**Azure Data Architecture Guide**

This guide presents a structured approach for designing data-centric solutions on Microsoft Azure. It is based on proven practices derived from customer engagements.

**Introduction**

The cloud is changing the way applications are designed, including how data is processed and stored. Instead of a single general-purpose database that handles all of a solution's data, *polyglot persistence* solutions use multiple, specialized data stores, each optimized to provide specific capabilities. The perspective on data in the solution changes as a result. There are no longer multiple layers of business logic that read and write to a single data layer. Instead, solutions are designed around a *data pipeline* that describes how data flows through a solution, where it is processed, where it is stored, and how it is consumed by the next component in the pipeline.

**How this guide is structured**

This guide is structured around two general categories of data solution, *traditional RDBMS workloads* and *big data solutions*.

**Traditional RDBMS workloads**. These workloads include online transaction processing (OLTP) and online analytical processing (OLAP). Data in OLTP systems is typically relational data with a predefined schema and a set of constraints to maintain referential integrity. Often, data from multiple sources in the organization may be consolidated into a data warehouse, using an ETL process to move and transform the source data.

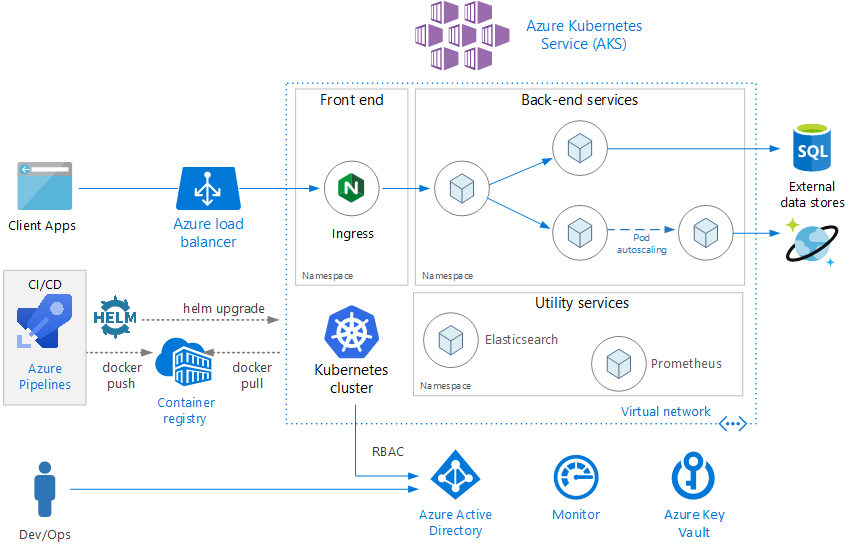
**Big data solutions**. A big data architecture is designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems. The data may be processed in batch or in real time. Big data solutions typically involve a large amount of non-relational data, such as key-value data, JSON documents, or time series data. Often traditional RDBMS systems are not well-suited to store this type of data. The term *NoSQL* refers to a family of databases designed to hold non-relational data. (The term isn't quite accurate, because many non-relational data stores support SQL compatible queries.)

Current trends

MICROSOFT AZURE MICROSERVICES

This reference architecture shows a microservices application deployed to Azure Kubernetes Service (AKS). It describes a basic AKS configuration that can be the starting point for most deployments. This article assumes basic knowledge of Kubernetes. The article focuses mainly on the infrastructure and DevOps considerations of running a microservices architecture on AKS. For guidance on how to design microservices, see [Building microservices on Azure](https://docs.microsoft.com/en-us/azure/architecture/microservices/index).

GitHub logo A reference implementation of this architecture is available on [GitHub](https://github.com/mspnp/microservices-reference-implementation).



*Download a*[*Visio file*](https://archcenter.blob.core.windows.net/cdn/aks-reference-architecture.vsdx)*of this architecture.*

**Architecture**

The architecture consists of the following components.

**Azure Kubernetes Service** (AKS). AKS is an Azure service that deploys a managed Kubernetes cluster.

**Kubernetes cluster**. AKS is responsible for deploying the Kubernetes cluster and for managing the Kubernetes masters. You only manage the agent nodes.

