CHALMERS
UNIVERSITY OF TECHNOLOGY

Infrastructure as Code and Automation

Using Azure Bicep and Azure Pipelines to Reduce Operational Costs of Infrastructure Through Automation

Degree project report in Computer Science and Engineering

Peach Larsson
Rozgar Khatab

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Degree project report 2024

Infrastructure as Code and Automation

Using Azure Bicep and Azure Pipelines to Reduce Operational Costs of Infrastructure Through Automation

PEACH LARSSON
ROZGAR KHATAB

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Infrastructure as Code and Automation
Using Azure Bicep and Azure Pipelines to Reduce Operational Costs of Infrastructure Through Automation

PEACH LARSSON
ROZGAR KHATAB

Supervisor: András Kovács, Department of Computer Science and Engineering
Examiner: Jonas Duregård, Department of Computer Science and Engineering

Degree project report 2024
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: Logo of Chalmers University of Technology.

Typeset in LaTeX
Gothenburg, Sweden 2024

Infrastructure as Code and Automation
Using Azure Bicep and Azure Pipelines to Reduce Operational Costs of Infrastructure Through Automation

PEACH LARSSON
ROZGAR KHATAB
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

This report describes the development of infrastructure as code to automate the Azure testing environment of the Information Technology department of the corporation New Wave Group. The primary goal is to reduce the monthly incurred costs of Azure resources by deleting them when not in use and recreating them when in use. A secondary goal is to minimize environmental impact by ensuring that idle resources, hosted in data centers, do not consume energy. Bicep, a domain-specific declarative language was used to encapsulate the existing infrastructure into Bicep files as code. These files were deployed, creating the infrastructure, using an Azure pipeline with YAML scripts. The deletion of resources was managed through a separate pipeline and YAML file. The pipelines for deployments and deletions create an automation cycle to terminate and reinitialize infrastructure. Due to time constraints, this work was conducted as a proof of concept in a sandbox environment and excluded certain resources and resource groups. Testing was performed by manually validating the results of deployments in the sandbox environment. It is recommended that New Wave Group further develop this proof of concept to include all resource groups and resources, especially databases. Automation of databases must comply with legal and ethical requirements such as GDPR. An economic analysis shows that automating the three resource groups in this project and an additional resource group for storage can save New Wave Group approximately 8690 SEK per month. Additional savings are possible by automating the remaining resource groups outside of the scope of this project.

Keywords: ARM, Bicep, Templates, Azure, Automation, IaC, Resources, Pipelines, YAML, Infrastructure

Acknowledgements

We would like to thank Attila Lundin and Syed Shahzaib Haider Kazmi for being great classmates and colleagues during this project, as they worked on the webhook part of the automation system in a different project. We would also like to thank Argon Omarson, Jimmy Balderud and Erik Olausson from New Wave Group and András Kovács, our Chalmers supervisor, for all the assistance throughout the project. Additionally, special thanks to Sakib Sistek for lending a helping hand, despite not being our supervisor, for any worries we had during our studies.

Peach Larsson, Rozgar Khatab, Gothenburg, June 2024

List of Acronyms

Acronyms in alphabetical order that have been used in this thesis report:

API	Application Programming Interface
ARM	Azure Resource Manager
CLI	Command-Line Interface
GDPR	General Data Protection Regulation
IaC	Infrastructure As Code
IP	Internet Protocol
IT	Information Technology
NAT	Network Address Translation Gateway

Contents

1

Introduction

1.1 Background

Software engineering is an integral branch of modern civilization that is becoming increasingly larger and arguably the most important industry of our future. As the industry and customer demand grow, the scope and complexity of software grow alongside. To accommodate the increasing complexity of software, companies such as Microsoft sell infrastructure on the cloud to enable companies to host their websites and apps on the web. Azure, owned by Microsoft, allows a company to create infrastructure for their web site including, but not limited to, Network Address Translation (NAT) gateways, virtual networks, route tables, SQL databases and servers. Setting up such infrastructure manually is an arduous and time-consuming process which leaves room for user error. Infrastructure as Code (IaC) solves this problem. Using IaC, the infrastructure of an application or website can be defined as code and be used, in an automatic, repeatable and consistent way, to create infrastructure [1].

New Wave Group has a net revenue of 2337.0 million SEK in the period from 1 July to 30 September 2023 [2]. New Wave Group's headquarters is situated in Gothenburg with offices located in, among other countries, Switzerland, China and India [3]. The company works within the promotional industry and additionally within sports, gifts and interior design. To conduct its business, the company has IT departments that develop their services and products to support its diverse business activities [4]. To test their services and products, the company has a testing environment in Azure which has operational costs on a monthly basis. The challenge is that the infrastructure for the testing environment exists even when not in use, incurring 100% monthly operational costs. This results in unnecessary financial expenditure. The purpose of this thesis is to use Azure Resource Manager (ARM) templates and Azure Pipelines to automate initialization and decommission of the infrastructure for their test environment resources in Azure. These resources include but are not limited to web application components, database storage, and network infrastructure. The primary goal is to enable New Wave Group to activate and deactivate resources in their testing environment as needed. Ultimately this will benefit New Wave Group economically by reducing the operational expenses of their infrastructure. Furthermore, decommissioning of infrastructure has a positive ecological impact as resources are hosted in energy-consuming data centers.

1.2 Purpose

The primary purpose of this thesis is to automate the initialization and decommission of Azure resources in the test environment of New Wave Group's IT department using ARM templates and Azure Pipelines. Such resources include web application components, databases, storage, network infrastructure, and more. The aim is to reduce operational costs due to not incurring costs for idle resources. Additionally, New Wave Group desires to minimize damage to the environment by shutting down idle resources which consume significant energy over time.

1.3 Objectives

The objectives for this project are:

- Creating ARM templates and using Azure Pipelines to build and configure test environment resources.
- Implementing functionality to securely delete test environment resources.
- Conducting comprehensive testing to ensure the reliability, security, and efficiency of the implemented solution.

1.4 Limitations

The project primarily focuses on automating the creation and deletion of Azure resources in the test environment using ARM templates and Azure Pipelines. The scope of this project will exclude covering all possible Azure resources and configurations. Resource types that will not be automated are:

- Databases
- Storage accounts

Due to time constraints, the project will automate the creation and deletion of the test environment resources in a sandbox resource group as a proof of concept.

2
Technical Background

This chapter describes the technical concepts necessary to understand the project.

2.1 Infrastructure as Code

Infrastructure as code is a method to define infrastructure components, such as networks, virtual machines, NAT gateways, databases, storage, and more, as code. This avoids manual creation and configuration of infrastructure, which facilitates consistency, reliability and enables automation of creating infrastructure. Infrastructure as code is written in files and can be used by pipelines for automation [1].

2.2 Azure - a Cloud Computing Platform

Azure is Microsoft's public cloud platform. Similar to other cloud platforms such as Amazon Web Services or Google Cloud Platform, Azure relies on virtualization technology [5]. This technology allows most computers to be emulated, enabling virtualized hardware to execute as if it were actual hardware. The cloud has physical servers in data centers executing virtualized hardware to provide computing services for the customers. In the cloud, millions of instances of virtualized hardware can be created, started, stopped and deleted simultaneously.

2.3 Azure Resource Manager (ARM) Template

In Azure, a resource refers to an entity controlled by Azure [6]. Resources correspond to infrastructure components such as web applications, virtual machines, NAT gateways, virtual networks, route tables, databases, and more. Deployment and management of Azure is conducted using Azure Resource Manager [7]. The manager provides a layer which allows the possibility to create, update and delete resources in an Azure account. After the deployment is completed, management features for securing and organizing the resources can be leveraged.

A consistent management layer simplifies the process of interacting with resources. All requests are directed to ARM for authentication and authorization. Once granted, ARM forwards the request to the appropriate Azure service [8]. Figure 2.1 illustrates the responsibility of ARM in handling Azure requests.

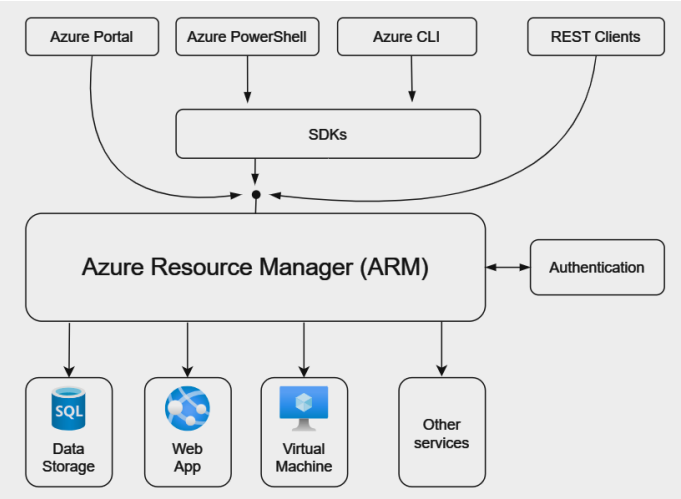

Figure 2.1: An illustration of how ARM handles Azure requests

With the shift to the cloud, agile development methods have become widespread, enabling teams to iterate quickly and deploy solutions to the cloud regularly. This necessitates a reliable infrastructure, blurring the lines between operations and development. To address these challenges, teams adopt automation and infrastructure as code practices [9].

Similar to application code, infrastructure code is stored in a version-controlled repository, allowing any team member to deploy consistent environments by running the code. This unified approach streamlines development, ensuring reliable and efficient cloud deployments [10].

ARM templates are utilized to implement infrastructure as code for Azure solutions. These templates are JavaScript Object Notation (JSON) files that define the infrastructure and configuration for the project. ARM templates use declarative syntax which allow deployment conveniently by declaring the desired result in code without needing to specify how to achieve it. In the template, you outline the resources to deploy and their respective properties [8].

2.3.1 Bicep

Instead of directly using JSON format for ARM templates. Bicep is a domain-specific language designed specifically for authoring ARM templates, providing a more streamlined and readable way to define infrastructure as code. It aims to simplify managing Azure resources while maintaining full compatibility with existing ARM templates [12].

Bicep improves upon JSON in several key ways [12]:

1. Simplicity and readability: Bicep syntax is concise and human-readable, reducing the complexity typically associated with JSON. This makes it easier for developers and operations teams to write, review, and maintain infrastructure code.

2. Modularity: Bicep supports modularization, allowing users to break down complex deployments into manageable and reusable components. This is achieved through the use of Bicep modules, which can be referenced in other Bicep files to promote code reuse and better organization.
3. Transparency and compatibility: Bicep files are transparently transpiled into standard JSON ARM templates before deployment. This ensures that all existing ARM capabilities are fully supported, and there is no performance penalty for using Bicep.
4. Tooling and support: Bicep is fully supported by Azure and integrates seamlessly with Azure DevOps and GitHub Actions. It also comes with dedicated tooling for validation, deployment, and resource management, making it a robust choice for infrastructure as code on Azure.

Figure 2.2 illustrates a comparison between Bicep and JSON defining the same storage account [3]:

```bicep
@description('Storage Account type')
@allowed([
  'Premium_LRS'
  'Premium_ZRS'
  'Standard_GRS'
  'Standard_GZRS'
  'Standard_LRS'
  'Standard_RAGRS'
  'Standard_RAGZRS'
  'Standard_ZRS'
])
param storageAccountType string = 'Standard_LRS'

@description('The storage account location.')
param location string = resourceGroup().location

@description('The name of the storage account')
param storageAccountName string = 'store${uniqueString(resourceGroup().id)}'

resource sa 'Microsoft.Storage/storageAccounts@2022-09-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: storageAccountType
  }
  kind: 'StorageV2'
  properties: {}
}

output storageAccountName string = storageAccountName
output storageAccountId string = sa.id
```

```json
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deployment
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.14.6.61914",
      "templateHash": "14166394750243150384"
    }
  },
  "parameters": {
    "storageAccountType": {
      "type": "string",
      "defaultValue": "Standard_LRS",
      "allowedValues": [
        "Premium_LRS",
        "Premium_ZRS",
        "Standard_GRS",
        "Standard_GZRS",
        "Standard_LRS",
        "Standard_RAGRS",
        "Standard_RAGZRS",
        "Standard_ZRS"
      ],
      "metadata": {
        "description": "Storage Account type"
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "The storage account location."
      }
    },
    "storageAccountName": {
      "type": "string",
      "defaultValue": "[format('store{0}', uniqueString(resourceGroup().id))]",
      "metadata": {
        "description": "The name of the storage account"
      }
    }
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2022-09-01",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
```

Figure 2.2: Comparison between Bicep (to the left) and JSON (to the right) for the same storage account. Bicep syntax is more concise and readable.

