

AZ-305T00A

Designing Microsoft
Azure Infrastructure
Solutions

Design an application architecture solution

<https://learn.microsoft.com/training/modules/design-application-architecture/>

Learning Objectives

- Describe message and event scenarios
- Design a messaging solution
- Design an event solution (Event Hub and Event Grid)
- Design an application optimization solution
- Design application lifecycle
- Case study
- Learning recap

AZ-305: Design infrastructure solutions
(30-35%)

Design an Application Architecture

- Recommend a messaging architecture
- Recommend an event-driven architecture
- Recommend a solution for API integration
- Recommend a caching solution for applications
- Recommend an application configuration management solution
- Recommend an automated deployment solution for applications

Describe message and event
scenarios

Determine message and event scenarios

Does the sending component expect the communication to be processed in a specific way?

Action	Description	When to use
Event	<ul style="list-style-type: none">• Light weight• Includes a publisher and a subscriber	Used for broadcasts and are often ephemeral. Ephemeral means the communication might not be handled by any receiver if none is currently subscribing.
Message	<ul style="list-style-type: none">• Contains raw data, produced by one component, that will be consumed by another component.• Contains the data itself, not just a reference to that data.	Used where the distributed application requires a guarantee that the communication will be processed.



The cloud is changing how applications are designed and secured. Instead of monoliths, applications are divided into smaller, decentralized services. These services communicate through APIs or by using asynchronous messaging or events. The services scale horizontally, adding new instances as demand requires.

These design changes bring new challenges. Application states are distributed, and operations are done in parallel and asynchronously. Applications must:

- Communicate with each other effectively.
- Be able to be deployed rapidly.
- Be resilient when failures occur.
- Be able to integrate with other systems seamlessly.



Your first decision in your application architecture design is to plan how the application components will communicate. Defining your component strategy helps you choose the appropriate Azure service.

Suppose you're designing the architecture for a home improvement video-sharing application for Tailwind Traders. You want your application to be as reliable and scalable as possible. You're planning to use Azure technologies to build a robust communication infrastructure. Before you can choose the appropriate Azure services, you need to design how each application component will communicate with the other components. For each type of communication, you might choose a different Azure technology.



Things to know about messages and events

Most application components communicate by sending messages or events. Azure offers various services to support the different communication strategies.

Messages

Let's examine the characteristics of [messages](#).

- Messages contain raw data that's produced by one component and consumed by another component.
- A message contains the data itself, not just a reference to that data.

In a message communication, the sending component expects the message data to be processed in a certain way by the destination component. The integrity of the overall system might depend on both the sender and receiver doing a specific job.

Suppose a user uploads a new video by using your mobile video-sharing app. Your mobile app must send the video to the web API that runs in Azure. The video file must be sent, not just an alert that indicates a new video has been added. The mobile app expects that the web API stores the new video in the database and makes the video available to other users.



Events

Now let's take a closer look at [events](#).

- Events are lighter weight than messages and are most often used for broadcast communications.
- An event has two components, a *publisher* and *subscribers*. The event publisher sends the event. The event subscribers receive events.

With events, receiving components generally decide the communications in which they're interested and then subscribe to those events. The subscription is managed by an intermediary. The intermediary can be provided by services like Azure Event Grid or Azure Event Hubs. When publishers send an event, the intermediary routes that event to any interested parties. This pattern is known as a *publish-subscribe* architecture and is the most used.

Events have the following characteristics:

- An event is a lightweight notification that indicates something occurred.
- An event can be sent to multiple receivers or to none.
- An event publisher has no expectations about actions by a receiving component.
- An event is often intended to "fan out" or have many subscribers for each publisher.
- An event is a discrete unit that's unrelated to other events, but an event might be part of a related and ordered series.



Things to consider when choosing messages or events

Review the following scenarios regarding when to choose message or event communication for your application architecture for Tailwind Traders.

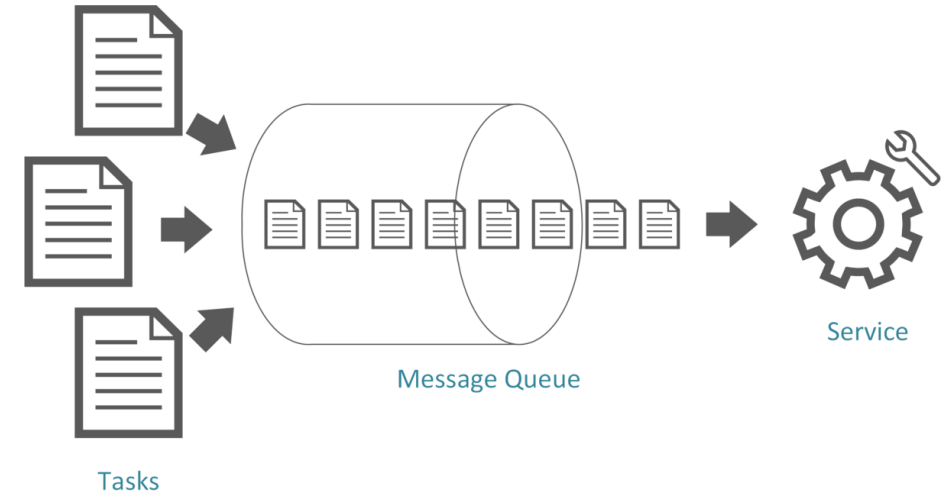
- **Consider messages and events.** It's not uncommon for an application to implement both events and messages. An app can use events for some components and functions and messages for other components. Choose each Azure service to meet the specific needs of each component of your app.
- **Consider sender expectations.** If the sending component in your application expects communication to be processed in a specific way by the destination component, consider implementing messages. If the sender component in your application has no requirements for the destination component, you might implement events rather than messages.
- **Consider guaranteed communication.** If you're building a distributed application and want to guarantee all communication is processed, consider using messages. In a message communication, there's an expectation that both the message sender and receiver complete their tasks.
- **Consider ephemeral communication.** Ephemeral means the communication might not be handled by any receiver if none is currently subscribing. If your application doesn't require subscribers or actions from any receiver, consider using events.

Design a messaging solution

Design for Azure Queue storage

Azure Storage Queue is a service for storing large number of messages.

- Accessed with authenticated calls using HTTP or HTTPS
- Messages can be up to 64 KB in size
- May contain millions of messages, up to the total capacity limit of a storage account



- Create a backlog of work to process asynchronously
- Example: customer placing orders online added to the queue and processed



Azure offers two message-based solutions, Azure Queue Storage and Azure Service Bus. Queue Storage stores large numbers of messages in Azure Storage. Service Bus is a message broker that decouples applications and services. We'll examine the different features and capabilities of these services and consider how to choose which service to implement.

One of your design tasks for Tailwind Traders is to recommend a design for their product demo application. Customers use the app to get the latest tips, reviews, and instructions for featured home improvement products.

You have two requirements for the app design:

- Ensure all content files are uploaded to the web API reliably from the mobile app. Files include text, images, and video.
- Deliver details about new files directly to the app, such as when a customer posts a new product review or a video is added.

For these app requirements, the ideal solution is a message-based system.



Things to know about Azure Service Bus

Azure Service Bus is a fully managed enterprise message broker. Service Bus is used to decouple applications and services from each other. Review the following benefits characteristics of the service.

- Azure Service Bus supports *message queues* and *publish-subscribe topics*.
- Azure Service Bus lets you load-balance work across competing workers.
- You can use Service Bus to safely route and transfer data and control across service and application boundaries.
- Service Bus helps you coordinate transactional work that requires a high degree of reliability.

Message queues

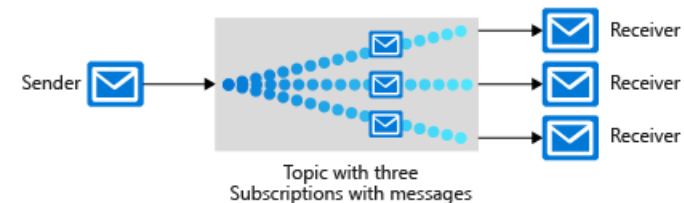
Azure Service Bus *message queues* is a message broker system built on top of a dedicated messaging infrastructure. Like Azure queues, Service Bus holds messages until the target is ready to receive them.



Azure Service Bus message queues are intended for enterprise applications, such as an app that uses communication protocols and different data contracts.


Publish-subscribe topics

Azure Service Bus *publish-subscribe topics* are like queues but can have multiple subscribers. When a message is sent to a topic, multiple components can be triggered to perform a task.



Things to consider when choosing messaging services

Each Azure messaging solution has a slightly different set of features and capabilities. You can choose one solution or use both to fulfill your design requirements. Review the following scenarios, and think about which messaging solutions can benefit the Tailwind Traders application architecture.

