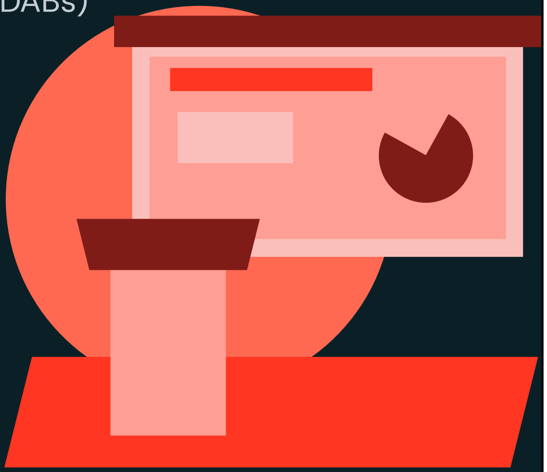
databricks

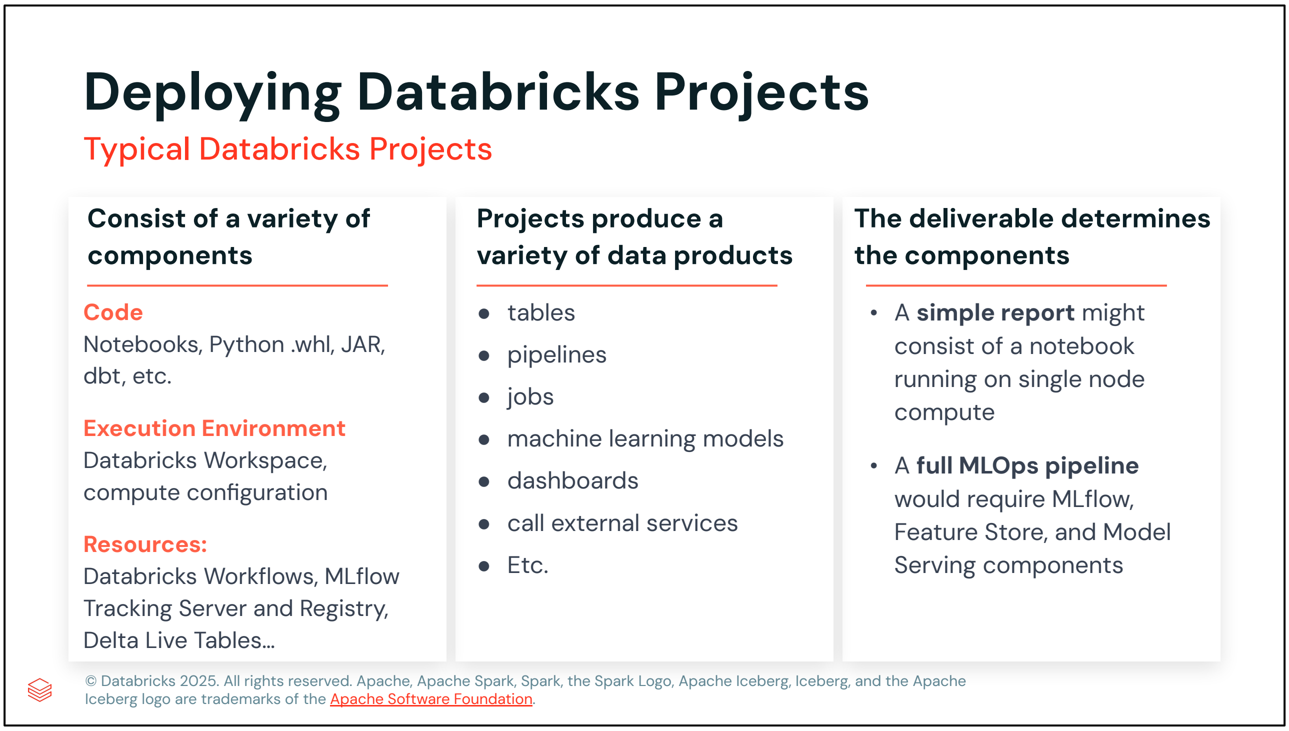
Deployment with Databricks Asset Bundles (DABs)

LECTURE

Deploying Databricks Projects



In this lecture, we cover the essentials of deploying projects on Databricks.



Deploying Databricks Projects

Typical Databricks Projects

Consist of a variety of components

Code
Notebooks, Python .whl, JAR, dbt, etc.

Execution Environment
Databricks Workspace, compute configuration

Resources:
Databricks Workflows, MLflow Tracking Server and Registry, Delta Live Tables...

