پروژه پایگاه داده پیشرفته: سامانه پایش شبکه با Apache Cassandra

مقدمه و اهداف پروژه

هدف این پروژه، طراحی و پیاده‌سازی یک "سامانه پایش شبکه (Network Monitoring System) " مقیاس‌پذیر با استفاده از پایگاه داده Apache Cassandra بود .سیستم می‌بایست قادر به ذخیره‌سازی، بازیابی و تحلیل داده‌های سری زمانی حجیم باشد که به صورت پیوسته (هر ۵ دقیقه) از صدها منبع شبکه (مانند روترها و فایروال‌ها) جمع‌آوری می‌شوند.

چالش اصلی

پاسخ‌گویی به دو کوئری کلیدی زیر با کارایی بالا (High Performance) بود:

- **Q1:** نمایش توالی مقادیر یک یا چند متریک خاص برای یک منبع، در بازه زمانی کوتاه (کمتر از یک هفته).
- **Q2:** استخراج لیست تمام متریک‌های مربوط به یک منبع خاص.

ابزارهای مورد استفاده

ابزار	کاربرد
Apache Cassandra	پایگاه داده NoSQL مقیاس‌پذیر برای ذخیره‌سازی داده‌های سری زمانی.
CQL (Cassandra Query Language)	زبان تعریف جداول و اجرای کوئری‌ها.
Python	زبان اسکریپت‌نویسی برای فرآیند ETL (تولید، تبدیل و بارگذاری داده‌ها).
Docker	برای اجرای سریع و ایزوله دیتابیس Cassandra در محیط توسعه.

مستندات طراحی و منطق انتخاب مدل داده

کلید موفقیت درCassandra ، استفاده از طراحی مبتنی بر کوئری (Query-Driven Design) است. بر خلاف مدل‌سازیSQL ، ما ابتدا کوئری‌های نهایی Q1 و Q2 را شناسایی کرده و سپس جداولی را طراحی می‌کنیم که مستقیماً به آن کوئری‌ها پاسخ دهند.

هدف اصلی این است که هر کوئری، پاسخ خود را با خواندن حداکثر یک پارتیشن دریافت کند .برای رسیدن به این هدف، از افزونگی داده (Data Redundancy) استفاده کرده و دو جدول مجزا طراحی نمودیم.

جدول1: metrics_by_resource_week برای پاسخ بهQ1

این جدول برای پاسخ به کوئری تحلیل سری زمانی (Q1) طراحی شده است.

طراحی :کوئری Q1 بازه کمتر از یک هفته را می‌خواهد ، اما فرضیات پروژه می‌گوید داده‌ها برای یک سال نگهداری خواهند شد .اگر ما تمام داده‌های یک ساله یک متریک را در یک پارتیشن ذخیره می‌کردیم، آن پارتیشن بسیار حجیم شده و خواندن آن ناکارآمد بود.

راه‌حل :(Time Bucketing) ما داده‌ها را در "سطل‌های زمانی (Time Buckets) "هفتگی دسته‌بندی کردیم.

```
PowerShell
PS C:\Users\samin\OneDrive\Desktop> docker run -d --name cassandra-project -p 9042:9042 cassandra:latest
Unable to find image 'cassandra:latest' locally
latest: Pulling from library/cassandra
Digest: sha256:8b55dd41d5d1220e11eb8cf80f26ab655c21f7cf271ca4a7577c1da7d9221624
Status: Downloaded newer image for cassandra:latest
00510531935ff671e64609298c18158f89ea045aa4f769213724ed54035247a0
PS C:\Users\samin\OneDrive\Desktop> docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS         PORTS
                        NAMES
00510531935f   cassandra:latest   "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes   7000-7001/tcp, 7199/tcp, 9160/tcp,
0.0.0.0:9042->9042/tcp   cassandra-project
PS C:\Users\samin\OneDrive\Desktop> docker exec -it cassandra-project cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.5 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh> CREATE KEYSPACE IF NOT EXISTS network_monitoring
   ... WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
cqlsh> USE network_monitoring;
cqlsh:network_monitoring> CREATE TABLE metrics_by_resource_week (
                      ...     resource TEXT,
                      ...     year INT,
                      ...     week_of_year INT,
                      ...     metric_name TEXT,
                      ...     collected_at TIMESTAMP,
                      ...     value DOUBLE,
                      ...     PRIMARY KEY ((resource, year, week_of_year), metric_name, collected_at)
                      ... ) WITH CLUSTERING ORDER BY (metric_name ASC, collected_at DESC);
```

