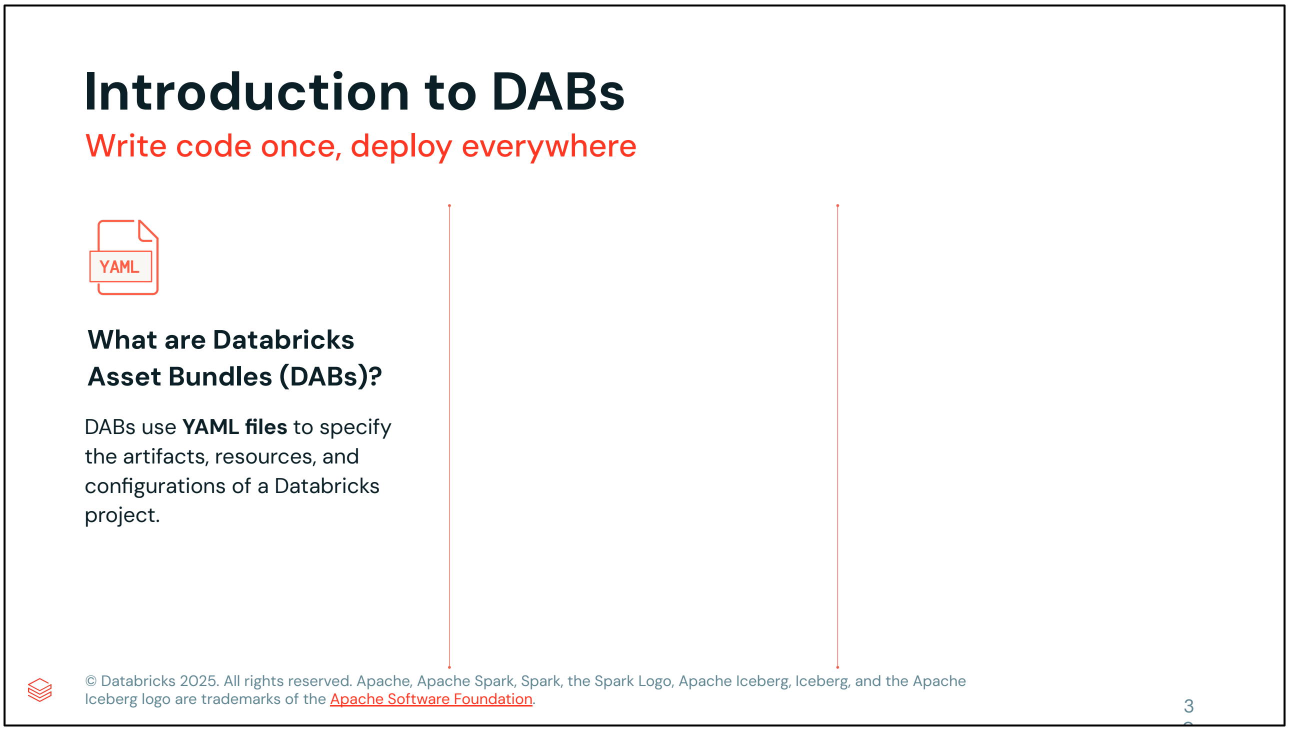
databricks

Deployment with Databricks Asset Bundles (DABs)

LECTURE

Introduction to Databricks Asset Bundles (DABs)

In this lecture, we introduce Databricks Asset Bundles (DABs), covering their benefits, CI/CD integration, project structure, and deployment validation.



Introduction to DABs

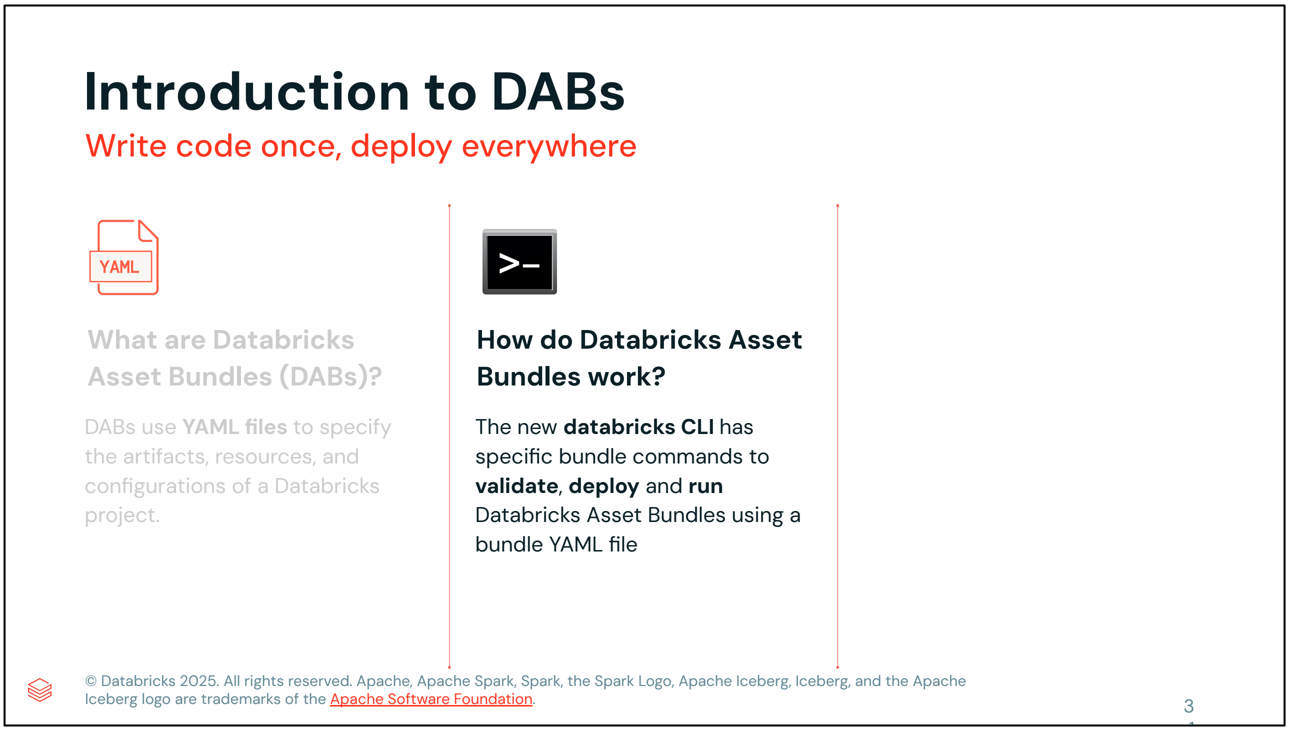
Write code once, deploy everywhere

What are Databricks Asset Bundles (DABs)?

