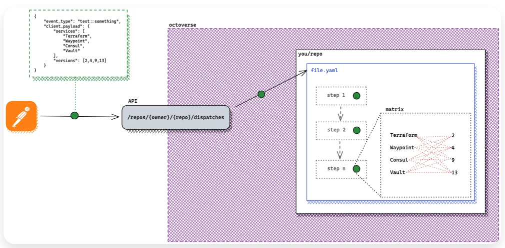
Dynamic Matrices in GitHub Actions

...from JSON payloads that you send! — This was a recent rabbit hole 🐰🕳 that took me 2 days to figure out. I couldn't find a quick and clear answer on Google so I figured I'd write about it.

Kevin Wang / September 19, 2021 9529 views

Tags

github github-actions matrix ci-cd yaml devops repository_dispatch



```
{
    "event_type": "test::something",
    "client_payload": {
        "services": [
            "Terraform",
            "Waypoint",
            "Consul",
            "Vault"
        ],
        "versions": [2,4,9,13]
    }
}
```

matrix diagram

I recently needed to understand GitHub actions' matrix strategy for work.

> ...A matrix allows you to create multiple jobs by performing variable substitution in a single job definition.
>
> For example, you can use a matrix to create jobs for more than one supported version of a programming language, operating system, or tool...

Unfortunately, the docs didn't cover my specific use case and the examples had hardcoded matrix values:

```
os: [macos-latest, windows-latest, ubu
node: [8, 10, 12, 14]
```

I wanted to figure out how to create dynamic matrices... and more specifically, from JSON

So, *how do you* send dynamic matrix values from repository dispatch payloads?

Trial and error

My dev process was pretty *clunky* and was not particularly pleasant either. At the time of development, I hadn't yet tried `nekos/act` , which could've helped speed up parts of my process.

Set up a repo with a workflow

I first created a repo — well, shoved actions into an existing repo — and created a workflow that listened for `repository_dispatch` events, and specific `event_type` s.

```
.github/workflows/dispatch_listener.yaı
```

```
name: Dispatch Listener

on:
  repository_dispatch:
    types:
      - "test run"
      - "foobar"
      - "look-wildcard-here:*"
# more she-yaml-gans to come...
```

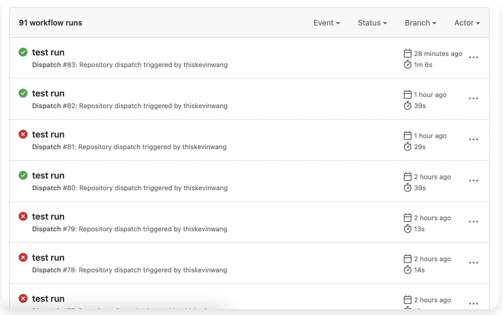
`POST /repos/{owner}/{repo}/dispatches`

The next thing I did was call the GitHub REST API to trigger the repository's workflow manually. I did so via Postman, but in actual implementation, I'd probably be calling it via another workflow with something like HTTP Request Action or from server-side code using Octokit and authenticating as a GitHub application.

- See "Create a rpository dispatch event" for endpoint authentication and details

```
curl --location --request POST 'https:     )i
--header 'Authorization: Bearer {{PERSONAL_
--header 'Content-Type: application/json' \
--data-raw '{
  "event_type": "test run",
  "client_payload": {
    "services": [
        "cat",
        "dog",
        "bird",
        "hamster"
```

I debugged the live workflow runs via the GitHub UI's "Actions" tab, which was not the worst given the simplicity of my workflow, but it was certainly not fun. The lack of IDE linting and type-system made it feel like I was developing in the dark.



91 workflow runs

✓	test run Dispatch #83: Repository dispatch triggered by thiskevinwang	28 minutes ago 1m 6s
✓	test run Dispatch #82: Repository dispatch triggered by thiskevinwang	1 hour ago 39s
✗	test run Dispatch #81: Repository dispatch triggered by thiskevinwang	1 hour ago 29s
✓	test run Dispatch #80: Repository dispatch triggered by thiskevinwang	2 hours ago 39s
✗	test run Dispatch #79: Repository dispatch triggered by thiskevinwang	2 hours ago 13s
✗	test run Dispatch #78: Repository dispatch triggered by thiskevinwang	2 hours ago 14s
✗	test run	2 hours ago

91 workflow runs 😵‍💫

Success

Skipping over a boring streak of unsuccessful workflows, here's the yaml for a workflow that finally *worked*.

