Tolerância a falhas bizantinas de aplicações provisionadas localmente com Kubernetes: uma proposta inicial

Leonardo Valério Anastácio[1], **Guilherme Piêgas Koslovski**[1]

[1]Programa de Pós-Graduação em Computação Aplicada
Universidade do Estado de Santa Catarina – UDESC

Resumo. *Executar soluções conteinerizadas em clusters Kubernetes locais representa um desafio para as empresas, e há diversas motivações por trás dessa escolha, desde requisitos legais até a redução de custos. No entanto, manter a mesma qualidade de serviço dos principais provedores de nuvem de forma local exige esforço tanto na engenharia de infraestrutura quanto no desenvolvimento. Nesse contexto, o trabalho atual propõe o desenvolvimento de um framework para gerenciamento de tolerância a falhas bizantinas, com o objetivo de apoiar implantações on-premise do Kubernetes.*

1. Introdução

Kubernetes é considerado um padrão do mercado para orquestração de contêineres [Domingus and Arundel 2022], oferecendo uma arquitetura modular e escalável para implantar, dimensionar e gerenciar aplicações [Bernstein 2014]. Dessa forma, *Kubernetes* é utilizado por diversas empresas para orquestrar contêineres em *data centers* (DC) locais, que não possuem indicadores de confiabilidade, resiliência e robustez comparáveis, em termos de escala, com DCs de nuvens computacionais.

Um dos pontos centrais acerca do funcionamento distribuído do *Kubernetes* está na utilização do algoritmo *Raft* Em DCs localmente provisionados, alguns exemplos de comportamentos bizantinos são: mensagens atrasadas, corrompidas ou processos executados que apresentam comportamentos fora da normalidade. Essa classe de problemas torna o *Raft* vulnerável, uma vez que falhas bizantinas acontecem com frequência em sistemas distribuídos impactando diretamente a confiabilidade geral do sistema e o desempenho das aplicações.

2. Impacto da ocorrência de Falhas Bizantinas no *Kubernetes*

Ao ser submetido a falhas bizantinas o *Kubernetes* sofre com imprecisões nos estados replicados dos contêineres, assim, comprometendo a disponibilidade das aplicações em execução no *cluster*. Sabendo que o protocolo de replicação é utilizado nos componentes do *Plane Control* do *Kubernetes*, isto é, os componentes responsáveis pelo funcionamento interno, as possíveis implicações das falhas bizantinas em ambiente *Kubernetes* são [Diouf et al. 2020]: *(i)* Inconsistência: Falhas bizantinas resultam na propagação de informação de estado incorreta para todo o sistema distribuído, acarretando em inconsistências e imprecisões nos estados replicados. *(ii)* Interrupção de serviços: A natureza bizantina é propícia a interromper a operação normal do protocolo de replicação, causando interrupções nos serviços impactando na disponibilidade das aplicações. *(iii)* Riscos de segurança: Falhas bizantinas representam riscos de segurança ao manipular de forma não autorizada e insegura estados replicados. Assim, compromete a integridade e confidencialidade dos dados do *Kubernetes*.

A literatura avançou em aspectos importantes em relação ao tratamento de falhas bizantinas em clusters *Kubernetes*, com destaque a resiliência do *Control Plane* mesmo sob condições bizantinas [Diouf et al. 2020]. Contudo, esses estudos estão focados nos componentes internos do *Kubernetes* e não nas aplicações em execução em si.

3. Proposta inicial

A literatura já explorou o tratamento de falhas bizantinas em alguns dos componentes internos do *Kubernetes*. Entretanto, as próprias aplicações são potenciais causadores de falhas bizantinas, muitas vezes devido a problemas de implementação. A presente proposta visa a construção de um *framework* de observabilidade e prevenção de falhas bizantinas detectáveis. O objetivo é tolerar comportamentos bizantinos que sejam previamente declarados com auxílio de gatilhos, que serão baseados em informações de normalidade da aplicação especificada pelo usuário anteriormente. Uma das maneiras de trabalhar nesse cenário está na criação de uma linguagem ubíqua declarativa que armazenará as condições normais da aplicação, condições de sobrecarga e os gatilhos necessários pra cada situação. Por exemplo, em um cenário em que a aplicação está gerando mais dados do que o habitual no banco de dados, uma solução ou abordagem seria isolar a comunicação com o banco de dados até que a situação se normalize (mediante outro gatilho) ou reiniciar o contêiner responsável pela aplicação. Por outro lado, essa situação pode ser provocada por um agente externo, como um usuário que esteja enviando uma quantidade anormal de informações para a aplicação. Uma possível solução seria bloquear a comunicação com esse usuário mal-intencionado, o que poderia ser interpretado como um tipo de gatilho.

O funcionamento do *framework* consistirá em 3 passos principais: Coleta de dados, Análise e Ação. Os dados poderão ser coletados de um repositório de monitoramento como *Prometheus* ou diretamente da aplicação com integração via *HTTP*. A partir dos dados coletados será feita uma análise específica para cada tipo de gatilho visando comportamentos bizantinos previamente declarados. Por fim, a ação surtirá alguma correção em relação a falha identificada, por exemplo, uma chamada corretiva enviada para *API* do *Kubernetes*. Além disso, o *framework* será implantado no formato de operador assim como outros serviços sofisticados construídos especificamente para o *Kubernetes*.

A principal ideia do *framework* é tratar os problemas bizantinos de forma simples e declarativa, por meio de uma camada alto nível que os desenvolvedores possam prevenir erros bizantinos identificáveis de forma elegante, sem interferir na implementação das regras de negócio da aplicação. Com essa abordagem, será possível a utilização do *framework* de prevenção bizantina independente de linguagens e tecnologias auxiliando soluções conteinerizadas.

Referências

Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

Diouf, G. M., Elbiaze, H., and Jaafar, W. (2020). On byzantine fault tolerance in multi-master kubernetes clusters. *Future Generation Computer Systems*, 109:407–419.

Domingus, J. and Arundel, J. (2022). *Cloud native devops with kubernetes: Building, deploying, and scaling modern applications in the cloud*. O'Reilly.