Scalable order processing on Azure

07/10/2018 • 6 minutes to read • Contributors all

In this article

This example scenario is relevant to organizations that need a highly scalable and resilient architecture for online order processing. Potential applications include e-commerce and retail point-of-sale, order fulfillment, and inventory reservation and tracking.

This scenario takes an event sourcing approach, using a functional programming model implemented via microservices. Each microservice is treated as a stream processor, and all business logic is implemented via microservices. This approach enables high availability and resiliency, geo-replication, and fast performance.

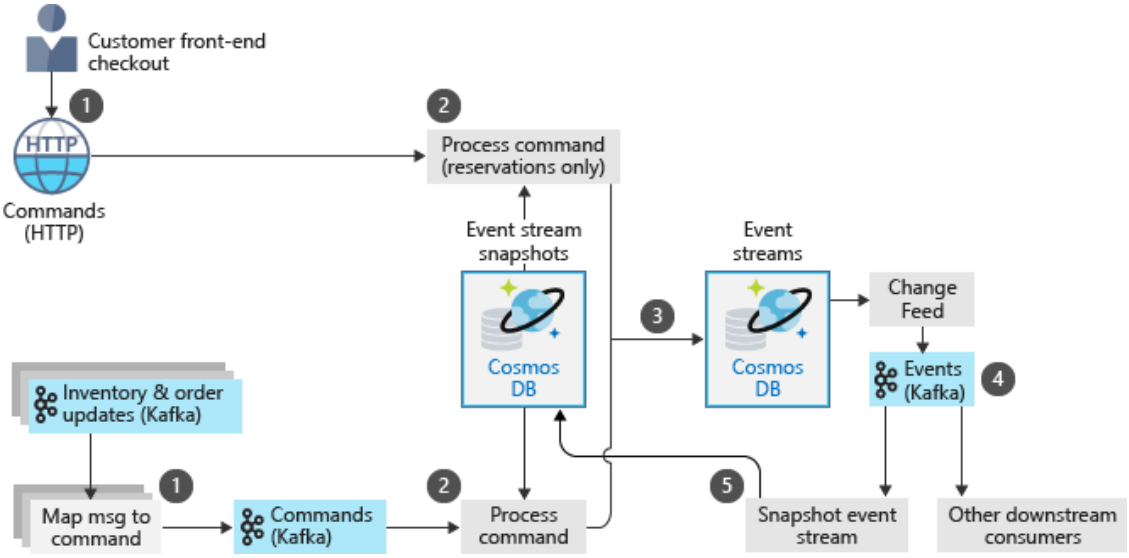
Using managed Azure services such as Cosmos DB and HDInsight can help reduce costs by leveraging Microsoft's expertise in globally distributed cloud-scale data storage and retrieval. This scenario specifically addresses an e-commerce or retail scenario; if you have other needs for data services, you should review the list of available fully managed intelligent database services in Azure.

Relevant use cases

Other relevant use cases include:

- E-commerce or retail point-of-sale back-end systems.
- Inventory management systems.
- Order fulfillment systems.
- Other integration scenarios relevant to an order processing pipeline.

Architecture



This architecture details key components of an order processing pipeline. The data flows through the scenario as follows:

1. Event messages enter the system via customer-facing applications (synchronously over HTTP) and various back-end systems (asynchronously via Apache Kafka). These messages are passed into a command processing pipeline.
2. Each event message is ingested and mapped to one of a defined set of commands by a command processor microservice. The command processor retrieves any current state relevant to executing the command from an event stream snapshot database. The command is then executed, and the output of the command is emitted as a new event.
3. Each event emitted as the output of a command is committed to an event stream database using Cosmos DB.
4. For each database insert or update committed to the event stream database, an event is raised by the Cosmos DB Change Feed. Downstream systems can subscribe to any event topics that are relevant to that system.
5. All events from the Cosmos DB Change Feed are also sent to a snapshot event stream microservice, which calculates any state changes caused by events that have occurred. The new state is then committed to the event stream snapshot database stored in Cosmos DB. The snapshot database provides a globally distributed, low latency data source for the current state of all data elements. The event stream database provides a complete record of all event messages that have passed through the architecture, which enables robust testing, troubleshooting, and disaster recovery scenarios.

