Tópicos Especiais em Sistemas de Informação

Aula 06

Sistemas Tolerantes a Falhas e Alta Disponibilidade

Prof. Msc. Francisco de Assis de Souza Rodrigues

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Objetivo

- Compreender os conceitos de tolerância a falhas e alta disponibilidade.
- Explorar como o Elixir e o Erlang/OTP oferecem suporte nativo a esses conceitos.
- Comparar abordagens em outras linguagens, como Go, Rust e Java (Spring Boot).
- Desenvolver um sistema simples e tolerante a falhas com supervisores no Elixir.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade

📌 O que são sistemas tolerantes a falhas?

Sistemas tolerantes a falhas são **projetados** para **continuar funcionando** mesmo quando **ocorrem falhas inesperadas** em componentes individuais.

Isso é essencial para aplicações críticas onde a interrupção do serviço pode causar grandes prejuízos financeiros ou operacionais.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade

Principais características:

✔️ **Isolamento de falhas:** Um erro em uma parte do sistema não deve afetar o restante.

✔️ **Redundância:** Sistemas duplicados atuam como backup para manter a operação.

✔️ **Detecção e recuperação automática:** Monitoramento contínuo para detectar falhas e restaurar automaticamente os serviços afetados.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade

Exemplos de falhas comuns em sistemas distribuídos:

- Falha de hardware (queda de servidores, discos corrompidos).
- Erros de software (bugs, memory leaks, falhas de lógica).
- Problemas de rede (desconexão, latência elevada, particionamento de rede).
- Sobrecarga de tráfego (quando um sistema recebe mais requisições do que pode processar).

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade

Estratégias para mitigar falhas:

✅ Replicação de serviços: Criar cópias do mesmo serviço rodando em diferentes servidores.

✅ Monitoramento e alertas: Uso de ferramentas como Prometheus, Grafana e Sentry para detectar falhas rapidamente.

✅ Fallback e Circuit Breaker: Implementação de mecanismos que evitam a propagação de falhas (exemplo: Resilience4J no Java).

✅ Retry automático: Tentar novamente requisições que falharam, dentro de limites razoáveis.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade

Alta disponibilidade refere-se à capacidade de um sistema permanecer operacional e acessível durante longos períodos, minimizando tempo de inatividade e garantindo um **uptime próximo de 100%**.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade

Principais componentes de um sistema altamente disponível:

1️⃣ Load Balancing (Balanceamento de carga)

- Distribui requisições entre múltiplos servidores para evitar sobrecarga.
- Exemplo: Nginx, HAProxy, AWS Elastic Load Balancer (ELB).

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade

2️⃣ Failover automático

- Se um servidor falha, outro assume sua função sem interrupção perceptível para os usuários.

- Exemplo: Bancos de dados replicados (PostgreSQL com Patroni, MySQL Cluster).

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade

3️⃣ Escalabilidade automática

- Aumentar ou reduzir recursos conforme a demanda, evitando desperdício de capacidade.

- Exemplo: Kubernetes Horizontal Pod Autoscaler (HPA).

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade

Métricas para avaliar HA:

📊 **MTBF (Mean Time Between Failures):** Tempo médio entre falhas.

📊 **MTTR (Mean Time to Repair):** Tempo médio para recuperação após uma falha.

📊 **SLA (Service Level Agreement):** Contratos que garantem um tempo mínimo de disponibilidade (exemplo: 99,99% uptime).

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade



WhatsApp

- Escrito em **Erlang/Elixir**, projetado para lidar com milhões de conexões simultâneas.
- Alta resiliência devido ao uso da **Erlang VM (BEAM)**, que permite processos isolados e supervisores para recuperação automática.



Sistemas Bancários

- Bancos precisam garantir transações financeiras seguras e contínuas.
- Uso de **replicação de banco de dados** e **arquitetura distribuída** para evitar falhas únicas (single points of failure - SPOF).

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Introdução à Tolerância a Falhas e Alta Disponibilidade

📌 Infraestrutura de Telecomunicações

- Operadoras como AT&T, Verizon e Vivo dependem de redes altamente disponíveis.
- Redes utilizam **sistemas distribuídos e roteamento inteligente** para evitar perda de conectividade.

