

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 pracy jest przegląd zagadnień związanych z systemem Kubernetes oraz przykładowa implementacja klastra na maszynach bezdyskowych.

W pierwszej części przedstawione zostały pojęcia kontenerów, wymienione dostępne narzędzia zarządzania nimi oraz

W drugiej części przedstawione zostały: Kubernetes, jego architektura i podstawowe zagadnienia pozwalające na zrozumienie i korzystanie z niego. Rozdział zakończy się teoretycznym przeglądem narzędzi konfigurujących klastry Kubernetes na sprzęcie bezdyskowym.

TODO: NAPISAC TO

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, zarządzanie kontenerami, maszyny bezdyskowe

Implementing Kubernetes on diskless machines

Abstract

Keywords: Kubernetes, containerization, container orchestration, diskless systems

TODO: WRITE THIS

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
1.1	Charakterystyka systemów bezdyskowych	2
2	System Kubernetes	4
2.1	Struktura klastra Kubernetes	4
2.2	Architektura	8
2.3	Kubernetes Incubator	12
2.4	Zarządzanie klastrem z linii komend	13
2.5	Zarządzanie klastrem za pomocą narzędzi graficznych	14
2.6	Kubernetes Dashboard	14
2.7	Lista materiałów dodatkowych	15
3	Praktyczny przegląd bezdyskowych systemów operacyjnych	16
3.1	Konfigurator cloud-init	17
3.2	CoreOS	18
3.3	RancherOS	18
3.4	Project Atomic	19
3.5	Alpine Linux	19
3.6	ClearLinux	20
3.7	Wnioski	20
4	Praktyczne rozeznanie w narzędziach administracji klastrem Kubernetesa	21
4.1	kubespary-cli	21
4.2	Rancher 2.0	22
4.3	OpenShift Origin	24
4.4	kubespary	28
4.5	Wnioski	29
5	Uruchamianie Kubernetesa w laboratorium 225	30
5.1	Przygotowanie węzłów CoreOS	30

5.2	Przeszkody związane z uruchamianiem skryptów na uczelnianym Ubuntu	31
5.3	Pierwszy dzień - uruchamianie skryptów z maszyny s6	32
5.4	Kolejne próby uruchamiania klastra z maszyny s2	33
6	Docelowa konfiguracja w sieci uczelnianej	38
6.1	Procedura uruchomienia klastra	38
6.2	Sprawdzanie, czy klaster działa	41
7	Rezultaty i wnioski	45
7.1	Lista alternatyw Kubernetes	45
A	Wykaz odnośników	47
B	Wykaz skryptów	50
B.1	repozytorium kubernetes-cluster	50
B.2	repozytorium ipxe-boot	59

Rozdział 1

Wstęp

Historia wirtualizacji i przydzielania zasobów sięga już pierwszych systemów komputerowych.

Podstawową przeszkodą w korzystaniu z komputerów była ich wysoka cena; bogate instytucje odpłatnie wynajmowały swoje systemy innym firmom. Umożliwiając dostęp tylko jednemu użytkownikowi jednocześnie, maszyny przez większość czasu nie pracowały. Wraz ze wzrostem liczby obsługiwanych użytkowników przypadających na jedną maszynę zwiększało się wysycenie jej zasobów, a zatem i zyski.

Następną przeszkodą okazał się brak ograniczeń dostępowych dla poszczególnych użytkowników: każdy świadomie lub nie mógł uszkodzić cudze zasoby. W ten sposób zrodził się pomysł `chroot`, czyli izolacji użytkowników na poziomie systemu plików.

Wraz z rozwojem technologii zwiększała się różnorodność sprzętowa i systemowa. W związku z pierwszą systemy operacyjne oferowały abstrakcje dostępu do sprzętu oraz powstawały narzędzia emulujące inny sprzęt. Różnorodność systemowa była natomiast abstrahowana przez aplikacyjne wirtualne maszyny, które umożliwiały jednnorodny dostęp do zasobów niejednorodnych systemów operacyjnych, ale w żaden sposób go nie ograniczały.

W centrach danych zyskała popularność wirtualizacja sprzętowa. Pozwala ona uruchomić na jednej fizycznej maszynie wiele niezależnych od siebie maszyn wirtualnych, na których działają dowolne systemy operacyjne. W toku rozwoju technologii po roku 2000 pojawił się trend rozpraszania oprogramowania i dążenia do uruchamiania jak najmniejszej jego funkcjonalnej części na pojedynczej maszynie wirtualnej. Główną zaletą takiego podejścia jest bezpieczeństwo: po uzyskaniu dostępu do jednego z serwerów włamywacz nadal nie ma dostępu do reszty infrastruktury. Wadą natomiast jest to, że jedna maszyna fizyczna jest w stanie uruchomić nie więcej niż kilka do kilkunastu maszyn wirtualnych. W związku z tym utrzymywanie całej maszyny

wirtualnej w celu uruchamiania pojedynczego procesu aplikacyjnego często jest nieekonomiczne.

Na przełomie ostatniej dekady zaczęły pojawiać się narzędzia pozwalające uruchamiać wiele w pełni odizolowanych procesów współdzielących jedynie jądro systemu operacyjnego. Znacznie zmniejszyło to narzut zasobów wymaganych do uruchamiania kolejnych aplikacji-procesów i umożliwiło współdzielenie jednej maszyny przez setki, a nawet tysiące programów. Metodologia ta zyskała miano konteneryzacji i jest z powodzeniem wykorzystywana w systemach produkcyjnych na całym świecie.

Technologia konteneryzacji jest młoda, a konkurencja duża. Z jednej strony trwa wyścig o stworzenie najlepszego rozwiązania, a z drugiej aktywnie rozwijane są otwarte standardy konteneryzacji w ramach Open Container Initiative ↪¹, ponieważ nikt nie może pozwolić sobie na niekompatybilność swojej implementacji. Obecnie najpopularniejszą implementacją kontenerów jest Docker ↪.

Aspektem równie ważnym jak sama implementacja kontenerów, jest kwestia zarządzania nimi. O ile kilkoma serwerami z kilkudziesięcioma kontenerami może zarządzać człowiek, o tyle do zarządzania tysiącami kontenerów wymagany jest w dużej mierze zautomatyzowany i wyspecjalizowany system.

Obecnie przodującym rozwiązaniem tego problemu jest Kubernetes (zamiennie `k8s`). Projekt powstał niedawno (w połowie 2014 roku) i mimo tego, że coraz bardziej umacnia swoją pozycję na rynku nadal umie go skonfigurować tylko nieliczne grono specjalistów. Celem niniejszej pracy jest przedstawienie sposobu konfiguracji klastra `k8s` na fizycznych maszynach* na przykładzie sieci komputerowej Wydziału Elektrycznego Politechniki Warszawskiej. Laboratoria komputerowe dostępne w tej sieci są wyposażone w maszyny bezdyskowe, a zatem należy pokrótce scharakteryzować tego typu systemy.

1.1 Charakterystyka systemów bezdyskowych

W dzisiejszych czasach rozwój sieci komputerowych i spadek cen pamięci RAM pozwolił na konfigurację maszyn nie posiadających dysku twardego. Bezdyskowe maszyny przy starcie pytają centralny serwer o swoją konfigurację (np. protokołem `PXE`), a następnie umieszczają kompletny system operacyjny w pamięci RAM.

W momencie wyłączenia maszyny cały stan jest tracony, a przy ponownym rozruchu konfigurowana jest od nowa. w celu trwałego przechowywania

*ang. *bare metal*, czyli uruchamianie na prostej sieci komputerowej bez zewnętrznych zależności, w odróżnieniu od kompleksowych usług chmurowych, które w znacznym stopniu automatyzują i ułatwiają konfigurację

danych systemy bezdyskowe korzystają z zewnętrznych serwerów plikowych.

Rozwiązanie pozwala to na jednorazową konfigurację bazowego systemu operacyjnego i uzyskanie konfiguracji na dowolnej ilości maszyn bez ingerencji człowieka. W efekcie koszt utrzymania centrów danych przenosi się z konfiguracji indywidualnych maszyn na konfigurację pojedynczych usług działających na dowolnej liczbie maszyn.

Zarządzanie klastrem `ks` poza środowiskiem chmurowym znacznie ogranicza wybór narzędzia konfiguracyjnego, a bezdyskowość jego węzłów ma duży wpływ na wybór systemu operacyjnego.

Rozdział 2

System Kubernetes

Kubernetes² został stworzony przez firmę Google na bazie ich wewnętrznego systemu Borg, a następnie udostępniony na zasadach otwartego oprogramowania (ang. *open-source*). Charakteryzuje się korzystnym stosunkiem kosztów zarządzania do oferowanych możliwości.

