DevOps Technical Documentation

Individual Project Semester 6
Veronika Valeva 4090349
Fontys, Eindhoven
16/5/2023

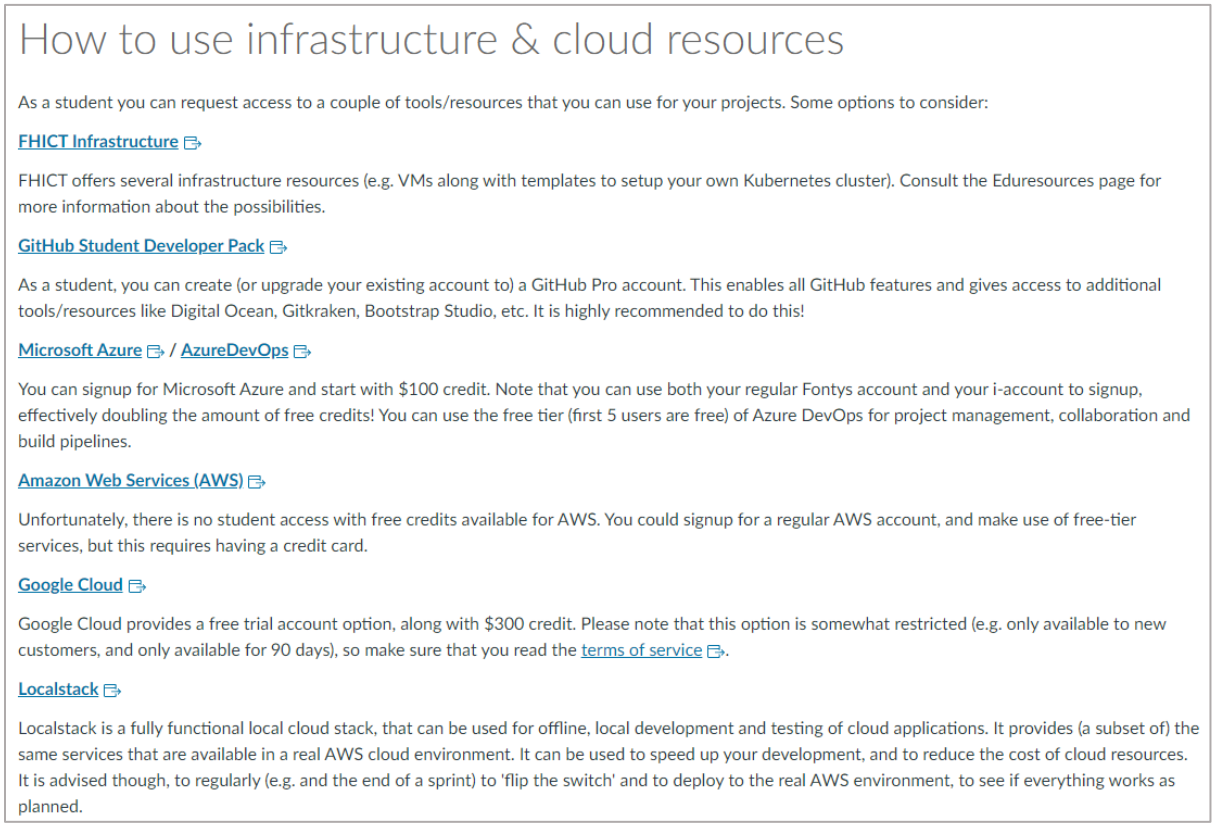
Table of Contents

Introduction

This document presents the development and deployment setup/architecture for the Quiz Game, which is the Semester 6 Individual Project. Since the Quiz Game project is currently scaled to a small university project, the objective of the DevOps architecture is to minimize costs while maintaining full functionality.

The project if going to have pipelines that will be established to host the codebase in GitLab for version control. They will utilize GitLab CI/CD for executing the CI/CD Pipeline, which involves building the application into Docker images. These images are then stored in Docker Hub repositories and subsequently deployed to Azure App Services.

