Bachelor's thesis

Information and Communications Technology

2024

Teemu Rintala

Migrating a Java Application to Microsoft Azure

– simplifying maintenance and scalability by migrating to a cloud-based solution

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2024 | 34 pages

Teemu Rintala

Migrating a Java Application to Microsoft Azure

- simplifying maintenance and scalability by migrating to a cloud-based solution

The increasing demand for scalable and flexible cloud solutions has driven many organizations to consider migrating their existing infrastructure to cloud platforms.

This thesis aimed to explore the process of migrating a Java-based application to Microsoft Azure with the help of Azure-managed services, focusing on the challenges and best practices involved in this transition. The primary objective was to understand the steps for a successful migration and the benefits that can be achieved, such as simplified maintenance and improved scalability. The results discussed in this thesis have been derived from testing done in a quality assurance environment, as the production environment had not yet been fully migrated to Azure.

The results from the migration were promising, with the application showing improved scalability, allowing it to adapt based on user demand without compromising performance. The use of Azure-managed services reduced the overhead of maintaining infrastructure. Challenges such as migrating database relations and integrations with third-party services required specific attention and were resolved through detailed planning and thorough testing.

Keywords:

Java, software architecture, cloud services, servers, database programs, system administrators, quality assurance, Azure

Opinnäytetyö AMK | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2024 | 34 sivua

Teemu Rintala

Java-sovelluksen siirtäminen Microsoft Azureen

- ylläpidon ja skaalautuvuuden yksinkertaistaminen pilvipohjaisen ratkaisun avulla

Kasvava tarve skaalautuville ja joustaville pilviratkaisuille on saanut monet organisaatiot harkitsemaan jo olemassa olevan infrastruktuurin siirtämistä pilvipalveluntarjoajien alustoille.

Tämän opinnäytetyön tarkoituksena oli tutkia prosessia, jossa Java-pohjainen sovellus siirretään Microsoft Azure -pilvipalveluun hyödyntämällä Azuren hallinnoituja palveluita keskittyen siirtymään liittyviin haasteisiin ja parhaisiin käytäntöihin. Päätavoitteena oli ymmärtää onnistuneen siirtymisen vaiheet ja saavutettavat edut, kuten yksinkertaistettu ylläpito ja parantunut skaalautuvuus. Tässä opinnäytetyössä käsitellyt tulokset on johdettu työn aikana tehdystä testauksesta, koska tuotantoympäristöä ei ollut vielä täysin siirretty Azureen.

Migraation tulokset olivat lupaavia, ja sovellus osoitti parantunutta skaalautuvuutta, jolloin se pystyi mukautumaan käyttäjäkysyntään ilman suorituskyvyn heikkenemistä. Azuren hallittujen palveluiden käyttö vähensi infrastruktuurin ylläpidon kuormitusta. Haasteet, kuten tietokantarakenteen muokkaaminen ja kolmannen osapuolen palveluiden integrointi, vaativat erityistä huomiota, ja ne ratkaistiin huolellisella suunnittelulla ja perusteellisella testaamisella.

Asiasanat:

Java, ohjelmistoarkkitehtuuri, pilvipalvelut, palvelimet, tietokantaohjelmat, pääkäyttäjät, laadunvarmistus, Azure

Contents

Figures

Pictures

Tables

List of abbreviations

AD	Active Directory
API	Application Programming Interface
AWS	Amazon Web Services
CD	Continuous Delivery
CI	Continuous Integration
CLI	Command-Line interface
DI	Dependency Injection
DMZ	Demilitarized Zone
GCP	Google Cloud Platform
HTTP	Hypertext Transfer Protocol
LDAP	Lightweight Directory Access Protocol
MVC	Model–view–controller
OIDC	OpenID Connect
QA	Quality Assurance
REST	Representational State Transfer
SSH	Secure Shell Protocol
VPN	Virtual private network
XML	Extensible Markup Language

1 Introduction

The rapid growth of cloud computing has transformed the way organizations develop, deploy, and maintain software. With the promises of simplified maintenance, enhanced scalability, and cost reductions, cloud-based solutions such as Microsoft Azure have become increasingly popular. The possibility to configure all services from a web user interface helps with unifying the configurations across different environments and applications.

The thesis commissioner, ATR Soft, has an on-premises infrastructure that is used to run business critical services. Maintaining these services requires considerable resources that could be utilized for tasks that generate revenue. The responsibility of general server management is transferred to Azure by utilizing managed services to simplify the maintenance and to reduce the resources used.

The goal of this thesis is to migrate a Java-based application called ATR Works from on-premises to Microsoft Azure. The migration process includes modifying the application to run as an Azure App Service, incorporating authentication using Microsoft Entra ID, and ensuring seamless data migration to Azure SQL Database. The current implementation is only accessible from the local network, meaning that the users must use a Virtual Private Network (VPN) connection to access the application outside the local network.

To make it possible for employees to mark hours without using a VPN to connect to the local network, a stripped-down version of ATR Works was created. This React application is called ATR Works Extension, and it runs on-premises on a demilitarized zone (DMZ) server, meaning that it is accessible from the Internet. Part of the migration process is to simplify the infrastructure by retiring the ATR Works Extension, as it will not be needed after the core application is hosted on Azure.

This thesis is structured as follows: Chapter 1 introduces the basis of the migration process, plan of implementation, and the goals. Chapter 2 presents

the methods and technologies used during the migration process. Chapter 3 lists the requirements for the project. Chapter 4 explains the starting architecture, the wanted cloud-based architecture, and justifies the used services and resources. Chapter 5 goes through the migration process and explains the steps taken to have the application running on Microsoft Azure. Chapter 6 describes conclusions of the migration process and possible future improvements.

