Entendendo Patroni

Uma solução de alta disponibilidade para Postgres

Israel Barth Rubio,
Senior Staff SDE
05/09/2025

A motivação

Por que precisamos falar sobre Alta Disponibilidade (HA)?

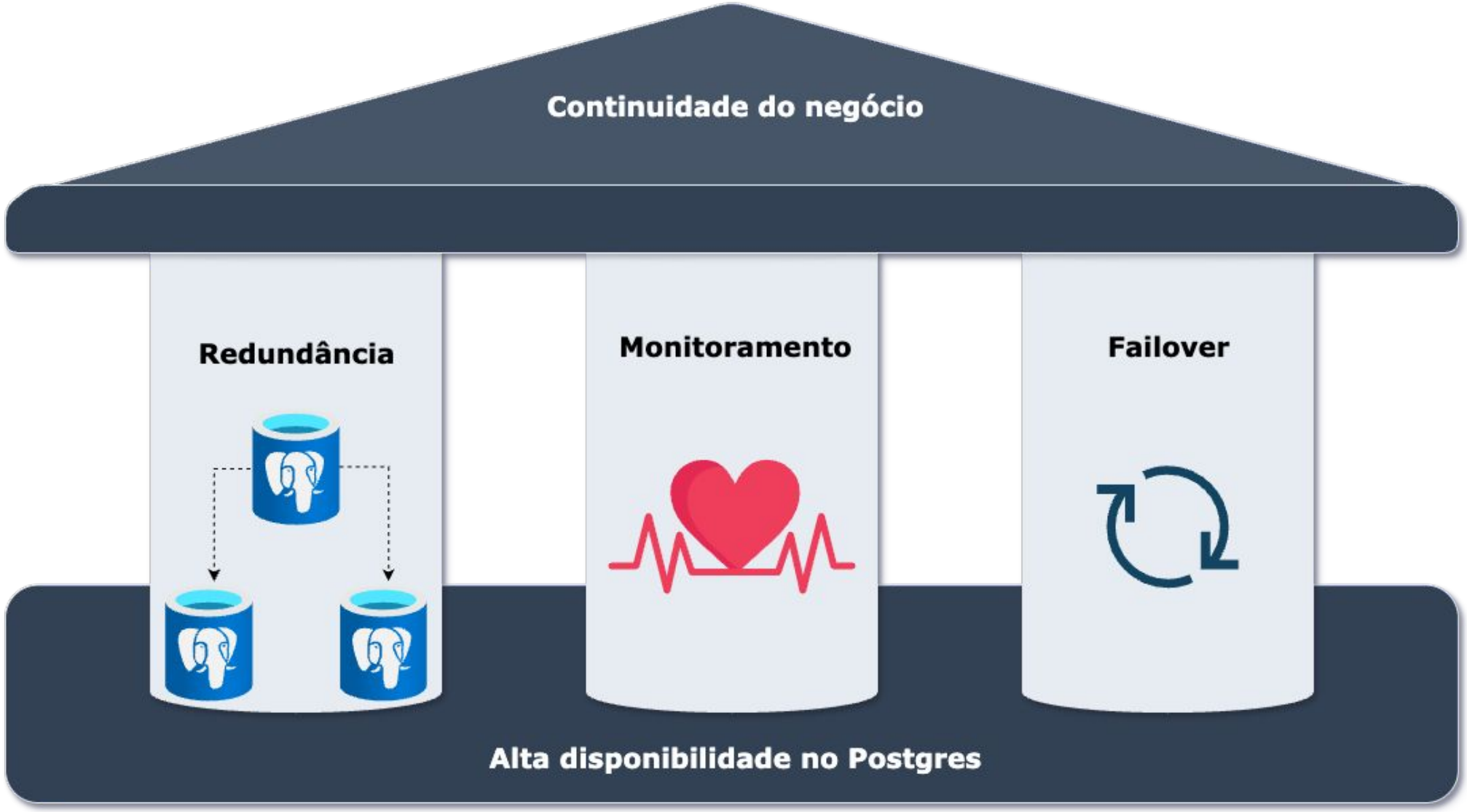
- O custo do serviço fora do ar (downtime)
  - Impacto no negócio
  - Perda de confiança do cliente
  - Possível perda de dados
- O que significa disponibilidade?
  - É a medida de quão acessível seu sistema está ao longo do tempo
  - Usamos o conceito dos "noves" para quantificar isso, por exemplo:
    - 99% ("dois noves"): Permite até 3,65 dias de downtime por ano
    - 99.9% ("três noves"): Permite até 8,76 horas de downtime por ano
    - …
- Como posso garantir que meu banco de dados sobreviva a uma falha sem intervenção manual imediata?

