Microsoft

First Response Demo

Last updated: 6/5/2017

|  |  |
| --- | --- |
| **Microsoft Contact**  Tara Shankar Jana  Sr Product Marketing Manager  Data Platform & IoT Mktg  [tarasha@microsoft.com](mailto:tarasha@microsoft.com) |  |

# Introduction

First Response Online is a communication and collaboration platform built to support first responders. It lets police officers, fire fighters, and paramedics share critical data with each other in near real-time. It supports iPhone, iPad, and PC and integrates with computer aided dispatch and GPS tracking. Units in the field can update their status, complete traffic stops, and even query state & federal databases all without using the radio.

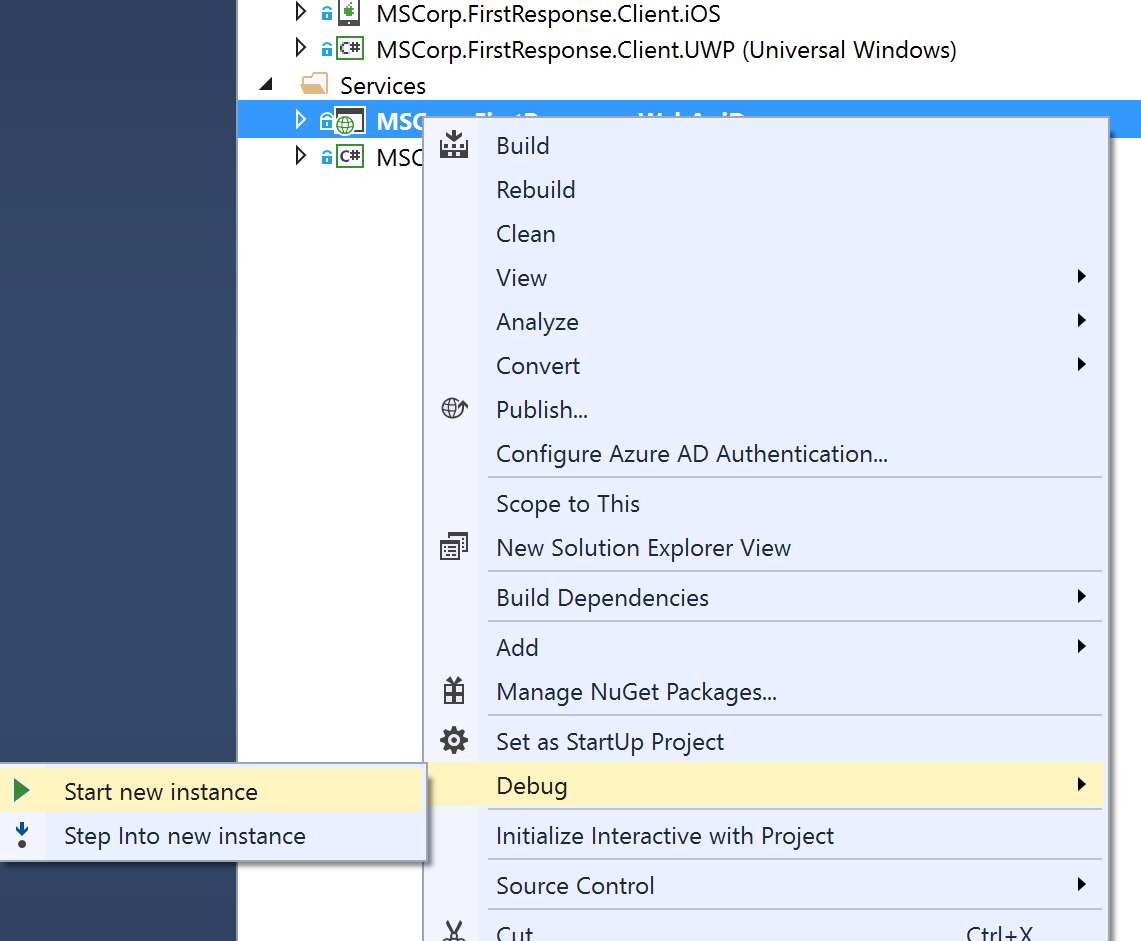
First Response Online is powered by Azure. In order to build a business that handles major life-threating incidents, they not only rely on Azure’s proven scalability and availability but also its breadth of features to get to market quickly. By using services like App Service, Azure SQL, Cosmos DB & Azure Search, First Response Online can instead focus on empowering first responders instead of maintaining infrastructure.

