**Research**

***Docker ~ Deployment for Cloud Applications***

*Energy Grid North*

|  |
| --- |
| **Datum : 1 juni 2021** |
| **Versie : 1** |
| **Status : Start** |
| **Auteur : Maarten Blömer** |

[**Introduction**](#_afahkdpm9yfy) **3**

[**Questions and methods**](#_49xkyac7xkla) **3**

[**Conclusion**](#_3lsge78c3sg0) **5**

[**Recommendations**](#_b8z2afw005de) **5**

[**Appendix/References**](#_3zzg1usuo9p8) **5**

# Introduction

This research document will mainly focus on how to transform your local java project into a docker image which can be run on any cloud platform to host your java project as a cloud application. This document is not ment as a step by step guide but only contains the information needed to do so.

# Questions and methods

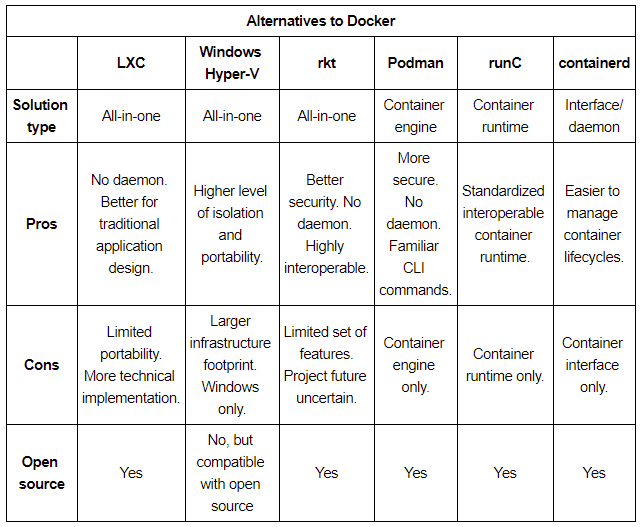
How does one deploy an application to the cloud? If any, what are the prerequisites?

From the library strategy we will be using the product analysis to look at what is available and apply it to the scope of our project. We will also be applying some of the best practices when using these products. Since we apply this knowledge to our own project some of the workshop and lab strategies such as prototyping will also apply, especially considering that the prototype is also tested along the way.

First of all we have to look at what is available on the market to make sure we can deploy our application to the cloud. Given the parameters from the cloud provider that have been allocated to us by the teachers for this semester; we have three options. Google cloud, Amazon, and Azure. All three are a viable option for hosting. After a short discussion regarding what the group members decided to use for their personal project we determined to take a closer look at Azure’s cloud platform. Reasoning behind this is also following the Microsoft development stack. For the hosting of the cloud application we need to make sure that our application can run in the cloud, this is done via the azure kubernetes service.[[1]](#footnote-0) This service also has several options for us to deploy the application via a CI/CD pipeline. Within this service we have to create an instance of our application. This is achieved via a container engine.

Looking into different solutions for the container engine we came across a whole list of possibilities. The easiest option would be to directly jump on the docker hype train before looking at what possibilities there are. Docker is not only the most popular choice but has been for a very long time. It is also one of the products that was already suggested by both Azure and Kubernetes to complete the deployment cycle. Kubernetes also wrote a blog post about them dropping a Docker component out of their code as the development of cloud providers already have container runtime engines embedded into their cloud platform.[[2]](#footnote-1)[[3]](#footnote-2)[[4]](#footnote-3)

The alternatives investigated are listed in the table below.

[[5]](#footnote-4)

Now that we know what we want to use to build our images and what engine our images will be running on, it's time to build the image.

From one of the workshops we got introduced to a plugin for Maven that can easily help build docker images for people with limited docker knowledge. It was also easy to install and implement into the project.[[6]](#footnote-5) Similar to how the alternatives to docker work, Jib provides us with a fast way to deploy applications into multiple layers, splitting dependencies from classes so Docker doesn’t have to rebuild the entire java application and only deploy the layers that changed. This also doesn’t trigger any redundant builds if nothing changed. And lastly it’s daemonless which reduces the CLI dependencies allowing you to build Docker image from within Maven and push to any registry of your choice. This ultimately creates no need for writing complicated Dockerfiles and calling docker build/push.

Now that we have our images it’s time to assemble it in the cloud. Constructing a docker-compose file puts everything into one place so the container engine knows what to run and in what order. It is important to note that the docker-compose file only works locally or within a docker swarm cluster, to get the docker-compose working within Kubernetes we use Kompose to automatically translate it to kubernetes manifests.[[7]](#footnote-6)

After running the kompose commands we have a fully functional application that’s running in a kubernetes cluster.

# Conclusion

There is an evolving trend to move away from Dockerfiles as a whole and use plugins such as Jib which allow for the development of local docker images without the use of a dockerfile. However with this new technology there are some drawbacks of requiring a Dockerfile to maintain your project on docker hub’s registries. This causes some issues where your project will not build correctly on docker hub. There is a workaround for this which we currently use in our group project by manually pushing the built images to docker hub via docker desktop. Obviously we want to make sure this is automated within a CI/CD pipeline where the docker image is built after pushing your code to a Git repository. Lastly, putting it in a docker-compose file for the deployment into the cloud. This file keeps all your images in one place and orchestrates the order in which the applications are started within your cloud provider.

Furthermore using Kompose you can translate your docker-compose file into bite sized chunks for kubernetes to keep track of which components need to be run and in which order they need to start. This can be run in a cloud environment where your kubernetes cluster is hosted allowing your application to fully enjoy the benefits of the cloud.

# Recommendations

Building your application with the help of a docker-compose gets all the components in one place, allowing for a quick and easy debug/boot up process. Using tools like Kompose can translate the docker-compose file into a kubernetes friendly version and run your application in a kubernetes cluster.

# Appendix/References

<https://www.youtube.com/watch?v=7KUdmFyefSA>

<https://kubernetes.io/blog/2020/12/02/dockershim-faq/>

<https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/>

<https://jfrog.com/knowledge-base/the-basics-7-alternatives-to-docker-all-in-one-solutions-and-standalone-container-tools/>

<https://azure.microsoft.com/en-us/services/kubernetes-service/#:~:text=Azure%20Kubernetes%20Service%20(AKS)%20offers,and%20scale%20applications%20with%20confidence>

<https://kubernetes.io/blog/2016/11/kompose-tool-go-from-docker-compose-to-kubernetes/>

1. <https://azure.microsoft.com/en-us/services/kubernetes-service/#:~:text=Azure%20Kubernetes%20Service%20(AKS)%20offers,and%20scale%20applications%20with%20confidence>. Azure Kubernetes Service (AKS) [↑](#footnote-ref-0)
2. <https://www.youtube.com/watch?v=7KUdmFyefSA> “Kubernetest is dropping docker support” TechWorld with Nana (December 10, 2020) [↑](#footnote-ref-1)
3. <https://kubernetes.io/blog/2020/12/02/dockershim-faq/> [↑](#footnote-ref-2)
4. <https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/> [↑](#footnote-ref-3)
5. <https://jfrog.com/knowledge-base/the-basics-7-alternatives-to-docker-all-in-one-solutions-and-standalone-container-tools/> JFrog, Last Updated May 6, 2021 [↑](#footnote-ref-4)
6. <https://github.com/GoogleContainerTools/jib/blob/master/README.md> Jib by Google [↑](#footnote-ref-5)
7. <https://kubernetes.io/blog/2016/11/kompose-tool-go-from-docker-compose-to-kubernetes/> [↑](#footnote-ref-6)