

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 środowiska Kubernetes na maszynach
bezdyskowych

Krzysztof Nazarewski

nr albumu 240579

promotor
mgr inż. Andrzej Toboła

WARSZAWA 2018

Implementacja środowiska Kubernetes na maszynach bezdyskowych

Streszczenie

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

Zacznę od wyjaśnienia pojęcia kontenerów, problemu orkiestracji nimi i krótkiego teoretycznego przeglądu dostępnych rozwiązań. Opiszę czym jest Kubernetes, jaką ma architekturę oraz przedstawię podstawowe pojęcia pozwalające na zrozumienie i korzystanie z niego. Opis Kubernetes zakończę przedstawieniem sposobów jego uruchomienia na maszynach bezdyskowych.

Następnie przeprowadzę krótki teoretyczno-praktyczny przegląd systemów operacyjnych i sposobów uruchamiania Kubernetes na nich.

Po ich wybraniu przeprowadzę testy na sieci uczelnianej, a na koniec doprowadzę ją do stanu docelowego pozwalającego na przeprowadzenie laboratoriów Kubernetes.

Słowa kluczowe: Kubernetes, konteneryzacja, orkiestracja, maszyny bezdyskowe

Implementing Kubernetes on diskless machines

Abstract

Primary goal of this document is to present basic concepts related to Kubernetes and running it on diskless systems.

I will start with explaining what are containers, problem of their orchestration and I will theoretically inspect available solutions.

I will describe what is Kubernetes, provide overview of its architecture and basic concepts allowing to understand and use it. I will end the description with overview of its provisioning tools working on diskless systems.

Then I will research and conduct brief practical tests of available operating systems and provisioning tools.

After selecting solutions I will conduct practical tests on university network and finally configure Kubernetes to run the network.

Keywords: Kubernetes, containerization, orchestration, diskless systems

WARSZAWA, 1 lutego 2018

POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa inżynierska pt. Implementacja 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	Wstęp	1
2	Przegląd pojęć i systemów związanych z konteneryzacją	2
2.1	Konteneryzacja	2
2.1.1	Open Container Initiative	2
2.1.2	Docker	3
2.2	Kubernetes	3
2.3	Alternatywne rozwiązania zarządzania kontenerami	3
2.3.1	Fleet	4
2.3.2	Docker Swarm	4
2.3.3	Nomad	4
2.3.4	Mesos	4
3	Kubernetes	5
3.1	Administracja, a korzystanie z klastra	5
3.2	Konfiguracja klastra	5
3.3	Infrastruktura Kubernetes	6
3.3.1	Węzeł zarządzający	7
3.3.2	Węzeł roboczy	7
3.3.3	Wtyczka sieciowa	8
3.3.4	Komunikacja sieciowa	8
3.3.5	Zarządzanie dostępami	9
3.4	Architektura	10
3.4.1	Obiekty Kubernetes API	10
3.4.2	Podstawowe rodzaje obiektów aplikacyjnych	11
3.5	Kubernetes Dashboard	14
3.6	Kubernetes Incubator	14
3.7	Administracja klastrem z linii komend	15
3.7.1	kubeadm	15
3.7.2	Kubespary	15
3.7.3	OpenShift Ansible	15

3.7.4	Canonical distribution of Kubernetes	15
3.7.5	Bootkube i Typhoon	16
3.7.6	Eksperymentalne i deprekowane rozwiązania	16
3.8	Administracja klastrem za pomocą graficznych narzędzi	17
3.8.1	Rancher	17
3.8.2	OpenShift by Red Hat	17
3.8.3	DC/OS	17
4	Systemy bezdyskowe	18
4.1	Proces uruchamiania maszyny bezdyskowej	18
5	Przegląd systemów operacyjnych	20
5.1	Konfigurator cloud-init	20
5.1.1	Dostępne implementacje	21
5.2	CoreOS	22
5.2.1	Konfiguracja	22
5.3	RancherOS	22
5.3.1	Konfiguracja	23
5.4	Project Atomic	23
5.5	Alpine Linux	24
5.6	ClearLinux	24
5.7	Wnioski	25
6	Praktyczne rozeznanie w narzędziach administracji klastrem	
	Kubernetes	26
6.1	kubespary-cli	26
6.1.1	Napotkane problemy	27
6.1.2	Wnioski	27
6.2	Rancher 2.0	27
6.2.1	Testowanie tech preview 1 (v2.0.0-alpha10)	28
6.2.2	Wnioski	29
6.3	OpenShift Origin	30
6.3.1	Korzystanie ze sterownika systemd zamiast domyślnego cgroupfs	30
6.3.2	Próba uruchomienia serwera na Arch Linux	31
6.3.3	Próba uruchomienia serwera na Fedora Atomic Host w VirtualBox'ie	32
6.3.4	Wnioski	34
6.4	kubespary	34
6.4.1	Kubernetes Dashboard	34
6.4.2	Napotkane błędy	35

6.4.3	Finalne skrypty konfiguracyjne	35
6.5	Wnioski	36
7	Uruchamianie Kubernetes w laboratorium 225	37
7.1	Przygotowanie węzłów CoreOS	37
7.2	Przeszkody związane z uruchamianiem skryptów na uczelnianym Ubuntu	38
7.2.1	Brak virtualenv'a	38
7.2.2	Klonowanie repozytorium bez logowania	38
7.2.3	Atrybut wykonywalności skryptów	39
7.2.4	Konfiguracja dostępu do maszyn bez hasła	39
7.2.5	Problemy z siecią	39
7.2.6	Limit 3 serwerów DNS	39
7.3	Pierwszy dzień - uruchamianie skryptów z maszyny s6	40
7.4	Kolejne próby uruchamiania klastra z maszyny s2	40
7.4.1	Generowanie inventory z HashiCorp Vault'em	41
7.4.2	Niepoprawne znajdowanie adresów IP w ansible	41
7.4.3	Dostęp do Kubernetes Dashboardu	42
7.4.4	Instalacja dodatkowych aplikacji z użyciem Kubespray	44
8	Docelowa konfiguracja w sieci uczelnianej	45
8.1	Procedura uruchomienia klastra	45
8.2	Sprawdzanie czy klaster działa	48
8.2.1	Korzystanie z klastra jako student	48

Rozdział 1

Wstęp

Rozdział 2

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.