Success: This workflow file allows you to:

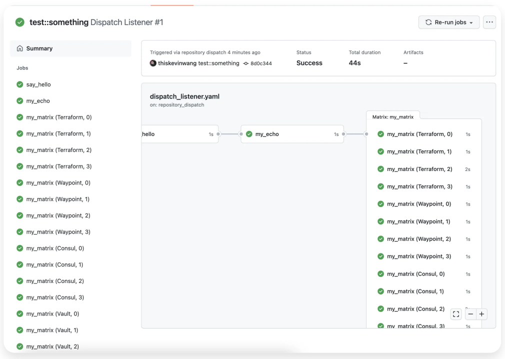
1. Post variable JSON arrays in a `client_payload`
2. Pass them from 1 job run to a following job
3. Trigger a matrix

.github/workflows/dispatch_listener.ya

```yaml
name: Dispatch Listener

on:
  repository_dispatch:
    types:
      - "test run"
      - "test::*"

jobs:
  # ------------------------
  say_hello:
    runs-on: ubuntu-latest
    outputs:
      services: ${{ steps.generate-matrix.c
      versions: ${{ steps.generate-matrix.c
    steps:
      - name: Generate Matrix
        id: generate-matrix
        run: |
```



test::something Dispatch Listener #1

Re-run jobs

Summary

Jobs

say_hello
my_echo
my_matrix (Terraform, 0)
my_matrix (Terraform, 1)
my_matrix (Terraform, 2)
my_matrix (Terraform, 3)
my_matrix (Waypoint, 0)
my_matrix (Waypoint, 1)
my_matrix (Waypoint, 2)
my_matrix (Waypoint, 3)
my_matrix (Consul, 0)
my_matrix (Consul, 1)
my_matrix (Consul, 2)
my_matrix (Consul, 3)
my_matrix (Vault, 0)
my_matrix (Vault, 1)
my_matrix (Vault, 2)

Triggered via repository dispatch 4 minutes ago	Status	Total duration	Artifacts
thiskevinwang test::something -o- 8d0c344	Success	44s	–

dispatch_listener.yaml
on: repository_dispatch

hello 1s — my_echo 1s —

Matrix: my_matrix

my_matrix (Terraform, 0) 1s
my_matrix (Terraform, 1) 1s
my_matrix (Terraform, 2) 2s
my_matrix (Terraform, 3) 1s
my_matrix (Waypoint, 0) 1s
my_matrix (Waypoint, 1) 1s
my_matrix (Waypoint, 2) 1s
my_matrix (Waypoint, 3) 1s
my_matrix (Consul, 0) 1s
my_matrix (Consul, 1) 1s
my_matrix (Consul, 2)
my_matrix (Consul, 3) 1s

16 parallel matrix jobs

Notes

You *could* pass the `client_payload` keys directly to a matrix value

```
strategy:
  matrix:
    service: ${{ github.event.client_payloa
    version: ${{ github.event.client_payloa
```

Or you can set them as outputs from prior jobs, leading up to a job with a matrix.

```
- name: Generate Matrix
  id: generate-matrix
  run: |
    SERVICES='${{ toJSON(github.event.clien
    echo ::set-output name=services::${SERV
    VERSIONS='${{ toJSON(github.event.clien
    echo ::set-output name=versions::${VERS
```

The key here was the single quotes `''` wrapping the `toJSON` function. Without it, the outputs were set to either `Array` or `Object` — the types of the underlying JSON — or `[` or `{` — the first line of pretty-printed JSON.

