UNIVERSIDADE ESTADUAL DO TOCANTINS – UNITINS
Curso de Sistemas de Informação – Disciplina: Sistemas Distribuídos

Período Letivo: 2025/2 – 7º Período – Câmpus Palmas

Alunos: João Victor Póvoa França e Hemilly Christinne Silva Pereira

Documentação de Planejamento de Arquitetura para Sistema Distribuído com Nginx, AWS e Node.js

PALMAS-TO

2025

1. Introdução

Este documento descreve a arquitetura distribuída a ser implementada no projeto, visando **alta escalabilidade**, **resiliência** e **alta disponibilidade** com a utilização do **Nginx** como balanceador de carga, **Node.js** para os microsserviços e **MongoDB** como banco de dados com replicação. O objetivo é criar uma solução que possa ser facilmente escalada e que continue operando sem interrupções, mesmo em caso de falha de servidores ou outros componentes críticos do sistema.

2. Arquitetura Proposta

A arquitetura proposta será baseada na seguinte configuração:

- **Nginx como Balanceador de Carga**: Nginx irá atuar como proxy reverso e balanceador de carga, distribuindo o tráfego de requisições entre diferentes instâncias de servidores backend (Node.js).

- **Servidores Web com Node.js (Express ou Nest.js)**: Cada instância do backend será implementada com Node.js, usando frameworks como Express.js ou Nest.js, para garantir escalabilidade e modularidade.

- **Banco de Dados com MongoDB Replicado:** Utilizaremos o MongoDB com replica set para garantir a persistência de dados, implementando a replicação master-slave (ou primária-secundária) entre as instâncias. A instância primária será responsável pelas operações de leitura e escrita, enquanto as instâncias secundárias replicarão os dados e poderão realizar leituras. Em caso de falha da instância primária, uma das instâncias secundárias será automaticamente promovida para primária, garantindo alta disponibilidade e redundância de dados.
- **Auto Scaling Manual com Nginx e Docker:** Para ajustar a infraestrutura conforme a variação de carga, implementaremos o auto scaling manual. Através de Nginx, configuraremos um balanceador de carga para distribuir as requisições entre múltiplas instâncias de servidores backend (Node.js) em containers Docker. Usaremos scripts de monitoramento de carga baseados no uso de CPU ou memória das instâncias para decidir quando adicionar ou remover instâncias do backend manualmente, escalando conforme a necessidade do tráfego.

3. Como a Arquitetura se Aplica no Projeto

A arquitetura foi projetada para garantir alta disponibilidade e escalabilidade. O Nginx será o ponto central para balanceamento de carga, garantindo que o tráfego de requisições seja distribuído de forma equilibrada. Em caso de falha de uma instância, o tráfego será automaticamente redirecionado para as instâncias saudáveis, evitando a interrupção do serviço. Com o MongoDB replica-se, a continuidade do serviço de banco de dados é garantida, com failover automático entre instâncias primária e secundária. Além disso, a escalabilidade manual através do Nginx e Docker permitirá ajustar a infraestrutura conforme a demanda do tráfego.

A configuração de **MongoDB com replica set** assegura que os dados do sistema estejam sempre disponíveis, com a capacidade de **failover automático** entre as instâncias primária e secundária. Caso a instância primária falhe, uma das secundárias será promovida automaticamente a primária, garantindo a continuidade dos serviços e a integridade dos dados.

Além disso, a arquitetura permite **escalabilidade manual** por meio do **Nginx** e **Docker**, o que possibilita a adição ou remoção de instâncias do backend conforme a demanda de tráfego. O **Nginx** será configurado para monitorar a carga dos servidores e realizar o balanceamento de forma eficiente, garantindo a elasticidade do sistema.

4. Ferramentas Escolhidas

Nginx

O **Nginx** será utilizado como **balanceador de carga** devido à sua alta performance e confiabilidade. Como **proxy reverso**, ele receberá todas as requisições e as direcionará para as instâncias backend. O Nginx será configurado para distribuir as requisições entre as instâncias de forma equilibrada, considerando a carga de cada servidor.

Docker

Para containerizar as instâncias de **Node.js** (Express.js ou Nest.js). O **Docker** garante que as instâncias sejam facilmente replicáveis e isoladas, facilitando o gerenciamento e deploy dos microsserviços.

