



Fakultät Wirtschaft  
Studiengang Wirtschaftsinformatik (B.Sc.)  
IT-Sicherheit - Studiumsprojekt

# Aufbau und Bereitstellung einer Website mit dem Azure Kubernetes Service

Verfasser: Daniel Landau  
Mat. Nr: 150523  
Lehrbeauftragter: Dr. Guido Söldner

Hochschule für angewandte Wissenschaften Ansbach  
Residenzstraße 8, 91522 Ansbach

# Inhalt

Inhalt.....	I
Abbildungsverzeichnis.....	III
Abkürzungsverzeichnis.....	IV
1 Webserverbereitstellung in Kubernetes.....	- 1 -
1.1 Ziel der Arbeit .....	- 1 -
2 Überblick über die Anwendung .....	- 2 -
2.1 Beschreibung der Anwendung und Grundfunktionen .....	- 2 -
2.1.1 Die Hauptfunktionen .....	- 2 -
2.1.2 Die Nebenfunktion .....	- 3 -
2.1.3 Verwendete Javascript Bibliotheken.....	- 3 -
2.2 Beschreibung der Systeme .....	- 4 -
2.2.1 Der Node.js-Server .....	- 4 -
2.2.2 Der MySQL-Server .....	- 5 -
2.2.3 Skalierbarkeit.....	- 6 -
3 Bereitstellung in der Azure Cloud .....	- 7 -
3.1 AKS - Azure Kubernetes Services .....	- 7 -
3.1.1 Das Webserver Manifest .....	- 7 -
3.1.2 Das Datenbank Manifest .....	- 8 -
3.1.3 Der persistente Datenbankspeicher .....	- 9 -
3.1.4 Der persistente Webserverpeicher .....	- 9 -
3.1.5 Die ConfigMap und Secrets .....	- 10 -
3.2 ARM-Templates .....	- 11 -
3.2.1 Der Dienstprinzipal.....	- 12 -

3.3	Verbinden mit dem bereitgestellten Cluster .....	- 13 -
3.4	Azure DNS-Zonen, Namensserver und Domänen .....	- 14 -
3.5	Überblick der Infrastruktur .....	- 15 -
3.6	Sicherheit .....	- 16 -
3.6.1	Verschlüsselter HTTPS-Verkehr .....	- 16 -
3.6.2	Authentifizierung und Bcrypt Passwort-Hashes .....	- 16 -
3.6.3	JWT und HTTP-Only Cookies .....	- 17 -
3.6.4	Verschlüsselung mit Crypto .....	- 19 -
4	Entwicklungsumgebung und Tool .....	- 20 -
4.1	Entwicklungsumgebung .....	- 20 -
4.2	Postman .....	- 20 -
4.2.1	Senden einer Suchanfrage .....	- 20 -
4.3	Fiddler .....	- 21 -
	Eidesstattliche Erklärung .....	- 1 -

## Abbildungsverzeichnis

Abbildung 1	Schematische Darstellung der Datenbank .....	- 5 -
Abbildung 2	Screenshot der Dateifreigabe in Azure.....	- 10 -
Abbildung 3	Screenshot des Access Control Panels in Azure .....	- 12 -
Abbildung 4	Screenshot des Kubectl-Clients .....	- 13 -
Abbildung 5	Screenshot der DNS-Einträge in Azure .....	- 14 -
Abbildung 6	Screenshot der Eingabe der URL .....	- 14 -
Abbildung 7	Schema der Kubernetes / Azure Infrastruktur .....	- 15 -
Abbildung 8	Screenshot der Decodierung eines JWT .....	- 17 -
Abbildung 9	Screenshot der Verifizierung eines JWT .....	- 18 -
Abbildung 10	Screenshot eines verschlüsselten JWT .....	- 19 -
Abbildung 11	Screenshot einer Anfrage in Postman .....	- 20 -
Abbildung 12	Screenshot der HTTPS-Anfrage in Fiddler .....	- 21 -

## Abkürzungsverzeichnis

- AKS** Der **A**zure **K**ubernetes **S**ervice ist die in Azure integrierte K8S Lösung.
- ARM** Der **A**zure **R**esource **M**anager ist eine REST-API Schnittstelle von Azure über die mittels Templates Ressourcen automatisch angelegt werden können.
- JWT** Das **J**son**W**eb**T**oken beschreibt ein validierbares Token, das bei jeder Anfrage an den Server gesendet wird.
- JSON** Die **J**ava **S**cript **O**bject **N**otation ist ein gängiges Format zur Datenserialisierung.
- K8S** **K**ubernetes ist eine gängige Kurzschreibweise, wobei die 8 als Platzhalter für Acht ausgelassene Zeichen dient.
- YAML** **Y**AML **A**in't **M**arkup **L**anguage. Rekursives Akronym für ein menschlich angenehm lesbares Format zur Datenserialisierung.

# 1 Webserverbereitstellung in Kubernetes

## 1.1 Ziel der Arbeit

Das Projekt befasst sich mit Kubernetes und der Bereitstellung von Softwarelösungen in Kubernetes. Als Grundlage dient ein Webserver, der zusammen mit einer Datenbank eine Website hostet. Die beiden Systeme Webserver und Datenbankserver werden dabei in Kubernetes bereitgestellt. Dabei wird auf den in Azure verfügbaren Kubernetes Dienst AKS zurückgegriffen und für alle anderen nötigen Funktionen, wie zum Beispiel das Persistieren verschiedener Daten, ebenfalls Azure Dienste verwendet. Weitere Anforderungen an die Lösung sind der Öffentlicher Zugang über das Internet mittels einer auflösbaren URL und verschlüsselter HTTPS-Datenverkehr mit einem gültigen Zertifikat, sowie einem Login-System und Authentifizierung aller Anfragen durch Anmeldung als Nutzer.

## 2 Überblick über die Anwendung

### 2.1 Beschreibung der Anwendung und Grundfunktionen

Die angestrebte Anwendung stellt eine Website mit verschiedenen Funktionen bereit. Zum einen kann der Nutzer der Seite ein neues Konto anlegen bzw. sich mit seinem bestehenden Konto anmelden oder auch nach der Anmeldung ausloggen kann. Bei der Erstellung eines neuen Kontos wird nach dem Displaynamen, dem Nutzernamen und einem Passwort gefragt. Der Nutzernamen, ist die eindeutige ID, mit der sich der Nutzer anmelden kann. Der Displayname hingegen ist der für andere Nutzer sichtbare Name. Nach der erfolgreichen Anmeldung landet der Nutzer auf der Hauptseite, auf welcher er mittels vier Links auf die zwei Hauptfunktionen oder auch auf die zwei Nebenfunktionen der Seite zugreifen kann.

#### 2.1.1 Die Hauptfunktionen

Die Hauptfunktionen der Seite umfassen zum einen das Zeichnen von Doodles und das automatische kategorisieren dieser in 345 verschiedene Klassen, wie beispielsweise Hund, Katze, Blitz, etc. Zum Zeichnen stehen dem Nutzer ein einfacher Pinsel zur Verfügung, dessen Größe und Farbe jeweils mittels eines Sliders und einer Farbauswahl verändert werden kann. Der Radiergummi wird entweder durch das Betätigen der rechten Maustaste oder das Klicken des Knopfes mit entsprechendem Icon aktiviert. Während dem Zeichnen erscheint rechts neben der Bildfläche zehn verschiedene Kategorien, nach Prozenanteilen sortiert, wie sehr das neuronale Netz sicher ist, das es sich bei diesem Bild um die Kategorie handelt. Der Nutzer kann der Zeichnung anschließend einen Namen geben, bevor er sie mit dem Knopf „Save Image to Server“ an den Server schickt und somit für andere Nutzer zur Verfügung stellt. Nach dem Senden an den Server kann der Nutzer nochmals nachträgliche Änderungen am Bild machen und durch erneutes Betätigen des Knopfes das Bild updaten. Erst wenn der Browser-Tab geschlossen, neugeladen oder der Knopf zum Erstellen eines neuen Doodles betätigt wird, ist die Zeichnung final und für den Nutzer unveränderbar auf dem Server gespeichert. Die zweite Hauptfunktion ist die Bildergalerie, in der der gesamte Datenbestand des Servers durchsucht werden kann. Die Suchparameter umfassen die Bilderkategorien, die von Nutzern vergebenen Bildernamen und die Displaynamen der Nutzer, die das entsprechende Bild erstellt haben. Werden Bilder gemäß den

gesetzten Suchparametern gefunden, werden sie in kleinen Zellen dargestellt. Dabei wird zuerst der Name des Bildes, der Displayname des Nutzers angezeigt, gefolgt von dem eigentlichen Bild, sowohl auch dessen Bildklassifizierung.