📌 Serviços Críticos (Hospitais, Transporte Aéreo, Data Centers)

- Sistemas de monitoramento de pacientes precisam operar **24/7** sem falhas.
- Controle de tráfego aéreo exige **redundância tripla** para garantir segurança máxima.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Como o Elixir e o Erlang/OTP resolvem esses desafios?

📌 **Erlang VM (BEAM)**: projetada para sistemas distribuídos e tolerantes a falhas.

📌 **Supervisores**: processos que reiniciam automaticamente componentes falhos.

📌 **Actor Model**: concorrência e isolamento entre processos.

📌 **Hot Code Swapping**: atualizações sem downtime.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Como o Elixir e o Erlang/OTP resolvem esses desafios?

⚙️ Erlang VM (BEAM) — A base do Elixir

O Elixir roda sobre a **Erlang VM**, conhecida como **BEAM**, que foi criada originalmente para sistemas de telecomunicação — onde a **confiabilidade, concorrência e tolerância a falhas** são fundamentais.

- A BEAM é diferente de outras VMs (como a JVM) porque:
  - Foi feita para **sistemas distribuídos**, **concorrentes** e que **não podem parar**.
  - É altamente otimizada para lidar com **milhões de processos leves** ao mesmo tempo.
  - Cada processo é **isolado**: se um falha, não afeta os outros.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Como o Elixir e o Erlang/OTP resolvem esses desafios?

👥 Actor Model — Cada parte isolada e concorrente

Elixir e Erlang usam o **modelo de ator (actor model)** de concorrência.

➡️ O que isso significa?

- Cada componente do sistema é um **processo leve** (não é thread do SO!).
- Processos **não compartilham memória** e **comunicam-se por mensagens**.
- Isso garante isolamento: se um processo falha, ele não arrasta os outros com ele.

💡 Exemplo:
Imagine um sistema que gerencia pedidos. Cada pedido pode ser tratado por um processo separado. Se um pedido dá erro, os outros continuam sendo processados normalmente.

UniFAP

Como o Elixir e o Erlang/OTP resolvem esses desafios?

🧠 Supervisores — Recuperação automática de falhas

A cereja do bolo da tolerância a falhas no Elixir: o **Supervisor**.

Um supervisor é um processo especial que **monitora outros processos**.
Se algum processo monitorado falhar, o supervisor pode:

- **Reiniciar apenas o processo com falha.**
- **Reiniciar todos os filhos.**
- **Reiniciar uma parte da árvore.**

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Como o Elixir e o Erlang/OTP resolvem esses desafios?

🔁 Isso cria o que chamamos de **árvore de supervisão**.

Essas árvores são **autorrecuperáveis**: o sistema reinicia sozinho as partes com problema, como um corpo se curando de um ferimento.

📌 **Estratégias de supervisão**:

- :one_for_one: reinicia apenas o processo que falhou.
- :one_for_all: reinicia todos os filhos se um falhar.
- :rest_for_one: reinicia o processo que falhou e todos os que vieram depois dele.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Como o Elixir e o Erlang/OTP resolvem esses desafios?

🛠️ Demonstração prática: Criando um sistema simples com supervisores.

```
defmodule Worker do
  use GenServer

  # Inicializa o processo
  def start_link(_) do
    GenServer.start_link(__MODULE__, %{}, name: __MODULE__)
  end

  def init(state), do: {:ok, state}

  def handle_cast(:fail, _state) do
    raise "Simulating failure!"
  end
end
```

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Como o Elixir e o Erlang/OTP resolvem esses desafios?

🛠️ Demonstração prática: Criando um sistema simples com supervisores.

```
defmodule SupervisorExample do
  use Supervisor

  def start_link(_) do
    Supervisor.start_link(__MODULE__, [], name: __MODULE__)
  end

  def init(_) do
    children = [
      {Worker, []}
    ]

    Supervisor.init(children, strategy: :one_for_one)
  end
end
```

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Como o Elixir e o Erlang/OTP resolvem esses desafios?
🛠️ Demonstração prática: Criando um sistema simples com supervisores.

```
# Iniciando o sistema
{:ok, _} = SupervisorExample.start_link([])
GenServer.cast(Worker, :fail)  # Simula falha e observa a reinicialização automática
```

Neste exemplo:

- Criamos um **processo que pode falhar**.
- Um **supervisor monitora esse processo** e, se ele quebrar, o reinicia automaticamente.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Como o Elixir e o Erlang/OTP resolvem esses desafios?

