Modernize JPetStore with Microservices

Estimated time needed: 20 minutes

In this lab, you will take a look at how an existing application can be modernized to use microservices. In addition, you will see how using microservices makes it easy to extend the application to use additional cloud services like artificial intelligence and messaging. This lab will be mostly exploratory, as later labs will provide additional hands-on opportunities.

> Several labs/assignments in this course involve the usage of IBM Cloud account, which did not previously require credit card registration. However, as of October 1st, 2021, IBM Cloud has stopped providing Lite accounts. In the meanwhile, if you don't want to disclose your payment card information to create an IBM Cloud account, you can create an account with a feature code that offers you a 252-day trial account. You can obtain the IBM cloud feature code from the course in the Activate Trial Account section.

Also, please create an IBM cloud account in the US region.

However, if you decide to create an IBM Cloud account with a credit card, it is recommended that you specify US dollars for billing for faster approval. The labs/assignments in the course are designed to be completed with free tiers or Lite plans to avoid costs. Be aware that your credit card may be charged if you exceed the free usage, so it is a good practice to delete your instances once they are not required. And in case of any services that are billed on an hourly basis, please be sure to stop or delete your instance as soon as your lab is complete.

Note: the IBM and Coursera course teams are not responsible for any billed usage you may incur while using IBM Cloud.

Objectives

After completing this lab, you will be able to:

1. Describe what tools are used to modernize a legacy monolithic application
2. Understand how modernization enables the enhancement of legacy apps
3. Have familiarity with IBM Cloud Kubernetes Service and Red Hat OpenShift on IBM Cloud

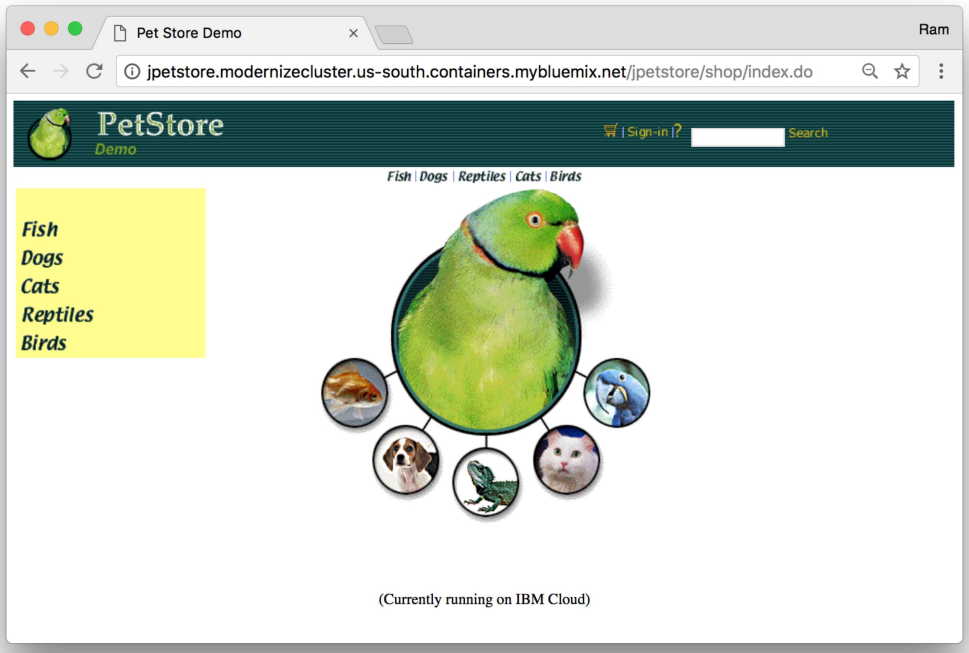
Exercise 1 : Modernize JPetStore with Containers and Microservices

In this exercise, you will see the architecture for a well-known Java monolithic application and see how it can be redesigned using microservices and containers.