**Virtual network**. By default, AKS creates a virtual network to deploy the agent nodes into. For more advanced scenarios, you can create the virtual network first, which lets you control things like how the subnets are configured, on-premises connectivity, and IP addressing. For more information, see [Configure advanced networking in Azure Kubernetes Service (AKS)](https://docs.microsoft.com/en-us/azure/aks/configure-advanced-networking).

**Ingress**. An ingress exposes HTTP(S) routes to services inside the cluster. For more information, see the section [API Gateway](https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/microservices/aks#api-gateway) below.

**Azure Load Balancer**. An Azure Load Balancer is created when the NGINX ingress controller is deployed. The load balancer routes internet traffic to the ingress.

**External data stores**. Microservices are typically stateless and write state to external data stores, such as Azure SQL Database or Cosmos DB.

**Azure Active Directory**. AKS uses an Azure Active Directory (Azure AD) identity to create and manage other Azure resources such as Azure load balancers. Azure AD is also recommended for user authentication in client applications.

**Azure Container Registry**. Use Container Registry to store private Docker images, which are deployed to the cluster. AKS can authenticate with Container Registry using its Azure AD identity. Note that AKS does not require Azure Container Registry. You can use other container registries, such as Docker Hub.

**Azure Pipelines**. Pipelines is part of Azure DevOps Services and runs automated builds, tests, and deployments. You can also use third-party CI/CD solutions such as Jenkins.

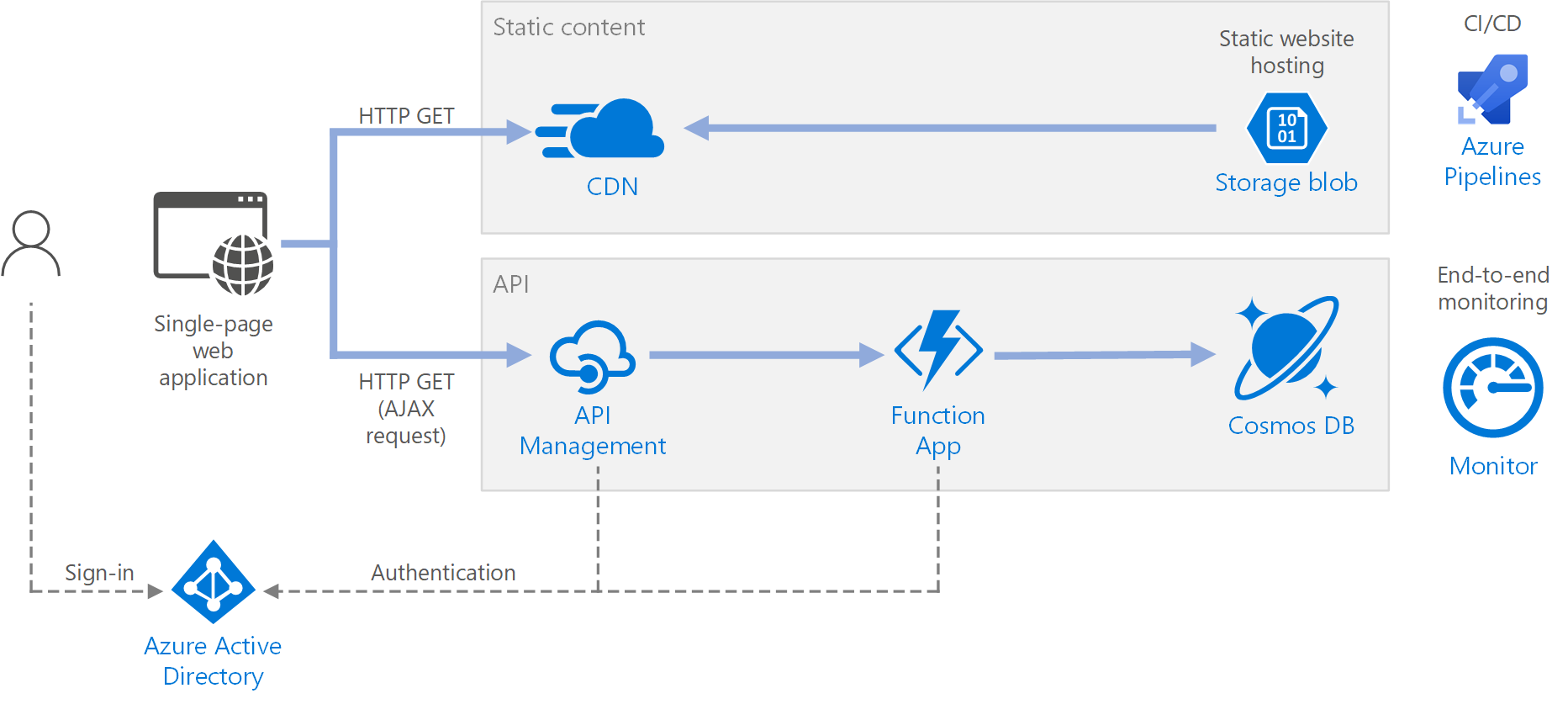
**Helm**. Helm is as a package manager for Kubernetes — a way to bundle Kubernetes objects into a single unit that you can publish, deploy, version, and update.

