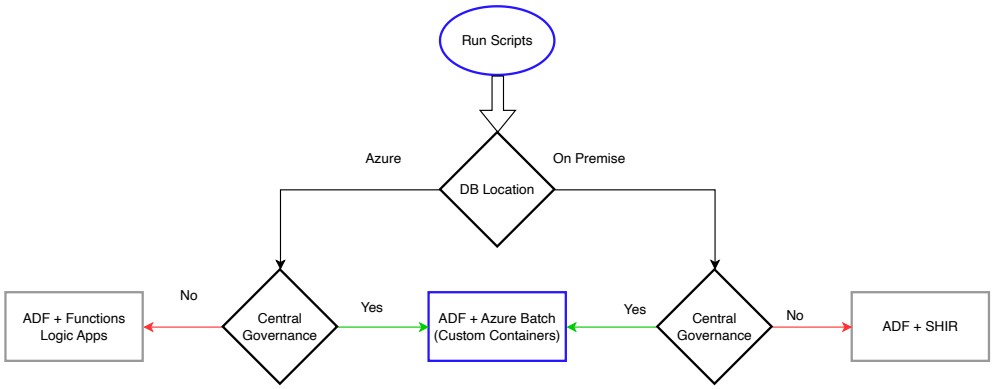
❌ 1. Azure Data Factory does **not natively support executing arbitrary scripts** — including SQL, Bash, Python, or PowerShell — on PostgreSQL targets, unlike its native support for Azure SQL via stored procedure and script activities.
⚠️ 2. While the PostgreSQL Connector v2.0 supports authentication via **Service Principal**, its functionality is **limited to data movement operations** such as **CopyActivity** and **metadata retrieval** (LookupActivity);
it does not support the execution of custom DDL/DML SQL scripts.
✅ 3. Future versions of the PostgreSQL connector (e.g., v3.0 and beyond) **may extend functionality** to support custom script execution and additional activities beyond data movement;
however, as of now, such capabilities are **not available**.



⚠️ **1. Limited Script Execution Scope and Flexibility**

• Azure Functions and Logic Apps are well-suited for **lightweight, event-driven tasks** but are **not ideal for full SQL script execution**, especially DDL-heavy workloads.
• You'll need to manually handle psql or custom connectors — they **don't natively support arbitrary script execution** against PostgreSQL like Batch can.

⚠️ **2. Complex Authentication and Execution Context**

• Managed Identity support exists but **requires explicit binding per app/function**, with more complexity in **multi-tenant or hybrid setups**.
• **Execution context is stateless and time-bound**, which can be problematic for long-running or dependent script chains.

⚠️ **3. Limited Control, Observability, and Reusability**

• Functions and Logic Apps offer **limited control over runtime environments** — you can't containerize or bundle custom tooling.
• Debugging, logging, and retry handling are **less granular** than Azure Batch, and scaling beyond basic scenarios often introduces more orchestration overhead.

✅ **1. Centralized, Scalable, and Consistent Execution**

• Enables **centralized job execution** across all regions, tenants, and network zones using reusable Batch pools.
• Custom containers provide a **consistent runtime** (e.g., with psql, Python, tools pre-installed), eliminating environment drift.

✅ **2. Strong Governance and Secure Identity Management**

• Supports **Service Principal or Managed Identity** authentication at the pool level — easy to audit, rotate, and control via Azure RBAC.
• Batch jobs are executed in **isolated, policy-governed containers**, enabling centralized control and logging.

✅ **3. Flexible Networking and Tooling**

• Easily integrates with **VNETs and private endpoints**, suitable for both cloud and hybrid (on-prem) PostgreSQL targets.
• Custom containers allow you to **bundle all required scripts, binaries, and tools**, reducing operational dependencies and setup time.

❌ **1. Decentralized Execution and Governance**

• SHIR requires per-environment installation, making it difficult to enforce **centralized control**, policy management, or consistent access via service principals.
• No centralized execution pool or unified authentication strategy across tenants, regions, or on-prem environments.

❌ **2. Operational Complexity and Poor Scalability**

• High setup and maintenance overhead: each SHIR needs **manual installation**, patching, and capacity planning.
• **Does not scale well** for large estates — every new DB or network zone may require a separate SHIR or config.

❌ **3. Rigid Networking and No Runtime Flexibility**

• Tightly coupled to the network layer — **must reside within the same VNET or firewall scope** as the target DB.
• Cannot run custom containers or pre-built execution environments like Azure Batch, making **tooling and dependency management** inflexible and error-prone.