KodeKloud

Follow us on https://kodekloud.com/ to learn more about us.

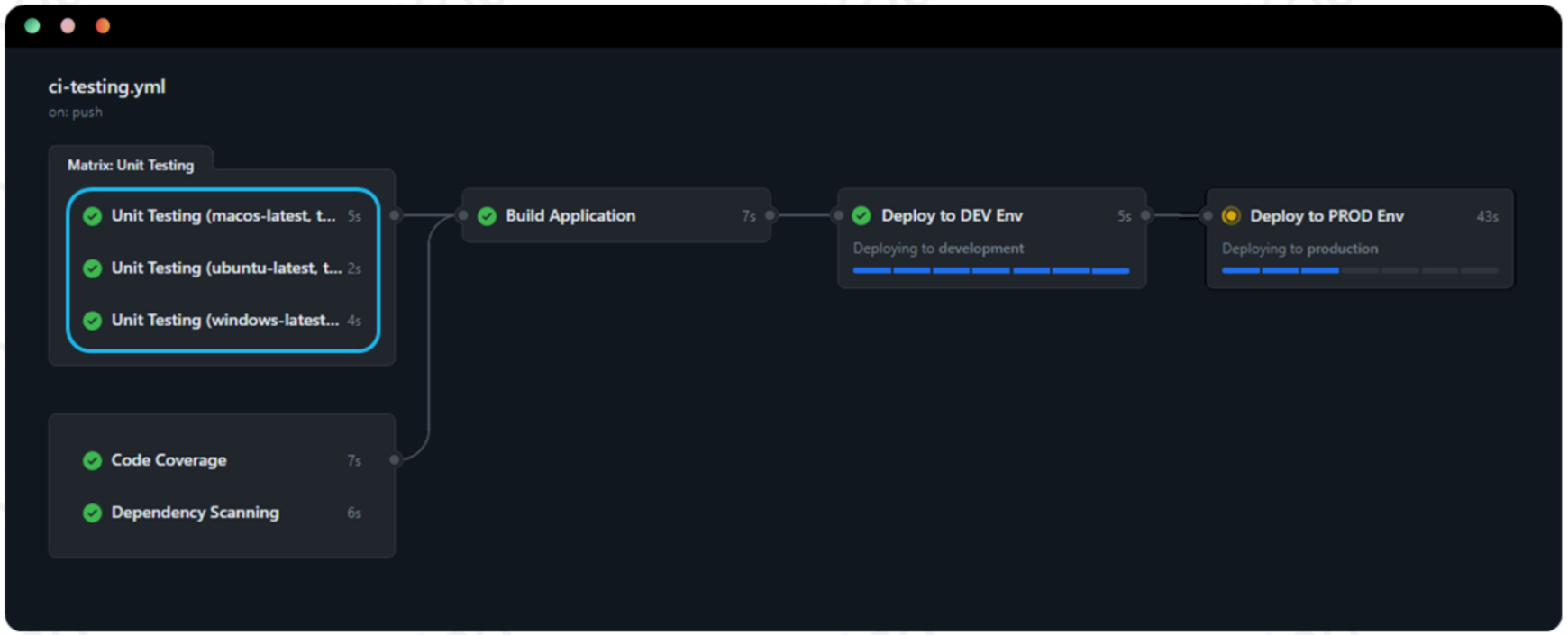
Project Status Meeting - 4

Project Status Meeting - 4

Priority	Task	Assigned	Status	Comments/Issue
0	Understand Requirement	Alice	Completed	
1	Unit Testing	Alice	Completed	
2	Code Coverage	Alice	Completed	
3	Containerization	Alice	Completed	
4	Kubernetes Dev Deployment	Alice	Completed	Re-use steps for other projects
5	Dev Integration Testing	Alice	Completed	
6	Manual Approval	Alice	Completed	
7	Kubernetes Prod Deployment	Alice	Completed	
8	Prod Integration Testing	Alice	Completed	Re-use steps for other projects

Understanding Reusable Workflows

Reusable Workflows



This workflow diagram is for an NodeJS application which contains unit testing, code coverage analysis, dependency scanning, build application and deployment jobs.

The deploy to dev environment job is only runs after all the previous jobs completes successfully.

The deploy to prod environment job only runs after the "deploy to dev" job has completed successfully.

Given that both these tasks are specific to particular environments, the workflow execution includes a progress bar showing the number of steps within each job. Both the development and production deployment jobs consist of 7 steps each, as shown in the diagram.