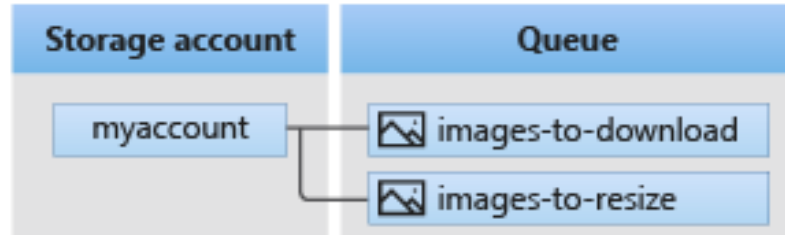
 Expand table

Messaging solution	Example scenarios
Azure Queue Storage	<i>You want a simple queue to organize messages.</i> <i>You need an audit trail of all messages that pass through the queue.</i> <i>You expect the queue storage to exceed 80 GB.</i> <i>You'd like to track progress for processing a message inside of the queue.</i>
Azure Service Bus message queues	<i>You require an at-most-once delivery guarantee.</i> <i>You require at-least-once message processing (PeekLock receive mode).</i> <i>You require at-most-once message processing (ReceiveAndDelete receive mode).</i> <i>You want to group messages into transactions.</i> <i>You want to receive messages without polling the queue.</i> <i>You need to handle messages larger than 64 KB.</i> <i>You expect the queue storage won't exceed 80 GB.</i> <i>You'd like to publish and consume batches of messages.</i>
Azure Service Bus publish-subscribe topics	<i>You need multiple receivers to handle each message.</i> <i>You expect multiple destinations for a single message but need queue-like behavior.</i>



Things to know about Azure Queue Storage

Azure Queue Storage is a service that uses Azure Storage to store large numbers of messages. Examine the following characteristics of the service.



- Queues in Azure Queue Storage can contain millions of messages.
- The number and size of queues is limited only by the capacity of the Azure storage account that owns the Queue Storage.
- Messages in Queue Storage can be securely accessed from anywhere in the world by using a simple REST-based interface.
- Queues generally provide increased reliability, guaranteed message delivery, and transactional support.



Design for Service Bus queues and topics

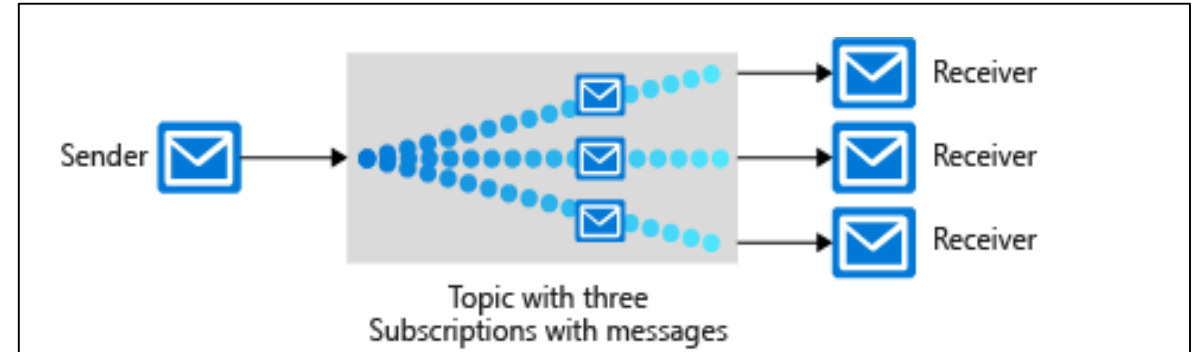
Service Bus decouples applications and services from each other.

Service bus queues



- Built on top of a dedicated messaging infrastructure
- Holds messages until the target is ready to receive them – different from queues

Service bus publish-subscribe topics



- Like bus queues but with multiple subscribers
- When a message is sent to a topic, multiple components can be triggered to perform a task

Compare messaging solutions

Solution	Usage cases	SLA
Queue storage	<ul style="list-style-type: none">• A simple queue to organize messages.• Queue to exceed 80 GB in size.• To track progress for processing a message inside of the queue.	Based on storage tier
Service bus queues	<ul style="list-style-type: none">• A first-in-first-out guarantee.• At-Least-Once message processing (PeekLock receive mode)• At-Most-Once message processing (ReceiveAndDelete receive mode)• Can group operations into transactions• Receive messages without polling the queue.• Publish and consume batches of messages.	99.9%
Service bus topics	<ul style="list-style-type: none">• Multiple receivers to handle each message.• Multiple destinations for a single message.	99.9%

Design an event solution



Design an Azure Event Hubs messaging solution

3 minutes

Certain applications produce a massive number of events from almost as many sources. These application scenarios are often referred to as *Big Data*. Big Data can require extensive infrastructure.

Suppose you're designing the architecture for a Tailwind Traders home security monitoring application. Each security system has a dozen or more cameras sensors. Before the house can be deemed secure, the sensors and cameras are connected to a test harness and put through their paces. Additionally, cached video camera footage data is streamed when the security system is connected to the datacenter monitoring headquarters.

For this architecture, you might choose a messaging solution that uses *event hubs*. Event hubs can receive and process millions of events per second. Data sent to an event hub can be transformed in real time and stored for later analysis.



Things to know about Azure Event Hubs

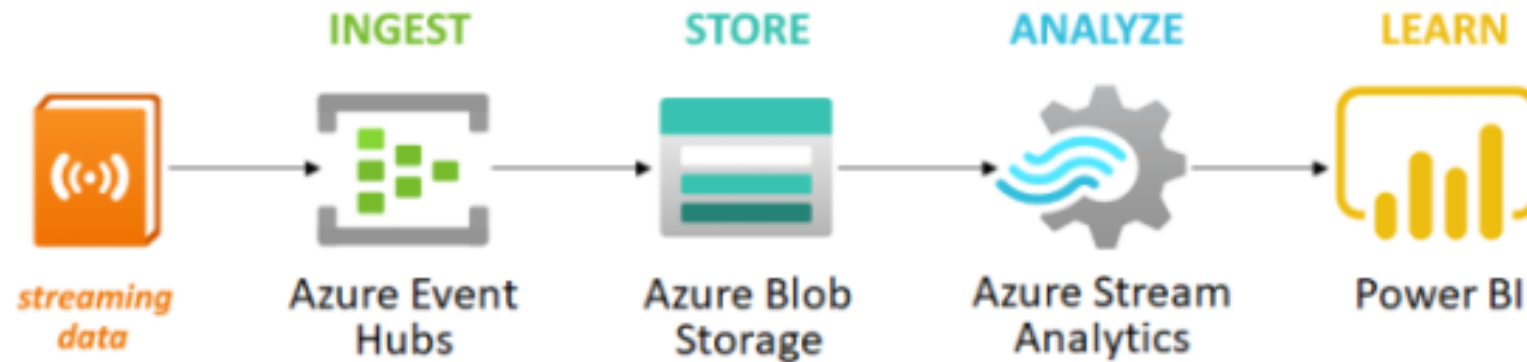
Azure Event Hubs is a fully managed, big data streaming platform and event ingestion service. Let's review the characteristics of the service:

- Azure Event Hubs supports real time data ingestion and microservices batching on the same stream.
- You can send and receive events in many different languages. Messages can also be received from Azure Event Hubs by using Apache Storm.
- Events received by Azure Event Hubs are added to the end of its data stream.
 - The data stream orders events according to the time they event is received.
 - Consumers can seek along the data stream by using time offsets.
- Event Hubs implements a *pull* model that differentiates it from other messaging services like Azure Service Bus queues.
 - Event Hubs holds each message in its cache and allows it to be read.
 - When a message is read from Event Hubs, it's not deleted. The message remains for other consumers.
- Event Hubs doesn't have a built-in mechanism to handle messages that aren't processed as expected.
- Azure Event Hubs scales according to the number of [purchased throughput \(processing\) units](#). Performance features vary for each pricing tier, such as Basic, Standard, or Premium.



Business scenario

Let's examine how Azure Event Hubs and other Azure services can contribute to the architecture for the home security monitoring application.



- Azure Event Hubs captures streaming video camera footage data from the camera and sensor testing equipment.
- Azure Blob Storage stores the video and sensor test data.
- Azure Stream Analytics identifies patterns in the video and sensor test data.
- Power BI makes decisions for monitoring alerts and improving security based on the test data patterns.



