**Databricks Deployment Workflow - Study Guide**

**1. bundle.yml env Blocks**

**Purpose**: Centralize environment-specific configurations.

**Key Concepts**:

* One bundle.yml file can define multiple environments (dev, test, prod).
* Environment blocks include workspace details and variable definitions.
* Secrets should **never** be stored directly in bundle.yml; use environment variables or secret scopes.

**Example**:

environments:

dev:

workspace:

host: https://dev.databricks.com

variables:

cluster\_id: abc123

prod:

workspace:

host: https://prod.databricks.com

variables:

cluster\_id: xyz456

**Best Practices**:

* Keep environment variables consistent across environments.
* Use version control to track changes.

**2. Deploy vs Sync**

**Deploy**:

* Used in CI/CD to push code, configurations, and jobs to Databricks.
* Ensures that the environment matches the committed state.

**Sync**:

* Used for local development.
* Automatically syncs changes between local files and Databricks workspace.

**Tip**: Use Deploy for production changes, Sync for dev/test iteration.

**3. Secret Scopes**

**Purpose**: Secure credentials and sensitive values.

**Types**:

* **Databricks-backed**: Stored in Databricks.
* **Azure Key Vault-backed**: Delegates storage to Azure Key Vault.

**Example**:

my\_secret = dbutils.secrets.get(scope="my-scope", key="my-key")

**Best Practices**:

* Use role-based access control (RBAC).
* Rotate secrets regularly.

**4. Validate & Tag**

**Validation**:

* Includes syntax checks, linting, and test runs.
* Prevents deployment of faulty code.

**Tagging**:

* Git tags mark release points.

git tag -a v1.0.0 -m "Release v1.0.0"

git push origin v1.0.0

**Best Practices**:

* Only tag after passing all checks.
* Use semantic versioning.

**5. Rollback**

**Purpose**: Restore a stable version after a failed release.

**Methods**:

* Redeploy a previous Git tag.
* Use Databricks job run history.

**Best Practices**:

* Keep artefacts from previous releases.
* Document rollback steps.

**6. Promote to Test**

**Process**:

1. Deploy validated code from dev to test.
2. Run integration tests.
3. Validate data quality and job performance.

**Best Practices**:

* Use isolated clusters.
* Use different storage paths per environment.

**7. Auto Release Notes**

**Purpose**: Automate changelog generation.

**Implementation**:

* Use tools like Release Drafter in GitHub Actions.

- name: Generate Release Notes

uses: release-drafter/release-drafter@v5

**Best Practices**:

* Use conventional commit messages.
* Store release notes in repo for traceability.

**Quick Reference Table**

| **Topic** | **Purpose** | **Best Practice Highlight** |
| --- | --- | --- |
| bundle.yml env blocks | Define env-specific configs | Keep secrets out of config |
| Deploy vs Sync | CI/CD vs local development | Deploy for prod, sync for dev/test |
| Secret Scopes | Secure sensitive values | Use RBAC and rotate secrets |
| Validate & Tag | Ensure quality & version control | Tag only after all checks pass |
| Rollback | Restore stable state | Keep previous release artefacts |
| Promote to Test | Controlled environment upgrade | Use isolated clusters & storage |
| Auto Release Notes | Automate changelog creation | Follow commit message conventions |