In this example, the Bicep code defines a storage account resource with its name, location, and other properties. This concise format simplifies the process of writing infrastructure as code, making it more approachable and less error-prone compared to the equivalent JSON syntax.

2.4 Azure Command-Line Interface (CLI)

The Azure Command-Line Interface facilitates the execution of commands via a terminal using command-line prompts or a script [14]. This tool enables connection to Azure which allows execution of administrative commands on Azure resources. Moreover, automation of repetitive tasks is possible by assembling and executing CLI commands into a shell script. Additionally, Azure CLI is compatible with Windows, Linux and macOS.

2.5 Azure Pipelines and YAML

Azure Pipelines automates the building, testing, and deployment of code. Pipelines can be defined using YAML which specifies the build and deployment process.

2.5.1 Pipelines

Azure Pipelines supports a wide range of languages, frameworks, and platforms, including Javascript, Python, Java, Ruby, PHP, C, C++, and more [15]. Azure Pipelines integrates with two version control systems (VCS), Git and Azure Repos. This integration allows automatic triggering of builds whenever new changes are made in the repository [16].

2.5.2 YAML

With Azure Pipelines, pipelines can be defined using YAML or a visual designer within the Azure DevOps portal [17]. These pipelines consist of one or more stages, and each stage can contain multiple jobs. Within each job, tasks can be defined, such as compiling code, running tests, and deploying artifacts [18].

3

Method And Workflow

This chapter covers research, communication, and time planning. It discusses Microsoft's learning modules for Bicep and Azure Pipelines, planned workflow and agile methodologies and a Gantt chart for time planning. The purpose of this chapter is to outline the anticipated approach towards achieving the project goal.

3.1 Research and Communication

Microsoft provides learning modules for Bicep and Azure Pipelines, which are essential to developing and deploying Azure resources. Three distinct modules are available free of cost, each requiring approximately three hours to complete [19]. These modules directly align with achieving the goals of this project and will be completed at the start of the project.

This project is being developed in parallel with another project for New Wave Group, which is led by two other students. Our project focuses on implementing Infrastructure as Code and using YAML to define pipelines. Meanwhile, their project utilizes a webhook in Microsoft Teams chat to trigger the pipelines.

The planned workflow is inspired by agile principles, which encourage and foster continuous communication, adaptability and close cooperation between stakeholders [12]. Working according to agile methodologies has been proven to increase software development efficiency [13]. Communication channels between the members of this project, the students from the other project, and team members from New Wave Group will be established through a Microsoft Teams channel provided by the company. This fosters real-time collaboration and ensures that communication flows freely among team members. Additionally, weekly meetings are planned, typically at the start of the week, to provide updates on completed tasks and to discuss upcoming plans. These regular meetings allow for continuous feedback, iteration, and adaptation, all of which are fundamental principles of agile project management. Moreover, the company will undertake the roles of both the product owner, providing direction and priorities for the project, and the technical overseer, ensuring that the project aligns with the company's technological standards and objectives.

3.2 Time Planning

The project timeline will be divided into 22 weeks, aligning with the project time frame from the middle of January to the end of May and additional two weeks for final revision. This division will be further organized into three main phases: Planning, Main Work, and Revision as shown in Figure 3.1. The main phase will begin with acquiring foundational knowledge of Azure services and ARM Templates over the first few weeks, followed by a deeper understanding of Bicep. The majority of the time will be dedicated to developing Bicep files and planning for scalability as we build. After developing the templates and testing, the report will be written. Finally, reviews and revisions will be conducted followed by final testing to validate functionality.

GANTT Chart

NAME OF PROJECT

Cost-Effective Automation of Azure Test Environment Instances using ARM Templates

Color-code explanation

Expected progress

Progress on track

Delayed progress

TASK #	TASK	START WEEK	DUE WEEK	DURATION (WEEKS)
1	Define Research Objectives and Scope	1	2	2
2	Literature Review	1	2	2
3	Familiarize with Azure and ARM Templates	3	3	1
4	Familiarize with Azure Bicep	4	4	1
5	Design Bicep files	5	7	3
6	Development	8	16	9
7	Integration Testing	17	18	2
8	Documentation	19	19	1
9	Review and Revision	20	20	1
10	Final Testing and Quality Assurance	21	22	2

PHASE 1: Planning | PHASE 2: Main work | Phase 3: Revision

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

Figure 3.1: Gantt chart of the project.

4
System Architecture

This section begins with describing the early development process of the Bicep files, such as unsuccessful attempts and the factors that influenced design choices. It then describes the final product. For this project, a sandbox resource group `studentlab` was created by New Wave Group. Due to time constraints, the project conducts all operations in `studentlab` as a proof of concept, instead of in existing resource groups used by the company.

In this project, resource groups are not deleted and redeployed, only the resources within them are. This distinction is not important in the context of this project. "Deploying resources" and "deploying resource groups" will be used interchangeably. Furthermore, "dev environment" is used instead of "development environment". Sensitive company information, such as resource and resource group names and IP-addresses are obscured or replaced.

The final product consists of three primary Bicep files:

- `main.bicep`
- `AppHosting.bicep`
- `NetworkInfrastructure.Bicep`

Each resource group contains its own copy of these files, except `genericcms` which does not contain `NetworkInfrastructure.bicep`. Additionally, each resource group has a Bicep parameter file which feeds its `main.bicep` file with parameter values. Bicep parameter files are denoted by the file extension `.bicepparam`. To automate the deployment of the resource groups using these files, an Azure pipeline is created. The pipeline uses a YAML file that runs an Azure CLI command to deploy a resource group using the Bicep files including a parameter file. Deletion of resources is accomplished in a similar manner by using a different pipeline and YAML file. The pipelines can be run manually through the Azure portal. However, New Wave Group intends to trigger the pipelines using a webhook. The webhook is not part of this project; it is created by two students working on a different project for New Wave Group.

4.1 Design Choices and Considerations

When designing Bicep templates for the infrastructure of a specific company, there are no obvious best practices. Instead, the general structure must be based on various factors such as:

- existing infrastructure
- future infrastructure the company may want to adopt
- how configurable the deployments need to be
- how frequently configurations in specific deployments will be changed from default values
- how the company wishes to configure deployments when necessary
- different resource groups

Naturally, the simplest solution is possible when the infrastructure in terms of resource groups and resources will remain unchanged, the configurations in each deployment will be identical and the resource groups are as similar as possible without being identical.

4.1.1 Configurable deployments

Complexity arises when deployments need to be more configurable. This means that the Bicep templates need to contain more parameters, with optional default values that can be overridden. Parameter values can be defined at different levels where a parameter input at a level will override any input in levels below it. The lowest level is in the Bicep file where the resource definitions exist. Parameters defined in a Bicep file are a level above parameters defined in any Bicep files it imports as modules. A level above that is a Bicep parameter file with its unique file extension `.bicepparam` which can be used as input for a Bicep file. The highest level is assigning parameter values directly through the CLI when deploying a Bicep template. The specification for the solution was to assign the parameters using a centralized system. Bicep parameter files are external to the Bicep files making them modular and therefore suitable to use for a centralized system.

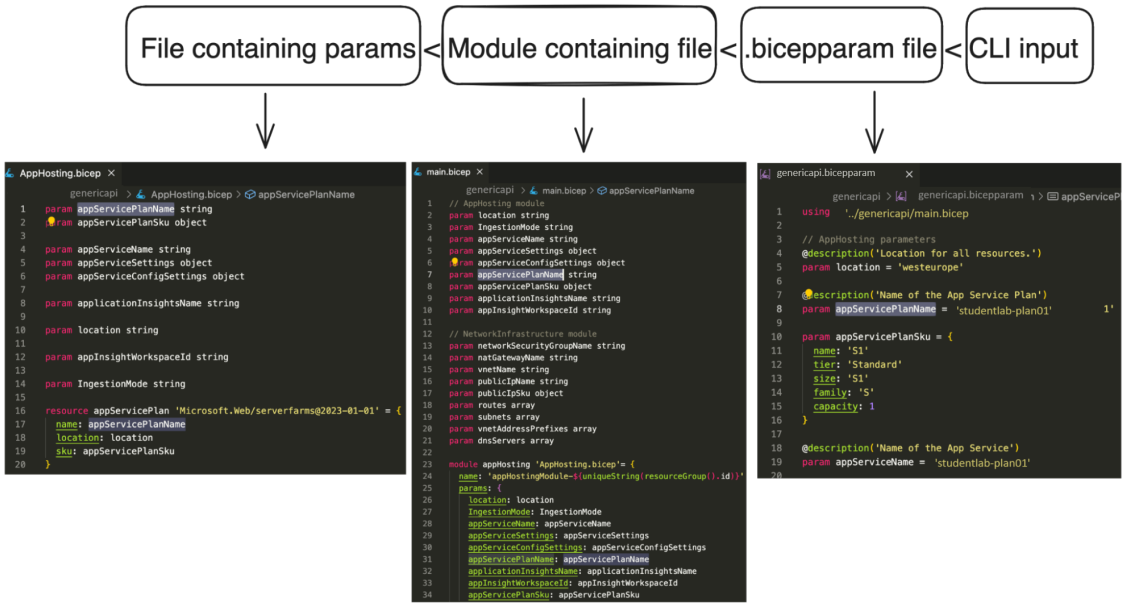

Figure 4.1: Figure showing the parameter `appServicePlanName`, highlighted in blue, defined at different levels with the highest being in the Bicep parameter file where the value is provided. In this project implementation, overriding through the CLI is not performed. Less-than sign is used to compare the priority of the parameter input sources.

The frequency of deployments, how often deployments use default values, how many and which configurations are changed all play a role in the final design of the system and its user interaction elements. If deployments happen frequently and a few parameters are regularly different between each deployment, it is important to put emphasis on the user experience of deploying resources. In such a case, it would be convenient to allow for overriding of parameters directly through the Azure CLI. A different scenario is that the parameters differ with a set of predetermined configurations. In this scenario, a choice superior to overriding through the CLI is creating a Bicep parameter file for each set of configurations and conveniently pointing to the correct file in the CLI for each deployment.

4.1.2 Designing readable and maintainable templates

In this project already existing resource groups are recreated as they are. However, the templates are designed to be modular and highly coherent with the aim to ease future development and modification. When a template is deployed, it creates a resource group such as the one in Figure 4.2 or puts resources into an already existing one.

Name	Type	Location
Failure Anomalies - genericapi-app01	Smart detector alert rule	Global
ndn	Network security group	West Europe
genericapi-app01	App Service	West Europe
genericapi-app01	Application Insights	West Europe
genericapi-nat01	NAT gateway	West Europe
genericapi-pip01	Public IP address	West Europe
genericapi-plan01	App Service plan	West Europe
genericapi-vnet01	Virtual network	West Europe
genericapi-vnet01-route01	Route table	West Europe

Figure 4.2: This figure shows the `genericapi` resource group and its resources. The resource group belongs to the dev environment of New Wave Group.

Name	Type	Location
genericshop-dashboard	Shared dashboard	West Europe
Failure Anomalies - genericshop-app01	Smart detector alert rule	Global
sanwgstoreinstoredev001	Storage account	West Europe
genericshop-app01	App Service	West Europe
genericshop-app01	Application Insights	West Europe
genericshop-plan01	App Service plan	West Europe
genericshop-vnet01	Virtual network	West Europe
genericshop-vnet01_route01	Route table	West Europe

Figure 4.3: This figure shows the `genericshop` resource group and its resources. The resource group belongs to the dev environment of New Wave Group.

Name	Type	Location
Failure Anomalies - genericcms-app01	Smart detector alert rule	Global
genericcms-app01	App Service	West Europe
genericcms-app01	Application Insights	West Europe
genericcms-plan01	App Service plan	West Europe
genericcmsstorage01	Storage account	West Europe

Figure 4.4: This figure shows the `genericcms` resource group and its resources. The resource group belongs to the dev environment of New Wave Group.