منطق انتخاب کلیدها:

- Partition Key: (resource, year, week_of_year)

  - این کلید تضمین می‌کند که *تمام متریک‌ها* (کمتر از ۱۰) تا ، برای یک *منبع* خاص، در یک *هفته* خاص، همگی در یک پارتیشن واحد ذخیره می‌شوند.
  - بنابراین، وقتی کوئری Q1 مثلاً داده‌های هفته ۵۲ سال ۲۰۲۲ برای (router-1) را اجرا می‌کنیم، Cassandra مستقیماً به همان یک پارتیشن مراجعه می‌کند.

- Clustering Key: (metric_name ASC, collected_at DESC)

  - metric_name: به ما اجازه می‌دهد تا به سرعت یک یا چند متریک را فیلتر کنیم.
  - collected_at DESC: داده‌های درون پارتیشن را بر اساس زمان (جدیدترین اول) مرتب می‌کند که برای تحلیل سری زمانی ایده‌آل است.

جدول ۲: metrics_by_resource_list برای پاسخ بهQ2

این جدول صرفاً برای پاسخ سریع به کوئری Q2 (لیست متریک‌های یک منبع) طراحی شده است.

```
cqlsh:network_monitoring> CREATE TABLE metrics_by_resource_list (
                     ...     resource TEXT,
                     ...     metric_name TEXT,
                     ...     PRIMARY KEY (resource, metric_name)
                     ... );
cqlsh:network_monitoring>
```

منطق انتخاب کلیدها:

- **Partition Key:** (resource)
  - تمام متریک‌های مربوط به یک منبع در یک پارتیشن ذخیره می‌شوند.
- **Clustering Key:** (metric_name)
  - این کلید، لیست متریک‌ها را برای آن منبع مرتب و (مهم‌تر) منحصربه‌فرد می‌کند.

اسکریپت Python برای ورود داده(ETL)

یادداشتی در مورد بهینه‌سازی فرآیندETL :

در مستندات پروژه، از ما خواسته شده بود که ابتدا داده‌ها را در یک فایل CSV ایجاد کرده و سپس آن فایل را با پایتون بخوانیم و در Cassandra درج کنیم.

اولین قدم، تحلیل این نیازمندی بود. بر اساس فرضیات خود پروژه (۲۰۰ منبع، هر ۵ دقیقه، برای ۱ سال)، محاسبات نشان داد که ایجاد این فایل CSV منجر به تولید فایلی با بیش از **۱۰۰ میلیون ردیف** داده می‌شد.

ایجاد یک فایل متنی با این حجم (چندین گیگابایت) و سپس خواندن مجدد کل آن از روی دیسک، یک تنگنای (bottleneck) شدید I/O ایجاد کرده و فرآیندی بسیار ناکارآمد و زمان‌بر است.

در نتیجه، برای بهینه‌سازی این فرآیند، تصمیم گرفتیم که به جای انجام دو فرآیند مجزای نوشتن روی دیسک و خواندن از دیسک، ما اسکریپت generate_and_ingest.py را توسعه دادیم.

این اسکریپت هر سه وظیفه ETL (تولید، تبدیل، و بارگذاری) را به صورت یکپارچه و در حافظه انجام می‌دهد:

۱. Generate (تولید): داده‌ها بر اساس فرضیات پروژه در لحظه تولید می‌شوند.

۲. Transform (تبدیل): ستون‌های year و week_of_year در همان لحظه استخراج می‌شوند.

۳. Load (بارگذاری): داده‌ها مستقیماً و به صورت بهینه (با بچ‌های کوچک) در Cassandra درج می‌شوند.

این روش، *منطق* خواسته شده در پروژه ETL با پایتون را در مقیاس بالا به شکلی کارآمد شبیه‌سازی می‌کند.

