FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Security aspects of software development in the Microsoft Azure cloud
Student:	David Mládek
Supervisor:	Ing. Tomáš Pajurek
Study Programme:	Informatics
Study Branch:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2019/20

Instructions

Examine security aspects of software development on the Microsoft Azure platform. Identify and review important services that the developers are exposed to. Focus on the interaction of developers with these services, point out security-critical areas (such as managing access tokens or other secrets). Next, propose Azure-specific solutions for the identified challenges.
Services on the infrastructure-as-a-service level (IaaS) that are not usually managed by developers but rather by IT admins, such as virtual networks or virtual machines, are out-of-scope. Proposed solutions should be usable for small and medium-size teams.
Essential solutions and approaches that are discussed must be accompanied with example source codes in C# on .NET Core platform. Relevant parts of the thesis should be also accompanied with example deployments in Azure (existing, physically deployed resources as well as in an infrastructure-as-a-code form).

References

- Michael Howard and David E. Leblanc: 2002. Writing Secure Code (2nd ed.). Microsoft Press, Redmond, WA, USA
- Michael Howard and Steve Lipner:2006. The Security Development Lifecycle. Microsoft Press, Redmond, WA, USA
- Ritesh Modi: 2017. Azure for Architects. Packt Publishing
- Steve McConnell: 2004. Code Complete, Second Edition. Microsoft Press, Redmond, WA, USA
- Joachim Hafner: 2018. Azure strategy and implementation guide (2nd ed.). Microsoft Corporation, Redmond, WA, USA
- Security best practices for Azure solutions. Microsoft Corporation, Redmond, WA, USA

prof. Ing. Pavel Tvrdík, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 28, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Security Aspects of Software Development in the Microsoft Azure Cloud

David Mládek

Department of Computer Systems
Supervisor: Ing. Tomáš Pajurek

May 15, 2019

Acknowledgements

First and foremost, I would like to thank my supervisor, Ing. Tomáš Pajurek, for all the time he invested in me and this work. I cannot imagine myself finishing this work on time, were it not for the frequent consultations and advice.

Furthermore, I would like to thank everyone at Datamole for giving me opportunities to gain experience in an incredible environment, and for their positive mindset and continuous support. They often helped me to relax and to put everything I do in the right perspective.

That is for the most part also the case for all my friends and family, who put up with my attitude when I started acting up due to stress from different sources. They supported me immensely, often without even knowing. Special thanks goes to my brother who proofread this whole thesis without any technical background. Because of that, the English here is incomparably better than it would have been.

Last but not least, I would like to extend my appreciation to everyone at the faculty for the time they spend on educating the next generations. Especially to those who continue even in their free time and are easy to approach at any time with any issue.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 15, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Mládek, David. *Security Aspects of Software Development in the Microsoft Azure Cloud*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Tato práce představí aspekty vývoje v cloudu, které jsou zajímavé z hlediska bezpečnosti. Konkrétně budou ukázány na platformě Microsoft Azure.

Některé služby provozované na Azure budou představeny se zaměřením na jejich bezpečnostní možnosti. Mimo jiné to budou způsoby dlouhodobého ukládání dat a autentizační služba Azure Active Directory.

V druhé polovině bude představeno několik problémů, které se mohou vyskytnout během vývoje cloudových služeb. Mezi ně budou patřit zejména operace se soukromými daty a ochrana přístupových bodů k internetu. Tyto problémy budou podrobně prozkoumány a poté budou poskytnuta řešení nezávislá na platformě i taková, která využívají některých dalších služeb provozovaných Microsoftem na Azure.

Klíčová slova cloud, Azure, bezpečnost, vývoj, aplikační tajemství, autentikace, Azure Active Directory

Abstract

This work reviews security-related aspects of cloud development. The Microsoft Azure platform will be used for demonstration of these aspects.

To that end, some services from Azure were investigated for the security measures they provide. This includes storage and identity services provided by Microsoft, as well as their respective security features.

In the second half, several problematic areas of cloud development will be introduced. Those will be mainly handling confidential data and application secrets, and properly securing Internet-facing endpoints in the cloud. They will be examined closely and the thesis will provide platform-agnostic solutions as well as solutions utilizing some services provided by Azure.

Keywords cloud, Azure, security, development, application secrets, authentication, Azure Active Directory

Contents

List of Figures

Introduction

Writing secure code is often a complicated task and in many projects security is overlooked or worked on as an afterthought. Functional requirements are often valued more than the non-functional ones, such as security. This is true for on-premises systems as well as cloud applications.

The development process and responsibilities have also shifted significantly in recent years. Security teams are not the only accountable party anymore because some of the workload has shifted more towards individual developers. With the rise of DevOps, security competence and responsibilities have also moved more towards the developer teams and although they do not need to be experts by any means, they do need to have a general idea about the implications for security when they use any pattern, service or library.

First, the Microsoft Azure platform, as well as generic cloud environment, will be introduced to the reader. Some principles that differ from on-premise systems will be shown. An important concept introduced there will be the separation of responsibilities in cloud. This concept shows which party holds what responsibilities in terms of security, based on the cloud model used (see Section 1.1.1).

Selected services offered by Microsoft on Azure will be overviewed in terms of usage and also in terms of security features they provide. Some of the security features are not enabled by default, and the impacts of using them will be described. The features and services also may have some limitations that are not apparent from the official documentation. These restrictions will be explained where necessary. The list of the Azure services will not be comprehensive and only services notable for their security aspects will be shown alongside some that are widely used and that would be useful in examples. All of those should be well known to all developers creating production systems on Azure.

Next, software development in the cloud environment will be examined and standard practices described. This will include not only the development life cycle, but also some security processes to be taken into consideration during

software creation. The DevOps approach will also be briefly described as it is prevalent in today's cloud-based software creation.

Finally, security-critical areas that developers are frequently and regularly exposed to will be examined, and their solutions will be provided. The problems will be briefly introduced in Chapter 3, and the respective solutions will be shown in Chapter 4. Sections inside these two chapters may not correspond to each other exactly because the problems may be defined in a more general manner or some parts of them may be solved separately. These problems will include secure handling of sensitive data and secrets, security of Internet-facing endpoints, and authentication.

The goal of this thesis is to point out security-related aspects of development on the Microsoft Azure cloud. It will not go over every security aspect of the software, but it will focus more or areas that may be different from on-premise systems. Some topics (such as Structured Query Language (SQL) injections) are omitted on purpose because they are already examined in depth by other works.

Chapter 1

Microsoft Azure Overview

This chapter will briefly explain what cloud computing is and then introduce Microsoft Azure. Separation of security concerns between the cloud provider and client based on cloud service type will also be shown. Different options to control and create Azure resources and services will be introduced, and then several security-related resources will be overviewed. Only resources that are important for security or that are often used by developers will be mentioned.

1.1 Introduction to Cloud

Cloud computing has been defined as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction." [1] A much simpler description is that the cloud is a set of interconnected computing resources located in one or more data centers [2].

It can be said that cloud computing is a way of obtaining virtual resources, usually without giving the users any access or deeper knowledge of their physical location or form. They can control the resources to some extent through configuration, but cannot change the underlying resources on their own, e. g., cannot reinstall an operating system on a physical machine.

One of the main differences from on-premises systems is also that the customers will have to share the resources with each other and an application may be moved between different physical machines, which must be addressed during development as resilience to unforeseen shutdowns, restarts, IP address changes, etc.

1.1.1 Cloud Service Types

There are three service models for resources on the cloud. From the lowest level of abstraction upwards, they are Infrastructure as a Service (IaaS), Platform

as a Service (PaaS) and Software as a Service (SaaS) [1]. The distinction is slowly being eroded with the maturing of cloud [3].

Infrastructure as a Service (IaaS) provides the customer with fundamental computing resources on which they can run arbitrary software including applications and operating systems. This type of service is usually provided for low-level components such as storage, virtual networks and data processing. Virtual machines are probably one of the most prominent offerings on this level. The user cannot usually change the configuration of the underlying hardware itself.

Platform as a Service (PaaS) gives the customer access to some kind of the runtime environment, in which applications written in a plethora of different programming languages and frameworks can be run. The user is not able to make changes to the underlying infrastructure but has total control over the application and usually to some extent over the hosting environment through configuration settings.

Software as a Service (SaaS) gives the customer access to the application only and nothing more. It is the product of cloud software development sold to end users. The users cannot change or communicate with the underlying infrastructure in any way [1].

1.2 Azure Overview

The Microsoft Azure[1] cloud service is a technology platform providing on-demand cloud-based computing. It offers a vast array of services from computing devices, such as virtual machines, through storage and virtual networks to web application runtime environments and serverless computing platforms.

Microsoft Azure provides all three levels of services (IaaS, PaaS and SaaS), and supports both Linux and Windows operating systems on virtual machines with popular third-party software, a number of open-source and proprietary SQL and NoSQL databases, several runtimes, container-oriented services, tools for continuous integration, and much more [4].

Microsoft Azure has data centers all over the world in over 40 locations with more announced.[2]

1.2.1 Interaction with Azure Resources

There are four main ways to interact with Azure: Azure portal, PowerShell, Azure CLI and Azure Representational State Transfer (REST) API. The portal and the REST API are the richest in features and both PowerShell and Azure CLI use the REST API internally to communicate with Azure [4].

[1] Formerly known as Windows Azure.

[2] Complete list of locations of Microsoft Azure data centers can be found at `https://azure.microsoft.com/en-us/global-infrastructure/locations/`

Azure Portal

The Azure portal provides a simple way to perform operations that do not need to be automated, such as provisioning of a single virtual machine. Most deployment operations and management processes can be done this way as well as setting up security policies, management of costs and interaction with Microsoft support. The portal is not sufficient if the customer needs to automate deployments or create large quantities of resources at once.[3]

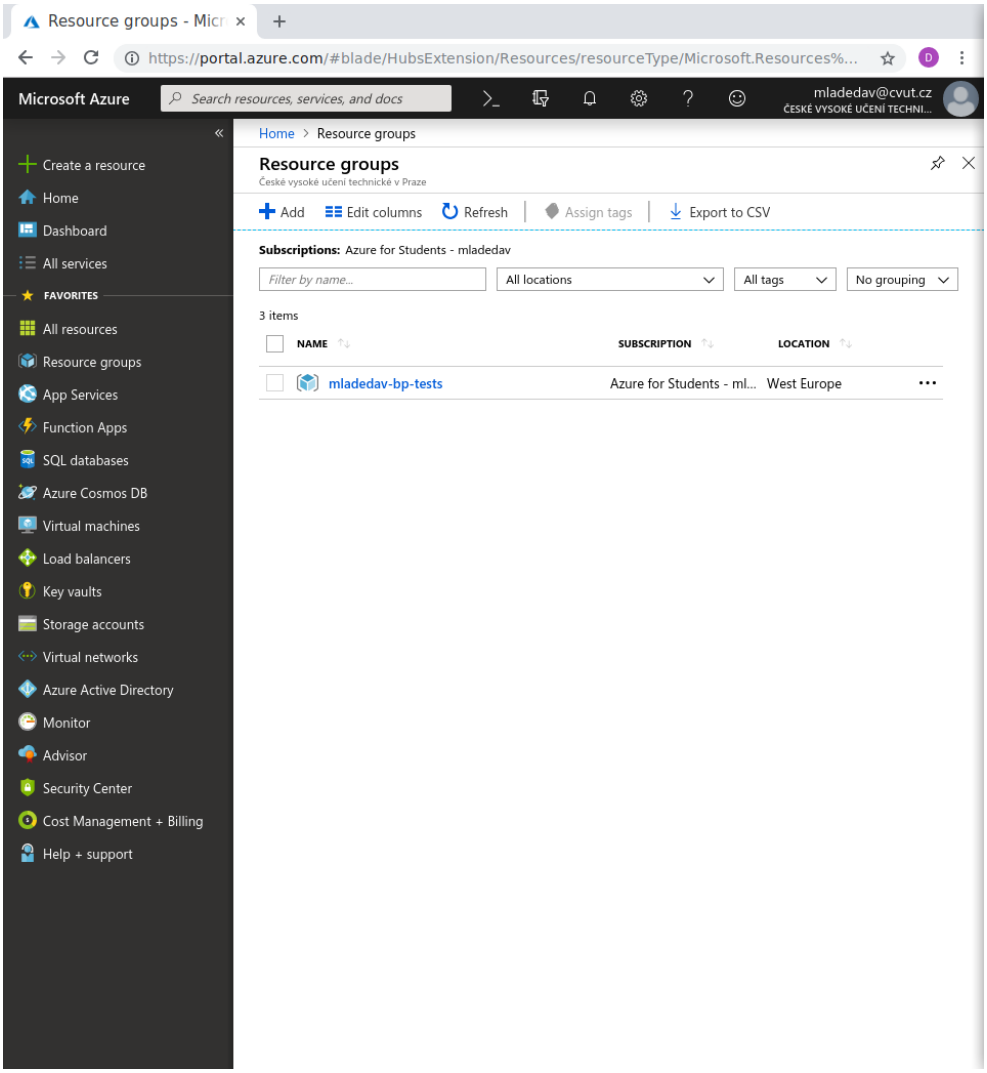

Figure 1.1: Example of Azure portal UI [5]

[3]Provisioning of a large batch of resources at once can be achieved by the usage of Azure Resource Manager Templates even through the portal.

PowerShell

PowerShell [6] is an interactive shell created by Microsoft. Its integration with Azure [7] is done via a module offering many cmdlets[4] for operations with Azure Resource Manager (ARM). The Azure module was written as an open-source project in .NET Standard,[5] meaning it can be run in PowerShell Core which in turn is executable not only on Windows but also on Linux and macOS. This has not always been the case and the module used to be available only on Microsoft Windows. It internally uses the Azure REST API for communication with the cloud.

PowerShell Input

```
1 > Connect-AzAccount # Connect to Azure with a browser sign in token
2 > Get-AzResourceGroup # List all my resource groups in the current
    subscription
```

Output

```
1 WARNING: To sign in, use a web browser to open the page \url{https://
    microsoft.com/devicelogin} and enter the code CHLD3WR3C to
    authenticate.
2
3 ResourceGroupName : mladedav-bp-tests
4 Location          : westeurope
5 ProvisioningState : Succeeded
6 Tags              :
7 ResourceId        : /subscriptions/c02c846b-e799-4497-995d-184229394101/
    resourceGroups/mladedav-bp-tests
```

Figure 1.2: Example of PowerShell usage with Azure

Azure CLI

Azure CLI has capabilities similar to those of the PowerShell module and also utilizes REST API. It can be run on Windows, Linux, and macOS, which makes it a popular choice especially for users with the need for automation of resource management using a shell other than PowerShell. Azure CLI is open-source and written in Python.[6]

[4] Lightweight PowerShell script that performs a single function. [8]
[5] Its repository can be found at `https://github.com/Azure/azure-powershell`
[6] Its repository can be found on GitHub at `https://github.com/Azure/azure-cli`.

Bash Input

```
$ az login
$ az group list
```

Output

```
Note, we have launched a browser for you to login. For old experience
    with device code, use "az login --use-device-code"
You have logged in. Now let us find all the subscriptions to which you
     have access...
[
  {
    "cloudName": "AzureCloud",
    "id": "c02c846b-e799-4497-995d-184229394101",
    "isDefault": true,
    "name": "Azure for Students - mladedav",
    "state": "Enabled",
    "tenantId": "f345c406-5268-43b0-b19f-5862fa6833f8",
    "user": {
      "name": "mladedav@cvut.cz",
      "type": "user"
    }
  }
]
[
  {
    "id": "/subscriptions/c02c846b-e799-4497-995d-184229394101/
        resourceGroups/mladedav-bp-tests",
    "location": "westeurope",
    "managedBy": null,
    "name": "mladedav-bp-tests",
    "properties": {
      "provisioningState": "Succeeded"
    },
    "tags": null
  }
]
```

Figure 1.3: Example of usage Azure CLI in Bash

Azure REST API

Azure REST API [9] provides Create, Retrieve, Update, Delete (CRUD) access to Azure resources via a well-known Hypertext Transfer Protocol (HTTP) endpoint.[7] The operations are protected by Azure Active Directory and an Authorization header must be set to a bearer token received from Azure Active Directory (AAD) for any HTTP request to be accepted.

Several languages and environments, such as .NET Core, Java, Node.js or Python, have a Software Development Kit (SDK) for Azure that provides programmatical access to the Azure REST API for operations on some Azure resources.

1.2.2 Separation of Responsibilities

The following paragraphs describe which party is responsible for which part of security in the Azure cloud. This separation is critical because the cloud service provider cannot enforce a secure code being written, same as the user cannot influence the physical security in any way.

Microsoft made a shared responsibility model, divided into seven layers from physical security to data classification and accountability and further split into four cases, on premises cloud, IaaS, PaaS and SaaS. [10]

The Cloud Service Provider (CSP) is responsible for all hardware as well as its physical security and virtualization technology so that customers cannot access other users' resources even if those are present on the same physical machine. The customer's responsibilities depend on the service, but for IaaS may include securing the operating systems, network configuration, applications, identity and data.

PaaS shifts more responsibilities to the CSP, namely patching the operating systems, runtimes, and network configurations.

As for SaaS, the cloud service customer should be completely abstracted from the underlying components and is responsible only for data classification and partly for managing the application's end users and end-point devices. [10]

In any case, an incident response procedure should be ready even for high-cost threats that would leverage the CSP's vulnerabilities, i. e., hardware vulnerabilities. There should be clear definitions and distinction between unaffected services; services affected, but not requiring any action; those that would have to be limited; and those that would have to be taken offline. This classification would of course also depend on the nature and severity of the threat.

[7] All ARM operations are directed towards `https://management.azure.com/`.

