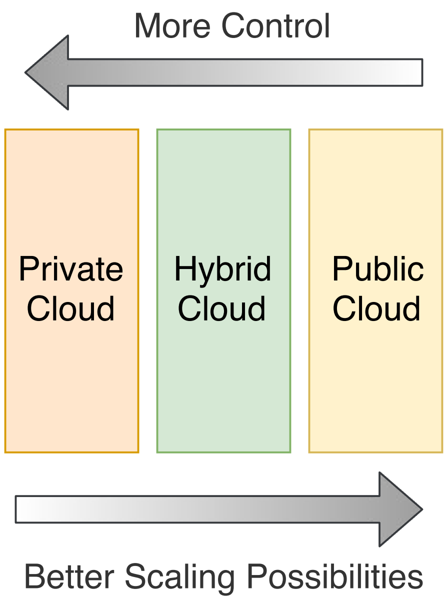

Figure 2: Overview of different cloud deployment methods.

The public cloud brings vast resources and computation but gives less control. This is because it is based on shared hardware provided and maintained by a third-party. If the data being handled is sensitive, then the public cloud might be a risk. Public cloud have many risk areas, many of them because of outsourcing[29]. This is explained more in detail in Section 3.2. For startups the public cloud option is still often very viable since it is a large commitment to buy and maintain a data center but with the public cloud you can scale up and down as demands fluctuates.

A private cloud is when the infrastructure is completely dedicated to the company. This provides greater levels of control and security but is a greater commitment. To some companies it is essential to have a private data center because all or some of their data might be very critical for the company or its customers. If only parts of the data is security sensitive, a hybrid cloud is a good option.

A hybrid cloud solution is where the company mix a private cloud with a public cloud. It could be that the they need higher security for critical data or that they have a burst of CPU demanding tasks, for example if the company receive some result or a big dataset they need to do calculation on. If they only work with the processing powers on premise, the result will come back slow. This, in turn, can slow down the business, so offloading the work to a cloud will increase computation speed. In the example the calculation must be able to be divided into several smaller parts for the computation speed to be increased.

2.3 Containerization vs Virtual Machines

Containerization Containers isolate the software it runs from other processes on the host and can be launched by running an image. The image is an executable package with all the dependencies and libraries needed to run the application as well as the executable code for the application. Unlike Virtual Machines (VM), containers do not need to use hypervisor and a guest Operating System (OS). A Hypervisor is software that creates and runs virtual machines. Containers use a persistent process that handles containers to communicate with the host kernel. For example when using Docker, a Docker Engine is used to communicate with the host OS, reducing the overhead of host platform, see Figure 3. All this makes the containers more light weight than a regular VM.

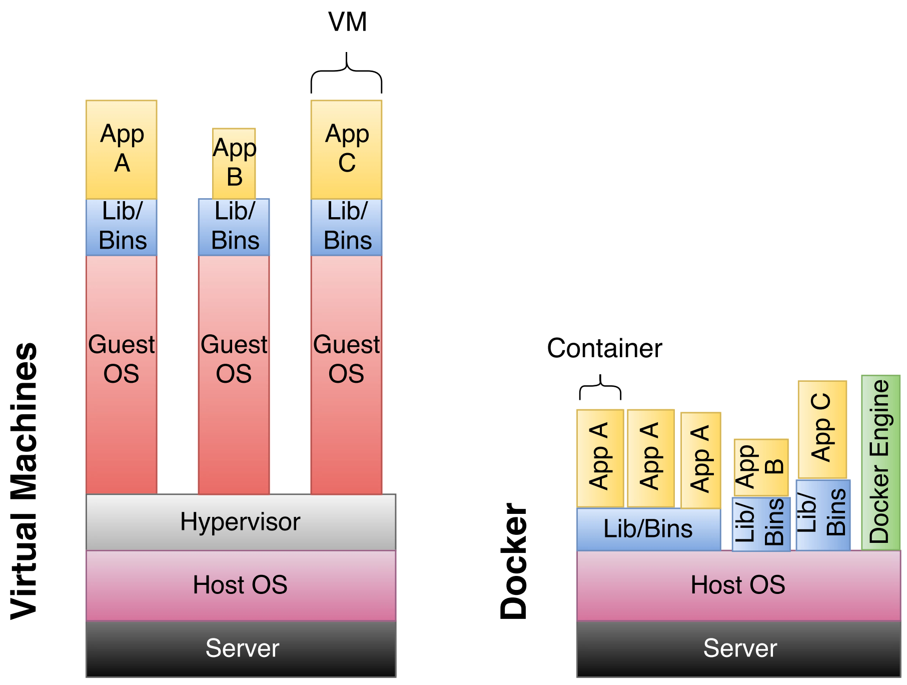

Figure 3: Multi-tenancy in a Virtual Machine (VM) vs Docker (Containers).

Virtual Machines A VM uses hypervisor and have a guest OS on top of the host infrastructure, which often allocates more resources than the application actually needs, and because the OS images are large the start up is slow, from one to ten minutes[24]. The lightweight nature of containers makes it possible for several more containers to run on a single server rather than if the VMs would run simultaneously[21]. Because containers do not need to start an entire OS when they spawn they become light weight and very flexible, making it possible to scale up or down fast, often in seconds[25].

Docker Docker is a computer program that performs containerization which means operating-system-level virtualization. Docker uses some features of the Linux kernel such as control groups (cgroups), kernel namespaces, and a union-capable file system[33]. This allows Docker to run applications completely separately on the same kernel[3]. As mentioned earlier, Docker can be used for multi-tenancy. The Docker platform is used to build and run containers; to schedule and scale these containers an orchestration program can be used.

With containerization technology it is also possible to divide the application into smaller parts, called *microservices.* Dividing the application into microservices have several advantages[27]: It is possible to scale up the specific part that is under stress as the demand fluctuates during the day, making the end user able to use the application as usual even at times when the service has very high traffic. Companies can also save money being able to scale the hardware the application runs on depending on how the load changes during the day, rather than buying the servers to run the applications and maintaining them. Armbrust et al.[17] states that "using 1000 servers for one hour costs no more than using one server for 1000 hours.". Another advantage of dividing an application into several microservices is that each microservice only is responsible for one specific thing. This will often result in better maintainability and comprehensiveness of the code[20]. Everything comes with a trade off; moving to the cloud and splitting the application in to ideal modules costs money, but in long term the initial fixed cost will be overshadowed by the cost for maintaining the application[23, 17].

