Agenda - Load Balancing

We have a great day ahead of us, split between two chapters. These are the things we will be working on.

Solution Setup

- Working locally

Chapter 1 - Local balancing

Exercise 1

Exercise 2

Exercise 3

Exercise 4

Exercise 5

Exercise 6

Exercise 7

Chapter 2 - Azure & Umbraco

Exercise 1

Exercise 2

Exercise 3

Exercise 4

Exercise 5

Appendix

Appendix 1 - KUDU

Appendix 2 - Application Initializing

Solution setup

Over the course of today we will be utilizing two different solutions as well as:

Local Environment

- Umbraco 10 (latest version)
- .NET6
- IIS
- SQL Server Engine

Bundled items:

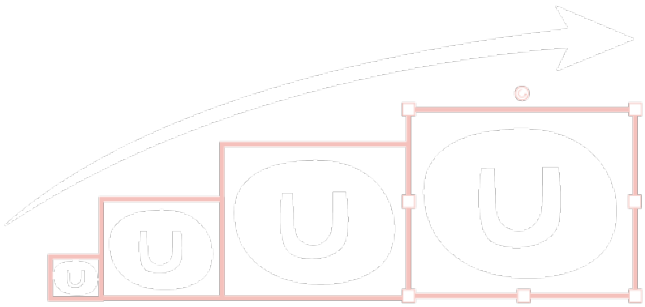
- Zip file of project solutions

Needed applications:

- Visual Studio 2022
- SQL Server Management Studio
- Active Microsoft Azure subscription (paid or trial)
- Azure Storage Explorer

Chapter 1 - Local Balancing

This chapter will take you through how to configure Umbraco v10 to exist in a load balanced environment on your local development machine.



Getting set up

Before we begin, we will need to get our environment and database created and configured.

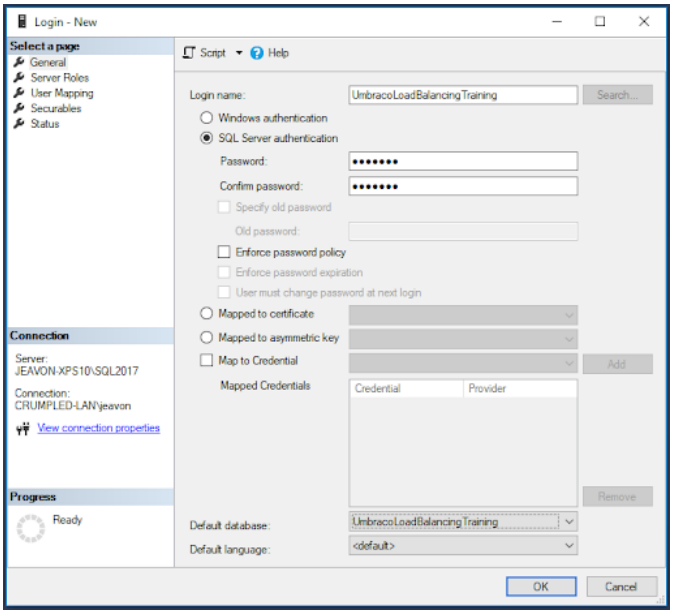
We will begin in SQL Server Management Studio to create the database and then move to Visual Studio from the UmbracoLoadBalancing.sln to install Umbraco utilizing our new database.

1. Create a database

Create an empty database on your local SQL Server called “UmbracoLoadBalancingTraining”

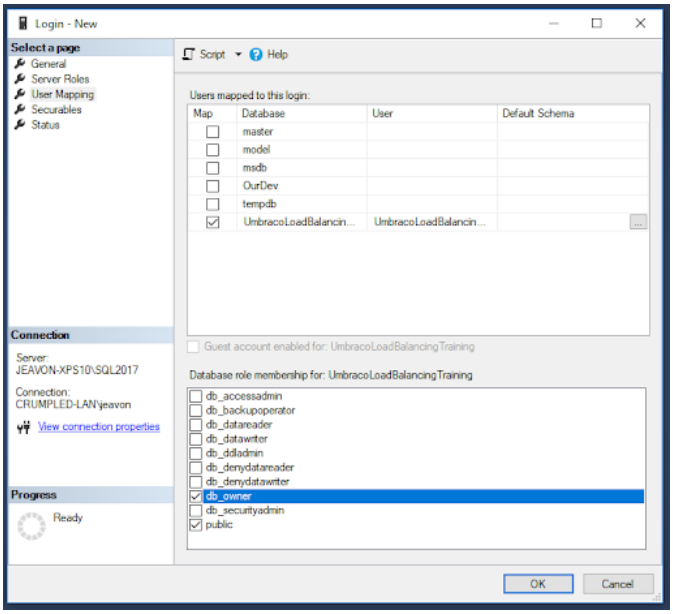
2. Create a login

In SQL server create a login called “UmbracoLoadBalancingTraining” (ensure SQL server is set to allow SQL server logins)



3. Create User Mapping

Create a user mapping between the new login and the “UmbracoLoadBalancingTraining” database you created and add “db_owner” permission

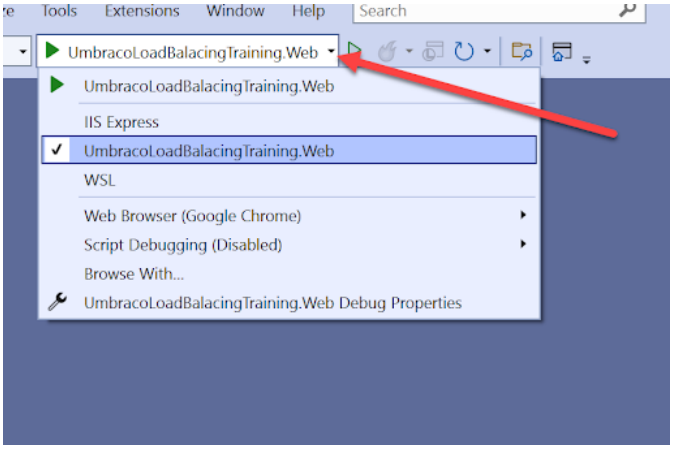


4. Open Correct Solution File

Open the UmbracoLoadBalancingTraining.sln solution from the Chapter 1 folder in Visual Studio as Administrator, also open the CopyAndPasteStuff.txt you will need this throughout the workbook

5. Run

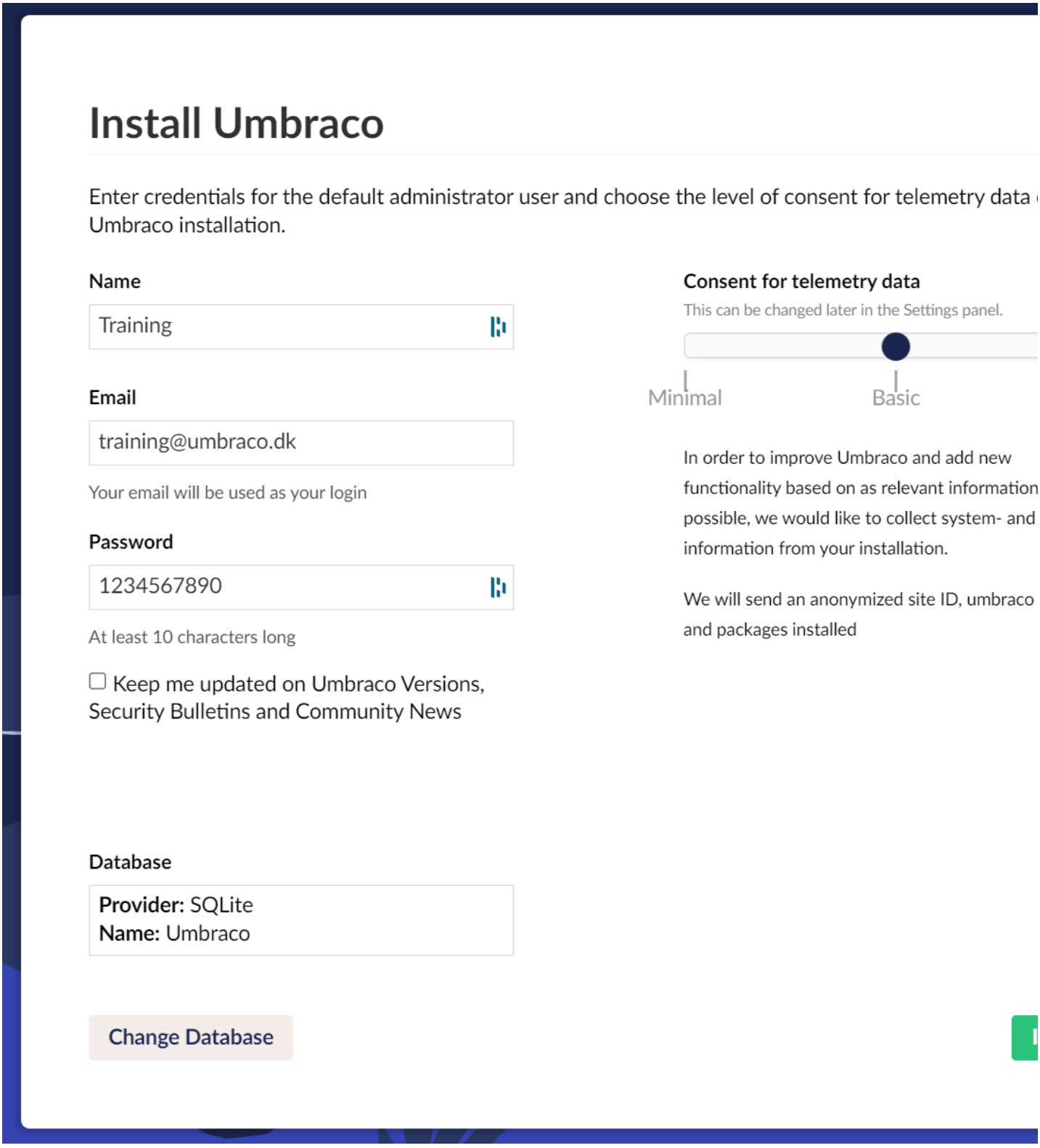
Run the UmbracoLoadBalancingTraining.Web project using the dotnet CLI by selecting “UmbracoLoadBalancingTraining.Web” from the drop down:



If prompted, add the certificate to be trusted to your computer

6. Configure your database

Choose “Change Database”



Then select SQL server and enter the credentials created in Step 2

Configure your database

Enter connection and authentication details for the database you want to install Umbraco on

What type of database do you use?

Database type: SQL Server

Where do we find your database?

Server: (local)\SQL2019
Enter server domain or IP

Database name: UmbracoLoadBalar
Enter the name of the da

What credentials are used to access the database?

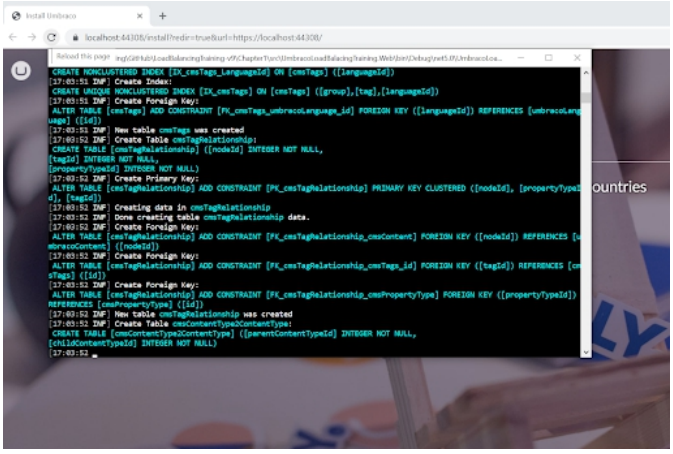
Login: UmbracoLoadBalancingTh
Enter the database user name

Password: •••••••
Enter the database pass

☐ Use integrated authentication

Go back

You should see log information about the Umbraco database being installed in the command window



7. Install Starter Kit

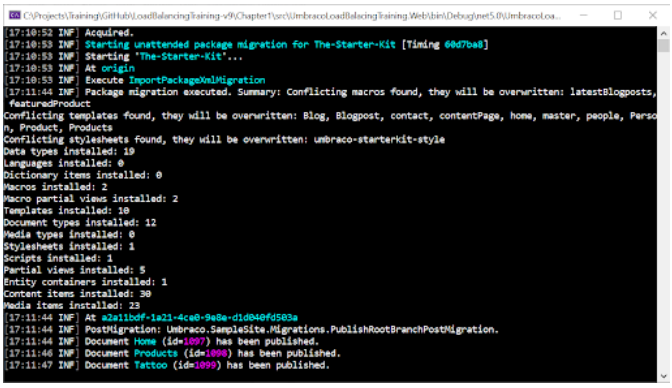
Close your browser to shut down the application

Install the starter kit package via the package manager console or dotnet command

```
Install-Package Umbraco.TheStarterKit

dotnet add package Umbraco.TheStarterKit
```

