Mariusz Dworniczak

Od Dockera do chmury Azure

Aplikacje cloud-native w Azure

Budowa, wdrażanie i bezpieczeństwo

Helion

Redaktor prowadzący: Natalia Hermansa

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Grafika na okładce została wykorzystana za zgodą AdobeStock.com.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/azupra
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:
https://ftp.helion.pl/przyklady/azupra.zip

ISBN: 978-83-289-4025-3

Printed in Poland.

Spis treści

WSTĘP

Dlaczego cloud-native i Microsoft Azure?

Witaj w świecie, w którym pojęcie „serwera pod biurkiem” odeszło do lamusa. Witaj w erze, w której Twoja aplikacja nie jest tylko zbiorem plików, ale żywym organizmem, który skaluje się, zabezpiecza i naprawia niemal automatycznie. Witaj w świecie **cloud-native**.

Jeśli kiedykolwiek męczyłeś się z ręcznym kopiowaniem plików na FTP, jeśli konfiguracja bazy danych na produkcji zajęła Ci całą noc albo jeśli boisz się, że Twój system padnie przy większym ruchu — ta książka jest dla Ciebie.

Co zyskasz dzięki tej lekturze? To nie jest teoretyczna rozprawa o wyższości chmury nad własnym serwerem. To kompletny, praktyczny warsztat. Budujemy wspólnie jedną, zaawansowaną aplikację — od pierwszego wiersza kodu, przez konteneryzację, aż po w pełni zautomatyzowane wdrożenie w Microsoft Azure.

- **Standardy rynkowe.** Nie uczymy się „klikania”. Uczymy się architektury 3-warstwowej, wzorców DTO i Repository — czyli tego, czego wymagają najlepsi pracodawcy.
- **Niezależność dzięki Dockerowi.** Nieważne, co masz zainstalowane na komputerze. Dzięki Dockerowi Twoje środowisko będzie identyczne jak to, które finalnie trafi do chmury. Zero niespodzianek typu „u mnie nie działa”.
- **Bezpłatne konto Azure.** Chmura Azure jest dostępna dla każdego, a ja pokażę Ci, jak zacząć w niej budować zupełnie za darmo.
  - **Jeśli jesteś studentem.** Wykorzystasz *Azure for Students* — bez karty kredytowej otrzymasz 100 USD kredytu i darmowe limity na rok.
  - **Jeśli jesteś profesjonalistą/hobbystą.** Pokażę Ci, jak aktywować *Bezpłatne konto platformy Azure*, które daje 200 USD na start i dostęp do najpopularniejszych usług za zero złotych przez pierwsze 12 miesięcy.
- **Inwestycja w przyszłość.** Umiejętność budowy aplikacji od razu pod chmurę (*cloud-native*) to dziś najkrótsza droga do roli senior developera lub cloud architecta.

Czego nauczysz się w praktyce? Moim celem jest przekazanie Ci „pamięci mięśniowej” nowoczesnego inżyniera. Nie będziemy tylko pisać kodu — będziemy budować kompletny ekosystem.

Dzięki tej lekturze:

- **Opanujesz cykl życia aplikacji.** Od lokalnego `docker-compose up`, do publicznego adresu URL w domenie azurewebsites.net.
- **Zrozumiesz separację warstw.** Nauczysz się, dlaczego frontend nie powinien wiedzieć o bazie danych i jak bezpiecznie połączyć je przez API.

- **Zabezpieczysz swoje dane.** Dowiesz się, jak używać Azure Key Vault, by Twoje hasła nigdy nie trafiły do repozytorium kodu.
- **Wdrożysz CI/CD.** Skonfigurujesz GitHub Actions tak, aby każdy Twój commit automatycznie testował i odświeżał aplikację na produkcji.

Dla kogo jest ta książka? Niezależnie od tego, czy jesteś studentem stawiającym pierwsze kroki w IT, programistą chcącym przeskoczyć na poziom *cloud-native*, czy administratorem, który chce zrozumieć świat deweloperów — ten materiał jest dla Ciebie.

Jak korzystać z tej lektury? To nie jest powieść do poduszki. To instrukcja obsługi Twojej nowej kariery. Każdy rozdział to kolejny artefakt — kolejna cegiełka Twojego projektu. Nie bój się psuć, nie bój się błędów w terminalu. Naprawianie ich to najskuteczniejsza lekcja, jaką możesz odebrać.

Twój kod zasługuje na coś więcej niż Twój dysk twardy. Zasługuje na chmurę. **Zaczynamy wdrożenie Twojego sukcesu!**

Kody źródłowe wybranych przykładów dostępne są pod adresem:

https://ftp.helion.pl/przyklady/azupra.zip

ROZDZIAŁ 1.

Architektura cloud-native — projektowanie dla nowoczesnej chmury

Zanim napiszemy pierwszą linię kodu i zanim uruchomimy jakikolwiek serwer, musimy zdefiniować fundamenty. W nowoczesnym IT nie budujemy już aplikacji „na serwer" — budujemy je **dla chmury**.

