Universidade Federal
do Rio de Janeiro
Escola Politécnica

ARQUITETURA RESILIENTE E ESCALÁVEL PARA UM SISTEMA DE EXECUÇÃO DE CÓDIGO REMOTO

Gabriel Monteiro de Castro Xará Wanderley

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Claudio Miceli de Farias
Co-orientadora: Laura de Oliveira Fernandes Moraes

Rio de Janeiro

Março de 2023

ARQUITETURA RESILIENTE E ESCALÁVEL PARA UM SISTEMA DE EXECUÇÃO DE CÓDIGO REMOTO

Gabriel Monteiro de Castro Xará Wanderley

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO

Autor:

Gabriel Monteiro de Castro Xará Wanderley

Orientador:

Prof. Claudio Miceli de Farias, D.Sc. UFRJ

Co-orientadora:

Prof. Laura de Oliveira Fernandes Moraes, D.Sc. UFRJ

Examinador:

Prof Geraldo Bonorino Xexéo, D. Sc. UFRJ

Examinador:

Prof. Toacy Cavalcante de Oliveira, PhD. Universidade de Waterloo

Rio de Janeiro – RJ, Brasil

Março de 2023

Declaração de Autoria e de Direitos

Eu, Gabriel Monteiro de Castro Xará Wanderley CPF 157.*513.317-29*, autor da monografia Proposta De Arquitetura Resiliente E Escalável Para Um Sistema Juiz Avaliador De Código, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.
2. Excetuam-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e idéias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.
3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.
4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.
5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.
6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.
7. Por ser verdade, firmo a presente declaração.

Gabriel Monteiro de Castro Xará Wanderley

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica – Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro – RJ CEP 21949-900

AGRADECIMENTO

Dedico este trabalho aos meus pais que desde meus primeiros anos sempre salientaram a importância da educação na formação do indivíduo. Dedico também aos contribuintes do povo brasileiro que possibilitaram minha permanência e formação nesta Universidade. Espero que este projeto possa de alguma forma retribuir o investimento e confiança em mim depositados.

RESUMO

A didática, parte da pedagogia que remete a técnicas de ensino, precisa ser constantemente adaptada e aprimorada na medida em que o contexto social e tecnológico evoluem. Uma vez que grandes inovações e transformações tecnológicas do mundo contemporâneo só são possíveis por conta do desenvolvimento de software, o campo de ensino das linguagens de programação é um ambiente perfeito para funcionar como "laboratório" no aprimoramento de recursos de didática. Neste trabalho foi criado um sistema executor de código, escalável e resiliente, cujo objetivo é tornar prático e acessível o ensino de programação a universitários da Escola Politécnica da Universidade Federal do Rio de Janeiro.

Palavras-chave: programação, orquestração de containers, sistema juiz, execução de código remoto

ABSTRACT

Didactics, part of pedagogy that refers to teaching techniques, needs to be constantly adapted and improved as the social and technological context evolves. Since major innovations and technological transformations in the contemporary world are only possible due to software development, the field of teaching programming languages is a perfect environment to behave as a “laboratory” in improving didactic resources. In this work, a scalable and resilient code execution system was created, whose objective is to make the teaching of programming practical and accessible to university students of the Polytechnic School of the Federal University of Rio de Janeiro.

Keywords: programming, container orchestration, judge system, remote code execution

SIGLAS

API Application Programming Interface
SAST Static Application Security Testing
UFRJ Universidade Federal do Rio de Janeiro
WSGI Web Server Gateway Interface

Sumário

1. Introdução

Sistemas juízes avaliadores de código (online judge services) são plataformas online onde usuários podem enviar códigos em resposta a desafios. Esse mecanismo de avaliação é bastante comum em desafios e competições de programação, mas também é amplamente utilizado no contexto de salas de aula.

Na Universidade Federal do Rio de Janeiro (UFRJ), uma parte significativa das disciplinas introdutórias de programação são ministradas com o auxílio do sistema juiz Machine Teaching [7]. O ambiente virtual de aprendizagem do Machine Teaching tem como objetivo suportar o aluno na prática de exercícios de programação e coletar os dados de progresso do aluno, de modo a ajudar os professores no acompanhamento e na tomada de decisão. A ferramenta se tornou de fundamental importância durante o período da pandemia de 2020, uma vez que as aulas precisaram ocorrer de maneira remota.

Conforme a adoção do Machine Teaching foi ganhando escala nas disciplinas de graduação, a equipe responsável pelo desenvolvimento identificou que o fato de o código estar sendo executado do lado do cliente (no caso, o navegador) torna o estudo suscetível a problemas de performance em decorrência do hardware à disposição do aluno. Em um cenário com execução de código do lado do servidor, o dispositivo utilizado, seja ele um notebook ou um smartphone, não seria mais um fator limitante para o aprendizado.

A proposta deste trabalho é, tendo o Machine Teaching como ponto de partida, implementar um sistema juiz escalável, seguro e replicável, com execução de código do lado do servidor. Este sistema, que será composto por alguns microsserviços operando de forma coordenada, processará o código enviado através do navegador do usuário, efetuará uma análise de segurança para garantir que não haja injeção de código malicioso e por fim executará o mesmo em um ambiente isolado (contêiner), devolvendo ao aluno ou competidor o resultado do processamento.

1.1 – Tema

O trabalho está inserido no contexto de orquestração de sistemas e execução de código remoto. A implementação proposta visa criar um ambiente resiliente para prática de exercícios de programação por estudantes universitários, aumentando a acessibilidade ao permitir que o estudo se dê a partir de qualquer dispositivo com conexão com a internet.

1.2 – Delimitação

O software a ser desenvolvido será capaz de executar apenas código escrito na linguagem de programação Python. O estudo será focado nos testes de escalabilidade e resiliência de um sistema remoto na execução de código durante o ensino de programação a alunos de graduação da Universidade Federal do Rio de Janeiro.

1.3 – Justificativa

A utilização de um sistema juiz como ferramenta pedagógica é um recurso que se mostra capaz de melhorar significativamente o desempenho dos alunos e o acompanhamento por parte dos professores. Conforme descrito em Moraes, Laura et al [8], isso se dá por alguns motivos, entre eles:

- Dados de desempenho: Sistemas juízes, em geral, armazenam dados relevantes, como o histórico de submissões de cada aluno, o tempo empreendido na resolução de um problema e quantidade de submissões com erro. Esse conjunto de dados se torna insumo para que o professor possa efetuar uma análise aprofundada a respeito da performance histórica de seus alunos, e a partir disso planejar ações e acompanhamentos. Esses dados também podem ser de grande valia para os próprios alunos, já que facilmente podem verificar sua evolução ao longo do curso e também reconhecer os módulos e exercícios que lhes trouxeram maior dificuldade.
- Acessibilidade: O fato de manter o processamento de código do lado do servidor faz com que a capacidade do hardware à disposição do aluno não seja mais um fator limitante para o seu aprendizado. Desta forma torna-se possível estudar e

resolver questões a partir de qualquer dispositivo com conectividade a internet, seja ele um computador ou mesmo um smartphone.

- Feedback rápido: Enquanto estuda e pratica o aluno recebe em tempo real o feedback a respeito de suas submissões. Esse fato torna o tempo de estudo mais eficiente e bem aproveitado, facilitando o aprendizado.

De acordo com o artigo [13], publicado no Simpósio Brasileiro de Informática na Educação-SBIE (2016), a utilização do sistema juíz CodeBench, desenvolvido como ferramenta de apoio para um ensino híbrido de programação, promoveu um aumento nas taxas de aprovação nos cursos introdutórios de programação da Universidade Federal do Amazonas (UFAM). O índice saltou de 50% para 70% de um ano para outro, corroborando para uma melhoria na dinâmica de ensino-aprendizagem.