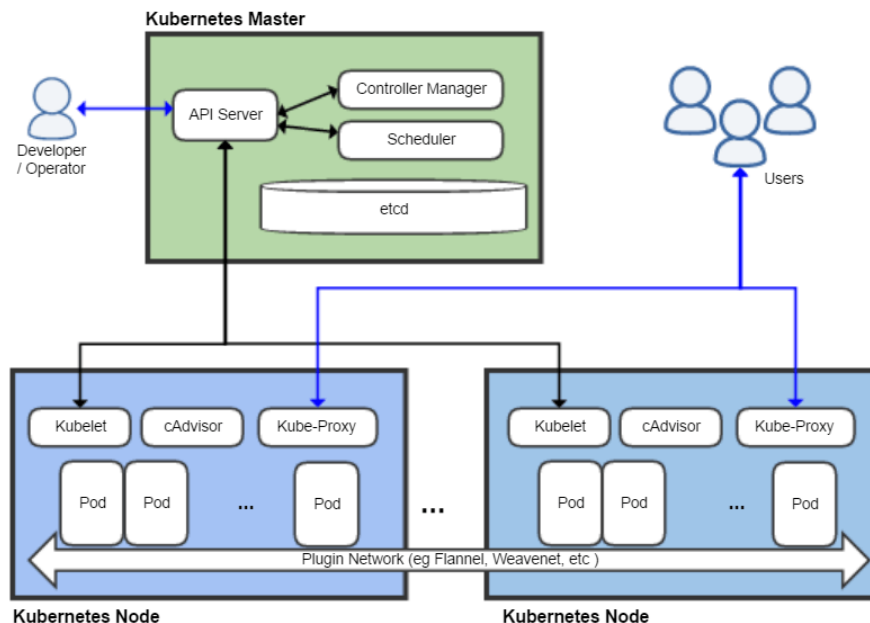
Zarządzanie klastrem rozumiem jako zbiór czynności i procesów polegających na przygotowaniu klastra do użytku oraz definiowanie aplikacji realizującymi usługi na poziomie klastra, takimi jak:

- tworzenie klastra,
- dodawanie i usuwanie fizycznych węzłów,
- integracja z sieciowym systemem plikowym,
- integracja z zewnętrznymi systemami zarządzania kontami użytkowników,
- nadawanie uprawnień użytkownikom i procesom,
- centralny system zbierania dzienników aplikacji (ang. *centralized logging system*),
- centralny system monitorujący obciążenie węzłów,

Wykorzystanie klastra rozumiem jako uruchamianie dowolnych aplikacji na wcześniej skonfigurowanym klastrze.

2.1 Struktura klastra Kubernetes

/TODO: write this



Na ilustracji wyróżnione zostało 5 grup funkcjonalnych:

1. administrator lub programista korzystający z klastra - Developer / Operator
→ ,
2. użytkowników końcowi aplikacji działających w klastrze - Users ,
3. jeden lub więcej węzeł zarządzający - Kubernetes Master ,
4. jeden lub więcej węzeł roboczy - Kubernetes Node ,
5. wtyczka sieciowa (Plugin Network), czyli oprogramowanie realizujące komunikację sieciową w ramach klastra k8s ,

Węzeł zarządzający

Stan k8s jest przechowywany w etcd³. 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-apiserver⁴. 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 modulem zarządzającym, który dba o doprowadzenia klastra do oczekiwanego stanu jest `kube-controller-manager`⁵. 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`⁶. Decyduje on na których węzłach uruchamiać aplikacje, żeby zaspokoić popyt na zasoby jednocześnie nie przeciążając pojedynczych węzłów klastra.

Węzeł roboczy

Podstawowym procesem działającym na węzłach roboczych jest `kubelet` ↪⁷. Monitoruje i kontroluje kontenery działające w ramach jednego węzła. 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`⁸ (Container Advisor), który monitoruje zużycie zasobów i wydajność kontenerów w ramach jednego klastra.

Wtyczka sieciowa

Podstawowym założeniem `k8s` jest posiadanie własnego adresu IP przez każdą aplikację działającą w klastrze, ale nie narzuca żadnego rozwiązania je realizującego.

Administrator (lub skrypt konfigurujący) 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: Flannel⁹ i Project Calico¹⁰.

Komunikacja sieciowa

Materiały źródłowe:

- <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:

- k8s 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,
- k8s ma wiele sposobów na wystawienie Serwisów poza klastery,

Zarządzanie dostępami

Podstawowymi pojęciami związanymi z zarządzaniem dostępami w k8s 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 klienckie 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 i uniwersalność rozwiązania w tej pracy będzie 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)¹¹. 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 `k8s`. Pozwala na współdzielenie jednego klastra przez wielu niezauważanych użytkowników. Standardowym zastosowaniem jest wydzielanie środowisk produkcyjnych, QA i deweloperskich.

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.

2.2 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 `k8s`.

Stan w klastrze Kubernetes zdefiniowany jest jako konfiguracja, do której klaster dąży.

Status w klastrze Kubernetes zdefiniowany jest jako rzeczywista konfiguracja, w której klaster się w danej chwili znajduje.

Zwykle stan i status klastra szybko się ze sobą zbiegają, ale nie jest to regułą. Dwoma najczęstszymi powodami braku ich synchronizacji są niewystarczające zasoby lub utrata 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, czyli stan aplikacji pokryje się z jej statusem.

W drugim przypadku mając aplikację uruchomioną w 4 replikach na 2 węzłach, po utraceniu drugiego węzła stan aplikacji będzie nadal wskazywał na 4 repliki, a status na 2. Następnie w ciągu kilku do kilkudziesięciu sekund

moduł kontrolujący klastr zauważył braki, uruchomił brakujące 2 repliki i przez to doprowadził status klastra do stanu docelowego.

Obiekty Kubernetes API

Obiekty Kubernetesa¹² są trwale przechowywane w etcd i definiują, jak wcześniej wyjaśniłem, pożądany stan klastra. Szczegółowy opis konwencji API obiektów możemy znaleźć w odnośniku¹³.

