FullCycle

Módulo 3 - Balanceamento de carga e roteamento

Definição

Balanceamento de carga (ou load balancing) é uma técnica utilizada para distribuir o tráfego de rede ou solicitações de serviços de forma equilibrada entre múltiplos servidores, recursos ou instâncias.

O objetivo é **garantir alta disponibilidade, escalabilidade e desempenho** dos sistemas, evitando que um único recurso fique sobrecarregado

Alta disponibilidade

Alta disponibilidade (HA) é uma característica de um sistema que visa garantir o máximo de tempo possível de funcionamento contínuo, minimizando interrupções não planejadas de serviço. Um sistema altamente disponível é resiliente a falhas, preparado para manter operações mesmo diante de problemas em componente individuais.

Logo o objetivo é atingir acordos de SLA (Service Level Agreeement) de disponibilidade, frequentemente expressos como "nove" percentuais (e.g. 99.9%, 99.99%, etc.)

Alta disponibilidade

Temos alguns pilares:

- **Redundância:**
  - Onde procuramos sempre eliminar os pontos únicos de falha, através da existência de componentes em duplicidade;
  - Aplicação de modelos como ativo-ativo e ativo-passivo;

- **Eliminação de Single Point of Failure (SPFOs):**
  - Todo componente da solução deve ter um substituto automático ou manual;

Alta disponibilidade

Temos alguns pilares:

- **Failover e Failback:**
  - Failover, que se refere a transferência automática ou manual para uma instância de backup;
  - Failback, que se refere ao retorno à instância primária original após a recuperação;

- **Monitoramento e Health Checks:**
  - Monitoramento contínuo da saúde dos componentes para permitir respostas rápidas a falhas;
  - Essencial para decisões automatizadas de failover e escalonamento;

Alta disponibilidade

Temos alguns pilares:

- **Escopo de Alta Disponibilidade**
  - **Intra-Zona:** redundância dentro de uma mesma zona de disponibilidade;
  - **Inter-Zonas (Multi-AZ):** alta disponibilidade dentro da mesma região, entre AZs;
  - **Inter-Regiões (Multi-Region):** disponibilidade geograficamente distribuída;
  - **Multi-Cloud ou Hybrid-Cloud:** redundância entre provedores ou on-premisses juntamente com cloud;

Alta disponibilidade

Topologia de componentes e cálculo:

- Os componentes da solução sempre estarão dispostos duas maneiras, sendo elas:
  - **Sequenciais,** onde se um falhar, todo o sistema falha, sendo assim, a disponibilidade total é o produto das disponibilidades dos componentes;
  - **Paralelos (redundantes),** onde dessa forma temos um aumento da disponibilidade geral, sendo assim, um componente sempre continua mesmo que o outro falhe;

- Exemplo:
  - Dois componentes em sequência com 99.9% cada → 0.999 × 0.999 = 99.8%
  - Dois em paralelo com failover → pode atingir > 99.999%

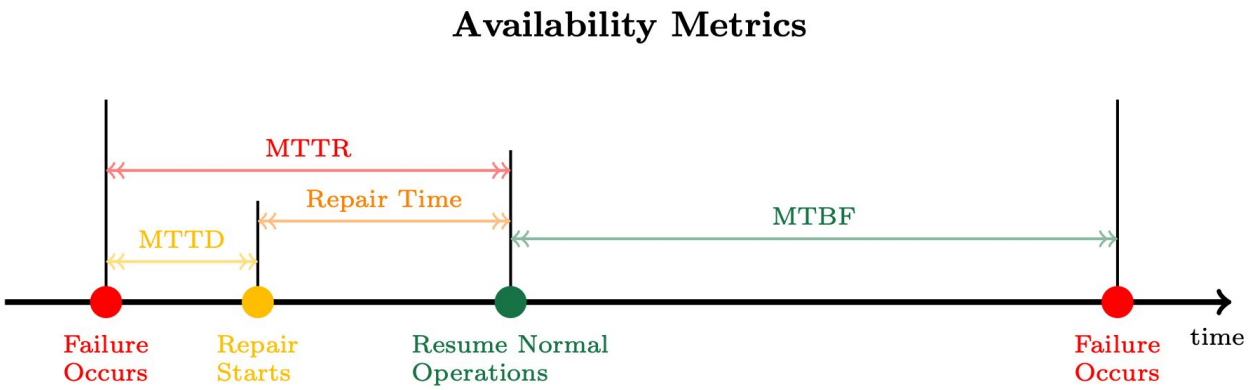
Alta disponibilidade

Topologia de componentes e cálculo:

$$A = \frac{uptime}{uptime + downtime}$$

Alta disponibilidade

Topologia de componentes e cálculo:



Alta disponibilidade

Topologia de componentes e cálculo:

$$A = \frac{MTBF}{MTBF + MTTR}$$

Alta disponibilidade

Arquiteturas de alta disponibilidade:

- **Arquiteturas Ativo-Ativo:**
  - Ideal para balanceamento geográfico e latência;
  - Mais complexas (consistência, conflitos de escrita, roteamento);
- **Arquiteturas Ativo-Passivo:**
  - Simples de configurar;
  - Úteis quando consistência forte é prioritária;
  - Failover precisa ser rápido para cumprir SLA;

Alta disponibilidade

Algumas boas práticas para ambientes em cloud pública:

- **AWS (Exemplo):**
  - **Zonas de Disponibilidade (AZs):** data centers discretos interconectados em uma região;
  - **Regiões:** separadas geograficamente, com latência e isolamento maiores;
  - **Serviços Gerenciados:** como RDS Multi-AZ, ALB/NLB, Auto Scaling, S3, Route 53;

- **Cloud Patterns:**
  - Stateless App + Auto Scaling + Load Balancer;
  - Stateful Layer com replicação (ex: Aurora, DynamoDB Global Tables);
  - Traffic routing inteligente (ex: DNS failover);

Alta disponibilidade

Algumas boas práticas para ambientes que realizam integrações com SaaS ou Micro-SaaS:

- Alta Disponibilidade com Serviços Externos;
- Avaliação de SLA do provedor externo;
- Modelos de fallback: ex. usar múltiplos gateways de pagamento;
- Timeouts e retries com circuit breakers;
- Buffering (mensageria, filas, cache local) para reduzir impacto de indisponibilidade temporária;
- Observabilidade ativa sobre os serviços externos;

Algoritmos de balanceamento de carga

- **Round Robin:** Distribui as requisições sequencialmente entre os servidores disponíveis em uma lista circular.
  - Vantagens:
    - Simples de implementar
    - Boa distribuição em cenários homogêneos
  - Desvantagens:
    - Não considera carga real dos servidores;

Algoritmos de balanceamento de carga

- **Least Connections:** A requisição é encaminhada para o servidor com o menor número de conexões ativas.
  - **Vantagens:**
    - Dinâmico e responsivo à carga atual
    - Boa escolha para conexões persistentes (ex: websockets)
  - **Desvantagens:**
    - Requer monitoramento em tempo real
    - Pode não refletir carga computacional real

Algoritmos de balanceamento de carga

- **IP Hash (Consistent Hashing):** O hash do IP do cliente (ou outro identificador, como session ID) determina o servidor de destino. O mesmo cliente tende a ir sempre para o mesmo backend.
  - **Vantagens:**
    - Mantém afinidade (session persistence)
    - Útil para cache, sessões em memória
  - **Desvantagens:**
    - Pode gerar distribuição desigual
    - Sensível a mudanças na lista de servidores

Algoritmos de balanceamento de carga

- **Weighted Round Robin:** Como o round robin, mas com pesos atribuídos aos servidores. Servidores com maior peso recebem mais requisições.
  - **Vantagens:**
    - Útil para ambientes heterogêneos
    - Fácil ajuste de distribuição
  - **Desvantagens:**
    - Pesos estáticos, não refletem carga em tempo real

Algoritmos de balanceamento de carga

- **Flow Hash:** é um algoritmo de balanceamento de carga que distribui tráfego com base em um hash determinístico calculado a partir de atributos da conexão de rede, comumente conhecidos como 5-tuple (Source IP, Source Port , Destination IP, Destination Port , Protocol);
  - **Vantagens:**
    - Determinístico;
    - Escalável;
    - Sem estado;.
  - **Desvantagens:**
    - Redistribuição total;
    - Sem afinidade por cliente;
    - Não adapta a carga atual dos targets;
    - Pode concentrar tráfego se muitos fluxos compartilham a mesma tupla;