Node.js (Express.js / Nest.js)

A aplicação será construída utilizando **Node.js** com **Express.js** ou **Nest.js**, ambos frameworks altamente eficientes para criar APIs RESTful e microsserviços escaláveis. O Node.js é a melhor opção devido ao seu desempenho, suporte a **event-driven architecture** e grande compatibilidade com ferramentas de escalabilidade na AWS.

MongoDB (Replica Set)

O **MongoDB** será utilizado como banco de dados, configurado em um **replica set** para garantir **alta disponibilidade** e **redundância**. A instância **primária** será responsável pelas operações de **leitura e escrita**, enquanto as **instâncias secundárias** replicarão os dados e poderão realizar leituras. Caso a instância primária falhe, uma das secundárias será automaticamente promovida a primária, garantindo que o banco de dados continue funcionando sem interrupções. Essa configuração assegura a escalabilidade e a resiliência do sistema.

5. Como Será Implementado e Como Funciona

1. Configuração do Nginx

O **Nginx** será configurado como **proxy reverso** e **balanceador de carga** para distribuir as requisições entre as instâncias do servidor Node.js. Cada instância do Node.js estará configurada em um **cluster de servidores** dentro da infraestrutura da AWS.

Exemplo de Configuração do Nginx:

```
None
http {
    upstream backend {
        server backend1.example.com;
        server backend2.example.com;
        server backend3.example.com;
```

```
    }

    server {
        listen 80;

        location / {
            proxy_pass http://backend;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
    }
}
```

Neste exemplo, o **upstream** define as instâncias backend e o **proxy_pass** distribui as requisições para essas instâncias.

2. Desenvolvimento do Backend com Node.js

O backend será desenvolvido utilizando Express.js ou Nest.js para criar APIs que atenderão as requisições enviadas pelo Nginx. Essas APIs serão distribuídas entre as várias instâncias do servidor, permitindo que o sistema seja escalável.

3. Configuração do Banco de Dados com MongoDB Replica Set

O banco de dados será configurado utilizando o MongoDB em um replica set, garantindo alta disponibilidade e redundância de dados. Uma instância primária será configurada

para realizar as operações de gravação, enquanto as instâncias secundárias replicarão os dados da primária e poderão ser utilizadas para leitura. Em caso de falha na instância primária, o MongoDB promoverá automaticamente uma das instâncias secundárias primárias, assegurando a continuidade do serviço e o balanceamento de carga entre as instâncias de banco de dados. Essa configuração oferece uma solução escalável e resiliente para o gerenciamento de dados.

sudo apt-get update

sudo apt-get install -y mongodb

Configurar o replica set no arquivo de configuração do MongoDB (/etc/mongod.conf):

yaml

```
replication:
  replSetName: "rs0"
```

- Inicializar o replica set. Conecte-se a uma instância do MongoDB (primeiro servidor) e execute os seguintes comandos:

bash

```
mongo --host <IP_da_primária>:27017
```

No shell do MongoDB:

javascript

```
rs.initiate({
  _id: "rs0",
  members: [
    { _id: 0, host: "<IP_da_primária>:27017" },
```

```
  { _id: 1, host: "<IP_da_secundária_1>:27017" },
  { _id: 2, host: "<IP_da_secundária_2>:27017" }
 ]
});
```

Verifique o status do replica set:

javascript

```
rs.status();
```

4. Auto Scaling e Monitoramento

- **Auto Scaling** será configurado para criar ou destruir instâncias de servidores Node.js automaticamente, dependendo da carga de tráfego.

- **Monitoramento**: Além disso, será implementado um sistema de monitoramento utilizando Nodemon ou outras ferramentas customizadas, para acompanhar o desempenho das instâncias e da aplicação. Em caso de falha em alguma instância do backend, o Nginx irá automaticamente redirecionar as requisições para as instâncias restantes, mantendo a continuidade do serviço sem interrupções. O monitoramento também irá incluir métricas como uso de CPU, memória e latência, ajustando a capacidade do sistema conforme a necessidade.

7. Conclusão

A arquitetura proposta, utilizando Nginx como balanceador de carga, Node.js para os microsserviços e MongoDB com replica set para alta disponibilidade, oferece uma solução robusta, escalável e de alto desempenho para o sistema. A implementação será realizada em fases, garantindo que cada componente seja configurado de forma adequada e integrada corretamente. Além disso, os testes de resiliência permitirão validar a capacidade do sistema de operar de forma contínua e eficiente, mesmo em condições adversas, assegurando sua estabilidade e confiabilidade a longo prazo.