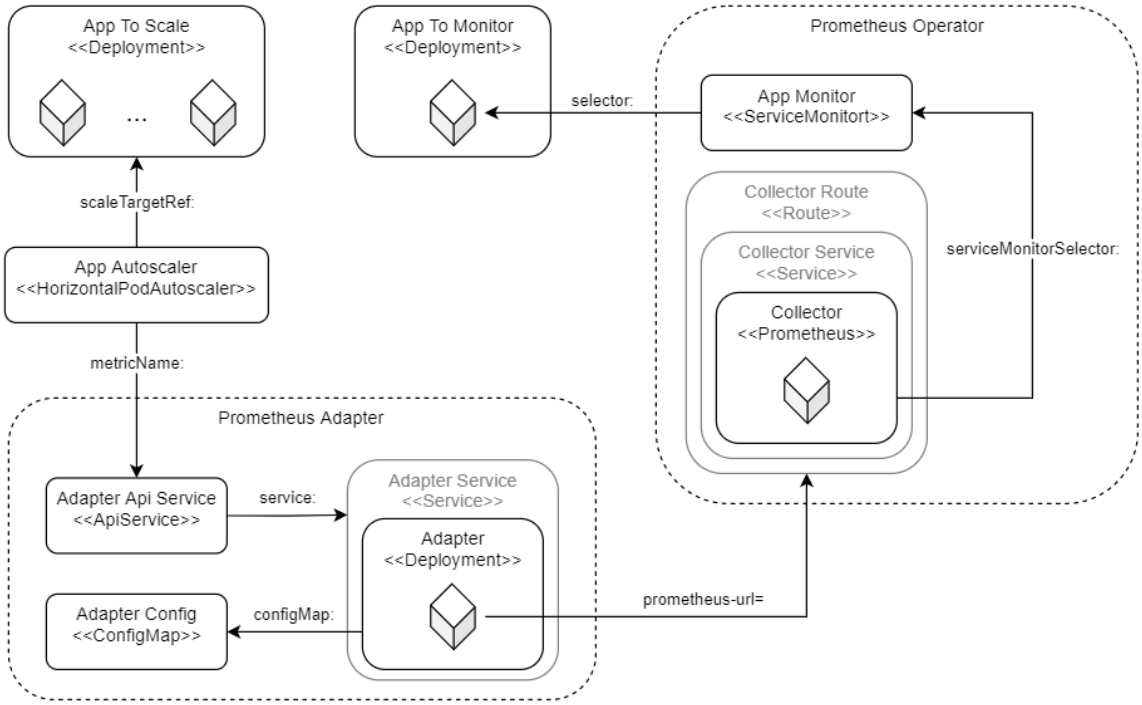
Custom metrics autoscaling

Openshift consente lo scaling automatico delle applicazioni dispiegate all'interno del suo Kubernetes Cluster, tramite l'utilizzo della risorsa *HorizontalPodAutoscaler* e di un *Custom Metrics Api Server*.

Un modo per implementare questo controllo, vede l'impiego sia delle risorse *Prometheus* e *ServiceMonitor,* messe a disposizione dal *Prometheus Operator* per il monitoraggio dell'applicazione, sia di un *Metrics Api Server* realizzato tramite *Prometheus Adapter.*
Quest'ultimo, andrà a leggere le metriche memorizzate da *Prometheus* e le esporrà estendendo il comportamento del *Kubernetes API server.*

L' *HorizontalPodAutoscaler* verrà quindi configurato per l'utilizzo di una metrica esposta dall' *ApiService* del *Prometheus Adapter.*



Custom Metrics Autoscaling architecture

Il dispiegamento delle risorse necessarie alla realizzazione del meccanismo di controllo è tratto da un articolo presente sulla piattaforma *medium.com*:

https://medium.com/ibm-cloud/autoscaling-applications-on-openshift-container-platform-3-11-with-custom-metrics-6e9c14474de3

Un primo dispiegamento è stato fatto installando tutti i componenti all'interno del namespace *default* di Openshift, come descritto dall'articolo.
E' stato poi necessario apportare alcuni accorgimenti rispetto a quanto descritto, per superare problemi di compatibilità ed errori di deployment.