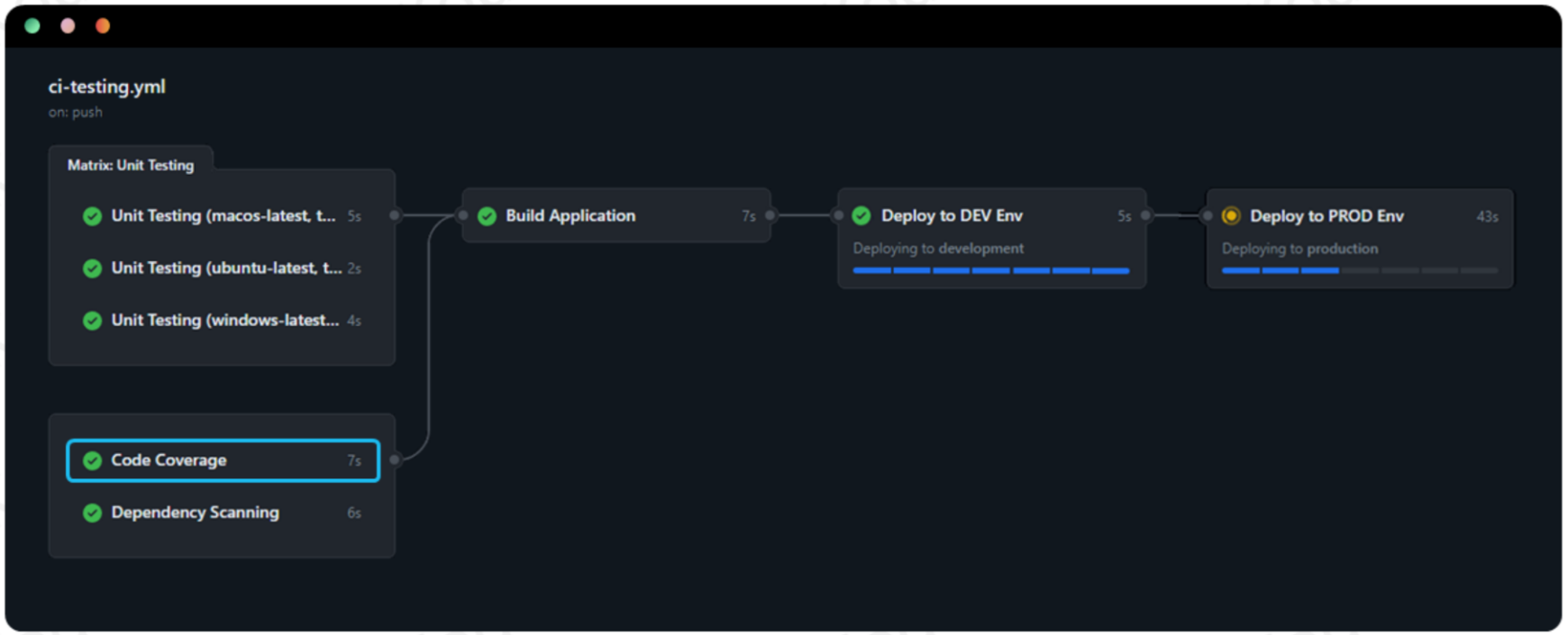
Let's assume the organization or team is responsible for various projects developed in different programming languages like Java, Python, .NET, Go, etc. Each project has a distinct unit testing, code coverage analysis, dependency scanning, and application building. However, when it comes to deployment, all these project workflows share the same deployment jobs and steps as those used in the NodeJS application workflow.

In general, one might consider copying and pasting these jobs from one workflow to another. However, this approach leads to to code duplication, and any necessary changes must be made in multiple workflows, which can be cumbersome.

This is where reusable workflows come into play.

Reusing workflows avoids duplication. Rather than copying and pasting from one workflow to another, you can make workflows reusable. This makes workflows easier to maintain and allows you to create new workflows more quickly by leveraging the work of others, similar to how actions work. eusable workflows also promote best practices by enabling the use of well-designed, tested, and proven-effective workflows.

Reusable Workflows



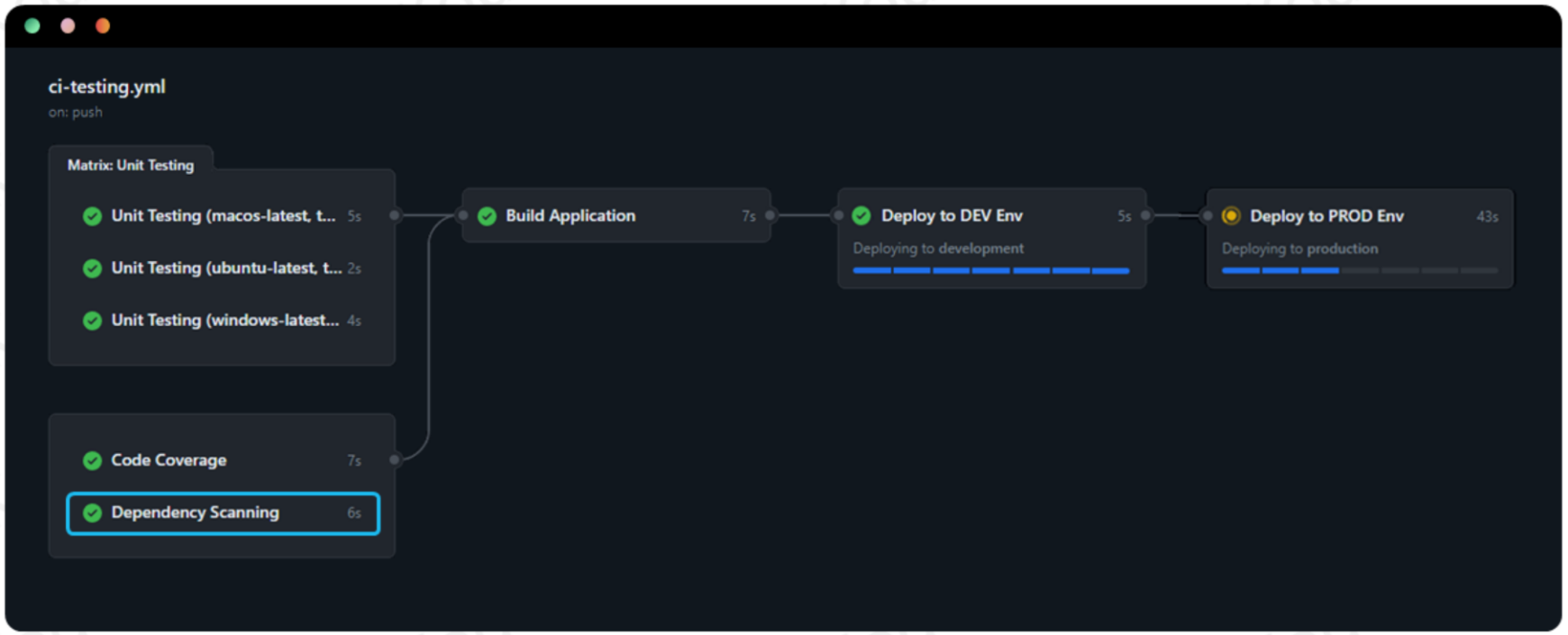
This workflow diagram is for an NodeJS application which contains unit testing, code coverage analysis, dependency scanning, build application and deployment jobs.

