Capstone

Weronika Gozdera, Alina Hryshko, Wojciech Domisiewicz

May 2024

1 *Opis wstępny*

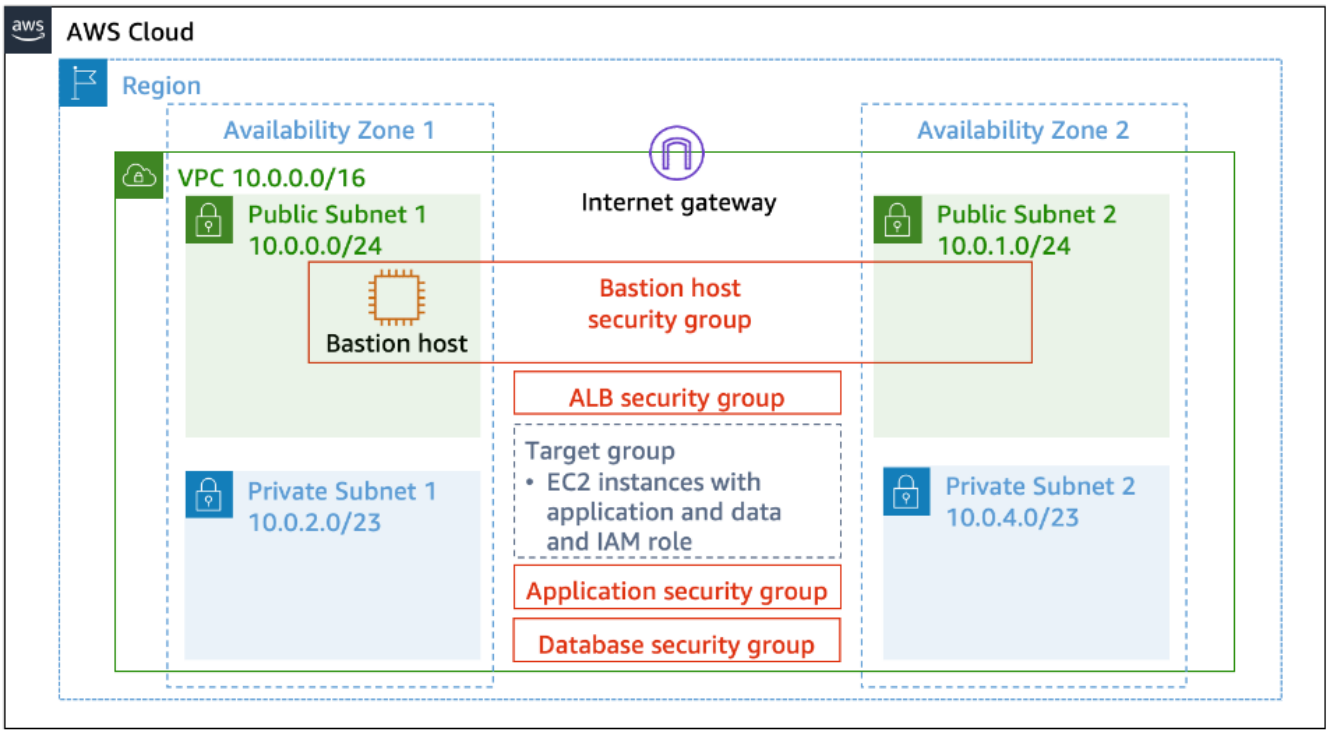
Naszym zadaniem jest dostarczenie aplikacji PHP, która spełnia następujące wymagania:

1. Wdrożenie aplikacji PHP, która działa na instancji *Amazon Elastic Compute Cloud* (Amazon EC2)
2. Utworzenie instancji bazy danych, z którą aplikacja PHP może się łączyć
3. Utworzenie bazy danych MySQL oraz jej zabezpieczenie.
4. Zabezpieczenie aplikacji w celu zapobiegania publicznemu dostępowi do systemów backendowych
5. Użycie *Load Balancer* oraz *Auto Scaling Group*.
6. Umożliwić bezpieczny dostęp dla użytkownika administracyjnego.
7. Umożliwić anonimowy dostęp użytkownikom serwisu.

Następujące zasoby są już dostarczone na początku zadania:

1. Przykładowy VPC (*Virtual Private Cloud*)
2. 4 podsieci, 2 prywatne z tabelami routingu skonfigurowanymi do łączenia się z zasobami w ramach VPC oraz 2 publiczne podsieci z trasami do IGW (*Internet Gateway*).
3. 4 *Security Groups*, jedna dla aplikacji, jedna dla bazy danych, jedna dla *Load Balancer* i jedna dla Bastion Host
4. Instancja EC2 Bastion Host

Niżej przedstawiony jest diagram architektury początkowej:



Rysunek 1: Architektura dostarczona na początku zadania.

