ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI INFORMATICA - SCIENZA e INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Analisi, Progettazione e Distribuzione in Cloud di applicativo multipiattaforma per l'organizzazione di eventi condivisi e la condivisione multimediale automatica in tempo reale

Relatore:
Chiar.mo Prof.
Michele Colajanni

Presentata da:
Giacomo Romanini

Sessione Marzo 2025

Anno Accademico 2024/2025

Abstract

Lo sviluppo di un applicativo multipiattaforma diretto all'organizzazione di eventi condivisi, caratterizzato in particolare dalla condivisione multimediale in tempo reale, richiede opportune capacità di scalabilità, atte a garantire una risposta efficace anche con alti volumi di richieste, offrendo prestazioni ottimali. Le tecnologie cloud, con la loro disponibilità pressoché illimitata di risorse e alla completa e continua garanzia di manutenzione, offrono l'architettura ideale per il supporto di simili progetti.

Tuttavia, l'integrazione tra la logica applicativa ed i molteplici servizi cloud, insieme alla gestione delle loro interazioni reciproche, comporta sfide specifiche, in particolare legate all'ottimizzazione di tutte le risorse.
L'individuazione e la selezione delle soluzioni tecnologiche più adatte per ogni obiettivo, e l'adozione delle migliori pratiche progettuali devono procedere parallelamente con lo sviluppo del codice, al fine di sfruttare efficacemente le potenzialità offerte.

In tale prospettiva, questa tesi illustra le scelte progettuali e implementative adottate nello sviluppo dell'applicativo in questione, evidenziando l'impatto dell'integrazione delle risorse cloud sul risultato finale.

Indice

Introduzione

In un contesto sempre più connesso, la crescente quantità di contatti, la rapidità delle comunicazioni e l'accesso universale alle informazioni rendono la creazione, l'organizzazione e la partecipazione ad eventi estremamente facile, ma al contempo generano un ambiente frenetico e spesso dispersivo.

Risulta infatti difficile seguire tutte le opportunità a cui si potrebbe partecipare, considerando le numerose occasioni che si presentano quotidianamente, rischiando di ricordarsene solo successivamente, una volta passate. Basti pensare, ad esempio, alle riunioni di lavoro, alle serate con amici, agli appuntamenti informali per un caffè, ma anche a eventi più strutturati come fiere, convention aziendali, concerti, partite sportive o mostre di artisti che visitano occasionalmente la città.

Questi eventi possono sovrapporsi, causando dimenticanze o conflitti di pianificazione, con il rischio di delusione o frustrazione. Quando si è invitati a un evento, può capitare di essere già impegnati, o di trovarsi in attesa di una conferma da parte di altri contatti. In questi casi, la gestione degli impegni diventa complessa: spesso si conferma la partecipazione senza considerare possibili sovrapposizioni, o dimenticandosi, per poi dover scegliere e disdire all'ultimo momento.

D'altra parte, anche quando si desidera proporre un evento, la ricerca di un'attività interessante può diventare un compito arduo, con la necessità di consultare numerosi profili social di locali e attività, senza avere inoltre la certezza che gli altri siano disponibili.

Tali problemi si acuiscono ulteriormente quando si tratta di organizzare eventi di gruppo, dove bisogna allineare gli impegni di più persone.

In questo contesto, emergono la necessità e l'opportunità di sviluppare uno strumento che semplifichi la proposta e la gestione degli eventi, separando il momento della proposta da quello della conferma di partecipazione. In tal modo, gli utenti possono valutare la disponibilità degli altri prima di impegnarsi definitivamente, facilitando l'invito e la partecipazione.

