

Metodologie e sistemi di caching



API REST

Un'API REST (Representational State Transfer) è un'interfaccia che consente a diverse applicazioni di comunicare tra loro attraverso il web, seguendo alcuni principi architetturali.

Utilizza i verbi HTTP standard (come GET, POST, PUT, DELETE) per operare su risorse identificate tramite URL.

Le risposte sono spesso in formato JSON o XML e sono progettate per essere stateless, il che significa che ogni richiesta contiene tutte le informazioni necessarie per essere gestita, senza dipendere dallo stato della sessione del server.

Creato da Roy Fielding nel 2000 basandosi sui Principi di design derivati dall'architettura web

Principi Fondamentali di REST

- Stateless: Ogni richiesta è indipendente
- Client-Server: Separazione delle preoccupazioni
- Cacheable: Le risposte devono essere definibili come cacheabili o non-cacheabili

Verbi HTTP nelle API REST

GET: Recupera dati

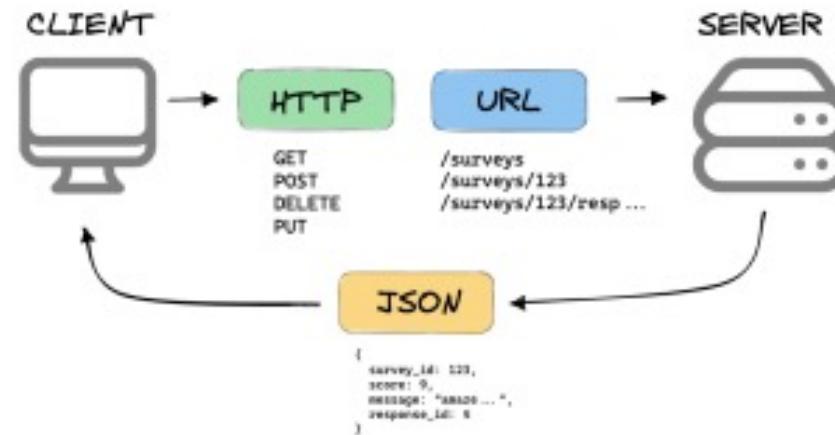
POST: Crea nuovi dati

PUT: Aggiorna dati esistenti

DELETE: Rimuove dati

PATCH: Modifica parziale dei dati

WHAT IS A REST API?



mannhowie.com

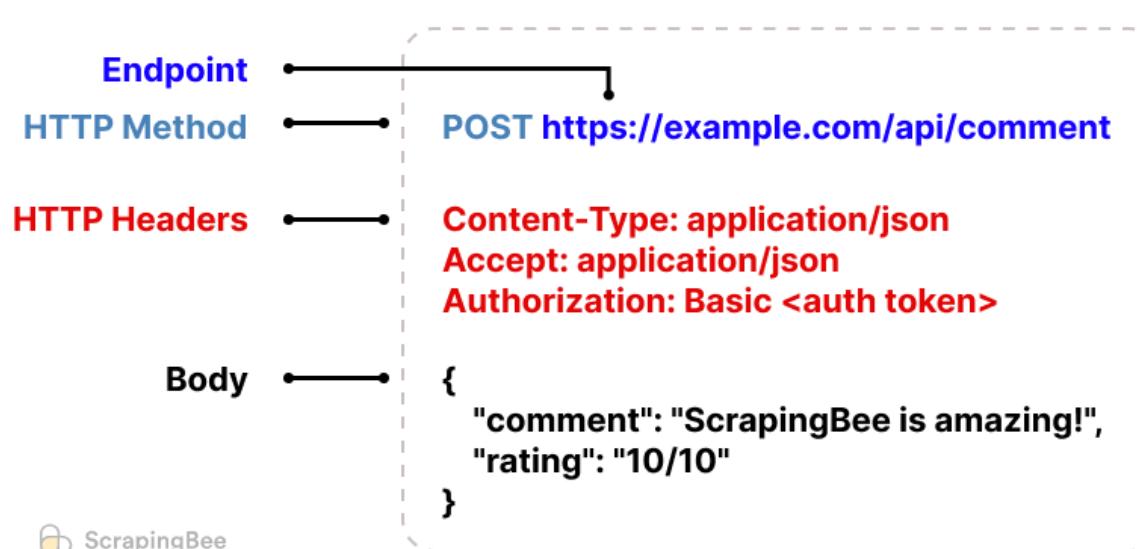
HTTP

Header HTTP nelle API REST

- Content-Type: Tipo di contenuto della richiesta/risposta
- Authorization: Per autenticazione
- Accept: Tipo di contenuto che il client è disposto a ricevere

Body delle Richieste

- Usato con POST, PUT, PATCH
- Formato: JSON, XML, Form Data
- Esempio di payload JSON per POST



Codici di Stato HTTP nelle API REST

- 200 OK: Successo
- 201 Created: Risorsa creata
- 204 No Content: Successo senza contenuto
- 400 Bad Request: Richiesta errata
- 404 Not Found: Risorsa non trovata
- 500 Internal Server Error: Errore del server

Esempio richieste

GET

URL: /products/123

Header: Accept: application/json

Risposta: Dettagli del prodotto in formato JSON

Risposta: Codice 200 OK

PUT

URL: /products/123

Header: Content-Type: application/json

Body: { "name": "Updated Product", "price": 29.99 }

Risposta: Codice 200 OK

POST

URL: /products

Header: Content-Type: application/json

Body: { "name": "New Product", "price": 19.99 }

Risposta: Codice 201 Created

DELETE

URL: /products/123

Risposta: Codice 204 No Content

Best Practices nella Progettazione di API REST

Utilizzare nomi di risorse al plurale

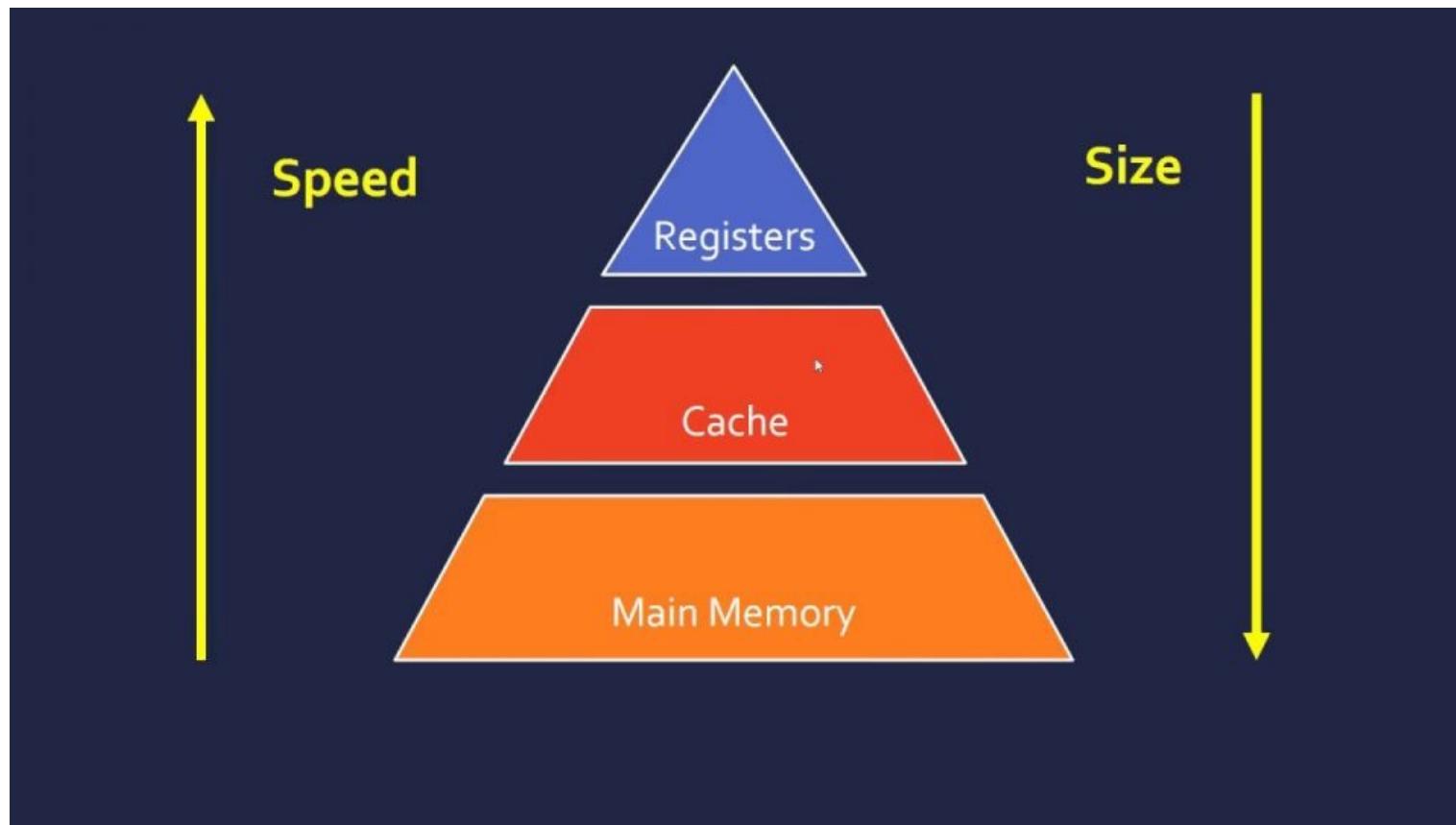
Versionare l'API (es. /v1/products)

