PT 4

AMARJEET BRIJNANDAN SINGH - A0150157M  
  
CHENG HAO - A0006467E  
  
GONG SHENGLIANG - A0112429M  
  
HU RENWEN - A0150352U  
  
SMITA RANI BISOYI - A0150422X

Cloud computing

Active Learning on the Cloud Project Report

Team member Contributions

|  |  |
| --- | --- |
| Team member | Contributions |
| AMARJEET BRIJNANDAN SINGH | * Explore and setup Azure Docker Service * Explore and setup Azure Storage Account * Explore and setup alert for Web App |
| CHENG HAO | * Modify application to utilize Azure blob storage * Explore and setup Traffic Manager * Explore and setup Application Insights * Explore and setup performance test * Explore and setup IIS remote management |
| GONG SHENGLIANG | * Deploy Web App * Deploy Azure DB * Explore and setup Web App auto scaling |
| HU RENWEN | * Explore and setup Azure DB Geo replication * Explore and setup alert for Web App Service Plan * Explore and setup alert for Azure DB |
| SMITA RANI BISOYI | * Explore and setup Azure Logic App * Explore and integrated SendGrid email API * Explore and setup Dashboard |

Some Notes:

The following were not selected for the presentation, reason being:

* Docker runs on VM which is too costly for the Geo redundancy
* SendGrid free account has limited service time, and it is not really unique cloud feature

Contents

[Background 4](#_Toc486783466)

[Architecture 5](#_Toc486783467)

[Setup Journal 6](#_Toc486783468)

[1. App Service 6](#_Toc486783469)

[a. App Service Plan with Geo Redundancy 6](#_Toc486783470)

[b. Web App 6](#_Toc486783471)

[c. Deploy application 7](#_Toc486783472)

[d. Auto scaling 8](#_Toc486783473)

[e. Web App Exception Alert 11](#_Toc486783474)

[f. App Service Plan Usage Alert 13](#_Toc486783475)

[2. Azure SQL database 13](#_Toc486783476)

[a. Create Azure SQL database Server 13](#_Toc486783477)

[b. Create Azure SQL database 14](#_Toc486783478)

[c. Geo Replication 16](#_Toc486783479)

[d. Application database connection string change 17](#_Toc486783480)

[e. Usage Alert 18](#_Toc486783481)

[3. Azure Storage Account 18](#_Toc486783482)

[a. Create Azure Storage Account 18](#_Toc486783483)

[b. Code change to utilize Blob Storage 19](#_Toc486783484)

[4. Application Insights 19](#_Toc486783485)

[a. Create Application Insights App 19](#_Toc486783486)

[b. Code change to trigger client page tracking 20](#_Toc486783487)

[c. Metrics monitoring 21](#_Toc486783488)

[d. Analytics tool 21](#_Toc486783489)

[e. Live stream 22](#_Toc486783490)

[5. Traffic Manager 23](#_Toc486783491)

[a. Create Traffic Manager Profile 23](#_Toc486783492)

[b. Add End Point 23](#_Toc486783493)

[6. Dashboard 24](#_Toc486783494)

[a. Create new dashboard 24](#_Toc486783495)

[b. Share the dashboard 24](#_Toc486783496)

[c. Pin desired metrics to the dashboard 25](#_Toc486783497)

[7. Performance Test 26](#_Toc486783498)

[a. Local performance test 26](#_Toc486783499)

[b. Cloud performance test 28](#_Toc486783500)

[8. Azure Logic App 33](#_Toc486783501)

[a. Data source preparation 33](#_Toc486783502)

[b. Azure Logic App Creation 33](#_Toc486783503)

[9. IIS Remote Management 36](#_Toc486783504)

[a. Download the Publish Profile 36](#_Toc486783505)

[b. Connect from IIS manager 37](#_Toc486783506)

[Lesson learned and challenges faced 37](#_Toc486783507)

[1. Changes needed in the application 37](#_Toc486783508)

[2. Choice of cloud provider and service 38](#_Toc486783509)

[3. Performance and resource requirement 38](#_Toc486783510)

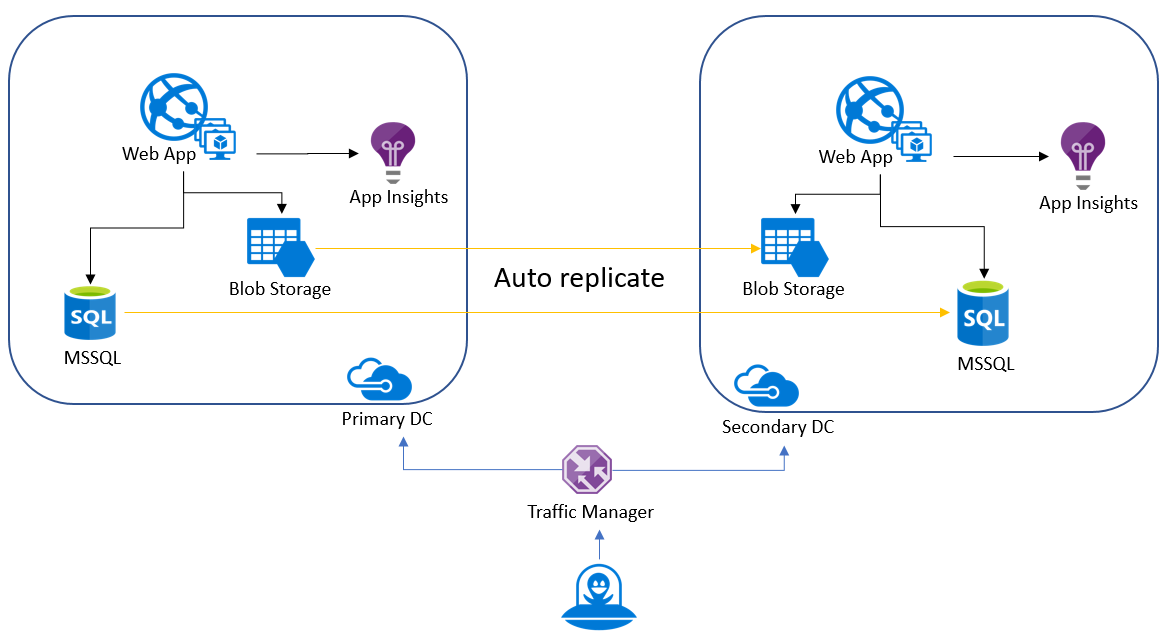
[4. Costing 38](#_Toc486783511)

[THANK YOU ! 39](#_Toc486783512)

# **Background**

* Active Learning Institute had launched an e-learning system in 2016. It was hosted on Microsoft Azure VM (IaaS) with localhost SQL database.
* There were high demands of the registration and course online browsing, which caused poor performance of the system.
* The IT department head of the institute is looking at optimizing the program of the system as well as a better hosting solution to achieve quick and flexible resource scaling, and yet at a reasonable cost.
* After a few internal meetings, the IT department head decided to migrate the system to Microsoft Azure
  + Web App (PaaS)
  + Azure MSSQL
  + Blob Storage
  + Traffic Manager

# **Architecture**



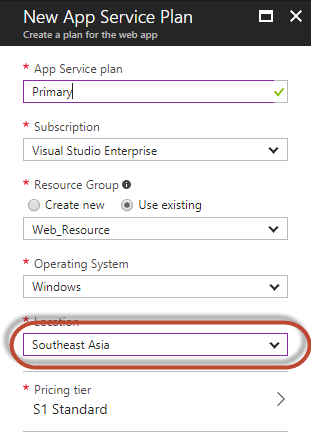
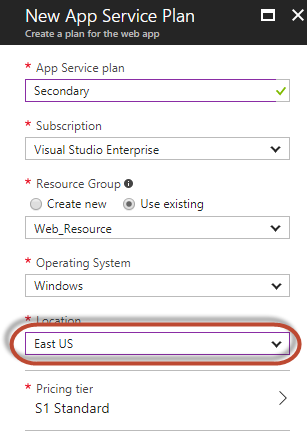
# **Setup Journal**

## App Service

### App Service Plan with Geo Redundancy

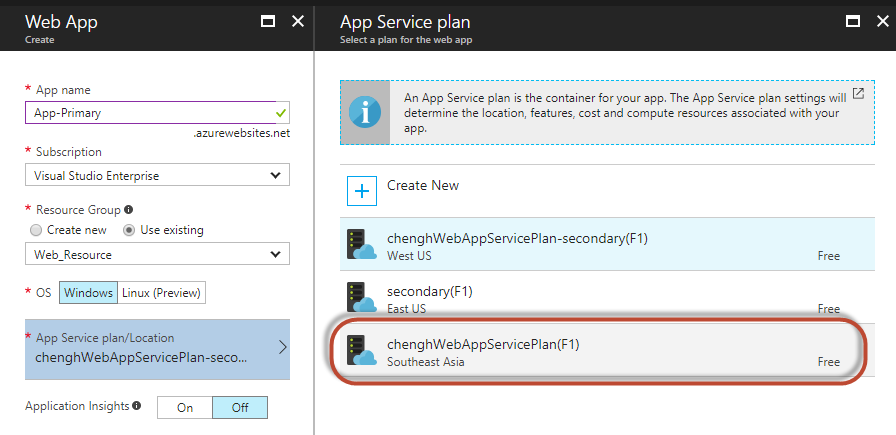
Create 2 App Service Plans in different zones

