Challenge #2

The Solution will be in .Net Language in AWS Cloud.

**Solution**

**Summary**

We need to write code that will query the meta data of an instance within AWS and provide a json formatted output. The choice of language and implementation is up to you.

* Our Aim:
* Query the metadata of an ec2 instance within AWS and provide a json formatted output.
* Retrieve the value of a particular data key.

What to Consider:

EC2 Instance Metadata

If this class is used on a non-EC2 instance, the properties in this class will return Null

Amazon EC2 instances can access instance-specific metadata, as well as data supplied when launching the instances, using a specific URI.

You can use this data to build more generic AMIs that can be modified by configuration files supplied at launch time.

For example, if you run web servers for various small businesses, they can all use the same AMI and retrieve their content from the Amazon S3 bucket you specify at launch. To add a new customer at any time, simply create a bucket for the customer, add their content, and launch your AMI.

More information about EC2 Metadata <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AESDG-chapter-instancedata.html>

* [Create an EC2 Linux instance on AWS](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html)
* [SSH into the instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html)

Coding:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Threading;

using Amazon.Runtime;

using ThirdParty.Json.LitJson;

using System.Globalization;

using Amazon.Runtime.Internal.Util;

using Amazon.Util.Internal;

using Amazon.Util;

namespace Amazon.EC2.Util

[Obsolete ("This class is deprecated and will be removed in a future release."

+ " Please update your code to use the Amazon.Util.EC2InstanceMetadata class, located in the AWSSDK.Core assembly.")]

public static class EC2Metadata

{

private static string

EC2\_METADATA\_SVC = "http://169.254.169.254",

EC2\_METADATA\_ROOT = EC2\_METADATA\_SVC + "/latest/meta-data",

EC2\_USERDATA\_ROOT = EC2\_METADATA\_SVC + "/latest/user-data/",

EC2\_APITOKEN\_URL = EC2\_METADATA\_SVC + "latest/api/token";

private static int

DEFAULT\_RETRIES = 3,

MIN\_PAUSE\_MS = 250,

MAX\_RETRIES = 3;

private static Dictionary<string, string> \_cache = new Dictionary<string, string>();

private static readonly string \_userAgent = InternalSDKUtils.BuildUserAgentString(string.Empty);

The AMI ID used to launch the instance:

public static string AmiId

{

get { return FetchData("/ami-id"); }

}

/// <summary>

/// The index of this instance in the reservation.

/// </summary>

public static string AmiLaunchIndex

{

get { return FetchData("/ami-launch-index"); }

}

/// <summary>

/// The manifest path of the AMI with which the instance was launched.

/// </summary>

public static string AmiManifestPath

{

get { return FetchData("/ami-manifest-path"); }

}

/// <summary>

/// The AMI IDs of any instances that were rebundled to create this AMI.

/// Will only exist if the AMI manifest file contained an ancestor-amis key.

/// </summary>

public static IEnumerable<string> AncestorAmiIds

{

get { return GetItems("/ancestor-ami-ids"); }

}

/// <summary>

/// The private hostname of the instance.

/// In cases where multiple network interfaces are present,

/// this refers to the eth0 device (the device for which the device number is 0).

/// </summary>

public static string Hostname

{

get { return FetchData("/hostname"); }

}

/// <summary>

/// Notifies the instance that it should reboot in preparation for bundling.

/// Valid values: none | shutdown | bundle-pending.

/// </summary>

public static string InstanceAction

{

get { return FetchData("/instance-action"); }

}

/// <summary>

/// The ID of this instance.

/// </summary>

public static string InstanceId

{

get { return FetchData("/instance-id"); }

}

/// <summary>

/// The type of instance.

/// </summary>

public static string InstanceType

{

get { return FetchData("/instance-type"); }

}

/// <summary>

/// The ID of the kernel launched with this instance, if applicable.

