<https://app.pluralsight.com/library/courses/asp-dot-net-core-health-checks/table-of-contents>

Para un proyecto web recién levantando…

services.AddHealthChecks();

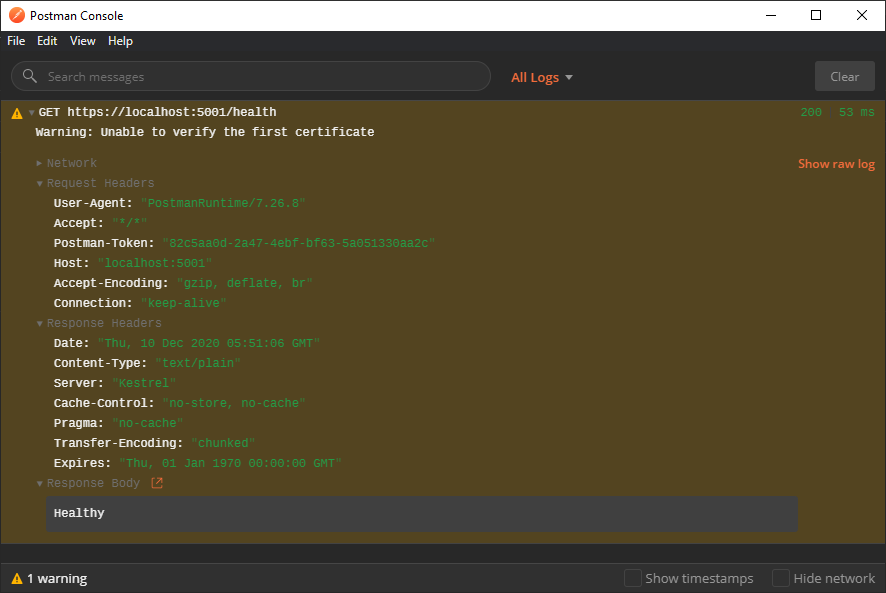
app.UseEndpoints(endpoints =>

{

endpoints.MapHealthChecks("/health");

});

Y si pedimos /health nos devolverá un 200 con el texto con el texto Healthy… ya hemos agregado un health check a nuestra aplicación, un liveness, la aplicación está levantada… podríamos hablar de *“un ping a nuestra aplicación”*.



Y con esto otro tenemos un readiness (chequeando una dependencia)

services.AddHealthChecks().AddCheck("db", () =>

{

try

{

using var cnn = new SqlConnection("Server=.;Database=master;Trusted\_Connection=True;");

cnn.Open();

return HealthCheckResult.Healthy();

}

catch (SqlException)

{

return HealthCheckResult.Unhealthy();

}

});

Y si metemos un paquete de terceros (como AspNetCore.HealthChecks.SqlServer, <https://github.com/Xabaril/AspNetCore.Diagnostics.HealthChecks>) AspNetCore.HealthChecks.SqlServer

En vez de tener que picarnos el AddCheck anterior, ahora pasamos a esto:

services.AddHealthChecks().AddSqlServer("Server=.;Database=master;Trusted\_Connection=True;");

Con este otro paquete (también de Xabaril), vamos a chequear Uris,  AspNetCore.HealthChecks.Uris

Este enlace devuelve [http://www.mocky.io/v2/5e2601892f00007c00ce2de6](http://www.mocky.io/v2/5e2601892f00007c00ce2de6" \t "blank)   
{ "name": "Sergio", "age": 44 } Con ?mocky-delay=100ms podemos decirle que vaya más lento

services.AddHealthChecks()

.AddUrlGroup(

new Uri("http://www.mocky.io/v2/5e2601892f00007c00ce2de6?mocky-delay=3000ms"),

"mocky",

failureStatus: HealthStatus.Degraded, // si falla, mi aplicación se degrada pero puedo seguir

timeout: new TimeSpan(0, 0, 2)); // falla si no responde en...