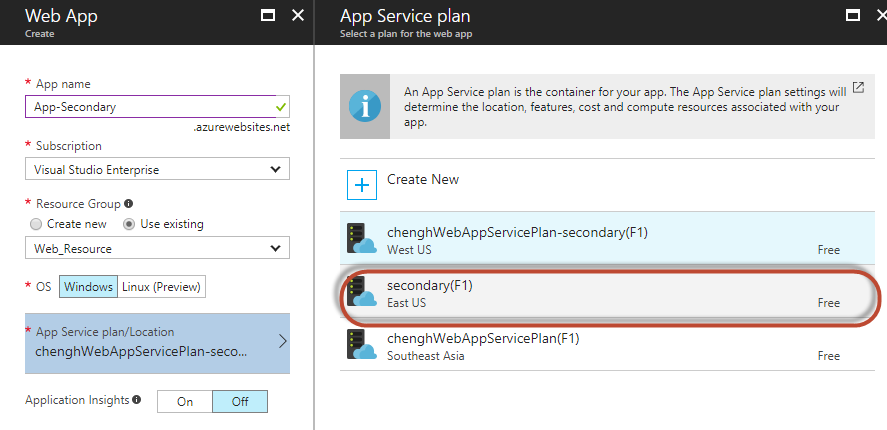
Consideration: Choose the same datacentre as database server or a datacentre near the database server datacentre, to archive low latency.

### Web App

Create 2 Web Apps, tagged under the respective App Service Plan





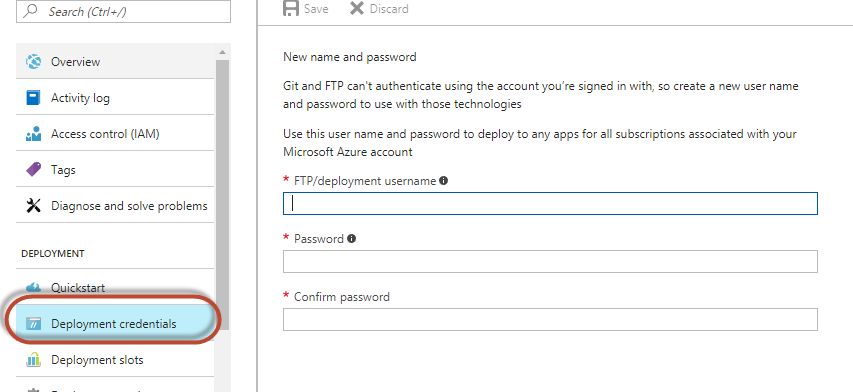
### Deploy application

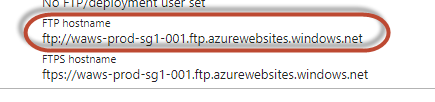
We have explored and learned 2 ways to deploy the runtime files to Azure Web App.

Each Web App will require one deployment. Hence 2 zone Geo-redundancy will require 2 separate manual deployments.

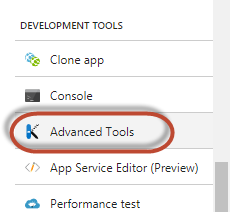
1. Via FTP

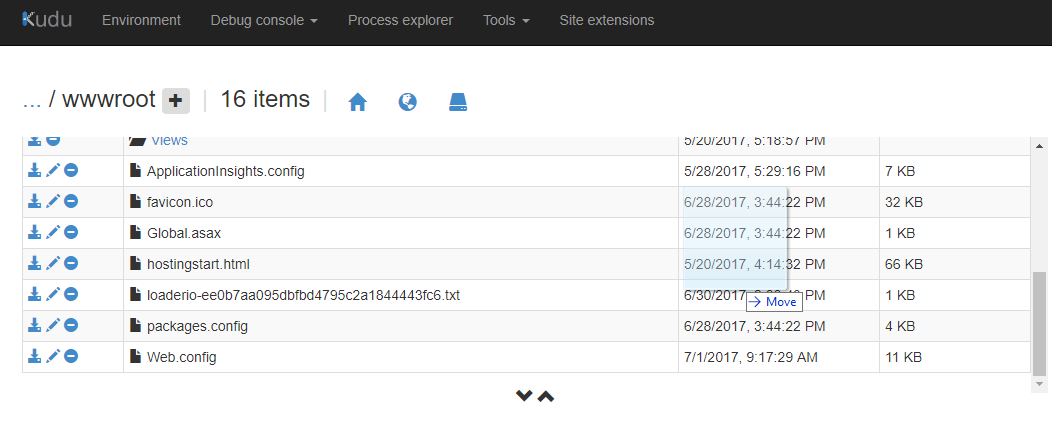
Configure the FTP account followed by accessing the FTP path under overview





1. Via Advanced Tools (Drag & Drop Web Deploy)

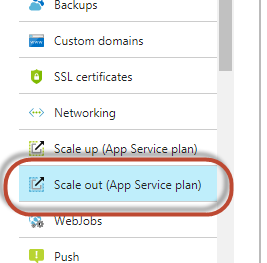




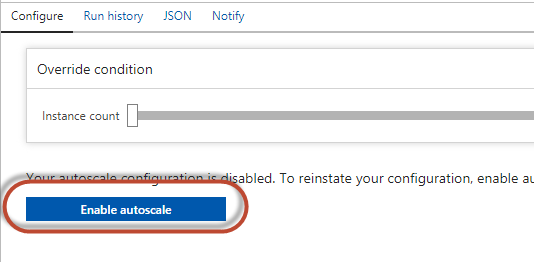
### Auto scaling

Note: Auto scaling requires min standard tier

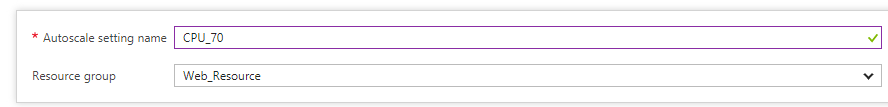
Go the Web App => Sale out.



Enable autoscale

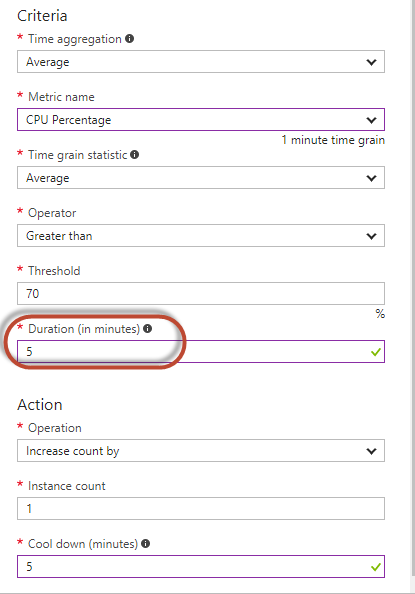


Specify autoscale configuration name



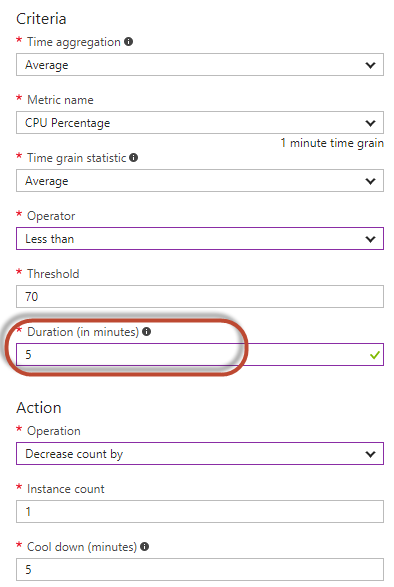
Create auto scale out rule

Set a shorter duration for a quicker auto scale out action, to minimize the impact by resource overload



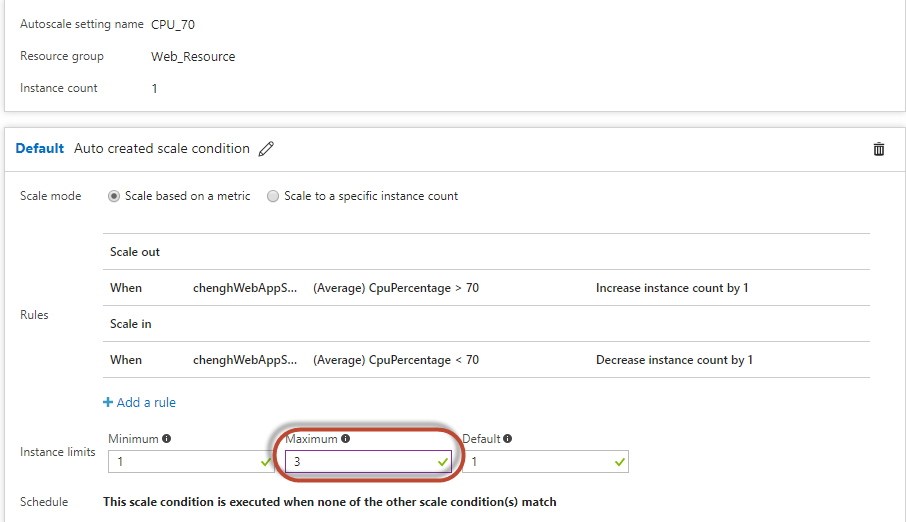
Create auto scale in rule

Set a shorter duration for a quicker auto scale in action, to archive more cost-effetiveness