Fundamentos de HA no Postgres

Os pilares da alta disponibilidade

- Redundância:
  - Duplicar componentes para não ter um único ponto de falha (SPOF)
  - No Postgres: Ter cópias dos seus dados
    - Isso é feito através da replicação (streaming replication)
- Monitoramento (health check):
  - Saber quando algo está errado com o componente principal e qual componente redundante pode assumir
  - No Postgres: Verificar a saúde da instância primária (primary) e o status das réplicas (standbys)
- Failover:
  - Quando o componente principal falha, um componente redundante saudável assume
  - No Postgres: Promover um standby saudável a novo primary quando o original falha

Os pilares da alta disponibilidade



Physical streaming replication

- Como funciona?
  - O primary envia continuamente os registros do WAL (Write-Ahead Log) para os standbys
  - Os standbys aplicam esses registros para se manterem atualizados, avançando a LSN (Log Sequence Number)
  - A replicação é física: Se o primary escreveu conteúdo X na página Y do arquivo Z, o standby também fará isso

- Modos de replicação:
  - Assíncrona (padrão): O primary confirma a transação sem esperar pelos standbys. Maior performance, mas com risco de perda de dados em um failover
  - Síncrona: O primary espera a confirmação de pelo menos um standby antes de confirmar a transação. Sem perda de dados em um failover, mas com impacto na latência de escrita

Physical streaming replication

- Mais informações sobre streaming replication em https://www.postgresql.org/docs/current/warm-standby.html#STREAMING-REPLICATION

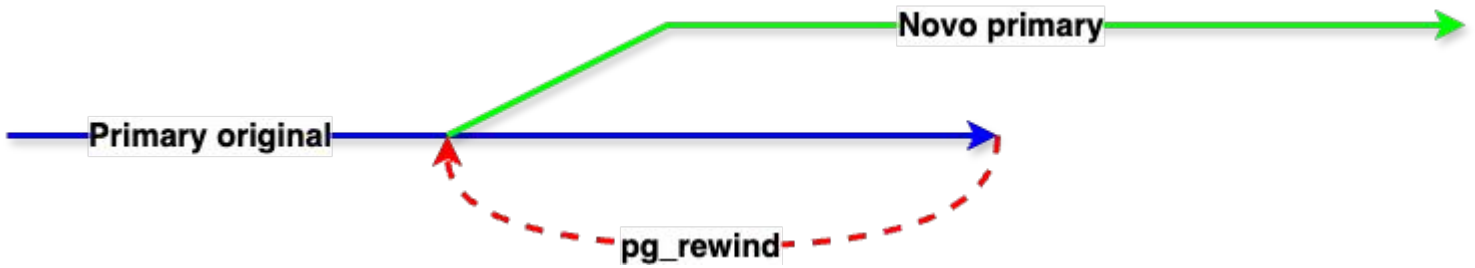
- O Postgres te fornece as ferramentas, mas ele não decide quando promover um standby:
  - Essa decisão cabe ao administrador

