Implementeren van Pydev in IBM Developer for z/OS om Python-applicaties uit te voeren op de mainframe.

Alif Monnoye.

Scriptie voorgedragen tot het bekomen van de graad van
Professionele bachelor in de toegepaste informatica

Promotor: Dhr. L. Blondeel
Co-promotor: Dhr. N. Sletten
Instelling: Den Norske Bank (DNB)
Academiejaar: 2023–2024
Eerste examenperiode

Departement IT en Digitale Innovatie .

HO
GENT

Woord vooraf

Dit is een proof of concept die ik wil uitwerken omdat dit hulp kan bieden bij de modernisatie van de mainframe systemen van IBM. Op het moment van schrijven heb ik al stage gedaan bij 2 bedrijven die gebruikmaken van een mainframe; zij gebruiken allebei IBM Developer for zOS om applicaties te schrijven in verschillende soorten programmeertalen zoals COBOL, PL/1, Python en Java. Python is momenteel één van de meest gebruikte programmeertalen en IBM doet veel inspanning om dit zo efficiënt mogelijk te laten werken op zijn systemen. De skills in de 'oudere' programmeertalen zoals COBOL en PL/1 zijn sterk aan het afnemen dus is de inbreng van Python bijna een must om nog goede programmeurs te vinden. Een goede editor is ook nodig wat momenteel niet het geval is in IDz. In dit onderzoek maak ik dan ook een stappenplan om dit zo goed mogelijk op te stellen.

Ik zou Njaal Sletten en Stian Botnevik van DNB, Noorwegen willen bedanken voor de hulp en inspiratie bij het schrijven en uitwerken van dit onderzoek. Ze hielpen allebei met het technische gedeelte en Stian kwam met de probleemstelling.

Samenvatting

IBM heeft voor zijn IBM Z Systems de deur opengezet naar vernieuwing door Java en Python toegankelijk te maken op de mainframe. Hoewel deze talen ondersteund worden, is het niet altijd even makkelijk om programma's hierin te schrijven. IBM Developer for z/OS is een stap in de goede richting door zijn functionaliteit om in een bekend programma bestanden die opgeslagen zijn op de mainframe te openen en aan te passen. Aangezien dit Eclipse based is, is er geen Python-interpreter waardoor Python scripts moeten worden aangepast in een text editor.

Deze paper is een proof of concept voor het opstellen van Pydev als Python IDE in IBM Developer for z/OS. Hierdoor kunnen Python developers efficiënter code schrijven en aanpassen in de Unix System Services omgeving van de mainframe.

In dit onderzoek wordt er gewerkt met de Pydev-plug-in voor Eclipse en worden er instructies gegeven over hoe dit ingesteld wordt. Ook wordt er onderzocht of deze scripts uitgevoerd kunnen worden via IBM Developer for z/OS in de mainframe-omgeving.

Om dit onderzoek te testen, zal er een Python API geschreven worden in IBM Developer for z/OS. Deze zal uitgevoerd worden via dit programma op de Unix System Services omgeving. Deze omgeving brengt andere problemen met zich mee die opgelost zullen worden.

Het resultaat is een volledige proof of concept om een Python IDE in IBM Developer for z/OS op te zetten. Verder volgt configuratie om een verbinding te maken met de Unix System Services omgeving om Python-applicaties in uit te voeren. In de testfase van dit onderzoek wordt een API geschreven en een stappenplan gegeven over de benodigdheden om Python-applicaties in de gebruikte omgeving uit te voeren. Dit is belangrijk voor de programmeurs van DNB die werken in dit programma en voorlopig hun Python code moeten testen via een externe ssh connectie.

Inhoudsopgave

Lijst van figuren

1

Inleiding

1.1. Probleemstelling

IBM heeft voor zijn IBM Z Systems de deur opengezet naar vernieuwing door Java en Python toegankelijk te maken op de mainframe. Hoewel deze talen ondersteund worden, is het niet altijd even makkelijk om programma's hierin te schrijven. IBM Developer for z/OS is een stap in de goede richting door zijn functionaliteit om in een bekend programma scripts die opgeslagen zijn op een mainframe, te openen en aan te passen. Aangezien dit Eclipse based is, is er geen Python-interpreter die gebruikt kan worden, waardoor Python-applicaties enkel kunnen worden geopend en aangepast in een text editor. Dit komt vanzelfsprekend niet met een syntaxcheck, code completion, enz.

Dit is een probleem voor developers zoals Stian Botnevik van DNB om Python-code te schrijven in dit programma. Hoewel scripts vaak ook lokaal worden geschreven in een programma zoals Visual Studio Code en dan worden overgezet naar de mainframe, moeten er nogsteeds aanpassingen gebeuren door de verschillen in runtime environment. In een klein Python programma lukt dit wel nog in een text editor maar als ze wat groter zijn, wordt dit veel moeilijker.

1.2. Onderzoeksvraag

In dit onderzoek wordt er gewerkt met de Pydev plug-in voor Eclipse en worden er stap voor stap instructies gegeven over hoe dit opgezet wordt in IBM Developer for z/OS. Aangezien dit programma verschillend is van Eclipse, kunnen er problemen opduiken die niet voorkomen in het standaard Eclipse-programma.

Eens dit is opgezet, wordt er onderzocht of deze scripts kunnen worden uitgevoerd in de mainframe via IBM Developer for z/OS. Hier wordt er een Python-API geschre-

ven die doormiddel van de Z Open Automation Utilities 2 datasets zal lezen en 1 zal schrijven. In deze testfase wordt er een oplossing geboden voor de meest voorkomende problemen bij het uitvoeren van Python-applicaties op de mainframe.

1.3. Onderzoeksdoelstelling

De doelstelling is om een Python-interpreter op te zetten in IBM Developer for z/OS door middel van Pydev. Verder volgt er een stappenplan om een terminal te openen die verbonden is met de mainframe om Python applicaties in uit te voeren.

1.4. Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4 wordt de gebruikte terminologie in detail besproken samen met andere onderwerpen die relevant zijn voor het onderzoek.

In Hoofdstuk 5 wordt de proof of concept duidelijk weergegeven en besproken.

In Hoofdstuk 6 wordt de connectie met de USS omgeving opgezet in IDz.

In Hoofdstuk 7 wordt de runtime environment getest om Python scripts in uit te voeren.

In Hoofdstuk 8 worden de functies van Pydev getest door een Python API te schrijven. Deze wordt ook direct getest.

In Hoofdstuk 9 wordt de testmethode geëvalueerd.

In Hoofdstuk 10, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2

Stand van zaken

2.1. Een andere manier van werken

Veel bedrijven die met een mainframe werken, hebben het probleem dat ze onvoldoende mensen vinden met de nodige kennis over deze technologie. De introductie van Unix-systemen op de mainframe in het jaar 2000 (Mertic, 2020) had dit probleem wat verminderd, maar zeker niet weggewerkt. De oorzaak van deze situatie is te vinden in de oude technieken die het systeem gebruikt. COBOL en PL/1 zijn niet de meest aantrekkelijke programmeertalen om te leren, en mensen kiezen liever Bash als scriptingtaal in plaats van REXX. Hoewel IBM veel inzet op documentatie en online leerplatformen zoals IBM Z Xplore, blijft het tekort aan ervaren mensen nog steeds te hoog.

De mainframe wereld zal zich dus moeten aanpassen aan de vaardigheden van de mensen door over te schakelen naar beter gekende manieren van werken. Python bijvoorbeeld is een programmeertaal die steeds meer populariteit krijgt sinds haar ontstaan in de vroege jaren 90. Dit is niet enkel bij ervaren programmeurs; ook mensen die net beginnen te programmeren, kiezen hier steeds vaker voor. Dit is vooral te danken aan de gebruiksvriendelijkheid, de inzetbaarheid voor verschillende doeleinden en actieve community. (Johnson, 2023)

IBM heeft dit opgemerkt en een Python-compiler en -interpreter ontwikkeld voor z/OS, genaamd IBM Open Enterprise SDK for Python. Hierdoor bestaat de mogelijkheid om met Python interactie te hebben met z/OS om bijvoorbeeld applicaties te ontwikkelen of de resources van het systeem beheren. Door de Unix-omgeving heeft de programmeur ook geen geavanceerde kennis van z/OS nodig. (Klaey, 2023)

2.2. Bedrijven in het werkveld

Een bank is een goed voorbeeld van een bedrijf dat gebruik maakt van Z Systems van IBM. Dit valt te concluderen uit het feit dat maar liefst 92 van de top 100 van banken wereldwijd een mainframe gebruiken. Dit is ook logisch, aangezien deze systemen gemiddeld 12.6 miljard financiële transacties per dag verwerkt. (Wagle, 2017)

Door dit aantal is de nood aan een mainframe toch niet te onderschatten. Het gaat evenwel niet enkel om het aantal transacties dat deze machine kan uitvoeren, ook de snelheid, schaalbaarheid en beveiliging van deze systemen spelen een grote rol. Niet alleen banken gebruiken deze technologie; ook de bedrijven die tot de top 10 van verzekeringsmaatschappijen behoren, maken allemaal gebruik van een IBM mainframe (Tozzi, 2022)

2.3. De Skill gap

Het is nog steeds moeilijk om nieuw talent aan te trekken in de mainframe wereld. Uit een onderzoek van Deloitte (2020) blijkt dat 79 % van de projectleiders moeilijkheden heeft met het zoeken naar mensen met de juiste skillset. Hetzelfde onderzoek toont ook aan dat er in de teams zelf een groot verschil is van kennis en vaardigheden.

Hoewel dit systeem gebruikt wordt door 71 % van de Fortune 500-bedrijven (Tozzi, 2022) , is de Skill gap nog steeds te groot, waardoor veel bedrijven vrezen voor een groot tekort aan werknemers om deze Z Systems te onderhouden.

Volgens Petra Goude (2023) zijn er verschillende manieren om dit probleem aan te pakken. Zo kunnen bedrijven lessen geven over mainframes en het gebruik ervan. Ze vertelt ook dat de vereiste vaardigheden beter gecomplimenteerd moeten worden en kunnen leiden naar een belonende carrière op lange termijn.
Hoewel dit zou helpen, vindt ze dit niet de kern van het probleem. Het zijn de oude technieken die mensen niet aantrekken. Ze stelt voor om meer te investeren in hedendaagse technologie en deze te installeren op de mainframe. Dit kan gaan over dezelfde test -en deployment technieken, maar ook over hedendaagse programmeertalen zoals Java of Python. Dit zou kunnen door middel van API's en zou nieuwe, jongere werknemers aantrekken.

2.4. Toch nog een kleurrijke toekomst

Ondanks deze probleemstelling ziet de toekomst er toch nog goed uit voor deze systemen. Zo wordt er meer gemoderniseerd met bijvoorbeeld modernere programmeertalen. Er wordt ook voorspeld dat we een introductie van DevOps en

self-service benaderingen gaan zien. (Pennaz, 2023)

Momenteel zijn Python en Java beschikbaar als programmeertaal op de mainframe naast PL/1 en COBOL. Sinds deze introductie past al bijna tweederde van de gebruikers op de mainframe Java op een bepaalde manier toe. (Watts, 2018)

3

Methodologie

3.1. Fase 1: Literatuurstudie

In deze fase wordt er informatie verzameld over relevante vakliteratuur. Dit gebeurt door betrouwbare bronnen op te zoeken en te bestuderen.
De tijd die hiervoor gereserveerd wordt, is 5 weken. Het resultaat is een volledig overzicht van de omgeving waarin dit onderzoek zich bevindt.

3.2. Fase 2: Pydev opstellen

Het doel in deze fase is het implementeren van Pydev in IDz. Zodra dit gedaan is, begint de implementatie van Pydev, wat 3 weken in beslag neemt. Na deze implementatie kunnen Python-applicaties worden geopend met Pydev voor functies zoals code completion, syntax check en code analyse.

3.3. Fase 3: Runtime environment opzetten

In deze fase wordt er in IDz een connectie gemaakt met de USS omgeving. Dit duurt 2 weken en gebruikers kunnen dan via IDz hun Python-applicaties uitvoeren in de USS omgeving.

3.4. Fase 4: Pydev en connectie met USS testen

In de testfase wordt er een testomgeving opgezet in USS via IDz en wordt er een Python API ontwikkeld. Hiervoor zijn 2 weken vrijgehouden. Het resultaat hiervan is een weergave van de mogelijkheden en beperkingen van een USS-connectie via IDz evenals de efficiëntie om Python-applicaties te schrijven met Pydev.

3.5. Fase 5: Evaluatie voorbereiden

Deze fase omvat de conclusie van de bachelorproef en de voorbereiding op de eindevaluatie. Dit duurt 2 weken.