Installazione del *Prometheus Operator framework*

Piuttosto che installare *Prometheus Operator* tramite *Ansible playbooks,* come descritto nell'articolo, è stato utilizzato un *bundle* messo a disposizione dal progetto *prometheus-operator*, all'interno del repository GitHub

https://github.com/prometheus-operator/prometheus-operator

Nel *README.md* del progetto, viene descritto il componente e le risorse che mette a disposizione, tra cui *Prometheus* e *ServiceMonitor.*
L'installazione in Openshift deve essere eseguita da un *cluster admin,* tramite comando:

oc apply -f bundle.yaml

Per superare problemi di compatibilità, è stato necessario modificare il file *bundle.yaml* e utilizzare l'*apiVersion*

apiextensions.k8s.io/v1beta1

in sostituzione di

apiextensions.k8s.io/v1

per le risorse di tipo *CustomResourceDefinition.*

L'installazione del *bundle* potrebbe andare comunque in errore a causa di un *user id* non valido, perché non presente all'interno del range di *uid* a cui Openshift consente di effettuare il dispiegamento di un Pod (nel nostro caso del pod che a runtime svolge il ruolo di Operator).

Tra gli eventi di errore associati al Pod, sarà possibile visualizzare il corretto range di *uid* consentito. Bisognerà quindi modificare il campo

securityContext.runAsUser: <uid corrente>

presente all'interno della risorsa *Deployment* del file *bundle.yaml,* ed utilizzare un *uid* valido.

Setup di *Prometheus*

Una volta dispiegata l'applicazione da monitorare e installato l'Operator, l'articolo illustra come dispiegare un'istanza di *Prometeus*, in modo che monitori l'applicazione tramite la risorsa *ServiceMonitor.*

In sostanza, il *SeriveMonitor* viene agganciato all'applicazione tramite il campo

selector.matchLabels: <app labels>

mentre *Prometheus* viene collegato al *SeriveMonitor* tramite il campo

serviceMonitorSelector.matchLabels: <service monitor labels>

In questo modo *Prometheus* potrà collezionare di volta in volta i valori delle metriche esposte dall'applicazione.

Dispiegamento del *Prometheus Adapter*

L'articolo prosegue con il dispiegamento del *Prometheus Adapter, per il quale* sarà necessaria la creazione di risorse *RBAC* quali *ServiceAccount, ClusterRole, RoleBinding e ClusterRoleBinding,* che consentiranno all'adapter di lavorare correttamente.

Verrà poi definita la risorsa *ConfigMap,* attraverso la quale potrà essere specificata sia la query per la lettura delle metriche collezionate da *Prometheus*, sia il modo in cui queste metriche saranno "wrappate" per essere esposte sull'*ApiServer*.

Al fine di correggere alcuni errori riscontrati durante il deploy dell'adapter, si è reso necessario modificare il *ClusterRole* di nome *custom-metrics-resource-reader*, aggiungendo al campo

rules.resource: <resource list>

la voce

- configmaps

E aggiungendo al campo

rules.verbs: <operations type list>

la voce

- watch

In modo da consentire all'adapter di poter accedere correttamente alle risorse di cui ha bisogno.

Verrà infine creata una risorsa di tipo *APIService* per esporre la metrica e verrà dispiegato, tramite *Deployment,* il *Prometheus Adapter.*
Quest'ultimo sarà agganciato all'istanza *Prometheus,* dispiegata in precedenza, tramite il flag

--prometheus-url= <prometheus host>

definito in fase di deployment.
Sarà quindi possibile verificare il corretto funzionamento dell'adapter interrogando l'api server tramite il comando:

oc get --raw "/apis/custom.metrics.k8s.io/v1beta1/namespaces/default/pods//<metric name>"*

Creazione di un *Horizontal Pod Autoscaler*

Step finale della configurazione, consiste nel dispiegamento di una risorsa di tipo *HorizontalPodAutoscaler*.
Questo verrà agganciato all'applicazione da scalare tramite il campo

spec.scaleTargetRef: <application deployment name>

e avrà come input la metrica esposta dal *Prometheus Adapter*, tramite configurazione del campo

metrics.pods.metricName: <metric name>

Ulteriori verifiche e configurazioni da effettuare

Per il dispiegamento di test che è stato fatto, l'*HorizontalPodAutoscaler* è stato configurato per scalare la stessa applicazione monitorata, come descritto dall'articolo, ed è stato osservato il corretto funzionamento dello scaling automatico al variare della metrica esposta.

Passaggio successivo sarà quello di verificare l'autoscaling, agganciando all' l'*HorizontalPodAutoscaler* un'applicazione diversa da quella monitorata.

Occorrerà poi configurare opportunamente i ruoli al'interno del cluster, in modo da consentire ad un utente diverso dal *cluster admin* di poter dispiegare le risorse necessarie all'autoscaling, all'interno di un namespace diverso da quello di default.