Utilizzare codici di stato HTTP appropriati

Introduzione alla Caching dei Dati

- **Definizione di Caching:** La cache è un'area di archiviazione temporanea utilizzata per velocizzare l'accesso ai dati richiesti di frequente.
- **Obiettivi della Caching:**
- - Riduzione della latenza: Rispondere rapidamente alle richieste degli utenti.
- - Riduzione del carico sui server di backend: Meno chiamate ai database o servizi esterni.
- - Miglioramento delle prestazioni delle applicazioni: Esperienza utente più fluida.

Gerarchia memoria



Principi di base

Località temporale

Località spaziale

Località Temporale

- **Definizione:** Il concetto di località temporale si riferisce alla probabilità che un dato elemento venga riutilizzato in un intervallo di tempo relativamente breve.
- **Esempio:** In un'applicazione web, se un utente richiede una pagina, è probabile che richieda di nuovo la stessa pagina o un'azione correlata in un breve lasso di tempo.

Utilizzo nella Cache:

- **Cache di Query:** Le risposte alle query frequenti di un database vengono memorizzate, poiché è probabile che vengano richieste di nuovo entro un breve periodo.
- **Session Store:** Le informazioni della sessione utente vengono memorizzate in cache, in quanto saranno utilizzate ripetutamente durante una sessione attiva.

Località Spaziale

Definizione: La località spaziale si riferisce alla probabilità che elementi di dati vicini a quelli appena utilizzati vengano richiesti subito dopo.

- **Esempio:** Se un utente richiede un file all'interno di una directory, è probabile che richieda anche altri file all'interno della stessa directory.

Utilizzo nella Cache:

- **Cache di Pagina:** Nei sistemi di memoria virtuale, se un blocco di dati (pagina) viene caricato in memoria, è probabile che le pagine adiacenti vengano richieste a breve.
- **Prefetching:** Tecniche di prefetching possono essere utilizzate per caricare in anticipo i dati correlati nella cache, migliorando le prestazioni di accesso ai dati.

Strategie di Caching dei Dati

Cache per richiesta (Per-Request Caching):

- **Esempio:** Memorizzazione delle risposte delle API REST in cache per richieste successive, riducendo il carico sul server.

- **Cache basata su tempo (TTL - Time-to-Live):**

- **Esempio:** Configurazione di una cache con TTL di 5 minuti per contenuti generati dinamicamente, come le previsioni del tempo.

- **Cache di contenuto statico:**

- **Esempio:** Utilizzo di una CDN per cache di immagini, CSS, e JavaScript per ridurre i tempi di caricamento delle pagine web.

- **Cache distribuita:**

- **Esempio:** Utilizzo di un cluster Redis distribuito per memorizzare le sessioni degli utenti in un'applicazione web, garantendo la persistenza delle sessioni in caso di failover.

Sistemi Open Source per la Caching

- **Redis:**
 - Descrizione di Redis come sistema di caching.
 - Caratteristiche principali: in-memory data structure store, supporto per strutture dati avanzate (hash, liste, set).
- Utilizzo di Redis in Kubernetes: Redis operator e Redis cluster.
- **Infinispan:**
 - Introduzione a Infinispan e alle sue caratteristiche.
 - Architettura distribuita di Infinispan.
 - Integrazione con Kubernetes: operator di Infinispan.



Architettura della Cache in Kubernetes

1. Redis Deployment in Kubernetes:

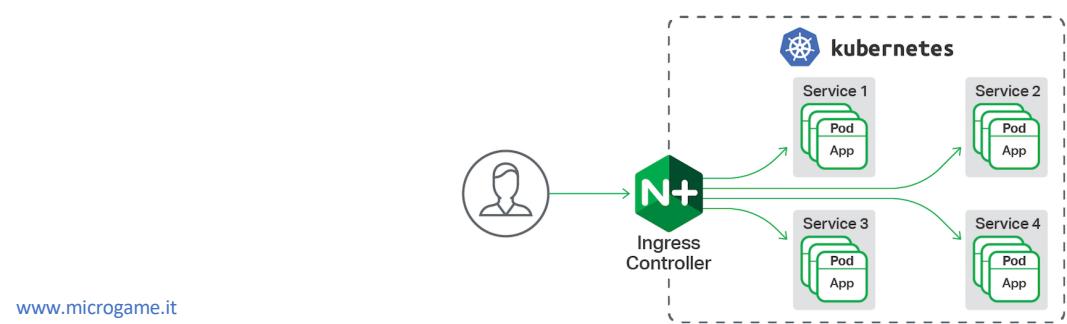
- **Pod Redis:** Un singolo Pod può essere utilizzato per eseguire un'istanza Redis standalone per caching semplice e a basso costo.
- **Redis Cluster:** Redis può essere distribuito come un cluster su più Pod per supportare il partizionamento dei dati (sharding) e la replica, migliorando scalabilità e tolleranza ai guasti.

2. Caching con Redis Operator:

- **Redis Operator:** Automatizza la gestione del ciclo di vita di Redis, inclusi il deploy, il scaling e il failover, all'interno di Kubernetes.

3. Componenti Coinvolti:

- **Service:** Espone Redis ai Pod dell'applicazione, permettendo di accedere alla cache tramite DNS interno.
- **PersistentVolume & PersistentVolumeClaim:** Garantiscono la persistenza dei dati Redis anche in caso di riavvio dei Pod.
- **ConfigMap & Secret:** Utilizzati per gestire la configurazione di Redis e le credenziali in modo sicuro.



Introduzione a Redis

Cos'è Redis?

Redis (Remote Dictionary Server) è un **database NoSQL in-memory** open-source, utilizzato principalmente come **database, cache, e broker di messaggi**.

- Supporta diverse **strutture dati** come stringhe, hash, liste, set, sorted set, bitmaps, hyperloglogs e stream.
- È noto per la sua **elevata velocità e bassa latenza** grazie all'archiviazione in memoria.

Architettura di Redis

Come funziona Redis?

- Redis è basato su un'architettura **client-server**, dove i client inviano comandi al server Redis.
 - **In-Memory**: Tutti i dati sono memorizzati in memoria (RAM), con opzioni di persistenza su disco.
- **Single-threaded**: Redis elabora i comandi in un singolo thread per evitare il locking e semplificare la gestione della concorrenza.

Principali Strutture Dati in Redis

Strutture Dati Supportate

- **Stringhe:** Tipo di dato base, utilizzato per memorizzare valori semplici.
- **Liste:** Sequenze ordinate di stringhe, ideali per code (FIFO) o pile (LIFO).
- **Set:** Collezioni non ordinate di stringhe, utilizzate per garantire l'unicità degli elementi.
 - **Hash:** Collezioni di coppie chiave-valore, simili a dizionari.
 - **Sorted Set:** Set ordinati in base a un punteggio associato agli elementi.
- **Streams:** Struttura per la gestione di flussi di dati, utile per scenari di log o code di messaggi

Persistenza dei Dati in Redis

Opzioni di Persistenza

- **RDB (Redis Database Backup)**: Salvataggi periodici dei dati in file dump su disco, utilizzati per il backup e il ripristino.
- **AOF (Append-Only File)**: Registra ogni operazione di scrittura, permettendo una maggiore affidabilità rispetto a RDB.
- **Hybrid Persistence**: Combinazione di RDB e AOF per ottimizzare il bilanciamento tra velocità di scrittura e affidabilità.

