Haal meer uit je Kubernetes cluster

Alles over Istio

In 2018 kwam ik een nieuwe toevoeging tegen in het Kubernetes landschap, genaamd Istio. Istio is een opkomende tool, die je kunt toevoegen aan je Kubernetes cluster. Het voegt een zogenaamde Service Mesh toe aan je cluster. Istio gebruikt de term *service mesh* om het netwerk van microservices, die je application opmaken en de interacties daartussen, te beschrijven. Istio zorgt ervoor dat je je microservices kunt verbinden, beveiligen, besturen en waarnemen. Op hoog niveau helpt Istio om de complexiteit van cloud deployments te reduceren en zal het de stress van je ontwikkelteams ontlasten.

We hebben ondertussen allemaal wel van Kubernetes gehoord. Het doet een hoop voor ons developers. Het zorgt voor service ontdekking, het aanroepen van de service en de elasticiteit ervan. Kubernetes laat je services ontdekt worden door elkaar. Het zorgt er ook voor dat je services gestart en gestopt worden wanneer je bijvoorbeeld, al dan niet automatisch met behulp van bijvoorbeeld metrics van Heapster, op of- afschaalt of een nieuwe versie deployed.

Openshift voegt weer een stukje meer toe aan deze mix. Openshift voegt monitoring, logging en pipelines toe aan Kubernetes met een Red Hat touch. Maar wat voegt Istio dan toe aan het landschap? Istio breidt de mogelijkheden voor service discovery uit. Het voegt een stuk resilience (zoals je wellicht kent van Hystrix), authenticatie, monitoring en tracing toe aan het Kubernetes landschap.

Hoe installeer je Istio?

Istio kan op verschillende manieren worden geïnstalleerd op je cluster. Alle manieren van installatie vereisen dat Istio zelf wordt gedownload en op je path gezet wordt. Vanaf hier heb je keuze uit 4 mogelijkheden:

1. Installeren van Istio zonder onderlinge
2. TLS authenticatie tussen sidecars.
3. Installeren van Istio met default onderlinge TLS authenticatie.

Een Kubernetes manifest maken met Helm en het daarna deployen met kubectl.
Gebruik Helm en Tiller om Istio deployments te beheren.
Alle opties staan verder beschreven op de quickstart van Istio: **https://istio.io/docs/setup/kubernetes/quick-start/**

Hoe gebruik je Istio?

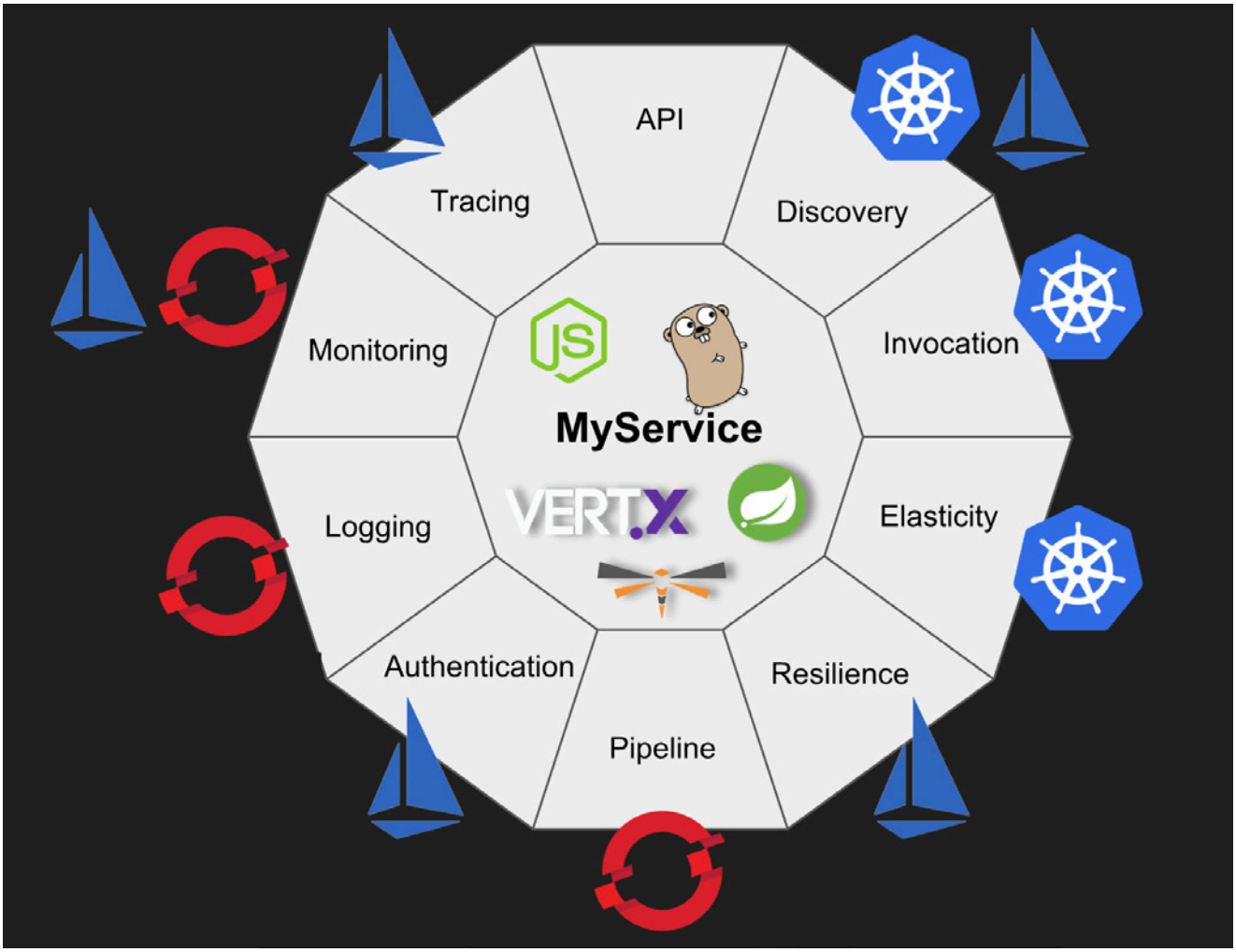
Istio maakt gebruik van het concept van een sidecar container. Dit betekent dat elke pod, die gedeployed is op je cluster, twee containers opstart. De eerste container bevat de service, die je wilt laten draaien en de andere container is een Istio sidecar, die verkeer, security en nog veel meer regelt.

