Projetando para HA e resiliência em aplicativos Amazon EKS

AWS Recomendações

AWS Recomendações: Projetando para HA e resiliência em aplicativos Amazon EKS

Table of Contents

Projetando para alta disponibilidade e resiliência em aplicativos Amazon EKS

Haofei Feng, Frank Fan e Rus Kalakutskiy, da Amazon Web Services ()AWS

Outubro de 2025 (histórico do documento)

Garantir alta disponibilidade (HA) e resiliência no design do aplicativo é crucial para alcançar objetivos de ponto de recuperação (RPO) e objetivos de tempo de recuperação (RTO) quase nulos. À medida que as organizações migram e modernizam cada vez mais seus aplicativos para ambientes Kubernetes, a demanda por soluções robustas e escaláveis continua aumentando. O Amazon Elastic Kubernetes Service (Amazon EKS) ajuda você a gerenciar com eficiência aplicativos em contêineres em grande escala.

Este guia aborda um conjunto de recomendações e melhores práticas amplamente reconhecidas para projetar e gerenciar aplicativos de microsserviços do Amazon EKS. Com base em ampla experiência e implantações no mundo real, esses insights oferecem orientação valiosa para arquitetos e desenvolvedores. Implemente essas recomendações para obter alto desempenho, confiabilidade e escalabilidade de seus aplicativos baseados em Kubernetes para obter operações robustas.

Considerações sobre design de alta disponibilidade e resiliência

O modelo de responsabilidade compartilhada se torna mais complexo com o Kubernetes. A disponibilidade e a resiliência do plano de controle do Amazon EKS são gerenciadas pela Amazon Web Services (AWS). Sua organização gerencia o plano de dados, o que pode afetar significativamente o desempenho e a disponibilidade de seus aplicativos de microsserviços.

Ao criar um aplicativo altamente disponível e resiliente no Amazon EKS, considere os seguintes componentes:

- O aplicativo de microsserviços: seus pods e contêineres
- O plano de dados da carga de trabalho: controlador de entrada, pod, componentes do sistema, como a interface de rede de contêineres (CNI) da Amazon Virtual Private Cloud (Amazon VPC), sidecars de service mesh e kube-proxy
- A camada de gerenciamento da carga de trabalho: controladores, controladores de admissão, mecanismos de política de rede e armazenamento persistente de dados para esses componentes
- O plano de controle do Kubernetes
- Infraestrutura: nós, rede e dispositivos de rede

Para as três primeiras considerações, que se referem aos componentes que são executados em um cluster Kubernetes, este guia aborda os seguintes tópicos:

- Distribuindo cargas de trabalho entre nós e zonas de disponibilidade
- Protegendo cargas de trabalho críticas com um PDB
- Configurando sondas e verificações de saúde
- Configurando ganchos do ciclo de vida do contêiner
- Entendendo o despejo de lagoas durante interrupções zonais

Distribua cargas de trabalho entre nós e zonas de disponibilidade

A distribuição de uma carga de trabalho em domínios de falha, como zonas de disponibilidade e nós, melhora a disponibilidade dos componentes e diminui as chances de falha em aplicativos escaláveis

horizontalmente. As seções a seguir apresentam formas de distribuir cargas de trabalho entre nós e zonas de disponibilidade.

Use restrições de dispersão da topologia do pod

As restrições de dispersão da topologia do pod do Kubernetes instruem o programador do Kubernetes a distribuir pods que são gerenciados por `ReplicaSet` ou `StatefulSet` entre diferentes domínios de falha (zonas de disponibilidade, nós e tipos de hardware). Ao usar restrições de dispersão da topologia do pod, você pode fazer o seguinte:

- Distribua ou concentre os pods em diferentes domínios de falha, dependendo dos requisitos do aplicativo. Por exemplo, você pode distribuir pods para resiliência e concentrar pods para desempenho de rede.
- Combine condições diferentes, como distribuição entre zonas de disponibilidade e distribuição entre nós.
- Especifique a ação preferida se as condições não puderem ser atendidas:
  - Use `whenUnsatisfiable: DoNotSchedule` com uma combinação de `maxSkew` e `minDomains` para criar requisitos rígidos para o agendador.
  - Use `whenUnsatisfiable: ScheduleAnyway` para reduzir`maxSkew`.