Cloud-native to podejście do projektowania, budowania i uruchamiania aplikacji, które w pełni wykorzystują zalety modelu obliczeń w chmurze. Nie chodzi tylko o to, *gdzie* aplikacja działa, ale *jak* została stworzona. Systemy *cloud-native* charakteryzują się:

- **Modułowością** — rozbiciem na niezależne komponenty.
- **Odpornością** — zdolnością do przetrwania awarii pojedynczych usług.
- **Skalowalnością** — łatwym dopasowaniem do rosnącej liczby użytkowników.
- **Przenośnością** — niezależnością od konkretnego sprzętu czy dostawcy.

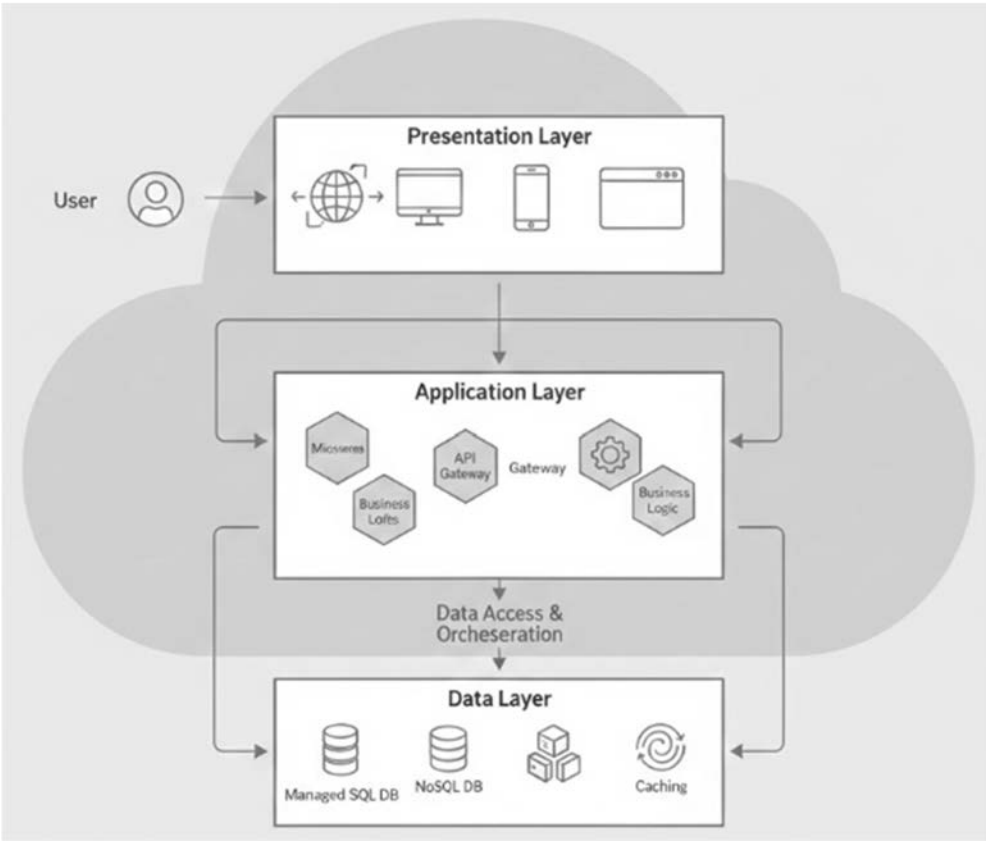
W tym rozdziale przejdziemy ścieżkę od teorii do praktycznego dowodu na to, że dobrze zaprojektowany system jest wolny od ograniczeń Twojego komputera.

1.1. Filozofia 3-warstwowa w praktyce cloud-native

W świecie cloud-native najskuteczniejszym sposobem na zachowanie porządku i skalowalności jest rygorystyczny podział na trzy warstwy (przykładową implementację przedstawia rysunek 1.1). Każda z nich pełni inną funkcję i może być rozwijana przez różne zespoły lub w różnych technologiach.

- Warstwa prezentacji (*Presentation Layer*):
  - **Technologia** — np. React.
  - **Rola** — to „twarz" aplikacji. Odpowiada wyłącznie za interfejs użytkownika i komunikację z API. W podejściu chmurowym traktujemy ją jako statyczny zasób, który można serwować globalnie z punktów brzegowych (*Edge*).
- Warstwa aplikacji (*Application Layer*):
  - **Technologia** — np. .NET Web API lub Node.js.
  - **Rola** — logika biznesowa, procesowanie danych, autoryzacja. To serce systemu, które w chmurze będzie skalować się w zależności od obciążenia.
- Warstwa danych (*Data Layer*):
  - **Technologia** — np. Azure SQL/Cosmos DB.

- **Rola** — trwałe składowanie informacji. W architekturze *cloud-native* dbamy o to, by baza była oddzielona od logiki, co pozwala na jej niezależne backupowanie i replikację.