فایل کامل این اسکریپت با نام generate_and_ingest.py به صورت جداگانه پیوست شده است.

```
 ... 3790000 records inserted. (Last timestamp: 2023-01-07 13:50:00)
 ... 3800000 records inserted. (Last timestamp: 2023-01-07 14:15:00)
 ... 3810000 records inserted. (Last timestamp: 2023-01-07 14:40:00)
 ... 3820000 records inserted. (Last timestamp: 2023-01-07 15:05:00)
 ... 3830000 records inserted. (Last timestamp: 2023-01-07 15:30:00)
 ... 3840000 records inserted. (Last timestamp: 2023-01-07 15:55:00)
 ... 3850000 records inserted. (Last timestamp: 2023-01-07 16:20:00)
 ... 3860000 records inserted. (Last timestamp: 2023-01-07 16:45:00)
 ... 3870000 records inserted. (Last timestamp: 2023-01-07 17:10:00)
 ... 3880000 records inserted. (Last timestamp: 2023-01-07 17:35:00)
 ... 3890000 records inserted. (Last timestamp: 2023-01-07 18:00:00)
 ... 3900000 records inserted. (Last timestamp: 2023-01-07 18:25:00)
 ... 3910000 records inserted. (Last timestamp: 2023-01-07 18:50:00)
 ... 3920000 records inserted. (Last timestamp: 2023-01-07 19:15:00)
 ... 3930000 records inserted. (Last timestamp: 2023-01-07 19:40:00)
 ... 3940000 records inserted. (Last timestamp: 2023-01-07 20:05:00)
 ... 3950000 records inserted. (Last timestamp: 2023-01-07 20:30:00)
 ... 3960000 records inserted. (Last timestamp: 2023-01-07 20:55:00)
 ... 3970000 records inserted. (Last timestamp: 2023-01-07 21:20:00)
 ... 3980000 records inserted. (Last timestamp: 2023-01-07 21:45:00)
 ... 3990000 records inserted. (Last timestamp: 2023-01-07 22:10:00)
 ... 4000000 records inserted. (Last timestamp: 2023-01-07 22:35:00)
 ... 4010000 records inserted. (Last timestamp: 2023-01-07 23:00:00)
 ... 4020000 records inserted. (Last timestamp: 2023-01-07 23:25:00)
 ... 4030000 records inserted. (Last timestamp: 2023-01-07 23:50:00)
--- Data Ingestion Complete ---
Total records inserted: 4032000
Total execution time: 6223.44 seconds
```

اجرای کوئری‌ها و تحلیل نتایج

پس از درج موفقیت‌آمیز بیش از ۴ میلیون رکورد داده، کوئری‌های کلیدی پروژه برای اعتبارسنجی مدل اجرا شدند.

الف) پاسخ به Q1: تحلیل سری زمانی (یک متریک)

```
PowerShell
(0 rows)
cqlsh:network_monitoring> SELECT collected_at, value
                        ...
                        ... FROM metrics_by_resource_week
                        ...
                        ... WHERE resource = 'router-1'
                        ...
                        ... AND year = 2022
                        ...
                        ... AND week_of_year = 52
                        ...
                        ... AND metric_name = 'cpu_usage'
                        ...
                        ... AND collected_at >= '2023-01-01 00:00:00'
                        ...
                        ... AND collected_at <= '2023-01-01 02:00:00';

 collected_at                    | value
---------------------------------+------------
 2023-01-01 02:00:00.000000+0000 | 4678.67137
 2023-01-01 01:55:00.000000+0000 | 4905.97398
 2023-01-01 01:50:00.000000+0000 | 2808.63844
 2023-01-01 01:45:00.000000+0000 | 4781.48266
```

```
PowerShell
                        ...
                        ... AND collected_at >= '2023-01-01 00:00:00'
                        ...
                        ... AND collected_at <= '2023-01-01 02:00:00';

 collected_at                    | value
---------------------------------+------------
 2023-01-01 02:00:00.000000+0000 | 4678.67137
 2023-01-01 01:55:00.000000+0000 | 4905.97398
 2023-01-01 01:50:00.000000+0000 | 2808.63844
 2023-01-01 01:45:00.000000+0000 | 4781.48266
 2023-01-01 01:40:00.000000+0000 |  877.60898
 2023-01-01 01:35:00.000000+0000 | 2676.60084
 2023-01-01 01:30:00.000000+0000 | 3300.24776
 2023-01-01 01:25:00.000000+0000 | 3509.37297
 2023-01-01 01:20:00.000000+0000 | 2637.40627
 2023-01-01 01:15:00.000000+0000 | 3218.97313
 2023-01-01 01:10:00.000000+0000 | 2807.81035
 2023-01-01 01:05:00.000000+0000 | 2618.41714
 2023-01-01 01:00:00.000000+0000 | 4540.42572
 2023-01-01 00:55:00.000000+0000 |  2052.0197
 2023-01-01 00:50:00.000000+0000 |  1530.3178
 2023-01-01 00:45:00.000000+0000 | 1315.00213
 2023-01-01 00:40:00.000000+0000 |  991.77634
 2023-01-01 00:35:00.000000+0000 | 4338.71232
 2023-01-01 00:30:00.000000+0000 | 3759.49822
 2023-01-01 00:25:00.000000+0000 |  1219.6477
 2023-01-01 00:20:00.000000+0000 | 2265.96563
 2023-01-01 00:15:00.000000+0000 | 4209.76449
 2023-01-01 00:10:00.000000+0000 | 2967.81817
 2023-01-01 00:05:00.000000+0000 |  1216.4635
 2023-01-01 00:00:00.000000+0000 | 1719.52087

(25 rows)
cqlsh:network_monitoring>
```

