



KodeKloud

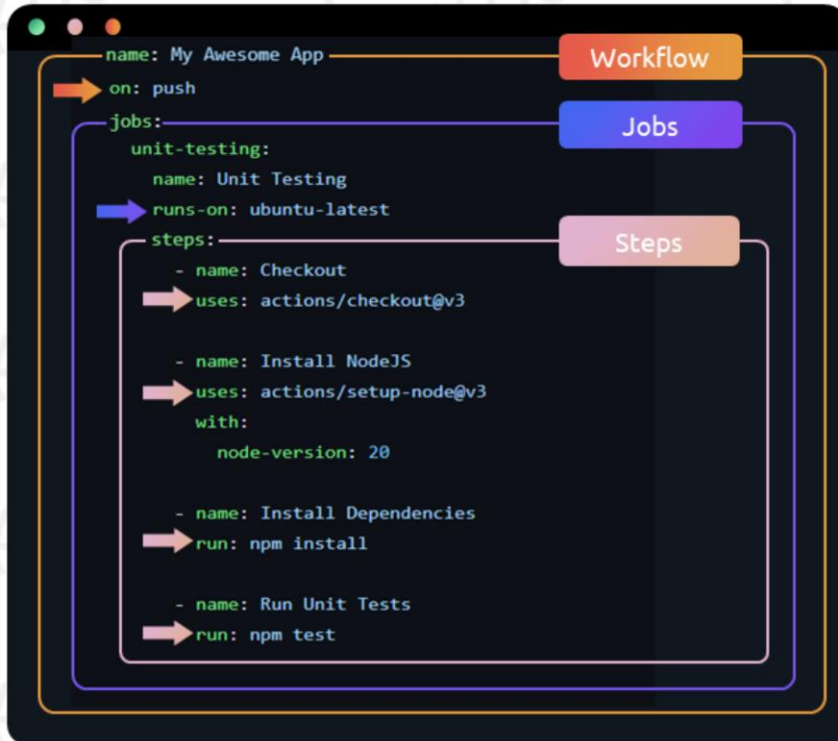
© Copyright KodeKloud

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

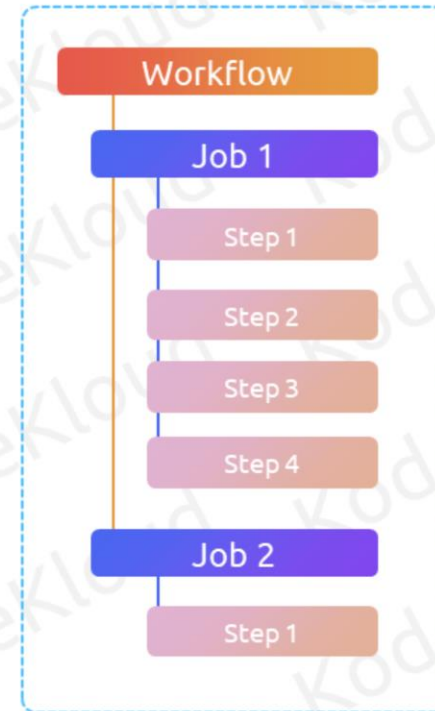


GitHub Actions – Core Components

Core Concepts



© Copyright KodeKloud



In this session we will explore the core components of a GitHub Action.

It consists of 3 key components, workflows, jobs and steps. Let's take an example and understand them,

A workflow is an automated process capable of executing one or more jobs. It is widely used in your software development process, from building and testing code to deploying it to various environments. These workflows are defined using YAML files and are located within your repository. A repository can have multiple workflows, each of which runs in response to specific events occurring in your repository.

To uniquely identify different workflows, each workflow can have an optional name keyword. This name will be visible in the "Actions" tab of the GitHub repository.

In this example, the workflow is triggered for a push event to the repository. We will talk more about triggers and events in a later session.

The next core component is a Job, Jobs are the building blocks of a workflow, and you can have one or more jobs within a single workflow. Each job is associated with a runner it could be a GitHub-hosted or self-hosted runner. We use the runs-on attribute to specify the runner.

If a workflow has multiple jobs, Jobs can run in parallel or sequentially, depending on your configuration.

In this left side example, we have only one job and it will run on an ubuntu-latest runner. We will talk more about runners and runner configurations in a later session.

The next component is Steps, Steps in GitHub workflows are individual tasks or actions that make up a job. There are executed sequentially within a job's runner environment. Steps can include commands, actions, or scripts, allowing you to automate specific actions like building, testing, or deploying code as part of your CI/CD process.

In this example, we have 4 steps, the first step uses an action named checkout of tag v3 to checkout the repository to the runner. Remember

the term actions, I will talk about it at the end of this session.

the second step also uses an action to install nodejs on the runner, which provides node and npm commands in runners PATH..

the third step uses a run keyword to execute shell commands on the runner, in this case it is using npm to install dependencies

and the last step again uses a run keyword to execute npm test command.

All these steps are executed sequentially.

So what are these Actions? –

Actions within GitHub workflows are pre-built, reusable automation components designed for specific tasks. These actions can be created by you or by members of the community, making it easy to share and reuse automation logic across repositories.