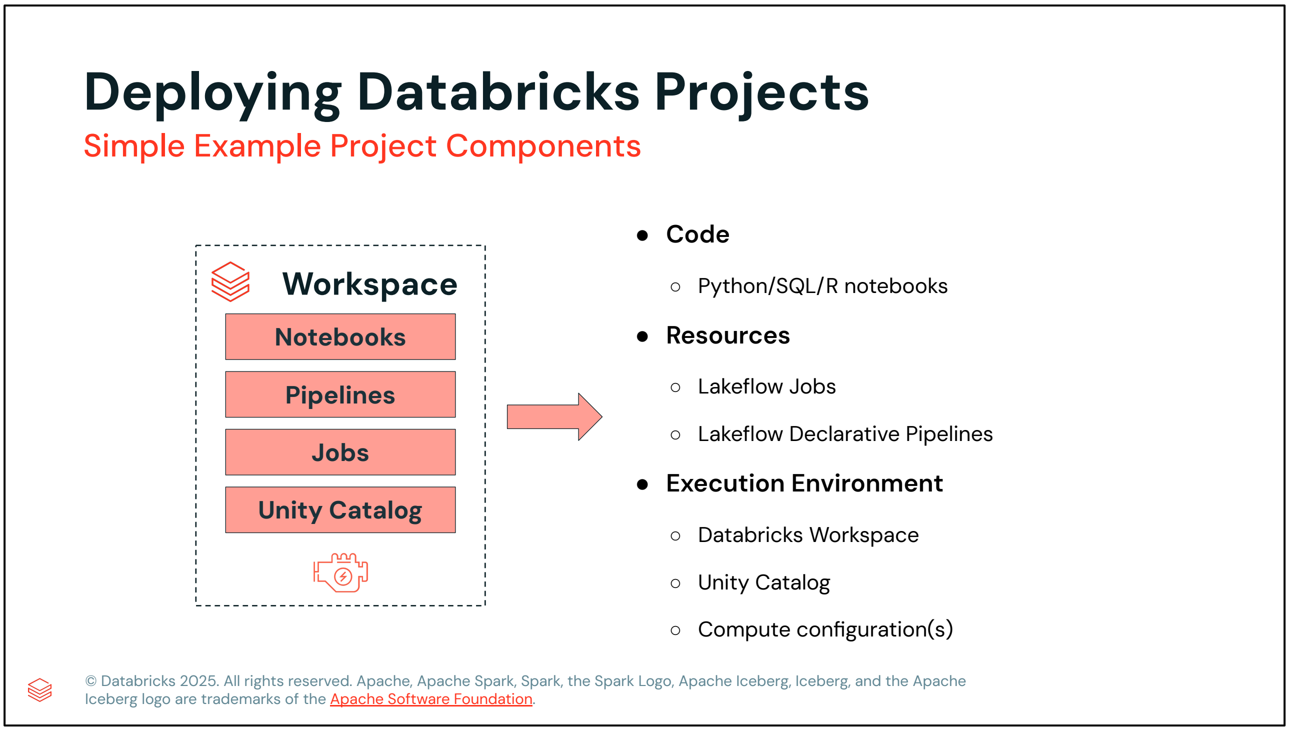
Projects produce a variety of data products

- tables
- pipelines
- jobs
- machine learning models
- dashboards
- call external services
- Etc.

The deliverable determines the components

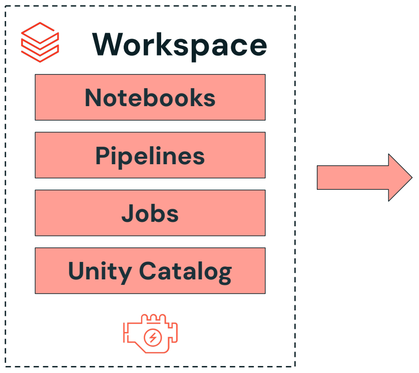
- A **simple report** might consist of a notebook running on single node compute
- A **full MLOps pipeline** would require MLflow, Feature Store, and Model Serving components

- Let's quickly recap the typical components of a Databricks project.
- First, a Databricks project consists of multiple components, including code, notebooks, Python files, Python wheel files, JARs, DBT, and more.
- It also includes an execution environment within one or more Databricks workspaces, along with specific compute configurations for processing.
- Lastly, the project may involve other Databricks resources, such as Lakeflow Jobs, MLflow, Lakeflow Declarative Pipelines, and more.
- Next, a Databricks project typically produces a variety of data products, including tables, pipelines, models, jobs, dashboards, and more.
- Lastly, the deliverables of your project determine the components you'll need.
- For example, a simple report might include one or more notebooks running on a basic single-node compute.
- On the other hand, a full MLOps pipeline would require multiple components, such as MLflow, the Feature Store, model serving components, notebooks, and more.