4

Literatuurstudie

4.1. De IBM Z Systems omgeving

Het is belangrijk om te weten wat een mainframe is en wat de hoofdpunten van de technologie zijn, maar het is niet zo eenvoudig om een goede definitie van deze term te geven. Een mainframe is ontwikkeld door IBM en wordt vooral gebruikt door grote bedrijven om belangrijke applicaties te hosten en/of veel transacties te kunnen uitvoeren. Hoewel dit ook mogelijk is op een kleinschalige server, zou het resultaat niet hetzelfde zijn omdat mainframes miljarden transacties per dag kunnen uitvoeren zonder enige vertraging. (BasuMallick, 2023)

De mainframe wordt vaak vergeleken met een traditionele server die te vinden is in een datacenter. Deze vergelijking is niet onterecht omdat ze wel een gelijkaardige functie hebben. Een mainframe zoals de hedendaagse IBM z16 kan 19 miljard transacties per dag uitvoeren, wat verklaart waarom deze systemen gebruikt worden door 71 % van de Fortune 500-bedrijven wereldwijd. Deze machine heeft ook 40 terabyte werkgeheugen, dat is 1200 keer meer dan hedendaagse high-performance computers. (Tozzi, 2022)

De z16 is ook “quantum safe”: de bedreiging van quantum computers in de toekomst blijft groeien en er is nog geen directe oplossing voor. IBM heeft hiervoor geïnvesteerd in Crypto Express 8S hardware security modules om data op de mainframe te beschermen en quantum safe te maken. De nieuwe modules bevatten nieuwe quantum safe encryptie algoritmes die geëvalueerd zijn door het US National Institute of Standards and Technology. (Sayer, 2022)

Op deze machines worden data niet opgeslagen in de ASCII encoding, maar in een binair formaat door middel van de EBCDIC encoding. (Singhal, 2023)

Dit geldt voor alle besturingssystemen die compatibel zijn op de mainframe, behalve voor Linux for System z, waar ASCII de standaard is. (IBM, z.d.-b)

4.1.1. Het hoofdbesturingssysteem z/OS

Een IBM mainframe ondersteunt meerdere besturingssystemen, maar het meest gebruikte is z/OS. Dit is, samen met de IBM Z Systems, ontwikkeld door IBM, waardoor dit het meest compatibel is met de hardware componenten. IBM blijft het ook ondersteunen. Het systeem beschikt over verschillende functies zoals workload manager (wlm) om het uitvoeren van jobs in te plannen. Dit besturingssysteem kan gezien worden als hybride omdat het moderne taken van andere besturingssystemen neemt en combineert met de architectuur van een IBM mainframe. Het heeft ook de mogelijkheid om terug te draaien naar een vorige versie zonder dat dit problemen zal veroorzaken met het systeem en zijn applicaties. (Rupp, 2022)

4.1.2. Software in z/OS

Dit besturingssysteem bevat tools zoals TSO, ISPF en SDSF. Via een 3270-terminal kan een gebruiker inloggen op de mainframe en gebruik maken van deze tools.

TSO staat voor Time Sharing Option en is in principe de command line interface op de mainframe. Hierdoor kunnen meerdere gebruikers een interactieve sessie opstarten met z/OS via hun eigen inloggegevens. Net als een traditionele CLI bestaat dit uit een prompt waarin een gebruiker commando's kan geven om acties uit te voeren op het systeem. In TSO wordt dit een "READY" prompt genoemd omdat het systeem een READY melding geeft om de gebruiker te laten weten dat het klaar is om commando's te ontvangen. (IBM, z.d.-d)

Hoewel TSO vrij krachtig is, gebruiken de meeste gebruikers het in combinatie met ISPF of Interactive System Productivity Facility. Dit is een GUI die bestaat uit menu's en panelen die allemaal verschillende functies kunnen uitvoeren (IBM, z.d.-d). Veel gebruikte functies zijn het aanmaken en schrijven van applicaties. ISPF biedt nog meer verschillende functies, zoals het aanmaken van datasets of het maken van een connectie met de database op de mainframe. Hierin kan alles gedaan worden wat het platform te bieden heeft, maar wel binnen de rechten die de gebruiker heeft. (IBM, z.d.-d)

System Display and Search Facility of SDSF is volgens de IBM (2023b) documentatie een interface om jobs en hun output te kunnen zien. Zo is er ook de mogelijkheid om een job te stoppen, bij te houden of vrij te geven. Met bijhouden wordt bedoeld niet laten uitvoeren tot iemand een teken geeft dat ze uitgevoerd mag worden. Vrijgeven wordt gebruikt om de fysieke middelen zoals de CPU vrij te geven, zodat een andere job deze kan gebruiken.

De functie biedt ook informatie over het z/OS-besturingssysteem zodat dit gemonitord, gemanaged en gecontroleerd kan worden. Hoewel deze tool vooral gebruikt wordt om de status van een uitgevoerde job te zien, wordt ze ook gebruikt om fysieke toestellen (zoals een printer) te controleren, fysieke middelen zoals CPU of geheugen te beheren en de system log en messages te bekijken.

4.1.3. Programmeertalen in z/OS

Applicaties op de mainframe worden meestal geschreven in Common Business-Oriented Language of COBOL. Dit lijkt sterk op het Engels en wordt gebruikt om bedrijf georiënteerde applicaties te ontwikkelen voor het verwerken van commerciële data. Een COBOL-programma wordt opgedeeld in 4 verschillende divisies (IBM, z.d.-a):

- Identification Division - hier wordt informatie over het programma gegeven zoals de naam;
- Environment Division - hier komt de beschrijving van onderdelen van het programma die afhangen van de omgeving waarin het programma wordt uitgevoerd;
- Data Division - hier worden de gebruikte data beschreven zoals input-/outputbestanden;
- Procedure Division - hier komt de effectieve logica van het programma.

Naast COBOL is er ook nog Programming Language 1 of PL/1. Dit is geschikt voor commerciële en wetenschappelijke applicaties. Dergelijke programma's bestaan uit "blocks", wat te vergelijken is met een groep van statements. Variabelen die in deze blocks worden gedeclareerd, zijn enkel zichtbaar in die block of in een groep van blocks. Dit maakt PL/1-programma's zeer modulair. (IBM, z.d.-a)

4.1.4. Datasets

z/OS heeft ook een andere manier om data op te slaan, met name door middel van datasets. Dit is volgens de documentatie van IBM (z.d.-d) een collectie van gerelateerde data records die opgeslagen en opgehaald wordt door een toegewezen naam. Dit is te vergelijken met een bestand in andere besturingssystemen zoals Windows of Linux.

Hier bestaan er verschillende soorten van:

- Sequentiële dataset;
- Partitionele dataset (PDS);
- Virtual Storage Access Method (VSAM).

Sequentiële datasets bevatten records die achter elkaar opgeslagen zijn. Dit heeft als nadeel dat als een gebruiker bijvoorbeeld record 20 wilt lezen, het systeem eerst de voorgaande 19 records zal lezen. Dergelijke datasets worden vooral gebruikt om grote hoeveelheden data op te slaan. (IBM, z.d.-d)

Een partitionele dataset is meer om applicaties in te schrijven. Dergelijke datasets bevatten verschillende "members" waarin de effectieve data worden opgeslagen. Dit heeft als voordeel dat de data in een willekeurige volgorde kunnen worden opgehaald. Deze vorm van een dataset wordt ook wel een library genoemd. (IBM, z.d.-d)

Een VSAM biedt een complexere manier van toegang tot verschillende soorten data en is vooral bedoeld voor applicaties. Door hun complexiteit kunnen ze niet frequent bekeken of aangepast worden in ISPF. Een VSAM valt te verdelen in 4 verschillende datasets:

- Key Sequence Data Set (KSDS):
  dit is het meest voorkomend en zal data opslaan op basis van een key, value systeem;
- Entry Sequence Data Set (ESDS):
  De records worden in een sequentiële volgorde bijgehouden en worden ook in deze volgorde gelezen. Dit wordt vooral gebruikt door databanken en z/OS Unix-bestanden;
- Relative Record Data Set (RRDS):
  Hierin worden records bijgehouden die opgehaald kunnen worden op basis van een nummer. Zo heeft een gebruiker toegang tot records die opgeslagen zijn op plaats 100 zonder eerst door de voorgaande 99 records te moeten gaan. Dit is te vergelijken met een KSDS.
- Lineair Data Set (LDS):
  Hierin worden data bijgehouden in een byte stream. Het is de enige vorm van deze soort in een traditionele z/OS file.

(IBM, z.d.-d)

4.1.5. Andere besturingssystemen

Zoals eerder vermeld is z/OS niet het enige besturingssysteem dat beschikbaar is op de mainframe. Hoewel dit door IBM aangeboden wordt, zijn er andere mogelijkheden die elk hun eigen sterktes en zwaktes hebben.

- z/Virtual Machine (z/VM)
  Dit is een type 1 hypervisor die gebruikt kan worden om meerdere besturings-

systemen in te hosten. Dit bestaat uit een control program (cp) en een conversation monitoring system (CMS). Het cp is verantwoordelijk voor het creëren van meerdere virtuele machines op basis van de fysieke hardware middelen. Het zorgt ook voor data- en applicatie-beveiliging voor alle systemen die in het systeem zitten. Het CMS zit in een eigen virtuele machine en biedt een interactieve sessie tussen de andere virtuele machines en eindgebruikers. (IBM, z.d.-b)

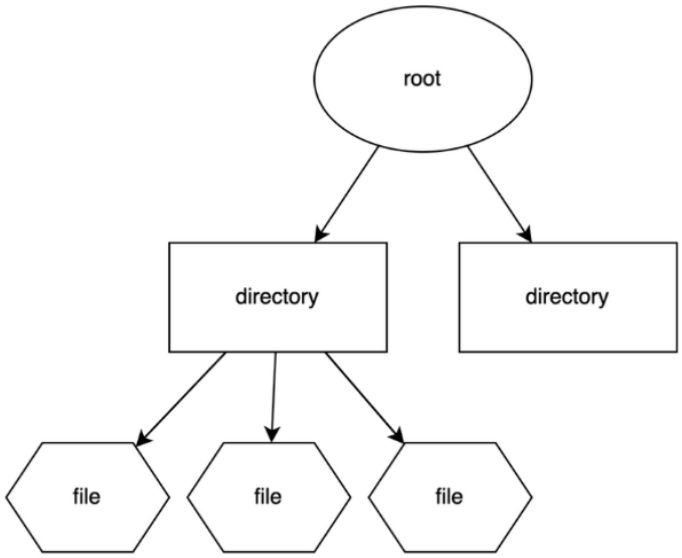
- z/Virual Storage Extended (z/VSE)
  Dit besturingssysteem is vooral nuttig voor kleinere bedrijven die geen complexe batch- of transactie-jobs moeten verwerken. Het is mogelijk dat ze, naar mate ze groeien, overgaan naar z/OS als z/VSE niet genoeg blijkt te zijn. Het design van dit besturingssysteem maakt het perfect om meer routineuze batch jobs parallel uit te voeren. Meestal wordt z/VM ook gebruikt als een terminal interface voor development en systeembeheer in z/VSE. (IBM, z.d.-b)

- Linux for System z
  Dit is, zoals de naam zegt, Linux op de mainframe. Er zijn hier 2 verschillende distributies voor: Linux for S/390 (gebruikt 31-bit adressering) en Linux for System z (gebruikt 64-bit adressering).
  Linux for System Z verwijst naar Linux for S/390. Dit besturingssysteem maakt geen gebruik van een 3270-terminal maar van X-Windows based terminals. Dit bestaat uit een cli waarmee er een telnet- of ssh-connectie gemaakt wordt met het Linux for System z besturingssysteem. Dit is ook volledig in de encoding ASCII en niet EBCDIC. (IBM, z.d.-b)

- z/Transaction Processing Facility (z/TPF)
  Dit besturingssysteem is vooral nuttig voor bedrijven die veel transacties moeten uitvoeren, zoals een bank of een luchvaartmaatschappij. Het systeem kan gebruik maken van verschillende mainframes om tienduizenden transacties per seconde uit te voeren zonder enige onderbreking. (IBM, z.d.-b)

4.1.6. Unix op een mainframe

IBM heeft veel ingezet op modernisatie. Zo is er een Unix System Services (USS) omgeving bijgekomen die geïntegreerd is in het traditionele besturingssysteem z/OS. Er is ook een volledige mainframe die enkel Linux heeft als besturingssysteem namelijk de LinuxOne. In dit onderzoek wordt er gebruik gemaakt van een mainframe met z/OS die een USS-omgeving bevat. Een LinuxOne zal dus niet besproken worden.



