



ТЕХНОЛОГИЧНО УЧИЛИЩЕ "ЕЛЕКТРОННИ СИСТЕМИ"  
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

# ДИПЛОМНА РАБОТА

по професия код 523050 „Техник на компютърни системи“  
специалност код 5230502 „Компютърни мрежи“

Тема: Реализиране и централно управление на среда с множество Kubernetes  
клъстери посредством Cluster API.

Дипломант:

*Михаил Георгиев Петров*

Дипломен ръководител:

*Александър Терзийски*

СОФИЯ

2025





## **ИЗПОЛЗВАНИ СЪКРАЩЕНИЯ**

ALB - Application Load Balancer  
API - Application Programming Interface  
AWS - Amazon Web Services  
CAPI - ClusterAPI  
CF - CloudFormation  
CI/CD - Continuous Integration and Continuous Deployment  
CLI - Command-Line Interface  
CRDs - Custom Resource Definitions  
CSI - Container Storage Interface  
DevOps - Development and Operations  
DevSecOps - Development, Security and Operations  
DNS - Domain Name System  
EC2 - Elastic Compute Cloud  
EKS - Elastic Kubernetes Service  
ELB - Elastic Block Storage  
IAM - Identity and Access Management  
IaaS - Infrastructure as a Service  
IaC - Infrastructure as Code  
NAT - Network Address Translation  
NLB - Network Load Balancer  
OIDC - OpenID Connect  
PaaS - Platform as a Service  
PV - Persistent Volume  
PVC - Persistent Volume Claim  
SaaS - Software as a Service  
STS - Security Token Service  
VM - Virtual Machine  
VPC - Virtual Private Cloud

## **УВОД**

С развитието на облачните технологии и навлизането на контейнеризацията в разработката и управлението на софтуер, нуждата от автоматизирано създаване, управление и мащабиране на тези приложения става все по-належаща. Използвайки Kubernetes - водеща платформа за оркестрация на контейнери, проблемът с автоматизацията се решава, но произлиза друг. Първоначалната му настройка и управлението на множество такива клъстери в различни среди може да бъде сложен и ресурсоемък процес. Тук на помощ идва Cluster API (CAPI) - проект, предлагащ унифициран декларативен подход за управление жизнения цикъл на Kubernetes клъстери в различни инфраструктурни среди.

Настоящата дипломна работа разглежда именно автоматизирането и централното управление на Kubernetes клъстери посредством CAPI, с фокус върху неговата интеграция в AWS. Демонстрира се създаването на управляващ клъстер, чрез който се разгръщат допълнителни клъстери с работни приложения. Освен това, дипломната работа разглежда и управлението на ресурсите, като се показват функционалности като мащабиране на работните машини, модифициране на работещи конфигурации и изтриване на клъстери чрез CAPI.

# ПЪРВА ГЛАВА

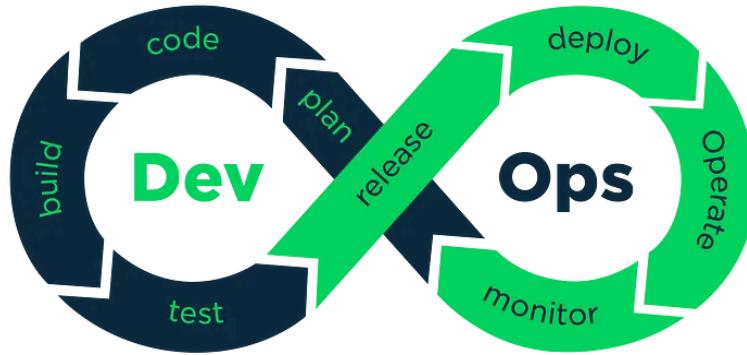
*Използвани принципи, технологии и практики и проучване на съществуващи решения за менажиране и управление на Kubernetes кълстери.*

## 1.1. DevOps

### 1.1.1. DevOps екипи

DevOps [1] (Разработка и Операции) (Фиг. 1.1) е популярна методология при разработването на софтуер, обединяваща екипите за разработка и операции, гарантирайки бързо и ефективно предоставяне на продукт. Често срещано е екипът “операции” да отговаря за управлението на инфраструктурата, наблюдение на производителността и осигуряване стабилността на приложението, докато разработчиците се фокусират върху кода и функционалностите. При този подход софтуерните инженери не само разработват и поддържат приложениета, но също така се фокусират върху специфични цели, като автоматизация на интеграцията на кода (CI) и непрекъснатото внедряване на нови версии (CD), което ускорява процесите на разработка и намалява времето за реакция при промени или проблеми.

Темата на дипломната работа, упомената по-горе, е пряко обвързана със сферата на DevOps, тъй като тя разглежда автоматизацията и управлението на сложни инфраструктури, напр. Кубернетес кълстери, както и внедряването на приложения чрез модерни инструменти и практики.



Фиг. 1.1 DevOps жизнен цикъл

### 1.1.2. DevOps методологии

Основният принцип в DevOps е кратки интеграции и непрекъснато усъвършенстване, което пряко съответства с "Agile" методологията. Освен това се използват и подходи като "Lean", който се фокусира върху разпределянето на работата по какъвто и да е начин, стига да добавя стойност за клиента възможно най-бързо и устойчиво, както и подходът Infrastructure as Code (IaC). Инфраструктурата като код е принцип, който позволява автоматизация на инфраструктурата чрез инструменти като Ansible и Terraform, което улеснява управлението и намалява риска от човешки грешки. Тези методологии се комбинират, за да създадат интегриран процес, който ускорява разработката, внедряването и поддръжката на софтуерни приложения, като същевременно осигурява стабилност, ефективност и високо качество при имплементацията им.

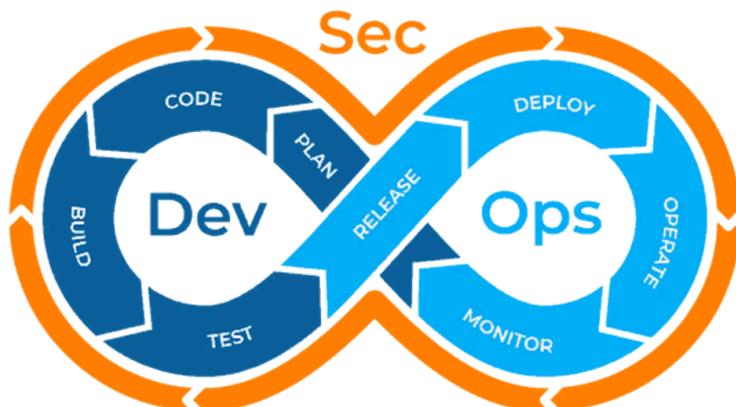
### 1.1.3. Технологии и инструменти

За подпомагане работата на софтуерните инженери се използват и други разнообразни технологии и инструменти, като Docker за контейнеризация, Kubernetes и Helm за оркестрация, Prometheus и Grafana за мониторинг, Jenkins и GitLab CI/CD за автоматизация, както и HashiCorp Vault за управление на сигурността. Това са само малка

част от технологиите, налични в DevOps сферата, където съществува широк набор от други инструменти, създадени да улеснят и оптимизират работата на разработчиците и операционните екипи.

#### 1.1.4. Практики за сигурност (DevSecOps)

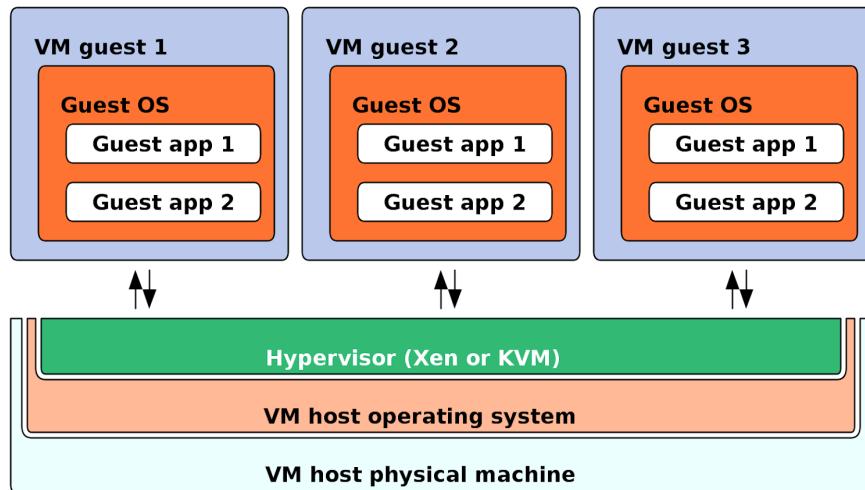
В DevOps съществуват и специализирани практики за сигурност, които се внедряват към основния подход, формирайки DevSecOps - "Разработка, Сигурност и Операции". Те се състоят от начини за интегриране на сигурността в ранните етапи на разработката, автоматизирано тестване за уязвимости чрез статично и динамично тестване на приложения и контейнери, както и осигуряване на контейнерна сигурност чрез сканиране на мрежовите политики. Така се осигурява защита през целия жизнен цикъл, като се намаляват рисковете и се подобрява качеството на софтуера. Както се забелязва от Фиг. 1.1, жизненият цикъл на DevSecOps много наподобява този на DevOps (Фиг. 1.2), но включва допълнителен фокус върху интегрирането на сигурността във всяка стъпка от процеса. Това гарантира, че сигурността е неразделна част от автоматизацията, непрекъснатото внедряване (CI/CD) и управлението на инфраструктурата.



Фиг. 1.2 DevSecOps жизнен цикъл

## 1.2. Виртуализация

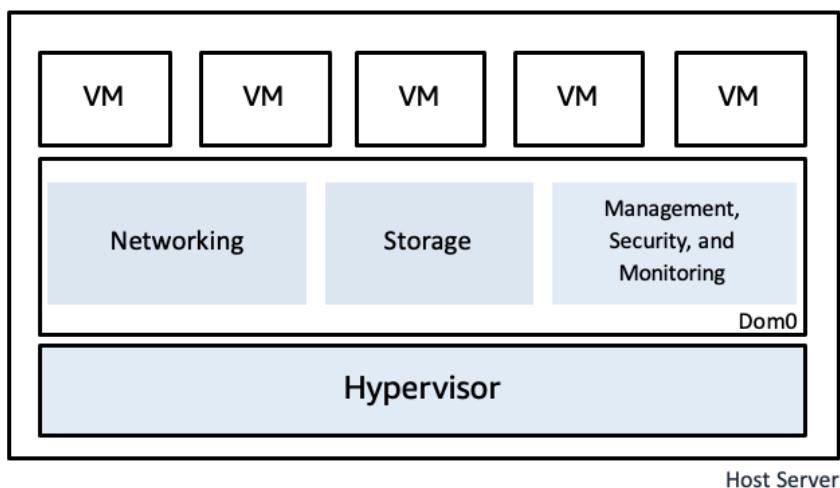
Ключово за предстоящото понятие контейнеризация е процесът на виртуализация [2]. Най-общо виртуализацията се отнася до използването на компютърни ресурси за симулиране на реалните хардуер, операционни системи, платформи и машини, което прави възможно стартирането на множество операционни системи и приложения на една хардуерна машина. Това позволява и ефикасното използване на наличните ресурси. Съществуват различни видове виртуализация, напр.: хардуерна, софтуерна, ресурсна и мрежова. От изброените най-често използваната е хардуерната виртуализация под формата на виртуални машини (VM) (Фиг. 1.3), които имат поведението на компютър с отделна операционна система (от гледна точка на виртуалната машина, тя е реален, физически компютър).



Фиг. 1.3 Схема на виртуална машина върху тип 2 “Hypervisor”

За да се различават истинската машина и симулираната, се използват термините “host” (хост/домакин) за физическата и “guest” (гост) - за виртуалната. Както се забелязва на Фиг. 1.3, освен операционните системи и приложенията върху машините, има и част - “Hypervisor” (хипервайзор). Това въщност е софтуерът за

виртуализация на виртуалната машина, който обработва виртуалния хардуер, включително процесорите, паметта, твърдия диск, мрежовите интерфейси и други устройства. Виртуалните хардуерни устройства предоставят информация на Hypervisor-а за действителния хардуер на физическата машина. Отделно хипервайзорът има два вида (типа): “Bare metal Hypervisor” (роден) (Фиг. 1.4), който взаимодейства директно с хардуера на машината и “Hosted Hypervisor” (хостван), който взаимодейства с хардуера на хост машината чрез операционната ѝ система. Тези два вида са съответно тип 1 и тип 2.



Фиг. 1.4 Схема на виртуална машина върху тип 1 “Hypervisor”

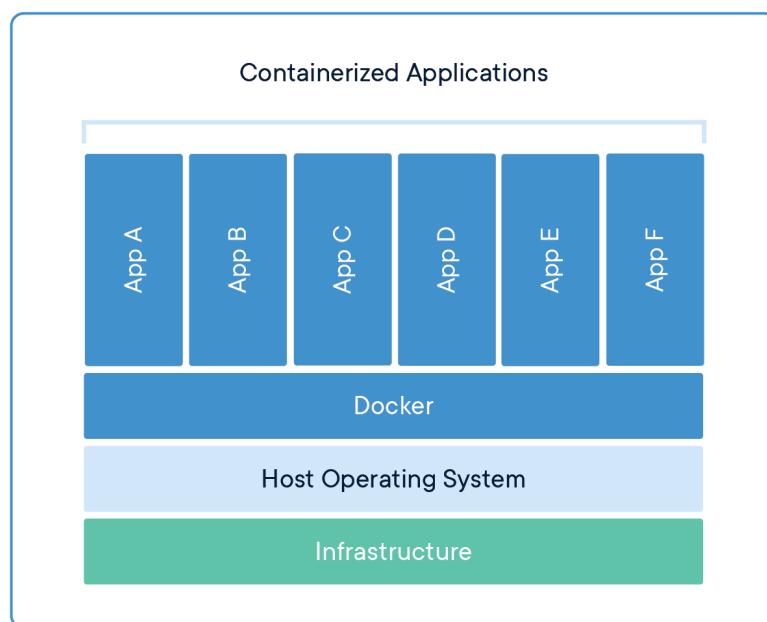
### 1.3. Контейнеризация

Контейнеризацията [3] е процес на внедряване на софтуер, който обединява кода на приложението с всички файлове и библиотеки, от които се нуждае, за да работи във всяка инфраструктура. Традиционно, за се стартира което и да е приложение на вашия компютър, трябва да инсталирате версията, която съответства на операционната система на вашето устройство. Например, трябва да инсталирате версията на Windows на софтуерен пакет на машина с Windows. С контейнеризацията обаче могат да се

създават софтуерни пакети, с други думи - контейнери, които работят на всички видове устройства и операционни системи. Контейнерите имат своите предимства пред виртуализацията и виртуалните машини:

- Лекота и мащабируемост:

Вместо виртуални машини с отделна операционна система, контейнерите използват обща ОС на хоста, което ги прави много леки и бързи. Те са леки софтуерни компоненти, които работят ефективно и елиминират излишния софтуерен слой (хипервайзора) (Фиг. 1.5), като позволяват приложенията да работят "по-близо" до хардуера. Например, една машина може да стартира контейнеризирано приложение по-бързо от виртуална машина, защото не е необходимо да зарежда операционна система, следователно могат лесно да добавят множество контейнери за различни приложения на една машина. Контейнерният клъстър (набор от контейнери) използва изчислителни ресурси от една и съща споделена операционна система, но един контейнер не пречи на работата на други контейнери.



Фиг. 1.5 Схема на контейнеризирани приложения

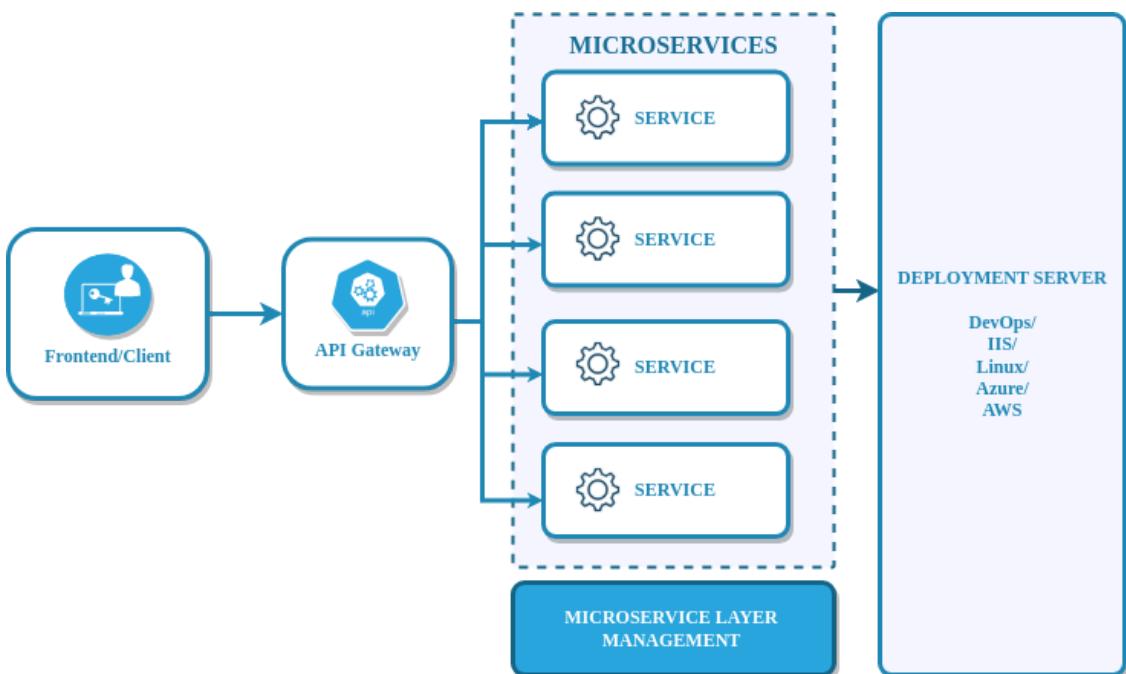
- Преносимост:

Контейнеризацията се използва, за да се разположат приложения в множество среди, без да пренаписват програмния код. Създават приложението веднъж и го внедряват на голям набор от операционни системи. Например, могат да се изпълняват еднакви контейнери на операционни системи като Linux и Windows. Софтуерните разработчици също могат да надграждат стария код на приложениета до съвременни версии.

- Тolerантност към грешки:

Софтуерните разработчици използват контейнери за изграждане на устойчиви на грешки приложения. Те използват групи от контейнери, за да изпълняват различни т.нар. микросървиси / микроуслуги в облака (Фиг. 1.6).

Тъй като контейнеризираните микроуслуги работят в изолирани потребителски пространства, един повреден контейнер не засяга другите контейнери, което повишава устойчивостта и достъпността на приложението.



Фиг. 1.6 Примерна инфраструктура на микросървисно приложение

- Гъвкавост:

Контейнеризираните приложения работят в отделни изчислителни среди, независими една от друга, поради което при откриването и отстраняването на проблеми, както и при промени на кода на приложението, това може да се случи без да засягат операционната система, хардуера или други приложни услуги. С помощта на контейнерите, издаването на нови версии на софтуера и актуализацията му могат да настъпват по-бързо.

Чрез обединяване на кода на приложението, конфигурационните файлове, библиотеките на операционната система (OS) и всички зависимости в едно, контейнерите помагат за решаването на често срещан проблем в софтуерната разработка: кодът, разработен в една среда, често проявява бъгове и грешки, когато бъде пренесен в друга среда. Например, код разработен в Linux среда и след това да го прехвърлен на виртуална машина (VM) или върху компютър с Windows, кодът вече не работи, както се очаква. За разлика от това, контейнерите са независими от носещата инфраструктура и осигуряват последователни среди за разработка, улеснявайки работата на софтуерните разработчици.

## 1.4. Docker

### 1.4.1. Предистория

Docker [4] (Фиг. 1.7) започва своята история през 2010 г., когато компанията "dotCloud Inc." е основана от Камел Фунади, Соломон Хайкс и Себастиен Пал. Първоначално dotCloud предлага платформа като услуга (PaaS), но през 2013 г. проектът им Docker привлича огромен интерес от разработчици и индустрията. Това води до преименуването на компанията на "Docker Inc." и преориентиране изцяло към контейнеризация и фокусиране върху Docker. През март 2013 г. Docker става open-source (софтуер с отворен код), което

значително ускорява приемането и развитието на технологията. В първите години от създаването на Docker като двигател за изпълнение на контейнери са използвани "LXC" (Linux Containers), но с версия 0.9 преминава към "libcontainer" - собствен компонент, написан на Go, който подобрява производителността и гъвкавостта на технологията. През 2014 г. Docker започва активно да се интегрира от големи облачни доставчици като Amazon Web Services (AWS), Microsoft Azure и Google Cloud, което допълнително укрепва позициите му като стандарт за контейнеризация в индустрията. Това позволява лесно внедряване на контейнеризирани приложения в облачни среди, което допринася за популярността на технологията.



Фиг. 1.7 Docker лого

#### 1.4.2. Docker като услуга

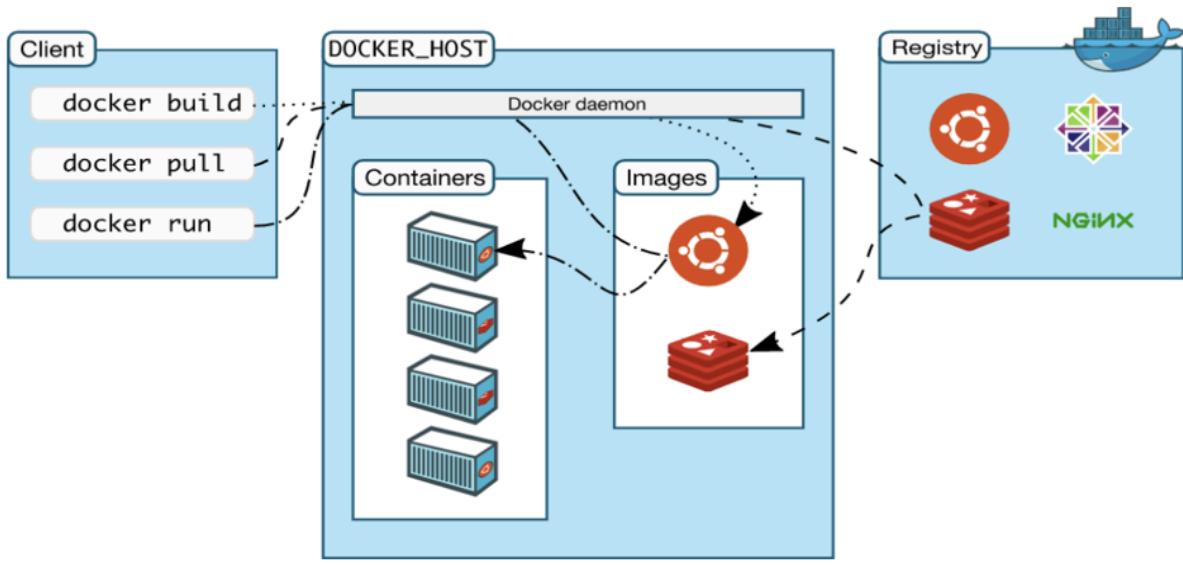
Docker е отворена платформа за разработка, доставка и изпълнение на контейнери. Позволява изолиране на приложението от инфраструктурата, което ускорява доставката на софтуер. С Docker може да се управлява инфраструктурата по същия начин, по който се управляват и приложението, което намалява времето между писането на код и внедряването му. Docker използва контейнеризация, за да даде възможност за пакетиране и изпълнение на приложението в изолирана среда, наречена контейнер. Те са леки, съдържат всичко необходимо за изпълнение на приложението и могат да работят едновременно на един хост. Това гарантира, че приложението ще функционира еднакво, независимо от средата.

Docker предоставя инструменти за управление на жизнения цикъл на контейнерите:

- Разработка - използване на контейнери за разработка на приложения и техните компоненти.
- Доставка - контейнерът служи като единица за разпространение и тестване.
- Внедряване - приложението се внедряват в производствена среда като контейнери или оркестрирани услуги, независимо дали средата е локална, облачна или хибридна.

Освен това, Docker позволява бърза и последователна доставка на приложението, чрез контейнерите, които улесняват CI/CD процесите. Така приложението могат да се разработват, тестват и внедряват в единен поток, което съкраща времето за корекции и промени. Също така, благодарение на Docker, е възможно гъвкаво внедряване и скалиране, тъй като контейнерите могат да работят във всяка среда, независимо дали това са лаптопи, физически или виртуални сървъри/машини, облака или хибридна среда. Това позволява динамично управление и скалиране на ресурсите в реално време.

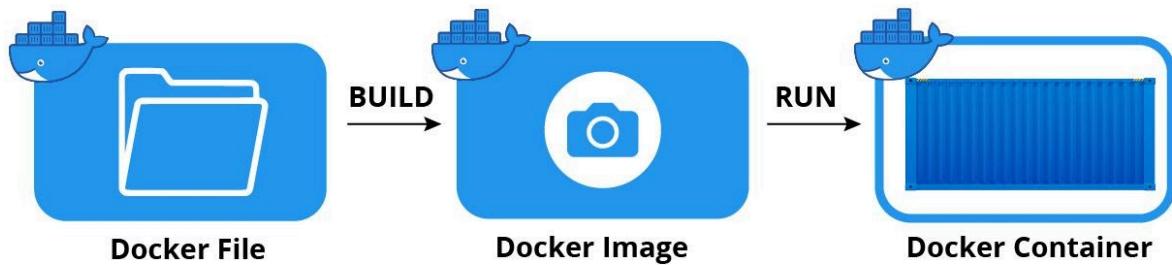
Инфраструктурата в Docker следва модела клиент-сървър, където клиентът говори с "docker daemon", посредством конзолни команди, който от своя страна управлява контейнерите при създаването, работата и дистрибуцията им. На Фиг. 1.8 ясно е демонстрирана връзката между клиента, сървъра и регистъра, на който регистър всъщност се намират т.нар. "имиджи"/"образи" на контейнерите.



Фиг. 1.8 Docker архитектура

#### 1.4.3. Docker images

В Docker за създаването на контейнерите е необходим скелет, по който те да се изграждат. Там на помощ идват docker images [5] (имиджи/образи). Образите представляват шаблони, които съхраняват конкретно състояние на един контейнер (Фиг. 1.9). Често, един образ е базиран на друг, като към него са добавени допълнителни персонализации. Например, шаблон може да бъде базиран на образа на Ubuntu, но да включва инсталация на уеб сървъра Apache, приложението, което ще се изпълнява, както и конфигурационни детайли, необходими за функционирането на приложението.