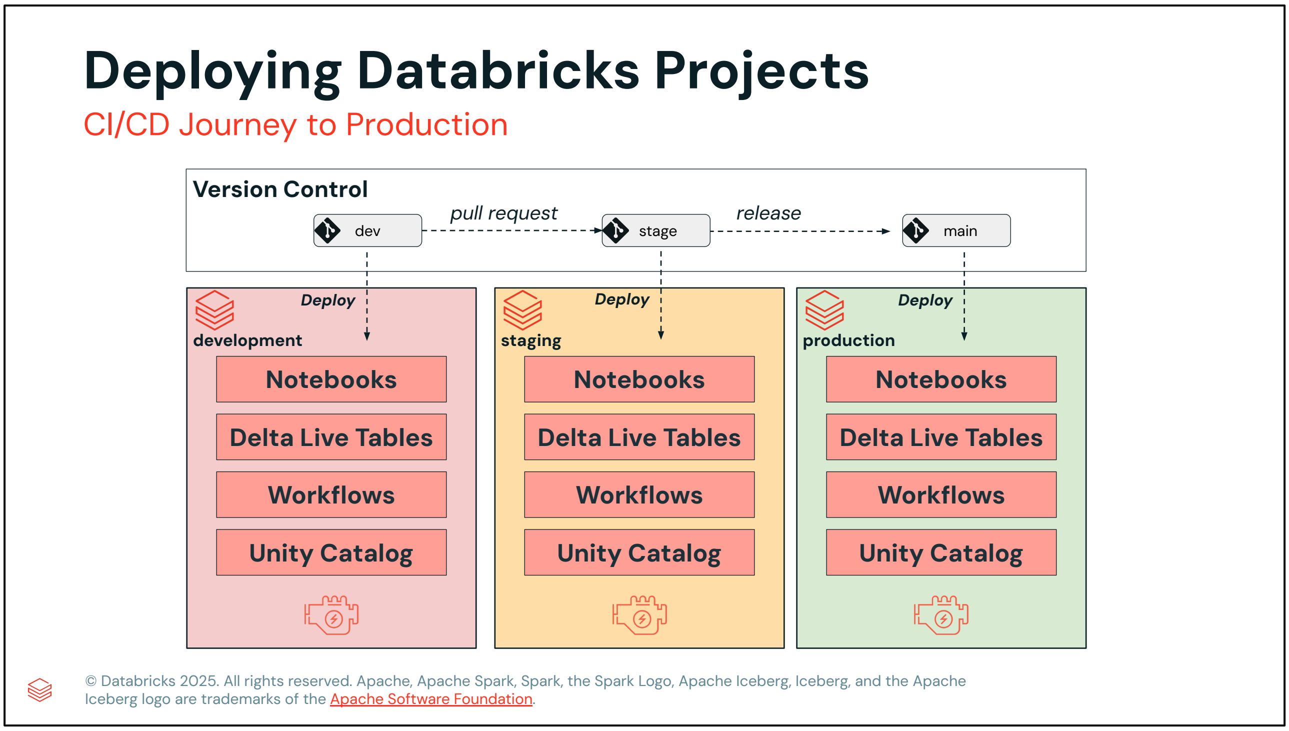
Deploying Databricks Projects

Simple Example Project Components



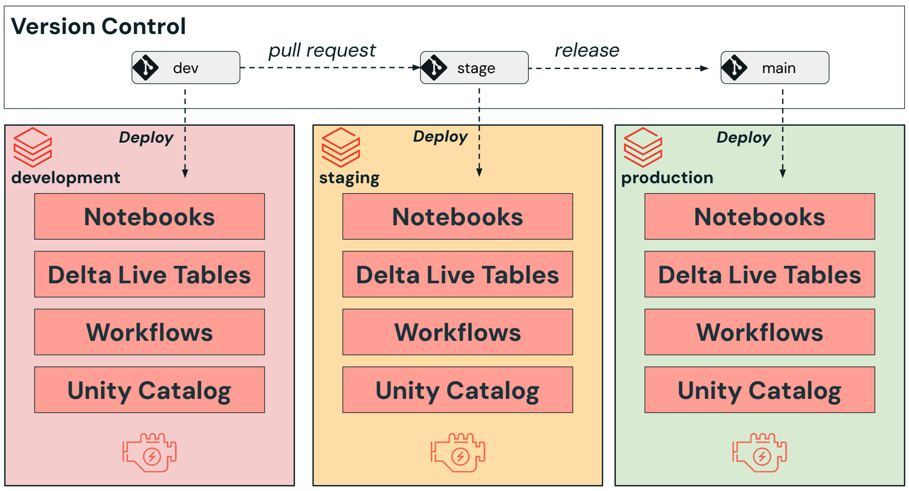
- **Code**
  - Python/SQL/R notebooks
- **Resources**
  - Lakeflow Jobs
  - Lakeflow Declarative Pipelines
- **Execution Environment**
  - Databricks Workspace
  - Unity Catalog
  - Compute configuration(s)

For example, a simple Data Engineering Databricks project could include notebooks, Delta Live Tables, a workflow, catalogs within Unity Catalog, and specific compute configurations for the project.



