Detecções de Anomalias na Rede através do uso de Machine Learning

Haroldo R. S. Silva[1]**,Pedro Jacobus**[1]**, Rafael Trevisan**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

Resumo. *O aumento da complexidade e sofisticação das ameaças cibernéticas demanda abordagens inovadoras para detectar anomalias no tráfego de rede. Este estudo explora a aplicação de um algoritmo de aprendizado de máquina para detecção de anomalias em ambientes de rede. Alavancando um conjunto de características extraídas de dados de tráfego de rede, empregamos um modelo de aprendizado de máquina para distinguir entre comportamentos normais e anômalos e exibir esta informação em uma interface.*

1. Introdução

A crescente complexidade de redes de computadores leva cientistas e engenheiros a considerarem métodos cada vez mais certeiros e sofisticados para garantir a segurança e estabilidade de suas redes. Uma nova ferramenta que pode auxiliar e aumentar tanto a segurança quanto o controle de gerentes de suas redes é o *Machine Learning*.

Machine Learning, ou Aprendizado de máquina é um subcampo da inteligência artificial que se concentra no desenvolvimento de algoritmos e modelos que permitem aos computadores aprender a partir de dados. Em vez de serem explicitamente programados para realizar uma determinada tarefa, os sistemas de aprendizagem de máquina utilizam dados para reconhecer padrões, fazer previsões ou tomar decisões com o mínimo de intervenção humana.

Esse trabalho tem como objetivo a implementação de um serviço capaz de detectar anomalias em um cluster Kubernetes enquanto sendo alimentado com dados do sistema. Para isso, este programa utiliza dados colhidos em tempo real pelo Prometheus, para realizar o método KNN levando em consideração instâncias de performance prévia desta rede.

O restante deste trabalho está organizado da seguinte maneira, a seção de metodologia, contendo detalhes da implementação, resultados, mostrando o funcionamento da arquitetura criada, e conclusões.

2. Metodologia

2.1. Setup

Para realizar essa tarefa, foi necessária a configuração de um *setup* utilizando o Minikube, o Istio e o Prometheus.

2.1.1. Minikube

Com a finalidade de executar clusters Kubernets locais na máquina virtual, foi necessária a instalação e configuração do minikube, juntamente com a instalação e configuração do

Kubectl, a fim de permitir a interação com estes clusters via linha de comando. Importante ressaltar o uso do Docker para a criação dos clusters.

2.1.2. Istio

Foi instalado o perfil demo do Istio para que as configurações base do Istio interfiram de uma maneira esperada. Para isso, foi escolhido o samples/bookinfo [Istio], configurando-se os pods *productPage*, *reviews* v1, v2 e v3, *ratings* e *details*.

2.1.3. Prometheus

O Prometheus é uma ferramenta nativa do Kubernetes criada para monitorar serviços em tempo real. Para fazer isso, serviços relevantes tem uma porta exposta, onde dados para o Prometheus são armazenados, e o servidor Prometheus periodicamente realiza o *scraping* destes pontos, mantendo sempre valores atualizados quanto a métricas e à saúde do sistema.

2.2. Anomaly Detector

Anomaly Detector[1] é uma aplicação desenvolvida para utilizar os dados fornecidos em tempo real pelo Prometheus para Machine Learning. Para isso, ela periodicamente faz *queries* para o Prometheus, recebendo os dados mais recentes referentes à saúde do ambiente. Esses dados são processados e adicionados a uma estrutura contendo as iterações passadas dessas métricas e o método KNN é utilizado para determinar quais são os momentos de maior estresse na rede, levando em consideração latência para a resposta de serviços e *throughput*. Para o desenvolvimento dessa aplicação foi utilizada a linguagem Python e as bibliotecas *Prometheus Python Client* [2], *scikit-learn* [3] e *Pandas* [4]. Para a detecção de anomalias, foram escolhidas as seguintes métricas disponibilizadas pelo istio: *istio_request_bytes_sum*, *istio_response_bytes_sum* e *istio_request_duration_milliseconds_sum*. Através dessas métricas são calculadas aproximações do *throughput* e da latência da rede. Os dados coletados montam um vetor bidimensional que é utilizado pela função *sklearn.neighbors.LocalOutlierFactor* [5] para detectar *outliers* com relação ao comportamento conhecido da rede.

