Understanding Advanced Hosted Service Concepts

Steve Gordon

.NET Engineer and Microsoft MVP

@stevejgordon | www.stevejgordon.co.uk

Overview

- Implementation details
- Handling exceptions
- Triggering application shutdown
- Ordering background services
- Overriding StartAsync and StopAsync
- Testing worker services

BackgroundService Implementation Details

Add exception handling

Why exception handling is important

Hosted Service Unhandled Exception Behavior

Since .NET 6, by default, unhandled exception cause the application to shutdown

In prior .NET versions, the application would continue to run

Unhandled exceptions represent a failure that cannot be automatically recovered

Exceptions are logged before shutdown

Unhandled exceptions should not be ignored

The shutdown behavior can be configured

How lifetime is managed by the host

How the shutdown process works

Trigger shutdown

Configure shutdown options

IHostApplicationLifetime

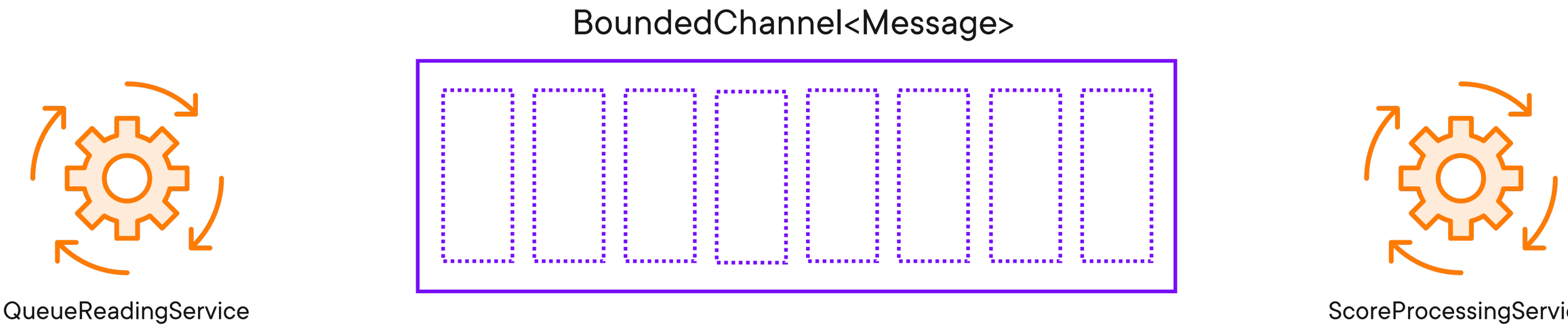
Allows consumers to be notified of application lifetime events.

IHostApplicationLifetime.cs

```
public interface IHostApplicationLifetime
{
    CancellationToken ApplicationStarted { get; }
    CancellationToken ApplicationStopping { get; }
    CancellationToken ApplicationStopped { get; }
    void StopApplication();
}
```

It’s safe to call the StopApplication method multiple times.