Deploying Databricks Projects

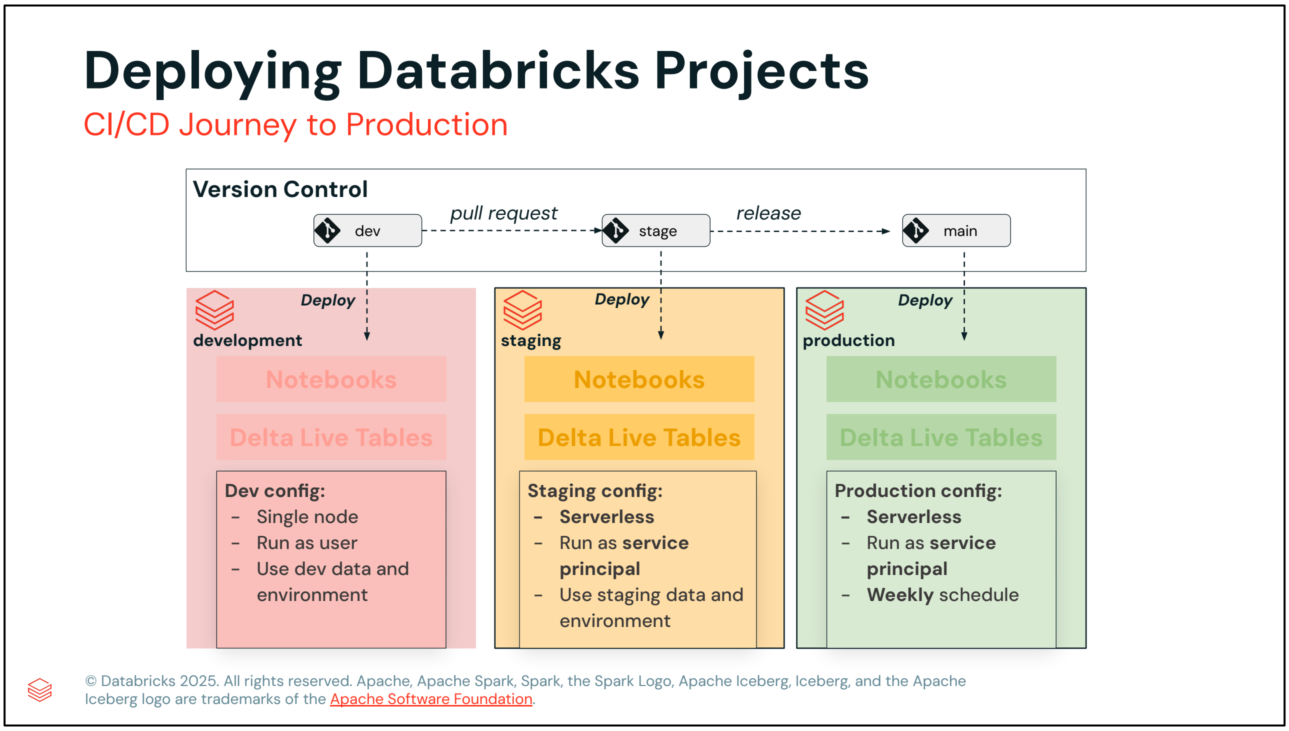
CI/CD Journey to Production



Let's take a look at a high-level CI/CD journey to production.

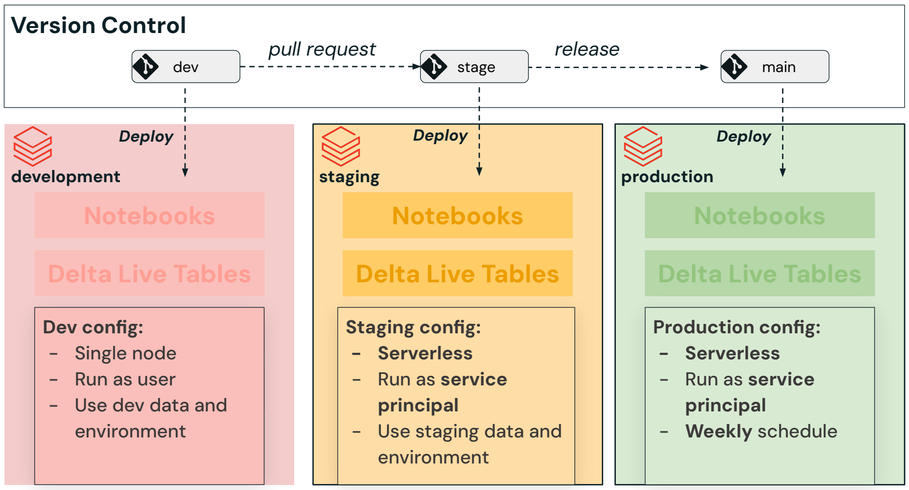
To start, you'll want to have a version control strategy in place. For example, your strategy might include dev, stage, and main branches.