/// </summary>

public static string KernelId

{

get { return GetData("kernel-id"); }

}

/// <summary>

/// The local hostname of the instance. In cases where multiple network interfaces are present,

/// this refers to the eth0 device (the device for which device-number is 0).

/// </summary>

public static string LocalHostname

{

get { return FetchData("/local-hostname"); }

}

/// <summary>

/// The instance's MAC address. In cases where multiple network interfaces are present,

/// this refers to the eth0 device (the device for which device-number is 0).

/// </summary>

public static string MacAddress

{

get { return FetchData("/mac"); }

}

/// <summary>

/// The private IP address of the instance. In cases where multiple network interfaces are present,

/// this refers to the eth0 device (the device for which device-number is 0).

/// </summary>

public static string PrivateIpAddress

{

get { return FetchData("/local-ipv4"); }

}

/// <summary>

/// The Availability Zone in which the instance launched.

/// </summary>

public static string AvailabilityZone

{

get { return FetchData("/placement/availability-zone"); }

}

/// <summary>

/// Product codes associated with the instance, if any.

/// </summary>

public static IEnumerable<string> ProductCodes

{

get { return GetItems("/product-codes"); }

}

/// <summary>

/// Public key. Only available if supplied at instance launch time.

/// </summary>

public static string PublicKey

{

get { return FetchData("/public-keys/0/openssh-key"); }

}

/// <summary>

/// The ID of the RAM disk specified at launch time, if applicable.

/// </summary>

public static string RamdiskId

{

get { return FetchData("/ramdisk-id"); }

}

/// <summary>

/// ID of the reservation.

/// </summary>

public static string ReservationId

{

get { return FetchData("/reservation-id"); }

}

/// <summary>

/// The names of the security groups applied to the instance.

/// </summary>

public static IEnumerable<string> SecurityGroups

{

get { return GetItems("/security-groups"); }

}

/// <summary>

/// Returns information about the last time the instance profile was updated,

/// including the instance's LastUpdated date, InstanceProfileArn, and InstanceProfileId.

/// </summary>

public static IAMInfo IAMInstanceProfileInfo

{

get

{

var json = GetData("/iam/info");

if (null == json)

return null;

IAMInfo info;

try

{

info = JsonMapper.ToObject<IAMInfo>(json);

}

catch

{

info = new IAMInfo { Code = "Failed", Message = "Could not parse response from metadata service." };

}

return info;

}

}

/// <summary>

/// Returns the temporary security credentials (AccessKeyId, SecretAccessKey, SessionToken, and Expiration)

/// associated with the IAM roles on the instance.

/// </summary>

public static IDictionary<string, IAMSecurityCredential> IAMSecurityCredentials

{

get

{

var list = GetItems("/iam/security-credentials");

if (list == null)

return null;

var creds = new Dictionary<string, IAMSecurityCredential>();

foreach (var item in list)

{

var json = GetData("/iam/security-credentials/" + item);

try

{

var cred = JsonMapper.ToObject<IAMSecurityCredential>(json);

creds[item] = cred;

}

catch

{

creds[item] = new IAMSecurityCredential { Code = "Failed", Message = "Could not parse response from metadata service." };

}

}

return creds;

}

}

/// <summary>

/// The virtual devices associated with the ami, root, ebs, and swap.

/// </summary>

public static IDictionary<string, string> BlockDeviceMapping

{

get

{

var keys = GetItems("/block-device-mapping");

if (keys == null)

return null;

var mapping = new Dictionary<string, string>();

foreach (var key in keys)

{

mapping[key] = GetData("/block-device-mapping/" + key);

}

return mapping;

}

}

/// <summary>

/// The network interfaces on the instance.

/// </summary>

public static IEnumerable<NetworkInterface> NetworkInterfaces

{

get

{

var macs = GetItems("/network/interfaces/macs/");

if (macs == null)

return null;

var interfaces = new List<NetworkInterface>();

foreach (var mac in macs)

{

interfaces.Add(new NetworkInterface(mac.Trim('/')));

}

return interfaces;

}

}

/// <summary>

/// The metadata sent to the instance.