**Azure Monitor**. Azure Monitor collects and stores metrics and logs, including platform metrics for the Azure services in the solution and application telemetry. Use this data to monitor the application, set up alerts and dashboards, and perform root cause analysis of failures. Azure Monitor integrates with AKS to collect metrics from controllers, nodes, and containers, as well as container logs and master node logs.

**Serverless web application on Azure**

This reference architecture shows a [serverless](https://azure.microsoft.com/solutions/serverless/) web application. The application serves static content from Azure Blob Storage, and implements an API using Azure Functions. The API reads data from Cosmos DB and returns the results to the web app.

GitHub logo A reference implementation for this architecture is available on [GitHub](https://github.com/mspnp/serverless-reference-implementation).



The term serverless has two distinct but related meanings:

* **Backend as a service** (BaaS). Backend cloud services, such as databases and storage, provide APIs that enable client applications to connect directly to these services.
* **Functions as a service** (FaaS). In this model, a "function" is a piece of code that is deployed to the cloud and runs inside a hosting environment that completely abstracts the servers that run the code.

Both definitions have in common the idea that developers and DevOps personnel don't need to deploy, configure, or manage servers. This reference architecture focuses on FaaS using Azure Functions, although serving web content from Azure Blob Storage is an example of BaaS. Some important characteristics of FaaS are:

1. Compute resources are allocated dynamically as needed by the platform.
2. Consumption-based pricing: You are charged only for the compute resources used to execute your code.
3. The compute resources scale on demand based on traffic, without the developer needing to do any configuration.

Functions are executed when an external trigger occurs, such as an HTTP request or a message arriving on a queue. This makes an [event-driven architecture style](https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven) natural for serverless architectures. To coordinate work between components in the architecture, consider using message brokers or pub/sub patterns. For help choosing between messaging technologies in Azure, see [Choose between Azure services that deliver messages](https://docs.microsoft.com/en-us/azure/event-grid/compare-messaging-services).

**Architecture**

The architecture consists of the following components:

**Blob Storage**. Static web content, such as HTML, CSS, and JavaScript files, are stored in Azure Blob Storage and served to clients by using [static website hosting](https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blob-static-website). All dynamic interaction happens through JavaScript code making calls to the backend APIs. There is no server-side code to render the web page. Static website hosting supports index documents and custom 404 error pages.

**CDN**. Use [Azure Content Delivery Network](https://azure.microsoft.com/services/cdn/) (CDN) to cache content for lower latency and faster delivery of content, as well as providing an HTTPS endpoint.

**Function Apps**. [Azure Functions](https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview) is a serverless compute option. It uses an event-driven model, where a piece of code (a "function") is invoked by a trigger. In this architecture, the function is invoked when a client makes an HTTP request. The request is always routed through an API gateway, described below.