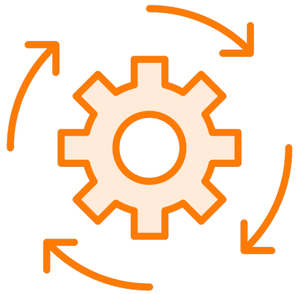
Failing to Call StopApplication



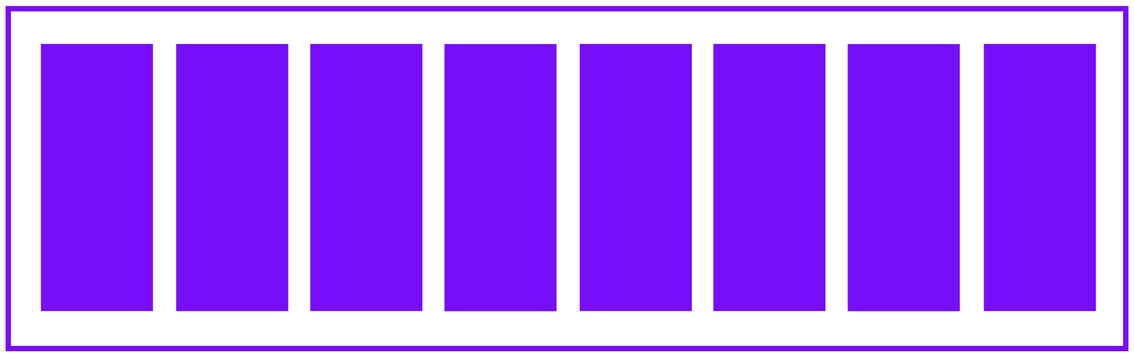
Failing to Call StopApplication

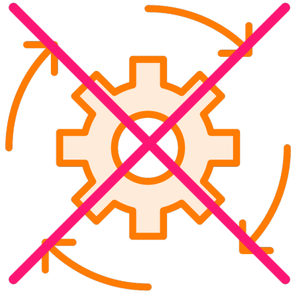
Unhandled Exception

BoundedChannel<Message>



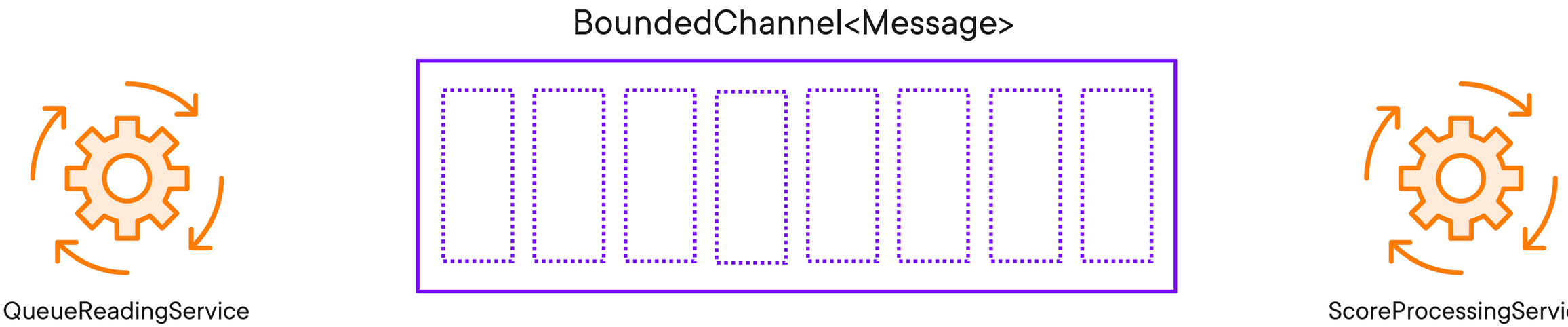
QueueReadingService





ScoreProcessingService

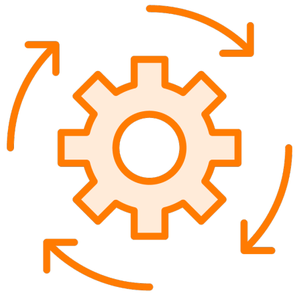
Failing to Call StopApplication



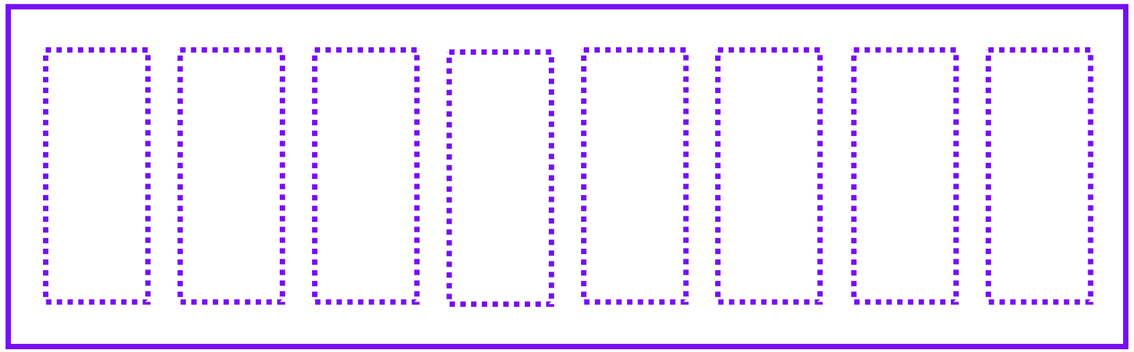
Failing to Call StopApplication

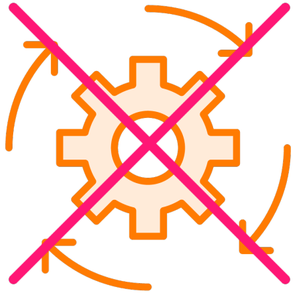
Unhandled Exception

BoundedChannel<Message>



QueueReadingService