2.1 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.

2.1.1 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

2.1.2 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.

Domyślnie obrazy są pobierane przez internet z Docker Huba⁷, co jest ograniczone przepustowością łącza. Wolnemu łączu możemy zaradzić kieszonując zapytania HTTP ub uruchamiając rejestr obrazów⁸ w sieci lokalnej. Lokalny rejestr może być ograniczony do obrazów ręcznie w nim umieszczonych lub udostępniać i kieszonować obrazy z zewnętrznego rejestru (np. Docker Hub). Pierwsze rozwiązanie w połączeniu z zablokowaniem dostępu do zewnętrznych rejestrów daje prawie pełną kontrolę nad obrazami uruchamianymi wewnątrz sieci.

2.2 Kubernetes

Kubernetes⁹ (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.

2.3 Alternatywne rozwiązania zarządzania kontenerami

- Choosing the Right Containerization and Cluster Management Tool¹⁰

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://hub.docker.com/>

⁸<https://docs.docker.com/registry/deploying/>

⁹<https://kubernetes.io/>

¹⁰<https://dzone.com/articles/choosing-the-right-containerization-and-cluster-management-tool>

2.3.1 Fleet

Fleet¹¹ jest nakładką na systemd¹² 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.

2.3.2 Docker Swarm

Docker Swarm¹³ jest rozwiązaniem orkiestracji kontenerami od twórców samego Docker'a. Proste w obsłudze, ale nie oferuje tak dużych możliwości jak inne rozwiązania.

2.3.3 Nomad

Nomad¹⁴ 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 Software¹⁵

2.3.4 Mesos

Apache Mesos¹⁶ jest najbardziej zaawansowanym i najlepiej skalującym się rozwiązaniem orkiestracji kontenerami. Jest również najbardziej skomplikowanym i trudnym w zarządzaniu rozwiązaniem.

Materiały:

- An Introduction to Mesosphere¹⁷

¹¹<https://github.com/coreos/fleet>

¹²<https://www.freedesktop.org/wiki/Software/systemd/>

¹³<https://docs.docker.com/engine/swarm/>

¹⁴<https://www.nomadproject.io/intro/index.html>

¹⁵<https://www.nomadproject.io/intro/vs/index.html>

¹⁶<http://mesos.apache.org/>

¹⁷<https://www.digitalocean.com/community/tutorials/an-introduction-to-mesosphere>

Rozdział 3

Kubernetes

Materiały:

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

3.1 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.

3.2 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

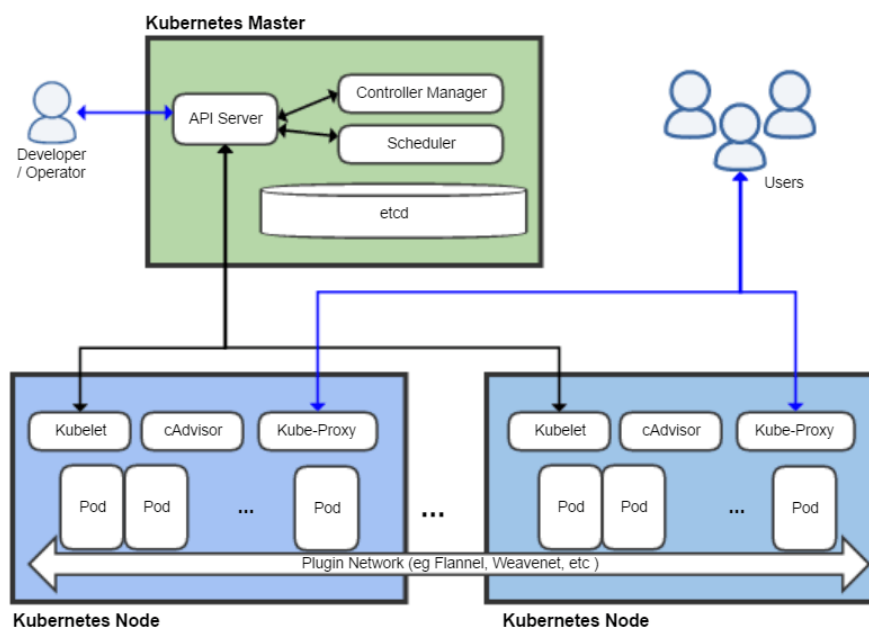
bez zmiany infrastruktury brakujące 2 instancje nigdy nie zostaną uruchomione. W momencie dołączenia kolejnego węzła klastra może się okazać, że posiadamy już oczekiwane zasoby i aplikacja zostanie uruchomiona w pełni.

W drugim przypadku założymy, że aplikacja jest uruchomiona w 9 kopiach na 4 węzłach, po 2 kopie 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.

3.3 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 znalezionej 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,

3.3.1 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). Konceptyjnie 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 zewnętrzni użytkownicy jak i wewnętrzne procesy klastra korzystają z interfejsu aplikacyjnego REST (ang. REST API) klastra w celu uzyskania informacji i zmiany jego stanu.

