**Autoscaling in Kubernetes**

Kubernetes è un orchestratore di container creato originariamente da Google e che recentemente si sta affermando come standard nel deploy di containerized applications. Anche se come detto Kubernetes ha lo scopo di effettuare deploy, ma anche scaling e management di continerized applications, esso aggiunge una nuova astrazione rispetto al semplice uso dei container con Docker. L’unita base di scheduling di Kubernetes, infatti, è il pod.

Un pod consiste in un insieme di container dei quali si ha la certezza che risiedono nella stessa host machine. I container all’interno del pod potranno condividere risorse e ogni pod avrà un solo indirizzo IP, mentre le varie applicazioni all’interno dei vari container saranno raggiungibili attraverso numeri di porta specifici. Questa scelta è stata presa poiché normalmente un’applicazione web risiede in più container, quindi la scelta più ovvia è quella di creare un nuovo livello di astrazione in cui risiede l’intera applicazione, per evitare che un’applicazione si trovi smembrata in container localizzati su più host machine, anche lontane tra loro.

Quindi, parlando di autoscaling in Kubernetes dobbiamo tenere presente che si considerano sempre i pod come unità base per effettuare le operazioni di scale in-out, up-down. I componenti che si occupano di effettuare l’autoscaling sono 3: horizontal pod autoscaler (HPA), cluster autoscaler (CA), vertical pod autoscaler (VPA).

L’HPA è implementato attraverso Kubernetes API e un controller, esso è responsabile della gestione del numero di repliche di un dato pod. Questa operazione viene effettuata dal controller che osserva periodicamente la quantità di CPU utilizzata dal pod (o, ancora in versione alpha, metriche diverse) e se viene superata una soglia prefissata, l’HPA provvederà a creare una nuova replica (o a cancellarne una nel caso inverso). Questo meccanismo non è centralizzato, infatti ogni pod avrà il proprio controller. Tutte le specifiche quali soglia CPU, numero minimo repliche e numero massimo repliche, sono definite alla creazione del controller e possono essere aggiornate a seconda delle esigenze.

Quindi ogni volta che un pod si troverà a ricevere molte richieste e la sua utilizzazione supererà la soglia prefissata, l’HPA proverà a creare una nuova copia di quel pod all’interno del cluster gestito da Kubernetes, ma nel caso in cui non ci siano risorse sufficienti per creare una nuova replica, sarà compito del CA di trovare un nuovo nodo in cui lanciare il nuovo pod. Il cluster autoscaler proverà ad aggiungere nuovi nodi, a seconda delle richieste, dove far risiedere le applicazioni e nel caso in cui il carico diminuisca e l’HPA elimini alcune repliche, se ci saranno nodi vuoti o sottoutilizzati, allora il CA provvederà all’eliminazione di questi. CA è stato sviluppato come un componente autonomo e attualmente è supportato su GCE, GKE, AWS e Azure.

L’ultima funzionalità di scaling risiede nel VPA che come per HPA effettuerà lo scaling a livello di pod. Proprio come per HPA, un dato pod viene monitorato riguardo l’utilizzo della CPU (o altre metriche) e a seconda dei valori di questo, verranno effettuate operazioni di scaling. Questa volta però invece di creare una nuova replica del pod, verrà effettuato uno scaling verticale dello stesso pod, allocando a questo dinamicamente le risorse necessarie a soddisfare il carico di richieste. Al momento però questa funzionalità si trova ancora sottosviluppo e non è compresa nell’ultima release di Kubernetes.

**Riferimenti**

<https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

<https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>

<https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md>

<https://kubernetes.io/docs/tasks/administer-cluster/cluster-management/#cluster-autoscaling>

<https://github.com/kubernetes/community/pull/338/files>