In the following demo we will examine some of the technologies that enable cloud born applications like First Response Online .

# Setup

Before starting the demo, follow the deployment instructions that you could find at .\Deploy\Deploy.md

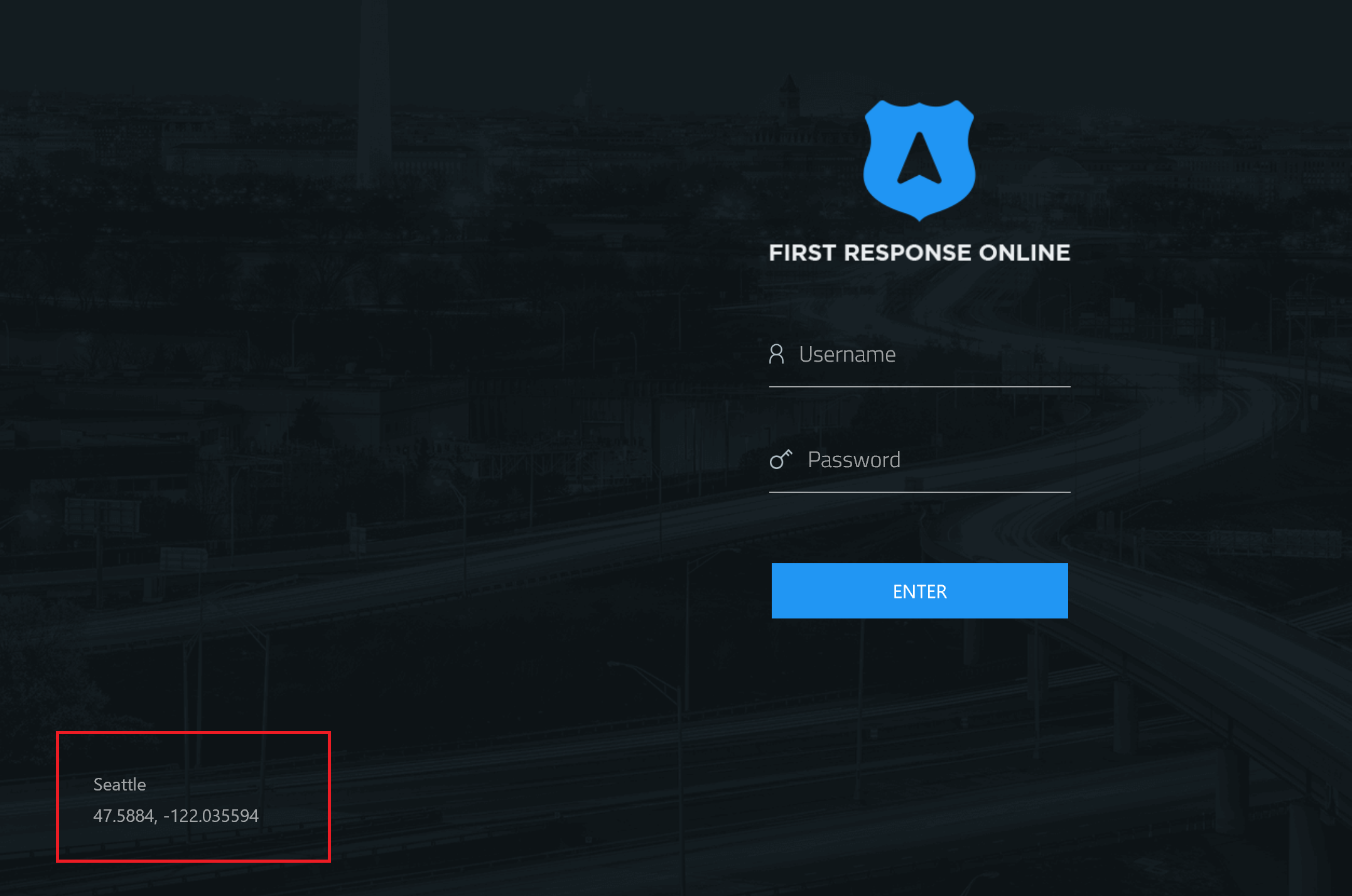
Once completed, you will be ready to open MSCorp.FirstResponse.sln and run locally **MSCorp.FirstResponse.WebApiDemo**