Ahora vamos a personalizar un poco la respuesta y a filtrar health checks…

private Task CustomResponse(HttpContext httpContext, HealthReport result)

{

httpContext.Response.ContentType = "application/json";

return httpContext.Response.WriteAsync(JsonConvert.SerializeObject(result, new Newtonsoft.Json.Converters.StringEnumConverter()));

}

var healthCheckOptions = new HealthCheckOptions()

{

ResultStatusCodes = // by default

{

[HealthStatus.Healthy] = StatusCodes.Status200OK,

[HealthStatus.Degraded] = StatusCodes.Status200OK,

[HealthStatus.Unhealthy] = StatusCodes.Status503ServiceUnavailable

},

ResponseWriter = CustomResponse,

Predicate = (HealthCheckRegistration check) =>

{

// Before healthcheck execution

logger.LogInformation(check.FailureStatus.ToString());

logger.LogInformation(check.Name);

logger.LogInformation(check.Tags.Count.ToString());

logger.LogInformation(check.Timeout.ToString());

if (check.Name == "db") // filter

{

return false;

}

return true;

},

AllowCachingResponses = false //bydefault

};

endpoints.MapHealthChecks("/health", healthCheckOptions); //You can add several endpoints with differente routes and healtcheck options

En vez del texto “Healthy, Unhealthy o Degraded”, ahora tenemos una respuesta “rica”…

|  |
| --- |
| {      "Entries": {          "sqlserver": {              "Data": {},              "Description": null,              "Duration": "00:00:00.3592291",              "Exception": null,              "Status": "Healthy",              "Tags": []          },          "mocky": {              "Data": {},              "Description": null,              "Duration": "00:00:02.0446711",              "Exception": {                  "ClassName": "System.Threading.Tasks.TaskCanceledException",                  "Message": "The operation was canceled.",                  "Data": null,                  "InnerException": {                      "ClassName": "System.IO.IOException",                      "Message": "Unable to read data from the transport connection: The I/O operation has been aborted because of either a thread exit or an application request..",                      "Data": null,                      "InnerException": {                          "ClassName": "System.Net.Sockets.SocketException",                          "Message": "The I/O operation has been aborted because of either a thread exit or an application request.",  en cancellationToken)\r\n   at System.Net.Http.HttpConnectionPool.SendWithNtConnectionAuthAsync(HttpConnection connection, …",                  "RemoteStackTraceString": null,                  "RemoteStackIndex": 0,                  "ExceptionMethod": null,                  "HResult": -2146233029,                  "Source": "System.Net.Http",                  "WatsonBuckets": null              },              "Status": "Degraded",              "Tags": []          }      },      "Status": "Degraded",      "TotalDuration": "00:00:02.0944052"  } |

private Task CustomResponse(HttpContext httpContext, HealthReport result)

{

httpContext.Response.ContentType = "application/json";

var dependencies = result.Entries.ToDictionary(entry => new KeyValuePair<string, object>(

entry.Key,

new

{

entry.Value.Status,

Duration = entry.Value.Duration.TotalSeconds,

Failure = entry.Value.Exception != null && !(entry.Value.Exception is TaskCanceledException) ? entry.Value.Exception : null

}));

var value = new

{

result.Status,

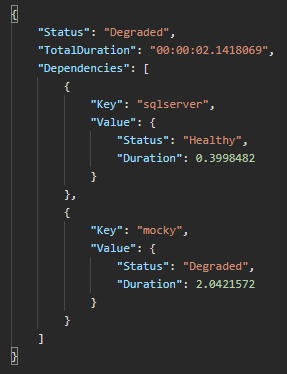
result.TotalDuration,

Dependencies = dependencies

};

return httpContext.Response.WriteAsync(JsonConvert.SerializeObject(value, new Newtonsoft.Json.Converters.StringEnumConverter()));

}



Si queremos meter tags (para filtrar)…

services.AddHealthChecks().AddCheck("my\_healthcheck", () =>

{

// ...