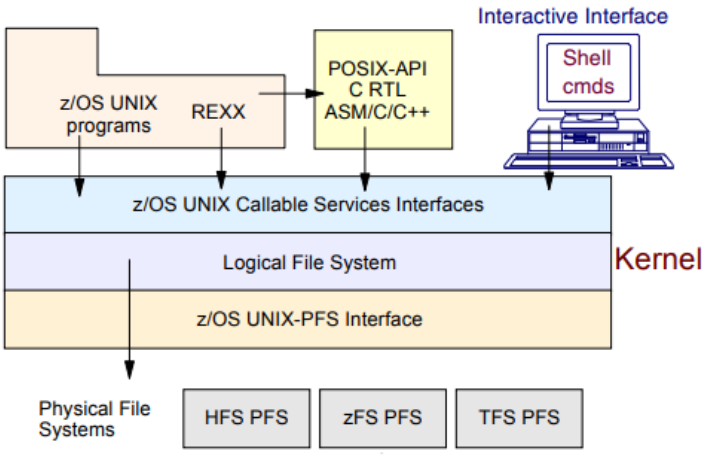
Figuur (4.1)
HFS voorbeeld (Codecadamy, 2022)

Omdat USS samenwerkt met z/OS, zijn er veel meer functies die gebruikt kunnen worden op de mainframe. Zo is er de mogelijkheid op XML parsing, OpenSSH, de IBM HTTP Server for z/OS, de z/OS SDK for Java en nog veel meer. (Dhawan, 2013)

Dit systeem biedt een hiërarchisch bestandssysteem (HFS) samen met een zSeries bestandssysteem (zFS). (Precisely, 2020)
De HFS is wel bekend bij de meeste UNIX-gebruikers: dit is een hiërarchie van directories die bestanden of andere directories bevat. Dit kan grafisch weergegeven worden in een tree view (HCL Technologies, 2022).

Zoals afgebeeld in figuur 4.1, is er een root (in Unix afgebeeld als "/") die directories bevat. Deze directories bevatten bestanden en andere directories. Ze kunnen ook andere informatie bevatten over deze bestanden. Directories die opgeslagen zijn in een andere directory, worden subdirectories genoemd. Vaak wordt er ook gebruik gemaakt van een parent-child-verwijzing, waar de parent directory een level boven de child of subdirectory zit. (Codecadamy, 2022)

zFS is iets minder bekend en kan gebruikt worden in plaats van of als toevoeging op het traditionele HFS. Dit heeft vooral zijn waarde door zijn sterke performantie in bestanden die vaak worden gebruikt. Het vermindert ook het risico op verlies in updates omdat het data asynchroon schrijft in plaats van te wachten op een sync interval. Bestanden in dit systeem kunnen aangepast worden door middel van een Application Programming Interface (API) en kunnen zelf in de HFS worden toegevoegd zonder enige problemen. (IBM, 2012)



Figuur (4.2)
zFS werking (IBM, 2012)

In figuur 4.2 worden de bestanden opgehaald vanuit het zFS-bestandsysteem door middel van de interactieve interface. Dankzij de API's tussen het logische- en fysieke bestandsysteem worden de juiste bestanden opgehaald en teruggegeven aan de gebruiker. (IBM, 2012)

Unix-bestanden op de mainframe worden bijna op dezelfde manier gebruikt als op een traditioneel Unix-systeem. Het kan een Java-, C++- of Python-applicatie bevatten. Deze programma's kunnen ook bestanden lezen of schrijven in een JSON- of YAML-formaat. Die kunnen op hun beurt dan gebruikt worden om analyses te doen op bepaalde data. Het hangt allemaal af van de use case om te zien op welke manier deze Unix-bestanden het best worden gebruikt. (Precisely, 2020)

Deze omgeving werkt ook met omgevingsvariabelen die invloed hebben op hoe het systeem werkt. Veel van deze variablen worden opgezet bij het inloggen, maar de gebruiker kan deze zelf aanpassen. Ze kunnen verwijzen naar de directory waarin de gebruiker zich bevindt, maar zijn ook nodig voor bepaalde packages om hun installatiepad te vinden. (Henry-Stocker, 2017)

4.2. Python in z/OS

Aangezien er een Python-interpreter zal worden opgezet, ligt de nadruk hier vooral op Python in z/OS. Dit is momenteel een van de meest gebruikte programmeertalen door zijn verschillende doeleinden en gebruiksvriendelijkheid (Johnson, 2023). Ook IBM heeft dit opgemerkt en besloot om dit compatibel te maken met de mainframe. De meest gebruikte doeleinden zijn voor data-analyse -en -visualisatie, machine learning en software development. Voor de laatste spelen de flexibiliteit en de kracht van de programmeertaal een grote rol omdat er zowel heel eenvoudige als heel complexe applicaties kunnen worden geschreven. In software develop-

ment worden vaak API's ontwikkeld met verschillende packages. (Kosourova, 2022)

4.2.1. Python in USS

Naast programmeren heeft dit ook de mogelijkheid om Python-commando's uit te voeren in de terminal van USS. Met het Python commando kan er bijvoorbeeld een virtuele omgeving worden aangemaakt. Dit is een directory met een bepaalde bestandsstructuur die een bin subdirectory bevat die verwijst naar de Python-interpreter en verschillende packages die in deze omgeving geïnstalleerd zijn. (Princeton University, 2022)
Het is gebouwd op een bestaande Python-installatie en gebruikt tools zoals pip om packages te installeren (Python Software Foundation, 2024).

Een Python virtuele omgeving zorgt voor een stabiele en herproduceerbare omgeving waar de gebruiker de volledige controle heeft over welke packages er geïnstalleerd zijn. Hierdoor heeft de gebruiker geen problemen met administratierechten die voorkomen als er niet in deze omgeving gewerkt wordt. Er is geen limiet op het aantal virtuele omgevingen dat een gebruiker kan opstellen en deze zijn gemaakt om eenvoudig te worden geactiveerd en gedeactiveerd. (Princeton University, 2022)

Bij het aanmaken zijn er verschillende opties mogelijk. De belangrijkste voor dit onderzoek is de *–system-site-packages* optie. Hiermee kan men in de gemaakte virtuele omgeving packages gebruiken die al geïnstalleerd zijn op het systeem. (Python Software Foundation, 2024).
Het aanmaken van een Python virtuele omgeving met de packages van het systeem zou als volgt gebeuren:

```
$ python -m venv /Path/To/venv --system-site-packages
```

In de meegegeven directory worden subdirectories aangemaakt die nodig zijn om de virtuele omgeving te onderhouden. Om het te activeren, moet het *activate* script in de *bin* subdirectory uitgevoerd worden. (Python Software Foundation, 2024)
Afhankelijk van het systeem, wordt een verschillend *activate* script gebruikt. In bash heet dit "activate" en het wordt als volgt uitgevoerd:

```
$ source ./bin/activate
```

Om deze omgeving af te sluiten, wordt volgend commando ingegeven:

```
$ deactivate
```

Voor het activeren moet er specifiek verwezen worden naar het *activate* script in de bin subdirectory. Het deactiveren kan in eender welke directory.

4.2.2. Packages in Python

In Python wordt er vaak gebruik gemaakt van externe packages en modules die gebruikt kunnen worden door programmeurs tijdens het schrijven van een applicatie.
Een module is een python bestand wat bestaat uit definities en statements. Dit kan bestaan uit variabelen en functies die gerelateerd zijn. Via het "import" statement, kunnen de functies in een module gebruikt worden in een extern Python applicatie (Singh, 2024). Dit zorgt ervoor dat programmeurs niet zelf de code moeten schrijven waardoor ze veel tijd kunnen winnen. Dit zorgt er ook voor dat repetitieve code niet elke keer opnieuw geschreven moet worden.

Een packages is een directory waar verschillende modules in opgeslagen worden en bevatten in sommige gevallen subdirectories. Omdat alles in een directory zit, zijn de modules met elkaar verbonden via de naam van de package. Dit bevat een "init.py" bestand wat het duidelijk maakt voor Python dat dit een package is. Dit bestand kan leeg zijn of code bevat die uitgevoerd wordt als de package geïnitialiseerd wordt. (Udacity, 2021)

Een voorbeeld van een package is FastAPI: dit kan gebruikt worden om API's te ontwikkelen. Tijdens het ontwikkelen, wordt er automatisch een interactieve documentatie opgesteld zodat de gebruiker dit niet hoeft te doen. Dit maakt gebruik van een andere package genaamd "pydantic" en dit zorgt ervoor dat de declaraties van datatypes hetzelfde blijft als in Python. Een andere package genaamd "starlette" zorgt voor sterke performantie, startup en shutdown events en support voor achtergrond taken tijdens het uitvoeren. (FastAPI, z.d.)

Om een API uit te voeren, kan er gebruik worden gemaakt van de "Uvicorn" package. Dit is een web server wat inkomende verzoeken van gebruikers ontvangt. Dit werkt samen met FastAPI om de inkomende verzoeken correct te verwerken. Dit zorgt ervoor dat Uvicorn focust op de netwerk connectie en inkomende verzoeken, en FastAPI op het verwerken van deze verzoeken. (Sentry, 2024)

4.2.3. Python AI toolkit for z/OS

Om gebruik te maken van deze packages met modules, moeten deze geïnstalleerd zijn op het systeem, wat een probleem kan zijn in USS. Door de beveiliging op de mainframe is het niet simpel om packages direct te installeren in z/OS (IBM, 2021a). Daarom heeft IBM een Python AI toolkit for z/OS opgezet.

In de aankondiging door Evan Rivera (2023) werd deze toolkit beschreven als een bekende en flexibele ervaring voor developers tijdens het ontwerpen van AI-oplossingen. De toolkit biedt open-source software aan die eenvoudig kan worden geïnstalleerd

door de Package Installer for Python (pip).

De Python AI toolkit is dus een bibliotheek met verschillende, wereldwijd gebruikte Python packages voor AI en Machine learning workloads, bijvoorbeeld NumPy, SciPy, Jupyter, enz. Elk van deze packages is volledig onderzocht op potentiële problemen in het systeem, waardoor alles in deze toolkit voldoet aan de beveiligingsvoorwaarden zoals andere z/OS producten. (Bostian & Rivera, 2023)

4.2.4. Z Open Automation Utilities

Python alleen op de mainframe is niet voldoende. Vaak willen developers datasets in z/OS kunnen lezen of schrijven, en Python biedt hiervoor geen functies aan. IBM is dan gekomen met de Z Open Automation Utilities (ZOAU). Dit bevat functies om acties uit te voeren op datasets in z/OS of op het systeem zelf. Dit is ontworpen door IBM om het zo natuurlijk mogelijk te laten aanvoelen voor developers die weinig kennis hebben over z/OS. Zo lijken de functies erg sterk op hun overeenkomstige Unix-commando's. Een commando om een lijst van datasets weer te geven bijvoorbeeld is *dls*. Dit is geïnspireerd op het *ls* Unix-commando om de inhoud van een bepaalde directory te zien. ZOAU bevat ook een *zopen*-statement om datasets te openen om te lezen of schrijven. In Python is dit het *open*-statement. (IBM, 2023a)

4.3. IBM Developer for z/OS

4.3.1. IDz

De omgeving waarin er gewerkt wordt is IBM Developer for z/OS (IDz) versie 16.0.2. Dit programma is volgens de definitie van Spohn (2023) een toolset voor het ontwikkelen en opzetten van hybride cloud applicaties op z/OS.
Het is een Eclipse based programma met de mogelijkheid om een connectie te maken met verschillende omgevingen op de mainframe, bijvoorbeeld de USS- of z/OS-omgeving. Dit biedt de mogelijkheid om alle bestanden te zien, te openen en aan te passen. Omdat de data nogsteeds op de mainframe staan, kunnen wijzigingen direct gezien worden in andere programma's. Als een gebruiker bijvoorbeeld een bestand wijzigt via IDz, kunnen deze wijzigingen direct te zien zijn in ISPF.
Dit programma is vooral ontwikkeld om een gekende werkomgeving te bieden om in te programmeren, aangezien niet iedereen bekend is met ISPF.

Zoals vermeld wordt er gebruik gemaakt van IDz versie 16.0.2. In het overzicht van IBM (2024b) ondersteunt deze versie syntax veranderingen voor COBOL 6.4, PL/1 6.1 en REXX. Er is ook een ZUnit-update wat vooral het gebruik van deze tool makkelijker maakt.

ZUnit staat voor z/OS Automated Unit Testing en is een framework dat gebruikt

wordt in IDz om COBOL- en PL/1-programma's te testen. Hiervoor maakt het gebruik van verschillende "samples" die door IBM ontworpen zijn (bv. Enterprise COBOL CALL02.cbl sample test case). Het gebruik van deze tool maakt het makkelijker voor developers om hun code (geschreven in COBOL of Pl/1) te testen op de mainframe. Dit maakt het schrijven van code in IDz veel efficiënter. (IBM, 2024a)

Het is ook mogelijk om bestanden van de lokale machine te kopiëren naar de mainframe in de USS-omgeving. Belangrijk hierbij is de manier waarop het bestand wordt gekopieerd. In IDz heet dit "File Transfer Mode", met 2 opties:

- Text
- Binary

Afhankelijk van het bestand wordt text of binary gekozen en het wordt ingesteld op basis van de extensie van het bestand.