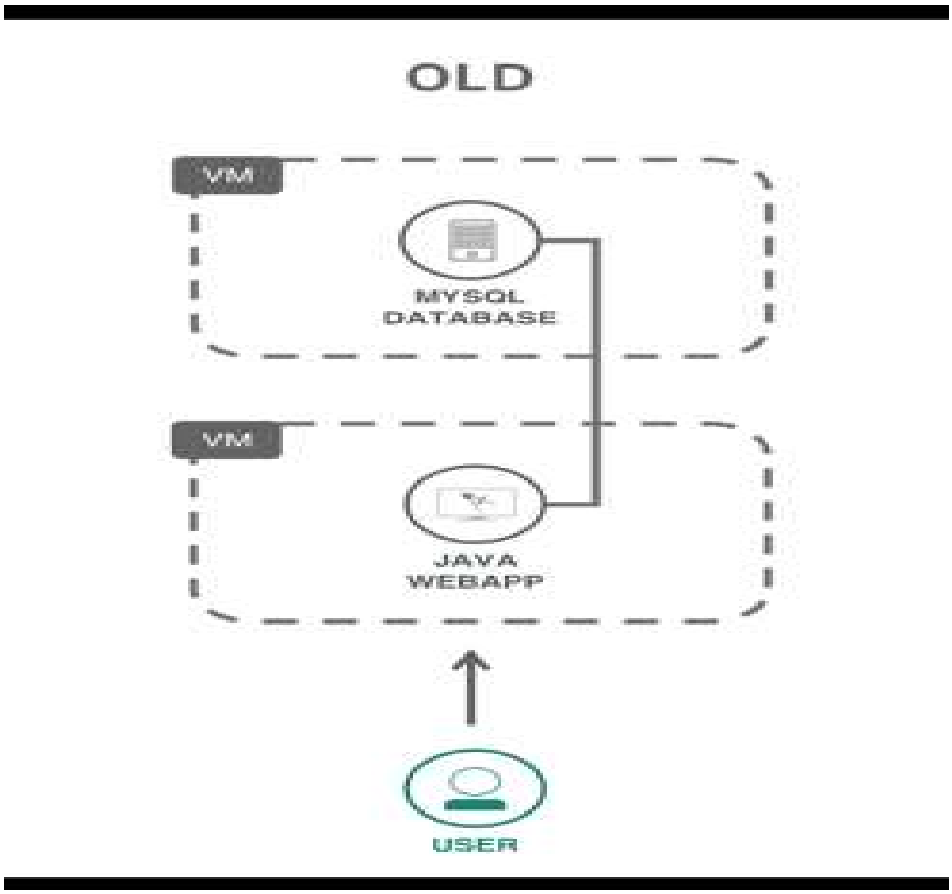
The legacy application we'll consider today is called Java Pet Store. It's a fun one that Java developers might remember from way back in 2004. Java Pet Store, or JPetStore, is a simple e-commerce site. If you don't recognize it, it's fun because it's pretty ugly! But modernizing this application allows us to update and extend it easily in a number of different ways.

1. Here is a look at our running legacy application. This e-commerce pet store has a very simple interface. You can click on the different pet types to view inventory. We will show how to make this interface more