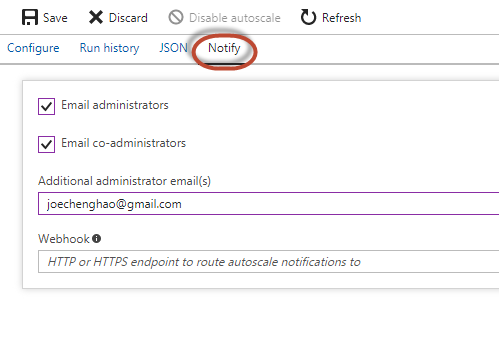


Overview

Change maximum provisioned instances to be more than 1

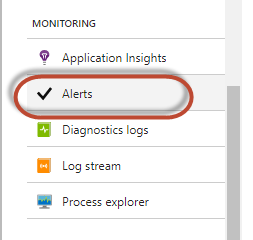


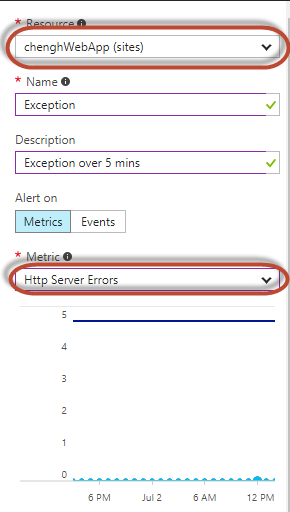
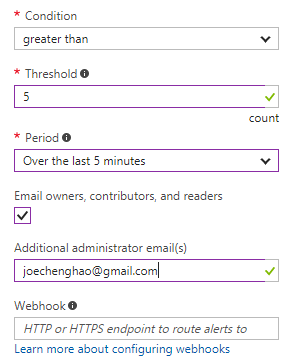
Autoscale alert



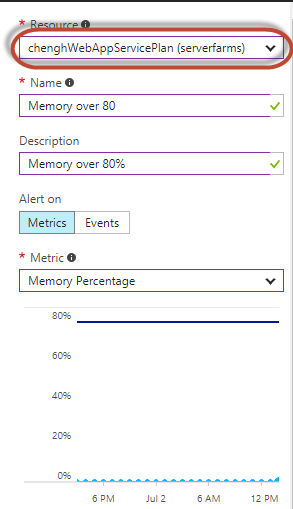
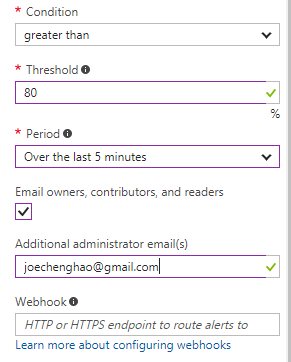
### Web App Exception Alert

Go to monitoring => alert



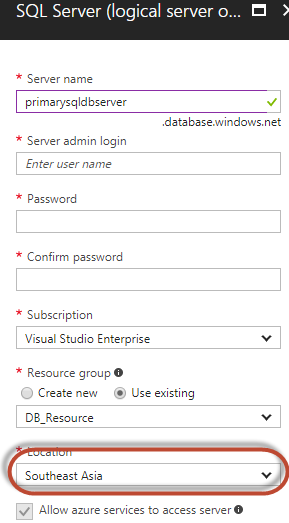
### App Service Plan Usage Alert

## Azure SQL database

### Create Azure SQL database Server

Create 1 SQL database server in the primary datacentre, to be in the same datacentre as the primary App Service Plan or nearest.



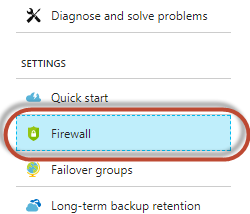
### Create Azure SQL database

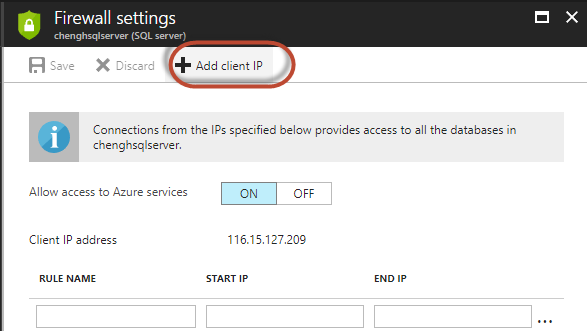
We explored 2 ways to create Azure SQL database

Since we will setup Geo Replication, we only need to setup the primary SQL database.

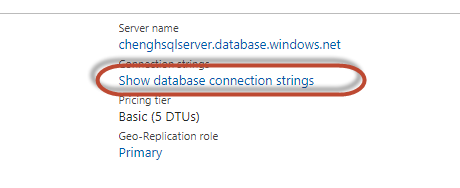
1. Via MSSQL management studio (SMSS)

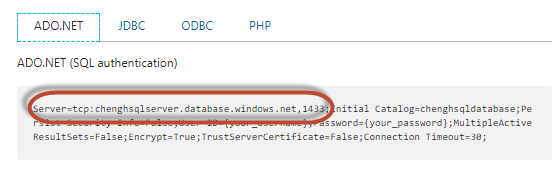
Prerequisite: Client IP is whitelisted in Azure SQL database server firewall



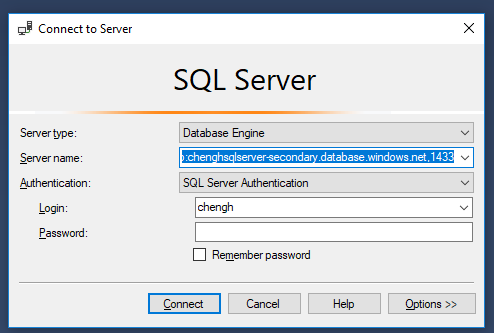


Get the connection string from SQL database

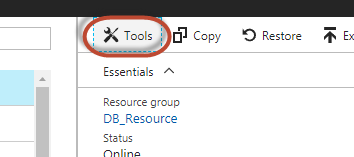




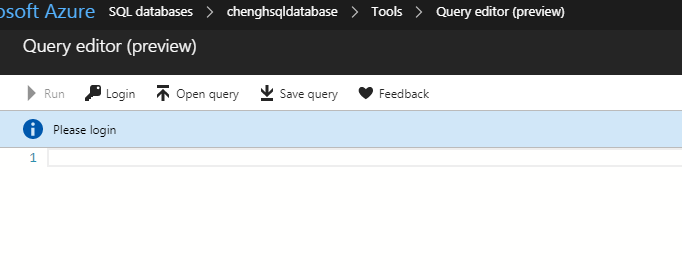
Connect via SMSS as per normal



1. Via query editor

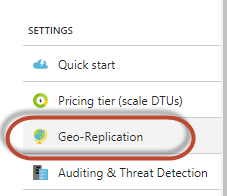


Run the full setup SQL script in the editor

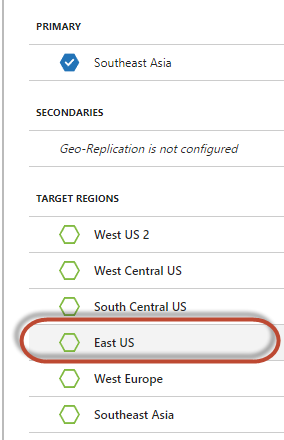


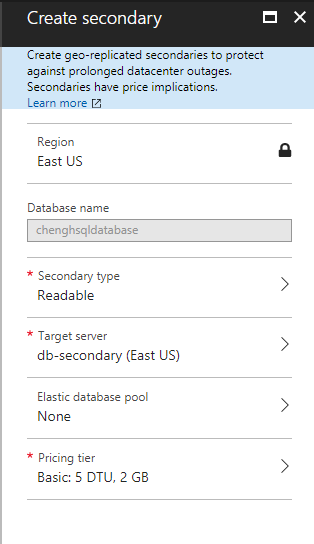
### Geo Replication

Go to the primary SQL database => Geo-Replication

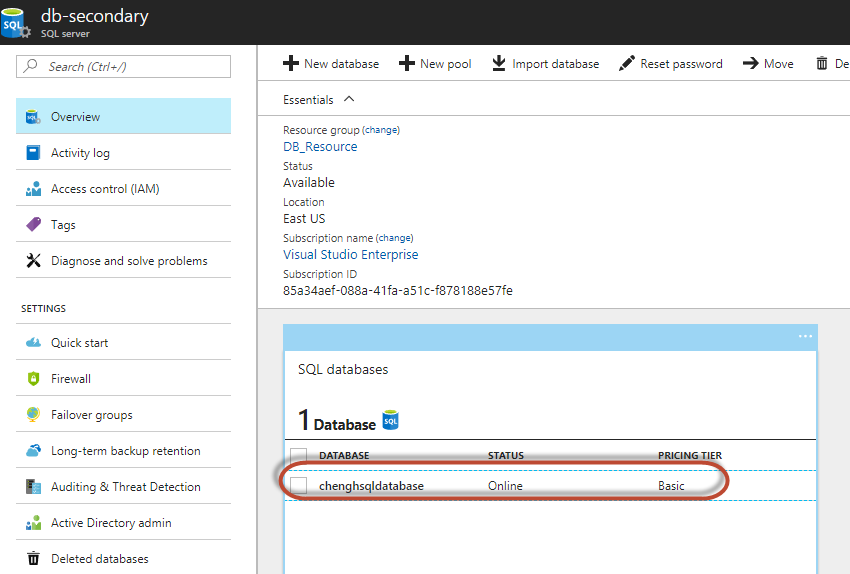


Choose a region the same as the secondary App Service Plan or nearest





A SQL database with the same name is created in the target replication SQL server.



