LIVE! 360
TECH EVENTS WITH PERSPECTIVE

The Ultimate Education Destination
ORLANDO 2024

The Background On Background Tasks in .NET 9

Scott Sauber
Director of Engineering
Lean TECHniques

Level: Introductory

Session Survey

- Your feedback is very important to us
- Please take a moment to complete the session survey found in the mobile app
- Use the QR code or search for “Converge360 Events” in your app store
- Find this session on the Agenda tab
- Click “Session Evaluation”
- Thank you!

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

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

Goals

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

Who am I?

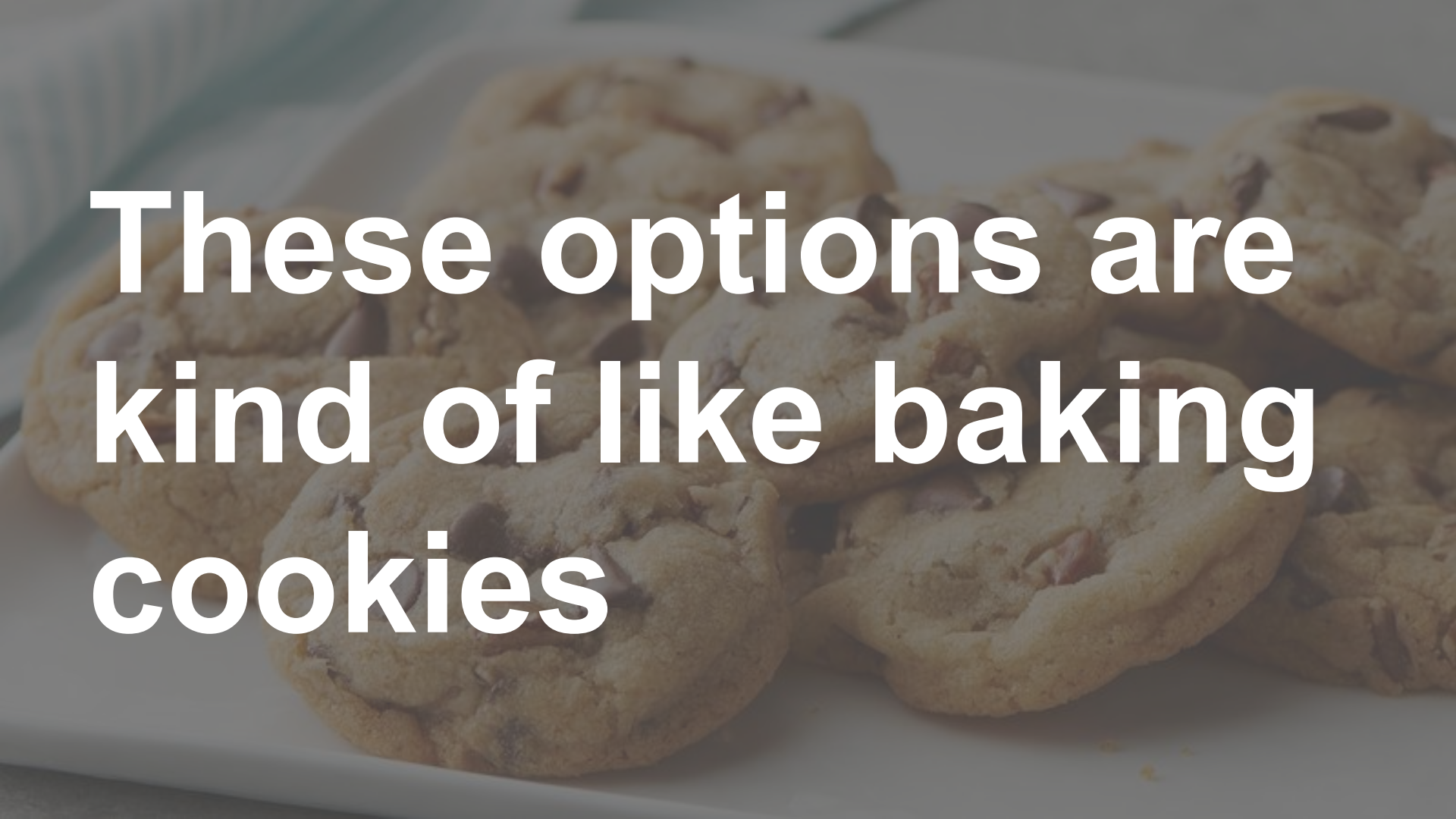
- Director of Engineering at Lean TECHniques
- Microsoft MVP
- Dometrain author
- Redgate Community Ambassador
- Co-organizer of Iowa .NET User Group

What problems do background tasks solve?

