Twin2Clouds: Cost-Aware Digital Twin Engineering and Deployment Across Federated Clouds

Philipp Gritsch[1], Deniz Pierer[1], Luca Berardinelli[2], Michael Felderer[34], Sashko Ristov[1]
[1]*University of Innsbruck*, Innsbruck, Austria, {philipp.gritsch, sashko.ristov}@uibk.ac.at, pierer.deniz@gmail.com
[2]*Johannes Kepler University Linz*, Linz, Austria, luca.berardinelli@jku.at
[3]*German Aerospace Center (DLR)*, Cologne, Germany, michael.felderer@dlr.de
[4]*University of Cologne*, Cologne, Germany, michael.felderer@uni-koeln.de

***Abstract*—Context: Digital Twins (DTs) undergo a rapid shift from monolithic cyber-physical systems to distributed, cloud-native stacks. While the elasticity of public cloud providers benefits this transition—opaque, provider-specific pricing renders economically viable deployment, especially across federated clouds, difficult.**

Objectives: This work aims to enable cost-aware deployment of Digital Twins across federated cloud providers by modeling DTs as multi-layered architectures and optimizing service selection per layer to minimize operational expenses.

Methods: We introduce Twin2Clouds, a cost-driven DT engineering framework that (1) organizes DT functionality into five cloud-oriented layers to help engineers map twin components to suitable cloud services, (2) offers cloud-agnostic *cost* and *pricing primitives* that capture heterogeneous pricing schemes across providers and services, and (3) integrates these elements in a cost-aware deployment model that predicts cloud costs and prescribes a reproducible deployment plan selecting the lowest-cost services for each layer.

Results: A quantitative evaluation of three real-world scenarios, ranging from a large smart building (30,000 devices) to a smart home (100 devices), demonstrates that Twin2Clouds consistently outperforms single-provider baselines. Depending on scale and workload, monthly costs decrease by an average of 24.6% and up to 79% compared with single-cloud alternatives.

Conclusion: Twin2Clouds equips DT engineers with a practical, vendor-neutral method for navigating today's complex cloud landscape and for realizing scalable, sustainable, and economically viable Digital Twins.

***Index Terms*—Digital Twin, Cloud Computing, Cost Optimization, Federated Clouds**

I. Introduction

Modern cyber-physical systems (CPS), ranging from smart homes to large factories, have exploded in scale and behavioral richness, making them challenging to comprehend without models that abstract, simulate, and predict system behavior. **Digital Twins (DTs)** embody this model-driven approach [1]: they maintain a continuously synchronized virtual counterpart of a physical asset, enabling closed-loop optimization, predictive maintenance, and what-if experimentation in many industries [2]. As DT workloads outgrow on-premises resources, they are increasingly offloaded to managed cloud services. Clouds offer elasticity and scalability, while hiding infrastructure details. However, they also fragment the design space with provider-specific pricing schemes, service bundling, and varying billing granularities. Some services bill strictly per use, whereas others impose minimum commitments or tiered plans, turning cost estimation and optimization into a non-trivial task. Deploying the whole twin within a single provider further risks vendor lock-in, constraining flexibility and negotiation power. Conversely, federated cloud [3] and Sky Computing [4] paradigms promise seamless cross-provider service selection, opening new avenues for optimizing cost, performance, and compliance. Engineering a cloud-native DT, therefore, revolves around finding the cost-optimal deployment across federated clouds.

To tame this complexity, the DT community has proposed conceptual models that separate key engineering concerns, such as monitoring, analytics, control, and planning, so each can be designed and validated independently [5]–[7]. Yet most of these models leave deployment choices—especially concerning cloud computing and cloud service selection—implicit. However, ignoring deployment choices, and thus cloud-pricing heterogeneity, can significantly inflate operating costs [8].

To bridge this gap, this paper presents **Twin2Clouds**, a cost-driven DT engineering framework, which enriches these conceptual models with a cloud-deployment dimension. Twin2Clouds maps every DT functionality to a dedicated deployment layer aligned with mainstream cloud services. Twin2Clouds formalizes *service cost and pricing primitives* to be used in a cost-aware deployment model, that predicts cloud costs including data-transfer charges, and prescribes a suitable deployment plan across federated clouds, comprising the most cost-efficient service and provider for each layer. The resulting layered deployment plan makes cloud-native DT engineering scalable and vendor-neutral, turning economic complexity into a tractable design decision.

We evaluate our Twin2Clouds approach on three representative scenarios: a large, medium, and small smart building deployments with 30,000, 4,000, and 100 IoT devices. Our results show that by leveraging Sky Computing principles and optimizing across federated clouds, the total deployment cost of the DT is consistently lower than that of single-provider deployments, even with the additional services required to migrate data across providers due to vendor lock-in. This demonstrates the importance of flexible, cost-aware deployment strategies when engineering scalable and sustainable DTs.

This paper makes the following key contributions to the

DOI 10.1109/MODELS-C68889.2025.00045

field of DT engineering:

1) *Cloud-oriented layered DT architecture*: We introduce a five-layer cloud-deployment abstraction that partitions DT functionalities around distinct cloud services. By mapping our layers to the conceptual framework by Eramo *et al.* [5], we connect established DT concepts to concrete cloud artifacts.
2) *Provider-agnostic cost- and pricing primitives*: We present a coherent set of cost- and pricing primitives that capture heterogeneous cloud pricing schemes and provide a consistent basis for estimating expenses across cloud providers.
3) *Cost-aware deployment model*: We integrate a predictive-prescriptive model that (1) predicts the cost of deploying a layered DT across federated clouds by estimating per-layer expenses and inter-layer data-transfer charges from user-specified workload parameters, and (2) prescribes a layer-wise deployment plan, that minimizes total cost by assigning each layer to its most economical provider.
4) *Empirical validation*: We evaluate Twin2Clouds using three DT scenarios of varying scale (large, medium, and small smart buildings). Our results show that federated, layer-wise deployment consistently reduces monthly cost by an average of 24.6 % compared to single-provider deployments.

By unifying conceptual DT models with deployment-centric abstractions, the publicly available Twin2Clouds framework [1] turns the economic complexity of today's cloud landscape into an optimization opportunity rather than an obstacle.

The remainder of the paper is organized as follows: Section II revisits existing DT research and related work and Section III motivates our cloud-oriented abstraction. Sections IV and V detail the layered architecture and present our formal service cost and pricing primitives, while Section VI describes the implementation and evaluates our approach on three realistic use-case scenarios. Finally, Sections VII–VIII discuss validity, limitations, conclusion, and future work.

II. Background and Related Work

This section reviews prior work related to the study: (1) foundational, technology-agnostic definitions of DTs, (2) architectural models that organize a DT's functionality, and (3) research on cloud-based deployment and cost-optimization of production-grade twins.

DTs have evolved from one-off simulation artifacts into long-lived CPS composed of many interconnected services that often run in the cloud [9]–[11]. While Gartner predicts that by 2027 at least 40 % of enterprises will operate a production-DT [12], most published work, such as [13]–[15], remains at the level of abstract concepts and offers little guidance on how a twin is modeled, deployed, and operated at scale [16].

Conceptual Definitions. Following the definition by Kritzinger [17], a DT is a virtual replica of a physical object, referred to as the *physical twin*, with continuous bi-directional synchronization between them. This concept expands on *digital models*—where data exchange is manual—and *digital shadows*, which automatically receive data from the physical twin but lack reverse communication. In the conceptual, technology-agnostic framework of Eramo *et al.* [5], digital models are classified as: i) descriptive models, which capture the asset's current state, ii) predictive models that project its future behavior, and iii) prescriptive models that prescribe corrective actions. These models operate within a continuous feedback loop in which data are acquired and analyzed to generate plans, and the resulting commands are executed on the physical twin, thus closing the cycle. ***Paper positioning:*** We introduce a cost-aware deployment model to predict the cost and prescribe a suitable DT deployment plan (Contribution 3).