In Figure 4.2, 4.3 and 4.4, the resource groups which will be deployed using templates are shown. Importantly, the resource groups have certain resources in common, such as the app service resource. Some resource types are not in all groups, for instance only `genericapi` contains a NAT gateway resource while only `genericshop` and `genericcms` have a storage account resource respectively. However, the property values can differ between common resources. Furthermore, each resource can have child resources, not visible in the figures, which can differ significantly between the resource groups. For instance, the virtual network and route table resources in the `genericapi` resource group may have a different set of subnet and routes child resources compared to those in the `genericshop` resource group.

4.1.2.1 Early design attempts and challenges encountered

Bicep templates provide flexibility in how to manage different configurations of resources, child resources and their respective properties. Bicep templates should strive to reduce code redundancy or duplication as much as possible while maintaining readability and maintainability. To do this, Bicep files can be imported into other Bicep files as modules. Modules are useful to logically group resources; for this project exists two files used as modules: `AppHosting.bicep` and `NetworkInfrastructure.bicep` which are imported into `main.bicep`. In the beginning `AppHosting.bicep` contained the resources:

- App Service
- App Service plan
- Application Insights

`NetworkInfrastructure.bicep` contained the resources:

- Network security group
- NAT gateway
- Public IP address
- Virtual network
- Route table

Initially, the intention was to design the modules and the `main.bicep` file as general as possible, so that they can be used for all resource groups, including resource groups outside of the scope of this project. If successful, this approach would make maintainability trivial since a single file could be modified to easily make changes across all resource groups. An important question arises: how are the same three Bicep files used for all deployments given that different resource groups contain different sets of resources? Different syntactic methods exist in Bicep and can be used in combination to achieve this, such as:

- conditional logic
  - boolean data type
  - `if`-statement
  - ternary operator
- feeding entire properties as a Bicep object data type through a Bicep parameter file

Feeding properties through a Bicep parameter file as an object allows the set of properties for each resource to be defined in the Bicep parameter file for each resource group. This can help reduce conditional logic in the Bicep files. A downside of this feature is that it is syntactically incompatible with assigning a property directly in the resource definition. This limitation makes it unsuitable for cases where a resource requires input from another resource defined in any of the Bicep files. The fundamental reason for this is that a Bicep parameter file cannot read from Bicep files to retrieve output and use it as input.

Listing 4.1: Figure showcasing the use of boolean `useVirtualNetworkSubnetID` together with the ternary operator to either assign a subnet to the App Service resource or set it to null.

```
resource appService 'Microsoft.Web/sites@2023-01-01' = {
  name: appServiceName
  location: location
  identity: {
    type: 'SystemAssigned'
  }
  properties: {
    serverFarmId: appServicePlan.id
    hostNameSslStates: hostNameSslStates
    clientAffinityEnabled: clientAffinityEnabled
    publicNetworkAccess: usePublicNetworkAccess ? publicNetworkAccess
        : null
    virtualNetworkSubnetId: useVirtualNetworkSubnetId ?
       networkInfrastructureSubnetId : null
    vnetRouteAllEnabled: vnetRouteAllEnabled
    keyVaultReferenceIdentity: 'SystemAssigned'
  }
}
```

Conditional logic is powerful and can be used to generalize Bicep templates. Conditional logic was utilized to solve the following template generalization problems between resource groups:

- different property values for resources
- different set up of resources and child resources in the Bicep files

However, this caused issues with readability and maintainability.

A boolean was created in the Bicep module, in this case it is called `useVirtualNetworkSubnetId`, however the value is assigned through a Bicep parameter file. If the boolean evaluates to `true`, it will select the first value of the ternary operator which in this case is `networkInfrastructureSubnetId`. If it evaluates to `false` it will select the second value which in this case is `null`. This approach solves the problem of having few Bicep modules to maintain, however, it shifts the problem of maintainability and readability elsewhere. Indeed, there are few files to maintain, however a significant number of boolean parameters must be created in the Bicep modules and parameter files, which creates bloat. This was laborious and poorly readable.

Listing 4.2: Figure shows how to use the boolean parameter `RG` along with an if-statement and logical OR operator. This is to ensure that the Virtual network resource only deploys RG equates to `'genericapi'` or `'genericshop'`. Notice that the properties `addressPrefixes` and `dnsServers` are given objects `vnetAddressPrefixes` respectively `dnsServers` as input which are from a Bicep parameter file.

```
resource vnet 'Microsoft.Network/virtualNetworks@2023-09-01' = if (RG
    == 'genericapi' || RG == 'genericshop') {
  name: vnetName
  location: location
```

```
    properties: {
        addressSpace: {
            addressPrefixes: vnetAddressPrefixes
        }
        dhcpOptions: {
            dnsServers: dnsServers
        }
        enableDdosProtection: false
    }
    dependsOn: [
        networkSecurityGroup
        natGateway
    ]
}
```

Boolean parameters can be used with if-statements and logical operators to enable different arrangements of resources and child resources based on which resource group is being deployed. In Listing 4.2 is shown a virtual network resource which will only deploy if the boolean parameter `RG` is set to the strings `'genericapi'` or `'genericshop'`. `RG` like other parameters, is assigned through a Bicep parameter file. This approach, although powerful, scaled poorly in complexity as more resource groups were added. The first resource group that the Bicep modules were created for was `genericapi`. During the process of generalizing the modules to make possible deployment of `genericshop`, complexity increased quickly. The large number of if-statements to ensure that each resource group only deploys the correct resources and child resources made this solution unfeasible. This issue worsens considering that additional resource groups will be added in the future, given that this project does not process all of New Wave Group's resource groups.

The implication of these hurdles is not that conditional logic has no uses; certain problems are appropriately solved using conditional logic. The issue lies in using conditional logic to solve the problem of generalizing templates to a high degree. In order to continue, a redesign was required.

4.2 Final Design

This section describes the final design while focusing on the general structure, layout and design philosophy. Specific resource properties and parameter values are not discussed in detail due to being trivial. Instead, we emphasize important implementation details and Bicep's syntactic capabilities through select examples. Finally, the automation of deployments using Azure Pipelines is described. To avoid repetition, this section mainly presents examples from the `genericshop` resource group files.

4.2.1 Folder structure

The final design of the IaC is straightforward and simple to understand as shown in Figure 4.5. Instead of making general Bicep files that will be used for all deployments, each resource group will have its corresponding folder with its own versions of `AppHosting.bicep`, NetworkInfrastructure.bicep, main.bicep and Bicep parameter file. This increases modularity, cohesion and readability while reducing parameter and conditional logic bloat, however it comes at the cost of code duplication.

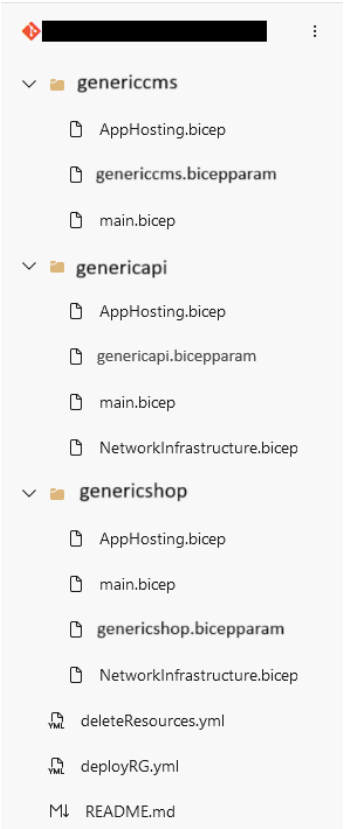

Figure 4.5: Azure repository containing the folders corresponding to each resource group. Additionally, the YAML files `deployRG.yml` and `deleteResources.yml` are shown which are used with Azure Pipelines to deploy the resource groups

This design is easier to maintain as making a change to a resource group is done in its own folder in a straightforward fashion. Adding additional resource groups is more convenient because each one has its own copies of the Bicep files and does not need to consider side effects on other resource groups when modified. Since this design iteration does not use conditional logic (for generalization), each resource group's files has only its required resources, child resources, properties and parameters, leading to significantly reduced bloat. Furthermore, in case a resource group needs any additional modules with distinct resources, it is trivial to create a new one in the folder of said resource group.

4.2.2 main.bicep

main.bicep

genericshop > main.bicep > ...

```
// AppHosting module
param location string
param IngestionMode string
param appServiceName string
param appServiceSettings object
param appServiceConfigSettings object
param appServicePlanName string
param appServicePlanSku object
param applicationInsightsName string

// NetworkInfrastructure module
param vnetName string
param routes array
param subnets array
param vnetAddressPrefixes array
param dnsServers array

module appHosting 'AppHosting.bicep'= {
  name: 'appHostingModule-${uniqueString(resourceGroup().id)}' // the name for the deployment, not the name in Azure portal
  params: {
    location: location
    IngestionMode: IngestionMode
    appServiceName: appServiceName
    appServiceSettings: appServiceSettings
    appServiceConfigSettings: appServiceConfigSettings
    appServicePlanName: appServicePlanName
    applicationInsightsName: applicationInsightsName
    appServicePlanSku: appServicePlanSku
  }

  dependsOn: [
    storageAccount
    networkInfrastructure
  ]
}

module networkInfrastructure 'NetworkInfrastructure.bicep' = {
  name: 'networkInfrastructureModule-${uniqueString(resourceGroup().id)}'
  params: {
    location: location
    vnetName: vnetName
    routes: routes
    subnets: subnets
    vnetAddressPrefixes: vnetAddressPrefixes
    dnsServers: dnsServers
  }
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2023-01-01' existing = {
  name: 'stdntlabdummystrgacc001'
}
output storageAccountId string = storageAccount.id
```

Figure 4.6: The `main.bicep` file of the resource group `genericshop`. Parameters are defined at the top, modules are imported using module keyword and a resource is created using resource keyword.

The `main.bicep` file is the central file which imports the modules and defines all necessary parameters for the deployment. Importing of modules can be used alongside the creation of new resources, such as the `storageAccount` resource shown in Figure 4.6. The purpose of this storage account resource will be explained in a later section. Notice the `dependsOn` clause in the `appHosting` module which ensures that the `networkInfrastructure` module is deployed first. This is useful when the resources in a module may need resources in another module to exist before deploying to ensure no inconsistencies.

Importing all modules in a central Bicep file makes it convenient for a module to use the output of another module as input. Although this is not utilized in the final design, Listing 4.3 and Figure 4.10 contain snippets from a previous design, showing how input and output between modules is utilized. In accordance with New Wave Group, this subnet was not to be deployed, hence why this output is not used in the final design.

Listing 4.3: `appHosting` module using the output of `networkInfrastructure` module as input. Both modules are in the `main.bicep` file. From previous design iteration.

```
52  module appHosting ' .. /Shared modules/AppHosting.bicep'= {

92          networkInfrastructureSubnetId: networkInfrastructure.outputs.
                snet02Id
93      }
94
95      dependsOn: [
96          networkInfrastructure
97      ]
98  }
99
100 module networkInfrastructure ' .. /Shared modules/
        NetworkInfrastructure.bicep' = {
101     name: 'networkInfrastructureModule-${uniqueString (resourceGroup
            ().id)}'
102     params: {
```

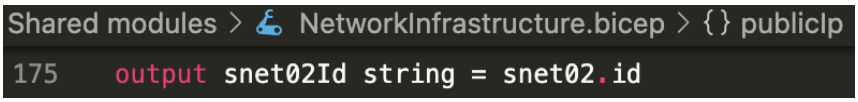

Figure 4.7: Inside of the `networkInfrastructure` file. An output of the subnet resource ID is created for other modules to use. From previous design iteration.

4.2.3 AppHosting.bicep

Listing 4.4: `appHosting.bicep` for the `genericshop` resource group. Figure shows the resource definition for an app service plan and an app service. Noteably, the app service uses the id of the `appServicePlan` resource and the `appServiceSettings` object to assign its property values. `appServiceSettings` is an object containing property values that is assigned from a Bicep parameter file.

```
resource appServicePlan 'Microsoft.Web/serverfarms@2023-01-01' = {
    name: appServicePlanName
    location: location
    sku: appServicePlanSku

param appServiceSettings object

resource appService 'Microsoft.Web/sites@2023-01-01' = {
    name: appServiceName
    location: location
    identity: {
        type: 'SystemAssigned'
    }

    properties: {
    serverFarmId: appServicePlan.id // from modules
    hostNameSs lStates: appServiceSettings. hostNameSslStates
    clientAffinityEnabled: appServiceSettings.clientAffinityEnabled
    vnetRouteAllEnabled: appServiceSettings. vnetRouteAllEnabled
    httpsOnly: appServiceSettings.httpsOnly

    keyVaultReferenceIdentity: 'SystemAssigned'
}
```

Listing 4.5: the `appServiceSettings` object defined in the Bicep parameter file for `genericshop`.

```
    param appServiceSettings = {
        hostNameSslStates: hostNameSslStates
        clientAffinityEnabled: true
        vnetRouteAllEnabled: true
        httpsOnly: true
    }
```

Listing 4.4 shows how a resource can use the ID of another resource as a property value, and how a Bicep object can be used to conveniently assign property values. An app service plan resource and an app service resource are defined. The app service resource uses the ID of the app service plan as the property value of `serverFarmId`.

