Azzurro

Microsoft Azure für Java-Entwickler

Holger Sirtl

Cloud Computing ist derzeit das Thema der IT-Welt. In der Tat sind die Versprechungen wie z. B. flexible Skalierbarkeit der bereitgestellten IT-Ressourcen, nutzungsabhängige Kosten und standardisierte Zugriffsschnittstellen für Anwender und Entwickler gleichermaßen interessant. Microsoft bietet mit Azure nun eine Cloud-Plattform für Entwickler an. Dieser Artikel beschreibt, wie diese Plattform aufgebaut ist, wie sie sich in die Microsoft-Plattform einbettet und welche Möglichkeiten sie insbesondere für Java-Entwickler bietet.

Aus den zahllosen Definitionen des Begriffs Cloud Computing haben sich ein paar Eigenschaften als wesentlich herauskristallisiert: Eine IT-Funktion wird servicebasiert auf Basis einer hoch skalierbaren Infrastruktur über standardisierte Schnittstellen angeboten. Es kann sich dabei um IT-Funktionen verschiedener Abstraktionsstufen – angefangen von virtualisierter Hardware (Infrastruktur-as-a-Service, IaaS), über Infrastruktur- und Anwendungsservices für Entwickler (Plattform-as-a-Service, PaaS) bis hin zu ganzen Softwaresystemen für Fachanwender (Software-as-a-Service, SaaS) – handeln.

Cloud Computing mit Microsoft

Microsoft hat das eigene Produktportfolio in den letzten Jahren in all den genannten Abstraktionsstufen erweitert. In weltweit aufgebauten Rechenzentren werden IaaS-Dienste, die sogenannten „Global Foundation Services" (GFS), erbracht. Diese bilden das technische Fundament für alle Cloud Services von Microsoft. Im November 2008 veröffentlichte Microsoft die Plattform Windows Azure, kurz: Azure, die ein PaaS-Angebot darstellt. Azure stellt Rechen- und Speicherdienste auf Basis der GFS bereit und erlaubt es Entwicklern, diese Dienste zu nutzen, um eigene Anwendungen um Cloud Services anzureichern bzw. eigene Anwendungen in der Cloud zu installieren, zu überwachen und zu betreiben.

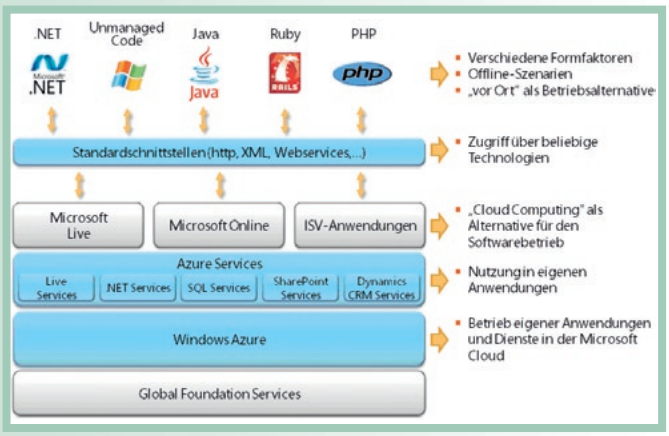

Abb. 1: Azure im Rahmen von Microsofts Cloud Services

Derzeit ist Azure als Community Technology Preview (CTP) für Entwickler verfügbar. Eine kommerzielle Verfügbarkeit ist für November 2009 geplant. Azure wird in Zukunft auch Basis für Microsofts SaaS-Angebote „Microsoft Live" und „Microsoft Online". Die Live Services sind Anwendungsdienste für Privatkunden und Kleinunternehmen (Office Live Workspace, Office Live Small Business, Windows Live, …), die Online Services sind Dienste für Anwender bei Unternehmenskunden (SharePoint Online, Exchange Online, …).

Alle Cloud Services sind über Standardschnittstellen (http, REST, SOAP, WS-*) zugreifbar. Damit erfüllen Microsofts Cloud Services ein weiteres Wesensmerkmal des Cloud Computings. Die Dienste können im Prinzip über jede Technologie genutzt werden, die ebenfalls diese Standards unterstützt. Hier hat Microsoft tatsächlich keine eigenen Schnittstellen geschaffen. Selbst bei der Programmierung mit den .NET SDKs bilden die bereitgestellten Hilfsklassen Azure-Aufrufe auf Webservices ab. Abbildung 1 zeigt den Aufbau von Microsofts Cloud Services-Angebot und die Einbettung der Plattform Windows Azure.