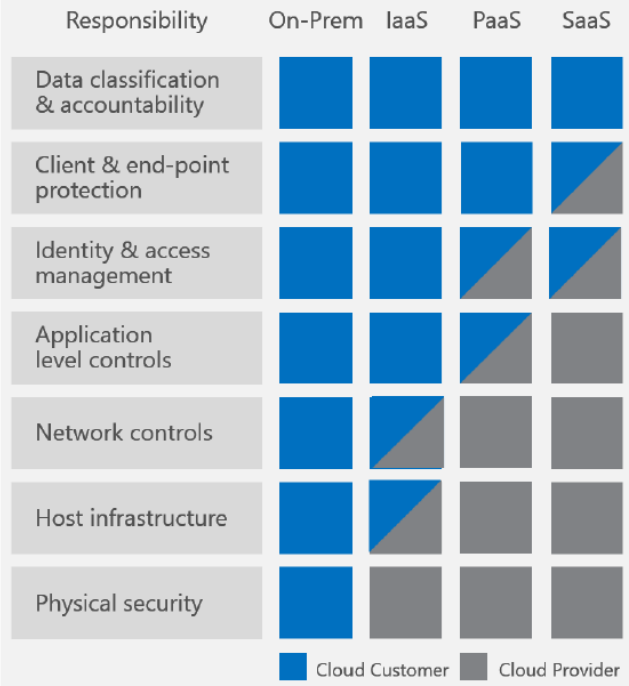

Figure 1.4: Shared responsibilities for different cloud service models [10]

1.3 Microsoft Azure Resources Related to Security

It is essential first to note what a "resource" means in the Azure context. The term refers to an entity managed by Azure, e. g., an instance of a virtual machine, virtual network, SQL database or serverless computing function [11].

Azure nowadays handles resource managagement through Azure Resource Manager, which is an approach in which resources can be grouped into resource groups for easier management of multiple resources. From the point of view of Azure Resource Manager, everything in Azure is a resource [4].

Every resource must belong to a resource group, a logical construct that lets the customer put together resources in a logical way, e. g., Azure resources for a single application [11]. Even though a resource group has a default location, its resources may be in any region.

A subscription is another logical construct, grouping together resource groups. Billing information is provided for each subscription, and each subscription has to have a single identity provider in the form of an Azure Active Directory (AD) tenant.[8] It can be useful to separate costs into different subscriptions to compare spending between larger projects or to control spending on resources in given areas, such as development.

[8] Azure AD and tenants are discussed in Section 1.3.1.

In place of the newer ARM, the old approach was Azure Service Manager (ASM). There are some implementation changes such as moving from Extensible Markup Language (XML) to JSON and also operational advancements such as making the management of resources easier by implementing Role-Based Access Control (RBAC). That means that any user or application may be assigned a role or set of roles, which in turn are authorized to access some resources. In Azure, those roles can be scoped to a single resource, to a resource group, or a subscription. The most standard roles are:

1. Owner — can manage everything about the object on which the operation is being authorized;

2. Contributor — can manage everything except access to the object;

3. Reader — can view everything but cannot make any changes.

There are usually also many more roles predefined for specific Azure resources (such as SQL DB Contributor for SQL servers) and customers can define their own roles,[9] when the built-in ones do not suffice.

Deployments were also done sequentially in ASM, which meant it could get very long for large projects. In ARM, resources that do not depend on each other can be provisioned at the same time (e. g., a managed disk and a networking interface for the same virtual machine can be created in parallel). As the new approach is much richer and some parts of the older one are not being supported anymore, and Service Management REST API is said to be retired on November 1, 2019, this work will not take ASM into account in any way besides this section.

The ARM features RBAC with Azure Active Directory authentication for access and management of Azure resources. Another level of security is added by the option of defining custom policies which can allow or restrict access to or operations on some resources based on aspects such as location or software version, even though the processes of authentication and authorization were successful [4]. These policies can be applied either on subscription or resource group scope. For example, a policy can state that resources in a specific resource group can only be created in the West Europe location. Then even if one has the Owner role in the said resource group, they cannot create resources in locations other than the one stated in the policy.

1.3.1 Azure Active Directory

One of the pivotal parts of Azure security is Azure Active Directory, which serves as the authentication service for the users of Azure. It provides identity and access management in the cloud. It offers these services also to SaaS

[9] Example tutorials can be found in Microsoft documentation [12, 13].

offerings such as Microsoft Office 365 [3] and can be used by developers to authenticate users in their own applications.

It is split into several logical units. There is Core Store for persisting all Directory Data, i. e., the identity and access data provided as well as all metadata accompanying it. Authentication services consume said data as well as user input, perform the credentials validation checks, and implement the authentication flows as well as operations on security tokens. At the same level of abstraction reside Identity Security and Protection Services which provide Multi-Factor Authentication (MFA) and other identity-driven protection for interaction with the system. Lastly, Identity Services themselves include Identity and Access Management (IAM) Services — self-service password resets and group management, dynamic group membership and other forms of user management of Directory Data [14].

Azure Active Directory comprises of multiple tenants, where an Azure tenant represents a single organization [15]. In Azure terminology, a tenant is a dedicated instance of Azure AD that you own when you sign up for a Microsoft cloud service (Azure, Office 365, etc.). Each tenant directory is isolated from the others in the service and designed to ensure user data is not accessible from other tenants, meaning others cannot access data in a specific directory unless an administrator grants explicit access [3].

A security principal is an entity that can be authenticated in Azure AD. In Azure, there are two types of security principals: user principals and service principals.

1. User principals represent a user that may be authenticated with the Azure AD, usually with an email and password and with possible MFA.

2. A service principal represents an application that may be authenticated in Azure AD with its client ID, which is an arbitrarily assigned application ID during application registration with Azure AD, and either with a password or an assertion of identity signed with a certificate that the application owns [16]. This flow is always non-interactive.

When an application is registered to Azure AD, a single application object along with a service principal is created in its home tenant. This application object then serves as the foundation for all subsequent service principals that may be created for the application in other tenants.

Each tenant that needs to authenticate an application or user needs to have their security principals. Therefore there may be multiple security principals for a single entity across multiple tenants. However, there can only be one in each tenant.

The Azure AD data is replicated for high availability and fault tolerance purposes, both inside the primary data center based on region selected during creation of tenant and in other geographical regions. Write operations can be done only to the primary replica, while reads can be done from any. Any

write operation is considered successful only upon replication of the new data to at least one geographically separated replica [17], thus ensuring consistency and fault tolerance. In case the data center with the primary AAD data fails, there are failover processes in place that change the designation of the primary replica to some other functioning replica. This process takes typically only 1–2 minutes during which read operations are still serviced and only write operations are affected [17].

The data is also secured both at rest and in transit. It is encrypted by a symmetric cipher with hash-based message authentication[10] when Directory Data is being moved between two Azure AD servers and by employing the Transport Layer Security (TLS) protocol when communicating with the users. The back end utilizes secret stores for storage of sensitive material using technology that is proprietary to Microsoft [14]. There is also a disk-level encryption employing AES-128 and Microsoft BitLocker drive encryption technology [14].

Authentication is the process of assessing the identity of an entity. This is often done to then make a decision to allow or deny access to or operation on a resource. This decision is the authorization of the agent to carry out the operation. The two processes are often very tightly coupled with authorization following authentication, but they can also be done independently, e. g., an agent can be authorized to perform an operation based only on presenting a bearer token as explained later. This agent may not have been authenticated prior to this operation in any way.

The authorization in Azure AD is based on the OAuth 2.0 [18] protocol and its extension for providing authentication, OpenID Connect 1.0 [19], for which some terminology from the standards must be introduced. When talking about OAuth 2.0, there are several roles defined by the Internet Engineering Task Force (IETF) in the proposed standard [18], which will be used in the rest of this section:

1. *Resource owner* — An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user;

2. *Resource server* — The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens;

3. *Client* — An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e. g., whether the application executes on a server, a desktop, or other devices);

[10] AES-256-CTS-HMAC-SHA1-96

4. *Authorization server* — The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

The same document [18] also specifies protocol endpoints in the following manner. The authorization process utilizes two authorization server endpoints (HTTP resources):

1. *Authorization endpoint* — used by the client to obtain authorization from the resource owner via user-agent redirection;

2. *Token endpoint* — used by the client to exchange an authorization grant for an access token, typically with client authentication.

It also introduces one client endpoint:

1. Redirection endpoint — used by the authorization server to return responses containing authorization credentials to the client via the resource owner user-agent.

Not every authorization grant type utilizes both endpoints. Extension grant types may define additional endpoints as needed [18].

There are also several different tokens to be considered:

1. *Access token* — Access tokens are pieces of data used as a proof of authorization to access some protected resources. The IETF also states: "In the general case, before a client can access a protected resource, it must first obtain an authorization grant from the resource owner and then exchange the authorization grant for an access token. The access token represents the grant's scope, duration, and other attributes granted by the authorization grant. The client accesses the protected resource by presenting the access token to the resource server." [20]

2. *Refresh token* — The IETF also defines refresh tokens: "Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope. [...] Unlike access tokens, refresh tokens are intended for use only with authorization servers and are never sent to resource servers. [...] The client may request a new access token by authenticating with the authorization server and presenting the refresh token." [18]

3. *Identity token* — Last, the authors of the OpenID Connect protocol have this to say about the identity tokens: "The primary extension that OpenID Connect makes to OAuth 2.0 to enable end-users to be authenticated is the ID token data structure. The ID token is a security

> token that contains claims[11] about the authentication of an end-user by an authorization server when using a client, and potentially other requested claims. The ID token is represented as a JWT." [19]

The advantage of using refresh tokens alongside access tokens is that access tokens, unlike refresh tokens, are short-lived and thus provide a smaller window for a potential attacker to take advantage of a leaked access token. On the other hand, when a refresh token is leaked, the attacker still needs to obtain valid client identifier and potentially password to obtain an access token. This also allows changing the scope of access tokens, because the client may request more restrictive permissions than the refresh token offers. This way it can always have only the permissions it needs at the moment.

In Azure AD, authorization is done with OAuth 2.0 access tokens[12] and users are authenticated with an extension of OAuth 2.0, the OpenID Connect protocol [19]. All of this communication happens over HTTPS to ensure confidentiality and integrity. When a service needs to be authorized to use dependency in the form of an Azure AD protected resource, it makes a call to Azure AD with one of the authorization flows as defined in the OAuth 2.0 standard [18, 22].

There are several authorization flows supported in Azure AD. For daemons and other applications that usually do not get direct interaction from users, there is the *client credential grant*, where the application authenticates with its own *client identifier* and *client secret*. The application acts under its own identity when communicating with the server. It must also be the resource owner or must have been authorized prior to access since there are no other active parties involved. The requests are directed to the token endpoint, which directly handles them and returns an access token for use with the required resource. The application may have been granted authorization to obtain the access token for a specific resource on behalf of a user, usually an administrator.

The *authorization code grant* is different in that the client application requesting access to resources redirects a browser to the authorization endpoint of Azure AD, where the user is presented with a login screen and the scope of access the client application is requesting. If they successfully authenticate against the authorization server and grants the desired authority to the client, Azure AD returns a code to the client application, which can be then exchanged for an access token and a refresh token from the token endpoint. The token endpoint may also demand a client secret from the application to

[11] "A claim is a piece of information asserted about an entity. [...] [An entity is] something that has a separate and distinct existence and that can be identified in a context. An end-user is one example of an entity." [19]

[12] The access tokens handed out by Azure AD are *bearer tokens* [20], which means that anyone in possession of those is authorized to access the resource without providing any keys and thus the tokens need to be kept secret and protected at all times including in transit. These tokens are JWT [21] that were signed by the authorization server.

ensure that the code is not misused by another party. The access token then serves as the proof of authorization to the given resource and a refresh token may be used to prolong the access if needed.

The *implicit grant* is similar, except the authorization endpoint provides the access token directly. Because this flow is intended for single-page applications (e. g., a client-side web page written in JavaScript), there is no client secret, since the application cannot protect it. After a user's successful authentication, the application is presented with an ID token and an access token. Since there is no client secret, there is no way to make guarantees that the tokens will be given only to registered clients. The application impersonates the resource owner whenever the tokens are to be used. It can then use the access token to connect to the dependencies and the ID token to acquire tokens to different resources if need be. There are no refresh tokens, but the application can request new tokens while in the same session, controlled by cookies provided by the authorization server typically via iframe.

Next, in the *resource owner password credentials grant* the client application sends a resource owner's username (or another form of ID) and password directly to the token endpoint of the authorization server along with the client's own client password. It then receives an ID token, access token and refresh token in the response. In this grant type, the client is impersonating the resource owner and acts in his stead. This flow requires a high level of trust because unlike in the others, the application requesting access handles the user's credentials directly.

Lastly, the *device grant* [23] is an extension to the OAuth 2.0 protocol. It is intended for use with a device connected to the Internet, where it is impossible to open a browser and undesirable to use the resource owner password credentials grant The proposed standard also defines a new endpoint for use with this protocol. The device communicates with this endpoint and is given back, among other things, a device code, verification URI, and a unique user code. The human operator is then asked to open a browser and navigate to the verification URI, where they are prompted to input the user code and to authenticate and consent to provide the application with the desired access. During this process the device polls the token endpoint continuously with its device code. After the user authenticates and inputs the user code that belongs with this device code, the device is given an access token and a refresh token.

The access tokens in Azure AD are in the form of a JSON Web Token [21]. The tokens consist of three parts, a header, payload and signature. Each part is Base-64 encoded, and they are delimited by a full stop. The signature algorithm[13] can be ascertained from the header parameter "alg" and the public key

[13]It is Rivest, Shamir, & Adleman (RSA) in Azure, but there are several other options including Hash-Based Message Authentication (HMAC). [24].

for validation from the header parameter "kid" and a well-known endpoint,[14] where a key with the same "kid" member can be found. The endpoint can be found on the address in the "jwks_uri" member of the OpenID Connect JSON configuration file,[15] which should be available on a standardized URI provided by the identity server (Azure AD).

1.3.2 Key Vault

One of the main security-related services to be used on a daily basis on Azure is without a doubt Azure Key Vault. It serves as safe storage for various security-critical data, such as application secrets, encryption keys and certificates. It can also serve as version control for secrets and keys, enabling updates with no downtime and running different versions of software even if it had for example configuration files encrypted with a key that has been rotated and is no longer used.

There are four types of objects in Azure Key Vault: secrets, keys, certificates, and managed storages. Secrets are arbitrary strings of data (only limited by about 25 kB in size in Key Vault inner representation). The secrets can be anything from database passwords to Base64 encoded encryption keys [25].

The keys stored in the Key Vault can be either for RSA (2048, 3072 or 4096-bit versions) or Elliptic-curve cryptography (ECC) (with one of the P-256, P-385, P-521 and SECP256K1 curves),[16] based on what cipher, security and performance are desired [25]. Key Vault also offers some key operations, such as signing payloads with a key and verifying signatures. It also supports encryption, but for symmetric ciphers only for a single block of data at the same time. The intention there is unclear, but this makes usage of modes that depend on last block's ciphered text (such as Cipher Block Chaining (CBC)) very slow, because of the roundtrip between the application and Key Vault needed for each block of data. Cryptographic keys in Key Vaults can also be optionally protected by a Federal Information Processing Standard (FIPS) 140-2 level 2 validated Hardware Security Module (HSM) produced by Thales [27] or even a FIPS 140-2 level 3 compliant HSM from Gemalto [28].

The difference between secrets is that keys serve the predefined role of a cryptographic key, while the secrets may serve any purpose. There are also some operations, such as signing, that require a Key to be present in the Key Vault.

Certificates are X.509 certificates that can be created or imported into the Key Vault. They are then managed by it, and the Key Vault may be configured to alert the owner on certain occasions such as approaching expiration, or it

[14] `https://login.microsoftonline.com/common/discovery/keys`

[15] `https://login.microsoftonline.com/common/.well-known/openid-configuration`

[16] More about elliptic curves cryptography can be found at [26].

may even automatically renew it with selected authorities. The certificates may also be stored in an HSM [25].

Finally, managed storages are references to Azure Storages, for which the Key Vault should manage the access keys. It can rotate the keys regularly or when requested and it gives out Shared Access Signature (SAS) tokens as needed. Azure Storage will be reviewed in depth in Section 1.3.3 and storages managed by a Key Vault in Section 4.2.2

To address erroneous deletion of Key Vault objects, there is also an optional soft-delete feature, that separates the delete operations from actual removal of the objects. These can be recovered in up to 90 days following the operation invocation [29].

It also provides AAD enabled authentication with either ARM RBAC for management of the Azure Key Vault resource,[17] e. g., setting up firewall rules, or Key Vault access policies for data access authorization, e. g., accessing stored secrets. Although there is RBAC[18] to the protected data, access policies can be set only on the type of a resource, such as read-only access to secrets. In that case, the authorized user can access all secrets, and there is no built-in way to give permissions only to needed secrets.

Redundancy has been addressed by the replication of the key vault content both within the region and to a second region, where both regions are in the same security world [30]. There are also logging and auditing capabilities dependent on other Azure services.

1.3.3 Azure Storage

Azure storage encompasses four basic services — Blob service, File Share service, Table service and Queue service [3]. A single resource in Azure containing these services is called an Azure storage account.

There are several variations of the storage account based on desired redundancy:

1. Locally redundant storage — data is replicated inside a single data center;

2. Zone-redundant storage — data is replicated in three different storage clusters in a single region;[19]

[17] This provides access to the management plane. It can allow for a group of users to control operational aspects of the Key Vault possibly without providing them access to the data stored there.

[18] Not the same as ARM RBAC. It still uses the same roles obtained from Azure AD, but this provides access only to the data plane, not to the management of the Key Vault itself.

[19] The clusters are physically separated, but they are still in the same region (e. g., the West Europe region)

3. Geo-redundant storage — data is replicated first locally to two additional replicas and then to a different region. Customers can read only from the data center in the primary region;

4. Read-access geo-redundant storage — data is replicated locally and then to a different region, but read operations can be performed from both data centers. There is a slight delay between finishing a write operation and the availability of data in the secondary region's data center [31].

The Blob storage is a space for any binary objects, may it be pictures, Hypertext Markup Language (HTML) or text files. The objects are accessible through URLs and the REST interface or any Azure SDK storage client library.[20]. Each Blob is placed in a container, which in turn belongs to one storage account. Containers cannot be placed into other containers and Blobs cannot be placed directly in a storage account. This relationship is shown in Figure 1.5.