In risposta a tali richieste è nata Wyd, un'applicazione che permette ai clienti di organizzare i propri impegni, siano essi confermati oppure proposti. Essa permette anche di rendere più intuitiva la ricerca di eventi attraverso la creazione di uno spazio virtuale centralizzato dove gli utenti possano pubblicare e consultare tutti gli eventi disponibili, diminuendo l'eventualità di perderne qualcuno. La funzionalità chiave di questo progetto si fonda sull'idea di affiancare alla tradizionale agenda degli impegni certi un calendario separato, che mostri tutti gli eventi a cui si potrebbe partecipare. Una volta confermata la partecipazione a un evento, questo verrà spostato automaticamente nell'agenda personale dell'utente. Gli eventi creati potranno essere condivisi con persone o gruppi, permettendo di visualizzare le conferme di partecipazione. Inoltre, considerando l'importanza della condivisione di contenuti multimediali, questo progetto prevede la possibilità di condividere foto e video con tutti i partecipanti all'evento, attraverso la generazione di link per applicazioni esterne o grazie all'ausilio di gruppi di profili. Al termine dell'evento, l'applicazione carica automaticamente le foto scattate durante l'evento, per allegarle a seguito della conferma dell'utente.

Figura 1: Il logo di Wyd

Come anticipato, la realizzazione di un progetto come Wyd implica la risoluzione e la gestione di diverse problematiche tecniche. In primo luogo, garantire la persistenza dell'agenda dell'utente, che deve essere aggiornata e mantenuta in modo affidabile e coerente, tale da permettere per un uso distribuito del servizio. Inoltre, la funzionalità di condivisione degli eventi richiede l'aggiornamento in tempo reale di tutti gli utenti coinvolti, al fine di rimanere sempre aggiornati. Infine, il caricamento e il salvataggio delle foto introduce la necessità di gestire richieste di archiviazione di dimensioni significative.

In questa prospettiva, risulta fondamentale esplorare il procedimento di analisi e progettazione che ha portato allo sviluppo di un applicativo multipiattaforma efficace, affidabile e scalabile, in grado di soddisfare le esigenze degli utenti nell'ambito dell'organizzazione di eventi condivisi e della condivisione multimediale automatica in tempo reale. Seguendo l'esame dei requisiti necessari, l'analisi dello sviluppo del progetto affronterà in una prima parte le scelte infrastrutturali che hanno portato a definire la struttura centrale dell'applicazione, spiegando brevemente i requisiti e le soluzioni trovate. Successivamente si osserverà lo studio effettuato per gestire la memoria, in quanto fattore che più incide sulle prestazioni. Particolare attenzione è stata dedicata, infatti, a determinare le tecnologie e i metodi che meglio corrispondono alle esigenze derivate dal salvataggio e dall'interazione logica degli elementi. Infine, per permettere la gestione delle immagini, che introducono problematiche impattanti sia sulle dimensioni delle richieste sia sull'integrazione con la persistenza, una terza parte si concentrerà sulle scelte implementative adottate per l'inserimento delle funzionalità.

0.1 L'architettura del server principale

Per poter essere in grado di gestire un numero elevato di richieste in tempi ridotti, l'applicazione deve garantire un'elevata scalabilità.

Capace di soddisfare al meglio questa esigenza è Azure Functions, un servizio serverless che consente di suddividere il codice in unità indipendenti, ciascuna eseguita in un ambiente di esecuzione unico per ogni richiesta. Questa caratteristica, combinata con la virtualizzazione dell'ambiente di esecuzione offerta dal cloud, consente una scalabilità potenzialmente illimitata.

L'indipendenza e la natura stateless di ogni funzione, ovvero la sua esecuzione svincolata dalle informazioni sullo stato o sulla sessione, sono responsabilità dello sviluppatore. Ogni funzione segue il principio di singola responsabilità, eseguendo un unico compito specifico. Tuttavia, nel caso di richieste che necessitino l'esecuzione coordinata di più funzioni, Azure Durable Functions fornisce una soluzione efficace.

Integrata all'interno delle Azure Functions, Azure Durable Function consente la creazione di una funzione orchestrator, incaricata di gestire l'ordine, lo stato, il ciclo di vita e le risposte delle varie funzioni coinvolte nell'elaborazione della richiesta.

Mantenendo un'architettura indipendente e scalabile, questa soluzione consente di gestire efficacemente scenari in cui un'esecuzione sequenziale delle operazioni è cruciale, dove è necessario effettuare tentativi aggiuntivi in caso di errore o fallimento o si richiede l'attesa del completamento di operazioni con un tempo di esecuzione prolungato.

