Azure Cloud Programming

**Build asp.net core Service Fabric application in local cluster**

**Introduction**

More and more businesses use cloud as platform for their e-business due to its super performance. Microsoft Azure Cloud Active Directory provides perfect identity management platform for web application based information system. Resource groups in Azure cloud can ideally categorize the assets of the business such as the structured and unstructured data which can be stored in azure cloud and be used for whole business applications.

Azure cloud can hoop the applications from on-premises into cloud. Now it can provide containerization of .net technology for next generations of web technology which has a big change from the traditional web technology that is the ability to scale up the resource backend for high density of client access requests such as over ten millions of users can login to the system at the same time without impacting the performance. Azure asp.net micro service and service fabric is such as new technology to be developed for the future business requirement.

This article tries to give you some wet food work to save your time in learning such an exciting scale up technology in local window environment.

**Environment**

Window 10, Visual studio 2019

**Scope of the project**

1, business requirement

* As an end user, I should be able to add some data from browser and show the data entered in the browser, so I could create some memo online without using the notes.
* As the system administrator, I should be able to view, create, update, and delete the data to maintain the system in safe way

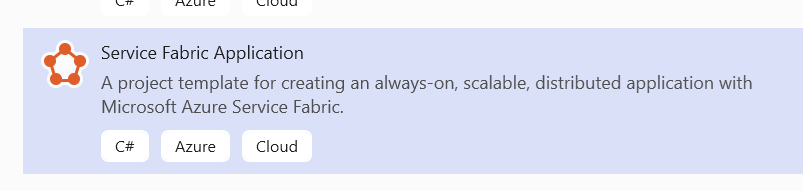
2. Technical requirement

* Develop a web service backend, a stateful asp.net core microservice,
* Develop a web service front end, a stateless asp.net core microservice,
* Develop a web client to consume those two web services
* Deploy the web applications in local cluster without IIS web server

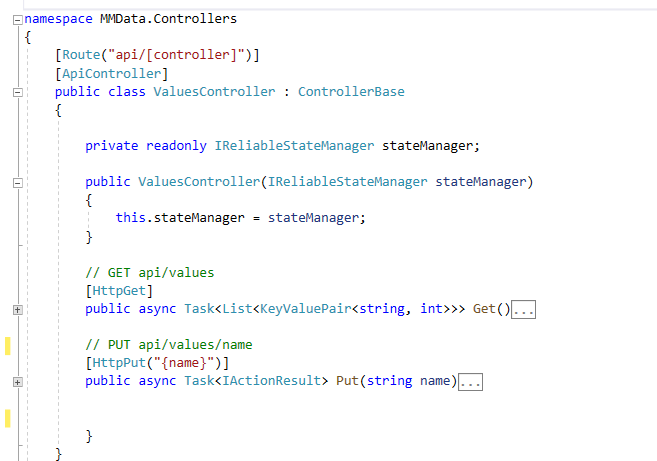
**Development**

**Asp.net core stateful microservice**

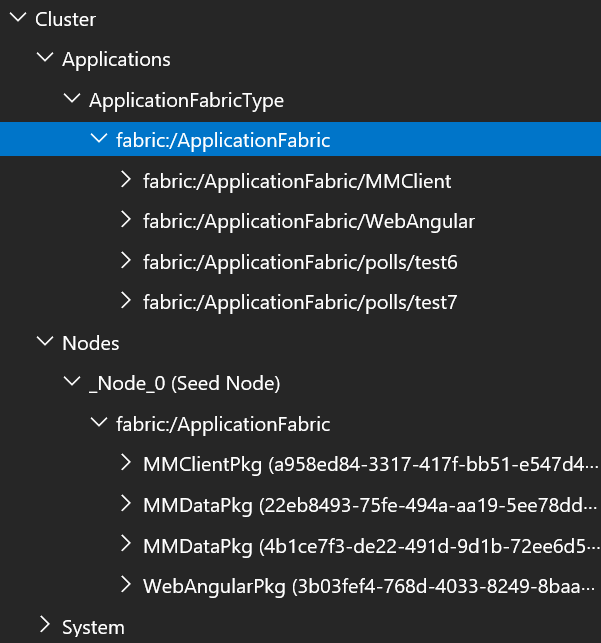
Create a new service fabric project from visual studio 2019, select stateful asp.net core web service, select API, and click Create button to bring the new project up see below

****

Add a new API controller into the project and update the get and put code as below

