Cloud design patterns

Static Content Hosting Pattern

- Web applications typically include some elements of static content. This static content may include HTML pages and other resources such as images and documents that are available to the client, either as part of an HTML page (such as inline images, style sheets, and client-side JavaScript files) or as separate downloads (such as PDF documents).
- Although web servers are well tuned to optimize requests through efficient dynamic page code execution and output caching, they must still handle requests to download static content. This absorbs processing cycles that could often be put to better use.

Static Content Hosting Pattern

Fully managed file shares in the cloud, accessible via standard Server Message Block (SMB) protocol. Enables sharing files between applications using Windows APIs or REST API.

- Amazon EBS: $.045 per GB for HDD, SDD $.10 per GB.
- Azure virtual disk: $.049 per GB for HDD. SSD are $19.71 for 128 GB per month.

Scalable object storage for documents, videos, pictures, and unstructured text or binary data. Choose from Hot, Cool, or Archive tiers.

Amazon S3: storage costs $0.023 per GB Per month through first 50 TB, then the price goes down. Prices are plus network usage

Free tier: gets 5 GB for free. Archive storage on Glacier is $0.004 per GB per month.

Azure Blob Storage: Azure blob storage starts at $0.0184 per GB for hot storage but goes down to $0.01 per GB per month for cool storage, and $0.002 for archive.

Static Content Hosting Pattern

WordPress can store the PHP program on

Amazon ECS, Amazon EBS, or an Azure Disk.

You can use the common configuration of a local database or use any of the MYSQL compatible managed database services from AWS or Azure.

WordPress can be set up to store all of its **images on Amazon S3**.

Combining managed DB with offsite image storage greatly speeds up access to the WordPress site and can offer high availability through setting Amazon S3 to make the images available in all regions, bringing them closer to the client (geo-replication).

This requires **less compute power, so you can use a cheap** Amazon EC2 or Azure VM to run the WordPress application.

Static Content Hosting Pattern

AWS vs. Azure vs. Google

Provider	Storage	Pricing
Amazon S3	S3 Standard Storage	First 50 TB / Month $0.023 per GB Next 450 TB / Month $0.022 per GB Over 500 TB / Month $0.021 per GB
	S3 Standard-Infrequent Access (S3 Standard-IA) Storage	All storage / Month $0.0125 per GB
	S3 One Zone-Infrequent Access (S3 One Zone-IA) Storage	All storage / Month $0.01 per GB
Amazon EBS	Amazon EBS General Purpose SSD (gp2) Volumes	$0.10 per GB-month of provisioned storage
	Amazon EBS Provisioned IOPS SSD (io1) Volumes	$0.125 per GB-month of provisioned storage $0.065 per provisioned IOPS-month
	Amazon EBS Throughput Optimized HDD (st1) Volumes	$0.045 per GB-month of provisioned storage
Amazon Glacier	S3 Glacier Storage	All storage / Month $0.004 per GB
	S3 Glacier Deep Archive Storage	All storage / Month $0.00099 per GB
Google Cloud Storage	Multi-Regional	$0.026 - $0.036 per GB/month
	Regional	$0.02 - $0.035 per GB/month
	Nearline	$0.01 - $0.02 per GB/month
	Coldline	$0.004 - $0.014 per GB/month
Microsoft Azure	Block Blobs	$0.002/GB per month
	Azure Data Lake Storage	$0.001/GB per month
	Managed Disks	$1.54 per month
	Files	$0.060/GB per month

Static Content Hosting Pattern

- In most cloud hosting environments it is possible to minimize the requirement for compute instances (for example, to use a smaller instance or fewer instances), by **locating some of an application's resources and static pages in a storage service**. The cost for cloud-hosted storage is typically much less than for compute instances.
- When hosting some parts of an application in a storage service, the main considerations are related to deployment of the application and to securing resources that are not intended to be available to anonymous users.

Static Content Hosting Pattern

- **Issues and Considerations**
- Consider the following points when deciding how to implement this pattern:
- The hosted storage service must expose an HTTP endpoint that users can access to download the static resources. Some storage services also support HTTPS, which means that it is possible to host resources in storage service that require the use of SSL.
- For maximum performance and availability, consider using a content delivery network (where available) to cache the contents of the storage container in multiple datacenters around the world. However, this will incur additional cost for the use of the content delivery network.
- Storage accounts are often geo-replicated by default to provide resiliency against events that might impact a datacenter. This means that the **IP address** may change, but the URL will remain the same.

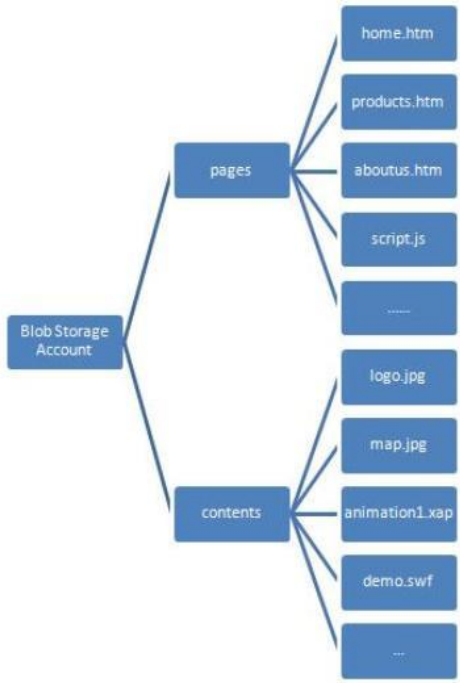
Static Content Hosting Pattern

- **When to Use this Pattern**
- This pattern is ideally suited for:
- Minimizing the hosting cost for websites and applications that contain some static resources.
- Minimizing the hosting cost for websites that consist of only static content and resources. Depending on the capabilities of the hosting provider's storage system, it might be possible to host a fully static website in its entirety within a storage account.
- Exposing static resources and content for applications running in other hosting environments or on-premises servers.
- Locating content in more than one geographical area by using a content delivery network that caches the contents of the storage account in multiple datacenters around the world.
- Monitoring costs and bandwidth usage. Using a separate storage account for some or all of the static content allows the costs to be more easily distinguished from hosting and runtime costs.

Static Content Hosting Pattern

- This pattern might **not** be **suitable** in the following situations:
- The application needs to perform some processing on the static content before delivering it to the client. For example, it may be necessary to add a timestamp to a document.
- The volume of static **content** is **very small**. The overhead of retrieving this content from separate storage may outweigh the cost benefit of separating it out from the compute resources.

Static Content Hosting



Static Content Hosting Pattern

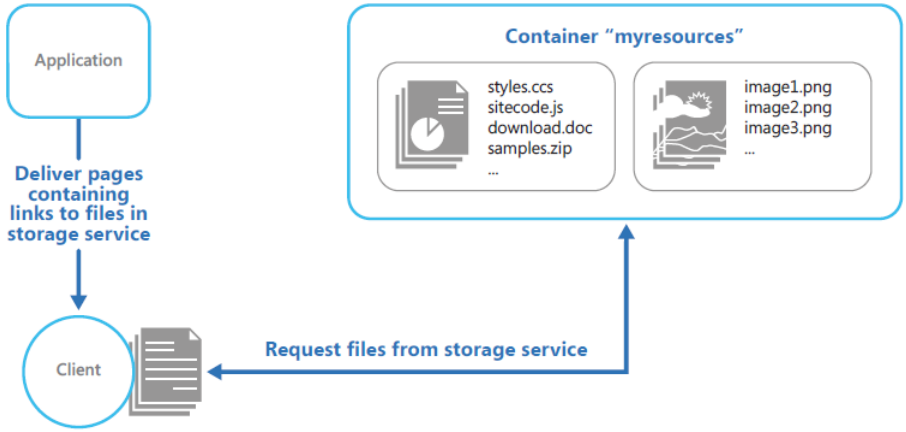


FIGURE 1
Delivering static parts of an application directly from a storage service

Static Content Hosting Pattern

- The links in the pages delivered to the client must specify the full URL of the blob container and resource. For example, a page that contains a link to an image in a public container might contain the following.
- **HTML**

