### **From Chaos to Order in Secrets Management: Secrets Management with Dapr.io**

Ciao everyone,

As we develop applications that evolve with the changing demands and needs of users, we increasingly opt for more modern architectures. When crafting modern applications, we often find ourselves in need of various confidential pieces of information. These confidential details may encompass API keys, database passwords, authentication documents, and other sensitive data. However, the insecure management and utilization of these confidential details can create a chaotic environment. To transition from this chaos to an organized state, dapr.io assists us, developers, with Secret Management.

### **dapr.io**

Dapr (Distributed Application Runtime) is an open-source application that puts an end to the chaos in managing microservices development and operations by providing a plethora of tools. Dapr offers the features your application needs to accomplish specific tasks. These include secrets management, state management, eventing, and many more. This article will delve into the magical tools of dapr.io, focusing on secrets management.

### **Secrets Management**

Secret Management enables your application to securely access confidential information. These confidential details typically include sensitive information such as passwords, API keys, or database connection strings. Dapr.io assists in handling secrets management in a simple and secure manner. dapr.io possesses powerful features to accomplish these tasks effectively.

#### **Multiple Secret Store Support**

Dapr.io supports a wide range of different Secret Stores, including Azure Key Vault, HashiCorp Vault, Redis, AWS Secrets Manager, AWS SSM Parameter Store, Kubernetes Secrets, Local Environment Variables, Local Files, AlibabaCloud OOS Parameter Store, and GCP Secret Manager. dapr.io ensures secure and stable access to these secret stores for your application. It provides the flexibility needed when you have to use multiple secret stores.

#### **Secret Management APIs**

Dapr.io provides REST APIs for secrets management. These APIs allow you to add, retrieve, update, and delete secrets. This enables your application to dynamically manage secrets as needed.

#### **Automatic Secret Refresh**

Dapr.io can automatically renew secrets, ensuring that they remain up-to-date and secure continuously. For instance, an API key can be automatically renewed after a specific period, enhancing the security and freshness of secrets.

#### **Secret Versioning**

Dapr.io supports secret versions, providing a convenient way to store and retrieve multiple versions of a secret. This allows you to roll back secret changes and revert to previous versions when needed.

#### **Secret Cache**

Dapr.io can renew and cache secrets within a specific timeframe. This capability enhances performance when continuous access to secrets is required, reducing the need for repetitive calls.

#### **Secret Scoping**

Dapr.io allows you to manage secrets under different scopes. This ensures that a specific microservice or application can only access designated secrets, providing a granular control over secret access.

#### **Secret Rotation**

Dapr.io supports events for secret changes, enabling your application to be automatically notified when a secret is modified.

**Role-Based Access Control (RBAC)**

Dapr.io offers Role-Based Access Control (RBAC) to restrict access to confidential information. This ensures that the application can only access specific secrets authorized for certain users.

#### **Temporary Secrets**

It allows you to create temporary secrets that are valid for a specific duration. This feature enables you to generate secrets for a limited time and automatically destroy them afterward, enhancing security.

### **How Does Secret Management Work?**

Dapr initially stores relevant confidential information (such as usernames, passwords, connection strings, etc.) in a designated secret store. These secret stores can vary based on the project, with Dapr supporting Azure Key Vault, HashiCorp Vault, Redis, AWS Secrets Manager, AWS SSM Parameter Store, Kubernetes Secrets, Local Environment Variables, Local Files, AlibabaCloud OOS Parameter Store, and GCP Secret Manager.

Access to secrets stored in these secret stores is facilitated through the Dapr Secret Management APIs. Consequently, secrets can be read, updated, deleted, and new ones can be added.

Secrets retrieved through the Secret APIs are then used where needed within the application.

Dapr achieves all this through four fundamental building blocks: Secret Stores, Secret Building Blocks, Dapr CLI…

* **Secret Stores:** These are responsible for storing and managing secrets, providing a secure interface for access.
* **Secret Building Blocks:** This category includes APIs and other tools that facilitate access to Secret Stores.
* **Dapr CLI:** Apart from APIs, Dapr CLI provides command-line tools for managing secrets stored in Secret Stores.

### **Demo**

Let’s experience Dapr’s magical tool in an application. Of course, the application will be a .NET application :)

#### **Scenario**

Let’s say we have secrets that we want to store in a local file. To access these secrets and to more easily test Dapr features, let’s design an API and explore the powerful capabilities of Dapr.

