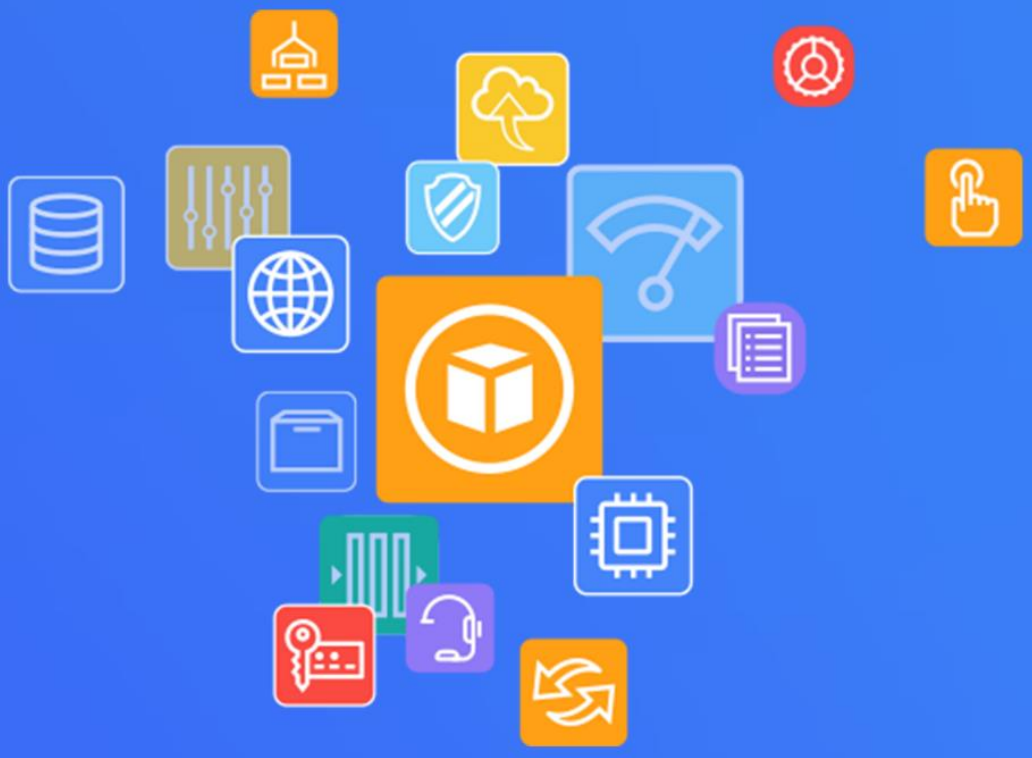
amazon web services

Implementando Roteamento Automático por Falha de Conteúdo para outras origens no Amazon CloudFront



por Ricardo Geh e Felipe Garcia

Implementando Roteamento Automático por Falha de Conteúdo para outras origens no Amazon CloudFront

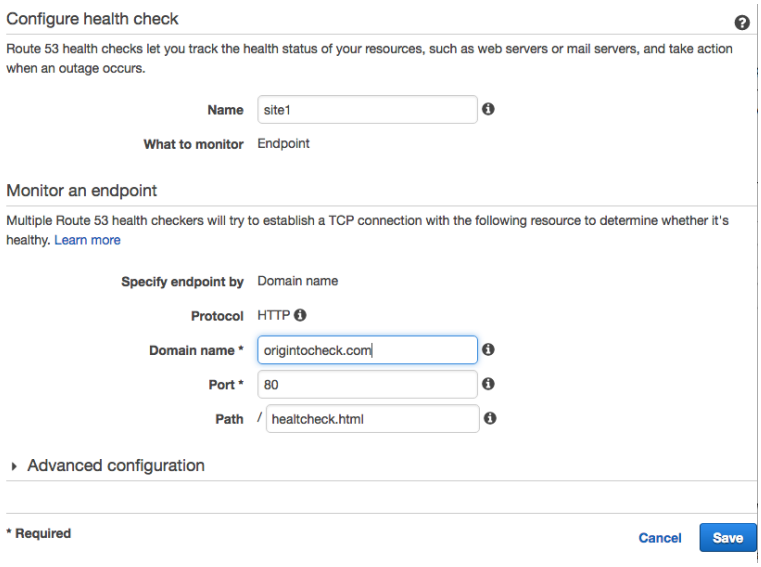
por Ricardo Geh e Felipe Garcia | em 15 JUL 2016 | in Amazon CloudFront | Permalink | Comments

Neste post vamos mostrar como o Amazon Cloudfront pode entregar conteúdo em origens secundárias baseado na falha de um conteúdo específico na origem principal. A solução implementada é simples, poderosa e pode endereçar uma variedade de casos de uso.

