NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

MIGUEL FERREIRA COELHO MORAIS ANDRADE

Licenciado em Engenharia Informática

ANÁLISE DE DESEMPENHO EM AMBIENTES CLOUD

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa

setembro, 2022

DEPARTAMENTO DE INFORMÁTICA

ANÁLISE DE DESEMPENHO EM AMBIENTES CLOUD

MIGUEL FERREIRA COELHO MORAIS ANDRADE

Licenciado em Engenharia Informática

Orientador: André Alexandre Santos Simões,
Especialista, Crossjoin Solutions Lda

Coorientador: Maria Cecília Gomes,
Professora Auxiliar, FCT-NOVA

Coorientador: Nuno Manuel Ribeiro Preguiça,
Professor Associado com Agregação, FCT-NOVA

Júri:

Presidente: Teresa Isabel Lopes Romão,
Professora Associada, FCT-NOVA

Arguente: Vítor Manuel Alves Duarte,
Professor Auxiliar, FCT-NOVA

Orientador: André Alexandre Santos Simões,
Especialista, Crossjoin Solutions Lda

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
setembro, 2022

Análise de desempenho em ambientes cloud

Agradecimentos

Agradeço em especial ao meu orientador, André Simões por toda a ajuda, apoio e orientação nos nove meses de realização desta dissertação. Também agradeço aos professores Cecília Gomes e Nuno Preguiça por me orientarem e darem feedback na escrita desta dissertação.

RESUMO

Existem várias razões para uma empresa optar por distribuir os seus serviços por vários *cloud providers*, tais como evitar o *vendor lock-in* ou minimizar custos. Como tal, é necessária uma ferramenta que permita, a um administrador de sistema, conseguir gerir e monitorizar o desempenho de cada aplicação e/ou serviço existentes nas várias clouds, sem necessitar de aceder a cada plataforma de monitorização disponibilizada por cada *cloud provider.*

Tal visão integrada constitui um problema pois cada *cloud provider* utiliza uma representação diferente para todos os dados métricos que disponibiliza, organiza os seus recursos de maneiras distintas, sendo necessário um modelo que seja genérico para acomodar as possíveis representações e disponibilizam APIs diferentes para a obtenção destes dados.

Com a realização deste trabalho foi possível extrair dados métricos de três *cloud providers* (Amazon Web Services, Google Cloud Platform e Microsoft Azure), guardar estes dados de forma eficiente e a sua apresentação.

Palavras chave: Computação Cloud, Gestão de Desempenho de Aplicações, Modelação, Bases de Dados

Abstract

There are several reasons for a business to choose to distribute its services across multiple cloud providers, such as to avoid vendor lock-in or minimize costs. As such, a tool is needed that allows a system administrator to manage and monitor the performance of each application and/or service running in various clouds without having to access each monitoring platform provided by the cloud provider.
This becomes a problem because each cloud provider uses a different representation for the metric data it provides, organizes its resources in different ways, requiring a model that is generic enough to accommodate the possible representations and uses a different API to get all that data.
At the end of this work, it was possible to extract metric data from the three main cloud providers (Amazon Web Services, Google Cloud Platform and Microsoft Azure), store this data efficiently and its presentation.

Keywords: Cloud Computing, Applicaton Performance Management, Modelling, Databases

ÍNDICE

Índice de Figuras

Índice de Tabelas

GLOSSÁRIO

Cloud Uma rede global de servidores em todo o mundo interligados entre si e que devem funcionar como um ecossistema único

Timestamp Uma sequência de caracteres que identifica quando um evento ocorreu, normalmente a sua data e hora

SIGLAS

API	Application Programming Interface
APM	Application Performance Management
AWS	Amazon Web Services
CPU	Central Processing Unit
DB	Database
EC2	Elastic Compute Cloud
ELB	Elastic Load Balancer
GCP	Google Cloud Platform
JSON	JavaScript Object Notation
OLAP	Online Analytical Processing
SDK	Software Development Kit
SQL	Structured Query Language
TSDB	Time series Database
XML	eXtensible Markup Language

1

INTRODUÇÃO

1.1 Contexto e motivação

Existem empresas como a Crossjoin, que operam em áreas como a otimização continua em IT, e que são utilizadas simultaneamente por milhares de utilizadores espalhados por todo o mundo. Como tal, é necessário assegurar que todos estes serviços dispõem de disponibilidades elevadíssimas e velocidades rápidas de resposta aos utilizadores. Quando um utilizador tem de ficar à espera que um serviço ou website estejam disponíveis, ele tem tendência a sair ou desistir da plataforma o que poderá trazer perdas de receitas a essas empresas. Muitos destes serviços assentam sobre a infraestrutura cloud, beneficiando de podermos ter a informação replicada por vários servidores em diversas partes do mundo [1]. Tal permite adicionar redundância no caso de perdas de dados como também melhorar tempos de resposta, pois um utilizador irá estar a aceder a informação mais próxima da sua localização. No entanto, continua a ser importante a monitorização do desempenho de um serviço como um todo de modo a garantir a sua operacionalidade, onde se inclui a qualidade de serviço percebida pelos utilizadores.

A monitorização de um serviço que utilize apenas um *cloud provider* é relativamente fácil pois é possível utilizar as ferramentas que este disponibiliza. O mesmo já não se pode dizer para serviços multi-cloud, isto é, que utilizam vários *cloud providers* em simultâneo [31], o que torna a monitorização destes serviços mais difícil. Existem várias razões para distribuir partes de uma aplicação por vários *providers* diferentes, tais como a redução de custos através de utilizar um *provider* mais barato para um tipo de serviço, evitar quebras de serviço de um *provider*, mudando um serviço para outro servidor disponível, ou evitar *vendor lock-in* (explicado na secção 2.1). No entanto, é difícil disponibilizar uma solução integrada de avaliação de desempenho em ambientes multi-cloud.

1.2 Problema e objetivos

Cada *cloud provider* utiliza maneiras heterogéneas de representar e disponibilizar os dados de monitorização aos administradores de sistema, seja em termos de estruturas internas e modelo de dados como de funções das várias APIs. É então necessária uma ferramenta que generalize os dados dos *cloud providers* e os agrupe numa única plataforma, de forma a que uma empresa consiga visualizar o desempenho dos seus serviços sem ter de usar várias ferramentas diferentes.

O objetivo deste trabalho é arranjar um modelo genérico que permita analisar a estrutura e as métricas em ambientes multi-cloud e permitir a extração desses dados.

1.3 Abordagem e contribuições

Para realizar este trabalho foi necessário desenvolver um modelo que seja genérico o suficiente para representar os vários recursos nas três clouds a estudar e guardar as métricas relevantes numa base de dados NoSQL que seja ideal para tratar de grandes volumes de dados métricos.

Com a conclusão deste trabalho produziu-se um protótipo funcional de extração de métricas para os principais *cloud providers*.

1.4 Organização do documento

No Capítulo 1 apresentámos o contexto e motivação para este trabalho bem como o seu problema e objetivos.

No Capítulo 2 é apresentado um estudo das tecnologias existentes de momento que estão relacionadas com este trabalho, como Cloud Computing, Gestão de desempenho de aplicações e Bases de Dados relevantes para captura dos dados de performance.

No Capítulo 3 é proposto uma solução para este problema.

No Capítulo 4 é apresentada a forma como foi implementada esta solução.

No Capítulo 5 é feita uma validação dos resultados obtidos pela aplicação desenvolvida comparando-os com os disponibilizados pelas plataformas Web dos *cloud providers*.

No Capítulo 6 é feita uma conclusão deste trabalho, incluindo o trabalho que foi feito, alguns problemas encontrados no desenvolvimento do mesmo e trabalho futuro.

2

Estado da arte

Este capítulo descreve as dimensões relacionadas com o trabalho a desenvolver. Sendo um trabalho sobre análise de desempenho em ambientes cloud, na primeira secção descreve-se o que é Cloud Computing, a sua origem e algumas vantagens e desvantagens, bem como diferentes tipos de clouds e serviços disponibilizados pelos *cloud providers*. Na segunda secção é feita uma introdução ao que é a Gestão do Desempenho de Aplicações (*Application Performance Management*), o quão importante isto é para empresas e como é feito. Na terceira secção descreve-se o porquê de se distribuírem sistemas por vários *cloud providers* diferentes (*multi-cloud*) e o porquê de isto ser um problema mostrando como os três principais *cloud providers* organizam os seus recursos internamente e como obter as suas métricas para posteriormente se fazer uma análise de desempenho. Como irá ser necessário guardar a informação sobre o que constituiu um serviço (recursos) e os dados sobre as suas métricas de desempenho, serão utilizadas bases de dados. Na última secção é feita uma introdução a bases de dados relacionais (também chamadas de bases de dados SQL) e bases de dados NoSQL (*not only* SQL). Como os recursos de um serviço têm uma estrutura hierárquica, como será explorado na secção três, é então também explicado base de dados em grafos, visto ser a melhor forma de representar esta estrutura. De forma semelhante, como os dados métricos são usualmente um par data/hora (*timestamp*)-valor, é também feita uma introdução a base de dados de séries temporais.

2.1 Introdução a Cloud Computing

Com os avanços que foram feitos nas últimas décadas em áreas como redes informáticas, começou-se a perceber que operações de computação eram mais eficientes quando feitas em grandes servidores de computação ou invés de em computadores pessoais. Quando computamos algo num centro de dados (*data center*) em vez de no nosso computador pessoal,

falamos em *network-centric compute* e *network-centric content* [1]. Algumas são vantagens deste paradigma são:

- Os recursos são partilhados e podem melhor ser agregados para suportar aplicações que requerem grande capacidade de processamento ou necessidade de armazenar elevados volumes de dados.
- A partilha da informação facilita a colaboração entre entidades.
- Reduz os custos, visto ser possível a partilha do hardware com outras a aplicações de outros utilizadores que estejam a executar na mesma máquina física.
- Consegue acomodar grandes picos de utilização.

A computação cloud (*Cloud Computing*) consiste no fornecimento de serviços informáticos como bases de dados, armazenamento, software, entre outros, em centros de dados dispersos geograficamente. O Cloud Computing baseia-se no paradigma *network-centric compute* e *network-centric content* [1].

Algumas das vantagens na utilização de Cloud Computing consistem em:

- Utilização da Internet para oferecer serviços elásticos. Isto refere-se à adaptação dinâmica à carga de trabalho atual do serviço de forma a ser possível aumentar ou diminuir recursos consoante a sua utilização.
- Estes serviços são medidos e o utilizador só paga o que utiliza.
- Os serviços estão acessíveis/disponíveis para serem utilizados quando necessário.
- Dada a replicação por diferentes centros de dados, os dados da aplicação podem estar mais próximos do utilizador, diminuindo a latência e custos de comunicação.
- A manutenção e segurança da infraestrutura é da responsabilidade do Cloud Provider.

No entanto, esta tecnologia também tem alguns desafios:

- Disponibilidade do serviço. É possível que o provedor de serviços Cloud não consiga disponibilizar recursos suficientes para assegurar o serviço ou até que não consiga disponibilizar o serviço de todo, devido a por exemplo, problemas em servidores.
- Confidencialidade dos dados.
- *Vendor lock-in.* Quando se começa a utilizar um *cloud provider*, é difícil trocar para outro *provider*. Algumas das causas para a necessidade de fazer uma troca são por exemplos o aumento do preço a pagar pelo serviço, o *provider* poderá falir e terminar o seu serviço ou o *provider* poderá alterar o seu produto de forma a que já não esta dentro das necessidades da empresa. Isto é um problema pois a troca de operador poderá trazer custos (de transferência de dados) bastante elevados ao ponto de não compensar à empresa fazer esta troca [32]. Também poderão existir incompatibilidades

técnicas [33]. Como tal, existem empresas com serviços Cloud em vários provedores dependendo das suas necessidades.

Segundo a Cloud Security Alliance [2], existem três *delivery models* (que representam uma combinação de recursos IT disponibilizadas pelo *cloud provider*) possíveis para um serviço na Cloud: *Software-as-a-Service* (SaaS), *Platform-as-a-Service* (PaaS) e *Infrastructure-as-a-Service* (IaaS) e podem ser implementadas em Clouds publicas ou privadas.

Software-as-a-Service (Saas) consiste em um Cloud Provider fornecer aplicações baseadas numa infraestrutura na Cloud. Neste tipo de serviço, o utilizador não tem qualquer controlo sobre a infraestrutura em si. Exemplos deste tipo de aplicação são por exemplo serviços de Email e Blogs.

Platform-as-a-Service (PaaS) possibilita que os utilizadores desenvolvam e implantem as suas próprias aplicações usando a infraestrutura na Cloud. O utilizador continua sem ter qualquer controlo sobre a infraestrutura em si, exceto sobre a aplicação em si. Um exemplo deste tipo de serviço é o Google App Engine.

Infrastructure-as-a-Service (IaaS) fornece aos utilizadores a infraestrutura, como armazenamento, rede e processamento, para ser possível executar qualquer tipo de software. Como exemplo temos o AWS Elastic Compute Cloud.

Numa Cloud privada a infraestrutura é operada apenas por uma organização enquanto que numa Cloud publica a infraestrutura é disponibilizada para o publico em geral, como é o caso da Amazon.

Esta é também uma área que apresenta grandes lucros. Segundo a CNBC, a AWS, ou Amazon Web Services reportou lucros no valor de 13 mil milhões de dólares em 2020 [25], contabilizando mais de 50% do lucro total da Amazon nesse ano. Segundo a TBR, o lucro de *cloud providers* subiu 5% entre 2015 e 2018 [24]. Isto faz com que a cloud seja uma área excelente para investimentos.

2.2 Introdução a *Application Performance Management*

Application Performance Management, ou APM, é o nome dado ao uso de tecnologia para monitorizar, corrigir, e/ou otimizar sistemas dentro de uma organização [3].

O software de APM assegura que aplicações críticas tenham o seu desempenho esperado e que este pode ser monitorizado. Isto é feito através da medição do desempenho da aplicação,

alertando administradores de sistema quando não é atingido um nível de desempenho mínimo. Ajuda-os também a conseguirem detetar a causa dos problemas de forma que possam ser resolvidos antes de terem um impacto nos utilizadores.

É importante que uma aplicação durante a produção tenha um bom desempenho visto que isto poderá trazer perdas de receita. A Google perde 20% do tráfego se os seus websites demorarem mais 500ms a responder [5]. Há cerca de 10 anos, a Amazon perdia 1% de receita por cada 100ms de latência [6]. Um estudo da Mozilla mostrou que se um website não carregar em menos de 5 segundos os utilizadores desistem do website [7].

As atividades de APM podem ser divididas em quatro etapas [4]:

- Coleção/recolha dos dados
- Armazenamento e processamento dos dados
- Apresentação dos dados
- Interpretação e uso dos dados

Estes dados também podem ser chamados de métricas. Cada métrica é usualmente um par *timestamp*-valor. Um *timestamp* é uma sequência de caracteres que identifica quando um evento ocorreu e é normalmente a sua data e hora com precisão ao nível dos segundos. Eis alguns exemplos de possíveis *timestamps*:

- 08/22/2022 @ 11:59am
- 2022-08-22T11:59:48+00:00
- Mon, 22 Aug 2022 11:59:48 +0000
- 1661169617 (segundos deste 1 de janeiro de 1970)

Coleção/recolha de dados

As métricas que podemos obter irão depender do tipo de sistema que estivermos a monitorizar. Um sistema de armazenamento irá disponibilizar métricas como o número de bytes transferidos ou quantos acessos foram feitos a um determinado ficheiro, o que é diferente de um sistema de base de dados por exemplo, em que podemos ter métricas como o número de linhas acedidas. Podemos obter métricas a diferentes níveis do nosso sistema. É possível obter métricas desde o nível do hardware, como o uso de CPU da máquina, até ao nível de utilizador, como o tempo em que o utilizador está na página ou o tempo de carregamento da página.

Também é importante que a coleção dos dados em si não tenha impacto na performance.

Armazenamento e processamento de dados

Application Performance Management resulta em grandes quantidades de dados. Usualmente teremos dezenas de métricas diferentes cada uma delas a ser registada imensas vezes por minuto. Como tal, é necessário armazenar e processar estes dados de forma eficiente. Para representar esta informação são usualmente usadas series temporais. Séries temporais representam estatísticas como contadores, percentagens, etc. Pode também ser usado *execution traces* que representam o fluxo interno de execução da aplicação quando algo é solicitado.

Apresentação dos dados

É necessário termos formas eficientes de apresentar os dados devido à sua grande quantidade. Os dados têm de ser apresentados de forma a que façam sentido. Estes dados podem ser apresentados em formas de *traces*, *time series*, *page flows*, *topologies*, *server health*, entre outras. Estas visualizações podem ser mais ou menos detalhadas dependendo de o que for necessário para responder a uma determinada questão.

Interpretação e uso dos dados

Os dados disponíveis podem ser utilizados com objetivos diferentes, como a deteção de problemas antes que estes ocorram ao utilizador. Neste caso são utilizadas técnicas estatísticas aplicadas aos dados de forma a detetar anomalias que podem indicar problemas antes de estes ocorrerem. Usualmente é também possível configurar alarmes para que quando seja detetado um problema, o operador do sistema ter conhecimento. Quando ocorre de facto um problema são então utilizados estes dados de forma semelhante, para perceber a causa e ser possível a correção do problema.

2.3 *Application Performance Management* em ambientes Multi Cloud

É possível que sistemas mais complexos estejam distribuídos por diversos *cloud providers* e dentro de um *cloud provider* por várias regiões. Possíveis razões para isto podem ser para evitar *vendor lock-in* em que se torna muito difícil a mudança de *cloud provider,* como explicado na secção 2.1 ou simplesmente para minimizar gastos, escolhendo um *cloud provider* que disponibilize um serviço mais em conta para cada tipo de serviço necessário.

Isto poderá trazer dificuldades quando queremos analisar o desempenho, tais como:

- Diferentes *cloud providers* usam tecnologias diferentes.

- Diferentes *cloud providers* usam formatos diferentes de apresentar e organizar os recursos e dados métricos.
- As métricas entre os vários *cloud providers* podem não ser consistentes (um pode disponibilizar uma métrica para um serviço que não existe noutro).
- Cada *cloud provider* tem os seus próprios *Software Development Kits* (SDKs) e *Application Programming Interfaces* (APIs) que são as ferramentas a usar para obter essas métricas.

Nas próximas secções iremos ver como os recursos estão organizados, como os obter e as suas métricas para as três grandes *clouds*: Amazon Web Services, Google Cloud Platform e Microsoft Azure.

2.3.1 Amazon Web Services

Os três *cloud providers* disponibilizam alguma forma de agrupar recursos (*resources)*, formando assim *resource groups* (grupos de recursos).

No AWS um *resource group* pode ser definido através de *tags* (ou etiquetas). Estes *tags* são basicamente um conjunto de pares chave-valor, em que o seu valor é opcional, que podem ser atribuídos aos vários recursos existentes no AWS. Os recursos que englobam um *resource group* são aqueles que têm pelo menos uma das *tags* definidas no *resource group*.

Na Figura 1 é possível ver um exemplo desta representação. Neste caso temos quatro recursos que são distribuídos pelos dois *resource groups* conforme as suas *tags*.

O Resource Group #1 é constituído pelos recursos que tenham as *tags* Tag #1 ou Tag #2 enquanto o Resource Group #2 é composto pelos recursos que tenham a tag Tag #3. Como vários recursos podem usar a mesma *tag*, temos ambos o Resource #1 e #2 a utilizar a mesma *tag*, e os Resource #3 e #4 a utilizar as *tags* #2 e #3 respetivamente. Temos então que o Resource Group #1 é composto pelos recursos #1, #2 e #3 e o Resource Group #2 é composto pelo Resource #4.

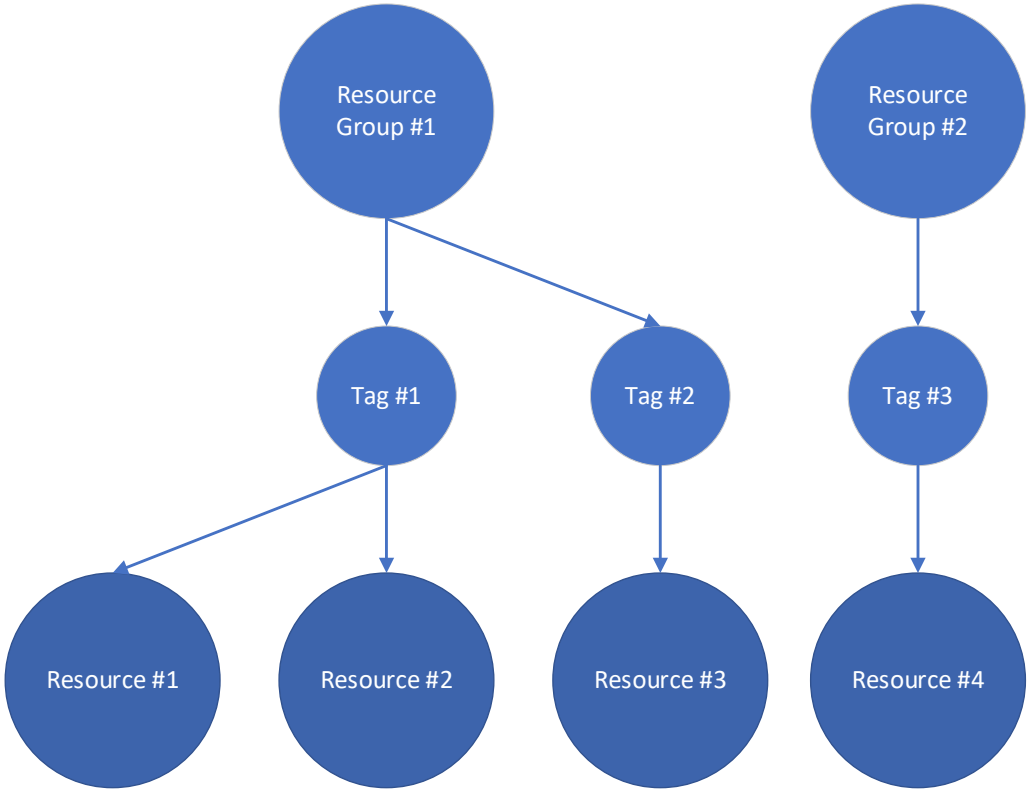


Figura 1 Exemplo da estrutura de recursos no AWS

Para obtermos os *resource groups* existentes numa conta na AWS usamos a Resource Groups API. A função utilizada para o efeito é a *ListGroup*.

A função *ListGroup* pode ainda receber filtros, é possível também especificarmos um máximo de resultados e um token para obtermos a página seguinte de resultados. O resultado desta função é uma listagem com os vários *resource groups* existentes, cada um contendo o seu nome e o seu Amazon Resource Name (ARN) e um token para a próxima página de resultados.

Um ARN funciona como identificador de recursos [40]. Cada ARN é composto por uma partition, service, region, account id e um resource id e/ou resource type. Vendo em detalhe cada um destes componentes temos:

- Partition: Existem três partições possíveis – aws, aws-cn e aws-us-gov. Uma partição é um grupo de regiões. aws-cn denomina regiões na china, aws-us-gov denomina regiões para o governo dos Estados Unidos e apenas aws para as restantes regiões.
- Service: Espaço de nomes (*namespace*) do tipo de serviço.
- Region: O código da região.
- Account Id: O identificador da conta a quem pertence o recurso.
- Resource Id: O identificador do recurso.

Depois de termos os *resource groups*, é necessário obtermos os recursos pertencentes em cada um. Para isso utilizamos a função *ListGroupResources* ainda da mesma API. Esta função pode receber o nome do *resource group*, filtros e também um limite de resultados e um *token*, tal como na função anterior. Como resultado temos os vários recursos, cada um com o seu tipo e ARN e um *token* que permite obter a próxima página com os restantes resultados.

Para obtermos as métricas disponíveis para um serviço usamos a função *ListMetrics* desta API. Para esta função, apesar de não ser necessário especificar um filtro, este é recomendado, caso contrário iremos obter todas as métricas disponíveis para os vários serviços. O campo que deveremos utilizar para a definição de filtros é o *namespace* onde podemos especificar o *namespace* das métricas para o serviço que queremos. Este *namespace* coincide com o tipo de serviço, sendo por exemplo, AWS/EC2 para o Elastic Compute Could. Também é possível especificar um *token* que serve para a paginação de resultados. Como resposta temos novamente um *token* para a página seguinte e um conjunto de métricas disponíveis para um serviço.

Depois de termos as métricas que podem ser obtidas para um recurso, é necessário obter as métricas em si. Novamente, utilizando o CloudWatch é possível usar a função *GetMetricData* para este efeito. Esta função necessita de receber timestamps do início e do fim do intervalo para o qual queremos os dados. Opcionalmente, podemos ainda definir rótulos (como o título dos dados e das séries temporais), o número máximo de pontos (pares *timestamp*-valor que contêm os dados) que queremos, um *token* para paginação e a ordem por que são apresentados os pontos. Claro que é necessário também especificar as métricas que queremos.

Para cada métrica é necessário especificarmos um identificador e, ou um objeto *MetricStat* com a métrica que queremos ou uma expressão de métricas utilizando *Metric Math*.

Cada objeto MetricStat tem um objeto com a descrição da métrica em si, a granularidade dos dados que tem de ser um múltiplo de 60 (ou 1 minuto), a não ser que seja uma métrica de alta resolução, o tipo de estatística que queremos (média, soma, etc) e as unidades para os dados.

Como resposta, recebemos uma mensagem com o resultado das operações, um *token* e um vetor com os resultados para as várias métricas. Cada métrica contém um rótulo, o identificador mencionado e dois vetores, um com os timestamps e outro com os valores para cada timestamp.

Um exemplo de um pedido deste tipo poderá ser como demonstrado pelo seguinte [36]:

```
{
    "StartTime": 1637061900,
    "EndTime": 1637072400,
    "MetricDataQueries": [
        {
            "Expression": "SELECT AVG(CPUUtilization) FROM SCHEMA(\"AWS/EC2\",
InstanceId)",
            "Id": "q1",
            "Period": 300,
            "Label": "Cluster CpuUtilization"
        },
        {
            "Id": "m1",
            "Label": "Unhealthy Behind Load Balancer",
            "MetricStat": {
                "Metric": {
                    "Namespace": "AWS/ApplicationELB",
                    "MetricName": "UnHealthyHostCount",
                    "Dimensions": [
                        {
                            "Name": "TargetGroup",
                            "Value": "targetgroup/EC2Co-Defau-EXAM-
PLEWNAD/89cc68152b367e5f"
                        },
                        {
                            "Name": "LoadBalancer",
                            "Value": "app/EC2Co-EcsEl-EXAM-
PLE69Q/fdd2210e799e4376"
                        }
                    ]
                },
                "Period": 300,
                "Stat": "Average"
            }
        }
    ]
}
```

Neste exemplo são pedidas ao Amazon CloudWatch duas métricas, especificadas em MetricDataQueries. A primeira, com o identificador q1, obtém o uso de CPU médio da instância InstanceId presente no EC2 através de uma expressão. Já a segunda obtém a média da métrica UnHealthyHostCount para duas instâncias do Elastic Load Balancer.

E obtendo como resposta:

```
{
    "Messages": [],
    "MetricDataResults": [
        {
            "Id": "m1",
            "Label": "Unhealthy Behind Load Balancer",
            "StatusCode": "Complete",
            "Timestamps": [
                1637074200,
                1637073900,
                1637073600
            ],
            "Values": [
                0,
                0,
                0
            ]
        },
        {
            "Id": "q1",
            "Label": "Cluster CpuUtilization",
            "StatusCode": "Complete",
            "Timestamps": [
                1637074245,
                1637073945,
                1637073645
            ],
            "Values": [
                1.2158469945359334,
                0.8678863271635757,
                0.7201860957623283
            ]
        }
    ]
}
```


2.3.2 Google Cloud Platform

No caso do Google Cloud Platform, ou GCP, os recursos são organizados de maneira ligeiramente diferente. No caso o GCP tem uma hierarquia de 4 níveis: *Organizations*, *Folders*, *Projects* e *Resources*. Fazendo uma correspondência com a organização da AWS temos que os *Projects* do GCP correspondem aos *Resource Groups* da AWS. Os dois níveis acima permitem uma melhor organização dos vários projetos e organizações que um utilizador possa ter, como é possível ver na Figura 2.

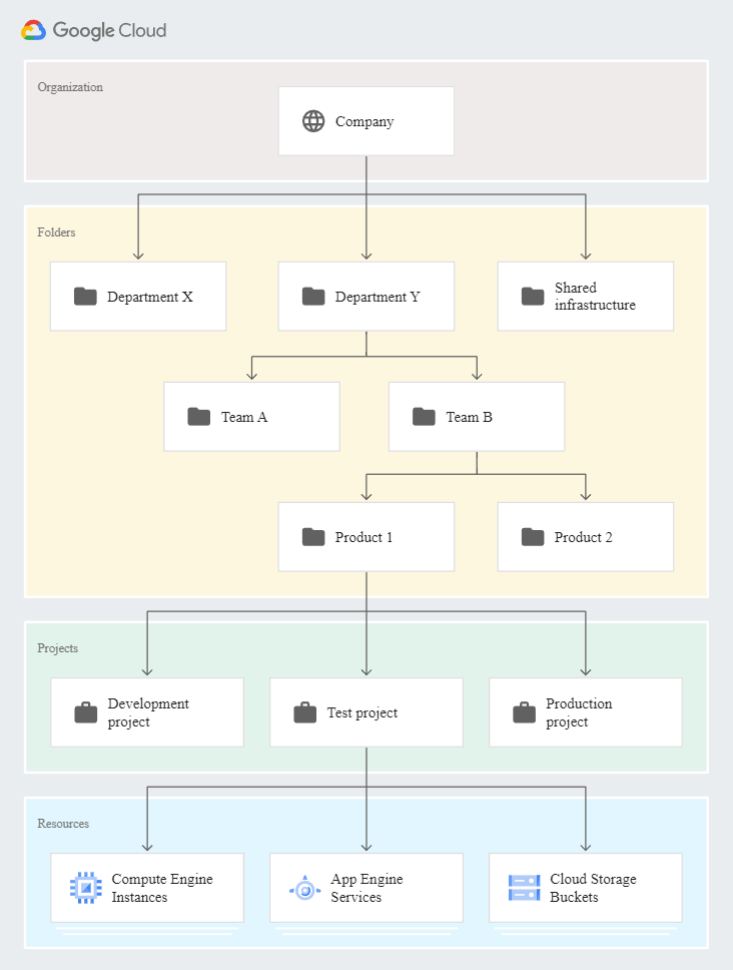


Figura 2 Estrutura de recursos no Google Cloud

A obtenção dos vários recursos pertencentes a um projeto faz-se por duas partes. Primeiro é necessário obter os projetos existentes. Para isso usamos a Resource Manager API. Ao usarmos a função *projects.search* obtemos uma listagem com os vários projetos aos quais temos acesso. Sendo este um método de pesquisa, é possível especificar um filtro. Esta operação suporta funções de paginação com *tokens*.

Depois de termos os projetos que queremos, é necessário obter os recursos existentes em cada um deles. Para isso usamos a função *assets.list* da API Cloud Asset Inventory. Esta função requer como parâmetro um elemento da hierarquia da Figura 2, à exceção de *resources*, pois é isso que queremos obter. É nos devolvido então todos os *assets* existentes para este elemento.

Um *asset* pode ser qualquer elemento da hierarquia presente na Figura 2 ou elementos que não sejam representados nesta como permissões de controlo de acesso de um utilizador. É possível também especificar o tipo de *asset* que queremos que neste caso, seria apenas os recursos para um determinado projeto. Nesta resposta, teremos de ter especial atenção ao tipo

de *asset*, pois isto será posteriormente utilizado para obter as métricas para um tipo de recurso. Também é aqui que podemos obter a localização de um recurso. Tal como outras funções, esta também suporta paginação com *tokens*. Utilizando a Figura 2 como exemplo, para obter os recursos do projeto *Test project* seria necessário primeiro pesquisar por este projeto com a função *projects.search* e, posteriormente, usar-se-ia a função *assets.list* para obter os três recursos associados a este.

Ao contrário do Amazon AWS, a Google disponibiliza uma listagem com todas as métricas disponíveis para todos os serviços [12]. No entanto, é possível, e mais prático obtermos apenas as métricas para um tipo de recurso, tal como no AWS.

A plataforma que a Google utiliza para isto é a Google Cloud Operations Suite. Tal como no Amazon CloudWatch, também aqui podemos obter as métricas dos vários recursos bem como definir alarmes ou obter registos.

Para obtermos as métricas para um projeto utilizamos a função *projects.metricDescriptors.list* desta API. Esta função requer novamente a especificação de qual projeto nos estamos a tratar. Apesar de não ser necessário, é recomendado utilizarmos um filtro para definir o tipo de métricas que queremos. É aqui que podemos usar o tipo de *asset* obtido anteriormente. Ao não definirmos um filtro, iremos obter todas as métricas disponíveis para todos os tipos de projetos. A resposta é um conjunto de métricas suportadas pelo tipo de *asset*, cada uma com a sua unidade, o tipo de valor (numérico, booleano, etc), entre outros dados importantes para a sua representação.

Depois de termos finalmente os projetos e que métricas são possíveis para cada um deles, podemos começar a requisitar métricas. Existem duas formas de o fazer, ou com a função *projects.timeSeries.list* ou com a função *projects.timeSeries.query*.

O primeiro destes requer um elemento da hierarquia (organização, diretoria ou projeto), um filtro onde podemos especificar o tipo de métrica que queremos, um intervalo de tempo e se queremos os dados completos ou apenas informação das métricas sem os seus dados. Existem ainda outros campos opcionais como definição de agregações, limite de resultados a apresentar e *tokens* para paginação.

Como resposta temos um conjunto de séries temporais, sendo cada série temporal do seguinte tipo [12]:

```
{
  "metric": {
    object (Metric)
  },
  "resource": {
    object (MonitoredResource)
  },
  "metadata": {
    object (MonitoredResourceMetadata)
  },
  "metricKind": enum (MetricKind),
  "valueType": enum (ValueType),
  "points": [
    {
      object (Point)
    }
  ],
  "unit": string
}
```

Cada objeto deste tipo temos:

- *metric*: contem informações sobre a métrica em si, como o seu rótulo.
- *resource*: descreve o recurso para o qual estamos a obter as métricas.
- *metadata*: contém meta dados adicionais sobre o recurso.
- *metricKind*: especifica o tipo de métricas que estamos a obter, pode ser GAUGE, DELTA ou CUMULATIVE. GAUGE será um valor medido num instante, DELTA será a diferença num intervalo de tempo e CUMULATIVE será um valor acumulado durante um intervalo de tempo.
- *valueType*: tipo do valor da métrica (númerico, booleano, etc).
- *points*: vetor com os vários pares *timestamp*-valor.
- *unit*: a unidade em que vêm as métricas.

