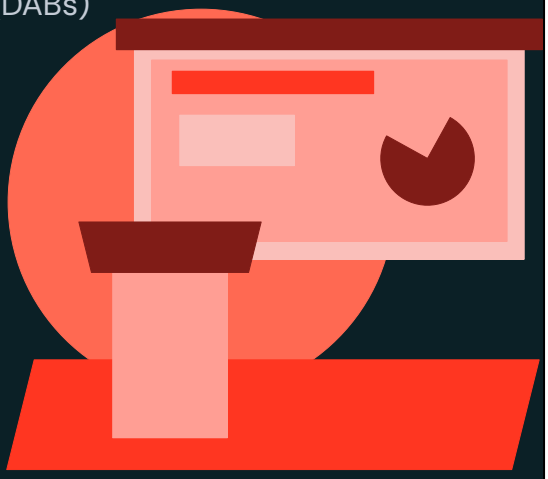
databricks

Deployment with Databricks Asset Bundles (DABs)

LECTURE

Variable Substitutions in DABs



In this lecture, we explore variable substitutions in Databricks Asset Bundles (DABs), including creating custom, complex, and lookup variables, plus environment overrides.

Variables in Databricks Asset Bundles (DABs)

Overview

Default Substitutions

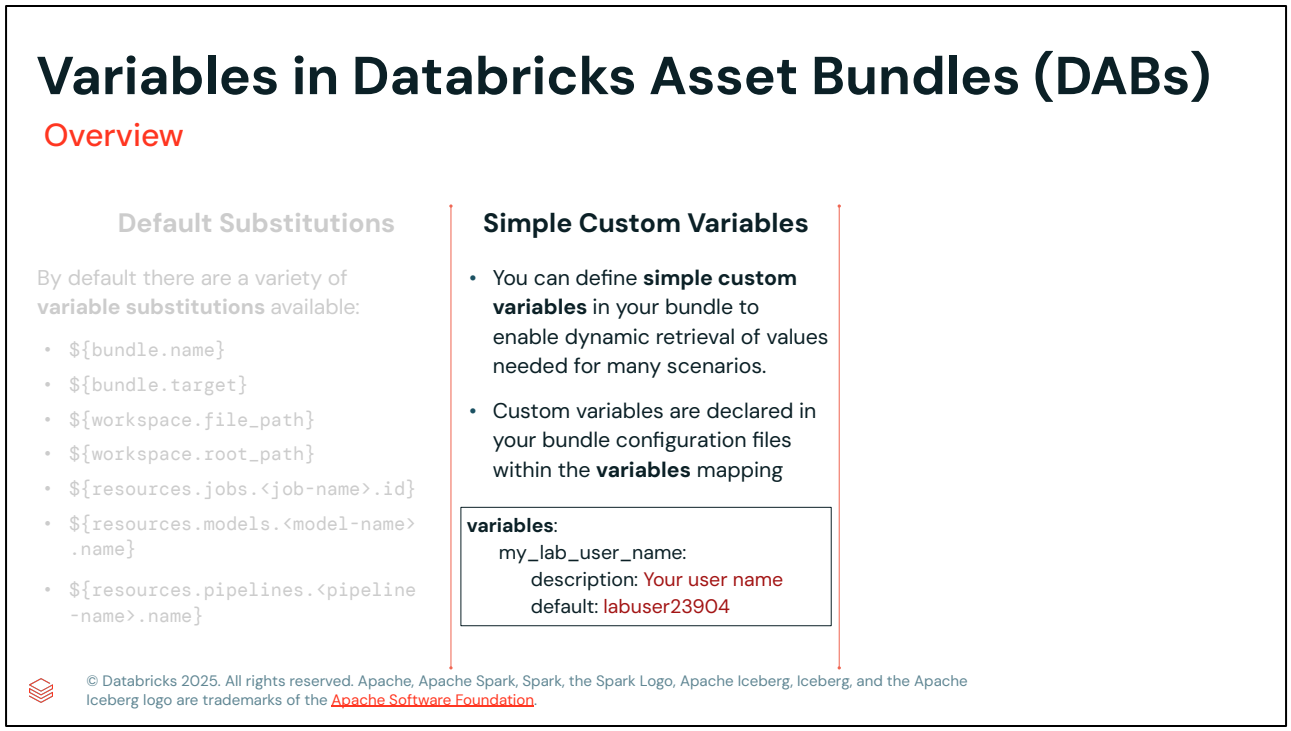
By default there are a variety of **variable substitutions** available:

- `${bundle.name}`
- `${bundle.target}`
- `${workspace.file_path}`
- `${workspace.root_path}`
- `${resources.jobs.<job-name>.id}`
- `${resources.models.<model-name>.name}`
- `${resources.pipelines.<pipeline-name>.name}`

Databricks Asset Bundles support substitutions and custom variables, making your bundle configuration files more modular and reusable. Both substitutions and custom variables allow dynamic retrieval of values, meaning that settings can be determined at the time the bundle is deployed and run.

