# Fault chaos strategy **CHEAT SHEET**

## Basics

This proactive chaos **strategy injects exception(s)** to simulate unexpected failure.

You can configure the behaviour of the strategy via the **ChaosFaultStrategyOptions** object.

## Specify single exception - short form

```
new ResiliencePipelineBuilder()
  .AddChaosFault(0.1,
    () => new HttpRequestException(
      HttpRequestError.ConnectionError))
```

## Specify single exception – long form

```
new ResiliencePipelineBuilder<int>()
  .AddChaosFault(new ChaosFaultStrategyOptions
  {
    InjectionRate = 0.1,
    FaultGenerator = static _ =>
      ValueTask.FromResult<Exception?>(
        new HttpRequestException(
          HttpRequestError.ConnectionError))
  })
```

## Specify multiple exceptions with switch expression

```
new ResiliencePipelineBuilder()
  .AddChaosFault(new ChaosFaultStrategyOptions
  {
    FaultGenerator = static _ =>
    {
      var rnd = Random.Shared.NextDouble();
      Exception? ex = rnd switch
      {
        < 0.4 => new HttpRequestException(),
        >= 0.4 => new SocketException(),
        _ => null
      };
      return new ValueTask<Exception?>(ex);
    }
  }
```

## Specify multiple exceptions with FaultGenerator

```
new ResiliencePipelineBuilder()
  .AddChaosFault(new ChaosFaultStrategyOptions
  {
    FaultGenerator = new FaultGenerator()
      .AddException<HttpRequestException>(weight: 40)
      .AddException<SocketException>(weight: 60)
  })
```

## Specify asynchronous delegate for injection notification

```
new ResiliencePipelineBuilder()
  .AddChaosFault(new ChaosFaultStrategyOptions
  {
    OnFaultInjected = static async args => await NotifyAsync(args.Fault)
  })
```