RYSUNEK 1.1. Schemat trzywarstwowej aplikacji

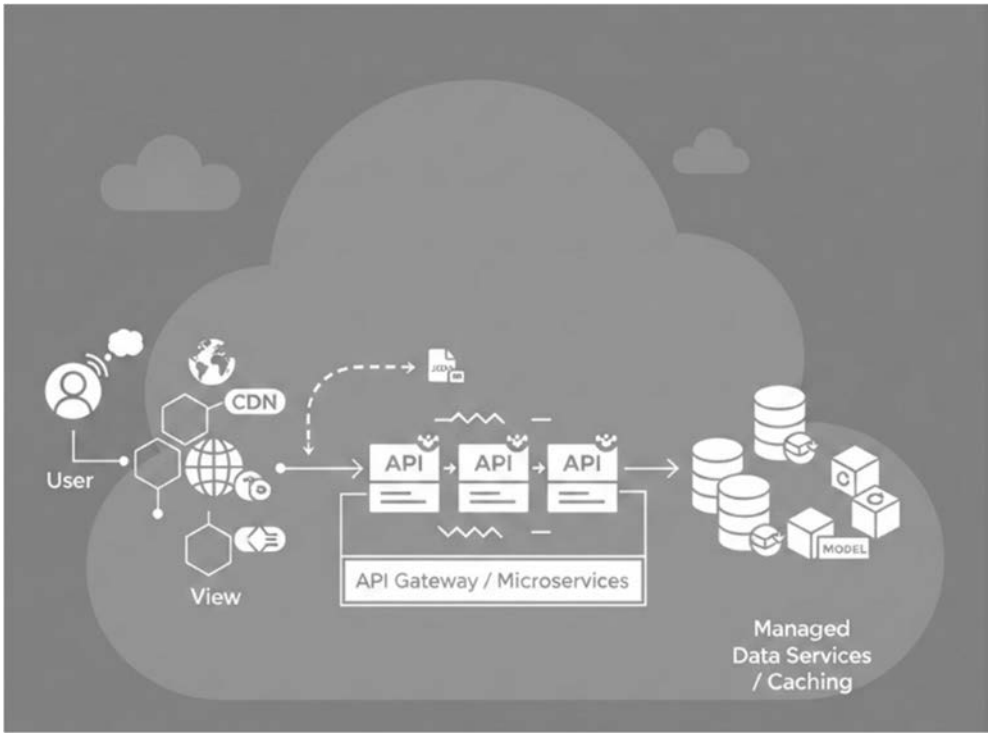
1.2. Kluczowe wzorce projektowe — fundament czystej architektury

W architekturze *cloud-native* kod musi nie tylko działać, ale przede wszystkim być łatwy do testowania, rozbudowy i wymiany. Stosujemy trzy fundamentalne wzorce, które dbają o to, by nasza aplikacja nie stała się „nienaruszalnym monolitem".

1.2.1. MVC (Model-View-Controller)

Jest to wzorzec strukturalny, który dzieli aplikację na trzy główne komponenty. W naszym projekcie *cloud-native* wzorzec ten ewoluuje w stronę **Web API** ze względu na podział na osobny frontend i backend.

- **Co to jest?** Mechanizm oddzielający dane (*Model*) i logikę sterowania (*Controller*) od sposobu ich prezentacji (*View*).
- **Po co to stosujemy?** Aby zmiany w wyglądzie aplikacji (np. zmiana przycisku w aplikacji korzystającej z biblioteki React) nie wymagały modyfikacji logiki na serwerze.
- **Przykład.** Użytkownik klika *Zapisz zadanie*. **Controller** odbiera to żądanie, instruuje **Model**, aby zaktualizował dane, a następnie zwraca informację o sukcesie, którą warstwa prezentacji wyświetla użytkownikowi. Przykładowy diagram wzorca MVC pokazuje rysunek 1.2.



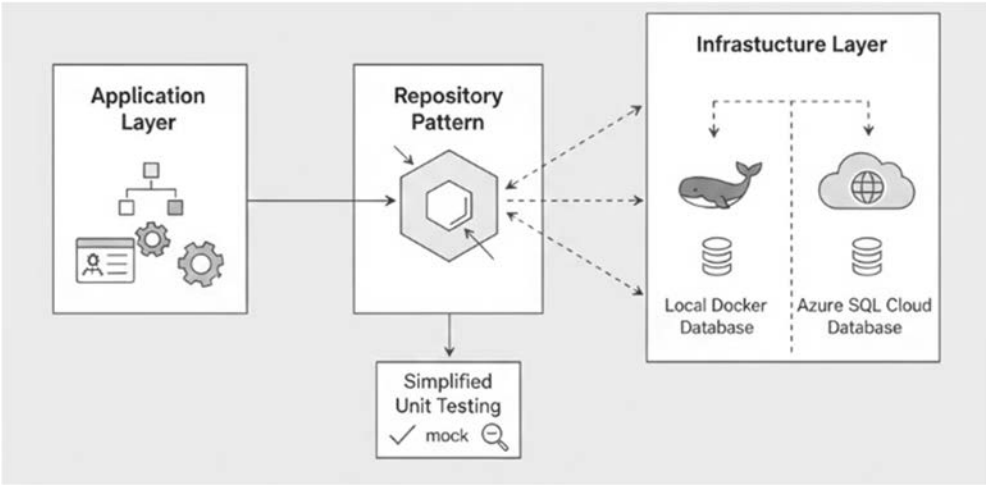
RYSUNEK 1.2. Diagram wzorca MVC

1.2.2. Repository Pattern (wzorzec repozytorium)

To warstwa pośrednicząca między logiką biznesową aplikacji a źródłem danych.