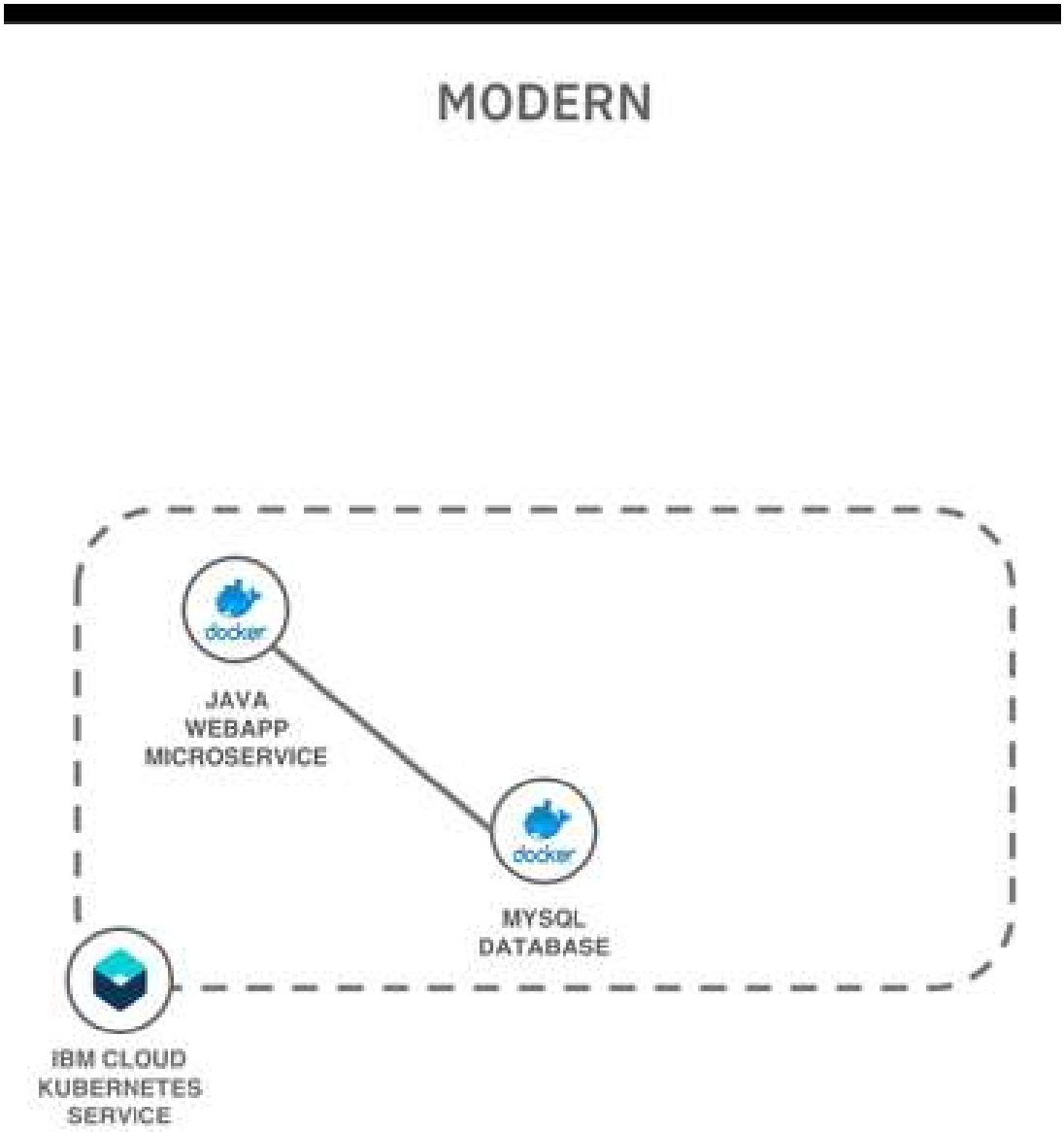
modern in later steps!



1. Below is the architecture for the original JPetStore application. This legacy architecture involves the Java web app running on a virtual machine, as well as a MySQL database running on a separate VM. This was a typical architecture – using multiple virtual machines, one for the database and one for the monolithic application. But let's see how microservices and containers can change this.



1. Here is the new architecture for the JPetStore application. At first glance, this might seem very similar, but there are some key differences that will later have a big impact.



1. The first thing to note is that the Java web app is now a microservice running as a container. The Docker logo indicates that this microservice is containerized. Even though we have not yet made any changes to the code itself – we have simply containerized the legacy app – this is the first step in modernization. Running this web app as a microservice and as a container makes refactoring easier if we want to decompose this application into additional microservices. It also lets us easily integrate this microservice with other backing services.

2. This leads to the second point: the MySQL database is also running as a container. Rather than using a virtual machine to run a dedicated MySQL instance, an instance of MySQL can easily be spun up as a container in our environment. This modern approach makes it easy for the web app to communicate with the database, and it also lets us run our microservices in one environment, which leads us to the next step.

3. Both the web app and the database are running as containers on IBM Cloud Kubernetes Service. By containerizing the web app and the database, both can be run on an IBM Cloud Kubernetes Service or Red Hat OpenShift on IBM Cloud cluster. This provides a lot of great integrations with other IBM Cloud tools, as well as a myriad of capabilities provided natively by these two services.