- Cron jobs
- Perform CPU intensive task async
- 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 IHostedService?

- Host background job inside ASP.NET Core
- ASP.NET Core is your cookie jar
- Interface – StartAsync, StopAsync
- Raw fundamental building block
- Register: services.AddHostedService<T>

Demo

How IHostedService works

- Register in DI

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

- StopAsync cancellation has 5 seconds to shutdown gracefully
- StopAsync might not get called if app shuts down unexpectedly

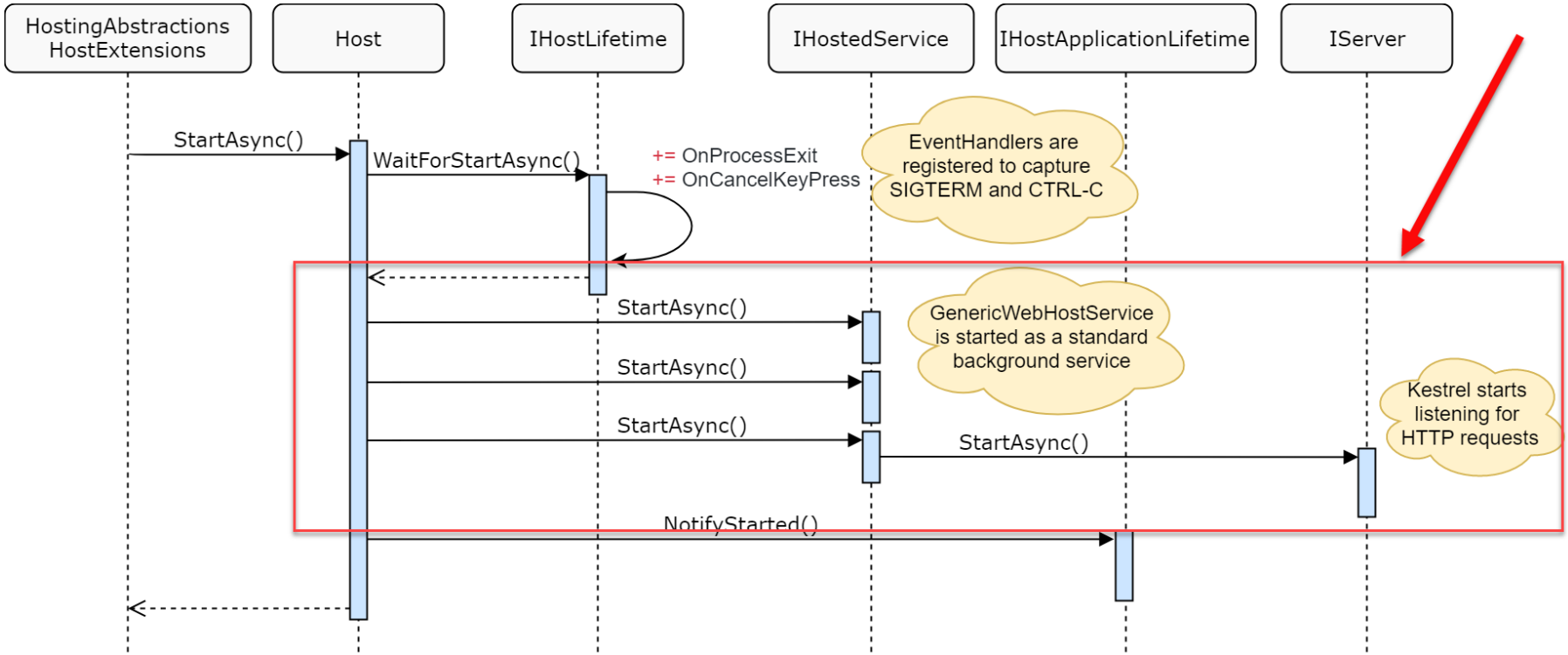


Image Credit: Andrew Lock

How IHostedService works

- StartAsync blocks rest of app from starting
- Push ***blocking*** long running work out of StartAsync
- UNLESS you truly don’t want your app to boot until it finishes
- Database Migrations, Cache Refresh, etc