O Desafio

Atualmente a AWS suporta nativamente uma falha completa de origem através do Amazon Route 53. Com o Route 53, é possível configurar uma página específica de "health check" (checkagem da saúde da origem), e no caso da detecção de uma falha, o DNS irá resolver para a origem de falha.

Segue abaixo como criar e configurar um "health check" no Amazon Route 53:



Existem alguns cenários que não conseguimos endereçar com o Route 53. Por exemplo: soluções que requerem um tempo muito reduzido de falha, já que existe um tempo necessário para a detecção e propagação do novo DNS no caso de falha.

Outro exemplo interessante seria a falha parcial de uma origem. Ou seja, se parte do seu site/aplicação/origem parar de funcionar, não é possível efetuar a rota de falha somente para o conteúdo que falhou.

Outros casos envolvem o momento de uma migração de um storage de objetos para o Amazon Simple Storage (Amazon S3), ou mesmo a construção de uma solução de Desastre e Recuperação (DR) de objetos estáticos do seu website.

Com esta solução, quando um asset específico, caminho ou API não é encontrada ou retorna um erro 4XX ou 5XX, ele automaticamente tentará buscar na origem secundária de falha.

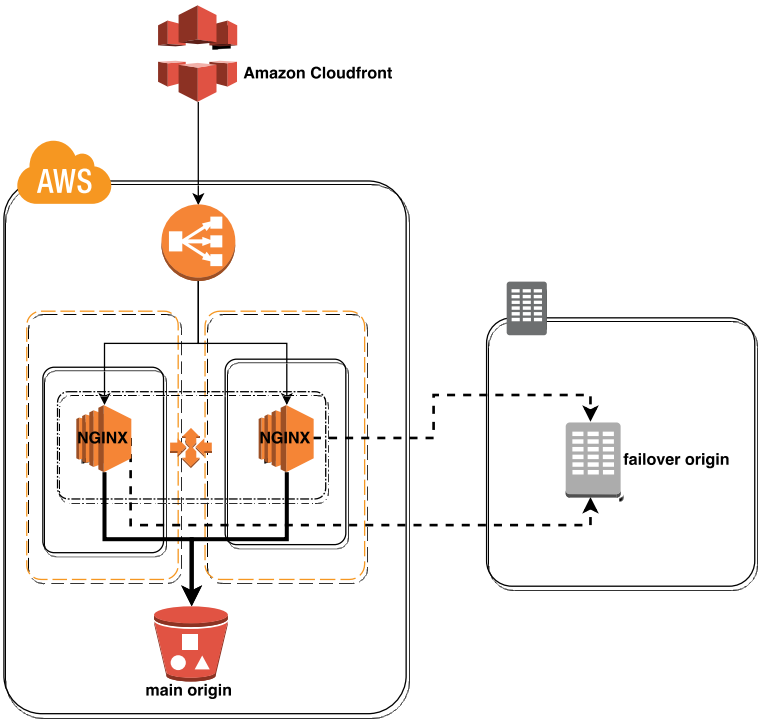
Exemplo: Criando uma distribuição Cloudfront resolvendo www.meusite.com.br, teríamos:

Ao fazer uma solicitação HTTP para a URL www.meusite.com.br/imagem1.jpg, o acesso será direcionado para a origem principal (http://123.123.123.123/imagem1.jpg)

Se no caso a imagem1.jpg não estiver retornando nesta origem, a solicitação HTTP será automaticamente roteada para a origem secundária (http://345.345.345.345/imagem1.jpg)

Implementando uma Solução Escalável e Altamente Disponível

Para obter este comportamento, basta direcionar sua distribuição Amazon Cloudfront para um AWS Elastic Load Balancer que irá direcionar tráfego para uma frota de web proxy reversos. Neste exemplo, iremos utilizar uma frota de servidores NGINX em múltiplas zonas de disponibilidade configurados em um grupo Auto Scaling. Com poucas modificações no arquivo de configuração do NGINX é possível fazer a configuração para uma origem de falha secundária baseado nos erros 4XX ou 5XX.