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

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
  uid: 343fc305-c854-44d0-9085-baed8965e0a9
  labels:
    resources: high
  annotations:
    app-type: qwe
spec:
  containers:
  - image: ubuntu:trusty
    command: ["echo"]
    args: ["Hello World"]
  ...
status:
  podIP: 127.12.13.14
  ...
```

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

- `apiVersion`: obowiązkowa wersja API k8s,
- `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,

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 `Pod`¹⁴ jest najmniejszą jednostką aplikacyjną w `k8s`. 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.

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

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

Dodatkowo kontenery wewnątrz `Pod`ów obsługują 2 rodzaje próbników¹⁵: `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`¹⁶ jest następcą `ReplicaController`, czyli imperatywnym kontrolerem dbającym o działanie określonej liczby `Pod`ów w klastrze.

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

Deployment `Deployment`¹⁷ pozwala na deklaratywne aktualizacje `Pod`ów i `ReplicaSet`ów. Korzystanie z ww. bezpośrednio nie jest zalecane.

Zmiany `Deployment`ów wprowadzane są przez tak zwane `rollouty`. Każdy ma swój status i może zostać wstrzymany lub przerwany. `Rollouty` mogą

zostać aktywowane automatycznie przez zmianę specyfikacji Podów 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`,

Przykładowy Deployment tworzący 3 repliki serwera nginx:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - 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`¹⁸ jest kontrolerem podobnym do `Deployment` u, ale umożliwiającym zachowanie stanu Podów.

W przeciwieństwie do `Deployment` `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 `DaemonSet`¹⁹ 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`²⁰ 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 uruchamianie są kolejne Pody aż zostanie uzyskana odpowiednia liczba sukcesów.

`CronJob`²¹ pozwala na tworzenie Jobów jednorazowo o określonym czasie lub je powtarzać zgodnie ze specyfikacją `cron`²².

2.3 Kubernetes Incubator

`Kubernetes Incubator`²³ gromadzi projekty rozszerzające `k8s`, 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 `k8s`,

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

2.4 Zarządzanie klastrem z linii komend

kubeadm

`kubeadm`²⁴ jest narzędziem pozwalającym na niskopoziomowe zarządzanie klastrem `k8s`. Stąd trendem jest bazowanie na `kubeadm` przy tworzeniu narzędzi z wyższym poziomem abstrakcji.

- Install with `kubadm`²⁵

Kubespray

`kubespray`²⁶ 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.

OpenShift Ansible

Konfiguracja `OpenShift Origin` realizowana jest zestawem skryptów Ansible'owych rozwijanych jako projekt `openshift-ansible`²⁷.

Canonical distribution of Kubernetes

Jest to prosta w instalacji dystrybucja `k8s`. Niestety wymaga infrastruktury chmurowej do uruchomienia klastra składającego się z więcej niż jednego węzła.

Opcja `bare metal`, która by mnie interesowała nadal wymaga działającego środowiska `Metal as a Service`²⁸.

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

Materiały źródłowe:

- pakiet (`Charm`) w oficjalnym repozytorium `Juju`²⁹
- materiał szkoleniowy dot. uruchamiania `k8s`³⁰
- opis instalacji lokalnego klastra³¹

Bootkube i Typhoon

Bootkube³² jest narzędziem napisanym w języku Go pozwalającym skonfigurować k8s na własnych maszynach.

W instalacji bare metal³³ proponowane jest wykorzystanie Terraform i Typhoon³⁴ do realizacji automatycznej konfiguracji klastra w trakcie procesu uruchamiania węzłów CoreOS.

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

W wyniku przeoczenia wzmianki (przypis na jednej z podstron dokumentacji) o możliwości uruchomienia w trybie bezdyskowym całkowicie odrzuciłem to narzędzie. W końcowych etapach pisanie pracy znalazłem ww. wpis i zdecydowałem się zawrzeć o nim informację.

Eksperymentalne i deprekowane rozwiązania

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

kubescape-cli Jest to narzędzie ułatwiające korzystanie z kubescape. Według użytkowników oficjalnego Slacka kubescape³⁷ kubescape-cli jest deprekowane i powinno się korzystać z czystego kubescape.

2.5 Zarządzanie klastrem za pomocą narzędzi graficznych

2.6 Kubernetes Dashboard

Kubernetes Dashboard³⁸ jest wbudowanym interfejsem graficznym klastra k8s. Umożliwia monitorowanie i zarządzanie klastrem w ramach funkcjonalności samego k8s.

Rancher

Rancher³⁹ jest platformą zarządzania kontenerami umożliwiającą między innymi zarządzanie klastrem k8s. Od wersji 2.0 twórcy skupiają się wyłącznie na zarządzaniu k8s porzucając wsparcie innych rozwiązań.

OpenShift by Red Hat


OpenShift jest komercyjną usługą typu PaaS (Platform as a Service), od wersji 3 skupia się na zarządzaniu klastrem k8s.

Rdzeniem projektu jest open source'owy OpenShift Origin⁴⁰ konfigurowany przez OpenShift Ansible.

Materiały źródłowe:

- dyskusja o wykorzystaniu OpenShift Origin i k8s⁴¹
- opis różnic między OpenShift Origin i k8s⁴²
- materiał wideo przedstawiający interfejs OpenShift⁴³ (po hebrajsku)

DC/OS

Datacenter Operating System⁴⁴ jest częścią Mesosphere⁴⁵ i Mesosa. Niedawno został rozszerzony o k8s⁴⁶ jako alternatywny (w stosunku do Marathon ⁴⁷) system zarządzania kontenerami.

2.7 Lista materiałów dodatkowych

Ze względu na obszerność tematu zdecydowałem przedstawić oddzielną listę materiałów dodatkowych:

- blog Julii Evans o k8s⁴⁸,
- dokument o uruchamianiu k8s od podstaw⁴⁹,
- materiał wideo o skalowaniu k8s⁵⁰,

Rozdział 3

Praktyczny przegląd bezdyskowych systemów operacyjnych

Wszystkie moduły `k8s` są uruchamiane w kontenerach, więc dwoma podstawowymi wymaganiami systemu operacyjnego są:

- możliwość uruchomienia Dockera,
- wsparcie wybranego narzędzia konfigurującego system do działania w klastrze `k8s`,

Dodatkowe wymagania związane z opisywanym w tej pracy przypadkiem użycia:

- zdalny dostęp SSH lub możliwość konfiguracji automatycznego dołączenia do klastra `k8s`,
- wsparcie dla środowiska bezdyskowego,
- możliwość bootu PXE.

Podstawowe wyznaczniki:

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

3.1 Konfigurator cloud-init

Ze względu na obszerność i niejednoznaczność tematu cloud-init zdecydowałem się wyjaśnić wszelkie wątpliwości z nim związane.

cloud-init⁵¹ jest standardem oraz implementacją konfiguratora kompatybilnego z wieloma systemami operacyjnymi przeznaczonymi do działania w chmurze.

Standard polega na dostarczeniu pliku konfiguracyjnego w formacie YAML⁵² w trakcie lub tuż po inicjalizacji systemu operacyjnego.

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

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 Dockera⁵³ (python:alpine - 89MB i python2:alpine - 72 MB) wnioskuję, że nie istnieje mniejsza dystrybucja Pythona.

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

Dodatkowe materiały:

- Wywiad z developerem cloud-init⁵⁴

coreos-cloudinit coreos-cloudinit⁵⁵ jest częściową implementacją standardu w języku Go udostępnioną przez twórców coreOS. Rok temu przestał być rozwijany⁵⁶ i wychodzi z użytku.

RancherOS + coreos-cloudinit Rancher cloud-init⁵⁷ jest przejętym⁵⁸ coreos-cloudinit przez zespół RancherOS.

clr-cloud-init clr-cloud-init⁵⁹ jest wewnętrzną implementacją standardu dla systemu ClearLinux. Powstała z chęci optymalizacji standardu pod ClearLinux → oraz pozbycia się zależności referencyjnej implementacji od Pythona.

3.2 CoreOS

CoreOS⁶⁰ jest pierwszą dystrybucją Linuxa przeznaczoną do zarządzania 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.

30 stycznia 2018 roku został wykupiony przez Red Hat⁶¹.

Konfiguracja

Konfiguracja obsługiwana jest przez Container Linux Config⁶² transpilowany do Ignition⁶³. Transpiler konwertuje ogólną konfigurację na przygotowaną pod konkretne chmury (AWS, GCE, Azure itp.). Minusem jest brak dystrybucji transpilatora pod FreeBSD.

Poprzednikiem Ignition jest `coreos-cloudinit`.

3.3 RancherOS

RancherOS⁶⁴ jest systemem operacyjnym, w którym tradycyjny system inicjalizacji został zastąpiony trzema poziomami Dockera⁶⁵:

- `bootstrap_docker` - działający w initramie, czyli przygotowuje system,
- `system-docker` - zastępuje tradycyjny init, zarządza wszystkimi programami systemowymi,
- `docker` - standardowy Docker, interakcja z nim nie może uszkodzić działającego systemu.

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

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

W związku z bugiem w systemie RancherOS nie zawsze czyta `cloud-config`⁶⁶, więc odrzucam ten system operacyjny w dalszych rozważaniach.

Konfiguracja

Narzędzie konfigurujące RancherOS zostało przejęte `coreos-cloudinit`.

Znaczną przewagą wobec oryginału jest możliwość sekwencyjnego uruchamiania dowolnej liczby plików konfiguracyjnych.

Minimalna konfiguracja pozwalająca na zalogowanie:

```
#cloud-config
ssh_authorized_keys:
  - ssh-rsa <klucz RSA>
```

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

```
#!/bin/sh

cat << EOF > ssh.yml
#cloud-config
ssh_authorized_keys:
$(ssh-add -L | sed 's/^/ - /g')
EOF
```

Przydatne jest wyświetlenie kompletnej konfiguracji komendą `ros config export --full`⁶⁷.

3.4 Project Atomic

Project Atomic⁶⁸ jest grupą podobnie skonfigurowanych systemów operacyjnych dedykowaną środowiskom cloud i kontenerom.

Dystrybucje Project Atomic nazywają się Atomic Host. Dostępne są ich następujące warianty:

- Red Hat Atomic Host⁶⁹,
- CentOS Atomic Host⁷⁰,
- Fedora Atomic Host⁷¹.

Żadna z dystrybucji domyślnie nie wspiera rozruchu bezdyskowego, więc nie zgłębiam dalej tematu.

Atomic Host są konfigurowane oficjalną implementacją `cloud-init`.

3.5 Alpine Linux

Alpine Linux⁷² jest minimalną dystrybucją Linuxa bazowaną na `musl-libc` i `busybox`.

Wygląda bardzo obiecująco w kontekście moich 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 Backup⁷³ lub Alpine Configuration Framework⁷⁴.

3.6 ClearLinux

ClearLinux⁷⁵ jest dystrybucją Linuxa wysoko zoptymalizowaną pod procesory Intel.

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

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

Ogromnym minusem jest trudna w nawigowaniu dokumentacja systemu operacyjnego.

Materiały źródłowe:

- 6 key points about Intel's hot new Linux distro⁷⁶

3.7 Wnioski

Głównymi czynnikami odróżniającymi poszczególne systemy operacyjne są częstotliwość aktualizacji oprogramowania oraz wsparcie narzędzi. Rozbieżność reszty parametrów jest pomijalnie mała.

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

Rozdział 4

Praktyczne rozeznanie w narzędziach administracji klastrem Kubernetesa

Najpopularniejszym rozwiązaniem konfiguracji klastra k8s jest kops⁷⁷, ale jak większość rozwiązań zakłada uruchomienie w środowiskach chmurowych, PaaS lub IaaS. W związku z tym nie ma żadnego zastosowania w tej pracy inżynierskiej.

4.1 kubespray-cli

Z powodu błędu⁷⁸ logiki narzędzie nie radzi sobie z brakiem Pythona na domyślnej dystrybucji CoreOSa, mimo że sam kubespray radzi sobie z nim świetnie.

Do uruchomienia na tym systemie potrzebne jest ręczne wywołanie roli `bootstrap-os`⁷⁹ z kubespray zanim przystąpi się do właściwego deployment'u. Skrypt uruchamiający:

```
#!/bin/sh
set -e

# pip2 install ansible kubespray
get_coreos_nodes() {
  for node in $@
  do
    echo -n node1[
    echo -n ansible_host=${node},
    echo -n bootstrap_os=coreos,
    echo -n ansible_user=core,
    echo -n ansible_default_ipv4.address=${node}
    echo ]
  done
}
```