Głównym modułem 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.

Modułem 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.

3.3.2 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.

3.3.3 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.

3.3.4 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-b6dedeb95f3

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 klaster,

3.3.5 Zarządzanie dostępami

Podstawowymi pojęciami związanymi z zarządzaniem dostępami w Kubernetes są uwierzytelnianie, autoryzacja i `Namespace`.

Uwierzytelnianie

Pierwszym krokiem w każdym zapytaniu do API jest uwierzytelnienie, czyli weryfikacja, że użytkownik (czy to aplikacja) jest tym za kogo się podaje. Podstawowymi sposobami uwierzytelniania są:

- certyfikaty klienta X509,
- statyczne przepustki (ang. token),
- przepustki rozruchowe (ang. bootstrap tokens),
- statyczny plik z hasłami,
- przepustki kont serwisowych (ang. `ServiceAccount` tokens),
- przepustki OpenID Connect,
- Webhook (zapytanie uwierzytelniające do zewnętrznego serwisu),
- proxy uwierzytelniające,

Ze względu na prostotę rozwiązania w tej pracy będę korzystał z `ServiceAccount`

→ .

Autoryzacja

Drugim krokiem jest autoryzacja, czyli weryfikacja, że użytkownik jest uprawniony do korzystania z danego zasobu.

Najpopularniejszym sposobem autoryzacji jest RBAC (Role Based Access Control)¹⁰. Odbywa się ona na podstawie ról (`Role` i `ClusterRole`), które nadają uprawnienia i są przypisywane konkretnym użytkownikom lub kontom przez `RoleBinding` i `ClusterRoleBinding`.

`Namespace` (przestrzeń nazw) jest logicznie odseparowaną częścią klastra Kubernetes. Pozwala na współdzielenie jednego klastra przez wielu niezauważanych użytkowników. Standardowym zastosowaniem jest wydzielanie środowisk produkcyjnych, QA i deweloperskich.

¹⁰<https://kubernetes.io/docs/admin/authorization/rbac/>

Jak nazwa wskazuje role z dopiskiem `Cluster` mogą dać dostęp do wszystkich przestrzeni nazw jednocześnie oraz zasobów takowych nie posiadających. Przykładem zasobu nie posiadającego swojej przestrzeni nazw jest węzeł (`Node`) lub zakończenie API `/healthz`.

Role bez dopisku `Cluster` operują w ramach jednej przestrzeni nazw.

3.4 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.

3.4.1 Obiekty Kubernetes API

Obiekty Kubernetes¹¹ są trwale przechowywane w `etcd` i definiują, jak wcześniej wyjaśniłem, pożądany stan klastra. Szczegółowy opis konwencji API obiektów możemy znaleźć w odnośniku¹².

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

¹¹<https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>

¹²<https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md>

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,

3.4.2 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

Pod¹³ jest najmniejszą jednostką aplikacyjną w Kubernetes. Reprezentuje nierozłącznie powiazaną (np. współdzielonymi zasobami) grupę jednego lub 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.

¹³<https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/>

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

Dodatkowo kontenery wewnątrz Podów obsługują 2 rodzaje próbników¹⁴: `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`¹⁵ 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`¹⁶ pozwala na deklaratywne aktualizacje Podów i `ReplicaSet` → ów. Korzystanie z ww. bezpośrednio nie jest zalecane.

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

Rewizje `Deployment`u są zmieniane tylko w momencie `rollout`u. Operacja operacja skalowania nie uruchamia `rollout`u, 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 `Deployment`u (poprzednie wersje Podów) komendą `kubectl rollout undo`,
- skalowanie `Deployment`u 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 `ReplicaSet`ów przez ograniczanie liczby wpisów w `.spec.revisionHistoryLimit`,


¹⁴<https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes>

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

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

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 , którymi ma zarządzać. Selektor powinien zgadzać się ze zdefiniowanym szablonem.

StatefulSet

`StatefulSet`¹⁷ jest kontrolerem podobnym do Deploymentu, ale umożliwiającym zachowanie stanu Podów.

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,

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

DaemonSet

`DaemonSet`¹⁸ 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

`Job`¹⁹ pozwala na jednorazowe uruchomienie Podów, które wykonują akcję i się kończą. Istnieją 3 tryby wykonania: niezrównoległony, równoległy i równoległy z zewnętrzną kolejką zadań.

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

`CronJob`²⁰ pozwala na tworzenie Jobów jednorazowo o określonym czasie lub je powtarzać zgodnie ze specyfikacją `cron`²¹.

3.5 Kubernetes Dashboard

Kubernetes Dashboard²² jest wbudowanym interfejsem graficznym klastra Kubernetes. Umożliwia monitorowanie i zarządzanie klastrem w ramach funkcjonalności samego Kubernetes.

3.6 Kubernetes Incubator

Kubernetes Incubator²³ 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:

¹⁸<https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>

¹⁹<https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/>

²⁰<https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/>

²¹<https://en.wikipedia.org/wiki/Cron>

²²<https://github.com/kubernetes/dashboard>

²³<https://github.com/kubernetes/community/blob/master/incubator.md>

- 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`

3.7 Administracja klastrem z linii komend

3.7.1 kubeadm

kubeadm²⁴ jest narzędziem pozwalającym na niskopoziomowe zarządzanie klastrem Kubernetes. Stąd trendem jest bazowanie na kubeadm przy tworzeniu narzędzi z wyższym poziomem abstrakcji.

