Automated Revenue Prediction Modeling (RPM)

Thom Miano M.S., RTI International
tmiano@rti.org

Key Topics: Python in production; DevOps; MLOps; process automation; data science; machine learning; Azure; financial analytics

Team Acknowledgement

Success of this project depended on close collaboration and teamwork.

Development Team*

Finance	Information Technology	Data Science
Laurie Braun	Suman Banjaree	Anna Godwin
Andres Forero	Radhe Dwivedi	Kim Janda
Kyle Hodgin	Adwitiya Pandey	Saki Kinney
Michael Justice	John Patterson	Thom Miano
Robyn Parker	Christine Piegza	Michael Wenger
Jolie Taylor	Maulik Shah	
	Kathleen Sheridan	
	Lee Whitbeck	
	Dharmveer Yadav	

* There are also many past and “non-development” contributors

Agenda – RPM System Design

1. Problem Definition
2. Solution: RPM
3. Machine Learning (ML) Layers
4. System Flow
    1. Data Integration
    2. Development-to-Deployment
    3. Automated Training
    4. Resource Summary
5. Conclusions

1. Problem Definition

Annually, financial analysts create budgets for the upcoming three fiscal years: (FY1) Very regulated and detailed; and (FY2 & FY3) High-level assessment and less detailed.

Limitations:

- Time consuming (~3 months)
- Manual (varying methodology)
- Detailed view constrained to FY1
- Expert-knowledge driven*

* Also a strength

2. Solution: RPM

Using historical and current project financial data, create an automated system that forecasts some of our most important financial metrics (e.g., revenue) over the next several years.

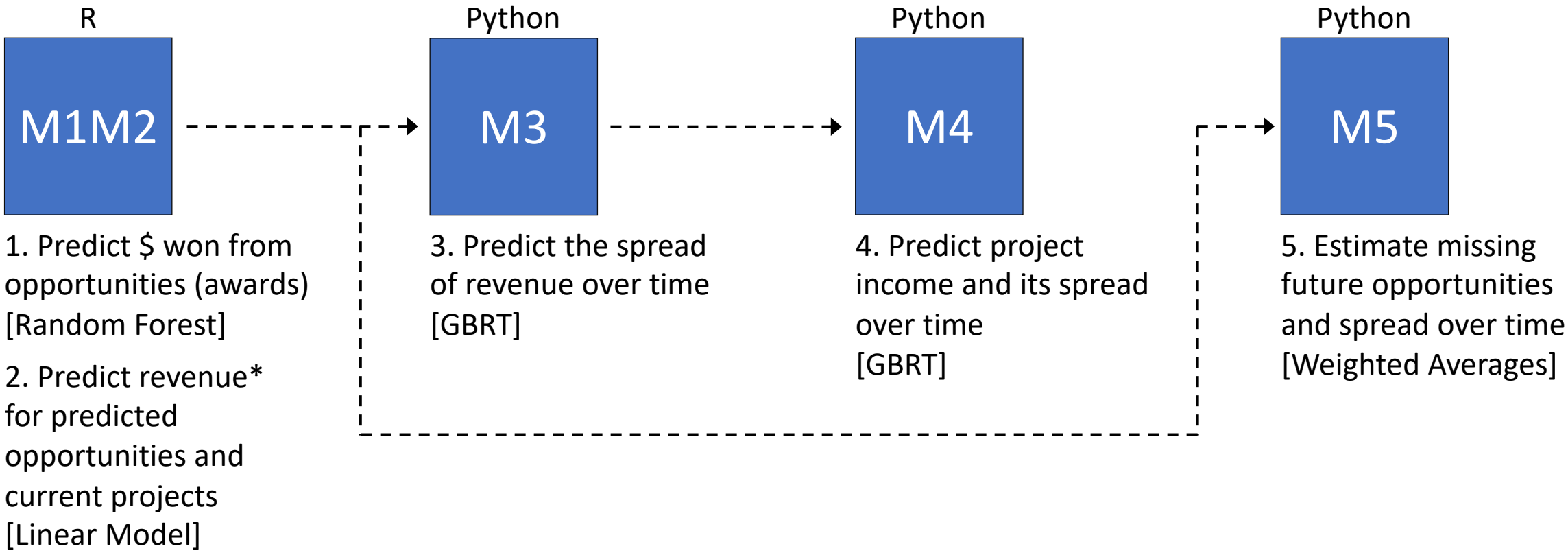
Key features:

- Automated daily updates
- Forecast models created using ML
- Multi-year forecasts
- User-friendly interface and reports

Benefits:

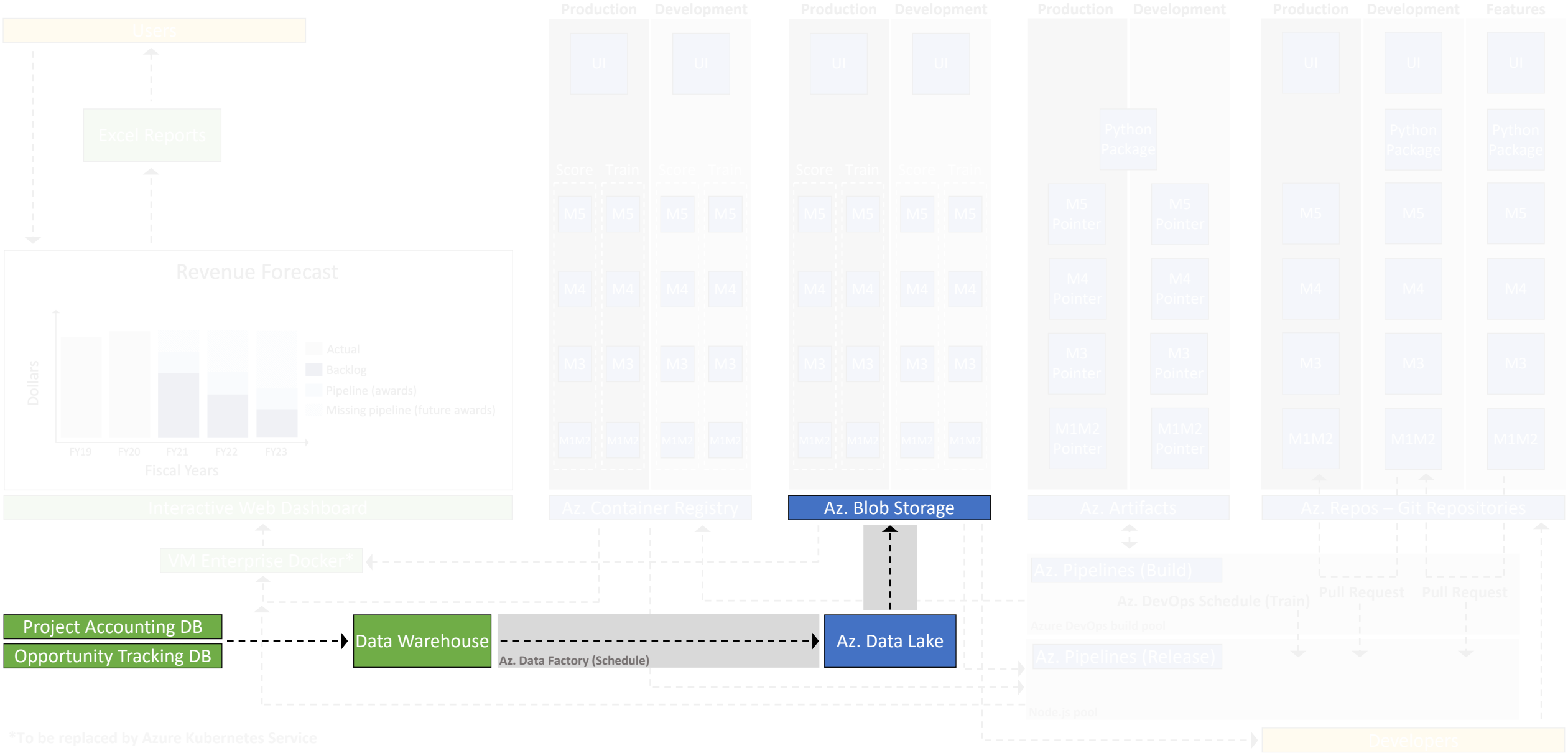
- Longer perspective
- Adaptive to data
- Codified methodology
- Assists financial analysts