/// </summary>

public static string UserData

{

get

{

return GetData(EC2\_USERDATA\_ROOT);

}

}

/// <summary>

/// Return the list of items in the metadata at path.

/// </summary>

/// <param name="path">Path at which to query the metadata</param>

/// <returns>List of items returned by the metadata service</returns>

public static IEnumerable<string> GetItems(string path)

{

return GetItems(path, DEFAULT\_RETRIES, false);

}

/// <summary>

/// Return the metadata at the path

/// </summary>

/// <param name="path">Path at which to query the metadata</param>

/// <returns>Data returned by the metadata service</returns>

public static string GetData(string path)

{

return GetData(path, DEFAULT\_RETRIES);

}

/// <summary>

/// Return the metadata at the path

/// </summary>

/// <param name="path">Path at which to query the metadata</param>

/// <param name="tries">Number of attempts to make</param>

/// <returns>Data returned by the metadata service</returns>

public static string GetData(string path, int tries)

{

var items = GetItems(path, tries, true);

if (items != null && items.Count > 0)

return items[0];

return null;

}

/// <summary>

/// Return the list of items in the metadata at path.

/// </summary>

/// <param name="path">Path at which to query the metadata</param>

/// <param name="tries">Number of attempts to make</param>

/// <returns>List of items returned by the metadata service</returns>

public static IEnumerable<string> GetItems(string path, int tries)

{

return GetItems(path, tries, false);

}

private static string FetchData(string path)

{

return FetchData(path, false);

}

private static string FetchData(string path, bool force)

{

try

{

if (force || !\_cache.ContainsKey(path))

\_cache[path] = GetData(path);

return \_cache[path];

}

catch

{

return null;

}

}

private static List<string> GetItems(string path, int tries, bool slurp)

{

return GetItems(path, tries, slurp, null);

}

private static List<string> GetItems(string path, int tries, bool slurp, string token)

{

var items = new List<string>();

//For all meta-data queries we need to fetch an api token to use. In the event a

//token cannot be obtained we will fallback to not using a token.

Dictionary<string, string> headers = null;

if (token == null)

{

token = Amazon.Util.EC2InstanceMetadata.FetchApiToken();

}

if (!string.IsNullOrEmpty(token))

{

headers = new Dictionary<string, string>();

headers.Add(HeaderKeys.XAwsEc2MetadataToken, token);

}

try

{

if (!Amazon.Util.EC2InstanceMetadata.IsIMDSEnabled)

{

throw new IMDSDisabledException();

}

HttpWebRequest request;

if (path.StartsWith("http", StringComparison.Ordinal))

request = WebRequest.Create(path) as HttpWebRequest;

else

request = WebRequest.Create(EC2\_METADATA\_ROOT + path) as HttpWebRequest;

request.Timeout = (int)TimeSpan.FromSeconds(5).TotalMilliseconds;

request.UserAgent = \_userAgent;

if(headers != null)

{

foreach(var header in headers)

{

request.Headers.Add(header.Key, header.Value);

}

}

using (var response = request.GetResponse())

{

using (var stream = new StreamReader(response.GetResponseStream()))

{

if (slurp)

items.Add(stream.ReadToEnd());

else

{

string line;

do

{

line = stream.ReadLine();

if (line != null)

items.Add(line.Trim());

}

while (line != null);

}

}

}

}

catch (WebException wex)

{

var response = wex.Response as HttpWebResponse;

if (response != null)

{

if (response.StatusCode == HttpStatusCode.NotFound)

{

return null;

}

else if (response.StatusCode == HttpStatusCode.Unauthorized)

{

EC2InstanceMetadata.ClearTokenFlag();

Logger.GetLogger(typeof(Amazon.EC2.Util.EC2Metadata)).Error(wex, "EC2 Metadata service returned unauthorized for token based secure data flow.");

throw;