- Install with kubadm²⁵

3.7.2 Kubespray

kubespray²⁶ jest zbiorem skryptów Ansibla konfigurujących klastry na różnych systemach operacyjnych i w różnych konfiguracjach. W tym jest w stanie skonfigurować klastery bare metal bez żadnych zewnętrznych zależności.

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

3.7.3 OpenShift Ansible

Konfiguracja OpenShift Origin realizowana jest zestawem skryptów Ansible'owych rozwijanych jako projekt openshift-ansible²⁷.

3.7.4 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 Service²⁸.

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

²⁴<https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm/>

²⁵<https://kubernetes.io/docs/setup/independent/install-kubeadm/>

²⁶<https://github.com/kubernetes-incubator/kubespray>

²⁷<https://github.com/openshift/openshift-ansible>

²⁸

Przydatne materiały: - Juju Charm the Canonical distribution of Kubernetes²⁹ - Install Kubernetes with conjure-up³⁰ - Kubernetes the not so easy way³¹ opisuje instalację lokalnego klastra.

3.7.5 Bootkube i Typhoon

Bootkube³² jest narzędziem napisanym w Go pozwalającym na konfigurację Kubernetes na własnych maszynach.

Proponowanym trybu bare-metal³³ jest wykorzystanie Terraform i Typhoon³⁴ do realizacji automatycznej konfiguracji klastra w trakcie uruchamiania węzłów CoreOS.

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

Niestety w trakcie pisania pracy przegapiłem wzmiankę (przypis na jednej z podstron dokumentacji) o możliwości uruchomienia w trybie bezdyskowym i całkowicie odrzuciłem to narzędzie.

3.7.6 Eksperymentalne i deprekowane rozwiązania

- Fedora via Ansible³⁵ deprekowane na rzecz kubespray
- Rancher Kubernetes Installer³⁶ jest eksperymentalnym rozwiązaniem wykorzystywanym w Rancher 2.0,

kubespray-cli

Jest to narzędzie ułatwiające korzystanie z kubespray. Według użytkowników oficjalnego Slacka kubespray³⁷ kubespray-cli jest deprekowane i powinno się korzystać z czystego kubespray.

²⁹<https://jujucharms.com/canonical-kubernetes/>

³⁰<https://tutorials.ubuntu.com/tutorial/install-kubernetes-with-conjure-up>

³¹<https://insights.ubuntu.com/2017/10/12/kubernetes-the-not-so-easy-way/>

³²<https://github.com/kubernetes-incubator/bootkube>

³³[https://github.com/coreos/matchbox/tree/master/examples/terraform/](https://github.com/coreos/matchbox/tree/master/examples/terraform/bootkube-install)
bootkube-install

³⁴<https://github.com/poseidon/typhoon>

³⁵[https://kubernetes.io/docs/getting-started-guides/fedora/fedora_](https://kubernetes.io/docs/getting-started-guides/fedora/fedora_ansible_config/)
ansible_config/

³⁶<http://rancher.com/announcing-rke-lightweight-kubernetes-installer/>

³⁷<https://kubernetes.slack.com/messages/kubespray>

3.8 Administracja klastrem za pomocą graficznych narzędzi

3.8.1 Rancher

Rancher³⁸ 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.

3.8.2 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 Origin³⁹ konfigurowany przez OpenShift Ansible.

Materiały:

- OpenShift Origin vs Kubernetes⁴⁰
- The Differences Between Kubernetes and OpenShift⁴¹
- Demo konsoli⁴² (niestety po hebrajsku)

3.8.3 DC/OS

Datacenter Operating System⁴³ jest częścią Mesosphere⁴⁴ i Mesosa. Niedawno został rozszerzony o Kubernetes⁴⁵ jako alternatywnego (w stosunku do Marathon⁴⁶) systemu orkiestracji kontenerami.

³⁸<https://rancher.com/>

³⁹<https://github.com/openshift/origin>

⁴⁰https://www.reddit.com/r/devops/comments/59ql4r/openshift_origin_vs_kubernetes/

⁴¹<https://medium.com/level-consulting/the-differences-between-kubernetes-and-openshift-aef7>

⁴²<https://youtu.be/-mFovK19aB4?t=6m54s>

⁴³<https://dcos.io/>

⁴⁴<https://mesosphere.com/>

⁴⁵<https://mesosphere.com/blog/kubernetes-dcos/>

⁴⁶<https://mesosphere.github.io/marathon/>

Rozdział 4

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

4.1 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 5

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,

5.1 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.

5.1.1 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.

5.2 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.

Od 30 stycznia 2018 został wykupiony przez Red Hat¹¹, co sugeruje, że jest lepszym rozwiązaniem od niżej wymienionych Atomic Hostów.

5.2.1 Konfiguracja

Konfiguracja przez Container Linux Config¹² transpilowany do Ignition¹³. 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.

5.3 RancherOS

RancherOS¹⁴ jest systemem operacyjnym, w którym tradycyjny system inicjalizacji został zastąpiony trzema poziomami Dockera¹⁵:

- `bootstrap_docker` działający w initramie, czyli przygotowujący system,
- `system-docker` zastępuje tradycyjnego inita, zarządza wszystkimi programami systemowymi,

⁹<https://clearlinux.org/blogs/announcing-clr-cloud-init>

¹⁰<https://coreos.com/>

¹¹<https://www.redhat.com/en/about/press-releases/red-hat-acquire-coreos-expanding-its-kubernetes-and-containers-leadership>

¹²<https://coreos.com/os/docs/latest/provisioning.html>

¹³<https://coreos.com/ignition/docs/latest/>

¹⁴<https://rancher.com/rancher-os/>

¹⁵<http://rancher.com/docs/os/latest/en/configuration/docker/>

- `docker` standardowy `docker`, interakcja z nim nie może uszkodzić działającego systemu,

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`¹⁶, więc na chwilę obecną odrzucam ten system operacyjny w dalszych rozważaniach.