DABs use **YAML files** to specify the artifacts, resources, and configurations of a Databricks project.

DABs enable us to write code once and deploy everywhere. But what exactly are they?

DABs (Databricks Asset Bundles) use YAML files to specify the artifacts, resources, and configurations of a Databricks project. They allow us to manage and deploy all the necessary components of a Databricks project in a consistent and efficient way to multiple environments.



Introduction to DABs

Write code once, deploy everywhere

What are Databricks Asset Bundles (DABs)?

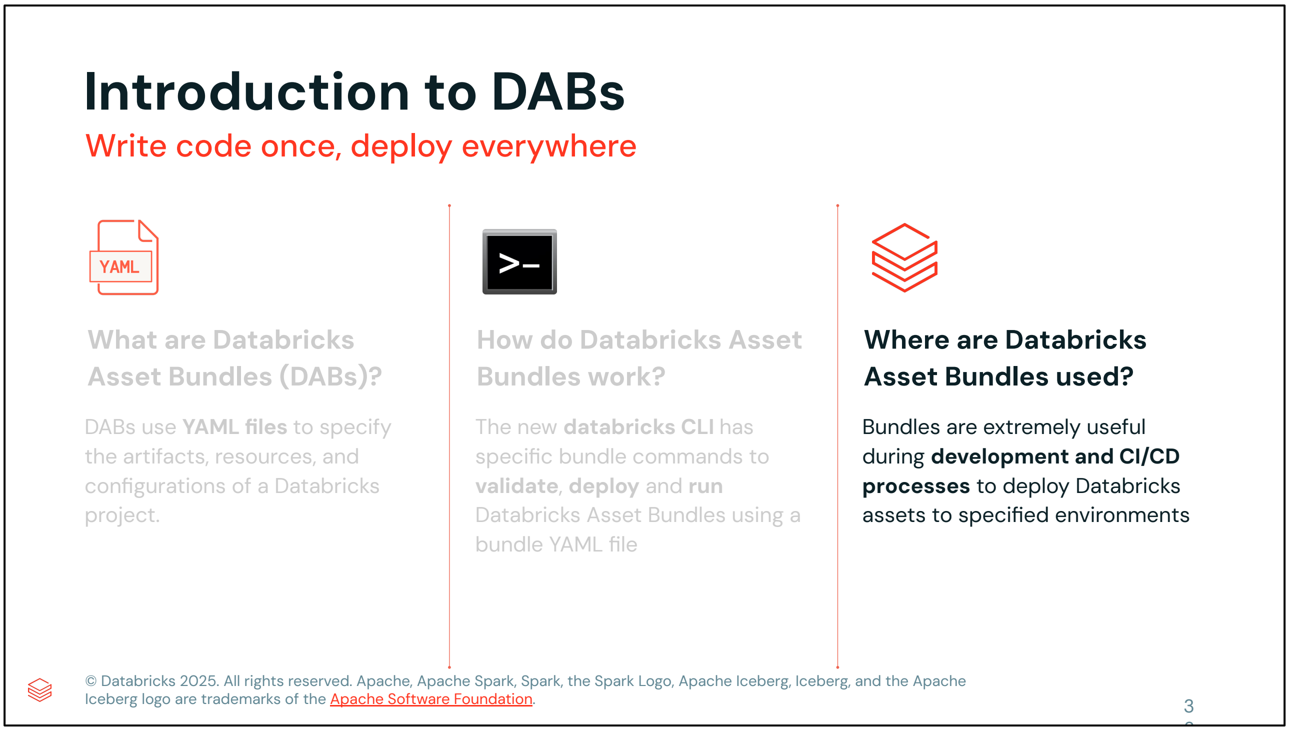
DABs use **YAML files** to specify the artifacts, resources, and configurations of a Databricks project.

How do Databricks Asset Bundles work?

The new **databricks CLI** has specific bundle commands to **validate**, **deploy** and **run** Databricks Asset Bundles using a bundle YAML file

How exactly do DABs work?

The new Databricks CLI has specific bundle commands to validate, deploy, and run Databricks Asset Bundles using a YAML file. We will learn how to do this throughout the course.



Introduction to DABs

Write code once, deploy everywhere

What are Databricks Asset Bundles (DABs)?

DABs use **YAML files** to specify the artifacts, resources, and configurations of a Databricks project.

How do Databricks Asset Bundles work?