}, tags: new[] { "readiness" });

if (check.Tags.Contains("readiness")) // filter

Para crear nuestros propio HealthCheck (de ASP.NET, no estamos hablando de Xabaril):

public class FilePathWriterHealthCheck : IHealthCheck

{

private readonly string \_filePath;

private readonly IReadOnlyDictionary<string, object> \_healthCheckData;

public FilePathWriterHealthCheck(string filePath)

{

\_filePath = filePath;

\_healthCheckData = new Dictionary<string, object>()

{

{"FilePath", \_filePath}

};

}

public async Task<HealthCheckResult> CheckHealthAsync(HealthCheckContext context, CancellationToken cancellationToken = new CancellationToken())

{

try

{

await File.WriteAllTextAsync(\_filePath, nameof(FilePathWriterHealthCheck), cancellationToken);

File.Delete(\_filePath);

return HealthCheckResult.Healthy();

}

catch (Exception e)

{

return context.Registration.FailureStatus switch

{

HealthStatus.Degraded => HealthCheckResult.Degraded("Issues writing to file path", e, \_healthCheckData),

\_ => HealthCheckResult.Unhealthy("Issues writing to file path", e, \_healthCheckData),

};

}

}

}

.AddCheck(

name: "File Path Health Check", // "File Path Health Check" aparecerá como Health Check en la respuesta

instance: new FilePathWriterHealthCheck(Path.Join(Directory.GetCurrentDirectory(), "test")), // With Directory.GetCurrentDirectory() will be Unhealthy

failureStatus: HealthStatus.Unhealthy,

tags: new[] { "ready" });

Y para ponerlo más bonito (y prepararlo para un paquete Nuget):

.AddFilePathWriter("C:\\Temp", HealthStatus.Unhealthy, new[] { "ready" });

namespace Microsoft.Extensions.DependencyInjection

{

public static class FilePathWriterHealthChecksBuilderExtensions

{

public static IHealthChecksBuilder AddFilePathWriter(this IHealthChecksBuilder builder, string filePath,

HealthStatus failureStatus, IEnumerable<string> tags = default)

{

if (filePath == null) throw new ArgumentNullException(nameof(filePath));

return builder.Add(

new HealthCheckRegistration(

"File Path Health Check",

new FilePathWriterHealthCheck(filePath),

failureStatus,

tags));

}

}

}

.AddFilePathWriter(Path.Join(Directory.GetCurrentDirectory(), "test"), HealthStatus.Unhealthy, tags: new[] { "readiness" });

Por ahora, las aplicaciones clientes que están consumiendo nuestros healthchecks somos nosotros mismos con Postman o pidiéndolo con Chrome… pero también hay un paquete que nos da una UI para chequear automáticamente los health checks, de Xabaril, AspNetCore.HealthChecks.UI, AspNetCore.HealthChecks.UI.InMemory.Storage, HealthChecks.UI.Client

services

.AddHealthChecksUI()

.AddInMemoryStorage();

ResponseWriter = UIResponseWriter.WriteHealthCheckUIResponse, // for MapHealthChecksUI

endpoints.MapHealthChecksUI(); // por defecto, /healthchecks-ui

Y en configuración hay que decirle “que va a chequear la UI”

"HealthChecksUI": {

"HealthChecks": [

{

"Name": "live",

"Uri": "https://localhost:5001/health/"

}

],

"EvaluationTimeInSeconds": 30,

"MinimumSecondsBetweenFailureNotifications": 60

}

endpoints.MapHealthChecksUI(); // por defecto, **/healthchecks-ui**

Además de tener una bonita UI, también podemos configurar el tiempo de refresco, política de reintentos, etc y también tiene webhooks… en el sentido de hacer POST automáticamente si el healthcheck ha fallado <https://github.com/Xabaril/AspNetCore.Diagnostics.HealthChecks/blob/netcore-3.1/README.md#webhooks-and-failure-notifications> <https://github.com/Xabaril/AspNetCore.Diagnostics.HealthChecks/blob/netcore-3.1/doc/webhooks.md> (Slack, Microsoft Teams, Azure Function, etc.)