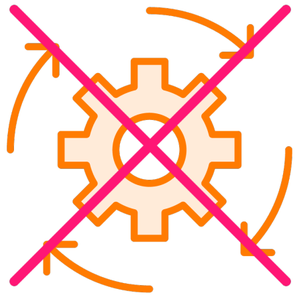




ScoreProcessingService

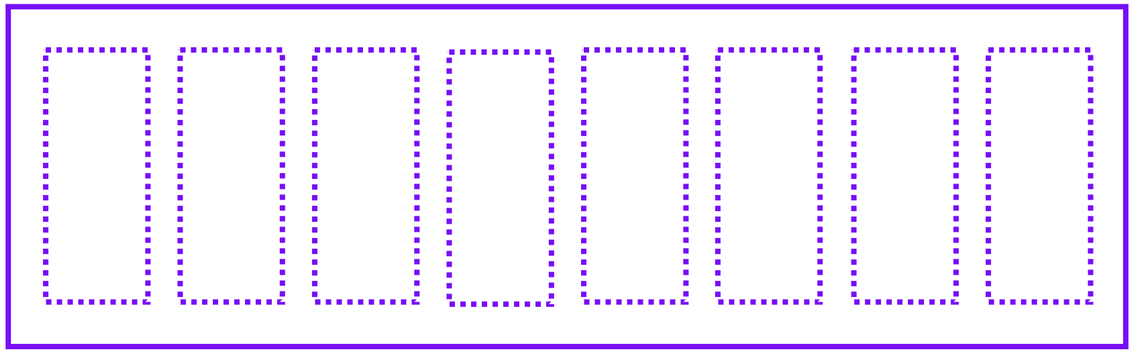
Failing to Call StopApplication

Graceful Shutdown

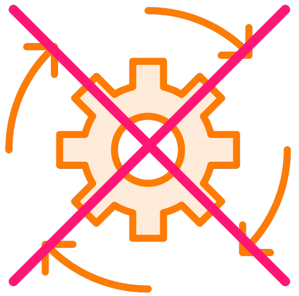


QueueReadingService

BoundedChannel<Message>



Unhandled Exception



ScoreProcessingService

Other Considerations

Monitor shutdowns in production

Respond to and correct any causes

Log and record metrics about shutdowns

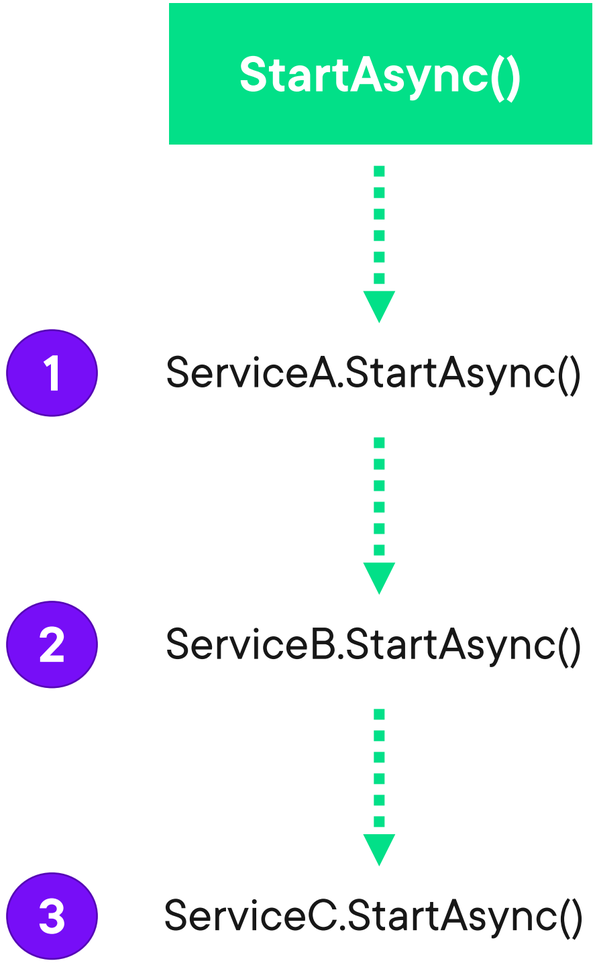
Registration Order of Background Services

The registration order of background services can be important.

