

Spiderpool

RDMA network solution for the Kubernetes

Piotr Czarnik, Bartosz Kucharz
Gabriela Piwar, Wojciech Szmelich

AGH Wydział Informatyki
2024

Spis treści

1	Wprowadzenie	1
2	Opis technologii	2
2.1	Kubernetes	2
2.2	AWS	2
2.3	Terraform	2
2.4	Ansible	2
3	Case Study - opis projektu	3
4	Architektura rozwiązania	3
5	Konfiguracja środowiska	3
6	Instalacja	4
7	Opis provisioningu i deploymentu	4
7.1	Terraform	4
7.1.1	Sieć	5
7.1.2	Security groups	5
7.1.3	Instancje	5
7.2	Playbooki Ansible	6
7.2.1	pre-deploy.yml	6
7.2.2	deploy-k8s.yml	6
7.2.3	deploy-spiderpool.yml	7
7.2.4	deploy-apps.yml	7
7.3	Dodatkowa konfiguracja	7
8	Podsumowanie	9

1 Wprowadzenie

Kubernetes to jedno z najpopularniejszych narzędzi do zarządzania aplikacjami kontenerowymi. Z tego też względu nieustannie wprowadzane są nowe moduły usprawniające jego działanie.

Jednym z nich jest Spiderpool - zaawansowane rozwiązanie zarządzania adresami IP (IPAM - IP Address Management) wykorzystujące technologię RDMA (Remote Direct Memory Access). Rozszerza on standardowe interfejsy sieciowe kontenerów (CNI - Container Network Interface) umożliwiając tworzenie interfejsów Macvlan, Ipvlan, oraz SR-IOV. Dzięki temu pozwala na większą dowolność w przypisywaniu adresów IP do kontenerów i w wykorzystaniu interfejsów sieciowych. Natomiast SR-IOV umożliwia kontenerowi na bezpośredni dostęp do fizycznego interfejsu sieciowego - szybszy transfer danych między węzłami w klastrze Kubernetesa, minimalizacja opóźnień i obciążenia procesora. Jest to szczególnie korzystne dla aplikacji wymagających wysokiej przepustowości i niskiego opóźnienia, jak aplikacje do przetwarzania dużych ilości danych, middleware, CNF (Container Network Functions) czy systemy baz danych.

2 Opis technologii

2.1 Kubernetes

Spiderpool działa na klastrach, czyli zestawie maszyn (węzłów) do uruchamiania skonteneryzowanych aplikacji. Kubernetes jest platformą open source do zarządzania takimi klastrami. Służy do zarządzania zadaniami i serwisami uruchamianymi w kontenerach, oraz umożliwia deklaratywną konfigurację i automatyzację. Najmniejsza i najprostsza jednostka w środowisku Kubernetes to pod, czyli grupa jednego lub wielu kontenerów aplikacji. W "czystym" k8s kontenery wewnątrz poda współdzielą adres IP i przestrzeń portów, zawsze są uruchamiane wspólnie w tej samej lokalizacji i współdzielą kontekst wykonawczy na tym samym węźle.

2.2 AWS

Spiderpool jest stworzone z myślą o działaniu na dowolnym środowisku chmurowym. Ułatwia również zarządzanie takimi rozwiązaniami jak multicloud czy chmura hybrydowa.

Jedną z najbardziej znanych i używanych platform chmurowych jest Amazon Web Services (AWS), która zapewnia szeroki wybór usług oraz zasobów obliczeniowych, sieciowych i przechowywania danych. Usługi Amazona są znacznie rozbudowane i umożliwiają skonfigurowanie środowiska w taki sposób, aby było jak najbardziej dopasowane do danych potrzeb. Jedną z najważniejszych usług dostępnych w AWS jest Elastic Compute Cloud (EC2), która umożliwia elastyczne skalowanie zasobów obliczeniowych. Szerokie zastosowanie tej platformy oznacza również, że istnieje ogromna ilość informacji, dokumentacji i pomocy dostępnych dla użytkowników. Dlatego zdecydowano się wdrożyć projekt na tym środowisku.

