

Chmurowy system analizy i monitoringu MongoDB

Jan Cielesz 36343. Infrastruktura Azure. Monitoring. Przygotowanie danych.

Marcin Grabski 42033 Testy. Analiza danych. Projekt infrastruktury.

Streszczenie

W niniejszym dokumencie przedstawiono proces projektowania, wdrożenia oraz analizy systemu do monitorowania wydajności bazy danych MongoDB z wykorzystaniem narzędzi Prometheus, MongoDB Exporter i Grafana. Omówiono kluczowe etapy, w tym konfigurację infrastruktury monitorującej, generowanie obciążeń testowych oraz zbieranie i wizualizację danych o wydajności. Testy obejmowały różne scenariusze operacyjne, takie jak intensywne operacje odczytu, zapisu oraz ich mieszane kombinacje, co pozwoliło na identyfikację istotnych różnic w latencjach między różnymi typami operacji i kolekcjami. Przeprowadzono analizy statystyczne, takie jak test t-Studenta i analiza wariancji (ANOVA), w celu oceny istotności różnic miar wydajności. Wyniki wykazały, że średnia latencja zapisu była znacząco wyższa niż odczytu w większości testowanych scenariuszy, a różnice między kolekcjami były istotne statystycznie. Na podstawie wyników zaproponowano potencjalne optymalizacje, takie jak poprawa indeksowania czy zmiana struktury danych. Dokument kończy się rekomendacjami dotyczącymi dalszych badań, szczególnie w kontekście skalowania systemu i wdrażania rozwiązań chmurowych.

Słowa kluczowe: MongoDB, wydajność, Prometheus, Grafana, testy obciążeniowe, analiza statystyczna, optymalizacja.

Cel projektu

Celem projektu jest zaprojektowanie, wdrożenie i przeprowadzenie analizy wydajności systemu opartego na bazie danych MongoDB, z wykorzystaniem narzędzi do monitorowania i wizualizacji danych, takich jak Prometheus, MongoDB Exporter oraz Grafana. Projekt zakłada zbadanie zachowania bazy danych w różnych scenariuszach obciążenia, obejmujących intensywne operacje odczytu, zapisu oraz ich kombinacje. Kluczowym aspektem jest identyfikacja istotnych różnic w wydajności między różnymi typami operacji i strukturami danych w kolekcjach. Analiza wydajności zostanie przeprowadzona za pomocą zebranych metryk i metod statystycznych, takich jak test t-Studenta oraz analiza wariancji (ANOVA), co pozwoli ocenić statystyczną istotność różnic w miarach wydajności.

Wymagania systemu informacyjnego

Zakres systemu:

- System monitoruje wydajność bazy danych MongoDB, gromadząc metryki takie jak latencja odczytu i zapisu, liczba operacji na sekundę oraz zużycie zasobów systemowych.
- Dane metryczne są zbierane przez MongoDB Exporter i przechowywane w Prometheus.
- Wizualizacja danych odbywa się w Grafana, z wykorzystaniem predefiniowanych dashboardów i dostosowanych wykresów.
- Testy obciążeniowe są przeprowadzane za pomocą narzędzi takich jak Python (pymongo) oraz Locust, symulując różne scenariusze użytkowania.
- Analiza zebranych danych odbywa się w Pythonie z wykorzystaniem bibliotek Pandas, NumPy, Matplotlib i SciPy.

Ograniczenia systemu:

- System działa w środowisku chmurowym i wymaga stabilnego połączenia z internetem.

- Monitorowanie metryk zależy od poprawnej konfiguracji Prometheus i MongoDB Exporter.
- System wymaga minimalnych zasobów serwerowych: 4 vCPU i 8 GB RAM dla MongoDB oraz 2 vCPU i 4 GB RAM dla monitorowania.

Użyte systemy i narzędzia

- Infrastruktura jako kod: Terraform, Ansible.
Terraform został użyty dla szybkiego stworzenia infrastruktury w chmurze Azure.
Ansible ustawia utworzoną maszynę wirtualną do stanu “Ready to go”.
- Monitoring i metryki: Prometheus, MongoDB exporter, Grafana.
Prometheus do zbierania metryk. Grafana wizualizuje je. MongoDB exporter eksportuje metryki z MongoDB do Prometheus’a.
- Google Colab do statycznej analizy danych.