4. Visit the IBM Cloud Kubernetes Service page as well as the IBM Cloud catalog. Read about the features and benefits provided by the service, as well as seeing how simple it is to deploy a cluster. Do not deploy a cluster since this will cost money. You will be provided with a sandbox environment in later labs.

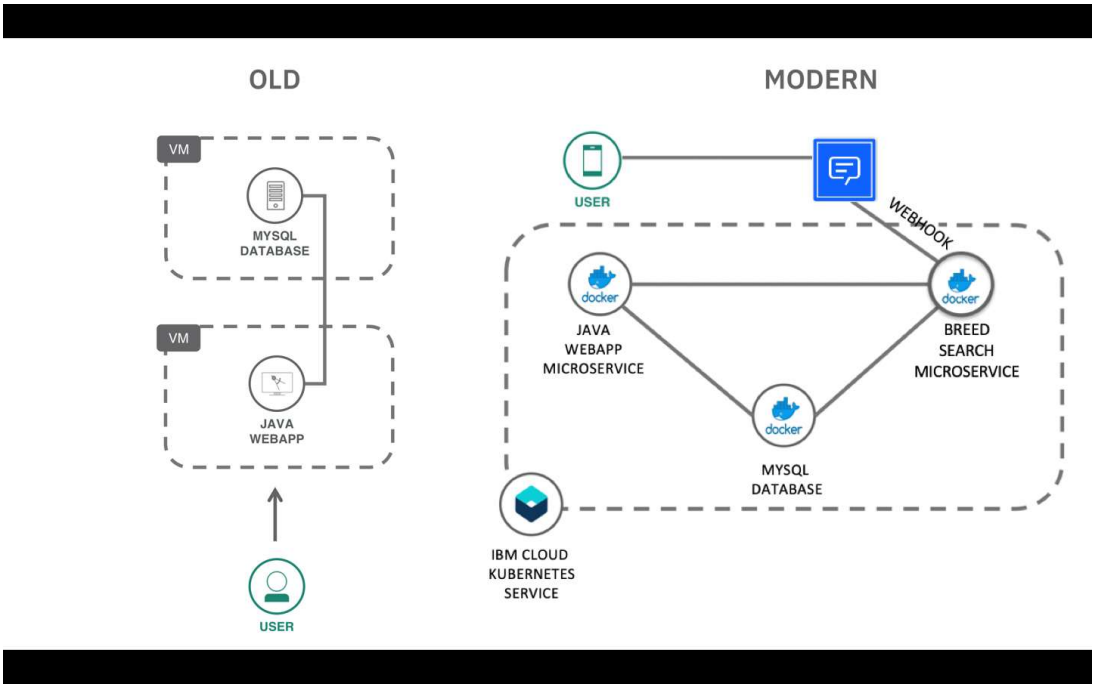
5. Visit the Red Hat OpenShift on IBM Cloud page as well as the IBM Cloud catalog. Read about the features and benefits provided by the service, as well as seeing how simple it is to deploy a cluster. Do not deploy a cluster since this will cost money. You will be provided with a sandbox environment later in the lab.

Exercise 2 : Extend the Modernized JPetStore with Artificial Intelligence

In this exercise, you will see how our modernized JPetStore application can be easily extended with modern capabilities thanks to microservices and containers.

While the first step of modernizing – containerization – didn't dramatically change the architecture of JPetStore, it opened the door to a lot of additional capabilities. The JPetStore interface you saw above was simple. You could search the inventory based on types of pets. But this could be modernized using artificial intelligence – namely, chat bot to create a more robust application.