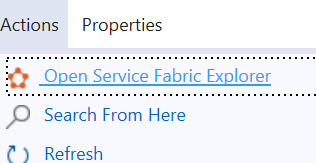


Build and deploy the solution to push the solution to the local cluster see image shown below

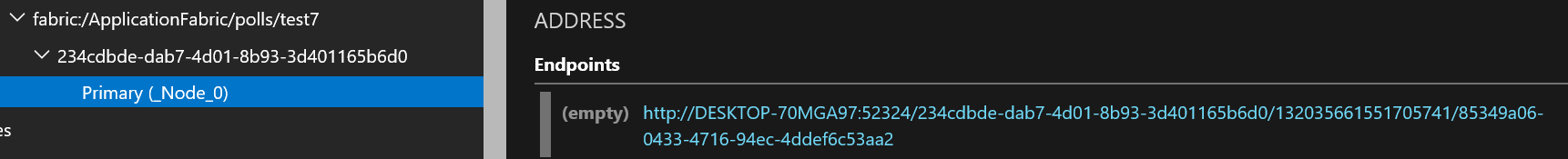


Due to the self-host capability of the asp.net core web service, local cluster can work as an IIS web server to manage the web application as IIS web server normally does. So nice new feature, which can save one IIS web server cost for us. However, it needs to get used to this new way of opening the web page from the container. The steps are listed as the following

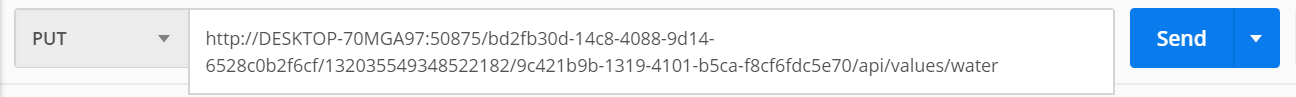
* open the cloud explorer as below to browse the cluster



* click instance in the left pane to get the micro service URL link see below



* due to the dynamic url and port number that changes all the time when the backend service is called, we can copy this dynamic url to the browser and test this get and put micro services inside the postman see below

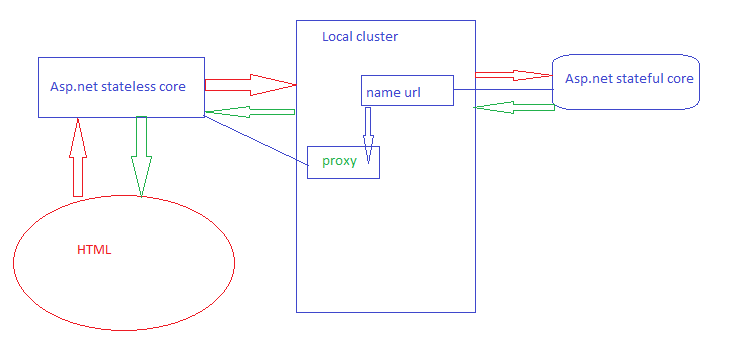


This backend micro service cannot be seen by the external third party client, dynamic url feature will allow this backend micro service could be scaled up based on the requirement. Therefore, front end microservice needs a proxy url to map to this url for communication, will discuss this later.

A**sp.net core stateless microservice**

We can add many other projects to this web solution such as the frontend stateless asp.net core MVC microservice .

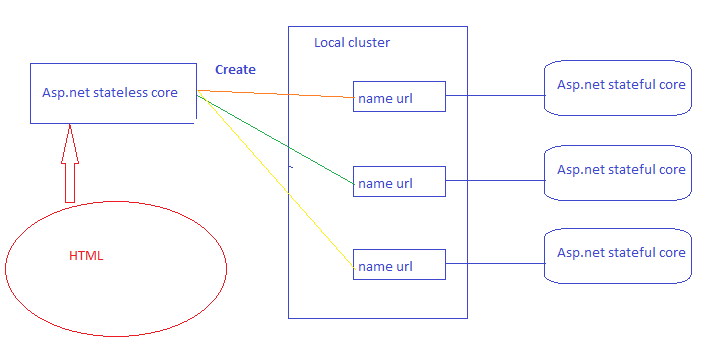
This MVC controller can use the proxy URL of stateful backend micro service to consume the data from backend.



So the method in the stateless MVC controller is the one to pass the request from UI to stateful backend micro service and wait for the response from back end and then parse the response into UI.

