

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
3.3	Alternatywne rozwiązania zarządzania kontenerami	4
3.3.1	Fleet	4
3.3.2	Docker Swarm	5
3.3.3	Nomad	5
3.3.4	Mesos	5
4	Kubernetes	6
4.1	Administracja, a korzystanie z klastra	6
4.2	Konfiguracja klastra	6
4.3	Infrastruktura Kubernetes	7
4.3.1	Węzeł zarządzający	8
4.3.2	Węzeł roboczy	8
4.3.3	Wtyczka sieciowa	9
4.3.4	Komunikacja sieciowa	9
4.3.5	Zarządzanie dostępami	10
4.3.6	Obiekty Kubernetes API	10
4.3.7	Podstawowe rodzaje obiektów aplikacyjnych	11
4.4	Kubernetes Dashboard	14
4.5	Kubernetes Incubator	15
4.6	Administracja klastrem z linii komend	15
4.6.1	kubeadm	15
4.6.2	Kubespary	15

4.6.3	OpenShift Ansible	16
4.6.4	Canonical distribution of Kubernetes	16
4.6.5	Bootkube i Typhoon	16
4.6.6	Eksperymentalne i deprekowane rozwiązania	16
4.7	Administracja klastrem za pomocą graficznych narzędzi	17
4.7.1	Rancher	17
4.7.2	OpenShift by Red Hat	17
4.7.3	DC/OS	18
5	Systemy bezdyskowe	19
5.1	Proces uruchamiania maszyny bezdyskowej	19
6	Przegląd systemów operacyjnych	21
6.1	Konfigurator cloud-init	21
6.1.1	Dostępne implementacje	22
6.2	CoreOS	23
6.2.1	Konfiguracja	23
6.3	RancherOS	23
6.3.1	Konfiguracja	24
6.4	Project Atomic	24
6.5	Alpine Linux	25
6.6	ClearLinux	25
6.7	Wnioski	26
7	Praktyczne rozeznanie w narzędziach administracji klastrem	
	Kubernetes	27
7.1	kubespary-cli	27
7.1.1	Napotkane problemy	28
7.1.2	Wnioski	28
7.2	Rancher 2.0	29
7.2.1	Testowanie tech preview 1 (v2.0.0-alpha10)	29
7.2.2	Wnioski	31
7.3	OpenShift Origin	31
7.3.1	Korzystanie ze sterownika systemd zamiast domyślnego cgroupfs	32
7.3.2	Próba uruchomienia serwera na Arch Linux	32
7.3.3	Próba uruchomienia serwera na Fedora Atomic Host w VirtualBox'ie	34
7.3.4	Wnioski	37
7.4	kubespary	37
7.4.1	Kubernetes Dashboard	37

7.4.2	Napotkane błędy	38
7.4.3	Finalne skrypty konfiguracyjne	38
7.5	Wnioski	39
8	Uruchamianie Kubernetes w laboratorium 225	40
8.1	Przygotowanie węzłów CoreOS	40
8.2	Przeszkody związane z uruchamianiem skryptów na uczelnianym Ubuntu	41
8.2.1	Brak virtualenv'a	41
8.2.2	Klonowanie repozytorium bez logowania	42
8.2.3	Atrybut wykonywalności skryptów	42
8.2.4	Konfiguracja dostępu do maszyn bez hasła	42
8.2.5	Problemy z siecią	42
8.2.6	Limit 3 serwerów DNS	42
8.3	Pierwszy dzień - uruchamianie skryptów z maszyny s6	43
8.4	Kolejne próby uruchamiania klastra z maszyny s2	44
8.4.1	Generowanie inventory z HashiCorp Vault'em	44
8.4.2	Niepoprawne znajdowanie adresów IP w ansible	45
8.4.3	Dostęp do Kubernetes Dashboardu	45
8.4.4	Instalacja dodatkowych aplikacji z użyciem Kubespray	47
9	Q&A	48
9.1	Czy wszystko zawsze trzeba ściągac z netu - nie mozna z lokalnego serwera?	48
9.2	Jak zachować stan bezdyskowego RancherOS'a?	48
9.3	Co musi zawierac cloud-config dla serwera a co dla agentow?	48
	Bibliografia	49

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 (w skrócie k8s) jest obecnie najpopularniejszym narzędziem orkiestracji kontenerami, a przez to tematem przewodnim tego dokumentu.

Został stworzony przez Google na bazie ich wewnętrznego systemu Borg.

W porównaniu do innych narzędzi Kubernetes oferuje najlepszy kompromis między oferowanymi możliwościami, a kosztem zarządzania.

3.3 Alternatywne rozwiązania zarządzania kontenerami

- Choosing the Right Containerization and Cluster Management Tool⁸

3.3.1 Fleet

Fleet⁹ 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.

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/>

3.3.2 Docker Swarm

Docker Swarm¹¹ jest rozwiązaniem orkiestracji kontenerami od twórców samego Docker'a. Proste w obsłudze, ale nie oferuje tak dużych możliwości jak inne rozwiązania.

3.3.3 Nomad

Nomad¹² od HashiCorp jest narzędziem do zarządzania klastrem, które również oferuje zarządzanie kontenerami.

Przy jego tworzeniu twórcy kierują się filozofią Unix'a. W związku z tym Nomad jest prosty w obsłudze, wyspecjalizowany i rozszerzalny. Zwykle działa w tandemie z innymi produktami HashiCorp jak Consul i Vault.

Materiały:

- HashiCorp Nomad vs Other Software¹³

3.3.4 Mesos

Apache Mesos¹⁴ jest najbardziej zaawansowanym i najlepiej skalującym się rozwiązaniem orkiestracji kontenerami. Jest również najbardziej skomplikowanym i trudnym w zarządzaniu rozwiązaniem.

Materiały:

- An Introduction to Mesosphere¹⁵

¹¹<https://docs.docker.com/engine/swarm/>

¹²<https://www.nomadproject.io/intro/index.html>

¹³<https://www.nomadproject.io/intro/vs/index.html>

¹⁴<http://mesos.apache.org/>

¹⁵<https://www.digitalocean.com/community/tutorials/an-introduction-to-mesosphere>

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

Przez zwrot “administracja klastrem” (lub zarządzanie) rozumiem zbiór czynności i procesów polegających na przygotowaniu klastra do użytku i zarządzanie jego infrastrukturą. Na przykład: tworzenie klastra, dodawanie i usuwanie węzłów.

Przez zwrot “korzystanie z klastra” rozumiem uruchamianie aplikacji na działającym klastrze.

Ze względu na ograniczone zasoby czasu w tej pracy inżynierskiej skupiam się na kwestiach związanych z administracją klastrem.

4.2 Konfiguracja klastra

Ważną kwestią jest zrozumienie pojęcia stanu w klastrze Kubernetes. Jest to stan do którego klastr dąży, a nie w jakim się w danej chwili znajduje.

Zwykle stan docelowy i aktywny szybko się ze sobą zbiegają, ale nie jest to regułą. Najczęstszymi scenariuszami jest brak zasobów do uruchomienia aplikacji w klastrze lub zniknięcie węzła roboczego.

