Deployment: production-ready Data Science

Prof. Donald Neumann, PhD.
Dezembro de 2025

Contatos:
DonaldN@insper.edu.br

Insper

Além de professor, sou consultor e lidero uma equipe de dados & AI

Quem sou?

- Chief Data & AI Officer @ Arvo
- PhD. Administração
  Supply Chain Management
  Universität Stuttgart
- M.Sc. Eng. de Sistemas
  Inteligência Artificial Aplicada
  PUCPR
- Administração de Empresas
  UFPR
- Eng. Computação
  PUCPR

Agenda

1. **Introdução ao deployment:**
   - O que é deployment?
   - Ciência de dados para deployment
2. **Desenvolvendo para deployment:**
   - Kedro como framework de desenvolvimento
   - Deployment de pipelines kedro utilizando Docker (agnóstico a nuvem)
3. **Deployment com serviços gerenciados:**
   - Utilizando serviços GCP para deployment

Material de aula: https://github.com/datadiversbr/deploy-workshop

Conversa Inicial

- Como você/seu time constrói suas soluções de ciência de dados?
  - Quais ferramentas são utilizadas?

- O que significa deploy para você/seu time?
  - Como a solução é colocada em produção?
  - Como é a o monitoramento e a manutenção?

- Quais são as dificuldades que você/seu time encontram?

Introdução ao Deployment

Deployment

AI Adoption in 2024: **74%** of Companies **Struggle to Achieve and Scale Value**

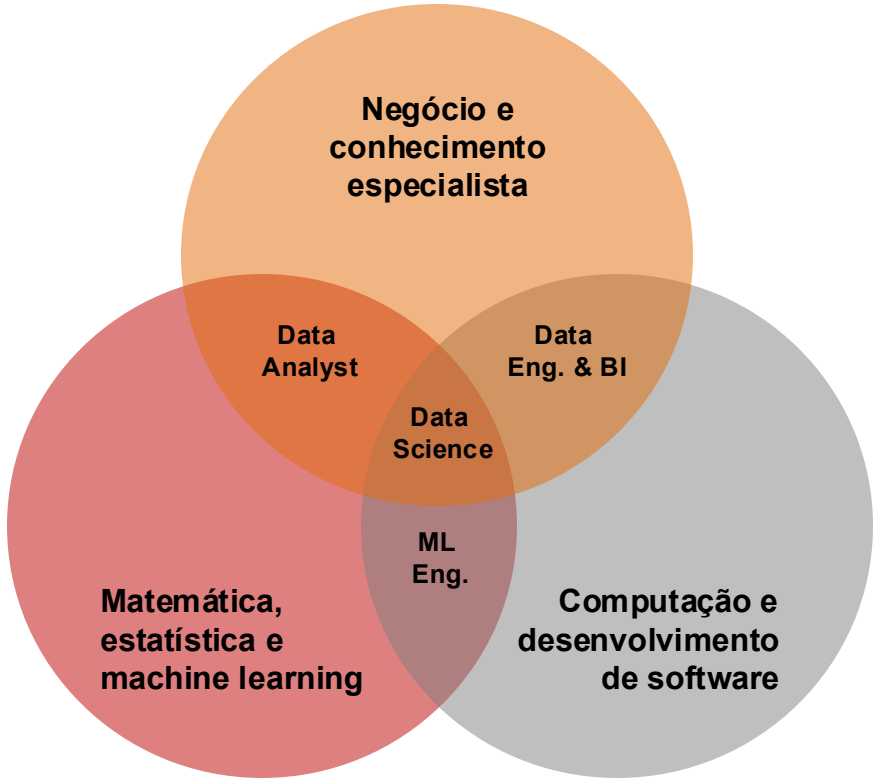
BCG, Press Release, 2024

We estimate that as much as **90 percent** of the failures in ML development come not from developing poor models but from **poor productization practices** and the challenges of integrating the model with production data and business applications, which keep the model from scaling and performing as intended.

Rewired, McKinsey & Company

O que é “deploy” no contexto de Ciência de Dados?

As diferentes perspectivas de uma solução digital



- Perspectiva de negócio:
  - Produto
- Perspectiva de computação:
  - Software
- Perspectiva matemática/estatística:
  - Solução analítica (incl. modelo)

Insper

A solução resolve um problema de negócio?

Perspectiva de Produto

- O produto é:
  - **Desejável** pelo usuário?
  - **Viável** do ponto de vista de negócio?
  - **Factível** do ponto de vista tecnológico?

- Macro-requisitos de produto:
  - **Desejável**: requisitos funcionais
    - Soluciona o problema correto
  - **Viável**: modelo de negócio
    - Existe um modelo de negócio para solução do problema
  - **Factível**: requisitos técnicos
    - Soluciona o problema com eficiência

A solução se traduz em um software eficiente?

Perspectiva Computacional

- O software possui eficiência:
  - **Funcional**: as features incluídas solucionam o problema do usuário com usabilidade adequada?
  - **Computacional**: o problema é solucionado em tempo adequado?
  - **De recursos**: o problema é solucionado com custos adequados?
  - **De disponibilidade**: o software é estável e está disponível quando necessário?
  - **De segurança**: segurança da informação é tratada *by design?*
  - **De manutenção**: manutenção é simples?

- Requisitos para o software:
  - UX
  - Funcionalidades
  - Arquitetura & Design Pattern
  - Disponibilidade
  - Escalabilidade
  - Segurança
  - Manutenibilidade
  - Custo
  - ...

A solução analítica atende aos requisitos de software e de produto?

Perspectiva Matemática/Estatística

- Os modelos possuem eficiência:
  - **Funcional**: os modelos solucionam o problema correto?
  - **De processo**: o usuário/a organização possuem um processo capaz de utilizar os modelos adequadamente?
  - **De desempenho**: o desempenho dos modelos é suficiente para gerar valor?
  - **Computacional**: os modelos são computacionalmente eficientes?
  - **De recursos**: os modelos possuem custos adequados?
  - **De disponibilidade**: os modelos não afetam disponibilidade?
  - **De segurança**: os modelos respeitam restrições de segurança?
  - **De manutenção**: o monitoramento e a manutenção dos modelos são simples?

- Requisitos para os componentes analíticos:
  - **Abordagem analítica**: define como a solução problema é decomposto em módulos interrelacionados
  - **Features do software**: como funcionalidades de treinamento, teste, backtest e inferência;
  - **Requisitos computacionais**: determinismo, tempo de execução, etc.
  - **Requisitos de desempenho**: como precisão, recall, AUC, MAPE, etc.
  - **Requisitos de segurança**: como anonimização, localização, viés e segurança de modelos
  - **Requisitos de disponibilidade, escalabilidade e custos**: ferramentas, design patterns, linguagens e tech stack;
  - **Requisitos de manutenibilidade**: como equipe necessária, processos, recursos de monitoramento, etc.

Deploy é muito mais do que um processo técnico de mudança do ambiente de um novo ativo

Deploy

- No seu core, "deployment" é o processo através do qual um ativo – uma linha de código, um modelo de machine learning, uma nova feature – é movido do ambiente de desenvolvimento (onde foi criado) para o ambiente de produção (onde será utilizado);

- É a ponte entre criação e geração de valor e envolve dimensões de software, analítica (de ciência de dados) e organizacionais;

- Dimensão de **software**:
  - Como o novo ativo é tecnicamente promovido para produção?

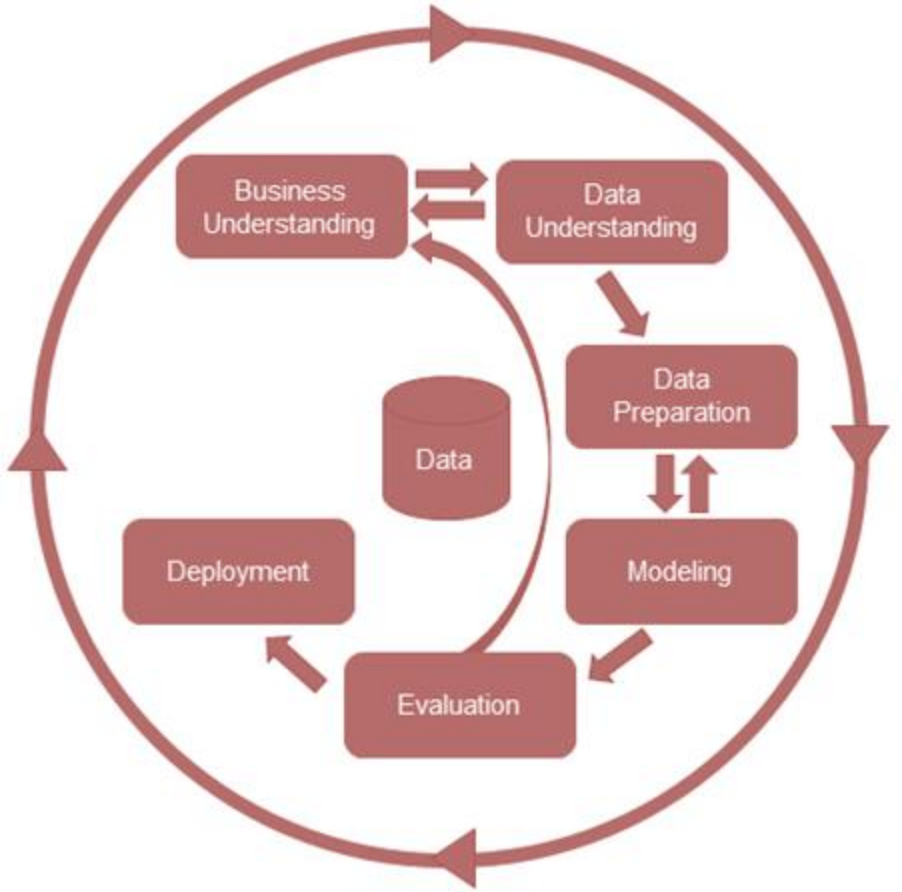
- Dimensão **analítica**:
  - Como o novo ativo é estabilizado em produção?

- Dimensão **organizacional**:
  - É necssário ajustar o processo com o deploy do novo ativo?
  - Como o novo ativo será monitorado, mantido, evoluído?

Desta forma, construir um modelo para deploy afeta o ciclo de ciência de dados, bem como as ferramentas e os métodos utilizados

Como deploy afeta o ciclo de ciência de dados?

- Requisitos para os componentes analíticos:
  - **Abordagem analítica**: define como a solução problema é decomposto em módulos interrelacionados
  - **Features do software**: como funcionalidades de treinamento, teste, backtest e inferência;
  - **Requisitos computacionais**: determinismo, reprodutibilidade, tempo de execução, etc.
  - **Requisitos de desempenho**: como precisão, recall, AUC, MAPE, etc.
  - **Requisitos de segurança**: como anonimização, localização, viés e segurança de modelos
  - **Requisitos de disponibilidade, escalabilidade e custos**: cobertura de testes, ferramentas, design patterns, linguagens e tech stack;
  - **Requisitos de manutenibilidade**: como equipe necessária, processos, recursos de monitoramento, etc.

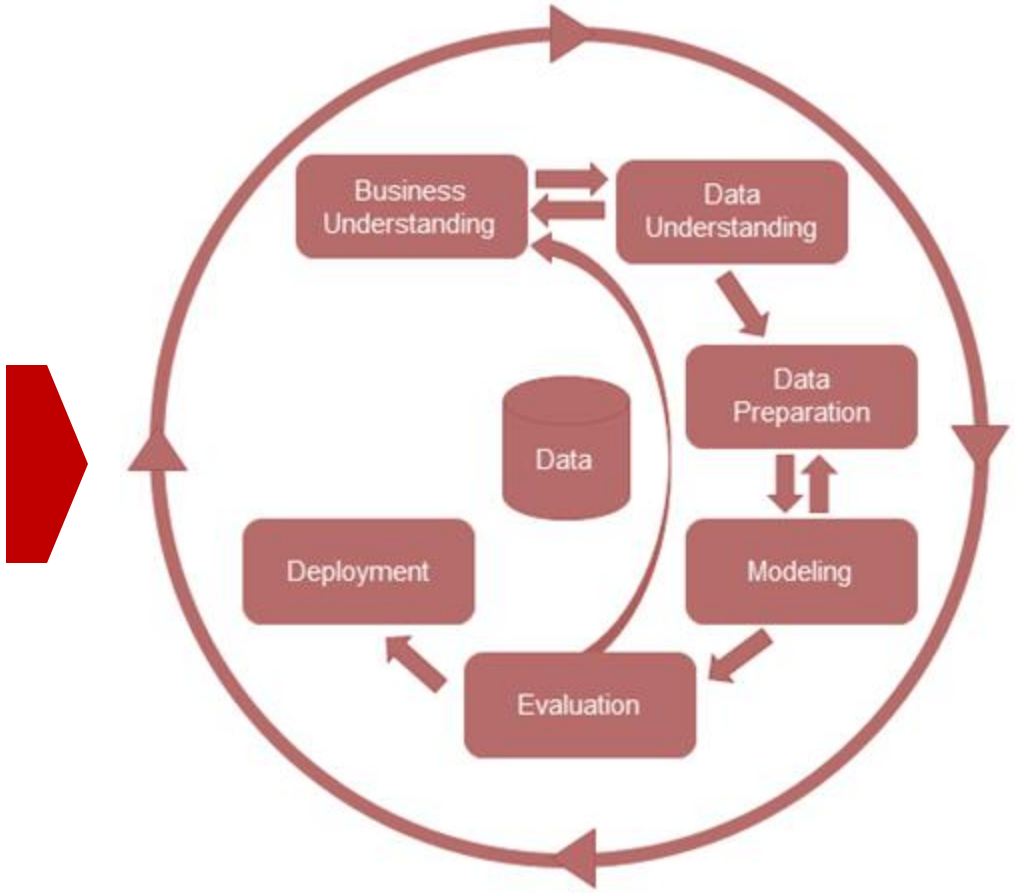


Insper

Isso implica absorver no ciclo de ciência de dados boas práticas de desenvolvimento com foco em 'software as a service'

Como deploy afeta o ciclo de ciência de dados?

1. **Base de Código:** base única de código com rastreamento, controle de versão e múltiplos deploys
2. **Dependências:** dependências declarativas e isoladas
3. **Configurações**: configurações armazenadas no próprio ambiente (não no software)
4. **Serviços de apoio:** serviços de apoio tratados como recursos conectados à aplicação
5. **Construa, teste, execute:** separação estrita dos builds com execução em estágios
6. **Stateless processes:** aplicação como um ou mais processos que não armazenam estado
7. **Port binding:** aplicação exposta como serviços com port-binding (API)
8. **Concorrência:** capacidade intrínseca de escalar horizontalmente no próprio app
9. **Descartabilidade**: robustez contra quebra eventual de processo
10. **Dev/prod semelhantes:** mantenha o desenvolvimento, teste, produção o mais semelhante possível
11. **Logs:** logs como fluxo de eventos
12. **Processos de admin:** tarefas de administração/gerenciamento de execução única em cada ambiente



Fonte: https://12factor.net/

Insper

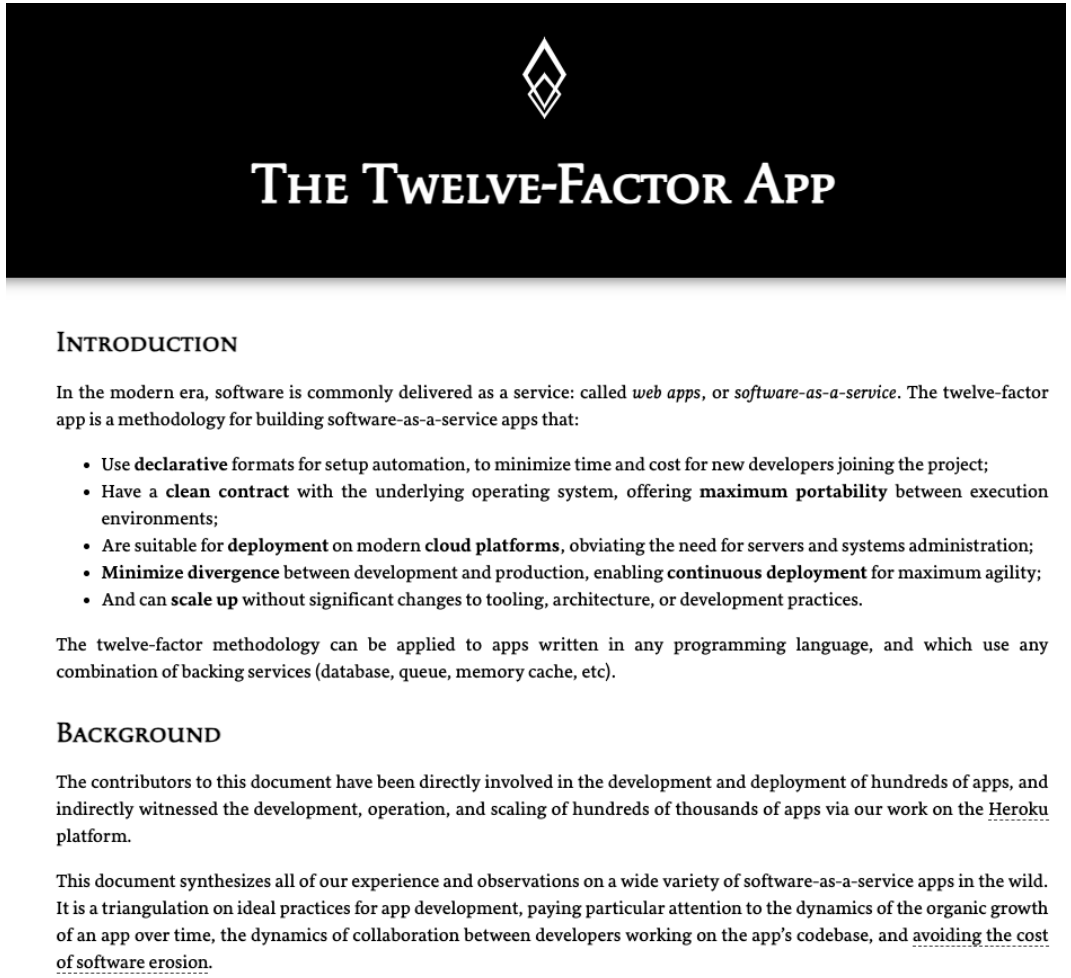
- Como essa perspectiva de deployment afeta como você constrói seus algoritmos de ciência de dados?
- O que isso significa para você, como cientista de dados/líder de ciência de dados?
- Como trazer as boas práticas de software para o ciclo de ciência de dados?
  - Como é feito na sua organização?

Production-ready Data Science

Princípios utilizando o Kedro

Existem diferentes metodologias e conjuntos de boas práticas de desenvolvimento de software que podem ser utilizadas

Exmplo: 12 Factor App



THE TWELVE-FACTOR APP

INTRODUCTION

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

BACKGROUND

The contributors to this document have been directly involved in the development and deployment of hundreds of apps, and indirectly witnessed the development, operation, and scaling of hundreds of thousands of apps via our work on the Heroku platform.

This document synthesizes all of our experience and observations on a wide variety of software-as-a-service apps in the wild. It is a triangulation on ideal practices for app development, paying particular attention to the dynamics of the organic growth of an app over time, the dynamics of collaboration between developers working on the app's codebase, and avoiding the cost of software erosion.

- Desenvolvido pela Salesforce
- Foco em software-as-a-service:
  - Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
  - Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
  - Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
  - **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
  - And can **scale up** without significant changes to tooling, architecture, or development practices.

Kedro: a toolbox for production-ready data science

O que é o kedro?

- **Kedro:**
  - Pacote python criado pela QuantumBlack, hoje com The Linux Foundation
  - Implementa diversas boas práticas de desenvolvimento paralelas ao 12-Factor App
  - Utiliza pipeline com design pattern com um DAG para controle de fluxo
  - Pensado para execução de workloads em batch
  - Encapsula algoritmos de engenharia e ciência de dados em software de qualidade com produtividade para a equipe desenvolvedora

- Canal no YouTube:
  - https://www.youtube.com/playlist?list=PL-JJgymPjK5LddZXbIzp9LWurkLGgB-nY

- **Vantagens:**
  - Fácil manutenção/configuração de pipelines
  - Rápida substituição de equipe técnica (DS/DE)
  - Reduz radicalmente/elimina:
    - Refatoração de código
    - Necessidade de eng. de SW e MLE no time
    - Documentação manual
  - Cloud-agnostic

- **Desvantagens:**
  - Design pattern pré-definido
    - Não adequado, por exemplo, para agentes
  - Foco no processamento em batch
  - Para workloads pequenos pode aumentar o overhead de processamento

https://docs.kedro.org/en/stable/

Desenvolver código produtivo com Kedro requer etapas claras

Production-ready data science

1. Planejar o pipeline
2. Construir o pipeline
3. Construir os nós
4. Preencher os catálogo
5. Preencher as configurações
6. Testar
7. Empacotar ou conteinerizar
8. Mover para infraestrutura de produção

Iris: classificando tipos de flores

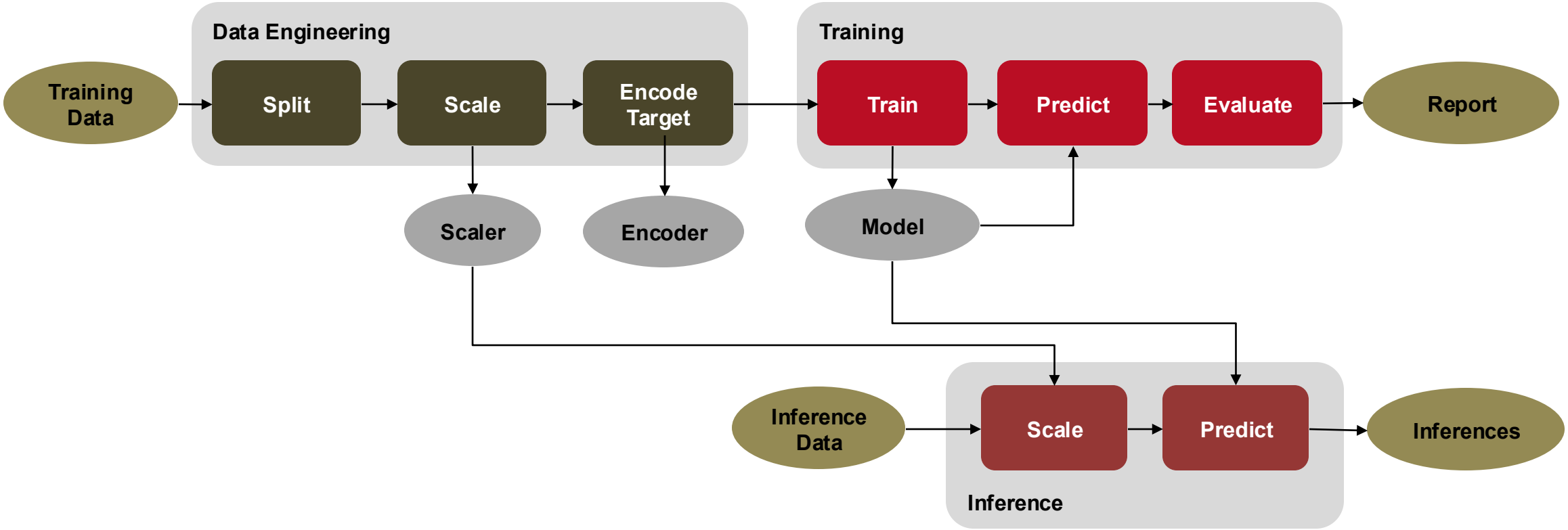
Exemplo

- Dataset:
  - https://www.kaggle.com/datasets/uciml/iris/data

- Notebook de exemplo:
  - https://www.kaggle.com/code/atifkhan12/iris-dataset-prediction-on-iris-dataset

A abordagem analítica para um problema simplificado de classificação envolve três pipelines

Planejamento do Pipeline



Insper

Uma base de código com versionamento e múltiplos deploys

Base de Código

- **Estrutura de projeto:**
  - Estrutura de projeto padrão, compatíveis com git (ex.: .gitignore);
- **Controle de versão:**
  - Commit de código (src), configurações (conf) e testes (tests), por padrão sem dados e chaves;
- **Múltiplos deploys:**
  - Mesmo commit pode ser utilizado para desenvolvimento, qualidade e produção;
- **Consistência**:
  - Mesmo código de dev é utilizado em produção;

Dependências isoladas e declaradas explicitamente

Dependências

- **Requirements.txt & pyproject.toml:**
  - Gerenciados com uv & pip

- **Isolamento de ambiente:**
  - Através de environments e/ou imagens docker;

- **Reprodutibilidade:**
  - Utilizando uv sync ou pip install –r requirements;

Configurações armazenadas no próprio ambiente e não no software

Configurações

- **Conf folder:**
  - Separa estritamente configurações locais (local), ignoradas pelo git, de configurações em produção (base);

- **Suporte a diferentes environments:**
  - Diferentes environments podem ser utilizados ao executar o pipeline com kedro run –env=<env>

- **Credenciais:**
  - Credenciais (credentials.yml) é ignorada pelo git por padrão;

Serviços de apoio, especialmente fontes de dados, são conectados como recursos declarativamente

Servicos de Apoio

- **Catálogo de Dados:**
  - Separa definição de formato e fonte de dados do código com configuração declarativa e explícita:
    - O código não sabe se o dado está sendo lido de um arquivo CSV ou de um Blob Storage.
  - Separação de níveis:
    - Abstrai do código como o dado é carregado/salvo;
  - Acoplamento fraco:
    - Facilmente modificável ao alterar o recurso no catálogo, não no código do pipeline;
  - Versionamento:
    - Pode-se ativar versionamento automático dos datasets, facilitando a reprodutibilidade

Separação estrita entre etapas de construção (build), disponibilização (release) e execução (run)

Build, release, run

- **Build:**
  - kedro package para criação de um .whl distribuível;
  - kedro docker para criação de uma imagem docker para distribuição;
  - kedro docker & kedro boot para exposição como API (FastAPI);

- **Release:**
  - O artefado construído é automaticamente pareado com o environment destino;

- **Run:**
  - Execução dos diferentes pipelines com kedro run

Aplicação como um ou mais processos que não armazenam estado entre rodadas

Stateless Processes

- **Nós sem estado:**
  - Cada nó (função) em um pipeline é uma função python pura;
- **Entradas e saídas:**
  - Dados são passados via argumentos e valores de retorno gerenciados pelo kedro runner;
- **Sem estado global:**
  - Execução dos pipelines não utiliza artefatos persistidos em memória entre rodadas;
- **Consequência:**
  - Execução dos nós pode ser automaticamente paralelizada quando não há dependência explícita no DAG com ParallelRunner;

A aplicação é auto-contida e exposta como API

Port Binding & Concorrência

- **Port binding:**
  - kedro boot permite exposição como API utilizando FastAPI

- **Concorrência:**
  - É possível escalar horizontalmente de diferentes maneiras;
  - ParallelRunner: roda nós e pipelines independentes em múltiplas CPUs de forma nativa pelo DAG;
  - Execução distribuída: através do deploy para Airflow, Kubeflow, etc. para execução em cluster;

Robustez contra quebra eventual de processo

Descartabilidade

- **Rodadas atômicas:**
  - Pipelines compostos de nós granulares com fechamento funcional;
- **Checkpoints:**
  - Datasets intermediários são persistidos via catálogo;
  - Se o pipeline quebra, pode-se resumir de onde parou (checkpoint);
- **Startup rápido:**
  - Dados persistidos são utilizados para execução dos nós restantes;
- **Sem processos zumbi:**
  - Kedro runner garante limpeza dos processos pendentes em caso de quebra;

Isolamento da camada de código da camada de infraestrutura, configuração e suporte aos logs

Paridade dev/prd, logs e processos de administração

- **Paridade dev/prd:**
  - Mesmo código, mesma estrutura de catálogo, apenas diferentes parâmetros de configuração (conf), origens e destinos de dados (catalog);

- **Logging:**
  - Utilização do padrão logging com configuração explícita em logging.yml;

- **Processos de administração:**
  - Configuração dos ambientes é única na instalação e não precisa ser refeita a cada deploy;

Além da implementação dos 12 fatores, o Kedro reduz significativamente a complexidade de manutenção

Testes, padronização, documentação e manutenção

- **/tests:**
  - Permite a construção de testes automáticos utilizando pytest
- **Padronização:**
  - Múltiplos cientistas e engenheiros, um único design pattern, maduro e padronizado;
- **Kedro viz:**
  - Permite inspeção do DAG com visualização do código reduzindo radicalmente a necessidade de documentação extra código;
- **Documentação (Docs com Sphinx):**
  - Possibilidade de adicionar uma pasta e gerar a documentação técnica
- **Debugging:**
  - Facilidade de integração com IDEs como o VSCode para debug do código

Discussão e Exercício

- Como o Kedro ou os princípios do Kedro podem ajudar você e seu time?

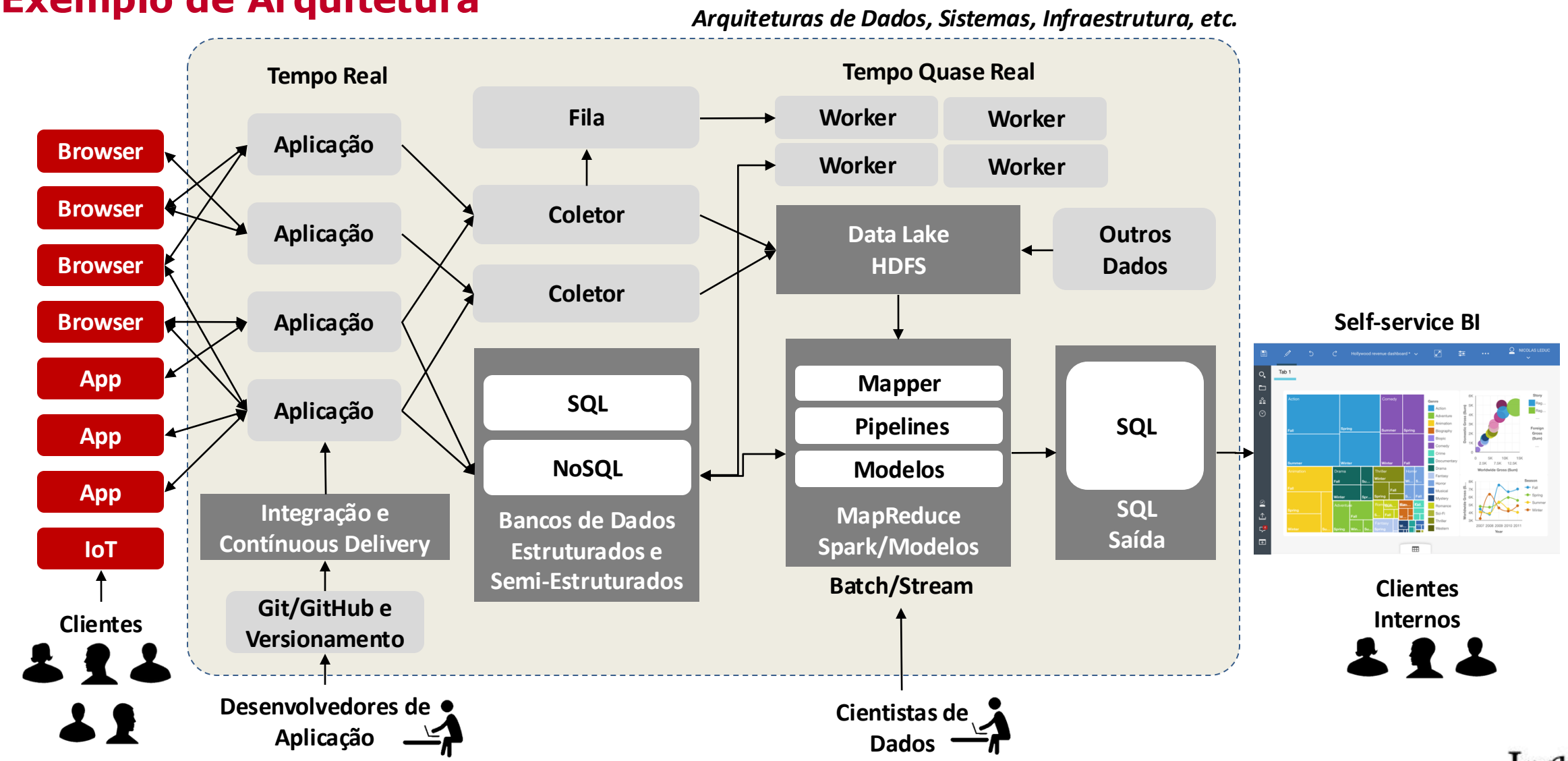
- Exercício:
  - Analise o notebook 00_diabetes_classifier.ipynb
    - https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset
  - Defina a abordagem analítica e os pipelines necessários para implementação
  - Implemente os pipelines de treino e inferência
  - Reflita como implementar otimização de hiperparâmetros, backtest e feature importances no pipeline. Como ficaria a abordagem analítica?
  - (opcional) Faça o deploy do pipeline em um container

Cloud-based Deployment:

Utilizando serviços gerenciados no GCP

Uma solução de tecnologia combina múltiplos módulos com funcionalidades específicas e que são similares entre diferentes aplicações

Exemplo de Arquitetura



Insper

It’s all about using somenone else’s computer!

Definição

Computação em nuvem é o fornecimento de serviços de computação—incluindo servidores, armazenamento, bancos de dados, rede, software, analytics e inteligência—pela Internet ("a nuvem") para oferecer inovação mais rápida, recursos flexíveis e economias de escala.

Características Essenciais

On-demand Self-service

Provisionamento automático de recursos

Broad Network Access

Acesso via rede por dispositivos diversos

Resource Pooling

Recursos compartilhados dinamicamente

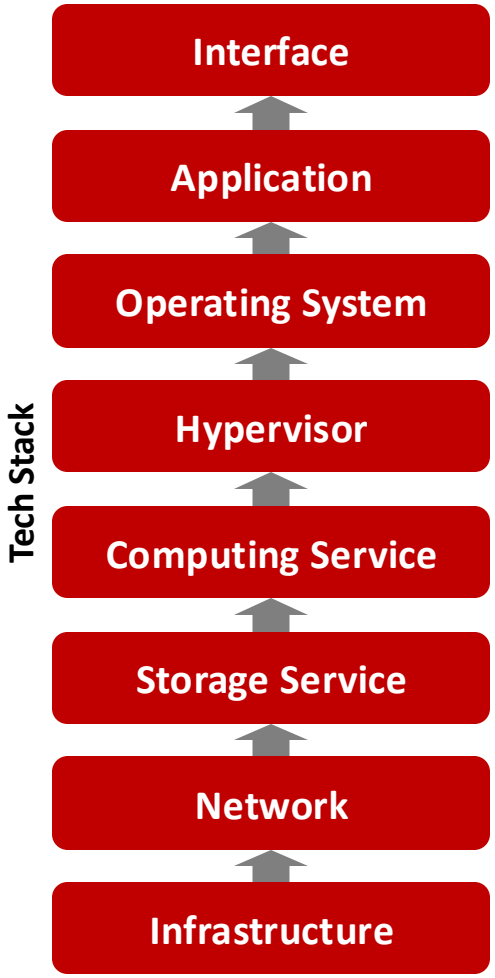
Rapid Elasticity

Escalabilidade automática

Measured Service

Pay-per-use e monitoramento

Existem diferentes modelos de serviço, dependendo do nível de abstração



Software as a Service (SaaS)

Platform as a Service (PaaS)

Infrastructure as a Service (IaaS)

IaaS - Infrastructure as a Service

- Máquinas virtuais, storage, redes
- Controle total sobre o SO
- Exemplos: EC2, Compute Engine
- Ideal para: migração lift-and-shift

PaaS - Platform as a Service

- Ambiente de desenvolvimento gerenciado
- Runtime, middleware, ferramentas
- Exemplos: App Engine, Cloud Functions
- Ideal para: desenvolvimento de aplicações

SaaS - Software as a Service

- Aplicações prontas para uso
- Acesso via browser/API
- Exemplos: BigQuery, AutoML
- Ideal para: soluções específicas

Insper

Muitos serviços preenchem, de formas diferentes, as mesmas necessidades

Exemplos de Nuvens

Category	Azure	AWS	GCP
Compute	Azure Virtual Machines (VMs)	EC2 (Elastic Compute Cloud)	Compute Engine
	Azure App Service	AWS Elastic Beanstalk	App Engine
	Azure Kubernetes Service (AKS)	Amazon EKS	Google Kubernetes Engine (GKE)
	Azure Functions	AWS Lambda	Cloud Functions
Storage (Object & Block)	Azure Blob Storage	Amazon S3	Cloud Storage
	Azure Disk Storage	Amazon EBS	Persistent Disks
	Azure Files (SMB/NFS)	Amazon EFS	Filestore
Databases	Azure SQL Database	Amazon RDS (incl. Aurora)	Cloud SQL
	Cosmos DB (multi-model NoSQL)	Amazon DynamoDB	Firestore / Datastore
	Azure Database for PostgreSQL/MySQL	Amazon RDS for PostgreSQL/MySQL	Cloud SQL for PostgreSQL/MySQL
	Azure Synapse Analytics	Amazon Redshift	BigQuery
Networking	Azure Virtual Network	Amazon VPC	Virtual Private Cloud (VPC)
	Azure Load Balancer / App Gateway	Elastic Load Balancing	Cloud Load Balancing
	Azure ExpressRoute	AWS Direct Connect	Cloud Interconnect
AI & ML	Azure Machine Learning	Amazon SageMaker	Vertex AI
	Azure OpenAI Service	Bedrock / Amazon Titan	Gemini / Generative AI Studio
	Cognitive Services (Vision, NLP, etc.)	AWS AI Services (Rekognition, Polly)	Cloud Vision, Speech-to-Text, etc.
Analytics & Big Data	Azure Data Factory	AWS Glue	Cloud Data Fusion
	Azure Databricks	Amazon EMR	Dataproc
	Azure Stream Analytics	Amazon Kinesis	Dataflow
DevOps & CI/CD	Azure DevOps / GitHub Actions	AWS CodePipeline, CodeBuild	Cloud Build, Cloud Deploy
Security & Identity	Azure Active Directory (AAD)	AWS IAM & Cognito	IAM & Identity Platform
	Azure Key Vault	AWS KMS	Cloud KMS
	Microsoft Defender for Cloud	AWS Security Hub	Security Command Center
Serverless	Azure Functions	AWS Lambda	Cloud Functions
	Azure Logic Apps	AWS Step Functions	Workflows
Monitoring & Logging	Azure Monitor, Log Analytics	Amazon CloudWatch, CloudTrail	Cloud Monitoring, Cloud Logging
IoT	Azure IoT Hub	AWS IoT Core	IoT Core
Hybrid & Edge	Azure Arc	AWS Outposts	Anthos
Marketplace & APIs	Azure Marketplace	AWS Marketplace	Google Cloud Marketplace

- Conceitos similares entre nuvens;
- Contudo:
  - Migração entre nuvens não é direta;
  - Escolha do CSP (Cloud Service Provider) é estratégica;
  - Ambientes multi-cloud tem sido cada vez mais frequentes;
- Atenção:
  - Custos;
  - Localização;
  - Restrições legais;

Principais Serviços do GCP

BigQuery

- Serverless data warehouse
- SQL padrão, escala automaticamente
- Integração nativa com ML

Vertex AI

- Plataforma unificada de ML
- Training, deployment, monitoring
- AutoML e custom models

Dataflow

- Apache Beam serverless
- Batch e stream processing
- Auto-scaling

Cloud Storage

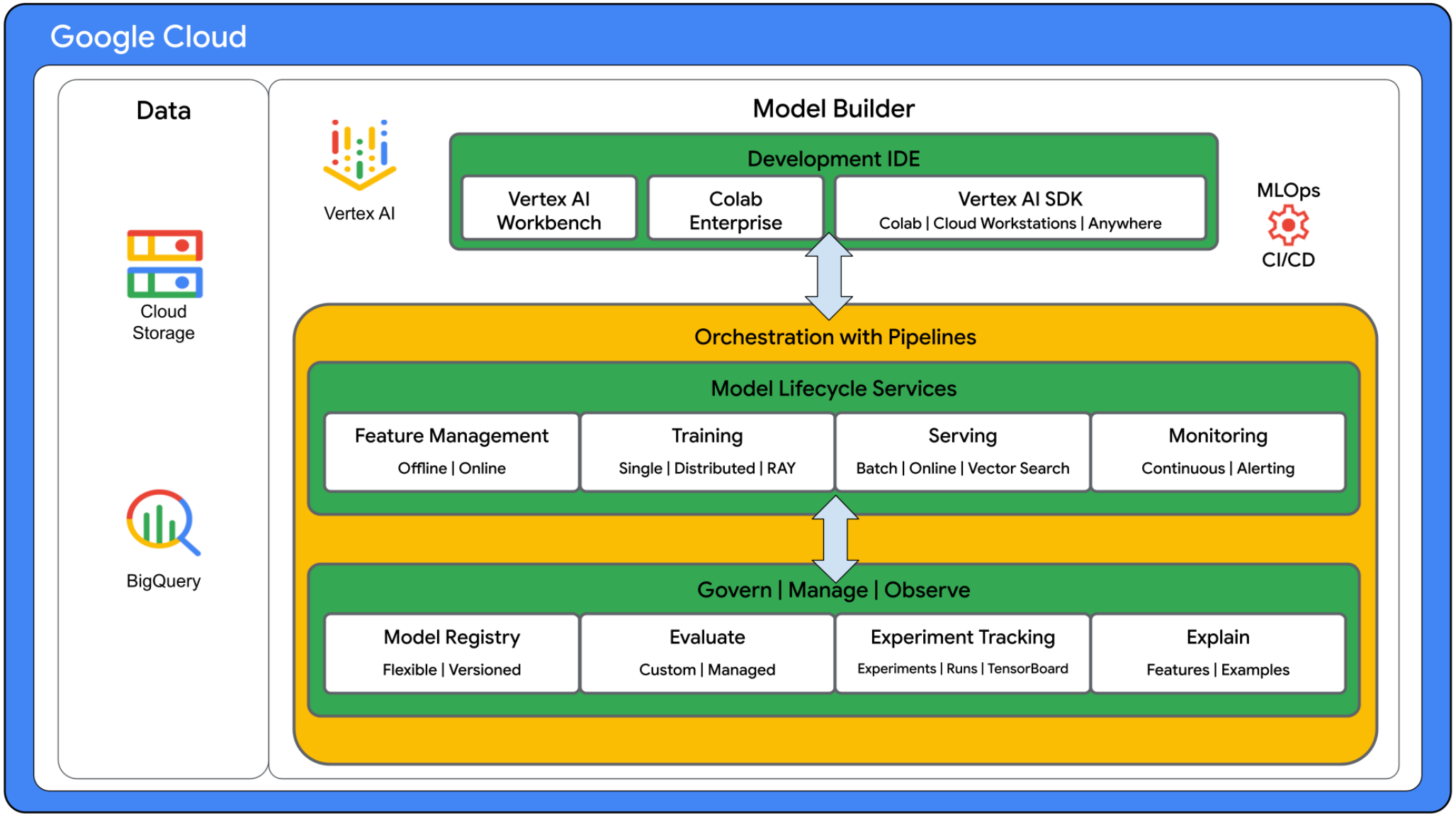
- Object storage global
- Classes de armazenamento
- Integração com todos serviços

Dataproc

- Hadoop/Spark gerenciado
- Clusters efêmeros
- Preemptible instances

Cloud Composer

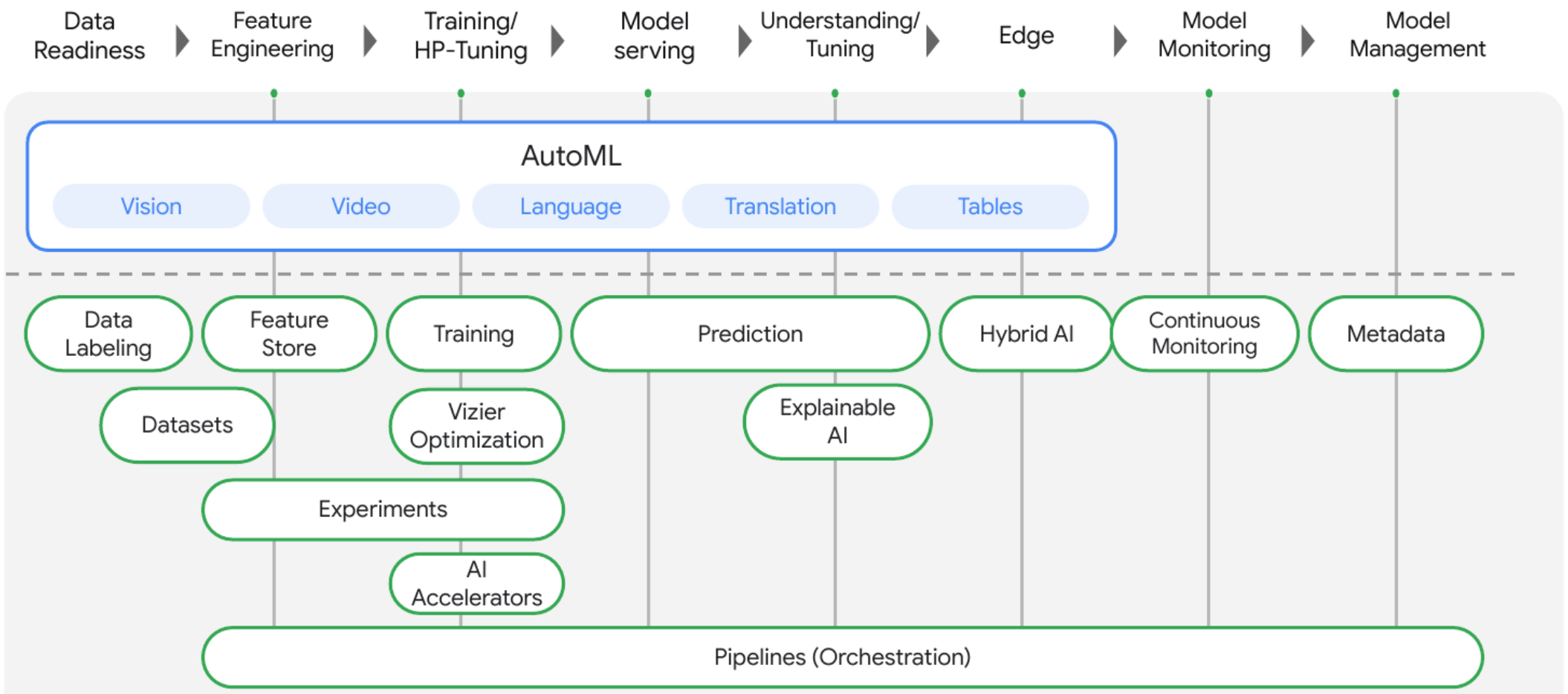
- Apache Airflow gerenciado
- Orquestração de workflows
- Integração com GCP


Google Cloud
Data
Cloud Storage
BigQuery
Model Builder
Vertex AI
Development IDE
Vertex AI Workbench
Colab Enterprise
Vertex AI SDK
Colab | Cloud Workstations | Anywhere
MLOps
CI/CD
Orchestration with Pipelines
Model Lifecycle Services
Feature Management
Offline | Online
Training
Single | Distributed | RAY
Serving
Batch | Online | Vector Search
Monitoring
Continuous | Alerting
Govern | Manage | Observe
Model Registry
Flexible | Versioned
Evaluate
Custom | Managed
Experiment Tracking
Experiments | Runs | TensorBoard
Explain
Features | Examples

Insper

O que isso significa para a ciência de dados?

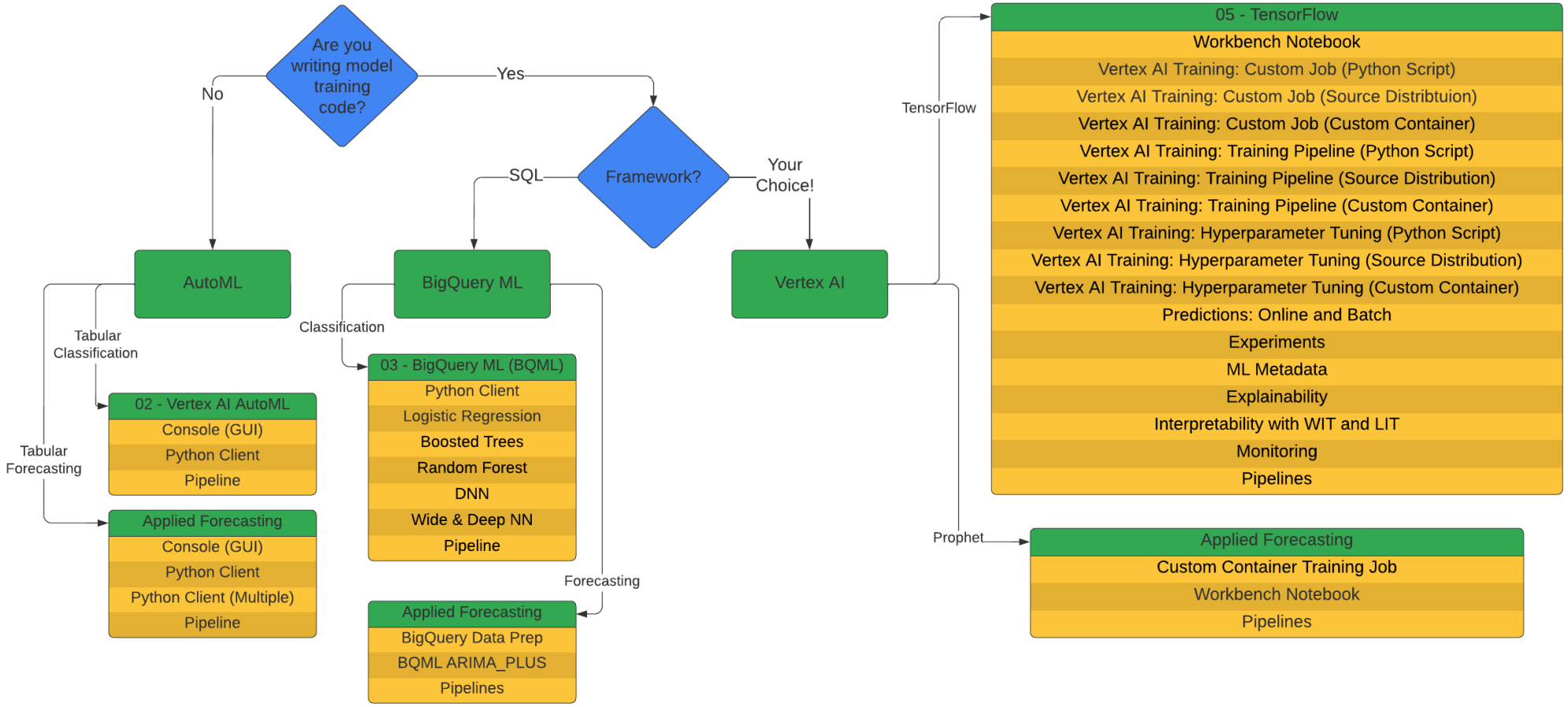
GCP Vertex AI



Insper

Existem múltiplas formas de construir uma solução de Machine Learning na nuvem

GCP Vertex AI

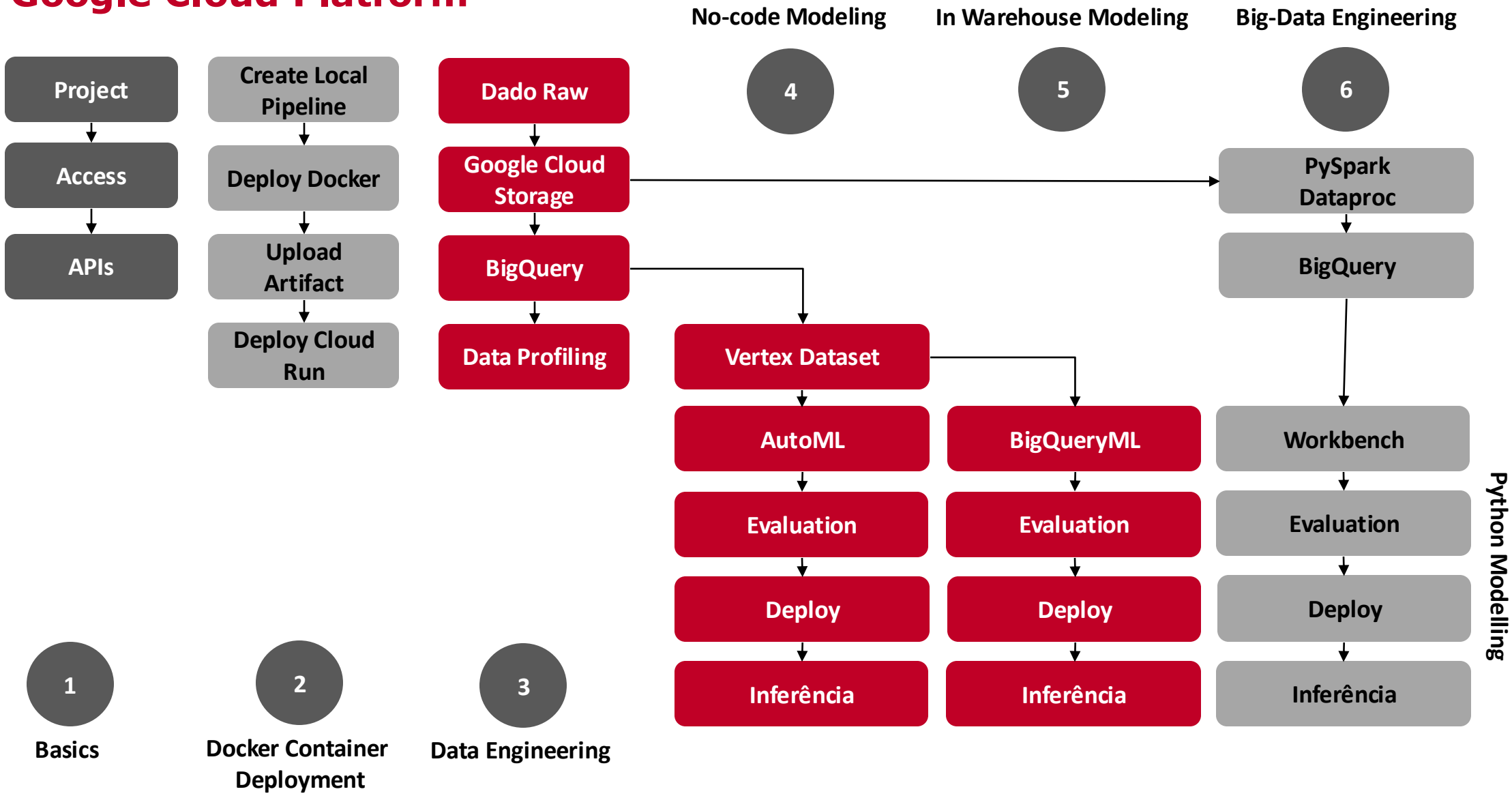


https://github.com/statmike/vertex-ai-mlops & https://www.youtube.com/playlist?list=PLgxF613RsGoUuEjJJxJW2JYyZ8g1qOUou

Insper

No GPC, diferentes workflows podem ser utilizados para deploy de serviços de ciência de dados

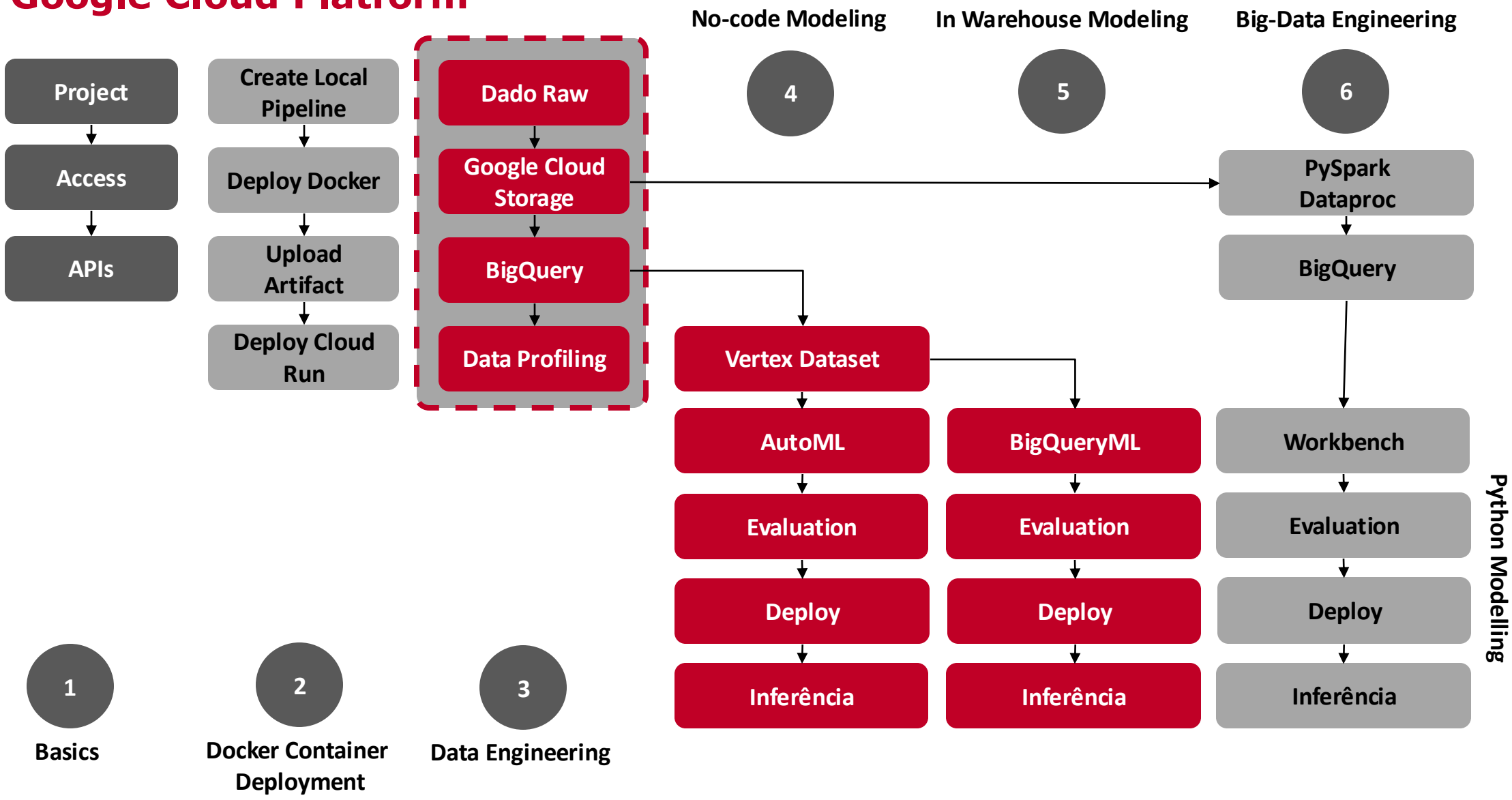
Google Cloud Platform



Insper

No GPC, diferentes workflows podem ser utilizados para deploy de serviços de ciência de dados

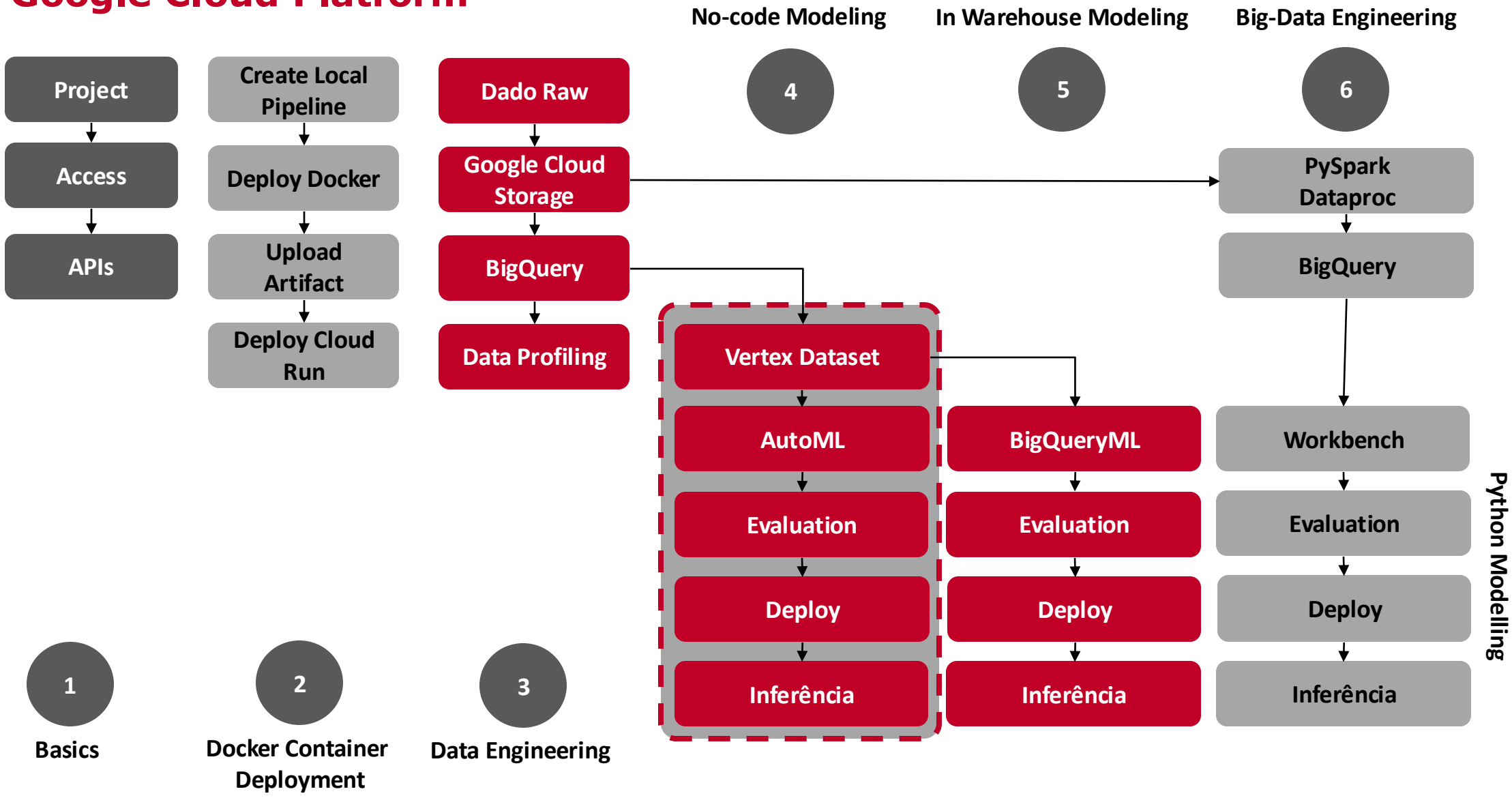
Google Cloud Platform



Insper

No GPC, diferentes workflows podem ser utilizados para deploy de serviços de ciência de dados

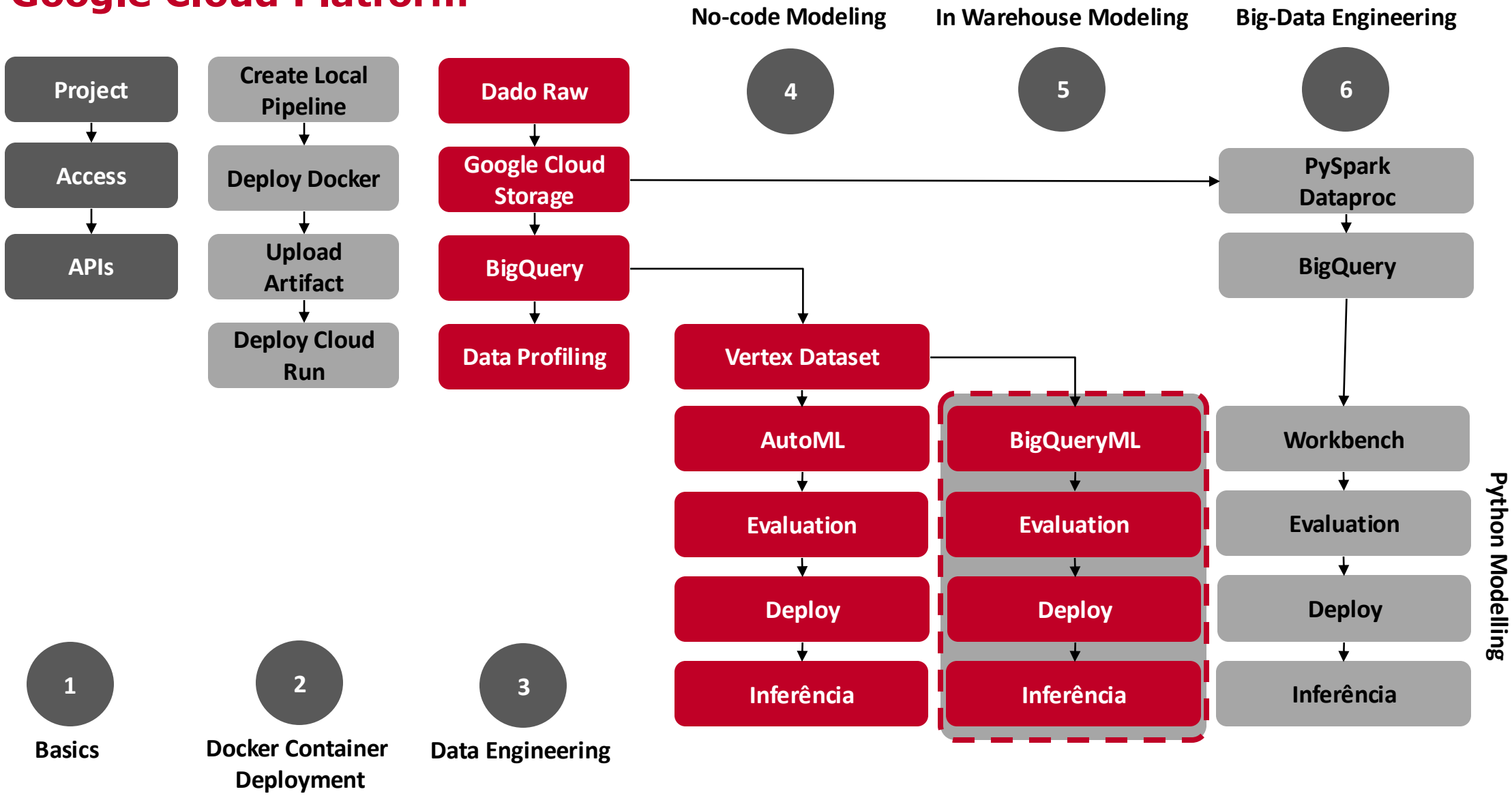
Google Cloud Platform



Insper

No GPC, diferentes workflows podem ser utilizados para deploy de serviços de ciência de dados

Google Cloud Platform



Insper

BigQuery ML – Machine Learning no DataWarehouse

O que é BigQuery ML?

Permite criar e executar modelos de machine learning diretamente no BigQuery usando SQL padrão, sem necessidade de mover dados ou conhecer Python/R.

Modelos Suportados

Supervised Learning

- Linear/Logistic Regression
- Random Forest
- Gradient Boosting
- Neural Networks

Unsupervised Learning

- K-means Clustering
- PCA
- Anomaly Detection
- Matrix Factorization

Time Series

- ARIMA
- ARIMA Plus
- Forecasting automático
- Seasonal decomposition

Streamming & Batch Processing

Batch Processing

- **Quando:** Grandes volumes, periodicidade definida
- **Latência:** Minutos a horas
- **Exemplos:** ETL noturno, relatórios diários
- **Ferramentas:** Apache Spark, MapReduce
- **Vantagens:** Throughput alto, custo eficiente
- **Casos de uso:** Data warehousing, ML training

Stream Processing

- **Quando:** Dados contínuos, tempo real
- **Latência:** Millisegundos a segundos
- **Exemplos:** Detecção de fraude, alertas
- **Ferramentas:** Apache Kafka, Flink
- **Vantagens:** Baixa latência, responsividade
- **Casos de uso:** IoT, monitoramento, recomendações

Ferramentas na Nuvem

Google Cloud

Batch: Dataproc, Dataflow
Stream: Dataflow, Pub/Sub

AWS

Batch: EMR, Glue
Stream: Kinesis, MSK

Azure

Batch: HDInsight, Data Factory
Stream: Stream Analytics, Event Hubs

Melhores Práticas

Segurança e Governança

- **IAM (Identity and Access Management):** Princípio do menor privilégio
- **Criptografia:** Em trânsito e em repouso
- **VPC:** Isolamento de rede
- **Audit Logs:** Rastreamento de todas as ações
- **Data Classification:** Identificação de dados sensíveis

Performance e Monitoramento

- **Observability:** Logs, métricas e traces
- **Alertas:** SLIs/SLOs bem definidos
- **Disaster Recovery:** Backups e replicação
- **Testing:** A/B testing para modelos

Otimização de Custos

Compute

- Preemptible instances
- Auto-scaling
- Sustained use discounts

Storage

- Lifecycle policies
- Storage classes
- Compression

BigQuery

- Partitioning
- Query optimization
- Slot reservations

Conclusões

Por que Nuvem para Data Science?

- **Escalabilidade:** Recursos sob demanda
- **Agilidade:** Foco no problema, não na infraestrutura
- **Colaboração:** Ambientes compartilhados
- **Inovação:** Acesso a tecnologias de ponta
- **Custo:** Pay-per-use, sem CAPEX

Insper