Nach Redmond über Wolke 7

Java Cloud Computing mit Windows Azure

Klaus Baumgartner, Benno Vogel

Wer nach einer Grundlage für seine eigene Java-Anwendung für die Cloud sucht, denkt wahrscheinlich zunächst eher an wohlbekannte Hersteller von Suchmaschinen und Shoppingplattformen als an den Betriebssystemhersteller aus Redmond. Doch gerade für die Migration bestehender Java EE-Anwendungen gibt es gute Gründe, sich Microsofts Windows Azure genau anzusehen. Mit Java-APIs und Eclipse-Plug-in kann das „Software plus Service"-Produkt Azure Anwendungen ein neues Zuhause sein, die sich auf APIs wie z. B. JAX-RS, JDBC oder EJB stützen. Der Artikel zeigt, wie (und wann) eine bestehende Java EE-Anwendung auf Wolke Sieben gehoben werden kann.

Einführung

Es ist nicht Ziel dieses Artikels, das Für und Wider von Cloud-basierten Architekturen oder der Cloud-Infrastruktur eines bestimmten Anbieters (hier: Windows Azure) zu diskutieren. Es geht auch nicht um den Vergleich von Windows Azure mit Plattformen anderer Cloud-Anbieter.

Worum geht es dann? Wir gehen davon aus, dass Sie sich bereits – aus verschiedenen Gründen – für Windows Azure als Plattform entschieden haben, und beschreiben Ihnen auf technischer Ebene, wie Sie Ihre bestehende Java EE-Anwendung auf Microsofts Cloud-Plattform Windows Azure migrieren. Mit diesem Wissen sollten Sie natürlich auch in der Lage sein, eine Java-Enterprise-Anwendung für Microsoft Azure von Grund auf selbst zu entwickeln.

Eine Schlüsselanforderung aus Sicht eines Java-Entwicklers ist die Verwendung einer Standard-Entwicklungsumgebung (hier Eclipse) und der Verzicht auf zusätzliche – insbesondere kostenpflichtige – Werkzeuge (z. B. Visual Studio). Kenntnisse über gängige Java EE-APIs setzen wir voraus.

Ergänzend verweisen wir auf einen mehrteiligen Screencast von Klaus Baumgartner, der die Migration einer Java EE-Beispielapplikation nach Windows Azure zeigt [Baum11].

Windows Azure

Die Cloud-Infrastruktur dient als „Rechenzentrum" für den Betrieb Ihrer Applikation(en). Die Cloud stellt Ihnen quasi beliebig viele Instanzen jeder Server-Konfiguration zur Verfügung, die unter einem gemeinsamen DNS-Namen erreichbar sind. Die Last wird per Loadbalancer transparent auf alle Instanzen verteilt.

Windows Azure definiert drei verschiedene Systemumgebungen, sogenannte „Roles", in die eigene Software eingebracht werden kann: Web Role, Worker Role und VM Role. Alle Roles führen ein Microsoft-Windows-System aus.

Die *Web Role* dient dem Betrieb einer ASP.NET-Applikation auf Basis eines IIS-Web-Containers. Sie ist für den Betrieb von Java-Anwendungen daher uninteressant.

Die *Worker Role* führt prinzipiell beliebige Windows-Executables aus. Bei jedem Systemstart wird ein Startup-Skript ausgeführt, über das die Umgebung angepasst und die notwendigen Programme gestartet werden können. Für unsere Zwecke wären hier zumindest eine Java VM und eine Applikation als Bytecode zu deployen, im Startup-Skript zu entpacken und zu starten.

Das „Windows Azure for Java Starter Kit" [AzurePlugIn4EJ] stellt die notwendige Ordnerstruktur, ein Skript-Template sowie einige nützliche Werkzeuge für den Initialisierungsprozess zur Verfügung.

Die *VM Role* erlaubt die Bereitstellung eines kompletten Windows-Betriebssystems. Für reine Java-Anwendungen ist auch diese Role uninteressant, da die Java VM ohnehin das Betriebssystem abstrahiert. Prinzipiell wäre der Betrieb als VM Role aber möglich, beispielsweise wenn von der Java-Applikation native Dienste mit spezifischen Anforderungen an das Betriebssystem verwendet werden.