- **Co to jest?** Abstrakcja, która udaje kolekcję obiektów w pamięci, ukrywając przed resztą systemu fakt, że pod spodem znajduje się baza danych SQL, plik JSON czy zewnętrzne API.
- **Po co to stosujemy?** Aby zapewnić **niezależność od dostawcy bazy danych**. Dzięki temu Twoja logika biznesowa (*Application Layer*) nie może stwierdzić, czy rozmawiasz z lokalnym kontenerem Docker, czy z usługą Azure SQL w chmurze. Znacznie ułatwia to pisanie testów jednostkowych (mockowanie bazy).

- **Przykład.** Zamiast pisać zapytanie SQL bezpośrednio w kontrolerze, wywołujesz metodę `_taskRepository.GetAllActive()`. Jeśli w przyszłości zmienisz bazę z SQL na NoSQL (Cosmos DB), zmienisz tylko kod wewnątrz repozytorium — reszta aplikacji pozostanie nietknięta. Na rysunku 1.3 przedstawiono diagram wzorca repozytorium.



RYSUNEK 1.3. Diagram wzorca repozytorium

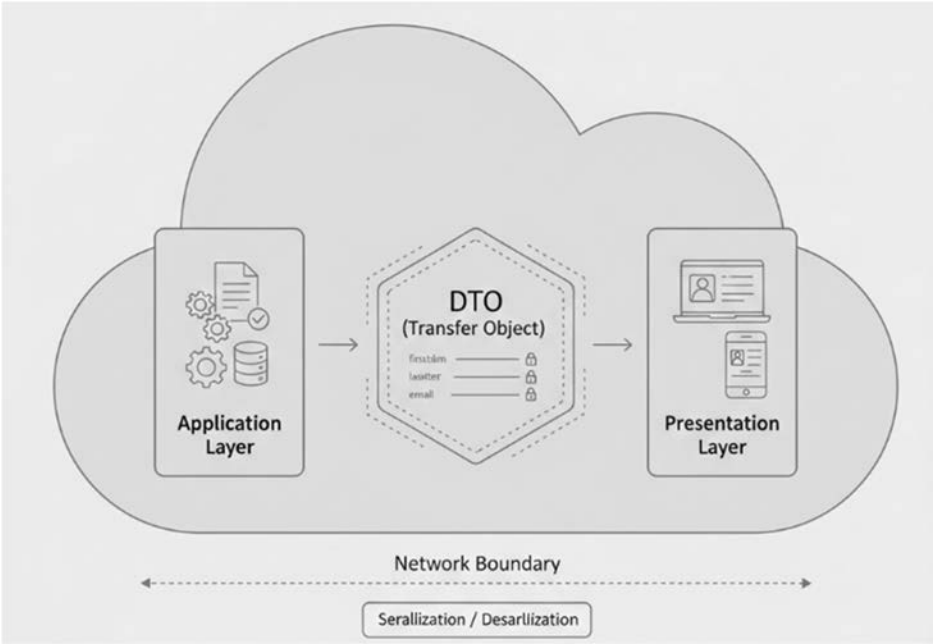
1.2.3. DTO (Data Transfer Object)

Obiekty DTO to proste kontenery służące wyłącznie do przesyłania danych między procesami (np. backend ↔ frontend). Diagram DTO (Data Transfer Object) przedstawiono na rysunku 1.4.

- **Co to jest?** To „paszporty dla danych". Są to klasy, które nie zawierają żadnej logiki biznesowej, a jedynie zestaw pól.
- Po co to stosujemy?
  - **Bezpieczeństwo.** Nie chcemy wysyłać do przeglądarki całego obiektu bazy danych (który może zawierać np. hash hasła lub ukryte ID). Wysyłamy tylko to, co UI musi wyświetlić.
  - **Wydajność.** Ograniczamy rozmiar przesyłanych paczek danych (tzw. *payload*).
  - **Odporność na zmiany.** Możesz zmienić strukturę tabeli w bazie danych, nie psując kontraktu z frontendem, o ile zmapujesz nowe dane do starego formatu DTO.
- **Przykład.** Twoja encja w bazie `User` ma 20 pól, w tym datę utworzenia i tajne tokeny. Twoje `UserDTO` wysyłane do biblioteki React zawiera tylko `FirstName`, `LastName` i `AvatarUrl`.

Uzasadnienie techniczne zastosowania wzorców:

Zastosowanie tych wzorców sprawia, że Twoja aplikacja staje się **luźno powiązana (*loosely coupled*)**. To kluczowa cecha cloud-native: każdy element systemu może ewoluować niezależnie, co jest niezbędne w dynamicznym środowisku chmurowym Azure.