نتیجه : کوئری با موفقیت اجرا شد و ۲۵ ردیف داده را برگرداند. این نشان می‌دهد که مدل "Time Bucketing" به درستی کار می‌کند. (دلیل استفاده از year=2022 وweek=52 برای تاریخ 2023-01-01، پیروی اسکریپت پایتون از استاندارد تقویم ISO است.

ب)پاسخ به :Q1 تحلیل سری زمانی (چند متریک)

```
PowerShell

(25 rows)
cqlsh:network_monitoring> SELECT metric_name, collected_at, value
                        ... FROM metrics_by_resource_week
                        ... WHERE resource = 'router-1'
                        ... AND year = 2022
                        ... AND week_of_year = 52
                        ... AND metric_name IN ('cpu_usage', 'memory_usage')
                        ... AND collected_at >= '2023-01-01 00:00:00'
                        ... AND collected_at <= '2023-01-01 02:00:00';

 metric_name  | collected_at                    | value
--------------+---------------------------------+------------
    cpu_usage | 2023-01-01 02:00:00.000000+0000 | 4678.67137
    cpu_usage | 2023-01-01 01:55:00.000000+0000 | 4905.97398
    cpu_usage | 2023-01-01 01:50:00.000000+0000 | 2808.63844
    cpu_usage | 2023-01-01 01:45:00.000000+0000 | 4781.48266
    cpu_usage | 2023-01-01 01:40:00.000000+0000 |  877.60898
    cpu_usage | 2023-01-01 01:35:00.000000+0000 | 2676.60084
    cpu_usage | 2023-01-01 01:30:00.000000+0000 | 3300.24776
    cpu_usage | 2023-01-01 01:25:00.000000+0000 | 3509.37297
    cpu_usage | 2023-01-01 01:20:00.000000+0000 | 2637.40627
    cpu_usage | 2023-01-01 01:15:00.000000+0000 | 3218.97313
```

```
PowerShell
    cpu_usage | 2023-01-01 01:00:00.000000+0000 | 4540.42572
    cpu_usage | 2023-01-01 00:55:00.000000+0000 |  2052.0197
    cpu_usage | 2023-01-01 00:50:00.000000+0000 |  1530.3178
    cpu_usage | 2023-01-01 00:45:00.000000+0000 | 1315.00213
    cpu_usage | 2023-01-01 00:40:00.000000+0000 |  991.77634
    cpu_usage | 2023-01-01 00:35:00.000000+0000 | 4338.71232
    cpu_usage | 2023-01-01 00:30:00.000000+0000 | 3759.49822
    cpu_usage | 2023-01-01 00:25:00.000000+0000 |  1219.6477
    cpu_usage | 2023-01-01 00:20:00.000000+0000 | 2265.96563
    cpu_usage | 2023-01-01 00:15:00.000000+0000 | 4209.76449
    cpu_usage | 2023-01-01 00:10:00.000000+0000 | 2967.81817
    cpu_usage | 2023-01-01 00:05:00.000000+0000 |  1216.4635
    cpu_usage | 2023-01-01 00:00:00.000000+0000 | 1719.52087
 memory_usage | 2023-01-01 02:00:00.000000+0000 | 2865.54714
 memory_usage | 2023-01-01 01:55:00.000000+0000 | 2106.00984
 memory_usage | 2023-01-01 01:50:00.000000+0000 | 1694.73602
 memory_usage | 2023-01-01 01:45:00.000000+0000 | 4221.20263
 memory_usage | 2023-01-01 01:40:00.000000+0000 |  511.64343
 memory_usage | 2023-01-01 01:35:00.000000+0000 |  2533.9883
 memory_usage | 2023-01-01 01:30:00.000000+0000 | 1359.51903
 memory_usage | 2023-01-01 01:25:00.000000+0000 | 3372.35254
 memory_usage | 2023-01-01 01:20:00.000000+0000 | 2955.63363
 memory_usage | 2023-01-01 01:15:00.000000+0000 | 2377.04244
 memory_usage | 2023-01-01 01:10:00.000000+0000 |  4563.6209
 memory_usage | 2023-01-01 01:05:00.000000+0000 | 4452.69456
 memory_usage | 2023-01-01 01:00:00.000000+0000 | 3101.87711
 memory_usage | 2023-01-01 00:55:00.000000+0000 | 4669.98975
 memory_usage | 2023-01-01 00:50:00.000000+0000 | 1976.31271
 memory_usage | 2023-01-01 00:45:00.000000+0000 | 1242.37952
 memory_usage | 2023-01-01 00:40:00.000000+0000 | 3709.20267
 memory_usage | 2023-01-01 00:35:00.000000+0000 | 3670.35352
 memory_usage | 2023-01-01 00:30:00.000000+0000 | 3710.63166
 memory_usage | 2023-01-01 00:25:00.000000+0000 | 2410.13365
 memory_usage | 2023-01-01 00:20:00.000000+0000 | 3965.03631
 memory_usage | 2023-01-01 00:15:00.000000+0000 | 4327.43359
 memory_usage | 2023-01-01 00:10:00.000000+0000 | 4898.49524
 memory_usage | 2023-01-01 00:05:00.000000+0000 |  705.83334
 memory_usage | 2023-01-01 00:00:00.000000+0000 | 3467.73945

(50 rows)
cqlsh:network_monitoring> |
```