Cloud Infrastructure



How to use infrastructure & cloud resources

As a student you can request access to a couple of tools/resources that you can use for your projects. Some options to consider:

FHICT Infrastructure

FHICT offers several infrastructure resources (e.g. VMs along with templates to setup your own Kubernetes cluster). Consult the Eduresources page for more information about the possibilities.

GitHub Student Developer Pack

As a student, you can create (or upgrade your existing account to) a GitHub Pro account. This enables all GitHub features and gives access to additional tools/resources like Digital Ocean, Gitkraken, Bootstrap Studio, etc. It is highly recommended to do this!

Microsoft Azure / AzureDevOps

You can signup for Microsoft Azure and start with $100 credit. Note that you can use both your regular Fontys account and your i-account to signup, effectively doubling the amount of free credits! You can use the free tier (first 5 users are free) of Azure DevOps for project management, collaboration and build pipelines.

Amazon Web Services (AWS)

Unfortunately, there is no student access with free credits available for AWS. You could signup for a regular AWS account, and make use of free-tier services, but this requires having a credit card.

Google Cloud

Google Cloud provides a free trial account option, along with $300 credit. Please note that this option is somewhat restricted (e.g. only available to new customers, and only available for 90 days), so make sure that you read the terms of service.

Localstack

Localstack is a fully functional local cloud stack, that can be used for offline, local development and testing of cloud applications. It provides (a subset of) the same services that are available in a real AWS cloud environment. It can be used to speed up your development, and to reduce the cost of cloud resources. It is advised though, to regularly (e.g. and the end of a sprint) to 'flip the switch' and to deploy to the real AWS environment, to see if everything works as planned.

Figure 1: The recommendations by teacher in Canvas

Here are several reasons why I chose working with Azure:

Azure provides a wide range of services and tools that can work together effectively, making it easier to build and deploy the quiz game. Azure's App Service would allow me to easily deploy and manage my web applications, which is beneficial for the React projects and ASP.NET web API projects. Using Azure gives the opportunity to scale the hosted applications, set up continuous integration and deployment, and monitor their performance using App Service.

Azure supports Docker containers, and by utilizing Docker Hub. This makes containerizing applications and deploying them to Azure fairly easy. Docker containers provide a consistent and isolated environment for applications, making it easier to manage and deploy them across different environments.

Azure Functions offer serverless computing capabilities, allowing you to run small pieces of code in response to events or triggers. This can be useful for implementing specific functionality within your quiz game without the need to manage the underlying infrastructure. You can use Azure Functions to handle tasks such as processing messages from RabbitMQ or executing specific logic for certain events.

Azure provides virtual machine (VM) and database server options, which allow to have more control over infrastructure and databases. This would be particularly relevant for my SQL and NoSQL databases, as you can set up and configure VMs and database servers tailored to your specific requirements.

As for the other cloud platforms, here are some potential drawbacks. While the GitHub Student Developer Pack offers access to various tools and resources, it doesn't provide the comprehensive cloud services and infrastructure needed for the project. It mainly focuses on code management and collaboration, which may not fulfill your hosting and deployment requirements.

Although AWS is a popular cloud platform, it doesn't offer specific student access with free credits. Additionally, the requirement of having a credit card may pose a limitation for some students. While one can make use of the AWS free-tier services, it might not cover all the services and scalability options you need for your project.

While Google Cloud provides a free trial account and $300 credit, it comes with certain restrictions, such as limited availability to new customers and a time limit of 90 days. This might not align well with the long-term development and deployment needs of the quiz game.

Localstack is a useful tool for local development and testing, but it is not a full cloud platform. While it can help speed up development and reduce costs, it lacks the benefits of a real cloud environment when it comes to scalability, reliability, and integration with other cloud services.