RYSUNEK 1.4. Diagram DTO (Data Transfer Object)

1.3. Projekt — Cloud Task Manager

W tej książce nie ograniczamy się do prostych przykładów typu „Hello, World". Budujemy rozbudowany system, który posłuży nam jako kompletna ilustracja procesu tworzenia natywnej aplikacji chmurowej. Przechodząc przez kolejne rozdziały, nauczysz się krok po kroku, jak przekuć wymagania biznesowe w działający, zabezpieczony i skalowalny produkt w Azure.

Cel projektu

Stworzenie platformy do zarządzania zadaniami, która demonstruje pełny cykl życia aplikacji: od lokalnego kontenera, przez automatyzację testów, aż po wdrożenie w chmurze Azure.

1.3.1. Zakres funkcjonalny (co budujemy?)

Nasza aplikacja, niezależnie od Twojej finalnej domeny (może to być system rezerwacji lub LMS), musi implementować następujące mechanizmy:

- **Zarządzanie tożsamością** — implementacja rejestracji i logowania użytkowników. Wykorzystamy lokalną bazę dla fazy dev, by docelowo zintegrować się z **Azure Entra ID** (standard korporacyjny).
- **Kompletny cykl CRUD** — pełna obsługa encji biznesowych (tworzenie, odczyt, edycja, usuwanie) z zachowaniem integralności danych.

- **Profesjonalny interfejs (UI/UX)** — wykorzystanie biblioteki React do stworzenia widoków list, filtrów oraz szczegółowych formularzy edycji.

W naszych przykładowych rozwiązaniach zrobimy Cloud Tasks Managera, ale Ty możesz zmienić nazwy pól i wykorzystać to jako szablon do Twoich przyszłych projektów programistycznych.

1.3.2. Wymagania techniczne (jak budujemy?)

Aby projekt można było uznać za zgodny z duchem cloud-native, musi on spełniać rygorystyczne warunki techniczne:

- **Wersjonowanie i historia** — repozytorium GitHub z jasną historią commitów (każdy etap to osobny artefakt).
- **Struktura monorepo** — czysty podział na foldery */frontend* i */backend*, co odzwierciedla architekturę 3-warstwową.
- **Orkiestracja lokalna (*Docker Compose*)** — cały system (React, API, SQL) musi „wstawać" po wpisaniu jednej komendy w terminalu. To gwarancja, że Twoje środowisko jest niezależne od systemu operacyjnego.
- **Konfiguracja 12-Factor App** — wykorzystanie zmiennych środowiskowych (*.env*) do konfiguracji połączeń, co umożliwi płynne przejście między bazą lokalną a chmurową bez zmiany kodu.

1.3.3. Wymagania chmurowe (gdzie wdrażamy?)

Finałem Twojej pracy będzie publicznie dostępny adres URL, pod którym aplikacja będzie działać w środowisku Microsoft Azure, podzielonym na:

- **Frontend** — hostowany w usłudze Azure Static Web Apps.
- **Backend** — uruchomiony w usłudze Azure App Service.
- **Database** — zarządzana baza Azure SQL w regionie Poland Central.

  (Uwaga: Jeśli Twoja subskrypcja blokuje ten region, wybierz np. Germany West Central).

Dlaczego nasze podejście jest lepsze?

Większość kursów uczy technologii w izolacji. My uczymy **procesu**. Widząc, jak wzorce MVC, Repository i DTO współpracują ze sobą w Dockerze, a potem idealnie mapują się na usługi Azure, zyskasz kompetencje, których poszukują architekci rozwiązań chmurowych. Nie uczysz się teorii o chmurze — uczysz się, jak do niej wejść z gotowym produktem.

ROZDZIAŁ 7.

Wdrożenie aplikacji w Azure i zarządzanie dostępem

Lokalny Docker służył nam do symulacji środowiska. W chmurze przechodzimy na model **PaaS (*Platform as a Service*)**. Oznacza to, że nie zarządzamy już systemem operacyjnym ani aktualizacjami silnika SQL — zajmuje się tym Microsoft Azure. Proces wdrażania (ang. *deployment*) to końcowy etap publikacji, polegający na przeniesieniu skompilowanego kodu źródłowego z lokalnego środowiska programistycznego do infrastruktury chmurowej. My skupiamy się wyłącznie na kodzie i bezpiecznym połączeniu usług.

Co zrobimy w tym rozdziale?

1. Przeniesiemy bazę danych do **Azure SQL** i zmigrujemy jej schemat.
2. Uruchomimy logikę biznesową w **Azure App Service**.
3. Opublikujemy interfejs w **Azure Static Web Apps**.
4. Zabezpieczymy całość za pomocą **firewalla** i ról **RBAC**.

7.1. Lab 06 — wdrożenie aplikacji w Azure (artefakt nr 6)

Checklista do wykonania:

☐ 6.1. Azure SQL Database utworzona i dostępna przez firewall.

☐ 6.2. Schemat bazy danych zmigrowany przy użyciu `dotnet ef`.