2.2.1. Configuração para Service Mesh

Tendo a aplicação desenvolvida, foi necessária uma série de configurações para que ela possa ser integrada como um serviço no *service mesh*. Inicialmente é preciso criar uma imagem da aplicação, a partir de um *Dockerfile* utilizando o Docker, para que o serviço

[1]Código disponível em `https://github.com/hrssilva/istio-netmanager/tree/main/anomaly-detection`

[2]`https://github.com/prometheus/client_python`

[3]`https://scikit-learn.org/stable/index.html`

[4]`https://pandas.pydata.org`

[5]`https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html#sklearn.neighbors.LocalOutlierFactor`

possa ser criado no *cluster*, definindo qual a imagem base a ser usada e quais dependências são necessárias.

```
FROM python:latest

RUN pip install requests
RUN pip install kubernetes
RUN pip install scikit-learn
RUN pip install prometheus-client
RUN pip install pandas

COPY ./app /app
WORKDIR /app

CMD python main.py
```

Após a criação da imagem é necessário configurar o serviço no kubernetes. Tal configuração tem algumas particularidades, visto que é necessário registrá-la para *scraping* pelo Prometheus.

```
  template:
    metadata:
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "9090"
        prometheus.io/path: "/metrics"
      labels:
        app: anomaly-detector
        version: v1
```

O arquivo de configurações do kubernetes também é utilizado para configurar a aplicação em si (assim como fornecer o endereço do Prometheus, que é dinâmico, para recuperação de dados) através de variáveis de ambiente, permitindo uma maior flexibilidade em seu uso.

```
      containers:
      - name: anomaly-detector
        image: hrssilva/anomaly_detector:latest
        imagePullPolicy: Always
        env:
        - name: PROMETHEUS_ADDRESS
          value: prometheus
        - name: PROMETHEUS_PORT
          value: '9090'
        - name: KNN_NEIGHBORS
          value: '5'
        - name: WAIT_SEC
          value: '30'
        - name: MAX_ELEMENTS_COUNT
```

```
value: '256'
```

Observe que as conviguraçõ̃es de *WAIT SEC* e *MAX ELEMENTS COUNT* definem qual a janela de tempo em segundos (sempre partindo do ponto de início do serviço) que a aplicação utiliza para definir um comportamento normal para a rede.

$$TIME\_WINDOW = WAIT\_SEC * MAX\_ELEMENT\_COUNT$$

Os arquivos de configuração completos podem ser encontrados junto com o código da aplicação.

3. Resultados

Com as configurações é possível observar os comportamentos da rede através de buscas no Prometheus. Para isso foram disponibilizadas quatro buscas que retornam a quantidade de anomalias atualmente presentes na rede (*current_anomalies*), a quantidade de novas anomalias desde a última medição (*new_anomalies*), a quantidade total de anomalias detectadas desde o início da execução da aplicação (*detected_anomalies*) e a quantidade de medições sendo utilizadas para compor os dados (*anomaly_detector_samples_count*). Vemos na figura 1 que inicialmente a rede não apresenta nenhuma anomalia, visto que

