DATADOG 교육 세션
Kafka Monitoring

Agenda

Kafka Monitoring Overview

Kafka Host Monitoring + 로그 수집

Kafka Metric Integration

APM을 이용한 Kafka 트랜잭션 추적

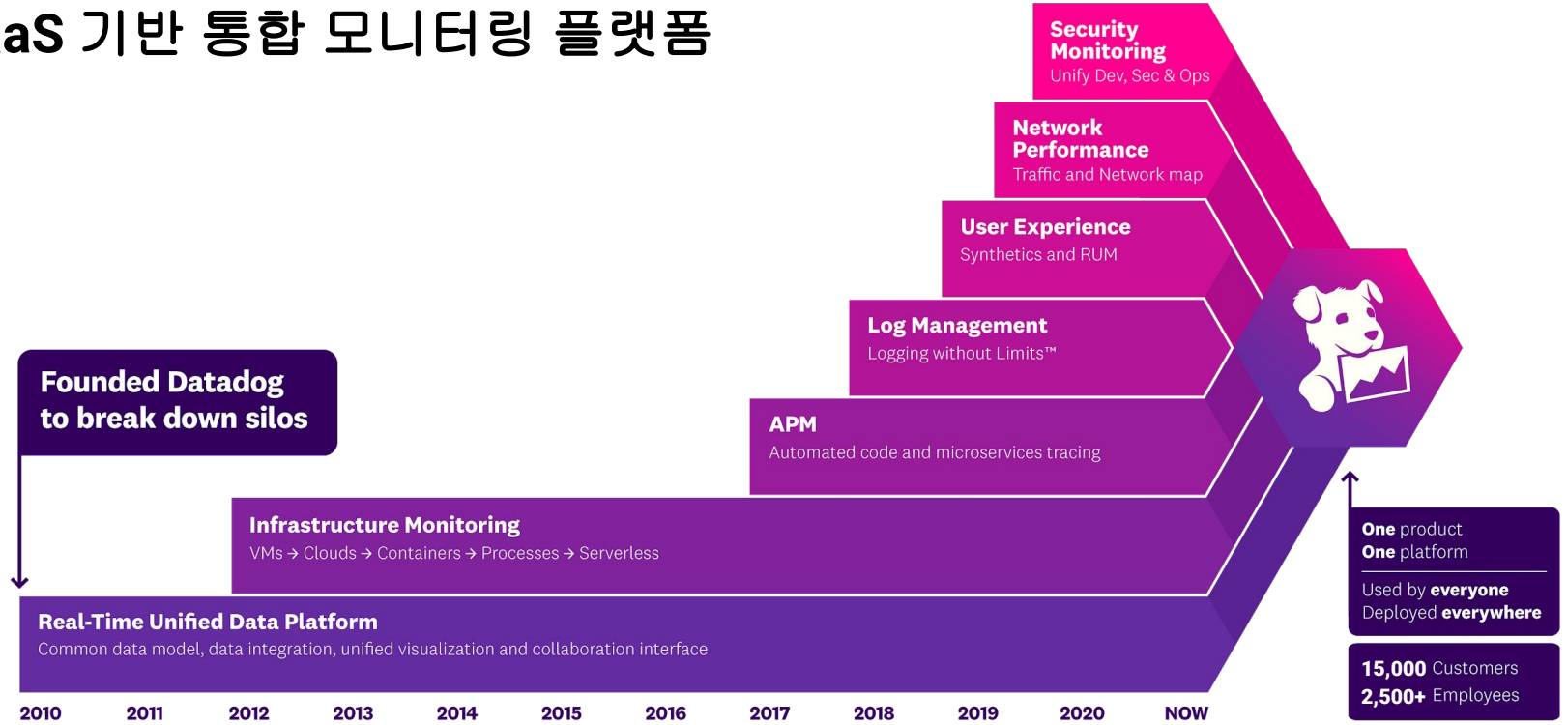
참고 Blog

Kafka deployments often rely on additional software packages not included in the Kafka codebase itself—in particular, Apache ZooKeeper. A comprehensive monitoring implementation includes all the layers of your deployment so you have visibility into your Kafka cluster and your ZooKeeper ensemble, as well as your producer and consumer applications and the hosts that run them all. To implement ongoing, meaningful monitoring, you will need a platform where you can collect and analyze your Kafka metrics, logs, and distributed request traces alongside monitoring data from the rest of your infrastructure.

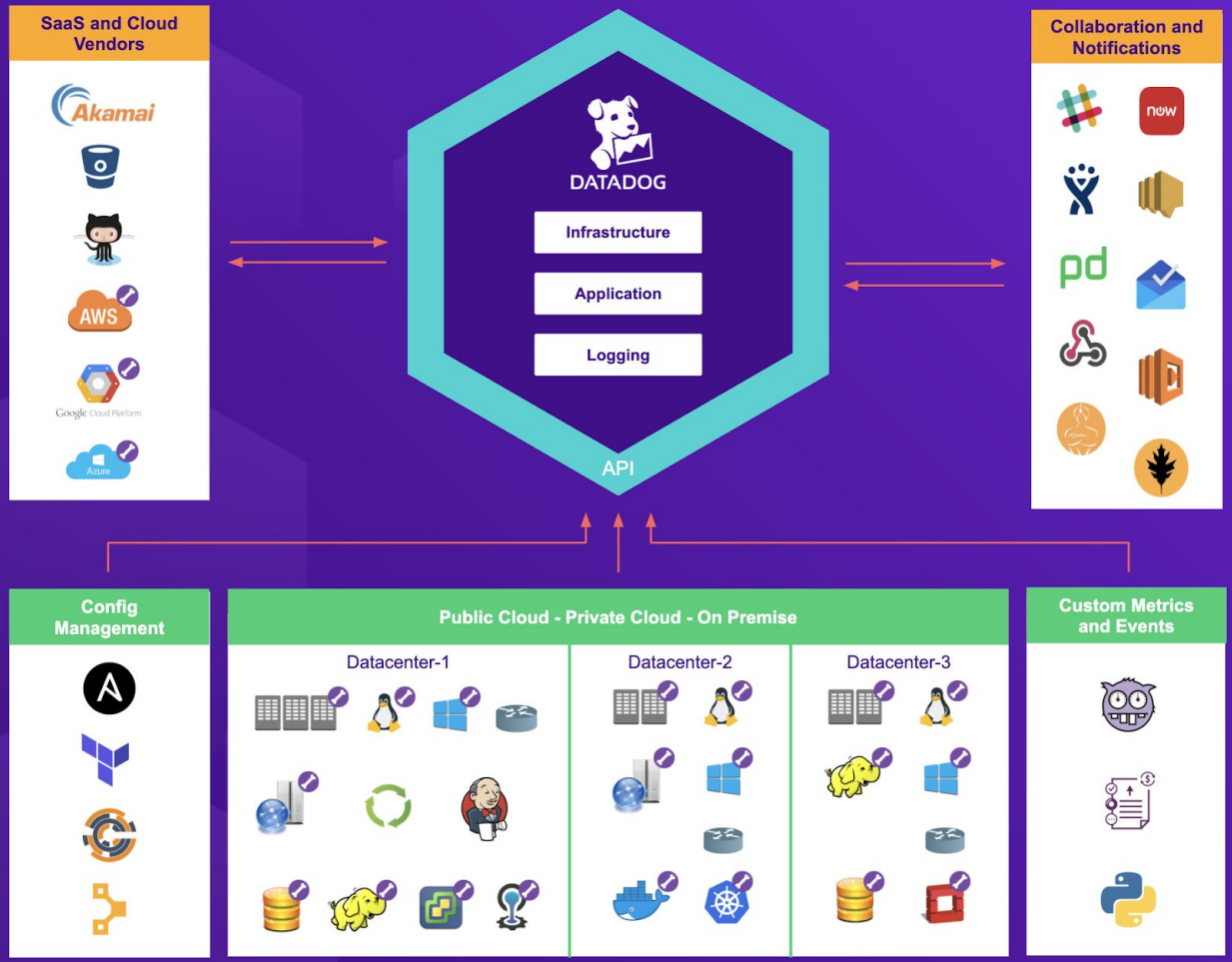
https://www.datadoghq.com/blog/monitor-kafka-with-datadog/

Datadog 소개

SaaS 기반 통합 모니터링 플랫폼



DATADOG



Infrastructure Metrics
Custom and out-of-the-box

APM
Java, Python, Ruby, Go, NodeJS, etc.

Logs & Events
Structured and unstructured

Tags/Metadata
Custom and out-of-the-box

Retention
15 months by default (extendable on request)

Granularity
No roll ups, full granularity

Secure Focused
Encrypted at Rest, SOC 2 Compliant

High Available
Spanning multiple AZ's

API
Bidirectional RESTful API supported across all product components for configuration, importing, exporting and more.

Encrypted Communication
Outbound only via HTTPS/443

System Metrics
80+ CPU, Disk, Load, Network, etc.

400+ Integrations
MySQL, NGinx, MongoDB, Hadoop, etc.

High resolution
15 sec host granularity, 1 sec. custom and business metric granularity

Kafka Monitoring Overview

공통 수집 사항

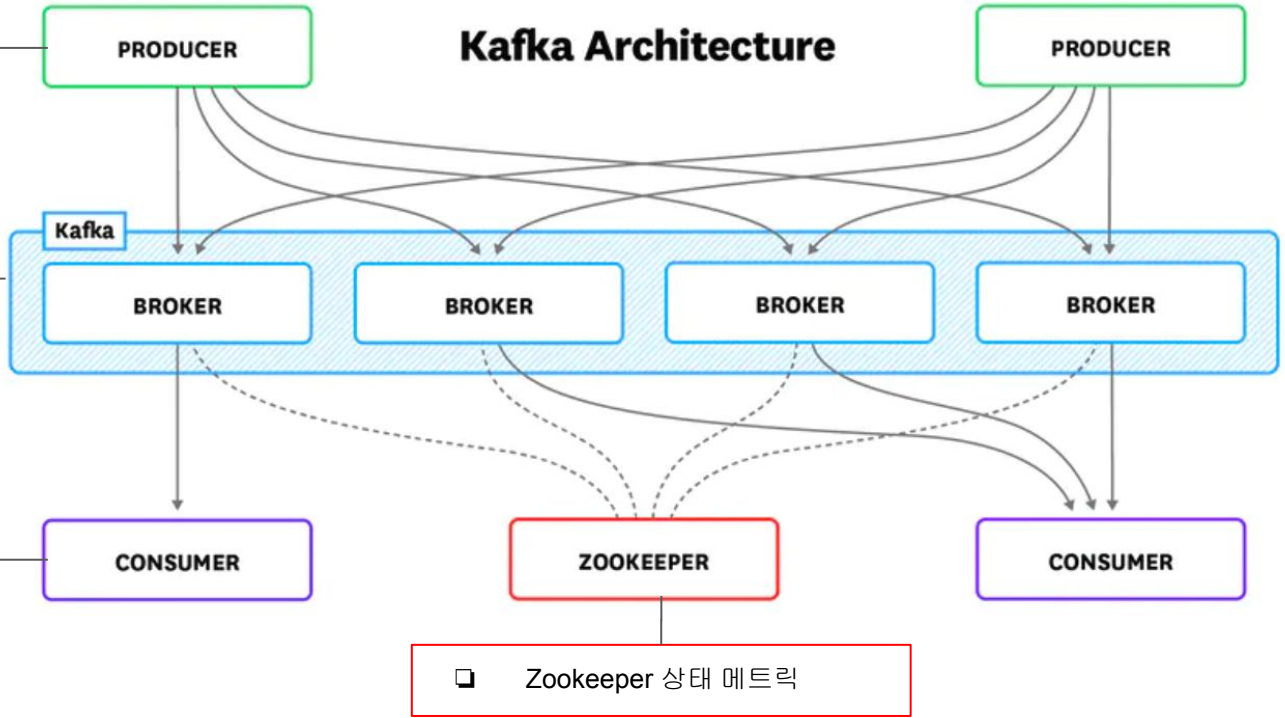
- ❑ Host 메트릭
- ❑ Log 수집
- ❑ Network Flow 수집