4.3.2. Eclipse IDE

IDz is gebaseerd op de Eclipse Integrated Development Environment (IDE) die in 2001 is ontwikkeld door de Eclipse Foundation. Alles begon toen IBM 3 miljoen lijnen code van hun Java tools doneerde om een open source IDE te creëren. De omgeving is dus Java gebaseerd, maar kan ook gebruikt worden voor andere programmeertalen zoals Python door de verschillende plug-ins die beschikbaar zijn gemaakt door de community achter Eclipse. (Hanna, 2021)

Plug-ins zijn toevoegingen aan de gebruikte software die het mogelijk maken om applicaties, computer-programma's en web browsers aan te passen. Dit zijn ook add-ons die extra functionaliteiten kunnen bieden aan het gebruikte programma. (George, 2021)

4.3.3. Pydev

Pydev is een voorbeeld van een plug-in die opgezet kan worden in Eclipse om een Python IDE te gebruiken. Dit biedt functies zoals code completion, code analysis, debugging, code coverage en nog meer. De meest recente versie is 12.0.0, die werd uitgebracht op 1 februari 2024. Ze bevatte updates voor de debugger en code analyse. Ze is enkel beschikbaar voor Python 3.8 en hoger, waardoor een oudere versie van Pydev nodig is als een gebruiker een lagere Python versie gebruikt. Pydev is volledig open source en de verdere ontwikkeling hangt af van externe sponsors. (Pydev, 2024)

4.3.4. Wizards

Een programma zoals IDz heeft veel mogelijkheden om instellingen aan te passen door middel van wizards. Dit is een serie van pagina's die de gebruiker begeleidt

om een complexe taak uit te voeren. Elke pagina vraagt wat informatie en als de gebruiker op 'finish' klikt, wordt de taak uitgevoerd. Er is ook altijd een mogelijkheid om het proces stop te zetten. (Eclipse, 2006)

4.4. IBM Z in een bankomgeving

Dit onderzoek wordt uitgevoerd in een bankomgeving, dus is het interessant om na te gaan welke voordelen een mainframe in deze omgeving te bieden heeft.

In een aankondiging van IBM (2022) over hun nieuw systeem de IBM z16, werd er gezegd dat 70% van wereldwijde transacties door een mainframe verwerkt wordt.

Volgens Turner (2022) gebruiken de meeste banken een IBM-mainframe omdat deze de rekenkracht kan bieden die banken nodig hebben om efficiënt te kunnen werken. Kenmerken zoals robuustheid, betrouwbaarheid en snelle verwerkingskracht spelen ook een grote rol aangezien het van groot belang is dat het systeem bijna altijd actief moet zijn. De mainframe toont hier zijn sterkte door de "8 nines" toe te passen. Dit betekent dat een mainframe 99,999999 % van de tijd beschikbaar moet zijn per jaar. (IBM, z.d.-c)

Dit is niet de enige reden waarom de mainframe gebruikt wordt door 92 van de top 100 banken ter wereld (Tozzi, 2022). Ook beveiliging speelt een grote rol. Uit een studie van Morning Consult en IBM (2022) blijkt dat krediet-kaart-fraude het meest voorkomende type fraude is bij klanten in 7 verschillende landen, waaronder Duitsland, de Verenigde Staten en China. De IBM zSystems heeft al een goede transactie beveiliging zonder teveel vertraging, maar de z16 voegt hier nog een AI-model aan toe. Hierdoor kunnen banken transacties controleren op een veel grotere schaal: 300 miljard door AI beveiligde transacties per dag met maar 1 milliseconde vertraging. Andere zaken zoals belastingfraude kunnen hiermee ook worden vermeden. (IBM, 2022)

IBM doet dit door middel van een nieuwe Telum Processor. Deze is ontworpen om deep learning-algoritmes naar workloads te brengen voor bedrijven om in real-time fraude te detecteren en te verwerpen. Dit is de eerste processor van IBM die gebruik maakt van AI terwijl de transactie wordt uitgevoerd. Naast het detecteren van fraude is deze chip nog nuttig voor loan processing, antiwitwaspraktijken en risico analyse. (IBM, 2021b)

4.4.1. Batch & online jobs

Een ander belangrijk voordeel dat een mainframe te bieden heeft, is batch processen. Dit zijn jobs die kunnen worden uitgevoerd met weinig tot geen interactie van

een gebruiker (IBM, z.d.-d). Dit is vooral nuttig voor taken die op vaste momenten uitgevoerd moeten worden.
Figuur 4.3 bevat een duidelijk voorbeeld van de werkwijze van een batch job:

1 De batch job wordt opgestart door de mainframe.

2 De job genereert statistieken van het bedrijf.

3 Er wordt een back-up gemaakt van de kritische bestanden en de database voor en na de job.

4 De statistieken worden verstuurd naar een specifieke plaats voor analyse.

5 Fouten in de statistieken worden in een andere locatie geplaatst.

6 Informatie van de bankrekening van klanten wordt aangemaakt en verstuurd.

7 Een verslag van de verwerkingstijd wordt verstuurd naar de partners van de bank

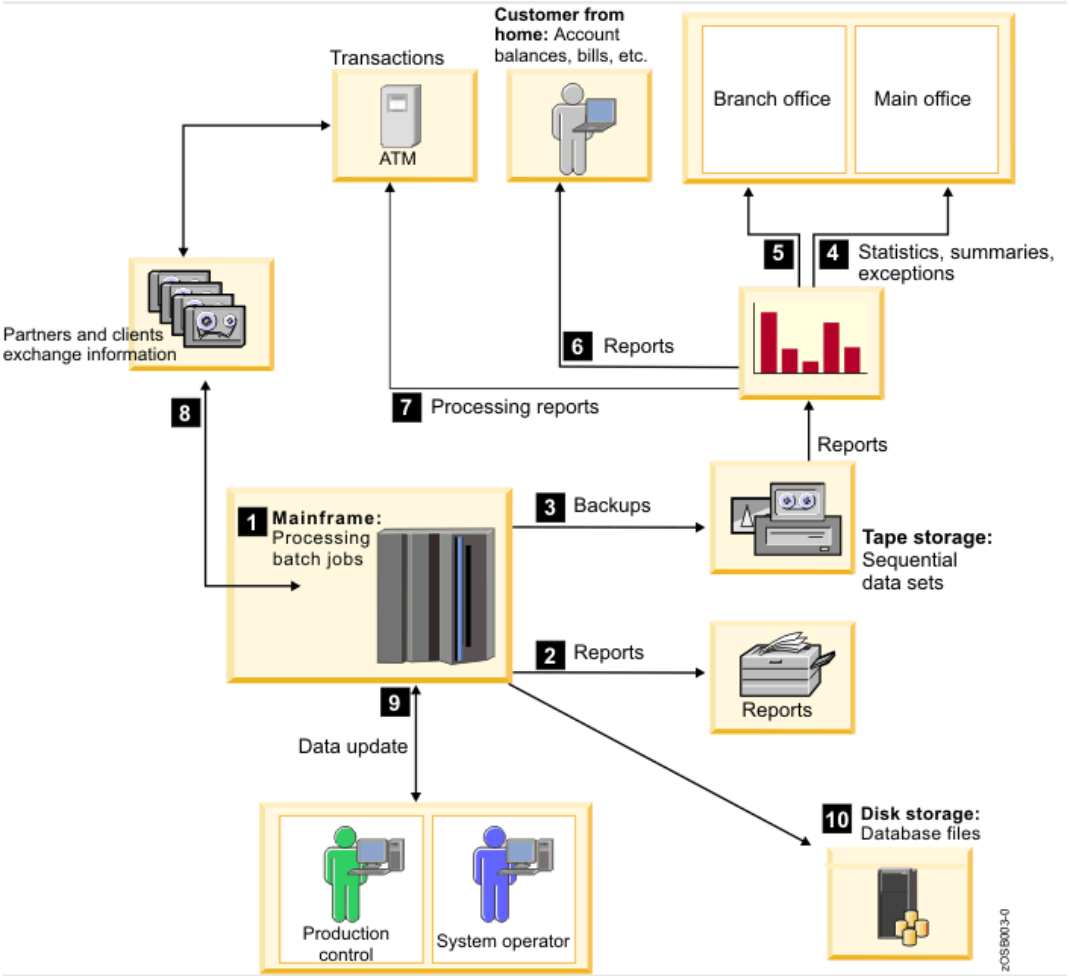
8 De partner ontvangt het verslag.

9 Het systeem monitort de uitkomst van de batch job.

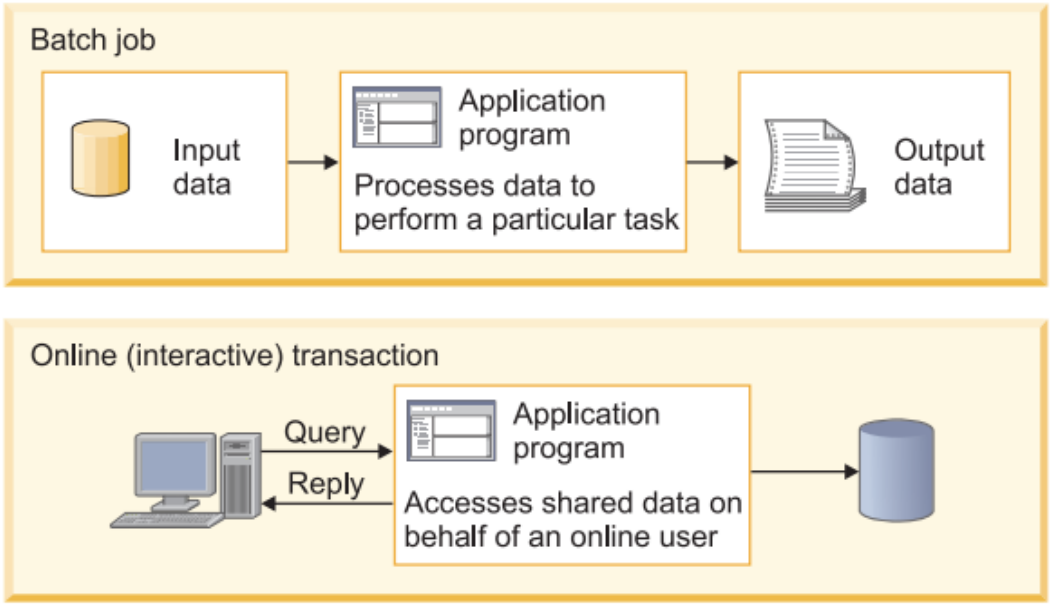
10 Jobs en transacties updaten de database zodat de progressie wordt opgeslagen.

Batch jobs zijn niet de enige vorm van jobs die op een mainframe worden uitgevoerd. Online transaction processing of OLTP is zeer belangrijk in een bank. In tegenstelling tot batch processing, is hierbij een eindgebruiker nodig die de job opstart. OLTP heeft ook geen vast moment waarop de job uitgevoerd wordt en kan dus op elk moment van de dag gebeuren. Meestal duren deze transacties niet lang, maar de systemen die verantwoordelijk zijn om deze jobs uit te voeren, moeten veel verschillende gebruikers op hetzelfde moment ondersteunen zonder enige vertraging. (IBM, z.d.-b)

Een grafische voorstelling van de verschillen tussen een batch job en een online transaction job is te zien in figuur 4.4. Een batch job heeft input die verwerkt wordt door een programma op een bepaald, vast moment. De uitkomst wordt dan geschreven in een bestand om analyses op uit te voeren.
Een online transactie daarentegen wacht op een query van een gebruiker om te starten. Data wordt dan opgehaald uit het systeem en teruggegeven aan de gebruiker. (IBM, z.d.-b)



Figuur (4.3)
Batch Job (IBM, z.d.-b)



Figuur (4.4)
Grafisch verschil tussen batch en OLTP (IBM, z.d.-b)

4.4.2. Batch jobs uitvoeren

Batch jobs worden uitgevoerd door middel van Job Control Language (JCL). Dit zijn aparte programma's die de middelen definiëren die een batch job nodig heeft bijvoorbeeld een input -of output bestand. Een JCL-programma bestaat uit 3 statements:

- JOB - hier wordt de naam van het programma bepaald (jobname). Er kunnen parameters toegevoegd worden die gelden voor de hele job;
- EXEC - dit bepaald het programma dat uitgevoerd moet worden. Een JCL kan meerdere EXEC-statements bevatten waarbij elke statement een *job step* wordt genoemd;
- DD - dit bepaalt de input- en/of output-bestanden die de batch job nodig heeft.