Die Bestandteile von Windows Azure

Azure umfasst zwei Hauptbestandteile: Windows Azure und die Azure Services. Windows Azure übernimmt in der Microsoft Cloud die Aufgaben eines Betriebssystems. Technisch setzt es auf Windows Server 2008 auf, kapselt dessen Funktionen aber in einer Serviceschicht, über die neben Speicher- und Rechendiensten auch Ausführungs- und Managementfunktionen für Anwendungen und Dienste angeboten werden. Der Windows Azure Storage umfasst Queues, Blob-Speicher und Tabellen. Zugriffe auf Windows Azure erfolgen stets über diese Services bzw. eine webbasierte Oberfläche.

Mit den Azure Services stellt Microsoft darüber hinaus eine Reihe von Infrastruktur- und Anwendungsdiensten zur Nutzung durch Entwickler zur Verfügung. Auf diese Dienste können Entwickler über Standardtechnologien (http, SOAP, REST, XML, …) zugreifen. Die Nutzung der Azure Services setzt also nicht voraus, dass die aufrufende Anwendung selbst auf Windows Azure ausgeführt wird. Die Azure Services gruppieren sich zu den folgenden fünf Dienstgruppen:

Die *Live Services* umfassen eine Reihe von Diensten, die in Anwendungen die Speicherung, Verarbeitung und Synchronisation von anwenderbezogenen Daten über verschiedene Geräte hinweg unterstützen. Dazu gehören unter anderem Diens-

te zur Speicherung von Authentifizierungsdaten (Live ID), Kommunikations- und Präsenzinformationen (Live Messenger), Geodaten (Live Maps), Informationssuche (Live Search).

Die Vernetzung, Integration und Orchestrierung von Services ist die Domäne der *.NET Services*. Diese stellen Infrastrukturdienste bereit, die zur Kommunikation und Integration verteilter Anwendungskomponenten verwendet werden können, und umfassen Dienste für Claims-basierte Zugriffskontrolle, einen Internet Service Bus (ISB) sowie Workflow-Dienste zur Definition von Service-Aufrufsequenzen, die ihrerseits als Service angeboten werden können.

SQL Services stellen Datenbankfunktionalitäten in der Microsoft Cloud zur Verfügung. Diese bieten derzeit die Möglichkeit, Daten in relationalen Strukturen zu speichern. In zukünftigen Ausbaustufen werden Analyse- und Reportingfunktionen folgen.

Die *SharePoint* bzw. *Dynamics CRM Services* bieten schließlich Dienste, die bei der Entwicklung von kollaborativen Anwendungen bzw. im Kontext von Anwendungen im Bereich des Kundenmanagements benötigt werden.

```
public static void main(String args[]) throws Exception {
  while (true) {
    String id = null;
    String popreceipt = null;
    String text = null;
    HttpClient httpclient = new DefaultHttpClient();

    HttpGet get =
      new HttpGet("http://" + account + ".queue.core.windows.net/" +
        queue + "/messages");
    Sign(get, account, key);
    HttpResponse response = httpclient.execute(get);

    BufferedReader rd = new BufferedReader(
      new InputStreamReader(response.getEntity().getContent(), "UTF-8"));
    rd.skip(1);
    NodeList messages =
      DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(
        new org.xml.sax.InputSource(rd)).getElementsByTagName(
          "QueueMessage");
    if (messages.getLength() > 0) {
      NodeList children = messages.item(0).getChildNodes();

      for (int i = 0; i < children.getLength(); i++) {
        Node node = children.item(i);

        if (node.getNodeName() == "MessageId") {
          id = node.getFirstChild().getNodeValue();
        }
        else if (node.getNodeName() == "PopReceipt") {
          popreceipt = node.getFirstChild().getNodeValue();
        }
        else if (node.getNodeName() == "MessageText") {
          text = new String(Base64Coder.decodeString(
                   node.getFirstChild().getNodeValue()));
          System.out.println("Text: " + text);
        }
      }

      rd.close();

      HttpDelete delete =
        new HttpDelete("http://" + account +
          ".queue.core.windows.net/" + queue +
          "/messages/" + id + "?popreceipt=" + popreceipt);
      Sign(delete, account, key);
      httpclient.execute(delete);
    }
    else {
      Thread.sleep(1000);
    }
  }
}
```

Listing 1: Verarbeitung von Nachrichten, die über Azure Queues empfangen werden (Datei: worker.java)

Möglichkeiten für Java-Entwickler

Die Nutzung von Azure ist prinzipiell von jeder Technologie aus möglich, die die von Azure bereitgestellten Webservices-Schnittstellen aufrufen kann. Selbstverständlich gehört hierzu auch Java. Für Java-Entwickler bieten sich drei Möglichkeiten zur Nutzung von Azure-Funktionalitäten:

- Aufruf von Azure Services aus einer Java-Anwendung,
- Integration von Java-Services über .NET Services aus Azure,
- Betrieb von Java-Anwendungen auf Windows Azure.