The deploy to dev environment job is only runs after all the previous jobs completes successfully.

The deploy to prod environment job only runs after the "deploy to dev" job has completed successfully.

Given that both these tasks are specific to particular environments, the workflow execution includes a progress bar showing the number of steps within each job. Both the development and production deployment jobs consist of 7 steps each, as shown in the diagram.

Reusable Workflows



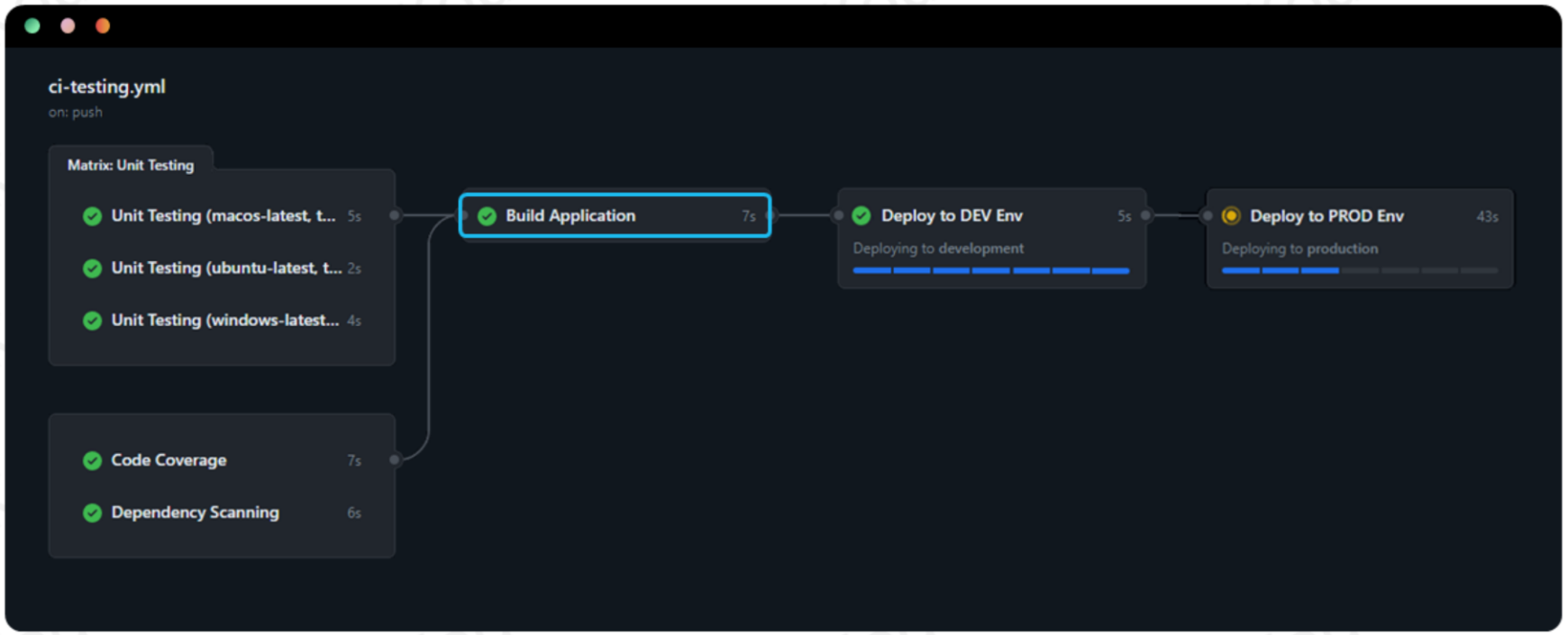
This workflow diagram is for an NodeJS application which contains unit testing, code coverage analysis, dependency scanning, build application and deployment jobs.

The deploy to dev environment job is only runs after all the previous jobs completes successfully.

The deploy to prod environment job only runs after the "deploy to dev" job has completed successfully.

Given that both these tasks are specific to particular environments, the workflow execution includes a progress bar showing the number of steps within each job. Both the development and production deployment jobs consist of 7 steps each, as shown in the diagram.