Já a função *query* permite obter métricas utilizando a *Monitoring Query Language* da Google. Esta função requer o projeto e uma *query* e obtemos como resposta uma resposta semelhante à anterior, se bem que num formato diferente.

2.3.3 Microsoft Azure

A hierarquia de recursos do Azure é algo semelhante à da Amazon Web Services. Temos *resource groups*, cada um com vários recursos. Para além destes dois níveis temos ainda mais

dois níveis: *management groups* e *subscriptions*; por ordem: *management groups, subscriptions*, *resource groups* e *resources*, como é possível observar na Figura 3.

Os *management groups* servem para organizar subscrições. Por exemplo, é possível separar as várias subscrições por setores de uma empresa e assim aplicar possíveis restrições com base do seu setor. A cada utilizador pode ser atribuído uma ou mais subscrições.

De notar que ao contrário da Amazon Web Services, no Azure os recursos existentes num *resource group* podem estar em regiões diferentes.

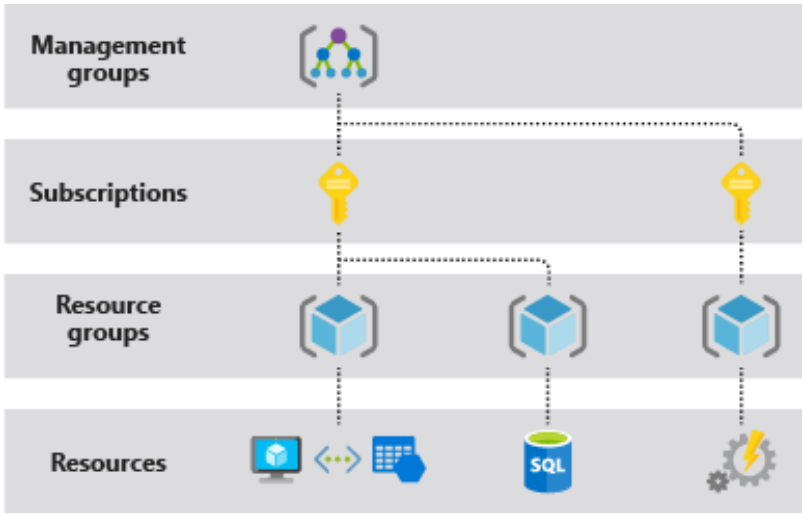


Figura 3 Estrutura de recursos no Microsoft Azure

Antes de conseguirmos obter os recursos no Azure é necessário primeiro obter a subscrição onde temos os recursos. A API Resource Management disponibiliza estas funções. A função *Subscriptions – List* dá-nos acesso a esta informação.

Para obtermos os *resource groups* no Azure usamos a função *Resource Groups – List* que permite obter todos os *resource groups* dentro de uma subscrição. Para isso temos de fornecer o identificador da subscrição que obtemos anteriormente e, opcionalmente, um filtro e um número máximo de resultados.

Depois de termos os *resource groups* podemos finalmente aceder aos recursos com a função *Resources - List By Resource Group*. Para esta função necessitamos dos identificadores da subscrição e *resource group* e podemos ainda opcionalmente usar filtros e especificar um número máximo de resultados tal como na função anterior. Aqui obtemos informação como o identificador de um recurso, o seu URI (*Uniform Resource Identifier*), a sua localização, nome, entre outros. Todas estas três funções suportam paginação.

Para a coleção de métricas o Azure tem o Azure Monitor. Novamente, nesta ferramenta temos uma visão geral do desempenho da nossa aplicação e podemos também obter registos e definir alarmes.

Obter as métricas disponíveis para um recurso no Azure é relativamente mais fácil que nos dois casos anteriores. O Azure Monitor disponibiliza a função *Metric Definitions – List*. Nesta função apenas temos de fornecer o URI do recurso ao contrário de no AWS e no GCP onde era necessário especificar o namespace da métrica ou o tipo de asset respetivamente. Como resultados temos as várias métricas e as suas informações como o seu namespace, nome, descrição, unidade, agregações suportadas, intervalo, entre outras.

Agora podemos finalmente obter as métricas para cada recurso. Metrics – List requer o URI do recurso e tem como argumentos opcionais:

- *filter*: possibilita o uso de filtros
- *aggregation*: lista de tipos de agregações a obter, com base nas agregações disponíveis como média, contagem, soma, etc (obtido na função *Metric Definitions – List*)
- *interval*: granularidade dos dados a obter (obtido na função *Metric Definitions – List*)
- *metricnames*: lista com as várias métricas a obter (obtido na função *Metric Definitions – List*)
- *metricnamespace*: nome do *namespace* das métricas que queremos obter (obtido na função anterior)
- *orderby*: ordenação dos resultados
- *timespan*: tempos de início e fim para as métricas que queremos
- *top*: máximo de resultados a apresentar (requer o uso de um filtro)

Como resposta obtemos algumas das informações inseridas anteriormente como o intervalo e o *namespace*, a localização do recurso e as várias métricas obtidas. Para cada métrica temos alguma da informação já disponibilizada com a função *Metric Definitions – List*, como o nome da métrica e a sua unidade e temos também as várias séries temporais que pertencem a essa métrica. Para cada série temporal temos os seus metadados e os dados em si.

Como exemplo temos o seguinte pedido: `GET https://management.azure.com/subscriptions/b324c52b-4073-4807-93af-e07d289c093e/resourceGroups/test/providers/Microsoft.Storage/storageAccounts/larryshoebox/blobServices/default/providers/Microsoft.Insights/metrics?timespan=2017-04-14T02:20:00Z/2017-04-14T04:20:00Z&interval=PT1M&aggregation=Average,count&top=3&orderby=Average asc&$filter=BlobType eq '*'&api-version=2018-01-01&metricnamespace=Microsoft.Storage/storageAccounts/blobServices`

Neste pedido vamos ao recurso `subscriptions/b324c52b-4073-4807-93af-e07d289c093e/resourceGroups/test/providers/Microsoft.Storage/storageAccounts/larryshoebox/blobServices/default` procurar métricas de 14/04/2017 02:20.00 até 14/04/2017 04:20.00, com um intervalo de 1 minuto, fazendo a média e uma contagem de cada medição num período de 1 minuto, apresentando apenas 3 resultados por pedido, ordenando os resultados por média crescente e usando como filtro qualquer tipo de BlobType (do namespace `Microsoft.Storage/storageAccounts/blobServices`). Como resultado obtemos o seguinte:


```
{
  "cost": 0,
  "timespan": "2017-04-14T02:20:00Z/2017-04-14T04:20:00Z",
  "interval": "PT1M",
  "namespace": "Microsoft.Storage/storageAccounts/blobServices",
  "resourceregion": "eastus2",
  "value": [
    {
      "id": "/subscriptions/b324c52b-4073-4807-93af-e07d289c093e/resource-
Groups/test/providers/Microsoft.Storage/storageAccounts/larryshoebox/blob-
Services/default/providers/Microsoft.Insights/metrics/BlobCapacity",
      "type": "Microsoft.Insights/metrics",
      "displayDescription": "The amount of storage used by the storage account's
Blob service in bytes.",
      "name": {
        "value": "BlobCapacity",
        "localizedValue": "Blob Capacity"
      },
      "unit": "Bytes",
      "timeseries": [
        {
          "metadatavalues": [
            {
              "name": {
                "value": "blobtype",
                "localizedValue": "blobtype"
              },
              "value": "PageBlob"
            }
          ],
          "data": [
            {
              "timeStamp": "2017-04-14T02:20:00Z",
              "count": 0
            },
            {
              "timeStamp": "2017-04-14T02:21:00Z",
              "count": 0
            },
            {
              "timeStamp": "2017-04-14T02:22:00Z",
              "count": 0
            },
            {
              "timeStamp": "2017-04-14T02:23:00Z",
```

```
            "count": 1,
            "average": 0
          }
        ]
      },
      {
        "metadatavalues": [
          {
            "name": {
              "value": "blobtype",
              "localizedValue": "blobtype"
            },
            "value": "BlockBlob"
          }
        ],
        "data": [
          {
            "timeStamp": "2017-04-14T02:20:00Z",
            "count": 0
          },
          {
            "timeStamp": "2017-04-14T02:21:00Z",
            "count": 0
          },
          {
            "timeStamp": "2017-04-14T02:22:00Z",
            "count": 0
          },
          {
            "timeStamp": "2017-04-14T02:23:00Z",
            "count": 1,
            "average": 245
          }
        ]
      }
    ]
  }
]
}
```


Neste resultado são obtidos dados para duas métricas. Cada valor irá conter um *timestamp* e uma função estatística, neste caso, como o pedido foi feito para média (average) e o total (count), não iremos obter a média quando o total é zero pois não é possível efetuar divisão por 0.

Com isto podemos concluir que todos os três sistemas são diferentes. Todos eles têm organizações de recursos diferentes e formas diferentes de obter e representar os dados relativos às métricas, o que torna difícil criar um sistema que consiga obter e visualizar estes dados como se se tratasse tudo de apenas um c*loud provider*.

2.4 Introdução a Bases de Dados

Uma base de dados é uma coleção de dados que contém informação relevante a certo assunto ou sistema. Os sistemas de bases de dados, que gerem um conjunto de bases de dados são desenhados para que seja possível gerir uma grande quantidade de informação. Esta gestão dos dados envolve não só a definição de estruturas de dados para armazenar estes dados, como também fornecer mecanismos de manipulação desses dados. Bases de dados estão presentes em muitas das aplicações que usamos no nosso dia a dia, como por exemplo em aplicações de *e-commerce*, *banking*, entre outras [8].

O que define a estrutura de uma base de dados é o seu modelo de dados. Existem vários modelos, sendo o mais conhecido o modelo relacional.

A linguagem usada para especificar e manipular bases de dados é composta por duas partes. Uma delas é a *Data Definition Language*, que serve para especificar o esquema da base de dados e possíveis restrições que possam existir enquanto que a *Data Manipulation Language* serve para manipular os dados dentro desta.

Os sistemas de bases de dados suportam transações. Uma transação é uma coleção de operações feitas sobre uma base de dados que realizam uma única função lógica. Cada transação tem de cumprir as propriedades ACID. ACID significa *Atomicity Consistency Isolation Duratibility* e diz que:

- Atomicidade: Quando queremos transferir dinheiro de uma conta bancária para outra, precisamos de primeiro subtrair a quantidade na origem e adicionar ao destino. É necessário garantir que esta operação só é dada por concluída se ambas a subtração e adição concluírem com sucesso. Isto é: é tudo feito, ou nada.
- Consistência: É necessário garantir que a base de dados esteja sempre num estado consistente, antes e depois de uma transação executar.
- Isolamento: Cada transação tem de ser executada sem interferir com outras transações.
- Durabilidade: Todas as transações que forem concluídas com sucesso são registadas de forma a que a base de dados consiga recuperar no caso de alguma falha.

A existência de transações é importante no especial caso de termos utilizadores concorrentes a modificar os mesmos valores. Dependendo do modo de concorrência definido no sistema de base de dados, é possível termos uma transação A, em que atualizamos um valor, mas que ainda não foi *committed* e ao mesmo tempo uma transação B, lê esse mesmo valor antes de ser atualizado.

2.4.1 Bases de Dados Relacionais

As bases de dados relacionais são baseadas no modelo relacional. Usam uma coleção de tabelas para representar dados e relações entre estes dados.

Cada tabela é composta por colunas, cada coluna irá representar um dado que queremos guardar. Usualmente, cada tabela tem, para além de uma coluna para cada dado a guardar, tem também uma coluna com um identificador. Este identificador tem de ser único e normalmente é um número inteiro automaticamente incrementado. Esse identificador também é chamado de chave primaria. Também é possível definir restrições para cada coluna.

Por exemplo, para um serviço, podemos ter uma coleção de utilizadores como:

ID_Utilizador	Nome	Morada	Data de Nascimento
1	João	Rua X, Lisboa	18/01/1980
2	Maria	Rua Y, Setúbal	30/07/1992

Tabela 1 Exemplo de uma tabela de pessoas numa base de dados relacional

Se assumirmos que isto é uma base de dados que contem possíveis recursos na cloud, podemos ter uma tabela de recursos e uma tabela dos recursos dos utilizadores como as seguintes:

ID_Recurso	Tipo	Nome	Preço
1000	Armazenamento	StorageV2	1€
1001	Máquina Virtual	VMMedium	5€

ID	ID_Utilizador	ID_Recurso	Nome	Preço
11000	1	1000	DiscoJoao	1€
11001	2	1001	VMMaria	5€

Cada tabela de recursos contém o seu identificador e chave primaria, um campo com o nome e tipo de recurso, e um campo com o preço tabelado do recurso que tem de ser superior a 0.

Já a tabela dos recursos dos utilizadores está diretamente relacionada com ambas as tabelas de cima. Temos um ID da associação como chave primaria e ambas as chaves das tabelas de utilizadores e produtos como chaves estrangeiras. Temos ainda o nome e o preço do recurso. Isto descreve uma relação N para M pois um utilizador pode ter vários recursos e um recurso pode ser vendido a mais que uma pessoa.

Uma base de dados relacional tem de cumprir as propriedades ACID, logo uma operação de uma compra de um produto que resulta numa transação que subtraia a quantidade comprada na tabela de produtos e adiciona uma entrada à tabela de vendas o produto comprado, só poderá ser concluída com sucesso caso exista quantidade suficiente em sistema (pela

restrição que a quantidade em armazém tem de ser superior ou igual a 0). Também não deve ser possível que apenas se faça uma parte da operação, como apenas subtrair a quantidade do produto disponível ou apenas adicionar o produto comprado à tabela de vendas sem o subtrair pois em ambos os casos a base de dados ficaria inconsistente.

Possíveis vantagens com o uso de bases de dados relacionais temos:

- Modelo simples. Apenas temos tabelas e relações entre estas.
- Redundância reduzida. Os dados aparecem apenas em um local (por exemplo apenas uma tabela de clientes) e são feitas ligações de essa tabela para onde for necessário.
- Facilidade de backup e recuperação em caso de desastres. Sendo que este tipo de bases de dados cumpre as propriedades ACID, o sistema estará sempre consistente a qualquer momento. Logo é possível executar cópias de segurança (*backups*) dos dados, dados a qualquer altura, mesmo enquanto a base de dados está em execução sem existir problemas de inconsistência de dados no momento em que estas são feitas.

A principal linguagem utilizada pelos sistemas de bases de dados relacionais é SQL, muitas vezes também chamadas de bases de dados SQL.

2.4.2 Bases de Dados NoSQL

As bases de dados NoSQL, ou *not only SQL*, surgiram pela necessidade de arranjar maneiras mais eficientes para a organização e representação dos dados de certo tipo de dados visto que nem sempre a representação de dados em forma de tabelas é a mais indicada [42]. Algumas das vantagens de uso deste tipo de bases de dados são:

- Escalabilidade: Grande parte dos sistemas de bases de dados relacionais foram feitos para serem executados em apenas um servidor. Tentar distribuir uma base de dados relacional por múltiplos servidores não é uma tarefa fácil pois é necessário lidar com problemas como a maneira como os dados estão particionados por múltiplos servidores ou a junção de tabelas em vários servidores [20].
- Modelos não relacionais: Bases de dados relacionais, não estão preparadas para lidar com outros modelos sem ser modelos relacionais. Por exemplo, representar um grafo, constituído por nós e arcos pode requerer várias tabelas para o representar numa base de dados relacional enquanto que numa base de dados NoSQL que utilize *graph store* a sua representação é o próprio grafo. Logo, nas bases de dados NoSQL, os dados não necessitam de estar organizados por tabelas.

- Disponibilidade: Bases de dados NoSQL deixam de seguir as propriedades ACID de forma a suportar mais alta disponibilidade e melhores desempenhos, trocando estas por um modelo de consistência mais fraco. Numa base de dados NoSQL, de forma a garantir que é capaz de responder a milhões de utilizadores concorrentes, nem sempre os dados que iremos ler irão estar consistentes, mas eventualmente, à medida que os dados são replicados por todos os servidores, estes ficaram consistentes. Claro que isto não é algo possível em todos os casos. Aplicações que façam transações bancárias continuam a ter de ser consistentes.

Existem vários tipos de bases de dados NoSQL que normalmente são agrupados em quatro categorias [21]:

- Key-Value store: Os dados são guardados como pares chave-valor como num dicionário.
- Document store: Este tipo é semelhante ao modelo key-value, mas em que todos os pares chave-valor são armazenados em documentos. Estes documentos podem ser vistos como uma coleção de pares chave-valor. Normalmente são codificados em formatos como XML (*eXtensible Markup Language*) ou JSON (*JavaScript Object Notation*).
- Column store: Este tipo guarda os dados por colunas em vez de por linhas como uma base de dados relacional.
- Graph store: Usado para guardar redes de grafos constituídos por nós e arcos.

Existem também outros tipos de bases de dados NoSQL como bases de dados de séries temporais (ou *Time Series Databases*).

Como um exemplo de uma base de dados NoSQL mais genérica temos o ClickHouse. Esta base de dados é uma base de dados *column-oriented* [17] em que os dados estão organizados por colunas. Isto é ideal para casos como:

- Quando queremos procurar toda a informação de um certo campo, podemos simplesmente ler uma linha inteira, em vez de termos de percorrer todas as linhas e dentro de cada linha, procurar a(s) coluna(s) que contenha(m) a(s) informações relevantes(s) visto que traz grandes aumentos de desempenho na pesquisa. Por exemplo, tendo uma tabela com informações de pessoas e quisermos obter a idade de milhões de pessoas para fazer uma média, em vez de irmos a cada linha, e dentro de cada linha procurar um campo com a idade (ou data de nascimento, que é mais comum) é possível extrair logo uma linha que contém logo toda a informação necessária.
- Os valores guardados são pequenos, tais como números ou pequenas *strings*.

- Não são necessárias transações visto que normalmente estes dados não vão ser atualizados ou serão introduzidos em grandes quantidades de uma vez e se referem a métricas para a sua posterior análise.
- É necessária uma taxa de transferência elevada quando processamos uma *query* visto que normalmente iremos estar a processar milhões de entradas. Normalmente estes resultados são também filtrados e agregados.

Este caso de uso também é conhecido por OLAP (ou OnLine Analytical Processing). OLAP consiste na análise multidimensional de dados empresariais e disponibiliza capacidades para cálculos complexos, análise de tendências e modelação de dados sofisticados [26], por exemplo, o quão bem um produto está a ser vendido em diferentes cidades. Dependendo do que estivermos a analisar, é possível termos várias dimensões, daí o nome multidimensional. Voltando ao exemplo anterior, para responder à questão, podemos ter dimensões como os produtos, várias datas e cidades. Neste caso, é possível ver isto como uma matriz tridimensional, também conhecida como *OLAP cube* (Figura 4).

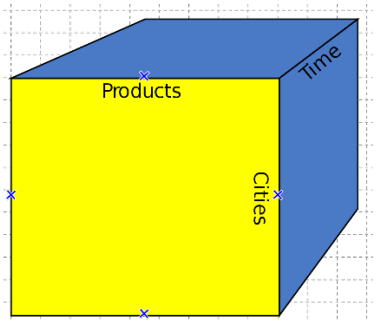


Figura 4 *OLAP Cube* com dimensões de produtos, data e cidades

Bases de dados que suportem este caso de uso têm de suportar *queries* complexas e que executem rapidamente. O ClickHouse é também o sistema de base de dados que está a ser utilizado na Crossjoin para guardar dados como métricas.

2.4.2.1 Bases de Dados em Grafos

As bases de dados baseadas em grafos usam grafos para guardar a informação em vez de tabelas, como é o caso em bases de dados relacionais [21].

Um grafo é composto um conjunto de objetos chamados de nós ou vértices relacionados entre si por um conjunto de arcos. Matematicamente temos $G = (V, E)$ onde V é um conjunto de nós e $E \subseteq V \times V$ é um conjunto de arcos que ligam pares de nós.

Em áreas em que é importante a inter conectividade e topologia dos dados é aconselhado usar um modelo baseado em grafos. Algumas das vantagens do uso de um modelo deste tipo são:

- Grafos são sistemas de representação naturais e simples

- Estruturas em grafos permitem uma modelação mais natural dos dados se estes também tiverem a estrutura em grafos. Por exemplo, é bem mais simples representar as ligações numa árvore genealógica com um grafo do que com tabelas.
- Podem usar estruturas de armazenamento mais eficientes e podem permitir queries que contenham operações especificas a grafos, como a pesquisa do caminho mais curto entre dois nós, que numa base de dados relacional teria de ser implementada pelo programador consoante a estrutura das tabelas.

Voltando ao exemplo da árvore genealógica, ao usarmos uma base de dados relacional para representar os dados necessitamos de duas tabelas, uma com as pessoas, e outra com as relações entre elas. Ao usar um grafo, temos as pessoas como nós e as relações entre elas nos arcos. Usar um modelo em grafo torna a representação de uma árvore genealógica mais natural.

Nome
João
Maria
Pedro

Tabela 2 Lista de pessoas

Pessoa	Pais
Pedro	João
Pedro	Maria

Tabela 3 Relações de parentesco entre várias pessoas

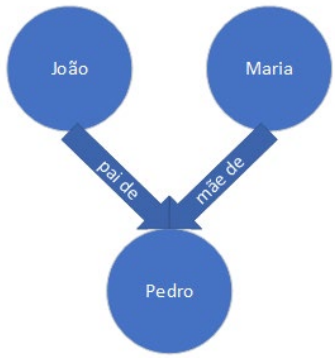


Figura 5 Representação em grafo

Existem vários modelos de grafos. Ao modelo presente na Figura 5 também se pode chamar de *labelled graph.* Neste tipo de grafos temos objetos nos nós e cada propriedade fica definida num arco, em vez de no objeto em si, como é o caso no modelo *property graph.* Um *labelled graph* é também um caso particular de um *property graph.*

No modelo *property graph* podemos ter propriedades nos nós e nos arcos. Estes grafos são *directed, labelled, attributed multigraph* [18]. Um multigrafo é quando temos vários arcos entre dois vértices. No caso dos *property graph* as propriedades são um conjunto de pares chave-valor. Este modelo é comum porque ao retirar ou adicionar componentes podemos

obter outros tipos de grafos [19], como exemplo, ao retirarmos as propriedades dos nós nos *property graph* obtemos um *labelled graph*.

Na Figura 6 temos um *property graph* que contém relações entre várias pessoas, cada uma com as suas propriedades (nome e ano de nascimento) e vários apartamentos (também com as suas propriedades como morada e área) e relações entre pessoas e apartamentos, também com propriedades como o início e fim de residência e ainda, no caso de aluguer do apartamento, o seu custo [38].

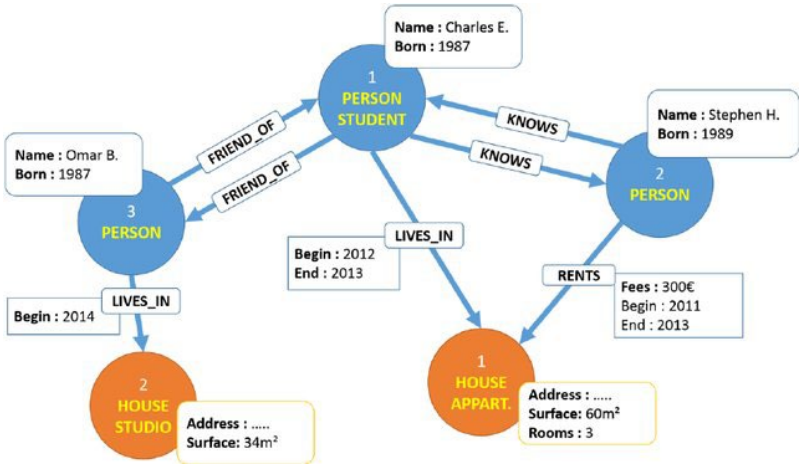


Figura 6 *Property graph* que contem relações entre pessoas e seus locais de residência [38]

Outro modelo também que deve ser referido é o Resource Description Framework (RDF). Este é um modelo desenvolvido pelo consórcio da World Wide Web e denota os vértices e arcos por Uniform Resource Identifiers (URIs). Exemplo na Figura 7 Modelo RDF que representa Joe, e tem propriedades como o seu nome, email e website. Este modelo é bastante utilizado no domínio da *Semantic Web*. Ao contrário de outros modelos, onde não existe uma linguagem standard para manipular estes dados numa base de dados, este modelo utiliza SPARQL.

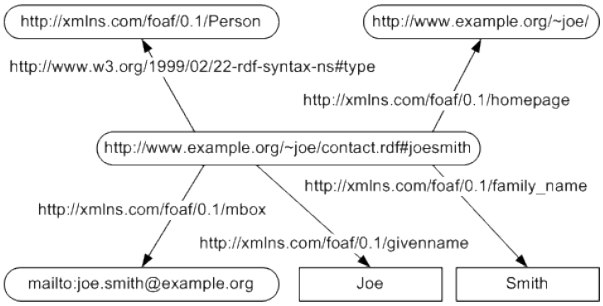


Figura 7 Modelo RDF que representa Joe, e tem propriedades como o seu nome, email e website [37]

No modelo *multi-layer* temos um grafo que contém várias camadas (*layers*). É na verdade, um conjunto de grafos, divididos em camadas, cada um a representar algo específico, que se ligam entre si. Para melhor representar um grafo ou uma rede que consista de múltiplos níveis ou com diferentes tipos de arcos ou nós, podemos considerar também camadas, para além de nós e arcos [34].

Existem duas formas de transformar um grafo num *multi-layer graph*. No caso em que os nós representem objetos diferentes, também chamados de *node-colored graphs*, é possível agrupar os tipos de objetos por camadas mantendo as ligações já existentes (Figura 8).

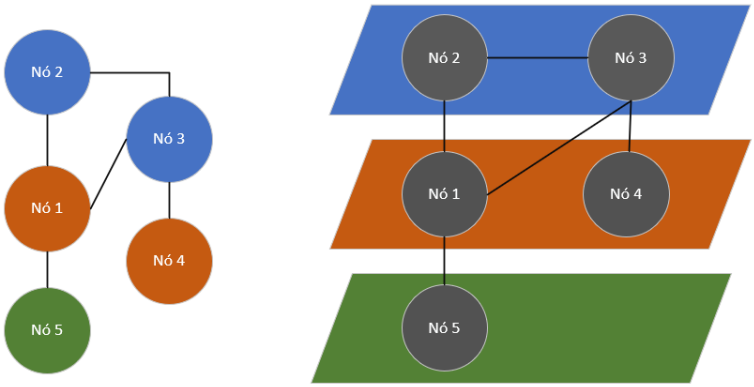


Figura 8 Exemplo de um *node-colored graph* e a sua adaptação a um *multi-layer graph*

Já no caso em que os arcos representem tipos de ligações diferentes (ou *edge-colored graphs*) replicam-se os nós por todas as camadas e cada camada irá conter apenas um tipo de ligações diferente e não vão existir ligações entre as camadas (Figura 9).

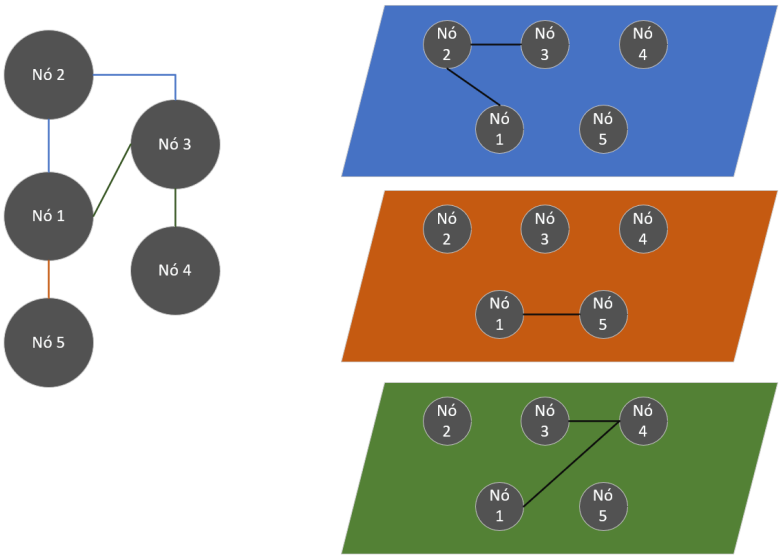


Figura 9 Exemplo de um *edge-colored gaph* e a sua adaptação a um *multi-layer graph*

Isto permite uma melhor organização dos dados porque estamos a dividir diferentes tipos de objetos ou relações por várias camadas diferentes.

É especialmente importante quando tratamos de grandes quantidades de dados e é necessário representar diferentes relações dentro de uma rede [28]. Isto tem usos nas áreas de análise de redes sociais [30], transportes urbanos e internacionais [29], medicina, entre outras.

2.4.2.2 Bases de Dados em séries temporais

Uma série temporal é uma sequência de valores ordenados de uma variável igualmente espaçados entre si [41].

Uma base de dados baseada em series temporais (*Time Series Databases* ou TSDB) é um sistema de base de dados otimizada para ligar com um conjunto de valores (normalmente numéricos) ordenados por uma data. Estas bases de dados foram pensadas de forma a que possam armazenar grandes volumes de dados de forma eficiente.

A chave que diferencia uma base de dados deste tipo relativamente a outros tipos de base de dados é que as *queries* que fazemos sobre esta estão sempre relacionadas com tempos e as operações que fazemos sobre esta normalmente incluem algum intervalo de tempo. Por exemplo, é possível ter uma tabela que guarda a temperatura num certo local a cada minuto em que depois podemos utilizar esses dados para formar médias.

Algumas das propriedades necessárias para as *Time Series Databases* (TSDB) são:

- Localização dos dados: É necessário que dados relacionados estejam juntos em disco visto que operações de I/O podem ser bastante lentas quando os dados não estão ordenados sequencialmente no disco.
- Alto desempenho de escrita: É necessário conseguir um grande desempenho a escrever visto que os dados chegam a todos os segundos.
- Compressão de dados: Como podem ser feitas imensas escritas por segundo são necessários bons métodos de compressão de dados.
- Usabilidade: Devem ser incluídas funções e operações para facilitar a análise de series temporais

Este tipo de bases de dados é otimizado para trabalhar com métricas, visto que estas são sempre um par *timestamp*-valor. Um sistema é capaz de gerar várias métricas, todas elas inúmeras vezes por minuto as quais têm de ser guardadas eficientemente.

2.5 Trabalho Relacionado

2.5.1 Dynatrace Smartscape

O Smartscape é a ferramenta da Dynatrace que nos permite obter uma visão em forma de grafo de um sistema. Aqui é disponibilizada alguma informação sobre os vários componentes de um sistema em produção, como aplicações, serviços, processos, entre outros. Para além de alguma informação básica sobre cada componente, também é possível visualizar quais os componentes tenham algum problema. Caso estejam a ser utilizados serviços na *cloud* também existe alguma informação sobre quais as regiões e zonas de disponibilidade estão a ser utilizadas [50]. Em cada nó existe também uma opção que nos dá acesso às métricas de cada componente, individualmente. A Figura 10 mostra como o Smartspace mostra toda esta informação, neste caso, para serviços

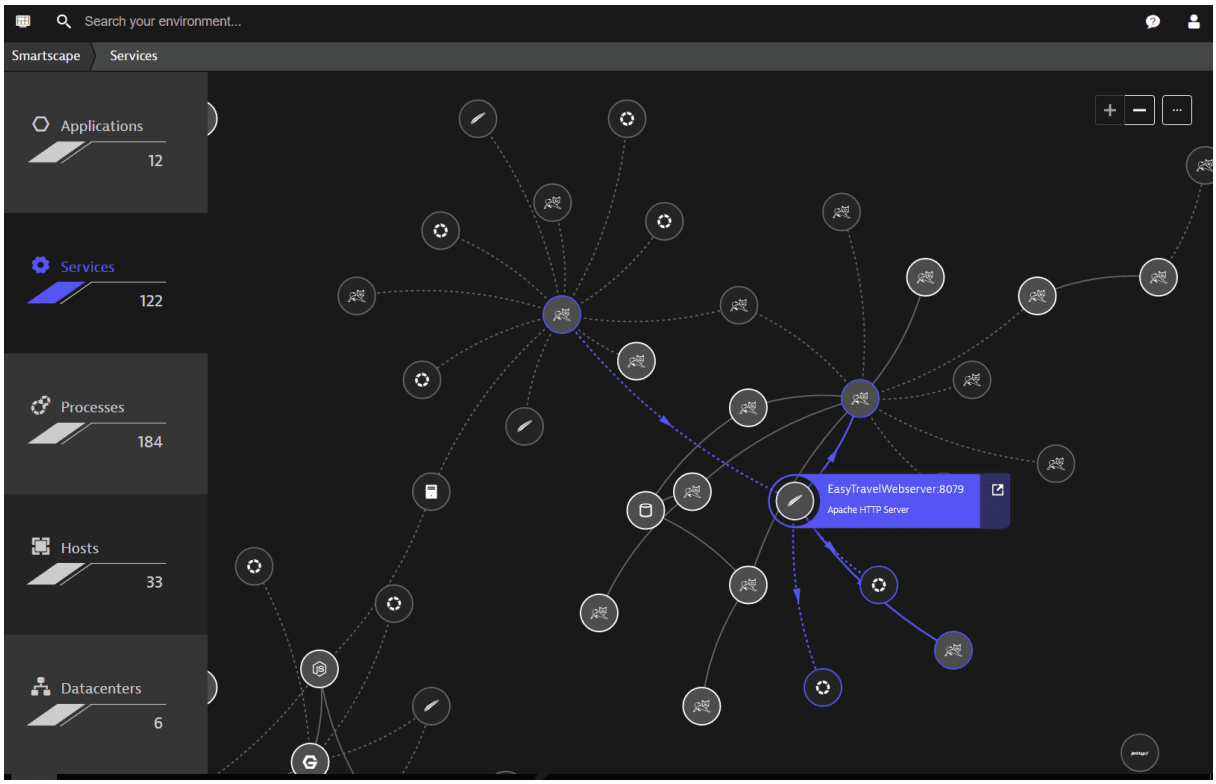


Figura 10 Como o Smartspace mostra serviços de um sistema em forma de grafo

Para além disto, existe ainda a possibilidade de visualizar como cada componente de cada nível do sistema de relaciona entre si, podendo ter uma visualização completa de um sistema e sendo possível identificar facilmente quais dos seus componentes possam ter problemas. É possível ver um exemplo disto na Figura 11, onde neste caso, existem problemas em dois processos necessários a um *website*.

