KodeKloud

Visit www.kodekloud.com to learn more.

Implementing Azure Key Vault

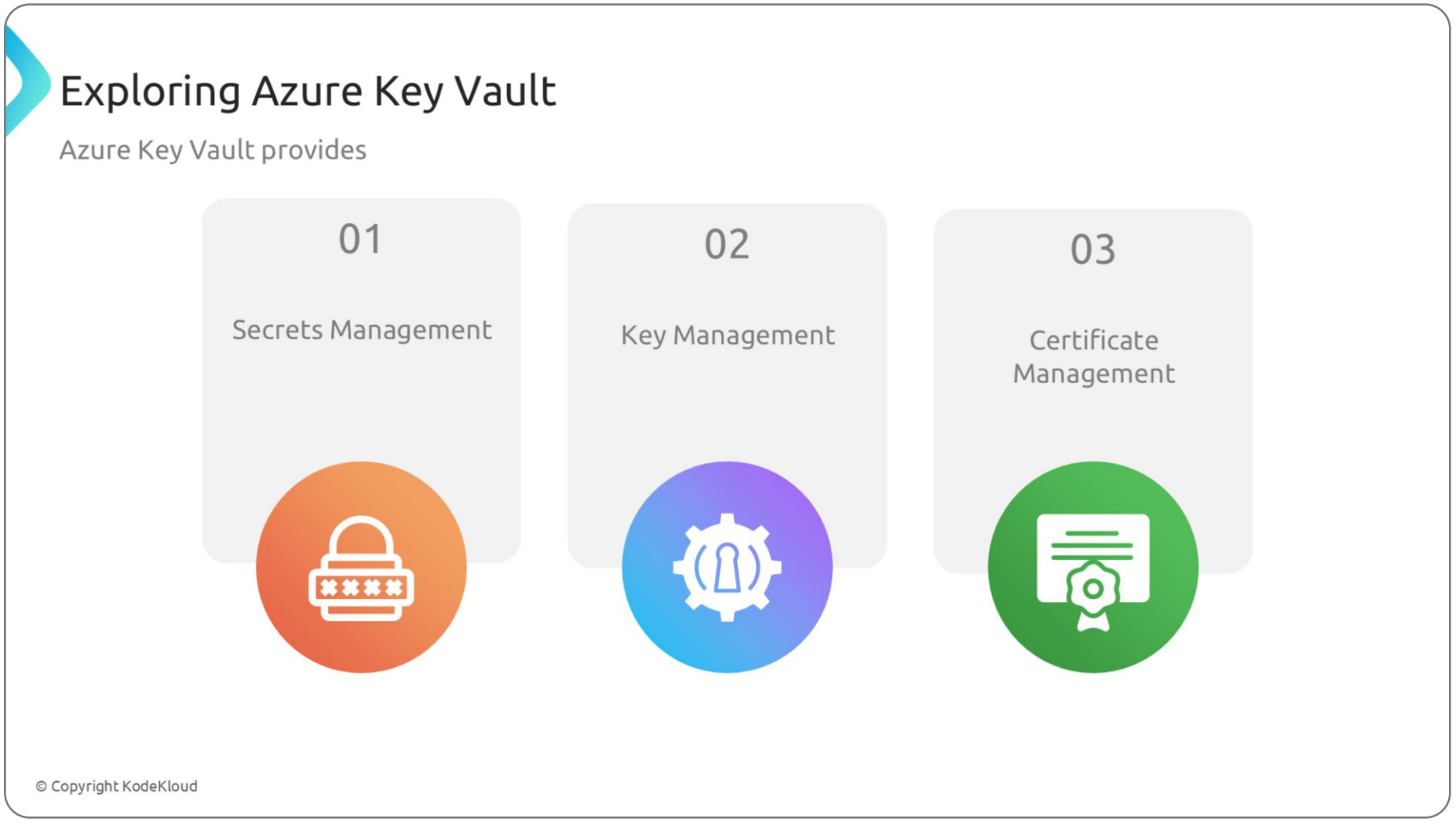
Introduction

01 Describe the benefits of using Azure Key Vault

02 Explain how to authenticate to Azure Key Vault

03 Set and retrieve a secret from Azure Key Vault by using Azure CLI

Exploring Azure Key Vault



Two types of containers: vaults and managed hardware security module(HSM) pools.