Failover manual

- Fluxo do processo:
  - Recebe um alerta do seu sistema de monitoramento: Seu primary está indisponível
  - Verifica o status dos standbys e qual está mais atualizado
  - Promove o standby escolhido:
    - `pg_ctl -D $PGDATA promote` ou `SELECT pg_promote();`
    - A promoção cria uma nova linha do tempo no cluster
  - Reconfigura os demais standbys para seguirem o novo primary
  - Reconfigura suas aplicações para usarem o novo primary

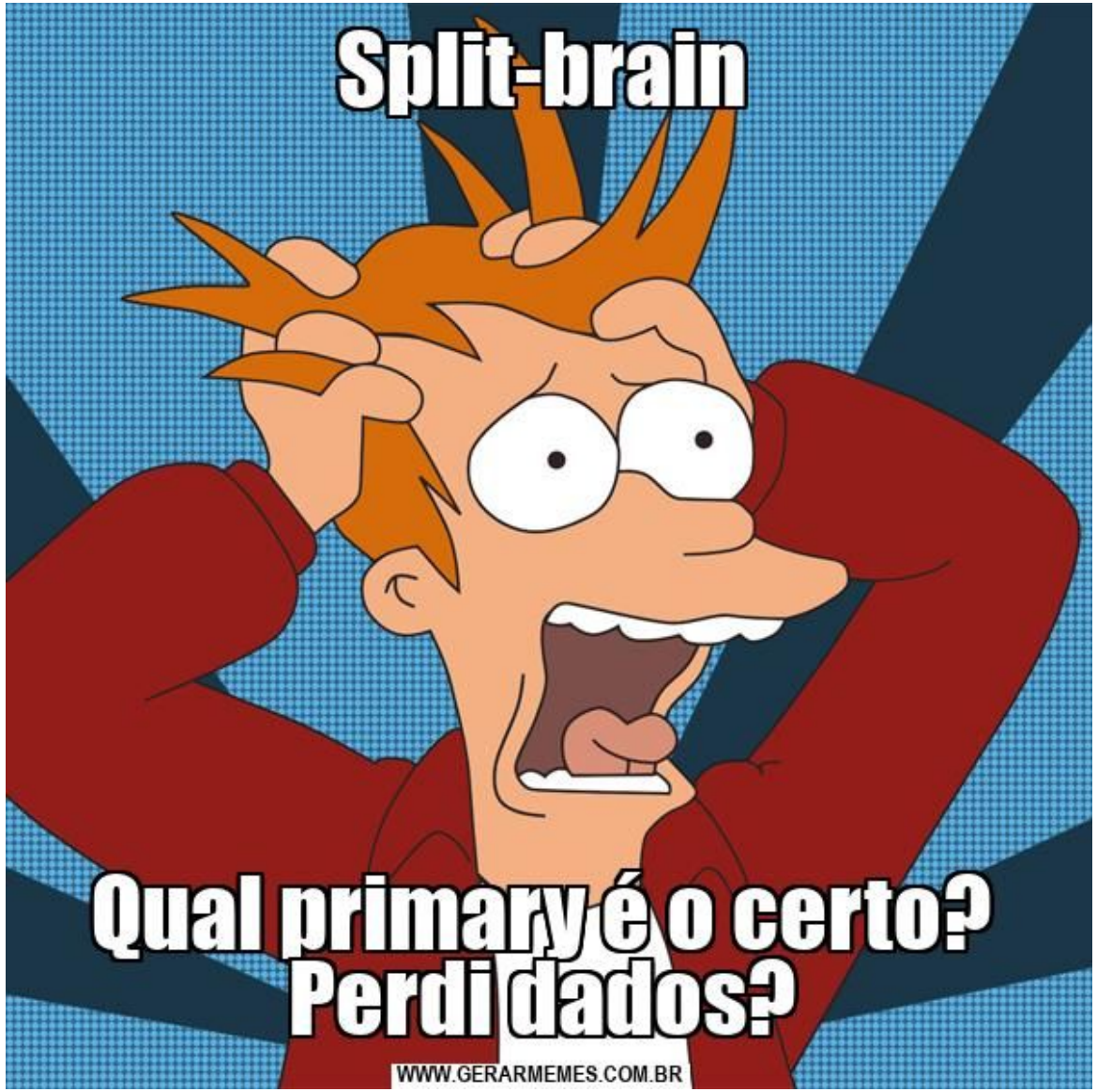
Failover manual

- Fluxo do processo:
  - Reingressa (rejoin) seu antigo primary como um standby do novo primary
    - Reconfigura: Caso o antigo e novo primary estivessem na mesma posição durante o failover; ou
    - Ressincroniza: Com **pg_rewind** , faz o antigo primary “voltar no tempo”, de forma que consiga replicar do novo primary; ou
    - Reconstrói: Com **pg_basebackup** , recriando o antigo primary “do zero” a partir do novo primary



Failover manual

- Desafios:
  - Processo lento
  - Propenso a erro humano
  - Risco de split-brain
    - 2 primaries simultâneos
    - Aplicações usando primaries diferentes



A solução automatizada: Patroni

O que é o Patroni

- Ferramenta do ecossistema Postgres
- Amplamente utilizado para gerenciamento e HA de clusters Postgres
- Open source
- Escrita em Python 3
- Links úteis:
  - Repositório de código: https://github.com/patroni/patroni
  - Documentação: https://patroni.readthedocs.io/en/latest/index.html

O papel do Patroni

- Automatiza todo o processo de failover descrito anteriormente