Static Content Hosting Pattern

- The **StaticContentHosting.Cloud** project contains configuration files that specify the storage account and container that holds the static content.
- **XML**
- <Setting name="StaticContent.StorageConnectionString"

value="UseDevelopmentStorage=true" />

- <Setting name="StaticContent.Container" value="static-content" />

Static Content Hosting Pattern

- Settings.cs of the StaticContentHosting.Web project
- methods to extract these values and build a string value containing the cloud storage account container URL.

```
public class Settings {
public static string StaticContentStorageConnectionString {
get { return RoleEnvironment.GetConfigurationSettingValue(
"StaticContent.StorageConnectionString");  } }
public static string StaticContentContainer {
get { return RoleEnvironment.GetConfigurationSettingValue("StaticContent.Container"); } }
public static string StaticContentBaseUrl {
get { var account = CloudStorageAccount.Parse(StaticContentStorageConnectionString);
return string.Format("{0}/{1}", account.BlobEndpoint.ToString().TrimEnd('/'),
StaticContentContainer.TrimStart('/'));
} }}
```

Static Content Hosting Pattern

- The **StaticContentUrlHtmlHelper** class in the file StaticContentUrlHtmlHelper.cs exposes a method named **StaticContentUrl** that generates a URL containing the path to the cloud storage account if the URL passed to it starts with the ASP.NET root path character (**~**).

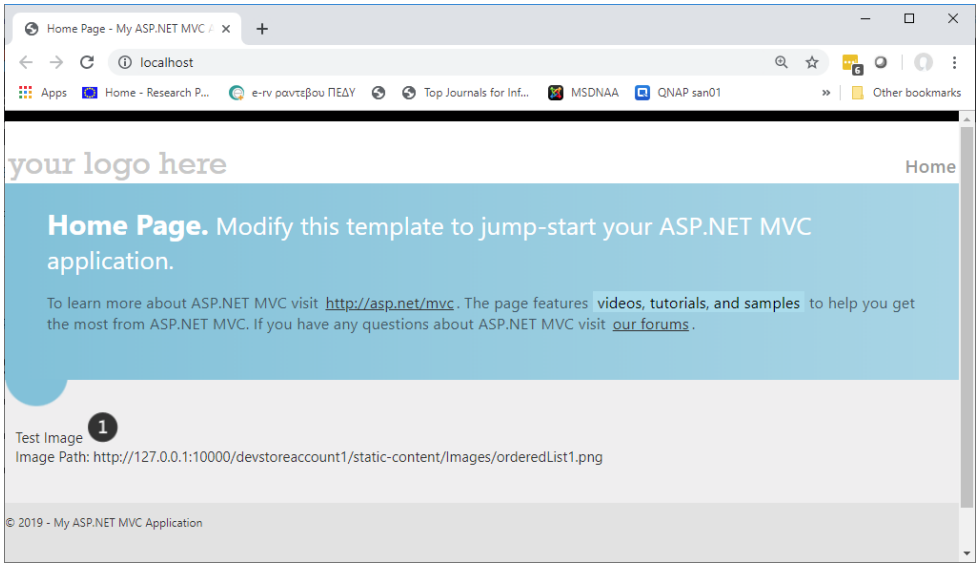
```
public static class StaticContentUrlHtmlHelper{
public static string StaticContentUrl(this HtmlHelper helper, string contentPath) {
if (contentPath.StartsWith("~")) {
contentPath = contentPath.Substring(1); }
contentPath = string.Format("{0}/{1}", Settings.StaticContentBaseUrl.TrimEnd('/'),
contentPath.TrimStart('/'));
var url = new UrlHelper(helper.ViewContext.RequestContext);
return url.Content(contentPath); }}
```

Static Content Hosting Pattern

- The file Index.cshtml in the Views\Home folder contains an image element that uses the **StaticContentUrl** method to create the URL for its **src** attribute.
- **HTML**

```
<img src="@Html.StaticContentUrl("~/Images/orderedList1.png")" alt="Test
Image" />
```

Static Content Hosting Pattern



Static Content Hosting Pattern

- **Related Patterns and Guidance**
- The following pattern may also be relevant when implementing this pattern:
- **Valet Key Pattern**. If the target resources are not supposed to be available to anonymous users it is necessary to implement security over the store that holds the static content. The Valet Key pattern describes how to use a token or key that provides clients with restricted direct access to a specific resource or service such as a cloud-hosted storage service.

Health Endpoint Monitoring.

- **Health Endpoint Monitoring**. This example shows how you can set up a web endpoint that checks the health of dependent services by returning appropriate status codes.
- The endpoints are designed to be consumed by a watchdog monitoring service such as Windows Azure Endpoint Monitoring, but *you can open and invoke the endpoint operations from a browser to see the results*. You can also deploy and configure your own endpoint monitoring tool of choice to send requests to the service operations and analyze the responses received.

Health Endpoint Monitoring.

- **Problem**
- How to detect that a running service instance is unable to handle requests?
- **Forces**
- An alert should be generated when a service instance fails
- Requests should be routed to working service instances

Health Endpoint Monitoring.

- **Solution**
- A service has an health check API endpoint (e.g. HTTP /health) that returns the health of the service. The API endpoint handler performs various checks, such as
- the status of the connections to the infrastructure services used by the service instance
- the status of the host, e.g. disk space
- application specific logic
- A health check client - a monitoring service or load balancer - periodically invokes the endpoint to check the health of the service instance.

Health Endpoint Monitoring.

- **Resulting Context**
- This pattern has the following benefits:
- The health check endpoint enables the health of a service instance to be periodically tested
- This pattern has the following drawbacks:
- The health check might not sufficiently comprehensive or the service instance might fail between health checks and so requests might still be routed to a failed service instance
- **Related patterns**
- Service registry - the service registry invokes the health check endpoint

Health Endpoint Monitoring.

- Start the example running and navigate to one of the monitoring endpoints. Use the F12 Developer Tools in your browser to watch the network traffic response pattern.
- The response will be 200 (OK), or a 500 error code that indicates a problem with a service.

Health Endpoint Monitoring.

- The sample contains four endpoints that you can test:
  - **/HealthCheck/CoreServices** This is a simple check on dependent services to determine if they are responding.
  - **/HealthCheck/ObscurePath/{***path***}** This demonstrates a check operation that uses a configurable obscure path defined in the service configuration file. Replace *path* with the value of the setting named Test.ObscurePathfrom in the file ServiceConfiguration.*.csfg. Note that this technique is not to be used as an alternative to properly securing an application and implementing authentication.

Health Endpoint Monitoring.

- **/HealthCheck/CheckUnstableServiceHealth** This will randomly return a 500 exception for approximately 20% of the requests.
- **/HealthCheck/TestResponseFromConfig** This returns a response code set in configuration (.cscfg) for testing. Deploy the solution with endpoint monitoring configured to use this operation, and then explicitly set the "Test.ReturnStatusCode" setting to return the required status code to test and demonstrate the effects of these codes on the monitoring service.