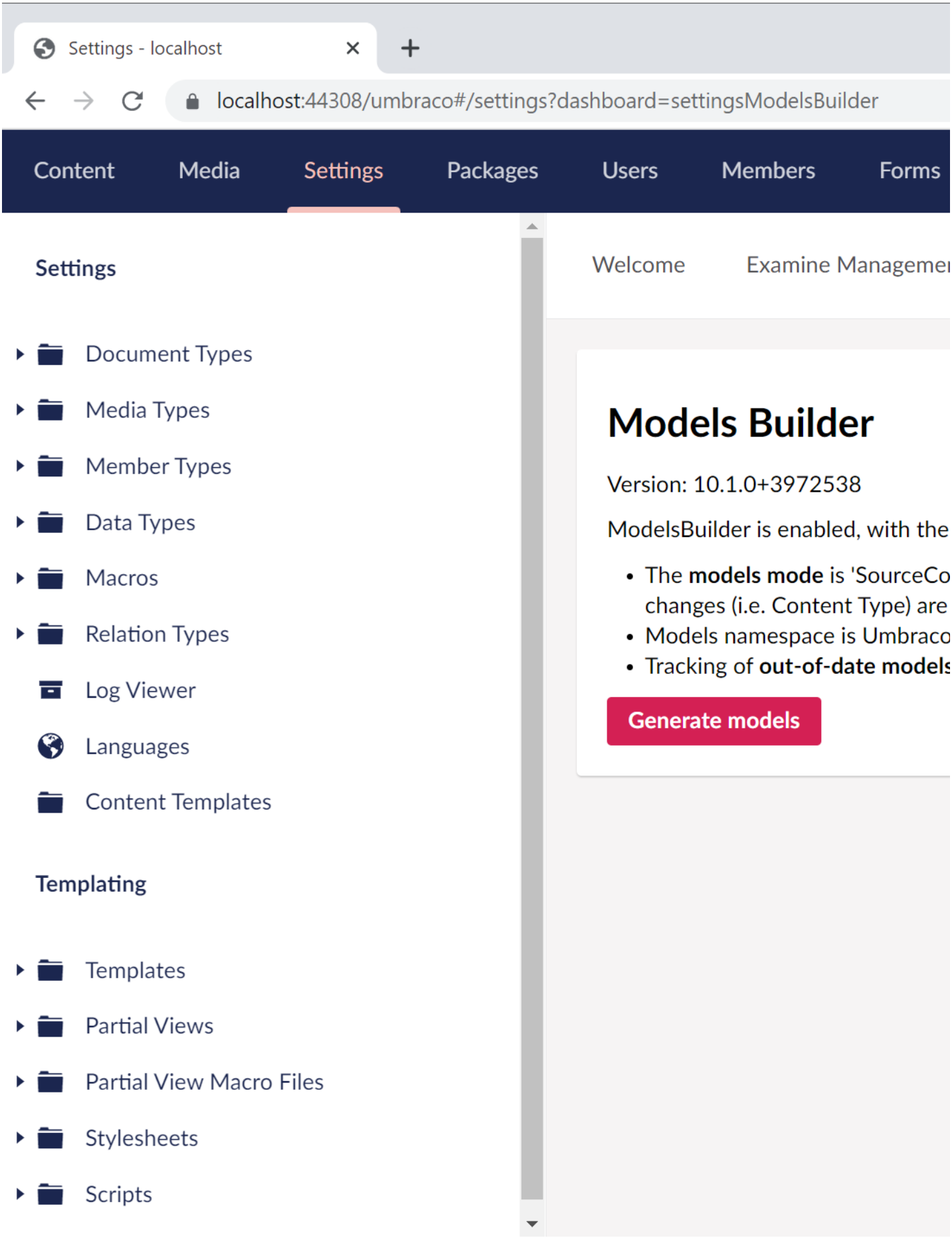
Run the UmbracoLoadBalancingTraining.Web project using the dotnet CLI, you should see output in the window showing the Starter kit installing



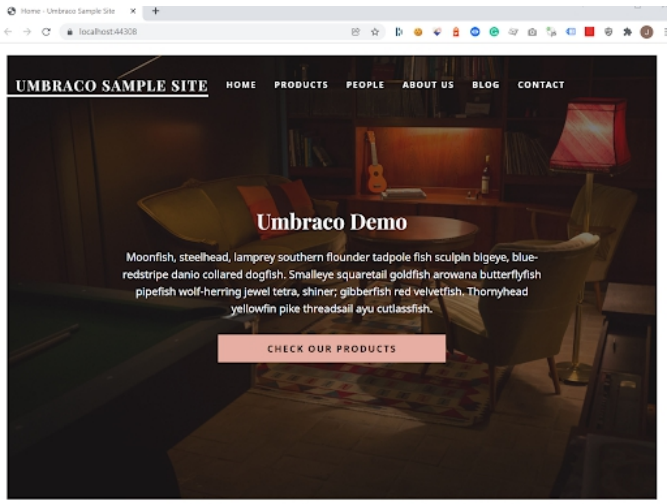
The website will error, don't worry! Go to /umbraco in your browser and login with the credentials created earlier.

8. Generate models

Generate the models in the backoffice dashboard and then stop/start the application (to compile the generated models)



You should now see the starter kit website.



Exercise 1: Session & Distributed Cache

If your application uses sessions for logins or forms then you will likely need to configure a distributed cache provider.

Additionally if using the Umbraco BeginUmbracoForm helper or Member methods then Umbrao makes use of TempData which uses Session.

In this exercise we will:

- Configure a SQL server distributed cache database
- Update Startup.cs to enable the provider
- Test that we have properly enabled the SQL distributed cache

1. In command prompt use the following command to create the Distributed Cache Database (replace the <> with your database credentials). There is an example in CopyAndPasteStuff.txt

```
dotnet tool install --global dotnet-sql-cache

dotnet sql-cache create "server=<Server>\<Instance>;database=<DBName>;user id=<UserID>;pas
sword='<Password>'" dbo DistCache
```

2. Install the NuGet Package

```
Install-Package Microsoft.Extensions.Caching.SqlServer

dotnet add package Microsoft.Extensions.Caching.SqlServer
```

3. Now you can configure the provider in the ConfigureServices method of Startup.cs, add it before the other services.

```
services.AddDistributedSqlServerCache(options =>
{
    options.ConnectionString = _config.GetConnectionString(
        "umbracoDbDSN");
    options.SchemaName = "dbo";
    options.TableName = "DistCache";
});
```

4. Test it works by adding something to the session in people.cshtml (it will error if it's not working)

Add at the top of the view

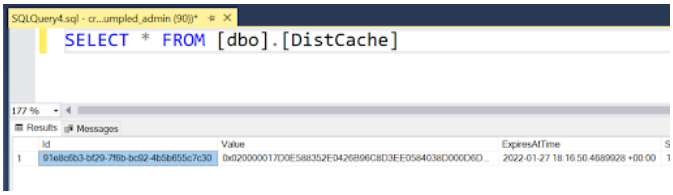
```
@using Microsoft.AspNetCore.Http
@inject IHttpContextAccessor _httpContextAccessor
```

Add within a code block

```
_httpContextAccessor.HttpContext.Session.SetString("mySessionKey", "Hello World");
```

Visit the /people page to add the key to the session.

You can also check the DistCache table in SQL Management Studio to see if a row has been created.



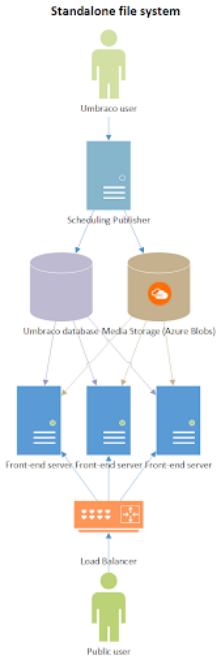
> ✪ Go further : Add code to display the session value on the master page (you will need to inject IHttpContextAccessor and add the same @using).
>
> ```
> <h3>@_httpContextAccessor.HttpContext.Session.GetString("mySessionKey")</h3>
> ```

Exercise 2: Standalone File System Simulation

In this exercise we will:

- Use a post build task in Visual Studio to publish two websites to local folders
- Create 2 x local IIS test websites with different folder paths
- Test local load balancing

This configuration is ideally suited to Azure Web Apps and other environments where it's not possible to synchronise the file system between all environments. You will need to create two websites in IIS using two different sets of files. These will simulate the same conditions as having two distinct servers.



> ✍ Note : With this configuration you cannot use the back-office to edit templates/views/partial views/macro partials/css or scripts as they will not be synchronised between front end and Umbraco servers.

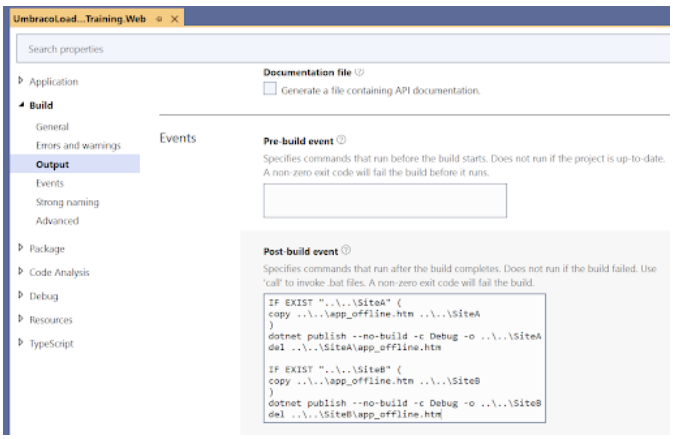
1. Add a post build event in the UmbracoLoadBalancingTraining.Web project to publish to two folders. Once enabled, rebuild the solution, this will now create a "SiteA" and a "SiteB" folder.

Everytime you build/rebuild the project, the contents of these folders will be updated.

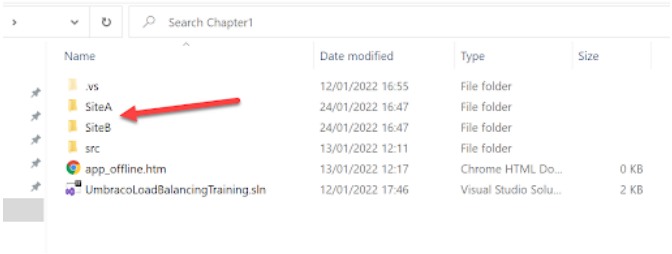
In order to unlock files whilst publishing a app_offline.htm is copied to the publish destination and then removed.

```
IF EXIST "..\..\SiteA" (
copy ..\..\app_offline.htm ..\..\SiteA
)
dotnet publish --no-build -c Debug -o ..\..\SiteA
del ..\..\SiteA\app_offline.htm

IF EXIST "..\..\SiteB" (
copy ..\..\app_offline.htm ..\..\SiteB
)
dotnet publish --no-build -c Debug -o ..\..\SiteB
del ..\..\SiteB\app_offline.htm
```



Check if the folders were created and if they contain the publish output.



Media Configuration

For this exercise the media will not automatically replicate but we can include the media folder in the solution so that the starter kit media files are published to SiteA and SiteB by editing our project file.

In the real world, you could create an Umbraco File System Provider to centralize media storage or use a package that implements one for you.

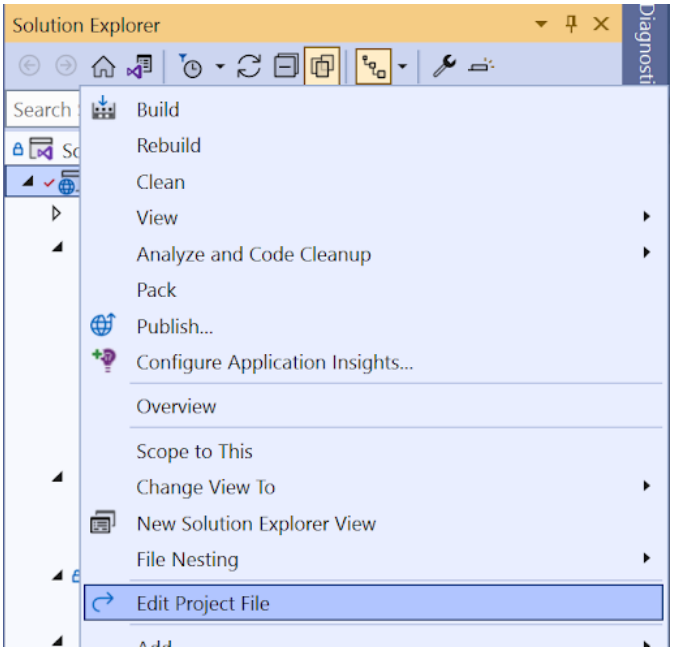
We'll see an example of this in a later exercise in Chapter 2, for now we'll focus on getting the load balancing up and running.

For Azure you can use Azure Blob Storage Provider https://our.umbraco.com/packages/developer-tools/azure-blob-storage-provider/

There is a early prototype of a Amazon S3 provider - https://github.com/DennisGlindhart/Umbraco.StorageProviders.S3

2. Edit the project file

Right click on the project and select “Edit Project File”.



Add the following snippet:

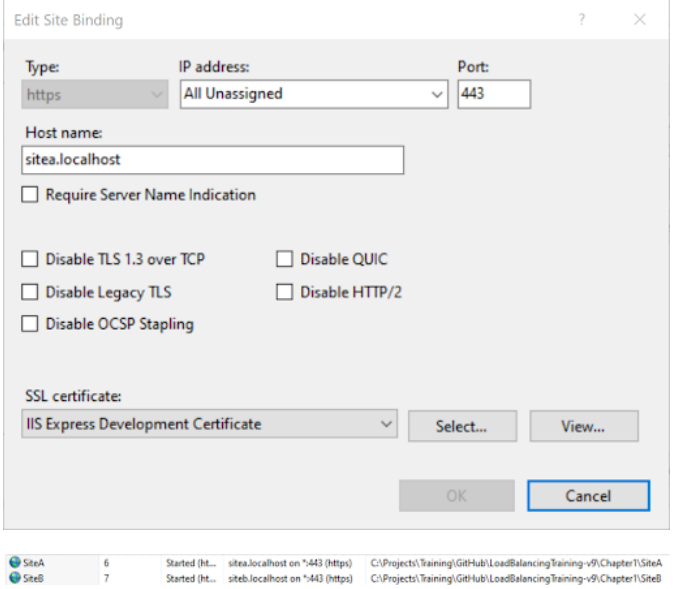
```
<ItemGroup>
  <Content Include="wwwroot\media\**\*">
    <CopyToPublishDirectory>Always</CopyToPublishDirectory>
  </Content>
</ItemGroup>
```

Rebuild your project.

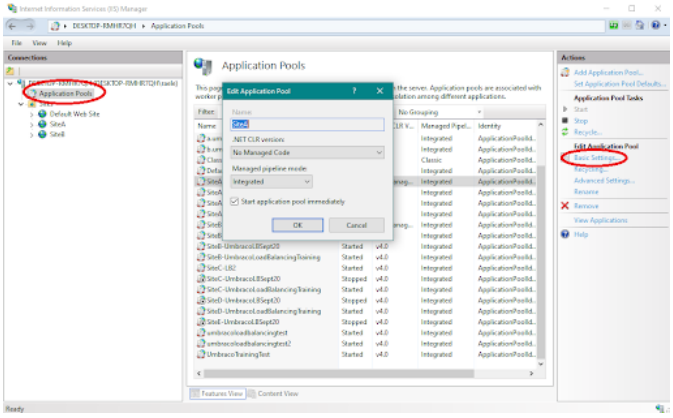
3. Create two sites in IIS (ensure you set permissions to write), using https type binding add sitea.localhost and siteb.localhost for the hostnames and use the “IIS Express Development Certificate”.

Once both sites have been added, start them up. You might need to accept the privacy warning about the certificate and continue to the site.

When using the Chrome or Edge browsers you can use any subdomain with the .localhost domain automatically, if you are using Firefox you will need to edit your hosts file.



4. Ensure you have set the application pools for both sites are set to “No Managed Code”



5. Login to Umbraco on SiteA and publish a text change, check the content updates on SiteB (you may need to keep refreshing for a while), if it does, it's working!

Exercise 3: Output Caching

In this exercise we will:

- Install Output caching NuGet Package
- Add default controller and enable output caching
- Add an event to clear the output cache on the Umbraco server

When building a website that is going to be load balanced you will likely also want to implement output caching to ensure the fastest possible page render time. To implement output caching in Asp.Net Core applications we will use the AspNetCore output caching middleware NuGet Package from https://github.com/madskristensen/WebEssentials.AspNetCore.OutputCaching.

You will now implement output caching into the load balanced website.