Esta ferramenta está relacionada com o trabalho a desenvolver pois aqui também temos uma representação de um sistema em grafo, no entanto, no trabalho a desenvolver este grafo terá uma organização diferente pois terá de ser feito para sistemas exclusivamente em *cloud* e que irá incluir, por exemplo, regiões geográficas onde se encontra um recurso.

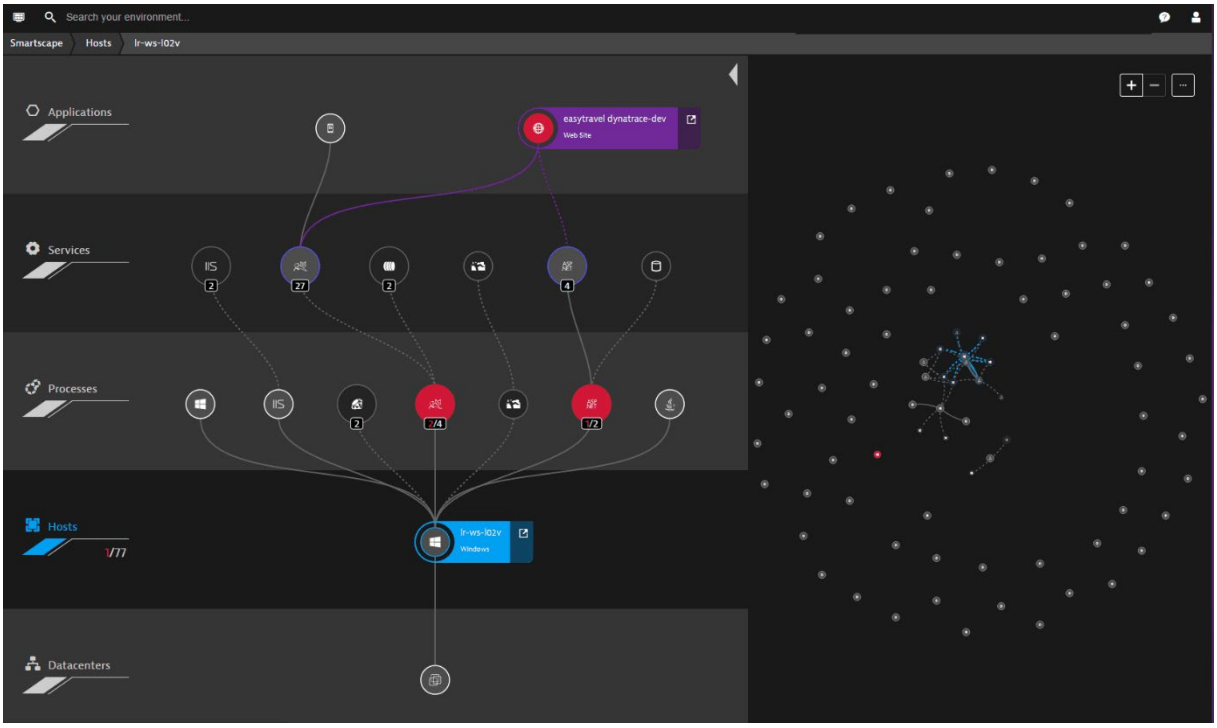


Figura 11 Visualização contendo os vários níveis de um sistema no Dynatrace Smartspace

No entanto, o Dynatrace Smartspace não disponibiliza uma forma para rapidamente ver as métricas de cada componente. No trabalho a desenvolver, para além de ser obtido um grafo com todos os recursos existentes nos três *cloud providers*, vão também ser coletados dados métricos para cada recurso podendo posteriormente ser apresentados de uma forma concisa. Existe ainda um limite de três dias relativo aos dados que são mostrados ao utilizador. Os dados métricos a recolher no trabalho a desenvolver vão ser guardados durante mais tempo visto que se pretende manter um histórico destes dados, fazendo disto um requisito.

2.5.2 Seekret

O Seekret é uma ferramenta que ajuda a construir e implantar APIs diretamente em múltiplos *cloud providers*. É possível ver informações sobre usos e desempenho das várias APIs como mostrado na Figura 12. Para além disto, é também possível visualizar as interações entre

várias APIs e serviços através de um grafo, incluindo também a ordem pela qual são executadas por utilizadores e a arquitetura geral da sua aplicação, a geração automática de modelos de testes e dados para estes para facilitar os testes de APIs e a geração automatica de documentação baseada em dados de tráfego [51], entre outras funcionalidades de gestão, construção e lançamento de APIs.

O Seekret é semelhante ao trabalho a desenvolver no sentido em que ambos recolhem informações de vários *cloud providers*, no entanto, o Seekret especializa-se em APIs e a gestão destas, enquanto que o trabalho a desenvolver deverá suportar a extração e monitorização de vários tipos de recursos na *cloud.*

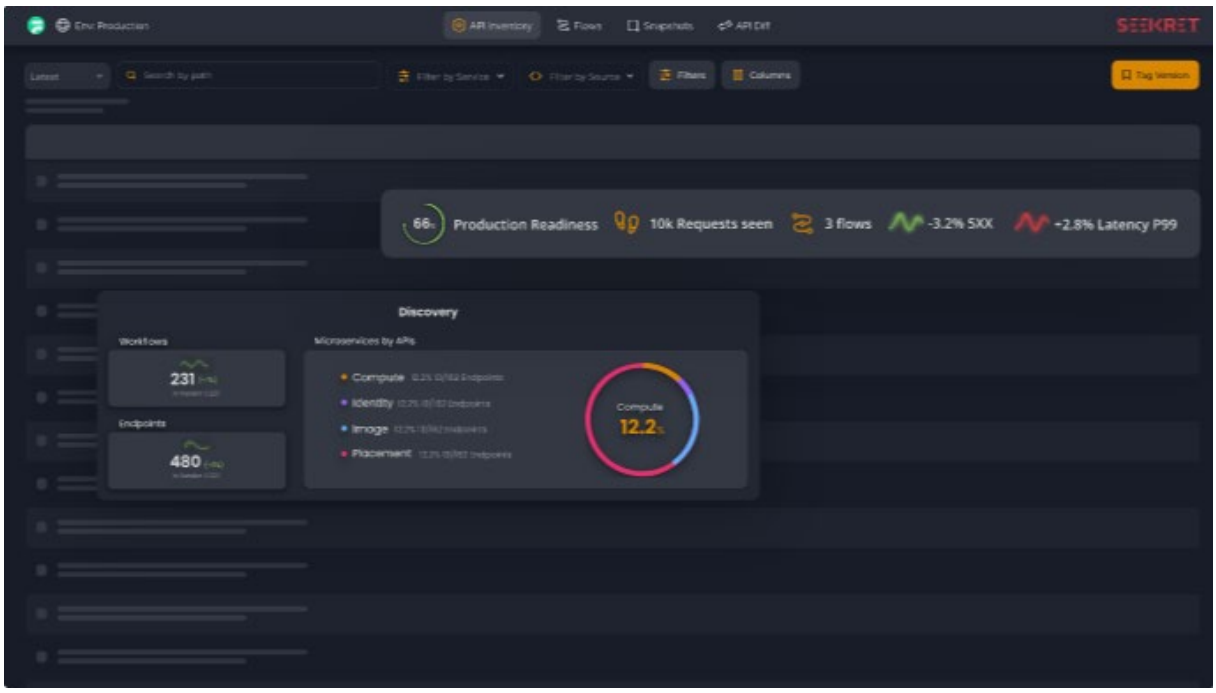


Figura 12 *Dashboard* com informações relevantes de uma API no Seekret

3

Solução Proposta

Para este trabalho pretende-se o desenvolvimento de um protótipo de extração de métricas para os três *cloud providers* (Azure, Google Cloud e AWS). Irá ser necessário extrair não só as métricas, mas também meta dados suficientes para ser possível associar esses dados métricos a um recurso, que neste caso serão dados sobre o recurso em si.

3.1 Modelação de dados

Para solucionar este problema será necessário mantar dois tipos de informação: que recursos existem para uma determinada aplicação e as suas métricas. Como tal, irá ser necessário bases de dados para guardar esta informação.

3.1.1 Modelação de recursos

Para a modelação de recursos irá ser utilizado um grafo *multi-layer*. Na secção 3.1.2 é explicado o porquê do uso de um grafo para guardar os recursos.

Neste grafo será possível encontrar seis camadas diferentes de informação, três relativas à localização dos recursos e outras três relativas à forma como estes estão organizados dentro de cada *cloud provider*. Assim temos as seguintes camadas: Cloud, ResourceGroup, Resource para recursos e Location, Region e AvailabilityZone para as localizações de recursos.

Começando pela localização de recursos, em Location temos a área geográfica onde podemos encontrar várias regiões, como por exemplo, "América do Norte" ou "Europa". A partir daí, podemos ser mais específicos com a localização de um recurso através dos nós Region. Estes mapeiam diretamente para as várias regiões oferecidas pelos *cloud providers* e por fim temos AvailabilityZone que representam as várias zonas de disponibilidade existentes dentro de uma região de um *cloud provider*, normalmente utilizadas por recursos como máquinas virtuais.

Para todos os recursos que não especifiquem uma zona de disponibilidade, cada Region deve ter também um nó extra 'Default Zone'. Existem ainda casos em que os *cloud providers* disponibilizam regiões 'dual-region' e 'multi-region' dentro de uma área geográfica para recursos que possam estar nessas situações logo são também devem ser criados nós para esses casos, um para cada Location que os disponibiliza.

Relativamente aos recursos, temos novamente três camadas. Começando por Cloud, aqui teremos um nó para cada um dos *cloud providers*, Azure, Google Cloud e AWS. Cada um destes relaciona-se com o tipo ResourceGroup. Neste tipo são mapeados os conceitos de *resource group* de cada *cloud provider* para um só tipo, sendo possível a sua uniformização. A Tabela 4 mostra como é feito este mapeamento. Cada nó deste tipo irá conter o nome e identificador de cada *resource group*. Cada *resource group* pode ter zero ou mais recursos. Os nós do tipo Resource, tal como os do tipo ResourceGroup, são mapeados dos conceitos de recurso de cada *cloud provider*. Cada nó do tipo recurso irá conter, pelo menos, um identificador, o seu nome e um tipo. Dependendo do *cloud provider*, podem existir outras propriedades necessárias. Isto poderá ser visto no capítulo 4.1.

A Tabela 4 mostra como é feito este mapeamento e na Figura 13 está representado, de forma simplificada, o grafo que se espera obter.

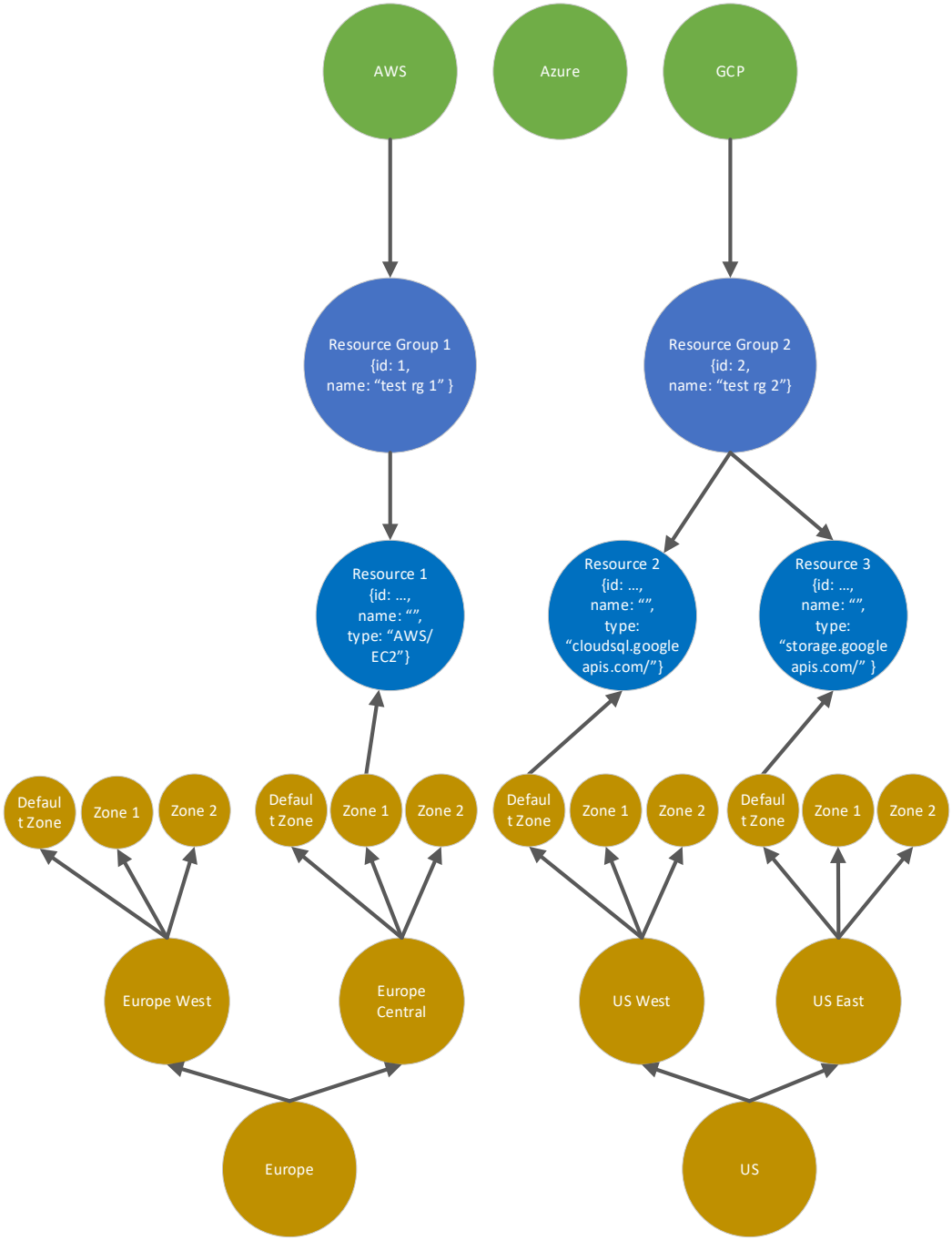


Figura 13 Exemplo de um grafo final gerado pela aplicação

	AWS	GCP	Azure
Resource Group	Resource Group	Project	Resource Group
Resource	Resource	Asset	Resource

Tabela 4 Mapeamento dos conceitos de recursos e grupo de recursos entre os três *cloud providers*

Desta forma é possível ter um modelo que é genérico o suficiente para suportar a organização de recursos e localizações dos três *cloud providers.*

Sobre este grafo, espera-se que as operações de inserção de dados estejam limitadas apenas a nós dos tipos ResourceGroup e Resource. Isto deve-se ao facto de que a restante informação é fixa e deverá ter muito poucas alterações. A única possível alteração para além da adição de *resource groups* e recursos seria a adição de uma nova região a um dos *cloud providers*, com as suas zonas de disponibilidade.

Relativamente a operações de consulta de dados, espera-se que seja possível obter os *resource groups* dentro de um *cloud provider* e os recursos de um *resource group.* Deve também ser possível obter as informações associadas a um recurso como as suas propriedades, *resource group* pai e informações sobre a sua localização, incluindo região e zona de disponibilidade. Também deverá ser possível obter todos os recursos numa determinada região.

3.1.2 Porquê o uso de um grafo

Para guardar a informação sobre os recursos foi utilizado um *multi-layer graph.* Esta foi a solução escolhida pois além de ser possível representar diferentes tipos de informação, tal como já demonstrado pelas figuras (Figura 1, Figura 2 e Figura 3), a distribuição dos recursos na Cloud segue um modelo hierárquico (Figura 14), sendo facilmente representado por um grafo. Isto é, é possível transformar cada componente desta hierarquia diretamente para nós de um grafo e manter as relações entre estes.

Utilizando como exemplo o Azure (Figura 3), é possível transformar *resource groups* e *resources* em nós e manter as relações que têm um com o outro, que neste caso é que um *resource* está contido em um *resource group* e um *resource group* pode contem zero ou mais recursos. O mesmo acontece para o Google Cloud e a Amazon Web Services.

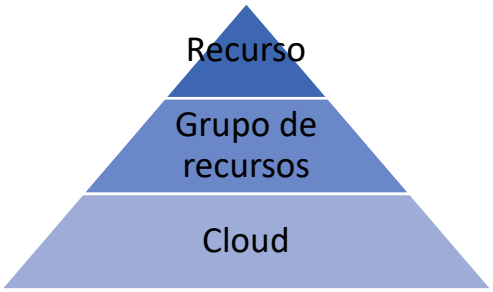


Figura 14 Hierarquia de recursos em *cloud providers*

Outra informação que também é importante guardar é relativa à localização geográfica de um recurso. A razão para isto ser é que quando estamos a analisar o desempenho de um sistema ou até de um recurso em particular a sua localização pode ser utilizada, por exemplo, para despistar certos comportamentos dentro de um servidor de uma região. Também pode ser utilizada para localizar situações em que um recurso ou serviço está sobrecarregado numa

certa região e se existe necessidade de este ser atualizado para poder suportar mais utilizadores e diminuir tempos de resposta ou até a situação inversa, em que temos uma máquina mais que suficiente alocada a um serviço que não tem assim tantos utilizadores e que possa ser reduzida de forma a poupar dinheiro à empresa.

Como tal, o grafo irá também conter informação sobre as regiões geográficas, localizações destes servidores e zonas de disponibilidade disponíveis em cada região. Esta informação também pode ser vista de forma hierárquica. Esta pode ser representada na Figura 15.

Temos uma região geográfica, como Europa ou Estados Unidos no nível mais baixo, uma localização mais concreta no nível do meio (como europa ocidental, por exemplo) e dentro de cada localização podemos encontrar várias zonas de disponibilidade, normalmente três.

Desta forma, tal como aconteceu com os recursos, é fácil converter estes três conceitos em três tipos diferentes de nós do grafo e relaciona-los entre si.

Figura 15 Hierarquia de regiões em *cloud providers*

Como cada recurso tem uma localização ou zona de disponibilidade associado, é por aí que é feita a relação entre ambas as hierarquias.

Podia ter sido utilizado outra forma de estruturar estes dados, tal como uma base de dados relacional, mas a representação dos dados seria menos natural pois já temos os dados que queremos guardar em forma de um grafo (ambas as hierarquias podem ser diretamente representadas por um grafo com várias camadas como já foi explicado) e no caso de termos uma base de dados relacional seriam necessárias pelo menos seis tabelas diferentes e relações entre elas.

Para além disto, retirar informação é também feita de forma mais eficiente quando temos estes dados em forma de grafo, invés de numa base de dados relacional. Por exemplo, no caso onde queríamos retirar todos os recursos dentro de um *cloud provider* seria necessário termos várias junções entre tabelas para também obtermos informação sobre a que *resource group* este recurso pertence, a sua localização e a sua zona de disponibilidade. Ora num grafo, escolhendo um nó de uma cloud, apenas teríamos de seguir as suas relações para encontrar toda esta informação.

3.1.3 Análise de Métricas a serem extraídas

Um dos grandes objetivos deste trabalho é a extração de métricas que reportem valores iguais para os três *cloud providers* e a sua uniformização. Para isso é necessário primeiro analisar os tipos de recursos para que queremos métricas e depois analisar quais as métricas são uniformes entre cada um dos *cloud providers*.

Para os tipos de recursos foi decidido que iriamos tentar obter métricas os seguintes serviços:

- Máquinas virtuais
- Armazenamento
- Funções *serverless*
- Aplicações Web
- *API Gateways*
- Bases de dados

Foram escolhidos estes tipos de recursos visto que foi considerado que estes são os utilizados mais frequentemente. No apêndice A.1 está uma listagem com as métricas observadas em cada *cloud provider* e como as métricas que serão retiradas de cada um.

Depois de termos os tipos de recursos para que queremos retirar as métricas é então necessário averiguar as métricas que são uniformizáveis. Uma métrica uniformizável é uma métrica que reporta a mesma informação nos três *cloud providers*. Isto é, por exemplo, os três *cloud providers* que estão a ser analisados têm métricas de uso de CPU para máquinas virtuais, logo esta é uma métrica uniformizável.

Não é necessário que os três *cloud providers* disponibilizem exatamente uma métrica com o nome exato ou semelhante para esta poder ser considerada uniformizável. Caso seja possível, pode-se derivar a mesma métrica a partir de outra métrica selecionando uma série (ou *dimension*) diferente. Por exemplo, para o armazenamento, o AWS disponibiliza métricas separadas para as operações de *requests* GET, POST, DELETE, etc, enquanto que no GCP e no Azure estas são obtidas através de uma métrica que contem informação sobre todo o tipo de *requests* em que podemos especificar uma série com o tipo de *request* para o tal queremos essa informação.

A Tabela 5 mostra ambos estes casos. Em ambas as linhas temos duas métricas uniformizáveis, a primeira é o primeiro caso referido, em que uma métrica reporta a mesma informação nos três *cloud providers* e o segundo caso mostra o uso de séries para se conseguir obter métricas uniformizáveis.

AWS	GCP	Azure
AllRequests	api/request_count	Transactions
GetRequests	api/request_count (utilizando a série ReadObject)	Transactions (utilizando a série GetBlob)

Tabela 5 Como obter métricas uniformizáveis através da seleção de uma série temporal

As métricas uniformizáveis entre si irão partilhar a mesma coluna numa tabela da base de dados visto que se estará a tratar efetivamente do mesmo tipo de informação.

Para a maioria dos tipos de recursos analisados foram encontradas uma percentagem considerável de métricas uniformizáveis. No entanto, este não foi o caso nos sistemas de bases de dados. Neste caso, as métricas disponíveis diferem muito entre si para serem uniformizadas. Isto deve-se principalmente ao facto de que cada *cloud provider* utiliza sistemas de bases de dados diferentes ou têm o seu próprio sistema. Foi considerado então que para os sistemas de bases de dados, as métricas iriam ser retiradas separadamente para o AWS DynamoDB, Google Cloud Firestore e Azure CosmosDB e não iriam ser uniformizadas na base de dados, isto é, cada sistema irá ter uma tabela com as suas métricas.

É então possível categorizar os tipos de recursos naqueles que são comuns aos três *cloud providers* e em sistemas de bases de dados. Novamente, o apêndice A.1 contém uma listagem das métricas uniformizáveis para cada um dos *cloud providers*, bem como as métricas disponíveis para os sistemas de bases de dados.

3.1.4 Modelação de Métricas

Para armazenar todos os dados relativos a métricas, foi utilizado um sistema de base de dados columnstore. Na secção 3.1.5 é explicado o porquê do uso de um sistema de base de dados columnstore.

Iremos então ter uma tabela por cada tipo de recurso para os quais as métricas sejam uniformizáveis, mais uma para cada sistema de bases de dados de cada *cloud provider*. As tabelas para as métricas comuns aos três *cloud providers* começam com o prefixo *commun,* enquanto que as restantes começam pelos prefixos: az, aws ou gcp dependendo de qual *cloud provider* se está a tratar nessa tabela.

O esquema das tabelas é bastante simples. Existe colunas para o *timestamp* para o qual a métrica foi retirada, o id do recurso para quais as métricas correspondem e informações extra sobre a sua localização e zona de disponibilidade. Um *timestamp* refere-se à data e à hora a que um evento aconteceu. Neste caso, um *timestamp* de uma métrica é uma data/hora a que uma métrica aconteceu.

Para além destas colunas, cada tabela também terá colunas para as várias métricas para o tipo de recurso que essa tabela representa. Estas colunas são também para cada agregação disponível, ou seja, é possível que uma tabela tenha mais que uma coluna para a mesma métrica, mas com agregações diferentes. A nomenclatura utilizada para representar cada métrica tem a forma *nome_da_métrica_agregação*, como por exemplo: cpu_utilization_average.

O tipo de dados dos valores de cada coluna de métricas foi definido para o tipo *float* de forma a permitir métricas que tenham valores que possam ser números não inteiros, como por exemplo percentagens. Estas colunas são também definidas como *nullable* de forma a permitir valores nulos. Isto serve para casos como em que o *cloud provider* não disponibiliza valores de uma métrica para esse *timestamp*, por exemplo, se se tratar de uma métrica que só disponibiliza dados de hora a hora, ou em casos em que simplesmente não houve acessos a tal recurso para existirem métricas, visto que nalguns casos, os *cloud providers* não disponibilizam esses valores em vez de disponibilizarem e mostrarem como zero.

Relativamente à chave primária de cada tabela, esta será a combinação entre as colunas de *timestamp* e do *resource id.* Isto é feito desta forma pois um par destes vai ser único para cada tabela. O *timestamp* por si só não seria suficiente visto que como cada tabela poderá conter vários recursos de vários *cloud providers* teríamos o caso em que iriamos ficar com *timestamps* repetidos, um para cada recurso, logo esta coluna não seria única e como tal, não poderia ser uma chave primária por si só. Algo semelhante acontece se fosse usado apenas o identificador do recurso como chave primária. Nesse caso, iriamos ficar com vários *timestamps* por recurso logo também seria impossível ser usado unicamente como chave primária, logo, optou-se por utilizar uma combinação de ambos. De notar que ao utilizar as colunas relativas à localização também não iria funcionar, visto que, estas vão ser sempre iguais por cada recurso.

Utilizando a tabela para métricas de máquinas virtuais como exemplo, iremos obter uma tabela como a representada na Figura 16:

```
create table common_vm
(
    timestamp   DateTime,
    resource_id String,
    location    String,
    azone       String,
    cpu_utilization_average Nullable(Float32),
    disk_read_bytes_total Nullable(Float32),
    disk_write_bytes_total Nullable(Float32),
    network_in_total Nullable(Float32),
    network_out_total Nullable(Float32)
)
    PRIMARY KEY (timestamp, resource_id)
```

Figura 16 Exemplo de uma *consulta* gerada para uma tabela com métricas de máquinas virtuais

3.1.5 Porquê o uso de uma base de dados *column-oriented*

Uma das grandes vantagens de um sistema de bases de dados *columnstore* para guardar métricas é que os dados vão ser bem compactados pois existirão muitos valores iguais numa coluna. Estas bases de dados também são usadas para o necessário OLAP já explicado na secção 2.4.2.

O caso de guardar dados métricos está dentro do cenário OLAP. Estamos a falar de dados que [17]:

- Normalmente apenas são lidos, mas nunca atualizados. O valor de uma métrica é fixo para um certo *timestamp.*
- Quando são lidos, iremos estar a tratar de uma grande quantidade de linhas, mas poucas colunas. Por exemplo se quisermos obter os dados de uma métrica de um recurso nas últimas doze horas e se os dados estivessem armazenados ao minuto iriamos estar a ler doze multiplicado por sessenta registos, mas apenas três ou quatro colunas, dependendo do esquema da tabela da base de dados.
- As tabelas contêm uma grande quantidade de colunas. Se tivermos uma tabela, por tipo de recurso e uma coluna por cada métrica a ser extraída podemos facilmente ter tabelas com bastantes colunas, especialmente se tivermos também uma coluna por cada agregação disponível para cada métrica.
- Os valores a guardar são pequenos. Normalmente os valores de métricas são numéricos.
- O resultado de uma consulta é normalmente mais pequeno que os dados de origem visto que neste tipo de casos os dados costumam ser filtrados ou agregados. Quando se extrai dados métricos apenas estamos focados num certo período de tempo e não nos seus dados de desempenho desde o seu início.

Como tal, foi decidido que esta seria uma boa solução para armazenar dados relativos a métricas.

3.2 Arquitetura da solução

A solução proposta para este problema envolve quatro componentes: uma base de dados NoSQL baseada em grafos, uma base de dados NoSQL do tipo *column-oriented* e uma aplicação desenvolvida em Python que tem como objetivo a extração de dados sobre recursos existentes nos *cloud providers*, extração de métricas desses recursos e a inserção destes dados num grafo e numa base de dados *column-oriented*, respetivamente para recursos e métricas. Estes

dados sobre recursos e métricas irão ser recolhidos dos três *cloud providers*, Amazon Web Services, Google Cloud e Azure utilizando as suas APIs e SDKs para o efeito.

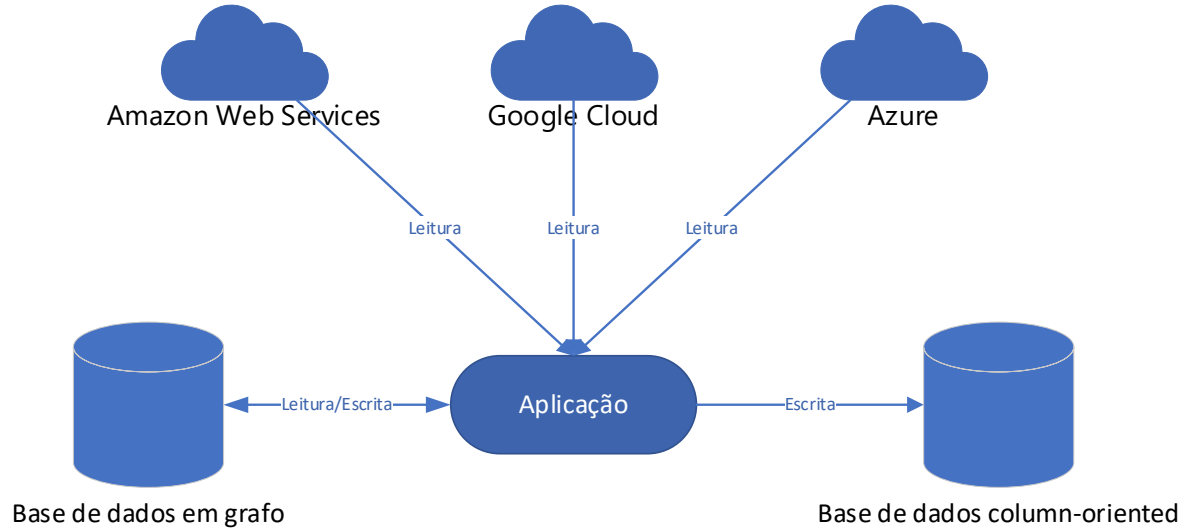


Figura 17 Arquitetura da solução proposta

Começando pela base de dados baseada em grafos, esta irá servir para guardar toda a informação relativa a recursos. Esta informação será, para além do seu nome e identificador no *cloud provider*, outra informação como o tipo de recurso que é e também informação sobre a sua localização como a sua região e zona disponibilidade. Estes recursos estão também organizados por *resource groups*, e estes por si estão organizados por *cloud provider*. O *software* a ser utilizado aqui será o Neo4J.

Teremos também uma base de dados *column-oriented* para guardar os dados relativos a métricas dos vários recursos organizados por tipo de recursos e para aqueles tipos de recursos com métricas uniformizáveis, apenas uma tabela para os três *cloud providers*. O *software* utilizado neste caso foi o Clickhouse.

A aplicação terá como objetivo a extração da informação relativa a recursos e regiões de cada *cloud provider* e armazena-los na base de dados baseada em grafos, construindo assim um grafo com os recursos existentes em cada *cloud provider* e relaciona-los com a sua região dentro deste. A aplicação irá também ser extrair estes dados e processa-los, de modo a obter as métricas. Estes dados métricos vão ser inseridos numa tabela na base de dados *column-oriented* dependendo do seu tipo de recurso.

Os *cloud providers* são onde a aplicação irá retirar as informações sobre os recursos existentes em cada um deles e as suas métricas disponíveis, para os tipos que recursos para os quais iremos recolher as métricas, tal como explicado na secção 3.1.3. As métricas que serão retiradas por tipo de recurso está disponível no apêndice A.1.

3.2.1 Funcionamento geral do sistema

A aplicação desenvolvida segue os seguintes passos:

1. Extração de informação sobre regiões, *resource groups* e recursos para cada *cloud provider*
2. Inserção no grafo da informação retirada sobre regiões, *resource groups* e recursos
3. Retirar informação relativa aos recursos do grafo
4. Extração de informação sobre métricas relativas a cada recurso
5. Inserção na base de dados da informação retirada sobre métricas

Descrevendo cada etapa com mais detalhe, na primeira etapa retira-se a informação sobre as várias regiões disponíveis em cada *cloud provider*. Obtém-se também para cada um deles, todos os *resource groups* e recursos de utilizador. Isto é feito através das APIs e SDKs disponibilizadas pelos *cloud providers*.

Depois de ter esta informação é necessário inseri-la no grafo bem como as relações entre os vários nós a criar. Já foi obtida informação sobre regiões, *resource groups* e recursos, no entanto, existe alguns casos em que é necessário derivar a informação extra sobre regiões visto que nem todos os *cloud providers* disponibilizam a informação necessária, como regiões geográficas e zonas de disponibilidade. É explicado como isto é feito no capítulo 4.1 e também está resumido na Tabela 6. Assim, é possível garantir o mesmo layout para qualquer um dos *cloud providers* mesmo que estes não disponibilizem diretamente toda a informação sobre regiões necessária para a construção do grafo.

Depois de toda a informação ser inserida no grafo é então necessário obter essa informação de volta já estruturada de forma a que possa ser processada para a aquisição de dados métricos. Aqui é feita uma consulta ao grafo para obter, em especial, os vários recursos existentes por cada *cloud provider*. No entanto, esta consulta para além de devolver os recursos, devolve também qual o seu *resource group* e informações sobre a sua localização e zona de disponibilidade para depois poderem ser inseridas na base de dados no último passo. Todo este passo pode parecer irrelevante, pois já temos esta informação antes de a inserimos no grafo no passo anterior, mas é importante pois num sistema em produção não deve ser esperado que o grafo tenha de ser reconstruído sempre que se execute uma extração de dados métricos, sendo assim possível saltar os dois primeiros passos na execução da aplicação e executar apenas a parte relativa à extração de métricas de recursos existentes nos *cloud providers*.

Tendo os recursos, é necessário obter métricas para cada recurso suportado. Por recurso suportado entenda-se um recurso para o qual o seu tipo de recurso é um dos tipos de

recursos selecionados para extração de métricas. Estes foram descritos na secção 3.1.3. Aqui são feitos pedidos às APIs de métricas dos três *cloud providers* utilizando os SDKs disponibilizados por cada um deles para cada um dos recursos suportados retirados do grafo. Estas métricas serão armazenadas numa estrutura de dados para o efeito para serem inseridos no passo seguinte.

Este processo termina depois de ser feita a inserção dos dados métricos obtidos para todos os recursos suportados na base de dados. Este passo é feito no fim de se obter todas as métricas para cada tipo de recurso selecionado de forma a minimizar as inserções na base de dados e assim garantir maior performance, visto que segundo a documentação do ClickHouse, este prefere quando são feitas menos operações de inserção com mais dados do que mais operações de inserção, mas com menos dados, a não ser que estas estejam ordenadas temporalmente [43], o que poderá não ser sempre o caso. Por exemplo, no Google Cloud, a informação é devolvida na ordem inversa. No final, a aplicação termina e podem ser feitas consultas a esta base de dados para obter dados métricos para os vários recursos.