Фиг. 1.9 Стъпки за създаване на Docker контейнер

#### 1.4.4. Docker Hub

“Docker Hub” е публичен регистър, който съхранява Docker образи и е достъпен за всеки. По подразбиране Docker използва Docker Hub като основен източник за изтегляне на образи, необходими за създаване на контейнери. Въпреки това, ако се налага по-голям контрол или сигурност, е възможно да се създаде собствен частен регистър, който да съхранява специфични образи. При изпълнение на команди като “docker pull” или “docker run”, Docker автоматично изтегля необходимите образи от конфигурирания регистър. Обратно, команда “docker push” се използва за качване на собствени образи в регистъра, било то в Docker Hub, или в частен регистър.

#### 1.4.5. Dockerfile

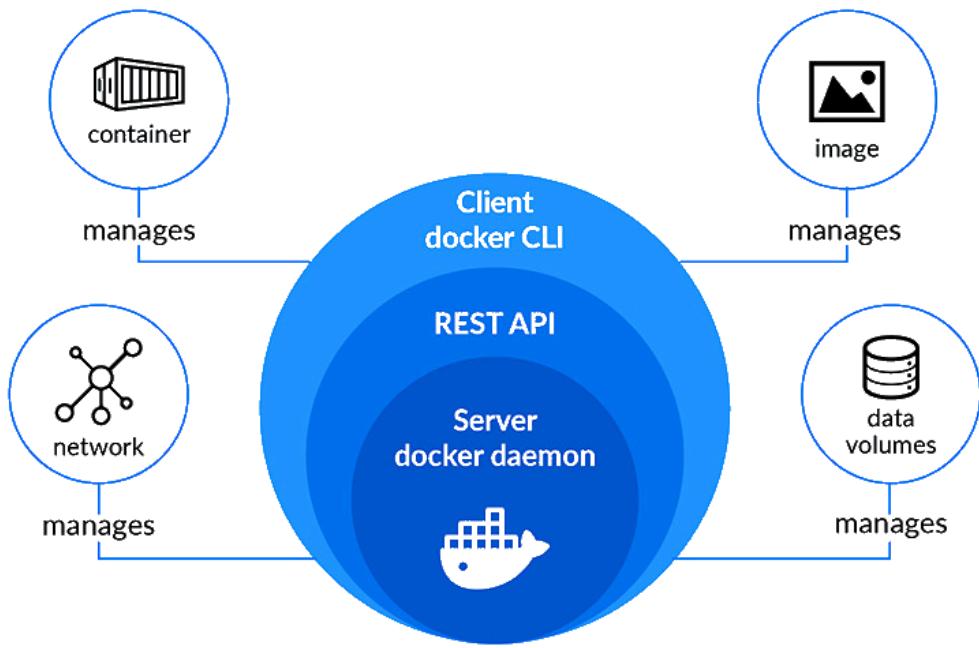
Образите могат да бъдат създавани ръчно или да се използват такива, публикувани от други потребители в регистри (registry), напр. Docker Hub. За създаването на собствен образ се използва т.нар. “Dockerfile” (докер файл) (Фиг. 1.9), който предоставя опростен синтаксис за дефиниране на стъпките, необходими за изграждането на образа и изпълнението му. Всяка инструкция в Dockerfile създава слой в образа. При промяна на Dockerfile и повторно изграждане се реконструират само тези слоеве, които са променени. Тази архитектура е една от причините образите да са леки, малки и бързи в сравнение с други технологии за виртуализация.

#### 1.4.6. Docker daemon

Всички тези обекти трябва да се изпълнят и менажират. За целта е създаден Docker daemon [6] (“dockerd”), демонстриран на Фиг. 1.8. Това е основният процес в Docker, който работи във фонов режим и управлява Docker обектите, като контейнери, образи, мрежи и

съхранението на данни. Той е отговорен за изпълнението на всички команди, подадени чрез "Docker CLI" (Command-Line Interface, интерфейс за управление на контейнери чрез командния ред) или "Docker API" (позволява взаимодействие с Docker) и осигурява основната функционалност на Docker платформата. Този "daemon" може да комуникира и с други подобни процеси, за да менажират Docker ресурсите. В обобщение, основните характеристики на Docker daemon са следните (Фиг. 1.10):

- Управление на Docker ресурси - Docker Daemon отговаря за създаването, стартирането, спирането и изтриването на контейнери. Той също така управлява съхранението на образи, създаването на мрежи и управление на данните.
- Комуникация чрез Docker API - Docker Daemon приема инструкции чрез REST API (Docker API), което позволява на разработчиците и приложенията да взаимодействват с Docker програмно.
- Оркестрация - в конфигурации с множество хостове, Docker Daemon може да работи съвместно с други процеси в кълстеририани среди, използвайки технологии като Docker Swarm или Kubernetes, с цел управление и оркестрация на контейнери.
- Изпълнение на команди - Docker Daemon приема и изпълнява инструкции, изпратени от Docker CLI, като "docker run" за стартиране на контейнер или "docker build" за създаване на образ.



*Фиг. 1.10 Docker Daemon свързаност*

#### 1.4.7. Docker engine

Docker Engine [7] е общото наименование за платформата, която предоставя всички основни компоненти и функционалности на Docker.

Той включва следните компоненти:

- Docker daemon ("dockerd") - процесът, който управлява Docker обектите.
- Docker CLI - конзолният инструмент за взаимодействие с Docker daemon.
- Docker API - интерфейс, който позволява автоматизация и взаимодействие с Docker от външни приложения.

### 1.5. Оркестрация

Оркестрацията на контейнери [8] е процесът на автоматизация на управлението, мрежовата свързаност и менажирането на контейнери, което позволява внедряване на приложения в по-голям

мащаб. Както вече заем, контейнеризацията обединява кода на приложението с всички необходими библиотеки и файлове, с помощта на платформи като Docker, за да може то да работи на всякаква инфраструктура. В архитектури, базирани на микроуслуги, има вероятност броят на контейнерите да достигне стотици или хиляди, което прави управлението им изключително сложно, ако трябва да бъде постигнато индивидуално. В тези ситуации, с помощта на инструментите за контейнерна оркестрация, този процес се опростява от автоматизация на жизнения цикъл на контейнерите – до създаването и разпределението, както и внедряването и премахването им.

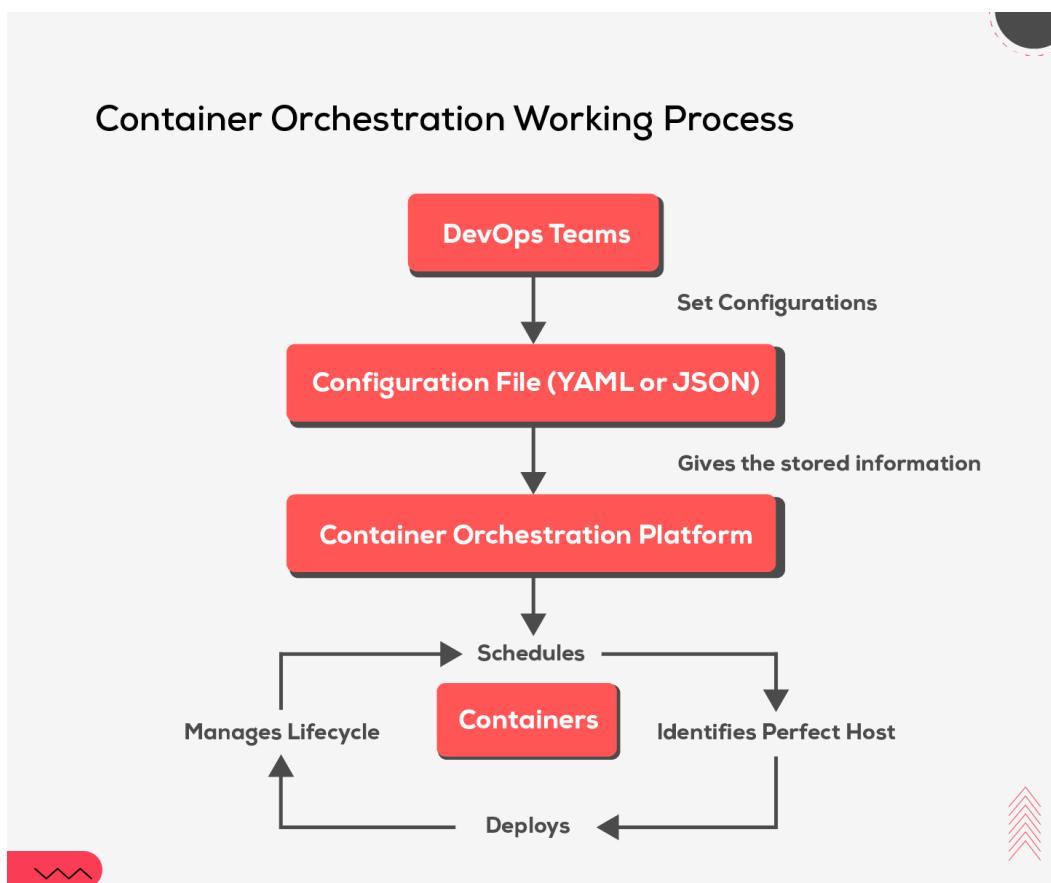
Преди създаването на платформите за оркестрация, управлението на контейнери е изисквало сложни скриптове, което водело до проблеми с мащабируемостта и поддръжката. Оркестрацията автоматизира тези процеси, премахвайки нуждата от ръчно управление и усложненията, свързани с него.

Главните приложенията на контейнерната оркестрация включват:

- Управление и скалиране на контейнери на множество сървъри.
- Едновременно изпълнение на различни контейнеризирани приложения.
- Стартиране на различни версии на приложения (например за тестова и продукционна среда) чрез CI/CD процеси.
- Поддържане на непрекъсната работа на приложенията чрез репликиране и повторно вдигане на контейнери при сървърни сривове.
- Разпределение на приложения в различни географски региони.
- Оптимално използване на сървърни ресурси, с цел намаляване на разходите.

- Поддръжка на големи приложения, базирани на микроуслуги, включващи хиляди контейнери.

При всички тези случаи оркестрацията на контейнери улеснява управлението на сложни приложения и позволява на организациите да използват мащабируемостта и предимствата на контейнеризацията, без да налагат допълнителни разходи за поддръжка.. На Фиг. 1.11 е демонстриран жизненият цикъл на оркестрацията. DevOps екипите определят начина, по който контейнерите се управляват чрез конфигурационни файлове, обикновено във формат “YAML” или “JSON”. Тези конфигурации се подават към платформите за контейнерна оркестрация, които планират разположението на контейнерите, избират подходящите хостове за тяхното изпълнение, внедряват ги и управляват жизнения им цикъл, автоматизират процеса по управление, свързване, деплой и скалиране на контейнерите.



Фиг. 1.11 Жизнен процес на контейнерната оркестрация

## 1.6. Kubernetes

### 1.6.1. Kubernetes като услуга

“Kubernetes” [9] (Фиг. 1.12) е предпочитаната система за оркестрация на контейнери в световен мащаб. Представлява отворена платформа за управление на контейнеризирани приложения и услуги, която улеснява декларативната конфигурация и автоматизацията. Платформата има богата екосистема, която се развива бързо, а услугите и инструментите за Kubernetes са широко достъпни. Името „Kubernetes“ произлиза от гръцки и означава „рулеви“ или „пилот“. Проектът е създаден от “Google” и става с отворен код през 2014 г. Контейнерите са отличен начин за опаковане и изпълнение на приложения, но в продукционна среда управлението на контейнерите и безпогрешната им работа може да бъде предизвикателство. Kubernetes автоматизира тези процеси, предоставяйки платформа за надеждно изпълнение на разпределени системи. Той се грижи за мащабирането, управлението на неизправности, моделите за внедряване и други.



Фиг. 1.12 Kubernetes лого

Основните функционалности на Kubernetes включват:

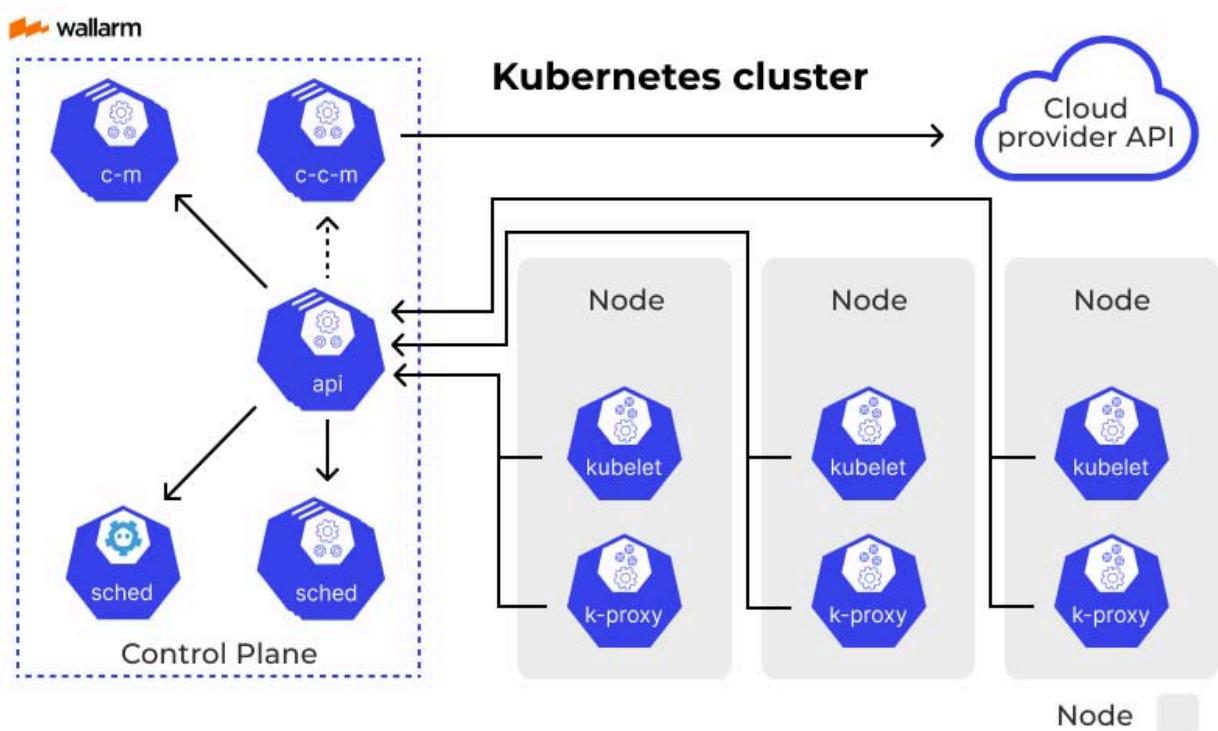
- Разкриване на ресурси и балансиране на натоварването - Kubernetes може да открие достъп до контейнер, използвайки DNS името или използвайки собствения му IP адрес. Ако трафикът към контейнер е голям, Kubernetes може да балансира натоварването и да разпредели мрежовия трафик, така че приложението върху контейнера да е стабилно.

- Оркестрация на хранилищата с данни - Kubernetes позволява автоматично монтиране на система за съхранение по избор, като например локални хранилища, доставчици на публичен облак и други.
- Автоматизирани внедрявания и отмени - Kubernetes поддържа декларативно управление на състоянието на контейнерите и автоматично синхронизира текущото състояние с желаното.
- Автоматично разпределение на ресурси - оптимизира използването на изчислителни ресурси като процесорна мощност (CPU) и оперативна памет (RAM), като поставя контейнери на най-подходящите хостове в клъстера, спрямо това колко се използват хостовете и колко изчислителна мощност е заделена.
- Самовъзстановяване - рестартира неуспешни контейнери, заменя проблемни, премахва неотговарящите и изчаква контейнерите да са готови преди да могат да се използват.
- Управление на поверителни данни - позволява съхранение и управление на поверителна информация (като пароли и токени), без да се налага преизграждане на контейнерните образи.
- Партидно управление - Kubernetes може да управлява партидни задачи, като автоматично заменя неуспешни контейнери.
- Машабиране - Kubernetes позволява както хоризонтално, така и вертикално машабиране на ресурсите. Хоризонталното увеличава или намалява броя на обвивките, които изпълняват дадено приложение, което спомага разпределенето на натоварването върху повече ресурси. Вертикалното, от друга страна, добавя или премахва изчислителни ресурси (CPU и RAM) на съществуващи обвивки, което подобрява тяхната производителност, без да променя броя им.
- Задаляне на адреси - има възможността да осигурява IPv4 и IPv6 адреси към ресурсите си.

- Разширяемост - позволява добавяне на нови функции към кълъстера, без промени в основния код.

### 1.6.2. Kubernetes контролен и работен слой (control&worker plane)

В Kubernetes платформата, комбинацията от всички компоненти образува един т.нар. "кълъстер" [10] (Фиг. 1.13). Този кълъстер се състои от работен (съвкупност от работни компоненти) и контролен слой.



Фиг. 1.13 Архитектура на Kubernetes кълъстер

Компонентите на контролния слой (Фиг.1.13 - горе вляво "Control-Plane") вземат глобални решения относно кълъстера (например планиране на задачи), както и за откриване и реакция на събития в кълъстера. Обикновено тези компоненти могат да се изпълняват на всяка машина в кълъстера. За улеснение обаче, при начална конфигурация, често се стартират всички компоненти на

контролния слой на една и съща машина, на която не се изпълняват потребителски контейнери.

Състои се от следните компоненти :

- kube-api-server - основният компонент на контролния слой, който предоставя достъп до Kubernetes API и служи като интерфейс за взаимодействие с кълстера. Може да мащабира кълстера хоризонтално, като добавя още инстанции/машини (nodes). Има възможността за стартиране на няколко API сървъра, които да балансират трафика помежду си.
- etcd - консистентна и високодостъпна база данни, тип key-value (ключ-стойност), която съхранява всички данни на кълстера, относно състоянието и ресурсите върху кълстера.
- kube-scheduler - компонент от контролния слой, който следи за новосъздадени обвивки ("Pods") без присвоена машина ("Nodes") и избира подходяща инстанция за изпълнение. Определя машината за изпълнение на контейнера спрямо: индивидуални и колективни изисквания за ресурси; хардуерни, софтуерни и политически ограничения; локалност на данните и времеви ограничения.
- kube-controller-manager - компонент, който комбинира всички различни контролери за управление на ресурси в кълстера. Някои от тези контролери които "kube-controller-manager" управлява са:
  - Node Controller - следи и контролира състоянието на машините.
  - Job Controller - следи за обекти тип "Job", които представляват еднократни задачи, след което създава обвивки ("Pods"), за да ги изпълнява.
  - EndpointSlice controller - свързва сервисите ("service") с обвивките ("Pods").

- cloud-controller-manager - компонент, който съдържа логика, специфична за облачните доставчици. Той позволява връзка между кълстерите с API на облачните доставчици и техните услуги. Също така разделя компонентите, които взаимодействат с облака, от тези, които работят само с кълстера. Cloud-controller-manager може да се управлява и да следи следните контролери например:

- Node controller - проверява дали възел е изтрит в облака след отпадане.
- Route controller - настройва маршрути в облачната инфраструктура.
- Service controller - управлява облачните разпределители на трафика.

Обобщено, контролният слой осигурява централно управление и координация на всички ресурси и операции в кълстера.

Работният слой/работните машини (възли) (т.нар. "nodes"), от друга страна, изпълняват контейнерите, поставяйки ги в "обвивки" ("подове"/"Pods"). Тези машини могат да бъдат виртуални или физически, в зависимост от кълстера. Всеки възел се управлява от контролния слой. Подобно на контролния слой, върху всяка работна машина също има контролери:

- kubelet - контролер, който гарантира, че контейнерите в обвивките са стартирани и работят правилно.
- kube-proxy - осигурява вътрешна и външна комуникация със сървиси ("Service"). Поддържа мрежови правила, които позволяват комуникация с обвивките както от вътрешни, така и от външни мрежи.
- Container runtime - основен компонент, който позволява на Kubernetes ефективно да изпълнява контейнери. Отговаря за изпълнението и жизнения цикъл на контейнерите. Kubernetes

поддържа "container runtimes" като "containerd", "CRI-O" и други, съвместими с Kubernetes CRI (Container Runtime Interface).

Тези контролери върху работните машини ("Nodes") осигуряват ключови функционалности за стартиране и управление на обвивки ("Pods"), мрежовите комуникации и изпълнението на контейнерите.

## 1.7. Kubernetes ресурси

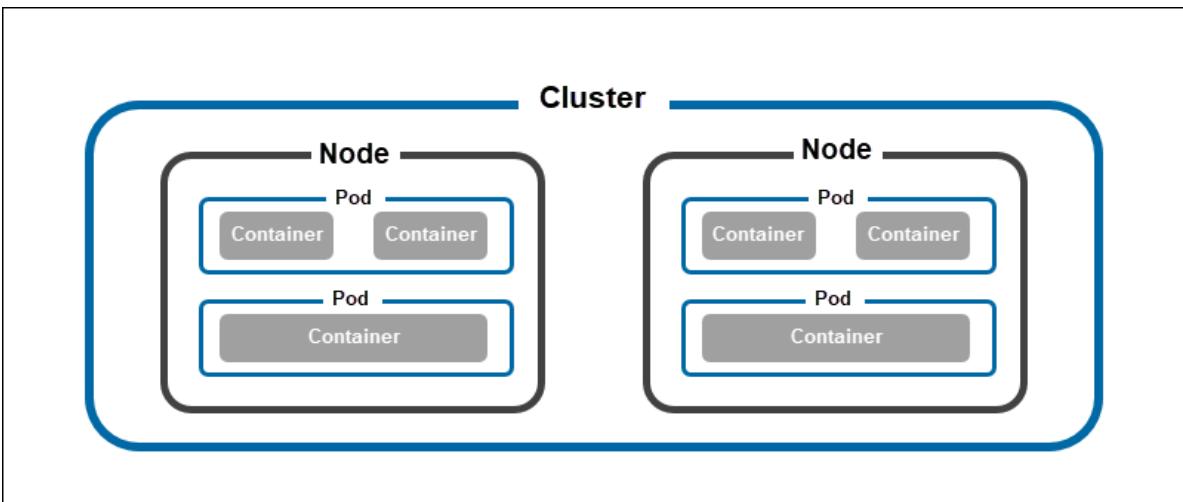
### 1.7.1. Pods (обвивка)

Обвивките са най-малките изпълними единици в Kubernetes [11]. Представляват група от един или повече контейнера, които споделят съхранение, мрежови ресурси и спецификация за начина на изпълнение на контейнерите. Съдържанието на обвивка винаги се изпълнява на едно и също място и в един и същи контекст, като поддържа съвместно разпределение и планиране. Може да се разглежда като „логически хост“ за приложения – съдържа един или повече приложни контейнери, които са свързани помежду си. В традиционните среди, приложениета, изпълнявани на една и съща физическа или виртуална машина, са аналогични на контейнерите, изпълнявани в една и съща обвивка в облачна среда.

Тези обвивки в Kubernetes се изпълняват главно по два начина (Фиг. 1.14):

- С един контейнер - това е най-често срещаният модел в Kubernetes, в който обвивката е само една около един контейнер, а Kubernetes управлява нея, вместо контейнерите директно.
- С множество контейнери, които работят заедно - обвивките могат да включват приложение, съставено от няколко съвместно разположени контейнера, които са тясно свързани и трябва да споделят ресурси. Това е рядко срещан сценарий обаче, който се

използва само когато контейнерите са тясно обвързани и зависят един от друг.



Фиг. 1.14. Архитектура на обвивки с един или множество контейнери

Подобно на всеки внедряем (deployable) ресурс в Kubernetes, обвивките се конфигурират с помощта на “YAML” формат конфигурационни файлове, наречени “манифести” (“manifests”).

### 1.7.2. ReplicaSet

“ReplicaSet” е ресурс в Kubernetes [12], чиято основна цел е да поддържа стабилен брой идентични обвивки по всяко едно време, като гарантира тяхната наличност и специфична бройка. ReplicaSet-а се дефинира с променливи, като т.нар. “selector”, който избира обвивките, които да присвои към дадения сет. Полето “реплики” оказва броя техни копия, които ресурсът трябва да поддържа, след което той ги създава или изтрива, за да постигне желания брой. Обикновено ReplicaSet се управлява автоматично от “Deployment”, който предлага допълнителни функции, като декларативни актуализации. Използването на ReplicaSet директно е препоръчително само при специфични случаи, например когато не се изискват актуализации или се налага персонализирано управление. Дори

самите Kubernetes препоръчват използването ексклузивно само на по-висшия ресурс “Deployment”.

### 1.7.3. Deployment

“Deployment” е Kubernetes обект [13], който управлява група от обвивки и служи за изпълнение на приложения, най-често такива, които не поддържат състояние. Той предоставя декларативен подход за обновяването им, както и на гореупоменатите ReplicaSets. С Deployment се описва желано състояние, а Deployment Controller постепенно синхронизира текущото състояние с желаното. Може да се сравни с термостат в стая. Температурата, до която искаме да стигнем, е т.нар. “желано състояние”, а сегашната температура е “текущото състояние”. Термостатът постепенно се грижи температурата да достигне нашето “желано състояние”. Deployment ресурсът също може да се използва за създаване на нови ReplicaSets, премахване на съществуващи Deployments или прехвърляне на ресурси от един Deployment към друг.

Основните функции на Deployment обекта включват:

- Декларитивни обновления - позволява промяна на състоянието на обвивките чрез актуализиране на техния параметър “PodTemplateSpec”.
- Контролирано внедряване - новите реплики се създават и синхронизират с текущите обвивки постепенно.
- Управление на ресурси - Deployment автоматично обработва създаването, мащабирането и премахването на обвивки и ReplicaSets.
- Възстановяване (Rollback) - връщане към по-ранна версия на Deployment, ако текущото състояние не е стабилно.

#### 1.7.4. StatefulSet

StatefulSet [14] е API обект за управление на приложения, които изискват съхранение на състояние (stateful applications). Той управлява група от обвивки и осигурява тяхната стабилна идентичност, което е полезно за приложения, нуждаещи се от постоянно хранилище или уникална мрежова идентичност. Една от най-важните характеристики на StatefulSet ресурса е именно, че при управлението на обвивките, той се грижи за мащабирането и внедряването, като гарантира уникалността на отделните обвивки и тяхната подреденост. За разлика от Deployment обекта, StatefulSet осигурява “залепена” идентичност за всяка обвивка, която се запазва дори при преместване, изтриване и повторно стартиране. Освен това създадени веднъж, те не са взаимнозаменяеми, тъй като всяка от тях има постоянен идентификатор. Този ресурс е най-често приложим за приложения с бази от данни, каквото присъства в тази дипломна работа, където връзката между контейнера и базата е критична.