Die folgenden Abschnitte erläutern genauer, wie Azure in den drei Szenarien genutzt werden kann. Voraussetzung zur Implementierung der einzelnen Beispiele sind Azure-Accounts (für Windows Azure bzw. die Azure .NET Services), die sich Entwickler kostenfrei über die Azure-Homepage [Azure] beantragen und einrichten können.

Nutzung von Azure Services aus einer Java-Anwendung heraus

Eines der Wesensmerkmale der Serviceorientierung ist die Bereitstellung von Schnittstellen, über die verschiedene Funktionalitäten verfügbar sind. Die innere Funktionsweise und Details zur Implementierung sind nicht bekannt bzw. für den Aufrufer irrelevant. Azure arbeitet genau nach diesem Konzept. Java-Entwickler haben die Möglichkeit, über Webservice-Schnittstellen auf Windows Azure bzw. die Azure Services zuzugreifen.

Dies soll anhand eines einfachen Beispiels aufgezeigt werden. In einem Java-Programm sollen Nachrichten, die über Azure Queues verschickt werden, empfangen werden. Listing 1 enthält die in Java geschriebene Programmlogik für Empfang, Verarbeitung und Löschen von Nachrichten aus Azure Queues.

In einer Endlosschleife wird eine über die Variable **`queue`** definierte Warteschlange eines Windows-Azure-Accounts (definiert über die Variable **`account`**) auf Inhalt geprüft. Hierzu wird ein **`HttpGet`**-Request an Azure geschickt. Wurde von Azure ein Ergebnis zurückgeliefert, wird dieses ausgewertet, d. h. die

```
public static void Sign(HttpRequestBase req, String account,
                        String key) throws Exception {
  SimpleDateFormat fmt =
    new SimpleDateFormat("EEE, dd MMM yyyy HH:mm:ss",
                         new Locale("en", "US") );
  fmt.setTimeZone(TimeZone.getTimeZone("GMT"));
  req.setHeader("x-ms-date",
              fmt.format(Calendar.getInstance().getTime()) + " GMT");

  StringBuilder sb = new StringBuilder();
  sb.append(req.getMethod().toUpperCase() + '\n');
  sb.append('\n');
  if (req.getFirstHeader("content-type") != null) {
    sb.append(req.getFirstHeader("content-type"));
  }
  sb.append('\n');
  sb.append('\n');
  sb.append("x-ms-date:" +
    req.getFirstHeader("x-ms-date").getValue() + '\n');
  sb.append('/' + account + req.getURI().getPath());

  Mac mac = Mac.getInstance("HmacSHA256");
  mac.init(new SecretKeySpec(Base64Coder.decode(key), "HmacSHA256"));
  req.setHeader("Authorization", "SharedKey " + account + ":" +
    new String(Base64Coder.encode(mac.doFinal(sb.toString().
      getBytes("UTF-8")))));
}
```

Listing 2: Methode Sign() zur Signierung der Azure-Aufrufrequests (Datei: worker.java)

enthaltenen Nachrichten werden aus dem gelieferten XML-Dokument ausgelesen und entsprechend auf der Konsole ausgegeben. Nachrichten werden durch das Auslesen nicht sofort gelöscht, sondern nur für einen festgelegten Zeitraum für weitere Nachrichtenempfänger zunächst unsichtbar gesetzt. Nach erfolgreicher Verarbeitung (in diesem Fall der Konsolenausgabe) müssen die empfangenen Nachrichten aus der Queue gelöscht werden, da Azure sie ansonsten wieder sichtbar macht. Die Löschung erfolgt durch Absenden eines `HttpDelete`-Requests.

Webservice-Requests an Azure müssen grundsätzlich signiert werden. Zur Signierung, die im vorliegenden Beispiel in der Methode `Sign()` erfolgt (s. Listing 2), wird ein Schlüssel benötigt, der beim Anlegen des Azure-Accounts generiert wird. Mit dessen Hilfe werden die Inhalte des Requests signiert.