Sobre NGINX

O NGINX é uma solução de servidor HTTP e proxy reverso gratuita, aberta e de alta performance além de ser também um servidor de proxy para IMAP/POP3. O NGINX é conhecido por sua alta performance, estabilidade e riqueza de recursos, fácil configuração e baixo consumo de recursos computacionais.

NGINX é um dos poucos servidores escritos para endereçar o problema C10K. Diferente de servidores tradicionais, o NGINX não conta com threads para endereçar os requests. Ao invés disso, ele utiliza uma arquitetura mais escalável baseada em eventos assíncronos. Esta arquitetura utiliza uma quantidade menor de memória, e mais importante, uma quantidade consistente de memória quando ele recebe muita carga.

O NGINX é fantástico! Para mais informações, segue a página oficial da documentação, e outros projetos bem interessantes como Tengine e Openresty que mostram o poder do NGINX.

Configuração do NGINX

Para efetuar a configuração, iremos utilizar as diretivas disponíveis no módulo ngx_http_proxy_module. Abaixo mostraremos uma configuração simples, explicando o que cada linha configura.

```
server {
        listen *:80;
        location / {
                #se um objeto não é encontrado na origem principal, precisaremos pegar
              #os erros 4XX ou 5XX e fazer a falha ou encaminhamento para o endpoint
              #secundário configurado chamado de @fallback
                proxy_intercept_errors on;

                #todo request que chegar, deverá ser redirecionado para a origem
              #primária da CDN.
                proxy_pass https://www.myprimarycdnoriginlocation.com;

                #se um objeto receber o erro 404 not found (ex: web server como uma
              #origem) ou um 403 access denied (ex: S3 como endpoint de origem),
              #vamos fazer um failover para o endpoint secundário @fallback
                error_page 404 403 @fallback;
        }
        location @fallback {
                #também estaremos interessados nos erros da origem de fallback, mas
              #desta vez se recebermos um 404 ou 403, iremos direcionar para uma
              #página estática padrão de erro. Se desejar, poderíamos ainda encadear
              #erros em outras origens e repetir o processo
              proxy_intercept_errors on;

              #todo request que chega para o @fallback deverá ser redirecionado para
              #a origem secundária da CDN;
              proxy_pass https://www.mysecondarycdnoriginlocation.com;

              #página de erro 404 e 403 seria uma página estática
              error_page 404 403 /var/nginx/sites/cdnfallback/objeto_nao_encontrado.html;
        }
}
```

Quickstart

Para criar esta solução, estamos provendo um template Cloudformation que pode fazer o deploy de um Elastic Load Balancer, Um grupo auto-scaling com 2 instâncias de NGINX pré configuradas. Os parâmetros necessários serão:

- 2 subnets públicas para o ELB;
- 2 subnets privadas para as instâncias EC2 com NGINX;
- DNS para a origem principal;
- DNS para a origem de falha secundária;
- Tipo de instância EC2;
- Nome da chave pública da conta AWS para logar na conta;

Benchmark

A escolha do tipo da instância deve ser feita baseada nas suas necessidades específicas de carga. Para um uso básico, testamos com instâncias m3.medium atrás de um Elastic Load Balancer, obtendo cerca de 5.000 requests/s. Seu autoscaling group pode ser configurado para CPU, Rede ou uma métrica Cloudwatch personalizada.

Conclusão

O propósito da ferramenta Well Architected Framework é auxiliar clientes no processo de criação e manutenção de aplicações visando de segurança, disponibilidade, performance e eficiência em custo. Com a solução que descrevemos, é possível trabalhar com múltiplas origens para a sua CDN de forma altamente disponível e escalável. Trabalhando o viés de otimização de custos, seria ainda possível aplicar uma combinação de instâncias reservadas e instâncias on-demand.

Em um cenário ainda mais interessantes, é possível fazer o deploy desta mesma solução usando instâncias Reservadas e o Amazon Spot Fleet com Containers rodando NGINX usando o Amazon ECS (Amazon EC2 Container Services). Esta solução atinge um nível ainda superior de otimização de custos.

@Authors: Ricardo Geh and Felipe Garcia, Solutions Architecture in LatAm