Things to consider when using Azure Event Hubs

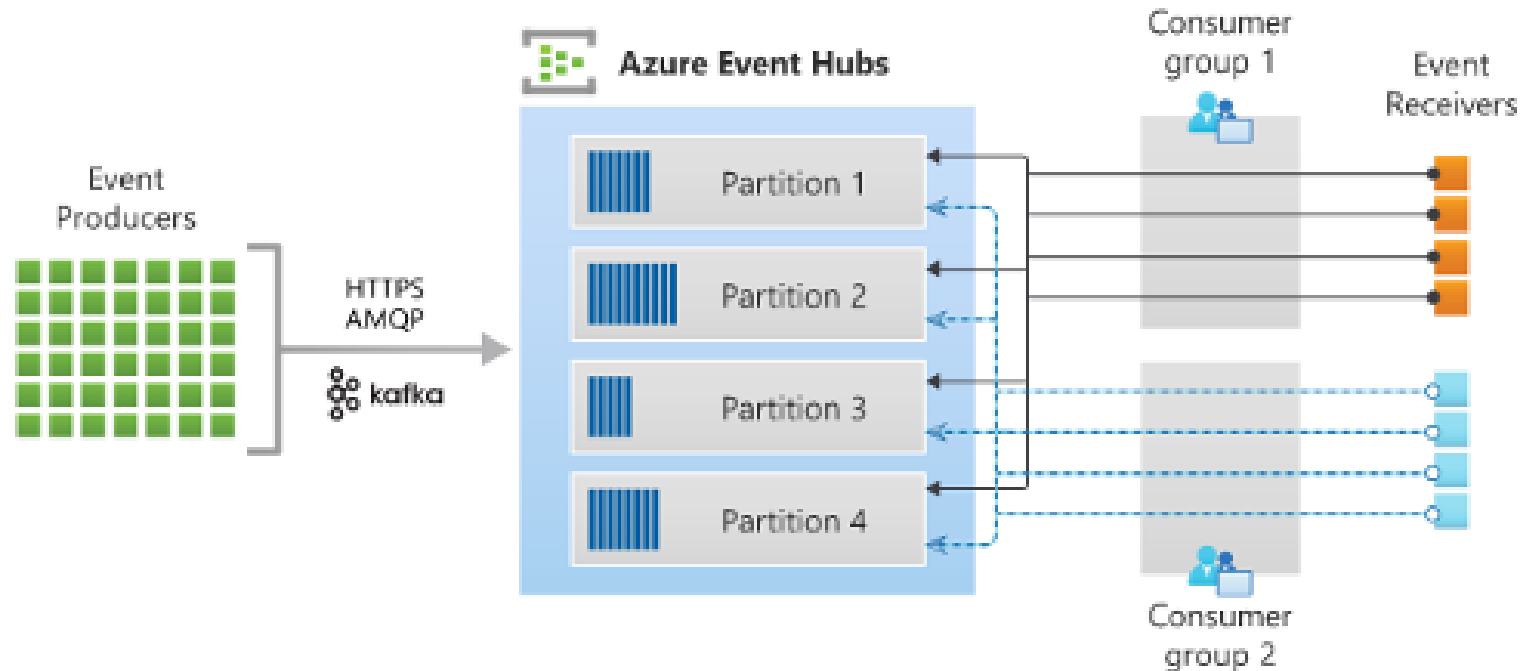
As you plan for how Azure Event Hubs can be a part of your messaging solution, consider the following points.

- **Consider common implementations.** Identify whether your application scenario is suited for event-hubs messaging. There are several common scenarios where Azure Event Hubs is a great messaging solution. Event hubs are ideal for live dashboarding, supporting analytics pipelines like clickstreams, and detecting anomalies like fraud or outlier actions. Event hubs are also a good solution for processing transactions with real-time analysis and archiving data.
- **Consider language and framework integration.** Azure Event Hubs supports sending and receiving events in many different languages. The robust language and framework support makes it easy to integrate Event Hubs with other Azure and non-Azure services.
- **Consider pricing tier and throughput units.** Choose the pricing tier that offers the features and capabilities required by your application. Control how your Azure Event Hubs implementation scales by purchasing the necessary throughput or processing units. A single throughput unit equates to:
 - **Ingress:** Up to 1 MB per second or 1,000 events per second (whichever comes first)
 - **Egress:** Up to 2 MB per second or 4,096 events per second
- **Consider pull model benefits.** Investigate how the pull model implemented by Event Hubs can benefit your application communication. Event Hubs holds a message in its cache and allows it to be read. When a message is read, it isn't deleted. The message remains for other consumers.
- **Consider message failures.** Remember Azure Event Hubs doesn't handle messages that aren't processed as expected. Suppose a message consumer malfunctions because of data format. Event Hubs won't detect this issue. The message remains until its `time-to-live` setting has expired.
- **Consider data stream access.** Event Hubs adds received events to the end of its data stream, and the events are ordered according to the time they're received. Event consumers can seek along the data stream by using time offsets.

Design an Event Hub messaging solution

Azure Event Hubs is a fully managed, real time data ingestion service

- Orders events by when they are received - by time offsets
- Uses a pull model allowing multiple reads from consumers
- Scaling is controlled by how many throughput units or processing units you purchase
- Receiving real-time streaming data



Design an event-driven solution

4 minutes

An event-driven architecture enables you to connect to the core application without needing to modify the existing code. When an event occurs, you can react with specific code to respond to the event. An event-driven application uses the *send and forget* principle. An event is sent toward the next system, which can be another service, an event hub, a stream, or a message broker.

Let's reconsider our design for the Tailwind Traders product demo application, and examine how to use a Web API that runs in Azure. When a new product review or demo video is uploaded, we need to notify all mobile apps on user devices around the world that are interested in the products. Azure Event Grid is an ideal solution for this requirement.

- The publisher of the review or video doesn't need to know about any subscribers who are interested in the affected products.
- We want to have a one-to-many relationship where we can have multiple subscribers. Subscribers can optionally decide whether they're interested in the affected products.

Things to know about Azure Event Grid

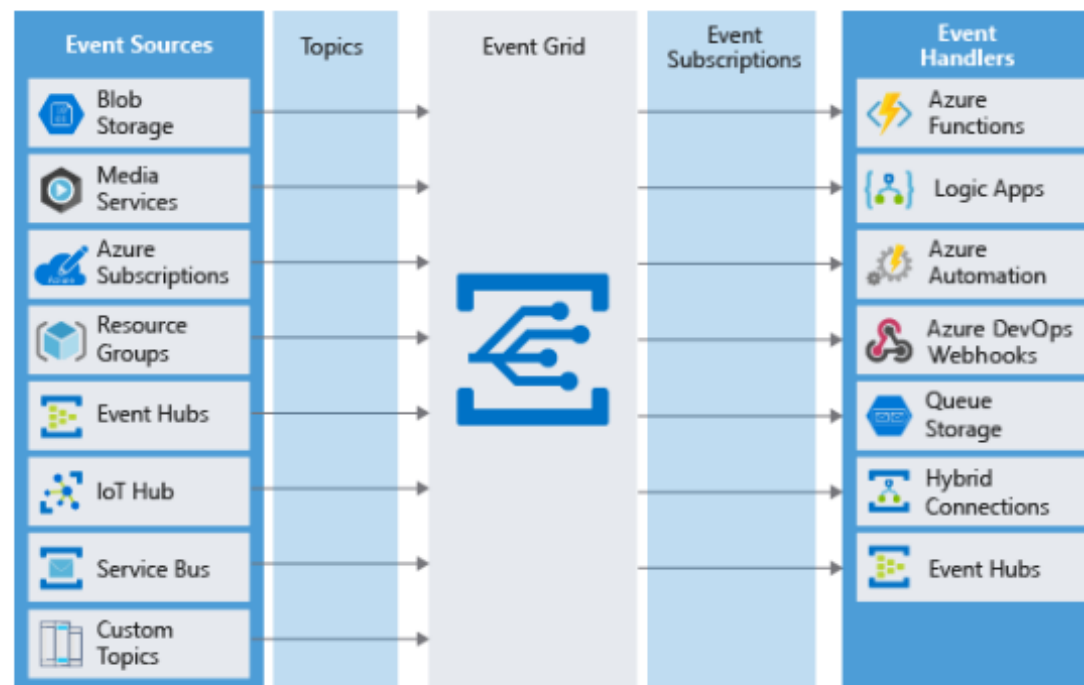
[Azure Event Grid](#) is a fully managed event routing service that runs on [Azure Service Fabric](#). Event Grid exists to make it easier to build event-based and serverless applications on Azure. Examine the following characteristics of the service.

- Azure Event Grid aggregates all your events and provides routing from any source to any destination.
- Event Grid distributes events from sources like Azure Blob Storage accounts and Azure Media Services.
- Events are distributed to handlers like Azure Functions and Azure DevOps Webhooks.
- The service manages the routing and delivery of events from many sources. The management helps to minimize cost and latency by eliminating the need for polling.



How Azure Event Grid works

The following illustration shows how Azure Event Grid manages the event process from multiple event sources to multiple event handlers.



- An event source such as Azure Blob Storage tags events with one or more topics, and sends events to Azure Event Grid.
- An event handler such as Azure Functions subscribes to topics they're interested in.
- Event Grid examines topic tags to decide which events to send to which handlers.
- Event Grid forwards relevant events to subscribers.
- Event Grid sends an event to indicate something has happened or changed. However, the actual object that was changed (text file, video, audio, and so on) isn't part of the event data. Instead, Event Grid passes a URL or identifier to reference the changed object.



