Best Practices for Building Pipelines and Notebooks for CI/CD in Microsoft Fabric

# 1. General Principles (Applicable to Both Pipelines and Notebooks)

- Avoid hardcoded identifiers: Do not hardcode workspace IDs, Lakehouse names, Warehouse IDs, paths, or connection strings. Use parameterized inputs or reference centralized environment configuration (like parameters.yml).  
- Design for environment-independence: Build once, deploy anywhere. Base logic on abstract identifiers that can be mapped at deploy-time.  
- Follow modular design: Break large logic blocks into smaller reusable components. Separate data ingestion, transformation, and output logic.  
- Enable reusability and versioning: Use Git integration effectively to maintain version control. Prefer notebooks/pipelines that can be re-used across data domains.

# 2. Notebooks – Best Practices

## A. Use Parameters at Notebook Level

- Declare notebook parameters at the beginning using dbutils.widgets.get().  
- Use these parameters to control I/O, storage zones, and destinations dynamically.

## B. Avoid Hardcoded Paths

- Construct paths dynamically using parameter values.

## C. Output to Logical Zones (Bronze/Silver/Gold)

- Maintain staging and clean zones for better data flow control.  
- Parameterize zone names (e.g., raw, curated).

## D. Include Validation and Logging

- Add basic checks to verify input data exists and output is written successfully.  
- Log notebook execution steps to a central store or table.

## E. Design for Reusability

- Avoid embedding too much logic in one notebook.  
- Use modular helpers and the %run command when applicable.

# 3. Pipelines – Best Practices

## A. Parameterize All Input Values

- Set relevant pipeline inputs (Lakehouse ID, file path, etc.) as parameters.  
- Use parameters.yml to map correct values during deployment.

## B. Use System Variables and Expressions

- Leverage expressions and system variables for dynamic behavior.

## C. Activity Modularization

- Group related activities into containers or sub-pipelines for reuse.

## D. Add Error Handling and Retry Logic

- Implement onFailure, onCompletion, and retry options to handle failures.

## E. Implement Metadata-driven Design (Optional)

- Reference metadata tables or config files instead of hardcoded logic.

# 4. Dynamic Deployment Considerations

- Build assets once using placeholders or dynamic parameters.  
- Drive environment-specific behavior via parameters.yml during CI/CD deployment.

# 5. Summary of Do's and Don'ts

|  |  |
| --- | --- |
| Do | Don't |
| Use pipeline and notebook parameters | Hardcode any IDs or paths |
| Reference config via parameters.yml | Use static lakehouse/table names |
| Add validations and retry policies | Assume success of external calls |
| Modularize logic | Put all logic in one monolithic pipeline |
| Write reusable components | Mix dev/test/prod logic inside a single script |