#### 1.7.5. DaemonSet

DaemonSet е друг Kubernetes ресурс [15], който дефинира обвивки, осигуряващи локални за машините услуги. Те могат да бъдат основополагащи за работата на клъстера (например инструменти за управление на мрежата) или да са част от допълнителни компоненти. DaemonSet гарантира, че всички (или някои) машини в клъстера изпълняват копие на дадена обвивка. Характерното за този ресурс е, че при добавяне на нови машини в клъстера, автоматично създава обвивка върху тях, а при премахване на машини, тя автоматично се изтрива. Често се употребява за стартиране на различни видове ресурси за: управление на клъстерно хранилище на машините, събиране на логове от всяка машина, наблюдение на машините. DaemonSet е особено полезен за сценарии, където локалните услуги,

като тези събиращи на логове или мониторинг, трябва да работят на всяка машина. При сложни среди могат да се използват множество DaemonSet ресурси за оптимизация.

#### 1.7.6. Service (сървис)

Service (т.нар. сървис) в Kubernetes е механизъм за осигуряване на мрежови достъп до приложение [16], което работи върху една или повече обвивки в кълстера. Този обект предоставя единна външна точка за достъп до приложението, дори и то да е разположено на множество от тях.

Характеризира се с:

- Абстракция на мрежовите приложения - Service обединява група обвивки, които изпълняват дадено приложение, и дефинира как те да бъдат достъпни в мрежата. Допълнително, политиките на Service ресурса определят правилата за достъп до тях.
- Динамично управление на обвивките - те биват управлявани от Deployment, могат да се създават и унищожават динамично, което води до промени в техните IP адреси и брой във всеки даден момент. Service решава проблема с намирането и достъпа до тях, като предоставя стабилна точка за свързване, независимо от промените настъпили от Deployment-а.
- Поддръжка на мрежова стабилност - всяка обвивка по подразбиране получава уникален IP адрес. За да определи обвивките, към които иска да се свърже, Service използва "selector" (селектор) (Фиг. 1.15), който ги избира спрямо шаблона, по който са създадени.

Пример за употреба на този Kubernetes ресурс е бекенд на приложение с три реплики. Фронтендът на приложението няма значение коя точно от трите реплики на бекенда да използва. За целта

съществува Service-а, който предоставя единна точка, независимо от броя или IP адресите на обвивките, което премахва необходимостта клиентите сами да следят тези параметри.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

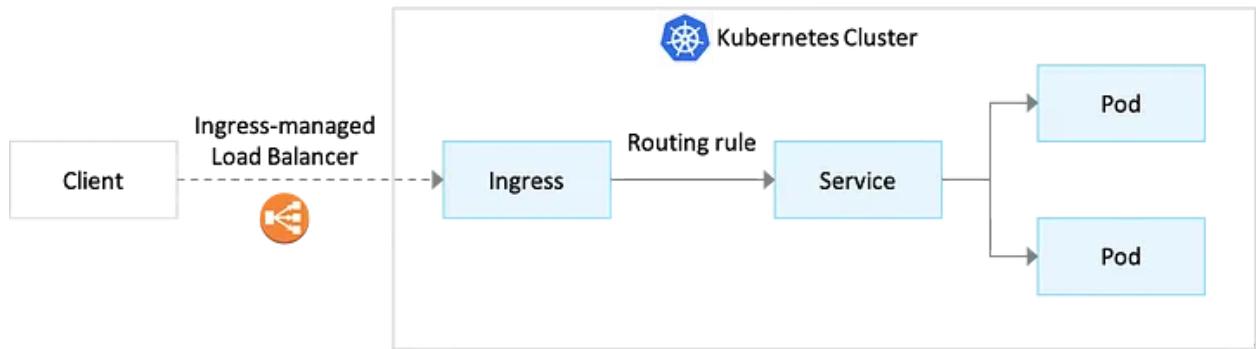
Фиг. 1.5 Примерен манифест на Service ресурса в YAML формат

Service ресурсът се конфигурира по подобен начин на всеки внедряем Kubernetes ресурс, а именно чрез манифест (конфигурационен файл) (Фиг. 1.15). Ако предположим, че имаме набор от обвивки, които слушат за трафик на “TCP” (Transmission Control Protocol) порт 9376, можем да създадем Service обект, чрез дадения манифест, който да пренасочва входящия трафик към съответния порт в обвивките, ако етикетите им съответстват на “app.kubernetes.io/name=MyApp”. Съответно, когато клиент изпрати заявка към IP адреса на Service-а на порт 80, тя ще бъде препратена към порт 9376 на обвивките, които отговарят на изискванията на селектора.

#### 1.7.7. Ingress (ингрес)

Ingress (т.нр. ингрес) предоставя HTTP и HTTPS достъп от външната мрежа към сървици вътре в клъстера (Фиг. 1.16).

Насочването на трафика се управлява чрез правила, дефинирани в ресурса Ingress (routing rule) [17].



Фиг. 1.16 Примерна схема на ингрес препращащ трафик към сървис

Ингрес използва ингрес контролер (Ingress Controller), който изпълнява конфигурациите на ингрес обекта. Обикновено това става чрез настройка на балансъор на натоварването (load balancer), както на Фиг. 1.16, където клиентът достъпва URL адрес, трафикът му преминава през балансъора, стига до ингреса, който, спрямо правилата за пренасочване, препраща трафика към сървиса, а той - към обвивките.

#### 1.7.8. Volumes (хранилища)

Хранилищата в Kubernetes предоставят начин за контейнерите в обвивките да достъпват и споделят данни чрез файловата система [18]. Има различни видове хранилища, които се използват за различни цели, като например:

- Споделяне на файлова система между два различни контейнера в една обвивка или между две различни обвивки (дори и те да работят на различни работни машини ("nodes")).
- Осигуряване на малко временно хранилище на обвивките.
- Трайно съхраняване на данни, така че те да останат достъпни, дори ако някоя обвивка се рестартира или бъде сменена.

- Споделянето на данни между различни локални процеси в рамките на контейнер, между различни контейнери, или между обвивки.
- Предоставяне на достъп само за четене до данни в различно изображение на контейнер.

Kubernetes хранилищата предоставят устойчивост на данните.

Усложнение, което възниква често, тъй като файловете на диска в контейнер са краткотрайни, създава проблеми при приложениета които разчитат данните да бъдат постоянни, с други думи - да не се губят. Когато контейнер се срине или бъде спрян, защото състоянието на контейнера не се запазва, всички файлове, които са създадени или модифицирани по време на жизнения цикъл на контейнера, се губят. По време на срив, kubelet рестартира контейнера в чисто състояние (без данни). Друг проблем възниква, когато множество контейнери работят в обвивката и трябва да споделят файлове. Това създава предизвикателство за настройката и получаването на достъп до споделена файлова система във всички контейнери. В тези случаи абстракцията на Kubernetes "хранилище" помага разрешаването проблемите.

#### 1.7.9. PV и PVC

Управлението на хранилищата е различен проблем от управлението на работните машини. Подсистемата PersistentVolume (постоянно хранилище) предоставя API за потребители и администратори, което да абстрагира подробности за това как се предоставя съхранение от това как се използва. Затова съществуват два нововъведени API ресурса: PersistentVolume и PersistentVolumeClaim:

- PersistentVolume (PV) (постоянно хранилище) е част от хранилището в кълстера, което е предоставено от

администратор, или динамично, с помощта на класове за съхранение [19]. Това е ресурс в клъстера, точно както работната машина (“node”) е ресурс на клъстера. На практика, PV са “дисковете” на обвивките, които осигуряват съхранение за техните данни. Те са хранилищни плъгини като Kubernetes Volumes, но имат жизнен цикъл, независим от всяка отделна обвивка, която използва PV.

- PersistentVolumeClaim (PVC) е заявка за хранилище от потребител. Подобно е на обвивка. Те консумират ресурси на работните машини, а PVC консумират PV ресурси. Обвивките могат да изискват определени нива на ресурси (CPU и памет). PVC могат да изискват специфичен размер и режими на достъп (напр. те могат да бъдат монтирани ReadWriteOnce, ReadOnlyMany, ReadWriteMany или ReadWriteOncePod)

Докато PersistentVolumeClaim позволява на потребителя да използва абстрактни ресурси за съхранение, обичайно потребителите се нуждаят от PersistentVolumes с различни свойства, като повече или по-малко производителност, за различни случаи. Клъстерните администратори трябва да могат да предлагат разнообразие от PersistentVolumes (постоянни хранилища), които се различават не само по размера и режимите на достъп, без да показват самата имплементация за това как тези хранилища са внедрени (вид абстракция). За тези нужди има ресурс StorageClass [20], който предоставя начин за администраторите да описват предлаганите от тях класове за съхранение. Различните класове могат да имат различни нива на качество на услугата или различни политики за архивиране, или с произволни политики, определени от администраторите на клъстера. За Kubernetes няма значение относно

това какво представляват класовете, предлагани от администраторите.

#### 1.7.10. CRDs

CRDs (Custom Resource Definitions или нестандартни, персонализирани ресурси) са разширения Kubernetes API, които не задължително са налице в инсталацията по подразбиране на Kubernetes. Те предоставят персонализиране на конкретна инсталация. Такива ресурси могат да се добавят или изтриват в работещ кълстър чрез динамична регистрация, а кълстър администраторите могат да актуализират персонализираните ресурси, независимо от самия кълстър. След като такъв ресурс бъде инсталиран, потребителите могат да осъществят достъп до него и неговите обекти, с помощта на kubectl, като при вградените Kubernetes ресурси като обвивки. По-този начин осигуряват гъвкавост за дефиниране и управление на специфични за приложениета ресурси, автоматизират процеси и интеграция с външни системи, без нуждата от модификация на основния Kubernetes код.

### 1.8. Облачни услуги

Терминът “облачна среда” [21] е често срещан в дипломната работа и обикновено се отнася за различни технологии, но най-често за облачни изчисления (cloud computing) или облачни хранилища на данни (cloud storage). Той изразява начините за предоставяне на компютърни ресурси, вариращи от даване на хардуерна инфраструктура под наем, до достъп до скъпи софтуерни приложения през интернет, при което потребителите плащат относително ниски месечни такси за ползване, въз основа на потреблението им. Всъщност облачните услуги се използват навсякъде и са особено полезни за компаниите, защото ги освобождават от грижата за

локална инфраструктура. В софтуерните компании със стотици служители, създаващи комерсиални програмни продукти за бизнеса и големите корпорации, създаването на всяка сложна програма съвсем разбираемо изисква огромно количество ресурси и машини. Те имат нужда и от много офисно пространство, електрозахранване в сериозни количества, мрежова инфраструктура, скъпи мощни сървъри, компютри, сериозен набор от съхраняващи устройства и дискове, мощна охлаждаща климатична система, за да поддържа всички важни техники от сървърните помещения охладени, предотвратявайки прегряване. Главната цел на компанията би била продуктът да бъде достъпен възможно най-дълго, без нежеланите прекъсвания и сривове, които малко или много са неизбежни при собствена инфраструктура. Затова са толкова полезни облачните услуги, те предоставят наготово, спрямо необходимостта и нуждите, всички ресурси и машини за програмите, срещу разбира се - заплащане, което предвид ползите от облачните изчисления и облачните хранилища за данни е в рамките на разумното.

Въщност облачните услуги се разделят на няколко категории, или т. нар. сервисни модели, като най-受欢迎ните представляват (Фиг. 1.17)[22]:

- Infrastructure as a Service (IaaS) (Инфраструктура като услуга) - IaaS съдържа основните градивни елементи за облака и обикновено осигурява достъп до мрежови функции, компютри (виртуални или на специален хардуер) и хранилище на данни. С други думи - предоставят инфраструктурата и оставят другите аспекти като операционни системи върху тези компютри, приложенията и данните на нас да изберем какво ще правим с тях. Главни доставчици на такъв тип услуги са Amazon Web Services, Microsoft

Azure, Google Cloud Provider. Освен IaaS решения, те предлагат и услуги от предстоящите сервизни модели.

- Platform as a Service (PaaS) - Платформа като услуга - PaaS премахва необходимостта от ръчно управление на основната инфраструктура (като хардуер и операционни системи, които са ключови за IaaS) и позволява съсредоточаване върху внедряването и управлението на приложениета. Това спомага ефективността, тъй като се елиминира нуждата от притеснение относно снабдяването с ресурси, планирането на капацитета, поддръжката на софтуера, корекциите или всяка друга недиференцирана тежка работа, свързана с работата на приложениета. Гореупоменатите облачни доставчици имат услуги, които са част от PaaS като: Google App Engine, AWS Lambda и AWS Elastic Beanstalk.
- Software as a Service (SaaS) (Софтуер като услуга) - SaaS предоставя цялостен продукт, който се управлява и менажира от доставчика на услуги. В повечето случаи говорейки за SaaS, се има предвид приложения за крайни потребители (като уеб базиран имайл). С подобно приложение не са необходими знания по поддръжката и управлението на основната инфраструктура. Пример за такива приложения са Google Apps, Dropbox, Microsoft Office.

| On-site        | IaaS           | PaaS           | SaaS           |
|----------------|----------------|----------------|----------------|
| Applications   | Applications   | Applications   | Applications   |
| Data           | Data           | Data           | Data           |
| Runtime        | Runtime        | Runtime        | Runtime        |
| Middleware     | Middleware     | Middleware     | Middleware     |
| O/S            | O/S            | O/S            | O/S            |
| Virtualization | Virtualization | Virtualization | Virtualization |
| Servers        | Servers        | Servers        | Servers        |
| Storage        | Storage        | Storage        | Storage        |
| Networking     | Networking     | Networking     | Networking     |

Фиг. 1.17 Разликите между различните сервизни модели

На Фиг. 1.17 ясно се забелязват разликите между сервизните модели, като в син цвят са маркирани ресурсите, които потребителите управляват, а в сив цвят това, което доставчиците на съответните услуги управляват. Става ясно, че понякога е по-лесна и удаяна употребата на предоставени услуги, намаляйки отговорността, но това не винаги е така. Понякога може да трябва част от инфраструктурата да е локална, например ако е нещо, за което не трябва да има постоянна достъпност, като чат услуга, вътрешна за компания, където, ако има срив в системата, това няма да доведе до сериозни загуби или прекъсвания за клиентите. В такива случаи локалното решение (on-site) би било по-икономично и лесно за управление, особено ако данните или услугите са доверителни и не могат да бъдат изнесени в облака. Освен локални, решенията могат да бъдат облачни или хибридни. Облачните решения се използват, когато има услуги, които изискват висока мащабируемост, постоянна достъпност и минимални

времена за реакция. Тогава става ненужно управлението на хардуерна инфраструктура и при нужда се осигурява автоматично разширение на ресурсите. От друга страна, хибридните решения обединяват най-доброто от двата "свята" – локална инфраструктура за чувствителни и вътрешни системи, и облачни услуги за публични и мащабируеми приложения. Това позволява гъвкавост и оптимизиране на разходите според нуждите на организацията. Изборът между тези модели (on-site, cloud или hybrid) зависи от нуждите, рисковете и целите на бизнеса.

## 1.9. Amazon Web Services

AWS (Amazon Web Services/Амазон Мрежови Услуги) [23] (Фиг. 1.18) е цялостна, развиваща се платформа за облачни услуги, предоставена от компанията Amazon. Включва комбинация от услуги като споменатите по-горе инфраструктура като услуга (IaaS), платформа като услуга (PaaS) и софтуер като услуга (SaaS). Стаптира през 2002 г. от вътрешната инфраструктура, която компанията изгражда, за да управлява своите онлайн операции на дребно. През 2006 г. започва да предлага своите IaaS услуги, като една от първите компании, които въвеждат модел за облачни изчисления с "плащане в хода на работа", за да предостави на клиентите облачни изчисления и съхранение според нуждите. С времето AWS се утвърждава като лидер в облачните услуги, благодарение на широкия спектър от решения, които предлага – от изчислителна мощност и съхранение до машинно обучение и управление на бази данни. Нейната глобална инфраструктура, изградена от множество зони за достъпност (т.нар. Availability Zones) и региони (т.нар. Regions), позволява на потребителите да изграждат и внедряват приложения с висока надеждност, мащабируемост и минимално забавяне.



Фиг. 1.18 AWS лого

Централното управление на ресурсите в AWS става чрез AWS Management Console и AWS CLI най-вече. Стандартният начин е използвайки AWS Management Console, уеб-базирано приложение, което предоставя графичен интерфейс за управление на AWS услуги и ресурси. Чрез него потребителите могат лесно да създават, конфигурират и наблюдават ресурси, без необходимост от използване на команден ред или скриптове. Често използван е и AWS CLI. Представлява мощен инструмент за командния ред, който осигурява управление на всички AWS услуги. Позволява автоматизиране на задачи чрез скриптове и директно взаимодействие с AWS API, а същевременно поддържа и функции като създаване на ресурси, управление на IAM роли и разрешения, и мониторинг на услуги.

Запознаването с цялостната платформа AWS помага да се разберат отделните нейни услуги, които ще бъдат от полза за дипломната работа.

### 1.9.1. Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) [24] предоставя мащабируеми виртуални сървъри (машини) - т.нар. инстанции по заявка в облака на AWS. Тази услуга позволява на потребителите да намалят хардуерните разходи и да ускорят разработката и внедряването на приложения. EC2 осигурява възможност за динамично добавяне (scale up) или намаляване (scale down) на

изчислителния капацитет, според нуждите, например при големи натоварвания или намалена активност.

EC2 се характеризира с:

- Инстанции - виртуални сървъри в облака на AWS, които според вида на инстанцията определят наличния хардуер (CPU, памет, мрежа и съхранение).
- Amazon Machine Images (AMIs) - предварително конфигурирани шаблони за инстанции, включващи операционна система и допълнителен софтуер.
- Amazon EBS - постоянно хранилище за данни чрез Amazon Elastic Block Store, което запазва информацията, независимо от състоянието на инстанцията.
- Instance Store обеми - временни обеми за данни, които се изтриват при спиране, хибернация или изтриване на инстанцията.
- Ключови двойки (Key pairs) - информация за сигурен достъп до инстанциите, където AWS съхранява публичния ключ, а потребителят - частния.
- Групи за сигурност (Security groups) - виртуален firewall, който контролира мрежовия достъп до и от инстанциите, определяйки протоколи, портове и IP адреси.

EC2 е мощен инструмент за управление на динамични натоварвания в облачната среда с помощта на виртуални машини.

### 1.9.2. Amazon VPC

Amazon Virtual Private Cloud (Amazon VPC) [25] позволява стартирането на ресурси в AWS в логически изолирана виртуална мрежа, която наподобява традиционна мрежа в център за данни. С VPC потребителите могат да се възползват от мащабируемата

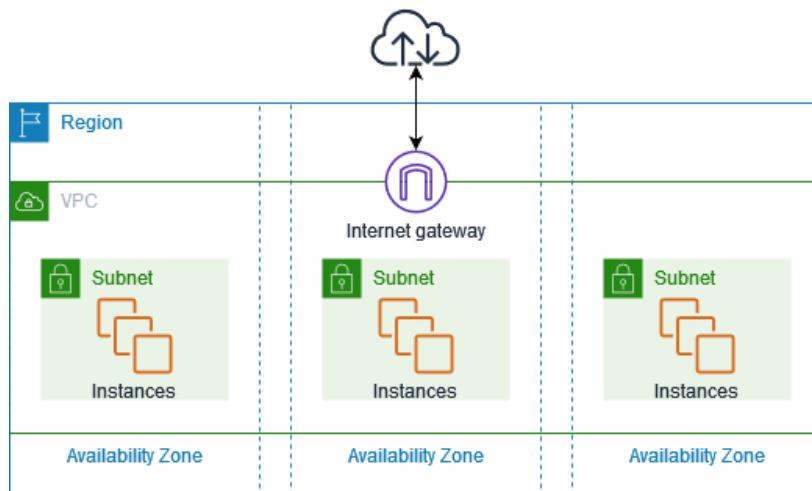
инфраструктура на AWS, като същевременно имат контрол върху мрежовата си среда.

Виртуалната мрежа има следните функции:

- VPC (Виртуален частен мрежа/облак) - виртуална мрежа, наподобяваща традиционните мрежи, като например тези в центровете за данни. След създаването и към нея могат да се добавят подмрежки (*subnets*).
- Subnets (Подмрежки) - те представляват диапазони от IP адресите в рамките на една виртуална мрежа. Всяка подмрежка е разположена в една зона на достъпност и чак след като бъде добавена, могат да се добавят и друг вид AWS ресурси в основната мрежа.
- IP адресиране - VPC поддържа присвояване на IP адреси, както IPv4, така и IPv6. Позволява също пренасяне на публични IPv4 адреси и IPv6 адреси в AWS и разпределението им към ресурси в дефинираните мрежи, като например EC2 инстанции, NAT gateways ("шлюзове") и балансьори на мрежовото натоварване.
- Маршрутизация - позволява използването на таблици с маршрути, за да определя посоката на мрежовия трафик от подмрежите или шлюзовете ("gateways").
- Шлюзове и крайни точки - шлюзът свързва виртуалната мрежа с друга мрежа. Например, използвайки интернет шлюз, е възможна връзката между VPC и интернет. Отделно, VPC крайната точка позволява частно свързване с услугите на AWS, без използването на интернет или NAT шлюзове.
- Пиъринг връзки - чрез VPC peering връзка се маршрутизира трафика между ресурсите на две виртуални мрежи.
- Копиране на мрежови трафик - с помощта на виртуалните мрежи е възможно копирането на мрежови трафик от мрежови интерфейси и препращането му към устройства за сигурност и

мониторинг, за да бъде направена дълбока проверка на пакетите, които се изпращат в мрежата.

- VPC логове на трафика и VPN връзка - логовете улавят информация относно IP трафика, отиващ към и от мрежовите интерфейси във виртуалната мрежа. Също така VPC мрежите могат да се свързват към локалните мрежи с помощта на AWS виртуална частна мрежа (VPN)



Фиг. 1.19 Примерна виртуална частна мрежа/облак (VPC)

Диаграмата на Фиг. 1.19 показва примерна виртуална мрежа с 3 подмрежки, EC2 инстанции във всяка от тях, както и интернет шлюз, който позволява комуникацията между ресурсите във вътрешната мрежа и интернет.

### 1.9.3. Amazon ELB

Amazon Elastic Load Balancing (Еластичното балансиране на натоварването) [26] автоматично разпределя входящия трафик на приложения във всички работещи инстанции на EC2. ELB помага да се управляват входящите заявки чрез оптимално маршрутизиране на трафика, така че никоя инстанция да не бъде претоварена.

ELB предоставя четири типа балансьора, които могат да се използват с групите за автоматично мащабиране на инстанциите:

Application Load Balancer (ALB), Network Load Balancer (NLB), Gateway Load Balancer (GLB) и Classic Load Balancer (CLB). Основната разлика между типовете е в начина на конфигуриране, при ALB, NLB и GLB инстанциите се регистрират като цели (targets) в целева група (target group) и трафикът се насочва към тази група. При CLB инстанциите се регистрират директно с балансьора.

Видовете балансьори на натоварването и техните особености са следните:

- Application Load Balancer (ALB) - работи и балансира натоварването на приложния слой (HTTP/HTTPS, Layer 7). Поддържа маршрутизиране на база път (path-based routing). Може да насочва заявки към портове на една или повече регистрирани цели, като EC2 инстанции в VPC.
- Network Load Balancer (NLB) - работи и балансира натоварването на транспортния слой (TCP/UDP, Layer 4). Използва информация от Layer 4 хедъра ("header") за маршрутизиране и е подходящ за големи натоварвания, защото поддържа запазване на изходния IP адрес на клиента и използва фиксиран IP адрес за продължителността на балансьора.
- Gateway Load Balancer (GLB) - разпределя трафика към група от инстанции. Осигурява мащабируемост, достъпност и простота за виртуални устройства от трети страни, като защитни стени, системи за откриване и предотвратяване на прониквания. Работи с виртуални устройства, които поддържат протокола "GENEVE", но изисква допълнителна техническа интеграция.
- Classic Load Balancer (CLB) - работи и балансира натоварването както на транспортния слой (TCP/SSL), така и на приложния слой (HTTP/HTTPS).

#### 1.9.4. Amazon CloudFormation

AWS CloudFormation (CF) е услуга [27], която помага моделирането и настройването на ресурсите в AWS, така че да се спести време от управление на време и за да се постави фокус върху приложениета, които работят в AWS. Създава се шаблон, който описва всички ресурси на AWS, които желаем (като екземпляри на Amazon EC2 или Amazon VPC), а CloudFormation се грижи за осигуряването и конфигурирането на тези ресурси вместо нас (т.нар. стек). Не е необходимо индивидуалното създаване и конфигуриране AWS ресурси и знания относно кое с кое работи. CloudFormation се справя с това. Помага с: опростяването, бързото пресъздаване, лесния контрол и наблюдението на инфраструктурата. Шаблонът в CF е YAML или JSON форматиран текстов файл и позволява запазване с всяко разширение, като .yaml, .json, .template или .txt. и CloudFormation ще ги използва като чертежи за изграждане на дефинираните AWS ресурси. Когато се използва CloudFormation обаче, за да се създаде стека, той извиква основни услуги на AWS, за да предостави и конфигурира описаните ресурси. Например, при създаване на екземпляри на EC2, CF ще ги конфигурира, стига да има необходимите разрешения. Те се управляват чрез AWS Identity and Access Management (IAM).

#### 1.9.5. Amazon IAM

AWS Identity and Access Management (IAM) (Управление на самоличността и достъпа) е уеб услуга [28], която осигурява сигурно управление на достъпа до ресурси в AWS. Чрез IAM се управляват разрешенията, определящи кои ресурси могат да бъдат достъпвани от потребителите. IAM предоставя инфраструктура за контрол на удостоверяването (автентикация) и упълномощаването (авторизация) на акаунти в AWS. При създаване на AWS акаунт се създава основна

идентичност със пълен достъп до всички услуги и ресурси в акаунта. Тази идентичност е известна като root потребител и се достъпва с имейл адреса и паролата, използвани за създаване на акаунта. Препоръчително е root потребителят да не се използва за ежедневни задачи, а неговите идентификационни данни да бъдат защитени и използвани само за действия, които изискват root права. Чрез IAM могат да се създават допълнителни идентичности, като администратори, анализатори и разработчици, които да получават достъп до необходимите ресурси за изпълнение на техните задачи. След като потребител е конфигуриран в IAM, неговите идентификационни данни се използват за удостоверяване. Удостоверяването се осъществява чрез съпоставяне на тези данни с принципал (например IAM потребител, федеративен потребител, IAM роля или приложение), който е доверен от AWS акаунта. Следва заявка за предоставяне на достъп до ресурси, достъпът се предоставя, ако принципалът притежава разрешения за съответния ресурс. Например, при влизане в конзолата и навигация към определена услуга, се изпраща заявка за упълномощаване. Услугата проверява дали идентичността е в списъка с оторизирани потребители, какви политики са приложени за контрол на достъпа и дали има други действащи политики. Заявките за авторизация могат да бъдат направени както от принципали в акаунта, така и от друг доверен AWS акаунт.