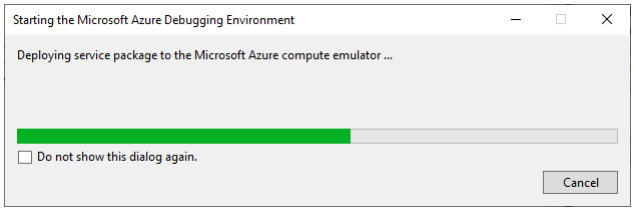
Health Endpoint in VS

- Presentation of VS structure

Health Endpoint in VS

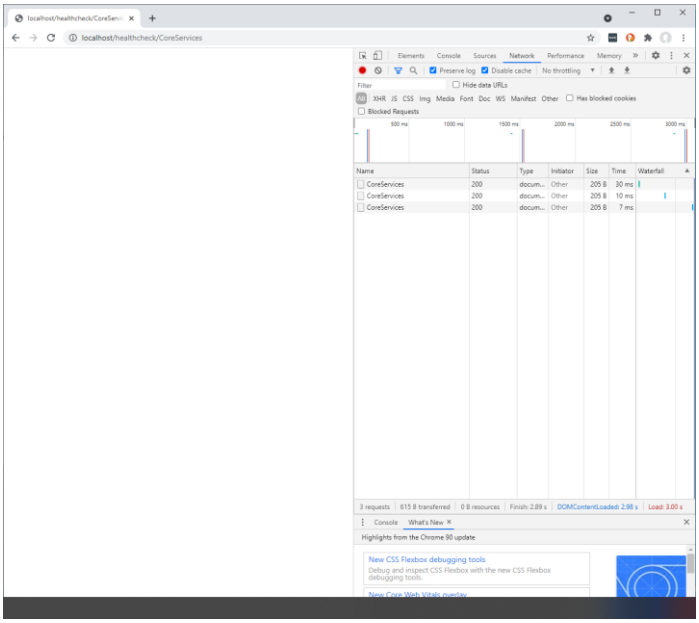
Start VS with Administrator Rights

When starting the sample Azure Debugging Environment starts locally



Health Endpoint CoreServices

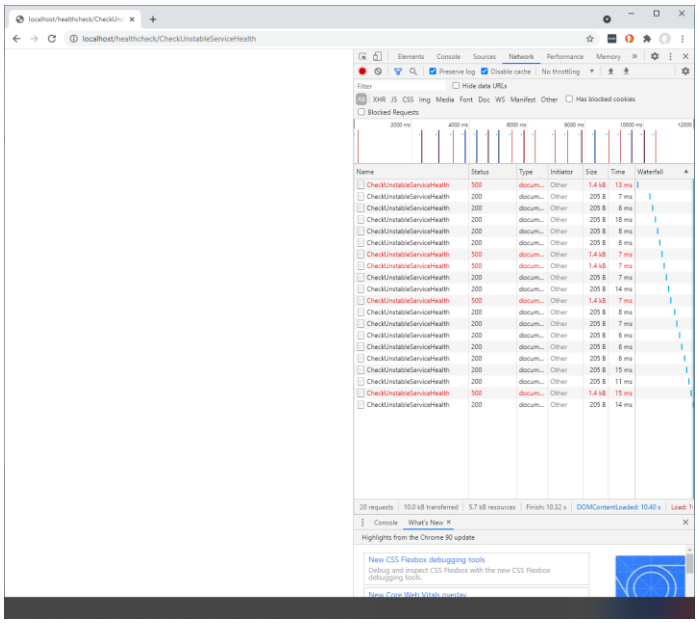
Check network tab in dev tools
Status 200 is returned after successful checks



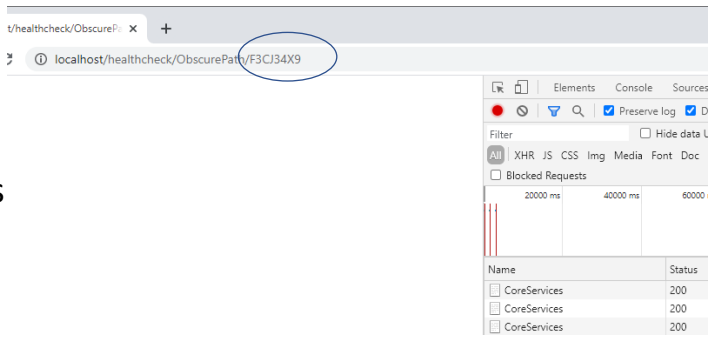
Health Endpoint CheckUnstableServiceHealth

Check network tab in dev tools

Status is returned after checks



Health Endpoint ObscurePath



Check network tab in dev tools

Status is returned after checks

Secret path is set in service configuration to avoid having a easily guessable healthcheck path

(and keep changing it easily in the future)

Endpoint Monitoring with Azure Application Insights

- **Application Insights** - a **Microsoft Azure** service - to monitor application endpoint /healthcheck.
- *Health Checks* endpoint up and running

Endpoint Monitoring with Azure Application Insights

- Applications Insights it's a great tool for monitoring, error logging, performance monitor, dependency mapping, and other things. In other words, it's a fully APM - Application Performance Management.
- Application Insights, a feature of Azure Monitor, is an extensible Application Performance Management (APM) service for developers and DevOps professionals.

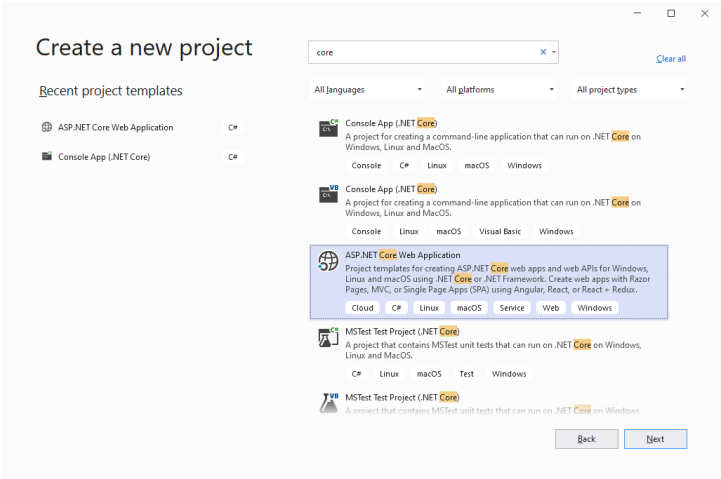
https://docs.microsoft.com/pt-br/azure/azure-monitor/app/app-insights-overview

Health Check into a .NET core App

- Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.
- This can help to verify that applications and services are performing correctly.
- Simple example in .Net Core application

Adding basic health check into a .NET core API

- .NET core API



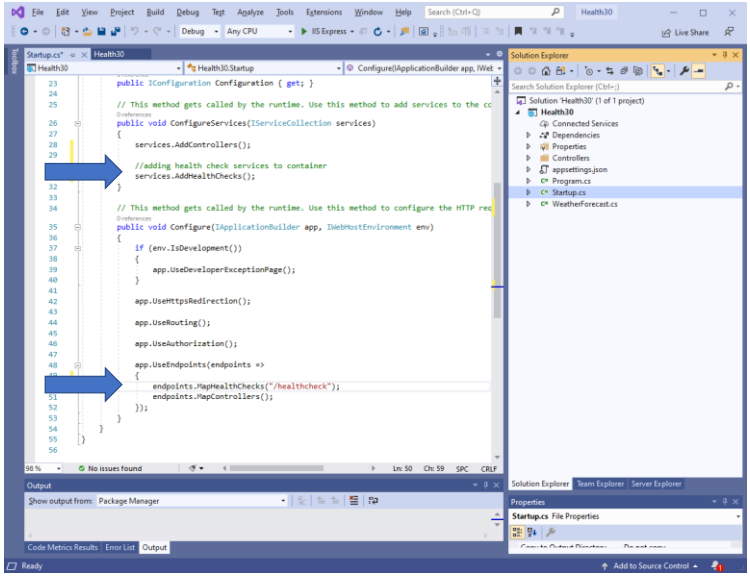
Adding basic health check into a .NET core API

- ConfigureServices add HealthCheck

app.UseHealthChecks("/healthcheck");

- Configure the endpoing (add pipeline)

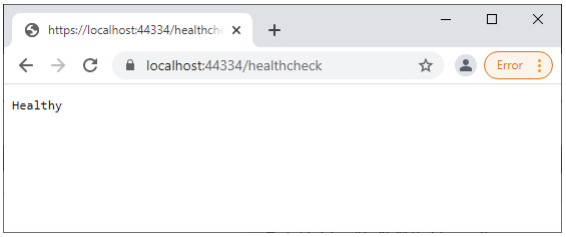
endpoints.MapHealthChecks("/healthcheck");



Adding basic health check into a .NET core API

- Build and run

/healthcheck



Publish to Azure (Deployment) with VS

- Before we proceed intro to Cloud choices

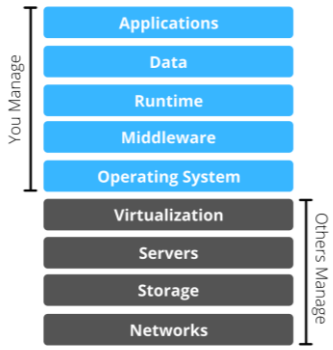
Figure: https://miro.medium.com/max/3840/1*pmJDtQhwbjviahlQnbf6Pg.jpeg