Tijdens het experimenteren wil je natuurlijk weten wat het allemaal precies doet en dus zal je de eerste keer wellicht de sidecar handmatig willen toevoegen. Dit kan vrij gemakkelijk door het volgende uit te voeren:

```
istioctl kube-inject -f <file path
of your deployment and service> |
kubectl apply -f -
```

Deze code zal de sidecar injecteren in je Kubernetes deployment en zal je applicatie en Kubernetes service deployen, zoals je gewend bent. Een alternatief (en wat je natuurlijk liever wilt) is dat je sidecar automatisch geïn-

Bob is Software Developer bij First8 en is gespecialiseerd in Java en open source technologie.



jecteerd wordt. Dit kan gedaan worden door een **mutating webhook admission controller** (**https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/**).

Hiervoor moet je wel Kubernetes 1.9 of hoger geïnstalleerd hebben. Je moet verifiëren dat het kube-apiserver process de admission-control vlag gezet is met de MutatingAdmissionWebhook en ValidatingAdmissionWebhook admission controllers toegevoegd zijn, in de correcte volgorde staan en dat de admission registration API ingeschakeld is. Zie hiervoor de documentatie van Kubernetes.

```
kubectl api-versions | grep
   admissionregistration
```

De output zal er zo uitzien:

```
admissionregistration.k8s.io/v1alpha1
admissionregistration.k8s.io/v1beta1
```

Als dit gedaan ingeschakeld is, kun je jouw deployment doen, zoals je gewend bent. Het injecteren van de sidecar zal op pod niveau gedaan worden door je cluster.

Wat is een Virtual Service?

Behalve de normale Kubernetes services die je nu al kent, zal je waarschijnlijk gebruik gaan maken van Virtual Services. Een Virtual Service binnen Istio definieert een set van verkeersregels, die toegepast worden zodra de host aangeroepen wordt. Zo breidt Istio de features van Kubernetes uit om de discovery van services flexibeler te maken. Zo kun je pattern matching doen op headers, URL en meer. Het kan bijvoorbeeld verkeer naar je pods opsplitsen aan de hand van percentages.

Stel, je hebt je cluster draaien en Istio geïnstalleerd en je wilt een specifiek hoeveelheid van je verkeer naar een nieuwe versie van je software leiden. Dit heet ook wel een **canary release**. Voor het gemak ga ik er hier vanuit

WAT ISTIO TEVENS DOET ALS SIDECAR CONTAINER, IS HET TOEPASSEN VAN CIRCUIT BREAKERS

dat je je applicatie gedeployed hebt. Nu willen we een routing toevoegen met Istio.

```
apiVersion: networking.istio.io/v1al-
pha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
```

Zoals je ziet, gaat al je verkeer naar de review applicatie. Nu is het tijd om een nieuwe versie van je applicatie te deployen. Nu moet je jouw VirtualService updaten om een percentage aan je route toe voegen en om een tweede route toe te creëren.

```
...
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
      weight: 90
    - destination:
        host: reviews
        subset: v2
      weight: 10
```

Zoals je ziet, gaat 10% van het verkeer naar de nieuwe versie van je deployment, met label `subset: v2`. De rest van het verkeer gaat naar de oude versie van je deployment.

