Building Apps for the Intelligent Cloud Day 3 – DEMOS

# service bus - sample use case scenario

Requirement: Visual Studio 2013 or higher, Azure SDK

Description: The demonstration aims at demonstrating how Service Bus endpoints can be invoked even from small devices like Windows Phone (because they are REST based).

In the demonstration, we will access an unpublished database via another process (console app) running on same VM where Database is and published using Service Bus.

Demo Time Estimation: 15 Minutes

What to demo:

1. Create a service bus queue in azure as shown here:

<https://azure.microsoft.com/en-in/documentation/articles/service-bus-dotnet-how-to-use-queues/#how-to-receive-messages-from-a-queue>

Pick only the relevant potions (do not build client etc.)

1. Create a simple console/winform application that accepts message from queue as shown here:

<https://azure.microsoft.com/en-in/documentation/articles/service-bus-dotnet-how-to-use-queues/#how-to-receive-messages-from-a-queue>

1. In the above application, write code to fetch some value from database and return.

Create a windows Phone Application (you can use any platform of your choice) that calls service bus endpoint as described here:

<http://www.4sln.com/Articles/sending-message-from-windows-phone-8-1-to-azure-service-bus>

# Demo – logic app

Requirement: Azure Subscription

Description: Demo how to create a simple logic app in Azure

Demo Time Estimation: 10 Minutes

What to demo:

* Show how easy it is to create a logic app from the Azure Portal.
* For the simplicity of the demo, you can choose to go with the designer approach, so that the audience can see the no-code feature through which they can design a workflow
* A simple recurrence trigger, with a HTTP action and an Office 365 Email actions would be good to start with
* Optionally, the concept of web hook triggers to trigger a logic app flow from an API app can be demonstrated. Refer

Reference video Source:

<https://azure.microsoft.com/en-in/documentation/articles/app-service-logic-create-a-logic-app/>

# App service hybrid connectors

Requirement: Azure Subscription

Description: Demo how you can connect to on-premise application and data from API apps and in-turn integrate with systems from other networks. These API apps can then be used within a logic app.

We will be creating a simple Web API, host it in Azure API App, and retrieve data from on-prem sql server.

Demo Time Estimation: 15 Minutes

What to Demo:

* Create a simple ASP.NET Web API that connects to SQL Server to retrieve and send data
* Publish this to Azure API App directly from visual studio
* Establish a hybrid connection by following these simple steps from the portal as described [here](https://azure.microsoft.com/en-in/documentation/articles/web-sites-hybrid-connection-connect-on-premises-sql-server/)
* Ensure that the necessary outgoing ports in the firewall as described in this [article](https://azure.microsoft.com/en-in/documentation/articles/integration-hybrid-connection-overview/)
* Call the API app endpoint from fiddler or from any client application and show how the data from the on-premise SQL Server is made available to the API APP in Azure
* Finally, call this API endpoint from a HTTP action in your logic app developed in the previous demo to demonstrate a full integration scenario and a workflow.

# app service authentication

Requirement: Visual Studio 2013 or higher, Azure subscription

Description: Introduce app service authentication as a no code feature to implement authentication out of the box for different providers.

Demo Time Estimation: 10 Minutes

What to Demo:

* Follow the steps from this [article](https://azure.microsoft.com/en-in/documentation/articles/app-service-mobile-how-to-configure-active-directory-authentication/) to demonstrate the Express and Advanced mode for configuring Azure AD authentication and authorization.

# azure ad auth – web app and native app consuming an api

Description: Demonstrate how to set up Azure AD authentication from ASP.NET MVC application within visual studio

**What to Demo – Web App**:

* Create a new ASP.NET MVC application in visual studio
* Go to Change Authentication during the template choosing dialogue box and choose Cloud SSO – Single organization.
* This will automatically register your app against the given Azure AD tenant.
* Open the Azure AD tenant and explain what happened behind the scenes
* Debug the MVC app and show how it is getting redirected to Azure AD for authentication

Resource: <http://www.asp.net/identity/overview/getting-started/developing-aspnet-apps-with-windows-azure-active-directory>

**What to Demo – Native app consuming a web API:**

* Follow the exact same steps as above, but create a Web API instead of Web App
* Create a simple console application from visual studio, which will act as the client for the web API
* Register a native application to your Azure Active directory tenant
* Provide access for this app to the Web API registered by performing step 1
* Add authentication code in the console application by using ADAL libraries
* Run the console app to show how the Azure AD asks for username and password through a browser based pop-up.

You can use the ready-made samples available here <https://azure.microsoft.com/en-in/documentation/articles/active-directory-code-samples/#native-application-to-web-api>

# azure ad multi-tenant authentication

Description: Demonstrate how a web application, that is used by multiple customers have their respective user stores in different Azure Ads (tenants)

The detailed list of steps to be performed and the application code for this demo can be found here

<https://azure.microsoft.com/en-in/documentation/samples/active-directory-dotnet-webapp-multitenant-openidconnect/>

# Deploying your mean stack application to azure

* the first thing we'll do is we'll set up MongoDB in the cloud
* we'll configure and deploy an Express application
* So the options that you have in the cloud are: standup your MongoDB server via VMs; there's other, like, technologies such as Docker and things like you can research; that works on any cloud provider; Azure also has a fully-managed option-fully-managed database option-which is MongoLab, and it provides MongoDB database as a service, PaaS or DBaaS-database as a service.
* So what it does: it allows us for a fully-managed database-you don't have to worry about patching or update the MongoDB server-as well as you don't have to really worry about scaling that server as well-it'll just scale automatically. You just pay the monthly fee, and for a sandbox instance, it won't cost us anything, and we can mess around with it as much as we want and get our application working live.
* "Add new," and we'll click on Marketplace, and under all these services-under here-you'll see MongoLab. There's also MongoDB; so MongoLab and MongoDB are two companies that offer MongoDB as a fully-managed service. One is by actual... the MongoDB Company-MongoDB Inc-and they are the maintainers of it. And then you also have MongoLab, which is the second option. The reason why I'm using MongoLab today and not MongoDB is because the minimum cost for MongoDB fully-managed is, like, four hundred dollars a month, and MongoLab has a free sandbox instance that we could use.
* We'll call this "chirp-app”.
* Create a new Azure website-go to plus-and so this be the URL to chirp. chirp-azure.azurewebsites.net
* Make some changes in the code: Instead of being hard-coded to connect to the local database for MongoDB, connect to the special MongoLab connection string.
* what you really should do is just do environment variable-set that up-and put the environment variable in your Azure website, and put it on your local, but just keep it out of the source, so that when you do the push, and you share the code around, you don't have those keys just floating around in there.
* In Website on Azure -> set up deployment from source control-right here
  + git remoteadd orgn http://xxx
  + git push orgn master
* Once its successful; browse it