#### 1.9.6. Amazon EKS

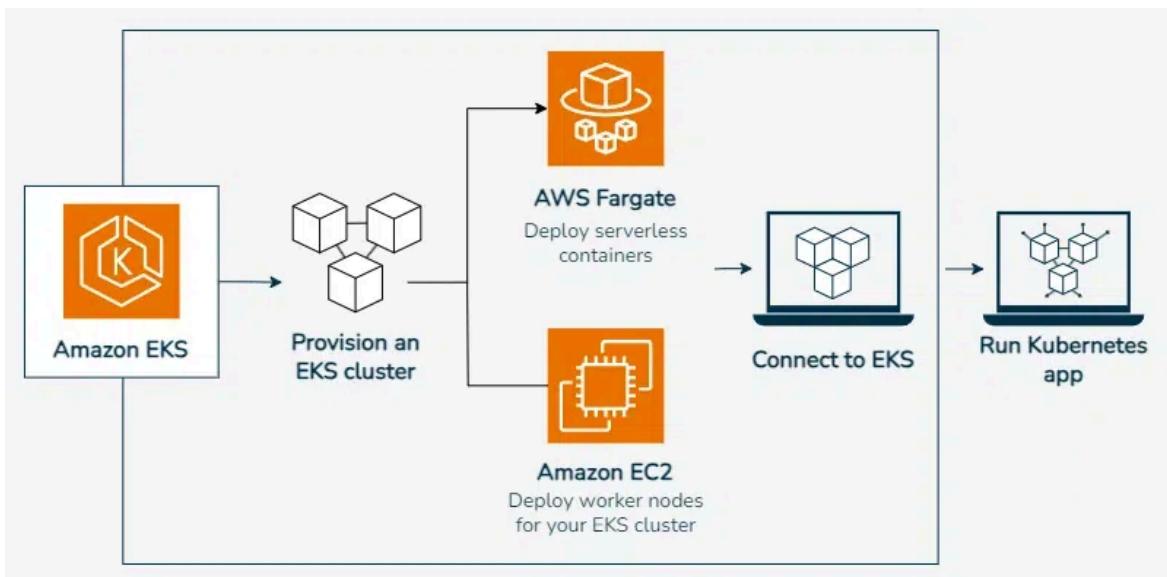
Последната, но не по важност, услуга на AWS, която има главна роля в дипломната работа, е Amazon Elastic Kubernetes Service (EKS) (Еластична Кубернетес услуга). Това е услуга на Kubernetes, която елиминира необходимостта от поддръжка на наличността и

мащабирането на Kubernetes кълъстери в Amazon Web Services (AWS) и собствените центрове за данни [29].

Основните характеристики на EKS представляват следните:

- Напълно управляван от AWS Kubernetes - Amazon EKS предлага скалируем и високо достъпен контролен слой за Kubernetes, разпределен между множество зони за достъпност (AZs) в AWS. Автоматично управлява наличността и скалируемостта на Kubernetes API и etcd, а с EKS Auto Mode има възможността да опрости и управлението чрез автоматично предоставяне на инфраструктура, динамично мащабиране, оптимизация на разходите и интеграция със системите за сигурност на AWS.
- Съвместимост - EKS използва upstream Kubernetes, което осигурява съвместимост с всички стандартни плъгини и инструменти. Приложението, работещи на EKS, са съвместими с тези в стандартни Kubernetes среди, като миграцията към EKS не изисква промяна на кода.
- Мащабириеми AI/ML натоварвания - EKS е оптимизиран за приложения с изкуствен интелект и машинно обучение (AI/ML) и поддържа GPU инстанции (с графичен процесор) и AWS Neuron, осигурявайки висока производителност за обучение и ниска латентност за инференция.
- Компютинг - EKS използва разнообразни типове инстанции на Amazon EC2 (Фиг. 1.20), като Graviton и Nitro, които служат за оптимизиране на производителността и разходите, но позволява и използването на Spot инстанции за икономия.
- Мрежова интеграция - EKS интегрира гореспоменатите виртуални мрежи за сигурност и управление на мрежови ресурси. Поддържа IPv4, IPv6 и инструменти като Application Load Balancers (ALB) и Network Load Balancers (NLB) за управление на трафика.

- Сигурност - за да гарантира сигурността, EKS интегрира IAM услугата за управление на достъпа до Kubernetes обектите.
- Наблюдение и логове - може да интегрира услуги като Amazon CloudWatch, AWS Managed Service for Prometheus и Amazon GuardDuty за мониторинг, събиране на логове и анализ на заплахи на кълстерите.
- Съхранение - поддържа множество AWS услуги за съхранение чрез Container Storage Interface (CSI) (Интерфейс за контейнерно хранилище), включително Amazon EBS, Amazon S3, Amazon EFS и Amazon FSx.
- Add-ons - EKS предоставя предварително конфигурирани добавки за мрежи, баланс на натоварването, сигурност и други, както и интеграция с Kubernetes инструменти от трети страни.
- Управление - предлага интерфейси като eksctl, AWS CLI, Terraform и CloudFormation за конфигуриране и поддръжка. AWS Controllers for Kubernetes (ACK) позволява управление на AWS услуги директно през Kubernetes.
- Операционни системи - поддържа оптимизирани AMI изображения за Amazon Linux, Bottlerocket, Windows, Ubuntu, както и възможност за използване на персонализирани AMI.



Фиг. 1.20 Архитектура използваща EKS с EC2 и Fargate инстанции

## 1.10. Cluster API

### 1.10.1. Принцип на работа

Cluster API (CAPI) е подпроект на Kubernetes [30], фокусиран върху предоставянето на декларативни API и инструменти за опростяване на осигуряването, надграждането и работата с множество Kubernetes кълстери. Проектът е стартиран от “Kubernetes Special Interest Group (SIG) Cluster Lifecycle” и използва API и шаблони, подобни на тези от Kubernetes, за да автоматизира управлението на жизнения цикъл на кълстерите. Поддържащата инфраструктура, като виртуални машини, мрежи, балансьори на натоварването и VPC, както и конфигурацията на кълстера на Kubernetes, са дефинирани по същия начин, по който разработчиците на приложения работят с внедряването и управлението на своите приложения. Това позволява последователни и повтарящи се внедрявания на кълстери в голямо разнообразие от инфраструктурни среди. Както се разбира, Kubernetes е сложна платформа, която изиска правилна конфигурация на множество компоненти за работещ кълстер. Предлага огромно разнообразие от над 100 дистрибуции и инсталатори с различни настройки, но това често създава затруднения за потребителите. Имайки това предвид, групата “SIG Cluster Lifecycle” стартира проекта “kubeadm” като инструмент за начално стартиране на кълстери. Въпреки че този инструмент, в съвкупност с други подобни такива, намаляват сложността на инсталацията на кълстерите, не решават проблеми с управлението им и дългосрочната поддръжка, която те изискват. Нерешени остават въпроси като консистентното създаване на ресурси, като машини, балансьори на натоварването и VPC през различните инфраструктурни доставчици (AWS, Azure); също автоматизацията на жизнения цикъл на кълстерите, включително обновления на версии и изтриване, както и управлението на множество кълстери в големи мащаби. Тъкмо там

идва на помощ Cluster API, като запълва тези пропуски с набор от поддекларитивни Kubernetes APIs, които автоматизират създаването, конфигурацията и управлението на кълстери. Поддържа голямо разнообразие от инфраструктурни доставчици, като гореспоменатия AWS, отделно Azure, vSphere и др.

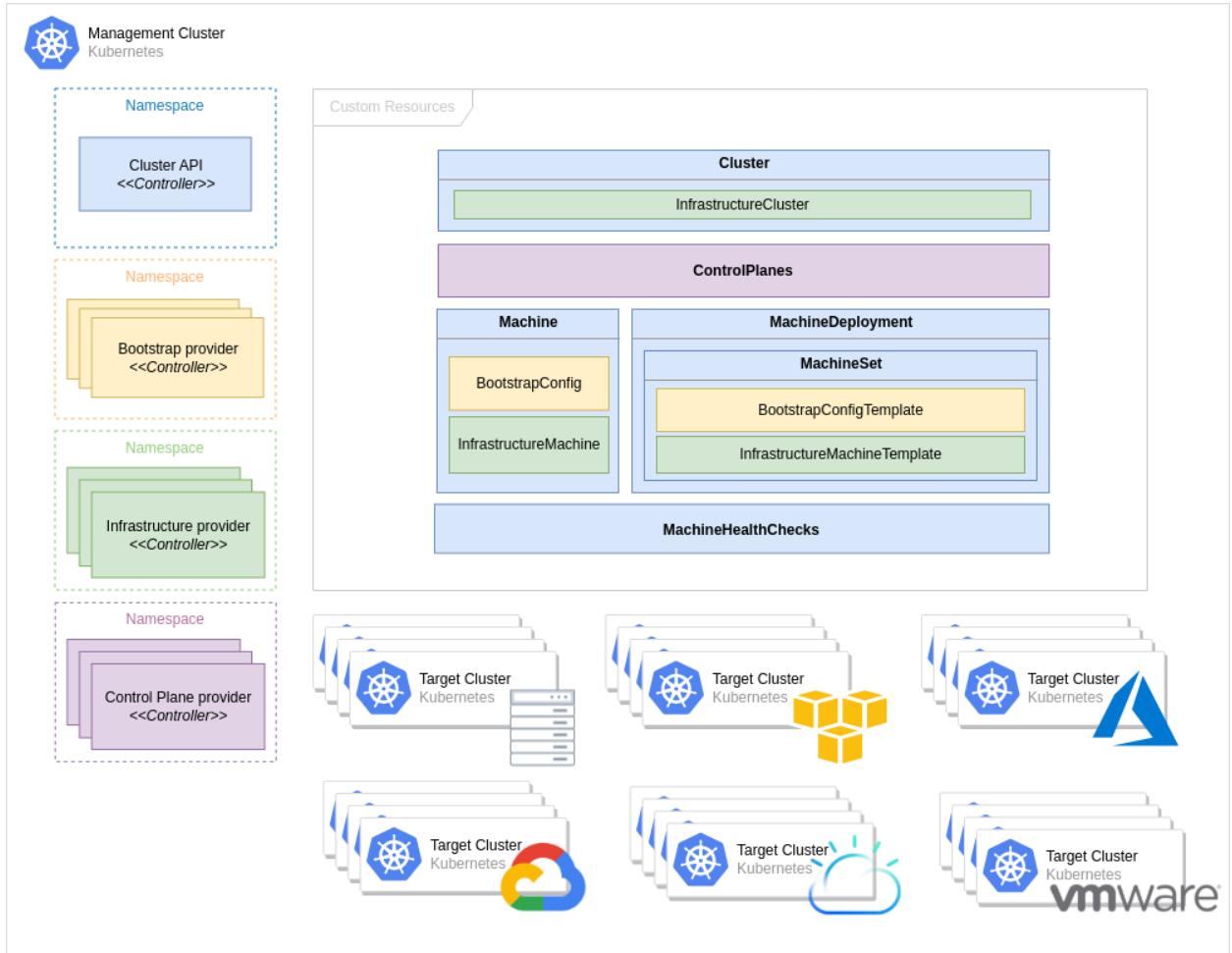
#### 1.10.2. Основни компоненти и архитектура

CAPI кълстери се разделят на два типа: управляващи (т.нар. management) и управлявани (т.нар. workload) [31]. Управляващият Kubernetes кълстер е този, който се грижи за жизнения цикъл на управляваните кълстери. Върху този управляващ кълстер работят инфраструктурните доставчици (providers) и се съхраняват работните машини (machines). Управляваният кълстер е Kubernetes кълстер, чийто жизнен цикъл се управлява от т.нар. "Management" кълстер. Такив тип кълстери се използват за хостване на работни натоварвания/приложения (workloads).

Отделно в CAPI има инфраструктурни доставчици ("Infrastructure Providers", които са отговорни за осигуряването на инфраструктурни ресурси - виртуални машини и мрежи. Примерни облачни доставчици са AWS, Azure и Google Cloud, а за физически доставчици: VMwage, MAAS и metal3.io. Различните ресурси, предоставяни от един доставчик, като например EC2 и EKS услугите на AWS, се наричат варианти.

Също има компонент в CAPI, който превръща сървърите в Kubernetes работни машини (nodes), наречен "Bootstrap provider". Този компонент генерира необходимите сертификати за кълстера, инициализира контролния слой и го присъединява, заедно с работните машини, към кълстера. Архитектурата е демонстрирана на Фиг. 1.21.

CAPI използва съществуващите в клъстерите контролни слоеве, за да постигне управлението на жизнените цикли, но добавя допълнителни APIs.



Фиг. 1.21 Архитектура на клъстер управляем с CAPI

Има вариации на тези контролни слоеве, като например:

- Self-provisioned (самоосигурен) - контролен слой в Kubernetes, изцяло управлявани от една инсталация на Cluster API (напр. kubeadm, който използва статични обвивки за компоненти като kube-apiserver, kube-controller-manager и kube-scheduler върху машините на контролния слой).

- Pod-based (базиран на обвивки) - контролен слой в друг кълъстер, като компонентите се разполагат чрез стандартни Deployment или StatefulSet обекти, а API се предоставя чрез Service.
- External/Managed (Външен/Управляван) - контролен слой, предлаган и управляван от външни системи, различни от Cluster API, като GKE, AKS, EKS или IKS. Такъв тип контролен слой се използва в дипломната работа, с помощта на AWS EKS.

#### 1.10.3. Интеграция на CAPI в Amazon AWS

Както става ясно, CAPI работи върху множество инфраструктурни доставчици, но фокус на дипломната работа е AWS. За тази цел се използва Cluster API Provider AWS (CAPA), основните характеристики представляват:

- Нативна интеграция с Kubernetes - използва Kubernetes манифести и API за управление на инфраструктурата.
- Управление на инфраструктурата - автоматизира създаването на виртуални мрежи, шлюзове, групи за сигурност и EC2 инсталации.
- Избор на ОС - позволява използването на предварително конфигурирани AMI изображения, които AWS предоставят.
- Частни подмрежи - разполага създадения контролен слой в частни подрежи с отделен достъп.
- Минимални компоненти - инсталира само основните компоненти, необходими за инициализация на контролния слой и работните машини (nodes), без да използва SSH.
- Поддръжка на EC2 и EKS - позволява разполагането на контролния слой върху EC2 инстанции и осигурява съвместимост и управление на EKS кълъстери.

#### 1.10.4. Clusterawsadm и clusterctl

Има два основни начини за стартиране на клъстери в CAPI, чрез “clusterctl” и чрез “CAPI Operator”. В обхвата на дипломната работа е използван интерфейсът за команден ред “clusterctl” [33]. Той е специално разработен за осигуряване на бърз старт с CAPI, като автоматизира създаването на YAML файловете, определящи компонентите на инфраструктурния доставчик и тяхното инсталлиране. Освен това, в него е кодиран набор от най-добри практики за управление на доставчиците, които помагат на потребителя да избягва неправилни конфигурации например.

Отделно има друг интерфейс за команден ред, който позволява инициализацията на CAPI. Т. нар. “clusterawsadm” [34] предоставя помощници за стартиране на Kubernetes CAPI. Чрез него могат да се видят необходимите политики за управление на идентичността и достъпа (IAM) на AWS, като JSON файлове, или да се създадат IAM роли и профили на инстанциите автоматично, с помощта на AWS CloudFormation.

## ВТОРА ГЛАВА

*Проектиране на инфраструктура и приложение за демонстриране на функционалностите на CAPI*

### 2.1. Основни и функционални изисквания (ФИ)

#### 2.1.1. ФИ за създаване и настройка на кълъстери в облачна среда

Според изискванията на дипломната работа кълъстерът, който в случая ще изпълнява ролята на т.нар. "управляващ" кълъстер в CAPI, който осигурява управление на жизнения цикъл на други Kubernetes кълъстери, трябва да бъде създаден и настроен в облачната среда AWS.

За целта, кълъстерът трябва да отговаря на следните функционални изисквания:

- Създадена виртуална частна мрежа (VPC), с помощта на AWS CF, и конфигурирани подмрежки (subnets) в различни зони за достъпност (availability zones), за да бъде осигурена висока наличност на кълъстера.
- Конфигуриране на групи за сигурност (security groups), които да ограничават достъпа до EC2 инстанциите, според протоколите, IP адресите и портовете, упоменати в групите.
- Първоначално конфигуриране и избор на изчислителни ресурси (EC2 инстанции) с помощта на Node Group (група от машини) функционалността на AWS, за да поддържат компонентите на Kubernetes и да осигурят успешната инициализация на контролния и работния слой.

Допълнително, настройката на инструмента за командния ред на Kubernetes (kubectl), за да разговаря с управляващият кълъстер, е необходима за по-нататъчно конфигуриране на CAPI. С помощта на

ключовете за достъп на AWS: access key (ключ за достъп) и secret access key (таен ключ за достъп), както и интерфейса за командния ред на AWS (AWS CLI), можем да променим конфигурацията на kubectl.

### 2.1.2. ФИ за инсталация и настройка на CAPI

Cluster API изисква няколко локално инсталирани интерфейса, подготовка на средата и конфигуриране на основните компоненти, които ще позволят създаването и управлението на Kubernetes кълстерите.

Необходими са:

- Създаден Kubernetes кълстер, който ще функционира като управляващ (management), с достатъчно изчислителни ресурси и мрежова конфигурация с интернет достъп, за да може CAPI да изтегли необходимите API контролери върху кълстера.
- Локално инсталирани и конфигурирани clusterctl и clusterawsadm. С clusterawsadm се създават правилните IAM политики и роли и се кодират идентификационните данни (credentials), необходими на clusterctl за инициализация и инсталация на необходимите контролери върху управляващият кълстер.
- Експортирани променливи на средата (environment variables), например двата ключа за достъп в AWS (Access key и Secret Access Key), типа EC2 инстанции, които CAPI да създаде за работните кълстери, като работни възли (nodes), и региона в който искаме новосъздадените кълстери и машини.

### 2.1.3. ФИ за създаване на допълнителни кълстери чрез CAPI

Допълнителните кълстери в CAPI се нуждаят от вече конфигуриран и инициализиран управляващ кълстер. Този кълстер отговаря за жизнения цикъл на допълнителните кълстери, включително

създаването, конфигурирането, мащабирането и изтриването им. Ключова част от процеса за създаването е манифеста, посредством който се създават конфигурациите за допълнителните кълстери. Той се създава чрез `clusterctl`, конзолния интерфейс на CAPI за генериране на кълстери. Чрез него се автоматизира конфигурирането на ресурси като контролния слой, работните възли, мрежовите настройки и останалите инфраструктурни зависимости. След като бъде създаден, манифестът се прилага върху управляващия кълстер с командния инструмент на Kubernetes, а именно `kubectl`. След като бъде приложен (`applied`), се започва процесът по инициализация на описаните в манифеста ресурси (кълстери, машини, контролен слой), като CAPI осигурява желаното състояние (`desired state`) на ресурсите. Също така се контролира достъпът до кълстера чрез IAM ресурсите и отделно групите за сигурност (`security groups`), които контролират трафика от и към кълстера, както и машините върху него.

#### 2.1.4. ФИ за инсталране на работни програми върху кълстерите създадени от CAPI

За инсталацията на работни програми върху новосъздадените кълстери е необходимо те да бъдат предварително конфигурирани и напълно функционални. Трябва да има също така достъп до кълстерите. Генерира се конфигурационен файл чрез конзолния интерфейс AWS CLI, с команда като `"aws eks update-kubeconfig"`. Тя променя действащата конфигурация на `kubectl`, относяща се за управляващия кълстер, за да може Kubernetes манифестите да бъдат изпълнени върху кълстерите, създадени с CAPI. Тези манифести трябва да са предварително направени и описващи поведението на Kubernetes ресурсите, които програмите ще използват, например `"Deployment"`, `"StatefulSet"` и `"Service"`. След прилагане на конфигурационните им файлове (манифестите) чрез командалата

“kubectl apply”, статусът на внедряването им в кълъстери може да се наблюдава чрез команди, като например “kubectl get” и “kubectl describe”, с името на ресурса, който искаме да следим. Добре конфигурираната и дефинирана инфраструктура гарантира, че приложенията ще работят стабилно и безпроблемно в Kubernetes средата, създадена с помощта на CAPI.

#### 2.1.5. ФИ за демонстриране на функционалност на CAPI

Демонстрацията на CAPI включва позване на възможностите му за автоматизация и управление на Kubernetes кълъстери. Например:

- Създаване на нови кълъстери чрез генериране и прилагане на манифести и демонстриране на инициализацията на контролния слой и работните машини върху инфраструктурния доставчик (AWS).
- Демонстриране на способността на CAPI за автоматично мащабиране, чрез добавяне или премахване на машини в кълъстера, както и актуализация на версията на Kubernetes на новосъздадените кълъстери.
- Показване на способността за автоматизирано изтриване на кълъстери, както и всички кълъстерни ресурси.
- Заделяне на ресурси чрез CAPI, демонстрирайки автоматичното създаване на ресурси като нови инстанции, по-големи хранилища, или промяна в мрежовите конфигурации.
- Показване на възможността за инсталiranе на работни приложения върху създадените кълъстери.

## 2.2. Избор на облачна среда

Ключова за демонстрацията на CAPI е реализацията на инфраструктурата и изборът ѝ. В контекста на дипломната работа е

избрана платформата Amazon Web Services (AWS) която е водещата платформа за облачни услуги и е призната за най-всеобхватната и широко разпространена облачна платформа в света, с над 200 функционални услуги и центрове в цял свят. Предпочитана е, защото осигурява високонадеждна, мащабируема, гъвкава и евтина инфраструктура за десетки хиляди предприятия. Далеч по-усъвършенствана е от останалите платформи и предоставя предимства като:

- Леснота на използване - проектирана е по такъв начин, че да позволява на разработчиците и продавачите бързо и сигурно да хостят приложенията си - независимо дали става въпрос за съществуващо приложение, или за ново приложение, базирано на SaaS (Софтуер като услуга). За достъп до платформата за хостинг на приложения на AWS може да се използва конзолата за управление на AWS или добре документираните API за уеб услуги.
- Гъвкавост - дава възможност за избор на операционна система, език за програмиране, платформа за уеб приложения, базите данни и други необходими услуги. С AWS се получава виртуална среда, която позволява зареждането на софтуера и услугите, от които се нуждаят приложенията, което улеснява процеса на миграция на съществуващите приложения, като същевременно запазва възможностите за изграждане на нови решения.
- Рентабилност - плаща се само за използваната изчислителна мощност, хранилища и други ресурси, без дългосрочни договори и ангажименти.
- Надеждност - с AWS имате достъп до мащабируема, надеждна и сигурна глобална изчислителна инфраструктура, която е "виртуалният гръбнак" на многомилиардния онлайн бизнес на

Amazon.com, който е усъвършенстван в продължение на повече от десетилетие.

- Машабируема и високопроизводителна - с инструментите като автоматичното машабиране, еластичното балансиране на натоварването, приложенията могат да се скалират в зависимост от търсенето.
- Сигурност - AWS използва цялостен подход за защита и укрепване на инфраструктурата, включително физически, оперативни и софтуерни мерки.

В сравнение с другите инфраструктурни доставчици, като GCP (Google Cloud Platform) и Microsoft Azure, в контекста на реализацията и демонстрацията на възможностите на CAPI (Cluster API), AWS предлага някои съществени предимства. За инфраструктурата на AWS доставчикът, предлаган от CAPI, т. нар. Cluster API Provider AWS, предоставя обширна поддръжка за създаване и управление на жизнения цикъл на Kubernetes клъстерите. GCP и Azure също предлагат своите CAPI доставчици, но поддръжката им не е документирана и разпространена на същия машаб като тази MAAWS, осъбено за сложни инфраструктури.

Откъм мрежова свързаност, от друга страна, услугите за виртуални мрежи (VPC) и техните компоненти, като подмрежи, интернет и NAT шлюзове, балансьори на натоварването, които AWS предлага, са несравними с тези на другите инфраструктурни доставчици, въпреки че и те предлагат мрежова свързаност, с решения като "Cloud Interconnect" на GCP и "Virtual Network" на Azure. С тях обаче поддръжката на CAPI мрежовите функционалности не е толкова богата, а конфигурацията е по-сложна и изисква повече ръчни настройки.

AWS също предлага лесно управление на достъпа чрез IAM, което лесно се интегрира с Kubernetes. AWS има и някои ограничения, като по-малката инфраструктурна гъвкавост, тъй като GCP и Azure предлагат по-голям брой региони и зони на достъпност (availability zones). Въпреки това, стабилната и глобално разпределена инфраструктура, съчетана с най-пълноценната поддръжка за CAPI, правят AWS предпочитания избор за демонстрация на възможностите на CAPI в рамките на дипломната работа.

### **2.3. Избор на инструмент за контейнеризация**

За избора на инструмент за контейнеризация няма много конкуренция на Docker. Най-широко разпространен е и се превръща в стандарт за контейнерите, благодарение на лекотата за използване, екосистемата си и силната поддръжка на платформата. Предлага мощни инструменти за създаване, управление и внедряване на контейнери, които осигуряват преносимост и лесно мащабиране при разработка на приложения.

Сред основните предимства на Docker са:

- Широката съвместимост, тъй като работи с почти всички операционни системи и инфраструктурни доставчици. Освен това се инсталира от единичен пакет и в рамките на минути може да е функционален.
- Лесно управление на изображенията, посредством Docker Hub, който предоставя хранилище за споделяне и достъп до хиляди съществуващи образи.
- Поддръжката на оркестрация, чрез безпроблемната интеграция, с инструменти като Kubernetes и други оркестрационни

платформи, което го прави ключов избор за платформи с мащабириуеми среди.

- Лекотата на използване, с помощта на CLI-базирания процес на работа, изграждането, споделянето и изпълнението на контейнеризирани приложения, е достъпно за разработчици с всякакви умения. В допълнение, с обширната документация и активната продължителна разработка на инструмента, Docker допълнително улеснява разработчиците.

В контекста на дипломната работа, Docker се използва за контейнеризация на приложениета, които ще бъдат внедрявани в Kubernetes кълстерите. Интеграцията на Docker с Kubernetes осигурява стабилност и ефективност при управлението на контейнерите. В комбинация с гореспоменатите предимства, Docker е най-подходящият избор за инструмент за контейнеризация.

## **2.4. Избор на инструмент за оркестрация**

Kubernetes се превръща в синоним за оркестрация на контейнери. Неговите функционалности включват автоматизирано разполагане на контейнери, мащабиране и управление, заедно с балансиране на натоварването и непрекъснати актуализации. Kubernetes има жизнена екосистема с широк набор от плъгини, инструменти и разширения, правейки го универсален избор както и за малки фирми и локални разработки, така и за големи предприятия.