5.3.1 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`¹⁷.

5.4 Project Atomic

Project Atomic¹⁸ 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 Host¹⁹

¹⁶<https://github.com/rancher/os/issues/2204>

¹⁷<https://forums.rancher.com/t/good-cloud-config-reference/5238/3>

¹⁸<https://www.projectatomic.io/>

¹⁹<https://www.redhat.com/en/resources/enterprise-linux-atomic-host-datasheet>

- CentOS Atomic Host²⁰
- Fedora Atomic Host²¹

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

Atomic Host są konfigurowane systemem oficjalną implementacją cloud-inita.

5.5 Alpine Linux

Alpine Linux²² 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 Backup²³ lub Alpine Configuration Framework²⁴.

5.6 ClearLinux

ClearLinux²⁵ 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 dokumentacji systemu operacyjnego.

Materiały:

²⁰<https://wiki.centos.org/SpecialInterestGroup/Atomic/Download/>

²¹<https://getfedora.org/atomic/download/>

²²<https://alpinelinux.org/>

²³https://wiki.alpinelinux.org/wiki/Alpine_local_backup

²⁴http://wiki.alpinelinux.org/wiki/Alpine_Configuration_Framework_Design

²⁵<https://clearlinux.org/>

- 6 key points about Intel's hot new Linux distro²⁶

5.7 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.

²⁶<https://www.infoworld.com/article/3159658/linux/6-key-points-about-intels-hot-new-linux-distro.html>

Rozdział 6

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.

6.1 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-os`³ 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 $@
7     do
```

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>

```

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

```

6.1.1 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`.

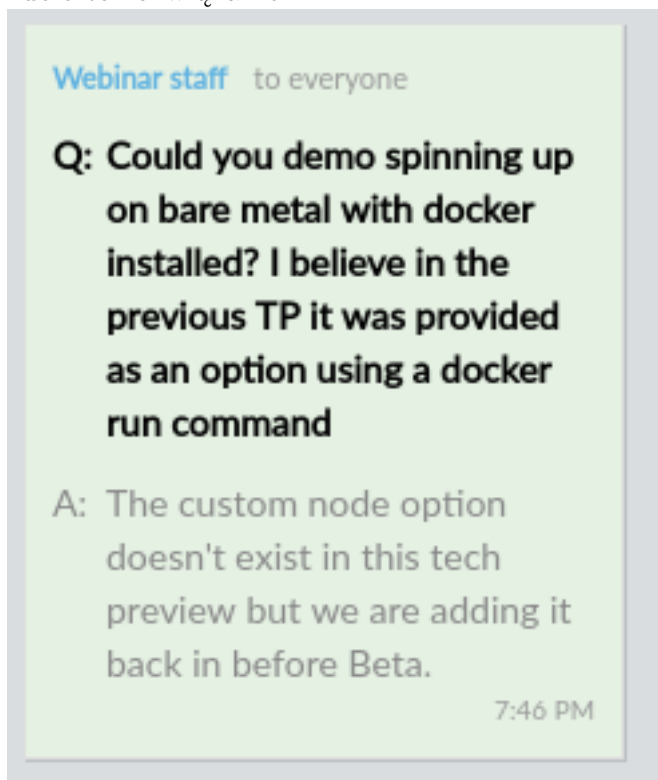
6.1.2 Wnioski

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.

6.2 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.



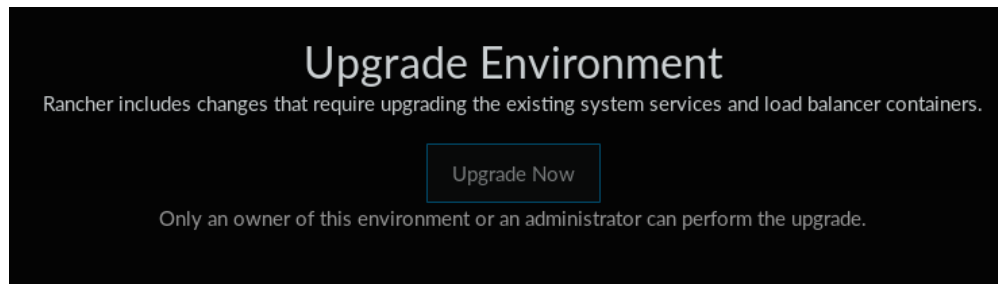
6.2.1 Testowanie tech preview 1 (v2.0.0-alpha10)

```
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
```



Rysunek 6.1: Błąd pt. Upgrade Environment

```
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.

6.2.2 Wnioski

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

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

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

6.3 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

6.3.1 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"
```

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

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

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
```

6.3.2 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:

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
```

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

4. Wystartowanie aplikacji:

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

5. Odczekanie 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.

6.3.3 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/
    ↪ openshift.local.volumes:rslave \
11 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 OpenShift:

```

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

```

W internecie nie znalazłem żadnych informacji na temat tego błędu. Próbowałem również uzyskać pomoc na kanale #openshift na irc.freenode.net, ale bez skutku.

6.3.4 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.

6.4 kubespray

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

6.4.1 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

```
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>

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>

6.4.2 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
    ↪ kubect1 (/opt/bin/kubect1 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
    ↪ kubect1 (/opt/bin/kubect1 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"}
```

6.4.3 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 kubect1 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}/kubect1 proxy
```

6.5 Wnioski

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

Rozdział 7

Uruchamianie Kubernetes w laboratorium 225

7.1 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
```

7.2 Przeszkody związane z uruchamianiem skryptów na uczelnianym Ubuntu

7.2.1 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`.

7.2.2 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`).

7.2.3 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`.

7.2.4 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
```

7.2.5 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]`.