🔄 Hot Code Swapping — Atualização sem downtime

Outra vantagem incrível herdada do Erlang: **trocar o código de um módulo enquanto o sistema está rodando**.

Sim, você pode atualizar o sistema **sem precisar parar o servidor**.

🧠 Ideal para sistemas que **precisam de alta disponibilidade contínua**, como sistemas bancários, serviços financeiros ou infraestrutura de telecom.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Como o Elixir e o Erlang/OTP resolvem esses desafios?

Recurso	O que faz	Benefício
BEAM VM	Roda milhões de processos simultâneos	Alta concorrência e resiliência
Actor Model	Processos isolados que se comunicam por mensagens	Segurança e isolamento de falhas
Supervisores	Reiniciam automaticamente processos com erro	Recuperação rápida e automática
Árvores de supervisão	Organização hierárquica de processos	Robustez e escalabilidade
Hot Code Swapping	Atualiza o sistema sem parar	Alta disponibilidade contínua

Como o Elixir e o Erlang/OTP resolvem esses desafios?

Elixir, graças à base sólida do Erlang, não apenas tolera falhas — ele espera que falhas aconteçam e lida com elas com elegância. Isso faz dele uma das melhores opções para quem quer construir sistemas confiáveis, escaláveis e altamente disponíveis.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Alternativas em Outras Linguagens

Apesar do Elixir (e Erlang) oferecerem suporte nativo e robusto à tolerância a falhas, outras linguagens populares também oferecem estratégias eficazes — embora geralmente dependam de bibliotecas externas, boas práticas arquiteturais e infraestrutura de apoio como containers e orquestradores.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Alternativas em Outras Linguagens
Go (Golang)

Concorrência leve com goroutines Go é famoso por sua abordagem simples e eficiente à concorrência, utilizando goroutines (threads leves gerenciadas pela runtime do Go) e channels para comunicação entre elas.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Alternativas em Outras Linguagens
Go (Golang)

Tratamento de falhas e resiliência Go não tem supervisores nativos como o Elixir. A tolerância a falhas é muitas vezes implementada de forma mais manual ou utilizando a infraestrutura ao redor da aplicação, como:

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Alternativas em Outras Linguagens
Go (Golang)

✅ **Retries com backoff:**
Uso de bibliotecas como go-retryablehttp ou implementação própria com time.Sleep para tentativas automáticas em caso de falha.

✅ **Padrões de resiliência:**
Implementação de *timeouts*, *circuit breakers*, e *fallbacks* com bibliotecas como goresilience.

✅ **Kubernetes e HA:**
A tolerância a falhas em produção geralmente depende de **containers orquestrados por Kubernetes**, que reiniciam aplicações quando falham, gerenciam **health checks** e realizam **failover automático**.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Resumo (Go):

- ✅ Concorrência eficiente com goroutines.
- ❌ Sem supervisão nativa.
- ✅ Resiliência via boas práticas e ferramentas externas (K8s, retries).

Foco em segurança e performance

Rust se destaca pela **segurança de memória** e ausência de *data races*, o que já elimina um conjunto inteiro de falhas comuns.

Concorrência com Actix / Tokio

Para sistemas tolerantes a falhas e altamente concorrentes, o Rust oferece frameworks como:

- **Actix:** Inspirado no modelo de atores (como Erlang/Elixir e Akka).
- **Tokio:** Runtime assíncrona para IO de alto desempenho.

Gerenciamento de falhas

Rust encoraja o uso explícito de tratamento de erros com os tipos Result<T, E> e Option<T>. Não há exceções, então tudo é feito de forma declarativa e segura.

Infraestrutura externa para HA:

Assim como Go, a alta disponibilidade geralmente é alcançada com apoio de ferramentas como Docker, Kubernetes, e sistemas de mensageria (NATS, Kafka).

Resumo (Rust):

- ✅ Concorrência segura e robusta.
- ✅ Modelo de atores disponível com Actix.
- ❌ HA depende da arquitetura externa e runtime.