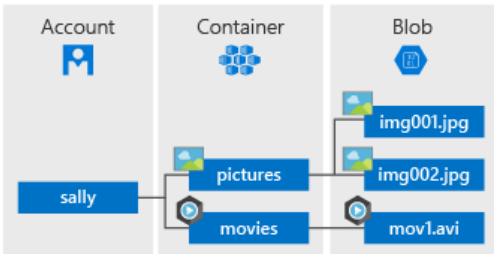

Figure 1.5: Blob storage hierarchy. [33]

There are three types of Blobs. The block Blobs are intended for files expected to be read and written sequentially (e. g., audio or text files). The page Blobs are to be accessed in a random manner with the primary use-case being a backing storage for a Virtual Hard Disk (VHD) used in Azure Virtual Machine (VM). Finally, the append Blobs, which are optimized for appending and do not support updating or deleting existing blocks of data, are primarily used for log files.

Azure Queue provides storage for short messages (up to 64 kB), usually serving as a communication means between two applications. Advanced messaging capabilities such as publish-subscribe or deduplication are not supported. The messages must be processed within seven days after being received [34].

Azure Table storage offers a depository for structured non-relational data. It can hold entities of up to 1 MB in size with up to 252 custom properties defined by the customer. NoSQL queries can be run on the storage. There

[20]Client libraries are available for a variety of languages, including .NET, Java, Node.js, Python, Go, PHP, and Ruby [32]

are also three system properties — a partition key, a row key, and a timestamp. The table may be distributed across multiple servers, but data with the same partition key are guaranteed to be on the same physical machine. The combination of partition and row keys must be unique across the table (the combination forms the primary key) [35]. It is designed for queries based on the two keys and does not scale well otherwise, especially when the partition key is omitted [36].

Azure Files offer a managed storage accessible via the Server Message Block (SMB) protocol. The storage can be mounted concurrently on Windows, Linux, and macOS, or it can be accessed programmatically. Azure Files support standard folder hierarchy [37]. Access to this service with Azure AD authentication and RBAC is being worked on at the time of writing of this thesis [38].

The data is automatically encrypted when written to Azure Storage by using Storage Service Encryption. This feature cannot be disabled, but the encryption keys can be either Microsoft managed, which includes secure storage and regular rotation, or keys provided by the user in an Azure Key Vault [39]. The cipher used is Advanced Encryption Standard (AES)-256 [40], and at the writing of this work, only Blobs and Azure Files support custom keys [41].

User data can also be encrypted in transit between an application and Azure by either client-side encryption, enforcement of HTTPS, or usage of SMB 3.0 [42]. The client-side encryption is handled with the envelope technique, where the data is encrypted by a key generated for a single use, which is then encrypted (wrapped) by an outside service, either Azure Key Vault or by a different locally managed key service and then the data gets sent with the wrapped key and some additional encryption metadata, such as the Initialization Vector (IV), to the storage service [43]. The used cipher is AES-256 with CBC mode [43]. It is important to note here that there are no safeguards against overwriting encrypted data with unencrypted data or data encrypted with different keys, potentially making the original data unreadable. Similar problems may occur when writing metadata to Blobs because metadata is not additive [43]. HTTPS can be enforced by enabling the Secure transfer required option on the storage account, which will force the service to refuse all HTTP connections to the REST API [39].

There are two kinds of storage access, one is for the management plane, the other for the data plane. Management plane controls the storage account, e. g., setting keys for Storage Service Encryption (SSE) or management of storage account keys. Data plane concerns itself only with the data, e. g., access to Blobs or tables stored in the storage account.

There are two 512-bit keys encoded in Base64 to any storage account that can be used for data operations and creation of SAS tokens for delegated access (discussed later in this subsection). These keys can be revoked and regenerated at any time by the customer. More information on the subject of Azure storage key rotation and regeneration can be found in later chapters.

Because the storage access keys can only provide full access to the whole account, they are not always an optimal means for authorization to the storage services and objects. Although they are technically easy to revoke, having to distribute new keys to every application using the storage too often is not desirable. Shared Access Signatures can be created to delegate access to chosen objects in Azure Storage with enumerated operations permissions for a specific amount of time. The access can be as granular as a single Blob or specific table entities. Usage of the SAS token can also be restricted by time, IP address range or HTTPS may be enforced with them.

When creating a SAS token, one first needs to make a string-to-sign containing all the parameters in a given order in Unicode Transformation Format — 8-bit (UTF-8) encoding. This string is then signed with HMAC-Secure Hash Algorithm (SHA)256 with one of the storage keys and then encoded in Base-64 [44, 45]. This key then can be used, along with all of the parameters that were used during the signing, in a URL when accessing the objects it is intended for.

A SAS token can also be linked to a stored access policy, which provides more control over the signatures that have been handed out. There is no way to revoke a SAS token after its issue other than revoking the key with which the token was signed or by means of stored access policies. When a SAS token is bound to a policy, it works only if the policy still exists and permits the requested access. One can thus revoke access of some SAS tokens by only changing or deleting a stored access policy without the need to change the storage keys.

1.3.4 Azure Data Lake Storage

Azure Data Lake Storage Gen1, formerly known as Azure Data Lake Store, is a storage service optimized for big data analytics implementing Hadoop Distributed File System (HDFS) [46] and its HTTP REST API called WebHDFS [47]. Therefore it can be used by many well-known tools in the Hadoop and Spark ecosystems. It can store arbitrary file types including structured (relational) as well as unstructured data. Files can be as large as petabytes in size, but it is advised for performance reasons to use sizes of 256 MB up to a suggested average size of 2 GB [48].

Data Lake Storage supports hierarchical containers, which can be thought of as folders and files. Unlike the Blob storage containers, in Data Lake Storage folders can contain more folders.

The storage is encrypted at rest, and there is local redundancy[21] in case the storage servers fail. The encryption algorithm used is AES-256. There are several keys in use with the asymmetric Master Encryption Key, that can be managed by the user, protecting the symmetric Data Encryption Key, from

[21] Local redundancy here has the same meaning as when talking about storage accounts. See the section about Azure storage accounts for more details.

which Block Encryption Keys are derived for individual data blocks. Only the Data Encryption Key is stored in Data Lake Storage with the Master Encryption Key stored in Key Vault.

This storage service also supports Portable Operating System Interface (POSIX) compliant Access Control List (ACL) permissions, which can be assigned to principals and groups in Azure AD. Identity can be verified either interactively with the end user's Azure credentials or non-interactively with the application's identity in Azure AD (service principal).

The next generation data lake service, Azure Data Lake Storage Gen2, which integrates the features of the first generation and Blob storage, providing access to some of the storage account's benefits, such as Secure Storage Encryption and SAS tokens, and also offering hierarchical namespace folders and Azure AD based ACL.

1.3.5 Azure App Service

Azure App Service is a PaaS offering that offers complete runtime environment for web applications, REST APIs or mobile back ends. The environments are handled by Azure, meaning that the developers do not need to e. g., update the operating system or manage scale-out and load balancers. A public IP address is also provided, as is an option to buy a custom domain and a Secure Sockets Layer (SSL) certificate for higher pricing tiers[22] of the service.

The base operating system can be either Windows or Linux. Supported languages, frameworks, and runtimes include ASP.NET, ASP.NET Core, Java, Ruby, Node.js, PHP, Python and Docker.

Patches to the OS are applied monthly, aligning with Patch Tuesday schedule. [49] High priority security updates are handled on a case-by-case basis. Language runtimes are updated regularly based on the specific technology's releases. Patch versions (e. g., bug fixes) update the runtime without keeping the older versions and migrating the application to the new version automatically,[23] while major or minor version updates are installed side by side and it is up to the customer to migrate the application to the new version.

All updates are run so that one region in each regional pair[24] [50] is kept online at all times. The updates are also run outside of the data center's local business hours. Machines that currently do not have running applications on them are updated first with subsequent migration of running applications

[22]There are six tiers: Free, Shared, Basic, Standard, Premium and Isolated. They differ in price, resources allocated, and additional options.

[23]With the exceptions of Node.js, where it is installed side by side, and Python, where new patch versions can be installed manually.

[24]Regional pairs are set up in a way that each datacenter in the pair is in the same geography, but still at least 300 miles apart. They are designed in a way that using both regions in a pair should minimize chances of downtime of services and accelerate their restart after catastrophic events.

to those servers and updates to the newly freed machines. This approach minimizes downtime[25] for any running applications.

Authentication of end users can be configured in App service to use Azure AD login as well as other well-known identity providers such as Google or Facebook. Developers then do not have to use a third-party library or even have to implement the communication with the OAuth services themselves. After the configuration, Azure App service handles all incoming HTTP requests and injects identity information into request headers. [51]

1.3.6 Managed Identity

One of the most essential security-related benefits of Azure Resource Manager is RBAC, which is also extended to cloud applications accessing other resources and services registered in Azure AD. In many cases, it is necessary to access an Azure resource from another, e. g., a web application in Azure App Service accessing an SQL database inside Azure. Usually, a developer would create a database user for the application and put the credentials in some kind of configuration file or a secrets storage. Managed identities make Azure AD handle the authentication, and then ARM handles the authorization to the end service or to ARM itself.

For this reason, some Azure resources can be assigned an identity that can be used for authentication purposes during communication with AAD and other resources. This feature is called Managed Identities for Azure and has been formerly known as Managed Service Identity (MSI).

This approach has been introduced so that there is no need to store passwords and other secrets in code or even have the need to access static credentials in runtime,[26] clearing up credentials from the codebase. It is important to note that this feature was announced in its original MSI form in September 2017 and so it is not available for all resources yet.[27]

There are two types of managed identities differing by assignment of the identity to services or resources and its life cycle. In both cases, a service principal is created in Azure AD and is used to authenticate the application and obtain an access token to the resource. For more information on service principals, see Section 1.3.1 about Azure Active Directory earlier in this chapter.

In the first case, it is the system-assigned managed identity, which is created in the Azure AD and enabled directly on an Azure resource that supports managed identities. In this case Azure provisions the credentials onto the managed identity resource and the identity's life cycle is directly tied to

[25] The application has to be restarted after the update, but this approach along with the fact that it is happening during the off hours minimizes the impact on end users.

[26] The application still needs the OAuth tokens, but those are created when needed with a set expiration date and time.

[27] The complete list of supported services can be found at [52].

the resource itself (i. e., it is deleted when the resource for which it provides the identity is deleted).

The other option is user-assigned managed identity, where the customer creates a 'Managed Identity' resource and then assigns it to a resource that may have other dependencies, such as a function app or a virtual machine [53].

One service can only have one system-assigned identity, but it can also have multiple user-assigned identities where it is allowed access based on under which identity the service authenticates against Azure AD.

A service can ask Azure AD through Azure Instance Metadata Service (AIMS)[28] using REST API for access tokens to some other given service under its protection, including Azure Resource Manager for operations with resources. AIMS then request an access token from Azure AD, providing the client identifier and certificate of the service principal. The call contains a service principal client identifier and a certificate of the calling service and a JSON Web Token is returned. This token is then sent with the call to the dependency [53].

[28]The endpoint is a well-known non-routable IP address `169.254.169.254` accessible only from within a VM inside Azure [53]

Chapter 2

Cloud Development

This chapter will introduce the software development life cycle and its models and then show some models optimized for cloud development. Security development life cycle will be covered next, and finally, the DevOps approach will be introduced. This chapter is necessary to understand better what the developer will be responsible for. The responsibilities, of course, depend on the Software Development Life Cycle (SDLC) model chosen as well as many other factors such as company culture.

2.1 Software Development Life Cycle

The software development life cycle is an abstraction over the phases in which an application or a system is constructed from the analysis until deployment, production or even deprecation. There are many different methodologies with different strengths, weaknesses and use-cases. Some are more rigid, such as the Waterfall model, while others are much more flexible, for example, the Agile method. However, all of them share the same basic stages: planning and requirement analysis; designing project architecture; development and implementation; testing; and deployment and maintenance. [54]

To start with, the most traditional one is the Waterfall model, a linear model where each of the basic respective parts is completely separated from all the others, and each is started only after the one before has finished. It is effortless to understand and is easy to manage, but the tradeoff is that when a mistake occurs in one of the earlier stages, there is no process to fix it and all of the next stages need to find workarounds or other ways to cope with the flaws. [55]

Another one, the V model, is directly built on the Waterfall model. It adds a phase of testing to each level of higher abstraction, namely unit, integration, system, and user acceptance testing. This method suffers from similar fallbacks as the Waterfall model — that there is no flexibility in the process as each layer must be done in turn. On the other hand, it is very much a

test-driven development and thus most foreseeable problems are caught, and bugs are fixed before moving out to production. [55]

A different approach to the two already introduced methods is the Iterative Model, where the development process is done in small parts, and effectively all of the stages of the development process are applied to smaller problems from which the final system is composed of. All of the requirements are added to the product one after the other. Therefore there is no need for all of the requirements to be known at the initial phase, and can be thought of or desired at one of the later iterations. If there is any foreseen need to adapt, it is generally advisable to use some form of an iterative methodology over a linear approach. There is a threat though in that it is not as strictly separated into phases, which could mean unforeseen delays or wasting of resources on repeating some of the processes. Also, management is non-trivial compared to the linear methods. [55]

The last model introduced here will be the Agile Model, where there are many much shorter iterations commonly called sprints. After each sprint, there is a working software product, and the written functionalities in each iteration are high-quality code that is integrated into the final product and is built upon by the next sprints. There are many different approaches to agile development such as extreme programming, scrum, clear crystal, and others. It is extremely adaptable both to new technologies and to changing requirements because the customer is part of the development process through the whole life cycle and therefore there can be a lot of input from them. On the other hand, it is extremely hard to manage and generally unsustainable for large teams and too complicated projects. Although the incremental design produces working applications very frequently, the time between releases did not typically change in comparison to the linear models such as Waterfall. [56]

2.2 Cloud Software Development Life Cycle

There have been several attempts to propose a software development life cycle for cloud applications or discuss the importance of such processes. Although the processes are generally similar, there are usually slight differences to fully leverage the potential of cloud-based services. The process itself can be built on any SDLC as described earlier [57].

Zack and Kommalapati propose that a SaaS Development Lifecycle consists of the Envisioning, Platform Evaluation, Planning, Subscribing, Developing, and Operations phases, where the first, third, and fifth are also explicitly present in the standard SDLC while the other three usually are implicit [58].

In the Envisioning phase, the leadership discusses and chooses ideal business opportunities to act upon. In the cloud environment, special care should be placed on some of the aspects of the platform such as readiness for scalability and different approaches to reachability. It is at this time that the critical

decisions as to whether some existing solutions should be reused, bought, or created. This concerns each individual component of the final service, e. g., some authentication service may be reused while frontend may be created from the ground up.

Second is the Platform Evaluation phase, which is implicitly present even in normal SDLCs, but in the cloud environment, there are decisions to be made about what cloud provider to use and services to be used. The process must include as complete information as possible about the different options and their capabilities and prices. Another part of this phase is the definition of the conceptual technical architecture and plans for the proof of concepts for each of the shortlisted cloud vendors. The SDLC for the development process should be selected here. The choice should be made whether the process should be linear or iterative with the latter being the generally preferred option for most use-cases, especially on cloud.

Next up is the Planning phase during which the standard high-level development process starts and plans are made to define delivery deadlines and other important milestones. This part is dependent upon the organizational culture, type of service and chosen development model. Among others, the project schedule, feature requirements and design specifications are created similarly as they would have been during the traditional software development life cycle. The Planning phase is also the first phase which may be repeated multiple times during the creation of the application based on whether the chosen style of SDLC is linear or iterative.

Another slightly more different part of the cycle is called Subscribing, during which the required cloud services of the application are investigated and chosen with optimal pricing and computational power. The chosen sizes are production quality. IT administrators also begin planning operational needs for each of the chosen services, which also directly influences the selection process. Communication with the cloud provider is most important and active during this part because selecting the ideal level of subscription and resources is usually made much easier by the provider's customer support. Guidelines for the management of disaster recovery, backups and others are also created at this time.

Finally, there is the Development process itself in the next phase. This phase is almost identical to that of the normal SDLC with the only difference being the platform for which the product is being developed. Another important part is, of course, the creation of documentation, both user and programming, to accompany the code, the final application, and its parts. This is also one of the phases, which will usually be undergone several times, once per SDLC iteration. Other critical procedures are testing, integrating with other APIs and integration with the customer support processes.

The last phase, Operations, is at first sight also similar to the already known SDLCs, but the cloud environment brings several changes. The system administrators do not have direct access to the servers and as such their

position is somewhat different. The operations are also built on the knowledge gained during the platform evaluation, subscription and development phases, and the procedures are thus tightly coupled. The development and operations on the cloud then very naturally move to the DevOps approach,[29] where the two groups cooperate much more. In this phase, the deployment process must be finalized, catastrophic events scenarios tested and operations procedures made ready for production.

2.3 Security Development Lifecycle

A way to ensure that security is not an afterthought, but rather an actively developed and essential part of an application, is the Security Development Lifecycle. There are different approaches to ensure security during the development process [59]. It is vital for the application safety to clearly define and follow some guidelines for secure development and security testing, as well as to have incident response processes in place in case of a security breach.

One such document is Microsoft Security Development Lifecycle (SDL), which is a process to be implemented into a company's own SDLC to heighten the security of products and to actively work on threat mitigation during the whole application life cycle. Its goal is to lower the number of vulnerabilities through a set of practices to be followed.

Some of the key points made in the Microsoft SDL are described here. The first focus is to provide training and education on security issues to all parties involved in the development and roll-out processes. It also puts a high value on precise definitions of security and design requirements, and a system of measurement for these goals. Another part is choosing the right tools and standards, both for cryptography and security-related procedures, as well as for other purposes. This is tightly coupled with mitigating the risks arising from using third-party software, which may introduce vulnerabilities into the dependent applications as well [60].

All software should also be extensively tested, passing all static analysis, dynamic analysis as well as penetration tests. Static analysis should be run regularly during development, dynamic analysis as often as possible, ideally during all builds, but this may not be feasible because of longer test times. It should still be tested regularly and if not during the build, then at least after builds to get relevant security information as soon as possible. Penetration tests should also be run to ensure safety against dedicated cyber attacks [60].