Além das características e vantagens mencionadas acima, é importante entender quais são os sistemas juízes existentes e quais são os fatores e requisitos que tornam um sistema juiz adequado para uso, seja no contexto de salas de aula ou no contexto de competições de programação. Conforme a revisão sistemática de literatura publicada por Francisco, Júnior e Ambrósio [12], os requisitos funcionais podem ser classificados em quatro categorias: *feedback,* integração do sistema com os cursos, análise de desempenho geral dos alunos e a oferta de diferentes tipos de atividades. Já os requisitos não-funcionais são: integração, usabilidade, segurança, escalabilidade e disponibilidade.

Do ponto de vista de engenharia de software é importante compreender como os requisitos não funcionais podem ser fundamentais para viabilizar que sistemas que comprovadamente satisfazem os requisitos funcionais, ou seja, atendem as necessidades do usuário, possam ganhar escala.

Tendo os pontos acima em vista, abordaremos na próxima seção a proposta deste trabalho: a adaptação de um sistema juiz já utilizado em universidades para que o mesmo passe a executar o código do aluno não mais no navegador, mas sim em um servidor (cluster) cloud-based.

1.4 – Objetivos

Utilizar computação distribuída para criar um ambiente seguro e escalável para execução de código de alunos estudantes de programação. Desta forma, tem-se como objetivos específicos: (1) Viabilizar execução de código python de forma remota em um cluster de contêineres orquestrados pelo sistema de código aberto Kubernetes.; (2) Utilizar alguma solução existente para varredura do código fonte em busca de código malicioso ou fraquezas que possam a vir causar alguma vulnerabilidade. A solução a ser utilizada será uma das sugeridas pela OWASP (Open Web Application Security Project), organização sem fins lucrativos que trabalha para melhorar a segurança de software. A OWASP desenvolve e mantém vários recursos e ferramentas relacionados à segurança de aplicativos da Web, incluindo o OWASP Top 10, que é uma lista dos riscos de segurança de aplicativos da Web mais comuns.

1.5 – Descrição

No capítulo 2 será apresentada a fundamentação teórica necessária para que se discuta a implementação do sistema. Serão apresentados conceitos chave como containers, virtualização, princípios de orquestração de containers, Kubernetes e clusters de containers e testes de segurança em aplicativos estáticos.

O capítulo 3 apresenta a arquitetura da proposta de solução, onde será apresentado o passo a passo da construção do sistema, uma breve discussão de sua arquitetura e também os resultados obtidos.

Por fim, no capítulo 4, são feitas as considerações finais e alguns comentários sobre possíveis trabalhos futuros que poderiam contribuir como continuidade a esta pesquisa.

2. Fundamentação Teórica

2.1 – Sistema Juiz

Sistema Juiz é uma ferramenta de avaliação de código automatizada que verifica o código de um programador e fornece feedback sobre o mesmo. O feedback pode ser sobre a qualidade, sobre a eficiência ou apenas indicar se o código atende ou não a solução de um determinado problema.

Sistemas juízes são usados como ferramenta de treinamento para ajudar programadores e estudantes de programação a melhorar suas habilidades ou como uma ferramenta de avaliação em competições (no contexto de programação competitiva). O sistema Juiz avaliador de código pode ser configurado para avaliar código escrito em uma ampla variedade de linguagens de programação.

Alguns exemplos de sistemas juízes: Google Code Jam [1], CodePad [2], DMOJ [3], beecrowd [4].

2.2 – Containers (Virtualização)

Containers são uma alternativa cada vez mais utilizada para possibilitar que códigos de software sejam entregues de forma consistente e portátil. Na prática, aplicações conteinerizadas são isoladas juntamente com todo o ambiente computacional requerido para sua execução, que envolve: bibliotecas, frameworks e demais dependências.

O funcionamento de containers se dá através de um conceito chamado virtualização em nível de sistema operacional, recurso em que o kernel do sistema operacional permite a criação de múltiplos espaços de usuário isolados, cada um executado por um processo. Deste modo, cada ambiente gerado possui acesso aos recursos compartilhados pelo kernel com o processo correspondente. Para a aplicação que estiver sendo executada no container, o ambiente criado funciona como um computador real. Essa abordagem de virtualização a nível de sistema operacional é diferente da virtualização a nível de servidor, pois nesta segunda um hypervisor cria e executa “máquinas virtuais” que dependem de um sistema operacional (e muitas vezes de uma licença) para executar a aplicação. Não por acaso, máquinas virtuais

normalmente requerem significativamente mais espaço de armazenamento que containers.

Existem diferentes soluções que implementam containers, sendo que a predominante é a tecnologia Docker, desenvolvida pela empresa de mesmo nome *Docker Inc.*

2.3 - Orquestração

Para garantir alta disponibilidade em aplicações é preciso assegurar que sempre haverá um servidor disponível para atender requisições que porventura venham a ser realizadas. Essa garantia de disponibilidade exige que sejam implementados mecanismos de implantação, gerenciamento e distribuição de carga. Tais mecanismos, que serão brevemente comentados abaixo, são responsabilidade de um sistema orquestrador.

Gerenciamento: É preciso que seja monitorado o estado de cada servidor que executa a aplicação. Deste modo, caso algum incidente comprometa o correto funcionamento da aplicação, o orquestrador deve identificar o problema a fim de restabelecer o comportamento esperado.

Implantação: Seja por motivos de escalabilidade ou como forma de se recuperar de uma falha, o orquestrador precisa ser capaz de provisionar novas instâncias da aplicação.

Distribuição de carga: Uma vez que múltiplas instâncias estejam a executar o mesmo serviço é preciso que exista uma forma designar a instância responsável por atender cada nova requisição. Existem diferentes estratégias e algoritmos para balanceamento de carga, a depender do contexto e tipo de aplicação, cada estratégia pode ser mais ou menos indicada. Alguns exemplos são:

Round Robin: Requisições são distribuídas igualmente e de forma ordenada entre os servidores.

Fastest Response: Requisições serão encaminhadas ao servidor que tiver apresentado o menor tempo de resposta.

Fewest Servers: Requisições serão distribuídas de modo a minimizar a quantidade de instâncias

2.4 – Kubernetes

Conforme a execução de aplicações em ambientes conteinerizados começa a se tornar um padrão, torna-se cada vez mais complexo gerenciar diferentes serviços e aplicações de forma conteinerizada em uma única máquina hospedeira ou em um cluster de máquinas. Esse fator pode vir a dificultar significativamente o trabalho do responsável pela infraestrutura do sistema, pois o mesmo precisa garantir que durante todo o ciclo de vida da aplicação os serviços envolvidos estarão disponíveis.

Para simplificar tal tarefa, algumas soluções para orquestração de containers foram desenvolvidas (ver explicação sobre orquestração na seção 2.3). A solução mais utilizada no mercado é o Kubernetes, desenvolvido por engenheiros do Google no ano de 2014.