This is achieved using the syntax `serverFarmId: appServicePlan.id`. Furthermore it is shown how the `appService` resources uses the `appServiceSettings` object which is a parameter. The value of `appServiceSettings` is created in and set by the Bicep parameter file, shown in Listing 4.5, for the resource group. The values of `appServiceSettings` are accessed using dot notation, for instance `appServiceSettings.httpsOnly` which in this case is set to `true`.

Notably, is it possible to set the `properties` clause to `properties: appServiceSettings` as discussed in 4.1.2.1. The issue is that it would not be possible to set the `serverFarmId` to `appServicePlan.id` as that value comes from a different source than the Bicep parameter file.

4.2.4 NetworkInfrastructure.bicep

Listing 4.6: The for-each loop in NetworkInfrastructure.bicep that dynamically creates subnet resources from the `subnets` array which is a parameter assigned from the Bicep parameter file.

```
1  // for-each loop, subnets are defined in the .bicepparam file for the
       main.bicep which contains this module
2  param subnets array
3  @batchSize(1)
4  resource snet 'Microsoft.Network/virtualNetworks/subnets@2023-09-01'
       = [for s in subnets: {
5      parent: vnet
6      name: '${vnetName}-${s.name}
7
8      properties: {
9          addressPrefix: s.addressPrefix
10         routeTable: {
11         id: routeTable.id
12
13         serviceEndpoints: s.serviceEndpoints
14         delegations: contains(s, 'delegations') ? s.delegations :
               null
15         privateEndpointNetworkPolicies: s.
               privateEndpointNetworkPolicies
16         privateLinkServiceNetworkPolicies: s.
               privateLinkServiceNetworkPolicies
17 }]
```

In the NetworkInfrastructure.bicep exists a segment which combines several Bicep features to create subnet resources in a powerful and dynamic manner. The virtual network for the `genericshop` resource group contains more than one subnet resource. It is possible to define each subnet as its own resource, however this introduces code duplication and severely hampers future changes to the setup of subnets in the resource group. Instead, a for-each loop is used along with the parameter

`subnets` which is an array containing objects which contain the property values for each subnet, as seen in Listing 4.7.

Listing 4.7: The subnets parameter in the Bicep parameter file for `genericshop`. Two objects are defined within the array. Each object contains the necessary property values for each subnet to deploy as resources in NetworkInfrastructure.bicep. The address prefixes are hidden and the subnet names are replaced with false names. For clarification, each pair of outmost curly brackets corresponds to one object.

```
param subnets = [
    {
        name: 'genericsubnet1'
        addressPrefix: 'xx.x.xx.x/xx'
        useNatGateway: false
        serviceEndpoints: [
            {
                service: 'Microsoft.Storage'
                locations: [
                    westeurope
                    'northeurope'
                ]
            }
        ]
        privateEndpointNetworkPolicies: 'Enabled'
        privateLinkServiceNetworkPolicies: 'Enabled'
    }
    {
        name: 'genericsubnet2'
        addressPrefix: 'x.x.xx.xx/xx'
        useNatGateway: false
        serviceEndpoints: [
            {
                service: 'Microsoft.Storage'
                locations: [
                    westeurope'
                    northeurope
                ]
            }
        ]
    privateEndpointNetworkPolicies: 'Enabled'
    privateLinkServiceNetworkPolicies: 'Enabled'
    }
]
```

The subnet for-each loop contains two additional noteworthy elements, the `contains` Bicep function and the `batchSize` decorator.

The `delegations` property makes use of the Bicep function `contains` which in

this case takes an object as the first argument and a string as the second argument. Some subnets may want to include the optional `delegations` property, which is made possible by using the `contains` function along with the ternary operator. The way it functions is straightforward; the `contains` functions evaluates to `true` if the object that represents the property values for the subnet contains the `delegations` property, otherwise it evaluates to `false`. If it evaluates to `true` the `delegations` property is set to `s.delegations`, where `s` is one of the objects in the array for each iteration, otherwise `delegations` is set to `null`.

The `batchSize` decorator solves an interesting problem. As the resource group was being test deployed, some subnets would not deploy properly. This was due to concurrency issues as for-each loops in Bicep make deployments in parallel. Setting the `batchSize` decorator with a value of `1` ensures that the for-each loop will deploy the resources in batches of 1 which is equivalent to sequential deployment.

4.2.5 genericshop.bicepparam

Bicep parameter files are straightforward; they contain values for properties that are used as input to the parameters in the `main.bicep` file which are then forwarded to each module and resource in `main.bicep`. Important snippets from the `genericshop.bicepparam` have been shown in previous section, however one further detail is the `using` keyword as shown in Listing 4.8. This simply sets which file will use the `.bicepparam` file to assign its parameter values, which in this case is the `main.bicep` as it contains all parameters for deployment.

Listing 4.8: The `using` keyword in `genericshop.bicepparam` determines which Bicep file will use the Bicep parameter file to give input to its parameters.

```
1 using '../genericshop/main.bicep'
```

4.3 Automation Using Azure Pipelines

This section describes the creation of two pipelines, one for deployment of resources and one for deletion of resources.

4.3.1 Deployment of resources

As shown in Figure 4.7, the Azure repository that contains the files for the resource groups also includes two YAML files, `deployRG.yml` and `deleteResources.yml`.

4.3.1.1 deployRG.yml parameters

Conveniently, given that YAML allows for parameters as shown in Listing 4.9, one YAML file is enough to deploy the different resource groups and to delete resources within them. It is as simple as changing the parameters when running the pipeline.

Listing 4.9: The `trigger` in the `deployRG` file is set to `none` since New Wave Group wants to eventually trigger the Pipelines through webhook API calls outside of the scope of this project. The parameters shown are what enables one YAML file to be used for deployments of different resource groups.

```
trigger: none

parameters:
- name: azureSubscription
  displayName: 'Azure Subscription'
  type: string
  default: "genericshop"

- name: resourceGroup
  displayName: 'Resource Group'
  type: string
  default: "studentlab"

- name: parameters
  displayName: 'Bicepparam File'
  type: string
  default: "genericshop/genericcms.bicepparam"

- name: name
  displayName: 'Deployment Name'
  type: string
  default: "testDeploygenericcms"

- name: storageAccName
  displayName: 'Storage Acc Name'
  type: string
  default: "stdntlabdummystrgacc001"
```

4.3.1.2 deployRG.yml Azure CLI task for deployment

The powerful part of YAML is that Azure CLI tasks can be created as shown in Listing 4.10. The task runs an `inlineScript` which is identical to a user manually running the command in their own Azure CLI. In this case, the `inlineScript` runs the command to deploy a resource group, or deploy into an already existing one. The script is self-explanatory aside from the lack of command to specify which `main.bicep`, which contains all the modules, to execute for deployment. This is instead handled by the `-parameters` flag which points to which `.bicepparam` file to use. As discussed in section 4.2.5, the `.bicepparam` file contains the `using` keyword, which points to the correct `main.bicep` file. This renders it unnecessary to specify in the CLI command which `main.bicep` file to use for deployment.

Listing 4.10: An Azure CLI task that can run Azure CLI commands, such as deployment of resources and resource groups.

```
1 - task: AzureCLI@2
2     inputs:
3         azureSubscription: ${{ parameters.azureSubscription }}
4         scriptType: 'bash'
5         scriptlocation: 'inlineScript'
6         inlineScript: |
7             az deployment group create -- resource-group $({
                parameters.resourceGroup ]] -- parameters $({
                parameters.parameters )] -- name $(( parameters.name ])
8         displayName: 'ARM (Bicep) template deployments of resource
              group to studentlab'
```

4.3.1.3 genericshop specific Azure CLI tasks

Listing 4.11: Azure CLI task in `deployRG.yml` that deploys a dummy storage account into `genericshop` before the other resources are deployed. Notably, this task is defined above the resource group deployment task in Listing 4.10 and hence executed beforehand.

```
# For demonstration purpose
steps:
- task: AzureCLI@2
    inputs:
        azureSubscription: ${{ parameters.azureSubscription }}
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
            az storage account create -- resource-group ${{ parameters
                .resourceGroup }} -- location 'westeurope' -- name ${{
                parameters. storageAccName }} -- sku Standard_LRS --
                kind StorageV2 -- access-tier Cool
    condition: eq('${{ parameters.name }}', 'testDeploygenericshop')
    displayName: 'Dummy storage account creation for resource group'
```

Listing 4.12: Azure CLI task in `deployRG.yml` that deploys a child resource for the App Service in `genericshop`. The task is defined below the resource group deployment task in Listing 4.10 and hence executed afterwards.

```
# Moved from defined in AppHosting.bicep to pipeline task due to
    issue where it would sometimes fail to deploy for unknown reason.
- task: AzureCLI@2
    inputs:
        azureSubscription: ${{ parameters.azureSubscription }}
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
            az resource create -- resource-group ${{ parameters.
                resourceGroup }} -- resource-type 'Microsoft.Web/sites/
                siteextensions' -- name '${{ parameters.resourceGroup
                }}- app01/siteextensions/Microsoft.AspNetCore.
                AzureAppServices.SiteExtension' -- api-version
                '2023-01-01' -- properties '{}'
    condition: eq('${{ parameters.name }}', 'testDeploygenericshop')
    displayName: 'Add ASP.NET Core Logging Integration site extension
        for app service'
```

Two additional Azure CLI tasks are defined in `deployRG.yml` that run when the `name` parameter in `deployRG.yml` is set to `testDeploygenericshop`. In accordance with New Wave Group, the storage account resources are not deleted and therefore not redeployed. Instead they remain in the resource groups. The purpose of the storage account deployment task in Listing 4.11 is to demonstrate to New Wave Group that newly deployed resources can be connected to already existing ones. First, the storage account is created, then the newly deployed `genericshop` connects to the existing dummy storage account. This is done by using the `existing` keyword in `main.bicep` for `genericshop` when deploying a storage account resource, as shown in Listing 4.13. The `name property` of said storage account resource must be set to the name of the existing storage account in `genericshop`.

Listing 4.13: Resource definition of dummy storage account in `main.bicep` of `genericshop`. Additionally, for demonstration purposes the ID of the resource is set as an output.

```
1 resource storageAccount 'Microsoft.Storage/storageAccounts@2023
     -01-01' existing = {
2     name: 'stdntlabdummystrgacc001'
3 }
4 output storageAccountId string = storageAccount.id
```

The purpose of the Azure CLI task in Listing 4.12 is to deploy a child resource for the app service resource of `genericshop`. Attempting to define this child resource in the Bicep files caused occasional failure of said child resource in deployments. The reason for this is unknown but is suspected to be concurrency related. Therefore, the child resource is deployed separately after the deployment of the resource group using the Azure CLI, which makes it successfully deploy in all deployments.

4.3.1.4 Pipeline to execute deployRG.yml

After the creation of the YAML file, a pipeline is created that is connected to `deployRG.yml`. This was done easily following Microsoft's basic guide for creating an Azure Pipeline through the Azure Portal [14]. When done, pipelines exist in the Azure Portal that can be clicked on and run manually through the `Run pipeline` button.

