Using a matrix for your jobs

In this article

Create a matrix to define variations for each job.

About matrix strategies

A matrix strategy lets you use variables in a single job definition to automatically create multiple job runs that are based on the combinations of the variables. For example, you can use a matrix strategy to test your code in multiple versions of a language or on multiple operating systems.

Using a matrix strategy

Use `jobs.<job_id>.strategy.matrix` to define a matrix of different job configurations. Within your matrix, define one or more variables followed by an array of values. For example, the following matrix has a variable called `version` with the value `[10, 12, 14]` and a variable called `os` with the value `[ubuntu-latest, windows-latest]`:

```
jobs:
  example_matrix:
    strategy:
      matrix:
        version: [10, 12, 14]
        os: [ubuntu-latest, windows-latest]
```

A job will run for each possible combination of the variables. In this example, the workflow will run six jobs, one for each combination of the `os` and `version` variables.

By default, GitHub Enterprise Cloud will maximize the number of jobs run in parallel depending on runner availability. The order of the variables in the matrix determines the order in which the jobs are created. The first variable you define will be the first job that is created in your workflow run. For example, the above matrix will create the jobs in the following order:

- `{version: 10, os: ubuntu-latest}`
- `{version: 10, os: windows-latest}`
- `{version: 12, os: ubuntu-latest}`
- `{version: 12, os: windows-latest}`
- `{version: 14, os: ubuntu-latest}`
- `{version: 14, os: windows-latest}`

A matrix will generate a maximum of 256 jobs per workflow run. This limit applies to both GitHub Enterprise Cloud-hosted and self-hosted runners.

The variables that you define become properties in the `matrix` context, and you can reference the property in other areas of your workflow file. In this example, you can use `matrix.version` and `matrix.os` to access the current value of `version` and `os` that the job is using. For more information, see "Contexts."

Example: Using a single-dimension matrix

You can specify a single variable to create a single-dimension matrix.

For example, the following workflow defines the variable `version` with the values `[10, 12, 14]`. The workflow will run three jobs, one for each value in the variable. Each job will access the `version` value through the `matrix.version` context and pass the value as `node-version` to the `actions/setup-node` action.

```
jobs:
  example_matrix:
    strategy:
      matrix:
        version: [10, 12, 14]
    steps:
      - uses: actions/setup-node@v3
        with:
          node-version: ${{ matrix.version }}
```

Example: Using a multi-dimension matrix

You can specify multiple variables to create a multi-dimensional matrix. A job will run for each possible combination of the variables.

For example, the following workflow specifies two variables:

- Two operating systems specified in the `os` variable
- Three Node.js versions specified in the `version` variable

The workflow will run six jobs, one for each combination of the `os` and `version` variables. Each job will set the `runs-on` value to the current `os` value and will pass the current `version` value to the `actions/setup-node` action.

```
jobs:
  example_matrix:
    strategy:
      matrix:
        os: [ubuntu-22.04, ubuntu-20.04]
        version: [10, 12, 14]
    runs-on: ${{ matrix.os }}
    steps:
      - uses: actions/setup-node@v3
        with:
          node-version: ${{ matrix.version }}
```

Example: Using contexts to create matrices

You can use contexts to create matrices. For more information about contexts, see "Contexts."

For example, the following workflow triggers on the `repository_dispatch` event and uses information from the event payload to build the matrix. When a repository dispatch event is created with a payload like the one below, the matrix `version` variable will have a value of `[12, 14, 16]`. For more information about the `repository_dispatch` trigger, see "Events that trigger workflows."

```
{
  "event_type": "test",
  "client_payload": {
    "versions": [12, 14, 16]
  }
}
```

```
on:
  repository_dispatch:
    types:
      - test

jobs:
  example_matrix:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        version: ${{ github.event.client_payload.versions }}
    steps:
      - uses: actions/setup-node@v3
        with:
          node-version: ${{ matrix.version }}
```

Expanding or adding matrix configurations

Use `jobs.<job_id>.strategy.matrix.include` to expand existing matrix configurations or to add new configurations. The value of `include` is a list of objects.

For each object in the `include` list, the key:value pairs in the object will be added to each of the matrix combinations if none of the key:value pairs overwrite any of the original matrix values. If the object cannot be added to any of the matrix combinations, a new matrix combination will be created instead. Note that the original matrix values will not be overwritten, but added matrix values can be overwritten.

