



A G H

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Projekt dyplomowy

*System lokalizacji spotów oparty na skalowalnej architekturze mikroserwisów
dla aplikacji SurfAdvisor*

*Spot map system based on a scalable microservice architecture for
SurfAdvisor application*

Autor:
Daniel Poznański
Kierunek studiów:
Informatyka
Opiekun pracy:
dr inż. Grzegorz Rogus

Kraków, 2020

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształcła taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej « sądem koleżeńskim ».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdeczne podziękowania dla promotora niniejszej pracy dra inż. Grzegorza Rogusa za cenne wskazówki i uwagi

Spis treści

1. Wprowadzenie	7
1.1. Zawartość pracy	7
1.2. Aplikacja obecnie	7
1.3. Cele pracy	8
2. Analiza wymagań	9
2.1. Wymagania funkcjonalne	9
2.2. Wymagania niefunkcjonalne.....	10
3. Wykorzystane technologie	13
3.1. Amazon Web Services	13
3.1.1. EC2.....	13
3.1.2. DynamoDB	14
3.1.3. S3.....	14
3.1.4. Route53	14
3.1.5. CloudFormation.....	14
3.2. Docker.....	14
3.2.1. Maszyny wirtualne.....	14
3.2.2. Docker containers	15
3.2.3. Docker image	15
3.3. Kubernetes	15
3.3.1. Podstawowe funkcje Kubernetes.....	16
3.3.2. Podstawowe pojęcia Kubernetes	16
3.4. Istio.....	18
3.5. Jenkins	19
3.5.1. Jenkinsfile	19
4. Implementacja	21
4.1. Architektura rozwiązania	21
4.2. Aplikacje biznesowe	22

4.3. IaC - Infrastructure as Code.....	23
4.3.1. CloudFormation.....	23
4.3.2. Kops.....	24
4.3.3. Kubernetes.....	25
4.3.4. Jenkins.....	26
4.3.5. Usuwanie.....	27
4.4. Zarządzanie ruchem i bezpieczeństwo	28
4.5. Realizacja wymagań funkcjonalnych	29
4.5.1. Diagram klas <i>spot-service</i> API.....	29
4.5.2. Diagram klas <i>map-supplier</i> API	30
4.5.3. Zasilanie mapy spotów.....	31
4.5.4. Aplikacja mobilna.....	34
5. Autoskalowanie	37
5.1. Horizontal Pod Autoscaling	37
5.2. Horizontal Node Autoscaling	39
6. Podsumowanie	41

1. Wprowadzenie

1.1. Zawartość pracy

1. Pierwszy rozdział zawiera wprowadzenie wyjaśniające czym jest aplikacja mobilna *SurfAdvisor* i jakie jej rozszerzenia są przedmiotem tej pracy.
2. W drugim rozdziale przedstawione są wymagania funkcjonalne i niefunkcjonalne jakie nowo powstały system ma spełniać.
3. Trzeci rozdział to niezbędna teoria. Opisuje zwięzłe usługi chmurowe *Amazon Web Services* i orkiestrację *Kubernetes* wraz z powiązanymi pojęciami.
4. W czwartym rozdziale omówiono implementację - architektura pod dwoma kątami, zastosowanie *IaC*, bezpieczeństwo i domenowe usługi.
5. Piąty rozdział obrazuje działanie zastosowanych mechanizmów autoskalowania.
6. W szóstym rozdziale zawarte zostało podsumowanie.

1.2. Aplikacja obecnie

SurfAdvisor to aplikacja mobilna na platformę *Android* przeznaczona dla surfer'ów, windsurfer'ów i kitesurfer'ów. Pozwala użytkownikom na dodawanie relacji ze spotu (*miejsca, w którym uprawiają sport*) zawierającej m.in. szczegóły pogodowe. Zbiór aktywnych relacji jest bardzo pomocny przy planowaniu surfowania. Oprócz tego w aplikacji możliwe jest przeglądanie ofert obozów i wyjazdów. Aplikacja dostępna jest w sklepie *Google Play*, twórcą jest Jan Gąsienica-Józkowy - kolega autora niniejszej pracy. Więcej szczegółów dostępnych jest na stronie <http://surf-advisor.info/>

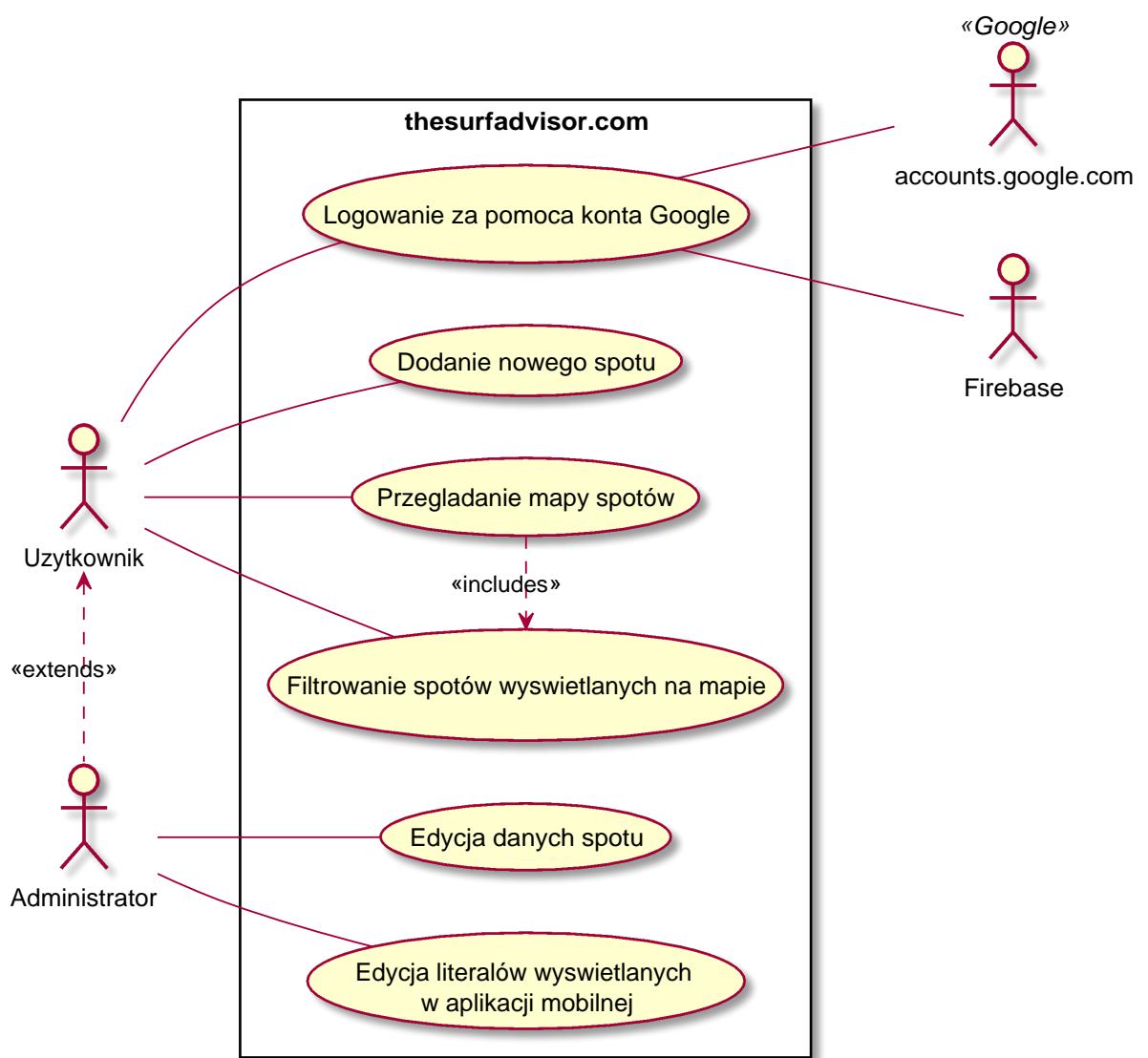
1.3. Cele pracy

Celem pracy jest projekt i implementacja usług, które zasilą nadchodzącą rozszerzoną wersję aplikacji *SurfAdvisor*. Nowe funkcjonalności koncentrują się wokół mapy spotów, użytkownik ma mieć możliwość wygodnego jej przeglądania mając dodatkowo do dyspozycji szeroką paletę filtrów.

Tworzone usługi muszą być automatycznie skalowane tak by były w stanie obsłużyć wzmożony ruch zwłaszcza latem, gdy sporty wodne uprawiane są najintensywniej. *SurfAdvisor* nie posiada własnej serwerowni, stąd wymagane jest wykorzystanie infrastruktury w chmurze. Nowo powstały system, zwany dalej *thesurfadvisor.com*, ma współpracować z istniejącymi usługami autentykacji i autoryzacji zapewnianymi przez *Firebase*.

2. Analiza wymagań

2.1. Wymagania funkcjonalne



Rys. 2.1. Diagram przypadków użycia `thesurfadvisor.com`

Przedmiotem tej pracy jest system nowych usług dla aplikacji *SurfAdvisor* - *thesurfadvisor.com*. Nowe rozszerzenie ma udostępniać:

– **Logowanie za pomocą konta Google**

Użytkownik ma mieć możliwość logowania poprzez swoje konto *Google* tak jak w obecnej wersji aplikacji. Autentykacją i autoryzacją będzie dalej zarządzać istniejący serwis *Firebase*, do którego nowy system *thesurfadvisor.com* będzie delegował ten aspekt.

– **Przeglądanie mapy spotów**

System przedstawia na mapie:

- Pojedyncze spoty w formie klikalnych punktów przekierowujących do widoku szczegółów.
- Gęste skupiska spotów w formie nieklikalnych punktów z informacją o ich liczbie. Taka klasteryzacja będzie wykonywana po stronie serwerowej by nie obciążać aplikacji mobilnej.

– **Filtrowanie spotów wyświetlanego na mapie**

Zawartość mapy można ograniczyć poprzez zastosowanie filtrowania po parametrach spotu.

– **Dodanie nowego spotu**

Trudno jest samemu skatalogować wszystkie spoty świata, stąd system będzie polegać na treści dodawanej przez użytkowników.

– **Edycja danych spotu**

Administracja *SurfAdvisor* musi mieć możliwość edycji istniejących spotów w tym zmiany ich widoczności poprzez zmianę statusu.

– **Edycja literałów wyświetlanego w aplikacji mobilnej**

Wszystkie labelki w nowej aplikacji mobilnej będą pobierane z repozytorium literałów. Pozwala to na modyfikację ich treści bez konieczności wypuszczania nowej wersji całego oprogramowania klienta.

2.2. Wymagania niefunkcjonalne

– **Wykorzystanie chmury**

SurfAdvisor nie posiada własnej serwerowni, dlatego nowy system powstanie na chmurze. Padło na AWS z uwagi na wcześniejsze doświadczenie z tą platformą autora pracy.

– **Autoskalowanie**

Usługi *thesurfadvisor.com* muszą się automatycznie skalować by sprostać zwiększeniom w nadchodzącym ruchu.

– Dostępność

Przy aplikacjach przeznaczonych dla użytkowników indywidualnych nie można sobie pozwolić na przerwy w dostępności usług. Szczególnie w przypadku surferów, którzy są aktywni również nocą często imprezując do rana. Aktualizacje oprogramowania serwisów muszą być przeprowadzane techniką *rolling update*, która nie pozwala na choćby sekundę niedostępności.

– Łatwe utrzymanie i rozwój

Programiści często zmieniają projekty lub pracę. Sytuacja, w której nowi pracownicy musieliby zgadywać jak dane środowisko powstało jest niedopuszczalna. Tutaj z pomocą przychodzi *IaC* o czym więcej w sekcji 4.3.

– Integralność aspektu bezpieczeństwa

Autentykacja i autoryzacja będzie delegowana do istniejącego serwisu *Firebase*. W ten sposób w całym systemie będą używane te same identyfikatory użytkowników pomimo istnienia usług na platformach dwóch różnych dostawców chmur.

– Monitorowanie

Parametry techniczne serwisów będą zbierane i wyświetlane w wygodnym GUI by pomóc zidentyfikować potencjalne problemy.