Reusable Workflows



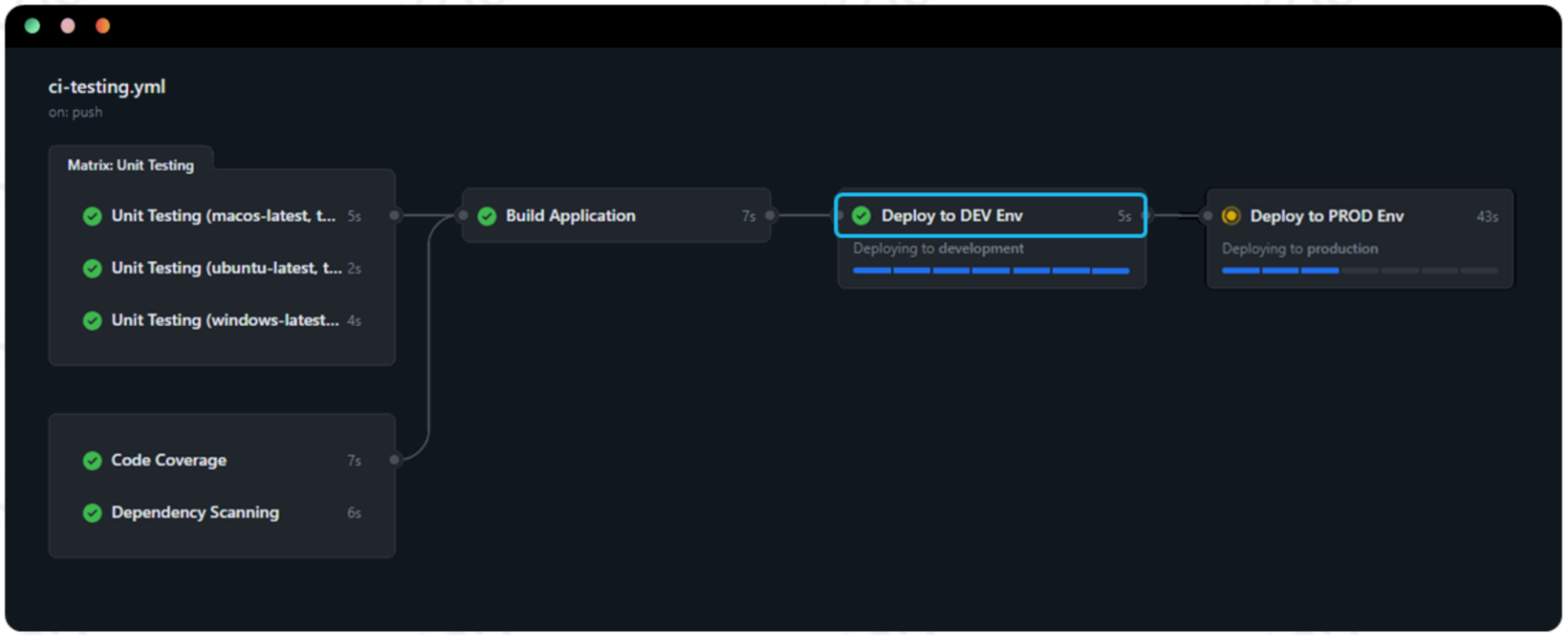
This workflow diagram is for an NodeJS application which contains unit testing, code coverage analysis, dependency scanning, build application and deployment jobs.

The deploy to dev environment job is only runs after all the previous jobs completes successfully.

The deploy to prod environment job only runs after the "deploy to dev" job has completed successfully.

Given that both these tasks are specific to particular environments, the workflow execution includes a progress bar showing the number of steps within each job. Both the development and production deployment jobs consist of 7 steps each, as shown in the diagram.

Reusable Workflows



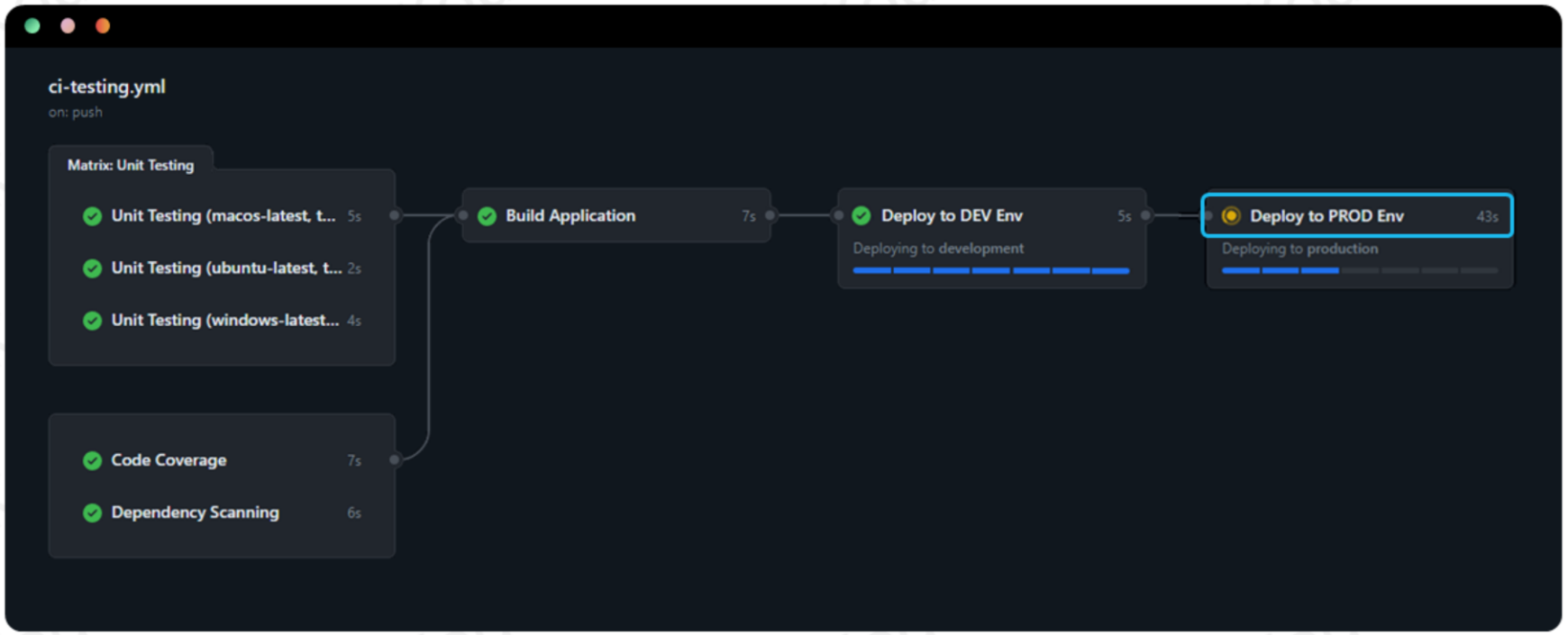
This workflow diagram is for an NodeJS application which contains unit testing, code coverage analysis, dependency scanning, build application and deployment jobs.