- Automatiza o processo de bootstrap
  - Por padrão:
    - **initdb** : Para inicializar o primary
    - **pg_basebackup** : Para inicializar os standbys
  - Customizável

O papel do Patroni

- Facilidades como REST API e **patronictl** para gerenciar o cluster
  - Alterar configuração das instâncias
  - Realizar switchover (trocar role entre um primary e um standby) imediato ou agendado
  - Reiniciar instâncias de forma imediata ou agendada
  - Listar membros e topologia do cluster
  - Pausar temporariamente o gerenciamento de failover automático
  - etc.

O papel do Patroni

- O Postgres é o avião com todos os seus instrumentos
  - O Patroni é o piloto automático que abstrai e automatiza o uso deles

https://eviation.com.br/wp-content/uploads/2023/11/painel-do-aviao.webp

https://reparadorsa.com.br/wp-content/uploads/2022/11/Header-Botao-de-piloto-automatico.jpg

Os componentes de um cluster Patroni

- Distributed Configuration Store (DCS)
  - Várias tecnologias de DCS são suportadas.
    - A mais utilizada é **etcd**
  - Fonte única de verdade para o cluster.
  - Onde é armazenado o estado e configuração global do cluster. Exemplos de chaves:
    - **initialize** : O systemd ID do cluster Postgres, gerado pelo **initdb**
    - **leader** : Nome do membro Patroni que é o líder atual do cluster, isto é, o primary do cluster Postgres
    - **members/NOME** : Informações do membro Patroni chamado **NOME**
    - **config** : Configuração global do cluster Patroni
- Postgres
  - Instâncias gerenciadas pelo Patroni