3. Wykorzystane technologie

3.1. Amazon Web Services

Amazon Web Services (AWS) jest najpopularniejszym dostawcą usług cloudowych na świecie [1]. Swoją pozycję zawdzięcza nieustannemu rozwojowi swoich produktów i wsłuchiwaniu się w potrzeby klientów. Duże znaczenie ma również fakt, że Amazon.com jest również zbudowany na swojej platformie [2]. Klienci z ponad 190 krajów aktywnie korzystają z rozrastającego się wachlarza 175+ produktów, serwowanych przez centra danych położone na całym świecie [3]. AWS innowacje wprowadza już na poziomie hardware'u - projektuje własne wyspecjalizowane komponenty na użytek swoich serwerowni. Jednym z najnowszych przykładów jest Nitro Card, który redefiniuje tradycyjne podejście do wirtualizacji obniżając przy tym koszty, zwiększając wydajność i bezpieczeństwo [4].

3.1.1. EC2

Podstawowy produkt AWS zapewniający nam wirtualne lub *bare metal* instancje serwerów. Przychodzący i wychodzący ruch internetowy kontrolujemy przy pomocy wirtualnego firewalla - Security Group. Poprzez mechanizm EC2 Auto Scaling możemy zdefiniować zasady ich skalowania horyzontalnego do aktualnych potrzeb. Oferowane typy instancji różnią się dedykowanym przeznaczeniem [4]:

- **M5: General Purpose**

Monolityczne aplikacje biznesowe

- **T3: General Purpose - Burstable**

Mikroserwisy, interaktywne aplikacje wymagające niskiej latencji

- **C5: Compute Optimized**

Aplikacje wymagające wysokiej wydajności

- **P3dn: Machine Learning**

Wysoko wydajne uczenie maszynowe

- i wiele innych

Obraz możemy wybrać z bazowego zestawu Linux'ów i Windows'ów lub zasięgnąć AWS Marketplace, gdzie zamieszczane są obrazy budowane przez szerszą społeczność.

3.1.2. DynamoDB

Baza danych NoSQL w pełni zarządzana przez AWS. Zaimplementowana z myślą o wysokiej wydajności niezależnie od aktualnej skali, operacje realizowane są w jedno cyfrowej ilości milisekund. Od użytkownika wymaga się jedynie zdefiniowania podstawowej struktury tabeli. Utrzymanie i skalowanie bierze na siebie AWS [3].

3.1.3. S3

Wszechstronny serwis do przechowywania plików, używany często przez inne produkty AWS (*np. DynamoDB przechowuje backup'y na S3*). Jednym z popularniejszych przypadków użycia jest hostowanie na S3 aplikacji webowych. Jak każdy z podstawowych serwisów zarządzanych przez AWS, S3 jest przygotowane na każdą skalę, zachowując przy tym wydajność i bezpieczeństwo [3].

3.1.4. Route53

Bogaty, zintegrowany z platformą serwis DNS. Najczęściej używany by nakierować ruch na instancje EC2. Udostępniona jest również opcja zakupu domeny bezpośrednio w konsoli webowej AWS co przyspiesza gotowość jej użycia.

3.1.5. CloudFormation

Obowiązkowe narzędzie przy pracy z większą ilością instancji produktów AWS. Całą swoją infrastrukturę definiuje się w czytelnym formacie pliku tekstowego *yaml* lub *json*. Z tak przygotowanym plikiem przełożenie definicji na świat rzeczywisty jest dosłownie jednym kliknięciem. Równie łatwo wykonamy aktualizację lub pozbędziemy się wszystkich utworzonych produktów [3]. Taką automatyzację udostępnia również open source'owy odpowiednik - Terraform.

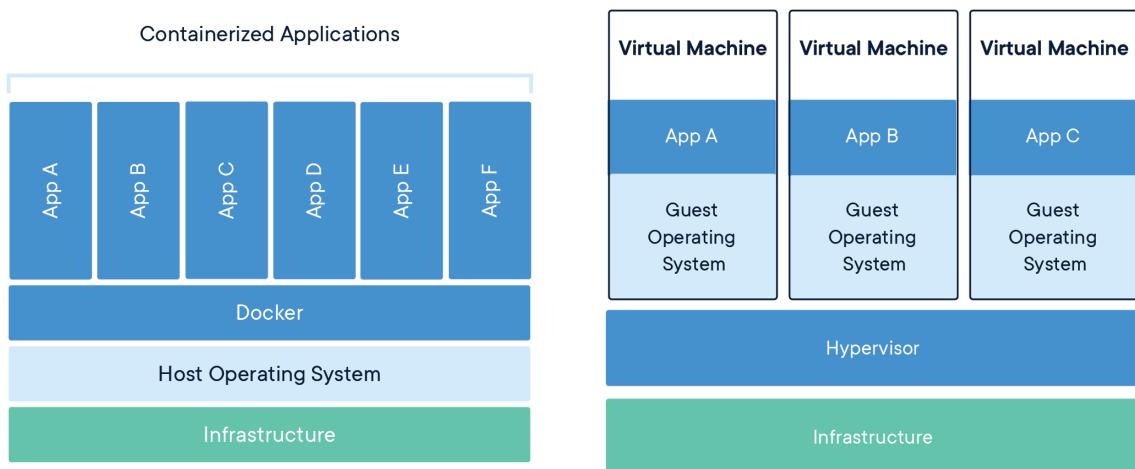
3.2. Docker

3.2.1. Maszyny wirtualne

Tradycyjne podejście do wirtualizacji skupia się na abstrakcji fizycznego sprzętu. Pozwala to na traktowanie jednego hosta jako wiele serwerów. Każda wirtualna maszyna zawiera pełną kopię systemu operacyjnego, wszystkie potrzebne narzędzia i pliki docelowej aplikacji, sumując wszystko często otrzymamy dziesiątki GB. Uruchomienie takich jednostek jest dosyć wolne (*kilka minut*) [5].

3.2.2. Docker containers

Docker natomiast jest abstrakcją systemu operacyjnego. Kontenery dockera są odciążone od warstwy OS i zawierają tylko aplikację i potrzebne jej narzędzia. Każdy kontener działa jako wyizolowany proces. Są dużo lżejsze (*zazwyczaj dziesiątki MB*), więc ten sam host pomieści więcej aplikacji uruchomionych jako kontenery niż jako wirtualne maszyny [5].



Rys. 3.1. Docker containers vs VMs [5].

3.2.3. Docker image

Jest to plik złożony najczęściej z kilku warstw [6]. Najczęściej wszystkie warstwy oprócz wierzchniej są *read-only*, stanowią narzędzia wykorzystywane przez docelową aplikację (*np. java sdk*). Najwyższa warstwa budowana jest przy pomocy pliku z instrukcją - **Dockerfile**, gdzie dokładamy naszą aplikację. Przygotowany w ten sposób image jest gotowy do uruchomienia jako kontener. Dużym udogodnieniem dla użytkownika jest globalnie publiczne repozytorium docker images - **hub.docker.com**.

3.3. Kubernetes

W zarządzaniu większą ilością produktów AWS stanowiących naszą platformę pomoże nam opisany wcześniej CloudFormation (3.1.5). Do orkiestracji skonteneryzowanymi aplikacjami posłużymy się wiodącym standardem w tej dziedzinie - Kubernetes. Projekt jest rezultatem ponad 10-cio letniego doświadczenia Google w tej dziedzinie opublikowanym na zasadach open-source w 2014 roku [7]. Rozwiążanie zyskało na nowych pomysłach społeczności i adaptacji przez wszystkich wiodących dostawców usług chmurowych. Istnieje również możliwość zastosowania

Kubernetes we własnej serwerowni, co oznacza duży krok w kierunku jednolitego środowiska niezależnie od sposobu hostowania systemu naszej aplikacji. Całą strukturę kontenerów jak również i wszystkie inne potrzebne zasoby definiujemy w plikach tekstowych *yaml* lub *json*.

3.3.1. Podstawowe funkcje Kubernetes

– Service discovery & load balancing

Funkcjonalności Kubernetes realizują skonteneryzowane aplikacje systemowe niewidoczne dla użytkownika na porządku dziennym. Wśród nich znajdziemy grupę odpowiedzialną za wewnętrzny systemowy DNS. Ruch rozkładany jest na ukryte za serwisową domeną instancje aplikacji.

– Storage orchestration

Automatyczne utworzenie zadanego zasobu pamięci na podstawie definicji tekstuowej i dołączenie jej pod wskazany kontener.

– Automatic bin packing

Mając do dyspozycji zestaw nodów (hostów), na których umożliwiamy stawianie kontenerów, Kubernetes sam podejmuje decyzje na temat wdrożenia. Dzięki temu nie musimy się martwić o przeciążenie pojedynczego noda. Poprzez definicje wymagań (*CPU, RAM*) danego kontenera możemy natomiast dostarczyć wskazówki.

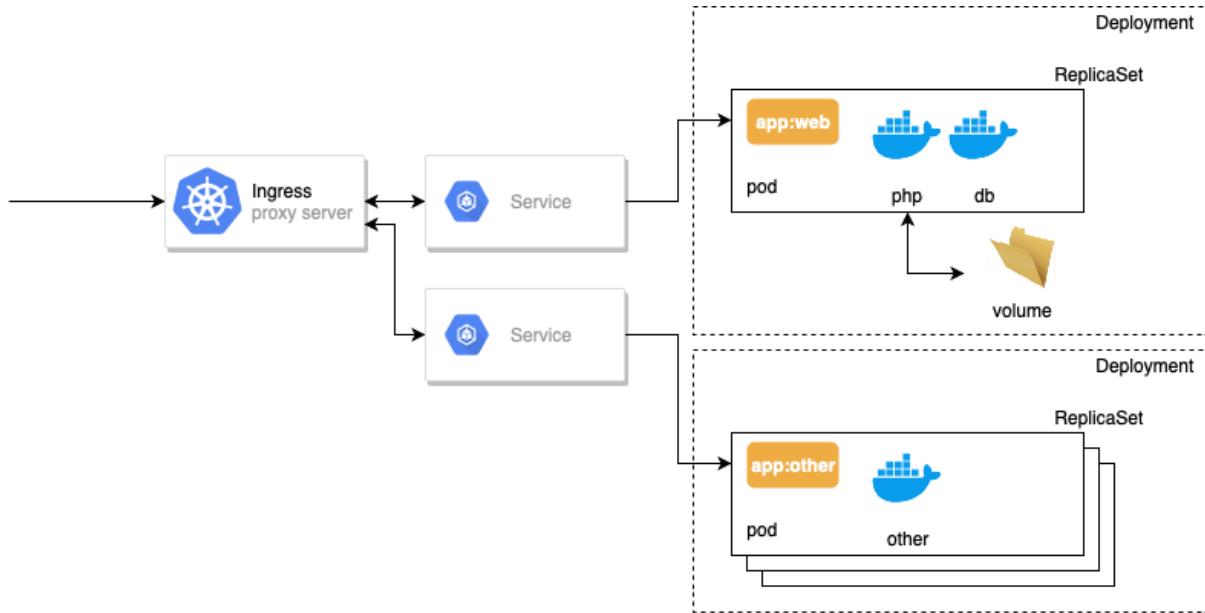
– Autoscaling, Self-healing, Automated rollouts and rollbacks

Za pomocą plików tekstowych definiujemy oczekiwana liczbę instancji naszej aplikacji i jej zasady skalowania horyzontalnego (*wpływ CPU, RAM*). Ponadto możemy określić sygnały (*np. zapytanie http*) w oparciu o które Kubernetes może weryfikować zdolność do pracy danej instancji, a następnie podjąć próby jej restartu. W przypadku utracenia całego noda (hosta) utracone tam kontenery zostaną przerzucone na resztę nodów do czasu startu nowego. Domyślnie update kontenera (*np. do nowej wersji docker image*) polega wpierw na uruchomieniu jego nowej wersji, a dopiero potem przepięciu ruchu i zabiciu starej. Jeśli nowy kontener nie spełni wymagań żywotności, zostanie on wycofany. Taki proces zapewnia 100% dostępności aplikacji [7].

3.3.2. Podstawowe pojęcia Kubernetes

3.3.2.1. Cluster

Node jest pojedynczą jednostką obliczeniową, może być to fizyczny komputer lub jak w większości przypadków - wirtualna maszyna. Cluster to grupa Node'ów stanowiąca jeden organizm. Wdrażając aplikacje nie specyfikujemy na którym Node mają stanąć nowe kontenery. Kubernetes nas w tym wręcza (3.3.1).