Ключови характеристики на Kubernetes включват:

- Мащабируемост - Kubernetes е проектиран да се мащабира без усилие, способен да управлява хиляди възли и контейнери в един кълстер.

- Поддръжка - с голяма и активна общност Kubernetes се възползва от непрекъснати подобрения, иновации и обширна документация.
- Гъвкавост - Kubernetes предлага детайлен контрол върху работни натоварвания (workload) в контейнери, позволявайки на потребителите да дефинират персонализирани изисквания за ресурси, мрежови политики и конфигурации за съхранение.

Съществуват и други инструменти за оркестрация, като например Docker Swarm и Apache Mesos. Docker Swarm е "родният" инструмент за оркестиране на Docker контейнери, проектиран да бъде прост, лесен за използване и тясно интегриран с екосистемата на Docker. За разлика от Kubernetes, който е отделен проект, създаден от Google, Docker Swarm е част от Docker Engine, което го прави без проблемен за екипи, вече запознати с инфраструктурата и командите на Docker. Отличава се с простота и лекота на използване, минималистичен дизайн и архитектура, която улеснява настройването, внедряването и управлението на контейнерни приложения. Другата алтернатива за оркестрация - Apache Mesos, е инструмент за изолация на изчислителни ресурси като CPU, памет и съхранение в клъстър от машини. Въпреки че Mesos не е специално проектиран за оркестрация на контейнери, той може да ги управлява, както и други видове натоварвания, като виртуални машини и традиционни приложения. Mesos абстрактира и обединява изчислителни ресурси в клъстър, позволявайки ефективно използване и изолиране на ресурси за различни работни натоварвания. Също така поддържа широка гама от работни натоварвания, включително контейнери, виртуални машини и традиционни приложения, което го прави универсален избор за различни среди.

Сравнителен анализ между Kubernetes и другите оркестрационни технологии:

- Kubernetes се отличава с мащабируемост, като поддържа големи и сложни внедрявания. Архитектурата му е проектирана да обработва ефективно хиляди възли и контейнери.
- Docker Swarm често е хвален за своята простота, което улеснява потребителите, които са нови в оркестрацията на контейнери. Kubernetes, макар и мощен, има по-стръмна крива на обучение, поради своята богата на функции природа.
- Kubernetes може да се похвали с голяма и активна общност, която допринася за нейната обширна екосистема от плъгини, инструменти и разширения. Това спомага с актуализацията и обновяването на платформата, което осигурява нейното непрекъснато подобреие.
- Kubernetes осигурява високо ниво на гъвкавост, позволявайки на потребителите да конфигурират и персонализират различни аспекти на процеса на оркестрация.

Въпреки че инструменти като Docker Swarm и Apache Mesos имат свои специфични предимства, Kubernetes предлага пълен набор от функции, които отговарят на нуждите на съвременните облачни и хибридни инфраструктури. Това го прави идеален за управление на сложни и мащабируеми приложения и в контекста на дипломната работа - най-добрият избор за оркестрация на контейнери.

## 2.5. Избор на инструмент за менажиране на клъстери

За централно управление на жизнените цикли на множество Kubernetes клъстери в дипломната работа е използван инструментът Cluster API, като най-подходящ за тази цел. Друг популярен инструмент, който би могъл да се използва в случая, е Crossplane.  
стр. 64/114

Въпреки че и двата инструмента предоставят възможност за управление на инфраструктурата и клъстерите, те се различават в подхода и функционалностите си.

CAPI е инструмент, фокусиран изцяло върху автоматизацията на жизнения цикъл на клъстерите. Чрез декларитивен подход, използвайки Kubernetes манифести, предоставя ефективен начин за създаване, управление и изтриване на клъстери в множество среди.

Основните предимства на Capi са:

- Декларитивният подход, чрез Kubernetes манифести, за управлението на клъстери;
- Използването на управляващ клъстер, който да служи като централен "хъб" за управление на клъстери;
- Поддръжката на различни инфраструктурни доставчици, например AWS, GCP, Azure, VMwage;
- Фокусът върху Kubernetes, правейки го оптимален избор за управление на клъстери, и техния жизнен цикъл.

От друга страна, Crossplane разширява API на Kubernetes и позволява декларитивно управление на инфраструктурни ресурси, като бази данни, VPC и балансьори на натоварването, както и на Kubernetes клъстери. Инструментът се стреми да обедини управлението на приложения и инфраструктурата им в една платформа.

Предимствата на Crossplane включват:

- Управление на цялостна инфраструктура, не само клъстери.
- Разширяемост с помощта на широк набор от доставчици;
- Интеграция за сложни зависимости, което го прави подходящ за хибридни и комплексни среди.

Изборът между CAPI и Crossplane зависи от функционалните изисквания на проекта. Фокусът на дипломната работа, бивайки управлението жизнените цикли на Kubernetes кълстери и автоматизацията им, CAPI е по-добрият избор, тъй като е създаден с тази цел, като подпроект на Kubernetes, напълно интегриран с тяхната екосистема.

## 2.6. Избор на работно приложение

В контекста на дипломната работа работните приложения, които ще бъдат разположени върху кълстерите, създадени с помощта на CAPI, са WordPress сайт, с MySQL база данни и Grafana, инсталарирана чрез Helm. Основната цел на тези приложения е демонстрацията на възможностите на CAPI за автоматизация и управление на жизнения цикъл на Kubernetes кълстери, като самите приложения са избрани, заради лекотата на конфигурация и съвместимостта с Kubernetes.

Wordpress е добре познат пример за приложение, което изискава съхранение чрез PersistentVolume и PersistentVolumeClaim и демонстрира взаимодействието между различни компоненти в Kubernetes.

Чрез него могат да се покажат основни концепции като:

- Декларативно управление на ресурси, с помощта на манифести.
- StatefulSet за съхранение на данните от MySQL базата.
- Използване на мрежови ресурси, като сървис и ингрес, за достъп до приложението.

Grafana се инсталира посредством Helm, който е пакетен мениджър за Kubernetes и улеснява управлението, внедряването и обновяването на приложения, чрез предварително дефинирани шаблони (т.нар. Helm Charts). С помощта на Grafana се демонстрира способността на Kubernetes да работи с приложения, които се

инсталират чрез популярни пакетни мениджъри и макар и да не включва Prometheus, тя може да визуализира и демонстрира настройките за достъпност и мрежовата конфигурация.

Освен демонстрацията на жизнения цикъл на клъстерите, ролята на работните приложения е да проверяват дали новосъздадените клъстери, мрежовите настройки и съхранението са конфигурирани правилно. Тези приложения не са сложни, тъй като основният фокус на дипломната работа е върху демонстрацията на възможностите на CAPI, а не върху сложността на това което върви върху клъстерите.

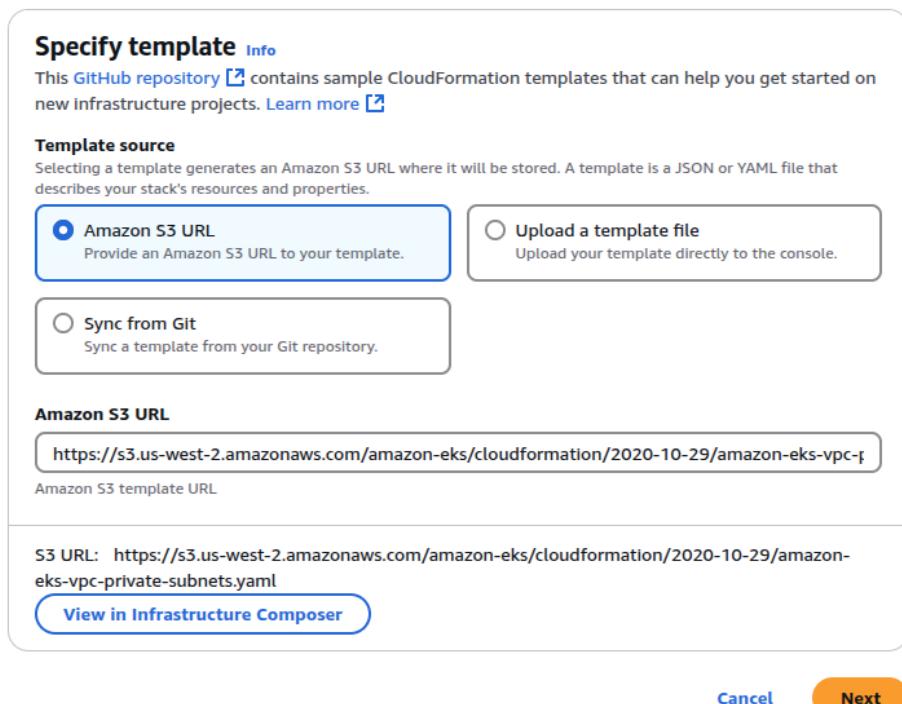
## ТРЕТА ГЛАВА

*Реализация на инфраструктура и приложения за демонстриране на функционалностите на CAPI*

### 3.1. Конфигуриране на VPC за EKS чрез CF

Създаването на инфраструктурата започва с управляващия кълстер, върху който са разположени API's на CAPI, с помощта на които се създават допълнителните кълстери. За целта, кълстерът трябва да бъде разположен във виртуална мрежа (VPC). Използвайки CF [35], направата на тази мрежа и нейните подмрежи е опростена, като това става чрез предефиниран шаблон за съответните ресурси от AWS.

През AWS конзолата в CloudFormation се създава стек с бутона "Create Stack". CF позволява два начина за конфигуриране на ресурси: шаблон и инфраструктурен композитор. Тъй като шаблонът вече съществува в полето за източник, се избира "Amazon S3 URL" и се поставя линкът [36], предоставен от документацията за създаване на VPC за EKS на AWS (Фиг. 3.1).



Фиг. 3.1 Създаване на VPC за EKS чрез CF

След избирането на име за стека конфигурацията по подразбиране на останалите параметри не подлежат на промяна и остават такива, каквото са, и стекът бива създаден. В последствие, след известно време, в конзолата на AWS при услугата CloudFormation ще се намира новосъздаденият стек, с името, което по-рано е избрано. Подаденият шаблон, предоставен от Amazon, създава 20 ресурса, вариращи от самата виртуална мрежа, до 2 публични и 2 частни подмрежи, техните маршрутизиращи таблици, NAT и интернет шлюзове, заделени публични адреси за съответните NAT шлюзове и групите за сигурност. Крайният успешно създаден стек чрез CF е демонстриран в конзолата на AWS на Фиг. 3.2.

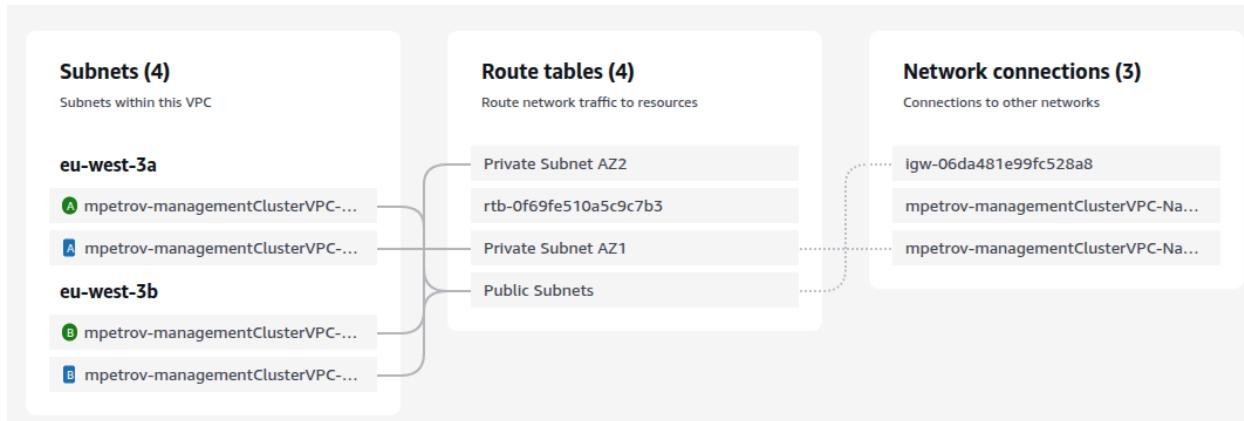
| Stack name                                   | Status          |
|--|-----------------|
| <a href="#">mpetrov-managementClusterVPC</a> | CREATE_COMPLETE |

Фиг. 3.2 Създаден и конфигуриран стек на VPC с помощта на CF

Освен този преконфигуриран шаблон с 2 публични и 2 частни мрежи, AWS предоставят и още два [33], единият от които се състои само от 3 частни подмрежи, другият само от 3 публични. За да могат да бъдат изтеглени гореупоменатите API, с помощта на които CAPI да управлява допълнителните клъстери, трябва достъп до интернет. Избраният шаблон за направата на VPC има налична интернет връзка, докато за шаблона, състоящ се само от частни подмрежи, трябва да бъде създаден ръчно NAT шлюз, за да има такава, което е излишна стъпка в конфигурацията. Другият шаблон, съдържащ само публични подмрежи, не се използва, тъй като не е необходим и без него се намалява излишното излагане на ресурси.

Във VPC конзолата на AWS се намира новосъздадената частна мрежа и карта на нейните ресурси (Фиг. 3.3). Тя демонстрира връзките

между отделните подмрежи, таблици за маршрутизация и мрежовите връзки (интернет и NAT шлюзовете).



Фиг. 3.3 Карта на ресурсите на VPC за управляващия клъстор

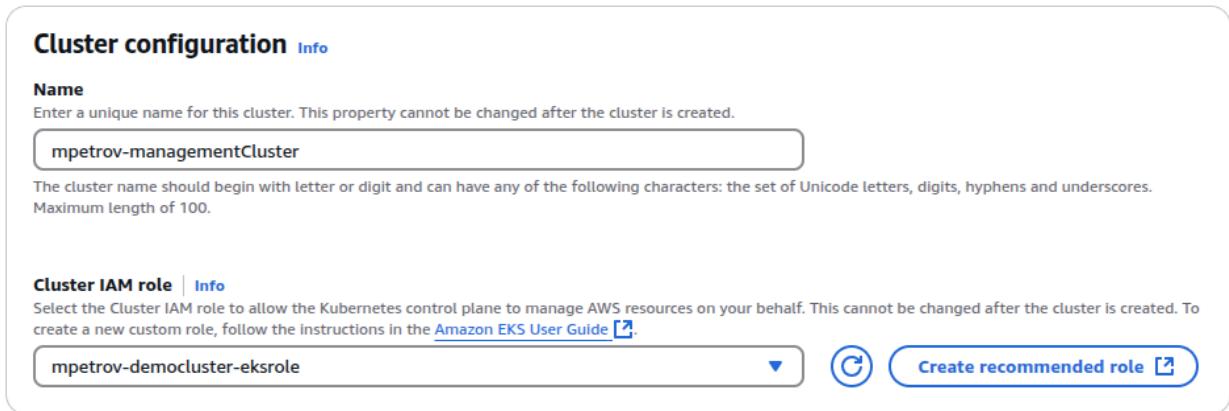
### 3.2. Изграждане на управляващ клъстор

След създаването на частна мрежа предстои изграждането на управляващия клъстор, върху който ще бъде инсталиран CAPI и неговите APIs.

През AWS конзолата, при услугата EKS, се избира “Create Cluster” (създаване на клъстор) бутонът, откъдето се отваря прозорец с конфигурационни опции за съответния клъстор. Използвайки персонализирана конфигурация за клъстера, се попълват име и IAM роля (Фиг. 3.4). Тази роля е задължително поле, което съдържа политики, които имат следните функции:

- AmazonEKSBlockStoragePolicy - позволява управление на блоковите хранилищни ресурси на клъстера.
- AmazonEKSClusterPolicy - дава на Kubernetes нужните правомощия, за да управлява ресурсите вместо нас.
- AmazonEKSComputePolicy - политика, която позволява управление на изчислителните ресурси на клъстера.

- AmazonEKSLoadBalancingPolicy - служи за управление на ресурсите, отговорни за балансирането на натоварването на кълстерите.
- AmazonEKSNetworkingPolicy - разрешава управлението на мрежовите ресурси на кълстера.



*Фиг. 3.4 Избор на име и роля за управляващия кълстер*

Следващата конфигурационна стъпка включва вече създадената виртуална мрежа. От полето за избор на VPC се задава новосъздадената частна мрежа, а подмрежите и се самоизбират (Фиг. 3.5). Освен това, EKS автоматично създава група за сигурност (security group) на кълстера при създаването му, за да улесни комуникацията между работните възли и контролната равнина. По желание може да се избират допълнителни групи за сигурност, които да бъдат приложени към EKS-управляваните мрежови интерфейси, които са създадени в подмрежите на контролния слой. При създаването на VPC, посредством CF, се създава и прилага такава група за сигурност, отделно от тази, която EKS ще създаде.

Също така, външният достъп до крайната точка (endpoint) на кълстера бива ограничен или напълно деактивиран. Тази крайна точка бива създадена от EKS за Kubernetes API сървъра, през който се комуникира с кълстера, с помощта на инструменти за управление на Kubernetes ресурсите като "kubectl". По подразбиране, тази крайна

точка на API е публична за интернет и достъпът до нея е защищен от IAM и RBAC (роден за Kubernetes механизъм за ограничаване на достъпа). Достъпът до сървъра може и да е частен, така че цялата комуникация между работните възли и него да остане вътрешна за мрежата на кълстера. Отделно, достъпът може и да се ограничи до IP адресите, които да достъпват API сървъра, или да бъде напълно деактивиран. В контекста на дипломната работа крайната точка до API сървъра е и публична, и частна, което позволява тя да е достъпна извън VPC-то на кълстера, но трафикът на работните машини към крайната точка да остане вътре в мрежата.

## Networking Info

IP address family and service IP address range cannot be changed after cluster creation.

### VPC Info

Select a VPC to use for your EKS cluster resources.

vpc-0c13c9550309ecbd5 | mpetrov-managementClusterVPC-VPC



### Subnets Info

Choose the subnets in your VPC where the control plane may place elastic network interfaces (ENIs) to facilitate communication with your cluster. To create a new subnet, go to the corresponding page in the [VPC console](#).

Select subnets



[Clear selected subnets](#)

subnet-0b8a802cb37581509 | mpetrov-managementClusterVPC-PrivateSubnet01 X  
eu-west-3a 192.168.128.0/18

subnet-087dc19447943a897 | mpetrov-managementClusterVPC-PublicSubnet01 X  
eu-west-3a 192.168.0.0/18

subnet-0935d8e72bbaabca0 | mpetrov-managementClusterVPC-PublicSubnet02 X  
eu-west-3b 192.168.64.0/18

subnet-0c139bef6ae32f6d9 | mpetrov-managementClusterVPC-PrivateSubnet02 X  
eu-west-3b 192.168.192.0/18

### Additional security groups - optional Info

EKS automatically creates a cluster security group on cluster creation to facilitate communication between worker nodes and control plane. Optionally, choose additional security groups to apply to the EKS-managed Elastic Network Interfaces that are created in your control plane subnets. To create a new security group, go to the corresponding page in the [VPC console](#).

Select security groups



[Clear selected security groups](#)

sg-094d1e51406aba641 | mpetrov-managementClusterVPC-  
ControlPlaneSecurityGroup-doPWvxpRUVwJ  
Cluster communication with worker nodes X

Фиг. 3.5 Конфигурация на мрежовите настройки

Изборът на допълнителните услуги, които да работят върху кълстера, е важна конфигурационна стъпка, която осигурява стр. 72/114

правилната му функционалност. По подразбиране, EKS избира няколко от т.нар. "add-ons" (добавки):

- kube-proxy мрежова услуга, която работи на всеки възел в клъстера на Kubernetes. Ролята ѝ е да отразява сървисите, дефинирани в клъстера, и да управлява мрежовите правила на възлите, за да позволи комуникация с обвивките, които поддържат сървиса.
- coreDNS - предоставя DNS услуги в EKS клъстери, което позволява на отделни контейнери лесно да откриват и да се свързват с други контейнери в рамките на клъстера.
- Amazon VPC CNI - EKS поддържа родни VPC мрежи с помощта на VPC CNI. С тази добавка Kubernetes обвивките могат да имат същия IP адрес както вътре в обвивката, така и във VPC мрежата. AWS препоръчва използването на тази добавка, когато работните възли за екземпляри са на Amazon EC2.
- Node monitoring agent - наблюдава регистрационните файлове (т.нар. logs) и открива проблеми с работните машини, с цел проучване на проблемите при необходимост.
- EKS Pod Identity Agent - използва се за предоставяне на AWS IAM разрешения на обвивки чрез сервисни акаунти (service accounts) на Kubernetes.

След попълването тези конфигурационни полета другите остават по подразбиране и клъстера бива създаден (Фиг. 3.6).

The screenshot shows the AWS EKS Clusters management interface. At the top, it displays 'Clusters (1)' with an 'Info' link. Below this is a search bar labeled 'Filter clusters'. A table lists the single cluster information:

| Cluster name              | Status | Kubernetes version               |
|---------------------------|--------|----------------------------------|
| mpetrov-managementCluster | Active | 1.31 <a href="#">Upgrade now</a> |

Фиг. 3.6 Създаден и активен AWS EKS клъстер

Новосъздаденият клъстър се нуждае от изчислителни ресурси, върху които да са разположени обвивките. За целта, в изчислителната секция (“compute”) на клъстера в AWS конзолата се добавя нова група възли (node group). Кръщава се и се поставя IAM роля за възлите. Тази роля се състои от следните политики:

- AmazonEC2ContainerRegistryReadOnly - осигурява достъп “само за четене” до хранилищата на AWS EC2 Container Registry.
- AmazonEKS\_CNI\_Policy - тази политика предоставя на Amazon VPC CNI добавката необходимите разрешения, нужни за модифициране на IP адресите на работните машини на клъстера, което позволява на CNI да изброява, описва и променя мрежовите интерфейси.
- AmazonEKSWorkerNodePolicy - разрешава на EKS работните машини да се свързват с EKS клъсторите.

Следва изборът на операционна система, тип машина и капацитет на диска на групата от машини. За операционна система стандартният “Amazon Linux 2023” е изборът по подразбиране и този, който ще бъде използван в рамките на дипломната работа. Отделно, тъй като работните приложения ще са върху клъсторите, които CAPI ще генерира впоследствие, самият управляващ клъстър не е необходимо да има силни изчислителни ресурси, затова възлите са от тип “t3.medium”, които имат 2 виртуални CPU и 4GB RAM памет, с по 20GB големина на диска и ще са 2 на брой (Фиг. 3.7).

## Node group compute configuration

These properties cannot be changed after the node group is created.

### AMI type Info

Select the EKS-optimized Amazon Machine Image for nodes.

Amazon Linux 2023 (x86\_64) Standard (AL2023\_x86\_64\_STANDARD)



### Capacity type

Select the capacity purchase option for this node group.

On-Demand



### Instance types Info

Select instance types you prefer for this node group.

Enter an instance type

t3.medium

vCPU: 2 vCPUs Memory: 4 GiB Network: Up to 5 Gigabit Max ENI: 3 Max IPs: 18



### Disk size

Select the size of the attached EBS volume for each node.

20

GiB

Фиг. 3.7 Конфигурация на изчислителна група

Следващата стъпка от конфигурацията на групата от машини е посочването на подмрежите в създадената по-рано виртуална мрежа (VPC), където те ще бъдат разположени. За целта се избират и 4-те от съществуващите подмрежи и като финална стъпка групата от възли бива създадена. Веднага щом групата стане “активна”, всяка една добавка под формата на обвивка, си намира място върху възел, който да я изпълнява.

**Nodes (2) Info**

| Node name                                    | Instance type | Compute    | Managed by                   | Created                  | Status |
|--|---------------|------------|------------------------------|--------------------------|--------|
| ip-192-168-136-17.eu-west-3.compute.internal | t3.medium     | Node group | mpetrov-managementCluster-ng | Created<br>3 minutes ago | Ready  |
| ip-192-168-78-119.eu-west-3.compute.internal | t3.medium     | Node group | mpetrov-managementCluster-ng | Created<br>3 minutes ago | Ready  |

**Node groups (1) Info**

Node groups implement basic compute scaling through EC2 Auto Scaling groups.

| Group name                   | Desired size | AMI release version | Launch template | Status |
|------------------------------|--------------|---------------------|-----------------|--------|
| mpetrov-managementCluster-ng | 2            | 1.31.4-20250123     | -               | Active |

Фиг. 3.8 Създадена и конфигурирана група от възли

Посредством изпълнението на тези стъпки, управляващият кълстер е конфигуриран, като следващата стъпка е инсталацията на CAPI върху него.