Typische Aspekte einer Java EE-Applikation

Enterprise-Systeme sind (nahezu ausnahmslos) verteilte Systeme. Drei Hauptaspekte sind Teil fast jeder Java EE-Applikation:

- HTTP-Kommunikation,
- Enterprise Java Beans (EJB) und
- Persistenzschicht (Relationale Datenbank).

HTTP

HTTP(S) ist das Standardprotokoll für Web-basierte Benutzungsoberflächen (Thin oder Rich Clients). Wegen der anhaltenden Beliebtheit von Webservices wird es auch zunehmend als Protokoll zur Kommunikation zwischen Anwendungen eingesetzt.

Im Java EE-Umfeld stellt ein Web-Container der Anwendung die protokollseitige Funktionalität zur Verfügung. Über die Servlet/JSP-API bzw. unter Benutzung der „Java API for XML Web Services" (JAX-WS) greifen eigene Komponenten (lesend oder schreibend) auf Nachrichtenheader, -body oder Parameter zu.

Soll die Anwendung ohne größere Eingriffe in die Cloud migriert werden, muss also ein Container als Laufzeitumgebung bereitgestellt werden.

Wichtig: Rückgriff auf Server-State ist prinzipiell zu vermeiden! Der Azure-Loadbalancer unterstützt keine Sticky Sessions. Das „Kleben" an einer Maschine hat ohnehin negativen Einfluss auf die Verfügbarkeit. Session Replication beeinträchtigt die Skalierbarkeit. Falls horizontale Skalierung nicht das Ziel einer Migration ist (sondern z. B. Betriebskosten), existieren mögliche Workarounds (s. Punkt „Server-State" in Tabelle 1).

EJB

Die EJB-API ist inzwischen weitläufig bekannt und eine bequeme Programmierschnittstelle für verteilte Anwendungen und Transaktionen. Sie ist ein Standard-Mittel für die Integration von Infrastrukturkomponenten.

Als API für Remote-Kommunikation ist EJB protokollbedingt (RMI/IIOP) netzwerktechnisch von jeher problematischer als das zustandslose HTTP. Man denke beispielsweise an Firewalls und dynamische Rückkanäle.

Die Integration von On-Premise-Komponenten oder -Diensten mit einer auf Windows Azure gehosteten Applikation kann nicht auf dem gewohnten Weg erfolgen, für diesen Zweck steht der Service-Bus der AppFabric [Sirt11] zur Verfügung.

Persistenzschicht

Heutzutage werden in der Persistenzschicht häufig Relationale Datenbanken und O/R-Mapper eingesetzt. Persistenz kann aber auch auf andere Weise realisiert werden, beispielsweise im Dateisystem. Windows Azure stellt verschiedene Speicher zur Verfügung:

- SQL Azure (JDBC): verteilte relationale Datenbank,
- TableStorage: strukturierte Daten in Tabellenform,
- BlobStorage: binäre Daten (BLOB = binary large object).

Die verschiedenen Speichertypen werden später genauer behandelt.

Entwicklungswerkzeuge

Details zu Download und Installation der einzelnen Tools bietet Teil 2 der Screencasts [Baum11]. Download-Links zu allen verwendeten Paketen finden sich am Ende des Artikels.

Der Entwicklungsrechner
Das Windows Azure SDK ist nur für Windows verfügbar, daher sollte die Entwicklung in einer Windows-Umgebung erfolgen. Abbildung 1 zeigt die Installation des Entwicklungsrechners.

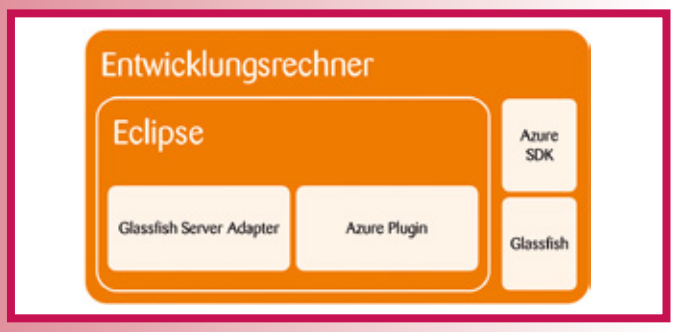

Abb. 1: Installation des Entwicklungsrechners

Entwicklungsumgebung (Eclipse)
Die Windows Azure Tools zur Erzeugung eines zu Azure konformen Deployment-Pakets sind derzeit für Eclipse verfügbar. Im Screencast [Baum11] wird Eclipse Indigo (3.7) verwendet.