Rys. 3.2. Droga zapytania przez obiekty Kubernetes [8].

3.3.2.2. Pod

Pod jest najmniejszym obiektem w świecie Kubernetes jaki możemy wdrożyć. Stanowi pojedynczą instancję aplikacji. Najczęściej zawiera jeden kontener, choć może zawierać ich więcej. Kontenery wewnętrz Po'd'a współdzielą zasoby i unikalny adres IP przydzielony przez orkiestra-tora [9].

3.3.2.3. ReplicaSet & Deployment

Zadaniem ReplicaSet jest utrzymanie danej liczby identycznych Pod'ów (*instancji danej aplikacji*). Jednak w większości przypadków nie będziemy pracować bezpośrednio z ReplicaSet tylko z obiektem wyżej w hierarchii - Deployment. Poprzez niego przeskalujemy oczekiwana liczbę Pod'ów i przeporwadzimy *rolling update* [9].

3.3.2.4. Service

Sposób wystawienia Pod'ów aplikacji jako serwis dostępny w sieci. Pozostałe aplikacje odnoszą się do niego korzystając z jego wewnętrz Cluster'owej nazwy DNS. Ruch jest następnie rozkładany pomiędzy cały ReplicaSet aplikacji [9]. Do typów Service należą:

- **ClusterIP**

Dostęp tylko wewnątrz Cluster'a

- **NodePort**

Dostęp również poza Cluster'em poprzez:

`<adres IP dowolnego Noda>:<zdefiniowany port Service'u>`

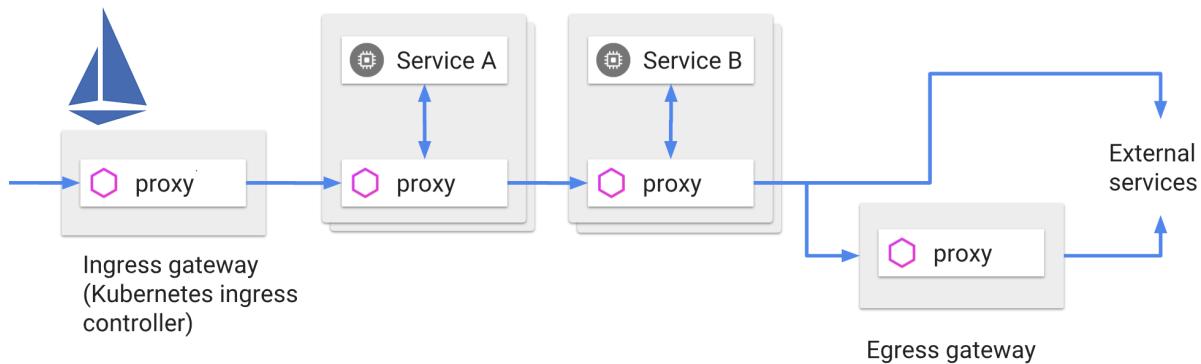
- **LoadBalancer**

Service jest wystawiony poprzez *LoadBalancer* utworzony przez dostawcę platformy chmurowej, na której działa nasz Cluster.

3.3.2.5. Ingress

Częstą praktyką jest utworzenie jednego punktu dostępu dla całego Cluster'a. Wdrażana jest wtedy jedna z implementacji *Ingress Controller* (najpopularniejsza to *nginx*, w projekcie używany jest *Istio* [3.4]), która stanowi ruter dla pozostałych Service'ów. Ingress to obiekt definiujący rutowanie nadchodzących pakietów (np. na podstawie ścieżki http) [9].

3.4. Istio



Rys. 3.3. Zastosowanie Istio w Kubernetes [10].

Wraz z transformacją z monolitycznej aplikacji do sieci mikroserwisów pojawia się wiele nowych problemów związanych z komunikacją pomiędzy nimi. Kubernetes nie dostarcza żadnego domyślnego rozwiązania, by nie ograniczyć doboru przez użytkownika. Istio jest jednym z nich, wypuszczone przez *Lyft* i zaadaptowane później przez m.in. *Google* i *Microsoft*. [11]

Istio zapewnia nam:

- Szczegółową kontrolę nad zachowaniem się ruchu poprzez definicje rutowań, ponownie, failovers i circuit breakers.
- Automatyczne metryki, odkładanie logów i śledzenie ruchu w naszym Cluster'rze.
- Bogate narzędzia do wprowadzenia autentykacji i autoryzacji

Włączenie funkcjonalności Istio do Pod'a odbywa się poprzez wstrzyknienie pobocznego kontenera, który stanowi proxy dla głównego kontenera aplikacji.

3.5. Jenkins

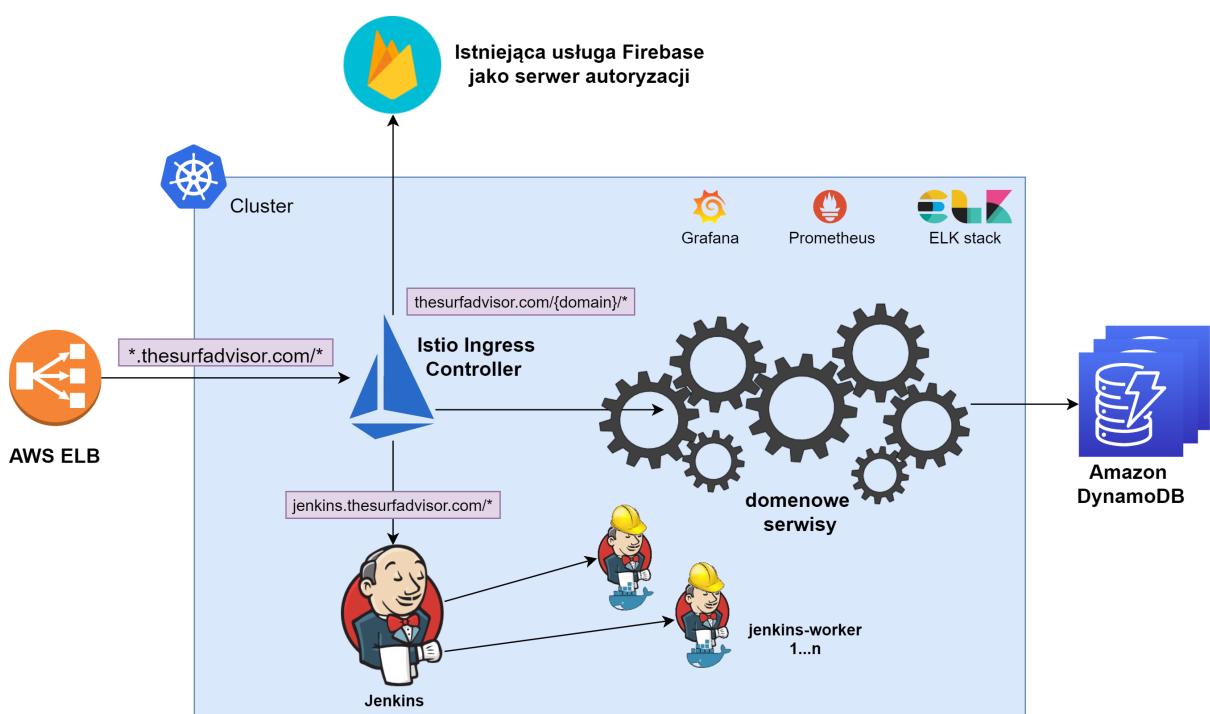
Jenkins jest najpopularniejszym narzędziem do **CI & CD** w branży. Utrzymywany przez społeczność open source nadąża za zmianami w świecie programowania. Podstawowe funkcjonalności rozszerza bogaty zbiór pluginów. Jenkins może być wdrożony na wiele sposobów, w tym projekcie stawiany jest jako docker container w środku Cluster'a.

3.5.1. Jenkinsfile

Jest to plik (*groovy syntax*), w którym definiowany cały ***pipeline*** - poszczególne kroki od zaciągnięcia plików źródłowych z SCM do wdrożenia na Cluster. Ponadto przetrzymywany w zdalnym repozytorium stanowi wskaźnik dla Jenkinsa, by dane repozytorium było brane pod uwagę w ramach job'a "*Github Organization*".

4. Implementacja

4.1. Architektura rozwiązania



Rys. 4.1. Cluster nowych usług SurfAdvisor

Cluster nowych usług SurfAdvisor w spoczynku składa się z:

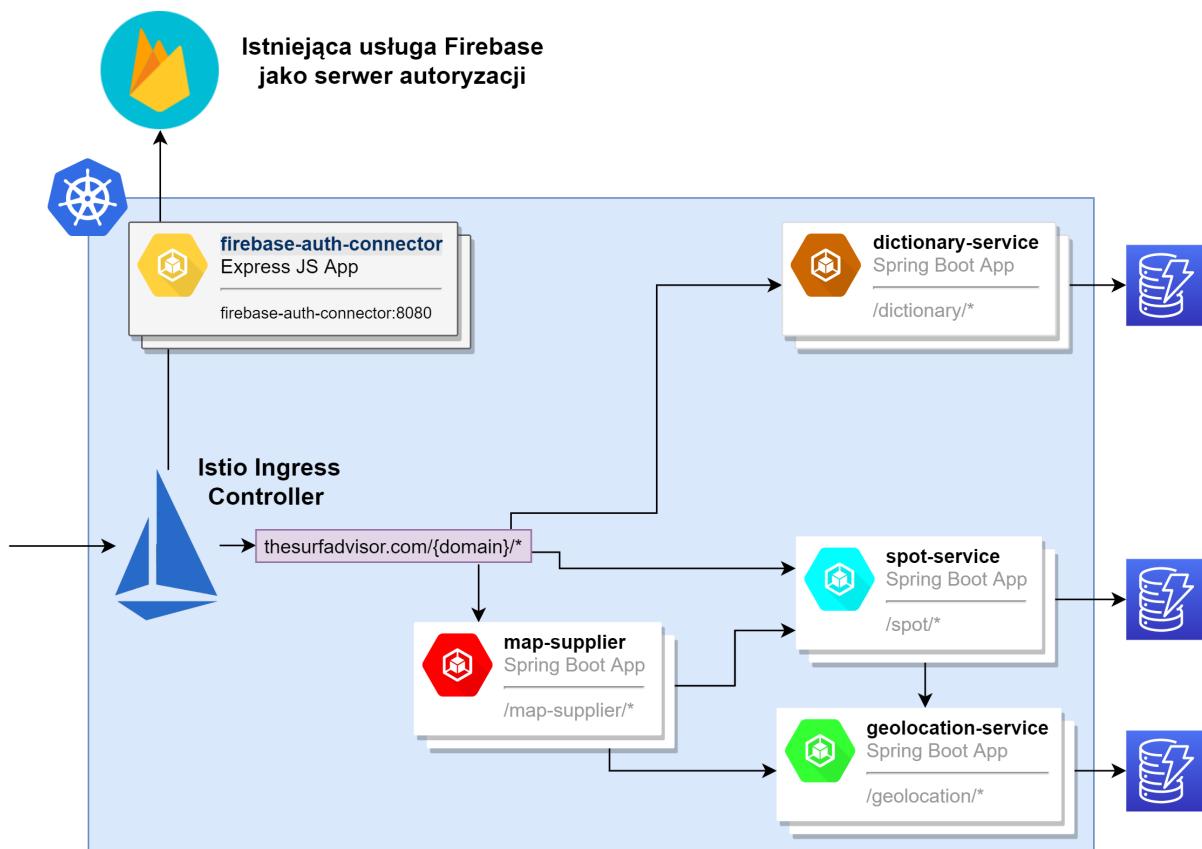
- 1 x master Node - EC2 **m3.medium** (*1 vCPU & 3.75 GiB RAM*)
- 2 x worker Node - EC2 **t3.medium** (*2 vCPU & 4 GiB RAM*)

W przypadku zwiększenia natężenia ruchu liczba Node'ów jest skalowana by sprostać wymaganiom. Pod globalnie dostępną domeną **thesurfadvisor.com** kryje się *Load Balancer* utrzymywany przez AWS. Stanowi on jedyny punkt dostępu, cały ruch jest następnie obsługiwany przez *Istio Ingress Controller*. Autoryzacja zintegrowana jest z istniejącym systemem Firebase. Każdy domenowy serwis uderza do własnej bazy danych DynamoDB.