2 *Opis poszczególnych komponentów architektury*

Regiony i strefy dostępności: AWS jest podzielona na regiony i strefy dostępności. Regiony to geograficzne obszary, w których znajdują się centra danych AWS. Strefy dostępności to oddzielne fizyczne lokalizacje w obrębie regionu, które są połączone ze sobą szybkimi połączeniami sieciowymi. Architektura przedstawiona na rysunku wykorzystuje dwie strefy dostępności w ramach jednego regionu (*Availability Zone 1* i *Availability Zone 2*). Zapewniają redundancję i wysoką dostępność. Oznacza to, że jeśli jedna strefa dostępności stanie się niedostępna, aplikacja nadal będzie działać w pozostałych strefach.

Sieci VPC: VPC (Virtual Private Cloud) to prywatna sieć wirtualna w chmurze AWS. Zapewniają izolację zasobów sieciowych i ochronę przed nieautoryzowanym dostępem. W naszej architekturze używana jest jedna sieć VPC z adresem 10.0.0.0/16.

Podsieci prywatne: Podsieci prywatne to podziały sieci VPC na mniejsze, izolowane segmenty. W tej architekturze używane są dwie podsieci prywatne: *Private Subnet 1* (10.0.2.0/24) i *Private Subnet 2* (10.0.4.0/24). Podsieci prywatne umożliwiają oddzielenie różnych typów ruchu sieciowego. Ruch ten można skonfigurować i kontrolować za pomocą *Security Groups*. To zapewnia bezpieczeństwo, ścisłe kontrolowanie dostępu do zasobów sieciowych i ochronę ich przed nieautoryzowanym dostępem.

Podsieci publiczne: Podsieci publiczne to podziały sieci VPC, które są dostępne z publicznego Internetu. W tej architekturze używane są dwie podsieci publiczne: *Public Subnet 1* (10.0.0.0/24) i *Public Subnet 2* (10.0.1.0/24). W *Public Subnet 1* jest uruchomiony Bastion Host.

Internet Gateway: *Internet Gateway* to narzędzie sieciowe, które umożliwia łączenie zasobów architektury z publicznym Internetem. W tej architekturze używany jest jeden *Internet Gateway*.

Bastion host: Bastion host to serwer wirtualny uruchomiony w podsieci publicznej sieci VPC

(Virtual Private Cloud) w chmurze AWS. Służy jako punkt dostępu do zasobów sieciowych w podsieciach prywatnych. Bastion host zapewnia bezpieczny sposób łączenia się z instancjami EC2 bez narażania ich na bezpośredni dostęp z publicznego Internetu (możemy powiedzieć, że jest to "most" pomiędzy naszymi instancjami a internetem).

Bastion host zmniejsza ryzyko ataku na instancje EC2 w podsieciach prywatnych, ponieważ te instancje nie są bezpośrednio dostępne z Internetu. Zamiast tego cały ruch sieciowy do tych instancji przechodzi przez bastion host, który można zabezpieczyć za pomocą silnych mechanizmów bezpieczeństwa, takich jak *Security Groups*.

Bastion host ułatwia zarządzanie dostępem do instancji EC2 w podsieciach prywatnych. Zamiast konfigurować dostęp do każdej instancji z osobna, można skonfigurować dostęp do bastion hosta, a następnie używać go do łączenia się z innymi instancjami.

Application Load Balancer (ALB): ALB to usługa równoważenia obciążenia, która rozkłada ruch sieciowy między instancjami EC2. W tej architekturze używany jest jeden ALB. ALB zapewnia dostępność aplikacji nawet w przypadku dużego ruchu i w przypadku awarii jednego lub kilku serwerów. ALB zmniejsza opóźnienia i poprawia wydajność aplikacji, co poprawia wrażenie klienta od aplikacji.

Instancje EC2: Instancje EC2 to wirtualne serwery w chmurze AWS. W tej architekturze używane są trzy instancje EC2: jedna, która pełni rolę Bastion Host oraz dwie, między którymi *Load Balancer* rozkłada ruch sieciowy.