Se uma zona de falha ficar indisponível, os pods dessa zona ficarão insalubres. O Kubernetes reprograma os pods enquanto segue a restrição de propagação, se possível.

O código a seguir mostra um exemplo do uso de restrições de distribuição da topologia de pod entre zonas de disponibilidade ou entre nós:

```
...
spec:
  selector:
    matchLabels:
      app: <your-app-label>
    replicas: 3
    template:
      metadata:
        labels: <your-app-label>
      spec:
        serviceAccountName: <ServiceAccountName>
...
        topologySpreadConstraints:
```

```
        - labelSelector:
            matchLabels:
              app: <your-app-label>
          maxSkew: 1
          topologyKey: topology.kubernetes.io/zone # <---spread those pods evenly over
 all availability zones
          whenUnsatisfiable: ScheduleAnyway
        - labelSelector:
            matchLabels:
              app: <your-app-label>
          maxSkew: 1
          topologyKey: kubernetes.io/hostname # <---spread those pods evenly over all
 nodes
          whenUnsatisfiable: ScheduleAnyway
```

Restrições de dispersão da topologia padrão em todo o cluster

Por padrão, o Kubernetes fornece um conjunto de restrições de distribuição de topologia para distribuição de pods entre nós e zonas de disponibilidade:

```
defaultConstraints:
  - maxSkew: 3
    topologyKey: "kubernetes.io/hostname"
    whenUnsatisfiable: ScheduleAnyway
  - maxSkew: 5
    topologyKey: "topology.kubernetes.io/zone"
    whenUnsatisfiable: ScheduleAnyway
```

> **Note**
>
> Aplicativos que precisam de diferentes tipos de restrições de topologia podem substituir a política em nível de cluster.

As restrições padrão definem um valor alto`maxSkew`, o que não é útil para implantações com um pequeno número de pods. No momento, não `KubeSchedulerConfiguration` pode ser alterado no Amazon EKS. Se você precisar aplicar outros conjuntos de restrições de dispersão de topologia, considere usar o controlador de admissão mutante, como na seção abaixo. Você também pode controlar as restrições de dispersão da topologia padrão se você executar um agendador alternativo. No entanto, o gerenciamento de agendadores personalizados aumenta a complexidade e pode ter

implicações na resiliência do cluster e na HA. Por esses motivos, não recomendamos o uso de um programador alternativo somente para restrições de dispersão de topologia.

A política do Gatekeeper para restrições de dispersão de topologia

Outra opção para impor restrições de dispersão de topologia é usar uma política do projeto Gatekeeper. As políticas do Gatekeeper são definidas no nível do aplicativo.

Os exemplos de código a seguir mostram o uso de uma `Gatekeeper OPA` política para implantação. Você pode modificar a política de acordo com suas necessidades. Por exemplo, aplique a política somente às implantações que tenham o rótulo `HA=true` ou escreva uma política semelhante usando um controlador de política diferente.

Este primeiro exemplo mostra o `ConstraintTemplate` uso com`k8stopologyspreadrequired_template.yml`:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8stopologyspreadrequired
spec:
  crd:
    spec:
      names:
        kind: K8sTopologySpreadRequired
      validation:
        openAPIV3Schema:
          type: object
          properties:
            message:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8stopologyspreadrequired

        get_message(parameters, _default) =3D msg {
          not parameters.message
          msg :=_default
        }


        get_message(parameters, _default) =3D msg {
```

```
          msg := parameters.message
        }

        violation[{"msg": msg}] {
          input.review.kind.kind ="Deployment"
          not input.review.object.spec.template.spec.topologySpreadConstraint
          def_msg :"Pod Topology Spread Constraints are required for Deployments"
          msg :get_message(input.parameters, def_msg)

        }
```

O código a seguir mostra o `k8stopologyspreadrequired_constraint.yml` manifesto `constraints` YAML:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sTopologySpreadRequired
metadata:
  name: require-topologyspread-for-deployments
spec:
  match:
    kinds:
      - apiGroups: ["apps"]
        kinds: ["Deployment"]
    namespaces:  ## Without theses two lines will apply to the whole cluster
      - "example"
```

Quando usar restrições de dispersão de topologia

Considere o uso de restrições de dispersão de topologia para os seguintes cenários:

- Qualquer aplicativo escalável horizontalmente (por exemplo, serviços web sem estado)
- Aplicativos com réplicas ativo-ativas ou ativas-passivas (por exemplo, bancos de dados NoSQL ou caches)
- Aplicativos com réplicas em espera (por exemplo, controladores)

