Circuit Breaker

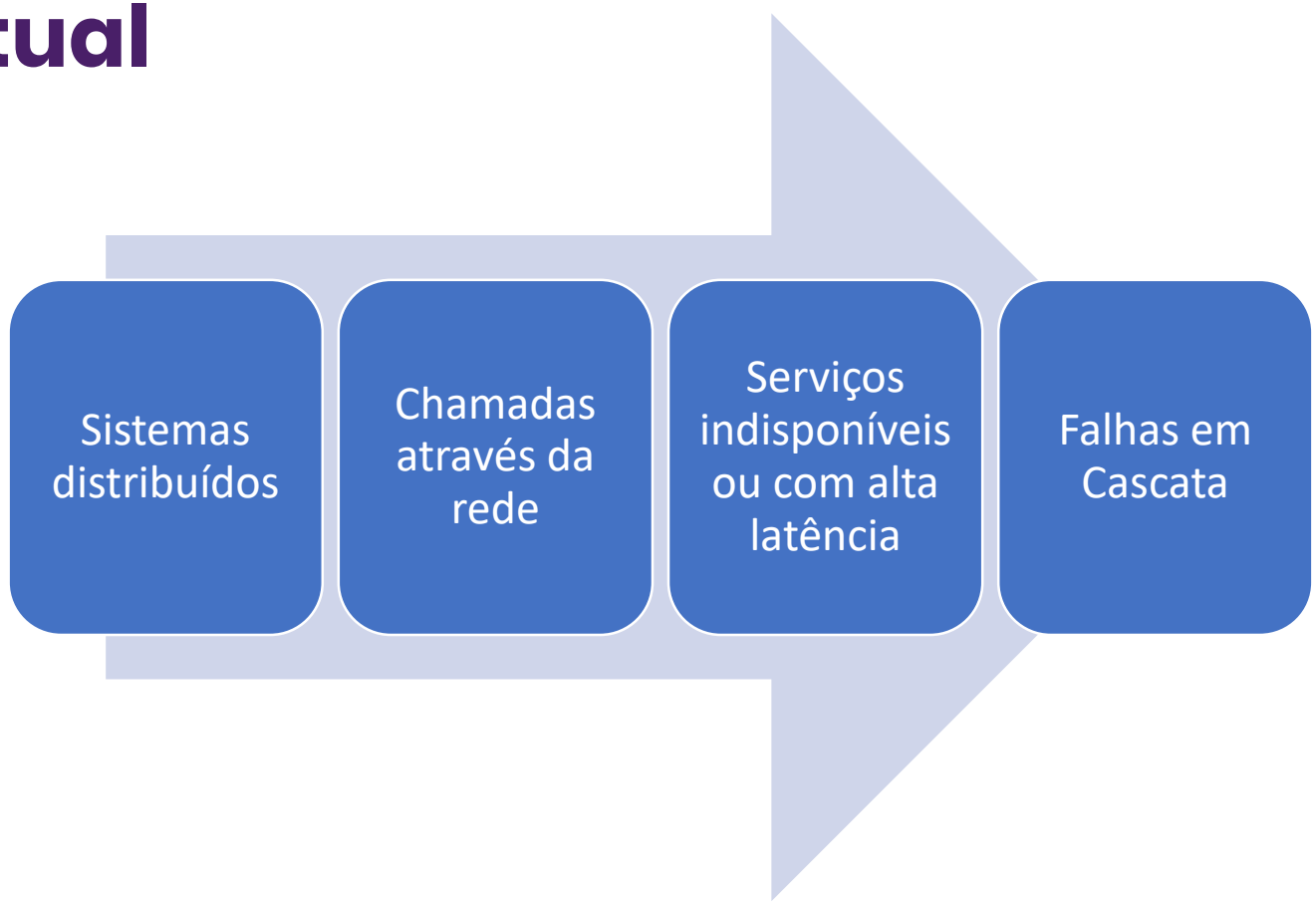
Implementação com Azure Functions e Service Bus