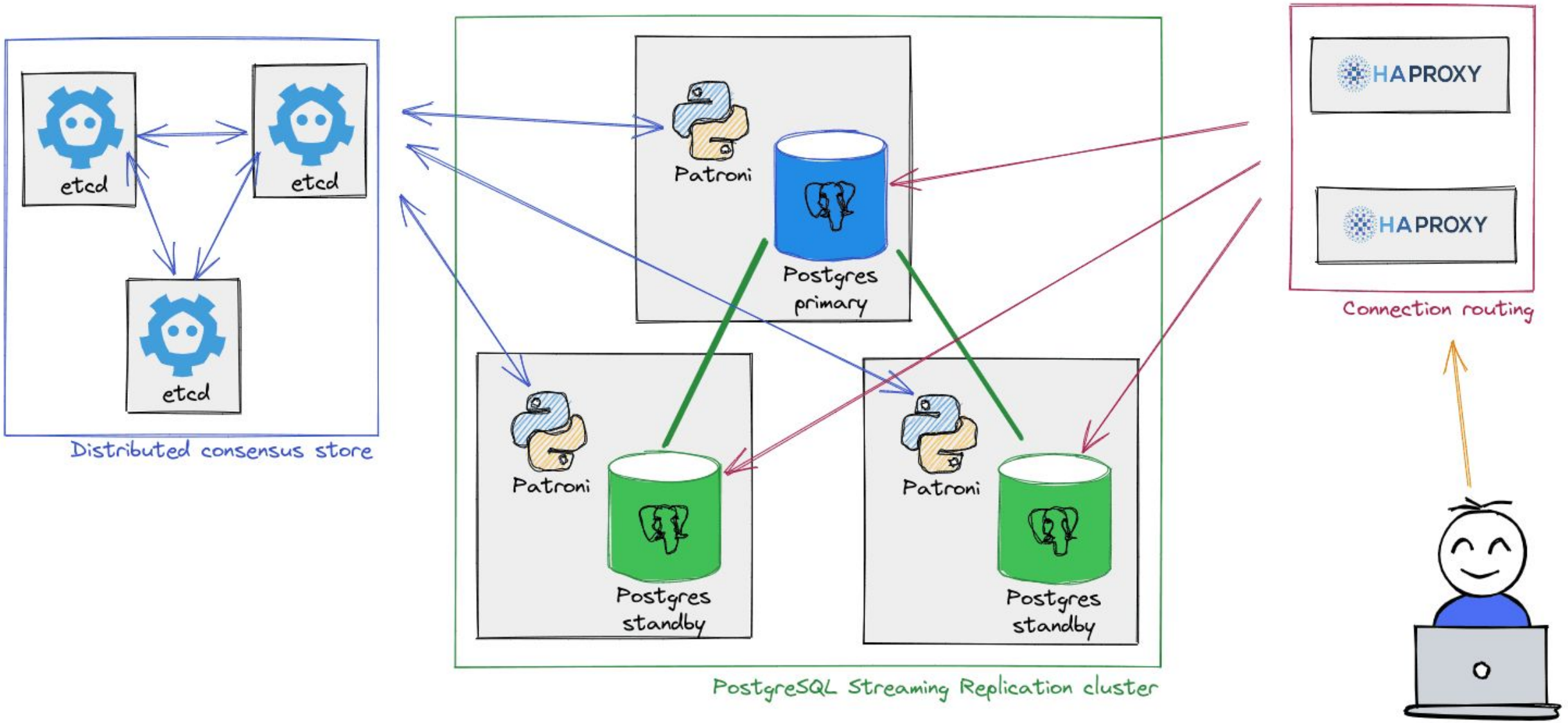
Os componentes de um cluster Patroni

- Patroni
  - Agente executado em cada máquina Postgres
  - Monitora a saúde da sua instância
  - Expõe uma REST API
  - Realiza operações automáticas ou operações requisitadas pelo administrador

Os componentes de um cluster Patroni

- Proxy
  - Componente opcional.
  - O mais utilizado é **HAProxy**
  - Realiza health checks contra a REST API do Patroni
  - Funcionalidades de balanceamento de carga e de roteamento de conexões. Exemplo de configuração:
    - Porta **5000** : Sempre conecta no primary atual, para executar consultas de leitura e escrita
    - Porta **5001** : Conecta cada vez em um standby diferente, para executar consultas somente leitura

Os componentes de um cluster Patroni



https://www.enterprisedb.com/docs/static/298a2bb5138b10db2c544b54eb6d9256/e1b7c/supported-arch.png

Failover automático

Atualização de status no DCS

- Cada membro do cluster Patroni é responsável por atualizar seu status no DCS
  - Os intervalos são configuráveis. Por padrão:
    - Faz os health checks e atualiza o status a cada 10 segundos
    - Se não for atualizado, o status expira depois de 30 segundos
- Todos os membros atualizam a sua chave **members/NAME**
  - Contém informações como:
    - Nome
    - LSN
    - Dados de conexão (connection string)
    - etc.
- O líder do cluster atualiza também a chave leader (**leader lock**)
  - Funciona como um lock que identifica quem deve ser o primary do cluster Postgres

O que acontece quando o líder falha?