Redis come Sistema di Cache

Implementare Redis come Cache

- **Caching:** Redis viene utilizzato per memorizzare temporaneamente i dati ad accesso frequente per ridurre il carico sui sistemi backend.
 - **TTL (Time-to-Live):** Impostazione di una scadenza per le chiavi nella cache per garantire che i dati non diventino obsoleti.
- **Eviction Policy:** Redis supporta varie politiche di espulsione per gestire la memoria, come LRU (Least Recently Used), LFU (Least Frequently Used), e volatile-ttl.

Redis Cluster

Scalabilità con Redis Cluster

- **Sharding:** Redis Cluster distribuisce i dati su più nodi, dividendo le chiavi in “hash slots”.
- **Replica:** Ogni nodo in Redis Cluster può avere uno o più nodi replica per garantire l’alta disponibilità.
 - **Failover Automatico:** In caso di guasto di un nodo, una replica può assumere automaticamente il ruolo del nodo principale.

Redis Pub/Sub e Messaggistica

Redis Pub/Sub

- **Pub/Sub:** Redis supporta il modello Publish/Subscribe per la messaggistica, dove i client possono pubblicare messaggi su un canale e altri client possono sottoscriversi a tali canali per ricevere messaggi in tempo reale.
- **Utilizzo:** Pub/Sub è utilizzato per notifiche, aggiornamenti in tempo reale, e comunicazione inter-servizi in architetture di microservizi.

Redis Streams

Gestione di Flussi di Dati con Redis Streams

- **Streams:** Struttura dati per la gestione di flussi di eventi in tempo reale, simile a un log di eventi.
- **Consumer Groups:** Permette la gestione di lettori concorrenti per processare i messaggi di un flusso.
- **Applicazioni:** Utilizzato per sistemi di log, code di messaggi, e elaborazioni di dati in tempo reale.

Redis in Kubernetes

Redis in Ambienti Kubernetes

- **Redis in Pods:** Redis può essere distribuito in un ambiente Kubernetes usando Pod, con opzioni di scaling e gestione tramite Kubernetes.
- **Redis Operator:** Facilita la gestione del ciclo di vita di Redis, incluso deploy, scaling, e failover all'interno di Kubernetes.
- **Persistenza:** Utilizzo di Persistent Volumes per garantire la persistenza dei dati Redis nel cluster.

Vantaggi e Use Cases di Redis

Vantaggi e Casi d'Uso

- **Elevata Performance:** Redis offre tempi di risposta rapidi grazie all'archiviazione in memoria.
- **Scalabilità:** Redis Cluster e la possibilità di aggiungere repliche permettono di scalare facilmente il sistema.
- **Casi d'Uso:**
- **Caching:** Miglioramento delle performance delle applicazioni.
- **Session Storage:** Gestione di sessioni utente in applicazioni web.
- **Rate Limiting:** Implementazione di limiti di accesso a servizi API.
- **Code di Messaggi:** Utilizzo come broker di messaggi leggero e rapido.



Infinispan

Introduzione a Infinispan

Cos'è Infinispan?

Infinispan è una piattaforma di caching distribuita e un data grid in-memory open-source sviluppata da Red Hat.

- Fornisce una cache distribuita, che permette di migliorare le prestazioni delle applicazioni riducendo il carico sui database backend.
- Supporta l'archiviazione di dati in memoria, con opzioni per la persistenza, scalabilità e alta disponibilità.

Architettura di Infinispan

Architettura Distribuita di Infinispan

- Data Grid: Infinispan distribuisce i dati su più nodi, creando una griglia di dati distribuita.
- Cache Distribuita: I dati possono essere replicati o distribuiti tra i nodi, garantendo scalabilità e tolleranza ai guasti.
- Cluster di Nodi: Infinispan può funzionare in modalità cluster, dove i dati sono distribuiti tra più nodi per migliorare la resilienza.

Modalità di Caching

Modalità di Caching in Infinispan

- Embedded Mode: Infinispan viene eseguito all'interno della JVM dell'applicazione, fornendo caching locale e rapido.
- Client-Server Mode: Infinispan viene eseguito in modalità server e le applicazioni accedono ai dati attraverso un client.
- Remote Cache: Le applicazioni client possono connettersi a un cluster Infinispan remoto per caching distribuito.

Gestione della Persistenza

Persistenza dei Dati in Infinispan

- Persistence Store: Infinispan supporta la persistenza su disco tramite diversi store, come file system, database relazionali, o NoSQL.
- Write-Through: Le scritture alla cache sono immediatamente sincronizzate con il sistema di persistenza.
- Write-Behind: Le scritture alla cache sono bufferizzate e scritte in modo asincrono nel sistema di persistenza.

Configurazione e API di Infinispan

Configurazione e API di Infinispan

- Configurazione XML/Programmatica: Infinispan può essere configurato tramite file XML o programmaticamente attraverso la sua API.
- CacheManager: Gestisce le istanze di cache, permettendo di creare, configurare e gestire diverse cache.
- API Java: Infinispan offre un'API Java intuitiva per interagire con le cache, includendo operazioni di base come put, get, e remove.

Consistenza e Disponibilità

Consistenza e Disponibilità in Infinispan

- Consistenza Eventuale: I dati sono replicati tra i nodi, e le letture possono non riflettere immediatamente le ultime scritture, migliorando la disponibilità.
- Replica Sincrona: I dati sono replicati immediatamente tra i nodi, garantendo la consistenza dei dati a discapito della latenza.
- Replica Asincrona: I dati sono replicati in modo asincrono, migliorando la velocità ma con possibili incoerenze temporanee.

Scalabilità e Bilanciamento del Carico

- **Partizionamento Dati (Sharding):** Infinispan divide i dati tra i nodi del cluster, permettendo di gestire grandi volumi di dati.
- **Scalabilità Orizzontale:** È possibile aggiungere nodi al cluster senza downtime, ridistribuendo i dati automaticamente.
- **Bilanciamento del Carico:** Le richieste sono distribuite tra i nodi per evitare sovraccarichi, garantendo prestazioni stabili.

Funzionalità Avanzate

Funzionalità Avanzate di Infinispan

- Transazioni: Supporto per transazioni ACID, con integrazione XA per transazioni distribuite.
- Eventi e Notifiche: Infinispan può notificare le applicazioni di eventi di modifica dei dati, come inserimenti o aggiornamenti.
- Querying: Supporto per query avanzate sui dati in cache, utilizzando DSL o linguaggi come JPA.

Infinispan in Kubernetes

Esecuzione di Infinispan in Kubernetes

- Infinispan Operator: Gestisce il deploy, il scaling e il monitoraggio di Infinispan in un cluster Kubernetes.
- Configurazione Automatica: L'operator facilita la configurazione di Infinispan su Kubernetes, includendo l'integrazione con Persistent Volumes.
- Alta Disponibilità: Infinispan può essere distribuito su più zone di disponibilità per garantire resilienza e tolleranza ai guasti.

Casi d'Uso e Vantaggi di Infinispan

- Caching Distribuito: Miglioramento delle performance delle applicazioni riducendo il carico sui sistemi backend.
- Data Grid In-Memory: Archiviazione di grandi volumi di dati in memoria per accessi rapidi e analisi in tempo reale.
- Event-Driven Applications: Utilizzo in applicazioni basate su eventi, dove le modifiche ai dati possono innescare azioni immediate.

Vantaggi:

- Elevata Scalabilità: Facile scalabilità orizzontale senza downtime.
- Alta Disponibilità: Resilienza e tolleranza ai guasti con replica e partizionamento dei dati.
- Versatilità: Supporto per diversi modelli di caching e archiviazione dei dati.