### 2.1.2 Die Nebenfunktion

Die Nebenfunktionen umfassen einmal das Fliegen einer Rakete in einem Mini-Browsergame. Zum jetzigen Standpunkt ist jedoch weder Highscore noch ein Ende des Spiels implementiert. Die andere Nebenfunktion zeigt eine dynamisch generierte Seite an, mit dem Displaynamen des angemeldeten Nutzers und dem Namen des Servers, von dem der Nutzer bedient wird.

### 2.1.3 Verwendete Javascript Bibliotheken

Die Website verwendet verschiedene Javascript Bibliotheken für verschiedene Zwecken:

- **JQuery** ist eine Bibliothek, die verschiedene Methoden zur Selektion und Manipulation von Elementen im DOM bereitstellt.
- **P5.js** ist eine Bibliothek, zum Erstellen von Grafiken und Interaktiven Elementen. Es findet Hauptanwendung für das Malen der Zeichnungen und das Ansprechen der Endpunkte via HTTP-Anfragen.
- **ML5.js** ist eine Bibliothek, die auf **Tensorflow.js** aufbaut mit dem Ziel Maschine Learning einfacher und für ein breiteres Publikum zugänglich zu machen. Es findet Verwendung zur Klassifizierung der Zeichnung in 345 verschiedene Begriffe mithilfe des DoodleNet Models, welches mit einem Bruchteil des 50 Millionen Bilder großen Google Quick, Draw! Datensets trainiert wurde.
- **Phaser.js** ist eine Bibliothek, die verschiedene Funktionen für eine vollständige Game Engine im Browser bereitstellt, wie eine Physiksimulation, Rendering mit Canvas oder WebGL, Animationen, Partikelsysteme, Sounds, etc. Sie findet Einsatz für das noch unfertige Spiel, das Anfangs geplant war, aufgrund von Zeitgründen jedoch nicht vollendet werden konnte, aber noch als Gimmick auf der Website verbleibt.



## 2.2 Beschreibung der Systeme

Das Gesamtsystem besteht aus einem Node.js-Server, der vorwiegend statische HTML-Seiten an den Nutzer liefert. Clientseitiges Javascript greift auf die bereitgestellten Endpunkte, wie beispielsweise zur Nutzeranmeldung oder Bildersuche zu, um diverse Daten auszutauschen und die Grundfunktionalitäten bereitzustellen. Der andere Bestandteil des Systems umfasst eine MYSQL-Datenbank, die wiederum alle Nutzerdaten für die Anmeldung speichert und entsprechende Metadaten für deren gezeichneten Doodles.

### 2.2.1 Der Node.js-Server

Node.js ist eine Javascript Laufzeitumgebung mit der Javascript serverseitig betrieben und damit ein Webserver bereitgestellt werden kann. Der Server selbst wird in einem Docker-Container gehostet, der sich wiederum innerhalb eines Pods befindet. Grundlage dafür ist ein selbsterstelltes Abbild, dem das Abbild *Node:alpine-3.12* zu Grunde liegt und welches durch Kopieren des *Package.json* und des Source-Codes erweitert wird. Das *Package.json* umfasst alle externen Module, von dem der Webserver abhängig ist. Das Containerabbild wird anschließend in einem öffentlichen Repository unter *daniellandau1998/node-webserver* gespeichert. Die externen Abhängigkeiten umfassen:

- **Express** ist ein Framework für Node.js zum Erstellen von Webservern.
- **MySQL** ist das Modul, welches für die Kommunikation mit der MySQL-Datenbank Verwendung findet.
- **Body-Parser** ist ein Modul, dass den Body eingehender HTTP-Anfragen automatisch auf das JSON-Format durchsucht und in ein Javascript-Objekt umwandelt. Dieser ist im Code dann als *request.body* verwendbar.
- **Bcrypt** ist ein Modul, das Methoden zur Umwandlung von Nutzerpasswörtern in sichere Hashes und überprüfen von Passwörtern mit Hashes bereitstellt. Es findet bei der Kontoerstellung und dem Login Einsatz.

- **Joi** ist ein Modul, welches Methoden zur Überprüfung von Objekten, anhand verschiedenen Schemas bereitstellt. Es überprüft die Eingaben der Daten, die an den Server gesendet werden.
- **JsonWebToken** ist ein Modul, das Methoden zur Erstellung und Validierung von JWTs bereitstellt. Es findet beim Login und der Authentifizierung an den Server Einsatz.

Der eigentliche Webserver befindet sich in der Datei Webserver.js und ist wiederum in eigens erstellte Module untergliedert, die sich im Ordner *modules\_private* befinden:

- **Joi Models** ist ein Modul, das alle Schemas zur Validierung der HTTP-Anfragen bereitstellt.
- **Sql Calls** ist ein Modul, das alle notwendigen Statements und Funktionen zur Erstellung der Tabellen und Abfragen dieser bereitstellt.
- **Image Data** ist ein Modul, das alle Serverendpunkte und Funktionen zur Verarbeitung, Speicherung und Suchen der Zeichnungen bereitstellt.
- **User Auth** ist ein Modul, das alle Serverendpunkte und Funktionen zur Registrierung, Anmeldung und Nutzerauthentifizierung bereitstellt. Das umfasst das Erstellen, verschlüsseln und Entschlüsseln der JWTs.

### 2.2.2 Der MySQL-Server

Bei dem MySQL-Server handelt es sich um ein unverändertes Abbild. Das verwendete Abbild ist „mysql:5.7“. Die Erstellung der Tabellen wird automatisch von dem Webserver übernommen.

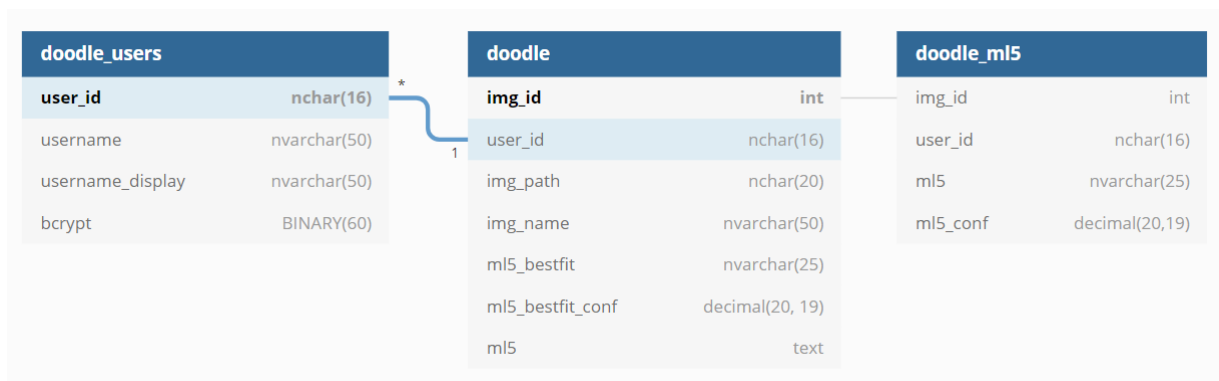


Abbildung 1 Schematische Darstellung der Datenbank

Quelle: Eigenentwurf mit <https://dbdiagram.io/home>

Abbildung 1 stellt ein Schema der Datenbank dar. Die Tabelle *doodle\_user* speichert alle Daten bezüglich des Nutzers ab. Hierzu gehören die *user\_id*, eine eindeutige 64-bit Hexadezimalzahl, der *username* zur Identifizierung beim Login, der *username\_display* als schöner Anzeigename und das mittels Bcrypt gehashte Passwort. Die Tabelle *doodle* speichert alle Metadaten zu den Zeichnungen ab. Angefangen mit einer eindeutige *img\_id* als Ganzzahlige Zahl und der Fremdschlüssel *user\_id*, der auf den Nutzer verweist in der Tabelle *doodle\_user* verweist, der die Zeichnung erstellt hat. Die Spalte *img\_path* speichert den eindeutigen Namen des Bildes in der Form einer 64-bit Hexadezimalzahl gefolgt von der Zeichenkette „.png“ ab. Die folgenden drei Spalten speichern jeweils den Namen des Bildes, die Klassifizierung mit der höchsten Prozentzahl, sowie dessen Prozentsatz als *ml5\_bestfit\_conf*. Die letzte Spalte speichert die restlichen Klassifizierungen als JSON-Zeichenkette ab. Die dritte Tabelle ist für erweiterte Suchfunktionen vorgesehen und soll die anderen Klassifizierungen zu einem Bild speichern, besitzt jedoch momentan keine Funktion.

