# The Background on Background Tasks in .NET 6

Slides and code at scottsauber.com

scottsauber

# Audience

- .NET Developers
- In need of running a background task

# Agenda

- What are background tasks/jobs?
- What type of problems are suitable for a background task/job?
- What options are out there?
  - IHostedService
  - BackgroundService
  - Worker Service
  - Hangfire
- Why would I choose one over the other?
- Deep dive into each
- Demos
- Questions

# Goal

- Know all your options for running background tasks
- Why choose one over another

# Who am I?

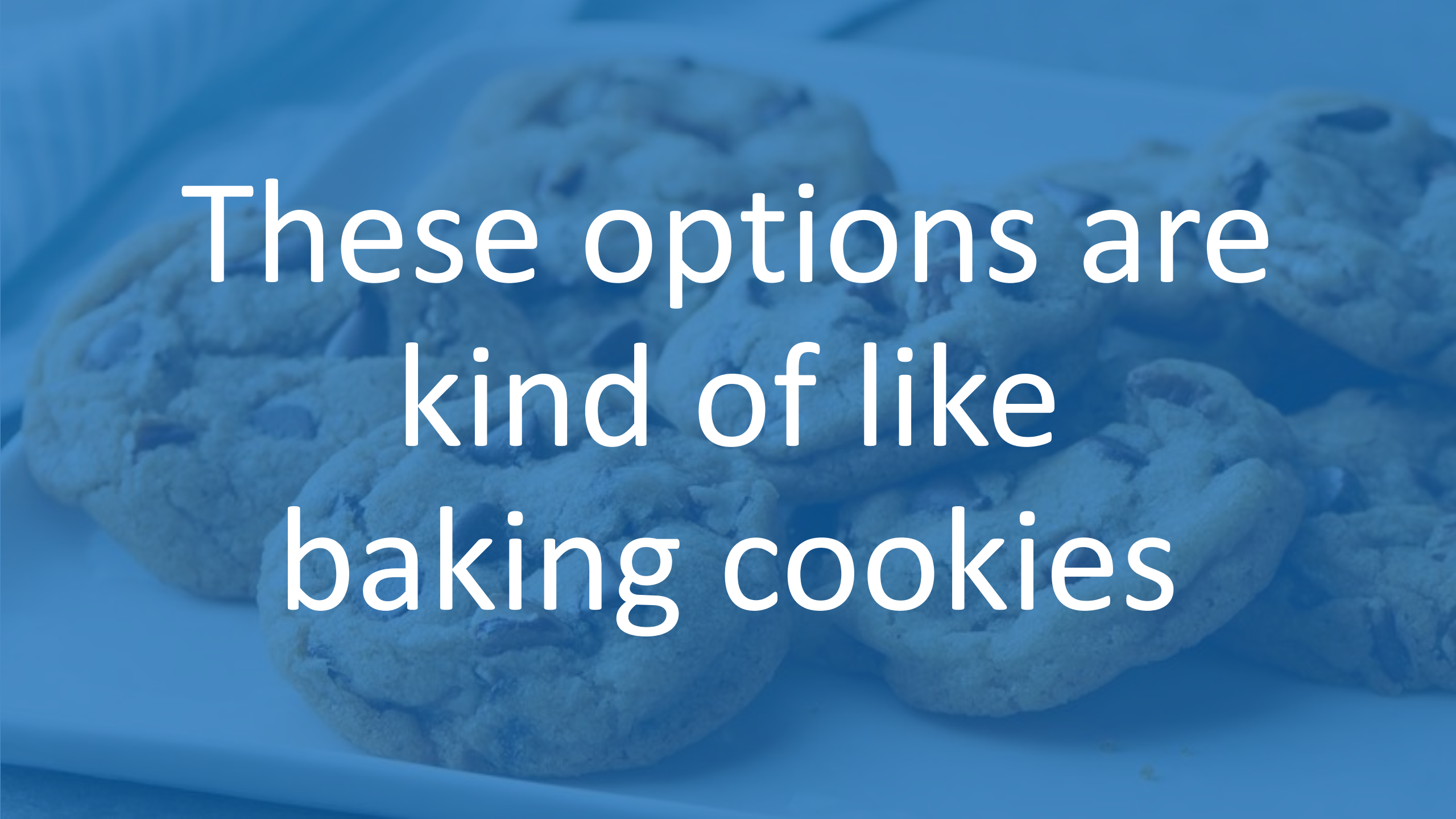
- Director of Engineering at Lean TECHniques
- Co-organizer of Iowa .NET User Group
- Friend of Redgate
- Blog at scottsauber.com