Cache in Cloud



Cache Cloud



Cache in Cloud - Sistemi CDN (Content Delivery Network)

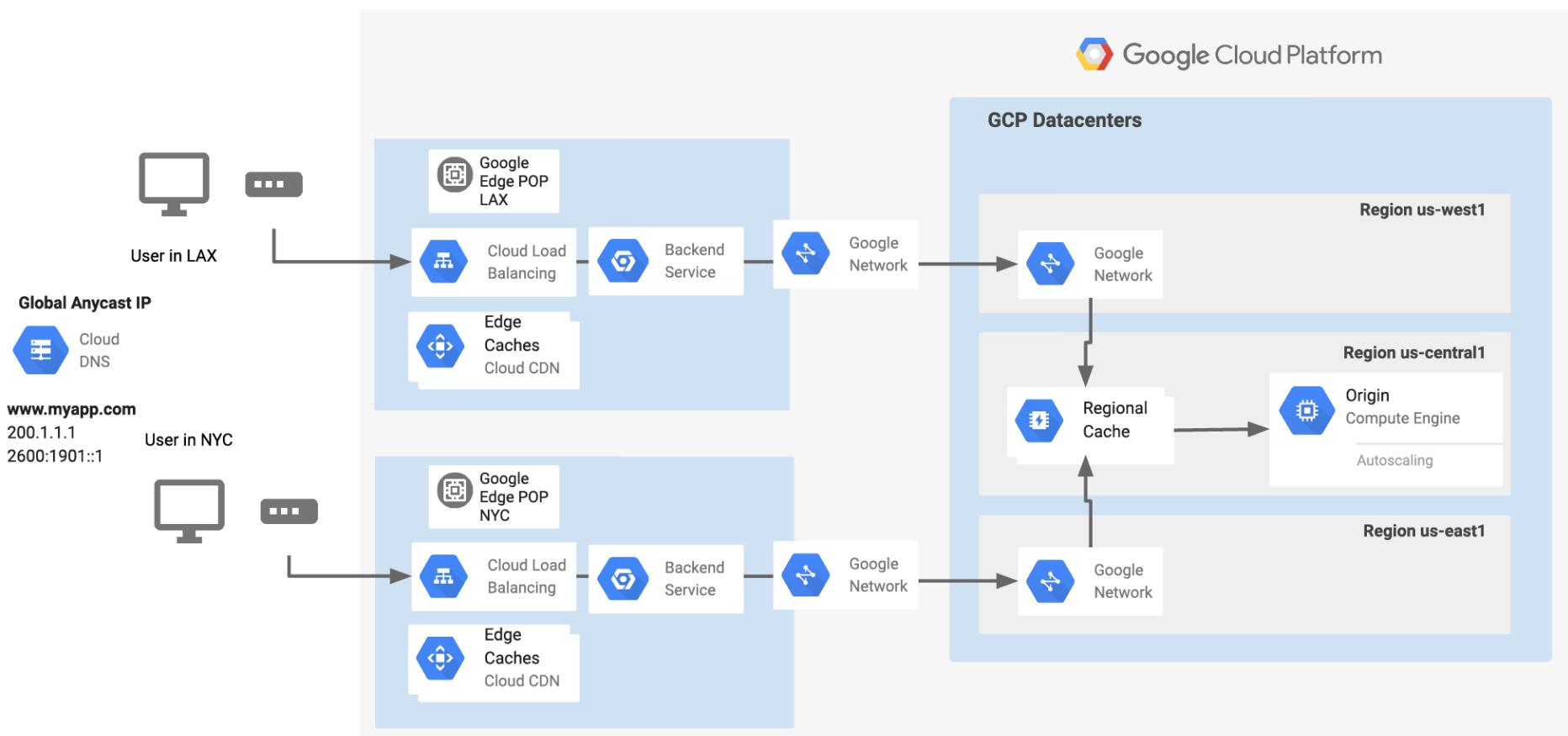
CDN Overview: Cosa sono i CDN e perché sono essenziali per la distribuzione globale dei contenuti.

- **Vantaggi dell'uso di CDN:** Bassa latenza, alta disponibilità, sicurezza migliorata.
- **Principali provider di CDN:** Introduzione a Google Cloud CDN, AWS CloudFront, e Akamai.

Google Cloud CDN

- **Funzionamento:** Come Google Cloud CDN lavora con Google Cloud Storage e Compute Engine.
- **Caratteristiche chiave:** Caching geografico, bilanciamento del carico, integrazione con servizi Google.
- **Esempi di implementazione:** Configurazione di un CDN con Google Cloud.

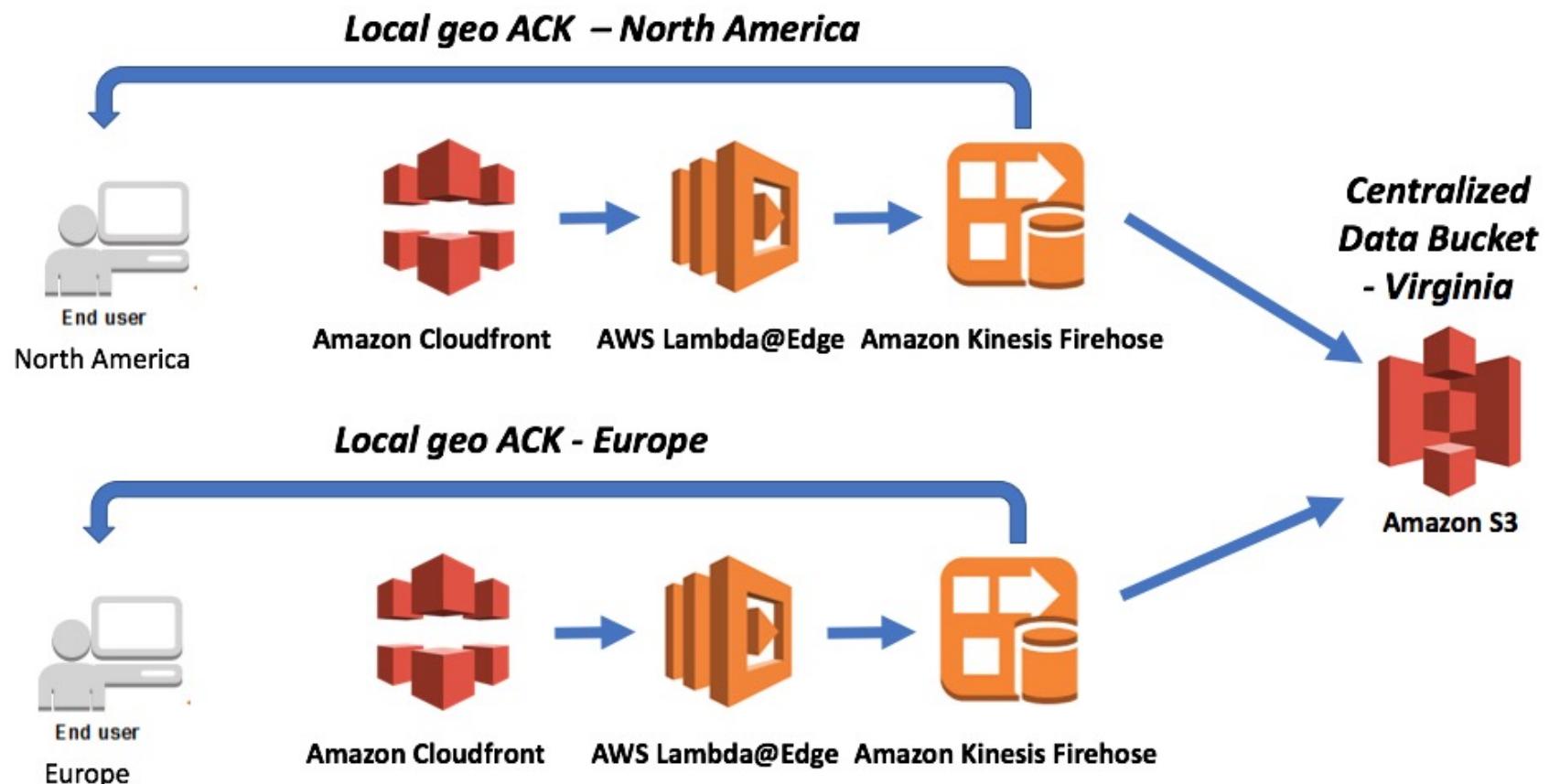
Google Cloud CDN



AWS - Amazon CloudFront

- **Overview di CloudFront:** Come Amazon CloudFront distribuisce contenuti in modo sicuro e rapido.
- **Caratteristiche principali:** Supporto per TLS/SSL, integrazione con AWS Shield per protezione DDoS, Lambda@Edge.
- **Esempi di utilizzo:** Configurazione di CloudFront con S3 e AWS API Gateway.