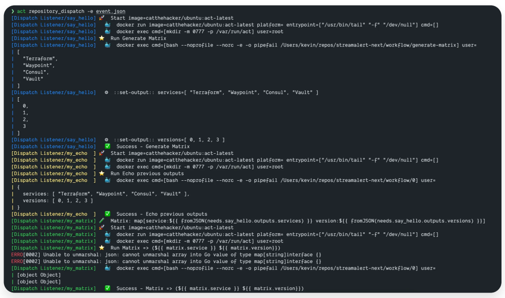
`nektos/act`

After finding my solution, I went back to try act, a tool for running GitHub actions/workflows locally. Given the same JSON payload I was sending via Postman,

event.json

```
{
    }
  }
```

I run `act repository_dispatch -e event.json` and can see the various steps and jobs of my workflow being executed.



```
❯ act repository_dispatch -e event.json
[Dispatch Listener/say_hello] 🚀  Start image=catthehacker/ubuntu:act-latest
[Dispatch Listener/say_hello]   🐳  docker run image=catthehacker/ubuntu:act-latest platform= entrypoint=["/usr/bin/tail" "-f" "/dev/null"] cmd=[]
[Dispatch Listener/say_hello]   🐳  docker exec cmd=[mkdir -m 0777 -p /var/run/act] user=root
[Dispatch Listener/say_hello] ⭐  Run Generate Matrix
[Dispatch Listener/say_hello]   🐳  docker exec cmd=[bash --noprofile --norc -e -o pipefail /Users/kevin/repos/streamalert-next/workflow/generate-matrix] user=
| [
|   "Terraform",
|   "Waypoint",
|   "Consul",
|   "Vault"
| ]
[Dispatch Listener/say_hello]   ⚙  ::set-output:: services=[ "Terraform", "Waypoint", "Consul", "Vault" ]
| [
|   0,
|   1,
|   2,
|   3
| ]
[Dispatch Listener/say_hello]   ⚙  ::set-output:: versions=[ 0, 1, 2, 3 ]
[Dispatch Listener/say_hello]   ✅  Success - Generate Matrix
[Dispatch Listener/my_echo  ] 🚀  Start image=catthehacker/ubuntu:act-latest
[Dispatch Listener/my_echo  ]   🐳  docker run image=catthehacker/ubuntu:act-latest platform= entrypoint=["/usr/bin/tail" "-f" "/dev/null"] cmd=[]
[Dispatch Listener/my_echo  ]   🐳  docker exec cmd=[mkdir -m 0777 -p /var/run/act] user=root
[Dispatch Listener/my_echo  ] ⭐  Run Echo previous outputs
[Dispatch Listener/my_echo  ]   🐳  docker exec cmd=[bash --noprofile --norc -e -o pipefail /Users/kevin/repos/streamalert-next/workflow/0] user=
| {
|   services: [ "Terraform", "Waypoint", "Consul", "Vault" ],
|   versions: [ 0, 1, 2, 3 ]
| }
[Dispatch Listener/my_echo  ]   ✅  Success - Echo previous outputs
[Dispatch Listener/my_matrix] 🧪  Matrix: map[service:${{ fromJSON(needs.say_hello.outputs.services) }} version:${{ fromJSON(needs.say_hello.outputs.versions) }}]
[Dispatch Listener/my_matrix] 🚀  Start image=catthehacker/ubuntu:act-latest
[Dispatch Listener/my_matrix]   🐳  docker run image=catthehacker/ubuntu:act-latest platform= entrypoint=["/usr/bin/tail" "-f" "/dev/null"] cmd=[]
[Dispatch Listener/my_matrix]   🐳  docker exec cmd=[mkdir -m 0777 -p /var/run/act] user=root
[Dispatch Listener/my_matrix] ⭐  Run Matrix => (${{ matrix.service }} ${{ matrix.version}})
ERRO[0002] Unable to unmarshal: json: cannot unmarshal array into Go value of type map[string]interface {}
ERRO[0002] Unable to unmarshal: json: cannot unmarshal array into Go value of type map[string]interface {}
[Dispatch Listener/my_matrix]   🐳  docker exec cmd=[bash --noprofile --norc -e -o pipefail /Users/kevin/repos/streamalert-next/workflow/0] user=
| [object Object]
| [object Object]
[Dispatch Listener/my_matrix]   ✅  Success - Matrix => (${{ matrix.service }} ${{ matrix.version}})
```

act repository_dispatch -e event.json

Warning: At the time of writing this post, act doesn't handle matrices yet.

Other

Some complex workflows that people dropped in some threads that I encountered.

- This
- ...this
- ...and this

github github-actions matrix ci-cd yaml devops repository_dispatch

← NEWER

Tree CLI & Benchmarks

01 November 2021

tree cli golang goreleaser benchmark

OLDER →

Next Stop, HashiCorp

29 August 2021

Comments