Azure Key Vault helps solve the following problems:
Secrets Management: Azure Key Vault can be used to Securely store and tightly control access to tokens, passwords, certificates, API keys, and other secrets
Key Management: Azure Key Vault can also be used as a Key Management solution. Azure Key Vault makes it easy to create and control the encryption keys used to encrypt your data.

Certificate Management: Azure Key Vault is also a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with Azure and your internal connected resources.

Azure Key Vault has two service tiers: Standard, which encrypts with a software key, and a Premium tier, which includes hardware security module(HSM)-protected keys.

Azure Key Vault can be configured to:
Archive to a storage account.
Stream to an event hub.
Send the logs to Azure Monitor logs.

Azure Key Vault simplifies the process of meeting these requirements by:
Removing the need for in-house knowledge of Hardware Security Modules
Scaling up on short notice to meet your organization’s usage spikes.
Replicating the contents of your Key Vault within a region and to a secondary region. Data replication ensures high availability and takes away the need of any action from the administrator to trigger the failover.
Providing standard Azure administration options via the portal, Azure CLI and PowerShell.
Automating certain tasks on certificates that you purchase from Public CAs, such as enrollment and renewal.

Exploring Azure Key Vault

Key benefits of Azure Key Vault

Two types of containers: vaults and managed hardware security module(HSM) pools.

Azure Key Vault helps solve the following problems:
Secrets Management: Azure Key Vault can be used to Securely store and tightly control access to tokens, passwords, certificates, API keys, and other secrets
Key Management: Azure Key Vault can also be used as a Key Management solution. Azure Key Vault makes it easy to create and control the encryption keys used to encrypt your data.

Certificate Management: Azure Key Vault is also a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with Azure and your internal connected resources.

Azure Key Vault has two service tiers: Standard, which encrypts with a software key, and a Premium tier, which includes hardware security module(HSM)-protected keys.

Azure Key Vault can be configured to:
Archive to a storage account.
Stream to an event hub.
Send the logs to Azure Monitor logs.

Azure Key Vault simplifies the process of meeting these requirements by:
Removing the need for in-house knowledge of Hardware Security Modules
Scaling up on short notice to meet your organization’s usage spikes.
Replicating the contents of your Key Vault within a region and to a secondary region. Data replication ensures high availability and takes away the need of any action from the administrator to trigger the failover.
Providing standard Azure administration options via the portal, Azure CLI and PowerShell.
Automating certain tasks on certificates that you purchase from Public CAs, such as enrollment and renewal.

Azure Key Vault – Best Practices

Azure Key Vault – Best Practices (1/2)