Windows Azure Starter Kit
Dieses Eclipse-Plug-in übernimmt die Paketierung der Anwendung mit den notwendigen Komponenten und dem Start-Skript für die Initialisierung der Windows-Azure-Instanz.

Windows Azure SDK
Das SDK erlaubt das Testen der Cloud-Applikation auf dem lokalen Windows-Arbeitsplatz. So kann man sich das Deployment in die Cloud während der Entwicklungsphase sparen.

Applikationsserver und Eclipse-Serverumgebung
In unserem Screencast [Baum11] wird Glassfish als Java EE-Container verwendet. Denkbar wäre ebenso der Betrieb von Apache Tomcat, JBoss AS oder einem anderen Container bzw. der Betrieb einer stand-alone Java-Applikation.

Der Applikationsserver wird für das Development auch auf dem Entwicklungsrechner installiert. Für die Integration in Eclipse und gegebenenfalls Hot Deployment kann außerdem ein entsprechender Server-Adapter installiert werden.

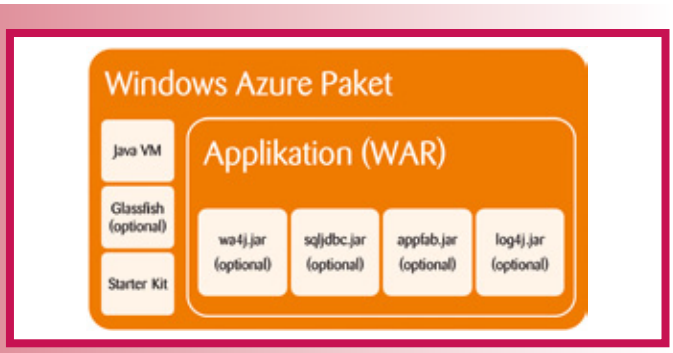

Abb. 2: Bestandteile eines Azure-Pakets

Das Azure-Paket
Folgende Komponenten (s. Abb. 2) sind Teil des Azure-Pakets und müssen entsprechend bereitgestellt werden.

Java VM
Eine Java VM ist in jedem Fall für den Betrieb einer Java-Anwendung notwendig. Zwecks automatischer Installation durch das Startup-Skript empfiehlt sich eine Version ohne Installer (z. B. von PortableApps).

Optional: Applikationsserver
Glassfish oder ein anderer Applikations-Container. Auch hier wählt man eine portable Version oder packt die bereinigte Installation auf dem Entwicklungsrechner in ein Archiv.

Starter Kit
Der Paket-Rahmen und die Startup-Skripte werden vom Eclipse-Plug-in (s. u.) generiert.

Archive: EAR/WAR
Die eigentliche Applikation in dem Format, wie sie in den eingesetzten Container deployed wird, inklusive sämtlicher verwendeter Bibliotheken – im Archiv integriert oder separat.

Eclipse-Plug-in

Das Eclipse-Plug-in für Windows Azure stellt einen neuen Projekttyp zur Verfügung, mit dem ein Paket für das Deployment in Windows Azure geschnürt werden kann. Der Projekt-Wizard erzeugt die notwendige Projektstruktur mit einem Template für das Startup-Skript sowie einigen Utilities.

Des Weiteren erlaubt das Eclipse-Plug-in – ebenfalls über mit dem Projekt generierte Skripte – die Steuerung der lokalen Windows Azure-Simulation, mit der eine Anwendung lokal getestet werden kann.

Die Verwendung des Plug-ins wird im Screencast, Teil 4 [Baum11] ausführlich gezeigt.

Paket-Erstellung
Die Grundstruktur eines Azure-Pakets wird durch das Anlegen eines neuen Projekts mit dem Typ „Windows Azure Project" generiert. Der Projekt-Wizard erlaubt die Angabe verschiedener Rollen mit der jeweils gewünschten Anzahl Instanzen. Für den lokalen Test ist hier darauf zu achten, nur eine Instanz zu erzeugen, um Konflikte um Netzwerk-Ports zu vermeiden. Denn jede Instanz würde versuchen, die konfigurierten Ports (z. B. 80 oder 8080 für Http) auf dem lokalen Rechner zu öffnen.

Hinweis: Projekt- und Role-Namen sollten keine Leerzeichen enthalten, da es sonst mit der Generierung Probleme geben kann.