DO THIS

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

    return Task.CompletedTask;
}
```

NOT THIS

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

When to use IHostedService

- Implicitly used with BackgroundService and Worker
- Also Kestrel!
- You need full control over Starting/Stopping
- Don’t want to use BackgroundService implementation

When NOT to use IHostedService

- Should be using BackgroundService/Worker 95%+ of the time
- Other reasons same as BackgroundService (next)

BackgroundService

“Follow the recipe”
(Cookie jar included)

What is BackgroundService?

- Host background job inside ASP.NET Core
- ASP.NET Core is your cookie jar
- Abstract class, implements IHostedService
- Exposes ExecuteAsync abstract method
- Handles starting and stopping
- Register: services.AddHostedService<T>

Demo

How BackgroundService works

- Register with DI

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

- Exposes ExecuteAsync abstract method
- Can still override StartAsync + StopAsync
- Default StartAsync implementation WILL NOT block app from starting
- Handles cancellations (app stopping)

```
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 to use BackgroundService

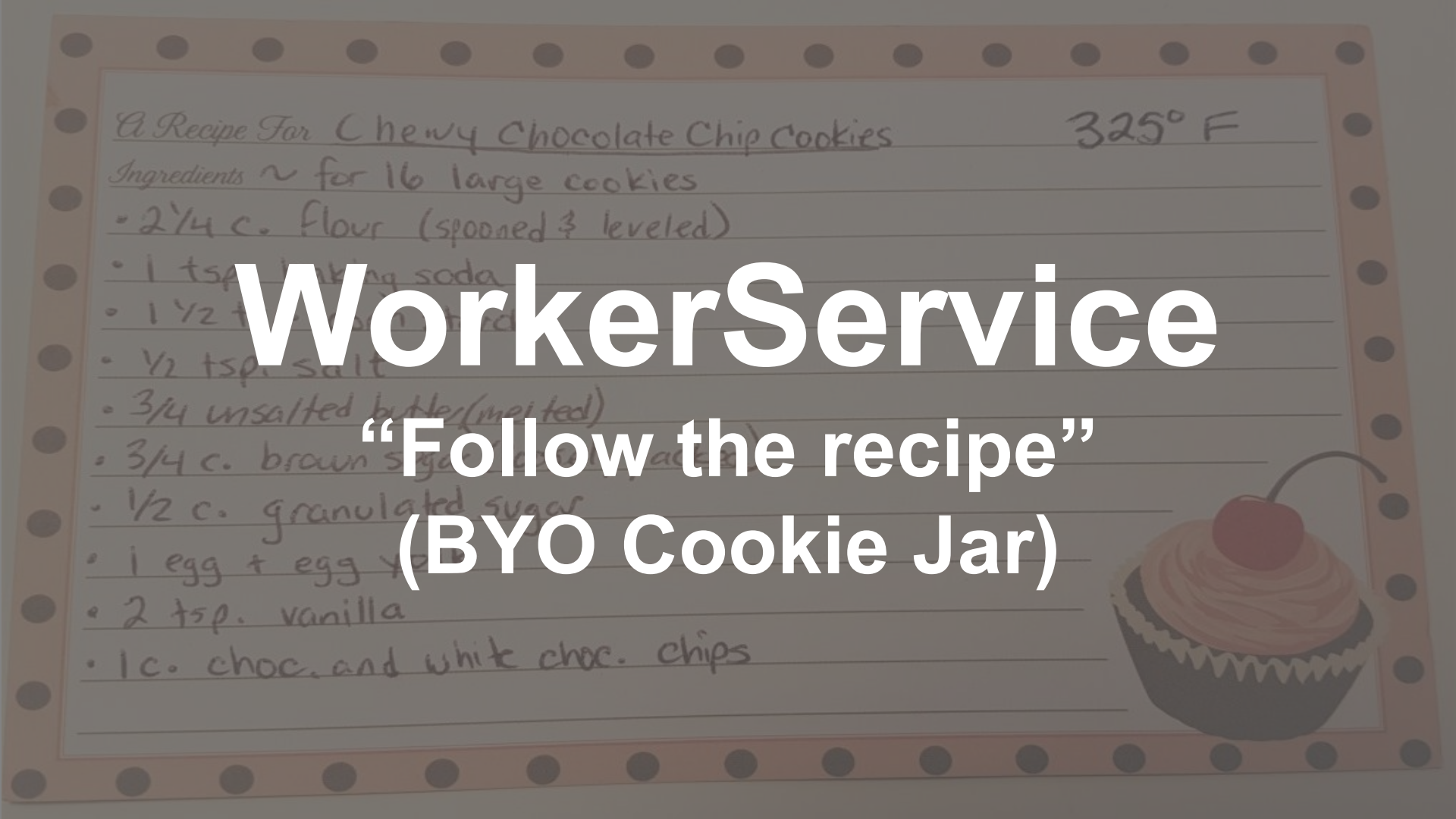
- Need simple background task runner as part of ASP.NET Core application
- Less gotchas than IHostedService
- Want an ASP.NET Core health check endpoint for your background task (instead of WorkerServices)