Zbiory danych:

- Sample Analytics
Przykładowe dane analityczne. Największa kolekcja z transakcjami zajmuje około 20mb.
- Sample Supplies
Przykładowe dane z zaopatrzeniami. Zawiera jedną kolekcję Sales (5mb)

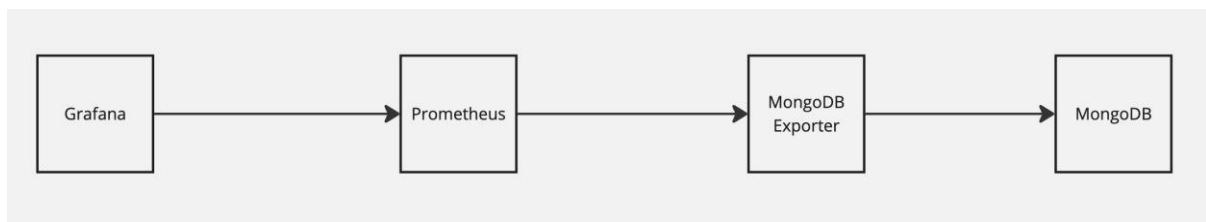
Opis stanowiska do analiz

Architektura systemu:

- Chmura Azure:
 - Resource group (odzd-project-resource-group)
 - Virtual network (odzd-project-vnet)

- Subnet (odzd-project-subnet)
- Network interface (odzd-project-nic)
- Network security group (odzd-project-network-security-group)
- Public ip
- Maszyna wirtualna
- Maszyna wirtualna:
 - MongoDB
 - Prometheus
 - Grafana
 - MongoDB exporter

Uproszczony diagram systemu monitorującego:



Jak to działa:

Expoter zbiera metryki z bazy danych i udostępnia ich na porcie 9216. Prometheus raz na 15 sekund zbiera tę metryki i dodaje do nich czas kiedy to zrobił. Dalej Grafana odpytuje o metryki Prometheus za jakiś określony okres np. ostatnie 30 min.

Instrukcja użytkowania systemu

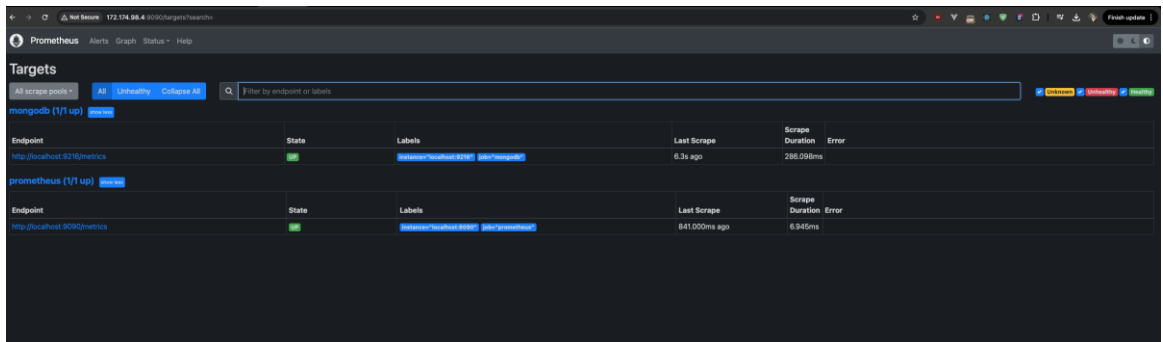
- Wdrożenie infrastruktury na Azure.

