**Title:**

Managing and Rotating Secrets with Azure KeyVault, Managed Services, and some automation – Part 1

**Teaser:**

Secret rotation is not a new problem. In cloud-based environments many services implement secret-based authentication schemes. For many organizations, these secrets must be rotated on a regular schedule. In addition to the actual problem of rotating the access keys, there exists a problem of how these newly rotated credentials are propagated to all the applications and systems that utilize them. In this multi-part blog series, we will discuss a solution that addresses both the scheduled rotation and dependency notification/update requirements through automation.

**Body:**

Many organizations have long standing security mandates to rotate application secrets. These secrets can range from specific identify passwords to service access keys. As more enterprises move their workloads to the cloud, the need for adherence to these policies rises even more. More and more workloads take advantage of a multitude of PaaS services, many of which utilize different access control schemes. As the number of such services and the custom applications that rely on them grows, IT administrators are often left to their own devices to figure out how to manage period rotation of the multitude of secrets as well as how to update their dependent applications when such change is made.

Azure offers some automation to help solve a portion of these problems, specifically [automated storage account rotation by Key Vault](https://docs.microsoft.com/en-us/azure/key-vault/secrets/overview-storage-keys) and general guidance on how to use [automation to solve these types of problems](https://docs.microsoft.com/en-us/azure/key-vault/secrets/tutorial-rotation-dual) for other services. Unfortunately, this is often not enough to ease the tasks associated with managing this problem space. Not only are these solutions insufficient in addressing a multitude of services organizations use in the cloud, they also do not address the “update dependent applications” part of the problem at all.

Having worked with multiple Azure customers who have faced these challenges, I decided to address the whitespace by building a solution to tackle this specific area. I am not super creative when it comes to naming things, so after a short deliberation period I went with my go to approach for naming projects – Greek / Roman Mythologies. And that is how project “[Harpocrates](https://en.wikipedia.org/wiki/Harpocrates)” was born. I have many ideas on how this solution could be distributed in the future, but for now, its source code is available on [GitHub](https://github.com/gkli/harpocrates). At the time of this writing, the codebase is at best in an alpha stage. The solution works and can rotate secrets for several different services. The codebase, however, can stand some refactoring and additional features, which I plan on working on over time.

At the root of the problem Harpocrates addresses is a relatively simple business process depicted below.



Figure Secret Rotation Business Process

Incorporating this business process with the [guidance given by Azure](https://docs.microsoft.com/en-us/azure/key-vault/secrets/tutorial-rotation-dual) one can utilize the following high level flow



Figure Harpocrates Logical Flow

Using the two diagrams depicted as the basic premise for Harpocartes we have a codebase that can monitor for events raised out of Key Vault. At the time of writing, Harpocrates can act on 3 specific events: Secret Expired, Secret Expiring, and “Secret Created”. In future, there’s a desire to add management KV Keys & Certificate objects as well. Harpocrates is specifically built to automate the following logical flow.



Figure Harpocrates Business Process

To allow for extensibility, specifically around solution’s ability to manage a multitude of downstream services, including those hosted in other clouds as well as custom applications, Harpocrates takes a factory-based approach to interactions with these downstream services. Thera are concrete implementations for managing Azure Storage Accounts, Azure CosmosDB, Azure Application Registration (Service Principals), etc. These constructs can be easily extended to add “Secret Management Providers” for AWS S3 buckets or your custom application. The only assumption Harpocrates makes is that the downstream services management endpoints are available for its consumption. This means the management endpoint needs to be addressable on the network and Harpocrates will have permissions to make changes to it. For the purpose of this solution, the term “management endpoint” is being used loosely to represent an entity the application can interact with to manage the service secrets. This can be a restful endpoint, an API, or simply a file path where config files are stored, thus allowing for a high degree of extensibility.

Harpocrates is also being designed to support deployment into various compute platforms such as AKS, App Service, VM, etc as well as support a number of different storage technologies as to not place a significant burden on deploying the solution into your own environment. Depicted below are high level deployment diagrams of Harpocrates in AKS and App Service.



Figure Harpocrates Azure Kubernetes Service Deployment Diagram



Figure Harpocrates Azure AppService Deployment Diagram

If you’re still reading this and are interested in learning more about Harpocrates and how its journey to addressing this problem space, please take a look at other articles in this blog series and as always, feel free to browse the codebase and provide feedback.