To use a variable in your YAML file, reference it using a dollar sign, curly braces, and the mapping, then the variable name.

By default, a variety of substitutions are available. Some common ones are shown on the screen. A few to keep in mind include bundle.target to get the value of the targe environment, workspace.file_path for the workspace file path, and resources.jobs.job name or id, to name a few.



You can also use simple custom variables you define in your Databricks Asset Bundles.

Custom variables allow you to enable dynamic retrieval of values needed for various scenarios. These variables are declared in your bundle configuration files within the variables mapping.

For example, under the variables top-level mapping, you can specify any number of custom variables.

In this example, we'll create a variable named **my_lab_user_name**, followed by a description and its default value of *labuser23904*.

Variables in Databricks Asset Bundles (DABs)

Overview

Default Substitutions

By default there are a variety of **variable substitutions** available:

- `${bundle.name}`
- `${bundle.target}`
- `${workspace.file_path}`
- `${workspace.root_path}`
- `${resources.jobs.<job-name>.id}`
- `${resources.models.<model-name>.name}`
- `${resources.pipelines.<pipeline-name>.name}`

Simple Custom Variables

- You can define **simple custom variables** in your bundle to enable dynamic retrieval of values needed for many scenarios.
- Custom variables are declared in your bundle configuration files within the **variables** mapping

```
variables:
  my_lab_user_name:
    description: Your user name
    default: labuser23904
```

Complex Variables

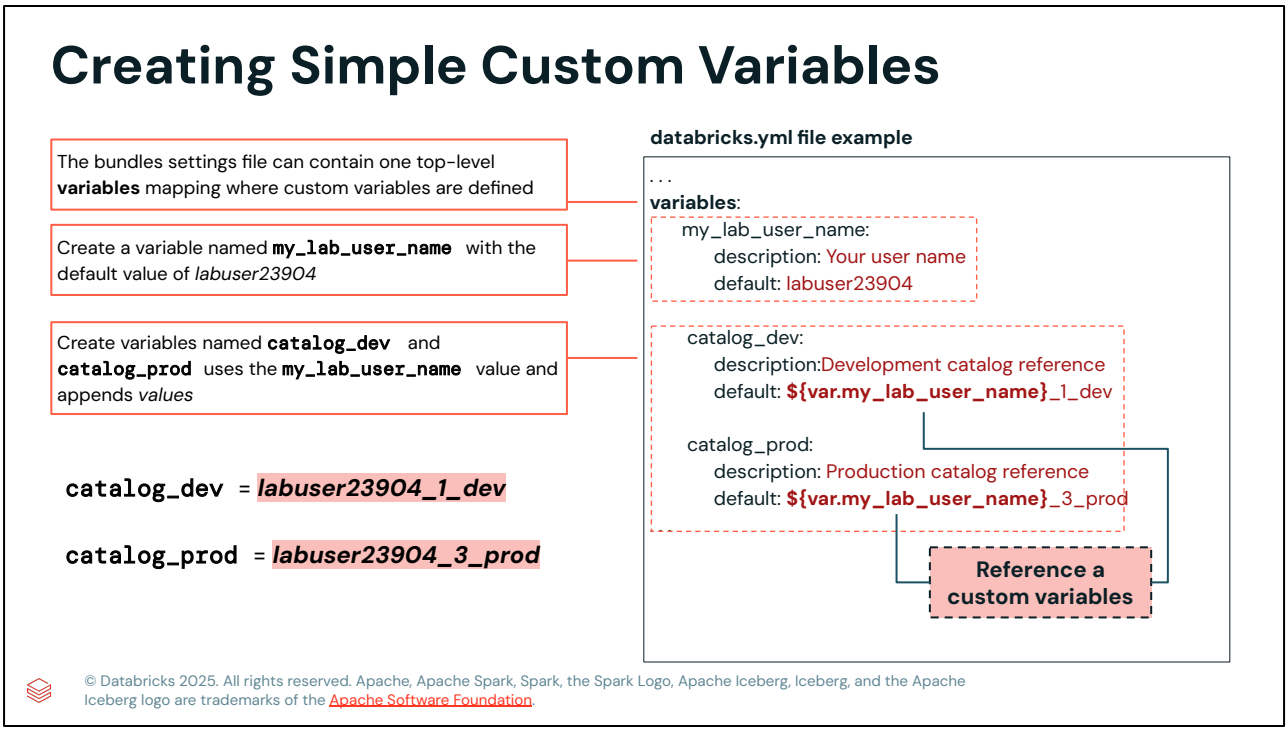
- A custom variable is assumed to be of type string unless you define it as a **complex variable**.
- Define a custom variable by setting the type to **complex**

```
variables:
  my_cluster:
    description: "My cluster"
    type: complex
    default:
      spark_version: "15.4.x-scala2.11"
      node_type_id: "Standard_DS3_v2"
      num_workers: 2
```