4

Implementação

Para a implementação desta solução foi usado dois sistemas de bases de dados, o Neo4J para a base de dados baseada em grafos e o ClickHouse para a base de dados *column-oriented*. A aplicação foi desenvolvida em Python 3.9 e contém três executáveis, um para cada *cloud provider*. O processo de execução é semelhante entre os três, mas sendo plataformas diferentes, cada um deles irá conter as suas diferenças.

O arranque de cada executável começa por ler um ficheiro que contem a listagem de métricas a ir buscar por *cloud provider* e como estas serão uniformizadas na base de dados. Depois é utilizada esta informação para criar nós base para cada *cloud provider* (nível Cloud) bem como criar as tabelas necessárias para guardar os valores de todas as métricas.

Tendo isto, é então possível começar a guardar informação no grafo. Primeiro, começa-se pelas localizações, visto que estas serão depois necessárias para associar aos recursos. Logo aqui, começa-se a ver algumas diferenças entre os três *cloud providers.*

4.1 Mapeamento de recursos e regiões para Azure, GCP e AWS

4.1.1 Regiões - Azure

No caso do Azure, temos uma função da API que nos dá informações bastante completas sobre as suas localizações incluindo até as coordenadas geográficas do servidor, à exceção de qualquer informação sobre zonas de disponibilidade. Esta função necessita do ID da subscrição que é passado como argumento ao programa. Daqui são retiradas informações sobre as regiões como o seu nome, da propriedade region.name e também a propriedade region.metadata.geographyGroup que nos dá a região geográfica desta (nível mais baixo). Relativamente às zonas de disponibilidade, estas são geradas pelo programa, três para cada região e uma

zona extra para os recursos que não tenham zona de disponibilidade associada, chamada de "Default Zone".

4.1.2 Regiões - Google Cloud

Relativamente ao Google Cloud, aqui temos um funcionamento ligeiramente diferente para a mesma tarefa. Começa-se por ir buscar os *resource groups* (ou *projects*, como são chamados no Google Cloud) visto que é necessário não só um *project* para chamar a função zones.list da API Compute Engine como esta também tem de estar ativa no *project*. A forma como são obtidos os *projects* é explicada na secção seguinte.

Aqui obtemos, para além de outra informação, cada região e URLs para as várias zonas de disponibilidade encontradas em cada região. Para o nome da região temos a propriedade region.name. Para as regiões geográficas foi observado que o nome de todas as regiões é composto pela sua região geográfica e pela sua localização dentro de uma zona geográfica separados por um hífen. Temos como exemplo "australia-southeast1", "eu-west2" ou "us-central1". Podemos assim extrair apenas a parte antes do primeiro hífen para utilizar como a região geográfica, visto que isto origina nomes como "eu", "us" ou "australia" que servem perfeitamente como regiões geográficas e são iguais (ou semelhantes dependendo do caso) aos nomes das regiões geográficas disponibilizados e derivados para o Azure e para o AWS, respetivamente. Para obter os nomes das zonas de disponibilidade corretas é também necessário truncar cada um dos URLs de cada zona de disponibilidade pela última barra, transformando algo como "googleapis.com/compute/v1/projects/testproject-351211/zones/asia-east1-c" para "asia-east1-c". Utilizando como exemplo a zona de disponibilidade "asia-east1-c", é possível dividir isto em três níveis de informação diferentes:

- asia: região geográfica
- asia-east1: região east1 dentro da região geográfica asiática
- asia-east1-c: zona de disponibilidade C dentro da região asia-east1

Novamente, também são criados nós "Default Zone" para cada região para os recursos que não tenham zona de disponibilidade associada. Para o Google Cloud são ainda criadas mais três regiões dentro de cada região geográfica, duas delas para servirem como dual-region, para recursos que possam usar duas regiões e uma multi-region, para recursos que possam usar mais do que duas regiões simultaneamente. Estas estão sempre dentro de uma região geográfica para as regiões geográficas que o suportam.

4.1.3 Regiões - AWS

Já para o AWS, temos outro caso diferente. No AWS cada região é tratada de forma independente entre si, como tal, não é possível obter uma listagem com todas as regiões ou zonas de disponibilidade de uma só vez. Para além disso, é também possível ativar (ou desativar) certas regiões no AWS, portanto também é necessário ter em atenção casos em que um utilizador tenha regiões desativadas.

Dito isto, começa-se por listar todas as regiões incluindo aquelas que são opcionais utilizando a função para o efeito da API EC2. Daqui é retirada a propriedade RegionName para servir como nome da região. Com as regiões, é necessário criar um API *Client* para cada região e só aí, é executado um pedido à função DescribeAvailabilityZones da mesma API que nos dá as zonas de disponibilidade para cada região definida no cliente. Aqui retiramos a propriedade zoneId que é o identificador real da zona, visto que utilizadores diferentes podem ter zonas de disponibilidade associadas diferentes, logo é necessário, por utilizador, transformar o nome da zona de disponibilidade lhe atribuída para o seu identificador real. A Figura 18 mostra como a mesma zona de disponibilidade pode ser diferente para dois utilizadores. Isto é feito de forma a distribuir melhor os utilizadores pelas várias zonas de disponibilidade.

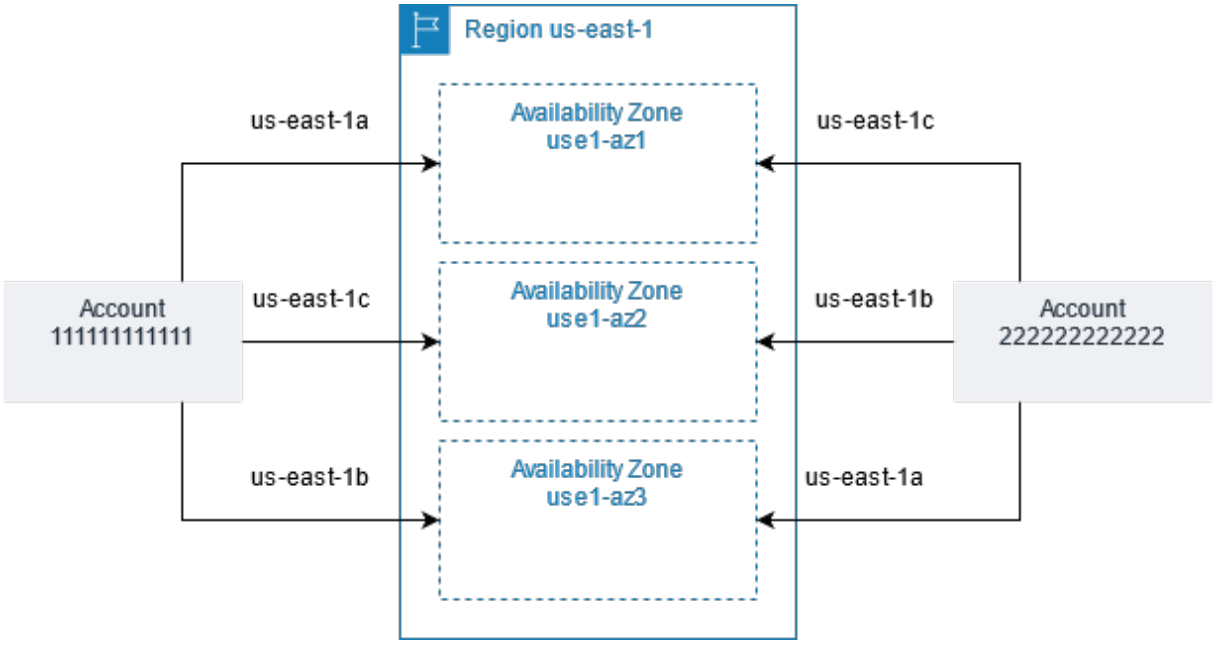


Figura 18 Variação de zona de disponibilidade por utilizador [44]

Para regiões geográficas, temos nomes de regiões semelhantes aos encontrados no Google Cloud, do tipo "eu-west-1", portanto é possível novamente truncar esta string pelo hífen e utilizar apenas a primeira parte. A forma como isto é feito está detalhada na secção anterior. Novamente, são criados nós de "Default Zone" para cada região.

4.1.4 Regiões – Inserção no grafo e conclusão

A inserção dos vários nós relativos a regiões no grafo é feito através de duas *queries* distintas. Primeiro são verificadas todas as regiões em procura regiões pelas várias regiões geográficas. Cada região geográfica ainda não inserida é inserida através da operação MERGE. Ao contrário da operação CREATE, que simplesmente cria um nó e/ou com as propriedades dadas, a operação MERGE primeiro tenta fazer MATCH (operação que tenta corresponder os valores das propriedades especificadas para um nó, ou conjunto de nós) e dependendo se for encontrado um nó ou não, é decidido o que fazer. Neste caso, caso já exista um nó com essa mesma região geográfica este não será criado, caso contrário é então criado. Isto é feito para tentar agrupar o melhor possível as regiões entre os vários *cloud providers*, no entanto, isto será sempre um problema e é discutido no final desta secção.

Depois de inseridas as regiões geográficas, todo o processo é feito para cada região existente em cada *cloud provider*. Primeiro começa-se por gerar a parte da *query* para a inserção das zonas de disponibilidade e relacionam-se estes com os nós da sua região que serão criados antes, na mesma *query*. Antes de criar estes nós de região é primeiro indispensável obter a sua região geográfica de modo a ser possível fazer a relação entre estes nós. Terminamos com uma *query* que efetua todas estas operações pela seguinte ordem, para cada região:

- MATCH do nó da região geográfica
- CREATE do nó da região e da relação com o resultado anterior contento o nó da sua região geográfica
- CREATE para cada zona de disponibilidade, incluindo a zona de disponibilidade "Default Zone" e associação com o resultado anterior contento o nó da sua região

No entanto, tal como foi referido anteriormente, a inserção de regiões tem um problema: não é possível, sem mapeamento manual, associar regiões geográficas corretamente entre os três *cloud providers*. Querendo dizer que apesar de ser possível associar a região geográfica "us" entre os três *cloud providers*, isto não é possível em todos os casos. Isto deve-se a vários fatores, como que à exceção do Azure, estes não disponibilizam informação suficiente sobre regiões geográficas, e mesmo se disponibilizassem, estes poderiam continuar a ser diferentes entre os vários *cloud providers*. Mesmo derivando o nome das regiões geográficas existem algumas inconsistências entre os *cloud providers*, tais como:

- Azure utiliza "Asia Pacific", GCP utiliza "asia" e AWS utiliza "ap".

- Azure e AWS associam as regiões australianas disponibilizadas como "Asia Pacific" e "ap", respetivamente, enquanto que no GCP estas são consideradas como a sua própria região geográfica

Relativamente às regiões, poderia ser possível fazer associações entre elas utilizando a o nome da sua região física (como Londres ou Bruxelas) mas o Google Cloud não disponibiliza esta informação e no AWS esta não é facilmente acedida. Também os diferentes *cloud providers* poderiam utilizar critérios diferentes para estas nomenclaturas.

A Tabela 6 mostra um resumo de como é feita a conversão dos nomes das regiões dos *cloud providers* para regiões geográficas e zonas de disponibilidade.

	AWS	GCP	Azure
Exemplo de nome de região	eu-west-1	eu-west1	westeurope
Região Geográfica	String até ao primeiro hífen	String até ao primeiro hífen	Incluindo na resposta
Zonas de Disponibilidade	Pedido específico	Incluindo na resposta	Geradas pela aplicação

Tabela 6 Resumo de como obter regiões geográficas e zonas de disponibilidade para cada *cloud provider*

4.1.5 Recursos - Azure

Com todas as regiões representadas no grafo é necessário inserir os recursos de um utilizador num *cloud provider*. Começando novamente pelo Azure, para além de necessário efetuar login, ao contrário dos outros *cloud providers*, é também necessário que o utilizador especifique o identificador da sua subscrição visto que isto é necessário para todas as operações dentro do Azure. Para os *resource groups* é utilizado a função Resource Groups – List. Esta função devolve uma listagem com todos os *resource groups* dentro da subscrição especificada, sendo retirado o identificador e nome de cada *resource group* para serem utilizados como propriedades do nó do grafo. Depois, para cada *resource group* é então recolhido os seus recursos.

A função Resources - List By Resource Group requer para além do identificador da subscrição, do nome do *resource group* para o qual estamos a listar os seus recursos. Isto dá-nos a grande maioria das informações necessárias para cada recurso. Ambas estas funções são da API Resource Management.

Para cada recurso retiramos o seu identificador e nome, tal como o tipo do recurso que é e o seu subtipo. O tipo de recurso é utilizado para a aplicação saber qual o tipo de métricas a serem retiradas, que são definidas através de um ficheiro de configuração para cada *cloud*

provider. Já o subtipo de um recurso é utilizado para distinguir alguns tipos de recursos com diferentes métricas disponíveis. Este é o caso do Azure Functions e do Azure Web Apps que ambos utilizam o mesmo tipo de métricas, mas algumas das métricas apenas podem ser utilizados num dos dois tipos de recurso. Acontece algo semelhante para contas de armazenamento Azure Storage em que apenas estamos a obter métricas para contas do tipo StorageV2 para evitar possíveis incompatibilidades como a de se tentar obter uma métrica que não existe para versões anteriores de armazenamento. Caso um recurso não tenha um subtipo definido, esta propriedade é definida como nula.

Também é gerado um identificador único para cada recurso baseado no *hashing* do seu identificador num *cloud provider*. Isto é feito para os três *cloud providers.*

É ainda retirado a região onde reside este recurso. Relativamente à zona de disponibilidade do recurso, esta é apenas necessária para máquinas virtuais, e para isso é utilizado a função Virtual Machines – Get da API Compute. Para esta função é necessário especificar o identificador da subscrição e os nomes do *resource group* e da máquina virtual em questão e é devolvido informações mais completas sobre uma máquina virtual, incluindo a sua zona de disponibilidade.

Outro caso a ter especial atenção é para recursos do tipo Azure Web Apps. Neste caso iremos também querer guardar informação sobre qual é o seu Azure Web App Plan de forma a ser possível recolher informações adicionais sobre uma aplicação web, tal como a sua utilização de CPU. Para tal, é necessário recolher a outra função de outra API. Para este caso é necessário a função Web Apps – Get da API App Service. Daqui, tal como no caso anterior, é necessário especificar o identificador da subscrição bem como os nomes do *resource group* e da aplicação web. O que são devolvidas são informações mais completas sobre a aplicação web especificada, tal como o seu Web App Plan.

Nem todos os recursos estão a ser incluídos no grafo. Recursos que não estão associados a uma região (que têm a propriedade location definida para "global" ou não definida) não são adicionados pois normalmente estes recursos não são essenciais na perspetiva de APM.

4.1.6 Recursos – Google Cloud

Passando ao Google Cloud, novamente, primeiro é necessário obter os *resource groups* (ou *projects*, no GCP) e só depois os recursos (ou *assets*, no GCP) existentes em cada um deles.

Começando pelos *projects,* estes são obtidos através da função projects.search. Para esta função não é necessário especificar qualquer argumento (apesar de ser possível especificar um filtro) e obtemos assim a listagem de todos os projetos existentes na conta Google Cloud para qual o utilizador fez login. Aqui novamente vamos buscar duas propriedades, project_id e display_name, que irão corresponder diretamente com as propriedades id e name de um nó

ResourceGroup no grafo. É também depois de obtermos os *projects* que se obtém as regiões existentes no GCP, processo descrito na secção anterior.

Tendo os *projects* é necessário obter todos os seus *assets*. Para isto é utilizado a função assets.list da API Cloud Asset Inventory. Aqui especificamos o *project* para o qual queremos os *assets* e também se define o campo *contentType* para *RESOURCE*. Isto é necessário para reduzir o número de informação na resposta visto que apenas estamos interessados em recursos e não noutros elementos que possam ser considerados *assets* pelo Google Cloud como por exemplo configurações de IAM (Identity and Access Management, ou gestão de identidade e acessos em português).

Desta resposta é retirada o identificador, nome e o tipo de *asset* de cada *asset.* Tal como no Azure, este tipo de *asset* irá ser utilizado para se saber que recurso estamos a tratar aquando a extração de métricas. Caso um *asset* seja uma base de dados Firestore, é retirado também o seu subtipo, que neste caso será se está a correr em modo nativo, ou em modo de compatibilidade com o Datastore. Isto é necessário pois tal como também aconteceu no Azure para outros tipos de recursos, aqui apenas são extraídas métricas para instâncias Firestore a correr em modo nativo visto que estes modos utilizam métricas completamente diferentes entre si.

Relativamente à região e zona de disponibilidade de um *asset*, ao contrário de no Azure, quando um *asset* tem uma zona de disponibilidade associada, esta está logo incluída no campo com a região sendo denominada por uma letra. É então necessário truncar os dois últimos caracteres desta string para obter o nome da região sem a zona de disponibilidade (que são um hífen utilizado como separador e a letra da zona de disponibilidade). Um exemplo de um valor desta propriedade seria "eu-west1-a". Caso um *asset* não tem uma zona de disponibilidade associada, retira-se apenas a sua região. Esta verificação é feita através da condição se o penúltimo caracter é um hífen, então tem uma zona de disponibilidade, caso contrário, não tem. Um exemplo de um nome de região sem zona de disponibilidade seria "eu-west1".

Existe ainda um caso em que *assets* do tipo aplicação web podem ter a sua localização definida como "europe-west" ou "us-central", no entanto, segundo a Google [45], estas regiões devem ser tratadas como "europe-west1" e "us-central1" respetivamente, portanto este caso tem também de ser verificado.

Relativamente a *assets* que não são adicionados ao grafo, estes são os *assets* sem localização associada ou com a localização "global", tal como no Azure. Também são ignorados recursos do tipo Subnetwork, pois existe um para cada região. Para além destes, ServiceAccounts, ou contas de serviço, que são usadas para poder gerir recursos sem ser necessário efetuar login com a conta Google e imagens do ContainerRegistry, usadas para a criação de máquinas virtuais também não vão ser adicionados ao grafo pois estes tipos de recurso não são uteis na perspetiva de APM.

4.1.7 Recursos - AWS

Para a Amazon Web Services, como está tudo dividido por regiões, obter os *resource groups* e recursos, envolve fazer pedidos separados para todas as regiões que se encontrem ativas. Para obter os *resource groups* de uma região é utilizado a função ListGroups da API Resource Groups. Daqui vão sair as propriedades GroupArn e GroupName que correspondem às propriedades id e nome no nó do grafo.

Com os *resource groups* é possível obter os recursos existentes em cada um deles com a função ListGroupResources da mesma API. É necessário especificar o *resource group* para qual queremos listar os recursos. Nesta função apenas é devolvido, para cada recurso encontrado, o seu tipo e identificador na forma de um Amazon Resource Name.

Este Amazon Resource Name é uma string com o seguinte formato: "`arn:partition:service:region:account-id:resource-type/resource-id`". Daqui é possível retirar o seu nome de recurso através da propriedade *resource-id*. Também podemos obter a região a partir do campo *region* mas como os recursos no AWS já estão divididos por regiões, todos os recursos que forem obtidos num pedido estão dentro da região selecionada.

Relativamente à zona de disponibilidade, é verificada para o caso em que o tipo de um recurso é uma máquina virtual. Tal como aconteceu no Azure, aqui também é preciso utilizar a API relacionada com máquinas virtuais para obter esta informação. Como tal, é utilizado a função DescribeInstances da API EC2. Aqui obtemos o nome da zona de disponibilidade em que se encontra a máquina virtual selecionada no pedido. Este nome vai ainda ser convertido para o identificador real da zona de disponibilidade antes de ser atribuído à propriedade do nó do grafo.

Para recursos do tipo S3 Bucket, é também retirada a informação relativa a filtros através da função ListBucketMetricsConfigurations da API S3. Isto é necessário pois algumas das métricas de armazenamento relacionadas com os tipos de *requests* feitos e respostas obtidas necessita de um filtro especificado. Caso este filtro não exista, este será criado a este ponto através da função PutBucketMetricsConfiguration. Estes filtros podem afetar todos os ficheiros existentes nesse *bucket* (ou pasta de armazenamento) ou apenas certos ficheiros. Para este caso apenas estão a ser considerados filtros que afetem todo o *bucket.* Esta informação é guardada no formato JSON numa propriedade adicional nos nós dos recursos no grafo.

Já para aplicações web do tipo ElasticBeanstalk é também necessária alguma informação extra. Semelhante ao Azure, aqui também temos um *environment* (ou ambiente) onde está a correr esta aplicação. Para obter a informação sobre o ambiente é necessário utilizar a função DescribeEnvironments da API ElasticBeanstalk. Esta função é utilizada em conjunto com um filtro que especifica o nome da aplicação web para o qual queremos o *environment*, caso contrário são devolvidos todos os *environments* disponíveis. Do resultado desta função é retirado o seu identificador e nome.

Depois de obtido o *environment*, é necessário obter algumas definições deste com a função DescribeConfigurationSettings da mesma API. Esta função devolve todas as definições para um determinado *environment*. O que é importante retirar desta resposta é o valor da definição EnvironmentType. Esta definição indica o tipo de *environment* em que a aplicação está a executar e pode ser de dois tipos: Single Instance (apenas uma réplica da aplicação web está a executar) ou Load Balanced (podem executar várias réplicas da aplicação web de modo a prevenir falhas e melhor o desempenho).

Para ambos os casos são necessários obter os seus EC2 Auto Scaling Group. De forma muito simplificada, o que este tipo de recurso faz é automaticamente aumentar ou reduzir o número de instâncias de máquinas virtuais para um certo serviço consoante o número de utilizadores concorrentes de modo a ser possível obter melhor performance e menores gastos [46, 47]. Para obter o Auto Scaling Group de um *environment* é invocado a função DescribeAutoScalingGroups da API AutoScaling. Novamente, aqui é utilizado um filtro para obtermos apenas o Auto Scaling Group associado a este *environment*. Isto é feito através do sistema de Tags (que também é utilizado para o agrupamento de recursos em grupos de recursos). O ElasticBeanstalk ao criar todos os componentes para o funcionamento de uma aplicação web, associa-lhes uma tag com a chave elasticbeanstalk:environment-id e com o valor sendo o seu identificador do *environment*. Depois de obtido o Auto Scaling Group associado é retirado o seu nome e identificador. Também, para obter a métrica relacionada com a latência da aplicação web é necessário retirar o nome do Load Balancer alocado.

Esta informação é necessária para obter as métricas de uma aplicação web pois parte das métricas vão ser obtidas através dos Auto Scaling Groups, outra parte das métricas é obtida através do nome do *environment* e a sua latência através do seu Load Balancer. Toda é informação é guardada no formato JSON, tal como ocorreu para os *buckets* de armazenamento.

4.1.8 Recursos – Inserção no grafo

Com todos os dados para *resource groups* e recursos devidamente organizados, estes são então inseridos no grafo. As propriedades relacionadas com a localização de um recurso serão utilizadas para criar as relações entre os nós destas e os recursos em si. Tudo o resto será propriedades dos nós de recursos. Isto é feito da forma seguinte.

Como a informação relativa a *resource groups* irá estar a ser guardada numa lista de contem objetos que os representam e como cada objeto deste tipo contem uma propriedade que contem uma lista de todos os recursos presentes neste *resource group* é possível iterar por *resource group* fazendo a sua inserção através de uma operação de CREATE e associando ao *cloud provider* correto utilizando uma operação do tipo MATCH com CREATE para a criação desta relação.

Para cada *resource group*, irá também ser necessário iterar os seus recursos e, novamente, utilizar uma operação do tipo CREATE para cada recurso e de seguida fazer o seu MATCH e CREATE com ambos o *resource group* e a zona de disponibilidade de uma região correta para a criação de ambas esta relações.

É de notar que neste caso ambas estas inserções são feitas com duas queries (CREATE para criar o nó e MATCH seguido de CREATE para criar a relação) no entanto é possível isto ser feito com apenas uma query à base de dados, neste caso, teríamos apenas um MATCH e um CREATE em que o nó é criado ao mesmo tempo que as relações entre nós em vez de forma separada de forma semelhante ao que foi feito para as regiões, exposto na secção 4.1.4.

4.1.9 Extração de Recursos do grafo

Para extrair a informação relevante do grafo para a extração de métricas é possível utilizar uma *query* como a representada na Tabela 7.

```
MATCH (c:Cloud {name: 'CLOUD_NAME'})-[*]->(rg:ResourceGroup)-[*]-
>(r:Resource)<-[*]-(az:AvailabilityZone)<-[*]-(region:Region)
RETURN rg, r, region, az
```

Tabela 7 Consulta à base de dados para obter todos os recursos existentes num *cloud provider*, com o seu *resource group,* região e zona de disponibilidade

Esta consulta ao grafo irá devolver todos *resource groups*, recursos e informações sobre a região e zona de disponibilidade para cada um destes, para o *cloud provider* indicado em `CLOUD_NAME` que poderá ser Azure, GCP ou AWS.

4.2 Extração e inserção de métricas para Azure, GCP e AWS

Depois de obtidos todos os recursos a aplicação irá verificar todos os tipos de recursos para os quais queremos métricas definidos nas configurações de cada *cloud provider*. Para cada tipo irá verificar todos os recursos extraídos e caso ambos os tipos de recursos (e subtipos, caso estejam definidos) sejam iguais, serão então extraídas todas as métricas de todos os recursos por tipo de recurso.

Em pseudocódigo, o que ficamos é com algo como demonstrado pelo seguinte (simplificação sem subtipos e múltiplas agregações a serem consideradas):

```
For Each resource_type in cloud_configuration:
      For Each resource in extracted_resources:
            If resource.type == resource_type:
                  For Each metric in resource_type[‘metrics’]:
                        extract_metric(metric)
            Else:
                  Continue
```

Aqui consideramos cloud_configuration como uma lista com os vários tipos de recursos com as suas informações e um vetor ‘metrics’ com a listagem de todas as métricas e suas informações a retirar por *cloud provider*. Já a variável extracted_resources representa todos os recursos extraídos do grafo com a consulta apresentada no subcapítulo anterior. É então feita uma comparação entre ambos os tipos e caso esta seja igual, são então extraídas estas com base nas especificações destas contidas no vetor ‘metrics’.

Como já foi discutido, a extração de métricas é também feita de forma diferente por cada *cloud provider*. Neste caso, o método extract_metric() será diferente por *cloud provider.*

As métricas a extrair estão definidas num ficheiro de configuração para cada *cloud provider*. Este ficheiro contem, para cada tipo de recurso, as métricas a extrair, e a informação necessária para a sua extração e inserção correta na base de dados.

Para cada recurso as métricas extraídas são guardadas dentro de um dicionário com chaves do tipo *timestamp* e valores do tipo dicionário com chaves e valores do tipo string. Isto é feito desta forma pois assim teremos, para um certo *timestamp,* todas as métricas com todos os seus valores, de forma a facilitar a sua inserção na base de dados. Cada item do dicionário de dentro corresponde diretamente a uma coluna da base de dados e ao seu valor. Em pseudocódigo isto será como demonstrado pelo seguinte:

```
Dictionary<Timestamp, Dictionary<String, String>>
```

A Tabela 8 mostra como esta estrutura de dados pode ser utilizada para facilitar a inserção de dados na base de dados. Chamemos ao dicionário exterior de *DictTimestamps* e ao interior de *DictData*.

Timestamp	*DictData*: *Chave #1*	*DictData*: *Chave #2*
DictTimestamps: Chave #1	*DictTimestamps Value #1 -> DictData*: *Value #1*	*DictTimestamps Value #1 -> DictData*: *Value #2*
DictTimestamps: Chave #2	*DictTimestamps Value #2 -> DictData*: *Value #1*	*DictTimestamps Value #2 -> DictData*: *Value #2*
DictTimestamps: Chave #3	*DictTimestamps Value #3 -> DictData*: *Value #1*	*DictTimestamps Value #3 -> DictData*: *Value #2*

Tabela 8 Representação em forma de tabela da estrutura de dados utilizada para guardar os dados métricos

4.2.1 Métricas - Azure

Começando pelo Azure, aqui é utilizado a função Metrics – List da API Monitor. Esta função irá receber o identificador do recurso, qual a métrica a ser extraída, um *timespan* (intervalo de tempo) para o qual queremos retirar as métricas, as agregações especificadas na configuração, e caso necessário, um filtro. Este filtro é criado com base na configuração da métrica e normalmente é utilizada para especificar uma certa série temporal de dados. Não é especificado a granularidade dos dados, portanto é utilizado um minuto por defeito. Isto vai resultar numa série temporal com os dados a inserir na base de dados.

Também para o Azure, relativamente a aplicações web, enquanto que ambos o Google Cloud e o AWS disponibilizam logo a métrica de uso de processador, aqui é necessário obter esta métrica a partir de um plano de aplicação web. Cada aplicação web tem associado um plano de aplicação web. Os dados desta métrica são guardados numa tabela à parte visto que este é um tipo de recurso diferente. A tabela que guarda os dados das aplicações web terá também uma coluna extra que indica qual o plano de aplicação web associado (o seu identificador) que poderá ser usado para retirar informações sobre métricas do plano a que está associado, que neste caso é apenas a utilização de processador.

A inserção destes dados é feita sempre que é detetado uma alteração nas chaves do dicionário interior. Isto deve-se ao facto que que quando existem falta de dados, por exemplo, quando é aplicado um filtro ou quando a métrica que estamos a tentar retirar não é atualizada a todos os minutos (por estarmos a utilizar granularidade a um minuto e a métrica que queremos só está disponível de cinco em cinco minutos) isto irá criar uma variação das colunas que são usadas para a inserção dos dados na base de dados. De notar que não são inseridas linhas que apenas tenham valores nulos ou que não tenham qualquer valor numérico. Isto é valido para os três *cloud providers* analisados.

4.2.2 Métricas – Google Cloud

No caso do Google Cloud é utilizado a função projects.timeSeries.list da Cloud Monitoring API. Esta função, tal como no Azure necessita de vários parâmetros. Aqui, ao invés do nome do recurso, é especificado o nome do *resource group* (ou *project*) e o nome do recurso apenas necessário em alguns casos e é especificado no filtro. Também, dependendo do tipo de recurso, poderá ser necessário filtrar por um elemento diferente o que fez com que houvesse necessidade de especificar qual a propriedade necessária para filtrar por tipo de recurso nas configurações da métrica. Para além disso, também é possível utilizar este filtro para selecionar séries tal como Azure e isso foi utilizado para algumas métricas. Também é especificado um *timespan* de uma hora e definido a propriedade *view* para FULL de forma a obtermos os valores das métricas e não apenas a informação sobre as métricas que vamos encontrar na resposta sem os seus valores.

Relativamente às agregações, o Google Cloud funciona de forma diferente que o Azure. Os dados que são devolvidos não estão agregados, sendo depois possível especificar a forma como os dados da resposta são organizados. Isto é composto por duas partes. Primeiro é necessário alinhar os dados das séries temporais através de um *aligner* e de um período. Para o período foi utilizado sempre valores de um minuto para corresponder com os valores retirados do Azure, como tal, também pode ser visto como a granularidade dos dados. Para o *aligner* foi utilizado o modo de soma, isto deve-se ao facto de mesmo sem *aligner* os dados já estão por períodos de um minuto, portanto este torna-se quase irrelevante. Se utilizássemos outra granularidade temporal, poderiam aqui ser feitas operações como médias, máximas, mínimas, etc definindo um *aligner* próprio para cada caso.

O que é importante neste caso é a escolha um redutor (ou *reducer*) para combinar as várias séries temporais numa só. Isto é especialmente importante para ser possível obter métricas como contagens que estão divididas por várias séries temporais e onde queremos o total das várias séries. Estes não têm necessariamente de ser somas, podem também ser médias entre as várias séries disponíveis, ou outras operações, tal como no caso dos *aligners*. Estes *reducers* estão configurados para cada métrica que necessita destes.

Tal como acontece com o Azure, quando não existem dados, ou dados incompletos para um período temporal de uma métrica, o GCP poderá ou não devolver nada, ou devolver apenas essa parte dos dados existente que nem sempre será consistente entre várias métricas, respetivamente para cada caso. Como tal, sempre que é detetado uma variação de colunas (chaves do dicionário interior) será feita logo a inserção nesse ponto. Assim não é necessário que as colunas estejam sempre consistentes e com um valor, mesmo que este seja nulo, para cada *timestamp,* visto que ao inserir as métricas, se é detetado uma variação entre colunas, estas são logo inseridas não causando assim erros durante a inserção devido ao número de colunas a inserir para um linha não difira das colunas que vão ser inseridas na *query*.

4.2.3 Métricas - AWS

Terminando com o AWS, a extração de métricas é feita com recurso à função GetMetricData disponibilizada pela API do CloudWatch, que é o serviço da Amazon que trata de métricas, registos, entre outros. Esta função recebe, para além de datas de início e fim para os quais queremos a informação, também podemos especificar um conjunto de métricas. Cada métrica requer, para além do seu nome, que seja especificado o seu *namespace*, a estatística em qual queremos que as métricas sejam representadas e um período.

Também são especificados filtros, dependendo que qual métrica se esteja a tratar. Ao contrário do GCP, aqui não é possível utilizar operadores lógicos do tipo 'ou' num filtro, sendo então necessário retirar a mesma métrica, mas utilizando filtros diferentes e depois utilizar uma função estatística para chegar ao resultado pretendido, de forma semelhante a um *reducer* no GCP, mas sendo feita após terem sido retiradas as séries temporais necessárias. Outra alternativa seria utilizar uma expressão matemática (utilizando uma sintaxe baseada em SQL) em vez de uma especificação de uma métrica, no entanto apenas é possível o uso de expressões matemáticas para obter as últimas três horas de informação [48] e como tal, foi considerado como opção não ideal para o contexto deste trabalho.

Um exemplo de onde esta estratégia é utilizada é a extração da métrica "BucketSizeBytes" do serviço S3. Esta métrica reporta o tamanho ocupado por um tipo de armazenamento num *bucket* no serviço S3. Como é possível coexistirem diferentes ficheiros com diferentes estratégias de armazenamento no mesmo *bucket*, não podendo utilizar um operador lógico do tipo 'ou' e o facto de ser necessário especificar um tipo de armazenamento para obter os dados métricos, isto faz com que seja preciso recolher os dados para cada um deles individualmente e, neste caso, somá-los no final. Também relativamente a filtros e ao serviço S3, como já foi referido aquando a extração dos recursos do AWS, para as métricas relativas a pedidos a um *bucket*, é necessário especificar um filtro adicional para o efeito.

