#github-actions

Dynamic matrix generation with GitHub Actions

18 Oct 2021 in **Infrastructure**

Using a build matrix with GitHub Actions allows us to run tests across multiple combinations of operating systems, platforms and languages. You can set up huge matrices (a matrix with three parameters, each of which has three values will run 27 jobs!), but most workflows that you see will have a single matrix entry.

In this example, we're going to run our tests on the currently supported versions of Node using the following workflow:

```
on: push
jobs:
  ci:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        version: [12, 14, 16]
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: ${{ matrix.version }}
      - run: npm ci
      - run: npm test
```

The list of supported Node.js versions is accurate today, but what happens in 6 months? How about 12 months? We'd have to come back and update this list each time the list of supported versions changes.

Let's update this workflow to build the list of supported versions dynamically so that it's always up to date.

Create a matrix from an output

In the workflow above we set `matrix.version` manually, but it doesn't have to be hardcoded. We can populate it using the output values a previous job. Let's update the workflow to use an `output` as the `matrix.version` value and add a new job that returns a list of currently supported `node` versions:

```
on: push
jobs:
  build-matrix:
    runs-on: ubuntu-latest
    steps:
      - id: set-matrix
        run: echo '::set-output name=version_matrix::["12","14","1
    outputs:
      version_matrix: ${{ steps.set-matrix.outputs.version_matrix
  ci:
    needs: build-matrix
    runs-on: ubuntu-latest
    strategy:
      matrix:
        version: ${{ fromJson(needs.build-matrix.outputs.version_ma
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: ${{ matrix.version }}
      - run: npm ci
      - run: npm test
```

The list of versions is still hardcoded in the `set-matrix` job, but we're one step closer to dynamically populating the list of active versions.

Fetch supported Node.js versions

The next step is to replace the hardcoded string (`["12","14","16"]`) with a data source that dynamically updates. For this post I'm using endoflife.date to fetch the list of active versions.

The API returns a list of objects that look like the following:

```
{
  "cycle": "16",
  "release": "2021-04-20",
  "lts": false,
  "support": "2022-10-18",
  "eol": "2024-04-30",
  "latest": "16.10.0"
}
```

Out action doesn't need all of that data - all we need is the `cycle` value, and to ensure that the `eol` date is before today. Fortunately the Actions runners have `jq` installed, which we can use to manipulate the data.

Here's the command that we use to return the list of active versions:

```
curl https://endoflife.date/api/nodejs.json | jq -c '[.[] | select
# ["16","14","12"]
```

We need to use a sub-shell to run the command in our workflow, which means wrapping the command in `$()`.

Putting it all together, this is what the workflow looks like:

```
on: push
jobs:
  build-matrix:
    runs-on: ubuntu-latest
    steps:
      - id: set-matrix
        run: echo "::set-output name=version_matrix::$(curl https:,
    outputs:
```

```
      version_matrix: ${{ steps.set-matrix.outputs.version_matrix
  ci:
    needs: build-matrix
    runs-on: ubuntu-latest
    strategy:
      matrix:
        version: ${{ fromJson(needs.build-matrix.outputs.version_ma
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: ${{ matrix.version }}
      - run: npm ci
      - run: npm test
```

When this workflow runs, it fetches the list of active Node.js versions in the first job then triggers one instance of the `ci` job per version returned.

Try adding it to one of your own projects to see it work 🥳

Conclusion

The example shown above is simple, but super-useful! As soon as there's a new version of node available, our code will start being tested against it. No more surprises where the community find out that your code doesn't work before you do.

I've seen a couple of interesting applications of this, but none more interesting than RectorPHP testing their refactoring tool against the top 50 packages in the PHP ecosystem. The top 50 list is populated dynamically to ensure that as new packages are released and picked up, they get early feedback on if their tool is compatible.