DT Architecture and Quality Attributes. A systematic study by Ferko *et al.* [9], comprising 140 peer-reviewed papers on software architectures and quality attributes for DTs, reveals that the majority of DTs utilize a combination of layered architecture and the service-oriented architecture (SOA) pattern. While maintainability, performance efficiency, and compatibility are commonly addressed quality attributes, deployment costs remain largely unexplored in the current literature. Works, such as Tuhaise *et al.* [18] and Qiuchen *et al.* [19], define a five-layer architecture for campus-scale twins, comprising data acquisition, data/model integration, digital modeling, and visualization. However, they do not discuss where each functional layer should run or how much it will cost to use the twin in a cloud setup. Talasila *et al.* [20] emphasize different structuring principles, by organizing DTs around reusable assets and life-cycle management phases, whereas Twin2Clouds introduces deployment-oriented layers aligned with cloud service categories. While the former focuses on asset re-use and the latter on cost-aware deployment, the two are complementary. ***Paper positioning:*** In line with these works, our cloud deployment layers follow a similar separation-of-concern principle, connecting established DT concepts to concrete cloud artifacts (Contribution 1).

Cloud Deployment and Cost Optimization. Prior work on cloud optimization spans multi-provider decision support [21], cost-efficient distribution [22], and stochastic provisioning [23]. Closer to DTs, surveys such as Siriweera and Narusa [24] show that cloud platforms are a suitable environment for DTs, giving engineers access to opaquely managed, highly scalable, and more resilient services. In our previous paper [25], we introduced a goal-driven approach to minimize the cloud costs by reducing the data transfer to cloud. Complementary to our previous work, Twin2Clouds minimizes the costs by selecting the appropriate cloud provider for the needed cloud service. Zhang *et al.* [26] optimize the placement and migration of DT artifacts in a Mobile-Edge-Computing network, modeling delay, bandwidth, and compute costs under the assumption of uniform pricing. Schroeder *et al.* [16] present a model-driven engineering workflow that guides practitioners from a platform-agnostic reference architecture to an automatically generated, deployable web service. While reducing modeling effort and functional reuse, they

[1]available at https://cloud-dts.github.io/twin2clouds/

do not incorporate runtime economics or user-specified usage parameters. ***Paper positioning**:* We complement these efforts by integrating unified, provider-agnostic cost and pricing primitives, which capture the nuances of heterogeneous cloud pricing schemes and supply a robust foundation for estimating operational costs of cloud-native DTs (Contribution 2).

To our knowledge, this paper is the first to offer a DT engineering solution that combines a layered DT architecture, including cloud-native cost and pricing primitives, yielding an extensible and modular blueprint for engineering production-grade DT at scale.

III. Motivating Study

This section highlights the key challenges engineers face when deploying DTs in public clouds: the complexity of cloud pricing schemes, as well as the wide variety of functionally equivalent services across providers.

A. Cost-driven Motivation: IoT Pricing Differences

To illustrate how deployment choices drive cost, we examine the data ingestion component that moves sensor data transfer into the twin (i.e., *Data Acquisition*). AWS IoT Core and Azure IoT Hub both handle message intake and distribution; however, their billing models yield notably different outcomes. Table I compares costs for a facility of $10{,}000$ devices sending $0.5\,\mathrm{kB}$ messages under two realistic workloads: one in which each message is immediately forwarded to the cloud, and another in which messages are buffered at the edge and sent in larger batches of $12\,\mathrm{kB}$. Under modest message rates, AWS's straightforward per-message pricing proves more economical, whereas under sustained high-throughput conditions, Azure's tiered model yields lower overall cost. This divergence makes it clear that each component incurs distinct, provider-specific costs and must be evaluated independently.

B. Pricing Divergence Across Other Layers

Pricing divergence extends well beyond Data Acquisition. Take storage as an example: Table II shows that AWS DynamoDB bills on-demand for each read and write, whereas Azure CosmosDB charges based on provisioned throughput, which may lead to increased costs due to over-provisioning or throttling under varying workloads. Furthermore, AWS IoT TwinMaker pricing depends on the number of modeled entities and query executions. In contrast, Azure DTs uses a message- and operation-based billing model, making them suitable for different modeling and interaction intensities. Finally, AWS Managed Grafana is priced solely per active user, offering predictable costs for small teams, while Azure combines per-instance infrastructure costs with per-user charges, which can prove more cost-effective at larger scales.

C. Summary

As DTs evolve into cloud-native, data-intensive systems, deployment cost and flexibility become critical considerations. The comparative analysis of AWS and Azure demonstrates that the economic footprint of a cloud-hosted DT cannot be expressed by a single scalar "cloud price". Even for data acquisition services, the two providers apply fundamentally different cost and pricing structures, yielding opposite cost optima under bursty versus sustained workloads. Divergence increases at higher tiers: storage, state management, model execution, and visualization are metered by distinct units and discount functions that vary by provider.

These examples underscore the importance of evaluating the cost of each component independently. The formal breakdown into cost and pricing primitives and the mapping of service domains and components to our deployment-oriented layered architecture is discussed in the following Section IV.

IV. Twin2Clouds Cloud-Oriented Layered DT Architecture

Research on DTs has produced standards (e.g., [6], [7]) and conceptual frameworks (e.g., [5]) that explain the functionalities of a twin. Yet, when engineers face a cloud deployment, they discover that the service portfolios of major providers, such as AWS and Azure, do not line up one-for-one with the conceptual functionalities. IoT brokers, stream processors, time-series databases, knowledge graphs, visual analytics workspaces, pricing tiers, events, and egress rules—all cut across the neat boundaries of the conceptual view.

To bridge this gap, Twin2Clouds introduces a cloud-oriented layered DT architecture with the following five layers:

1) *Data Acquisition*: Sensor and device data ingestion via IoT platforms;
2) *Data Storage*: Hot, warm, and cold data storage for operational and historical data;
3) *Data Processing*: Transformation, filtering, and event processing;
4) *DT Management*: Core DT services, including entity graphs, relationships, and synchronization; and
5) *Visualization*: Presentation layer for analytics and monitoring, typically through dashboards or web UIs.

Figure 1 presents the Twin2Clouds 5-layer DT architecture. For clarity, it also lists examples of cloud services offered by AWS and Azure for each DT layer. We use these two cloud providers as an example because they are the leaders in Gartner's Magic Quadrant for Cloud Platform Services [27]. These layers reflect common deployment patterns observed in both AWS and Azure. Each layer collects cloud services that share a common technical role, allowing services within a layer to be swapped or combined without disrupting other layers, thereby enabling modular reuse and separation of concerns. Moreover, federated deployments can mix providers layer-wise, choosing the cheapest or most compliant service without redesigning the whole twin. Finally, we link these layers back to the foundational DT model of Eramo *et al.* [5], introduced in Section II.

A. Layer 1—Data Acquisition

Layer 1, the *Data Acquisition layer*, is the point where the physical system first intersects the cloud. Its purpose is to move telemetry from edge devices, such as temperature

TABLE I: Cost comparison of Data Acquisition services

Scenario	Devices	Msg Size	Freq.	Daily Msgs	AWS ($/day)	Azure ($/day)	Cheaper
High Frequency	10,000	0.5 KB	1/5 s	172.8B	197.3	83.3	Azure (2.33 ×)
Low Frequency	10,000	12KB	1/2 min	7.2M	29.2	33.3	AWS (1.14 ×)

TABLE II: Examples of pricing model differences across layers

Layer	AWS Service	Azure Service	Pricing Model Difference
Storage	DynamoDB	Cosmos DB	On-demand R/W units vs. provisioned RU/s
DT Management	TwinMaker	Digital Twins	Per-entity/query vs. per-message + provisioned operation/message
Visualization	Managed Grafana	Managed Grafana	Per-user (AWS) vs. instance + user (Azure)