1. Circuit Breaker Pattern
2. Azure Functions
3. Estudo de caso

Circuit Breaker Pattern

Circuit Breaker Pattern

Cenário atual



Circuit Breaker Pattern

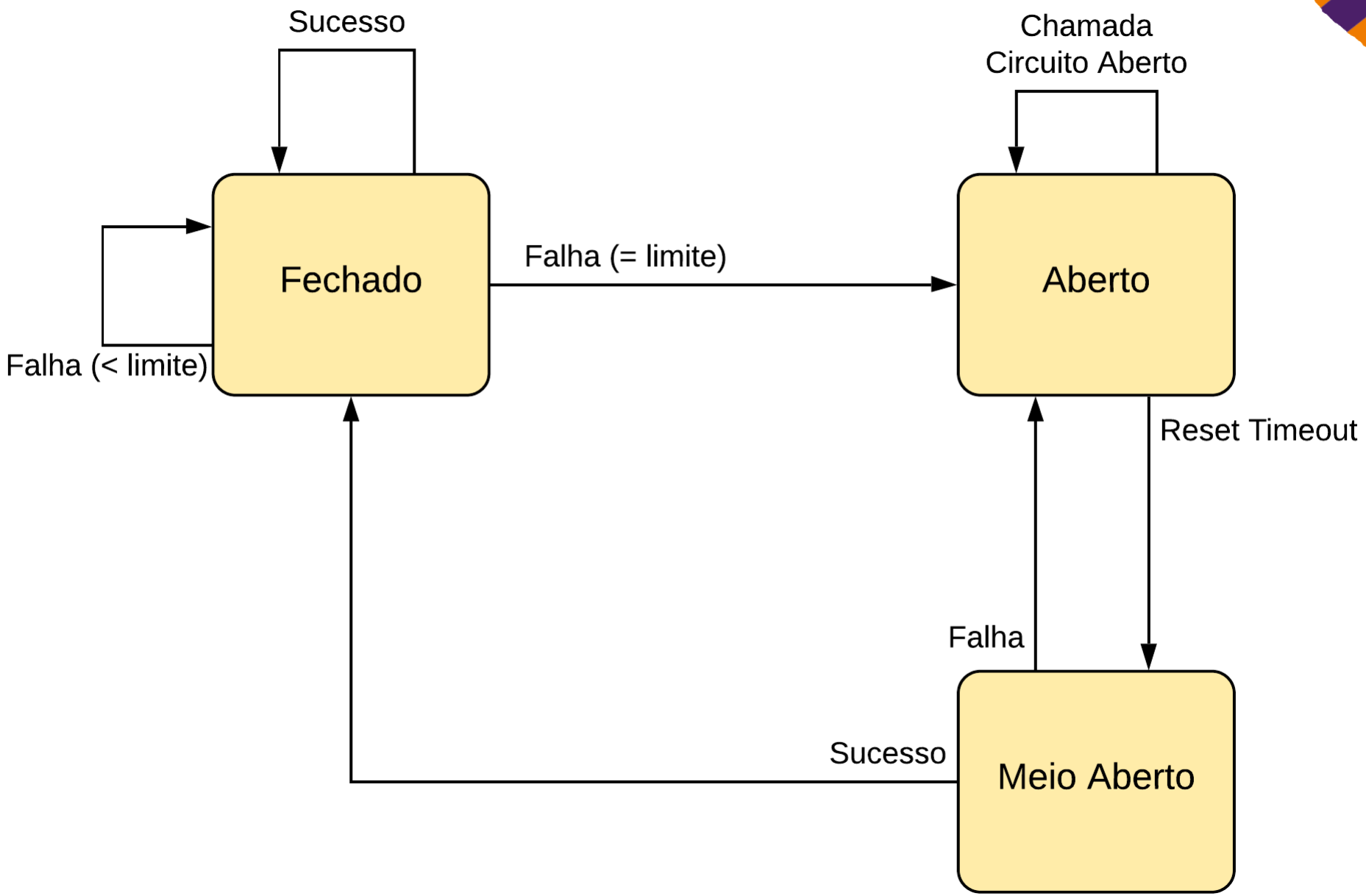
Definição

"O Circuit Breaker atua como um proxy em operações que podem falhar. Esse proxy monitora o número de falhas ocorridas em uma chamada e deve decidir se permite que a operação continue ou retorne uma exceção imediatamente."
(Cruz, Rafael)