A proposta que diferencia uma solução de orquestração de contêineres como o Kubernetes é a de que

- Escalabilidade
- Resiliência
- Facilidade de manutenção

A arquitetura do Kubernetes é uma composição de múltiplos serviços com responsabilidades específicas. Esses serviços serão explicados de forma breve abaixo:

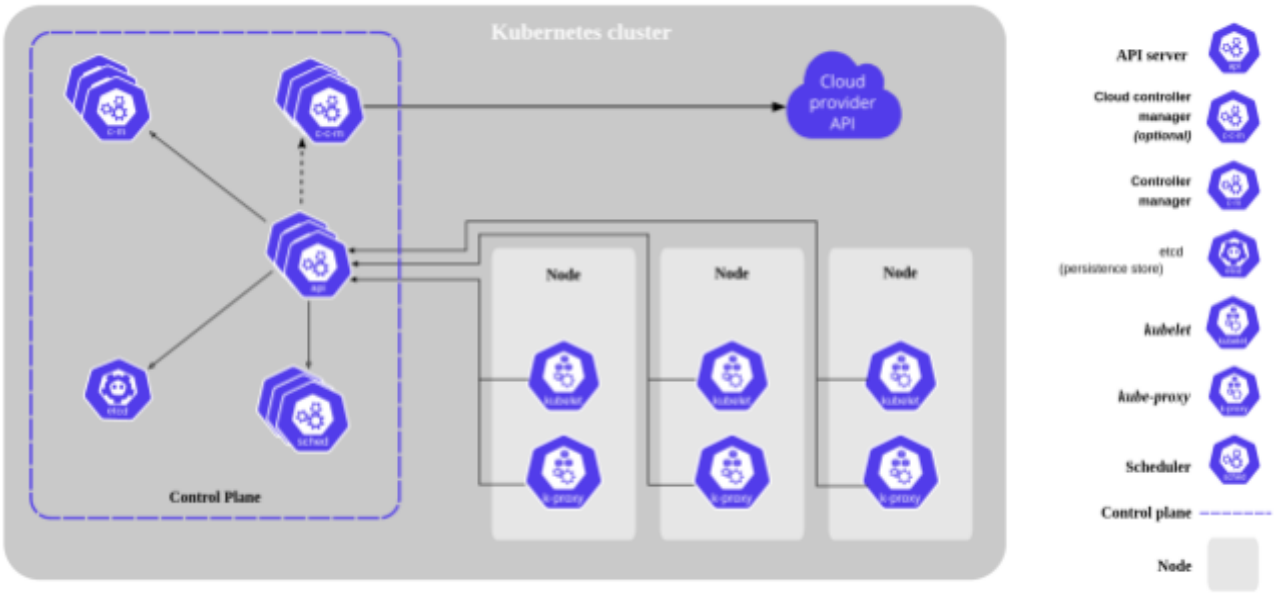

Figura 1: Diagrama com os componentes de um cluster Kubernetes.
Retirado de https://kubernetes.io/pt-br/docs/concepts/overview/components/

Todo cluster de kubernetes possui um ambiente de gerenciamento, denominado *control plane*, e pelo menos um servidor de processamento (*worker node*).
Cada nó de processamento hospeda um ou múltiplos *pods*, que funcionam como abstrações para containers Docker [10].
Já o ambiente de gerenciamento é composto por:

- kube-apiserver
- etcd
- kube-scheduler
- kube-controller-manager
- cloud-controller-manager

2.5 – Static application security testing (SAST)

Um dos fatores de risco de se disponibilizar um servidor para executar código de maneira remota é que o usuário pode, de forma intencional ou não, introduzir algum código malicioso. A execução de código malicioso pode vir a comprometer um ou mais dos 4 princípios básicos de segurança, sendo eles: disponibilidade, integridade, confidencialidade e autenticidade.

Existem algumas ferramentas e bibliotecas que servem para prevenir que códigos com vulnerabilidades ou códigos maliciosos venham a ser executados. Tais ferramentas são conhecidas como “Teste de segurança de aplicativo estático”, ou do inglês “Static application security testing”.

Uma vez que o objetivo deste projeto é criar um ambiente para execução remoto destinado a alunos de graduação torna-se de fundamental importância que uma solução de análise de código seja implementada, evitando que o servidor venha a ser comprometido de alguma forma.

Existem diversas soluções disponíveis na web, algumas de código aberto e outras de código proprietário. Para este projeto, optou-se pela utilização do software “bandit”, que por ser amplamente utilizado na comunidade de desenvolvedores Python já possui uma ampla gama de vulnerabilidades mapeadas.

3. Proposta de Solução

3.1 – Problema

O sistema Machine Teaching tem sido ferramenta valiosa na dinâmica de ensino-aprendizagem de programação para alunos da Universidade Federal do Rio de Janeiro e da Universidad de Alcalá.

Entretanto, um dos aspectos de sua arquitetura pode acabar se tornando fator impeditivo na execução dos exercícios: o processamento do código escrito pelo aluno ocorre no navegador do próprio aluno. O processamento estar do lado do cliente exige que o hardware utilizado atenda a requisitos mínimos para ser capaz de executar o código.

O problema que este projeto visa contornar é a criação de um ambiente remoto para execução de código, que permita aos alunos praticarem os exercícios a partir de qualquer dispositivo com conexão à internet. O tempo de processamento não será mais consequência do hardware utilizado pelo aluno, uma vez que um cluster de servidores estará dedicado ao processamento dos exercícios submetidos pelo Machine Teaching.

3.2 – Metodologia

Este trabalho está sendo desenvolvido em conjunto com a equipe responsável pelo Machine Teaching (www.machineteaching.tech), e a pretensão é que com esta adaptação a plataforma seja capaz de executar o código do aluno no servidor, e não mais no cliente (navegador). Para que isso seja concretizado é importante que exista uma infraestrutura segura, resiliente e escalável para suportar diferentes cargas de trabalho.

A solução proposta neste trabalho está indicada no diagrama da Figura 2. Para realizar a adaptação da plataforma será utilizado o sistema de orquestração de contêineres “Kubernetes”, de modo que cada contêiner do cluster se tornará um ambiente isolado para execução de código dos alunos e usuários do Machine Teaching. Após o término, será devolvido ao usuário o output gerado durante a execução.

Os fatores que fundamentam a decisão de utilizar Kubernetes são:

Balanceamento de carga

O Kubernetes pode balancear a carga e distribuir o tráfego de rede. Desta forma as submissões dos alunos serão direcionadas ao contêiner que estiver disponível ou, no caso de todos estarem ocupados, no primeiro que for liberado.

Definição de limites de CPU e memória

É possível definir limites de CPU e memória (RAM) para cada contêiner do cluster. Desta forma é possível impedir que uma carga de trabalho elevada, seja ela causada por um código malicioso ou por um código não performático, não venha a impactar o ambiente de execução dos demais alunos.

Autocorreção

O Kubernetes reinicia os contêineres que falham, substitui os contêineres, elimina os contêineres que não respondem à verificação de integridade definida pelo usuário e não os anuncia aos clientes até que estejam prontos para servir.

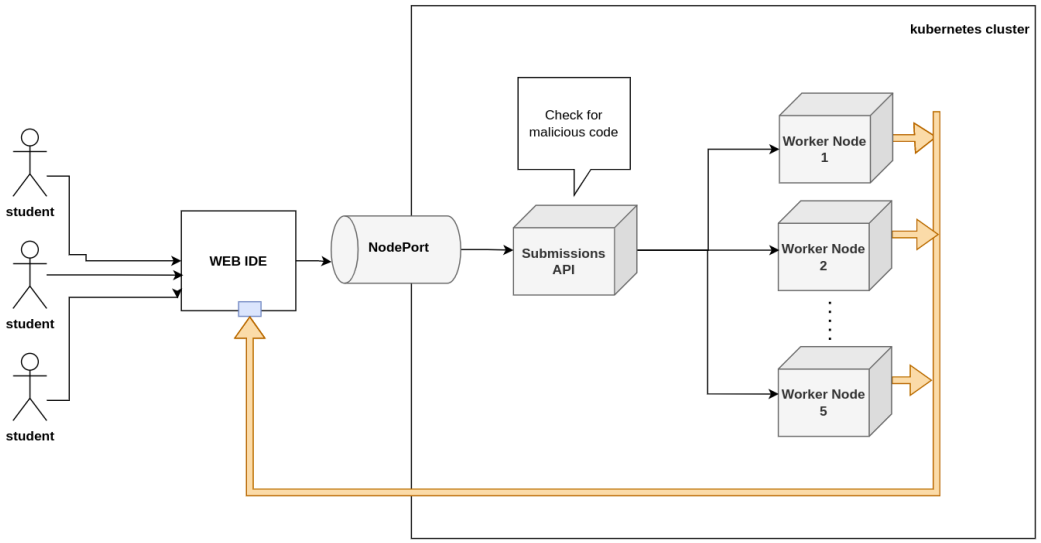

Figura 2 - Arquitetura do sistema proposto

Além do ambiente de execução de código dos alunos, algumas regras de negócio precisam ser implementadas para que o sistema possa se tornar seguro e cumpra seu

objetivo. Tais regras serão implementadas em uma API que atuará como proxy entre o cliente e o cluster kubernetes. Essa REST API, identificada como “Submissions API”, deverá ser uma interface que ao receber uma submissão de código efetuará uma varredura no código fonte em busca de código malicioso ou fraquezas que possam a vir causar alguma vulnerabilidade. Esse procedimento é conhecido como SAST (Static Application Security Testing) e existem diversas soluções amplamente utilizadas, tanto de código aberto quanto de código fechado.