2 Methods & Technologies

In this chapter, we will explore the various methods and technologies used to accomplish the migration of ATR Works to Microsoft Azure. When working with cloud-based solutions, the specific methods and technologies differ between service providers. Most of the Azure-specific tools listed here have equivalent counterparts from Google Cloud Platform (GCP) and Amazon Web Services (AWS).

2.1 Microsoft Azure

Microsoft Azure is a cloud computing service provided by Microsoft. It offers various products and services such as Windows and Linux virtual machines, fully managed services, and databases. As Azure was created by Microsoft, it is usually easier to integrate with other Microsoft software. For example, adding a custom application to the company's Office 365 application menu is a simple on-off switch. [1]

There is intense competition in the cloud computing services business and the primary competitors for Azure are AWS and GCP. In 2023 AWS has a market share of around 33%, Azure 23%, and GCP 11%. Overall, the use of cloud computing services for data storage has grown from 30% in 2015 to 60% in 2022. [2]

2.2 Microsoft Azure Portal

Microsoft Azure Portal is a web-based management portal provided by Microsoft for managing Azure cloud services. This portal serves as a central hub for administrators, developers, and operators, to configure, deploy, and monitor Azure resources and services. [3]

2.3 Microsoft Azure CLI

Microsoft Azure Command-Line-Interface (CLI) is a cross-platform command-line tool provided by Microsoft for executing administrative tasks on Azure resources and services. It is possible to install the tool locally and run it from the operating systems shell or use it fully utilizing Azure Cloud Shell through a web browser. [4]

2.4 Microsoft Entra ID

Microsoft Entra ID, formerly known as Azure Active Directory (Azure AD), is an identity and access management (IAM) solution provided by Microsoft. It works to secure access to various resources across cloud and on-premises environments. Entra ID integrates with third-party applications and other Microsoft services like Microsoft 365 and Azure Postal. [5]

2.5 Apache Maven

Apache Maven is a build automation and project management tool created for Java-based applications. It aims to make the build process easy, provide a uniform build system, provide quality project information, and encourage better development practices. These objectives are fulfilled with the help of an XML based project object model (POM) file. This file includes detailed information about the project, dependencies, and possible Maven plugins used. [6]

2.6 Java Spring Framework

The Java Spring Framework is an open-source framework for building enterprise-level Java-based applications. It provides a robust infrastructure for developing Java applications, allowing the developers to focus on business logic while the framework handles configuration, security, and integration. Spring’s core features include dependency injection (DI), aspect-oriented

programming (AOP), and a wide ecosystem that supports web applications, data access, and cloud-native development.

2.7 Liquibase

Liquibase is a database schema management solution that provides the possibility to unite an application with the database schema. If the database schema is modified a new changeset is created, added to the application's version control, and when Liquibase runs, the changes are made to the database schema. This ensures that the application is always running against the correct database schema and removes the need for updating the database schema by hand. [7]

3 Requirements for the project

There were three different types of requirements for this project, hosting specific that were in our case limited to the Microsoft Azure and Microsoft Entra ID platforms, modifications to the Java-based application, and general requirements that are not tied to any platform of software.

3.1 Hosting requirements

The application along with the surrounding infrastructure was hosted on on-premises servers. After the migration the application must be migrated to Microsoft Azure and all on-premises dependencies should be removed. If the application has on-premises dependencies that cannot be removed, they should also be migrated or replaced with cloud-based solutions. The performance of the application must at least stay the same and the possibility to scale the application should be added.

All identity and access management (IAM) configuration must be transferred to Microsoft Entra ID and signing in to the application will happen with the Entra ID credentials. The roles for the application must be automatically mapped from configured groups and only the users inside of these groups can sign in to the application.

Maintaining the application must be simplified by reducing the amount of recurring maintenance tasks and should take up less time. Monitoring all the application resources should be possible from a single location e.g. from the Microsoft Azure Portal. The possibility to add alerts for evets such as high CPU usage could be added.

3.2 Application requirements

Signing into the application must be possible using Microsoft Entra ID credentials without the on-premises Active Directory (AD). Application specific

privileges must be mapped to users during authentication based on the roles assigned in Entra ID.

The application database will be migrated to work with a user ID and email combination instead of a username. Changing the email address of a user must be possible and should only require changing a single value in the database.

Features from ATR Works Extension that are missing from ATR Works will be implemented as new features. These features include a survey management page and adding bookings for multiple days using a date range.

The on-premises version of ATR Works has a CI/CD pipeline that uses TeamCity and Octopus. These tools will not be migrated to Microsoft Azure. Deploying the application to Microsoft Azure must be possible but can involve more manual steps compared to the on-premises solution.

3.3 General requirements

The application is a crucial part of the commissioner’s daily work, and it must stay functional during work hours. Accessing the application was previously not possible outside of the on-premises network without a VPN connection, this must be possible after the migration. All business-critical Power-BI reports must stay functional during and after the migration. The Netvisor integration should be migrated to run on Microsoft Azure.

4 Architecture

In this chapter we are focusing on the application architecture and the surrounding infrastructure. Comparing how the planned infrastructure for Microsoft Azure differs from the one running on-premises. This chapter justifies the required modifications made to the application architecture and the Microsoft Azure services picked to achieve the wanted result. The on-premises infrastructure was used as reference for planning, but the end goal was to create a more cloud optimized solution.

The result includes two separate environments running on Microsoft Azure, one for production and the other for quality assurance (QA). For development, it was made possible to run the application outside of Microsoft Azure, with different levels of separation from the cloud. This separation not only speeded up the development, as the developers were able to make updates to their own environments at a faster pace without the pressure of disturbing testing or other development work. The QA environment closely imitates the production environment and helps with verification before changes are deployed to production. [8]

4.1 Existing System Overview

The Java-based ATR Works application was created using the Spring Framework, with a combination of Spring MVC and Spring Web. Originally, the application only utilized Spring MVC and did not support any real-time updates. During the latest update to the application, a RESTful API was created using Spring Web to enable the use of reactive UI.

