projectAnnoScorso.md 6.69 KB

Helm/OpenShift controller

Implementare un micro-servizio che offra delle HTTP API per controllare un cluster OpenShift, tramite Helm. Il micro-servizio deve far riferimento ad un db per le informazioni persistenti: utilizzare preferibilmente Neo4J ed organizzare i dati come grafi). Per l'implementazione del micro-servizi, nella scelta dei linguaggi, si consigliano in ordine di "linguaggio economicamente piu' vantaggioso":

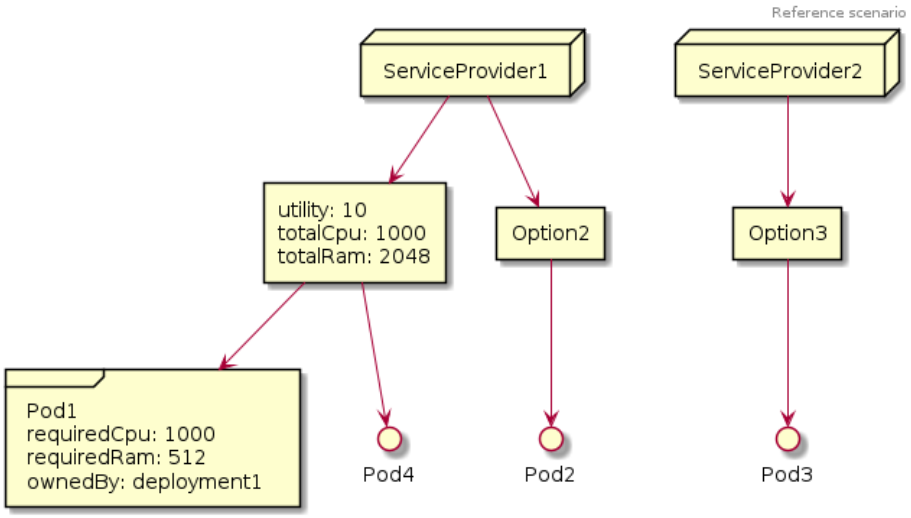
1. Golang (con BeeGo, Template, neo4j-go-driver)
2. Python (con Flask, neo4j)
3. Java (Spring, Neo4j)

Nello scenario di riferimento, piu' Service Provider competono per le risorse del cluster OpenShift.

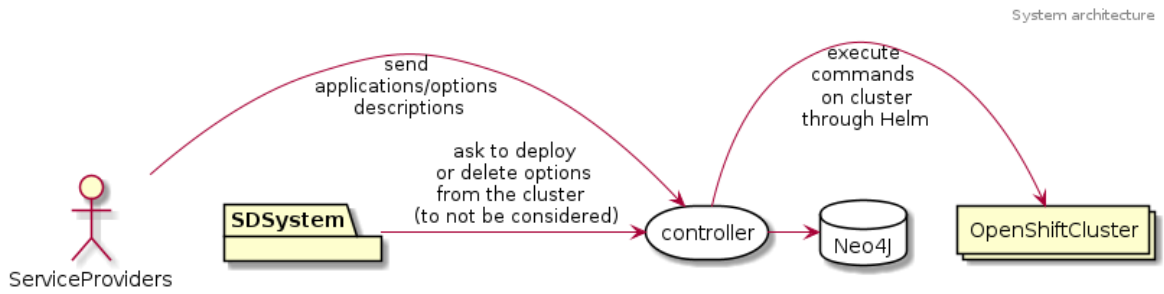
Ogni Service Provider e' associato a una Applicazione, descritta da un template Helm.

Ogni applicazione e' costituita da una o piu' opzioni che sono gestite attraverso piu' file values.yaml: ogni values.yaml istruisce Helm su come effettuare il deploy dell'applicazione.

Una descrizione piu' approfondita dello scenario di riferimento si puo' trovare in sezione 3 di *"EdgeMORE: improving resource allocation with multiple options from tenants"* (link disponibile in calce).



Nota: Lo schema sopra va considerato per solo scopo esemplificativo: non costituisce uno strict requirement sul modello dei dati da rappresentare in database.



Ogni file values.yaml riportera' informazioni sulle risorse da assegnare e sulla "utility" fornita per il deploy della specifica opzione.