Authenticating to Azure Key Vault

Managed identities for Azure resources

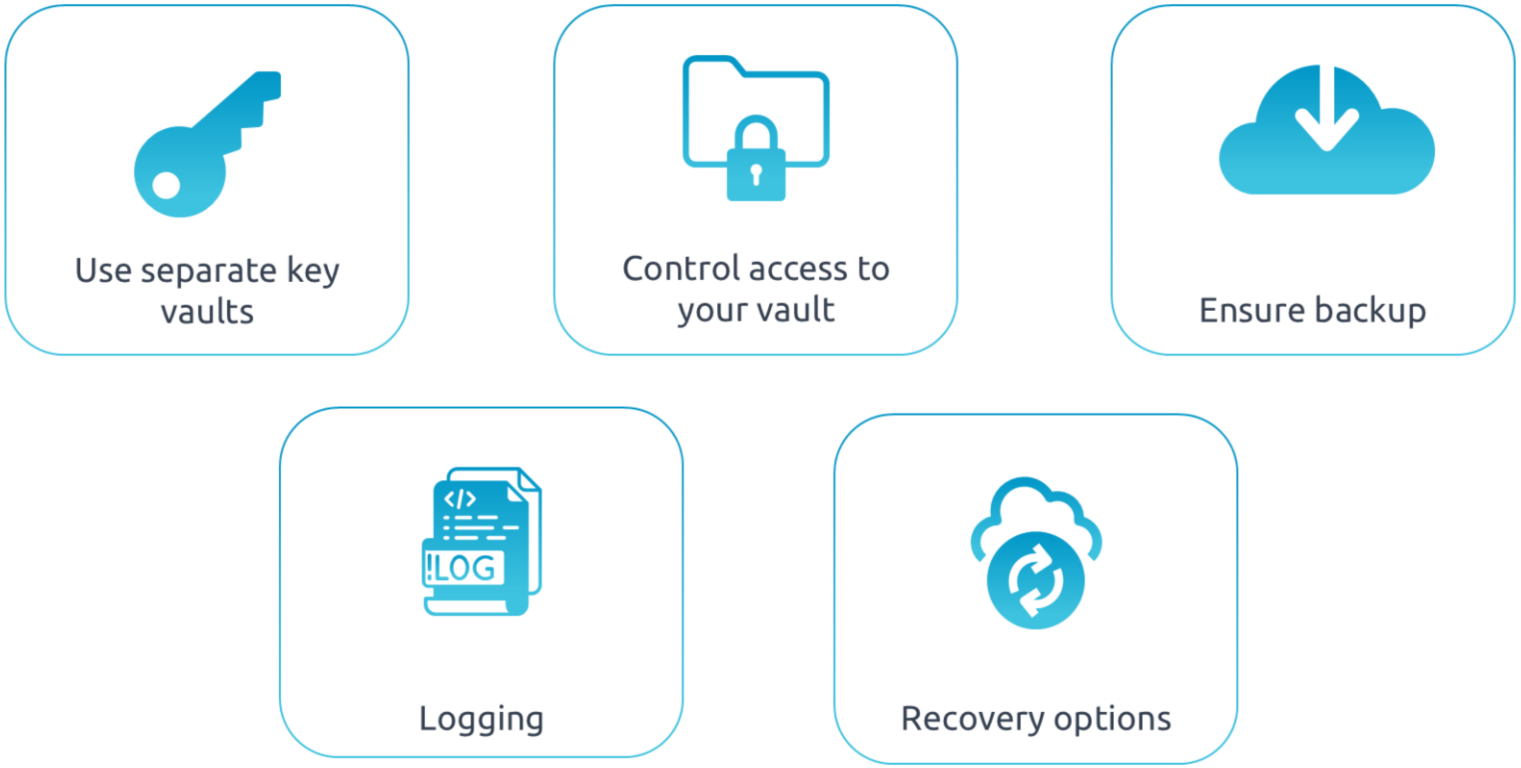
Service principal and certificate

Service principal and secret

To do any operations with Key Vault, you first need to authenticate to it. There are three ways to authenticate to Key Vault.

Managed identities for Azure resources: You can also assign identities to other Azure resources. Azure automatically rotates the service principal client secret associated with the identity. We recommend this approach as a best practice.
Service principal and certificate: We don't recommend this approach because the application owner or developer must rotate the certificate.
Service principal and secret: It's hard to automatically rotate the bootstrap secret that's used to authenticate to Key Vault.

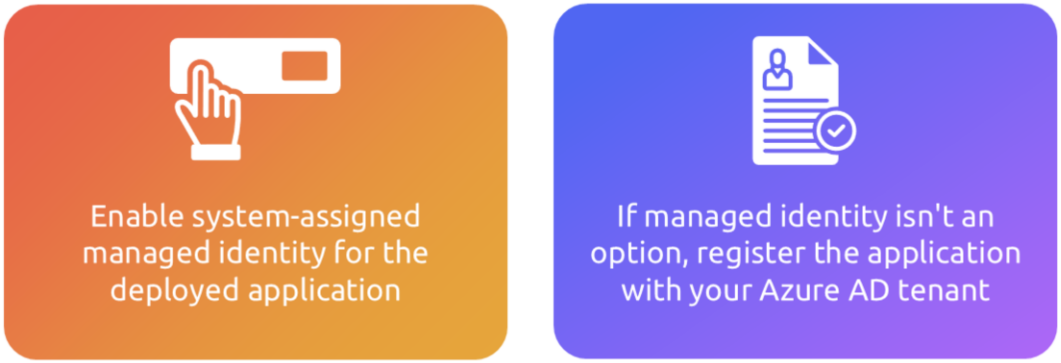
Azure Key Vault – Best Practices (2/2)



- Use separate key vaults: Recommended to use a vault per application per environment.
- Control access to your vault: Key Vault data is sensitive and business critical.
- Backup: Create regular back ups of your vault on update/delete/create of objects within a Vault.
- Logging: Be sure to turn on logging and alerts.
- Recovery options: Turn on soft-delete and purge protection if you want to guard against force deletion of the secret.

Authenticating to Azure Key Vault

Authenticating to Azure Key Vault (1/2)



Note: It is recommended to use a system-assigned managed identity.