RDS Database: Baza danych RDS zapewnia skalowalną, zarządzaną bazę danych w chmurze. Główna instancja bazy danych znajduje się w *Private Subnet 1*. Replika bazy danych RDS, która znajduje się w *Private Subnet 2*, zapewnia kopię zapasową bazy danych w czasie rzeczywistym, która może być używana do odtwarzania danych w przypadku awarii. Replikacja umożliwia również skalowanie odczytu bazy danych poprzez kierowanie ruchu odczytu do repliki. Do baz danych przypisana jest specjalna *Security Group*, która odpowiada za bezpieczeństwo danych i kontroluje dostęp do instancji baz danych.

Auto Scaling Group: *Auto Scaling Group* automatycznie skaluje liczbę uruchomionych instancji EC2 w celu dopasowania do zmiennego obciążenia. Użycie *Auto Scaling Group* pozwala na zmniejszenie kosztów, zwiększenie dostępności aplikacji oraz możliwość automatycznego skalowania zasobów (instancji EC2).

3 *Rozwiązanie*

Rozwiązanie zaczęliśmy od skonfigurowania bramki NAT, wybraliśmy następujące opcje:

- Subnet - wybraliśmy publiczną podsieć
- Connectivity type - Public

Ten krok wykonujemy dla obu podsieci publicznych. Następnie zaktualizowaliśmy tablice rutingu, aby używały utworzonej bramki.

- Source - 0.0.0.0/0
- Target - local
- Destination - NAT Gateway

Ten krok wykonujemy dla obu bramek NAT Tworzymy Load Balancera, wybraliśmy Application Load Balancer. Wybraliśmy następujące opcje:

- Schemat - internet facing
- IP address type - IPv4
- VPC - ustawiliśmy otrzymany VPC
- Mappings - ustawiliśmy publiczne podsieci
- Security groups - wybraliśmy dostarczony dla Load Balancera security group
- Listeners and routing - wybraliśmy protokół http na porcie 80 i stworzyliśmy security group z takimi samymi ustawienia na protokole HTTP1

Następnie utworzyliśmy Auto Scalling group, wybraliśmy następujące opcje

- Launch Template - ustawiliśmy dostarczony Launch Template
- Network - wybraliśmy przykładowy VPC oraz 2 podsieci prywatne
- Load balancing - wybraliśmy wcześniej utworzoną grupę
- Group size - maximum capacity ustawliśmy na dwa
- wyłączyliśmy scale-in protection
- resztę rzeczy zostawiliśmy bez zmian

Następnym krokiem było utworzenie DB subnet group, aby móc utworzyć RDS. Tutaj wybraliśmy następujące pola

- VPC - przykładowe VPC
- Subnets - 2 podsieci prywatne

Po tym kroku przeszliśmy do utworzenia bazy danych.

- Choose a database creation method - wybraliśmy standardowe tworzenie
- Engine type - wybraliśmy MySQL
- Templates - wybraliśmy Dev/Test
- Availability and durability - wybraliśmy Multi-AZ DB instance
- Instance configuration - wybraliśmy Burstable classes
- Storage type - wybraliśmy General Purpose SSD (gp2)
- Allocated storage - wybraliśmy 20 GB
- Storage autoscaling - zaznaczyliśmy Enable storage autoscaling
- Compute resource - zaznaczyliśmy Don't connect to an EC2 compute resource