```

done
}

NODES=$(get_coreos_nodes 192.168.56.{10,12,13})
echo NODES=${NODES[@]}
kubespray prepare -y --nodes ${NODES[@]}

cat << EOF > ~/.kubespray/bootstrap-os.yml
- hosts: all
  become: yes
  gather_facts: False
  roles:
    - bootstrap-os
EOF

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

```

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 ręczne podawanie `ansible_default_ipv4 . address`.

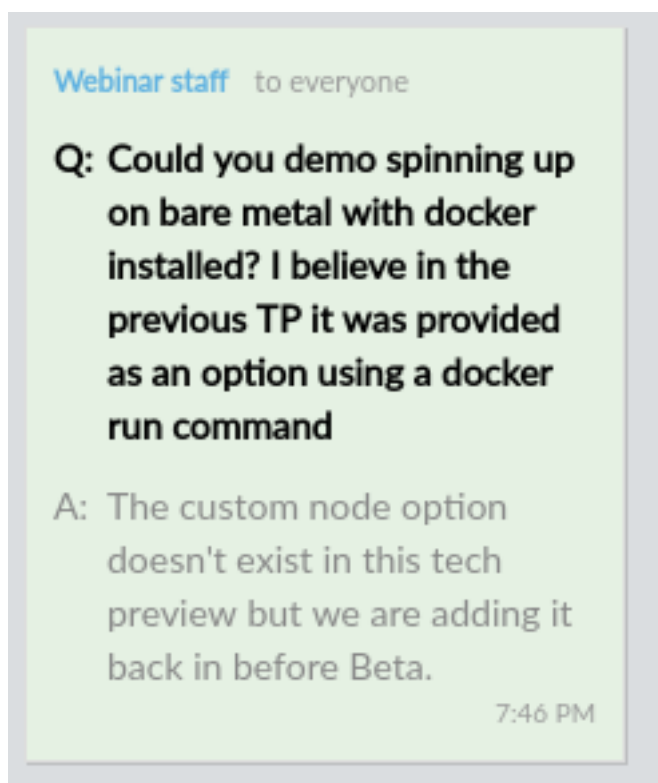
Wnioski

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.

4.2 Rancher 2.0

Jest to wygodne narzędzie do uruchamiania i monitorowania klastra `k8s` ➡, ale wymaga interakcji użytkownika. Wersja 2.0 (obecnie w fazie alpha) oferuje lepszą integrację z `k8s` 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.



Testowanie tech preview 1 (v2.0.0-alpha10)

```
#rancher_version=latest  
#rancher_version=preview  
rancher_version=v2.0.0-alpha10  
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:

```
wersja_agenta=v1.2.9  
ip_ranchera=192.168.56.1  
skrypt=B52944BEFAA613F0CE90:1514678400000:E2yB6KfxzSix4YHti39BTw5RbKw  
  
sudo docker run --rm --privileged \  
-v /var/run/docker.sock:/var/run/docker.sock \  
-v /var/lib/rancher:/var/lib/rancher \
```

```
rancher/agent:${wersja_agenta} \
http://${ip_ranchera}:8080/v1/scripts/${skrypt}
```

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

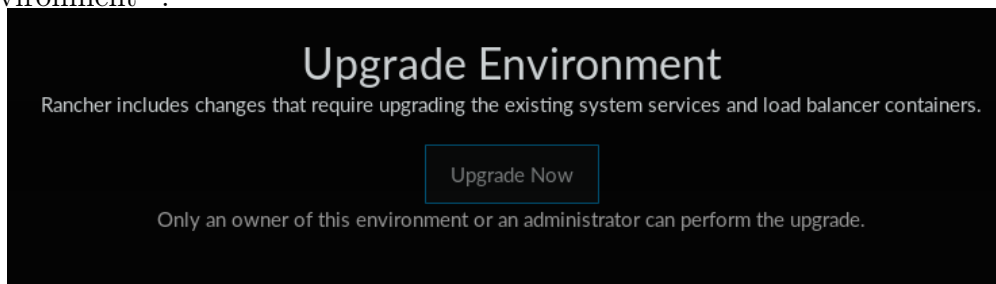
Następnie w `cloud-config` Rancher0Sa możemy dodać ww. komendę w formie:

```
#cloud-config
runcmd:
- 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:

```
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 Environment`⁸⁰.



Wnioski

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

Ze względu na brak stabilności odrzucam Ranchera jako rozwiązanie problemu uruchamiania klastra `k8s`.

4.3 OpenShift Origin

Według dokumentacji⁸¹ są dwie metody uruchamiania serwera, w Dockerze ↪ i bezpośrednio na systemie Linux.

```
# https://docs.openshift.org/latest/getting\_started/administrators.html#
  ↪ installation-methods
```



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

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

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`:

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

OpenShift natomiast konfiguruje k8s do korzystania z cgroup przez systemd ↪. Kubelet przy starcie weryfikuje zgodność silników cgroup, co skutkuje niekompatybilnością z domyślną konfiguracją Dockera⁸², czyli poniższym błędem:

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

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

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

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 Out⁸³:

1. Uruchomienie shella na serwerze:

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

2. Logowanie jako testowy użytkownik:

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

3. Stworzenie nowego projektu:

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

4. Pobranie aplikacji z Docker Hub:

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

5. Wystartowanie aplikacji:

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

6. Odczekanie aż aplikacja się uruchomi:

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

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

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

Próba uruchomienia serwera na Fedora Atomic Host w VirtualBox

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

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

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

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

Kroki 1-5 były analogiczne do uruchamiania na Arch Linux, następnie:

6. Odczekanie aż aplikacja się uruchomi i weryfikacja działania:

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

svc/deployment-example - 172.30.221.105:8080
  dc/deployment-example deploys istag/deployment-example:latest
  deployment #1 deployed 3 seconds ago - 1 pod
$ curl http://172.30.221.105:8080 | grep v1
<div class="box"><h1>v1</h1><h2></h2></div>
```

7. Aktualizacja, przebudowanie i weryfikacja działania aplikacji:

```
$ oc tag --source=docker openshift/deployment-example:v2 deployment-
↪ example:latest
Tag deployment-example:latest set to openshift/deployment-example:v2.
$ watch -n 5 oc status
In project test on server https://10.0.2.15:8443

svc/deployment-example - 172.30.221.105:8080
  dc/deployment-example deploys istag/deployment-example:latest
    deployment #2 running for 8 seconds - 1 pod
    deployment #1 deployed 8 minutes ago - 1 pod
$ curl -s http://172.30.221.105:8080 | grep v2
<div class="box"><h1>v2</h1><h2></h2></div>
```

8. Nie udało się uzyskać dostępu do panelu administracyjnego OpenShift:

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

Wnioski

Panel administracyjny klastra OpenShift Origin jest jedyną znaczącą przewagą nad Kubesprayem. Reszta zarządzania klastrem odbywa się również za pomocą repozytorium skryptów Ansibla (w tym dodawanie kolejnych węzłów klastra⁸⁴).

Z powodu braku dostępu do ww. panelu próbę uruchomienia OpenShift ↪ Origin uznaję za nieudaną i odrzucam to narzędzie.

4.4 kubespray

Cały kod znajduje się w moim repozytorium kubernetes-cluster⁸⁵.

Kubernetes Dashboard

Dostęp do Dashboardu najprościej można uzyskać poprzez:

1. nadanie wszystkich uprawnień roli kubernetes – dashboard⁸⁶,
2. Wejście pod adres `http://localhost:8001/api/v1/namespaces/kube-system`
↪ `/services/https:kubernetes-dashboard:/proxy/#!/login`,

3. Kliknięcie skip.

Materiały źródłowe:

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

Napotkane błędy

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

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

4.5 Wnioski

Na moment pisania tej pracy kubespray jest jedynym aktywnie rozwijanym i działającym rozwiązaniem uruchamiania klastra k8s.

Rozdział 5

Uruchamianie Kubernetesa w laboratorium 225

5.1 Przygotowanie węzłów CoreOS

Na wstępie przygotowałem `coreos.ipxe` i `coreos.ign` do rozruchu 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.

```
passwd:
  users:
    - name: nazarewk
      groups: [sudo, docker]
      ssh_authorized_keys:
        - ssh-rsa <klucz RSA> nazarewk
        - ssh-rsa <klucz RSA> nazarewk@ldap.iem.pw.edu.pl
```

Następnie skompilowałem go do formatu Ignition narzędziem `ct`, skryptem `bin/render-coreos` z wykazu.

Przygotowałem skrypt `IPXE` do uruchamiania CoreOS `zetis/WWW/boot/coreos.ipxe`.

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

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

5.2 Przeszkody związane z uruchamianiem skryptów na uczelnianym Ubuntu

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`.

Klonowanie repozytorium bez logowania

W celu umożliwienia anonimowego klonowania repozytorium z Githuba, zmieniłem protokół z `git` na `https`:

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

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

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`.

Konfiguracja dostępu do maszyn bez hasła

Poza konfiguracją `CoreOS` wypełniłem konfigurację `SSH` do bezhasłowego dostępu. W pliku `~/.ssh/config` umieściłem:

```
Host s?
  User admin

IdentityFile ~/.ssh/id_rsa
IdentitiesOnly yes

Host s?
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null
```

Problemy z siecią

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

Limit 3 serwerów DNS

Napotkałem limit 3 serwerów DNS⁸⁷:

```
TASK [docker : check system nameservers]
  ↳ *****
Friday 26 January 2018  14:47:09 +0100 (0:00:01.429)          0:04:26.879
  ↳ *****
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
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"]}
...
TASK [docker : add system nameservers to docker options]
  ↳ *****
Friday 26 January 2018  14:47:13 +0100 (0:00:01.729)          0:04:30.460
  ↳ *****
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}
...
TASK [docker : check number of nameservers]
  ↳ *****
Friday 26 January 2018  14:47:15 +0100 (0:00:01.016)          0:04:32.563
  ↳ *****
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`.

