THE IMPLEMENTATION OF A METHOD FOR WORKING WITH SENSITIVE DATA USED BY VARIOUS SERVICES AND SYSTEMS

Ulyana Dzelendzyak, PhD, As.-Prof., Nazar Mashtaler, PhD Student
Lviv Polytechnic National University, Ukraine; e-mail: u.dzelendzyak@gmail.com

Abstract. The article describes the method of working with sensitive data used by various services and systems, including CRM/ERM systems, as well as the implementation of storing this data using the classic .NET FRAMEWORK. The main driver of this initiative is a missing centralized repository for connection strings to various systems like databases, CRM/ERM systems (for example. Netsuite or Salesforce), system variables, other sensitive info (for example tokens), and third-party components. The problem here is that each application has stored these connection strings in its configuration (usually in the web. config). It means one connection string is multiplied in many places and if there is a change in credentials, for example, the change must be done in all these application configurations. Finally, it would be better to have any registry of which connection string is used where. This is adding complexity for global updates, and it also doesn't help with security (since credentials to production systems are in the configuration and thus in source control, where they are visible to anybody).

Key words: Net Framework security, storing sensitive information, secrets, securing Microservices and Web Applications

1. Introduction

Nowadays, the processing and storage of sensitive data play an important role. To connect with protected resources and other services and systems .NET applications typically need to use tokens, connection strings, passwords, or other credentials that contain sensitive information. These sensitive pieces of information are called secrets. It's a best practice to not include secrets in source code and to make sure not to store secrets in source control. Instead, it would be better to store read the secrets in more secure locations [2]. A good approach would be to separate the secrets for accessing development and staging resources from the ones used for accessing production resources because different individuals will need access to those different sets of secrets [1, 5].

2. Drawbacks

The described approach suits well for big systems that have parts implemented using different technologies and for systems that are implemented mostly using the classic .Net framework. The alternatives for smaller systems or systems implemented via ASP.NET Core also will be mentioned but the main focus would be on the classic .Net framework.

3. Goal

The goal of the current article is to introduce the approach of implementing a centralized and secured repository for sensitive info which could fit enterprise-level systems for different technologies, components, and CRM/ERM systems.

4. Implementation of Configuration Provider

Configuration Provider is a client library that provides a persistent store for all stored configurations. The library contains a registry of requested configurations by applications, which allows us to see what configuration is used by what application. Implemented the ability to locally store the configuration for cases when the Configuration provider is not available for any reason to allow the application to start without it. Each environment is separated to have a specific Configuration provider for Production, QA, and Staging. Also, the app provides access control for particular keys based on Active Directory (AD) Groups. Active Directory has two forms of common security principles: user accounts and computer accounts. These accounts represent a physical entity that is either a person or a computer. A user account also can be used as a dedicated service account for some applications. Security groups are a way to collect user accounts, computer accounts, and other groups into manageable units. Fig. 1 shows the concept of the Configuration Provider process.

The management console is a simple web application with strongly restricted access. It should provide a few screens where the user should be able to:

- See all configurations
- Add/Edit/Remove configuration
- See apps, which are using a particular configuration
- Permit/Deny app/AD group to be able to get the configuration
- See the log of requests for configuration

Configuration Provider Client is a standard RestSharp client which is hiding REST service and which can return configuration in C# objects. It should also allow the creation of a local store of the configuration to ensure that the app is the particular configuration in case of the Configuration provider's unavailability.

We have several separate applications: 'Service Tier', 'Customer Portal', 'Orchestrations', and so on. Such applications could be not related to each other but they have one common characteristic. It is injected into the Configuration Provider package that contains the method 'GetConfig'. The Configuration Provider is injected into our .Net applications via the NuGet package.