Algoritmos de balanceamento de carga

- **Least Outstanding Requests:** é um algoritmo de balanceamento de carga que distribui novas requisições para o servidor que tiver o menor número de requisições pendentes (em processamento) no momento;
  - **Vantagens:**
    - Adaptativo;
    - Melhora o tempo de resposta médio;
    - Evita sobrecarga em servidores lentos ou com menor capacidade;
  - **Desvantagens:**
    - Requer estado;
    - Pode gerar latência adicional dado ineficiência do monitoramento;

Algoritmos de balanceamento de carga

Demonstração:

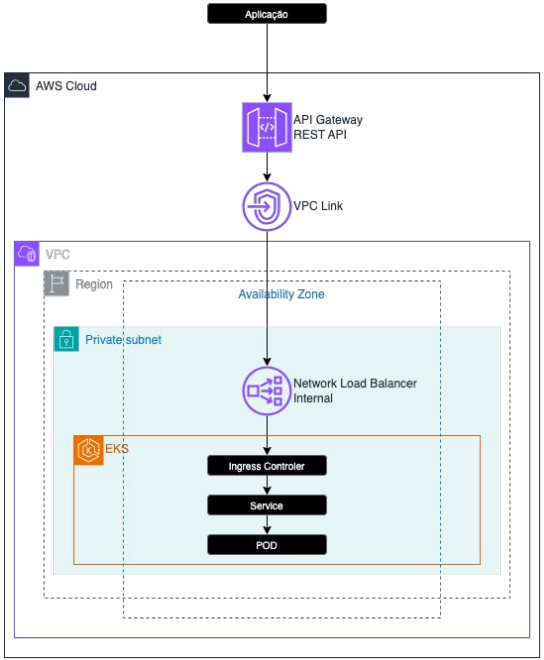
Nesta primeira demonstração iremos observar o comportamento de dois balanceadores de carga, onde um estará trabalhando no modo **Round Robin** e o outro no modo **Weighted Round Robin**, onde definimos um peso no balanceamento para cada workload definido no balanceador.