The deploy to dev environment job is only runs after all the previous jobs completes successfully.

The deploy to prod environment job only runs after the "deploy to dev" job has completed successfully.

Given that both these tasks are specific to particular environments, the workflow execution includes a progress bar showing the number of steps within each job. Both the development and production deployment jobs consist of 7 steps each, as shown in the diagram.

Reusable Workflows



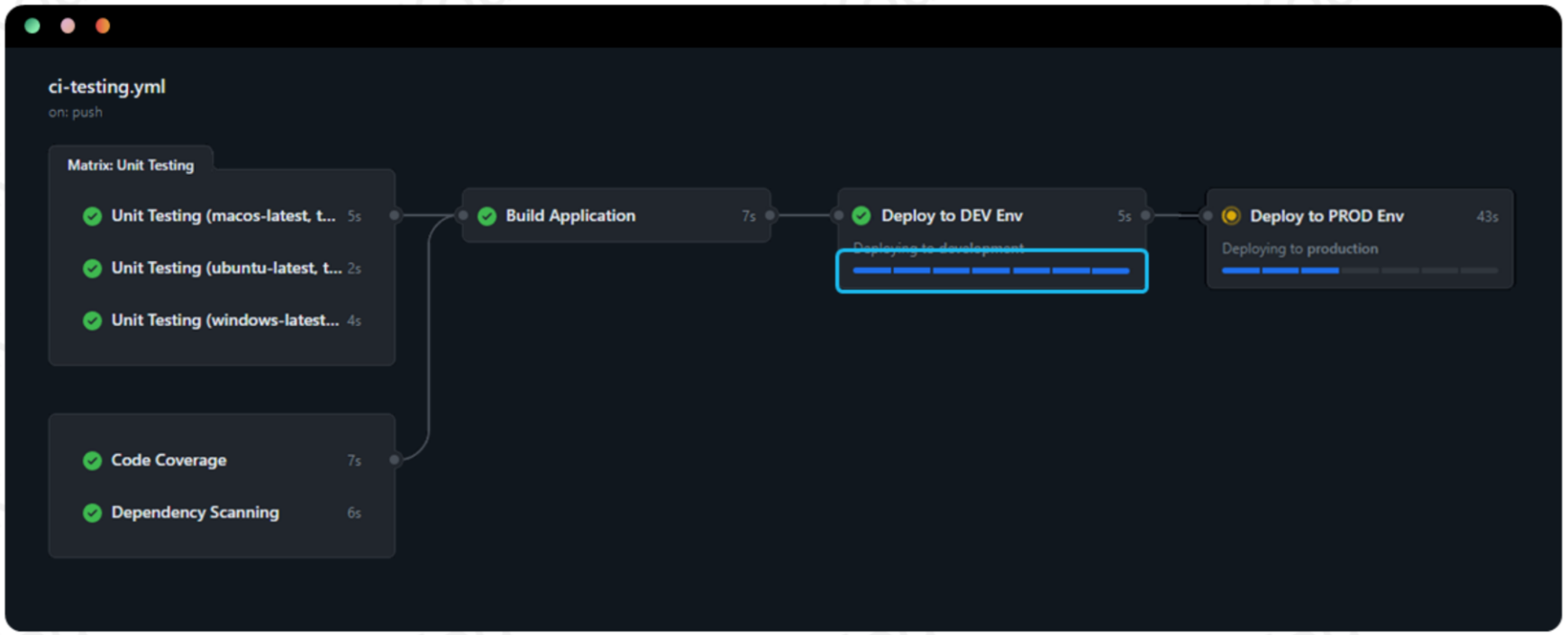
This workflow diagram is for an NodeJS application which contains unit testing, code coverage analysis, dependency scanning, build application and deployment jobs.

The deploy to dev environment job is only runs after all the previous jobs completes successfully.

The deploy to prod environment job only runs after the "deploy to dev" job has completed successfully.

Given that both these tasks are specific to particular environments, the workflow execution includes a progress bar showing the number of steps within each job. Both the development and production deployment jobs consist of 7 steps each, as shown in the diagram.

