APL als Webservice

Bestehende Anwendungen und neue Schnittstellen

Finn Flug - DPC

Agenda

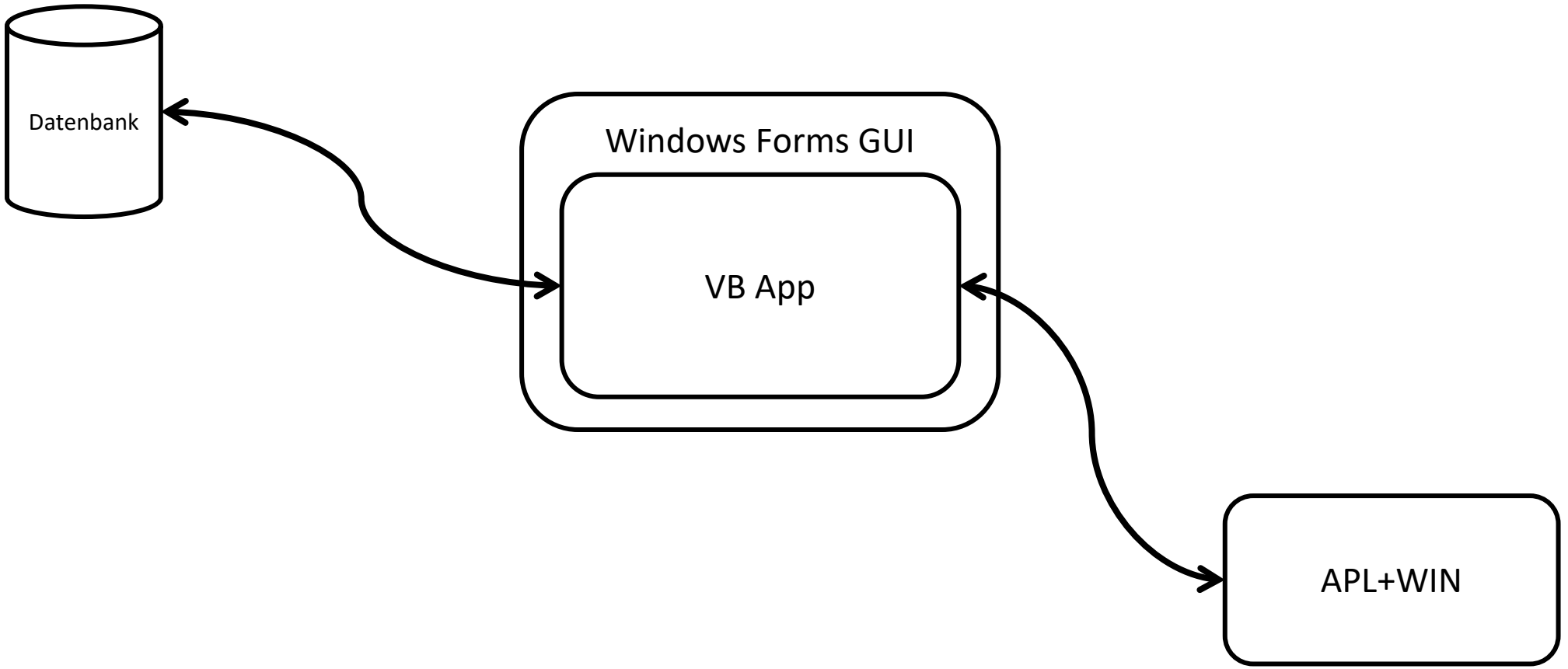
- Überblick aktuelle Anwendung
- Migration von APL+WIN nach Dyalog-APL
- Bereitstellung als Jarvis-basierten Webservice
- Deployment des Webservices als Docker Container