Additionally, another version of the same application called ATR Works Extension was created that only supported core features, like creating bookings and travels. This application was running on a DMZ server and could be accessed from the Internet without a VPN connection. Having multiple versions of the same application resulted in more maintenance and the decision to retire

ATR Works Extension was made, as it had less features compared to the main application.

The on-premises application infrastructure consisted of three main parts show in Figure 1, the application server for ATR Works, a Microsoft SQL Server, and a Microsoft Active Directory (AD) environment. In total there were two separate environments in use, one for QA and the other for production. Both environments had their own Linux servers, databases, and AD users. As the AD and databases were not accessible outside of the on-premises network a VPN connection was needed to run the application locally for development purposes.

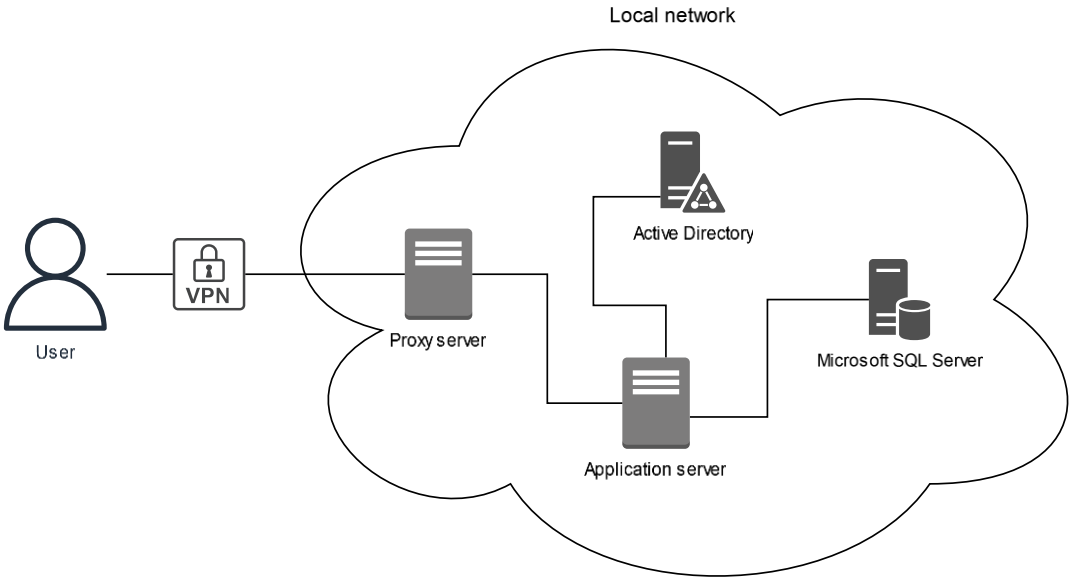


Figure 1 Application infrastructure before the migration.

ATR Works was running on a Linux server and was hosted using Apache Tomcat. A separate Linux server running an Apache HTTP Server was used to proxy requests to the application, secure the web traffic using an SSL certificate, and gather access logs. The only way of reading application logs and monitoring performance was from the Linux servers CLI that could be accessed using a SSH connection.

Roles are an essential part of the application for hiding and restricting features like editing someone else's bookings. Configuring these roles was done using

Keycloak, which integrated with the on-premises AD. When a user signed in to the application the request was sent to Keycloak, that verified it against the on-premises AD and on success responded with a JSON Web Token (JWT) that included the configured roles as claims. [9] These claims could be read by the application, where more detailed privileges were configured.

4.2 Target Cloud Architecture

The goal when designing the architecture to Microsoft Azure for ATR Works, was to make it easy to maintain and scale based on the usage. Usually when scalability is mentioned the goal is to prepare for a sudden growth in users accessing the application, by creating new instances of the same application and load balancing the users to them evenly. [10] The scalability we were after was about scaling down or even shutting off the application when it is not used. For example, the application rarely gets accessed during the weekends and midweek holidays, so it could shut down to save resources.

Having a separate QA environment where new features and bug fixes can be tested works as a filter on what gets deployed to production. It is important for these environments to be as similar as possible to minimize the probability of changes not working as expected in the production environment. Creating a duplicate environment does increase the hosting costs but in general it ends up saving time and money by reducing the number of bugs and faulty features deployed to the production environment. [11]

Like the original on-premises architecture, the cloud-based solution had two environments. Both environments used an Azure App Service for the Java application, Azure SQL Database as the database, and Microsoft Entra ID to replace the on-premises AD (see Figure 2). The responsibilities of Keycloak were transferred fully to Microsoft Entra ID, where mapping users and groups to application specific roles was possible, further simplifying the maintenance by limiting the number of locations where the application is configured.

