### **1. Why is storing cleaned data in Azure Blob Storage important for real-time pipelines?**

* **Centralized Storage:** Blob Storage acts as a scalable data lake, allowing all downstream systems (analytics, ML models, dashboards) to consume the same cleaned data.
* **Reliability:** It ensures that real-time cleaned data is durably stored and available for recovery in case of failures.
* **Scalability:** Blob Storage can handle large volumes of streaming data without performance issues.
* **Decoupling:** Storing cleaned data separates the data processing stage from the consumption stage. This allows multiple consumers (BI tools, Spark jobs, APIs) to access the same dataset independently.
* **Audit & Replay:** Cleaned data stored in Blob can be replayed or reprocessed for debugging, auditing, or compliance.

In short, it provides a **reliable, scalable, and reusable layer** for real-time data pipelines.

### **2. What’s the difference between pipeline artifacts and Blob Storage uploads?**

* **Pipeline Artifacts (Azure DevOps):**
  + Temporary storage tied to the build/release pipeline.
  + Used to pass files (binaries, logs, scripts, test results) between pipeline stages.
  + Lifecycle is short; mainly exists until the pipeline completes or retention policies expire.
  + Intended for DevOps workflows, **not large-scale or long-term data storage**.
* **Blob Storage Uploads:**
  + Permanent, scalable cloud storage.
  + Used to store structured or unstructured data (CSV, JSON, images, logs, cleaned data, etc.) for long-term access.
  + Supports analytics, data pipelines, and real-time streaming.
  + Independent of DevOps pipelines; data persists until explicitly deleted.

**Artifacts = pipeline-centric & short-lived** **Blob Storage = data-centric & long-term**

### **3. How would you handle failures in file uploads in a production setup?**

* **Retry Logic:** Implement automatic retries with exponential backoff for transient errors (network issues, throttling).
* **Error Logging & Alerts:** Log failures to monitoring tools (Azure Monitor, Log Analytics, App Insights) and trigger alerts for immediate investigation.
* **Dead Letter Storage (Fallback):** Redirect failed uploads to a separate container (e.g., failed-uploads/) for later inspection and reprocessing.
* **Transactional/Atomic Uploads:** Use staging and commit pattern (upload to a temporary location, then rename/move once complete) to avoid partial files.
* **Idempotent Uploads:** Ensure re-uploads don’t create duplicates by using unique file naming (GUIDs, timestamps, hash-based).
* **Monitoring & Dashboard:** Set up dashboards to monitor success/failure rates of uploads in real time.

Together, these strategies make the pipeline **resilient, fault-tolerant, and production-ready**.