W pierwszym przypadku stan klastra może wskazywać na istnienie 5 instancji aplikacji, ale pamięci RAM wystarcza na uruchomienie tylko 3. Więc

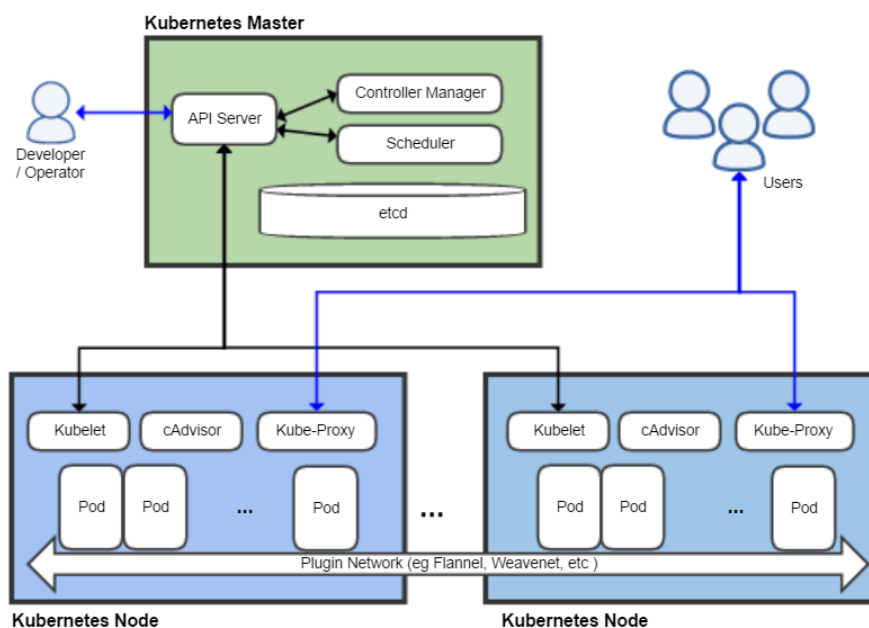
bez zmiany infrastruktury brakujące 2 instancje nigdy nie zostaną uruchomione. W momencie dołączenia kolejnego węzła klastra może się okazać, że posiadamy już oczekiwane zasoby i aplikacja zostanie uruchomiona w pełni.

W drugim przypadku założymy, że aplikacja jest uruchomiona w 9 kopiach na 4 węzłach, po 2 kopię na pierwszych trzech węzłach i 3 kopie na ostatnim. W momencie wyłączenia ostatniego węzła aplikacja będzie miała uruchomione tylko 6 z 9 docelowych instancji. Zanim moduł kontrolujący klaster zauważy braki aktywny stan 6 nie będzie się zgadzał z docelowym 9. W ciągu kilku do kilkudziesięciu sekund kontroler uruchomi brakujące 3 instancje i uzyskamy docelowy stan klastra: po 3 kopie aplikacji na 3 węzłach.

4.3 Infrastruktura Kubernetes

Infrastrukturę definiuję jako część odpowiadającą za funkcjonowanie klastra, a nie za aplikacje na nim działające. Z infrastrukturą wiąże pojęcie administracji klastrem.

Zdecydowałem się przybliżyć temat na podstawie jednego diagramu znalezionego na [wikimedia.org](https://commons.wikimedia.org/wiki/File:Kubernetes.png)¹:



Na ilustracji możemy wyróżnić 5 grup tematycznych:

¹<https://commons.wikimedia.org/wiki/File:Kubernetes.png>

1. **Developer/Operator**, czyli administrator lub programista korzystający z klastra,
2. **Users**, czyli użytkowników aplikacji działających w klastrze,
3. **Kubernetes Master**, czyli węzeł zarządzający (zwykle więcej niż jeden),
4. **Kubernetes Node**, czyli jeden z wielu węzłów roboczych, na których działają aplikacje,
5. **Plugin Network**, czyli wtyczka sieciowa realizująca lub konfiguruje połączenia pomiędzy kontenerami działającymi w ramach klastra,

4.3.1 Węzeł zarządzający

Stan Kubernetes jest przechowywany w `etcd`². Nazwa wzięła się od Unixowego folderu `/etc` przechowującego konfigurację systemu operacyjnego i litery `d` oznaczającej system rozproszony (ang. distributed system). Jest to baza danych przechowująca jedynie klucze i wartości (ang. key-value store). Koncepcyjnie jest prosta, żeby umożliwić skupienie się na jej wydajności, stabilności i skalowaniu.

Jedynym sposobem zmiany stanu `etcd` (zakładając, że nie jest wykorzystywane do innych celów) jest komunikacja z `kube-apiserver`³. Zarówno zewnątrzni użytkownicy jak i wewnętrzne procesy klastra korzystają z interfejsu aplikacyjnego REST (ang. REST API) klastra w celu uzyskania informacji i zmiany jego stanu.

Głównym modulem zarządzającym, który dba o doprowadzenia klastra do oczekiwanego stanu jest `kube-controller-manager`⁴. Uruchamia on pętle kontrolujące klaster, na której bazuje wiele procesów kontrolnych jak na przykład kontroler replikacji i kontroler kont serwisowych.

Modulem zarządzającym zasobami klastra jest `kube-scheduler`⁵. Decyduje on na których węzłach uruchamiać aplikacje, żeby zaspokoić popyt na zasoby jednocześnie nie przeciążając pojedynczych węzłów klastra.

4.3.2 Węzeł roboczy

Podstawowym procesem działającym na węzłach roboczych jest `kubelet` → ⁶. Monitoruje i kontroluje kontenery działające w ramach jednego węzła.

²<https://coreos.com/etcd/>

³<https://kubernetes.io/docs/reference/generated/kube-apiserver/>

⁴<https://kubernetes.io/docs/reference/generated/kube-controller-manager/>

⁵<https://kubernetes.io/docs/reference/generated/kube-scheduler/>

⁶<https://kubernetes.io/docs/reference/generated/kubelet/>

Na przykład wiedząc, że na węźle mają działać 2 instancje aplikacji dba o to, żeby restartować instancje działające nieprawidłowo i/lub dodawać nowe.

Drugim najważniejszym procesem węzła roboczego jest `kube-proxy` odpowiadające za przekierowywanie ruchu sieciowego do odpowiednich kontenerów w ramach klastra.

Ostatnim opcjonalnym elementem węzła roboczego jest `cAdvisor`⁷ (Container Advisor), który monitoruje zużycie zasobów i wydajność kontenerów w ramach jednego klastra.

4.3.3 Wtyczka sieciowa

Podstawowym założeniem Kubernetes jest posiadanie własnego adresu IP przez każdą aplikację działającą w klastrze, ale nie narzuca żadnego rozwiązania je realizującego.

Administrator (lub skrypt konfiguracyjny) klastra musi zadbać o to, żeby skonfigurować wtyczkę sieciową realizującą to założenie.

Najprostszym koncepcyjnie rozwiązaniem jest stworzenie na każdym węźle wpisów `iptables` przekierowujących adresy IP na wszystkie inne węzły.

Jednymi z najpopularniejszymi rozwiązaniami są: Flannel⁸ i Project Calico⁹.