### 2.2.3 Skalierbarkeit

Da die Webserver Instanzen zustandslos sind, können diese beliebig hoch und herunterskaliert werden, während die Datenbank nur mittels eines einzigem Pod bereitgestellt wird.

## 3 Bereitstellung in der Azure Cloud

### 3.1 AKS - Azure Kubernetes Services

Das System aus Webserver und MYSQL-Datenbank soll letztendlich unter einer öffentlich erreichbaren URL im Internet bereitgestellt werden. Der Webserver selbst und der Datenbankserver werden dabei mittels Kubernetes in Azure gehostet. Der Azure Kubernetes Service ist der in Azure integrierte Kubernetes Dienst. Zur Bereitstellung werden die deklarativen Kubernetes Manifest im YAML-Format verwendet.

#### 3.1.1 Das Webserver Manifest

Das Manifest für den Webserver befindet sich im Sourcecode im Verzeichnis `./k8s-azure-aks/k8s-deploy-web.yaml` und beschreibt ein Kubernetes Deployment. Das Deployment selbst beinhaltet den Schlüssel *Replicas*, welcher auf einen beliebigen Wert gesetzt werden kann und die Anzahl aller Webserver-Instanzen beschreibt. Der Webserver wird als Pod unter dem Schlüssel *Template* definiert. Die Pods werden vom Deployment über Selektoren mit den zwei Labels *app: doodles-webservice* und *tier: frontend-webserver* miteinander implizit verlinkt. Der Pod beinhaltet einen Container, der das eigens erstellte Abbild namens *daniellandau1998/node-webserver:k8s* mit dem Node.js Webserver verwendet. Zusätzlich wird einmal der HTTP-Port 80 und der HTTPS-Port 443 geöffnet, auf welchen das nachfolgende Kubernetes Serviceobjekt zugreift. Die standardmäßige Kommunikation verläuft über den verschlüsselten HTTPS-Verkehr. Der HTTP-Port ist lediglich offen, um den Browser des Endnutzers automatisch auf die HTTPS-Seite umzuleiten, sofern dieser eine normale HTTP-Anfrage tätigt. Die Umgebungsvariablen werden durch die ConfigMap *node.webserver.config* und den Secrets *sql.pass.data*, *ssl.cert.data*, *jwt.rsa.data* und *jwt.enc.data* in den Pod eingespeist. Die einzelnen Secrets und ConfigMap werden in einem späteren Abschnitt weiter beschrieben. Da der Pod die einzelnen Zeichnungen speichern soll, aber dessen Dateninhalt an den Lebenszyklus des Pods gebunden ist, wird ein Datenvolumen eingebunden, das den Pod überdauert. Zudem wird dieses Datenvolumen von allen Pods eingebunden und geteilt, wodurch jede Instanz des Webserver Zugriff auf die Daten hat. Das Volumen wird dabei nicht direkt eingebunden, sondern stattdessen ein PersistentVolumeClaim, dessen Umsetzung in einem folgenden Abschnitt erläutert wird.

Nach dem Manifest des Deployment folgt in derselben Datei ein zweites Manifest für einen Kubernetes Server. Dieser ist vom Typ Loadbalancer und stellt die Pods über eine öffentliche IP im Internet zur Verfügung. Dieser arbeitet nach demselben Selektorprinzip, wie das vorangegangene Deployment. Der Loadbalancer wird unter dem Schlüssel *Ports* angewiesen einmal alle Anfragen über den Port 443 auf den entsprechend gleichen Port im Container weiterzuleiten. Analoges gilt mit dem HTTP-Port 80.

### 3.1.2 Das Datenbank Manifest

Das Manifest für die Datenbank befindet sich im Sourcecode im Verzeichnis *./k8s-azure-aks/k8s-deploy-db.yaml* und beschreibt ein Kubernetes Deployment. Es ist ähnlich aufgebaut wie das Deployment für den Webserver mit dem Unterschied, dass der Selektor nach Pods mit dem Label *tier: backend-database* sucht. Das Label *app: doodles-webservice* ist bei beiden Selektoren gleich. Da die Datenbank nicht zustandslos ist und konsistent gehalten werden muss, darf die Zahl unter dem Schlüssel *Replicas* nur einen einzigen Pod umfassen. Aufgrund der gewählten persistenten Speicherlösung wird eine zweite Instanz jedoch grundlegend fehlschlagen, da das Datenvolumen kein zweites Mal eingebunden werden kann. Der Grund warum trotzdem ein Deployment der Definition eines einzigen Pods bevorzugt wird ist die Tatsache, dass Kubernetes versucht die angegeben Replicazahl einzuhalten. Das heißt, wenn der Datenbankserver fehlschlägt, wird ein neuer Pod gestartet und das Datenvolumen mit der bestehenden Datenbank eingebunden. Bei dem Bereitstellen eines einzigen Pods besteht diese Funktionalität nicht, heißt man müsste den neuen Ersatzpod manuell erstellen. Der definierte Pod beinhaltet wiederum nur einen Container, der das unveränderte Abbild *mysql:5.7* verwendet. Der einzige geöffnete Port ist der Port 3306, da die Kommunikation der MySQL-Servers über diesen abläuft. Ebenso wie der Webserver, müssen die Daten der Datenbank über die Lebensdauer des Pods persistiert werden. Dafür wird auch wieder ein zweites PersistentVolumeClaim eingebunden. Ebenso wie in dem vorangegangenen Webserver Manifest wird auch gleich das Manifest für das Serviceobjekt definiert. Dieses arbeitet wieder nach demselben Selektorprinzip, wie das gerade beschriebene Deployment. Der Typ ist eine ClusterIP, da dieser nur innerhalb des Clusters zugänglich sein muss. Die Portkonfiguration bildet den Serviceport 3306 auf denselben Port des Pod ab.

### 3.1.3 Der persistente Datenbankspeicher

Unter dem Pfad `./k8s-azure/k8s-pv-db.yaml` befindet sich das Manifest für den PersistentVolumeClaim, der vom Datenbankpod eingebunden wird, und das dahinterstehende PersistentVolume. Das PersistentVolume ist eine Kubernetes Ressource, die allgemein ein Datenvolumen beschreibt, welches verschiedenen Ausprägungen bzw. Umsetzungen umfassen kann. Das definierte Volumen, erhält die Labels `app: doodles-webservice`, `tier: backend-database`, `typ: azure`, die vom Claim verwendet werden, um ein entsprechend passendes Volumen zu finden. Die Definition des Volumens unter dem Schlüssel `spec` ist kurz. Es umfasst die Speichergröße von einem Gigabyte initialisiert mit einem Dateisystem und dem `AccessMode: ReadWriteOnce`. Der Zugriffsmodus sagt aus, dass dieses Volumen nur einmal eingebunden werden kann. Der letzte Schlüssel mit den drei Unterschlüsseln beschreibt die tatsächliche Ressource, die als Speichermedium dient. In diesem Falle ist es eine Azure Disk, welche eine virtuelle Ressource in der Azure Cloud ist, die im Grunde eine Blockspeichermedium darstellt, also eine Festplatte oder SSD. Die drei Angaben, die noch getätigt werden müssen, sind `kind`, `diskURI` und `diskName`. Die Art besetzt mit `Managed` beschreibt eine gemangte Disk von Azure. Der Name definiert den Namen der Azure Ressource und die URI den eindeutigen *Resourceidentifier* der Azure Ressource.

Der Claim befindet sich in derselben Datei und selektiert nach den definierten Labels das passende Datenvolumen. Der Claim und das dahinterstehende Volume sind grundsätzlich voneinander unabhängig. Der Vorteil davon ist, dass die Implementation des eigentlichen Volumens vom eingebundenen Claim des Pods entkoppelt ist. Das heißt der Claim sucht nach einem Volumen das dessen Anforderung erfüllt. Wie das Datenvolumen tatsächlich umgesetzt ist, ist für den PersistentVolumeClaim aber nicht relevant.

