Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Instytut Elektrotechniki Teoretycznej
i Systemów Informacyjno-Pomiarowych
Zakład Elektrotechniki Teoretycznej
i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria oprogramowania

Implementacja i testy wydajnosci środowiska Kubernetes na maszynach bezdyskowych

Krzysztof Nazarewski
nr albumu 123456

promotor
mgr inż. Andrzej Toboła

WARSZAWA 2018

Implementacja i testy wydajnosci środowiska Kubernetes na maszynach bezdyskowych

Streszczenie

Celem tej pracy inżynierskiej jest przybliżenie czytelnikowi zagadnień związanych z uruchamianiem systemu Kubernetes na maszynach bezdyskowych.

Zacznę od wyjaśnienia pojęcia systemu bezdyskowego oraz sposobu jego funkcjonowania na przykładzie sieci uczelnianej i wzorującego się na niej przygotowanie przeze mnie lokalnego środowiska.

Następnie opiszę problem izolacji i przydzielania zasobów systemowych na przykładzie wirtualnych maszyn, chroot i konteneryzacji.

W głównej części dokumentu przedstawię pojęcie orkiestrami kontenerami, w jaki sposób odnosi się do wcześniej postawionych problemów. Opiszę alternatywy Kubernetes, jego architekturę oraz sposoby uruchamiania. Na koniec spróbuję uruchomić Kubernetes na maszynach bezdyskowych, problemy z tym związane oraz przedstawię wyniki.

Słowa kluczowe: praca dyplomowa, LaTeX, jakość

Implementing and testing Kubernetes running on diskless machines

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi ac dolor scelerisque, malesuada ex vel, feugiat augue. Suspendisse dictum, elit efficitur vestibulum eleifend, mi neque accumsan velit, at ultricies ex lectus et urna. Pellentesque vel lorem turpis. Donec blandit arcu lacus, vitae dapibus tellus tempus et. Etiam orci libero, mollis in dapibus tempor, rutrum eget magna. Nullam congue libero non velit suscipit, vel cursus elit commodo. Praesent mollis augue quis lorem laoreet, condimentum scelerisque ex pharetra. Sed est ex, gravida a porta in, tristique ac nunc. Nunc at varius sem, sit amet consectetur velit.

Keywords: thesis, LaTeX, quality

WARSZAWA, 1 lutego 1234

POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa inżynierska pt. Implementacja i testy wydajnosci środowiska Kubernetes na maszynach bezdyskowych:

- została napisana przeze mnie samodzielnie,
- nie narusza niczyich praw autorskich,
- nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam, że przedłożona do obrony praca dyplomowa nie była wcześniej podstawą postępowania związanego z uzyskaniem dyplomu lub tytułu zawodowego w uczelni wyższej. Jestem świadom, że praca zawiera również rezultaty stanowiące własności intelektualne Politechniki Warszawskiej, które nie mogą być udostępniane innym osobom i instytucjom bez zgody Władz Wydziału Elektrycznego.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Krzysztof Nazarewski.....

Spis treści

Podziękowania

Dziękujemy bardzo serdecznie wszystkim, a w szczególności Rodzinom i Unii Europejskiej...

Zdolny Student i Pracowity Kolega

Rozdział 1

Test

The seminal work (Pizza i in. 2000)

Rozdział 2

Wstęp

Rozdział 3

Przegląd pojęć i systemów związanych z konteneryzacją

W związku z mnogością pojęć związanych z izolacją, konteneryzacją i zarządzaniem systemami komputerowymi zdecydowałem się w dużym skrócie przybliżyć najważniejsze pojęcia z tematem związane.

3.1 Konteneryzacja

Konteneryzacja jest sposobem izolacji aplikacji i jej zależności. Jest kolejnym krokiem po wirtualnych maszynach w dążeniu do minimalizacji kosztów ogólnych izolacji aplikacji.

W związku z działaniem na poziomie procesu w systemie operacyjnym konteneryzacja umożliwia izolację aplikacji stosunkowo niewielkim kosztem w porównaniu do wirtualizacji systemów operacyjnych (libvirt, VirtualBox itp.).

Wiodącym, ale nie jedynym, rozwiązaniem konteneryzacji jest Docker.

3.1.1 Open Container Initiative

Open Container Initiative[1] jest inicjatywą tworzenia i utrzymywania publicznych standardów związanych z formatem i uruchamianiem kontenerów.

Większość projektów związanych z konteneryzacją dąży do kompatybilności ze standardami OCI, m. in.: - Docker[2] - Kubernetes CRI-O[3] - Docker on

[1] `https://www.opencontainers.org/about`

[2] `https://blog.docker.com/2017/07/demystifying-open-container-initiative-oci-specifications`

[3] `https://github.com/kubernetes-incubator/cri-o`

FreeBSD[4] - Running CloudABI applications on a FreeBSD based Kubernetes cluster, by Ed Schouten (EuroBSDcon '17)[5]

3.1.2 Docker

Docker jest najstarszym i w związku z tym aktualnie najpopularniejszym rozwiązaniem problemu konteneryzacji.

Dobrym przeglądem alternatyw Dockera jest porównianie `rkt` (kolejna generacja Dockera) z innymi rozwiązaniami[6]

3.2 Kubernetes

Kubernetes[7] jest jednym z najpopularniejszych narzędzi orkiestracji kontenerami i jednocześnie tematem przewodnim tego dokumentu. Został stworzony przez Google na bazie ich wewnętrznego systemu Borg.

- Choosing the Right Containerization and Cluster Management Tool[8]

3.2.1 Alternatywne rozwiązania zarządzania kontenerami

Fleet

Fleet[9] jest nakładką na systemd[10] realizująca rozproszony system inicjalizacji systemów w systemie operacyjnym CoreOS

Kontenery są uruchamiane i zarządzane przez systemd, a stan przechowywany jest w etcd.