4.3.4 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

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:

⁷<https://github.com/google/cadvisor>

⁸<https://github.com/coreos/flannel#flannel>

⁹<https://www.projectcalico.org/>

- 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,

4.3.5 Zarządzanie dostępami

Namespace

Namespace jest logicznie odseparowaną częścią klastra Kubernetes. Pozwala na współdzielenie jednego klastra przez wielu niezauważanych użytkowników. Standardowym zastosowaniem jest wydzielanie środowisk produkcyjnych, QA i deweloperskich. ## Architektura

Architekturę klastra definiuję jako część aplikacyjną, czyli wszystkie funkcjonalności dostępne po przeprowadzeniu prawidłowej konfiguracji klastra i oddaniu węzłów do użytku. Z architekturą wiąże pojęcia korzystania z klastra, stanu i obiektów Kubernetes.

4.3.6 Obiekty Kubernetes API

Obiekty Kubernetes¹⁰ są trwale przechowywane w `etcd` i definiują, jak wcześniej wyjaśniłem, pożądaną stan klastra. Szczegółowy opis konwencji API obiektów możemy znaleźć w odnośniku¹¹.

Jako użytkownicy klastra operujemy na ich reprezentacji w formacie YAML, a rzadziej JSON, na przykład:

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: my-pod
5   namespace: my-namespace
6   uid: 343fc305-c854-44d0-9085-baed8965e0a9
7   labels:
8     resources: high
9   annotations:
10    app-type: qwe
11 spec:
```

¹⁰<https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>

¹¹<https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md>

```

12 containers:
13   - image: ubuntu:trusty
14     command: ["echo"]
15     args: ["Hello World"]
16   ...
17 status:
18   podIP: 127.12.13.14
19   ...

```

W każdym obiekcie możemy wyróżnić trzy obowiązkowe i dwa opcjonalne pola:

- `apiVersion`: obowiązkowa wersja API Kubernetes,
- `kind`: obowiązkowy typ obiektu zdefiniowanego w specyfikacji `apiVersion`
↪ ,
- `metadata`
 - `namespace`: opcjonalna (domyślna `default`) przestrzeń nazw do której należy obiekt,
 - `name`: obowiązkowa i unikalna w ramach przestrzeni nazw nazwa obiektu,
 - `uid`: unikalny identyfikator obiektu tylko do odczytu,
 - `labels`: opcjonalny zbiór kluczy i wartości ułatwiających identyfikację i grupowanie obiektów,
 - `annotations`: opcjonalny zbiór kluczy i wartości wykorzystywanych przez zewnętrzne lub własne narzędzia,
- `spec`: z definicji opcjonalna, ale zwykle wymagana specyfikacja obiektu wpływająca na jego funkcjonowanie,
- `status`: opcjonalny aktualny stan obiektu tylko do odczytu,

4.3.7 Podstawowe rodzaje obiektów aplikacyjnych

Ważną kwestią jest rozróżnienie obiektów imperatywnych i deklaratywnych. Obiekty imperatywne reprezentują wykonanie akcji, a deklaratywne określają stan w jakim klaster powinien się znaleźć.

Pod

Pod¹² jest najmniejszą jednostką aplikacyjną w Kubernetes. Reprezentuje nierozłącznie powiazaną (np. współdzielonymi zasobami) grupę jednego lub

¹²<https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/>

więcej kontenerów.

Pod w odróżnieniu od innych obiektów reprezentuje aktualnie działającą aplikację. Są bezustannie uruchamiane i wyłączane przez kontrolery. Trwałość danych można uzyskać jedynie przydzielając im zasoby dyskowe.

Pody nie powinny być zarządzane bezpośrednio, jedynie przez kontrolery. Najczęściej konfigurowane są przez `PodTemplateSpec`, czyli szablony ich specyfikacji.

Kontenery wewnątrz Poda współdzielą adres IP i mogą komunikować się przez `localhost` i standardowe metody komunikacji międzyprocesowej.

Dodatkowo kontenery wewnątrz Podów obsługują 2 rodzaje próbników¹³: `livenessProbe` i `readinessProbe`. Pierwszy określa, czy kontener działa, jeżeli nie to powinien być zrestartowany. Drugi określa czy kontener jest gotowy do obsługi zapytań, kontener jest wyrejestrowywany z `Service` na czas nieprzechodzenia `readinessProbe`.

ReplicaSet

`ReplicaSet`¹⁴ jest następcą `ReplicaControllera`, czyli imperatywnym kontrolerem dbającym o działanie określonej liczby Podów w klastrze.

Jest to bardzo prosty kontroler i nie powinien być używany bezpośrednio.

Deployment

`Deployment`¹⁵ pozwala na deklaratywne aktualizacje Podów i `ReplicaSet`ów. Korzystanie z ww. bezpośrednio nie jest zalecane.

Zmiany `Deployment`ów wprowadzane są przez tak zwane `rollouts`. Każdy ma swój status i może zostać wstrzymany lub przerwany. `Rollouts` mogą zostać aktywowane automatycznie przez zmianę specyfikacji Poda przez `.spec` ➡ `.template`.

Rewizje `Deploymentu` są zmieniane tylko w momencie `rolloutu`. Operacja operacja skalowania nie uruchamia `rolloutu`, a więc nie zmienia rewizji.

Podstawowe przypadki użycia `Deployment` to:

- uruchamianie `ReplicaSet`ów w tle przez `.spec.replicas`,
- deklarowanie nowego stanu Podów zmieniając `.spec.template`,
- cofanie zmian do poprzednich rewizji `Deploymentu` (poprzednie wersje Podów) komendą `kubectl rollout undo`,

¹³<https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes>

¹⁴<https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>

¹⁵<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

- skalowanie Deploymentu w celu obsługi większego obciążenia przykładową komendą `kubectl autoscale deployment nginx-deployment`
 ↪ `--min=10 --max=15 --cpu-percent=80`,
- wstrzymywanie Deployment w celu wprowadzenia poprawek komendą `kubectl rollout pause deployment/nginx-deployment`,
- czyszczenie historii ReplicaSetów przez ograniczanie liczby wpisów w `.spec.revisionHistoryLimit`,

Przykładowy Deployment tworzący 3 repliki serwera `nginx`:

```

1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: nginx-deployment
5   labels:
6     app: nginx
7 spec:
8   replicas: 3
9   selector:
10    matchLabels:
11      app: nginx
12   template:
13     metadata:
14       labels:
15         app: nginx
16     spec:
17       containers:
18         - name: nginx
19           image: nginx:1.7.9
20           ports:
21             - containerPort: 80

```

Pole `.spec.selector` definiuje w jaki sposób Deployment ma znaleźć Pod
 ↪ y, którymi ma zarządzać. Selektor powinien zgadzać się ze zdefiniowanym szablonem.

StatefulSet

`StatefulSet`¹⁶ jest kontrolerem podobnym do Deploymentu, ale umożliwiającym zachowanie stanu Podów.

¹⁶<https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>

W przeciwieństwie do Deploymentu StatefulSet nadaje każdemu uruchomionemu Podowi stały unikalny identyfikator, który zostają zachowane mimo restartów i przenoszenia Podów. Identyfikatory można zastosować między innymi do:

- trwałych i unikalnych identyfikatorów wewnątrz sieci,
- trwałych zasobów dyskowych,
- sekwencyjne uruchamianie i skalowanie aplikacji,
- sekwencyjne zakańczanie i usuwanie aplikacji,
- sekwencyjne, zautomatyzowane aktualizacje aplikacji,

DaemonSet

DaemonSet¹⁷ jest kontrolerem upewniającym się, że przynajmniej jeden Pod działa na każdym lub wybranych węzłach klastra.

Do jego typowych zastosowań należy implementacja narzędzi wymagających agenta na każdym z węzłów:

- rozproszone systemy dyskowe, np. glusterd, ceph,
- zbieracze logów, np. fluentd, logstash,
- monitorowanie węzłów, np. Prometheus Node Exporter, collectd,

Job i CronJob

Job¹⁸ pozwala na jednorazowe uruchomienie Podów, które wykonują akcję i się kończą. Istnieją 3 tryby wykonania: niezrównoleglony, równoległy i równoległy z zewnętrzną kolejką zadań.

Domyślnie przy niepowodzeniu uruchamianie są kolejne Pody aż zostanie uzyskana odpowiednia liczba sukcesów.

CronJob¹⁹ pozwala na tworzenie Jobów jednorazowo o określonym czasie lub je powtarzać zgodnie ze specyfikacją cron²⁰.

4.4 Kubernetes Dashboard

Kubernetes Dashboard²¹ jest wbudowanym interfejsem graficznym klastra Kubernetes. Umożliwia monitorowanie i zarządzanie klastrem w ramach

¹⁷<https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>

¹⁸<https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/>

¹⁹<https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/>

²⁰<https://en.wikipedia.org/wiki/Cron>

²¹<https://github.com/kubernetes/dashboard>

funkcjonalności samego Kubernetes.

4.5 Kubernetes Incubator

Kubernetes Incubator²² gromadzi projekty rozszerzające Kubernetes, ale nie będące częścią oficjalnej dystrybucji. Został stworzony, aby opanować bałagan w głównym repozytorium oraz ujednolicić proces tworzenia rozszerzeń.

Aby dołączyć do inkubatora projekt musi spełnić szereg wymagań oraz nie może spędzić w inkubatorze więcej niż 18 miesięcy. Dostępne opcje opuszczenia inkubatora to:

- awansować do rangi oficjalnego projektu Kubernetes,
- połączyć się z istniejącym oficjalnym projektem,
- po 12 miesiącach przejść w stan spoczynku, a po kolejnych 6 miesiącach zostać przeniesiony do `kubernetes-incubator-retired`

4.6 Administracja klastrem z linii komend

4.6.1 kubeadm

kubeadm²³ jest narzędziem pozwalającym na niskopoziomowe zarządzanie klastrem Kubernetes. Stąd trendem jest bazowanie na kubeadm przy tworzeniu narzędzi z wyższym poziomem abstrakcji.

- Install with kubadm²⁴

4.6.2 Kubespray

kubespray²⁵ jest zbiorem skryptów Ansibla konfigurujących klastry na różnych systemach operacyjnych i w różnych konfiguracjach. W tym jest w stanie skonfigurować klastry bare metal bez żadnych zewnętrznych zależności.

Projekt na dzień dzisiejszy znajduje się w inkubatorze i jest aktywnie rozwijany.

²²<https://github.com/kubernetes/community/blob/master/incubator.md>

²³<https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm/>

²⁴<https://kubernetes.io/docs/setup/independent/install-kubeadm/>

²⁵<https://github.com/kubernetes-incubator/kubespray>

4.6.3 OpenShift Ansible

Konfiguracja OpenShift Origin realizowana jest zestawem skryptów Ansible'owych rozwijanych jako projekt openshift-ansible²⁶.

4.6.4 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 Service²⁷.

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

Przydatne materiały: - Juju Charm the Canonical distribution of Kubernetes²⁸ - Install Kubernetes with conjure-up²⁹ - Kubernetes the not so easy way³⁰ opisuje instalację lokalnego klastra.

4.6.5 Bootkube i Typhoon

Bootkube³¹ jest narzędziem napisanym w Go pozwalającym na konfigurację Kubernetes na własnych maszynach.

Proponowanym trybu bare-metal³² jest wykorzystanie Terraform i Typhoon³³ do realizacji automatycznej konfiguracji klastra w trakcie uruchamiania węzłów CoreOS.

Domyślnie ww. narzędzia konfigurują instalację CoreOS na dysku, a następnie restartują

Niestety w trakcie pisania pracy przegapiłem wzmiankę o możliwości TODO: rozwinąć

4.6.6 Eksperymentalne i deprecowane rozwiązania

- Fedora via Ansible³⁴ deprecowane na rzecz kubenspray

²⁶<https://github.com/openshift/openshift-ansible>

²⁷

²⁸<https://jujucharms.com/canonical-kubernetes/>

²⁹<https://tutorials.ubuntu.com/tutorial/install-kubernetes-with-conjure-up>

³⁰<https://insights.ubuntu.com/2017/10/12/kubernetes-the-not-so-easy-way/>

³¹<https://github.com/kubernetes-incubator/bootkube>

³²<https://github.com/coreos/matchbox/tree/master/examples/terraform/>

bootkube-install

³³<https://github.com/poseidon/typhoon>

³⁴https://kubernetes.io/docs/getting-started-guides/fedora/fedora_ansible_config/

- Rancher Kubernetes Installer³⁵ jest eksperymentalnym rozwiązaniem wykorzystywanym w Rancher 2.0,

kubespray-cli

Jest to narzędzie ułatwiające korzystanie z kubespray. Według użytkowników oficjalnego Slacka kubespray³⁶ kubespray-cli jest deprecowane.

4.7 Administracja klastrem za pomocą graficznych narzędzi

4.7.1 Rancher

Rancher³⁷ 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.

4.7.2 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 Origin³⁸ konfigurowany przez OpenShift Ansible.

- OpenShift Origin vs Kubernetes³⁹
- The Differences Between Kubernetes and OpenShift⁴⁰
- Demo konsoli⁴¹ (niestety po hebrajsku)

³⁵<http://rancher.com/announcing-rke-lightweight-kubernetes-installer/>

³⁶<https://kubernetes.slack.com/messages/kubespray>

³⁷<https://rancher.com/>

³⁸<https://github.com/openshift/origin>

³⁹https://www.reddit.com/r/devops/comments/59ql4r/openshift_origin_vs_kubernetes/

⁴⁰<https://medium.com/levvel-consulting/the-differences-between-kubernetes-and-openshift-aef7>

⁴¹<https://youtu.be/-mFovK19aB4?t=6m54s>

4.7.3 DC/OS

Datacenter Operating System⁴² jest częścią Mesosphere⁴³ i Mesosa. Niedawno został rozszerzony o Kubernetes⁴⁴ jako alternatywnego (w stosunku do Marathon⁴⁵) systemu orkiestracji kontenerami.

⁴²<https://dcos.io/>

⁴³<https://mesosphere.com/>

⁴⁴<https://mesosphere.com/blog/kubernetes-dcos/>

⁴⁵<https://mesosphere.github.io/marathon/>

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 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)

Pełną lokalną konfigurację bazowaną na Dockerze przechowuję w moim repozytorium ipxe-boot³.

³<https://github.com/nazarewk/ipxe-boot>

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 dołączania do klastra Kubernetes,
- wsparcie dla środowiska bezdyskowego,
- możliwość bootu PXE,

Podstawowe wyznaczniki:

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

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 jednnorodnej 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 i posiada najnowsze wersje Dockera i OverlayFS.

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,

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/>

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

Czysta instalacja zajmuje około 700 MB pamięci RAM. Niestety nie jest często aktualizowany i posiada stare wersje zarówno Dockera (17.06 z przed pół roku) jak i `overlay` (zamiast `overlay2`).

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 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

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/>

- Fedora Atomic Host²⁰

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 błąd w procesie inicjalizacji systemu aktualnie nie ma możliwości jego uruchomienia 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:

- 6 key points about Intel's hot new Linux distro²⁵

²⁰<https://getfedora.org/atomic/download/>

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

Poza aktualnością oprogramowania systemy przeznaczone do działania w chmurze są pod kątem naszego zastosowania efektywnie identyczne.

Najczęściej aktualizowanym z powyższych systemów jest CoreOS, więc na nim skupię się w dalszej części pracy.

Rozdział 7

Praktyczne rozeznanie w narzędziach administracji klastrem 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 kubespray-cli

Z powodu błędu² 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-os³ z kubespray zanim przystąpimy do właściwego deploy’u. Skrypt uruchamiający:

```
1 #!/usr/bin/env bash
2 set -e
3
4 #pip2 install ansible kubespray
5 get_coreos_nodes() {
6     for node in $@
```

¹<https://github.com/kubernetes/kops>

²<https://github.com/kubespray/kubespray-cli/issues/120>

³<https://github.com/kubernetes-incubator/kubespray/blob/master/roles/bootstrap-os/tasks/main.yml>

```

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

```

7.1.1 Napotkane problemy

Narzędzie kończy się błędem 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`.

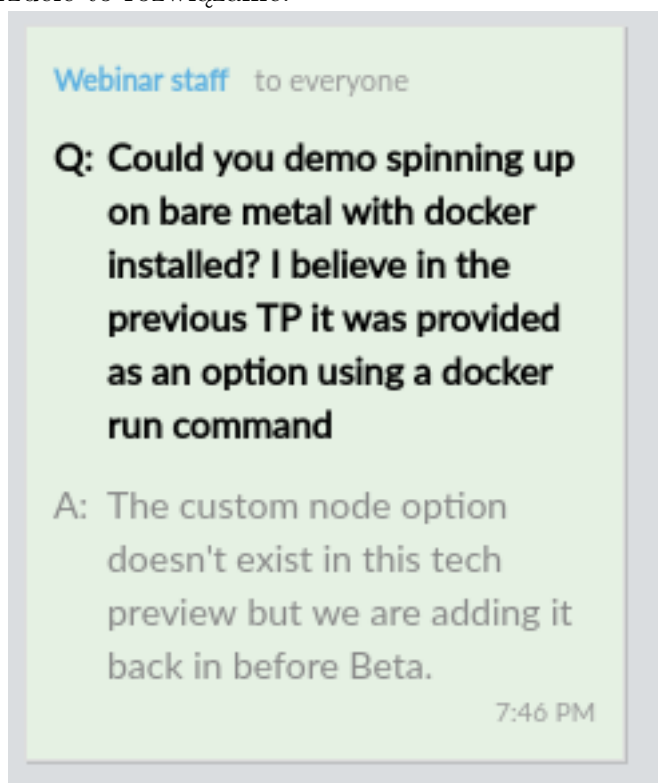
7.1.2 Wnioski

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

7.2 Rancher 2.0

Jest to 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.

W trakcie pisania pracy (24 stycznia 2018) pojawiło się drugie Tech Preview. W stosunku do pierwszego Tech Preview aplikacja została mocno przebudowana i nie wspiera jeszcze konfiguracji bare-metal, więc jestem zmuszony odrzucić to rozwiązanie.



7.2.1 Testowanie tech preview 1 (v2.0.0-alpha10)

```
1 #rancher_version=latest
2 #rancher_version=preview
3 rancher_version=v2.0.0-alpha10
4 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
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
  ↪ /...
```

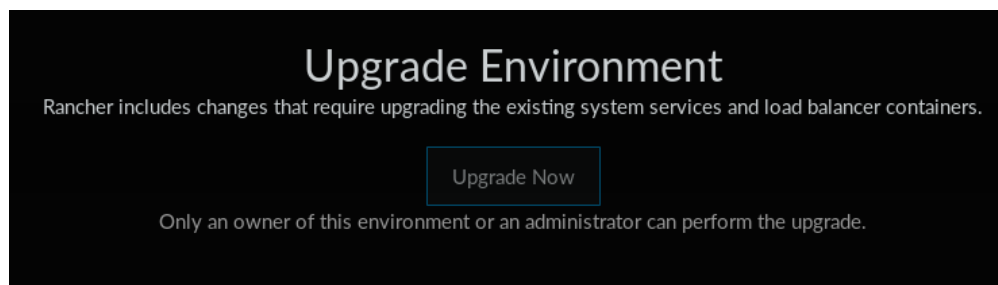
Od wersji 2.0 umożliwia połączenie się z istniejącym klastrem:

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

Napotkane błędy

W wersji v2.0.0-alpha10 losowo pojawia się błąd Upgrade Environment⁴.

⁴<https://github.com/rancher/rancher/issues/10396>



Rysunek 7.1: Błąd pt. Upgrade Environment

7.2.2 Wnioski

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

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

7.3 OpenShift Origin

Według dokumentacji⁵ 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

⁵https://docs.openshift.org/latest/getting_started/administrators.html

7.3.1 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`:

```
1 $ docker info | grep -i cgroup
2 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 Dockera⁶, czyli poniższym błędem:

```
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"
```

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:

```
1 $ cp /usr/lib/systemd/system/docker.service /etc/systemd/system
  ↳ /docker.service
2 $ vim /etc/systemd/system/docker.service
3 $ diff /usr/lib/systemd/system/docker.service /etc/systemd/
  ↳ system/docker.service
4 13c13
5 < ExecStart=/usr/bin/dockerd -H fd://
6 ---
7 > ExecStart=/usr/bin/dockerd -H fd:// --exec-opt native.
  ↳ cgroupdriver=systemd
```

7.3.2 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 Out⁷:

⁶<https://github.com/openshift/origin/issues/14766>

⁷https://docs.openshift.org/latest/getting_started/administrators.html#try-it-out

0. Uruchomienie shella na serwerze:

```
1 $ docker exec -it origin bash
```

1. Logowanie jako testowy użytkownik:

```
1 $ oc login
2 Username: test
3 Password: test
```

2. Stworzenie nowego projektu:

```
1 $ oc new-project test
```

3. Pobranie aplikacji z Docker Hub:

```
1 $ oc tag --source=docker openshift/deployment-example:v1
   ↪ deployment-example:latest
```

4. Wystartowanie aplikacji:

```
1 $ oc new-app deployment-example:latest
```

5. Oczekanie aż aplikacja się uruchomi:

```
1 $ watch -n 5 oc status
2 In project test on server https://192.168.0.87:8443
3
4 svc/deployment-example - 172.30.52.184:8080
5   dc/deployment-example deploys istag/deployment-example:latest
6   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.

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

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

```
1 # -*- mode: ruby -*-
2 # vi: set ft=ruby :
3
4 Vagrant.configure("2") do |config|
5   config.vm.box = "fedora/27-atomic-host"
6   config.vm.box_check_update = false
7   config.vm.network "forwarded_port", guest: 8443, host: 18443,
8     ↪ host_ip: "127.0.0.1"
9   config.vm.network "forwarded_port", guest: 8080, host: 18080,
10     ↪ host_ip: "127.0.0.1"
11   config.vm.provider "virtualbox" do |vb|
12     vb.gui = false
13     vb.memory = "8192"
14   end
15   config.vm.provision "shell", inline: <<-SHELL
16   SHELL
17 end
```