Os componentes do sistema que podem ser usados para o cenário escalável horizontalmente, por exemplo, incluem o seguinte:

- Cluster Autoscaler e Karpenter (com e) `replicaCount > 1` `leader-elect = true`
- AWS Controlador de balanceador de carga

- CoreDNS

Afinidade e antiafinidade do pod

Em alguns casos, é vantajoso garantir que não mais do que um pod de um tipo específico esteja sendo executado em um nó. Por exemplo, para evitar o agendamento de vários pods com muita rede no mesmo nó, você pode usar a regra de antiafinidade com o rótulo ou. `Ingress` `Network-heavy` Ao usar`anti-affinity`, você também pode usar uma combinação do seguinte:

- Manchas em nós otimizados para rede
- Tolerâncias correspondentes em cápsulas com muita rede
- Afinidade de nós ou seletor de nós para garantir que pods com muita rede usem instâncias otimizadas para rede

Network-heavy os frutos são usados como exemplo. Você pode ter requisitos diferentes, como GPU, memória ou armazenamento local. Para ver outros exemplos de uso e opções de configuração, consulte a documentação do Kubernetes.

Rebalancear cápsulas

Esta seção discute duas abordagens para rebalancear pods em um cluster Kubernetes. O primeiro usa o Descheduler for Kubernetes. O Descheduler ajuda a manter a distribuição dos pods aplicando estratégias para remover os pods que violam as restrições de dispersão da topologia ou as regras de antiafinidade. A segunda abordagem usa o recurso de consolidação e empacotamento do Karpenter. A consolidação avalia e otimiza continuamente o uso de recursos consolidando cargas de trabalho em menos nós compactados com mais eficiência.

Recomendamos usar o Descheduler se você não estiver usando o Karpenter. Se você estiver usando o Karpenter e o Autoescalador de Cluster juntos, poderá usar o Desagendador com o Autoescalador de Cluster para grupos de nós.

Desagendador para nós sem grupos

Não há garantia de que as restrições de topologia permaneçam satisfeitas quando os pods são removidos. Por exemplo, reduzir uma implantação pode resultar em uma distribuição desequilibrada de pods. No entanto, como o Kubernetes usa restrições de dispersão da topologia do pod somente no estágio de agendamento, os pods ficam desbalanceados em todo o domínio da falha.

Para manter uma distribuição equilibrada de pods nesses cenários, você pode usar o Descheduler for Kubernetes. O Descheduler é uma ferramenta útil para vários propósitos, como impor a idade máxima do pod ou o tempo de vida (TTL) ou para melhorar o uso da infraestrutura. No contexto de resiliência e alta disponibilidade (HA), considere as seguintes estratégias do Descheduler:

- RemovePodsViolatingTopologySpreadConstraint
- RemovePodsViolatingInterPodAntiAffinity
- RemoveDuplicates

Recurso de consolidação e empacotamento de caixas do Karpenter

Para cargas de trabalho que usam o Karpenter, você pode usar a funcionalidade de consolidação e empacotamento para otimizar a utilização de recursos e reduzir custos em clusters Kubernetes. O Karpenter avalia continuamente o posicionamento dos pods e a utilização dos nós e tenta consolidar as cargas de trabalho em menos nós compactados com mais eficiência, sempre que possível. Esse processo envolve analisar os requisitos de recursos, considerar restrições como regras de afinidade de pods e potencialmente mover pods entre os nós para melhorar a eficiência geral do cluster. O código a seguir fornece um exemplo:

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  disruption:
    consolidationPolicy: WhenUnderutilized
    expireAfter: 720h
```

Para`consolidationPolicy`, você pode usar `WhenUnderutilized` ou`WhenEmpty`:

- Quando `consolidationPolicy` está definido como`WhenUnderutilized`, o Karpenter considera todos os nós para consolidação. Quando o Karpenter descobre um nó vazio ou subutilizado, o Karpenter tenta remover ou substituir o nó para reduzir o custo.
- Quando `consolidationPolicy` está definido como`WhenEmpty`, o Karpenter considera para consolidação somente nós que não contêm pods de carga de trabalho.

As decisões de consolidação do Karpenter não se baseiam apenas nas porcentagens de utilização da CPU ou da memória que você pode ver nas ferramentas de monitoramento. Em vez disso, o

Karpenter usa um algoritmo mais complexo baseado em solicitações de recursos do pod e possíveis otimizações de custos. Para obter mais informações, consulte a documentação do Karpenter.

Proteja cargas de trabalho críticas com um PDB

Um orçamento de interrupção de pod (PDB) é um recurso essencial para manter a alta disponibilidade dos aplicativos em um cluster. O PDB especifica um tamanho alvo, que é a disponibilidade mínima para um tipo específico de pod. Isso significa que um número mínimo de réplicas de um determinado tipo de pod deve estar em execução a qualquer momento. Se o número de réplicas em execução ficar abaixo do tamanho desejado, o Kubernetes evitará mais interrupções nas réplicas restantes até que o tamanho desejado seja atingido. Os PDBs ajudam a garantir que as cargas de trabalho não sejam afetadas por esses eventos e possam continuar sendo executadas sem interrupções. Quando ocorre uma interrupção, o Kubernetes tenta expulsar os pods dos nós afetados com facilidade, mantendo o número de réplicas especificado no PDB.

Você pode usar um PDB para declarar o `maxUnavailable` número `minAvailable` e o número de réplicas. Por exemplo, se você quiser que pelo menos três cópias do seu aplicativo estejam disponíveis, crie um PDB semelhante ao exemplo a seguir:

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: my-svc-pdb
spec:
  minAvailable: 3
  selector:
    matchLabels:
      app: my-svc
```

Configurar PDBs corretamente para seus aplicativos ajuda a minimizar a interrupção durante eventos planejados ou não planejados. Você pode usar a regra de antiafinidade para programar os pods de uma implantação em nós diferentes e evitar atrasos no PDB durante as atualizações dos nós.

Configurar sondas e verificações de integridade do balanceador de carga

O Kubernetes fornece várias maneiras de realizar verificações de integridade do aplicativo, além das verificações de integridade do balanceador de carga. Você pode executar os seguintes testes

integrados do Kubernetes junto com a verificação de integridade do balanceador de carga como um comando no contexto do pod ou como um teste para o kubelet ou o HTTP/TCP endereço IP do host.

As sondas de vivacidade e prontidão devem ser diferentes e independentes (ou, pelo menos, com valores de tempo limite diferentes). Se um aplicativo tiver um problema temporário, a sondagem de prontidão marcará o pod como não pronto até que o problema seja resolvido. Se as configurações da sonda de atividade não estiverem corretas, a sonda de atividade poderá encerrar a cápsula.

Sonda de inicialização

Use sondas de inicialização para proteger aplicativos com ciclos de inicialização longos. Até que a sondagem de inicialização seja bem-sucedida, as outras sondas serão desativadas.

Você pode definir o tempo máximo que o Kubernetes deve esperar pela inicialização do aplicativo. Se, após o tempo máximo configurado, o pod ainda falhar nos testes de inicialização, o aplicativo será encerrado e um novo pod será criado.

Use sondas de inicialização quando o tempo de inicialização de um aplicativo for imprevisível. Se você sabe que seu aplicativo precisa de 10 segundos para ser iniciado, use uma sonda de vivacidade ou uma sonda de prontidão. `initialDelaySeconds`

Sonda de vivacidade

Use sondas de atividade para detectar problemas no aplicativo ou se o processo está sendo executado sem problemas. Uma sonda de atividade pode detectar condições de impasse em que o processo continua em execução, mas o aplicativo deixa de responder. Ao usar uma sonda de vivacidade, faça o seguinte:

- Use `initialDelaySeconds` para atrasar a primeira sonda.
- Não defina a mesma especificação para sondas de vivacidade e prontidão.
- Não configure uma sonda de atividade para depender de um fator externo ao seu pod (por exemplo, um banco de dados).
- Defina a sonda de vivacidade para um específico. `terminationGracePeriodSeconds` Para obter mais informações, consulte a documentação do Kubernetes.