2.4 Orchestration frameworks

Now when organizations deploy containers more widely and the use for containers increases, the need to manage these containers also grow[9]. Container orchestration frameworks assist companies to deploy and manage containers. The frameworks aim to simplify container management and determine initial container deployment but also to manage varying amounts of containers as one entity, for example for the purposes of example availability, scaling, and networking. There are many frameworks that exist and offers various feature sets. Docker Swarm and Mesosphere Marathon are two example of these orchestration frameworks, Kubernetes is another and the one used in this project.

Kubernetes

Kubernetes is an orchestration service that handles containers[32]. Kubernetes is an open-source platform which scale, manage and automate deployment for containerized applications. These characteristics align with cloud technology and the benefits the cloud provides, see 3.1, making Kubernetes a great fit for when deploying and managing applications in a cloud environment.

In the following sections comes an explanation of the Kubernetes entities, starting from the smallest and then moving out to the largest.

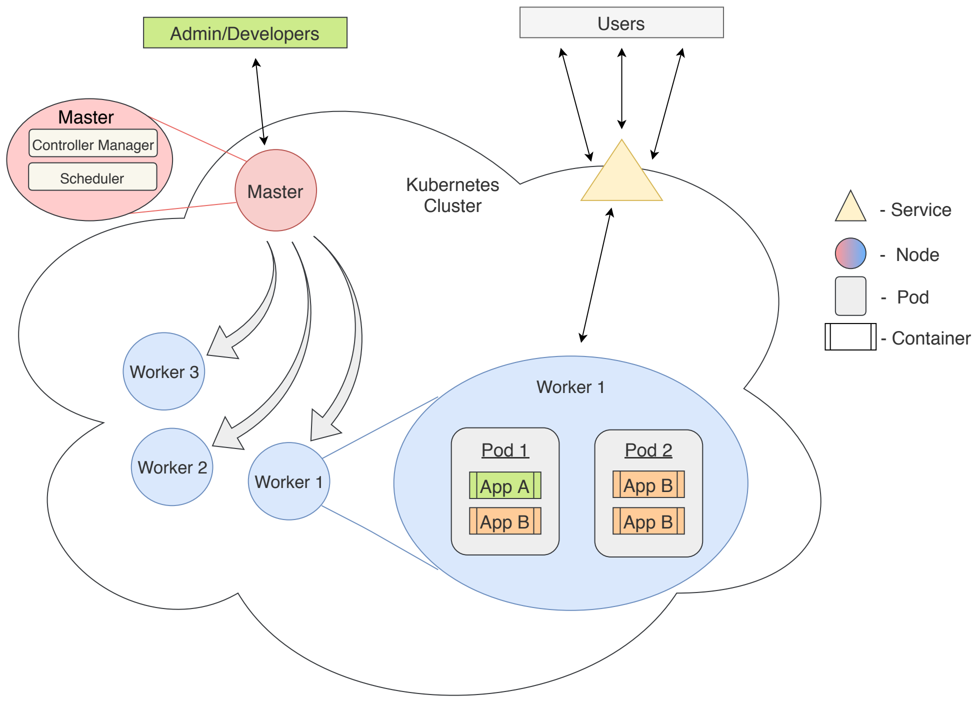

Figure 4: Simplified view of the Kubernetes cluster.

Pods Applications run within containers, and each container is, in turn, hosted in a pod. To simplify, the pod is like a user on a computer. One container per pod is often the case, but tightly coupled applications can share a pod, for example if they are latency sensitive. When containers share pod they can also share the file-system, using volumes, see Volumes. They can also communicate with each other using, for example, localhost. All pods within the cluster will have a unique IP address, making them reachable to every other pod in the cluster[13]. The pods in Kubernetes are like containers in the way that they are mortal, meaning that when they die they are not resurrected, instead a new pod spawns.

Services While each pod has its own IP address, that IP address is not stable over time. A service defines a logical set of pods and a policy to access them, sometimes called micro-service. The service uses labels to group the pods it wants to expose. When someone calls the service, it acts as a Load Balancer for all the pods that has that specific label. This could be both from outside of the cluster but also services for within the cluster.

Node Each pod is hosted in a node and there can be several pods within a node. One can think of a node as a machine, physical or virtual; pods like users on that computer and containers as the processes the users run. All the nodes exists in a cluster, where you have at least one master node. The master node is the entry point for developers and admins when they need to communicate with the cluster using either an API, UI or CLI.

Volumes In Kubernetes there are different types of volumes. A regular volume solves the problem with containers not being able to retrieve the data from a container when it crashes. A volume also makes it possible for the different containers that run within a pod to share files between each other. When the pod terminates, the data is then lost. To solve the problems of a regular volume, a *PersistentVolume* (PV) is needed. The PV subsystem provides an API for administrators and users that abstracts the details of how storage is provided from how it is consumed independent from the cloud provider. Two resources are used, PV and *PersistentVolumeClaim* (PVC). A PV is similar to a regular volume but instead of existing in the Pod the volume is stored externally, yet still in the cluster, see Figure 6, making its life cycle independent of any pod that uses the PV. The PV is a resource just like a node in the cluster. PVC is a request for storage by the users and administrators. Like pods, PVC consumes cluster resources. Pods consume node resources and PVCs consume PV resources.

Deployment When the developers want to deploy a new application. They create a deployment that states they need three nodes with two pods and that the pods should host two containers each, one App A and three B. The deployment is sent to the master node. The master receives the deployment and has the task of keeping this configuration of the deployment required. This means that the *Controller Manager* spawns the worker nodes and pods, allowing the *Scheduler* to assign the pods to the nodes. It is the Controller Manager that is responsible for that the deployment requirements are satisfied, for example if a node or pod fails the Controller Manager starts a new one. Because of this Kubernetes allows the application to have zero down time[5, 32], for example when updating to a new release of the application. The Controller Manager always makes sure that the application is running on the specified amount of pods when an update is deployed. Kubernetes updates one pod at the time, killing the old pod and replacing it with the new updated version.

State A *Stateless Application* is one which does not depends on persistent storage. Different pods in the cluster can work independently with multiple requests at the same time. If something goes wrong, the application can be restarted and it will go back to the initial state with little downtime.

A Stateful Application is an application that stores some type of state, this can for example be data of a user. This leads to a pod needing at least one volume.

Persistent storage and Databases Whether data needs to be stored for a longer period of time or just during execution, data storage is important. The data can be stored directly in the containers local file system, a database or in a volume. Both the database and the volume can be located externally and do not have to exist on the same machine as the container the application is running in.