```
1 $ vagrant up
2 $ vagrant ssh
3 $ sudo docker run -d --name "origin" \
4   --privileged --pid=host --net=host \
5   -v /:/rootfs:ro \
6   -v /var/run:/var/run:rw \
7   -v /sys:/sys \
8   -v /sys/fs/cgroup:/sys/fs/cgroup:rw \
9   -v /var/lib/docker:/var/lib/docker:rw \
10  -v /var/lib/origin/openshift.local.volumes:/var/lib/origin/
11     ↪ openshift.local.volumes:rslave \
12  openshift/origin start
```

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

5. Oczekanie aż aplikacja się uruchomi i weryfikacja działania:

```
1 $ watch -n 5 oc status
```

```

2 In project test on server https://10.0.2.15:8443
3
4 svc/deployment-example - 172.30.221.105:8080
5   dc/deployment-example deploys istag/deployment-example:latest
6     deployment #1 deployed 3 seconds ago - 1 pod
7 $ 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 Open-Shift:

```

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 kubespary
 ↪ 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
30 [17:50] <nazarewk> i need it to run multi-host
31 [17:50] <nazarewk> (in later stages)
32 [17:50] <meta4knox> gotcha
33 [17:50] <meta4knox> Sorry I couldn't help
34 [17:51] <meta4knox> good luck
35 [17:51] <nazarewk> thanks for trying :)

```

7.3.4 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 klastra⁸).

Z powodu braku dostępu do ww. panelu próbę uruchomienia OpenShift Origin uznaję za nieudaną.

7.4 kubespray

Cały kod znajduje się w moim repozytorium `kubernetes-cluster`⁹.

7.4.1 Kubernetes Dashboard

Dostęp do Dashboardu najprościej można uzyskać:

1. nadanie wszystkich uprawnień roli `kubernetes-dashboard`¹⁰
2. Wejście na `http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login`
3. Kliknięcie skip

⁸https://docs.openshift.com/enterprise/3.0/admin_guide/manage_nodes.html#adding-nodes

⁹<https://github.com/nazarewk/kubernetes-cluster>

¹⁰<https://github.com/kubernetes/dashboard/wiki/Access-control#admin-privileges>

```

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>

7.4.2 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"}

```

7.4.3 Finalne skrypty konfiguracyjne

```

1 #!/bin/sh
2 set -a
3 base_dir=$(readlink -f $(dirname "$(readlink -f "$0")")/..)
4 bin=${base_dir}/bin
5 kubespray_dir=${base_dir}/kubespray
6 inventory=my_inventory

```

```

7 DASHBOARD_URL=http://localhost:8001/api/v1/namespaces/kube-
  ↪ system/services/https:kubernetes-dashboard:/proxy/#!/
  ↪ login
8 PATH="${base_dir}/bin:${PATH}"
9 CONFIG_FILE=${inventory}/inventory.cfg
10 KUBECONFIG=${kubespray_dir}/artifacts/admin.conf
11 ANSIBLE_CONFIG="${base_dir}/ansible.cfg"
12 set +a

1 #!/bin/sh
2 set -e
3 cd $(dirname "$(readlink -f "$0")")
4 bin=$(pwd)
5 if [ -z "$@" ]; then
6   . activate $@
7 else
8   . setup-cluster-configure $@
9 fi
10
11 (
12   cd ${kubespray_dir};
13   ansible-playbook -i ${inventory}/inventory.cfg cluster.yml -b
     ↪ -v
14 )
15
16 echo Staring kubectl proxy
17 echo http://localhost:8001/api/v1/namespaces/kube-system/
     ↪ services/https:kubernetes-dashboard:/proxy/#!/login
18 echo user: kube
19 echo password: $(cat ${base_dir}/credentials/kube_user)
20 ${bin}/kubectl proxy

```

7.5 Wnioski

Na moment pisania tej pracy Kubespray jest jedynym aktywnie rozwijanym i działającym rozwiązaniem uruchamiania klastra Kubernetes.

Rozdział 8

Uruchamianie Kubernetes w laboratorium 225

8.1 Przygotowanie węzłów CoreOS

Musiałem przygotować `coreos.ipxe` i `coreos.ign` do bootu i bezhasłowego dostępu.

Po pierwsze stworzyłem Container Linux Config (plik `coreos.yml`) zawierający:

1. Tworzenie użytkownika `nazarewk`
2. Nadanie mu praw do `sudo` i `dockera` (grupy `sudo` i `docker`)
3. Dodanie dwóch kluczy: wewnętrznego uczelnianego i mojego używanego na codzień w celu zdalnego dostępu.

```
1 passwd:
2   users:
3     - name: nazarewk
4       groups: [sudo, docker]
5       ssh_authorized_keys:
6         - ssh-rsa ... nazarewk
7         - ssh-rsa ... nazarewk@ldap.iem.pw.edu.pl
```

Następnie skompilowałem go do formatu Ignition narzędziem `ct`, skryptem `prepare-coreos`:

```
1 #!/bin/sh
2 cd "$(dirname "$(readlink -f "$0")")/.."
3 ct_version=${1:-0.6.1}
4 base_dir=$(pwd)
```



```

5 bin=${base_dir}/bin
6 boot="${base_dir}/zetis/WWW/boot"
7
8 wget -nc "https://github.com/coreos/container-linux-config-
  ↳ transpiler/releases/download/v${ct_version}/ct-v${
  ↳ ct_version}-x86_64-unknown-linux-gnu" -O ${bin}/ct
9 chmod +x ${bin}/ct
10 ${bin}/ct -pretty -in-file "${boot}/coreos.yml" -out-file "${
  ↳ boot}/coreos.ign"

```

Przygotowałem skrypt IPXE do uruchamiania CoreOS `coreos.ipxe`:

```

1 #!ipxe
2 set ftp http://ftp/pub/
3
4 set base-url ${ftp}/Linux/CoreOS/alpha
5 set ignition ${ftp}/Linux/CoreOS/zetis/kubernetes/boot/coreos.
  ↳ ign
6
7 set opts ${opts} coreos.autologin
8 set opts ${opts} coreos.first_boot=1 coreos.config.url=${
  ↳ ignition}
9 set opts ${opts} systemd.journald.max_level_console=debug
10 kernel ${base-url}/coreos_production_pxe.vmlinuz ${opts}
11 initrd ${base-url}/coreos_production_pxe_image.cpio.gz
12
13 boot

```

Umieściłem skrypt w `/home/stud/nazarewk/WWW/boot` i wskazałem go maszynom, które będą węzłami:

```

1 sudo lab 's4 s5 s6 s8 s9' boot http://vol/~nazarewk/boot/coreos
  ↳ .ipxe

```

8.2 Przeszkody związane z uruchamianiem skryptów na uczelnianym Ubuntu

8.2.1 Brak `virtualenv`'a

Moje skrypty nie przewidywały braku `virtualenv`, więc musiałem ręcznie zainstalować go komendą `apt-get install virtualenv`. Dodałem ten krok do skryptu `setup-packages`.