- Ele para de atualizar a leader lock
  - Depois de 30 segundos (por padrão), a leader lock expira
- Assim que a leader lock expira, uma “corrida” (**leader race**) é iniciada no cluster Patroni
  - Os standbys competem pela obtenção da leader lock
  - O standby saudável e mais atualizado que vencer a corrida, adquire a leader lock
- O standby que adquire a leader lock é promovido a primary
  - Os demais standbys do cluster Patroni são reconfigurados como standby do novo primary
- Benefício: O cluster se cura sozinho automaticamente (self-healing), elegendo um novo primary para receber consultas de leitura e escrita

O que acontece quando o antigo líder volta?

- O agente Patroni inicia no líder e consulta o status no DCS
  - Ele percebe que não é mais o líder, e que há outro líder no cluster Patroni
  - Ele inicia um processo de rejoin no cluster Patroni
- O processo de rejoin:
  - Se possível, utiliza **pg_rewind** para sincronizar o primary antigo com o novo primary
    - Se não for possível, utiliza **pg_basebackup** para recriar o primary antigo como um standby
  - Reconfigura o primary antigo como standby do novo primary
- Benefício: O primary antigo se cura sozinho automaticamente, se reintegrando ao cluster como um standby

Configuração do Patroni

Tipos de configuração

- Configuração global (dynamic configuration):
  - Inicialmente definida através do arquivo de configuração **patroni.yml**
    - Quando o primeiro membro do cluster Patroni é criado, ele copia essa configuração para o DCS
    - Depois disso, essa configuração é ignorada nesse arquivo e mantida através do DCS
  - Aplica-se a todos os membros do cluster Patroni

- Configuração local por arquivo (local configuration):
  - Definida através do arquivo de configuração **patroni.yml**
  - Aplica-se somente ao membro Patroni local e tem precedência sobre a dynamic configuration

Tipos de configuração

- Configuração local por variáveis de ambiente (environment configuration):
  - Semelhante à local configuration, mas definida através de variáveis de ambiente
  - Tem precedência sobre a local configuration

- Mais informações em https://patroni.readthedocs.io/en/latest/patroni_configuration.html

Overview do **patroni.yml** – configuração geral

```
# Namespace dentro do DCS onde as chaves desse cluster são salvas. A combinação
# namespace + scope define o prefixo das chaves.
namespace: todos-meus-clusters-patroni

# Nome do cluster Patroni ao qual este membro pertence.
scope: meu-cluster-patroni-a

# Nome deste membro Patroni. Deve ser único dentro do cluster Patroni.
name: meu-membro-patroni-1
```

Overview do **patroni.yml** – configuração de log

```
# Configuração de log desse membro.
log:
  # DEBUG, INFO, WARNING, ERROR ou CRITICAL.
  level: INFO

  # plain ou json.
  type: plain

  # Diretório onde os arquivos de log do Patroni serão armazenados.
  # Se "dir" estiver definido, o Patroni registra em arquivos, caso contrário,
  # registra em stdout/stderr.
  dir: /var/log/patroni
```

Overview do **patroni.yml** – configuração do DCS

```
# etcd3 usa etcd API 3 como DCS.
etcd3:
  # Lista de endereços IP (ou host) e portas dos membros do etcd
  hosts:
    - meu-membro-etcd-1:2379
    - meu-membro-etcd-2:2379
    - meu-membro-etcd-3:2379

  # Protocolo para conectar aos hosts etcd -- http ou https.
  protocol: http
```