### 3.1.4 Der persistente Webserverspeicher

Der Persistente Speicher für den Webserver unter dem Pfad `./k8s-azure/k8s-pv-web.yaml` ist ähnlich aufgebaut, wie der für die Datenbank. Die Unterschiede liegen zum einen im Label `tier`, das hier den Wert `frontend-webserver` trägt. Entsprechend sucht der Claim nach dieser Wertausprägung. Der Zugriffsmodus ist `ReadWriteMany`, was bedeutet, dass das Datenvolumen mehrmals eingebunden werden kann. Passend für den Webserver, da dieser, aufgrund seiner

Zustandslosigkeit, beliebig viele Instanzen bereitstellen kann, muss das Volumen auch beliebig oft eingebunden werden können. Der Schlüssel *azureFile* mit seinen drei Unterschlüsseln beschreibt eine in Azure bereitgestellte Dateifreigabe. Eine Azure Disk, wie bei der Datenbank, ist in diesem Fall nicht möglich, da diese nur einmal eingebunden werden kann. Die Dateifreigabe hingegen kann von allen Webserverinstanzen gleichzeitig eingebunden und beschrieben werden. Als Unterschlüssel werden der Name angegeben, eine Referenz auf ein Secret und der Namensraum, in dem sich das Secret befindet. Die Dateifreigabe wird innerhalb von Azure in einem Speicherkonto umgesetzt. Das Secret enthält sowohl den Namen als auch den Zugangsschlüssel für das Speicherkonto.

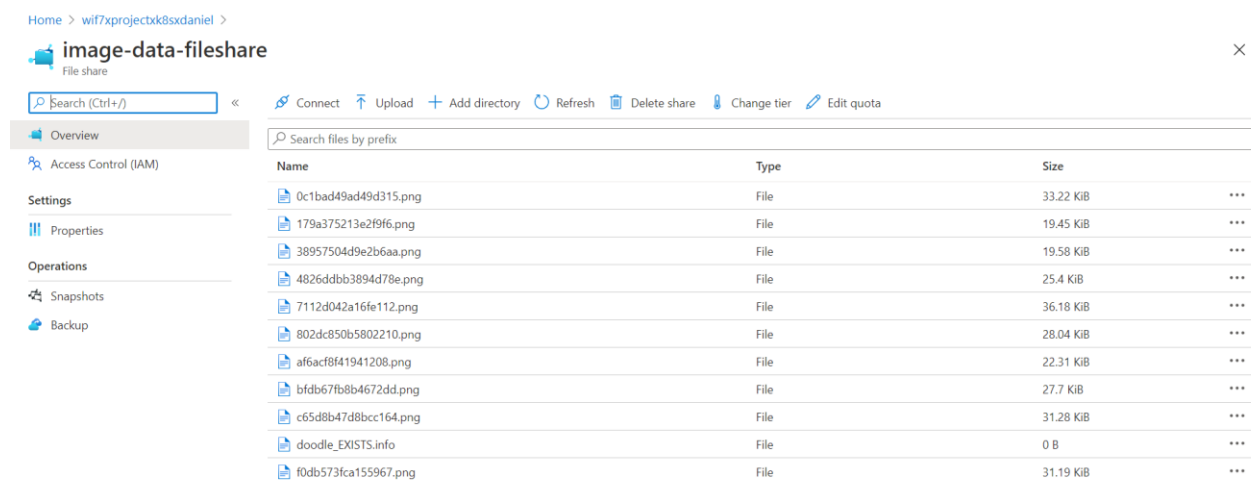


Abbildung 2 Screenshot der Dateifreigabe in Azure  
Quelle: Eigens erstellter Screenshot

Ist die Dateifreigabe eingebunden, werden darauf, wie in Abbildung 2 dargestellt, die Bilder abgelegt. Die Datei *doodle\_EXISTS.info* wird bei der Initialisierung der Datenbank angelegt. Sobald ein Webserver die nötigen Tabellen erstellt hat, legt dieser eine solche Datei ab, um nachfolgenden Instanzen mitzuteilen, dass die Tabellen bereits angelegt wurden.

### 3.1.5 Die ConfigMap und Secrets

Die ConfigMap ist eine Möglichkeit, um in Kubernetes eigene spezifische Konfigurationsinformationen in Pods einzuspeisen. Es gibt zwei Typen. Einmal eine normale ConfigMap für normal Konfigurationsdaten, wie zum Beispiel Ports, DNS-Name der Datenbank, etc. Die Secrets umfassen geheime Daten, wie Passwörter und Zertifikate. Obwohl sie Secret

heißen sind diese nicht verschlüsselt, sondern nur in Basis 64 enkodiert. Die ConfigMap unter dem Pfad `./k8s-azure/config/k8s-app-config.yaml` ist als ein Kubernetes Manifest deklariert und enthält Schlüssel, um den DNS-Namen des Datenbankservers festzulegen, welcher auf den DNS-Namen des entsprechenden Kubernetes Serviceobjektes gesetzt ist. Zusätzlich ist noch der Port, der Datenbankname, Tabellenname und der Nutzernamen für den MySQL-Server definiert. Es bestehen auch Möglichkeiten die HTTPS-Verschlüsselung auszuschalten, sowie auch die Verschlüsselung der JWT-Token. Die Secrets können in Kubernetes als Manifeste definiert werden, sind in diesem Fall aber als env-Dateien umgesetzt und werden anschließend über den Kommandozeilenbefehl als Kubernetes Secret erstellt:

```
kubectrl create secret generic <NAME>  
--from-env-file=<DATEI-NAME>  
--from-file=<DATEI-NAME>
```

Der Grund dafür ist, dass im YAML-Format die Angaben in Basis 64 umgewandelt werden müssen. Mit dem Befehl übernimmt Kubernetes die Umwandlung automatisch von selbst. Zum anderen können direkt die Dateien, die die nötigen Zertifikate enthalten, verwendet werden, anstatt deren Inhalt in ein Manifest zu kopieren.

## 3.2 ARM-Templates

ARM Templates sind vom Prinzip her dasselbe wie Manifeste in Kubernetes. Der einzige Unterschied liegt darin, dass ARM Templates gewöhnlich im JSON-Format vorliegen und K8S Manifeste im YAML-Format. Beides sind Angaben zur Ressourcenerstellung nach dem Deklarativen Schema. Die Kubernetes Manifeste beschreiben Ressourcen in Kubernetes, wie Pods und Deployments. Die ARM Templates hingegen beschreiben Ressourcen in der Azure Cloud, wie zum Beispiel eine Azure Disk, einen Fileshare oder ein AKS-Cluster selbst. Das Template befindet sich zusammen mit einer Parameter Datei und einem Powershellskript zum Bereitstellen unter dem Pfad `./k8s-azure-aks/arm-templates/`. Das Template definiert die Bereitstellung eines AKS-Clusters, einer Azure Disk für die Datenbank und eines Speicherkontos mitsamt Dateifreigabe. Die Datei mit den Parametern stellt Angaben zur Erstellung der Ressourcen bereit, wie zum Beispiel den Speicherkontonamen, die Größe der Dateifreigabe, die Größe der Disk, etc.



### 3.2.1 Der Dienstprinzipal

Zwei der Parameter beziehen sich auf den Dienstprinzipal für das AKS-Cluster. Azure verwendet ein rollenbasiertes System zur Berechtigung eines Azure-Nutzers auf bestimmte Ressourcen. Dieser kann gemäß den Berechtigungen mit den Ressourcen verfahren. Da K8S als Speicherlösung für die Datenbank eine Azure Disk verwendet, benötigt das Cluster die nötigen Berechtigungen, um auf die Ressource zugreifen zu können. Auch für die zweite Gruppe an Ressourcen, die das Cluster anlegt, werden Berechtigungen benötigt. Da das Cluster kein Nutzer ist, bekommt es diese nicht via eines Nutzerkontos, sondern dem Gegenstück für Dienste. Diese werden als Dienstprinzipale bezeichnet. In den Parametern wird jeweils die *ClientID* des Prinzipals und ein dazu erstelltes *Secret* übergeben. Das Cluster nutzt diese anschließend, um sich bei Azure zu Authentifizieren. Damit beweist das Cluster dem Portal seine Identität.