AWS - Amazon CloudFront

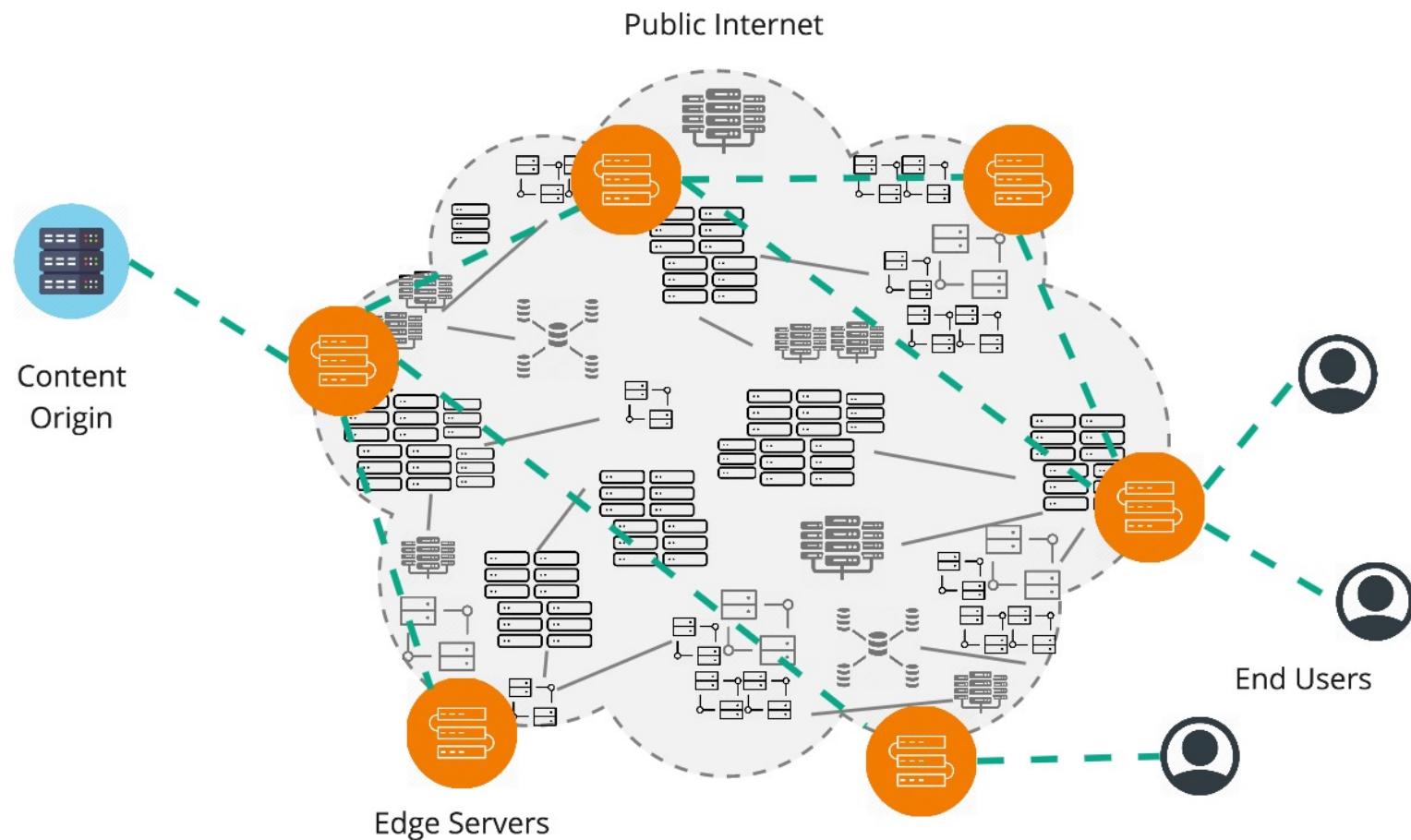


Akamai

Descrizione di Akamai: Leader nel mercato CDN, con un focus su prestazioni e sicurezza.

- **Caratteristiche chiave:** Distribuzione globale, ottimizzazione delle prestazioni web, sicurezza avanzata.
- **Casi d'uso:** Esempi di come grandi aziende utilizzano Akamai per migliorare l'esperienza utente.

Akamai



Esercitazione

Sviluppo di una API sicura e performante che gestisce un catalogo di prodotti con particolare attenzione ai prodotti dedicati ai clienti VIP.

Confrontare in maniera critica le varie implementazioni in termini di sicurezza e performance

SW e Tool da installare:

VSCode (<https://code.visualstudio.com/download>)

.NET 8.0 (<https://dotnet.microsoft.com/it-it/download/dotnet/8.0>)

Docker Desktop (<https://www.docker.com/products/docker-desktop/>)

Redis (https://hub.docker.com/_/redis)

Redis Insight (<https://redis.io/insight/>)

Keycloak (<https://hub.docker.com/r/bitnami/keycloak>)

Postman (<https://www.postman.com>)

Esercitazione

Punti chiave:

Autorizzazione

Solo gli utenti di tipo VIP (claim) possono ricevere l'elenco completo dei prodotti

Performance

I dati del catalogo utente devono essere memorizzati in cache per evitare chiamate ripetute alla base di dati

--

Il catalogo dei prodotti è memorizzato in un formato json e presente localmente al webserver

Creazione progetto

Terminale VSCode

Riepilogo dei Comandi

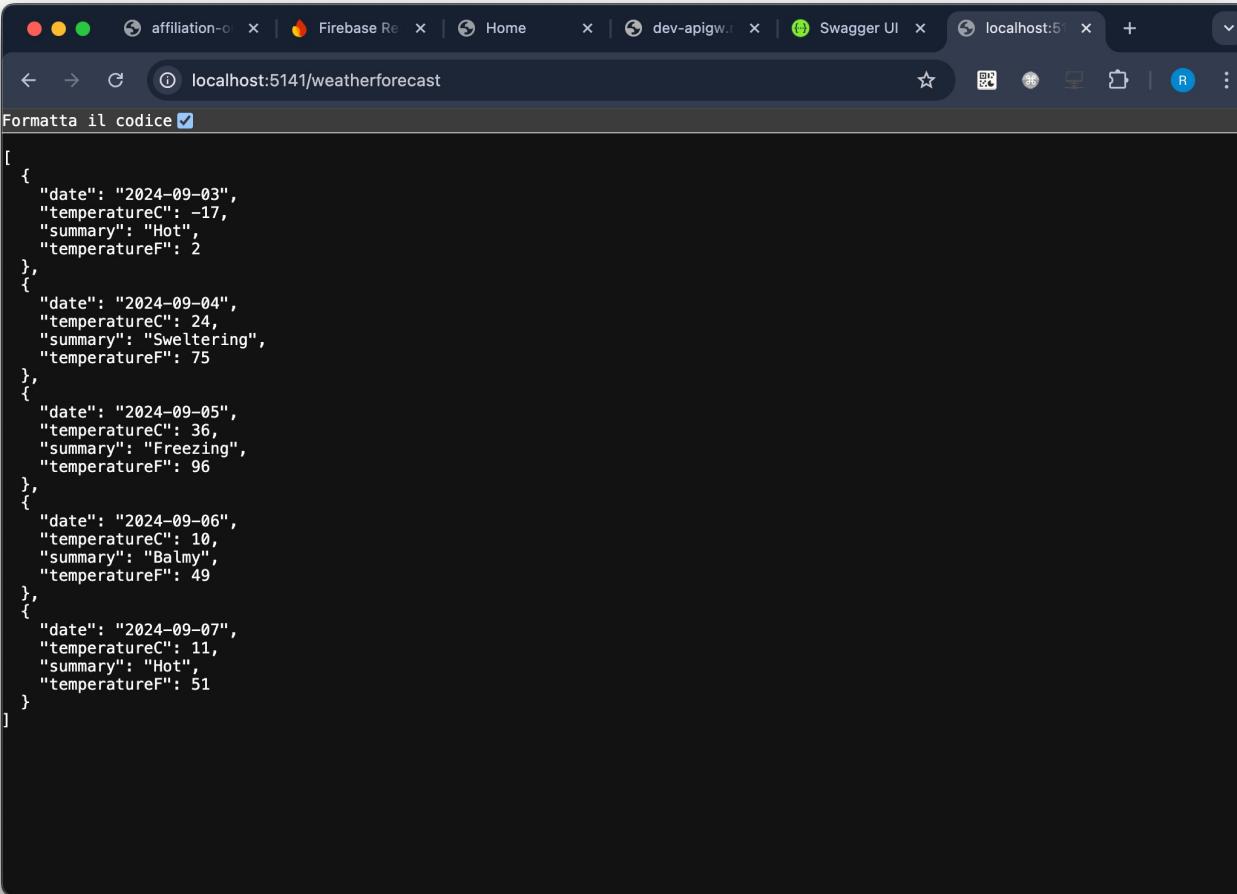
```
bash
dotnet new webapi -n IFTSApi
code IFTSApi
dotnet run
```

Copia

Questi passaggi ti permetteranno di creare un nuovo progetto Web API con .NET 8 utilizzando Visual Studio Code.

