Artigo
Vinicius Maranh... · Maio 11, 2021 7min de leitura

Escalando horizontalmente o IRIS no Kubernetes de acordo com o tamanho da fila - Parte 1

Introdução

Suponha que você desenvolveu uma nova aplicação utilizando a parte de Interoperabilidade do InterSystems IRIS e você tem certeza de que será um sucesso! No entanto, você ainda não tem um número concreto de quantas pessoas irão utilizá-la. Além disso, pode haver dias específicos em que há mais pessoas utilizando sua aplicação e dias em que quase ninguém irá acessar. Deste modo, você necessita de que sua aplicação seja escalável!

O Kubernetes já nos ajuda bastante nesta tarefa com um componente chamado Horizontal Pod Autoscaler, que permite escalar horizontalmente uma aplicação baseado em uma métrica específica. Através do componente " Metrics Server " do próprio Kubernetes, é possível obter métricas como utilização de CPU ou de memória. Mas, e se você necessitar escalar sua aplicação utilizando outra métrica, como por exemplo o tamanho da fila de todos os " Business Hosts " presentes em uma determinada Produção da Interoperabilidade do IRIS? Por se tratar de uma métrica customizada, será necessário desenvolver um serviço que irá expor essa métrica customizada para o Kubernetes.

Nesta série de artigos, veremos em mais detalhes como funciona o processo de escalonamento horizontal do Kubernetes. Na parte 1, será abordado o funcionamento do componente responsável pelo escalonamento horizontal do Kubernetes e a criação e exposição de uma métrica customizada do IRIS no formato esperado pelo Prometheus. Já na parte 2, abordaremos como configurar o Prometheus para ler essa métrica e como configurar o prometheus-adapter para expor essa métrica ao Kubernetes.

O projeto utilizado para este artigo está totalmente disponível no GitHub em:

https://github.com/vmrcastro/iris-autoscale

Kubernetes - Horizontal Pod Autoscaler

Como mencionado na Introdução, o Horizontal Pod Autoscaler é um recurso do Kubernetes que permite escalar horizontalmente aplicações baseado no valor de uma métrica específica. No exemplo abaixo, é exibido um arquivo yaml contendo a definição de um Horizontal Pod Autoscaler:

```
 kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2beta1
metadata:
  name: iris-autoscale
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: iris-autoscale
  # autoscale between 1 and 5 replicas
  minReplicas: 1
  maxReplicas: 5
  metrics:
  # use a "Pods" metric
  - type: Pods
    pods:
```

```
metricName: queuesize
# target queue size of 10
targetAverageValue: 10
```

Observe que nele definimos o que queremos escalar horizontalmente, neste caso o Deployment denominado " iris-autoscale " , qual é o valor mínimo e máximo de réplicas de IRIS que desejamos que executem simultaneamente, a quem essa métrica se refere (no nosso caso será PODs), qual é a métrica que utilizaremos como balizadora (" queuesize " neste exemplo) e qual é o valor médio dessa métrica que utilizaremos como alvo.

O funcionamento simplificado do Horizontal Pod Autoscaler segue o seguinte diagrama:

Fonte: https://learnk8s.io/autoscaling-apps-kubernetes

A cada 15 segundos, o Horizontal Pod Autoscaler busca o valor da métrica configurada, calcula a quantidade de réplicas da aplicação que devem estar rodando baseado na métrica obtida dessa busca e no valor alvo configurado no Horizontal Pod Autoscaler e, se necessário, adiciona ou remove réplicas da aplicação.

O cálculo da quantidade de réplicas que devem estar rodando é realizado através da seguinte fórmula:

$$X = N * (c/t)$$

onde X é a quantidade de réplicas que devem estar rodando, N é o número atual de réplicas em execução, c é o valor atual da métrica e t é o valor alvo da métrica.

No entanto, esse é o funcionamento simplificado. Na verdade, o Horizontal Pod Autoscaler não busca diretamente na aplicação o valor da métrica configurada. Ao invés disso, o Horizontal Pod Autoscaler busca esses valores no " Metrics Registry " , um outro componente do Kubernetes cuja função principal é fornecer uma interface padrão para todas as métricas que serão expostas para diversos clientes – no nosso caso, o Horizontal Pod Autoscaler.

Fonte: https://learnk8s.io/autoscaling-apps-kubernetes

O " Metrics Registry " possui 3 APIs distintas: Resource Metrics API, Custom Metrics API e a External Metric API. No nosso caso, utilizaremos a Custom Metrics API, que é destinada para métricas associadas a um objeto do Kubernetes mas não é nenhuma daquelas métricas de CPU e memória já " fornecidas por padrão " no Kubernetes.

Portanto, será necessário expor nossa métrica customizada para a Custom Metrics API do Metrics Registry. Para isso, necessitaremos 2 componentes: o " metric API server " , que irá efetivamente expor essa métrica customizada e o " metric collector " , responsável por coletar as métricas de cada um dos PODs que estão executando a aplicação e fornecê-las ao " metric API server " .