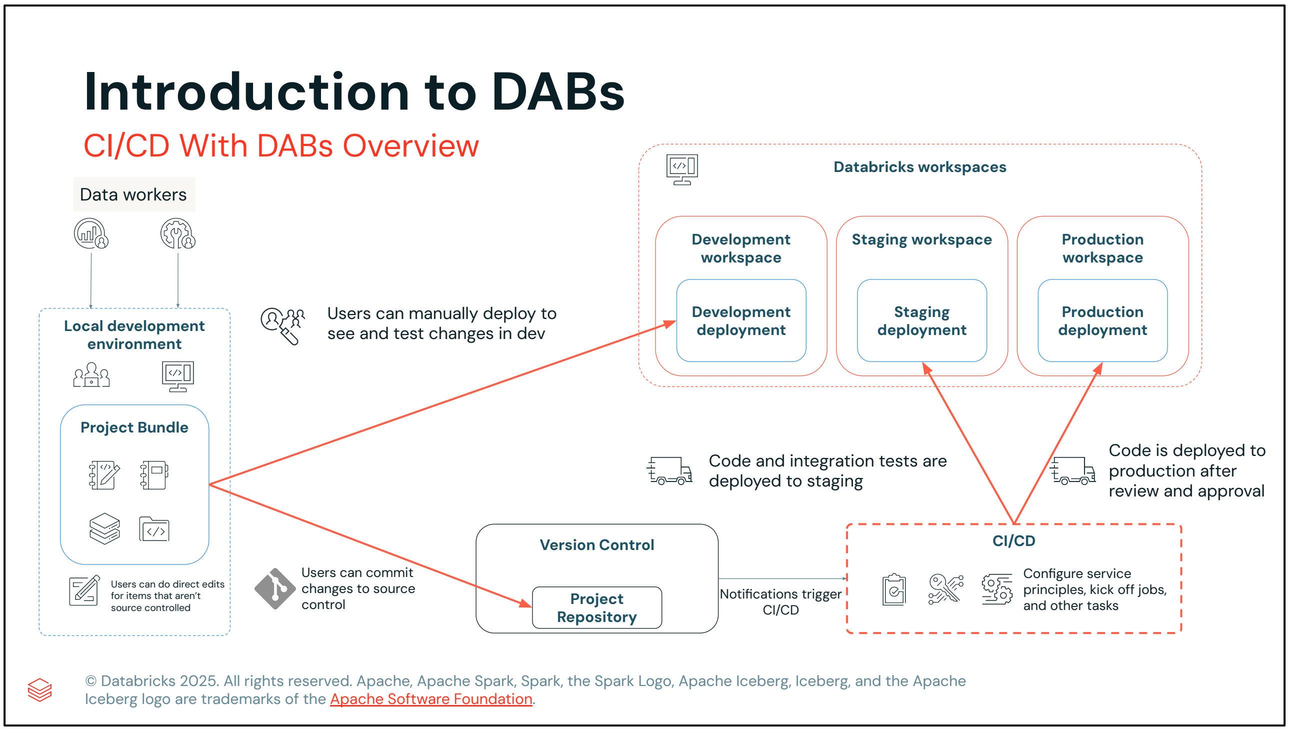
The new **databricks CLI** has specific bundle commands to **validate**, **deploy** and **run** Databricks Asset Bundles using a bundle YAML file

Where are Databricks Asset Bundles used?

Bundles are extremely useful during **development and CI/CD processes** to deploy Databricks assets to specified environments

So, where are Databricks Asset Bundles used?

DABs are extremely useful during development and CI/CD processes for deploying Databricks assets to target environments, while modifying specific configurations for each environment. We will see how to do this throughout the course.



Introduction to DABs

CI/CD With DABs Overview

Users can manually deploy to see and test changes in dev

Users can do direct edits for items that aren't source controlled

Users can commit changes to source control

Code and integration tests are deployed to staging

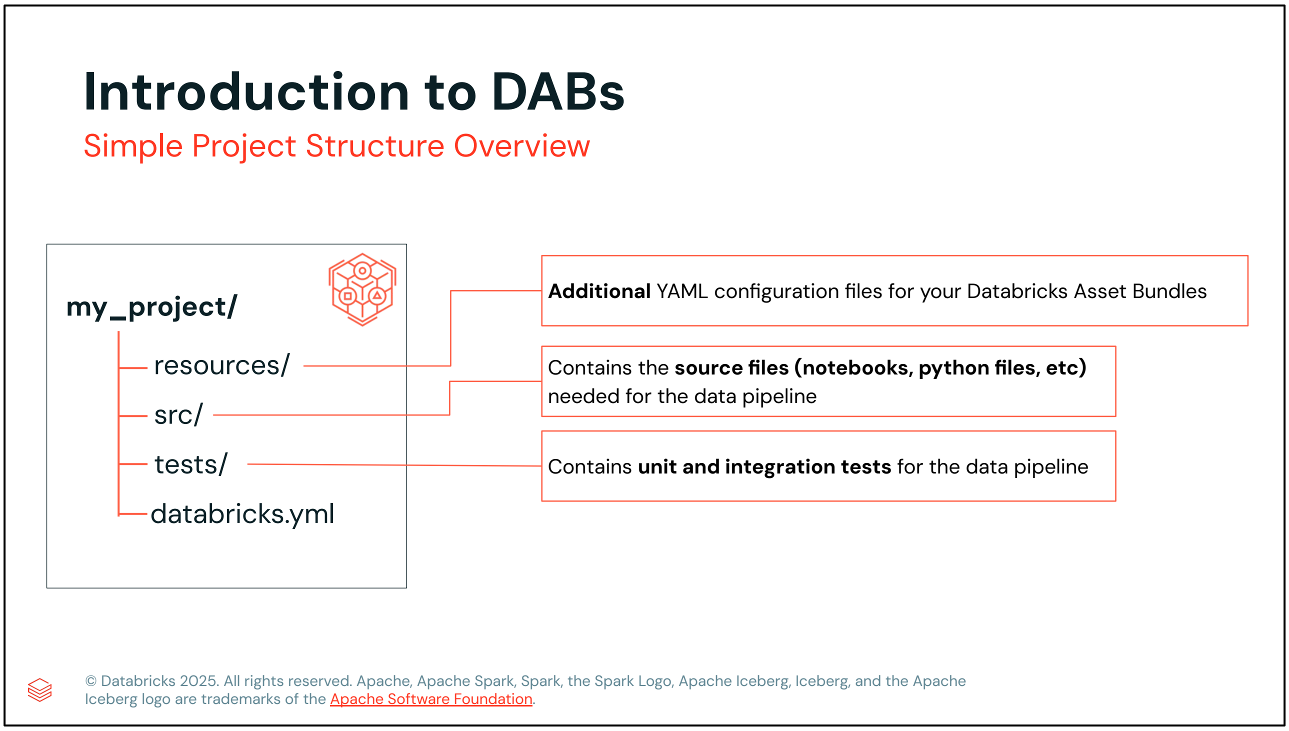
Code is deployed to production after review and approval

Configure service principles, kick off jobs, and other tasks

Let's take a look at the following diagram, which provides a high-level view of a development and CI/CD pipeline with bundles.