### 3.3. Инсталация на CAPI

Инсталацията на CAPI изискава няколко предпоставки. Необходима е конфигурацията на инструмента за командния ред на Kubernetes "kubectl", с цел комуникация с управляващия кълстер. За свързаността с кълстера в EKS обаче са нужни идентификационни ключове (т.нар. access keys), които удостоверяват самоличността на потребителя пред AWS. Това, от своя страна, позволява изпълнението на команди, посредством които да се променя инфраструктурата на AWS. Тези идентификационни ключове могат да бъдат открити в падащото меню на профила в AWS конзолата, под името "идентификационни данни за сигурност" (security credentials). Там се избира опцията за създаване на ключове за интерфейса за команден ред и тези ключове се изтеглят. Използването им изискава споменатия интерфейс за командния ред на AWS - AWS CLI. Необходимо е да бъде локално инсталиран и чрез командалата "aws configure", конфигуриран с идентификационния номер на ключа, както и самият ключ и регионът, в който е създаден управляващият кълстер (Фиг. 3.9).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ aws configure
AWS Access Key ID [*****IIME]: *****IIME
AWS Secret Access Key [*****iP0r]: *****iP0r
Default region name [eu-west-3]: eu-west-3
Default output format [None]:
```

Фиг. 3.9 Конфигурация на AWS CLI

Конфигурацията на kubectl - инструментът за команден ред на Kubernetes, е лесна стъпка, след като AWS CLI е настроен чрез командалата "aws eks --region (региона в който се намира управляващия кълстер) update-kubeconfig --name (името на кълстера)". Както се

забелязва на Фиг. 3.10, това променя текущия “контекст” в kubeconfig (конфигурационният файл на kubectl) на този на управляващия клъстер. Правилната конфигурация може да се тества с команда “kubectl get pods -A” например, която показва всички налични обвивки в клъстера. Ако в командния ред се покажат имената на обвивките на добавките (напр. coredns), “kubectl” е конфигуриран правилно и общува с управляващия клъстер. Контекстът в Kubernetes са група параметри за достъп, с които се определя с кой клъстер се взаимодейства, кой потребител се използва и в кое пространство (namespace) се работи.

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ aws eks --region eu-west-3 update-kubeconfig  
--name mpetrov-managementCluster  
Updated context arn:aws:eks:eu-west-3:722377226063:cluster/mpetrov-managementCluster  
in /home/mpetrov/.kube/config
```

Фиг. 3.10 Конфигурация на kubectl

Настройването на CAPI, от друга страна, изисква инсталацията на допълнителни инструменти за командния ред: clusterctl и clusterawsadm. Посредством clusterawsadm се създава CF стек за стартиране на CAPI, състоящ се от IAM роли и разрешения. Използването на този инструмент изисква администраторски акаунт в AWS, както и следните експортирани параметри в командния ред (Фиг. 3.11):

- AWS\_REGION - регион за създаване на съответния стек (трябва да съответства на региона, където се намира управляващият клъстер).
- AWS\_ACCESS\_KEY\_ID - идентификационен номер на ключа за достъп.
- AWS\_SECRET\_ACCESS\_KEY - ключът за достъп до AWS

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ export AWS_REGION=eu-west-3  
mpetrov@MishoXPS13:~/Documents/Diplomna$ export AWS_ACCESS_KEY_ID=[REDACTED] IME  
mpetrov@MishoXPS13:~/Documents/Diplomna$ export AWS_SECRET_ACCESS_KEY=[REDACTED]  
[REDACTED] iPOr
```

Фиг. 3.11 Експортиране на променливи на средата (*environment variables*)

В комбинация с тези параметри “clusterawsadm” се нуждае и от конфигурационен файл, с помощта на който да създаде и допълнителна IAM роля, която позволява създаването на CAPI ресурса “MachinePool” (машинни пулове). Този ресурс позволява управлението на множество машини като един обект. CAPI има експериментална поддръжка за EKS управлявани групи от възли (Managed Node Groups), чрез “MachinePool”, използвайки инфраструктурния тип “AWSManagedMachinePool”. Контролерът “AWSManagedMachinePool” създава и управлява “EKS Managed Node Group”, която, от своя страна, контролира “AWS AutoScaling Group” (група за автоматично мащабиране на възли) от управлявани EC2 инстанции. За да се използват тези машинни пулове, са необходими определени IAM разрешения, а най-лесният начин да се гарантира, че те са налични, е чрез “clusterawsadm”, който автоматично ги създава. Използването на управлявани машинни пулове (AWSManagedMachinePool) предоставя по-интуитивен и автоматизиран начин за управление на работните възли, като намалява сложността, подобрява мащабируемостта и автоматизира поддръжката. Създаването на допълнителната роля за машинните пулове е възможно посредством конфигурационния файл на Фиг. 3.12, където флагът “disable: false” позволява създаването на ролята.

```

mpetrov@MishoXPS13:~/Documents/Diplomna$ cat config-bootstrap.yaml
apiVersion: bootstrap.aws.infrastructure.cluster.x-k8s.io/v1beta1
kind: AWSIAMConfiguration
spec:
  eks:
    iamRoleCreation: false
    managedMachinePool:
      disable: false

```

*Фиг. 3.12 Конфигурационен файл за създаване на доп. IAM роля*

Следва създаването на CF стека, посредством конзолния ред. С помощта на командалата “clusterawsadm bootstrap iam create-cloudformation-stack --config config-bootstrap.yaml”, където “config-bootstrap.yaml” е конфигурационният файл от Фиг. 3.12., стекът бива създаден (Фиг. 3.13).

```

mpetrov@MishoXPS13:~/Documents/Diplomna$ clusterawsadm bootstrap iam create-cloudformation
-stack --config config-bootstrap.yaml
Attempting to create AWS CloudFormation stack cluster-api-provider-aws-sigs-k8s-io

Following resources are in the stack:

Resource           |Type
                   |Status
AWS::IAM::InstanceProfile |control-plane.cluster-api-provider-aws.sigs.k8s.io
                           |CREATE_COMPLETE
AWS::IAM::InstanceProfile |controllers.cluster-api-provider-aws.sigs.k8s.io
                           |CREATE_COMPLETE
AWS::IAM::InstanceProfile |nodes.cluster-api-provider-aws.sigs.k8s.io
                           |CREATE_COMPLETE
AWS::IAM::ManagedPolicy |arn:aws:iam::722377226063:policy/control-plane.cluster-api-prov
ider-aws.sigs.k8s.io   |CREATE_COMPLETE
AWS::IAM::ManagedPolicy |arn:aws:iam::722377226063:policy/nodes.cluster-api-provider-aws
.sigs.k8s.io          |CREATE_COMPLETE
AWS::IAM::ManagedPolicy |arn:aws:iam::722377226063:policy/controllers.cluster-api-provid
er-aws.sigs.k8s.io    |CREATE_COMPLETE
AWS::IAM::ManagedPolicy |arn:aws:iam::722377226063:policy/controllers-eks.cluster-api-pr
ovider-aws.sigs.k8s.io|CREATE_COMPLETE
AWS::IAM::Role         |control-plane.cluster-api-provider-aws.sigs.k8s.io
                           |CREATE_COMPLETE
AWS::IAM::Role          |controllers.cluster-api-provider-aws.sigs.k8s.io
                           |CREATE_COMPLETE
AWS::IAM::Role          |eks-controlplane.cluster-api-provider-aws.sigs.k8s.io
                           |CREATE_COMPLETE
AWS::IAM::Role          |eks-nodegroup.cluster-api-provider-aws.sigs.k8s.io
                           |CREATE_COMPLETE
AWS::IAM::Role          |nodes.cluster-api-provider-aws.sigs.k8s.io
                           |CREATE_COMPLETE

```

*Фиг. 3.13 Резултат от изпълнението на командата за създаване на CF стек в командиния ред*

Освен потвърждението, че стека е създаден в командния ред, това може да се забележи и в AWS конзолата, където се виждат и отделните роли, създадени от “clusterawsadm” (Фиг. 3.14).

| Resources (12)                               |   |                           |                 |
|--|---|---------------------------|-----------------|
| Logical ID                                   | Physical ID   | Type                      | Status          |
| AWSIAMInstanceProfileControllers             | controllers.cluster-api-provider-aws.sigs.k8s.io  | AWS::IAM::InstanceProfile | CREATE_COMPLETE |
| AWSIAMInstanceProfileControlPlane            | control-plane.cluster-api-provider-aws.sigs.k8s.io  | AWS::IAM::InstanceProfile | CREATE_COMPLETE |
| AWSIAMInstanceProfileNodes                   | nodes.cluster-api-provider-aws.sigs.k8s.io  | AWS::IAM::InstanceProfile | CREATE_COMPLETE |
| AWSIAMManagedPolicyCloudProviderControlPlane | <a href="#">arn:aws:iam::722377226063:policy/control-plane.cluster-api-provider-aws.sigs.k8s.io</a>   | AWS::IAM::ManagedPolicy   | CREATE_COMPLETE |
| AWSIAMManagedPolicyCloudProviderNodes        | <a href="#">arn:aws:iam::722377226063:policy/nodes.cluster-api-provider-aws.sigs.k8s.io</a>           | AWS::IAM::ManagedPolicy   | CREATE_COMPLETE |
| AWSIAMManagedPolicyControllers               | <a href="#">arn:aws:iam::722377226063:policy/controllers.cluster-api-provider-aws.sigs.k8s.io</a>     | AWS::IAM::ManagedPolicy   | CREATE_COMPLETE |
| AWSIAMManagedPolicyControllersEKS            | <a href="#">arn:aws:iam::722377226063:policy/controllers-eks.cluster-api-provider-aws.sigs.k8s.io</a> | AWS::IAM::ManagedPolicy   | CREATE_COMPLETE |
| AWSIAMRoleControllers                        | <a href="#">controllers.cluster-api-provider-aws.sigs.k8s.io</a>                                      | AWS::IAM::Role            | CREATE_COMPLETE |
| AWSIAMRoleControlPlane                       | <a href="#">control-plane.cluster-api-provider-aws.sigs.k8s.io</a>                                    | AWS::IAM::Role            | CREATE_COMPLETE |
| AWSIAMRoleEKSClusterPlane                    | <a href="#">eks-controlplane.cluster-api-provider-aws.sigs.k8s.io</a>                                 | AWS::IAM::Role            | CREATE_COMPLETE |
| AWSIAMRoleEKSNodegroup                       | <a href="#">eks-nodegroup.cluster-api-provider-aws.sigs.k8s.io</a>                                    | AWS::IAM::Role            | CREATE_COMPLETE |
| AWSIAMRoleNodes                              | <a href="#">nodes.cluster-api-provider-aws.sigs.k8s.io</a>  | AWS::IAM::Role            | CREATE_COMPLETE |

Фиг. 3.14 Създадените IAM роли, с помощта на *clusterawsadm*, изобразени в AWS конзолата

Всяка една от тези роли отговаря за безпогрешната работа на CAPI, когато той бъде инсталиран върху управляващия клъстер.

Като финална стъпка от инсталацията на CAPI, са допълнителни променливи на средата, които трябва да бъдат експортирани. “EXP\_MACHINE\_POOL” е променлива, която позволява поддръжката на машинни пулове. Флагът на този параметър, отговарящ за “MachinePool” обектите, трябва да бъде зададен на “true”. По подразбиране, EKS клъстерите използват едни и същи IAM роли (т.е. контролен слой, роли на група възли). Има функция, която позволява

на всеки кълъстер да има свои собствени IAM роли. Това става чрез активиране на флага на функцията “EKSEnableIAM”. Това може да се направи, преди стартиране на “clusterctl init” (командата за инициализация на CAPI), чрез променливата на средата “CAPA\_EKS\_IAM”, с флаг “true”. Също така, с помощта на “clusterawsadm”, се създават “base64” кодирани идентификационни данни. Използва вече експортирани променливи на средата и ги кодира в стойност, която да се съхранява в Kubernetes секретен (Secret) параметър “AWS\_B64ENCODED\_CREDENTIALS”. Всички променливи, експортирани в средата, са демонстрирани на Фиг. 3.15.

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ export AWS_B64ENCODED_CREDENTIALS=$(clusterawsadm bootstrap credentials encode-as-profile)
WARNING: `encode-as-profile` should only be used for bootstrapping.

mpetrov@MishoXPS13:~/Documents/Diplomna$ export EXP_MACHINE_POOL=true
mpetrov@MishoXPS13:~/Documents/Diplomna$ export CAPA_EKS_IAM=true
```

*Фиг. 3.15 Експортирани променливи на средата необходими за инсталацията на CAPI*

Накрая се инициализира управляващият кълъстер, посредством “clusterctl init --infrastructure aws”, където се назовава и инфраструктурата, в която се намира кълъстерът, в случая AWS (Фиг. 3.16). Важно е да се отбележи, че “clusterctl” използва текущата конфигурация на “kubectl” (инструмента за команден ред на Kubernetes) за инициализацията на CAPI.

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ clusterctl init --infrastructure aws
Fetching providers
Installing cert-manager version="v1.16.1"
Waiting for cert-manager to be available...
Installing provider="cluster-api" version="v1.9.4" targetNamespace="capi-system"
Installing provider="bootstrap-kubeadm" version="v1.9.4" targetNamespace="capi-kubeadm-bootstrap-system"
Installing provider="control-plane-kubeadm" version="v1.9.4" targetNamespace="capi-kubeadm-control-plane-system"
Installing provider="infrastructure-aws" version="v2.7.1" targetNamespace="capa-system"
[KubeAPIWarningLogger] spec.template.spec.affinity.preferredDuringSchedulingIgnoredDuringExecution[1].preference.matchExpressions[0].key: node-role.kubernetes.io/master is use "node-role.kubernetes.io/control-plane" instead

Your management cluster has been initialized successfully!

You can now create your first workload cluster by running the following:

  clusterctl generate cluster [name] --kubernetes-version [version] | kubectl apply -f -
```

*Фиг. 3.16 Успешна инициализация на управляващ кълъстер*

В AWS конзолата, в ресурсите на управляващия клъстър, могат да се забележат новосъздадените обвивки (Фиг. 3.17), които се грижат за функционалността на CAPI.

|   |  |                                 |
|---|--|---------------------------------|
| ○ | <a href="#">capa-controller-manager-567d6d57c8-qlbc6</a>                       | <b>Created</b><br>4 minutes ago |
| ○ | <a href="#">capi-controller-manager-59c7f9c475-7cdl2</a>                       | <b>Created</b><br>4 minutes ago |
| ○ | <a href="#">capi-kubeadm-bootstrap-controller-manager-676545ff7c-z4gxm</a>     | <b>Created</b><br>4 minutes ago |
| ○ | <a href="#">capi-kubeadm-control-plane-controller-manager-656f587d76-kjzz9</a> | <b>Created</b><br>4 minutes ago |
| ○ | <a href="#">cert-manager-5c887c889d-4cgzv</a>                                  | <b>Created</b><br>5 minutes ago |
| ○ | <a href="#">cert-manager-cainjector-58f6855565-rxk8d</a>                       | <b>Created</b><br>5 minutes ago |
| ○ | <a href="#">cert-manager-webhook-6647d6545d-cckcr</a>                          | <b>Created</b><br>5 minutes ago |

Фиг. 3.17 Новосъздадените обвивки за контролерите на CAPI

### 3.4. Създаване на допълнителни клъстери чрез CAPI

За да бъдат създадени нови клъстери, те първо трябва да бъдат генериирани. CAPI предоставя начин за генериране на нови клъстери, спрямо т.нар “flavor” (вид) на клъстера. Този вид, в комбинация с експортираните променливи на средата, се използват за избиране на шаблон, по който да бъдат създавани клъстерите. Типът на машините, които ще бъдат създадени при изпълнението на конфигурационния файл на новите клъстери, също трябва да бъде упоменат, под формата на променливи на средата: “AWS\_CONTROL\_PLANE\_MACHINE\_TYPE” и “AWS\_NODE\_MACHINE\_TYPE”, съответно типа на машините на контролния и на работния слой. В контекста на дипломната работа, тъй като работните приложения не изискват голяма изчислителна

мощност, машините ще бъдат от същия тип като управляващия кълстер, а именно “t3.medium” (Фиг. 3.18).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ export AWS_CONTROL_PLANE_MACHINE_TYPE=t3.medium
mpetrov@MishoXPS13:~/Documents/Diplomna$ export AWS_NODE_MACHINE_TYPE=t3.medium
```

*Фиг. 3.18 Експортиране на променливи на средата за типа на машините*

Чрез командалата “clusterctl generate cluster mpetrov-capitesteks --flavor eks-managedmachinepool --kubernetes-version v1.30.5 --control-plane-machine-count=2 --worker-machine-count=2 > wordpressEks.yaml” се генерира кълстер от вида “eks-managedmachinepool” (управлявани машинни пулове в EKS), състоящ се от 2 машини на контролния слой и две на работния. Файлът, който тази команда генерира, е демонстриран на Фиг. 3.19. Там се забелязват основните ресурси, които “clusterctl” генерира, като например “Cluster”, “AWSManagedControlPlane”, “AWSManagedCluster”, “MachinePool” и “AWSManagedMachinePool”. Обектът “Cluster” дефинира кълстера като цяло. Параметърът му - “controlPlaneRef”, отговаря за управлението на контролния слой чрез EKS, като се позовава на “AWSManagedControlPlane”, докато “infrastructureRef” дефинира инфраструктурните настройки, с помощта на “AWSManagedCluster” обекта. Целта му е да дефинира основните параметри на кълстера, но реалната инстанция на този Kubernetes кълстер е EKS. Този кълстер в EKS се дефинира от “AWSManagedControlPlane” (EKS контролен слой). Той има различни параметри като “eksClusterName” - името на кълстера в AWS EKS конзолата, “region” - регионът, където ще бъде създаден кълстерът, “sshKeyName” - определя SSH ключ, който може да се използва за достъп до възлите при необходимост, и “version” - версията на Kubernetes кълстера. Друг ресурс - “MachinePool”, който управлява “EKS Managed Node Group” (управлявана група от възли в EKS),

създава две реплики на работните възли, използвайки "AWSManagedMachinePool" за дефиниция на инфраструктурния слой за тези възли. Всъщност, точно чрез "AWSManagedMachinePool" ресурса, възлите биват дефинирани като "EKS Managed Node Group", което от своя страна позволява автоматичното мащабиране на възли с помощта на AWS Auto Scaling Groups.

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ cat wordpressEKS.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: mpetrov-wordpresscluster
  namespace: default
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.0.0/16
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta2
    kind: AWSManagedControlPlane
    name: mpetrov-wordpresscluster-control-plane
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta2
    kind: AWSManagedCluster
    name: mpetrov-wordpresscluster
  ---
  apiVersion: infrastructure.cluster.x-k8s.io/v1beta2
  kind: AWSManagedCluster
  metadata:
    name: mpetrov-wordpresscluster
    namespace: default
  spec: {}
  ---
  apiVersion: controlplane.cluster.x-k8s.io/v1beta2
  kind: AWSManagedControlPlane
  metadata:
    name: mpetrov-wordpresscluster-control-plane
    namespace: default
  spec:
    eksClusterName: mpetrov-wordpressCluster
    region: eu-west-3
    sshKeyName: capi-eks
    version: v1.30.5
  network:
    vpc:
      availabilityZoneUsageLimit: 2
      availabilityZoneSelection: Random
  ---
  apiVersion: cluster.x-k8s.io/v1beta1
  kind: MachinePool
  metadata:
    name: mpetrov-wordpresscluster-pool-0
    namespace: default
  spec:
    clusterName: mpetrov-wordpresscluster
    replicas: 2
    template:
      spec:
        bootstrap:
          dataSecretName: ""
          clusterName: mpetrov-wordpresscluster
        infrastructureRef:
          apiVersion: infrastructure.cluster.x-k8s.io/v1beta2
          kind: AWSManagedMachinePool
          name: mpetrov-wordpresscluster-pool-0
  ---
  apiVersion: infrastructure.cluster.x-k8s.io/v1beta2
  kind: AWSManagedMachinePool
  metadata:
    name: mpetrov-wordpresscluster-pool-0
    namespace: default
  spec: {}
```

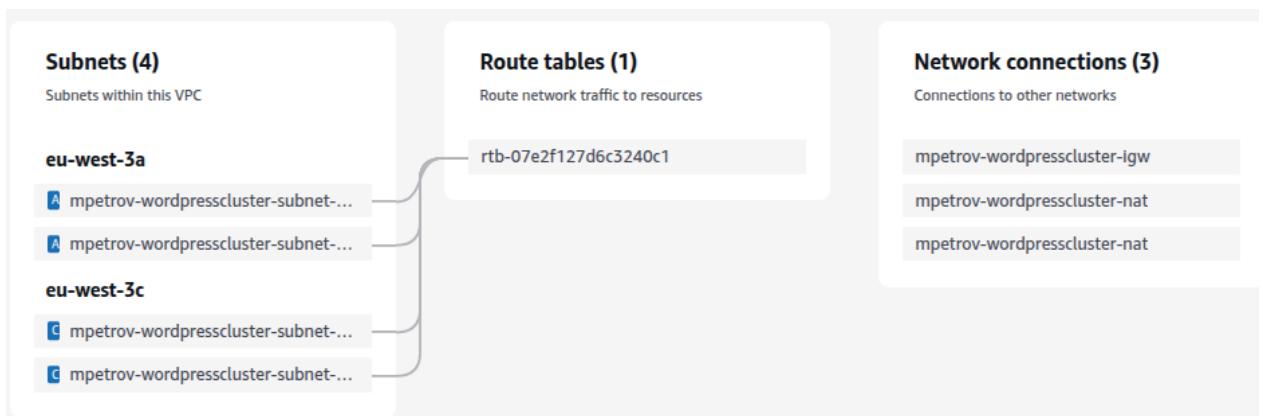
Фиг. 3.19 Шаблон за създаване на допълнителен кълстмер

Освен генерирането на шаблона за кълстера, той трява да бъде приложен. Уверявайки се, че контекстът на “kubectl” все още е насочен към управляващия кълстер, посредством “kubectl apply -f {име на файла}”, той се прилага (Фиг. 3.20).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ kubectl apply -f wordpressEKS.yaml
cluster.cluster.x-k8s.io/mpetrov-wordpresscluster created
awsmanagedcluster.infrastructure.cluster.x-k8s.io/mpetrov-wordpresscluster created
awsmanagedcontrolplane.controlplane.cluster.x-k8s.io/mpetrov-wordpresscluster-control-plane created
machinepool.cluster.x-k8s.io/mpetrov-wordpresscluster-pool-0 created
awsmanagedmachinepool.infrastructure.cluster.x-k8s.io/mpetrov-wordpresscluster-pool-0 created
```

Фиг. 3.20 Прилагане на шаблон за wordpress кълстер

След това CAPI се грижи за надигането на цялостната инфраструктура, състояща се от самия кълстер, отделна негова виртуална мрежа, върху която кълстера да разположи ресурсите си, и работните възли. По подразбиране, CAPI създава 6 подмрежи, разположени във всяка зона на достъпност на регион, в който кълстерът е създаден, но тъй като в конфигурацията са ограничени на 2 зони, чрез параметъра “availabilityZoneUsageLimit”, се създават само 4 подмрежи - 2 публични и 2 частни, с цел оптимизиране на използването на наличните ресурси (Фиг. 3.21).



Фиг. 3.21 Ресурсна карта на VPC на допълнителен кълстер

Всички ресурси, конфигурирани от дадения шаблон, са за къмпинга, където ще е разположено WordPress приложението.

Другото работно приложение - Grafana, ще бъде върху отделен къмпинг, отново създаден посредством инструмента за команден ред на CAPI - `clusterctl`. Генерира се със същите параметри като този за WordPress, но с различно име. Работните възли остават от същия тип "t3.medium", бройката им отново е 2 и Kubernetes версията остава 1.30.5. Grafana къмпингът отново генерира своя собствена частна мрежа, ограничена на 2 зони за достъпност. Общата конфигурация на двата къмпинга е почти еднаква и е демонстрирана на Фиг. 3.19, единствената разлика са имената на генерираните ресурси.

След като YAML конфигурационният файл е готов, трябва да бъде приложен. Отново е важно "kubectl" да е с правилна конфигурация (за управляващия къмпинг) преди изпълнението на команда за прилагане на YAML файла, а именно "`kubectl apply -f {име на файла}`" (Фиг. 3.22).

```
mpetrov@MishoXPS13:~/Documents/Diploma$ kubectl apply -f grafanaEKS.yaml
cluster.cluster.x-k8s.io/mpetrov-grafanacluster created
awsmanagedcluster.infrastructure.cluster.x-k8s.io/mpetrov-grafanacluster created
awsmanagedcontrolplane.controlplane.cluster.x-k8s.io/mpetrov-grafanacluster-control-plane created
machinepool.cluster.x-k8s.io/mpetrov-grafanacluster-pool-0 created
awsmanagedmachinepool.infrastructure.cluster.x-k8s.io/mpetrov-grafanacluster-pool-0 created
```

Фиг. 3.22 Прилагане на шаблон за grafana къмпинг

Състоянието на новосъздадените къмпингове може да се провери чрез "`clusterctl describe cluster {име на къмпинга}`", където флагът "READY" описва състоянието на къмпинговете, или през AWS конзолата, където могат ръчно да се провери дали всички ресурси са били създадени (в случая "TRUE", което означава, че са създадени/конфигурирани правилно) (Фиг. 3.23). След създаването на Grafana къмпинга и успешното прилагане на конфигурацията, могат да се направят допълнителни настройки за свързване с база данни или

други мониторингови инструменти, които биха могли да бъдат интегрирани с работната среда. Grafana обикновено се използва в комбинация с инструменти като Prometheus за събиране и съхранение на метрични данни, но въпреки това настройка и използването на този инструмент не са обект на тази дипломна работа, тъй като основният фокус е върху демонстрирането на възможностите на CAPI за автоматизирано управление на Kubernetes кълстери.