# What problem do background tasks solve?

- Cron jobs
  - Process messages from a queue every X minutes
  - Clean up database or file system every X minutes
  - Send email notification every X minutes under certain circumstances
  - Refresh cache every X minutes
  - Check for updates to database every X minutes and push updates via SignalR
- Perform some CPU intensive work asynchronously
- Eventual consistency
- Re-train ML datasets

# Options

- IHostedService
- BackgroundService
- WorkerService
- Hangfire
- Cloud options


These options are kind of like baking cookies


# IHostedService

## “Make Your Own Recipe”

## (Cookie Jar Included)

# What is an IHostedService?

- Lets you host a background job inside an ASP.NET Core App
  - ASP.NET Core app is your cookie jar
- Interface with StartAsync and StopAsync
- Raw, fundamental building block for other options
- Register via dependency injection and services.AddHostedService<T>

# Demo

COOKIE
TASTE TEST

# How does an IHostedService work?

- Register with DI

```
services.AddHostedService<HostedServiceExample>();
```

- StopAsync's cancellation token has 5 seconds to shutdown gracefully
- StopAsync might not be called if the app shuts down unexpectedly

# How does an IHostedService work?

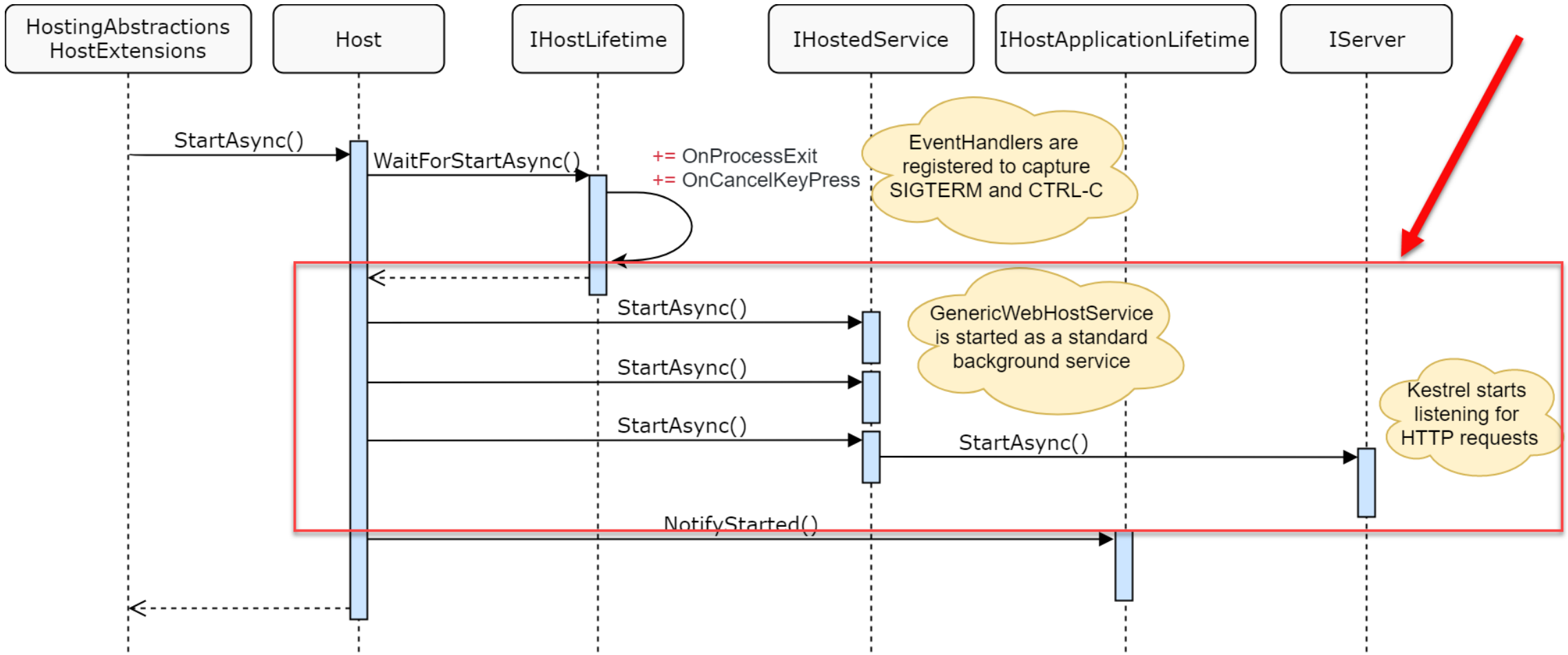



Image Credit: [Andrew Lock]

scottsauber

# How does an IHostedService work?

- StartAsync blocks the rest of your app from starting
- Push ***blocking*** long-running work out of StartAsync
  - This goes for BackgroundService later

DO THIS

```
public Task StartAsync(CancellationToken cancellationToken)
{
    LongRunningThingAsync(cancellationToken);

    return Task.CompletedTask;
}
```

NOT THIS

```
public async Task StartAsync(CancellationToken cancellationToken)
{
    await LongRunningThingAsync(cancellationToken);
}
```

- UNLESS, you truly don't want your app to boot until this finishes
  - i.e. Database Migrations

# When do I use IHostedService?

- You will implicitly use it with BackgroundService and Worker Services
- You need full control over Starting and Stopping
  - AND will not use the base BackgroundService implementation