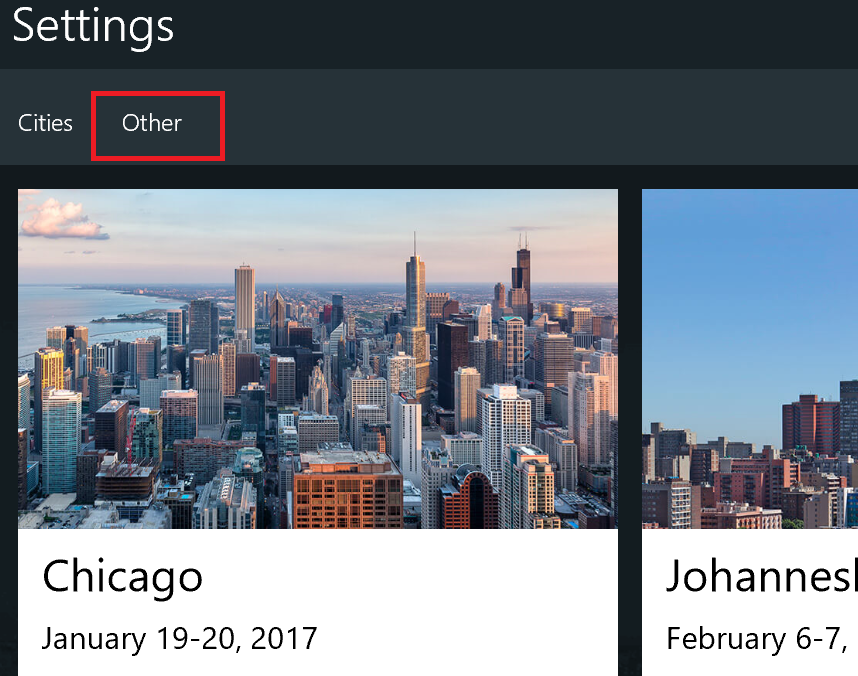
This should launch the API at <http://localhost:50002/>

Now that we have the API ready you can run the client application you like to start the demo. In the following instructions we’ve used UWP app as sample.

By default, the app will load Seattle data. If you are in a different city or would like to use different data click in the City name on login page to access the settings page:



You will be able to change city or set different API endpoint in Other tab:



Use “Mock Services” flag if you don’t want to get data from the API. The app will then use his own datasets locally.

In this document, there are references to a CityID number. Use the one from the following list based on your city choice:

* 24 | Chicago
* 25 | Johannesburg
* 26 | Frankfurt
* 27 | Washington, DC
* 28 | Singapore
* 29 | Bangalore
* 30 | Milan
* 31 | Amsterdam
* 32 | Birmingham
* 33 | Copenhagen
* 34 | Seoul
* 35 | Seattle

# Contents

|  |  |
| --- | --- |
| Demo Story Elements | Minutes |
| *Easy to Build*  Illustrates how easy it is to build a modern SaaS application on Azure with technologies like Azure SQL database and its elastic capabilities. | 5 |
| *Evolve and grow*  Shows how the breadth of Azure capabilities can be used to expand application capabilities sets using technologies like Cosmos DB and Search. | 5 |
| *Innovate and differentiate*  Turn data into actionable insight using technologies like Power BI and its ability to connect and consume from multiple different Azure data services. | 5 |