7.2.6 Limit 3 serwerów DNS

Napotkałem limit 3 serwerów DNS¹:

```
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"]}
```

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

```

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`.

7.3 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.

7.4 Kolejne próby uruchamiania klastra z maszyny s2

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

Najwięcej czasu spędziłem na rozwiązaniu problemu z DNSami opisanym wyżej.

7.4.1 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`.

7.4.2 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.

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

7.4.3 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
```

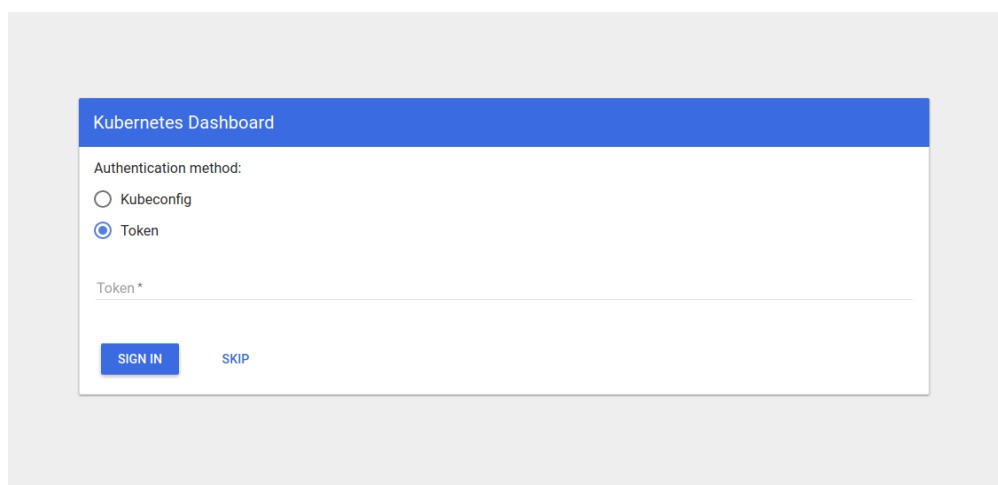
2. Argumenty `ssh`, np.:

```
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:



Kubernetes Dashboard

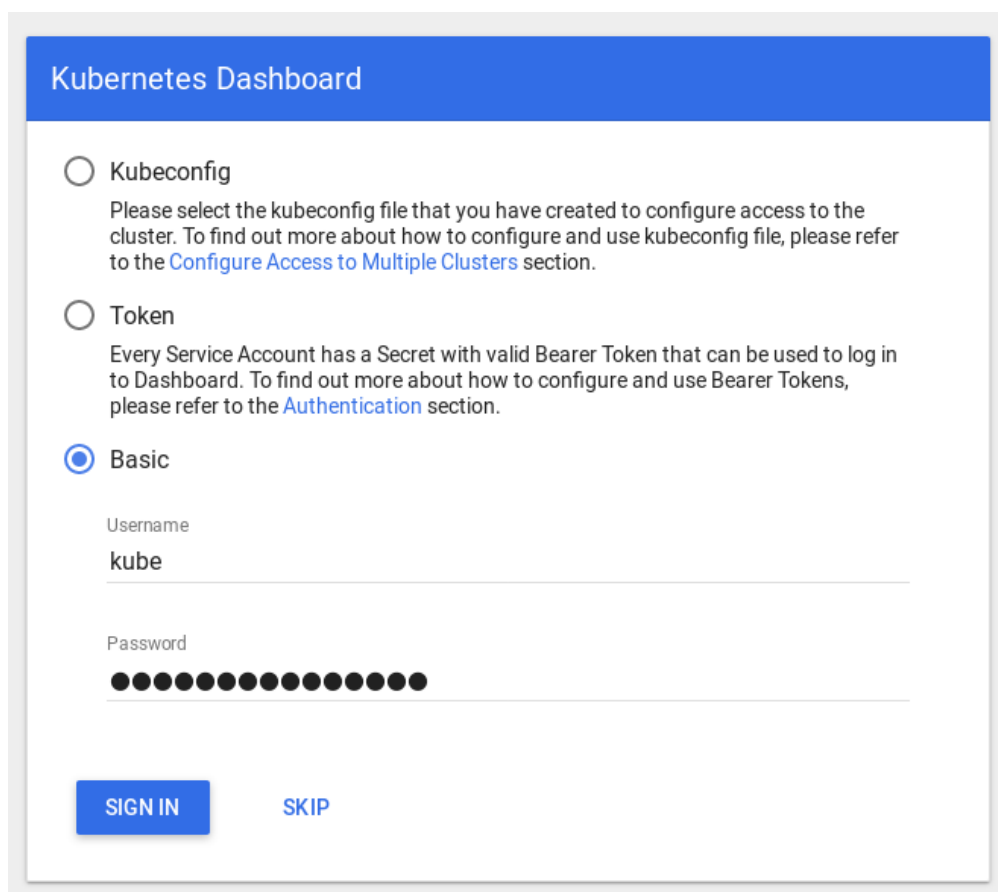
Authentication method:

☐ Kubeconfig

☒ Token

Token *

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



Kubernetes Dashboard

☐ Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the [Configure Access to Multiple Clusters](#) section.

☐ Token

Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the [Authentication](#) section.

☒ Basic

Username

kube

Password

●●●●●●●●●●●●●●●●

7.4.4 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 PATHie na CoreOS,
- po doczytaniu na CoreOS zawsze powinien być `/opt/bin`

Rozdział 8

Docelowa konfiguracja w sieci uczelnianej

Pełna konfiguracja Kubernetes znajduje się w folderze `/pub/Linux/CoreOS`
↪ `/zetis/kubernetes`, możemy ją uruchomić z maszyny `ldap`, znajdują się w nim foldery:

- `kubernetes-cluster` - moje repozytorium zawierające konfigurację i skrypty pozwalające uruchomić klaster,
- `boot` - skrót do folderu `kubernetes-cluster/zetis/WWW/boot` zawierającego konfigurację iPXE oraz Ignition:
 - `coreos.ign` - plik konfiguracyjny CoreOS, wygenerowany z pliku `coreos.yml` narzędziem do transpilacji konfiguracji `ct`¹, narzędzie domyślnie nie jest skompilowane na FreeBSD i musimy uruchomić je z Linuxa,
- `log` - standardowe wyjście uruchamianych komend,

8.1 Procedura uruchomienia klastra

1. Wchodzimy na maszynie `ldap` do folderu `/pub/Linux/CoreOS/zetis/`
↪ `kubernetes/kubernetes-cluster`
2. Upewniamy się, że nasz klucz SSH znajduje się w `boot/coreos.ign`
3. Włączamy maszyny-węzły wybierając z menu iPXE CoreOS -> Kubernetes lub wybierając w narzędziu `boot` bezpośrednio `coreos kub`. Następnie upewniamy się, że mamy bezhasłowy dostęp do tych maszyn, minimalna konfiguracja `~/.ssh/config`:

¹<https://github.com/coreos/container-linux-config-transpiler>