3. Machine Learning Layers

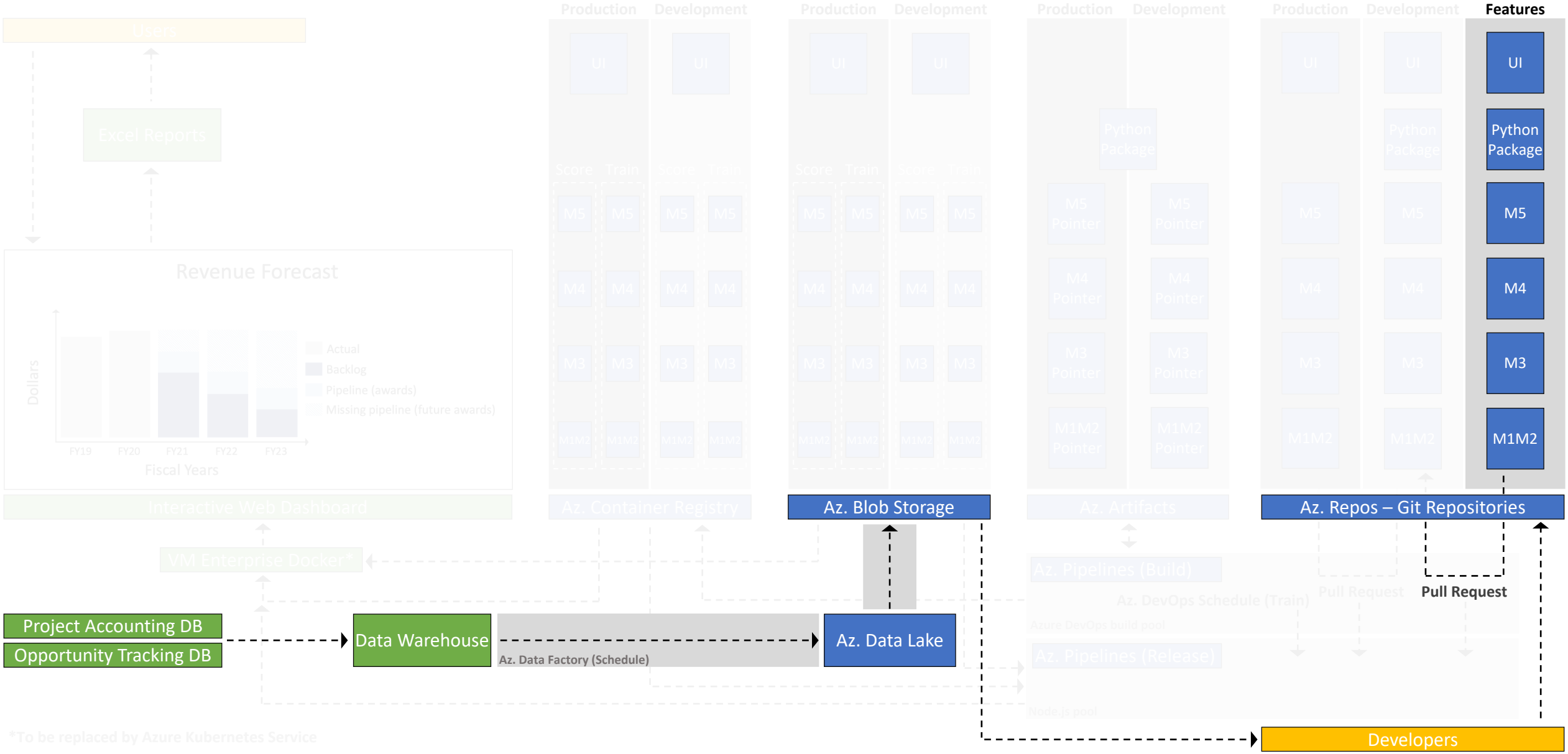


*After leakage

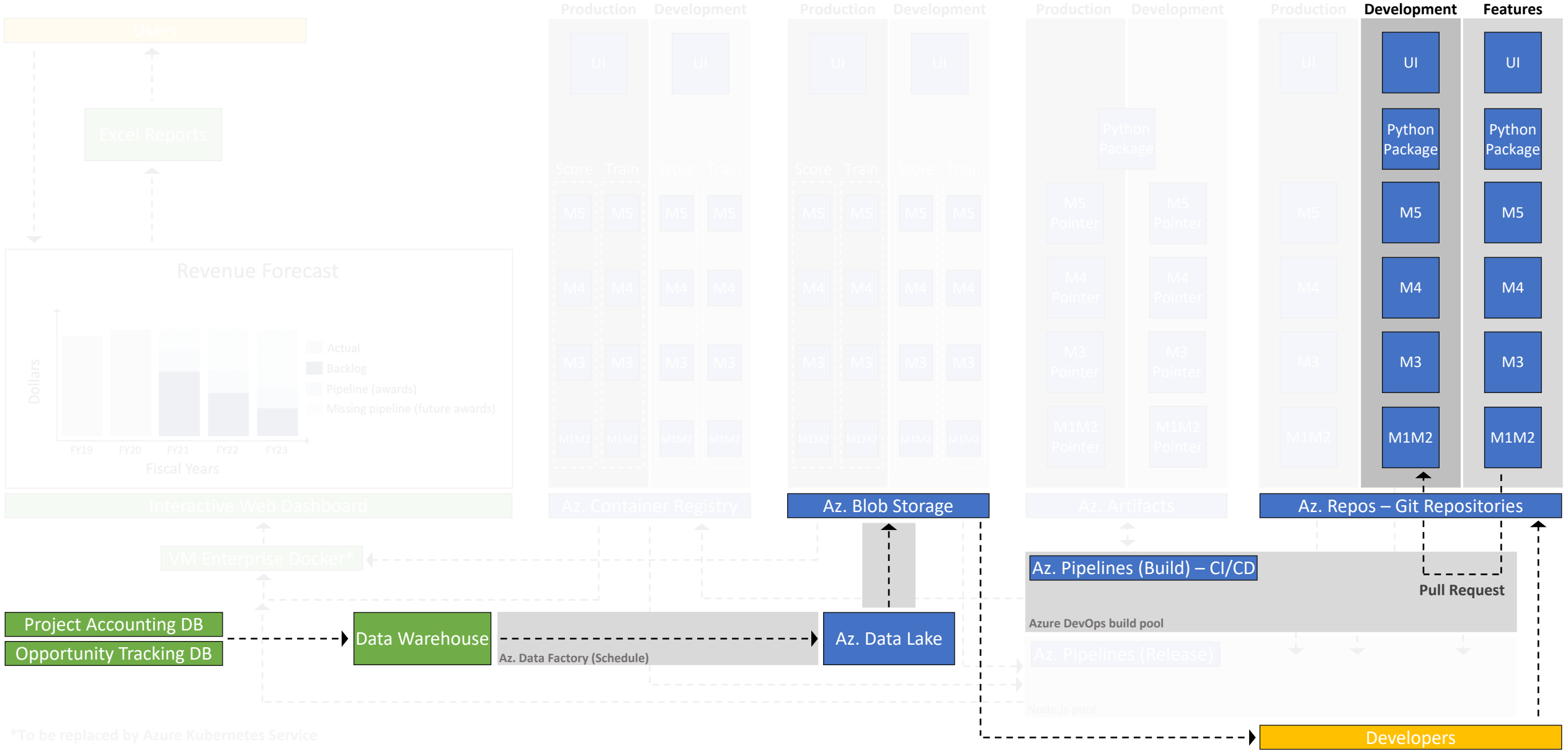
4.1. System Flow: Data Integration



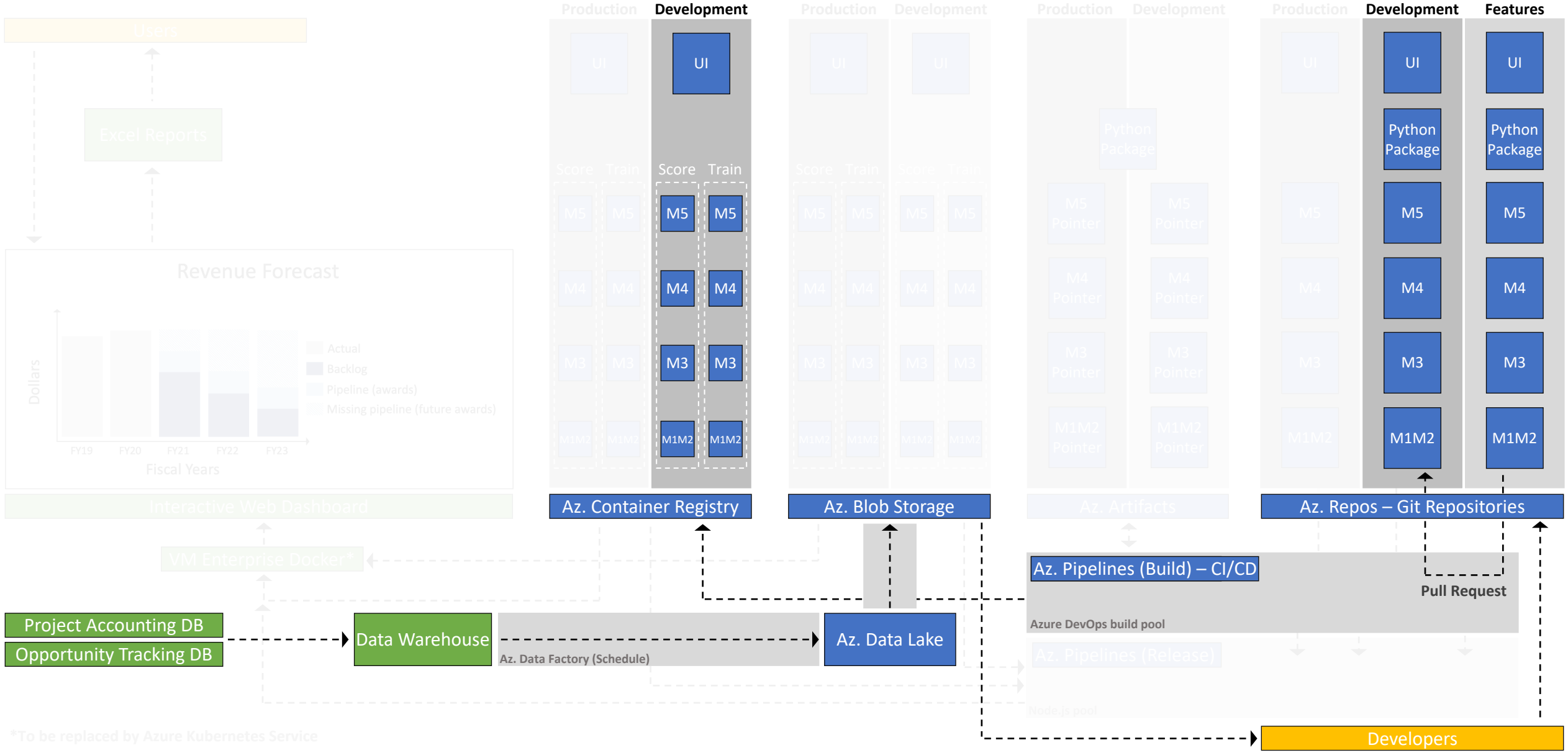
4.2. System Flow: Development-to-Deployment