Tuttavia, l'architettura stateless e l'accoppiamento debole tra orchestrator e funzioni in esecuzione causano un tempo di risposta delle Azure Durable Functions più elevato. Per ottimizzare l'allocazione delle risorse, il sistema avvia perciò solo le funzioni strettamente necessarie all'esecuzione del compito determinato, stanziando al minimo il consumo computazionale.

0.1.1 La scelta del linguaggio

Come linguaggio di programmazione per lo sviluppo delle funzioni è stato utilizzato C#. La consapevolezza che sia l'ambiente di sviluppo di C#, ovvero il framework .Net, sia la piattaforma Azure siano entrambi sviluppati e mantenuti dalla stessa azienda, Microsoft, garantisce elevati livelli di stabilità, supporto e coordinamento delle tecnologie adottate.

Inoltre, l'integrazione con Entity Framework Core permette la mappatura direttamente in oggetti dei componenti del dominio, semplificando così la logica delle relazioni ed astraendo le comunicazioni con il database. Grazie all'utilizzo delle proprietà virtuali degli oggetti si può applicare il lazy loading, riducendo il numero di richieste al database solo a quando esse sono strettamente necessarie, mantenendo in codice a livello logico ed ottimizzandone le prestazioni.

Azure Functions in ambiente .Net supporta due modelli di esecuzione e sviluppo: in-process worker o isolated worker. Il worker è il processo all'interno dell'applicativo che gestisce la creazione delle risorse e l'esecuzione delle funzioni in risposta alle richieste. Nella modalità in-process, la funzione viene eseguita all'interno dello stesso processo del worker che l'ha generata, riducendo la quantità di allocazione delle risorse necessarie ma condividendo l'ambiente di esecuzione. Nel modello isolated, invece, ogni funzione viene eseguita attraverso un processo indipendente dedicato, garantendo maggiore isolamento e quindi riducendo le possibili dipendenze tra le funzioni.

Inoltre, il modello isolated worker offre ulteriori vantaggio grazie al maggiore supporto fornito: innanzitutto esso prevede una maggiore compatibilità, grazie al più ampio numero di versioni del framework .Net a disposizione, a differenza del modello in-process, limitato alle sole versioni con supporto a lungo termine. In secondo luogo, il supporto per la creazione di middleware personalizzati permette l'elaborazione di un codice intermedio tra la chiamata e l'esecuzione della funzione, funzionalità invece non disponibile nel modello in process. Considerati questi vantaggi, le funzioni sono state sviluppate utilizzando il modello isolated worker per garantire maggiore flessibilità, compatibilità e modularità dell'architettura.

0.1.2 La logica applicativa

Per quanto ogni funzione ricopra un unico compito, alcuni parti possono dover essere condivise. Per questo motivo, la logica applicativa è stata suddivisa creando un metono specifico per ogni singola esigenza, che le funzioni chiameranno ogni volta che ne hanno bisogno. I metodi vengono quindi raggruppati in classi in base all'inerenza dei loro scopi, concentrando il codice che condivide le stesse necessità e uniformando il suo stile. Le dipendenze vengono quindi inizializzate un'unica volta a livello di classe, creando un software più ordinato.

Nella stessa modalità di suddivisione delle responsabilità del client, le classi sono state sviluppate tenendo conto delle divisioni del dominio e di ulteriori responsabilità specifiche. Per ogni elemento principale del dominio è stata sviluppata una classe service che implementa le operazioni relative, mentre, per compiti che richiedono particolare attenzione o che astraggono l'interazione con una particolare risorsa, vengono implementate classi apposite.

Ogni servizio relativo agli elementi strutturali ha bisogno di una connessione con il database per poter applicare le modifiche eseguite. Questa viene implementata da una classe chiamata WydDbContext che racchiude la logica e le impostazioni legate alla persistenza principale. Si concentrano così in un'unico luogo tutte le necessità e le configurazioni di basso livello relative alla sua interazione, quali la definizione del dominio e delle sue relazioni ed eventuali operazioni da applicare in automatico qualora la natura dell'oggetto lo richieda. Ad esempio, alcune classi richiedono l'aggiornamento automatico della data di ultima modifica. L'implementazioni delle classi del dominio viene trattata nei capitoli seguenti.

