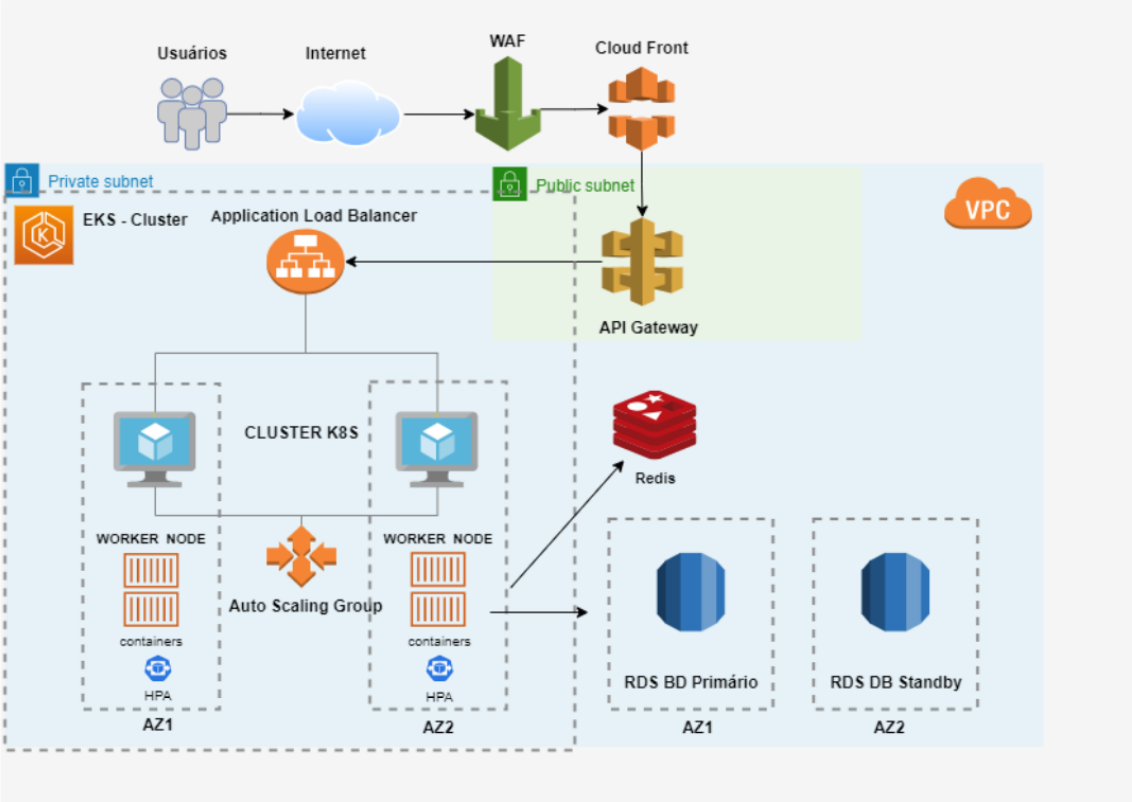
Arquitetura



Justificativa da arquitetura:

WAF: adicionar segurança a ataques ex: ddos

Cloud Front: fazer um cache nos edge points da amazona mais próximos do cliente

API Gateway: o api gateway seria a nossa parte exposta que redirecionaria as requests para nosso loadbalancer, talvez numa arquitetura mais simples poderíamos utilizar outras tratativas como o Istio Service mesh para fazer essa função de ingress.

Application Load Balancer: ponto de entrada do nosso service dentro do cluster Kubernetes

Arquitetura Kubernetes escolhida: EKS – Serviço da AWS em que o master node é serverless, a amazon garante a alta disponibilidade dele.

Auto Scaling group: Função de escalar nossos workers nodes baseados em métricas do cloudwatch ex: uso de cpu / ram, evitando uso de instancias desnecessárias e também melhorando a performance da aplicação caso necessário.

Redis: Utilizado para o cache para solucionar o problema de indisponibilidade devido as queries repetidasm neste caso coloquei o serviço serverless da AWS.

RDS: como nossa aplicação é um banco de dados relacional optei pelo RDS que é um banco serverless da AWS o qual podemos configurar como multi-az dependendo da necessidade, e é garantido que temos sempre uma instancia em standby para caso de problemas.

Dentro do Kubernetes: além dos pods da aplicação, adicionaria o prometheus e o grafana para cuidar do monitoramento, o Kubernetes já possui um auto-scaling integrado chamado HPA porém caso escalar baseado em métricas como cpu e ram não forem o suficiente estudaria o uso do KEDA para basear o scale em coisas como tamanho de fila, pra aumentar o nível de segurança poderíamos adicionar o istio service mesh para as comunicações leste-oeste.

Como faria o CI/CD

Em relação ao CI/CD para aplicação eu faria os seguintes passos:

1. Adicionar a aplicação a um repositório GIT
2. Deixar o Dockerfile dentro do meu repositório
3. Utilizar um trigger como um commit no código para rodar o pipeline
4. Adicionaria um teste unitário como um Ruby Linter e um Yaml validator
5. Adicionaria ao menos uma validação de qualidade do código como, utilizando métricas no Quality Gate.
6. Após passar nos testes neste pipeline faria a build do container
7. Faria o deployment remoto no cluster Kubernetes
8. Adicionaria um teste de performance como Taurus ou Jmeter
9. Adicionaria o OWASP ZAP para fazer testes de vulnerabilidade
10. Caso não passar nos dois últimos testes faria uma logica de rollback para a versão estável da imagem, poderia ser executar o comando kubectl remoto para a imagem mais estável da aplicação.