\$ terraform plan

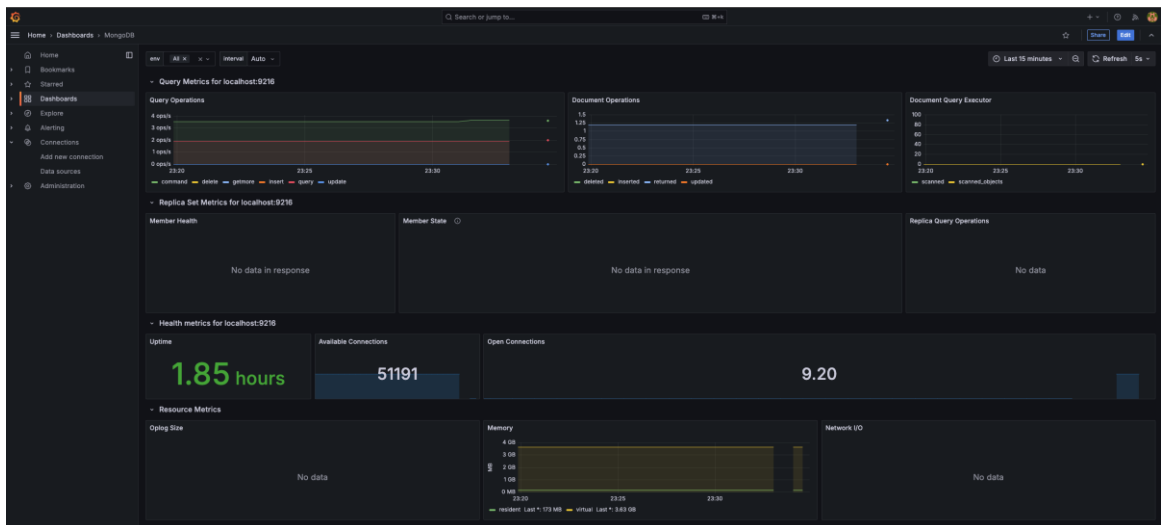
\$ terraform apply

Po ukończeniu wdrożenia mamy działający Prometheus, MongoDB z danymi, Grafana i eksporterem. Do grafany należy zaimportować dashboard grafana_dashboard.json. Plik jest w repozytorium.

Prometheus:



Grafana:



Exporter:


```
# HELP collector_scrape_time_ms Time taken for scrape by collector
# TYPE collector_scrape_time_ms gauge
collector_scrape_time_ms{collector="collstats",exporter="mongodb"} 269
collector_scrape_time_ms{collector="currentop",exporter="mongodb"} 4
collector_scrape_time_ms{collector="dbstats",exporter="mongodb"} 4
collector_scrape_time_ms{collector="diagnosticdata",exporter="mongodb"} 36
collector_scrape_time_ms{collector="fcv",exporter="mongodb"} 0
collector_scrape_time_ms{collector="general",exporter="mongodb"} 1
collector_scrape_time_ms{collector="indexstats",exporter="mongodb"} 6
collector_scrape_time_ms{collector="ism",exporter="mongodb"} 3
collector_scrape_time_ms{collector="profile",exporter="mongodb"} 8
collector_scrape_time_ms{collector="replset_config",exporter="mongodb"} 0
collector_scrape_time_ms{collector="replset_status",exporter="mongodb"} 0
collector_scrape_time_ms{collector="top",exporter="mongodb"} 4
# HELP pg_gc_duration_seconds A summary of the wall-time pause (stop-the-world) duration in garbage collection cycles.
# TYPE pg_gc_duration_seconds summary
pg_gc_duration_seconds{quantile="0"} 3.2481e-05
pg_gc_duration_seconds{quantile="0.25"} 4.7881e-05
pg_gc_duration_seconds{quantile="0.5"} 5.3181e-05
pg_gc_duration_seconds{quantile="0.75"} 5.74e-05
pg_gc_duration_seconds{quantile="1"} 0.008382284
pg_gc_duration_seconds_sum 0.003353087
pg_gc_duration_seconds_count 336
# HELP pg_gc_percent Gauge heap size target percentage configured by the user, otherwise 100. This value is set by the GOGC environment variable, and the runtime/debug.SetGCPercent function. Sourced from /gc/gc:percent
# TYPE pg_gc_percent gauge
pg_gc_percent 100
# HELP pg_gomemlimit_bytes Go runtime memory limit configured by the user, otherwise math.MaxInt64. This value is set by the GOMEMLIMIT environment variable, and the runtime/debug.SetMemoryLimit function. Sourced from /gc/gomemlimit:bytes
# TYPE pg_gomemlimit_bytes gauge
pg_gomemlimit_bytes 3.22337838554776e+18
# HELP pg_goroutines Number of goroutines that currently exist.
# TYPE pg_goroutines gauge
pg_goroutines 28
# HELP pg_info Information about the Go environment.
# TYPE pg_info gauge
pg_info{version="go1.22.4"} 1
# HELP pg_memstats_alloc_bytes Number of bytes allocated in heap and currently in use. Equals to /memory/classes/heap/objects:bytes.
# TYPE pg_memstats_alloc_bytes gauge
pg_memstats_alloc_bytes 1.189284e+9
# HELP pg_memstats_alloc_bytes_total Total number of bytes allocated in heap until now, even if released already. Equals to /gc/heap/allocs:bytes.
# TYPE pg_memstats_alloc_bytes_total counter
pg_memstats_alloc_bytes_total 2.865795552e+9
# HELP pg_memstats_alloc_bytes_total_counter Total number of bytes used by the profiling bucket hash table. Equals to /memory/classes/profiling/buckets:bytes.
# TYPE pg_memstats_alloc_bytes_total_counter gauge
pg_memstats_alloc_bytes_total_counter 1.571838e+9
# HELP pg_memstats_buck_hash_sys_bytes Number of heap bytes allocated and currently in use, same as pg_memstats_alloc_bytes. Equals to /memory/classes/heap/objects:bytes.
# TYPE pg_memstats_buck_hash_sys_bytes gauge
pg_memstats_buck_hash_sys_bytes 1.08748e+9
# HELP pg_memstats_heap_alloc_bytes Number of heap bytes allocated and currently in use, same as pg_memstats_alloc_bytes. Equals to /memory/classes/heap/objects:bytes.
# TYPE pg_memstats_heap_alloc_bytes gauge
pg_memstats_heap_alloc_bytes 1.08748e+9
# HELP pg_memstats_heap_idle_bytes Number of heap bytes waiting to be used. Equals to /memory/classes/heap/released:bytes + /memory/classes/heap/free:bytes.
# TYPE pg_memstats_heap_idle_bytes gauge
pg_memstats_heap_idle_bytes 1.571838e+9
# HELP pg_memstats_heap_inuse_bytes Number of heap bytes that are in use. Equals to /memory/classes/heap/objects:bytes + /memory/classes/heap/unused:bytes
# TYPE pg_memstats_heap_inuse_bytes gauge
pg_memstats_heap_inuse_bytes 2.1815296e+9
# HELP pg_memstats_heap_objects Number of currently allocated objects. Equals to /gc/heap/objects:objects.
# TYPE pg_memstats_heap_objects gauge
pg_memstats_heap_objects 32223
# HELP pg_memstats_heap_released_bytes Number of heap bytes released to OS. Equals to /memory/classes/heap/released:bytes.
# TYPE pg_memstats_heap_released_bytes gauge
pg_memstats_heap_released_bytes 1.5382556e+9
# HELP pg_memstats_heap_sys_bytes Number of heap bytes obtained from system. Equals to /memory/classes/heap/objects:bytes + /memory/classes/heap/unused:bytes + /memory/classes/heap/released:bytes + /memory/classes/heap/free:bytes.
# TYPE pg_memstats_heap_sys_bytes gauge
pg_memstats_heap_sys_bytes 3.719168e+9
# HELP pg_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage collection.
# TYPE pg_memstats_last_gc_time_seconds gauge
pg_memstats_last_gc_time_seconds 1.7276641582999e+9
# HELP pg_memstats_malloc_total Number of heap objects allocated, both live and gc-ed. Semantically a counter version for pg_memstats_heap_objects gauge. Equals to /gc/heap/allocs:objects + /gc/heap/tiny/allocs:objects.
# TYPE pg_memstats_malloc_total counter
```

- Uruchomienie testów:
Uruchomić plik `perfomance_test.py`. Po testach zostanie wygenerowany plik z pobranymi metrykami z Prometheus. To przyda się do dalszej analizy.
- Jupiter notebook do analizy

```
[1] import pandas as pd
import matplotlib.pyplot as plt

# Wczytaj dane z pliku CSV
data = pd.read_csv("sample_analytics_collected_metrics.csv")


# Wyświetl podstawowe informacje o danych
print(data.head())
print(data.info())
```

 Pokaż ukryte dane wyjściowe

```
# Filtracja danych dla różnych metryk
read_latency = data[data['metric'] == 'mongodb_collstats_latencyStats_reads_latency']
write_latency = data[data['metric'] == 'mongodb_collstats_latencyStats_writes_latency']

# Przekształcenie kolumny timestamp na obiekt datetime
read_latency['timestamp'] = pd.to_datetime(read_latency['timestamp'])
write_latency['timestamp'] = pd.to_datetime(write_latency['timestamp'])
print("Read Latency Stats:")
print(read_latency['value'].describe())

print("Write Latency Stats:")
print(write_latency['value'].describe())
```

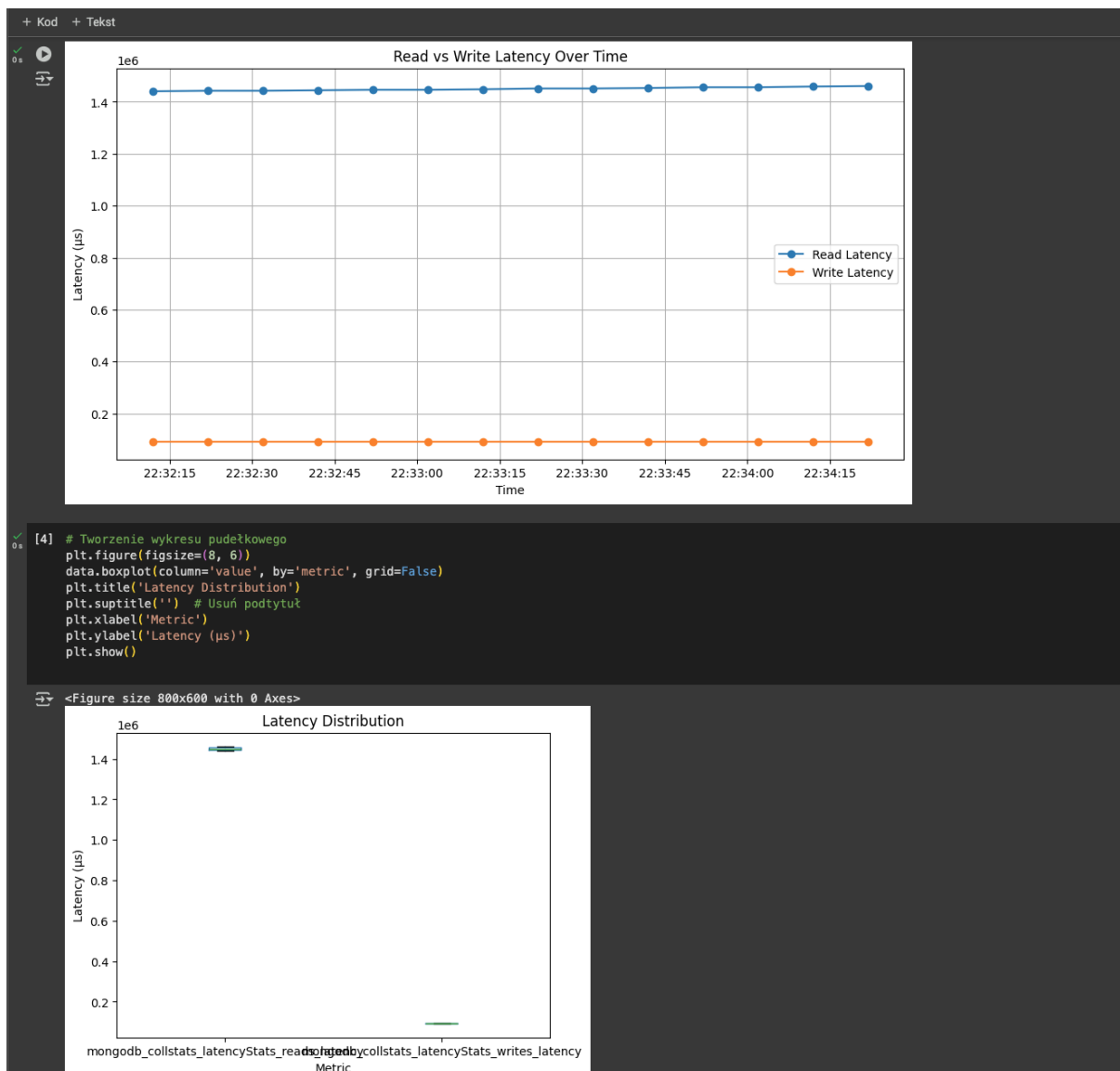
 Pokaż ukryte dane wyjściowe

```
[3] plt.figure(figsize=(12, 6))

# Wykres latencji odczytu
plt.plot(read_latency['timestamp'], read_latency['value'], label='Read Latency', marker='o')

# Wykres latencji zapisu
plt.plot(write_latency['timestamp'], write_latency['value'], label='Write Latency', marker='o')

plt.title('Read vs Write Latency Over Time')
plt.xlabel('Time')
plt.ylabel('Latency (µs)')
plt.legend()
plt.grid()
plt.show()
```

```
[5] from scipy.stats import ttest_ind

# Test t-Studenta
t_stat, p_value = ttest_ind(read_latency['value'], write_latency['value'])
print(f"T-statistic: {t_stat}, P-value: {p_value}")

if p_value < 0.05:
    print("Istnieją istotne różnice między odczytem a zapisem.")
else:
    print("Nie znaleziono istotnych różnic między odczytem a zapisem.")
```

T-statistic: 786.0283336240246, P-value: 2.010526040959604e-58
Istnieją istotne różnice między odczytem a zapisem.
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:531: RuntimeWarning: Precision loss occurred in
res = hypotest_fun_out(*samples, **kws)

```
0s from scipy.stats import f_oneway

# Przykład: dane grupowane według kolekcji
grouped = data.groupby('metric')['value']
anova_result = f_oneway(*[group for _, group in grouped])
print(f"F-statistic: {anova_result.statistic}, P-value: {anova_result.pvalue}")

if anova_result.pvalue < 0.05:
    print("Istnieją istotne różnice między grupami.")
else:
    print("Nie znaleziono istotnych różnic między grupami.")
```

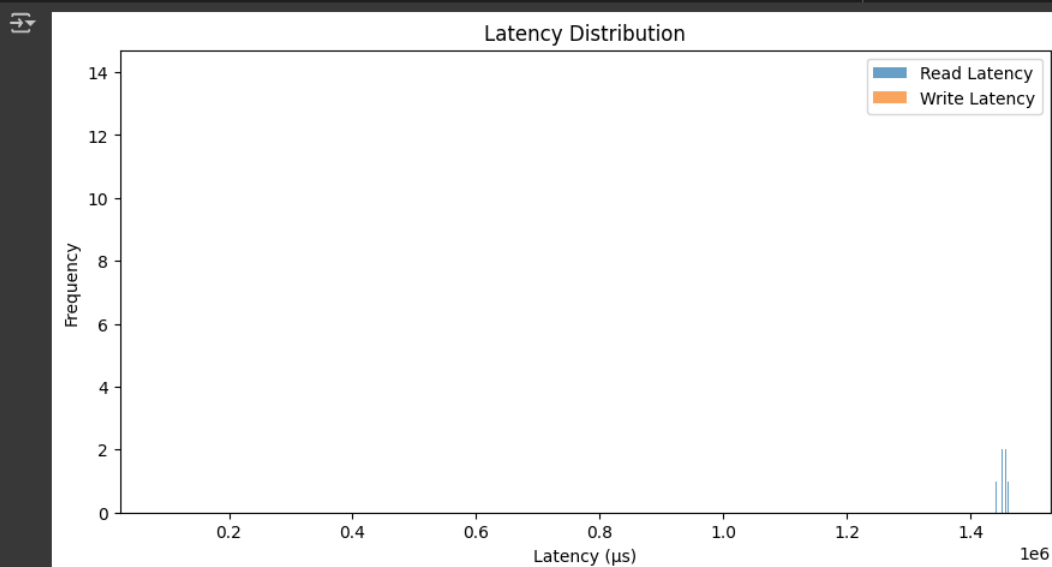
F-statistic: 617840.5412594436, P-value: 2.0105260409729953e-58
Istnieją istotne różnice między grupami.

```
0s [8] plt.figure(figsize=(10, 5))

# Histogram odczytów
plt.hist(read_latency['value'], bins=20, alpha=0.7, label='Read Latency')

# Histogram zapisów
plt.hist(write_latency['value'], bins=20, alpha=0.7, label='Write Latency')

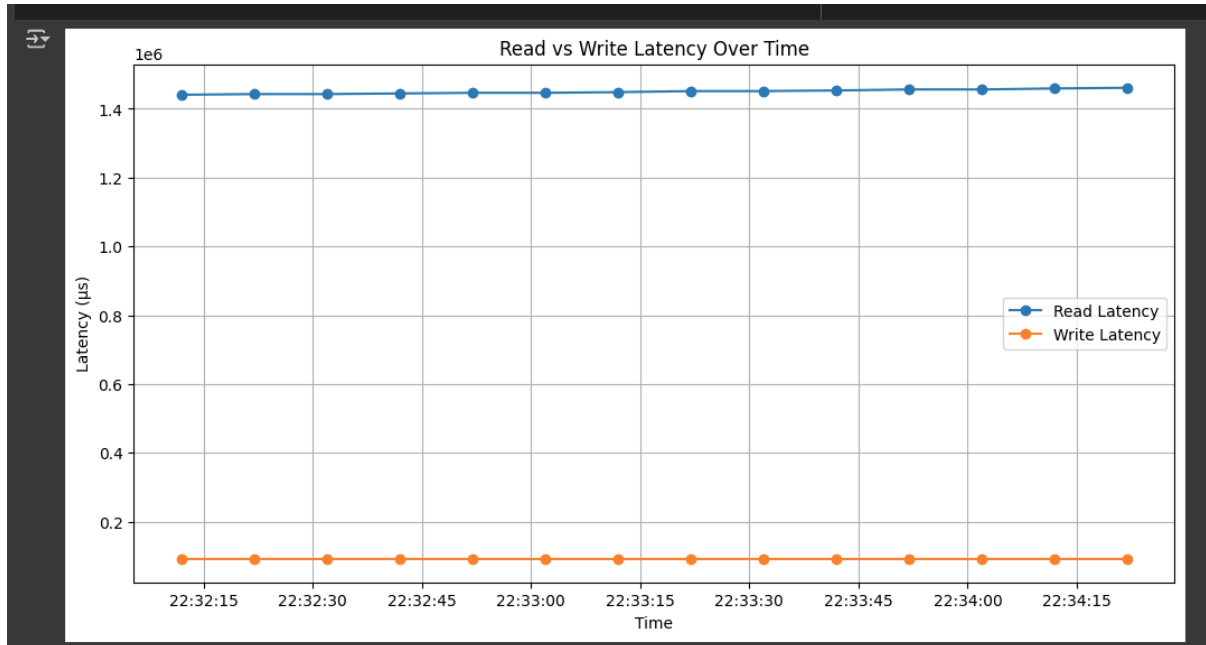
plt.title('Latency Distribution')
plt.xlabel('Latency (μs)')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



Wystarczy załadować dane z testów do colab i w razie potrzeby zmienić nazwę pliku.

Uruchomić wszystkie celę.

Z zrzutów wynika że istnieje spora różnica pomiędzy zapisywaniem i odczytem:



Spis literatury

1. API prometheus: <https://prometheus.io/docs/prometheus/latest/querying/api/>
2. Terraform: https://developer.hashicorp.com/terraform?product_intent=terraform

Spis załączników

1. Main.tf plik IoC terraform
2. *.yaml pliki z ansible playbookami
3. Data/* z przykładowymi danymi
4. Performance_test.py - testy
5. Check_mongo.py - test połączenia mongo.
6. Projekt_odzd.ipynb - Analiza danych jupyter notebook.

Podsumowanie

System został zaprojektowany w celu monitorowania i analizy wydajności bazy danych MongoDB w środowisku chmurowym, z wykorzystaniem narzędzi takich jak Prometheus, MongoDB Exporter i Grafana. Dane metryczne, takie jak latencja operacji, liczba odczytów i zapisów na sekundę oraz zużycie zasobów systemowych, są zbierane i wizualizowane, co umożliwia identyfikację wąskich gardeł i optymalizację działania systemu. Obciążenie bazy danych generowane jest za pomocą Python (pymongo), symulując różne scenariusze użytkowania, takie jak intensywne operacje odczytu, zapisu czy ich kombinacja. Analiza danych odbywa się w Pythonie za pomocą bibliotek Pandas, NumPy i SciPy, co pozwala na przeprowadzenie statystycznej analizy istotności różnic w wydajności. System został wdrożony w środowisku chmurowym Azure.