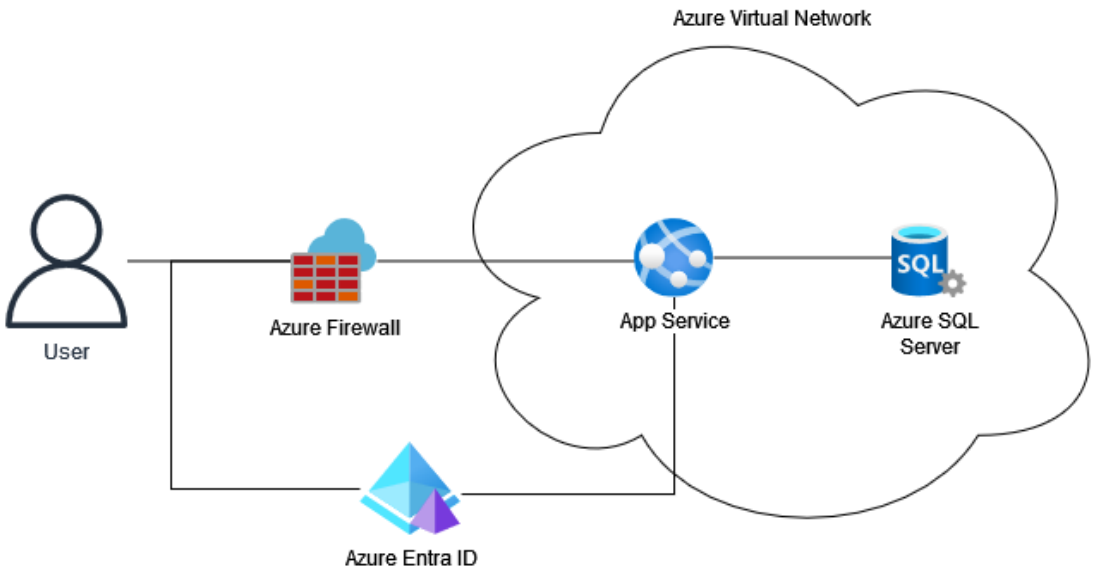


Figure 2 Planned application infrastructure on Microsoft Azure

Both environments were sandboxed using virtual networks and accessing different environments resources was blocked. This was important for ensuring security for the data and minimizing bugs and problems caused by the testing process. To improve the testing quality of the QA environment, a separate fully local database was used for development. [12]

Finally, the Netvisor integration that was running on a separate Windows server on-premises was migrated to run as a timed Azure Function. The original code was modified to make it possible for the Azure Function code to call the integration in the same way it was called on-premises.

4.3 Service Selection and Justification

Before deciding to use an Azure App Service to host the application, solutions such as a Linux virtual machine (VM) and Azure Spring Apps were considered. Using a Linux VM would have made the migration process simpler by allowing the use of the on-premises infrastructure as a blueprint, but it did not provide the wanted scalability and maintenance simplification. Azure Spring Apps was a

promising option, but the additional features compared to Azure App Service did not justify the higher price. [13]

Using Azure App Service provided the wanted scalability by having an option to scale it down or even stop the service after a certain amount of idle time. Also, the App Service granted ease of use by having all the needed configuration such as environment variables and deployment options available from the Azure Portal.

5 Implementation

In this chapter, we go through the practical steps taken to implement the cloud-based solution. This includes planning, creating the resources to Azure, implementing authentication and authorization using Microsoft Entra ID, modifying the source code of ATR Works, and migrating the database. The following sections provide an overview of the planning, good practices about cloud migration, and a step-by-step execution of the implementation process.

5.1 Planning

Like the on-premises implementation, the cloud-based solution was planned to have two environments, one from QA and other for production. These environments would both consist of the Java-based ATR Works application, a SQL server, and a SQL database. Both environments would use the same Microsoft Entra ID tenant for authentication with separate accounts.

The plan was to first create the QA environment, test that it is working correctly, harden the security of it and write documentation about the way it was configured. After the QA environment is working as it should, creating the production environment should be simpler, as it would be possible to follow the same steps. When the production environment is configured, the database can be migrated from on-premises to Azure.

The production migration was scheduled to happen during the next summer holiday that was out of the thesis timeframe. Only the preparation and testing of the production environment was planned for this thesis.

5.2 Creating Azure Resources

While this application does not require a lot of separate resources, it is still a good idea to organize them well. One of the first organizational problems when creating multiple resources for testing purposes was with naming them, and just

writing the application's name for each resource made it difficult to distinguish between the resource types. To fix this issue, a naming convention was followed. Each resource name started with the environment, followed by the applications name, and ending resource type (see Table 1). This made it easy to distinguish between the resources and most importantly made it easy to confirm what environment the resource belongs to.

Table 1 Examples of the Azure resource naming convention used.

Environment	Application name	Resource type	Resource name
Production	Example	App Service	prod-example-app
QA	Example	SQL Server	qa-example-sql
Development	Example	SQL Database	dev-example-db

Before creating the application-specific resources, the Azure's four-level deep resource hierarchy (see Figure 3) was used to add separation between the QA and production environment. Both environments got their own subscriptions and resource groups. This separation provides many benefits, such as improved security, by only allowing communication between resources in the same resource groups, easier access management, tracking costs of specific subscriptions, and visual separation to distinguishing the environments from each other when using the Azure Portal. [14]