Reusable Workflows



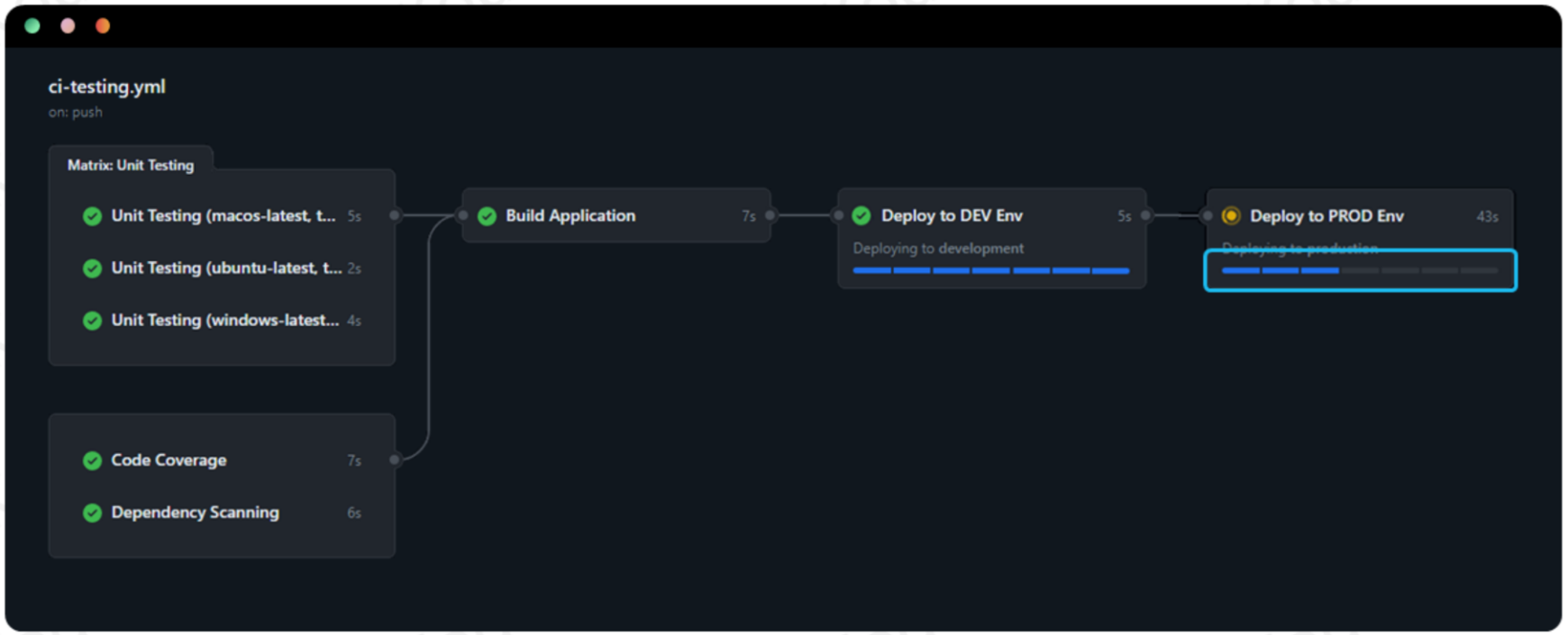
This workflow diagram is for an NodeJS application which contains unit testing, code coverage analysis, dependency scanning, build application and deployment jobs.

The deploy to dev environment job is only runs after all the previous jobs completes successfully.

The deploy to prod environment job only runs after the "deploy to dev" job has completed successfully.

Given that both these tasks are specific to particular environments, the workflow execution includes a progress bar showing the number of steps within each job. Both the development and production deployment jobs consist of 7 steps each, as shown in the diagram.

Reusable Workflows



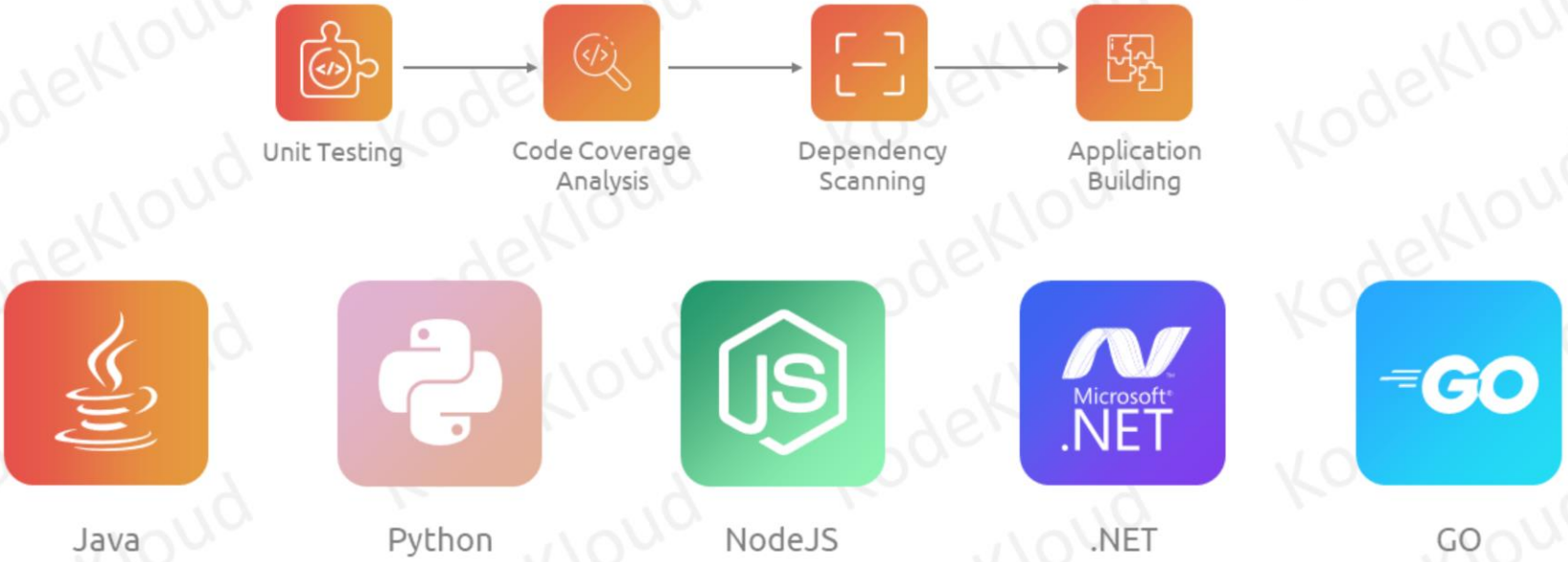
This workflow diagram is for an NodeJS application which contains unit testing, code coverage analysis, dependency scanning, build application and deployment jobs.