```

1 Host s?
2   User admin
3
4 Host *
5   User nazarewk
6   IdentityFile ~/.ssh/id_rsa
7   IdentitiesOnly yes
8
9 Host s? 10.146.255.*
10  StrictHostKeyChecking no
11  UserKnownHostsFile /dev/null

```

4. Upewniamy się, że istnieje folder `kubespray/my_inventory`, jeżeli nie to go tworzymy kopiując domyślną konfigurację:

```

1 cp -rav kubespray/inventory kubespray/my_inventory

```

5. Otwieramy plik `inventory/inventory.cfg` i upewniamy się, że uruchomione maszyny są obecne w sekcji `[all]` oraz przypisane do odpowiednich ról: `[kube-master]` i `[etcd]` lub `[kube-node]`. Identyfikatorem maszyny jest pierwsze słowo w grupie `[all]`, przykładowa konfiguracja dla maszyn `s4`, `s5` i `s6` z jednym zarządcą to:

```

1 [all]
2 ;s3  ansible_host=s3 ip=10.146.225.3
3 s4   ansible_host=s4 ip=10.146.225.4
4 s5   ansible_host=s5 ip=10.146.225.5
5 s6   ansible_host=s6 ip=10.146.225.6
6 ;s7  ansible_host=s7 ip=10.146.225.7
7 ;s8  ansible_host=s8 ip=10.146.225.8
8 ;s9  ansible_host=s9 ip=10.146.225.9
9
10 [kube-master]
11 s4
12
13 [kube-node]
14 s5
15 s6
16
17 [etcd]
18 s4
19
20 [k8s-cluster:children]
21 kube-node
22 kube-master

```


opcjonalnie możemy do każdego węzła dopisać `ansible_python_interpreter`
→ `=/opt/bin/python` żeby ułatwić uruchamianie ansible partiami

6. Upewniamy się, że plik `inventory/group_vars/all.yml` zawiera naszą konfigurację, minimalny przykład:

```
1 cluster_name: zetis-kubernetes
2 bootstrap_os: coreos
3 kube_basic_auth: true
4 kubeconfig_localhost: true
5 kubectl_localhost: true
6 download_run_once: true
7 cert_management: "{{ 'vault' if groups.get('vault', None) else '
  → script' }}"
8 helm_enabled: true
9 helm_deployment_type: docker
10 kube_script_dir: /opt/bin/kubernetes-scripts
```

7. Uruchamiamy konfigurowanie maszyn `bin/setup-cluster` lub bez skryptu:

```
1 ldap% cd kubespray
2 ldap% ansible-playbook -i my_inventory/inventory.cfg cluster.yml -b
  → -v
```

Po około 10-20 minutach skrypt powinien zakończyć się wpisami pokroju:

```
1 ...
2 PLAY RECAP *****
3 localhost                : ok=2    changed=0    unreachable=0
  → failed=0
4 s4                       : ok=281  changed=94   unreachable=0
  → failed=0
5 s5                       : ok=346  changed=80   unreachable=0
  → failed=0
6 s6                       : ok=186  changed=54   unreachable=0
  → failed=0
7 ...
```

8. Zweryfikujmy instalację:

```
1 ldap% bin/kubectl get nodes
2 NAME      STATUS   ROLES    AGE   VERSION
3 s4        Ready   master   2m    v1.9.1+coreos.0
4 s5        Ready   node     2m    v1.9.1+coreos.0
5 s6        Ready   node     2m    v1.9.1+coreos.0
```

8.2 Sprawdzanie czy klastr działa

Wywołanie skryptu `bin/students nazarewk create` jest równoważne uruchomieniu komendy `kubectl create -f nazarewk.yml`, gdzie plik `nazarewk`.

↪ yml to:

```
1 apiVersion: v1
2 kind: Namespace
3 metadata:
4   name: nazarewk
5   labels:
6     name: nazarewk
7 ---
8 apiVersion: v1
9 kind: ServiceAccount
10 metadata:
11   name: nazarewk
12   namespace: nazarewk
13 ---
14 kind: RoleBinding
15 apiVersion: rbac.authorization.k8s.io/v1
16 metadata:
17   name: nazarewk-admin-binding
18   namespace: nazarewk
19 roleRef:
20   kind: ClusterRole
21   name: admin
22 apiGroup: rbac.authorization.k8s.io
23 subjects:
24 - kind: ServiceAccount
25   name: nazarewk
```

Czyli: - stworzeniu Namespace - skonfigurowaniu dostępu za pomocą ServiceAccount

↪ - przypisaniu domyślnej Role o nazwie admin do ServiceAccount o nazwie nazarewk za pomocą RoleBinding

8.2.1 Korzystanie z klastra jako student

- stwórzmy użytkownika z jego własnym Namespace

```
1 ldap% bin/students nazarewk create
2 namespace "nazarewk" created
3 serviceaccount "nazarewk" created
4 rolebinding "nazarewk-admin-binding" created
5 Tokens:
6 eyJhbGciOiJSUzI1NiIsImtpdiI6IHR5cGU6ImF1dG8iLCJ1aWQiOiJhHfxU-TRw
7 ldap% bin/students
8 NAME          STATUS      AGE
```

```

9 default      Active    3m
10 kube-public  Active    3m
11 kube-system  Active    3m
12 nazarewk     Active    16s

```

- skopiujmy token na s2 z uruchomionym ubuntu:

```

1 ldap% bin/student-tokens nazarewk | ssh nazarewk@s2 "cat > /tmp/
  ↪ token"

```

- pobierzmy kubectl

```

1 s2% cd /tmp
2 s2% curl -LO https://storage.googleapis.com/kubernetes-release/
  ↪ release/$(curl -s https://storage.googleapis.com/kubernetes-
  ↪ release/release/stable.txt)/bin/linux/amd64/kubectl
3 sw% chmod +x kubectl
4 s2% sudo mv kubectl /usr/local/bin
5 s2% source <(kubectl completion zsh)

```

- sprawdźmy czy mamy dostęp do klastra

```

1 s2% kubectl get nodes
2 The connection to the server localhost:8080 was refused - did you
  ↪ specify the right host or port?

```

- skonfigurujmy kubectl (najprościej aliasem)

```

1 s2% alias kubectl='command kubectl -s "https://s4:6443" --insecure-
  ↪ skip-tls-verify=true --token="$(cat /tmp/token)" -n nazarewk'

```

- zweryfikujmy brak dostępu do zasobów globalnych

```

1 s2% kubectl get nodes
2 Error from server (Forbidden): nodes is forbidden: User "system:
  ↪ serviceaccount:nazarewk:nazarewk" cannot list nodes at the
  ↪ cluster scope

```

- stwórzmy deployment z przykładową aplikacją

```

1 s2% kubectl run echoserver \
2 --image=gcr.io/google_containers/echoserver:1.4 \
3 --port=8080 \
4 --replicas=2
5 deployment "echoserver" created
6 s2% kubectl get deployments
7 NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
8 echoserver     2         2         2            2           3m
9 s2% kubectl get pods
10 NAME                                READY   STATUS    RESTARTS   AGE
11 echoserver-7b9bbf6ff-22df4          1/1     Running   0          4m
12 echoserver-7b9bbf6ff-c6kbv          1/1     Running   0          4m

```

- wystawmy port, żeby dostać się do aplikacji z poza klastra

```

1 s2% kubectl expose deployment echoserver --type=NodePort
2 service "echoserver" exposed
3 s2% kubectl describe services/echoserver | grep -e NodePort:
4 NodePort:                <unset> 30100/TCP
5 s2% curl s4:30100
6 CLIENT VALUES:
7 client_address=10.233.107.64
8 command=GET
9 real path=/
10 query=nil
11 request_version=1.1
12 request_uri=http://s4:8080/
13
14 SERVER VALUES:
15 server_version=nginx: 1.10.0 - lua: 10001
16
17 HEADERS RECEIVED:
18 accept=/*/*
19 host=s4:30100
20 user-agent=curl/7.47.0
21 BODY:
22 -no body in request-

```

- sprawdźmy, czy z ldap'a też mamy dostęp do aplikacji:

```

1 ldap% curl s4:30100
2 CLIENT VALUES:
3 client_address=10.233.107.64
4 command=GET
5 real path=/
6 query=nil
7 request_version=1.1

```

```

8 request_uri=http://s4:8080/
9
10 SERVER VALUES:
11 server_version=nginx: 1.10.0 - lua: 10001
12
13 HEADERS RECEIVED:
14 accept=/*/*
15 host=s4:30100
16 user-agent=curl/7.58.0
17 BODY:
18 -no body in request-

```

- usuńmy użytkownika

```

1 ldap% bin/students nazarewk delete
2 namespace "nazarewk" deleted
3 serviceaccount "nazarewk" deleted
4 rolebinding "nazarewk-admin-binding" deleted
5 Tokens:
6 Error from server (NotFound): serviceaccounts "nazarewk" not found

```

- sprawdźmy czy coś zostało po koncie użytkownika

```

1 ldap% curl s4:30100
2 curl: (7) Failed to connect to s4 port 30100: Connection refused
3 ldap% bin/kubectl get namespace
4 NAME          STATUS    AGE
5 default        Active    46m
6 kube-public    Active    46m
7 kube-system    Active    46m

```