Überblick Anwendung - aktuell

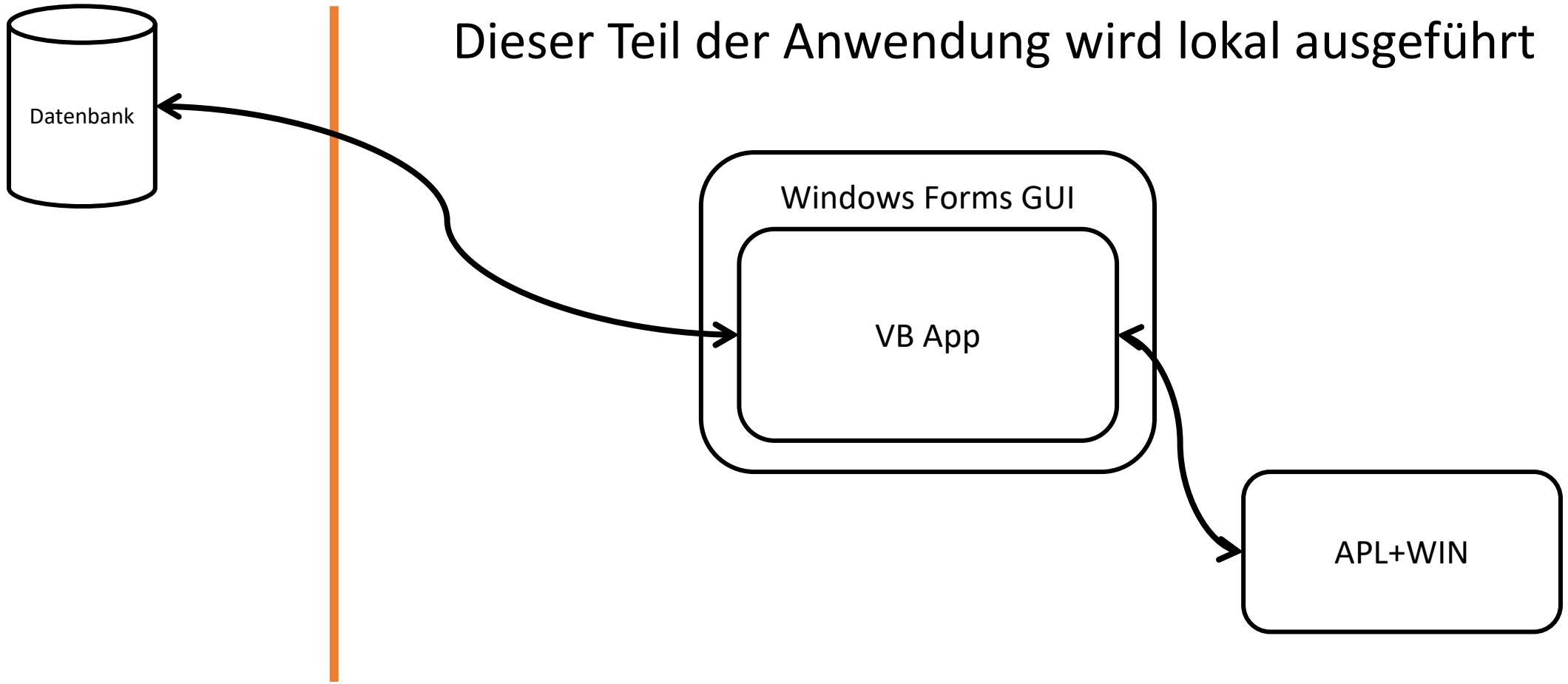
- Anwendung geschrieben in Visual Basic 6
- Windows Forms GUI
- Rechenkern geschrieben in APL+WIN
- Rechenkern wird als COM-Server bereitgestellt
  - Die Visual Basic Anwendung ruft den Rechenkern im Wesentlichen wie folgt auf (analoger Dyalog-Code):

```
WSengine←⎕NEW'OLEClient'(⊂'ClassName' 'APLW.WSEngine')
WSengine.SysCommand⊂'Load /path/to/workspace'
WSengine.Call1 'foo' arg
```