Para este trabalho será utilizado o SAST bandit, capaz de encontrar vulnerabilidades comuns em códigos Python. Isso é feito através do processamento de cada arquivo individualmente. Ao final do processo, a ferramenta Bandit gera um relatório indicando a gravidade de cada vulnerabilidade encontrada.

A “Submission API” também será a aplicação responsável por receber, dos containers, o resultado da execução do código submetido pelos alunos. Após o recebimento, será feita a comparação entre o resultado esperado (gabarito da questão) e o resultado obtido (output do contêiner). A “Submission API” enviará para a aplicação WEB do machine teaching a “nota” atribuída à resolução do aluno

3.3 – Implementação

A solução idealizada consiste na criação de um sistema através do qual o usuário pode escrever o código de seu programa a partir de um navegador Web (normalmente presente na maioria dos dispositivos) e obter o output do programa sem que a execução do mesmo consuma recursos do hardware do usuário.

Desta forma, a implementação demandou a criação de dois serviços:

- Uma interface de desenvolvimento para que os alunos possam submeter códigos Python. Chamaremos este serviço de Web IDE.
- Um cluster com um ou mais nós para processamento de código remoto.

3.3.1 Preparação do Cluster

A aplicação requer que exista um cluster com alta disponibilidade. Isto é: todas as vezes em que um aluno submeter um código deve existir nó disponível para processamento. A solução adotada foi a implementação de um cluster kubernetes, cujo funcionamento foi melhor detalhado na seção de fundamentação teórica. Praticamente

todos os principais provedores de infraestrutura em nuvem disponibilizam a criação de clusters como serviço. O ambiente produtivo do projeto foi construído com a solução Google Kubernetes Engine (GKE), mas, para reproduzir de forma mais simples em ambiente local, pode-se replicar o processo com a criação de um cluster local utilizando o projeto de código aberto "minikube".

O passo a passo para instalação e inicialização de um cluster com minikube está disponível na documentação oficial através do endereço https://minikube.sigs.k8s.io/docs/start/.

Além do minikube, é necessária a instalação do *kubectl*, ferramenta de linha comando para interagir com o cluster e executar instruções. As instruções para instalação encontram-se em https://kubernetes.io/docs/tasks/tools/ e as etapas do processo podem variar de acordo com o sistema operacional do usuário. Os comandos digitados em linha de comando através do *kubectl* serão comentados abaixo. Cabe ressaltar que os mesmos são válidos independentemente de onde esteja rodando o cluster, ou seja, funcionam para um cluster local (com minikube) ou para um ambiente produtivo como GKE.

3.3.1.1 Criação de um *namespace* para o projeto

Os recursos definidos dentro de um cluster kubernetes são organizados em clusters virtuais. A utilização correta de *namespaces* é uma prática recomendada para garantir que além dos recursos de cada projeto estarem segregados também seja possível implementar diferentes políticas de permissionamento e de consumo por cluster.

Para criar um cluster dedicado ao projeto deste trabalho, o seguinte comando deve ser executado:

```
kubectl create namespace machine-teaching
```

3.3.1.2 Inicializando os serviços

As instruções para criação e configuração de recursos são passadas através do comando kubectl apply. Entretanto, normalmente as definições são longas e impraticáveis de serem repetidas várias vezes. Por isso, recomenda-se a utilização de arquivos para persistir a definição de cada recurso a ser criado no cluster. Esses arquivos possuem a

extensão ".yml", e podem ser versionados através de uma solução de versionamento de código como Github ou GitLab.
Os comandos abaixo foram executados para informar ao Kubernetes qual a configuração desejada para o projeto, sendo que cada um destes arquivos de configuração será explicado abaixo.

```
kubectl apply -f worker-configmap.yaml
kubectl apply -f svc-worker.yaml
kubectl apply -f worker-deployment.yaml
```

3.3.1.2.1 Configmaps

Configmaps são arquivos de configuração que definem um conjunto de pares chave-valor a serem armazenados no cluster de modo a serem aplicados nos objetos.

```
# worker-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: machine-teaching-worker-configmap
data:
  PORT: "5000"
```

3.3.1.2.2 Services

Services são recursos responsáveis por prover endereços IP para comunicação com os pods do cluster, além de realizarem o balanceamento de carga das requisições. Conforme explicado anteriormente na seção 2.3, o balanceamento pode ser feito com diferentes estratégias, como Round Robin e Fastest Response.
O trecho abaixo contém a definição do *service* declarado para o projeto.

```
# svc-worker.yaml
apiVersion: v1
kind: Service
metadata:
```

```
  name: svc-machine-teaching-worker
spec:
  type: NodePort
  ports:
    - port: 5000

  selector:
    app: machine-teaching-worker
```

3.3.1.2.3 Deployments

Deployments representam o estado desejável para os pods. Através da declaração do arquivo de deployments o cluster compara o estado atual com o estado desejável, de modo a tentar refletir este último.

Conforme definido no arquivo abaixo, são esperadas 5 réplicas do tipo machine-teaching-worker, que é o container responsável pela execução do código dos alunos. Este container é criado a partir de uma imagem Docker disponível no projeto machine-teaching-347613 do Google Container Registry (gcr.io). A aplicação escuta por requisições na porta 5000 e importa as variáveis de ambiente definidas em machine-teaching-worker-configmap (definido acima).

Enquanto o estado real do cluster for diferente do estado declarado no deployment o Kubernetes continuará tentando reproduzir o estado desejável de 5 pods saudáveis executando a aplicação. Para isso, são feitos health-checks (também chamados *probes)* constantemente. Esse processo consiste no envio de *liveness probes* para checar se a aplicação precisa ser reiniciada e *readiness probes* para checar se a aplicação está pronta para receber novas requisições.

```
# worker-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: machine-teaching-workers-deployment
  labels:
    app: machine-teaching-worker
spec:
  replicas: 5
  selector:
    matchLabels:
```

```
      app: machine-teaching-worker
  template:
    metadata:
      labels:
        app: machine-teaching-worker
    spec:
      containers:
        - name: machine-teaching-containers
                                                          image:
gcr.io/machine-teaching-347613/machine-teaching-worker:kuberne
tes
          imagePullPolicy: Always
          ports:
            - containerPort: 5000
          envFrom:
            - configMapRef:
                name: machine-teaching-worker-configmap
```

3.3.2 WEB IDE

Uma Web IDE (Interface de desenvolvimento integrado) é um ambiente que, de forma facilitada, permite ao usuário a codificação utilizando uma ou várias linguagens de programação.

Para que a Web IDE possa ser usada a partir de qualquer dispositivo o ideal é que esta possa ser acessada a partir de qualquer navegador Web conectado à internet. A solução desenvolvida foi construída com a utilização do framework React.js, uma biblioteca Javascript de código aberto criada em 2011 pelo Facebook e atualmente mantida por milhares de desenvolvedores ao redor do mundo.

Como parte do projeto, foi feito o desenvolvimento de uma Web IDE onde o usuário consegue submeter código Python para processamento. O código fonte da Web IDE foi disponibilizado no Github. Sua interface é bem simples, basicamente um editor de texto incorporado ao navegador e um botão para que o código implementado seja enviado ao cluster de servidores para processamento. O site encontra-se aberto, e pode ser acessado através do link https://mt-client-j54jk7yrsa-uc.a.run.app.