Outro caso a ter em atenção é a extração de métricas para recursos do tipo ElasticBeanstalk, ou aplicações web. Aqui é necessário o uso de alguma da informação que foi retirada anteriormente. Para as métricas que reportam dados sobre o uso da máquina, como uso de CPU e E/S estes são obtidos pelo *namespace* do EC2 e usam como filtro o nome do Auto Scaling Group antes identificado. Já as métricas que reportam informação sobre o seu *environment* como o número de pedidos, são feitos através do próprio ElasticBeanstalk e usam o identificador do *environment* como filtro, no entanto, estas métricas têm de ser ativadas manualmente nas configurações do ElasticBeanstalk para que apareçam no CloudWatch, e consequentemente, para que possam ser extraídas. Já para a latência é necessário o seu Load Balancer. Caso a aplicação web esteja a correr em modo Single Instance não é possível obter esta

métrica visto que apenas está disponível para Load Balancers e em modo Single Instance não é necessário um Load Balancer para a aplicação web.

Para as métricas com múltiplas agregações a retirar, ao contrário de no Azure onde é possível especificar várias agregações no mesmo pedido, aqui é preciso fazer múltiplos pedidos, um para cada agregação que seja necessária.

A inserção das métricas na base de dados é feita por recurso, à semelhança do que é feito para o Azure e Google Cloud, descrito nas secções 4.2.1 e 4.2.2.

4.2.4 Métricas - Conclusão

Caso exista necessidade, todas as métricas são convertidas para a mesma unidade visto que diferentes *cloud providers* podem representar métricas uniformes com unidades diferentes, como é o caso da métrica que nos dá o tempo de execução de uma função *serverless*. Esta está representada em milissegundos no AWS, nanossegundos no GCP e em segundos no Azure. Para este caso específico estão a ser convertidas para milissegundos. Isto é algo que também está definido nas configurações de cada métrica.

No final de um recurso ser processado, são inseridos todos os valores das métricas na base de dados. A *query* necessária para inserir os dados é gerada através dos valores existentes dentro do dicionário conforme representado na Tabela 8. Também são inseridas informações sobre o recurso em questão, como o seu identificador, região e zona de disponibilidade.

Obtemos então várias tabelas com as métricas de vários recursos, tudo devidamente organizado em que para tipos de recursos com métricas comuns aos três, existe uniformização. Desta forma obtém-se tabelas com valores de vários *cloud providers*. No caso em que as métricas não são uniformizáveis, existe uma tabela para cada tipo de recurso, para cada *cloud provider*, como é o caso para os vários sistemas de bases de dados.

5

VALIDAÇÃO DE RESULTADOS

Para a validação de resultados, irá ser necessário verificar se os dados guardados correspondem aos dados retirados dos *cloud providers* e se estes são organizados da maneira correta. Para o caso dos recursos e regiões, isto implica averiguar se todos os nós e as relações entre estes são feitas corretamente. Já para os dados métricos, é importante confirmar se estes estão a ser inseridos nas tabelas e colunas corretas e se os seus valores correspondem aos que são representados dentro das ferramentas dos próprios *cloud providers*. Estes testes foram feitos primeiro de forma individual para verificar se os dados estão corretos, e depois, são testados todos em simultâneo de forma a simular um serviço multi-cloud.

5.1 Validação de resultados para Azure

Para o Azure teremos vários *resource groups*, com vários recursos em várias regiões. A Tabela 9 mostra um resumo destes recursos:

Resource Group	Recurso	Tipo	Região
CJ_CUSTOM	cj-cosmosdb	CosmosDB	UK West
	cj-testfunction	Function App	North Europe
	cjcustom9f75	Storage	North Europe
	cjnewstorage	Storage	West Europe
	cjpythonapp	App Service	West Europe
	vmtest	Virtual Machine	North Europe
	ASP-CJCUSTOM-8e13	App Service plan	West Europe
	ASP-CJCUSTOM-9fb6	App Service plan	North Europe
	CJ-TestAPI	API Management	North Europe
CJ-testapp-windows_group	CJ-testapp-windows	App Service	West Europe
	ASP-CJtestappwindowsgroup-870c	App Service plan	West Europe
CJ_ACADEMY_ENV0	cjpwshell	Storage	West Europe
vmdefault_group	vmdefault	Virtual Machine	West Europe

Tabela 9 Recursos presentes no Azure

Para além destes recursos, existem também outros recursos, no entanto, estes são os principais para os quais são retiradas as métricas.

Após o grafo ser gerado pela aplicação, utilizando a *query* descrita na secção 4.1.8, é possível um utilizador utilizar a consulta seguinte para obter um grafo como o da Figura 19. Este não é o grafo completo que é gerado pela aplicação e apenas contem nós de localizações com pelo menos um recurso associado. Este também é a mesma consulta utilizada para obter os grafos com os dados correspondestes aos outros *cloud providers* nas secções 5.2 e 5.3

```
MATCH (g:Location)-[*]->(re:Region)-[*]->(az:AvailabilityZone)-
[*]->(r:Resource)<-[*]-(rg:ResourceGroup)<-[*]-(c:Cloud)
RETURN r,rg,c,az,re,g
```

Este grafo irá contém os seguintes nós, que podem ser divididos tal como enunciado na secção 3.1.1:

- 1 do tipo Cloud, a vermelho, que neste caso apenas será o nó correspondente ao Azure
- 1 do tipo Região Geográfica, a lilás
- 3 do tipo Região, a laranja

- 5 do tipo Zona de Disponibilidade, a azul
- 8 do tipo ResourceGroup, a bege
- 39 do tipo Resource, a verde

Depois de feita a extração das métricas, iremos ficar com as seguintes tabelas, com o tipo de recurso presente nestas entre parenteses:

- common_apigateway (API Management services)
- common_vm (instâncias de máquinas virtuais)
- common_function (funções *serverless*)
- common_storage (*buckets* de armazenamento)
- common_webapp (App Services)
- az_cosmosdb (bases de dados do CosmosDB)
- az_webappplan (App Service plans)

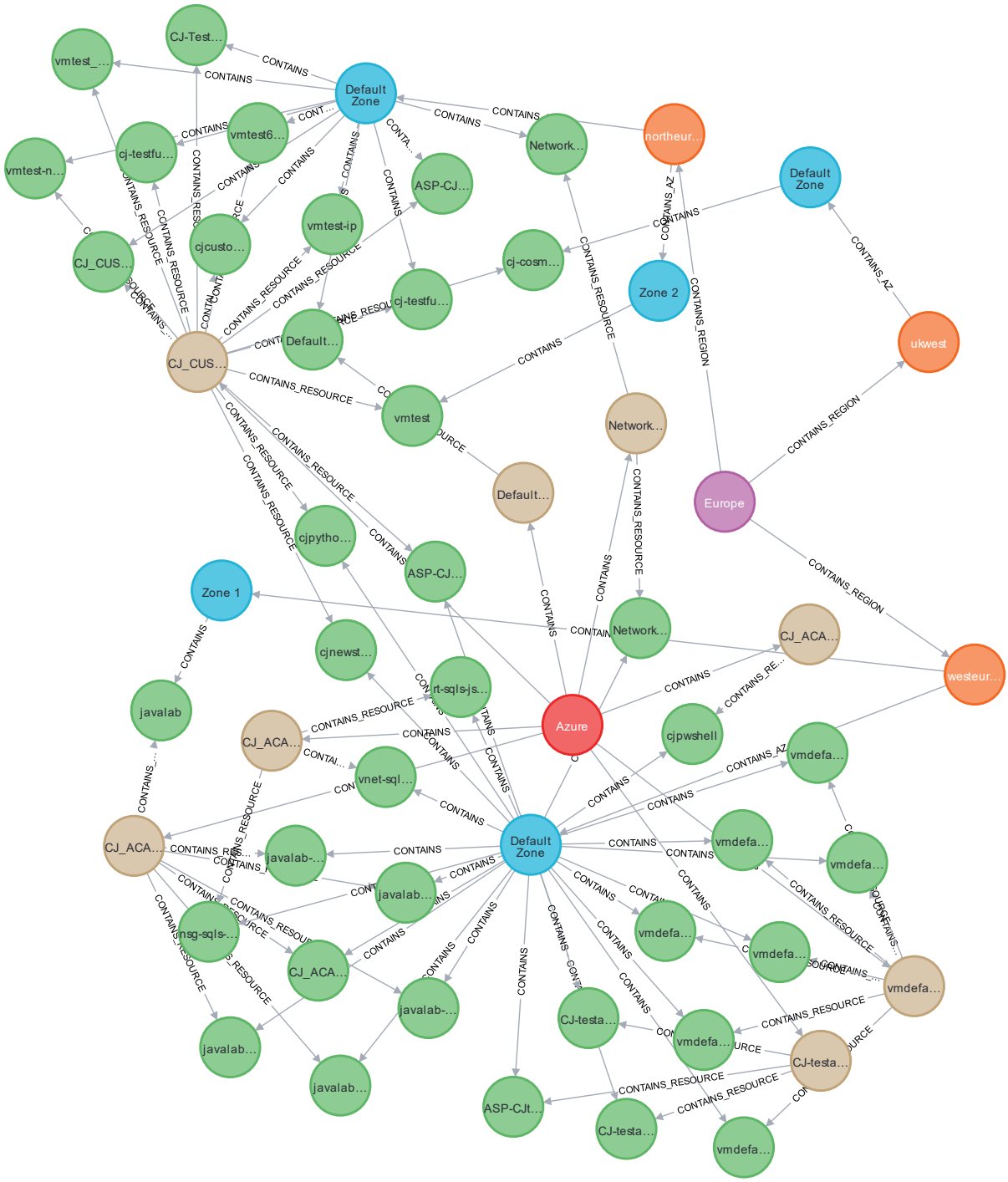


Figura 19 Grafo obtido para recursos no Azure

Para verificar se os dados extraídos estão corretos, iremos para cada tipo de recurso e para cada *cloud provider* comparar estes dados com aqueles que são apresentados pelas plataformas existentes nestes que disponibilizam esta informação, sendo no caso do Azure, o Azure Monitor.

Começando pelas APIs, iremos obter uma tabela como a da Figura 20.

	timestamp	resource_id	location	azone	4xx_error_total	5xx_error_total	count_total
1	2022-09-23 12:11:00	a50a68e5f5f1565b7293…	North Europe	Default Zone	<null>	<null>	2
2	2022-09-23 12:13:00	a50a68e5f5f1565b7293…	North Europe	Default Zone	<null>	<null>	1
3	2022-09-23 12:14:00	a50a68e5f5f1565b7293…	North Europe	Default Zone	3	<null>	5
4	2022-09-23 12:15:00	a50a68e5f5f1565b7293…	North Europe	Default Zone	<null>	<null>	2

Figura 20 Dados obtidos para APIs no Azure

É possível agora confirmar estes dados no Azure Monitor. Para estas métricas, o Azure Monitor reporta a seguinte informação, presente na Figura 21.

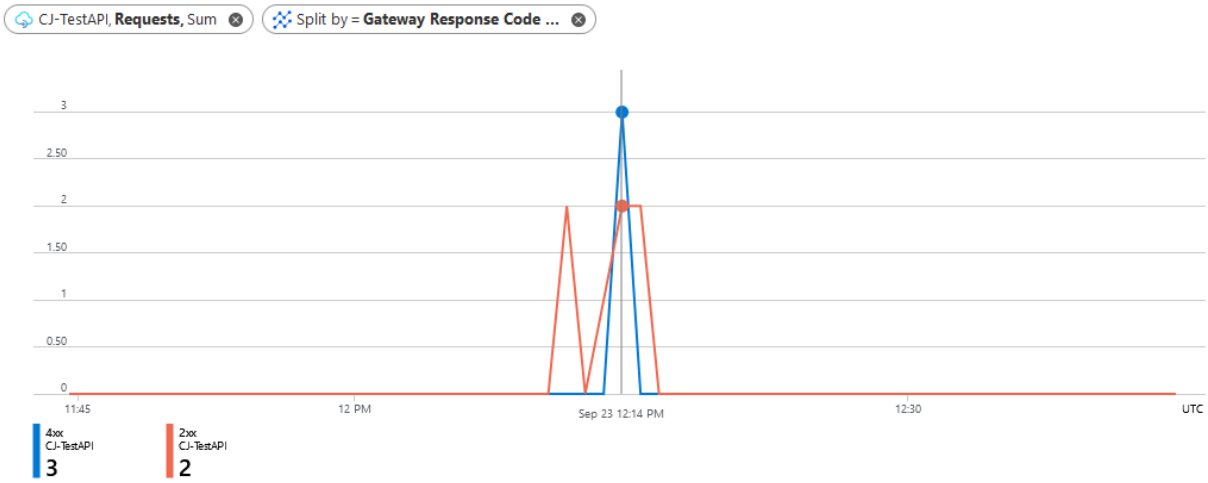


Figura 21 Dados métricos para recursos do tipo API Management no Azure Monitor

Nesta figura temos a métrica Request, que é equivalente à coluna com o nome count_total na Figura 20, no entanto, esta está dividida pelo tipo de resposta que é retornada. Para obtermos o valor total de cinco na linha das 12 horas e 14 minutos é necessário somar ambas as séries temporais presentes na Figura 21.

Passando para as máquinas virtuais, na Figura 22 está presente uma parte dos dados obtidos para estas.

	timestamp	resourc…	loc…	azone	cpu_util…	disk_read…	disk_write_…	network_in…	network_out…
1	2022-09-21 08:56:00	0312948a34b1b0…	West Europe	Default Zone	0.37	<null>	106543.72	40856	55983
2	2022-09-21 08:57:00	0312948a34b1b0…	West Europe	Default Zone	0.38	<null>	98364.27	40856	55983
3	2022-09-21 08:58:00	0312948a34b1b0…	West Europe	Default Zone	0.395	<null>	122908.85	41289	59790
4	2022-09-21 08:59:00	0312948a34b1b0…	West Europe	Default Zone	0.365	<null>	122985.81	40942	56037
5	2022-09-21 09:00:00	0312948a34b1b0…	West Europe	Default Zone	0.36	<null>	94250.47	40856	55983
6	2022-09-21 09:01:00	0312948a34b1b0…	West Europe	Default Zone	0.37	<null>	90159.94	40856	55983
7	2022-09-21 09:02:00	0312948a34b1b0…	West Europe	Default Zone	0.375	<null>	90137.1	42350	56061
8	2022-09-21 09:03:00	0312948a34b1b0…	West Europe	Default Zone	0.39	<null>	135165.77	41375	59846

Figura 22 Dados obtidos para máquinas virtuais no Azure

O Azure Monitor converte também algumas grandezas para facilitar a leitura, neste caso, alguns valores estão em kilobytes (1 kilobyte = 1024 bytes). É assim possível confirmar que os valores apresentados na Figura 23 para as 8 horas e 56 minutos correspondem aos apresentados na Figura 22.

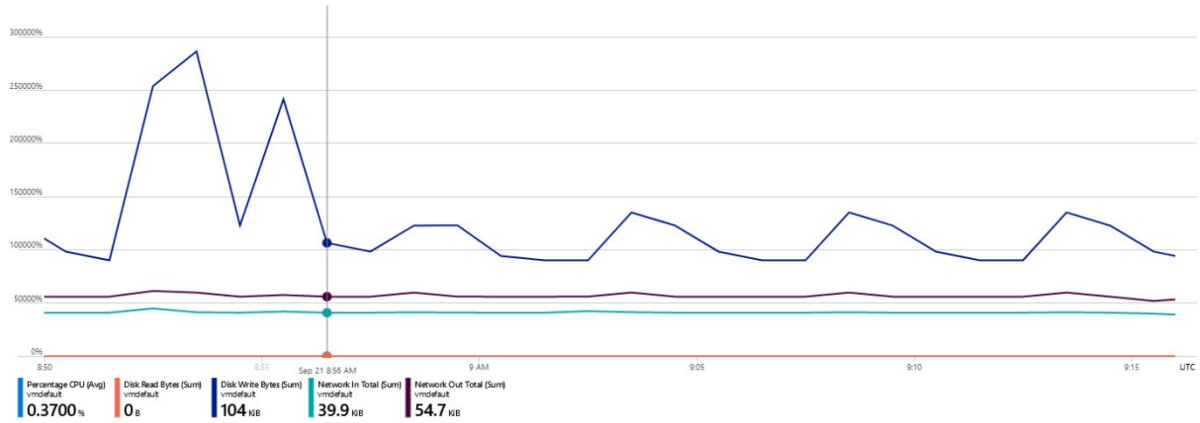


Figura 23 Dados métricos para máquinas virtuais no Azure Monitor

É possível fazer então esta análise para os restantes tipos de recursos para os quais iremos fazer a extração de dados métricos apresentados anteriormente. Para funções *serverless* iremos obter os dados da Figura 24.

	timestamp	resource_id	location	azone	function_execution_count...	duration...
1	2022-09-22 10:38:00	b2499ef2e6c5dff…	North Europe	Default Zone	12	26.958334
2	2022-09-22 10:39:00	b2499ef2e6c5dff…	North Europe	Default Zone	15	23.935484

Figura 24 Dados obtidos para funções *serverless* no Azure

Aqui só estão presentes dois registos visto que o Azure só disponibiliza estes dados para o tempo selecionado. Confirmando na Figura 26.

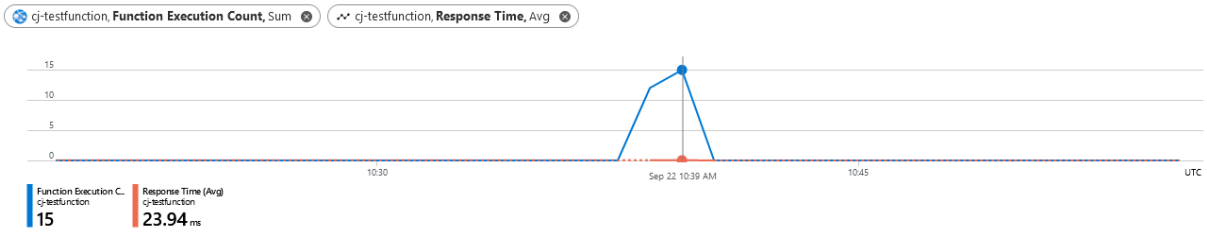


Figura 26 Dados métricos para funções *serverless* no Azure Monitor

Passando para *buckets* de armazenamento, na Figura 25.

	timestamp	r...	...		tota...	...	...	...	d...	h...	p...	l...	...	b...	4xx_...	5xx_...
1	2022-09-22 09:40…	09846bd7…	West Eu…	Def…	5372097500	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
2	2022-09-22 09:40…	de4a08c8…	West Eu…	Def…	11425568	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
3	2022-09-22 10:24…	de4a08c8…	West Eu…	Def…	<null>	1	<null>	<null>	<null>	<null>	<null>	<null>	5838	841	<null>	<null>
4	2022-09-22 10:27…	de4a08c8…	West Eu…	Def…	<null>	5	<null>	<null>	<null>	<null>	<null>	<null>	29116	4205	<null>	<null>
5	2022-09-22 10:32…	09846bd7…	West Eu…	Def…	<null>	1	<null>	<null>	<null>	<null>	<null>	<null>	5829	841	<null>	<null>
6	2022-09-22 10:35…	09846bd7…	West Eu…	Def…	<null>	5	<null>	<null>	<null>	<null>	<null>	<null>	29071	4203	<null>	<null>

Figura 25 Dados obtidos para recursos do tipo *buckets* de armazenamento no Azure

As colunas aqui presentes são as seguintes, para além das quatro colunas base (timestamp, resource_id, location, azone) que estão presentes em todos os recursos:

- total_size_used_total
- all_requests_total
- get_requests_total
- put_requests_total
- delete_requests_total
- head_requests_total
- post_requests_total
- list_requests_total
- bytes_uploaded_total
- bytes_downloaded_total
- 4xx_errors_total
- 5xx_errors_total

Figura 27 Dados métricos para *buckets* de armazenamento no Azure Monitor

Confirmando no Azure Monitor, para o recurso com o identificador a começar com "09846db7" iremos ter a Figura 27. Como o Azure Monitor não suporta métricas com granularidades diferentes simultaneamente, não foi possível colocar a métrica correspondente ao total de espaço ocupado (total_size_used_total) no grafo. Também, esta métrica, tal como métricas semelhantes, ao ser obtida, irá reportar no primeiro registo (correspondente ao *timestamp* mais antigo) o último valor conhecido, neste caso, correspondente às linhas um e dois da Figura 25. Também, no Azure Monitor, se for necessário aplicar um filtro, este é aplicado a todas as métricas presentes no grafo.

Já para aplicações web, iremos obter uma tabela como a da Figura 28 seguinte:

	timestamp	reso...	...	...	plan_id	.	network_in...	network_out...	http_2xx...	http_4xx...	latency...	request...
1	2022-09-22 11:35:00	a68b8b764ca…	West Eu…	Def…	1688671f8d0…	<null>	20955	5843	1	1	1911.1765	17
2	2022-09-22 11:36:00	a68b8b764ca…	West Eu…	Def…	1688671f8d0…	<null>	32232	5167	<null>	8	12.444445	27
3	2022-09-22 11:37:00	a68b8b764ca…	West Eu…	Def…	1688671f8d0…	<null>	5749	1750	<null>	5	7.8	5

Figura 28 Dados obtidos para recursos do tipo aplicações web no Azure

Aqui, para além das quatro colunas base, temos também uma coluna com o identificador do plano em que está a correr a aplicação web. Isto é necessário para obter a métrica de uso de CPU que não está preenchida intencionalmente nesta tabela. É possível então ver os resultados para a tabela az_webappplan apresentados na Figura 29 para obter métricas que venham de um plano como, neste caso, o uso de CPU. Esta separação é feita pois um plano web poderá ter mais que uma aplicação a correr.

É possível então confirmar ambos estes dados no Azure Monitor. Aqui é necessário utilizar dois gráficos pois o Azure Monitor apenas suporta um tipo de recurso por gráfico. Isto é apresentado na Figura 30. No gráfico de cima temos o uso de CPU para o plano web corres-

	timestamp	resource_id	location	azone	cpu_utilization_average
1	2022-09-22 11:36:00	1688671f8d091c5…	West Europe	Default Zone	22
2	2022-09-22 11:37:00	1688671f8d091c5…	West Europe	Default Zone	5
3	2022-09-22 11:38:00	1688671f8d091c5…	West Europe	Default Zone	4
4	2022-09-22 11:39:00	1688671f8d091c5…	West Europe	Default Zone	4
5	2022-09-22 11:40:00	1688671f8d091c5…	West Europe	Default Zone	5
6	2022-09-22 11:41:00	1688671f8d091c5…	West Europe	Default Zone	23
7	2022-09-22 11:42:00	1688671f8d091c5…	West Europe	Default Zone	8
8	2022-09-22 11:43:00	1688671f8d091c5…	West Europe	Default Zone	17
9	2022-09-22 11:44:00	1688671f8d091c5…	West Europe	Default Zone	15
10	2022-09-22 11:45:00	1688671f8d091c5…	West Europe	Default Zone	20
11	2022-09-22 11:46:00	1688671f8d091c5…	West Europe	Default Zone	33
12	2022-09-22 11:47:00	1688671f8d091c5…	West Europe	Default Zone	15
13	2022-09-22 11:48:00	1688671f8d091c5…	West Europe	Default Zone	16
14	2022-09-22 11:49:00	1688671f8d091c5…	West Europe	Default Zone	16
15	2022-09-22 11:50:00	1688671f8d091c5…	West Europe	Default Zone	16

Figura 29 Dados obtidos para planos de aplicações web no Azure

ponde à aplicação web presente no gráfico de baixo, com as suas métricas retiradas, pela mesma ordem que estão representadas na tabela da Figura 28.

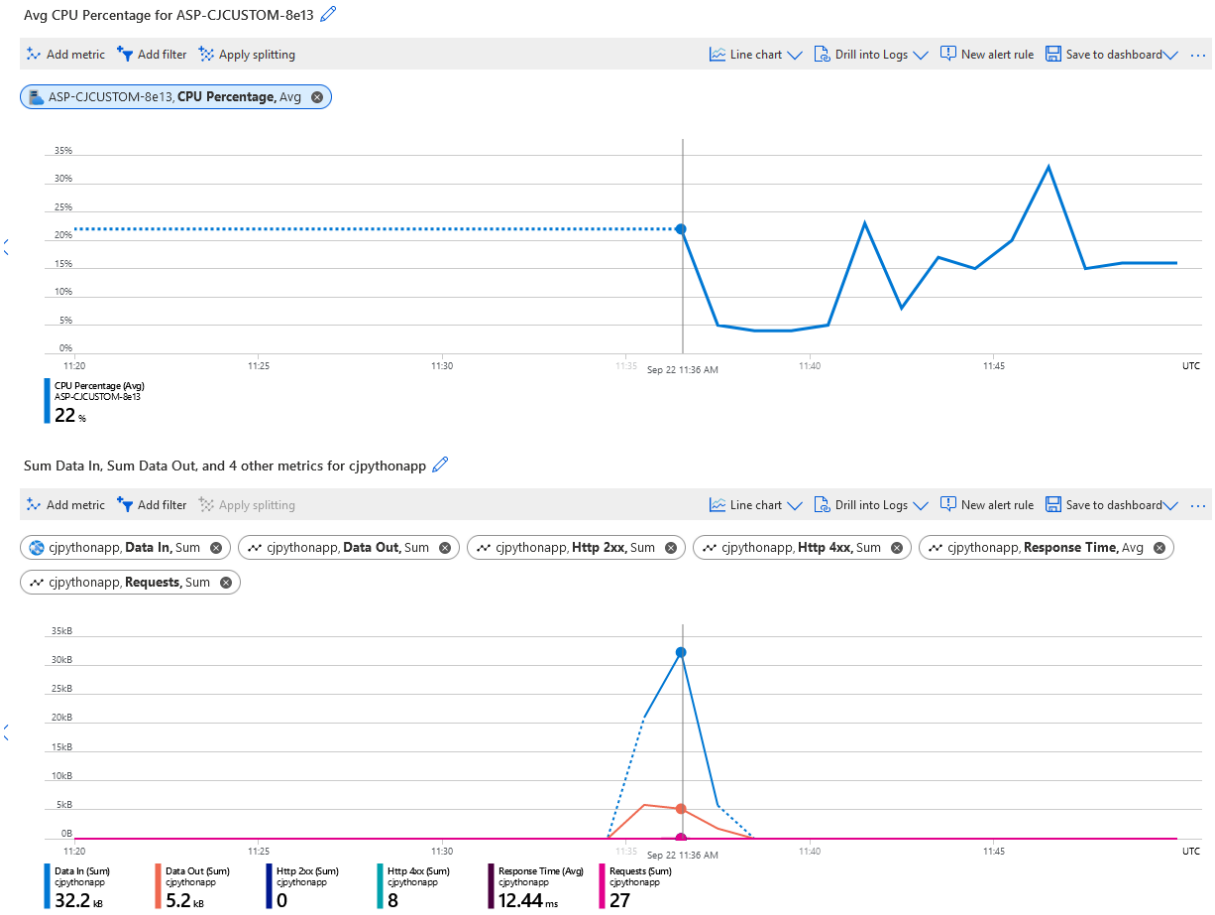


Figura 30 Dados métricos para planos de aplicações web e para aplicações web no Azure Monitor

Terminando com as bases de dados do CosmosDB, iremos obter uma tabela como a da Figura 31, que é completada com as métricas da Figura 32.

	timestamp	r...	...	azone	...	to...	...	...	...	docu...	...	...	...	...
1	2022-09-22 11:31…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	107374182000	2	<null>	<null>	<null>
2	2022-09-22 11:34…	0de18f62…	UK West	Default Zone	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
3	2022-09-22 11:36…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	107374182000	2	<null>	<null>	<null>
4	2022-09-22 11:41…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	107374182000	2	<null>	<null>	<null>
5	2022-09-22 11:44…	0de18f62…	UK West	Default Zone	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
6	2022-09-22 11:46…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	107374182000	2	<null>	<null>	<null>
7	2022-09-22 11:51…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	107374182000	2	<null>	<null>	<null>
8	2022-09-22 11:55…	0de18f62…	UK West	Default Zone	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
9	2022-09-22 11:56…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	107374182000	2	<null>	<null>	<null>
10	2022-09-22 12:01…	0de18f62…	UK West	Default Zone	3	4.1904764	400	<null>	<null>	107374182000	2	<null>	<null>	<null>
11	2022-09-22 12:05…	0de18f62…	UK West	Default Zone	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
12	2022-09-22 12:06…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	107374182000	2	<null>	<null>	<null>
13	2022-09-22 12:11…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	107374182000	2	<null>	<null>	<null>
14	2022-09-22 12:15…	0de18f62…	UK West	Default Zone	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
15	2022-09-22 12:16…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	107374182000	2	<null>	<null>	<null>
16	2022-09-22 12:20…	0de18f62…	UK West	Default Zone	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
17	2022-09-22 12:21…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	107374182000	3	<null>	<null>	<null>
18	2022-09-22 12:26…	0de18f62…	UK West	Default Zone	<null>	<null>	400	<null>	<null>	<null>	<null>	<null>	<null>	<null>
19	2022-09-22 12:27…	0de18f62…	UK West	Default Zone	5	6.2956266	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
20	2022-09-22 12:28…	0de18f62…	UK West	Default Zone	7	31.761906	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
21	2022-09-22 12:29…	0de18f62…	UK West	Default Zone	7	17.986738	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>

Figura 31 Dados obtidos para bases de dados do CosmosDB

	timestamp	server_side...	server_side...	server_side...	service...	service...	service...
1	2022-09-22 11:31…	<null>	<null>	<null>	100	100	100
2	2022-09-22 11:34…	0.02173913	0.02173913	0.02173913	<null>	<null>	<null>
3	2022-09-22 11:36…	0.02631579	0.02631579	0.02631579	<null>	<null>	<null>
4	2022-09-22 11:41…	<null>	<null>	<null>	<null>	<null>	<null>
5	2022-09-22 11:44…	6.571429	6.571429	6.571429	<null>	<null>	<null>
6	2022-09-22 11:46…	<null>	<null>	<null>	<null>	<null>	<null>
7	2022-09-22 11:51…	<null>	<null>	<null>	<null>	<null>	<null>
8	2022-09-22 11:55…	0.015625	0.015625	0.015625	<null>	<null>	<null>
9	2022-09-22 11:56…	<null>	<null>	<null>	<null>	<null>	<null>
10	2022-09-22 12:01…	0.016129032	0.016129032	0.016129032	<null>	<null>	<null>
11	2022-09-22 12:05…	0.015625	0.015625	0.015625	<null>	<null>	<null>
12	2022-09-22 12:06…	<null>	<null>	<null>	<null>	<null>	<null>
13	2022-09-22 12:11…	<null>	<null>	<null>	<null>	<null>	<null>
14	2022-09-22 12:15…	0.015625	0.015625	0.015625	<null>	<null>	<null>
15	2022-09-22 12:16…	0.02631579	0.02631579	0.02631579	<null>	<null>	<null>
16	2022-09-22 12:20…	0.015625	0.015625	0.015625	<null>	<null>	<null>
17	2022-09-22 12:21…	<null>	<null>	<null>	<null>	<null>	<null>
18	2022-09-22 12:26…	<null>	<null>	<null>	<null>	<null>	<null>
19	2022-09-22 12:27…	2.4827585	2.4827585	2.4827585	<null>	<null>	<null>
20	2022-09-22 12:28…	1.1833333	1.1833333	1.1833333	<null>	<null>	<null>
21	2022-09-22 12:29…	1.3859649	1.3859649	1.3859649	<null>	<null>	<null>

Figura 32 Dados de latência do servidor e taxa de disponibilidade para bases de dados do CosmosDB

As colunas apresentadas na Figura 31 são as seguintes:

- total_requests_count
- total_request_units_total
- provisioned_throughput_maximum
- data_usage_total
- index_usage_total
- document_quota_total
- document_count_total
- replication_latency_average
- replication_latency_maximum
- replication_latency_minimum

Já as que estão presentes na Figura 32 são as seguintes:

- server_side_latency_average
- server_side_latency_maximum
- server_side_latency_minimum
- service_availability_average
- service_availability_maximum
- service_availability_minimum

Muitos destes dados têm granularidades diferentes, portanto não é possível arranjar um gráfico que seja compatível com todos. Utilizando, por exemplo a linha 20, que corresponde às 12 horas e 28 minutos será possível obter um gráfico como o da Figura 33 para as métricas para quais existem dados.

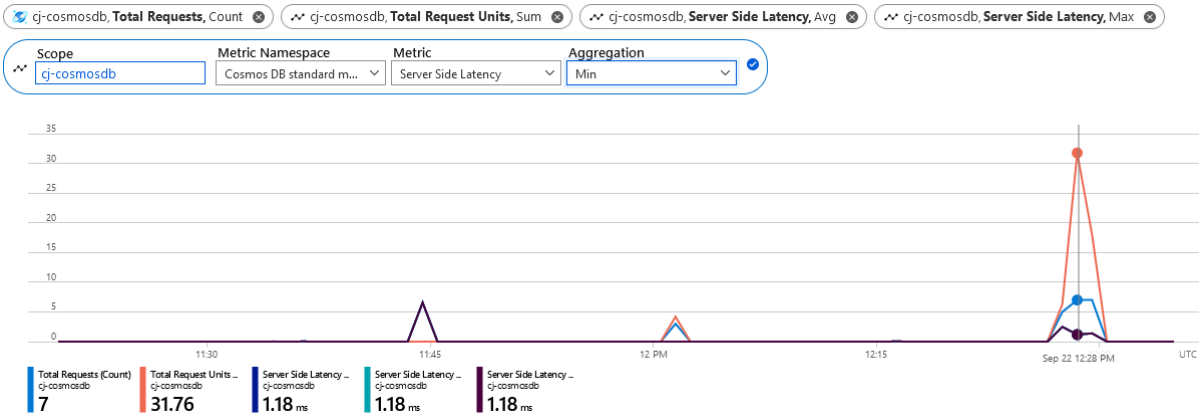


Figura 33 Dados métricos para bases de dados do CosmosDB no Azure Monitor

Novamente aqui, teremos os resultados corretos entre os extraídos e os disponibilizados. De notar, temos a métrica da latência do servidor reporta a mesma informação nas três agregações disponibilizadas. Isto deve-se ao facto de dados para esta métrica só estarem a ser recolhidos pelo sistema uma vez por minuto e como estamos a ver e extrair as métricas ao minuto, apenas existe um valor para os dados formar os dados estatísticos.