### Application database connection string change

For the primary and secondary Web App to connect to its respective SQL database, database connection string has to be changed, however only the SQL server address.

E.g.

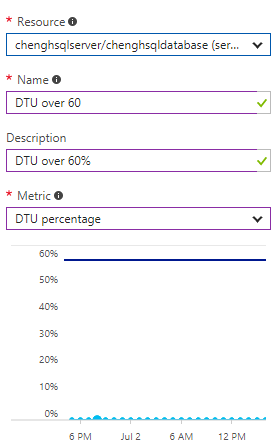
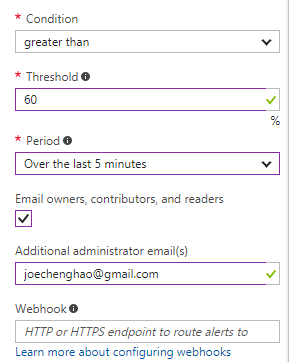
Primary: tcp:chenghsqlserver.database.windows.net,1433

Secondary: tcp:chenghsqlserver-secondary.database.windows.net,1433

### Usage Alert

Go to Monitoring => Alert rules to add alert

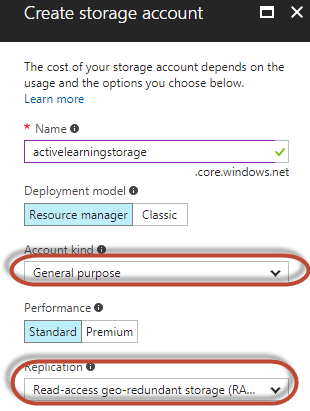


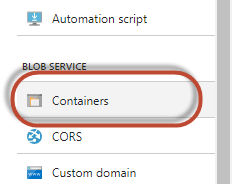
## Azure Storage Account

### Create Azure Storage Account

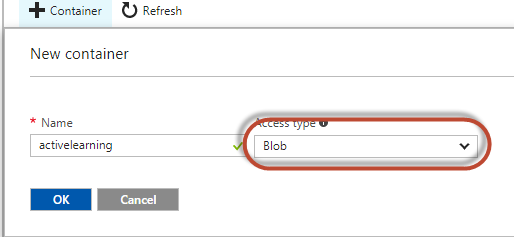
Create storage account



Create storage container



Choose blob for access type for the application to be able to access by the address of the file



### Code change to utilize Blob Storage

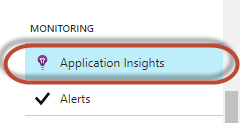
Some amount of codes need to be written in order to upload the contents to Azure storage, download content from and delete from Azure storage. Please refer to the solution codes => ActiveLearning.Business => Common => Util.cs

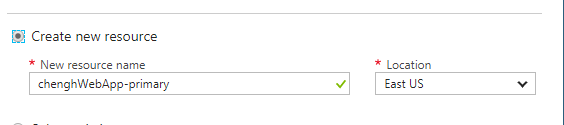
## Application Insights

### Create Application Insights App

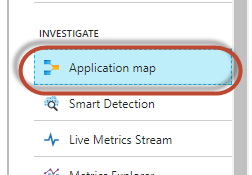
We will need to configure 2 times, for both the primary and secondary Web Apps.

Go to Web App => monitoring => Application Insights

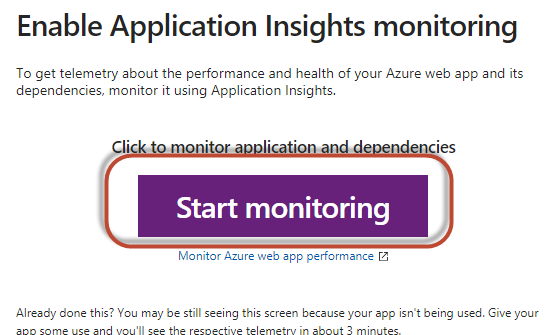




Go to Application Insights => Investigate => Application map

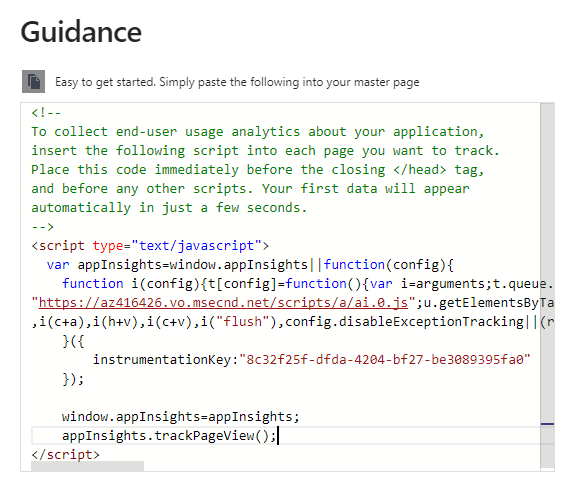


Start the monitoring.



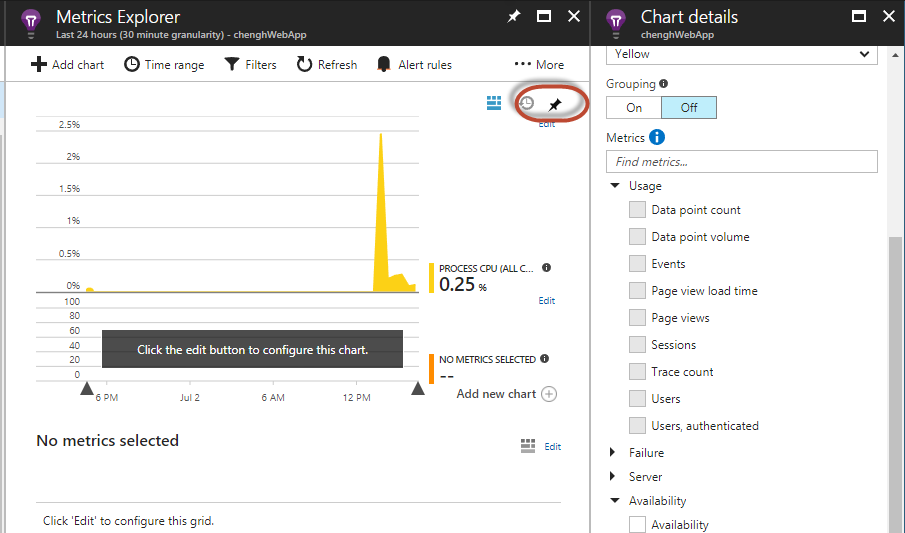
### Code change to trigger client page tracking

Click on the Client box in the application map, and copy the JavaScript code to your master page or any standalone frontend web pages.



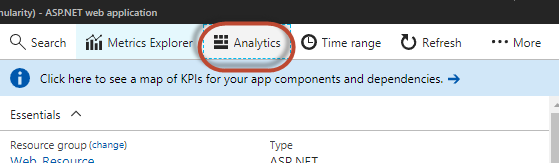
### Metrics monitoring

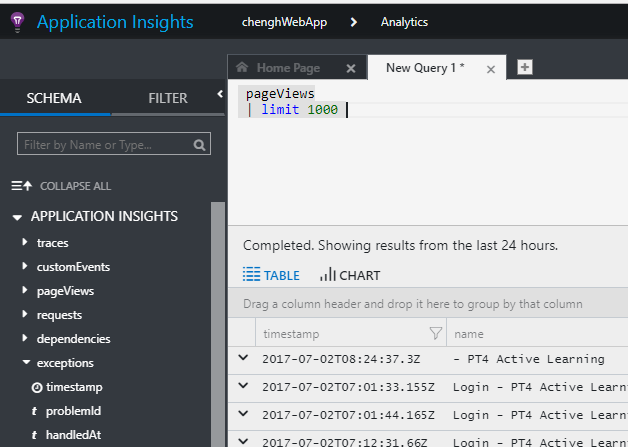
Go to the metric explorer to customize the metric, and Pin to the dashboard



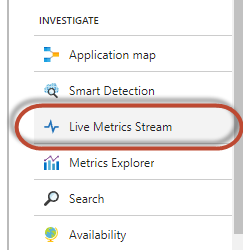
### Analytics tool

The analytics tool provide more flexibility when building customized query in the metric data.