Authenticating to Azure Key Vault (2/2)

Authentication to Key Vault in application code

.NET	Python	Java	JavaScript
Azure Identity SDK .NET	Azure Identity SDK Python	Azure Identity SDK Java	Azure Identity SDK JavaScript

Authentication to Key Vault with REST

Access tokens must be sent to the service using the HTTP Authorization header:

```
PUT /keys/MYKEY?api-version=<api_version> HTTP/1.1
Authorization: Bearer <access_token>
```


Key Vault SDK is using Azure Identity client library, which allows seamless authentication to Key Vault across environments with same code.
Additional resources
[Azure Key Vault developer's guide]
[Azure Key Vault availability and redundancy]

Implementing Managed Identities

Introduction

01 Configure managed identities

02 Differentiate between the two types of managed identities

03 Describe the flows for user- and system-assigned managed identities

Exploring Managed Identities

Exploring Managed Identities (1/2)

01

System-Assigned Managed Identity

02

User-Assigned Managed Identity

Types of managed identities

- A system-assigned managed identity is enabled directly on an Azure service instance.
- A user-assigned managed identity is created as a standalone Azure resource.

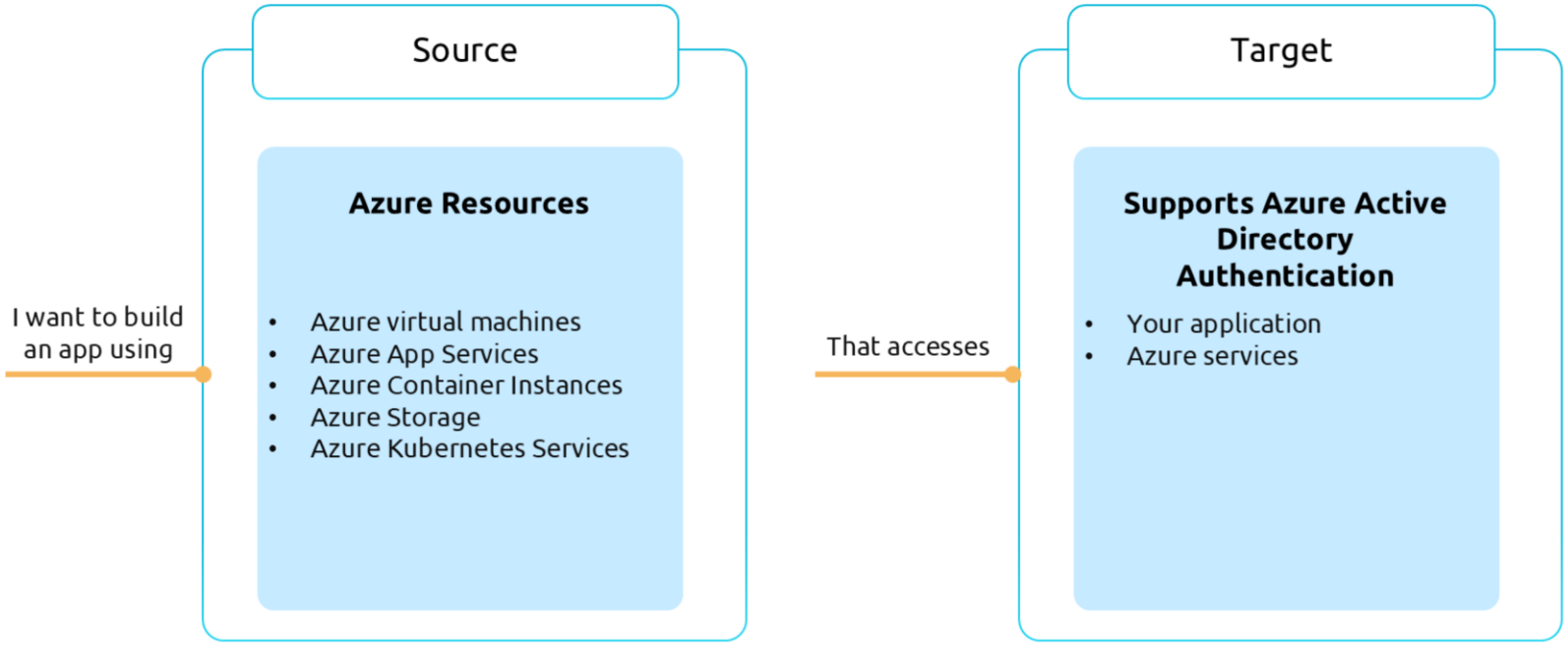
Exploring Managed Identities (1/2)