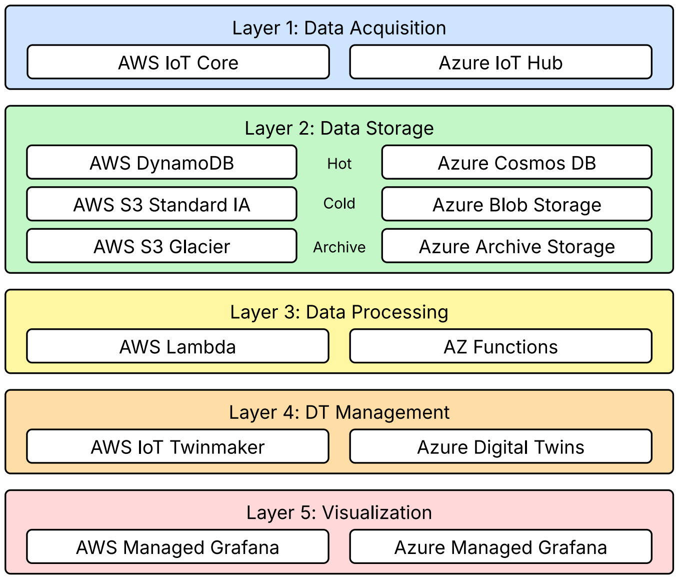


Fig. 1: Twin2Clouds' 5-layer DT architecture and examples of the corresponding AWS/Azure services

probes or power meters into the provider's backbone with as little latency and overhead as possible while enforcing rigorous security. In practice, this means devices speak a lightweight publish/subscribe protocol (e.g., MQTT). The cloud-side broker manages those connections, authenticates every certificate or token, and applies routing rules that decide whether a payload is streamed into analytics, parked in a hot store, or fanned out to alerting functions.

Because the provider manages the broker, a great deal of governance is built in. Identity registries keep track of each device, access-control policies restrict who can publish or subscribe, and the system records immutable audit logs for compliance teams. With AWS, the relevant service bundle is IoT Core, whose rules engine can forward messages directly to data-analytic services (e.g., Kinesis), object store (e.g., S3), or processing services (e.g., Lambda). Azure offers the analogous IoT Hub for telemetry, which integrates with Event Grid for routing and the Stream Analytics engine. Both services scale to millions of devices, and users primarily pay for message volume, as well as the capacity tier they select in Azure's case.

By encapsulating these concerns in a dedicated layer, we decouple edge connectivity from all subsequent elements. Engineers are free to optimize throughput, swap one broker for another, or even mix providers at the edge without disturbing the processing, storage, and management logic upstream.

B. Layer 2—Data Storage

This layer is responsible for maintaining accessibility, availability, and durability of data and models. However, cloud storage is one of the main cost contributors in DT operations, so efficient engineering is essential. We adopt a tiered-storage model, separating storage by access frequency and latency requirements. Correct tiering is crucial for controlling operational cost, as demonstrated in our evaluation (see Section VI).

Hot Storage. Recent data is accessed frequently for tasks like analytics, simulation, and monitoring. It should be stored in low-latency, high-throughput systems such as key-value or document stores to enable near real-time access. Example services for hot storage include *AWS DynamoDB* (a fully managed NoSQL key-value and document database) and *Azure Cosmos DB* (a globally distributed multi-model database service). DynamoDB offers on-demand or provisioned throughput pricing, while Cosmos DB charges based on request-unit throughput (RU/s), which varies with operation complexity and consistency guarantees.

Cold Storage. Older data used for trend analysis or batch processing is accessed less frequently. It can be moved to cheaper storage classes optimized for infrequent access. Cold storage (e.g. *AWS S3 Standard-IA* (Infrequent Access) and *Azure Blob Storage – Cool tier*) reduces cost but introduces higher retrieval latency and fees. Both services offer significantly lower storage costs per GB than hot tiers, but impose higher access and early deletion costs.

Archive Storage. When data is no longer needed for day-to-day or even monthly operations but must be preserved (e.g., for compliance, forensic analysis, or future reconstruction of the DT), it is moved to archive storage. This tier offers very low storage costs but imposes substantial penalties on access time and retrieval cost. Example services for archive storage include *AWS S3 Glacier* and *Azure Blob Storage – Archive tier*. These services are optimized for long-term retention at minimal cost, with retrieval times ranging from minutes to hours depending on the retrieval mode selected.

C. Layer 3—Data Processing

After telemetry has been durably captured in Layer 2, it advances to the third Layer (Data Processing), where data is transformed into the structured knowledge that underpins simulation, prediction, and optimization. This analytical phase resamples time-series data to a common timeline, reconciles measurement units, and enriches each record with contextual metadata such as asset identity. From the enriched stream, the layer derives higher-level information, such as windowed aggregates that reveal trends, statistical figures that indicate the system's status, and engineered features that carry predictive power for higher-layer analytics. As these transformations operate on a persistent corpus rather than transient packets, they can choose their cadence freely, and processing failures affect only the derived products, not the raw data.

Cloud providers supply event-driven, serverless runtimes that enable elastic and scalable processing. AWS Lambda and Azure Functions execute code on demand in response to IoT messages, storage events, or APIs and charge solely for invocation count and runtime. While AWS offers fine-grained control over concurrency and runtime limits, Azure integrates more tightly with the broader event-driven ecosystem of the Azure platform.

D. Layer 4—DT Management

This layer brings semantic structure to the data collected and processed in lower layers. It manages the internal representation of DT models, including entity graphs, metadata, synchronization logic, and lifecycle operations. Beyond passive monitoring, this layer often hosts simulation and prediction components that influence the physical system. Insights generated here can be propagated downward to reconfigure sensing parameters or trigger control actions, thus enabling closed-loop integration with the physical environment.

Example services at the DT management layer include *AWS IoT TwinMaker* and *Azure Digital Twins*. AWS IoT TwinMaker supports entity modeling, scene graphs, and data connectors to integrate IoT data into a spatially and semantically structured twin model. Azure Digital Twins focuses on modeling physical environments using the DTs Definition Language (DTDL), enabling rich ontologies, relationships, and query-based state management. The two platforms differ primarily in their pricing schemes: TwinMaker charges per query and component usage, while Azure Digital Twins is billed based on messages and API operations.

E. Layer 5—Visualization

The final layer is responsible for user interaction with the DT. It presents both collected and derived information (e.g., simulation outcomes or predictive analytics) through dashboards and visualization interfaces. This layer plays a key role in human-in-the-loop systems, as it enables stakeholders to monitor, evaluate, and act on the DT's outputs, while also feeding manual decisions back into the system. Common use cases include real-time dashboards, historical data trend analysis, and user-configurable monitoring widgets.

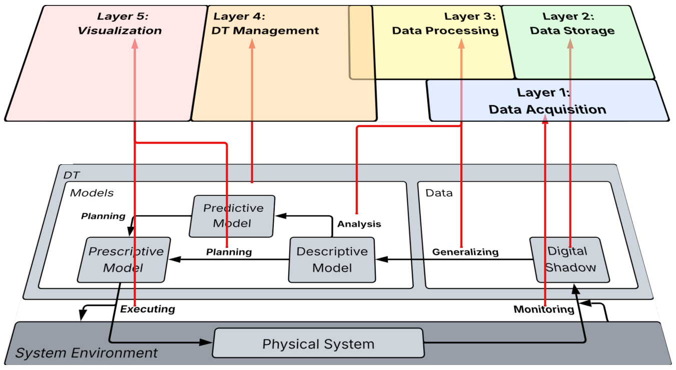


Fig. 2: Twin2Clouds layers mapped on to the conceptual framework for DTs, from [5, Figure 1].

Example services at the visualization layer include *AWS Managed Grafana* and *Azure Managed Grafana*. Both offerings provide hosted, scalable Grafana instances for building dashboards and visualizations across time-series and operational data. AWS charges on a per-user basis, while Azure combines infrastructure (instance size) and per-user costs, making pricing sensitive to the number of users and the duration of dashboard usage.

F. Mapping the Twin2Clouds Cloud-Oriented Architecture to Foundational DT Models

Eramo *et al.* [5] base their conceptual framework on the Models and Data (MODA) [28] paradigm, as shown in Figure 2. Their framework describes a DT as a cycle that monitors an actual system, builds a digital shadow, generalizes that shadow into descriptive, predictive, and prescriptive models, and finally executes planned actions back on the system. We adopt this framework because its clearly separated roles align well with the layered service abstraction of public clouds. While other DT models exist [29], [30], Twin2Clouds depends on provider-agnostic cost and pricing, and thus remains compatible with alternative decompositions.