Hoewel dit niet veel functies zijn, bestaan er veel verschillende parameters die meegegeven kunnen worden en die de werking van het programma zullen beïnvloeden. Dit maakt JCL moeilijk om te schrijven. Dit kan zowel in TSO als in ISPF uitgevoerd worden. (IBM, z.d.-d)

In DNB wordt de JCL opgesplitst in 2 programma's:

1. Job card - bevat de JOB statement;
2. Procedure - bevat de EXEC- en DD-statements.

Voor het uitvoeren wordt er enkel verwezen naar de Job card en die roept op zijn beurt de procedure op.

4.5. Samenvatting

Omdat het hoofdbesturingssysteem z/OS van een mainframe niet gekend is door velen en gebruik maakt van verschillende programmeertalen en manieren van data opslaan, heeft IBM ervoor gezorgd dat er een Unix-systeem aanwezig is, genaamd de USS. Hierdoor kunnen gebruikers die niet bekend zijn met z/OS toch nog gebruik maken van dit systeem. Dit biedt dankzij IBM verschillende tools om packages te installeren en om interactie te hebben met datasets via de Python AI toolkit for z/OS en de Z Open Automation Utilities (ZOAU). Beide zijn beschikbaar voor Python en Java.

Via IDz kunnen gebruikers bestanden in de USS openen en aanpassen. Via een externe ssh-connectie kunnen deze getest worden. IDz is gebaseerd op Java waardoor er een Java-interpreter aanwezig is. Dit is niet het geval voor Python en heeft als gevolg dat er geen Python-interpreter aanwezig is. In IDz kan er wel gebruik worden gemaakt van verschillende plug-ins die dit mogelijk kunnen maken.

Hoewel de USS-omgeving het eenvoudiger maakt om applicaties te schrijven, is het (voorlopig) niet de bedoeling dat de applicaties die in COBOL of PL/1 geschreven zijn, vervangen worden. Deze programmeertalen zijn zeer belangrijk om batch- en online-jobs uit te voeren op de mainframe en zijn van cruciaal belang voor bedrijven zoals banken om hun infrastructuur staande te houden en de juiste services aan de klanten te kunnen bieden.

De plug-in Pydev speelt een belangrijke rol aangezien deze geïmplementeerd wordt in deze bachelorproef.

Tot slot werden batch- en OLTP-jobs besproken om zo de hoofdfunctie van een mainframe in een bankomgeving duidelijk te maken.

4.6. Conclusie

Deze literatuurstudie heeft een overzicht gegeven van de mainframes van IBM om duidelijkheid te scheppen over de omgeving waarin er gewerkt wordt. Dit ging over het hoofdbesturingssysteem z/OS en zijn verschillende onderdelen zoals TSO, ISPF, datasets en programmeertalen. Hoe IBM moderniseert door middel van de USS omgeving speelt ook een grote rol. Verschillende tools die gebruikt worden in deze modernisatie, zoals IDz, de Python AI toolkit en ZOAU, zijn zeer belangrijk om het gebruik van Python en Java zo efficiënt mogelijk te laten gebeuren.

Op basis van deze analyse kan er geconcludeerd worden dat Pydev een goede functie kan zijn in IDz aangezien er geen Python-interpreter aanwezig is in dit programma. Om het schrijven van Python applicaties nog efficiënter te maken, is dit zeer belangrijk. Aangezien IDz Eclipse based is, zou het geen complexe opgave moeten zijn om dit te implementeren. IDz heeft ook de mogelijkheid om een connectie te maken met de USS-omgeving, dus is er mogelijk een manier om via IDz Python-applicaties op de mainframe uit te voeren.

Python en Java zullen meer gebruikt worden door bedrijven omdat ze vrezen voor een sterk werknemerstekort in de toekomst. Hoewel dit onderzoek het eenvoudiger maakt om Python-applicaties te schrijven, is het interessant om te onderzoeken of batch- en OLTP-jobs die geschreven zijn in COBOL en PL/1, volledig vervangen kunnen worden door applicaties in de USS-omgeving die geschreven zijn in Python en Java.

5

Pydev opzetten

Het programma waarin het onderzoek wordt uitgevoerd, is IBM Developer for z/OS (IDz) versie 16.0.2. IDz is gebaseerd op Eclipse versie 4.23.0 .
In dit onderzoek wordt versie 8.2.0 van PyDev geïnstalleerd. Dit is een oudere versie, maar ze is het best compatibel met IDz; de andere versies brengen problemen met zich mee tijdens en na de installatie. Versie 8.2.0 bevat onder andere een debugger, een code formatter en code analyse. Het is belangrijk om te weten dat IDz door IBM beschikbaar wordt gesteld en dat IBM dus support zal bieden waar nodig mochten er problemen zijn met hun programma. Dit is niet het geval met PyDev omdat dit door een externe partij is ontwikkeld en dus niet ondersteund wordt door IBM.

5.1. Vereisten voor installatie

- Eclipse versie 4.6 based programma nodig - IDz;
- Java 11 of hoger;
- Python 2.6 of hoger.

5.2. Installatie

Pydev kan geïnstalleerd worden via de Github repository https://github.com/fabioz/Pydev/releases.
Hierdoor wordt een .zip geïnstalleerd die uitgepakt moet worden in een directory naar keuze. Het uitpakken kan enige tijd duren.

Navigeer in IDz naar *help -> install New Software*

Hier komt er een installatie wizard waar Pydev geïnstalleerd kan worden in IDz via *add -> local*.

Selecteer de map waarin Pydev is uitgepakt. Klik vervolgens op *add*

Er verschijnt een scherm met de installatie opties. Als dit leeg is, kan dit veranderd worden door de checkbox “Group items by category” uit te vinken. Hierna komen 3 opties tevoorschijn:

- PyDev for Eclipse 8.2.0.202102211157;
- PyDev for Eclipse Developer Resources 8.2.0.202102211157;
- PyDev Mylyn Integration 0.6.0.

De eerste is noodzakelijk en de tweede is optioneel. De derde mag niet aangevinkt worden aangezien MyLyn niet geïnstalleerd is in IDz, waardoor het opzetten van PyDev zal mislukken.

Klik op *next*, dit zal alles opzetten en kan even duren.

Wanneer dit gedaan is, wordt er een overzicht gegeven van de installatie details. Klik op *finish* om de installatie te beginnen.

In de balk rechtsonder wordt de status van de installatie weergegeven.

Wanneer de installatie voltooid is, geeft IDz een pop-up om het programma opnieuw op te starten. Als er nog niet opgeslagen projecten open staan, worden deze best opgeslagen voor het vóór het heropstarten. Klik in het andere geval op *Restart Now*

Navigeer om te zien of de installatie gelukt is naar *help -> about -> Installation details*.

Geef in de zoekbalk “Pydev” in. Hier zou de geïnstalleerde software moeten staan.

Als er een Python script geopend wordt, zal dit nogsteeds gebeuren in de interne text editor. Om dit te wijzigen, moeten de file associations aangepast worden. Deze zullen bepalen welke editor er gebruikt wordt bij een bepaalde extensie. Hier wordt de Python editor van Pydev gelinkt aan de .py-extensie.

Ga naar *Window -> Preferences -> General -> Editors -> File Associations*

Hier komt een lijst met allemaal extensies en het programma waarmee ze geopend worden. IDz heeft niet standaard een python-editor, dus deze moet toegevoegd worden. Klik bij “file types” op *add*.

Geef hier “.py” in en klik op *ok*.

In de lijst van extensies staat nu .py.

Selecteer de .py-extensie en bij “associated editors” komen er 3 opties:

- Python Editor;
- Text Editor;
- Generic Text Editor.

Selecteer “Python Editor” en klik op *Default*.

Als een Python-applicatie geopend wordt, zal dit automatisch gebeuren in de Python-editor en zijn alle functies van Pydev beschikbaar.

6

Runtime environment opzetten

6.1. Python-applicaties uitvoeren op een lokale machine via IDz

In IDz is er de mogelijkheid om programma's uit te voeren op de lokale machine waarop IDz uitgevoerd wordt. Hiervoor moet Python geïnstalleerd zijn op deze machine en in IDz moet ernaar worden verwezen.

DNB heeft strenge regels voor het installeren van externe software op laptops die een connectie kunnen maken met de mainframe. Door deze veiligheidsredenen is er geen mogelijkheid om Python te installeren en kan deze stap ook niet uitgevoerd worden.

Als dit mogelijk is, kan er naar de Python executable verwezen worden in IDz via *Window -> preferences -> Pydev -> Interpreters -> Python Interpreter*.
Door te klikken op *New...*, verschijnt er een wizard om de interpreter te kiezen. Zoek de geïnstalleerde Python-interpreter en klik op *Apply*.

Door rechts te klikken op een Python-bestand en naar *Run* te navigeren, kan het programma uitgevoerd worden.
Vermoedelijk zal dit het Python-applicatie uitvoeren op de gebruikte machine en niet in de USS-omgeving op de mainframe.

6.2. Python-applicaties uitvoeren in de USS omgeving via IDz

De Python-interpreter is niet nodig op de lokale machine voor het uitvoeren van Python programma's in de USS-omgeving. Hiervoor moet het remote systems ex-

plorer perspectief geopend worden via *Window -> Perspective -> Open Perspective -> Other -> Remote System Explorer*.

Maak een connectie met de USS-omgeving met de juiste inloggegevens en navigeer naar het tabblad *Remote Shell*.

Dit zal leeg zijn, maar door te klikken op de 3 puntjes met als naam *View Menu* kan de connectie toegevoegd worden. Klik op *Launch* en kies de juiste connectie. Dit zou 2 connecties moeten tonen. De eerste is de connectie met de lokale machine en de tweede is die met de USS-omgeving. Selecteer de connectie met de USS omgeving, dit zal 2 keuzes geven:

1 z/OS UNIX Shells;

2 TSO Commands.

Om connectie te maken met de USS-omgeving, wordt optie 1 gekozen: *z/OS UNIX Shells*. De tweede optie is om TSO-commando's te geven in de z/OS-omgeving waarmee een connectie is gemaakt.

Hiermee wordt een terminal opgestart waar er commando's ingegeven kunnen worden in de USS-omgeving. Hierdoor kan er genavigeerd worden naar verschillende directories met Python bestanden. Deze kunnen worden uitgevoerd met het Python-commando. Dit heeft ook als functie om directories of bestanden aan te maken in deze terminal zonder een externe ssh-connectie nodig is.

7

Runtime environment testen

In dit hoofdstuk wordt er getest hoe krachtig de ingebouwde terminal van IDz is en wat zijn beperkingen zijn. Dit gebeurt vóór hoofdstuk 8 omdat in dit hoofdstuk de omgeving wordt opgezet waarin de API zal worden uitgevoerd.
Er wordt verwacht dat Python correct geïnstalleerd is in de USS-omgeving.

7.1. Testomgeving opzetten

Alle bestanden die worden geschreven om te testen, zullen worden aangemaakt in de directory */tmp/BPtestfolder*. Deze wordt aangemaakt via de Bash-commando's:

```
$ mkdir /tmp/BPtestfolder
$ cd /tmp/BPtestfolder
```

7.2. Test script aanmaken en uitvoeren

Om te testen of Python correct is geconfigureerd, is het best om een simpele applicatie te schrijven in deze programmeertaal. Door deze test wordt het ook duidelijk of er effectief applicaties uitgevoerd kunnen worden via IDz in USS. De test applicatie wordt als volgt aangemaakt:

```
$ touch test.py
$ echo `print('Hello World')` >> test.py
$ python test.py
```

In de pas aangemaakte directory wordt het bestand "test.py" aangemaakt, dat "Hello World" op het scherm weergeeft. Aangezien dit geen groot programma is, wordt dit met het *echo*-commando geschreven in het Python-bestand. Tijdens het uivoeren geeft dit programma "Hello World" terug, wat aangeeft dat het perfect werkt.

7.3. Python virtuele omgeving opzetten

Een Python virtuele omgeving wordt vooral gebruikt om een eigen omgeving op te zetten waar de gebruiker toegang heeft tot alle functies, zoals bijvoorbeeld packages installeren. Om de API uit te voeren, zijn er verschillende packages nodig, dus wordt er gebruik gemaakt van een dergelijke virtuele omgeving. Deze wordt opgezet en geactiveerd in IDz. Als dit lukt, zullen de nodige packages, die gebruikt worden in de test API, worden geïnstalleerd.

Om de virtuele omgeving aan te maken, wordt het Python "venv"-commando gebruikt. Deze omgeving wordt geactiveerd door middel van het "activate"-script in de *bin* subdirectory. Deze 2 stappen gebeuren als volgt:

```
$ python -m venv --system-site-packages ./virtualEnv
$ cd virtualEnv/bin
$ activate
$ pip list
```

Hoewel de virtuele omgeving aangemaakt wordt, kan deze niet opgestart worden met het source commando, wat wel gaat in een ssh-connectie. In IDz moet er genavigeerd worden naar de *bin* subdirectory om hierin het *activate* programma uit te voeren. Om met de virtuele omgeving te werken, moeten alle commando's in de *bin* subdirectory gebeuren anders zal het niet lukken. Het commando *pip list* wordt gebruikt om te testen of de virtuele omgeving actief is. Als het niet actief is, zal dit commando niet lukken. Het geeft een lijst van geïnstalleerde packages in de omgeving. Voorlopig zijn dit de packages die al op het systeem stonden. Bij het aanmaken van de virtuele omgeving werden deze overgenomen door middel van de optie *–system-site-packages*.