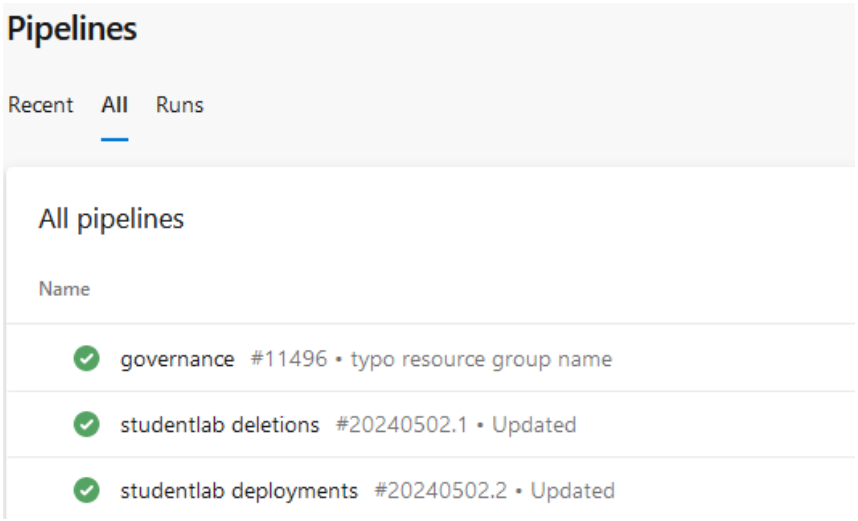

Figure 4.8: Figure showing existing pipelines. The pipelines "studentlab deletions" and "studentlab deployments" are a part of this project whereas "governance" is unrelated.

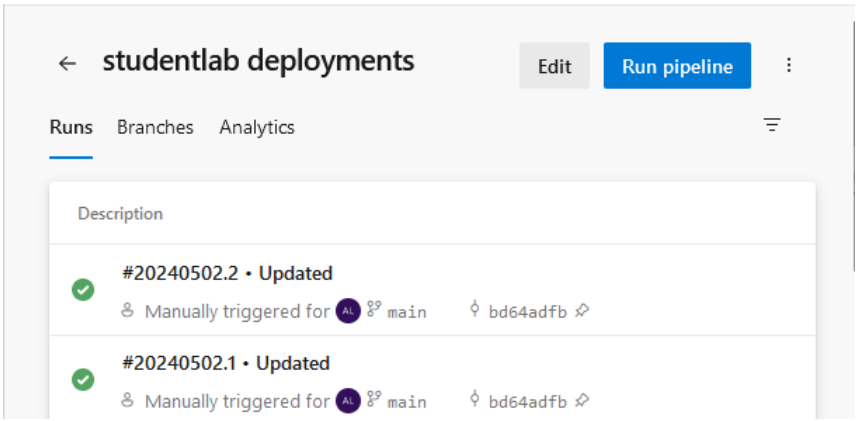

Figure 4.9: Figure showing the interface after clicking on "studentlab deployments" with the `Run pipeline` button and previous pipeline runs. Notably, the successful pipeline runs indicated by the green checkmarks were triggered through a Microsoft Teams webhook by New Wave Group.

When the Pipeline is run it triggers the `deployRG.yml` file which executes the Azure CLI task that deploys the resource group specified by the YAML parameters. In the case of `genericshop`, two additional Azure CLI tasks are executed as discussed in

4.3.1.3. In order for the pipeline to communicate with `deployRG.yml` in the Azure repository, the `azureSubscription` parameter in `deployRG.yml` must be set to the name of an Azure service connection. An Azure service connection is used to enable communication between the pipeline and Azure resources, and was created easily by following Microsoft's basic guide [15]. Although the pipeline can be run manually in the Azure Portal, New Wave Group intends to run it using a Microsoft Teams webhook which is outside the scope of this project.

4.3.2 Deletion of resources

The method to delete resources is conducted in a straightforward manner using `deleteResources.yml` which contains Azure CLI tasks, and a separate pipeline.

4.3.2.1 deleteResources.yml parameters

Listing 4.14: The parameters in `deleteResources.yml`. `azureSubscription` is to connect to the Azure service connection. `resourceGroup` is to specify which resource group to delete resources from.

```
trigger: none

parameters:

  - name: azureSubscription
    displayName: 'Azure Subscription'
    type: string
    default: "studentlab"

  - name: resourceGroup
    displayName: 'Resource Group'
    type: string
    default: "genericcms"
```

Listing 4.15: Three Azure CLI tasks in `deleteResources.yml` that remove resources from resource groups. The first task is for `genericapi`, the second task is for `genericshop` and the third task is for `genericcms`. Only one task at a time is executed. The task to be executed is dictated by value of the `resourceGroup` parameter.

```
1  # Deletion order matters due to dependencies between resources! Remove
       resources that depend first.
2  steps:
3    - task: AzureCLI@2
4      inputs:
5          azureSubscription: ${{ parameters.azureSubscription }}
6          scriptType: 'bash'
7          scriptLocation: 'inlineScript'
8          inlineScript: |
9              az resource delete -- resource-group studentlab -- name
                   studentlab-app01 -- resource-type 'Microsoft.Web/sites'
10             az resource delete -- resource-group studentlab -- name
                   studentlab-app01 -- resource-type 'Microsoft. Insights/
                   components'
11             az resource delete -- resource-group studentlab -- name
                   studentlab-nat01 -- resource-type 'Microsoft. Network/
                   natGateways'
12             az resource delete -- resource-group studentlab -- name
                   studentlab-pip01 -- resource-type 'Microsoft. Network/
                   publicIPAddresses'
13             az resource delete -- resource-group studentlab -- name
                   studentlab-plan01 -- resource-type 'Microsoft.Web/
                   serverFarms'
14             az resource delete -- resource-group studentlab -- name
                   studentlab-vnet01 -- resource-type 'Microsoft. Network/
                   virtualNetworks'
15             az resource delete -- resource-group studentlab -- name
                   studentlab-vnet01-route01 -- resource-type 'Microsoft.
                   Network/routeTables'
16             az resource delete -- resource-group studentlab -- name '
                   Failure Anomalies - studentlab-app01' -- resource-type '
                   Microsoft.AlertsManagement/smartDetectorAlertRules
17             az resource delete -- resource-group studentlab -- name ndn --
                   resource-type 'Microsoft.Network/networkSecurityGroups'
18     condition: eq('${{ parameters.resourceGroup }}', 'genericapi')
19     displayName: 'Deletion of resources from genericapi dev RG
20
21   - task: AzureCLI@2
22     inputs:
23         azureSubscription: ${{ parameters. azureSubscription }}
24         scriptType: 'bash'
25         scriptLocation: 'inlineScript'
26         inlineScript: |
27             az resource delete -- resource-group studentlab -- name '
```

```
            Failure Anomalies - studentlab-app01' -- resource-type '
            Microsoft.AlertsManagement/smartDetectorAlertRules
28          az resource delete -- resource-group studentlab -- name
            stdntlabdummystrgacc001 -- resource-type 'Microsoft.Storage
            /storageAccounts'
29          az resource delete -- resource-group studentlab -- name
            studentlab-app01 -- resource-type 'Microsoft.Web/sites'
30          az resource delete -- resource-group studentlab -- name
            studentlab-app01 -- resource-type 'Microsoft. Insights/
            components'
31          az resource delete -- resource-group studentlab -- name
            studentlab-plan01 -- resource-type 'Microsoft.Web/
            serverFarms'
32          az resource delete -- resource-group studentlab -- name
            studentlab-vnet01 -- resource-type 'Microsoft.Network/
            virtualNetworks'
33          az resource delete -- resource-group studentlab -- name
            studentlab-vnet01-route01 -- resource-type 'Microsoft.
            Network/routeTables'
34    condition: eq('${{ parameters. resourceGroup }}', 'genericshop')
35    displayName: 'Deletion of resources from genericshop dev RG'
36
37  - task: AzureCLI@2
38    inputs:
39        azureSubscription: ${{ parameters.azureSubscription }}
40        scriptType: 'bash'
41        scriptLocation: 'inlineScript'
42        inlineScript: |
43            az resource delete -- resource-group studentlab -- name '
            Failure Anomalies - studentlab-app01' -- resource-type '
            Microsoft.AlertsManagement/smartDetectorAlertRules
44            az resource delete -- resource-group studentlab -- name
            studentlab-app01 -- resource-type 'Microsoft.Web/sites'
45            az resource delete -- resource-group studentlab -- name
            studentlab-app01 -- resource-type 'Microsoft. Insights/
            components'
46            az resource delete -- resource-group studentlab -- name
            studentlab-plan01 -- resource-type 'Microsoft.Web/
            serverFarms'
47    condition: eq('${{ parameters.resourceGroup }}', 'genericcms')
48    displayName: 'Deletion of resources from genericcms dev RG'
```

`deleteResources.yml` is straightforward. As shown in Figure 4.8, two parameters exist. The first parameter `azureSubscription` is to connect the YAML file to a pipeline using an Azure service in the same manner as discussed in 4.3.1.4 for `deployRG.yml`. The same Azure service connection can be used for the deletion pipeline as the one for deployment. The second parameter `resourceGroup` is self-explanatory, it specifies which resource group to delete resources from.

4.3.2.2 deleteResources.yml Azure CLI tasks for deletion of resources

The approach to delete resources from a resource group is simple and powerful. Fundamentally, Azure CLI tasks are used to replicate the actions of a user manually selecting the resources in the Azure portal and clicking the `Delete` button. Each task shown in Figure 4.9 is responsible for deleting resources from one of the three resource groups.

A task works by containing a standard Azure CLI resource deletion command for each resource we want to delete from a resource group. For instance, the first task empties `genericapi` from resources using an `az resource delete` command for each resource that needs to be deleted. The flags are straightforward: `-resource-group` specifies which resource group to delete from, `-name` specifies the name of the resource to delete and `-resource-type` specifies the resource type. Specifying the resource type is important since two resources of different types can have the same name. Additionally, the value of `-resource-type` must be the name of the resource in Azure Resource Manager, not the colloquial name. For instance, `'Microsoft.Web/sites'` corresponds to an app service resource. Importantly, the deletion of resources must occur in a specific order in the case of dependencies. If resource A depends on resource B, resource B cannot be deleted until resource A is deleted. This situation occurs in the case of `genericapi`; the NAT gateway resource depends on the public IP address resource. The deletion command for the NAT gateway resource is therefore placed before the deletion command of the public IP address resource.

4.3.2.3 Pipeline to execute deleteResources.yml

A separate pipeline called "studentlab deletions" is created for `deleteResources.yml` as seen in Figure 4.8. The process of creating this pipeline was identical to creating the pipeline for `deployRG.yml` as described in 4.3.1.4 and the same Azure service connection is used. In the same manner as for deployments, this pipeline is intended by New Wave Group to be triggered through a Microsoft Teams webhook.

5

Results

This chapter presents a manual run of the pipeline `studentlab deployments` for `genericshop` followed by a run of `studentlab deletions` to delete the `genericshop` resources. The deployments and deletions target the sandbox `studentlab` resource group. `genericshop` is used in this section due to containing all interesting elements. The process for deployment and deletion of `genericshop` respectively `genericcms` is identical to that of `genericshop`.

5.1 Manual Run of Pipeline to Deploy `genericshop`

As shown in Figure 4.22, a pipeline can be run by clicking "Run pipeline". When clicked, a menu appears that allows you to override the default values of any YAML parameters as shown in Figure 5.1. In this case, the values are changed to deploy `genericshop`.

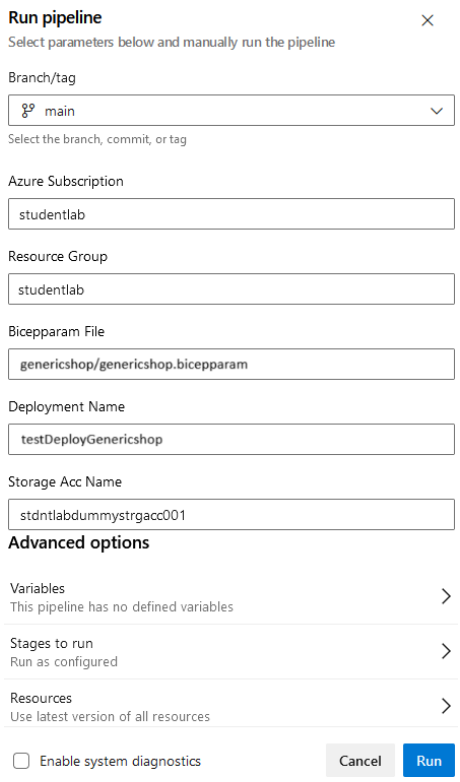

Figure 5.1: Menu to provide parameter values before pipeline run.