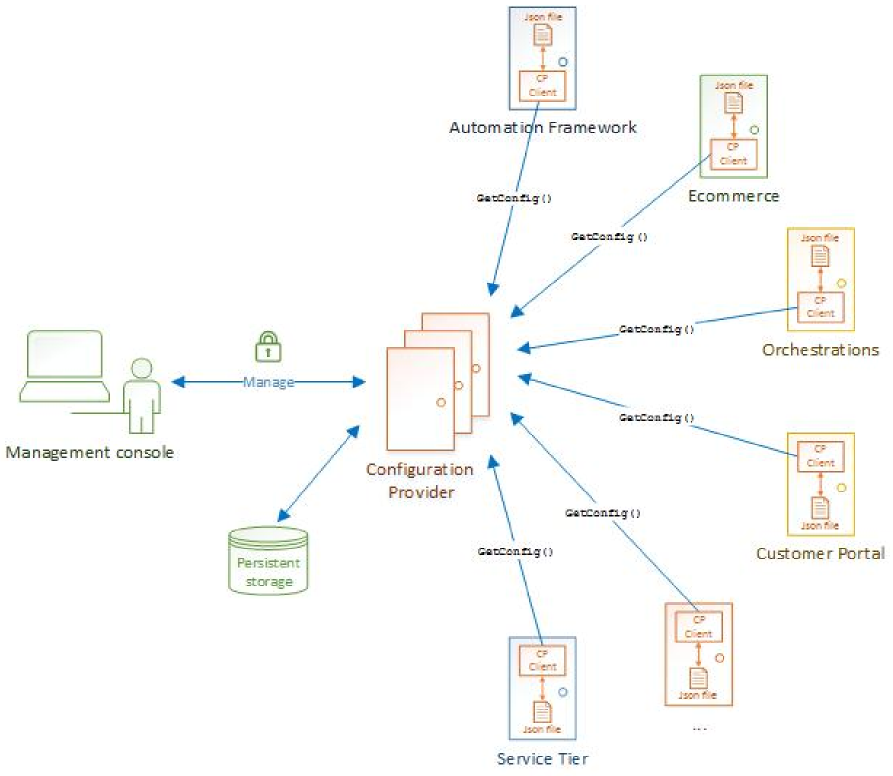


Fig.1. Concept of Process

A NuGet package contains reusable code that other developers have made available to you for use in your projects. You can install a NuGet package in a Microsoft Visual Studio project by using the NuGet Package Manager, the Package Manager Console, or the .NET CLI. After you install a NuGet package, you can then make a reference to it in your code using <namespace> statement, where <namespace> is the name of the package you're using. After you've referred, you can then call the package through its API [6].

An essential tool for any modern development platform is a mechanism through which developers can create, share, and consume useful code. Often such code is bundled into "packages" that contain compiled code (as DLLs) along with other content needed in the projects that consume these packages.

For .NET (including .NET Core), the Microsoft-supported mechanism for sharing code is NuGet, which defines how packages for .NET are created, hosted, and consumed, and provides the tools for each of those roles.

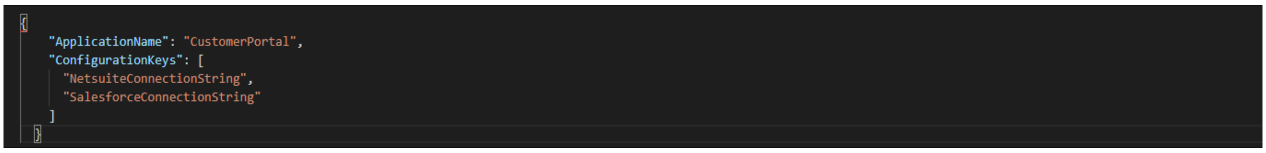
Put simply, a NuGet package is a single ZIP file with the .nupkg extension that contains compiled code (DLLs), other files related to that code, and a descriptive manifest that includes information like the package's version number. Developers with code to share create packages and publish them to a public or private host. Package consumers obtain those packages from suitable hosts, add them to their projects, and then call a package's functionality in their project code. NuGet itself then handles all of the intermediate details.