Things to consider when using Azure Event Grid

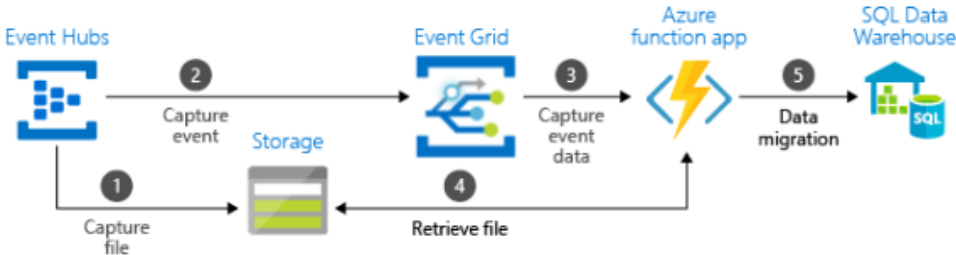
Azure Event Grid can be an ideal solution for an event-driven application architecture. As you review the following considerations, think about how Event Grid can benefit the Tailwind Traders application architecture.

- **Consider multiple services.** Choose one or multiple Azure services to fulfill your design requirements.

Expand table

Azure service	Purpose	Message or Event	Usage scenario
Azure Event Grid	Reactive programming	Event distribution (discrete)	React to status changes
Azure Event Hubs	Big data pipeline	Event streaming (series)	Conduct telemetry and distributed data streaming
Azure Service Bus	High-value enterprise messaging	Message	Fulfill order processing and financial transactions

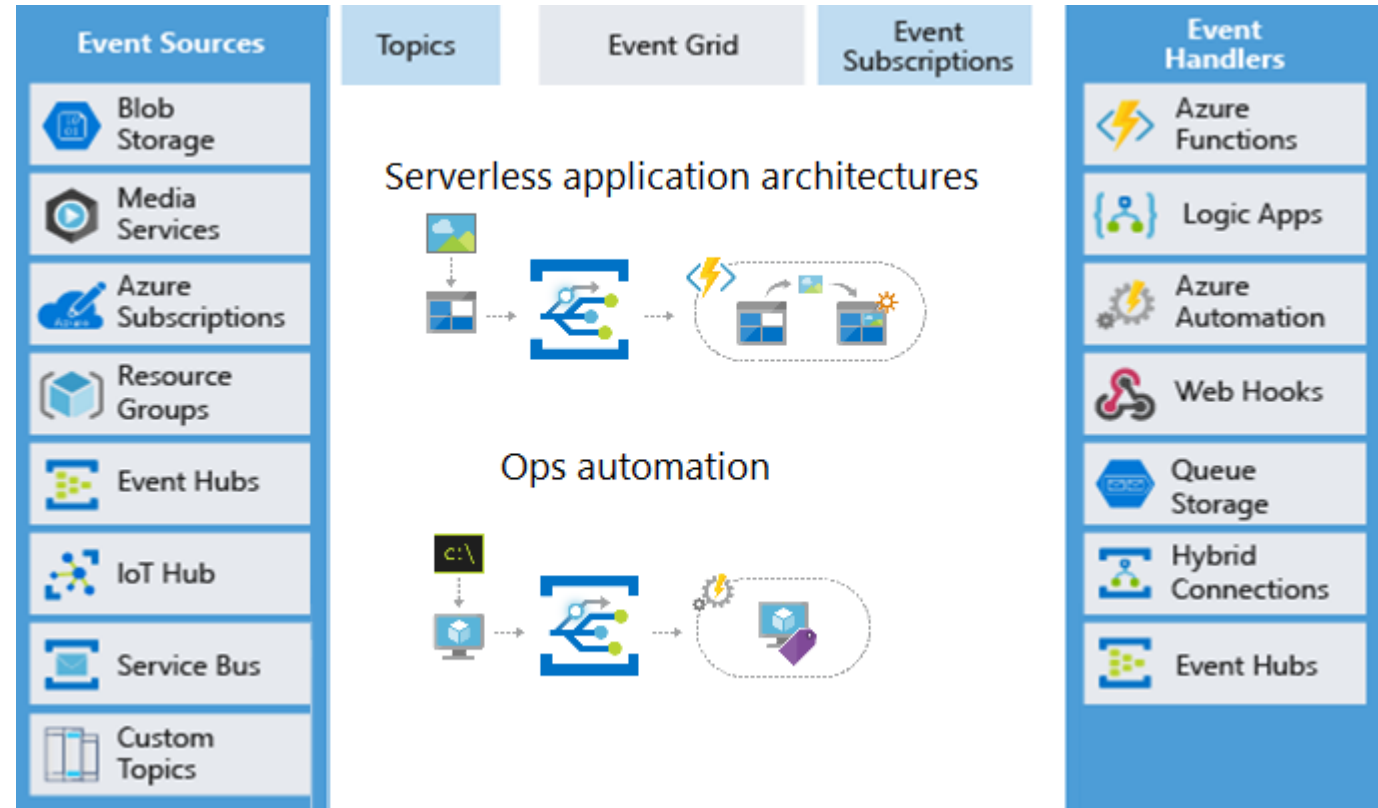
- **Consider distinct roles for services.** Investigate using Azure services side by side to fulfill distinct roles. An e-commerce site can use Azure Service Bus to process an order, Azure Event Hubs to capture site telemetry, and Azure Event Grid to respond to events like an item being shipped.
- **Consider linking services.** Link Azure services together to form an event and data pipeline stream. In this scenario, Azure Event Grid responds to events in other services. The following illustration demonstrates how several Azure services can be linked together as an event and data pipeline to stream data.



Design an event-driven solution

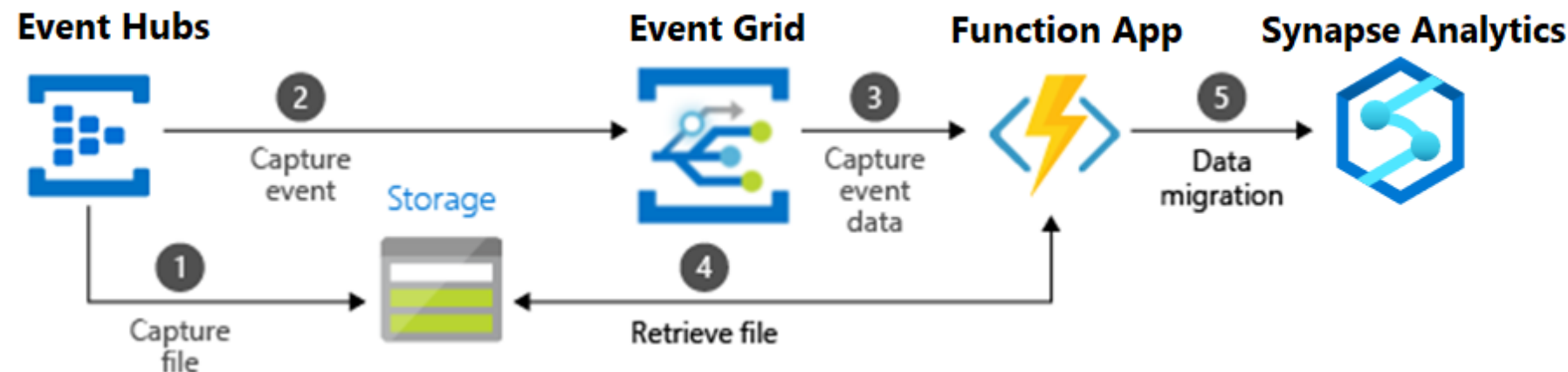
Azure Event Grid is a routing service connecting data sources with event handlers.

- Events sources include Azure resources or custom topics (you create)
- Event handlers react to an event
- Useful for serverless applications and operations automation
- Uses a pay-per-operation or pay-per-use pricing models



Comparison of message and event solutions

Consider combining several solutions



Service	Purpose	Type	When to use
Event Grid	Reactive programming	Event distribution (discrete)	React to status changes
Event Hubs	Big data pipeline	Event streaming (series)	Telemetry and distributed data streaming
Service Bus	High-value enterprise messaging	Message	Order processing and financial transactions
Storage Queues	Large-volume messaging queues	Message	Cost-effective, simple messaging mechanism