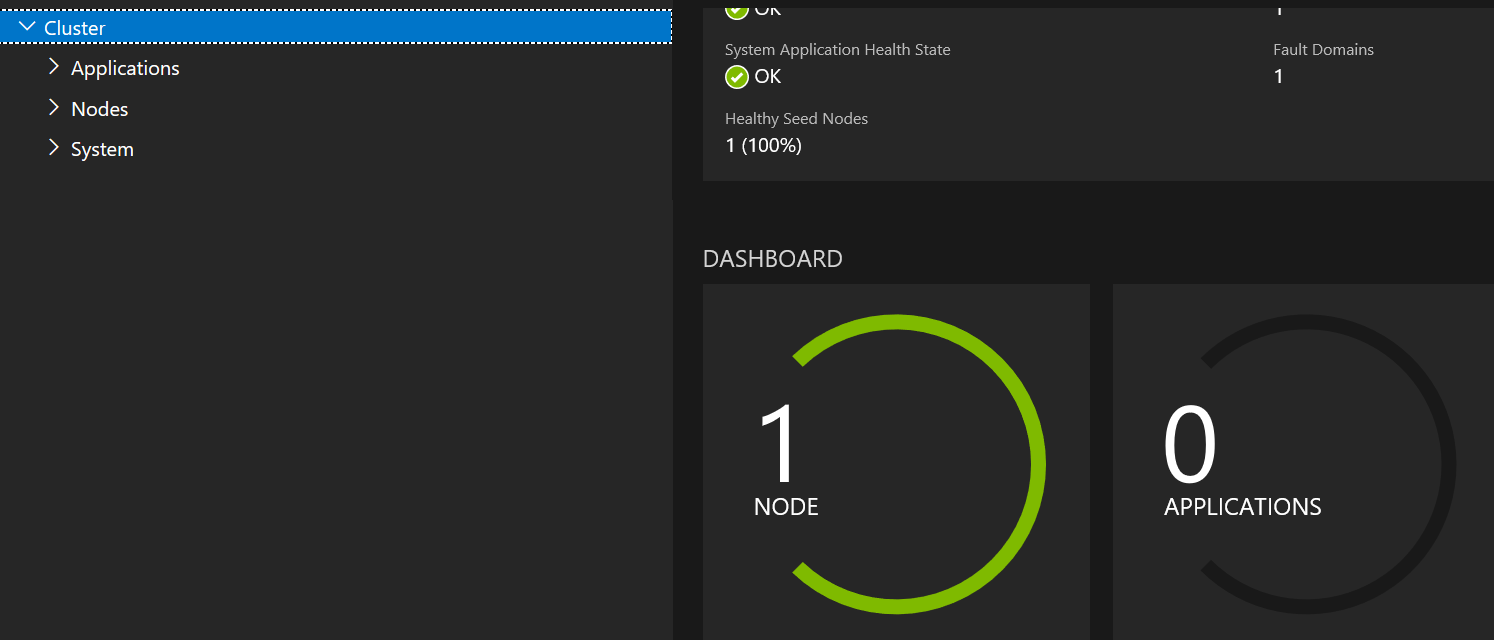
**Scale up stateful backend data micro service**

The stateful microservice is the replica service that can be copied as a new instance for new front end request. This can allow us to scale up the back end service when the front end needs to handle huge requests. Front end stateless microservice could trigger a new service in cluster that is copied from the primary backend service. After copied, this service is taken as a primary for a new client request. Then this new client service uses the new proxy URL to connect to this new backend service