Because NuGet supports private hosts alongside the public nuget.org host, you can use NuGet packages to share code that's exclusive to an organization or a work group. You can also use NuGet packages as a convenient way to factor your code for use in nothing but your projects. In short, a NuGet package is a shareable unit of code but does not require nor imply any particular means of sharing [7].

In its role as a public host, NuGet itself maintains the central repository of over 100,000 unique packages at nuget.org. These packages are employed by millions of .NET/.NET Core developers every day. NuGet also enables you to host packages privately in the cloud (such as on Azure DevOps), on a private network, or even on just your local file system. By doing so, those packages are available to only those developers that have access to the host, giving you the ability to make packages available to a specific group of consumers. The options are explained for Hosting your NuGet feeds. Through configuration options, you can also control exactly which hosts can be accessed by any given computer, thereby ensuring that packages are obtained from specific sources rather than a public repository like nuget.org.

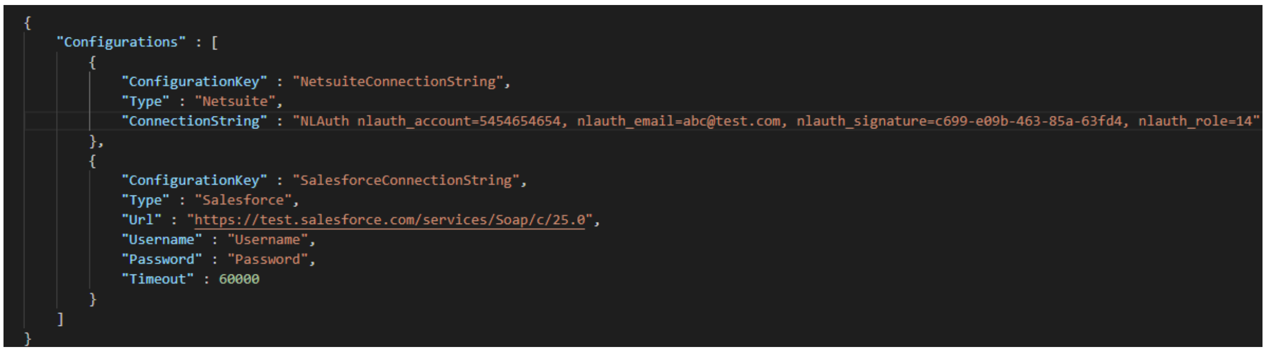
Whatever its nature, a host serves as the point of connection between package creators and package consumers. Creators build useful NuGet packages and publish them to a host. Consumers then search for useful and compatible packages on accessible hosts, downloading and including those packages in their projects. Once installed in a project, the packages' APIs are available to the rest of the project code [6, 7].

To have a whole picture of how it works let's see how we get all-need Configuration Keys('Netsuite ConnectingString', 'SalesforceConnectingString') for the 'Customer Portal' app. Fig. 2 shows the request that gets Keys for the 'Customer Portal' app and Fig. 3 shows the response that returns sensitive data.



```
{
    "ApplicationName": "CustomerPortal",
    "ConfigurationKeys": [
      "NetsuiteConnectionString",
      "SalesforceConnectionString"
    ]
  }
```

Fig.2. Request example



```
{
    "Configurations" : [
        {
            "ConfigurationKey" : "NetsuiteConnectionString",
            "Type" : "Netsuite",
            "ConnectionString" : "NLAuth nlauth_account=5454654654, nlauth_email=abc@test.com, nlauth_signature=c699-e09b-463-85a-63fd4, nlauth_role=14"
        },
        {
            "ConfigurationKey" : "SalesforceConnectionString",
            "Type" : "Salesforce",
            "Url" : "https://test.salesforce.com/services/Soap/c/25.0",
            "Username" : "Username",
            "Password" : "Password",
            "Timeout" : 60000
        }
    ]
}
```