2.3 Terraform

Terraform jest narzędziem do deklaratywnego konfigurowania i provisionowania infrastruktury cloud'owej z wykorzystaniem paradygmatu „infrastructure as a code”. Z wykorzystaniem deklaratywnego języka HCL (HashCorp Language) pozwala na opisanie stanu końcowego infrastruktury - a doprowadzeniem jej do tego stanu zajmuje się Terraform.

Terraform udostępnia dużą liczbę „provider'ów”, w tym provider dla AWS, umożliwiający konfigurowanie instancji EC2 i sieci VPC.

2.4 Ansible

Ansible jest silnikiem orkiestracji pozwalającym na tworzenie oprogramowania w paradygmacie „infrastructure as a code”. Umożliwia automatyzację provisioningu, konfiguracji i deploymentu systemów oraz oprogramowania za pomocą Playbook-ów - zestawów tasków, które mają się wykonać na wcześniej zdefiniowanych node'ach.

Ansible udostępnia moduły dedykowane do konfiguracji AWSa i Kubernetesa. Wartą uwagi cechą playbooków jest idempotentność operacji - taski sprawdzają, czy dane zadanie już nie zostało wykonane, a jeśli tak to go nie powtarzają - umożliwia to wielokrotne ich uruchamianie bez konieczności czyszczenia całego środowiska.

3 Case Study - opis projektu

Za pomocą wymienionych uprzednio technologii, dążymy do uzyskania automatycznego deploymentu środowiska Spiderpool na klastrze Kubernetesa na instancjach EC2 na chmurze AWS za pomocą Terraform'a i Ansible'a. Do zespolenia tych dwóch narzędzi zostanie użyty skrypt bash'owy.

W ramach projektu zautomatyzowane zostaną więc:

1. provisioning infrastruktury na AWS - VPC i EC2 za pomocą Terraforma,
2. deployment K8s za pomocą Ansible,
3. deployment Spiderpoola za pomocą kubernetesowych manifestów i Ansible,
4. deployment przykładowej aplikacji za pomocą kubernetesowych manifestów i Ansible.

4 Architektura rozwiązania

Nasz deployment składa się z następujących komponentów:

1. jedna sieć publiczna (dla mastera) i dwie sieci prywatne (dla workerów).
2. instancje EC2 skonfigurowane za pomocą terraforma:
 - master z publicznym adresem IP,
 - 4 workery z dostępem do internetu przez NAT, każdy z dwoma interfejsami sieciowymi na dwóch różnych sieciach prywatnych (do każdego interfejsu przypisana pula adresów prywatnych).

Wszystkie instancje są typu `t2.large` i wykorzystują system RHEL9.4.

3. klaster kubernetesa (master i 4 workery) zdeployowany za pomocą Ansible (wykorzystujący pod spodem `kubeadm`).
4. spiderpool skonfigurowany za pomocą kubernetesowych manifestów i Ansible - dwie różne IP pool'e dla dwóch sieci prywatnych, korzystające z adresów przypisanych do interfejsów workerów.
5. pody z obrazem `busybox` umożliwiającym proste sprawdzenie działania sieci.

Na rys. 1 pokazano konfigurację klastra z dwoma workerami.

5 Konfiguracja środowiska

Konieczne jest systemu Linux. Należy posiadać klucz `ssh` typu `ed25519`. Należy mieć dostęp do konsoli AWS (np. przez AWS Academy) z lokalnej maszyny - w pliku `~/.aws/credentials` należy umieścić dane konfiguracyjne z konsoli.

Konieczne jest zainstalowanie następujących pakietów na lokalnej maszynie:

- ansible,
- terraform,
- helm,
- kubectl.