When an application needs to save data that it is creating or handling over the timespan of its existence, some sort of persistent memory is needed. For example, Figure 5 represents an application running inside a container where the database is running inside as well. The application can run as normal and retrieve and store data in the database. When the container goes down, restarts or fails, the data is lost. If however, the application uses a volume, the volume can be mounted on several containers and if a container goes down within the pod, then a new container can be spawned and use the data from the volume, see Figure 6. This makes it possible for the containers within the pod to share files between each other. However the volume exists only in the pod and when the pod goes down, like containers, all the data is lost. The container does not have to use a database or a volume to store data, only if the container needs to save or use the data during its own existence. If that is not the case then using the local file system on the container will suffice.

In Figure 6 the database and the application is decoupled and the database is using a PV. This is the ideal case for a deployment, as it allows for example the administrators to perform updates on the database as well as the application with zero down time, see Section 3.1.2. Decoupling the applications and the database that the application is using are also good practice, making it easier to update or restart each individual entity without effecting the other one. Although the code becomes more understandable and easier to maintain, the code might run slower and require more memory[27].

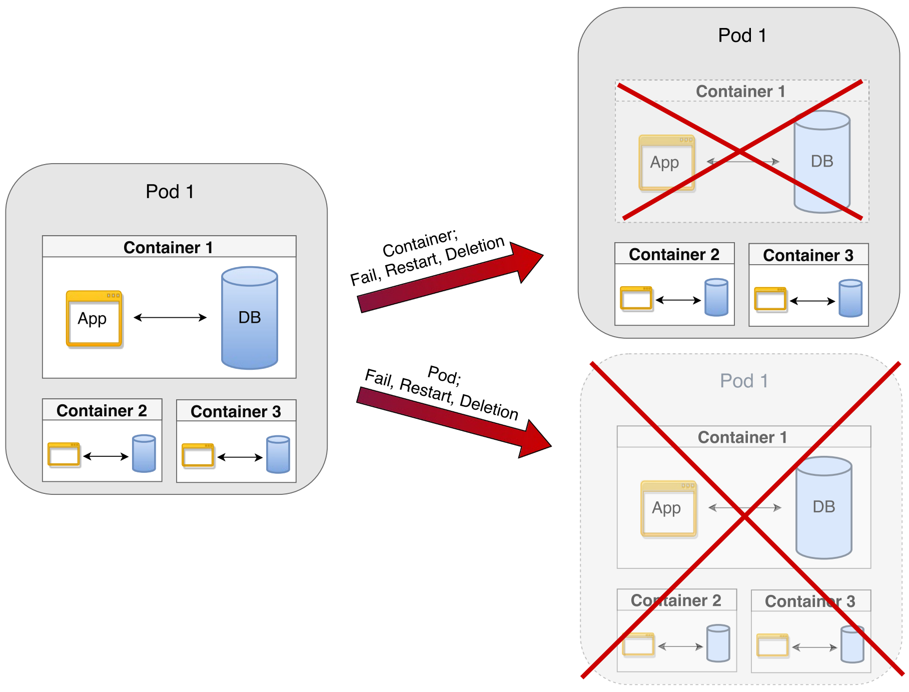

Figure 5: When a container or Pod is removed, restarted or has a failure the data is lost

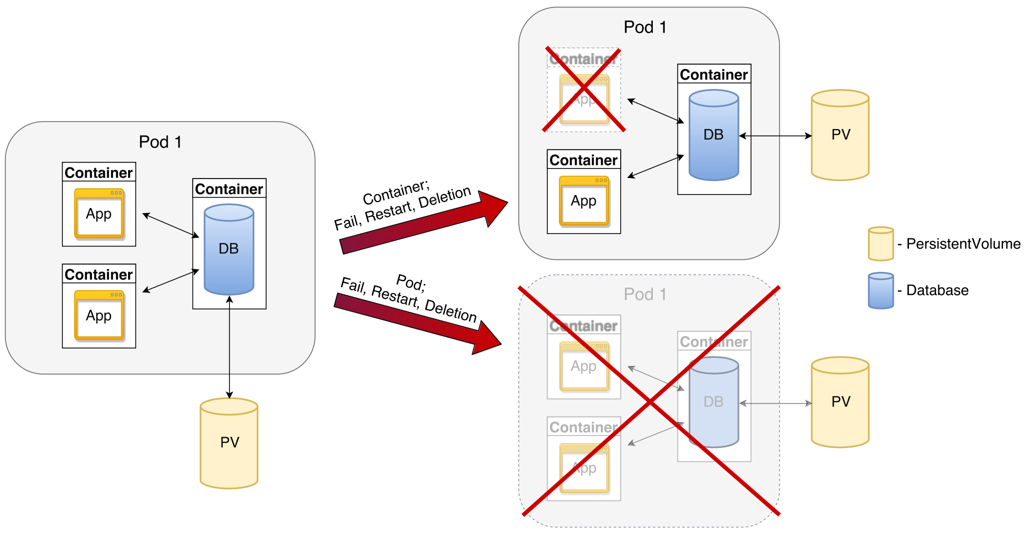

Figure 6: When a container or pod goes down the data is still saved in a PersistentVolume

2.5 Solutions to move legacy applications to the cloud

There are many ways to move a legacy application to a cloud environment, but can often be separated into two different categories: *redevelopment*, where the application is rewritten in the new environment; and *migration* which moves the legacy application to the cloud and at the same time keeps all the functionality and current structure.

2.5.1 Redevelopment

Redevelopment focuses on reworking the legacy application from scratch using new and modern tools, design and architecture. The goal is to transform the current centralized environment to a distributed one. One way to approach this is to recognize the different features and processes of the legacy application and develop them incrementally, making it possible to move bit by bit over to the cloud. There are several approaches to this, for example model-driven development, software product line and software pattern[18]. This approach will also make it possible to enable elasticity and multi-tenancy in the application, since the application can be built from scratch with a microservice mindset. This approach can cause problems because of the nature of some legacy applications, being that legacy applications can be hard to decouple, making the newly developed application modules tangled and hard to maintain, just like the legacy application. Redevelopment is also very time consuming and with the risk of failing, many originations choose other ways to migrate their legacy application[15].

2.5.2 Migration

Migration is when you move the entire legacy application, scoop it up and put it in a cloud environment. Doing this helps the developers manage the legacy application, see Section 3.1.3. Migration makes the legacy applications behave like SaaS, see Section 2.1. By performing a migration users can access the legacy application, which previously existed on premise, and now it can be accessed independent of location with only an Internet connection and at the same time make it easier for the developers to manage the application.

3 Opportunities and Drawbacks with Cloud Technology

Cloud technology has enabled a world full of new opportunities, especially with the networking society and 5G[30]. While this cloud technology have both many advantages it also has disadvantages that should be considered.