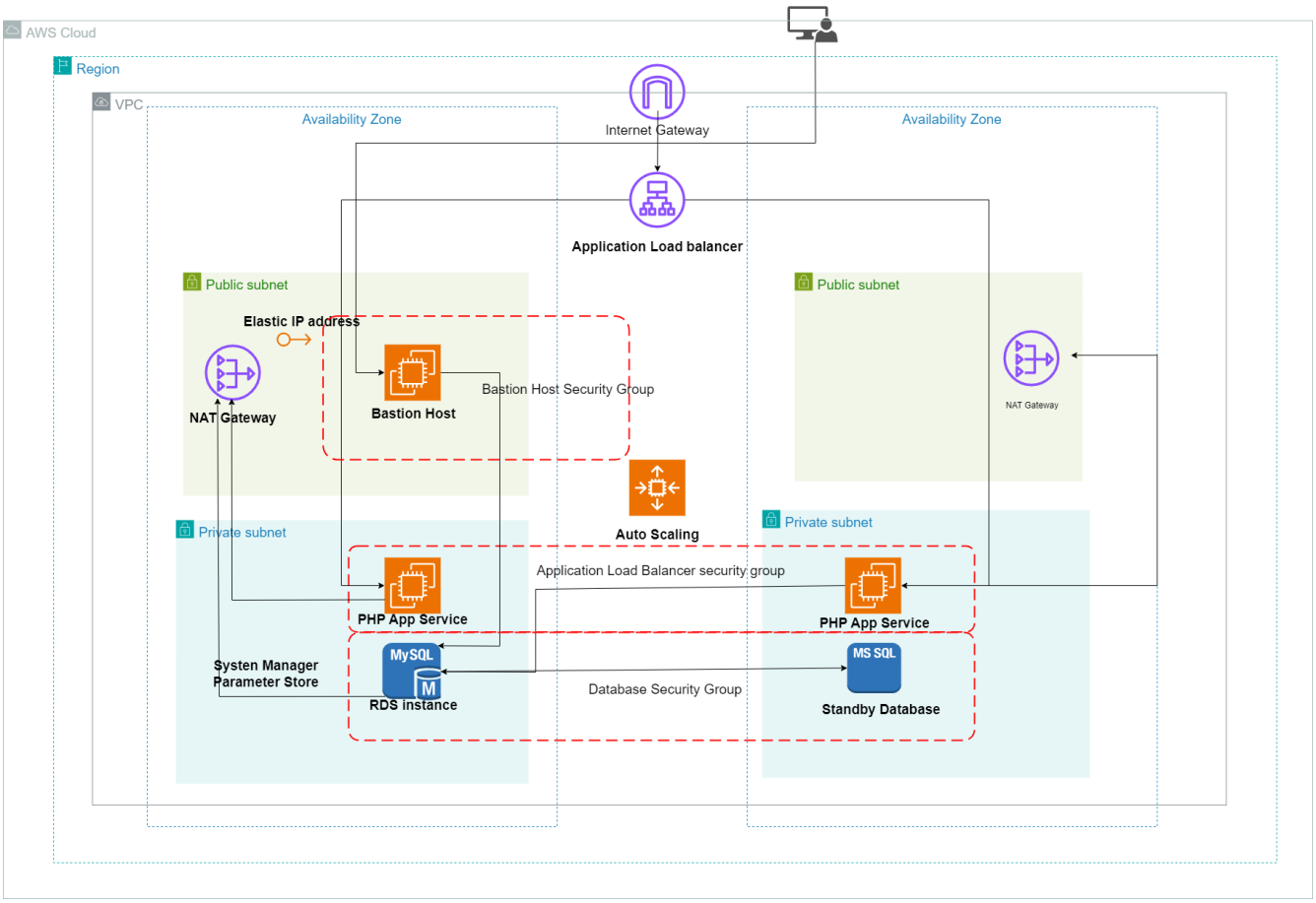
- VPC - wybraliśmy przykładowy VPC
- DB subnet group - wybraliśmy wcześniej utworzoną grupę
- Public access - zaznaczyliśmy no
- VPC security group (firewall) - wybraliśmy dostarczony Example-DB
- Encryption - włączone
- Automatic backup - wyłączone

Następnie przeszliśmy do Parameter Store i utworzyliśmy następujące parametry.

- /example/database z wartością CapstoneDb
- /example/username z wartością admin
- /example/password z wartością będącą hasłem RDS
- /example/endpoint z wartościa będacą endpointem naszej RDS

Następnie przeszliśmy do Instance i połączyliśmy się z Bastion. Otworzyła się konsola, w która wpisywaliśmy następujące polecenia

- $ sudo yum update -y
- zapisaliśmy nasz klucz prywatny RSA jako labsuser.pem
- $ chmod 400 labsuser.pem
- $ ssh -i labsuser.pem ec2-user@<private ip address>, aby ta komenda się powiodła musieliśmy dodać inbound rules dla ssh i Bastion Security Group
- $ mysql -u username -p –hostrds endpoint database name < name of SQL dump file



Rysunek 2: Diagram ilustrujący rozwiązanie.

Zalety tak zbudowanej architektury:

1. Zapewnienie wysokiej dostępności do aplikacji (ALB)

2. Baza danych jest zabezpieczona przed niechcianym dostępem, ponieważ baza danych oraz jej replika znajdują się w różnych *Private Subnet*, do tego użyte zostały *Security Groups* dla bezpiecznego dostępu.

3. Informacje, które są potrzebne do dostępu do bazy danych, przechowywane są w Magazynie Parametrów (*Parametr Store*). Działa on jak scentralizowany magazyn typu klucz-wartość, w którym można definiować nazwy kluczy i przypisywać im wartości tekstowe. Te wartości są następnie pobierane przez aplikacje uruchomione na usługach AWS.