Wellicht wil je onderscheid maken tussen verkeer dat binnenkomt vanaf FireFox en de rest. Dit om ze bijvoorbeeld naar andere applicaties te sturen. Met Istio kan dit gemakkelijk. Je kunt een Match definiëren (in dit geval kijkt het naar de headers) en daarna een route om aan te geven hoe je verkeer geleid moet worden. Al het verkeer van Firefox gaat naar versie 1 en de rest gaat default naar versie 2.

```
...
  http:
  - match:
    - headers:
        user-agent:
          regex: ".*firefox.*"
    route:
    - destination:
        host: reviews
        subset: v1
  - route:
    - destination:
        host: reviews
        subset: v2
```

Resilience

Wat Istio tevens doet als sidecar container, is het toepassen van circuit breakers. De meesten zullen wel van Hystrix gehoord hebben en het wellicht zelf gebruikt hebben. Nu is er een alternatief voor circuit breakers, die je niet in je code hoeft toe te passen. Je code hoeft nu alleen maar op de hoofdzaak te letten en zich niet meer bezig houden met externe factoren.

Om dit te doen, zul je een **Istio Destination Rule** moeten toevoegen. Hierin geef je een policy aan, die toegepast wordt, nadat routing van een service plaatsgevonden heeft. De regels binnen de Destination Rule kunnen load balancing, connection pool size of outlier detection bevatten.

De outlier detection settings kunnen ongezonde hosts detecteren en uit de load balancing pool verwijderen. Een simpel voorbeeld van een load balancing policy om connecties naar de hosts te sturen met de minste verbindingen ziet er als volgt uit:

```
apiVersion: networking.istio.io/v1al-
pha3
kind: DestinationRule
metadata:
  name: bookinfo-ratings
spec:
  host: ratings.prod.svc.cluster.
local
  trafficPolicy:
    loadBalancer:
      simple: LEAST_CONN
```

Je kunt dit uitbreiden naar een versie-specifiek beleid in subsets, die voor iedere versie bijvoorbeeld een ander load balancing beleid instelt.

Metrics

Om gebruik te maken van de metrics van Istio, moet je het echter wel aanzetten. Dit kan gedaan worden met de template tijdens installatie, met `tracing enabled` op `true` gezet of met Helm charts:

```
--set tracing.enabled=true
```

Je kunt ook andere zaken aanzetten, zoals tracing sample en service graph. Nu dat dit aanstaat, kun je toegang opzetten voor bijvoorbeeld een Jeager dashboard met:

```
kubectl port-forward -n istio-system
$(kubectl get pod -n istio-system -l
app=jaeger -o jsonpath='{.items[0].
metadata.name}') 16686:16686 &
```

MET ISTIO KUN JE GEMAKKELIJK ONDERSCHEID MAKEN TUSSEN VERKEER DAT BINNENKOMT VANAF BIJVOORBEELD FIREFOX EN ANDERE BROWSERS

Door naar `localhost:16686` te gaan, kom je op het dashboard. Zodra verkeer binnenkomt, zal je data in je dashboard te zien zijn. Je zult ook details kunnen zien, die corresponderen met het laatste verkeer. Om tracing mogelijk te maken, moet je wel de volgende headers van service naar service laten propageren: `x-request-id`, `x-b3-traceid`, `x-b3-spanid`, `x-b3-parentspanid`, `x-b3-sampled`, `x-b3-flags` en `x-ot-span-context`. Met deze headers, kan Istio alle requests volgen en bijhouden welk verkeer waar naartoe gaat. Met tracing aan, kun je ook het verkeer binnen je microservices applicatie visueel maken.

Wil je zelf ook experimenten met Istio? Katacoda (**https://katacoda.com/openshift/courses/servicemesh**) heeft goede workshops, die je kunt proberen. Er staan ook goede sessies online van bijvoorbeeld Devoxx 2018 over Kubernetes en Istio. Uiteraard kun je natuurlijk kijken op istio.io. ■

NU IS ER EEN ALTERNATIEF VOOR CIRCUIT BREAKERS, DIE JE NIET IN JE CODE HOEFT TOE TE PASSEN