☐ 6.3. Backend (.NET) działający pod publicznym adresem HTTPS.

☐ 6.4. Frontend (React) zintegrowany z produkcyjnym adresem API.

- Działająca aplikacja z Azure.

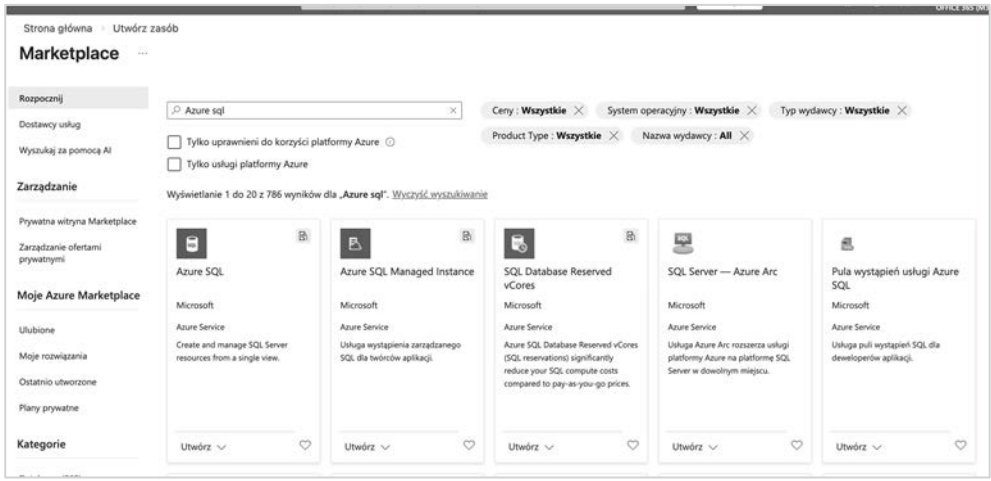
☐ 6.5. Zaktualizowana dokumentacja i kod wysyłany do GitHuba.

7.2. Przykładowe rozwiązanie lab 06 (artefakt nr 6)

Krok 1. Azure SQL — Twoja baza w chmurze

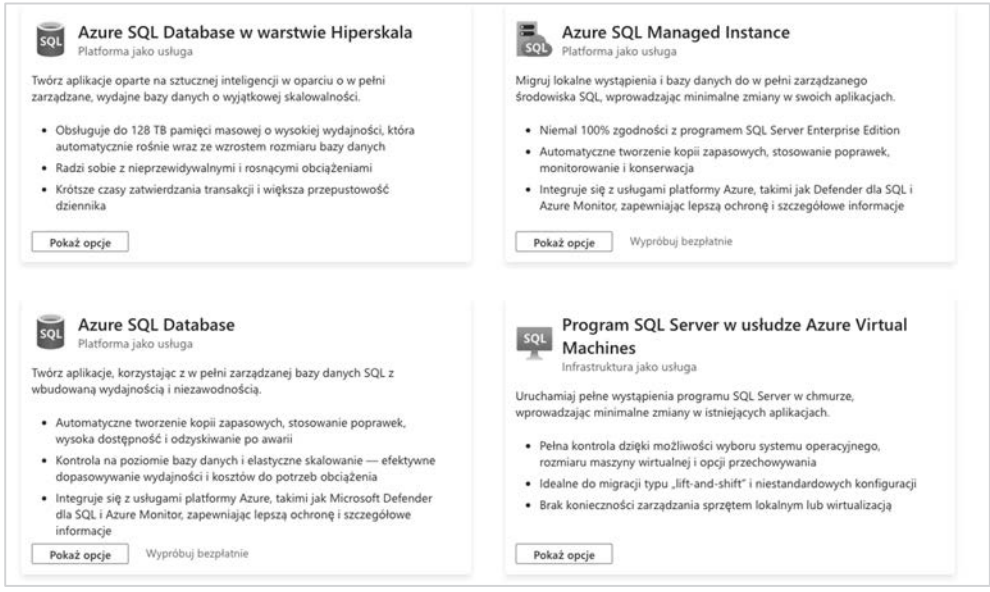
Zaczynamy od bazy, ponieważ backend już w momencie startu musi wiedzieć, gdzie się połączyć.