Jako administrator możemy się dostać do subdomen takich jak:

- *jenkins.thesurfadvisor.com*
Jenkins WEB UI pracującego wewnątrz Cluster'a
- *grafana.thesurfadvisor.com*
Grafana WEB UI śledząca metryki stanu technicznego
- *kibana.thesurfadvisor.com*
Kibana WEB UI do wygodnego przeszukiwania logów

4.2. Aplikacje biznesowe



Rys. 4.2. Zbliżenie na domenowe serwisy SurfAdvisor

– **firebase-auth-connector**

Warstwa pośrednia pomiędzy Istio *Ingress Controller* a istniejącym serwisem Firebase.

– **dictionary-service**

Repozytorium literałów wyświetlanych w aplikacji mobilnej.

- **geolocation-service**

Wiedza o położeniu obiektów na kuli ziemskiej.

- **spot-service**

Atlas spotów - miejsc do uprawiania sportu wodnego.

- **map-supplier**

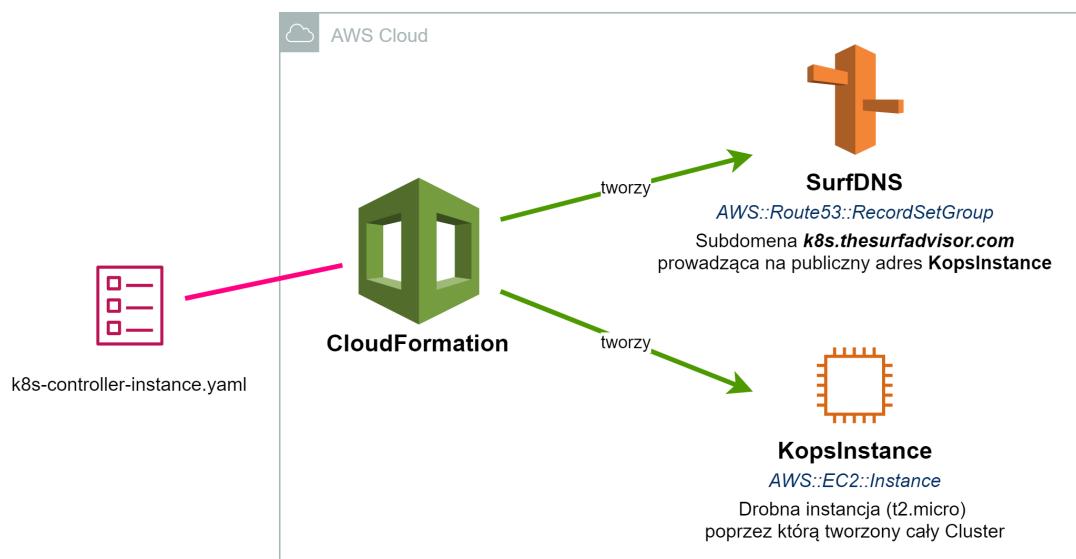
Agreguje resztę serwisów związanych z mapą, by wydajniej obsłużyć aplikacje mobilne.

4.3. IaC - Infrastructure as Code

Jest to podejście do zarządzania infrastrukturą (*siecią, wirtualnymi maszynami, kontenerami etc.*) poprzez pliki tekstowe, których treść w sposób jednoznaczny definiuje potrzebne zasoby. Taką deklaratywną formę konfiguracji bardzo łatwo można śledzić poprzez ten sam system kontroli wersji co ten używany wokół kodu aplikacji. IaC jest odpowiedzią na problem różnic pomiędzy środowiskami narastającymi jeszcze bardziej z czasem, jakie powodowały ich manualne imperatywne modyfikacje. Założeniem IaC jest **idempotentność** wdrożeń definicji z plików - za każdym razem rezultatem ma być takie samo środowisko [12].

W projekcie realizuje tę koncepcję stawiając cały system w czterech krokach. Pierwsze trzy stawiają Cluster na AWS, są w pełni zautomatyzowane: *CloudFormation*, *Kops* i *Kubernetes*. Ostatni wdraża domenowe aplikacje, wymaga paru kliknięć w konsoli webowej *Jenkins'a*.

4.3.1. CloudFormation



Rys. 4.3. Inicjacja Cluter'a SurfAdvisor - 1. CloudFormation

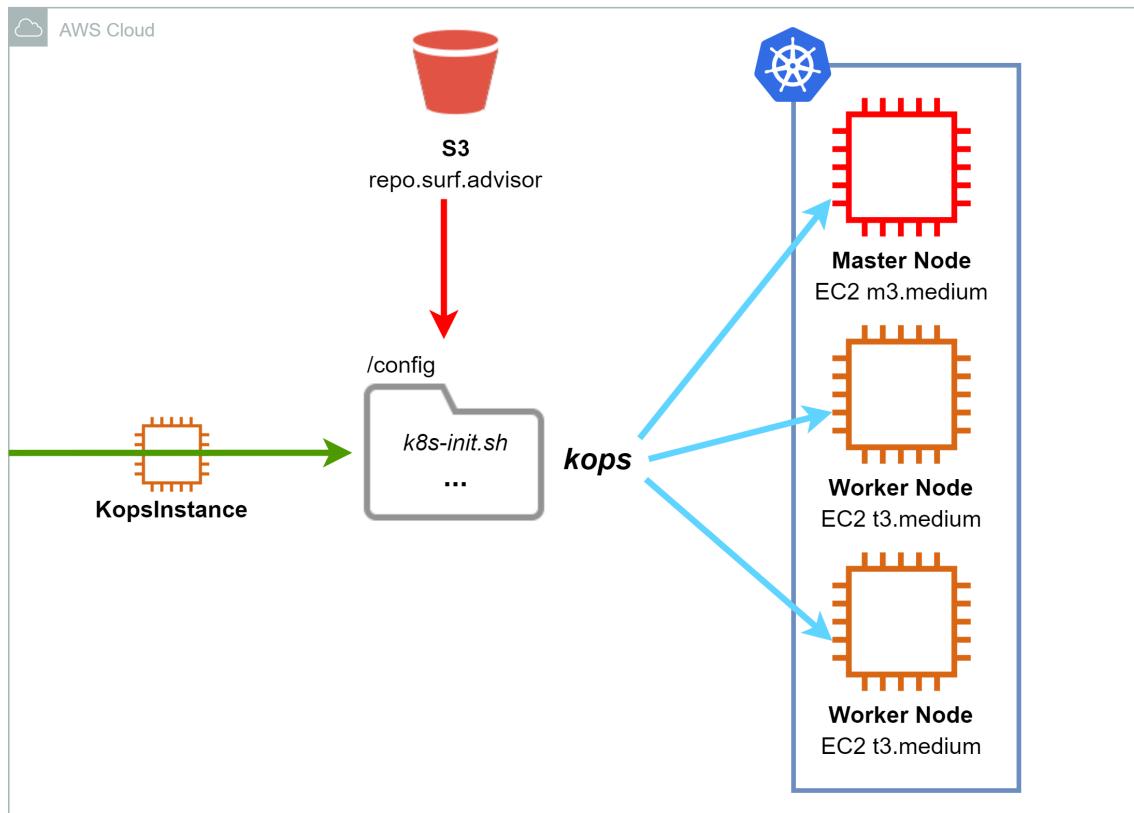
Najistotniejszym plikiem CloudFormation w projekcie jest ten inicjujący drobną instancję przeznaczoną do zarządzania docelowym Cluster'em. Plik k8s-controller-instance.yaml definiuje głównie:

- **KopsInstance** - drobna instancja linuxa
- **SurfDNS** - subdomena oszczędzająca znajomości dokładnego adresu IP (*np. w razie potrzeby ssh*)

Kluczowe znaczenie ma możliwość definicji **UserData** - skryptu *bash* wykonywanego zaraz po starcie instancji. W ramach projektu zdefiniowano w nim polecenia:

- ściągnięcia dalszych instrukcji *bash* z S3
- instalacji potrzebnych narzędzi (*kops, kubectl, helm*)
- zbudowania Cluster'a poprzez *kops* (4.3.2)
- wdrożeniu pomocniczych Service'ów poprzez *kubectl & helm* (4.3.3)

4.3.2. Kops

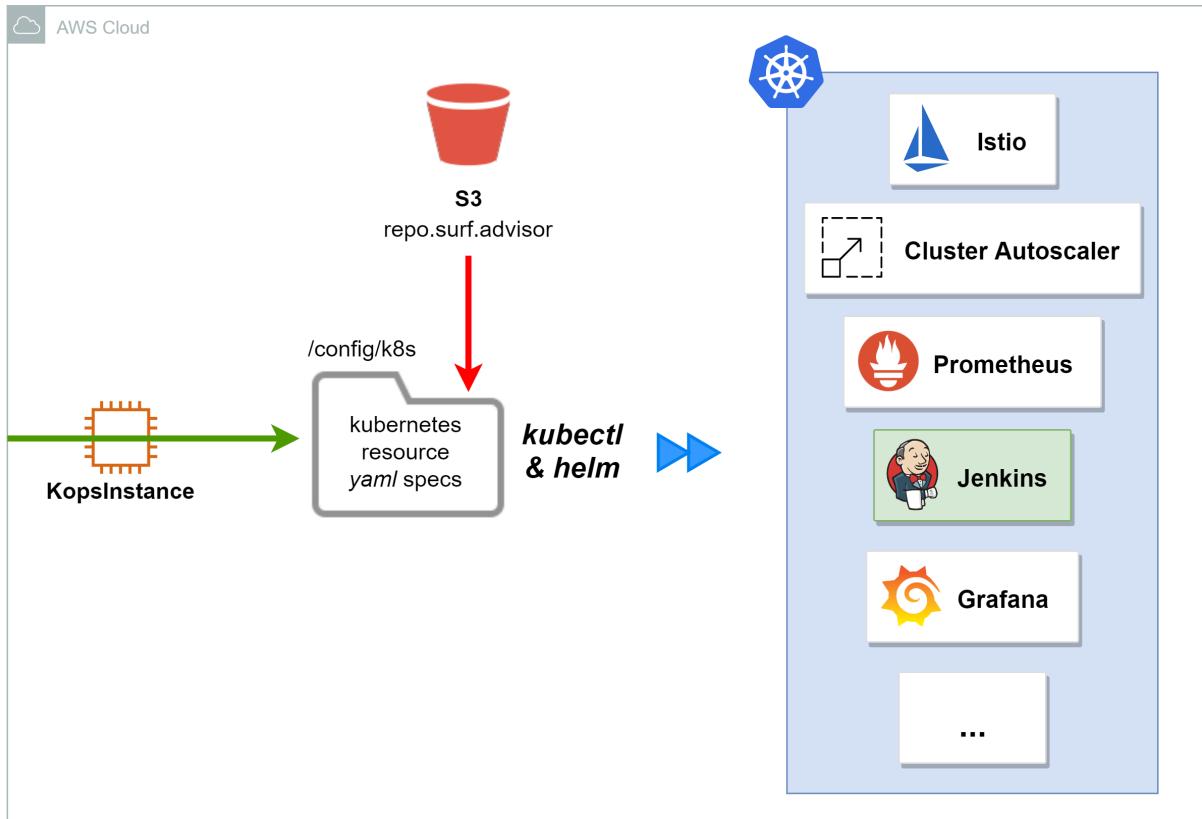


Rys. 4.4. Inicjacja Cluter'a SurfAdvisor - 2. Kops

kops to open-source CLI do budowania Cluster'ów Kubernetes na platformie AWS [13]. Stanowi alternatywę dla stosunkowo drogiego na start oficjalnego rozwiązania - EKS [14].

Przy użyciu tego narzędzia dalszy ciąg skryptów, uruchomionych w ramach *UserData Kops Instance* (4.3.1), buduje Node'y składające się na Cluster.