Dodatkowo po zainstalowaniu `ansible`, należy doinstalować moduły do `kubernetesa`:

```
ansible-galaxy collection install kubernetes.core
pip install "Jinja2<3.1"
pip install kubernetes
```

6 Instalacja

Należy sklonować repozytorium i uruchomić skrypt `provision_and_deploy.sh`, który przeprowadzi proces provisioningu instancji, deployment `kubernetesa`, `spiderpoola` i przykładowej aplikacji.

```
git clone https://github.com/gpiwar/suu_spiderpool_project.git
cd suu_spiderpool_project
./provision_and_deploy.sh
```

Skrypt ten uruchamia po kolei następujące komendy - w razie przerwania lub napotkania błędu można wznowić deployment wykorzystując następujące komendy:

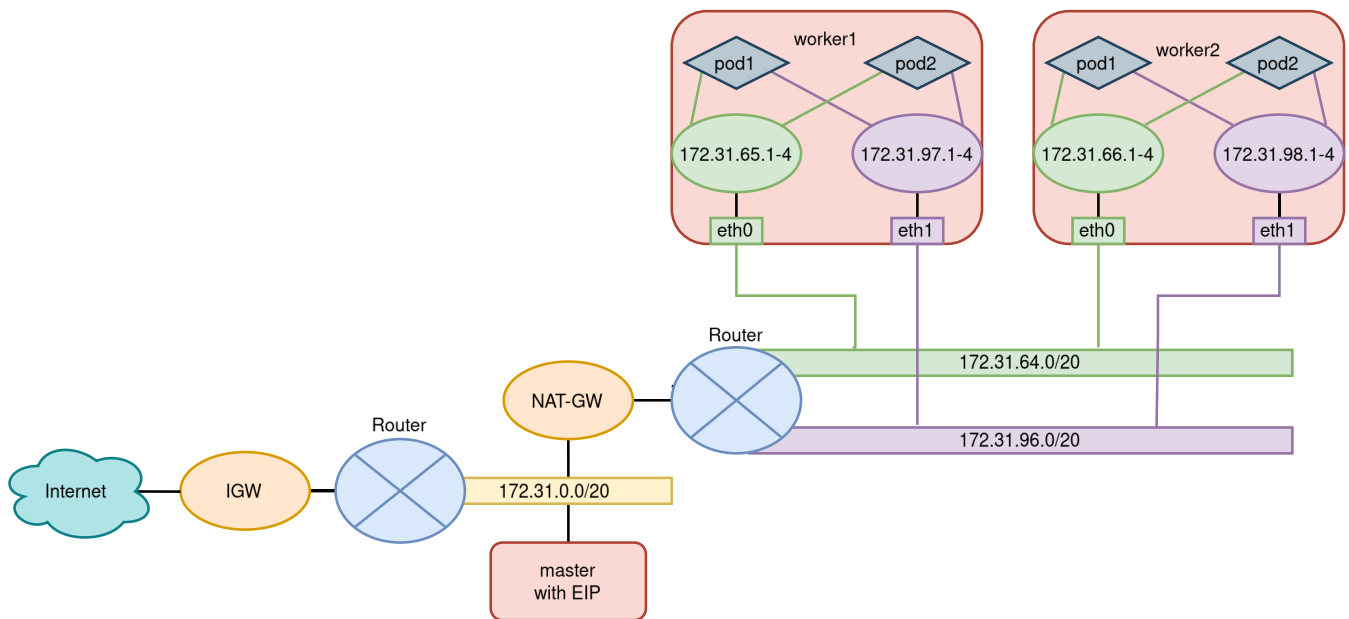
```
cd terraform
terraform init
terraform apply --auto-approve
cd -

ansible-playbook -i ansible/inventory/hosts.cfg ansible/pre-deploy.yml
ansible-playbook -i ansible/inventory/hosts.cfg ansible/deploy-k8s.yml
ansible-playbook -i ansible/inventory/hosts.cfg ansible/deploy-spiderpool.yml
ansible-playbook -i ansible/inventory/hosts.cfg ansible/deploy-apps.yml
```

7 Opis provisioningu i deploymentu

7.1 Terraform

Terraform provisionuje instancje `EC2` i tworzy sieci (w tym tablice routingu, `NAT` i `Internet Gateway`'e, security groupy). Na rysunku 1 przedstawiono konfigurację sieci i instancji `EC2`.