5.3 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

```
cd ~/kubernetes/kubernetes-cluster
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.

5.4 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.

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`:

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

Próbowałem dostosować parametr `cert_management`⁸⁸, 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`.

Niepoprawne znajdowanie adresów IP w ansible

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

```
TASK [kubernetes/preinstall : Stop if ip var does not match local ips]
  ↳ *****
Friday 26 January 2018  16:37:48 +0100 (0:00:01.297)                0:00:48.587
  ↳ *****
fatal: [node4]: FAILED! => {
  "assertion": "ip in ansible_all_ipv4_addresses",
  "changed": false,
  "evaluated_to": false
```

```
}
fatal: [node3]: FAILED! => {
  "assertion": "ip in ansible_all_ipv4_addresses",
  "changed": false,
  "evaluated_to": false
}
```

Trzy dni później nie wprowadzając po drodze żadnych zmian uruchomiłem klaster bez problemu.

Przyczyną błędu okazały się pozostałości konfiguracji maszyn niezależne ode mnie.

Dostęp do Kubernetes Dashboardu

Kubernetes Dashboard jest dostępny pod poniższą ścieżką HTTP:

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

Można się do niego dostać na dwa 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:

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

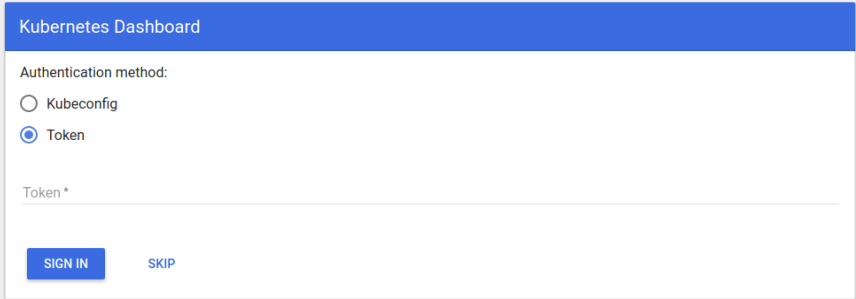
```
Host s2
  LocalForward 127.0.0.1:8001 localhost:8001
  LocalForward 127.0.0.1:6443 10.146.225.4:6443
```

2. Argumenty `ssh`, np.:

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

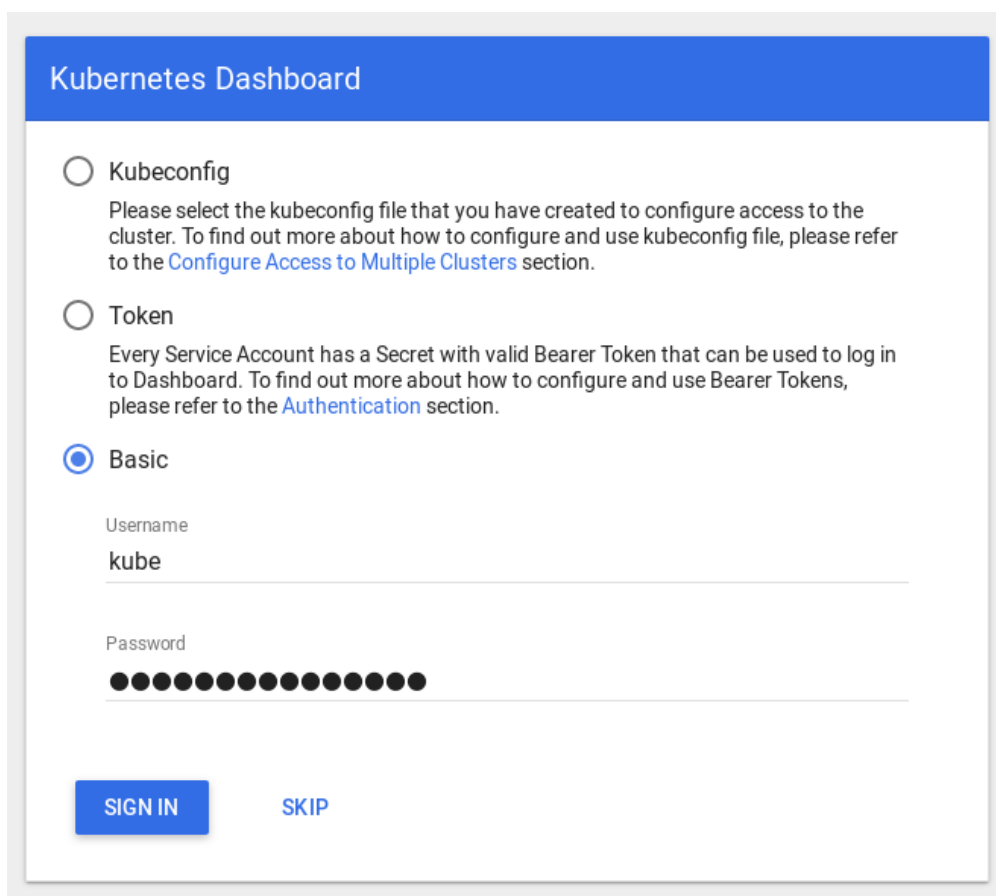
Użytkownik i hasło Domyślna nazwa użytkownika Dashboardu to `kube`, a hasło znajduje się w pliku `credentials / kube_user`.

W starszej wersji (uruchamianej wcześniej) `k8s` i/lub `Kubespray` brakowało opcji logowania przy pomocy nazwy użytkownika i hasła:



The screenshot shows the 'Kubernetes Dashboard' login interface. It features a blue header bar with the text 'Kubernetes Dashboard'. Below the header, the 'Authentication method:' section contains two radio button options: 'Kubeconfig' and 'Token'. The 'Token' option is selected, indicated by a blue dot. Below the radio buttons is a text input field labeled 'Token *'. At the bottom of the form, there are two buttons: a blue 'SIGN IN' button and a grey 'SKIP' button.

Od 29 stycznia 2018 roku 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

●●●●●●●●●●●●●●●●

SIGN IN SKIP

Instalacja dodatkowych aplikacji z użyciem Kubespray

Kubespray ma wbudowaną instalację kilku dodatkowych aplikacji playbookiem `upgrade-cluster.yml` z tagiem `apps` (skrypt `bin/setup-cluster-upgrade` ↪).

Zmieniłem `kube_script_dir` na lokalizację z poza `/usr/local/bin`, bo w systemie CoreOS jest read-only squashfs, wybrałem `/opt/bin` ponieważ znajdował się już w `PATH`ie na CoreOS. Później dowiedziałem się, że domyślnie zmiany CoreOS powinny być umieszczane w folderze `/opt`.

W końcu ze względu na liczne błędy zarzuciłem temat.

Instalacja Helm

Helm⁸⁹ jest menadżerem pakietów dla `k8s`. Jego głównym zadaniem jest standaryzacja, automatyzacja i ułatwienie instalacji aplikacji w `k8s`.

Helm składa się z:

- programu `helm` uruchamianego lokalnie i korzystającego z danych dostępowych `kubectla`,
- aplikacji serwerowej `Tiller`, z którą `helm` prowadzi interakcje,
- pakietów `Charts` i ich repozytoriów, domyślnie jest to `kubernetes / charts`
↪ `90`,

Jego instalacja sprowadza się do:

1. ściągnięcia pliku wykonywalnego dla obecnej architektury,
2. dodania roli RBAC dla `Tillera`,
3. Wywołanie komendy `helm init --service-account tiller`

Wszystkie kroki zawierają się w skrypcie `bin/install-helm`. Ze względu na brak dystrybucji `Helm` na `FreeBSD` całość uruchamiam przez `SSH` na węźle-zarządcy (domyślnie `s4`).

Szybko okazało się, że większość pakietów wymaga trwałych zasobów dyskowych i nie uda się ich uruchomić bez ich konfiguracji w sieci uczelnianej.

Rozdział 6

Docelowa konfiguracja w sieci uczelnianej

Pełną konfiguracją k8s można uruchomić z maszyny ldap; znajduje się ona w folderze `/pub/Linux/CoreOS/zetis/kubernetes` maszyny ldap, który zawiera podane 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`⁹¹, narzędzie domyślnie nie jest skompilowane na FreeBSD i musimy uruchomić je z Linuxa,
- `log` - standardowe wyjście uruchamianych komend,

6.1 Procedura uruchomienia klastra

1. Wchodzę na maszynie ldap do folderu `/pub/Linux/CoreOS/zetis/kubernetes`
→ `/kubernetes-cluster`
2. Upewniam się, że mój klucz SSH znajduje się w `boot/coreos.ign`,
3. Włączam maszyny-węzły wybierając z menu iPXE CoreOS -> k8s lub wybierając w narzędziu boot bezpośrednio `coreos kub`,
4. Upewniam się, że mam bezhasłowy dostęp do tych maszyn, minimalna konfiguracja `~/.ssh/config` to:

```
Host s?  
  User admin  
  StrictHostKeyChecking no  
  UserKnownHostsFile /dev/null  
  
Host *  
  IdentityFile ~/.ssh/id_rsa  
  IdentitiesOnly yes
```

5. Upewniam się, że istnieje folder `kubescape / my_inventory`, jeżeli nie, to go tworzę kopiując domyślną konfigurację:

```
cp -rav kubescape/inventory kubescape/my_inventory
```

6. Otwieram plik `inventory / inventory .cfg` i upewniam 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:

```
[all]  
;s3 ip=10.146.225.3  
s4 ip=10.146.225.4  
s5 ip=10.146.225.5  
s6 ip=10.146.225.6  
;s7 ip=10.146.225.7  
;s8 ip=10.146.225.8  
;s9 ip=10.146.225.9  
;sa ip=10.146.225.10  
;sb ip=10.146.225.11  
;sc ip=10.146.225.12  
  
[kube-master]  
s4  
  
[kube-node]  
s5  
s6  
  
[etcd]  
s4  
  
[k8s-cluster:children]  
kube-node  
kube-master
```

Opcjonalnie można do każdego węzła:

- dopisać `ansible_python_interpreter = /opt/bin/python`, żeby ułatwić uruchamianie ansible partiami,
- dopisać `ansible_host =< prawdziwa_nazwa_hosta >`, jeżeli chce się korzystać z pierwszego wyrazu opisu węzła jako aliasu, a nie faktycznej jego nazwy w sieci uczelnianej,

7. Upewniam się, że plik `inventory / group_vars / all . yml` zawiera naszą konfigurację; minimalny przykład:

```
cluster_name: zetis-kubernetes
bootstrap_os: coreos
kube_basic_auth: true
kubeconfig_localhost: true
kubectl_localhost: true
download_run_once: true
cert_management: "{ { 'vault' if groups.get('vault', None) else 'script'
    ↪ }}"
helm_enabled: true
helm_deployment_type: docker
kube_script_dir: /opt/bin/kubernetes-scripts
```

8. Uruchamiam konfigurowanie maszyn `bin / setup-cluster` lub bez skryptu:

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

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

```
...
PLAY RECAP *****
localhost                : ok=2      changed=0    unreachable=0
    ↪ failed=0
s4                        : ok=281   changed=94   unreachable=0
    ↪ failed=0
s5                        : ok=346   changed=80   unreachable=0
    ↪ failed=0
s6                        : ok=186   changed=54   unreachable=0
    ↪ failed=0
...
```

9. Weryfikuję instalację:

```
ldap% bin/kubectl get nodes
NAME      STATUS    ROLES    AGE      VERSION
s4        Ready     master   2m       v1.9.1_coreos.0
s5        Ready     node     2m       v1.9.1_coreos.0
s6        Ready     node     2m       v1.9.1_coreos.0
```


6.2 Sprawdzanie, czy klaster działa

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

```
apiVersion: v1
kind: Namespace
metadata:
  name: nazarewk
  labels:
    name: nazarewk
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nazarewk
  namespace: nazarewk
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nazarewk-admin-binding
  namespace: nazarewk
roleRef:
  kind: ClusterRole
  name: admin
apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: nazarewk
```

W skrócie:

- tworzę Namespace
- tworzę ServiceAccount
- przypisuję wbudowaną Role o nazwie admin do ServiceAccount o nazwie nazarewk za pomocą RoleBinding ,

Korzystanie z klastra jako student

- tworzę użytkownika z jego własnym Namespace

```
ldap% bin/students nazarewk create  
namespace "nazarewk" created  
serviceaccount "nazarewk" created  
rolebinding "nazarewk-admin-binding" created  
Tokens:  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTJwIiwiaWF0IjoxNTEyMzkwOTY5LCJpc29udGVudCI6ImFhHfxU-TRw  
ldap% bin/students
```

NAME	STATUS	AGE
default	Active	3m
kube-public	Active	3m
kube-system	Active	3m
nazarewk	Active	16s

- kopiuję token na s2 z uruchomionym ubuntu:

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

- pobieram kubectl

```
s2% cd /tmp
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
s2% chmod +x kubectl
s2% sudo mv kubectl /usr/local/bin
s2% source <(kubectl completion zsh)
```

- sprawdzam, czy mam dostęp do klastra

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

- konfiguruje kubectl (najprościej aliasem)

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

- weryfikuję brak dostępu do zasobów globalnych

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

- tworzę deployment z przykładową aplikacją

```
s2% kubectl run echoserver --image=gcr.io/google_containers/echoserver
↪ :1.4 --port=8080 --replicas=2
deployment "echoserver" created
```

```
s2% kubectl get deployments
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
echoserver	2	2	2	2	3m

```
s2% kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
echoserver-7b9bbf6ff-22df4	1/1	Running	0	4m
echoserver-7b9bbf6ff-c6kbv	1/1	Running	0	4m

- wystawiam port, żeby dostać się do aplikacji spoza klastra

```
s2% kubectl expose deployment echoserver --type=NodePort
service "echoserver" exposed
```

```
s2% kubectl describe services/echoserver | grep -e NodePort:
NodePort:                <unset> 30100/TCP
```

```
s2% curl s4:30100
CLIENT VALUES:
client_address=10.233.107.64
command=GET
real path=/
query=nil
request_version=1.1
request_uri=http://s4:8080/

SERVER VALUES:
server_version=nginx: 1.10.0 - lua: 10001

HEADERS RECEIVED:
accept=/*/*
host=s4:30100
user-agent=curl/7.47.0
BODY:
-no body in request-
```

- sprawdzam, czy z ldap też mam dostęp do aplikacji:

```
ldap% curl s4:30100
CLIENT VALUES:
client_address=10.233.107.64
command=GET
real path=
```

```
query=nil
request_version=1.1
request_uri=http://s4:8080/

SERVER VALUES:
server_version=nginx: 1.10.0 - lua: 10001

HEADERS RECEIVED:
accept=/*/*
host=s4:30100
user-agent=curl/7.58.0
BODY:
-no body in request-
```

- usuwam użytkownika

```
ldap% bin/students nazarewk delete
namespace "nazarewk" deleted
serviceaccount "nazarewk" deleted
rolebinding "nazarewk-admin-binding" deleted
Tokens:
Error from server (NotFound): serviceaccounts "nazarewk" not found
```

- sprawdzam, czy coś zostało po koncie użytkownika

```
ldap% curl s4:30100
curl: (7) Failed to connect to s4 port 30100: Connection refused
```