4.3.3. Kubernetes



Rys. 4.5. Inicjacja Cluter'a SurfAdvisor - 3. Kubernetes

Gdy *kops* skończy już stawiać Cluster do akcji wkracza:

- ***kubectl*** - oficjalne CLI do operacji na zasobach Kubernetes
- ***helm*** - package manager dla Kubernetes

Wdrażane są Service'y zaspokajające wymagania bardziej niefunkcjonalne, m.in.:

- *Jenkins* - który kontynuuje proces inicjacji systemu [4.3.4]
- *Istio* - sekcja [4.4]
- *Prometheus* & *Grafana* - monitoring parametrów technicznych Cluster'a
- *Cluster Autoscaler* - rozdział [5]

4.3.4. Jenkins

Jenkins jest gotowy do użycia po paru minutach (*wraz ze wszystkimi plugin'ami zdefiniowanymi w pliku konfiguracyjnym*). Jedynym manualnym krokiem jaki trzeba wykonać jest stworzenie zadania "*Github Organization*", gdzie podajemy ID naszej organizacji.

The screenshot shows the Jenkins web interface. At the top, it says 'Not secure | jenkins.thesurfadvisor.com/newJob'. Below that is the Jenkins logo and the word 'Jenkins'. A navigation bar shows 'Jenkins >'. The main content area has a title 'Enter an item name' and a text input field containing 'surfadvisor'. Below the input field is a note '» Required field'. Underneath, there is a section titled 'GitHub Organization' with a GitHub icon, describing it as 'Scans a GitHub organization (or user account) for all repositories matching some defined markers.'

Rys. 4.6. Jenkins: wdrożenie domenowych serwisów - job "Github Organization"

Następnie Jenkins skanuje wszystkie repozytoria widniejące pod daną organizacją i wybiera te zawierające ***Jenkinsfile***. Po chwili będzie dostępna zakładka, na której wylistowane są wszystkie repozytoria spełniające wyżej wymienione kryterium:

The screenshot shows the Jenkins 'SurfAdvisor 2.0' dashboard. At the top, it features a logo of a beach hut and the text 'SurfAdvisor 2.0'. Below that, it says 'Folder name: surfadvisor'. A 'Repositories (5)' section is shown with a table. The columns are labeled 'S', 'W', 'Name ↓', and 'Description'. The data is as follows:

S	W	Name ↓	Description
		dictionary-service	simple dictionary µService
		firebase-auth-connector	Middle layer between k8s ingress controller and firebase auth interface
		geolocation-service	geolocation data repo
		map-supplier	aggregating other services to satisfy map related client queries
		spot-service	surfing spot details registry

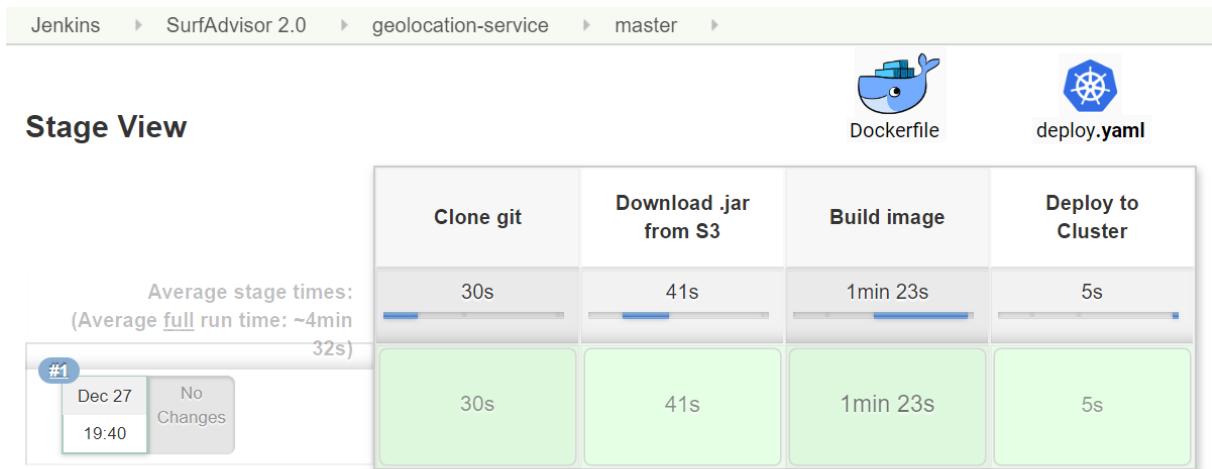
At the bottom left, it says 'Icon: S M L'.

Rys. 4.7. Jenkins: wdrożenie domenowych serwisów - folder SurfAdvisor

Dla każdego z repozytorium Jenkins od razu uruchamia zdefiniowany w *Jenkinsfile* pipeline. W przypadku tego projektu repozytoria łączy jeszcze obecność dwóch innych plików:

- **Dockerfile** - instrukcja budowy docker image wstawianego potem na DockerHub
- **deploy.yaml** - deklaracja obiektów Kubernetes: Service, Deployment, Pod, na których stanie dana aplikacja. Wdrażane są poprzez *kubectl*.

Poniżej przedstawiony jest wycinek procesowania pipeline dla aplikacji *geolocation-service*. Dwa końcowe kroki udekorowane są dodatkowo symbolem pliku, na którym polegają.



Rys. 4.8. Jenkins: wdrożenie domenowych serwisów - pipeline

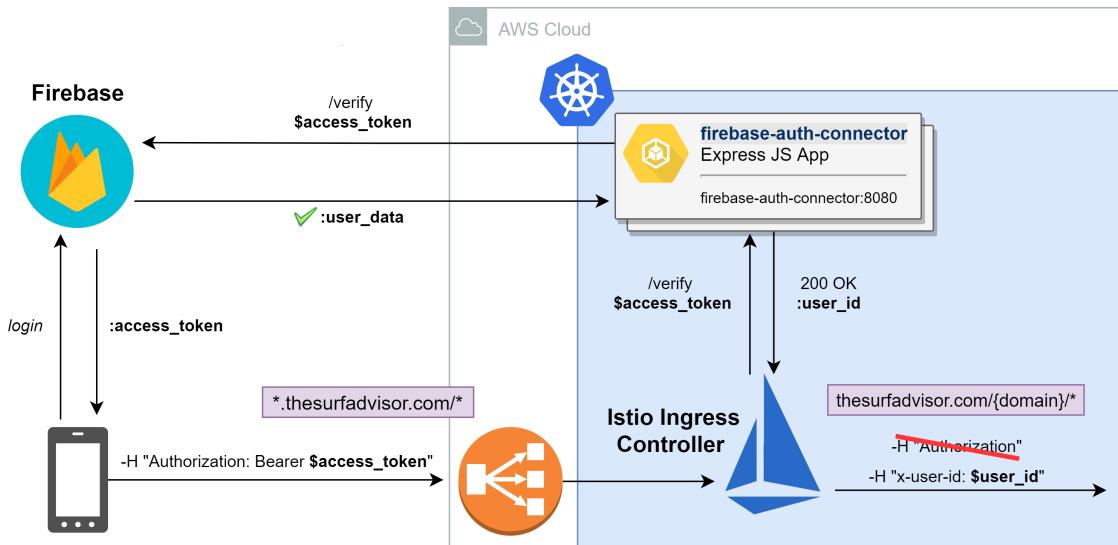
4.3.5. Usuwanie

Cały powstały w sposób opisany wyżej Cluster wraz ze wszystkimi powiązanymi produktami AWS (*EC2, LoadBalancer, etc.*) usuwa się jedną komendą:

- `kops delete cluster --name ${NAME} --yes`

Bardzo przydatne na etapie developmentu - usuwamy system kończąc pracę w danym dniu, co przynosi spore oszczędności.

4.4. Zarządzanie ruchem i bezpieczeństwo



Rys. 4.9. Diagram autentykacji / autoryzacji

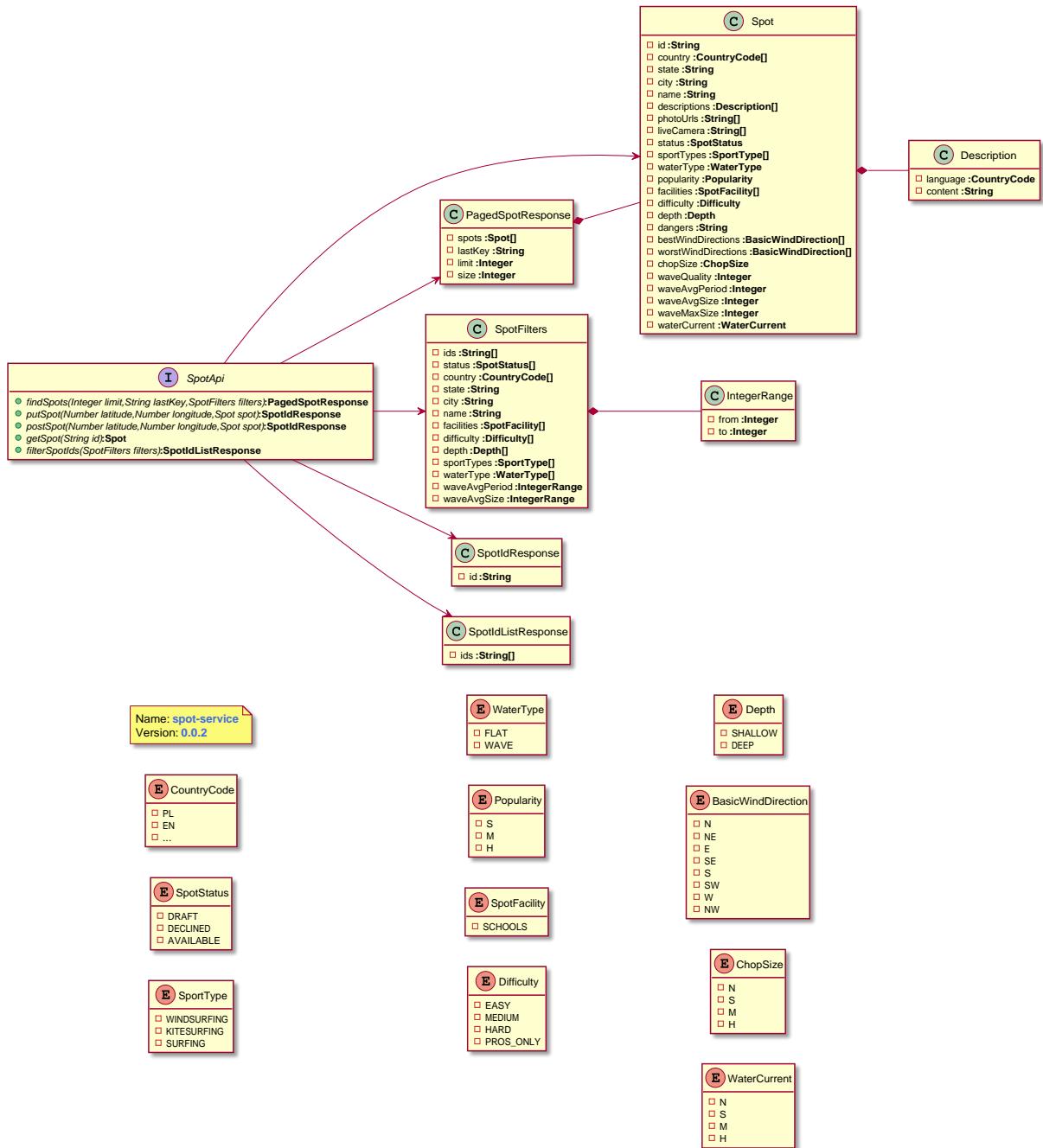
Poprzez ustawienie w *Route53* ruch w stronę globalnej domeny ***thesurfadvisor.com*** jest kierowany na Load Balancer, który następnie rozrzuca go po Node'ach Cluster'a. Na każdym z tych Node'ów ruch trafia wpierw do instancji *Istio Ingress Controller*. Po pomyślniej autentykacji ruch jest kierowany do odpowiedniego Service'u na podstawie ścieżki.

Diagram 4.9 ilustruje przebieg autentykacji i autoryzacji:

1. Proces logowania w aplikacji mobilnej pozostaje bez zmian. Tak jak w starej wersji użytkownik loguje się do Firebase ktrymś z dostępnych sposobów.
2. Aplikacja mobilna dekoruje zapytania do *thesurfadvisor.com* header'em z tym samym tokenem dostępu, który jest używany do komunikacji z usługami Firebase.
3. Istio *Ingress Controller* deleguje weryfikację nadchodzącego tokena do Service'u **firebase-auth-connector**.
4. *firebase-auth-connector* poprzez oficjalne SDK zleca Firebase weryfikację tokena. W przypadku sukcesu Firebase zwraca mały zbiór podstawowych danych użytkownika. Service ten następnie zwraca do Istio tylko jego identyfikator, w planach jest dołożenie jeszcze zbioru uprawnień.
5. Istio usuwa niepotrzebny już header **"Authorization"** i ustawia **"x-user-id"** z identyfikatorem użytkownika. W ten sposób dalsze domenowe serwisy są odciążone od implementowania tego aspektu bezpieczeństwa.