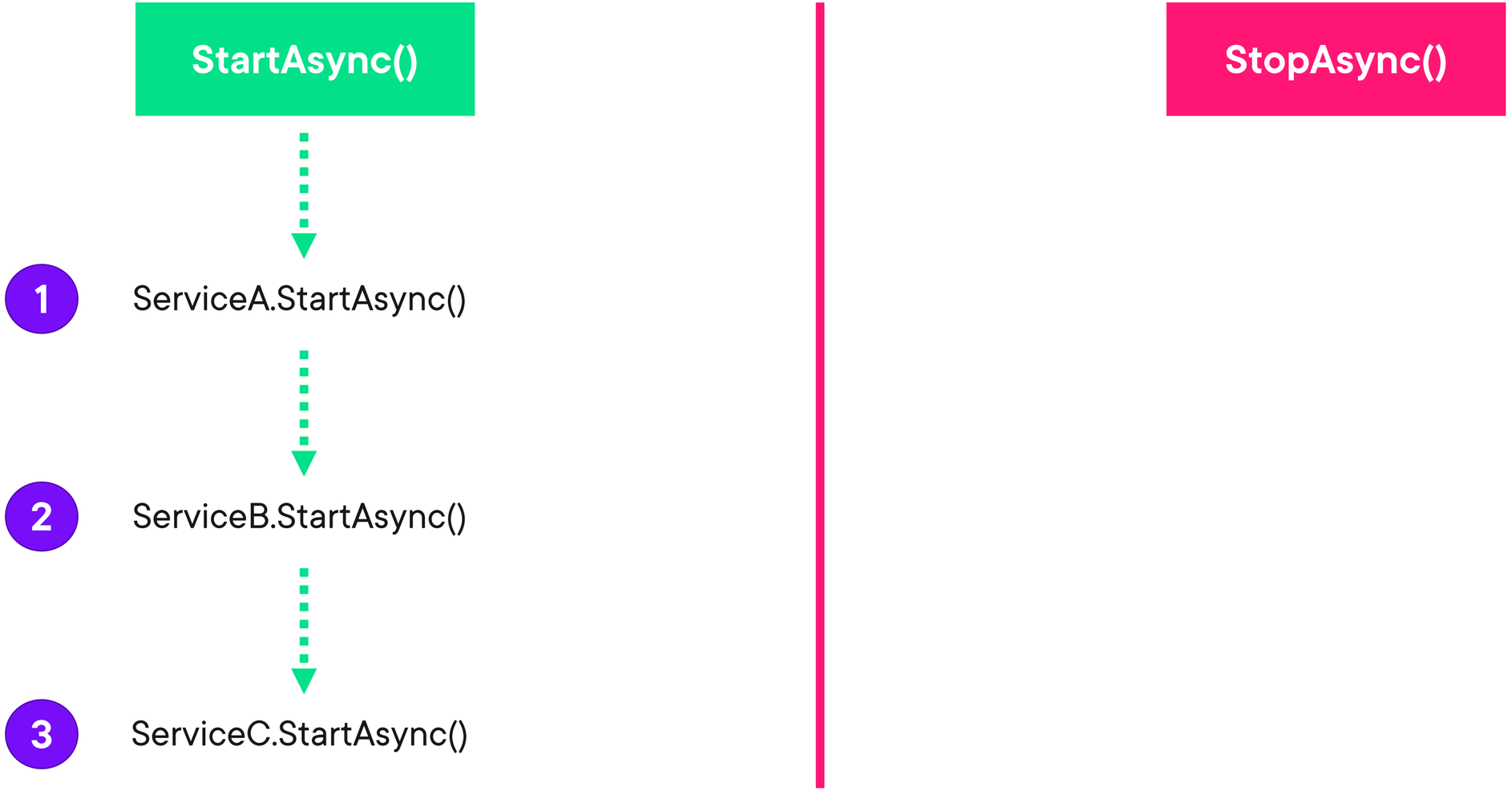
Starting and Stopping Background Services