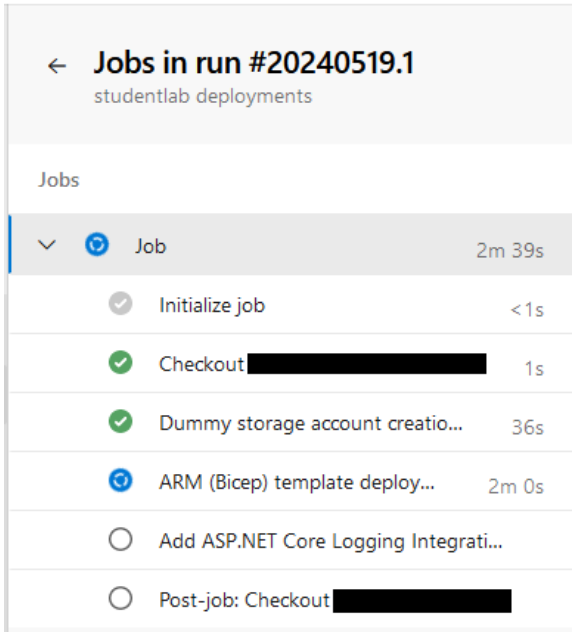

Figure 5.2: Different stages of the run of `studentlab deployments` for `genericshop`. After 36 seconds, the task that deploys the placeholder storage account is successful.

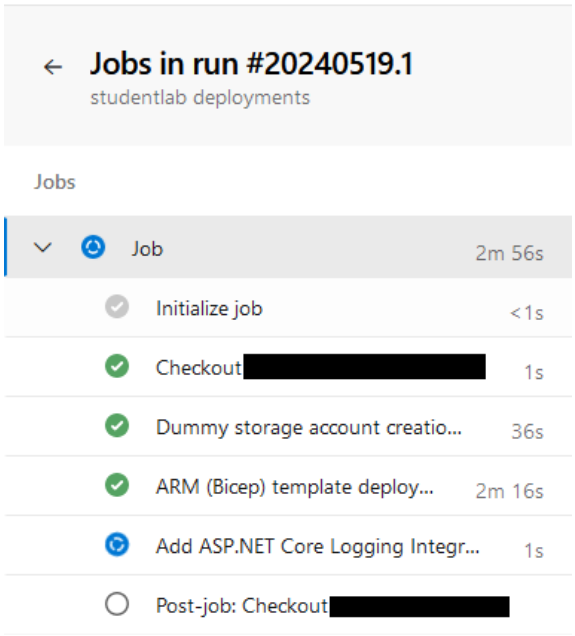

Figure 5.3: The second task which deploys the `genericshop` resources into `-studentlab` succeeds in 2 minutes and 16 seconds.

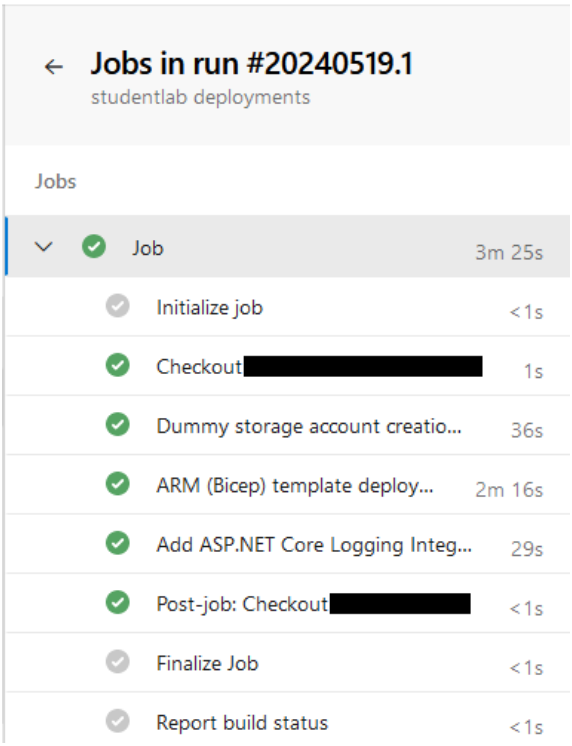

Figure 5.4: The task to add ASP.NET Core Logging Integration as a child resource in `genericshop` succeeds in 29 seconds and the pipeline run is complete.

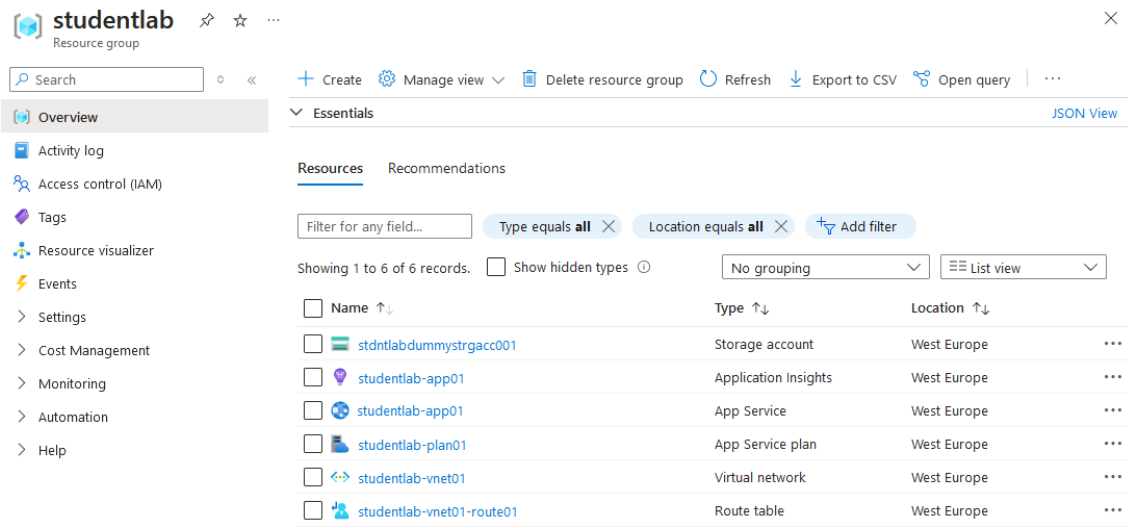

Figure 5.5: `studentlab` after successful deployment of `genericshop` resources.

Figures 5.2, 5.3 and 5.4 show the different stages after the pipeline is run. As per the structure of `deployRG.yml`, the first task to be run is the creation of a placeholder storage account. Afterwards, the `genericshop` resources are deployed, and finally the child resource that is added separately to `genericshop`, as discussed in 4.3.1.3. Finally, as seen in Figure 5.5, `studentlab` is now populated with the resources of `genericshop`, including the placeholder storage account. Compared to the actual resource group for `genericshop` in Figure 4.3, the resources `Shared dashboard` and `Smart detector alert rule` are missing. `Smart detector alert rule` is deployed automatically by Azure after an unspecified time period. `Shared dashboard` is, due to a lack of importance, not redeployed in this project.

5.2 Manual Run of Pipeline to Delete `genericshop` Resources from `studentlab`

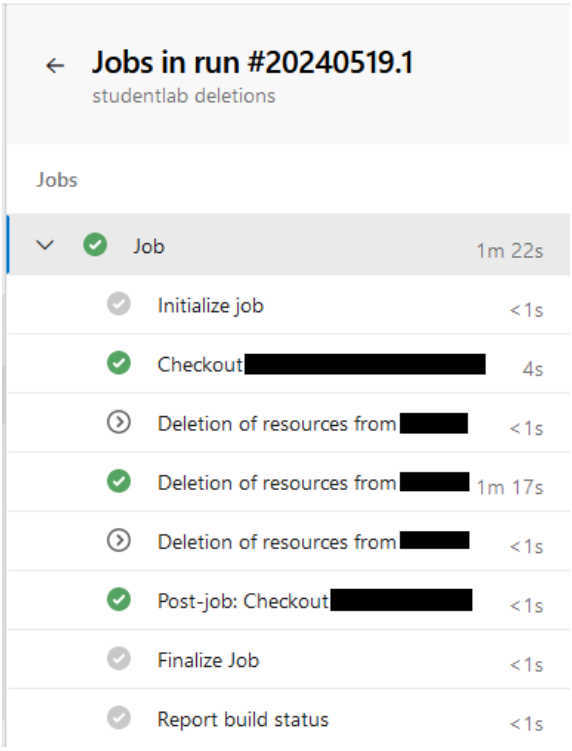

Figure 5.6: Successful pipeline run of `studentlab deletions` targeting `genericshop`. The conditional logic ensures only the resources for `genericshop` are deleted from `studentlab`.

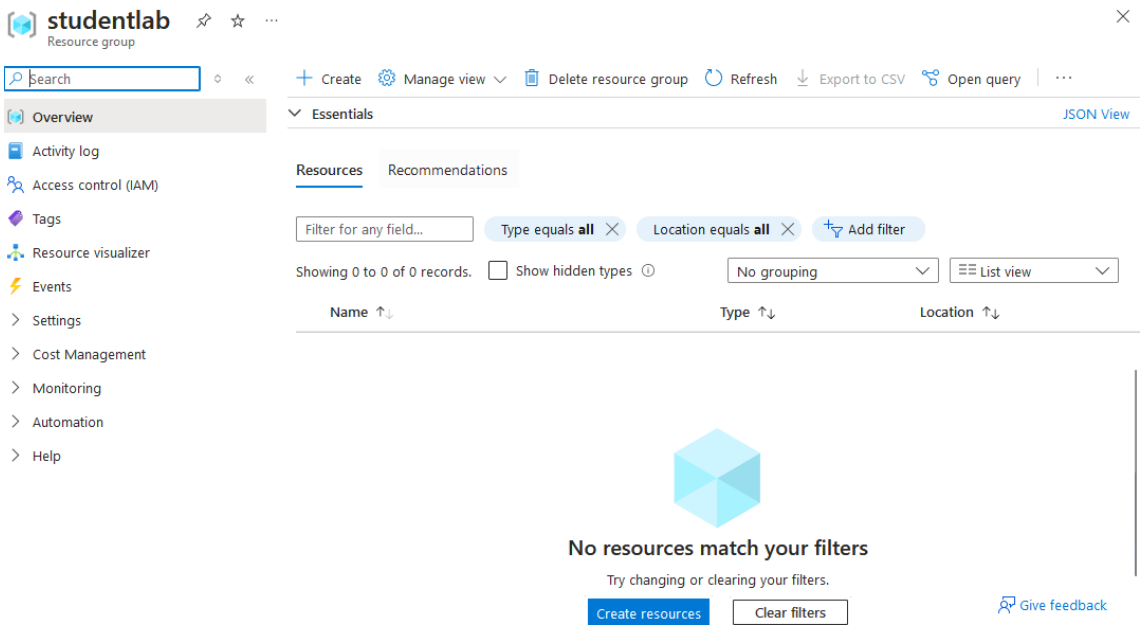

Figure 5.7: `studentlab` after successful run of `studentlab deletions`.

The run of `studentlab deletions` is similar to that of `studentlab deployments`. The value for the parameter `Resource group` in `deleteResources.yml` is set to `genericshop` for this run. As shown in Figure 5.6, among the three deletion tasks, only the one for `genericshop` is run as indicated by the green checkmark. This is due to the conditional logic in `deleteResources.yml`. After the run is successfully finished, `studentlab` is empty as seen in Figure 5.7.

5.3 Economic Impact of the Project

The key objective for this project is to reduce the operational expenses incurred by the Azure resources of New Wave Group. This project has not fully met this objective, however substantial progress has been made. This project demonstrates, as a proof of concept, that using automation can fulfill this objective. An economic analysis can be made assuming that New Wave Group further develops the proof of concept to work with their real dev environment. This would entail adding the missing resources, especially the databases, and setting up a proper configuration of parameters based on their own specifications.

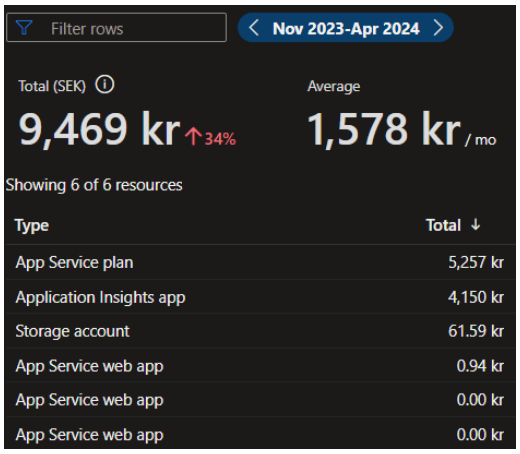

Figure 5.8: Cost for `genericshop` the last six months.