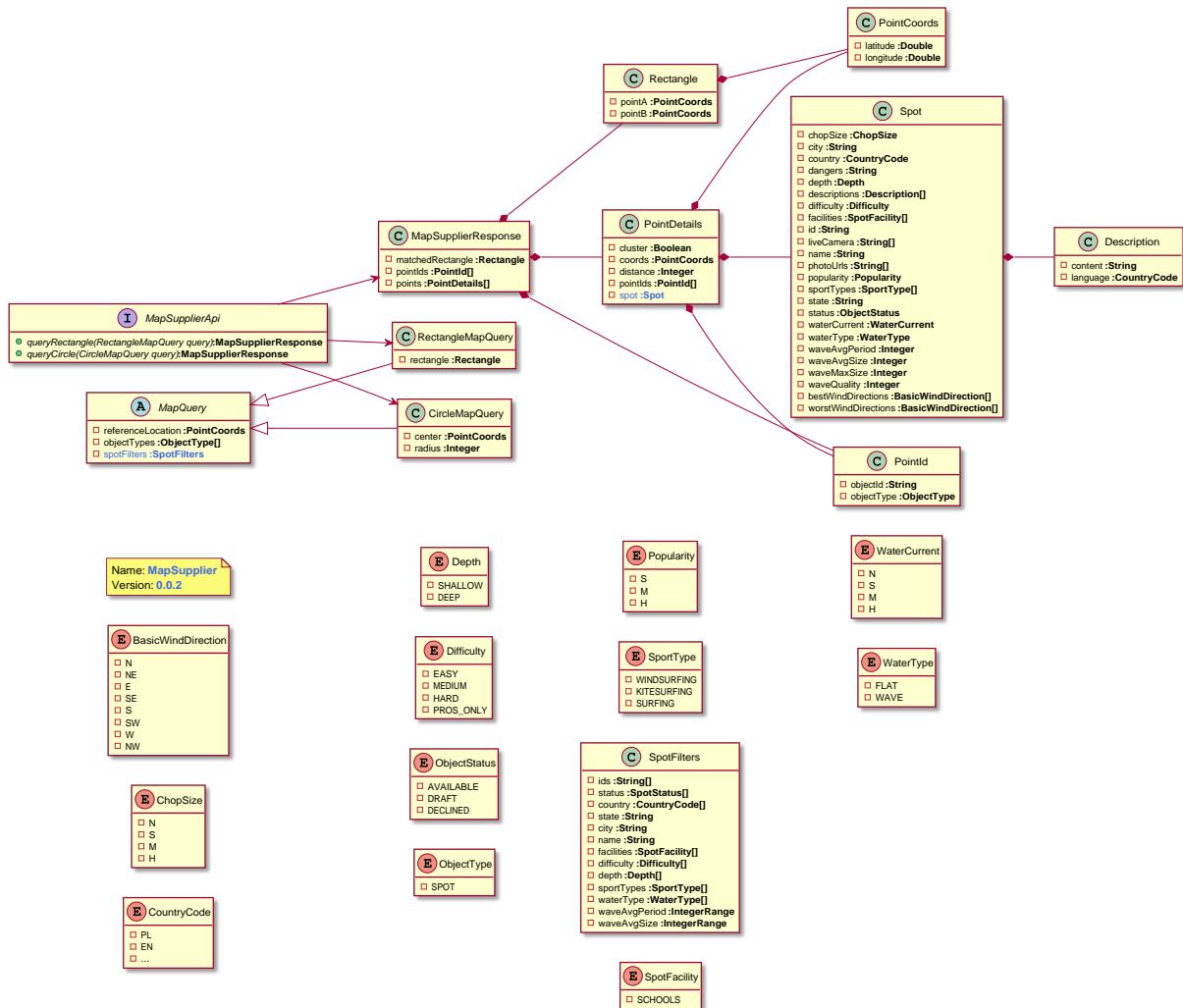
4.5. Realizacja wymagań funkcjonalnych

4.5.1. Diagram klas *spot-service* API



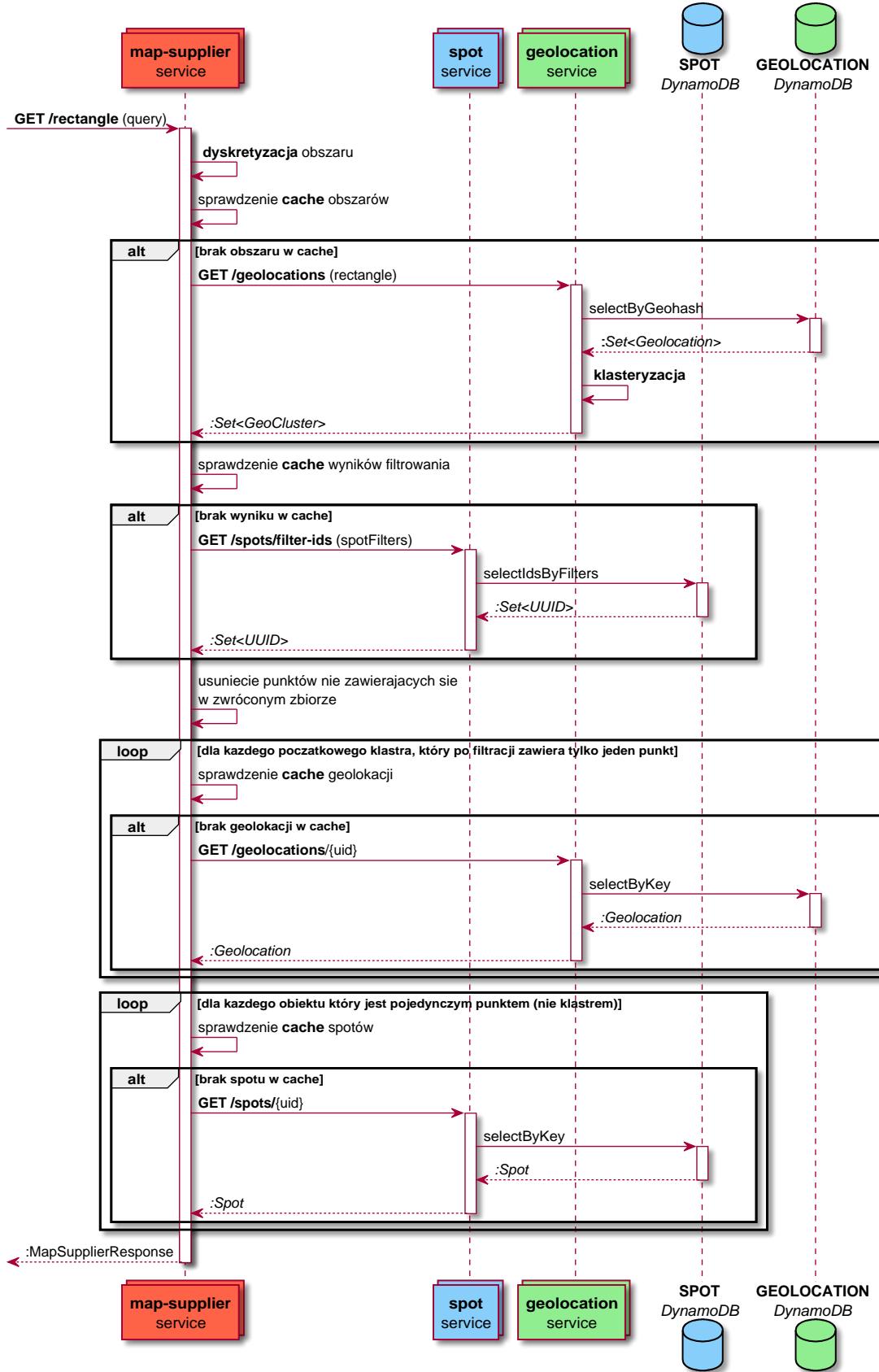
Rys. 4.10. Diagram klas *spot-service* API

4.5.2. Diagram klas *map-supplier* API



Rys. 4.11. Diagram klas *map-supplier* API

4.5.3. Zasilanie mapy spotów



Większość funkcjonalności zaimplementowanych w ramach tego projektu bierze udział podczas zasilania mapy spotów. Topologia domenowych Service'ów pokazana jest na diagramie 4.2. Poprzez analizę przebiegu zapytania o punkty na mapie poznamy głębiej poszczególne Service'y.

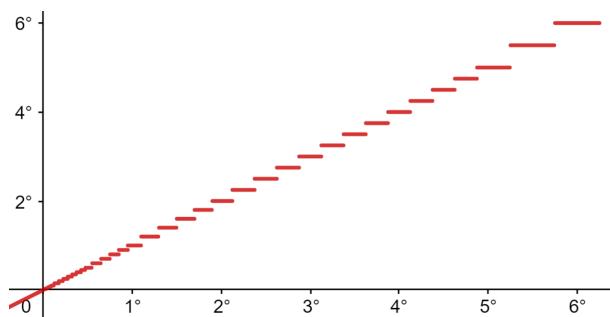
1. [map-supplier] - GET /rectangle (query)

nadchodzące od aplikacji mobilnej zapytanie jest typu RectangleMapQuery (4.11), zawiera:

- **rectangle** - w zamyśle obszar widoczny na mapie
- **objectTypes** - żądane typy obiektów, aktualnie są to tylko spoty, ale w przyszłości typów może być więcej
- **spotFilters** - filtry dla obiektów typu spot, w przyszłości w zapytaniu mogły by się pojawić filtry dla pozostałych typów
- **referenceLocation** - punkt na mapie (*w zamyśle położenie użytkownika*), wobec którego liczone są odległości zwracanych punktów

2. [map-supplier] - dyskretyzacja obszaru

Wartości szerokości i długości geograficznej jakie mogą nadjeść od klienta są praktycznie liniowe. Szansa na powtórzenie się obszaru jest bardzo niska. Stąd pojawiła się potrzeba przybliżenia tych wymiarów do najbliższych zdefiniowanych dyskretnych wartości. W ten sposób szansa potwórki jest znacznie większa. Oznacza to, że aplikacja częściej będzie mogła skorzystać z **cache'a**. Poniższy wykres przedstawia użytą funkcję dyskretyzacji, która zmniejsza swój stopień wraz z przybliżeniem na mapie.



Rys. 4.12. Funkcja dyskretyzacji kąta

3. [geolocation-service] - selectByGeohash

Geohash w dużym skrócie jest funkcją dyskretyzacji powierzchni globu. Zamienia długość i szerokość geograficzną na identyfikator wycinka Ziemi, w którym punkt się znajduje. W ten sposób poindeksowane są obiekty w tabeli GEOLOCATION. Znacznie przyspiesza to zapytania o punkty w zadanym obszarze.

4. *[geolocation-service]* - **klasteryzacja**

Wykonywana jest po stronie backendu, aby nie obarczać tym zadaniem aplikacji mobilnej.

W zależności od stopnia przybliżenia wykorzystywany jest jeden z trzech algorytmów:

- **Geohash** - klasteryzacja na podstawie Geohash'u
- **DBSCAN** - algorytm opierający się na gęstości punktów
- **k-means** - kieruje się średnią odległością dzielącą punkty

5. *[spot-service]* - **GET /spots/filter-ids** (spotFilters)

Geolocation-service zwraca sklasteryzowane identyfikatory obiektów w zadanym obszarze.

Teraz pora na filtrowanie tego zbioru po zadanach w zapytaniu parametrach. To zadanie należy do *spot-service*. Zwraca on zbiór identyfikatorów spełniających kryteria.

6. *[geolocation-service]* - **GET /geolocations/{uid}**

W przypadku gdy grupa identyfikatorów stanowiąca klaster zostanie zredukowana po filtrowaniu do pojedynczego punktu, wypada skorygować jego koordynaty. Wołany jest wtedy *geolocation-service* by pozyskać dokładny punkt.

