Sistemas Paralelos e Distribuídos

Práticas – Aula 11

Práticas

Infraestrutura e redes em sistemas distribuídos

- Intro
- Componentes e conceitos
- Load balancers
- Exemplos com Nginx
- Exemplos com EnvoyProxy
- Conclusões

Intro

- Infraestrutura e redes são os componentes físicos que formam a arquitetura de um sistema distribuído.

- A evolução e aumento de capacidade da infrastrutura foi um facilitador para o desenvolvimento dos sistemas distribuídos.

- Além da rede, a infraestrutura pode englobar também os servidores e nós de computação, sistemas de armazenamento, data centers, dispositivos de borda, etc.

- No contexto de SD em cloud, a grande maioria da infraestrutura é transparente, mas ainda assim é possível explorar a qualidade da automação e elasticidade.

- Dispõe não somente de hardware, mas também de software (e.g: middleware).

Conceitos

- As qualidades de SD estão ligadas as decisões de design da infra.

- Largura de banda e latência.

- Consistência (Imediata/forte ou eventual).

- Gestão de recursos compartilhados.

- Performance.

- Escalabilidade (horizontal vs vertical).

- Segurança (autenticação, autorização, criptografia em trânsito/armazenamento, etc).

- Continuidade e resiliência (Redundância, replicação, recuperação, alta disponibilidade etc).

- Componentes: servidores, dispositivos de rede, software middleware, protocolo de rede, scripts para automação, sistema de controlo de configuração e ativos, firewalls, VPN, load balancers, etc.

Load Balancers

- É responsável por controlar o tráfego destinado a um ou mais serviços.

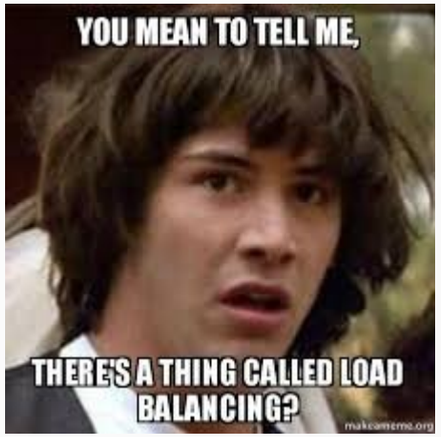
- É mecanismo chave para distributir tráfego de forma adequada aos servidores de núcleo do sistema.

- O LB facilita a performance (distribuição justa).

- O LB promove tolerância a falhas, por entregar maior disponibilidade e confiabilidade quando alguns servidores destinos estiverem fora de operação.

- O LB promove escalabilidade por facilitar a adição/remoção de nós do backend do sistema.

- Principais e mais populares LB em software: nginx e haproxy.



Exemplo com Nginx

- Uso de um sistema em containers. O ***nginx.conf*** é o arquivo principal onde as diretivas de frontend and backend são definidas.

```
events {
    worker_connections 1024;
}

http {
    upstream backend {
        server backend1:5000 max_fails=1 fail_timeout=5s;
        server backend2:5000 max_fails=1 fail_timeout=5s;
    }

    server {
        listen 80;
        server_name localhost;

        location / {
            proxy_pass http://backend;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_connect_timeout 1s;
            proxy_read_timeout 5s;
        }
    }
}
```

Exemplo com HAproxy

- Lógica similar ao do nginx, agora usando haproxy.cfg.

```
global
  daemon
  maxconn 256
  stats socket /var/run/haproxy.sock mode 660 level admin expose-fd listeners

defaults
  mode http
  timeout connect 1s #to backend
  timeout client 5s #client activity
  timeout server 5s #backend response
  timeout check 1s #health check
  log global
  option httplog

frontend stats
  bind *:8404
  stats enable
  stats uri /
  stats refresh 5s

frontend http_front
  bind *:80
  default_backend http_back

backend http_back
  balance roundrobin
  option forwardfor
  server backend1 backend1:5000 check inter 1s fall 1 rise 1
```

Atividade com HAproxy

- Crie um haproxy em docker que vai direcionar requisições HTTP, porta 80, de seu sistema local para dois diferentes destinos:
    - httpbin.org
    - jsonplaceholder.typicode.com

Verifique as requisições em ação e confirme qual o destino das mesmas.