- ❑ Producer 상태 메트릭
- ❑ Producer 분산 트랜잭션 수집

- ❑ Kafka Broker 상태 메트릭

- ❑ Consumer 상태 메트릭
- ❑ Consumer 분산 트랜잭션 수집

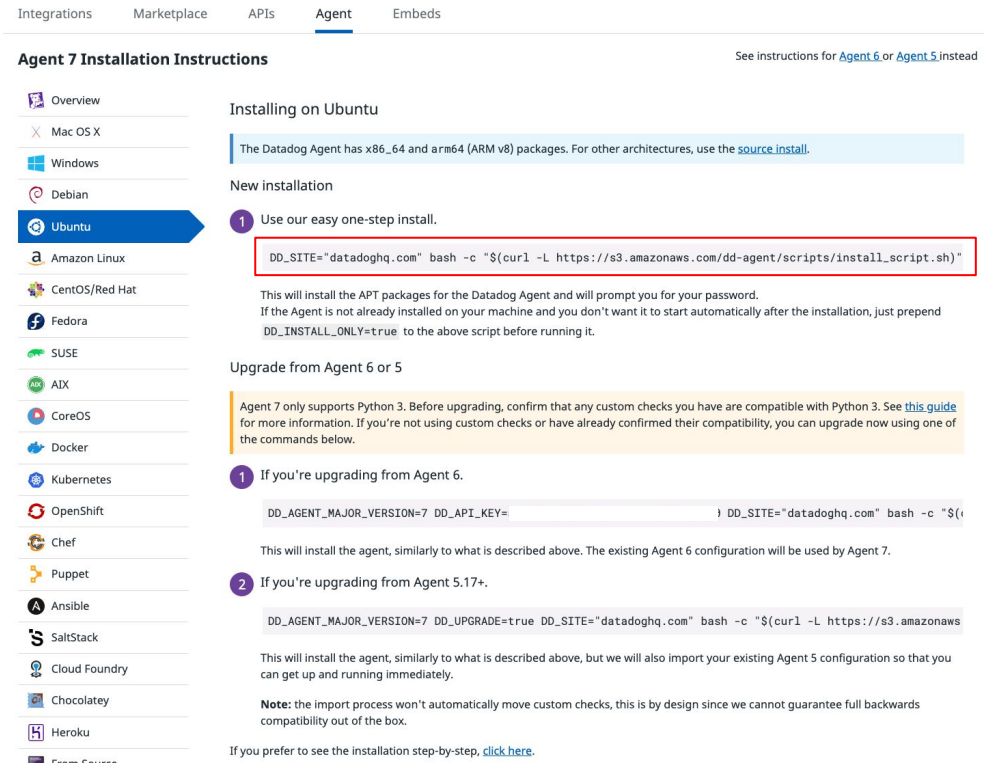
- ❑ Zookeeper 상태 메트릭



1. Host 모니터링 + 로그 수집

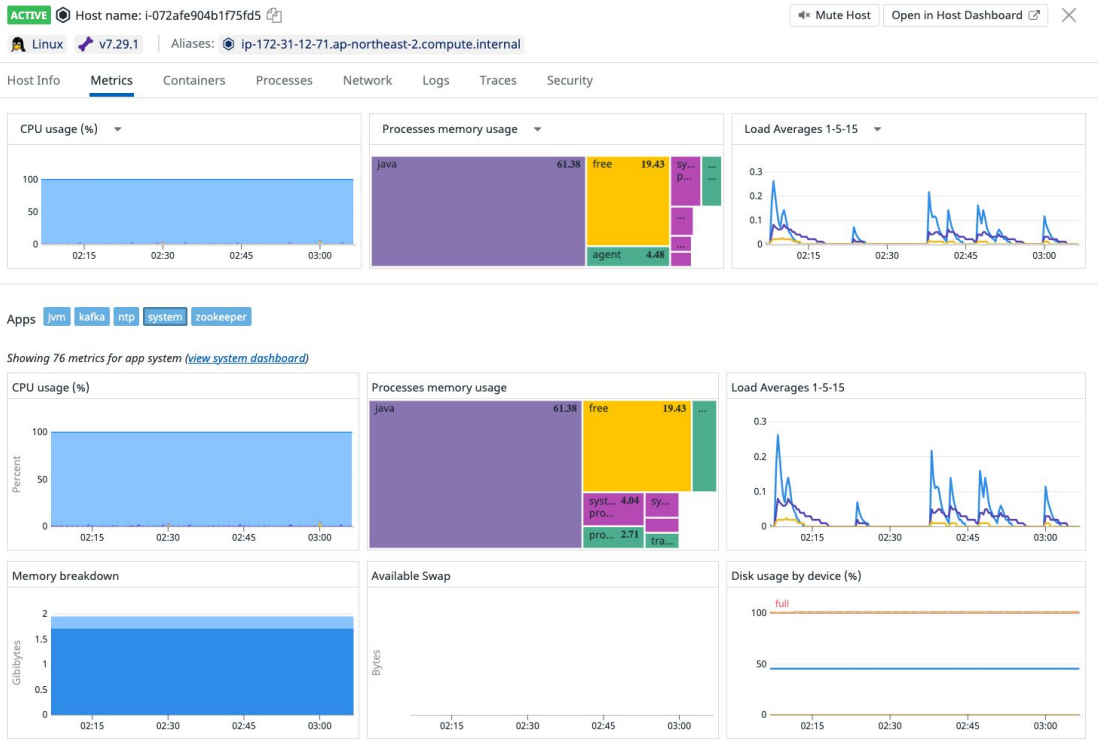
Host 메트릭 수집

API Key를 포함한 싱글 커맨드를 복사 후 Datadog agent 설치



Host 메트릭 수집

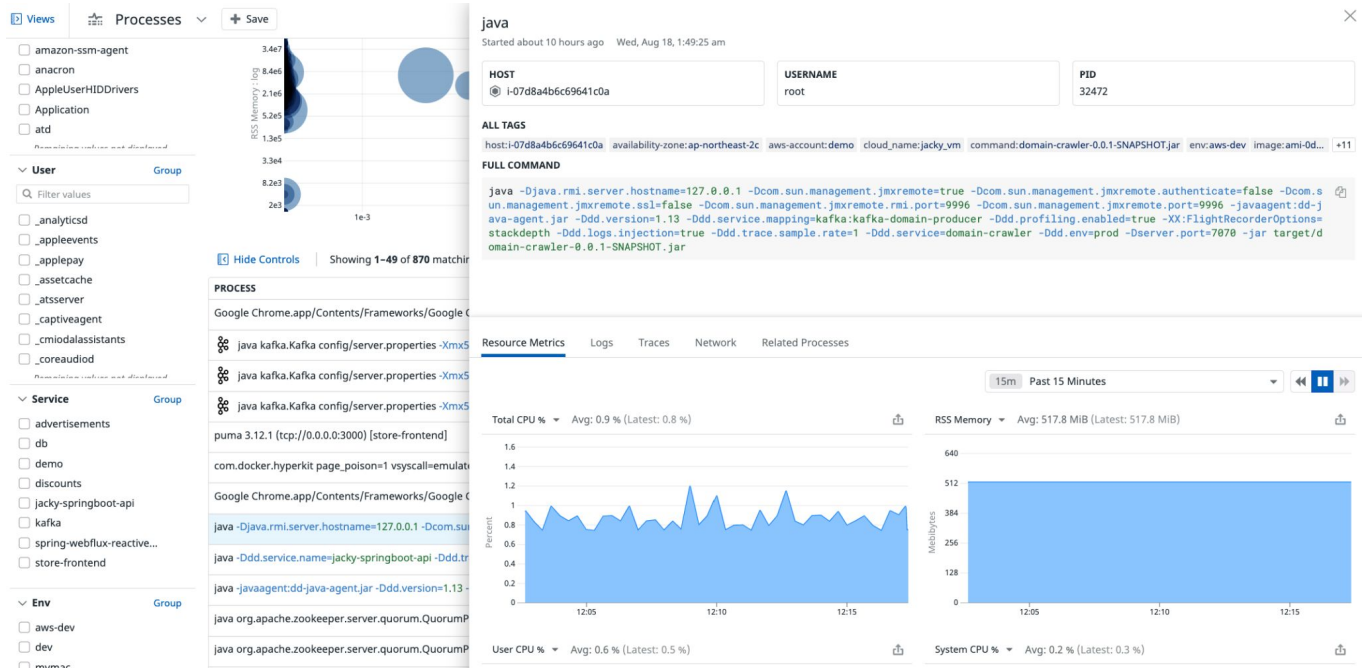
Agent 설치 후 CPU/Memory/Disk 등의 기본 시스템 메트릭 수집



Process 모니터링 활성화

1. /etc/datadog-agent/datadog.yaml에서 Process 기능 활성화 후 재시작

```
process_config:
  enabled: "true"
```



참고 링크

Network 모니터링 활성화

1. /etc/datadog-agent/system-probe.yaml 에 network 기능 활성화

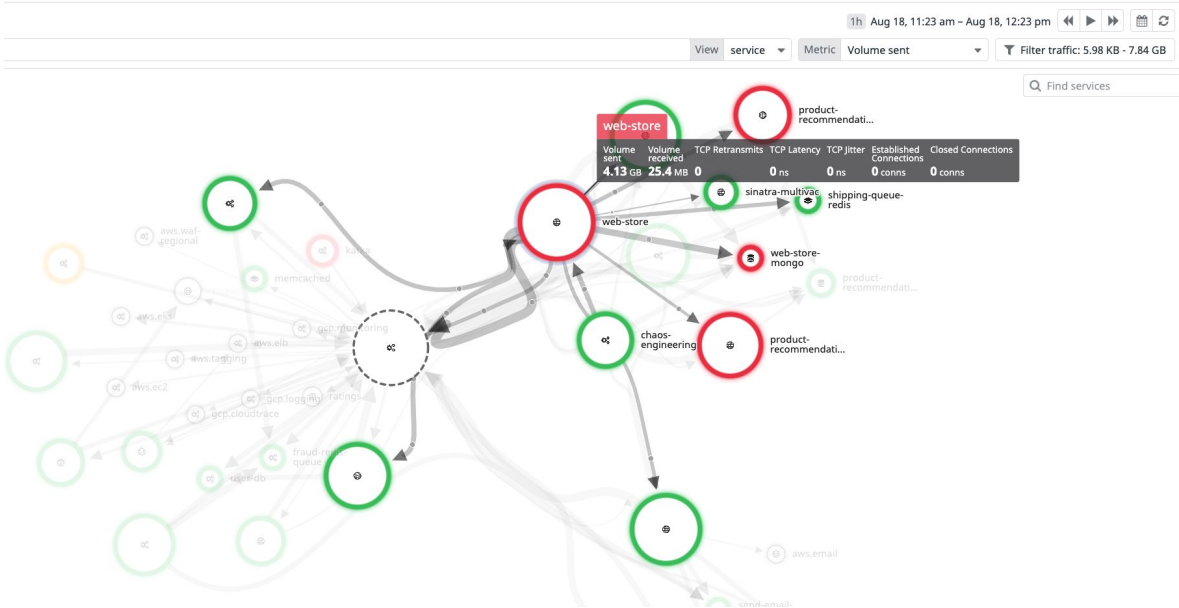
```
network_config:
    enabled: true
```