Considering the context of your project, Azure stands out as a strong choice due to its comprehensive set of cloud services and tools, seamless integration capabilities, and support for hosting the projects specific technology stack. Azure's App Services, Docker Hub, Azure Functions, and VM/database server options provide the necessary infrastructure and services for the quiz game’s React projects, ASP.NET web APIs, FastAPI, SQL and NoSQL databases, and RabbitMQ messaging service. By utilizing Azure, you can leverage its robust ecosystem and maximize the potential of your quiz game project.

Question & Answer service

In my “How to build a Question & Answer service?” research document I had researched and described several way of building the Question & Answer Service. I had come to the conclusion that for local development the best case for this service would be choosing Python as the programming language and FastAPI as the building framework. In this document I have also explored the possibilities of integrating the Question & Answer service into Azure Functions.

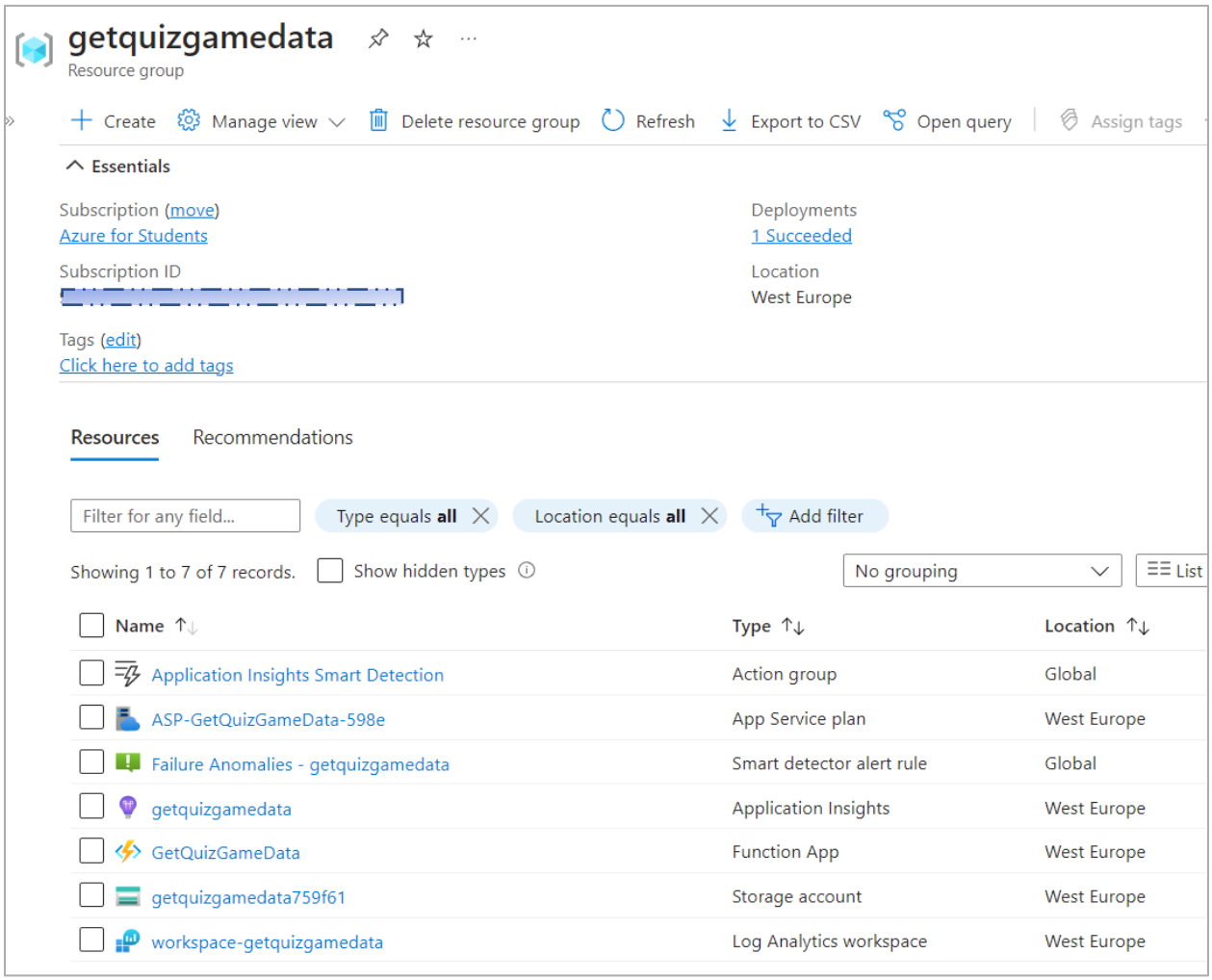
Azure Functions offer serverless computing, eliminating the need to manage infrastructure. With Azure Functions, only the execution time of the code is billed, making it cost-effective for sporadic API usage. Azure Functions allow for seamless scalability and remove the burden of maintaining dedicated infrastructure for less frequently accessed components.