5.2 Validação de resultados para Google Cloud

No Google Cloud iremos ter três *projects*, o primeiro deles com o nome "TestProject". Este *project* irá conter a maioria dos recursos a testar. Já o segundo e o terceiro são semelhantes em que vão conter apenas uma aplicação web do App Engine, uma base de dados do Firestore em modo nativo e dois *buckets* de armazenamento do Google Storage usados pela aplicação web. Estes recursos foram principalmente utilizados durante o desenvolvimento da aplicação. Já o *project* "TestProject" irá conter principalmente os recursos apresentados na Tabela 10. Este também vai ser o *project* que será principalmente testado, no entanto, os dados métricos para recursos (ou *assets* no Google Cloud) são recolhidos apenas para os tipos de recursos suportados, para todos os *projects*.

Para além destes recursos, existem também recursos extra criados pelo Google Cloud para suportar estes serviços, como por exemplo, *buckets* de armazenamento para guardar informação sobre aplicações web.

Nome do Recurso	Tipo	Região
cj-dr-bucket	Storage	eur4
function-1	Functions	europe-west6
testgateway	API Gateway	us-central-1
instance-a	Compute	us-central-1
(default)	Firestore (Modo Datastore)	europe-west3
cj-exemple-bucket	Storage	europe-west3
apps/testproject-351211	App Engine	europe-west3

Tabela 10 Recursos dentro do project "TestProject"

A listagem das métricas a retirar por cada tipo de recurso está disponível no apêndice A.1. Começando por gerar o grafo, vamos obter algo como na Figura 34. Este grafo foi obtido através da consulta seguinte. Esta consulta permite obter todos os recursos, os seus *resource groups* e localizações associadas. Para simplificar, este não apresenta todas as localizações existentes no Google Cloud, apenas apresentando aquelas que têm recursos associados. Como atualmente só temos recursos do GCP no grafo, não é necessário especificar na consulta o nome do *cloud provider* para o qual queremos ir buscar os recursos e restante informação.

```
MATCH (g:Location)-[*]->(re:Region)-[*]->(az:AvailabilityZone)-
[*]->(r:Resource)<-[*]-(rg:ResourceGroup)<-[*]-(c:Cloud)
RETURN r,rg,c,az,re, g
```

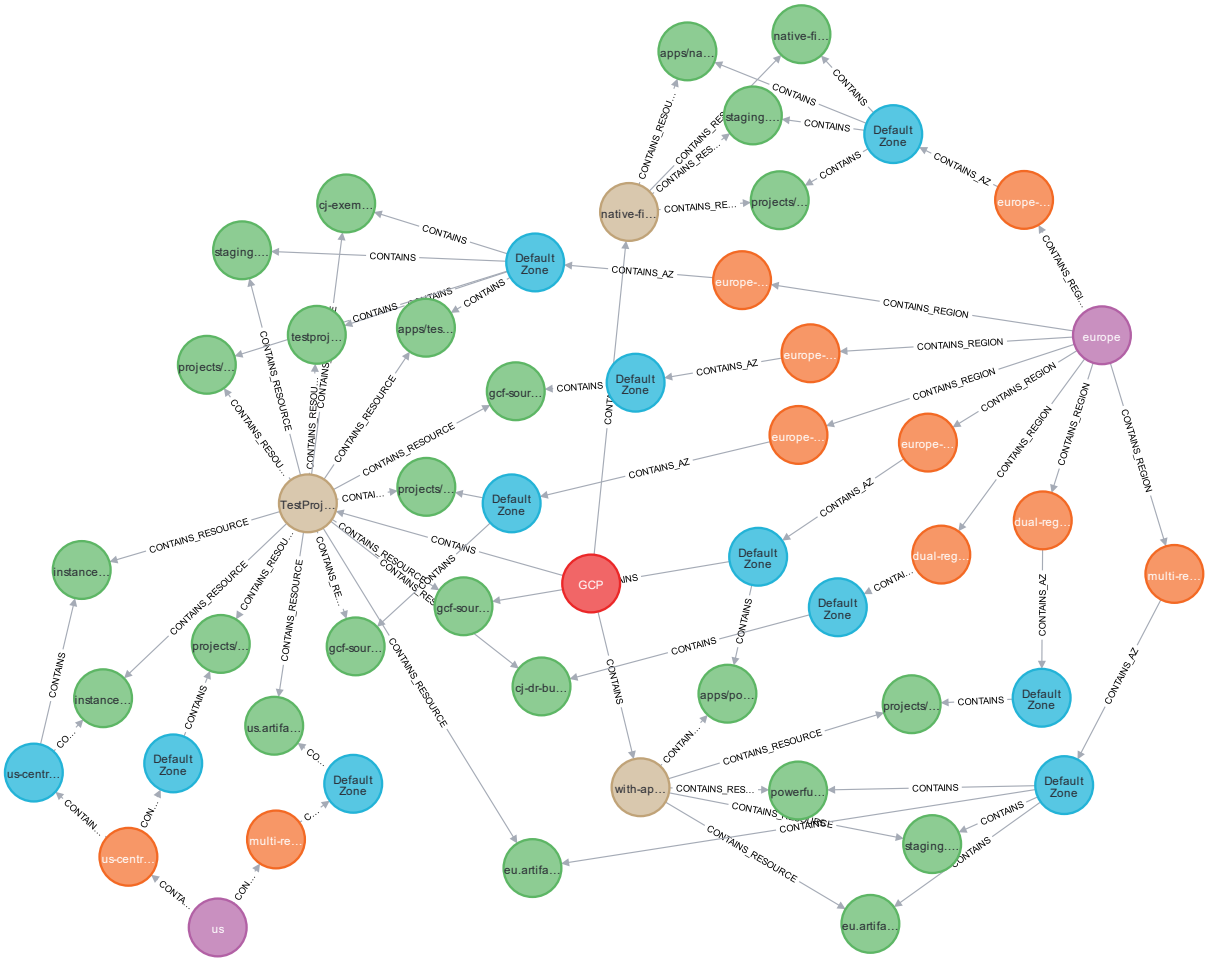


Figura 34 Grafo obtido para os recursos presentes no Google Cloud

Este grafo irá contém os seguintes nós, que podem ser divididos tal como enunciado na secção 3.1.1:

- 1 do tipo Cloud, a vermelho, que neste caso apenas será o nó correspondente ao Google Cloud
- 2 do tipo Região Geográfica, a lilás
- 10 do tipo Região, a laranja
- 11 do tipo Zona de Disponibilidade, a azul
- 3 do tipo ResourceGroup, a bege
- 24 do tipo Resource, a verde

De forma semelhante ao Azure, depois de executar a extração de métricas iremos obter as seguintes tabelas com informação, e, entre parenteses, os tipos de recursos que são extraídas métricas para cada tabela:

- common_apigateway (API Gateways)

- gcp_firestore (bases de dados do Firestore em modo nativo)
- common_vm (instâncias de máquinas virtuais)
- common_function (funções *serverless*)
- common_storage (*buckets* de armazenamento)
- common_webapp (aplicações web do App Engine)

Começando por recursos do tipo API Gateway, iremos obter uma tabela como a da Figura 35.

	timestamp	resource_id	location	azone	4xx_error_total	5xx_error_total	count_total
1	2022-09-16 09:37:00	0d8c9bf6863dced…	us-central1	Default Zone	3	<null>	8
2	2022-09-16 09:44:00	0d8c9bf6863dced…	us-central1	Default Zone	3	<null>	8

Figura 35 Tabela obtida para recursos do tipo API Gateway do Google Cloud

Aqui vamos ter apenas duas entradas pois as linhas sem valores numéricos ou apenas com valores nulos não são inseridas na base de dados. Podemos então entrar no Metrics Explorer do Google Cloud para confirmar estes valores. Nesta plataforma do Google Cloud é possível ver dados métricos para qualquer tipo de recursos existente num *project*. Podemos então ver na Figura 36 ou dados reportados e confirmar com os da Figura 35 para ver se estes estão corretos. Aqui não temos um valor para o total de *requests* (coluna count_total) no entanto podemos somar todas as séries temporais apresentadas para obter este valor, neste caso os cinco da série 3xx mais os três da série 4xx obtendo assim os oito de *requests* totais. Também de notar existe uma hora de diferença entre os valores disponibilizados no Metrics Explorer e os valores capturados. Todos os valores capturados estão em UTC e o Metrics Explorer está a corrigir estes valores para a hora local, neste caso, UTC+1.

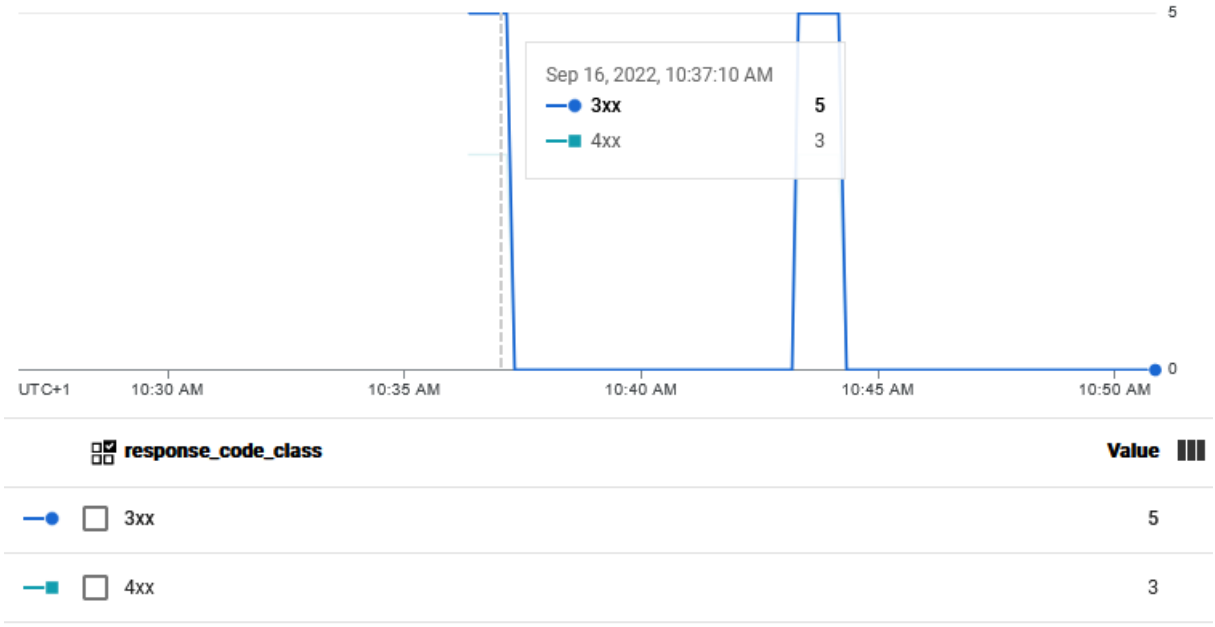


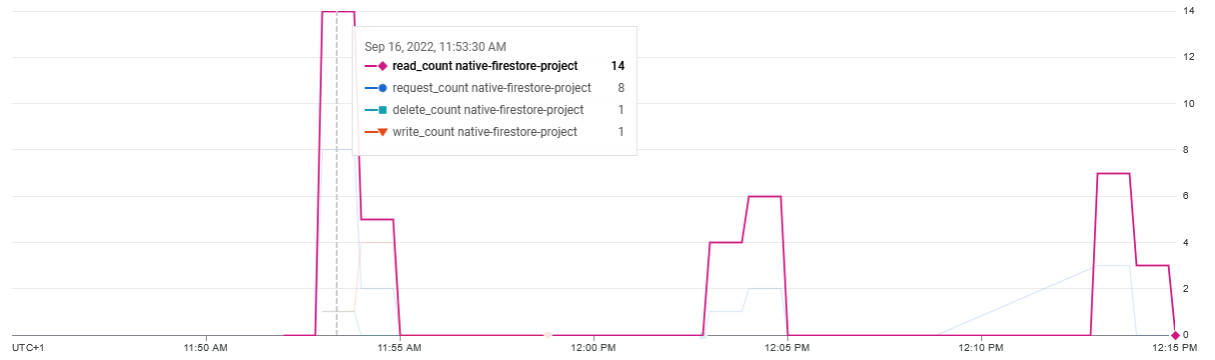
Figura 36 Métricas a retirar do AWS para ApiGateways no Metrics Explorer

O mesmo é feito para os outros tipos de recursos analisados. Para bases de dados do tipo Firestore, obtém-se uma tabela como a da Figura 37. Aqui apenas estamos a considerar aquelas que estão em modo nativo e não as que estão em modo Datastore.

	timestamp	res...	loca...	azone	request_count...	delete_count...	read_count_...	write_count...
1	2022-09-16 11:03:00	5024983fd6…	europe-west2	Default Zone	1	<null>	4	<null>
2	2022-09-16 11:04:00	5024983fd6…	europe-west2	Default Zone	2	<null>	6	<null>
3	2022-09-16 10:53:00	5024983fd6…	europe-west2	Default Zone	8	1	14	1
4	2022-09-16 10:54:00	5024983fd6…	europe-west2	Default Zone	2	<null>	5	4

Figura 37 Tabela obtida para bases de dados do Firestore do Google Cloud

Já no Metrics Explorer é possível confirmar estes dados, como representado na Figura 38.



Metric	project_id	Value
delete_count	native-firestore-project	1
read_count	native-firestore-project	14
request_count	native-firestore-project	8
write_count	native-firestore-project	1

Figura 38 Métricas para bases de dados do Firestore no Metrics Explorer

Para as máquinas virtuais a tabela obtida será a da Figura 39.

	timestamp	re...	loc...	azone	cpu_...	disk_read...	disk_write...	network_in...	network_out...
1	2022-09-16 11:49:00	c33e52c8b…	us-central1	us-central1-c	0.5996564	<null>	<null>	5667	2098
2	2022-09-16 11:50:00	c33e52c8b…	us-central1	us-central1-c	0.5713019	<null>	4130	1506	1336
3	2022-09-16 11:51:00	c33e52c8b…	us-central1	us-central1-c	0.59518886	<null>	22578	1820	1738
4	2022-09-16 11:52:00	c33e52c8b…	us-central1	us-central1-c	0.6667309	<null>	83290	3866	2796
5	2022-09-16 11:53:00	c33e52c8b…	us-central1	us-central1-c	0.61206657	<null>	156242	5719	2125
6	2022-09-16 11:54:00	c33e52c8b…	us-central1	us-central1-c	0.5514607	<null>	<null>	1564	1358
7	2022-09-16 11:55:00	c33e52c8b…	us-central1	us-central1-c	0.5845302	<null>	11356	1971	1968
8	2022-09-16 11:56:00	c33e52c8b…	us-central1	us-central1-c	0.7338466	<null>	155523	4124	3293
9	2022-09-16 11:57:00	c33e52c8b…	us-central1	us-central1-c	0.78527987	<null>	374926	6451	2675
10	2022-09-16 11:58:00	c33e52c8b…	us-central1	us-central1-c	0.62944317	<null>	97641	2780	2880
11	2022-09-16 11:59:00	c33e52c8b…	us-central1	us-central1-c	0.5570736	<null>	171562	1667	1300
12	2022-09-16 12:00:00	c33e52c8b…	us-central1	us-central1-c	0.74235046	202344	103237	3099	1599
13	2022-09-16 12:01:00	c33e52c8b…	us-central1	us-central1-c	0.78523135	600472	362111	6565	2952

Figura 39 Tabela com dados obtidos para máquinas virtuais no Google Cloud

Podemos validar estes dados no Metrics Explorer, como representado na Figura 40. De notar que os dados apresentados pelo Metrics Explorer vêm em kilobytes e os recolhidos estão em bytes (1 kilobyte = 1024 bytes).

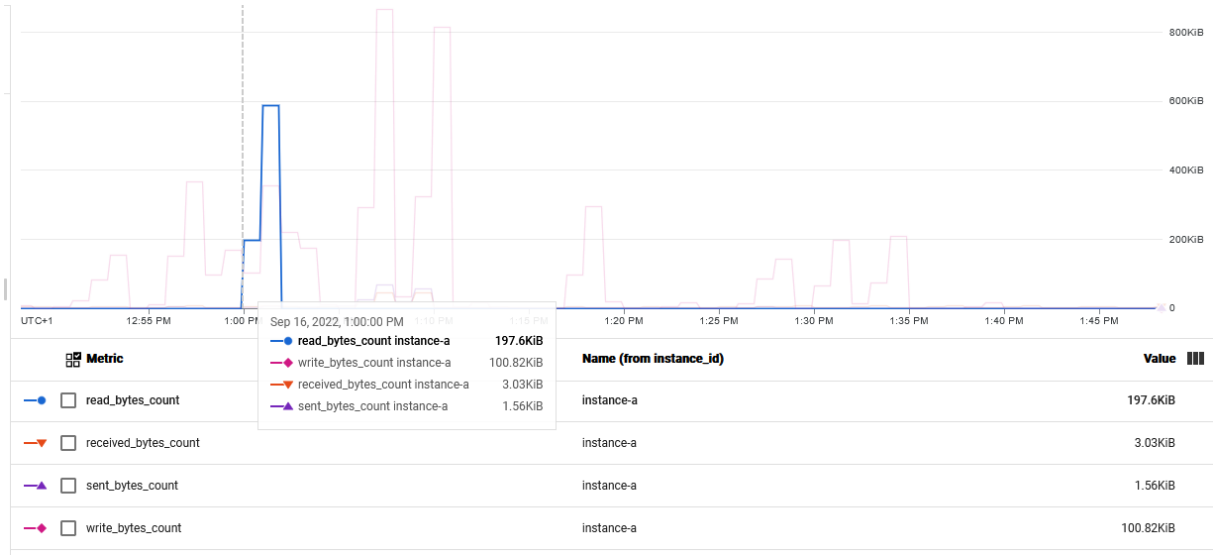


Figura 40 Métricas de máquinas virtuais no Metrics Explorer

Como o Metrics Explorer não suporta corretamente graficamente métricas com unidades de medida diferentes, não foi possível incluir a métrica de utilização de CPU no gráfico da Figura 40 no entanto este também apresenta valores corretos.

Passando às funções *serverless*, a tabela obtida será algo como a da Figura 41 seguinte.

	timestamp	resourc...	loca...	azone	function_execution...	duration_a...
1	2022-09-16 12:51:00	e329210ebde1b…	europe-west6	Default Zone	24	7.2732925
2	2022-09-16 12:52:00	e329210ebde1b…	europe-west6	Default Zone	39	11.841077

Figura 41 Tabela obtida para funções serverless do Google Cloud

Novamente, esta informação coincide com a disponibilizada pelo Metrics Explorer. Tal como aconteceu para as máquinas virtuais, o Metrics Explorer não suporta corretamente métricas com unidades de medida diferentes no mesmo grafo. Como tal, em vez de o valor da duração de execução (execution_times) ser representado num valor mais legível (milissegundos), é representado no seu valor original, em nanossegundos. Na base de dados, as unidades entre métricas dos três *cloud providers* são logo unificadas, portanto este valor é logo convertido para milissegundos. Um exemplo disto pode ser visto na Figura 42.

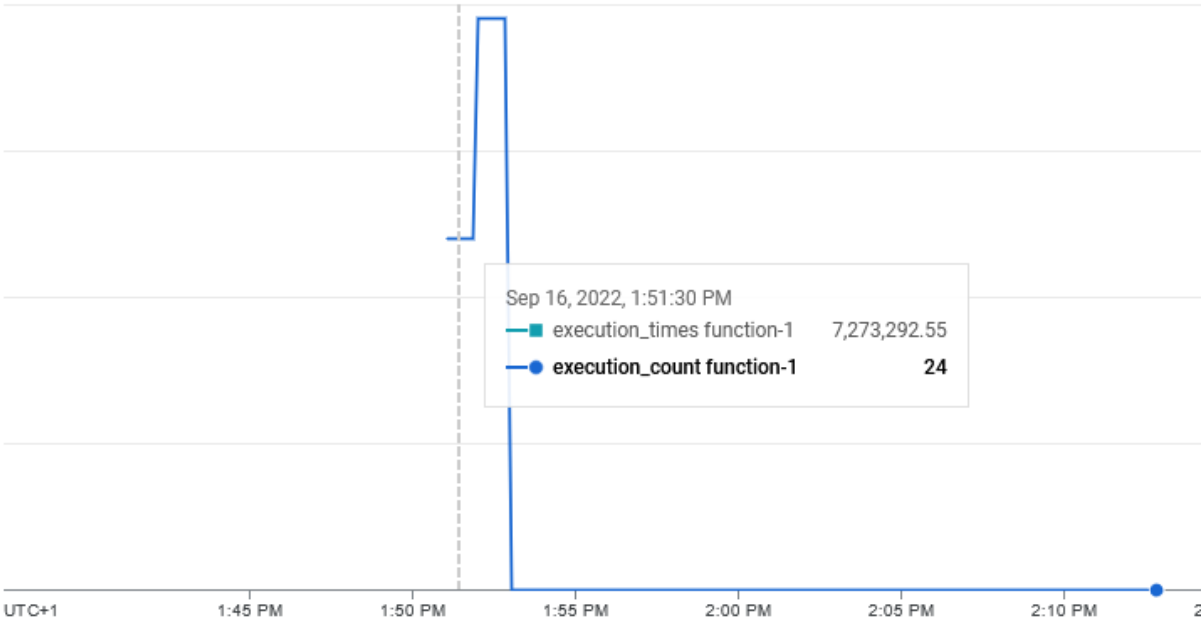


Figura 42 Métricas para funções *serverless* no Metrics Explorer

Já para o armazenamento do Cloud Storage, como temos vários *buckets* de armazenamento criados pelo Cloud Storage aqui teremos mais recursos visto o Google Cloud cria automaticamente *buckets* para suportar outros tipos de recurso, como aplicações web, no entanto, iremos analisar a informação do recurso com o identificador a começar por "aa9875ded" na linha 31 da Figura 43, que corresponde ao recurso com o nome " cj-exemple-bucket".

	timestamp	res...	loca...		t...	...	...	...	...	...	...	...	...	...	...	...
29	2022-09-16 12:50…	aa9875ded…	europe-west3	Def…	24000194	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
30	2022-09-16 12:50…	ae0886f1c…	Multi-Region	Def…	558224700	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
31	2022-09-16 12:51…	aa9875ded…	europe-west3	Def…	<null>	6	<null>	<null>	1	2	1	1	8000023	4987	1	<null>
32	2022-09-16 12:52…	aa9875ded…	europe-west3	Def…	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	66	<null>	<null>
33	2022-09-16 12:55…	0582031b9…	europe-cent…	Def…	641	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
34	2022-09-16 12:55…	23f2fdd8b…	Multi-Region	Def…	428729088	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>

Figura 43 Tabela com dados obtidos para *buckets* no Google Cloud

Para além das quatro primeiras colunas que são iguais em todas as tabelas (timestamp, resource_id, location e azone), as seguintes são:

- total_size_used_total
- all_requests_total
- get_requests_total
- put_requests_total
- delete_requests_total
- head_requests_total

- post_requests_total
- list_requests_total
- bytes_uploaded_total
- bytes_downloaded_total
- 4xx_errors_total
- 5xx_errors_total

Infelizmente, como o Metrics Explorer não lida corretamente com métricas com unidades diferentes, é necessário dividir esta análise em duas partes. Primeiro, para métricas que representam uma contagem na Figura 44 apenas estão representadas as métricas para o total de *requests*, pedidos do tipo HEAD e para pedidos que resultaram num erro 404, ou de ficheiro não encontrado. Depois para métricas que representam dimensões de ficheiros na Figura 45, estão representadas as métricas do tamanho total ocupado e do número de bytes recebidos.

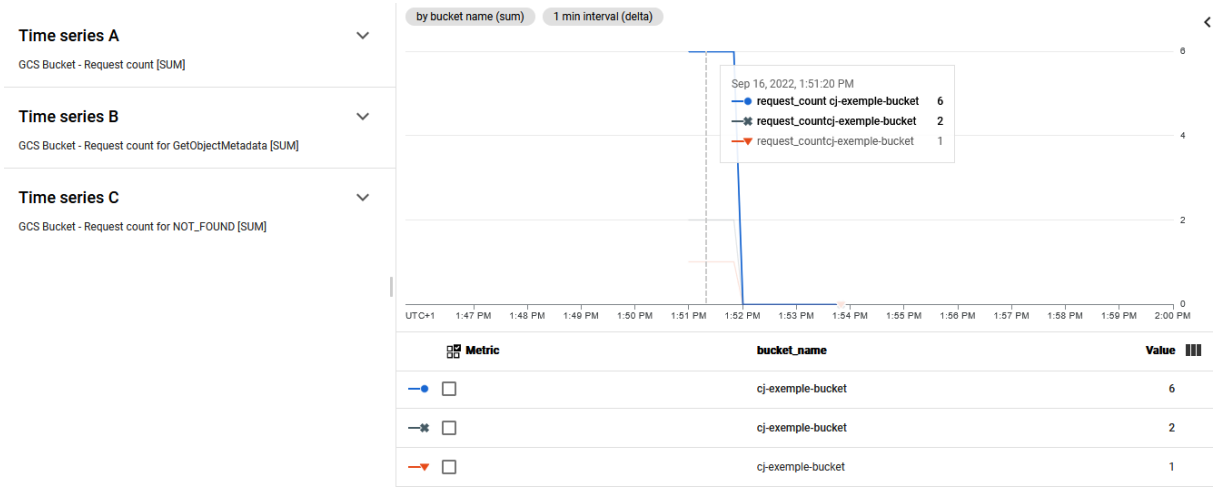


Figura 44 Métricas de armazenamento no Metrics Explorer para *requests*

O Metrics Explorer também não tem nenhuma opção que permita facilmente renomear as séries de forma a ser mais legível a leitura dos dados. Na Figura 44 temos a azul o total de *requests*, a cinzento o número de *requests* do tipo HEAD, que devolvem o apenas informação sobre o ficheiro em vez de o ficheiro em si e a laranja, pedidos que resultaram em erro 404, ou não encontrado.

Já relativamente à Figura 45, temos aqui o total armazenado no *bucket* cj-exemple-bucket, a azul, o qual corresponde ao recurso com o identificador a começar por " aa9875ded" tal como referido anteriormente. O motivo de na linha 31 da Figura 43 esta métrica não estar presente é, pois, esta métrica só é reportada a cada cinco minutos. No entanto, é possível ver o valor esta métrica na linha 29 da mesma figura. Representado também na Figura 45 está o número de bytes recebidos, ou o número de bytes carregados para o *bucket,* a laranja.

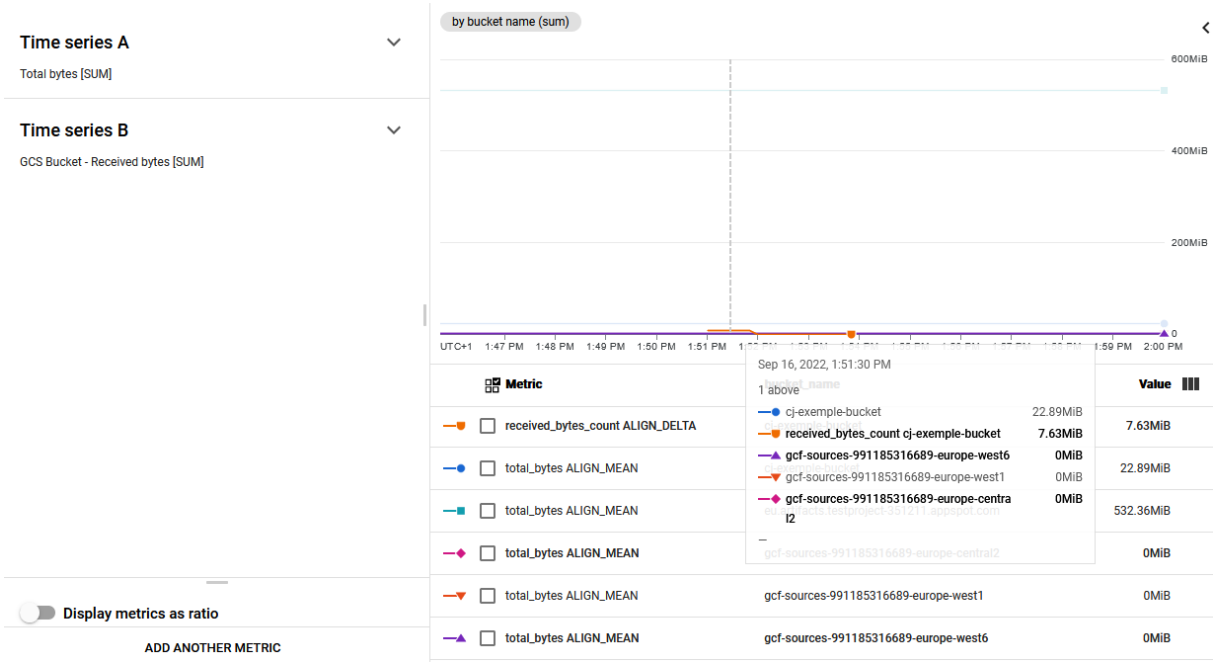


Figura 45 Métricas de armazenamento no Metrics Explorer

Terminando com as aplicações web, a tabela obtida será algo como a da Figura 46. Aqui, para além das quatro colunas base, existe também uma coluna com o *plan_id* utilizado apenas pelo Azure.

	timestamp	r...	loca...		...	cpu...	network_i...	network_o...	http_2...	http_4...	late...	req...
1	2022-09-16 12:51…	c0fc7f0c…	europe-west3	Def…	<null>	<null>	1188	497	3	<null>	4.1968217	3
2	2022-09-16 12:52…	c0fc7f0c…	europe-west3	Def…	<null>	<null>	2349	983	7	<null>	213.94254	7

Figura 46 Tabela obtida para aplicações web do App Engine

Já no Metrics Explorer, iremos obter algo como na Figura 47. Nesta figura foram dados como exemplos os dados de entrada e saída de uma aplicação web em bytes.

Figura 47 Bytes de entrada e saida numa aplicação web no Metrics Explorer

5.3 Validação de resultados para AWS

Como no AWS os recursos estão todos divididos por região, os *resource groups* apenas vão ter recursos na mesma região. Como tal, para estes testes iremos ter dois *resource groups*. O primeiro é o "TestResourceGroup" e está colocado na região eu-west-2. O segundo é o "AnotherResourceGroup" e está colocado na região eu-west-1. Este último é constituído pelos recursos com a tag "Alt", que neste caso, é apenas um recurso: um *bucket* no S3 com o nome "cj-altbucket". Já o *resource group* "TestResourceGroup" é constituído pelos recursos com a tag "Tests". Este *resource group* contem os recursos descritos na Tabela 11.

Identificador (Nome do recurso)	Serviço	Tipo
restapis/dh3kpmlp0h (PetStore)	ApiGateway	RestApi
testtable	DynamoDB	Table
i-0b0a62ac150e96853 (AWS-TestVM)	EC2	Instance
TestApplication	ElasticBeanstalk	Application
testfunction	Lambda	Function
cj-s3test	S3	Bucket

Tabela 11 Recursos dentro do *resource group* "TestResourceGroup"

Para além destes recursos é possível também adicionar a tag ao *environment* das aplicações web do ElasticBeanstalk. Isto faz com que também sejam automaticamente adicionados ao *resource group* todos os recursos necessários para o *environment* que tenham sido criados pelo ElasticBeanstalk. Na Tabela 12, estão descritos os recursos extra que são criados e associados ao *resource group* pelo ElasticBeanstalk. Estes recursos vão ser diferentes dependendo se o ambiente está no modo Single Instance ou Load Balanced. Neste teste, a aplicação web TestApplication tem o seu ambiente a correr em modo Load Balanced.

Identificador (Nome do recurso)	Serviço	Tipo
TestApplication/Testapplication-env	ElasticBeanstalk	Environment
sg-013cb559e1f0619d9	EC2	SecurityGroup
awseb-AWSEB-17HXSG7UIUNQG/d25358caef5f8c39	ElasticLoadBalancerV2	TargetGroup
awseb-e-svm6naaqv2-stack/30262980-0c14-11ed-a675-023937822d20	CloudFormation	Stack
app/awseb-AWSEB-3QAIHRCDOSAX/ff5da2bb62457248	ElasticLoadBalancerV2	LoadBalancer
i-0a2b53f8e66cc9013	EC2	Instance
sg-00e8927fee903df99	EC2	SecurityGroup

Tabela 12 Recursos do ambiente da aplicação web que são adicionados ao *resource group*

Para além da instância EC2 criada pelo *environment* (i-0a2b53f8e66cc9013), não serão extraídas métricas para qualquer outro tipo de recurso presente na Tabela 12. A listagem das métricas a serem retiradas por tipo de recursos está presente no apêndice A.1.

Estes treze recursos são aqueles que devem aparecer no grafo, com ligações às zonas de disponibilidade da região eu-west-2. Algo semelhante deverá acontecer para o *resource group* "AnotherResourceGroup".

Executando a geração do grafo teremos o resultado apresentado na Figura 48. Este grafo foi obtido através da seguinte consulta:

```
MATCH (g:Location)-[*]->(re:Region)-[*]->(az:AvailabilityZone)-
[*]->(r:Resource)<-[*]-(rg:ResourceGroup)<-[*]-(c:Cloud)
RETURN r,rg,c,az,re, g
```

Esta consulta é a mesma apresentada na secção 5.2 e permite obter todos os recursos, os seus *resource groups* e localizações associadas. Para simplificar, este não apresenta todas as localizações existentes no AWS, apenas apresentando aquelas que têm recursos associados. Esta consulta também só mostra apenas recursos presentes no AWS pois neste momento a extração de recursos foi apenas executada para o AWS, caso contrário, seria necessário especificar na consulta que estamos apenas à procura de *resource groups* que pertençam à cloud "AWS".