Sonda de prontidão

Use uma sonda de prontidão para detectar o seguinte:

- Se o aplicativo está pronto para aceitar tráfego
- Disponibilidade parcial, em que o aplicativo pode estar temporariamente indisponível, mas espera-se que volte a funcionar após a conclusão de uma determinada operação

Os testes de prontidão ajudam a garantir que a configuração e as dependências do aplicativo sejam executadas sem problemas ou erros, para que o aplicativo possa atender ao tráfego. No entanto, uma sonda de prontidão mal configurada pode causar uma interrupção em vez de evitá-la. Sondas de prontidão que dependem de fatores externos, como conectividade com um banco de dados, podem fazer com que todos os pods falhem na sondagem. Essas falhas podem resultar em uma interrupção e levar a uma falha em cascata de um serviço de back-end para outros serviços que usaram os pods com falha.

Verificações de integridade do recurso de entrada e do balanceador de carga

O Application Load Balancer e o Kubernetes `ingress` fornecem recursos de verificação de integridade. Para as verificações de integridade do Application Load Balancer, especifique as portas e o caminho de destino.

> **Note**
>
> Para o Kubernetes`ingress`, haverá uma latência de cancelamento de registro. O padrão para o Application Load Balancer é 300 segundos. Considere configurar seu recurso de entrada ou a verificação de integridade do balanceador de carga usando os mesmos valores que você usou para sua análise de prontidão.

O NGINX também fornece uma verificação de integridade. Para obter mais informações, consulte a documentação do NGINX.

Os gateways de entrada e saída do Istio não têm um mecanismo de verificação de integridade comparável à verificação de integridade HTTP do NGINX. No entanto, você pode obter uma funcionalidade semelhante usando o disjuntor ou a detecção de `DestinationRule` valores discrepantes do Istio.

Para obter mais informações, consulte Disponibilidade e ciclo de vida do pod no Guia de melhores práticas do Amazon EKS.

Configurar ganchos do ciclo de vida do contêiner

Durante um desligamento normal do contêiner, seu aplicativo deve responder a um `SIGTERM` sinal iniciando o desligamento para que os clientes não tenham nenhum tempo de inatividade. Seu aplicativo deve executar procedimentos de limpeza como os seguintes:

- Salvando dados
- Fechando descritores de arquivo
- Fechando conexões de banco de dados
- Concluindo solicitações em voo com elegância
- Saindo em tempo hábil para atender à solicitação de encerramento do pod

Defina um período de carência que seja longo o suficiente para que a limpeza termine. Para saber como responder ao `SIGTERM` sinal, consulte a documentação da linguagem de programação que você usa para seu aplicativo.

Os ganchos do ciclo de vida do contêiner permitem que os contêineres estejam cientes dos eventos em seu ciclo de vida de gerenciamento. Os contêineres podem executar código implementado em um manipulador quando o gancho de ciclo de vida correspondente é executado. Os ganchos do ciclo de vida do contêiner fornecem uma solução alternativa para a natureza assíncrona do Kubernetes e da nuvem. Essa abordagem pode evitar a perda de conexões que são encaminhadas para o pod de encerramento antes do recurso de entrada e `iptables` são atualizadas para não enviar novo tráfego para o pod.

O ciclo de vida do contêiner e `EndpointSlice` fazem parte de diferentes APIs. `Endpoint` É importante orquestrar essas APIs. No entanto, quando um pod está sendo encerrado, a API Kubernetes notifica simultaneamente o kubelet (para o ciclo de vida do contêiner) e o controlador. `EndpointSlice` Para obter mais informações, incluindo um diagrama, consulte Lidar com elegância com as solicitações do cliente no Guia de melhores práticas do Amazon EKS.

Quando `kubelet` envia `SIGTERM` para o pod, o `EndpointSlice` controlador está encerrando o `EndpointSlice` objeto. Esse encerramento notifica os servidores da API Kubernetes para notificarem cada nó a ser `kube-proxy` atualizado. `iptables` Embora essas ações ocorram ao mesmo tempo, não há dependências ou sequências entre elas. Há uma grande chance de o contêiner receber o `SIGKILL` sinal muito antes de cada nó atualizar as `iptables` regras locais. `kube-proxy` Nesse caso, os cenários possíveis incluem o seguinte:

- Se sua solicitação cancelar imediatamente e sem rodeios as solicitações e conexões em voo após o recebimento`SIGTERM`, os clientes verão erros. `500`
- Se seu aplicativo garantir que todas as solicitações e conexões em andamento sejam processadas completamente após o recebimento`SIGTERM`, durante o período de carência, as novas solicitações do cliente ainda serão enviadas ao contêiner do aplicativo, pois `iptables` as regras talvez ainda não tenham sido atualizadas. Até que o procedimento de limpeza feche o soquete do servidor no contêiner, essas novas solicitações resultarão em novas conexões. Quando o período de carência termina, as novas conexões estabelecidas após o `SIGTERM` envio são descartadas incondicionalmente.