The deploy to dev environment job is only runs after all the previous jobs completes successfully.

The deploy to prod environment job only runs after the "deploy to dev" job has completed successfully.

Given that both these tasks are specific to particular environments, the workflow execution includes a progress bar showing the number of steps within each job. Both the development and production deployment jobs consist of 7 steps each, as shown in the diagram.

Reusable Workflows

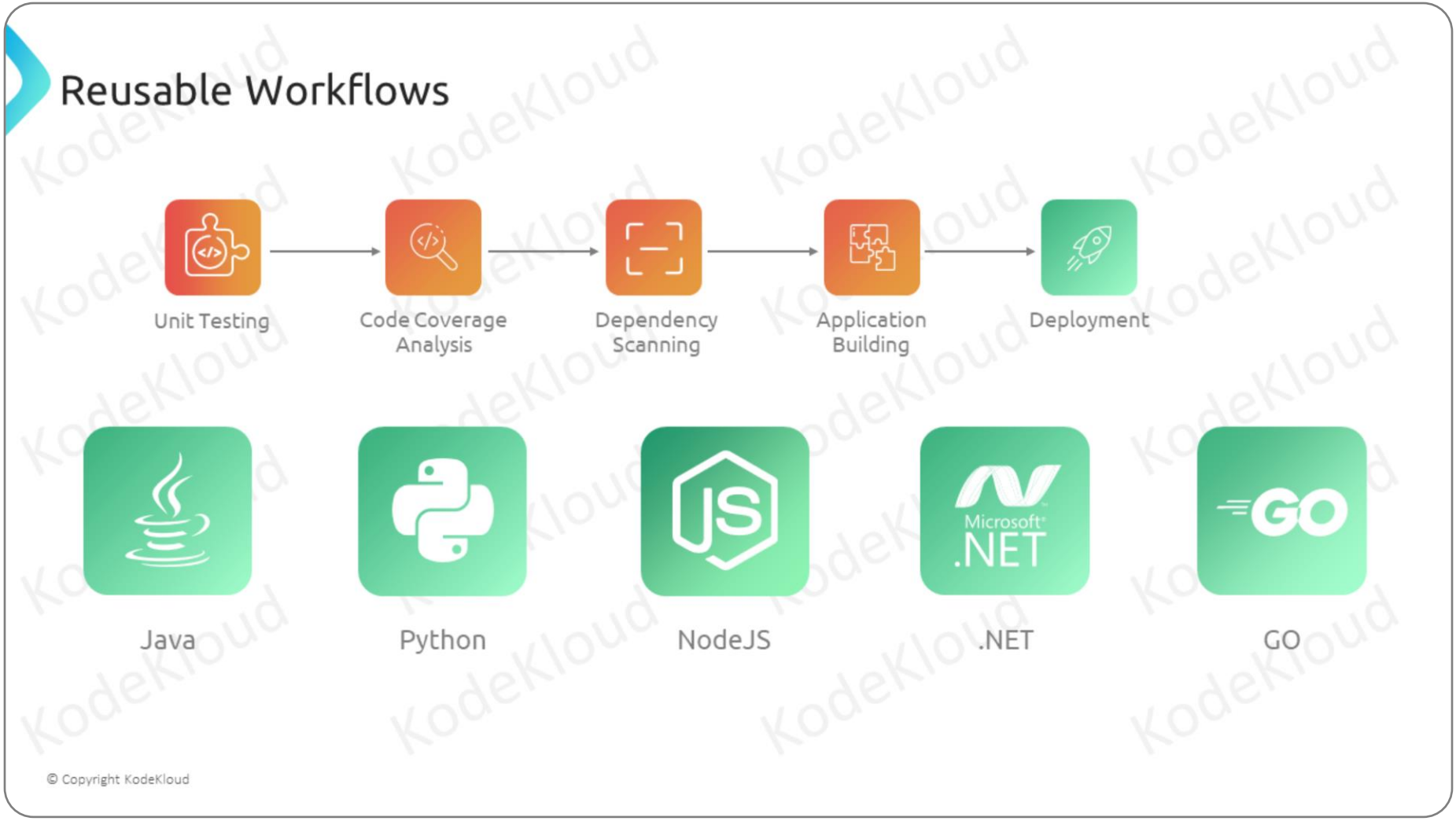


Let's assume the organization or team is responsible for various projects developed in different programming languages like Java, Python, .NET, Go, etc. Each project has a distinct unit testing, code coverage analysis, dependency scanning, and application building. However, when it comes to deployment, all these project workflows share the same deployment jobs and steps as those used in the NodeJS application workflow.