7. *[spot-service]* - **GET /spots/{uid}**

Każdy pojedynczy punkt dekorowany jest skojarzonymi danymi spotu, klastry są pomijane.

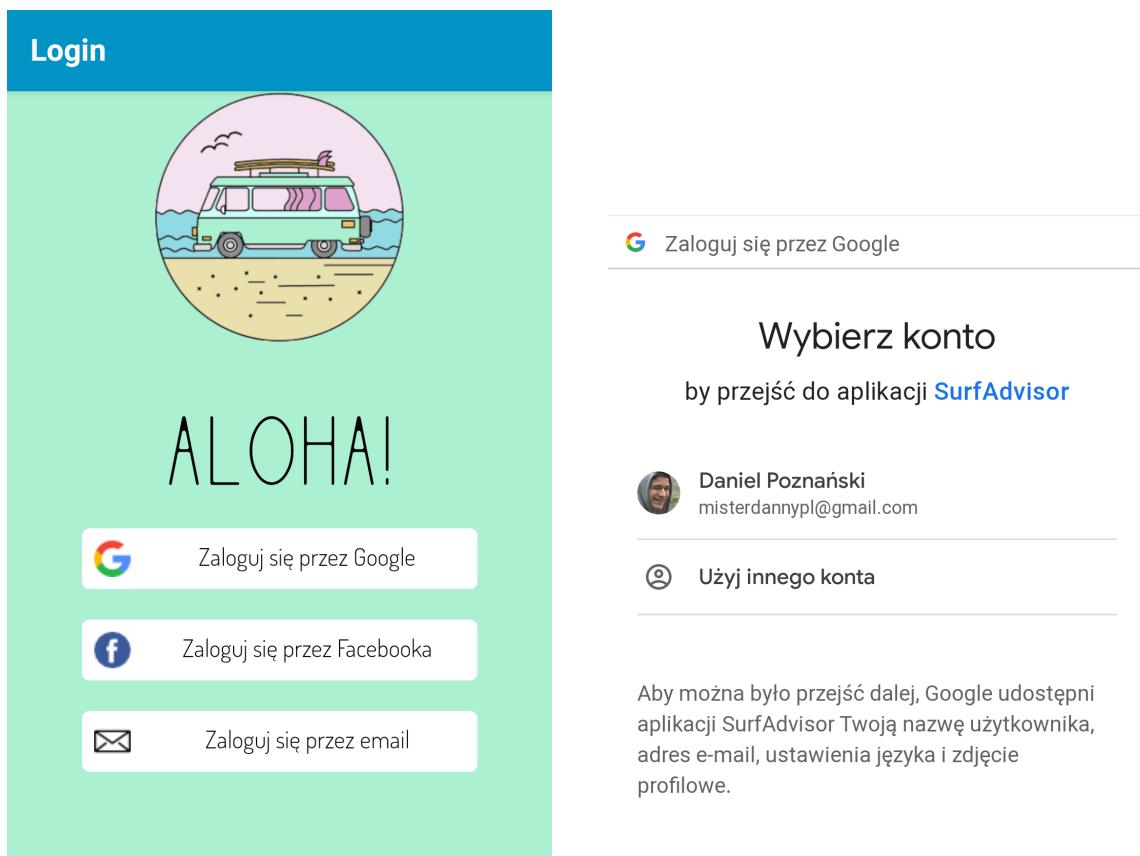
8. *[map-supplier]* - **MapSupplierResponse**

- **matchedRectangle** - użyty obszar po dyskretyzacji
- **points** - wynikowe obiekty - klastry lub pojedyncze spoty

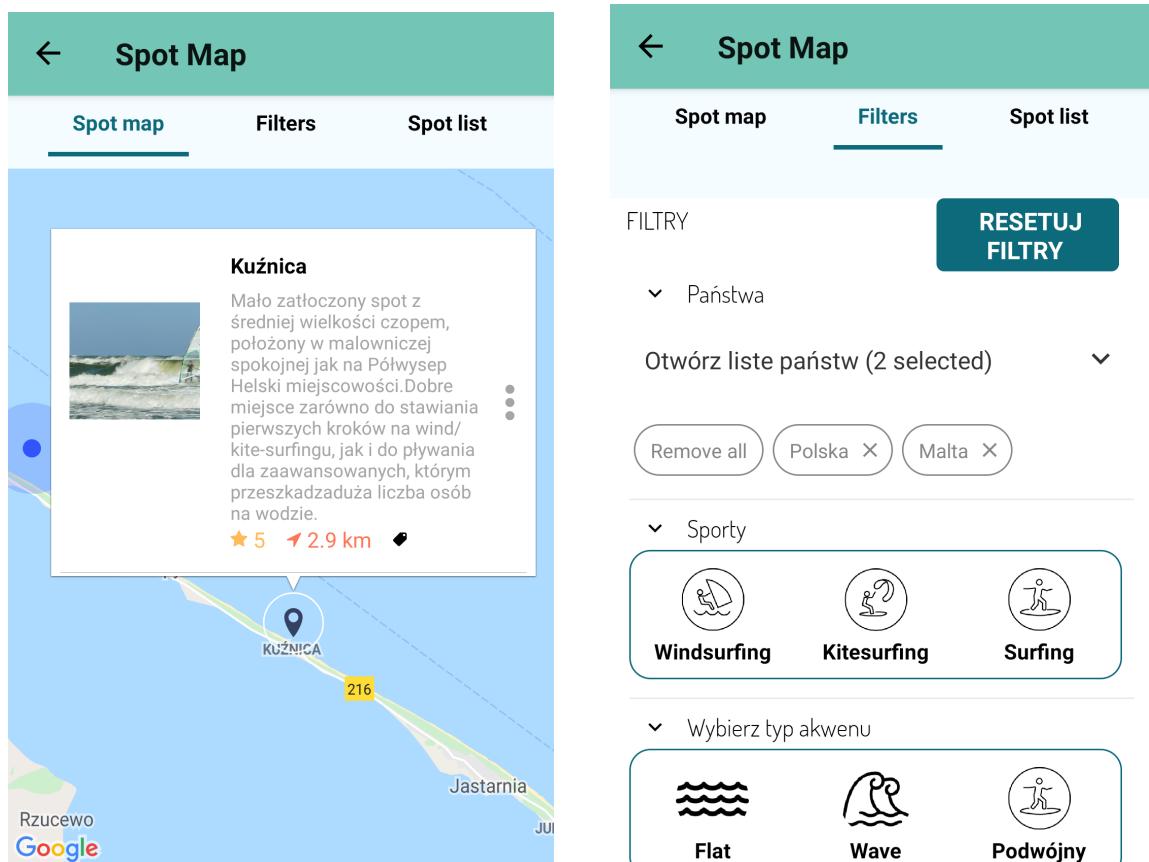
4.5.4. Aplikacja mobilna

Napisana z wykorzystaniem *React Native* - translacja kodu pisanej w *Javascript* na natywne elementy UI platform *Android* i *iOS*. W projekcie aplikacji autor niniejszej pracy zajmował się jedynie spinaniem jej z zewnętrznymi serwisami:

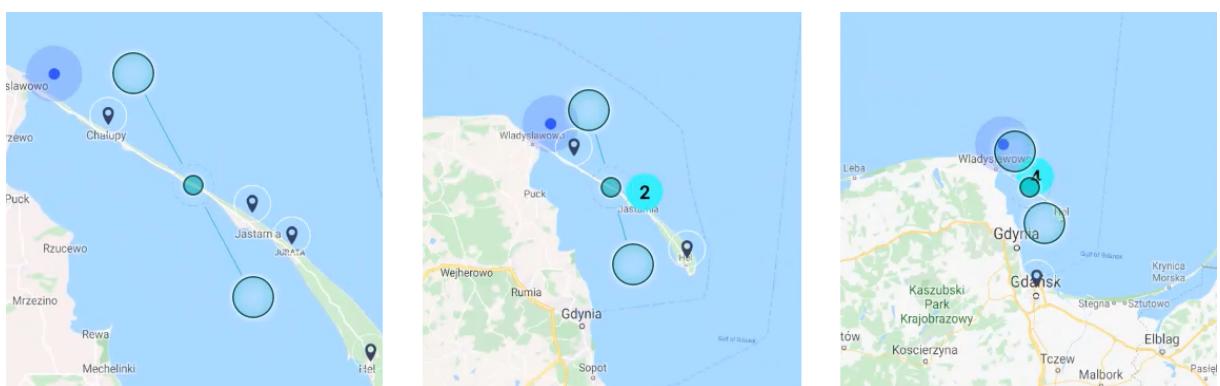
- *Google* - logowanie
- *Firebase* - autentykacja z użyciem tokenu od *Google*
- *thesurfadvisor.com* - nowe usługi postawione na AWS (4.2)



Rys. 4.13. Ekran logowania wraz z przekierowaniem do Google



Rys. 4.14. Mapa spotów wraz z zakładką filtrów



Rys. 4.15. Prezentacja klasteryzacji spotów na mapie

5. Autoskalowanie

Cluster SurfAdvisor skaluje się w dwóch wymiarach:

- Pod - **Horizontal Pod Autoscaling**
ilość Pod'ów (*instancji*) danej aplikacji
- Node - **Horizontal Node Autoscaling**
ilość Node'ów (*wirtualnych maszyn*) składających się na Cluster

5.1. Horizontal Pod Autoscaling

Autoskalowanie Pod'ów definiuje się poprzez skojarzony obiekt `HorizontalPodAutoscaler`. Kluczowe znaczenie mają właściwości:

- minimalna ilość Pod'ów
- maksymalna ilość Pod'ów
- metryka, która wzbudza skalowanie

Aby skalowanie było możliwe, definicja Pod'a musi zawierać wymagania co do zużycia zasobów (*CPU, RAM*).

Działanie autoskalowania zaprezentowane zostanie podczas testu obciążeniowego *dictionary-service*. W stanie spoczynkowym liczba Pod'ów tego Service'u wynosi **1**, maksymalnie może osiągnąć **10**, skalowanie wzbudzane jest przekroczeniem **50%** zużycia **CPU**.

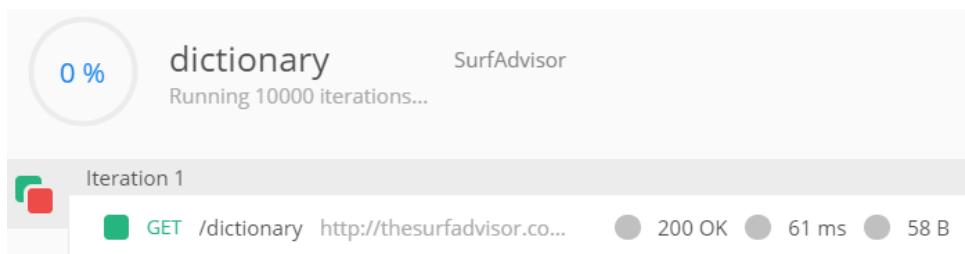
➡ Autoscalers 2					
<input type="checkbox"/>	NAME ▾	TYPE	READY	STATUS	AGE
<input type="checkbox"/>	➡ dictionary	HorizontalPod...	● 1/1	Target 50% / Min 1 / Max 10	2 hours ago
<input type="checkbox"/>	➡ geolocation-service	HorizontalPod...	● 1/1	Target 50% / Min 1 / Max 2	2 hours ago

Rys. 5.1. Spoczynkowy stan Autoscaler'ów

<input type="checkbox"/> NAME ▾	TYPE	READY	STATUS	AGE
<input type="checkbox"/> dictionary-9578ff775-mijct	Pod	<div style="width: 100%;">2/2</div>	Running	2 hours ago
<input type="checkbox"/> firebase-auth-connector-9d...	Pod	<div style="width: 100%;">2/2</div>	Running	2 hours ago
<input type="checkbox"/> geolocation-service-8ddb58...	Pod	<div style="width: 100%;">2/2</div>	Running	50 minutes...
<input type="checkbox"/> map-supplier-5b8fdff77f-lb6sj	Pod	<div style="width: 100%;">2/2</div>	Running	2 hours ago
<input type="checkbox"/> spot-service-69d5b98b79-...	Pod	<div style="width: 100%;">2/2</div>	Running	2 hours ago