1. W portalu Azure wybierz *Utwórz zasób/Azure SQL* (rysunek 7.1).



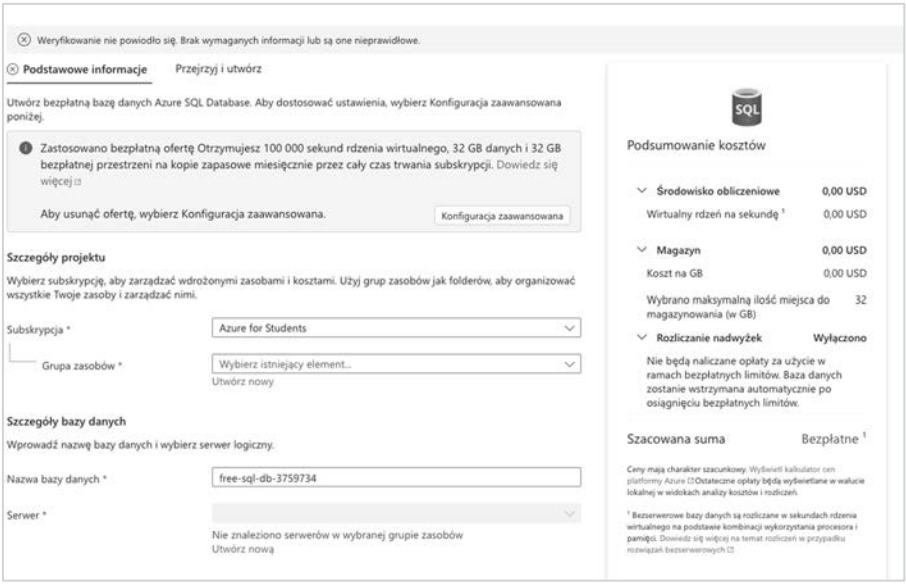
RYSUNEK 7.1. Wybór bazy Azure SQL

2. Wybierz typ bazy. Azure zapyta, który typ bazy chcesz wybrać. Wybierz *Azure SQL Database. Platforma jako usługa*. Kliknij *Wypróbuj bezpłatnie* (rysunek 7.2).



RYSUNEK 7.2. Wybór typu bazy

3. Skonfiguruj bazę. Po wyborze typu bazy dostaniesz informację podobną do tej: *Zastosowano bezpłatną ofertę Otrzymujesz 100 000 sekund rdzenia wirtualnego, 32 GB danych i 32 GB bezpłatnej przestrzeni na kopie zapasowe miesięcznie przez cały czas trwania subskrypcji* (rysunek 7.3).



RYSUNEK 7.3. Konfiguracja bazy Azure SQL

Konfiguracja bazy:

- **Grupa zasobów.** Utwórz nową grupę, np. **`rg-cloud-task-manager`**. Później będziesz używał tej samej grupy dla reszty projektu.

Dlaczego to ważne? Grupa zasobów pozwala jednym kliknięciem usunąć cały projekt (bazę, serwer, API), gdy skończysz naukę, co chroni Cię przed niechcianymi kosztami w przyszłości.

- Polecenie *Serwer/Utwórz nowy* — przeniesie Cię do nowego okna (rysunek 7.4).

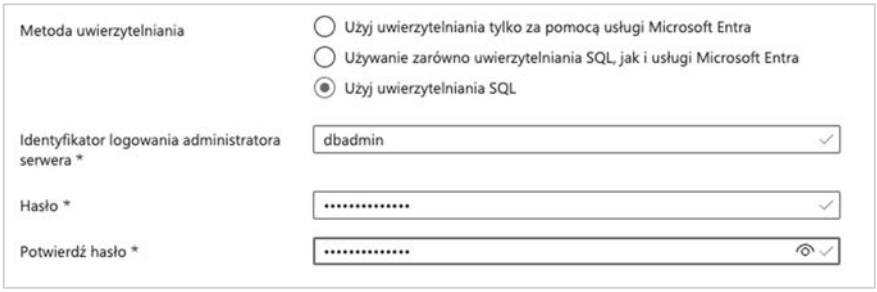
RYSUNEK 7.4. Konfigurowanie nazwy serwera SQL

***Uwaga**: w lutym 2026 region Poland Central nie był dozwolony w ramach subskrypcji studenckiej **Azure for Students** w Polsce. Dozwolone regiony: "germanywestcentral", "francecentral", "swedencentral", "norwayeast", "spaincentral". W takim wypadku wybierz: Germany West Central.*

- **Nazwa serwera:** `cloud-app-db`.

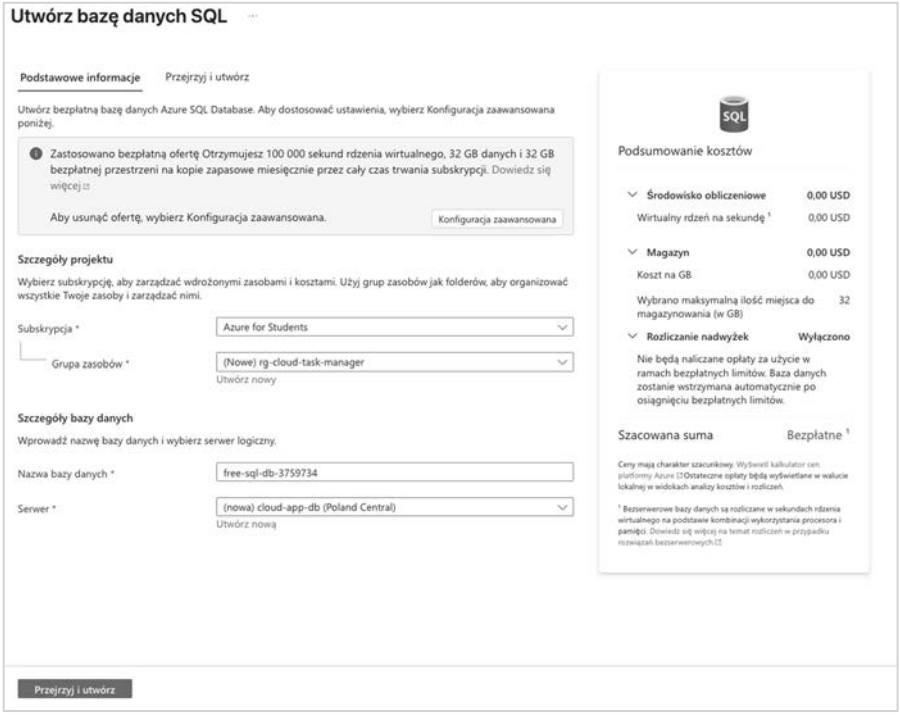
Uwaga: nazwa powinna być unikalna w skali całego Azure, więc jeśli wybrana przez Ciebie nazwa okaże się zajęta, dodaj coś unikalnego, np. datę utworzenia tej bazy).