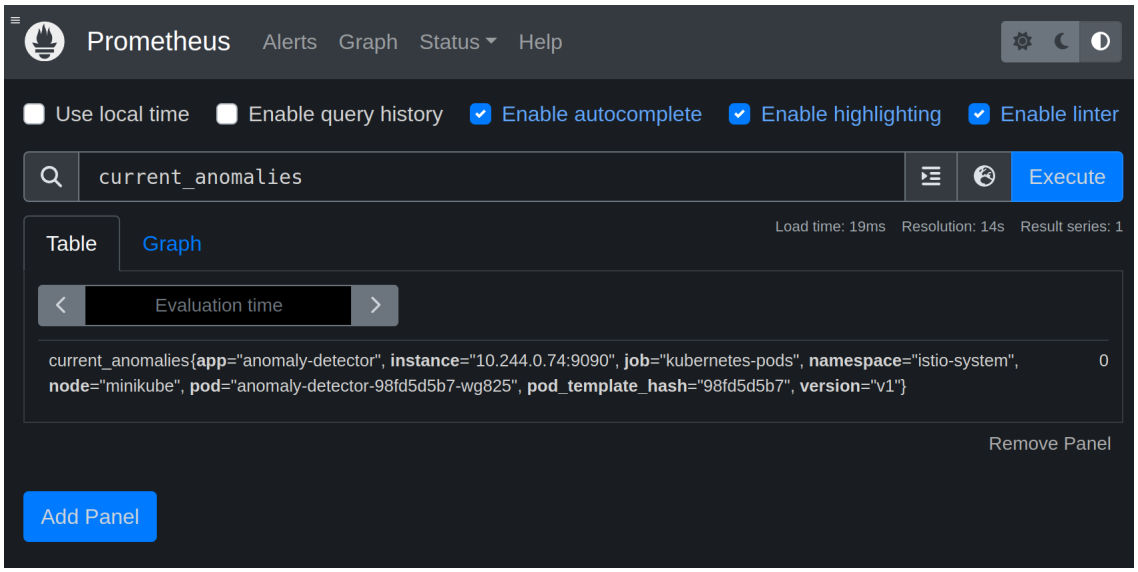

Figura 1. Métricas com a rede estável

não há inicialmente nenhum tráfego na rede, entretanto o estado atual da rede é coletado (neste exemplo a cada 30 segundos) pela aplicação desenvolvida como demonstrado na figura 2

Após realizarmos uma série de requisições para a aplicação de exemplo *bookinfo*, vemos na figura 3 que a aplicação acusa anomalias na rede, visto que a quantidade de dados sendo trafegados aumentou considerável e repentinamente.

Também vemos na figura 4 que apesar da aplicação reportar 2 anomalias, apenas 1 delas foi resultado da última medição.

Este, normalmente, é o resultado esperado, entretanto, é possível que diversas anomalias resultem de uma única medição. Isso ocorre quando o comportamento da rede

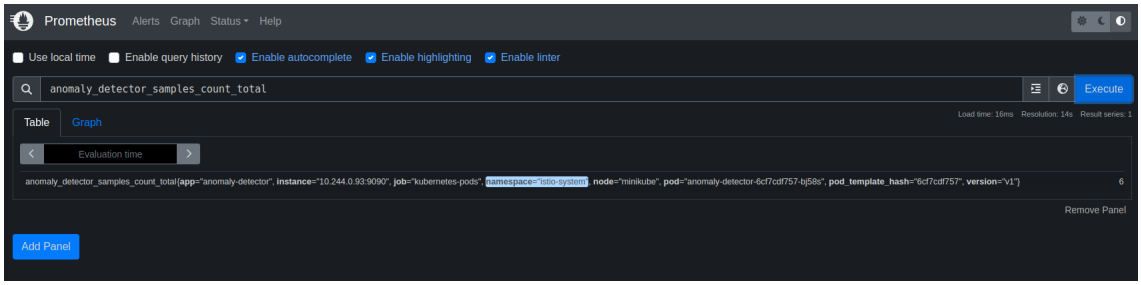

Figura 2. Total de medições utilizadas pelo modelo de ML

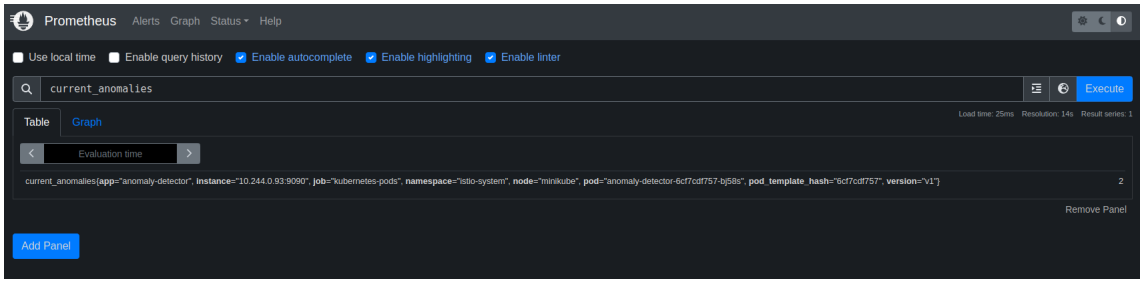

Figura 3. Total de anomalias detectadas após aumento de tráfego

Figura 4. Anomalias detectadas pela última medição

muda de forma que seu novo estado passe a ser considerado o de normalidade, podendo resultar que um *cluster* de dados, previamente representativo do estado normal da rede, passe a ser considerado anômalo de uma só vez.

As figuras 5, 6 e 7 demonstram o comportamento da aplicação durante um experimento de aproximadamente 15 minutos e foram retiradas diretamente do Prometheus, demonstrando como um administrador poderia utilizar a aplicação desenvolvida para analisar o comportamento da rede.

A figura 5 mostra o crescimento da quantidade de amostras usadas para detecção de anomalias durante o experimento. A figura 6 demonstra o comportamento esperado de registro de anomalias, visto que foi gerado um tráfego aumentado durante 5 minutos. Já a figura 7 é o mais interessante, visto que ele demonstra como a aplicação se adapta a situação. Esse comportamento parece não fazer muito sentido inicialmente, mas, se observado em conjunto com as figuras 5 e 6, podemos notar que esse comportamento se da devido a quantidade de amostras de trafego baixo em relação a quantidade de amostras de tráfego alto, gerando uma indeterminação a o que é considerado o estado normal da rede, mas estabilizando no final, quando mais amostras são coletadas.