# When do I *NOT* use IHostedService?

- Should be using BackgroundService or WorkerService 95%+ of the time
- Other reasons will be the same as BackgroundService (next)

# BackgroundService

## “Follow The Recipe”
## (Cookie Jar Included)

# What is a BackgroundService?

- Lets you host a background job inside an ASP.NET Core App
  - ASP.NET Core app is your cookie jar
- Abstract class, implements IHostedService
- Exposes ExecuteAsync abstract method
- Handles Starting and Stopping

# Demo

COOKIE
TASTE TEST

# How does a BackgroundService work?

- Register with DI

```
services.AddHostedService<BackgroundServiceExample>();
```

- Exposes ExecuteAsync abstract method
- Can still override StartAsync and StopAsync

```
public abstract class BackgroundService : IHostedService, IDisposable
{
    private Task _executingTask;
    private readonly CancellationTokenSource _stoppingCts = new CancellationTokenSource();

    1 usage  1 override
    protected abstract Task ExecuteAsync(CancellationToken stoppingToken);

    public virtual Task StartAsync(CancellationToken cancellationToken)
    {
        // Store the task we're executing
        _executingTask = ExecuteAsync(_stoppingCts.Token);

        // If the task is completed then return it, this will bubble cancellation and failure to the caller
        if (_executingTask.IsCompleted)
        {
            return _executingTask;
        }

        // Otherwise it's running
        return Task.CompletedTask;
    }

    public virtual async Task StopAsync(CancellationToken cancellationToken)
    {
        // Stop called without start
        if (_executingTask == null)
        {
            return;
        }

        try
        {
            // Signal cancellation to the executing method
            _stoppingCts.Cancel();
        }
        finally
        {
            // Wait until the task completes or the stop token triggers
            await Task.WhenAny( params tasks: _executingTask, Task.Delay(Timeout.Infinite, cancellationToken));
        }

    }

    public virtual void Dispose()
    {
        _stoppingCts.Cancel();
    }
}
```