Design an IoT Hub solution

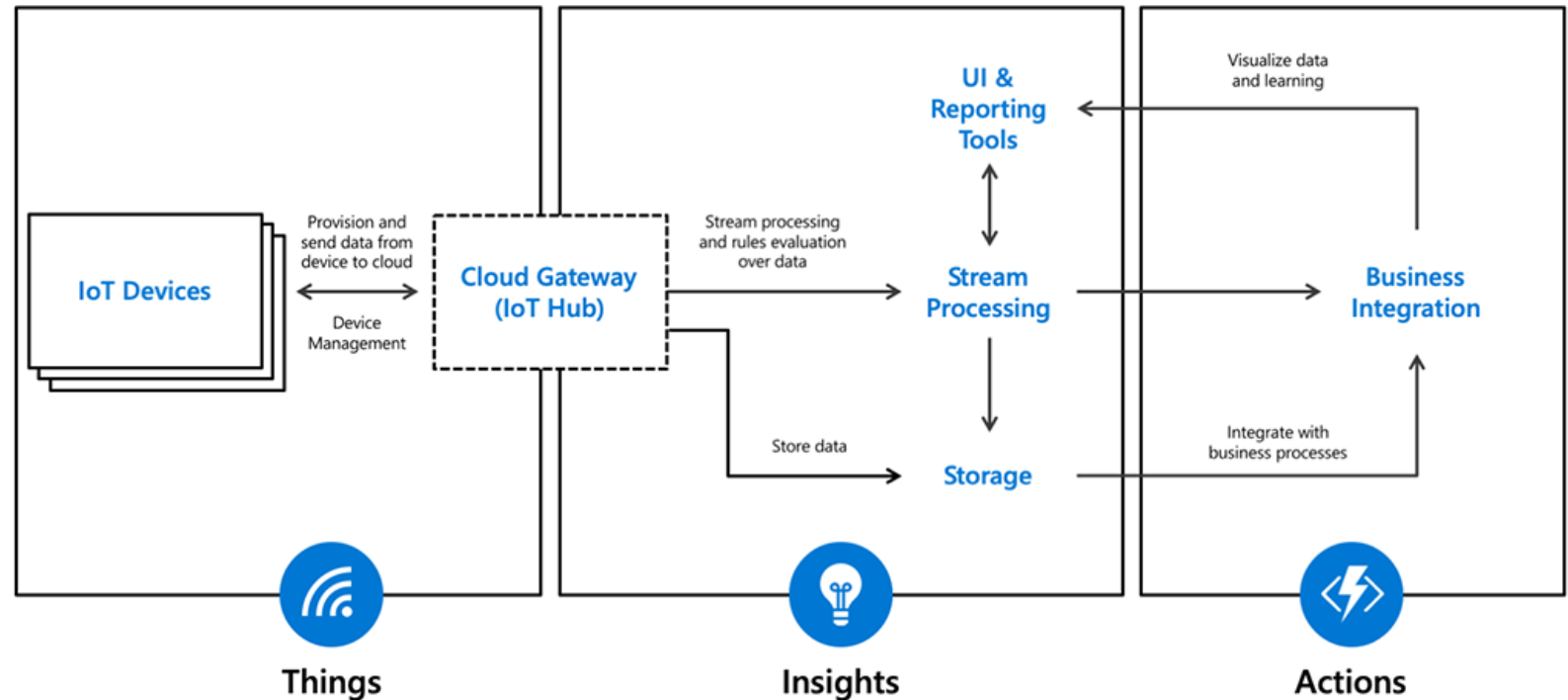
Central message hub for IoT applications and its attached devices.

When to use IoT Hub?

- Application complexity
- Data throughput
- Securing solution end to end allowing for per-device authentication
- Bi-directional communication

Capabilities over Event Hub:

- Per-device identity
- File upload from devices
- Device provisioning service



Design an application
optimization solution

Design a caching solution

4 minutes

Caching is a common technique that aims to improve the performance and scalability of a system. Caching temporarily copies frequently accessed data to fast storage located close to the application. When the fast data storage is located closer to an application than its original data store, caching can significantly improve response times for client applications by serving data more quickly.

Caching is most effective when a client instance repeatedly reads the same data, especially when the following conditions apply to the original data store:

- The original data store remains relatively static.
- It's slow compared to the speed of the cache.
- It's subject to a high level of contention.
- It's far away, and network latency can result in slow access to the store.

Suppose Tailwind Traders is adding a new feature to the product demo application to increase customer traffic to their retail website. The event feature adds a banner to the top of the mobile app to announce special offers and limited product discounts. New offers are posted on the hour, and the remaining product availability for each offer is updated after every order is processed. The first customer to respond to a new offer receives a double discount! Customers are encouraged to check their mobile app frequently for updates to the offers and product availability. To implement this new feature, you need to design a caching solution that can support in-memory fast read and writes.



Things to know about Azure Cache for Redis

Azure Cache for Redis provides an in-memory data store based on the Redis software. Redis improves the performance and scalability of an application that uses back-end data stores heavily. It's able to process large volumes of application requests by keeping frequently accessed data in the server memory, which can be written to and read from quickly. Redis brings a critical low-latency and high-throughput data storage solution to modern applications.

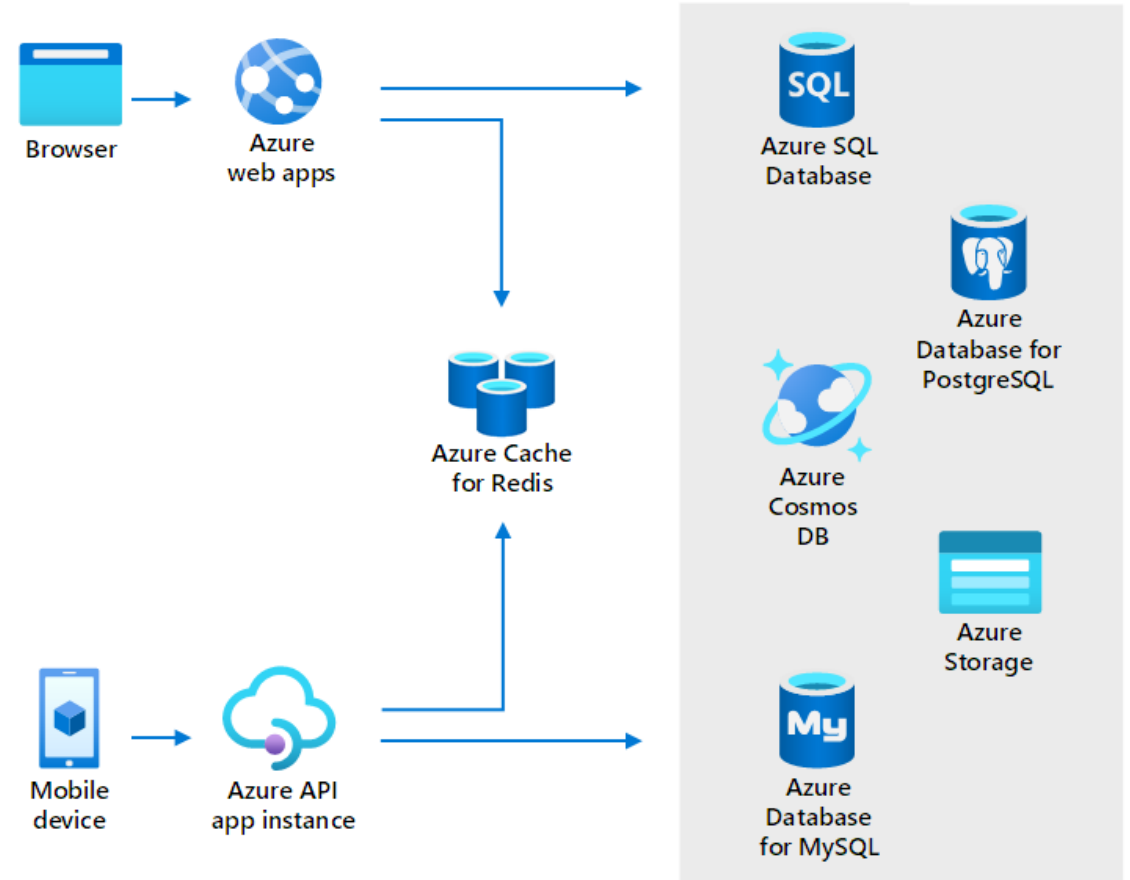
Let's review the characteristics of the service:

- Azure Cache for Redis offers two implementation options for developers:
 - The Redis open source (OSS Redis)
 - A commercial product from Redis Labs (Redis Enterprise) as a managed service
- Azure Cache for Redis provides secure and dedicated Redis server instances and full Redis API compatibility.
- You can use Azure Cache for Redis as a distributed data or content cache, session store, or message broker.
- Deploy Azure Cache for Redis as a standalone or with other Azure database services, such as Azure SQL or Azure Cosmos DB.