```
ldap% bin/kubectl get namespace
NAME          STATUS    AGE
default       Active   46m
kube-public   Active   46m
kube-system   Active   46m
```

Rozdział 7

Rezultaty i wnioski

Główne założenia pracy inżynierskiej zostały spełnione. Wyjaśniłem dużą ilość zagadnień związanych z `k8s` oraz oddałem do użytku skrypty konfigurujące klaster `k8s` wraz z prostym w obsłudze dodawaniem i usuwaniem jego użytkowników.

W trakcie pisania pracy temat okazał się zbyt obszerny, żeby go kompletnie i wyczerpująco opisać w pracy inżynierskiej. W związku z tym musiałem wybrać tylko najważniejsze informacje i przekazać je w możliwie najkrótszej formie.

Projekt jest bardzo aktywnie rozwijany, więc wiele informacji wyszło na jaw w końcowych etapach pisania pracy. W samej pracy pojawiły się jedynie wzmianki o nich bez dogłębnej analizy.

Nie udało mi się przeprowadzić testów wydajnościowych klastra ze względu na brak czasu.

7.1 Lista alternatyw Kubernetes

`Kubernetes` oferuje duże możliwości w stosunku do progu wejścia w system, ale istnieją również inne rozwiązania, o których warto wspomnieć.

Fleet jest nakładką na `systemd` realizującą rozproszony system inicjalizacji systemów operacyjnych `CoreOS`. Kontenery są uruchamiane i zarządzane przez `systemd`, a stan przechowywany jest w `etcd`.

Aktualnie projekt kończy swój żywot na rzecz `k8s` i w dniu 1 lutego 2018, został wycofany z domyślnej dystrybucji `CoreOS`. Nadal będzie dostępny w rejestrze pakietów `CoreOS`.

Docker Swarm jest rozwiązaniem zarządzania kontenerami zintegrowanym z systemem `Docker`. Główną zaletą jest niski próg wejścia i prostota, a wadą są małe możliwości w stosunku do innych rozwiązań.

Nomad jest narzędziem do zarządzania siecią komputerową, które również oferuje zarządzanie kontenerami.

Przy jego tworzeniu twórcy kierują się filozofią `unix`. W związku z tym `Nomad` jest prosty w obsłudze, wyspecjalizowany i rozszerzalny. Zwykle działa w tandemie z innymi produktami firmy “HashiCorp” jak `Consul` i `Vault`.

Mesos jest najbardziej zaawansowanym i najefektywniej skalującym się rozwiązaniem zarządzania kontenerami. Jest również najbardziej skomplikowanym i trudnym w zarządzaniu rozwiązaniem. W związku z tym znajduje swoje zastosowanie tylko w największych sieciach komputerowych o zasięgu globalnym.

Dodatek A

Wykaz odnośników

- 1 <https://www.opencontainers.org/about>
- 2 <https://kubernetes.io/>
- 3 <https://coreos.com/etcd/>
- 4 <https://kubernetes.io/docs/reference/generated/kube-apiserver/>
- 5 <https://kubernetes.io/docs/reference/generated/kube-controller-manager/>
- 6 <https://kubernetes.io/docs/reference/generated/kube-scheduler/>
- 7 <https://kubernetes.io/docs/reference/generated/kubelet/>
- 8 <https://github.com/google/cadvisor>
- 9 <https://github.com/coreos/flannel#flannel>
- 10 <https://www.projectcalico.org/>
- 11 <https://kubernetes.io/docs/admin/authorization/rbac/>
- 12 <https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>
- 13 [https://github.com/kubernetes/community/blob/master/contributors/devel/
api-conventions.md](https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md)
- 14 <https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/>
- 15 <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes>
- 16 <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>
- 17 <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>
- 18 <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>
- 19 <https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>
- 20 <https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/>
- 21 <https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/>
- 22 <https://en.wikipedia.org/wiki/Cron>
- 23 <https://github.com/kubernetes/community/blob/master/incubator.md>
- 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 <https://maas.io/>
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/bootkube-install>
34 <https://github.com/poseidon/typhoon>
35 https://kubernetes.io/docs/getting-started-guides/fedora/fedora_ansible_config/
36 <http://rancher.com/announcing-rke-lightweight-kubernetes-installer/>
37 <https://kubernetes.slack.com/messages/kubespray>
38 <https://github.com/kubernetes/dashboard>
39 <https://rancher.com/>
40 <https://github.com/openshift/origin>
41 https://www.reddit.com/r/devops/comments/59ql4r/openshift_origin_vs_kubernetes/
42 <https://medium.com/levvel-consulting/the-differences-between-kubernetes-and-openshift-aef>
43 <https://youtu.be/-mFovK19aB4?t=6m54s>
44 <https://dcos.io/>
45 <https://mesosphere.com/>
46 <https://mesosphere.com/blog/kubernetes-dcos/>
47 <https://mesosphere.github.io/marathon/>
48 <https://jvns.ca/categories/kubernetes/>
49 <https://github.com/kelseyhightower/kubernetes-the-hard-way>
50 <https://www.youtube.com/watch?v=4-pawkiazEg>
51 <https://cloud-init.io/>
52 <http://yaml.org/>
53 https://hub.docker.com/_/python/
54 <https://www.podcastinit.com/cloud-init-with-scott-moser-episode-126>
55 <https://github.com/coreos/coreos-cloudinit>
56 <https://github.com/coreos/coreos-cloudinit/commit/3460ca4414fd91de66cd581d997bf453fd895b6>
57 <http://rancher.com/docs/os/latest/en/configuration/>
58 <https://github.com/rancher/os/commit/e2ed97648ad63455743ebc16080a82ee47f8bb0c>
59 <https://clearlinux.org/blogs/announcing-clr-cloud-init>
60 <https://coreos.com/>
61 <https://www.redhat.com/en/about/press-releases/red-hat-acquire-coreos-expanding-its-kuber>
62 <https://coreos.com/os/docs/latest/provisioning.html>
63 <https://coreos.com/ignition/docs/latest/>
64 <https://rancher.com/rancher-os/>
65 <http://rancher.com/docs/os/latest/en/configuration/docker/>

66 <https://github.com/rancher/os/issues/2204>
67 <https://forums.rancher.com/t/good-cloud-config-reference/5238/3>
68 <https://www.projectatomic.io/>
69 <https://www.redhat.com/en/resources/enterprise-linux-atomic-host-datasheet>
70 <https://wiki.centos.org/SpecialInterestGroup/Atomic/Download/>
71 <https://getfedora.org/atomic/download/>
72 <https://alpinelinux.org/>
73 https://wiki.alpinelinux.org/wiki/Alpine_local_backup
74 http://wiki.alpinelinux.org/wiki/Alpine_Configuration_Framework_Design
75 <https://clearlinux.org/>
76 <https://www.infoworld.com/article/3159658/linux/6-key-points-about-intels-hot-new-linux.html>
77 <https://github.com/kubernetes/kops>
78 <https://github.com/kubespary/kubespary-cli/issues/120>
79 <https://github.com/kubernetes-incubator/kubespary/blob/master/roles/bootstrap-os/tasks/main.yml>
80 <https://github.com/rancher/rancher/issues/10396>
81 https://docs.openshift.org/latest/getting_started/administrators.html
82 <https://github.com/openshift/origin/issues/14766>
83 https://docs.openshift.org/latest/getting_started/administrators.html#try-it-out
84 https://docs.openshift.com/enterprise/3.0/admin_guide/manage_nodes.html#adding-nodes
85 <https://github.com/nazarewk/kubernetes-cluster>
86 <https://github.com/kubernetes/dashboard/wiki/Access-control#admin-privileges>
87 <https://github.com/kubernetes-incubator/kubespary/blob/master/docs/dns-stack.md#limitations>
88 <https://github.com/kubernetes-incubator/kubespary/blob/master/docs/vault.md>
89 <https://github.com/kubernetes/helm>
90 <https://github.com/kubernetes/charts>
91 <https://github.com/coreos/container-linux-config-transpiler>

Dodatek B

Wykaz skryptów

B.1 repozytorium kubernetes-cluster

kubernetes-cluster (<https://github.com/nazarewk/kubernetes-cluster>) jest repozytorium Git, które zawiera kompletny kod źródłowy wykorzystywany w tej pracy inżynierskiej.

Kod można znaleźć i uruchomić w sieci uczelnianej z katalogu /pub/Linux
↔ /CoreOS/zetis/kubernetes/kubernetes-cluster/ na maszynie ldap.

bin/pull

Pobiera aktualną wersję repozytorium wraz z najnowszą wersją zależności:

```
#!/bin/sh
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>
# Purpose: git pull wrapper including submodule updates

git pull || (git fetch --all && git reset --hard origin)
git submodule update --init --remote --recursive
chmod +x ${0%/*}/*
```

bin/vars

Zawiera wspólne zmienne środowiskowe wykorzystywane w reszcie skryptów oraz linii komend:

```
#!/bin/sh
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>
# Purpose: Holds common variables used in all other scripts

if [ -z "$KC_CONFIGURED" ]; then
    KC_CONFIGURED=1
```

```

BIN="$(realpath ${0%/*})"
PATH="$BIN:$PATH"
PDIR="${BIN%/*}"
VENV="$PDIR/.env"
KUBESPRAY="$PDIR/kubespray"
INVENTORY="$KUBESPRAY/my_inventory"
CONFIG_FILE="$INVENTORY/inventory.cfg"

export KUBECONFIG="$KUBESPRAY/artifacts/admin.conf"
export ANSIBLE_CONFIG="$PDIR/ansible.cfg"
fi

```

bin/ensure-virtualenv

Konfiguruje środowisko Pythona, włącznie z próbą instalacji brakującego virtualenv przez apt:

```

#!/bin/sh
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>
# Purpose: installs python virtualenv

. ${0%/*}/vars
python="$(which python)"
activate="$VENV/bin/activate"

find_virtualenv() {
    which virtualenv 2> /dev/null || which virtualenv
}

install_virtualenv () {
    # Installs pip and/or virtualenv

    local pip=$(which pip)
    local apt=$(which apt)
    if [ ! -z "$pip" ]; then
        sudo $pip install virtualenv
        return 0
    elif [ ! -z "$apt" ]; then
        sudo $apt install virtualenv
        return 0
    else
        echo 'Virtualenv, pip and apt-get are both missing, cannot proceed'
        exit 1
    fi
}

create_env () {
    local cmd=$(find_virtualenv)
    [ -z "$cmd" ] && install_virtualenv && cmd=$(find_virtualenv)

    $cmd -p $python $VENV

```

```

    . $activate

    pip install -U pip setuptools
}

[ -z "$python" ] && echo 'python is missing' && exit 1
[ -e "$activate" ] && . $activate || create_env
pip install -U -r $PDIR/requirements.txt

```

bin/ensure-configuration

Generuje brakujące pliki konfiguracyjne SSH, inventory.cfg i group_vars

↪ /all.yml nie nadpisując istniejących:

```

#!/bin/sh
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>
# Purpose: configures cluster (SSH, inventory, group_vars)