Azure Functions automatically scale based on demand. If there is a sudden surge in traffic to the infrequently used API, Azure Functions can accommodate the load by scaling out. This ensures reliable and consistent processing of API calls, even during peak periods. The high availability of Azure Functions guarantees continuous accessibility and responsiveness.

Azure Functions streamline the development and deployment process. API logic can be written as individual functions, simplifying management and testing. Azure Functions support multiple programming languages, including Python, making them suitable for FastAPI built on the FastAPI framework. Updates or modifications to specific functions can be easily deployed without affecting the rest of the application.

Azure Functions provide built-in monitoring and logging capabilities. They enable tracking of API requests, response times, and any errors or exceptions that occur. This visibility facilitates quick identification and resolution of issues related to the integration of FastAPI

By leveraging Azure Functions to integrate the infrequently used FastAPI, cost efficiency, scalability, and reliability can be ensured for specific API calls in the quiz game project. The serverless nature of Azure Functions, coupled with auto-scaling, event-driven triggers, simplified development, and deployment, make them an ideal choice for seamless integration of the FastAPI into the overall application architecture.



The resource group "getquizgamedata." is a logical container that holds related Azure resources for easier management and organization. Application Insights is a service that enables monitoring and diagnostics for applications.

Smart Detection is a feature within Application Insights that automatically detects anomalies or performance issues in your application and provides alerts. An action group is a collection of notification preferences and actions that define how alerts are handled.

ASP-GetQuizGameData-598e refers to an App Service plan, which defines the compute resources and settings for hosting web applications, including your quiz game application.

Failure Anomalies represents a smart detector alert rule within Application Insights. It is configured to detect anomalies related to failures in your quiz game application and generate alerts when such anomalies occur.

“getquizgamedata” is the name of an Application Insights instance, which is a monitoring and diagnostics service that helps you understand the performance and usage of the application. It collects telemetry data from the application for analysis and provides insights.

GetQuizGameData refers to a Function App, which is a serverless compute resource in Azure for hosting and running your serverless functions.

“getquizgamedata759f61” represents a storage account, which provides a place to store and access your data, such as files, blobs, tables, and queues.

“workspace-getquizgamedata” is the name of a Log Analytics workspace.

