**Request for Information**

Dockerizing Magento 2 and deploying it on Kubernetes

**Overview:**

This document details the advantages in Dockerizing an application built on Magento2 CMS and deploying it on the Kubernetes platform.

**Docker image and Kubernetes: The trivial description**

To detail about the advantages of Dockerizing an application and deploying it into Kubernetes, one must understand the core ideology of both technologies. The details for the same are as follows:

**Docker Image:**

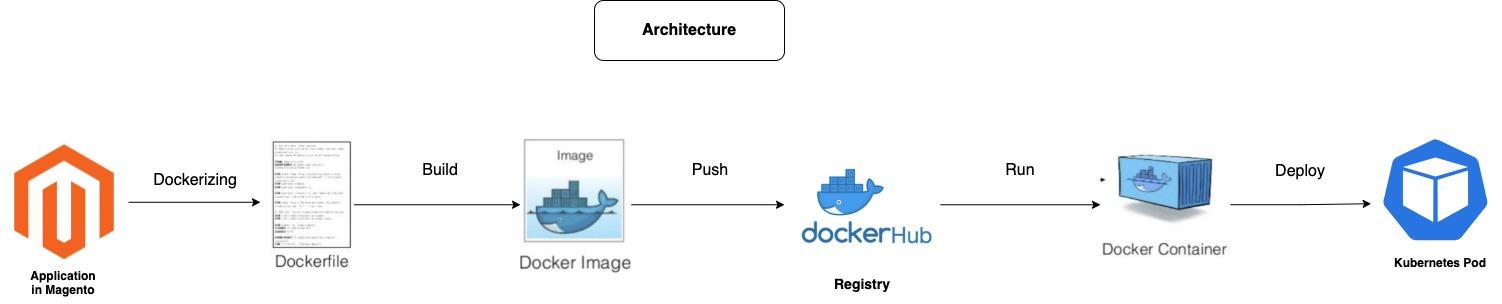
A docker image in its most basic form is a compressed file of the given deployable. In our case, a Magento2 application. Developing applications in a competent CMS environment like Magento2 means that the file size is robust and prone to accommodation issues on the server. To combat this issue and add a multitude of security features to the deployable environment (Kubernetes, as is the popular choice and our said environment), the said application is converted into a docker image. This docker image has a series of access ports, which unlock with a numeric key. Usually, one access port and one database port are opened and linked in these containers. This concept will further be broken down in the following section in the document.

**Kubernetes:**

As is the gold standard of the infotech industry, Kubernetes is the open-source platform that allows the deployment of containerized applications with facilities of scaling on an infinite level, both horizontally and vertically. Kubernetes today directly translates to speed, precision, reliability, automation and availability for all those who seek its services. Thus, all application hosts mostly throng to the lines to integrate a Kubernetes platform to their environment. Popular examples include EKS and Azure Kubernetes in AWS and Azure platforms respectively

**Process Flow:**

To explain the process of Dockerizing a Magento2 application and hosting the said application on Kubernetes, the following flow diagram should be referred to. The verbal debriefing of the same is as follows:



1. The dockerfile is written for the said application and a docker image is built around this dockerfile.
2. The docker image is pushed into DockerHub, which facilitates the developer to pull the pushed image at any given time.
3. This docker image is then pushed into a container. A MySQL database is integrated to the container so that Magento applications are met with their need for one.
4. The database-integrated docker container is finally deployed on Kubernetes with K8 manifests. These K8 manifests are basically YAML script files that direct Kubernetes with basic details like access ports, container kind, restart policy and persistent volume chain.

The technical terms stated in the final step of the process is stated in the section next to this one. It is to note that this process is not dependent on any application hosting service environments but rather, Kubernetes itself. The whole process is platform independent, meaning that this docker container could be deployed literally anywhere with K8 support.

