

Politechnika Warszawska

W Y D Z I A Ł E L E K T R Y C Z N Y



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 wydajności środowiska Kubernetes na
maszynach bezdyskowych

Krzysztof Nazarewski

nr albumu 123456

promotor
mgr inż. Andrzej Toboła

WARSZAWA 2018

Implementacja i testy wydajności 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 wydajności 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

1	Test	1
2	Wstęp	2
3	Przegląd pojęć i systemów związanych z konteneryzacją	3
3.1	Konteneryzacja	3
3.1.1	Open Container Initiative	3
3.1.2	Docker	4
3.2	Kubernetes	4
3.2.1	Alternatywne rozwiązania zarządzania kontenerami . .	4
3.3	Zarządzanie Kubernetes z linii komend	5
3.3.1	kubeadm	5
3.3.2	Kubescape	5
3.3.3	OpenShift Ansible	5
3.3.4	Canonical distribution of Kubernetes	6
3.3.5	Eksperymentalne i deprecowane rozwiązania	6
3.4	Graficzne nakładki na Kubernetes	6
3.4.1	Kubernetes Dashboard	6
3.4.2	Rancher	7
3.4.3	OpenShift by Red Hat	7
3.4.4	DC/OS	7
4	Kubernetes	8
4.1	Administracja, a korzystanie z klastra	8
4.2	Architektura	8
4.2.1	Składowe kontrolujące klastr	8
4.2.2	Składowe workera	9
4.2.3	Komunikacja sieciowa	9
5	Systemy bezdyskowe	10
5.1	Proces uruchamiania maszyny bezdyskowej	10

6	Przegląd systemów operacyjnych	12
6.1	Konfigurator cloud-init	12
6.1.1	Dostępne implementacje	13
6.2	CoreOS	14
6.2.1	Konfiguracja	14
6.3	RancherOS	14
6.3.1	Konfiguracja	15
6.4	Project Atomic	15
6.5	Alpine Linux	16
6.6	ClearLinux	16
6.7	Wnioski	17
7	Konfiguracja klastra Kubernetes	18
7.1	Rancher 2.0	18
7.2	kubespray-cli	19
7.3	kubespray	21
7.3.1	Kubernetes Dashboard	21
7.3.2	Napotkane błędy	22
7.4	OpenShift Origin	22
8	Q&A	24
8.1	Czy wszystko zawsze trzeba ściągac z netu - nie mozna z lokalnego serwera?	24
8.2	Jak zachować stan bezdyskowego RancherOS'a?	24
8.3	Co musi zawierac cloud-config dla serwera a co dla agentow?	24
	Bibliografia	25

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 Initiative¹ 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.: - Docker² - Kubernetes CRI-O³ - 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>

FreeBSD⁴ - 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 porównanie `rkt` (kolejna generacja Dockera) z innymi rozwiązaniami⁶

3.2 Kubernetes

Kubernetes⁷ 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 Tool⁸

3.2.1 Alternatywne rozwiązania zarządzania kontenerami

Fleet

Fleet⁹ jest nakładką na `systemd`¹⁰ realizującą 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 Swarm¹¹ 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=akLa9L500NY>

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 Kubernetes¹²

Mesos

Apache Mesos¹³ jest najbardziej zaawansowanym i najlepiej skalującym się rozwiązaniem orkiestracji kontenerami. Jest również najbardziej skomplikowanym i trudnym w zarządzaniu rozwiązaniem.

- An Introduction to Mesosphere¹⁴

3.3 Zarządzanie Kubernetes z linii komend

3.3.1 kubeadm

kubeadm¹⁵ jest narzędziem pozwalającym na niskopoziomowe zarządzanie klastrem Kubernetes. Stąd trendem jest bazowanie na kubeadm przy tworzeniu narzędzi z wyższym poziomem abstrakcji.

- Install with kubadm¹⁶

3.3.2 Kubespray

- kubespray¹⁷
- zestaw skryptów Ansible konfigurujących klaster na jednym z wielu systemów operacyjnych
- dąży do zostania tzw. Operatorem¹⁸ korzystającym z kubeadm

3.3.3 OpenShift Ansible

Konfiguracja OpenShift Origin realizowana jest zestawem skryptów Ansible'owych rozwijanych jako projekt openshift-ansible¹⁹.