Überblick Anwendung - aktuell



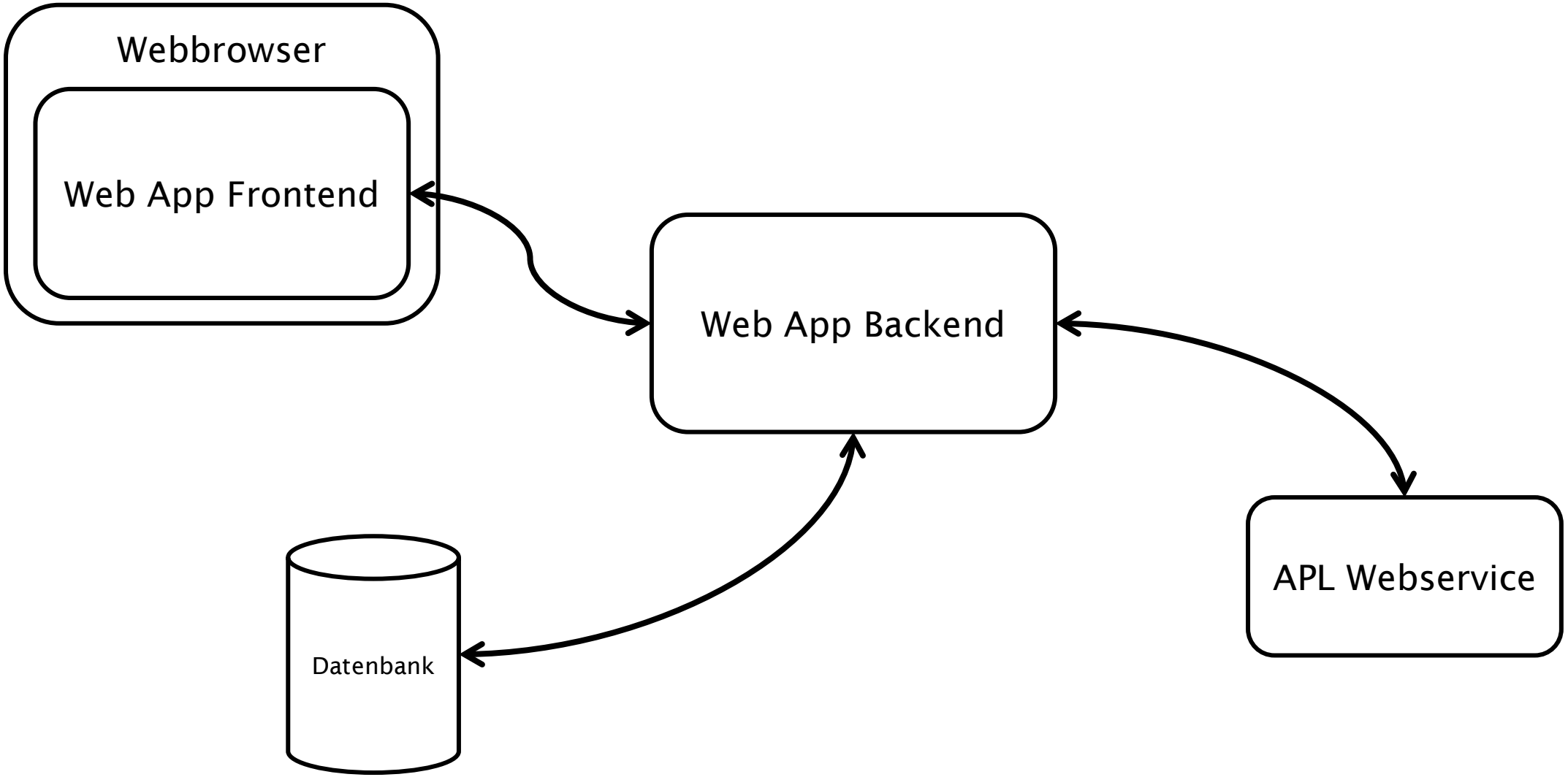
Überblick Anwendung - aktuell



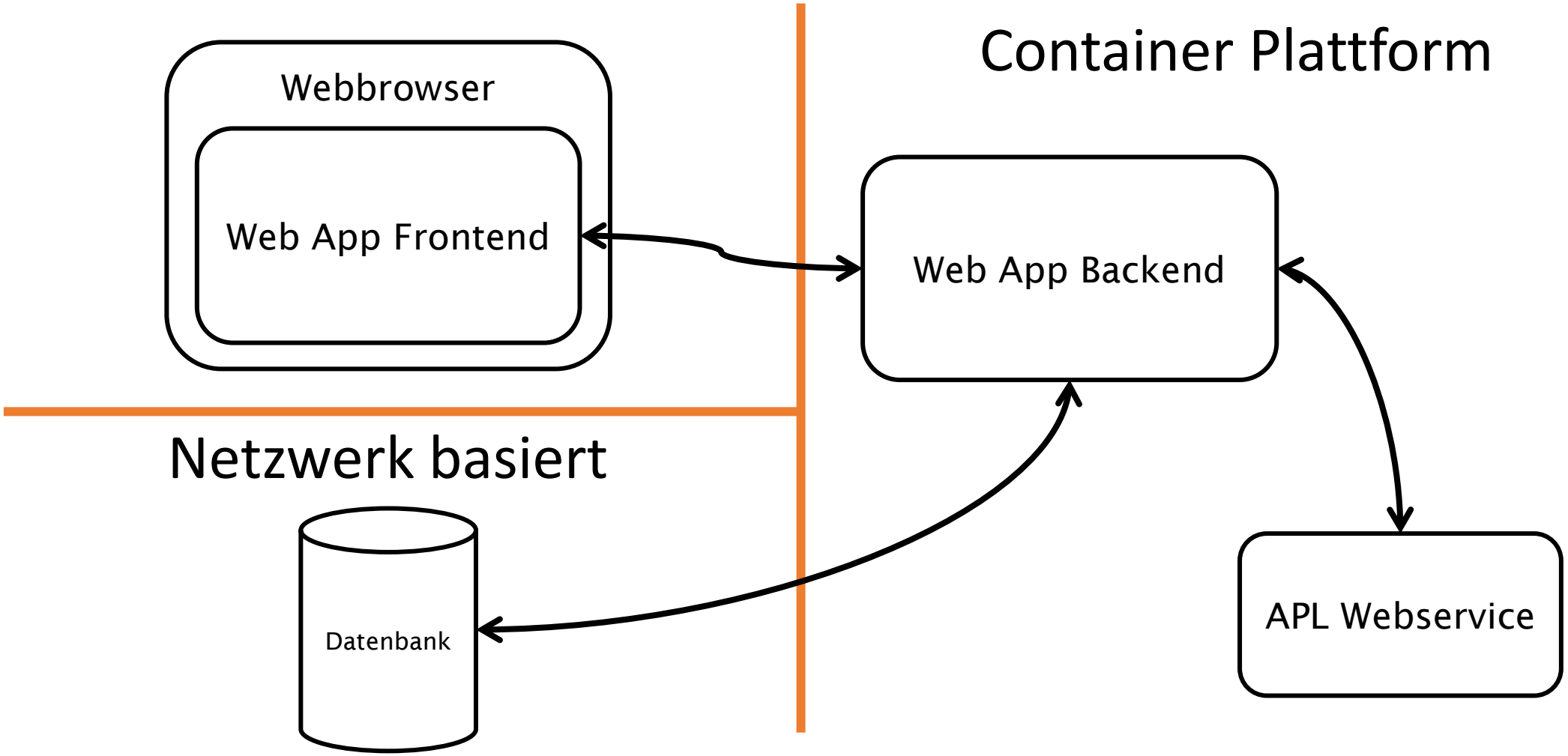
Überblick Anwendung - Ziel

- Warum soll das System umgestellt werden?
  - Support für Visual Basic 6 läuft aus
  - Die aktuelle Architektur des Systems passt nicht zur Infrastruktur des Kunden
- Das Ziel ist:
  - Umstellung auf eine Browser-basierte Lösung
  - Veraltete Komponenten ersetzen