7.4. Packages installeren

Zoals eerder vermeld zijn er nog packages nodig waar Python-programma's gebruik van kunnen maken. Deze zullen geïnstalleerd worden van de Python AI toolkit for z/OS of van de Pypi website. De packages die geïnstalleerd zullen worden zijn nodig voor het testprogramma dat wordt geschreven in hoofdstuk 8. Deze packages zijn *FastApi*, *Uvicorn* en *Jsonschema*.

FastAPI en Uvicorn maken geen onderdeel uit van de Python AI toolkit en moeten geïnstalleerd worden van https://pypi.org/. Het is belangrijk om alle packages waarvan FastAPI gebruik maakt, ook te installeren. Dit installatiebestand heeft als extensie ".whl" en via drag and drop is het mogelijk om deze in de USS-omgeving te plaatsen vanaf de lokale machine. Dit moet in File Transfer Mode "binary" gebeuren omdat het anders niet correct geformatteerd is. Dit kan worden ingesteld in IDz via *Window -> Preferences -> Remote Systems -> Files*. Hier wordt een lijst gegeven

van extensies en de wijze waarop ze worden overgezet naar de USS-omgeving. Om een .whl-bestand correct over te zetten naar de mainframe, wordt *Add...* geselecteerd. Hier is het mogelijk om een extensie van een bestand in te geven, en in dit geval is dit ".whl". Dit bestandstype komt in de lijst terecht en als dit geselecteerd is, moet de *Default File Transfer Mode* op *Binary* staan. Nu is het mogelijk om installatiebestanden in USS te krijgen in het juiste formaat.

Jsonschema is beschikbaar via de Python AI toolkit en staat dus al op de mainframe. Dit is ook een .whl-bestand.

Er wordt ook gebruik gemaakt van de ZOAU, maar deze zijn al geïnstalleerd op het systeem en hebben geen verdere configuratie nodig.

Alle nodige packages worden opgeslagen in de directory */tmp/BPtestfolder/virtualEnv/packages*. Via het *cp*-commando worden de packages uit de Python AI toolkit gekopieerd naar de aangemaakte directory:

```
$ mkdir /tmp/BPtestfolder/virtualEnv/packages
$ cp /Path/To/AI/Toolkit/jsonschema-4.17.3-py3-none-any.whl \
                                    /tmp/BPtestfolder/virtualEnv/packages
```

Al deze packages moeten apart geïnstalleerd worden via het *pip install*-commando. Aangezien deze bestanden op het systeem staan, wordt de optie *–no-index* toegevoegd. Dit laat het systeem weten dat het niet online moet zoeken om de packages te installeren. De installatie gebeurt als volgt:

```
$ pip install ../packages/annotated_types-0.4.0-py3-none-any.whl --no-index
$ pip install ../packages/typing_extensions-4.10.0-py3-none-any.whl --no-index
$ pip install ../packages/pydantic-1.10.15-py3-none-any.whl --no-index
$ pip install ../packages/idna-3.6-py3-none-any.whl --no-index
$ pip install ../packages/sniffio-1.3.1-py3-none-any.whl --no-index
$ pip install ../packages/anyio-4.3.0-py3-none-any.whl --no-index
$ pip install ../packages/starlette-0.35.1-py3-none-any.whl --no-index
$ pip install ../packages/fastapi-0.109.0-py3-none-any.whl --no-index

$ pip install ../packages/h11-0.14.0-py3-none-any.whl --no-index
$ pip install ../packages/uvicorn-0.25.0-py3-none-any.whl --no-index

$ pip install ../packages/attrs-23.2.0-py3-none-any.whl --no-index
$ pip install ../packages/pyrsistent-0.20.0-py3-none-any.whl --no-index
$ pip install ../packages/jsonschema-4.17.3-py3-none-any.whl --no-index
```

7.5. Omgevingsvariabelen instellen

Zoals eerder vermeld, wordt er gebruik gemaakt van ZOAU om datasets te lezen en te schrijven. Deze package is al geïnstalleerd maar er moeten nog omgevingsvariabelen worden ingesteld zodat dit correct werkt. De omgevingsvariabelen die gewijzigd moeten worden, zijn *ZOAU_HOME*, *PATH* en *LIBPATH*. Hiervoor wordt het het *export*-commando gebruikt als volgt:

```
$ export ZOAU_HOME=/pp/idz/v16r0/zoautil/
$ export PATH=$ZOAU_HOME/bin:$PATH
$ export LIBPATH=$ZOAU\_HOME/lib:$LIBPATH
```

Als deze variabelen niet gewijzigd zijn, geeft het systeem volgende foutmelding:

```
CEE3201S The system detected an operation exception
                                        (System Completion Code=0C1).
From entry point _zoau_io_zopen at compile unit offset +0000000029B242B4
            at entry offset +00000000000002D4 at address 0000000029B242B4.
Killed
```

8

Pydev testen

Om Pydev te testen, wordt er een API in IDz geschreven. Deze API heeft als functie om van een batch job de JCL Job card en JCL Procedure aan elkaar te schrijven in een PDS-member. Hierdoor wordt het gebruik van geïnstalleerde packages alsook de verbinding met z/OS getest.

Het doel van dit hoofdstuk blijft om de ervaring te testen om een volledige Python-applicatie te schrijven door middel van Pydev en te zien hoe efficiënt dit is. Er wordt gebruikgemaakt van de ZOAU om datasets te lezen en te schrijven. Er is ook een klein JSON-bestand aanwezig die de naam van de job card -en procedure JCL bevat.

Door deze verschillende technologieën te gebruiken, wordt niet enkel getest welke complexe programma's er geschreven kunnen worden in Pydev, maar ook hoe efficiënt het is om op de mainframe gebruik te maken van deze moderne technieken in combinatie met de oudere.

8.1. Schrijven van een test API

Er werd een test API geschreven. Deze bevat een functie die 2 JCL's uit het JSON-bestand van een meegegeven job ophaalt en aan elkaar schrijft. Dit wordt geschreven in een dataset die gedefinieerd is in de variabele "dsMemberPath". Deze functie maakt gebruik van 2 andere functies om het JSON-bestand te lezen en om te schrijven naar een dataset. De API werd geschreven als volgt:

```
from fastapi import FastAPI
import json
from zoautil_py import zsystem, datasets
```

```
from zoautil_py.zoau_io import zopen

app = FastAPI()

@app.get("/api/proc/{proc}")
def get_jcl(proc: str):
    # ========================================================== #
    # ================= Declaring variables ================== #
    # ========================================================== #

    # header_path = path of the jobcard
    # body_path = path of the procedure
    header_path, body_path = get_values()

    # List to store the lines in of both the job card and procedure
    records = []

    # Output dataset
    outds = "AD12215.API.OUT" # Dataset
    member = proc # Dataset member
    dsMemberPath = f"{outds}({member})" # Full path of the dataset member


    # ========================================================== #
    # ============ Check if input datasets exist ============= #
    # ========================================================== #
    # Both files exist
    if (proc in datasets.list_members(header_path)) \
                        & (proc in datasets.list_members(body_path)):
        print(f"Both members found ({member}), \
            returning both the job card and procedure")

        # Reading and appending JOB file
        with zopen(f"//'{header_path}({proc})'", 'r', "cp1047") \
                        as record_stream:
            for record in record_stream:
                records.append(record.upper())

        # Reading and appending PROC file
        with zopen(f"//'{body_path}({proc})'", 'r', "cp1047") \
                        as record_stream:
```

```
        for record in record_stream:
            records.append(record.upper())

# Only header file exists
elif (proc in datasets.list_members(header_path)) \
                    & (proc not in datasets.list_members(body_path)):
    print(f"Member {body_path}({proc}) not found in procedure path, \
                    returning only {header_path}({proc})")

    # Reading and appending JOB file
    with zopen(f"//'{header_path}({proc})'", 'r', "cp1047") \
                    as record_stream:
        for record in record_stream:
            records.append(record.upper())

# Only body file exists
elif (proc not in datasets.list_members(header_path)) \
                    & (proc in datasets.list_members(body_path)):
    print(f"Member {header_path}({proc}) not found in job card, \
                    returning only {body_path}({proc})")

    # Reading and appending PROC file
    with zopen(f"//'{body_path}({proc})'", 'r', "cp1047") \
                    as record_stream:
        for record in record_stream:
            records.append(record.upper())

# No files exist
else:
    return "files don't exist"

# ========================================================== #
# ============ Writing output into a dataset ============= #
# ========================================================== #

# Deleting older output written if it exists
if member in datasets.list_members(outds):
    datasets.delete_members(f"{dsMemberPath}")
    if member in datasets.list_members(outds):
        return f"Dataset member {dsMemberPath} \
                    can't be deleted because it is open"
```

```
        # Writing the content of "records" into the dataset "dsMemberPath"
        return write_into_dataset(dsMemberPath, records)


def get_values():

    # Opening the json file with the file paths
    with open("config.json", 'r', encoding="utf-8") as test_file:

        # Loading the json in the variable json_data
        json_data = json.load(test_file)

        # Assign the return values with the correct filepaths
        header_path = json_data["HEADER"]
        body_path = json_data["BODY"]
        return header_path, body_path


def write_into_dataset(dsname: str, records: list):
    for record in records:
        # Remove all control characters for the record
        mapping = dict.fromkeys(range(32))
        res = record.translate(mapping)
        # Writing record into dataset
        datasets.write(dsname, res, append=True)

    return f"Output written in PDS {dsname}"
```

Bij het oproepen van de API geeft de gebruiker de naam van de job mee. Deze wordt opgeslagen in de variabele “proc”. Van deze job wordt de JCL job card en JCL procedure opgehaald door middel van de functie “get_values”. Hier wordt een JSON bestand gelezen dat de naam van de PDS bevat waarin de JCL members opgeslagen zijn. Dit is een andere dataset voor de job card en de procedure. Het JSON bestand ziet er als volgt uit:

```
{
    `HEADER`: `HOG0007.JOB.CNTL`
    `BODY`: `HOG0007.PROC.CNTL`
}
```

Hierna moet er gecontroleerd worden of deze JCL's bestaan. Er bestaat een "datasets.exists" functie in ZOAU maar deze is bedoeld om te controleren of een sequentiële dataset bestaat. Omdat de JCL bestanden in een PDS-member worden opgeslagen, kan dit niet gebruikt worden. Daarom wordt er gebruik gemaakt van de "datasets.list_members" functie. Zoals de naam vermeldt, geeft dit een lijst van members in een PDS terug. Als de naam van de job in deze lijst voorkomt, bestaat de JCL.

Om de datasets te lezen, wordt er gebruik gemaakt van het *zopen* statement uit de ZOAU. Dit is het equivalent van het traditionele *open* statement in Python om een bestand te openen. De "zopen" functie opent dus een dataset en hier worden er 3 positionele parameters gebruikt:

1 Het te openen bestand. Dit bevat de naam van de dataset.

2 Dit bepaalt wat er gebeurt met het bestand. *"r"* voor te lezen en *"w"* voor te schrijven

3 De encoding van het meegegeven bestand

Het is belangrijk om de encoding mee te geven bij het oproepen van deze functie. Datasets worden opgeslagen in de encoding *IBM-1047* en dit wordt gedefinieerd als *'cp1047'*.

Om de data te schrijven in een PDS-member, worden alle records in volgorde opgeslagen in een list. De job card en procedure zitten in dezelfde list genaamd "records". Eerst wordt de job card opgehaald en daarna de procedure. Deze records worden dan geschreven in de PDS-member die is gedefinieerd in de variabele "dsMemberPath". Voor het schrijven wordt er gebruik gemaakt van de functie "write_into_dataset" die 2 argumenten meekrijgt: de naam van de PDS-member waarin er geschreven wordt en de lijst met alle records. Deze functie maakt gebruik van de ZOAU functie "datasets.write". Als deze dataset niet bestaat, wordt ze automatisch aangemaakt.

8.2. De volledige applicatie uitvoeren

Om de API uit te voeren, wordt er gebruik gemaakt van de Uvicorn package met het volgend Bash-commando:

```
$ python -m uvicorn main:app
```

Dit commando moet uitgevoerd worden in dezelfde directory als het Python-bestand. De API luistert hierdoor op poort 8000 van het loopback adres van z/OS voor inkomende verzoeken.