Per alcuni compiti specifici sono state implementate classi apposite. In particolare, AuthorizationService si occupa dell'autenticazione e dell'autorizzazione della richiesta, mentre NotificationService astrae la relazione con il servizio di aggiornamento in tempo reale.

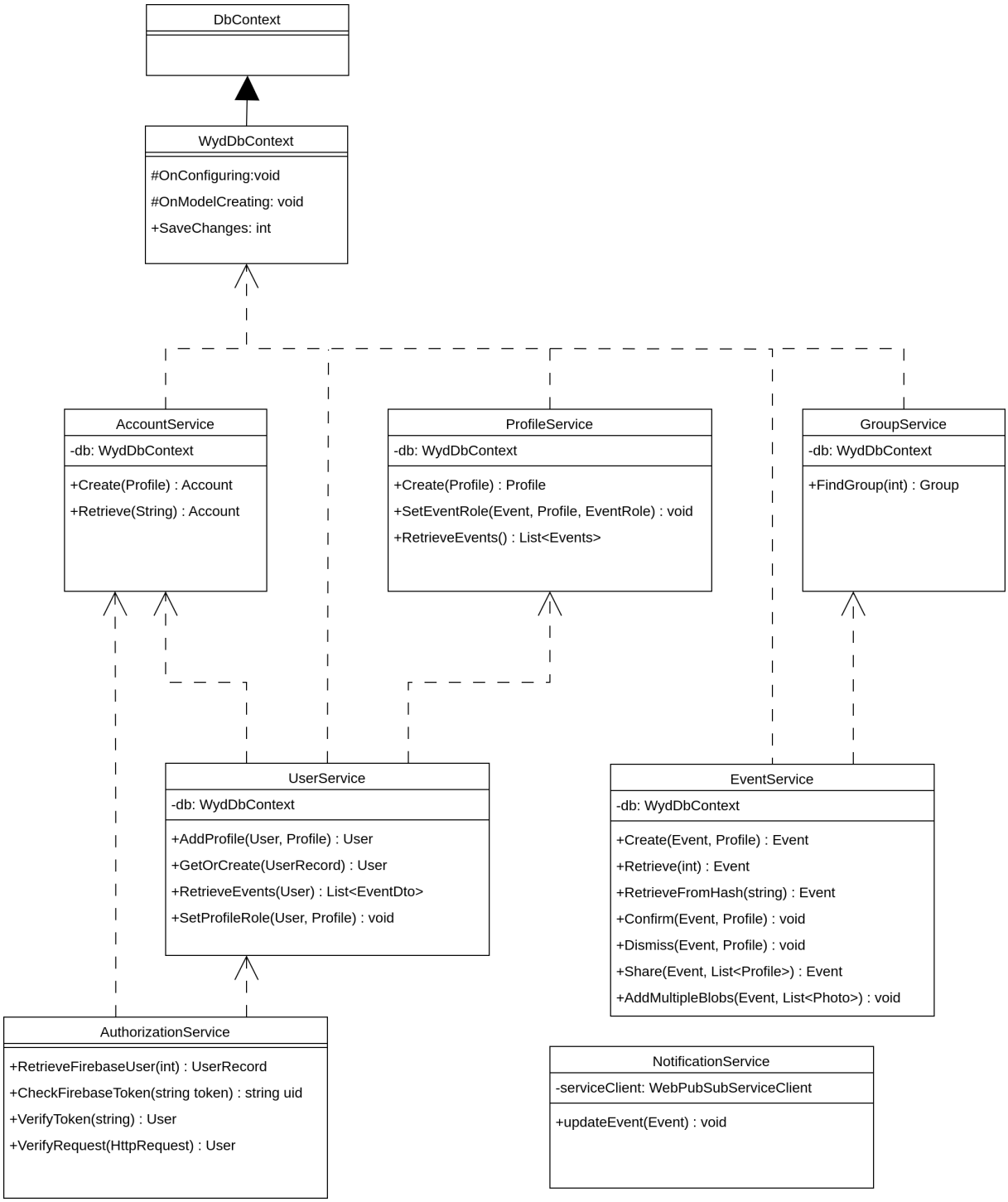

Figura 2: Modello delle classi del server

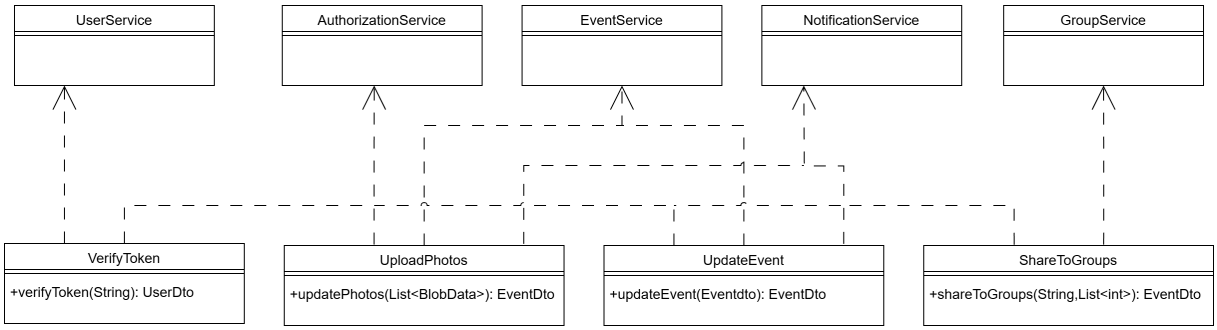

Figura 3: Modello delle relazioni tra Functions e i servizi usati

Per allineare i dati a disposizione del server con il dominio del client e per ridurre l'invio delle informazioni non necessarie, sono stati creati dei Data Transfer Object(DTO). I DTO sono classi logiche che prevedono almeno un costruttore che, dato l'elemento del dominio, ne copia solo le informazioni necessarie. Questo permette di creare rappresentazioni dei dati come necessarie al client, mascherando le logiche applicative e di fatto separando le dipendenze del dominio dai requisiti di comunicazione.