- From here, begin by deploying your Databricks project to the development environment. This allows you to test any changes or updates made to your project on development data, using specific development configurations.
- After completing development and creating a pull request into your stage branch, you will then deploy to the staging environment. This ensures testing of your project continues on new staging data with the appropriate staging configurations.
- Once both your development and staging environments have completed testing and passed all tests, you can deploy to your main or production branch. This will fully deploy your project to production, utilizing production data and the necessary production configurations.



Deploying Databricks Projects

CI/CD Journey to Production



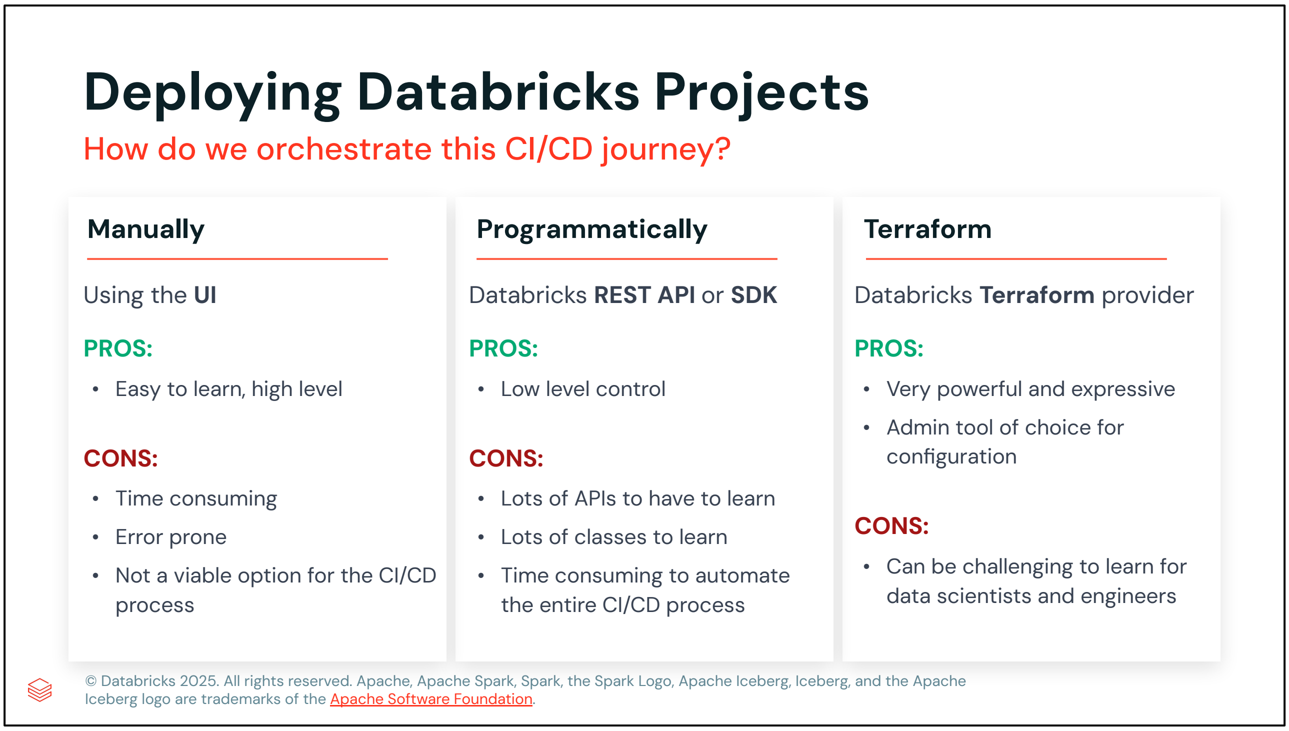
Dev config:
- Single node
- Run as user
- Use dev data and environment

Staging config:
- **Serverless**
- Run as **service principal**
- Use staging data and environment

Production config:
- **Serverless**
- Run as **service principal**
- **Weekly** schedule

Let's go over some example configurations for each environment.
For development, we'll configure the environment to use single-node compute, as our data is small and doesn't require a large cluster. We'll run the project using our user account on the development data.

- In staging, we aim to closely resemble the production environment. Here, we'll use Serverless compute to scale our resources as needed. We'll run the project using a service principal identity, which gives automated tools access to only the necessary Databricks resources, offering better security than using user or group access. This will be done with our staging data.
  Lastly, for production, we'll continue using Serverless compute and the service principal identity. The project will run using production data, and we'll set the project to run weekly to update the data for our consumers.



Deploying Databricks Projects

How do we orchestrate this CI/CD journey?

