INSTITUTO DE GESTÃO E TECNOLOGIA
DA INFORMAÇÃO

Fundamentos em Arquitetura de Dados e Soluções em Nuvem

Neylson Crepalde

2022

Fundamentos em Big Data

Bootcamp: Arquiteto(a) de Big Data

Neylson Crepalde

Sumário

Capítulo 1. Fundamentos em Arquitetura de Dados

A computação em nuvem já é uma realidade presente na grande maioria dos projetos de dados das organizações contemporâneas. A pergunta óbvia que se apresenta ao se discutir uma arquitetura *cloud* based é: “por que levar a infraestrutura da empresa ou do projeto para a nuvem?”.

Neste capítulo vamos conhecer as principais vantagens de uma arquitetura *cloud based* e estudar *Well Architected Framework.*

Por que cloud?

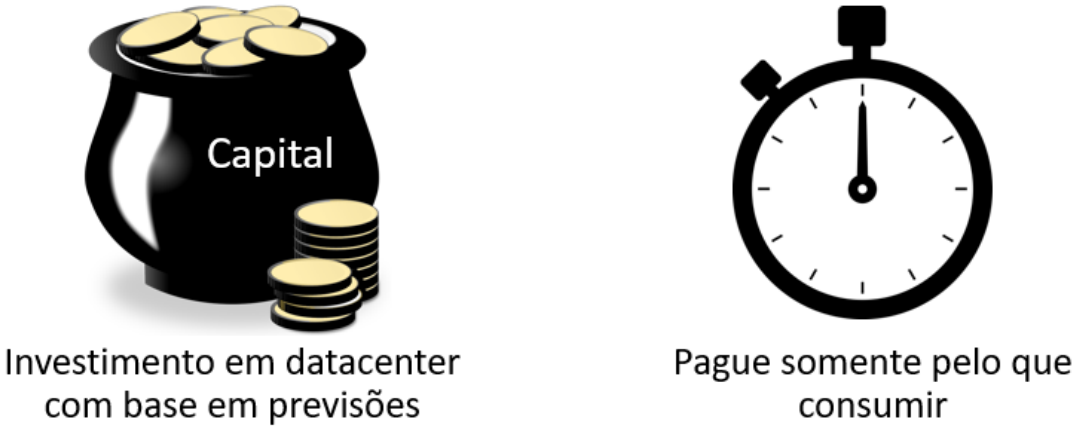
A primeira grande diferença entre uma arquitetura gerenciada localmente (*on premisses*) e uma arquitetura em nuvem é o modelo de custos. Numa estrutura *on premisses* são necessários grandes investimentos na construção de data centers, aquisição de máquinas e recursos computacionais adequados, além de pagamento de equipe responsável pela configuração, monitoramento e suporte de toda a infraestrutura. Os custos são calculados com base em previsões realizadas normalmente pela equipe de TI.

Já numa arquitetura em nuvem, o modelo de custo obedece à regra “pague pelo que utilizar”. O modelo é válido tanto para quantidades de dados armazenados quanto para o tamanho de recursos computacionais alocados e tráfego de dados (Fig. 1).

Em uma estrutura *on premisses*, com relação à utilização dos recursos de infra disponíveis, inevitavelmente cairemos em uma das duas situações a seguir: ao utilizarmos menos capacidade do que os servidores possuem, estamos desperdiçando capacidade e deixando de aproveitar um recurso no qual investimos; ao utilizarmos a

total capacidade do servidor, estaremos colocando nossa infra e nossas soluções em risco de falha com a sobrecarga.

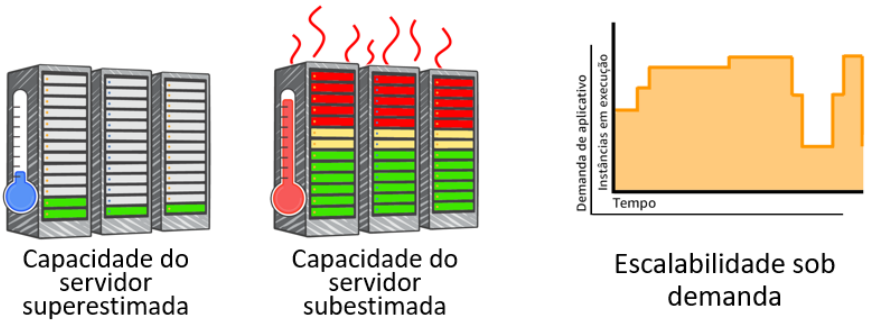
Figura 1 – Modelos de Custo



Fonte: Material Educacional AWS Academy

Já em estruturas em nuvem, a quantidade de recursos utilizados pode se adaptar à demanda de maneira dinâmica, o que chamamos de *elasticidade*. Dessa maneira, a utilização dos recursos alocados será sempre ótima tanto quanto à capacidade alocada quanto aos custos (Fig. 2).

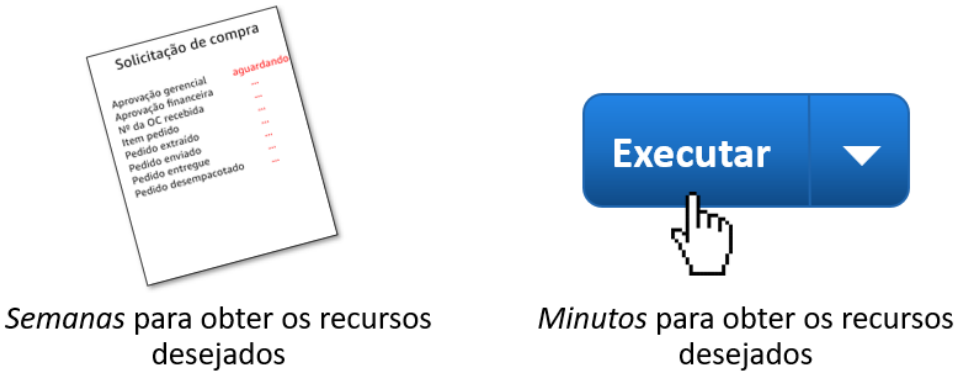
Figura 2 – Capacidade de recursos.



Fonte: Material Educacional AWS Academy

Quando é necessário a compra de novos recursos, os processos internos das organizações costumam levar algumas semanas ou meses. Esse tempo é crítico se houver alguma demanda urgente da área de negócio ou no caso de algum teste de hipótese momentâneo. Em contrapartida, a obtenção de recursos em nuvem é bastante rápida. Com alguns cliques e poucas configurações é possível obter os recursos desejados (Fig. 3).

Figura 3 – Velocidade e agilidade.

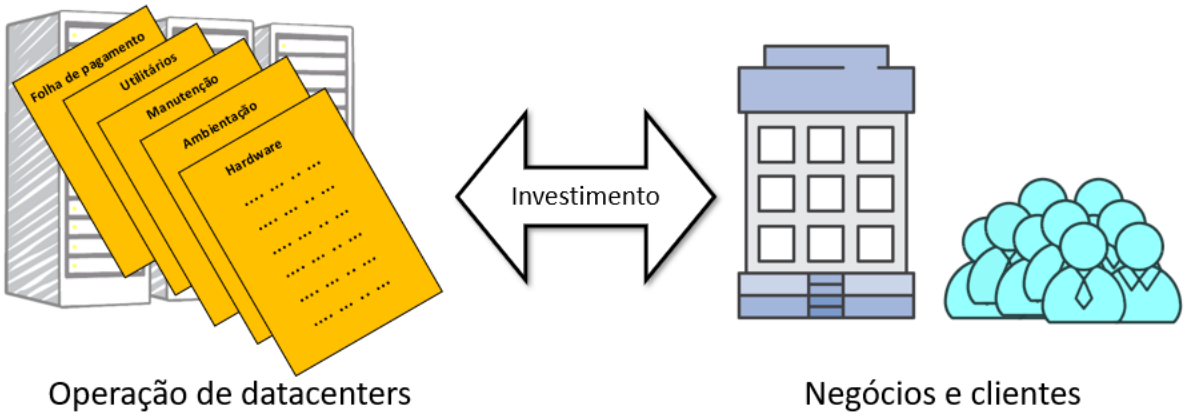


Fonte: Material Educacional AWS Academy

Trabalhar com data centers locais exige um grande investimento operacional de *setup*, monitoramento e manutenção da infraestrutura. Uma equipe grande de profissionais deve ser alocada para acompanhar as operações do data center e executar qualquer melhoria ou manutenção necessária.

No modelo de soluções em nuvem, o custo da operação de data centers é absorvido pelo provedor, de modo que a organização pode focar na parte central de seu negócio, na geração de valor comercial e na experiência/satisfação dos clientes (Fig. 4).

Figura 4 – Operação e valor para o negócio.



Fonte: Material Educacional AWS Academy

Os modelos de serviço em nuvem

Os três modelos básicos de serviços em nuvem são conhecidos como IaaS (*Infrastructure as a Service*), PaaS (*Platform as a Service*) e SaaS (*Software as a Service*) – cf. Fig. 5.

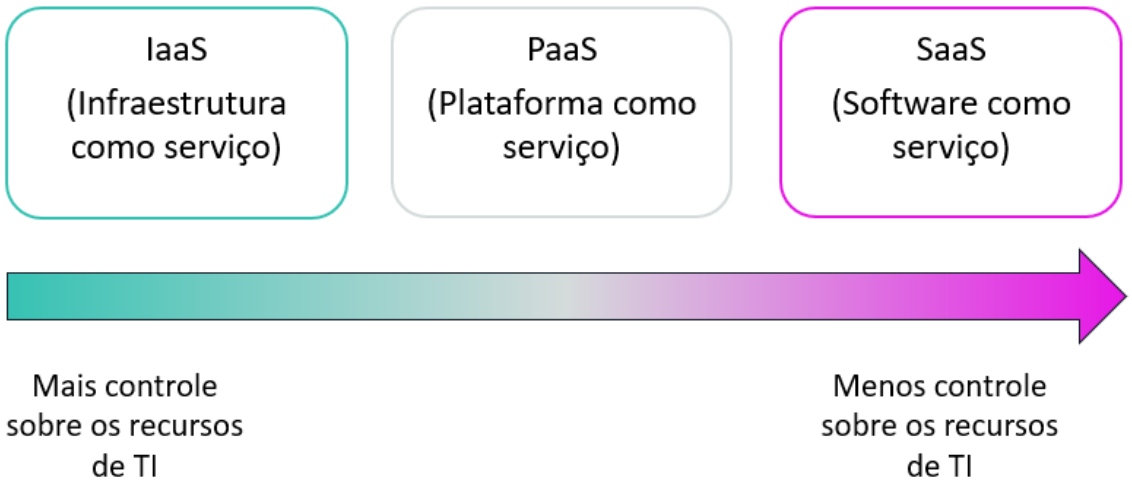
Nos serviços do tipo IaaS, apenas a infraestrutura é provida como serviço, cabendo ao usuário gerenciar a configuração de rede e o armazenamento, aspectos de

segurança e gerenciar o controle de acesso. Este é o caso das máquinas virtuais, e dos File Systems em nuvem.

No modelo PaaS, mais uma camada é absorvida pelo *provider*. O *provider* gerencia sistema operacional, a aplicação de *patches* de bancos de dados, configuração de firewall e a recuperação de desastres. O usuário, por sua vez, fica mais concentrado no desenvolvimento do código da aplicação ou no gerenciamento de dados. Este é o caso das funções *serverless* e dos serviços de bases de dados gerenciados.

No modelo SaaS, o software utilizado é hospedado de maneira centralizada e licenciado ao usuário num modelo de assinatura ou pagamento por utilização. O serviço normalmente é acessado pelo navegador ou por uma API ou aplicativo *mobile* e o usuário final não participa em nada do gerenciamento da infraestrutura que sustenta o serviço. Este é o caso de utilizar o famoso *Office 365* ou o Google Drive e o Gmail.

Figura 5 – Capacidade de recursos

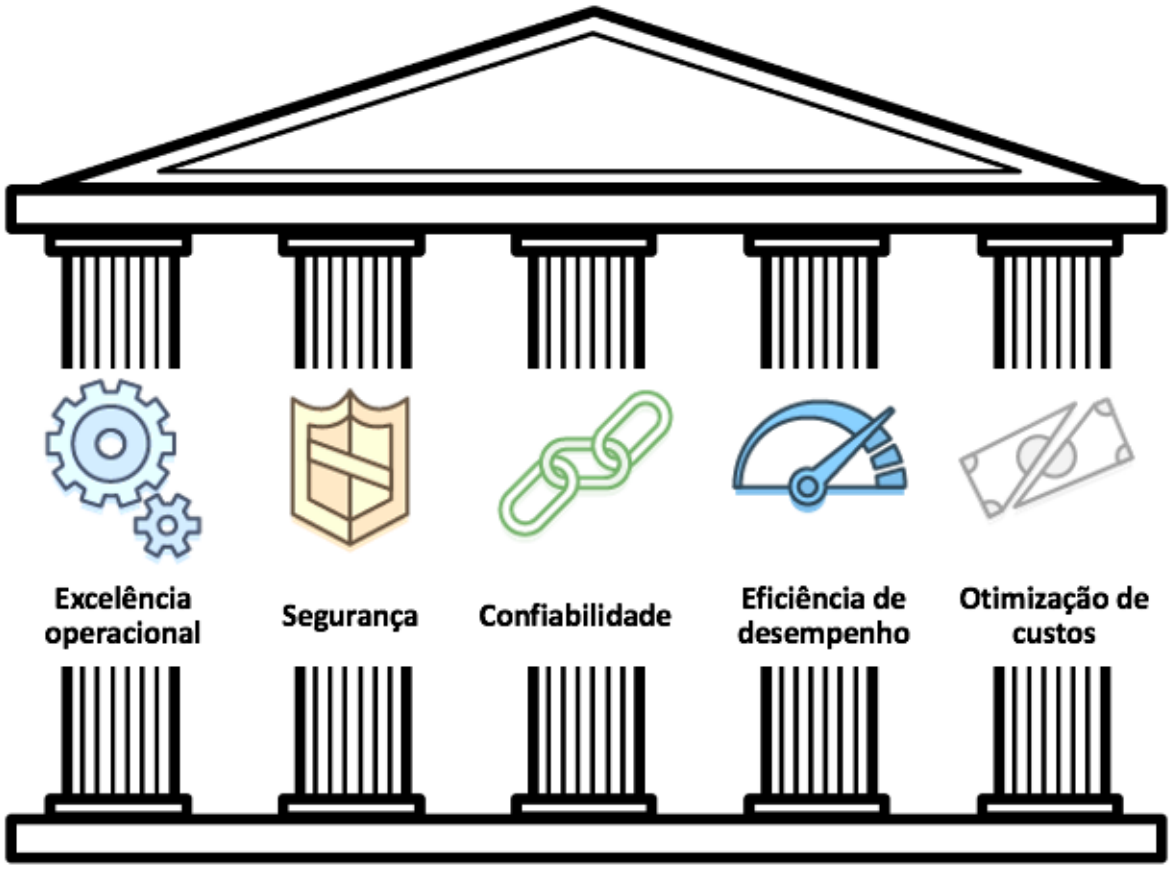


Fonte: Material Educacional AWS Academy.

Well Architected Framework

O *Well Architected Framework* é um conjunto de boas práticas e recomendações para o desenho de uma "boa" solução de arquitetura em nuvem (Fig. 6). Ele consiste em um guia para projetar arquiteturas que sejam seguras, resilientes, de alta performance e eficientes. O modelo foi proposto pela AWS mas pode ser adotado como um guia geral para o desenho de qualquer solução em nuvem.

Figura 6 – *Well Architected Framework.*



Fonte: Material Educacional AWS Academy.

Nesse modelo há 5 pilares a serem observados no desenho de soluções em nuvem, a saber, excelência operacional, segurança, confiabilidade, eficiência de desempenho e otimização de custos.

O pilar da excelência operacional possui foco em executar e monitorar sistemas para agregar valor de negócio e melhorar continuamente os processos e procedimentos de suporte. Seus principais tópicos são gerenciar e automatizar alterações, conseguir responder de maneira rápida e eficiente a eventos na infraestrutura e definir padrões para gerenciar com êxito as operações diárias.

A este pilar, podemos relacionar as práticas de infraestrutura e operações como código, produção de documentação, a realização de alterações pequenas, frequentes e reversíveis, o refinamento frequente de procedimentos operacionais e a previsão de eventos e falhas operacionais.

O pilar da segurança possui foco em proteger informações e sistemas ativos e, ao mesmo tempo, agregar valor para o negócio por meio de avaliações de risco e estratégias de mitigação. Seus principais tópicos são identificar e gerenciar os papeis e perfis atuantes no ambiente bem como suas respectivas permissões, o estabelecimento de controles para detectar eventos de segurança, a proteção de sistemas e serviços, bem como a confidencialidade e integridade dos dados.

A este pilar, podemos relacionar as práticas de implementação de uma base de controle de identidade e autenticação sólida, habilitar a rastreabilidade no ambiente, aplicar segurança em todas as camadas, automatizar as melhores práticas de segurança, proteger dados em trânsito e em repouso bem como preparar-se para eventos de segurança.

O pilar da confiabilidade possui foco em prevenir-se e recuperar-se rapidamente de falhas para atender às demandas do negócio e dos clientes. Seus

principais tópicos são configuração, requisitos entre projetos, planejamento de recuperação e tratamento de alterações. A este pilar podemos relacionar as práticas de testar procedimentos de recuperação, a implementação de recuperação automatizada de falhas, o escalonamento horizontal de recursos para aumentar a disponibilidade agregada do ambiente e o gerenciamento de alterações na automação.

O pilar da eficiência de desempenho possui foco em usar recursos de computação e TI de forma eficiente para atender aos requisitos do sistema e manter a eficiência à medida que as mudanças na demanda e as tecnologias evoluem. Seus principais tópicos são a correta seleção de tipos e tamanhos de recursos com base nos requisitos das cargas de trabalho, o monitoramento de todo o ambiente e o foco no desempenho, bem como a tomada de decisão embasada. A este pilar podemos relacionar as práticas de democratização de tecnologias avançadas, o uso preferencial de arquiteturas *serverless* e a frequente experimentação.

Por fim, o pilar da otimização de custos possui foco em executar sistemas para agregar valor de negócio pelo menor preço. Seus principais tópicos são a compreensão e o controle de onde os investimentos financeiros estão sendo realizados, a seleção do número e tamanho correto de recursos, a análise de gastos ao longo do tempo e a escalabilidade constante para atender à demanda empresarial sem gastos excessivos.

A este pilar podemos relacionar as práticas de adoção de um modelo de consumo junto ao provider de nuvem que otimize os custos, a medição da eficiência geral do ambiente em termos de custos, a eliminação de despesas operacionais com data centers, a análise e atribuição segmentada de despesas e a utilização de serviços gerenciados para a redução do custo total de propriedade.

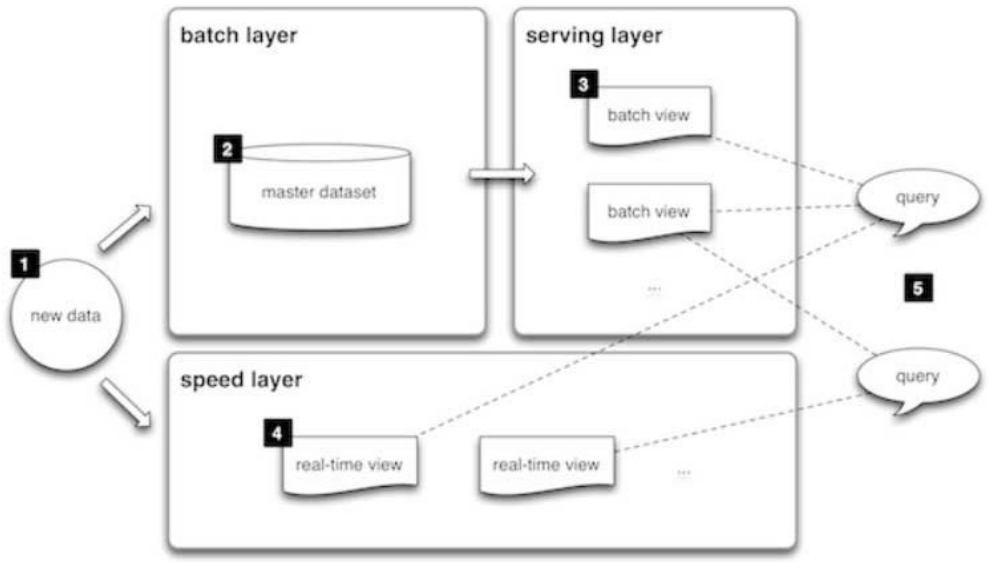
Capítulo 2. Desenhos de Arquitetura

Neste capítulo, vamos estudar alguns dos principais desenhos de arquitetura conhecidos no mercado, a saber, a arquitetura Lambda, arquitetura Kappa, arquitetura Unifield e a arquitetura Data Lakehouse.

Arquitetura Lambda

A arquitetura Lambda é conhecida por possuir duas camadas de processamento em paralelo, uma camada de processamento em lote (*batch layer*) e uma camada de processamento rápido ou quase em tempo real (*speed layer*). Além disso ela possui uma camada de serviço (*serving layer*) onde os dados serão de fato disponibilizados para consumo dos usuários (cf. Fig. 7).

Figura 7 – Arquitetura Lambda.



Fonte: https://databricks.com/glossary/lambda-architecture.

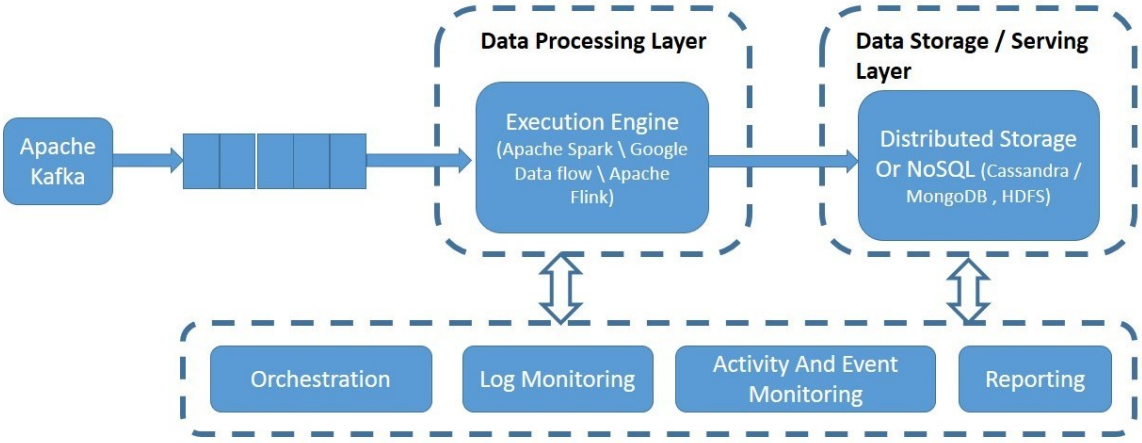
Esse modelo é bastante conhecido e a maioria das implementações de mercado hoje estão baseadas nela ou alguma de suas variantes. Ela possui a grande vantagem de

permitir, ao mesmo tempo, a análise de dados online e off-line (dados históricos e em tempo real) e é bastante indicada quando os requisitos de negócio possuem ambas as necessidades analíticas. Contudo, as duas *layers* de processamento simultâneo causam um certo excesso de armazenamento de dados e duplicações.

Arquitetura Kappa

A arquitetura Kappa se propõe a resolver o problema de duplicação de dados da Lambda a partir de uma única camada de processamento, que irá servir ambos os dados em *real time* e em *batch* (cf. Fig. 8). Os dados são ingeridos e armazenados numa estrutura de *data lake* através de um processo de *streaming* e todo o processamento analítico é realizado em uma camada posterior. Quando alguma análise histórica mais robusta precisa ser feita, os dados podem ser transferidos novamente para o sistema de filas do *streaming* para reprocessamento.

Figura 8 – Arquitetura Kappa



Siddharth Mittal

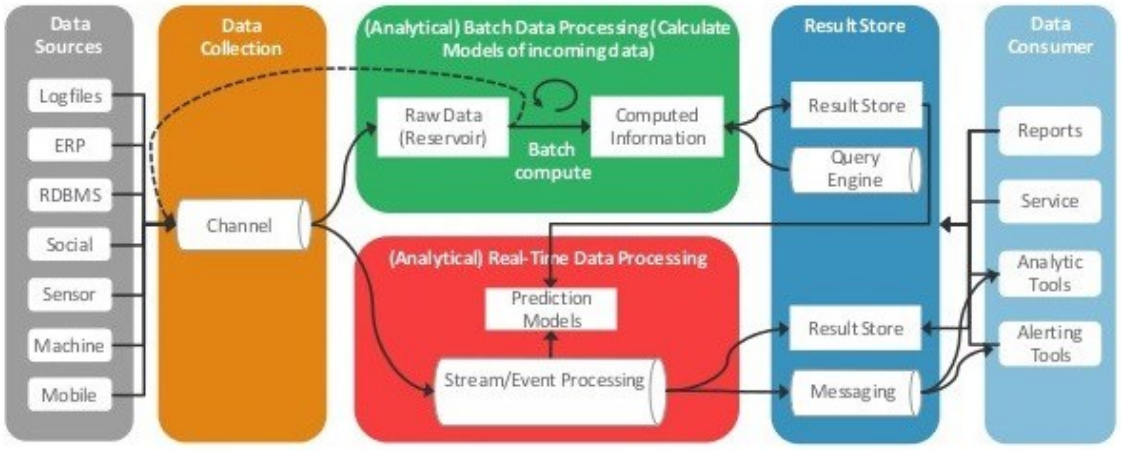
Fonte: https://www.linkedin.com/pulse/from-lambda-architecture-kappa-using-apache-beam-siddharth-mittal/.

A arquitetura Kappa resolve o problema de dados duplicados e ineficiência no armazenamento no *data lake* com a engenhosa ideia de reprocessar o dado histórico e, em sua concepção geral, é bem simples. Contudo, é uma arquitetura de difícil implementação, especialmente o reprocessamento de dados armazenados no lake pelo *streaming*.

Arquitetura Unifield

A arquitetura Unifield também é voltada para Big Data e se parece bastante com a arquitetura Lambda (cf. Fig. 9). A principal diferença é que, neste caso, é inserido uma combinação de processamento de dados e *Machine Learning.*

Figura 9 – Arquitetura Unifield.



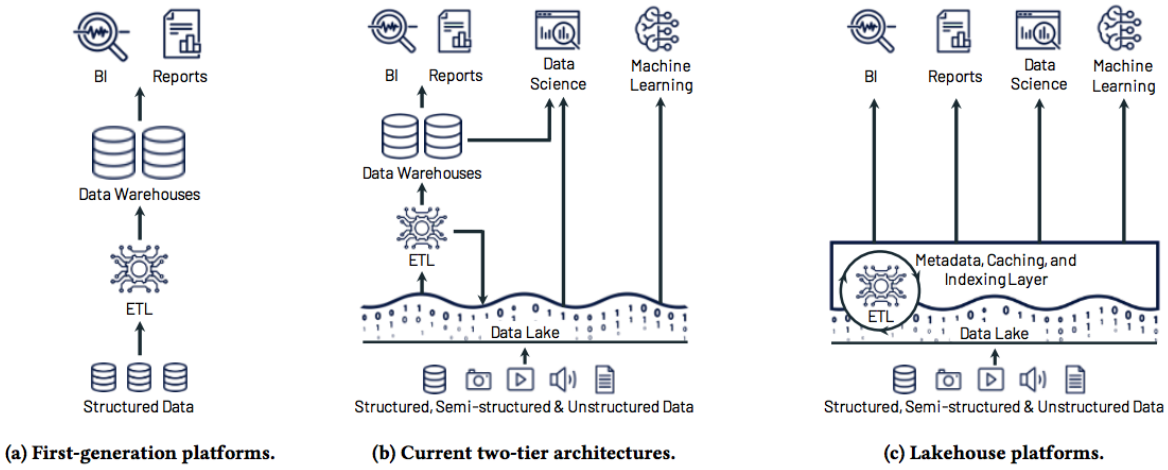
Fonte: https://www.yanboyang.com/bigdataarchitectures/.

Modelos de *Machine Learning* são treinados e adicionados à *speed layer* para predições em tempo real. Essencialmente a combinação de *data analytics* com *real time machine learning* pode ser bastante frutífera e gerar insights potencializados para os usuários, facilitando e empoderando ainda mais a tomada de decisão. Contudo, esta arquitetura possui um grau maior de complexidade em sua implementação.

Arquitetura Data Lakehouse

A arquitetura Lakehouse possui uma estrutura moderna que traz flexibilidade, otimização de custos e escalonamento para o data lake além de introduzir operações ACID (Atômicas, Consistentes, Isoladas e Duráveis), comuns em bancos de dados relacionais, nesta estrutura (cf. Fig. 10).

Figura 10 – Data Lakehouse.



Fonte: ARMBRUST et al, 2021.

Os projetos de Big Data normalmente utilizavam uma arquitetura em duas camadas baseada na Lambda, a saber, o *Data Lake* como camada de armazenamento e o *Data Warehouse* como camada de serviço. A princípio, os DWs eram a principal fonte de armazenamento de informação analítica para as organizações. Contudo, eles mostraram um alto custo para manusear dados semiestruturados ou não estruturados e com alta velocidade e volume.

Nesse contexto, surgiram os *Data Lakes* com a possibilidade de centralizar o armazenamento de dados não estruturados, de fontes variadas e com tamanhos enormes. O grande problema da estrutura de *Data lake* é que ela não permitia

transações nos registros, o que era comum aos DWs, e não asseguravam a qualidade e consistência dos dados. Daí, surgiu a combinação da estrutura em duas camadas urtilizando tanto o *Data Lake* quanto o DW. Essa combinação se mostrou eficiente por muitos anos, porém contém a dificuldade de manutenção e monitoramento da arquitetura como um todo.

O surgimento do *Data Lakehouse* permitiu a inclusão de operações ACID diretamente nos arquivos armazenados no *storage* do *lake* através de arquivos de metadados de formato aberto. Além de possibilitar esse tipo de transação, o *Data Lakehouse* também possibilita o *time travel* entre versões diferentes dos dados, uma vez que todos os arquivos permanecem armazenados no *storage*. Além disso, a arquitetura tem uma performance excepcional, assegura a qualidade e a consistência dos dados e possibilita a evolução de *schemas* com o passar do tempo. Além do mais, o formato aberto de seu mecanismo de controle de logs e metadados possibilita a compatibilidade com uma grande variedade de sistemas de *storage* e, por conseguinte, a fácil integração com diversos provedores de nuvem.

A partir de agora vamos revisar os principais serviços de bases de dados em nuvem.

Database as a Service

Todos os provedores de nuvem trabalham com o conceito de *Database as a Service*, ou seja, um serviço de base de dados gerenciado onde o usuário não se preocupa com a instalação, configuração, atualização ou qualquer outra questão de administração da infra sobre a qual a base de dados está implantada. Apenas faz seu uso através das configurações de conexão disponibilizadas. Esse modelo PaaS de serviço de nuvem permite que o usuário se concentre na administração dos dados em si e na elaboração da aplicação para geração de valor para o negócio.

Na AWS, os principais serviços de bancos relacionais disponíveis são:

- RDS (*Relational Database Service*) compatível com Oracle, Microsoft SQL Server, MySQL, PostgreSQL, MariaDB.
- Amazon Aurora – Engine de database própria da Amazon compatível com MySQL e PostgreSQL. Essa database funciona como cluster e pode ser utilizada tanto na sua forma convencional (ancorada em máquinas EC2 cujas configurações, quantidade e tamanho o usuário pode escolher) ou *serverless* (o usuário define apenas a quantidade de capacidade computacional que deseja e a database escala ou se desliga automaticamente).

Os serviços de Data Warehouse oferecidos pela AWS são:

- Amazon Redshift (Database relacional em cluster baseada em PostgreSQL);
- Amazon Redshift Spectrum (capacidade do Redshift de fazer a leitura de arquivos que estão no Data Lake. Não se confunde com as *engines* de data lake, pois para executar a interação com os arquivos do lake, existe a necessidade de provisionar um cluster Redshift).

No Google Cloud os principais serviços de bancos de dados relacionais são:

- Solução Bare Metal (consiste em uma migração *lift-and-shift* para databases Oracle);
- Cloud SQL (database relacional compatível com MySQL, PostgreSQL e Microsoft SQL Server);
- Cloud Spanner (serviço compatível com Oracle e com o Amazon DynamoDB. Apesar de o DynamoDB ser uma solução NoSQL, o Cloud

Spanner é o serviço indicado pela Google para realizar essa migraçãso de provedores).

A solução de Data Warehouse do Google é o famoso BigQuery. Trata-se de uma solução de base de dados relacional *serverless* com grande escalabilidade e poder computacional.

Na Microsoft Azure, os serviços de bancos relacionais disponíveis são:

- Bancos de Dados SQL do Azure;
- Instância Gerenciada de SQL do Azure (SQL Server);
- SQL Server em Máquinas Virtuais;
- Bancos de Dados do Azure para PostgreSQL;
- Bancos de Dados do Azure para MySQL;
- Bancos de Dados do Azure para MariaDB.

Os serviços de DW da Azure são:

- Azure SQL Data Warehouse;
- Azure Synapse Analytics.

Com relação aos bancos de dados NoSQL, os principais serviços oferecidos pela AWS são:

- DynamoDB (banco de dados chave-valor);
- DocumentDB (banco de dados de documentos compatível com MongoDB);
- Amazon Neptune (banco de dados em grafo);
- Amazon Elasticache (banco de dados em memória compatível com Memcached e Redis);

- Amazon ElasticSearch Service.

No Google Cloud, os principais serviços de bancos de dados NoSQL são:

- Cloud Bigtable (chave-valor);
- Firestore (banco de dados de documentos);
- Firebase Realtime Database (banco de dados de documentos);
- Memorystore (banco de dados em memória compatível com Redis e Memcached).

O Google Cloud possui também integração com outros bancos de dados através da sua rede de parceiros (MongoDB Atlas, Datastax, Redis Labs, Neo4j e ElasticSearch).

Na Microsoft Azure os principais serviços de bancos de dados NoSQL são:

- Azure Cosmos DB (compatível com os tipos chave-valor, banco de dados de documentos e banco de dados em grafo);
- Cache do Azure para Redis (banco de dados em memória);
- Elastic no Azure (Elastic Search).

A seguir, vamos estudar as peças de construção de uma arquitetura de Data Lake separadamente, focando nos serviços AWS disponíveis e suas características.

Armazenamento

Nos providers de nuvem, é comum utilizarmos soluções de armazenamento de objetos como infraestrutura para a construção de Data Lakes.

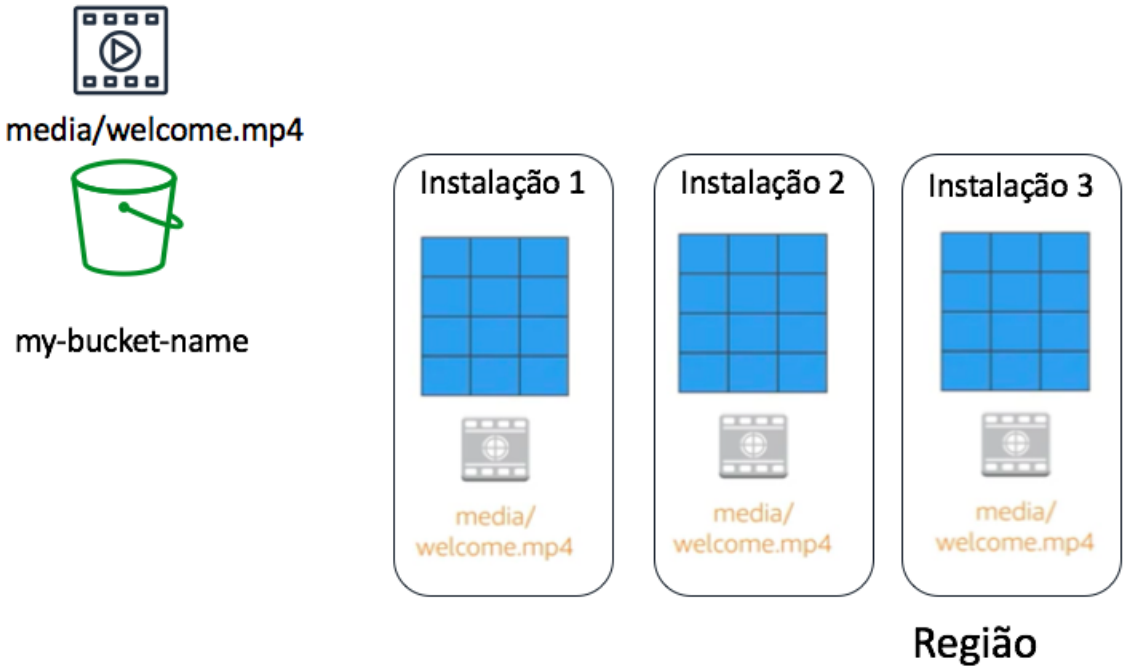
Na AWS, a principal solução de armazenamento de objetos é o Amazon S3 (Amazon *Simple Storage Service*). Este serviço é praticamente ilimitado quanto ao volume de dados que é possível armazenar. Há um limite de tamanho para objetos

únicos de 5 TB. O serviço foi projetado para oferecer uma durabilidade 11 noves (isto é, 99,999999999%) e permite acesso granular a objetos e buckets de armazenamento.

O Amazon S3 faz replicação automática de todos os objetos nele depositados em todas as Zonas de Disponibilidade (datacenters físicos) de uma região (Fig. 11).

Isto permite uma alta disponibilidade e durabilidade de todos os objetos armazenados.

Figura 11 – Replicação de Dados.



Fonte: Material Educacional AWS Academy.

O Amazon S3 possui um modelo de custo *pay as you go* ou pague pelo que consumir. O custo é calculado por GBs armazenados por mês. Além disso, o tráfego de dados para fora da AWS ou para outras regiões é pago bem como solicitações PUT, COPY, POST, LIST e. GET. Não entram no cálculo de custos o tráfego para dentro do S3 e

tráfego para fora do S3 quando o destino é o serviço Amazon Cloud Front ou instâncias EC2 que estejam na mesma região.

A seguir, vamos estudar modelos de ingestão e os serviços AWS para realizar a tarefa.

Ingestão de Dados

A ingestão de dados no Data Lake pode acontecer das seguintes formas:

- Em batch, uma única vez (processos únicos de ingestão *full* de tabelas ou fontes de dados completas);
- Em batch, por substituição (*overwrite*) programada (similar ao caso anterior, porém com repetição periódica e sobrescrita dos dados depositados no lake);
- Em batch incremental (ingestão em lotes, porém incremental);
- *Near real time* incremental (ingestão por eventos ou micro batches de eventos).

As fontes de dados nesses casos normalmente são:

- Tabelas em bases relacionais ou NoSQL;
- Sinais ou eventos vindos de sensores e equipamentos IoT;
- APIs;
- Arquivos em *file system* externo.

Para realizar a ingestão em batch, podemos utilizar as seguintes ferramentas:

- Spark;
- Python (com processos desenvolvidos de maneira customizada);

- Apache Nifi (apesar de ser uma solução de ETL, é bastante utilizado para ingestão de dados).

Para ingestão em *real time*, a principal ferramenta utilizada no mercado é o Apache Kafka. Trata-se de uma ferramenta extremamente robusta, escalável e com poder de processamento impressionante.

Existem também alguns serviços de nuvem já preparados para realizar atividades de ingestão. Para ingestão de dados que estão em bases de dados, a AWS possui o DMS (*Database Migration Services*) que é capaz de realizar todos os tipos de ingestão mencionados acima, desde que a fonte seja uma base de dados relacional.

Para streaming de eventos há alguns concorrentes do Kafka nos provedores de nuvem:

- O Amazon Kinesis na AWS;
- O PubSub no Google Cloud;
- O Azure Event Hub na Azure.

A partir de agora, vamos falar de ferramentas de processamento de Big Data.

Processamento de Big Data

O processamento de Big Data pode se dar em dois principais contextos, a saber, processamento em lotes (batch) ou processamento em tempo real (*real time*). Para realizar essa tarefa há tanto soluções open source quanto soluções gerenciadas pelos provedores de nuvem.

As duas principais ferramentas de processamento utilizadas hoje no mercado são:

- Apache Spark (capaz de realizar processamento em batch e em tempo real utilizando o módulo *Spark Structured Streaming*);
- ksqlDB (módulo de processamento de Big Data em *real time* compatível com Kafka).

Obviamente existem outras ferarmentas disponíveis no mercado, contudo, as apresentadas acima são as mais utilizadas. Na AWS temos algumas soluções gerenciadas para realizar a tarefa. Na AWS:

- Amazon Elastic Map Reduce – EMR (trata-se de um cluster Hadoop gerenciado pela AWS. Podemos utilizar Hadoop, Spark, Hive e diversas outras ferramentas compatíveis com o ecossistema);
- Amazon Glue (Possui o módulo Jobs que permite executar Jobs Spark de maneira *serverless*. É necessário somente depositar o código Spark - com algumas customizações para o ambiente Glue – e iniciar as execuções. A AWS cuida do gerenciamento de toda a infra por trás a qual não aparece para o usuário. É necessário definir a quantidade de DPUs – Data Processing Units – a serem utilizadas);
- Kinesis Data Analytics (solução para processamento de dados em streaming).

Na Microsoft Azure:

- HD Insight (processamento batch e real time);
- Azure Stream Analytics (processamento de dados em streaming).

No Google Cloud:

- Dataproc;
- PubSub;

- BigQuery.

É interessante mencionar ainda a plataforma Databricks. A Databricks é a empresa mantenedora da tecnologia Delta Lake, umas das mais utilizadas hoje no mercado para implementação de arquiteturas Data Lakehouse. Além disso, possui um ambiente próprio e bastante amigável para execução de Spark. Sua plataforma é compatível com os principais provedores de nuvem.

A seguir, vamos estudar as *engines* de consultas em Data Lake.

Engines de Data Lake

As *engines* de consultas a Data Lake surgiram como uma alternativa às estruturas de Data Warehouse que não permitiam a melhor otimização de custos nem a fácil integração com dados semiestruturados. Quando gerenciados pelos provedores de nuvem, seu modelo de custo é baseado no volume de dados varridos pela consulta.

Na AWS, o serviço disponibilizado é o Amazon Athena. O Athena é uma *engine* de consultas SQL distribuídas baseada no Presto. Trata-se de um serviço *serverless* e seu funcionamento depende da construção de *schemas* para os dados que estão no lake utilizando o serviço Glue Data Catalog. O Glue possui uma ferramenta bastante interessante, o Glue Crawler, que perpassa as pastas do Data Lake, entende os *schemas* dos dados e exporta metadados para a construção de tabelas no Athena.

A Microsoft Azure possui o Azure Data Lake Analytics e o Google Cloud não possui uma solução de consultas SQL. O serviço mais próximo seria o BigQuery que não se define de tal forma, mas antes como um Data Warehouse *serverless*.

Existem ainda várias soluções open source para utilização na infraestrutura de nuvem. Trabalhar com as soluções open source exige que o usuário cuide do deploy,

monitoramento e manutenção das soluções (modelo IaaS). Contudo, elas são igualmente ou até mais robustas do que as soluções gerenciadas. Elas são:

- Presto;
- Trino (antigo Presto SQL);
- Apache Drill;
- Dremio;
- e vários outros.

No próximo capítulo vamos abordar Infraestrutura como código, uma das principais ferramentas para o(a) engenheiro(a) de dados.

Capítulo 3. IaC – Infraestrutura como Código

De acordo com Morris,

> Infraestrutura como código é uma abordagem para automação da infraestrutura baseada em práticas do desenvolvimento de software. Ela enfatiza rotinas consistentes e repetitivas para provisionar e modificar sistemas e suas configurações. Você faz modificações no código e usa automação para testar e aplicar essas mudanças em seus sistemas (MORRIS, 2020, p. 4, tradução livre).

Ainda segundo Morris (2020), a prática de Infraestrutura como código (IaC) pode ativar os seguintes benefícios:

- Uso de infraestrutura como ativo para a entrega de valor ágil;
- Redução de risco e esforço relacionados a mudanças no ambiente;
- Possibilita o uso do recurso que preciso, na hora em que preciso;
- Ferramentas comuns para todas as etapas do processo (desenvolvimento, operações e outros);
- Sistemas confiáveis, seguros e otimizados com relação ao custo;
- Visibilidade de governança, segurança e controles de *compliance;*
- Rápida resolução de problemas e resposta a eventos.

É muito comum associar IaC com DevOps e há casos dos termos sendo tratados quase como sinônimos. De fato, a prática de IaC é mandatória quando adotamos os princípios DevOps ou DataOps, mas os conceitos não se confundem. IaC está relacionado a automação e versionamento de infraestrutura. Já DevOps e DataOps são princípios norteadores da construção de processos para desenvolvimento e elaboração de soluções de dados.

Para mais informações sobre o Manifesto DataOps, seus valores e princípios, acesse https://dataopsmanifesto.org/pt-pt/.

Ferramentas de IaC

Existem diversas ferramentas de IaC no mercado, tanto open source quanto gerenciadas por provedores de nuvem. Alguns exemplos de ferramentas gerenciadas:

- AWS CloudFormation.
- Azure Resource Manager.
- Google Cloud Dpeloyment Manager.

Entre as opções open source, podemos citar:

- Terraform;
- Pulumi;
- Ansible;
- Chef Infra;
- Puppet;
- e várias outras.

Nos exemplos práticos deste módulo, vamos focar nas soluções utilizando o Terraform (https://www.terraform.io/). A ferramenta é mantida pela Hashicorp e o desenvolvimento de código se dá utilizando a linguagem declarativa HCL (*Hashicorp Configuration Language*). A linguagem possui compatibilidade com os principais provedores de nuvem do mercado. Abaixo, alguns exemplos de código para implantação de infraestrutura usando Terraform (cf. Fig. 12, 13 e 14).

Figura 12 – Criação de um bucket S3 com Terraform.

```
resource "aws_s3_bucket" "b" {
  bucket = "my-tf-test-bucket"
  acl    = "private"

  tags = {
    Name        = "My bucket"
    Environment = "Dev"
  }
}
```

Fonte:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket.

Figura 13 – Criação de um bucket GCS com Terraform.

```
resource "google_storage_bucket" "static-site" {
  name          = "image-store.com"
  location      = "EU"
  force_destroy = true

  uniform_bucket_level_access = true

  website {
    main_page_suffix = "index.html"
    not_found_page   = "404.html"
  }
  cors {
    origin          = ["http://image-store.com"]
    method          = ["GET", "HEAD", "PUT", "POST", "DELETE"]
    response_header = ["*"]
    max_age_seconds = 3600
  }
}
```

Fonte:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/storage_bucket.

Figura 14 – Criação de um blob storage na Azure com Terraform.

```
resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "West Europe"
}

resource "azurerm_storage_account" "example" {
  name                     = "examplestoracc"
  resource_group_name      = azurerm_resource_group.example.name
  location                 = azurerm_resource_group.example.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
}

resource "azurerm_storage_container" "example" {
  name                  = "content"
  storage_account_name  = azurerm_storage_account.example.name
  container_access_type = "private"
}

resource "azurerm_storage_blob" "example" {
  name                   = "my-awesome-content.zip"
  storage_account_name   = azurerm_storage_account.example.name
  storage_container_name = azurerm_storage_container.example.name
  type                   = "Block"
  source                 = "some-local-file.zip"
}
```

Fonte:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/storage_blob.

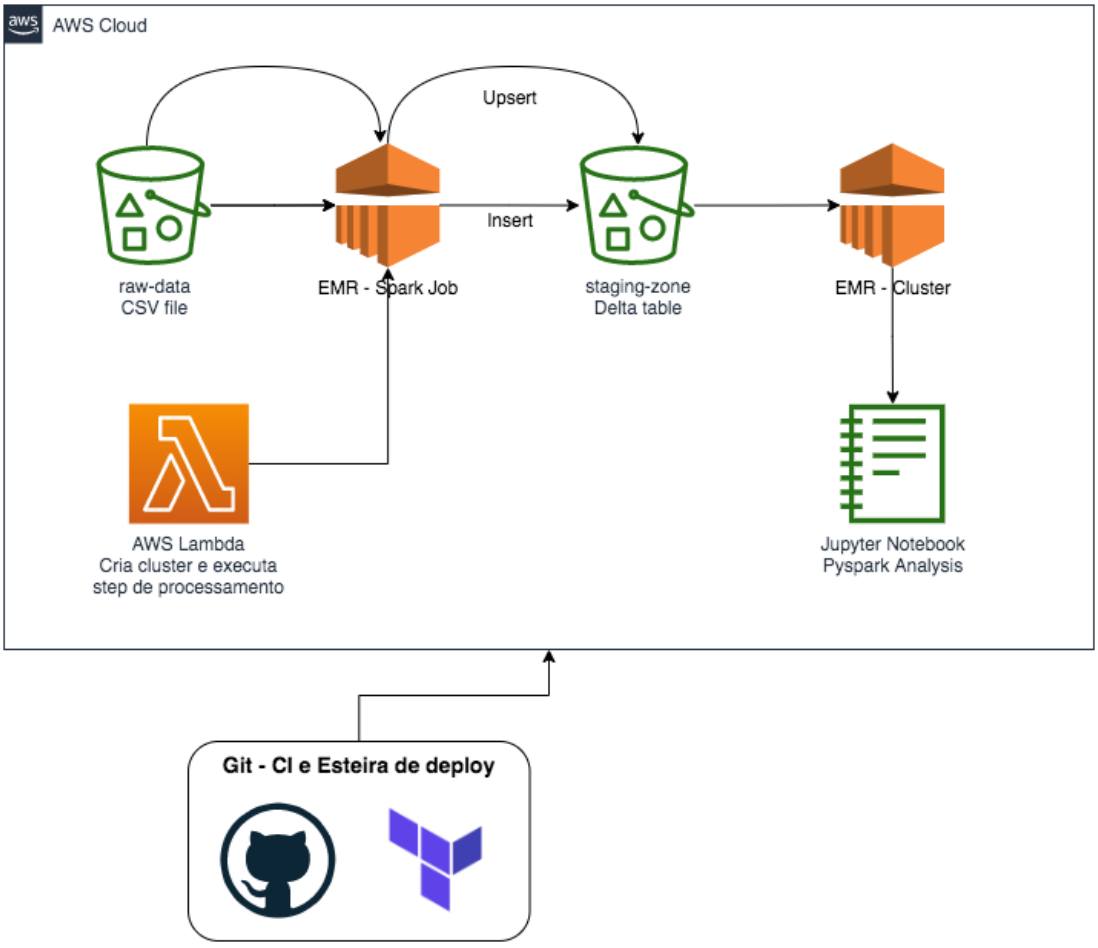
Capítulo 4. Use cases – Prática

Neste capítulo, apresentaremos as arquiteturas dos *use cases* práticos que serão implementados nos vídeos. Os códigos necessários para a implementação podem ser consultados em https://github.com/neylsoncrepalde/edc-mod1-exercise-igti.

Use case 1 – Data Lakehouse com Delta Lake e Amazon EMR

Neste *use case* vamos implementar a arquitetura abaixo:

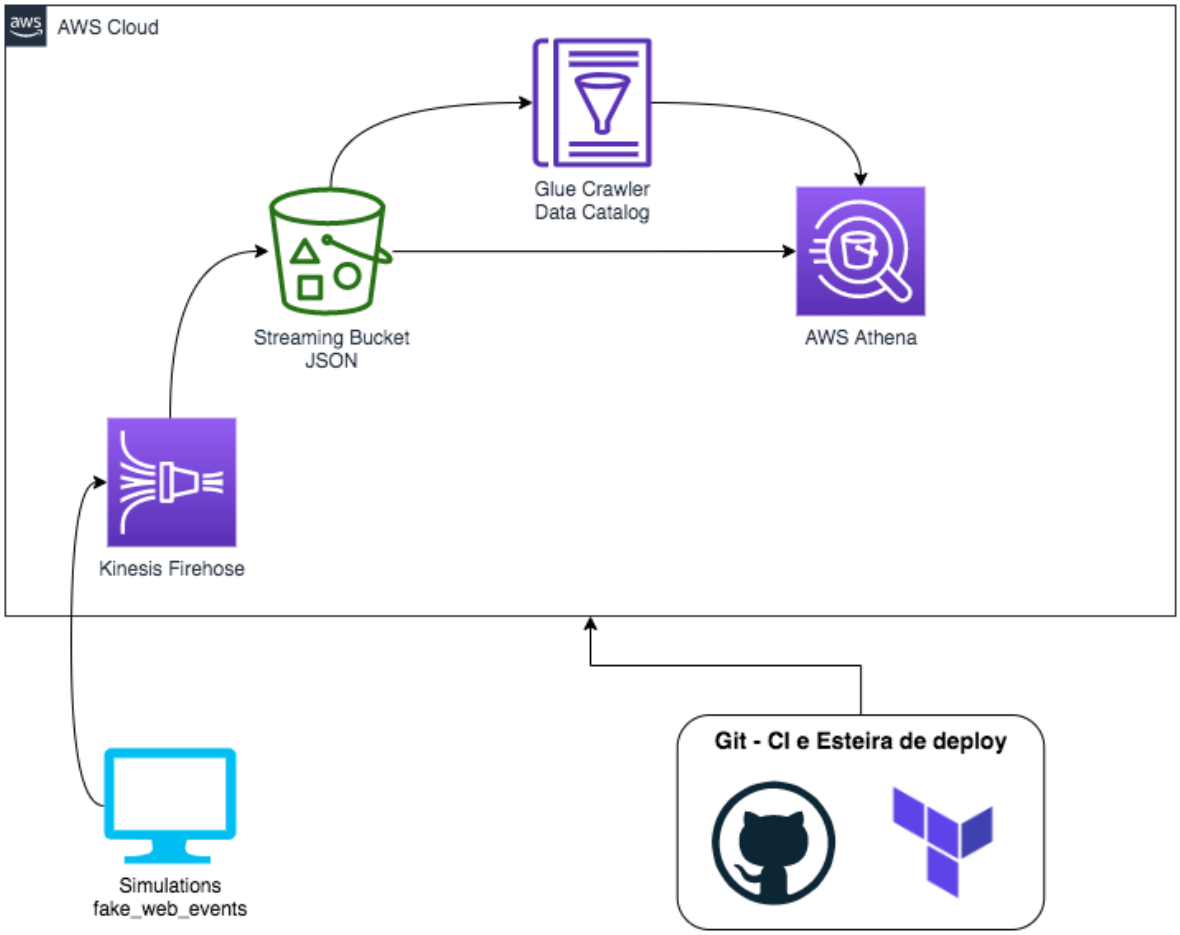
Figura 15 – Arquitetura do Use Case 1.



Use case 2 – Streaming de eventos com Kinesis

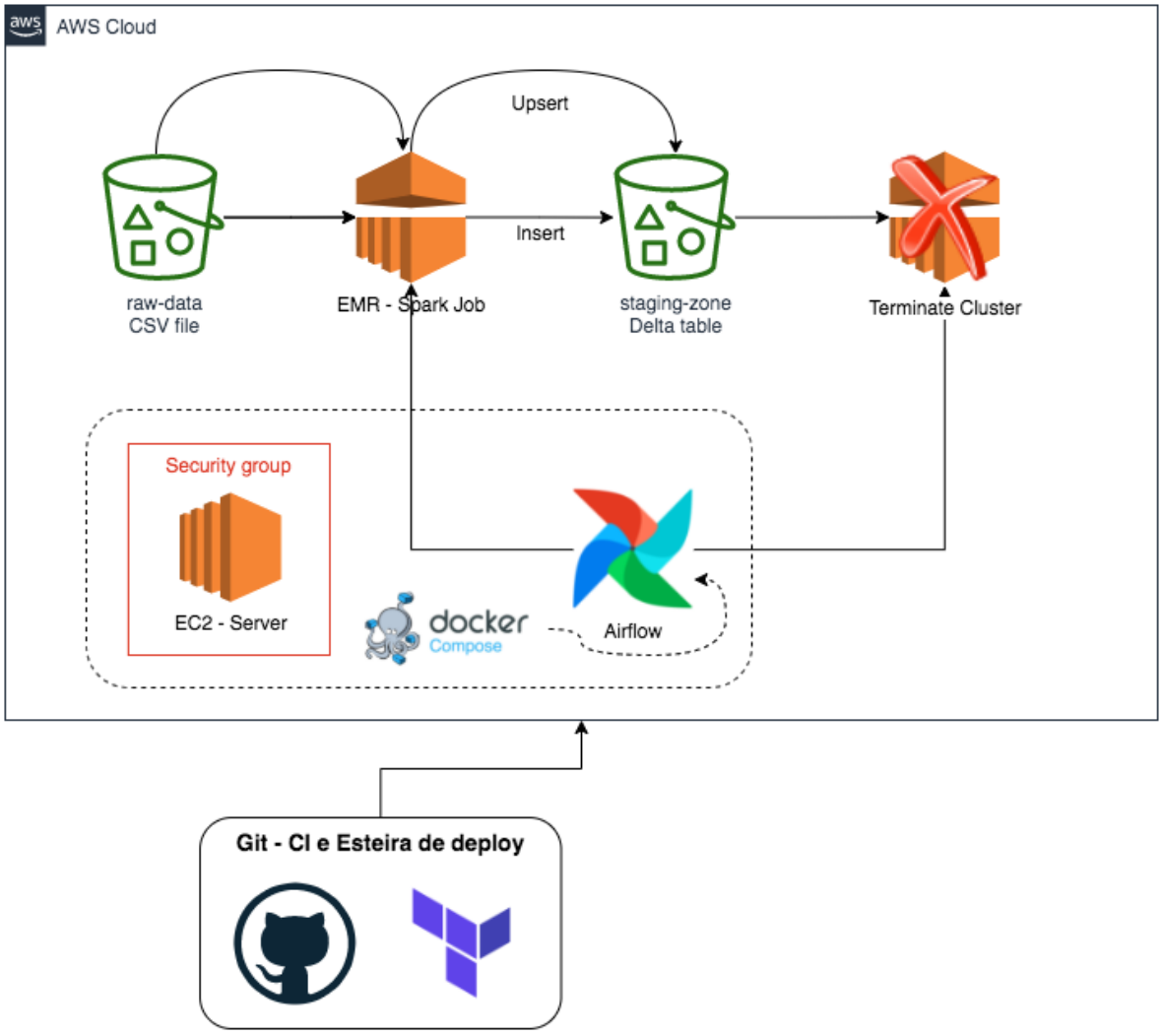
Neste *use case* vamos implementar a arquitetura abaixo:

Figura 16 – Arquitetura do Use Case 2.



Neste *use case* vamos implementar a arquitetura abaixo:

Figura 17 – Arquitetura do Use Case 3.



Referências

ARMBRUST, Michael et al. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. *In:* Annual Conference on innovative Data Systems Research - CIDR, 11., 2021, [*S.l.*]. *Online Proceedings* […]. Berlin: Computer Science Bibliographic, 2021. Disponível em: http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf. Acesso em: 01 ago. 2021.

MORRIS, Kief. *Infrastructure as Code*. Massachusetts: O'Reilly Media, 2020.