Within Twin2Clouds, each phase is realized by a dedicated deployment layer, as shown in Figure 2. *Data Acquisition* implements the monitoring part of the conceptual framework. Managed MQTT/HTTP brokers stream raw telemetry from edge devices into the cloud, while enforcing identity, encryption, and routing—the prerequisites for creating the digital shadow. *Data Storage* contains the shadow itself: hot stores capture live-state and warm/cold tiers preserve history, and each tier's retention policy mirrors the temporal scope envisioned by the Data component (of the conceptual framework). *Data Processing* realizes the generalization. Stream analytics, feature extraction, and batch pipelines transform unstructured sensor feeds into clean, aggregated datasets that calibrate descriptive models and feed predictive ones. *DT Management* hosts the DT models, tracks their version, and supplies orchestration hooks for simulations, rule engines, or optimization routines. *Visualization* complements this by presenting insights to users and providing an entry point for manual control.

Prescriptive actions (e.g., device or sensor reconfiguration, adjusting setpoints) can be triggered automatically in Layers 4 or 5. In both cases, they are propagated through Layer 2, which updates the shadow and generates commands that Layer 1 streams to the physical twin, thereby closing the feedback loop.

By mapping concrete cloud layers onto these abstract roles, we preserve the conceptual clarity while exposing clear cost and governance boundaries for each functionality of the twin. Engineers can therefore reason about provider choices and pricing per layer without losing sight of the end-to-end semantics defined by the foundational conceptual model. It is essential to note that the complexity of the DT Management layer can vary significantly depending on the variety of underlying DT models and the complexity of associated model-management capabilities. Within Model-Driven Engineering [31], [32], DT model management may encompass model editing via model-specific APIs (e.g., as provided by the Eclipse Modeling Framework API [33]), model validation against their metamodels, or executing transformations to support diverse engineering tasks [34]. In principle, descriptive, predictive, and prescriptive models can all be managed within our DT Management layer—hosted on AWS TwinMaker or Azure Digital Twins—while workload-intensive models that rely on AI, optimization solvers, or other specialized algorithms require special attention. Those workloads are typically executed on clusters (cloud or on-premises) or dedicated cloud-based ML services, where compute profiles and toolchains vary by domain. Consequently, our architecture defines Layers 1–5 and their corresponding services, while remaining neutral about the implementation of domain-specific functionalities, thereby keeping the stack provider-agnostic and open to specialized extensions.

V. Twin2Clouds' Cost Model and Deployment Optimization

Section III showed that cost and pricing strategies varied substantially across cloud providers. This section aims to introduce generic cost and pricing primitives, together referred to as *cost model*, that can be mapped to any cloud provider. Finally, we map these primitives to cloud providers and optimize their deployment costs.

A. Cost Primitives $\mathcal{C}$

To enable cost-aware deployment decisions, we define a set of generic *cost primitives* that abstract the pricing logic used by common cloud services across providers. Each cloud service used in Twin2Clouds architecture can be expressed as a function of one or more of these primitives.

Message-Based ($\mathcal{C}_M$) cost primitive captures services that are billed per message, typically found in IoT ingestion services. The cost is computed as:

$$\mathcal{C}_M = c_m \cdot N_m$$

where c_m is the unit cost per message and N_m is the total number of messages sent in the billing period.

Execution-Based ($\mathcal{C}_E$) cost primitive applies to event-driven compute services such as serverless functions. It charges per execution and execution duration:

$$\mathcal{C}_E = c_e \cdot N_e + c_t \cdot T_e$$

where N_e is the number of executions, T_e the total execution time (in GB-seconds), and c_e, c_t are the fixed execution-based and time-based rates respectively.

Storage-Based ($\mathcal{C}_S$) cost primitive is used for storage services where cost depends on data volume and duration:

$$\mathcal{C}_S = c_s \cdot V \cdot D$$

V denotes the volume stored (in GB), D the duration (the billing period), and c_s the per-GB rate.

Action-Based ($\mathcal{C}_A$) cost primitive is relevant for databases, storage, or service APIs with metered requests. The cost is calculated as:

$$\mathcal{C}_A = c_a \cdot N_a$$

where N_a is the number of actions, such as read/write operations, API calls, but even automatically triggered actions.

User-Based ($\mathcal{C}_U$) cost primitive is typically used in visualization platforms and access-controlled services. It charges per active user:

$$\mathcal{C}_U = c_u \cdot N_u$$

where N_u is the number of users and c_u is the cost per user.

These cost primitives abstract over the variety of pricing schemes used by commercial providers and form the foundation of our deployment cost formulation.

B. Pricing Primitives $\mathcal{P}$

While cost primitives C define the mathematical structure of a service's billing logic, *pricing primitives* $\mathcal{P}$ capture the provider-specific economic policies applied to that structure. Cloud services from different providers may share the same cost primitive (e.g., per-message), but follow fundamentally different pricing strategies. To distinguish such cases, we define several pricing primitives $\mathcal{P}$.

Pay-Per-Use ($\mathcal{P}_{\textbf{ppu}}$) pricing primitive applies a linear pricing scheme where users are billed directly based on usage, such as number of messages, GBs stored, or API calls. Examples include AWS IoT Core (per message) and AWS S3 (per GB-month).

Bundled Tiered Capacity ($\mathcal{P}_{\textbf{bundle}}$) pricing primitive introduces a pricing tier that includes a fixed quota for resources such as messages, devices, or storage. Exceeding the quota incurs overage fees. Azure IoT Hub exemplifies this model, offering S1–S3 tiers with bundled capacity.

Provisioned Throughput ($\mathcal{P}_{\textbf{prov}}$) primitive is used when users allocate capacity (measured in I/O or computations) in advance. Billing is based on provisioned capacity regardless of usage, with optional autoscaling or reserved units. This is common in AWS DynamoDB and Azure Cosmos DB.

Per-User ($\mathcal{P}_{\textbf{user}}$) pricing primitive is determined by the number of active users per billing period. This pricing scheme applies to visualization services like Managed Grafana.

TABLE III: AWS services mapped to $\mathcal{C}$ and $\mathcal{P}$

Layer	**AWS Service**	$\mathcal{C}$	$\mathcal{P}$
Data Acquisition	IoT Core	$\mathcal{C}_M$	$\mathcal{P}_{\text{ppu}}$
Data Storage (Hot)	DynamoDB	$\mathcal{C}_A$, $\mathcal{C}_S$	$\mathcal{P}_{\text{prov}}$, $\mathcal{P}_{\text{ppu}}$
Data Storage (Cold)	S3 Standard-IA	$\mathcal{C}_S$	$\mathcal{P}_{\text{ppu}}$
Data Storage (Archive)	S3 Glacier-Deep Archive	$\mathcal{C}_S$	$\mathcal{P}_{\text{ppu}}$
Data Processing	Lambda	$\mathcal{C}_E$	$\mathcal{P}_{\text{ppu}}$
DT Management	IoT TwinMaker	$\mathcal{C}_A$	$\mathcal{P}_{\text{infra}}$
Visualization	Managed Grafana	$\mathcal{C}_U$	$\mathcal{P}_{\text{user}}$

TABLE IV: Azure services mapped to $\mathcal{C}$ and $\mathcal{P}$

Layer	**Azure Service**	$\mathcal{C}$	$\mathcal{P}$
Data Acquisition	IoT Hub	$\mathcal{C}_M$	$\mathcal{P}_{\text{bundle}}$
Data Storage (Hot)	Cosmos DB	$\mathcal{C}_A$, $\mathcal{C}_S$	$\mathcal{P}_{\text{prov}}$
Data Storage (Cold)	Blob – Cool Tier	$\mathcal{C}_S$	$\mathcal{P}_{\text{ppu}}$
Data Storage (Archive)	Blob – Archive Tier	$\mathcal{C}_S$	$\mathcal{P}_{\text{ppu}}$
Data Processing	Functions	$\mathcal{C}_E$	$\mathcal{P}_{\text{ppu}}$
DT Management	Digital Twins	$\mathcal{C}_A$, $\mathcal{C}_M$	$\mathcal{P}_{\text{prov}}$, $\mathcal{P}_{\text{infra}}$
Visualization	Managed Grafana	$\mathcal{C}_U$	$\mathcal{P}_{\text{user}}$, $\mathcal{P}_{\text{infra}}$

Instance-Based ($\mathcal{P}_{\textbf{infra}}$) pricing primitive is tied to managed infrastructure (e.g., virtual machines, container instances, hosted endpoints). Some components of Azure Digital Twins or AWS TwinMaker may fall into this category when data connectors or endpoints are involved. In contrast to $\mathcal{P}_{\text{ppu}}$, this pricing is static and independent from the workload.