Rysunek 1: Schemat przedstawiający oczekiwaną konfigurację.

7.1.1 Sieć

Tworzone są 3 sieci:

- jedna publiczna 172.31.0.0/20,
- dwie prywatne 172.31.64.0/20 i 172.31.96.0/20.

Tworzone są interfejsy sieciowe, dla każdego z workerów po dwa (eth0 i eth1, z dwóch sieci prywatnych). Każdy interfejs ma pulę adresów IP.


7.1.2 Security groups

Master korzysta z security group'y z otwartym portem ingress dla ssh. Dla sieci prywatnych dla workerów jest stworzona osobna security group'a z otwartymi wszystkimi portami (ingress i egress) - jedynie w celu ułatwienia developmentu PoC.

7.1.3 Instancje

master posiada publiczny adres IP i zarządza całym klastrem kubernetes.

workery posiadają dwa interfejsy sieciowe - eth0 i eth1 podłączone do dwóch różnych sieci prywatnych - każdy z interfejsów ma przypisaną pulę adresów IP (rys. 2).

Private IPv4 addresses	VPC ID
📄 172.31.65.1	📄 vpc-08fe03c8afa24eb53 
📄 172.31.97.1	
Private IP DNS name (IPv4 only)	
📄 ip-172-31-65-1.ec2.internal	
IPv6 addresses	Secondary private IPv4 addresses
–	📄 172.31.65.2
	📄 172.31.65.3
	📄 172.31.65.4
	📄 172.31.97.2
	📄 172.31.97.3
	📄 172.31.97.4

Rysunek 2: Adresy IP przypisane do workera 1.

7.2 Playbooki Ansible

7.2.1 pre-deploy.yml

Dokonujemy konfiguracji połączenia z master i workerami. Dodajemy klucz ssh, ustawiamy hostname'y, dodajemy adresy IP do `/etc/hosts`. W wyniku tego kroku uzyskujemy dostęp do jeszcze nie skonfigurowanego klastra - punktem wejścia jest master z publicznym IP. Do zalogowania się należy użyć użytkownika `ec2-user`.

7.2.2 deploy-k8s.yml

Przygotowujemy node'y pod kubernetesa (wyłączenie SELinux i swap'u, włączenie modułów kernela). Instalujemy CRI-O i kubelet, następnie uruchamiamy je.

Następnie na masterze pobieramy obrazy dla kubernetesa i inicjalizujemy mastera za pomocą `kubeadm` z endpointem na publicznym IP mastera - umożliwi to dostęp do klastra za pomocą komend `kubectl` z poziomu lokalnej maszyny użytkownika. Po inicjalizacji mastera, dodajemy workery do klastra.

W wyniku tego kroku otrzymujemy działający klaster kubernetesa, co można sprawdzić wykonując komendy:

```
$ export KUBECONFIG="./kubeconfig"
$ kubectl get nodes
NAME        STATUS    ROLES    AGE   VERSION
master      Ready    control-plane   3m    v1.30.2
worker1     Ready    <none>        1m    v1.30.2
worker2     Ready    <none>        1m    v1.30.2
worker3     Ready    <none>        1m    v1.30.2
worker4     Ready    <none>        1m    v1.30.2
```

7.2.3 deploy-spiderpool.yml

Za pomocą helm-a instalujemy spiderpool-a. Dodajemy dwie konfiguracje ipvlan dla spiderpool-multusa - dla interfejsów eth0 i eth1. Następnie dodajemy pulę adresów IP. Powyższe jest równoznaczne ręcznemu zaplikowaniu manifestów:

```
kubectl apply -f manifests/SpiderMultusConfig.yaml
kubectl apply -f manifests/SpiderIPPool.yaml
```

W wyniku tego kroku uzyskujemy działającego spiderpoola z pulami adresów IP:

```
$ kubectl get spidermultusconfigs.spiderpool.spidernet.io -n kube-system
NAME          AGE
ipvlan-eth0   22m
ipvlan-eth1   22m
$ kubectl get spiderippools
NAME          VERSION  SUBNET          ALLOCATED-IP-COUNT  TOTAL-IP-COUNT
172-31-64-0   4        172.31.64.0/20   0                   16
172-31-96-0   4        172.31.96.0/20   0                   16
```