2. system-probe 서비스 등록 및 시작

```
sudo systemctl start datadog-agent-sysprobe
sudo systemctl enable datadog-agent-sysprobe
```

3. Datadog agent 재시작

```
sudo systemctl restart datadog-agent
```



참고 링크

로그 수집 설정

1. /etc/datadog-agent/datadog.yaml 에 로그 수집 기능 활성화

```
###################################
## Log collection Configuration ##
###################################

## @param logs_enabled - boolean - optional - default: false
## Enable Datadog Agent log collection by setting logs_enabled to true.
#
logs_enabled: true
```

2. 로그 수집을 희망하는 파일과 tag를 입력한 후 datadog agent 재시작하면 로그 수집 시작

```
logs:
  - type: file
    path: /home/ubuntu/kafka_2.12-2.8.0/logs/*.log
    source: kafka
    tags:
    - cluster:order-kafka
    - project:apple
    - kafka-index:3
    - broker:3
```

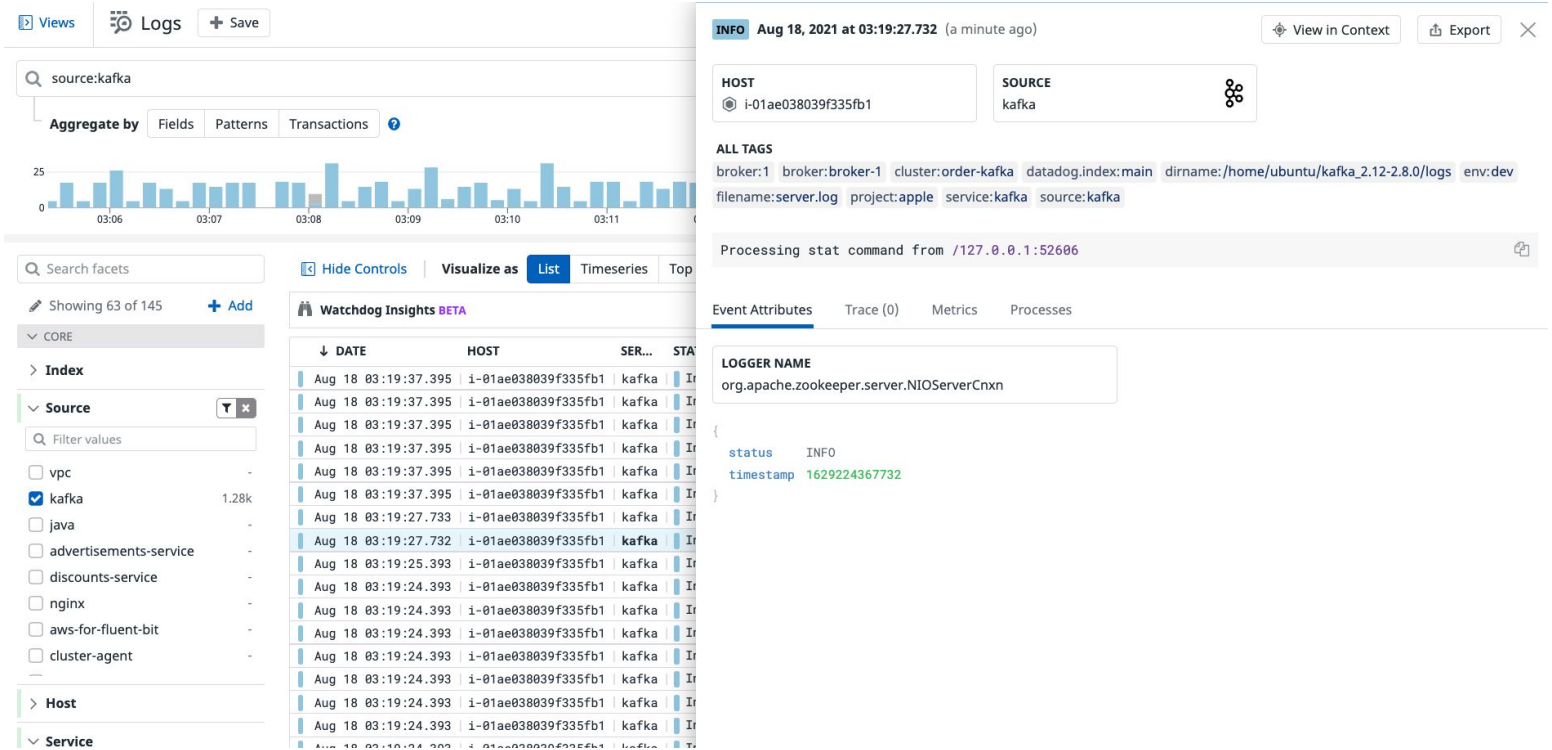
로그 설정 확인

3. datadog-agent status 명령어를 이용하여 로그 설정이 잘 되었는지 확인

```
root@ip-172-31-13-174:/home/ubuntu# datadog-agent status
Getting the status from the agent.

==========
Logs Agent
==========

    Sending compressed logs in HTTPS to agent-http-intake.logs.datadoghq.com on port 443
    BytesSent: 2.006076e+06
    EncodedBytesSent: 331420
    LogsProcessed: 5542
    LogsSent: 5542

  kafka
  -----
    - Type: file
      Path: /home/ubuntu/kafka_2.12-2.8.0/logs/*.log
      Status: OK
        8 files tailed out of 8 files matching
      Inputs:
        /home/ubuntu/kafka_2.12-2.8.0/logs/kafka-authorizer.log
        /home/ubuntu/kafka_2.12-2.8.0/logs/controller.log
        /home/ubuntu/kafka_2.12-2.8.0/logs/zookeeper-gc.log
        /home/ubuntu/kafka_2.12-2.8.0/logs/state-change.log
        /home/ubuntu/kafka_2.12-2.8.0/logs/server.log
        /home/ubuntu/kafka_2.12-2.8.0/logs/log-cleaner.log
        /home/ubuntu/kafka_2.12-2.8.0/logs/kafkaServer-gc.log
        /home/ubuntu/kafka_2.12-2.8.0/logs/kafka-request.log
      BytesRead: 562399
      Average Latency (ms): 1
      24h Average Latency (ms): 1
      Peak Latency (ms): 720
      24h Peak Latency (ms): 720
      MultiLine matches: 5542
```