Cloud choices

- **Azure Virtual Machine**
- Azure Virtual Machine is a IaaS

(Infrastructure-as-a-Service) cloud service

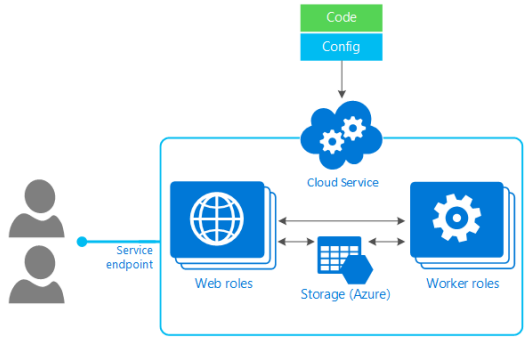
https://docs.microsoft.com/en-us/azure/virtual-machines/windows/quick-create-portal



Cloud choices

- **Azure Cloud Services**
- Azure Cloud Services is a PaaS

(Platform-as-a-Service) cloud service



Like Azure App Service, this technology is designed to support applications that are scalable, reliable, and inexpensive to operate.

In the same way that App Service is hosted on virtual machines (VMs), so too is Azure Cloud Services.

However, you have more control over the VMs. You can install your own software on VMs that use Azure Cloud Services, and you can access them remotely.

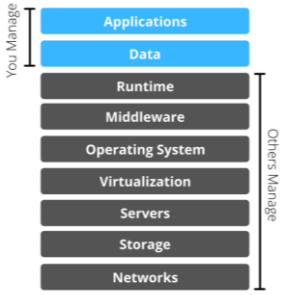
More control also means less ease of use.

Web role: Automatically deploys and hosts your app through IIS

Worker role: Does not use IIS, and runs your app standalone.

https://docs.microsoft.com/en-us/azure/cloud-services/cloud-services-choose-me

Cloud choices



- **Azure App Service**
- Azure App Service is PaaS
  (Platform-as-a-Service) cloud service by Microsoft.

https://docs.microsoft.com/en-us/azure/app-service/overview

Cloud choices

- Azure VMs are more expensive to run in comparison to Azure App Service.
- Azure App Service have constraints in comparison to Azure VMs in terms of scalability. Hence, Azure VMs are preferred for apps, which have scope to expand for future.
- Azure App Service requires much less managerial efforts in comparison to Azure Virtual Machines.
- The development of app is much simpler and faster in Azure App Service.

Cloud choices

- Azure VMs offer developer more control over the environment. Like, one can't choose underlying OS of VM in an Azure App Service.
- Azure App Services do not offer Pay-as-you-Go. Hence, you're paying for the service plan, even if you're not using it.
- There may be constraints for the support of certain programming languages on Azure App Service. In that case, one has to use Azure VM to create environment for the programming language.

Cloud choices

With **Azure Cloud Services**

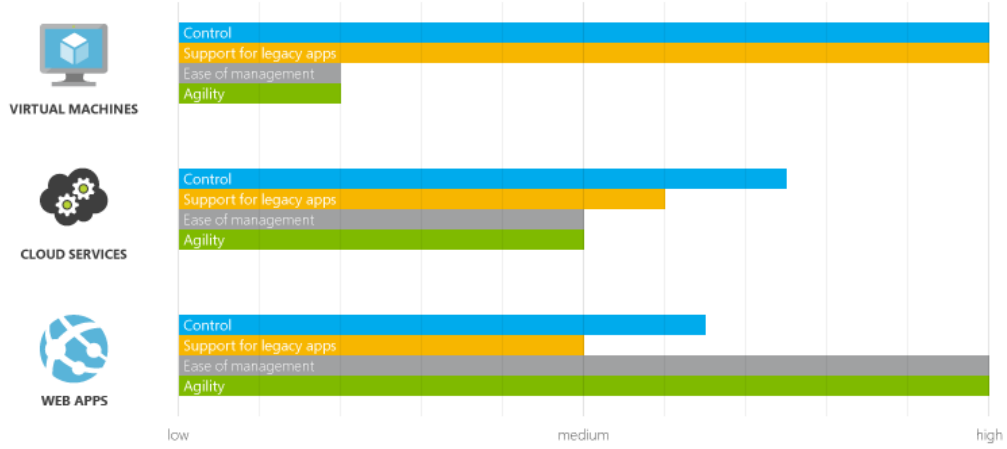
- You don't create virtual machines. Instead, you provide a configuration file that tells Azure how many of each you'd like, such as "three web role instances" and "two worker role instances." The platform then creates them for you.
- You still choose what size those backing VMs should be, but you don't explicitly create them yourself.
- Unlike VMs created with Virtual Machines, writes made to Azure Cloud Services VMs aren't persistent.
- There's nothing like a Virtual Machines data disk. Instead, an Azure Cloud Services application should explicitly write all state to Azure SQL Database, blobs, tables, or some other external storage.
- Like in App Service, Azure Cloud gives support for multiple deployment, automatic OS upgrades, and seamless platform switching.
- In addition, you'll also get Remote Desktop (RDP) access to servers, custom MSI installations, the ability to define and execute start-up tasks, and to listen to Event Tracing for Windows (ETW) events

Cloud choices

- Azure App Service
- run some background jobs along with your App Services deployment, Azure offers an integrated service called WebJobs

- users trigger custom programs or scripts on demand, continuously, or according to a set schedule. You can upload and run executable files built as cmd, bat, exe (.NET), ps1, sh, php, py, js and jar

Cloud choices

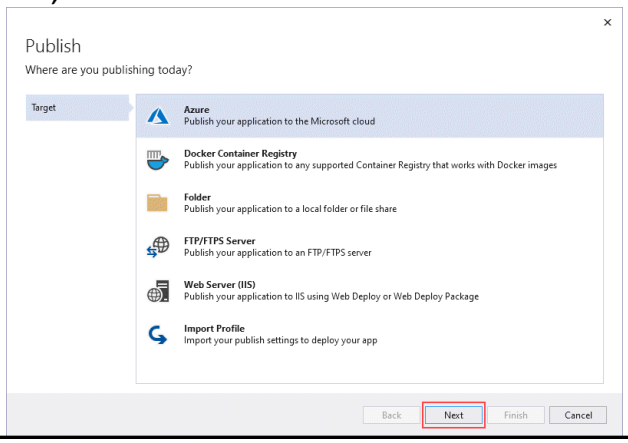


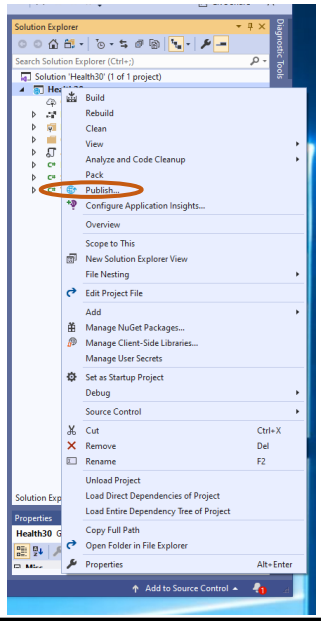
Deploy APS.NET Web App (or API) with VS

- An Azure account with an active subscription. Create an account
- Visual Studio with the **ASP.NET and web development** workload
- Given a Solution and a Project in VS next
- Create and configure a new App Service that you can publish your app to.

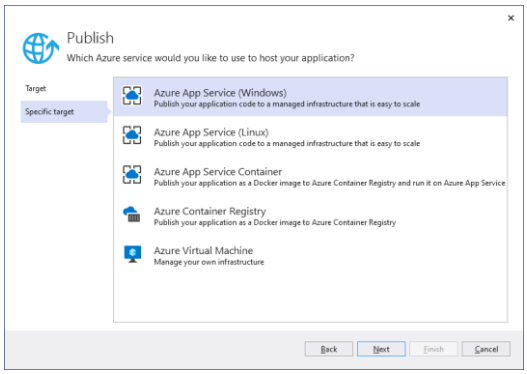
Create and configure a new App Service that you can publish your app to.