Overview do **patroni.yml** – configuração da REST API

```
# Configuração da REST API desse membro.
restapi:
  # IP (ou host) e porta em que o Patroni escuta as requisições.
  listen: meu-membro-patroni-1:8008

  # IP (ou host) e porta que os outros membros usam para conectar na REST API.
  connect_address: meu-membro-patroni-1:8008
```

Overview do **patroni.yml** – configuração de tags

```
# Customizações de comportamento.
tags:
  # Controla se esse nodo é promovível (false) ou não (true).
  nofailover: false

  # Ao invés do primary, use este membro como origem de replicação.
  replicatefrom: meu-membro-patroni-2
```

Overview do **patroni.yml** – configuração do Postgres

```
# Configuração do Postgres desse membro.
postgresql:
  # IP (ou host) e porta em que o Postgres escuta as requisições.
  listen: meu-membro-patroni-1:5432
  # IP (ou host) e porta que os membros usam para conectar.
  connect_address: meu-membro-patroni-1:5432

  # Caminho para o diretório de dados do Postgres.
  data_dir: /var/lib/pgsql/17/data
  # Caminho para o diretório que contém os binários do Postgres.
  bin_dir: /usr/pgsql-17/bin

  # Habilita o uso de pg_rewind para rejoin dessa instância.
  use_pg_rewind: on
```

Overview do **patroni.yml** – configuração do Postgres

```
# Configuração do Postgres desse membro.
postgresql:
  ...
  # Credenciais de acesso ao Postgres.
  authentication:
    # Superusuário para monitorar a instância e executar ações.
    superuser:
      username: usuario_superusuario
      password: password_superusuario

    # Usuário para conexões de replicação.
    replication:
      username: usuario_replicacao
      password: password_replicacao
```

Overview do **patroni.yml** – configuração do Postgres

```
# Configuração do Postgres desse membro.
postgresql:
  ...
  # Lista de regras pg_hba.
  pg_hba:
    - 'host all all 0.0.0.0/0 scram-sha-256'
    - 'host replication all 0.0.0.0/0 scram-sha-256'

  # Customização de parâmetros.
  parameters:
    shared_buffers: '4GB'
    work_mem: '16MB'
```

Overview do **patroni.yml** – configuração de bootstrap

```
# Usado somente a criação do primeiro membro do cluster Patroni.
bootstrap:
  # Opções para customizar o initdb
  initdb:
    - data-checksums  # Para --data-checksums
    - locale: 'C.UTF-8'  # Para --locale='C.UTF-8'
```

Overview do **patroni.yml** – configuração de bootstrap

```
bootstrap:
  ...
  # A dynamic configuration initial do cluster Patroni.
  dcs:
    # Quanto tempo (segundos) esperar entre as rodadas de health check.
    loop_wait: 10
    # Tempo de vida (segundos) da leader lock antes da sua expiração.
    ttl: 30
    # Timeout (segundos) para tentar operações novamente no etcd e Postgres
    # Ao enfrentar falhas, o Patroni torna o líder temporariamente em read-only por
    # precaução.
    retry_timeout: 10
```

Overview do **patroni.yml** – configuração de bootstrap

```
bootstrap:
  ...
  dcs:
    ...
    # Máximo de lag (bytes) para ser considerado promovível.
    maximum_lag_on_failover: 1048576
    # off = async, on = sync (priority) ou quorum = sync (quorum).
    synchronous_mode: off
    # Quantos dos standbys devem ser síncronos.
    synchronous_node_count: 1

    # Semelhante ao que vimos antes, mas para todos os membros do cluster Patroni
    postgresql:
      use_pg_rewind: on
      ...
```

Operações no Patroni

Usando o **patronictl**

- Aplicação CLI para facilitar as operações no cluster Patroni. Tem vários subcomandos, entre eles:
  - **edit-config** :
    - Altera a dynamic configuration no DCS
    - Faz um **reload** implícito nos membros do cluster Patroni
    - Dica: Use **show-config** se quiser somente consultar a dynamic configuration do DCS
  - **list**:
    - Mostra os membros do cluster Patroni
    - Dica: Use **topology** se quiser ver a lista em modo de “árvore”
  - **pause**:
    - Para pausar o gerenciamento de failover automático no cluster Patroni
    - Útil para fazer manutenções no Postgres, evitando failovers inesperados
    - Dica: Use **resume** para resumir o gerenciamento de failover automático

