Sebastian HIRTIE
Anul IV, AC-IS

Tema 2 – DATC

Background Jobs

In mod normal cand se creaza un software codul se ruleaza pe acelasi thread ca si el. Cee a ce inseamna ca, inafara instructiunilor respective, nimic altceva nu se mai poate executa. Fapt ce poate surprinde pe programatori incepatori care constata ca interfata aplicatiei lor se "blocheaza" in timp ce in spate ruleaza un proces. O solutie poate fi decuplarea functiilor care consuma timp de partea grafica, adica partea de procesare sa fie delegate sau atribuita altei entitati.

Functinalitatea unui task care ruleaza in spate trebuie sa fie separabila de restul, sa poata fi facuta automat fara interventia unui utilizator.

In functie de motivul folosiri unui worker s-au format urmatoarele categori :

- Volum mare de date la operatii de scriere sau citire dintr-un fisier de pe disk sau din baza de date
- Functii care necesita putere mare de calcul, de procesare, algoritmi complexi
- Traseul lung al datelor de la o componenta la alta
- Data transmise si procesate sau verificate intr-un task separate pentru a minimiza sansa ca cineva sa le poata accesa
- Procese de back-up, de patch sau de update care au loc periodic

Folosirea unui background worker imbunatateste timpul de raspuns al aplicatiei si previne blocarea ei in ce priveste preluarea datelor deoarece nu prelucrarea lor se face in alta parte, deci mareste numarul ultilizatorilor care pot accesa aplicatia.

La un worker de fundal se poate asculta un eveniment care ofera un raport asupra progresului sau un semnal in cazul in care treaba este finalizata. Deasemenea functionalitatea lui poate fi declansata:

- de un eveniment cum ar fi un apel de tip HTTP al unui API, de aparitia unui mesaj intr-o coada la care se asculta, sau modificarea starii unui spatiu de stocare
- programate periodic de exemplu zilnic , o data pe luna sau pricinuite de un timer sincronicat la sistem, fie initializat intr-un alt proces
- executate la cerere

Astlfe de procese de fundal trebuie sa fie active cat timp aplicatia este folosita sau trebuie sa ruleze in permanenta ceea ce necesita o stocare in cloud. Aici vine in ajutor Microsoft Azure pentru rularea lor in cloud dar si oferind webjob cu trigger-ele obisnuite sau azure functions care sunt o extindere de la webjob cu trigger HTTP sau o schimbare in baza de date.

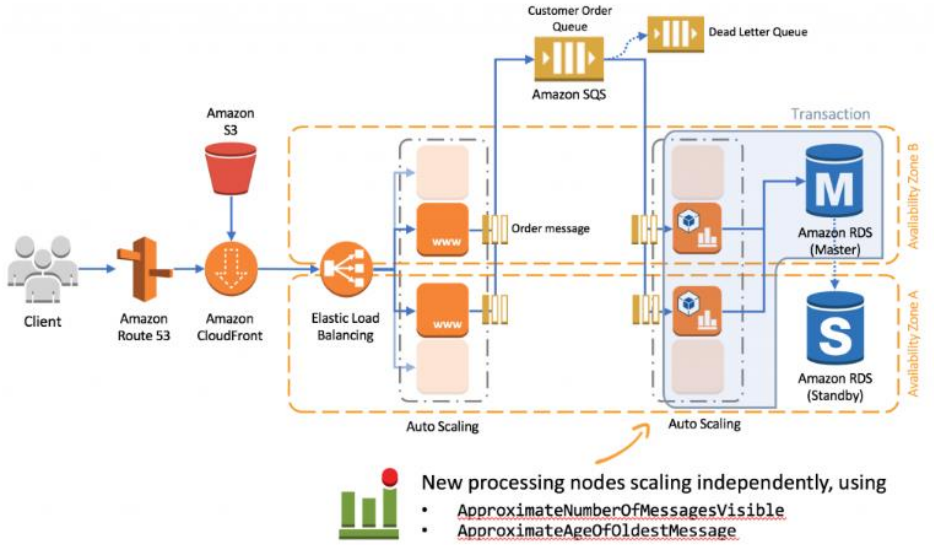
Web worker presupune rularea unui script in fundal, ruland pe un context diferit de cel in care este initializat, el nu poate folosi elemente de interfatare sau dom, insa are acces la obiecte utile cum ar fi socket-uri, http, xml, baza de date. Desi un worker obisnuit care creat pe sistemul

de operare poate ajunga la o problema de siguranta a firului de executie, web worker nu are acces la dom sau la elemente care ar putea pune in pericol de thread.

Cea mai mare problema in aplicati windows este ca partea grafica nu poate fi modificata de pe un alt fir de executie. Pe partea de c# avem BackgroundWorker care permite rularea pe un thread dedicate operatiilor care necesita mult timp puse in DoWork, care poate fi inceputa prin RunWorkerAsync. Pentru a anunta un alt fir de executie destarea procesului sau daca este nevoie de o actualizare a interfetei vizuale se poate folosi evenimentul ProgressChanged sau RunWorkerCompleted pentru notificare dupa terminare completa.

S-ar putea intampla sa apara conflicte sau inconsistente in baza de date in cazul in care exista mai mult instante al worker-ului. Ca si solutie se poate merge pe conceptul de concurenta pesimista (blocarea resursei in timpul modificarii ei, doar un singur utiliza resursa la un moment dat) sau procesul sa fie definit de tip singleton astfel incat sa se garanteze ca o singura instanta ruleaza.

Conceptul de background worker si utilitatea lui se poate observa analizand exemplul de procesare si pasi de executie ai unei comenzi. Provocarile care se ridica sunt administrarea si consistenta datelor cat si numarul ridicat de utilizatori, trafic care ar putea provoca blocari ale sistemului software.



Adaugarea unei cozi ajuta prin decuplarea procesului de prelucrare a datelor de aplicatia web astfel incat in momentele de suprasolicitare, deconectarea de la retea sau caderea tensiunii datele sunt pastrate in coada si nu sunt piedute.

```
var sendMessageRequest = new SendMessageRequest
{
    QueueUrl = _queueUrl,
    MessageBody = JsonConvert.SerializeObject(order),
    MessageGroupId = Guid.NewGuid().ToString("N"),
    MessageDeduplicationId = Guid.NewGuid().ToString("N")
};
```

Pentru prelucrarea informatiilor se foloseste background worker care preia mesaju din coada si il proceseaza in functie de logica dorita.

```
if (receiveMessageResponse.Result.Messages != null)
{
    foreach (var message in receiveMessageResponse.Result.Messages)
    {
        Console.WriteLine("Received SQS message, starting worker thread");

        // Create background worker to process message
        BackgroundWorker worker = new BackgroundWorker();
        worker.DoWork += (obj, e) => ProcessMessage(message);
        worker.RunWorkerAsync();
    }
}
```

Prin arhitectura sugerata se ofera mechanism de decuplare prin coada astfel incat logica se poate schimba independent de interfata grafica, un sistem robust, tolerant la defectiuni si scalabil

ukihttps://www.hanselman.com/blog/IntroducingWindowsAzureWebJobs.aspx
https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers
https://docs.microsoft.com/en-us/azure/architecture/best-practices/background-jobs
https://aws.amazon.com/blogs/compute/building-loosely-coupled-scalable-c-applications-with-amazon-sqs-and-amazon-sns/