8.2.2 Klonowanie repozytorium bez logowania

W celu umożliwienia anonimowego klonowania repozytorium z Githuba musiałem zmienić protokół z `git` na `https`:

```
1 git clone https://github.com/nazarewk/kubernetes-cluster.git
```

problem pojawił się również dla submodułów gita (`.gitmodules`).

8.2.3 Atrybut wykonywalności skryptów

W konfiguracji uczelnianej git nie ustawia domyślnie atrybutu wykonalności dla plików wykonywalnych i zdejmuje go przy aktualizacji pliku. Problem rozwiązałem dodaniem komendy `chmod +x bin/*` do skryptu `pull`.

8.2.4 Konfiguracja dostępu do maszyn bez hasła

Ponad konfigurację CoreOS musiałem wypełnić konfigurację SSH do bezhasłowego dostępu, w pliku `~/.ssh/config` umieściłem:

```
1 User nazarewk
2 IdentityFile ~/.ssh/id_rsa
3 IdentitiesOnly yes
4
5 Host s? 10.146.255.*
6   StrictHostKeyChecking no
7   UserKnownHostsFile /dev/null
8
9 Host s3 s4 s5 s6 s7 s8 s9
10  User core
```

8.2.5 Problemy z siecią

W trakcie pierwszego uruchamiania występowały problemy z siecią uczelnianą, więc rozszerzyłem plik `ansible.cfg` o ponowne próby wywoływania komend dodając wpis `retires=5` do sekcji `[ssh_connection]`.

8.2.6 Limit 3 serwerów DNS

Napotkałem limit 3 serwerów DNS¹:

¹<https://github.com/kubernetes-incubator/kubespray/blob/master/docs/dns-stack.md#limitations>

```

1 TASK [docker : check system nameservers]
  ↳ *****
2 Friday 26 January 2018  14:47:09 +0100 (0:00:01.429)
  ↳ 0:04:26.879 *****
3 ok: [node3] => {"changed": false, "cmd": "grep \"^nameserver\"
  ↳ /etc/resolv.conf | sed 's/^nameserver\\s*/'/", "delta":
  ↳ "0:00:00.004652", "end": "2018-01-26 13:47:11.659298", "
  ↳ rc": 0, "start": "2018-01-26 13:47:11.654646", "stderr":
  ↳ "", "stderr_lines": [], "stdout": "172.29.146.3\n1
4 72.29.146.6\n10.146.146.3\n10.146.146.6", "stdout_lines":
  ↳ ["172.29.146.3", "172.29.146.6", "10.146.146.3",
  ↳ "10.146.146.6"]}
5 ...
6 TASK [docker : add system nameservers to docker options]
  ↳ *****
7 Friday 26 January 2018  14:47:13 +0100 (0:00:01.729)
  ↳ 0:04:30.460 *****
8 ok: [node3] => {"ansible_facts": {"docker_dns_servers":
  ↳ ["10.233.0.3", "172.29.146.3", "172.29.146.6",
  ↳ "10.146.146.3", "10.146.146.6"]}, "changed": false}
9 ...
10 TASK [docker : check number of nameservers]
  ↳ *****
11 Friday 26 January 2018  14:47:15 +0100 (0:00:01.016)
  ↳ 0:04:32.563 *****
12 fatal: [node3]: FAILED! => {"changed": false, "msg": "Too many
  ↳ nameservers. You can relax this check by set
  ↳ docker_dns_servers_strict=no and we will only use the
  ↳ first 3."}

```

Okazało się, że maszyna s8 była podłączona również na drugim interfejsie sieciowym, w związku z tym miała zbyt dużo wpisów serwerów DNS.

Rozwiązałem problem ręcznie logując się na maszynę i wyłączając drugi interfejs sieciowy komendą `ip 1 set eno1 down`.

8.3 Pierwszy dzień - uruchamianie skryptów z maszyny s6

Większość przeszkód opisałem w powyższym rozdziale, więc w tym skupię się tylko na problemach związanych z pierwszą próbą uruchomienia skryptów

na maszynie s6.

Najpierw próbowałem uruchomić skrypty na maszynach: s2, s4 i s5

```
1 cd ~/kubernetes/kubernetes-cluster
2 bin/setup-cluster-full 10.146.255.{2,4,5}
```

Po uruchomieniu okazało się, że maszyna s2 posiada tylko połowę RAMu (4GB) i nie mieszczą się na niej obrazy Dockera konieczne do uruchomienia klastra.

Kolejną próbą było uruchomienie na maszynach s4, s5, s8 i s9. Skończyło się problemami z Vaultem opisanymi w dalszych rozdziałach.

8.4 Kolejne próby uruchamiania klastra z maszyny s2

Dalsze testy przeprowadzałem na maszynach: s4, s5, s6, s8 i s9.

Najwięcej czasu spędziłem na rozwiązywaniu problemu z DNSami opisanym wyżej.

8.4.1 Generowanie inventory z HashiCorp Vault'em

Skrypt `inventory_builder.py` z Kubespray generuje wpisy oznaczające węzły jako posiadające HashiCorp Vaulta.

Uruchomienie z Vault'em zakończyło się błędem, więc wyłączyłem Vault'a rozbijając skrypt `bin/setup-cluster-full` na krok konfiguracji i krok uruchomienia, pomiędzy którymi mogłem wyedytować `inventory/inventory`.

→ `cfg`:

```
1 bin/setup-cluster-configure 10.146.255.{4,5,6,8,9}
2 bin/setup-cluster
```

Próbowałem dostosować parametr `cert_management`², żeby działał zarówno z Vault'em jak i bez, ale nie dało to żadnego skutku. Objawem było nie uruchamianie się etcd.

Uznałem, że taka konfiguracja jeszcze nie działa i zarzuciłem dalsze próby. Aby rozwiązać problem trzeba usunąć wpisy pod kategorią `[vault]` z pliku `inventory.cfg`.

²<https://github.com/kubernetes-incubator/kubespray/blob/master/docs/vault.md>

8.4.2 Niepoprawne znajdowanie adresów IP w ansible

Z jakiegoś powodu konfiguracje s6 (node3) i s8 (node4) w piątek kończyły się błędem:

```
1 TASK [kubernetes/preinstall : Stop if ip var does not match
   ↪ local ips] *****
2 Friday 26 January 2018 16:37:48 +0100 (0:00:01.297)
   ↪ 0:00:48.587 *****
3 fatal: [node4]: FAILED! => {
4     "assertion": "ip in ansible_all_ipv4_addresses",
5     "changed": false,
6     "evaluated_to": false
7 }
8 fatal: [node3]: FAILED! => {
9     "assertion": "ip in ansible_all_ipv4_addresses",
10    "changed": false,
11    "evaluated_to": false
12 }
```

Trzy dni później nie wprowadzając po drodze żadnych zmian uruchomiłem klaster bez problemu. W związku z tym nie wiem co było jego przyczyną.