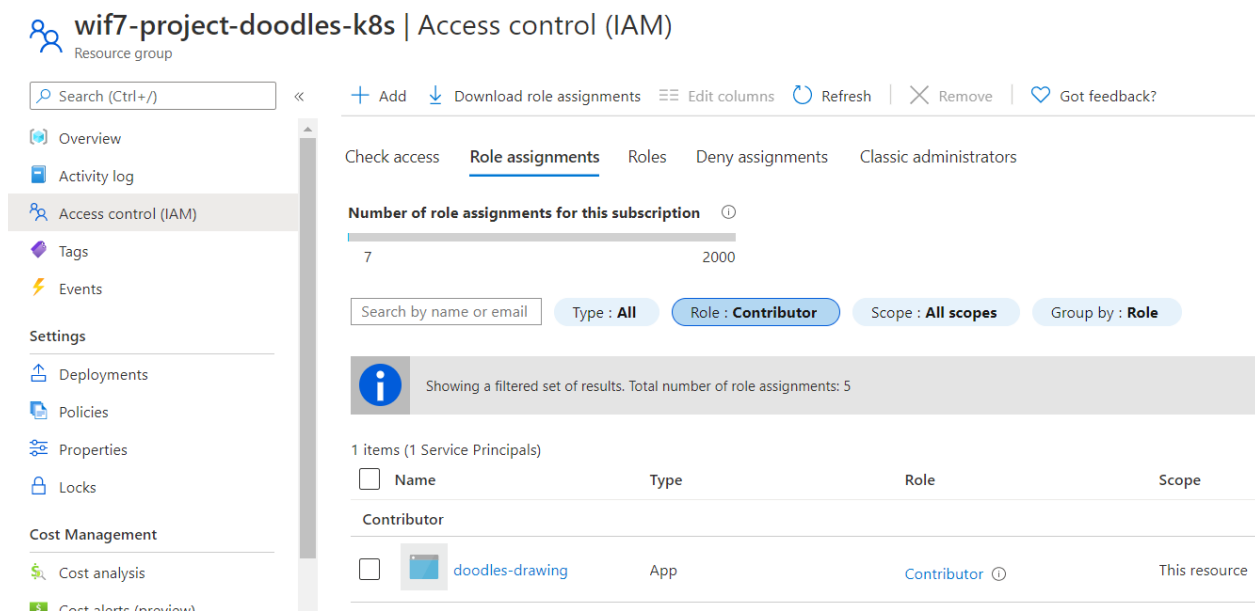


Abbildung 3 Screenshot des Access Control Panels in Azure

Quelle: Eigens erstellter Screenshot

Die Authentifizierung ist nur der erste Schritt. In einem zweiten Schritt wird, wie in Abbildung 3, dargestellt, auf Ebene der Ressourcengruppe dem Dienstprinzipal eine Rolle zugewiesen, die das Cluster dazu autorisiert auf die darin befindliche Disk zuzugreifen und dieses zu beschreiben. Mit dem Powershellskript, wird dieser Schritt nach Erstellung der Ressourcengruppe automatisch abgewickelt.

### 3.3 Verbinden mit dem bereitgestellten Cluster

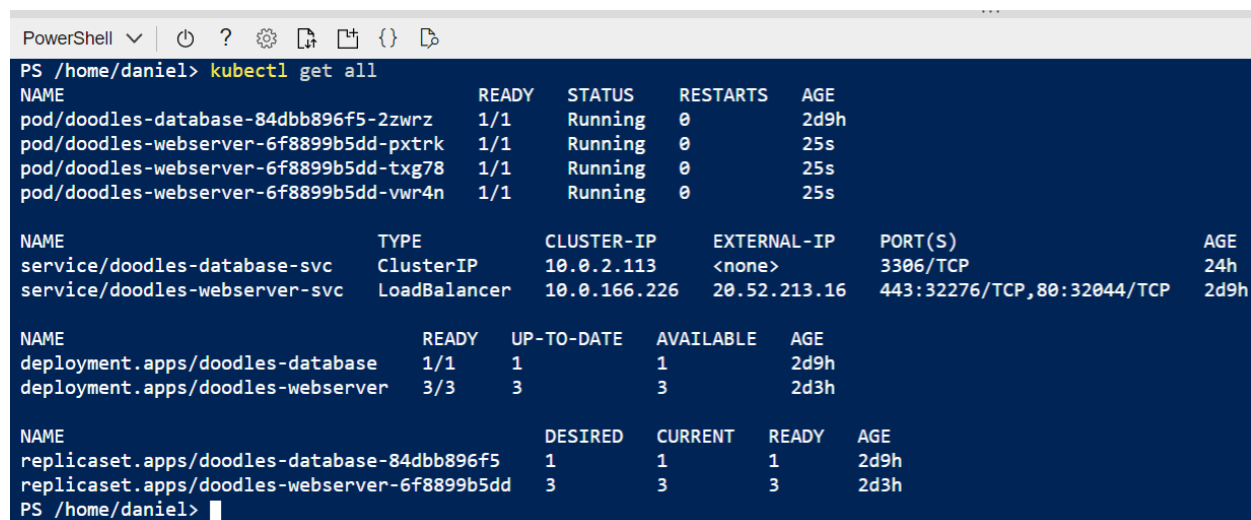
Das Cluster ist über eine öffentlich erreichbare IP zugänglich. Jedoch kann nicht jeder mit diesem kommunizieren. Es werden die nötigen Zertifikate zur Verifizierung und Verschlüsselung benötigt. Vorher muss jedoch noch in der Azure Cloud Shell der Kubectl-Client installiert werden:

```
Install-AzAksKubectl
```

Danach können die Zugangsdaten abgeholt werden. Der nachfolgende Befehl speichert das CA-Zertifikat des Clusters, zur Zertifikatsauthentifizierung des Clusters, und das Clientzertifikate, sowie dessen Privatschlüssel in der Datei `$HOME/.kube/config` ab. Das Clientzertifikat ist von einem CA-Zertifikat im Cluster signiert. Die Zertifikatsdaten können anschließend ausgelesen und auch in einem lokalen Kubectl-Client verwendet werden.

```
Import-AzAksCredential -ResourceGroupName 'wif7-project-k8s'-Name 'doodles-cluster'
```

Ist dieser Schritt abgeschlossen, kann direkt über Kubectl mit dem Cluster gesprochen und alle Ressourcen angelegt werden.



```
PS /home/daniel> kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/doodles-database-84dbb896f5-2zwrz	1/1	Running	0	2d9h
pod/doodles-webserver-6f8899b5dd-pxtrk	1/1	Running	0	25s
pod/doodles-webserver-6f8899b5dd-txg78	1/1	Running	0	25s
pod/doodles-webserver-6f8899b5dd-vwr4n	1/1	Running	0	25s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/doodles-database-svc	ClusterIP	10.0.2.113	<none>	3306/TCP	24h
service/doodles-webserver-svc	LoadBalancer	10.0.166.226	20.52.213.16	443:32276/TCP,80:32044/TCP	2d9h

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/doodles-database	1/1	1	1	2d9h
deployment.apps/doodles-webserver	3/3	3	3	2d3h

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/doodles-database-84dbb896f5	1	1	1	2d9h
replicaset.apps/doodles-webserver-6f8899b5dd	3	3	3	2d3h