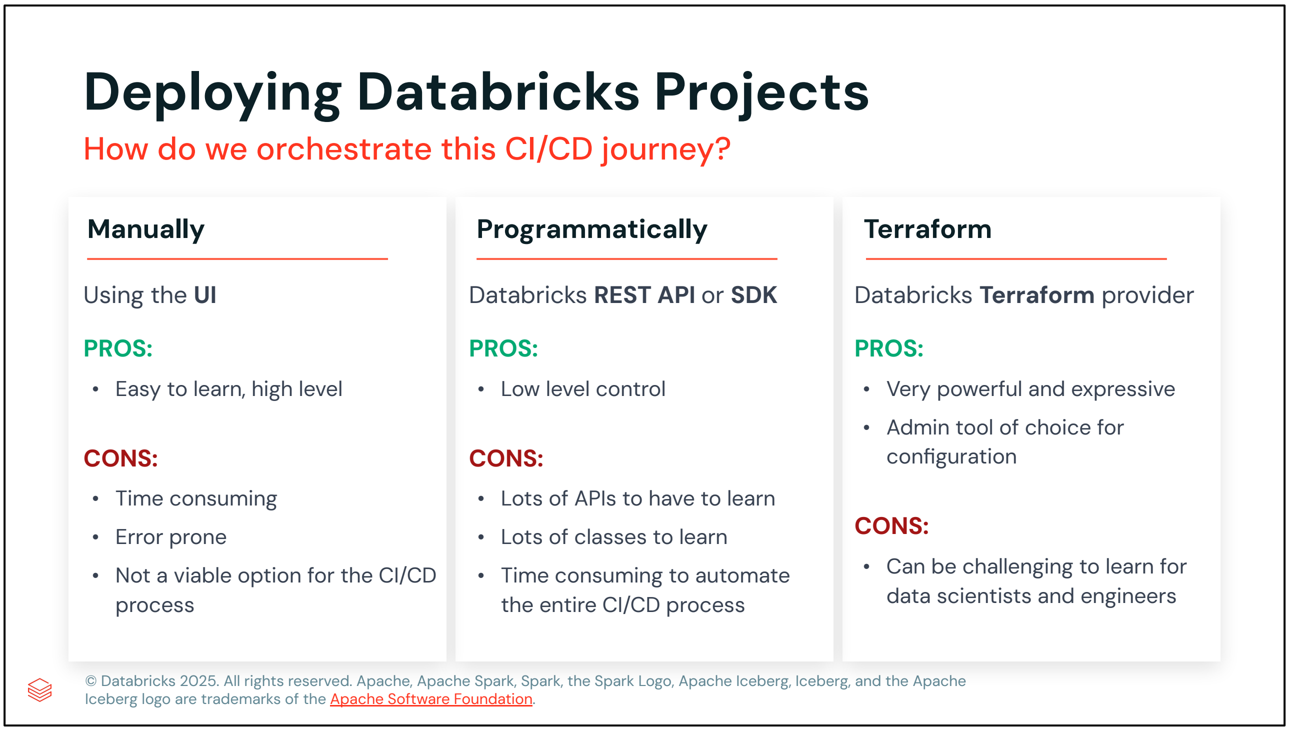
Manually	Programmatically	Terraform
Using the **UI**	Databricks **REST API** or **SDK**	Databricks **Terraform** provider
PROS:	**PROS:**	**PROS:**
• Easy to learn, high level	• Low level control	• Very powerful and expressive • Admin tool of choice for configuration
CONS:	**CONS:**	**CONS:**
• Time consuming • Error prone • Not a viable option for the CI/CD process	• Lots of APIs to have to learn • Lots of classes to learn • Time consuming to automate the entire CI/CD process	• Can be challenging to learn for data scientists and engineers

So, how can we orchestrate this CI/CD journey to production?

Let’s explore three different methods.
First, we can do this manually.

A pro of this method is that the user interface is easy to learn and navigate.

However, the cons include that it is extremely time-consuming to manually deploy the project every time. It’s also highly error-prone since it involves human interaction, and, importantly, it’s not a viable option for the CI/CD process where we want to automate the deployment process and easily move from one environment to the next.
Next, we can do this programmatically using the Databricks REST API or Databricks SDK.



Deploying Databricks Projects

How do we orchestrate this CI/CD journey?

Manually

Using the **UI**

PROS:

- Easy to learn, high level

CONS:

- Time consuming
- Error prone
- Not a viable option for the CI/CD process

Programmatically

Databricks **REST API** or **SDK**

PROS:

- Low level control

CONS:

- Lots of APIs to have to learn
- Lots of classes to learn
- Time consuming to automate the entire CI/CD process

Terraform

Databricks **Terraform** provider

PROS:

- Very powerful and expressive
- Admin tool of choice for configuration

CONS:

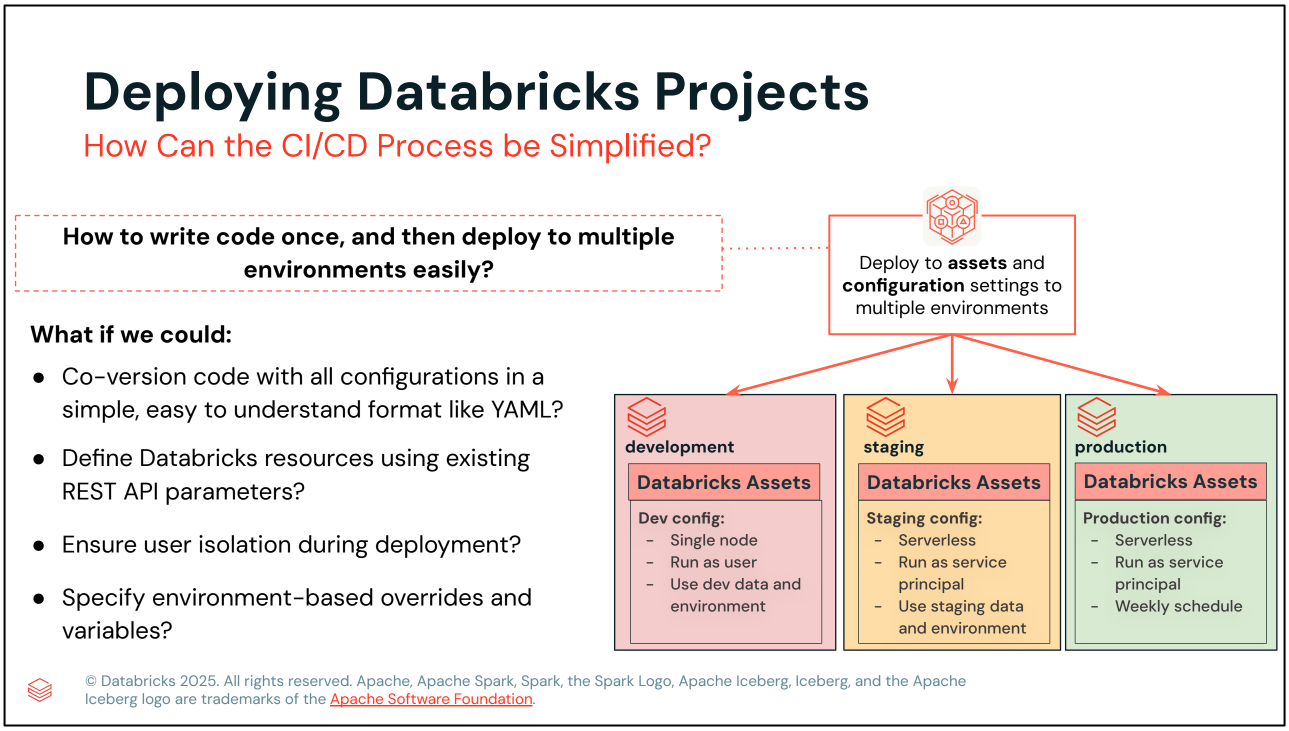
- Can be challenging to learn for data scientists and engineers

A pro of this method is that it gives you low-level control, allowing you to program everything you need.

However, the cons are that this approach requires intermediate to advanced programming knowledge. You'll need to learn a lot of APIs if you're using the REST API or a lot of classes for the SDK method, and coding everything to automate the entire CI/CD process can be extremely time-consuming.
Lastly, for those who are experienced with Terraform, you can use that.

It's a very powerful and expressive tool that administrators typically use to manage infrastructure.

While this method might work well for some, it can be challenging for data scientists and engineers who may not have a deep background in infrastructure management. It also adds another tool that you have to learn and manage.



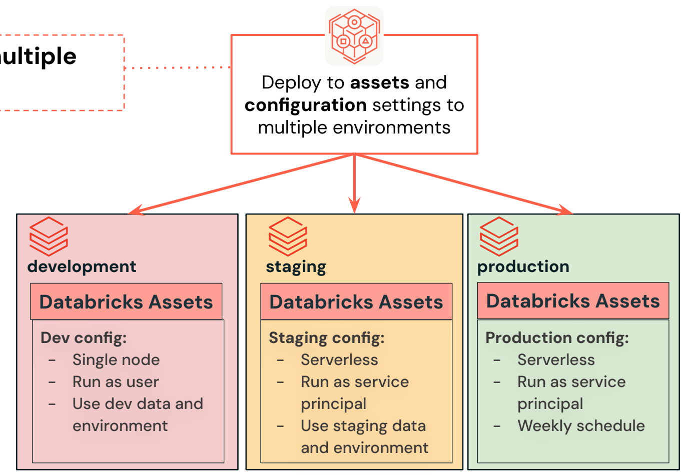
Deploying Databricks Projects

How Can the CI/CD Process be Simplified?

How to write code once, and then deploy to multiple environments easily?