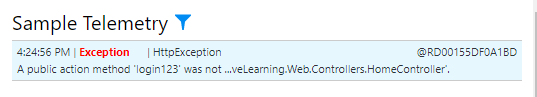


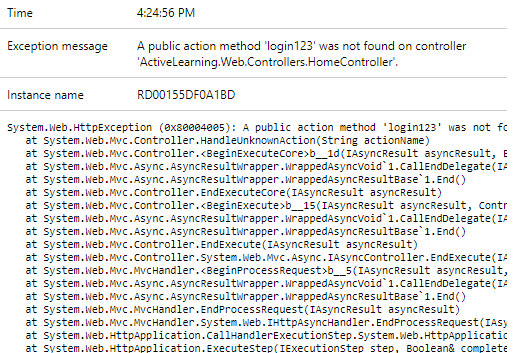


### Live stream



Realtime requests and resource usage are shown, as well as exceptions. It is very useful during a live event. Able to provide real time performance of the application for the support team.



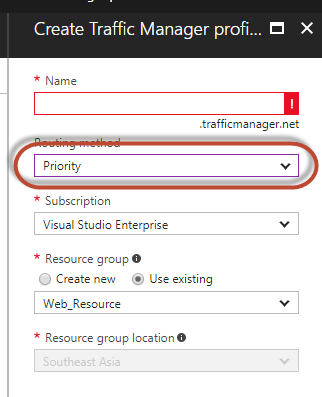


## Traffic Manager

### Create Traffic Manager Profile

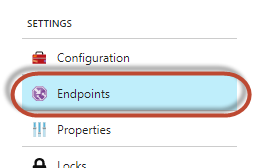
Note: All the Web Apps must be min standard tier in order to be included in Traffic Manager monitoring and traffic forwarding.

Choose Priority method for Geo Redundancy

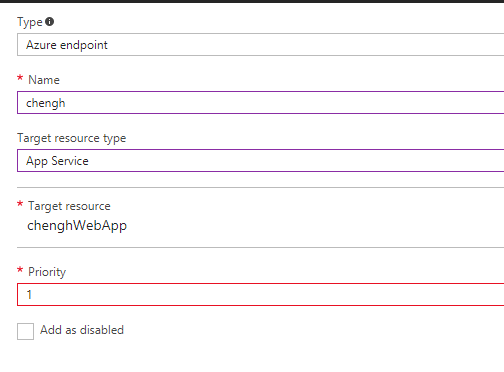


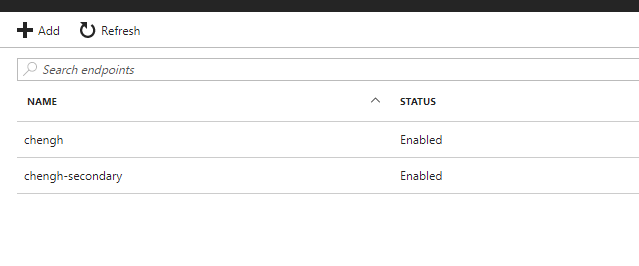
### Add End Point

Add both Web Apps in the end point



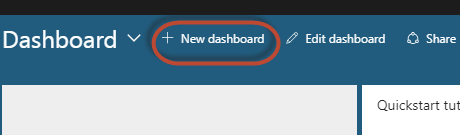
Smaller priority number denotes higher priority. Hence primary is 1, secondary is 2.





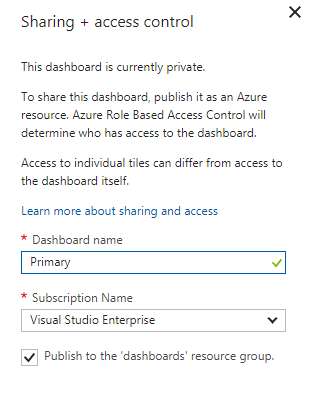
## Dashboard

### Create new dashboard



### Share the dashboard

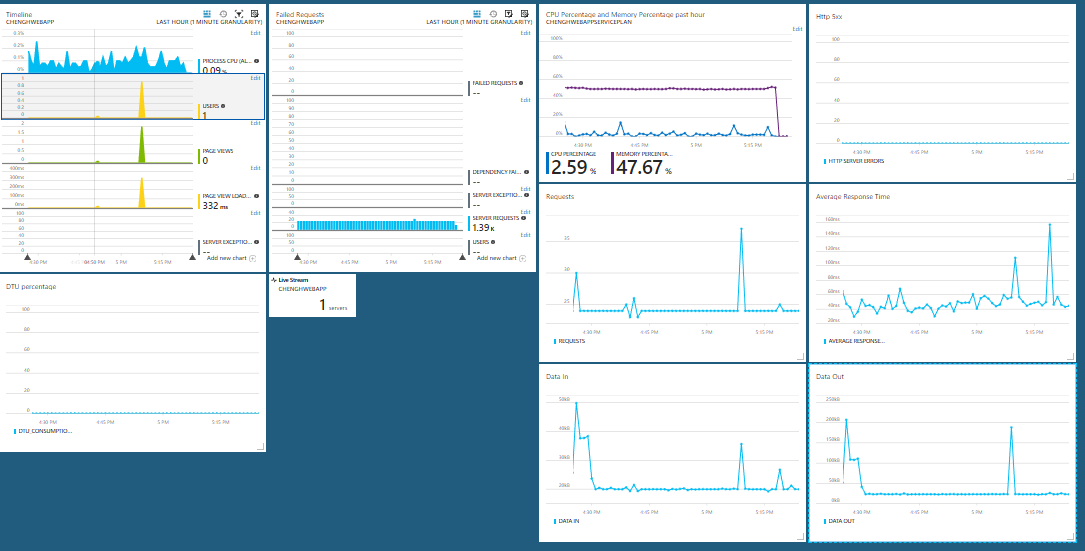
So that the users under the same subscription will be able to see the same dashboard for more effective monitoring



### Pin desired metrics to the dashboard

Metrics/Item pined

* Application Insights: resource, page view, session, user, page load time
* Application Insights: exception details
* Application Insights Live Steam Shortcut
* App Service Plan CPU & Memory usage
* Web App server error
* Web App Requests
* Web App Response Time
* Web App Data In
* Web App Data Out
* Database DTU usage



With all these metrics, we will have a better overall understanding of the performance of the application, and this could help with the cost estimation and next phase planning.

## Performance Test

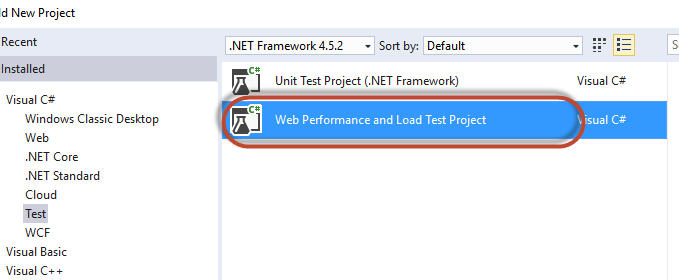
There are 2 types of performance test we can conduct to the application

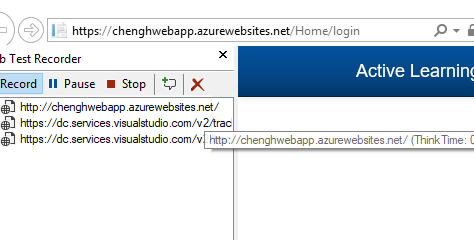
### Local performance test

Different from the literal understanding of “local”, it means the requests are triggered from local PC to the cloud application

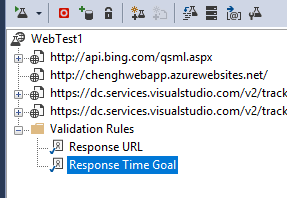
We can use Visual Studio Web Test project to facilitate the performance test

Create Web Performance and load Test project

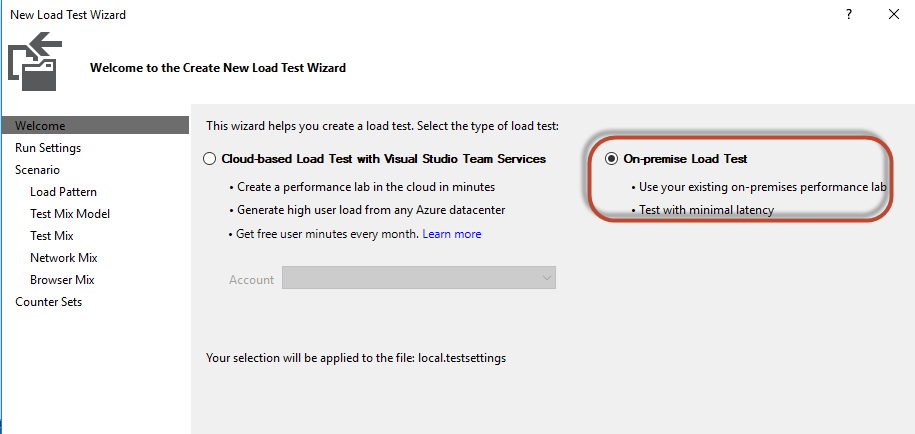


Add recording of the flow. It could be a complex flow or just page load.