You can define both simple and complex custom variables in your Databricks Asset Bundles to enable dynamic retrieval of values for various scenarios.

By default, a simple custom variable is assumed to be of type string. However, if you need more complex data, you can define a custom variable as a complex type using the **type complex** mapping.

For example, in this case, we create a complex variable named **my_cluster**, with a default value specifying the Spark version, node type, and the number of workers.

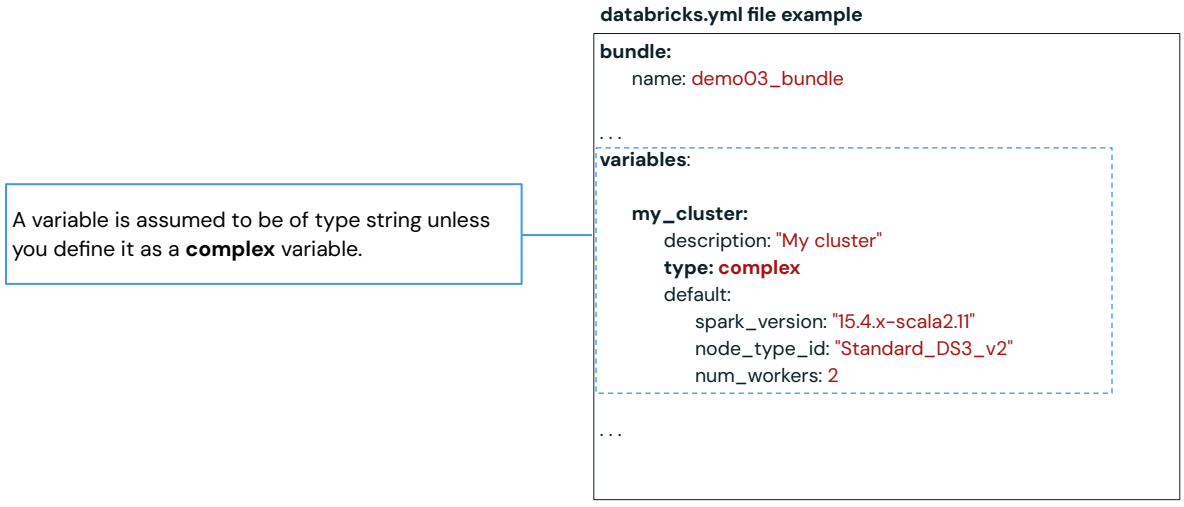


Let's look at an example of creating and referencing simple custom variables in your databricks.yml file.

- First, you define the **variables** top-level mapping. Under **variables**, you can add your custom variables. The bundle settings file can only contain one top-level variables mapping, where all your custom variables are defined.
- Create a variable named **my_lab_user_name** with a default value of *labuser23904*.
- You can then reference the custom variable **my_lab_user_name** within other variables or mappings. In this example, we will use the **my_lab_user_name** variable to create two new variables: **catalog_dev** and **catalog_prod**, appending the strings *_1_dev* and *_3_prod* to the user name. To reference a custom variable, use the top-level mapping var, followed by a dot and the variable name **my_lab_user_name** within curly braces, and then append the literal string.
- As a result, the **catalog_dev** variable will be equal to *labuser23904_1_dev*, and the **catalog_prod** variable will be equal to *labuser23904_3_prod*.

This technique allows you to reference or override variables throughout the databricks.yml file. We'll look at more examples of how to apply this.