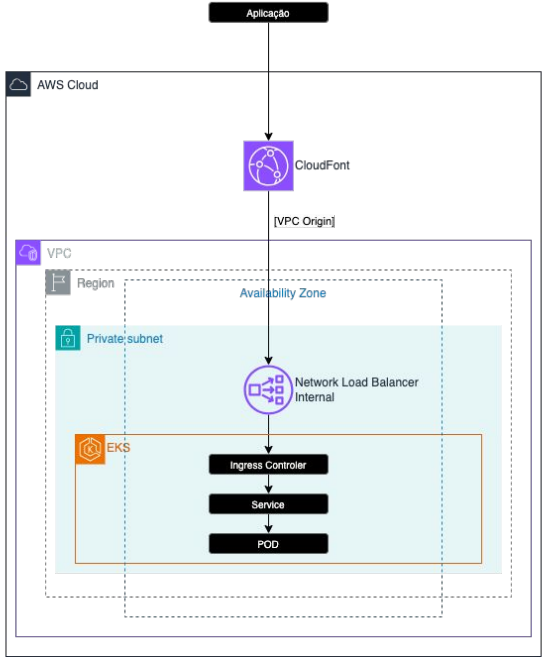
Balanceadores L4

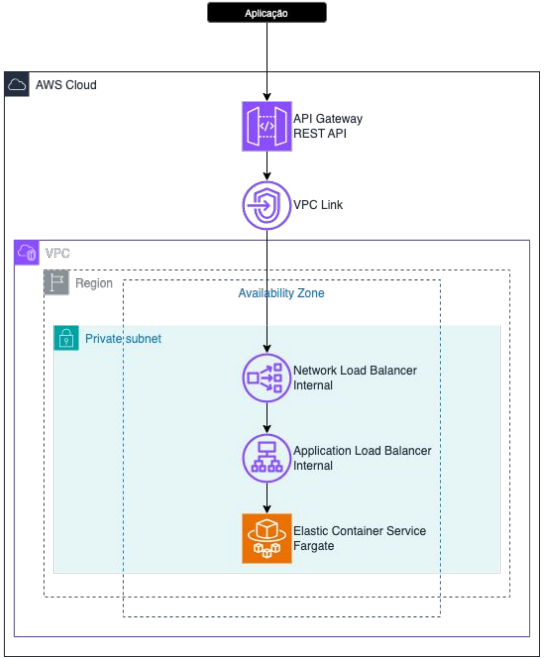
- Operam na camada de transporte (TCP/ UDP) do modelo OSI;
- Realizam o roteamento de tráfego com base em um IP de origem/ destino e porta;
- São mais eficientes e adicionam pouca latência na comunicação dado terem lógicas de roteamento mais limitadas;
- Alguns exemplos são:
  - HAProxy;
  - AWS Network Load Balancer;
  - MetalLB;
  - Keeplived com VIP;

Balanceadores L4

Exemplos de topologias





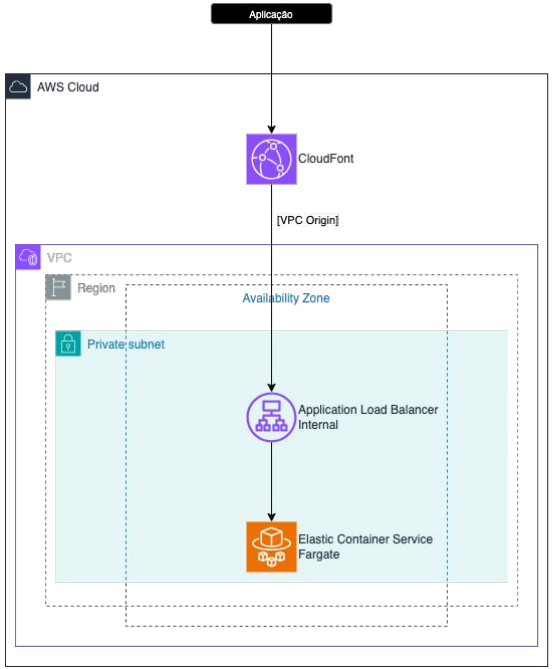


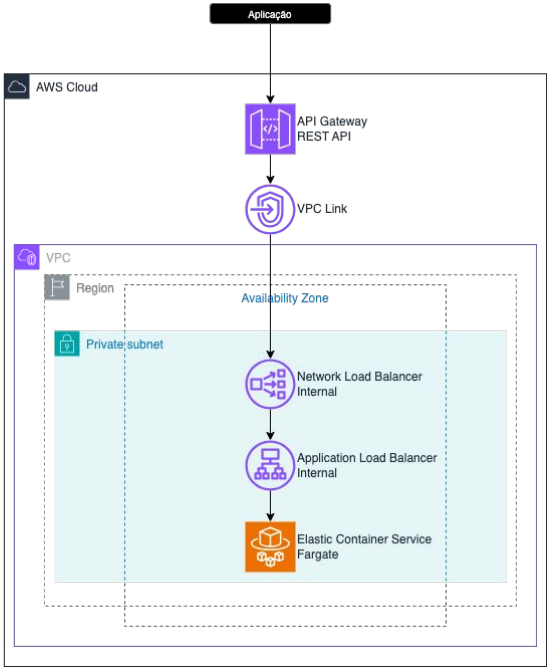
Balanceadores L7

- Operam na camada de aplicação, através de protocolos HTTP e HTTPS;
- Realizam o roteamento com base URL, cookies, headers, hostnames, etc.;
- Ideais para APIs, aplicações web e ingress controllers;
- Alguns exemplos são:
  - NGINX;
  - HAProxy;
  - AWS Application Load Balancer (ALB);
  - Kubernetes Ingress Controller;
  - Traefik;
  - Istio Gateway;
  - Kong (Ingress Controller & Gateway);

Balanceadores L7

Exemplos de topologias





Balanceadores L7

Tipos de roteamento

- **Path-based Routing:** Realiza o roteamento da requisição com base na URL do caminho.
    - Exemplos:
        - /api/* → serviço backend da API
        - /admin/* → painel administrativo
        - /imagens/* → serviço de CDN interna