When to use Azure Cache for Redis

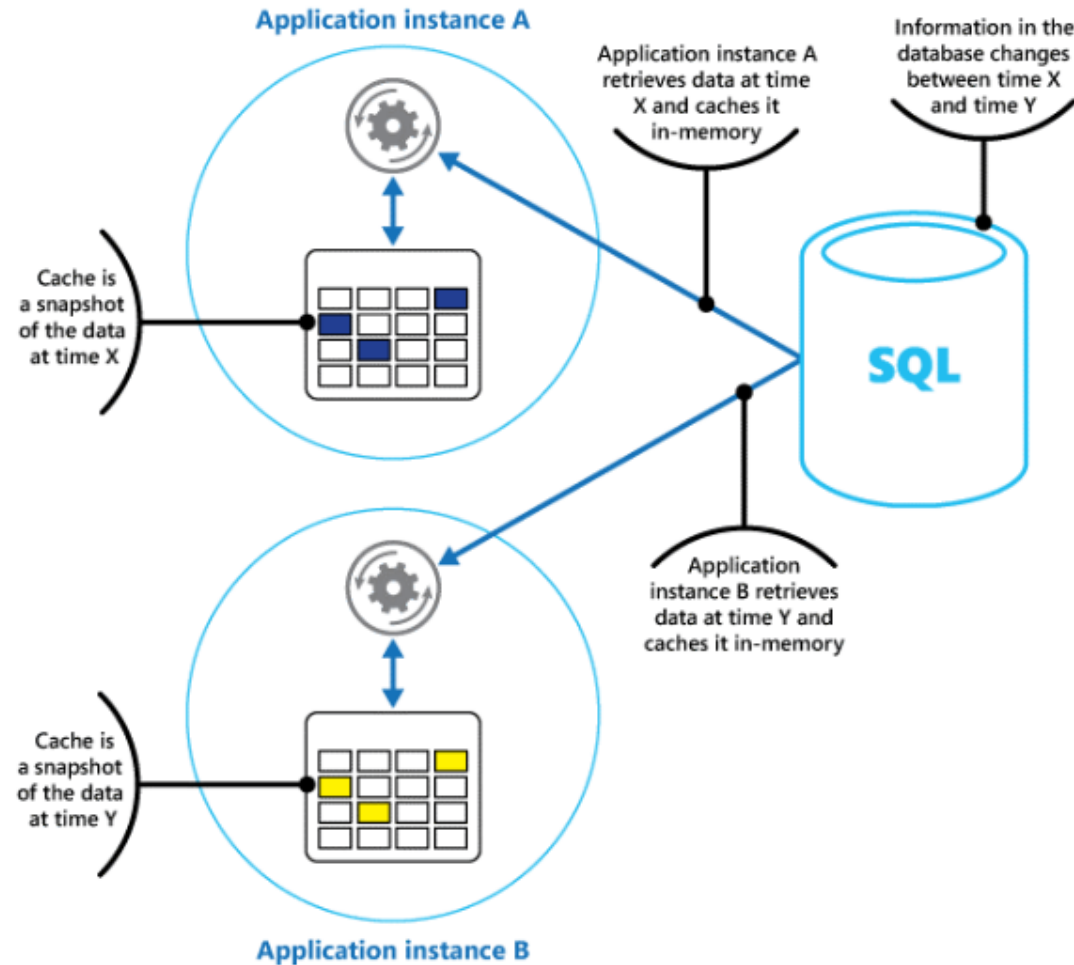
Store frequently accessed data so that applications can be responsive to users.

- Key scenarios - data cache, content cache, session store, job and message queuing, and distributed transactions
- Fully managed solution
- High availability - responds automatically to both anticipated and unanticipated changes in demand
- Same performance and scaling benefits throughout the world – network isolation, data encryption in transit



How Azure Cache for Redis works

Azure Cache for Redis is operated by Microsoft, hosted on Azure, and usable by any application within or outside of Azure. The following illustration shows how Azure Cache for Redis works in applications.



Application instance A has a cache with a snapshot of the data at time **X**. It retrieves data at time **X** and caches it in-memory. Application instance B has a cache with a snapshot of the data at time **Y**. It retrieves data at time **Y** and caches it in-memory. Information in the SQL database changes between time **X** and time **Y**.



Things to consider when using Azure Cache for Redis

Azure Cache for Redis improves application performance by supporting common application architecture patterns.

As you review the following patterns, consider patterns that might be exhibited in the Tailwind Traders application architecture. Think about how Azure Cache for Redis can supply the pattern requirements.

Pattern	Scenario	Solution
Data cache	Databases are often too large to load directly into a cache.	It's common to use the <i>cache-aside</i> pattern to only load data into the cache as needed. When the system makes changes to the data, the system can also update the cache, which is then distributed to other clients. Additionally, the system can set an expiration on data, or use an eviction policy to trigger data updates into the cache.
Content cache	Many web pages are generated from templates that use static content such as headers, footers, banners. These static items shouldn't change often.	Using an in-memory cache provides quick access to static content compared to back-end datastores. This pattern reduces processing time and server load and allows web servers to be more responsive. A content cache can allow you to reduce the number of servers needed to handle loads. Azure Cache for Redis provides the <i>Redis Output Cache Provider</i> to support this pattern with ASP.NET.
Session store	A session store is commonly used with shopping carts and other user history data that a web application might associate with user cookies. Storing too much in a cookie can have a negative effect on performance as the cookie size grows and is passed and validated with every request.	A typical solution uses the cookie as a key to query the data in a database. It's faster to use an in-memory cache like Azure Cache for Redis to associate information with a user than interacting with a full relational database.
Job and message queuing	Some application operations take significant time to complete, which might prevent other unrelated jobs or messages from starting.	Applications often add tasks to a queue when the operations associated with the request take time to execute. Longer running operations are queued to be processed in sequence, often by another server. This method of deferring work is called <i>task queuing</i> . Azure Cache for Redis provides a distributed queue to enable this pattern in your application.
Distributed transactions	Applications sometimes require a series of commands against a back-end datastore to execute as a single atomic operation. All commands must succeed, or all commands must be rolled back to the initial state.	Azure Cache for Redis supports executing a batch of commands as a single transaction.

Design API integration

4 minutes

Publishing an API is a great way to increase market share, generate revenue, and foster innovation. However, maintaining even one API brings significant challenges, such as onboarding users, managing revisions, and implementing security.

Developers need a way to reduce the complexity involved in supporting numerous APIs and their management. They require an API Management technology that can serve as a *front door* for all their APIs. They need tools to implement security, manage revisions, and perform analytics.

Suppose Tailwind Traders has started a new service for premier subscribers to request home delivery, product assembly, and in person assistance for promoted products. The service is available in the mobile app and the online retail website. Customers browse the list of promoted products, and place their order. Tailwind Traders connects with contracted specialists who deliver and assemble the product, and provide the customer with hands on instruction.

The backbone of the new service is a large collection of published APIs, some of which are used by the following entities:

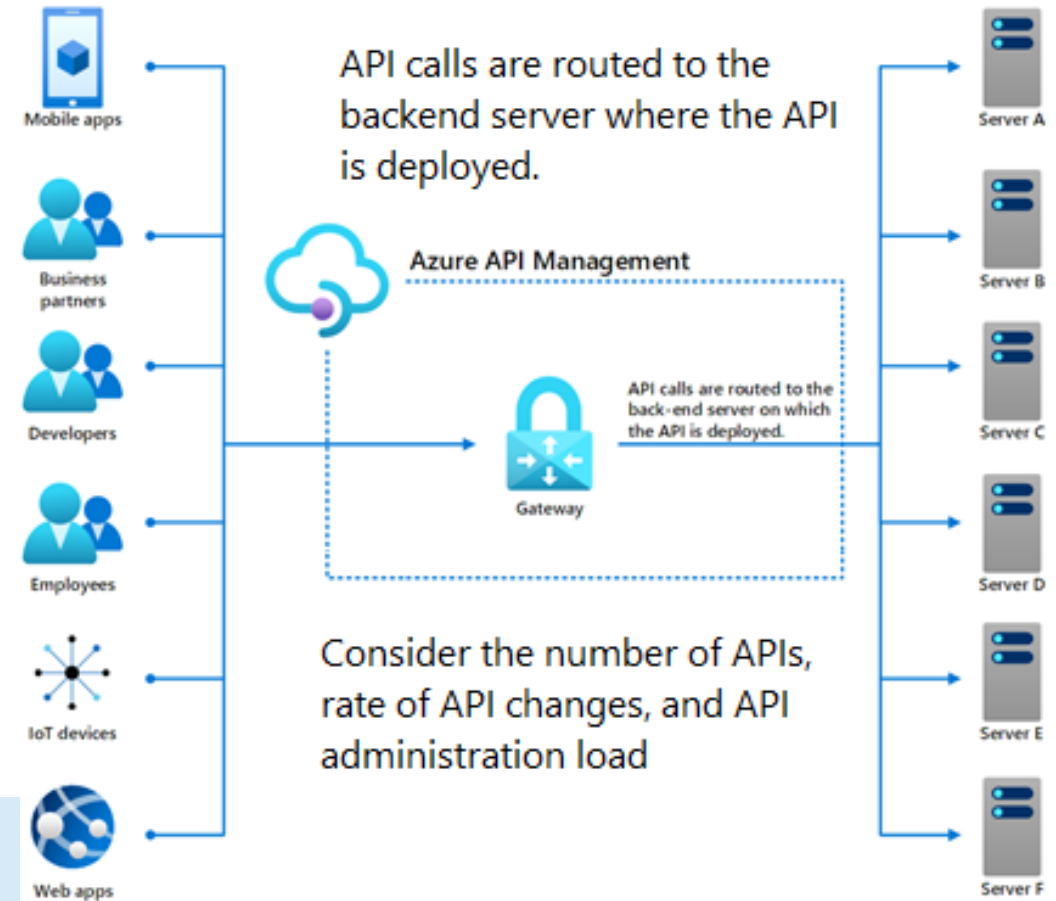
- Tailwind Traders mobile app and online website
- IoT devices on the delivery vehicles
- Vendor product specialists
- Tailwind Traders in-house development teams
- Tailwind Traders employees, such as business analysts

Each published API resides on a different server. Each API has its own process for onboarding users, and its own policies for security, revisions, analytics, and more. You're looking for an Azure solution that can help reduce this complexity.