Überblick Anwendung - Ziel



Überblick Anwendung - Ziel



Plan

1. Migration von APL+WIN nach Dyalog
2. Bereitstellung des Rechenkerns als Jarvis-basierten Webservice
3. Bereitstellung als Docker Container

Migration von APL+WIN nach Dyalog

- Zu beachten ist beispielsweise Replicate-Each:

```
      1 0/¨(1 2)(3 4) ⍝ Dyalog
1 2
      1 0/¨(1 2)(3 4) ⍝ APL+WIN
1 3
```

- Ersetzen von Systemfunktionen und Systemvariablen (⎕…)
- In APL+WIN sind alle linken Argumente optional, in Dyalog jedoch nicht
- …

Bereitstellung als Jarvis-basierter Webservice

- Festlegung des Paradigmas:
  - Jarvis (**J**SON **a**nd **R**est ser**vis**) unterstützt zwei Paradigmen, JSON und REST
  - Wir haben uns aus folgenden Gründen für JSON entschieden:
    - Es ist geeignet für funktionale Endpunkte (d.h. Webservice-Endpunkte entsprechen APL-Funktionen)
    - Es ist einfacher zu implementieren
- Wichtige Frage: Ist die Anwendung zustandslos?

Bereitstellung als Jarvis-basierter Webservice

- Überarbeitung des existierenden APL-Codes
  - Endpunkte sind monadische oder dyadische APL-Funktionen mit Rückgabewert
    - Rechtes Argument ist der Request-Payload
    - (Optionales) linkes Argument ist das Request-Objekt selbst
  - Jarvis übernimmt die Konversion zwischen JSON und APL-Datenstrukturen (mit Hilfe von `⎕JSON`)
    - Die Signatur der APL-Funktionen muss unter Umständen überarbeitet werden
    - Die Endpunkte des Rechenkerns nehmen durch Semikolons separierte Parameter als Textvektor entgegen (z.B., `'Finn;1234;3.1415926'`)
    - Es waren daher keine Änderungen erforderlich (obwohl dies eine suboptimale Lösung darstellt)

Bereitstellung als Jarvis-basierter Webservice

- Konfiguration
  - Konfigurationsparameter können in einer .json-Datei spezifiziert werden
  - Wir verwenden in etwa folgende Konfiguration:

```
{
    AppCloseFn: "myAppCloseFn",        // wird aufgerufen wenn Jarvis startet
    AppInitFn: "myAppInitFn",          // wird aufgerufen wenn Jarvis stoppt
    CodeLocation: "./APLSource",       // Standardmäßig werden alle Funktionen in CodeLocation als Endpunkt bereitgestellt
    ExcludeFns: "Errorhandling.*",     // Diese Funktionen werden nicht als Endpunkte bereitgestellt
    Port: 8000,                        // An Jarvis vergebener Port
}
```