Balanceadores L7

Tipos de roteamento

- **Host-based Routing**: Realiza o roteamento da requisição com base no hostname requisitado.
    - Exemplos:
        - www.meusite.com → frontend React
        - api.meusite.com → backend Node.js
        - cdn.meusite.com → arquivos estáticos

Balanceadores L7

Tipos de roteamento

- **Header-based Routing**: Realiza o roteamento da requisição com base nos valores presentes nos cabeçalhos HTTP.
    - Exemplos:
        - X-Region: US → cluster nos EUA
        - X-Tenant-ID: tenant1 → microsserviço exclusivo por cliente
        - User-Agent contém Mobile → rota para layout mobile

Balanceadores L7

Tipos de roteamento

- **Query-based Routing**: Realiza o roteamento da requisição com base nos valores presentes nos parâmetros da URL.
    - Exemplos:
        - ?env=dev → ambiente de desenvolvimento
        - ?abtest=variant2 → experimento de A/B testing
        - ?feature=novo-carrinho → ativação de feature flag

Balanceadores L7

Tipos de roteamento

- **Cookie-based Routing**: Realiza o roteamento da requisição com base nos valores presentes nos cookies enviados.
    - Exemplos:
        - Cookie user_group=vip → servidor premium
        - Cookie experiment=a → target group da variante A

Balanceadores L7

Tipos de roteamento

- **Method-based Routing**: Realiza o roteamento da requisição com base nos verbos HTTP enviados (GET, POST, DELETE, etc.).
    - Exemplos:
        - GET → API cacheável
        - POST → API de escrita com maior isolamento

Balanceadores L7

Sticky Sessions

São uma técnica usada em balanceadores de carga para garantir que todas as requisições de um mesmo cliente sejam direcionadas sempre para o mesmo servidor, onde geralmente são implementado usando cookies, por isso é uma funcionalidade presente em balanceadores L7.

Geralmente, os casos que necessitam deste tipo de implementação são soluções e aplicações com característica stateful, cujo mantém dados da sessão do usuário na memória do servidor de backend.

Caso não seja uma solução de WebSocket, geralmente iremos ver esta necessidade mais em aplicações legadas.

Balanceadores L7

Demonstração

Nesta segunda demonstração iremos observar o comportamento de um balanceador de carga, cujo estará trabalhando no modo **Round Robin**, porém, com definições de regras de roteamento, sendo elas Hosted-based, Path-based e rota de fallback (default).

Smart Routing/ Policy-Based Routing

Smart Routing ou **Policy-Based Routing** é uma abordagem de roteamento que permite decidir o caminho que o tráfego deve seguir com base em políticas específicas, em vez de depender apenas das tabelas de roteamento tradicionais baseadas em destino IP.

O objetivo é redirecionar o tráfego para locais específicos, seja por uma questão de regulação, melhor experiência do usuário ou por restrição de conteúdo;

Smart Routing/ Policy-Based Routing

Tipos de smart ou policy-based routing

- **Geolocation-Based Routing:** Realiza o roteamento das requisições com base na localização geográfica do cliente, normalmente determinada pelo IP de origem;
    - Casos de usos:
        - Direcionar usuários para data centers ou regiões mais próximas fisicamente;
        - Servir conteúdo local ou com restrições de localização (ex: leis regionais de dados);

Smart Routing/ Policy-Based Routing

Tipos de smart ou policy-based routing

- **Latency-Based Routing:** Realiza o roteamento das requisições para o endpoint que oferece a menor latência de rede percebida, com base em medições entre os resolvers DNS e os endpoints;

    - Casos de usos:
        - Melhorar a experiência do usuário em aplicações globais;
        - Reduzir a latência de aplicações interativas (ex: jogos, streaming, SaaS);
        - Garantir que clientes acessem sempre o servidor mais rápido, não apenas o mais próximo;

Smart Routing/ Policy-Based Routing

Demonstração

Para a última demonstração iremos observar o comportamento referente à **Smart Routing** ou **Policy-Based Routing**, neste contexto usando o serviço do **Route53** da AWS. Neste laboratório iremos provisionar duas infraestruturas iguais, porém em regiões diferentes, no caso, **us-east-1** que seria Norte Virgínia, Estados Unidos; e **sa-east-1** que no caso é São Paulo, Brasil.