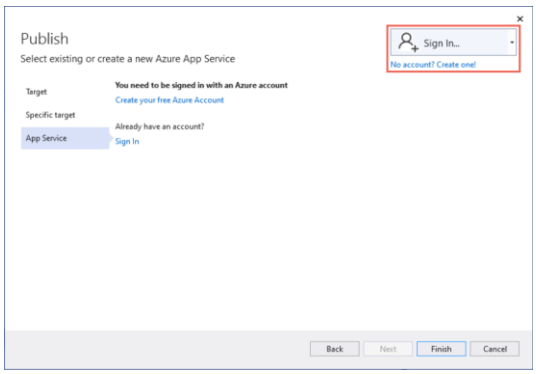
- Publish the project (right click)
- In **Publish**, select **Azure** and



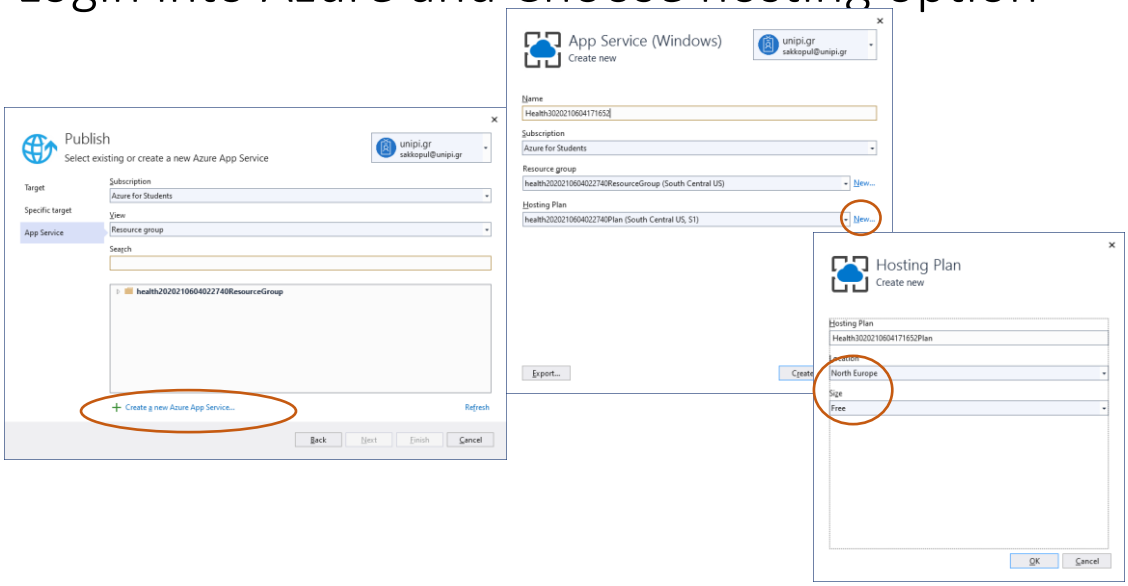


Login into Azure and Choose hosting option





Login into Azure and Choose hosting option



Login into Azure and Choose hosting option

Each App Service plan defines:

- Region (West US, East US, etc.)
- Number of VM instances
- Size of VM instances (Small, Medium, Large)
- Pricing tier (Free, Shared, Basic, Standard, Premium, PremiumV2, PremiumV3, Isolated)

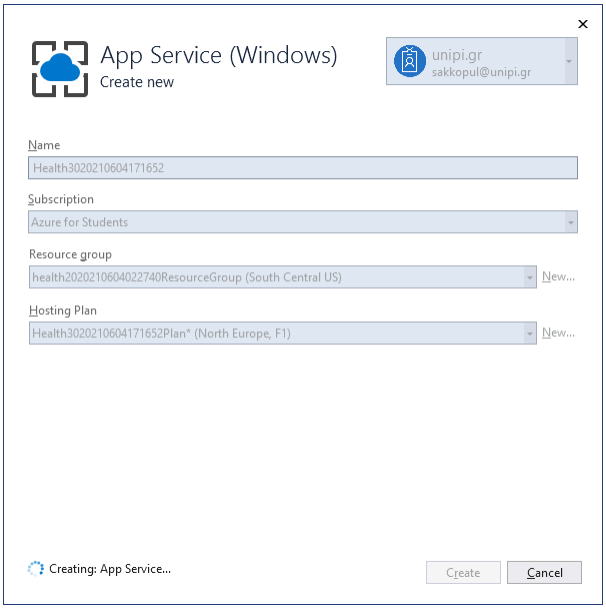
The *pricing tier* of an App Service plan determines what App Service features you get and how much you pay for the plan. There are a few categories of pricing tiers:

- **Shared compute**: **Free** and **Shared**, the two base tiers, runs an app on the same Azure VM as other App Service apps, including apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources cannot scale out.
- **Dedicated compute**: The **Basic**, **Standard**, **Premium**, **PremiumV2**, and **PremiumV3** tiers run apps on dedicated Azure VMs. Only apps in the same App Service plan share the same compute resources. The higher the tier, the more VM instances are available to you for scale-out.
- **Isolated**: This tier runs **dedicated Azure VMs on dedicated Azure Virtual Networks**. It provides network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities.

* App Service Free and Shared (preview) hosting plans are base tiers that run on the same Azure virtual machines as other App Service apps. Some apps might belong to other customers. These tiers are intended to be used only for development and testing purposes.

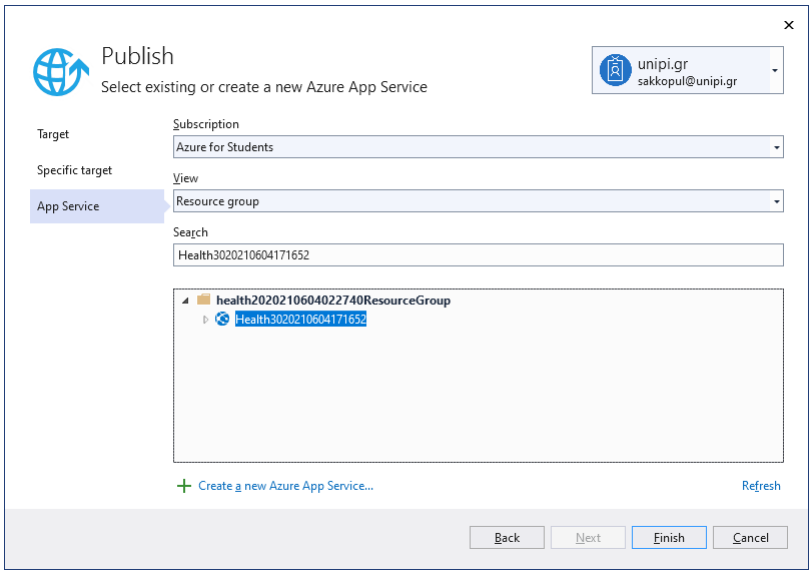
Creating App Service

- Takes 2-3 minutes max



Creating App Service

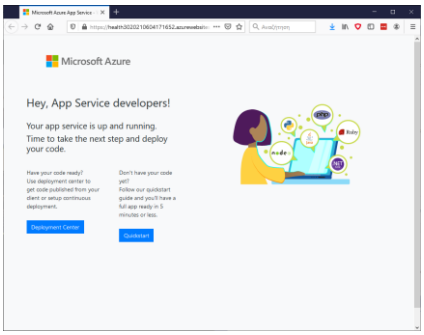
- Finish

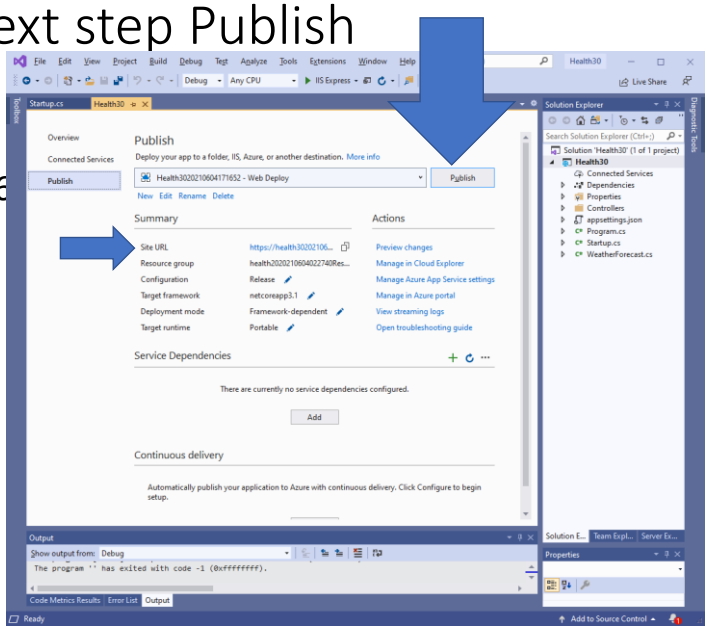


App Service done next step Publish

- Site Url:

https://health30202106041716

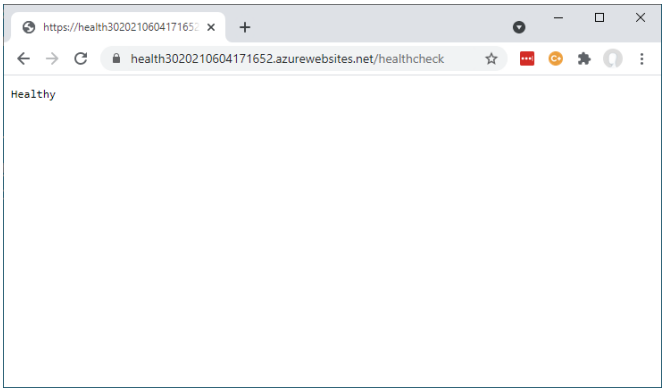




App Service done next step Publish

- Site Url:

https://health30202106041716 52.azurewebsites.net/healthcheck



Update the App and redeploy

- Click Publish

Health checks in ASP.NET Core

- ASP.NET Core offers Health Checks Middleware and libraries for reporting the health of app infrastructure components.
- Health checks are exposed by an app as HTTP endpoints
- For many apps, a basic health probe configuration that reports the app's availability to process requests (*liveness*) is sufficient to discover the status of the app.

Health checks in ASP.NET Core

- Register health check services with AddHealthChecks in Startup.ConfigureServices.
- Create a health check endpoint by calling MapHealthChecks in Startup.Configure.
- In the sample app, the health check endpoint is created at /health (BasicStartup.cs):

Health checks in ASP.NET Core

- Health checks are created by implementing the IHealthCheck interface.
- The Microsoft.AspNetCore.Diagnostics.HealthChecks package is referenced implicitly for ASP.NET Core apps.
- To perform health checks using Entity Framework Core, add a reference to the Microsoft.Extensions.Diagnostics.HealthChecks.EntityFrameworkCore package.

Create health check randomly returned

```
public class ExampleHealthCheck : IHealthCheck
  { public Task<HealthCheckResult> CheckHealthAsync(
        HealthCheckContext context,
        CancellationToken cancellationToken = default(CancellationToken))
    {  // Get a random number
       var rnd = new Random().Next(10);
       // If the random number is less than 8 return a 200 else return 500
       if (rnd < 8)
       {      return Task.FromResult(
            HealthCheckResult.Healthy("A healthy result."));           }
       else {
       return Task.FromResult(
          new HealthCheckResult(context.Registration.FailureStatus,
          "An unhealthy result."));           }
    }
  }
```

Register health check services

Startup.ConfigureServices:

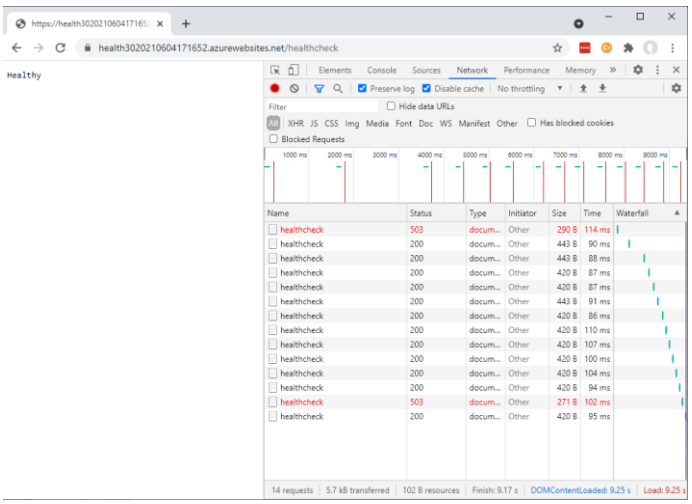
```
services.AddHealthChecks()
   .AddCheck<ExampleHealthCheck>("example_health_check");
```

Use Health Checks Routing

```
app.UseEndpoints(endpoints =>
{
   endpoints.MapHealthChecks("/health");
});
```

Deploy To Azure

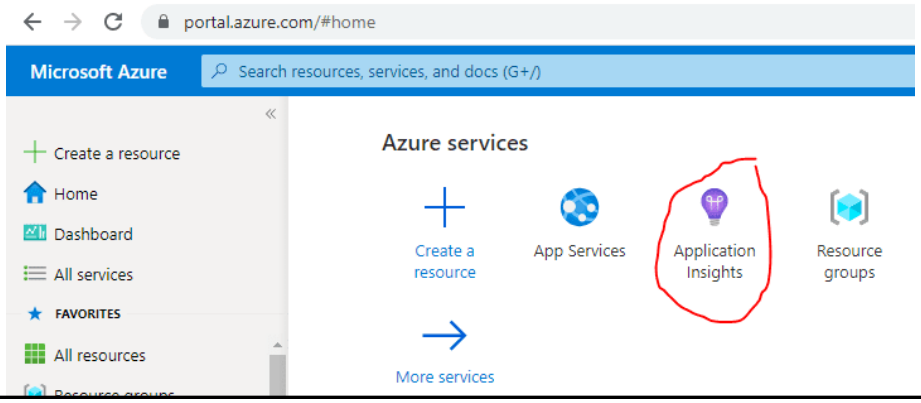
Publish



Endpoint Monitoring with Azure Application Insights

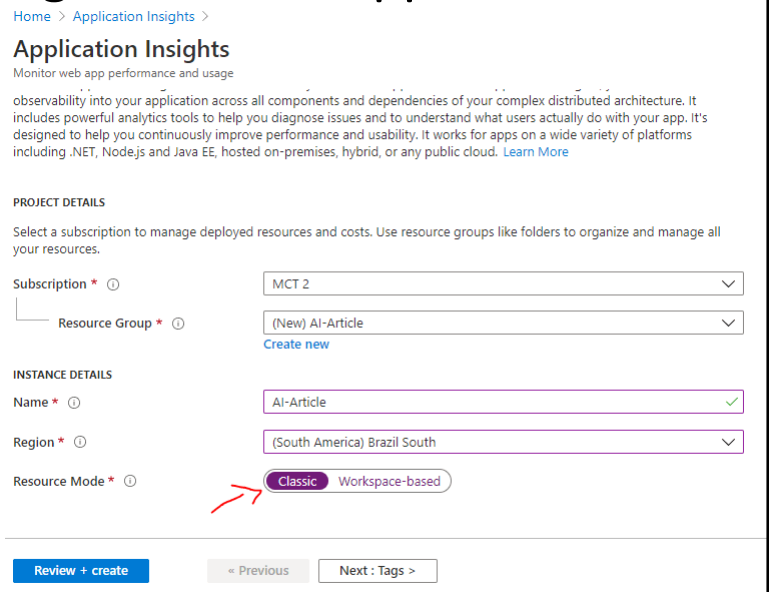
- **Creating Application Insights Resource**

Access the Azure Portal and access the Application Insights blade and then click Add to create a new resource.