É importante destacar que por se tratar de um MVP a interface não pode ser equiparada às IDEs mais utilizadas pelos desenvolvedores. Esta primeira versão não

conta com recursos como destaque de sintaxe (*syntax highlighting*), preenchimento automático de texto (*autocomplete*) ou integração nativa com sistemas de versionamento. Tais melhorias são indicadas para trabalhos futuros que tenham por objetivo melhorar a ferramenta.

Figura 3 - Interface da Web IDE.

3.3.2.1 Submissão bem sucedida

Caso o código escrito pelo usuário seja processado com sucesso pelo servidor, uma mensagem é retornada, exibindo o resultado (*output),* o tempo total de processamento e um identificador único que indica qual foi a instância que realizou o processamento de código.

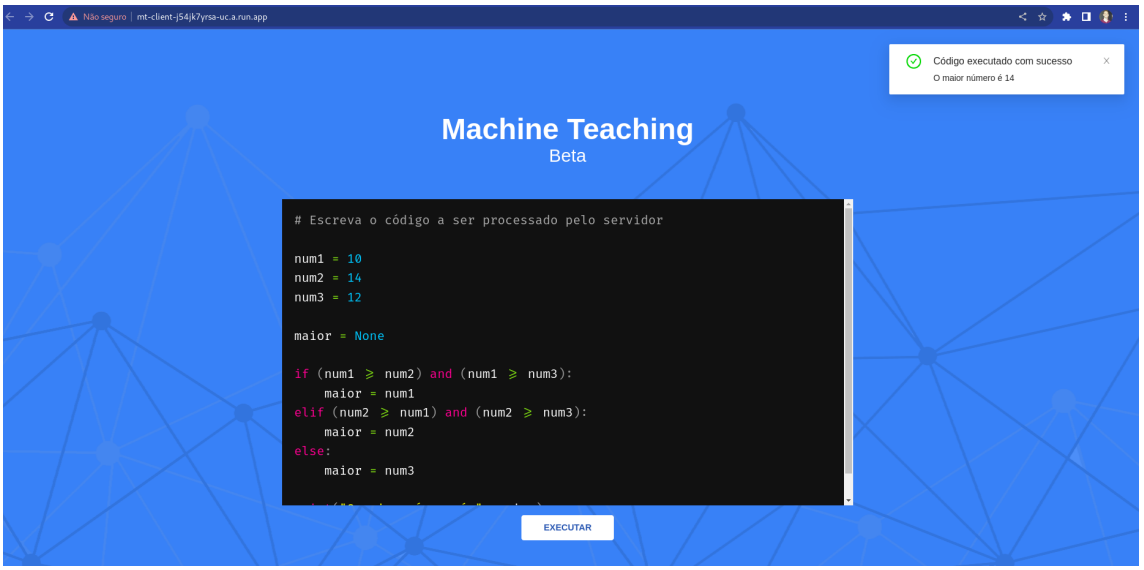

Figura 4 - Exemplo de submissão de código executado com sucesso.

3.3.2.1 Detecção de código malicioso

A funcionalidade de detecção de código malicioso previne que um usuário mal intencionado possa vir a causar danos ao servidor visando afetar sua disponibilidade. O sistema utiliza o SAST (Static Application Security Testing - ver seção 2.5) Bandit, solução de código aberto para projetos Python, disponível através do seguinte endereço https://bandit.readthedocs.io/en/latest/.

A imagem abaixo mostra que comandos potencialmente danosos são detectados e retornam uma mensagem de erro ao usuário.

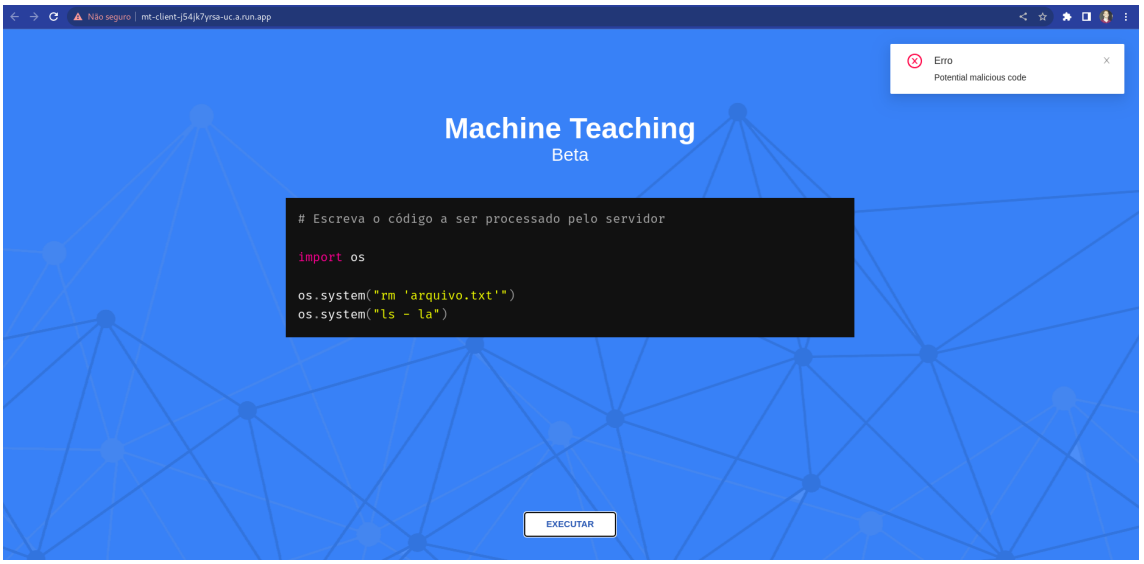

Figura 5 - Exemplo de submissão de código potencialmente perigoso.

3.4 – Resultados

A avaliação de resultados do sistema implementado foi realizada considerando dois aspectos principais: escalabilidade e segurança. A análise de segurança levou em conta os três pilares de segurança da informação, conforme BEAL [11], sendo eles: confidencialidade, integridade e disponibilidade. Já a análise de escalabilidade tem por objetivo estimar o número de usuários simultâneos que podem utilizar o sistema sem que seja apresentada lentidão na experiência de uso do sistema.

3.4.1 Avaliação de segurança

A confidencialidade é comprometida quando dados que deveriam ser acessados apenas por um conjunto definido de usuários são expostos de alguma maneira a usuários que extrapolam este conjunto.

A fim de garantir a **confidencialidade** dos dados, o sistema machine teaching deve impedir que um usuário do sistema tenha acesso a dados produzidos durante a sessão de outro usuário.

A implementação desenvolvida gera um diretório único para cada submissão de código realizada. Através da utilização do SAST (2.5), uma varredura de cada código submetido é realizada, impedindo que o mesmo seja executado caso alguma ameaça seja detectada. Com alguns testes foi possível verificar que operações de manipulação e leitura de arquivos são detectadas como potencialmente perigosas, logo, sendo impedidas de serem realizadas. Desta forma, durante os testes, não foi possível fazer com que através de um código malicioso um usuário viesse a conseguir acesso ao código submetido por outro usuário ou aos arquivos que por ele foram gerados.