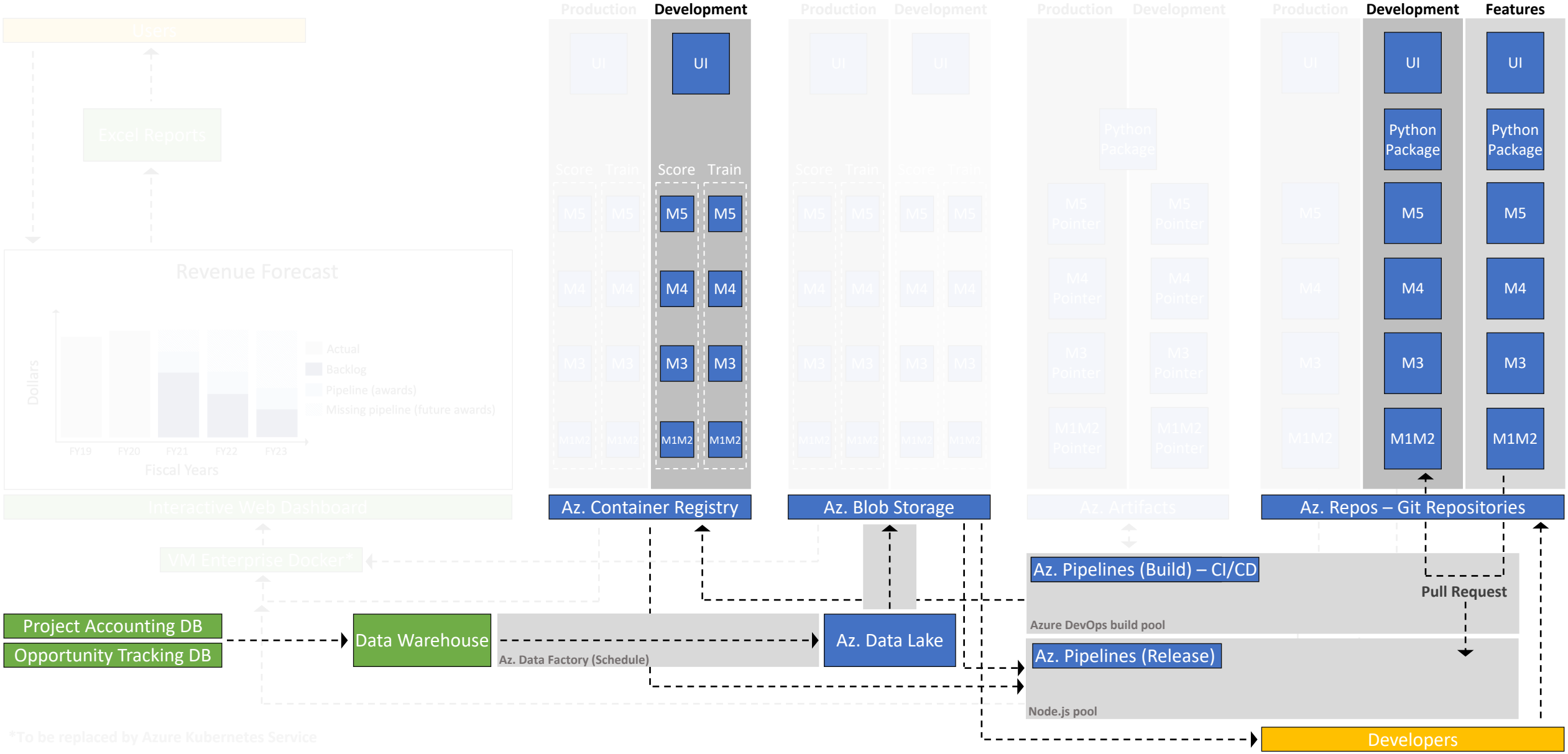
4.2. System Flow: Development-to-Deployment



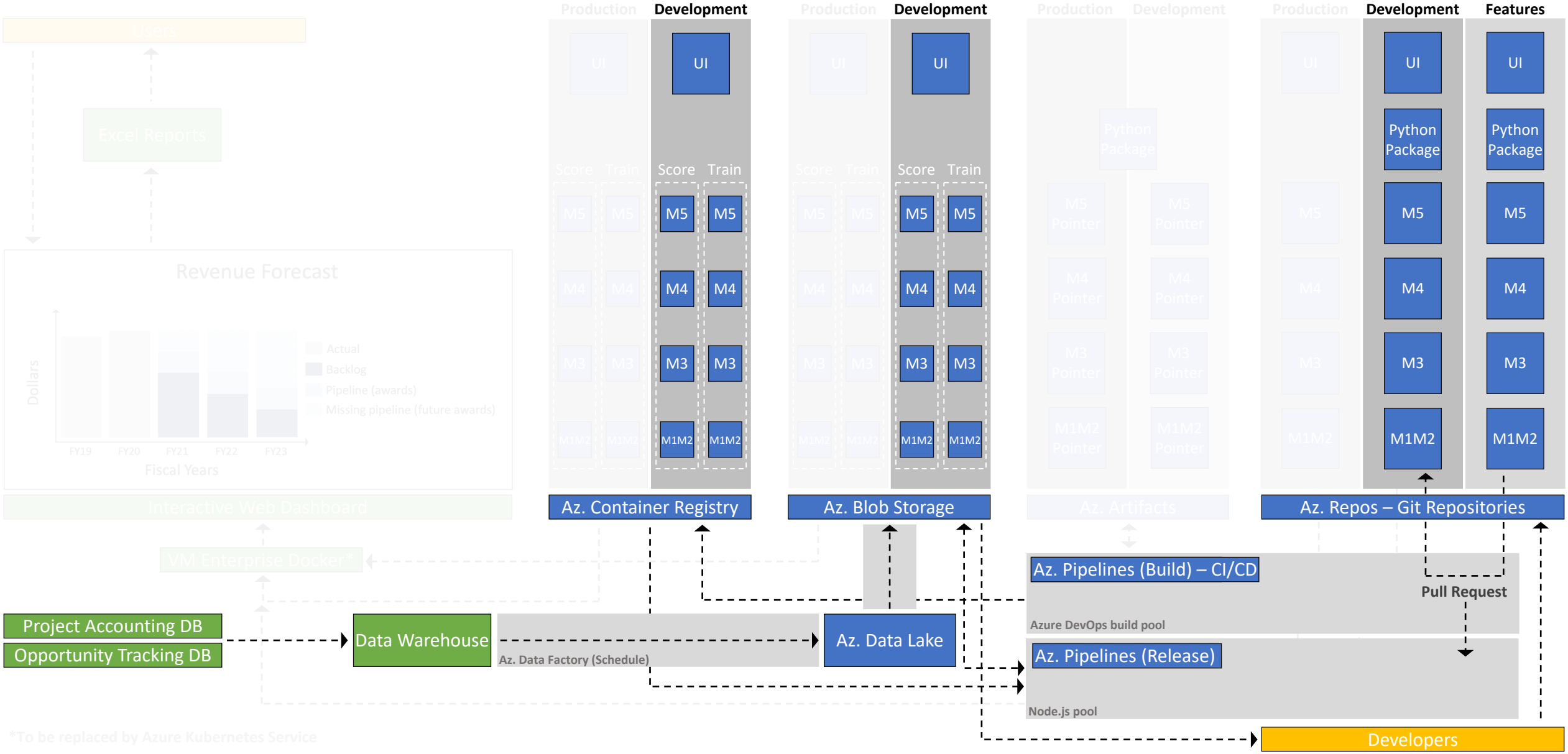
4.2. System Flow: Development-to-Deployment