8.4.3 Dostęp do Kubernetes Dashboardu

Kubernetes Dashboard jest dostępny pod poniższą ścieżką HTTP:

```
1 /api/v1/namespaces/kube-system/services/https:kubernetes-
   ↪ dashboard:/proxy/#!/service/default/kubernetes
```

Można się do niego dostać na dwa poniższe sposoby:

1. `kubectl proxy`, które wystawia dashboard na adresie `http://127.0.0.1:8001`
↪
2. Pod adresem `https://10.146.225.4:6443`, gdzie 10.146.225.4 to adres IP dowolnego mastera, w tym przypadku maszyny s4

Kompletne adresy to:

```
1 http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/
   ↪ https:kubernetes-dashboard:/proxy/#!/service/default/
   ↪ kubernetes
2 https://10.146.225.4:6443/api/v1/namespaces/kube-system/
   ↪ services/https:kubernetes-dashboard:/proxy/#!/service/
   ↪ default/kubernetes
```

Przekierowanie portów

Jeżeli nie pracujemy z maszyny uczelnianej porty możemy przekierować przez SSH na następujące sposoby (jeżeli skrypty uruchamialiśmy z maszyny s2 i łączymy się do mastera na maszynie s4):

1. Plik ~/.ssh/config:

```
1 Host s2
2   LocalForward 127.0.0.1:8001 localhost:8001
3   LocalForward 127.0.0.1:6443 10.146.225.4:6443
```

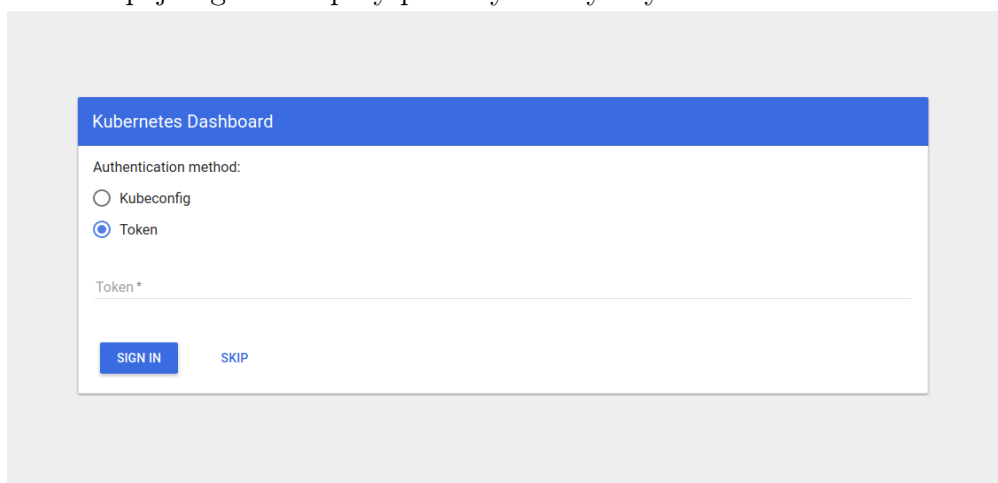
2. Argumenty ssh, np.:

```
1 ssh -L 8001:localhost:8001 -L 6443:10.146.225.4:6443
   ↪ nazarewk@s2
```

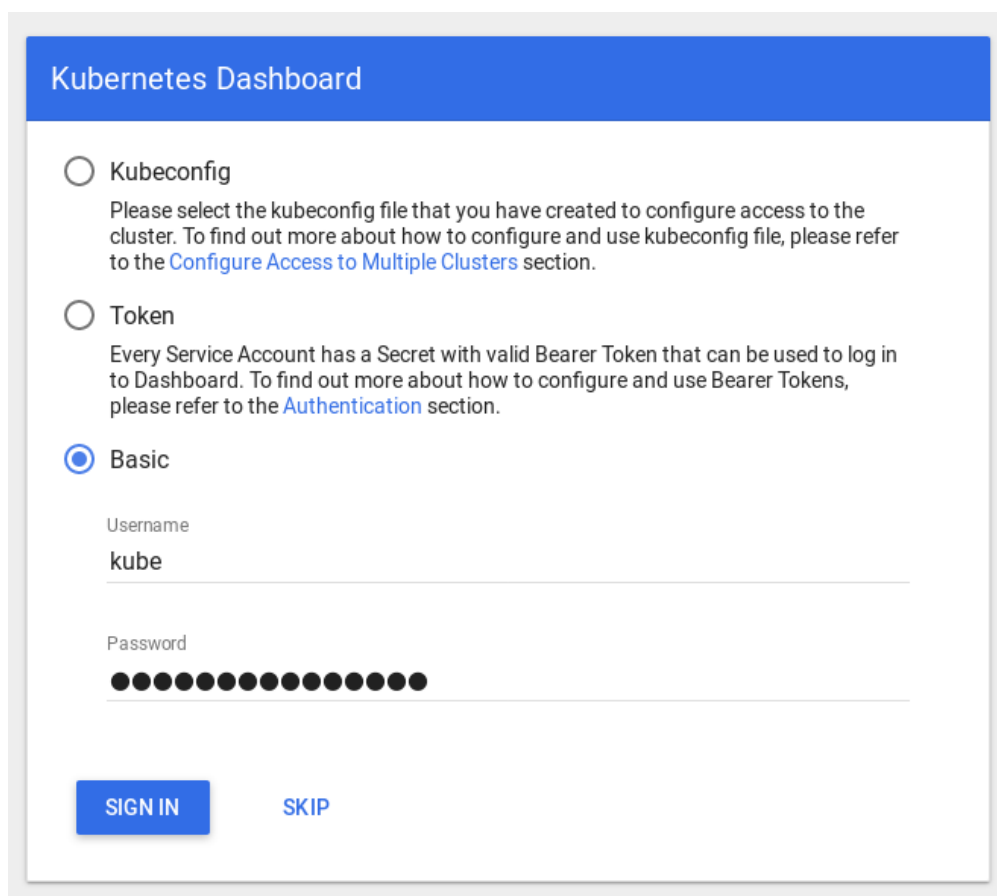
Użytkownik i hasło

Domyślna nazwa użytkownika Dashboardu to kube, a hasło znajduje się w pliku credentials/kube_user.

W starszej wersji (uruchamianej wcześniej) Kubernetes i/lub Kubespray brakowało opcji logowania przy pomocy nazwy użytkownika i hasła:



Od dnia 29 stycznia 2018 widzę poprawny ekran logowania (opcja Basic):

The image shows the login interface of the Kubernetes Dashboard. At the top is a blue header with the text "Kubernetes Dashboard". Below the header, there are three radio button options for authentication: "Kubeconfig", "Token", and "Basic". The "Basic" option is selected. Under "Kubeconfig", there is a paragraph explaining that the user should select a kubeconfig file and refer to the "Configure Access to Multiple Clusters" section. Under "Token", there is a paragraph explaining that every Service Account has a Secret with a valid Bearer Token and refers to the "Authentication" section. Under "Basic", there are two input fields: "Username" with the value "kube" and "Password" which is masked with dots. At the bottom, there are two buttons: "SIGN IN" (blue) and "SKIP" (blue text).

8.4.4 Instalacja dodatkowych aplikacji z użyciem Kubespray

- uruchamiane playbookiem `upgrade-cluster.yml` z tagiem `apps`, skrypt `bin/upgrade-cluster`
- trzeba było zmienić `kube_script_dir` na lokalizację z poza `/usr/` → `local/bin`, bo w systemie diskless jest read-only squashfs, wybrałem `/opt/bin` ponieważ znajdował się już w `PATH`ie na CoreOS,
- po doczytaniu na CoreOS zawsze powinien być `/opt/bin`

Rozdział 9

Q&A

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

Można zestawić lokalny rejestr Dockera¹ jako proxy cachujące².

9.2 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.

9.3 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.