Bereitstellung als Jarvis-basierter Webservice

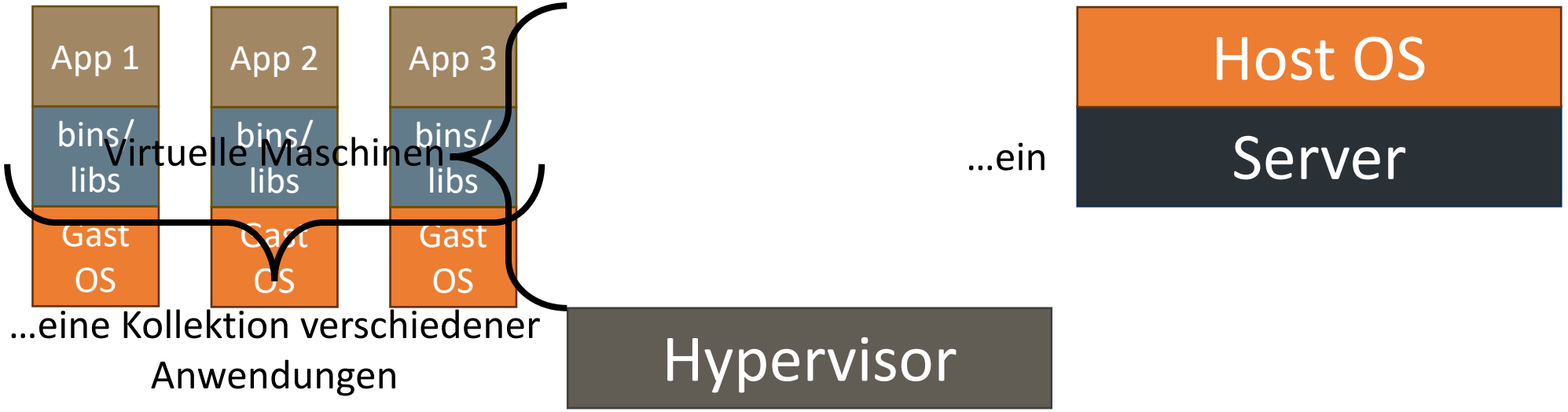
Nun kann die Anwendung als Webservice bereitgestellt werden…

…zumindest auf Localhost

```
Jarvis.Run '/Pfad/zu/config.json'
```

Exkurs: Was ist ein Docker Container?

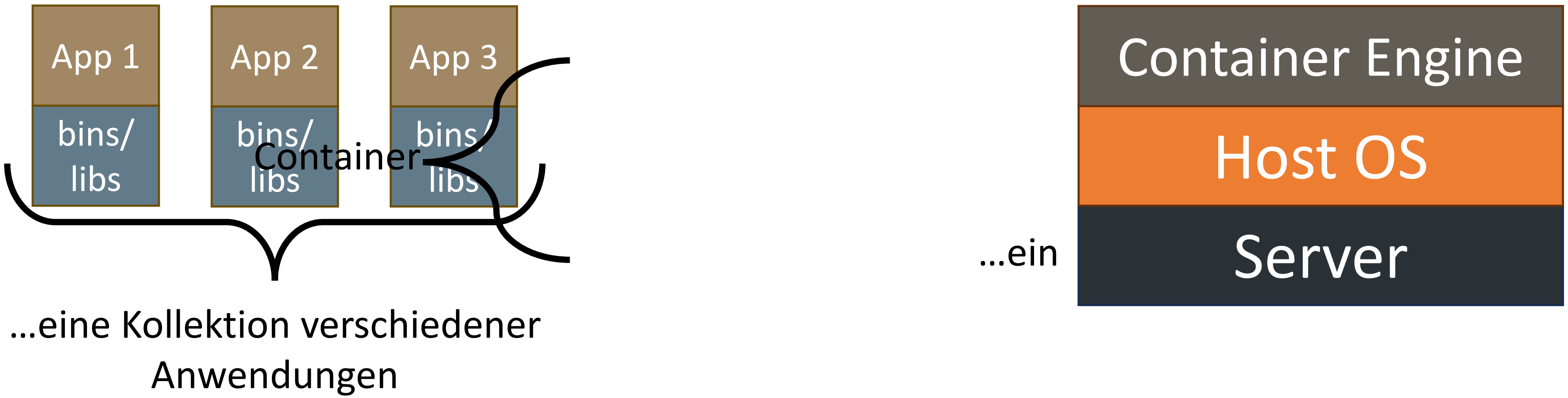
- Auffrischung virtuelle Maschine (VM):
  Gedanklich Betriebssysteme zusammenfügen



Exkurs: Was ist ein Docker Container?