```
PS /home/daniel>
```

Abbildung 4 Screenshot des Kubectl-Clients

Quelle: Eigens erstellter Screenshot

Abbildung 4 zeigt das Cluster mit einem Deployment für die Datenbank und einem für den Webserver an. Der Webserver trägt drei Replikate, während die Datenbank nur einen Pod besitzt. Zudem wurde ein ClusterIP Serviceobjekt für die Datenbank angelegt und ein Serviceobjekt vom Typ Loadbalancer mit der öffentlichen IP 20.52.213.16 für die Pods des Webserver Deployments.

### 3.4 Azure DNS-Zonen, Namensserver und Domänen

Die im ersten Kapitel angeführten Anforderungen verlangen das Bereitstellen der Website unter einer auflösbaren URL. Das heißt der Nutzer soll keine IP eingeben müssen, sondern eine gängige URL, die auf die entsprechende IP auflöst. Für diese Zwecke wurde bei dem Domänen Registrar GoDaddy die Domäne *daniel-testing.cloud* erworben und eine öffentliche DNS-Zone in Azure eingerichtet. Die DNS-Zone ist eine Azure Ressource mit der DNS-Einträge zu einer Domäne eingetragen werden können. Nach der Erstellung der Zone und einem Verweis auf die Namensserver im Domänen Registrar, können sogleich einige Einträge angelegt werden.

Name	Type	TTL	Value	Alias resource type	Alias target
@	NS	172800	ns1-05.azure-dns.com. ns2-05.azure-dns.net. ns3-05.azure-dns.org. ns4-05.azure-dns.info. Email: azuredns-hostmaster.... Host: ns1-05.azure-dns.com. Refresh: 3600 Retry: 300 Expire: 2419200 Minimum TTL: 300 Serial number: 1		
@	SOA	3600			
doodles	A	3600	-	Public IP Address	kubernetes-afde6d793410b4380affed65d79d8a6
www.doodles	CNAME	3600	doodles.daniel-testing.cloud		
www	CNAME	3600	doodles.daniel-testing.cloud		

Abbildung 5 Screenshot der DNS-Einträge in Azure  
Quelle: Eigens erstellter Screenshot

Abbildung 5 zeigt die in Azure angelegte DNS-Zone mit einem A-Record, der direkt auf die IP der Loadbalancer Ressource verweist. Die anderen Zwei sind CNAME-Records, die wiederum auf den A-Record verweisen. Abbildung 7 stellt den erfolgreichen Zugriff auf die Website über die URL [www.daniel-testing.cloud](http://www.daniel-testing.cloud) dar.

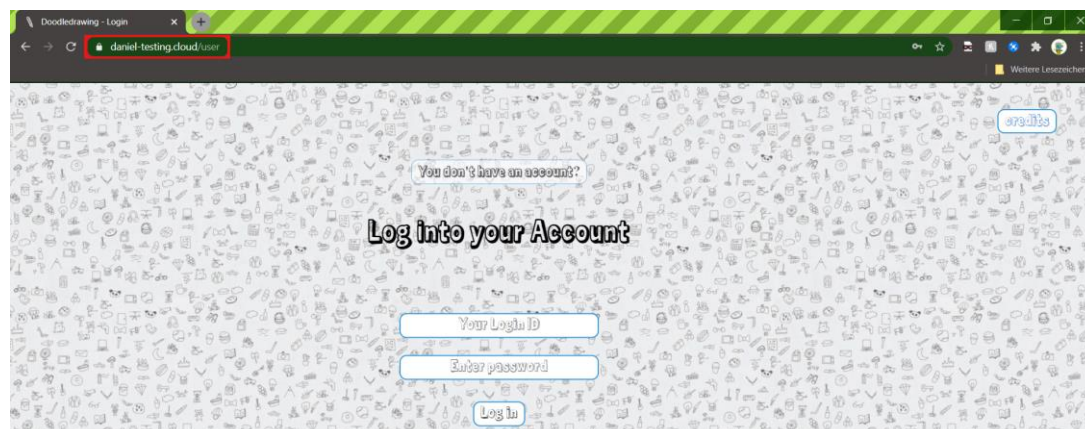


Abbildung 6 Screenshot der Eingabe der URL  
Quelle: Eigens erstellter Screenshot

### 3.5 Überblick der Infrastruktur

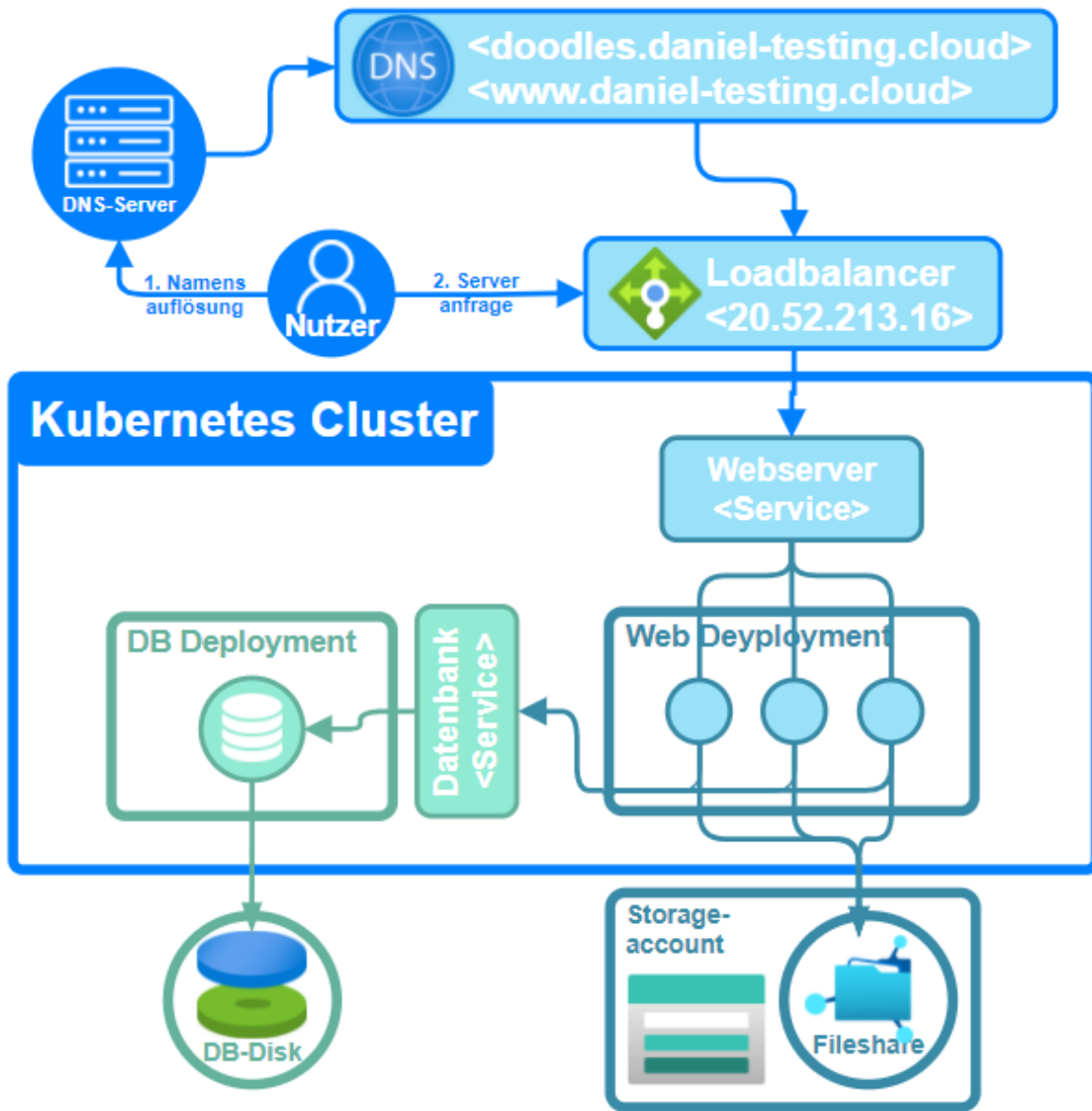


Abbildung 7 Schema der Kubernetes / Azure Infrastruktur  
Quelle: Eigentwurf erstellt mit <https://online.visual-paradigm.com/>

Abbildung 7 stellt einen schematischen Überblick der gesamten Infrastruktur innerhalb des Kubernetes Clusters, als auch in Azure dar. Der Nutzer will auf die Website zugreifen und gibt dessen URL ein. Da er die IP der Domäne nicht kennt, schickt dieser seinem DNS-Server eine Anfrage zur Namensauflösung. Dieser bedient den Nutzer entweder aus seinem Cache oder durchläuft den Prozess der Auflösung von den Rootservern, zu den Top-Level-Domains bis zu den Namensservern der Domäne *daniel-testing-cloud*. Da dort die Namensserver der Azure DNS-Zone hinterlegt sind, bedienen diese die Anfrage gemäß den Einträgen. Die DNS-Zone besitzt mehrere

Einträge, die auf die IP des Loadbalancers verweist. Die IP wird dem Nutzer zugeschickt, welcher in einem nächsten Schritt die IP anwählt. Im Cluster befinden sich zwei Deployments, jeweils eins für den Webserver und eines für die Datenbank. Der Webserver besitzt einen Service vom Typ LoadBalancer zu dem der öffentlicher Loadbalancer gehört. Das Deployment des Webserver besteht aus drei Pods. Verbindet sich der Nutzer nun mit der IP, leitet der Loadbalancer die Anfrage auf eine der drei Pods weiter. Das Deployment zur Datenbank besitzt nur einen Pod, der über ein Serviceobjekt mit dem Typ ClusterIP verfügt. Diese ist nur innerhalb Cluster und nicht von außerhalb erreichbar. Jegliche Kommunikation zwischen den Webserver Pods und der Datenbank verläuft über das Serviceobjekt. Sendet der Nutzer eine Zeichnung an den Server, wird sie im Datenvolumen des Servers gespeichert, welches als Dateifreigabe implementiert wurde. Die Metadaten sind in der Datenbank gesichert, dessen Daten auf einer Azure Disk persistieren.

## 3.6 Sicherheit

Die Anwendung verfolgt ein Login-System mit gängigen Sicherheitskonzepten, wie verschlüsselte Kommunikation und das Speichern der Nutzerpasswörter in Hashes.

### 3.6.1 Verschlüsselter HTTPS-Verkehr

Der Webserver besitzt zwei geöffnete Port. Zum einen den HTTPS-Port 443 zur Hauptkommunikation und den normalen unverschlüsselten HTTP-Port, der lediglich die Aufgabe hat Nutzer auf den HTTPS-Port umzuleiten. Das verwendete Zertifikat und der dazugehörige private Schlüssel wurde kostenlos und für eine Gültigkeit von 90 Tagen von der Non-Profit Organisation und Zertifizierungsstelle [Let's Encrypt](#) ausgestellt.

### 3.6.2 Authentifizierung und Bcrypt Passwort-Hashes

Die Authentifizierung funktioniert über eine Login-Maske. Der Nutzer muss bei Kontokreierung oder Login seinen Nutzernamen und ein Passwort vorlegen. Das Passwort wird als Hash mittels dem Modul *Bcrypt* umgewandelt und in der Datenbank abgelegt. Beim Login wird das erhaltene Passwort mit dem Hash in der Datenbank überprüft. Ist das Passwort gültig, wird der Nutzer eingeloggt. Eine erfolgreiche Anmeldung ist gekennzeichnet durch die Übergabe eines JWT, dass bei jedem Serveraufruf beigelegt wird und den Aufruf vom jeweiligen Nutzer authentifiziert.

### 3.6.3 JWT und HTTP-Only Cookies

Ein JWT ist eine Zeichenkette bestehend aus drei Teilen. Einem Header, der Metadaten des Tokens enkodiert, eine Verifizierungssignatur und eine Payload. Die Payload ist eine Zeichenkette, die JSON-Objekte enkodiert. Die Validierungssignatur ist eine spezielle Zeichenkette, die bei der Erstellung im Server durch einen geheimen privaten Schlüssel erzeugt wird. Sie stellt eine Art einzigartige Unterschrift für das gesamte Token dar. Wird das Token an den Nutzer übergeben und bei einer erneuten Anfrage an den Server übergeben, womöglich ein anderer Pod, kann dieser anhand der Signatur den gültigen Ursprung des Tokens validieren und somit den Anfragenden als Nutzer authentifizieren.

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Ijg4Njg2ODZkMDNhNmQ0OTYiLCJ1c2VybmFtZV9kaXNwbGF5IjoiaWVudC5kaDQ0OTY1MzZ9.kDxWw2Y-0YB-78H5_-TFICWA-4dYT5LQN8gd05S10aW0nA
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "RS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "id": "8868686d03a6d496",  "username_display": "Daniel",  "iat": 1609359336,  "exp": 1609402536}
```