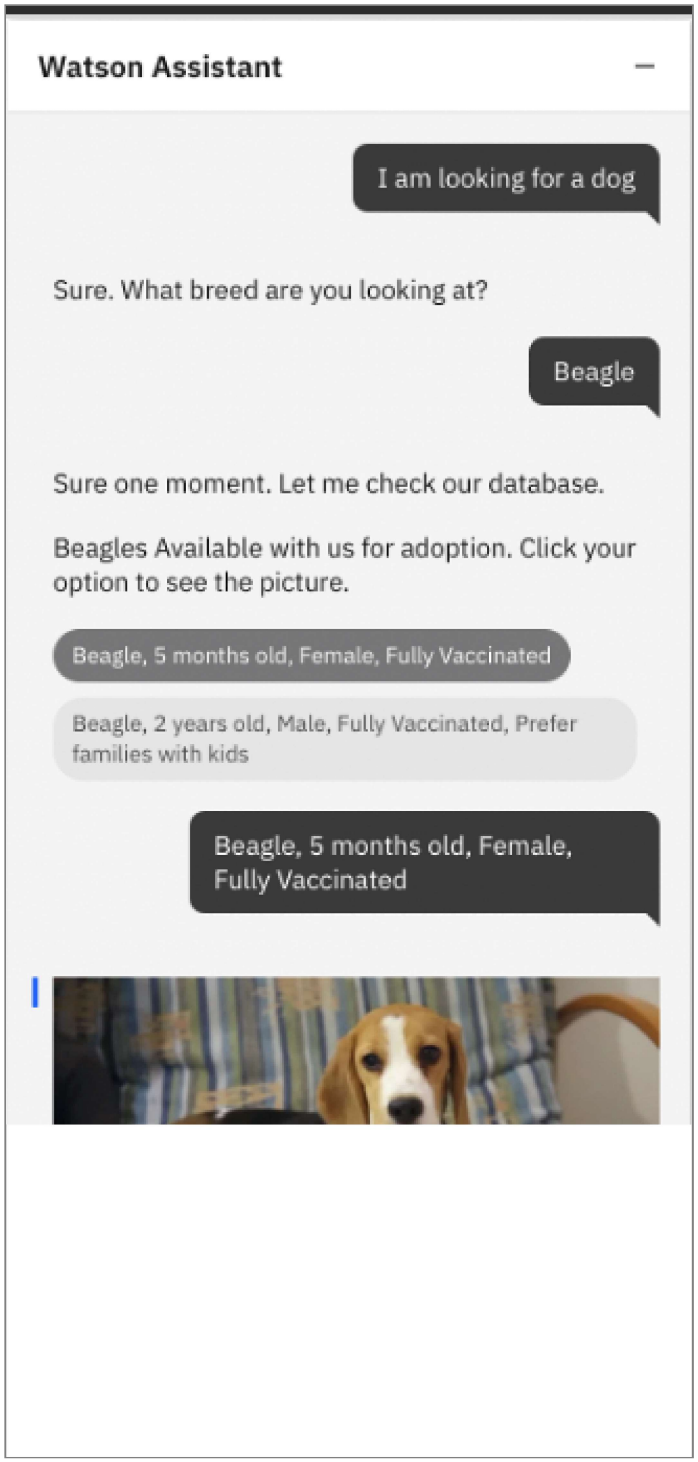
1. Check out this architecture, which extends the basic JPetStore with additional capabilities. We'll walk through each change.



1. The interface of JPetStore is dated. In this iteration, the user can interact with the pet store through the chatbot from their phone or computer. So instead of the main entry point for the service being the web app itself, this modern architecture has the user interacting by using the chatbot.

2. In this case, on the virtual chatbot, you can ask for the animal that you are interested in. The chatbot then uses web hook to the search microservice, which you can see is present as a container running on IBM Cloud Kubernetes Service. If we hadn't modernized with microservices and containers in the first step, we would have had to add to the monolithic JPetStore application to accomplish these enhancements, which would have involved understanding that codebase so that it could be extended without breaking any existing functionality.

3. This chatbot will inturn call the search microservice which will look through the MySQL database and list all the pets that match the search criteria entered by the client. If so, information on the pet's availability and price is provided. If not, an information message is returned.



Conclusion

In this example, you saw how modernizing a legacy application involves containerizing and using microservices. You also observed how containers allowed us to run all of our applications within a single environment with Kubernetes or OpenShift. And by using microservices, it was very easy to extend the pet store to include more modern capabilities like artificial intelligence and chatbot-like interaction.

Author(s)

Alex Parker
Lavanya

Change Log

Date	Version	Changed by	Change Description
2023-01-18	1.1	K Sundararajan	Small UI/rendering updates