Figura 5. Captura de estados da rede para definição de comportamento

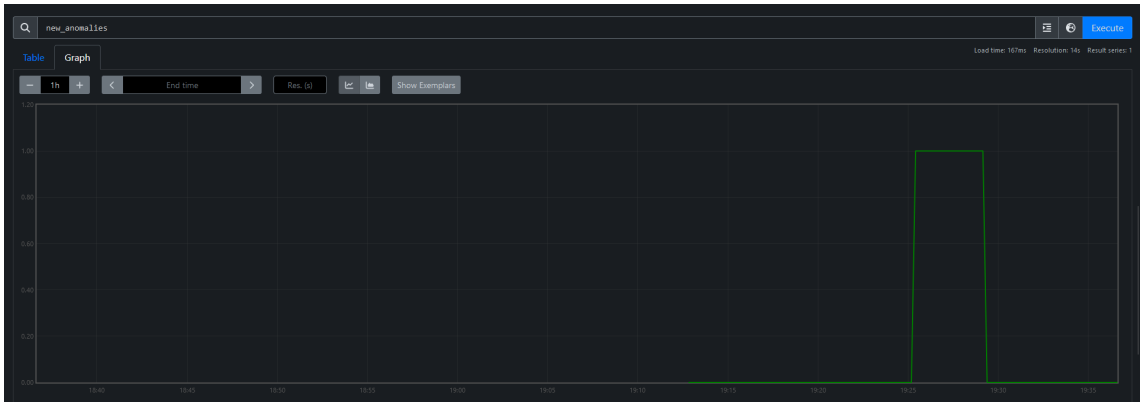

Figura 6. Período de surgimento de anomalias

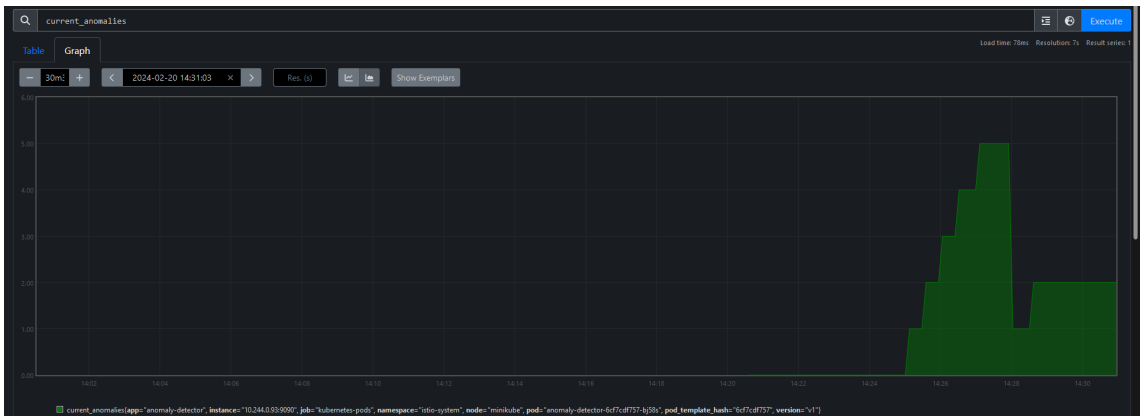

Figura 7. Comportamento da detecção de anomalias

4. Conclusão

É importante para os gerentes se assegurar que sua rede está funcionando de maneira satisfatória. O uso do KNN para a detecção de *outliers* é uma maneira prática de detectar comportamentos imprevistos, levando em consideração o histórico de funcionamento. Esses comportamentos podem ser investigados a fundo e então resolvidos pelos especialistas encarregados pela rede.

Mesmo sendo uma implementação simples, ela pode ser tornada mais acurada com mais tempo de execução em um ambiente normal, ou pode ter sua performance limitada, para a economia de memória e processamento. Além disso é possível facilmente aprimorar a aplicação desenvolvida para que ela utilize os dados já fornecidos pelo Prometheus para reportar detalhes em relação aos serviços que mais contribuíram com as anomalias detectadas, assim como utilizar alarmes para tal.

Referências

Istio. Bookinfo application. `https://istio.io/latest/docs/examples/bookinfo/`. [Online; acesso em 07-dezembro-2023].