Aktualnie projekt kończy swój żywot na rzecz Kubernetesa i w dniu 1 lutego 2018, zostanie wycofany z domyślnej dystrybucji CoreOS. Nadal będzie dostępny w rejestrze pakietów CoreOS.

Docker Swarm

Docker Swarm[11] jest rozwiązaniem orkiestracji kontenerami od twórców samego Docker'a. Proste w konfiguracji, nie oferuje tak dużych możliwości jak niżej wymienione.

[4] `https://wiki.freebsd.org/Docker`
[5] `https://www.youtube.com/watch?v=akLa9L5O0NY`
[6] `https://coreos.com/rkt/docs/latest/rkt-vs-other-projects.html`
[7] `https://kubernetes.io/`
[8] `https://dzone.com/articles/choosing-the-right-containerization-and-cluster-management-tool`
[9] `https://github.com/coreos/fleet`
[10] `https://www.freedesktop.org/wiki/Software/systemd/`
[11] `https://docs.docker.com/engine/swarm/`

Nomad

HasiCorp Nomad vs Kubernetes[12]

Mesos

Apache Mesos[13] jest najbardziej zaawansowanym i najlepiej skalującym się rozwiązaniem orkiestracji kontenerami. Jest również najbardziej skomplikowanym i trudnym w zarządzaniu rozwiązaniem.

- An Introduction to Mesosphere[14]

3.3 Zarządzanie Kubernetes z linii komend

3.3.1 kubeadm

kubeadm[15] jest narzędziem pozwalającym na niskopoziomowe zarządzanie klastrem Kubernetes. Stąd trendem jest bazowanie na kubeadm przy tworzeniu narzędzi z wyższym poziomem abstrakcji.

- Install with kubadm[16]

3.3.2 Kubespray

- kubespray[17]
- zestaw skryptów Ansible konfigurujących klaster na jednym z wielu systemów operacyjnych
- dąży do zostania tzw.
  Operatorem[18] korzystającym z kubeadm

3.3.3 OpenShift Ansible

Konfiguracja OpenShift Origin realizowana jest zestawem skryptów Ansible'owych rozwijanych jako projekt openshift-ansible[19].

[12] https://www.nomadproject.io/intro/vs/kubernetes.html
[13] http://mesos.apache.org/
[14] https://www.digitalocean.com/community/tutorials/an-introduction-to-mesosphere
[15] https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm/
[16] https://kubernetes.io/docs/setup/independent/install-kubeadm/
[17] https://github.com/kubernetes-incubator/kubespray
[18] https://github.com/kubernetes-incubator/kubespray/blob/master/docs/comparisons.md
[19] https://github.com/openshift/openshift-ansible

3.3.4 Canonical distribution of Kubernetes

Jest to prosta w instalacji dystrybucja Kubernetes. Niestety wymaga infrastruktury chmurowej do uruchomienia klastra składającego się z więcej niż jednego węzła.

Opcja bare-metal, która by mnie interesowała nadal wymaga działającego środowiska Metal as a Service[20].

W związku z powyższym nie będę dalej zajmował się tym narzędziem.

Przydatne materiały: - Juju Charm the Canonical distribution of Kubernetes[21] - Install Kubernetes with conjure-up[22] - Kubernetes the not so easy way[23] opisuje instalację lokalnego klastra.

3.3.5 Eksperymentalne i deprekowane rozwiązania

- Fedora via Ansible[24] deprekowane na rzecz kubespray
- Rancher Kubernetes Installer[25] jest eksperymentalnym rozwiązaniem wykorzystywanym w Rancher 2.0,

kubespray-cli

Jest to narzędzie ułatwiające korzystanie z `kubespray`. Według użytkowników oficjalnego Slacka kubespray[26] `kubespray-cli` jest deprekowane.

3.4 Graficzne nakładki na Kubernetes

3.4.1 Kubernetes Dashboard

Kubernetes Dashboard[27] jest wbudowanym interfejsem graficznym klastra Kubernetes. Umożliwia monitorowanie i zarządzanie klastrem w ramach funkcjonalności samego Kubernetes.

---

[20]

[21] `https://jujucharms.com/canonical-kubernetes/`

[22] `https://tutorials.ubuntu.com/tutorial/install-kubernetes-with-conjure-up`

[23] `https://insights.ubuntu.com/2017/10/12/kubernetes-the-not-so-easy-way/`

[24] `https://kubernetes.io/docs/getting-started-guides/fedora/fedora_ansible_config/`

[25] `http://rancher.com/announcing-rke-lightweight-kubernetes-installer/`

[26] `https://kubernetes.slack.com/messages/kubespray`

[27] `https://github.com/kubernetes/dashboard`

3.4.2 Rancher

Rancher[28] jest platformą zarządzania kontenerami umożliwiającą między innymi zarządzanie klastrem Kubernetes. Od wersji 2.0 twórcy skupiają się tylko i wyłącznie na zarządzaniu Kubernetes porzucając inne rozwiązania.

3.4.3 OpenShift by Red Hat

OpenShift jest komercyjną usługą typu PaaS (Platform as a Service), od wersji 3 skupia się na zarządzaniu klastrem Kubernetes.

Rdzeniem projektu jest open sourcowy OpenShift Origin[29] konfigurowany przez OpenShift Ansible.

- OpenShift Origin vs Kubernetes[30]
- The Differences Between Kubernetes and OpenShift[31]
- Demo konsoli[32] (niestety po hebrajsku)

3.4.4 DC/OS

Datacenter Operating System[33] jest częścią Mesosphere[34] i Mesosa. Niedawno został rozszerzony o Kubernetes[35] jako alternatywnego (w stosunku do Marathon[36]) systemu orkiestracji kontenerami.