Characteristic	System-Assigned Managed Identity	User-Assigned Managed Identity
Creation	Created as part of an Azure resource (for example, an Azure virtual machine or Azure App Service)	Created as a stand-alone Azure resource
Lifecycle	Shared lifecycle with the Azure resource that the managed identity is created with; when the parent resource is deleted, the managed identity is deleted as well	Independent lifecycle; must be explicitly deleted
Sharing across Azure resources	Cannot be shared, but can only be associated with a single Azure resource	Can be shared; the same user-assigned managed identity can be associated with more than one Azure resource

Characteristics of managed identities

Exploring Managed Identities (2/2)

When to use managed identities



Managed identities for Azure resources can be used to authenticate to services that support Azure Active Directory authentication.
The rest of this module will use Azure virtual machines in the examples, but the same concepts and similar actions can be applied to any resource in Azure that supports Azure Active Directory authentication.

Managed Identity Authentication Flow

Managed Identity's Authentication Flow

Managed identity authentication flow In this lesson, let's walk through how a system assigned managed identity works with an Azure virtual machine The process is somewhat similar for user assigned managed identity as well The only differences the creation of user assigned manager identity because it's a standalone resource and you have to create it upfront anyways The flow that we are trying to observe today has several key steps

Managed Identity's Authentication Flow

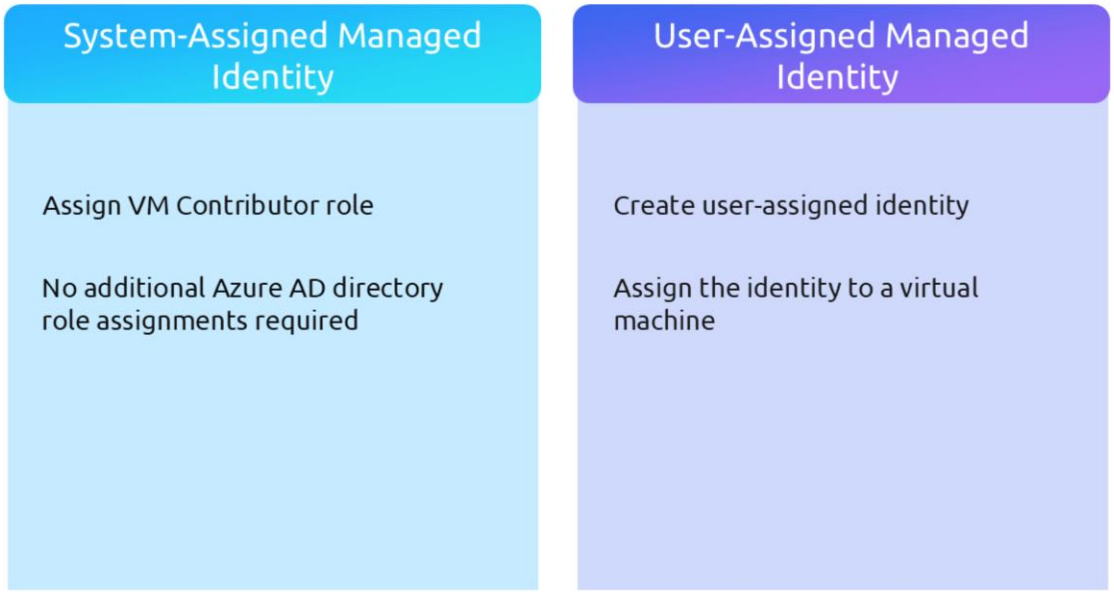
Understanding how a system-assigned managed identity works with an Azure virtual machine (VM)

1. ARM enables managed identity on VM
2. Creates AD service principal for VM
3. Configures identity on VM using Azure Instance Metadata Service
4. Grants VM access to Azure resources
5. Code on VM requests a token from Metadata service
6. Azure AD provides JSON Web Token access token
7. Code sends access token to Azure AD-authenticated service

The authentication flow for user-assigned identities follows the same pattern.

Configuring Managed Identities

Configuring Managed Identities (1/3)



- Managed identities for Azure resources can be used to authenticate to services that support Azure Active Directory authentication.
- The rest of this module will use Azure virtual machines in the examples, but the same concepts and similar actions can be applied to any resource in Azure that supports Azure Active Directory authentication.