By combining cost primitives with pricing primitives, we can more accurately model total deployment cost and explain differences across providers even for seemingly equivalent services.

C. Mapping Services to Cost and Pricing Primitives

The following tables summarize how the cloud services used in each layer of our architecture relate to both the cost primitives ($\mathcal{C}$) and the pricing primitives ($\mathcal{P}$).

Table III maps AWS services with the cost and pricing primitives. The AWS ecosystem predominantly adopts the pay-per-use pricing primitive across most services, including IoT Core, Lambda, and S3. AWS offers granular, pay-per-use pricing for most services, which aligns well with dynamic DT workloads. Notably, DynamoDB additionally supports provisioned pricing primitive, allowing users to balance between predictable cost and usage-based flexibility. AWS S3 supports multi-tiered cold and archive storage, including both Glacier and Glacier Deep Archive, which differ in price and retrieval latency—providing finer optimization than Azure's single archive tier. TwinMaker pricing is tied to connector activity and storage endpoints, reflecting infrastructure pricing primitive, similar to DT management workloads. Finally, while AWS Managed Grafana charges per user, infrastructure costs (e.g., for workspace hosting) are generally implicit or included.

The mappings of Azure services with the cost and pricing primitives are summarized in Table IV. Azure favors coarser-grained pricing logic. IoT Hub uses tiered bundles with fixed quotas, potentially leading to cost inefficiencies when underutilized. Cosmos DB requires provisioning throughput and lacks a pure pay-per-request model, making it less elastic than DynamoDB in bursty conditions. Unlike AWS, Azure's Blob Storage Archive tier is singular, offering fewer options than Glacier vs. Deep Archive. However, retrieval and access patterns are similarly priced. Azure Managed Grafana includes both per-user and infrastructure pricing, where users select capacity-based tiers. Azure Digital Twins also combines infrastructure-level pricing with operation-level charges, adding complexity but improving predictability in stable workloads.

D. Inter-Layer Transfer Requirements

While each layer in the DT architecture can be analyzed and priced individually, communication between layers often incurs additional cost, particularly when data needs to cross service or provider boundaries. These inter-layer transfers are critical to maintaining the end-to-end flow of the DT pipeline and must be explicitly modeled in cost-sensitive deployments.

We identify the following inter-layer transfer points that may generate additional cost or require auxiliary services:

- *Layer 1 (Data Acquisition) to Layer 2 (Data Storage)*: Incoming telemetry might undergo lightweight preprocessing before being persisted (e.g., discarding invalid messages). This task may be handled by built-in filtering features in Layer 1 or 2, or it may need to be delegated to serverless mechanisms, which can introduce additional costs.
- *Tiered storage transitions within Layer 2 based on data aging thresholds*: As data becomes older, it is migrated from hot to cold and eventually to archival storage according to user-defined retention boundaries (e.g., 1 month in hot, 2–12 months in cold, up to 5 years in archive). This process, which we assume to be triggered periodically (e.g., monthly), may incur transfer and function execution costs, especially if implemented via serverless mechanisms.
- *Layer 2 (Storage) to Layer 4 (DT Management)*: Twin management services often rely on historical data to simulate system behavior or update digital models. If storage and DT management services are deployed on different providers, or if explicit queries are required, the transfer may incur egress and processing costs, often due to vendor lock-in.
- *Layer 4 (DT Management) to Layer 5 (Visualization)*: Visualization services like Grafana typically query DT data through APIs or time-series databases. If these services are hosted on different providers or if they operate in loosely coupled modes, additional data movement or querying costs arise.

In each of these cases, the transfer may either be internal to the same provider and service family or span different services or clouds. While the costs for the former cases may be negligible or policy-based, the costs for the latter often would require explicit transfer mechanisms and billing based on volume, request count, or function execution. Section V-E details the specific services used to perform these transfers in both AWS and Azure ecosystems and maps them to the cost and pricing primitives introduced earlier.

TABLE V: Transfer services mapped to $\mathcal{C}$ and $\mathcal{P}$

Transfer	**Example Services**	$\mathcal{C}$	$\mathcal{P}$
L1 (Data Acquisition) $\rightarrow$ L2 (Data Storage)	AWS Lambda, AZ Functions	$\mathcal{C}_E$,	$\mathcal{P}_{\text{ppu}}$
Tiered storage transitions (Hot $\rightarrow$ Cold $\rightarrow$ Archive)	*AWS Lambda + S3 APIs, AZ Function + Blob APIs*	$\mathcal{C}_E$, $\mathcal{C}_S$	$\mathcal{P}_{\text{ppu}}$
L3 (Storage) $\rightarrow$ L4 (DT Management)	*TwinMaker $\rightarrow$ S3, AZ DT $\rightarrow$ Blob/Cosmos DB*	$\mathcal{C}_A$, $\mathcal{C}_S$	$\mathcal{P}_{\text{ppu}}$
L4 (DT Management) $\rightarrow$ L5 (Visualization)	*TwinMaker or AZ DT $\rightarrow$ Managed Grafana*	$\mathcal{C}_A$, $\mathcal{C}_S$	$\mathcal{P}_{\text{ppu}}$

TABLE VI: User-defined workload parameters $\mathbf{w}$ and their mapping to $\mathcal{C}$

User-defined Workload Parameter	**Affected Primitives**
#IoT devices	$\mathcal{C}_S, \mathcal{C}_M$ (message/storage volume)
Message Frequency	$\mathcal{C}_S, \mathcal{C}_M$ (message/storage volume)
Average Message Size	$\mathcal{C}_S$ (bandwidth, storage)
Data Retention Period	$\mathcal{C}_S$ (storage volume per tier)
3D Model required (yes/no)	Affects service choice (supported features)
#3D Entities	$\mathcal{C}_S, \mathcal{C}_A$ (DT complexity, stored data, computational load)
#Dashboard Editors	$\mathcal{C}_U$ (users with write access)
#Dashboard Viewers	$\mathcal{C}_U$ (read-only users)
Dashboard Active Hours	$\mathcal{C}_A$ (#queries to layer 4)

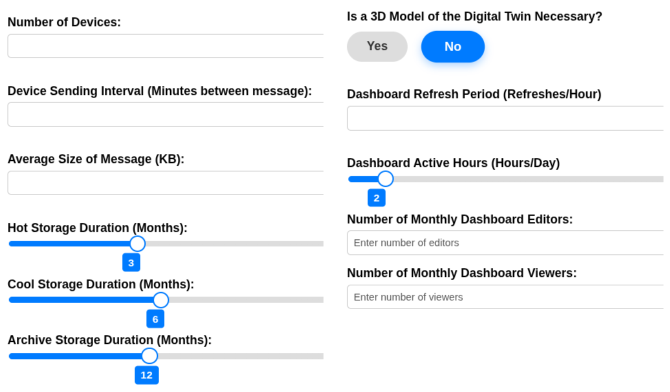
E. Mapping Transfer Services to Cost and Pricing Primitives

The inter-layer transfer points identified in Section V-D require supporting cloud services to enable data movement, orchestration, and access control. These services may incur costs related to data transfer volume, function execution time, and request count. Table V maps the services used for these transfers to the corresponding cost primitives $\mathcal{C}$ and pricing primitives $\mathcal{P}$, consistent with the taxonomy introduced in Sections V-A and V-B.

All transfer services share two cost characteristics: (1) They are primarily usage-based and thus modeled using $\mathcal{P}_{\text{ppu}}$, and (2) they rely on multiple cost primitives depending on the transfer type—typically involving data volume ($\mathcal{C}_S$) and either execution time ($\mathcal{C}_E$) or actions ($\mathcal{C}_A$). These primitives apply to both automated (e.g., lifecycle rules, time-triggered functions) and user-invoked mechanisms (e.g., API-based transfer pipelines).