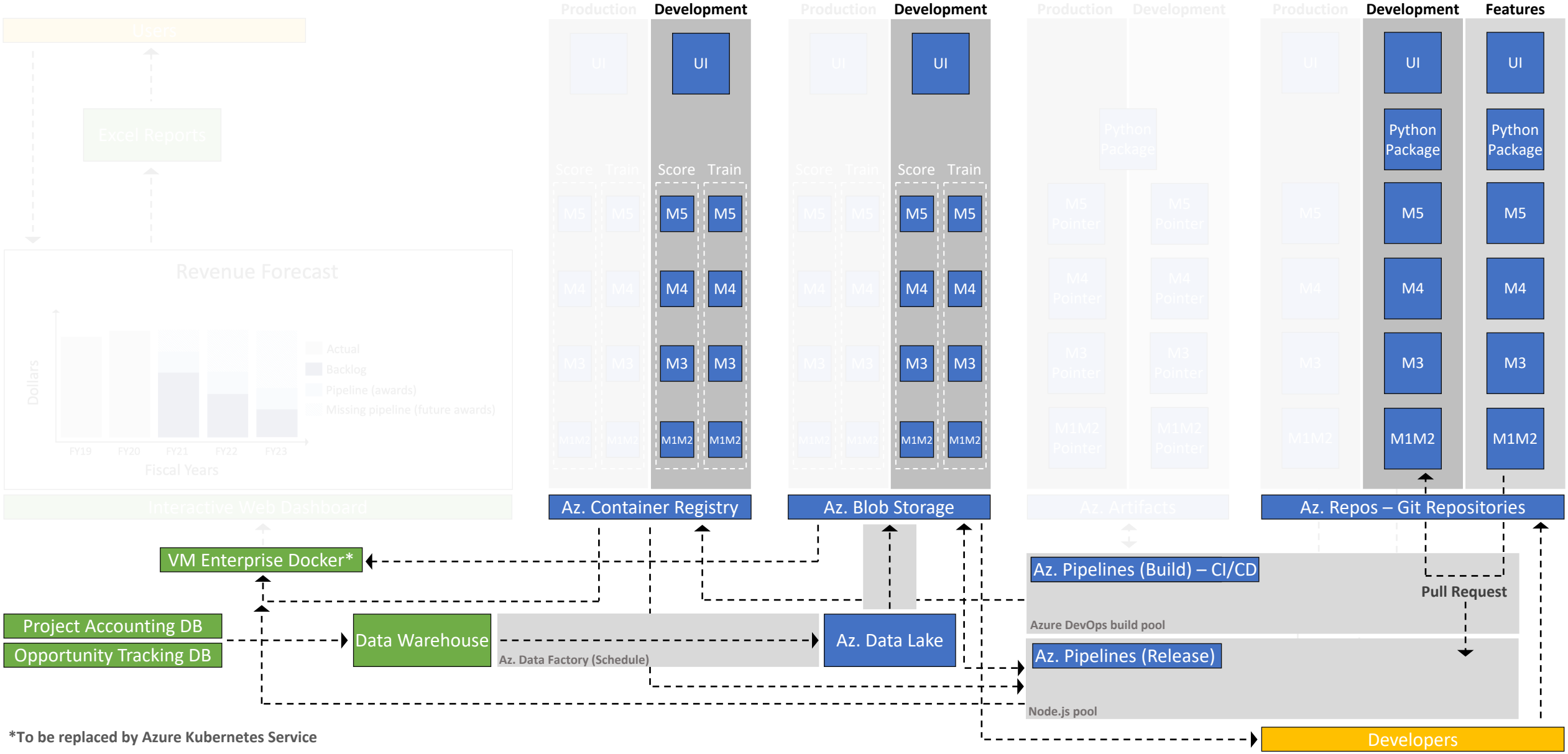
4.2. System Flow: Development-to-Deployment



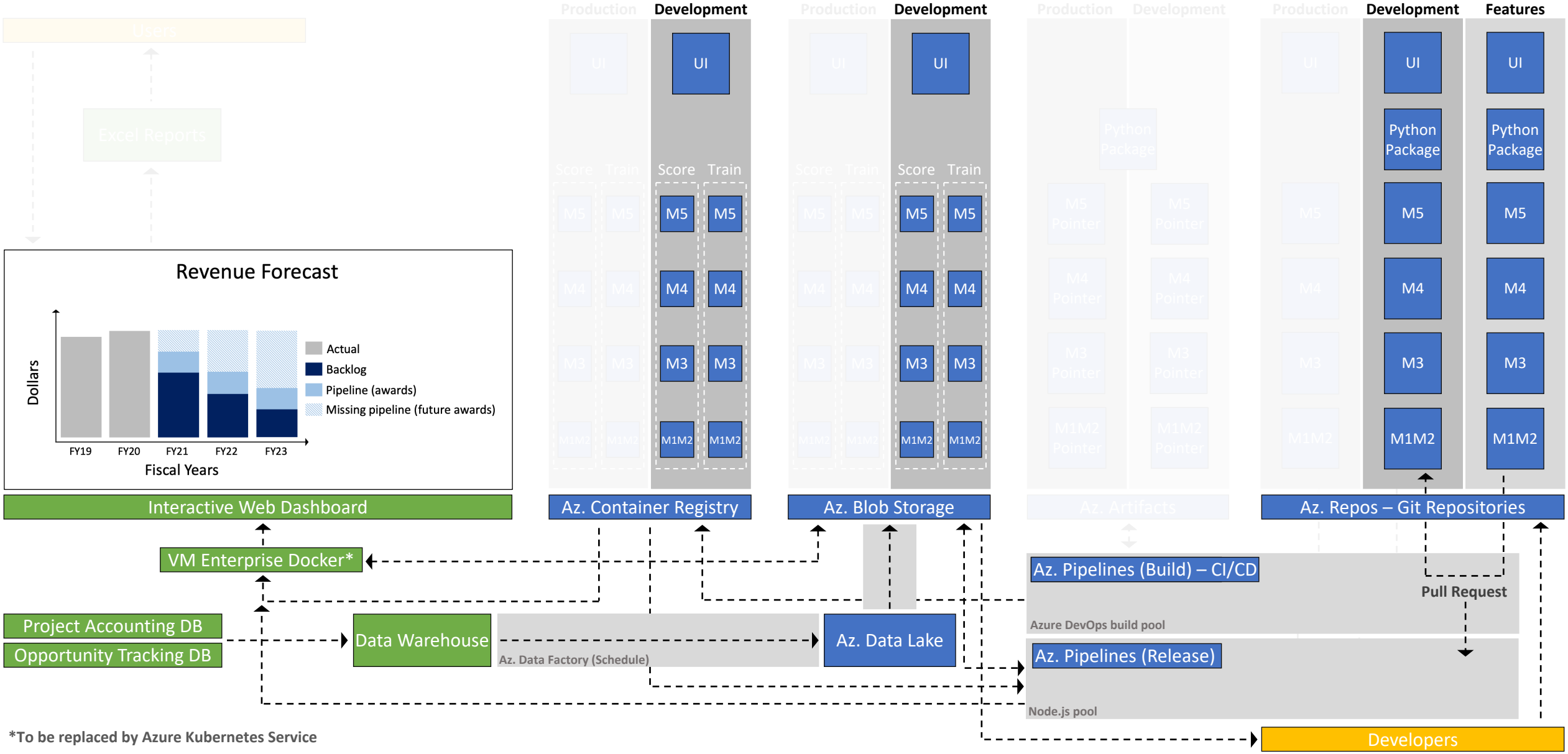
4.2. System Flow: Development-to-Deployment



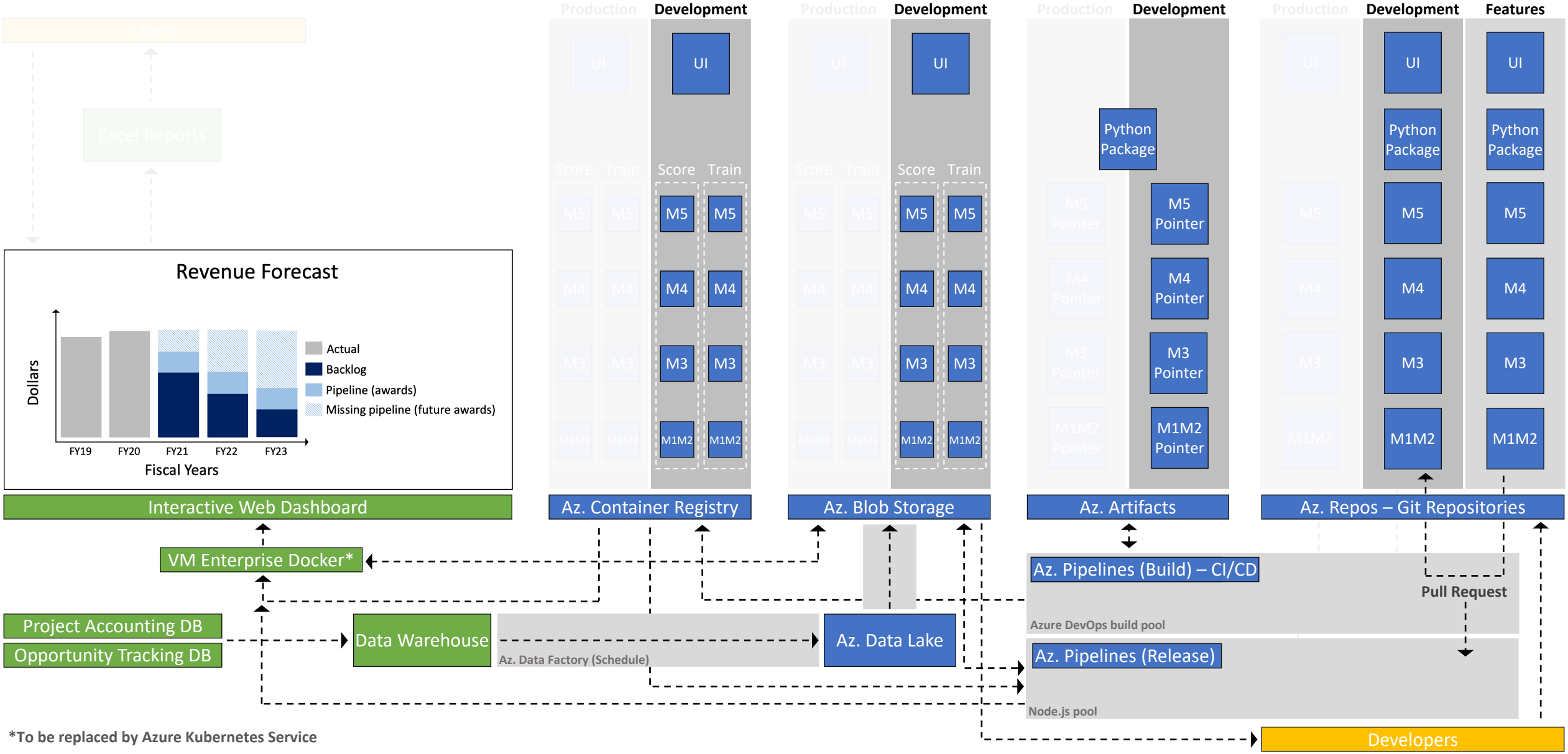
4.2. System Flow: Development-to-Deployment