# When do I use BackgroundService?

- Need a simple background task runner
  - Either as part of your ASP.NET Core application or by itself
- Less gotchas than IHostedService
  - Can’t accidentally prevent app from booting unless override StartAsync
  - Handles cancellations
- Want an ASP.NET Core endpoint to health check your background task
  - Instead of WorkerServices

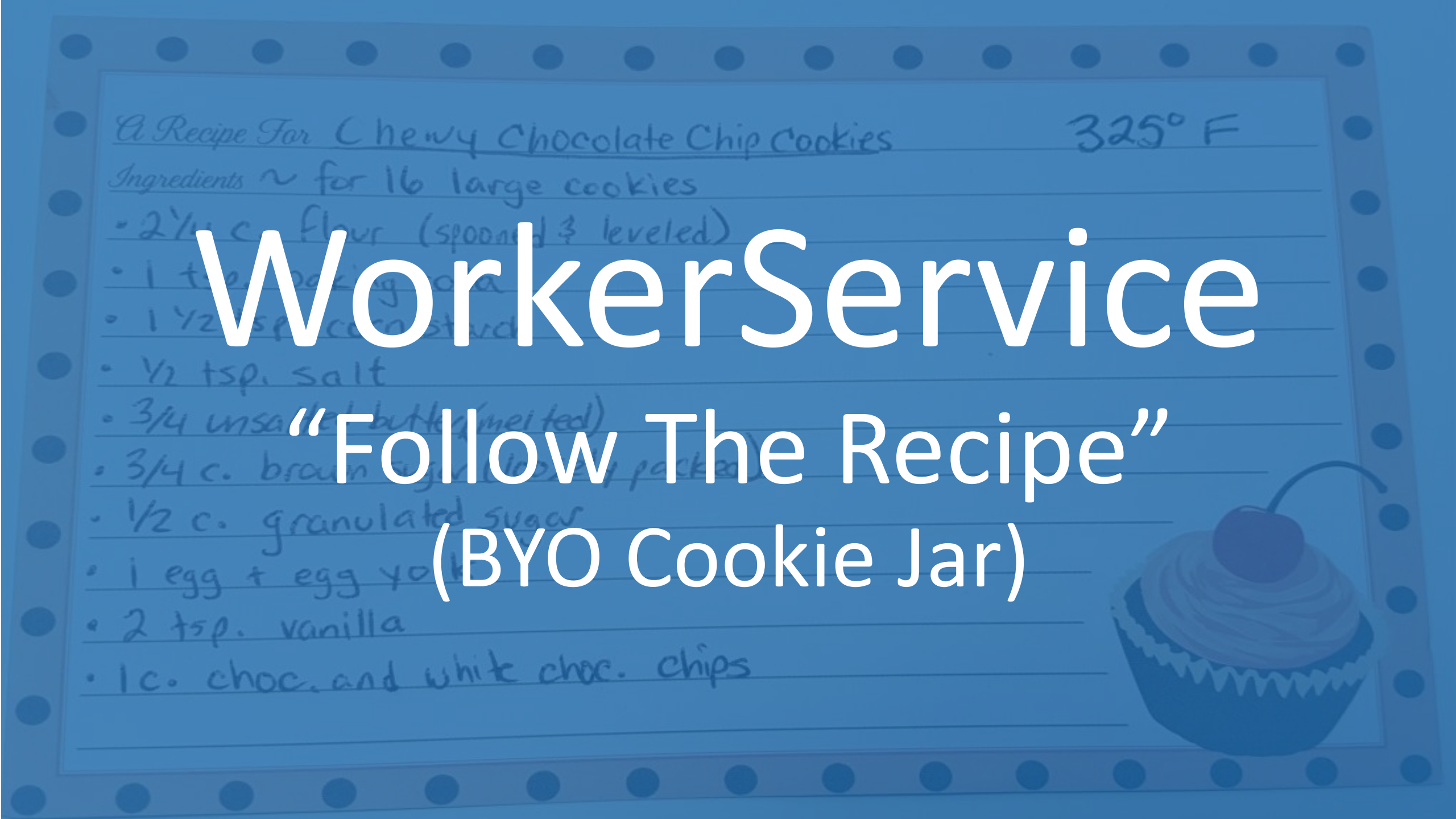
# When do I *NOT* use BackgroundService?