Circuit Breaker Pattern

Vantagens

- Falhar rápido
- Retorno de erro customizado
- Não derrubar o ambiente


Sucesso
Chamada
Circuito Aberto
Fechado
Falha (= limite)
Aberto
Falha (< limite)
Reset Timeout
Falha
Sucesso
Meio Aberto

Azure Functions

Azure Functions

Definição

- Serverless
- Dimensionamento escalonável
- Múltiplas linguagens de programação
  - .NET, Javascript, Java, Python

Azure Functions

Cobrança

- Quantidade de execuções
- Tempo de execução
- Consumo de memória

Azure Functions

Gatilhos

- Blob Trigger
- Cosmos DB Trigger
- Http Trigger
- **ServiceBus Trigger**
- **Timer Trigger**

Estudo de caso

Circuit Breaker Pattern

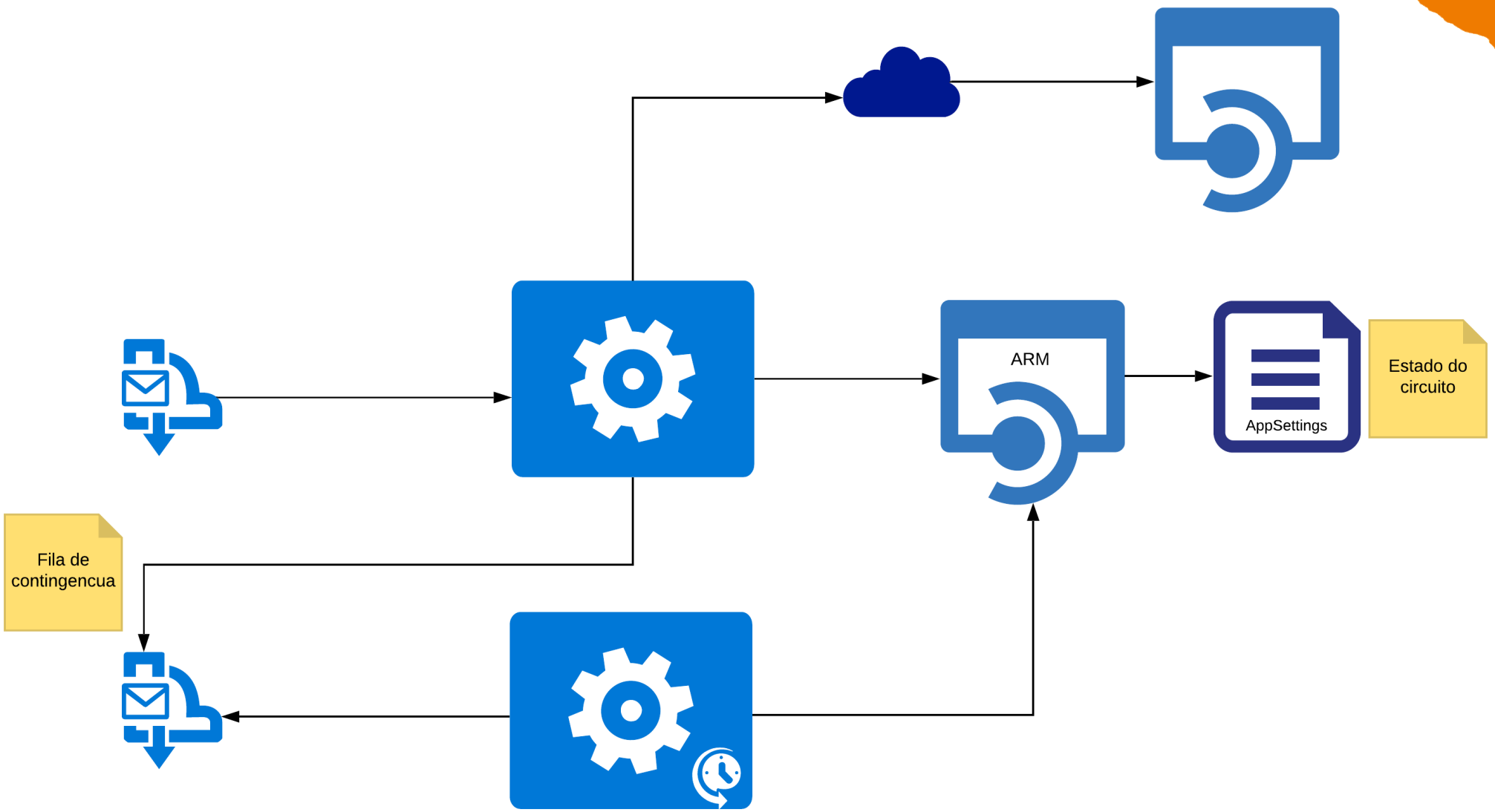
Estudo de caso

- Integração com sistema terceiro
- Qtde. Mensagens: 200.000 / dia
- Indisponibilidade: 45 minutos / mês
- Timeout padrão: 30 s

Circuit Breaker Pattern

Estudo de caso

- Tentativas de entrega: 10 (padrão)
- Memória: 512 MB
- Qtde. mens. afetadas: 6250 / mês
- Custo de execução: R$ 109,53 / mês
  - Tam. Memória x Tempo Exec. x Qtde. Exec.


ARM
AppSettings
Estado do circuito
Fila de contingencua

```
public static class EnviarDadosFunction
{
    [Disable("CircuitBreaker:Open")]
    [FunctionName("ProcessarFaturaFunction")]
    public static async Task Run(
        [ServiceBusTrigger(Constantes.NOME_FILA, Connection = Constantes.CONNECTION)]string message)
    {
         await Call(
            async () => { await new Api().EnviarDados(message); }, message
        );
    }

    private static async Task Call<T>(Func<Task> action, T message)
    {
        var timeoutPolicy = Policy
            .Handle<TimeoutException>()
            .RetryAsync(Constantes.QUANTIDADE_RETENTATIVAS);

        var fallbackPolicy = Policy
            .Handle<TimeoutException>()
            .FallbackAsync(async f =>
            {
                await new FilaContingencia().EnviarMensagem(message);
                await new GerenciadorCircuito().AbrirCircuito();
            });

        var circuitBreakerPolicy = fallbackPolicy.WrapAsync(timeoutPolicy);

        await circuitBreakerPolicy.ExecuteAsync(action);