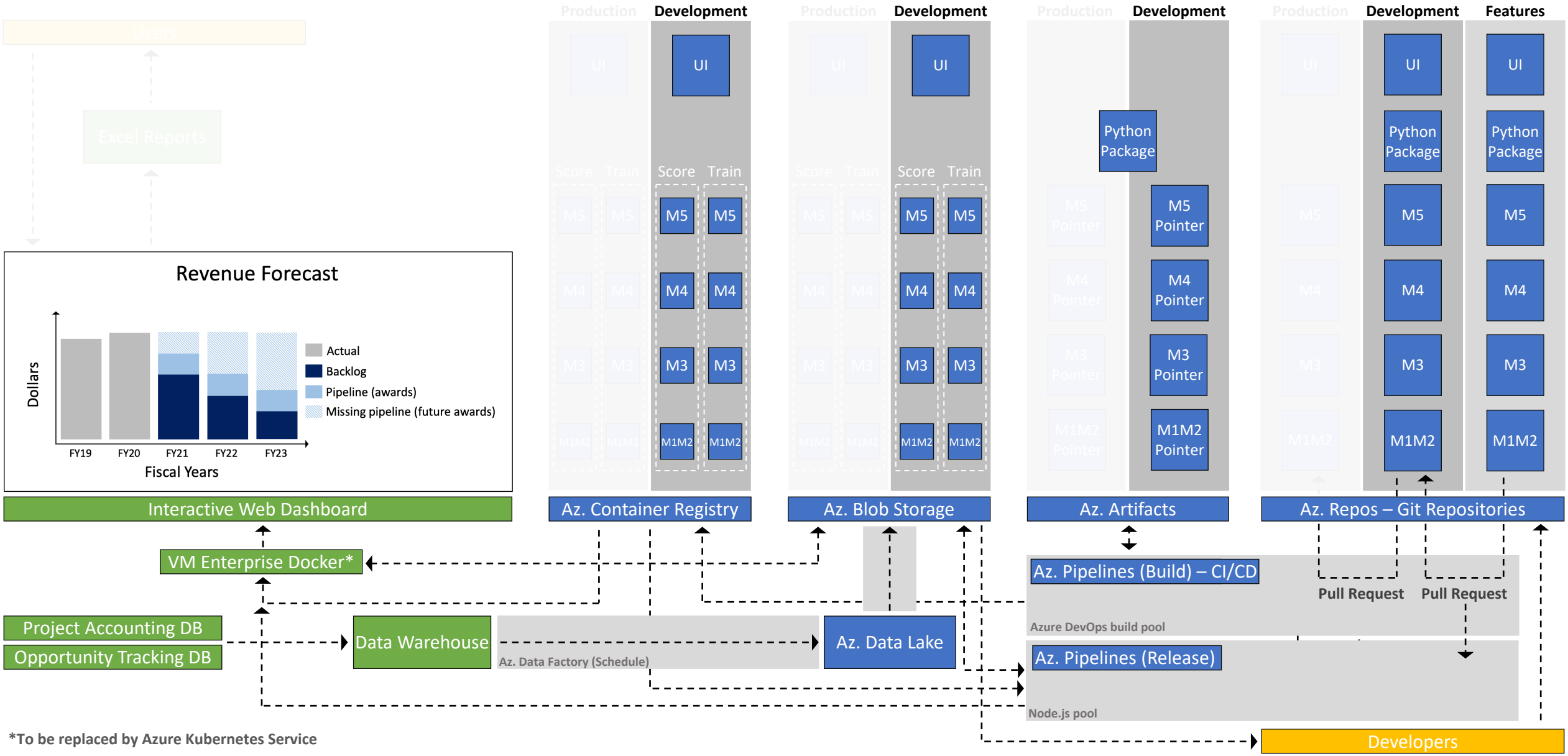
4.2. System Flow: Development-to-Deployment



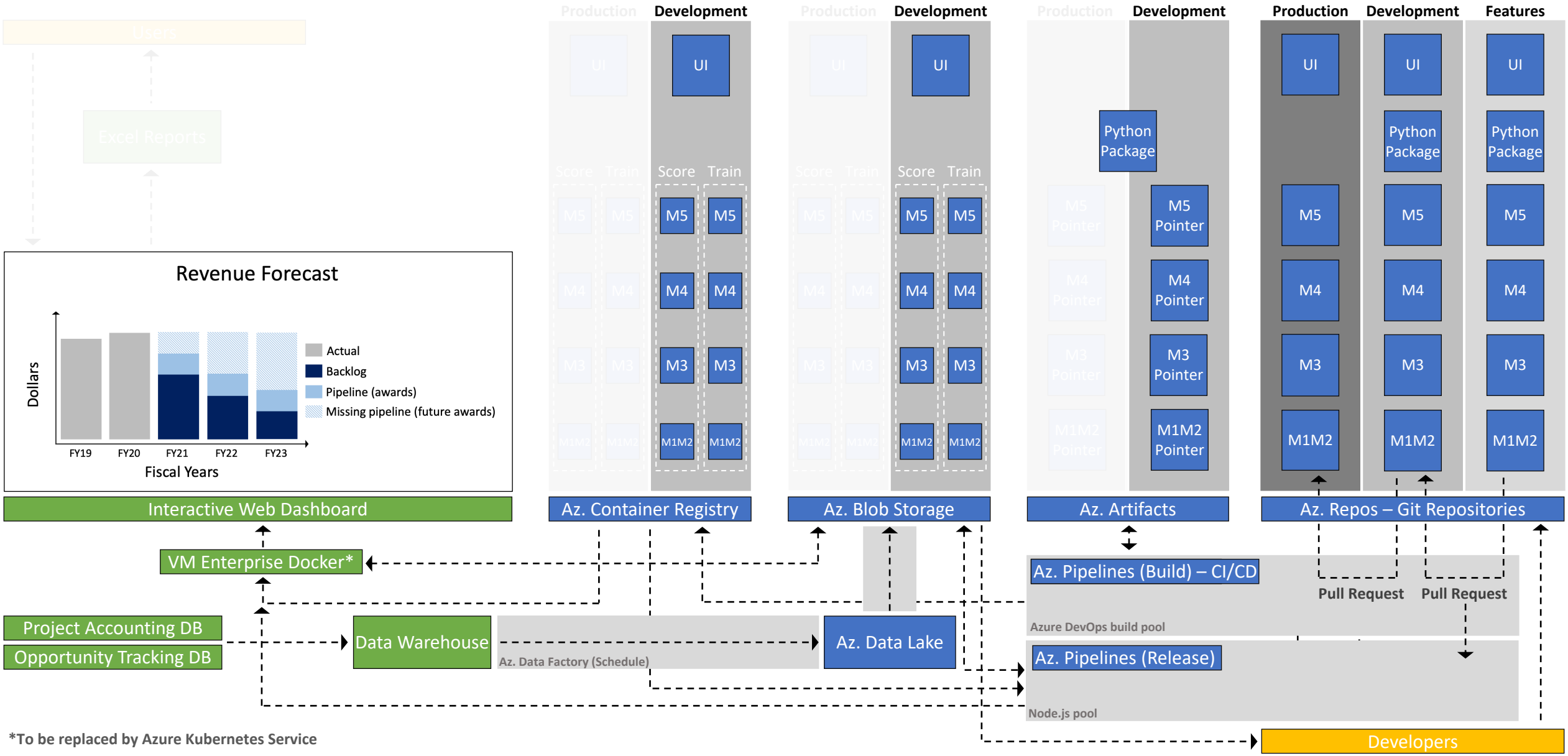
4.2. System Flow: Development-to-Deployment