Om te testen of het Python programma werkt, wordt er een nieuwe Python applicatie ontwikkeld die wordt uitgevoerd terwijl de API aan het luisteren is voor binnenkomende verzoeken. Voor deze applicatie wordt de “requests” package gebruikt.

```
import requests

api_url = `http://127.0.0.1:8000/api/proc/TC379700`
response = requests.get(api_url)
print(response.text)
```

In deze applicatie wordt er een request verzonden naar het loopback adres van de mainframe met de correcte url die de job “bn379” bevat.

9

Evaluatie test

Deze test applicatie maakte gebruik van de meeste functies waarvoor Python bedoeld is in combinatie met de mainframe. Een API wordt vaak geschreven in Python en kan dus ook uitgevoerd worden op de mainframe. Het lezen van JSON bestanden is ook een zeer belangrijke functie. Dit in combinatie met het gebruik van datasets op z/OS maakt Python alleen maar krachtiger.

Het schrijven van de API ging vrij vlot met Pydev, waardoor het zeer nuttig is om deze plug-in te implementeren. Dit werkte perfect met code completion voor traditionele statements zoals “return” of “with”. Als er variabelen gedeclareerd werden, konden deze ook automatisch ingevuld worden. Het werkt zoals een IDE zou moeten werken.

Hoewel het schrijven van de API vlot ging, was het uitvoeren in IDz niet zeer efficiënt. Er waren problemen met het opstarten van de Python virtuele omgeving wat niet voorkwam in een externe ssh connectie. De oorzaak van dit probleem is nog niet helemaal duidelijk. Het uitvoeren van de API bracht ook problemen met zich mee. Dit maakt duidelijk dat het testen van Python-applicaties in de IDz-terminal lukt als deze niet zeer complex zijn.

10

Conclusie

Deze proof of concept biedt een meerwaarde aan de modernisatie van het mainframe systeem die sterk bezig is. Deze systemen zijn nog heel belangrijk, maar werken met te oude technieken waardoor er niet veel mensen zijn die ze kunnen onderhouden. Het toevoegen van modernere manieren van werken is niet voldoende, ze moeten ook efficiënt zijn om mee te kunnen werken, wat niet het geval is in de huidige versie van IDz. Dit kan opgelost worden door middel van de Pydev plug-in.

Het opstellen van deze plug-in is niet zeer complex, maar er zijn instellingen die niet voor de hand liggend zijn en voor problemen kunnen zorgen. Aangezien IDz ook veel verschillende functionaliteiten heeft, is het moeilijk om te weten waar er gezocht moet worden bij het opstellen van deze plug-in.

Het is zeer eenvoudig om applicaties te schrijven en direct te testen in hetzelfde programma. Door de Pydev plug-in is het schrijven zeer efficiënt. De terminal in Idz om Python-applicaties in uit te voeren daarentegen brengt wat problemen met zich mee die niet voorkomen in een externe ssh-connectie. Tijdens het testen kwamen er problemen op bij iets complexere zaken, bijvoorbeeld het uitvoeren van de API of zelfs het opzetten van de Python virtuele omgeving.

Het hoofddoel van dit onderzoek was om een Python-interpreter op te zetten in IDz door middel van de Pydev plug-in wat goed gelukt is. Dit maakt het mogelijk om Python-applicaties direct te schrijven op de mainframe zonder een andere IDE te moeten gebruiken. Voor het opstellen werd er niet verwacht dat dit veel problemen met zich zou meebrengen behalve voor een paar instellingen.
Dit maakt het testen deels eenvoudiger, maar het wordt best gedaan in een externe ssh connectie om problemen te vermijden. Het is nog niet helemaal duidelijk wat de oorzaken zijn van de problemen in de terminal in IDz.

Dit onderzoek heeft duidelijk gemaakt dat Python compatibel is met de mainframe en nu is er ook een mogelijkheid om applicaties in deze programmeertaal direct te schrijven op de mainframe. Tools zoals ZOAU en de Python AI toolkit for z/OS maken de mogelijkheden enorm groot waardoor er zeer complexe applicaties geschreven kunnen worden. Door de afname van skills in COBOL en PL/1, is het interessant om te onderzoeken of Python en Java de oudere programmeertalen voor batch- en online-jobs volledig kunnen vervangen en of dit invloed zou hebben op de snelheid waarmee deze jobs worden uitgevoerd.

A

Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

Samenvatting

IBM heeft voor zijn IBM Z Systems de deur opengezet naar vernieuwing door Java en Python toegankelijk te maken op de Mainframe. Hoewel deze talen ondersteund worden, is het niet altijd even makkelijk om programma's hierin te schrijven. IBM Developer for z/OS is een stap in de goede richting door zijn functionaliteit om in een bekend programma scripts te openen en aan te passen die opgeslagen zijn op mainframe. Aangezien dit Eclipse based is, is er geen Python interpreter waardoor Python scripts aangepast moeten worden in een text editor.

Deze paper zal een proof of concept zijn voor het opstellen van Pydev en een Python IDE in IBM Developer for z/OS. Hierdoor kunnen Python developers efficiënter code schrijven, aanpassen en submitten in de Unix System Services omgeving van de mainframe. Verder zal er ook nog onderzocht worden om de output van de Python programma's te zien in de console van IBM Developer for z/OS. Dit is belangrijk voor de programmeurs van DNB die werken in dit programma die nu hun Python code moeten testen in een ssh connectie.

In dit onderzoek zal er gewerkt worden met de Pydev plug-in voor Eclipse en zal er een stap voor stap instructie gegeven worden over hoe dit ingesteld moet worden. Ook zal er onderzocht worden of deze scripts uitgevoerd kunnen worden door IBM Developer for z/OS in de Mainframe omgeving.

Het resultaat zal een volledige Proof of concept zijn voor een Python interpreter op te zetten in dit programma. Dit is vooral relevant voor Python developers in bedrijven die IBM Developer for z/OS gebruiken in combinatie met een IBM Z Mainframe.

A.1. Introductie

In de huidige technologische wereld is het moeilijk bij te houden welke mogelijkheden er zijn om bepaalde projecten uit te voeren. Zo heb je altijd nieuwe technieken die net iets efficiënter of krachtiger zijn dan een ander. Door deze voortgaande evolutie zullen individuen niet direct op de hoogte zijn van recente ontwikkelingen, maar vergeten ze ook de oudere toepassingen van een bepaalde techniek. Dit is vooral merkbaar bij IBM hun z Systems of mainframes die zo belangrijk, maar zo snel vergeten worden in de IT wereld. Hoewel het een zeer hoogwaardige technologie is, zijn de technieken nog steeds zeer oud. Hier wordt wel op ingezet door modernere talen zoals Java en Python compatibel te maken met dit systeem. Een veel gebruikte tool, aangeboden door IBM, is IBM Developer for z/OS. Dit is een Eclipse based programma geschreven in Java en kan een connectie maken met de mainframe en zijn opgeslagen bestanden. Dit maakt het eenvoudiger voor gebruikers om verschillende bestanden te openen maar heeft geen Python interpreter waardoor het voor developers niet zo efficiënt is om Python programma's aan te passen.

A.1.1. Probleemstelling

Het bedrijf waar het onderzoek in wordt uitgevoerd is Den Norske Bank (DNB) met een hoofdvestiging in Oslo, Noorwegen. Hun developers zoals Stian Botnevik zitten vast met het beschreven probleem en zoeken een efficiëntere manier om Python scripts te schrijven en testen. Momenteel passen ze Python scripts aan in een text editor van IBM Developer for z/OS (IDz) en testen ze de code in een ssh connectie. Python scripts worden opgeslagen en uitgevoerd in de Unix System Services (USS) omgeving. Dit is een Unix omgeving op de mainframe. In IDz kunnen developers alle bestanden openen in de USS omgeving wat duidelijker is dan code aanpassen in een terminal via bijvoorbeeld de vim tool.

A.2. State-of-the-art

A.2.1. Een andere manier van werken

Veel bedrijven die werken met een mainframe hebben het probleem dat ze onvoldoende mensen vinden met de nodige kennis over deze technologie. De introductie van Unix systemen op de mainframe in het jaar 2000 (Mertic, 2020) had dit probleem wat verminderd maar zeker niet weggewerkt. De oorzaak van deze situatie is door de oude technieken die het systeem gebruikt. COBOL en PL/1 zijn niet de meest aantrekkelijke programmeertalen om te leren en mensen kiezen liever Bash als scripttaal in plaats van REXX. Hoewel IBM veel inzet op documentatie en online leerplatformen zoals IBM Z Xplore, blijft het tekort van ervaren mensen nog steeds te laag.

De mainframe wereld zal zich dus moeten aanpassen aan de vaardigheden van

de mensen door over te schakelen naar bekendere manieren van werken. Python bijvoorbeeld is een programmeertaal die steeds meer populariteit krijgt sinds zijn creatie in de vroege jaren 90 (Johnson, 2023). Dit is niet enkel bij reeds ervaren programmeurs, maar mensen die net beginnen programmeren kiezen hier steeds vaker voor. Dit is vooral door zijn begin vriendelijkheid, verschillende doeleinden en een actieve community. (Johnson, 2023)

IBM heeft dit opgemerkt en een Python compiler en interpreter ontwikkeld voor z/OS genaamd IBM Open Enterprise SDK for Python. Hierdoor kun je met Python interactie hebben met z/OS om bijvoorbeeld applicaties te ontwikkelen of de resources van het systeem te gaan beheren. Door de Unix omgeving heeft de programmeur ook geen speciale z/OS kennis nodig. (Klaey, 2023)

A.2.2. Bedrijven in het werkveld

Een bank is een goed voorbeeld van een bedrijf die gebruik maakt van IBM hun mainframes. Dit valt te concluderen door het feit dat maar liefst 92 van de top 100 banken wereldwijd een mainframe gebruikt. Dit is ook logisch aangezien er gemiddeld 12.6 miljard financiële transacties per dag uitgevoerd worden. (Wagle, 2017)

Door dit aantal is de nood voor een mainframe toch niet te onderdrukken maar het is niet enkel het aantal transacties dat deze machine kan uitvoeren, het is ook de snelheid, schaalbaarheid en beveiliging van deze systemen dat een grote rol speelt. Het zijn ook niet enkel banken die van deze technologie gebruik maken, maar ook verzekeringsmaatschappijen bijvoorbeeld. De top 10 van deze soort bedrijven maken allemaal gebruik van een IBM mainframe. (Tozzi, 2022)

A.2.3. De Skill gap

Het is nog steeds moeilijk om nieuw talent aan te trekken in de mainframe wereld. Een onderzoek van Deloitte (2020) toont aan dat 79% van de projectleiders moeilijkheden heeft met het zoeken naar mensen met de juiste skillset. Hetzelfde onderzoek toont ook aan dat er in de teams zelf een groot verschil is van kennis en vaardigheden. Hoewel dit systeem gebruikt wordt door 71% van de fortune 500 companies (Tozzi, 2022) , is de Skill gap nog steeds te groot waardoor veel bedrijven vrezen voor een groot tekort aan werknemers om deze z Systems te onderhouden.

Volgens Petra Goude (2023) zijn er verschillende manieren om dit probleem aan te pakken. Zo kunnen bedrijven lessen geven over mainframes en hoe ze dit gebruiken. Ze vertelt ook dat de vaardigheden die nodig zijn beter gecomplimenteerd moeten worden en kunnen leiden naar een belonende en lange termijn carrière.

Hoewel dit zou helpen, vind ze dit niet de kern van het probleem. Het zijn de oude technieken die mensen niet aantrekt. Ze stelt voor om meer te investeren in hedendaagse technologie en dit te installeren op de mainframe. Dit kan gaan over dezelfde test -en deployment technieken, maar ook over hedendaagse programmeertalen zoals Java of Python. Dit zou kunnen door middel van APIs en zou een nieuwe, jongere werkkracht aantrekken. (Goude, 2023)

A.2.4. Toch nog een kleurrijke toekomst

Ondanks de grote nood aan mensen met de juiste vaardigheden, ziet de toekomst er toch nog goed uit. Zo wordt er meer gemoderniseerd met bijvoorbeeld modernere programmeertalen. Er wordt ook voorspeld dat we een introductie van DevOps en self-service benaderingen gaan zien. (Pennaz, 2023)

Momenteel zijn Python en Java beschikbaar als programmeertaal op de mainframe naast PL/1 en COBOL. Sinds deze introductie zijn al bijna 2/3de van gebruikers op de mainframe Java aan het toepassen op een bepaalde manier. (Watts, 2018)

A.3. Methodologie

Deze paper zal als resultaat een volledig stappenplan geven om Pydev op te stellen in IDz en verdere configuratie om de scripts uit te voeren in IDz.

A.3.1. Fase 1: Literatuurstudie

- **Doel:** Relevante informatie verzamelen van IDz, Pydev en de Unix System Services omgeving.
- **Aanpak:**
  - Opzoeken betrouwbare bronnen
  - Gelijke projecten onderzoeken
  - Overzicht van nodige software
- **Tijd:** 5 weken
- **Opbrengst:** Een volledig onderzoek naar vakliteratuur en nodige software die nodig is. Ook een volledige schets naar de omgeving waarin er wordt gewerkt.