7.2.4 deploy-apps.yml

Deployujemy prostą aplikację z obrazem busybox, pozwalającą na sprawdzenie przydziału adresów IP do interfejsów podów.

```
kubectl apply -f manifests/Deployment-busybox.yaml
```

W wyniku tego uzyskujemy 4 pody, każdy z nich ma dwa interfejsy sieciowe: eth0 i net1.

```
$ kubectl get pods -owide
NAME                                READY  STATUS   AGE    IP             NODE
busybox-5dc5cb4bf6-6kv5d           1/1    Running  24m    172.31.65.3    worker1
busybox-5dc5cb4bf6-9hxxb           1/1    Running  24m    172.31.66.2    worker2
busybox-5dc5cb4bf6-9jsts           1/1    Running  24m    172.31.68.1    worker4
busybox-5dc5cb4bf6-9ml79           1/1    Running  24m    172.31.67.4    worker3
$ kubectl exec -it busybox-5dc5cb4bf6-6kv5d -- ip -4 addr show scope global
2: eth0@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc noqueue
    inet 172.31.65.3/20 brd 172.31.79.255 scope global eth0
        valid_lft forever preferred_lft forever
4: net1@veth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc noqueue
    inet 172.31.97.3/20 brd 172.31.111.255 scope global net1
        valid_lft forever preferred_lft forever
```

7.3 Dodatkowa konfiguracja

Nasz deployment charakteryzuje się dosyć dużą swobodą w konfiguracji kubernetesa i spiderpoola. Użytkownik może:

- dodać lub usunąć workery do klastra - plik `terraform/variables.tf`, sekcja `locals/workers`. Przykładowy 5. worker:

```
worker5 = {
  ami          = data.aws_ami.this.id
  instance_type = "t2.large"
  interfaces = [
    aws_network_interface.this["worker5_eth0"].id,
    aws_network_interface.this["worker5_eth1"].id
  ]
}
```

- dodać interfejsy do workerów i przypisać im adresy z sieci prywatnych - `terraform/variables.tf`, sekcja `locals/interfaces`. Przykładowy interfejs dla worker5:

```
worker5_eth1 = {
  subnet_id = aws_subnet.private[1].id
  ip_list   = [for i in range(1, 5) : "172.31.101.${i}"]
}
```

- zmodyfikować sieci prywatne - `terraform/variables.tf`, sekcja `private_subnet_cidrs`.

```
variable "private_subnet_cidrs" {
  type          = list(string)
  description    = "Private Subnet CIDRs"
  default        = ["172.31.64.0/20", "172.31.96.0/20"]
}
```

- dodać pule adresów przypisanych do workera do spiderpoola - `manifests/SpiderIPPool.yaml`:

```
kind: SpiderIPPool
metadata:
  name: 172-31-96-0
spec:
  subnet: 172.31.96.0/20
  ips:
    ...
    - 172.31.101.1-172.31.101.4
  gateway: 172.31.96.1
  default: true
  multusName: ["kube-system/ipvlan-eth1"]
```

8 Podsumowanie

Nasz projekt automatyzuje provisioning infrastruktury AWS i deployment klastra kubernetes z pluginem spiderpool. Wykorzystuje do tego paradygmat „infrastructure as a code”: Z wykorzystaniem Terraforma provisionuje instancje AWS EC2, tworzy i konfiguruje sieci na AWS VPC. Następnie za pomocą Ansible’a deployuje klaster kubernetesa na stworzonych wcześniej instancjach EC2. Na tak skonfigurowanym klastrze deployowany jest spiderpool z dwoma pulami adresów IP. Na sam koniec uruchamiana jest przykładowa aplikacja z czterema podami, z których każdy ma dwa interfejsy sieciowe z adresami IP z różnych sieci prywatnych.

W ramach testów wielokrotnie sprawdzono poprawność alokacji adresów IP w utworzonych podach.