[28] https://rancher.com/

[29] https://github.com/openshift/origin

[30] https://www.reddit.com/r/devops/comments/59ql4r/openshift_origin_vs_kubernetes/

[31] https://medium.com/levvel-consulting/the-differences-between-kubernetes-and-openshift-ae7

[32] https://youtu.be/-mFovK19aB4?t=6m54s

[33] https://dcos.io/

[34] https://mesosphere.com/

[35] https://mesosphere.com/blog/kubernetes-dcos/

[36] https://mesosphere.github.io/marathon/

Rozdział 4

Kubernetes

Materiały: - https://jvns.ca/categories/kubernetes/ - https://github.com/kelseyhightower/kube
the-hard-way - https://www.youtube.com/watch?v=4-pawkiazEg

4.1 Administracja, a korzystanie z klastra

Administracja klastrem polega na jego skonfigurowaniu i dopasowaniu komponentów, aby umożliwić korzystanie z niego.

Korzystanie z klastra polega na uruchamianiu aplikacji na klastrze.

W pracy inżynierskiej skupiam się przede wszystkim na kwestiach związanych z administracja klastrem.

4.2 Architektura

4.2.1 Składowe kontrolujące klaster

- etcd - przechowywanie stanu klastra
- kube-apiserver - interfejs konfiguracyjny klastra (zarówno wewnętrzny jak i zewnętrzny), prowadzi interakcję tylko ze stanem klastra w etcd
- kube-scheduler - proces decydujący na którym węźle klastra uruhamiać Pody (na podstawie dostępnych zasobów, obecnego obciążenia itp.). W skrócie zarządza popytem i podażą na zasoby klastra.
- kube-controller-manager - kontroler klastra dążący do doprowadzenia obecnego stanu klastra do pożądanego na podstawie informacji z kube-apiserver

4.2.2 Składowe workera

- kubelet - monitoruje i kontroluje stan pojedynczego węzła. Na przykład restartuje Pod, który przestał działać na tym samym węźle.
- kube-proxy - proxy i load balancer odpowiedzialny za przekierowanie ruchu do odpowiedniego kontenera
- cAdvisor - monitoruje zużycie zasobów i wydajność kontenerów w ramach jednego węzła

4.2.3 Komunikacja sieciowa

Materiały:

- https://www.slideshare.net/weaveworks/kubernetes-networking-78049891
- https://jvns.ca/blog/2016/12/22/container-networking/
- `https://medium.com/@anne_e_currie/kubernetes-aws-networking` ↪ `-for-dummies-like-me-b6dedeeb95f3`

Info:

- Kubernetes **zakłada**, że każdy Pod ma swój własny adres IP, ale w żadnym stopniu nie zajmuje się konfiguracją i przedziałem adresów IP
- Kubernetes polega na zewnętrznych rozwiązaniach zajmujących się przydzielaniem adresów IP

4 rodzaje komunikacji sieciowej:

1. wewnątrz Podów (localhost)
2. między Podami (trasowanie lub nakładka sieciowa - overlay network)
3. między Podami i Serwisami (kube-proxy)
4. świata z Serwisami

W skrócie:

- Kubernetes uruchamia Pody, które implementują Serwisy,
- Pody potrzebują Sieci Podów - trasowanych lub nakładkę sieciową,
- Sieć Podów jest sterowana przez CNI (Container Network Interface),
- Klient łączy się do Serwisów przez wirtualne IP Klastra,
- Kubernetes ma wiele sposobów na wystawienie Serwisów poza klaster,

Rozdział 5

Systemy bezdyskowe

Maszyny bezdyskowe jak nazwa wskazuje nie posiadają lokalnego medium trwałego przechowywania informacji. W związku z tym wszystkie informacje są przechowywane w pamięci RAM komputera i są tracone w momencie restartu maszyny.

System operacyjny musi wspierać uruchamianie się w takim środowisku. Wiele systemów nie wspiera tego trybu operacyjnego zakładając obecność dysku twardego w maszynie.

W niektórych przypadkach mimo braku domyślnego wsparcia istnieje możliwość przygotowania własnego obrazu systemu operacyjnego wspierającego ten tryb pracy:

- Fedora Atomic Host[1].

Potencjalnymi rozwiązaniami problemu przechowywania stanu maszyn bezdyskowych mogą być:

- przydziały NFS
- replikacja ZFS[2]
- przechowywanie całego stanu w cloud-init

5.1 Proces uruchamiania maszyny bezdyskowej

Na uruchamianie maszyn bezdyskowych w protokole PXE składają się 3 podstawowe elementy:

[1] `https://www.projectatomic.io/blog/2015/05/building-and-running-live-atomic/`

[2] `https://arstechnica.com/information-technology/2015/12/rsync-net-zfs-replication-to-the-cloud-is-finally-here-and-its-fast/`

1. serwer DHCP, np. isc-dhcp-server lub dnsmasq
2. firmware wspierające PXE, np. iPXE
3. serwer plików (np. TFTP, HTTP, NFS) i/lub sieciowej pamięci masowej (np. iSCSI)

Rozdział 6

Przegląd systemów operacyjnych

Ze względu na obszerność i niejednoznaczność tematu cloud-init rozdział rozpocznę od jego omówienia.

Wszystkie moduły Kubernetes'a są uruchamiane w kontenerach, więc dwoma podstawowymi wymaganiami systemu operacyjnego są:

- możliwość instalacji i uruchomienia Dockera,
- wsparcie wybranego narzędzia konfigurującego system do działania w klastrze Kubernetes,

Dodatkowe wymagania związane z naszym przypadkiem użycia:

- zdalny dostęp SSH lub możliwość konfiguracji automatycznego dołączania do klastra Kubernetes,
- wsparcie dla środowiska bezdyskowego,
- możliwość bootu PXE,

Podstawowe wyznaczniki:

- sposób konfiguracji maszyny,
- rozmiar minimalnego działającego systemu spełniającego wszystkie wymagania,

6.1 Konfigurator cloud-init

cloud-init[1] jest standardem oraz implementacją konfiguratora kompatybilnego z wieloma systemami operacyjnymi przeznaczonymi do działania w chmurze.