In general, one might consider copying and pasting these jobs from one workflow to another. However, this approach leads to to code duplication, and any necessary changes must be made in multiple workflows, which can be cumbersome.

This is where reusable workflows come into play.

Reusing workflows avoids duplication. Rather than copying and pasting from one workflow to another, you can make workflows reusable. This makes workflows easier to maintain and allows you to create new workflows more quickly by leveraging the work of others, similar to how actions work. eusable workflows also promote best practices by enabling the use of well-designed, tested, and proven-effective workflows.



Let's assume the organization or team is responsible for various projects developed in different programming languages like Java, Python, .NET, Go, etc. Each project has a distinct unit testing, code coverage analysis, dependency scanning, and application building. However, when it comes to deployment, all these project workflows share the same deployment jobs and steps as those used in the NodeJS application workflow.

In general, one might consider copying and pasting these jobs from one workflow to another. However, this approach leads to to code duplication, and any necessary changes must be made in multiple workflows, which can be cumbersome.

This is where reusable workflows come into play.

Reusing workflows avoids duplication. Rather than copying and pasting from one workflow to another, you can make workflows reusable. This makes workflows easier to maintain and allows you to create new workflows more quickly by leveraging the work of others, similar to how actions work. reusable workflows also promote best practices by enabling the use of well-designed, tested, and proven-effective workflows.

Reusable Workflow

Terminal

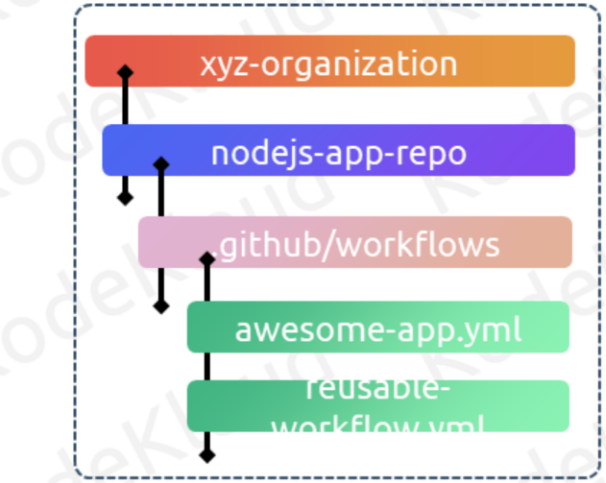
```yaml
name: My Awesome App
on: push
jobs:
  unit-testing:
    .. ..
  code-coverage:
    .. ..
  build:
    .. ..
  dev-deploy:
    needs: [build]
    uses: ./.github/
          workflows/
     reusable-workflow.yml
```

Caller Workflow

Terminal

```yaml
name: Reusable Workflow
on:
  workflow_call:
```

Called Workflow



`./.github/workflows/reusable-workflow.yml`

`xyz-organization/nodejs-app-repo/.github/workflows/reusable-workflow.yml`

Assume a github organization named xyz-organization containing a repository named nodejs-app-repo.

This repo contains a awesome-app.yml workflow file within the .github/workflows directory.

This workflow contains multiple jobs, one of the job is dev-deploy which contains 5 steps.

Using this workflow lets understand how to extract this and use it as a reusable workflow.

We need to create another yml within the same .github/workflows directory of a repository., we will name it reusable-workflow.yml

For a workflow to be reusable, the values for on must include a workflow_call

Whatever job/steps needs to be reused should copied to the reusable workflow.

Referencing/calling a reusable workflow depends on the location of reusable workflows. In this example both workflows are in the same repository and hence the relative path of the reusable workflow can be used.

To use a reusable workflow, you need to use the uses keyword in the workflow file.

For referencing reusable workflow from other public private reposiotries, we need to also include the organization and repository names in the path.

Apart from this a reusable workflows can also contain Inputs, outputs, and secrets to pass data between workflows. he use of inputs, outputs, and secrets can make reusable workflows more flexible. We will discuss more about them in the next demo.

In Github terminology A workflow that uses another workflow is referred to as a "caller" workflow. The reusable workflow is a "called" workflow.

Project Status Meeting - 5

Project Status Meeting - 5

Priority	Task	Assigned	Status	Comments/Issue
0	Understand Requirement	Alice	Completed	
1	Unit Testing	Alice	Completed	
2	Code Coverage	Alice	Completed	
3	Containerization	Alice	Completed	
4	Kubernetes Dev Deployment	Alice	Completed	
5	Dev Integration Testing	Alice	Completed	
6	Manual Approval	Alice	Completed	
7	Kubernetes Prod Deployment	Alice	Completed	
8	Prod Integration Testing	Alice	Completed	
9	Upload Reports to AWS	Alice	In Progress	

KodeKloud

Follow us on https://kodekloud.com/ to learn more about us.