Sin usar la capacidad de la UI para hacer POST, ¿Si quisiéramos que nuestra aplicación “publicara” proactivamente su información de healthchecks en algún sitio, por ejemplo, un POST a otro endpoint, o publicar un mensaje en una cola, o en un fichero en disco en un recurso compartido?

public class FileHealthCheckPublisher : IHealthCheckPublisher

{

private readonly string \_path;

public FileHealthCheckPublisher(string path)

{

\_path = path;

}

public async Task PublishAsync(HealthReport report, CancellationToken cancellationToken)

{

var message = JsonConvert.SerializeObject(report, new JsonSerializerSettings()

{

Formatting = Formatting.Indented,

ReferenceLoopHandling = ReferenceLoopHandling.Ignore,

Converters = new List<JsonConverter>()

{

new StringEnumConverter()

}

});

await File.AppendAllTextAsync(\_path, message + "\n", cancellationToken);

}

}

services.Configure<HealthCheckPublisherOptions>(options =>

{

options.Delay = TimeSpan.FromSeconds(30); // Publicar por primera vez 30 segundos después de iniciar la aplicación, por defecto 5 segundos, lo que tarden nuestras dependencias (readiness) en levantar

options.Period = TimeSpan.FromSeconds(15); // volver a publicar cada 15 segundos, por defecto 10 segundos

options.Predicate = check =>

{

// filtrar healthchecks

return true;

};

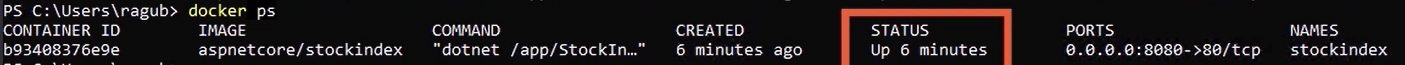
options.Timeout = TimeSpan.FromSeconds(5); // ¿cuánto esperar a que respondan nuestros healtchecks?, por defecto 30

});

services.AddSingleton<IHealthCheckPublisher>(new FileHealthCheckPublisher(@"C:\Temp\health.txt")); // No pasa por app.UseEndpoints, usa directamente los servicios registrados de tipo IHealthCheck

Hay otros escenarios donde aplica los healtchecks:

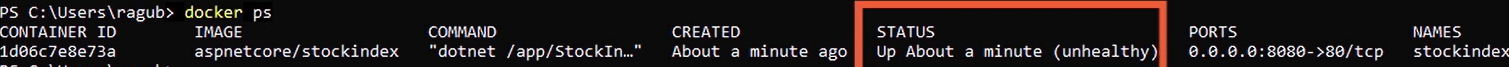
* Docker
* Load balancer
* Azure DevOps



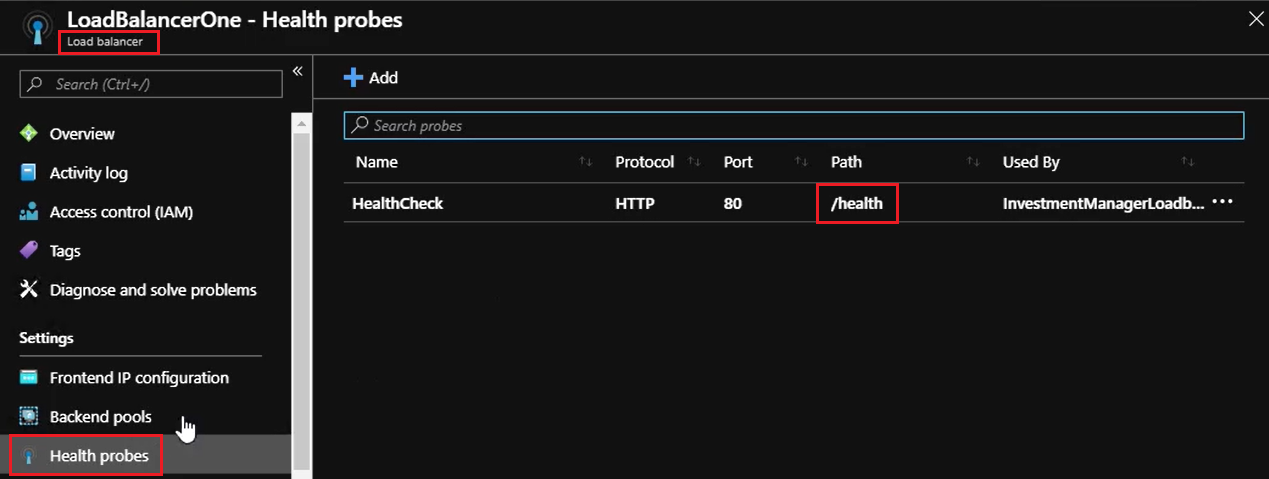
La columna STATUS de un contenedor nada tiene que ver con nuestros health-checks, STATUS es un tema de Docker, no usa nuestros health-checks. Puede estar petando (no conectando a la BD) y STATUS decir que OK, que Up X minutes. Sin embargo, en Dockerfile hay una instrucción HEALTHCHECK <https://docs.docker.com/engine/reference/builder/> y ahí le podemos decir que mire los nuestros (no hace falta k8s para esto).