StartAsync()

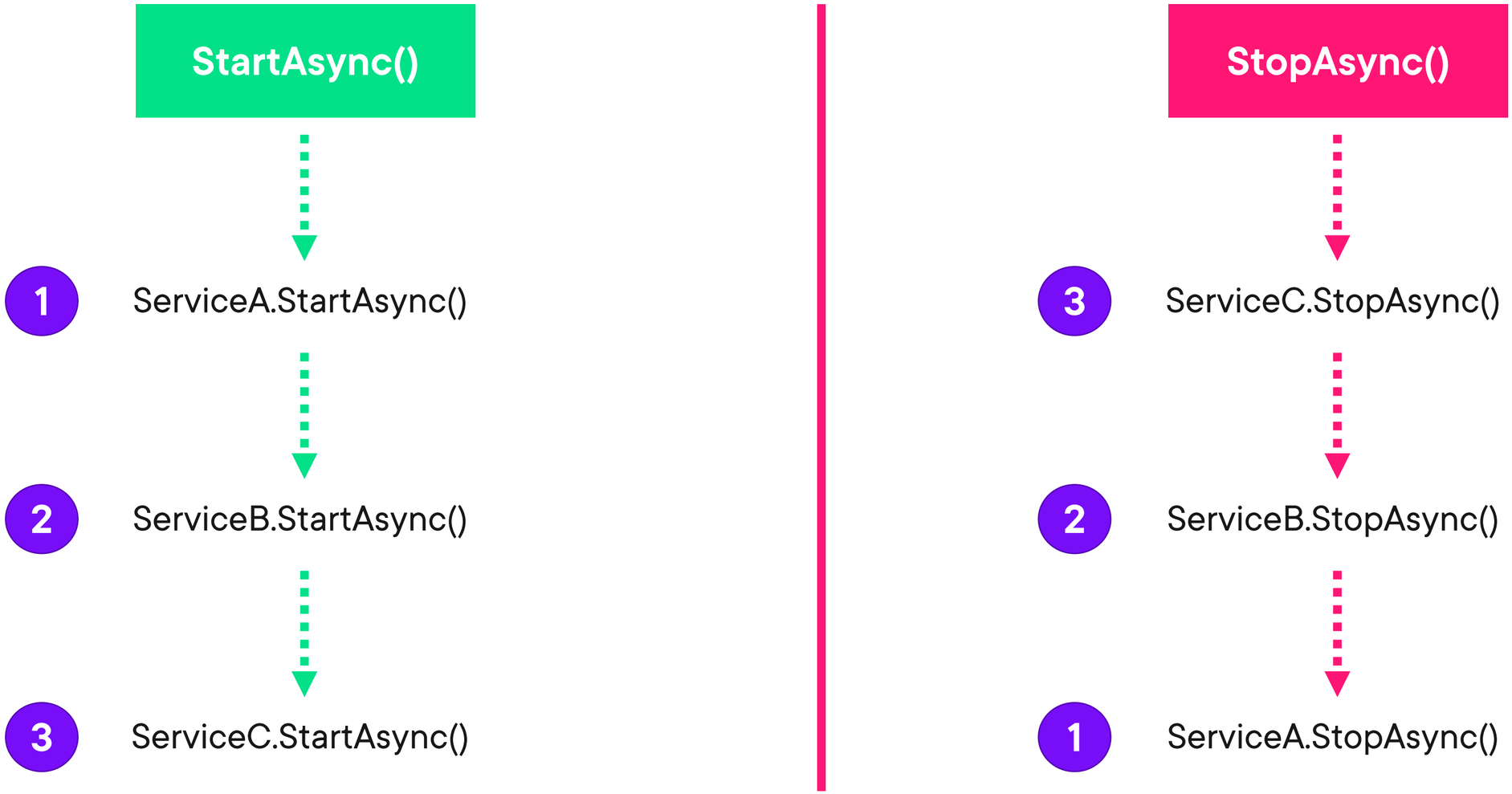
Starting and Stopping Background Services