Usando o **patronictl**

- Aplicação CLI para facilitar as operações no cluster Patroni. Tem vários subcomandos, entre eles:
  - **reload** :
    - Para recarregar as configurações
    - Faz o agente Patroni ler o arquivo de configuração novamente
    - Além disso, executa **pg_ctl reload** na sua instância Postgres
  - **restart** :
    - Reinicia o Postgres
    - A operação é conduzida pelo Patroni, o que evita failovers inesperados
    - Útil para aplicar upgrades de versão no Postgres, ou aplicar configurações do Postgres que requerem restart
    - Dica: Pode ser agendado para um momento futuro

Usando o **patronictl**

- Aplicação CLI para facilitar as operações no cluster Patroni. Tem vários subcomandos, entre eles:
  - **switchover** :
    - Troca a role primary e standby entre dois membros do cluster
    - Dica: Pode ser agendado para um momento futuro
  - **reinit** :
    - Recriar um standby com **pg_basebackup** (por padrão)

- Mais informações em https://patroni.readthedocs.io/en/latest/patronictl.html

Overview da **REST API**

- A REST API disponibiliza diversos endpoints
  - Permite fazer operações parecidas com as expostas pelo **patronictl**
  - Usando HTTP requests como **GET**, **PUT**, **PATCH** e **DELETE**

- Além disso, permite também:
  - Monitoramento dos membros do cluster Patroni
  - Integração com ferramentas como HAProxy

- Mais informações em https://patroni.readthedocs.io/en/latest/rest_api.html

Outras funcionalidades

Outras funcionalidades

- Standby cluster
  - É um cluster Patroni onde todos os membros são standbys
  - O líder desse cluster conecta em uma instância Postgres externa ao cluster Patroni
  - Mais informações em: https://patroni.readthedocs.io/en/latest/standby_cluster.html

- Integração com watchdog
  - Maior proteção contra split-brain
  - Caso o agente do Patroni líder “morra” por qualquer motivo, evita que o Postgres continue rodando como primary
    - O dispositivo watchdog força o desligamento da máquina, derrubando tudo, inclusive o Postgres (SMITH)
  - Mais informações em https://patroni.readthedocs.io/en/latest/watchdog.html

Outras funcionalidades

- DCS Failsafe Mode
  - Caso o DCS esteja temporariamente indisponível, evita que o líder entre em modo read-only
    - Desde que todos os membros Patroni estejam acessíveis entre si
  - Mais informações em: https://patroni.readthedocs.io/en/latest/dcs_failsafe_mode.html

Outras dicas

Outras dicas

- Mantendo uma arquitetura segura:
  - A maioria dos exemplos citados durante a apresentação mostraram cenários mais simples:
    - Sem criptografia do canal de comunicação
    - Em alguns casos até mesmo sem autenticação do cliente
  - Em um cluster de produção, pelo menos criptografe o canal de comunicação e autentique os clientes:
    - Criptografia do canal de comunicação é obtida com um certificado SSL/TLS em cada servidor de serviço
    - Autenticação básica é obtida com usuário e senha em cada servidor de serviço
  - Indo além:
    - Restrinja acesso aos serviços somente aos clientes desejados (usando firewall, **pg_hba.conf** , allowlists, etc.)
    - Use mTLS entre cada cliente e servidor, de forma que eles verifiquem o certificado SSL/TLS um do outro

Outras dicas

- Perguntas frequentes sobre Patroni:
  - Explore a página https://patroni.readthedocs.io/en/latest/faq.html
    - Contém diversas perguntas frequentes respondidas
    - É uma boa fonte de aprendizado sobre Patroni

Perguntas?

barthisrael@gmail.com