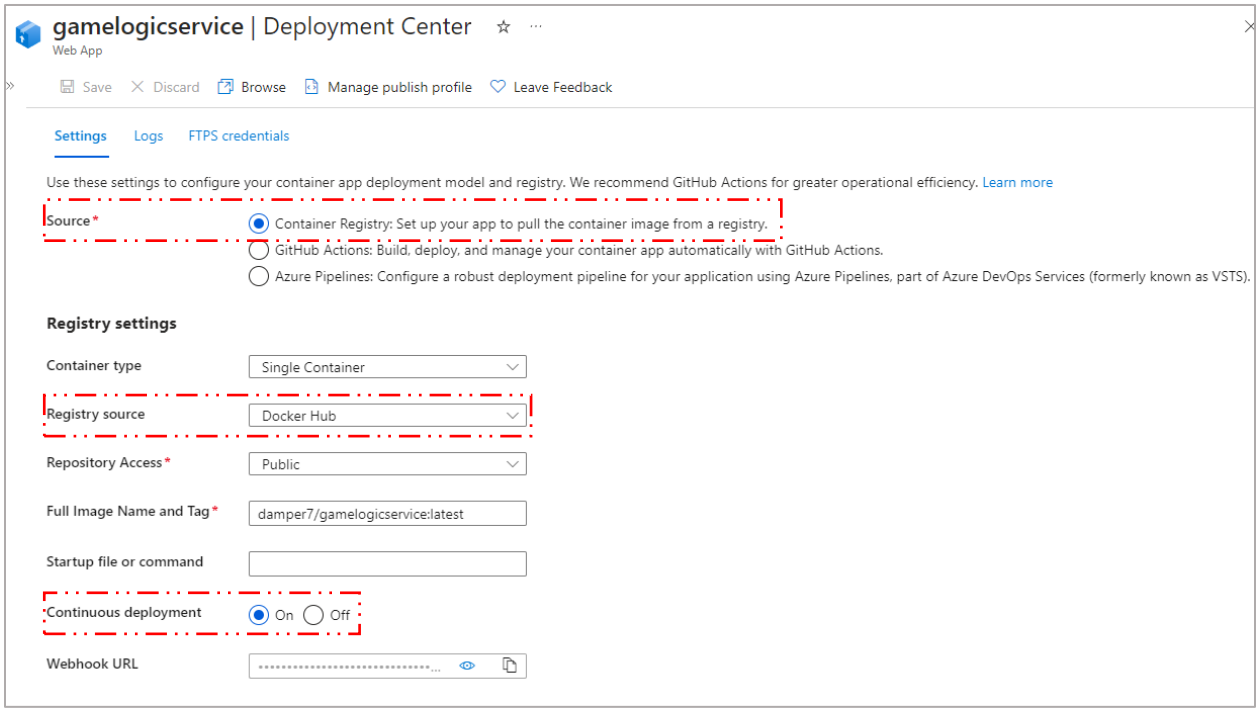
Game Logic Service/ Game Scoreboard Service

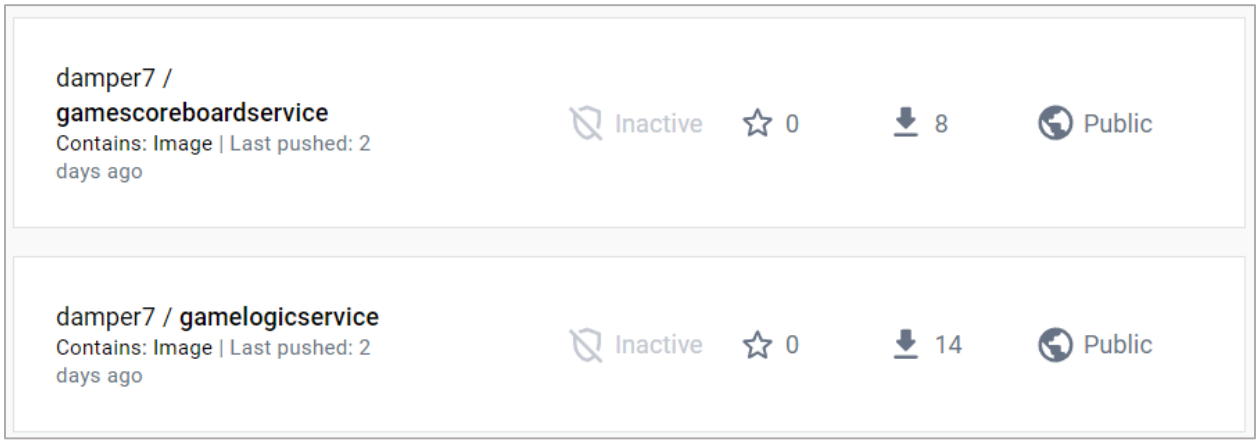
I have decided that it is best to deploy both the Game Logic Service and the Scoreboard Service, which are ASP.NET Web APIs to a Docker Hub images and then to Azure App Services because it offers several benefits, including portability, consistency, scalability, and ease of deployment.