**API Management**. [API Management](https://docs.microsoft.com/en-us/azure/api-management/api-management-key-concepts) provides an API gateway that sits in front of the HTTP function. You can use API Management to publish and manage APIs used by client applications. Using a gateway helps to decouple the front-end application from the back-end APIs. For example, API Management can rewrite URLs, transform requests before they reach the backend, set request or response headers, and so forth.

API Management can also be used to implement cross-cutting concerns such as:

* Enforcing usage quotas and rate limits
* Validating OAuth tokens for authentication
* Enabling cross-origin requests (CORS)
* Caching responses
* Monitoring and logging requests

If you don't need all of the functionality provided by API Management, another option is to use [Functions Proxies](https://docs.microsoft.com/en-us/azure/azure-functions/functions-proxies). This feature of Azure Functions lets you define a single API surface for multiple function apps, by creating routes to back-end functions. Function proxies can also perform limited transformations on the HTTP request and response. However, they don't provide the same rich policy-based capabilities of API Management.

**Cosmos DB**. [Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/introduction) is a multi-model database service. For this scenario, the function application fetches documents from Cosmos DB in response to HTTP GET requests from the client.

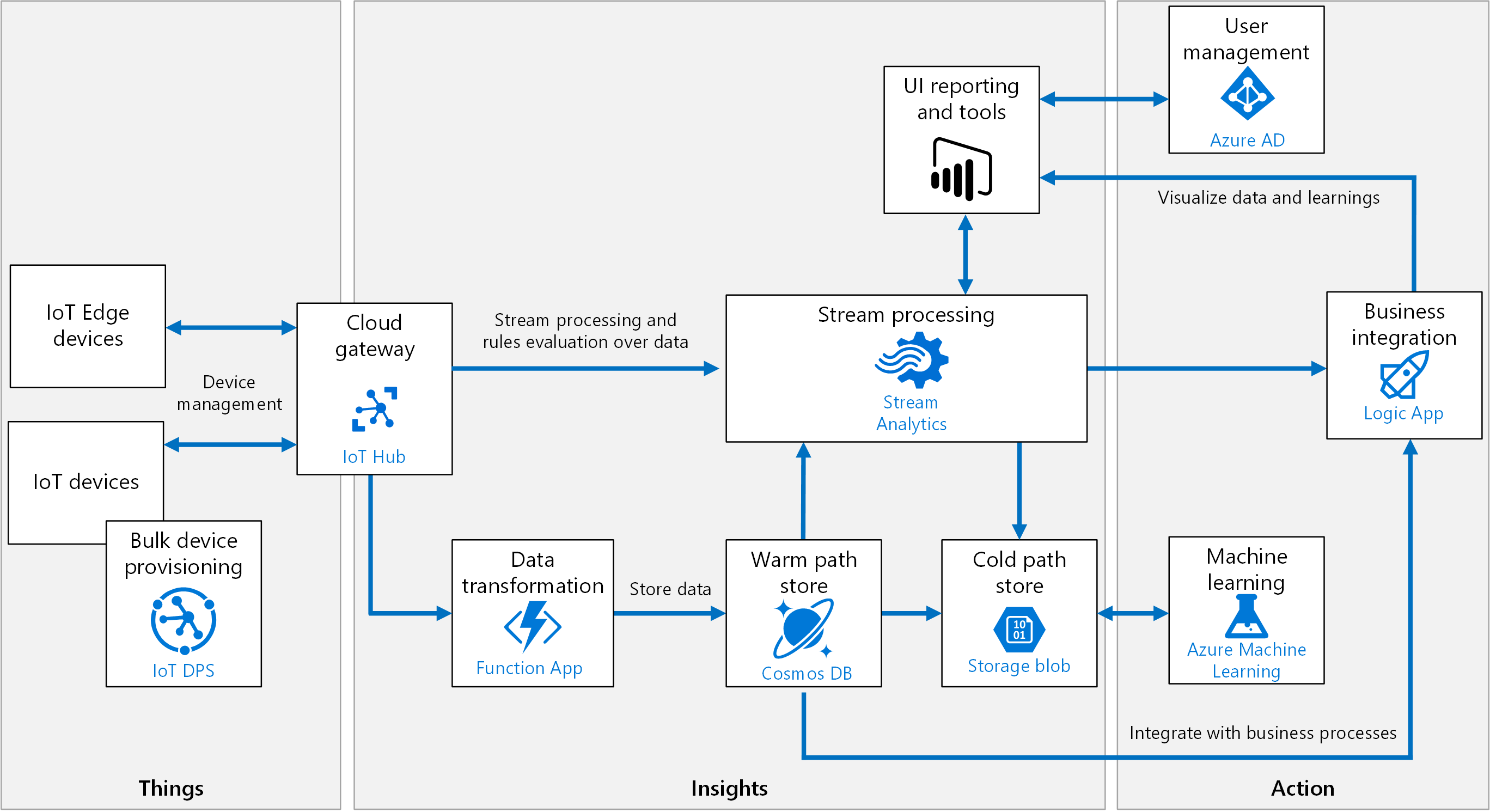
**Azure Active Directory** (Azure AD). Users sign into the web application by using their Azure AD credentials. Azure AD returns an access token for the API, which the web application uses to authenticate API requests (see [Authentication](https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/serverless/web-app#authentication)).