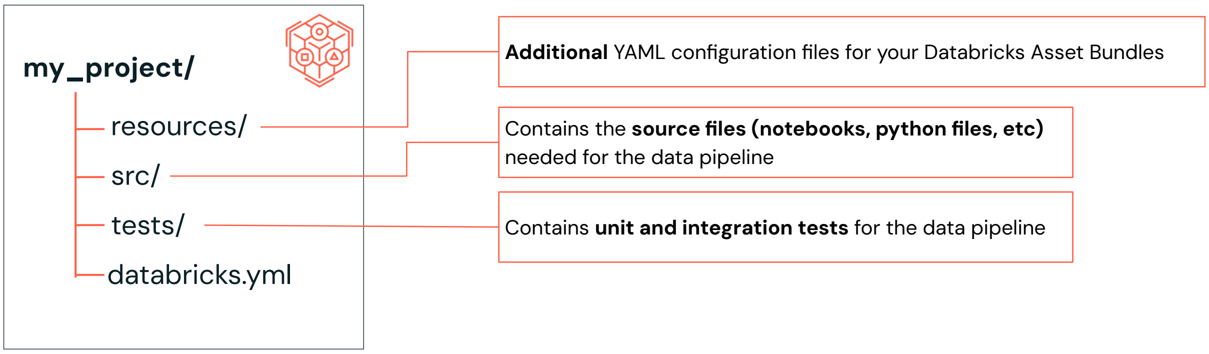
If you're working locally, you'll build the project bundle with your team using a local environment setup (or you can do this within your Databricks Workspace).

- From here, you can either use bundles or manually deploy to your development environment, which can include a development workspace or catalogs. This is where you can test your changes away from production.
- After development, users can then commit changes to version control within a project repository and push your development changes.
- After the changes are committed, you can set a notification to trigger to initiate the CI/CD pipeline. In a typical pipeline, the tests are first deployed to a staging environment.
- Once all tests pass in staging, the code can then be deployed to the production environment after following your organization's processes, such as code review and approval.



Introduction to DABs

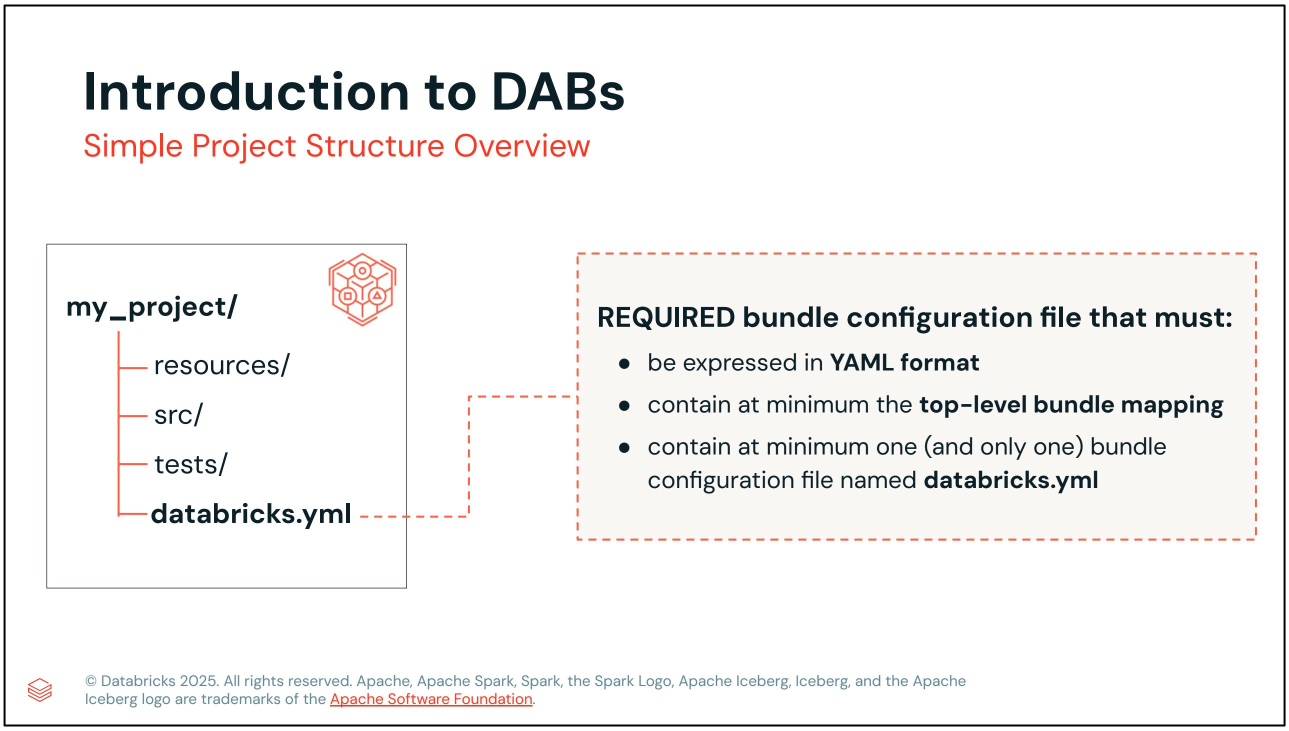
Simple Project Structure Overview


my_project/
resources/
src/
tests/
databricks.yml
Additional YAML configuration files for your Databricks Asset Bundles
Contains the **source files (notebooks, python files, etc)** needed for the data pipeline
Contains **unit and integration tests** for the data pipeline

Now that we have a good overview of the CI/CD process, let's dive into a typical, simple project structure for your bundle.
In your project folder, you'll want to create multiple subfolders to organize your assets. A simple project structure should start with your project folder, followed by various folders and files, such as resources, src, tests, and a main databricks.yaml file. These folders may contain additional subfolders to store specific files.

