Omdat Aucxis met een Microsoft georiënteerde werkwijze zit, moet er worden aangetoond dat het mogelijk is om dit te realiseren op GCP. Voor deze Proof Of Concept (POC) wordt er een CI/CD pijpleiding geconfigureerd. Hierin wordt er een simpele .NET Core applicatie gecompileerd. Ook de meegeleverde testen worden uitgevoerd tijdens de compilatie. Hierna wordt de applicatie en de bijbehorende componenten gecomprimeerd en naar een lokale fileserver geüpload. De gecompileerde applicatie kan dan uitgebreid getest worden in een lokale test omgeving. Ook is er een stap voorzien in de pijpleiding om de gecompileerde applicatie tijdelijk op het Cloud platform te bewaren. Dit voor het geval er iets verkeerd loopt tijdens het testen of tijdens het uploaden.

De .Net applicatie is een simpele .Net core console applicatie. Het bestaat uit een klasse, ‘MessageUtil’ (verwijzing). Deze klasse bevat een variabele, twee constructors, een methode om de variabele op te vragen, een methode om de variabele te tonen met een toevoegsel en een hoofdmethode die uitvoerbaar is.

using System;

using System.Threading;

namespace MessageUtil

{

public class MessageUtilProgram

{

private String p\_message;

private MessageUtilProgram() { }

public MessageUtilProgram(String message) {

p\_message = message;

}

public String Message

{

get { return p\_message; }

}

public String SaluteMessage(String m)

{

Console.WriteLine("Hello\n${0}", m);

return "hello" + m;

}

static void Main(string[] args)

{

MessageUtilProgram mup = new MessageUtilProgram(@" \_ \_ \_ \_ \_ \_ \_

/ /\ /\\_\ /\ \ /\_/\ /\ \ /\ \ / /\

/ / \ / / / \_ / \ \ \ \ \ \ \\_\ \ \ \ / / \

/ / /\ \ \ \ \\_\_ /\\_\ / /\ \ \ \ \ \\_\_/ / / /\ \\_\ / / /\ \\_\_

/ / /\ \ \ \ \\_\_\_\ / / // / /\ \ \ \ \\_\_ \/\_/ / /\/\_/ / / /\ \\_\_\_\

/ / / \ \ \ \\_\_ / / / // / / \ \\_\ \/\_/\\_\_/\ / / / \ \ \ \/\_\_\_/

/ / /\_\_\_/ /\ \ / / / / / // / / \/\_/ \_/\/\_\_\ \ / / / \ \ \

/ / /\_\_\_\_\_/ /\ \ / / / / / // / / / \_/\_/\ \ \ / / / \_ \ \ \

/ /\_\_\_\_\_\_\_\_\_/\ \ \ / / /\_\_\_/ / // / /\_\_\_\_\_\_\_\_ / / / \ \ \ \_\_\_/ / /\_\_ /\_/\\_\_/ / /

/ / /\_ \_\_\ \\_\/ / /\_\_\_\_\/ // / /\_\_\_\_\_\_\_\_\_\/ / / /\_/ //\\_\_\/\_/\_\_\_\\ \/\_\_\_/ /

\\_\\_\_\_\ /\_\_\_\_/\_/\/\_\_\_\_\_\_\_\_\_/ \/\_\_\_\_\_\_\_\_\_\_\_\_/\/\_/ \\_\/ \/\_\_\_\_\_\_\_\_\_/ \\_\_\_\_\_\/

");

Console.WriteLine("{0}", mup.Message);

//mup.SaluteMessage(mup.Message);

Thread.Sleep(60000);

}

}

}

Daarnaast is er ook een testklasse voorzien, ‘MessageUtilTest’ (verwijzing). Hierin staan er twee methodes. De eerste methode test de constructor die de variabele moet instellen. De tweede methode test een van de print methodes.

using Microsoft.VisualStudio.TestTools.UnitTesting;

using MessageUtil;

namespace MessageUtilTest

{

[TestClass]

public class MessageUtilTests

{

[TestMethod]

public void ConstructWorks()

{

string testmessage = "Test";

MessageUtilProgram mup = new MessageUtilProgram(testmessage);

string value = mup.Message;

Assert.AreEqual(testmessage, value,"Message didn't set correctly");

}

[TestMethod]

public void SaluteWorks()

{

string testmessage = "Test";

string expected = "helloTest";

MessageUtilProgram mup2 = new MessageUtilProgram(testmessage);

string value = mup2.SaluteMessage(mup2.Message);

Assert.AreEqual(expected, value, "Message didn't salute correctly");

}

}

}

De applicatie wordt in beide gevallen onveranderd gebruikt zodanig dat een potentieel verschil zichtbaar wordt. Dit is de basis waaruit vertrokken is voor de vergelijkende POC. Om de prijs en het aantal benodigde producten zo laag mogelijk te houden, is er in dit onderzoek gekozen om Git en GitHub te gebruiken als versiebeheersysteem. GitHub is volledig ondersteund door beide platformen. Ook voorzien beide Cloud platformen applicaties op GitHub om automatisch de gewenste repositories te verbinden aan de CI/CD pijpleidingen.

Het uploaden van de gecomprimeerde applicatie wordt gedaan door middel van Secure File Transfer Protocol (SFTP) op een Linux Ubuntu machine. Deze maakt verbinding met een Docker container die lokaal op een Windows Server 2019 draait. Deze Docker container (verwijzing) deelt een directory met Windows zodanig dat dit volledig modulair is met andere besturingssystemen of situaties. SFTP werkt onderliggend op basis van Secure Shell (SSH). Om verbinding te maken is dus een wachtwoord nodig. Dit is een probleem aangezien de virtuele systemen in de Cloud niet interactief mogen zijn. Dit is opgelost door het gebruik van SSHPASS (verwijzing). Dit Linux pakket heeft de mogelijkheid om aan SSH-toepassingen het wachtwoord mede te geven. Weliswaar zonder encryptie van het wachtwoord. Hierdoor is het wachtwoord leesbaar. Er wordt enkel op deze wijze gewerkt om het geheel relatief simpel te houden. Voor productie omgevingen zou er met SSH-sleutels gewerkt moeten worden. De SFTP operatie op de Linux Ubuntu in de Cloud wordt uitgevoerd doormiddel van een script (verwijzingen). Dit Bash script zorgt ervoor dat het SSHPASS pakket beschikbaar is en voert de SFTP transactie uit.

#! /bin/bash

apt-get update

apt-get -y upgrade

apt-get -y install sshpass

sshpass -p 'haai' sftp -o StrictHostKeyChecking=accept-new -P 5151 -oBatchMode=no -b - TestU@213.119.34.127 << !

cd documents

put /workspace/MessageUtil/bin/Release/netcoreapp3.1/win10-x64/messageutil-win10-x64.tar.gz

bye

!

De Cloud specifieke configuratie bestanden worden in de volgende secties verder verduidelijkt. Voor GCP in sectie (verwijzing) en voor het Azure DeVops platform sectie (verwijizng).