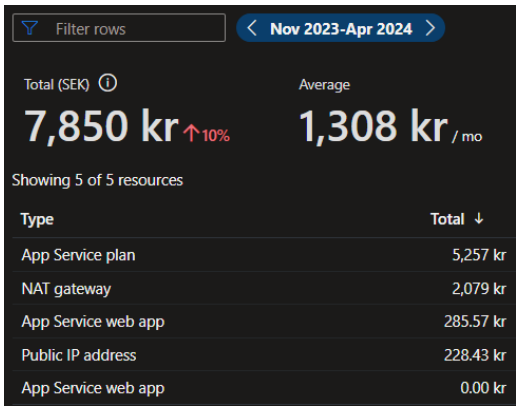

Figure 5.9: Cost for `genericapi` the last six months.

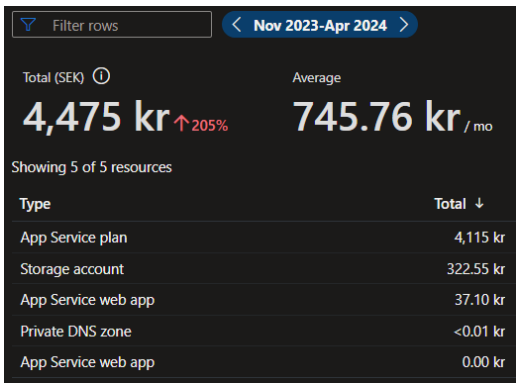

Figure 5.10: Cost for `genericcms` the last six months.

Figure 5.11: Cost for `genericstorage` the last six months.

Figure 5.12: Costs for all four resource groups over the last six months. Total sum of average cost per month is 8,840.76 SEK.

New Wave Group utilizes their resource groups approximately 2 days per month, with each session lasting approximately 6 hours per day. Over the past six months, their monthly cost of `genericshop`, `genericapi`, `genericcms`, and `genericstorage`

has averaged 8,840.76 SEK, as illustrated in Figure 6.5 and equation 6.1. This results in a cost of 294.692 SEK per day as shown in equation 6.2. Utilisation of 12 hours per month is equal to 147.346 SEK/month, as shown in equation 6.3.

$$1578_{\text{genericshop}} + 1308_{\text{genericapi}} + 745.76_{\text{genericcms}} + 5,209_{\text{genericstorage}} = 8,840.76 \quad (5.1)$$

$$\text{Daily cost (30-day month)} = \frac{8,840.76\,\text{SEK/month}}{30\,\text{days/month}} = 294.692\,\text{SEK/day} \quad (5.2)$$

$$\begin{aligned} \text{Daily cost (6 hours)} &= \frac{294.692\ \text{SEK/day}}{4\,\text{parts}} = 73.673\,\text{SEK/day} \\ \text{2 days cost (12 hours)} &= 73.673\,\text{SEK/day} \times 2 = 147.346\,\text{SEK} \end{aligned} \quad (5.3)$$

Based on these calculations, New Wave Group can save approximately 8,840.76 - 147.346 = 8,693.414 SEK per month by further developing the implementation to automate the dev environment resource groups corresponding to `genericshop`, `genericapi`, `genericstorage` and `genericcms`. Furthermore, additional savings can be achieved by using the proof of concept to automate resource groups not included in this project.

6

Discussion

6.1 Testing of the Implementation

One of the specified goals of this project is

- Conducting comprehensive testing to ensure reliability, security and efficiency of the implemented solution.

The main purpose of testing is to confirm that the code produces the expected results. Bicep code does not include built-in testing, such as the commonly known unit tests that exist for object-oriented languages. Bicep cannot understand what the user intends to deploy, and infrastructure is too complex and interconnected to simply test with code. Instead, testing is conducted by either simulating deployments or by doing real deployments. Deployments are conveniently simulated through the built-in `what-if` flag [16]. A deployment into a resource group with the `what-if` flag will not deploy the resources into the resource group. Instead, the Azure CLI will print out any additions, modifications and deletions to the resource group in case the deployment is made. This result needs to then be evaluated by a human.

In this project, unrestricted deployments into `studentlab` were allowed and deployments were not an exact replica of their real counterparts. Due to these reasons, the `what-if` flag was not utilized to conduct testing. Instead, manual user testing was conducted by deploying the resource groups into `studentlab`. Thereafter, it was manually checked that deployed resources and their configurations were similar, but not necessarily identical to, their real counterparts. And additionally that their configurations aligned with what was specified in the Bicep files. The test deployments into `studentlab` aligned with expectations and were therefore satisfactory as a proof of concept for the scope of this project.

Taking into consideration the nature of IaC, it can be concluded that the reliability and efficiency of the implemented solution was sufficient for this project. In order to further improve the reliability, New Wave Group can utilize decorators [17] on parameters to ensure that only allowed values (according to their own specifications) can be assigned. It was not deemed important for this project as the deployments were made into a sandbox environment. Testing the security of the implementation was not necessary as this project did not handle security sensitive information. Although, in cases where sensitive information is handled, Bicep includes functionality to ensure that such information is handled safely.

6.2 Workflow

The following section describes the workflow throughout the project, its efficiency and the time planning.

6.2.1 Communication and implementation

The workflow had a loose structure, but cooperation was maintained through constant communication and updates on incremental changes and plans. With no convenient access to an office, communication and cooperation with New Wave Group employees was facilitated primarily through a Microsoft Teams group chat. Booking meetings digitally was convenient as New Wave Group was flexible. Meetings were regularly held once or twice a week to discuss progress, specifications and the direction of the project such as limitations. Achieving the goals of this project was result-oriented; there were no strict requirements on how to build the solution, as long as results were achieved. The project members frequently worked in parallel attempting different solutions, and continuously discussing them. This resulted in a rich and comprehensive evaluation of different design choices leading to the final design.

6.2.2 Time planning

The overall planning of the project proceeded smoothly; however, there was a one-week delay in week five as shown in Figure 6.1 due to the extensive material in Microsoft's learning modules for Bicep and pipelines. The associated exercises required more time to complete than anticipated, contributing to a delay in the subsequent five weeks. However, the thorough reading gained from the modules made the development phase smooth, which brought the project timeline to get back on track by the middle of the main phase. The project members and New Wave Group agreed on finishing the development, which involved implementing Bicep files, Azure pipelines, and testing. Regarding documentation, the company considers the thesis report sufficient.

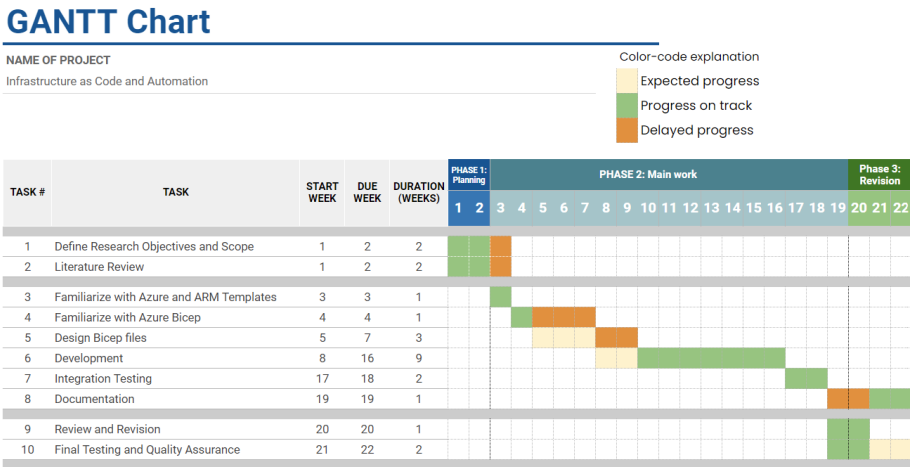

Figure 6.1: Gantt chart displaying the timeline of the project.

6.3 Sustainable Aspects in Automation of Azure Test Environment

This section addresses ethical and sustainability aspects in automating the Azure test environment.

6.3.1 Ethical aspects

Some databases include customer information, although in this project the database is not included due to time constraints. It's crucial to address ethics of data storage and anonymization, as mandated by regulations such as the General Data Protection Regulation (GDPR) [26]. Ethical use of data involves avoiding practices that may harm individuals or violate their rights, such as discriminatory or exploitative uses of data [27]. Prioritizing privacy protection by anonymizing customer information before storage is an ethical practice that helps protect sensitive data and preserves the identities of individuals.

6.3.2 Social aspects

This project does not engage with societal aspects due to its primary focus on technical automation and Azure deployment. While social considerations are critical in numerous projects, such considerations do not pertain to this project. The project's goals and results do not directly affect broader societal aspects.

6.3.3 Ecological aspects

Considering ecological factors is important in the IT industry as physical hardware consumes energy, which pertains to the secondary objective of this project. One key aspect is the energy efficiency of data centers, which consume significant energy, used by Azure [28]. Azure resources are hosted on the cloud in data centers, and by deleting them when not in use, the environmental footprint can be reduced. Additionally, using virtualization and cloud solutions decreased the need for physical hardware, reducing resource consumption and environmental impact [29, 30].

6.4 Suggestions for Future Work

This section presents potential areas for future work, specifically focusing on the unfinished tasks of deploying a storage account and a database.

6.4.1 Storage account

Storage accounts were not automated in this project due to specification from New Wave Group. This chapter describes a potential solution using AzCopy which is a powerful command-line tool designed for copying data to and from Azure storage

[31]. It can copy various types of data including blobs, files, tables, and queues between storage accounts. After the authorization credentials are provided, appending a shared access signatures (SAS) token [32] is recommended to be able to recursively copying data from a local directory to a blob container. After the identity has been authorized or a SAS token has been obtained, transfer of data can begin.

6.4.1.1 Copy containers, directories, and blobs

Copy all containers, directories, and blobs to another storage account by using the azcopy copy command.

```
azcopy copy
'https://<source-storage-account-name>.<blob or dfs>.core.windows.net/'
'https://<destination-storage-account-name>.<blob or dfs>.core.windows.net/'
--recursive
```

6.4.2 Potential approach to database deployment using bicep

Due to time constraints, the database deployment was not completed. The workflow involves anonymizing sensitive data and then copying the database from production to dev. This chapter outlines a potential solution for copying databases while keeping current data, adding more data, and also protecting sensitive data.

6.4.2.1 Anonymize sensitive data

Azure Key Vault can be utilized to store and anonymizing customer information [33]. Modify the Bicep file to include parameters for customer information, such as customer names, emails, or any other sensitive data that needs to be anonymized [34]. Use the Bicep file to deploy an Azure Key Vault instance. Once the Key Vault is created, store the sensitive customer information as secrets within the Key Vault. These secrets will be securely encrypted and can only be accessed by authorized applications or users [35].

To retrieve sensitive customer information from the Azure Key Vault, instead of hard coding the sensitive data in the template, fetch the data securely from the Key Vault using its URI or identifier [36].

6.4.3 Copying a database

A database copy is a transactionally consistent snapshot created using geo-replication technology [37]. It can be made on the same or a different server, with options to adjust backup redundancy and compute size. Copies can be made between service tiers with exceptions. Once completed, the copy is a fully functional, independent database with separate login, user, and permission management from the source [30].

6.4.3.1 Logins in the database copy

When copying a database to the same server, the original logins can be used on both databases, and the initiator of the copy operation becomes the owner of the new database [38]. If copying to a different server, which is our case, the initiator on the target server becomes the owner. All database users, permissions, and security identifiers (SIDs) are copied. For seamless access, contained database users must be used [39]; otherwise, server-level logins may encounter issues due to differences in logins or SIDs. After copying to a different server, only the database owner or server admin can initially access the new database, requiring subsequent login resolution.

6.4.3.2 Copying using Azure portal, Powershell, CLI

To copy a database using the Azure portal, navigate to the database's page, select "Copy," and proceed to fill in the required details for the destination server where the duplicate will be created.