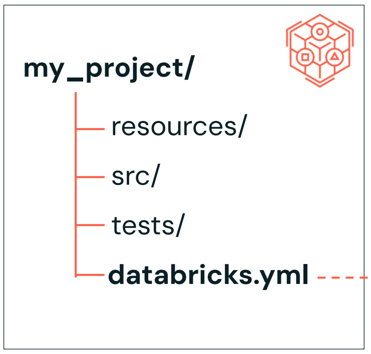
- The resources folder should contain additional YAML configuration files for your Databricks Asset Bundles, if needed.
- The src (or source) folder contains the source files needed for your data pipeline, such as notebooks and Python files.
- The tests folder contains your unit and integration tests for the pipeline.

Organizing and modularizing your folders and files will help you during development and maintenance as your project grows.
This is a simple example, and your project may have additional files and folders, or a different organizational structure.



Introduction to DABs

Simple Project Structure Overview

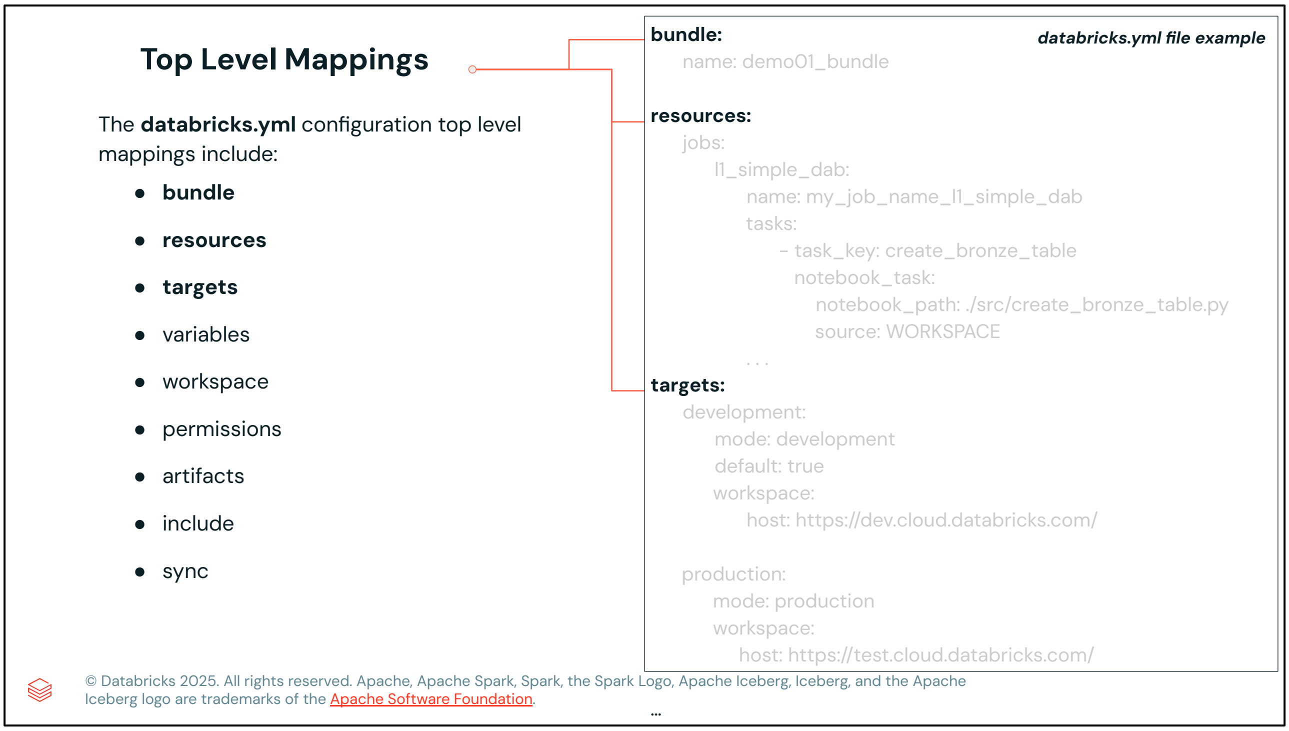


REQUIRED bundle configuration file that must:

- be expressed in **YAML format**
- contain at minimum the **top-level bundle mapping**
- contain at minimum one (and only one) bundle configuration file named **databricks.yml**

The databricks.yml file is a required bundle configuration file used to deploy your Databricks assets. This file must:

- Be expressed in YAML format.
- Contain at minimum the top-level bundle mapping.
- Contain at least one (and only one) bundle configuration file named databricks.yml.



Top Level Mappings

The **databricks.yml** configuration top level mappings include:

- **bundle**
- **resources**
- **targets**
- variables
- workspace
- permissions
- artifacts
- include
- sync