Components

- Cosmos DB is Microsoft's globally distributed, multi-model database that enables your solutions to elastically and independently scale throughput and storage across any number of geographic regions. It offers throughput, latency, availability, and consistency guarantees with comprehensive service level agreements (SLAs). This scenario uses Cosmos DB for event stream storage and snapshot storage, and leverages Cosmos DB's Change Feed features to provide data consistency and fault recovery.
- Apache Kafka on HDInsight is a managed service implementation of Apache Kafka, an open-source distributed streaming platform for building real-time streaming data pipelines and applications. Kafka also provides message broker functionality similar to a message queue, for publishing and subscribing to named data streams. This scenario uses Kafka to process incoming as well as downstream events in the order processing pipeline.

Considerations

Many technology options are available for real-time message ingestion, data storage, stream processing, storage of analytical data, and analytics and reporting. For an overview of these options, their capabilities, and key selection criteria, see Big data architectures: Real-time processing in the Azure Data Architecture Guide.

Microservices have become a popular architectural style for building cloud applications that are resilient, highly scalable, independently deployable, and able to evolve quickly. Microservices require a different approach to designing and building applications. Tools such as Docker, Kubernetes, Azure Service Fabric, and Nomad enable the development of microservices-based architectures. For guidance on building and running a microservices-based architecture, see Designing microservices on Azure in the Azure Architecture Center.

Availability

This scenario's event sourcing approach allows system components to be loosely coupled and deployed independently of one another. Cosmos DB offers high availability and helps organization manage the tradeoffs associated with consistency, availability, and performance, all with corresponding guarantees. Apache Kafka on HDInsight is also designed for high availability.

Azure Monitor provides unified user interfaces for monitoring across various Azure services. For more information, see Monitoring in Microsoft Azure. Event Hubs and Stream Analytics are both integrated with Azure Monitor.

For other availability considerations, see the availability checklist.

Scalability

Kafka on HDInsight allows configuration of storage and scalability for Kafka clusters. Cosmos DB provides fast, predictable performance and scales seamlessly as your application grows. The event sourcing microservices-based architecture of this scenario also makes it easier to scale your system and expand its functionality.

For other scalability considerations, see the scalability checklist available in the Azure Architecture Center.

Security

The Cosmos DB security model authenticates users and provides access to its data and resources. For more information, see Cosmos DB database security.

For general guidance on designing secure solutions, see the Azure Security Documentation.

Resiliency

The event sourcing architecture and associated technologies in this example scenario make this scenario highly resilient when failures occur. For general guidance on designing resilient solutions, see Designing resilient applications for Azure.

Pricing

To examine the cost of running this scenario, all of the services are pre-configured in the cost calculator. To see how pricing would change for your particular scenario, change the appropriate variables to match your expected data volume. For this scenario, the example pricing includes only Cosmos DB and a Kafka cluster for processing events raised from the Cosmos DB Change Feed. Event processors and microservices for originating systems and other downstream systems are not included, and their cost is highly dependent on the quantity and scale of these services as well as the technologies chosen for implementing them.

The currency of Azure Cosmos DB is the request unit (RU). With request units, you don't need to reserve read/write capacities or provision CPU, memory, and IOPS. Azure Cosmos DB supports various APIs that have different operations, ranging from simple reads and writes to complex graph queries. Because not all requests are equal, requests are assigned a normalized quantity of request units based on the amount of computation required to serve the request. The number of request units required by your solution is dependent on data element size and the number of database read and write operations per second. For more information, see Request units in Azure Cosmos DB. These estimated prices are based on Cosmos DB running in two Azure regions.

We have provided three sample cost profiles based on amount of activity you expect:

- Small: this pricing example correlates to 5 RUs reserved with a 1 TB data store in Cosmos DB and a small (D3 v2) Kafka cluster.
- Medium: this pricing example correlates to 50 RUs reserved with a 10 TB data store in Cosmos DB and a midsized (D4 v2) Kafka cluster.
- Large: this pricing example correlates to 500 RUs reserved with a 30 TB data store in Cosmos DB and a large (D5 v2) Kafka cluster.

Related resources

This example scenario is based on a more extensive version of this architecture built by Jet.com for its end-to-end order processing pipeline. For more information, see the jet.com technical customer profile and jet.com's presentation at Build 2018.

Other related resources include:

- *Designing Data-Intensive Applications* by Martin Kleppmann (O'Reilly Media, 2017).

- *Domain Modeling Made Functional: Tackle Software Complexity with Domain-Driven Design and F#* by Scott Wlaschin (Pragmatic Programmers LLC, 2018).
- Other Cosmos DB use cases
- Real time processing architecture in the Azure Data Architecture Guide.