Deploying the ASP.NET Web API to a Docker Hub image encapsulates the application and its dependencies into a container. Containers provide a consistent and isolated environment for the application, ensuring reliable execution across different environments. Docker Hub serves as a central repository for storing and sharing container images, facilitating distribution and deployment.

Azure App Service provides flexible scaling options, allowing horizontal or vertical scaling based on demand. Deploying the Docker image to Azure App Service leverages the scalability and elasticity offered by the platform. Azure App Service handles the underlying infrastructure management, enabling efficient scaling to handle increased traffic.

Docker Hub integrates well with CI/CD pipelines, enabling streamlined build, test, and deployment processes. By utilizing Docker Hub as an intermediate step, the CI/CD pipeline automatically builds the Docker image, runs tests within the container, and pushes the image to Docker Hub. Subsequently, the deployment to Azure App Service is triggered, enabling continuous integration and deployment of the ASP.NET Web API. The images below showcase the continues deployment of the public gamelogicservice (same for the gamescorebaordservice) docker hub image. And the second picture shows how many times has the docker image been pulled by App Service since it has been first deployed. A CI/CD pipeline could be built to only build the image in Docker Hub and the deployment would continue on Azure’s side. However, in the upcoming sprint I am planning on making my own CI/CD pipeline script. To build and deploy right to Azure App Service.





Workflow of the deployment process:

- Develop and test the ASP.NET Web API locally, ensuring proper functionality.
- Create a Dockerfile defining the dependencies, configurations, and build instructions for the ASP.NET Web API Docker image. The Dockerfile specifies the base image, adds the application code, sets environment variables, and exposes the required ports.
- Configure a CI/CD pipeline to initiate the build process whenever changes are committed to the source code repository. The pipeline builds the Docker image using the Dockerfile and pushes it to Docker Hub, serving as a centralized image repository.
- Configure Azure App Service to retrieve the Docker image from Docker Hub. Specify the necessary environment variables, networking configurations, and additional settings required by the application. Azure App Service deploys the image to the designated instance(s), making the ASP.NET Web API accessible through the assigned URL.

- Enable continuous integration and deployment by integrating the source code repository, Docker Hub, and Azure App Service. With each commit to the repository, the CI/CD pipeline triggers automatically. It builds the Docker image, runs tests (if applicable), pushes the image to Docker Hub, and deploys it to Azure App Service. This ensures a seamless and automated deployment process for the ASP.NET Web API.

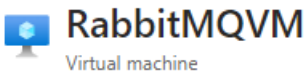
By following this workflow and deploying the ASP.NET Web API to Docker Hub and subsequently to Azure App Service, containerization, portability, streamlined CI/CD, and scalability can be achieved for the application. This approach enables efficient development, testing, and deployment in a consistent and scalable manner.