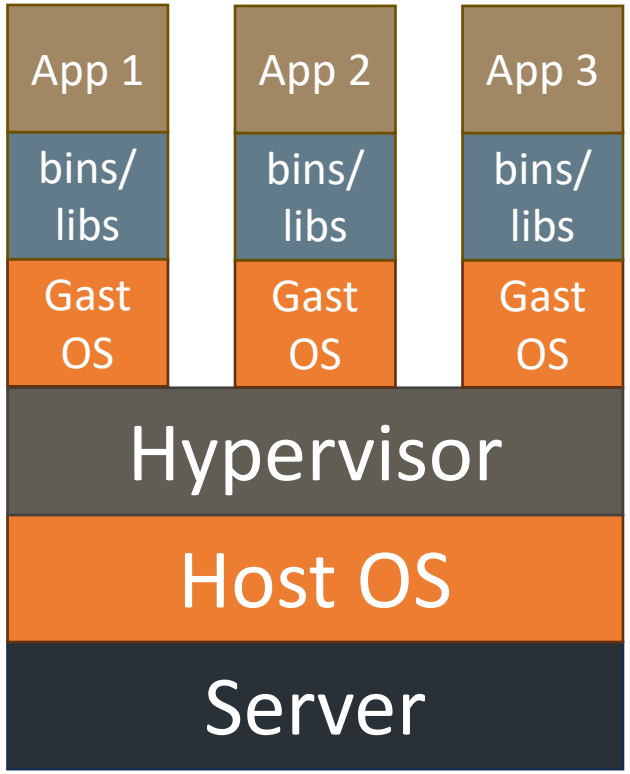
- Kurzeinführung (Docker) Container

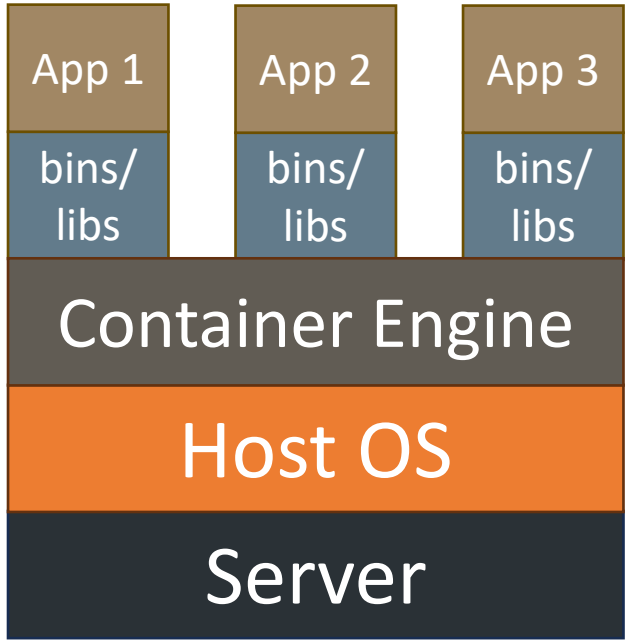
[illegible]



Exkurs: Was ist ein Docker Container?

- Gegenüberstellung der Modelle





Erstellung eines eigenen Docker Images

- Dyalog stellt zu Testzwecken Docker Images bereit
  - Diese sind nicht für den Einsatz in Produktion vorgesehen!
- Wir haben das Docker File für das dyalog/jarvis Image als Ausgangspunkt verwendet
  - Grundlegende Unterschiede:
    - Das Base Image
    - Laden von Abhängigkeiten
    - Zusätzliche Konfiguration
  - Komponenten des Docker Images
    - Base Image
    - Interpreter
    - Jarvis
    - Quellcode (als Textdateien gespeichert)

Erstellung eines eigenen Docker Images

- Vereinfachte Version des Docker Files

```
FROM redhat/ubi8-minimal:8.8

ADD APLSource /app
ADD linux_64_18.2.45405_unicode.x86_64.rpm /dyalog.rpm
RUN git clone https://github.com/dyalog/Jarvis /Jarvis

ENV JarvisConfig="/app/Config.json“
ENV LOAD="/Jarvis/Source“

ENTRYPOINT dyalog
```