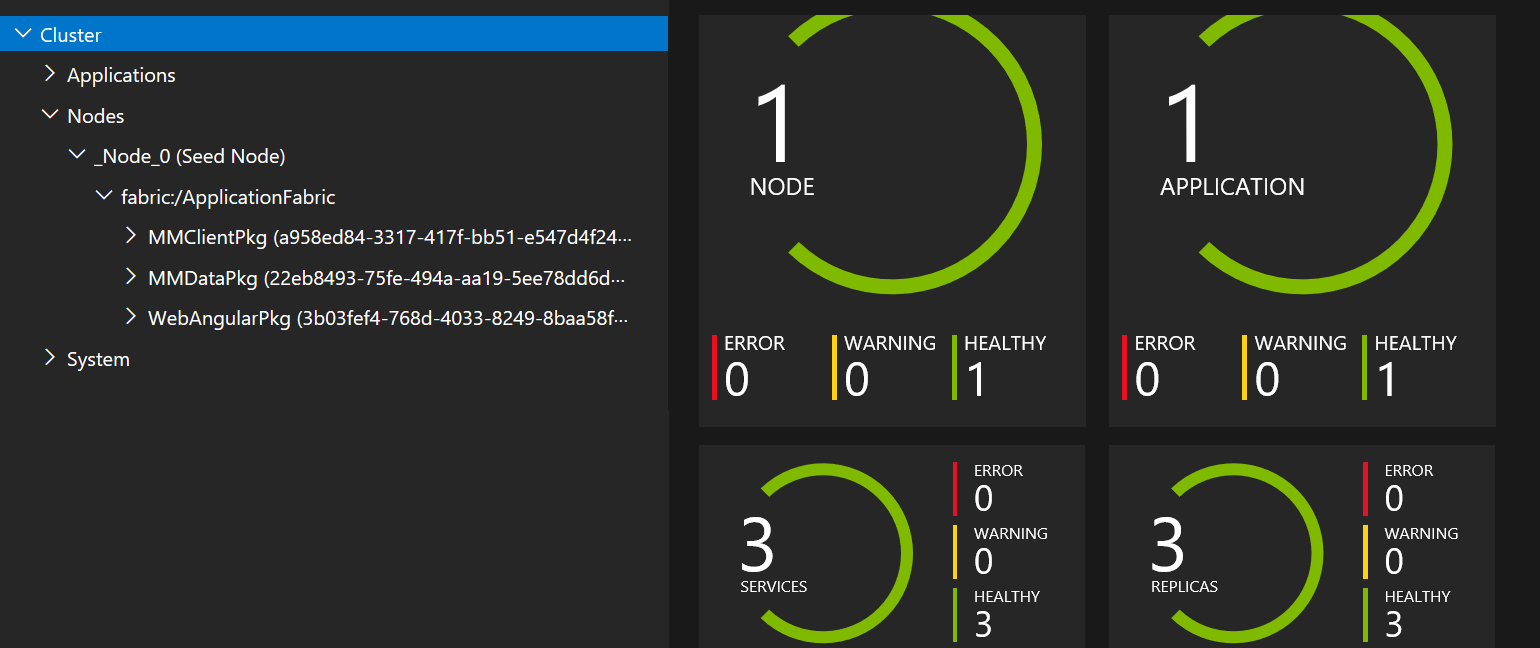


**Some discussions**

before deploying a service fabric application to local cluster, the empty cluster looks as below



run “deploy solution” or “publish” or click “start” in VS to debug the code, the application can be hosted in the local cluster as below

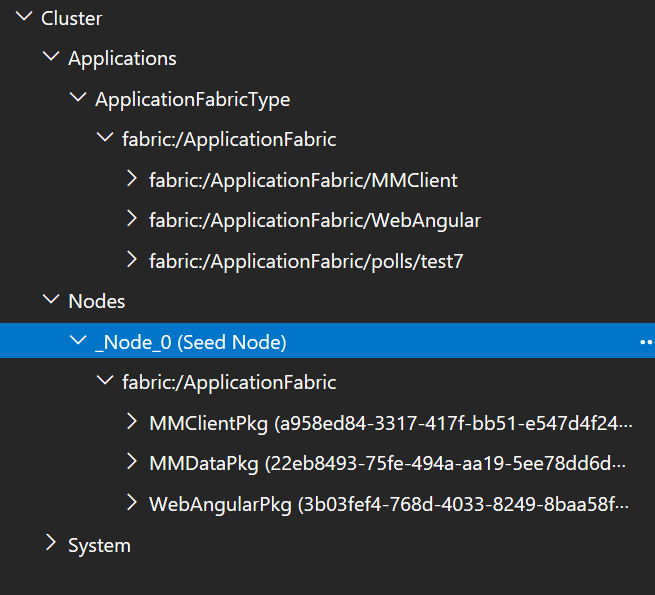


we can see we have 3 replicas are created in the local cluster.

Service fabric application is composed of stateless frontend and stateful backend microservices. one front end microservice can communicate with many stateful backend microservices, one stateful backend microservice can be communicated with many stateless frontend microservice. each microservice is hosted in a docker container inside the node. for example, three docker containers in the example above are created to host two frontend microservices and on dynamic backend microservice. This is the charmed new feature of service fabric application which allows us to scale up the backend service for the heavy requests from frontend such as 10000 login users login to the same login page at the same time, the traditional web application uses one backend to handle such a login, this normally can cause the spinning screen and waiting. This is not the good practice for the business. Business needs login page which will not cause the waiting for user. what we can do is to scale up the backends such as creating more and more backend microservices for this. If the traffic is low, then backend can be scaled down to save the resources. Definitely the service fabric is the solution.