Every company should take security into consideration during all phases of the development cycle, and the ideal process for a given company or project should be found, but that is outside the scope of this work.

[29] More discussed in Section 2.4.

2.4 DevOps in the Cloud

It is almost impossible to completely characterize the DevOps approach because there has never been any form of manifesto or definition and it seems there is no intention to do so [61]. However, there have been many attempts to define the term [62, 63].

What most descriptions of the term have in common are efforts to bridge the gap between developers and IT operations through clearer communication and collaboration. There are some other common factors, such as approaches to automate as much as possible, deploying often (even multiple times in a day), close monitoring, or microservices-oriented cloud development, but these are only enablers or the reason to look into DevOps. However, they are sometimes still crucial parts of DevOps, without which it usually cannot properly work and which are often thought of as the first thing when talking about DevOps. It is stressed by some authors that DevOps is primarily about the culture and not about the tools [64].

It was initially created to reduce the number of clashes between developers and IT operations, where one did not understand the other and often did not consider the other party when making decisions. An example for such clashes was finger-pointing when a server had gone down, instead of trying to fix the issues as well as change the process that may had led to the failure. One of the pivotal presentations on the theme was Flickr's 10+ Deploys per Day [65], where the problem was illustrated along with one of the first real-life examples of cooperation of this kind.

DevOps promotes inter-team cooperation with a mindset of people walking in each others' shoes. Developers are partly responsible for the monitoring of their applications because they have a better idea what the logs mean and what would have been possible reasons for any failures. They have greater responsibility for the code that is being run and that makes them more aware of the situations when something may go wrong. So they often make more meticulous fail-safes into the code and communicate better with the operations how to employ them.

This builds trust on the operations' side, which strives for the stability of services, is more open to faster release of the new features. They may also be more involved in the development process and give their expertise more often when decisions about the implementation are made. Through this close cooperation, some responsibilities are redistributed, and the strict separation of developers and operations disappears further.

In smaller teams, this may even create a single team, where both development and operations are shared responsibility, but each member usually retains their main expertise to some extent and is more devoted to that area.

DevOps is also often coupled with a high degree of automation of deployments, and infrastructure and configuration changes. This can boost reliability because less time is spent on those and therefore more resources are available if

something needs fixing. Furthermore, all the steps of the change are implicitly documented in the scripts.

Infrastructure is provisioned in an Infrastructure as a Code manner, highly simplifying the process and making it extremely easy to replicate. Configuration of virtual machines and runtime environments is preferred to be done in the same way. This is highly dependent on cloud and virtualization technologies.

The changes that development in the cloud brings often result in an ideal environment for DevOps. A large part of operations' work is moved to cloud configuration, in some instances even moved to the responsibilities of a CSP, e. g., patching of OS in PaaS models. It is also easier to keep the different environments closer together, which means the developers may have a much better understanding of the system as a whole than the operations engineers. Cloud also promotes and usually provides many tools for a high degree of automation of all parts of deployment.

Because DevOps is very tightly coupled with fast delivery, it often leads teams to overlook or disregard some security aspects. However, this approach also provides them with the tools to fix them very fast once identified. [66] This is the reason that some companies try to integrate security into DevOps more, creating what is commonly known as DevSecOps. However, some authors suggest that DevOps should already contain more teams and does not need extending the name with more company branches [62].

DevOps also generally promotes security as a responsibility of everyone, not just a dedicated group of security engineers. Developers need to understand the risks and vulnerabilities they may have introduced during development. This is not to say they should become experts on the topic, it again only promotes clearer communication as well as the shift of some responsibilities.

Chapter 3

Security Critical Areas

Several problems that the developers may encounter during development for cloud will be briefly introduced here. This section serves solely as an introduction to the solutions in Chapter 4.

This work does not focus on general security aspects of development and does not intend to list all the areas that the developers must be mindful of. Whenever an application is being developed, care should be taken to ensure its security and some general guidelines are presented in different works such as Writing Secure Code [67]. The platform, language and framework should always be taken into consideration because they usually have different best practices and security issues.

This work will only focus on cloud or Azure specific issues that developers may encounter when creating cloud-ready applications.

3.1 Securing Data

Correctly handling data in all possible scenarios is not an easy task, but it should be one of the developers' priorities concerning security. There are problems with correctly assessing the data and how much effort and resources should be put into storing and securing it, as well as obtaining all the relevant information about the options available. Section 4.1 will focus on handling different types of data, while Section 1.3.3 and Section 1.3.2 have introduced some of the possible storage options for data on Azure.

However, it is essential to not only be aware of the options available but also when to use them. There may be some situations where either the data is not worth the additional work of developers to use different encryption schemes than for example the ones already present on Azure storage. In other cases, time may be of the essence and not only calls to outside services such as Azure Key Vault are out of the question, but even encryption of streamed data may prove to introduce a significant overhead. In the most basic example, a HSM complying with some standards may be too expensive to operate and as

such should only be used when necessary. For these reasons, there must be a process in place to accurately assess the desired protection.

3.2 Securing API

If microservice architecture is employed, as is now very popular on the cloud, different security measures may be employed for each service. However, very high priority should be placed on securing the API, which will ideally serve as the only point of entry for all requests. This significantly reduces the attack surface because more developer time may be allocated on securing the API rather than on securing other services more than necessary. The API may also run in several instances to ensure reliability even when one instance may stop working. Also, some Azure resources may provide additional safety and performance measures such as load balancing or Distributed Denial of Service (DDoS) attack recognition and prevention.

3.3 Identity Provider

Authentication schemes should be decided upon early in the development process, and usage of some identity providers or lack thereof provides some specific advantages and difficulties. Section 4.5 will introduce some of the properties of federating identity services to an external service. It will also touch on the relation of authentication and authorization if they are separated in different services.

Protocols that the Azure Active Directory uses for authentication and authorization of services were described at length in Section 1.3.1.

3.4 Management of Secrets

Secrets are crucial assets used very often, but there is the ever-present risk of their disclosure. Through them, an attacker may obtain direct access to sensitive data in persistent storage or other information that could seriously endanger the business.

In Section 4.3 different ways to access secrets by an application will be overviewed and discussed. It will be examined based on where the secrets are persisted and how are they made accessible to the application, as well as when this process occurs.

Section 4.2 will also describe a possible process to change secrets such as keys periodically with no service downtime and ensuring security of the secrets. This process will assume two interchangeable keys. This situation is not uncommon on Microsoft Azure, where access to the data plane of several resources is controlled by exactly two keys.

Chapter 4

Proposed solutions

4.1 Data Classification

All data should be classified and handled based on the value it provides for the business, its volume, and the structure, e. g., although telemetry data is usually not disclosed to third-parties, it is to be handled differently than an Azure Storage account access key which may protect data of all customers. It will be commonly in the development team's scope to determine what data is considered critical and how it is going to be stored. Storing some sensitive information may be consulted with information security engineers or lawyers, because it may be subject to security standards or low (e. g., credit card information).

It may not be worth the additional effort to store some data using the highest security standards. A risk assessment model, such as Damage, Reproducibility, Exploitability, Affected, Discoverability (DREAD), should be used to make a fully informed decision about the extent to which resources will be spent on the security. For example, anonymized usage data of an application may not pose a seriously large risk to a company, but it may still be preferable not to disclose such information. On the other hand, payment information needs special attention when handled or persisted for later use and the resources that would have been spent on securing the former may be used on the latter.

The data should also be handled differently based on who the owner is. For example, keys to access storage account owned by the company itself can be cached in the application's memory for later use, unlike passwords or access tokens belonging to users. Generally, secrets that the party running the application is only consuming should be discarded as soon as possible.

This section will introduce and differentiate between *secrets* and *sensitive data*. Although both of these are confidential, there is a difference in its nature and therefore also in its usage. Secrets do not have any obvious meaning assigned to them and instead provide access to some other service or add

some security to the system. Examples include credentials to databases, SSL certificates, or WiFi passwords. On the other hand, sensitive information holds meaning by itself and is usually protected to some extent. Phone numbers and other personal information, or email conversations are examples of sensitive information. In most cases, authentication or some secret is required to access this sensitive information.

Passwords are also secrets, but there must be a distinction between secrets that the company operating the software owns or is authorized to store and use, and between third-party secrets. User passwords and other secrets not actively used by the software should never be stored in any form that makes them retrievable, including encryption. Outputs of hash functions run over these secrets may be stored for authentication reasons.

We will focus more on the secrets that the software in question is actively using, not the ones it is consuming from users and other services.

Because of the secret's nature, they do not provide business value by themselves and as such are used only in communication during authentication, or confidentiality, integrity and authenticity purposes. If a secret is used to access storage or some data, it should be usually separated from the actual usage of the data. In this case, the secret is only the means to obtain the data, while developers creating business logic concerning said data are not accessing the secrets. The usage of those secrets should be handled by the runtime, framework, or infrastructure part of the codebase. Similarly, the secret may be used only to send information securely. Here also the usage of the secret is to be separated from business logic, and in some cases, it can also become the responsibility of the framework or platform to locate and use the secrets.

On the other hand, sensitive information should be considered as having no meaning on levels other than business. That is not to say that they are not to be checked for validity or be formatted based on the content, but the data itself should not be interpreted, but rather only transported.

There are several options for data storage on Azure, as shown in Section 1.3. Preferred storage for secrets would be Azure Key Vault, which may also be used when an HSM is needed for compliance. It provides version control and auditing options as well as access control, although not on the scope of individual secrets. This is of course not scalable for large quantities of data, and structured data cannot be stored there at all. In most cases, Azure storage or SQL databases would suffice, and there is no compelling security reason to pick one over the other. That may be decided purely on the usage needs in this scenario.

4.1.1 Disposing of Data in Memory

Another factor is whether the data is to be persisted or only used once or for a very brief period. Special care should be taken to dispose of critical

information after its usage and not to keep it in memory for longer than necessary, if the language and runtime provide this ability.

It is essential to know how the runtime handles data structures that we might want to not store in memory for prolonged periods of time. This must be managed manually in some languages such as C, where the developer must zero out the memory and then potentially release it. However, some languages do not provide this kind of control and manage the memory themselves, often in a non-deterministic manner and without any guarantees.

For example, strings in C# are immutable objects with no way to be scheduled for garbage collection by the developer. [68] That means that any secrets stored in the memory may reside there indefinitely.

There has been an improvement on that with the introduction of class *SecureString*, which implements the *IDisposable* interface and thus has a method to be disposed of immediately.

It, however, does not solve the problem that the contents of the string are unencrypted in memory. It was supposed to use the host operating system's API to encrypt the data and have it in plain text only when it is needed. That would limit the time the secret data is accessible in memory, but it would still be there. Furthermore, this API is not available on non-Windows platforms, and the contents of *SecureString* are not encrypted there. [68]

For these reasons, it is advised not to use *SecureString* in new code [68]. The recommendation put forth by Microsoft also suggests that it is better to avoid credentials and instead use certificates and other means for authentication [69]. *SecureStrings*, however, still may be used to control that the memory content is cleared and disposed of when requested. When a developer decides to use it for this reason, he should be also aware where the data came from and what is he going to do with it. For example, if he gets a password as a string from user input and then computes and compares its hash against stored value, there is already one copy of the password in memory and creating another *SecureString* would not achieve much. Reason of any usage should be explained in comments for future reference so that the intention is not misunderstood.

4.2 Key Rotation

It is important to keep access to persisted data as restricted as possible to impede data leaks.

Access keys to Azure Storage accounts and other resources with authorization via static tokens should be kept confidential at all times. Since they may provide total access to and control over all the data in the given resource, its disclosure could have catastrophic consequences.

When applicable, one should prefer to use SAS tokens, because they provide more granular control about what data can be accessed and what opera-

tions may be performed on it. Using delegated access also enables usage of the Valet Key pattern [70]. The only other approach to handle access to the data without providing others with keys would be using a service that would fetch and store the data on others' behalf. That may, however, be very expensive in terms of bandwidth and computational resources. If further supported, even more control over access can be achieved by connecting a SAS token with a stored access policy, which puts another set of restrictions on the usage of the SAS token, but is evaluated at the time of access. Unlike the SAS tokens, which cannot be changed once signed, the access policy can be modified and through its deletion or alternation the tokens can be revoked or made more restrictive.

Due to the SAS tokens being signed with one of the access keys, there is no avoiding the keys' usage. Therefore they need to be protected and kept confidential.

Some Azure services such as Azure Storage have two access keys that can be used interchangeably. Ideally, there should be only one key in use at the same time, except during rotation as stated later here. [39]

Rotation of keys in this work will mean the act of changing the key that is actively used to access the protected resource. This may involve more than a single key. The old key is usually revoked during this process but may also be valid after the rotation. Regeneration of a key will mean the act of changing the value of a single key to a different value. The old key is always revoked during this process.

One way to enhance the security of access keys is to rotate them both on a regular basis and when a security breach incident occurs. [39] There should be a policy in place on how to regenerate and rotate keys so that there is not an application that ends up with an old nonfunctioning key that would render the application unable to connect to the service. One option to make this process much easier is to centralize the keys by putting the ones currently in use into Azure Key Vault as a secret and letting only authorized applications to access them there. [39]

Developers must expect that rotations of keys will occur and they should make the applications ready for these events. The most important thing is not having hard-coded access keys in the source code. Ideally, the application can react appropriately to the change of secrets. Some strategies to achieve this are shown in Section 4.3. Before the application fails, it should first check for a new value of the secret and only if it has not changed or it is still not working should it exit with response or status code signalling failure.

4.2.1 Key rotation process

Key rotations should have minimal influence on the end-users. In Figure 4.1 there is one possibility of the process of rotating the keys for a resource that has two keys providing the same level access as described by Microsoft [39].

If the applications utilizing the keys were made ready for key rotations during development (the keys propagate to the application without the need for restarting any of the services), there will be no downtime of the applications.

To fully explain Figure 4.1, a textual representation follows. If there are keys *Key1* and *Key2*, with *Key1* being the actively used one, the rotation would proceed as follows:

1. *Key2* is regenerated as to ensure that no party could have had prior access to it;

2. All appearances of *Key1* in configuration files and source code[30] are to be replaced with *Key2*;

3. The applications are tested and restarted or deployed, if needed for the changes to take effect. There may be some time needed between the regeneration of the key and its propagation to the services[31] and the applications should not be made to use the new key before such interval passes;

4. *Key1* is regenerated so that no application that was not explicitly given the new key can access the protected resource. At this point only *Key2* should be actively used. Any attackers in possession of the old key or applications that were not given the new *Key2* should not be able to connect to the protected resource. [39]

Next rotation would look the same, but with the keys' roles reversed.

It is vital to correctly assess the extent to which the rotation may disrupt the operation of different services. It will of course influence all services directly operating with the rotated keys, but also for example all services that have obtained a SAS token, because those had become invalid when the key used for signing was revoked.

There may be some time during the rotation when both keys are being actively used because only some of the applications have already migrated to the new key.

One way to make this process much simpler is by centralizing the keys in a single place such as a Key Vault and letting the dependent applications access them there. In this case, there is only one place where a change of a key must occur, and testing may be less rigorous, especially from the second rotation onward.

This process has modelled mainly the situation of a scheduled rotation when there is no specific reason to believe that an information disclosure event occurred. In case of such a breach, the process may change to proceed faster.

[30] It should be noted here that having the keys in source code is a sign of bad design and should not happen.

[31] For example, it may take up to ten minutes in Azure storage. [39]

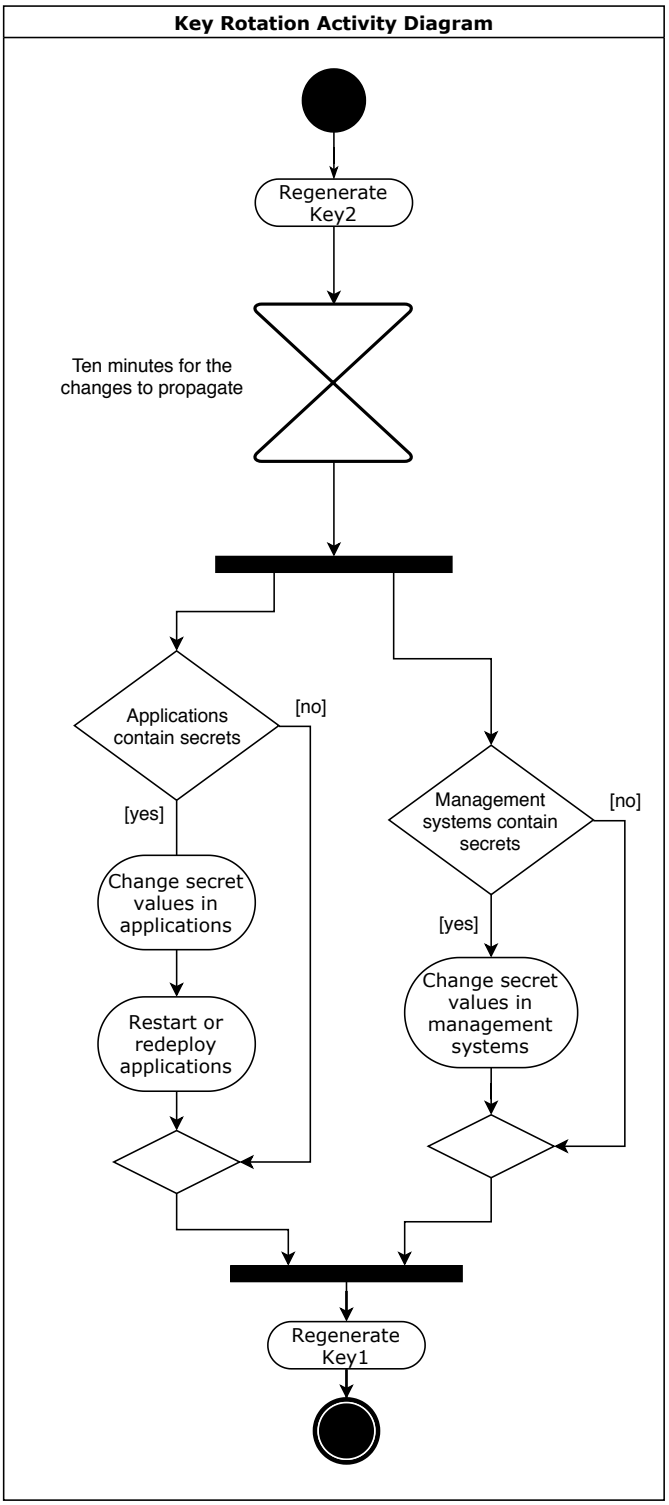

Figure 4.1: Key rotation activity diagram

Based on the value and confidentiality of the data at risk, and the importance of the services accessing it, if a key or a SAS token was leaked, there must be a decision made whether data security or service availability should take higher priority.