Fonte: https://learnk8s.io/autoscaling-apps-kubernetes

Há diversas formas de implementar o " metric collector " e o " metric API server " . Neste artigo, utilizaremos o Prometheus como " metric collector " e o Prometheus Adapter como " metric API server " para aproveitar a facilidade que o InterSystems IRIS fornece para criar e expor uma métrica ao Prometheus.

InterSystems IRIS – Criando uma métrica customizada

O InterSystems IRIS já expõe, por padrão, uma série de métricas de monitoramento no padrão conhecido pelo Prometheus. Essas métricas são comumente utilizadas por uma outra grande ferramenta que a InterSystems disponibiliza para monitoramento de instâncias do InterSystems IRIS, o InterSystems SAM (sigla para System Alerting and Monitoring). Neste artigo, não utilizaremos o SAM por enquanto, mas se você quiser saber mais, a documentação deste produto está disponível em:

https://docs.intersystems.com/sam/csp/docbook/DocBook.UI.Page.cls?KEY=ASAM

A lista completa de métricas disponíveis por padrão em uma instância do IRIS pode ser encontrada neste endereço:

https://irisdocs.intersystems.com/irislatest/csp/docbook/Doc.View.cls?KEY=GCMrest#GCMrestmetricstable

Caso você possua uma instância de IRIS rodando, você pode enviar uma requisição REST diretamente para o endpoint de exposição dessas métricas e analisar a resposta. Para isso, basta executar:
curl http://*<IPdoIRIS>* :*<PortaWebServerIRIS>* /api/monitor/metrics

Esta requisição acima será exatamente a requisição que o Prometheus ficará enviando às instâncias do IRIS de tempos em tempos para monitorá-las.

Como não há nenhuma métrica relacionada ao tamanho da fila de uma Produção da Interoperabilidade do IRIS, iremos criar uma métrica que denominaremos " queuesize " seguindo as instruções da documentação disponível em:

https://docs.intersystems.com/irislatest/csp/docbook/Doc.View.cls?KEY=GCMrest#GCMrestmetricsapplication

Como vocês podem perceber, é muito simples criar e expor uma nova métrica no IRIS. Basta criar uma classe que estende a classe " %SYS.Monitor.SAM.Abstract " , definindo um nome no parâmetro " PRODUCT " para o prefixo das métricas que serão criadas e implementar o método " GetSensors() " para definir o nome e valor das métricas atráves da chamada para o método " SetSensor() " . Esta foi a classe desenvolvida para criarmos e expormos a métrica " queuesize " :

```
Class CustomMetric.QueueMetric Extends %SYS.Monitor.SAM.Abstract
{

Parameter PRODUCT = "queue";

/// Collect metrics from the specified sensors
Method GetSensors() As %Status
{
    &sql(SELECT SUM("Count") INTO :queuesize FROM EnsPortal.Queues_EnumerateQueues())
    If (SQLCODE<0 || SQLCODE=100) {
        Return $$$ERROR($$$SQLError, $System.SQL.SQLCODE(SQLCODE))
    }
    do ..SetSensor("size",queuesize)
    Return $$$OK
}

}
```

Com essa classe compilada, basta executar no Terminal do IRIS:

```
%SYS>set status = ##class(SYS.Monitor.SAM.Config).AddApplicationClass("CustomMetric.Q
```

```
ueueMetric", "AUTOSCALE")

%SYS>w status

status=1
```

e editar a Web Application /api/monitor adicionando a Application Role necessária para acessar essa classe. Feito isso, é possível enviar uma nova requisição
curl http://*<IPdoIRIS>* :*<PortaWebServerIRIS>* /api/monitor/metrics

e verificar que a métrica " queuesize " está sendo exibida.

Conclusão

Nesta primeira parte, abordamos com mais detalhes o funcionamento e a definição do Horizontal Pod Autoscaler, componente do Kubernetes responsável por promover o escalonamento horizontal de entidades do Kubernetes (no nosso caso PODs) de maneira automática. Além disso, abordamos a criação de uma métrica customizada no IRIS e a sua disponibilização em um formato padrão do Prometheus.

No próximo artigo da série, veremos como configurar o Prometheus para ler essa métrica customizada e como configurar o prometheus-adapter para expô-la ao Kubernetes.

Referencias

https://learnk8s.io/autoscaling-apps-kubernetes

https://prometheus.io/docs/introduction/firststeps/

https://prometheus.io/docs/prometheus/latest/configuration/configuration/

https://github.com/kubernetes-sigs/prometheus-adapter/blob/master/docs/config-walkthrough.md

https://github.com/kubernetes-sigs/prometheus-adapter/blob/master/docs/walkthrough.md

https://github.com/prometheus/prometheus/blob/release-2.26/documentation/examples/prometheus-kubernetes.yml

#Conteinerização #Entrega Contínua #Integração Contínua #Kubernetes #Nuvem #InterSystems IRIS

URL de origem:https://pt.community.intersystems.com/post/escalando-horizontalmente-o-iris-no-kubernetes-de-acordo-com-o-tamanho-da-fila-parte-1