F. Workload Parameters $\mathbf{w}$ and Their Mapping to Cost Primitives

While Sections V-C–E mapped each service and transfer to abstract cost and pricing primitives, the magnitude of the billed costs still depends on the workload itself. Consequently, this section introduces a set of user-supplied workload parameters $\mathbf{w}$—summarized in Table VI—that translate the abstract cost and pricing primitives into a concrete total-cost estimate. Instantiating the primitives with $\mathbf{w}$ bridges the layer-level taxonomy and the comprehensive cost model (Section V-G).



Number of Devices:

Device Sending Interval (Minutes between message):

Average Size of Message (KB):

Hot Storage Duration (Months): 3

Cool Storage Duration (Months): 6

Archive Storage Duration (Months): 12

Is a 3D Model of the Digital Twin Necessary? Yes [✓ No]

Dashboard Refresh Period (Refreshes/Hour)

Dashboard Active Hours (Hours/Day) 2

Number of Monthly Dashboard Editors: Enter number of editors

Number of Monthly Dashboard Viewers: Enter number of viewers

Fig. 3: Twin2Clouds's workload parameters GUI input

G. Total Cost Model Composition

This section presents a generalized cost model to quantify the overall deployment cost of a DT architecture deployed across multiple cloud providers. The model considers both the costs of executing services at each architectural layer and the costs of transferring data between layers when they reside on different providers.

Let:

- $\mathbb{L} = \{l_1, l_2, \ldots, l_n\}$ be the set of DT layers
- $\mathbb{P}$ be the set of supported providers.
- $\mathbf{p} = (p_1, \ldots, p_n)$, where $p_i \in \mathbb{P}$ specifies the cloud provider for layer l_i, fixing the concrete service for each layer (see Tables III and IV)
- $\mathbf{w}$ denotes the user-defined workload parameters, introduced in Section V-F (Table VI)
- $C_{\text{S}}(l_i, p_i, \mathbf{w})$ be the *service* cost of layer l_i on its assigned provider p_i; this cost is a function of the workload parameters $\mathbf{w}$
- $C_{\text{T}}(l_i \rightarrow l_j, \mathbf{p}, \mathbf{w})$ denote the *traffic* cost, of moving data form layer l_i to l_j; it depends on both the provider assignment vector $\mathbf{p}$ and the workload parameters $\mathbf{w}$

The *total deployment cost* $C(\mathbf{p}, \mathbf{w})$ is then defined as:

$$C(\mathbf{p}, \mathbf{w}) = \sum_{l_i \in \mathbb{L}} C_{\text{S}}(l_i, p_i, \mathbf{w}) + \sum_{\forall (l_i \rightarrow l_j)} C_{\text{T}}(l_i \rightarrow l_j, \mathbf{p}, \mathbf{w})$$

The optimization objective is to determine the provider allocation vector

$$\mathbf{p}^\star = \underset{\mathbf{p} \in \mathbb{P}^n}{\arg\min}\, C(\mathbf{p}, \mathbf{w})$$

possibly subject to additional constraints, such as provider affinity, mandatory features support (e.g. 3D modeling available only via AWS TwinMaker), data-residency rules, or SLA requirements. The resulting vector $\mathbf{p}^\star$ specifies the cost-optimal assignment of cloud services to layers, together with the associated inter-provider transfer paths, for the given workload.

VI. Twin2Clouds Evaluation

A. Twin2Clouds Implementation

To evaluate Twin2Clouds' cost model and optimization strategy, we developed a deployment planning tool that computes the optimal provider-layer configuration for a cost-aware DT deployment model in Sky computing [3], [4]. According to [5], this model supports both predictive and prescriptive roles: it estimates deployment costs and generates a cost-aware deployment plan that accounts for the calculated costs. The tool takes user-defined workload parameters $\mathbf{w}$, evaluates the total cost across valid deployment combinations, and outputs a structured optimal deployment plan. All monetary calculations assume the standard cloud billing interval of one month; therefore, every usage metric is normalized to this period before the optimizer evaluates the cost.

Twin2Clouds consists of three components. An input form (Fig. 3) collects workload parameters $\mathbf{w}$, including the number of devices, message rates, storage durations, and dashboard usage. Then Twin2Clouds loads pricing configurations from a JSON file containing cost and pricing primitive mappings for AWS and Azure services. To calculate the total deployment cost C, Twin2Clouds performs a path search across the layers, where each layer is a decision point and each provider option is a branching edge with cost derived from primitives, workload parameters, and transfer services. The path with the minimum cumulative cost is selected, and the resulting deployment specification is output in JSON format.

Twin2Clouds is designed to be modular and extensible. It uses the current pricing schemes of the respective cloud services of two cloud providers AWS and Azure. Users can simply change these pricing schemes in a configurable JSON file. New cloud services or additional layers can be integrated by extending the pricing configuration and mapping tables. The optimizer can optionally incorporate constraints such as provider restrictions or feature requirements (e.g., 3D visualization support).

B. Evaluation Scenarios

We evaluate Twin2Clouds using three realistic DT scenarios: (1) a large, smart building (e.g., the Edge [35], one of the smartest buildings in the world), (2) a medium-sized smart industrial facility, and (3) a small smart home, hereinafter referred to as large, medium and small. These cases vary significantly in scale, message frequency, storage requirements, and dashboard complexity. Table VII summarizes the key input parameters.

These inputs are processed by Twin2Clouds (Section VI-A) to determine the most cost-efficient deployment strategy across cloud providers.

C. Results and Analysis

Table VIII shows the output (deployment configuration) of the Twin2Clouds optimizer to optimize cost for the deployed DT, based on the three defined scenarios.

Smart Building—Large. In this large-scale scenario, Azure dominates across all data layers due to better cost scaling with high-volume device traffic and storage. Furthermore, IoT Hub follows a bundling pricing primitive ($\mathcal{P}_{\text{bundle}}$) which is more cost-effective for high-volume scenarios. While the system selects Azure for all data layers, the DT Management and Visualization layers are selected on AWS TwinMaker and Grafana, which are more cost-efficient than Azure's services. This is mainly due to Azure's provisioned throughput pricing primitive $\mathcal{P}_{\text{prov}}$ for Azure Digital Twins, instance-based pricing primitive $\mathcal{P}_{\text{infra}}$ for Azure Managed Grafana, message-based cost primitive $\mathcal{C}_M$, and differences in user costs ($\mathcal{C}_U$) for Grafana. The monthly cost is **\$45,770.37**, representing:

- **23.9%** savings vs. AWS-only (\$60,156.43), and
- **23.7%** savings vs. Azure-only (\$59,995.54).

Both single-cloud options cost nearly the same at scale, yet Fig. 4a (left) reveals significant service-level price gaps, clearly benefiting a multi-cloud approach that can select the best service for each layer, reducing costs by almost one fourth.

Industrial Facility—Medium. In this scenario, the best allocation is to place Data Acquisition and DT Management on AWS, but using Azure for all storage layers, data processing and visualization. While employing similar cost and pricing primitives for storage, Azure's tiered storage transitions prices are cheaper. Moreover, retrieval from DynamoDB ($\mathcal{P}_{\text{prov}}$, $\mathcal{P}_{\text{ppu}}$) is more expensive than from CosmosDB ($\mathcal{P}_{\text{prov}}$). Similar to the large building scenario, Azure's Digital Twins are more expensive as it additionally bills messages ($\mathcal{C}_M$) and follows a provisioned throughput pricing scheme. The per-layer breakdown in Figure 4b highlights that the DT Management layer yields the largest share of savings. The total monthly cost is **\$1,660.71**, resulting in:

- **2.6%** savings vs. AWS-only (\$1,704.44), and
- **18.8%** savings vs. Azure-only (\$2,046.05).