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 Service²⁰.

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

Przydatne materiały: - Juju Charm the Canonical distribution of Kubernetes²¹ - Install Kubernetes with conjure-up²² - Kubernetes the not so easy way²³ opisuje instalację lokalnego klastra.

3.3.5 Eksperymentalne i deprekowane rozwiązania

- Fedora via Ansible²⁴ deprekowane na rzecz kubescape
- Rancher Kubernetes Installer²⁵ jest eksperymentalnym rozwiązaniem wykorzystywanym w Rancher 2.0,

kubescape-cli

Jest to narzędzie ułatwiające korzystanie z kubescape. Według użytkowników oficjalnego Slacka kubescape²⁶ kubescape-cli jest deprekowane.

3.4 Graficzne nakładki na Kubernetes

3.4.1 Kubernetes Dashboard

Kubernetes Dashboard²⁷ 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/>

²⁴https://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/kubescape>

²⁷<https://github.com/kubernetes/dashboard>

3.4.2 Rancher

Rancher²⁸ 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 source'owy OpenShift Origin²⁹ konfigurowany przez OpenShift Ansible.

- OpenShift Origin vs Kubernetes³⁰
- The Differences Between Kubernetes and OpenShift³¹
- Demo konsoli³² (niestety po hebrajsku)

3.4.4 DC/OS

Datacenter Operating System³³ jest częścią Mesosphere³⁴ i Mesosa. Niedawno został rozszerzony o Kubernetes³⁵ jako alternatywnego (w stosunku do Marathon³⁶) systemu orkiestracji kontenerami.

²⁸<https://rancher.com/>

²⁹<https://github.com/openshift/origin>

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

³¹<https://medium.com/level-consulting/the-differences-between-kubernetes-and-openshift-aef7>

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/kubernetes-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 administracją 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 uruchamiać 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 klastery,

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 Host¹.

Potencjalnymi rozwiązaniami problemu przechowywania stanu maszyn bezdyskowych mogą być:

- przydziały NFS
- replikacja ZFS²
- 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 rozpoczne 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 dołączenia 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-init¹ 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 YAML² w trakcie lub tuż po inicjalizacji systemu operacyjnego.

Główną zaletą cloud-init jest tworzenie automatycznej i jednolitej 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 Dockera³ (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-init⁴

coreos-cloudinit

coreos-cloudinit⁵ jest częściową implementacją standardu w języku Go przez twórców CoreOS. Niestety rok temu przestał być rozwijany⁶ i wychodzi z użytku.

RancherOS + coreos-cloudinit

rancher cloud-init⁷ jest spadkobiercą⁸ coreos-cloudinit rozwijanym przez zespół RancherOS, na jego potrzeby.

²<http://yaml.org/>

³https://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-init⁹ 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

CoreOS¹⁰ 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 Config¹¹ transpilowany do Ignition¹². 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

RancherOS¹³ jest systemem operacyjnym, w którym tradycyjny system inicjalizacji został zastąpiony trzema poziomami Dockera¹⁴:

- **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-config¹⁵, 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 skrypcem 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 Atomic¹⁷ 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 Host¹⁸
- CentOS Atomic Host¹⁹
- Fedora Atomic Host²⁰

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 Linux²¹ jest minimalną dystrybucją Linuxa bazowaną na musl-libc i busybox.

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

Alpine Linux może być skonfigurowany przez Alpine Backup²² lub Alpine Configuration Framework²³.

6.6 ClearLinux

ClearLinux²⁴ 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 distro²⁵

²¹<https://alpinelinux.org/>

²²https://wiki.alpinelinux.org/wiki/Alpine_local_backup

²³http://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 kops¹, 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.