4.2. System Flow: Development-to-Deployment

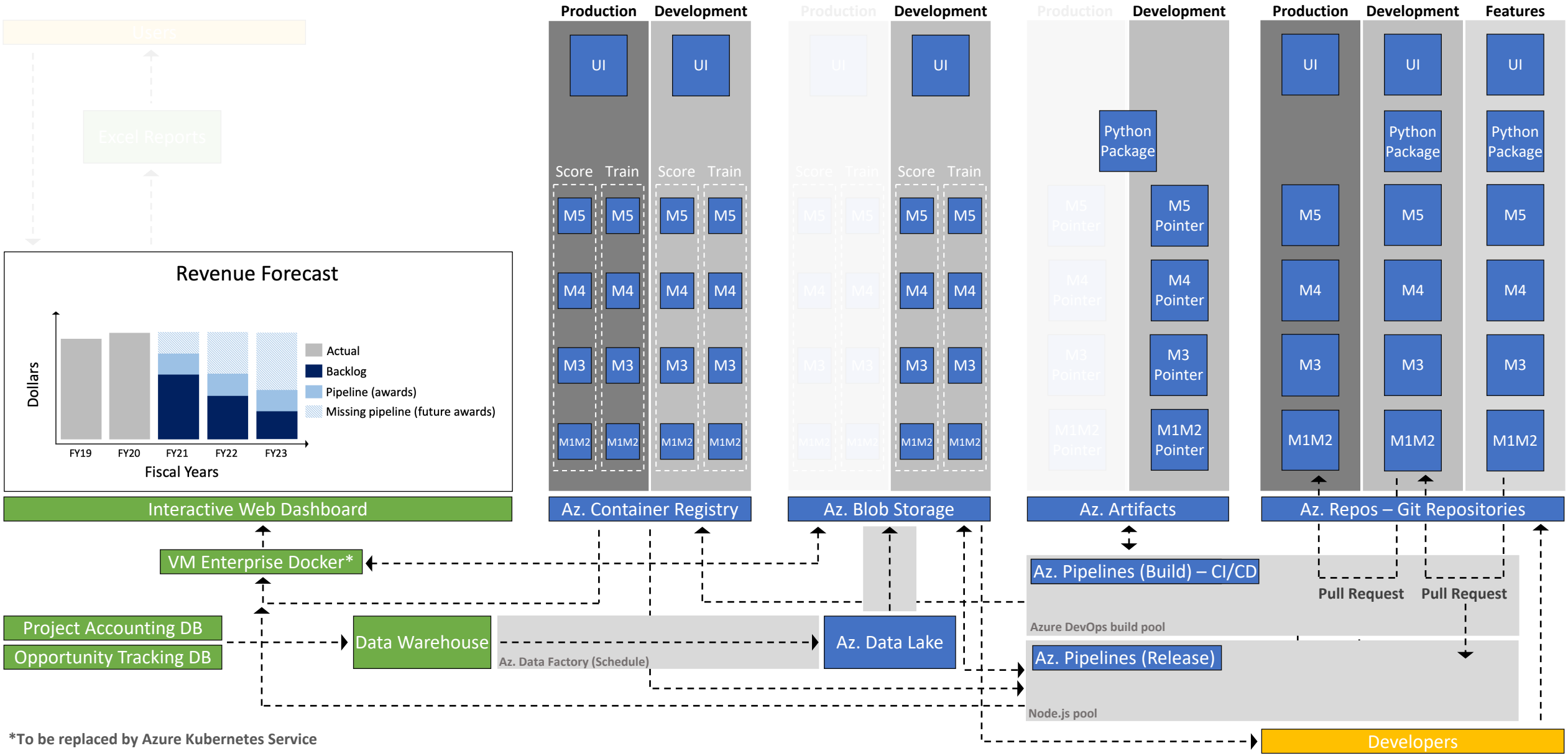


4.2. System Flow: Development-to-Deployment

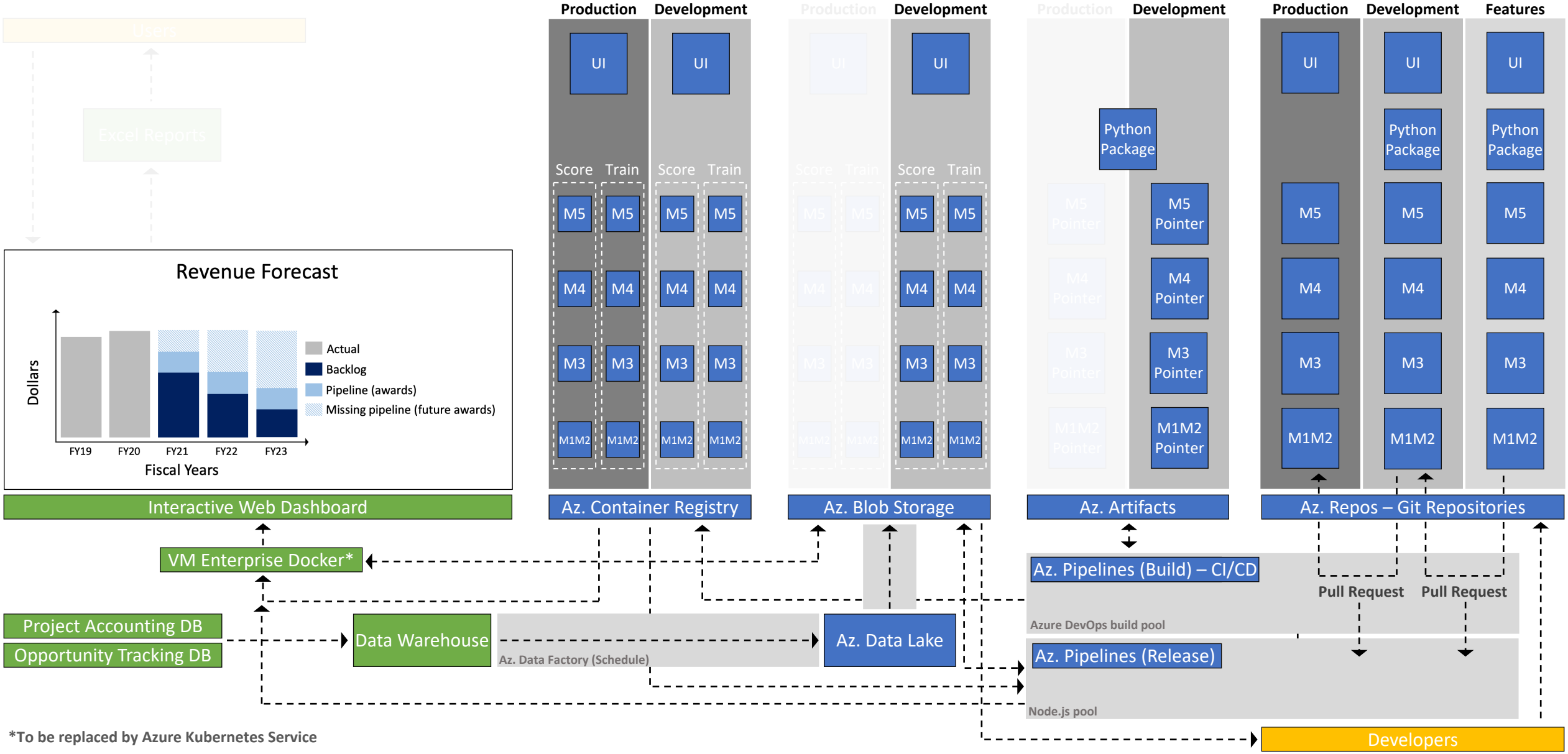


*To be replaced by Azure Kubernetes Service

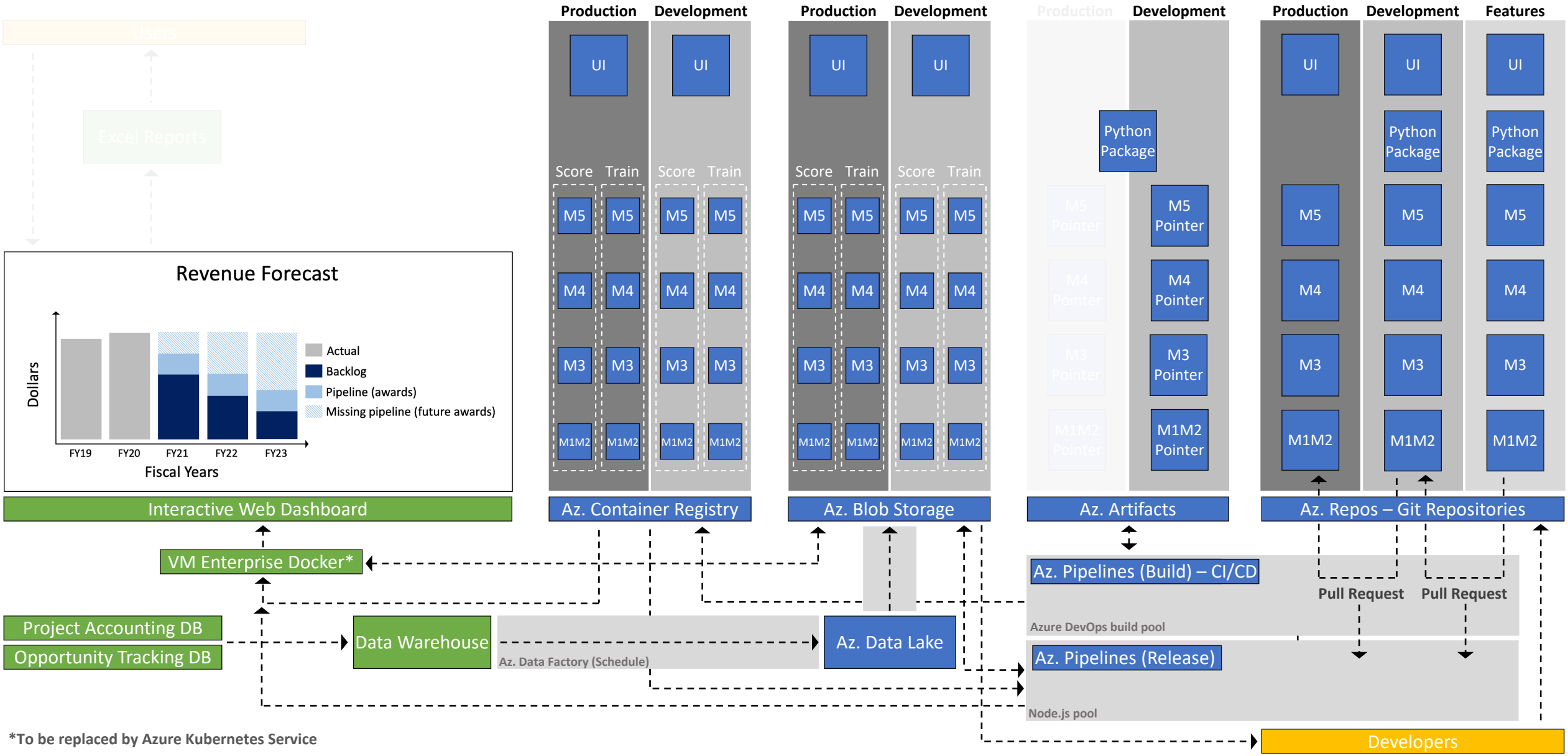
4.2. System Flow: Development-to-Deployment