Lancio progetto

Terminale VSCode – dotnet run



The screenshot shows a web browser window with the URL `localhost:5141/weatherforecast` in the address bar. The page content displays a JSON array of weather forecast data. Each element in the array contains a date, temperature in Celsius, a summary, and a temperature in Fahrenheit.

```
[{"date": "2024-09-03", "temperatureC": -17, "summary": "Hot", "temperatureF": 2}, {"date": "2024-09-04", "temperatureC": 24, "summary": "Sweltering", "temperatureF": 75}, {"date": "2024-09-05", "temperatureC": 36, "summary": "Freezing", "temperatureF": 96}, {"date": "2024-09-06", "temperatureC": 10, "summary": "Balmy", "temperatureF": 49}, {"date": "2024-09-07", "temperatureC": 11, "summary": "Hot", "temperatureF": 51}]
```

Creazione api /catalog

```
39     app.MapGet("/catalog", () =>
40     {
41         var elements = Enumerable.Range(1, 5).Select(index =>
42             new Item
43             {
44                 Name = $"Item {index}",
45                 Description = $"Description {index}"
46             })
47             .ToArray();
48
49         return elements;
50     })
51     .WithName("GetCatalog")
52     .WithOpenApi();
53
54     app.Run();
55
56 > record WeatherForecast(DateOnly Date, int TemperatureC, string? Summary) ...
57
58
59
60
61     public class Item
62     {
63         public string Name { get; set; }
64         public string Description { get; set; }
65     }
66
```

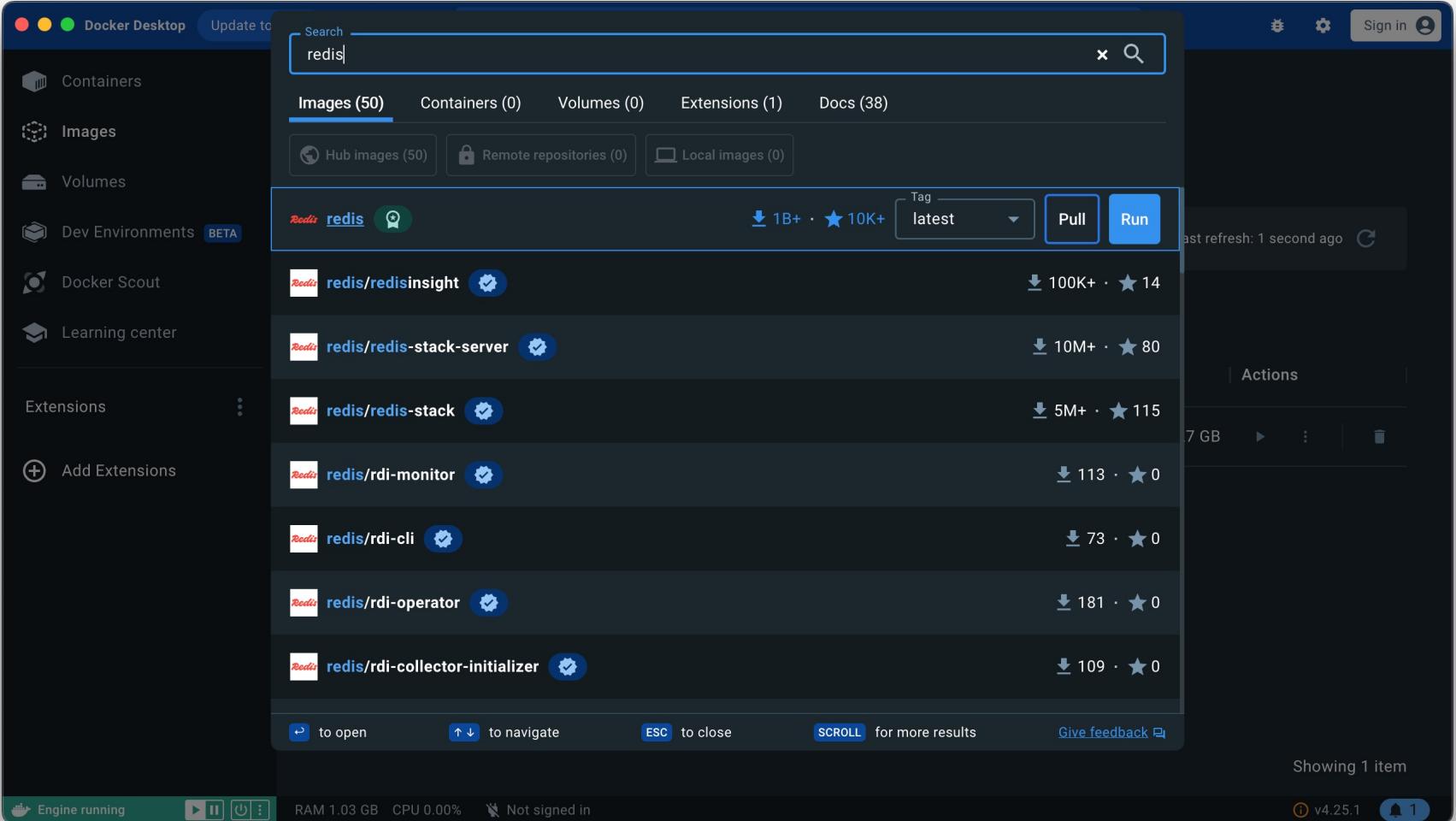
Creazione catalog - json

```
{} catalog.json > ...
1 [ 
2 {
3   "name": "Elemento1",
4   "description": "Questa è la descrizione del primo elemento."
5 },
6 {
7   "name": "Elemento2",
8   "description": "Questa è la descrizione del secondo elemento."
9 },
10 {
11   "name": "Elemento3",
12   "description": "Questa è la descrizione del terzo elemento."
13 },
14 {
15   "name": "Elemento4",
16   "description": "Questa è la descrizione del quarto elemento."
17 }
18 ]
```

Lettura catalog

```
22
23 app.MapGet("/catalog", async () =>
24 {
25     // Legge il file catalog.json
26     var json = await File.ReadAllTextAsync("catalog.json");
27
28     // Deserializza il contenuto JSON in una lista di oggetti
29     var elementi = JsonSerializer.Deserialize<List<Elemento>>(json);
30
31     // Seleziona i primi 3 elementi
32     var primiTreElementi = elementi.Take(3);
33
34     // Restituisce i primi 3 elementi come risposta JSON
35     return Results.Json(primiTreElementi);
36 });
37
38 app.Run();
39
40 public class Elemento
41 {
42     [JsonPropertyName("name")]
43     public string Name { get; set; }
44     [JsonPropertyName("description")]
45     public string Description { get; set; }
46 }
```