RabbitMQ Messaging Service

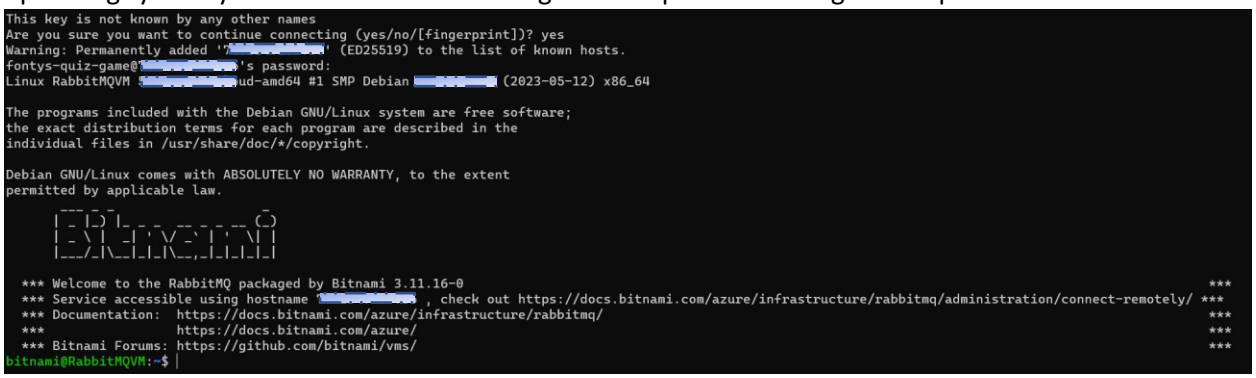
The Quiz Game locally makes use of a RabbitMQ Messaging Server. So, I order to make the whole Quiz Game available on a cloud – I had to figure out a way through researching the possibilities, to deploy the messaging server. This was done using RabbitMQ packaged by Bitnami

Workflow of the deployment process:

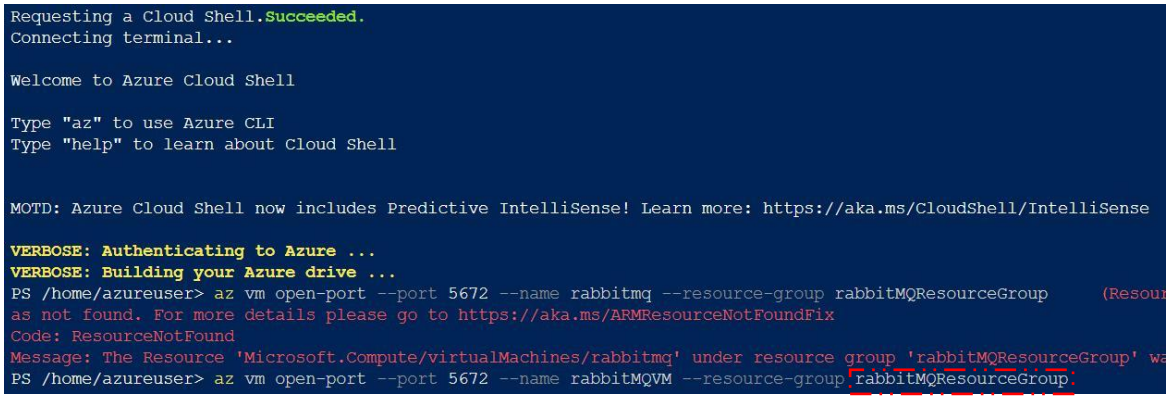
- Create an Azure VM within a resource group, specifying the appropriate VM size, operating system, and network configurations. This VM will serve as the host for the RabbitMQ messaging server.