Die erzeugte Projektstruktur enthält im Ordner `cert` ein Zertifikat, das *testweise* für Remote-Desktop-Verbindungen auf Azure-Instanzen verwendbar ist. Für Entwicklung und Betrieb ist dieses öffentlich bekannte Zertifikat durch ein eigenes zu ersetzen.

Für jede konfigurierte Rolle gibt es einen weiteren Unterordner von `approot`, der eine Beispielapplikation (HelloWorld.zip) und einen weiteren Ordner `util` mit einigen nützlichen Skripten enthält:

- startup.cmd: Das schon erwähnte Startup-Skript.
- download.vbs: VB-Script, um weitere Artefakte aus der Cloud nachzuladen.
- unzip.vbs: VB-Skript zum Entpacken von ZIP-Archiven.
- log.cmd: Skript, das im Startup-Skript verwendet werden kann, um ein Log zu schreiben.

Im Ordner `approot` werden Artefakte, die für den Betrieb der Java-Applikation notwendig sind (VM, Container, Applikation

usw.), bereitgestellt. Diese werden dann bei der Initialisierung einer Instanz durch das Startup-Skript installiert und gestartet.

Schließlich liegt im Projektordner das Ant-Buildskript `package.xml`, das eigentliche Starter-Kit, das aus der Projektstruktur fertige Pakete für das Deployment in Windows Azure erzeugt (default-target createwapackage).

Erstellen des Startup-Skripts

Das Skript `startup.cmd` initialisiert bei jedem Start die WorkerRole-Instanz. Anstatt sie direkt in das Azure-Paket einzubinden, kann man große Archive (JVM, AppServer) hier auch on demand von einem BlobStorage-Space oder einer anderen URL herunterladen.

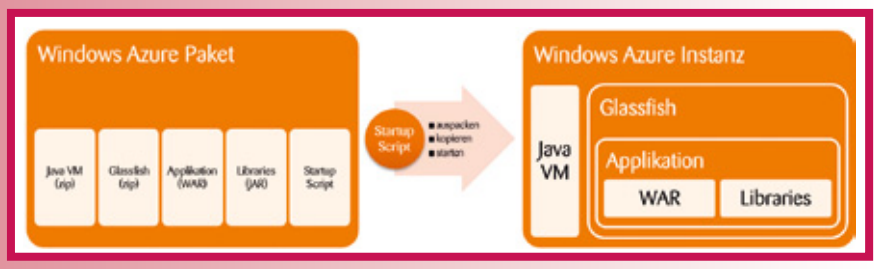

Abb. 3: Initialisierung der Azure-Instanz

Beim Startup sind folgende Aufgaben zu erledigen:

- Gegebenenfalls Herunterladen weiterer Pakete.
- Entpacken und/oder Installieren der JVM, des Applikations-Containers (wenn vorhanden) und gegebenenfalls weiterer Artefakte.
- Gegebenenfalls Deployen der Applikations-Artefakte in den Container.
- Konfigurieren der Betriebsumgebung, das heißt: Setzen der benötigten Umgebungsvariablen (JAVA_HOME, PATH) und gegebenenfalls Anpassungen im Betriebssystem (z. B.: der Registry).
- Starten der Applikation (VM bzw. Container) und gegebenenfalls weiterer benötigter Dienste.

Lokales Testen und Debugging

In den Properties des Windows Azure-Projekts muss angegeben werden, ob ein Paket für den Einsatz in der lokalen Testumgebung oder in der wirklichen Azure-Cloud geschnürt werden soll. Das Ant-Buildskript generiert das jeweils passende Paket.

Hinweise: Für lokale Tests nur eine WorkerRole-Instanz konfigurieren! Das Port-Mapping scheint lokal nicht zu funktionieren. Daher sind nur die internen Ports verwendbar.

Cloud-Deployment

Haben wir ein Azure-Paket erzeugt, wollen wir dieses auch in die Cloud bringen.

Deployment über das Management Portal

Im Windows Azure Management Portal [Azure] wählt man für jede WorkerRole das erzeugte Azure-Paket und die dazugehörige Konfigurationsdatei zum Upload aus. Nach der Übertragung wird die konfigurierte Anzahl von Instanzen gestartet.