3.1 Advantages of Cloud Technology

Cloud technology has many advantages compared to on premise hosting, a common and well known one is **cost savings**. Cloud technology allows companies to have zero in-house servers, storage and application requirements. "Pay for what you use" is a common mantra in cloud technology, meaning if you need to have three servers running in the afternoon and only two servers the rest of the day, you only pay for the three servers in the afternoon and pay for two the rest of the day. Without cloud technology the company would have to buy three servers to handle the demand in the afternoon but only use it a fraction of the time of the day. This can be referred to as scaling, see Section 3.1.1. Without the servers running, costs for things like power, cooling and administration will reduce as well.

Another advantage with cloud technology is **reliability**. The cloud brings a vast amount of storage and computation capability, which means companies can use this for having redundancy with their applications. If a server has a failure or an update is due, another server can already be running and ready to take over the users, see Section 3.1.2.

Like mentioned before, **manageability** is also an advantage with the cloud, giving the user access to the software, application and services through a web-based user interface instead of having the user to install the software on their local computer or mobile phone. This makes sure that all of the dependencies are managed and that they do not intertwine with other programs that might have non-compatible dependencies. Also, instead of plugging in cables and setting up configuration physically this can now be done through software and with a click of a button. This makes the company able to focus more on developing the product, spend less money on managing dependency problems and more time with key business activities and objectives.

3.1.1 Scalable Applications

One of the great advantages of cloud technology is the *scaling* of computing resources up and down depending on the current load of the system[11]. Not only does it benefit the company providing the service, which not have to buy more servers and resources than necessary to meet the Service Level Agreement (SLA) requirements, but also giving the customer a better and faster experience.

Computation speed When an application, or parts of it, can be scaled up, the computation speed will increase. For example, if an application using an external module, in this case another container, which it offloads workloads to so it can reduce the amount of stress on the system and also at the same time increase computation speed. If the workload can not be divided and spread over several instances of that helping module, the result will be calculated at the normal rate.

One condition to make an application scalable is that the workload is available for distribution, making it possible for the application to spawn the appropriated amount of workers. Each of these spawned workers can handle a piece of the workload. In the Figure 7 example, each of the workers spawned will receive a task from the application. Once the work has been finished the result is sent back to the application. Each of the workers are responsible for only a piece of the total workload. The application will then put the result together and give it to the user or another application.

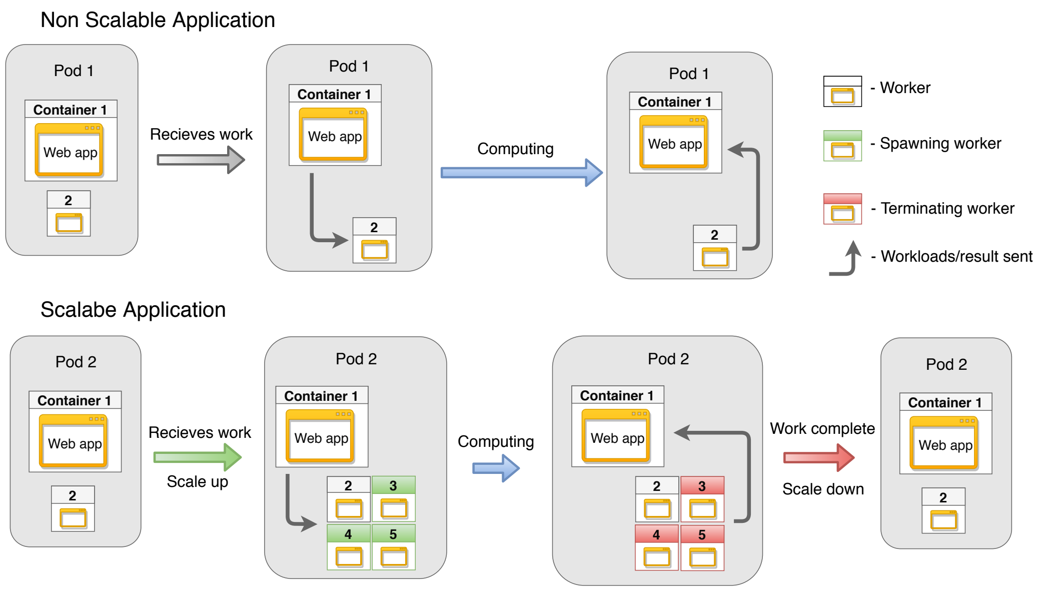

Figure 7: Non scalable application vs scalable application

This will lead to the application finishing the entire task almost as fast as each of the small workloads, with only the little overhead of dissembling the task and putting the results back together.

Varying load Scaling applications for heavy computations to go faster is not the only advantage with scaling in the cloud. Scaling can also make sure the application can handle all the requests from users when load increases. The example in Figure 4 in Section 2.4, one deployment is built with one application 'App A' and three sub applications called 'App B'. There are 3 replicas of that setup and a service so the users can access it from outside the cluster. When the load decreases the workers can be scaled down to only one or two workers, making the application work while at the same time costing the company less. Because they do not have to buy the hardware to support the load during peaks, they can instead rent the hardware in the cloud to provide the users with their services.

Utilization Another advantage is that because scaling is possible and with multi-tenancy, each server and CPUs can be higher utilized than when having the server locally. This making the cloud a greener alternative than local servers, which some of the time are underutilized.

When dealing with legacy applications some scaling can be difficult, for example doing computation speed upgrades since the applications are not often written for a distributed workload. To make an application scalable, one often needs to rewrite the code, which in some cases is not possible since no source code is available. Many legacy applications can still benefit from the scaling possibilities, for example when the load on the system goes up and it is possible to use horizontal scaling on the application. Horizontal scaling is when you scale up by adding more machines with the same software, making it possible for a load balancer to distribute the load from request to the newly spawned machines, which in turn makes it possible to serve more users and still meet the SLA.

3.1.2 Updates

Many of todays companies have started to use agile development or similar development methods. In the past products would typically have one or two updates every year[34]. The developers would often be stressed before an upcoming release date and bugs were common. Today, when using other development methods than plan-based the frequency of the updates can be improved, releasing multiple updates per day. But users today expect a service to be up and running every hour of every day, meaning that when an update is released users should still be able to use the service. Rolling update development is one possible way of doing that. In the example pictured in Figure 8, a rolling update is demonstrated on a Web application that has version 1.0 and is updated to version 2.0.

At least one of the two instances of the application is running at any given time, giving the user the ability to use the application even though it is currently being updated.

When dealing with legacy applications, pushing updates and developing the application are no longer the main focus. But the rolling update technique can still be beneficial when dealing with legacy applications. For example if a restart is necessary, it can do so without any downtime using the rolling update technique.