Para abordar os cenários anteriores, você pode implementar a integração no aplicativo ou o gancho do PreStop ciclo de vida. Para obter mais informações, incluindo um diagrama, consulte Encerrar aplicativos normalmente no Guia de melhores práticas do Amazon EKS.

> **Note**
>
> Independentemente de o aplicativo ser encerrado normalmente ou do resultado de um problema, os contêineres do `preStop` aplicativo acabam sendo encerrados no final do período de carência. `SIGKILL`

Use o `preStop` gancho com um `sleep` comando para atrasar o envio`SIGTERM`. Isso ajudará a continuar aceitando as novas conexões enquanto o objeto de entrada as encaminha para o pod. Teste o valor temporal do `sleep` comando para garantir que qualquer latência do Kubernetes e de outras dependências do aplicativo seja levada em consideração, conforme mostrado no exemplo a seguir:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      lifecycle:
        # This "sleep" preStop hook delays the Pod shutdown until
        # after the Ingress Controller removes the matching Endpoint or EndpointSlice
        preStop:
          exec:
```

```
            command:
              - /bin/sleep
              - "20"
              # This period should be turned to Ingress/Service Mesh update latency
```

Para obter mais informações, consulte Container hooks na documentação do Kubernetes e desligar aplicativos normalmente no Amazon EKS Best Practices Guide.

Entenda o despejo de lagoas durante interrupções zonais

Quando ocorre uma interrupção total da zona de disponibilidade, ou seja, quando todos os nós dessa zona de disponibilidade perdem a conectividade com o plano de controle do Kubernetes, o controlador do ciclo de vida do nó no Kubernetes detecta a situação e expulsa os pods da zona afetada. Os pods em nós inacessíveis são marcados como `Terminating` e novos pods são programados em nós saudáveis nas zonas de disponibilidade disponíveis. Durante esse período, os nós afetados exibem um `NotReady` status, o agendador impede que novos pods sejam colocados nesses nós e o EndpointSlice controlador remove os endpoints associados à Zona de Disponibilidade prejudicada do roteamento do serviço até que a conectividade seja restaurada.

Para cenários que envolvem falhas parciais de nós em uma zona, em que somente um subconjunto de nós se torna inacessível, o controlador do ciclo de vida do nó aplica diferentes comportamentos de despejo. Se a interrupção persistir além do período de tolerância configurado (por padrão, cinco minutos), os pods nos nós desconectados serão marcados como `Terminating` e os novos pods serão programados nos nós íntegros nas zonas de disponibilidade disponíveis.

Implementando a mudança de zona do Amazon EKS para melhorar a resiliência

A mudança de zona do Amazon EKS, que se integra ao Amazon Application Recovery Controller (ARC), fornece um mecanismo para gerenciar proativamente o tráfego durante deficiências na zona de disponibilidade. Esse recurso permite o redirecionamento temporário do tráfego de rede de uma zona de disponibilidade insalubre para zonas íntegras dentro da mesma, a fim de minimizar Região da AWS a interrupção do serviço.

Compreendendo o mecanismo de mudança zonal

A mudança de zona do Amazon EKS aborda o tráfego leste-oeste (comunicação entre pods dentro do cluster). Quando a mudança de zona é configurada com Application Load Balancers ou Network Load Balancers, ela também oferece suporte ao roteamento de tráfego de entrada. O

mecanismo opera coordenando vários componentes do Kubernetes e do plano de AWS controle para redirecionar o tráfego com segurança sem interromper as cargas de trabalho em execução. Durante uma mudança de zona ativa, o Amazon EKS executa automaticamente as seguintes ações coordenadas:

- Isolamento de nós: todos os nós na Zona de Disponibilidade comprometida são isolados. Isso impede que o programador do Kubernetes coloque novos pods nos nós enquanto mantém as cargas de trabalho existentes.
- Suspensão do rebalanceamento da zona de disponibilidade: para grupos de nós gerenciados, as operações de rebalanceamento da zona de disponibilidade são suspensas e os grupos do Auto Scaling são atualizados para iniciar novos nós do plano de dados exclusivamente em zonas de disponibilidade saudáveis. Isso garante que a nova capacidade não seja provisionada na zona prejudicada.