4. Zapewnienie ochrony instancji oraz danych, ponieważ używamy *Security Group*, innej dla każdego rodzaju zasobów.

5. Aplikacji jest utrzymywana w jednej podsieci VPC, co pozwala na szybką komunikację pomiędzy komponentami. Poza tym, w przypadku awarii jednej strefy dostępności, mamy pewność, że aplikacja będzie działać dalej, ponieważ istnieją działające zasoby w innej strefie dostępności.

6. Posiadanie Bastion Host zabezpiecza nasze instancje przed nieautoryzowanym dostępem.

4 *Disaster recovery*

Pierwszy krokiem w zabezpieczeniu się przed brakiem dostępu do niektórych zasobów naszej aplikacji jest oczywiście zapewnienie jej wysokiej dostępności. Nasze rozwiązanie przewiduje odporność systemu na awarię jednej *Availability Zone'y* poprzez replikację zasobów. W szczególności zdecydowaliśmy się na zastosowanie *Multi-AZ deploymentu* naszej bazy danych. Oznacza to, że poza główną bazą danych utworzona została również instancja rezerwowa, do której synchronicznie replikowane są wszystkie dane. Tym sposobem w razie niesprawności instancji podstawowej jej funkcje mogą być automatycznie przejęte przez zastępczą bazę danych. Koniecznym dla uzyskania wysokiej dostępności było również uruchomienie dwóch instancji *EC2* w różnych *Availability Zone'ach*, które w standardowych warunkach mogą równolegle obsługiwać ruch w aplikacji, zarządzane *Load Balancerem*. W sytuacji kryzysowej wspomniany *Load Balancer* będzie zaś kierował cały ruch na jedną instancję *EC2*, wyłączając niesprawną instancję z obsługi.

Należy jednak zdawać sobie sprawę, że realnym zagrożeniem dla aplikacji, zasługującym na miano *disaster*, tzn. *katastrofy* nie jest niedostępność jednego zasobu czy nawet *Availability Zone'y*, ale awaria obejmująca cały region. W celu zabezpieczenia się przed tego typu dysfunkcją należy zastosować jednej z czterech wzorców *disaster recovery* spośród:

- *Backup and restore*,
- *Pilot light*,
- *Warm standby*,
- *Mulit-site*.

Każda kolejna strategia gwarantuje coraz lepsze parametry przywracania aplikacji do funkcjonowania, definiowane jako:

- *Recovery point objective* (*RPO*) – maksymalna akceptowalna strata danych mierzona w czasie,
- *Recovery time objective* (*RTO*) – maksymalny akceptowalny czas po awarii przez który aplikacja może pozostać wyłączona z użytku.

Niestety, wraz ze wzrostem jakości idzie również oczywiście wzrost kosztów, a jako niewielka organizacja *non-profit* nie możemy sobie pozwolić na przeznaczanie znaczących środków w tym obszarze. Musieliśmy zatem zdecydować się na pierwszy z wymienionych wzorców, kosztem wzrostu *RPO* i *RTO* oraz koniecznością rezygnacji z automatyzacji możliwej do uzyskania poprzez utrzymywanie przynajmniej częściowo gotowych rozwiązań w innych regionach.

Strategia *Backup and restore* to przede wszystkim skupienie się nad zabezpieczeniem przed utratą danych poprzez umiejscowienie ich kopii w innym regionie, niż aplikacja. W przypadku zasobów naszej aplikacji trzeba więc określić możliwości odtworzenia instancji *EC2* oraz *RDS*. *Amazon RDS* oferuje możliwość przechowywania *snapshotów* bazy danych z danego regionu, w innym regionie. Jest to rozwiązanie dokładne i umożliwiające stosunkowo szybkie odtworzenie instancji w razie awarii, jednak trzeba wziąć pod uwagę fakt, że wykonanie takiego rodzaju zrzutu danych nie przebiega w sposób natychmiastowy. W celu zapewnienia spójności i poprawności *snapshotu* należy czasowo zablokować możliwość wykonywania transakcji na danej instancji, a zatem nie możemy wykonywać takich kopii dowolnie często – optymalnym czasem mogłoby być np. raz

na dobę, w okresie najniższej aktywności na stronie (albo rzadziej w zależności od tego, jak często przewidujemy zmiany w zawartości bazy danych). Uzupełnieniem takiej strategii zabezpieczenia jest przechowywanie logów transakcyjnych (w *S3* w *backupowym* regionie), które umożliwią odtworzenie zapytań do bazy danych od czasu wykonania ostatniego *snapshotu*. Uwzględniając charakter oferowanej usługi, opierający się jedynie na umożliwianiu użytkownikom odczytu danych w bazie można zakładać, że *backapowanie* instancji *RDS* będzie jedynie okazjonalne (kiedy organizacja pozyska nowe dane statystyczne).

