Ubiquity: An Extensible Framework for Persistence in Container Environments

Mohamed Mohamed, Robert Engel, Amit Warke, Heiko Ludwig

Almaden Research Center, IBM Research, San Jose, CA, USA
mmohamed,engelrob,aswarke,hludwig@us.ibm.com

Abstract. Within the last few years, containers are being used for a broad set of applications, many of which have extensive requirements in terms of persisting their state. These stateful applications are still not well supported in container-based environments due to the challenges of adding persistence support. There are many efforts being done recently to tackle these challenges but most of them are focused on one environment or one storage provider. In this paper, we present the Ubiquity framework, which provides an extensible way of provisioning persistent storage for stateful containers. Ubiquity can be used to provide different types of persistent volumes from heterogeneous providers to be consumed by heterogeneous container frameworks in a seamless manner.

Keywords: Persistence, CloudFoundry, Kubernetes, Docker

1 Introduction

In the last few years micro-services became the new trend for designing software applications. In this paradigm, software applications are developed as a set of independent components that focus on small functionalities, can be deployed separately, and use some lightweight communication mechanism such as REST, gRPC, etc. These components could be easily deployed in containers that present a lightweight operating system level virtualization mechanism where the application running inside the container shares the kernel with the host operating system but has its own root file system [5]. Many platforms and orchestration systems are based on container concepts to offer an agile way of building micro-service based software such as Cloudfoundry, Docker, Kubernetes, Mesos, etc. While using these container orchestrators (COs) is beneficial for large scale deployments, there is still a lively discussion as to which type of applications they might be best suited for. Most of these COs favor stateless micro-services due to the challenges of managing state in concurrency situations.

Onboarding stateful applications into these COs in a scalable way is not well supported particularly in scenarios where state is accessed from workloads in different deployment platforms. In these scenarios, even though we can have the same persistence backend used to maintain the state, the frameworks that are being used have heterogeneous ways of consuming persistence storage. For example, CloudFoundry uses persistent volumes through its persistence drivers

and service brokers, whereas Kubernetes has a dynamic provisioner responsible for the creation and deletion of volumes and a volume plugin responsible for the other consumption functions (e.g., attach/detach, mount/unmount). The other challenge is that the heterogeneous management functionalities for persistent storage may change from one provider to another and from one persistent volume type to another (filesystem or block device based).

In this demonstration paper, we present the Ubiquity framework that enables seamless access to storage for heterogeneous COs. Ubiquity integrates with the widely used orchestrators (CloudFoundry, Kubernetes, Docker, Mesos, Openshift). It also provides an easy way of adding new storage backends without the need to understand the specificities of the COs that are supported. We will demonstrate how Ubiquity can be used in different scenarios to provide different types of persistent storage in heterogeneous environments. In Section 2, we will present Ubiquity and its different components. Afterwards, we will give different use cases and demonstrate how ubiquity can help managing the persistent volumes for the used COs. Then, we will give a brief literature review in Section **??** and conclude the paper in Section 4.

2 Overview of Ubiquity Framework

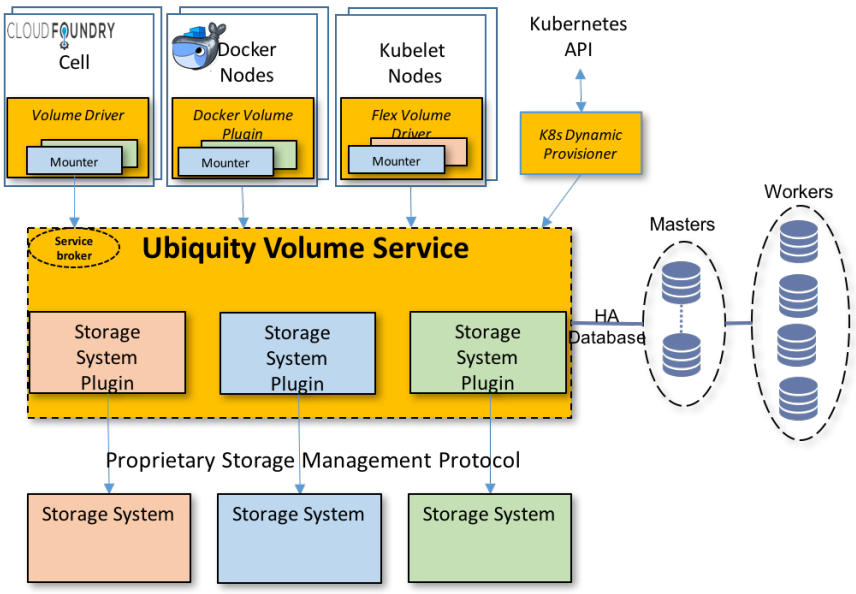

Fig. 1. Ubiquity Architecture Overview

Ubiquity framework allows COs operators to offer persistent storage from various providers without the need to understand the intrinsics of storage backends.