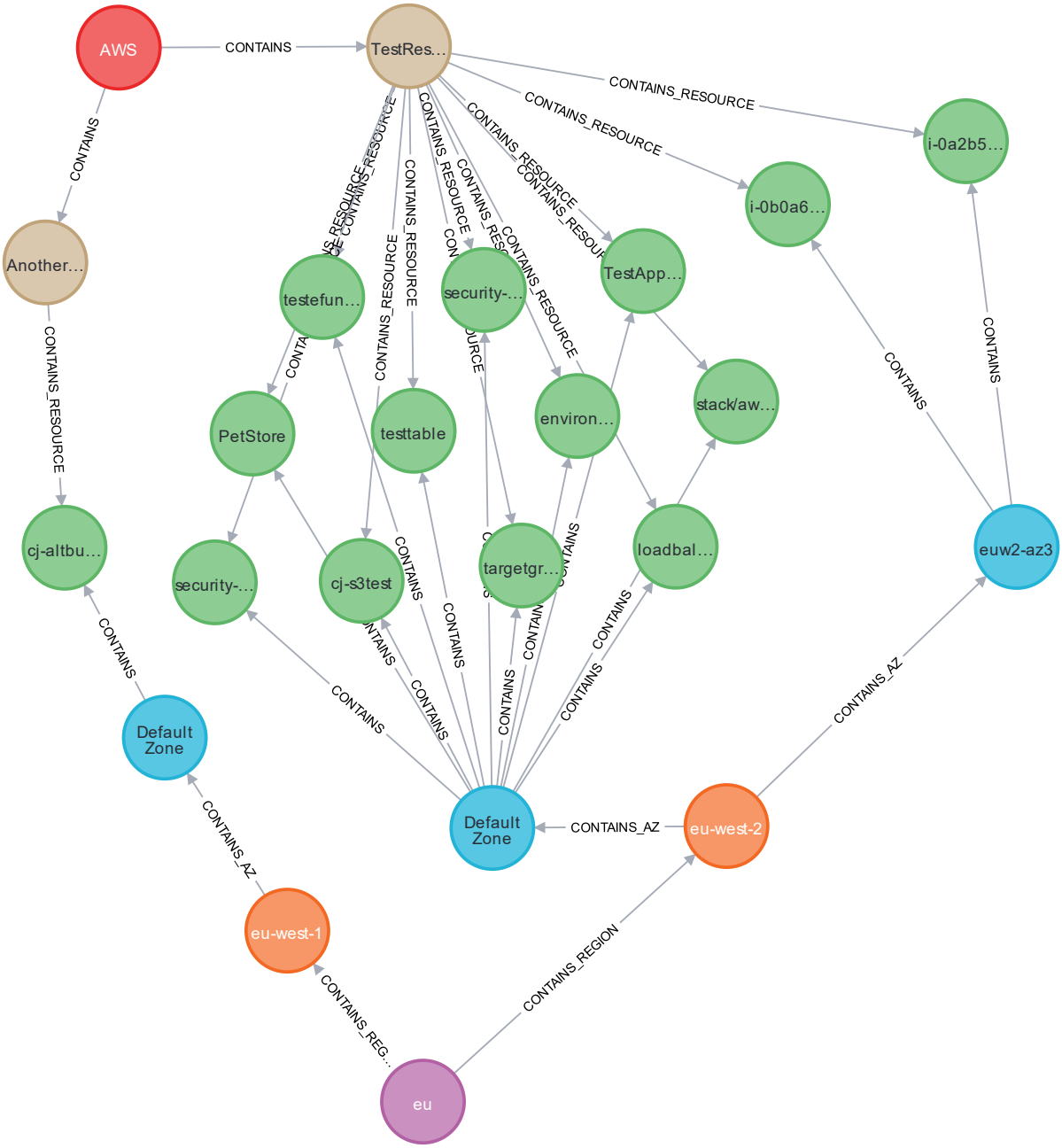


Figura 48 Grafo final com os recursos no AWS

Na Figura 48 temos o grafo gerado através da consulta anterior para o AWS. Temos um total de 23 nós. Podemos dividir estes nós pelos seus seis tipos distintos, tal como especificado na secção 3.1.1:

- 1 do tipo Cloud, a vermelho, que neste caso apenas será o nó correspondente ao AWS
- 1 do tipo Região Geográfica, a lilás
- 2 do tipo Região, a laranja

- 3 do tipo Zona de Disponibilidade, a azul
- 2 do tipo ResourceGroup, a bege
- 14 do tipo Resource, a verde

É também possível verificar que temos os 14 nós relativos aos recursos, distribuídos corretamente pelos seus *resource groups* e associados de forma correta às suas regiões e zonas de disponibilidade.

Tendo o grafo, são então recolhidas as métricas para os recursos dos seguintes tipos:

- ApiGateway - RestApi
- DynamoDB - Table
- EC2 - Instance
- Lambda - Function
- S3 - Bucket
- ElasticBeanstalk - Application

Todas as métricas destes tipos de recursos à exceção das tabelas pertencentes ao DynamoDB serão unificadas com os outros *cloud providers*, tal como explicado nos capítulos 3.1.3 e 3.1.4. As tabelas seguintes devem conter os dados métricos, para os tipos de recursos pela mesma ordem da listagem em cima:

- common_apigateway
- aws_dynamodb
- common_vm
- common_function
- common_storage
- common_webapp

Começando pelo ApiGateway, ficaremos com uma tabela como a da Figura 49:

	timestamp	resource_id	location	azone	4xx_error_total	5xx_error_total	count_total
1	2022-09-14 12:20:00	7b8c50c3079235…	eu-west-2	Default Zone	<null>	<null>	11
2	2022-09-14 12:21:00	7b8c50c3079235…	eu-west-2	Default Zone	10	<null>	27
3	2022-09-14 12:23:00	7b8c50c3079235…	eu-west-2	Default Zone	<null>	<null>	1

Figura 49 Tabela obtida para recursos do tipo ApiGateway

Neste caso temos apenas três registos, visto que os registos apenas com valores nulos, ou sem valores numéricos não estão a ser inseridos na base de dados. Podemos entrar no Amazon CloudWatch para confirmar estes valores, tal como é mostrado na Figura 50.

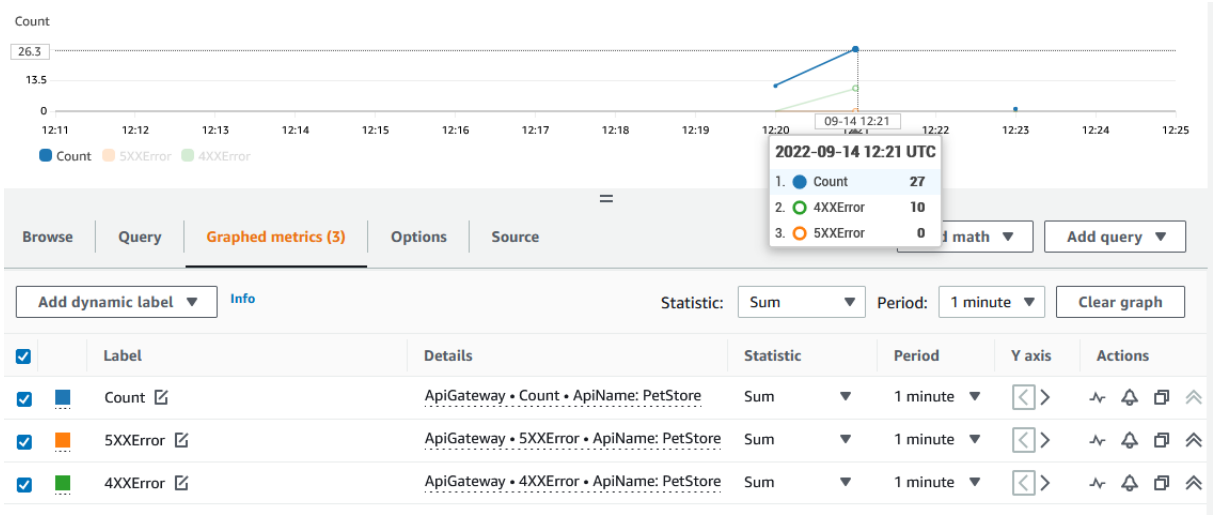


Figura 50 Métricas a retirar do AWS para ApiGateways no CloudWatch

Como é possível verificar, as métricas para as 12 horas e 21 minutos coincidem nos dois casos. Também pode ser visto que não é inserido na base de dados os dados para as 12 horas e 22 minutos visto que o próprio CloudWatch não reporta estes valores.

O mesmo é feito para outros recursos. Para os recursos do tipo DynamoDB iremos ficar com uma tabela como a da Figura 51.

	timestamp	resourc...	l...	azone	con...	co...	pro...	...	r...	succe...	succ...	suc...	s...	s...
1	2022-09-14 10:57:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
2	2022-09-14 11:02:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
3	2022-09-14 11:07:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
4	2022-09-14 11:12:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
5	2022-09-14 11:17:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
6	2022-09-14 11:22:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
7	2022-09-14 11:27:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
8	2022-09-14 11:31:00	bec4c65b0e32f5…	eu-west-2	Default Zone	0.5	0.6666667	<null>	<null>	2	4.816069	3.0811255	3.109064	4.425818	2.954361
9	2022-09-14 11:32:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
10	2022-09-14 11:34:00	bec4c65b0e32f5…	eu-west-2	Default Zone	0.33333334	0.33333334	<null>	<null>	<null>	<null>	<null>	2.170937	4.917796	<null>
11	2022-09-14 11:37:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
12	2022-09-14 11:42:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
13	2022-09-14 11:47:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>
14	2022-09-14 11:52:00	bec4c65b0e32f5…	eu-west-2	Default Zone	<null>	<null>	1	1	<null>	<null>	<null>	<null>	<null>	<null>

Figura 51 Tabela obtida para recursos do tipo DynamoDB

Para além das colunas base (timestamp, resource_id, location e azone) apresentadas existem ainda as colunas com as seguintes métricas, pela mesma ordem apresentadas na Figura 51:

- consumed_read_capacity_units_average
- consumed_write_capacity_units_average
- provisioned_read_capacity_units_average
- provisioned_write_capacity_units_average
- returned_item_count_scan_sum
- successful_request_latency_scan_average
- successful_request_latency_putitem_average
- successful_request_latency_getitem_average

- successful_request_latency_updateitem_average
- successful_request_latency_deleteitem_average

Já no CloudWatch, a informação que retiramos iremos é apresentada como na Figura 52. Ao contrário do que acontece no Google Cloud com o Metrics Explorer, no CloudWatch é possível no mesmo gráfico coexistirem métricas com diferentes unidades de medida.

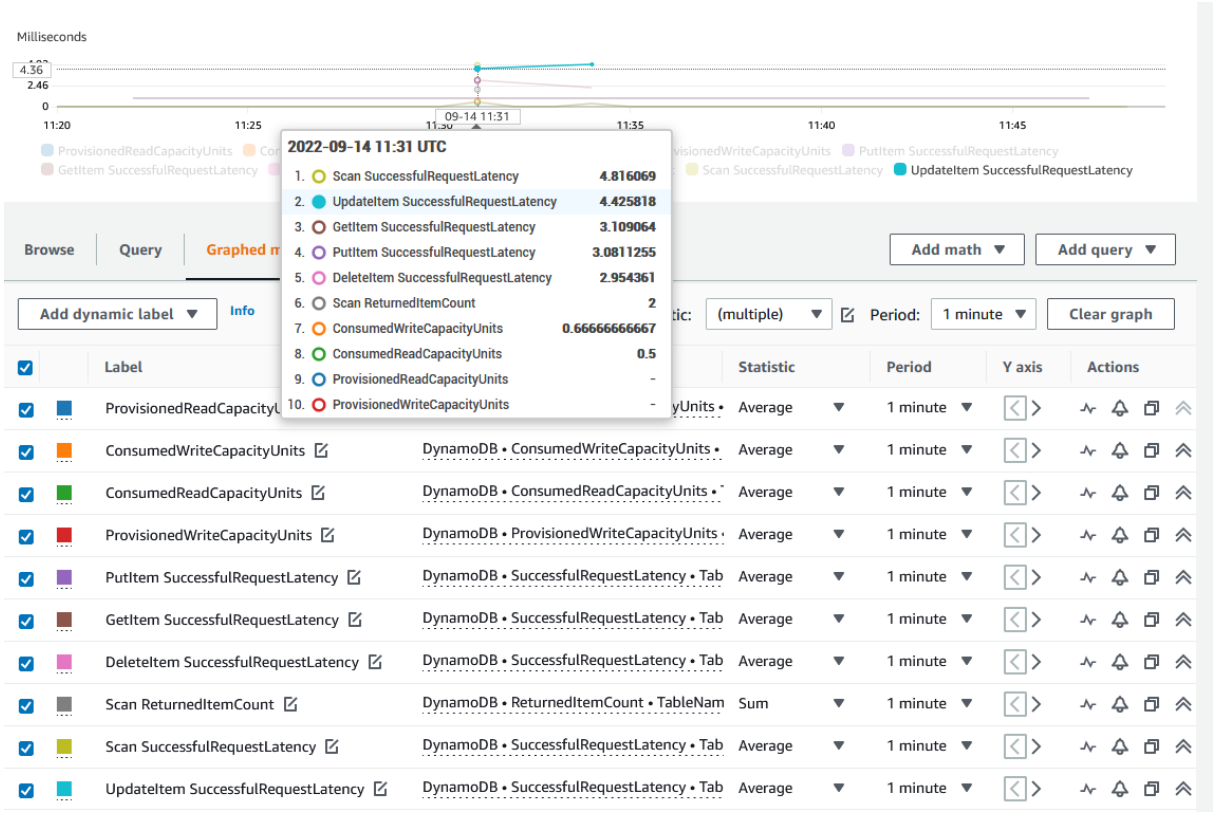


Figura 52 Métricas a retirar do AWS para bases de dados do DynamoDB no CloudWatch

Já para máquinas virtuais obtemos dados como os da Figura 53.

	timestamp	resource_id	l...	...	cpu_utilization_average	disk_read_bytes_total	disk_write_bytes_total	network_in_total	network_out_total
1	2022-09-14 11:29:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.10056497	<null>	<null>	9338	11975
2	2022-09-14 11:34:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.10056497	<null>	<null>	612	600
3	2022-09-14 11:39:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.23389831	<null>	<null>	7198	12081
4	2022-09-14 11:44:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.06723164	<null>	<null>	280	140
5	2022-09-14 11:49:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.10112994	<null>	<null>	2345	2997
6	2022-09-14 11:54:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.26284152	<null>	<null>	52019	65608
7	2022-09-14 11:59:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.06612022	<null>	<null>	446	516
8	2022-09-14 12:04:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.1	<null>	<null>	280	140
9	2022-09-14 12:09:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.06668519	<null>	<null>	1345	786
10	2022-09-14 12:14:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.23278688	<null>	<null>	8163	3829
11	2022-09-14 12:19:00	bbd24c18897c65…	eu-west-2	euw2-az3	0.10001852	<null>	<null>	702	496
12	2022-09-14 11:29:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.3641938	<null>	<null>	145046	245119
13	2022-09-14 11:34:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.1629711	<null>	<null>	145064	246644
14	2022-09-14 11:39:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.2047791	<null>	<null>	144724	244642
15	2022-09-14 11:44:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.228545	<null>	<null>	147200	246629
16	2022-09-14 11:49:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.2001296	<null>	<null>	155725	249467
17	2022-09-14 11:54:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.1667963	<null>	<null>	144461	244282
18	2022-09-14 11:59:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.3713161	<null>	<null>	159020	265839
19	2022-09-14 12:04:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.194998	<null>	<null>	150269	244520
20	2022-09-14 12:09:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.2039548	<null>	<null>	144858	244703
21	2022-09-14 12:14:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.2326361	<null>	<null>	148773	253874
22	2022-09-14 12:19:00	a4fd7004c7e280…	eu-west-2	euw2-az3	1.2001296	<null>	<null>	155906	249800

Figura 53 Tabela obtida para máquinas virtuais

Ao contrário dos outros tipos de recursos, o EC2, o serviço da AWS que suporta máquinas virtuais, por defeito reporta as métricas de cinco em cinco minutos. Existe também a opção para reportar de um em um minuto. Utilizando como exemplo a máquina virtual com o resource_id começado por "bbd24c1", obtemos os dados presentes na Figura 54. Aqui, apesar de as métricas estarem configuradas para um minuto, é possível ver pelos pontos que apenas estão a ser reportadas a cada cinco minutos.

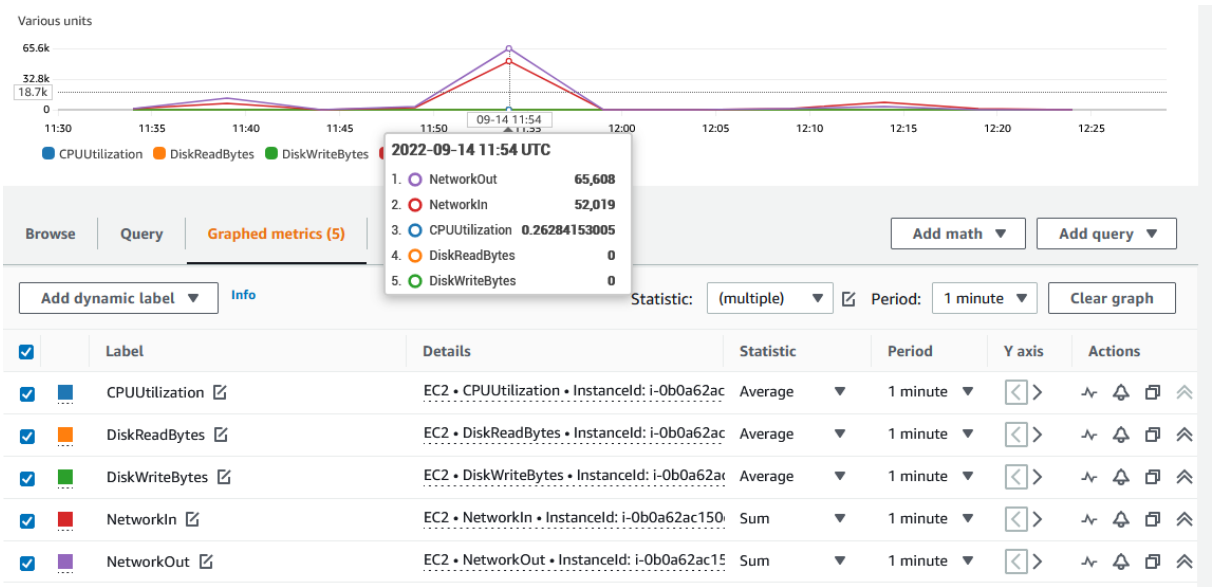


Figura 54 Métricas a retirar do AWS para máquinas virtuais do EC2 no CloudWatch

Já para as funções do AWS Lambda, temos uma tabela como a da Figura 55.

	timestamp	resource_id	location	azone	function_execution_count_total	duration_average
1	2022-09-14 11:37:00	4622586919e237c3…	eu-west-2	Default Zone	4	9.8475
2	2022-09-14 11:38:00	4622586919e237c3…	eu-west-2	Default Zone	32	6.9671874
3	2022-09-14 11:39:00	4622586919e237c3…	eu-west-2	Default Zone	19	1.2947369

Figura 55 Tabela obtida para funções *serverless*

Novamente aqui temos as métricas disponibilizadas ao minuto, como na Figura 56.

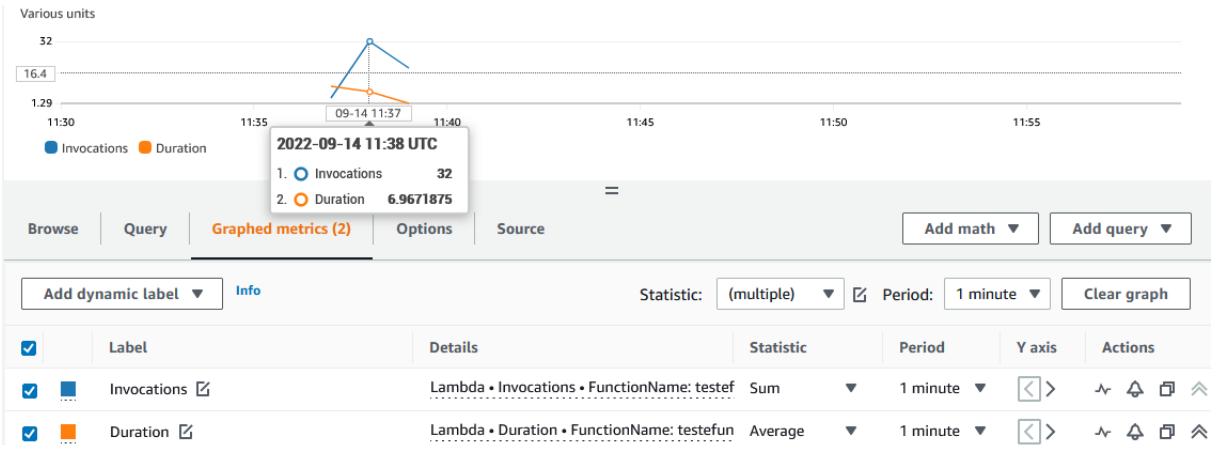


Figura 56 Métricas a retirar do AWS para funções *serverless* no CloudWatch

Relativamente ao armazenamento, a tabela será algo como a da Figura 57.

	timestamp	resource_id	l...	azone	t...	a...	g...	p...	d...	h...	p...	l...	b...	b...	4...	5...
1	2022-09-14 11:36:…	a784906231dcc80…	eu-west-2	Default Zone	<null>	7	<null>	1	<null>	2	<null>	3	31686	2721	<null>	<null>
2	2022-09-14 11:37:…	a784906231dcc80…	eu-west-2	Default Zone	<null>	5	1	<null>	1	<null>	<null>	3	<null>	1076657	<null>	<null>

Figura 57 Tabela obtida para *buckets* de armazenamento do S3

Tal como aconteceu para o DynamoDB, para além das colunas base, temos ainda as seguintes colunas, pela mesma ordem apresentadas na Figura 57:

- total_size_used_total
- all_requests_total
- get_requests_total
- put_requests_total
- delete_requests_total
- head_requests_total
- post_requests_total
- list_requests_total
- bytes_uploaded_total
- bytes_downloaded_total
- 4xx_errors_total
- 5xx_errors_total

Começando pela métrica de total_size_used, esta é a soma do tamanho total das várias classes de armazenamento, e apenas é reportada diariamente. Como para estes testes estamos a buscar os dados métricos para os últimos 60 minutos, não iremos ter quaisquer dados métricos para este valor, tal como mostrado na Figura 58. Seria necessário um período de pelo menos 2880 minutos (ou dois dias) para garantir que iriamos obter os dados desta métrica.

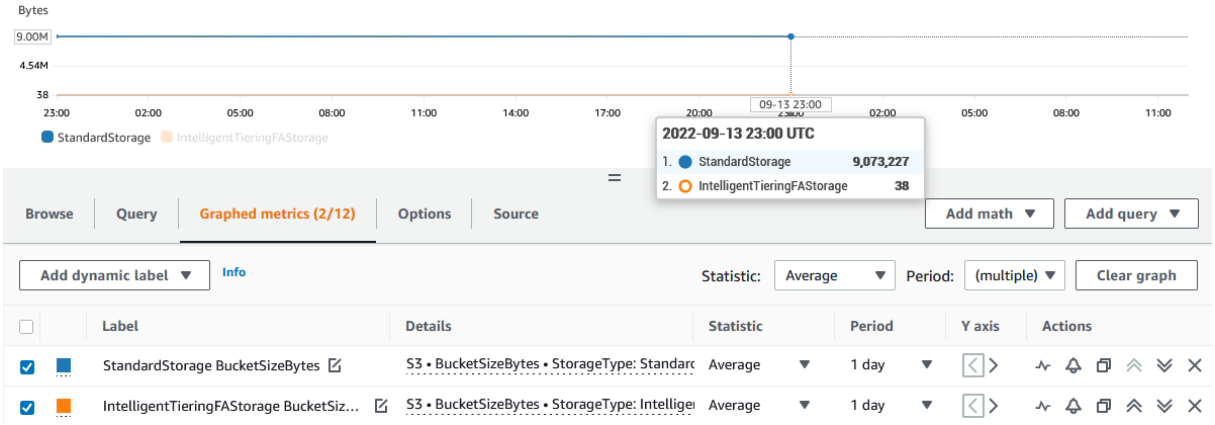


Figura 58 Estatísticas para o total de armazenamento utilizado pelos vários tipos de armazenamento no CloudWatch

Para os restantes dados métricos, os dados apresentados pelo CloudWatch vão corresponder com os dados presentes na tabela obtida, tal como mostra a Figura 59.

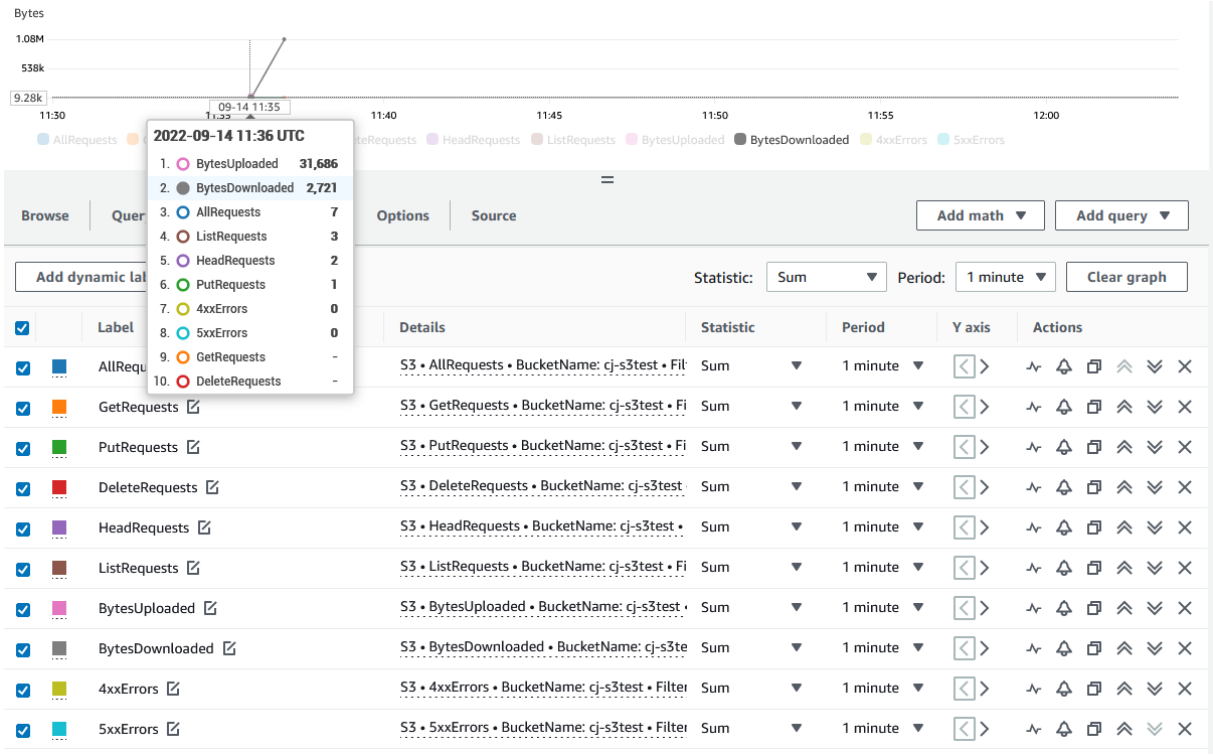


Figura 59 Restantes dados métricos de armazenamento no CloudWatch

Para terminar, para as aplicações web do ElasticBeanstalk iremos obter uma tabela como a da Figura 60. Para facilitar a visualização, os dados obtidos foram truncados até às 11 horas e 50 minutos. Usualmente, nesta tabela iriamos ter cerca de 60 registos em vez de 24. É também possível ver que os dados vindos do EC2 estão a ser reportados apenas de cinco em cinco

	timestamp	resourc...	l...	azone	plan_id	cpu_utiliz...	networ...	net...	h...	h...	l...	r...
1	2022-09-14 11:27:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
2	2022-09-14 11:28:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
3	2022-09-14 11:29:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	1.364193757525368	145046	245119	4	<null>	<null>	4
4	2022-09-14 11:30:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
5	2022-09-14 11:31:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
6	2022-09-14 11:32:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
7	2022-09-14 11:33:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
8	2022-09-14 11:34:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	1.162971195702506	145064	246644	4	<null>	<null>	4
9	2022-09-14 11:35:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	0.693	4
10	2022-09-14 11:36:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
11	2022-09-14 11:37:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
12	2022-09-14 11:38:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
13	2022-09-14 11:39:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	1.20477910530652	144724	244642	4	<null>	<null>	4
14	2022-09-14 11:40:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
15	2022-09-14 11:41:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
16	2022-09-14 11:42:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
17	2022-09-14 11:43:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
18	2022-09-14 11:44:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	1.2285449661945562	147200	246629	4	<null>	<null>	4
19	2022-09-14 11:45:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
20	2022-09-14 11:46:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
21	2022-09-14 11:47:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4
22	2022-09-14 11:48:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	5	<null>	<null>	5
23	2022-09-14 11:49:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	1.2001296656479759	155725	249467	4	<null>	<null>	4
24	2022-09-14 11:50:00	bf16f8d569f753…	eu-west-2	Default Zone	<null>	<null>	<null>	<null>	4	<null>	<null>	4

Figura 60 Tabela com os dados obtidos para aplicações web do ElasticBeanstalk

minutos ao contrário dos restantes, que são reportados a todos os minutos, quando existem dados, como é o caso das métricas de erros 4xx e da latência.

Para além das quatro colunas base, existe também a coluna para o plan_id. Este valor é apenas usado para recursos do Azure. Depois, temos as seguintes colunas para as métricas:

- cpu_utilization_average
- network_in_total
- network_out_total
- http_2xx_total
- http_4xx_total
- latency_average
- requests_total

Como já foi referido nas secções 4.1.7 e 4.2.3, os dados métricos de aplicações vêm do EC2, ELB (Elastic Load Balancer) e do próprio ambiente do Elastic Beanstalk. Como tal, os dados a obter são os apresentados na Figura 61. Nesta figura faz-se referência aos registos para as 11 horas e 44 minutos o que corresponde à linha 18 da Figura 60.

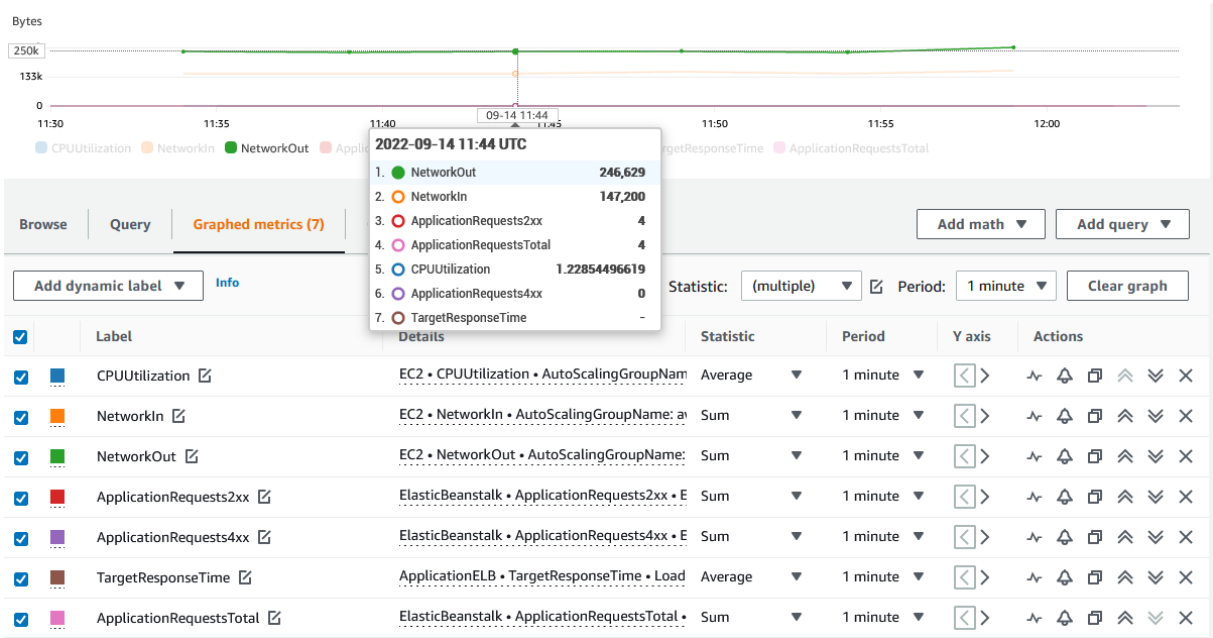


Figura 61 Métricas a retirar para aplicações web no CloudWatch

Também para aplicações web, é impossível obter as métricas para a latência caso o ambiente da aplicação web esteja em modo Single Instance, visto que nesse caso, não existe Load Balancer, que é necessário para obter esta métrica.

5.4 Validação de resultados em ambientes Multi-cloud

Para garantir que esta aplicação funciona num ambiente multi-cloud, vamos então fazer a extração de recursos e métricas para os três *cloud providers*. Para esta validação vão ser utilizados os mesmos recursos usados para validar cada *cloud provider* individualmente.

Começando pelo grafo, e utilizando a mesma consulta usada nas secções anteriores, iremos obter um grafo como o da Figura 62 seguinte.

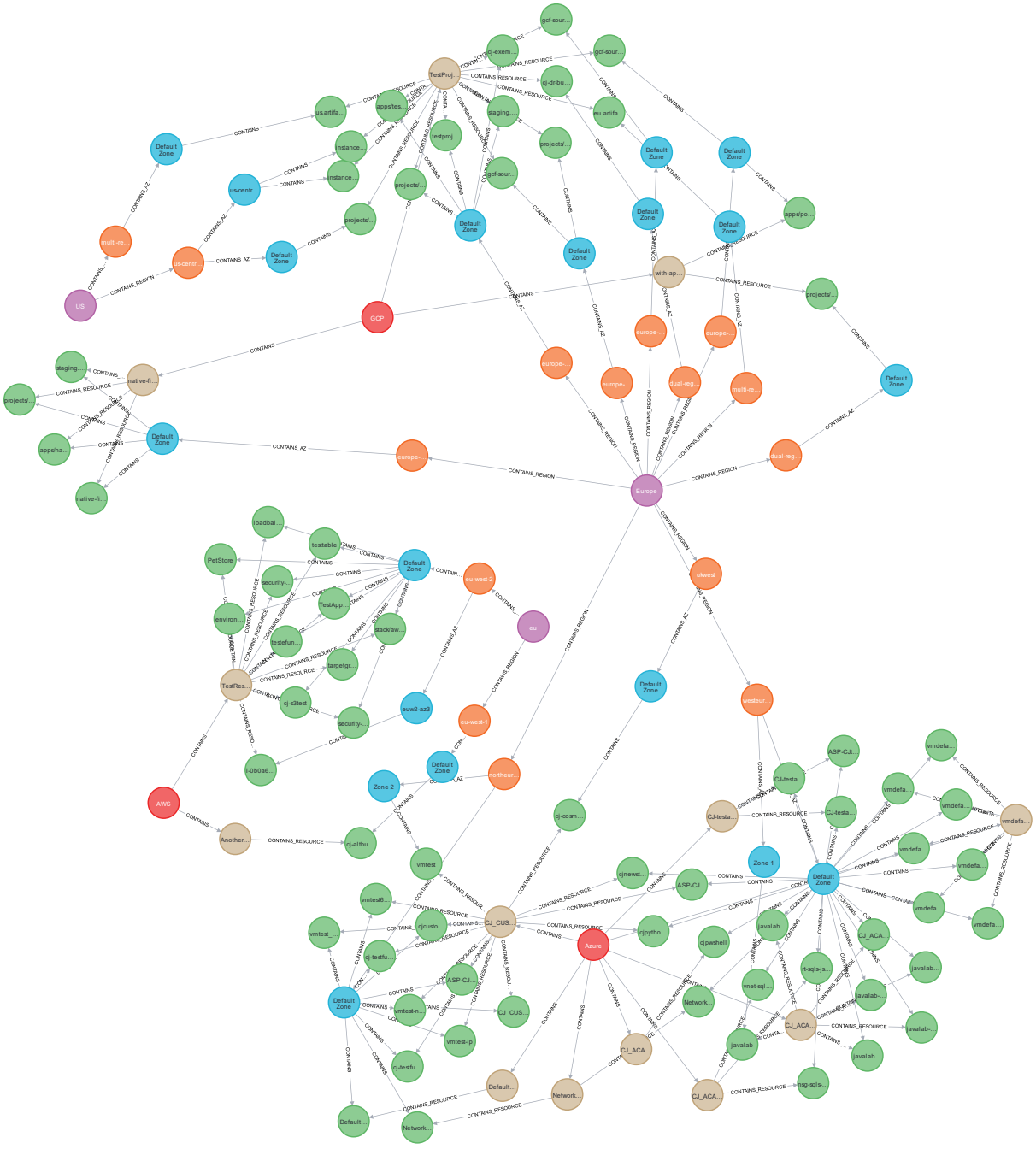


Figura 62 Grafo obtido para um ambiente multi-cloud

Este grafo irá conter os seguintes nós:

- 3 do tipo Cloud, a vermelho, que correspondem aos três *cloud providers* analisados
- 3 do tipo Região Geográfica, a lilás, que correspondem, 1 delas a à região US, e as duas restantes à região europeia, uma delas pertencente ao AWS e outra pertencente Google Cloud e ao Azure
- 15 do tipo Região, a laranja
- 19 do tipo Zona de Disponibilidade, a azul
- 13 do tipo ResourceGroup, a bege
- 72 do tipo Resource, a verde

Também é possível observar no grafo os recursos anteriores distribuídos corretamente pelos seus *resource groups* e *cloud providers* bem como mantendo as suas regiões.

Relativamente aos dados métricos extraídos, podemos conferir estes dados com uma tabela que contenha dados comuns, como por exemplo, a tabela de dados para máquinas virtuais presente na Figura 63.

	timestamp ▲ 1	resource_id	loc...	azone	cpu...	disk_r...	disk_w...	network_i...	network_o...
92	2022-09-26 12:08:00	0312948a34b1b0e…	West Europe	Default Zone	0.39	<null>	102425	41289	59792
93	2022-09-26 12:09:00	c33e52c8b536636…	us-central1	us-central1-c	0.58764786	<null>	29717	2154	2208
94	2022-09-26 12:09:00	0312948a34b1b0e…	West Europe	Default Zone	0.24	<null>	49258.07	40976	56041
95	2022-09-26 12:10:00	c33e52c8b536636…	us-central1	us-central1-c	0.5269439	<null>	2437	1605	1568
96	2022-09-26 12:10:00	0312948a34b1b0e…	West Europe	Default Zone	0.345	<null>	94239.75	40976	56041
97	2022-09-26 12:11:00	c33e52c8b536636…	us-central1	us-central1-c	0.5308395	<null>	<null>	1391	1275
98	2022-09-26 12:11:00	0312948a34b1b0e…	West Europe	Default Zone	0.37	<null>	139336.56	40856	55983
99	2022-09-26 12:12:00	bbd24c18897c652…	eu-west-2	euw2-az3	0.06723164	<null>	<null>	1199	682
100	2022-09-26 12:12:00	c33e52c8b536636…	us-central1	us-central1-c	0.5418878	<null>	18739	6638	2217
101	2022-09-26 12:12:00	0312948a34b1b0e…	West Europe	Default Zone	0.36	<null>	262203.12	41289	58503
102	2022-09-26 12:13:00	c33e52c8b536636…	us-central1	us-central1-c	0.5184939	<null>	1741	1861	1342
103	2022-09-26 12:13:00	0312948a34b1b0e…	West Europe	Default Zone	0.38	<null>	131104.33	41375	59845
104	2022-09-26 12:14:00	c33e52c8b536636…	us-central1	us-central1-c	0.5102806	<null>	<null>	1486	1252
105	2022-09-26 12:14:00	0312948a34b1b0e…	West Europe	Default Zone	0.355	<null>	106526.76	40856	55983
106	2022-09-26 12:15:00	c33e52c8b536636…	us-central1	us-central1-c	0.4941991	<null>	<null>	1818	1252
107	2022-09-26 12:15:00	0312948a34b1b0e…	West Europe	Default Zone	0.35	<null>	94244.84	40856	55983
108	2022-09-26 12:16:00	c33e52c8b536636…	us-central1	us-central1-c	0.54696506	<null>	<null>	6468	2122
109	2022-09-26 12:16:00	0312948a34b1b0e…	West Europe	Default Zone	0.355	<null>	98347.39	40942	56037
110	2022-09-26 12:17:00	bbd24c18897c652…	eu-west-2	euw2-az3	0.10007416	<null>	<null>	702	496
111	2022-09-26 12:17:00	c33e52c8b536636…	us-central1	us-central1-c	0.5894368	<null>	108686	2186	1333
112	2022-09-26 12:17:00	0312948a34b1b0e…	West Europe	Default Zone	0.38	<null>	290868.28	41378	57484
113	2022-09-26 12:18:00	c33e52c8b536636…	us-central1	us-central1-c	0.5679789	<null>	362388	1920	1317
114	2022-09-26 12:18:00	0312948a34b1b0e…	West Europe	Default Zone	0.4	<null>	294874.03	41375	61573
115	2022-09-26 12:19:00	c33e52c8b536636…	us-central1	us-central1-c	0.52523345	<null>	40230	1871	1324
116	2022-09-26 12:19:00	0312948a34b1b0e…	West Europe	Default Zone	0.355	<null>	114719.6	40856	55983
117	2022-09-26 12:20:00	c33e52c8b536636…	us-central1	us-central1-c	0.5172224	<null>	696	6628	2242
118	2022-09-26 12:20:00	0312948a34b1b0e…	West Europe	Default Zone	0.385	<null>	98323.93	42471	58318
119	2022-09-26 12:21:00	c33e52c8b536636…	us-central1	us-central1-c	0.5298088	<null>	<null>	2237	1409
120	2022-09-26 12:21:00	0312948a34b1b0e…	West Europe	Default Zone	0.395	<null>	417799.72	42501	58264
121	2022-09-26 12:22:00	bbd24c18897c652…	eu-west-2	euw2-az3	0.101149425	<null>	<null>	2439	2930

Figura 63 Dados obtidos para um ambiente multi-cloud

Aqui podemos ver os dados os três *cloud providers* já agrupados numa mesma tabela. É possível distinguir qual a origem dos dados através das nomenclaturas utilizados para as localizações. Neste caso teremos "West Europe" para recursos no Azure, "us-central1" para

recursos no Google Cloud e "eu-west-2" para recursos no AWS. É possível afirmar que estes dados estão corretos pois individualmente estes dados estão corretos, como pode ser visto nas secções 5.1, 5.2 e 5.3 e não existem operações que modifiquem os dados já existentes, sendo apenas feitas inserções nas tabelas. Da mesma forma, os dados também irão estar corretos para as restantes tabelas.

Para estas tabelas podem ser executadas consultas de dados para obter por exemplo, os dados apenas para um período de tempo, agregar os dados por localização, selecionar apenas certas colunas, entre outras.

As consultas ao grafo e à base de dados são também são iguais, qualquer que seja o *cloud provider*. Por exemplo a consulta seguinte para obter um recurso do grafo irá ser igual, quer o recurso esteja guardado no Azure, AWS ou Google Cloud.

```
MATCH (r:Resource)
WHERE r.id = "arn:aws:ec2:eu-west-2:191215594086:instance/i-
0b0a62ac150e96853"
RETURN r
```

Utilizando o recurso devolvido no exemplo anterior, podemos usar o dbid deste para procurar métricas na base de dados. Sendo o recurso anterior devolvido uma máquina virtual, podemos entrar obter métricas como o uso de CPU, como indicado na consulta seguinte, obtendo os dados da Figura 64. Isto é válido para qualquer que seja o recurso, em qualquer dos três *cloud providers* analisados. É ainda possível fazer a operação inversa em que se verifica primeiro que recursos têm metricas a necessitar de atenção e a partir daí se obter o recurso e suas ligações no grafo.

```
SELECT timestamp, cpu_utilization_average
FROM common_vm
WHERE resource_id = 'bbd24c18897c652addc8322edf155c441ac-
cbf49bc8697f930ad278a8f8876cd'
```

	timestamp	cpu_utilization...
1	2022-09-26 11:47:00	0.083333336
2	2022-09-26 11:52:00	0.86723167
3	2022-09-26 11:57:00	0.19781421
4	2022-09-26 12:02:00	0.06723164
5	2022-09-26 12:07:00	0.099489704
6	2022-09-26 12:12:00	0.06723164
7	2022-09-26 12:17:00	0.10007416
8	2022-09-26 12:22:00	0.101149425

Figura 64 Resultados obtidos com a execução da consulta

6

Conclusões e trabalho futuro

O objetivo deste trabalho foi a criação de um modelo homogéneo para guardar recursos e métricas de vários *cloud providers*. Com este trabalho conseguiu-se produzir não só uma aplicação que é capaz de extrair recursos, as regiões destes e as suas métricas, mas também um modelo para um grafo que é genérico o suficiente para acomodar recursos, regiões e as relações entre estes para cada *cloud provider* considerado (AWS, GCP e Azure). Foi também feito um modelo semelhante de forma a que possa ser possível guardar, numa mesma tabela de uma base de dados, dados métricos pertencentes aos três *cloud providers* para um mesmo tipo de recursos. Para ser possível especificar todas as métricas que devem ser extraídas foi criada uma linguagem de representação de métricas, que é utilizada durante a execução da aplicação e que a informa para todos os tipos de recurso, quais são as métricas a extrair para este, quais agregações a utilizar, quais são os filtros ou séries temporais necessárias para obter a informação correta e também em qual tabela da base de dados os dados de uma tal métrica devem ser guardados.

Durante o desenvolvimento deste trabalho também foram encontrados alguns desafios para serem endereçados. O primeiro problema está relacionado com a uniformização de regiões geográficas e de regiões como já referido na secção 4.1.4. Perante a forma como os três *cloud providers* dispõem a informação relativa às suas regiões disponíveis, não foi possível uniformizar corretamente as regiões entre os três *cloud providers*. Isto deve-se principalmente a dois motivos:

- Nem todos os *cloud providers* classificam uma região como parte de uma região geográfica equivalente. Como por exemplo: enquanto que o Azure e o AWS classificam regiões australianas como pertencentes à região geográfica "Asia Pacific" e "ap" respetivamente, o Google Cloud classifica esta região como a sua própria região geográfica ficando assim, nós com regiões australianas pertencendo a diferentes regiões geográficas.

- Nem todos os *cloud providers* têm os mesmos nomes para regiões geográficas idênticas. Como referido anteriormente, o Azure e o AWS utilizam nomes diferentes para regiões asiáticas (e o Google Cloud também utiliza um nome diferente dos outros dois). Outro exemplo é o uso de "europe" para o Azure e Google Cloud e o uso de "eu" para o AWS.

Isto faz com que, sem mapeamento manual, esta informação seja difícil de uniformizar e foi algo que não foi realizada neste trabalho. Para além disso, como o Google Cloud e o AWS não disponibilizam nenhuma maneira de obter o nome ou localização dos seus *data centers* na API, isto faz com que, para obter regiões geográficas, seja necessário cortar a string com o nome da região.

Se os três *cloud providers* disponibilizassem o nome da região de cada *data center* seria possível uniformizar os nós de regiões e assim teríamos, por exemplo, apenas um nó de região para Paris, que seria utilizado pelos três *cloud providers*. No entanto, mesmo com esta informação a ser disponibilizada, continuaria a ser impossível uniformizar as regiões geográficas visto que estas continuariam a utilizar nomenclaturas diferentes.

No entanto, o problema principal é a uniformização de dados métricos. Para recursos como bases de dados, visto que cada *cloud provider* tem o seu próprio sistema (neste trabalho foram analisados o AWS DynamoDB, Google Firestore e Azure CosmosDB), e, consequentemente, tendo um espectro de métricas diferentes entre si, isto faz com que seja muito difícil de criar um sistema universal, isto é, em que se consiga numa mesma tabela guardar métricas dos três *cloud providers*.

Algo semelhante aconteceu também com outros tipos de recursos que foram analisados. Inicialmente, previa-se incluir também recursos do tipo Load Balancer, no entanto, após feita uma análise, foi visto que são muito poucas as métricas que podem ser uniformizadas e nunca entre os três *cloud providers*. O mesmo acontece para *clusters* Kubernetes, que neste caso, apesar de serem todos sistemas iguais, ao contrário do que acontece com os sistemas de bases de dados, são poucas as métricas disponibilizadas pelos *cloud providers* que podem realmente ser uniformizadas. Uma possível solução para este caso seria a extração de métricas diretamente dos *clusters* Kubernetes em vez de se estar dependente da informação disponível pelos *cloud providers*. Outra vantagem disto é que assim os dados já estariam uniformizados visto que as métricas que seriam reportadas são as mesmas para cada em todos os *clusters* Kubernetes. As métricas analisadas para ambos estes casos estão no apêndice A.2.

Como trabalho futuro seria interessante que fosse feito um melhor mapeamento de regiões entre os vários *cloud providers* de forma a ser possível uniformizar também esta informação e evitar nós duplicados para regiões semelhantes entre vários *cloud providers*.

Algo que também poderia ser feito seria a adição de mais níveis ao grafo, com por exemplo, mais informação sobre um determinado recurso (por exemplo, no caso de uma base de dados, as suas tabelas ou coleções) e a extração de informação especifica para este.

Para terminar, poderiam ainda ser analisadas métricas a uniformizar para mais tipos de recursos para além dos que foram analisados neste trabalho.

BIBLIOGRAFIA

[1] D. C. Marinescu, Cloud computing : theory and practice. Waltham Ma: Morgan Kaufmann, 2013.

[2] Cloud Security Alliance. "Security guidance for critical areas of focus in cloud computing V3.0." https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf, 2011.

[3] M. J. Sydor, APM Best Practices : Realizing Application Performance Management. Berkeley, Ca: Apress, 2011.

[4] Christoph Heger, André van Hoorn, Mario Mann e Dušan Okanovi . "Application Performance Management: State of the Art and Challenges for the Future". Em: ICPE '17: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, April 2017, L'Aquila, Italy. pp. 429–432. isbn: 78-1-4503-4404-3. doi: https://doi.org/10.1145/3030207.3053674.

[5] G. Linden. Marissa Mayer at Web 2.0. http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html, 2006.

[6] Y. Einav. "Amazon found every 100ms of latency cost them 1% in sales.," GigaSpaces, Jan. 20, 2019. https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales.

[7] B. Cutler. Firefox and page load speed (part I). https://blog.mozilla.org/metrics/2010/03/31/refox-page-load-speed-part-i/, 2010.

[8] A. Silberschatz, H. F. Korth e S. Sudarshan. Database System Concepts. Sixth. McGraw-Hill, 2010. isbn: 0-07-352332-1.

[9] Amazon CloudWatch Documentation, 2019. https://docs.aws.amazon.com/cloudwatch/index.html.

[10] "Google Cloud metrics | Cloud Monitoring," Google Cloud. https://cloud.google.com/monitoring/api/metrics_gcp.

[11] “APIs & Reference | Cloud Monitoring,” Google Cloud. https://cloud.google.com/monitoring/docs/apis (accessed Feb. 15, 2022).

[12] “TimeSeries | Cloud Monitoring,” Google Cloud. https://cloud.google.com/monitoring/api/ref_v3/rest/v3/TimeSeries (accessed Feb. 15, 2022).

[13] “APIs & Reference | Cloud Asset Inventory Documentation,” Google Cloud. https://cloud.google.com/asset-inventory/docs/apis (accessed Feb. 15, 2022).

[14] “Azure REST API Reference,” Microsoft.com, May 24, 2019. https://docs.microsoft.com/en-us/rest/api/azure/.

[15] “Overview - Azure Resource Manager,” Microsoft.com, Aug. 29, 2019. https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview.

[16] “Organize your resources with management groups - Azure Governance - Azure governance,” docs.microsoft.com. https://docs.microsoft.com/en-us/azure/governance/management-groups/overview.

[17] ClickHouse, “Overview | ClickHouse Documentation,” Clickhouse.com, 2022. https://clickhouse.com/docs/en/ (accessed Feb. 15, 2022).

[18] R. Angles and C. Gutierrez, “An Introduction to Graph Data Management,” Data-Centric Systems and Applications, pp. 1–32, 2018, doi: 10.1007/978-3-319-96193-4_1.

[19] M. A. Rodriguez and P. Neubauer. Constructions from dots and lines. Bul. Am. Soc. Info. Sci. Tech., 36(6):35{41, 2010.

[20] N. Leavitt, “Will NoSQL Databases Live Up to Their Promise?,” Computer, vol. 43, no. 2, pp. 12–14, Feb. 2010, doi: 10.1109/mc.2010.58.

[21] Z. Li, “NoSQL Databases,” Geographic Information Science & Technology Body of Knowledge, vol. 2018, no. Q2, Apr. 2018, doi: 10.22224/gistbok/2018.2.10.

[22] T. P. Sanaboyina, “Performance Evaluation of Time series Databases based on Energy Consumption,” 2016.

[23] “Application Performance Management,” IBM. https://www.ibm.com/cloud/learn/application-performance-management.

[24] “The state of cloud profitability has never been stronger,” TBR, Jun. 18, 2019. https://tbri.com/special-reports/cloud-vendor-profitability/.

[25] J. Novet, “How Amazon’s cloud business generates billions in profit,” CNBC, Sep. 05, 2021. https://www.cnbc.com/2021/09/05/how-amazon-web-services-makes-money-estimated-margins-by-service.html.

[26] “What is the Definition of OLAP? OLAP Definition,” OLAP.com, 2018. https://olap.com/olap-definition/.

[27] M. Angelini, S. Bonomi, C. Ciccotelli, and A. Palma, “Toward a Context-Aware Methodology for Information Security Governance Assessment Validation,” Cyber-Physical Security for Critical Infrastructures Protection, pp. 171–187, 2021, doi: 10.1007/978-3-030-69781-5_12.

[28] A. C. Kinsley, G. Rossi, M. J. Silk, and K. VanderWaal, “Multilayer and Multiplex Networks: An Introduction to Their Use in Veterinary Epidemiology,” Frontiers in Veterinary Science, vol. 7, Sep. 2020, doi: 10.3389/fvets.2020.00596.

[29] M. De Domenico, A. Sole-Ribalta, S. Gomez, and A. Arenas, “Navigability of interconnected networks under random failures,” Proceedings of the National Academy of Sciences, vol. 111, no. 23, pp. 8351–8356, May 2014, doi: 10.1073/pnas.1318469111.

[30] B. Oselio, A. Kulesza, and A. O. Hero, “Multi-Layer Graph Analysis for Dynamic Social Networks,” IEEE Journal of Selected Topics in Signal Processing, vol. 8, no. 4, pp. 514–523, Aug. 2014, doi: 10.1109/jstsp.2014.2328312.

[31] “What is Multi-cloud? Multi-cloud Definition and Related FAQs,” Avi Networks. https://avinetworks.com/glossary/multi-cloud/

[32] “What Is Vendor Lock-In? | Vendor Lock-In and Cloud Computing | Cloudflare,” Cloudflare.

[33] J. Opara-Martins, R. Sahandi, and F. Tian, “Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective,” Journal of Cloud Computing, vol. 5, no. 1, Apr. 2016, doi: 10.1186/s13677-016-0054-z.

[34] M. Kivela, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, “Multilayer networks,” Journal of Complex Networks, vol. 2, no. 3, pp. 203–271, Jul. 2014, doi: 10.1093/comnet/cnu016.

[35] Z. Hammoud and F. Kramer, “Multilayer networks: aspects, implementations, and application in biomedicine,” Big Data Analytics, vol. 5, no. 1, Jul. 2020, doi: 10.1186/s41044-020-00046-0.

[36] “GetMetricData - Amazon CloudWatch,” docs.aws.amazon.com. https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_GetMetricData.html (accessed Feb. 17, 2022).

[37] M. Obitko, “RDF Graph and Syntax - Introduction to ontologies and semantic web - tutorial,” www.obitko.com. https://www.obitko.com/tutorials/ontologies-semantic-web/rdf-graph-and-syntax.html (accessed Feb. 17, 2022).

[38] R. Bouhali and A. Laurent, “Exploiting RDF Open Data Using NoSQL Graph Databases,” IFIP Advances in Information and Communication Technology, pp. 177–190, 2015, doi: 10.1007/978-3-319-23868-5_13.

[39] “What is AWS CloudFormation? - AWS CloudFormation,” docs.aws.amazon.com. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html.

[40] "Amazon Resource Names (ARNs)", Amazon.com, 2020. https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html.

[41] A. Hayes, “Understanding Time Series,” Investopedia, Mar. 31, 2020. https://www.investopedia.com/terms/t/timeseries.asp.

[42] C. Strauch, "NoSQL Databases", 2011

[43] INSERT INTO Statement | ClickHouse Docs. (2016). Clickhouse.com. https://clickhouse.com/docs/en/sql-reference/statements/insert-into#performance-considerations

[44] Regions and Zones - Amazon Elastic Compute Cloud. (n.d.). Docs.aws.amazon.com. Retrieved August 22, 2022, from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html#az-ids

[45] App Engine Locations | App Engine Documentation. (n.d.). Google Cloud. https://cloud.google.com/appengine/docs/locations

[46] What Is Amazon EC2 Auto Scaling? - Amazon EC2 Auto Scaling. (2019). Amazon.com. https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html

[47] Benefits of Auto Scaling - Amazon EC2 Auto Scaling. (n.d.). Docs.aws.amazon.com. https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-benefits.html

[48] Metrics Insights limits - Amazon CloudWatch. (n.d.). Docs.aws.amazon.com. Retrieved August 22, 2022, from https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch-metrics-insights-limits.html

[49] Unix Time Stamp - Epoch Converter. (n.d.). Www.unixtimestamp.com. https://www.unixtimestamp.com/

[50] Smartscape. (2017, July 19). Dynatrace Docs. https://www.dynatrace.com/support/help/how-to-use-dynatrace/smartscape

[51] Deliver API with Confidence. (n.d.). www.seekret.io. Retrieved August 29, 2022, from https://www.seekret.io/product

A

Apêndice

A.1 Métricas a serem retiradas dos *cloud providers*

As métricas identificadas com verde são aquelas que são uniformizáveis e estão a ser extraídas pela aplicação. As seguintes tabelas mostram as métricas a serem retiradas para recursos dos tipos máquinas virtuais, armazenamento, funções serverless, aplicações web e *api gateways*, por esta ordem. As métricas sombreadas a verde são aquelas que podem-se considerar homogéneas e que estão a ser retiradas e uniformizadas.

AWS/EC2	Unit	Aggregation	compute.googleapis.com/	Unit	Aggregation	Microsoft.Compute/virtualMachines	Unit	Aggregation
CPUUtilization	Percent		instance/cpu/utilization	0.0-1.0		Percentage CPU	Percent	Average
DiskReadOps	Count		instance/disk/read_ops_count	Count		Disk Read Operations/Sec	CountPerSecond	Average
DiskWriteOps	Count		instance/disk/write_ops_count	Count		Disk Write Operations/Sec	CountPerSecond	Average
DiskReadBytes	Bytes		instance/disk/read_bytes_count	Bytes		Disk Read Bytes	Bytes	Total
DiskWriteBytes	Bytes		instance/disk/write_bytes_count	Bytes		Disk Write Bytes	Bytes	Total
			instance/memory/balloon/ram_used	Bytes				
NetworkIn	Bytes		instance/network/received_bytes_count	Bytes		Network In Total	Bytes	Total
NetworkOut	Bytes		instance/network/sent_bytes_count	Bytes		Network Out Total	Bytes	Total
NetworkPacketsIn	Count		instance/network/received_packets_count	Count				
NetworkPacketsOut	Count		instance/network/sent_packets_count	Count				
			instance/uptime_total	Seconds				

AWS/S3	Unit	Aggregation	storage.googleapis.com/	Unit	Aggregation	Microsoft.Storage/storageAccounts/blobServices	Unit	Aggregation
BucketSizeBytes*	Bytes	Average	storage/total_bytes*	Bytes		BlobCapacity	Bytes	Average
AllRequests	Count	Sum	api/request_count	Count		Transactions	Count	Total
GetRequests			api/request_count (with *method* specified)			Transactions (with *ApiName* specified)		
PutRequests								
DeleteRequests								
HeadRequests								
PostRequests								
ListRequests								
BytesUploaded	Bytes	Average, Sum, Count, Min, Max, p0.0-p99.9	network/received_bytes_count	Bytes		Ingress	Bytes	Total
BytesDownloaded			network/sent_bytes_count	Bytes		Egress		
4xxErrors	Count	Average, Sum, Min, Max, Count	api/request_count (with *response_code* specified)	Count		Transactions (with *ResponseType* specified)	Count	Total
5xxErrors								
TotalRequestLatency	Milliseconds	Average, Sum, Min, Max, Count,				SuccessE2ELatency	Milliseconds	Average

		p0.0-p100						
BytesPendingReplication	Bytes	Max						
OperationsPendingReplication	Count	Max						

AWS/Lambda	Unit	Aggregation	cloudfunctions.googleapis.com/	Unit	Aggregation	Microsoft.Web/sites	Unit	Aggregation
Invocations Errors	Count	Sum	function/execution_count (with *status* specified)	Count		FunctionExecutionCount	Count	Total
Duration	Milliseconds (?)	Average, Max, p0.0-p9.99?	function/execution_times	Nanoseconds		HttpResponseTime		
ConcurrentExecutions	Count	Max	function/instance_count (with state = active)	Count		FunctionExecutionUnits		
			function/network_egress	Bytes		BytesSent	Bytes	Total

AWS/ElasticBeanstalk	Unit	Aggregation	appengine.googleapis.com/	Unit	Aggregation	Microsoft.Web/sites	Unit	Aggregation
CPUUtilization*	Percent		system/cpu/utilization or flex/cpu/utilization	0.0-1.0		CPU Percentage*	Percent?	
DiskReadBytes*	Bytes		flex/disk/read_bytes_count	Bytes		IoReadBytesPerSecond	BytesPerSecond	Total
DiskReadOps*	Count					IoReadOperationsPerSecond		

DiskWriteBytes*	Bytes		flex/disk/write_bytes_count	Bytes		IoWriteBytesPerSecond		
DiskWriteOps*	Count					IoWriteOperations-PerSecond		
NetworkIn*	Bytes		system/network/received_bytes_count	Bytes		BytesReceived	Bytes	Total
NetworkOut*	Bytes		system/network/sent_bytes_count	Bytes		BytesSent		
ApplicationRequests2xx	Count		http/server/response_count (with *response_code* specified)	Count		Http2xx	Count	Total
ApplicationRequests4xx	Count					Http4xx		
TargetResponseTime***	Seconds		http/server/response_latencies	Milliseconds		HttpResponseTime	Seconds	Average
ApplicationRequestsTotal	Count		http/server/response_count	Count		Requests	Count	Total
			flex/instance/connections/current	Count		AppConnections	Count	Average

*Métricas disponibilizadas no *namespace:* AWS/EC2

**Métricas disponibilizadas no *namespace:* Microsoft.Web/serverFarms

***Métricas disponibilizadas no *namespace*: AWS/ApplicationELB

AWS/ApiGateway	apigateway.googleapis.com/	Microsoft.ApiManagement/service
4XXError	proxy/request_count (with response_code_class specified)	Requests (with BackendResponseCodeCategory specified)
5XXError		
Count		Requests
Latency		Duration

Já para sistemas de bases de dados, a verde estão as métricas que estão a ser recolhidas

AWS/DynamoDB	Unit	Aggregation	Dimensions	Update Interval
AgeOfOldestUnreplicatedRecord	Milliseconds	Avg, Min, Max	TableName, DelegatedOperation	5 minutes
FailedToReplicateRecordCount	Count	Avg, Min, Max, SampleCount	TableName, DelegatedOperation	
PendingReplicationCount	Count	Avg, SampleCount, Sum	TableName, ReceivingRegion	
ReplicationLatency	Milliseconds	Avg, Min, Max	TableName, ReceivingRegion	
ReturnedBytes	Bytes	Avg, Min, Max, Sample Count, Sum	Operation, StreamLabel, TableName	1 minute
ReturnedItemCount	Count	Avg, Min, Max, SampleCount, Sum	TableName, Operation	
ReturnedRecordsCount	Count	Avg, Min, Max, SampleCount, Sum	Operation, StreamLabel, TableName	
SuccessfulRequestLatency	Milliseconds	Avg, Min, Max, SampleCount	TableName, Operation	
ThrottledRequests	Count	SampleCount, Sum	TableName, Operation	
TransactionConflict	Count	Sum, SampleCount, Min, Max, Average	TableName	
SystemErrors	Count	SampleCount, Sum	TableName, Operation	
UserErrors	Count	SampleCount, Sum	TableName, Operation (?)	
ReadThrottleEvents	Count	SampleCount, Sum	TableName, GlobalSecondaryIndexName	
WriteThrottleEvents	Count	SampleCount, Sum	TableName, GlobalSecondaryIndexName	
ConsumedReadCapacityUnits	Count	Avg, Min, Max, SampleCount, Sum	TableName, GlobalSecondaryIndexName	
ConsumedWriteCapacityUnits	Count	Avg, Min, Max, SampleCount, Sum	TableName, GlobalSecondaryIndexName	

MaxProvisionedTableReadCapacityUtilization	Percent	Avg, Min, Max		5 minutes
MaxProvisionedTableWriteCapacityUtilization	Percent	Avg, Min, Max		
OnlineIndexConsumedWriteCapacity	Count	Avg, Min, Max, SampleCount, Sum	TableName, GlobalSecondaryIndexName	
OnlineIndexPercentageProgress	Count	Avg, Min, Max, SampleCount, Sum	TableName, GlobalSecondaryIndexName	
OnlineIndexThrottleEvents	Count	Avg, Min, Max, SampleCount, Sum	TableName, GlobalSecondaryIndexName	
ProvisionedReadCapacityUnits	Count	Avg, Min, Max	TableName, GlobalSecondaryIndexName	
ProvisionedWriteCapacityUnits	Count	Avg, Min, Max	TableName, GlobalSecondaryIndexName	

firestore.googleapis.com/	Unit	Aggregation	Dimensions	Update Interval
api/request_count	Count		api_method, response_code	1 minute
document/delete_count	Count		module, version	
document/read_count	Count		module, version, type	
document/write_count	Count		module, version, operation	
network/active_connections	Count		module, version	

Microsoft.DocumentDB/databaseAccounts	Unit	Aggregation	Dimensions	Update Interval
TotalRequests	Count	Count	DatabaseName, CollectionName, Region, StatusCode	1 minute
MetadataRequests	Count	Count	DatabaseName, CollectionName, Region, StatusCode	
TotalRequestUnits	Count	Total	DatabaseName, CollectionName, Region, StatusCode	
ProvisionedThroughput	Count	Max	DatabaseName, ContainerName	5 minutes
AvailableStorage	Bytes	Total	DatabaseName, CollectionName, Region	

DataUsage	Bytes	Total	DatabaseName, CollectionName, Region	
IndexUsage	Bytes	Total	DatabaseName, CollectionName, Region	
DocumentQuota	Bytes	Total	DatabaseName, CollectionName, Region	
DocumentCount	Count	Total	DatabaseName, CollectionName, Region	
ReplicationLatency	Milliseconds	Avg, Min, Max	SourceRegion, TargetRegion	1 minute
ServerSideLatency	Milliseconds	Avg, Min, Max	CollectionName, ConnectionMode, DatabaseName, OperationType, PublicAPIType, Region	
ServiceAvailability	Percent	Min, Max		1 hour

A.2 Métricas analisadas que não serão retiradas dos *cloud providers*

Para load balancers:

Classic Load Balancer - AWS/ELB	loadbalancing.googleapis.com/	Microsoft.Network/loadBalancers
HealthyHostCount		
HTTPCode_Backend_2XX	https/request_count (with response_code_class specified)	
HTTPCode_Backend_3XX		
HTTPCode_Backend_4XX		
HTTPCode_Backend_5XX		
HTTPCode_ELB_4XX		
HTTPCode_ELB_5XX		
	https/request_bytes_count	
	https/response_bytes_count	

Latency	https/total_latencies? or tcp_ssl_proxy/frontend_tcp_rtt?	
RequestCount	https/request_count	
		ByteCount
		PacketCount
SpilloverCount		
SurgeQueueLength		
UnHealthyHostCount		

Para Kubernetes:

EKS	GKS	AKS
cluster_failed_node_count		
cluster_node_count		
node_cpu_limit	node/cpu/allocatable_cores	kube_node_status_allocatable_cpu_cores
node_cpu_reserved_capacity		
node_cpu_usage_total	node/cpu/total_cores	
node_cpu_utilization		
	node/cpu/core_usage_time	
node_filesystem_utilization		
node_memory_limit	node/memory/total_bytes (?)	kube_node_status_allocatable_memory_bytes
node_memory_reserved_capacity		
	node/memory/allocatable_bytes (?)	
node_memory_utilization (%)	node/memory/allocatable_utilization (0.0-1.0)	
node_memory_working_set	node/memory/used_bytes	
node_network_total_bytes	node/network/sent_bytes_count + node/network/received_bytes_count	

	node/pid_limit	
	node/pid_used	
node_number_of_running_containers		
node_number_of_running_pods		
pod_cpu_reserved_capacity		
pod_cpu_utilization		
pod_cpu_utilization_over_pod_limit		
pod_memory_reserved_capacity		
pod_memory_utilization		
pod_memory_utilization_over_pod_limit		
pod_number_of_container_restarts		
pod_network_rx_bytes	pod/network/received_bytes_count	
pod_network_tx_bytes	pod/network/sent_bytes_count	
	pod/volume/total_bytes	
	pod/volume/used_bytes	
	pod/volume/utilization	
		kube_node_status_condition
		kube_pod_status_phase
		kube_pod_status_ready
service_number_of_running_pods		