



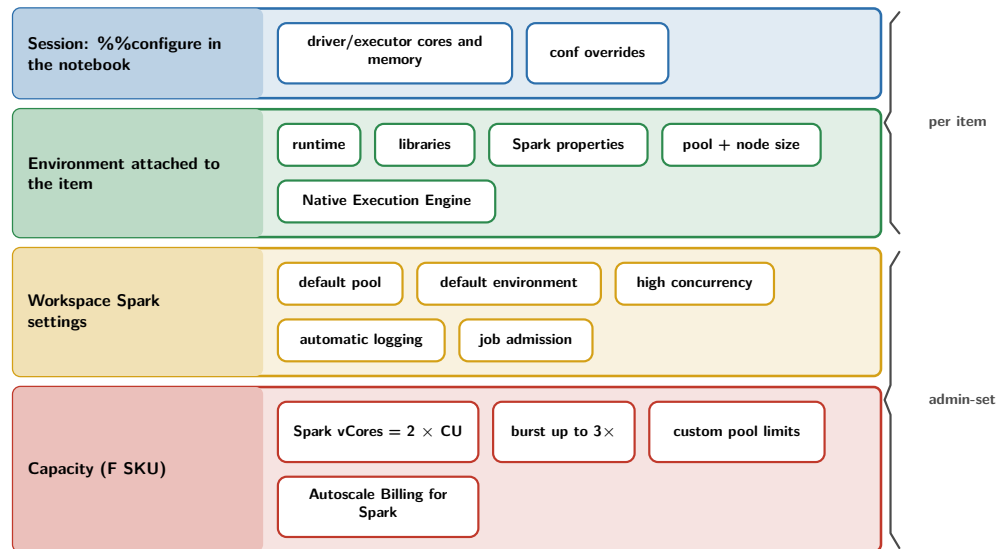
DP-700 Domain 1: Implement and Manage the Solution

Microsoft Learn DP-700 skills outline and Microsoft Fabric documentation, as of 2026 | Workspace and Spark settings, pools and environments, domains, OneLake settings, security and governance, network protection, lifecycle (Git, deployment pipelines, database projects, variable libraries), orchestration with pipelines, notebooks and dataflows, parameters, schedules and event triggers

DP-700 Domain 1 Spark settings Orchestration Security

⚙️ Configure the workspace

Where Spark settings come from (top wins)



Setting	Where	Exam point
Starter pool vs custom pool	Workspace settings → Data Engineering/Science	starter = seconds to start, medium nodes; custom = specific node size or library-heavy jobs, minutes to start
Environment	item + publish	install libraries once (PyPI, conda, .whl, .jar); attach to notebooks and Spark job definitions; workspace default environment
High concurrency	Spark settings	share sessions between notebooks and pipeline runs, tag sessions with a session tag
Runtime version	environment / workspace	Fabric Runtime 1.3 = Spark 3.5, Delta 3.2 (newer runtimes appear: check)
Domains	Admin portal → Domains	group workspaces by business area; domain admins; default sensitivity label

OneLake settings

workspace and tenant

external app access to OneLake, file explorer, short-cut caching

Network

tenant and workspace

private links (tenant or workspace), managed private endpoints for Spark, trusted workspace access, outbound access protection

Worked example: how many Spark nodes fit on an F64?

PROBLEM

A workspace on F64 uses a custom pool of medium nodes (8 vCores each). How many Spark vCores and nodes can run at once, with burst?

GIVEN

- $C = 64$ capacity units
- $1 \text{ CU} = 2 \text{ Spark vCores}$
- burst factor $b = 3$ (Spark only)
- medium node = 8 vCores

1 Base Spark vCores

$$64 \times 2 = 128 \text{ vCores}$$

Why: the capacity's steady Spark allowance

2 With burst

$$128 \times 3 = 384 \text{ vCores}$$

Why: short bursts are allowed and smoothed later

3 Medium nodes at burst

$$384 / 8 = 48 \text{ nodes}$$

Why: shared by all concurrent Spark sessions on the capacity

4 Queueing

jobs beyond the limit are queued (batch) or rejected (interactive)

Why: job admission control

ANSWER

128 Spark vCores steady, up to 384 with burst: at most 48 medium nodes at once across the capacity.

CHECK

- ✓ Background Spark usage still counts against CU smoothing; Autoscale Billing for Spark moves Spark to pay-as-you-go instead.

🔒 Security and governance

- **Workspace roles** Admin, Member, Contributor, Viewer; engineers need Contributor to create items and run notebooks; Viewer can query SQL endpoints but cannot read OneLake files with Spark

- **Item permissions** Read, ReadData (SQL), ReadAll (OneLake/Spark), Write, Execute, Reshare; share an item instead of granting a workspace role
- **Warehouse and SQL endpoint** GRANT/DENY on objects and columns, row-level security policies, dynamic data masking with UNMASK
- **OneLake security** data access roles on lakehouse tables and folders (with row/column rules as they become GA), enforced across engines; check current status
- **Sensitivity labels and endorsement** Purview labels inherited downstream; Promoted, Certified, Master data; workspace monitoring and Purview audit for activity
- **Network isolation** managed private endpoints from Spark to Azure SQL / storage; trusted workspace access to firewalled ADLS via workspace identity; tenant or workspace private link

MYTH

❌ A workspace Viewer can read lakehouse files from a notebook

❌ Libraries must be pip-installed at the top of every notebook

❌ Custom pools start as fast as the starter pool

FACT

✅ **Viewers have no ReadAll on OneLake**
Grant ReadAll on the item or an OneLake security role.