- Too much co-location with your app/API can get unruly and outweigh the convenience of co-location
  - It Depends
- Scaling out can be a problem if your code isn't idempotent
  - Fix by making code idempotent or not allowing scale out

# WorkerService

## “Follow The Recipe”

## (BYO Cookie Jar)



# What is a WorkerService?

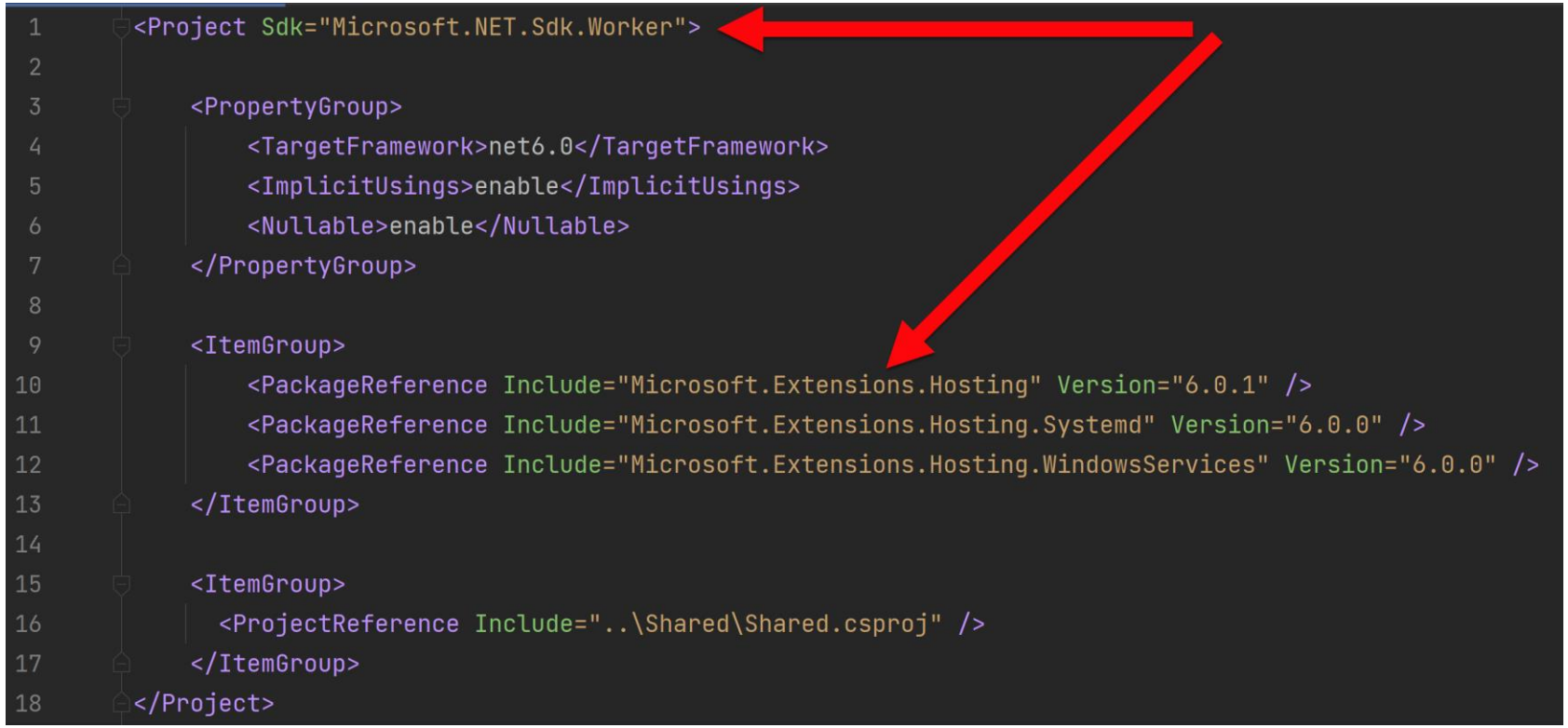
- Enhanced .NET Console App template
  - dotnet new worker –o my-custom-worker
- Allows you to have an IHost
  - Configuration, Dependency Injection, Logging, etc.
- Registers a Worker class as a HostedService
- Does not take an opinion on how to host console app
  - No cookie jar
  - Console app called from scheduler
  - Windows Service
  - systemd