At the same time, it allows storage providers to offer their persistent storage to containers without understanding the intrinsic requirements of the different COs. As shown in Figure 1, Ubiquity is made up of different loosely coupled components that are easy to extend or replace. We briefly introduce these components in the following subsections.

Ubiquity Volume Service: This service is the main component of Ubiquity playing the role of the mediator between the COs and the storage backends. It is offering southbound interfaces to be consumed by COs to allow them to create persistent volumes and manage them. The management operations are continuously evolving based on the functionalities supported by the COs. These operations include attaching/detaching volumes to nodes, setting quota/size of volumes, maintaining the coherence of the attachment of volumes.

Container Orchestrators Plugins: Ubiquity framework has support for different COs using their specific storage plugins. The plugins generally live in all the nodes managed by the CO and they allow to execute host side operations to make the persistent storage ready to be consumed by the containers. So far, we have support for persistence for CloudFoundry (through an implementation of the Open Service Broker API [2]), Docker (through an implementation of the docker plugin API [4]), and Kubernetes environments (through a dynamic provisioner [1] and a volume plugin implementing the Flex Volume API [7]).

Storage Backends: These are the mechanisms needed from the storage provider perspective to make their storage ready to COs.These components implement the needed mapping between Ubiquity API and the specific provider API to allow the creation and management of persistence storage.

Database Service: This service is used for our locking mechanism to manage concurrent access to volumes. It is also used to enable high availability based on leader election algorithms.

3 Demonstration

In this section, we will describe the two demos showing how to use Ubiquity in different scenarios.

Shared volumes: In this demo, we will show how Ubiquity allows providing persistent volumes across different container environments. We have a multi-tier application where the visual part of the application runs in CloudFoundry and the processing part runs in docker. The application on CloudFoundry shows a catalog of pictures saved in a persistent volume bound to the application. We start a docker deployment using the same persistent volume to run some face recognition algorithms using OpenCV. We show how the content is changed in the CloudFoundry side as well. In the demo, we will show the detailed steps towards enabling both environments to share the same persistent volume. The persistent storage is created out of distributed filesystem.

Dedicated volumes: In this demo, we show how we create a dynamic Cassandra cluster on kubernetes. Since Cassandra uses its own filesystem, we need to use persistent volumes based on block devices. One challenge here is to make each instance of the cluster get its own block device to avoid data corruption. Whenever a new instance is added (scale up), a new persistent volume is created and mounted into the new pod. Ubiquity allows to do that by creating a storage class that refers to our dynamic provisioner. We refer to the storage class in the persistent volume claim template related to the descriptor of the Cassandra container. Scaling the cluster up or down becomes a straight forward action backed up with Ubiquity. We can easily use Cassandra from any instance and check that the data is replicated and well maintained. If ever we loose a pod, kubernetes will recreate it and Ubiquity will ensure that the pod is bound to the right persistent volume to maintain the consistency of the cluster. In this demo, we can simulate a failure by killing one or more cluster nodes. Kubernetes will recreate the nodes and Ubiquity will make sure that the data will be recovered.

4 Conclusions

Using containers to deploy stateful applications is a challenging task. Given that each container orchestrator (CO) has a different set of APIs and storage features, the burden becomes on the storage vendor to integrate into each CO. So far there are little efforts being done to integrate this diverse set of COs and storage systems. Efforts such Rexray [3], Trident [8] and Torus [6] are either specific to one framework or one storage provider. So far, none of them offers support for Cloudfoundry, nor heterogeneous support for different container environments and different storage providers at the same time. In this demonstration paper, we proposed the Ubiquity framework that addresses the complexity of bringing together the different COs and Storage providers in the context of providing persistent storage across COs.

References

1. Dynamic provisioning and storage classes in kubernetes. http://blog.kubernetes.io/2016/10/dynamic-provisioning-and-storage-in-kubernetes.html
2. Open service broker api. https://github.com/openservicebrokerapi/
3. Rex-ray openly serious about storage. https://rexray.readthedocs.io/en/stable
4. Volume plugins. https://docs.docker.com/engine/extend/plugins_volume
5. Azab, A.: Enabling docker containers for high-performance and many-task computing. In: IC2E (2017)
6. Michener, B.: Presenting torus: A modern distributed storage system by coreos. https://coreos.com/blog/torus-distributed-storage-by-coreos.html
7. Nelluri, C.: Flexvolume explored. https://www.diamanti.com/blog/flexvolume-explored/
8. Sullivan, A.: Introducing trident: A dynamic persistent volume provisioner for kubernetes. http://netapp.io/2016/12/23/introducing-trident-dynamic-persistent-volume-provisioner-kubernetes