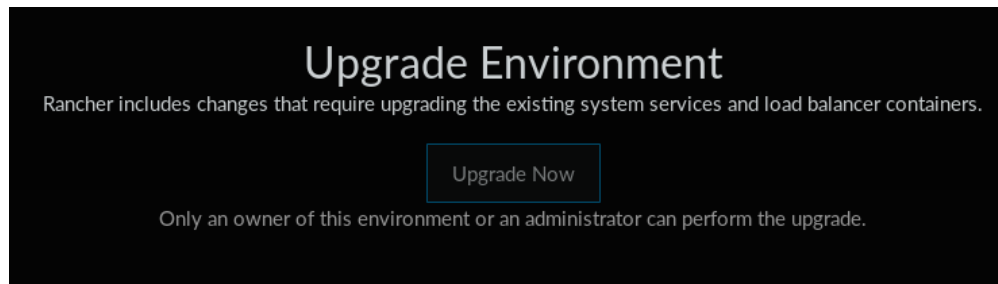
```
1 #rancher_version=latest
2 rancher_version=preview
3 docker run --rm --name rancher -d -p 8080:8080 rancher/server:$
  ↪ {rancher_version}
```

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 Environment².

7.2 kubespray-cli

Z powodu błędu³ logiki narzędzie nie radzi sobie z brakiem Python'a na domyślnej dystrybucji CoreOS'a, mimo że sam kubespray radzi sobie z nim

²<https://github.com/rancher/rancher/issues/10396>

³<https://github.com/kubespray/kubespray-cli/issues/120>

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

Wykracza 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`.

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

7.3 kubespray

Kod znajduje się w moim repozytorium `kubernetes-cluster`⁵.

```
1 #!/usr/bin/env bash
2 cd $(dirname "$(readlink -f "$0")")
3 source ./setup-cluster-vars
4 cd ${dir}
5
6 based on https://github.com/kubernetes-incubator/kubespray/
   ↳ blob/master/docs/getting-started.md#building-your-own-
   ↳ inventory
7 cp -r inventory -T ${inventory}
8 python3 contrib/inventory_builder/inventory.py ${IPS[@]}
9
10 cat > ${inventory}/group_vars/all.yml << EOF
11 bootstrap_os: coreos
12 #loadbalancer_apiserver:
13 # address: 0.0.0.0
14 # port: 8080
15 kube_basic_auth: true
16 kubeconfig_localhost: true
17 kubectll_localhost: true
18 download_run_once: True
19 EOF
20
21 ansible-playbook -i ${inventory}/inventory.cfg cluster.yml -b -
   ↳ v
22
23 echo Staring kubectll proxy
24 echo http://localhost:8001/api/v1/namespaces/kube-system/
   ↳ services/https:kubernetes-dashboard:/proxy/#!/login
25 ./kubectll proxy
```

7.3.1 Kubernetes Dashboard

Dostęp do Dashboardu najprościej można uzyskać:

1. nadanie wszystkich uprawnień roli `kubernetes-dashboard`⁶

⁵<https://github.com/nazarewk/kubernetes-cluster>

⁶<https://github.com/kubernetes/dashboard/wiki/Access-control#admin-privileges>

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 0=system:nodes to register]
  ↪ *****
2 Wednesday 17 January 2018  22:22:59 +0100 (0:00:00.626)
  ↪ 0:08:31.946 *****
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 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

```

OpenShifta próbowałem uruchomić na następujących systemach obiema metodami:

- Arch Linux Linux klna 4.14.13-1-ARCH #1 SMP PREEMPT Wed Jan
 ↪ 10 11:14:50 UTC 2018 x86_64 GNU/Linux
- CoreOS Linux localhost 4.14.11-coreos #1 SMP Fri Jan 5 11:00:14
 ↪ UTC 2018 x86_64 Intel(R) Core(TM) i5-2500K CPU @ 3.30GHz
 ↪ GenuineIntel GNU/Linux

We wszystkich konfiguracjach obie metody skutkują błędem⁷:

```

1 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"

```

⁷<https://github.com/openshift/origin/issues/14766>

Rozdział 8

Q&A

8.1 Czy wszystko zawsze trzeba ściągac z netu - nie mozna z lokalnego serwera?

Można zestawić lokalny rejestr Dockera¹ jako proxy cachujące².

8.2 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.

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

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

¹<https://docs.docker.com/registry/>

²<https://docs.docker.com/registry/recipes/mirror/>

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.