Configuring Managed Identities (2/3)

```
az vm create --resource-group myResourceGroup \
    --name myVM --image win2016datacenter \
    --generate-ssh-keys \
    --assign-identity \
    --admin-username azureuser \
    --admin-password myPassword12
```


Enable system-assigned identity during resource creation

Enable user-assigned identity during resource creation

Configuring Managed Identities (3/3)

```
# Create the identity
az identity create -g myResourceGroup \
    -n myUserAssignedIdentity

# Assign identity during creation
az vm create \
    --resource-group <RESOURCE GROUP> \
    --name <VM NAME> \
    --image UbuntuLTS \
    --admin-username azureuser \
    --admin-password myPassword12 \
    --assign-identity <USER ASSIGNED IDENTITY NAME>
```

Enable system-assigned identity during resource creation

Enable user-assigned identity during resource creation

Acquiring Access Tokens

Acquiring an Access Token

A client application can request managed identities for Azure resources app-only access token for accessing a given resource.

The token is based on the managed identities for Azure resources service principal.

It is recommended to use DefaultAzureCredential.

Acquiring an Access Token

```
// When deployed to an azure host, the default azure credential will authenticate
the specified user assigned managed identity.

string userAssignedClientId = "<your managed identity client Id>";
var credential = new DefaultAzureCredential(new DefaultAzureCredentialOptions {
        ManagedIdentityClientId = userAssignedClientId });
var blobClient = new BlobClient(new Uri("URI"), credential);
```

Implementing Azure App Configuration

Introduction

01 Explore Azure App Configuration and its benefits

02 Understand how Azure App Configuration stores information

03 Implement feature management

04 Secure app configuration data

Exploring App Service Configuration

Azure App Configuration Service

- Centralized Management
- Dynamic Configuration
- Security
- Flexibility
- Scalability

Exploring Azure app configuration So Azure App configuration is a service that helps centralize and manage application settings and feature flags enabling you to dynamically configure your applications without redeployment It ensures your application configuration as secure, flexible, and scalable across multiple environments So in this slide, we will explore the benefits and key implementation scenarios of Azure App Service configuration

Azure App Configuration Service

Benefits of application configuration

01

A fully managed service that can be set up in minutes

02

Flexible key representations and mappings

03

Tagging with labels

04

Point-in-time replay of settings

05

Dedicated UI for feature flag management

06

Comparison of two sets of configurations

07

Enhanced security through Azure-managed identities

08

Complete data encryptions, at rest or in transit

09

Native integration with popular frameworks

Azure App Configuration Service

App Configuration implementation scenarios

Centralize management and distribution of hierarchical configuration data

Dynamically change application settings

Control feature availability in real time

Centralize management and distribution of hierarchical configuration data for different environments and geographies
Dynamically change application settings without the need to redeploy or restart an application
Control feature availability in real-time

Azure App Configuration Service

Programming Language and Framework	How to Connect
.NET Core and ASP.NET Core	App Configuration provider for .NET Core
.NET Framework and ASP.NET	App Configuration builder for .NET
Java Spring	App Configuration client for Spring Cloud
JavaScript/Node.js	App Configuration client for JavaScript
Python	App Configuration client for Python
Others	App Configuration REST API

The easiest way to add an App Configuration store to your application is through a client library that Microsoft provides.

Creating Paired Keys and Values

Creating Paired Keys and Values

Keys

Design Key Namespaces

Two general approaches: flat and hierarchical

Label Key

Key values in App Configuration can optionally have a label attribute

Version Key Values

App Configuration doesn't version key values automatically

Query Key Values

Each key value is uniquely identified by its key plus a label that can be null

There are two general approaches to naming keys used for configuration data: flat or hierarchical. These methods are similar from an application usage standpoint, but hierarchical naming offers a number of advantages:
Easier to read. Instead of one long sequence of characters, delimiters in a hierarchical key name function as spaces in a sentence.
Easier to manage. A key name hierarchy represents logical groups of configuration data.
Easier to use. It's simpler to write a query that pattern-matches keys in a hierarchical structure and retrieves only a portion of configuration data.