Rys. 5.2. Spoczynkowy stan Pod'ów

W ramach testu obciążeniowego wysyłany jest za pomocą programu *Postman* nieprzerwany szturm 10 tysięcy zapytań GET o treść jakiegoś słownika:



Rys. 5.3. Test obciążeniowy *dictionary-service* z Postmana

Autoscaler'y reagują po około 10 sekundach zwiększając liczbę Pod'ów *dictionary-service*:

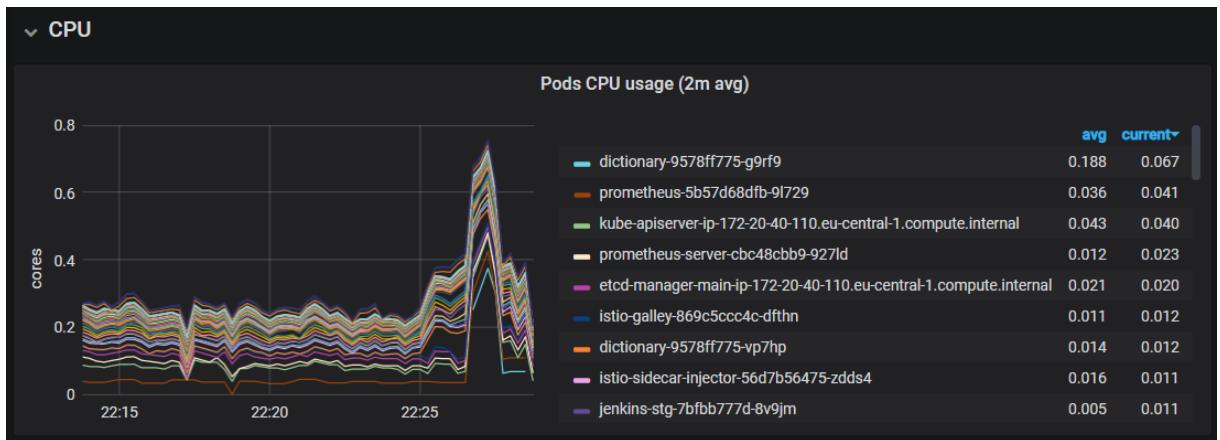
<input type="checkbox"/> NAME ▾	TYPE	READY	STATUS	AGE
<input type="checkbox"/> dictionary-9578ff775-jqg7t	Pod	<div style="width: 50%;">1/2</div>	Running	8 seconds ...
<input type="checkbox"/> dictionary-9578ff775-llhxp	Pod	<div style="width: 50%;">1/2</div>	Running	9 seconds ...
<input type="checkbox"/> dictionary-9578ff775-mijct	Pod	<div style="width: 100%;">2/2</div>	Running	3 hours ago
<input type="checkbox"/> dictionary-9578ff775-ts7t8	Pod	<div style="width: 50%;">1/2</div>	Running	8 seconds ...
<input type="checkbox"/> firebase-auth-connector-...	Pod	<div style="width: 100%;">2/2</div>	Running	3 hours ago
<input type="checkbox"/> geolocation-service-8ddb...	Pod	<div style="width: 100%;">2/2</div>	Running	1 hour ago
<input type="checkbox"/> map-supplier-5b8fdff77f-l...	Pod	<div style="width: 100%;">2/2</div>	Running	3 hours ago
<input type="checkbox"/> spot-service-69d5b98b79-...	Pod	<div style="width: 100%;">2/2</div>	Running	3 hours ago

Rys. 5.4. Stan Pod'ów przy skalowaniu w górę

<input type="checkbox"/>	NAME ▾	TYPE	READY	STATUS	AGE
<input type="checkbox"/>	↳ dictionary	HorizontalPod...	● 4/4	Target 50% / Min 1 / Max 10	2 hours ago
<input type="checkbox"/>	↳ geolocation-service	HorizontalPod...	● 1/1	Target 50% / Min 1 / Max 2	2 hours ago

Rys. 5.5. Stan Autoscalerów przy skalowaniu w górę

Diagram poniżej pochodzi z konsoli webowej *Grafana* - narzędzia, które wspólnie z *Prometheus* umożliwiają śledzenie metryk technicznych Cluster'a. Przedstawione zostało zużycie CPU poszczególnych Podów podczas testu obciążeniowego:



Rys. 5.6. Zużycie CPU podczas testu obciążeniowego *dictionary-service*

5.2. Horizontal Node Autoscaling

Z kolei włączenie autoskalowania Node'ów jest jeszcze prostsze. Ogranicza się do wdrożenia jednej z dostępnych online rekomendowanych definicji Service'u *cluster-autoscaler*. Skalowanie pobudzane jest ilością Podów jakie Kubernetes próbuje wdrożyć. Jeżeli istniejące Node'y nie pomieszczą już więcej Podów, tworzone są kolejne.

Ilość instancji EC2 w stanie początkowym:

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
<input type="checkbox"/>	nodes.surfad...	i-00e39a5cc5b8db180	t3.medium	eu-central-1b	running	✓ 2/2 checks ...
<input type="checkbox"/>		i-052730ffadaf77874	t2.micro	eu-central-1a	running	✓ 2/2 checks ...
<input type="checkbox"/>	nodes.surfad...	i-0d7417f9fbe9250dd	t3.medium	eu-central-1a	running	✓ 2/2 checks ...
<input type="checkbox"/>	master-eu-ce...	i-0f4d2fd2e5fa2dae3	m3.medium	eu-central-1a	running	✓ 2/2 checks ...

Rys. 5.7. Spoczynkowy stan Node'ów

Poprzez manualne rozkazy zwiększania ilości Pod'ów doprowadzono do przepelnienia Cluster'a. W logach Service'u *cluster-autoscaler* zarejestrowana jest reakcja na zastąpę sytuacji:

```

Pods > cluster-autoscaler-847ccc5576-v6cx6
>Status Specifications View Labels Logs
cluster-autoscaler
1 scale_up.go:199] Best option to resize: nodes.surfadvisor.k8s.local
1 scale_up.go:203] Estimated 1 nodes needed in nodes.surfadvisor.k8s.local
1 scale_up.go:292] Final scale-up plan: [{nodes.surfadvisor.k8s.local 2->3 (max: 5)}]
1 scale_up.go:344] Scale-up: setting group nodes.surfadvisor.k8s.local size to 3
1 aws_manager.go:305] Setting asg nodes.surfadvisor.k8s.local size to 3

```

Rys. 5.8. Logi serwisu odpowiedzialnego za skalowanie horyzontalne Node'ów

Wynikiem tego działania jest uruchomienie dodatkowej instancji EC2, która rozszerzy pojemność Cluster'a i umożliwi wdrożenie wszystkich Pod'ów:

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
<input type="checkbox"/>	nodes.surfad...	i-00e39a5cc5b8db180	t3.medium	eu-central-1b	running	✓ 2/2 checks ...
<input type="checkbox"/>		i-052730ffadaf77874	t2.micro	eu-central-1a	running	✓ 2/2 checks ...
<input type="checkbox"/>	nodes.surfad...	i-093f4249d9cd85b09	t3.medium	eu-central-1b	running	⌚ Initializing
<input type="checkbox"/>	nodes.surfad...	i-0d7417f9fbe9250dd	t3.medium	eu-central-1a	running	✓ 2/2 checks ...
<input type="checkbox"/>	master-eu-ce...	i-0f4d2fd2e5fa2dae3	m3.medium	eu-central-1a	running	✓ 2/2 checks ...

Rys. 5.9. Stan Node'ów przy skalowaniu w góre

6. Podsumowanie

W ramach niniejszej pracy inżynierskiej zbudowany został system, na którym postawiono nowe usługi pod przyszłą wersję aplikacji SurfAdvisor. Planowo skorzystają z niej użytkownicy latem 2020 roku zarówno na platformie *Android* jak i *iOS*. Podczas korzystania z mapy spotów może zostać określona szeroka gama filtrów. Nowy Cluster spełnia dodatkowo zadane wymagania niefunkcjonalne będąc gotowym na wzmożony ruch. Autentykacja i autoryzacja nowych usług jest połączona z pierwotnym serwerem *Firebase* pomimo faktu różnych dostawców chmur (*AWS* i *GCP*). Cała infrastruktura jest wygodna w utrzymaniu dzięki wykorzystaniu narzędzi wspierających *IaC*. Rolę programisty w tym projekcie dodatkowo ułatwia wdrożenie serwisów takich jak *Jenkins* czy *Grafana*.

Połączenie produktów *Amazon Web Services* i orkiestracji *Kubernetes* tworzy łatwe w utrzymaniu i elastyczne w wymiarach środowisko. Synteza jest jeszcze prostsza, gdy zamiast tworzenia Cluster'a na własną rękę poprzez *kops* CLI, zdecydujemy się na oficjalne rozwiązanie jakim jest *AWS EKS*.

Należy mieć jednak świadomość, że uruchomione całodobowo maszyny wirtualne tworzące nasz system nie są darmowe. Minimalny sensowny zestaw instancji *EC2* wraz z *LoadBalancer*'em kosztować będzie w przybliżeniu ponad 140 dolarów miesięcznie. Cenę tę można zredukować zobowiązując się do korzystania z instancji przez dłuższy okres (*1-3 lata*), ale nie więcej niż 50%. Sytuacja ma się podobnie u pozostałych dostawców usług chmurowych.

Dla mniejszych projektów znacznie lepiej sprawdzą się platformy *serverless*. Główna różnica polega na tym, że aplikacja uruchomiona jest tylko wtedy, gdy nadchodzi żądanie od klienta i tylko ten czas jest opłacany. Aplikacje te są z reguły lekkie tak by mogły szybko wstać i obsłużyć zapytanie. Najczęściej przybierają formę funkcji *Javascript* (*AWS Lambda*, *Google Cloud Function*). W oparciu o nie można zbudować sieć jednostek ludząco podobną do mikroserwisów.

Inną formą *serverless* jest korzystanie z *Firebase* (*Google*) lub *AWS Amplify*. Oba narzędzia zapewniają m.in. już gotowe podstawowe usługi baz danych, autentykacji i powiadomień. To co wykracza poza ich możliwości można zaimplementować za pomocą wcześniej opisanych funkcji.

Charakterystyczną cechą pracy z rozwiązaniami *serverless* jest brak troski o to jak kod jest uruchamiany. Zmartwienie to należy do dostawcy chmury, podobnie jak kwestia autoskalowania.

Bibliografia

- [1] *Cloud computing with AWS.* <https://aws.amazon.com/what-is-aws/>. Amazon Web Services, 2019.
- [2] *Amazon.com Case Study.* <https://aws.amazon.com/solutions/case-studies/amazon/>. Amazon Web Services.
- [3] *Overview of Amazon Web Services.* <https://d1.awsstatic.com/whitepapers/aws-overview.pdf>. Amazon Web Services, 2019.
- [4] Ravi Murty. *Powering next-gen Amazon EC2: Deep dive into the Nitro system.* https://d1.awsstatic.com/events/reinvent/2019/REPEAT_2_Powering_next-gen_Amazon_EC2_Deep_dive_into_the_Nitro_system_CMP303-R2.pdf. Amazon Web Services, 2019.
- [5] *What is a Container?* <https://www.docker.com/resources/what-container>. Docker, Inc.
- [6] *About images, containers, and storage drivers.* <https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/>. Docker, Inc.
- [7] *What is Kubernetes.* <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. Cloud Native Computing Foundation (CNCF).
- [8] Aleksei Burlutskii. *Kubernetes for dummies.* <https://www.burlutsky.su/management/kubernetes-for-begginner/>.
- [9] *Kubernetes Concepts.* <https://kubernetes.io/docs/concepts>. Cloud Native Computing Foundation (CNCF).
- [10] *Traffic Management.* <https://istio.io/docs/concepts/traffic-management/>. Istio Authors.
- [11] *What is Istio?* <https://istio.io/docs/concepts/what-is-istio/>. Istio Authors.
- [12] Sam Guckenheimer. *What is Infrastructure as Code?* <https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code>. Microsoft.
- [13] *Installing Kubernetes with kops.* <https://kubernetes.io/docs/setup/production-environment/tools/kops/>. Cloud Native Computing Foundation (CNCF).
- [14] *Amazon EKS pricing.* <https://aws.amazon.com/eks/pricing/>. Amazon Web Services.