Azure Queue Storage vs Azure Service Bus

Aspect	Azure Queue Storage	Azure Service Bus Queues
Purpose / Use case	Simple queuing for decoupling, buffering, load leveling	Full-fledged messaging with advanced features (transactions, routing, Pub/Sub, etc.)
Max message size	64 KB (≈48 KB if Base64-encoded)	~256 KB in Standard; higher (up to 1 MB+ or 100 MB in Premium)
Queue / storage limits	Very large — limited by storage account (hundreds of TB)	Limited size per queue (1 GB to ~80 GB depending on tier)
Message TTL / retention	Up to 7 days (older versions)	Unlimited / TimeSpan.Max by default if not set
Delivery semantics	At-least-once; simple visibility timeout / lease model	More control: Peek-Lock (with explicit complete), Receive & Delete, lock renewal, dead-lettering
Ordering / grouping	Best-effort FIFO, not strict	Stronger ordering via sessions (group messages by session ID)
Transactions / batch operations	Not supported (each message independent)	Supported (atomic send / receive / settle)

Aspect	Azure Queue Storage	Azure Service Bus Queues
Duplicate detection / dead-letter	No built-in duplicate detection; you must build poison-queue logic via DequeueCount	Duplicate detection + built-in dead-letter queue support
Scheduled / delayed delivery	No built-in support	Yes, you can schedule messages to appear in future time
Latency / performance	Lower overhead; typical ~10 ms latency	Slightly higher due to richer semantics (e.g. ~20–25 ms)
Security / authentication	Uses storage keys, SAS (delegated access)	Supports SAS, also Azure AD / RBAC at finer granularity
Number of queues / entities	Unlimited (no rigid quota)	Limited (e.g. ~10,000 queues per namespace in Standard)

When To Use Each?

Use Azure Queue Storage when:

- Your needs are simple: you just need to buffer work (decouple producer and consumer), smooth spikes, maintain a backlog. You don’t require advanced features like transactions, ordering, or deduplication.

- The message volumes are large and the messages are small, and cost is important. Storage queues are cheaper (less overhead) for simple workloads.
- You don't need strict ordering guarantees. If your system can tolerate some reordering in rare failure/retry scenarios, queue storage is acceptable.
- Duplicates are okay (or your logic can handle duplicates). Since queue storage doesn't provide duplicate detection, your consumer logic should be idempotent or able to detect duplicates.
- The message payload size is within its limits (64 KB). If you have bigger payloads, you might store the payload in blob or another store, and just enqueue a pointer.
- You don't need built-in dead-lettering or retry policies at the queue layer. You can implement your own logic (e.g. if a message is dequeued too many times, move it to a "poison-queue") using the DequeueCount metadata.
- You prefer simplicity and minimal operational overhead. There's less to configure, fewer failure modes to worry about.
- You want as much scale as possible in terms of storing large quantities of messages across many queues without worrying about per-queue quotas.

Example: A web app accepting user tasks (e.g. image processing jobs): you enqueue a "process this image" message; a background worker dequeues and does the processing. The order doesn't matter strongly, duplicates are tolerable (or prevented at the consumer), and you want a low-cost, scalable solution.

Use Azure Service Bus (Queues) when:

- You need stronger delivery semantics and more control: e.g. you want "exactly once" *effectively* (or robust handling of duplicates), dead-lettering, retries, failure isolation.

- Ordering matters. You have workflows where messages must be processed in the same sequence they were produced (or certain subsets). Use sessions to bind order within a session.
- You need transactional operations: group sends/receives and other actions together so they either commit or rollback as a single unit.
- You want duplicate detection built in, so you don't have to write custom logic to filter duplicate messages.
- You want to schedule messages to appear later (delayed delivery) or perform message deferral.
- You have multiple subscribers or need publish/subscribe patterns — though that goes beyond plain queues (you'd use Topics/Subscriptions in Service Bus), but choosing Service Bus gives you that path forward.
- You want fine-grained permissions and security (sender, receiver roles, Azure AD integration, RBAC) rather than only storage-level control.
- Your message sizes fit within what Service Bus allows, and your queue's storage quota can accommodate your peak backlog.
- You want AMQP support, lower-overhead connections, better performance under high load with more advanced messaging behaviors.
- You want built-in facilities for dead-lettering, poison message handling, and built-in routing/forwarding of messages.

Example: A financial transaction processing pipeline: step A must complete before B, messages must be handled in order, failures must be isolated and retried, duplicates must be suppressed, and you may need to schedule certain operations or send messages to multiple downstream services.