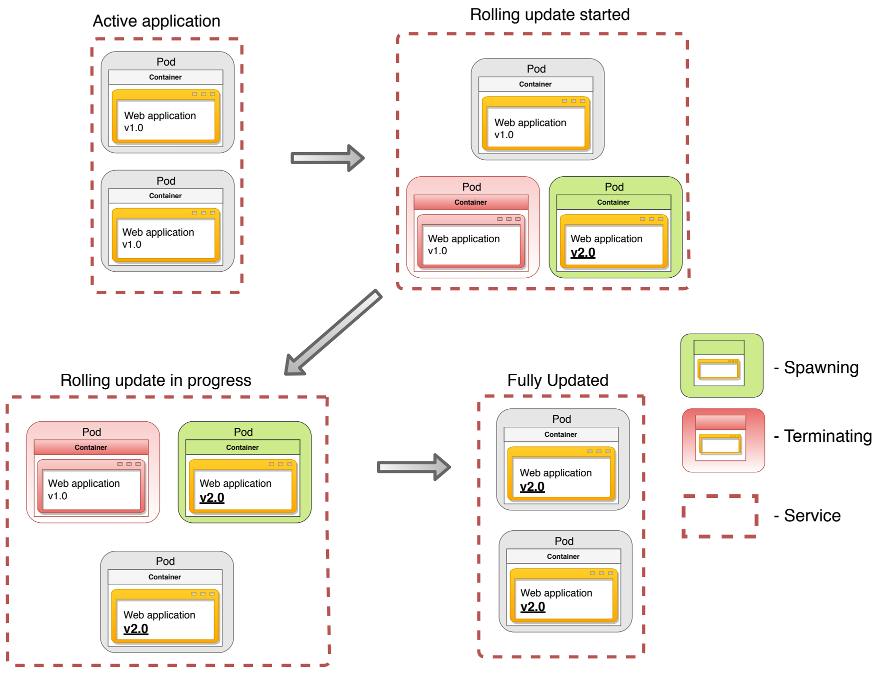

Figure 8: How a rolling update is preformed in Kubernetes

3.1.3 Manageability

Cloud computing refers to the ability to access applications through the Internet but also that the software and hardware is located in data centers that host these applications[19]. Depending on what type of cloud service and deployment methods, see Sections 2.1–2.2, manageability can look different for the same application. One key point with manageability in the cloud is that the software and hardware are located in the data centers, not residing on each user's personal devices, giving the developers an easier task when dealing with dependencies between programs. Legacy applications benefit from this especially because of their legacy nature, i.e the programs and dependencies that they have are most likely are outdated and not compatible with the applications written today. So if both the legacy application and a newly developed application reside on the same computer the applications would fail. Because of this, having the applications in the data center, isolated from all other programs and applications, will increase manageability. If for example a new user wants to use a legacy application a company is providing, the company does not have to set up the hardware. Instead of having to buy more new servers and install the hardware, they can instead have scripts that install certificates and licenses and have different configurations ready to go. By doing this, deploying the application to the user's managing the application is reduced significantly, and also the risk. Previously, companies had to consider if the investment in more servers was worth it, now they can scale up and down in seconds.

3.2 Disadvantages of Cloud Technology

Even though cloud technology has many advantages many things comes with a trade-off. These disadvantages have different levels of severity, in some cases business threatening, in other cases only minor flaws. Here are some examples of disadvantages with cloud technology.

Internet Connection One of the building blocks for cloud technology is the Internet. Without the Internet no one would be able to use the applications when they are in the cloud. Although Internet has become common for almost everyone in the world, the connection might not be stable at all times. In critical applications, losing the Internet connection can be devastating, which previously would not have mattered when the application was found locally on the same computer.

Data locality When using the public or hybrid cloud you borrow resources from another company, this borrowed resource can be located inside or outside your country. Moving your data outside the country can be a sensitivity issue. One example is from the Swedish Transport Agency in 2015[4], where they outsourced critical data to foreign countries in Europe, leading to a massive scandal on government level. When using outsourcing it is important to consider that other people and companies handle the data.

Specific Hardware requirements When using cloud technology, you are borrowing other peoples hardware. Similar to outsourcing, you give up control for comfort. When using the public cloud it is the public cloud provider that is providing and maintaining the hardware, See section 2.2. This means you might have some applications that are not able to run because it needs some special dependency that is not available in that specific cloud.

4 Development of method

The work for developing the guidelines was divided into iterations to incrementally build better understanding and allow step-by-step development of the method, discovering and focusing on a few new things each iteration and by doing research in that area. The work started from scratch and in order to develop and test my method two applications were used. One developed by Data Ductus, with source code. The other is from ManageEngine, IT management division of Zoho Corporation. Other than using these two applications, research about general practices also helped developed the method.

4.1 Service Desk Plus

The first application was *Service Desk Plus* (SDP)[6]. SDP is a service desk application, handling cases and incidents. SDP is a *Tomcat* application with a *Postgres* database as the default database running locally[16, 22]. The SDP application did not have any source code available, only an installation file. It was important to find a 64-bit version, because running 32-bit applications require extra setup. Kubernetes also uses a Linux master node, so when choosing what installation file to use, I chose the 64-bit Linux version. When having a Windows node in the cluster, more requirements and setups are needed. The application was installed in a Docker container, which had a Tomcat base image. Tomcat have many different images, the default was good enough for this case because there was no special requirements on the image size. In order to install the application in a Docker container a script had to be created and then run in the Dockerfile so the container would not shutdown instantly. When a container does not have any process to run, it automatically shutdown. A small dummy script was created and run in the Dockerfile. The installation file was also added in the container through the Dockerfile. When the container was up and running, I was able to install the application manually and then start the application. Once the installation was complete, a Docker image was created. This image was then used in the Dockerfile instead of the previous Tomcat base image, and the dummy script was replaced by the applications run file. In order to access the application, a Kubernetes service was created. This encapsulated the application, which gave the end user the ability to access it over the Internet.

The database was running locally in the container, but it was possible to decouple the database through different configurations in the installation process. When decoupling the database from the application, the right image from the database had to be found. The database running locally was Postgres, so naturally a Postgres database should be used for the external database. The default Postgres image was used, since no special requirements had to be met, such as a space requirement. Some setup was required from the application, for example creating user and a database in the container which then the application configuration used.

4.2 EIdentificationAPI

EIdentificationAPI is an application in Windows, which is used to identify users with BankID[1]. EIdentificationAPI is a Data Ductus developed application so the source code was available. In Kubernetes version 1.9 Windows containers are possible but it is only a beta feature, which creates some extra requirements. Visual studios Docker tool[7] was used to generate the base image needed for this application. In order for BankID to work, certificates must be installed on the container the application is running from. Just like SDP, a dummy script was created and run in the Dockerfile so that it was possible install the scripts manually. After the certificates and licenses had been installed an image could be created. However, certificates and licenses often have a expire date so installing and creating a Docker image for every new update or expired certificates and licenses will be very time consuming and inefficient.