Zur Implementierung dieses Beispiels wurden HttpClient-Bibliotheken von Apache (`commons-logging-x.jar`, `httpclient-x.jar` und `httpcore-x.jar`) sowie eine Codierungshilfsklasse von Source-code.biz (`Base64Coder.java`) verwendet.

Das Java SDK für .NET Services

Bei der Integration von verteilten Anwendungen und Services über Unternehmensgrenzen hinweg besteht häufig die Herausforderung, dass die einzelnen Dienste durch Firewalls und sonstige Sicherheitsmechanismen vor Zugriffen von außen geschützt sind. Inbound-Calls sind auf diese Dienste also nicht möglich. Möglich sind hingegen in der Regel Outbound-Calls. .NET Services schaffen mit ihrem Service-Bus in dieser Konstellation die Möglichkeit zu gesicherten Serviceaufrufen. Abbildung 2 skizziert die Funktionsweise.

Ein Service A, der von außen aufgerufen werden soll, kann sich bei den .NET Services registrieren. Diese weisen dem Service einen Endpunkt zu, über den der Service aufrufbar wird. Ein möglicher Aufrufer, Service B, kann über die .NET Services nun den Endpunkt von Service A ermitteln und diesen dann letztlich über die .NET Services aufrufen. Rein technisch sind Aufrufe von Service A Antworten auf die Registrierungsanfrage. Beide Services haben somit über Outbound-Calls eine Kommunikation initiiert. .NET Services bieten darüber hinaus auch das Mapping von Claims, d. h. Authentifizierungsinformationen, die Service B an Service A übergeben möchte, in ein Format, welches Service A auswerten kann.

Der Microsoft-Partner Schakra Inc. bietet ein Java Software Development Kit (SDK) für .NET Services an. Dieses setzt auf Metro (ein Open-Source-Webservices-Stack basierend auf Glassfish von Sun Microsystems) auf und stellt Bibliotheken bereit, die es erlauben, Java-basierte Webservices über .NET Services in der Cloud anzubieten, d. h. aufrufbar zu machen, bzw. aus Java-Clients entsprechende Cloud Services aufzurufen.

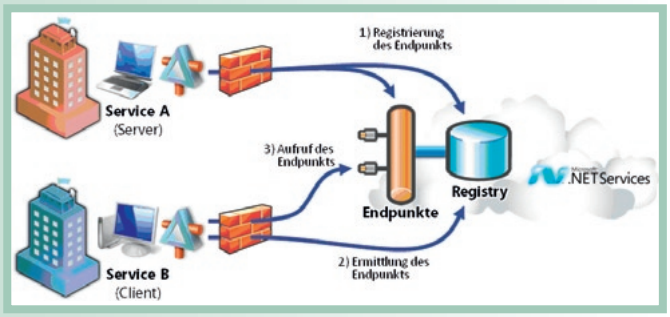

Abb. 2: .NET Services für die Anwendungs- und Serviceintegration

Betrieb von Java-Anwendungen auf Windows Azure

Zum Zeitpunkt der erstmaligen Veröffentlichung von Azure war es möglich, .NET-basierte Anwendungen auf Windows Azure zu installieren und zu betreiben. Die Vorgaben an die Softwarearchitektur sind hierbei nur sehr gering: Azure-Anwendungen müssen serviceorientiert aufgebaut sein. Dabei werden zwei Arten von Services unterschieden:

- *Web Roles* können von außen über den Azure-Loadbalancer aufgerufen werden und sind für die Bereitstellung der Aufrufschnittstellen (z. B. Webservices oder Weboberflächen) verantwortlich.
- *Worker Roles* sind für die Implementierung von Hintergrundprozessen verantwortlich und können von Web Roles über Azure Queues aufgerufen werden.

Der zentrale Azure-Verwaltungsprozess, die sogenannte Azure Fabric, überwacht die Ausführung von Anwendungen, die auf Azure betrieben werden.

Im März 2009 hat Microsoft Windows Azure eine entscheidende Funktion hinzugefügt: die Möglichkeit nativen Code auf Azure auszuführen. Damit wird es möglich, quasi beliebigen Code, der auch auf einem Windows-Server ausführbar ist, auch auf Azure zum Laufen zu bringen. In den zugehörigen Konfigurationsdateien von Web oder Worker Roles muss hierzu nur die Ausführung von nativem Code, wie in Listing 3 gezeigt, aktiviert werden.

Hierüber wird es nun möglich, auch ein Java-Laufzeitsystem zu instanziieren und dort eine Java-Anwendung auszuführen. Abbildung 3 skizziert den Aufbau einer Gesamtanwendung bestehend aus .NET- und Java-Elementen.