- Remoção do endpoint: o EndpointSlice controlador remove os endpoints do pod na zona de disponibilidade prejudicada de todos os itens relevantes. EndpointSlices Isso garante que os mecanismos de descoberta de serviços e balanceamento de carga direcionem o tráfego somente para pods que estão sendo executados em zonas de disponibilidade saudáveis.
- Preservação da carga de trabalho: o Amazon EKS se abstém de encerrar nós ou remover pods na zona de disponibilidade afetada. Ele mantém a capacidade total na zona prejudicada para que, quando a mudança zonal expirar ou for cancelada, o tráfego possa retornar com segurança sem exigir operações adicionais de escalonamento.

Métodos de ativação por mudança zonal

Você pode escolher entre duas abordagens para iniciar mudanças zonais, dependendo do seu modelo operacional:

- O deslocamento zonal manual fornece controle orientado pelo operador quando problemas específicos da zona de disponibilidade são detectados por meio de monitoramento, alertas ou relatórios de clientes. Esse método requer ação explícita por meio do console ARC, AWS Command Line Interface (AWS CLI) ou de APIs de mudança zonal, nas quais os operadores especificam a zona de disponibilidade prejudicada e definem um prazo de expiração para o turno. Os turnos manuais são apropriados quando as equipes têm recursos dedicados de monitoramento e plantão e preferem manter o controle direto sobre as decisões de gerenciamento de tráfego.
- O deslocamento automático zonal AWS autoriza o início automático de turnos quando o ARC detecta possíveis falhas ou deficiências na zona de disponibilidade com base em telemetria

interna e sinais de saúde em vários, Serviços da AWS incluindo métricas de rede, Amazon Elastic Compute Cloud (Amazon EC2) e Elastic Load Balancing. AWS encerra automaticamente uma mudança automática quando os indicadores mostram que o problema foi resolvido. Se você deseja a postura de maior disponibilidade com o mínimo de intervenção manual, recomendamos essa abordagem, pois ela permite uma resposta em menos de um minuto às deficiências detectadas na Zona de Disponibilidade.

Pré-requisitos para uma mudança zonal efetiva

Para que a mudança zonal proteja com sucesso os aplicativos durante deficiências na Zona de Disponibilidade, você deve arquitetar seus clusters para obter Multi-AZ resiliência antes de ativar o recurso de mudança zonal:

- Multi-AZ distribuição de nós: provisione nós de trabalho em pelo menos três zonas de disponibilidade para garantir redundância suficiente quando uma zona ficar indisponível.
- Planejamento de capacidade: capacidade computacional Pre-provision suficiente em zonas de disponibilidade saudáveis para acomodar toda a carga de trabalho quando uma zona de disponibilidade é removida do serviço, pois as operações de escalabilidade durante uma interrupção ativa podem encontrar capacidade insuficiente.
- Distribuição e pré-escalonamento de pods: implante várias réplicas de cada aplicativo em todas as zonas de disponibilidade e pré-escale componentes críticos do sistema, como o CoreDNS, em todas as zonas. Isso ajuda a garantir que a capacidade suficiente permaneça após a mudança de uma zona.

Recomendações para resiliência à interrupção zonal

- Habilite a mudança zonal na criação do cluster: Para novos clusters EKS, habilite a integração do deslocamento zonal com o ARC durante o provisionamento inicial por meio do console Amazon EKS ou de ferramentas de infraestrutura como código (IaC) AWS CLI, como. AWS CloudFormation Os clusters do EKS Auto Mode criados com configuração rápida têm a mudança de zona ativada por padrão.
- Selecione o método de ativação apropriado: escolha o deslocamento automático zonal para ambientes de produção que exigem disponibilidade máxima com resposta automatizada, especialmente para aplicativos voltados para o cliente, nos quais minutos de inatividade durante um comprometimento da zona de disponibilidade podem ter um impacto significativo nos negócios. Use a mudança de zona manual para ambientes em que as equipes de operações preferem

fornecer aprovação explícita antes das mudanças de tráfego ou onde os testes e a validação de aplicativos ainda estão em andamento.