Alternativas em Outras Linguagens
Java (Spring Boot, Akka)

⚙️ Spring Boot + Resilience4J

Java tradicionalmente é mais pesado, mas tem um ecossistema rico para lidar com falhas em aplicações distribuídas.

Resilience4J: biblioteca moderna (substituta do Hystrix) com suporte para:

- **Circuit Breaker**: evita chamadas para serviços instáveis.
- **Retry**: tenta novamente chamadas que falharam.
- **Bulkhead**: isola falhas para não afetar o sistema todo.
- **Rate Limiter**: limita o número de chamadas para proteger o sistema.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Alternativas em Outras Linguagens

🛠️ Exemplo de anotação no Spring Boot: java

```java
@Retry(name = "serviceA", fallbackMethod = "fallback")
public String callServiceA() {
    return restTemplate.getForObject("http://servico-a", String.class);
}
```

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Alternativas em Outras Linguagens

🎭 **Akka (com Scala ou Java)**

Modelo de Atores semelhante ao Elixir/Erlang

- Cada ator é um processo isolado que se comunica por mensagens.
- Suporte a **supervisão** e **estratégias de reinício** automáticas.
- Usado para sistemas de alto desempenho, como jogos multiplayer, bancos e IoT.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Alternativas em Outras Linguagens

Akka Cluster permite tolerância a falhas em sistemas distribuídos — com replicação, sharding e failover automático.

Resumo (Java/Scala):

- ✅ Ecossistema rico em padrões de resiliência (Resilience4J).
- ✅ Akka traz tolerância a falhas nativa ao estilo Erlang.
- ❌ Java puro (sem Akka) depende bastante de infraestrutura.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Alternativas em Outras Linguagens

Linguagem	Modelo Concorrente	Tolerância a Falhas Nativa	Requer K8s/Infra?	Padrão Usado
Elixir	Actor Model (BEAM)	Sim (Supervisores, VM)	Não obrigatório	OTP
Go	Goroutines + Channels	Parcial (infra + código)	Sim	Retry, timeout, circuit breaker
Rust	Actix, Tokio	Parcial (isolamento forte)	Sim	Result/Error handling
Java (Spring)	Threads, anotação (AOP)	Parcial (Resilience4J)	Sim	Retry, CB, fallback
Akka (Scala/Java)	Actor Model	Sim (supervisão real)	Opcional	Akka supervision tree