W celu odtworzenia instancji *EC2* posłużymy się usługa *Amazon Machine Image*, która przechowuje wszystkie informacje niezbędne do uruchomienia instancji będącej odwzorowaniem tej, która uległa awarii. Należy również zadbać o właściwie zabezpieczenie danych zawartych w *Elastic Block Store'ze*. Aby to osiągnąć skorzystamy z *Amazon Data Lifecycle Managera*, który zarządza regularnym *backupowaniem* oraz usuwa przestarzałe *snapshoty* gwarantując zmniejszenie kosztów przechowywania. Wspomniane *snapshoty* są tworzone przyrostowo, tzn. wskazują jedynie zmiany dokonane od ostatniego zrzutu danych. Dzięki temu minimalizowany jest czas ich wykonania oraz nie tworzymy zbędnych duplikatów danych. Oczywiście miejscem przechowywania *snapshotów* jest S3 w *backupowym* regionie. W razie awarii możemy zatem odtworzyć daną instancję *EC2* na podstawie jej obrazu *AMI*, a następnie pobrać wszystkie niezbędne dane z odpowiedniego *bucketu S3*.

W przypadku awarii regionu nie wystarczą nam jednak same zrzuty danych i konfiguracji aplikacji. Brak dostępności regionu zmusza nas do odwzorowania całej infrastruktury rozwiązania w *backupowym* regionie. Próba manualnego odwzorowania wszystkich zasobów byłaby oczywiście przyczyną kolejnego, niepotrzebnego wzrostu *RTO*. Dobrą praktyką jest więc stosowanie *IaC* (*infrastructure as code*) umożliwiającego szybkie, automatyczne wdrażanie rozwiązania w nowym regionie. W tym celu skorzystamy z usługi *AWS Cloud Formation*, która służy do modelowania infrastruktury za pomocą pliku tekstowego, który w razie awarii staje się wzorcem do automatycznego zduplikowania środowiska. Po zakończeniu tego procesu możemy już kierować ruch aplikacji do nowego systemu.

Szacunek wartości *RPO* jest uwarunkowany planowaną częstotliwością tworzenia kopii zapasowych. Oczywiście strategia *backupowania* jest indywidualną decyzją organizacji i zależy w dużej mierze od poziomu krytyczności danych aplikacji. Wydaje nam się, że właściwym byłoby możliwe najczęstsze transferowanie *snapshotów* czy logów transakcyjnych do odpowiednich *bucketów S3* w awaryjnym regionie, dlatego za szacowaną wartość *RPO* przyjmujemy 2 godziny.

Zakładając możliwość skorzystania z automatycznego powiadamiania użytkowników o awarii regionu przez AWS szacujemy, że czas wykrycia katastrofy przez naszą organizację może wynosić maksymalnie 8 godzin (biorąc pod uwagę, że osoby powiadamiane o awarii, jako pracownicy *non-profit* nie są zobowiązani do ciągłego monitorowania stanu aplikacji). Po takim czasie może zostać uruchomiony plan przywracania infrastruktury, który sam w sobie wydaje się czynnikiem mniej czasochłonnym, a więc szacujemy jego długość na ok. 1 godzinę. Zatem przybliżona wartość *RTO* wynosi 9 godzin.

Oczywiście we wdrażaniu tak rozpisanego rozwiązania trzeba uwzględnić regularne monitorowanie i testowanie, aby upewnić się, że wszystkie procedury działają zgodnie z oczekiwaniami. Odpowiednią formą takich testów mogłyby być np. symulacje awarii o różnym poziomie nasilenia (pojedynczych zasobów, *Availabilty Zone'y*, czy też całego regionu), co umożliwiłoby również weryfikacje oszacowanych wartości *RPO* i *RTO*.

Tak skonstruowany plan przywracania aplikacji do działania w razie katastrofy wydaje się być optymalny dla organizacji wskazanej w zadaniu. Gwarantuje on możliwie najbardziej korzystne warunki finansowe, a równocześnie umożliwia uruchomienie aplikacji w innym regionie w razie

wystąpienia sytuacji kryzysowej.