نتیجه :کوئری با موفقیت اجرا شد و ۵۰ ردیف داده (۲۵ ردیف برای هر متریک) را برگرداند. این ثابت می‌کند که مدل ما قادر است "یک یا چند متریک" را به صورت کارآمد و تنها با خواندن یک پارتیشن واکشی کند.

پاسخ به :Q2 استخراج لیست متریک‌ها

```
PowerShell
cqlsh:network_monitoring> SELECT metric_name
                        ... FROM metrics_by_resource_list
                        ... WHERE resource = 'router-1';

 metric_name
--------------
    cpu_usage
   latency_ms
 memory_usage
   packets_in
  packets_out

(5 rows)
cqlsh:network_monitoring> SELECT * FROM metrics_by_resource_list LIMIT 10;

 resource    | metric_name
-------------+--------------------
 firewall-31 | active_connections
 firewall-31 |           bytes_in
 firewall-31 |          bytes_out
 firewall-31 |          cpu_usage
 firewall-31 |    dropped_packets
  switch-104 |          cpu_usage
  switch-104 |      packets_total
  switch-104 |      port_1_errors
  switch-104 |      port_2_errors
  switch-104 |         uptime_sec

(10 rows)
cqlsh:network_monitoring>
```

نتیجه :کوئری بلافاصله اجرا شد و لیست ۵ متریک مربوط به router-1 را برگرداند. این ثابت می‌کند که جدول دوم (جدول افزونه) به درستی به کوئری Q2 پاسخ می‌دهد.

نتیجه‌گیری

این پروژه با موفقیت تمام اهداف خود را محقق کرد. با استفاده از اصول Query-Driven Design و Time Bucketing در Apache Cassandra ، یک مدل داده بسیار کارآمد طراحی شد که می‌تواند میلیون‌ها رکورد داده سری زمانی را مدیریت کند و در عین حال، کوئری‌های تحلیلی کلیدی را با کارایی بالا (خواندن از یک پارتیشن) پاسخ دهد.