Erstellung eines eigenen Docker Images

- Vereinfachte Version des Docker Files

Base Image →

```
FROM redhat/ubi8-minimal:8.8
```

Hinzufügen von Komponenten (Achtung bei Versionen!) →

```
ADD APLSource /app
ADD linux_64_18.2.45405_unicode.x86_64.rpm /dyalog.rpm
RUN git clone https://github.com/dyalog/Jarvis /Jarvis
```

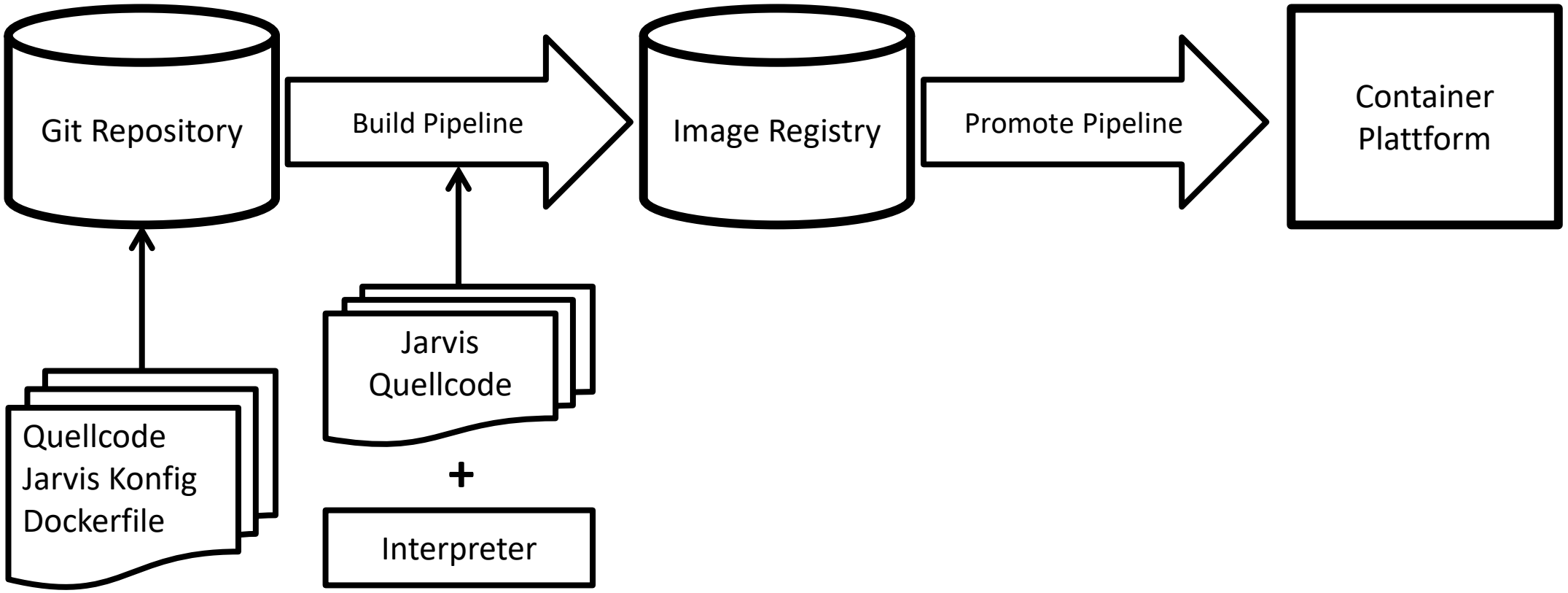
Umgebungsvariablen →

```
ENV JarvisConfig="/app/Config.json“
ENV LOAD="/Jarvis/Source“
```

Anwendung, die beim Start ausgeführt werden soll →

```
ENTRYPOINT dyalog
```

Build & Deploy



Was noch?

- Sicherheit
- Testen
- Fehlerbehandlung
- Logging
- Aktualisierung der verschiedenen Komponenten
- …

Zusammenfassung

- Bis jetzt läuft alles reibungslos
- Die Anwendung ist noch nicht produktiv, die Tests sehen jedoch vielversprechend aus
- Umstellung verlief ohne Probleme