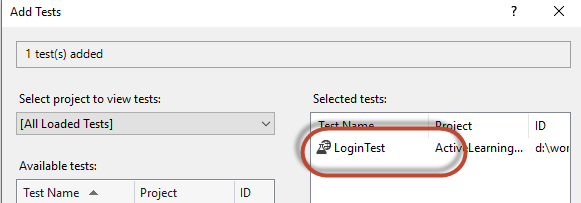
Remove unnecessary requests



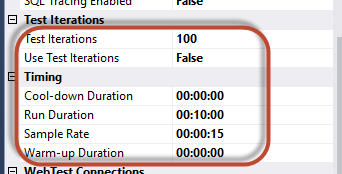
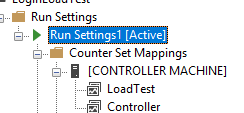
Add Load Test file and choose On-premise

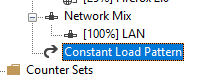
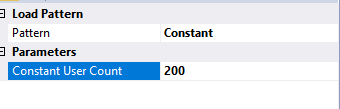


Include the recorded Test item

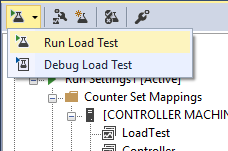


Configure the pattern (Iteration or duration) and user load

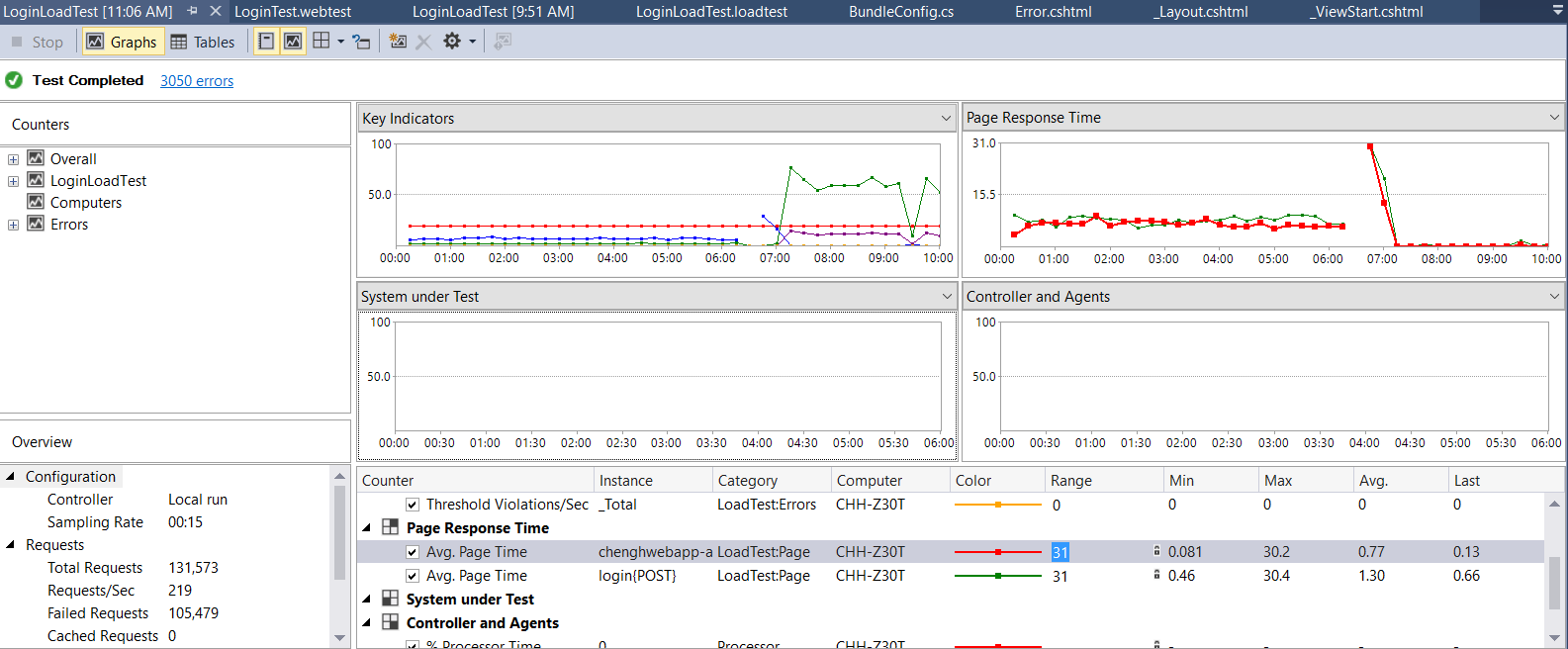


Run load test



We will get the result like below

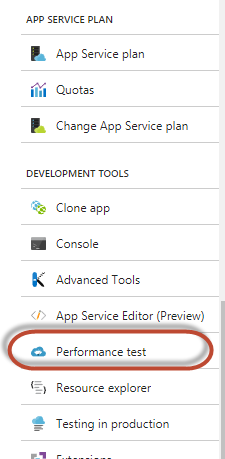


### Cloud performance test

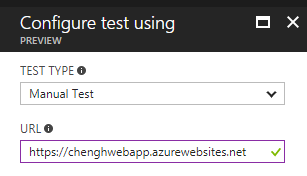
There are 3 ways to conduct performance test

1. Manual test

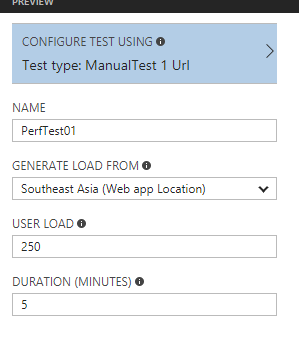
Go to Web App => App Service Plan => Performance test

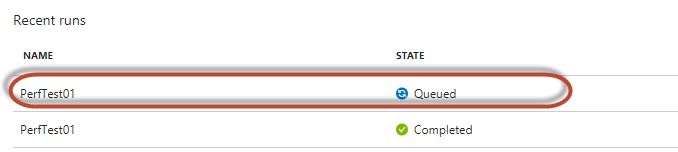


Add new test. Choose manual test and input the target URL.



Choose the desired test source region, user load and duration

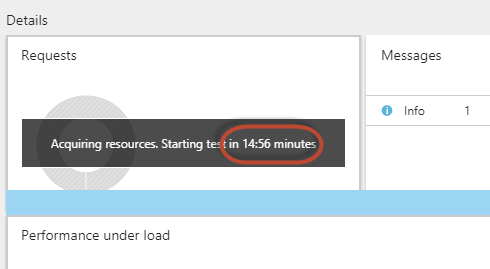




This approach could only test one single URL via GET method. Only suitable for simple test.

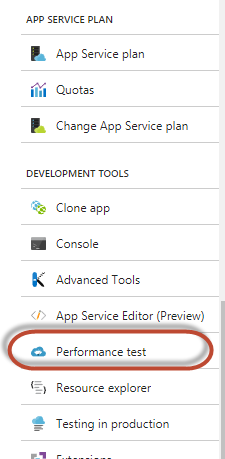
Important notes:

Every Cloud test triggered on Azure portal will need 15 minutes to acquire resources. Hence the time buffer needs to be considered.

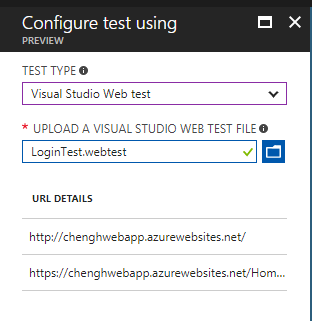


1. Cloud test using the Web Performance and Lod Test file

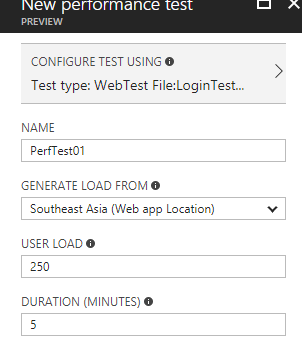
Go to Web App => App Service Plan => Performance test



Add new test. Choose Visual Studio Web test and upload the previously recorded test flow.



Choose the desired test source region, user load and duration

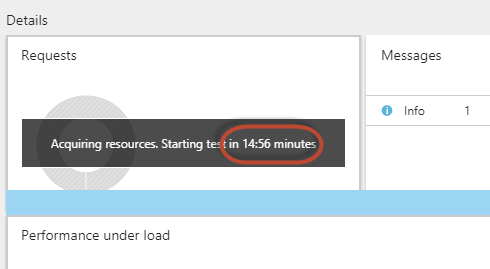




This approach could only test a complete flow including multiple URLs via both GET and POST method. It is suitable for more complex and heavier test.

Important notes:

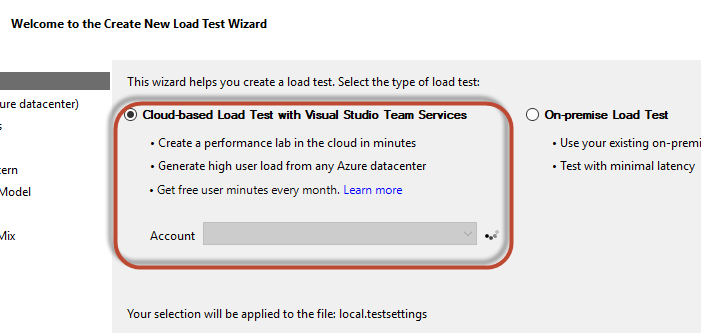
Every Cloud test triggered on Azure portal will need 15 minutes to acquire resources. Hence the time buffer needs to be considered.