Fig.3. Response example

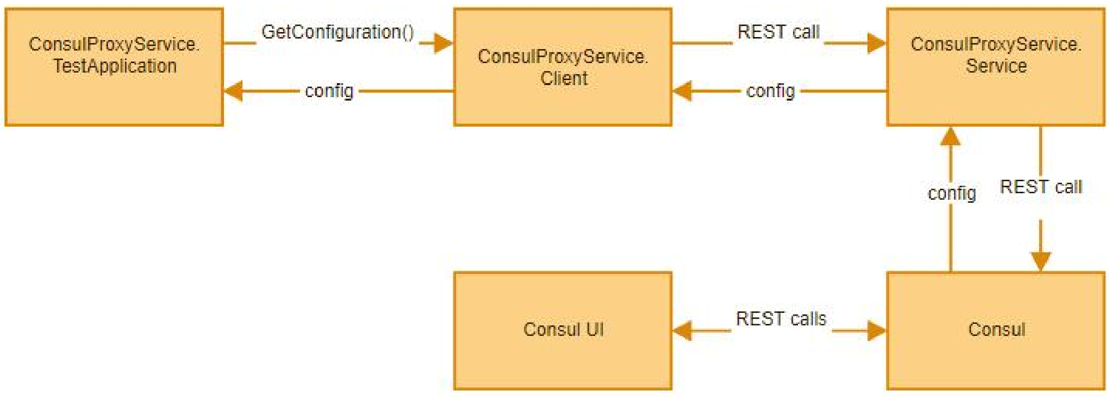


Fig.4. Usage workflow

Table 1. Description of public methods

Method	Description
PreloadConfiguration(configurationKeys)	Preloads the configuration to the internal cache and it is expected that this method will be called within the application start
GetConfiguration<type>(key)	Returns the configuration of a particular type and key. If the configuration doesn't exist or the app doesn't have permission on it, then null is returned
SetConfigurationProviderUrl(url)	Allows to change URL to configuration provider
SetDomainUser(username, password)	Set the Domain username and password for the user, which should connect to the Configuration provider

The usage workflow is described in Fig. 4

Some of the applications call ConsultProxy Service. The client that takes a request validates it and sends it ConsultProxyService.Service that connects our Configuration Provided with the Consul/Consul UI.

If the app and Configuration Key(s) exist and the request was valid, Consul sends sensitive data via ConsultProxyService.Service and ConsultProxyService.Client.

Consul/Consul UI manages Secrets and Protect Sensitive Data. Secure, store and tightly control access to tokens, passwords, certificates, and encryption keys for protecting secrets and other sensitive data using a UI, CLI, or HTTP API. Also, Consul/Consul UI is used as a storage and UI for editing keys ConsultProxyService. The client is responsible for validation and sending responses. ConsultProxyService.Service responsible for working with Consul that is 3rd party component.

Let's take a look at the class diagram for the implementation of the Configuration Provider shown in Fig. 5. Table 1 below described all public methods.

5. Alternatives

The configuration management database (CMDB) is an ITIL term for a database used by an organization to store information about hardware and software assets (commonly referred to as configuration items). It is useful to break down configuration items into logical layers. This database acts as a data warehouse for the organization and also stores information regarding the relationships among its assets. The CMDB provides a means of understanding the organization's critical assets and their relationships, such as information systems, upstream sources or dependencies of assets, and the downstream targets of assets[3]. An example of the configuration management database is shown in Fig. 6.