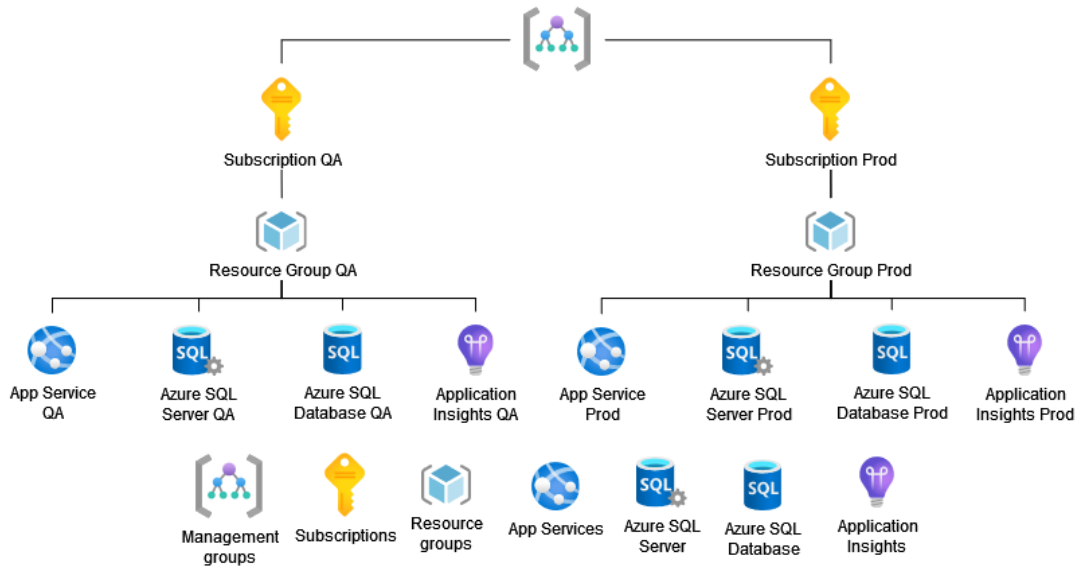


Figure 3 Planned Azure resource hierarchy of the application

After creating a subscription and a resource group for each environment the application resources were created inside of their own resource groups. The application resources included an App Service for ATR Works, Azure SQL Server with a SQL Database for application data, and an Application Insights instance for monitoring all application resources. Additionally, a timed Azure Function App was created to run the Netvisor integration.

5.3 Authentication

The application used OAuth for authentication (see Figure 4). Building the OAuth authentication was done by configuring the Entra ID tenant to work as an OAuth provider for ATR Works and ATR Works was configured using the “Spring Boot Starter OAuth2 Client” Maven dependency to work as an OAuth client. [15]

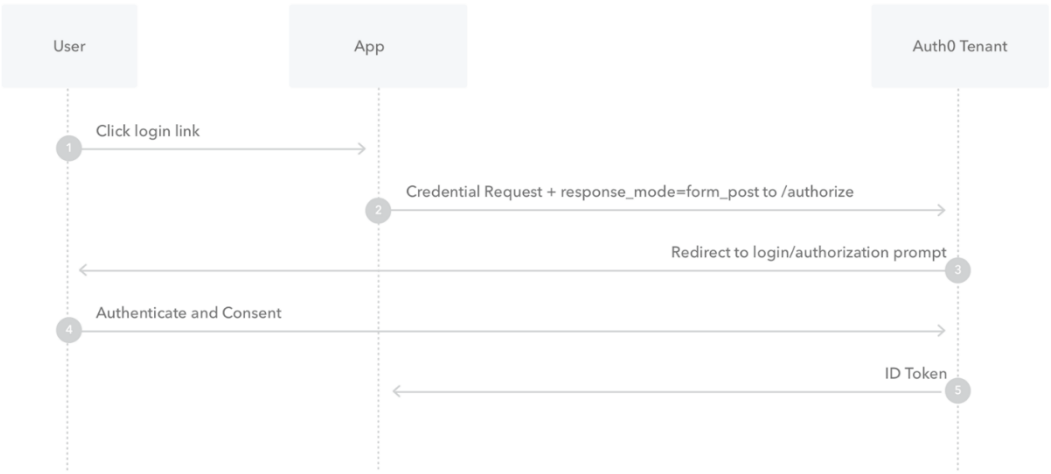
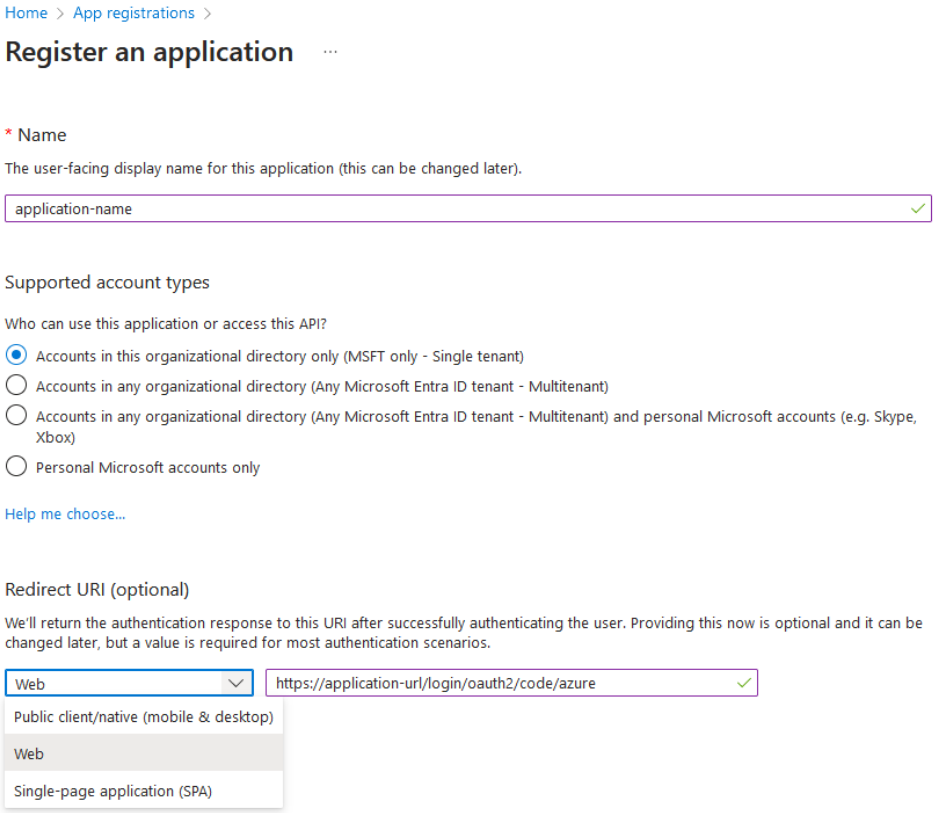


Figure 4 OAuth authorization flow diagram. [15]

Configuring Microsoft Entra ID for the authentication required using the web-based identity portal called Microsoft Entra admin center. Before it was possible to create any application specific configuration, the application needed to be registered (see Picture 1). During the registration the single tenant option was selected from the options for supported account types, as we only wanted the users of the same organization to be able to sign into the application. [16]

Configuring where Entra ID sends the security tokes and redirects the clients after successful authentication was done by configuring the redirect URI show in Picture 1. This URI establishes a link from Entra to the application and ensures that the security tokens are only sent to the configured applications. The redirect URI can be configured and modified after the registration process, additionally it is possible to add multiple redirect URIs for the same application. For example, to make running ATR Works locally with Entra ID authentication a redirect URI that pointed to localhost was added. [17]



Picture 1 Microsoft Entra admin center “Register an application” form.