If the data is of low value, the process may commence as it would have if it were scheduled for this time. This would speed up the rotation schedule, but it would not disrupt the operations in any other way.

In case the data is valued more, service uptime is critical for business, the first key regeneration may be skipped. This would make the applications use a key that was never in use but may have the same value from the time of the last rotation. It may be reasonable to assume its confidentiality, but it is better to use a fresh key, especially if the rotation occurred as a consequence of information disclosure incident. For this reason, it is advisable to have one more standard rotation after the sped up one.

To further lower the time data is exposed, the key may be changed only for selected services that are critical for the business. The other services will then become unavailable.

If the data in question is critical, both of the keys may be regenerated at earliest available time. This may incur a more extended period of service unavailability for all services in question (because they need to access the storage, but they do not have valid access keys), but the data leakage is stopped as fast as possible.

In all events, there must follow a thorough investigation as to how the incident started and how the access key or SAS token was leaked. That comes hand in hand with monitoring and logging operations of services such as Azure Storage and Azure Key Vault.

This whole process, of course, works for any resources or services that have at least two valid keys with the same function. The case with one key only depends on the development and operations team and how fast they can change all the occurrences of the key after it is regenerated. This process could not be done differently for the lack of any intermediate period between the revocation of the old key and introduction of the new one. The cases with three or more keys are trivial to transform to the problem with only two keys by discarding the others. Another option for more secrets is to have multiple pairs of keys, that would be used for different scenarios, e. g., one pair for the development team, one pair for reading services and another pair for writing services. This could prove beneficial if there is an incident with one key because it does not disrupt the others in any way. Another much harder to manage option would be only to have one inactive key with which any other key may be rotated if needed. This means that the inactive key would be at any given time the last key that was rotated out. This would be harder to manage because there is no easy and clear way to identify what key is used where.

4.2.2 Key Vault Managed Storage Access Keys

Azure Key Vault can also rotate the storage keys automatically without any outside intervention after initial setup.

Key Vaults must be first permitted to manage access keys to storage accounts.[32] Then a storage account can be added to a Key Vault as a managed storage account. The Key Vault can be configured to either rotate the keys periodically or only when requested.

The Key Vault never gives out the storage access keys (although they can still be found in the storage account if need be), but instead issues SAS tokens based on prior definitions.

There are some limitations put on Key Vault managed storages. There should always be only one managing Key Vault for any given storage account. [71, 72] That, along the fact that Azure Key Vault cannot have access policies set for specific secrets as stated in Section 1.3.2, means, that the principle of least privilege cannot be implemented. For example, when one application needs a SAS token to read table entries, and another one needs to add entries into tables, although there can be two different SAS definitions in the Key Vault, both applications will be given either token when requested. This provides some safety when one of the tokens is disclosed because the attacker can only access some part of the storage as set in the SAS token, but if the application is also made to request a token with different privileges, the Key Vault will provide it, even though the application should not need it.

If a Key Vault is managing the keys, there should also not be direct regeneration requests on the storage account, but those should instead be directed towards the Key Vault.

Experiments have also shown that Key Vaults do not use the key rotation scheme as described earlier, but instead only change one of the two keys during each rotation. This would suggest that it relies on the assumption that the keys have not been compromised. This ensures that the SAS tokens that would have expired after the next key rotation are not revoked in the event of key rotation, meaning that there is no need to reacquire SAS tokens in the event of key rotation.

4.3 Secret Handling Methods

One of the most important aspects of the development of secure applications on cloud is handling secrets. It is in the nature of cloud-based distributed resources that most services will have many dependencies against which they need to prove authorization. In the simplest approach, which is still widely used, it may be implemented by a pre-shared secret. Such secret may be used

[32] Azure Key Vaults are pre-registered in all non-government clouds with ApplicationId cfa8b339-82a2-471a-a3c9-0fc0be7a4093. [71, 72]

as a password to an SQL database or credentials in some OAuth access grant flow.

Secrets here will refer only to pre-shared secrets or private keys and other data essential for authentication or authorization. There may be much more sensitive data, such as payment information, but that is handled differently, as discussed in Section 4.1, and is secured in the same way, as it would have been in an on-premise system. General sensitive data usually has to be fetched when needed, because there is a much larger quantity of it than an application can store in its memory and it may change more often than for example a database password. On the other hand, there are several different access scenarios for secrets access, each with its advantages, drawbacks and limitations.

4.3.1 Secrets Storage Mechanism

The first point of view from which this issue can be explored is the system that persists and provides the secrets. The issue is not only how well the system is protected, but also how hard it is to update the secrets and who can access them. To be able to automatize some processes such as key rotations without disrupting the system (see Section 4.2), it should be possible to update the secrets at any time, even when the service is running. Limiting authorization to accessing and changing the secrets is also essential because it limits threats of unauthorized disclosure, both malicious and accidental.

Plain Text Secrets in Configuration Files

The most basic and insecure way of handling secrets is having them in plain text in configuration files. This is very dangerous because such secrets are often saved in version control systems, either on purpose or by mistake. Such practice should be disallowed, and the developers are encouraged to research and use different methods, so that the secrets are visible only to people and applications that need them. Anyone with access to the source code or configuration files may have direct access to those secrets, and through them to any service they should be protecting.

Such secrets are not protected at all, and the only advantages are the simplicity of the solution and the fact that the secret is transferred only when the application is deployed. On the other hand, that also means that if one of the secrets needs to change, the application needs to be restarted to reload the configuration files. There is the threat that if the configuration file is leaked, all secrets are disclosed.

An even more naive approach is to have the secrets in the source code. Although that might be a viable solution for one-time personal projects and the most basic proofs of concept, this solution is not sustainable. The secrets are not centralized, so it is harder to find them, and some may even figure in multiple places. When changed, the applications need to be even rebuilt and

redeployed for the changes to take effect. What is even more critical is that the secrets are saved in version control systems and anyone who has access to the code for any reason, also has direct access to all the secrets, same as with secrets in configuration files.

Encrypted Secrets in Configuration Files

A more secure approach is encrypting the secrets or the whole configuration files using utilities such as git-crypt. Because the configuration files are encrypted, only people and applications with the decryption key may access the secrets. It is then safe to commit such configuration to a version control system.

However, this approach only transforms the problem of sharing the original set of secrets into sharing the decryption key for the configuration, which is just a different secret. Not only do all the developers working on a given application need to be given the decryption key as well as having it updated if it is changed, but the execution environment that will build the application in the cloud needs it too. This new decryption key must be input manually, or other means to share this secret must be used.

This can be improved by having several files with different decryption keys for each environment. This way developers do not have access to production secrets, and unauthorized users with access to source code cannot use even development or testing environment secrets.

Another problem may arise if the configuration mixes general settings for the application with environment-specific secrets. Then either all configuration files must be changed individually at the same time to use the same settings, or the general options must be split from the secrets. This would create multiple configuration files, which may be harder to manage.

Also, some secrets may have to be duplicated if they are needed by multiple applications with different configuration files.

Environment Variables

A somewhat different approach is to put the secrets directly into a runtime environment (e. g., Azure App service) directly via environment variables. This can also be achieved either manually, which would certainly not be recommended, or as a part of the deployment process. In any case, the secrets are generally not shared with developers, except for people with access to the running instances or deployment pipelines if that is where the environment variables are set. The deployment process may usually be much more restrictive than source code for the application, therefore giving access to fewer people.

If the environment variables were set automatically during start or deployment by a process on the cloud resource, there would have to be a configuration

script accessible by this process. This is then almost precisely the same case as if there were a configuration file read by the application. The main difference between the two is which process accesses the secrets and when this is happening. Here it would be the parent process (or another ancestor), that would set the environment and pass the configuration variables to the application. In the case of regular configuration files accessed by the running application, the parent processes do not access the secrets at all.

Therefore there are now multiple processes operating with the secrets and most of the drawbacks of configuration files would remain, because there would have been a way to configure the environment variables automatically. The main advantage would be that people responsible for deployment, rather than developers, would have access to the secrets, which may be a smaller group.

Furthermore, on Linux, the environment variables are always tied to the running process. Although .NET Core supports persistent environment variables for users and the local machine through the use of registry keys on Windows, Azure Web App does not let the running application change these. Therefore even on the Windows platform are all relevant environment variables process-specific. Experiments have also shown that Azure restarts the App Service every time an environment variable is changed. This can disrupt the operation as well as induce some downtime when the application is starting up.

Secrets Management Systems

Secrets can also be centralized in a configuration management system or a secrets management system, such as Azure Key Vault. This way the secrets are completely split from the code and it offers much higher agility when changing the secrets, because the application may be written to expect and react to events when the secrets change, mitigating the need for restarts in these scenarios. Responsibility is also split with clear borders, where one developer may be responsible for the management system itself (i. e., the secrets at rests), while a group of developers creating the application accessing those secrets need to worry themselves only with handling the secrets during runtime.

These systems also offer a way to version secrets without putting them in a version control system such as git. It also allows multiple applications to access the same secrets if need be, meaning it is centralized and if the secret changes, it needs to be changed only in the management system. Furthermore, it allows for logging of accessing and changing the secrets, which may be invaluable information in security breach situations.

The applications know only a URI, where the secret is supposed to reside. They have to be authorized to access the desired secret in the management system and then they can retrieve and use it.

This solution can also be described as a mere transformation of the original problem into a problem of obtaining credentials for safe authentication against the secrets management system, but this feature has been implemented in Azure by employing managed identities as discussed in Section 1.3.6. Usage of managed identities also makes a clear separation of concerns between the developers and the cloud service provider. The developers are directly responsible for safely storing and disposing of the secrets in memory, including disabling caching for high-value secrets, while the CSP handles storage and accessibility of the credentials. The application is still responsible for making all the calls to acquire the Azure AD tokens to access the Key Vault, but may do so with the use of an official SDK if it is available for the given programming language. When the call is serviced by the underlying virtual machine, the credentials for Azure AD are injected, and thus the application cannot leak them by mistake.

Additional security can also be achieved by managing access to the secrets management system based on IP addresses or restricted to a VPN that contains the management system. Then even in case of credentials being leaked, the secrets may still be safe if the attacker cannot use or pretend to be using an authorized IP address, the connection will have been refused and the secrets are not leaked.

This approach enables some more options to be explored, specifically when the secrets are to be resolved. One may resolve the URI during deployment, startup of the application or during runtime, where there are also different options as to for how long the secrets are cached or if they are to be disposed of immediately after use.

4.3.2 Secrets Resolution Time

The other issue is when the resolution occurs. The previous section showed that not all storage options offer much freedom as to when the secrets are retrieved. However, it is crucial to be able to periodically change values of the secrets without disrupting the service operations. Because of that, this section will mostly focus on the case when a secrets management system is employed.

Secrets Resolved During Deployment

In case the secrets are resolved during deployment, there are usually steps in the build pipeline, where the management system is queried for the secrets, which are then injected into configuration files. Because of that, this approach shares some of its strengths and weaknesses with storing the secrets in unencrypted configuration files, while tackling the largest weakness of the exposure of the secrets.

The only system having to access the management system is the one where the software is built. There must be a way to programmatically access the secrets, e. g., via a REST endpoint and the system must be able to communicate with it including providing authentication credentials or authorization tokens. The applications using these secrets do not have to have any knowledge of these processes or how to communicate with the management system.

This certainly simplifies management of access policies because only the deployment process must access the secrets, but it comes at the price of agility when the secrets have to be changed. If any secret is rotated, the application must be wholly redeployed, possibly making the service unavailable for some time. Because of this, the approach is very impractical, especially if the secrets change on a regular basis.

Secrets Resolved During Bootstrap-time

Another option is to resolve the secrets on application start-up. This usually separates the secrets from the main configuration, and programmatical access to the the management system must be available in the form of an SDK or an implementation must be made for each language, in which an application that needs to access the secrets is written.

Developers also need to solve the issue of storing the secrets indefinitely while preventing the risk of exposing them through, for example memory dumps. Purging the memory is almost impossible in managed languages such as Java and C# as briefly discussed in Section 4.1.1

Because the application only needs to be restarted, it is faster and easier to make it use new secrets when needed, but it is still rather slow and straining the virtual machine's resources too much. There may still be a time when the service is unavailable during the restart if multiple instances are not running at the same time.

Because the communication with the secrets management system has to be already implemented in the application, it can serve as a starting point for developing more advanced solutions. For example, the secrets may be loaded during bootstrap-time and then requested again later using the same routines if the secret is being rejected by the resources where it should work as an authorization token.

Secrets Resolved Just-in-time

The most agile way to handle changes of secrets is to acquire them only when needed and dispose of them when used. This approach makes the possibility that a secret is no longer valid very small. It does not completely eliminate it though, and retry policies still need to be implemented for critical services, which have to have as high availability as possible. With this method, the problem of safely caching the secrets is not relevant because they are not

cached for later use at all. However, the developers should still take care to signal the compiler or runtime to not use hardware cache for the secrets, as with any other method.

On the other hand, if the service is for example a back-end application servicing hundreds of requests every second, it can strain not only the secrets management system, but also the network. It may also introduce an increase in execution time and lower the service's availability because a call has to be made to the management system as its dependency for each request to the service. Also, in this case, the communication with the secrets management system must be fully supported by all the applications requiring it, but greater emphasis should be put on optimization of those calls if possible (e. g., by ensuring that the dependency and the management system are in the same data center), since it is often used during normal operations, not only during start-up, and because it is to be called much more often.

This approach can be very well extended to store the secrets safely in memory for some time if the secret is expected to be reused soon. That way some of the disadvantages are tackled while still being reasonably agile with secrets changes. On the other hand, ways to store the secrets securely and dispose of them after a given time-out are further challenges for development possibly introducing some vulnerabilities to the code, which may dissuade some teams from fully implementing it for regular use. A short introduction to this topic can be found in Section 4.1.1.

4.3.3 Discussion

It is undoubtedly advisable to store secrets securely and not use a standard version control system for such tasks. The management system used should support authentication of applications through Azure AD or another identity provider, otherwise some other method for accessing the secret to authenticate against the management system must be available beforehand.

Strategy for resolving secrets is dependent on the nature of the secrets as well as the application accessing them. If the secrets are static and may change exceptionally rarely or never, they can be saved into the application during deployment or during start-up. This also provides superior availability because it does not introduce any dependencies in runtime. If the secrets have a high value and storing them for prolonged periods of time in memory is not desirable, one can query them when needed and dispose of them right after use. For more standard secrets, such as access keys that may be regularly rotated but excessive querying is undesirable, a combination of the given solutions may be the most appropriate solution. The secrets are then queried only when needed, but they are stored in memory for later use. There must be systems in place that clear these secrets after given time as well as routines to try to acquire new values of secrets, that were changed between usages.

4.4 Data Protection API

Multiple instances of the same application may need some data to operate and they either need to negotiate the creation of the state or must be able to share and use some other instance's state. They may run simultaneously and some may also be stopped automatically when the load lessens, so the state should not be dependent on a single running instance and should be instead persisted in some external system.[33] Data protection API in ASP.NET Core is an interesting example of how a state of the application may be needed to be preserved explicitly in shared storage when running in the cloud environment.

It is made to encrypt data while automatically handling the key ring and key rotation. The keys that the Data Protection API uses for encryption are usually saved on the machine running the application and in the case of Windows, the keys are also protected by the operating system's Data Protection API [74]. That is an entirely different system and because it is not system independent, portable applications written in .NET Core should not assume it will provide any protection. The ASP.NET Core Data Protection API also uses purpose strings, which serve to identify whether the given operation should be permitted, or the application is trying to decrypt data that was encrypted by some other application deriving its keys from the same entropic material. This, however, does not secure the keys themselves in any way, and if an application may execute any code, it can obtain the master keying material and through it derive the other subkeys. It serves only to separate usage in different contexts, e. g., the string can be based on a username and then any user other than the one, on whose behalf the data was encrypted.

It is made to work out-of-the-box on standard systems, but some problems may surface when moved to the cloud. The problem is that the creation of the keys is dependent on an output of a Pseudorandom Number Generator (PRNG). Therefore, if the application is being run on multiple machines at once, these will not be able to access each other's keys and there is an extremely high probability that they will generate different keys.

For example, if the application would create bearer tokens, it would work as intended as long as the application minting the tokens was also the one checking them later. If one application creates a token and another one validates it, it will be unable to decrypt it properly.

This problem occurs because the application is not stateless, but the state does not propagate between its instances. Although the developer does not explicitly generate or save any keys, they need to understand the components they are using and what it means for the system as a whole.

In the example of the Data Protection API, there is also already a solution ready to be used with the Data Protection API having configuration options

[33] There may be patterns used for electing a leader instance, which would provide the necessary keys, but since the state must be both shared and persisted in this case, it would not solve the problem on its own. More on this can be found at [73].

to store the keys in Azure Blob storage. However, because the ASP.NET Data Protection API depends on Windows Data Protection Application Programming Interface (DPAPI) for encryption of the persisted keys, the keys are stored in plaintext when the storage location is changed or the application does not run on Windows. This too can be solved with an already existing solution, where the keys for encryption stored in a Blob storage are themselves encrypted by a key in Azure Key Vault. The reason that the Key Vault is not used for storage of the encryption keys is that they have to conform to a given structure and hold some metadata and because the keys are not deleted after expiration, so it may use a lot of space in the Key Vault.