1. Install the Output caching NuGet package https://www.nuget.org/packages/WebEssentials.AspNetCore.OutputCaching

```
PM> Install-Package WebEssentials.AspNetCore.OutputCaching

dotnet add package WebEssentials.AspNetCore.OutputCaching
```

2. You need to enable the middleware in Startup.cs.

In the ConfigureServices method on the line before the services.AddUmbraco() add:

```
services.AddOutputCaching();
```

In the Configure method on the line before app.UseUmbraco() add:

```
app.UseOutputCaching();
```

3. Add output caching to the Index method in DefaultController.cs:

```
HttpContext.EnableOutputCaching(TimeSpan.FromMinutes(60));
```

This requires a using statement

```
using WebEssentials.AspNetCore.OutputCaching;
```

4. Enable the default controller in the ConfigureServices method of Startup.cs:

```
services.Configure<UmbracoRenderingDefaultsOptions>(c =>
{
    c.DefaultControllerType = typeof(DefaultController);
});
```

5. Build the solution.

6. Start-up SiteA & SiteB so that the cache gets populated, then publish something new, it will not change as the output cache is set to 60 minutes, so you will have to wait until it refreshes.

7. On a single server we can add a notification handler to clear the output cache when something is published. This will only refresh the output cache on the server where Umbraco is being accessed. Add the following line in the ConfigureServices method of Startup.cs before the .Build()

```
.AddNotificationHandler<ContentPublishedNotification, OutputCachePublishedHandler>()
```

8. Edit the notification handler code in OutputCachePublishedHandler.cs to remove the comments to inject IOutputCachingService. Build the solution.
9. Start-up SiteA & SiteB, this will populate the cache, then publish something new. You will notice that the content will only change on the site with which you are doing the publishing not the other one.

Exercise 4: Distributed Cache Notification Handlers

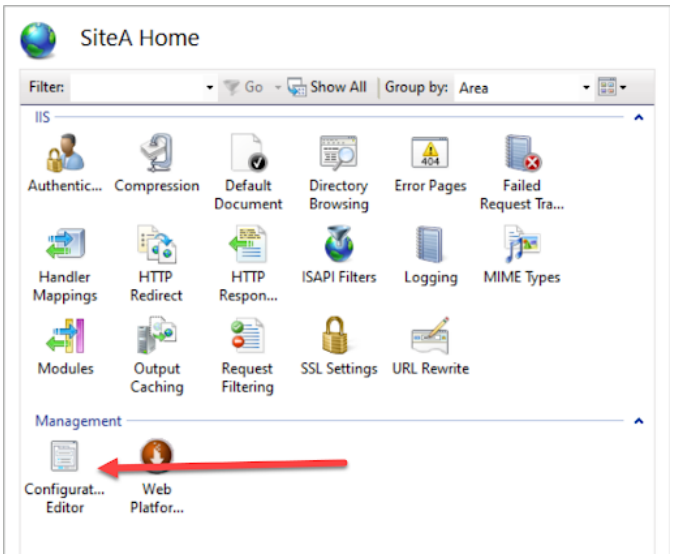
In this exercise we will:

- Add Environment Variables in IIS to identify each site name
- Replace the publish event added in Exercise 3 with cache notification handlers
- Test that the output cache is cleared on all sites after a content publish

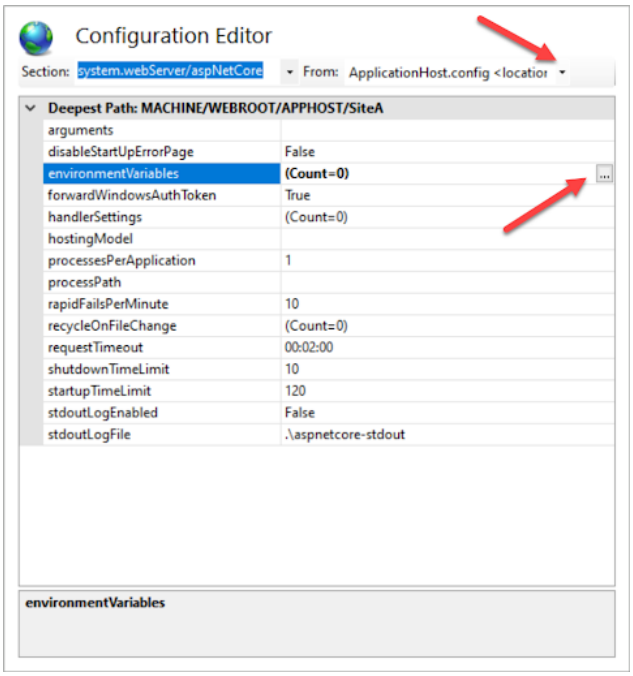
Often in custom code we use events to perform certain actions, sometimes these actions should occur on all servers that are being load balanced. For this there are some cache notifications in Umbraco that we can hook into.

In this topic we are going to adjust the output caching added in Exercise 4 to ensure that the output cache is cleared on every server, not just the server doing the publishing.

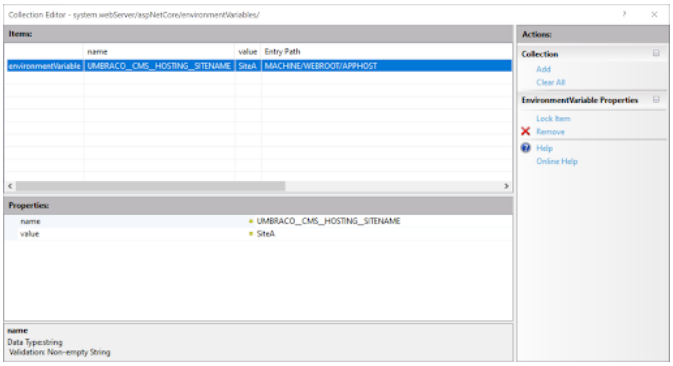
1. Add a environment variable to SiteA & SiteB. Locate the configuration editor for SiteA.



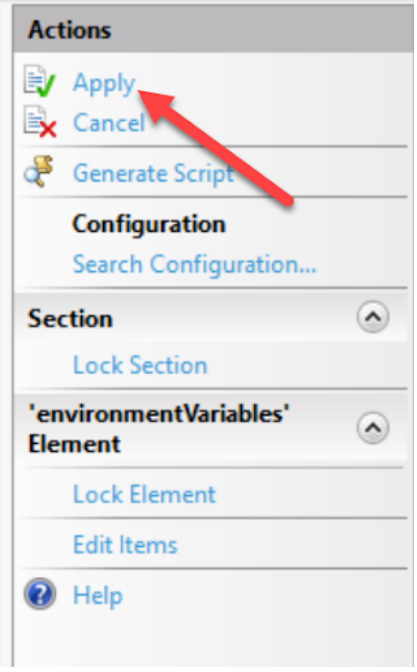
In the “Section” dropdown, select “system.webServer/aspNetCore”. Using the “From” drop down menu select “ApplicationHost.config” - this ensures the variable is stored on the system and not in web.config that is overwritten by publishing. Click on the the three dots button next to “environmentVariables”



You can now add the variable named `UMBRACO__CMS__HOSTING__SITENAME` with a value of “SiteA”



Close that window, and make sure to click “Apply” on the next one.



Repeat adding this variable for SiteB.

2. In Exercise 3 you added a notification handler to clear the output cache on the server doing the publishing but that’s not going to work for multiple servers. Instead we need to use cache notification handlers that will occur on all servers to clear the output cache. Delete or comment the notification handler added in Step 7 of Exercise 3.

```
//.AddNotificationHandler<ContentPublishedNotification, OutputCachePublishedHandler>()
```

Add the following handlers in the ConfigureServices method of Startup.cs before the .Build()

```
.AddNotificationHandler<ContentCacheRefresherNotification, OutputCacheContentCacheHandler>
()
.AddNotificationHandler<MediaCacheRefresherNotification, OutputCacheContentCacheHandler>()
.AddNotificationHandler<DictionaryCacheRefresherNotification, OutputCacheContentCacheHandl
er>()
```

Edit the notification handler code in OutputCacheContentCacheHandler.cs to remove the comments to inject IOutputCachingService, also ensure you uncomment all the instances of _outputCachingService.Clear();

Build the solution.

3. Start-up SiteA & SiteB so that the cache gets populated, then publish something new, hopefully this time the cache will get cleared on both servers and the content will show. It may take a few seconds for the non publishing site to update. Also check on the log viewer to see the entries from both websites clearing their output cache (you will have to login to Umbraco on both sites to use the log viewer).

Exercise 5: Custom Cache Refresher Notifications

In this exercise we will:

- Add a cache refresher notification
- Add a custom cache refresher notification handler that will clear the output cache
- Enable code to execute when the RefreshAll method is executed
- Test the output cache clears on all sites when a backoffice user clicks the button

Sometimes you need to be able to create your own distributed cache notifications. In this topic, we will create a dashboard button that allows the Output Cache to be cleared manually.

1. Review OutputCacheRefresherNotification.cs - the cache refresher notification (this is already included):

```
public class OutputCacheRefresherNotification : CacheRefresherNotification
{
    public OutputCacheRefresherNotification(object messageObject, MessageType messageType)
 : base(messageObject, messageType)
    {
    }
}
```

Yes, it is an empty class!

2. Review OutputCacheControlController.cs – the code to trigger the distributed event, in this case a WebApi method.

```
public class OutputCacheControlController : UmbracoAuthorizedApiController
{
    private readonly DistributedCache _distributedCache;

    public OutputCacheControlController(DistributedCache distributedCache)
    {
        _distributedCache = distributedCache;
    }

    // /umbraco/api/outputcachecontrol/clearcache
    [HttpGet]
    public bool ClearCache()
    {
        _distributedCache.RefreshAll(OutputCacheNotificationHandler.UniqueId);
        return true;
    }
}
```

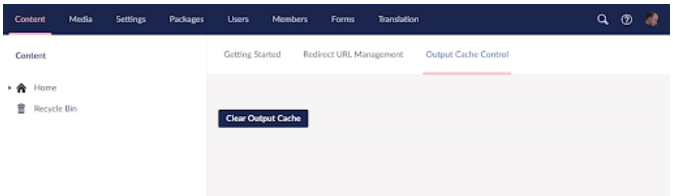
3. Edit the notification handler code in OutputCacheNotificationHandler.cs to remove the comments to inject IOutputCachingService (comment out all the handlers from Exercises 3 & 4 but don't comment the setting of the default controller added in Exercise 3, Step 4)

```
public override void RefreshAll()
{
    _outputCachingService.Clear();

    _logger.LogInformation("Output Cache Cleared (Custom) for {SiteName}",
        _hostingEnvironment.SiteName);
}
```

Make sure to build.

4. Startup SiteA & SiteB and publish some new content, it should not show on either site. Find the Output Cache Control dashboard and click the button, now the content should update on both sites (you may need to refresh a few times. If it still isn't working, restart the app pool)! Check the logs in both SiteA and SiteB again and see that the event occurred for each site.



Exercise 6: Custom ServerRoleAccessor

In this exercise we will:

- Enable a custom ServerRoleAccessor
- Check that SiteA and SiteB are being assigned different roles
- There are some scenarios where you may want to fix the SchedulingPublisher role to a particular server so that the particular server is always responsible for scheduled tasks and publishing execution.

To accomplish this we need to create a custom ServerRoleAccessor.

1. Review LoadBalancingTrainingServerRoleAccessor.cs and add new Environment Variables to SiteA & SiteB called `UMBRACO__CMS__SERVERROLE` with values of “SchedulingPublisher” for SiteA and “Subscriber” for SiteB. Ensure you select “ApplicationHost.config” when adding the variables.

2. Add the following line in the ConfigureServices method of Startup.cs before the .Build()

```
.SetServerRegistrar<LoadBalancingTrainingServerRoleAccessor>()
```

> 💡 In this example, the SchedulingPublisher server role is determined from a configuration setting. If you cannot determine what role a server should be via a condition in code then you may need to deploy separate dlls for scheduling publisher and subscriber servers, this is explained in the documentation https://our.umbraco.org/documentation/Getting-Started/Setup/Server-Setup/load-balancing/flexible-advanced

3. Edit the home.cshtml view and output the current server role

```
@using Umbraco.Cms.Core.Sync
@inject IServerRoleAccessor _serverRoleAccessor
```

Add within a code block

```
<h3>_serverRoleAccessor.CurrentServerRole: @_serverRoleAccessor.CurrentServerRole</h3>
```

Build your project.

Start up your SiteA and SiteB websites, SiteA should now be SchedulingPublisher and SiteB should now be Subscriber.

Exercise 7: Custom DatabaseServer MessengerOptions

There are some configurations where you may want to control how often the load balancing instructions from the database are processed and pruned.

Add the following in appsettings.json within the “Global” section to allow you set your own option values:

```
        "DatabaseServerMessenger": {
          "MaxProcessingInstructionCount": 1000,
          "TimeBetweenPruneOperations": "00:01:00",
          "TimeBetweenSyncOperations": "00:00:05",
          "TimeToRetainInstructions": "2.00:00:00"
        }
```

Chapter 2: Azure with Umbraco

This chapter will take you through how to configure Umbraco v10 to be hosted as a scalable Azure Web App

Getting Setup

Before we begin, we will need to set up our new resources to work on Azure with Umbraco. This first section will guide you through all the necessary components you need to have configured to continue.

1. Create Resource Group

Create an Azure Resource Group called “UmbracoAzure” and choose which Azure data centre you would like your project to be located.

Create a resource group ...

Basics Tags Review + create

Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Learn more

Project details

Subscription * Visual Studio Enterprise with MSDN

Resource group * UmbracoAzure

Resource details

Region * (Europe) West Europe

2. Create Azure Cache for Redis

Create a Azure Cache for Redis service in the Azure Portal. Ensure you select the same data centre and the Resource Group you created. Select a Basic C0 (suitable for demo projects) or better pricing tier.

Basics Networking Advanced Tags Review + create

Azure Cache for Redis helps your application stay responsive even as user load increases. It does so by leveraging the low latency, high-throughput capabilities of the Redis engine. Learn more

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Visual Studio Enterprise with MSDN

Resource group * UmbracoAzure

Create new

Instance Details

DNS name * umbracoazureunicorn

.redis.cache.windows.net

Location * West Europe

Cache type (View full pricing details) * Basic C0 (250 MB Cache, No SLA)

3. Create SQL Database

Create an SQL Database “UmbracoAzure”, you will also need to create a Server (ensure it’s in the same data centre) and make sure you note down your Server name & password

Create SQL Database ...

Microsoft

Basics Networking Security Additional settings Tags Review + create

Create a SQL database with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize. Learn more

Did you know that new users in Azure can create a free Azure SQL Database and use it for 12 months using Azure free account? Learn more

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ — Visual Studio Enterprise with MSDN

Resource group * ⓘ — UmbracoAzure

Create new

Database details

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name * — UmbracoAzure

Server * ⓘ — (new) umbracoazureunicorn (West Europe)

Create new

Want to use SQL elastic pool? ⓘ — ○ Yes ◉ No

Workload environment — ○ Development ◉ Production

Default settings provided for Production workloads. Configurations can be modified as needed.

Compute + storage * ⓘ — **Standard S2**
50 DTUs, 250 GB storage
Configure database

Backup storage redundancy

Choose how your PITR and LTR backups are replicated. Geo restore or ability to recover from regional outage is only available when geo-redundant storage is selected.

Backup storage redundancy ⓘ — ○ Locally-redundant backup storage ○ Zone-redundant backup storage ◉ Geo-redundant backup storage

Selected value for backup storage redundancy is Geo-redundant backup storage. Database backups will be geo-replicated which might impact your data residency requirements. Learn more

Create SQL Database Server

Microsoft

Server details

Enter required settings for this server, including providing a name and location. This server will be created in the same subscription and resource group as your database.

Server name * umbracoazureunicorn

.database.windows.net

Location * (Europe) West Europe

Authentication

Select your preferred authentication methods for accessing this server. Create a server admin login and password to access your server with SQL authentication, select only Azure AD authentication Learn more using an existing Azure AD user, group, or application as Azure AD admin Learn more , or select both SQL and Azure AD authentication.

Authentication method

- (●) Use SQL authentication
- ( ) Use only Azure Active Directory (Azure AD) authentication
- ( ) Use both SQL and Azure AD authentication

Server admin login * umbracoazureunicorn-admin

Password * ••••••••••••

Confirm password * ••••••••••••

Service and compute tier

Select from the available tiers based on the needs of your workload. The vCore model provides a wide range of configuration controls and offers Hyperscale and Serverless to automatically scale your database based on your workload needs. Alternately, the DTU model provides set price/performance packages to choose from for easy configuration. Learn more

Service tier Standard (For workloads with typical performance requirements)

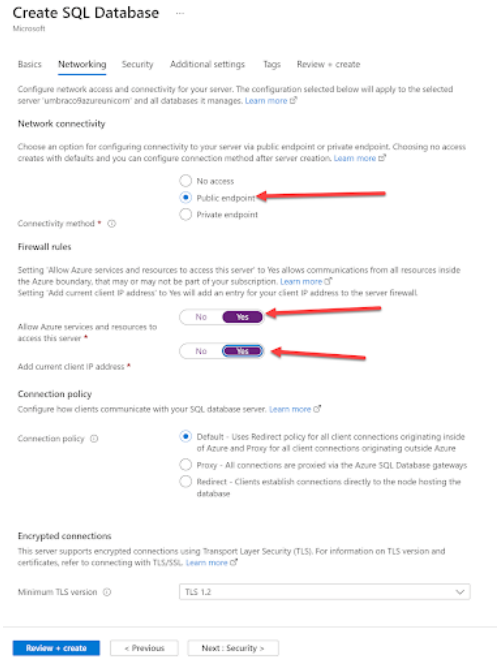
Compare service tiers

DTUs Compare DTU options

50

Data max size (GB)

250



It is recommended that you provision Umbraco with at least 50 DTUs, this can be a S2 tier or an Elastic Pool. On a free trial you can only use S1, which is fine for this course.

4. Allow IP to connect to Database

If you didn't check "Add current client IP Address" in Step 2, you will need to allow your current IP address to connect to your new SQL Azure Database.

umbracoazureunicorn | Networking
SQL server

Search (Ctrl+/)

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Quick start

Settings

- Azure Active Directory
- SQL databases
- SQL elastic pools
- DTU quota
- Properties
- Locks

Data management

- Backup
- Deleted databases
- Failover groups
- Import/Export history

Security

- Networking
- Microsoft Defender for Cloud
- Transparent data encryption
- Identity
- Auditing

Feedback

Public access | Private access | Connectivity

Public network access

Public Endpoints allow access to this resource through the internet using a public IP address. An application or resource th

Public network access: Disable / Selected networks

Connections from the IP addresses configured in the Firewall rules section bel

Virtual networks

Allow virtual networks to connect to your resource using service endpoints. Learn more

Add a virtual network rule

Rule	Virtual network	Subnet	Address range	Endpoint status	Resource group

Firewall rules

Allow certain public internet IP addresses to access your resource. Learn more

Add your client IPv4 address (77.99.29.246) | Add a firewall rule

Rule name	Start IPv4 address	End IPv4 address
ClientIp-2022-8-23_10-28-6	77.99.29.246	77.99.29.246

Exceptions

Allow Azure services and resources to access this server

5. Connect to SQL Server Management Studio

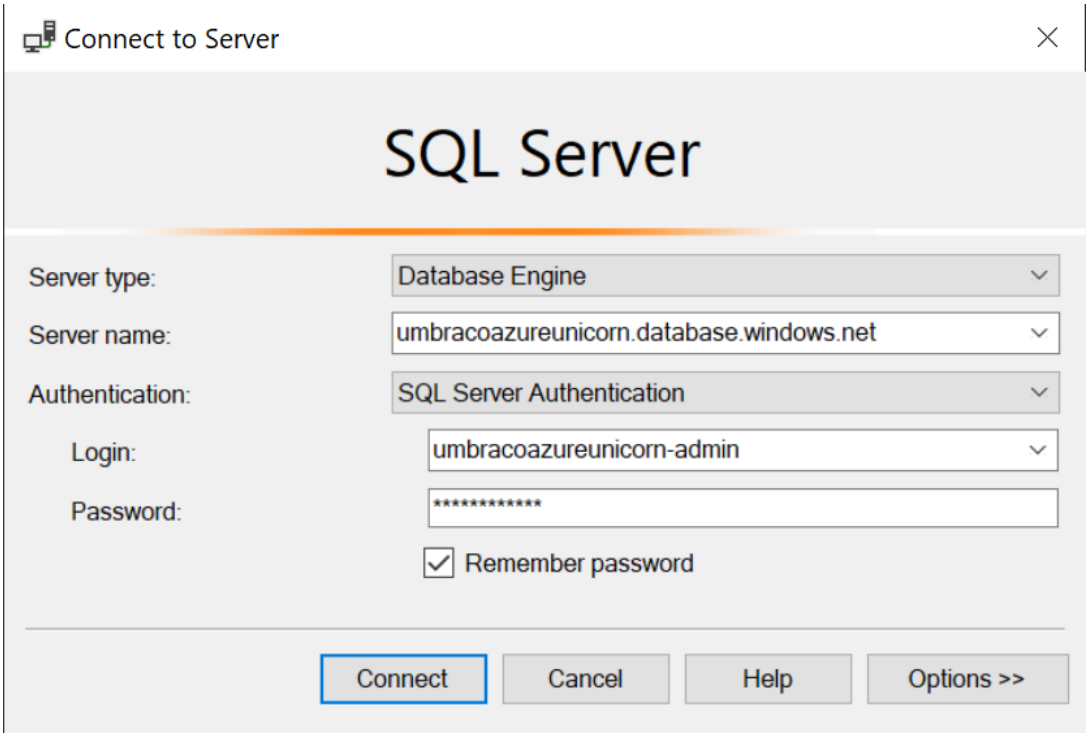
Using SQL Server Management Studio on your computer, connect to the SQL Azure database using the credentials you create in Step 2 and the database server domain from the portal:

Copy | Restore | Export | Set server firewall | Delete | Connect with... | Feedback

This database was just created. Do you need any help getting started?

Essentials

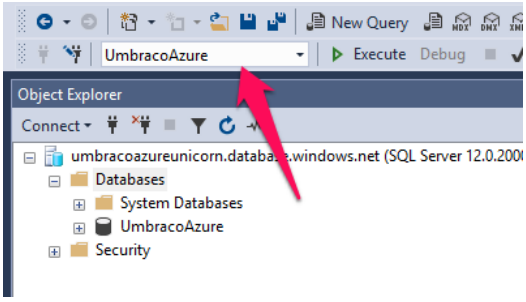
Resource group (move)	: UmbracoAzure	Server name	: umbracoazureunicorn.database.windows.n
Status	: Ready	Elastic pool	: No elastic pool
Location	: West Europe	Connection strings	: Show database connection strings
Subscription (move)	: Visual Studio Enterprise with MSDN	Pricing tier	: Standard S2: 50 DTUs
Subscription ID	: 14aced4c-cda4-49f4-a85d-cbe8cd4f430d	Earliest restore point	: 2022-08-23 09:43 UTC
Tags (edit)	: Click here to add tags		



6. Create Login

Create a Login using a new Query:

```
CREATE LOGIN UmbracoAzure WITH password='abcd1234!';
```



Switch to the database you created (instead of master)

Create a user:

```
CREATE USER UmbracoAzure FROM LOGIN UmbracoAzure;
```

Give the user db_owner permissions to the database:

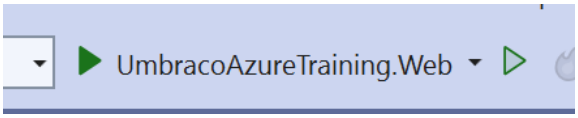
```
EXEC sp_addrolemember N'db_owner', N'UmbracoAzure'
GO
```

7. Open the Solution

Open the UmbracoAzureTraining.sln solution from the Chapter 2 folder in Visual Studio as Administrator, also open the CopyAndPasteStuffAzure.txt you will need this throughout the workbook

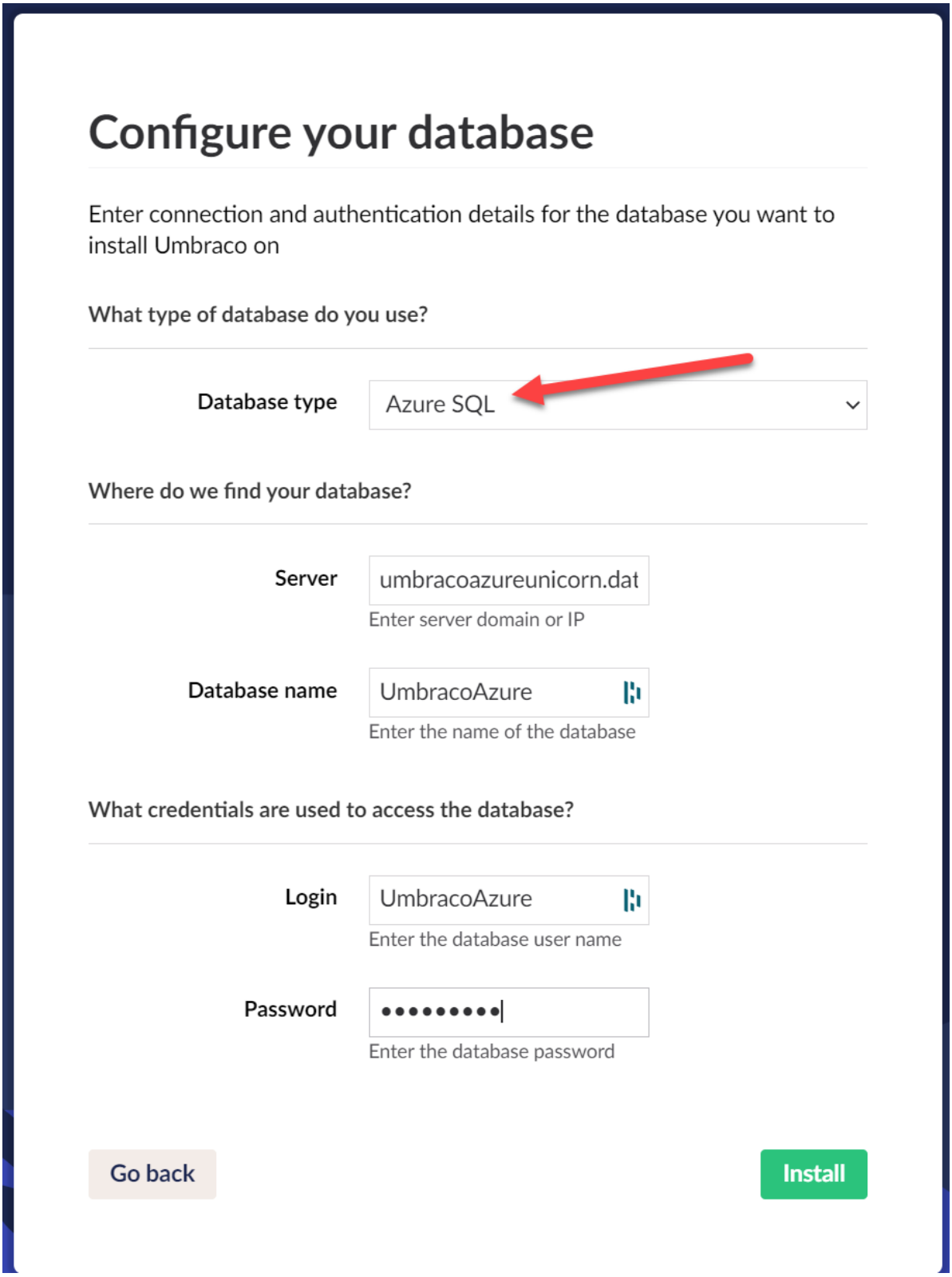
8. Run UmbracoAzureTraining.Web

Run the UmbracoAzureTraining.Web project using the dotnet CLI by selecting “UmbracoAzureTraining.Web” from the drop down.



9. Install and Customize Umbraco

Install Umbraco and ensure you choose “Change Database” when installing Umbraco and select "Azure SQL", enter the credentials created in Step 6



10. Install the Starter Kit

Close your browser to shut down the application. Install the starter kit package via the package manager console or dotnet command:

```
Install-Package Umbraco.TheStarterKit
```

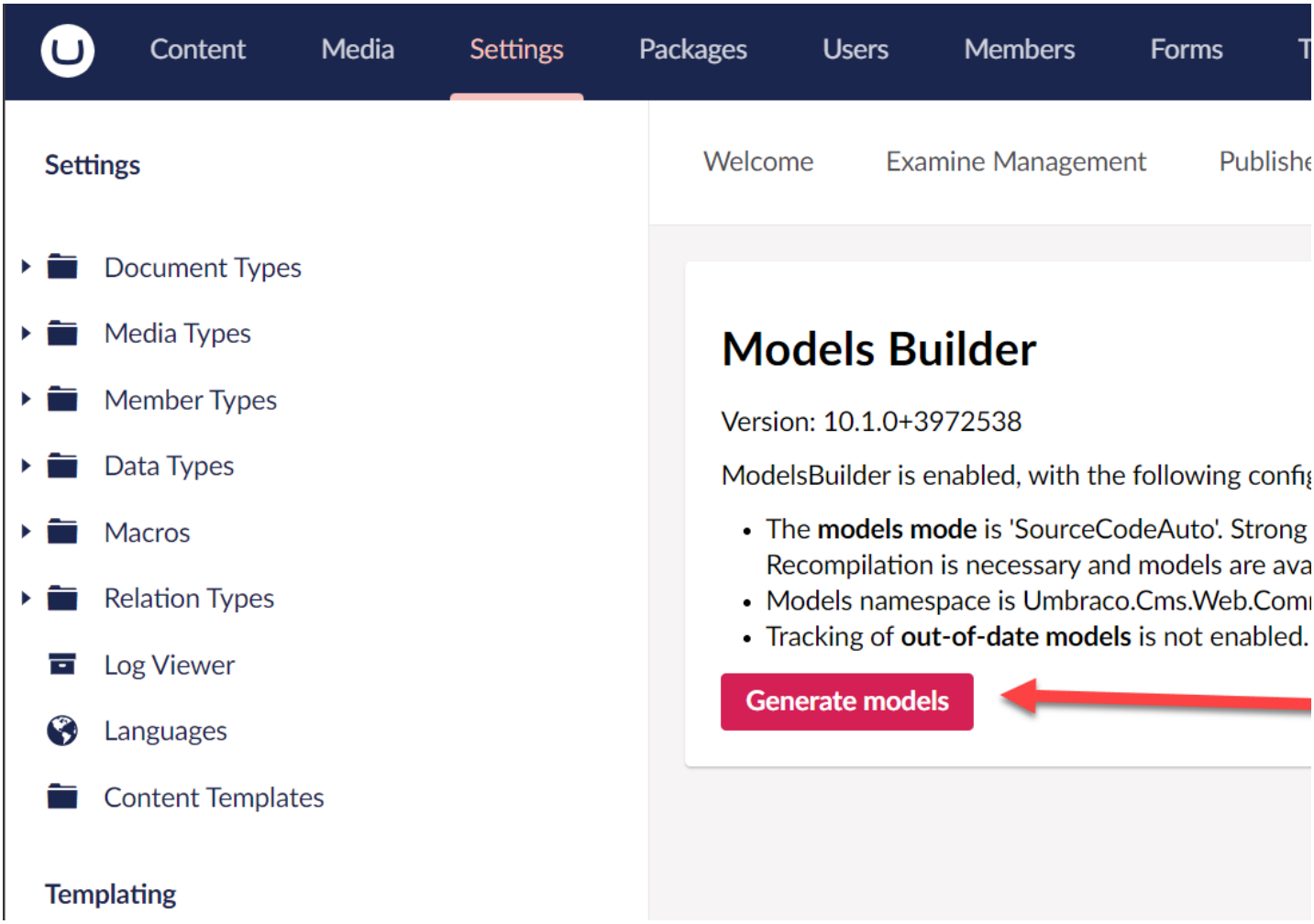
```
dotnet add package Umbraco.TheStarterKit
```

Once complete, run the UmbracoAzureTraining.Web project using the dotnet CLI.

Check the CLI Output Window to see the Starter Kit installation, once complete, login into Umbraco (the front end site will error).

11. Generate Models

Generate the models in the backoffice dashboard and then stop/start the application (to compile the generated models).



Build your project.

Exercise 1: Azure Specific Configuration

When deploying Umbraco to Azure Web Apps with multiple instances (load balancing) you will require two Web Apps: 1 x single Instance for Umbraco access & 1 x multiple Instance for the front end website.

> Warning : Don't load balance the backoffice, it's not supported and terrible things can happen if you do!

In this exercise we will:

- Configure Azure Cache for Redis as a Distributed Cache Provider
- Configure Examine to use the TempFileSystemDirectoryFactory
- Configure Umbraco to use the Environment Temporary folder to store all Umbraco temporary files

> Detailed information about this can be found at https://our.umbraco.com/Documentation/Fundamentals/Setup/Server-Setup/Load-Balancing/azure-web-apps

You can use the dotnet CLI from Visual Studio for this exercise. For Distributed caching in ASP.NET Core on Azure you will need to use the recommended provider for Redis.

> More information about the Distributed Redis Cache https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed#distributed-redis-cache

1. Configure Redis for Distributed Caching

Using the NuGet package manager or dotnet CLI:

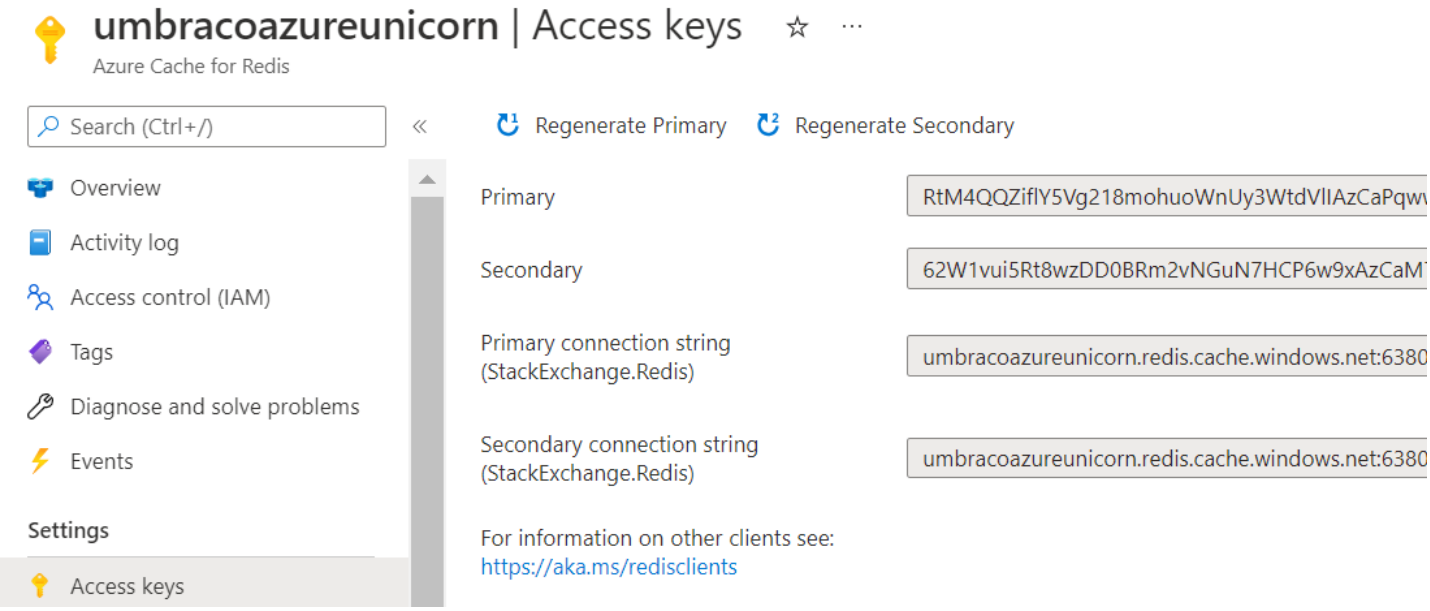
```
PM> Install-Package Microsoft.Extensions.Caching.StackExchangeRedis


dotnet add package Microsoft.Extensions.Caching.StackExchangeRedis
```

Now you can configure the provider in the ConfigureServices method of Startup.cs

```
services.AddStackExchangeRedisCache(options =>
{
    options.Configuration = _config.GetConnectionString("MyRedisConStr");
    options.InstanceName = "UmbracoTraining";
});
```

You will need to retrieve the connection string from the Azure Portal and add it to appsettings.json.



2. Check that Session is working

Test it works by adding something to the session in people.cshtml (it will error if it's not working).

Add at the top of the view:

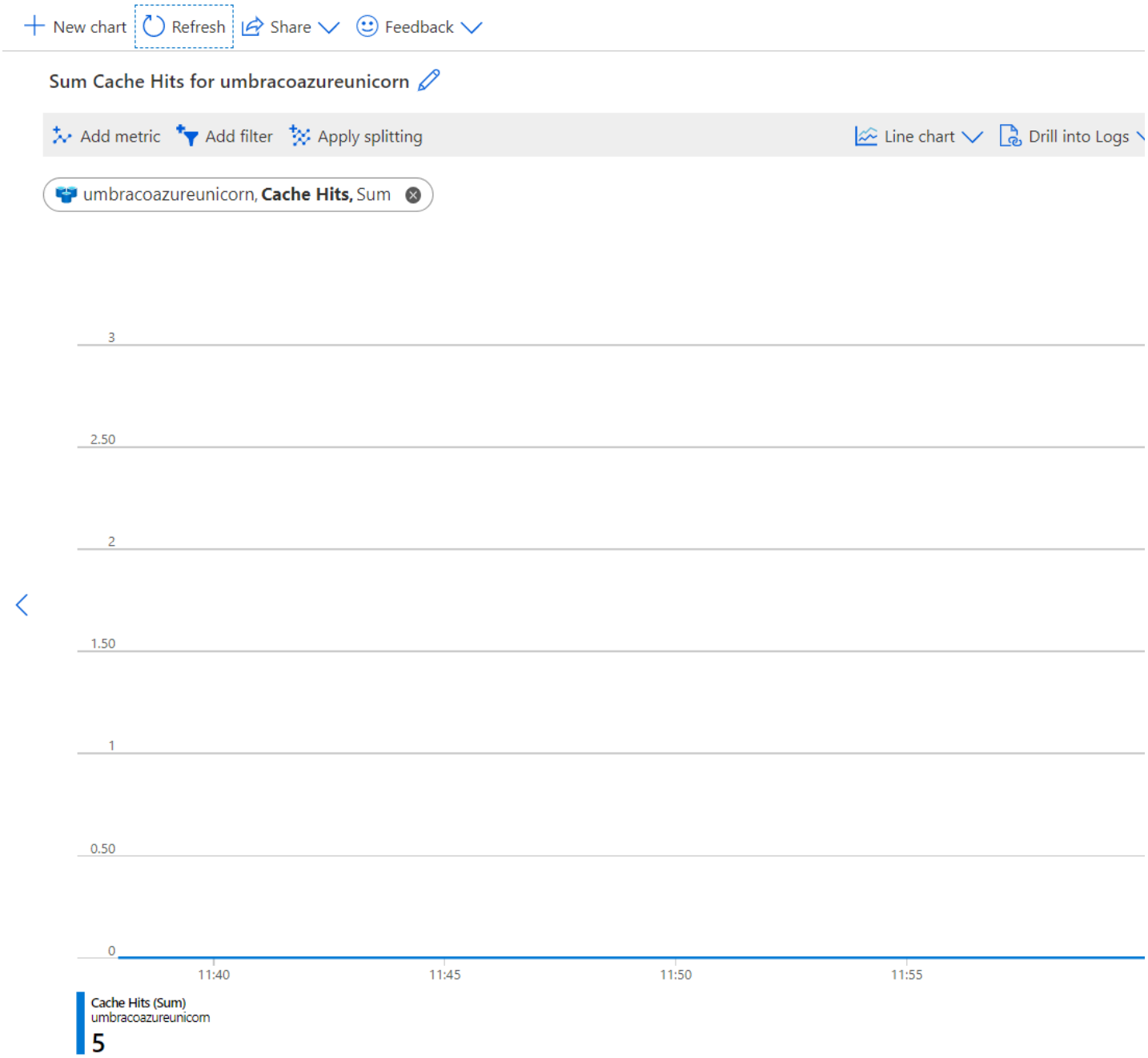
```
@using Microsoft.AspNetCore.Http
@inject IHttpContextAccessor _httpContextAccessor
```

Add within a code block:

```
_httpContextAccessor.HttpContext.Session.SetString("mySessionKey", "Hello World");
```

Visit the /people page to add the key to the session.

You should also now be able to see hits within the Metrics section, it will take a few minutes for the data to appear.



3. Configure Examine to use the TempFileSystemDirectoryFactory

Examine requires a special configuration when running with Azure Web App to avoid issues with indexes becoming locked and corrupt. To do this we will use a directory factory that will configure Examine to store it's indexes in the Environment Temporary folder.

Add the following to appsettings.json:

```
{
    "Umbraco": {
        "CMS": {
            "Examine": {
                "LuceneDirectoryFactory" : "TempFileSystemDirectoryFactory"
            }
        }
    }
}
```

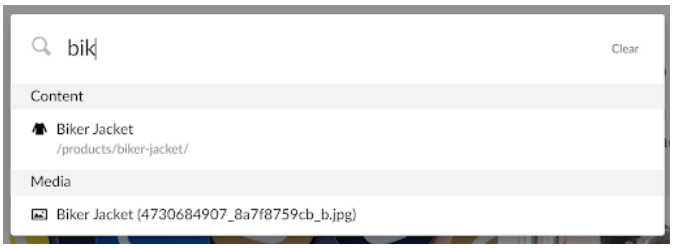
This configuration will work when developing locally so there is no need for changes when deploying to Azure.

> ✍ Note: For optimal performance, your Umbraco back-office Web App deployment (the single instance Umbraco Web App) can use a slightly different configuration, you would likely want a process to automatically transform this configuration for the back-office deployment only or use the Azure Portal managed AppSettings.

```
{
    "Umbraco": {
        "CMS": {
            "Examine": {
                "LuceneDirectoryFactory" : "SyncedTempFileSystemDirectoryFactory"
            }
        }
    }
}
```

4. Verify Examine is working

Check Examine is working by using the back-office search.



You can also check the path of the Examine indexes via the Examine dashboard.

5. Configure Umbraco to use the Environment Temporary folder

Move Umbraco temp files to the non-synced environment temp storage so that different servers don't try to share the same temporary files and end up in a horrible fight!

Add the following to appSettings.json and delete the /umbraco/Data/TEMP folder

```
{
    "Umbraco": {
        "CMS": {
            "Hosting": {
                "LocalTempStorageLocation" : "EnvironmentTemp"
            }
        }
    }
}
```

6. Configure Umbraco MainDom locking When Azure Web Apps auto transition between hosts, you scale the instances or you utilise slot swapping you may experience issues with the Umbraco Published Cache becoming locked unless your app is configured to use SQL for MainDom locking. Add the following to appsettings.json

```
{
    "Umbraco": {
        "CMS": {
            "Global": {
                "MainDomLock" : "SqlMainDomLock"
            }
        }
    }
}
```

Exercise 2: Media Blob Storage

In this exercise we will:

- Create a storage account
- Configure Umbraco to use the Media Blob Storage
- Remove local media to ensure usage from Blob Storage

1. Create Storage Account

Create a storage account in the Azure Portal in the same Resource Group.

Create a storage account ···

Basics | Advanced | Networking | Data protection | Encryption | Tags | Review

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. Learn more about Azure storage accounts

Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription * — Visual Studio Enterprise with MSDN

Resource group * — UmbracoAzure

Create new

Instance details

If you need to create a legacy storage account type, please click here.

Storage account name * — umbracoazureunicorn

Region * — (Europe) West Europe

Performance * — **Standard:** Recommended for most scenarios (general-purpose v2 account)

Premium: Recommended for scenarios that require low latency.

Redundancy * — Geo-redundant storage (GRS)

Make read access to data available in the event of regional unavailability.

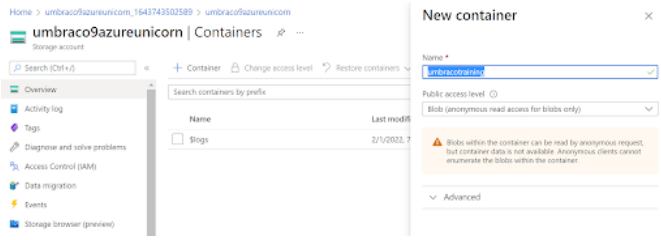
2. Install the Umbraco.StorageProviders.AzureBlob package

Using the NuGet package manager or dotnet CLI:

```
PM> Install-Package Umbraco.StorageProviders.AzureBlob


dotnet add package Umbraco.StorageProviders.AzureBlob
```

3. Create a container in the storage account



4. Add credentials to appsettings.json

Add the following to appsettings.json and add the connection string copied from the Azure Portal and container name

```
{
  "Umbraco": {
    "Storage": {
      "AzureBlob": {
        "Media": {
          "ConnectionString": "",
          "ContainerName": ""
          }
        }
      }
    }
  }
}
```

5. Configure the middleware

You need to enable the middleware in Startup.cs.

In the ConfigureServices method on the line before the .Build() add:

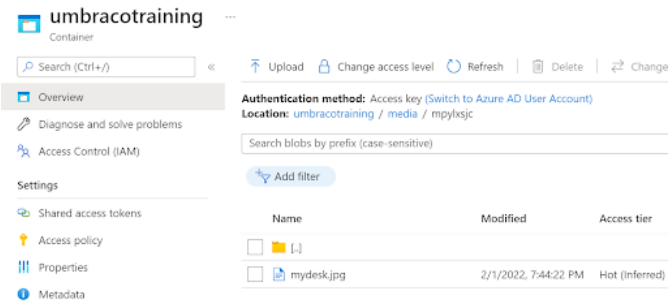
```
.AddAzureBlobMediaFileSystem()
```

6. Start the website and add a new image

Start up the website (this will ensure the container is created in the storage account). Add a new image in the media section and ensure you can view the image.

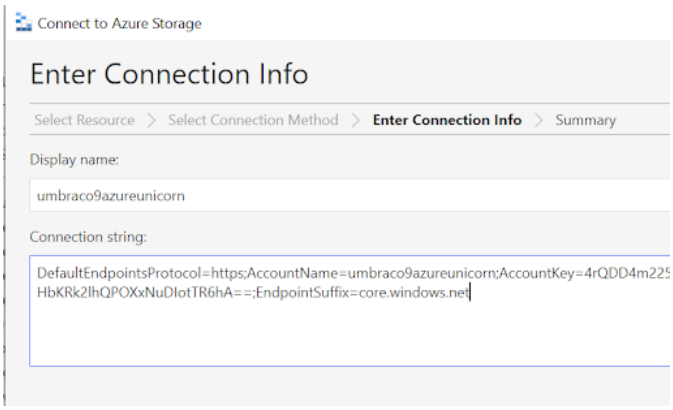
7. Verify the image is in the storage account

Navigate in the Azure Portal to the container and check the contents.

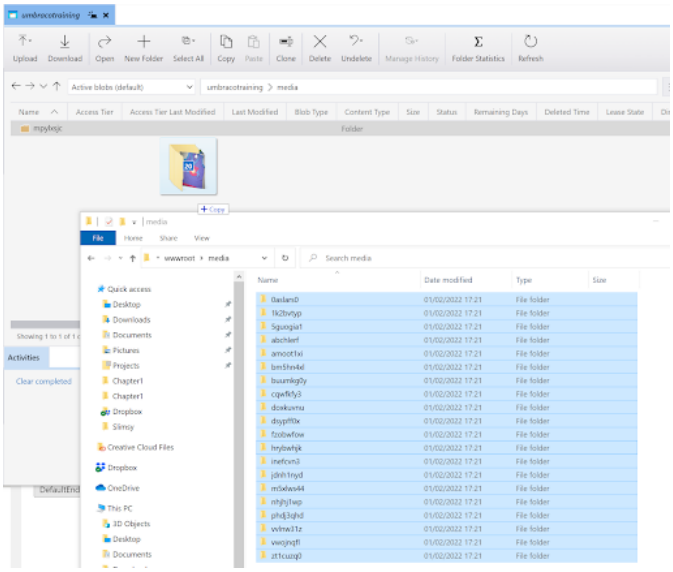


8. Install Storage Explorer and Upload Local Media

Upload existing media to blob storage, for this you will need to download and install the Microsoft Storage Explorer http://storageexplorer.com/.



Once connected, upload the contents of your local “wwwroot/media” folder into the “media” container.



9. Remove Local Media

Delete the media folder from within the wwwroot folder. Refresh the product page on the website, the images should have returned and are now stored as storage blobs.

Exercise 3: Deploying Azure Web Apps

There are many ways to deploy to Azure Web Apps such as using build services however for this training we will deploy directly from Visual Studio.

In this exercise we will:

- Create 2 x Azure Web Apps and Service Plans
- Set UmbracoApplicationUrl
- Restrict access to Umbraco from front end web apps
- Scale out the front end web app to multiple instances

1. Create two App Services (1 is for the front end & 1 is for Umbraco access), it is recommended that you use either Basic, Standard or Premium pricing tier applications as Free/Shared cannot be scaled.

You will also need to create a “App Service plan” for each app so that the front end app can be scaled out independently of the Umbraco app. Ensure all services are within the same “Resource Group”.

Select “No” for Application Insights in the Monitoring Tab (we will do this in a later exercise).

Create the app to host the front end.

Create Web App ...

Basics | Deployment | Networking | Monitoring | Tags | Review + create

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. Learn more

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * : Visual Studio Enterprise with MSDN

Resource Group * : UmbracoAzure
Create new

Instance Details

Need a database? Try the new Web + Database experience.

Name * : umbracoazure-jl-web
.azurewebsites.net

Publish * : (•) Code ( ) Docker Container ( ) Static Web App

Runtime stack * : .NET 6 (LTS)

Operating System * : ( ) Linux (•) Windows

Region * : West Europe
Not finding your App Service Plan? Try a different region or select your App Service Environment.

App Service Plan

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. Learn more

Windows Plan (West Europe) * : (New) umbracoazure-jl-web
Create new

Sku and size * : **Standard S1**
100 total ACU, 1.75 GB memory
Change size

Zone redundancy

An App Service plan can be deployed as a zone redundant service in the regions that support it. This is a deployment time only decision. You can't make an App Service plan zone redundant after it has been deployed Learn more

Zone redundancy
( ) **Enabled:** Your App Service plan and the apps in it will be zone redundant. The minimum App Service plan instance count will be three.
(•) **Disabled:** Your App Service Plan and the apps in it will not be zone redundant. The minimum App Service plan instance count will be one.

Create the app to host Umbraco.

Create Web App ...

Basics | Deployment | Networking | Monitoring | Tags | Review + create

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. Learn more

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * : Visual Studio Enterprise with MSDN

Resource Group * : UmbracoAzure
Create new

Instance Details

Need a database? Try the new Web + Database experience.

Name * : umbracoazure-jl-umbraco
.azurewebsites.net

Publish * : (•) Code ( ) Docker Container ( ) Static Web App

Runtime stack * : .NET 6 (LTS)

Operating System * : ( ) Linux (•) Windows

Region * : West Europe
Not finding your App Service Plan? Try a different region or select your App Service Environment.

App Service Plan

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. Learn more

Windows Plan (West Europe) * : (New) umbracoazure-jl-umbraco
Create new

Sku and size * : **Standard S1**
100 total ACU, 1.75 GB memory
Change size

Zone redundancy

An App Service plan can be deployed as a zone redundant service in the regions that support it. This is a deployment time only decision. You can't make an App Service plan zone redundant after it has been deployed Learn more

Zone redundancy
( ) **Enabled:** Your App Service plan and the apps in it will be zone redundant. The minimum App Service plan instance count will be three.
(•) **Disabled:** Your App Service Plan and the apps in it will not be zone redundant. The minimum App Service plan instance count will be one.

2. Add the following to the top of the home.cshtml View so we will be able to see the server name that is serving the page later

```
<h3>Environment.MachineName: @Environment.MachineName</h3>
<h3>Environment.GetEnvironmentVariable("WEBSITE_INSTANCE_ID"): @Environment.GetEnvironment
Variable("WEBSITE_INSTANCE_ID")</h3>
```

3. Set UmbracoApplicationUrl

Add the following to appsettings.json and set the value to the Url of your Umbraco web App

```
{
    "Umbraco": {
        "CMS": {
            "WebRouting": {
                "UmbracoApplicationUrl": ""
            }
        }
    }
}
```

e.g.

```
"WebRouting": {
  "UmbracoApplicationUrl": "https://umbracoazure-jl-umbraco.azurewebsites.net/"
}
```

4. Enable the ServerRole Accessor

Add the following line in the ConfigureServices method of Startup.cs before the .Build()

```
.SetServerRegistrar<LoadBalancingTrainingServerRoleAccessor>()
```

Edit the home.cshtml view and output the current server role

```
@using Umbraco.Cms.Core.Sync
@inject IServerRoleAccessor _serverRoleAccessor
```

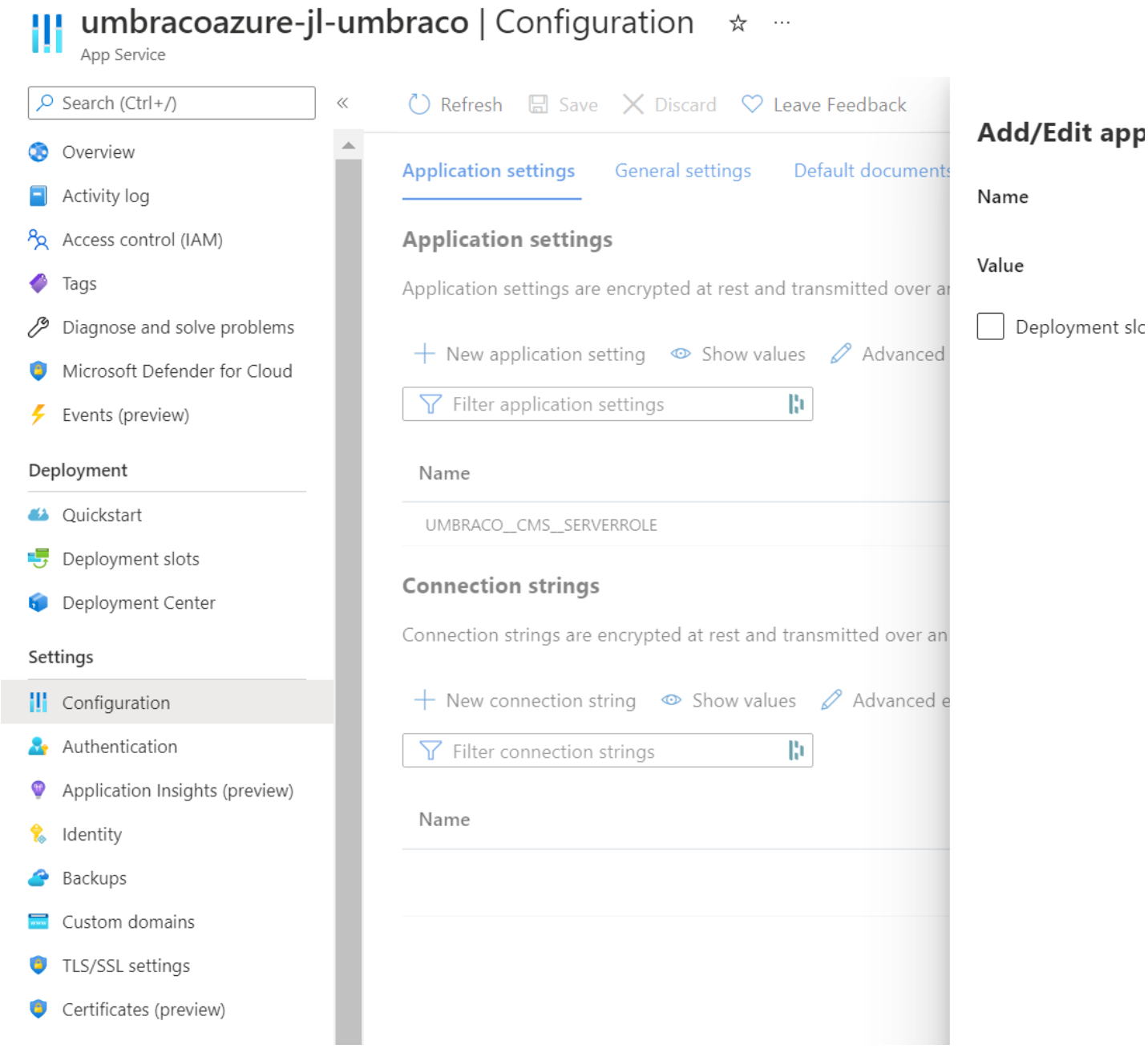
Add within a code block

```
<h3>_serverRoleAccessor.CurrentServerRole: @_serverRoleAccessor.CurrentServerRole</h3>
```

Review the following in appsettings.json within the “CMS” section, this will set the default role to Subscriber

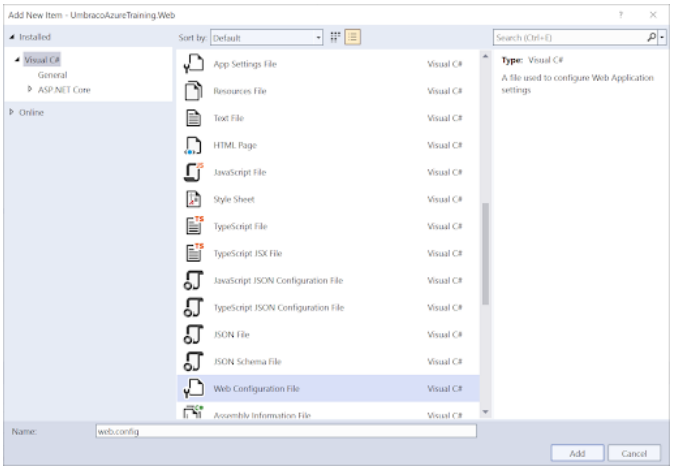
```
"CMS": {
  "ServerRole" : "Subscriber"
}
```

Override the default ServerRole for the Umbraco Web App by adding an Application Setting in the Azure Portal. Add a setting called `UMBRACO__CMS__SERVERROLE` with a value of `SchedulingPublisher`. Ensure you “Save”.