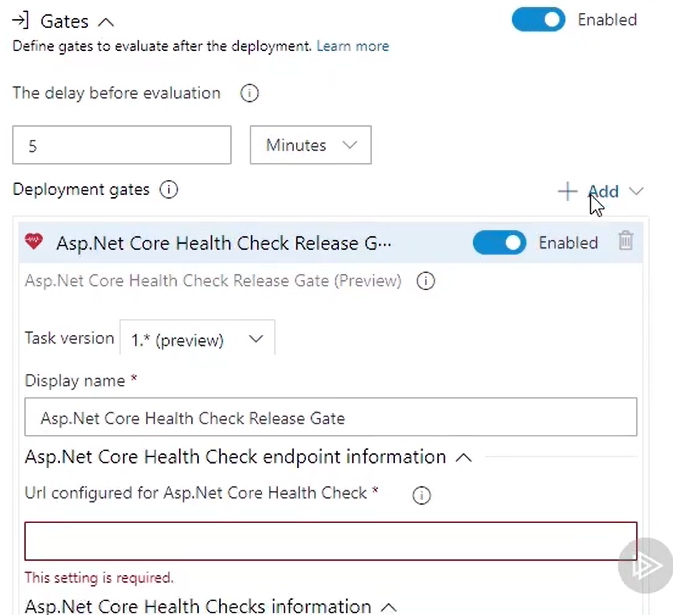
Y ahora ya STATUS de Docker nos hará caso