Defining a Complex Variable



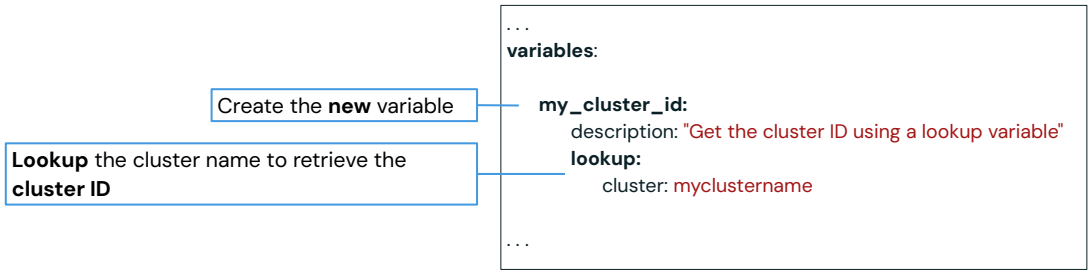
By default, a variable is assumed to be of type string unless you define it as a complex variable.

In this example, we define the variable **my_cluster** as type complex and specify its default values. This allows you to provide more structured data for the variable, such as specifying a Spark version, node type, and number of workers.

Lookup Variables

Dynamically Retrieve an Object's Value

For specific object types you can **define a lookup** for your custom variable to retrieve the object's ID



For specific object types, you can define a lookup for your custom variable to retrieve the object's ID using this format:

First, specify the variable name. In this example, the variable is called **my_cluster_id**. Then, use the lookup mapping to define what to look up. In this case, we are looking up the cluster name **myclustername** to retrieve the corresponding cluster ID value.

If a lookup is defined for a variable, the ID of the object with the specified name is used as the value of the variable. This ensures the correct resolved ID of the object is always used for the variable.

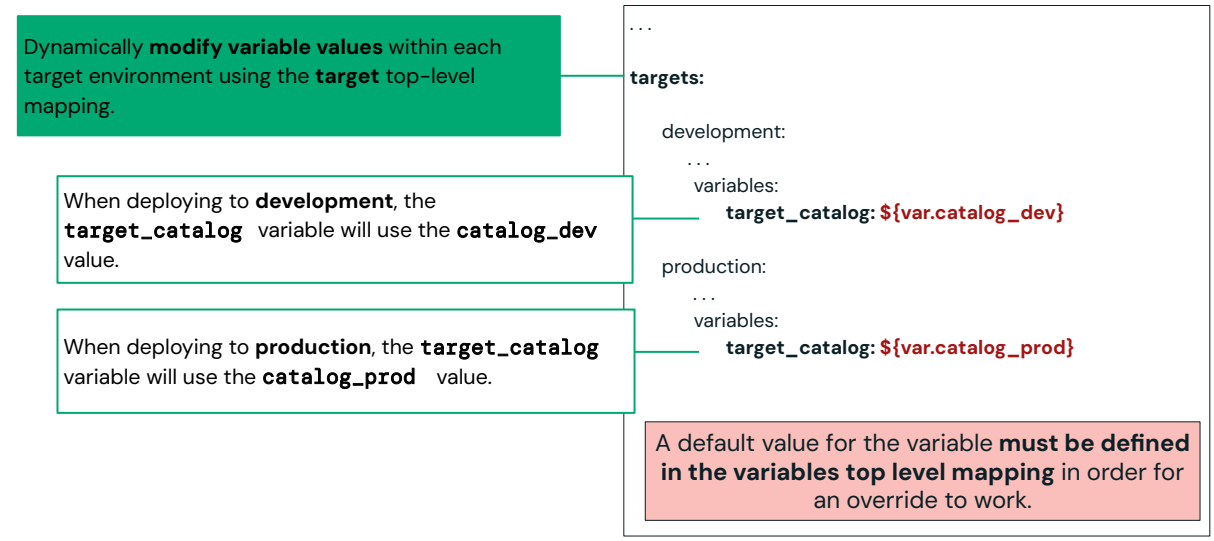
Lookup Variables

Can Define Lookups for the a Variety of Environment Configurations

- alert
- cluster_policy
- cluster
- dashboard
- instance_pool
- job
- metastore
- notification_destination
- pipeline
- query
- service_principal
- warehouse