4.2. System Flow: Development-to-Deployment

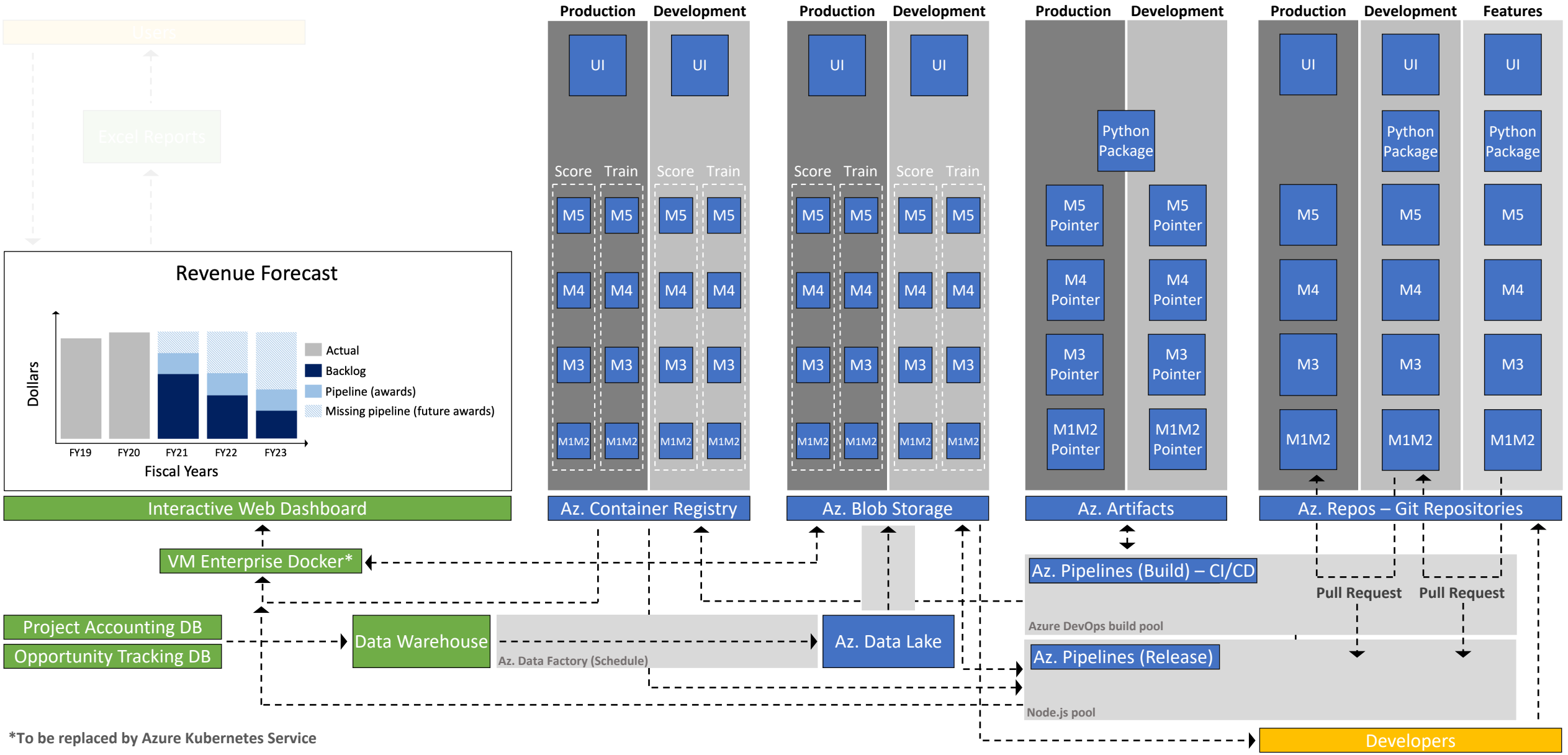


4.2. System Flow: Development-to-Deployment

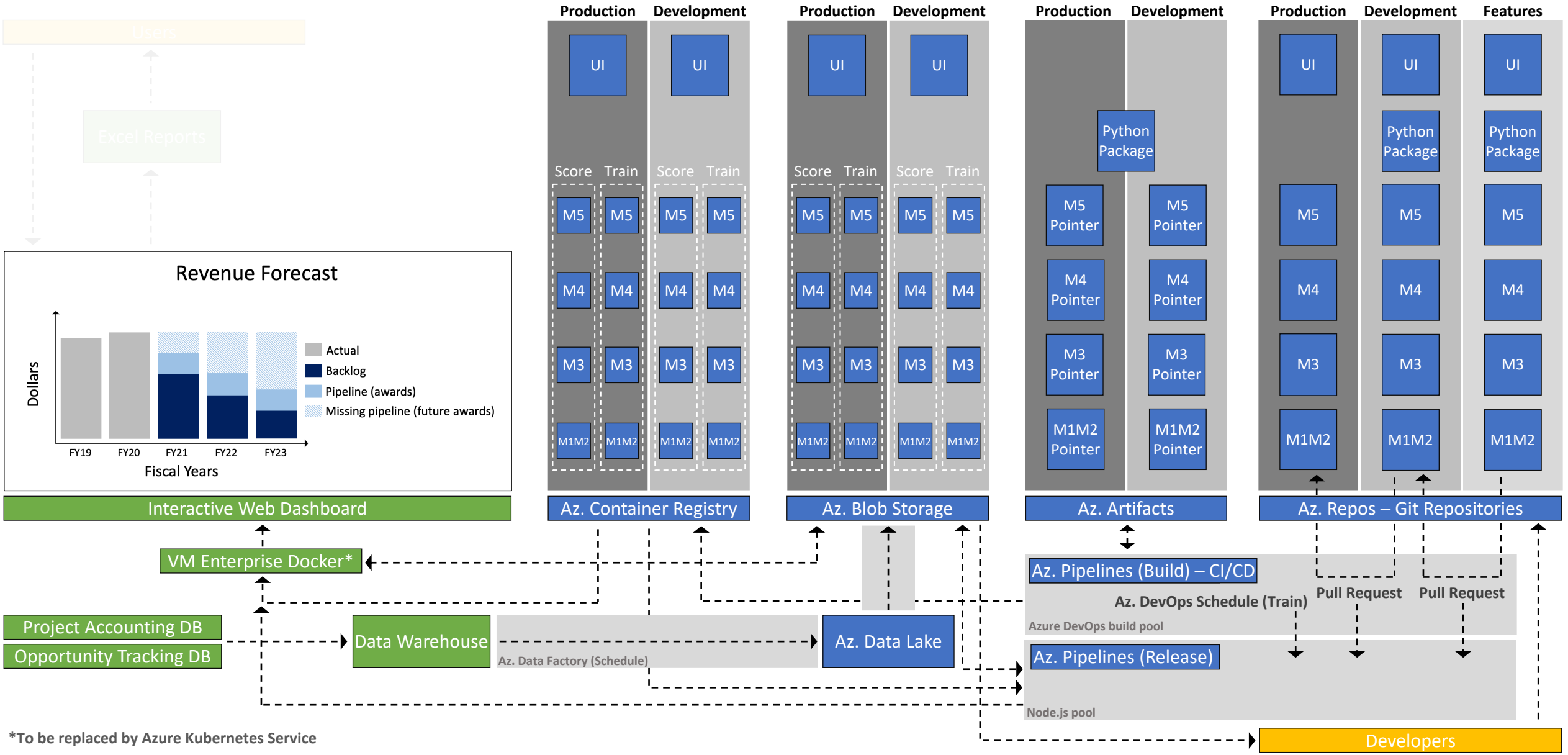


*To be replaced by Azure Kubernetes Service

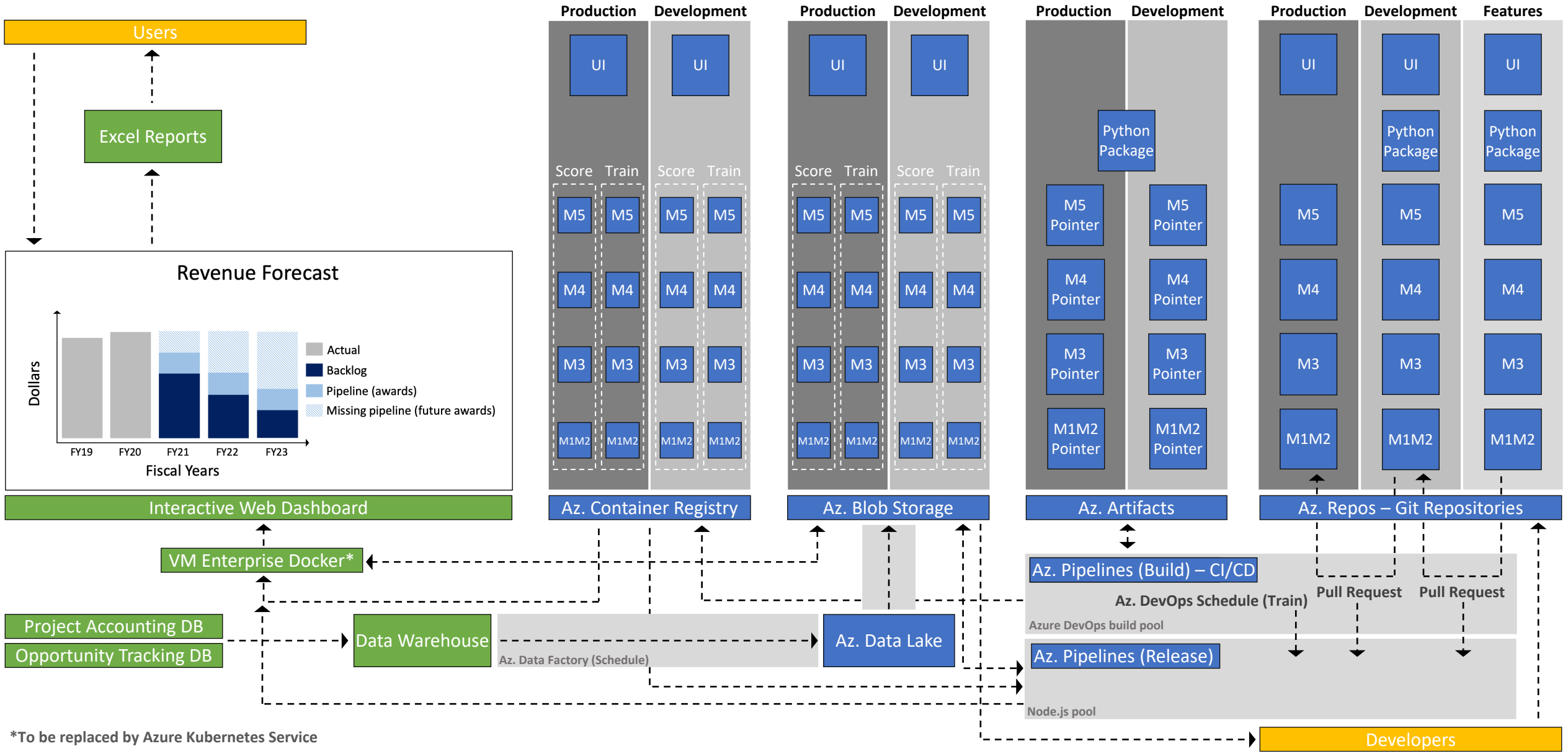
4.2. System Flow: Development-to-Deployment



4.2. System Flow: Development-to-Deployment



4.2. System Flow: Development-to-Deployment



*To be replaced by Azure Kubernetes Service

4.3. System Flow: Automated Training

Train & Score Pipelines

1. **Build – <u>Train</u> Container** (model n)
- Triggered by:
  - Development and Production branch merges.
- Performs:
  - Launch Docker build
  - Updates Azure Container Registry with new Dockerfile and tag
- Triggers: Build – AT Train (model n)

2. **Build – AT Train** (model n)
- Triggered by:
  - Schedule in Azure DevOps
  - Build – Train Container (model n)
- Performs:
  - Launch training container:
    - Runs preprocessing, training, evaluation
    - Writes artifacts to registry (Blob Storage)
  - Run curl POST:
    - Var_1, timestamp to publish
    - Var_2, job status (pass/fail)
    - Var_3, evaluation status (pass/fail)
    - Var_n, . . .
- Triggers: Build – AT Publish Pointer (model n)

4.3. System Flow: Automated Training

Train & Score Pipelines

1. **Build – Train Container** (model n)
- Triggered by:
  - Development and Production branch merges.
- Performs:
  - Launch Docker build
  - Updates Azure Container Registry with new Dockerfile and tag
- Triggers: Build – AT Train (model n)

2. **Build – AT Train** (model n)
- Triggered by:
  - Schedule in Azure DevOps
  - Build – Train Container (model n)
- Performs:
  - Launch training container:
    - Runs preprocessing, training, evaluation
    - Writes artifacts to registry (Blob Storage)
  - Run curl POST:
    - Var_1, timestamp to publish
    - Var_2, job status (pass/fail)
    - Var_3, evaluation status (pass/fail)
    - Var_n, . . .
- Triggers: Build – AT Publish Pointer (model n)

4.3. System Flow: Automated Training

Train & Score Pipelines

3. **Build – AT Publish Pointer** (model n)
- Triggered by:
  - Build – AT Train (model n)
- Performs:
  - Publish pointer in Azure Artifacts
    - Timestamp
    - Status variables
- Triggers**:** Build – Score Container (model n)

4. **Build – Score Container** (model n)
- Triggered by:
  - Build – AT Publish Pointer (model n)
  - Development and Production branch merges.
- Performs:
  - Launch Docker build
  - Updates Azure Container Registry
- Triggers: Release – Score (model n)

5. **Release – Score Container** (model n)
- Triggered by:
  - Build – Score (model n)
- Performs:
  - Runs scoring container
  - Updates bash script used for daily scoring

4.3. System Flow: Automated Training