Instead of doing it manually, a script was created for installing the certificates and licenses. The initial work is more time consuming for the script setup to work than just installing the certificates and licenses manually, but in the long run it is much more efficient. In this project there was not enough time to make the script to install the certificates and licenses to run perfectly from the Dockerfile.

5 Moving Legacy Applications to the Cloud

From the observation when moving the two legacy application, SDP and EIdentification, and research, the following guidelines were created. In this project Kubernetes version 1.9.2. was used and this has been considered in the answers.

5.1 Questions

Please see next page for all the questions.

1. **Source code**
   Legacy application can look very different. Depending on whether the legacy source code or the executables for the application is present, the migration can look little different. *What is available?*

   a) Source code

   b) Executables

   c) Configuration files

2. **Operating System**
   Some applications have operating system requirements. *What Operating System could the application run on?*

   a) Windows b) Linux

3. **Installation File**
   Some applications have no source code present, they only have an installation file. *Can it be installed in a container?*

   a) Yes b) No

4. **Base image**
   Is some hosting service used when running the application?

   a) Yes b) No

5. **Special Dependencies**
   In some cases applications need different types of dependencies, not only binaries, but also licenses and certificates. For the applications to run, the licenses and certificates needs to be installed. In some cases located in a specific file path with certain privileges. *Does the application have any special dependencies?*

   a) Licenses and Certificates

   b) Other

6. **Stateful or Stateless**
   In Section 2.4, state in application is described. *Is the application stateful or stateless?*

   a) Stateful b) Stateless

7. **Database**
   Depending on if and/or how the legacy application is using a database, the result can be different. If the application is using a database. *Can the database be decoupled?* If it can be decoupled, *Is it latency sensitive?*

   a) Coupled b) Decoupling possible

8. **Special runtime requirements**
   Some application have runtime requirements. These requirements can be for example that they need to use a computer with a Solid State Drive (SSD). *What runtime requirements does the application have?*

   a) Specific hardware

   b) Specific software affiniation/anti-affiniation

5.2 Course of Action

1. *What code is available?*

   a) Source code

   If you have the source code of the legacy application it is possible to do everything from tweaks to major changes to the application. To do that, knowledge about the application is needed, which in the case of legacy applications might be limited. Doing small changes can be, for example, changing the database from local to a external, or connecting the application via IP or a service name in Kubernetes by changing the connection string. It can also mean redeveloping the entire or parts of the application, see Section 2.5.1 for more info.

   b) Executables

   If only the executables from the legacy application is available, an migration to the cloud is still possible. To do that, find out the requirements for the application and prepare an Docker image containing those. Then use the executable in a Dockerfile with the prepared image and create a container from it. This container can run in any environment[33]. In some cases only a installation file is present, this will lead to having to install the application in the Docker container, see Course of Action 3.

   c) Configuration files

   Depending on type of configuration files and how they are used, the result differs. Using Kubernetes ConfigMap makes it possible to control command line arguments, environment variables and files in a volume. If the application is using configuration files as an argument when starting, the Dockerfile needs to be altered, so that the configurations files are arguments when starting the container.

2. *What Operating System could the application run on?*

   Depending on what OS is used Kubernetes' setup varies. Kubernetes is standard an Linux-based cluster and has a Linux-master node.

a) Windows

As of now, Windows Server Containers on Kubernetes is a Beta feature in Kubernetes v1.9[8]. Being only a Beta feature leads to specific version requirements on both Docker and Windows. This means though that you can deploy Windows application into a Linux-based cluster. When your Kubernetes cluster has both Linux and Windows nodes, it is important to specify the Windows Pod. You must explicitly set the Kubernetes nodeSelector constraint to be able to schedule pods to Windows nodes. When using Windows applications it is important to know that it is still in beta and there are many limitations still, for example Windows container OS must match the Host OS. If it does not, the pod will get stuck in a crash loop[8].

b) Linux

Linux nodes can run any Kubernetes supported Docker version.

3. *Can the application be installed in a container?*

a) Yes

Because containers are ephemeral they cease to exist when they do not longer run anything. Doing an installation on a Docker container can cause problems. It is a two step operation.

The installation file must support console installation.

Step 1 First you need to create a small dummy script that will run in the container and copy the installation file to the container. If you are trying to install the program in the Docker container and the Docker container does not have anything to run it will shut down instantly. When the container is up and running the dummy script, you can use '`kubectl exec`' to obtain the containers shell. Through the shell you can install the application. After the installation is done, step one is complete.

Step 2 Create an image from the running container where you just installed the application. Use that new image in another Dockerfile and start the application.

It is also important to think about what type of application it is, 32- or 64-bit. If not, errors can occur.

b) No

If the application can not be installed, try to use run files instead.

4. *Is some hosting service used when running the application?*

When setting up an application in Docker it is important to think about what is needed. It is is a good practice to create the smallest possible image, because it makes the containers' startup time decrease, which in turn leads to using less resources and less money spent. As seen in Section 4.2, Visual Studio can provide a tool for Docker with ASP.NET Core that can be used to generate a image automatically[7].

a) Yes

When finding the right image, it is also important to see what is required and what it not. For example when using the 'microsoft/iis', no entrypoint in the Dockerfile is needed, since the base image with the same name 'microsoft/iis' already includes an entrypoint application, wc3scv, an application that monitors the status of the IIS. If you are using Tomcat there are several different images that can be used. The images can be small, containing only the dependencies to run Tomcat, or big. For example, if you are using an environment where only the tomcat image will be deployed and you have space constraints, 'tomcat:slim' should be your choice.

b) No

If the application does not need a hosting service, running it from the Dockerfile like usual is enough.

5. *Does the application have any special dependencies?*

a) Licenses and Certificates

To solve this problem there are two options, either a manual installation of the certificate in a container and then create a Docker image from it, or modify the Dockerfile to run a script that installs and places the certificate in the right place. *Option 2 is recommended.*

Option 1 See Course of Action 3, but instead of the application install the certificate and/or licenses. The problem with this is that certificates and licenses do not last forever, using the Docker image might cause problems in the future when the certificates and licenses run out. To get the application to work again, a new Docker image must be created and someone has to manually install and create it. In cases where the legacy applications have a lot of certificates and licenses renewing them manually can take a long time and doing a new Docker image is quite unpractical.