- Teste a resiliência antes da implantação na produção: valide o comportamento do cluster em caso de Single-AZ perda iniciando manualmente as mudanças zonais de teste ou habilitando a prática de mudança automática zonal para verificar se os aplicativos mantêm a disponibilidade, o desempenho permanece aceitável e a capacidade é suficiente ao operar com uma contagem reduzida de zonas de disponibilidade. É altamente recomendável esse teste para que você possa identificar lacunas na configuração antes que ocorram deficiências reais na Zona de Disponibilidade.
- Coordene com a configuração do balanceador de carga: para aplicativos que recebem tráfego externo, ative a mudança zonal ARC nos balanceadores de carga de aplicativos e balanceadores de carga de rede associados para garantir que o tráfego de entrada e o tráfego no cluster leste-oeste mudem juntos durante deficiências na zona de disponibilidade. Essa coordenação evita cenários em que solicitações externas cheguem a pods saudáveis, mas esses pods não conseguem se comunicar com dependências na zona afastada.
- Monitore as operações de turno: depois de ativar o turno zonal, configure o monitoramento e o alerta para eventos de turno, incluindo ativações de turnos automáticos, iniciações manuais de turnos e vencimentos de turnos, para manter a visibilidade operacional das ações de gerenciamento de tráfego e seu impacto no comportamento do aplicativo.

Conclusão e recuperação do turno

Quando um deslocamento zonal expira com base em sua duração configurada ou é cancelado manualmente após a resolução do comprometimento da Zona de Disponibilidade, o EndpointSlice controlador atualiza automaticamente tudo EndpointSlices para reincorporar os endpoints na Zona de Disponibilidade restaurada. O tráfego retorna gradualmente à zona anteriormente afetada à medida que os clientes atualizam as informações do endpoint e estabelecem novas conexões. Isso permite a utilização total da capacidade do cluster sem exigir intervenção manual ou reprogramação do pod.

Conclusão

Ao projetar sua arquitetura para alta disponibilidade e resiliência de aplicativos, considere os seguintes componentes:

- O aplicativo de microsserviços (seus pods e contêineres)
- O plano de dados da carga de trabalho (Ingress Controller, pod, componentes do sistema, como Amazon VPC CNI, service mesh sidecars e kube-proxy)
- A camada de gerenciamento da carga de trabalho (controladores, controladores de admissão, mecanismos de políticas de rede e armazenamento persistente de dados para esses componentes)
- O plano de controle do Kubernetes
- Infraestrutura (nós, rede e dispositivos de rede)

Para abordar essas considerações de componentes, use as seguintes estratégias principais:

- Para ajudar a garantir alta disponibilidade e tolerância a falhas, distribua as cargas de trabalho entre nós e zonas de disponibilidade.
- Para proteger cargas de trabalho críticas, mantenha a estabilidade do aplicativo durante interrupções usando orçamentos de interrupção do pod (). PDBs
- Para ajudar a garantir que os pods estejam funcionando e veiculando o tráfego corretamente, configure testes de inicialização, testes de atividade, testes de prontidão e verificações de integridade do balanceador de carga.
- Para gerenciar as transições de estado do contêiner com eficiência, configure os ganchos do ciclo de vida do contêiner.
- Para fornecer controle sobre o processo de despejo durante falhas ou manutenção do nó, configure o tempo de despejo do pod.

Ao implementar essas práticas, você pode melhorar significativamente a confiabilidade e a resiliência dos aplicativos executados no Amazon EKS, garantindo desempenho robusto e alta disponibilidade.

Recursos

- Restrições de dispersão da topologia do pod Kubernetes (documentação do Kubernetes)
- Karpenter FAQs (documentação do Karpenter)
- Desagendador para Kubernetes (repositório) GitHub
- Guia de melhores práticas do Amazon EKS para disponibilidade e ciclo de vida do pod
- Encerre aplicativos normalmente | Guia de melhores práticas do Amazon EKS
- [EKS] [solicitação]: capacidade de configuração pod-eviction-timeout e soluções alternativas (repositório Containers Roadmap)

Histórico do documento

A tabela a seguir descreve alterações significativas feitas neste guia. Se desejar receber notificações sobre futuras atualizações, inscreva-se em um feed RSS.

Alteração	Descrição	Data
Atualização	Revisou a seção sobre despejos de cápsulas durante interrupções zonais.	29 de outubro de 2025
Atualização	A seção Usar restrições de dispersão da topologia do pod foi revisada.	27 de janeiro de 2025
Publicação inicial	—	23 de outubro de 2024

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.