4.5 Authentication and Authorization of End Users on Azure

Most applications must be able to correctly assess the identity of users to be able to authorize them properly. Authentication is the process of assessing the identity of an entity, while authorization is the decision wether an authenticated entity may access a resource or request an operation. While authorization cannot be very well delegated, authentication may be performed by an external identity provider such as Azure AD or social media identity providers. This would immensely simplify development, user administration, and in most cases also enhance the user experience.

Authorization is usually implemented as a set of rules which are looked up and then access is either allowed or denied. If the identity provider is out of our control and we cannot receive user's roles in the token, user's roles must be saved internally.

On the other hand, authentication involves handling user credentials and storing their information, which may prove very difficult to do securely and in compliance with all laws and best practices. Using outside providers may limit the need to store personal information and prevents the mishandling of credentials because they may never be obtained. However, some personal information may still be needed as some providers only provide a unique identifier, while the application may need much more information about the user. The usage of external identity providers does not create a safe application by itself because great care should also be taken to misuse received authorization tokens, but these have much shorter lifetime than standard users' passwords and can be revoked and regenrated without requiring user interaction.

It can also benefit the users, who do not need to create and remember a new password or reuse an old one. They may also be signed in with a single click if they already used the given identity provider in recent time. This significantly improves user experience and also protects the user's credentials, because their information is stored only at the identity provider's server. Users are also more likely to handle their passwords securely if they do not have multiple complex

passwords but instead a single or very small number of passwords. That limits the number of places where an attacker may obtain the user's credentials.

Federating identity[34] also simplifies user management in the way that many features related to authentication may not be implemented and instead consumed from the outside service. For example, resetting passwords may be difficult because the user needs to obtain a token for the operation, but standard communication may be unreliable (the token may end up in the SPAM folder), as well as not secure. Furthermore, some advanced features such as MFA may be very hard to implement correctly and reliably. Furthermore, if standards concerning credentials storage change, the changes will be implemented upstream at the identity provider with much smaller changes left to the developers using the service.

Integration with multiple identity providers is also a possibility. For example, an application can then serve users from two different companies, each implementing their own identity services.

Even if it is not desirable to federate authentication to some external provider, it can be worthwile to invest the resources needed to create a private identity provider. Usage of any such service clearly separates authentication code from the applications. This solution may also be reused multiple times, and provide single sign-on capabilities.

4.6 API Security

Securing the API should be one of the top priorities, but that will not ensure that it will work flawlessly and that there will be no vulnerabilities. Developers should create other services with this in mind and balance the complexity of input validation and layered security. The API should sanitize all user input and then make calls to other resources with a low probability of invalid format. However, services operating with high-value data should take further measures to ensure safety even if the API can be breached, such as limiting the inbound traffic to a VPN where the service resides, or even only to the IP address of the API.

All servers and ports except for the API may be made inaccessible from the outside Internet and instead be only available from a VPN in the cloud. For example, all back-end servers may be serviced through the API and refuse all connections from other sources. This would shift any potential attacker's attention either to the API, the virtual network itself, or to gaining access through other means, such as social engineering.

[34]Federated identity [75] is a pattern, where multiple identities of a user are linked together. The participating identity services need to have some trust relationship (it may be only one-way). This is similar to Single Sign-On (SSO), where only authentication is federated.

Of course, there may be a situation, where one of the back-end services is required to be accessible to some other service that may not be inside the VPN. In that case, a decision should be made if the service can also be made accessible through the API and if this additional work is worth it. In some scenarios it may be best to move this one service out of the VPN, worsening its security, but ensuring that there is only one open endpoint in the network.

There are several things to do to secure an API properly. The intended users should be identified first, and the service should be made accessible only to them, if applicable. If the service is, for example, an internal company accounting system, a firewall can be configured to only allow access from the company's internal network. Also, there should be robust authentication in place.

At all user-facing endpoints, all input and requests must be validated and potentially sanitized. As the API is now the central point entry, it becomes the main defense perimeter. Special care should be taken to prevent buffer-overflows, SQL injection, cross-site scripting, and other possible vulnerabilities. This is similar to the Gatekeeper pattern [76], where the public endpoint only validates and sanitizes requests and then passes them to some internal trusted service. In this case, the Gatekeeper cannot access other services such as storage on its own, and no service other than the gatekeeper may make requests to the trusted service.

When paired with some other Azure services (Azure DDoS Protection, Application Gateway, Load Balancer, etc.), this approach may for example prevent some DDoS attacks, and ensure load balancing and failover in critical situations. This means that the developers do not need to make custom code for redirections or failover procedures, enabling them to focus their work elsewhere.

Many Azure resources such as Azure Storage, Azure Key Vault, or Azure App service may be put inside a VPN, but are instead accessible from the Internet by default. If all the relevant resources are in a virtual private network and it is reasonable to assume the security of the outside-facing endpoints, the communication inside the network may not be encrypted. This decision should not be taken lightly. The security of the VPN must ensure that there would not be any malicious party inside the network. There are no guarantees about the confidentiality, integrity or authenticity of the messages when using HTTP and therefore there must be high trust between the services in the network. On the other hand, it may be very challenging to properly configure all the certificates to be used in the private network. The decision to do so must be well-documented, and reasons to believe that the network will not be compromised should be provided. It should also be taken into consideration that the service may evolve later and this decision can add to the technological debt.

Another part is data security in transit. All the APIs should enforce secure protocols and refuse any connections via unencrypted channels. This is

especially vital if the exchange may include credentials or personal or confidential information. If that is not the case and for example only static content is being served by the API or web page, it is still inadvisable to use insecure connections, since the communication may be changed by a man-in-the-middle attack. This may lead to providing false information to the end users or even create a threat of phishing some information from users.

The Azure App Service can either handle the certificates and secure communication automatically, or it can be provided with the certificate by the developer. Azure always provides a certificate when the application does not use a custom domain, but that is not a viable solution for most businesses for reasons of brand recognizability and user experience.

On the other hand, a company may already own a certificate and wish to use that certificate in Azure without buying another one from Microsoft. The first option is to use HTTPS implemented by the framework. This is usually simple in code, but the certificate must be provided to the applications directly. It can be either uploaded to Azure Portal and loaded directly into the application (only with App Service with tier Basic or higher running on Windows), or it can be uploaded as a file [77]. Because a certificate is in its nature a secret, it is not advisable to have it as a file along with the source code, as was discussed in Section 4.3. A better option is to have them in Azure Key Vault and access the certificate there. This is again similar to the case with secrets and provides the same benefits and drawbacks, except that the certificates are usually loaded during application start-up and never reloaded. Therefore, the service must be restarted when a certificate changes. This scenario should not occur too often.

Another option is to use SSL termination, where an upstream service handles the secure communication with clients and then relays the messages unencrypted to the intended recipient. This approach assumes that the channel between the SSL termination service and the backend server is entirely safe and there are no eavesdropping agents. We can make these assumptions for example with Azure VPN. Uploading and setting up the rules for redirection and SSL termination can be done with Azure Application Gateway [78] via several tools including Azure CLI and PowerShell. Application Gateway can also work with certificates stored in Azure Key Vault, again providing more straightforward control over them and the ability to use the certificates in multiple services with a single centralized point of configuration.

TLS can also be used for mutual authentication. Then only clients with valid certificates can use the service, and all other attempts to connect are rejected. This type of authentication is currently supported only at the App Service level and it cannot yet be done in upstream services on Azure, such as Application Gateway. This also works only over HTTPS, so if such level of security is desired, it is advisable to completely disable HTTP except for some specific use-cases where it cannot be replaced with the secure protocol.

Conclusion

This work has introduced some basic notions of cloud computing and specifically its use in the Microsoft Azure cloud. It then focused on the security aspects of development there.

One of the notable contributions of this work may be the introduction to some of the Azure services and their security-related capabilities, because much of the information is scattered across multiple documentation pages and some of it may be even missing there and must be instead found out using experiments or by reading the source code. The security features, especially of the Azure Storage and Azure Key Vault, were examined and several principles are commonly found in multiple other Azure services. Some limitations were also found and explained. The reader should now be able to make an informed decision as to what kind of service they want to use based on their specific needs.

Some essential security chellanges during development process were identified and different solutions introduced and explained. Namely operations for secure and appropriate data use and storage were described at length. This topic was tightly coupled with the information obtained in Section 1.3, where the Azure services were introduced and their properties explained.

The work then built on this information to provide fundamental differences between different categories of data based on some properties of the data. Different approaches to handle secrets and other sensitive or confidential information were described. There was also a brief introduction to some aspects the developer or other people responsible for the data security should pay attention to to accurately assess the amount of resources to be spent on the security of different kinds of data.

Related to this is the handling of secrets in the application on the source code level. This topic was examined in two aspects, from the viewpoint of accessing the secrets, and rotating them. In the first case, there was a comprehensive list of solutions shown, and their different advantages and problems explained. The solutions were split first by the system that provides them to

the application and then by the time when the secrets are accessed. This thesis has argued that for most applications a centralized secrets management system is the best solution because of it is clearly separated both from version control systems for source code and is not dependent deployment procedures. The time of secret resolution (fetching it from the secrets management system) is even more dependent on the given application and its use of the secrets, and therefore it is impossible to pick an ideal process for the general case.

Some features specific to .NET Core and ASP.NET Core for handling data were introduced in Section 4.1.1 and Section 4.4. These were also used to illustrate some problems with handling data specific to each language or framework, which the developer must understand. They should also be able to enforce some non-trivial behaviour, such as disposing of sensitive data from memory.

The other part of handling secrets was their rotation, i. e., the process of changing the currently used secret usually including the revocation of the old one, as shown on the example of rotating two interchangeable access keys in Section 4.2. This process was described in depth with possibilities for customization based on different number of keys.

The management of Azure Storage access keys using Azure Key Vault was also introduced. There were some limitations in its implementation found, such as not revoking both keys during the rotation as proposed to have better guarantees about the safety of the keys.

Authentication in Azure was also described at length, specifically in Section 1.3.1 about Azure Active Directory and Section 4.5 about the usage of identity providers and the possibility to use an external one, and what that would bring. The most important part of the former in this context is the review of Azure AD capabilities and protocols used, namely the OAuth 2.0 protocol.

The problem of securing APIs was also addressed, with discussion of why it may be preferred to focus more on security of this service and possibly even invest more resources into it than the other services' individual security. This needs to be evaluated on project-to-project basis. There were mentions of some Azure services, that can potentially be used to provide enhanced security against threats such as a DDoS attack. Different possible scenarios for using a secure channel on an Azure Web App based API were also reviewed and discussed.

Future works in this field could build on the Chapter 1 by reviewing additional Azure resources, may it be Azure SQL Servers and other storage options, or even services on different levels of abstraction, such as Azure Virtual Networks or Virtual Machines, and on the other side management and monitoring tools such as Azure Security Center or Application Insights. Services securing an API such as Azure Application Gateway and Azure Front Door can also be compared, because some of the features they provide are similar and therefore it may be hard to choose the right service for a given scenario. Azure Active

Directory is also very broad topic and it could be examined much more closely with concrete examples of protocol communication. A service could also be created to handle rotation of keys and handing out SAS tokens, not just for Azure Storage, but also for other services such as Event Hub.

Bibliography

[1] Mell, P.; Grance, T. The NIST Definition of Cloud Computing. National Institute of Standards and Technology. Special Publication 800-145, [online], September 2011, [cit. 2019-02-18]. Available from: `https://csrc.nist.gov/publications/detail/sp/800-145/final`

[2] Narumoto, M.; et al. *Moving Applications ot the Cloud*. Microsoft Corporation, second edition, June 2012.

[3] Collier, M.; Shahan, R. *Azure Essentials*. Microsoft Press, 2015, ISBN 978-0-7356-9722-5.

[4] Modi, R. *Azure for Architects*. Packt Publishing, October 2017, ISBN 978-1-78839-739-1.

[5] Microsoft. Azure Portal. [online], [cit. 2019-04-17]. Available from: `portal.azure.com`

[6] Aiello, J.; et al. PowerShell. In: *docs.microsoft.com*, [online], 2018-08-27, [cit. 2019-04-19]. Available from: `https://docs.microsoft.com/en-gb/powershell/scripting/overview?view=powershell-6`

[7] Tramer, S.; et al. Overview of Azure PowerShell. In: *docs.microsoft.com*, [online], 2019-01-10, [cit. 2019-04-19]. Available from: `https://docs.microsoft.com/en-gb/powershell/azure/overview?view=azps-1.7.0`

[8] Aiello, J.; et al. Cmdlet Overview. In: *docs.microsoft.com*, [online], 2016-09-13, [cit. 2019-04-13]. Available from: `https://docs.microsoft.com/en-gb/powershell/developer/cmdlet/cmdlet-overview`

[9] Lamos, B.; et al. Azure REST API Reference. In: *docs.microsoft.com*, [online], 2019-03-26, [cit. 2019-04-19]. Available from: `https://docs.microsoft.com/en-gb/rest/api/azure/`

[10] Simorjay, F. *Shared Responsibilities for Cloud Computing*. Second edition, April 2017.

[11] Taylor, P.; et al. Resource access management in Azure. In: *docs.microsoft.com*, [online], 2019-02-11, [cit. 2019-04-13]. Available from:

https://docs.microsoft.com/en-gb/azure/architecture/cloud-adoption/getting-started/azure-resource-access

[12] Lyon, R.; et al. Tutorial: Create a custom role for Azure resources using Azure PowerShell. In: *docs.microsoft.com*, [online], 2019-02-20, [cit. 2019-04-19]. Available from: https://docs.microsoft.com/en-gb/azure/role-based-access-control/tutorial-custom-role-powershell

[13] Lyon, R. Tutorial: Create a custom role for Azure resources using Azure CLI. In: *docs.microsoft.com*, [online], 2019-02-20, [cit. 2019-04-19]. Available from: https://docs.microsoft.com/en-gb/azure/role-based-access-control/tutorial-custom-role-cli

[14] Drumea, A.; Simons, A.; et al. *Azure Active Directory Data Security Considerations*. First edition, January 2019.

[15] Ross, E.; et al. What is Azure Active Directory? In: *docs.microsoft.com*, [online], 2018-11-13, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/active-directory/fundamentals/active-directory-whatis

[16] de Guzman, C.; et al. Certificate credentials for application authentication. In: *docs.microsoft.com*, [online], 2018-07-24, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/active-directory/develop/active-directory-certificate-credentials

[17] Ross, E.; et al. What is the Azure Active Directory architecture? In: *docs.microsoft.com*, [online], 2019-08-23, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/active-directory/fundamentals/active-directory-architecture

[18] The OAuth 2.0 Authorization Framework. Request for Comments: 6749, [online], October 2012, [cit. 2019-04-14]. Available from: https://tools.ietf.org/html/rfc6749

[19] Sakimura, N.; Bradley, J.; et al. OpenID Connect Core 1.0 incorporating errata set 1. [online], [cit. 2019-04-14]. Available from: https://openid.net/specs/openid-connect-core-1_0.html

[20] Jones, M.; Hardt, D. The OAuth 2.0 Authorization Framework: Bearer Token Usage. Request for Comments: 6750, [online], October 2012, [cit. 2019-04-14]. Available from: https://tools.ietf.org/html/rfc6750

[21] Jones, M.; Bradley, J.; et al. JSON Web Token (JWT). Request for Comments: 7519, [online], May 2015, [cit. 2019-04-14]. Available from: https://tools.ietf.org/html/rfc7519

[22] de Guzman, C.; et al. Microsoft identity platform protocols. In: *docs.microsoft.com*, [online], 2019-04-11, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/active-directory/develop/active-directory-v2-protocols

[23] Denniss, M.; Bradley, D.; et al. OAuth 2.0 Device Authorization Grant. Internet-Draft, expires on 2019-09-12, [online], March 2019, [cit. 2019-04-23]. Available from: https://tools.ietf.org/html/draft-ietf-oauth-device-flow-15

[24] Jones, M. JSON Web Algorithms (JWA). Request for Comments: 7518, [online], May 2015, [cit. 2019-04-14]. Available from: https://tools.ietf.org/html/rfc7518

[25] Baldwin, M.; et al. About keys, secrets, and certificates. In: *docs.microsoft.com*, [online], 2019-01-07, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/key-vault/about-keys-secrets-and-certificates

[26] Shemanske, T. R. *Modern cryptography and elliptic curves: a beginner's guide*, volume 83. Providence, Rhode Island: American Mathematical Society, 2017, ISBN 1470435829;9781470435820;.

