9. Obsługa zdarzeń przychodzących z usług AWS

W tym rozdziale zajmiemy się wyjaśnieniem, jak wyzwalać funkcje Lambda po zdarzeniach przychodzących z usług AWS, np. wgranie pliku na S3. Poznamy także różnicę w synchronicznej i asynchronicznej obsłudze wywołań Lambdy.

Nowe zasoby wdrożone w poprzednim rozdziale umożliwiają użytkownikom końcowym wgrywać w łatwy i szybki sposób pliki, ale nasza aplikacja nie przetwarza ich w żaden sposób. A my jako twórcy aplikacji mamy kilka opcji do wyboru, jak je przetwarzać.

Pierwsza opcja to stworzenie końcówki w *API*, która wywoływałaby funkcję *Lambda* i synchronicznie przygotowała miniaturkę wysłanego obrazka. W takim wypadku możemy odesłać wynik bezpośrednio lub zapisać go na *S3*, a następnie przekierować użytkowników do nowego miejsca. Zaletą tego podejścia jest jego prostota. Jest jednak olbrzymia wada: to rozwiązanie nie zadziała w przypadku większych plików. Zapewne pamiętasz, że *API Gateway* kończy każde połączenie, które trwa dłużej niż 29 sekund. Górny limit czasu wykonania dla funkcji *Lambda* to 15 minut, ale nie, jeśli zintegrujemy się z usługą *API Gateway*. Jeżeli napotkamy takie ograniczenia, zamiast je obchodzić, musimy wrócić do tablicy i zaprojektować całe rozwiązanie na nowo. Długo żyjące połączenia *HTTP* są podatne na przerwania. Im dłużej żądanie trwa, tym bardziej prawdopodobne jest, że np. *WiFi* w Twojej ulubionej kawiarni rozłączy się i połączy ponownie, urządzenie klienta przerwie wszystkie żądania, aby zredukować zużycie baterii, lub telefon przełączy się z mobilnego połączenia na domowe *WiFi*, zmieniając adres *IP*. Lepiej unikać oczekiwania na długo żyjące połączenia.

Kolejna opcja to komunikacja asynchroniczna. W typowej trójwarstwowej aplikacji najczęściej spotykany mechanizm obsługi zadań, które długo żyją, to stworzenie kilku końcówek *API*. Pierwsza końcówka służy do wystartowania zadania w tle, wysyłając w odpowiedzi identyfikator do klienta. Druga z nich pozwala na sprawdzenie stanu wykonania konkretnego zadania opartego na podanym identyfikatorze. Aplikacja klienta może poradzić sobie w ten sposób z tymczasowymi problemami po swojej stronie bez obawy, że utraci cały proces. Ostatnia końcówka służy do pobrania rezultatu całej operacji po zakończeniu zadania. Bardzo łatwo można taki mechanizm przygotować, opierając się na usługach *API Gateway* oraz funkcjach *Lambda*, ale będzie to dużo bardziej skomplikowane i dużo droższe rozwiązanie niż powinno być. Wykorzystując podejście *serverless*, możemy to zrobić taniej i szybciej.

W poprzednim rozdziale usunęliśmy potrzebę działania jako punkt wejścia do naszej aplikacji i oddaliśmy ją do usług dostępnych na platformie *AWS*. W podobny sposób możemy oddelegować orkiestrację naszych zadań do usług dostępnych na platformie, zamiast implementować je w klasyczny sposób. To znacząco obniża koszt utrzymania, ale też wykonania. Przykładowo, zamiast *API*, które pozwala klientowi na sprawdzenie postępu w wykonaniu zadania, możemy przekazać klientowi wstępnie podpisany adres *URL*, aby sprawdził, czy plik jest dostępny na *S3*. W przypadku podejścia opartego na *API Gateway* płacimy za żądanie przychodzące do *API*, wykonanie funkcji *Lambda* i dostęp do *S3*. W przypadku klienta, który odwołuje się bezpośrednio do *S3*, płacimy tylko za ten konkretny element. W ogólnym rozrachunku to będzie znacznie tańsze, a im mniej komponentów znajduje się między, tym lepszy efekt dla użytkownika końcowego, ponieważ redukujemy opóźnienia. Jak widać, wykorzystanie podejścia typu *serverless* czyni tę operację zarówno tańszą, jak i szybszą.

Analogicznie nie potrzebujemy również osobnej końcówki do rozpoczęcia całego procesu. Wiele z usług na platformie *AWS* może wysłać zdarzenie do usługi *AWS Lambda* na temat tego, co zaszło. W naszym przypadku *S3* bezpośrednio wyśle zdarzenie do funkcji *Lambda* wówczas, gdy plik zostanie wgrany lub usunięty z konkretnego *bucketu*. Nie ma potrzeby, żeby płacić za dodatkowe żądanie do *API* i wykonania funkcji, tylko po to, aby klient cyklicznie pobierał status utworzonego zadania.