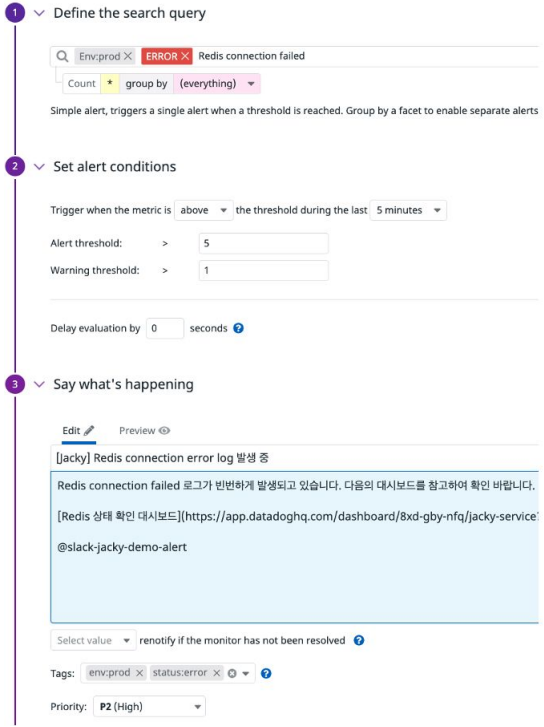
로그 수집 확인

4. Tag 및 로그 메시지를 이용하여 검색 혹은 시각화 가능

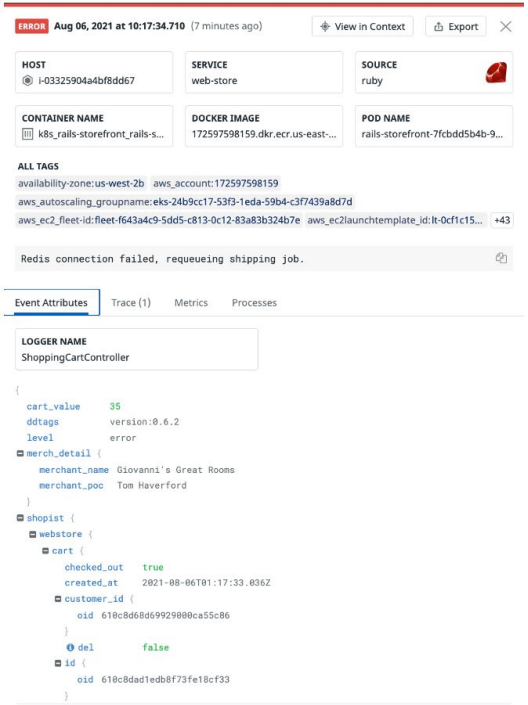


3. [로그 기반] 특정 에러 로그 발견시 트러블 슈팅

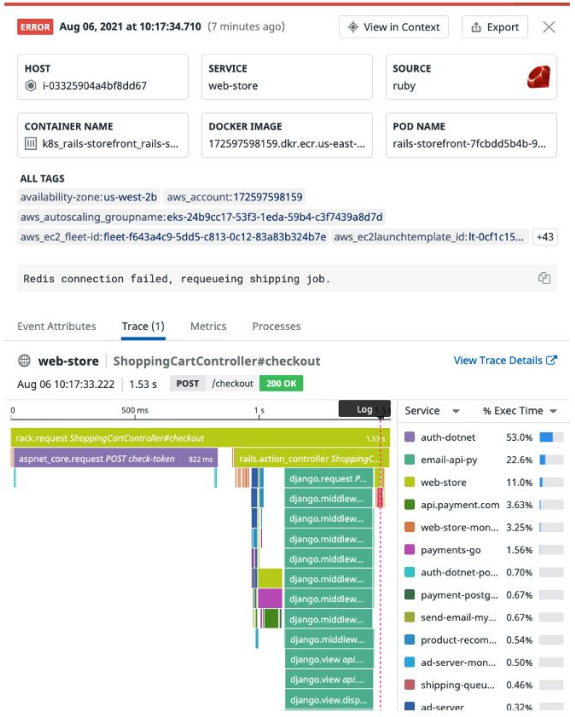
1) 알람 조건 설정



2) 문제 로그 확인



3) 문제 로그가 발생된 Application 위치 확인



2. Kafka Metric Integration

Kafka broker 메트릭 수집

공통 수집 사항

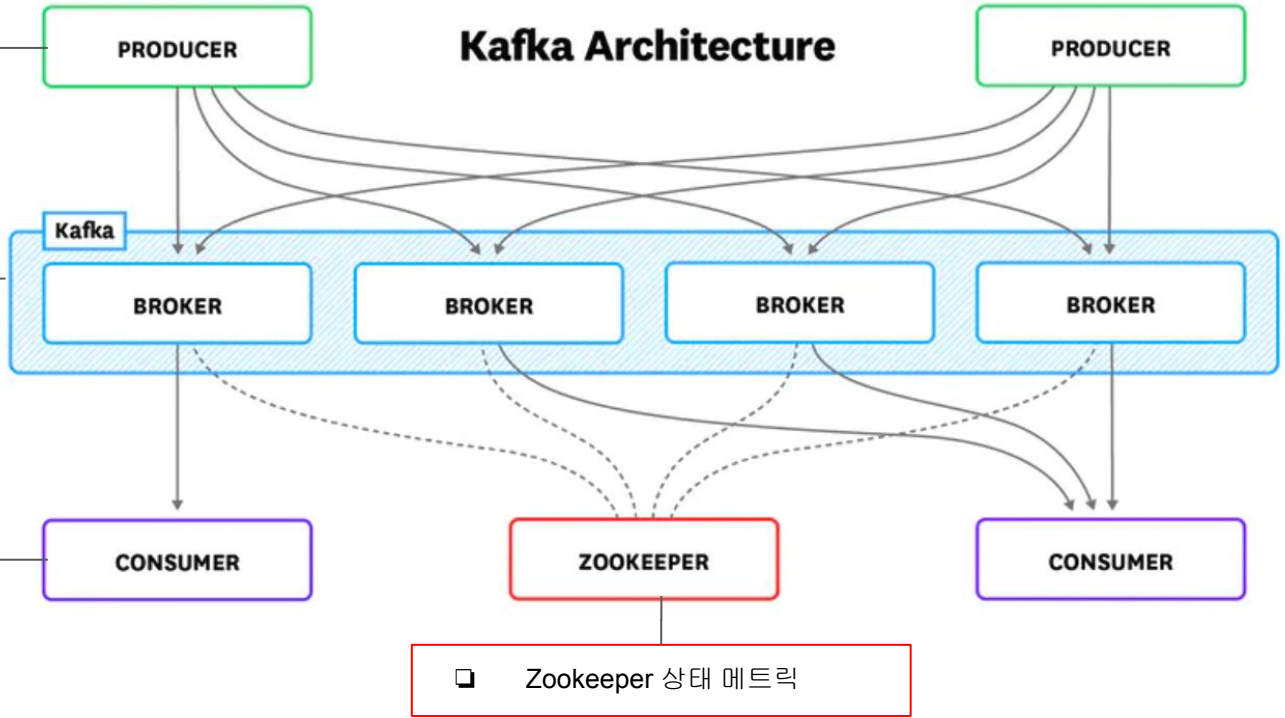
- ❑ Host 메트릭
- ❑ Log 수집
- ❑ Network Flow 수집

- ❑ Producer 상태 메트릭
- ❑ Producer 분산 트랜잭션 수집

- ❑ Kafka Broker 상태 메트릭

- ❑ Consumer 상태 메트릭
- ❑ Consumer 분산 트랜잭션 수집

- ❑ Zookeeper 상태 메트릭



Kafka Integration (Broker)

1. Kafka broker에서 JMX remote port 활성화 (이미 활성화 되어 있다면 다음 단계로 진행)

```
export KAFKA_JMX_OPTS="-Djava.rmi.server.hostname=127.0.0.1 -Dcom.sun.management.jmxremote=true
-Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.rmi.port=9999 -Dcom.sun.management.jmxremote.port=9999"
```

2. /etc/datadog-agent/conf.d/kafka.d/conf.yaml 파일에 kafka jmx 주소와 포트 지정

- 샘플 파일 kafka.d/conf.yaml.example를 kafka.d/conf.yaml 로 변경하면 기본 준비 완료
- default로 jmx port 9999로 되어 있는 부분 확인
- 설정 완료 후 Datadog Agent 재시작

```yaml
init_config:
    is_jmx: true
    collect_default_metrics: true
instances:
  - host: localhost
    port: 9999
```

참고 링크

Kafka broker metrics

All metrics reporting across your infrastructure in the past 1 hour

Metric kafka.

Configuration

Hide Controls | Showing **1–45** of **45** metric

- Has percentiles 0
- Configured tags 0

Metric Type

- Distributions 0
- Counts, Rates, Gauges 45

METRIC NAME
kafka.consumer_lag
kafka.broker_offset
kafka.consumer_offset
kafka.messages_in.rate
kafka.net.bytes_in.rate
kafka.net.bytes_out.rate
kafka.log.flush_rate.rate
kafka.replication.max_lag
kafka.session.fetch.count
kafka.session.fetch.eviction
kafka.topic.messages_in.rate
kafka.net.bytes_rejected.rate
kafka.topic.net.bytes_in.rate
check_run.kafka.can_connect.ok
kafka.replication.leader_count
kafka.request.produce.time.avg
kafka.topic.net.bytes_out.rate
kafka.request.fetch.failed.rate
kafka.request.metadata.time.avg
kafka.request.channel.queue.size
kafka.replication.partition_count
kafka.request.produce.failed.rate
kafka.session.zookeeper.sync.rate
kafka.replication.isr_expands.rate
kafka.replication.isr_shrinks.rate
kafka.session.zookeeper.expire.rate
kafka.net.processor.avg.idle.pct.rate
kafka.request.fetch_consumer.time.avg
kafka.request.fetch_follower.time.avg
kafka.session.zookeeper.readonly.rate
kafka.request.update_metadata.time.avg
kafka.replication.leader_elections.rate
kafka.request.handler.avg.idle.pct.rate
kafka.request.produce.time.99percentile
kafka.session.zookeeper.disconnect.rate
kafka.request.metadata.time.99percentile
kafka.replication.active_controller_count
kafka.replication.offline_partitions_count
kafka.request.fetch_request_purgatory.size
kafka.replication.under_replicated_partitions
kafka.request.producer_request_purgatory.size
kafka.request.fetch_consumer.time.99percentile
kafka.request.fetch_follower.time.99percentile
kafka.replication.unclean_leader_elections.rate
kafka.request.update_metadata.time.99percentile

METRIC NAME
jvm.heap_memory
jvm.gc.cms.count
jvm.gc.eden_size
jvm.thread_count
jvm.gc.parnew.time
jvm.loaded_classes
jvm.cpu_load.system
jvm.gc.old_gen_size
jvm.heap_memory_max
jvm.non_heap_memory
jvm.cpu_load.process
jvm.gc.survivor_size
jvm.heap_memory_init
jvm.gc.metaspace_size
jvm.non_heap_memory_max
jvm.non_heap_memory_init
jvm.heap_memory_committed
jvm.buffer_pool.direct.used
jvm.buffer_pool.mapped.used
jvm.buffer_pool.direct.count
jvm.buffer_pool.mapped.count
jvm.os.open_file_descriptors
jvm.non_heap_memory_committed
jvm.buffer_pool.direct.capacity
jvm.buffer_pool.mapped.capacity

Kafka Producer/Consumer 메트릭 수집

공통 수집 사항

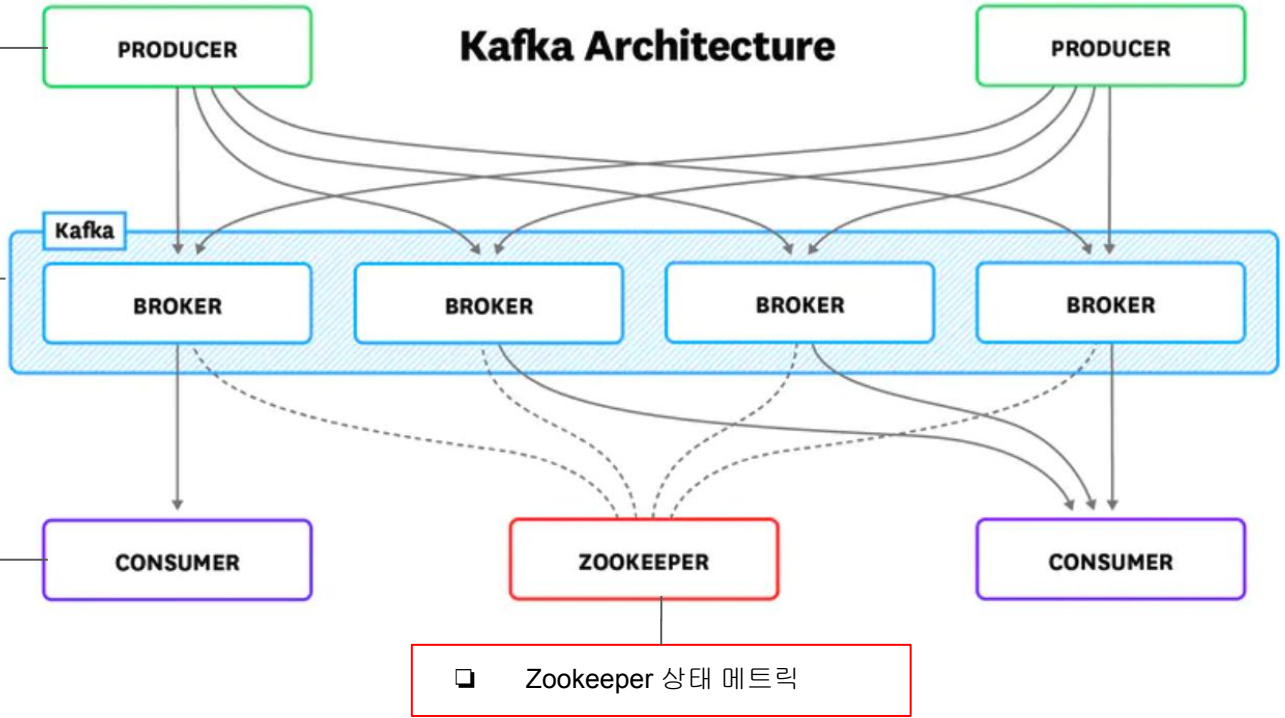
- ❑ Host 메트릭
- ❑ Log 수집
- ❑ Network Flow 수집

- ❑ Producer 상태 메트릭
- ❑ Producer 분산 트랜잭션 수집

- ❑ Kafka Broker 상태 메트릭

- ❑ Consumer 상태 메트릭
- ❑ Consumer 분산 트랜잭션 수집

- ❑ Zookeeper 상태 메트릭



Kafka Integration (Producer / Consumer)

1. Producer/Consumer Application에서 JMX remote port 활성화

```
export KAFKA_JMX_OPTS="-Djava.rmi.server.hostname=127.0.0.1 -Dcom.sun.management.jmxremote=true
-Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.rmi.port=9999 -Dcom.sun.management.jmxremote.port=9999"

java $KAFKA_JMX_OPTS -jar myapp.jar
```

2. /etc/datadog-agent/conf.d/kafka.d/conf.yaml 파일에 kafka jmx 주소와 포트

- 샘플 파일 kafka.d/conf.yaml.example를 kafka.d/conf.yaml 로 변경 후 사용
- default로 jmx port 9999로 되어 있는 부분 확인
- 설정 완료 후 Datadog Agent 재시작

```
init_config:
    is_jmx: true
    collect_default_metrics: true
instances:
  - host: localhost
    port: 9999
    tags:
```

참고 링크

Kafka Producer/Consumer metrics

All metrics reporting across your infrastructure in the past 1 hour

Metric kafka.

Configuration

- Has percentiles 0
- Configured tags 0

Metric Type

- Distributions 0
- Counts, Rates, Gauges 32

Hide Controls | Showing **1–32** of **32** metrics

METRIC NAME
kafka.producer.io_wait
kafka.consumer.bytes_in
kafka.producer.bytes_out
kafka.consumer.messages_in
kafka.producer.metadata_age
kafka.producer.request_rate
kafka.producer.response_rate
kafka.consumer.fetch_size_avg
kafka.consumer.fetch_size_max
kafka.producer.batch_size_avg
kafka.producer.batch_size_max
check_run.kafka.can_connect.ok
kafka.producer.record_size_avg
kafka.producer.record_size_max
kafka.producer.waiting_threads
kafka.producer.compression_rate
kafka.producer.record_send_rate
kafka.producer.record_error_rate
kafka.producer.record_retry_rate
kafka.producer.records_send_rate
kafka.producer.throttle_time_avg
kafka.producer.throttle_time_max
kafka.producer.buffer_bytes_total
kafka.producer.requests_in_flight
kafka.producer.records_per_request
kafka.producer.request_latency_avg
kafka.producer.request_latency_max
kafka.producer.compression_rate_avg
kafka.producer.record_queue_time_avg
kafka.producer.record_queue_time_max
kafka.producer.available_buffer_bytes
kafka.consumer.records_per_request_avg
jvm.heap_memory
jvm.gc.cms.count
jvm.gc.eden_size
jvm.thread_count
jvm.gc.parnew.time
jvm.loaded_classes
jvm.cpu_load.system
jvm.gc.old_gen_size
jvm.heap_memory_max
jvm.non_heap_memory
jvm.cpu_load.process
jvm.gc.survivor_size
jvm.heap_memory_init
jvm.gc.metaspace_size
jvm.non_heap_memory_max
jvm.non_heap_memory_init
jvm.heap_memory_committed
jvm.buffer_pool.direct.used
jvm.buffer_pool.mapped.used
jvm.buffer_pool.direct.count
jvm.buffer_pool.mapped.count
jvm.os.open_file_descriptors
jvm.non_heap_memory_committed
jvm.buffer_pool.direct.capacity
jvm.buffer_pool.mapped.capacity

Zookeeper Integration

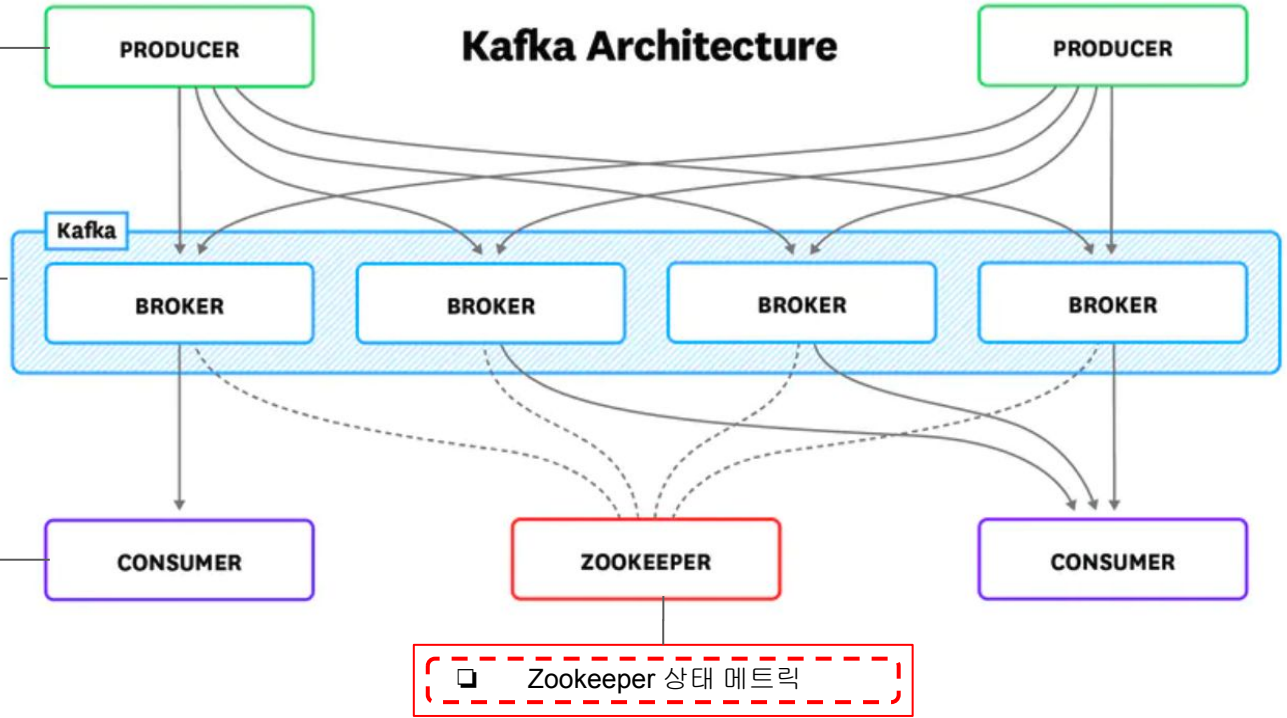
공통 수집 사항

- ❑ Host 메트릭
- ❑ Log 수집
- ❑ Network Flow 수집

- ❑ Producer 상태 메트릭
- ❑ Producer 분산 트랜잭션 수집

- ❑ Kafka Broker 상태 메트릭

- ❑ Consumer 상태 메트릭
- ❑ Consumer 분산 트랜잭션 수집



Zookeeper Integration

1. conf/zookeeper.properties 설정 파일에 모니터링 위한 command 추가

    4lw.commands.whitelist=stat, ruok, conf, isro, mntr

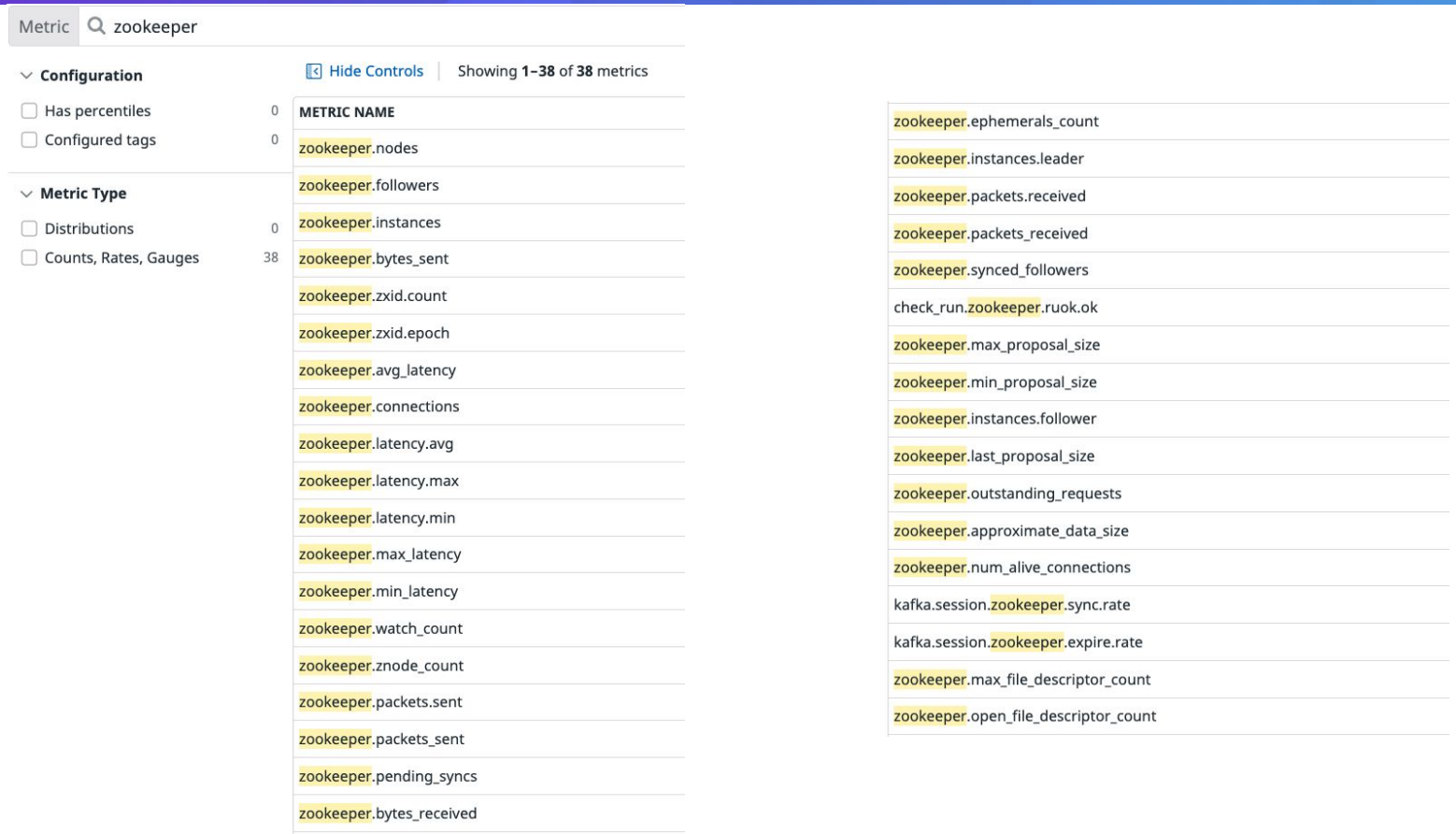
2. /etc/datadog-agent/conf.d/zk.d/conf.yaml 파일에 zookeeper 주소와 포트 지정 후 Agent 재시작

- 샘플 파일 conf.d/zk.d/conf.yaml.example를 conf.d/zk.d/conf.yaml 로 변경 후 사용
- 설정 완료 후 Datadog Agent 재시작

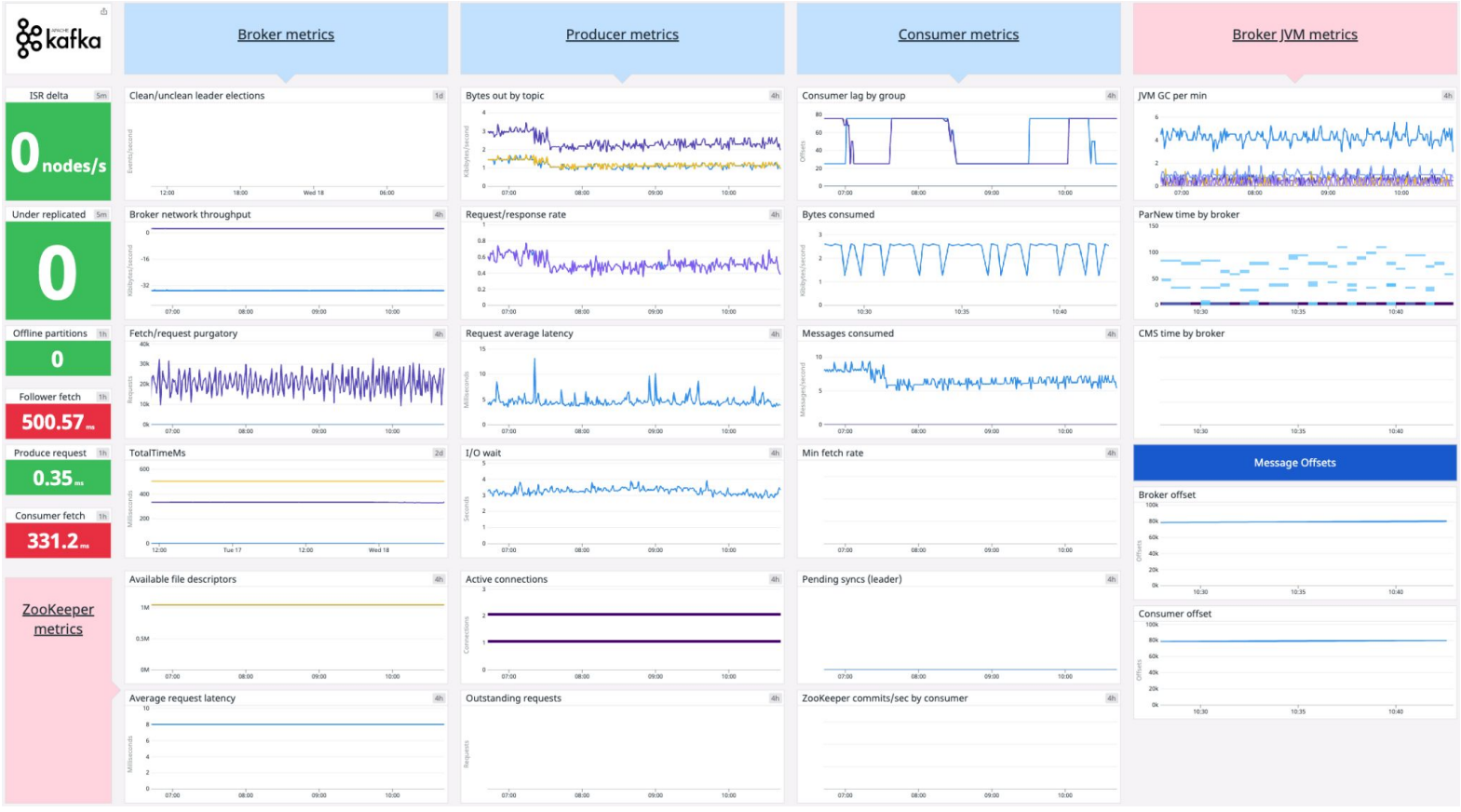
```yaml
init_config:
instances:
  - host: localhost
    port: 2181
```

참고 링크

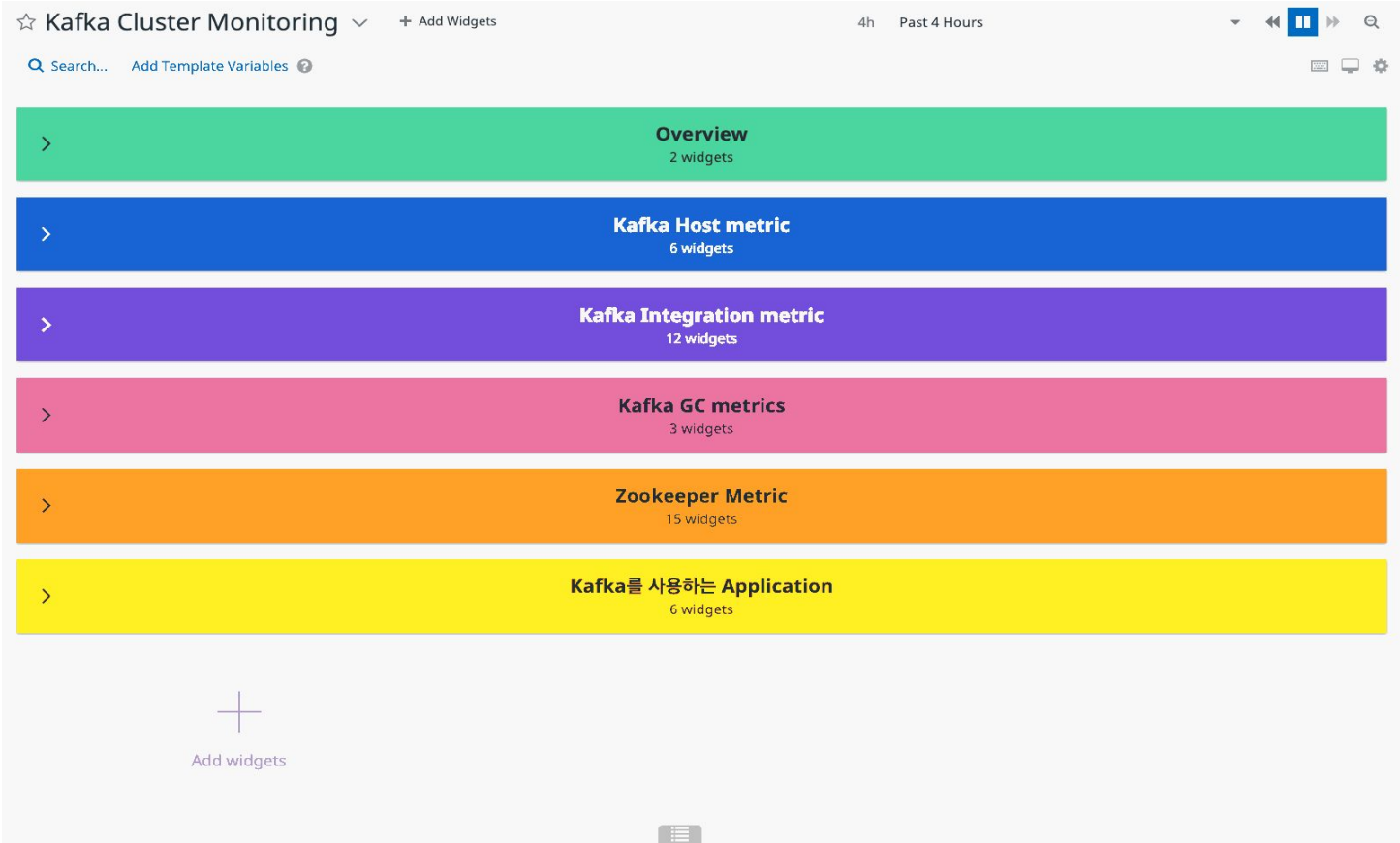
Kafka Producer/Consumer metrics



Kafka 기본 대시보드

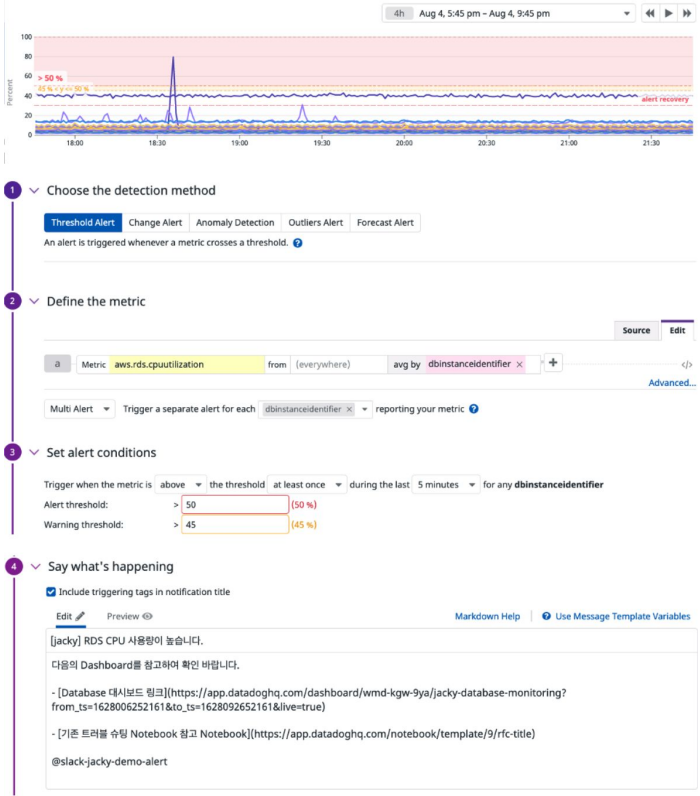


Kafka Custom 대시보드

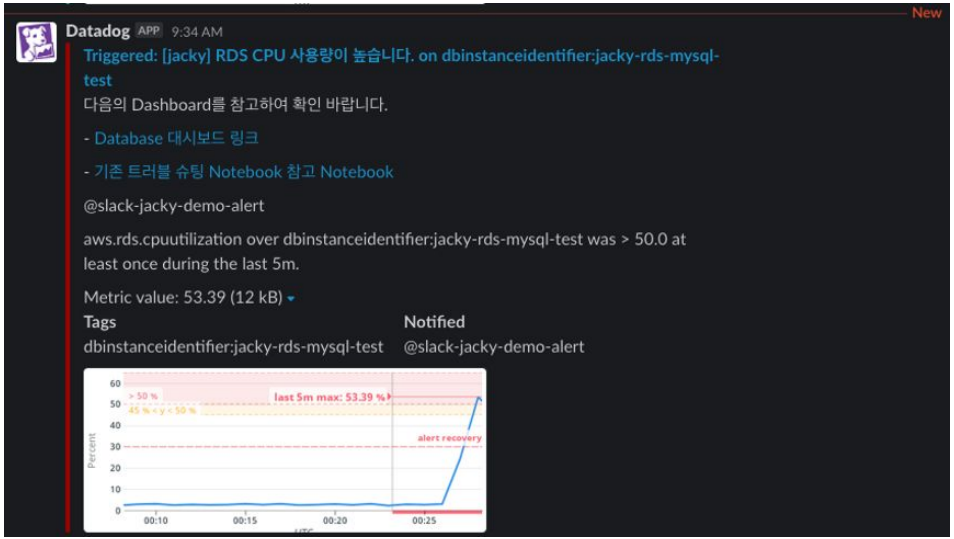


메트릭 기반 알람 활용

1) 알람 조건 설정

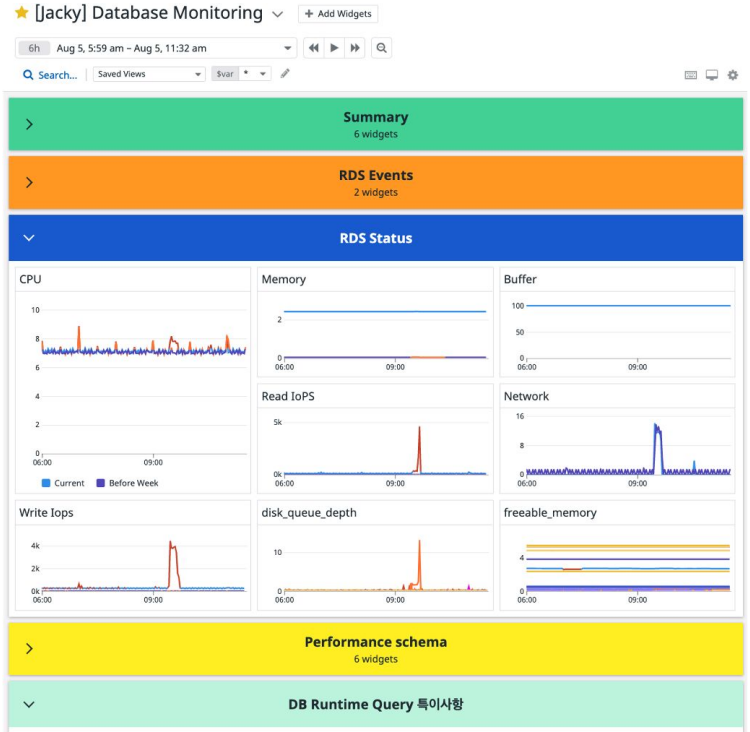


2) 알람 메시지(Slack)를 통한 이슈 확인



메트릭 기반 알람 활용

3) 알람 메시지를 참고하여 연계 대시보드로 넘어와 분석



4) 장애 분석 내용을 **Notebook**에 정리하여 팀원과 협업

3. APM을 이용한 Kafka 트랜잭션 추적

Producer 및 Consumer 트랜잭션 추적

공통 수집 사항

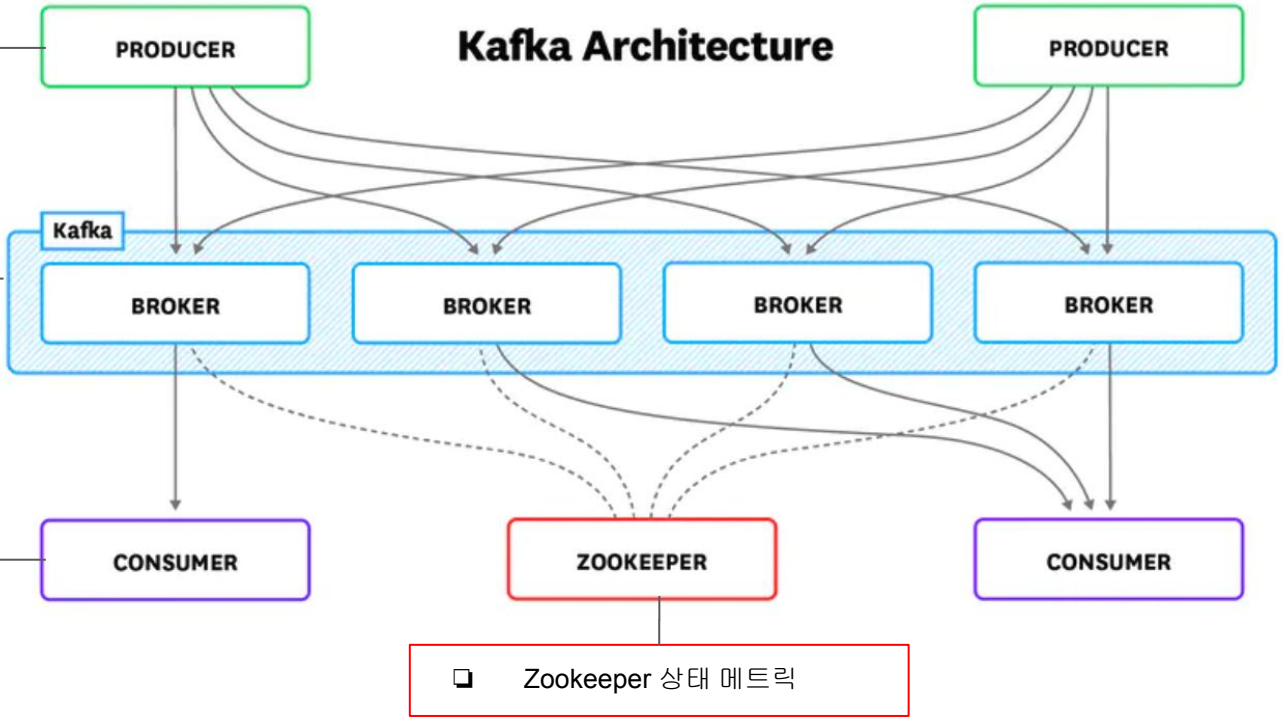
- ❑ Host 메트릭
- ❑ Log 수집
- ❑ Network Flow 수집

- ❑ Producer 상태 메트릭
- ❑ Producer 분산 트랜잭션 수집

- ❑ Kafka Broker 상태 메트릭

- ❑ Consumer 상태 메트릭
- ❑ Consumer 분산 트랜잭션 수집

- ❑ Zookeeper 상태 메트릭



APM 연동 방법 (Producer/Consumer)

1. Datadog Trace 라이브러리 다운로드 (Java application 기준 설명)

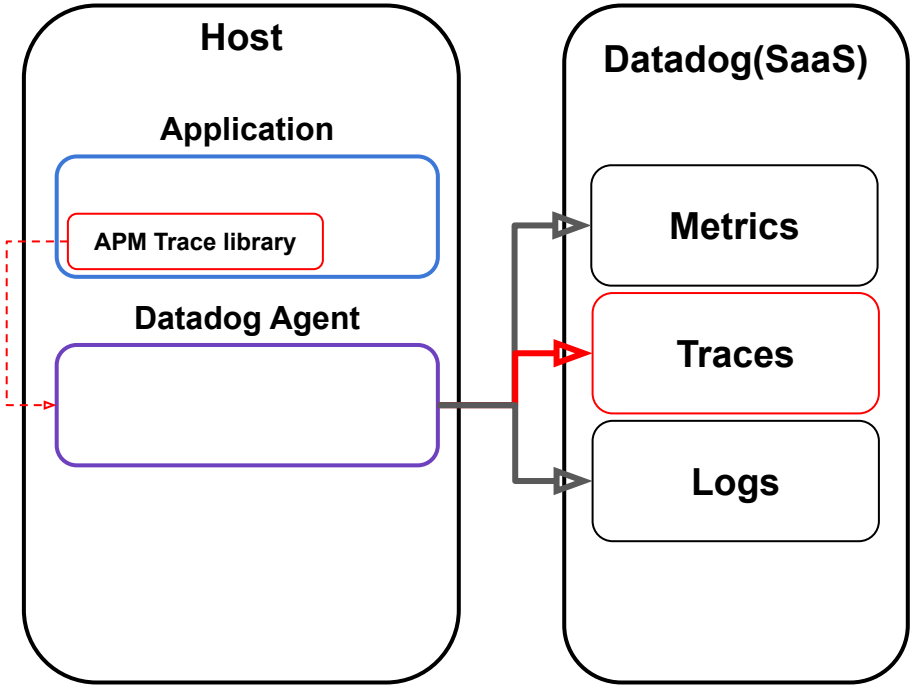
```
wget -O dd-java-agent.jar 'https://dtdg.co/latest-java-tracer'
```

2. Java application 실행 옵션에 -javaagent 옵션을 사용하여 trace 라이브러리 삽입 후 app 시작

```
export KAFKA_JMX_OPTS="-Djava.rmi.server.hostname=127.0.0.1 -Dcom.sun.management.jmxremote=true
-Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.rmi.port=9999 -Dcom.sun.management.jmxremote.port=9999"

java $KAFKA_JMX_OPTS -javaagent:dd-java-agent.jar -Ddd.version=1.13 -Ddd.logs.injection=true -Ddd.trace.sample.rate=1
-Ddd.service=domain-crawler -Ddd.env=prod  -Dserver.port=7070 -jar myapp.jar
```

참고 링크

APM 연동 아키텍쳐


Host
Application
APM Trace library
Datadog Agent
Datadog(SaaS)
Metrics
Traces
Logs

APM Compatibility

1. 다음의 언어에 대해 **APM** 기능 지원

2. Java의 경우 **Java7** 이상부터 **APM** 지원

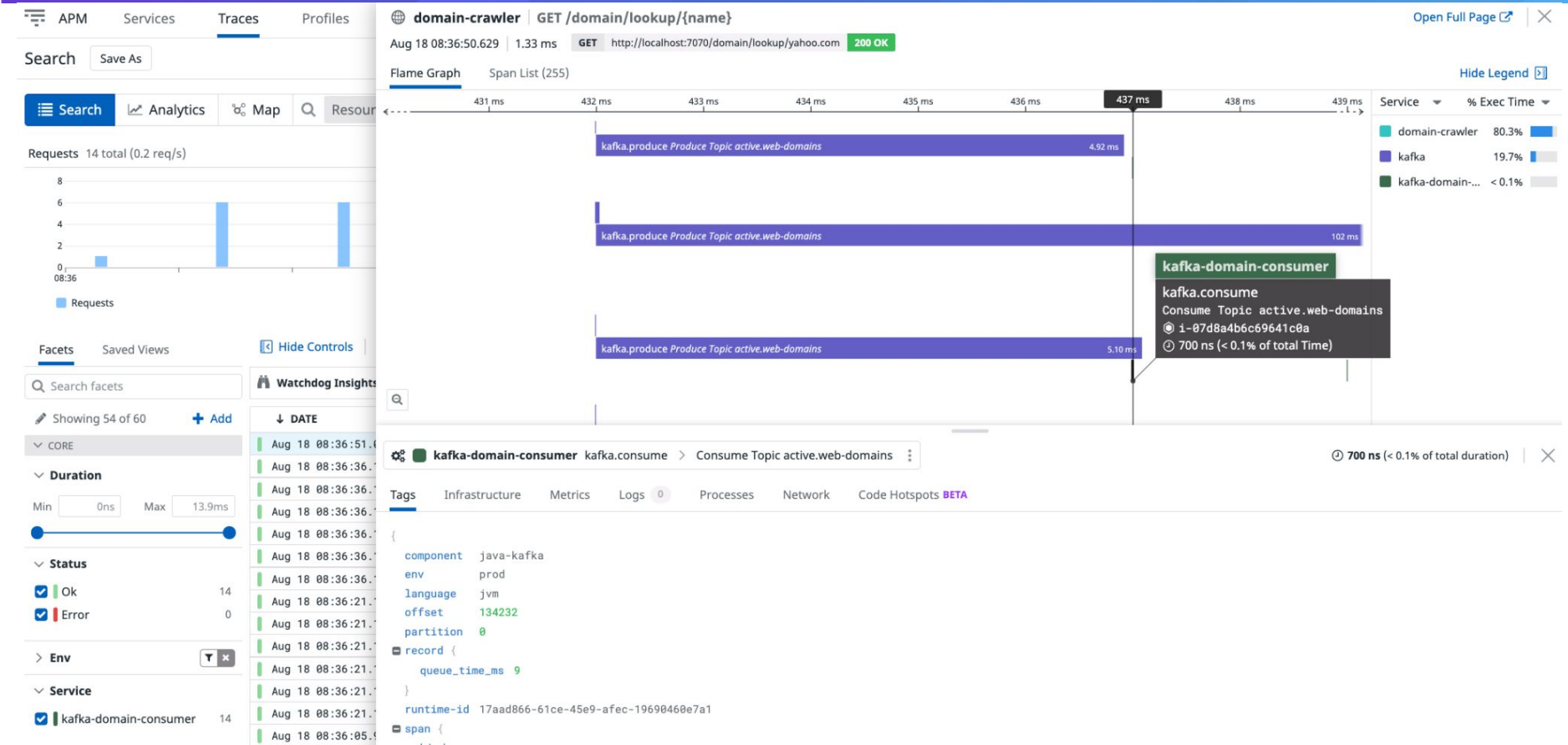
3. 언어별 탐지되는 라이브러리 및 버전 확인 필요

- 언어별 Compatibility 페이지 참고
- Java의 경우 Kafka-Clients 0.11+, Kafka-Streams 0.11+ 에 대해 가시성 지원

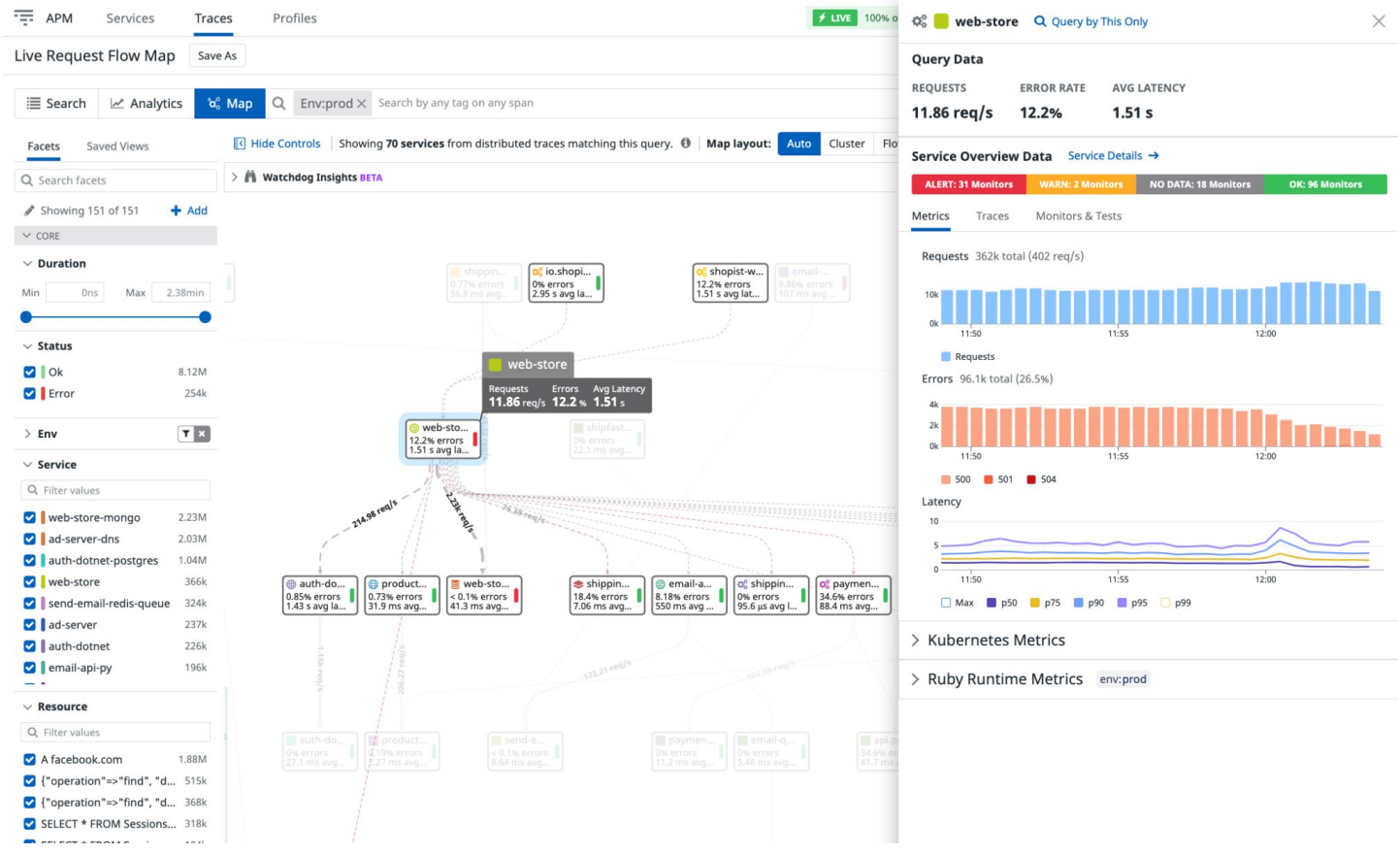
FRAMEWORK	VERSIONS	SUPPORT TYPE	INSTRUMENTATION NAMES (USED FOR CONFIGURATION)
Apache HTTP Client	4.0+	Fully Supported	`httpclient`, `apache-httpclient`, `apache-http-client`
Apache HTTP Async Client	4.0+	Fully Supported	`httpasyncclient`, `apache-httpasyncclient`
AWS Java SDK	1.11+, 2.2+	Fully Supported	`aws-sdk`
Commons HTTP Client	2.0+	Fully Supported	`commons-http-client`
Google HTTP Client	1.19.0+	Fully Supported	`google-http-client`
Grizzly HTTP Client	1.9+	Beta	`grizzly-client`
gRPC	1.5+	Fully Supported	`grpc`, `grpc-client`, `grpc-server`
HttpURLConnection	all	Fully Supported	`httpurlconnection`, `urlconnection`
Kafka-Clients	0.11+	Fully Supported	`kafka`
Kafka-Streams	0.11+	Fully Supported	`kafka`, `kafka-streams`

참고 링크

APM 연동 후 Kafka 분산 트랙잭션 트래킹



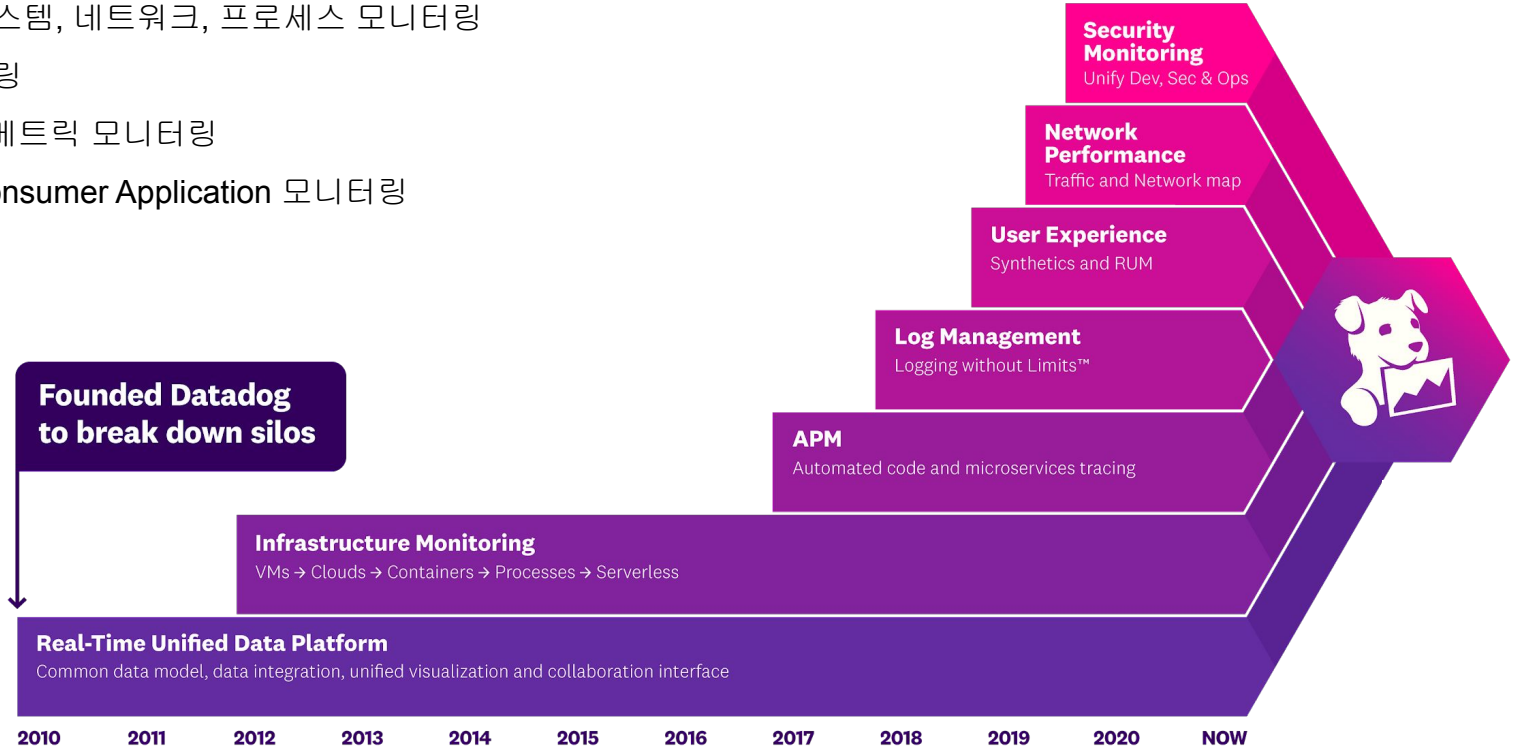
APM을 이용한 Microservice 분산 트레이싱



요약

다음의 모니터링 영역에 대해 살펴 보았습니다

1. 호스트의 시스템, 네트워크, 프로세스 모니터링
2. 로그 모니터링
3. Kafka 주요 메트릭 모니터링
4. Producer/Consumer Application 모니터링



Datadog 사용에 대한 기대 효과

Datadog 모니터링 플랫폼에 주요 모니터링 요소를 통합함으로써

1) 모니터링 연동을 위한 개발 및 관리 시간 최소화
2) 빠른 연계 분석을 통해 서비스 장애 분석 시간 최소화
3) 개발팀과 운영팀 협업 효율 증가

Q&A

감사합니다