Train & Score Pipelines

3. **Build – AT Publish Pointer** (model n)
- Triggered by:
  - Build – AT Train (model n)
- Performs:
  - Publish pointer in Azure Artifacts
    - Timestamp
    - Status variables
- Triggers**:** Build – Score Container (model n)

4. **Build – <u>Score</u> Container** (model n)
- Triggered by:
  - Build – AT Publish Pointer (model n)
  - Development and Production branch merges.
- Performs:
  - Launch Docker build
  - Updates Azure Container Registry
- Triggers: Release – Score (model n)

5. **Release – <u>Score</u> Container** (model n)
- Triggered by:
  - Build – Score (model n)
- Performs:
  - Runs scoring container
  - Updates bash script used for daily scoring

4.3. System Flow: Automated Training

Train & Score Pipelines

3. **Build – AT Publish Pointer** (model n)
- Triggered by:
  - Build – AT Train (model n)
- Performs:
  - Publish pointer in Azure Artifacts
    - Timestamp
    - Status variables
- Triggers**:** Build – Score Container (model n)

4. **Build – Score Container** (model n)
- Triggered by:
  - Build – AT Publish Pointer (model n)
  - Development and Production branch merges.
- Performs:
  - Launch Docker build
  - Updates Azure Container Registry
- Triggers: Release – Score (model n)

5. **Release – Score Container** (model n)
- Triggered by:
  - Build – Score (model n)
- Performs:
  - Runs scoring container
  - Updates bash script used for daily scoring

4.4. System Flow: Resource Summary

- **Azure Repos**
  - Stores repositories that contain:
    - Data processing and modeling code
    - Tests
    - Pipeline configuration files
    - Dockerfiles (Train and Score)
- **Blob Storage**
  - Store inputs and outputs to modeling pipeline
- **Azure Container Registry**
  - Store successfully built Docker container images
- **Azure Pipelines**
  - **Build**: Perform CI/CD tests, build Docker images, publish artifacts, run training containers
  - **Release**: Update model pipeline script on VM with up-to-date Docker tags
- **Azure Artifacts**
  - Store Python package
  - Store model artifact pointers

5. Conclusions

- Azure Machine Learning Cloud Environment?
  - “[E]nables you to track / version / audit / certify / re-use every asset in your ML lifecycle and provides orchestration services to streamline managing this lifecycle.” – github/microsoft/MLOps
  - Through the combination of Artifacts, Azure Blob Storage, and Python evaluation scripts, we can reproduce this. In our case, *faster to implement.*
  - Environment independence via containers and configuration files.
- Azure Python support good overall but documentation inconsistent.
- R support limited and more challenging to implement in production.

Thank you! Questions?

Thom Miano M.S., RTI International

tmiano@rti.org