Cache - setup

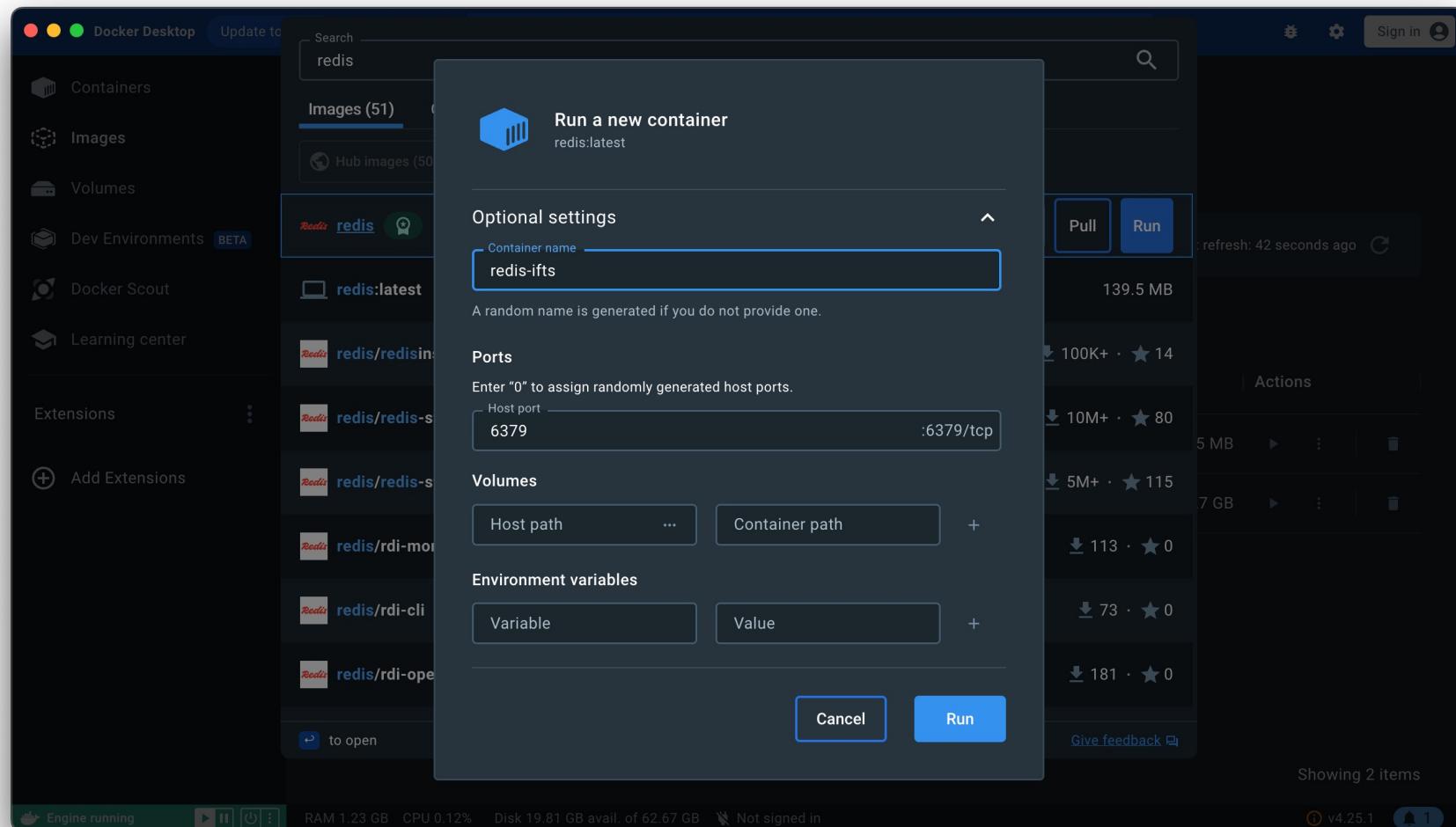


The screenshot shows the Docker Desktop interface with a search bar at the top containing the text "redis". Below the search bar, there are tabs for "Images (50)", "Containers (0)", "Volumes (0)", "Extensions (1)", and "Docs (38)". The "Images (50)" tab is selected. Under this tab, there are three categories: "Hub images (50)", "Remote repositories (0)", and "Local images (0)". The main list displays several Redis-related images:

Image	Downloads	Stars
redis/redis	1B+	10K+
redis/redis/redisinsight	100K+	14
redis/redis/redis-stack-server	10M+	80
redis/redis/redis-stack	5M+	115
redis/redis/rdi-monitor	113	0
redis/redis/rdi-cli	73	0
redis/redis/rdi-operator	181	0
redis/redis/rdi-collector-initializer	109	0

At the bottom of the search results, there are navigation instructions: "to open", "to navigate", "ESC to close", "SCROLL for more results", and "Give feedback". The status bar at the bottom of the screen shows "Engine running", "RAM 1.03 GB CPU 0.00%", "Not signed in", "v4.25.1", and a notification icon with the number 1.

Cache - run





Cache - check

The screenshot shows the Redis UI interface at 127.0.0.1:6379. The left sidebar has icons for Key, Edit, Histogram, and Cluster. The main area displays a list of keys: sample_jobQueue (19%, 4), sample_leaderboard (5%, 1), and sample_session (71%, 15). A selected key is test_ifts, which is a STRING type with a value of 80 B. The top right shows system metrics: 0.71% CPU, 0 operations, 1 MB memory, 21 connections, and 2 clients. The bottom navigation bar includes links for CLI, Command Helper, and Profiler.

127.0.0.1:6379 - Browser

Databases > 127.0.0.1:6379 db0 🖊 ⓘ

All Key Types Filter by Key Name or Pattern

Results: 21. Scanned 21 / 21

Key	Value Type	Value	Size
sample_jobQueue	STRING	...	19%
sample_leaderboard	STRING	...	5%
sample_session	STRING	...	71%
test_ifts	STRING	test_ifts	8 min 80 B

Last refresh: < 1 min

Insights

Tutorials Tips

Redis tutorials

- Redis basic and RAG use cases
- Data structures explained
- How to query your data
 - Introduction
 - Exact match
 - Full-text search
 - Range queries
 - Geospatial queries
 - Combined queries
 - Analytic and transformative queries
 - Learn more
- Vector search explained

My tutorials

Create your tutorial ↗ + Upload

Let us know what you think

Cache - StackExchange

1. Aggiunta del Pacchetto NuGet StackExchange.Redis

Prima di tutto, aggiungi il pacchetto StackExchange.Redis al tuo progetto .NET. Puoi farlo eseguendo il seguente comando nel terminale del tuo progetto:

```
bash
```

```
dotnet add package StackExchange.Redis
```

 Copia

Cache – Connessione a redis

```
11  
12     var app = builder.Build();  
13  
14     // Configurazione della connessione a Redis  
15     var redis = ConnectionMultiplexer.Connect("127.0.0.1:6379");  
16     var db = redis.GetDatabase();  
17  
18     // Configure the HTTP request pipeline.  
19     if (app.Environment.IsDevelopment())  
20     {  
21         app.UseSwagger();  
22         app.UseSwaggerUI();
```

Cache – Utilizzo di redis

```
28 app.MapGet("/catalog", async () =>
29 {
30     // Verifica se i dati sono già presenti in cache
31     string cachedCatalog = await db.StringGetAsync("catalog");
32
33     if (string.IsNullOrEmpty(cachedCatalog))
34     {
35         // Legge il file catalog.json
36         var json = await File.ReadAllTextAsync("catalog.json");
37
38         // Deserializza il contenuto JSON in una lista di oggetti
39         var elementi = JsonSerializer.Deserialize<List<Elemento>>(json);
40
41         // Serializza nuovamente per salvarlo in Redis
42         cachedCatalog = JsonSerializer.Serialize(elementi);
43
44         // Salva i dati in cache Redis con una scadenza di 2 minuti (120 secondi)
45         await db.StringSetAsync("catalog", cachedCatalog, TimeSpan.FromMinutes(2));
46     }
47
48     // Restituisce i dati del catalogo dalla cache Redis
49     return Results.Json(JsonSerializer.Deserialize<List<Elemento>>(cachedCatalog));
50 });
```

Cache – ritardo DB

```
if (string.IsNullOrEmpty(cachedCatalog))
{
    // Legge il file catalog.json
    var json = await File.ReadAllTextAsync("catalog.json");

    // Deserializza il contenuto JSON in una lista di oggetti
    var elementi = JsonSerializer.Deserialize<List<Elemento>>(json);

    // Serializza nuovamente per salvarlo in Redis
    cachedCatalog = JsonSerializer.Serialize(elementi);

    Thread.Sleep(10000); // Simula un ritardo di 10 secondi

    // Salva i dati in cache Redis con una scadenza di 2 minuti (120 secondi)
    await db.StringSetAsync("catalog", cachedCatalog, TimeSpan.FromMinutes(2));
}
```