}

}

if (tries <= 1)

{

Logger.GetLogger(typeof(Amazon.EC2.Util.EC2Metadata)).Error(wex, "Unable to contact EC2 Metadata service.");

return null;

}

PauseExponentially(tries);

return GetItems(path, tries - 1, slurp, token);

}

catch (IMDSDisabledException)

{

// Keep this behavior identical to when HttpStatusCode.NotFound is returned.

return null;

}

return items;

}

private static void PauseExponentially(int tries)

{

tries = Math.Min(tries, MAX\_RETRIES);

var pause = (int)(Math.Pow(2, DEFAULT\_RETRIES - tries) \* MIN\_PAUSE\_MS);

Thread.Sleep(pause < MIN\_PAUSE\_MS ? MIN\_PAUSE\_MS : pause);

}

#if !NETSTANDARD

[Serializable]

#endif

private class IMDSDisabledException : InvalidOperationException { };

}

/// <summary>

/// Returns information about the last time the instance profile was updated,

/// including the instance's LastUpdated date, InstanceProfileArn, and InstanceProfileId.

/// </summary>

[Obsolete("This class is deprecated and will be removed in a future release."

+ " Please update your code to use the Amazon.Util.IAMInstanceProfileMetadata class, located in the AWSSDK.Core assembly.")]

public class IAMInfo

{

/// <summary>

/// The status of the instance profile

/// </summary>

public string Code { get; set; }

/// <summary>

/// Further information about the status of the instance profile

/// </summary>

public string Message { get; set; }

/// <summary>

/// The date and time the instance profile was updated

/// </summary>

public DateTime LastUpdated { get; set; }

/// <summary>

/// The Amazon Resource Name (ARN) of the instance profile

/// </summary>

public string InstanceProfileArn { get; set; }

/// <summary>

/// The Id of the instance profile

/// </summary>

public string InstanceProfileId { get; set; }

}

/// <summary>

/// The temporary security credentials (AccessKeyId, SecretAccessKey, SessionToken, and Expiration) associated with the IAM role.

/// </summary>

[Obsolete("This class is deprecated and will be removed in a future release."

+ " Please update your code to use the Amazon.Util.IAMSecurityCredentialMetadata class, located in the AWSSDK.Core assembly.")]

public class IAMSecurityCredential

{

/// <summary>

/// The status of the security credential

/// </summary>

public string Code { get; set; }

/// <summary>

/// Further information about the status of the instance profile

/// </summary>

public string Message { get; set; }

/// <summary>

/// The date and time the security credential was last updated

/// </summary>

public DateTime LastUpdated { get; set; }

/// <summary>

/// The type of the security credential

/// </summary>

public string Type { get; set; }

/// <summary>

/// The uniqe id of the security credential

/// </summary>

public string AccessKeyId { get; set; }

/// <summary>

/// The secret key used to sign requests

/// </summary>

public string SecretAccessKey { get; set; }

/// <summary>

/// The security token

/// </summary>

public string Token { get; set; }

/// <summary>

/// The date and time when these credentials expire

/// </summary>

public DateTime Expiration { get; set; }

}

/// <summary>

/// All of the metadata associated with a network interface on the instance.

/// </summary>

[Obsolete("This class is deprecated and will be removed in a future release."

+ " Please update your code to use the Amazon.Util.NetworkInterfaceMetadata class, located in the AWSSDK.Core assembly.")]

public class NetworkInterface

{

private string \_path;

private string \_mac;

private IEnumerable<string> \_availableKeys;

private Dictionary<string, string> \_data = new Dictionary<string, string>();

private NetworkInterface() { }

/// <summary>

/// Construct an instance of NetworkInterface

/// </summary>

/// <param name="macAddress"></param>

public NetworkInterface(string macAddress)

{

\_mac = macAddress;

\_path = string.Format(CultureInfo.InvariantCulture, "/network/interfaces/macs/{0}/", \_mac);

}

/// <summary>

/// The interface's Media Access Control (mac) address.