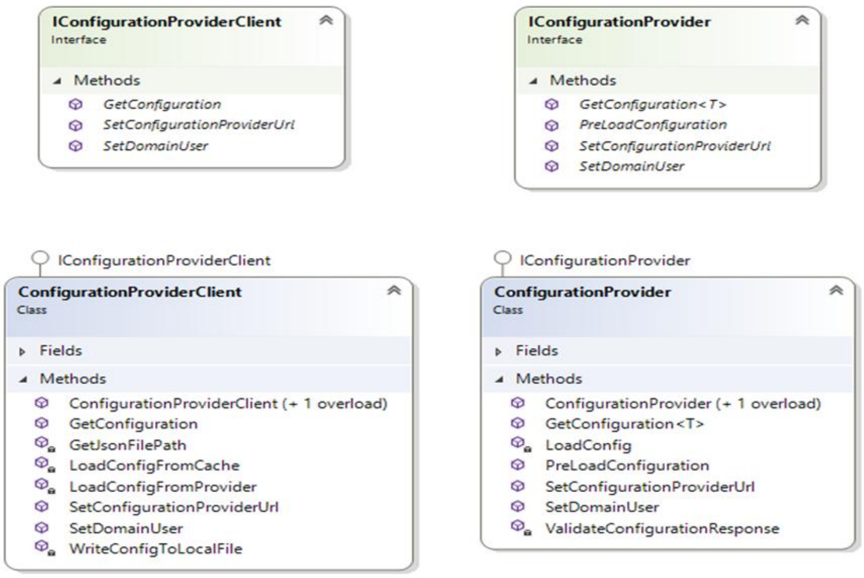


Fig.5. Configuration Provider Class Diagram

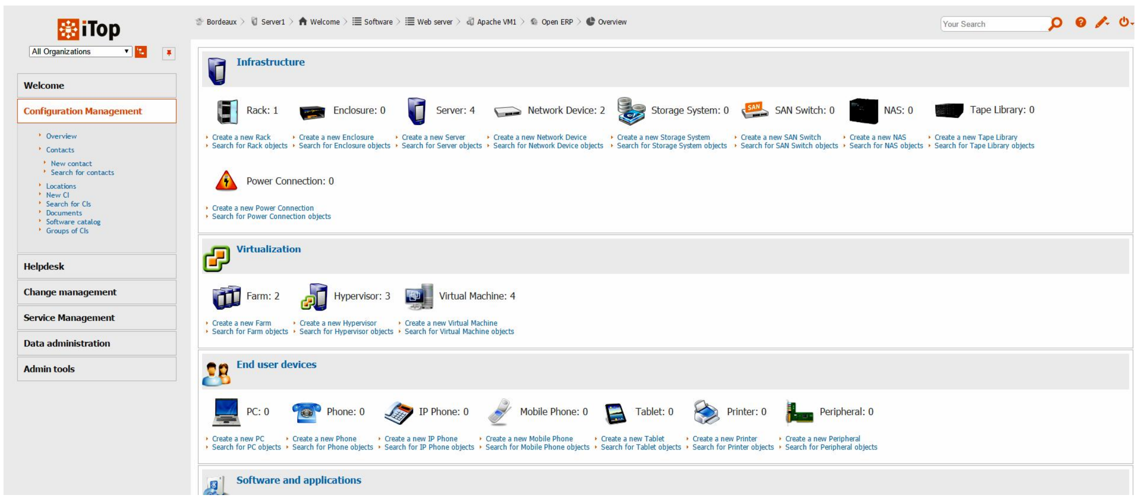


Fig.6. Example of CMDB

CMDB Features:

- Fully configurable CMDB
- HelpDesk and Incident Management
- Service and Contract Management
- Change Management
- Configuration Management
- Automatic impact analysis
- CSV import tool for all data
- Data synchronization (data federation)

CMDB's Pros&Cons are described in Table 2.

Table 2. CMDB Pros&Cons

Pros	Cons
• Could allow more complex management than classical CM tools	• Too heavy • Expensive

.NET Core has its secret manager. The ASP.NET Core Secret Manager tool provides another method of keeping secrets out of source code during development[1]. To use the Secret Manager tool, install the package Microsoft.Extensions.Configuration.SecretManager in your project file. Once that dependency is present and has been restored, the .NET user-secrets command can be used to set the value of secrets from the command line. These secrets will be stored in a JSON file in the user's profile directory (details vary by OS), away from the source code.