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ clusterctl describe cluster mpetrov-grafanacluster
NAME                                     READY  SEVERITY
REASON SINCE MESSAGE
Cluster/mpetrov-grafanacluster          True
  23m
  |-ClusterInfrastructure - AWSManagedCluster/mpetrov-grafanacluster
  |-ControlPlane - AWSManagedControlPlane/mpetrov-grafanacluster-control-plane True
    23m
  |-Workers
    |-MachinePool/mpetrov-grafanacluster-pool-0      True
      21m
mpetrov@MishoXPS13:~/Documents/Diplomna$ clusterctl describe cluster mpetrov-wordpresscluster
NAME                                     READY  SEVERITY
REASON SINCE MESSAGE
Cluster/mpetrov-wordpresscluster        True
  5h14m
  |-ClusterInfrastructure - AWSManagedCluster/mpetrov-wordpresscluster
  |-ControlPlane - AWSManagedControlPlane/mpetrov-wordpresscluster-control-plane True
    5h14m
  |-Workers
    |-MachinePool/mpetrov-wordpresscluster-pool-0      True
      5h12m
```

Фиг. 3.23 Състояние на новосъздадените кълстери през *clusterctl*

Освен през командния ред, AWS конзолата също показва състоянието на създадените EKS кълстери, включително техните възли, мрежови настройки и свързаните ресурси - VPC, NAT и интернет шлюзове (Фиг. 3.24).

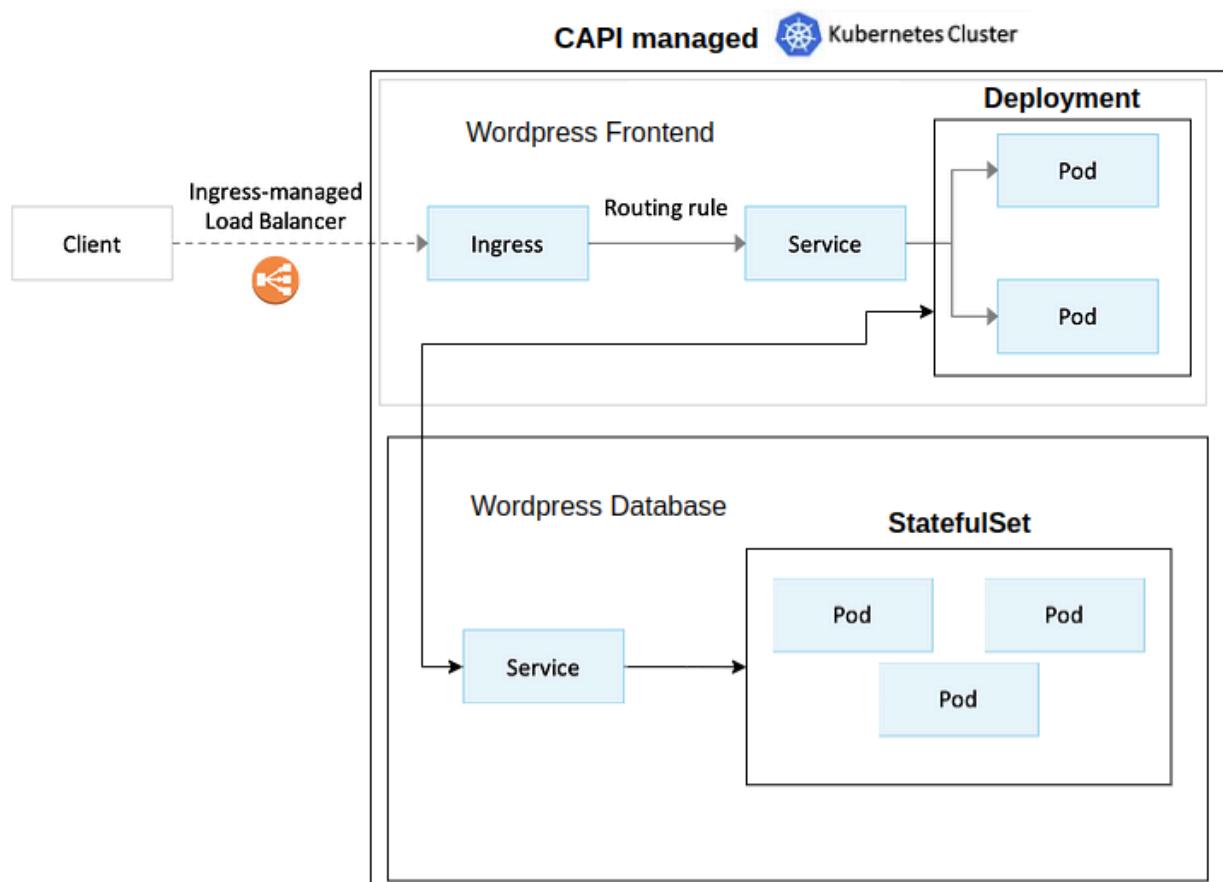
| Clusters (3) <a href="#">Info</a>    |   |
|--------------------------------------|---|
| <input type="text"/> Filter clusters |   |
|                                      | Cluster name ▲   Status ▼   |
| <input type="radio"/>                | <a href="#">mpetrov-grafanaCluster</a> <span style="color: green;">Active</span>    |
| <input type="radio"/>                | <a href="#">mpetrov-managementCluster</a> <span style="color: green;">Active</span> |
| <input type="radio"/>                | <a href="#">mpetrov-wordpressCluster</a> <span style="color: green;">Active</span>  |

Фиг. 3.24 Състояние на всички кълстери през AWS конзолата

След успешното конфигуриране на допълнителните Kubernetes кълстери чрез CAPI, следващата стъпка е разгръщането на работните приложения – WordPress с MySQL база данни и Grafana, използвайки Helm. Те ще бъдат разположени върху създадените кълстери, като ще използват наличната инфраструктура и необходимите конфигурации, за да се демонстрират интеграцията на CAPI с реални работни натоварвания.

### 3.5. Конфигуриране и разполагане на работните приложения

На Фиг. 3.25 е демонстрирана блоковата схема на WordPress работното приложение и интеракцията на клиента с кълстера. Посредством AWS, балансърът на натоварването (Application Load Balancer), управляем от ингрес Kubernetes ресурса, клиентът осъществява достъп до WordPress интерфейса.



Фиг. 3.25 Блокова схема на wordpress работно приложение

На Фиг. 3.25 се забелязва входящият трафик, който първоначално преминава през дадения ингрес ресурс, разпределящ и пренасочващ заявките към Kubernetes сървис, който от своя страна отговаря за насочването им към отделните обвивки в Deployment обекта.

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-service
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: wordpress
    tier: frontend

---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: wordpress-ingress
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: wordpress-service
                port:
                  number: 80
```

Фиг. 3.26 YAML конфигурация на ингрес и сървис за *wordpress* приложението.

Kubernetes ресурсите на Фиг. 3.26 дефинират начина, по който WordPress трафикът да бъде пренасочен в рамките на клъстера.

- Сървисът (Service “wordpress-service”) действа като вътрешна точка за достъп до WordPress обвивките, осигурявайки комуникация с тези от тях, които отговарят на критериите на селектора, а именно “app: wordpress” и “tier: frontend”. Определя

и че заявките, получени на порт 80, ще бъдат пренасочени вътрешно в контейнера на същия порт.

- Ингресът (Ingress "wordpress-ingress") дефинира външния достъп до приложението чрез AWS балансър на натоварването на приложенията (Application Load Balancer), като задава интернет-достъпен балансър, който приема и обработва HTTP заявки, съответстващи на "/", или т.нар. "коренов път", и ги пренасочва към порт 80 на сървис ресурса.

Използвайки този начин на организация на ресурсите се гарантира, че клиентите достъпват приложението през интернет, за което се грижи ингрес обектът, а вътрешната комуникация между ресурсите е осигурена от сървиса.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  replicas: 2
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
        - image: wordpress:6.2.1-apache
          name: wordpress
          env:
            - name: WORDPRESS_DB_HOST
              value: wordpress-mysql
            - name: WORDPRESS_DB_PASSWORD
              value: password
            - name: WORDPRESS_DB_USER
              value: wordpress
        ports:
          - containerPort: 80
            name: wordpress
```

Фиг. 3.27 Конфигурация на Deployment за wordpress приложението

Заедно с ингреса и сървиса, Deployment обектът (Фиг. 3.27) декларира управлението на жизнения цикъл и автоматичното мащабиране на приложението. Освен това, той създава две реплики/обвивки, които осигуряват по-добра достъпност до къмъстера, а селекторът гарантира, че Deployment управлява само обвивки с етикети "app: wordpress" и "tier: frontend". Самата конфигурация на контейнера - "wordpress:6.2.1-apache", което означава, че контейнерът е базиран на версия 6.2.1 с Apache, се състои от няколко променливи, които WordPress изисква:

- WORDPRESS\_DB\_HOST - определя MySQL сървъра, към който интерфейсът да се свърже, в случая - името на сървиса на базата данни ("wordpress-mysql").
- WORDPRESS\_DB\_USER и WORDPRESS\_DB\_PASSWORD - определят идентификационните данни за достъп до MySQL.
- Ports - дефинира порта, който ще "слуша" за заявки на порт 80 - стандартният HTTP порт за WordPress.

Другата част на приложението се състои от гореспоменатата MySQL база данни.

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
  type: ClusterIP
```

Фиг. 3.28 Сървис на MySQL базата данни

Сървисът на Фиг. 3.28 е достъпен само вътрешно в Kubernetes кълъстера, без външен достъп, а WordPress може да се свързва с базата данни, чрез DNS името му - “wordpress-mysql”. Конфигуриран е да използва стандартния порт за MySQL - 3306, през който WordPress ще осигурява връзка. Също така, сървисът ще пренасочва трафика към обвивки, които имат етикети, съответстващи на “app: wordpress” и “tier: mysql”.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  serviceName: wordpress-mysql
  replicas: 3
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
        - image: mysql:8.0
          name: mysql
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: password
            - name: MYSQL_DATABASE
              value: wordpress
            - name: MYSQL_USER
              value: wordpress
            - name: MYSQL_PASSWORD
              value: password
          ports:
            - containerPort: 3306
              name: mysql
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
  volumeClaimTemplates:
    - metadata:
        name: mysql-persistent-storage
        labels:
          app: wordpress
    spec:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 1Gi
      storageClassName: gp2
```

Фиг. 3.29 StatefulSet за WordPress базата данни

За базата данни, освен сървис, има и StatefulSet (Фиг. 3.29), който създава уникални идентификатори за всяка MySQL обвивка (напр. wordpress-mysql-0, wordpress-mysql-1) и 3 реплики за по-добра устойчивост. Чрез параметърът “serviceName” се определя името на сървиса, който ще осъществява връзката с този StatefulSet и осигурява вътрешния достъп между компонените в клъстера. Самата конфигурация на контейнера за базата използва официалния MySQL 8.0 образ. За нея, също както за конфигурацията на WordPress интерфейса, са необходими някои променливи:

- MYSQL\_DATABASE - името на самата база данни (“wordpress”).
- MYSQL\_USER и MYSQL\_PASSWORD - определят достъпа за WordPress приложението.
- MYSQL\_ROOT\_PASSWORD - администраторската парола.

StatefulSet използва заявки за хранилища (PVC), за да осигури устойчивостта на данните. Всяка обвивка има отделен диск (PV), така че ако дадена обвивка се премести на друг възел, или бъде пресъздадена, тя да запази данните си. Дисковете им използват 1GiB памет и са монтирани в директорията “/var/lib/mysql” в контейнера. Освен това използват StorageClass - “gp2”, който е стандартен “Elastic Block Storage” (EBS, еластично блоково хранилище) диск в AWS.

Разполагането на WordPress работното приложение изисква работещ клъстер. Текущата конфигурация на “kubectl” трябва да бъде заменена с тази на новия клъстер. Посредством командата “aws eks –region eu-west-3 update-kubeconfig –name myetrov-wordpressCluster”, текущият контекст се променя и командите, изпълнявани чрез “kubectl”, са насочени към новосъздадения клъстер, където ще бъде разположено приложението.

Преди да бъдат приложени конфигурациите на работното приложение, трябва да бъдат изпълнени няколко предпоставки. Една

от тях е инсталацирането на контролер, който да управлява създаването и асоциирането на балансьори на натоварването. При използване на ингрес ресурс в EKS кълстър, AWS автоматично осигурява Application Load Balancer (ALB). Той се използва с работни възли или AWS Fargate машини и се разполага както в публични, така и в частни мрежи. Трафикът на приложението (application traffic) се балансира на L7 (приложния слой) от OSI модела. Разликата между ALB и AWS Network Load Balancer (NLB) е, че NLB балансира натоварването на транспортния слой (L4) и за него се използва Kubernetes сървис от тип "LoadBalancer".

За инсталацията на контролера трябва да бъде създаден OIDC (OpenID Connect) доставчик, който се използва за предоставяне на IAM роли на т.нар. "service accounts" в Kubernetes. Това са идентичности, които обвивките и различните ресурси в Kubernetes използват за да се удостоверяват пред Kubernetes API сървъра. По този начин се позволява прилагането на роли, разрешения и политики за сигурност например. За създаването на OIDC доставчик се използва инструментът за команден ред на AWS EKS - eksctl. Задава се име на кълстъра като променлива в текущата сесия на терминала, взима се OIDC идентификационния номер, проверява се дали такъв OIDC доставчик вече не съществува и се създава (Фиг. 3.30).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ cluster_name=mpetrov-wordpressCluster
mpetrov@MishoXPS13:~/Documents/Diplomna$ oidc_id=$(aws eks describe-cluster --name $cluster_name --query
"cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
mpetrov@MishoXPS13:~/Documents/Diplomna$ aws iam list-open-id-connect-providers | grep $oidc_id | cut -d
"/" -f4
mpetrov@MishoXPS13:~/Documents/Diplomna$ eksctl utils associate-iam-oidc-provider --cluster $cluster_name
--approve
[?] will create IAM Open ID Connect provider for cluster "mpetrov-wordpressCluster"
in "eu-west-3"
[✓] created IAM Open ID Connect provider for cluster "mpetrov-wordpressCluster" in "eu-west-3"
```

Фиг. 3.30 Създаване на OIDC доставчик за WordPress кълстър

Има и друга предпоставка за инсталацията на контролера на балансьорите на натоварването и това е IAM ролята, която

контролерът ще използва. Тя се създава в комбинация със стандартната политика на AWS за балансьорите на натоварването (EKSLoadBalancingPolicy), но и с друга доверителна политика - "Trusted entities policy", която позволява на новосъздадения OIDC доставчик да използва тази IAM роля. Обобщено, позволява на "service account" на контролера "aws-load-balancer-controller" да използва IAM ролята, удостоверявайки се чрез OIDC в AWS EKS (Фиг 3.31).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Federated": "arn:aws:iam::722377226063:oidc-provider/oidc.eks.eu-west-3.amazonaws.com/id/743C809324363A93DD1A83899C188250"  
            },  
            "Action": "sts:AssumeRoleWithWebIdentity",  
            "Condition": {  
                "StringEquals": {  
                    "oidc.eks.eu-west-3.amazonaws.com/id/743C809324363A93DD1A83899C188250:aud":  
                        "sts.amazonaws.com",  
                    "oidc.eks.eu-west-3.amazonaws.com/id/743C809324363A93DD1A83899C188250:sub":  
                        "system:serviceaccount:kube-system:aws-load-balancer-controller"  
                }  
            }  
        }  
    ]  
}
```

Фиг. 3.31 Доверителна политика за контролер за балансъор на натоварването

След създаването на дадената политика, контролерът може да бъде инсталиран. Това се случва посредством мениджъра на пакети за Kubernetes - helm. Той улеснява инсталацията, управлението и актуализацията на приложения в клъстера. Използва се в комбинация с Helm Charts, които представляват шаблони с конфигурации за Kubernetes ресурси. Процесът на инсталiranе на този контролер започва с инсталацията на тези Helm Charts за EKS и актуализацията им. Чак тогава той се инсталира (Фиг. 3.32).

```

mpetrov@MishoXPS13:~/Documents/Diplomna$ helm repo add eks https://aws.github.io/eks-charts

"eks" already exists with the same configuration, skipping
mpetrov@MishoXPS13:~/Documents/Diplomna$ helm repo update eks
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "eks" chart repository
Update Complete. *Happy Helm-ing!*
mpetrov@MishoXPS13:~/Documents/Diplomna$ helm install aws-load-balancer-controller eks/aws-load-balancer-
-controller -n kube-system --set clusterName=mpetrov-wordpressCluster --set serviceAccount.create=
true --set serviceAccount.name=aws-load-balancer-controller --set serviceAccount.annotations."eks\amazonaws\.com/role-arn"="arn:aws:iam::722377226063:role/mpetrov_stsAssumeRoleWithWebIdentity_eksSAalb"
NAME: aws-load-balancer-controller
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
AWS Load Balancer controller installed!

```

*Фиг. 3.32 Инсталация на контролер за балансър на натоварване*

Втората предпоставка, преди разполагането на WordPress работното приложение върху създавания посредством CAPI клъстър, е инсталиранието на EBS CSI драйвер. Той се тегли като т.нар. “добавка” към клъстера, посредством “eksctl”. CSI (Container Storage Interface/ интерфейс за контейнерните хранилища) драйверът позволява използването на Amazon EBS като постоянни хранилища (PV) за StatefulSet обвивки в Kubernetes и тъй като базата данни на работното приложение използва именно StatefulSet, EBS CSI свързва контейнерите в обвивките с хранилищата в EBS. Също така драйверът използва доверителна политика, подобна на тази на Фиг. 3.31., с разликата, че “service account” името е “ebs-csi-controller-sa” и се заменя с “aws-load-balancer-controller”. В комбинация с друга политика за драйвера “AmazonEBSCSIDriverPolicy”, IAM ролята за него се създава. След като необходимите IAM роли и политики са създадени, EBS CSI драйверът се инсталира върху клъстера (Фиг. 3.33).

```

mpetrov@MishoXPS13:~/Documents/Diplomna$ eksctl create addon --name aws-ebs-csi-driver --cluster
mpetrov-wordpressCluster --service-account-role-arn arn:aws:iam::$(aws sts get-caller-identity --
query Account --output text):role/mpetrov_stsAssumeRoleWithWebIdentity_eksSAsci
[!] Kubernetes version "1.30" is in use by cluster "mpetrov-wordpressCluster"
[!] IRSA is set for "aws-ebs-csi-driver" addon; will use this to configure IAM permissions
[!] the recommended way to provide IAM permissions for "aws-ebs-csi-driver" addon is via pod identity associations; after addon creation is completed, run `eksctl utils migrate-to-pod-identity`
[!] using provided ServiceAccountRoleARN "arn:aws:iam::722377226063:role/mpetrov_stsAssumeRoleWithWebIdentity_eksSAsci"

```

*Фиг. 3.33 Инсталациране на EBS CSI драйвер чрез eksctl*

След успешното инсталиране и конфигуриране на EBS CSI драйвера, WordPress работното приложение се разполага върху кълстера, като базата данни вече използва Amazon EBS за постоянно хранилище. Освен това, с инсталацието на AWS Load Balancer Controller, ингрес ресурсът ще създава и управлява балансъор на натоварването (ALB), което от своя страна осигурява достъп до приложението през интернет.

Последната стъпка от конфигурацията е прилагането на конфигурационните файлове върху CAPI управляемия кълстер, посредством командата “kubectl apply” (Фиг. 3.34).

```
mpetrov@MishoXPS13:~/Documents/Diplomna/wordpressTask$ kubectl apply -f wordpress-deployment.yaml -f mysql-statefulset.yaml
service/wordpress-service created
ingress.networking.k8s.io/wordpress-ingress created
deployment.apps/wordpress created
service/wordpress-mysql created
statefulset.apps/wordpress-mysql created
```

*Фиг. 3.34 Прилагане на WordPress работното приложение*

В AWS конзолата могат да се наблюдават промените, настъпили след прилагането на конфигурационните файлове на работното приложение, както и останалите контролери и драйвери (Фиг. 3.35).

**Workloads: Pods (13)**

Pod is the smallest and simplest Kubernetes object. A Pod represents a set of running containers on your cluster.

[Learn more](#)

| All Namespaces  | Filter Pods by name       | < 1 2 > |
|---|---------------------------|---------|
| Name  | Age                       |         |
| <a href="#">aws-load-balancer-controller-5c7b7b6789-49srm</a> | Created<br>an hour ago    |         |
| <a href="#">aws-load-balancer-controller-5c7b7b6789-65p89</a> | Created<br>an hour ago    |         |
| <a href="#">aws-node-jl6qw</a>                                | Created<br>4 hours ago    |         |
| <a href="#">aws-node-rk2bw</a>                                | Created<br>4 hours ago    |         |
| <a href="#">coredns-cc6ccd49c-8zm2j</a>                       | Created<br>4 hours ago    |         |
| <a href="#">coredns-cc6ccd49c-jhrc1</a>                       | Created<br>4 hours ago    |         |
| <a href="#">kube-proxy-slcnl</a>                              | Created<br>4 hours ago    |         |
| <a href="#">kube-proxy-v49wx</a>                              | Created<br>4 hours ago    |         |
| <a href="#">wordpress-54897f8cb6-bgfs9</a>                    | Created<br>29 minutes ago |         |
| <a href="#">wordpress-54897f8cb6-c8xn6</a>                    | Created<br>29 minutes ago |         |

Фиг. 3.35 Списък с обвивките в WordPress кълстера

В AWS конзолата, както и през командния ред, чрез kubectl, могат да се забележат всички ресурси, като StatefulSet, сървиси и Deployment, приложени върху кълстера (Фиг. 3.36).

|   | Name                              | Namespace | Type         | Age                          | Pod count | Status                         |
|---|-----------------------------------|-----------|--------------|------------------------------|-----------|--------------------------------|
| ○ | <a href="#">wordpress-mysql</a>   | default   | statefulsets | <b>Created</b><br>1 hour ago | 3         | 3 Ready   0 Failed   3 Desired |
| ○ | <a href="#">wordpress</a>         | default   | deployments  | <b>Created</b><br>1 hour ago | 2         | 2 Ready   0 Failed   2 Desired |
| ○ | <a href="#">wordpress-mysql</a>   |           |              | <b>Created</b><br>1 hour ago |           |                                |
| ○ | <a href="#">wordpress-service</a> |           |              | <b>Created</b><br>1 hour ago |           |                                |
| ○ | <a href="#">wordpress-ingress</a> |           |              | <b>Created</b><br>1 hour ago |           |                                |

Фиг. 3.36 Разположените върху кълстера Kubernetes ресурси на WordPress работното приложение

Доказателство за правилната конфигурация на EBS CSI драйвера са създадените постоянни хранилища и прикачените към тях обвивки (Фиг. 3.37).

```
mpetrov-wordpressCluster-dynamic-pvc-9ba12706-33d2-4f4f-94b2-d14b2520d... vol-0198cefe68fa5a9b5 gp2 1 GiB
mpetrov-wordpressCluster-dynamic-pvc-f1bba68b-5191-4e7f-ac30-50870ad13... vol-09d3949b942750221 gp2 1 GiB
mpetrov-wordpressCluster-dynamic-pvc-aa549ff9-5794-47a1-b0ba-0c9deb142... vol-0c0d00f1f98eb8c32 gp2 1 GiB

External provisioner is provisioning volume for claim "default/mysql-persistent-storage-wordpress-mysql-0"
Successfully provisioned volume pvc-aa549ff9-5794-47a1-b0ba-0c9deb142456
```

Фиг. 3.37 Създадени и прикачени PV с помощта на EBS CSI драйвер

Също така, при ингрес ресурса се забелязва автоматично създаденият балансър на натоварването чрез ALB контролера (Фиг. 3.38).



| Details   |                           |   |
|---|---------------------------|---|
| Created<br><span>37 minutes ago</span>  | Namespace<br>default      | Finalizers<br>ingress.k8s.aws/resources |
| Load balancer URLs<br><a href="#">k8s-default-wordpres-cfe5cd06bf-808323321.eu-west-3.elb.amazonaws.com</a> | Ingress class name<br>alb |   |

**Load balancers (1)**

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

| Filter load balancers    |  |  |                     |                      |                      |  |
|--------------------------|--|--|---------------------|----------------------|----------------------|--|
|                          | Name                                     | DNS name                                     | State               | VPC ID               | Availability Zones   |  |
| <input type="checkbox"/> | <a href="#">k8s-default-wordpres-...</a> | <a href="#">k8s-default-wordpres-cfe5...</a> | <span>Active</span> | vpc-0abee2296a105... | 2 Availability Zones |  |

Фиг. 3.38 Конфигуриран ALB чрез AWS ALB контролер

Крайният функционален тест на приложението се прави като се отвори линкът на балансьора на натоварването, който изкарва началната страница на WordPress сайта (Фиг. 3.39).

Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Do not worry, you can always change these settings later.

Site Title

Username

Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password

.....

Show

Strong

**Important:** You will need this password to log in. Please store it in a secure location.

Фиг. 3.39 Начална страница на WordPress работното приложение

С това, WordPress работното приложение е правилно конфигурирано и разположено върху клъстера, изграден с помощта на CAPI, и е време за инсталацията на второто приложение - Grafana, върху другия клъстер. Конфигурацията на Grafana приложението е по-проста, в сравнение с тази на WordPress. За нея се използва само пакетният мениджър на Kubernetes - helm.

Първоначално трябва да се промени конфигурацията на "kubectl" и да се настрои хранилището на Grafana. За целта, с помощта на AWS CLI, "kubectl" конфигурацията се настройва със стойностите за Grafana клъстера, и чрез helm шаблоните и се инсталират и актуализират. (Фиг. 3.40).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ aws eks --region eu-west-3 update-kubeconfig --name mpetrov-grafanaCluster
Updated context arn:aws:eks:eu-west-3:722377226063:cluster/mpetrov-grafanaCluster in /home/mpetrov/.kube/config
mpetrov@MishoXPS13:~/Documents/Diplomna$ helm repo add grafana https://grafana.github.io/helm-charts
"grafana" already exists with the same configuration, skipping
mpetrov@MishoXPS13:~/Documents/Diplomna$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "eks" chart repository
...Successfully got an update from the "grafana" chart repository
Update Complete. *Happy Helm-ing!*
```

Фиг. 3.40 Инсталране на Grafana хранилище чрез Helm

След настройката на хранилището, Grafana приложението се разполага върху клъстера. При използване на шаблоните за конфигурация на Grafana е препоръчително използването на различно пространство (namespace) от стандартното, за да се избегнат конфликти с други приложения. Създава се ново пространство върху клъстера и Grafana се инсталира в него (Фиг. 3.41)

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ kubectl create namespace monitoring
namespace/monitoring created
mpetrov@MishoXPS13:~/Documents/Diplomna$ helm install my-grafana grafana/grafana --namespace monitoring
NAME: my-grafana
LAST DEPLOYED: [REDACTED]
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
```

Фиг. 3.41 Инсталране на Grafana приложение чрез Helm

Статуса на новосъздаденото приложение се наблюдава през командния ред (Фиг. 3.42).

```
mpetrov@MishoXPS13:~/Documents/Diploma$ helm list -n monitoring
NAME          NAMESPACE      REVISION      UPDATED             STATUS
CHART        APP VERSION
my-grafana   monitoring     1            [REDACTED] +0200 EET deployed
grafana-8.9.0 11.5.1

mpetrov@MishoXPS13:~/Documents/Diploma$ kubectl get all -n monitoring
NAME           READY   STATUS    RESTARTS   AGE
pod/my-grafana-6ffdbb6f96-96jrf   1/1     Running   0          6m15s

NAME          TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/my-grafana   ClusterIP  172.20.112.81 <none>        80/TCP    6m15s

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-grafana   1/1      1           1          6m15s

NAME           DESIRED  CURRENT   READY   AGE
replicaset.apps/my-grafana-6ffdbb6f96  1         1         1          6m15s
```

Фиг. 3.42 Статус на създаните Kubernetes ресурси от Grafana

Достъпът до сайта се случва локално през уеб браузъра. Въщност, за разлика от WordPress работното приложение, което през балансьора на натоварването позволява достъп през интернет, Grafana няма външен достъп. Това се дължи на факта, че достъпът до интерфейса ѝ се осъществява чрез друг вид Kubernetes механизми, като порт-пренасочване (port-forwarding). Функцията за пренасочване на порт на “kubectl” тунелира трафика от определен порт на локалната хост машина към посочения порт на посочената обвивка.

Преди това обаче е необходимо извлечането на паролата от т. нар. Kubernetes секрет (Secret). Той представлява обект, съдържащ малко количество поверителни данни, като например пароли, токени и ключове. На Фиг. 3.43 е демонстрирано извлечането на декодираната в “base64” формат парола.

