

Grafana를 이용한 OpenStack 클라우드 성능 모니터링



Stella team

Operating Systems Lab, Korea University
(starlab@os.korea.ac.kr)

OpenInfra Days Korea 2018

발표 목차

☑ Stella 프로젝트

- SLA 보장의 필요성
- Stella 프로젝트 소개
- Stella 클라우드 아키텍처

☑ Ceilometer-Gnocchi 전달 과정 분석

- Gnocchi의 필요성
- Ceilometer-Gnocchi 전달 과정

☑ Grafana-Gnocchi 연동

☑ Stella 프로젝트 사이트 및 코드 공개

Stella 프로젝트

차세대 클라우드에서 성능 SLA 보장의 필요성

❑ SLA(Service Level Agreement)

- 클라우드 서비스 제공자와 사용자 간, 서비스의 품질 수준을 정의하는 계약
- 성능, 보안 등 모든 서비스 품질 요소가 대상이며 구체적인 수치(SLO, Service Level Objective)로 체결
e.g., 10ms 이하 처리지연, 1초당 1000개 쿼리 처리, 99.9% 데이터 가용성
- 서비스 수준 미달 시 페널티를 부과하기도 함 -> 서비스 비용 증대

❑ 성능 SLA를 만족하지 못하는 경우 경제적 피해 발생

- Microsoft의 사례
 - ✓ 검색 지연 증가 → 사용자 당 수익감소
- Amazon의 사례
 - ✓ 처리 지연 100ms 증가
→ 매출 1% 감소 (약 9억 달러)

	Distinct Queries/User	Query Refinement	Revenue/User	Any Clicks	Satisfaction	Time to Click (increase in ms)
50ms	-	-	-	-	-	-
200ms	-	-	-	-0.3%	-0.4%	500
500ms	-	-0.6%	-1.2%	-1.0%	-0.9%	1200
1000ms	-0.7%	-0.9%	-2.8%	-1.9%	-1.6%	1900
2000ms	-1.8%	-2.1%	-4.3%	-4.4%	-3.8%	3100

Microsoft Bing test result

SLA 위반 사례

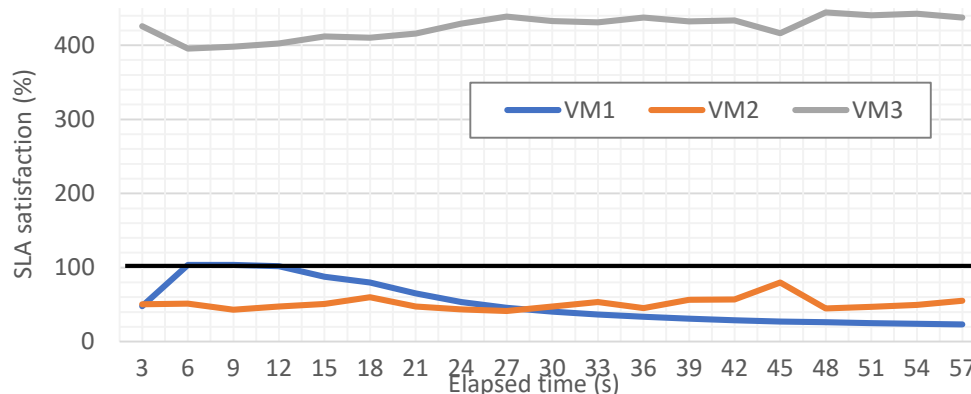
❑ 다양한 자원 요구로 인한 SLA 위반 발생

- CPU 자원 부족에 의한 급격한 스토리지 I/O 성능 저하 발견
 - ✓ VM1과 VM2는 스토리지 I/O 수행 (Iometer)
 - ✓ VM3은 반복적인 CPU 연산 수행 (Sysbench-cpu)
- 실험 결과
 - ✓ CPU VM(VM3)는 SLA 대비 평균 425% 수준
 - 평균 CPU 사용률: 850.1 %
 - ✓ I/O VM(VM1, VM2)은 SLA 대비 평균 51% 수준
 - VM1- 평균 대역폭: 51MB/s, 평균 CPU 사용률: 50.1%
 - VM2- 평균 대역폭: 104MB/s, 평균 CPU 사용률: 91%

		Specification
Host server	CPU	10 cores Xeon CPU (E5-2650 v3)
	Storage	Samsung SSD 850 PRO 256GB
	Kernel version	Linux-4.4.0 (Ubuntu 16.04)
Virtual machine	vCPU	10 cores
	Storage	SSD 10GB
	Kernel version	Linux-4.4.0 (Ubuntu 16.04)

SLA 만족도 실험 환경

❑ SLA 위반 원인: VM1 과 VM2의 I/O 처리를 위한 CPU 자원 부족



	Job Type	SLO
VM 1	Random write 4k	100 MB/s
VM 2	Random read 4k	200 MB/s
VM 3	CPU	Guarantee 2 cores

SLA 만족도 달성 실험 (SLA 보장 스토리지 스케줄러 동작 중)

Stella 프로젝트 소개

❑ 프로젝트 책임자

- 고려대학교 유혁 교수

❑ Stella 프로젝트의 목표

- 성능 SLA를 보장할 수 있는 클라우드 소프트웨어 개발
 - ✓ 다양한 자원(CPU, 스토리지, 네트워크)의 SLA 만족 달성
 - ✓ SLO를 할당하는 컴포넌트 디자인 및 SLA 만족도 관련 통계 수집 및 달성
- 노드 수준 및 클라우드 수준의 SLA 보장
 - ✓ 노드 수준: 하이퍼바이저 계층에서 SLA를 보장
 - ✓ 클라우드 수준: 노드 별 서비스 용량 계산 및 작업 할당

❑ 현재 진행 상황

- 현재, 노드 수준 Stella 스케줄러 구현 및 공개 (CPU-네트워크, CPU-스토리지)
- 클라우드 수준에서 통합 스케줄링 연구 진행 중

Stella 프로젝트 단계별 목표

1단계 (2015~2018)

- 노드 수준 CPU, 네트워크, 스토리지 개별 SLA 보장 기법 개발
- 노드 수준 CPU, 네트워크, 스토리지 통합 SLA 보장 스케줄러 개발

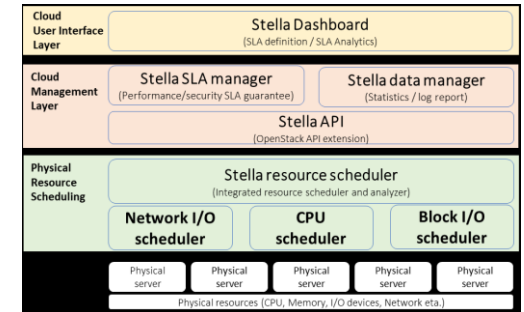
1단계 목표: 성능 SLA
통합 보장 시스템 SW 기반 기술

2단계 (2019~2022)

- 클라우드 수준 자원 스케줄링 기법 개발
- 클라우드 수준 작업 할당 및 관리 기법 개발
- 클라우드 수준 SLA 지원에 따른 클라우드 서비스 성능/품질 저하 방지 방안 연구

2단계 목표: SLA 통합 보장
클라우드 플랫폼 기술

최종 (2022)

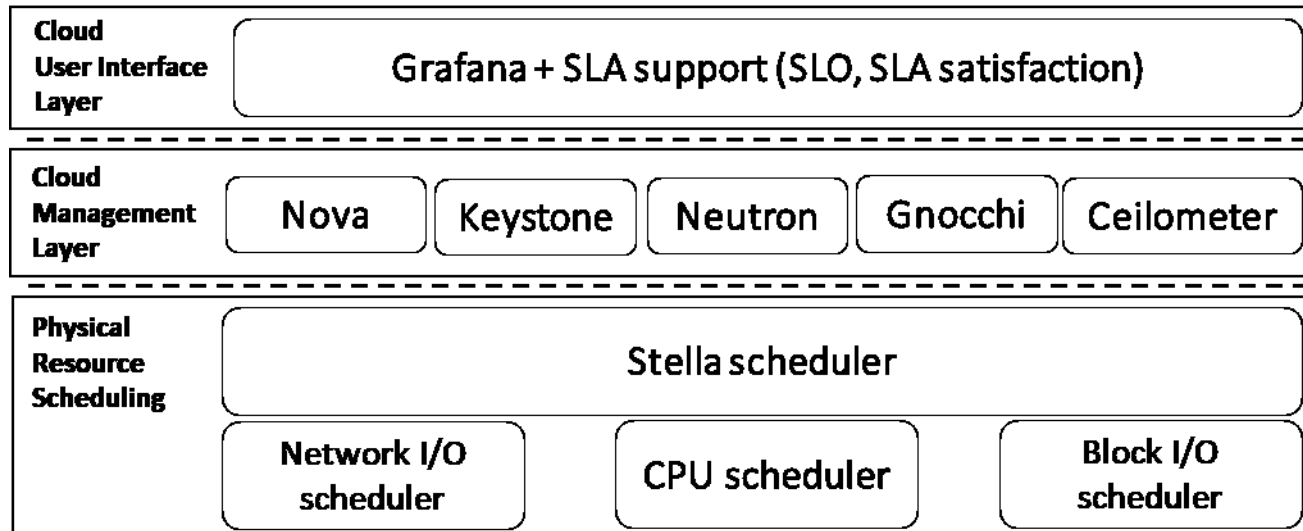


최종 목표:
성능 SLA 보장
차세대 클라우드 시스템 플랫폼

결과물 : 오픈스택 기반 컴포넌트

- 성능 SLA 보장 스케줄러
- 통합 스케줄링 프레임워크
- SLA 보장 서비스 및 노드 관리 기법

Stella 클라우드 아키텍처



☑ Stella 스케줄러

- 노드 수준 CPU, 네트워크, 블록I/O 자원의 통합 스케줄링

☑ 오픈스택 컴포넌트

- 클라우드 수준의 VM 할당 및 관리, 자원 활용 및 성능 모니터링/보고, 인증
- 모니터링 관련 컴포넌트
 - ✓ Ceilometer(모니터링), Gnocchi(통계 수집), Grafana(통계표현 대시보드)

Stella 스케줄러

✓ SLA 매니저(SLA Manager)

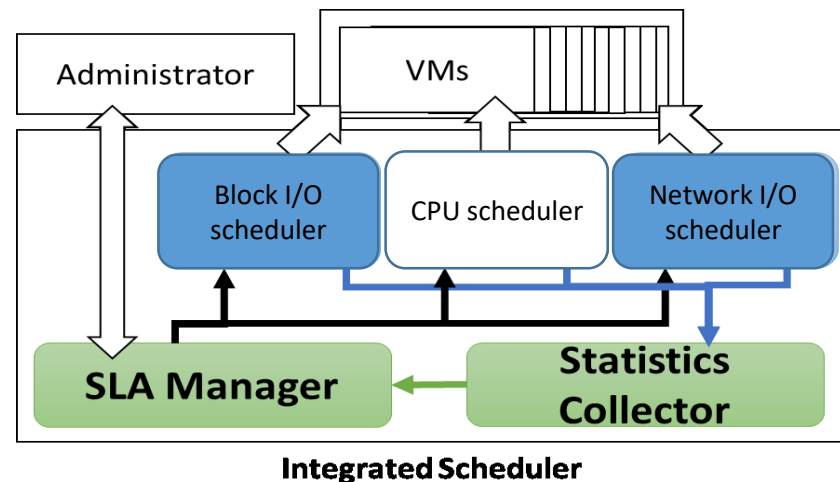
- 주어진 SLO를 스케줄링 파라미터로 변환
e.g., 100MB/s (스토리지 대역폭)를 스케줄링 파라미터로 변환 (CPU quota, I/O 요청 시간 등을 조절)
- 관리자에게 SLA 만족도 보고
- SLA 만족도 달성을 위해 스케줄링 파라미터 조절

✓ SLA를 고려한 자원 스케줄러

- 스토리지 I/O 스케줄러 및 네트워크 I/O 스케줄러 통합

✓ 통계 수집기(Statistics collector)

- 스케줄러로부터 통계 데이터 수집
- SLA 매니저에게 통계정보 전송



Stella 스케줄러의 오픈스택 확장

- 현재 **Stella** 스케줄러는 클라우드 노드 수준으로 동작
- 클라우드 수준의 **SLA** 관련 통계 모니터링 대상과 과정의 분석이 필요
 - 목적: Stella 스케줄러와 통합을 위한 오픈스택 확장
 - 모니터링 대상
 - ✓ 자원 사용량: I/O 대역폭, CPU 사용량 등을 수집
 - ✓ SLA 파라미터: SLA 목표치 설정 및 SLA 만족도 수집
 - 모니터링 과정
 - ✓ 수집된 통계를 대시보드 까지 전달하는 경로
 - ✓ 관련 오픈소스 및 오픈스택 컴포넌트 (Pike 버전 기준)
 - Grafana: 통계 정보를 표현하는 웹 기반 대시보드
 - Ceilometer: 오픈스택 서비스로 부터 통계 정보 모니터링
 - Gnocchi: Ceilometer가 모니터링 한 정보 수집 및 관리
- **Stella** 스케줄러를 위한 **SLA** 목표치 전달 인터페이스 설계 및 구현 예정
 - 분석 내용을 기반으로 인터페이스 설계 및 구현
 - ✓ 대시보드를 통해 사용자 또는 관리자가 SLA 정의
 - ✓ 정의한 SLA 목표치를 Stella 스케줄러에 전달
- 본 발표에서는 모니터링 과정에 초점

Ceilometer-Gnocchi 전달 과정 분석

Gnocchi 분석의 필요성

✓ Gnocchi를 통해 오픈스택과 연동

- 오픈소스 시계열 데이터 수집 서비스
 - ✓ 시계열 데이터: 시간 순으로 나열된 데이터
- Gnocchi는 Ceilometer의 성능 문제를 해결하기 위해 제안
 - ✓ Ceilometer는 기존 오픈스택에서 통계 수집기능 제공
 - ✓ Ceilometer는 정보를 조회할 때 성능 저하 발생
 - 특히 데이터를 탐색할 때 $O(n)$ 복잡도를 가짐
(데이터의 숫자가 늘어나면 탐색 시간이 선형적으로 증가)
 - 노드의 규모가 늘어날 수록 성능 문제가 커짐

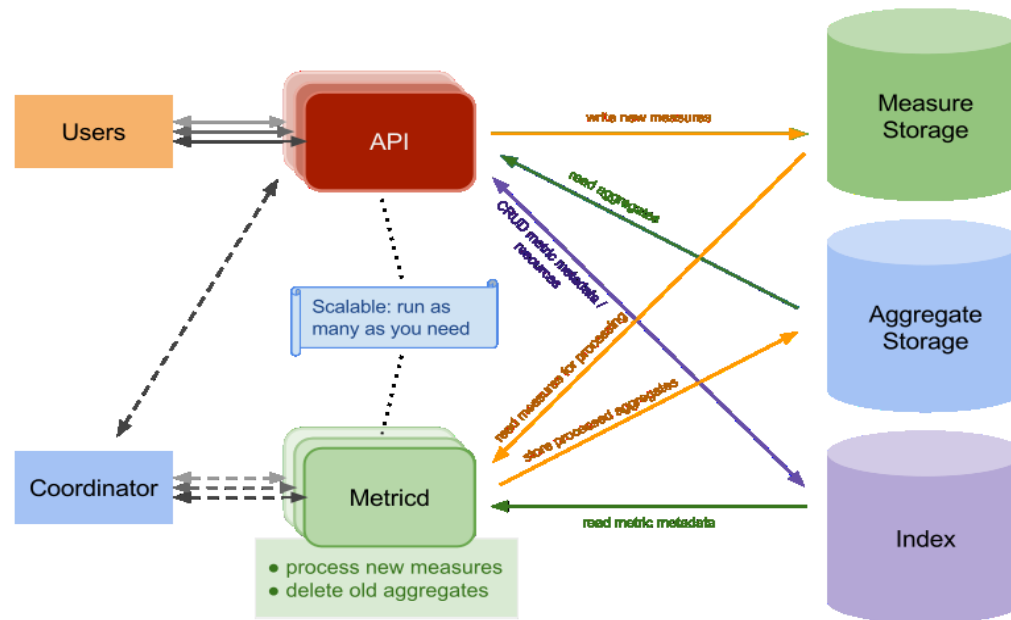
✓ Gnocchi는 시간 복잡도를 $O(1)$ 로 개선

- 통계 정보를 시계열순으로 정렬해서 저장
 - ✓ Measure: gnocchi가 입력 받은 데이터 기록
 - ✓ Aggregated: Measure의 데이터를 기반으로 정렬된 정보 기록
- 인덱스를 통해 빠른 접근 제공
 - ✓ Index: Gnocchi를 위한 메타데이터, 클라우드 리소스 정의 및 정책 저장

Gnocchi의 구성 요소

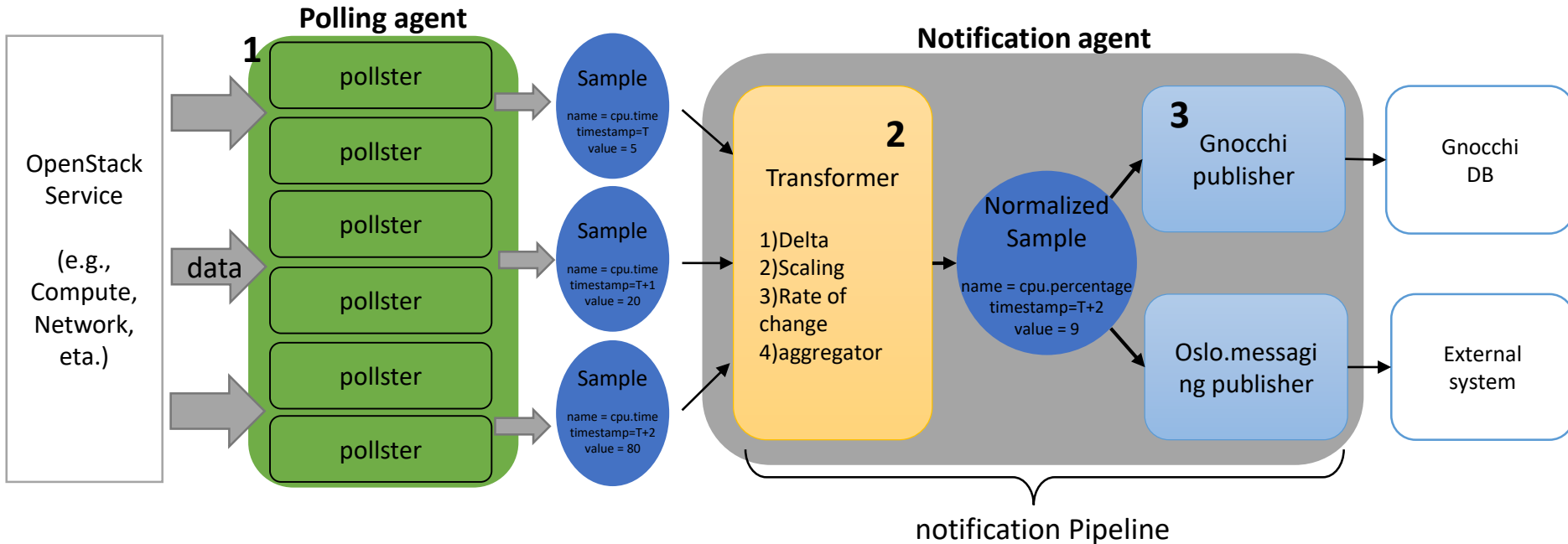
❑ Gnocchi 구성 컴포넌트: gnocchi-api, gnocchi-statsd, gnocchi-metric

- API: Gnocchi 연산 및 데이터 통계 수집을 위한 REST API 정의
- Statsd: 통계 데이터 수집
- Metricd: 수집된 데이터를 기반으로 명령 처리 수행



Gnocchi 아키텍처

Ceilometer-Gnocchi 전달 과정 요약

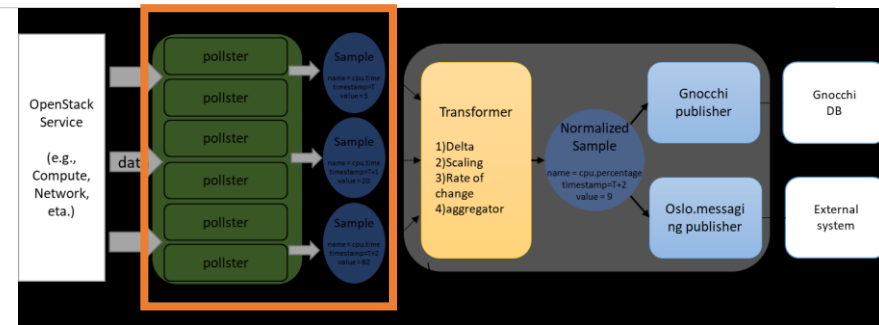


- **Polling agent:** 오픈스택 서비스로 부터 통계를 수집
 - 주기적인 통계 수집 API 호출
 - Pollster는 서비스의 통계정보 수집을 위한 플러그인 (서비스에 맞는 API 호출 및 데이터 형식 지정)
 - 수집데이터는 sample이라고 불림
- **Notification agent:** 수집된 데이터를 기반으로 Ceilometer의 이벤트 또는 샘플로 변환
 - Pipeline: 수집된 데이터를 외부로 전달하기 위한 처리 절차
 - Transformer: 수집된 데이터를 모아 정규화 및 통합
 - Publisher: 수집된 데이터를 외부 형식에 맞게 변환 및 전송

과정 1. Polling agent

❑ Polling agent는 sample을 수집

- 오픈스택 서비스마다 독립적인 pollster가 동작
 - ✓ Pollster는 특정서비스의 API 호출
- Sample 데이터 형식으로 통계 정보 수집
 - ✓ 호출된 API를 통해 통계 정보 수집
 - ✓ Sample은 키, 값으로 구성된 데이터의 집합



run -> start_polling_tasks -> for interval.pollingtask (반복)

Class PollingTask(object)

```
def poll_and_notify(self):
```

```
    """Polling sample and notify."""
```

```
    for source_name, pollsters in iter_random(
        self.pollster_matches.items()):
        for pollster in iter_random(pollsters):
```

사용할 pollster들을 추가하는 과정
(pollster config는 compute/pollsters/분류.py)

```
        self.resources[key].get(discovery_cache))
        for x in candidate_res:
            if x not in history:
                history.append(x)
            if x not in black_res:
                polling_resources.append(x)
        poll_history[pollster.name] = history
```

History에 resource 추가하고,
각 pollster에 맞는
Polling_resource들을 append

```
        polling_timestamp = timeutils.utcnow().isoformat()
        samples = pollster.obj.get_samples(
            manager=self.manager,
            cache=cache,
            resources=polling_resources
        )
```

Timestamp찍은 후,
샘플 데이터를 수집하는 함수

```
        for sample in samples:
```

받은 샘플을 notification에 전송

```
            self._send_notification([sample_dict])
```

Polling/manager.py

```
def setup_polling_tasks(self):
```

```
    polling_tasks = {}
```

```
    for source in self.polling_manager.sources:
```

```
        for pollster in self.extensions:
```

```
            if source.support_meter(pollster.name):
```

```
                polling_task = polling_tasks.get(source.get_interval())
```

```
                if not polling_task:
```

```
                    polling_task = PollingTask(self)
```

```
                    polling_tasks[source.get_interval()] = polling_task
```

```
                    polling_task.add(pollster, source)
```

```
    return polling_tasks
```

compute/pollsters/_init_.py

```
def get_samples(self, manager, cache, resources):
```

```
    self._inspection_duration = self._record_poll_time()
```

```
    for instance in resources:
```

```
        try:
```

```
            polled_time, result = self._inspect_cached(
                cache, instance, self._inspection_duration)
```

```
            if not result:
```

```
                continue
```

```
            for stats in self.aggregate_method(result):
```

```
                yield self._stats_to_sample(instance, stats, polled_time)
```

```
def _send_notification(self, samples):
```

```
    self.manager.notifier.sample(
```

```
        {},
```

```
        'telemetry.polling',
```

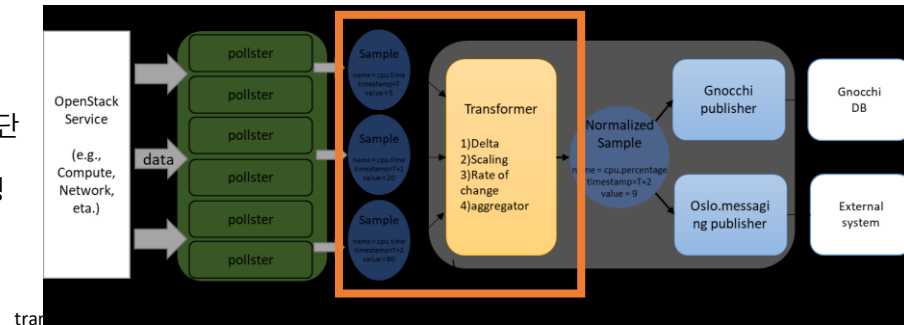
```
        {'samples': samples}
```

```
)
```

과정 2-1. Transformer의 sample 처리

Transformer는 sample 수집 및 처리

- Pollster로 부터 수집한 sample을 가공하는 과정을 수행
 - ✓ Accumulator: Sample의 버퍼, 일정 수 이상이 모이면 다음 단계로 넘김
 - ✓ ArithmeticTransformer: sample 데이터의 수학 계산을 수행
- 가공한 sample은 정규화된 데이터 형식으로 변환 수행
 - ✓ BaseConversionTransformer를 통해 호출
 - ✓ 세부 내용은 과정 2-2(다음 장)에 계속



```
class TransformerAccumulator(transformer.TransformerBase):
    def __init__(self, size=1, **kwargs):
        if size >= 1:
            self.samples = []
            self.size = size
        super(TransformerAccumulator, self).__init__(**kwargs)
    def handle_sample(self, sample):
        if self.size >= 1:
            self.samples.append(sample)
        else:
            return sample
```

transformer/accumulator.py

```
class ArithmeticTransformer(transformer.TransformerBase):
    def _calculate(self, resource_id):
        """Evaluate the expression and return a new sample if successful."""
        ns_dict = dict((m, s.as_dict()) for m, s
                        in six.iteritems(self.cache[resource_id]))
        ns = transformer.Namespace(ns_dict)
        try:
            new_volume = eval(self.expr_escaped, {}, ns)
            if math.isnan(new_volume):
                raise ArithmeticError(_('Expression evaluated to '
                                      'a NaN value!'))
            reference_sample = self.cache[resource_id][self.reference_meter]
            return sample.Sample(
                name=self.target.get('name', reference_sample.name),
                unit=self.target.get('unit', reference_sample.unit),
                type=self.target.get('type', reference_sample.type),
                volume=float(new_volume),
                user_id=reference_sample.user_id,
                project_id=reference_sample.project_id,
                resource_id=reference_sample.resource_id,
                timestamp=self.latest_timestamp,
                resource_metadata=reference_sample.resource_metadata
            )
```

transformer/arithmetic.py

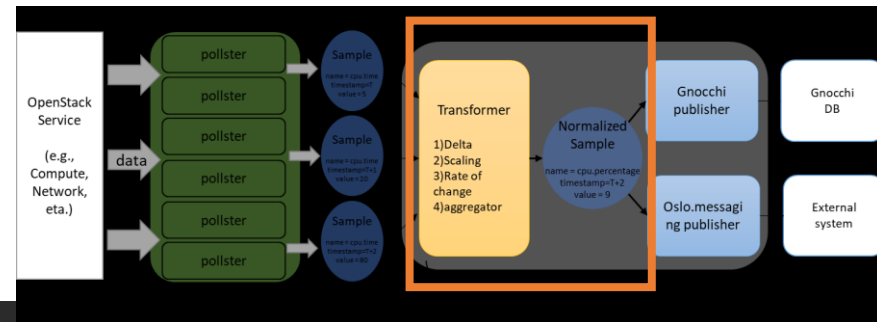
```
class BaseConversionTransformer(transformer.TransformerBase):
    def __init__(self, source=None, target=None, **kwargs):
        """Initialize transformer with configured parameters.
        :param source: dict containing source sample unit
        :param target: dict containing target sample name, type,
            unit and scaling factor (a missing value
            connotes no change)
        """
        self.source = source or {}
        self.target = target or {}
        super(BaseConversionTransformer, self).__init__(**kwargs)
    def _map(self, s, attr):
        """Apply the name or unit mapping if configured."""
        mapped = None
        from_ = self.source.get('map_from')
        to_ = self.target.get('map_to')
        if from_ and to_:
            if from_.get(attr) and to_.get(attr):
                try:
                    mapped = re.sub(from_[attr], to_[attr], getattr(s, attr))
                except Exception:
                    pass
        return mapped or self.target.get(attr, getattr(s, attr))
```

transformer/conversions.py

과정 2-2. Conversion transformer 세부 내용

수집한 sample을 정규화 및 통합

- DeltaTransformer: Sample간 값의 차이를 기준으로 통합 수행
- ScalingTransformer: Sample 간 값의 단위 차이를 통합
e.g., 나노초, 밀리초 단위 값을 통합
- RateOfChangeTransformer: Sample의 rate를 변경 해서 통합
e.g., CPU time을 CPU 사용량



```

class DeltaTransformer(BaseConversionTransformer):
    def _convert(self, s, delta):
        """Transform the appropriate sample fields.
        return sample.Sample(
            name=self._map(s, 'name'),
            unit=s.unit,
            type=sample.TYPE_DELTA,
            volume=delta,
            user_id=s.user_id,
            project_id=s.project_id,
            resource_id=s.resource_id,
            timestamp=s.timestamp,
            resource_metadata=s.resource_metadata
        )
  
```

```

    def handle_sample(self, s):
        """Handle a sample, converting if necessary.
        key = s.name + s.resource_id
        prev = self.cache.get(key)
        timestamp = timeutils.parse_isotime(s.timestamp)
        self.cache[key] = (s.volume, timestamp)

        if prev:
            LOG.warning('Dropping sample with no pred
            s = None
        return s
  
```

```

class ScalingTransformer(BaseConversionTransformer):
    def _convert(self, s, growth=1):
        """Transform the appropriate sample fields.
        volume = self._scale(s) + growth
        return sample.Sample(
            name=self._map(s, 'name'),
            unit=self._map(s, 'unit'),
            type=self.target.get('type', s.type),
            volume=min(volume, self.max) if self.max else
            user_id=s.user_id,
            project_id=s.project_id,
            resource_id=s.resource_id,
            timestamp=s.timestamp,
            resource_metadata=s.resource_metadata
        )
  
```

```

    def _scale(self, s):
        """Apply the scaling factor.
        Either a straight multiplicative factor or else a string to be eval'd.
        ns = transformer.Namespace(s.as_dict())

        scale = self.scale
        return ((eval(scale, {}, ns) if isinstance(scale, six.string_types)
                else s.volume * scale) if scale else s.volume)
  
```

Handle_sample은 DeltaTransformer와 유사

```

class RateOfChangeTransformer(ScalingTransformer):
  
```

```

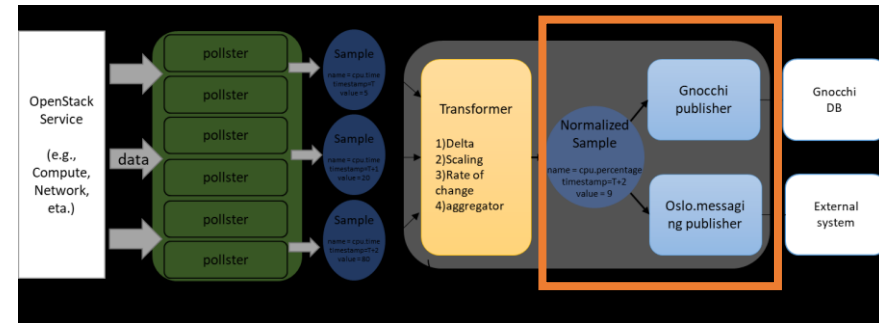
    def handle_sample(self, s):
        """Handle a sample, converting if necessary.
        LOG.debug('handling sample %s', s)
        key = s.name + s.resource_id
        prev = self.cache.get(key)
        timestamp = timeutils.parse_isotime(s.timestamp)
        self.cache[key] = (s.volume, timestamp, s.monotonic)

        if prev:
            LOG.warning('dropping sample with no pred
            s = None
        return s
  
```

과정 3. Gnocchi Publisher

❑ Publisher는 데이터를 외부 저장소로 전송

- 정규화 및 통합된 sample을 외부 저장소 형식에 맞게 변경
e.g., Gnocchi 데이터 형식
- 변경한 데이터를 전송



```
class GnocchiPublisher(publisher.ConfigPublisherBase):
```

```
    def publish_samples(self, data):
```

```
        for sample in samples_of_resource:
```

```
            metric_name = sample.name
```

```
            rd = self.metric_map.get(metric_name)
```

```
            if rd is None:...
```

publisher/gnocchi.py

Gnocchi에게 전달을 위해 sample 를
metric data 형식으로 전환

만약 수집한 sample의 resource_id가
gnocchi에 등록되어 있지 않다면,
gnocchi data 배열에 새정보를 등록

```
def publish_data(self, samples):
    if not isinstance(samples, list):
        samples = [samples]
    supported = [s for s in samples if self.supported(s)
                  and self._validate_volume(s)]
    self.sink.publish_samples(supported)
```

사용자가 설정한
publisher에서 실제 사용
가능한 data들이 return
되면, 마지막으로 외부에
publishing한다.

```
def serializer(self, sample):
    return publisher_utils.meter_message_from_counter(
        sample, self.conf.publisher.telemetry_secret)
```

```
def supported(self, sample):
    return self.source.support_meter(sample.name)
```

pipeline/sample.py

데이터를 전송하기 위해
시리얼라이즈(바이트화)

```
if resource_id not in gnocchi_data:
    gnocchi_data[resource_id] = {
        'resource_type': rd.cfg['resource_type'],
        'resource': {'id': resource_id,
                     "user_id": sample.user_id,
                     "project_id": sample.project_id,
                     "metrics": rd.metrics}}
```

```
gnocchi_data[resource_id].setdefault(
    "resource_extra", {}).update(rd.sample_attributes(sample))
measures.setdefault(resource_id, {}).setdefault(
    metric_name,
    {"measures": [],
     "archive_policy_name":
        rd.metrics[metric_name]["archive_policy_name"],
     "unit": sample.unit})["measures"].append(
    {'timestamp': sample.timestamp,
     'value': sample.volume})
```

publisher/gnocchi.py

```
# TODO(gordc): unit should really be part of metric definition
gnocchi_data[resource_id]['resource']['metrics'][
    metric_name]['unit'] = sample.unit
```

Grafana-Gnocchi 연동

Grafana 설치 및 환경 구성 요약

✓ Grafana 패키지 설치

- 우분투 16.04 환경에서 진행

```
wget https://s3-us-west-2.amazonaws.com/grafana-releases/release/grafana_5.1.4_amd64.deb  
sudo apt-get install -y adduser libfontconfig  
sudo dpkg -i grafana_5.1.4_amd64.deb
```

Ubuntu 환경에서 Grafana 다운 로드 및 설치

```
sudo service grafana-server start  
sudo update-rc.d grafana-server defaults
```

Grafana 구동 및 서비스 등록

✓ Keystone 및 Gnocchi 설정

- Keystone의 URL 설정
 - ✓ Keystone: 오픈스택 환경에서 사용자 및 서비스의 인증 관리 수행
 - ✓ Gnocchi가 다른 서비스와 동작하기 위해 필요

✓ Grafana 및 Gnocchi 연동

- Grafana에서 데이터 저장소를 Gnocchi 로 설정 (Web GUI 에서 진행)
- Gnocchi 접근 권한을 위한 인증 설정 진행

Keystone 및 Gnocchi 설정

Keystone 설정

```
[cors]
#
# From oslo.middleware
#
# Indicate whether this resource may be shared with the domain received in the
# requests "origin" header. Format: "<protocol>://<host>[:<port>]", no trailing
# slash. Example: https://horizon.example.com (list value)
#allowed_origin = http://163.152.20.141:8041
allowed_origin = http://163.152.20.141:3000
# Indicate that the actual request can include user credentials (boolean value)
allow_credentials = true
```

Keystone.conf

Gnocch에서 keystone API 호출을 위한 IP
및 포트 설정 (URL 및 포트 번호를
Gnocchi 설정에 등록)

Keystone-paste.ini

```
[filter:cors]
allowed_origin = http://163.152.20.141:3000
use = egg:oslo.middleware#cors
oslo_config_project = keystone
```

Gnocchi 설정

```
[cors]
#
# From oslo.middleware.cors
#
# Indicate whether this resource may be shared with the domain received in the
# requests "origin" header. Format: "<protocol>://<host>[:<port>]", no trailing
# slash. Example: https://horizon.example.com (list value)
#allowed_origin = http://163.152.20.141:8041
allowed_origin = http://163.152.20.141:3000
# Indicate that the actual request can include user credentials (boolean value)
allow_credentials = true
```

gnocchi.conf

Keystone의 URL 기록

Grafana Gnocchi 연동

The screenshot shows the Grafana 'Data Sources / Gnocchi Source' configuration page. The 'Type' is set to 'Gnocchi'. Under the 'HTTP' section, the 'URL' is 'http://163.152.20.141:35357' and 'Access' is 'Browser'. Under the 'Auth' section, 'Basic Auth' is selected. Below this, a note states: 'When "direct" access is used, Gnocchi and Keystone MUST have CORS configured correctly on the server side.' In the 'Gnocchi Details' section, 'Auth Mode' is 'keystone'. A table for 'The Keystone URL is expected in Http settings' contains the following fields: Domain (default), Project (admin), User (admin), and Password (ADMIN_PASS).

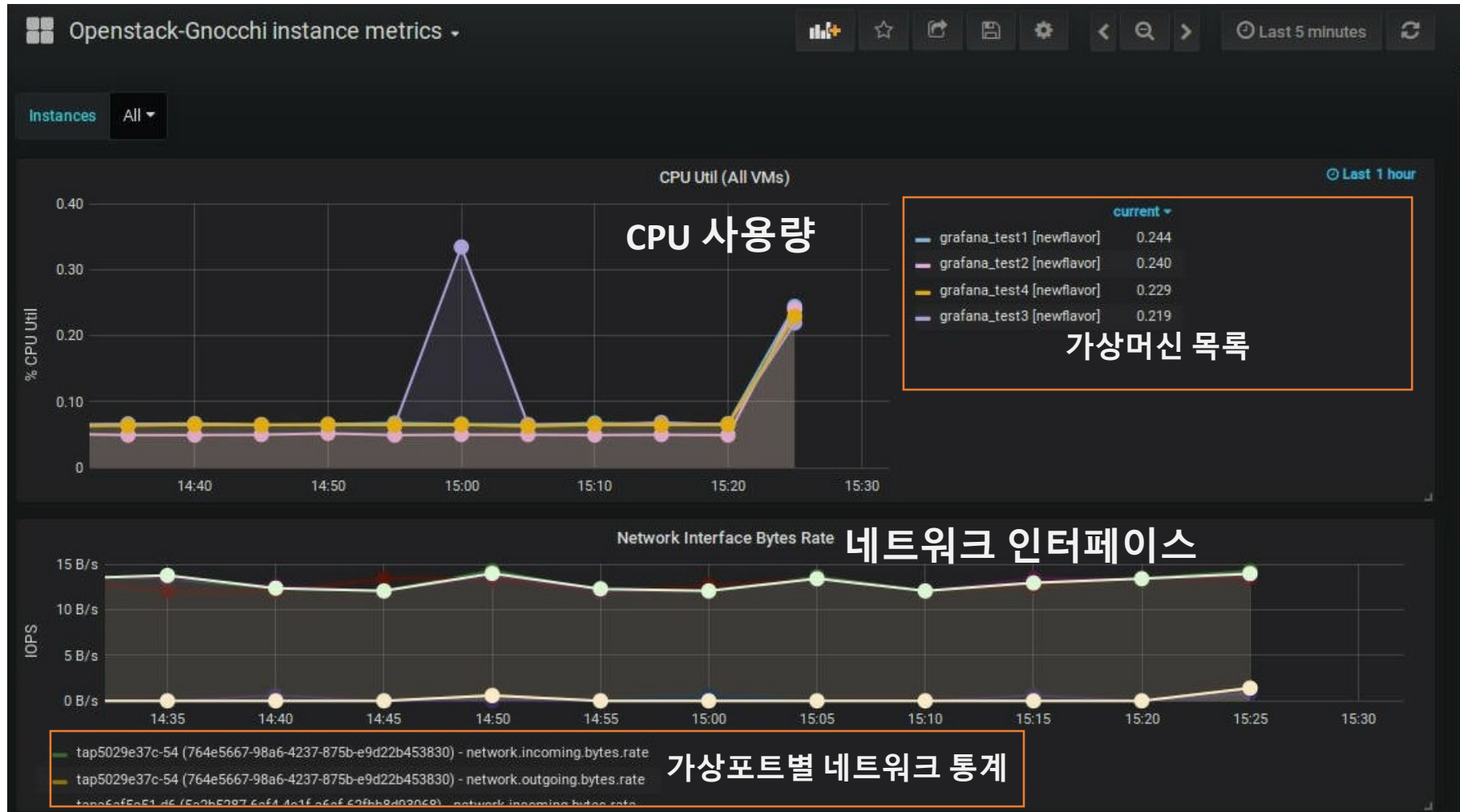
Field	Value
Domain	default
Project	admin
User	admin
Password	ADMIN_PASS

데이터 종류를 gnocchi
선택

오픈스택 Keystone
서비스의 IP 및 포트

keystone 접속을 위한 로그인
정보
(편의를 위해 관리자 정보 기입)

Grafana 대시보드 구성 예시 (CPU 및 네트워크)



Stella 프로젝트 사이트 및 코드 공개

Stella 프로젝트 사이트

☑ Stella 프로젝트 홈페이지(<http://stella.korea.ac.kr>)

- 프로젝트 소개 및 관련 기술 문서 업로드
- 블록 스토리지 및 네트워크 스케줄러 관련 자료 제공
- 클라우드 구축 현황 정보

☑ Stella Github 저장소 (<https://github.com/KUoslab>)

- Stella 개발 코드 제공
(현재 Stella 스케줄러 코드 및 설치 문서 제공)

☑ KU.Stella 런치패드 (<https://launchpad.net/ku.stella>)

- 프로젝트관리에 사용
(Github 저장소 미리 제공, 버그 리포팅)
- 오픈스택 관련 컴포넌트 제공 예정
(Stella 스케줄러 인터페이스 공개 등)

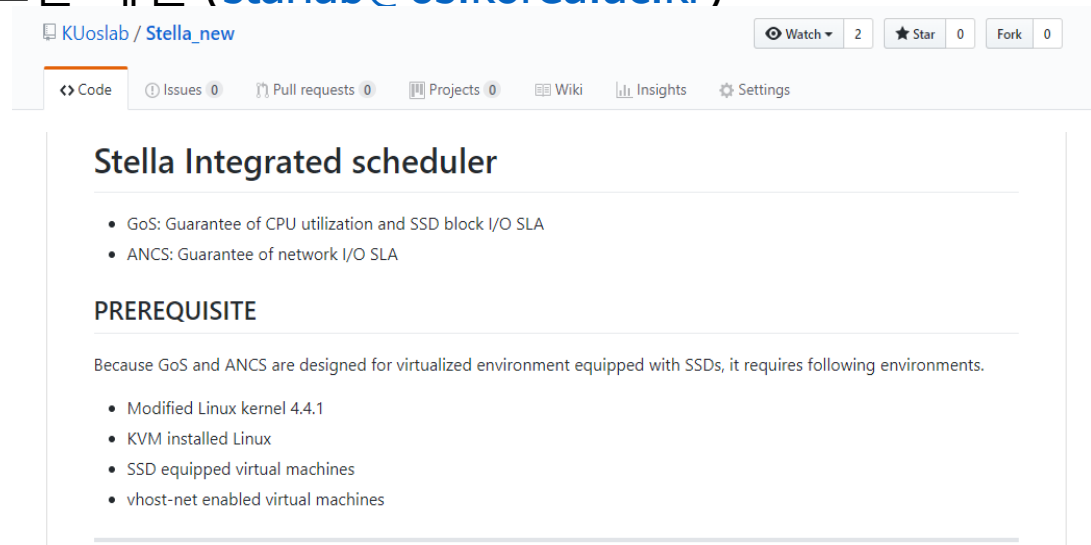
Stella 스케줄러 테스트 및 리포팅

☑ Stella 스케줄러 테스트 방법

- Stella Github 저장소 접속
(https://github.com/KUoslab/Stella_new)
- README 파일을 통해 설치 정보 제공
(https://github.com/KUoslab/Stella_new/blob/master/README.md)

☑ 사용 코멘트 및 버그 리포팅

- KU.Stella 런치패드 또는 메일 (starlab@os.korea.ac.kr)



Github 저장소 접속 화면

감사합니다



고려대학교