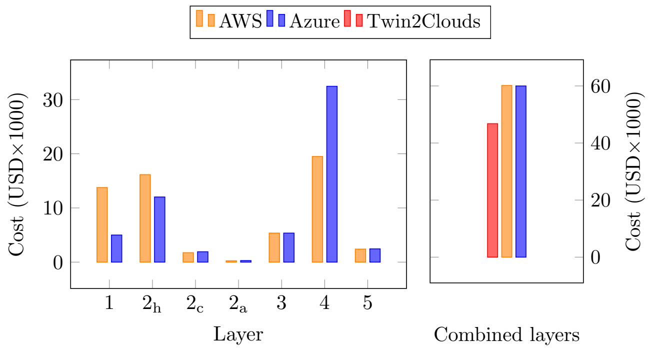
Even though AWS handles most twin-related layers (e.g. Data Acquisition, DT Management) a federated approach is

TABLE VII: Scenario configuration parameters

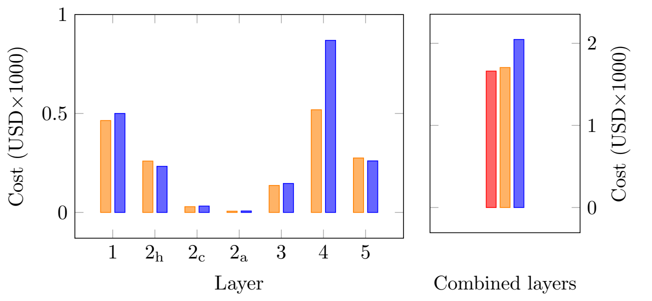
Parameter	Smart Building (L)	I. Facility (M)	Home (S)
# IoT Devices	30,000	4,000	100
Message Size	800 B	500 B	250 B
Message Interval	6 sec	30 sec	2 min
Hot Storage	3 months	1 month	1 month
Cold Storage	12 months	12 months	3 months
Archive Storage	24 months	36 months	12 months
# Editors	100	25	2
# Viewers	300	10	0
Dashboard Refreshes	2/min	1/min	1/5min
Dashboard Active	8 h/day	4 h/day	1 h/day
3D Model	Optional	No	No

TABLE VIII: Decision for each layer per scenario

Layer	Smart Building (L)	I. Facility (M)	Home (S)
Data Acquisition	Azure	AWS	AWS
Data Storage (Hot)	Azure	Azure	AWS
Data Storage (Cold)	Azure	Azure	Azure
Data Storage (Archive)	Azure	Azure	AWS
Data Processing	Azure	Azure	AWS
DT Management	AWS	AWS	AWS
Visualization	AWS	Azure	AWS



(a) Smart Building



(b) Industrial Facility

Fig. 4: Per-layer (left) and combined costs (right) for two use cases: Smart Building and Industrial Facility. Results compare individual providers with Twin2Clouds, which selects the minimum-cost provider per layer to minimize total cost.

still preferable due to efficient IoT ingestion on AWS and better storage pricing across all storage tiers on Azure.

Smart Home—Small. The optimal setup uses AWS for all layers except cold storage, which is placed on Azure Blob's cool tier. Although transferring data incurs additional networking overhead, the cost captured by the $\mathcal{C}_S$ primitive remains lower on Azure than on AWS S3. This hybrid strategy yields a monthly cost of **$25.18**, saving:

- $< \mathbf{1}\%$ vs. AWS-only ($25.23), and
- $\mathbf{78.4}\%$ vs. Azure-only ($116.88).

The significant savings of $4.64\times$ in comparison to the Azure-only configuration originate—similar to the large and medium scenarios—in Azure's cost and pricing model for DTs to charge messages and provisioned throughput additionally ($\mathcal{C}_M$, and $\mathcal{P}_{\text{prov}}$).

Across all scenarios, Twin2Clouds demonstrates that hybrid, federated deployments can outperform single-provider setups, especially at medium and large scale. For larger deployments, a cost-driven provider selection remains essential, even if the final configuration is not federated.

VII. Threats to Validity and Platform Limitations

Following the classical framework by Cook and Campbell [36], we report the following threats.

Internal Validity. Our evaluation relies on AWS and Azure pricing as of June 2025 and assumes perfect service availability and zero network latency, omitting the impact of future pricing changes, outages, or degraded performance.

External Validity. We assessed our cost-aware deployment on three smart-building use cases with two cloud providers; results may not extend to other DT domains (e.g., manufacturing, healthcare), or to domains such as automotive, where DTs are often model-centric [37].

Construct Validity. Our layering follows Eramo et al. [5]. While our cost primitives are, in principle, compatible with alternative conceptual models, different decompositions may influence how DT functionality is mapped to services. We also optimize only monetary cost, whereas real-world DT deployments must balance latency, performance, energy, or compliance—factors that are often use case specific or not exposed by public clouds.

Conclusion Validity. Our deterministic cost estimates omit confidence intervals and sensitivity analyses, so conclusions hold only if workload parameters (message size, frequency, storage duration) remain stable.

Platform Limitations. Support is currently limited to AWS and Azure pricing schemes, but cost and pricing primitives are applicable to other providers (e.g., GCP or IBM); we do not integrate latency, availability, energy, compliance, or CI/CD-driven reconfiguration, which constrains real-time reconfiguration and broader QoS- or sustainability-driven deployments. Our evaluation assumes a single optimization run; in practice, re-optimization may be needed as workloads or prices change, incurring migration costs not captured here.

VIII. Conclusion and Future work

This work presents Twin2Clouds, a cost-aware engineering framework that decomposes a DT into five deployable cloud layers, models each layer's pricing across providers, and automatically recommends the least-cost deployment plan. In three case studies—a large smart building, a medium-sized industrial facility, and a smart home—Twin2Clouds cut monthly costs by up to 78.4%, demonstrating that modest federation preserves functionality while delivering substantial savings. Beyond the empirical results, Twin2Clouds contributes (1) a generalizable catalog of cost and pricing primitives, (2) an open-access optimization tool, and (3) a clear mapping between DT abstractions and mainstream cloud services.

Future work includes incorporating additional providers, integrating latency, availability, energy, and compliance constraints, accounting for migration costs in repeated re-optimization, and exposing a CI/CD API to evolve Twin2Clouds into a comprehensive DT deployment advisor.

Acknowledgment

This work was funded by the Key Digital Technologies (KDT) Joint Undertaking through the project MATISSE, grant agreement No. 101140216; and *Land Tirol, Austria*, under contract F.35499. The authors used OpenAI GPT-4o and GPT-5 for grammar and language editing. All content was reviewed and approved by the authors.

References

[1] F. Bordeleau, B. Combemale, R. Eramo, M. Van Den Brand, and M. Wimmer, "Towards model-driven digital twin engineering: Current opportunities and future challenges," in *International conference on systems modelling and management*. Springer, 2020, pp. 43–54.

[2] M. Attaran and B. G. Celik, "Digital twin: Benefits, use cases, challenges, and opportunities," *Decision Analytics Journal*, vol. 6, p. 100165, 2023.

[3] T. Larcher, P. Gritsch, S. Nastic, and S. Ristov, "Baasless: Backend-as-a-service (baas)-enabled workflows in federated serverless infrastructures," *IEEE Transactions on Cloud Computing*, vol. 12, no. 4, pp. 1088–1102, 2024.

[4] Z. Yang, Z. Wu, M. Luo, W.-L. Chiang, R. Bhardwaj, W. Kwon, S. Zhuang, F. S. Luan, G. Mittal, S. Shenker, and I. Stoica, "SkyPilot: An intercloud broker for sky computing," in *Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX, Apr. 2023, pp. 437–455.

[5] R. Eramo, F. Bordeleau, B. Combemale, M. v. d. Brand, M. Wimmer, and A. Wortmann, "Conceptualizing digital twins," *IEEE Software*, vol. 39, no. 2, pp. 39–46, 2022.

[6] K. Schweichhart, "Reference Architectural Model Industrie 4.0 (RAMI 4.0): An Introduction," Plattform Industrie 4.0, Technical Report, 2016, accessed: July 3, 2025. [Online]. Available: https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference%5Farchitectural%5Fmodel%5Findustrie%5F4.0%5Frami%5F4.0.pdf

[7] International Organization for Standardization. (2021) ISO 23247-1:2021 – Automation systems and integration — Digital twin framework for manufacturing — Part 1: Overview and general principles. [Online]. Available: https://www.iso.org/standard/75066.html

[8] L. U. Khan, Z. Han, W. Saad, E. Hossain, M. Guizani, and C. S. Hong, "Digital twin of wireless systems: Overview, taxonomy, challenges, and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 4, pp. 2230–2254, 2022.