5. As you now have two Web Apps and Umbraco should only be accessed via the single instance Umbraco specific Web App you can optionally add a IIS rewrite rule to prevent Umbraco being accessed via the multi instance Front End Web App.

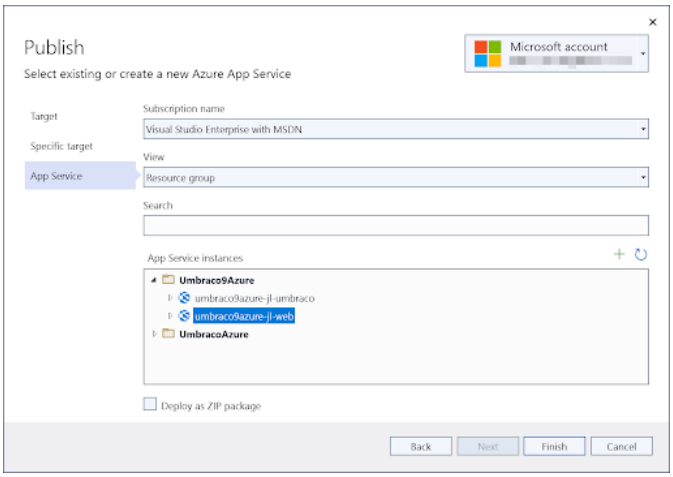
Add a web.config to the project:



Add the below snippet within the configuration element adjusting the “MYSITE-UMBRACO” URL to your own.

```
<system.webServer>
  <rewrite>
    <rules>
      <!-- Restrict access to Umbraco -->
      <rule name="Restrict access" stopProcessing="true">
        <match url="umbraco$|umbraco/(?!surface\/)(?!api\/)" />
        <conditions logicalGrouping="MatchAll" trackAllCaptures="false">
          <add input="{HTTP_HOST}" pattern="(([^.]+)\.)?MYSITE-UMBRACO\.azurewebsites\.net
" negate="true" />
          <add input="{HTTP_HOST}" pattern="(([^.]+)\.)?localhost" negate="true" />
        </conditions>
        <action type="CustomResponse" statusCode="404"/>
      </rule>
    </rules>
  </rewrite>
</system.webServer>
```

6. Right click on the UmbracoAzureTraining.Web project in the Solution Explorer and select “Publish”, then choose “Microsoft Azure App Service”. Now choose your Subscription and select Resource Group for the View”



umbracoazure-jl-web - Web Deploy.pubxml

Azure App Service (Windows)

New More actions

Ready to publish.

Settings

Configuration	Release
Target Framework	net6.0
Deployment Mode	Framework-dependent
Target Runtime	Portable

Show all settings

Hosting

Subscription	14aced4c-cda4-49f4-a85d-cbe8cd4f430d
Resource group	UmbracoAzure
Resource name	umbracoazure-jl-web

Site: https://umbracoazure-jl-web.azurewebsites.net

Service Dependencies

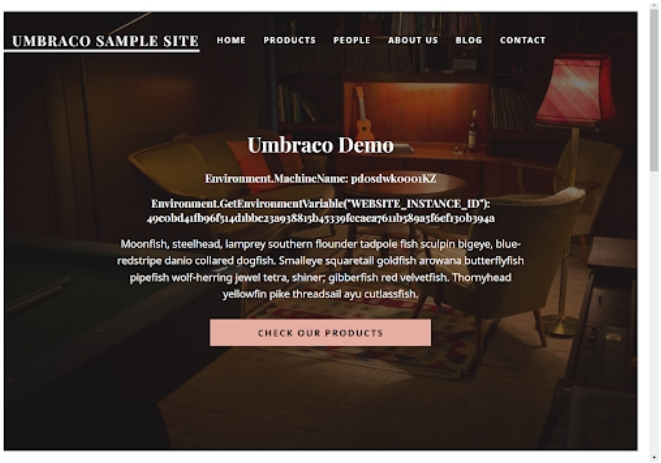
SQL Server Database

Connection string name: ConnectionStrings:umbracoDbDSN

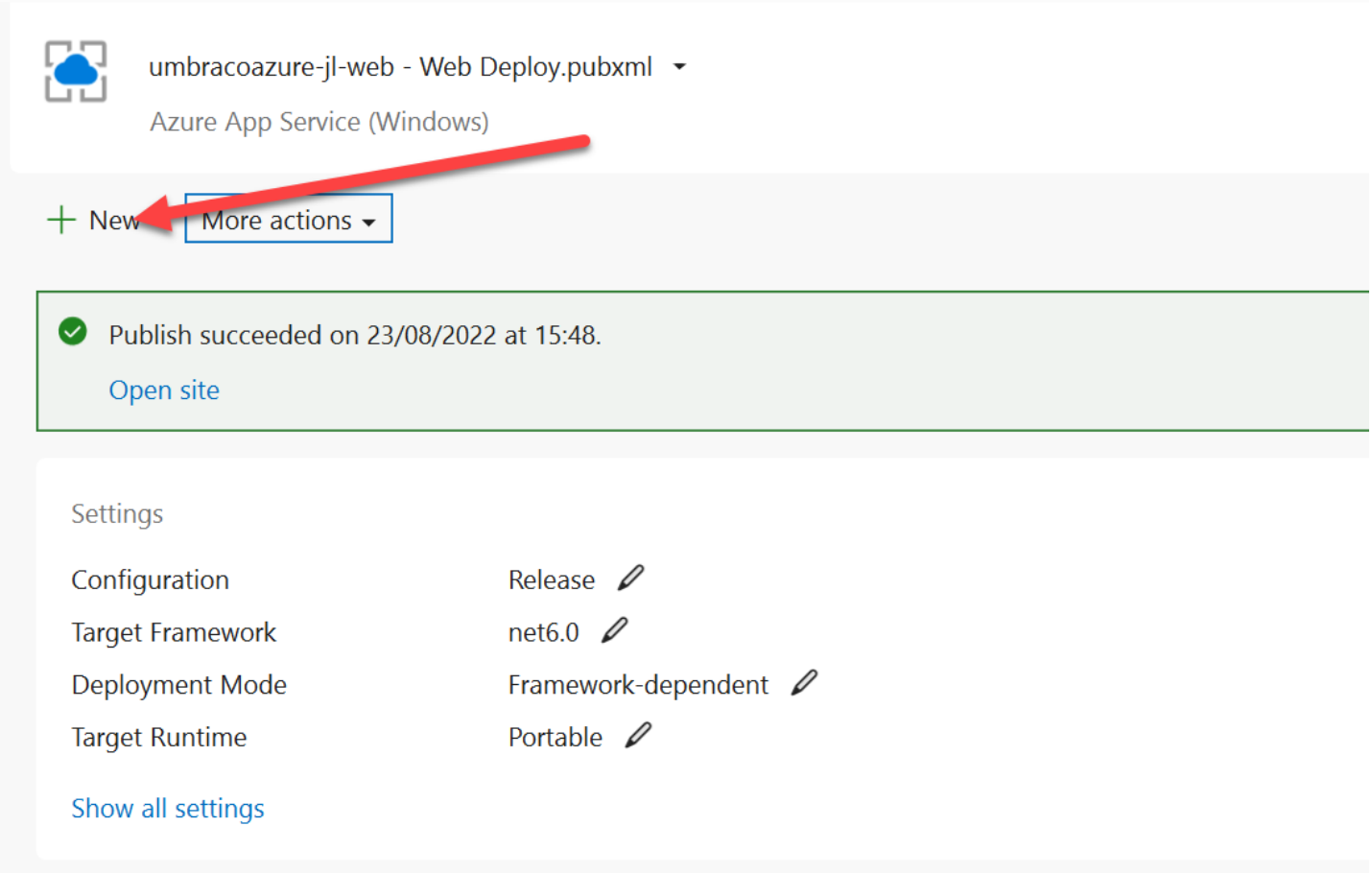
Storage

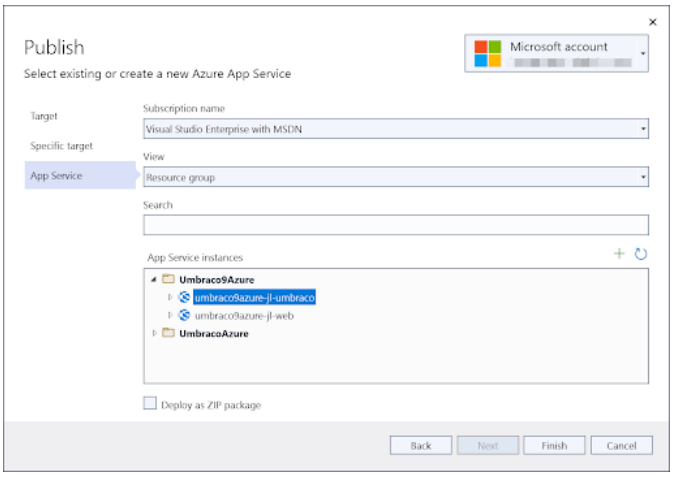
Connection string name: Umbraco:Storage:AzureBlob:Media:ConnectionString

7. Visual Studio should open the website url and you will see the homepage looking something like this!



8. Create a “New Profile” and repeat Steps 6 & 7 for the Umbraco Web App





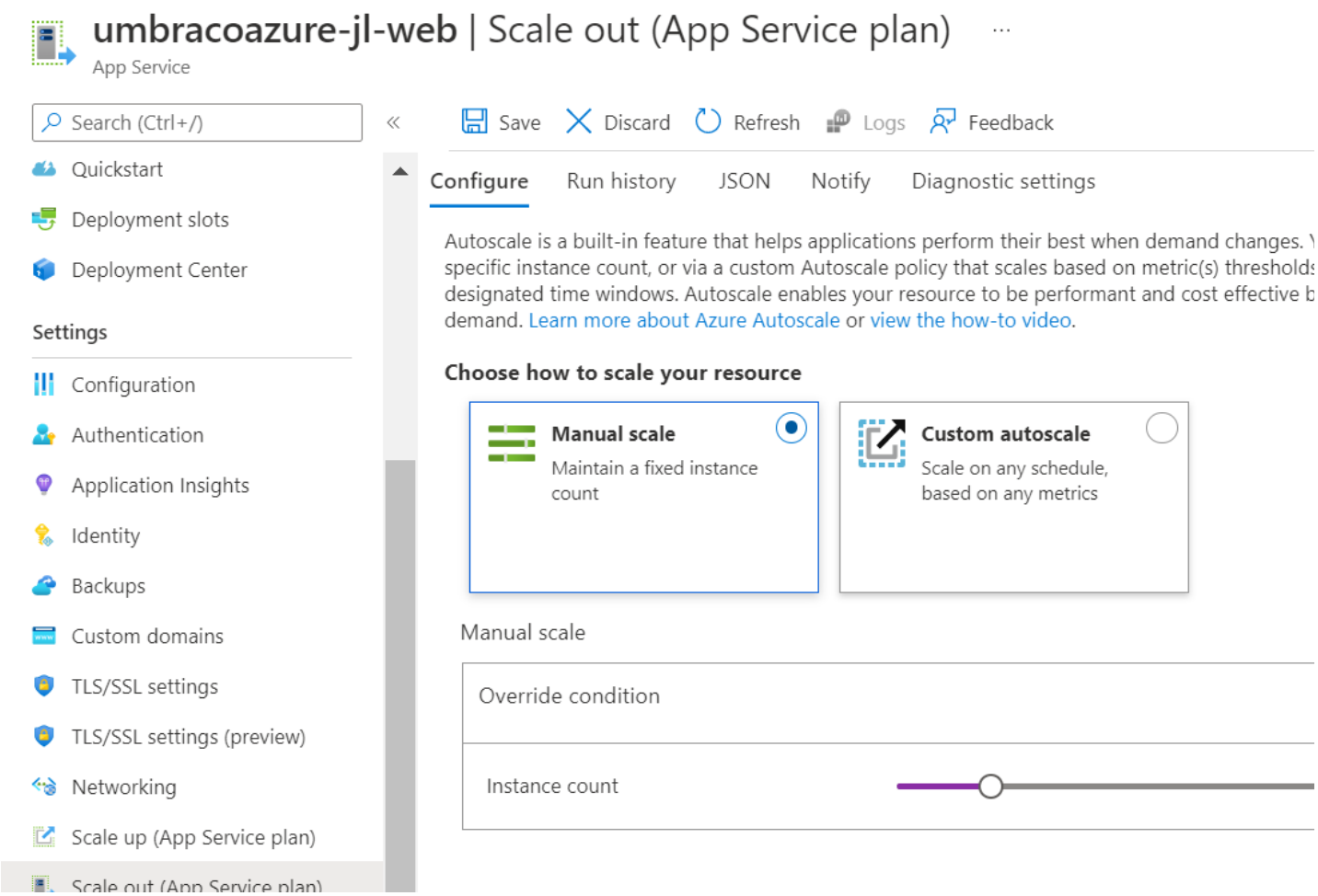
9. Log into Umbraco via the Umbraco Web App (this should be the only Url that you access Umbraco from now)

e.g. https://umbracoazure-jl-umbraco.azurewebsites.net/umbraco/

Perform some publishing of content and media and ensure it appears on the front end website

e.g. http://umbracoazure-jl-web.azurewebsites.net/

10. Scale out the instances running the front end website app to at least 2.



If you are using a Basic Tier Web App you can only scale manually (up to 3 instances), if you are using a Standard Tier Web App you scale automatically based on performance metrics such as CPU usage by clicking “Custom autoscale”

11. Disable AAR affinity

Disabling the affinity cookie by navigating to the “Configuration” section of your Web App, this will stop the load balancer making sticky user sessions.

umbracoazure-jl-web | Configuration

App Service

Search (Ctrl+/)

Quickstart
Deployment slots
Deployment Center

Settings

Configuration
Authentication
Application Insights
Identity
Backups
Custom domains
TLS/SSL settings
TLS/SSL settings (preview)
Networking
Scale up (App Service plan)
Scale out (App Service plan)
WebJobs
Push
MySQL In App
Service Connector
Properties
Locks

App Service plan

App Service plan
Quotas
Change App Service plan

Development Tools

Refresh | Save | Discard | Leave Feedback

Application settings | General settings * | Default documents | Path mappings

Stack settings

Stack: .NET

.NET version: .NET 6 (LTS)

Platform settings

Platform: 64 Bit

Managed pipeline version: Integrated

FTP state: Disabled

FTP based deployment can be disabled or configured t

HTTP version: 2.0

Web sockets: On / Off

Always on: On / Off

Prevents your app from being idled out due to inactivit

ARR affinity: On / Off

Improve performance of your stateless app by turning

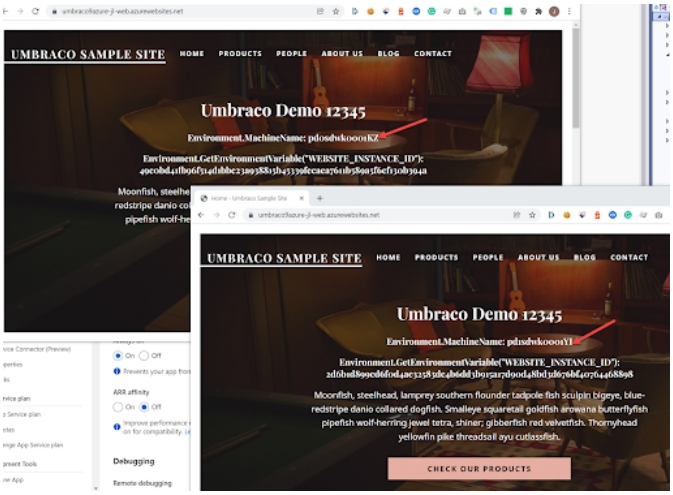
HTTPS Only: On / Off

Enable this feature to redirect all HTTP traffic to HTTPS

Minimum TLS Version: 1.2

Select the minimum TLS encryption version required b

12. Open a few different browser windows to see if you can get one serving from each server (check the machine name) and then do some publishing and ensure everything updates accordingly.