The trust from Entra to ATR Works was established by configuring the redirect URI but Entra could not trust the applications requests, as the application did not have a way of identifying itself. To solve this problem a client secret was generated from the Microsoft Entra admin center for the registered application. Finally, the client ID, client secret, and tenant ID from Entra were added to the applications configuration (see Picture 2) and the needed trust between Entra and the application was achieved.

```
security:
    oauth2:
      client:
        registration:
          azure:
            client-id: <client-id>
            client-secret: <client-secret>
            scope: openid, profile, email
            client-name: Entra ID
        provider:
          azure:
            issuer-uri: https://login.microsoftonline.com/<tenant-id>/v2.0
```

Picture 2 Spring OAuth 2.0 configuration in application.yml.

Enabling the OAuth authentication to ATR Works was done through the security configuration provided by the Java Spring Framework (see Picture 3) by adding the “oauth2Login” method call to the security filter chain configuration. Java Spring offers a default configuration for the OAuth authentication but for our use case a custom “OidcUserService” and a custom “UserAuthoritiesMapper” were created. The OIDC (OpenID Connect) user service was configured to link the user data from the database to the authenticated user using the email address. The user authorities mapper was used to map the user roles to privileges, this is discussed more in section 5.4.

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
    // other configurations
        .oauth2Login(oauth2 -> oauth2
                .userInfoEndpoint(userInfo -> userInfo
                        .oidcUserService(this.oidcUserService())
                        .userAuthoritiesMapper(this.userAuthoritiesMapper())
                )
    // other configurations
    return http.build();
}
```

Picture 3 The OAuth configuration from ATR Works security configuration.

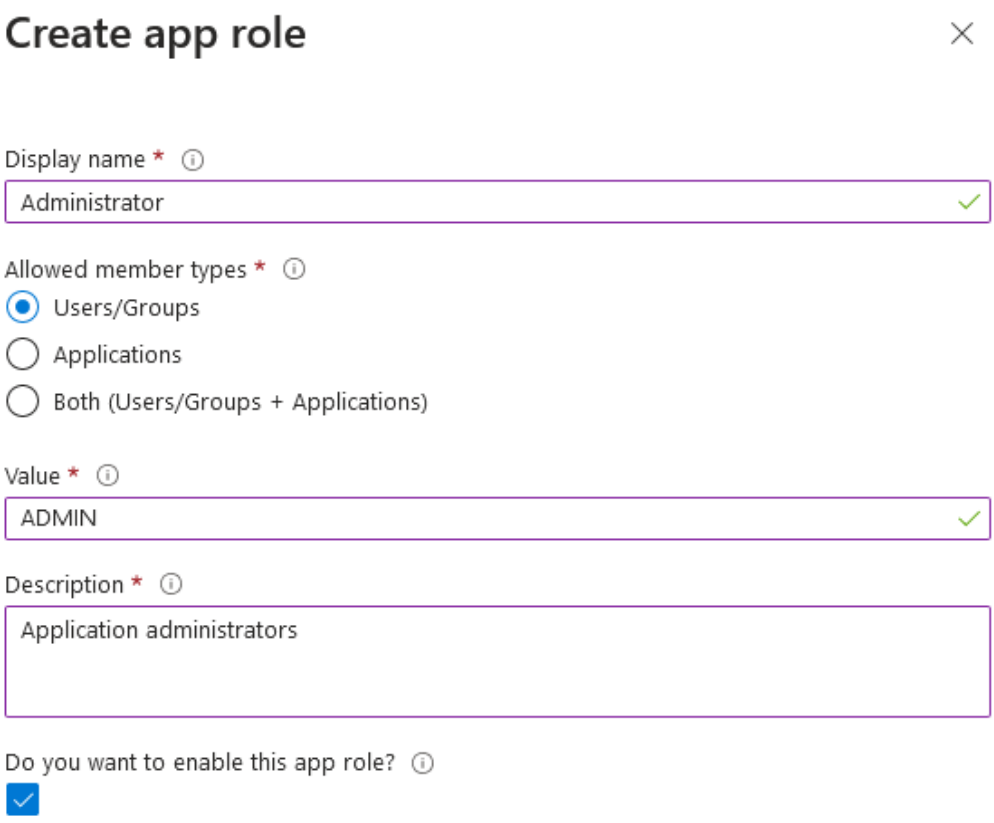
5.4 Authorization

Configuring authorization for the application required that the Entra ID tenant would send the user roles as a claim to ATR Works and that the application could map the roles to application specific privileges. The application has a configuration file for the privilege configurations and an example can be seen in the Picture 4. The example shows privileges for creating, modifying, and searching users. Based on this example the application would hand the privileges for creating and modifying users to everyone with the ADMIN role and the privilege to search for users to everyone with the USER role.