#### **Setup**

Let’s start the process by assuming that we have already obtained Docker and .NET 6 SDK. First, we will begin with the installation of Dapr. While performing these steps, I will be focusing on the MacOS operating system and Apple M2 silicon processor. However, I will include the necessary guides for other operating systems as well.

#### **Install the Dapr CLI**

Before starting the demo, it is necessary to install Dapr CLI. For the latest version of CLI, we must run the following command in the /usr/local/bin file path.

wget -q <https://raw.githubusercontent.com/dapr/cli/master/install/install.sh> -O - | /bin/bash

We now have the latest version of Dapr CLI.

Since we can find the most suitable instructions for this installation in dapr Docs, I am adding the guide link below for other processors and operating systems.

You can find the link to the CLI installation instructions [here](https://docs.dapr.io/getting-started/install-dapr-selfhost/).

#### **Install the Dapr**

Now that we have the Dapr CLI, we can start installing Dapr. The following command will be sufficient to install Dapr.

dapr init

Or for a specific version installation

dapr init --runtime-version 1.10.3

The important thing at this point is that the CLI and runtime must be installed in the **same version**.

#### **Verify containers are running**

As mentioned earlier, the **dapr init** command launches several containers that will help you get started with Dapr. Verify you have container instances with **daprio/dapr**, **openzipkin/zipkin**, and **redis** images running.

docker ps

#### **Code Code Code**

Now that the installations are completed, the wonderful project can begin. Let’s start by creating the Secret model. The model will contain Key and Value.

public class SecretModel  
{  
 public string Key { get; set; }  
 public string Value { get; set; }  
}

Next, we will create some configuration files for dapr. Let’s create a “components” folder in the same directory as the project. Next, let’s create the “localSecretStore.yaml” file in this folder. This file is the configuration of where dapr will store secrets locally and in which directory the secrets are kept.

apiVersion: dapr.io/v1alpha1  
kind: Component  
metadata:  
 name: local-secret-store  
spec:  
 type: secretstores.local.file  
 metadata:  
 - name: secretsFile  
 value: ./components/secrets.json

Let’s create the secrets.json file in the same folder. The secrets we want to store locally here must be in json format. I have defined two completely fictitious secrets.

{  
 "my-secret": "important-secret",  
 "another-secret": "another-important-secret"  
}

The code below will be very useful for the Dapr Nuget package.

dotnet add package Dapr.AspNetCore

The following code will be useful for a service and controller that returns the secret given as a key in the request.

public class SecretService : ISecretService  
{  
 private readonly DaprClient \_daprClient;  
  
 public SecretService(DaprClient daprClient)  
 {  
 \_daprClient = daprClient;  
 }  
  
 public async Task<string> GetSecretAsync(string key)  
 {  
 var secret = await \_daprClient.GetSecretAsync("local-secret-store", key);  
 return secret[key];  
 }  
}[ApiController]  
[Route("[controller]")]  
public class SecretController : ControllerBase  
{  
 private readonly ISecretService \_secretService;  
  
 public SecretController(ISecretService secretService)  
 {  
 \_secretService = secretService;  
 }  
  
 [HttpGet("{key}")]  
 public async Task<IActionResult> GetSecret(string key)  
 {  
 var secret = await \_secretService.GetSecretAsync(key);  
 return Ok(secret);  
 }  
}

The Dapr and SecretService we use should be added in program.cs like this.

builder.Services.AddSingleton<DaprClient>(\_ => new DaprClientBuilder().Build());  
builder.Services.AddScoped<ISecretService, SecretService>();  
builder.Services.AddControllers().AddDapr();

#### **Run**

A simple demo that will test the Dapr secret management component and fetch the secrets stored locally is now ready. Let’s test it by running.

We need the following command to run the application. With this command, we make a name for the application and the necessary port settings. We are also running our dapr and dotnet projects at the same time.

dapr run --app-id mydaprsecretapp --app-port 5019 --placement-host-address localhost:50006 --dapr-grpc-port 3501 --app-protocol grpc --app-protocol http --resources-path ./components --dapr-http-port 3500 dotnet run

#### **Result**

When we ran the run command above, our application started running on port 5019 and dapr http on port 3500 at the same time.

Now, when we send the relevant request to the endpoint we created, it will be returned as a secret response that we have stored locally.

### **Inferences**

In this article, while talking about the wonderful features of dapr secret management, we also designed a simple, guiding demo that provides access to the secrets we store locally. Based on this demo, an application can be developed that provides access to all secrets stored not only locally but also on the platforms supported by dapr.

Please contact me if you would like to add to or correct any parts of this article. All the best…