Si possono usare diversi progetti helm di test, ad esempio: https://github.com/helm/charts-repo-actions-demo/tree/master/charts/example-v2/templates

In particolare, le API da implementare sono:

- GET /service-providers => esegue il retrieve della lista dei service providers che includono un nome, un id e un serviceProviderPath che

consiste di una directory in cui trovare il progetto helm da deployare;

- GET /service-providers/:id/options/ => Esegue il retrieve delle opzioni disponibili per l'applicazione del service provider :id. Questa lista sara' costituita da oggetti json che riporteranno un insieme di coppie chiave valore che rappresentano il totale delle risorse richieste dall'applicazione e la sua utilita'... Esempio:

```
[ {
 optionId: 4,
 cpu: 1000,
 ram: 2048,
 storage: 8196,
 utility: 10,
 valuesFilePath: "path/to/values.yaml",
},
]
```

- `GET /service-providers/:id/options/:id2` => Esegue il retrieve della singola opzione del service provider `:id` con id `:id2`;
- `PUT /service-providers/:id/options/:id2/deploy` => esegue il deploy dell'opzione `:id` del service provider `:id2`;
- `DELETE /service-providers/:id/options/:id2/deploy` => elimina il deploy dell'opzione `:id2` del service prodiver `:id`;
- `POST /service-providers/:id/options` => Aggiunge una opzione prendendo in ingresso il valueFilePath. Quindi esegue il parsing dello yaml per calcolare utility e le risorse totali associate alla opzione come somma delle risorse di ogni pod (per l'eventuale numero di repliche). **HINT**: un punto di partenza potrebbe essere utilizzare l'opzione "dry-run" di helm.
- `POST /service-providers` => aggiunge un nuovo service provider prendendo in ingresso almeno un nome e un "serviceProviderPath"

Rappresentazione delle opzioni

Progettare il modello dei dati in db affinche' per ogni opzione vengano conservate le informazioni dettagliate del grafo dell'applicazione. Ovvero, la call POST deve essere tale da conservare (fare parsing dal values e dal progetto helm) non solo il totale delle risorse richieste con la relativa utility, ma anche l'insieme dei Pod che verrebbero deployati insieme alle loro risorse richieste. Di conseguenza la GET call per una specifica opzione dovra' restituire un json come il seguente, in cui l'informazione sul numero di repliche viene normalizzata (se per un deployment vengono richieste due repliche, nell'array resources si troveranno due pod 'uguali' che rappresentano le repliche, la chiave OwnedBy potra' essere utilizzata per indicizzare tutti i pod afferenti la stesso gruppo di repliche):

```
{
 optionId: 4,
 cpu: 1000,
 ram: 2048,
 storage: 8196,
 utility: 10,
 valuesFilePath: …..,
 Resources: [
    {
       Name: pod-x-replica-1
       OwnedBy: deployment-x-1
       Cpu: x
       Ram: y
       Storage: z
    }, ...

  ]
}
```

Values.yaml example

Un esempio di file .yaml che definisce una opzione

```
routes:
```

```
  # This is the domain to which a client is redirected if the Edge cannot serve her request
  cloudURL: 'cloud-vms-1.master.particles.dieei.unict.it'
  # This is the domain to which expose the main route used by the clients (actually this would be achieved by a locati
  edgeURL: 'edge-vp-1.master.particles.dieei.unict.it'

vms:
  isCloud: "false" # Let the micro-services know they are executing in the Edge
  maxVideo: 10 # Set limits of the capped collection (i.e., the maximum number of video stored in the Edge, leveraging
  variantType: "0" # Set the VMS to act as the Edge Cache Variant
  requiredCPU: 100m
  requiredRAM: 250Mi
  requiredStorage: 1Gi


vps:
  # Number of replicas of the VPS (the more the replicas the more used resources in the Edge by the SP)
  replicas: 1
  requiredCPU: 1000m
  requiredRAM: 512Mi
  requiredStorage: 512Mi

lb:
  # Sets the maximum number of concurrent users an Edge Deployment can serve
  maxConcurrentUsers: 1
  requiredCPU: 10m
  requiredRAM: 128Mi
  requiredStorage: 0
```

References

- https://helm.sh/
- https://github.com/helm/charts-repo-actions-demo
- https://www.okd.io/minishift/ : utile per avere un equivalente di openshift in locale
- EdgeMORE: improving resource allocation with multiple options from tenants: https://ieeexplore.ieee.org/document/9045173
- https://github.com/mittwald/go-helm-client: utile spunto nel caso si scelga di utilizzare Golang