[1] `https://cloud-init.io/`

Standard polega na dostarczeniu pliku konfiguracyjnego w formacie YAML[2] w trakcie lub tuż po inicjalizacji systemu operacyjnego.

Główną zaletą cloud-init jest tworzenie automatycznej i jednorodnej konfiguracji bazowych systemów operacyjnych w środowiskach chmurowych, czyli częstego podnoszenia nowych maszyn.

6.1.1 Dostępne implementacje

cloud-init

Referencyjny cloud-init zaimplementowany jest w Pythonie, co częściowo tłumaczy duży rozmiar obrazów przeznaczonych dla chmury. Po najmniejszych obrazach Pythona dla Dockera[3] (python:alpine - 89MB i python2:alpine - 72 MB) wnioskuję, że nie istnieje mniejsza dystrybucja Pythona.

```
1 docker pull python:2-alpine > /dev/null
2 docker pull python:alpine > /dev/null
3 docker images | grep alpine
```

Dodatkowe materiały:

- Wywiad z developerem cloud-init[4]

coreos-cloudinit

coreos-cloudinit[5] jest częściową implementacją standardu w języku Go przez twórców CoreOS Niestety rok temu przestał być rozwijany[6] i wychodzi z użytku.

RancherOS + coreos-cloudinit

rancher cloud-init[7] jest jest spadkobiercą[8] coreos-cloudinit rozwijanym przez zespół RancherOS, na jego potrzeby.

[2] http://yaml.org/
[3] https://hub.docker.com/_/python/
[4] https://www.podcastinit.com/cloud-init-with-scott-moser-episode-126
[5] https://github.com/coreos/coreos-cloudinit
[6] https://github.com/coreos/coreos-cloudinit/commit/3460ca4414fd91de66cd581d997bf453fd895b67
[7] http://rancher.com/docs/os/latest/en/configuration/
[8] https://github.com/rancher/os/commit/e2ed97648ad63455743ebc16080a82ee47f8bb0c

clr-cloud-init

clr-cloud-init[9] jest wewnętrzną implementacją standardu dla systemu ClearLinux. Powstała z chęci optymalizacji standardu pod ClearLinux oraz pozbycia się zależności referencyjnej implementacji od Python'a.

6.2 CoreOS

CoreOS[10] jest pierwszą dystrybucją linuxa dedykowaną zarządzaniu kontenerami. Zawiera dużo narzędzi dedykowanych klastrowaniu i obsłudze kontenerów, w związku z tym zajmuje 342 MB.

Czysta instalacja zajmuje około 600 MB pamięci RAM

6.2.1 Konfiguracja

Konfiguracja przez Container Linux Config[11] transpilowany do Ignition[12]. Transpiler konwertuje ogólną konfigurację na przygotowaną pod konkretne chmury (AWS, GCE, Azure itp.). Dyskwalifikującą wadą tego typu konfiguracji jest brak wsparcia transpilatora dla systemów z rodziny BSD

Poprzednikiem Ignition jest coreos-cloudinit.

6.3 RancherOS

RancherOS[13] jest systemem operacyjnym, w którym tradycyjny system inicjalizacji został zastąpiony trzema poziomami Dockera[14]:

- `bootstrap_docker` działający w initramie, czyli przygotowujący system,
- `system-docker` zastępuje tradycyjnego inita, zarządza wszystkimi programami systemowymi,
- `docker` standardowy docker, interakcja z nim nie może uszkodzić działającego systemu,

Jego głównymi zaletami są mały rozmiar plików startowych (45 MB) oraz prostota konfiguracji.

[9] `https://clearlinux.org/blogs/announcing-clr-cloud-init`
[10] `https://coreos.com/`
[11] `https://coreos.com/os/docs/latest/provisioning.html`
[12] `https://coreos.com/ignition/docs/latest/`
[13] `https://rancher.com/rancher-os/`
[14] `http://rancher.com/docs/os/latest/en/configuration/docker/`

Czysta instalacja zajmuje około 700 MB pamięci RAM.

W związku z bugiem w systemie RancherOS nie zawsze czyta cloud-config[15], więc na chwilę obecną odrzucam ten system operacyjny w dalszych rozważaniach.

6.3.1 Konfiguracja

RancherOS jest konfigurowany przez własną wersję coreos-cloudinit.

Znaczną przewagą nad oryginałem jest możliwość sekwencyjnego uruchamiania dowolnej ilości plików konfiguracyjnych.

Minimalna konfiguracja pozwalająca na zalogowanie:

```
1 #cloud-config
2 ssh_authorized_keys:
3   - ssh-rsa AAAAB3N...
```

Generuję ją poniższym skryptem na podstawie komendy `ssh-add -L`:

```
1 echo "#cloud-config
2 ssh_authorized_keys:
3 $(ssh-add -L | sed 's/^/  - /g')" > ${cc_dir}/ssh.yml
```

Przydatne jest wyświetlenie kompletnej konfiguracji komendą `ros config export --full`[16].

6.4 Project Atomic

Project Atomic[17] jest grupą podobnie skonfigurowanych systemów operacyjnych dedykowaną środowiskom cloud i kontenerom.

Dystrybucje Project Atomic nazywają się Atomic Host, dostępne są następujące warianty:

- Red Hat Atomic Host[18]
- CentOS Atomic Host[19]
- Fedora Atomic Host[20]

[15] https://github.com/rancher/os/issues/2204
[16] https://forums.rancher.com/t/good-cloud-config-reference/5238/3
[17] https://www.projectatomic.io/
[18] https://www.redhat.com/en/resources/enterprise-linux-atomic-host-datasheet
[19] https://wiki.centos.org/SpecialInterestGroup/Atomic/Download/
[20] https://getfedora.org/atomic/download/

Żadna z dystrybucji domyślnie nie wspiera bootowania bezdyskowego, więc nie zgłębiałem dalej tematu.