When NOT to use BackgroundService

- Too much co-location with app can get unruly
- It Depends™
- Scaling out if code isn’t idempotent
- Or you could just make your code idempotent or not allow scale out (I guess)

WorkerService

“Follow the recipe”
(BYO Cookie Jar)



What is a WorkerService

- Enhanced .NET Console app template
- dotnet new worker –o my-worker
- Gives you IHost
  - Configuration, DI, Logging, etc
- Registers Work class as HostedService
- Does not take opinion on how to host console app
- No cookie jar… scheduler? Windows Service? systemd?

Demo

How WorkerService works

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

```
<Project Sdk="Microsoft.NET.Sdk.Worker">

    <PropertyGroup>
        <TargetFramework>net9.0</TargetFramework>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="Microsoft.Extensions.Hosting" Version="9.0.0"/>
        <PackageReference Include="Microsoft.Extensions.Hosting.WindowsServices" Version="9.0.0"/>
        <PackageReference Include="Microsoft.Extensions.Hosting.Systemd" Version="9.0.0"/>
    </ItemGroup>
</Project>
```

How do I host WorkerServices?

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

```
var builder = Host.CreateApplicationBuilder(args);
builder.Services.AddHostedService<Worker>();
builder.Services.AddWindowsService(options => options.ServiceName = "My Worker");
builder.Services.AddSystemd();
```

Pick one

When to use WorkerService

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

When NOT to use WorkerService

- Prefer deploying web apps
- Want to co-locate with existing wen app/API
- Want a health check endpoint

Hangfire

“Buy pre-packaged cookies”

What is Hangfire?

- Full featured library for running jobs in ASP.NET Core
- Free commercial use, but paid for support
- Comes with UI for monitoring and history
- Supports Cron and ad-hoc running of jobs
- Allows for continuations
- Automatic retries
- Support concurrency limiting
- Persists job state to database

Demo

How does Hangfire work?

- Serialize method call and arguments
- Creates background job based on that info
- Saves job to persistent storage
- Starts background job if immediate

When to 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 3rd party library

When NOT to use Hangfire?

- Do not want to host jobs in ASP.NET Core
- Have basic needs, don’t need Hangfire’s features
- Do not want to rely on 3rd party library
- Want more control over what happens

Cloud options

- Azure Functions with Scheduling timer
- Azure WebJobs
- AWS Lambdas
- GCP Cloud Scheduler + Cloud Functions
- Didn’t cover these to avoid cloud specific
- Everything we covered works with any cloud

Takeaways

- Awareness to all the options available to you
- Awareness of the pro’s and con’s of the options
- Make the best decision for you and your company

Resources

- https://learn.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/background-tasks-with-ihostedservice
- https://hangfire.io
- This slide deck

Questions?

Session Survey

- Your feedback is very important to us
- Please take a moment to complete the session survey found in the mobile app
- Use the QR code or search for “Converge360 Events” in your app store
- Find this session on the Agenda tab
- Click “Session Evaluation”
- Thank you!

LIVE! 360
TECH EVENTS WITH PERSPECTIVE