```
user:
  create: ROLE_ADMIN
  modify: ROLE_ADMIN
  search: ROLE_USER
```

Picture 4 Example of the application privilege configurations.

Adding application roles to Entra ID was done from the application registration page by clicking the “Create app role” button. When the button was pressed a form opened (see Picture 5) for which it was possible to create a new application role. The most important field for our configuration was the “Value”, as that is the value added to the roles claim. Entra ID automatically prefixed the value with “APPROLE_”. This meant that the value added to the roles claim for the example of Picture 4 was “APPROLE_ADMIN”.



Picture 5 Entra ID form for creating an application role.

After the application role was created it still needed to be mapped to a user or a group from the “Users and groups” menu that could be found from the “Enterprise applications” section. After mapping a user or a group to an application role the role was sent in the roles claim from Entra ID to ATR Works during the authentication process. This claim was then read by ATR Works and mapped to application specific privileges using a custom user authorities mapper. The mapper was configured using the Spring Boot security configurations (see Picture 3).

5.5 Database Migration

Before the migration, the application used the authenticated user’s username that was obtained from the “sAMAccountName” authentication claim, for most of the database queries that were made. For example, when a new booking was created, the booking was linked to the user by attaching the username to the booking row created in the database. Using the username to add relations to the data was made impossible when the authentication flow was changed, as the new implementation did not include a replacement for the “sAMAccountName” attribute. [16]

To fix the issue of linking data to users a few different options were considered before landing on the solution used, using the user’s email address, and using the Object ID provided by Azure. Using the email addresses was discarded, as it would have made changing the user’s email a difficult and error prone process. The use of Azures Object IDs would have otherwise been a valid choice but the fear of vendor lock-in made it less appealing, as the whole database would need to be migrated if the cloud service provider changed in the future.

The solution with the least drawbacks was to add an autoincrementing field to the database that would be linked to each user using the email address they authenticate with. This method made changing users email addresses a lot simpler by only needing to change a single field in the database. This solution should also work with any cloud service provider in the same way as it does

with Microsoft Azure. Additionally, having a user ID instead of a username that could easily be linked to an employee helped with General Data Protection Regulation (GDPR) by making it impossible to link data to a specific user without additional knowledge about the database.

Shifting from usernames to user IDs required a full migration of the database, as each row had a column specifying the user that created or modified the row. The database schema was migrated by replacing the username field on every table with the user ID. Migration scripts were created to make it possible to migrate the existing database without losing any data.

Liquibase was used for schema migration, by first running a command with the Liquibase CLI tool to generate XML files containing the old database tables, relations, and triggers. These XML files were then modified to work with the user IDs instead of usernames. With these changes, the application was able to create the whole database schema to an empty database.

To transfer the data from the old database to the new format without losing any data, SQL migration scripts were written. The data migration would start with a script creating a temporary username field to the database and transferring the users from the old database to the new one. During this transfer, the autoincrementing user ID field would get populated automatically. This script would also create a function for fetching the user ID of a user using the old username. After making it possible to fetch the user ID based on the username, the rest of the migration scripts were simply copying the data from the old database, except for the username columns which were replaced by calling the previously created function.

5.6 Codebase Modifications

Migrating the database from using a username to using a user ID resulted in a lot of modifications to the application code. The username was used in most of the SQL queries, JSP templates, and controllers. The modifications were done in stages by first adding the user ID column to the database, modifying the part

of the code to use it instead of the username, and finally deleting the username column. The database modifications were only done to a local development database that was easy to clear and reset.

5.7 Deployment Strategy

Before the migration, deploying a new version of the application was done with the combination of Bitbucket for version control, an on-premises TeamCity instance for building the application, and an on-premises Octopus instance for deploying the package built by TeamCity to on-premises servers. To deploy a new version, the developer would commit their changes to Bitbucket, use TeamCity to build a new package from the source code on Bitbucket, and deploy the package to an on-premises server using the Octopus web interface. As the application was not at the time under daily development TeamCity and Octopus were retired, simplifying the maintenance, and reducing the total costs by removing the need for the TeamCity and Octopus licenses.

Replacing the work previously done by TeamCity was done with Maven, meaning that testing and building the package happened on the developer's machine. This removed the need for any external services and did not require as much additional knowledge, as Maven was already used for development. To build a new version of the application, the developer would first commit their changes to Bitbucket, package the application with Maven, triggering Maven to compile, test, and package the application. [18] At this point the packaged application was ready to be deployed.

Deploying the package, previously done by Octopus was replaced with a Maven plugin provided by Microsoft, called "azure-webapp-maven-plugin". The configuration for this plugin can be found from the project POM file (see Picture 6) and includes information such as the subscription ID, application name, and resource group. This information is used to identify the resource where the application should be deployed or if the resource does not exist, where to create a new resource for the deployment. Executing the Mave command "mvn azure-

webapp:deploy” deploys the application by authenticating to Microsoft Azure using the Azure’s az CLI tool, clarifying where to deploy the application with the configuration information, and finally deploying the built package. [19]

```
<build>
    <plugins>
        <plugin>
            <groupId>com.microsoft.azure</groupId>
            <artifactId>azure-spring-apps-maven-plugin</artifactId>
            <version>x.xx.x</version>
            <configuration>
                <subscriptionId>xxxxxxxxx-xxxx-xxxx-xxxxxxxxxxx</subscriptionId>
                <resourceGroup>rg-example-dev</resourceGroup>
                <appName>app-example-dev</appName>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Picture 6 Example configuration of “azure-webapp-maven-plugin”.

5.8 Monitoring

Monitoring the application was previously done by connecting to the application server via SSH and using Linux tools such as top for process activity monitoring and netstat for network statistics. Errors and possible bugs were investigated from the Apache Tomcat and Apache HTTP server logs. This approach made monitoring and resolving issues a tedious and time-consuming task, as the performance metrics were not logged, and each server needed to be checked individually.

For each environment a Microsoft Azure Application Insights instance was created. This instance included all environment specific resources in one place, making the monitoring less tedious. The Application Insights resource has options to automatically send alerts on events such as high CPU usage and slow HTTP response times, making it possible to automate part of the monitoring process.

6 Conclusion and future work

The goal of this thesis was to migrate a Java-based application called ATR Works from on-premises infrastructure to Microsoft Azure, simplifying the maintenance and improving the scalability. Different hosting options on Microsoft Azure were researched and from that selection, Azure App Service was selected, as it matched the commissioner`s' need the best by providing simplified maintenance and scalability.