## Story 1: Easy to Build

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open the **First Response** app from the Start menu. 2. Enter Username : jclarkson 3. Enter Password : jclarkson 4. Click Right arrow | First Response is a multi platform app built with Xamarin.Forms. It supports three platforms – from Universal Windows Platform, to iOS and Android tablets – and provides first responders with key features like offline, data security, GPS, first-class touch support, and more.  Let’s login as one of the officers on duty. |
|  | 1. Wait for the **dashboard** to load. 2. Click on the **Responders** tab. 3. Click back to the **Incidents** tab. | Police officers in the city of Sammamish can see a visual representation of active incidents (Incidents tab). They can also see their current location (centered blue icon).  They can also see other units in the field (Responders tab). Police data is isolated; however, they can opt to securely share with neighboring agencies and counties enabling better co-operation. |
|  | 1. Click on the **speeding report** (red icon in the lower right corner or the red incident on the left hand panel). | A speeding car has been reported in Sammamish. We can see the details of the call alongside the map and see that sensitive information has been masked from officers (like John Clarkson). Full visibility is restricted to supervisors. |
|  | 1. Click **navigate** button that appears in a popup window. 2. In theleft panelyou will see the latest incident information while our position is moving to the incident location. | Let’s respond to the call using the app and mark ourselves on route. GPS directions are provided from our current location and the map updates our progress in real-time (for both us and other users).  Updates from dispatchers are also available immediately. It looks like the speeding driver has collided with the curb and potentially sustained an injury. |
|  | 1. Switch to the **Azure Portal** in Edge. | At this point, it’s time to talk about how Azure makes building apps like this one easy. |
|  | 1. Open the **Elastic database pool**. 2. Point out the **eDTU** and **GB** tiles. | By using an elastic database pool, SaaS services like First Response can provide burstable performance and reduce costs all while ensuring data isolation. The throughput units and storage are shared across all their customer databases rather than being tied to individual customers. |
|  | 1. Click on **Configure pool** in the **Settings** blade. | As First Response grows – or even just for busy periods of the year – it can easily allocate additional capacity on demand. |
|  | 1. Open the **list** of databases in the pool. 2. Click on the **police** database. | Here we can see separate, isolated databases for the fire, police, and ambulance services. While they are all in the pool, we can manage them like normal databases. |
|  | 1. Click on **Pricing tier (scale DTUs)** under the **Settings** blade. 2. Click **View all** to see the full set of pricing options. | If needed, we can allocate additional dedicated capacity to individual databases and scale them vertically on demand. In the Frist Response Online we may want to upscale the Police DB proactively to allow for increased traffic during an event. |
|  | 1. Switch to **SSMS**. 2. Open **PoliceDynamicDataMasking.sql** (read only; don’t run)   **Note:** These SQL scripts can be found in the deployment folder. | Azure SQL also allows us to protect data at the database level through features like dynamic data masking. Let’s first see how easy it is to set up. We simply provide a mask to the column in the table and then specify which roles can unmask the data. |
|  | 1. Open **PoliceDynamicDataMaskingQuery.sql** 2. Select the **police** database. 3. Press **Execute**.   "UserName" =[YourPrefixName]-admin  "Password" =f1r4tR3sp0Ns1 | Next, let’s see it in action. This script runs a query to retrieve incident data twice: once as an officer, and again as a supervisor. Look how the database returns masked data for users without supervisor permissions. |
|  | 1. Open **PoliceRowLevelSecurity.sql** (read only; don’t run) | Azure SQL also allows us enforce security at a row level based on user permissions.  To use RLS, we need to set up a predicate function. This will be automatically appended to the WHERE clause of any queries run on the table. It allows users in the supervisor role to see all data and non-supervisor users to see only the data in the regions they are assigned to patrol.  The security policy then connects the predicate to the table we want to protect. |
|  | 1. Open **PoliceRowLevelSecurityQuery.sql** 2. Select the **police** database. 3. Run the **first** query. 4. Run the **second** query. | Let’s look at John’s assigned regions. He can access information about the zip codes where he works. In this case, it’s just two.  When a query for all incident data is run as John, you can see only information from those zip codes is returned. |
|  | 1. Run the **third** query. 2. Run the **fourth** query. | Now let’s look at Ben’s assigned regions. He works in a different area to John. When he queries the incidents table, he can only see data from is region. |
|  | 1. Run the **fifth** query. 2. Run the **sixth** query. 3. Run the **seventh** query. | Now let’s look at Evan’s assigned regions. But we can see through the next query that he’s in a supervisor role. When he queries for incidents, he can see all the data. |