Abbildung 8 Screenshot der Decodierung eines JWT  
Quelle: Eigens erstellter Screenshot des JWT-Debuggers unter <https://jwt.io/>

Abbildung 8 zeigt auf der rechten Seite ein JWT und auf der linken Seite den Headerinhalt und dessen Payload. Als Payload wird der Nutzernamen, sowie dessen ID abgespeichert. Nach der Validierung des Ursprungs des Tokens greift der Server auf die Daten im Token zu und weiß damit um exakt welchen Nutzer es sich handelt. Der Wert *iat* und *exp* encodieren das Ausstellungs- und Verfalldatum des Tokens.

Abbildung 9 zeigt die Validierung desselben Tokens. Eine erfolgreiche Validierung bedeutet, dass das Token von einer Instanz des Webserver ausgestellt wurde und nicht modifiziert wurde. Wird es nachträglich verändert, wie zum Beispiel das Modifizieren der ID, und anschließend den Server geschickt, schlägt der Validierungsprozess fehl und der Nutzer gilt als nicht authentifiziert.

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Ijg4Njg2ODZkMDNhNmQ0OTYiLCJ1c2VybmFtZV9kaXNwbGF5IjoiaRGFuaWVzIiwiaWF0IjoxNjA5MzU5MzZMLCJleHAiOiJlMjMDk0MDI1MzZ9.kDUxt2pYn0YgZKkxWqMYfHdReO6TfLjo59arCQ08uzxWw2Y-0YB-78H5\_-TFICWA-4dYT5LQN8gd05S0aW0Na