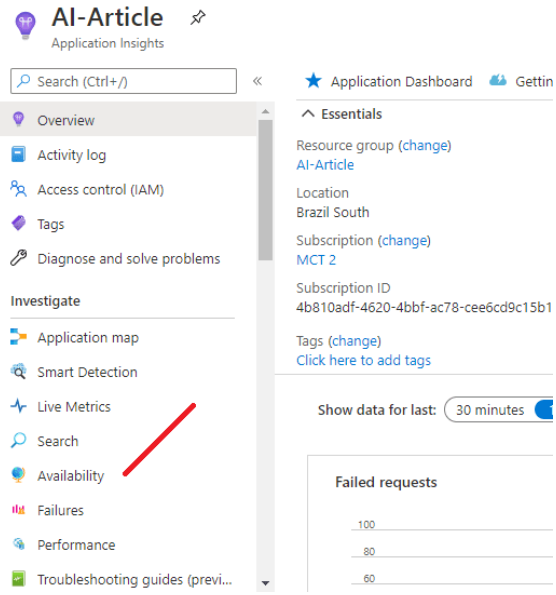
Endpoint Monitoring with Azure Application Insights

- Enter all the required information to create the resource, such as subscription, resource group, etc.
- When asked for the Resource Mode choose the Classic.



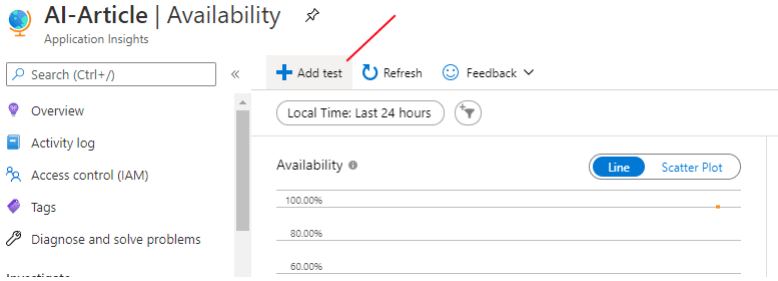
Endpoint Monitoring with Azure Application Insights

- Once the Application Insights it's created, go to the Availability section. This is where the monitoring tool is configured to watch our endpoint.



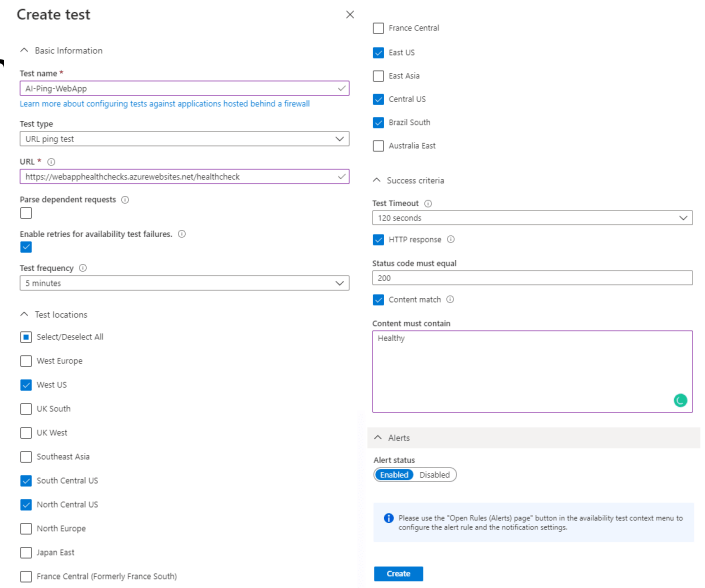
Endpoint Monitoring with Azure Application Insights

- ADD TEST



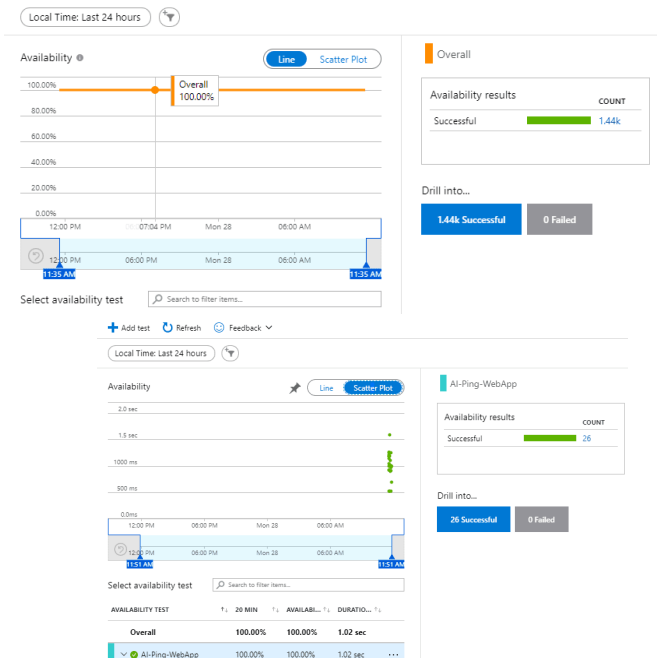
Endpoint Monitoring with Azure Application Insights

- Test name: *AI-Ping-WebApp*
- Test Type: *URL ping test*
- URL: *https://webapphealthchecks.azurewebsites.net/healthcheck*
- Parse dependent requests: *unchecked*
- Enable retries for availability test failures: *checked*
- Test frequency: *5 minutes*
- Test locations: *default values*
- Test Timeout: *120 seconds*
- HTTP Response: *checked*
- Status code must equal: *200*
- Content match: *checked*
- Content must contain: *Healthy*
- Alerts: *leave default value*



Endpoint Monitoring with Azure Application Insights

- After a few minutes, you should get something like this.

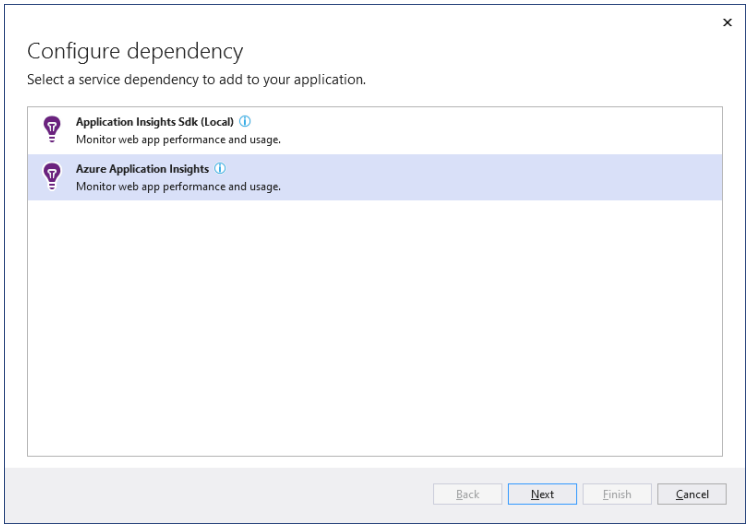


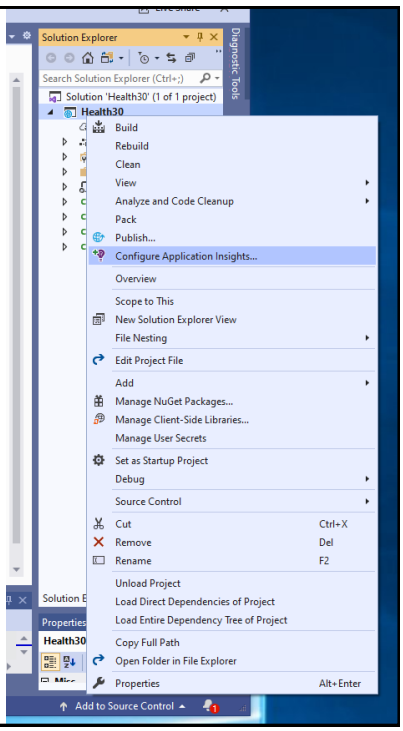
Endpoint Monitoring with Azure Application Insights

Paradigm by:

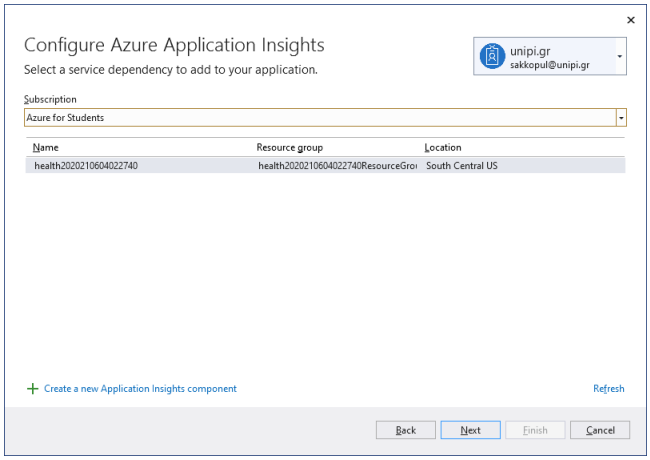
https://dev.to/rmaurodev/endpoint-monitoring-with-azure-application-insights-45kf

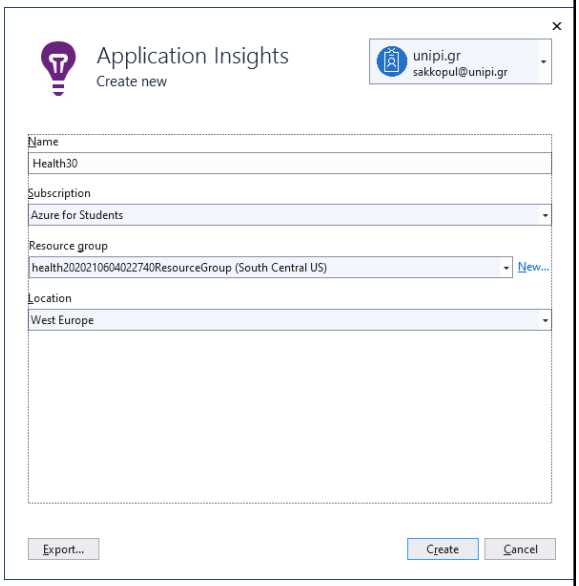
Application Insight in VS



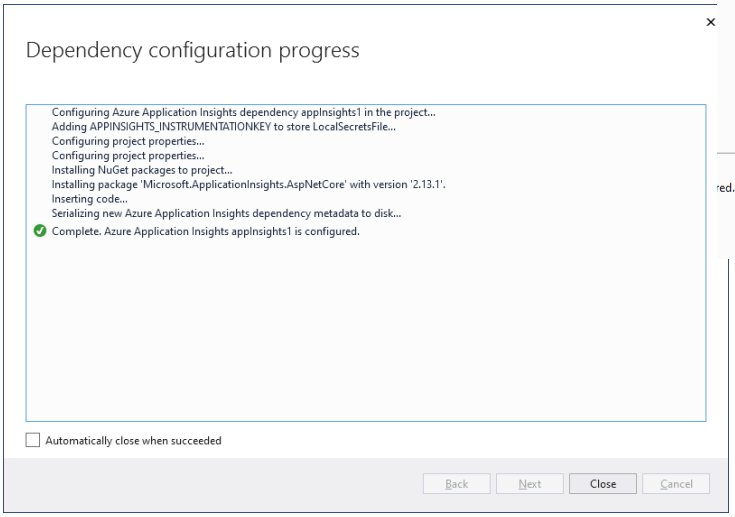


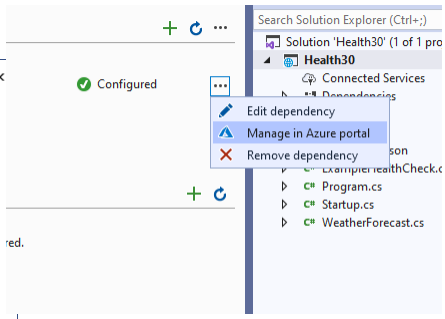
Application Insight in VS



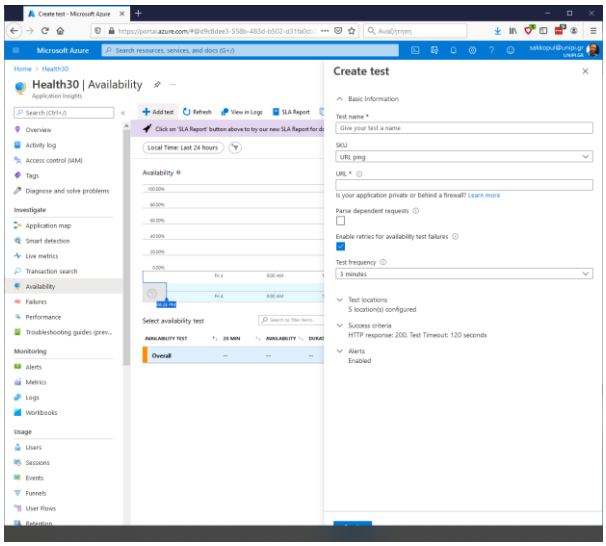


Application Insight in VS

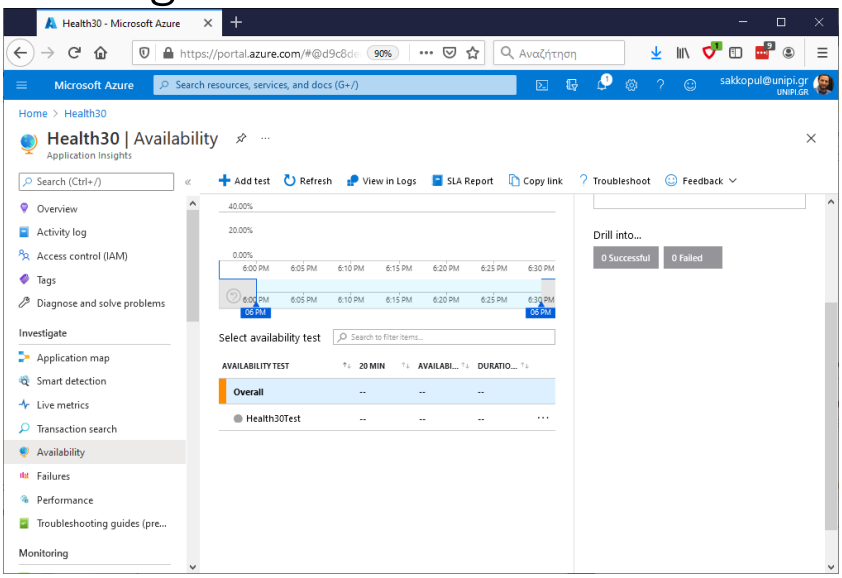




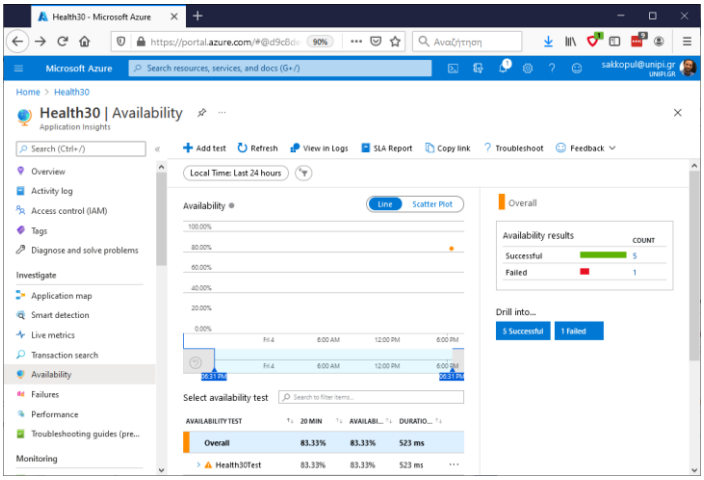
Application Insight in VS



Application Insight in VS



Application Insight in VS



Pipes and Filters Pattern

- **Pipes and Filters**. This example contains two filters that could perform some part of the overall processing for a task. The two filters are combined into a pipeline; the output of one filter is passed as the input to the next. The filters are implemented as separate worker roles and a Windows Azure Service Bus queue provides the infrastructure that acts as the pipe.