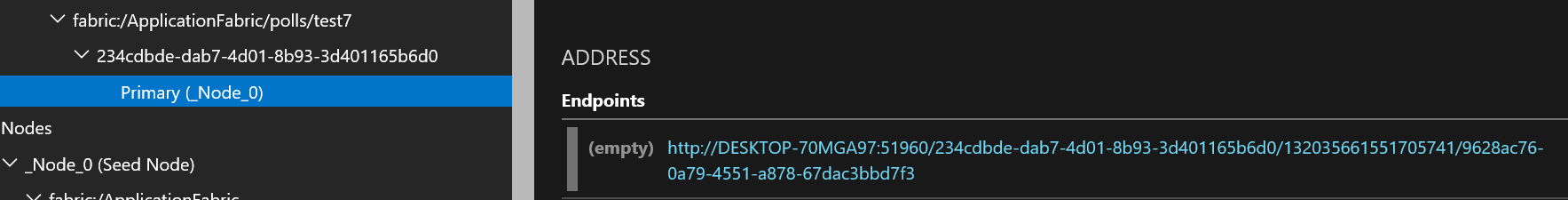
The local cloud is the azure cloud development and test environment. We can register, debug, test, and deploy the developed local application in this local cloud as we did in the real azure cloud before publishing the application to the production environment. This can save us a lots of resources for azure cloud programming.

The local cloud actually is a container in which many pods can be created. the default pod contains the seed node which is actually a docker container, this seed docker container can be replicated as required. inside this pod. each node represents a VM with sharing OS. Each node is used to store stand alone microservice artifacts see example below.

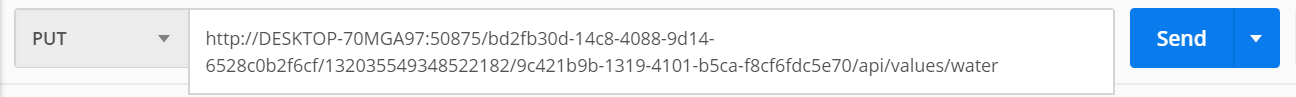


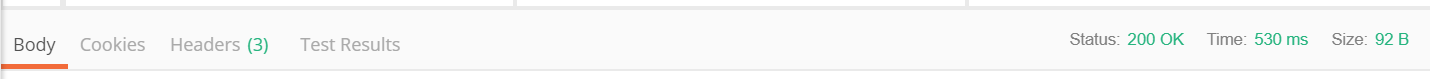
Here, we can we create three docker container inside default Node0. those three docker containers exists independently to work with their own domain functions. It means three applications in those three docker containers do not interfere each other unless we make them work together (this is called the microservice communication, it is the normal case in the real applications we need to handle).

stateful asp.net core microservice created dynamically. The example above has a test7 asp.net core microservice which is created from the code see below



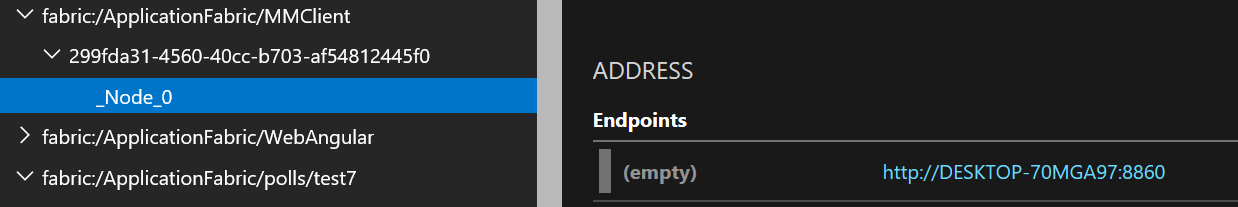
It can be created easily via a button click. Copying this endpoint URL to postman, we can be easy to test its get/put functions as example below



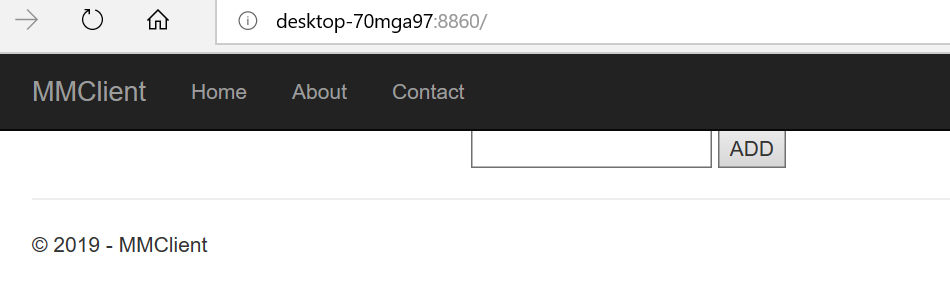


It means this stateful backend asp.net core API microservice can be used to do the CRUD for API client. However, this endpoint URL is dynamic generated, so we can not used it for API client call. We can create a proxy in API client to mapping proxy URL to this dynamic URL via naming server and DNS server in azure/local cloud.