    }
}
```

```
{
  "generatedBy": "Microsoft.NET.Sdk.Functions-1.0.29",
  "configurationSource": "attributes",
  "bindings": [
    {
      "type": "serviceBusTrigger",
      "connection": "connection-string-service-bus",
      "queueName": "nome-fila",
      "name": "message"
    }
  ],
  "disabled": "CircuitBreaker:Open",
  "scriptFile": "../bin/CircuitBreakerAzureFunctions.dll",
  "entryPoint": "CircuitBreakerAzureFunctions.EnviarDadosFunction.Run"
}
```

```
public static class MonitoramentoCircuitoFunction
{
    [FunctionName("MonitoramentoCircuitoFunction")]
    public static async Task Run([TimerTrigger("0 */10 * * * *")]TimerInfo myTimer)
    {
        string message;
        FilaContingencia filaContingencia = new FilaContingencia();
        GerenciadorCircuito gerenciadorCircuito = new GerenciadorCircuito();

        if(gerenciadorCircuito.EstaAberto)
        {
            while((message = await filaContingencia.LerMensagem<string>()) != null)
            {
                try
                {
                    await new Api().EnviarDados(message);

                    await filaContingencia.ProcessarMensagem();
                }
                catch(TimeoutException ex)
                {
                    await filaContingencia.RejeitarMensagem();
                    return;
                }
            }

            await gerenciadorCircuito.FecharCircuito();
        }
    }
}
```

"Breakers on their own are valuable, but clients using them need to react to breaker failures."

(Fowler, Martin)

Referências

- https://martinfowler.com/bliki/CircuitBreaker.html
- https://itnext.io/understand-circuitbreaker-design-pattern-with-simple-practical-example-92a752615b42
- https://itnext.io/understand-circuitbreaker-design-pattern-with-simple-practical-example-92a752615b42
- https://rafaelcruz.azurewebsites.net/2018/05/22/implementando-o-circuit-breaker-pattern-parte-1/
- https://rafaelcruz.azurewebsites.net/2018/06/11/implementando-o-circuit-breaker-pattern-parte-2/

MUITO OBRIGADO!

Givaldo Moreira

www.dtidigital.com.br