```
mpetrov@MishoXPS13:~/Documents/Diploma$ kubectl get secret --namespace monitoring my-grafana -o jsonpath='{.data.admin-password}' | base64 --decode ; echo
2vziwTMVGighncG24tZYSHL3YENcwKl4s7DjXmKC
```

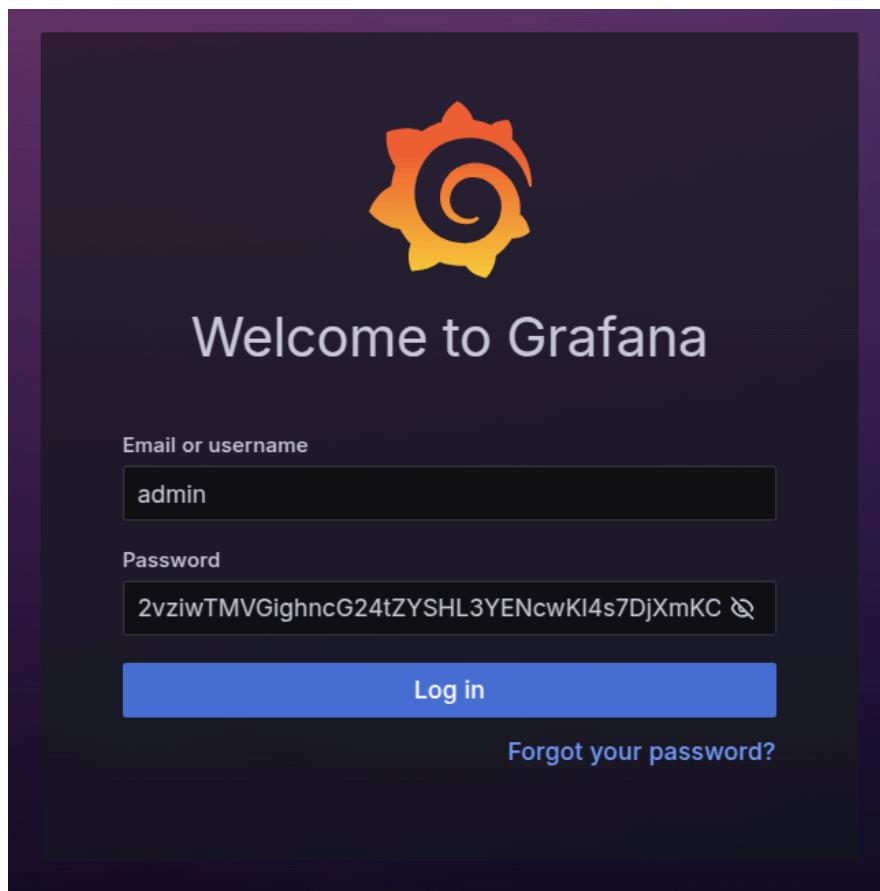
Фиг. 3.43 Извличане на администраторската парола за достъп до Grafana работното приложение

Взима се също и името на обвивката, в която се намира Grafana, и се пренасочва нейният порт към този на локалната машина (Фиг. 3.44).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ export POD_NAME=$(kubectl get pods --namespace monitoring -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=my-grafana" -o jsonpath=".items[0].metadata.name")
mpetrov@MishoXPS13:~/Documents/Diplomna$ echo $POD_NAME
my-grafana-6ffdbb6f96-96jrf
mpetrov@MishoXPS13:~/Documents/Diplomna$ kubectl --namespace monitoring port-forward $POD_NAME 3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
```

Фиг. 3.44 Пренасочване на порт на обвивка на Grafana приложение

В браузъра, на адрес “localhost:3000” или “127.0.0.1:3000”, се намира началната страница на Grafana, където посредством паролата, която бе извлечена на Фиг. 3.43, и потребител с име “admin”, се достъпва самото Grafana приложение (Фиг. 3.45).



Фиг. 3.45 Начален екран за достъп до Grafana

# ЧЕТВЪРТА ГЛАВА

## Тестване на създадената конфигурацията и функционалностите на CAPI

### 4.1. Промяна в конфигурацията на работещ кълстер чрез CAPI

След като двете работни приложения са разположени върху кълстера, функционалността на CAPI подлежи на тестване. CAPI позволява промени върху конфигурации на работещи кълстери, като Grafana Kubernetes кълстера. Примерна промяна в неговата конфигурация е в броя на работните възли в кълстера, с цел осигуряване на повече изчислителни ресурси. Сегашната конфигурация е настроена с две копия на съответните възли, което може да се забележи чрез “kubectl”, или през AWS конзолата (Фиг. 4.1).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ kubectl get nodes
NAME                      STATUS   ROLES      AGE        VERSION
ip-10-0-177-157.eu-west-3.compute.internal   Ready    <none>    4h2m      v1.30.8-eks-aeac579
ip-10-0-91-157.eu-west-3.compute.internal     Ready    <none>    3h58m     v1.30.8-eks-aeac579
```

The screenshot shows two main sections of the Grafana Kubernetes configuration interface:

- Nodes (2) Info**: A table listing two nodes:

| Node name                                  | Instance type | Compute    | Managed by                            | Created     | Status |
|--|---------------|------------|---------------------------------------|-------------|--------|
| ip-10-0-177-157.eu-west-3.compute.internal | t3.medium     | Node group | default_mpetrov-grafanacluster-pool-0 | 4 hours ago | Ready  |
| ip-10-0-91-157.eu-west-3.compute.internal  | t3.medium     | Node group | default_mpetrov-grafanacluster-pool-0 | 4 hours ago | Ready  |
- Node groups (1) Info**: A table listing one node group:

| Group name                            | Desired size | AMI release version | Launch template | Status |
|---------------------------------------|--------------|---------------------|-----------------|--------|
| default_mpetrov-grafanacluster-pool-0 | 2            | 1.30.8-20250203     | -               | Active |

Фиг. 4.1 Работни възли в Grafana кълстер преди промяна в конфигурацията

Промяна в текущата бройка, както и в минималната и максималната бройка при скалиране на възлите, става възможно, чрез модифициране на “AWSManagedMachinePool” и “MachinePool” обектите в конфигурационния файл (Фиг. 4.2).

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachinePool
metadata:
  name: mpetrov-grafanacluster-pool-0
  namespace: default
spec:
  clusterName: mpetrov-grafanacluster
  replicas: 3
  template:
    spec:
      bootstrap:
        dataSecretName: ""
      clusterName: mpetrov-grafanacluster
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta2
        kind: AWSManagedMachinePool
        name: mpetrov-grafanacluster-pool-0
      ...
apiVersion: infrastructure.cluster.x-k8s.io/v1beta2
kind: AWSManagedMachinePool
metadata:
  name: mpetrov-grafanacluster-pool-0
  namespace: default
spec:
  instanceType: "t3.medium"
  scaling:
    minSize: 1
    maxSize: 4
```

Фиг. 4.2 Промяна в конфигурациите на работните възли на *Grafana* къмъстера

При прилагане текущата бройка възли се увеличава с една (“replicas: 3”), инстанциите остават от тип “t3.medium” (“instanceType: t3.medium”), а диапазонът от броя на работните възли се променя от една до четири (Фиг. 4.3).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ kubectl apply -f grafanaEKStest.yaml
cluster.cluster.x-k8s.io/mpetrov-grafanacluster unchanged
awsmanagedcluster.infrastructure.cluster.x-k8s.io/mpetrov-grafanacluster unchanged
awsmanagedcontrolplane.controlplane.cluster.x-k8s.io/mpetrov-grafanacluster-control-plane unchanged
machinepool.cluster.x-k8s.io/mpetrov-grafanacluster-pool-0 configured
awsmanagedmachinepool.infrastructure.cluster.x-k8s.io/mpetrov-grafanacluster-pool-0 configured
```

Фиг. 4.3 Прилагане на модифицирания конфигурационен файл за къмъстера

На Фиг. 4.4 се забелязва най-скорошният новосъздаден работен възел, както и модифицираната група възли (Node Group), където се намират промените относно диапазона на възлите (Minimum size; Maximum Size).

### Nodes (3) Info

Filter Nodes by property or value

| Node name  | Instance type | Compute    | Managed by  | Created                   | Status              |
|--|---------------|------------|---|---------------------------|---------------------|
| <a href="#">ip-10-0-174-102.eu-west-3.compute.internal</a> | t3.medium     | Node group | <a href="#">default_mpetrov-grafanacluster-pool-0</a> | Created<br>14 minutes ago | <span>Read</span> y |
| <a href="#">ip-10-0-91-157.eu-west-3.compute.internal</a>  | t3.medium     | Node group | <a href="#">default_mpetrov-grafanacluster-pool-0</a> | Created<br>5 hours ago    | <span>Read</span> y |
| <a href="#">ip-10-0-177-157.eu-west-3.compute.internal</a> | t3.medium     | Node group | <a href="#">default_mpetrov-grafanacluster-pool-0</a> | Created<br>5 hours ago    | <span>Read</span> y |

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ kubectl get nodes
NAME           STATUS   ROLES   AGE    VERSION
ip-10-0-174-102.eu-west-3.compute.internal   Ready    <none>  22m   v1.30.8-eks-aeac579
ip-10-0-177-157.eu-west-3.compute.internal   Ready    <none>  4h48m  v1.30.8-eks-aeac579
ip-10-0-91-157.eu-west-3.compute.internal   Ready    <none>  4h44m  v1.30.8-eks-aeac579
```

### Node group configuration Info

|   |   |                            |
|---|---|----------------------------|
| Kubernetes version<br>1.30                                  | AMI type <a href="#">Info</a><br>AL2_x86_64 | Status <span>Active</span> |
| AMI release version <a href="#">Info</a><br>1.30.8-20250203 | Instance types<br>t3.medium                 | Disk size<br>20 GiB        |

[Details](#) [Nodes](#) [Health issues 0](#) [Kubernetes labels](#) [Update config](#) [Kut](#)

#### Details

|  |   |                                   |
|--|---|-----------------------------------|
| <b>Node group ARN</b><br><a href="#">arn:aws:eks:eu-west-3:72237226063:nodegroup/mpetrov-grafanaCluster/default_mpetrov-grafanacluster-pool-0/e4ca7281-87ba-8515-fe46-04611154f28e</a> | <b>Autoscaling group name</b><br><a href="#">eks-default_mpetrov-grafanacluster-pool-0-e4ca7281-87ba-8515-fe46-04611154f28e</a> | <b>Capacity type</b><br>On-Demand |
| <b>Created</b><br>5 hours ago  | <b>Desired size</b><br>3 nodes  | <b>Minimum size</b><br>1 node     |
|  |   | <b>Maximum size</b><br>4 nodes    |

Фиг. 4.4 Разглеждане на промените върху работните машини създадени чрез CAPI

Дори след настъпилите промени, Grafana работното приложение остава напълно функционално и достъпно, като сега разполага с повече изчислителни ресурси за обработка на данни.

Този подход за промяна на конфигурациите демонстрира как CAPI позволява динамичното управление и мащабиране на Kubernetes кълстерите по декларативен начин, осигурявайки автоматизация и намалявайки необходимостта от ръчна намеса при промяна на инфраструктурата.

## 4.2. Изтриване на Kubernetes кълстер

Изтриването на кълстери, с помощта на CAPI, е също толкова просто, колкото прилагането им. Уверявайки се, че "kubectl" конфигурацията взаимодейства с управляващия кълстер, посредством "kubectl delete -f {името на файла}", кълстерът се изтрива (Фиг. 4.5).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ kubectl delete -f grafanaEKS.yaml
cluster.cluster.x-k8s.io "mpetrov-grafanacluster" deleted
awsmanagedcluster.infrastructure.cluster.x-k8s.io "mpetrov-grafanacluster" deleted
awsmanagedcontrolplane.controlplane.cluster.x-k8s.io "mpetrov-grafanacluster-control-plane" deleted
machinepool.cluster.x-k8s.io "mpetrov-grafanacluster-pool-0" deleted
awsmanagedmachinepool.infrastructure.cluster.x-k8s.io "mpetrov-grafanacluster-pool-0" deleted
```

Фиг. 4.5 Изтриване на Grafana Kubernetes кълстера посредством *kubectl*

След стартирането на процеса за изтриването на Kubernetes кълстера (Фиг. 4.6), ресурсите му се премахват постепенно, за да се гарантира успешното им изтриване.

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ kubectl get clusters
NAME          CLUSTERCLASS      PHASE      AGE        VERSION
mpetrov-grafanacluster           Deleting   5h32m
```

Фиг. 4.6 Статус на кълстера през *kubectl*

Последователността на премахване е следната:

- Изтриване на обвивки и работни възли (Фиг. 4.7) - всички приложения и обвивки, работещи в клъстера, включително Grafana, се спират, а после работните възли се изтриват един по един.

The screenshot shows two separate sections of the AWS CloudWatch Metrics interface. The top section displays a table of terminated EC2 instances, each with its Name, Instance ID, Instance state (terminated), and Instance type (t3.medium). The bottom section shows a table of Node groups, with one group named 'default\_mpetrov-grafanacluster-pool-0' currently being deleted.

|  | Name | Instance ID         | Instance state | Instance type |
|--|------|---------------------|----------------|---------------|
|  |      | i-08c7c76c2c87b0e95 | Terminated     | t3.medium     |
|  |      | i-0b13510ffb5f1f1a3 | Terminated     | t3.medium     |
|  |      | i-008143c4ac42f6640 | Terminated     | t3.medium     |

| Node groups (1)  |              | Info                | Edit            | Delete   | Add node group |
|--|--------------|---------------------|-----------------|----------|----------------|
| Node groups implement basic compute scaling through EC2 Auto Scaling groups. |              |                     |                 |          |                |
| Group name   | Desired size | AMI release version | Launch template | Status   |                |
| default_mpetrov-grafanacluster-pool-0  | 3            | 1.30.8-20250203     | -               | Deleting |                |

Фиг. 4.7 Наблюдение на процеса на изтриване работните възли и групи

- Изтриване на контролните компоненти - контролният слой на клъстера се премахва след групата от възли (Фиг. 4.8.).

The screenshot shows a table of clusters, with one cluster named 'mpetrov-grafanaCluster' currently being deleted. The cluster is running on Kubernetes version 1.30.

| Cluster name           | Status   | Kubernetes version |
|------------------------|----------|--------------------|
| mpetrov-grafanaCluster | Deleting | 1.30               |

Фиг. 4.8 Изтриване на контролният слой

- Изтриване на мрежовите ресурси - финална стъпка, при която всички автоматично създадени посредством CAPI мрежови ресурси, като виртуални мрежи и подмрежи, биват изтрити.

След успешно премахване на всички ресурси може да се провери дали клъстера е напълно изтрит (Фиг. 4.9).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ kubectl get clusters
NAME          CLUSTERCLASS   PHASE      AGE     VERSION
mpetrov-wordpresscluster   Provisioned   19m
```

Фиг. 4.9 Останалите кълстери след изтриването на Grafana кълстер чрез CAPI

Както се забелязва, въпреки че има един наличен кълстер, "mpetrov-grafanacluster" не фигурира след извикването на команда, което е индикация за успешното му изтриване.

### 4.3. Актуализиране и надграждане на съществуващ кълстер

Актуализацията и надграждането на съществуващ Kubernetes кълстер, посредством CAPI, също не е сложна задача. За демонстрацията на тази функционалност ще се използва WordPress кълстерът. Кълстерната конфигурация се променя, където Kubernetes версията на "AWSManagedControlPlane" се задава на по-новата, в случая "1.31.4" (Фиг. 4.10).

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta2
kind: AWSManagedControlPlane
metadata:
  name: mpetrov-wordpresscluster-control-plane
  namespace: default
spec:
  eksClusterName: mpetrov-wordpressCluster
  region: eu-west-3
  sshKeyName: capi-eks
  version: v1.31.4
  network:
    vpc:
      availabilityZoneUsageLimit: 2
      availabilityZoneSelection: Random
```

Фиг. 4.10 Промяна на Kubernetes версията на WordPress кълстера

След промяната на конфигурационния файл, той бива приложен (Фиг. 4.11).

```
mpetrov@MishoXPS13:~/Documents/Diplomna$ kubectl apply -f wordpressEKS.yaml
cluster.cluster.x-k8s.io/mpetrov-wordpresscluster unchanged
awsmanagedcluster.infrastructure.cluster.x-k8s.io/mpetrov-wordpresscluster unchanged
awsmanagedcontrolplane.controlplane.cluster.x-k8s.io/mpetrov-wordpresscluster-control-plane configured
machinepool.cluster.x-k8s.io/mpetrov-wordpresscluster-pool-0 unchanged
awsmanagedmachinepool.infrastructure.cluster.x-k8s.io/mpetrov-wordpresscluster-pool-0 unchanged
```

Фиг. 4.11 Прилагане на модифицирания конфигурационен файл

През AWS EKS конзолата се забелязва, че статусът на кълстера е "актуализиране" (Updating) (Фиг. 4.12).

The screenshot shows the AWS EKS Cluster details page for 'mpetrov-wordpressCluster'. At the top, it says 'mpetrov-wordpressCluster' and has a 'Copy' button. Below that is a summary section with 'Cluster info' and 'Info'. It shows the 'Status' as 'Updating', 'Kubernetes version' as '1.30', and a 'Support period' from 'Standard support until July 23, 2025'. Under 'Cluster health issues', there are 0 issues. Under 'Upgrade insights', there are 5 issues. Under 'Node health issues', there are 0 issues. Below this is a table with columns 'Cluster name', 'Status', and 'Kubernetes version'. The first row shows 'mpetrov-wordpressCluster' with 'Status: Updating' and 'Kubernetes version: 1.30'.

| Cluster name                             | Status   | Kubernetes version |
|--|----------|--------------------|
| <a href="#">mpetrov-wordpressCluster</a> | Updating | 1.30               |

Фиг. 4.12 WordPress кълстър в процес на актуализиране

След известно време кълстърът отново придобива статус "Активен", но този път с обновената Kubernetes версия (Фиг. 4.13).

The screenshot shows the AWS EKS Cluster details page for 'mpetrov-wordpressCluster'. At the top, it says 'mpetrov-wordpressCluster'. Below that is a message: 'End of standard support for Kubernetes version 1.31 is November 26, 2025. period with additional fees. For more information, see the [pricing page](#)'. Below that is a summary section with 'Cluster info' and 'Info'. It shows the 'Status' as 'Active', 'Kubernetes version' as '1.31', and a note about standard support ending on November 26, 2025. Under 'Cluster health issues', there are 0 issues. Under 'Upgrade insights', there are 5 issues. Under 'Node health issues', there are 0 issues. Below this is a table with columns 'Cluster name', 'Status', and 'Kubernetes version'. The first row shows 'mpetrov-wordpressCluster' with 'Status: Active' and 'Kubernetes version: 1.31'. There is also a link 'Upgrade now'.

| Cluster name                             | Status | Kubernetes version               |
|--|--------|----------------------------------|
| <a href="#">mpetrov-wordpressCluster</a> | Active | 1.31 <a href="#">Upgrade now</a> |

Фиг. 4.13 Актуализиран Kubernetes кълстър

## ЗАКЛЮЧЕНИЕ

Дипломната работа представя приложението на Cluster API (CAPI) като инструмент за автоматизирано създаване и управление на Kubernetes кълстери в AWS. В процеса на разработка са конфигурирани реални инфраструктурни компоненти, като посредством CAPI са създадени управляващ и допълнителни кълстери, върху които биват разположени работни приложения.

Важен аспект също така е демонстрацията на управлението на жизнените цикли на Kubernetes кълстерите, включително тяхното автоматизирано създаване, мащабиране и изтриване. Тестването на тези функционалности показва как CAPI интегрира инфраструктурни услуги, като AWS Kubernetes услугата (EKS), балансьори на натоварването (ELB) и блокови хранилища (EBS), за да осигури стабилна среда за приложенията.

Чрез показването на практическото приложение на CAPI в контекста на DevOps се подчертава ролята му в автоматизацията на облачните среди. С помощта на работните приложения - Wordpress с MySQL, и Grafana, излагачи често срещани сценарии, допълнително се демонстрира как CAPI улеснява управлението на сложни Kubernetes инфраструктури.

Използването на CAPI в контекста на дипломната работа не само осигурява гъвкавост и автоматизация, но и значително намалява сложността на администрирането на кълстери, което го прави ценен инструмент за ефективен и стандартизиран подход към управлението на Kubernetes.

## Използвана литература

1. <https://www.atlassian.com/devops>
2. <https://www.programist.bg/virtual/v1.html>
3. <https://aws.amazon.com/what-is/containerization/>
4. <https://docs.docker.com/get-started/docker-overview/#responsive-deployment-and-scaling>
5. <https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/>
6. <https://docs.docker.com/reference/api/engine/>
7. <https://docs.docker.com/engine/>
8. <https://softuni.bg/blog/containerization-orchestration-kubernetes>
9. <https://kubernetes.io/docs/concepts/overview/>
10. <https://kubernetes.io/docs/concepts/architecture/#kube-apiserver>
11. <https://kubernetes.io/docs/concepts/workloads/pods/>
12. <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>
13. <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>
14. <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>
15. <https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>
16. <https://kubernetes.io/docs/concepts/services-networking/service/>
17. <https://kubernetes.io/docs/concepts/services-networking/ingress/>
18. <https://kubernetes.io/docs/concepts/storage/volumes/>
19. <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>
20. <https://kubernetes.io/docs/concepts/storage/storage-classes/>
21. <https://hicomm.bg/hi-tech/kakvo-predstavljavat-ay-oblachnite-uslugi-au.html>
22. <https://www.leanix.net/en/wiki/apm/iaas-vs-paas-vs-saas>
23. [https://www.techtarget.com/searchaws/definition/Amazon-Web-Services#:~:text=AWS%20\(Amazon%20Web%20Services\)%20is,computing%20platform%20provided%20by%20Amazon](https://www.techtarget.com/searchaws/definition/Amazon-Web-Services#:~:text=AWS%20(Amazon%20Web%20Services)%20is,computing%20platform%20provided%20by%20Amazon)
24. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
25. <https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html>
26. <https://docs.aws.amazon.com/autoscaling/ec2/userguide/autoscaling-load-balancer.html#integrations-aws-elastic-load-balancing-types>
27. <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cloudformation-overview.html#cfn-whatis-howdoesitwork>
28. <https://aws.amazon.com/iam/>
29. <https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html>
30. <https://cluster-api.sigs.k8s.io/introduction>
31. <https://cluster-api.sigs.k8s.io/user/concepts>
32. <https://cluster-api-aws.sigs.k8s.io/introduction>
33. <https://cluster-api.sigs.k8s.io/clusterctl/overview>
34. <https://cluster-api-aws.sigs.k8s.io/clusterawsadm/clusterawsadm.html>
35. <https://docs.aws.amazon.com/eks/latest/userguide/creating-a-vpc.html#create-vpc>
36. <https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml>

# СЪДЪРЖАНИЕ

|  |          |
|--|----------|
| <b>ИЗПОЛЗВАНИ СЪКРАЩЕНИЯ.....</b>                                      | <b>4</b> |
| <b>УВОД.....</b>   | <b>5</b> |
| <b>ПЪРВА ГЛАВА.....</b>  | <b>6</b> |
| 1.1. DevOps.....   | 6        |
| 1.1.1. DevOps екипи.....   | 6        |
| 1.1.2. DevOps методологии.....   | 7        |
| 1.1.3. Технологии и инструменти.....                                   | 7        |
| 1.1.4. Практики за сигурност (DevSecOps).....                          | 8        |
| 1.2. Виртуализация.....  | 9        |
| 1.3. Контейнеризация.....  | 10       |
| 1.4. Docker.....   | 13       |
| 1.4.1. Предистория.....  | 13       |
| 1.4.2. Docker като услуга.....   | 14       |
| 1.4.3. Docker images.....  | 16       |
| 1.4.4. Docker Hub.....   | 17       |
| 1.4.5. Dockerfile.....   | 17       |
| 1.4.6. Docker daemon.....  | 17       |
| 1.4.7. Docker engine.....  | 19       |
| 1.5. Оркестрация.....  | 19       |
| 1.6. Kubernetes.....   | 22       |
| 1.6.1. Kubernetes като услуга.....                                     | 22       |
| 1.6.2. Kubernetes контролен и работен слой (control&worker plane)..... | 24       |
| 1.7. Kubernetes ресурси.....   | 27       |
| 1.7.1. Pods (обвивка).....   | 27       |
| 1.7.2. ReplicaSet.....   | 28       |
| 1.7.3. Deployment.....   | 29       |
| 1.7.4. StatefulSet.....  | 30       |
| 1.7.5. DaemonSet.....  | 30       |
| 1.7.6. Service (сървис).....   | 31       |
| 1.7.7. Ingress (ингрес).....   | 32       |
| 1.7.8. Volumes (хранилища).....  | 33       |
| 1.7.9. PV и PVC.....   | 34       |
| 1.7.10. CRDs.....  | 36       |
| 1.8. Облачни услуги.....   | 36       |
| 1.9. Amazon Web Services.....  | 40       |

|  |            |
|--|------------|
| 1.9.1. Amazon EC2.....   | 41         |
| 1.9.2. Amazon VPC.....   | 42         |
| 1.9.3. Amazon ELB.....   | 44         |
| 1.9.4. Amazon CloudFormation.....  | 46         |
| 1.9.5. Amazon IAM.....   | 46         |
| 1.9.6. Amazon EKS.....   | 47         |
| 1.10. Cluster API.....   | 50         |
| 1.10.1. Принцип на работа.....   | 50         |
| 1.10.2. Основни компоненти и архитектура.....  | 51         |
| 1.10.3. Интеграция на CAPI в Amazon AWS.....   | 53         |
| 1.10.4. Clusterawsadm и clusterctl.....  | 54         |
| <b>ВТОРА ГЛАВА.....</b>  | <b>55</b>  |
| 2.1. Основни и функционални изисквания (ФИ).....                                     | 55         |
| 2.1.1. ФИ за създаване и настройка на клъстери в облачна среда.....                  | 55         |
| 2.1.2. ФИ за инсталация и настройка на CAPI.....                                     | 56         |
| 2.1.3. ФИ за създаване на допълнителни клъстери чрез CAPI                            | 56         |
| 2.1.4. ФИ за инсталиране на работни програми върху клъстерите създадени от CAPI..... | 57         |
| 2.1.5. ФИ за демонстриране на функционалност на CAPI.....                            | 58         |
| 2.2. Избор на облачна среда.....   | 58         |
| 2.3. Избор на инструмент за контейнеризация.....                                     | 61         |
| 2.4. Избор на инструмент за оркестрация.....   | 62         |
| 2.5. Избор на инструмент за менажиране на клъстери.....                              | 64         |
| 2.6. Избор на работно приложение.....  | 66         |
| <b>ТРЕТА ГЛАВА.....</b>  | <b>68</b>  |
| 3.1. Конфигуриране на VPC за EKS чрез CF.....  | 68         |
| 3.2. Изграждане на управляващ клъстер.....   | 70         |
| 3.3. Инсталация на CAPI.....   | 76         |
| 3.4. Създаване на допълнителни клъстери чрез CAPI.....                               | 82         |
| 3.5. Конфигуриране и разполагане на работните приложения...                          | 88         |
| <b>ЧЕТВЪРТА ГЛАВА.....</b>   | <b>104</b> |
| 4.1. Промяна в конфигурацията на работещ клъстер чрез CAPI                           | 104        |
| 4.2. Изтриване на Kubernetes клъстер.....  | 107        |
| 4.3. Актуализиране и надграждане на съществуващ клъстер....                          | 109        |
| <b>ЗАКЛЮЧЕНИЕ.....</b>   | <b>111</b> |
| <b>Използвана литература.....</b>  | <b>112</b> |