# Demo

COOKIE
TASTE TEST

# How does a WorkerService work?

- Project Sdk of Microsoft.NET.Sdk.Worker
- PackageReference to Microsoft.Extensions.Hosting



# How do I host WorkerServices?

- Scheduler calls Console App
  - Windows Scheduled Tasks, k8s cron jobs, Azure Logic Apps, AWS Scheduled Tasks, GCP Cloud Scheduler
- Windows Service or Systemd (Windows or Linux)

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .UseWindowsService() // Microsoft.Extensions.Hosting.WindowsService
        .UseSystemd() // Microsoft.Extensions.Hosting.Systemd
        .ConfigureServices((hostContext, services) => { services.AddHostedService<Worker>(); });
```

Pick one

# When do I use WorkerServices?

- Want an out-of-proc way of running background tasks
- Prefer hosting background services outside of a web app
  - Avoid app pool recycles
- Natural migration for a full .NET framework Windows Service

# When do I *NOT* use WorkerServices?

- Prefer deploying as a web app
- Want to co-locate with existing web app/API
- Want a healthcheck endpoint


# Hangfire

## “Buy pre-packaged cookies”

# What is Hangfire?

- Full featured library for running jobs in ASP.NET Core
  - Free for commercial use but paid if you want support ($500-$4500/yr)
- Comes with UI for monitoring and history
- Supports Cron and ad-hoc running of jobs
- Allows for continuations
- Automatic retries
- Supports concurrency limiting
- Persists job state to database

# Demo

COOKIE
TASTE TEST

# How does Hangfire work?

- Serializes method call and all arguments
- Creates background job based on that information
- Saves job to persistent storage
- Starts background job if immediate

# When do I use Hangfire?

- Want to host jobs in ASP.NET Core
- Need features Hangfire offers
- Don’t want to write plumbing code
- Ok with relying on a 3rd party library

# When do I *NOT* use Hangfire?

- Do not want to host jobs in ASP.NET Core
- Have basic needs and do not need Hangfire's features
- Do not want to rely on 3rd party library
- More control over what happens

# Cloud options

- Azure Functions
- Azure WebJobs
- AWS Lambdas
- GCP Cloud Scheduler + Cloud Functions
- Didn’t cover these to avoid cloud specific

# Takeaways

- Awareness to all the options available to you
- More information to make the best decision for you and your company

# Resources

- https://docs.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/background-tasks-with-ihostedservice
- https://www.hangfire.io/
- https://app.pluralsight.com/library/courses/building-aspnet-core-hosted-services-net-core-worker-services/
- This slide deck

# Questions?

Contact: ssauber@leantechniques.com

Slides and code at scottsauber.com

scottsauber

# Thanks!

Slides and code at scottsauber.com

scottsauber