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

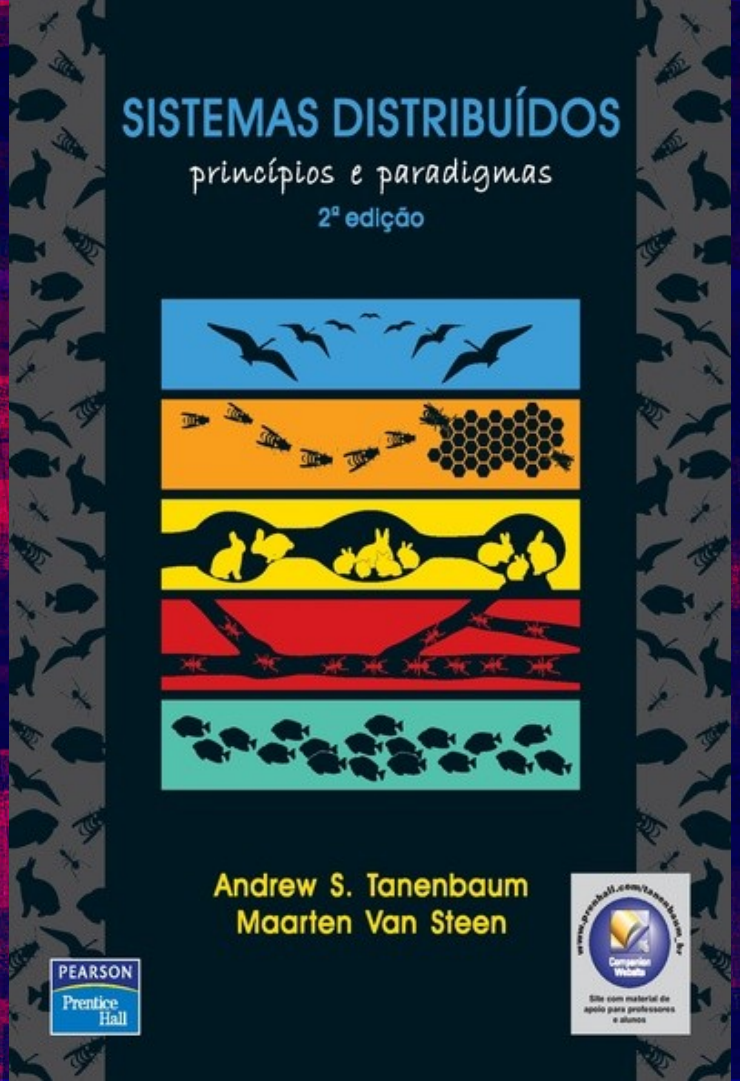
Discussão

- Qual abordagem seria melhor para diferentes cenários?
- Quando usar Elixir em vez de Go ou Java?
- Demonstração prática: Deploy de um sistema tolerante a falhas com supervisores no Elixir.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Bibliografia

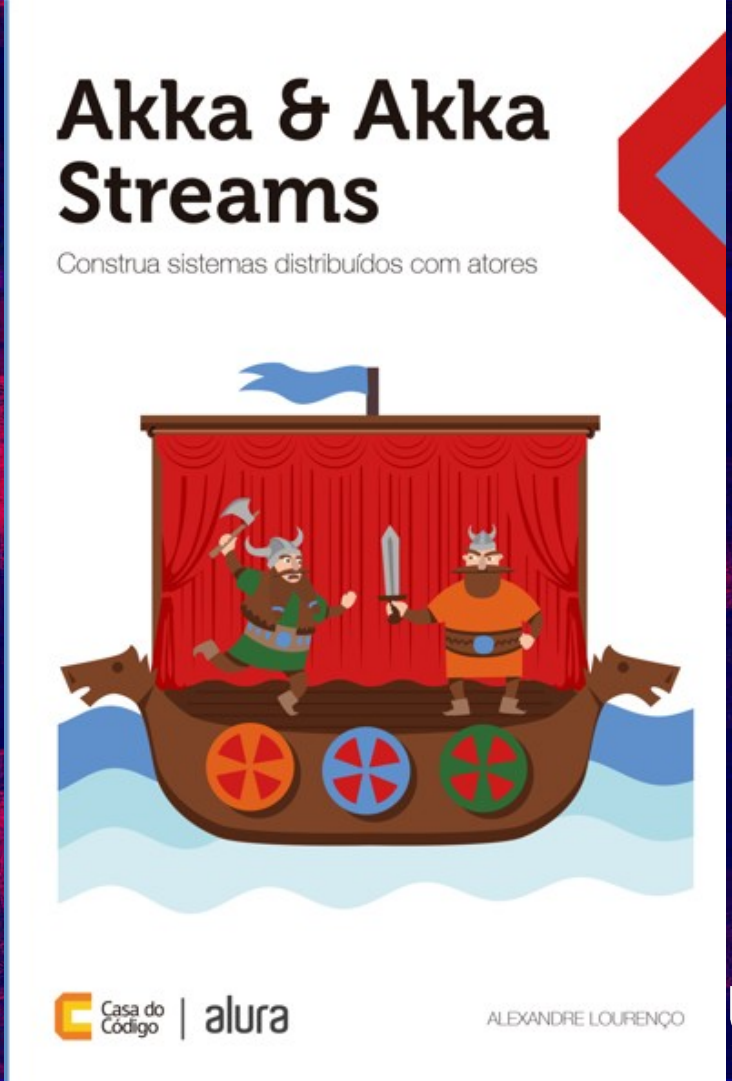
TANENBAUM, Andrew Stuart; STEEN, Marteen Van. Sistemas distribuídos: princípios e paradigmas. 2. ed. São Paulo, SP: Pearson, 2007. E-book. Disponível em: https://plataforma.bvirtual.com.br/Acervo/Publicacao/411 . Acesso em: 04 abr. 2025.



UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Bibliografia

LOURENÇO, Alexandre Eleutério Santos. Akka e Akka Streams: construa sistemas distribuídos com atores. São Paulo, SP: Casa do Código, 2021. E-book. Disponível em: https://plataforma.bvirtual.com.br/Acervo/Publicacao/212576 . Acesso em: 04 abr. 2025.

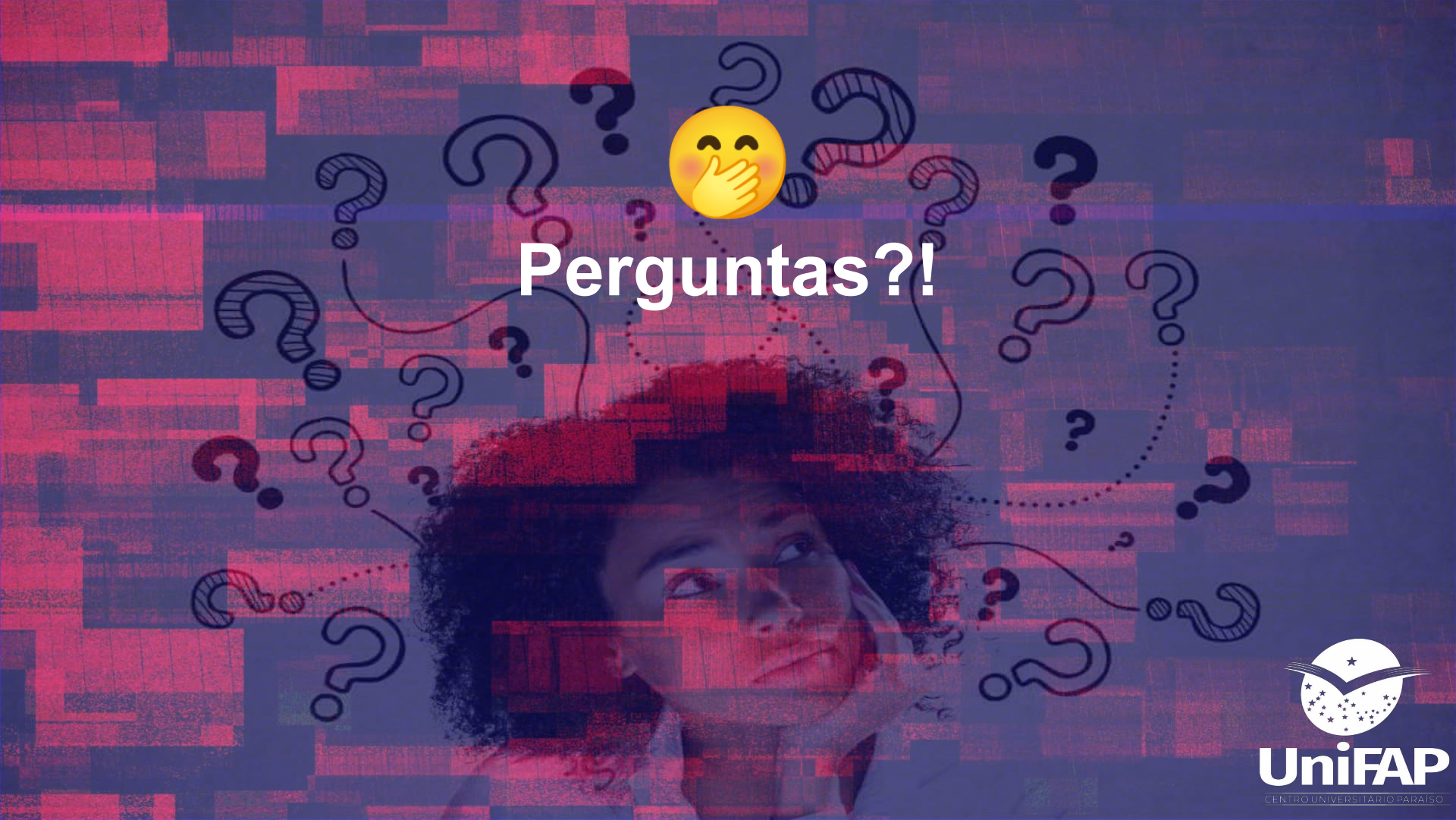


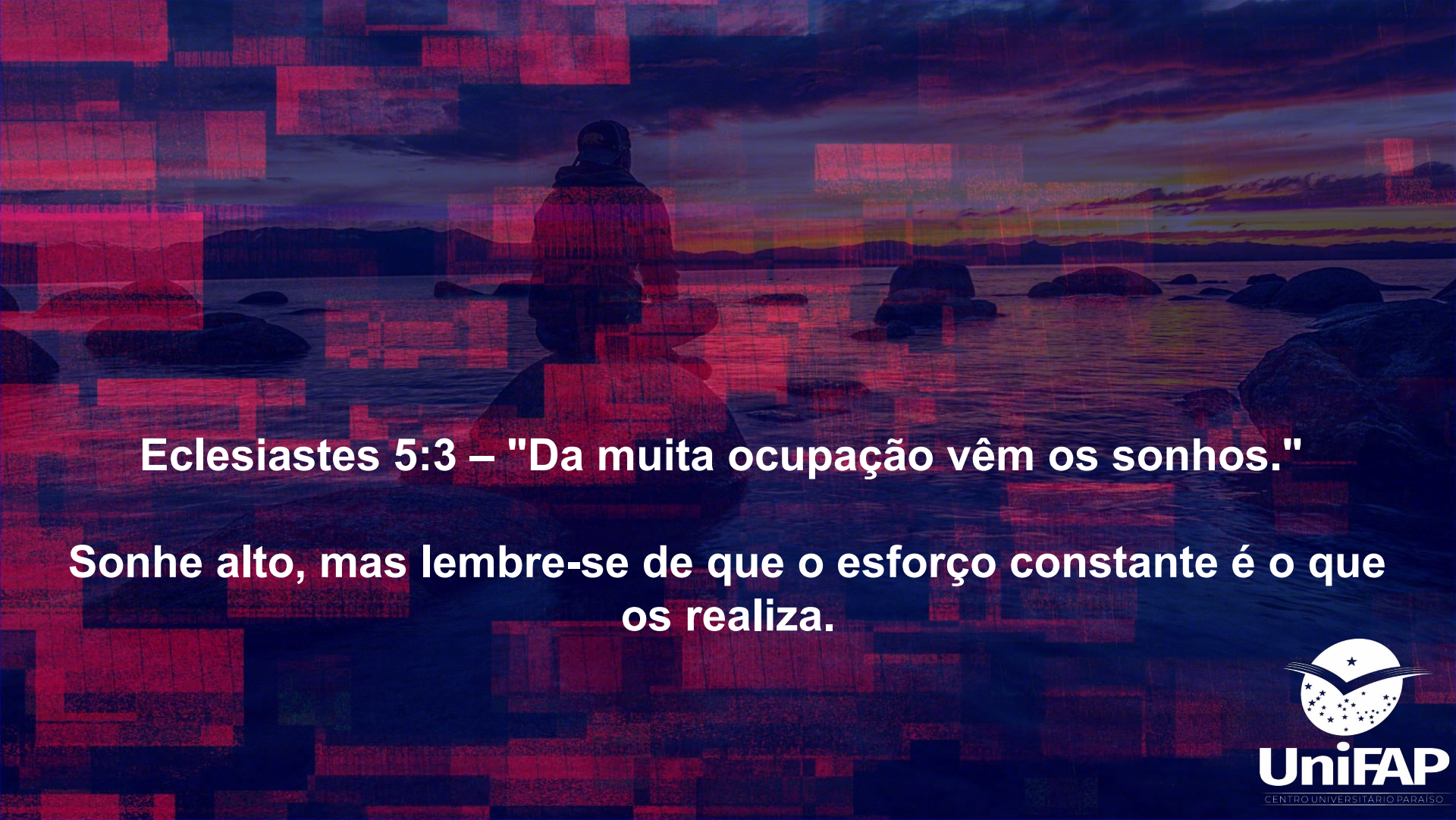
UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Referências

CHATGPT. **Fundamentos de Inteligência Artificial**. OpenAI, 2025. Disponível em: https://chat.openai.com. Acesso em: *20/03/2025*.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO


Perguntas?!


Eclesiastes 5:3 – "Da muita ocupação vêm os sonhos."
Sonhe alto, mas lembre-se de que o esforço constante é o que os realiza.
UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO



Obrigado!

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO