Kundenportale – vom Intranet zum Internet mit Oracle APEX

Philipp Ostmeyer, Team

In einer Welt, in der die Digitalisierung in allen Lebensbereichen voranschreitet, stehen Unternehmen vor der Herausforderung, ihre Kundenkommunikation zu optimieren. In vielen traditionellen Unternehmen werden diese Funktionen noch immer über direkten, menschlichen Kundenkontakt abgebildet. Dafür benutzen wir viele Ressourcen und binden unsere Kunden an die Verfügbarkeit unserer Mitarbeiter:innen.

Die Digitalisierung im privaten Raum, welche durch amerikanische Anbieter auch hier Einzug gehalten hat, erzeugt neue Erwartungen hinsichtlich eines schnellen, transparenten und interaktiven Zugangs zu Informationen und Dienstleistungen. Jeder Mensch kann seine privaten Pakete heutzutage direkt auf dem Smartphone verfolgen bis hin zum präzisen Ort des Zustellers und stellt sich die Frage, warum dies nicht auch im Business-Bereich möglich ist.

In diesem Artikel möchte ich einen Weg beschreiben, wie wir unsere bestehende Infrastruktur ausbauen können, um diese Anforderungen zu erfüllen, ohne dass wir unsere bisherigen Systeme komplett neu aufstellen müssen. Dazu werden wir sowohl Klassiker der IT-Systemarchitektur (Linux, SSH und weitere) als auch Oracle APEX einsetzen.

Das Ausgangsszenario

In unserem Beispiel haben wir ein Lagerverwaltungssystem. Vielleicht benutzen wir Oracle Forms oder Oracle APEX, vielleicht setzen wir eine Drittsoftware ein, zum Beispiel Java-Server, um unseren Mitarbeiter:innen Zugriff auf unsere Business-Daten zu geben. Unsere Daten sind in einer Oracle-Datenbank gespeichert und unsere Business-Logik in PL/SQL geschrieben. Alles unsere Systeme sind ausschließlich in unserem lokalen Intranet erreichbar.

Ich möchte hier der Reproduzierbarkeit halber das Sample Dataset „Customer Orders“ als Basis nutzen (installierbar in Oracle APEX über SQL-Workshop → Utilities → Sample Datasets). Hier werden Artikel, Bestellungen aber auch Kunden- und Filial-Daten gespeichert. Zusätzlich zu diesem Schema werden noch ein Trigger und eine Sequenz erstellt, welche der Tabelle ORDER_ITEMS bei neuen Zeilen einen Primärschlüssel generieren, da dieser nicht direkt enthalten ist.

Wir wollen unseren Kund:innen bestimmte Funktionalitäten zur Verfügung stellen, ohne die bisherige Sicherheit unserer Architektur zu kompromittieren. Kund:innen sollen selbstständig Informationen zu ihren eigenen Bestellungen anschauen und neue Bestellungen tätigen können. Im Gegensatz dazu sollen keine Daten der Kunden- und Filial-Daten ausgelesen werden können.

Folgende Anforderungen müssen wir in unserer Entwicklung erfüllen:

- Der Entwicklungsaufwand soll gering sein und die interne Software unverändert bleiben.
- Nutzer:innen haben Zugriff, aber nur auf bestimmte Business-Objekte und -Prozesse.
- Nutzer:innen sollen keine Software zusätzlich installieren müssen.
- Daten- und Systemsicherheit bleiben erhalten.

Unsere Architektur

Wir wollen mit Kund:innen im Internet kommunizieren, also brauchen wir eine Maschine, welche aus diesem erreichbar ist. Dazu können On-Premise-Maschinen in der DMZ oder Instanzen in einer Cloud genutzt werden. Diese Maschinen müssen eine Verbindung über SSH vom Intranet ermöglichen und eine Oracle-Datenbank-Installation beinhalten. Hier kann ohne Probleme das kostenlose Oracle XE DB Release genutzt werden. Im Folgenden wird angenommen, dass alle genutzten Systeme auf Linux-Basis laufen; alle genutzten Programme haben Äquivalente auf Windows-Basis.

Als Grundlage dieser Kommunikation werden wir REST-Services nutzen. Diese werden wir mit Hilfe von Oracle Rest Data Services (ORDS) und Oracle APEX bereitstellen, weshalb die aktuelle Version installiert sein sollte.

Im Folgenden werden wir immer wieder von drei verschiedenen Systemen sprechen:

- Die bestehende, interne Datenbank mit Oracle APEX und ORDS-Installation (nachfolgend DB)
- Die oben beschriebene neue, externe Maschine (nachfolgend EXT)
- Eine im Intranet befindliche Maschine mit Verbindung zur DB und EXT (nachfolgend INT)

Die Systeme DB und INT können das gleiche System sein.

Aufbau eines REST-Services mit Oracle APEX

Durch REST-Services bieten Systeme Schnittstellen über HTTP an. Für jede Business-Operation werden wir einen

Name	Bind Variable.	Access Method	Source Type	Data Type	Comments
mdt	MDT	IN	URI	STRING	-

Abbildung 1: Eigenschafften des Mandanten-Parameters (Quelle: Philipp Ostmeyer)

Name	Bind Variable.	Access Method	Source Type	Data Type
mdt	MDT	IN	URI	STRING
x-api-key	KEY	IN	URI	STRING

Abbildung 2: Eigenschafften des API-Key-Parameters (Quelle: Philipp Ostmeyer)

```
with p as (
    select p.PDL_ARTIKEL_ID,
           count(*) paletten_anzahl
    from PDL_PALETTEN p
    group by p.PDL_ARTIKEL_ID
)
select ID,
       MDT,
       ARTIKELNAME,
       ARTIKELNUMMER,
       PREIS,
       GEWICHT,
       GEWICHTSEINHEIT,
       BESCHREIBUNG,
       (select "KATEGORIE" from "PDL_KATEGORIE" k where k.KATEGORIE_ID = a.PDL_KATEGORIE_ID) KATEGORIE,
       paletten_anzahl
  from PDL_ARTIKEL a
  left join p on a.ID = p.PDL_ARTIKEL_ID
--  where mdt = :MDT
--  and PDL_CHECK_KEY(:KEY, :MDT)='1'scott
```

Listing 1: Implementierung des Templates in SQL

```
create or replace function "PDL_CHECK_KEY"
    (p_key in VARCHAR2,
     p_mdt in VARCHAR2)
return VARCHAR2
is
begin
    if nvl(p_key,'#') != '348rz384f349' and nvl(p_mdt,'#') != '02'
    then
       RAISE_APPLICATION_ERROR (-20001, 'Nicht Autorisiert');
    end if;
    return '1';
end;
```

Listing 2: PL/SQL-Funktion zum Überprüfen eines API-Keys

```
vi /etc/ssh/sshd_config
```

Listing 3: Befehl zum Editieren des SSH-Daemons

```
/usr/bin/ssh {Username auf der EXT}@{IP der EXT} -R
8080:localhost:{Port des ORDS} -N -i {Pfad auf der INT zum SSH-Key des Users auf der EXT}
```

Listing 4: Befehl zum Aufbauen eines SSH-Tunnels

```
curl http://localhost:8080/ords/demos/lager/bestand?mdt=02 --
header 'x-api-key: 348rz384f349'
```

Listing 5: cURL-Befehl zum Testen der Verbindung

Endpunkt aufbauen. Oracle APEX und ORDS bieten im Zusammenspiel eine einfache Möglichkeit, diese Services zu erstellen, zu verwalten und in SQL beziehungsweise PL/SQL zu implementieren. Der ORDS-Server leistet hier die technische Bereitstellung der Services und die Datenbankverbindung sowie APEX geben uns eine einfache Verwaltungs-/Entwicklungsoberfläche. Um die sogenannten „RESTful Data Services" freizuschalten, können wir im APEX Workspace unter SQL-Workshop → RESTful Services dieses Feature für ein gewähltes Schema anschalten.

Für unseren ersten REST-Service wollen wir den Bestand unseres Lagersystems ausgeben. Alle unsere Endpoints werden in einem sogenannten Modul gebündelt, im Folgenden „lager" genannt. Der Modulname wird später in der URL des Endpoints genutzt. Innerhalb dieses Moduls erstellen wir ein Template mit dem Namen „bestand". Auch dieser Name ist Bestandteil der URL. Ein Modul symbolisiert eine lose Gruppierung, in diesem Fall aller Endpoints des Lagerverwaltungssystems; ein Template steht für Operationen auf einem bestimmten Business-Objekt.

Um hinter dieses Template Funktionalität zu legen, erstellen wir einen neuen, sogenannten Handler mit der Methode GET und dem Source Type „Collection Query". Hiermit legen wir fest, dass wir SQL nutzen möchten, um bei einer lesenden Anfrage unsere Antwort zu generieren. Als Source geben wir den in *Listing 1* dargestellten Code an.

Nach dem Speichern können wir unseren ersten Endpoint direkt ausprobieren, indem wir die weiter oben im Dialog angegebene URL aufrufen. Hier sehen wir, dass wir aus einem simplen SQL-Statement direkt eine JSON-Repräsentation

```
[Unit]
Description=SSH Tunnel
After=network.target

[Service]
User=root
WorkingDirectory=/root

Restart=on-failure
RestartSec=1

ExecStart=/usr/bin/ssh {Username auf der EXT}@{IP der EXT} -R
8080:localhost:{Port des ORDS} -N -i {Pfad auf der INT zum SSH-Key
des Users auf der EXT}

[Install]
WantedBy=multi-user.target'
```

Listing 6: Inhalt der Konfiguration des System-Services in der Datei apex_rest.service

```
systemctl start apex_rest
systemctl stop  apex_rest
```

Listing 7: Befehle zum Starten und Stoppen des SSH-Tunnel-Services

erzeugt haben, welche von den meisten REST-Service-Consumern unterstützt werden sollten. Wer sich die Daten im Detail anschaut, wird sehen, dass wir hier den gesamten Bestand unseres Lagers ausgeben. Dies ist für unseren Use-Case, ein öffentliches Kundenportal, viel zu undifferenziert und unsicher!

Als erstes soll der Endpoint eingeschränkt werden, sodass nur Bestände von bestimmten Kund:innen ausgegeben werden. Dazu kommentieren wir die vorletzte Zeile ein und erstellen einen Parameter mit den folgenden Eigenschafften *(siehe Abbildung 1)*.

Um diese Änderung zu testen, erweitern wir die URL um das Suffix „?mdt=02". Nun wird nur noch der Bestand des Mandanten 02 ausgegeben.

Der letzte Punkt, den wir jetzt noch angehen müssen, ist, für die Sicherheit zu sorgen. Die Einschränkung auf den Mandanten gibt uns hier keinerlei Bonus. Jeder, der Zugriff auf den REST-Service hat, könnte einfach einen anderen Mandanten eintragen; dies müssen wir verhindern. Dazu erstellen wir die in *Listing 2* ersichtliche Funktion.

Zusätzlich kommentieren wir die letzte Zeile des Codes oben ein und erstellen einen Parameter wie in *Abbildung 2* gezeigt .

Durch den Aufruf dieser Funktion können wir für jeden Mandanten einen API-Key definieren, welcher die Daten des jeweiligen Mandanten vor Fremdzugriff schützt. Diese Funktionalität können wir testen, indem wir „&x-api-key=348rz384f349" an die URL hängen.

Wenn alle Tests abgeschlossen sind, ändern wir die Source-Type-Eigenschaft des x-api-key-Parameters auf „HTTP HEADER". Dies ist die Standardeinstellung vie-

ler Services, welche wir auch bei unseren Services einhalten können.

Damit ist die Konfiguration der DB-Maschine abgeschlossen und wir können uns dem Tunnel zuwenden, welcher das Internet mit unserem Service verbindet.

Aufbau des SSH-Tunnels

Auf der INT-Maschine werden wir einen neuen systemd-Service erstellen, welcher einen SSH-Tunnel zur EXT-Maschine aufbaut. Zum Testen werden wir dazu erst einmal ein einfaches Bash-Command aufbauen. Eine Sicherheitseinstellung sollten wir vorher noch überprüfen. Auf der EXT-Maschine können wir den SSH-Daemon in der folgenden Datei konfigurieren, hierfür sollten SUDO-Rechte benötigt werden *(siehe Listing 3)*.

Hier suchen wir die Zeile beginnend mit „GatewayPorts“. Diese sollte auf „no“ gestellt sein, da sonst Anfragen von anderen externen Maschinen über die EXT-Maschine an die INT-Maschine weitergeleitet werden können. Diese Einstellung erlaubt keinerlei Weiterleitung von Aufrufen, welche nicht direkt von der EXT-Maschine selbst kommen.

Das folgende Bash-Command baut den Tunnel auf *(siehe Listing 4)*.

Wenn wir dieses Command ausführen, bauen wir direkt einen Tunnel auf. Diesen können wir mit dem in *Listing 5* dargestellten cURL-Befehl testen.

Wenn dieser Test-Daten im JSON-Format zurückliefert, können wir den Tunnel in einen Service gießen. Dazu erstellen wir eine Datei „/etc/systemd/system/apex_rest.service“ und füllen diese mit dem in *Listing 6* dargestellten Inhalt.

Diesen Service können wir nun nach Belieben lokal, sprich vom INT gesteuert, starten oder abschalten *(siehe Listing 7)*.

Damit haben wir einen SSH-Tunnel aufgebaut und können uns jetzt um das Internet- Portal kümmern.

Konsumieren des REST-Services

Zum Konsumieren des neuen REST-Service können wir jedes Framework nutzen, welches dies erlaubt. In diesem Artikel werden wir dazu Oracle APEX nutzen.

In dem Workspace der EXT-Maschine erstellen wir eine neue Application und navigieren im „Shared Components“-View zum Punkt „REST Data Sources“. Hier wollen wir eine neue Datenquelle anlegen.

Im ersten Dialog wählen wir „From Scratch“ aus und im zweiten „Simple HTTP“, geben ihr einen Namen und fügen die URL ein, welche wir vorher über den cURL-Befehl getestet haben. Danach gehen wir weiter durch den Dialog bis zum Punkt Authentication. Hier schalten wir diese an, definieren einen neuen Credentials-Eintrag und geben den HTTP-Header „x-api-key“ mit unserem Schlüssel als „value“ an. Zum Abschluss lassen wir APEX die Quelle mit Hilfe des „Discover“-Button analysieren.

APEX wird durch diesen einen Dialog mehrere Dinge in unserer Applikation und dem Workspace ablegen. Im Workspace wird der API-Key als Credential gespeichert. Zusätzlich wurde der Parameter MDT erkannt und explizit angelegt, hierdurch können wir diesen im Low-Code-Stil von APEX einfach in die Application setzen.

Wenn wir den API-Key dynamisch füllen wollen, würden wir diesen hier als Parameter anlegen und nicht über den Anlage-Dialog. Dadurch können wir zum Beispiel die Nutzer:innen beim Einloggen nach diesem fragen und die gesamte Session damit autorisieren.

Wo wir nun die Definition hinter uns haben, ist der technische Teil des REST-Service-Benutzens hinter uns. Alles, was wir noch machen müssen, ist, eine neue Seite zum Beispiel mit einem Interactive Report zu erzeugen und als „Data Source“ die eben erstellte Quelle anzugeben. Alles nachfolgende funktioniert genauso wie wir es in APEX gewohnt sind.

Wir haben also unsere interne Applikation ohne zusätzlichen systemischen Aufwand unseren Kund:innen zur Verfügung gestellt. Der hier dargestellte Weg kann für viele verschiedene Operation, wie das Erstellen oder Bearbeiten von Datensätzen, erweitert werden; auch hier können wir uns auf die Hilfe von Oracle APEX verlassen.

Über den Autor

Philipp Ostmeyer ist Consultant und Projektleiter im Software- und Consulting Bereich der TEAM GmbH. Er beschäftigt sich vorwiegend mit den Themen Oracle APEX und Java Backend-Anwendungen, zu welchen er auch regelmäßig Schulungen und Vorträge hält.

Philipp Ostmeyer
po@team-pb.de