1. Visual Studio Cloud Test (Advanced)

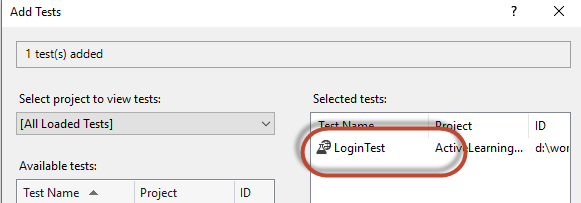
Create load test project and choose Cloud-based load test

Prerequisites: Login VSST account in Visual Studio

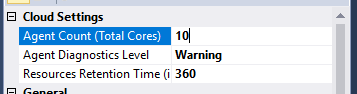


From here follow the same steps as on-premise test setting

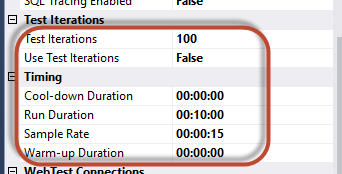
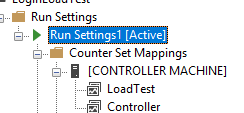
Include the recorded Test item

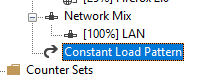
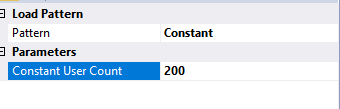


One difference is that VS cloud load test allows to configure the agent count and resource retention time (subsequent tests will not have any waiting time to acquire resource)

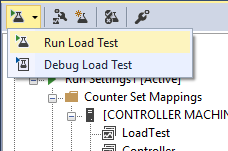


Configure the pattern (Iteration or duration) and user load



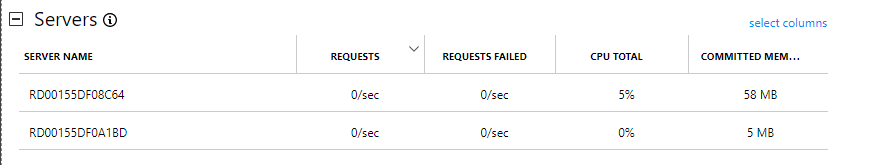
 

Run load test



While the test is running, we could monitor the dashboard to view the resource usage and performance of the application.

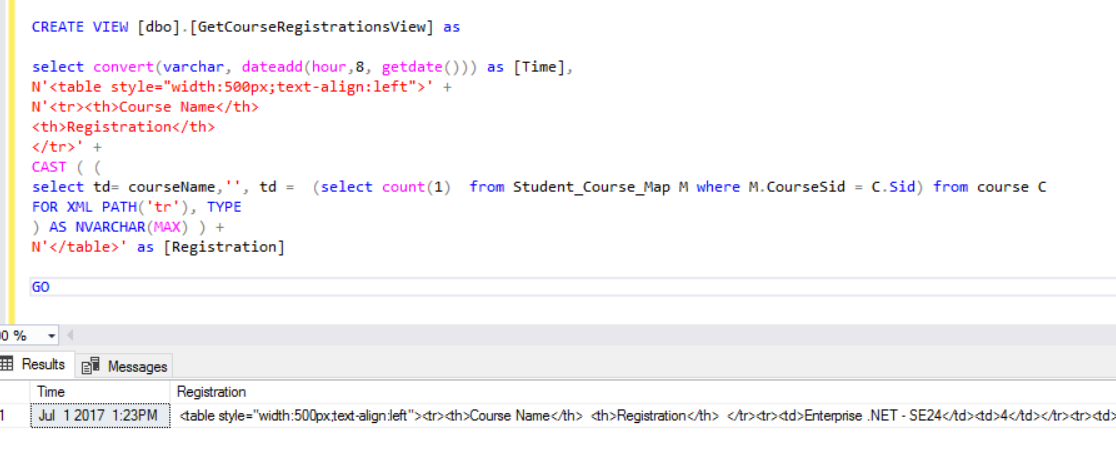
If everything is configured correctly, the autoscale up will be triggered.



## Azure Logic App

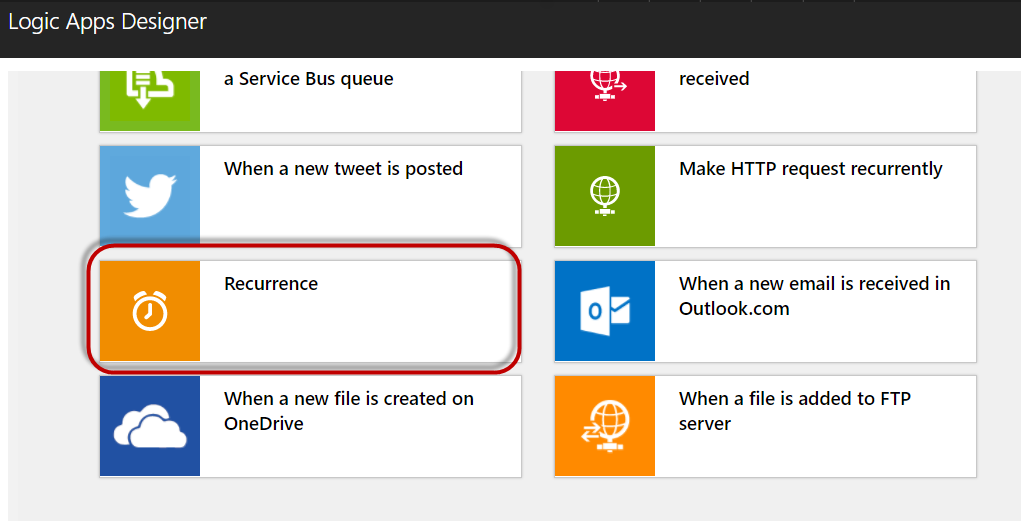
### Data source preparation

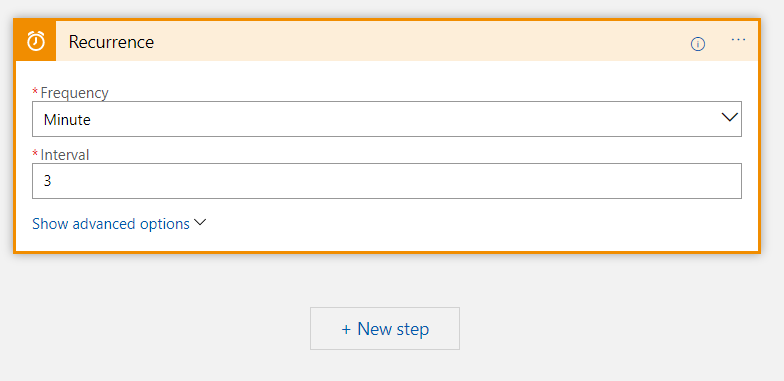
MSSQL View to construct the HTML contents



