UNIVERSITÀ DI PISA

POD SNIFFER

Progetto Gestione Reti 2018/2019

Studente: **Emilio Panti**

Matricola: **531844**

INTRODUZIONE

Negli ultimi anni la tecnologia dei container (applicazioni isolate contenenti anche le relative dipendenze) ha registrato una importante crescita ed un ampio utilizzo.

Ciò ha comportato la nascita di diversi strumenti per la loro creazione, gestione ed esecuzione.

Tra questi spicca Kubernetes: sistema open-source di orchestrazione e gestione di container. Il suo scopo principale è quello di coordinare cluster di nodi su larga scala in modo efficiente.

L' elemento base di Kubernetes è il pod, che aggiunge un più alto livello di astrazione raggruppando container. All' interno di un pod i container condividono le risorse, mentre il pod si occupa di astrarre rete e storage al fine di poter spostare più facilmente un container all' interno del cluster.

I nodi nei cluster gestiti da Kubernetes possono essere responsabili di più pod contemporaneamente.

Le astrazioni e gli isolamenti che sfruttano i container e Kubernetes possono complicare il monitoraggio e la raccolta di informazioni che li riguardano.

Scopo di questo progetto è fornire un software in grado di catturare i pacchetti di rete di un determinato pod.

POD SNIFFER

Lo script *podsniffer.sh* permette la cattura dei pacchetti di rete destinati ad un determinato pod. E’ possibile specificare anche la durata (in secondi) della cattura (valore di default: 100s) ed il namespace relativo al pod (valore di default: “default”).

Prima di iniziare la cattura stampa anche la lista dei namespace/pod/container attivi.

Utilizzo:

```
Usage: ./podsniffer.sh POD-NAME [-ns NAMESPACE] [-d DURATION]
Options:
  -ns|--namespace    The namespace where the target pod lives (default: default)
  -d|--duration      The capture duration (default: 100s)
```

Esempio:

```
emilio@emilio-VivoBook:~/Scrivania/GR$ ./podsniffer.sh kube-apiserver-minikube -ns kube-system -d 10

PODS LIST:
namespace:default
   pod:ping-test-68d6784b56-c99sg
      container:docker://6a5a684777d17a82e1ec8928471c8925802eb898dd4fc20adeb145baf8c485c7
namespace:kube-node-lease
namespace:kube-public
namespace:kube-system
   pod:coredns-fb8b8dccf-6sg4w
      container:docker://66b524ef5fa9c4205d49dce5b3145244807555dc2275e819833c5de9725b0237
   pod:coredns-fb8b8dccf-rfsbz
      container:docker://9efaa3aaaae5ea3bf198f556cb7cc164953ddc710448738a3443a3f5d704a1f4
   pod:etcd-minikube
      container:docker://6b0559c21b28a80031b5eb4c35366df33c2bcc7c4a0d1e97d92098156daf2417
   pod:kube-addon-manager-minikube
      container:docker://381831c4775ace606b43d3dedb3be352a29ff6efd779613c30fc4fb6b5a9f4f8
   pod:kube-apiserver-minikube
      container:docker://1fb3dbf25f7c8d4c488fe4bb7c260bfaaefa834b82387851e58e439a07ca924e
   pod:kube-controller-manager-minikube
      container:docker://a17dec2b38c69ece46fce5c58d891ce25122c7a07507ea7089d7f87c4ec3e2b7
   pod:kube-proxy-jr898
      container:docker://f05a4db7baa0a75b8e8c6ac45072474da0135e3dc7b09111ef279f454e9b663b
   pod:kube-scheduler-minikube
      container:docker://87160071c7c6357b768cdaa5c4a234967fb9d9f482725c2d75669f2458d97269
   pod:storage-provisioner
      container:docker://1892e64197d69294b6d1fb90aedc5a188d4ad595ee4720cecf467c7afcd5ca3d
namespace:test01

Start capture on:
  Node: minikube
  Namespace: kube-system
  Pod: kube-apiserver-minikube
  Duration: 10 seconds
Capturing...
Capture over!
The capture has been downloaded to your hard disk at:
 /home/emilio/Scrivania/GR/capture-kube-apiserver-minikube-1557658464.pcap
```

Descriverò brevemente come opera lo script; il quale:

- controlla i parametri e stampa la lista dei pod attivi (raccolti per namespace e specificandone i container interni)
- interroga il pod di interesse per conoscere l'indice dell' interfaccia di rete su cui è in ascolto.
- fa partire un job (il cui file .yaml viene creato ad hoc e poi eliminato una volta applicato il job), che provoca la creazione di

un nuovo pod. All'interno di quest'ultimo è presente un container con tutti gli strumenti necessari alla cattura (immagine utilizzata: *"docker.io/centos/tools"*). Il pod di cattura appena creato vede tutte le interfacce di rete della macchina host su cui è in esecuzione (grazie al flag *hostNetwork=true* specificato nel file .yaml del job). Riesce in questo modo a recuperare l'interfaccia di rete relativa all' indice trovato al punto precedente. Da essa inizia a catturare i pacchetti destinati al pod di interesse (tramite *tcpdump).*

- esegue una sleep (della durata specificata al lancio dello script). Una volta risvegliato recupera dal pod di cattura il file .pcap (contenente i pacchetti di interesse raccolti).
- ripulisce l'ambiente terminando il job sopra creato (la cui terminazione provoca anche la cancellazione del pod utilizzato per la cattura)

Nota: l' approccio utilizzato dallo script (creare un nuovo pod destinato alla cattura) è pensato per i casi in cui risulti impossibile/difficile operare direttamente all' interno delle macchine ospitanti i nodi di interesse.

Questa soluzione può infatti risultare invasiva in contesti in cui siano disponibili poche risorse o nei quali si voglia mantenere inalterato l'ambiente di lavoro.

TESTING

Il progetto è stato testato sul sistema operativo **Ubuntu 18.04.2 LTS**, utilizzando come cluster Kubernetes **minikube v1.0.0**.

Lo script *test.sh* fornisce un test di cattura su pod:

- Crea un pod di testing (che semplicemente esegue un ping)
- Esegue lo script *podsniffer.sh* sul pod di testing
- Ripulisce l'ambiente, terminando il pod di testing