Nachdem Azure nicht direkt eine Java-Umgebung bereitstellt, muss diese von einer Worker Role instanziiert werden. In der zentralen Methode `Start()`, die von Azure aufgerufen wird, wird zu diesem Zweck das Java-Laufzeitsystem als neuer Prozess gestartet. Listing 4 zeigt die hierzu erforderliche, in C# geschriebene Aufrufsequenz. Das Java-Laufzeitsystem wird initialisiert und Ein- und Ausgabedatenströme werden auf Azure umgeleitet, sodass Ein- und Ausgaben über Azure abgewickelt werden können. In diesem Laufzeitsystem wird

```
<WorkerRole name="WorkerRole"
            enableNativeCodeExecution="true" />
```

Listing 3: Aktivierung der Unterstützung von nativem Code

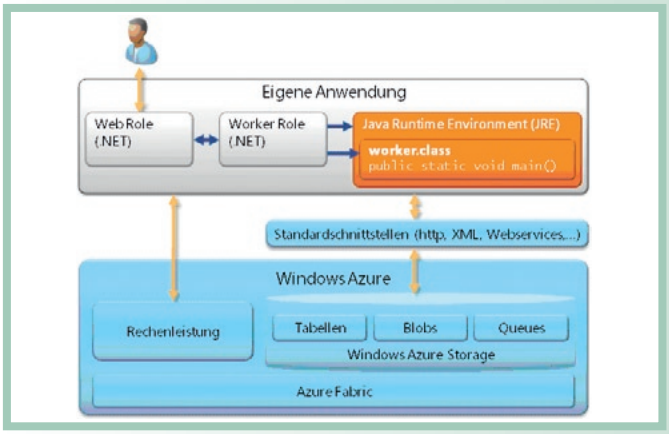

Abb. 3: Betrieb einer Java-Anwendung auf Windows Azure

dann eine Java-Klasse gestartet. Dies kann beispielsweise die in Java geschriebene Worker-Klasse sein, deren wichtigste Methoden bereits in den Listings 1 und 2 aufgeführt sind. In der in Java geschriebenen **Worker**-Klasse wird nun die Methode **main()** aufgerufen. Mit diesem Beispiel wird es also möglich, die Java-Klasse zur Auswertung von Queue-Nachrichten selbst auch auf Azure zu betreiben.

```
public override void Start() {
  Process p = new Process() {
    StartInfo = new ProcessStartInfo(Path.Combine(
      Environment.GetEnvironmentVariable("RoleRoot"),
      @»jre\bin\java.exe»), @»-cp lib;lib\*;. worker»)
    {
      RedirectStandardInput = true,
      RedirectStandardOutput = true,
      RedirectStandardError = true,
      UseShellExecute = false,
      WindowStyle = ProcessWindowStyle.Hidden,
      WorkingDirectory = Environment.GetEnvironmentVariable(«RoleRoot»)
    }
  };

  p.ErrorDataReceived += (sender, e) =>
    { RoleManager.WriteToLog(«Error», e.Data); };
  p.OutputDataReceived += (sender, e) =>
    { RoleManager.WriteToLog(«Information», e.Data); };
  p.Start();
  p.BeginOutputReadLine();
  p.BeginErrorReadLine();
  p.WaitForExit();
  throw new Exception(«Java Prozess beendet!»);
}
```

Listing 4: Instanziieren einer JRE und Ausführung einer Java-Klasse [Datei: WorkerRole.cs]

Fazit

Die Plattform Windows Azure stellt eine Reihe von Cloud Services zur Verfügung, die über Standardschnittstellen (http, SOAP, REST, WS*) aufgerufen werden können. Für Java-Entwickler bietet diese Plattform verschiedene Möglichkeiten:

- Azure Services können in eigenen Anwendungen genutzt werden,
- über die in Azure enthaltenen .NET Services können Java-basierte Anwendungen über die Cloud vernetzt werden und
- durch die Unterstützung von nativem Code können diese Java-Anwendungen letztlich auch selbst auf Windows Azure in der Microsoft Cloud betrieben werden.

Links

[Azure] Windows® Azure™ Platform, **http://www.azure.com/**
[J.NETServices] Java SDK for Microsoft .NET Services, **http://www.jdotnetservices.com/**

Holger Sirtl arbeitet bei der Developer Platform & Strategy Group der Microsoft Deutschland GmbH. E-Mail: holger.sirtl@microsoft.com.