Następnym krokiem dla naszej aplikacji będzie utworzenie szkieletu takiej asynchronicznej komunikacji (rysunek 9.1). Stworzymy w tym celu nową funkcję, która będzie wywoływana w odpowiedzi na wgranie pliku, dzięki czemu nie musimy obawiać się o limit czasu wykonania w usłudze *API Gateway*. Na ten moment implementacja tylko skopiuje plik do innego *bucketu*, dzięki czemu będziemy mogli przetestować cały proces. W poprzednim rozdziale wykorzystaliśmy funkcję, która

potwierdzała wgranie pliku (krok trzeci na rysunku 9.1) do wygenerowania wstępnie podpisanego linku dla plików wgrywanych na *S3*. Możemy zmodyfikować tę odpowiedź, tak aby zwracać adres do docelowego *bucketu*. Aplikacje klienckie mogą wykorzystać ten link do pobrania rezultatów. W kolejnym rozdziale zajmiemy się samym procesem generowania miniaturki.

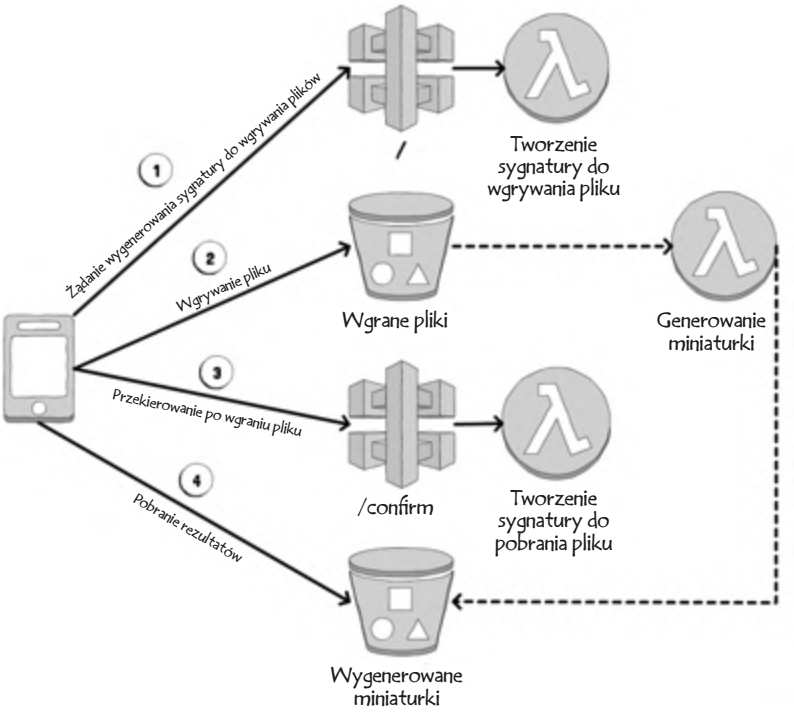
Na początek stworzymy nową funkcję do tworzenia miniaturek. Nie ma potrzeby, żeby mieszać kod przetwarzania formularza z tą konkretną funkcjonalnością, więc stworzymy nowy podkatalog, a w nim nową funkcję. Upewnij się, że twój katalog startowy to ten, który zawiera plik `template.yaml` (jeśli podążasz za przykładem z rozdziału 2, to będzie to katalog o nazwie `app`), i wykonaj następujące dwie komendy:

```
mkdir image-conversion
cd image-conversion
```

Wykorzystamy w przypadku implementacji tej funkcji język *JavaScript* oraz *Node.js*. *AWS SAM* wykorzystuje narzędzie *NPM* do zarządzania zależnościami i budowania paczek, więc musimy zainicjalizować nowy projekt z wykorzystaniem tego narzędzia:

```
npm init --yes
```

Zauważ, że moglibyśmy zaimplementować tę funkcję z wykorzystaniem innego środowiska uruchomieniowego, ponieważ to zupełnie inna funkcja, a narzędzie *SAM* pozwala na mieszanie różnych języków programowania. Jeśli język *JavaScript* nie jest Twoim docelowym wyborem, możesz znaleźć wersję tego samego projektu napisaną w innych językach na stronie https://runningserverless.com.



Rysunek 9.1. *Zamiast serwera aplikacyjnego, który koordynuje akcje, S3 wywoła bezpośrednio funkcję odpowiedzialną za wygenerowanie miniaturki*