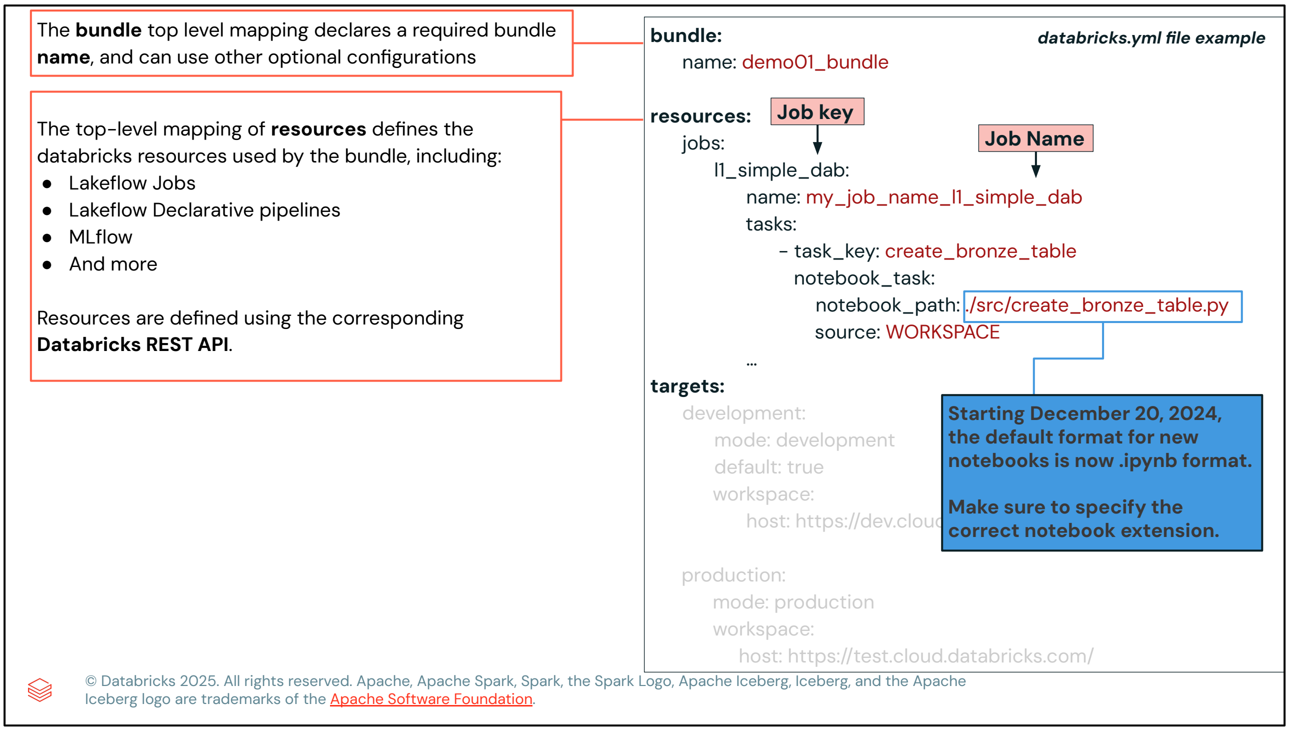
databricks.yml file example

```
bundle:
    name: demo01_bundle

resources:
    jobs:
        l1_simple_dab:
            name: my_job_name_l1_simple_dab
            tasks:
                - task_key: create_bronze_table
                  notebook_task:
                    notebook_path: ./src/create_bronze_table.py
                    source: WORKSPACE
            . . .
targets:
    development:
        mode: development
        default: true
        workspace:
            host: https://dev.cloud.databricks.com/

    production:
        mode: production
        workspace:
            host: https://test.cloud.databricks.com/
...
```

The databricks.yml file is key to your bundle. Let's take a look at a simple example.

First, the YAML file contains several top-level mapping keys that are left-aligned. In this example databricks.yml file, it includes the top-level mappings: bundle, resources, and targets.

Other top-level mappings include variables, workspace, permissions, artifacts, include, and sync. We will cover many of these top-level mappings in this course.



The **bundle** top level mapping declares a required bundle **name**, and can use other optional configurations

The top-level mapping of **resources** defines the databricks resources used by the bundle, including:

- Lakeflow Jobs
- Lakeflow Declarative pipelines
- MLflow
- And more

Resources are defined using the corresponding **Databricks REST API**.

databricks.yml file example

```
bundle:
    name: demo01_bundle

resources:
    jobs:
        l1_simple_dab:
            name: my_job_name_l1_simple_dab
            tasks:
                - task_key: create_bronze_table
                  notebook_task:
                     notebook_path: ./src/create_bronze_table.py
                     source: WORKSPACE
            ...
targets:
    development:
        mode: development
        default: true
        workspace:
            host: https://dev.cloud
    production:
        mode: production
        workspace:
            host: https://test.cloud.databricks.com/
```

Starting December 20, 2024, the default format for new notebooks is now .ipynb format.

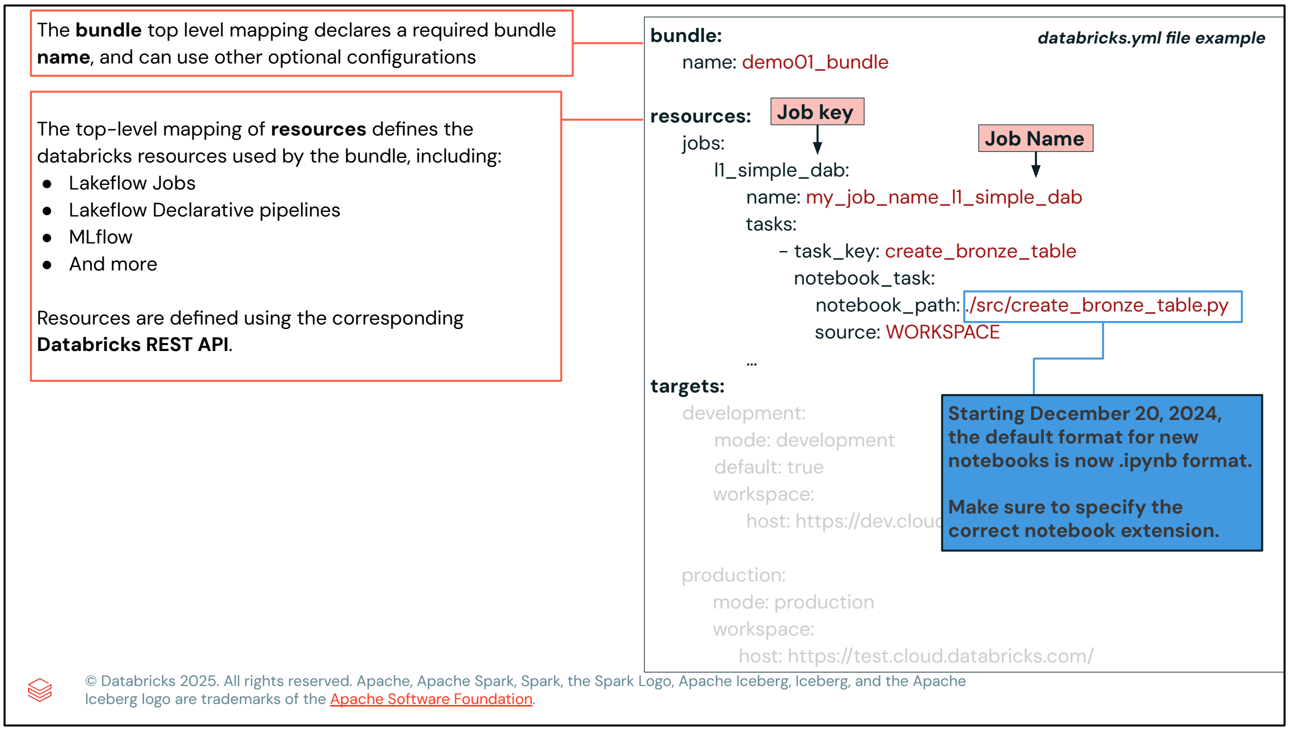
Make sure to specify the correct notebook extension.