What if we could:

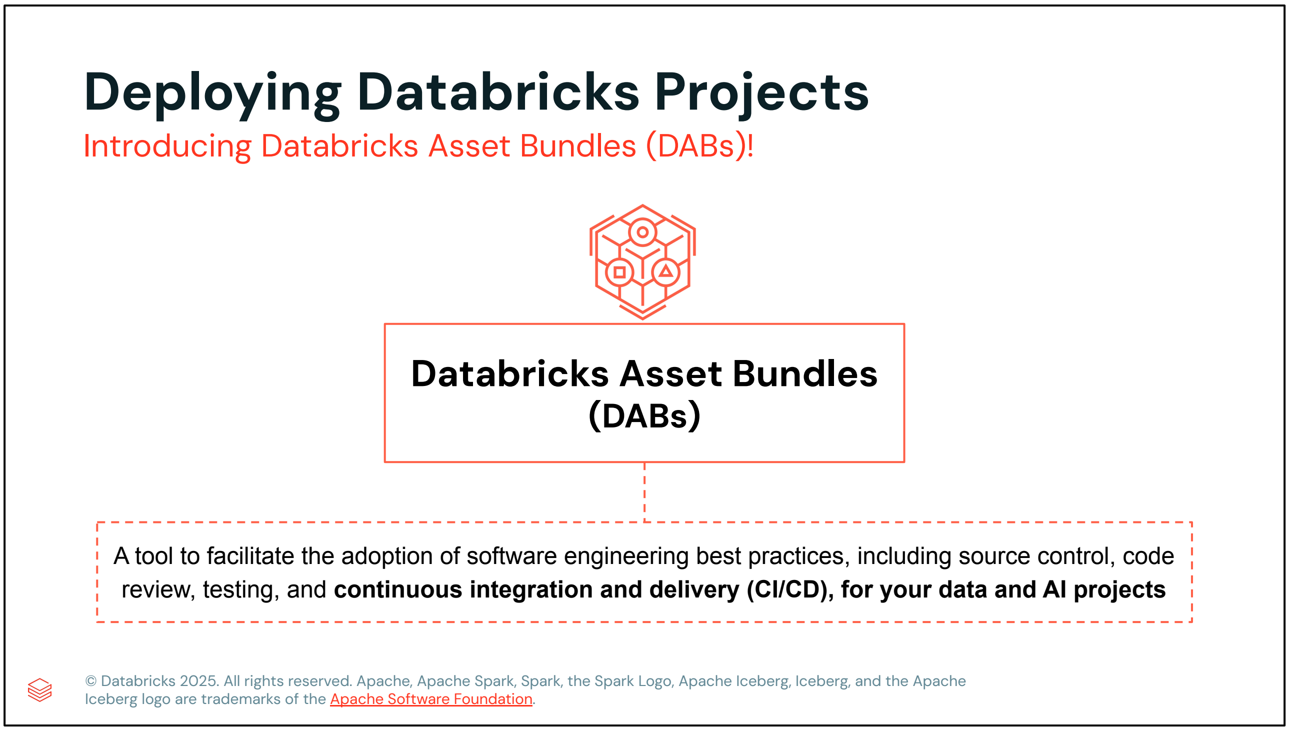
- Co-version code with all configurations in a simple, easy to understand format like YAML?
- Define Databricks resources using existing REST API parameters?
- Ensure user isolation during deployment?
- Specify environment-based overrides and variables?


Deploy to **assets** and **configuration** settings to multiple environments
development
Databricks Assets
Dev config:
– Single node
– Run as user
– Use dev data and environment
staging
Databricks Assets
Staging config:
– Serverless
– Run as service principal
– Use staging data and environment
production
Databricks Assets
Production config:
– Serverless
– Run as service principal
– Weekly schedule

So, how can we simplify the CI/CD process for Databricks projects? How can we write code once for our project, then easily deploy it to multiple environments while modifying our configurations?

What if we could?

- Co-version code with all configurations in a simple, easy-to-understand format like YAML? YAML is a human-readable format that organizes data in a simple, easy-to-read structure using indentation and minimal syntax. This makes it easier to understand and edit compared to other formats like XML or JSON.
- Define Databricks resources using existing REST API parameters? This ensures we keep parameters consistent with the other techniques.
- Ensure user isolation during deployment? This helps prevent users from interfering with each other's work during deployment.
- Specify environment-based overrides and variables? This allows us to easily override values based on the target environment we want to deploy to, ensuring flexibility in our configuration for each environment.



Deploying Databricks Projects

Introducing Databricks Asset Bundles (DABs)!

Databricks Asset Bundles (DABs)

A tool to facilitate the adoption of software engineering best practices, including source control, code review, testing, and **continuous integration and delivery (CI/CD), for your data and AI projects**

Introducing Databricks ASset Bundles (or DABs)!

A tool to facilitate the adoption of software engineering best practices, including source control, code review, testing, and continuous integration and delivery (CI/CD), for your data and AI projects.

Moving forward, Databricks recommends DABs for creating, developing, deploying, and testing jobs and other Databricks resources as source code.

databricks

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