Atomic Host są konfigurowane systemem oficjalną implementacją cloud-inita.

6.5 Alpine Linux

Alpine Linux[21] jest minimalną dystrybucją Linuxa bazowaną na musl-libc i busybox.

Wygląda bardzo obiecująco do naszych zastosowań, ale ze względu na buga w procesie inicjalizacji systemu aktualnie nia ma możliwości uruchomienia systemu operacyjnego w trybie bezdyskowym.

Alpine Linux może być skonfigurowany przez Alpine Backup[22] lub Alpine Configuration Framework[23].

6.6 ClearLinux

ClearLinux[24] jest dystrybucją linuxa wysoko zoptymalizowaną pod procesory Intel.

Poza intensywną optymalizacją ciekawy w tej dystrybucji jest koncept `bundle` zamiast standardowych pakietów systemowych. Żaden z bundli nie może zostać zaktualizowany oddzielnie, w zamian cały system operacyjny jest aktualizowany na raz ze wszystkimi bundlami. Znacznie ułatwia to zarządzanie wersjami oprogramowania i stanem poszczególnych węzłów sieci komputerowej.

Czysta instalacja z Dockerem i serwerem SSH również zajmuje 700 MB w RAMie więc nie odbiega od innych dystrybucji.

Ogromnym minusem jest trudność w nawigacji dokumentacja systemu operacyjnego.

Materiały dodatkowe:

- 6 key points about Intel's hot new Linux distro[25]

[21] `https://alpinelinux.org/`

[22] `https://wiki.alpinelinux.org/wiki/Alpine_local_backup`

[23] `http://wiki.alpinelinux.org/wiki/Alpine_Configuration_Framework_Design`

[24] `https://clearlinux.org/`

[25] `https://www.infoworld.com/article/3159658/linux/6-key-points-about-intels-hot-new-linux-distro.html`

6.7 Wnioski

Systemy operacyjne przeznaczone do działania w chmurze są bardzo do siebie zbliżone i wybór dystrybucji nie ma wielkiego znaczenia dla działania bezdyskowego klastra Kubernetes.

W dalszej części skupię się na wykorzystaniu CoreOSa jako systemu od początku wspierającego Kubernetes.

Rozdział 7

Konfiguracja klastra Kubernetes

Najpopularniejszym rozwiązaniem konfiguracji klastra Kubernetes jest kops[1], ale jak większość rozwiązań zakłada uruchomienie w środowiskach chmurowych, PaaS lub IaaS. W związku z tym nie ma żadnego zastosowania w tej pracy inżynierskiej.

7.1 Rancher 2.0

Wygodne narzędzie do uruchamiania i monitorowania klastra Kubernetes, ale wymaga interakcji użytkownika. Wersja 2.0 (obecnie w fazie alpha) oferuje lepszą integrację z Kubernetes całkowicie porzucając inne platformy.

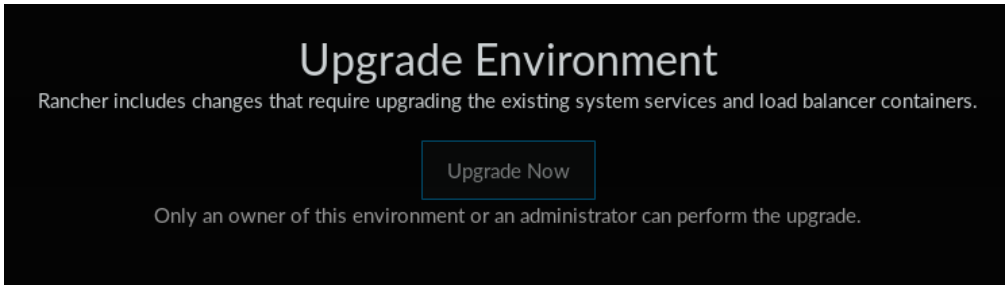
```
1 #rancher_version=latest
2 rancher_version=preview
3 docker run --rm --name rancher -d -p 8080:8080 rancher/server:$
    ↪ {rancher_version}
```

Najpierw należy zalogować się do panelu administracyjnego Ranchera i przeprowadzić podstawową konfigurację (adres Ranchera + uzyskanie komendy).

Następnie w celu dodania węzła do klastra wystarczy wywołać jedną komendę udostępnioną w panelu administracyjnym Ranchera na docelowym węźle, jej domyślny format to:

```
1 wersja_agenta=v1.2.9
2 ip_ranchera=192.168.56.1
3 skrypt=B52944BEFAA613F0CE90:1514678400000:
    ↪ E2yB6KfxzSix4YHti39BTw5RbKw
4
```

[1] https://github.com/kubernetes/kops



Rysunek 7.1: Błąd pt. Upgrade Environment

```
5 sudo docker run --rm --privileged \
6   -v /var/run/docker.sock:/var/run/docker.sock \
7   -v /var/lib/rancher:/var/lib/rancher \
8   rancher/agent:${wersja_agenta} \
9   http://${ip_ranchera}:8080/v1/scripts/${skrypt}
```

W ciągu 2 godzin przeglądu nie udało mi się zautomatyzować procesu uzyskiwania ww. komendy.

Następnie w cloud-config'u RancherOS'a możemy dodać ww. komendę w formie:

```
1 #cloud-config
2 runcmd:
3 - docker run --rm --privileged -v /var/run/docker.sock:/var/run
    ↪ /docker.sock -v /var/lib/rancher:/var/lib/rancher
    ↪ rancher/agent:v1.2.9 http://192.168.56.1:8080/v1/scripts
    ↪ /...
```

Od wersji 2.0 umożliwia połączenie się z istniejącym klastrem:

```
1 kubectl apply -f http://192.168.56.1:8080/v3/scripts/303
    ↪ F60E1A5E186F53F3F:1514678400000:
    ↪ wstQFdHpOgHqKahoYdmsCXEWMW4.yaml
```

W wersji `v2.0.0-alpha10` losowo pojawia się błąd Upgrade Environment[2].

