# Dia 2

Overlopen inhoud

* Toelichting onderwerp
* Git branching strategie
* DBB configuratie
* Pipeline software & strategie
* Compileren & binden
* Coolgen code opsplitsen
* Demo’s

# Dia 3 - Toelichting onderwerp

Huidig systeem

* Panapt
* Meta data

Toekomstig systeem

* Pipelines
* Meta data

Belang meta data in beide systemen

# Dia 4 – Git branching strategie

Main (productie) 🡪 always deployable

Develop (ontwikkel) 🡪 aanpassen programma’s via hier

Release (QA) 🡪 tussen fase voor van ontwikkeling naar productie te gaan

Lokale branches 🡪 feature branches om aanpassingen lokaal te maken en te testen

Hotfix branch (nood) 🡪 probleem productie op hotfix branch zetten en van daaruit rechtstreeks naar productie mergen

# Dia 5 – DBB configuratie

Link naar dbb-zappbuild github 🡪 sample scripts en configuratie bestanden zijn daar te vinden

# Dia 6 – DBB configuratie

Build-conf bestanden:

* Cobol.properties
* Build.properties
* Datasets.properties

# Dia 7 – DBB configuratie

Application-conf bestanden:

* File.properties
* Application.properties
* Cobol.properties
* .gitattributes

# Dia 8 – DBB configuratie

Language scripts (groovy)

* Cobol.groovy 🡪 voor cobol compileren/binden
* Coolgen.groovy 🡪 voor opsplitsen Coolgen code

# Dia 9 – DBB configuratie

Utility scripts (groovy)

* Bindutilities.groovy
* Buildutilities.groovy

# Dia 10 – DBB configuratie

Shell scripts

* AzRocketGit-init.sh
* AzDBB-build.sh

# Dia 11 – Pipeline software & strategie

Software 🡪 Azure DevOps (Repos & Pipelines) + IBM DBB (shell- & groovy scripts)

# Dia 12 – Pipeline software & strategie

Pipeline opbouw 🡪 tasks (clone & build) + agent + ssh conn + variables + triggers

# Dia 13 – Pipeline software & strategie

2 tasks

* Clone task
  + Opmaken build omgeving
  + Clone repo
* Build task
  + Starten groovy build script
  + Cleanup na uitvoering

# Dia 14 – Pipeline software & strategie

Ssh connective 🡪 toegang tot mainframe (via functionele user normaal) 🡪 Personal Acces Token (PAT) 🡪 .gitconfig & .profile

# Dia 15 – Pipeline software & strategie

Variables

* Pipeline variables
  + MyApp 🡪 map waarin source code staat
  + MyRepo 🡪 verwijzing naar repository (link)
  + MyWorkDir 🡪 locatie waardat de working directory van de pipeline komt (workspace)
* Standaard variables 🡪 worden allemaal gebruikt in de pipeline variables voor automatisatie
  + Build.Repository.Uri
  + System.teamProject
  + Build.buildid

# Dia 16 – Compileren & binden

Compileren 🡪 build.groovy & Cobol.groovy

Build.groovy

* buildList (changed files)
* buildOrder
* buildProperties

Cobol.groovy

* buildList
* LogicalFile
* Compileropties
* Syslib concatenatie
* Extra DD (SYSUEXIT, SYSDEFSD, …)

# Dia 17 – Compileren & binden

Binden 🡪 build.groovy & Cobol.groovy

Build.groovy

* Net zoals eerder!!!!

Cobol.groovy

* Net zoals eerder
* + bindopties in plaats van compileropties
* + include statements (imports-bestand)

# Dia 18 – Coolgen code opsplitsen

Groovy scripts

* Build.groovy
* Coolgen.groovy

Shell scripts 🡪 Hetzelfde als bij een normale compile/bind

Input 🡪 /coolgen/\*.cg bestand

Output 🡪 cobol code en pseudo code apart in coolgen map :

/coolgen/\*.cbl & /coolgen/\*.ief

# Dia 19 – Demo’s (demo splitsen Coolgen code)

Er kunnen meerdere Coolgen programma’s tegelijk gesplitst worden

Coolgen wordt gesplitst in pseudo code (IEF) en Cobol code (CBL)

* Via dezelfde pipeline als normale compile/bind
* Na aanmaken van de output wordt automatisch een push gedaan.
* De pseudo code wordt gevonden met behulp van regex (+->)
* Er kan zowel Coolgen programma’s opgesplitst worden als Cobol programma’s gecompileerd worden in dezelfde commit.

# Dia 20 – Demo’s (demo volledige compile/bind)

* Meerdere programma’s tegelijk
* Verschillende load libraries (hangt af van meta data)
* In de AM volgorde (sub, fsub, main)
* De juiste subsystemen worden toegevoegd
* Meta data is belangrijk
* logicalFile = opsomming van de meta data van een programma
* automatisch package- en plan bind

JCL voor de output van de voorbeeld programma’s te testen is: **COBDB2**

# Dia 21 – Demo’s (demo alleen fsub en sub compile

* Toon verschil tussen output van vorige keer en deze keer
* De fsubs zijn mooi aangepast maar de sub niet
* Dit komt doordat de sub niet dynamisch wordt opgeroepen en de fsubs wel
* Fsubs worden aan de hand van een IMPORTS bestand ingeladen 🡪 SID0U0.PLX0.IDZ.PARMLIB(IMPORTS)