P.S.: Caso não tenha docker instalado no seu sistema, utilize https://labs.play-with-docker.com/

EnvoyProxy

- Uso de loadbalancer avançado com capacidade de filtragem e múltiplos protocolos.
- Capacidade de operação entre L4-L7.
- Alta performance.
- Open source.
- Permite configuração dinâmica com uso de API.
- Observabilidade nativa.
- Empregue em diversos produtos atuais: Istio, AWS, App Mesh, etc.

Exemplo com EnvoyProxy

- Configurações principais do envoy proxy:

  - “static_resources” com “listeners”, “filter_chains” e “request_mirror_policy”.

  - “clusters” com “endpoints”, “health_checks” e “outlier_detection”.

Exemplo com EnvoyProxy

```
static_resources:
 listeners:
   - name: redis_proxy_listener
     address:
       socket_address:
         address: 0.0.0.0
         port_value: 16379
     filter_chains:
       - filters:
           - name: envoy.filters.network.redis_proxy
             typed_config:
               "@type": type.googleapis.com/envoy.extensions.filters.network.redis_proxy.v3.RedisProxy
               stat_prefix: redis_stats
               settings:
                 op_timeout: 5s
               prefix_routes:
                 catch_all_route:
                   cluster: primary
                   request_mirror_policy:
                     - cluster: mirror
                       exclude_read_commands: true
```

Exemplo com EnvoyProxy

```
clusters:
 - name: primary
   connect_timeout: 1s
   type: STATIC
   lb_policy: ROUND_ROBIN
   type: STRICT_DNS
   load_assignment:
     cluster_name: primary
     endpoints:
     - priority: 0
       lb_endpoints:
       - endpoint:
           address:
             socket_address:
               address: redis-primary
               port_value: 6379
     - priority: 1
       lb_endpoints:
       - endpoint:
           address:
             socket_address:
               address: redis-secondary
               port_value: 6379
```

(Repete-se a mesma definição para o cluster “mirror” invertendo-se primary e secondary)

Exemplo com EnvoyProxy

- Uso de loadbalancer avançado com capacidade de filtragem e protocolos para além de HTTP.

```
version: '3.8'

services:
  redis-primary:
    image: redis:7.2-alpine
    command: redis-server --appendonly yes
    ports:
      - "6379:6379"
    volumes:
      - redis-primary-data:/data
    networks:
      - redis-network

  redis-secondary:
    image: redis:7.2-alpine
    command: redis-server --appendonly yes
    ports:
      - "6380:6379"
    volumes:
      - redis-secondary-data:/data
    networks:
      - redis-network

  envoy-proxy:
    image: envoyproxy/envoy:v1.28-latest
    ports:
      - "16379:16379"
      - "9901:9901"
    volumes:
      - ./envoy-config.yaml:/etc/envoy/envoy.yaml
    depends_on:
      - redis-primary
      - redis-secondary
    networks:
      - redis-network
```

KeepaliveD

- Uma dupla camada de resiliência pode ser adicionada ao LB. Para isto podemos tomar vantagem do serviço KeepaliveD.

- Solução que entrega alta disponibilidade através do uso do protocolo Virtual Router Redundancy Protocol (VRRP).

- Utiliza “health-checks” para determinar estado do “link”.

- Manipula IP virtual para promover “uptime” de servidores web, load balancers, banco de dados, etc.

KeepaliveD

- Exemplo de configuração simples

```
vrrp_instance VI_1 {
    state MASTER
    interface wlo1
    virtual_router_id 51
    priority 101
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass 1234
    }

    virtual_ipaddress {
        192.168.1.100
    }
}
```

```
vrrp_instance VI_1 {
    state BACKUP
    interface wlo1
    virtual_router_id 51
    priority 100
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass 1234
    }

    virtual_ipaddress {
        192.168.1.100
    }
}
```

Conclusões

- Load Balancer tem caracaterísticas de rede e middleware.

- Uso de suporte (keepalived) para promover alta disponibilidade na infra.

- Outras funções igualmente importantes como formatação de URL/URI.

- Uso para rate limiting e segurança (com autenticação).

- Uso para SSL offload.

- Outra arquitetura/técnica bastante popular que atualmente domina as redes de sistemas distribuídos: “Content Delivery Network (CDN)”.

- Nginx mais usado quando se deseja embutir/integrar com códigos de serviços, e.g.: nginx com nodejs.

- https://medium.com/@rzuolo/splitting-traffic-with-nginx-is-a-cakewalk-1959289c7432.

FIM