

**ANG-274** *POC Document*

|  |  |
| --- | --- |
| POC Title: | **External Configuration Store Pattern** |
| Created by: | Md. Sahidul Islam, Jr. Software Engineer |
| Reviewed by: | Md. Mahedee Hasan, Head of Soft. Dev Eng |
| Approved by: | Md. Mahedee Hasan, Head of Soft. Dev Eng |
| Confidentiality level: | Internal |

**DOCUMENT CHANGE HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Description of change** |
| 21-Sept-2022 | 1.0 | Md. Sahidul Islam | Initial Draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[**Context and problem** 1](#_Toc114650519)

[Solution 2](#_Toc114650520)

[Issues and considerations 2](#_Toc114650521)

[When to use this pattern 4](#_Toc114650522)

**External Configuration Store Pattern**

**Context and problem**

The majority of application runtime environments include configuration information that's held in files deployed with the application. In some cases, it's possible to edit these files to change the application behavior after it's been deployed. However, changes to the configuration require the application be redeployed, often resulting in unacceptable downtime and other administrative overhead.

Local configuration files also limit the configuration to a single application, but sometimes it would be useful to share configuration settings across multiple applications. Examples include database connection strings, UI theme information, or the URLs of queues and storage used by a related set of applications.

It's challenging to manage changes to local configurations across multiple running instances of the application, especially in a cloud-hosted scenario. It can result in instances using different configuration settings while the update is being deployed.

In addition, updates to applications and components might require changes to configuration schemas. Many configuration systems don't support different versions of configuration information.

## Solution

Store the configuration information in external storage, and provide an interface that can be used to quickly and efficiently read and update configuration settings. The type of external store depends on the hosting and runtime environment of the application. In a cloud-hosted scenario it's typically a cloud-based storage service or dedicated configuration service, but could be a hosted database or other custom system.

The backing store you choose for configuration information should have an interface that provides consistent and easy-to-use access. It should expose the information in a correctly typed and structured format. The implementation might also need to authorize users' access in order to protect configuration data, and be flexible enough to allow storage of multiple versions of the configuration (such as development, staging, or production, including multiple release versions of each one).

Many built-in configuration systems read the data when the application starts up, and cache the data in memory to provide fast access and minimize the impact on application performance. Depending on the type of backing store used, and the latency of this store, it might be helpful to implement a caching mechanism within the external configuration store. For more information, see the [Caching Guidance](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/dn589802(v=pandp.10)). The figure illustrates an overview of the External Configuration Store pattern with optional local cache.

## Issues and considerations

Consider the following points when deciding how to implement this pattern:

Choose a backing store that offers acceptable performance, high availability, robustness, and can be backed up as part of the application maintenance and administration process. In a cloud-hosted application, using a cloud storage mechanism or dedicated configuration platform service is usually a good choice to meet these requirements.

Design the schema of the backing store to allow flexibility in the types of information it can hold. Ensure that it provides for all configuration requirements such as typed data, collections of settings, multiple versions of settings, and any other features that the applications using it require. The schema should be easy to extend to support additional settings as requirements change.

Consider the physical capabilities of the backing store, how it relates to the way configuration information is stored, and the effects on performance. For example, storing an XML document containing configuration information will require either the configuration interface or the application to parse the document in order to read individual settings. It'll make updating a setting more complicated, though caching the settings can help to offset slower read performance.

Consider how the configuration interface will permit control of the scope and inheritance of configuration settings. For example, it might be a requirement to scope configuration settings at the organization, application, and the machine level. It might need to support delegation of control over access to different scopes, and to prevent or allow individual applications to override settings.

Ensure that the configuration interface can expose the configuration data in the required formats such as typed values, collections, key/value pairs, or property bags.

Consider how the configuration store interface will behave when settings contain errors, or don't exist in the backing store. It might be appropriate to return default settings and log errors. Also consider aspects such as the case sensitivity of configuration setting keys or names, the storage and handling of binary data, and the ways that null or empty values are handled.

Consider how to protect the configuration data to allow access to only the appropriate users and applications. This is likely a feature of the configuration store interface, but it's also necessary to ensure that the data in the backing store can't be accessed directly without the appropriate permission. Ensure strict separation between the permissions required to read and to write configuration data. Also consider whether you need to encrypt some or all of the configuration settings, and how this'll be implemented in the configuration store interface.

Centrally stored configurations, which change application behavior during runtime, are critically important and should be deployed, updated, and managed using the same mechanisms as deploying application code. For example, changes that can affect more than one application must be carried out using a full test and staged deployment approach to ensure that the change is appropriate for all applications that use this configuration. If an administrator edits a setting to update one application, it could adversely impact other applications that use the same setting.

If an application caches configuration information, the application needs to be alerted if the configuration changes. It might be possible to implement an expiration policy over cached configuration data so that this information is automatically refreshed periodically and any changes picked up (and acted on).

While caching configuration data can help address transient connectivity issues with the external configuration store at application runtime, this typically doesn't solve the problem if the external store is down when the application is first starting. Ensure your application deployment pipeline can provide the last known set of configuration values in a configuration file as a fallback if your application cannot retrieve live values when it starts.

## When to use this pattern

This pattern is useful for:

* Configuration settings that are shared between multiple applications and application instances, or where a standard configuration must be enforced across multiple applications and application instances.
* A standard configuration system that doesn't support all of the required configuration settings, such as storing images or complex data types.
* As a complementary store for some of the settings for applications, perhaps allowing applications to override some or all of the centrally-stored settings.
* As a way to simplify administration of multiple applications, and optionally for monitoring use of configuration settings by logging some or all types of access to the configuration store.