[27] Neira, B.; et al. What is Azure Key Vault? In: *docs.microsoft.com*, [online], 2019-01-07, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/key-vault/key-vault-overview

[28] Neira, B.; et al. What is Azure Dedicated HSM? In: *docs.microsoft.com*, [online], 2018-12-07, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/dedicated-hsm/overview

[29] Baldwin, M.; et al. Azure Key Vault soft-delete overview. In: *docs.microsoft.com*, [online], 2019-03-19, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/key-vault/key-vault-ovw-soft-delete

[30] Neira, B.; et al. Azure Key Vault availability and redundancy. In: *docs.microsoft.com*, [online], 2019-01-07, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/key-vault/key-vault-disaster-recovery-guidance

[31] King, J.; et al. Geo-redundant storage (GRS): Cross-regional replication for Azure Storage. In: *docs.microsoft.com*, [online], 2018-10-20, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/common/storage-redundancy-grs

[32] Myers, T.; et al. What is Azure Blob storage? In: *docs.microsoft.com*, [online], 2018-11-19, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/blobs/storage-blobs-overview

[33] Myers, T.; et al. Introduction to Azure Blob storage. In: *docs.microsoft.com*, [online], 2019-01-03, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/blobs/storage-blobs-introduction

[34] Myers, T. What are Azure Queues? In: *docs.microsoft.com*, [online], 2019-02-06, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/queues/storage-queues-introduction

[35] Gunda, S.; et al. Introduction to Table storage in Azure. In: *docs.microsoft.com*, [online], 2018-04-23, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/tables/table-storage-overview

[36] McGee, M.; et al. Design for querying. In: *docs.microsoft.com*, [online], 2018-04-23, [cit. 2019-04-19]. Available from: https://docs.microsoft.com/en-us/azure/storage/tables/table-storage-design-for-query

[37] Shah, R.; et al. What is Azure Files? In: *docs.microsoft.com*, [online], 2018-07-19, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/files/storage-files-introduction

[38] Myers, T.; et al. Overview of Azure Active Directory authentication over SMB for Azure Files (preview). In: *docs.microsoft.com*, [online], 2018-09-19, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/files/storage-files-active-directory-overview

[39] Myers, T.; et al. Azure Storage security guide. In: *docs.microsoft.com*, [online], 2019-03-21, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/common/storage-security-guide

[40] Kasarabada, L.; et al. Azure Storage Service Encryption for data at rest. In: *docs.microsoft.com*, [online], 2018-08-01, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/common/storage-service-encryption

[41] Kasarabada, L.; et al. Storage Service Encryption using customer-managed keys in Azure Key Vault. In: *docs.microsoft.com*, [online], 2018-10-11, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/common/storage-service-encryption-customer-managed-keys

[42] Lanfear, T.; et al. Azure Storage security overview. In: *docs.microsoft.com*, [online], 2019-02-01, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/security/security-storage-overview

[43] Myers, T.; et al. Client-Side Encryption and Azure Key Vault for Microsoft Azure Storage. In: *docs.microsoft.com*, [online], 2017-10-20, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/storage/common/storage-client-side-encryption

[44] Shahan, R.; et al. Constructing an Account SAS. In: *docs.microsoft.com*, [online], 2018-03-21, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/rest/api/storageservices/constructing-an-account-sas

[45] Myers, T.; et al. Constructing a Service SAS. In: *docs.microsoft.com*, [online], 2018-03-21, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/rest/api/storageservices/constructing-a-service-sas

[46] Borthakur, D. HDFS Architecture Guide. The Apache Software Foundation. [online], [cit. 2019-04-14]. Available from: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[47] Foundation, T. A. S. WebHDFS REST API. The Apache Software Foundation. [online], [cit. 2019-04-14]. Available from: https://hadoop.apache.org/docs/r1.0.4/webhdfs.html

[48] Sheth, S.; et al. Best practices for using Azure Data Lake Storage Gen1. In: *docs.microsoft.com*, [online], 2018-06-27, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/data-lake-store/data-lake-store-best-practices

[49] Lin, C. OS and runtime patching in Azure App Service. In: *docs.microsoft.com*, [online], 2018-02-02, [cit. 2019-04-13].

[50] Wiselman, R.; et al. Business continuity and disaster recovery (BCDR): Azure Paired Regions. In: *docs.microsoft.com*, [online], 2018-12-23, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/best-practices-availability-paired-regions

[51] Lin, C.; et al. Authentication and authorization in Azure App Service. In: *docs.microsoft.com*, [online], [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/app-service/overview-authentication-authorization

[52] Vilcinskas, M.; et al. Services that support managed identities for Azure resources. In: *docs.microsoft.com*, [online], 2018-11-28, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/active-directory/managed-identities-azure-resources/services-support-msi

[53] Vilcinskas, M.; et al. What is managed identities for Azure resources? In: *docs.microsoft.com*, [online], 2018-10-23, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/active-directory/managed-identities-azure-resources/overview

[54] Barjtya, S.; Sharma, A.; et al. A detailed study of Software Development Life Cycle (SDLC) Models. *International Journal Of Engineering And Computer Science*, volume 6, July 2017: pp. 22097 – 22100, ISSN 2319-7242, doi: 10.18535/ijecs/v6i7.32.

[55] SDLC Models Explained: Agile, Waterfall, V-Shaped, Iterative, Spiral. [online], [cit. 2019-04-14]. Available from: https://existek.com/blog/sdlc-models/

[56] Hafner, J.; Schwingel, S.; et al. *Azure strategy and implementation guide*. Microsoft Corporation, second edition, 2018. Available from: https://azure.microsoft.com/en-us/resources/azure-strategy-and-implementation-guide/en-us/

[57] Jagli, D. S. CloudSDLC: Cloud Software Development Life Cycle. *International Journal of Computer Applications*, volume 168, June 2017: pp. 6 – 10, doi: 10.5120/ijca2017914468.

[58] Zack, W. H.; Kommalapati, H. The SaaS Development Lifecycle. [online], [cit. 2019-04-14]. Available from: https://www.infoq.com/articles/SaaS-Lifecycle

[59] Mohammed, N. M.; Niazi, M.; et al. Exploring software security approaches in software development lifecycle: A systematic mapping study. *Computer Standards & Interfaces*, volume 50, 2017: pp. 107 – 115, ISSN 0920-5489, doi:10.1016/j.csi.2016.10.001. Available from: http://www.sciencedirect.com/science/article/pii/S0920548916301155

[60] Microsoft. Microsoft Security Development Lifecycle Practices. [online], 2019, [cit. 2019-04-13]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices

[61] Little, C. Why Is There No DevOps Manifesto? In: *DevOps.com Where the World Meets DevOps*, [online], 2016-05-06, [cit. 2019-03-25]. Available from: https://devops.com/no-devops-manifesto/

[62] Dyck, A.; Penners, R.; et al. Towards Definitions for Release Engineering and DevOps. *2015 IEEE/ACM 3rd International Workshop on Release Engineering*, May 2015, ISSN 15378417, doi:10.1109/releng.2015.10.

[63] Jabbari, R.; bin Ali, N.; et al. What is DevOps? A Systematic Mapping Study on Definitions and Practices. *XP '16 Workshops*, May 2016, doi:10.1145/2962695.2962707.

[64] Hüttermann, M. *DevOps for Developers*. Apress, 2012.

[65] Allspaw, J.; Hammond, P. 10+ Deploys per Day. O'Reilly, *Velocity Conference*. Dev & ops cooperation at Flickr, [video of presentation], 2009. Available from: https://www.youtube.com/watch?v=LdOe18KhtT4

[66] Mansfield-Devine, S. DevOps: finding room for security. *Network Security*, volume 2018, no. 7, 2018: pp. 15 – 20, ISSN 1353-4858, doi:10.1016/S1353-4858(18)30070-9. Available from: http://www.sciencedirect.com/science/article/pii/S1353485818300709

[67] Howard, M.; Leblanc, D. E. *Writing Secure Code*. Redmond, Washington, USA: Microsoft Press, second edition, 2002, ISBN 0735617228.

[68] SecureString Class. [online], [cit. 2019-04-20]. Available from: https://docs.microsoft.com/en-us/dotnet/api/system.security.securestring?view=netcore-2.2

[69] Wenzel, M.; Landwerth, I. DE0001: SecureString shouldn't be used. In: *github.com*, [online], 2018-02-16, [cit. 2019-04-20]. Available from: https://docs.microsoft.com/en-us/dotnet/api/system.security.securestring?view=netcore-2.2

[70] Narumoto, M.; et al. Valet Key pattern. In: *docs.microsoft.com*, [online], 2017-06-23, [cit. 2019-05-09]. Available from: https://docs.microsoft.com/en-gb/azure/architecture/patterns/valet-key

[71] Yerramilli, P. Azure Key Vault managed storage account - CLI. In: *docs.microsoft.com*, [online], 2019-03-01, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/key-vault/key-vault-ovw-storage-keys

[72] Baldwin, M.; et al. Azure Key Vault managed storage account - PowerShell. In: *docs.microsoft.com*, [online], 2019-03-01, [cit. 2019-04-13]. Available from: https://docs.microsoft.com/en-gb/azure/key-vault/key-vault-overview-storage-keys-powershell

[73] Narumoto, M.; et al. Leader Election pattern. In: *docs.microsoft.com*, [online], 2017-06-23, [cit. 2019-05-09]. Available from: https://docs.microsoft.com/en-gb/azure/architecture/patterns/leader-election

[74] Kennedy, J.; et al. CNG DPAPI. In: *docs.microsoft.com*, [online], 2018-05-31, [cit. 2019-05-06]. Available from: https://docs.microsoft.com/en-gb/windows/desktop/seccng/cng-dpapi

[75] Narumoto, M.; et al. Federated Identity pattern. In: *docs.microsoft.com*, [online], 2017-06-23, [cit. 2019-05-09]. Available from: https://docs.microsoft.com/en-gb/azure/architecture/patterns/federated-identity

[76] Narumoto, M.; et al. Gatekeeper pattern. In: *docs.microsoft.com*, [online], 2017-06-23, [cit. 2019-05-09]. Available from: https://docs.microsoft.com/en-gb/azure/architecture/patterns/gatekeeper

[77] Lin, C.; et al. Use an SSL certificate in your application code in Azure App Service. In: *docs.microsoft.com*, [online], 2018-12-01, [cit. 2019-05-06]. Available from: https://docs.microsoft.com/en-gb/azure/app-service/app-service-web-ssl-cert-load

[78] Lin, C.; et al. What is Azure Application Gateway? In: *docs.microsoft.com*, [online], 2018-04-30, [cit. 2019-05-06]. Available from: https://docs.microsoft.com/en-gb/azure/application-gateway/overview

[79] Jones, M. JSON Web Key (JWK). Request for Comments: 7517, [online], May 2015, [cit. 2019-04-14]. Available from: https://tools.ietf.org/html/rfc7517

[80] Jones, M.; Bradley, J.; et al. JSON Web Signature (JWS). Request for Comments: 7515, [online], May 2015, [cit. 2019-04-23]. Available from: https://tools.ietf.org/html/rfc7515

[81] Baldwin, M.; et al. Service-to-service authentication to Azure Key Vault using .NET. In: *docs.microsoft.com*, [online], 2019-05-03, [cit. 2019-05-15]. Available from: https://docs.microsoft.com/en-gb/azure/key-vault/service-to-service-authentication

Appendix A

Obtaining JWT

This is an example of how to manually obtain and verify an access token for a virtual machine form Azure AD. The process to obtain a token on behalf of the VM to access Azure Resource Manager will be shown.

A prerequisite is a running virtual machine in Azure with a managed identity. In this example Ubuntu 18.04-LTS will be used but the same can be accomplished on machines with different OS with substitution of some commands (e. g., curl for Bash and Invoke-RestMethod for PowerShell).

The reader is encouraged to experiment along with the text, as some members in several JSON objects may be omitted. The sequence "`[...]`" shall denote omissions of members in JSON objects or lines in strings representing binary data. JSON strings may also be split over multiple lines to fit the page without any indication.

Obtaining the token can be done with a call to a well-known IP address `169.254.169.254`,[35] where AIMS resides. AIMS will make the necessary communication with Azure AD and also provide the client ID and certificate to authenticate the virtual machine.

```
GET /metadata/identity/oauth2/token?
    api-version=2018-02-01&
    resource=https://management.azure.com/
Metadata: true
```

Azure AD will respond with a JSON similar to the following one:

```
{
  "access_token": "<See below>",
  "client_id": "298026b7-6d04-4fdb-94a6-52fa42fd0547",
  "expires_in": "28800",
  "expires_on": "1552698432",
```

[35] It is a non-routable address and can be accessed only from within a VM in Azure.

```
  "ext_expires_in": "28800",
  "not_before": "1552669332",
  "resource": "https://management.azure.com/",
  "token_type": "Bearer"
}
```


The most important part there is the access token, which is shown next. It is important to note that there are three parts separated by a single dot in each case (line breaks are here only for illustration purposes). The three parts separated by dots are the header, payload and signature respectively. Parts of the token are omitted to be more consise. There are twelve and four lines omitted from the payload and signature respectively.


```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik4tbEMwbi05REFM
cXdodUhZbkhRNjNHZUNYYyIsImtpZCI6Ik4tbEMwbi05REFMcXdodUhZbkhR
NjNHZUNYYyJ9
.
eyJhdWQiOiJodHRwczovL21hbmFnZW1lbnQuYXp1cmUuY29tLyIsImlzcyI6
[...]
b21wdXRlL3ZpcnR1YWxNYWNoaW5lcy9tbGFkZWRhdi1icC12bSJ9
.
tPWr7OKWL0YwDPV3NNHvDxCEK_tWXWp5l9BFhIs-naGcBae5PcfElG4StTwL
[...]
woimhq0l7dQCI6Q62m1peTte--Aa-4q7nIZg-lFEGw
```


Each part can is encoded in Base64 and the header and payload can be decoded to these two JSON objects:


```
{
  "typ": "JWT",
  "alg": "RS256",
  "x5t": "N-lC0n-9DALqwhuHYnHQ63GeCXc",
  "kid": "N-lC0n-9DALqwhuHYnHQ63GeCXc"
}

{
  "aud": "https://management.azure.com/",
  "iss": "https://sts.windows.net/
          f345c406-5268-43b0-b19f-5862fa6833f8/",
  "iat": 1552669332,
  "nbf": 1552669332,
  "exp": 1552698432,
[...]
  "ver": "1.0",
  "xms_mirid": "/subscriptions/
```

```
                c02c846b-e799-4497-995d-184229394101/
                resourcegroups/mladedav-bp-tests/
                providers/Microsoft.Compute/
                virtualMachines/mladedav-bp-vm"
}
```

To verify the integrity and authenticity of the token, a call must be made to the authorization server for Azure AD to obtain the certificate of the key that signed the JWT. It resides at `https://login.microsoftonline.com/`. As per OpenID Connect documentation, the configuration document must be at accessible at an address obtained by concatenating the URL of the OpenID issuer location and `/.well-known/openid-configuration`. For Azure AD there is an issuer for each tenant, e. g., the issuer for the CTU tenant issuer resides at `https://login.microsoftonline.com/f345c406-5268-43b0-b19f-5862fa6833f8/`. If we then make a call there, we receive the configuration JSON, where in the member "jwks_uri" we can find the URI, which identifies the location of the certificates to verify the signature.

At this address we can find an array of JSON Web Key (JWK) structures [79] from which we can either extract the X.509 certificate or in case of RSA the public key directly.[36]

```
GET /f345c406-5268-43b0-b19f-5862fa6833f8/
    .well-known/openid-configuration

{
  "authorization_endpoint": "https://login.microsoftonline.com/
                            f345c406-5268-43b0-b19f-5862fa6833f8/
                            oauth2/authorize",
  "token_endpoint": "https://login.microsoftonline.com/
                    f345c406-5268-43b0-b19f-5862fa6833f8/
                    oauth2/token",
  "token_endpoint_auth_methods_supported": [
    "client_secret_post",
    "private_key_jwt",
    "client_secret_basic"
  ],
  "jwks_uri": "https://login.microsoftonline.com/
               common/discovery/keys",
[...]
  "id_token_signing_alg_values_supported": [
    "RS256"
  ],
```

[36]The member "e" denotes exponent and "n" modulus. In both cases the numbers are Base64 encoded.

```
[...]
  "end_session_endpoint": "https://login.microsoftonline.com/
                           f345c406-5268-43b0-b19f-5862fa6833f8/
                           oauth2/logout",
  "response_types_supported": [
    "code",
    "id_token",
    "code id_token",
    "token id_token",
    "token"
  ],
[...]
}
```

We can then follow to the URI in the "jwks_uri" member, i. e., `https://login.microsoftonline.com/common/discovery/keys`. There we find all the keys currently in use.

```
{
  "keys": [
    {
      "kty": "RSA",
      "use": "sig",
      "kid": "N-lC0n-9DALqwhuHYnHQ63GeCXc",
      "x5t": "N-lC0n-9DALqwhuHYnHQ63GeCXc",
      "n": "t3J1hnS4aRZaZGq5JUw1iKsHynCUV9lMBe2MDArXGeQlN-w8
            Xw9vU6InqmPVvJsUVyUkKE0jzn4dYLcwbTuttQ0hmN-lzNfG
            ol04KKMIVdtTs1P0wo_-VyJ88EuWM3lvDxyTw1PLim14UJ18
            56zdp2_kZLOSy-B46K96ENJ8b2yCP_VHRTd3GgNTrx-xeU66
            WJdlon6SSkxI85KIAzOR4vxrl2XZZx_DkVcsAHa8KXQRkbMw
            82F2SHAbgJTv8qjSHR_WXjoGs3Wgds9UUqgNDXSK6qTjoG53
            zj8-faRkK0Px4wRD9rVXt-pPcGaul3TEkUVhpe8SyrLWETFe
            xJesSQ",
      "e": "AQAB",
      "x5c": [
        "MIIDBTCCAe2gAwIBAgIQP8sUV4hf2ZxPfw5DB0O9CjANBgkqhki
        G9w0BAQsFADAtMSswKQYDVQQDEyJhY2NvdW50cy5hY2Nlc3Njb25
        0cm9sLndpbmRvd3MubmV0MB4XDTE5MDIwMTAwMDAwMFoXDTIxMDI
        wMTAwMDAwMFowLTErMCkGA1UEAxMiYWNjb3VudHMuYWNjZXNzY29
        udHJvbC53aW5kb3dzLm5ldDCCASIwDQYJKoZIhvcNAQEBBQADggE
        PADCCAQoCggEBALdydYZ0uGkWWmRquSVMNYirB8pwlFfZTAXtjAw
        K1xnkJTfsPF8Pb1OiJ6pj1bybFFclJChNI85+HWC3MG07rbUNIZj
        fpczXxqJdOCijCFXbU7NT9MKP/lcifPBLljN5bw8ck8NTy4pteFC
        dfOes3adv5GSzksvgeOivehDSfG9sgj/1ROU3dxoDU68fsXlOuli
```

```
        XZaJ+kkpMSPOSiAMzkeL8a5dl2Wcfw5FXLAB2vCl0EZGzMPNhdkh
        wG4CU7/Ko0h0f1l46BrN1oHbPVFKoDQ10iuqk46Bud84/Pn2kZCt
        D8eMEQ/a1V7fqT3Bmrpd0xJFFYaXvEsqy1hExXsSXrEkCAwEAAaM
        hMB8wHQYDVR0OBBYEFH5JQzlFI3FE9VxkkUbFT9XQDxifMA0GCSq
        GSIb3DQEBCwUAA4IBAQCb7re2PWF5ictaUCi4Ki2AWE6fGbmVRUd
        f0GkI06KdHWSiOgkPdB7Oka1Fv/j4GCs/ezHa1+oAx8uU96GECBB
        EMnCYPqkjmNKdLYkIUrcwEe9qz12MOCKJkCuYsDdLUqv+e4wHssb
        AnJn2+L13UmfAb6FM1VTaKIQtPs4yZsdhnk4M+Ee2EpcvgwOl2na
        +m58ovspieEyI6II/TolzwP9NWbvHw5VlF0IYttQprjmQU3tQ2E6
        j3HpZ31B0nrnFWglUB7lEC+0mkyJUGzovNECsr+BIEMhTlCp2/rb
        ruCCbZBppYAlbWlTFwXA8TqfE4DNATYgm90ObQANcTnHJeRV1"
      ]
    },
    {
      "kty": "RSA",
      "use": "sig",
      "kid": "HBxl9mAe6gxavCkcoOU2THsDNa0",
      "x5t": "HBxl9mAe6gxavCkcoOU2THsDNa0",
      "n": "0afCaiPd_xl_ewZGfOkxKwYPfI4Efu0COfzajK_gnviWk7w3
[...]
           fZr4MQ",
      "e": "AQAB",
      "x5c": [
        "MIIDBTCCAe2gAwIBAgIQWcq84CdVhKVEcKbZdMOMGjANBgkqhki
[...]
        ynYmkl89MleOfKIojhrGRxryZG2nRjD9u/kZbPJ8e3JE9px67"
      ]
    },
    {
      "kty": "RSA",
      "use": "sig",
      "kid": "M6pX7RHoraLsprfJeRCjSxuURhc",
      "x5t": "M6pX7RHoraLsprfJeRCjSxuURhc",
      "n": "xHScZMPo8FifoDcrgncWQ7mGJtiKhrsho0-uFPXg-OdnRKYu
[...]
      oftfpWr3hFRdpxrwuoQEO4QQ",
      "e": "AQAB",
      "x5c": [
        "MIIC8TCCAdmgAwIBAgIQfEWlTVc1uINEc9RBi6qHMjANBgkqhki
[...]
        wP6cOzgZpjdPMwaVt5432GA=="
      ]
    }
  ]
```

```
}
```