|  | 1. Run the **eighth** query. | Finally, let’s run the query as ourselves (active directory user). Our user hasn’t been granted any rights to this table so it is not available to us. |
|  | 1. Switch to **PowerShell**. | While keeping customer data in separate databases – combined with the security features discussed here – ensures a high level of data protection and isolation, it obviously increases complexity. We’ve already seen how Elastic Pools help manage cost & performance; now we’ll look at how Elastic Database helps us manage and query *N* databases as if they were one. |
|  | 1. Open **ShardMapManager.ps1** 2. Run steps #1 and #2.   **Note**: Replace [YOUR\_PREFIX] with the demo prefix used at database creation time. These can be also be retrieved from Azure portal or the demo application web.config file.  **Note:** These scripts can be found in the deployment folder. | Let’s connect to the master database and retrieve the list of shards. The ‘Value’ property is a piece of business information that determines which shard should be used to store the information. This mapping can evolve over time as clients come and go.  For First Response this ‘Value’ is the key which represents the service / departments type. We have one for Police, Ambulance and Fire.  We are using a range shard map here, because of this in the future we could easily have two services stored within one database. |
|  | 1. Let’s stay in the same ps1 file and execute Step #3 | Here are some of the other commands available (note: add & remove). Additionally, Azure services like Elastic Jobs (to run scripts across all databases) and Elastic Query (to return a single result set for a query over all shards) also help make management simple. |
|  | 1. Switch to **Visual Studio**. 2. Open **MSCorp.FirstResponse.WebApiDemo.sln** which is in the demo folder | Now that we’ve reviewed the management tools, let’s look at what elastic databases mean for application developers. Today we’ll be looking at raw ADO.NET; however, it works just as well with Entity Framework. |
|  | 1. Open **IncidentController.cs** and place a breakpoint on the first line of **GetAllIncidents**. 2. Scroll the end of the file and show the **IncidentQuery**. 3. Press **F5**. | Let’s run through a simple query against an elastic database. We’ll spin up the service locally and debug through it. As you can see the query is pretty simple; but how much more complicated is it to run this one query across our police, fire, and ambulance databases? |
|  | 1. Let’s go back to **PowerShell** and open **ShardMapManager.ps1**. 2. Run step #4 3. Wait for the breakpoint to be hit and switch back to Visual Studio. | We’ll call the service using PowerShell.  The port (i.e. XX) will be shown in the system tray under IIS Express. |
|  | 1. Step over the **connection string**.   (F10, or Debug menu, Step Over option)   1. Step over the **shards**. (F10) 2. Step into **QueryHelper**.   (F11, or Debug menu, Step Into option) | First, we get the connection string. It’s just like a normal SQL connection string except without the database details (as we don’t know which shards we’re working with yet).  Next, we’ll get the shard list. This is retrieved from the master shard database and helps the application understand where data list for each customer. In this case, we want to query all shards (but you can see us querying individual shards elsewhere). |
|  | 1. Step over until you reach **MultiShardConnection**. (F10) 2. Press **F5**. 3. Switch back to the powershell window and view results (notice Ambulance and Fire results being returned)   **Note** : Police are not being returned as we have RLS enabled and we are connecting as a dbo service account in this project. | As you can see, it’s just like querying using a SqlConnection, SqlCommand, and a SqlReader. They inherit from the same interfaces and share the same query pattern. The queries to individual databases – and the aggregation of results – is hidden from us entirely.  Let’s have a look at how we would query an individual database. |
|  | 1. Open **IncidentController.cs** and place a breakpoint on the first line of **GetAmbulanceIncidents**. 2. Run Step #5 from file **ShardMapManager.ps1** 3. Wait for the breakpoint to be hit and switch back to Visual Studio. | Now lets call the ambulance end point to load only Ambulance incidents. |
|  | 1. Step over the **connection string**.   (F10, or Debug menu, Step Over option)   1. Step over the **shard**. (F10) 2. Expand out the **Shard.Location** show the database we are hitting is called **XXX-ambo** 3. Press **F5** | Notice that the setup of the method is the same, we are re-using the same query and connection string to the master, I.e we are not adding an explicit where clause.  Have a look at how we load the shard, previously we executed the query across all shards, now we use a single shard based on a Key.  In First Response, we have each shard setup to store incidents based on the department type / service as seen previously. In this case we are achieving the filtering based on the shard we issue the query too.  At this point, we’ve seen how you can quickly create modern applications using the latest SaaS, security, and scalability features available with Azure SQL. But what’s next for First Response? |