A bundle configuration file must contain only one top-level **bundle** mapping that associates the entire bundle's contents with a name. Additionally, you can include other Databricks workspace settings, such as **cluster_id**, **compute_id**, **git**, and a few others, if needed. The additional mappings beneath the top-level mapping must be indented.

The following example declares the top-level mapping key **bundle**, with a **name** mapping that specifies the bundle name, demo01_bundle.

The **resources** mapping specifies information about the Databricks resources used by the bundle, such as:

Lakeflow Jobs
Lakeflow Declarative pipelines
MLflow
And more



The **bundle** top level mapping declares a required bundle **name**, and can use other optional configurations

The top-level mapping of **resources** defines the databricks resources used by the bundle, including:

- Lakeflow Jobs
- Lakeflow Declarative pipelines
- MLflow
- And more

Resources are defined using the corresponding **Databricks REST API**.

databricks.yml file example

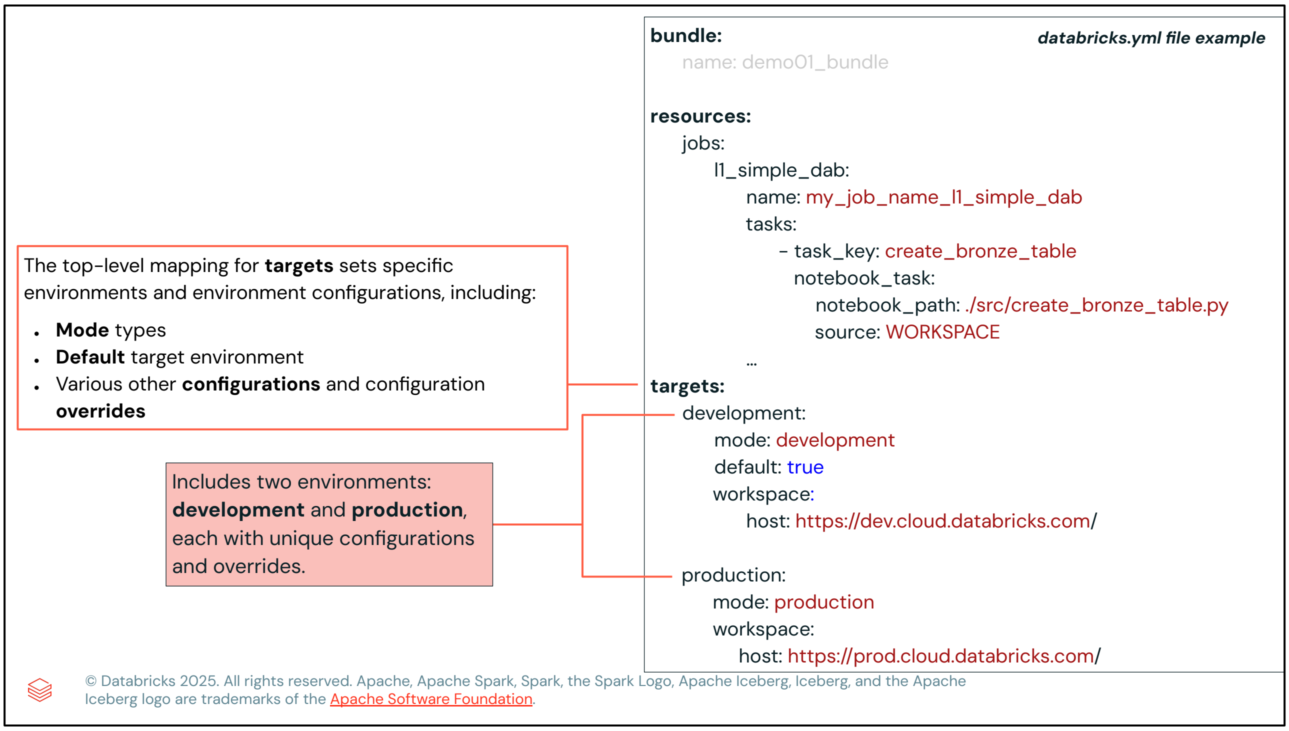
```
bundle:
  name: demo01_bundle

resources:
  jobs:
    l1_simple_dab:
      name: my_job_name_l1_simple_dab
      tasks:
        - task_key: create_bronze_table
          notebook_task:
            notebook_path: ./src/create_bronze_table.py
            source: WORKSPACE
      ...
targets:
  development:
    mode: development
    default: true
    workspace:
      host: https://dev.clou

  production:
    mode: production
    workspace:
      host: https://test.cloud.databricks.com/
```

Starting December 20, 2024, the default format for new notebooks is now .ipynb format.

Make sure to specify the correct notebook extension.

The resources are defined using the corresponding Databricks REST API parameters.

Now, each **resource** type mapping includes one or more individual resource declarations, each of which must have a unique name.
In this example, the job has the resource mapping **name** *l1_simple_dab*. This job creates a job named *my_job_name_l1_simple_dab* using the **name** mapping key, and this job contains one or more tasks specified using the corresponding REST API parameters.
Within the **tasks** mapping, the task is named *create_bronze_table* and uses the **notebook_path** mapping key to specify the notebook to use. It's important to use a relative path of the notebook within your project here, with the correct extension for the notebook.
Be aware that starting December 20, 2024, the default format for new notebooks will be the .ipynb format. If you do not supply the correct notebook extension, an error will be returned. In this example this notebook is using the traditional.py extension.



databricks.yml file example

```
bundle:
    name: demo01_bundle

resources:
    jobs:
        l1_simple_dab:
            name: my_job_name_l1_simple_dab
            tasks:
                - task_key: create_bronze_table
                  notebook_task:
                     notebook_path: ./src/create_bronze_table.py
                     source: WORKSPACE
            ...
targets:
    development:
        mode: development
        default: true
        workspace:
            host: https://dev.cloud.databricks.com/

    production:
        mode: production
        workspace:
           host: https://prod.cloud.databricks.com/
```

The top-level mapping for **targets** sets specific environments and environment configurations, including:

- **Mode** types
- **Default** target environment
- Various other **configurations** and configuration **overrides**

Includes two environments: **development** and **production**, each with unique configurations and overrides.