Design an Azure API management solution

Publish, secure, maintain, and analyze all your company's APIs.

- Bring multiple APIs under a single administrative umbrella – centralized management
- Manage permissions and access
- Ensure compliance across API
- Standardize API specs
- Protect the APIs from malicious usage



❗ Important

Azure API Management doesn't host your actual APIs. Your APIs remain where they were originally deployed.

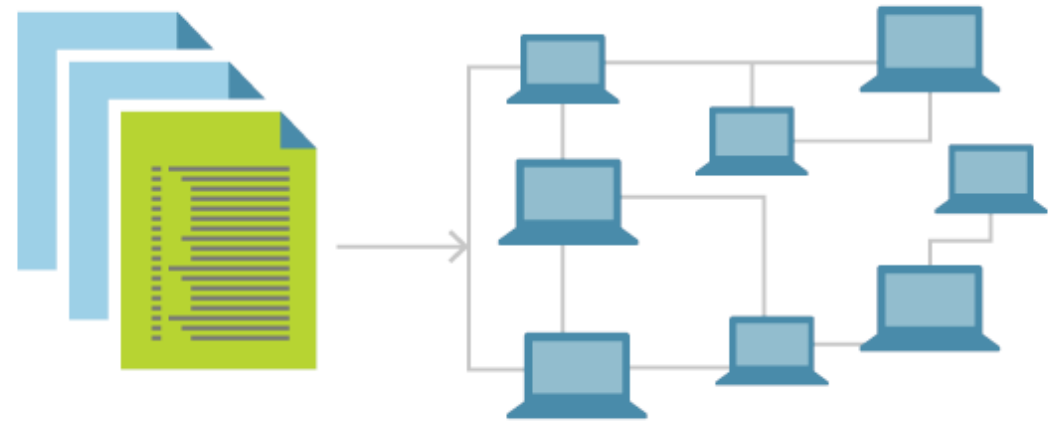
Azure API Management serves as a front door for your APIs. In this way, Azure API Management is said to **decouple** your APIs. You set API policies and other management options in Azure, while leaving your deployed back-end APIs untouched.

Design an application
lifecycle

What is Infrastructure as Code?

Infrastructure as Code (IaC) is the process of automating your infrastructure provisioning.

- The IaC model generates the same environment every time it is applied
- Solves the problem of environmental drift
- Enables teams to test applications in production-like environments early
- Where possible, uses declarative definition files



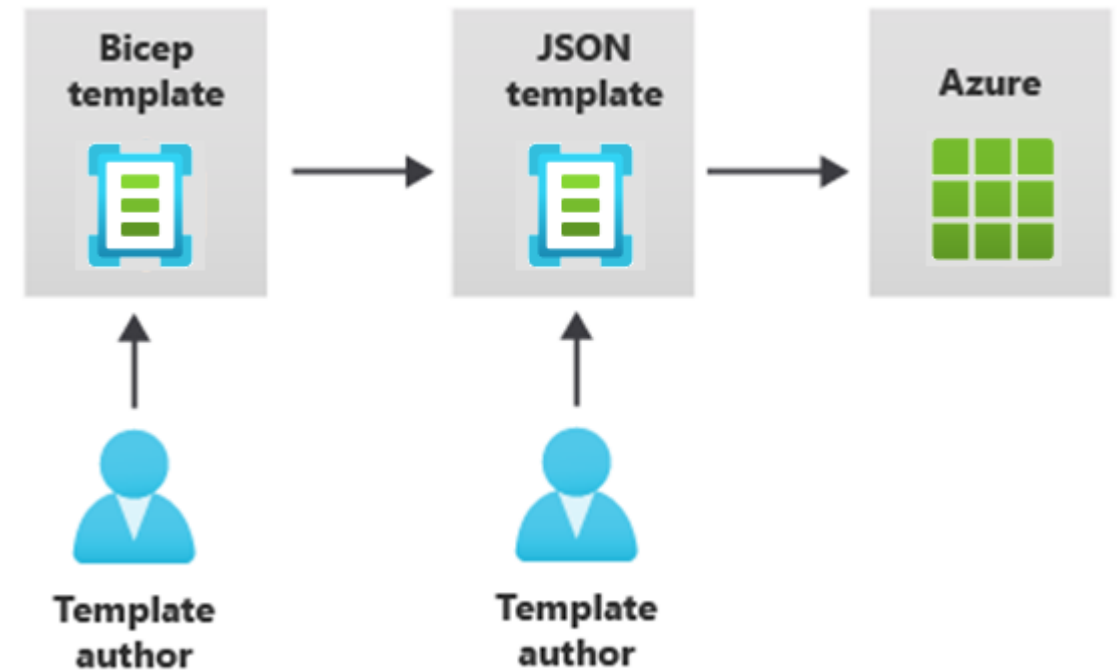
With the move to the cloud, many teams have adopted agile development methods. These teams must iterate quickly and repeatedly deploy their solutions to the cloud. Teams must be assured their infrastructure is in a reliable state. Application code must be managed through a unified process.

To meet these challenges in your design for Tailwind Traders, you're investigating how to automate deployments by using the practice of [infrastructure as code](#). Let's explore two Azure solutions for deployment and automation of your applications: Azure Resource Manager templates and Azure Automation.

Provision resources with Infrastructure as Code


Azure supports IaC with Azure Resource Manager and third-party platforms.

- Azure Resource Manager templates – Bicep, JSON
- Azure Automation
- Azure DevOps services
- GitHub actions
- Terraform
- Jenkins



Things to know about Azure Automation

Azure Automation delivers a cloud-based automation and configuration service that supports consistent management across your Azure and non-Azure environments. Azure Automation gives you complete control in three service areas: process automation, configuration management, and update management. Let's examine the details of this service, and consider how it can be implemented in the Tailwind Traders application architecture.

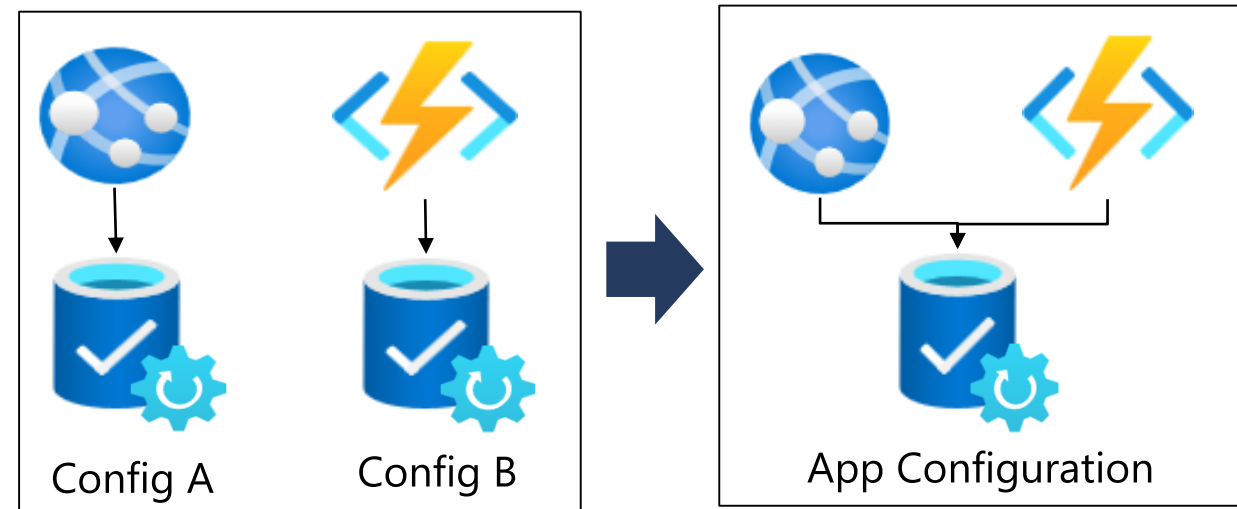
 Expand table

Service	Description
Process automation	Process automation enables you to automate frequent, time-consuming, and error-prone cloud management tasks. This service helps you focus on work that adds business value. By reducing errors and boosting efficiency, it also helps to lower your operational costs. The service allows you to author runbooks graphically in PowerShell or by using Python.
Configuration management	Configuration management enables access to two features, Change Tracking and Inventory and Azure Automation State Configuration. The service supports change tracking across services, daemons, software, registry, and files in your environment. The change tracking helps you diagnose unwanted changes and raise alerts.
Update management	The update management service includes the Update Management feature for Windows and Linux systems across hybrid environments. The feature allows you to create scheduled deployments that orchestrate the installation of updates within a defined maintenance window.