We find the correct key by matching the kid member in the header of the access token obtained at the beginning. In our case it is the first key. We can take the appropriate x5c member and decode it as a X.509 certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3f:cb:14:57:88:5f:d9:9c:4f:7f:0e:43:07:43:bd:0a
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=accounts.accesscontrol.windows.net
        Validity
            Not Before: Feb  1 00:00:00 2019 GMT
            Not After : Feb  1 00:00:00 2021 GMT
        Subject: CN=accounts.accesscontrol.windows.net
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                  00:b7:72:75:86:74:b8:69:16:5a:64:6a:b9:25:4c:
[...]
                  c4:91:45:61:a5:ef:12:ca:b2:d6:11:31:5e:c4:97:
                  ac:49
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                7E:49:43:39:45:23:71:44:F5:5C:
                64:91:46:C5:4F:D5:D0:0F:18:9F
    Signature Algorithm: sha256WithRSAEncryption
         9b:ee:b7:b6:3d:61:79:89:cb:5a:50:28:b8:2a:2d:80:58:4e:
[...]
         c9:79:15:75
```

We can see that it is a self-signed certificate. We can then use the certificate to check for validity of the original access token. That can be done either online on a site such as `https://jwt.io` or manually, as further described in [80].

Appendix B

Code Examples

Managed Identities

This example application lists all secrets and their values currently present in a given Key Vault.

Authorization on Azure is proven by presenting an access token. When a developer wants to obtain these tokens in C# code, the easiest way is with an `AzureServiceTokenProvider`. This class deduces the environment it runs in and choses an appropriate method to communicate with Azure AD. It first tries the AIMS endpoint similarly as in Appendix A. If this does not work, `AzureServiceTokenProvider` can asses that the application is not running in Azure and proceeds to try to communicate with Azure AD on behalf of the developer. It first tries to use App Authentication extension for Visual Studio, then the Azure CLI. This behaviour can be overrident by specifying a connection string in the constructor, or providing it through the `AzureServicesAuthConnectionString` environment variable [81].

A `KeyVaultClient` is then instantiated. It performs all the necessary cryptographic and vault operations against Azure Key Vault. It needs an authentication callback, which can be build with the `AzureServiceTokenProvider` or created by the developer.

```
1 var azureServiceTokenProvider = new AzureServiceTokenProvider();
2 var keyVault = new KeyVaultClient(
3     new KeyVaultClient.AuthenticationCallback(
4         azureServiceTokenProvider.KeyVaultTokenCallback
5     )
6 );
```

It is then very easy to access the secrets in Azure Key Vault, based on the Key Vault URI and the secret name. Accessing other Key Vault objects would be similar.

```
var secretBundle = await keyVault.GetSecretAsync(keyVaultUri,
    secretName);
var secretValue = secretBundle.Value;
```

It is also important to note, that the Azure App Service must be given a managed identity and access to the Key Vault's secrets. This can be done for example via PowerShell. The same access policies must be also set for developers, if the same Key Vault instance is to be used for development and testing.

```
# Create a web app.
New-AzWebApp `
    -ResourceGroupName $ResourceGroup `
    -AppServicePlan $WebAppName `
    -Name $WebAppName `
    -Location $Location

# Use system-assigned managed identity
Set-AzWebApp `
    -ResourceGroupName $ResourceGroup `
    -Name $WebAppName `
    -AssignIdentity $true

# Get objectId
$ServicePrincipal = Get-AzADServicePrincipal `
    -DisplayName $WebAppName

# Set policy to access the Key Vault
Set-AzKeyVaultAccessPolicy `
    -ResourceGroupName $ResourceGroup `
    -VaultName $KeyVault `
    -ObjectId $ServicePrincipal.Id `
    -PermissionsToSecrets list,get
```

Secret Resolving

This application shows four secrets obtained in four different scenarios.

This example illustrates different strategies of accessing secrets. All this information is only complementary to the information in Section 4.3.

First, configuration secret is loaded from the `appsettings.json` into the `SecureOptions` class, which will be passed through dependency injection to any class or method requiring the configuration it contains.

The deployment secret is passed in the same manner, but it was not manually inserted into the configuration file. It was added during the deployment by the `deploy.ps1` PowerShell script. Both this and the configuration secret can be changed only by recompiling and redeploying the application.

Next there are the bootstrap and just-in-time secrets, which are both fetched from Azure Key Vault. The difference between those is the time of resolution. The secrets are obtained in the same manner as in the last example with managed identities.

While the bootstrap secret is fetched in a `PostConfigure` callback and saved to the same class as the other secrets fetched during startup, the just-in-time secret is fetched when requested by the user. This has consequence that while the former is static until the application is restarted, the latter can be changed at any time and fetched by refreshing the page.

ASP.NET Core Data Protection API

This application saves user's input using the Data Protection API and shows all the saved data to the user in both encrypted and unencrypted form.

The keys to Data Protection API must be saved in a shared location accessible by all instances of the service. In Azure, they can be saved in a Blob, ideally limiting the read and write access to only the applicatoin. To enhance their security further, the keys (specifically the master keying material containing the entropy) may be protected with an Azure Key Vault. These operations are requested in code at the time of registration of the Data Protection API to the services to be later injected.

```
1 services.AddDataProtection()
2     .PersistKeysToAzureBlobStorage(blob)
3     .ProtectKeysWithAzureKeyVault(keyVault, keyIdentifier);
```

We have seen in the first example how to obtain a `KeyVaultClient` instance to be passed to the `ProtectKeysWithAzureKeyVault` method. The other parameter is the full identifier of the key, including version and can be found in the Azure Portal or inquired using PowerShell.

Giving access to the Blob can be done in one of several ways. Either a SAS token can be provided or access can be granted through instances already provided with means for authentication. In all cases it is important to think about the possibility that the storage account keys may be rotated, rendering these classes unable to access the Blob with the keys. The same applies to the SAS tokens, because they are valid only as long as the key used for their signing is.

Because of these issues, the code uses managed identity for Blob storage, which is in public preview at the time of writing this thesis. First,

we use an `AzureServiceTokenProvider` instance to obtain a token for accessing the storage. Then a `TokenCredential` instance is constructed with the obtained token,[37] a callback for token renewal,[38] and the frequency at which new tokens should be acquired. This class can then be used to create `StorageCredentials`, which along with a URI specifying the Blob can be in turn used for constructing a `CloudBlockBlob`. This is the Blob that is then used to store the encrypted keys.

```
const string StorageResource = "https://storage.azure.com/";
AppAuthenticationResult authResult = await azureServiceTokenProvider
    .GetAuthenticationResultAsync(StorageResource);

TimeSpan frequency = authResult.ExpiresOn - DateTimeOffset.UtcNow;
TokenCredential tokenCredential = new TokenCredential(
                                    authResult.Token,
                                    TokenRenewerAsync,
                                    azureServiceTokenProvider,
                                    frequency);

StorageCredentials storageCredentials = new StorageCredentials(
    tokenCredential);
CloudBlockBlob blob = new CloudBlockBlob(new Uri(blobName),
    storageCredentials);
```

After this initial configuration, the other classes and methods can request the `IDataProtectionProvider` interface through dependency injection. They can then create a `IDataProtector` with a purpose string, which can then be used to protect and unprotect data.

```
IDataProtector protector = provider.CreateProtector("index");
string encryptedText = protector.Protect(plaintext);
string decryptedText = protector.Unprotect(encryptedText);
```

The minimal example application uses an in-memory database and only stores only the encrypted text. It is decrypted every time the page is requested for viewing.

The application must also be given access to the wrap and unwrap operations on keys in Key Vault, as well as read and write access to the Blob containing the keys.

[37] In the example code, it is acquired through the callback function mentioned next.

[38] The callback is passed current state as an argument. The same state class is also passed to the `TokenCredential` constructor.

OAuth 2.0 Device Grant

This application obtains an access token from Azure AD.

This is a console application and is not supposed to be run in the cloud. It instead obtains an access token from Azure AD using the device grant.

First we need to create a class representing the application. This class needs to know the ApplicationId obtained during application registration in Azure AD and TenantId of the application's home tenant.

```
// The TenantId can be found in the Azure Portal
string tenantId = "ffbc9686-2bf6-496f-acf8-f5a61907ffc2";
// The ApplicationId can be obtained in the Azure Portal
string clientId = "dde72f2b-ed50-4d64-8339-50cedc4b94fe";
IPublicClientApplication app = PublicClientApplicationBuilder
        .Create(clientId)
        .WithAuthority(AzureCloudInstance.AzurePublic, tenantId)
        .Build();
```

Then tokens based on desired scopes should be first looked up in cache and if there is one that would have been sufficient for the desired access, it will be used. In our scenario this can never happen, but the code is still present for educational purposes. First we obtain a list of all users whose tokens are in the cache and then inquire about present tokens for each of them. In real scenario we would not do it this way, because we could misuse other user's cached tokens easily. There is also an overload that accepts username as a parameter to specify whose token we are looking for. If there is no sufficient token available in the cache for given scopes and account, an exception will be thrown to indicate the result.

```
var accounts = await app.GetAccountsAsync();
foreach (var account in accounts)
{
    try
    {
        result = await app.AcquireTokenSilent(scopes, account)
            .ExecuteAsync();
    }
    catch (MsalUiRequiredException)
    {
        // This indicates that there is not a token present in cache
    }
}
```

Finally, we can try to acquire the token. We must provide the desired scopes as well as a callback function, which will provide the user with infor-

mation about how to authenticate, namely the verification uri and user code to enter there. Then the library polls the token endpoint asynchronously until the user successfully authenticates and the token is obtained, or the protocol times out.

```
result = await app.AcquireTokenWithDeviceCode(scopes,
    deviceCodeCallback =>
    {
        Console.WriteLine(deviceCodeCallback.Message);
        return Task.FromResult(0);
    }).ExecuteAsync();
```

The example code also includes an example with a resource owner password credentials grant, but it is for educational purposes only. It should not be used, because the application should not handle user's credentials directly and other flows are generally a better choice. This flow also cannot be used for Microsoft Accounts, but only for company or education accounts.

In both cases, the accounts must be registered in the tenant, whose TenantId is provided in the code.

Managing Storage Access Keys with Azure Key Vault

This example is infrastructure-as-a-code in PowerShell script form. It configures a Key Vault to manage an Azure storage account's and regenerate them periodically. It also issues SAS tokens.

A thing to note is the static ApplicationId of Key Vault. All Azure Key Vaults in tenants in the public cloud share this AplicationId (it is different in government clouds).

First the resource group, Azure Key Vault and Azure storage account are provisioned. Then Azure Key Vault application is granted the role of "Storage Account Key Operator Service Role", letting it read and regenerate keys, but nothing else. This also indirectly provides access to the data, because the keys can be used for data operations in the storage account.

```
New-AzRoleAssignment `
    -ApplicationId 'cfa8b339-82a2-471a-a3c9-0fc0be7a4093' `
    -RoleDefinitionName 'Storage Account Key Operator Service Role' `
    -Scope $storageAccount.Id
```

Then the Azure storage account an Key Vault may be connected so that the Key Vault can regenerate the keys based on a regular basis as set by the

last parameter. It can also be omitted and the Key Vault will then regenerate keys only when requested.

```
Add-AzKeyVaultManagedStorageAccount `
    -VaultName $KeyVaultName `
    -AccountName $StorageAccountName `
    -AccountResourceId $storageAccount.Id `
    -ActiveKeyName "key1" `
    -RegenerationPeriod $RegenerationPeriod
```

A template SAS token must be then created and provided to the Key Vault. All the tokens handed out by the Key Vault will be based on this template, although some fields are incurred from the parameters, such as services and permissions. We can then obtain new SAS tokens from the Key Vault as if they were standard secrets.

```
$storageContext = New-AzStorageContext `
    -StorageAccountName $storageAccount.StorageAccountName `
    -Protocol Https `
    -StorageAccountKey Key1
$start = [System.DateTime]::Now.AddDays(-1)
$end = [System.DateTime]::Now.AddMonths(1)
$templateSas = New-AzStorageAccountSasToken `
    -Service blob,file,Table,Queue `
    -ResourceType Service,Container,Object `
    -Permission "racwdlup" `
    -Protocol HttpsOnly `
    -StartTime $start `
    -ExpiryTime $end `
    -Context $storageContext

$sasDefinition = Set-AzKeyVaultManagedStorageSasDefinition `
    -AccountName $storageAccount.StorageAccountName `
    -VaultName $keyVault.VaultName `
    -Name $SasDefinitionName `
    -TemplateUri $templateSas `
    -SasType 'account' `
    -ValidityPeriod $SasValidityPeriod

Get-AzKeyVaultSecret `
    -VaultName $keyVault.VaultName `
    -Name $sasDefinition.Sid.Substring($sasDefinition.Sid.LastIndexOf
        ('/')+1)
```

Appendix C

Acronyms

AAD Azure Active Directory.

ACL Access Control List.

AD Active Directory.

AES Advanced Encryption Standard.

AIMS Azure Instance Metadata Service.

API Application Programming Interface.

ARM Azure Resource Manager.

ASM Azure Service Manager.

Bash Bourne Again Shell.

Blob Binary Large object.

CBC Cipher Block Chaining.

CLI Command Line Interface.

CRUD Create, Retrieve, Update, Delete.

CSP Cloud Service Provider.

CTS Ciphertext Stealing.

DDoS Distributed Denial of Service.

DPAPI Data Protection Application Programming Interface.

DREAD Damage, Reproducibility, Exploitability, Affected, Discoverability.

ECC Elliptic-curve cryptography.

FIPS Federal Information Processing Standard.

GB Gigabyte.

HDFS Hadoop Distributed File System.

HMAC Hash-Based Message Authentication.

HSM Hardware Security Module.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IaaS Infrastructure as a Service.

IAM Identity and Access Management.

ID Identifier.

IETF Internet Engineering Task Force.

IP Internet Protocol.

IT Information Technology.

IV Initialization Vector.

JSON JavaScript Object Notation.

JWK JSON Web Key.

JWT JSON Web Token.

kB Kilobyte.

LTS Long Term Support.

MB Megabyte.

MFA Multi-Factor Authentication.

MSI Managed Service Identity.

OS Operating System.

PaaS Platform as a Service.

POSIX Portable Operating System Interface.

PRNG Pseudorandom Number Generator.

RBAC Role-Based Access Control.

REST Representational State Transfer.

RSA Rivest, Shamir, & Adleman.

SaaS Software as a Service.

SAS Shared Access Signature.

SD Secure Digital.

SDK Software Development Kit.

SDL Security Development Lifecycle.

SDLC Software Development Life Cycle.

SHA Secure Hash Algorithm.

SMB Server Message Block.

SQL Structured Query Language.

SSE Storage Service Encryption.

SSL Secure Sockets Layer.

SSO Single Sign-On.

TLS Transport Layer Security.

UI User Interface.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

UTF-8 Unicode Transformation Format — 8-bit.

VHD Virtual Hard Disk.

VM Virtual Machine.

VPN Virtual Private Network.

XML Extensible Markup Language.

Appendix D

Contents of Enclosed SD Card

- `readme.txt` ..... The file with SD card contents description
- `src` ..... The directory with all source code
  - `examples` ..... The directory with source code of examples
    - `examples.txt` ..... The file with examples usage instructions
    - `managed-identities` ..... Using managed identities
      - `key-vault-browser` ..... ASP.NET Core source code
      - `deploy.ps1` ..... PowerShell script used to deploy the example
    - `secret-resolving` ..... SSecret resolution strategies
      - `secret-resolver` ..... ASP.NET Core source code
      - `deploy.ps1` ..... PowerShell script used to deploy the example
    - `data-protection` ..... ASP.NET Core Data Protection API
      - `data-protection` ..... ASP.NET Core source code
      - `deploy.ps1` ..... PowerShell script used to deploy the example
    - `oauth` ..... The example of the OAuth 2.0 Device grant
    - `key-vault-managed-storage` . Key Vault managed storage account
      - `keyvault-sas.ps1` ..... The infrastructure-as-a-code
  - `thesis` ..... The directory with LaTeX source code of the thesis
- `text` ..... The thesis text directory
  - `thesis.pdf` ..... The thesis text in PDF format