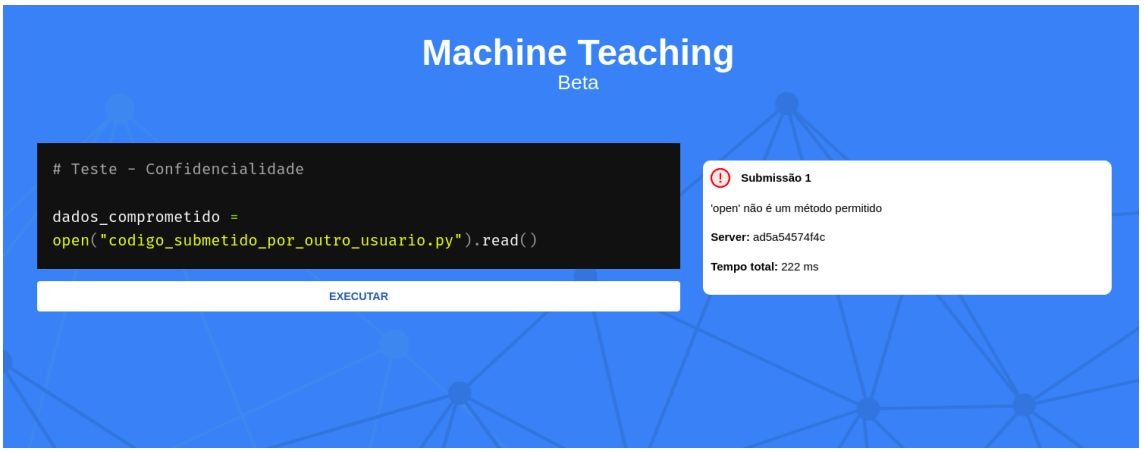

Figura 6 - Código submetido a fim de tentar violar a confidencialidade do sistema.

Considerou-se que a **integridade** do sistema seria comprometida caso um usuário ou agente malicioso conseguisse violar, de forma indevida, algum arquivo do sistema. Assim como demonstrado no teste de confidencialidade, não foi possível violar a integridade do sistema, uma vez que o mesmo detecta e impede qualquer tipo de manipulação de arquivo.

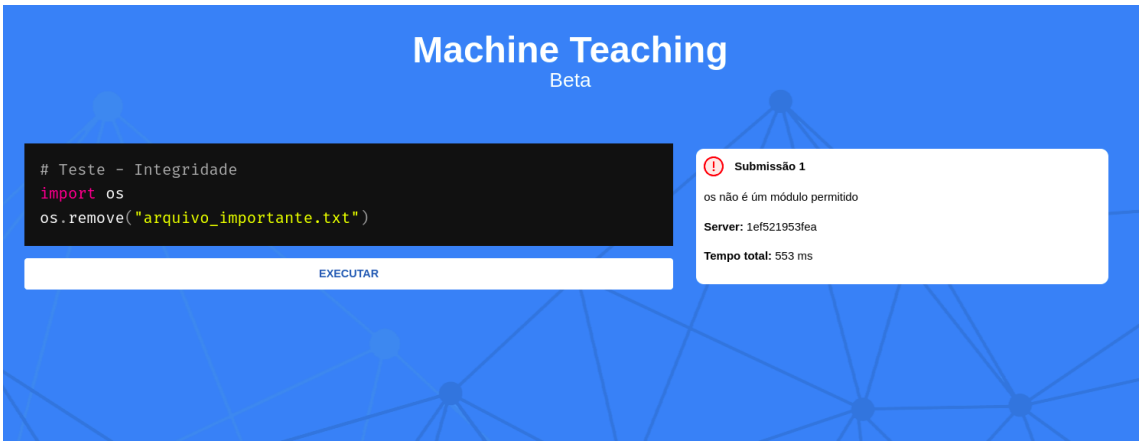

Figura 7 - Código submetido a fim de tentar violar a integridade do sistema.

Por fim, para testar a **disponibilidade** do sistema, foi feita a consideração de que a mesma seria comprometida caso alguma submissão de código pudesse tornar um dos servidores indisponível, afetando o processamento das submissões de outros usuários. Afetar a disponibilidade do sistema foi comprovadamente possível de forma bastante descomplicada, conforme exibido na imagem a seguir:

Figura 8 - Código submetido a fim de tentar violar a disponibilidade do sistema.

3.4.2 Avaliação de escalabilidade

A avaliação de escalabilidade e resiliência do sistema compreende estudar como o mesmo se comporta com múltiplos usuários, e quais são as limitações que podem se apresentar conforme o número de requisições simultâneas progride.

Conforme pode ser consultado no repositório da aplicação, cada código submetido é enviado como requisição a uma API construída com Flask (framework Python). Durante o processamento da requisição o código submetido é executado e o resultado é retornado para o cliente.

É importante destacar que em um único contêiner podem ser executadas múltiplas instâncias da mesma API. Isto é possível pois o WSGI (Web Server Gateway Interface) cria múltiplos *forks* do processo para que assim múltiplas requisições possam ser processadas de forma simultânea. A quantidade de processos é customizável e estabelecida com base na configuração do WSGI.

Os testes abaixo foram realizados considerando um cluster de servidores com um único nó de processamento, ou seja, uma única instância executando o contêiner machine-teaching-worker (ver 3.3.1.2.3). Foi feito um teste de carga para avaliar quantos usuários o sistema conseguiria atender **sem que o tempo de resposta ultrapassasse 1 segundo.** Utilizou-se a métrica de RPS (requisições por segundo) e foi feita a consideração de que cada usuário realizaria cerca de 0,3 requisições por segundo.

Nº de Réplicas (machine-teaching-workers-deployment)	Processos por réplica (contêiner)	Requisições por segundo (RPS)	Usuários simultâneos
1	1	5	17
1	3	15	50
1	5	18	60

Tabela 1 - Resultados obtidos com a variação das configurações do sistema

Sabendo que cada réplica funciona de maneira isolada e tendo em vista que o sistema apresentou melhor performance com processos por réplica, podemos estabelecer a seguinte relação:

$$RPS = 18\ x\ nº\ de\ réplicas$$

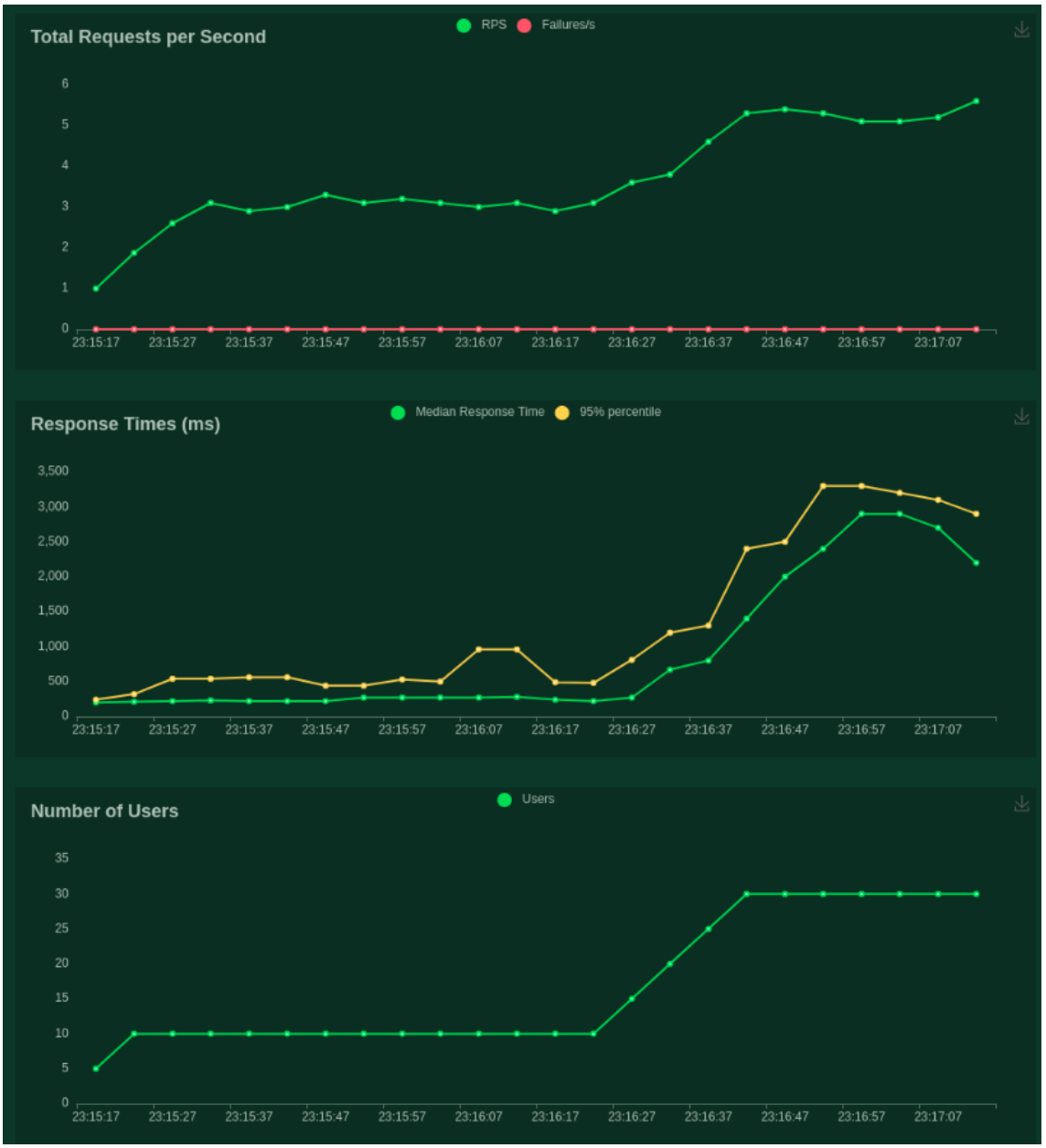

Figura 9 - Métricas obtidas com a utilização de um único worker

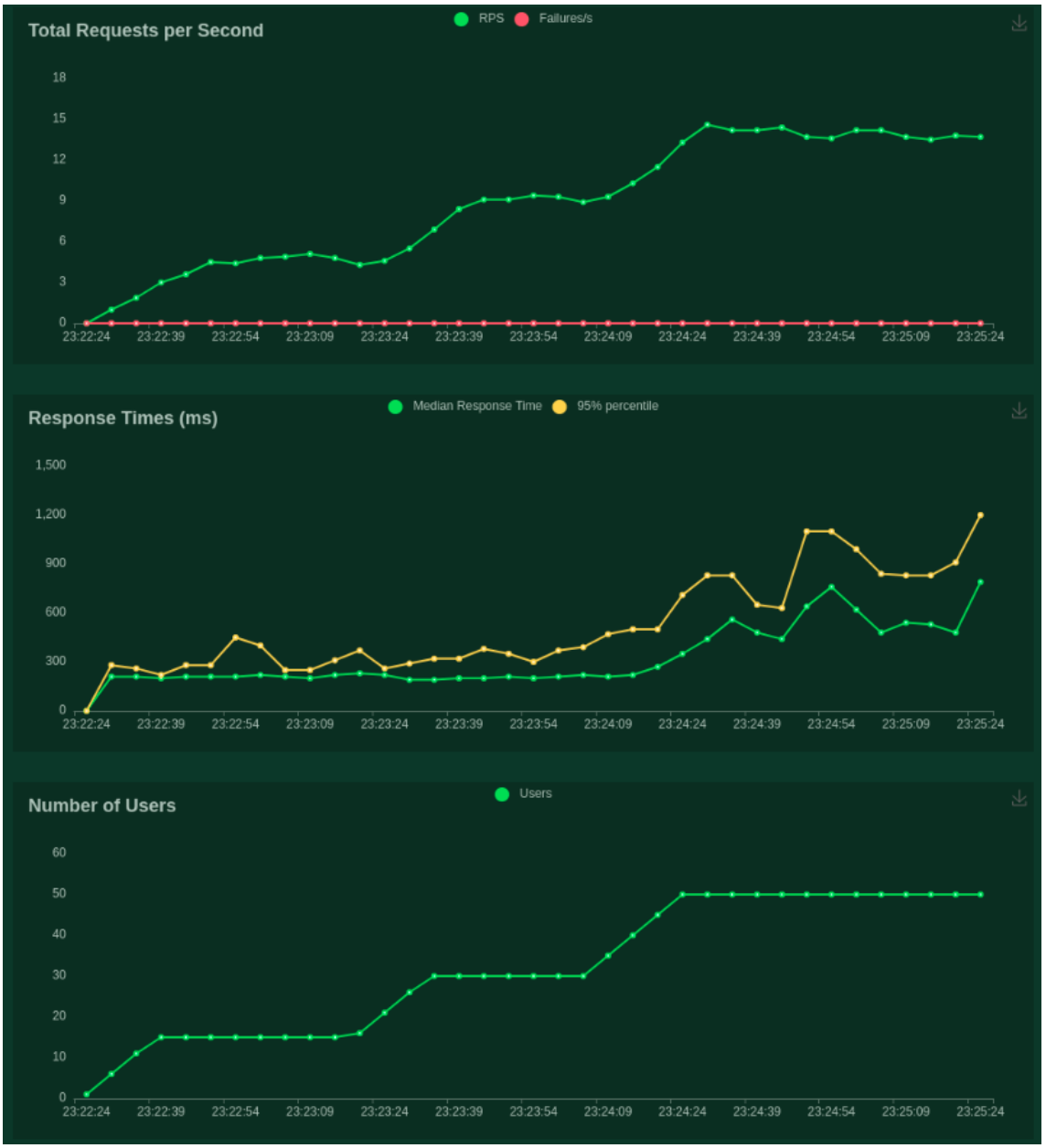

Figura 10 - Métricas obtidas com a utilização de 3 workers

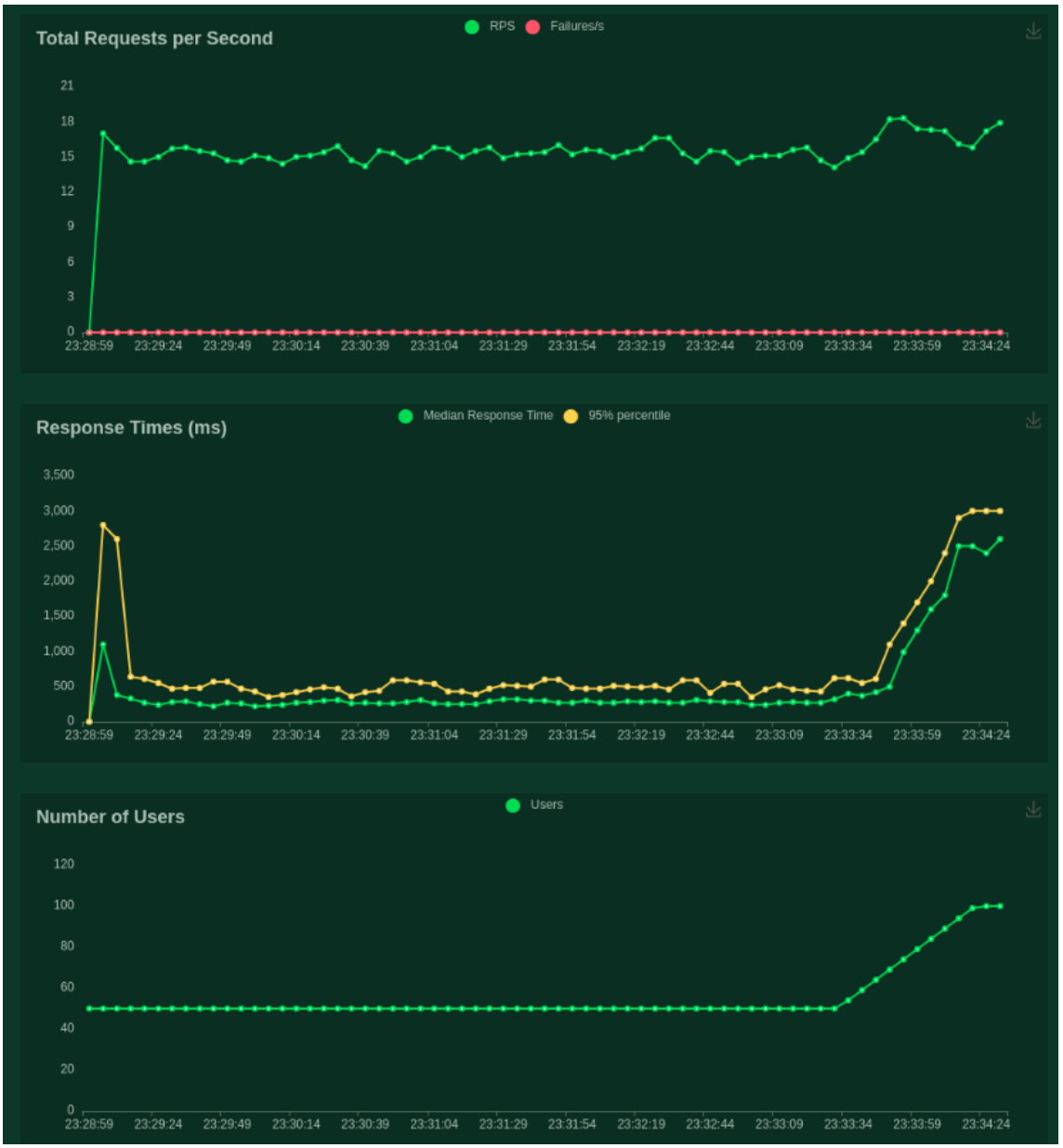

Figura 10 - Métricas obtidas com a utilização de 5 workers

4. Considerações Finais

4.1 – Conclusões

O sistema Machine Teaching, já amplamente utilizado nos cursos de graduação da Universidade Federal do Rio de Janeiro e da Universidade de Alcalá, tem apresentado bons resultados conforme [8].

A proposta deste trabalho cumpriu o objetivo de implementar, como mínimo produto viável, uma variação que possa vir a se tornar uma alternativa para que ferramentas educacionais como o Machine Teaching ou sistemas juízes (programação competitiva) possam executar código remoto de forma escalável, mesmo com um número elevado de usuários.

A utilização da ferramenta Kubernetes como orquestrador de containers mostra que as diferentes técnicas de Load Balancing aliadas aos mecanismos de *liveness probes e readiness probes* permitem ao cluster escalar horizontalmente o número de containers, visando manter a estabilidade do *deployment.*

A solução, disponibilizada como código aberto, pode ser implementada em ambientes produtivos tanto em serviços de *cloud* quanto em servidores *on-premise,* ficando a critério da instituição de ensino que quiser oferecer o ambiente para execução de código para seus alunos.

É importante destacar que ambientes compartilhados para execução de código tendem a ser potenciais alvos de ataques maliciosos. A utilização de um SAST pode ajudar a minimizar os riscos, uma vez que algumas categorias de ataques são mais facilmente identificados. Entretanto, um grande número de vulnerabilidades ainda podem ser exploradas, como, por exemplo, códigos que intencionalmente ou não afetem a disponibilidade do sistema através de chamadas bloqueantes executadas por longos períodos de tempo.

Por fim, espera-se que como ambiente de execução de código remoto, este trabalho possa tornar a prática de programação mais acessível àqueles que não possuem equipamentos convencionais de programação, como computadores e notebooks. Possibilitando inclusive a prática de exercícios e a execução de trabalhos a partir de smartphones com conexão à internet.

4.2 – Trabalhos Futuros

Este trabalho foi desenvolvido como um MVP (produto mínimo viável, do inglês *minimum viable product*) que permitisse a estudantes de programação praticarem a codificação a partir de dispositivos de baixo poder computacional, através de um sistema escalável e resiliente de processamento de código remoto. Para que o projeto possa ser utilizado de forma mais ampla como ferramenta de ensino ou como sistema juiz em competições de programação, seriam necessárias algumas adaptações e melhorias para tornar o mesmo mais completo.

Aqui serão mencionadas algumas propostas de continuidade ao trabalho implementado.

4.2.1 – Adaptação do sistema para que atue como sistema juiz

Sistemas juízes avaliadores de código são amplamente utilizados em competições de programação para avaliar as submissões dos candidatos, atribuindo uma nota ou apenas classificando a submissão como correta ou incorreta.

Um possível trabalho futuro seria estender o comportamento do sistema desenvolvido para que o mesmo, além de processar o código submetido, seja capaz de avaliar, sob critérios qualitativos, a qualidade e a corretude do código enviado pelo aluno ou competidor.