Fehlersuche

Auch in der Cloud können Fehler auftreten. Wie können diese ohne zeitaufwendige Roundtrips gefunden und behoben werden? Die Fehlersuche fällt wesentlich leichter, wenn nur eine Instanz gestartet wird, sodass alle Requests von dieser behandelt werden.

Damit man sich via Remote Desktop auf die Instanz verbinden kann, müssen für die Worker Role Benutzer und Zertifikat konfiguriert werden. Dort angemeldet, kann man Logs analysieren, das Startup-Skript ändern und schrittweise ausführen, um die Initialisierung der Instanz durchzuspielen. Im Teil 5 der Screencasts [Baum11] wird sowohl das Deployment als auch die Fehlersuche gezeigt.

Tipp: Über Remote Desktop funktioniert das Clipboard für Texte und Dateien!

Azure-Speichertypen

Die Abbildung existierender Persistenzmechanismen auf die Speichertypen von Windows Azure sollte wohl überlegt sein. Es handelt sich immerhin um verteilte Systeme, die hohe Skalierbarkeit und Verfügbarkeit bieten, aber dies zulasten der Konsistenz (BASE-Prinzip, s. Screencast, Teil 1 [Baum11]). Wer bisher auf ACID-Transaktionen setzte, muss sich hier Gedanken machen.

Einen Anhaltspunkt für die Wahl der Persistenzabbildung könnten die Gründe liefern, die den Schritt in die Cloud motiviert haben, z. B.:

- Günstige Realisierung hoher Verfügbarkeit: Zur Erreichung dieses Ziels ist SQL Azure wahrscheinlich die beste Wahl. Bestehende Anwendungen können oft einfach durch Austausch des JDBC-Treibers und der Datenbank-URL sowie kleinerer Änderungen am Datenbankschema migriert werden.
- Horizontale Skalierung oder die Verwaltung großer Datenmengen: Hier kommt es darauf an. Oft ist eine Anpassung der Persistenzschicht hin zu Distributed Storage (Table- oder BlobStorage) die einzig mögliche Lösung. Vor allem im Falle hoher Skalierung muss man sich der Auswirkungen des BASE-Prinzips [Baum11] bewusst sein.

SQL Azure entspricht im Wesentlichen einem in der Cloud verteilten SQL-Server, der es erlaubt, ein relationales Datenbankschema horizontal zu skalieren. In Enterprise-Anwendungen ist das Datenaufkommen oft gut abschätzbar. Dann bietet sich die Möglichkeit, nach SQL Azure umzuziehen. Diese Migration wird im Screencast, Teil 5 [Baum11] ausführlich behandelt.

Hinweis: SQL Azure-Datenbanken sind derzeit auf eine Größe von 50 GB begrenzt. Dieses Limit will Microsoft in naher Zukunft anpassen.

Je nachdem, wie die Applikation auf die Datenbank zugreift, sind für die Migration auf SQL Azure verschiedene Anpassungen notwendig. Bei JDBC (Java Database Connectivity):

- JDBC-Treiber austauschen und
- Connection String (ggf. in Container Resource) konfigurieren.

Bei JPA/ORM (Java Persistence API/Object-Relational Mapping):

- JDBC-Ressource konfigurieren (JPA: persistence.xml), evtl. SQL-Dialekt anpassen und
- bei komplexen Schemata kann Detailarbeit notwendig sein. Hierzu kann es helfen, vom ORM-Framework ein SQL-Schema erzeugen zu lassen und dieses vor der Installation manuell anzupassen.

Generell kann es natürlich bei der Umstellung von einer anderen Datenbank zu implementierungsspezifischen Problemen im Schema oder in den SQL-Statements kommen.

Tipp: Dank JDBC funktionieren Java-Datenbank-Werkzeuge auch mit SQL Azure!