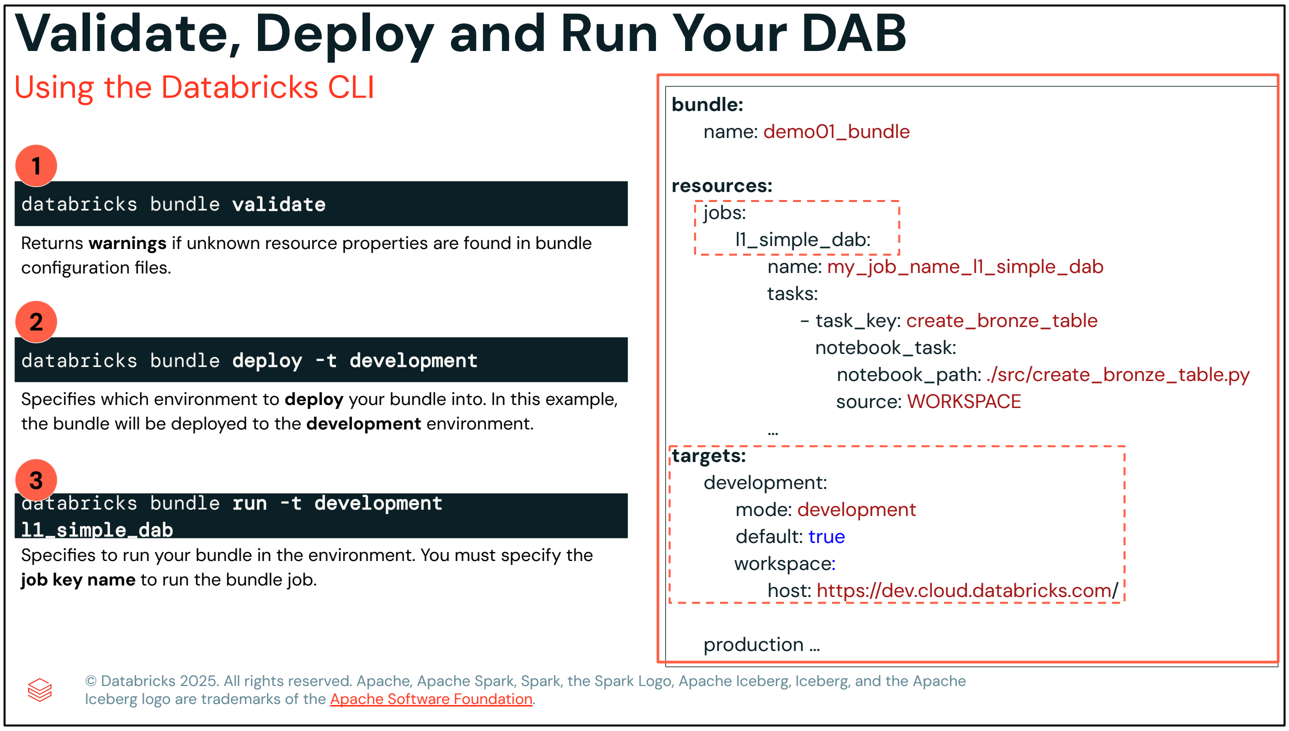
The top-level mapping key targets sets specific environments and environment configurations, including:

- The environment mode type, such as development or production.
- The default target environment, which should be set be set to the development environment. This ensures that if you don't specify where to deploy this bundle and run, it will default to the development environment.
- In the targets mapping, you can include various other configurations and configuration overrides for that specific target.

In this example, we include two target environments: **development** and **production**, each with unique configurations and overrides.

In this simple example, we specify the **mode** development and **default true** mappings for the development environment, with a specific development workspace URL.

For the production environment, we use the **mode production** and specify the production workspace URL.



Validate, Deploy and Run Your DAB

Using the Databricks CLI

1

```
databricks bundle validate
```

Returns **warnings** if unknown resource properties are found in bundle configuration files.

2

```
databricks bundle deploy -t development
```

Specifies which environment to **deploy** your bundle into. In this example, the bundle will be deployed to the **development** environment.

3

```
databricks bundle run -t development
l1_simple_dab
```

Specifies to run your bundle in the environment. You must specify the **job key name** to run the bundle job.

```
bundle:
    name: demo01_bundle

resources:
    jobs:
        l1_simple_dab:
            name: my_job_name_l1_simple_dab
            tasks:
                - task_key: create_bronze_table
                  notebook_task:
                      notebook_path: ./src/create_bronze_table.py
                      source: WORKSPACE
            …
targets:
    development:
        mode: development
        default: true
        workspace:
            host: https://dev.cloud.databricks.com/

    production …
```

Now that we have a good understanding of the databricks.yml file, let's explore how to validate, deploy, and run our Databricks Asset Bundle.

- First, you'll want to validate your bundle. You can do this with the Databricks CLI command **databricks bundle validate**.
- Next, you'll deploy the bundle to your Databricks Workspace. To do this, use the **databricks bundle deploy** command, the **-t** flag for target, and specify the environment you want to deploy to. In this example, we will deploy to the **development** environment. By default, development is set as the default target, so if you don't specify where to deploy, it will go to development. However, it's best practice to be explicit.
- Once the bundle is deployed to Databricks, you will typically want to run it. To run the job in the bundle, use the **databricks bundle run** command, the **-t** flag for development, and the key that specifies the job to run. Here the key is **l1_simple_dab**.

And that's it! We've covered how to create, validate, deploy, and run a simple Databricks Asset Bundle!

databricks

Thank you for completing this lesson and continuing your journey to develop your skills with us.