Transient fault handling and retry logic

Last updated by | Lisa Liu | Nov 6, 2020 at 10:34 AM PST

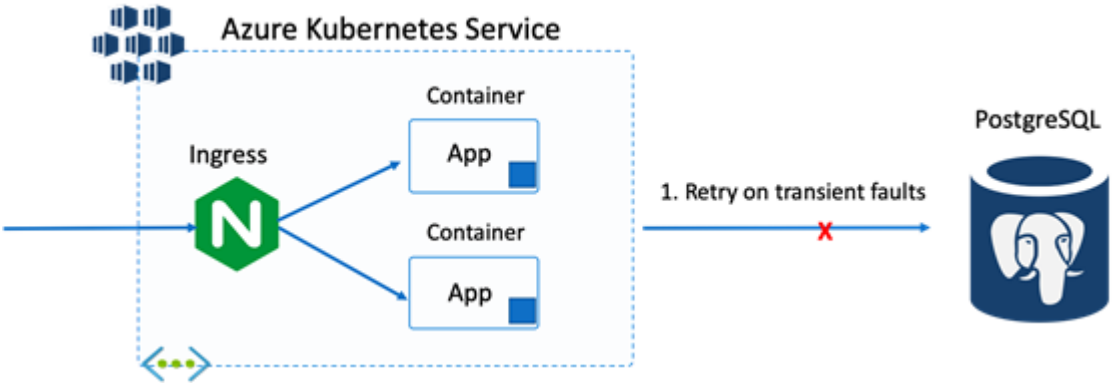
Problem

You should expect more transient faults in the cloud than on-premises due to the commodity hardware and shared environment. No retry or too many retries can cause different problems. If you don't retry on the transient fault, you lose that transaction. Too many retries would damage the database by hammering it. You may encounter transient faults in these scenarios.

- An error occurs when you try to open a connection.
- An idle connection is dropped on the server side. When you try to issue a command, it can't be executed.
- An active connection that currently is executing a command is dropped.

Solution

Implement proper retry strategy. It has to be the right number of retry count and interval between each attempt. Use short retry for interactive usecase where users are waiting for the operation to be completed, longer retry for non-interactive usecase to increase the chance of recovery.



This article shows best practice of implementing the retry strategy.

https://docs.microsoft.com/en-us/azure/architecture/best-practices/transient-faults

Example

This is the code snippet to implement retry using .NET core entity framework.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
   => optionsBuilder.UseNpgsql(
      "<connection_string>",
      options => options.EnableRetryOnFailure(maxRetryCount, maxRetryDelay, errorNumbersToAdd));
```

How good have you found this content?