Option 2 Using the Dockerfile to run scripts that install the special dependences in the container makes the update for renewing the certificates and licenses almost effortless. By just updating the script in the Dockerfile, it installs another updated version of the certificate or license. This will reduce the human factor and minimize the risk of error, and increase the speed when updating the certificates and licenses.

b) Other

Update the Dockerfile to install programs needed, see Course of Action 3. Or update the Dockerfile to run scripts needed for the application to start correctly. Use different configuration files, preferably used with Kubernetes ConfigMap.

6. *Is the application stateful or stateless?*

a) Stateful application

In Kubernetes StatefulSets are used for scaling and management of a set of Pods instead of the regular Deployment when dealing with stateful application. StatefulSets are intended to be used with stateful applications and distributed systems. See more in 7. Database.

b) Stateless application

Kubernetes is excellent for handling stateless applications. The cluster is only responsible for is the application's code, and other static content, hosted on it.

7. *Can the database be decoupled?* If it can be decoupled, *Is it latency sensitive?*

   a) Coupled

   If you know that the legacy application is using a database and only has an executable, it is important to find out what type of database it is. When the database type is known it is possible to find the correct Docker image and to know what other dependencies the database has. After that the application should be able to use the database as intended. See top left in Figure 9.

   b) Decoupling possible

   When the database is decoupled, it is necessary to find what database the application is using and then find the appropriate Docker image and start the database in a container with a volume connected to it. In some cases some setup needs to be done in the database. Depending on how the application handles the database, adding user and setting the privileges might be necessary to set up before the application can run properly. Depending on how the application connects to the database, the set up might look different. If a connection string is used, this has to be altered to match the new database; configMaps in Kubernetes can be used for setting up environment variables. The database also needs to have a service linked to it. The service will enable the connection string or the environment variables to be very abstract since it allows the application to access the services in the cluster by only specifying the name of the service, see Section 2.4.

   Is it latency sensitive?

   Kubernetes' documentation states that it is preferred to use only one container per pod, which makes the bottom example in Figure 9 the solution to strive for. Some applications have some requirements so that the two containers exists in the same pod, for example if there is a special requirement that has specific hardware for that application, or if the application is latency sensitive. Then top right in Figure 9 is the best option.

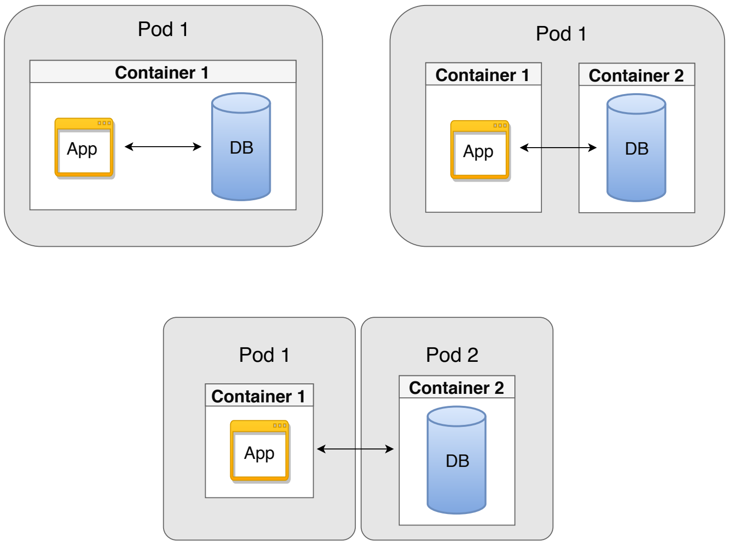

Figure 9: Three different ways of containerize an application with a database.

8. *Special runtime dependencies*

   a) Specific hardware

      If your application is required to run on specific hardware, specifying this in the Pod is necessary, using nodeSelector. This is, for example, your Pod only should run in a Node that has a SSD.

   b) Specific software affiniation/anti-affiniation

      If you want to your application to run in the same Node as another application, node affinity can be used. Also if you do not want the application to co-exist with another application in a Node anti-affinity can be used. When Kubernetes schedules containers, it looks on the current free resources and use them effectively, for example, if the cluster is located in both North and South America and the applications running on a server in North America but the database container is scheduled to be ran in South America. In some application this latency can not be tolerated. So when the application and database have to be located on the same node or pod to reduce latency, affinity in Kubernetes can be used. If instead you want the opposite and have applications that do not want to coexist on the same node or pod, anti-affinity can be used.

6 Future work and Discussion

The guidelines of the project were meant to be as general as possible, therefore the two applications (SDP and EIdentificationAPI) investigated in this project were not chosen randomly. Instead they were selected so that they would have some fundamental differences, for example one being a Windows application and the other being a Linux one. Another example that differed between the two applications were the differences with source code. In SDP the source code was not available and it only had an installation file but for EIdentificationAPI that was not the case. In EIdentificationAPI the source code was available which makes it possible to tweak or do major changes to the application as mentioned in Section 2.5.1. The importance of having these fundamental differences are because you can create a more general guide if you consider all types of applications. If focus were on a specific type of legacy application a more precise guide could have been developed for that type of legacy application with the price of being less general. However, as the goal of this project was to create a general guide, choosing different types of applications were of great importance.

Data Ductus provided a pool of legacy application to use for this project. Having picked two different applications for the guide would have probably made the guidelines look different and other aspects might have been highlighted. Being able to do more research on more applications would have provided a more general guide and probably a more specific guide for legacy applications in particular. Having only a small sample set of legacy application probably leads to the guidelines being less general than it would if more applications would have been used for the research. However, as this project only has spanned over a few months there was not enough time for conducting research on additional legacy applications, which would have been more ideal.

Data Ductus is a relative small size company and their biggest customer is Svenska Kyrkan. Being so, the applications that was provided by Data Ductus have a tendency to be in a particular area of business, so the pool of legacy applications might have looked different if another company was the provider. For example if a company within manufacture or finance had provided the pool of legacy applications the result would have looked different. So future work would have been to go to another business sector and consider those types of applications as well.

7 Conclusion

The purpose of this report was to develop a guide for moving legacy application to a cloud environment, taking into account what are the possible benefits and what parameters from the application are highly correlated with these benefits. Research was conducted to find out what key aspects to consider, and what problems and risks to be aware of, from a general stand point.

All legacy applications share some properties as mentioned in Section 1, but they can differ a great deal, just like any other application. Therefore, when investigating how to move legacy application to a cloud environment and create guidelines from it, it is nearly impossible to cover all different approaches for the different applications. Nonetheless, from these guidelines, knowledge about how to start executing a move and what pitfalls to avoid when moving legacy applications to a cloud environment are presented.