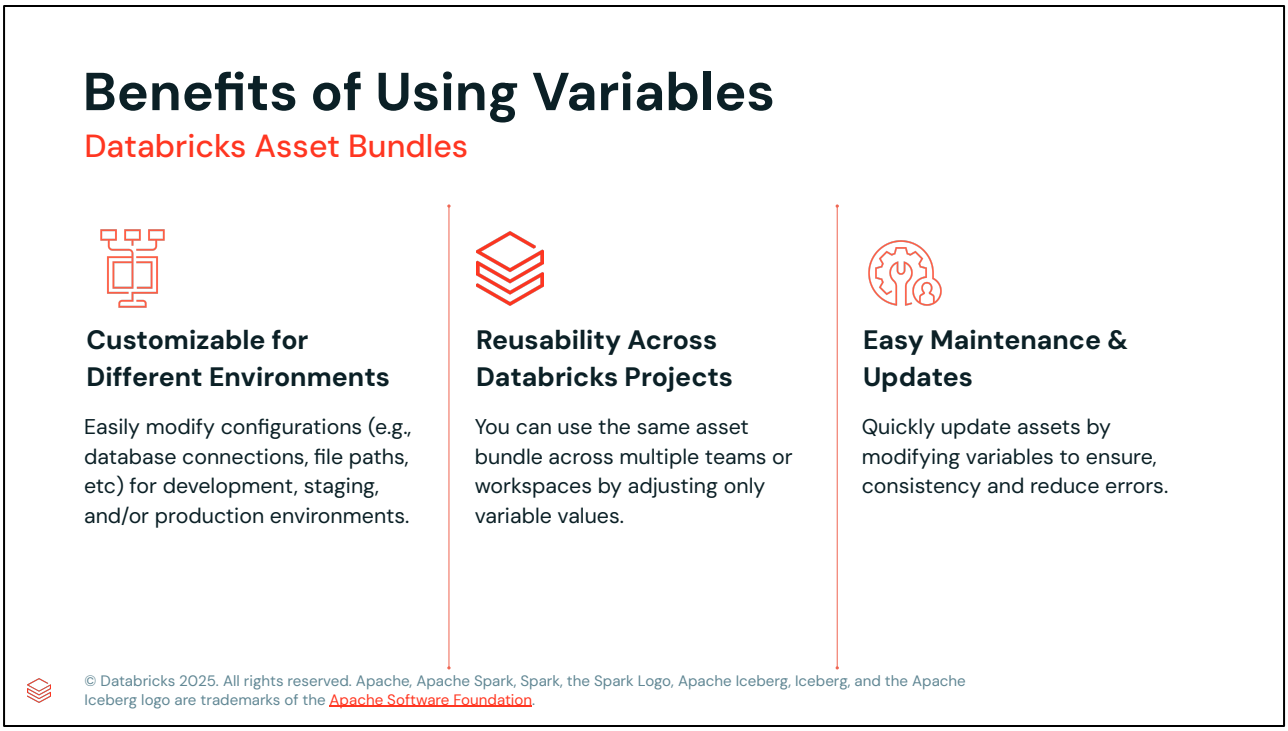
You can use lookup variables for the alert, cluster_policy, cluster, dashboard, instance_pool, job, metastore, notification_destination, pipeline, query, service_principal, and warehouse object types at the time of this recording.

Target Environment Variable Overrides



Custom variables can then be used throughout your databricks.yml file. One of the places you can use these is within your targets mapping. This allows you to dynamically modify variable values for each target environment.

- For example, let's say we have a custom variable named **target_catalog** that is used to populate a job parameter to read and write data from a specific catalog. You can modify its value for each environment. In this case, when deploying to the development environment, the **target_catalog** variable will use the **catalog_dev** value.
- When deploying to production, the **target_catalog** variable will use the **catalog_prod** value.
- If you do not specify an override for your variable, it will default to its original value.
- Please be aware that a default value for the variable must be defined in the variables top-level mapping for an override to work. If a variable is not assigned a default value, the override will not function as expected.



There are several benefits to using variables in your databricks.yml bundle configuration file:

- First, you can customizable them for Different Environments: Variables allow you to easily modify configurations, such as catalogs, file paths, and other settings, for development, staging, and production environments.
- They provide Reusability Across Databricks Projects: You can use the same asset bundle across multiple teams or workspaces by adjusting only the variable values, promoting consistency and reducing redundancy.
- Lastly, you can Easy maintain and update them: With variables, you can quickly update assets and configurations, ensuring consistency across environments and reducing the risk of errors.

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