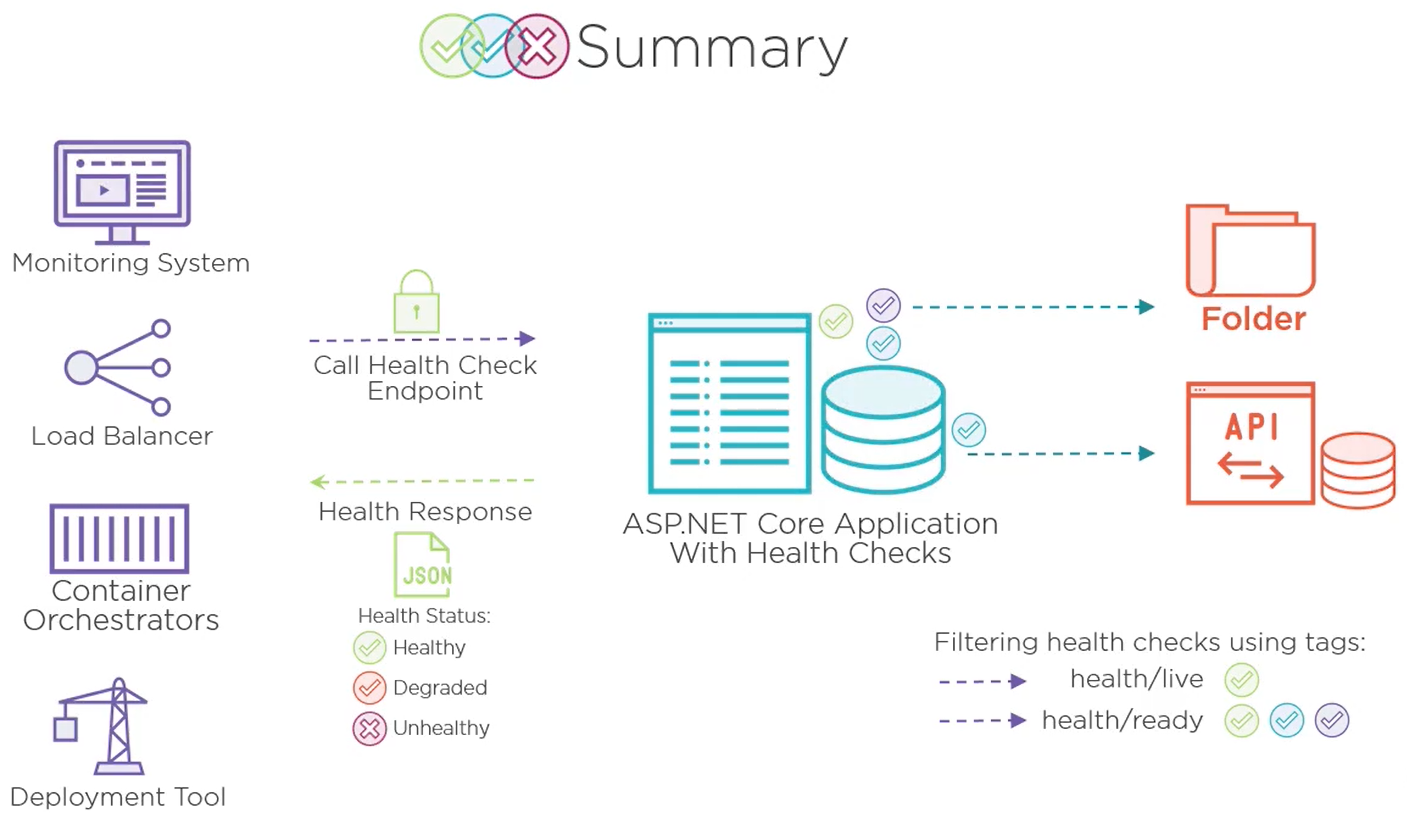


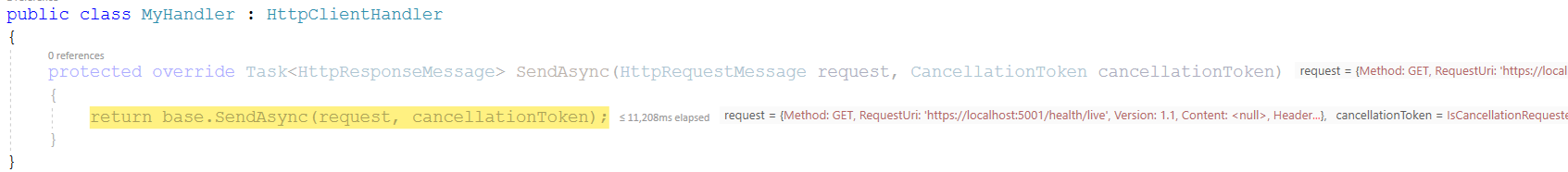
También podemos usar nuestros healthchecks en un **load balancer de Azure** para dirigir el tráfico solo a instancias del backend pool que estén healthy



Y con “gates” en los pipelines de deploy de Azure DevOps, “si fallan nuestros healthcheck haz un rollback de nuestro despliegue”





****

services.AddHealthChecksUI(setupSettings: settings =>

{

settings.UseApiEndpointHttpMessageHandler(provider => new MyHandler()

{

});

});