Actions also facilitate integration with third-party tools and services, enhancing your project's automation capabilities.

A few examples of official GitHub actions include:

1. Build and push Docker images – Github actions for building and pushing docker images
2. Create AKS cluster - GitHub action for creating AKS cluster
3. Vault Secrets - A Github Action that allows you to consume HashiCorp Vault™ secrets as secure environment variables

We will talk more about actions in a later session.

Now that we understood the core components, let's proceed to create our very first workflow.

GitHub Actions



Build and push
Docker images



Create AKS
Cluster



Consume
Vault Secrets

So what are these Actions? –

Actions within GitHub workflows are pre-built, reusable automation components designed for specific tasks.

These actions can be created by you or by members of the community, making it easy to share and reuse automation logic across repositories.

Actions also facilitate integration with third-party tools and services, enhancing your project's automation capabilities.

A few examples of official GitHub actions include:

1. Build and push Docker images – Github actions is for building and pushing images to multiple registries.
2. Create AKS cluster - GitHub action is for creating AKS cluster
3. Vault Secrets - Github Action allows you to consume HashiCorp Vault™ secrets as secure environment variables

We will talk more about actions in a later session.

Now that we understood the core components, let's proceed to create our very first workflow.



GitHub Action

Actions

GitHub-Verified Actions



Setup Java JDK

By actions Creator verified by GitHub

Set up a specific version of the Java JDK and add the command-line tools to the PATH

☆ 1.2k stars



Authenticate to Google Cloud

By google-github-actions Creator verified by GitHub

Authenticate to Google Cloud from GitHub Actions via Workload Identity Federation or service account keys

☆ 672 stars



Create secret in Kubernetes cluster

By Azure Creator verified by GitHub

Create a generic secret or docker-registry secret in a Kubernetes such as Azure Kubernetes Service (AKS) clusters

☆ 31 stars

Third-Party/ Community Actions



Deploy to GitHub Pages

By Jameslves

This action will handle the deployment process of your project to GitHub Pages

☆ 3.8k stars



OpenCommit — improve commits with AI

By di-sukharev

Replaces lame commit messages with meaningful AI-generated messages when you push to remote

☆ 4.4k stars



ChatGPT CodeReviewer

By anc95

A Code Review Action Powered By ChatGPT

☆ 2.9k stars

Actions within GitHub workflows are pre-built, reusable automation components designed for specific tasks.

Actions also facilitate integration with third-party tools and services, enhancing your project's automation capabilities.

These actions can be created by you or by members of the community, making it easy to share and reuse automation

logic across repositories.

You can locate and explore Actions within the GitHub Marketplace which the primary place to find GitHub Actions. We have hundred's of actions which are created by GitHub and other community members.


Actions bearing the "TICK" badge indicate that GitHub has verified the creator of the Action as a partner organization.

Actions without tick badge are created by community members. While you can utilize these Actions, it is advisable to thoroughly review the Action's source code to ensure it handles your repository's content and secrets as intended. For instance, you should confirm that secrets are not inadvertently shared with unintended hosts or logged.



We will explore more about security considerations in a separate module.

Actions

Marketplace / Search results / Checkout




Checkout

By actions  v3.6.0  4.3k

Checkout a Git repository at a particular version

Installation

Copy and paste the following snippet into your .yaml file.

Version: v3.6.0 

```
- name: Checkout
  uses: actions/checkout@v3.6.0
```

Action

steps:

- uses: actions/checkout@v3.6.0

Tag

steps:

- uses: actions/checkout@main

Branch

steps:

- uses: actions/checkout@a824008085750b8e136effc585c3cd6082bd575f

SHAs

Once you've decided on the Action to use, the process of adding it to a workflow involves exploring the Action's documentation page. This documentation will provide details about the Action's version and the required workflow syntax.

You can define the Action's version in three ways: through tags, branches, or SHAs.

Tags are useful for deciding when to switch between major and minor versions. For example, you can use an Action tagged as "v3.6.0."

Branches - Specifying a branch for the action means it will always run the version currently on that branch. This approach can create problems if an update to the branch includes breaking changes. This example uses an action which targets a branch named @main

SHAs - For more dependable versioning, using the SHA value associated with the Action's version is recommended. SHAs are immutable, making them a more reliable choice compared to tags or branches.



GitHub Action Expressions

Expressions

Context

github

env

vars

job

jobs

steps

runner

secrets

matrix

needs

inputs

strategy

```
steps:  
- name: Testing  
  if: <expression> || <expression>  
  run: ./script.sh
```

if

```
jobs:  
  Deploy:  
    if: github.ref == 'refs/heads/main'  
    steps:
```

if

© Copyright KodeKloud

Lets look into github expressions,

GitHub Actions expressions are used to dynamically configure workflows to execute **JOBS** and **STEPS** based on conditional logic.

Two primary conditions come into play: "if" and "continue-on-error."

In this video we will only focus on the "if" condition which lets you to control whether a particular step or job should proceed based on a defined condition. The if condition is typically used within a job's or step's configuration to determine its execution.

An expression can be any combination of literal values, references to a context, or functions.



KodeKloud

© Copyright KodeKloud

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