Starting and Stopping Background Services



Starting and Stopping Background Services



Configuring the Host

There are other methods we can call on the HostBuilder to further configure the host.

```
Host.CreateDefaultBuilder()
```

HostBuilder Defaults

Loads application configuration

appSettings.json

appsettings.{Environment}.json

Secret Manager (when the app runs in the development environment)

Environment variables

Command-line arguments

Adds logging providers

HostBuilder Defaults

Loads application configuration

appSettings.json

appsettings.{Environment}.json

Secret Manager (when the app runs in the development environment)

Environment variables

Command-line arguments

Adds logging providers

Console

Debug

EventSource

EventLog (Windows only)

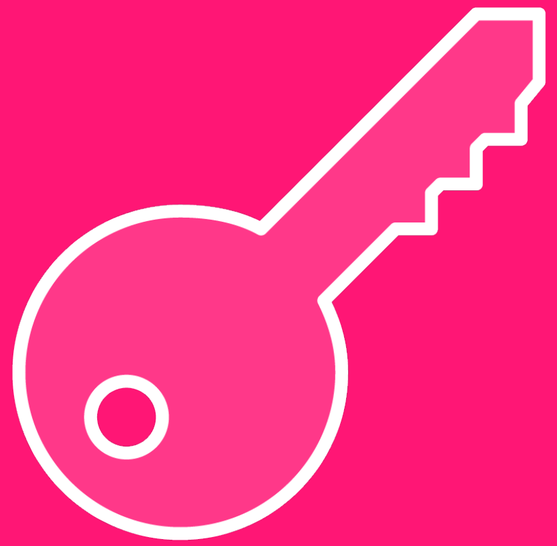
Learn More

Configuration and Options in ASP.NET Core 6

Steve Gordon

Program.cs

```
IHost host = Host.CreateDefaultBuilder()
    .ConfigureServices(services =>
    {
        services.AddSingleton<ISomething, Something>();
    })
    .ConfigureAppConfiguration(config =>
    {
        config.AddInMemoryCollection(new[]
        {
            new KeyValuePair<string, string>("Key", "Value")
        });
    })
    .ConfigureLogging(logging =>
    {
        logging.AddSimpleConsole(opt => opt.IncludeScopes = true);
    })
    .Build();
```



Takeaway

It’s possible to very specifically configure the host.

Override StartAsync and StopAsync

- Add timer logging to StopAsync

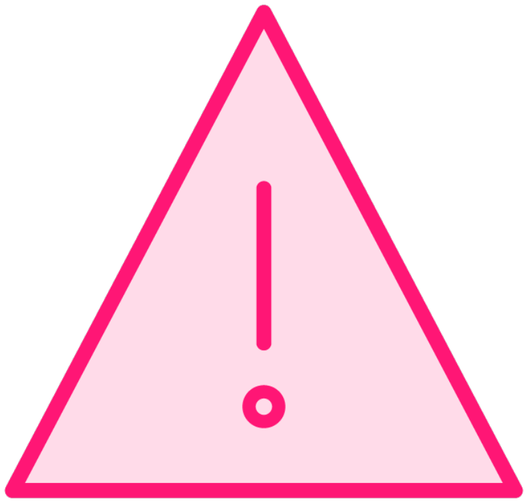
When inheriting from BackgroundService, you won't often need to override StartAsync or StopAsync.

Unit testing background services

Extract most functionality into targeted and more easily testable classes with a single responsibility.

Avoiding Blocking Code in StartAsync

Each hosted service is started sequentially by awaiting its StartAsync method.



StartAsync blocks application startup

StartAsync should complete quickly

Avoid long-running work in StartAsync

ExecuteAsync also blocks startup until the first await

Awaiting an immediately completed Task is effectively synchronous

Summary

- Handling exceptions
- Triggering and handling shutdown
- Registration order of hosted services
- Overriding StartAsync and StopAsync
- Unit testing hosted services
- Avoiding startup delays in hosted services

Up Next:

Running Worker Services in Production