- **Uwierzytelnienie**. Użyj *SQL Authentication* (np. *login*: `dbadmin`, *hasło*: `Silne ↪Haslo123!`). Bezpieczeństwem haseł zajmiemy się w dalszych rozdziałach. Na razie zapisz gdzieś te dane, nazwę serwera, identyfikator administratora i hasło (rysunek 7.5).



RYSUNEK 7.5. Uwierzytelnianie bazy

4. Aby zakończyć, kliknij *Przejrzyj i utwórz* (rysunek 7.6).



RYSUNEK 7.6. Finalizacja tworzenia bazy danych

Potwierdzamy przyciskiem *Utwórz*.

Poczekaj, aż baza zostanie utworzona. Dostaniesz komunikat:

```
Wdrożenie zostało ukończone
```

W szczegółach wdrożenia będą widoczne trzy zaznaczenia zasobów (rysunek 7.7).

RYSUNEK 7.7. Szczegóły wdrożenia bazy Azure SQL

Krok 2. Połączenie aplikacji z Azure SQL

Baza danych już działa, ale jest niczym sejf w innym mieście — musisz mieć klucz i adres, żeby się do niego dostać.

Krok 2.1. Pobranie Connection String (parametry połączenia)

Baza danych już działa, ale musimy pobrać „adres i klucz", aby backend mógł się z nią połączyć.

1. Znajdź bazę (nie serwer!).
   - Na stronie głównej portalu Azure lub wewnątrz swojej grupy zasobów znajdź zasób, który ma ikonę niebieskiego walca i typ *Baza danych SQL* (*SQL database*).
   - Najprawdopodobniej nazywa się ona *free-sql-db-xxxxxxx* (lub podobnie). Kliknij ją (rysunek 7.8).

RYSUNEK 7.8. Wybór bazy danych

2. Otwórz parametry połączenia:
   - Będąc już w widoku bazy, spójrz na menu po lewej stronie.
   - W sekcji *Ustawienia* (*Settings*) kliknij *Parametry połączenia* (*Connection strings*).
3. Skopiuj ADO.NET:
   - Upewnij się, że wybrana jest pierwsza zakładka — *ADO.NET*.

Zbuduj swoją pierwszą aplikację specjalnie dla chmury

Aplikacje typu *cloud-native*, na których budowie i implementacji skupia się ta książka, są tworzone specjalnie dla środowisk chmurowych. Tego typu rozwiązanie, korzystające z elastyczności chmury, opiera się na architekturze mikroserwisów, konteneryzacji, interfejsach API i automatyzacji. Dzięki temu jest szybkie we wdrażaniu, odporne na awarie i łatwe do rozbudowy.

Ten poradnik pomoże Ci stworzyć od zera zaawansowaną aplikację: od pierwszego wiersza kodu, przez konteneryzację (Docker), aż po pełną automatyzację (CI/CD) w chmurze Microsoft Azure. W trakcie pracy nad projektem dowiesz się, jak pozyskać odpowiedni budżet i darmowe limity od Microsoftu, poznasz standardy rynkowe (architektura 3-warstwowa, wzorce DTO i Repository), zaznajomisz się również z zagadnieniami dotyczącymi bezpieczeństwa, takimi jak korzystanie z Azure Key Vault, by hasła nigdy nie trafiały do repozytoriów kodu.

Helion

helion.pl

HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-289-4025-3

9 788328 940253

Cena: 67,00 zł

Od Dockera do chmury Azure

Mariusz Dworniczak

Doświadczony menedżer techniczny i certyfikowany architekt chmurowy (AWS, Azure, Google Cloud). Od ponad dwudziestu lat zarządza złożonymi programami IT i migracjami infrastruktury w globalnych organizacjach, takich jak IBM, BOX czy Toyota. Wykładowca na Uniwersytecie WSB Merito oraz doświadczony mentor techniczny. Specjalizuje się w automatyzacji infrastruktury (IaC), a także łączeniu twardych kompetencji technicznych z nowoczesnymi metodykami zarządzania projektami. Prywatnie pasjonat technologii blockchain, AI i gry w szachy.