7.1.1 Wnioski

Rancher na chwilę obecną (styczeń 2018) jest bardzo wygodnym, ale również niestabilnym rozwiązaniem.

[2] `https://github.com/rancher/rancher/issues/10396`

Ze względu na brak stabilności odrzucam Ranchera jako rozwiązanie problemu uruchamiania klastra Kubernetes.

7.2 kubespray-cli

Z powodu błędu[3] logiki narzędzie nie radzi sobie z brakiem Python'a na domyślnej dystrybucji CoreOS'a, mimo że sam `kubespray` radzi sobie z nim świetnie. Do uruchomienia na tym systemie potrzebne jest ręczne wywołanie roli `bootstrap-os`[4] z `kubespray` zanim przystąpimy do właściwego deploy'u.

```
1  #!/usr/bin/env bash
2  set -e
3
4  #pip2 install ansible kubespray
5  get_coreos_nodes() {
6    for node in $@
7    do
8      echo -n node1[
9      echo -n ansible_host=${node},
10     echo -n bootstrap_os=coreos,
11     echo -n ansible_user=core,
12     echo -n ansible_default_ipv4.address=${node}
13     echo ]
14   done
15 }
16
17 NODES=($(get_coreos_nodes 192.168.56.{10,12,13}))
18 echo NODES=${NODES[@]}
19 kubespray prepare -y --nodes ${NODES[@]}
20 cat > ~/.kubespray/bootstrap-os.yml << EOF
21 - hosts: all
22   become: yes
23   gather_facts: False
24   roles:
25   - bootstrap-os
26 EOF
27
```

[3] https://github.com/kubespray/kubespray-cli/issues/120

[4] https://github.com/kubernetes-incubator/kubespray/blob/master/roles/bootstrap-os/tasks/main.yml

```
(cd ~/.kubespray; ansible-playbook -i inventory/inventory.cfg
    ↪ bootstrap-os.yml)
kubespray deploy -y --coreos
```

Wykrzacza się na kroku czekania na uruchomienie `etcd` ponieważ oczekuje połączenia na NATowym interfejsie z adresem `10.0.3.15` zamiast host network z adresem `192.168.56.10`, stąd `ansible_default_ipv4.address`.

7.2.1 Wnioski

W trakcie testowania okazało się, że `kubespray-cli` nie jest aktywnie rozwiązane i stało się niekompatybilne z samym Kubespray. W związku z tym uznaję `kubespray-cli` za nie mające zastosowania w tej pracy inżynierskiej.

7.3 kubespray

Kod znajduje się w moim repozytorium kubernetes-cluster[5].

```
#!/usr/bin/env bash
cd $(dirname "$(readlink -f "$0")")
source ./setup-cluster-vars
cd ${dir}

 based on https://github.com/kubernetes-incubator/kubespray/
    ↪ blob/master/docs/getting-started.md#building-your-own-
    ↪ inventory
cp -r inventory -T ${inventory}
python3 contrib/inventory_builder/inventory.py ${IPS[@]}

cat > ${inventory}/group_vars/all.yml << EOF
bootstrap_os: coreos
#loadbalancer_apiserver:
#  address: 0.0.0.0
#  port: 8080
kube_basic_auth: true
kubeconfig_localhost: true
kubectl_localhost: true
download_run_once: True
EOF

```

[5] https://github.com/nazarewk/kubernetes-cluster

```
21 ansible-playbook -i ${inventory}/inventory.cfg cluster.yml -b -
       ↪ v
22
23 echo Staring kubectl proxy
24 echo http://localhost:8001/api/v1/namespaces/kube-system/
       ↪ services/https:kubernetes-dashboard:/proxy/#!/login
25 ./kubectl proxy
```

7.3.1 Kubernetes Dashboard

Dostęp do Dashboardu najprościej można uzyskać:

1. nadanie wszystkich uprawnień roli `kubernetes-dashboard`[6]
2. Wejście na http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login
3. Kliknięcie skip

```
1 #!/bin/sh
2 cd $(dirname "$(readlink -f "$0")")/..
3
4 bin/kubectl create -f dashboard-admin.yml
5 xdg-open http://localhost:8001/api/v1/namespaces/kube-system/
       ↪ services/https:kubernetes-dashboard:/proxy/#!/login
```

Linki:

- https://github.com/kubernetes/dashboard/wiki/Access-control
- https://github.com/kubernetes-incubator/kubespray/blob/master/docs/getting-started.md#accessing-kubernetes-dashboard

7.3.2 Napotkane błędy

Błąd przy ustawieniu `loadbalancer_apiserver.address` na `0.0.0.0`:

```
1 TASK [kubernetes-apps/cluster_roles : Apply workaround to allow
       ↪  all nodes with cert O=system:nodes to register]
       ↪ ************************************
2 Wednesday 17 January 2018  22:22:59 +0100 (0:00:00.626)
       ↪ 0:08:31.946 *****
```

[6] `https://github.com/kubernetes/dashboard/wiki/Access-control#admin-privileges`

```
3 fatal: [node2]: FAILED! => {"changed": false, "msg": "error
    ↪ running kubectl (/opt/bin/kubectl apply --force --
    ↪ filename=/etc/kubernetes/node-crb.yml) command (rc=1):
    ↪ Unable to connect to the server: http: server gave HTTP
    ↪ response to HTTPS client\n"}
4 fatal: [node1]: FAILED! => {"changed": false, "msg": "error
    ↪ running kubectl (/opt/bin/kubectl apply --force --
    ↪ filename=/etc/kubernetes/node-crb.yml) command (rc=1):
    ↪ Unable to connect to the server: http: server gave HTTP
    ↪ response to HTTPS client\n"}
```