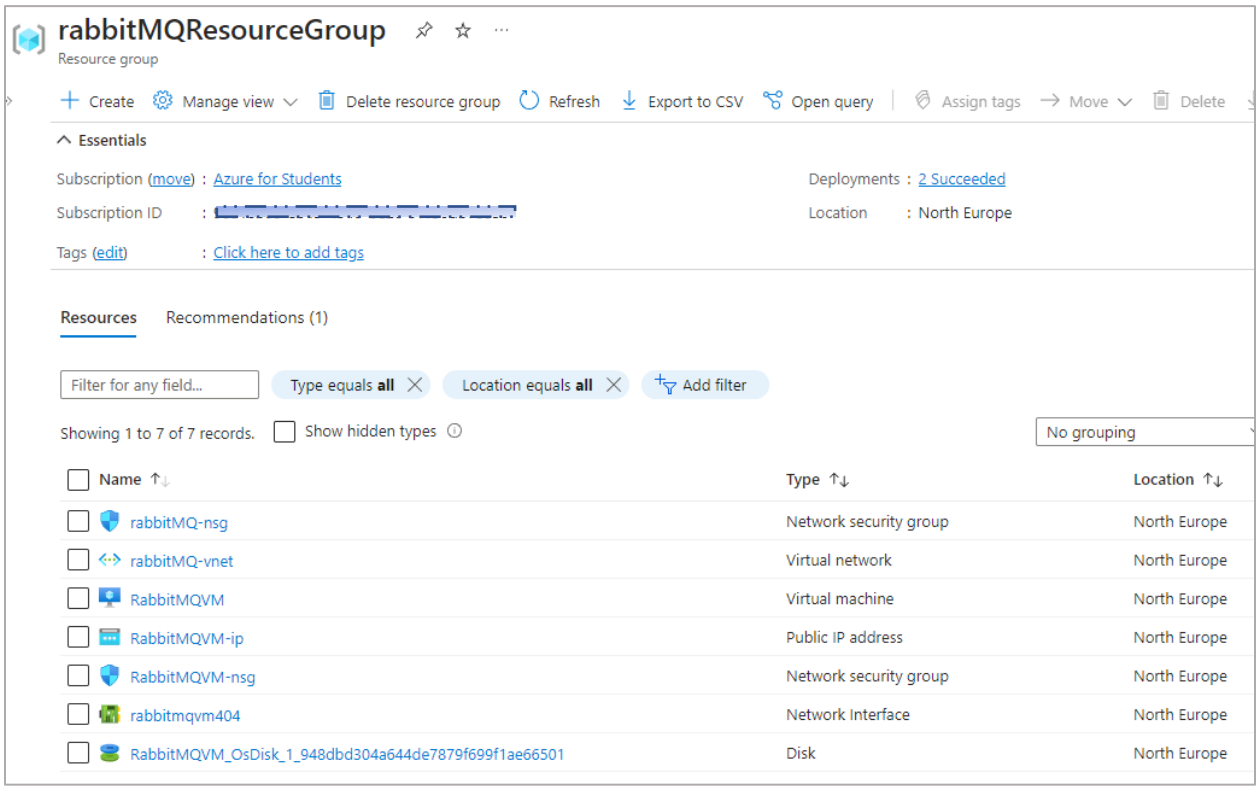
- Install RabbitMQ on the Azure VM by following the appropriate installation instructions for the operating system you selected. Connect trough an SSH portal to configure the ports.



- Configure the Azure VM's network security group to allow inbound and outbound connections on the necessary RabbitMQ ports (e.g., 5672 for AMQP, 15672 for management interface). This was made using Azure CLI.



- Connecting App Services to the Messaging Server: Update your App Services' configurations to include the connection details (such as hostname, port, credentials) for the RabbitMQ messaging server. This allows your App Services to establish connections and exchange messages with the RabbitMQ broker.
- Testing and Validation: Perform thorough testing to ensure that the App Services can successfully communicate with the RabbitMQ messaging server. Verify message exchanges, queues, and any other necessary configurations to ensure proper functioning of the messaging system.



By setting up a dedicated RabbitMQ messaging server, I establish a centralized infrastructure to handle messaging and communication between the App Services. RabbitMQ is a robust and widely-used message broker that provides reliable message queuing and delivery mechanisms.

Azure VMs offer scalable computing resources, allowing to adjust the VM size and capacity based on the application's requirements. This ensures optimal performance for the RabbitMQ messaging server, enabling efficient handling of messages and accommodating potential increases in traffic or demand.

By connecting the App Services to the RabbitMQ messaging server, a decoupled communication mechanism between different components of the quiz game is established. This enables efficient asynchronous communication, allowing services to exchange messages without direct dependencies, resulting in improved scalability, reliability, and flexibility.

VMs provide robust security features, including network security groups and virtual network configurations, to protect your RabbitMQ messaging server from unauthorized access. One can define access controls, restrict inbound and outbound traffic, and implement encryption measures to ensure the integrity and confidentiality of the messages being exchanged.