. ${0%*/}/vars
SSH_CFG_DST="$HOME/.ssh/config"
SSH_CFG="$PDIR/zetis/.ssh/kubernetes.conf"

is_ssh_config_missing () {
    cat << EOF | python
import sys
expected = open('$SSH_CFG', 'rb').read()
existing = open('$SSH_CFG_DST', 'rb').read()
sys.exit(expected in existing)
EOF
    return $?
}

is_ssh_config_missing && cat $SSH_CFG >> $SSH_CFG_DST

file="$INVENTORY/inventory.cfg"
[ -f "$file" ] || cat << EOF > "$file"
[all]
;s3 ip=10.146.225.3
s4 ip=10.146.225.4
s5 ip=10.146.225.5
s6 ip=10.146.225.6
;s7 ip=10.146.225.7
;s8 ip=10.146.225.8
;s9 ip=10.146.225.9
;sa ip=10.146.225.10
;sb ip=10.146.225.11
;sc ip=10.146.225.12

[kube-master]
s4

```

```

[kube-node]
s5
s6

[etcd]
s4

[k8s-cluster:children]
kube-node
kube-master
EOF

file="$INVENTORY/group_vars/all.yml"
[ -f "$file" ] || cat << EOF > "$file"
cluster_name: zetis-kubernetes
bootstrap_os: coreos
kube_basic_auth: true
kubeconfig_localhost: true
kubectl_localhost: true
download_run_once: true
cert_management: "{ { 'vault' if groups.get('vault', None) else 'script'
    ↪ } }"
helm_enabled: true
helm_deployment_type: docker
kube_script_dir: /opt/bin/kubernetes-scripts
EOF

```

bin/render-coreos

Generuje konfigurację Ignition (JSON) na podstawie Container Linux Config (YAML):

```

#!/bin/sh
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>
# Purpose: renders CoreOS Ignition (incl. retrieving binary)

. ${0%/*}/vars
ct_version=${1:-0.6.1}
boot="$PDIR/zetis/WWW/boot"
url='https://github.com/coreos/container-linux-config-transpiler'
url="$url/releases/download"
url="$url/v${ct_version}/ct-v${ct_version}-x86_64-unknown-linux-gnu"

wget -nc $url -O $BIN/ct
chmod +x $BIN/ct
$BIN/ct -pretty \
    -in-file "$boot/coreos.yml" \
    -out-file "$boot/coreos.ign"

```

bin/setup-cluster

Właściwy skrypt konfiguruje klaster na działających maszynach CoreOS:

```
#!/bin/sh
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>
# Purpose: sets up the cluster

. ${0%/*}/vars
. $BIN/ensure-configuration
. $BIN/activate

(
  cd $KUBESPRAY;
  ansible-playbook -i $INVENTORY/inventory.cfg cluster.yml -b -v $@
)

chmod -R 700 $KUBESPRAY/{artifacts/admin.conf,credentials}

cat << EOF
Login credentials:
  user: kube
  password: $(cat $PDIR/credentials/kube_user)
EOF
```

bin/setup-cluster-full

Skrót do pobierania najnowszej wersji, a następnie uruchamiania klastra:

```
#!/bin/sh
. ${0%/*}/vars

$BIN/pull
$BIN/setup-cluster $@
```

bin/setup-cluster-upgrade

Skrypt analogiczny do setup-cluster, ale wywołujący upgrade-cluster.yml zamiast cluster.yml:

```
#!/bin/sh
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>
# Purpose: runs upgrade-cluster.yml instead of cluster.yml

. ${0%/*}/vars
. $BIN/ensure-configuration
. $BIN/activate

cd $KUBESPRAY
```

```
ansible-playbook -i $INVENTORY/inventory.cfg upgrade-cluster.yml -b -v  
↪ $@
```

bin/kubect1

Skrót kubect1 z automatycznym dołączaniem konfiguracji kubespray :

```
#!/bin/sh  
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>  
# Purpose: kubect1 wrapper which uses generated configurations  
  
# Handle running kubect1 from PATH  
[ "${0%/*}" == "$0" ] \  
  && _bin=$(dirname $(which kubect1)) \  
  || _bin=${0%/*}  
  
. $_bin/vars  
  
local_bin="$PDIR/artifacts/kubect1"  
  
if [ -f $local_bin ]; then  
  chmod +x $local_bin  
  kubect1=$local_bin  
else  
  kubect1=/usr/bin/kubect1  
fi  
  
$kubect1 $@
```

bin/students

Skrypt zarządzający obiektami k8s użytkowników, czyli studentów:

```
#!/bin/sh  
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>  
# Purpose: executes operations on users (create, get, describe, delete)  
  
. ${0%/*}/vars  
  
kubect1_command () {  
  local users="$1"  
  local cmd="$2"  
  for name in $users; do  
    cat <<EOF | $BIN/kubect1 $cmd -f -  
apiVersion: v1  
kind: Namespace  
metadata:  
  name: ${name}  
  labels:  
    name: ${name}  
  
```

```

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ${name}
  namespace: ${name}
---

kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: ${name}-admin-binding
  namespace: ${name}
roleRef:
  kind: ClusterRole
  name: admin
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: ${name}
EOF
echo Tokens:
$BIN/student-tokens ${name}
done
}

case $1 in
  ""|list)
    $BIN/kubect1 get namespace
    ;;
  -h)
    cat << EOF
    Usage: $0 "<usernames>" create|get|describe|delete
      where <usernames> is a single-argument list of usernames
      You can also call it without arguments to list users (namespaces)
    EOF
    ;;
  *)
    kubect1_command "$@"
    ;;
esac

```

bin/student-tokens

Listuje przepustki konkretnego użytkownika:

```

#!/bin/sh
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>
# Purpose: lists kubernetes authentication tokens for specified user

```



```

. ${0%/*}/vars
name=$1
args="--namespace $name get serviceaccount $name"
args="$args -o jsonpath={.secrets[*].name}"
for token in ` $BIN/kubectl $args `; do
    args="get secrets $token --namespace $name"
    args="$args -o jsonpath={.data.token}"
    $BIN/kubectl $args | base64 --decode
    echo
done

```

bin/install-helm

Instaluje menadżer pakietów Helm:

```

#!/bin/sh
# Author: Krzysztof Nazarewski <nazarewk@gmail.com>
# Purpose: installs helm package manager into Kubernetes cluster

. ${0%/*}/vars

host=${1:-s4}

ssh () {
    command ssh $host sudo $@
}

cat <<EOF | ssh bash
export HELM_INSTALL_DIR=/opt/bin
url='https://raw.githubusercontent.com/kubernetes/helm/master/scripts/
↪ get'
[ -f /opt/bin/helm ] || curl $url | bash
EOF

cat <<EOF | ssh kubectl create -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: tiller

```

```

    namespace: kube-system
EOF

ssh helm version | grep Server: || ssh helm init --service-account
↪ tiller

```

zetis/.ssh/kubernetes.conf

Częściowy plik konfiguracyjny SSH do umieszczenia w ~/.ssh/config:

```

Host s?
    User admin

IdentityFile ~/.ssh/id_rsa
IdentitiesOnly yes

Host s?
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null

```

zetis/WWW/boot/coreos.ipxe

Skrypt iPXE uruchamiający maszynę z CoreOS:

```

#!/ipxe
set ftp http://ftp/pub/

set base-url ${ftp}/Linux/CoreOS/alpha
set ignition ${ftp}/Linux/CoreOS/zetis/kubernetes/boot/coreos.ign

set opts ${opts} coreos.autologin
set opts ${opts} coreos.first_boot=1 coreos.config.url=${ignition}
set opts ${opts} systemd.journald.max_level_console=debug
kernel ${base-url}/coreos_production_pxe.vmlinuz ${opts}
initrd ${base-url}/coreos_production_pxe_image.cpio.gz

boot

```

zetis/WWW/boot/coreos.yml

Plik konfiguracyjny Container Linux Config w formacie YAML, docelowo do przepuszczenia przez narzędzie ct. Wyjątkowo skróciłem ten skrypt, ze względu na długość kluczy SSH:

```

passwd:
  users:
    - name: admin
      groups: [sudo, docker]
      ssh_authorized_keys:

```

```
- ssh-rsa <klucz RSA> ato@volt.iem.pw.edu.pl
- ssh-rsa <klucz RSA> nazarewk
- ssh-rsa <klucz RSA> nazarewk@ldap.iem.pw.edu.pl
```

zetis/WWW/boot/coreos.ign

Z powyższego pliku wygenerowany zostaje (również skrócony) plik JSON:

```
{
  "ignition": {
    "config": {},
    "timeouts": {},
    "version": "2.1.0"
  },
  "networkd": {},
  "passwd": {
    "users": [
      {
        "groups": [
          "sudo",
          "docker"
        ],
        "name": "admin",
        "sshAuthorizedKeys": [
          "ssh-rsa <klucz RSA> nazarewk",
          "ssh-rsa <klucz RSA> nazarewk@ldap.iem.pw.edu.pl",
          "ssh-rsa <klucz RSA> ato@volt.iem.pw.edu.pl"
        ]
      }
    ]
  },
  "storage": {},
  "systemd": {}
}
```

B.2 repozytorium ipxe-boot

ipxe-boot (<https://github.com/nazarewk/ipxe-boot>) jest repozytorium kodu umożliwiające uruchomienie środowiska bezdyskowego poza uczelnią.

Repozytorium nie zostało wykorzystane w finalnej konfiguracji, więc nie będę opisywał jego zawartości.