A.3.2. Fase 2: Opstellen Pydev

- **Doel:** Opzetten van de Pydev plug-in in IDz
- **Aanpak:**
  - De opgezochte literatuur bestuderen

- Pydev opstellen met de juiste vereisten
- Testen

- **Tijd:** 3 weken
- **Opbrengst:** Pydev voor functies zoals code completion, syntax check, code analysis, ...
  Bij het openen van een Python programma zal dit ook gebeuren via een Python editor

A.3.3. Fase 3: Python interpreter opzetten

- **Doel:** Python Interpreter opstellen om Python programma's uit te voeren in IDz.
- **Aanpak:**
  - De opgezochte literatuur bestuderen met expert
  - Opzetten van de juiste interpreter
  - Testen
- **Tijd:** 2 weken
- **Opbrengst:** Python programma's kunnen uitvoeren in IDz.

A.3.4. Fase 4: Verdere configuratie om de output van de mainframe in de IDz console te krijgen

- **Doel:** Onderzoeken welke functies er beschikbaar zijn in Pydev
- **Aanpak:**
  - Pydev documentatie bekijken
  - Zelf onderzoeken in IDz
- **Tijd:** 2 weken
- **Opbrengst:** Een overzicht van alle functies die beschikbaar zijn.

A.3.5. Fase 5: Evaluatie voorbereiding

- **Doel:** Eind evaluatie voorbereiden
- **Aanpak:**
  - Resultaat bespreken, vergelijken en concluderen
  - Op orde stellen van nodige documenten

- **Tijd:** 2 weken
- **Opbrengst:** Een eind evaluatie waarin de opdracht besproken wordt

A.4. Verwacht resultaat, conclusie

Een volledig stappenplan voor de opstelling van Pydev en een Python interpreter die compatibel is met IDz. Ook zal de configuratie getoond worden om de output van Python programma's te zien in IDz. Hierbij zal er bij elke stap uitleg gegeven worden over waarom het op die bepaalde manier gedaan wordt. Ook zal er besproken worden wat er mogelijks mis zou kunnen gaan. Screenshots zullen gebruikt worden voor duidelijk te maken wat er gebeurd.

Door de Pydev plug-in zal de interpreter eenvoudig opgesteld kunnen worden. Hier worden wel wat problemen verwacht omdat Pydev voor standaard Eclipse is ontwikkeld en IDz is hier een afstamming van waardoor er bepaalde onderdelen anders kunnen zijn.

Bibliografie

BasuMallick, C. (2023). What Is a Mainframe? Features, Importance, and Examples. Verkregen maart 28, 2024, van https://www.spiceworks.com/tech/tech-101/articles/what-is-mainframe/

Bostian, J., & Rivera, E. (2023). Securely Leverage Open-Source Software with Python AI Toolkit for IBM z/OS. Verkregen mei 7, 2024, van https://www.redbooks.ibm.com/redpapers/pdfs/redp5709.pdf

Codecadamy. (2022). File System Structure. Verkregen mei 10, 2024, van https://www.codecademy.com/resources/docs/general/file-system-structure

Deloitte. (2020, juni 1). *Mainframe talent drain* (Survey). https://www2.deloitte.com/content/dam/Deloitte/us/Documents/Alliances/us-alliances-deloitte-ibm-mainframe-market-pulse-survey.pdf

Dhawan, A. (2013). What is USS? Verkregen mei 3, 2024, van https://www.zmainframes.com/viewtopic.php?t=49

Eclipse. (2006). What is a wizard? Verkregen mei 7, 2024, van https://wiki.eclipse.org/FAQ_What_is_a_wizard?

FastAPI. (z.d.). Features. Verkregen mei 21, 2024, van https://fastapi.tiangolo.com/features/

George, A. (2021). What Are Plugins, and How Do They Work? Verkregen mei 7, 2024, van https://www.lifewire.com/what-are-plugins-4582189

Goude, P. (2023). 5 ways to bridge the Mainframe skills gap. Verkregen december 13, 2023, van https://www.informationweek.com/it-infrastructure/5-ways-to-bridge-the-mainframe-skills-gap

Hanna, K. (2021). Eclipse (Eclipse Foundation). Verkregen mei 7, 2024, van https://www.techtarget.com/searchapparchitecture/definition/Eclipse-Eclipse-Foundation

HCL Technologies. (2022). HCL Z Data Tools User's Guide and Reference. Verkregen mei 3, 2024, van https://help.hcltechsw.com/zdt/1.1.0/en/base/hfs.html#:~:text=z/OS%C2%AE%20UNIX%E2%84%A2%20provides%20a%20Hierarchical%20File%20System,contain%20files%20or%20other%20subdirectories.

Henry-Stocker, S. (2017). All you need to know about Unix environment variables. Verkregen mei 15, 2024, van https://www.networkworld.com/article/964109/all-you-need-to-know-about-unix-environment-variables.html

IBM. (z.d.-a). Application Programming on z/OS. Verkregen mei 15, 2024, van https://www.ibm.com/docs/en/zosbasics/com.ibm.zos.zappldev/zappldev_book.pdf

IBM. (z.d.-b). Mainframe Concepts. Verkregen mei 3, 2024, van https://www.ibm.com/docs/en/zosbasics/com.ibm.zos.zmainframe/zmainframe_book.pdf

IBM. (z.d.-c). Resilient server solutions with IBM Z. Verkregen mei 2, 2024, van https://www.ibm.com/z/resiliency

IBM. (z.d.-d). z/OS concepts. Verkregen april 26, 2024, van https://www.ibm.com/docs/en/zosbasics/com.ibm.zos.zconcepts/zconcepts_book.pdf

IBM. (2012). z/OS Distributed File Service zSeries File System Implementation z/OS V1R13. Verkregen mei 3, 2024, van https://www.redbooks.ibm.com/redbooks/pdfs/sg246580.pdf

IBM. (2021a). Network security. Verkregen mei 7, 2024, van https://www.ibm.com/docs/pl/zos/2.4.0?topic=intermediate-network-security

IBM. (2021b). IBM Unveils On-Chip Accelerated Artificial Intelligence Processor. Verkregen mei 10, 2024, van https://newsroom.ibm.com/2021-08-23-IBM-Unveils-On-Chip-Accelerated-Artificial-Intelligence-Processor

IBM. (2022). Announcing IBM z16: Real-time AI for Transaction Processing at Scale and Industry's First Quantum-Safe System. Verkregen mei 10, 2024, van https://newsroom.ibm.com/2022-04-05-Announcing-IBM-z16-Real-time-AI-for-Transaction-Processing-at-Scale-and-Industrys-First-Quantum-Safe-System

IBM. (2023a). Why Z Open Automation Utilities. Verkregen mei 16, 2024, van https://www.ibm.com/docs/en/cloud-paks/z-modernization-stack/2023.4?topic=utilities-why-z-open-automation

IBM. (2023b). Introduction to SDSF. Verkregen mei 2, 2024, van https://www.ibm.com/docs/en/zos/2.5.0?topic=guide-introduction-sdsf

IBM. (2024a). Unit testing Enterprise COBOL and PL/I applications. Verkregen mei 2, 2024, van https://www.ibm.com/docs/en/developer-for-zos/16.0?topic=eclipse-unit-testing-enterprise-cobol-pli-applications

IBM. (2024b). What's new in IBM Developer for z/OS and Developer for z/OS Enterprise Edition. Verkregen mei 2, 2024, van https://www.ibm.com/docs/en/developer-for-zos/16.0?topic=zos-whats-new-in-developer

Johnson, A. (2023). Python Popularity: The Rise of A Global Programming Language. Verkregen december 13, 2023, van https://flatironschool.com/blog/python-popularity-the-rise-of-a-global-programming-language/#:~:text=Python%20Popularity%20Over%20The%20Years&text=Stack%20Overflow's%20developer%20survey%20results,as%20the%20most%20popular%20language.

Klaey, W. (2023). Python Programming on z/OS. Verkregen december 13, 2023, van https://www.linkedin.com/pulse/python-programming-zos-walter-klaey/?trackingId=eYKhSv70Tc6dnZLPooMDYQ==

Kosourova, E. (2022). What is Python Used For? 7 Real-Life Python Uses. Verkregen mei 16, 2024, van https://www.datacamp.com/blog/what-is-python-used-for

Mertic, J. (2020). Getting Jiggy With It: How a Linux Anniversary Led to the Creation of the Open Mainframe Project. Verkregen december 12, 2023, van https://newsroom.ibm.com/How-a-Linux-Anniversary-Led-to-the-Creation-of-the-Open-Mainframe-Project

Morning Consult & IBM. (2022). 2022 IBM GLOBALFINANCIAL FRAUDIMPACT REPORT. Verkregen mei 10, 2024, van https://filecache.mediaroom.com/mr5mr_ibmnewsroom/193031/MC%20+%20IBM%20Financial%20Fraud%20Study%20-%20Global%20Report%20Updated%203.8.22.pdf

Pennaz, M. (2023). The secret to attracting mainframe talent during a skills crisis. Verkregen december 13, 2023, van https://venturebeat.com/data-infrastructure/secret-to-attracting-mainframe-talent-during-skills-crisis/

Precisely. (2020). The Ultimate Guide to Mainframe Machine Data: SMF Data Beyond. Verkregen mei 3, 2024, van https://www.precisely.com/blog/mainframe/ultimate-guide-mainframe-machine-data-smf

Princeton University. (2022). Python and Virtual Environments. Verkregen mei 16, 2024, van https://csguide.cs.princeton.edu/software/virtualenv#:~:text=A%20Python%20virtual%20environment%20(venv,installed%20in%20the%20specific%20venv.

Pydev. (2024). What is PyDev? Verkregen mei 16, 2024, van https://www.pydev.org/

Python Software Foundation. (2024). venv — Creation of virtual environments¶. Verkregen mei 16, 2024, van https://docs.python.org/3/library/venv.html

Rivera, E. (2023). Python® AI Toolkit for IBM® z/OS®. Verkregen mei 7, 2024, van https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/evan-rivera/2023/02/24/python-ai-toolkit-for-ibm-zos?communityKey=038560b2-e962-4500-b0b5-e3745175a065

Rupp, M. (2022). AN INTRODUCTION TO Z/OS AND THE IBM COMMON CRYPTOGRAPHIC ARCHITECTURE. Verkregen mei 2, 2024, van https://www.cryptomathic.com/news-events/blog/payment-banking-an-introduction-to-z/os-and-the-ibm-common-cryptographic-architecture

Sayer, P. (2022). IBM's z16 mainframe boasts on-chip AI acceleration. Verkregen mei 7, 2024, van https://www.cio.com/article/307884/ibms-z16-mainframe-boasts-on-chip-ai-acceleration.html

Sentry. (2024). Understand the purpose of Uvicorn in FastAPI applications. Verkregen mei 21, 2024, van https://sentry.io/answers/understand-the-purpose-

of-uvicorn-in-fastapi-applications/#:~:text=Uvicorn%20is%20a%20web%20server,servers%20that%20run%20asynchronous%20code.

Singh, V. (2024). Difference Between Module and Package in Python. Verkregen mei 21, 2024, van https://www.shiksha.com/online-courses/articles/difference-between-module-and-package-in-python/#:~:text=Definition,with%20an%20_init_.py%20file.

Singhal, A. (2023). Parsing Mainframe files (EBCDIC files). Verkregen mei 7, 2024, van https://medium.com/@amar.singhal/data-migration-from-mainframes-parsing-ebcdic-files-83d4900eb0ab

Spohn, J. (2023). IBM Developer for z/OSEnterprise Edition. Verkregen maart 28, 2024, van https://www.ibm.com/downloads/cas/DJ5P7W3N

Tozzi, C. (2022). 10 Mainframe Statistics That May Surprise You. Verkregen december 13, 2023, van https://www.precisely.com/blog/mainframe/9-mainframe-statistics

Turner, D. (2022). PAYMENT BANKING CRYPTOGRAPHY: THE BENEFITS OF Z/OS THE Z PLATFORM. Verkregen mei 2, 2024, van https://www.cryptomathic.com/news-events/blog/payment-banking-cryptography-an-overview-of-the-benefits-of-z/os-and-the-z-platform

Udacity. (2021). What Is A Python Package? Verkregen mei 21, 2024, van https://www.udacity.com/blog/2021/01/what-is-a-python-package.html

Wagle, L. (2017). The modern mainframe. Verkregen december 12, 2023, van https://www.ibm.com/thought-leadership/institute-business-value/en-us/report/modernmainframe

Watts, S. (2018). Java on the Mainframe: z/OS vs Linux. Verkregen december 13, 2023, van https://www.bmc.com/blogs/java-mainframe-zos-linux/