To copy using Powershell:

```
New-AzSqlDatabaseCopy -ResourceGroupName "<resourceGroup>"
-ServerName $sourceserver -DatabaseName "<databaseName>" ‘
-CopyResourceGroupName "myResourceGroup" -CopyServerName $targetserver
-CopyDatabaseName "CopyOfMySampleDatabase"
```

To copy using Azure CLI:

```
az sql db copy --dest-name "CopyOfMySampleDatabase"
--dest-resource-group "myResourceGroup" --dest-server $targetserver
--name "<databaseName>" --resource-group "<resourceGroup>"
--server $sourceserver
```

6.4.3.3 Copy from production environment to dev environment

Since production and dev are in different subscriptions, Database1 needs to be copied from production to Database2 on dev, follow these steps:
1. Connect to the master database on dev.
2. Use a login that has the same username and password as the owner of Database1 on production.
3. Ensure that the login on dev is a member of the dbmanager role or is the server administrator login.

After ensuring these conditions, execute the command to copy the database. Be aware that the copying process might take some time, depending on the size of Database1. The following command copies Database1 on production to a new database named Database2 on dev. Depending on the size of your database, the copying operation might take some time to complete.

Execute on the master database of the target server (dev) to start copying from

production to dev

SQL:
CREATE DATABASE Database2 AS COPY OF production.Database1;

Ensure a login is made with the same username and password as the owner of the source database. Make sure this login is a member of the dbmanager role or is a server administrator on both the source and target servers.

```
--Create login and user in the master database of the source server.

CREATE LOGIN loginname WITH PASSWORD = 'xxxxxxxxx'
GO
CREATE USER [loginname] FOR LOGIN [loginname] WITH DEFAULT_SCHEMA=[dbo];
GO
ALTER ROLE dbmanager ADD MEMBER loginname;
GO

--Create the user in the source database and grant dbowner permission
to the database.

CREATE USER [loginname] FOR LOGIN [loginname] WITH DEFAULT_SCHEMA=[dbo];
GO
ALTER ROLE db_owner ADD MEMBER loginname;
GO

--Capture the SID of the user "loginname" from master database

SELECT [sid] FROM sysusers WHERE [name] = 'loginname';

--Connect to Destination server.
--Create login and user in the master database, same as of the source server.

CREATE LOGIN loginname WITH PASSWORD = 'xxxxxxxxx', SID = [SID of loginname
login on source server];
GO
CREATE USER [loginname] FOR LOGIN [loginname] WITH DEFAULT_SCHEMA=[dbo];
GO
ALTER ROLE dbmanager ADD MEMBER loginname;
GO

--Execute the copy of database script from the destination server
using the credentials created

CREATE DATABASE new_database_name
AS COPY OF source_server_name.source_database_name;
```

6.4.3.4 Resolve logins

After the new database is up and running on the target server, use the ALTER USER [40] statement to connect the users from the new database to the logins on the target server.

It's important to note that all users in the new database maintain the same permissions they had in the source database. Additionally, the user who initiated the database copy automatically becomes the database owner of the new database. Until the copying process completes successfully and other users are remapped, only the database owner will be able to log in to the new database.

6.4.3.5 Database copy errors

Here's a list of errors that can be encountered while copying a database in Azure SQL Database [38]:

Table 6.1: Azure SQL Database Copy Error Codes

Error Code	Description
40635	Client with IP address '%.*ls' is temporarily disabled.
40637	Create database copy is currently disabled.
40561	Database copy failed. Either the source or target database does not exist.
40562	Database copy failed. The source database has been dropped.
40563	Database copy failed. The target database has been dropped.
40564	Database copy failed due to an internal error. Please drop target database and try again.
40565	Database copy failed. No more than 1 concurrent database copy from the same source is allowed. Please drop target database and try again later.
40566	Database copy failed due to an internal error. Please drop target database and try again.
40567	Database copy failed due to an internal error. Please drop target database and try again.
40568	Database copy failed. Source database has become unavailable. Please drop target database and try again.
40569	Database copy failed. Target database has become unavailable. Please drop target database and try again.
40570	Database copy failed due to an internal error. Please drop target database and try again later.
40571	Database copy failed due to an internal error. Please drop target database and try again later.

7
Conclusion

In this project, Microsoft's domain-specific declarative language Bicep was used to recreate the software infrastructure of New Wave Group using infrastructure as code. Bicep files were created and can be deployed using Azure CLI commands to reinitialize specific resource groups. By using Azure Pipelines and YAML files with scripts that can run Azure CLI commands, the files can be conveniently deployed by running a pipeline. The resource group to deploy can be specified by assigning the appropriate YAML parameter values for the pipeline run. To complete the automation cycle, a pipeline for deleting resources was also created. The resource group to delete resources from is specified by assigning a YAML parameter value, in a similar manner as for deployments.

Due to time constraints, a proof of concept was all that was provided. Not all of New Wave Group's resource groups and resources were automated in this project. Furthermore, the infrastructure was deployed in a sandbox environment instead of the real development environments. The proof of concept was successful and can be built upon to automate the real development environment, with all of its resources and any additional resource groups.

Given that the deployments were made in a sandbox environment, the goals of reducing operational expenses and minimizing harm to the environment were not fully met. Despite this, the proof of concept is substantial progress to finally achieve these goals. Regarding testing of the program, Bicep does not have built-in testing functionality. Instead, considering that the sandbox environment did not affect the existing infrastructure of New Wave Group, deployments were made and manually validated.

Moving forward, it is recommended that the company further develop the proof of concept implementation to include additional resource groups and the missing resources omitted from this project, including databases. Importantly, the automation of databases must comply with legal and ethical requirements such as GDPR. By completing the automation of the resource groups included in this project, New Wave Group can save approximately 8690 kr per month according to an economic analysis.

Bibliography

[1] Microsoft, “What is infrastructure as code (IaC)?” 2022. [Online]. Available: https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code (accessed on: 2024-04-16)

[2] New Wave Group, “Delårsrapport för tredje kvartalet 2023,” 2023. [Online]. Available: https://www.nwg.se/press/delarsrapport-for-tredje-kvartalet-2023/ (accessed on: 2024-01-16)

[3] New Wave Group, “Kontakta oss,” [Online]. Available: https://www.nwg.se/kontakt/ (accessed on: 2024-01-16)

[4] New Wave Group, “Om New Wave Group,” [Online]. Available: https://www.nwg.se/om-new-wave-group/ (accessed on: 2024-01-16)

[5] Microsoft. “How does Azure work?” 2023. [Online] Available: https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/get-started/what-is-azure (accessed on: 2024-01-17)

[6] Microsoft, “Resource access management in Azure” 2023. [Online] Available: https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/get-started/how-azure-resource-manager-works (accessed on: 2024-01-17)

[7] Microsoft, “What is Azure Resource Manager?” 2023. [Online] Available: https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview (accessed on: 2024-01-17)

[8] Microsoft, “What are ARM templates?” 2023. [Online] Available: https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/overview (accessed on: 2024-01-17)

[9] McKinsey Digital, "Cloud adoption to accelerate IT modernization" 2018. [Online] Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/cloud-adoption-to-accelerate-it-modernization (accessed on: 2024-03-10)

[10] Atlassin, "Infrastructure as code". [Online] Available: https://www.atlassian.com/microservices/cloud-computing/infrastructure-as-code (accessed on: 2024-03-10)

[11] Atlassin, "Infrastructure as code". [Online] Available: https://www.atlassian.com/microservices/cloud-computing/infrastructure-as-code (accessed on: 2024-03-10)

[12] Microsoft, "What is Bicep?," 2022. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview?tabs=bicep (accessed on: 2024-02-13)

[13] TechTarget, "Compare Azure Bicep vs. ARM templates," 2022. [Online]. Available: https://www.techtarget.com/searchcloudcomputing/tip/Compare-Azure-Bicep-vs-ARM-templates (accessed on: 2024-02-13)

[14] Microsoft, "What is the Azure CLI?," 2024. [Online]. Available: https://learn.microsoft.com/en-us/cli/azure/what-is-azure-cli (accessed on: 2024-01-19)

[15] Microsoft, "What is the Azure Pipelines?," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops (accessed on: 2024-04-19)

[16] K21 Academy, "Azure Pipelines | Uses | It's Advantages," 2024. [Online]. https://k21academy.com/microsoft-azure/azure-pipelines-azure-devops-pipeline-concept-uses-its-advantages/ (accessed on: 2024-04-19)

[17] Microsoft, "Use Azure Pipelines," 2024 [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops (accessed on: 2024-04-19)

[18] Microsoft, "Specify jobs in your pipeline," 2024 [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=azure-devops&tabs=yaml

[19] Microsoft, "Fundamentals of Bicep," 2024 [Online]. Available: https://learn.microsoft.com/en-us/training/paths/fundamentals-bicep/ (accessed on: 2024-03-24)

[20] Atlassian, "What is Agile?," n.d. [Online]. Available: https://www.atlassian.com/agile (accessed on: 2024-01-19)

[21] E. M. Alzeyani and C. Szabó, "A study on the effectiveness of agile methodology using a dataset," Acta Electrotechnica et Informatica, vol. 23, no. 1, pp. 3–10, 2023. doi:10.2478/aei-2023-000 (accessed on: 2024-01-19)

[22] Microsoft, "Create your first pipeline," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/pipelines/create-first-pipeline?view=azure-devops&tabs=java%2Cbrowser (accessed on: 2024-05-12)

[23] Microsoft, "Connect to Azure by using an Azure Resource Manager service connection," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/pipelines/library/connect-to-azure?view=azure-devops (accessed on: 2024-05-12)

[24] Microsoft, "Bicep deployment what-if operation," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/deploy-what-if?tabs=azure-powershell%2CCLI (accessed on: 2024-05-14)

[25] Microsoft, "Understand the structure and syntax of Bicep files," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/file (accessed on: 2024-05-14)

[26] GDPR.EU, "What is GDPR, the EU's new data protection law?," 2024. [Online]. Available: https://gdpr.eu/what-is-gdpr/ (accessed on: 2024-05-16)

[27] DataCamp, "An Introduction to Data Ethics: What is the Ethical Use of Data?," 2024. [Online]. Available: https://www.datacamp.com/blog/introduction-to-data-ethics (accessed on: 2024-05-16)

[28] DatacenterDynamics, "Global data center electricity use to double by 2026 - IEA report," 2024. [Online]. Available: https://www.datacenterdynamics.com/en/news/global-data-center-electricity-use-to-double-by-2026-report/ (accessed on: 2024-05-16)

[29] TechStack, "How to Achieve Energy Efficiency and Sustainability in Cloud-Based Solutions," 2023. [Online]. Available: https://tech-stack.com/blog/how-to-achieve-energy-efficiency-and-sustainability-in-cloud-based-solutions/ (accessed on: 2024-05-16)

[30] ServerWatch, "What Is Green IT? Definition and Benefits," 2023. [Online]. Available: https://www.serverwatch.com/guides/green-it/ (accessed on: 2024-05-16)

[31] Microsoft, "Get started with AzCopy," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/common/storage-use-azcopy-v10 (accessed on: 2024-05-11)

[32] Microsoft, "Grant limited access to Azure Storage resources using shared access signatures (SAS)," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/common/storage-sas-overview (accessed on: 2024-05-11)

[33] Microsoft, "About Azure Key Vault," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/key-vault/general/overview (accessed on: 2024-05-11)

[34] Microsoft, "Quickstart: Create an Azure key vault and a key by using Bicep," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/key-vault/keys/quick-create-bicep?tabs=CLI (accessed on: 2024-05-11)

[35] Microsoft, "Quickstart: Set and retrieve a key from Azure Key Vault using Azure CLI," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/key-vault/keys/quick-create-cli (accessed on: 2024-05-11)

[36] Microsoft, "Azure Key Vault keys, secrets and certificates overview," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/key-vault/general/about-keys-secrets-certificates (accessed on: 2024-05-13)

[37] Microsoft, "Active geo-replication," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-sql/database/active-geo-replication-overview?view=azuresql (accessed on: 2024-05-13)

[38] Microsoft, "Copy a transactionally consistent copy of a database in Azure SQL Database," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-sql/database/database-copy?view=azuresql&source=recommendations&tabs=azure-powershell (accessed on: 2024-05-13)

[39] Microsoft, "Authorize database access to SQL Database, SQL Managed Instance, and Azure Synapse Analytics," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-sql/database/logins-create-manage?view=azuresql (accessed on: 2024-05-13)

[40] Microsoft, "ALTER USER (Transact-SQL)," 2023. [Online]. Available: https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-user-transact-sql?view=azuresqldb-current&preserve-view=true (accessed on: 2024-05-13)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY