

Operationalizing AI Models in Industrial Settings

Mayukha Pal, Ph.D.






Global R&D Leader – Cloud & Advanced Analytics

ABB Ability Innovation Center, Hyderabad

© 2024 ABB. All rights reserved.



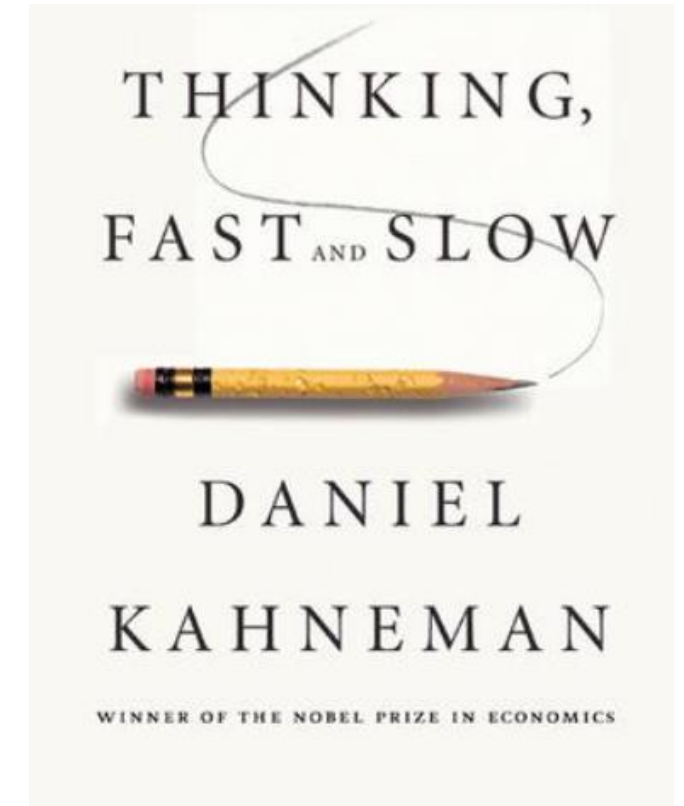
Agenda

01	 Brief on Causal AI at ABB Electrification...
02	 and The problem statement...
03	 Recap Docket and Microservices...
04	 Our Solution approach...
05	 Scope for future development, Engagements with Mathworks...

Human and Computational Thinking

System 1 & 2 thinking comparison

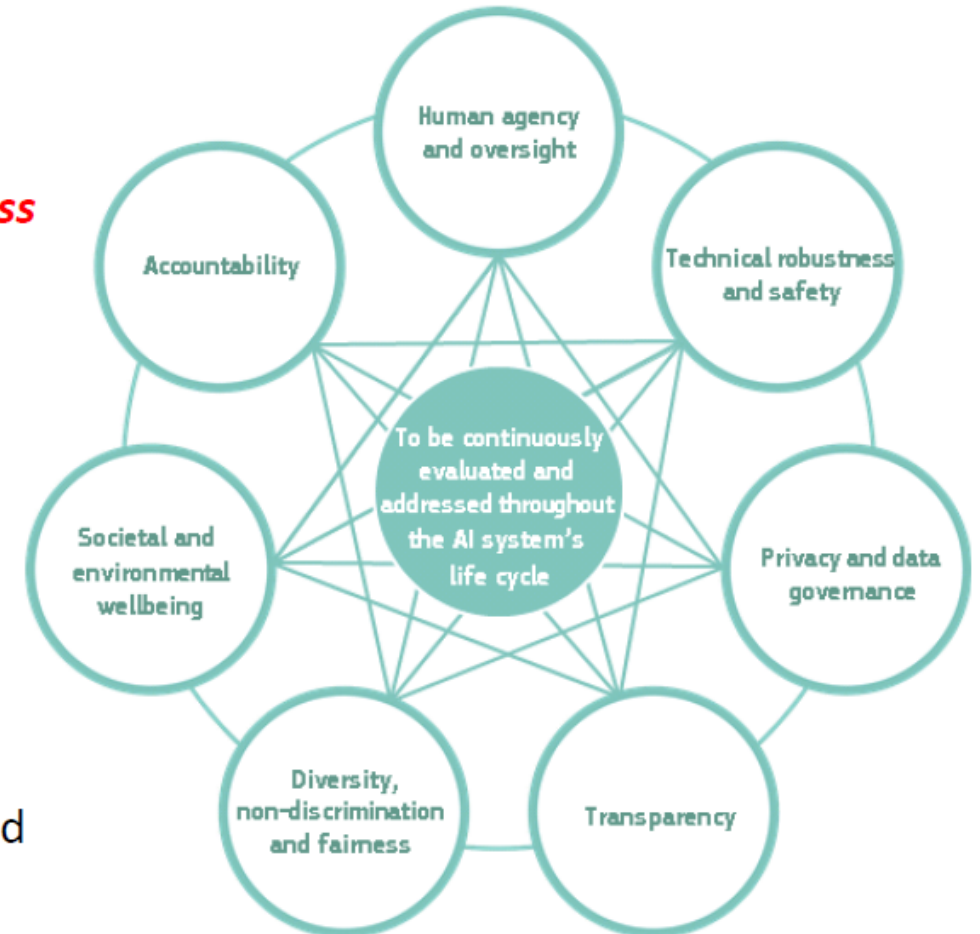
System 1 "Fast"	System 2 "Slow"
DEFINING CHARACTERISTICS Unconscious Effortless Automatic	DEFINING CHARACTERISTICS Deliberate and conscious Effortful Controlled mental process
WITHOUT self-awareness or control "What you see is all there is."	WITH self-awareness or control Logical and skeptical
ROLE Assesses the situation Delivers updates	ROLE Seeks new/missing information Makes decisions



Trustworthy AI

Definition

- Goal
 - establish a continuous interdisciplinary dialogue for investigating methods and methodologies
 - ***“To create AI systems that incorporate trustworthiness by design”***
- Organized along the 6 dimensions of Trustworthy AI:
 - Explainability,
 - Safety and Robustness,
 - Fairness,
 - Accountability,
 - Privacy, and
 - Sustainability
- One transversal task that links the 6 dimensions among and ensures coherence and coordination across the activities.



The Paradigm

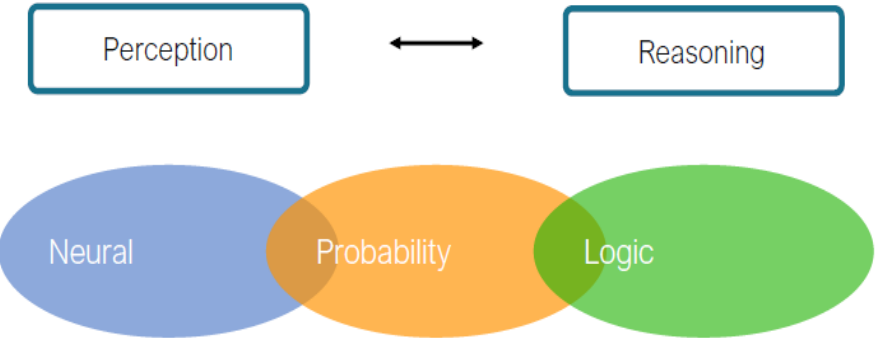
Core Learning Communities that need Integration

- ❖ Deep & Probabilistic Learning
- ❖ Neuro-Symbolic Computation (NeSy)
- ❖ Statistical Relational AI (StarAI)
- ❖ Constraint Programming & Machine Learning
- ❖ Knowledge graphs for reasoning
- ❖ And apply ... in e.g. computer vision

INTEGRATION

INGREDIENTS

COMPLEXITY
OF
INTEGRATION

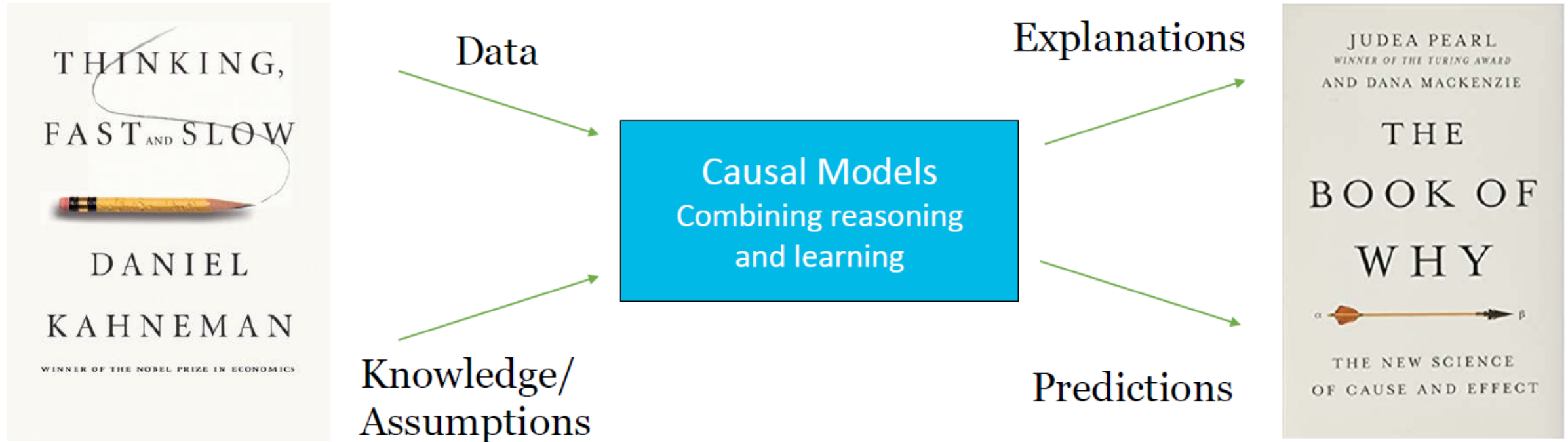


7 dimensions for integration

- 1.Underlying Structure (directed/undirected)
- 2.Inference (grounding/proof-oriented)
- 3.Paradigms (Logic, Probability, Neural)
- 4.Representations (Symbolic, SubSymbolic)
- 5.Discrete/Continuous (Boolean vs Fuzzy)
- 6.Learning (Parameter versus Structure)
- 7.Type of Logic (Propositional, First Order Logic or other forms)

Causal AI – the way Forward

the WHY – Cause and effect to the prediction



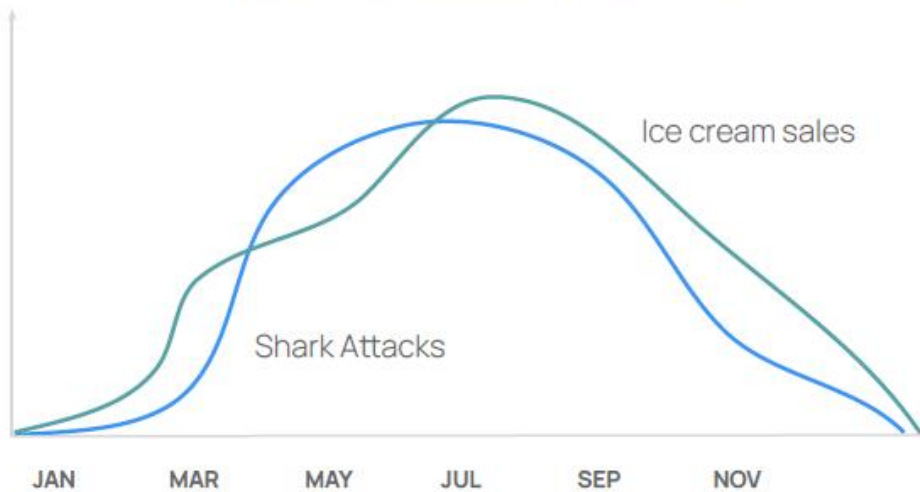
Why Causal AI

Actionable AI Possibilities

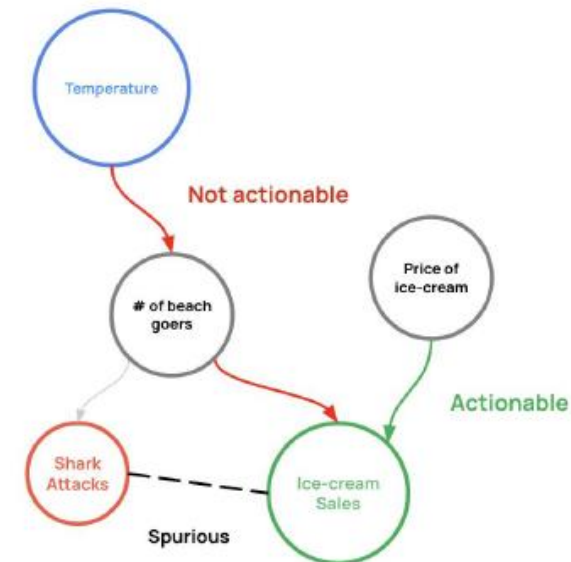
This is because causal AI is fundamentally different to traditional machine learning approaches

Traditional ML models are purely **associative (correlation) based**, spurious correlations may lead to models giving wrong conclusions and bad decisions.

E.g. Shark attacks drive ice cream sales?

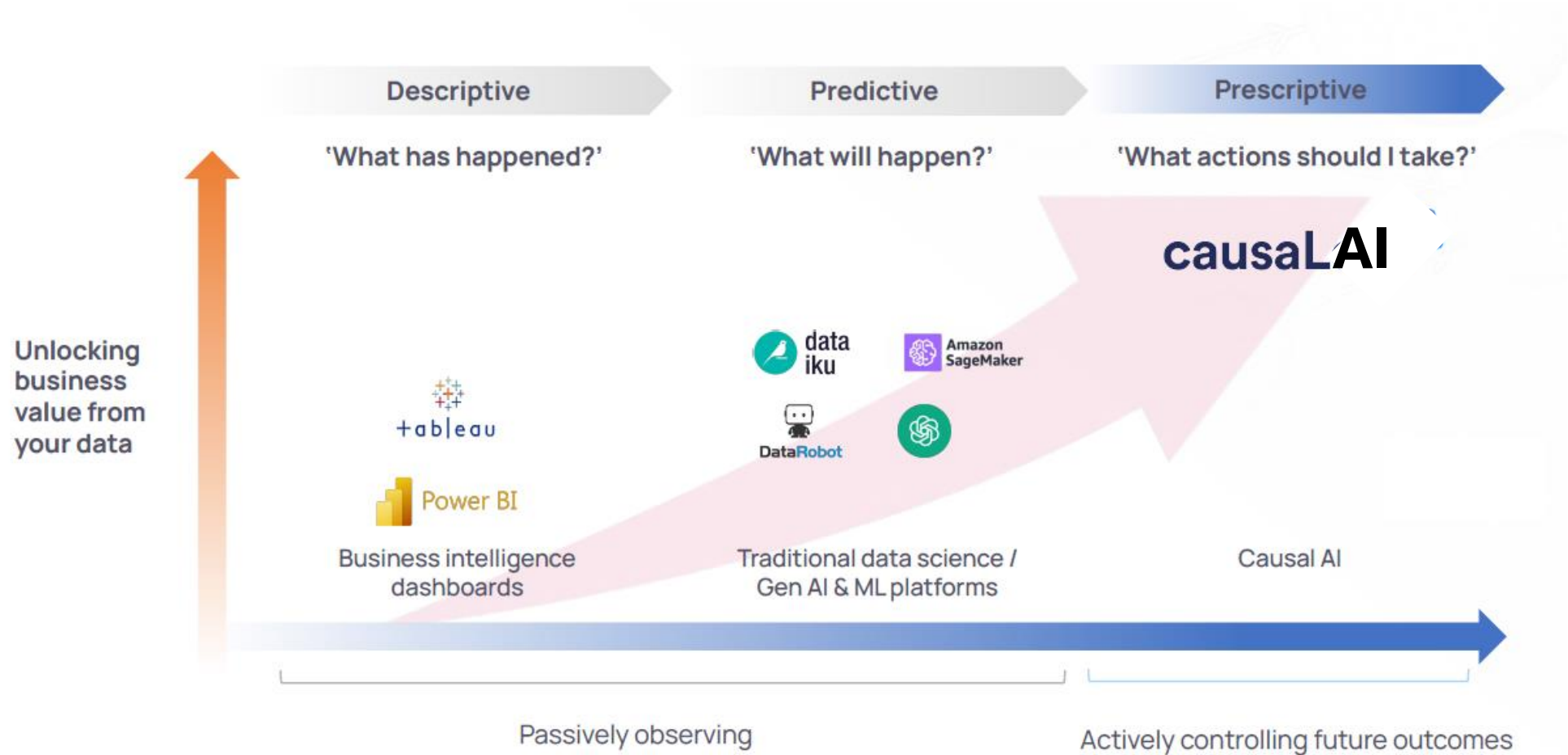


A causal analysis of the data **explains why this is false** and offers **actionable insights**.



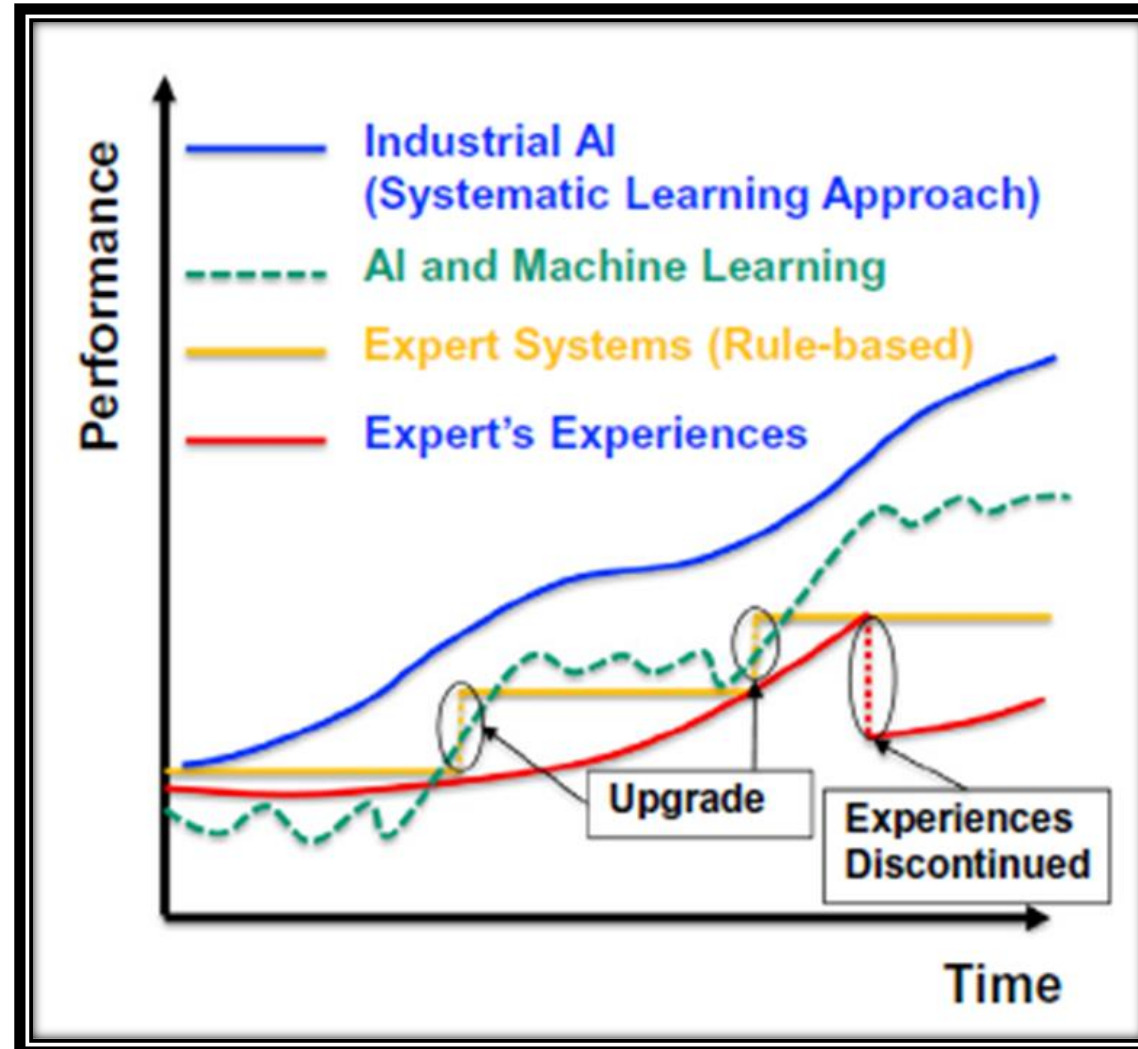
Shape the future of AI

Prescription with Causal AI



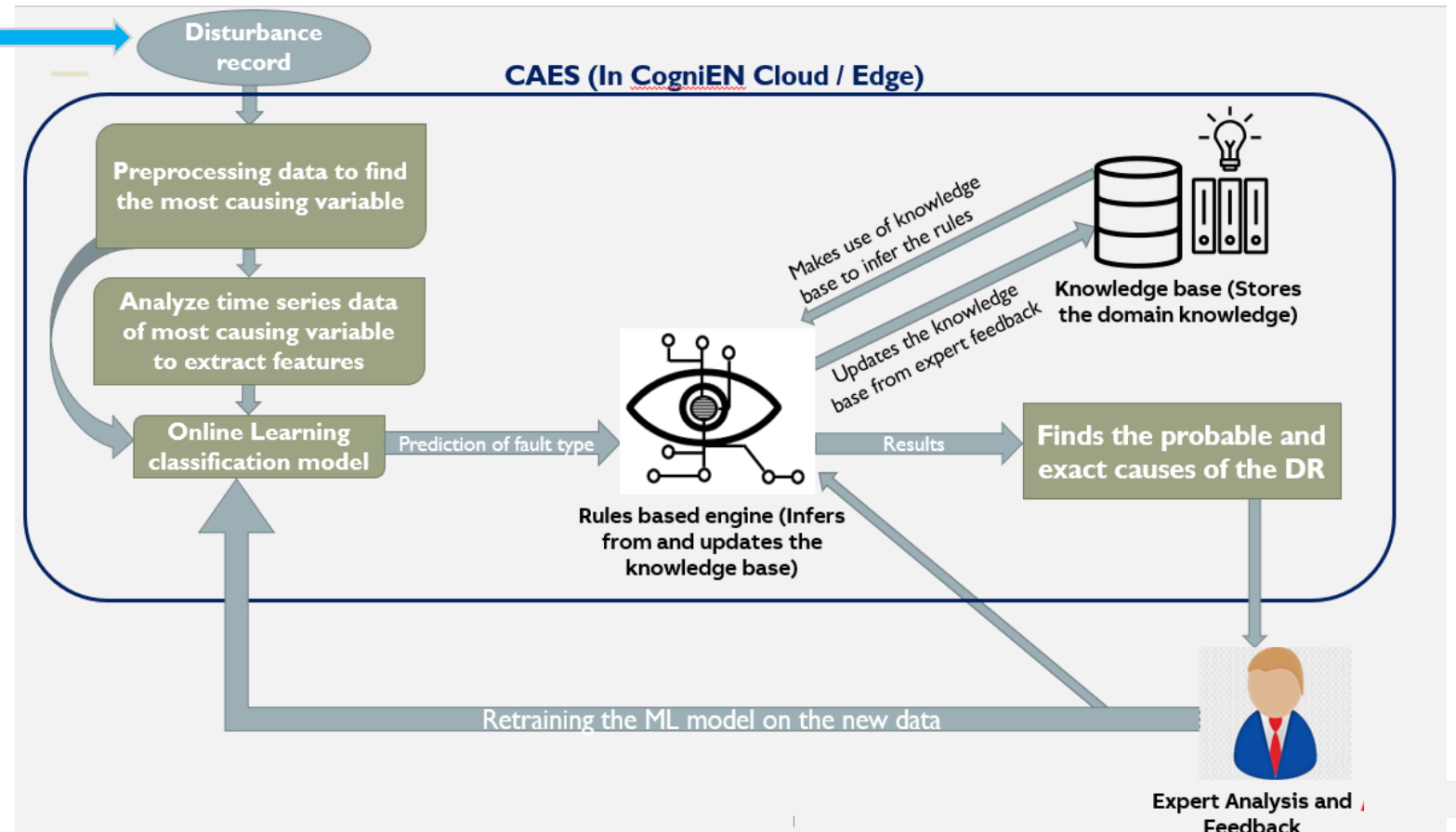
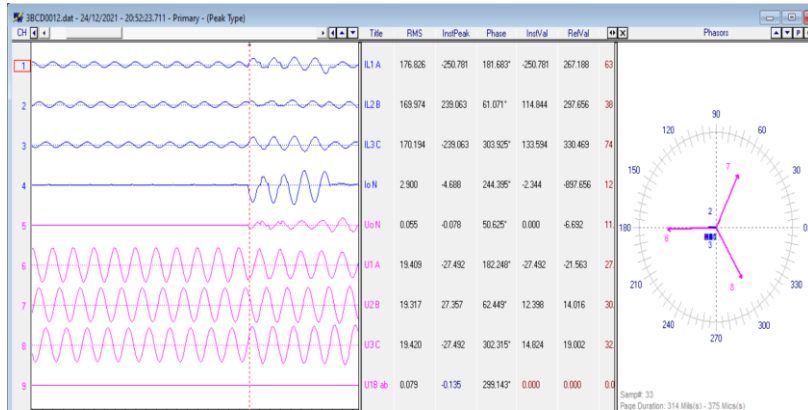
Industrial AI + Causal AI

A Systematic Learning Approach



CogniEN's Causal AI for Electrical Substations

ABB's Electrical Disturbance Record Causal Analysis Expert System



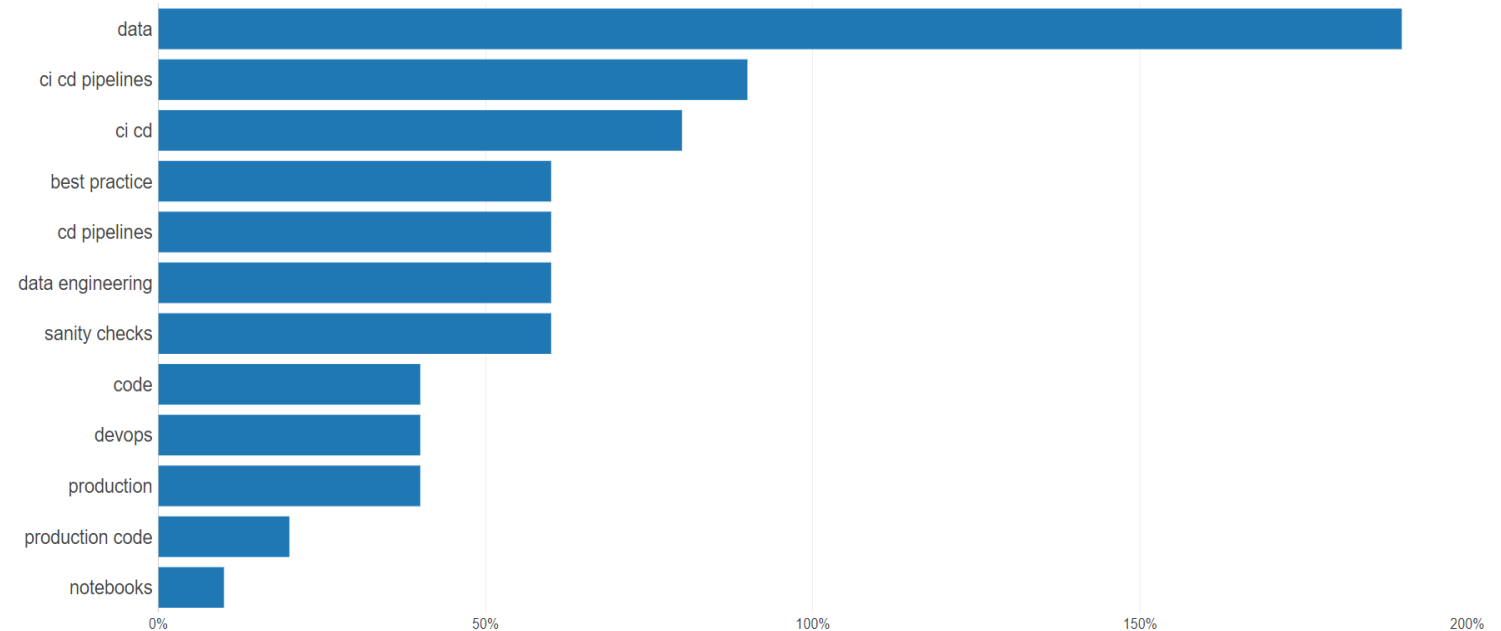
Traditional AI Deployment Challenges

Traditional AI deployment methods often face challenges

- Manual translation of AI models into production-ready code
- Integration into Existing Systems
- Lack Of Standardization
- The High Costs of Implementing AI
- Navigating Data Infrastructure Challenges
- Integration complexities with existing systems and frameworks
- Extensive functionality testing to ensure model accuracy and performance

These challenges lead to:

- Prolonged deployment cycles and time-to-market
- Difficulty in scaling AI applications to meet demand
- Increased risk of errors and inconsistencies



Overview on ML Model Operationalization

Containers and microservice architectures have transformed IT infrastructure

- Enable repeatable, scalable, and zero-downtime application deployment
- Applicable both in the cloud and on-premises environments

Infrastructure as Code (IaC) paradigm

- Application deployment becomes declarative and version-controlled
- Enables consistent and reproducible deployments

Approach: Building Docker and microservices for operationalizing AI applications

- Generating microservice containers for seamless deployments
- Streamlining data exchange using RestAPI communication
- Reducing manual efforts in translation, integration, and functionality testing



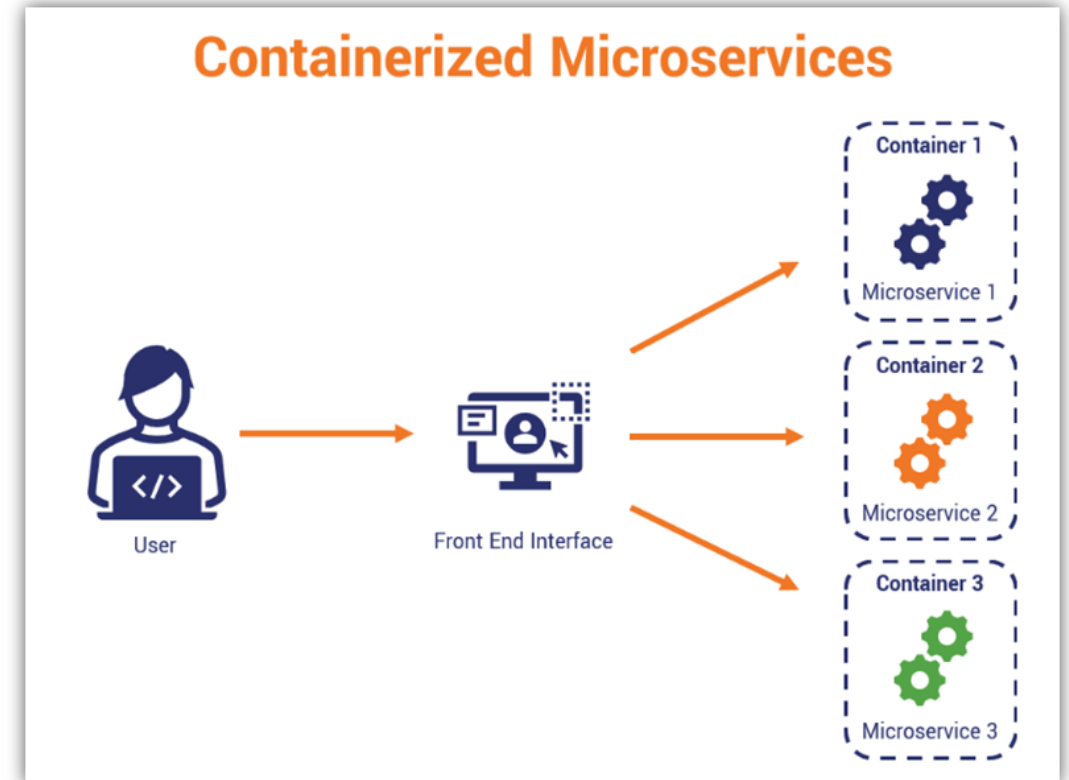
Repeatability



Scalability



Zero Downtime



Introduction to Docker

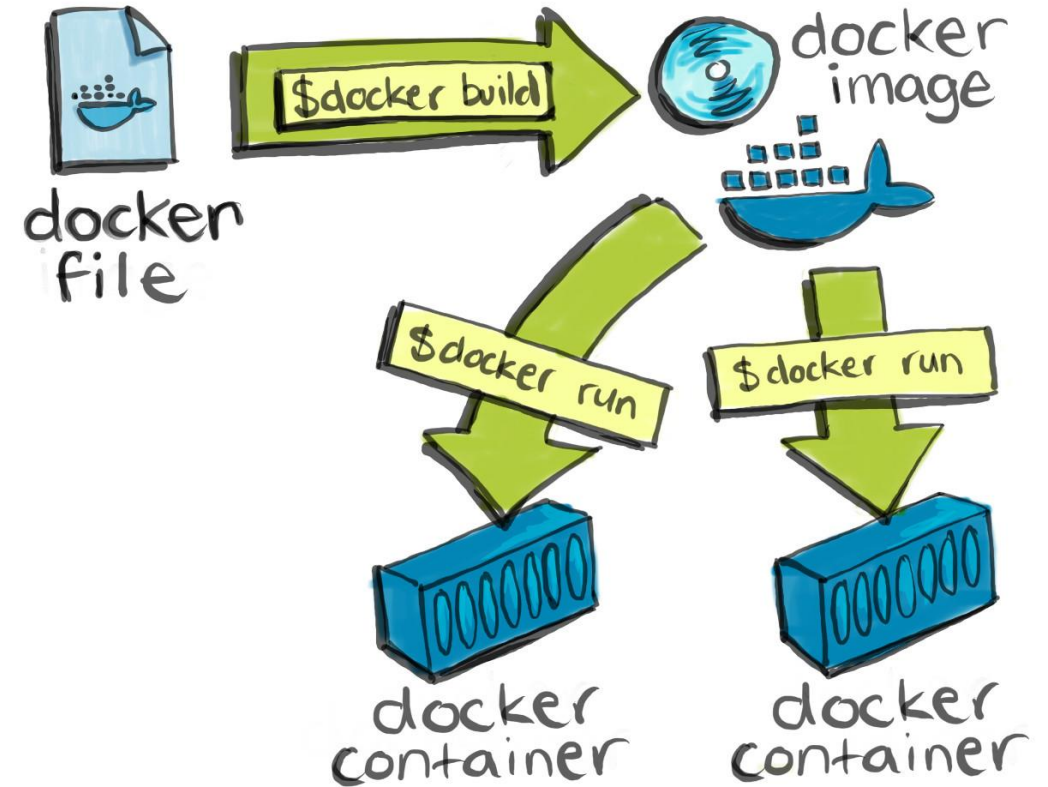
A platform for developing, shipping, and running applications using containers

Key concepts:

- Containers: Lightweight, standalone executable packages that include everything needed to run an application
- Images: Read-only templates used to create containers, defined by a Dockerfile
- Dockerfile: A text file containing instructions to build a Docker image

Benefits of using Docker for application deployment:

- Consistency across environments (development, testing, production)
- Isolation and resource efficiency
- Portability and ease of scaling
- Faster deployment and updates



Introduction to Microservices

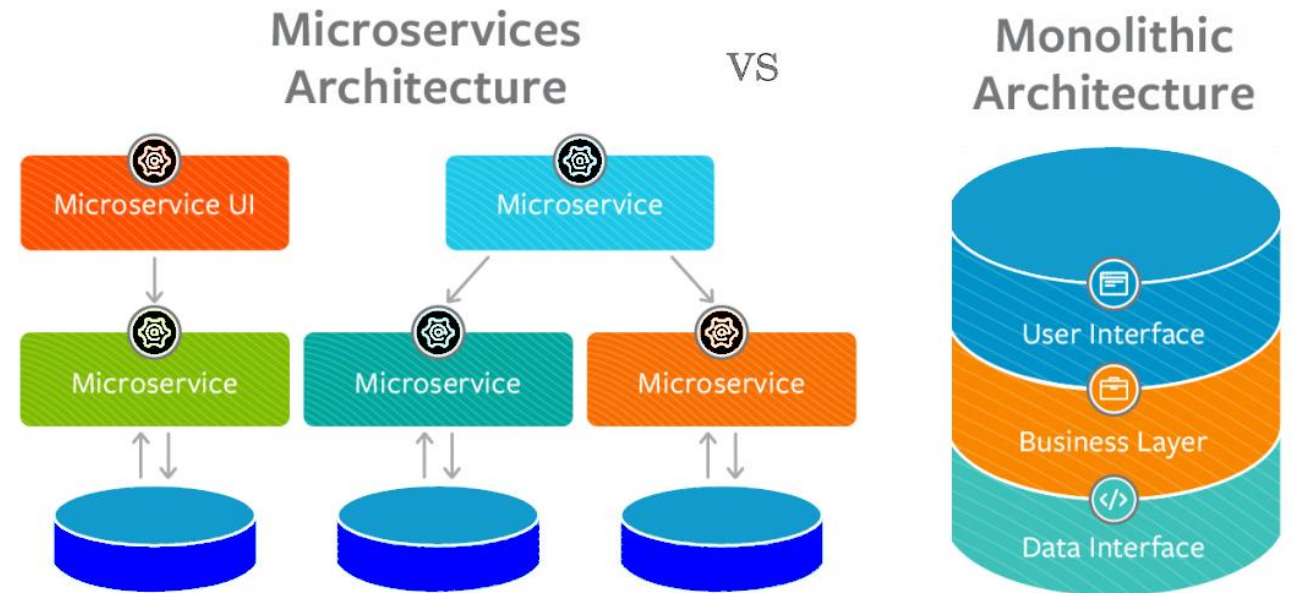
An architectural approach to develop applications as a collection of small, independently deployable services

Key characteristics of microservices:

- Focused on a single business capability or function
- Autonomous and loosely coupled
- Communicate through well-defined APIs (e.g., REST)
- Independently deployable and scalable

Advantages of microservices for AI applications:

- Modularity and flexibility in development and deployment
- Easier scaling of individual components based on demand
- Faster iteration and updates to specific services
- Improved fault isolation and resilience



Building a Microservice Docker Image with MATLAB

MATLAB is a powerful platform for scientific computing and AI development

- Offers a wide range of toolboxes and libraries for machine learning, deep learning, Statistical Analytics and data analysis etc
- Provides a productive environment for developing and prototyping AI algorithms

Creating a microservice Docker image using MATLAB

- Enables the deployment of MATLAB-based AI models as scalable microservices
- Simplifies the integration of MATLAB code with other systems and frameworks

Key steps in building a MATLAB microservice Docker image:

- Generating file paths for the main function and its dependencies
- Creating a production server archive using MATLAB Compiler SDK
- Packaging the archive into a Docker image

Microservices

Deploy MATLAB® functions as microservice

R2024a

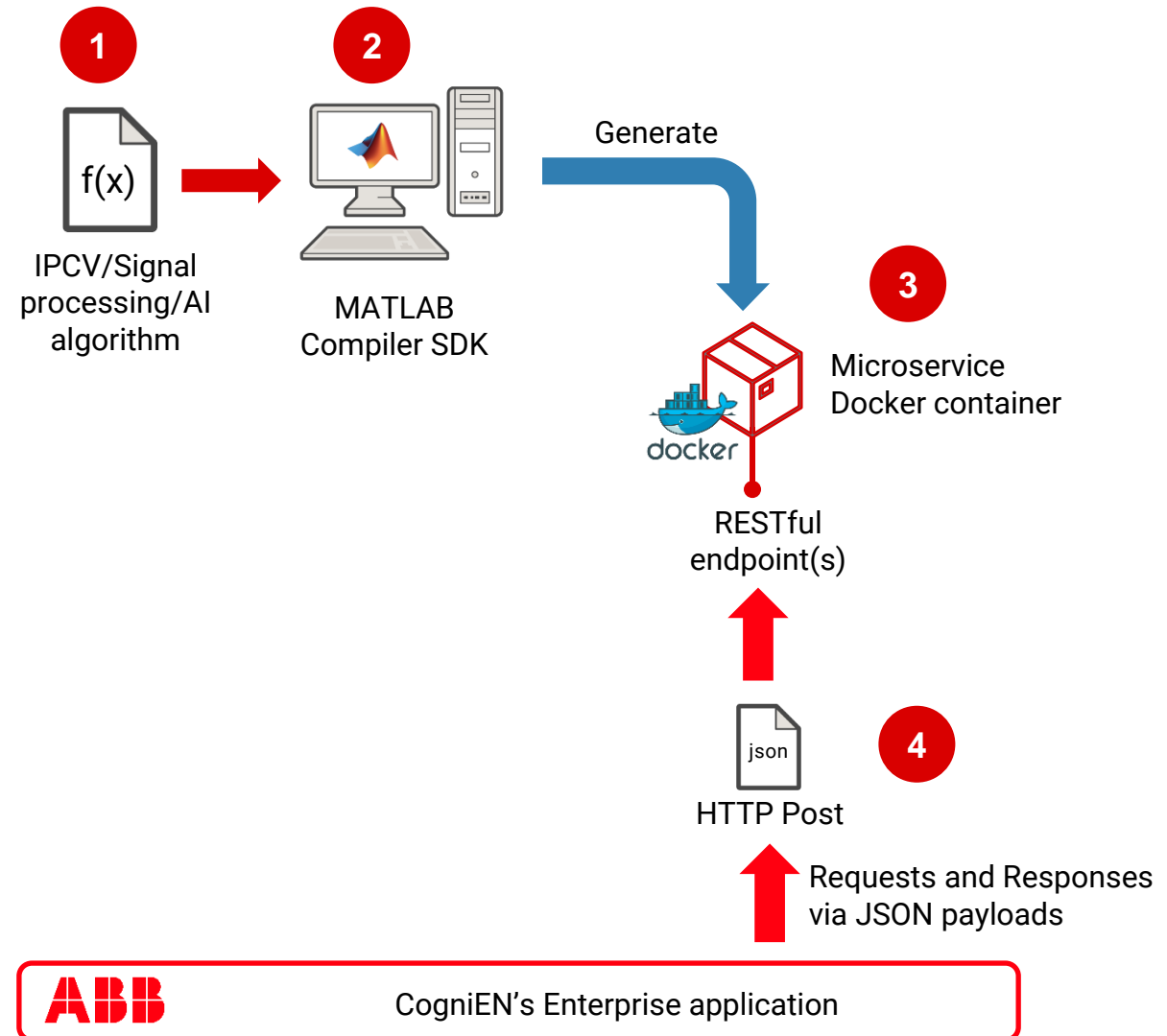
You can deploy MATLAB functions as a microservice by packaging them into a Docker® container. The microservice Docker container provides an HTTP or HTTPS endpoint to the MATLAB function and accepts RESTful requests.

To deploy a MATLAB function as a microservice, you package a MATLAB function into a deployable archive, and then create a Docker image that contains the archive and a minimal MATLAB Runtime package. You can then run the microservice Docker image and make RESTful requests to the service using any programming language that has HTTP libraries, including MATLAB Production Server™ client APIs.

Streamline the Deployment of IPCV/SPC and AI Algorithms

Artificial Intelligence (AI)

- **Goal: Deploying IPCV/Signal Processing and AI algorithms as Microservice**
- **Challenges:**
 - Translating the MATLAB algorithms into Python and Java Scripts to integrate with their larger deployment infrastructure.
 - Data handling and exchange was challenging with shared library workflow for their algorithms.
- **Approach:**
 - Customized pipeline to generate Microservice containers for seamless deployments
 - Data exchange is now streamlined with RestAPI communication
 - Reduction in several hours of manual translation, integration and functionality testing.



Solution Brief

Example use case: Building a microservice for fault causal analytics using MATLAB

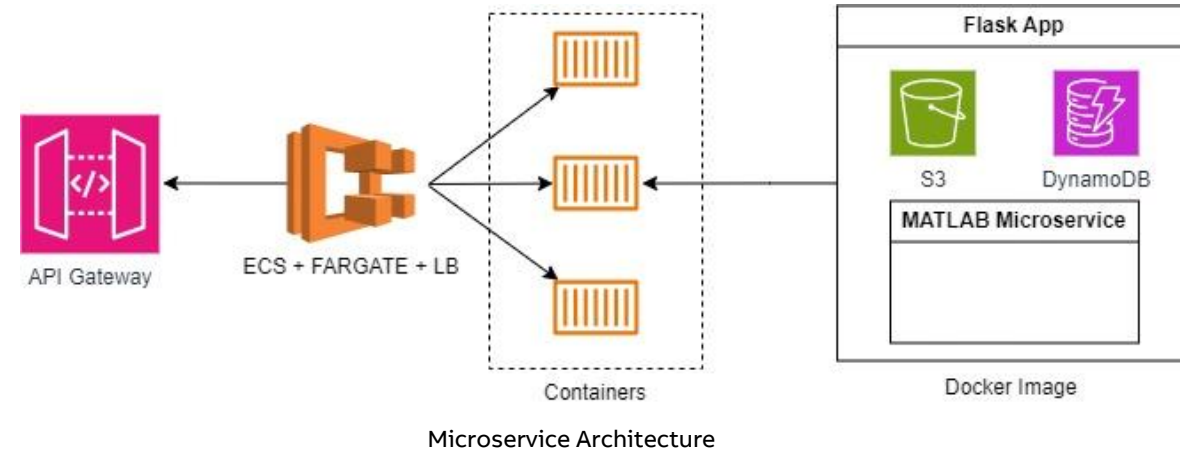
- Objective: Deploy a MATLAB-based fault analytics model as a scalable microservice
- Input: Disturbance record files from electrical systems
- Processing: MATLAB code for feature extraction and fault prediction

Microservice architecture:

- MATLAB microservice deployed on Amazon ECS (Elastic Container Service) with Fargate and Load Balancer
- RestAPI communication for data exchange (Flask)
- Integration with Amazon S3 & DynamoDB for data storage and retrieval

Benefits of the microservice approach:

- Scalability and flexibility in deploying the analytics model
- Seamless integration with other systems and data sources
- Reduced manual effort in translation and functionality testing
- Time to market and production deployable



Example Use Case: Dockerizing the MATLAB Microservice

Dockerfile: A text file containing instructions to build the Docker image

Key components of the Dockerfile:

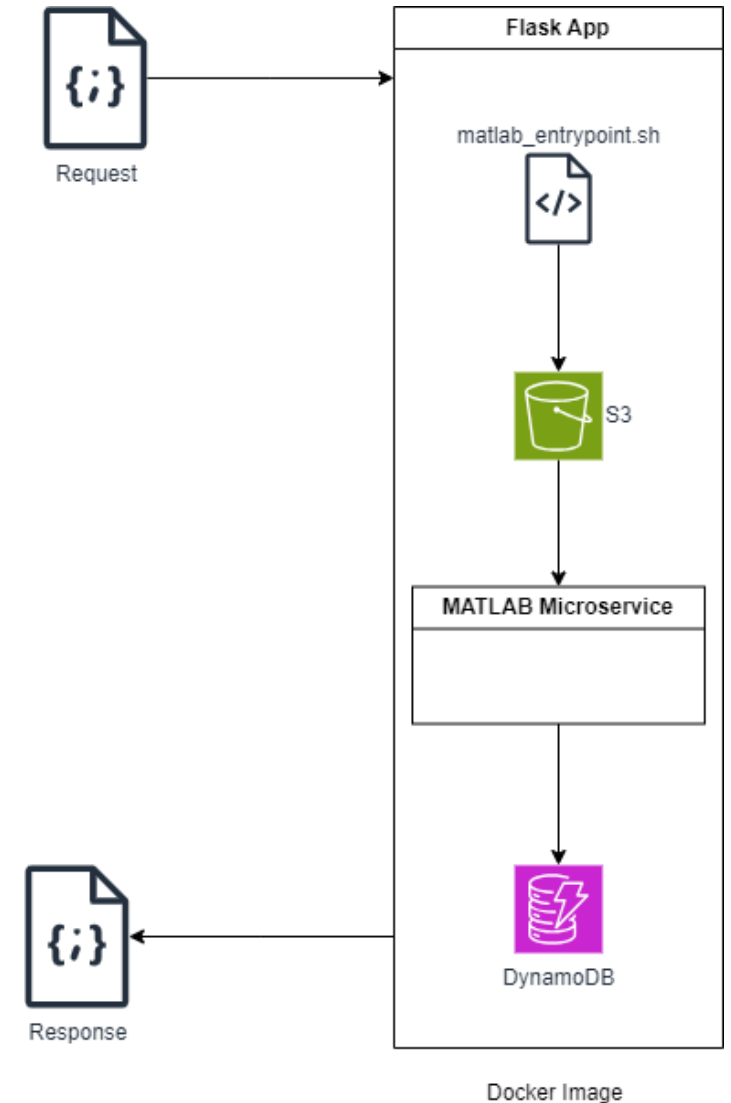
- Base image: MATLAB Production Server runtime (gccca-image)
- Installation of Python and required packages using requirements.txt
- Setup of the custom function directory and copying of function code
- Setting executable permissions for the MATLAB entry point script
- Exposure of the necessary port and setting the entry point

MATLAB entry point script (matlab_entrypoint.sh):

- Required to start the MATLAB Production Server and keep it running in the background
- Ensures the MATLAB microservice is available to handle incoming requests

Final Docker image:

- Includes the MATLAB Production Server runtime, Python dependencies, and custom function code
- Ready to be deployed as a microservice on a container orchestration platform

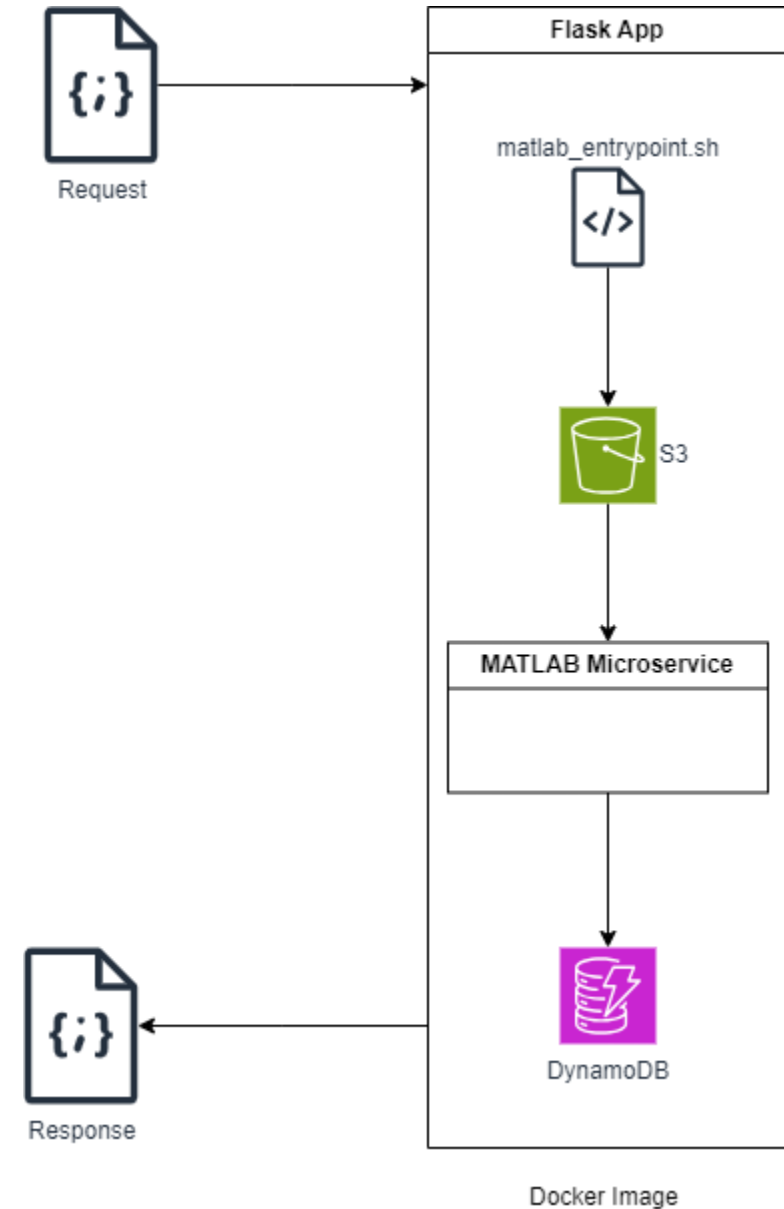


Example Use Case: Python(Flask) Application (app.py) - Part 1

- app.py: The main Python application file serving as the Flask web server
- Handles the /process endpoint for incoming POST requests
- Utilizes the MATLAB microservice for processing analytics data
- Important code snippet:
 - Executes the MATLAB entry point script in the background using subprocess.Popen()
 - Ensures the MATLAB Production Server is running and ready to handle requests

```
script_path = './matlab_entrypoint.sh'  
subprocess.Popen([script_path])
```

- Interacts with Amazon S3 for retrieving input data and storing results
- Stores fault analytics results in Amazon DynamoDB for further analysis and visualization



Example Use Case: Python(Flask) Application (app.py) - Part 2

Amazon S3 (Simple Storage Service):

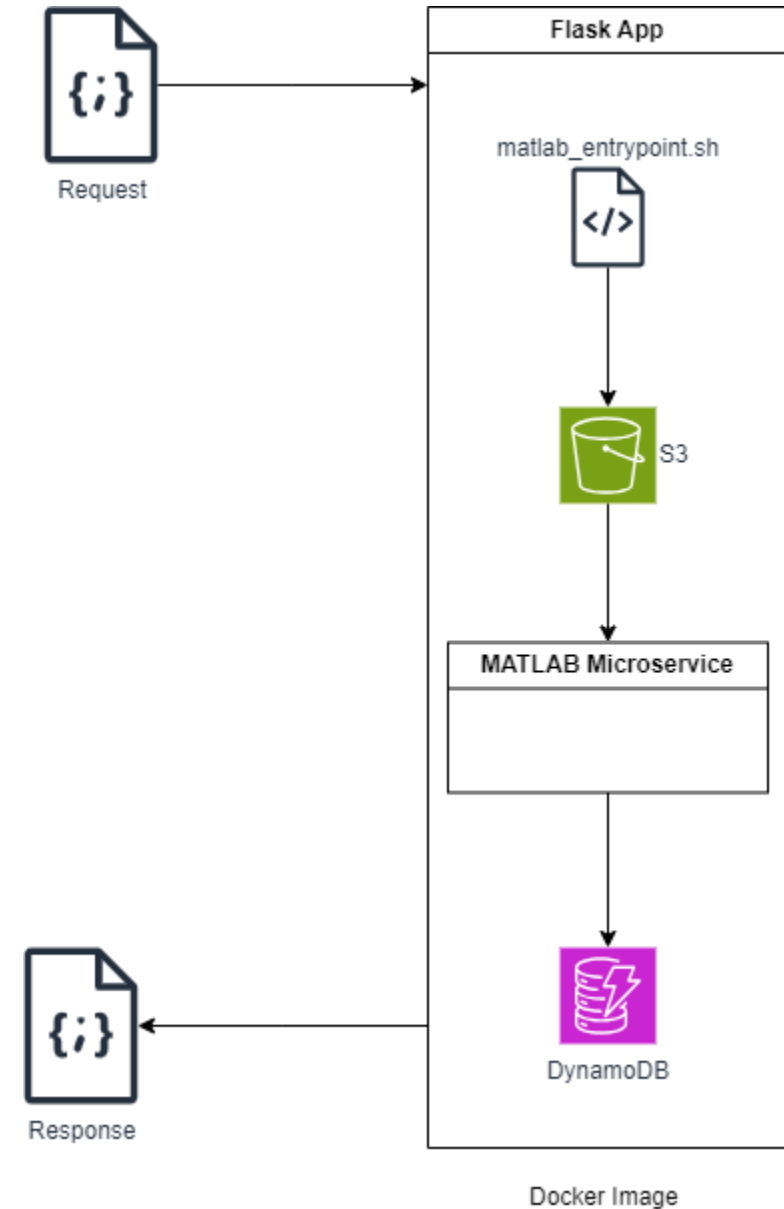
- Used for storing input data files (disturbance records) and processed results
- Provides scalable and durable object storage
- Enables easy access to data from multiple microservice instances

Interaction with the MATLAB microservice:

- app.py retrieves input data files from Amazon S3
- Sends the data to the MATLAB microservice running in background
- Receives the processed results from the MATLAB microservice
- Stores the results in Amazon S3 and updates the corresponding metadata in DynamoDB

Amazon DynamoDB:

- Used for storing fault analytics results and metadata
- Offers a fast and flexible NoSQL database service
- Allows for efficient querying and analysis of fault analytics data



Example Use Case:

Amazon ECS Cluster with Fargate and Load Balancer

Amazon ECS (Elastic Container Service):

- A fully managed container orchestration service
- Allows for easy deployment, management, and scaling of containerized applications

AWS Fargate:

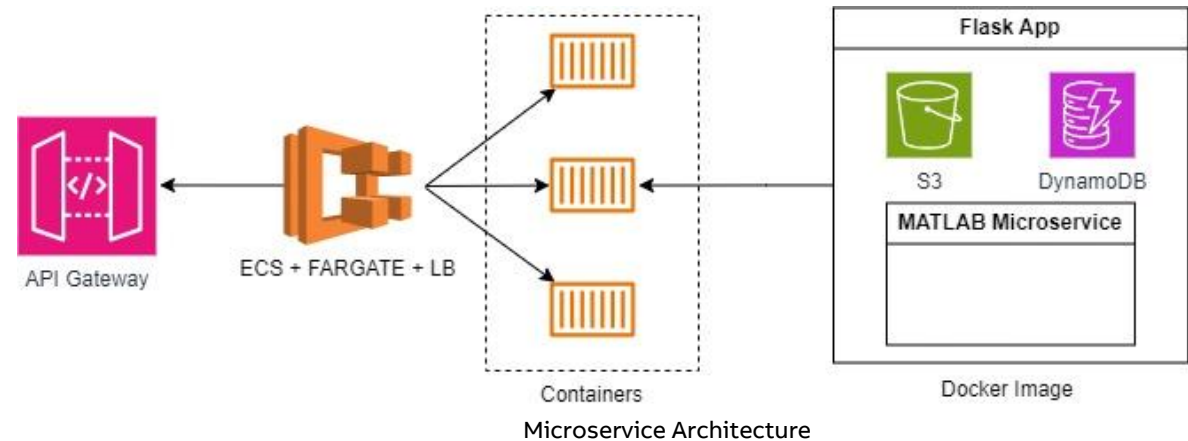
- A serverless compute engine for containers
- Eliminates the need to manage underlying infrastructure
- Automatically scales and manages the resources required to run containers

Benefits of using Fargate for serverless compute:

- Focus on application development rather than infrastructure management
- Automatic scaling based on application demands
- Pay only for the resources consumed by the containers

Load Balancer:

- Distributes incoming traffic across multiple instances of the application
- Ensures high availability and fault tolerance
- Automatically scales to handle increased traffic loads



Introduction to AWS CDK (TypeScript)

This architecture can be built using AWS CDK!

AWS CDK (Cloud Development Kit):

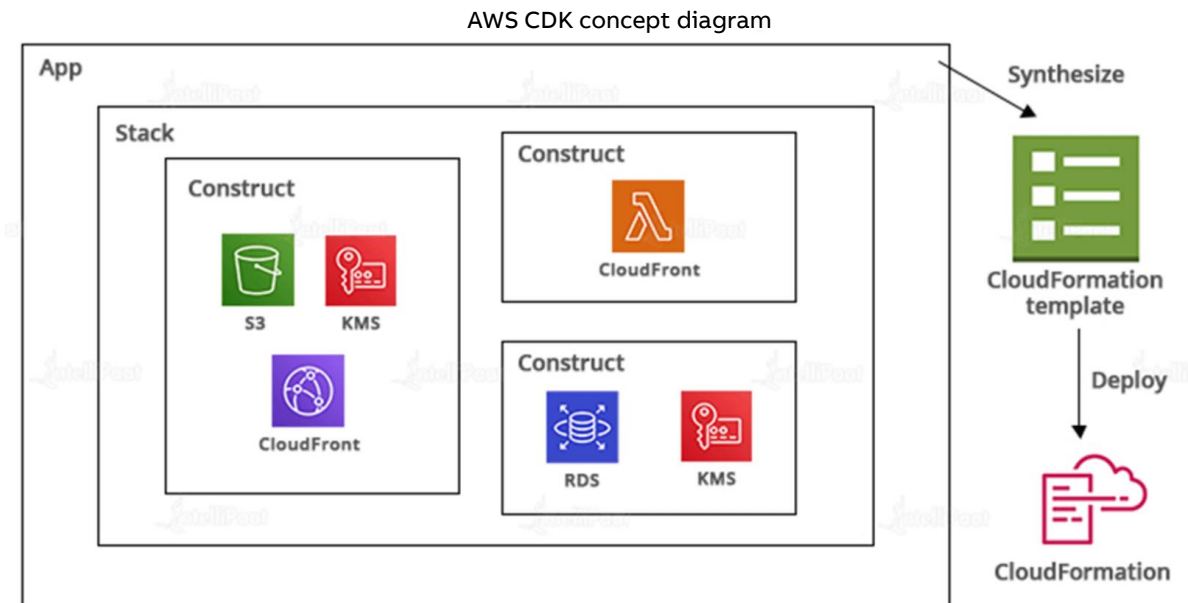
- An open-source software development framework for defining cloud infrastructure as code
- Allows for the provisioning of AWS resources using familiar programming languages (e.g., TypeScript)

Benefits of using AWS CDK:

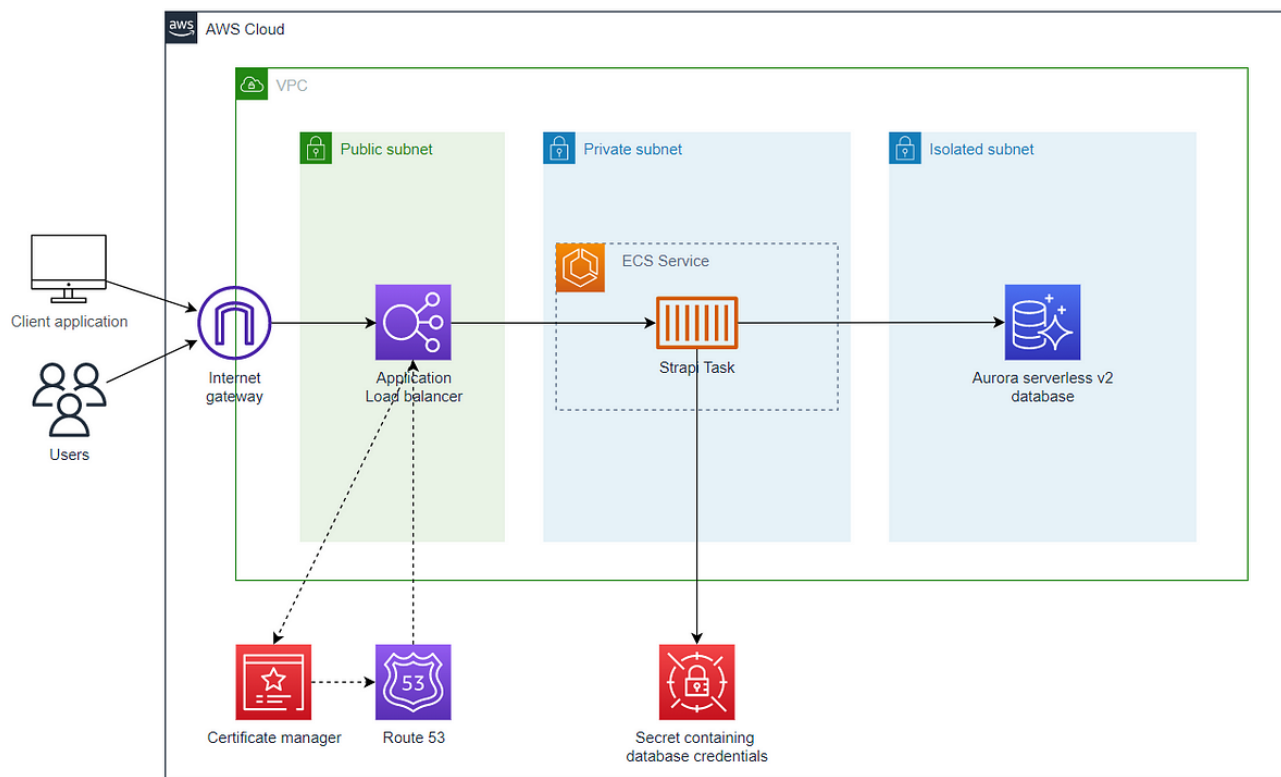
- Infrastructure as Code (IaC):
 - Define and manage infrastructure resources using code
 - Enable version control, code review, and collaboration
 - Ensure consistent and reproducible infrastructure deployments
- Abstraction and Reusability:
 - Provides high-level abstractions for AWS resources
 - Allows for the creation of reusable components and patterns
- Integration with AWS Services:
 - Seamless integration with various AWS services
 - Enables the provisioning of complex architectures with ease

TypeScript:

- A typed superset of JavaScript
- Offers static typing, enhancing code quality and maintainability
- Provides a rich development experience with IDE support and error detection



- The CDK code follows a declarative approach
- Resources are defined using TypeScript classes and constructs
- The CDK synthesizes the TypeScript code into an AWS CloudFormation template
- The CloudFormation template is used to provision the resources in the AWS account



AWS CDK ECS + FARGATE + LB architecture

Importance of a Deployment Pipeline

Importance of a deployment pipeline for microservices:

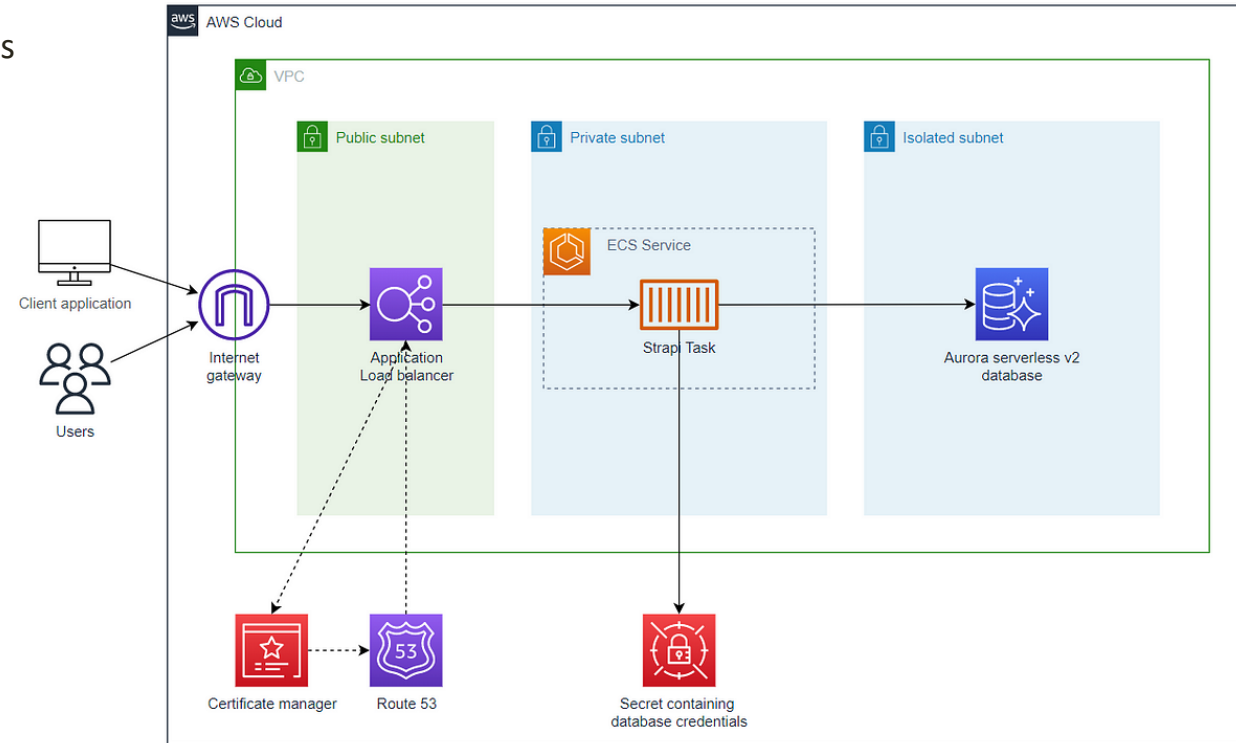
- Automates the process of building, testing, and deploying microservices
- Ensures consistent and reliable deployments
- Enables continuous integration and continuous deployment (CI/CD)

Components of a deployment pipeline:

- Source control management (e.g., Git)
- Build automation (e.g., Jenkins, AWS CodeBuild)
- Automated testing (unit tests, integration tests)
- Containerization (Docker)
- Deployment automation (e.g., AWS CodeDeploy, AWS ECS)

Benefits of a deployment pipeline:

- Faster time-to-market
- Reduced manual errors and inconsistencies
- Improved collaboration between development and operations teams
- Easier rollback and recovery in case of issues



AWS CDK ECS + FARGATE + LB architecture

Importance of RestAPI Communication

Role of RestAPI in microservice communication:

- Enables communication between microservices using HTTP protocol
- Provides a standardized and platform-independent interface
- Allows for loose coupling and independent development of microservices

RestAPI principles:

- Stateless communication
- Resource-based URLs
- HTTP methods (GET, POST, PUT, DELETE) for CRUD operations
- JSON or XML for data exchange

Example code snippets demonstrating RestAPI endpoints and requests:

- Endpoint definition in app.py:

```
@app.route('/process', methods=['POST'])
def process_data():
    # Process the request and return a response
    pass
```

- API request:

```
response = requests.post('http://api.example.com/process', json=data)
```

Benefits of the Microservice Architecture

Recap:

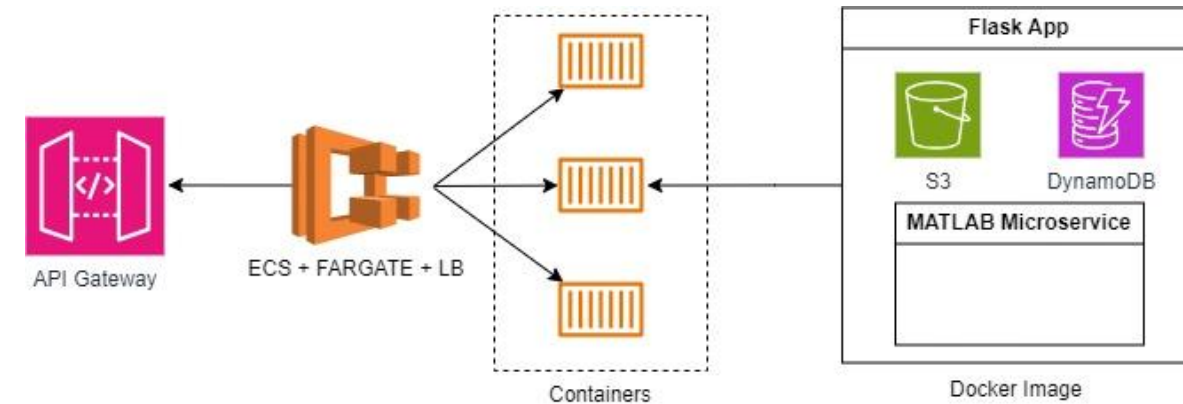
- Scalability: Independently scale individual microservices based on demand
- Flexibility: Develop, deploy, and update microservices independently
- Maintainability: Smaller, focused codebases that are easier to understand and maintain
- Technology diversity: Use different technologies and frameworks for each microservice

Reduced effort in manual tasks:

- Automated translation of AI models into production-ready microservices
- Seamless integration with other systems and data sources
- Automated testing and deployment processes

Improved development agility:

- Faster development cycles and time-to-market
- Easier experimentation and iteration
- Reduced dependencies between teams



Microservice Architecture

Real-World Use Cases

Netflix:

- Migrated from a monolithic architecture to microservices
- Achieved improved scalability, fault tolerance, and development agility
- Enabled personalized content recommendations and faster feature releases

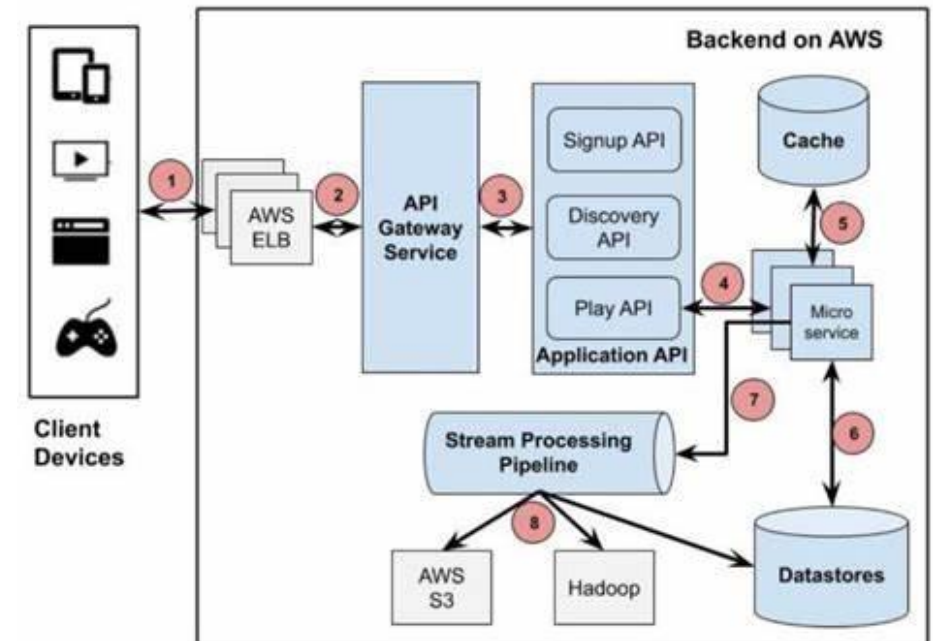
Spotify:

- Adopted microservices to handle the growing complexity of their music streaming platform
- Improved scalability and allowed for independent development of features
- Facilitated experimentation and A/B testing for better user experiences

Challenges faced and lessons learned:

- Managing the increased complexity of a distributed system
- Ensuring proper communication and data consistency between microservices
- Implementing effective monitoring and logging for troubleshooting
- Investing in automation and DevOps practices for efficient deployment and operations

Microservices Architecture at **NETFLIX**



Credits: Cao Duc Nguyen

Best Practices and Considerations

Best practices for designing and implementing microservice architectures:

- Define clear boundaries and responsibilities for each microservice
- Use domain-driven design (DDD) principles to align microservices with business domains
- Implement loose coupling and high cohesion
- Ensure backward compatibility and versioning of APIs
- Adopt a DevOps culture and automate deployment processes

Considerations:

- Security: Implement authentication, authorization, and secure communication between microservices
- Monitoring: Establish centralized logging, monitoring, and alerting for microservices
- Resilience: Implement fault tolerance, circuit breakers, and graceful degradation
- Testing: Perform comprehensive testing at various levels (unit, integration, end-to-end)
- Continuous Integration/Deployment (CI/CD): Automate the build, test, and deployment processes

Tools and frameworks:

- Container orchestration: Kubernetes, Amazon ECS
- Service mesh: Istio, AWS App Mesh
- Monitoring and logging: Prometheus, Grafana, ELK stack
- CI/CD: Jenkins, GitLab CI, AWS CodePipeline

Conclusion

Recap:

- Microservices and Docker enable scalable and efficient deployment of AI applications
- The example use case demonstrated the process of building a MATLAB microservice using Docker and AWS services
- Microservice architectures offer benefits such as scalability, flexibility, and reduced manual effort
- Real-world examples showcase the successful adoption of microservices by companies like Netflix and Spotify
- Best practices and considerations ensure the effective design and implementation of microservice architectures

ABB