## Story 2: Evolve and grow

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. **View** the zoomed in incident map view. | Police Officer John Clarkson arrives at the scene. The application provides a zoomed in incident-level view of the surrounding area. |
|  | 1. **Notice** the highlighted polygon within the application showing previous incidents within. | The First Response solution used to stop here. Now, with the power of Azure Cosmos DB, they have evolved it to support e-ticketing.  Prior to exiting the vehicle, it’s important for John to get some context of previous tickets within the area, this will assist in determining whether a ticket will be issued.  Cosmos DB is used as the data store for tickets issued by first responders. It supports a range of powerful query operations including geospatial indexing and querying, We can use this capability to retrieve previous tickets within a bounded area. It’s as simple as issuing a query to Cosmos DB. Let’s take a look. |
|  | 1. Switch to the **Azure Portal** and navigate to the Cosmos DB blade. 2. Click on **Query Explorer** icon in the header. | We issue a geospatial query from within the application to display the incidents. Let’s look at how to write one.  Using the Query Explorer you can easily work on your query and validate the results. |
|  | 1. Run the default query: **SELECT TOP 100 \* FROM c** | As you can see these queries are constructed using a familiar SQL like syntax. You can also issue commands to filter, order and simple TOP commands.  Each ticket has a Location object within its document. This is specified using the GeoJSON format.    We will use this information to run our geospatial query. |
|  | 1. Copy the below query into **Query Explorer** and **Run query**.   SELECT x.Id, x.Type, x.Location FROM x  WHERE  ST\_WITHIN(x.Location, { "type": "Polygon", "coordinates": [[ [-122.022430, 47.584540], [-122.022217, 47.580847], [-122.014425, 47.577210], [-122.003700, 47.577954], [-122.003914, 47.584410], [-122.022430, 47.584540] ]]}) | Using the geospatial function ST\_WITHIN we can easily determine if the Location of the incident is within the polygon defined by a set of coordinates.  But what if we wanted to find the closest incidents (rather then look at incidents within a polygon)? |
|  | 1. Copy the below query into **Query Explorer** and **Run query**.   SELECT x.Id, x.Type, x.Location FROM x  WHERE ST\_DISTANCE(x.Location, {'type': 'Point', 'coordinates':[-122.022430, 47.584540]}) > 10 | Using the geospatial function ST\_DISTANCE we can filter incidents which have occurred close to a location.  There are two other geospatial methods currently available: ST\_ISVALID and ST\_ISVALIDDETAILED. These return a validation result of your GeoJSON points / polygon. |
|  | 1. Switch back to the **First Response** application | Cosmos DB has allowed the First Response app to quickly provide important context for the responding officer. They can identify whether this is a high risk area – e.g. for accidents – as part of deciding whether to write a ticket. |
|  | 1. Click **Identify**. | The officer now exits the vehicle and approaches vehicle and driver. Based on the prior toast notification, we know there has been a minor accident and that vehicle driver is in shock.  Responding officer asks: “Are you OK sir? Can I please have your name?” |
|  | 1. Type in “Joe” to **Identity Search** textbox and click **Search**. | Driver responds with what the officer hears as “Joe.”  We can make use of the power of Azure Search to assist in identifying the driver of the vehicle with a similar sounding name. |
|  | 1. Select **Joao Casqueiro**. 2. Click **Done**. | The officer see that the driver in the image is the same as seen in the application, this assisting in confirmation with the name of the person being pulled over is Joao.  Let’s see how this is achieved using Azure search. |
|  | 1. Switch across to **Visual Studio**. 2. Open **MSCorp.FirstResponse.WebApiDemo.sln**. | Now that we have seen how this feature works, let’s look at how it’s implemented. Let’s run through a simple query against an Azure Search index. We’ll spin up the service locally and debug through it. |
|  | 1. Open **PersonController.cs** and place a breakpoint on the line of **Search** whichsets the **scoringParam** variable. 2. Press **F5**. | We are using the Azure Search client library delivered through NuGet. Alternatively, we could make use of the Rest API.  As you can see with just a few lines of code we can issue a query to return person results. | |