MySQL

```
dotnet add package Pomelo.EntityFrameworkCore.MySql
```

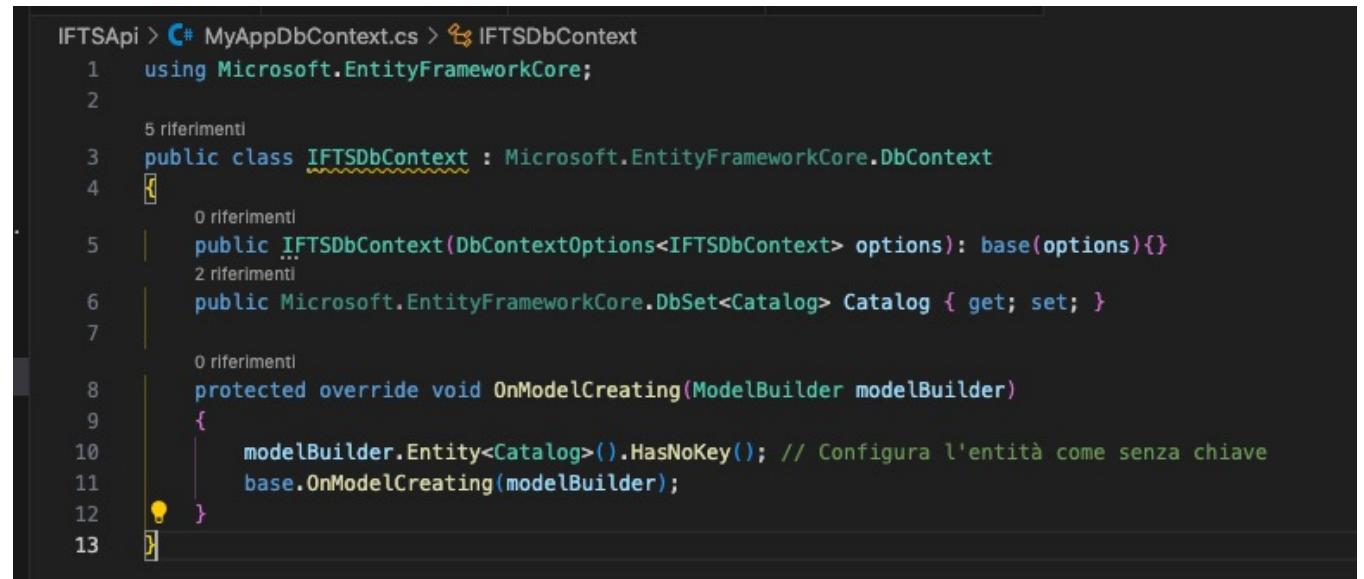
```
dotnet add package Microsoft.EntityFrameworkCore
```

```
dotnet add package EntityFramework
```

```
dotnet add package MySql.Data
```

appsettings.json

```
"ConnectionStrings": {  
    "MySqlConnection": "Server=localhost;Database=iftsdb;User=root;Password=ifts;"  
}
```



The screenshot shows a code editor window with the following code:

```
IFTSApi > C# MyAppDbContext.cs > IFTSDbContext  
1  using Microsoft.EntityFrameworkCore;  
2  
3  public class IFTSDbContext : Microsoft.EntityFrameworkCore.DbContext  
4  {  
5      public IFTSDbContext(DbContextOptions<IFTSDbContext> options) : base(options){}  
6      public Microsoft.EntityFrameworkCore.DbSet<Catalog> Catalog { get; set; }  
7  
8      protected override void OnModelCreating(ModelBuilder modelBuilder)  
9      {  
10         modelBuilder.Entity<Catalog>().HasNoKey(); // Configura l'entità come senza chiave  
11         base.OnModelCreating(modelBuilder);  
12     }  
13 }
```

MySQL

```
64 // Aggiungi il contesto del database (se usi EF Core)
65 builder.Services.AddDbContext<IFTSDbContext>(options =>
66     options.UseMySql(builder.Configuration.
67     GetConnectionString("MySQLConnection"),
68     ServerVersion.AutoDetect(builder.Configuration.
69     GetConnectionString("MySQLConnection"))));
70
71 var app = builder.Build();
72
```

```
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
```

```
app.MapGet("/catalog", async (HttpContext context, IFTSDbContext dbContext) =>
{
    // Verifica se una richiesta per il catalogo VIP
    var isVip = Boolean.TryParse(context.Request.Query["vip"], out var vip) && vip;
    var keyForCatalog = isVip ? "catalog-vip" : "catalog";

    // Verifica se i dati sono già presenti in cache
    string cachedCatalog = await db.StringGetAsync(keyForCatalog);

    if (string.IsNullOrEmpty(cachedCatalog))
    {
        var allCatalog = await dbContext.Catalog.Where(i => i.Vip == isVip).ToListAsync();
        // Serializza nuovamente per salvarlo in Redis
        cachedCatalog = JsonSerializer.Serialize(allCatalog);

        // Salva i dati in cache per 2 minuti
        await db.StringSetAsync(keyForCatalog, cachedCatalog, TimeSpan.FromMinutes(2));
    }

    // Deserializza i dati dalla cache Redis
    var catalogo = JsonSerializer.Deserialize<List<Catalog>>(cachedCatalog);

    // Restituisce i dati del catalogo come risposta JSON
    return Results.Json(catalogo);
});
```

Package

```
10 <ItemGroup>
11   <PackageReference Include="EntityFramework" Version="6.5.1" />
12   <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="8.0.8" />
13   <PackageReference Include="Microsoft.AspNetCore.Authorization" Version="8.0.8" />
14   <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="8.0.0" />
15   <PackageReference Include="Microsoft.EntityFrameworkCore" Version="8.0.8" />
16   <PackageReference Include="Microsoft.Extensions.DependencyInjection.Abstractions" Version="8.0.1" />
17   <PackageReference Include="Microsoft.IdentityModel.Tokens" Version="8.0.2" />
18   <PackageReference Include="MySql.Data" Version="9.0.0" />
19   <PackageReference Include="Pomelo.EntityFrameworkCore.MySql" Version="8.0.2" />
20   <PackageReference Include="StackExchange.Redis" Version="2.8.0" />
21   <PackageReference Include="Swashbuckle.AspNetCore" Version="6.7.3" />
22   <PackageReference Include="System.Text.Json" Version="8.0.4" />
23 </ItemGroup>
24
25 </Project>
```

Node.js

```
npm init -y  
npm install express mysql2 redis dotenv  
node index.js
```

Index.js

... provateci voi !!

Creiamo il –env

```
PORT=3000  
MYSQL_HOST=127.0.0.1  
MYSQL_USER=root  
MYSQL_PASSWORD=ifts  
MYSQL_DATABASE=iftsdb  
REDIS_HOST=127.0.0.1  
REDIS_PORT=6379
```



GO?

... provateci voi !!

Microservizi: Metodologie e sistemi di autenticazione

- Cos'è l'autenticazione
- Come funziona l'autenticazione
- Tipi di autenticazione
 - Con certificato, con token, One-time password, a più fattori
- Protocollo di autenticazione
 - Open ID Connect OIDC
- MFA Multi-Factor Authentication

Panoramica del corso

Architetture Serverless e Microservizi

Microservizi: Metodologie e sistemi di caching

- Strategie di caching dei dati
- Architettura dei sistemi di cache in ambiente Kubernetes
- Sistemi Open Source
 - Redis
 - Infinispan
- Cache in cloud - sistemi CDN
 - Google Cloud
 - AWS - Amazon CloudFront
 - Akamai

Microservizi: Metodologie e sistemi di comunicazione asincrone

- Panoramica sulla comunicazione asincrona tra microservizi basata su messaggi
- Event-driven architecture
 - Perché è fondamentale in architetture a microservizi
- Communication e Consumption Pattern
- Sistemi di message broker per microservizi
 - RabbitMQ
 - Apache Kafka
 - Redis : un uso alternativo
- Casi d'uso e scenari di applicazione



Grazie

WWW.MICROGAME.IT



Roma

Viale Luca Gaurico, 9/11 IV piano scala A
00143 - Roma - Italia



Benevento

Via Giovanni Agnelli, People's House
Zona Industriale Contrada Olivola - Lotto D/4
82100 - Benevento