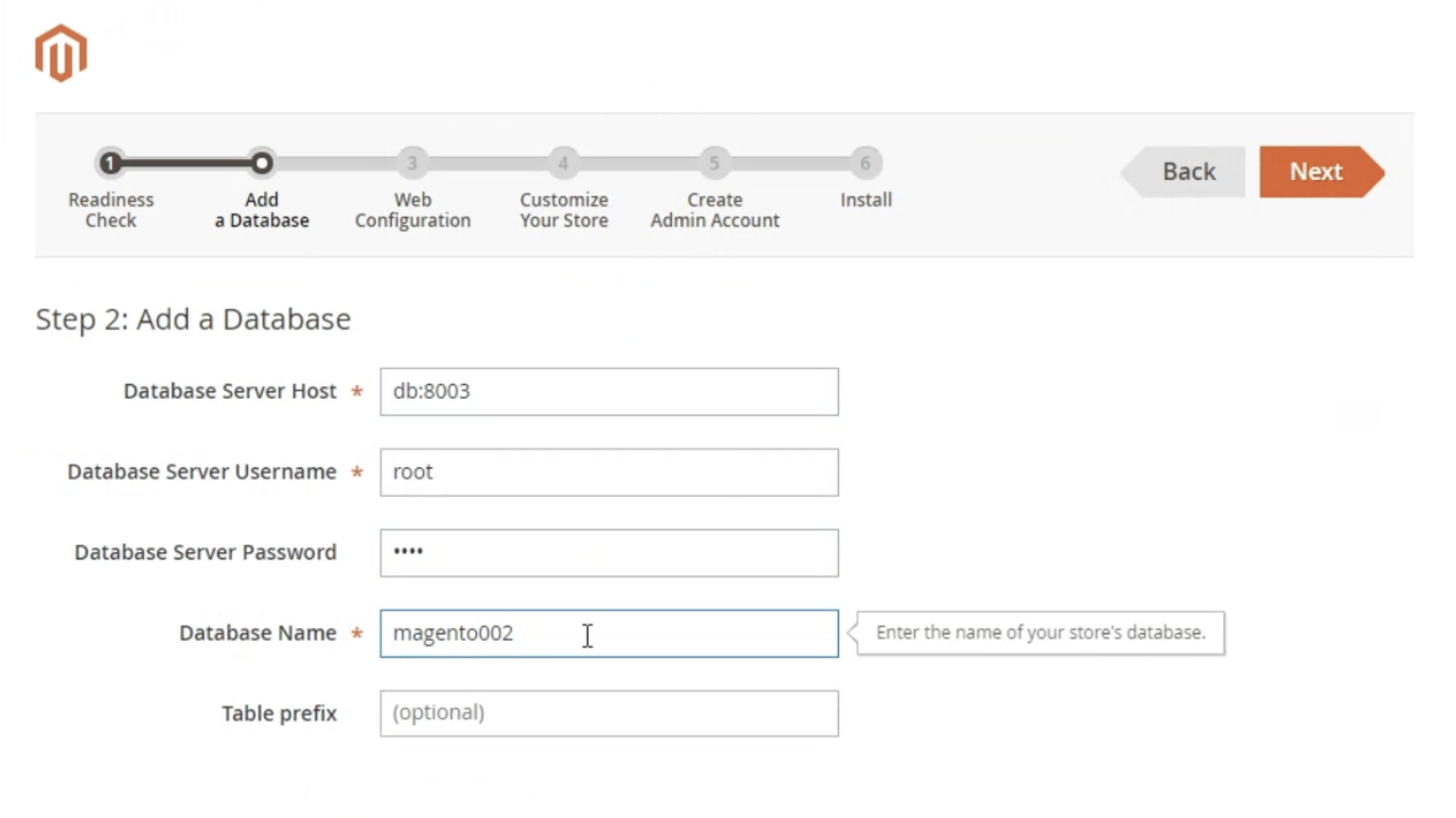
**Feature Specifications:**

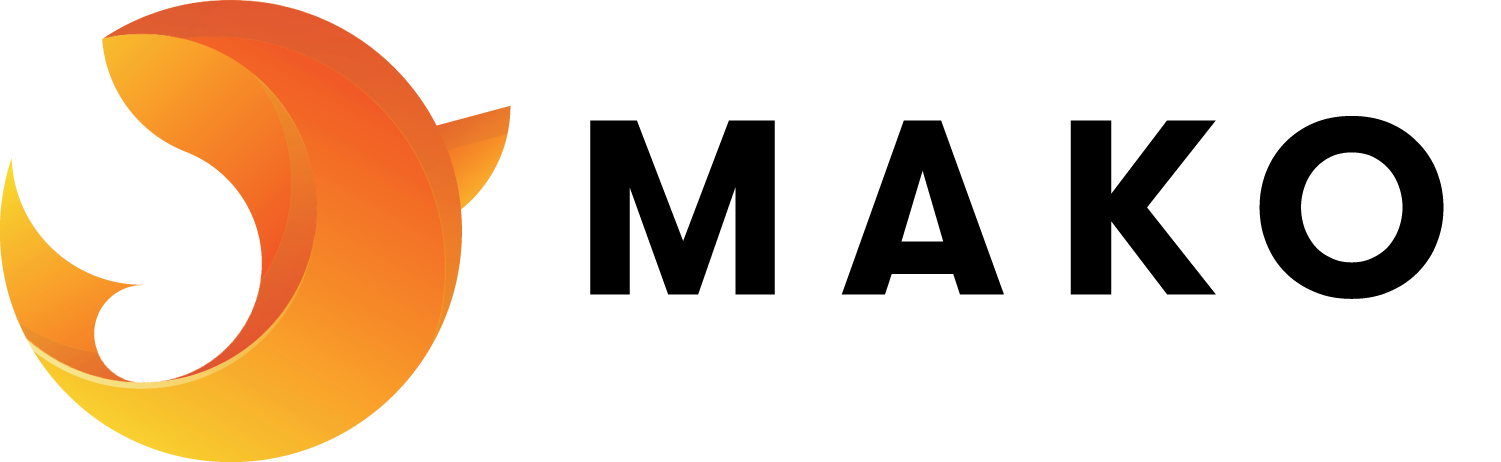
To explain the technicalities of the operation in discussion, here are a few pointers:

1. **Container Kind:** This field of data specifies the nature of the container. If the kind is labelled **“service”**, a developer has access to edit and update the content infinitely. On the other hand if the container kind is marked **“deployable”**, the container acts like the final version of the application and goes up on the K8 platform.
2. **Consistent Port Key:** Consistent Port Key is the only access point for the application within the container. In the YAML code furnished, the consistent port key is 80, which the Kubernetes platform will run operations from.
3. **Restart Policy:** Restart policy is the setting that instructs the Kubernetes platform what to do in the event of a server crash. To restart the server automatically, the setting is toggled to be **“always”**.
4. **Persistent Volume Claim:** This YAML file is the most important one among them all. It basically contains data about the volume of data that the container needs in the Kubernetes platform. Essentially, the YAML file that tells the Kubernetes environment about the needs of the docker container is the persistent volume claim manifest.

**Front-end Outlook:**

The front-end of the process looks a little something like this:





THANK YOU