|  | 1. Open **PowerShell**. 2. Run Step #6 from file **ShardMapManager.ps1** 3. Wait for the breakpoint to be hit and switch back to Visual Studio. | Let’s trigger our service using PowerShell. | |
|  | 1. Switch back to **PersonController.cs**. | Within our search results, we want to return people whose home location is closest to the incident higher up the list.  This ordering is achieved using a scoring profile. |
|  | 1. Open **Deploy\AzureSearch\PersonIndex.json**. 2. Scroll to **“scoringProfiles”**. | Using scoring profiles we can quickly weight particular documents based attributes of the document to be returned above others.  In this case, we are using a referencePoint (Officer’s location) compared to the fieldname (HomeLocation) and boosting the score by a factor of 35. |
|  | 1. Switch back to debugging code 2. Press F5 | Here we are passing in the officer’s location as a scoring profile parameter. This is what Azure Search uses to compare against the HomeLocation of the person. | |
|  | 1. Switch back to PowerShell window. 2. Run the following command **$searchResults.Results** | This determines the score. By default, results are ordered with the highest score first as you can see in the results list.  Also note that there are people who have been returned who have a name other than “Joe”. Using the power of Azure Search we are able to trigger a Phonetic Search across the name.  Phonetic searching enables searching based on how a word sounds, not how it’s spelled. This is configured as part of the index. Let’s have a look at the index structure. | |
|  | 1. Open file **Deploy\AzureSearch\PeopleIndex.json** and scroll to analyzers. | Enabling phonetic search is a simple two-step process.  First: activate a custom analyzer (in preview) which lets the index know how about phonetic search capability. Note: see “phonetic” token filter. | |
|  | 1. Scroll to **FirstName** / **LastName** columns in **Deploy\AzureSearch \PeopleIndex.json**. | Second: link the fields which you want to apply phonetic search across. See how we define the  “analyzer” as “phonetic” on the FirstName and LastName columns.  Having completed these two steps, we have now enabled phonetic search across a search index. | |
|  | 1. Scroll to **HairColor** / **Sex** columns in **Deploy\AzureSearch\PeopleIndex.json**. | As we don’t know the user’s full name and phonetic search may return a large range of results, we need a way to show categories to help the user identify the driver of the vehicle.  This is easily achieved using another out of the box feature: facets. Facets will help the responding officers of the First Response application to identify the user when they are unable to effectively communicate.  To use facets, we need to first tell Azure Search that it is a column which can be “facetable”. | |
|  | 1. Switch back to **PersonController.cs in** Visual Studio. | Also, when the search is issued we need to tell Azure Search we want the results returned with facets.  We can do this by sending the columns to be faceted with the search in the SearchParameters. |
|  | 1. Switch back to PowerShell window. 2. Run the following command **$searchResults.Facets** | Notice that we are returning only results being faceted by “Sex”. | |
|  | 1. Switch back to the **First Response** application. 2. Click over to **Tickets** tab. 3. Click on “**Select Ticket**” drop down. 4. Scroll through the **Ticket types** showing the different ticket entry forms. | We’ve seen how we can use Azure Search to identify the driver of the vehicle. Next, let’s look at writing a ticket.  First, we need to determine the ticket type. As you can see for each ticket type, we have several different ticket categories and related attributes. | |
|  | 1. Select **Traffic**. 2. Driver:Joao Casqueiro. 3. LicenseNumber: ACK406. 4. Details: Reckless Driving. 5. Click **Submit**. | Cosmos DB’s schema-less design allows us to store a range of different tickets types in one database. For First Response, this allows new ticket types & structures to be added or even extended for certain regions without having to modify the structure of the database. This empowers First Response to rapidly iterate as requirements and customers change.  It also automatically indexes the documents and supports powerful query operations like geospatial (as previously covered). |