/// </summary>

public string MacAddress

{

get { return \_mac; }

}

/// <summary>

/// The ID of the owner of the network interface.

/// </summary>

/// <remarks>

/// In multiple-interface environments, an interface can be attached by a third party, such as Elastic Load Balancing.

/// Traffic on an interface is always billed to the interface owner.

/// </remarks>

public string OwnerId

{

get { return GetData("owner-id"); }

}

/// <summary>

/// The interface's profile

/// </summary>

public string Profile

{

get { return GetData("profile"); }

}

/// <summary>

/// The interface's local hostname.

/// </summary>

public string LocalHostname

{

get { return GetData("local-hostname"); }

}

/// <summary>

/// The private IP addresses associated with the interface.

/// </summary>

public IEnumerable<string> LocalIPv4s

{

get { return GetItems("local-ipv4s"); }

}

/// <summary>

/// The interface's public hostname.

/// </summary>

public string PublicHostname

{

get { return GetData("public-hostname"); }

}

/// <summary>

/// The elastic IP addresses associated with the interface.

/// </summary>

/// <remarks>

/// There may be multiple IP addresses on an instance.

/// </remarks>

public IEnumerable<string> PublicIPv4s

{

get { return GetItems("public-ipv4s"); }

}

/// <summary>

/// Security groups to which the network interface belongs.

/// </summary>

public IEnumerable<string> SecurityGroups

{

get { return GetItems("security-groups"); }

}

/// <summary>

/// IDs of the security groups to which the network interface belongs. Returned only for Amazon EC2 instances launched into a VPC.

/// </summary>

public IEnumerable<string> SecurityGroupIds

{

get { return GetItems("security-group-ids"); }

}

/// <summary>

/// The ID of the Amazon EC2-VPC subnet in which the interface resides.

/// </summary>

/// <remarks>

/// Returned only for Amazon EC2 instances launched into a VPC.

/// </remarks>

public string SubnetId

{

get { return GetData("subnet-id"); }

}

/// <summary>

/// The CIDR block of the Amazon EC2-VPC subnet in which the interface resides.

/// </summary>

/// <remarks>

/// Returned only for Amazon EC2 instances launched into a VPC.

/// </remarks>

public string SubnetIPv4CidrBlock

{

get { return GetData("subnet-ipv4-cidr-block"); }

}

/// <summary>

/// The CIDR block of the Amazon EC2-VPC subnet in which the interface resides.

/// </summary>

/// <remarks>

/// Returned only for Amazon EC2 instances launched into a VPC.

/// </remarks>

public string VpcId

{

get { return GetData("vpc-id"); }

}

/// <summary>

/// Get the private IPv4 address(es) that are associated with the public-ip address and assigned to that interface.

/// </summary>

/// <param name="publicIp">The public IP address</param>

/// <returns>Private IPv4 address(es) associated with the public IP address</returns>

public IEnumerable<string> GetIpV4Association(string publicIp)

{

return EC2Metadata.GetItems(string.Format(CultureInfo.InvariantCulture, "{0}ipv4-associations/{1}", \_path, publicIp));

}

private string GetData(string key)

{

if (\_data.ContainsKey(key))

return \_data[key];

// Since the keys are variable, cache a list of which ones are available

// to prevent unnecessary trips to the service.

if (null == \_availableKeys)

\_availableKeys = EC2Metadata.GetItems(\_path);

if (\_availableKeys.Contains(key))

{

\_data[key] = EC2Metadata.GetData(\_path + key);

return \_data[key];

}

else

return null;

}

private IEnumerable<string> GetItems(string key)

{

if (null == \_availableKeys)

\_availableKeys = EC2Metadata.GetItems(\_path);

if (\_availableKeys.Contains(key))

{

return EC2Metadata.GetItems(\_path + key);

}

else

return new List<string>();

}

}

}