Creating Paired Keys and Values

Values

Values assigned to keys are also unicode strings

An optional user-defined content type is associated with each value

Configuration data stored in an App Configuration store, with all keys and values, is encrypted at rest and in transit

There are two general approaches to naming keys used for configuration data: flat or hierarchical. These methods are similar from an application usage standpoint, but hierarchical naming offers a number of advantages:

- Easier to read. Instead of one long sequence of characters, delimiters in a hierarchical key name function as spaces in a sentence.
- Easier to manage. A key name hierarchy represents logical groups of configuration data.
- Easier to use. It's simpler to write a query that pattern-matches keys in a hierarchical structure and retrieves only a portion of configuration data.

Managing Application Features

Managing Application Features

Basic concepts

Feature Flag

A variable with a binary state of on/off

Feature Manager

An application package that handles the lifecycle of all feature flags in an application

Filter

A rule for evaluating the state of a feature flag

Feature management is a modern software-development practice that decouples feature release from code deployment and enables quick changes to feature availability on demand.

Managing Application Features

Components that implement effective feature management

01

An application that makes use of feature flags

02

A separate repository that stores the feature flags and their current states

Feature management is a modern software-development practice that decouples feature release from code deployment and enables quick changes to feature availability on demand.

Managing Application Features

Feature flag declaration

01

Has two parts: a name and a list of one or more filters that are used to evaluate if a feature's state is on

02

When a feature flag has multiple filters, the filter list is traversed in order until one of the filters determines the feature should be enabled

03

The feature manager supports appsettings.json as a configuration source for feature flags

Managing Application Features

Feature flag repository

01

To use feature flags effectively, externalize all the feature flags used in an application

02

Azure App Configuration is designed to be a centralized repository for feature flags

Securing App Configuration Data

Securing App Configuration Data

Encrypt configuration data by using customer-managed keys

01

Standard-tier Azure App Configuration instance

02

Azure Key Vault with soft-delete and purge-protection features enabled

03

An RSA or RSA-HSM key within the Key Vault

The following components are required to successfully enable the customer-managed key capability for Azure App Configuration:
Standard tier Azure App Configuration instance
Azure Key Vault with soft-delete and purge-protection features enabled
An RSA or RSA-HSM key within the Key Vault: The key must not be expired, it must be enabled, and it must have both wrap and unwrap capabilities enabled

Once these resources are configured, two steps remain to allow Azure App Configuration to use the Key Vault key:

Assign a managed identity to the Azure App Configuration instance
Grant the identity GET, WRAP, and UNWRAP permissions in the target Key Vault's access policy.

Securing App Configuration Data

Encrypt configuration data by using customer-managed keys

01

Assign a managed identity to the Azure App Configuration instance

02

Grant the identity permissions in the target Key Vault's access policy

The following components are required to successfully enable the customer-managed key capability for Azure App Configuration:
Standard tier Azure App Configuration instance
Azure Key Vault with soft-delete and purge-protection features enabled
An RSA or RSA-HSM key within the Key Vault: The key must not be expired, it must be enabled, and it must have both wrap and unwrap capabilities enabled

Once these resources are configured, two steps remain to allow Azure App Configuration to use the Key Vault key:

Assign a managed identity to the Azure App Configuration instance
Grant the identity GET, WRAP, and UNWRAP permissions in the target Key Vault's access policy.

Securing App Configuration Data

Private endpoints

01

Secure your application configuration details by configuring the firewall to block all connections to App Configuration

02

Increase security for the virtual network (VNet) ensuring data doesn't escape from the VNet

03

Securely connect to the App Configuration store from on-premises networks

Private endpoints:
You can use private endpoints for Azure App Configuration to allow clients on a virtual network (VNet) to securely access data over a private link. The private endpoint uses an IP address from the VNet address space for your App Configuration store.
When creating a private endpoint, you must specify the App Configuration store to which it connects.
When you create a private endpoint, the DNS CNAME resource record for the configuration store is updated to an alias in a subdomain with the prefix privatelink.

Securing App Configuration Data

Managed identities

01

A system-assigned identity is tied to your configuration store

02

A user-assigned identity is a standalone Azure resource that can be assigned to your configuration store

Private endpoints:
You can use private endpoints for Azure App Configuration to allow clients on a virtual network (VNet) to securely access data over a private link. The private endpoint uses an IP address from the VNet address space for your App Configuration store.
When creating a private endpoint, you must specify the App Configuration store to which it connects.
When you create a private endpoint, the DNS CNAME resource record for the configuration store is updated to an alias in a subdomain with the prefix privatelink.

KodeKloud

Visit www.kodekloud.com to learn more.