To have a bigger picture let's take a look at cloud solutions that are popular nowadays. Let's overview 2 the most popular Azure Key Vault and AWS Secrets Manager.

Azure Key Vault is a secret store: a centralized cloud service for storing app secrets; configuration values like passwords and connection strings that must remain secure at all times[4]. Key Vault helps you control your apps' secrets by keeping them in a single central location and providing secure access, permissions control, and access logging.

The main benefits of using Key Vault are:

- Separation of sensitive app information from other configurations and code, reducing the risk of accidental leaks
- Restricted secret access with access policies tailored to the apps and individuals that need them
- Centralized secret storage, meaning required changes only have to be made in one place
- Access logging and monitoring to help you understand how and when secrets are accessed

Secrets are stored in individual vaults, which are Azure resources used to group secrets together. Secret access and vault management are accomplished via a REST API, which is also supported by all of the Azure management tools, and client libraries available for many popular languages. Every vault has a unique URL where its API is hosted.

AWS Secrets Manager helps you protect the sensitive data needed to access your applications, services, and IT resources. The service allows you to easily switch, manage, and retrieve database credentials, API keys, and other sensitive data throughout its lifecycle[5]. Users and applications retrieve sensitive data with a call to the Secrets Manager APIs, eliminating the need to encode sensitive information in plain text. Secrets Manager offers sensitive data shifting with built-in integration for Amazon Relational Database Service (Amazon RDS), Amazon Redshift, and Amazon DocumentDB. Additionally, the service can be extended to other types of sensitive data, including API keys and OAuth tokens. Secrets Manager also enables you to control access to sensitive data through fine-grained permissions and centrally audit sensitive data rotation for resources located in the AWS Cloud, third-party services, or on-premises.

6. Conclusions

The advantages of described approach for storing secrets are the next. It provides:

- A persistent store for all stored configurations.
- A configuration based on the requested key; for example, the connection string to the CRM system.
- Access control for particular keys based on AD Groups.
- A management console (web app) with restricted access to allow user-friendly management of the configuration.
- A registry of requested configurations by applications, which enables monitoring of the certain configuration used by the particular application.
- A client library with the ability to locally store the configuration for cases when the Configuration provider is not available for any reason to allow the application to start without it
- A separation for each environment; we should support a specific Configuration provider for Production, QA, and Staging.

7. Gratitude

The authors express their gratitude to the Staff members of the Departments of Computerized Automatic Systems and the Information-Measuring Technologies of Lviv Polytechnic National University.

8. Conflict of Interest

The authors state that there are no financial or other potential conflicts regarding this work.

References

[1] C. de la Torre, B. Wagner, M. Rousos, .NET Microservices Architecture for Containerized NET Applications, One Microsoft Way Redmond, Washington 98052-6399, 2022. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/

[2] Safe storage of app secrets in development in ASP.NET, Microsoft 2022. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-6.0&tabs=windows

[3] CMDB Design Guidance, Servicenow, 2020. https://www.servicenow.com/content/dam/servicenow-assets/public/en-us/doc-type/resource-center/white-paper/wp-cmdb-design-guidance.pdf

[4] Azure Key Vault configuration provider in ASP.NET Core, Microsoft 2022. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/key-vault-configuration?view=aspnetcore-6.0

[5] AWS Secrets Manager: User Guide, Amazon Web Services, Inc, 2022. https://docs.aws.amazon.com/managedservices/latest/userguide/secrets-manager.html

[6] Quickstart: Install and use a NuGet package, Microsoft 2022. [Online]. Available: https://learn.microsoft.com/en-us/nuget/quickstart/install-and-use-a-package-in-visual-studio

[7] An introduction to NuGet, Microsoft 2022. [Online]. Available: https://learn.microsoft.com/en-us/nuget/what-is-nuget