[9] E. Ferko, A. Bucaioni, and M. Behnam, "Architecting digital twins," *IEEE Access*, vol. 10, pp. 50 335–50 350, 2022.

[10] J. Li, S. Guo, W. Liang, J. Wang, Q. Chen, Z. Xu, and W. Xu, "Aoi-aware user service satisfaction enhancement in digital twin-empowered edge computing," *IEEE/ACM Transactions on Networking*, vol. 32, no. 2, pp. 1677–1690, 2024.

[11] J. Li, J. Wang, Q. Chen, Y. Li, and A. Y. Zomaya, "Digital twin-enabled service satisfaction enhancement in edge computing," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, 2023, pp. 1–10.

[12] D. Groombridge and G. Analysts, "2023 gartner top strategic technology trends," https://emt.gartnerweb.com/ngw/globalassets/en/publications/documents/2023-gartner-top-strategic-technology-trends-ebook.pdf, 2023, accessed: July 1, 2025. Page 33 referenced in citation.

[13] J. Lee, C. Jin, and Z. Liu, *Predictive Big Data Analytics and Cyber Physical Systems for TES Systems*. Cham: Springer International Publishing, 2017, pp. 97–112. [Online]. Available: https://doi.org/10.1007/978-3-319-49938-3%5F7

[14] K. M. Alam and A. El Saddik, "C2ps: A digital twin architecture reference model for the cloud-based cyber-physical systems," *IEEE Access*, vol. 5, pp. 2050–2062, 2017.

[15] T. Gabor, L. Belzner, M. Kiermeier, M. T. Beck, and A. Neitz, "A simulation-based architecture for smart cyber-physical systems," in *2016 IEEE International Conference on Autonomic Computing (ICAC)*, 2016, pp. 374–379.

[16] G. N. Schroeder, C. Steinmetz, R. N. Rodrigues, R. V. B. Henriques, A. Rettberg, and C. E. Pereira, "A methodology for digital twin modeling and deployment for industry 4.0," *Proceedings of the IEEE*, vol. 109, no. 4, pp. 556–567, 2021.

[17] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018, 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896318316021

[18] V. V. Tuhaise, J. H. M. Tah, and F. H. Abanda, "Technologies for digital twin applications in construction," *Automation in Construction*, vol. 152, p. 104931, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580523001917

[19] Q. Lu, A. K. Parlikad, P. Woodall, G. D. Ranasinghe, X. Xie, Z. Liang, E. Konstantinou, J. Heaton, and J. Schooling, "Developing a digital twin at building and city levels: Case study of west cambridge campus," *Journal of Management in Engineering*, vol. 36, no. 3, p. 05020004, 2020. [Online]. Available: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29ME.1943-5479.0000763

[20] P. Talasila, C. Gomes, L. B. Vosteen, H. Iven, M. Leucker, S. Gil, P. H. Mikkelsen, E. Kamburjan, and P. G. Larsen, "Composable digital twins on digital twin as a service platform," *SIMULATION*, vol. 101, no. 3, pp. 287–311, 2025. [Online]. Available: https://doi.org/10.1177/00375497241298653

[21] M. Xiu and V. Andrikopoulos, "The nefolog & midsus systems for cloud migration support," Universität Stuttgart, Fakultät Informatik, Electrotechnik und Informationstechnik, Technical report, 2013, accessed: August 8, 2025. [Online]. Available: https://www2.informatik.uni-stuttgart.de/bibliothek/ftp/ncstrl.ustuttgart_fi/TR-2013-08/TR-2013-08.pdf

[22] V. Andrikopoulos, A. Reuter, M. Xiu, and F. Leymann, "Design support for cost-efficient application distribution in the cloud," in *2014 IEEE 7th International Conference on Cloud Computing*, 2014, pp. 697–704.

[23] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 164–177, 2012.

[24] A. Siriweera and K. Naruse, "Survey on cloud robotics architecture and model-driven reference architecture for decentralized multicloud heterogeneous-robotics platform," *IEEE Access*, vol. 9, pp. 40 521–40 539, 2021.

[25] S. Ristov, A. Meshcheriakova, P. Gritsch, P. Zech, and R. Breu, "Goal-driven building automation using serverless computing," in *The third international workshop on intelligent and adaptive edge-cloud operations and services*, Jun. 2025.

[26] Y. Zhang, W. Liang, W. Xu, Z. Xu, and X. Jia, "Cost minimization of digital twin placements in mobile edge computing," *ACM Trans. Sen. Netw.*, vol. 20, no. 3, May 2024. [Online]. Available: https://doi.org/10.1145/3658449

[27] David Wright, Dennis Smith, Miguel Angel Borrega, Alessandro Galimberti, and Carolin Zhou, "Magic quadrant for strategic cloud platform services," Gartner, Research Report G00806356, Oct 2024, accessed: July 9, 2025. [Online]. Available: https://www.gartner.com/doc/reprints?id=1-2J4WMCRK&ct=241021&st=sb

[28] B. Combemale, J. Kienzle, G. Mussbacher, H. Ali, D. Amyot, M. Bagherzadeh, E. Batot, N. Bencomo, B. Benni, J.-M. Bruel, J. Cabot, B. H. Cheng, P. Collet, G. Engels, R. Heinrich, J.-M. Jezequel, A. Koziolek, S. Mosser, R. Reussner, H. Sahraoui, R. Saini, J. Sallou, S. Stinckwich, E. Syriani, and M. Wimmer, "A hitchhiker's guide to model-driven engineering for data-centric systems," *IEEE Software*, vol. 38, no. 4, pp. 71–84, 2021.

[29] A. De Benedictis, F. Flammini, N. Mazzocca, A. Somma, and F. Vitale, "Digital twins for anomaly detection in the industrial internet of things: Conceptual architecture and proof-of-concept," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 12, pp. 11 553–11 563, 2023.

[30] G. Beaumont, A. Beugnard, S. Martínez, C. Urtado, and S. Vauttier, "Towards re-engineering digital twins: Preliminary experiments on three use cases," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS Companion '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 453–458. [Online]. Available: https://doi.org/10.1145/3652620.3688259

[31] L. Burgueño, F. Ciccozzi, M. Famelis, G. Kappel, L. Lambers, S. Mosser, R. F. Paige, A. Pierantonio, A. Rensink, R. Salay, G. Taentzer, A. Vallecillo, and M. Wimmer, "Contents for a Model-Based Software Engineering Body of Knowledge," *Software and Systems Modeling*, vol. 18, no. 6, pp. 3193–3205, Dec. 2019. [Online]. Available: http://link.springer.com/10.1007/s10270-019-00746-9

[32] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice, Second Edition*, ser. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2017. [Online]. Available: https://doi.org/10.2200/S00751ED2V01Y201701SWE004

[33] D. Steinberg, Ed., *EMF: Eclipse Modeling Framework*, 2nd ed., ser. The eclipse series. Upper Saddle River, NJ: Addison-Wesley, 2009, oCLC: ocn182662768.

[34] L. Lúcio, M. Amrani, J. Dingel, L. Lambers, R. Salay, G. M. K. Selim, E. Syriani, and M. Wimmer, "Model transformation intents and their

properties," *Software and Systems Modeling*, vol. 15, no. 3, pp. 647–684, 2016.

[35] T. Randall and Bloomberg, "The smartest building in the world," https://www.bloomberg.com/features/2015-the-edge-the-worlds-greenest-building/, 2015, accessed: July 1, 2025.

[36] T. Cook and D. Campbell, *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Houghton Mifflin, 1979.

[37] J. Pfeiffer, D. Fuchß, T. Kühn, R. Liebhart, D. Neumann, C. Neimöck, C. Seiler, A. Koziolek, and A. Wortmann, "Modeling languages for automotive digital twins: A survey among the german automotive industry," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 92–103. [Online]. Available: https://doi.org/10.1145/3640310.3674100