Design an Azure App Configuration solution

Azure App Configuration centrally manages application settings and feature flags.

- Flexible key representations and mappings
- Point-in-time replay of settings
- Dedicated UI for feature flag management
- Comparison of two sets of configurations on custom-defined dimensions
- Enhanced security through Azure-managed identities and encryption



Traditionally, shipping a new application feature requires a complete redeployment of the application itself. Testing or deployment of a feature often requires multiple versions of the application. Each deployment might require different configurations, credentials, changing settings or parameters for testing.

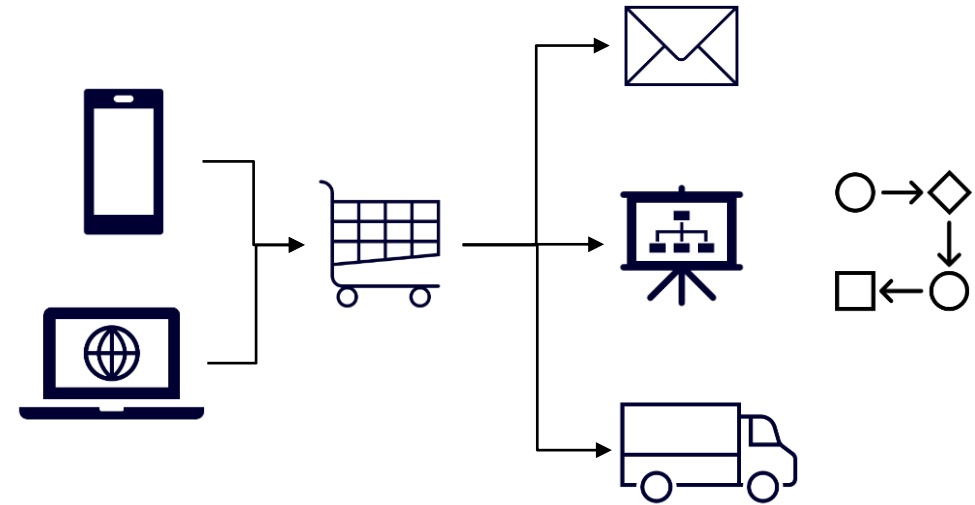
Configuration management is a modern software-development practice that **decouples configuration from code deployment and enables quick changes to feature availability on demand**. Decoupling configuration as a service enables systems to dynamically administer the deployment lifecycle.

Case study and review

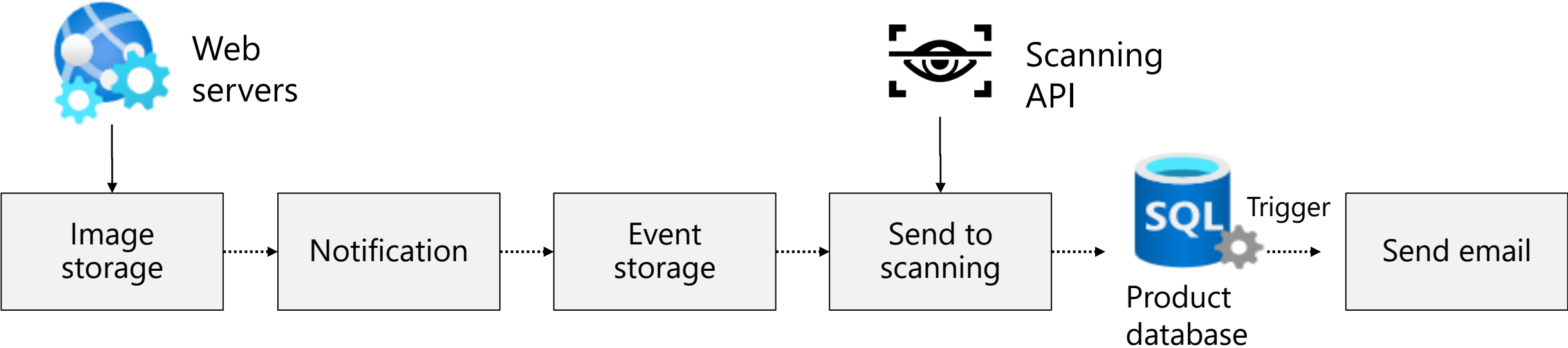
Case Study – Application architecture

A new product catalog design

- New product catalog, ordering process, and shopping cart
- Services will rely on a combination of relational and non-relational data
- It is critical that the service hosting the application supports rapid autoscaling and high availability



case study discussion



Event Grid



Function



Logic App



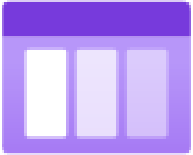
Blob



Automation

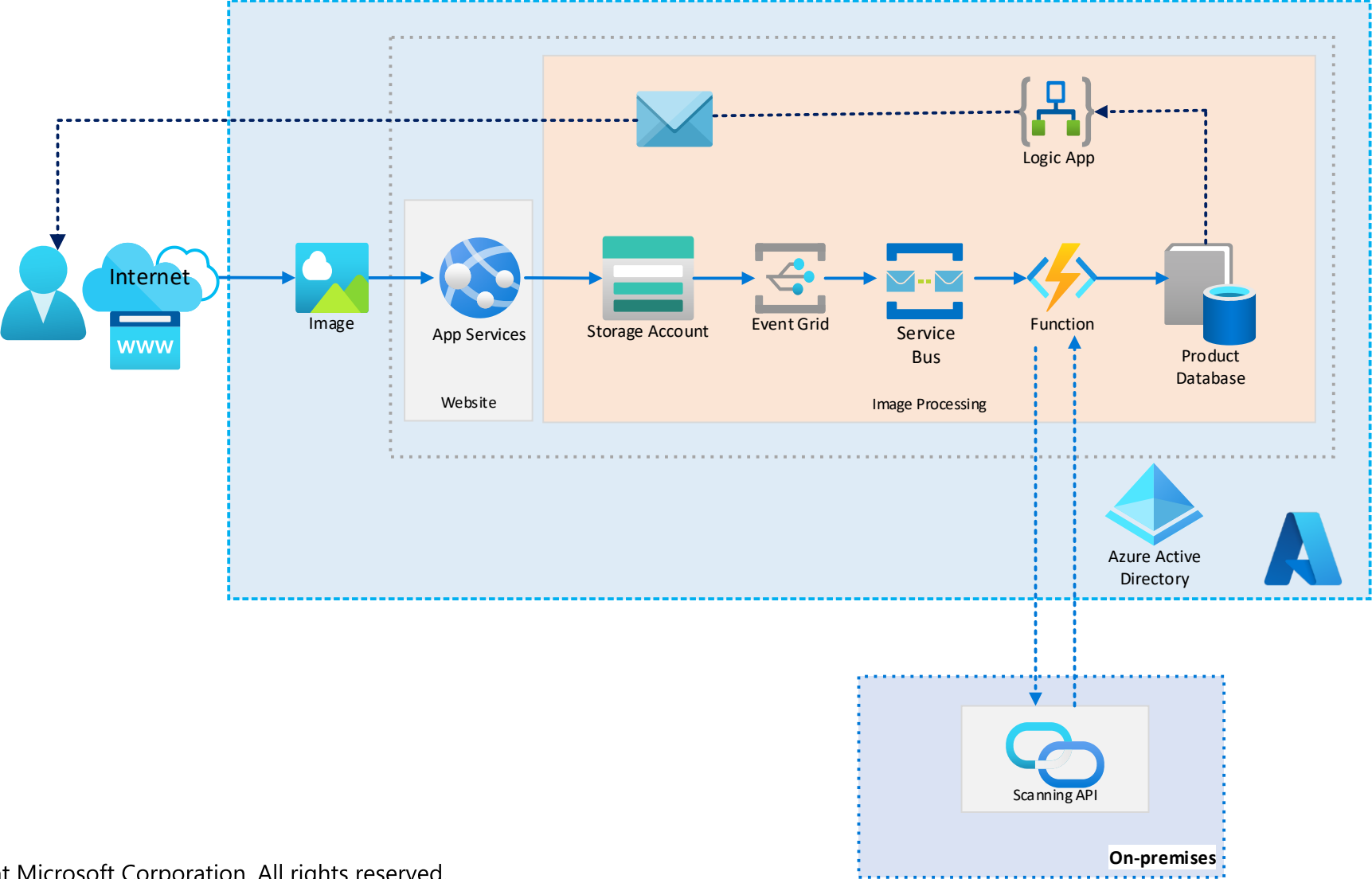


Service Bus



Queues

Solution Diagram



End of presentation