## Story 3: Innovate and differentiate

Supervising police officers require ‘up to the minute’ information about incidents occurring across a city. Based on this information he or she is able to take intelligent actions. Hence it is important that the information is current, complete and presented in a way that is easy to understand and interact with.

With Azure SQL in-memory OLTP, using a Rest API connection to Power BI, the information presented on dashboards is current, enabling officers to react in real-time.

So the supervisor has all of the required information available to them in one place, it’s important to be able to combine data from multiple sources. PowerBI does this, with a range of native connectors.

With Power BI, anyone can create rich and compelling stories that perfectly visualize data and share insights in real time. It is a key component of Microsoft’s market leading BI and Analytics platform, as acknowledged in this year’s Gartner magic quadrant.

Evan Dodds uses the Supervisor Dashboard, built using Power BI, to gain an understanding of the current situation across the city. He sees summary information about incidents and their location.

He will observe that the current average response time (an indicator of the police forces ability to respond) and the number of incidents have been increasing over the past 3 hours.

To better understand what is happening he will drill into each of the last 3 hours to see where incidents are occurring, what types of incident these are and the themes that are emerging about these incidents. Based on this information, and a list of current events, he is able to determine that the Spring Jazz concert at Pine Lake is the likely cause.

He decides to take action by deploying more officers to this location. He reviews who is available, and has not exceeded their limit of hours, and deploys these officers.

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Click on **John Clarkson** 2. Click **Logout** | Let’s log out as the officer and see what extra functionality another type of user has. |
|  | 1. Enter Username : edodds 2. Enter Password : edodds 3. Click Right arrow | We are now logged in as Evan Dodds who is the supervisor. |
|  | 1. Click on link to Power BI dashboard | Now as the supervisor, we have unlocked extra functionality to look at the incident and ticket reports. |
| C:\Users\gerard\AppData\Local\Microsoft\Windows\INetCacheContent.Word\powerbi.png | 1. Explain that Power BI dashboards can combine data from multiple sources. 2. Observe that the dashboard combines incident information, stored in an Azure SQL db, and Ticket information stored in Doc db. | Power BI dashboards can combine data from multiple sources.  This dashboard demonstrates how you can combine incident information, stored in an Azure SQL db, and Ticket information stored in Doc db.  A key metric for this police force is the ‘Current average response time’ as this is an indicator of their ability to respond.  Also, the Average Incident Response Time has increased over the last 3 hours  In memory OLTP ensures that the average response time data is ‘up to the minute’ and by passing data using Rest API the data in the dashboard reflects what is current in the database.  *Note: There will need to be approx. 30 seconds of talking at this stage to allow time for the cards consuming Rest API data to update, demonstrating how real time OLTP works.*  *Note: This dashboard is connected to csv extracts to allow re-use at a later date.* |