Para implementar um sistema juiz se faz necessário que o sistema esteja integrado a um banco de dados que armazene todos os exercícios e as respostas corretas para cada.

4.2.2 – Novas Linguagens de Programação

As disciplinas de introdução a computação das universidades que utilizam o sistema Machine Teaching são ensinadas com a utilização da linguagem de programação Python.

Um possível trabalho futuro seria adaptar o projeto para que alunos e competidores que utilizam o sistema possam praticar e submeter respostas para exercícios com diferentes linguagens de programação.

Uma vez que todo o processamento se dá no lado do servidor, nenhuma mudança seria necessária no lado do cliente. Entretanto, é necessário que o cluster de servidores seja adaptado para compilar e executar códigos escritos em diferentes linguagens.

Bibliografia

[1] GOOGLE. Google Code Jam: Competição de programação organizada pela Google. Disponível para programadores de todo o mundo. Consiste na resolução de desafios através da implementação de algoritmos. Disponível em: <https://codingcompetitions.withgoogle.com/codejam/>. Acesso em: 06 de junho de 2022.

[2] Steven Hazel. CodePad: Site web que permite execução e compilação de código em diferentes linguagens. Disponível em: <http://codepad.org/>. Acesso em: 06 de junho de 2022.

[3] Steven Hazel. DMOJ: Sistema juiz de código aberto. Sedia diversas competições de programação. Possui mais de 125 mil usuários cadastrados. Disponível em: <https://dmoj.ca/>. Acesso em: 06 de junho de 2022.

[4] BEECROWD. beecrowd: Sistema juiz de código fechado. Sedia diversas competições de programação. Também possibilita que professores criem exercícios e desafios para seus alunos. Disponível em: <https://www.beecrowd.com.br/judge/pt/login>. Acesso em: 06 de junho de 2022.

[5] Michal Forišek. 2006b. Security of programming contest systems. Information Technologies at School (2006), 553–563.
https://people.ksp.sk/~misof/publications/copy/2006attacks.pdf

[6] https://owasp.org/www-community/Source_Code_Analysis_Tools

[7] UFRJ. Maching Teaching: Sistema juiz avaliador de código utilizado como recurso didático na Universidade Federal do Rio de Janeiro. Disponível em: <http://www.machineteaching.tech/pt-br/>. Acesso em: 06 de junho de 2022.

[8] Moraes, Laura O., et al. "Machine Teaching: uma ferramenta didática e de análise de dados para suporte a cursos introdutórios de programação." Anais do II Simpósio Brasileiro de Educação em Computação. SBC, 2022.

[9] Hogg, Scott (26 de maio de 2014). «Software Containers: Used More Frequently than Most Realize». Network World. Network World, Inc. Consultado em 9 de julho de 2015. There are many other OS-level virtualization systems such as: Linux OpenVZ, Linux-VServer, FreeBSD Jails, Workload Partitions (WPARs), HP-UX Containers (SRP), Solaris Containers, among others.AIX

[10] C. Anderson, "Docker [Software engineering]," in IEEE Software, vol. 32, no. 3, pp. 102-c3, May-June 2015, doi: 10.1109/MS.2015.62.

[11] BEAL, Adriana. Segurança da Informação: princípios e melhores práticas para a proteção dos ativos de informação nas organizações - São Paulo: Atlas, 2005.

[12] FRANCISCO, Rodrigo; JÚNIOR, Cleon Pereira; AMBRÓSIO, Ana Paula. Juiz online no ensino de programação introdutória-uma revisão sistemática da literatura. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2016. p. 11.

[13] GALVÃO, Leandro; FERNANDES, David; GADELHA, Bruno. Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2016. p. 140.