**Azure Monitor**. [Monitor](https://docs.microsoft.com/en-us/azure/azure-monitor/overview) collects performance metrics about the Azure services deployed in the solution. By visualizing these in a dashboard, you can get visibility into the health of the solution. It also collected application logs.

**Azure Pipelines**. [Pipelines](https://docs.microsoft.com/en-us/azure/devops/pipelines/index) is a continuous integration (CI) and continuous delivery (CD) service that builds, tests, and deploys the application.

**Azure IoT reference architecture**

This reference architecture shows a recommended architecture for IoT applications on Azure using PaaS (platform-as-a-service) components.



IoT applications can be described as **things** (devices) sending data that generates **insights**. These insights generate **actions** to improve a business or process. An example is an engine (the thing) sending temperature data. This data is used to evaluate whether the engine is performing as expected (the insight). The insight is used to proactively prioritize the maintenance schedule for the engine (the action).

This reference architecture uses Azure PaaS (platform-as-a-service) components. Other options for building IoT solutions on Azure include:

* [Azure IoT Central](https://docs.microsoft.com/en-us/azure/iot-central/). IoT Central is a fully managed SaaS (software-as-a-service) solution. It abstracts the technical choices and lets you focus on your solution exclusively. This simplicity comes with a tradeoff in being less customizable than a PaaS-based solution.
* Using OSS components such as the SMACK stack (Spark, Mesos, Akka, Cassandra, Kafka) deployed on Azure VMs. This approach offers a great deal of control but is more complex.

At a high level, there are two ways to process telemetry data, hot path and cold path. The difference has to do with requirements for latency and data access.

* The **hot path** analyzes data in near-real-time, as it arrives. In the hot path, telemetry must be processed with very low latency. The hot path is typically implemented using a stream processing engine. The output may trigger an alert, or be written to a structured format that can be queried using analytical tools.
* The **cold path** performs batch processing at longer intervals (hourly or daily). The cold path typically operates over large volumes of data, but the results don't need to be as timely as the hot path. In the cold path, raw telemetry is captured and then fed into a batch process.

**Architecture**

This architecture consists of the following components. Some applications may not require every component listed here.

**IoT devices**. Devices can securely register with the cloud, and can connect to the cloud to send and receive data. Some devices may be **edge devices** that perform some data processing on the device itself or in a field gateway. We recommend [Azure IoT Edge](https://docs.microsoft.com/en-us/azure/iot-edge/) for edge processing.

**Cloud gateway**. A cloud gateway provides a cloud hub for devices to connect securely to the cloud and send data. It also provides device management, capabilities, including command and control of devices. For the cloud gateway, we recommend [IoT Hub](https://docs.microsoft.com/en-us/azure/iot-hub/). IoT Hub is a hosted cloud service that ingests events from devices, acting as a message broker between devices and backend services. IoT Hub provides secure connectivity, event ingestion, bidirectional communication, and device management.