Exercise 4: Application Insights

Application Insights is an application performance monitoring service, it will automatically collect data and logs from your application but in order for it to have access to Umbraco logs we need a way of shipping them to it.

Without using Application Insights to collect Umbraco Logs it would be difficult to view logs being generated by the front end web application.

In this exercise we will:

- Create an Application Insights service
- Configure your solution to use the Application Insights service
- Add Umbraco logs to Application Insights

1. Create an Application Insights service within your Resource Group

Application Insights ...

Monitor web app performance and usage

Basics Tags Review + create

Create an Application Insights resource to monitor your live web application. With Application Insights, you have full observability into your application across all components and dependencies of your complex distributed architecture. It includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. It's designed to help you continuously improve performance and usability. It works for apps on a wide variety of platforms including .NET, Node.js and Java EE, hosted on-premises, hybrid, or any public cloud. Learn More

PROJECT DETAILS

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * : Visual Studio Enterprise with MSDN

Resource Group * : UmbracoAzure

Create new

INSTANCE DETAILS

Name * : UmbracoAzure

Region * : (Europe) West Europe

Resource Mode * : Classic | Workspace-based

WORKSPACE DETAILS

Subscription * : Visual Studio Enterprise with MSDN

*Log Analytics Workspace : DefaultWorkspace-14aced4c-cda4-49f4-a85d-cbe8cd4f430d-WEU [weste...

2. Find the connection string for your Application Insights

Search (Ctrl+/)

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems

Investigate

Application map
Smart detection

Application Dashboard | Getting started | Search | Logs | Monitor resou

Essentials

Resource group (move) : UmbracoAzure
Location : West Europe
Subscription (move) : Visual Studio Enterprise with MSDN
Subscription ID : 14aced4c-cda4-49f4-a85d-cbe8cd4f430d
Tags (edit) : Click here to add tags

Show data for last: 30 minutes | 1 hour | 6 hours | 12 hours | 1 day | 3 days

Failed requests | Server response time

3. Using the NuGet package manager or dotnet CLI:

```
PM> Install-Package Microsoft.ApplicationInsights.AspNetCore


dotnet add package Microsoft.ApplicationInsights.AspNetCore
```

4. In the ConfigureServices method of Startup.cs add

```
services.AddApplicationInsightsTelemetry();
```

In the appsettings.json add the below snippet adding in your connection string from Step 2

```
"ApplicationInsights": {
  "ConnectionString": ""
}
```

5. Install Serilog.Sinks.ApplicationInsights

Using the NuGet package manager or dotnet CLI:

```
PM> Install-Package Serilog.Sinks.ApplicationInsights


dotnet add package Serilog.Sinks.ApplicationInsights
```

6. Replace the Serilog section in appsettings.json with the below snippet, add the connectionString.

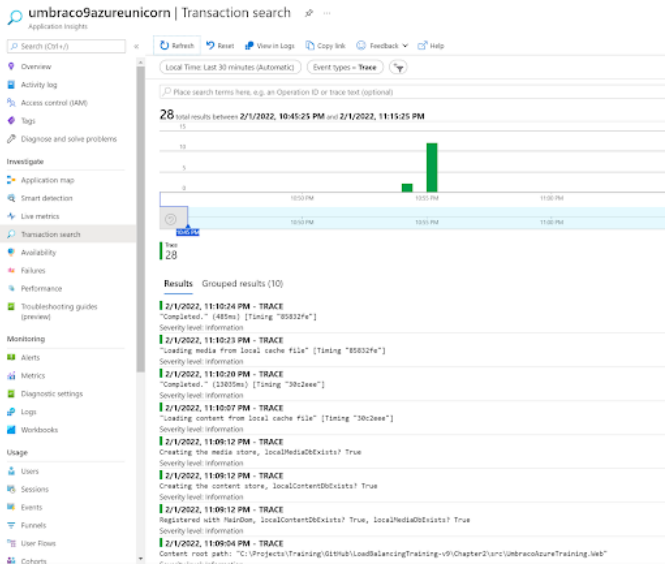
This will enable Umbraco logs to be written to Application Insights. This also sets the default UmbracoFile sink to only log “Error” level.

```
  "Serilog": {
    "Using": [
      "Serilog.Sinks.ApplicationInsights"
    ],
    "MinimumLevel": {
      "Default": "Information",
      "Override": {
        "Microsoft": "Warning",
        "Microsoft.Hosting.Lifetime": "Information",
        "System": "Warning"
      }
    },
    "WriteTo": [
      {
        "Name": "ApplicationInsights",
        "Args": {
          "restrictedToMinimumLevel": "Information",
          "connectionString": "",
          "telemetryConverter": "Serilog.Sinks.ApplicationInsights.TelemetryConverters.Tra
ceTelemetryConverter, Serilog.Sinks.ApplicationInsights"
        }
      },
      {
        "Name": "Async",
        "Args": {
          "configure": [
            {
              "Name": "Console"
            }
          ]
        }
      }
    ]
  }
```

Delete the entire Serilog section from appsettings.Development.json - the default configuration will prevent your local computer from using the new sink.

Run the project using the dotnet CLI to generate some logs.

7. Goto “Transaction search” within Application insights and filter by “Trace”, you should now see Umbraco Logs, it may take a few minutes for them to appear.



> ✪ Go further: Deploy to both Umbraco and Front end Web Apps.

Exercise 5: Accessing Temp Files

The setting we enabled in Exercise 1 moves Umbraco temporary files into the Azure Web App temporary folder, occasionally it's helpful to view these files and folders for debugging issues.

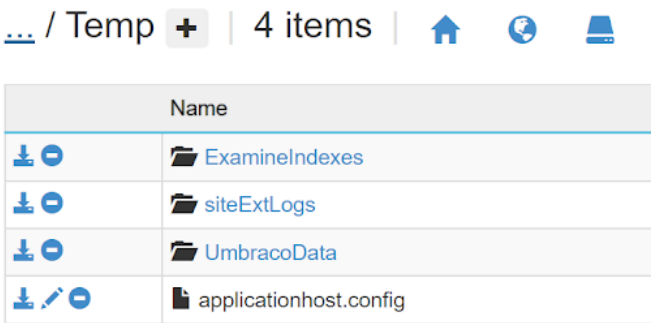
By default it is not possible to do this, we need to add an environmental variable to make this possible.

In this exercise we will:

- Add an Application Setting in the Azure portal
- Browse the temp files in Kudu

> ✍ Known issue: With `WEBSITE_DISABLE_SCM_SEPARATION` set to true WebDeploy may not be able to successfully deploy to the Web App slot.

1. Within the Azure portal navigate to your web application and then select "Configuration" and add an application setting `WEBSITE_DISABLE_SCM_SEPARATION` with the value `true` and Save
2. Within the Azure portal go to "Advanced Tools" and click "Go", this will launch Kudu
3. Within Kudu select "Debug Console" and then enter `cd..` to change the folder up to the root level



Browse to "local" and then "temp", here you should find "ExamineIndexes" & "UmbracoData" folders.

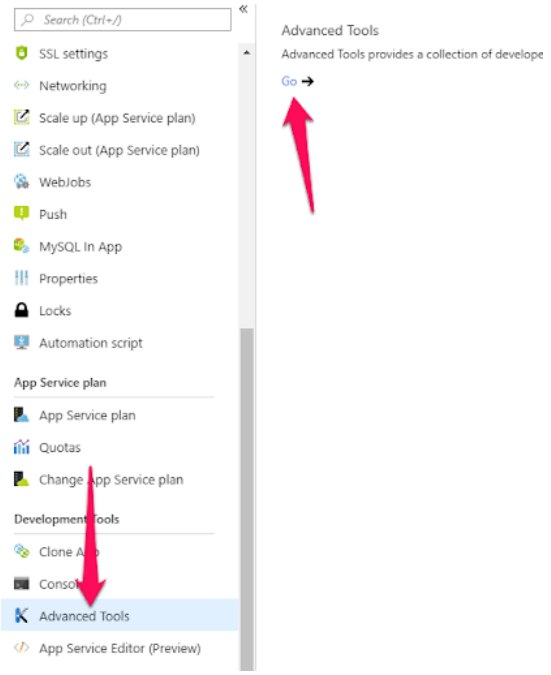
Appendix

Appendix 1: Kudu

Every Azure Web App has a Kudu application that runs alongside the App. The Kudu UI provides access to settings, deployment information, files, processes, runtime versions etc…

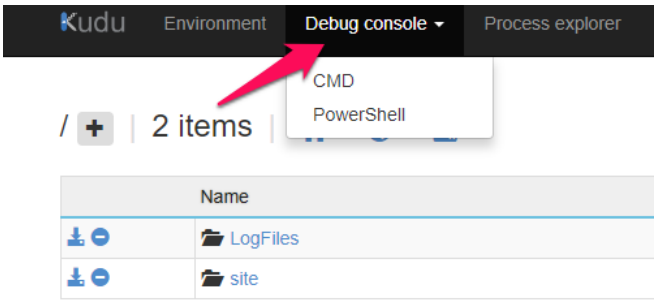
Accessing Kudu for a Web App

In the left menu goto “Advanced Tools” then click “Go” in the right panel.

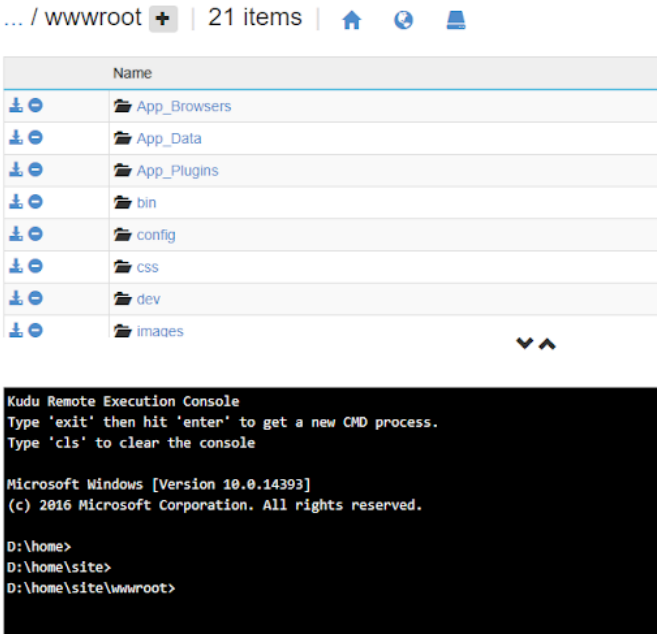


Browsing Files

Click “Debug Console” then select “CMD”.



Then browse to “site” and “wwwroot”, here you will find the Umbraco deployment files.



Appendix 2: Application Initializing

Application Initialization was released as an out-of-band module for IIS 7.5 and included with IIS 8.0 to allow web applications to perform startup processing before serving the first HTTP request.

For Azure Web Apps it can be utilised for mitigating downtime during instance migrations and to support application warmup on cold instances when scaling.

To enable Application Initializing you add a new element within the system.webServer element of web.config, for example if you didn't want your web application to server http requests until the homepage, about page and team page returned 200 you would add the below snippet.

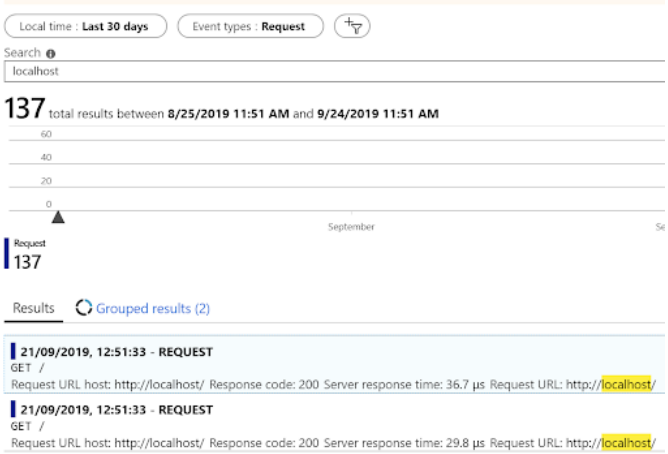
```
<applicationInitialization doAppInitAfterRestart="true">
 <add initializationPage="/"/>
 <add initializationPage="/about/"/>
 <add initializationPage="/team/"/>
</applicationInitialization>
```

> ✍ Known issue: The internal Application Initialization requests do not support https so you must ensure that your rewrite rules support this and do not use the "HTTPS Only" setting within the Azure Portal

Example of this can be implemented:

```
<rule name="No redirect on warmup request (request from localhost with warmup user agent)"
 stopProcessing="true">
 <match url="(.*)"/>
 <conditions>
   <add input="{HTTP_HOST}" pattern="localhost"/>
   <add input="{HTTP_USER_AGENT}" pattern="Initialization"/>
 </conditions>
 <action type="Rewrite" url="{URL}" redirectType="Permanent"/>
</rule>
<rule name="HTTP/S to HTTPS Redirect" stopProcessing="true">
 <match url="(.*)"/>
 <conditions logicalGrouping="MatchAny">
   <add input="{HTTPS}" pattern="off" ignoreCase="true"/>
 </conditions>
 <action type="Redirect" url="https://{HTTP_HOST}/{R:1}" redirectType="Permanent"/>
</rule>
```

To verify that Application Initialization is functioning correctly you can search the request logs in Application Insights for "localhost"



Detailed information on IIS 8.0 Application Initialization https://docs.microsoft.com/en-us/iis/get-started/whats-new-in-iis-8/iis-80-application-initialization