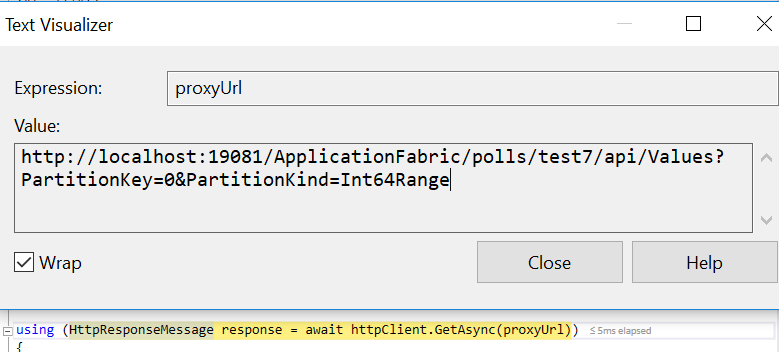
Stateless asp.net core microservice API client. We can create asp.net core MVC as API client as example below

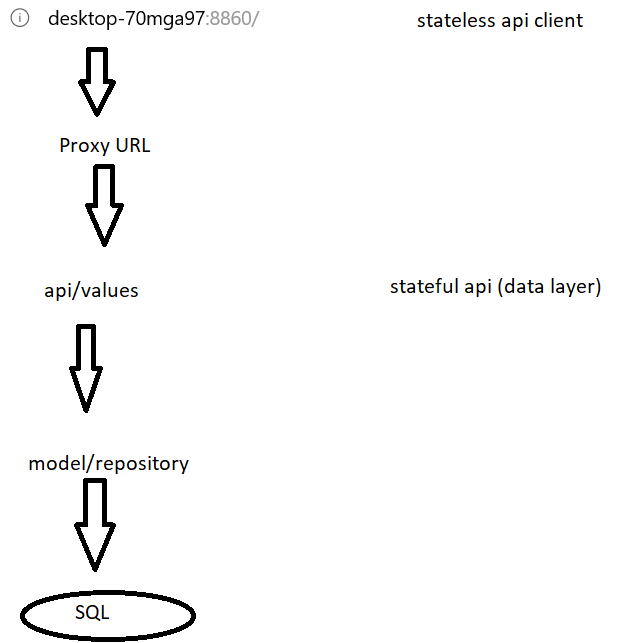


this client is hosted in another docker container. it has a static endpoint URL so we can call it from the browser as below



Microservice communication in service fabric. As example I developed here, stateless asp.net core MVC is a microservice hosted in a docker container as API client which will call the stateful asp.net core API microservice in another docker container. API service endpoint URL is created dynamically so API client will not use it to call it. So how this communication could happen? we use the proxy URL in API client to achieve this see below



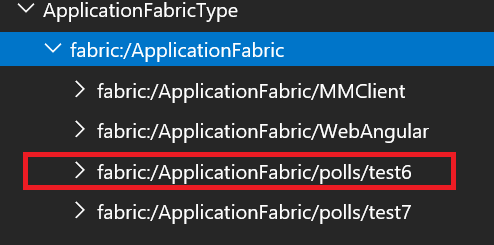


Proxy URL allows API CLient call API service to get data back even it does not know the endpoint URL, it still can find out the API methods through Naming server and DNS server to map the backend service endpoint url to the front API client url for call. this is the magic technique.

Scale up Stateful asp.net core microservice, Some time we can find out that the “/poll/test7” API does not response the requests quickly due to its bandwidth. We need to scale up this backend bottle neck so the requests can be assigned to the replica backend via the following example,



Here we create a new method in home controller of API client, when a new query string is passed in, a replica API is created in backend see below



We can image how many application can be replicated here for the huge client requests. We can scale up the login backend based on the region organization and user name, etc. this can partition the bundle requests into small bundle to access to login database for updating.