**Device provisioning.** For registering and connecting large sets of devices, we recommend using the [IoT Hub Device Provisioning Service](https://docs.microsoft.com/en-us/azure/iot-dps/) (DPS). DPS lets you assign and register devices to specific Azure IoT Hub endpoints at scale.

**Stream processing**. Stream processing analyzes large streams of data records and evaluates rules for those streams. For stream processing, we recommend [Azure Stream Analytics](https://docs.microsoft.com/en-us/azure/stream-analytics/). Stream Analytics can execute complex analysis at scale, using time windowing functions, stream aggregations, and external data source joins. Another option is Apache Spark on [Azure Databricks](https://docs.microsoft.com/en-us/azure/azure-databricks/).

Machine learning allows predictive algorithms to be executed over historical telemetry data, enabling scenarios such as predictive maintenance. For machine learning, we recommend [Azure Machine Learning Service](https://docs.microsoft.com/en-us/azure/machine-learning/service/).

**Warm path storage** holds data that must be available immediately from device for reporting and visualization. For warm path storage, we recommend [Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/introduction). Cosmos DB is a globally distributed, multi-model database.

**Cold path storage** holds data that is kept longer-term and is used for batch processing. For cold path storage, we recommend [Azure Blob Storage](https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction). Data can be archived in Blob storage indefinitely at low cost, and is easily accessible for batch processing.

**Data transformation** manipulates or aggregates the telemetry stream. Examples include protocol transformation, such as converting binary data to JSON, or combining data points. If the data must be transformed before reaching IoT Hub, we recommend using a [protocol gateway](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-protocol-gateway) (not shown). Otherwise, data can be transformed after it reaches IoT Hub. In that case, we recommend using [Azure Functions](https://docs.microsoft.com/en-us/azure/azure-functions/), which has built-in integration with IoT Hub, Cosmos DB, and Blob Storage.

**Business process integration** performs actions based on insights from the device data. This could include storing informational messages, raising alarms, sending email or SMS messages, or integrating with CRM. We recommend using [Azure Logic Apps](https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-overview) for business process integration.

**User management** restricts which users or groups can perform actions on devices, such as upgrading firmware. It also defines capabilities for users in applications. We recommend using [Azure Active Directory](https://docs.microsoft.com/en-us/azure/active-directory/) to authenticate and authorize users.

**Machine learning**

Microsoft Azure Machine Learning Studio

[Azure Machine Learning](https://azure.microsoft.com/en-us/services/machine-learning/) is aimed at setting a powerful playground both for newcomers and experienced data scientists. The roster of ML products from Microsoft is similar to the ones from Amazon, but Azure, as of today, seems more flexible in terms of out-of-the-box algorithms.

Services from Azure can be divided into two main categories: Azure Machine Learning Studio and Bot Service. Let’s find out what’s under the hood of Azure ML Studio. We’ll return to Bot Service in the section dedicated to specific APIs and tools.

[ML Studio](https://azure.microsoft.com/en-us/services/machine-learning-studio/) is the main MLaaS package to look at. Almost all operations in Azure ML Studio must be completed manually. This includes data exploration, preprocessing, choosing methods, and validating modeling results.

Approaching machine learning with Azure entails some learning curve. But it eventually leads to a deeper understanding of all major techniques in the field. On the other hand, Azure ML supports graphical interface to visualize each step within the workflow. Perhaps the main benefit of using Azure is the variety of algorithms available to play with. The Studio supports around 100 methods that address classification (binary+multiclass), anomaly detection, regression, recommendation, and text analysis. It’s worth mentioning that the platform has one clustering algorithm (K-means).

Another big part of Azure ML is [Cortana Intelligence Gallery](https://gallery.cortanaintelligence.com/). It’s a collection of machine learning solutions provided by the community to be explored and reused by data scientists. The Azure product is a powerful tool for starting with machine learning and introducing its capabilities to new employees.