Two separate environments for the application were created, one for quality assurance and one for production. The production environment has not yet replaced the on-premises instance, as the shift was scheduled to happen outside of the thesis timeframe. Both environments have their own resources on Microsoft Azure and follow the same infrastructure, except for the Microsoft Entra ID tenant that is shared across both environments. Using the QA environment enables the testing of code changes and having two almost identical environments running verifies that the changes should also work in the production environment.

Additionally, configuring Azure's Application Insights monitoring service provided enhanced visibility of the application behavior and performance, decreasing the amount of manual monitoring work and the amount of time to respond to potential issues. This further helped with quality assurance and maintenance processes.

Managing authorization for the application was modified to work with specific groups created in Microsoft Entra ID, meaning that the user would automatically gain the configured privileges based on what groups they belong to. This made managing authorization easier, as all the configuration was done from the same place.

Switching to Microsoft Azure did come with its own set of challenges. While using a cloud-based solution reduced the amount of hardware maintenance and yielded the desired results for the commissioner, it introduced a new learning

curve. The traditional method of configuring servers and services on-premises in a Linux environment was less applicable, requiring the development team and system administrators to acquire Azure specific knowledge.

Looking ahead, configuring a CI/CD service such as Azure Pipelines for the application would simplify the development process and accelerate the development pipeline. The current implementation that utilizes a Maven plugin does function well but is prone to human error and requires considerable communication regarding deployments, as multiple developers could be deploying at the same time without any feedback from the Maven plugin.

Overall, the results of this thesis were an important part of the commissioner's goal to move fully away from on-premises infrastructure to cloud-based solutions.

References

[1] Microsoft, "Azure for developers overview," 18 October 2022. [Online]. Available: https://learn.microsoft.com/en-us/azure/developer/intro/azure-developer-overview. [Accessed 12 May 2024].

[2] C. Griffiths, "The Latest Cloud Computing Statistics," AAG, 1 May 2024. [Online]. Available: https://aag-it.com/the-latest-cloud-computing-statistics/. [Accessed 12 May 2024].

[3] Microsoft, "What is the Azure portal?," Microsoft, 4 April 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-portal/azure-portal-overview. [Accessed 12 May 2024].

[4] Microsoft, "What is the Azure CLI?," Microsoft, 8 March 2024. [Online]. Available: https://learn.microsoft.com/en-us/cli/azure/what-is-azure-cli. [Accessed 12 May 2024].

[5] Microsoft, "What is Microsoft Entra ID?," Microsoft, 29 March 2024. [Online]. Available: https://learn.microsoft.com/en-us/entra/fundamentals/whatis. [Accessed 20 May 2024].

[6] Apache, "What Is Maven?," Apache, 17 May 2024. [Online]. Available: https://maven.apache.org/what-is-maven.html. [Accessed 20 May 2024].

[7] Liquibase, "Introduction to Liquibase," Liquibase, 2024. [Online]. Available: https://docs.liquibase.com/concepts/introduction-to-liquibase.html. [Accessed 29 May 2024].

[8] S. Masuda, J. Hagar, Y. Nishi and K. Suzuki, "Software Test Architecture Definition by Analogy with Software Architecture," in *2022 IEEE*

International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Valencia, Spain, 2022.

[9] A. B. Kretarta and H. Kabetta, "Secure User Management Gateway for Microservices Architecture APIs Using Keycloak on XYZ," in *2022 5th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, Yogyakarta, Indonesia, 2022.

[10] C. Escoffier and K. Finnigan, Reactive Systems in Java, O'Reilly, 2021.

[11] S. M. Marebane and R. T. Hans, "Factors Affecting Software Quality Improvements in South African Software Development Environments," in *2023 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, Cape Town, South Africa, 2023.

[12] M. Abbadini, M. Beretta, D. Facchinetti, G. Oldani, M. Rossi and S. Paraboschi, "Lightweight Cloud Application Sandboxing," in *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Naples, Italy, 2023.

[13] Microsoft, "What is Azure Spring Apps?," Microsoft, 30 January 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/spring-apps/enterprise/overview. [Accessed 22 May 2024].

[14] Microsoft, "Organize your Azure resources effectively," Microsoft, 6 June 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-setup-guide/organize-resources. [Accessed 23 May 2024].

[15] Okta, "Authorization Code Flow," Okta, [Online]. Available: https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow. [Accessed 2 June 2024].

[16] Microsoft, "Configure group claims for applications by using Microsoft Entra ID," Microsoft, 6 November 2023. [Online]. Available:

https://learn.microsoft.com/en-us/entra/identity/hybrid/connect/how-to-connect-fed-group-claims. [Accessed 16 May 2024].

[17] Microsoft, "Register an application with the Microsoft identity platform," Microsoft, 23 October 2023. [Online]. Available: https://learn.microsoft.com/en-us/entra/identity-platform/quickstart-register-app. [Accessed 1 June 2024].

[18] The Apache Software Foundation, "Maven in 5 Minutes," The Apache Software Foundation, 30 May 2024. [Online]. Available: https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html. [Accessed 30 May 2024].

[19] Microsoft, "Create a Java app on Azure App Service," Microsoft, 2 October 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/app-service/quickstart-java?tabs=springboot&pivots=java-maven-tomcat. [Accessed 30 May 2024].

[20] F. Gao, C. Liu, B. Meng and S. Chen, "A Software Cost Model Considering the Difference between Testing Environment and Operating Environment," in *2014 Seventh International Joint Conference on Computational Sciences and Optimization*, Beijing, China, 2014.

[21] R. Aljamal, A. El-Mousa and F. Jubair, "A comparative review of high-performance computing major cloud service providers," in *2018 9th International Conference on Information and Communication Systems (ICICS)*, Irbid, Jordan , 2018.