✅ **Publish them once in an environment**
Inline %pip is for experiments; environments make sessions reproducible and faster.

✅ **Starter pools are pre-warmed; custom pools take minutes**
Use custom pools only when node size or libraries require it.

Lifecycle

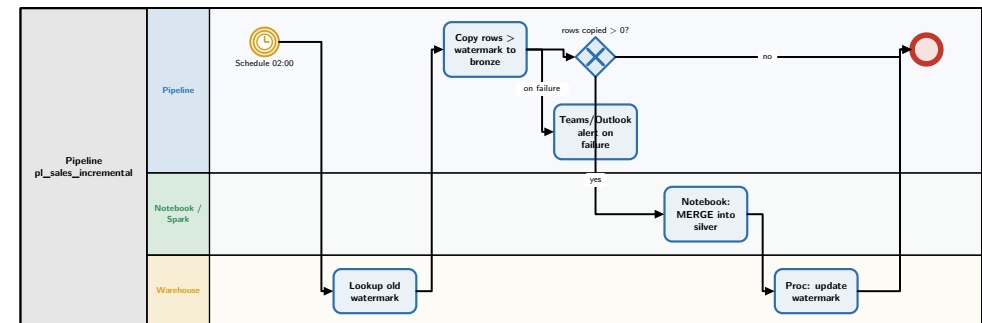
Tool	Use it for	Remember
Git integration	version item definitions (notebooks, pipelines, lakehouse meta-data, warehouse via database project)	Azure DevOps or GitHub; data never in Git
Deployment pipelines	promote Dev → Test → Prod (2-10 stages)	deployment rules, selective deploy, compare
Variable libraries	per-stage values read by pipelines and other items	replaces hard-coded IDs and connections
SQL database projects	warehouse schema as code (.sql-proj → DACPAC)	deploy with SqlPackage or pipelines
fabric-cicd / Fabric REST APIs / Fabric CLI	code-first deployment from a repository	Python library, fab CLI

→ Orchestrate

Pipeline, notebook or dataflow?

Conditions	Analyst	One job	Notebook DAG	End to end
Low-code transform by analysts	Y	N	N	N
Heavy or custom code (PySpark)	N	Y	Y	N
Several notebooks with dependencies only	N	N	Y	N
Mix of copy, notebooks, procedures, alerts	--	--	N	Y
Actions				
Dataflow Gen2	X			
Notebook (scheduled)		X		
notebookutils.notebook.runMultiple (DAG)			X	
Data pipeline orchestrating activities				X

Nightly incremental load pipeline



Parameters and dynamic content

```

# notebook: cell toggled as "parameters"
load_date = "2026-01-01"
source_table = "orders"
# pipeline Notebook activity -> Base parameters:
# load_date = @formatDateTime(utcNow(), 'yyyy-MM-dd')
# source_table = @pipeline().parameters.table
# read a Lookup result in the next activity:
# @activity('Lookup old watermark').output.firstRow.wm
# run notebooks as a DAG from a notebook
dag = {"activities": [
  {"name": "bronze", "path": "nb_bronze"},
  {"name": "silver", "path": "nb_silver",
    "dependencies": ["bronze"]}]}
notebookutils.notebook.runMultiple(dag)
notebookutils.notebook.exit("rows=1250") # value back to caller
  
```

EXAM TIP

Triggers: **schedule** for fixed times (several schedules per item possible); **event-based trigger** (Real-Time hub: OneLake or Azure Blob Storage file events, job events) when 'start as soon as the file lands'. Use **ForEach** (sequential or batch count up to 50 in parallel) to loop tables from a Lookup; **Until** for polling; **If Condition** and **Switch** for branching; activity dependencies: on success, on failure, on completion, on skip.

Recap and self-test

Key takeaways

01

Settings cascade: session > environment > workspace > capacity

1 CU = 2 Spark vCores, burst 3×

02

Starter pool for speed, custom pool for specific sizes

environments for libraries

03

Least privilege on items

ReadAll for Spark, ReadData for SQL

04

Git + deployment pipelines + variable libraries

promote without hard-coding

05

Pick the orchestrator by work type

dataflow, notebook, runMultiple, pipeline

Q1 Which Spark pool starts sessions in seconds?

A The starter pool (pre-warmed medium nodes).

Q2 Where do you install a custom .whl for all notebooks of a project?

A In an environment (custom libraries), published and attached or set as workspace default.

Q3 Setting that lets several notebooks share one Spark session?

A High concurrency mode.

Q4 How many Spark vCores does one capacity unit give?

A Two.

Q5 How does a pipeline pass a value into a notebook?

A Base parameters on the Notebook activity override the notebook's parameter cell variables.

Q6 Function to run notebooks as a dependency graph?

A `notebookutils.notebook.runMultiple(dag)`.

Q7 Feature to reach a firewall-protected ADLS Gen2 account from Fabric?

A Trusted workspace access with a workspace identity (or a managed private endpoint for Spark).

Q8 Item holding per-stage values like connection IDs?

A Variable library.

Q9 Trigger a pipeline when a file lands in OneLake?

A Event-based trigger (OneLake events through the Real-Time hub).