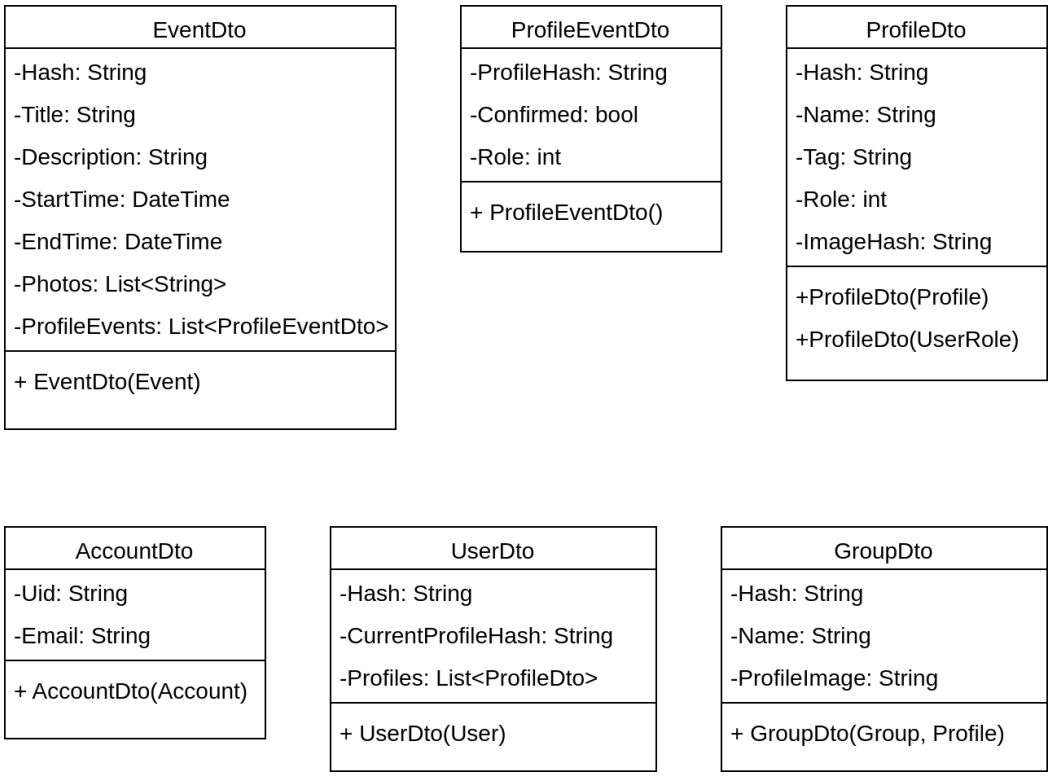

Figura 4: Modello delle classi dei data tranfer object

Per ogni metodo pubblico delle classi service sono stati implementati dei test. I test permettono la simulazione di differenti situazioni per controllare che il codice segua il comportamento desiderato. La loro implementazione è quindi precedente allo sviluppo stesso delle classi, in quanto le aspettative sono già note, e il superamento dei test determina la correttezza del metodo. Inoltre, in caso di necessità particolari che escono dalle normali aspettative della funzione, quali, ad esempio, il controllo di un valore particolare o l'implementazione di un vincolo specifico, i test assicurano la loro futura presa in carico anche in caso di modifica totale del codice.

```
5 references
private static WydDbContext GetInMemoryDbContext()
{
    var options = new DbContextOptionsBuilder<WydDbContext>()
        .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())
        .Options;

    return new WydDbContext(options);
}
```

Figura 5: Simulazione del database

```
[Fact]
0 references
public void Create_ValidAccount_ReturnsAccount()
{
    var dbContext = GetInMemoryDbContext();
    var service = new AccountService(dbContext);

    var newUser = new User { };
    dbContext.Users.Add(newUser);
    dbContext.SaveChanges();

    var account = new Account
    {
        Mail = "test@example.com",
        Uid = "uid123",
        User = newUser
    };

    var result = service.Create(account);

    Assert.NotNull(result);
    Assert.Equal("test@example.com", result.Mail);
    Assert.Equal("uid123", result.Uid);
}
```

Figura 6: Test di creazione di un account

0.1.3 La distribuzione

Lo sviluppo è stato condotto utilizzando Visual Studio Code, programma sviluppato dalla stessa Microsoft per la creazione di codice. Visual Studio Code permette l'integrazione con molteplici estensioni fornendo il supporto per la maggior parte delle tecnologie.

In particolare, grazie alle estensioni dedicate al provider Azure, è possibile collegare il proprio ambiente di lavoro con i servizi in cloud. Il legame così creato consente un aggiornamento immediato ed intuitivo del codice, gestito interamente dal programma.

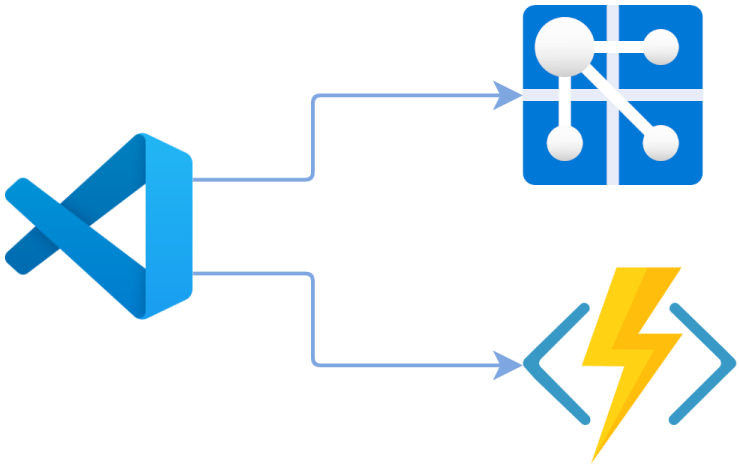

Figura 7: Diagramma di aggiornamento e distribuzione del server