The general advantages of what cloud technology brings are described in Section 3.1. Regardless of the application being legacy or not, moving an application to the cloud will entail these advantages, some more beneficial than others. In the case with legacy application the increased manageability is the most noticeable: being able to deploy instantly and not worrying about setting up the hardware, installing all the dependencies and running the application. This can instead be done with a push of a button or by executing a script.

The parameters that are correlated with the benefits that cloud brings are seen in more or less all of the questions presented in Section 5.1. The parameters that are related with manageability are the most noticeable and highly correlated, for example if the legacy application have a lot of dependencies or installation files the manageability will increase a lot with a move to a cloud environment.

8 References

[1] "BankID,"
https://www.bankid.com/om-bankid/detta-ar-bankid, [Accessed: 2018-04-23].

[2] "Data Ductus,"
https://www.dataductus.com/about-us/, [Accessed: 2017-12-13].

[3] "Docker Security,"
https://docs.docker.com/engine/security/security/, [Accessed: 2018-03-12].

[4] "It-skandalen hos Transportstyrelsen snart i KU,"
https://www.idg.se/2.1085/1.686237/transportstyrelsen-data-sakerhet, [Accessed: 2018-05-04].

[5] "Kubernetes - Updating a Deployment,"
https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#updating-a-deployment, [Accessed: 2018-05-24].

[6] "ServiceDesk Plus,"
https://www.manageengine.com/products/service-desk/?index, [Accessed: 2018-02-12].

[7] "Visual Studio Tools for Docker with ASP.NET Core,"
https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/visual-studio-tools-for-docker?view=aspnetcore-2.0, [Accessed: 2018-05-22].

[8] "Windows in Kubernetes,"
https://kubernetes.io/docs/getting-started-guides/windows/, [Accessed: 2018-05-07].

[9] A. Amber, "8 container orchestration tools to know," April 2016, [Online; posted 12-April-2016]. [Online]. Available: https://www.linux.com/NEWS/8-OPEN-SOURCE-CONTAINER-ORCHESTRATION-TOOLS-KNOW

[10] Antonopoulos, Nick and Gillam, Lee and SpringerLink (Online service), *Cloud Computing: Principles, Systems and Applications*, 2nd ed. Cham: Springer International Publishing, 2017.

[11] C. Baun, *Cloud computing: web-based dynamic IT services.* New York;Heidelberg, [Germany];: Springer, 2011.

[12] BCS The Chartered Institute for IT and BCS, The Chartered Institute for IT and ebrary,Inc and Books24x7,Inc, *Cloud computing: moving IT out of the office.* Swindon, GBR: British Informatics Society Ltd, 2012.

[13] D. Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes,” *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[14] S. Bhardwaj, L. Jain, and S. Jain, “Cloud computing: A study of infrastructure as a service (iaas),” *International Journal of engineering and information Technology*, vol. 2, no. 1, pp. 60–63, 2010.

[15] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, “Legacy information systems: Issues and directions,” *IEEE software*, vol. 16, no. 5, pp. 103–111, 1999.

[16] J. Brittain and I. F. Darwin, *Tomcat: The Definitive Guide: The Definitive Guide.* ” O’Reilly Media, Inc.”, 2007.

[17] M. A. et al, “Above the clouds: A berkeley view of cloud computing,” *UC Berkeley Reliable Adaptive Distributed Systems Laboratory*, Febuary 2009, http://home.cse.ust.hk/~weiwa/teaching/Fall15-COMP6611B/reading_list/AboveTheClouds.pdf.

[18] M. F. Gholami, F. Daneshgar, G. Low, and G. Beydoun, “Cloud migration process — A survey, evaluation framework, and open challenges,” *The Journal of Systems and Software*, vol. 120, pp. 31–69, 2016.

[19] B. Jennings and R. Stadler, “Resource management in clouds: Survey and research challenges,” *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, 2015.

[20] S. Kleinschmager and I. ebrary, *Aspect-oriented programming evaluated: a study on the impact that aspect-oriented programming can have on software development productivity*, 1st ed. Hamburg: Anchor Academic Pub, 2012;2013;.

[21] C. Kniep, “Containerization of high performance compute workloads using docker,” *doc. qnib. org*, 2014.

[22] M. J. R. K. Krosing, Hannu, *PostgreSQL Server Programming.* Packt Publishing, 2013. [Online]. Available: https://app.knovel.com/hotlink/toc/id:kpPSQLSP0U/postgresql-server-programming/postgresql-server-programming

[23] P.-J. Maenhaut, H. Moens, V. Ongenae, and F. De Turck, “Migrating legacy software to the cloud: approach and verification by means of two medical software use cases: Migrating Legacy Software to the Cloud: Approach and Verification,” *Software: Practice and Experience*, vol. 46, no. 1, pp. 31–54, 2016.

[24] M. Mao and M. Humphrey, “A Performance Study on the VM Startup Time in the Cloud.” IEEE, 2012, pp. 423–430.

[25] V. Medel, O. Rana, J. a. Bañares, and U. Arronategui, "Modelling performance & resource management in kubernetes," in *Proceedings of the 9th International Conference on Utility and Cloud Computing*, ser. UCC '16. New York, NY, USA: ACM, 2016, pp. 257–262. [Online]. Available: http://doi.acm.org.ezproxy.its.uu.se/10.1145/2996890.3007869

[26] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *Association for Computing Machinery. Communications of the ACM*, vol. 53, no. 6, p. 50, 2010.

[27] M. Mortensen, S. Ghosh, and J. M. Bieman, "Aspect-Oriented Refactoring of Legacy Applications: An Evaluation," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 118–140, 2012.

[28] S. Newman, *Building microservices: designing fine-grained systems.* " O'Reilly Media, Inc.", 2015.

[29] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Computing*, vol. 16, no. 1, pp. 69–73, 2012.

[30] B. P. Rimal and I. Lumb, *The Rise of Cloud Computing in the Era of Emerging Networked Society.* Cham: Springer International Publishing, 2017, pp. 3–25. [Online]. Available: https://doi.org/10.1007/978-3-319-54645-2_1

[31] N. Serrano, G. Gallardo, and J. Hernantes, "Infrastructure as a service and cloud technologies," *IEEE Software*, vol. 32, no. 2, pp. 30–36, 2015.

[32] Vohra, Deepak and SpringerLink(Online service), *Kubernetes Microservices with Docker.* Berkeley, CA: Apress, 2016.

[33] ——, *Pro Docker*, 1st ed. Berkeley, CA: Apress, 2016;2015;.

[34] B. Winter, *Agile performance improvement: The new synergy of agile and human performance technology.* Springer, 2015.

[35] J. P. Zagal, R. S. Ahues, and M. N. Voehl, "Maintenance-oriented design and development: a case study," *IEEE Software*, vol. 19, no. 4, pp. 100–106, 2002.