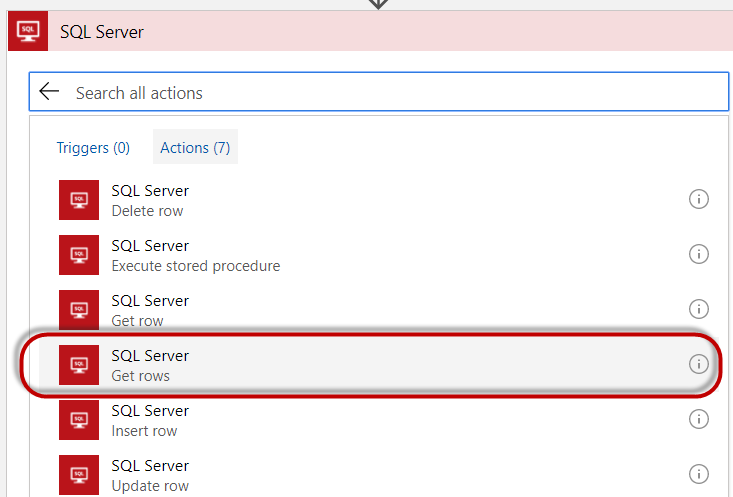
### Azure Logic App Creation

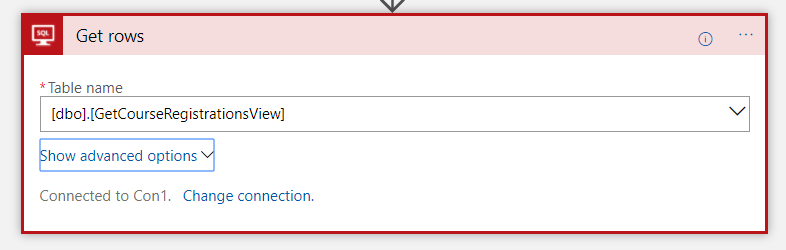
Setup trigger



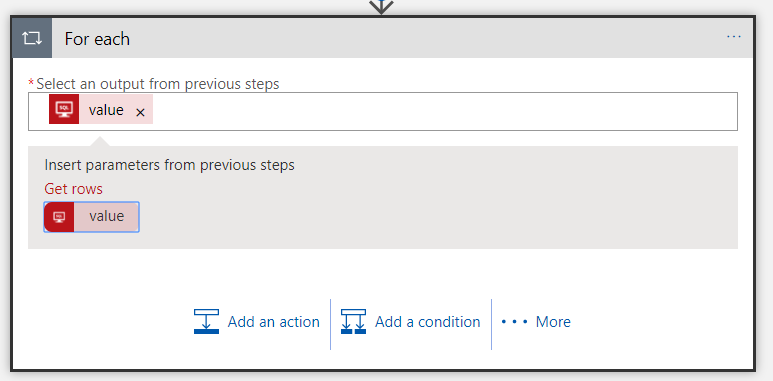


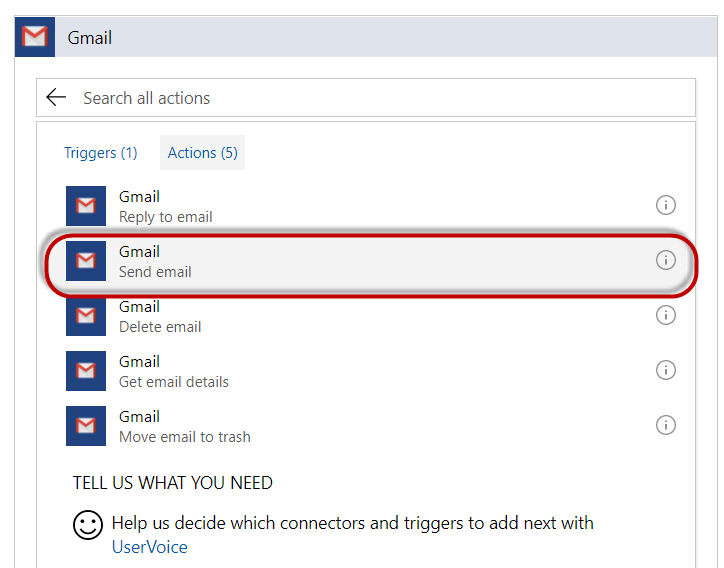
Retrieve data



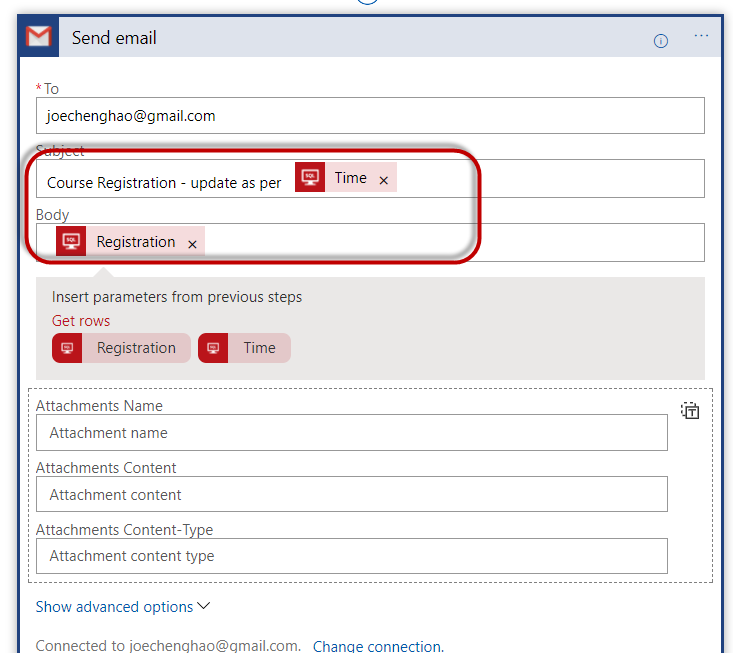


Process data and send email

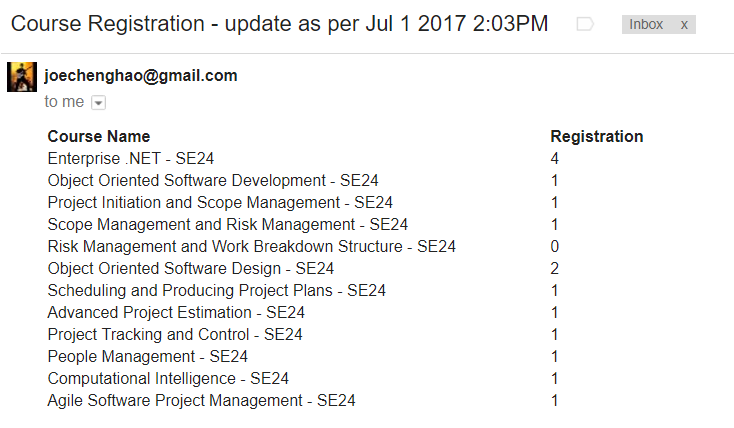




Define the subject and content using populated fields from the query result.



Emails generated.

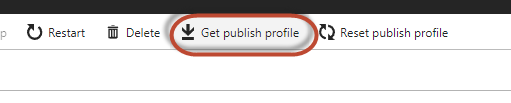


## IIS Remote Management

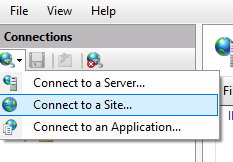
We could manage the Azure Web App remotely in the GUI IIS manager.

### Download the Publish Profile

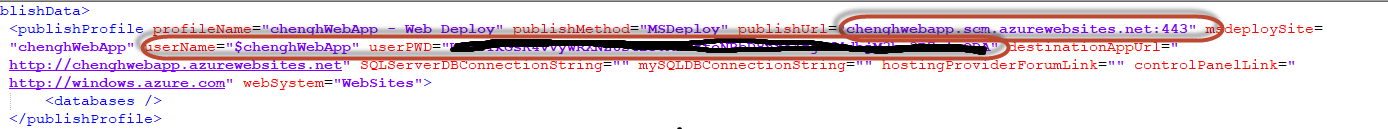
Go to Web App => Overview



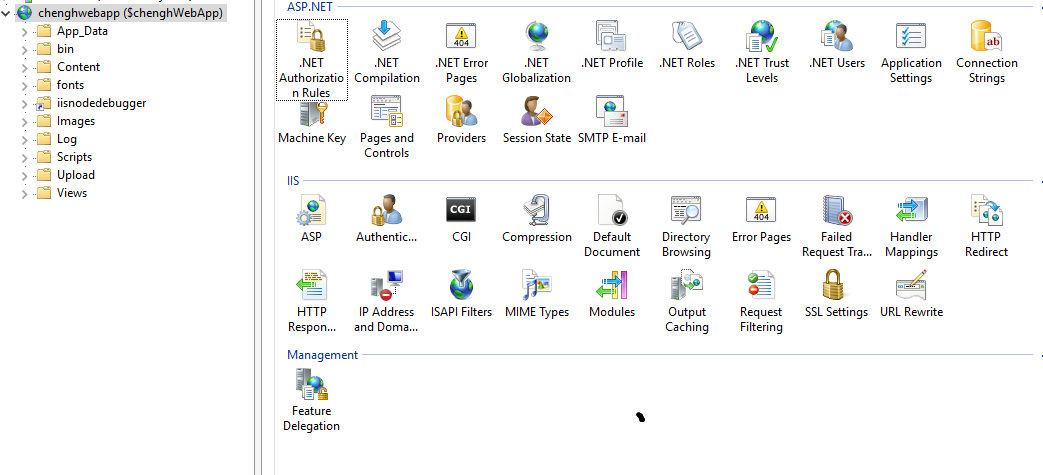
### Connect from IIS manager



User the address, account and password found in the publishsetting file



And we can manage it as per a normal localhost server



# **Lesson learned and challenges faced**

## Changes needed in the application

One of the problems arise when migrating an on-premise application to cloud is that existing stateful applications might not work as per expected on cloud, reason being when the application is scaled out, session state is no more guaranteed persistent.

There are 2 solutions we can consider

1. No change in the application architecture design, but to use centralized state management server, such as Redis Cache to archive high performance and persist the session state. However, load balanced Redis Cache servers could be costly.
2. Drastic change in the application architecture design, to add in service layer in order to decouple the presentation layer and business layer. However, the downside is the cost and time for the change.

## Choice of cloud provider and service

There are a few big players and a lot more small players in the market, with each of them providing many common services and some their own unique services. How does one determine which provider to go with?

Perhaps there is no straightforward answer. It depends on a few factors, such as costing, feature, SLA, compliance, reputation, consumer review, innovation initiative. Etc. We might still look at the nature of the business, the criticality of the application and the budget, and might as well carry out POCs and trial runs.

## Performance and resource requirement

For users without cloud experience, it will difficult to really estimate the resources needed to be provisioned. One way is to conduct load test. However, the question could be how much could the real situation deviate from the load test, plus how much will it be considering the need for multiple load test. In short, where should we stop?

## Costing

Although cloud hosting seems reduce the Capex, it might end up even more costly than on-premises infra cost if resource planning and provisioning is not done properly.

One tip will be “always remember to scale down and in when necessary”

# **THANK YOU !**