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  pw1E76a07Vb2PRG/5Q+0ckZbugXS
  Ac3hoovL76/ToflDxABuQSHKnWMC
  AwEAAQ==
  -----END PUBLIC KEY-----
)
```

Signature Verified

SHARE JWT

Abbildung 9 Screenshot der Verifizierung eines JWT

Quelle: Eigens erstellter Screenshot des JWT-Debuggers unter <https://jwt.io/>

Anders als bei Session Tokens ist bei der Verwendung von JWT kein abspeichern und auslesen von Werten in der Datenbank nötig. Als gute Analogie lässt sich der Token als ein versiegelter Brief verstehen, den der Nutzer bekommt und in dem sich eine eindeutige Identifikationsdaten für ihn befindet. Er kann mit dem versiegelten Brief weggehen und später wiederkommen. Der Aussteller, in diesem Zusammenhang der Webserver, kann anhand des Siegels feststellen, dass er in der Tat den Brief zu einem früheren Zeitpunkt ausgestellt und versiegelt hat und den Nutzer somit identifizieren. Ist das Siegel gebrochen, also wurde das Token nachträglich verändert, kann der Aussteller dies feststellen und den Nutzer den Zugang verweigern. Der Nutzer muss zu keinem Zeitpunkt den Inhalt des Briefes kennen. Er muss ihn nur aufbewahren und zur Authentifizierung wieder mitbringen und übergeben.

Das JWT wird als HTTP-Only Cookie unter dem Name *doodle\_token* im Browser abgespeichert. Die Speicherung und Übergabe des JWT wird vom Browser automatisch abgewickelt. Auf dem Webserver ist die Lebensdauer eine JWT auf zwölf Stunden gesetzt, kann aber mit Umgebungsvariablen angepasst werden. Die Lebensdauer des Cookies, wird dementsprechend auf denselben Wert gesetzt. Ein HTTP-Only Cookie ist ein spezielles Cookie, das den Browser anweist, den Cookie nur für den Datenverkehr zu benutzen. Das heißt der Nutzer oder andere Akteure mit niederen Motiven können nicht ohne weiteres Aushebeln von Sicherheitsmechanismen das Cookie modifizieren.

### 3.6.4 Verschlüsselung mit Crypto

Da nur der Webserver den Inhalt des Cookies lesen und verstehen muss, wird das JWT noch zusätzlich mit dem internen Crypto Modul verschlüsselt. Dafür wird der Algorithmus *aes256*, ein 128-bit langer Hexadezimalschlüssel als Initialvektor und ein 256-bit Hexadezimalschlüssel als Verschlüsselungsschlüssel verwendet.

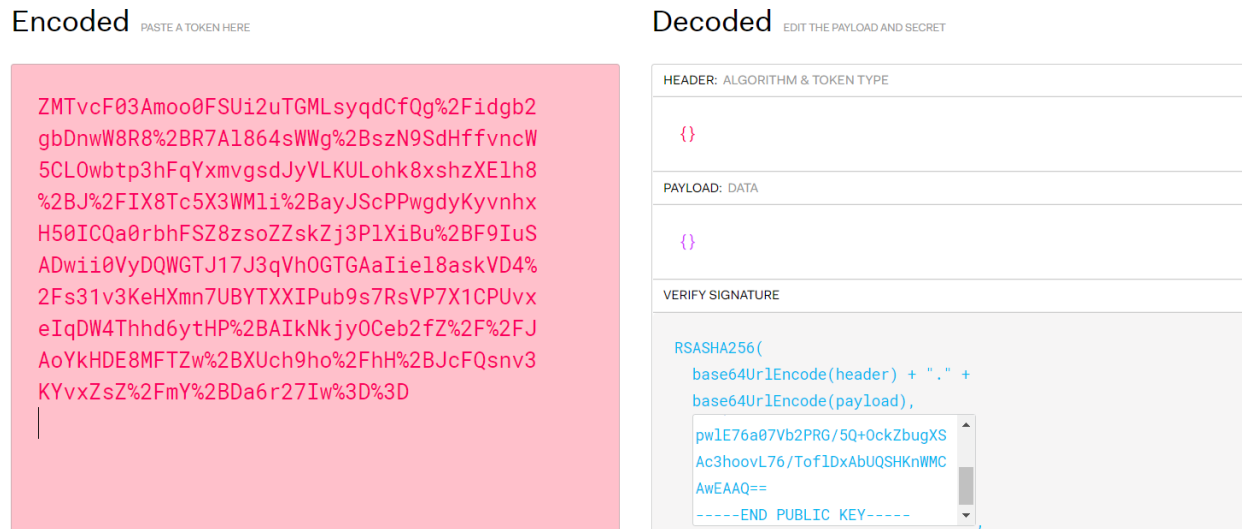


Abbildung 10 Screenshot eines verschlüsselten JWT  
Quelle: Eigens erstellter Screenshot

Abbildung 10 zeigt dasselbe JWT nun in verschlüsselter Form dar. Selbst wenn Akteure mit bösen Hintergedanken an das Token kommen, können diese es weder in unverschlüsselter Form verändern, noch können sie es in jetziger verschlüsselter Form entschlüsseln. Nur noch der Webserver mit den entsprechenden Schlüsseln, kann den Token auslesen und dessen Ursprung verifizieren.

Der private, öffentliche Schlüssel und Algorithmus für die JWT-Validierung und die beiden Schlüssel, als auch der Algorithmus zur Verschlüsselung werden aus Umgebungsvariablen ausgelesen, die mittels der Einbindung der K8S Secrets besetzt werden.



## 4 Entwicklungsumgebung und Tool

### 4.1 Entwicklungsumgebung

Zur Entwicklung wurden verschiedene Tools und Entwicklungsumgebungen zu Rate gezogen. Als Codeeditor wurde *Visual Studio Code* verwendet mit verschiedenen Plugins. Für die Kommunikation und Abfragen der Datenbank wurde *MySQL Workbench* verwendet. Lokale Tests mit dem erstellten Webserver-Container und MySQL-Container wurden mittels *Docker Desktop for Windows* durchgeführt. Vor der Erstellung in Azure wurden auch mit K8S lokale Tests durchgeführt. Hierfür fand das in *Docker Desktop for Windows* integrierte Kubernetes Cluster seinen Einsatz.

### 4.2 Postman

*Postman* ist ein Entwicklungswerkzeug, dass es ermöglicht benutzerdefinierte HTTP-Anfragen und HTTPS-Anfragen an den Server zu senden. Mit diesem Programm wurden die Serverendpunkte des Webserver getestet und abgefragt, als noch kein Abfragemasken und zugehöriger Javascriptcode auf der Website implementiert war. Mit benutzerdefinierbaren Headern, Bodys und das automatische Speichern und Verwalten gesendeter Cookies, stellte sich Postman als ein sehr hilfreiches Tool heraus.

#### 4.2.1 Senden einer Suchanfrage

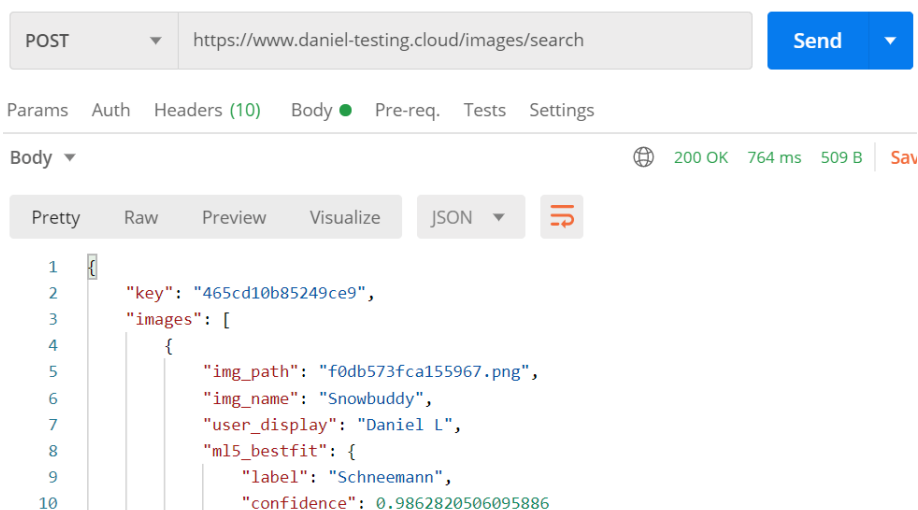


Abbildung 11 Screenshot einer Anfrage in Postman  
Quelle: Eigens erstellter Screenshot

Abbildung 11 zeigt eine POST-Anfrage an den `/images/search` Endpunkt des Webservers unter der URL <https://dooldes.daniel-testing.cloud>. Dieser ist für die Bearbeitung von Suchanfragen zuständig und nimmt die Parameter `Nutzername`, `Bildname` und `Zuordnung eines Bildes zu einer Klasse` an, entsprechend der Suchmaske auf der Website. Anschließend durchforstet er mit den Werten die Datenbank und liefert die Befundnisse als Antwort zurück. In der Abbildung x sieht man das ein Array aus Bildern zurückgegeben wurde, angefangen mit einem Bild namens *Snowbuddy*, der Klassifizierung *Schneemann*, dem Dateinamen *f0db573fca155867.png*, etc.

### 4.3 Fiddler

*Fiddler* ist ein weiteres Entwicklungswerkzeug und praktisch das Gegenstück zu *Postman*. Während *Postman* Anfragen erstellt, sendet und auf deren Antwort wartet, überwacht *Fiddler* den gesamten Netzwerkverkehr zwischen Rechner und Server. Dabei listet *Fiddler* alle ausgehenden Anfragen und eingehenden Antworten zum Auslesen auf. Dies ist hilfreich beim Testen der Graphischen Oberflächen der Website und dessen Kommunikation via HTTP.

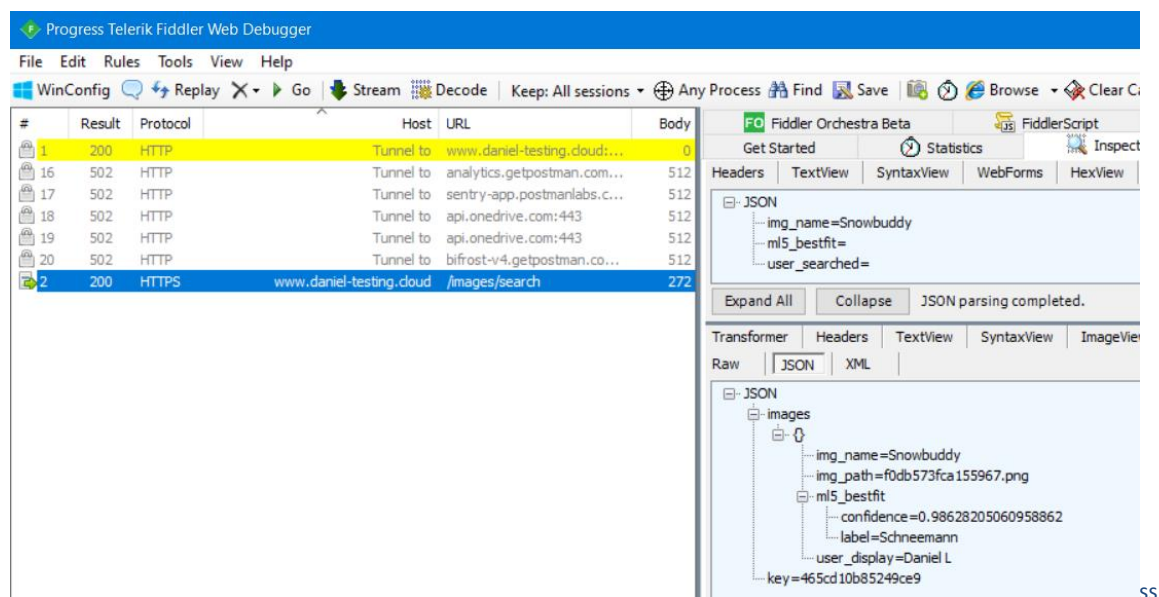


Abbildung 12 Screenshot der HTTPS-Anfrage in Fiddler

Quelle: Eigens erstellter Screenshot

Abbildung 12 zeigt die vorher von *Postman* gesendete Anfrage, die von *Fiddler* abgefangen und aufgelistet wurde. Es wurde eine Suchanfrage nach dem Bildnamen *Snowbuddy* abgeschickt und als Antwort ein Array aus den Befundnissen geliefert. Ebenfalls kann die Suchmaske der Seite selbst verwendet werden, während *Fiddler* die gesendeten Anfragen überwacht und auflistet.

## Eidesstatliche Erklärung

Ich versichere, dass ich das Projekt, den beigelegten Sourcecode, sowie die schriftliche Arbeit vollständig selbst angefertigt habe.

Weitere verwendete externe Ressourcen sind in den Credits der Website aufgelistet. Diese umfassen Hintergrundbilder, Icons, Fonts und die 404-Seite.

15.01.21, Dentlein

Ort, Datum

Daniel Lander

Unterschrift