7.4 OpenShift Origin

Według dokumentacji[7] są dwie metody uruchamiania serwera, w Dockerze i na systemie linux.

```
1 # https://docs.openshift.org/latest/getting_started/
    ↪ administrators.html#installation-methods
2 docker run -d --name "origin" \
3   --privileged --pid=host --net=host \
4   -v /:/rootfs:ro \
5   -v /var/run:/var/run:rw \
6   -v /sys:/sys \
7   -v /sys/fs/cgroup:/sys/fs/cgroup:rw \
8   -v /var/lib/docker:/var/lib/docker:rw \
9   -v /var/lib/origin/openshift.local.volumes:/var/lib/origin/
      ↪ openshift.local.volumes:rslave \
10  openshift/origin start --public-master
```

Dodałem opcję `--public-master` aby uruchomić konsolę webową

7.4.1 Korzystanie ze sterownika systemd zamiast domyślnego cgroupfs

Większość dystrybucji linuxa (np. Arch, CoreOS, Fedora, Debian) domyślnie nie konfiguruje sterownika cgroup Dockera i korzysta z domyślnego `cgroupfs`.

Typ sterownika cgroup można wyświetlić komendą `docker info`:

[7] `https://docs.openshift.org/latest/getting_started/administrators.html`

```
$ docker info | grep -i cgroup
    ↪
    ↪ :(
Cgroup Driver: systemd
```

OpenShift natomiast konfiguruje Kubernetes do korzystania z `cgroup` przez `systemd`. Kubelet przy starcie weryfikuje zgodność silników cgroup, co skutkuje niekompatybilnością z domyślną konfiguracją Dockera[8], czyli poniższym błędem:

```
F0120 19:18:58.708005   25376 node.go:269] failed to run
    ↪ Kubelet: failed to create kubelet: misconfiguration:
    ↪ kubelet cgroup driver: "systemd" is different from
    ↪ docker cgroup driver: "cgroupfs"
```

Problem można rozwiązać dopisując `--exec-opt native.cgroupdriver =systemd` do linii komend `dockerd` (zwykle w pliku `docker.service`). Dla przykładu w Arch Linux'ie zmiana wygląda następująco:

```
$ cp /usr/lib/systemd/system/docker.service /etc/systemd/system
    ↪ /docker.service
$ vim /etc/systemd/system/docker.service
$ diff /usr/lib/systemd/system/docker.service /etc/systemd/
    ↪ system/docker.service
13c13
< ExecStart=/usr/bin/dockerd -H fd://
---
> ExecStart=/usr/bin/dockerd -H fd:// --exec-opt native.
    ↪ cgroupdriver=systemd
```

7.4.2 Próba uruchomienia serwera na Arch Linux

Po wystartowaniu serwera zgodnie z dokumentacją OpenShift Origin i naprawieniu błędu z konfiguracją cgroup przeszedłem do kolejnego kroku Try It Out[9]:

0. Uruchomienie shella na serwerze:

```
$ docker exec -it origin bash
```

[8] https://github.com/openshift/origin/issues/14766

[9] https://docs.openshift.org/latest/getting_started/administrators.html#try-it-out

1. Logowanie jako testowy użytkownik:

```
$ oc login
Username: test
Password: test
```

2. Stworzenie nowego projektu:

```
$ oc new-project test
```

3. Pobranie aplikacji z Docker Huba:

```
$ oc tag --source=docker openshift/deployment-example:v1
    ↪ deployment-example:latest
```

4. Wystartowanie aplikacji:

```
$ oc new-app deployment-example:latest
```

5. Odczekanie aż aplikacja się uruchomi:

```
$ watch -n 5 oc status
In project test on server https://192.168.0.87:8443

svc/deployment-example - 172.30.52.184:8080
  dc/deployment-example deploys istag/deployment-example:latest
    deployment #1 failed 1 minute ago: config change
```

Niestety nie udało mi się przejść kroku 5, więc próba uruchomienia OpenShift Origin na Arch Linux zakończyła się niepowodzeniem.

7.4.3 Próba uruchomienia serwera na Fedora Atomic Host w VirtualBox'ie

Maszynę z najnowszym Fedora Atomic Host uruchomiłem za pomocą poniższego `Vagrantfile`:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "fedora/27-atomic-host"
  config.vm.box_check_update = false
  config.vm.network "forwarded_port", guest: 8443, host: 18443,
     ↪ host_ip: "127.0.0.1"
  config.vm.network "forwarded_port", guest: 8080, host: 18080,
     ↪ host_ip: "127.0.0.1"
  config.vm.provider "virtualbox" do |vb|
    vb.gui = false
    vb.memory = "8192"
  end
  config.vm.provision "shell", inline: <<-SHELL
  SHELL
end
```

```
$ vagrant up
$ vagrant ssh
$ sudo docker run -d --name "origin" \
  --privileged --pid=host --net=host \
  -v /:/rootfs:ro \
  -v /var/run:/var/run:rw \
  -v /sys:/sys \
  -v /sys/fs/cgroup:/sys/fs/cgroup:rw \
  -v /var/lib/docker:/var/lib/docker:rw \
  -v /var/lib/origin/openshift.local.volumes:/var/lib/origin/
     ↪ openshift.local.volumes:rslave \
  openshift/origin start
```

Kroki 0-4 były analogiczne do uruchamiania na Arch Linux, następnie:

5. Odczekanie aż aplikacja się uruchomi i weryfikacja działania:

```
$ watch -n 5 oc status
In project test on server https://10.0.2.15:8443

svc/deployment-example - 172.30.221.105:8080
  dc/deployment-example deploys istag/deployment-example:latest
    deployment #1 deployed 3 seconds ago - 1 pod
$ curl http://172.30.221.105:8080 | grep v1
```

```
8 <div class="box"><h1>v1</h1><h2></h2></div>
```

6. Aktualizacja, przebudowanie i weryfikacja działania aplikacji:

```
1 $ oc tag --source=docker openshift/deployment-example:v2
    ↪ deployment-example:latest
2 Tag deployment-example:latest set to openshift/deployment-
    ↪ example:v2.
3 $ watch -n 5 oc status
4 In project test on server https://10.0.2.15:8443
5
6 svc/deployment-example - 172.30.221.105:8080
7   dc/deployment-example deploys istag/deployment-example:latest
8     deployment #2 running for 8 seconds - 1 pod
9     deployment #1 deployed 8 minutes ago - 1 pod
10 $ curl -s http://172.30.221.105:8080 | grep v2
11 <div class="box"><h1>v2</h1><h2></h2></div>
```

7. Nie udało mi się uzyskać dostępu do panelu administracyjnego OpenShift:

```
1 $ curl -k 'https://localhost:8443/console/'
2 missing service (service "webconsole" not found)
3 missing route (service "webconsole" not found)
```

W internecie nie znalazłem żadnych informacji na temat tego błędu. Próbowałem również uzyskać pomoc na kanale `#openshift` na `irc.freenode.net`:

```
1 [17:29] <nazarewk> i'm trying to evaluate openshift origin, but
    ↪  when i start server and try to go to https://localhost
    ↪ :8443 i'm getting missing service (service "webconsole"
    ↪ not found)
2 [17:30] <nazarewk> any ideas how do i get into the console?
3 [17:40] <meta4knox> In your terminal, type 'oc status' to
    ↪ confirm that https://localhost:8443 is your actual
    ↪ cluster address
4 [17:41] <nazarewk> meta4knox: i'm getting this https://dpaste.
    ↪ de/7qPu
5 [17:41] <jbossbot> Title: dpaste
```

```
6 [17:43] <nazarewk> tried all options: curl -k https
    ↪ ://172.30.0.1:443/console/ and curl -k https
    ↪ ://10.0.2.15:8443/console/
7 [17:43] <nazarewk> and still getting exactly the same message
8 [17:44] <meta4knox> did you visit https://10.0.2.15:8443?
9 [17:44] <meta4knox> ok
10 [17:45] <meta4knox> Have you previously modified your hosts
    ↪ file such that it could be overriding this request?
11 [17:45] <nazarewk> nope i'm on fresh fedora atomic host vagrant
12 [17:46] <meta4knox> hmm
13 [17:46] <meta4knox> And this worked on other nodes without
    ↪ issue? (i.e. using the same configs?)
14 [17:47] <nazarewk> well i never managed to get it working
15 [17:47] <meta4knox> If so, then I'd just blow this one away and
    ↪  start fresh.
16 [17:47] <nazarewk> i just started researching openshift origin
    ↪ yesterday
17 [17:47] <meta4knox> OK
18 [17:47] <nazarewk> tried to run it
19 [17:47] <nazarewk> got though the try it out without issues on
    ↪ fedora atomic
20 [17:47] <nazarewk> but can't get to the console
21 [17:47] <meta4knox> Cloud hosting provider?
22 [17:48] <nazarewk> nope, i'm on my local machine and running it
    ↪  with vagrant
23 [17:48] <nazarewk> (i'm researching ways to get the Kubernetes
    ↪ onto bare metal without any extra infrastructure)
24 [17:49] <nazarewk> already went through Rancher and kubespray
    ↪ without issues
25 [17:49] <nazarewk> OpenShift looks the most promising but can't
    ↪  get it to work
26 [17:49] <meta4knox> Sounds like something's not exposed
    ↪ properly. I seem to remember (from long ago) that you
    ↪ need to expose your vm/container to the host in order to
    ↪  access it.
27 [17:49] <nazarewk> i'm trying from withing VM
28 [17:50] <meta4knox> Also, if you're trying to get Kube or
    ↪ OpenShift working without much hassle, I strongly
    ↪ recommend looking into Minishift
29 [17:50] <nazarewk> vagrant ssh -> sudo docker exec -it origin
    ↪ bash
```

```
[17:50] <nazarewk> i need it to run multi-host
[17:50] <nazarewk> (in later stages)
[17:50] <meta4knox> gotcha
[17:50] <meta4knox> Sorry I couldn't help
[17:51] <meta4knox> good luck
[17:51] <nazarewk> thanks for trying :)
```

7.4.4 Wnioski

Panel administracyjny klastra OpenShift Origin jest jedyną znaczącą przewagą nad Kubespray. Reszta zarządzania klastrem odbywa się również za pomocą repozytorium skryptów Ansibla (w tym dodawanie kolejnych węzłów klastra[10]).

Z powodu braku dostępu do ww. panelu próbę uruchomienia OpenShift Origin uznaję za nieudaną. # Q&A

7.5 Czy wszystko zawsze trzeba sciagac z netu - nie mozna z lokalnego serwera?

Można zestawić lokalny rejestr Dockera[11] jako proxy cachujące[12].

7.6 Jak zachować stan bezdyskowego RancherOS'a?

Jedynym narzędziem do "zachowywania stanu" bezdyskowego Ranchera i praktycznie wszystkich cloudowych systemów uruchamianych bez dysku jest cloud-init.

Normalnie konfigurowany jest przez własny cloud-init, aktualnie nie zawsze działa ze względu na bugi.

[10] https://docs.openshift.com/enterprise/3.0/admin_guide/manage_nodes.html#adding-nodes

[11] https://docs.docker.com/registry/

[12] https://docs.docker.com/registry/recipes/mirror/

7.7 Co musi zawierac cloud-config dla serwera a co dla agentow?

Sam RancherOS nie zarządza kontenerami, do tego potrzebne jest uruchomienie serwera Ranchera.

Bibliografia

Pizza, Mariagrazia, Vincenzo Scarlato, Vega Masignani, Marzia Monica Giuliani, Beatrice Arico, Maurizio Comanducci, Gary T Jennings, i in. 2000. „Identification of vaccine candidates against serogroup B meningococcus by whole-genome sequencing". *Science* 287 (5459). American Association for the Advancement of Science:1816–20.