For example, this matrix:

```
strategy:
  matrix:
    fruit: [apple, pear]
    animal: [cat, dog]
    include:
      - color: green
      - color: pink
        animal: cat
      - fruit: apple
        shape: circle
      - fruit: banana
      - fruit: banana
        animal: cat
```

will result in six jobs with the following matrix combinations:

- `{fruit: apple, animal: cat, color: pink, shape: circle}`
- `{fruit: apple, animal: dog, color: green, shape: circle}`
- `{fruit: pear, animal: cat, color: pink}`
- `{fruit: pear, animal: dog, color: green}`
- `{fruit: banana}`
- `{fruit: banana, animal: cat}`

following this logic:

- `{color: green}` is added to all of the original matrix combinations because it can be added without overwriting any part of the original combinations.
- `{color: pink, animal: cat}` adds `color:pink` only to the original matrix combinations that include `animal: cat`. This overwrites the `color: green` that was added by the previous `include` entry.
- `{fruit: apple, shape: circle}` adds `shape: circle` only to the original matrix combinations that include `fruit: apple`.
- `{fruit: banana}` cannot be added to any original matrix combination without overwriting a value, so it is added as an additional matrix combination.
- `{fruit: banana, animal: cat}` cannot be added to any original matrix combination without overwriting a value, so it is added as an additional matrix combination. It does not add to the `{fruit: banana}` matrix combination because that combination was not one of the original matrix combinations.

Example: Expanding configurations

For example, the following workflow will run four jobs, one for each combination of `os` and `node`. When the job for the `os` value of `windows-latest` and `node` value of `16` runs, an additional variable called `npm` with the value of `6` will be included in the job.

```yaml
jobs:
  example_matrix:
    strategy:
      matrix:
        os: [windows-latest, ubuntu-latest]
        node: [14, 16]
        include:
          - os: windows-latest
            node: 16
            npm: 6
    runs-on: ${{ matrix.os }}
    steps:
      - uses: actions/setup-node@v3
        with:
          node-version: ${{ matrix.node }}
      - if: ${{ matrix.npm }}
        run: npm install -g npm@${{ matrix.npm }}
      - run: npm --version
```

Example: Adding configurations

For example, this matrix will run 10 jobs, one for each combination of `os` and `version` in the matrix, plus a job for the `os` value of `windows-latest` and `version` value of `17`.

```yaml
jobs:
  example_matrix:
    strategy:
      matrix:
        os: [macos-latest, windows-latest, ubuntu-latest]
        version: [12, 14, 16]
        include:
          - os: windows-latest
            version: 17
```

If you don't specify any matrix variables, all configurations under `include` will run. For example, the following workflow would run two jobs, one for each `include` entry. This lets you take advantage of the matrix strategy without having a fully populated matrix.

```
jobs:
  includes_only:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        include:
          - site: "production"
            datacenter: "site-a"
          - site: "staging"
            datacenter: "site-b"
```

Excluding matrix configurations

To remove specific configurations defined in the matrix, use `jobs.<job_id>.strategy.matrix.exclude`. An excluded configuration only has to be a partial match for it to be excluded. For example, the following workflow will run nine jobs: one job for each of the 12 configurations, minus the one excluded job that matches `{os: macos-latest, version: 12, environment: production}`, and the two excluded jobs that match `{os: windows-latest, version: 16}`.

```
strategy:
  matrix:
    os: [macos-latest, windows-latest]
    version: [12, 14, 16]
    environment: [staging, production]
    exclude:
      - os: macos-latest
        version: 12
        environment: production
      - os: windows-latest
        version: 16
runs-on: ${{ matrix.os }}
```

Note: All `include` combinations are processed after `exclude`. This allows you to use `include` to add back combinations that were previously excluded.

Handling failures

You can control how job failures are handled with `jobs.<job_id>.strategy.fail-fast` and `jobs.<job_id>.continue-on-error`.

`jobs.<job_id>.strategy.fail-fast` applies to the entire matrix. If `jobs.<job_id>.strategy.fail-fast` is set to `true` or its expression evaluates to `true`, GitHub Enterprise Cloud will cancel all in-progress and queued jobs in the matrix if any job in the matrix fails. This property defaults to `true`.

`jobs.<job_id>.continue-on-error` applies to a single job. If `jobs.<job_id>.continue-on-error` is `true`, other jobs in the matrix will continue running even if the job with `jobs.<job_id>.continue-on-error: true` fails.

You can use `jobs.<job_id>.strategy.fail-fast` and `jobs.<job_id>.continue-on-error` together. For example, the following workflow will start four jobs. For each job, `continue-on-error` is determined by the value of `matrix.experimental`. If any of the jobs with `continue-on-error: false` fail, all jobs that are in progress or queued will be cancelled. If the job with `continue-on-error: true` fails, the other jobs will not be affected.

```
jobs:
  test:
    runs-on: ubuntu-latest
```

```yaml
    continue-on-error: ${{ matrix.experimental }}
    strategy:
      fail-fast: true
      matrix:
        version: [6, 7, 8]
        experimental: [false]
        include:
          - version: 9
            experimental: true
```

Defining the maximum number of concurrent jobs

By default, GitHub Enterprise Cloud will maximize the number of jobs run in parallel depending on runner availability. To set the maximum number of jobs that can run simultaneously when using a `matrix` job strategy, use `jobs.<job_id>.strategy.max-parallel`.

For example, the following workflow will run a maximum of two jobs at a time, even if there are runners available to run all six jobs at once.

```yaml
jobs:
  example_matrix:
    strategy:
      max-parallel: 2
      matrix:
        version: [10, 12, 14]
        os: [ubuntu-latest, windows-latest]
```