Die Applikation verwendet …	
JDK Version X.Y	☑ die entsprechende VM wird mitgeliefert ⚠ für Windows verfügbar?
JDBC	☑ SQL Azure JDBC Treiber benutzen ⚠ Keepalive-Bug (s. [Baum11]!) ⚠ Maximale Datenbankgröße: derzeit 50 GB
ORM	☑ siehe JDBC ⚠Table-/BlobStorage sind keine geeigneten Provider für Object-Relational Mapping (ORM)
Log4j/java.util.logging	☑ Custom Appender/Handler! Mögliches Backend: TableStorage (s. [Baum11])
Dateisystem	☑ für lokale temporäre Daten ☑ sonst Lösung mit BlobStorage, CDN oder SMB-Share
Container	☑ Windows Azure for Java Starter Kit: VM, Container und Startup-Skript im Paket
Servlet/JSP	☑ siehe Container
EJB	☑ siehe Container ⚠ Einschränkung: RMI/IIOP zu On Premise-Servern →AppFabric-Service-Bus
Webservices	☑ siehe Container
CDI	☑ siehe Container
JMS, JCA, JavaMail, …	☑ Bereitstellung durch Container oder im Azure-Paket
On-Premise-Integration	☑ AppFabric-Service-Bus
Server-State	⚠ Server-State ist (nicht nur in der Cloud) zu vermeiden Beeinträchtigt Performance und Skalierbarkeit Besser: State persistieren ☑ Workaround 1: Session Stickyness via IIS und Application Request Router (ARR) Instanz-Reboots bei Betriebssystem-Patches ☑ Workaround 2: State Replication AppServer Clustering u. Replication, z. B. über distributed Cache ⚠ Alle Session-Attribute müssen serialisierbar sein!

Tabelle 1: Checkliste – Wann flutscht die Migration?

TableStorage dient zur Ablage strukturierter Daten. Dieser Speichertyp wird in Teil 6 der Screencasts [Baum11] behandelt. *Blob Storage* dient zur Ablage binärer Objekte und kann für die Bereitstellung von Installationspaketen für Worker Roles verwendet werden.

Fazit

Abschließend fasst Tabelle 1 zusammen, für welche Arten von Java EE-Anwendungen die Migration auf „Wolke Sieben“ mit Microsofts Windows Azure Erfolg verspricht.

Literatur und Links

[AppFabric] AppFabric SDK 4 Java, `http://jdotnetservices.sourceforge.net`

[Azure] Windows Azure Management Portal, `http://www.microsoft.com/windowsazure`

[Azure4J] Windows Azure SDK 4 Java, `http://www.windowsazure4j.org`

[AzureJDBC] SQL Azure JDBC Treiber, `http://www.microsoft.com/download/en/details.aspx?id=19847`

[AzurePlugIn4EJ] Windows Azure Starter Kit (Eclipse-Plug-in), `http://webdownload.persistent.co.in/windowsazureplugin4ej`

[AzureSDK] Windows Azure SDK, `http://www.microsoft.com/windowsazure/sdk`

[Baum11] Screencasts von K. Baumgartner zum Thema „Java in the Cloud on Windows Azure“,
Teil 1 (Azure Overview): `http://www.youtube.com/watch?v=doSXy9WC-o4`
Teil 2 (Tools &Download): `http://www.youtube.com/watch?v=eU7n9G0kqOM`
Teil 3 (Beispiel-App): `http://www.youtube.com/watch?v=i5bwEVC4cuQ`
Teil 4 (Eclipse-Plug-in): `http://www.youtube.com/watch?v=1LwXxhN4hnw`
Teil 5 (SQL Azure): `http://www.youtube.com/watch?v=sFOiNf9HgcQ`
Teil 6 (Table Storage): `http://www.youtube.com/watch?v=NrKZoujNebE`

[Eclipse] `http://www.eclipse.org/downloads`

[Glassfish] `http://glassfish.java.net`

[JavaVMPortable] `http://portableapps.com/apps/utilities/java_portable`

[Log4j] `http://logging.apache.org/log4j/1.2/download.html`

[Sirt11] H. Sirtl, Microsoft Azure für Java-Entwickler, in: JavaSPEKTRUM, 2/2011

Klaus Baumgartner ist Softwarearchitekt bei der Zühlke Engineering GmbH. Für einschlägige Bücher und interaktive Trainings, die im Verlag Addison-Wesley erschienen sind, zeichnet er als Autor und Co-Autor verantwortlich. Mitarbeiter zahlreicher großer Unternehmen zählen zu seinen Schulungs-teilnehmern und Projektkunden.
E-Mail: kba@zuehlke.com

Benno Vogel arbeitet bei der Zühlke Engineering GmbH als Softwarearchitekt mit dem Schwerpunkt Java-Enterprise-Architekturen. Er beschäftigt sich außerdem mit Integrationsthemen sowie Qualitäts-aspekten wie Performance, Skalierbarkeit, Verfügbarkeit und Wartbarkeit.
E-Mail: benno.vogel@zuehlke.com