GitHub Actions User Guide

Document Level Classification

200

- FAQ
    - How do I allow the global GitHub Actions Runners to access my service?

Use cases

GitHub Actions is a general-purpose CI/CD platform, but lends itself well to several use cases which would make it preferred over our other CI/CD platforms.

- Existing experience. If you've already written GitHub Actions workflows for a public project on github.com, you've already got the experience to do the same here, with almost no learning curve.
- CI coupled tightly to a group's development workflow. You can write a GitHub Actions workflow which lives in-repo and does a sanity check build and lint which can be directly referenced during a PR review, without any external configuration.
- Repo management. Actions have tokens which allow them to manage themselves, so you could write workflows which help with issue/PR triage, for example.
- Human-focused artifacts. Workflows can generate arbitrary artifacts (they're just a zip of a directory). This can be used to give people access to the build's debug symbols. Or if you just have a utility you want to be built and distributed to your teammates, you can point them at the artifact.

However, our GitHub Actions implementation does not try to be a replacement for our other CI/CD systems. There are limitations which would be better served by other systems.

- GitHub Actions tends to be a bit of a walled garden. For example, while failed workflows will send an email to the person who triggered the action, it would be difficult to get that failure to OpsGenie.

- Artifacts are available, but they are mostly for human consumption. For process-oriented artifacts, sending to Artifactory would be preferred over GitHub Actions' own artifacts system.

An architecture philosophy of our GitHub Actions support is not to be prescriptive, and not to pigeonhole other CI/CD systems' methodologies into Workflows here. GitHub's system stands somewhat on its own compared to Jenkins/TeamCity, Codefresh/ArgoCD, etc, and we don't want to introduce an "Epic GitHub" flavor of methodology, instead allowing for cross-domain knowledge and experience to drive usage within Epic.

Workflow jobs are run in a fully ephemeral environment which is not shared with other jobs (i.e. the instance is destroyed immediately after your job completes), so access/data cross-contamination with other jobs is not a concern.

Enabling runners on GitHub organizations

All organizations automatically have access to the `gha-epic` runner group which contains the Epic-wide runners.

Unfortunately, due to a GitHub limitation, Enterprise-wide runners can only be used with orgs, not in users' namespaces, so you are encouraged to use organization repos whenever possible.

Getting started

If you do not yet have experience with GitHub Actions, a good place to start learning is with GitHub's own getting started guide, which provides guided instruction on creating workflows. Here's their example workflow, which can be placed in `.github/workflows/learn-github-actions.yml` within your repository:

```
name: learn-github-actions
on: [push]
```

```
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '14'
      - run: npm install -g bats
      - run: bats -v
```

If you do have existing experience through public github.com workflows, you should feel right at home! There are some differences between our environment and public github.com which are detailed below, but for the most part it's a one-to-one mapping of functionality and knowledge.

Runners

The following runners are available:

Name	Alias	Status	Arch	Notes
`ubuntu-22.04`		Legacy	x86_64	
`ubuntu-22.04-arm64`		Legacy	arm64	
`ubuntu-24.04`	`ubuntu-latest`	Production	x86_64	Current supported AMD64 runner

`ubuntu-24.04-arm64`	`ubuntu-latest-arm64`	Production	arm64	Current supported ARM64 runner

Unless you have a reason to do otherwise, it's recommended to always `runs-on: ubuntu-latest`. The other runners exist only for runner development or pre-production purposes, and are likely to be broken.

Runner releases follow GitHub's upstream releases, but on a 3 month delay. For example, `ubuntu-24.04` was promoted to `ubuntu-latest` on github.com in October 2024, and we plan to do the same in January 2025. Likewise, runner release retirements will following GitHub's lead, but on a 3 month delay.

To repeat, it's recommended to always `runs-on: ubuntu-latest`, the same as recommended for workflows on github.com. If you, say, depend directly on `ubuntu-20.04` because it is known good, you'll mitigate potential issues when `ubuntu-22.04` is eventually promoted to `ubuntu-latest`, but the problem is just delayed until `ubuntu-20.04` is eventually retired. GitHub workflows are a living ecosystem; future runner releases are always available for months before they are promoted, and can be tested ahead of scheduled promotion for important functionality. But for most workflows, 1) 95% of the time a new runner release won't affect existing workflows, and 2) for the other 5%, it'll likely be a simple fix once noticed.

Also, do **not** use the default runner label `self-hosted`, as this will pick any available runner, including runners which could be broken. Unfortunately if you used the "New Workflow" option in the UI to start with, it'll pre-populate `self-hosted`. (We have an open ticket with GitHub support about this.) If you notice a workflow which is using `self-hosted`, please immediately get it changed to `ubuntu-latest`.

ARM64 runners

If you would like to run workflows on ARM64 runners, simply use `runs-on: ubuntu-latest-arm64` . However, there are a few things to note:

- ARM64 is officially not supported by GitHub. They do release their base runner software as ARM64 (which we use), but they do not support it.
- ARM64 language toolchains are not built by GitHub. This means that `setup-go` / `setup-node` / `setup-python` actions are completely broken on ARM64. You will likely need to do some bespoke stuff to work with anything other than Ubuntu-default language versions.
- Non-core third-party actions are likely to be hit or miss as well.
- While you're free to use `ubuntu-latest-arm64` as needed, please continue to use the default `ubuntu-latest` (AMD64) for architecture-agnostic workloads, as we will not be allocating as many concurrent ARM64 runners.

Large runners

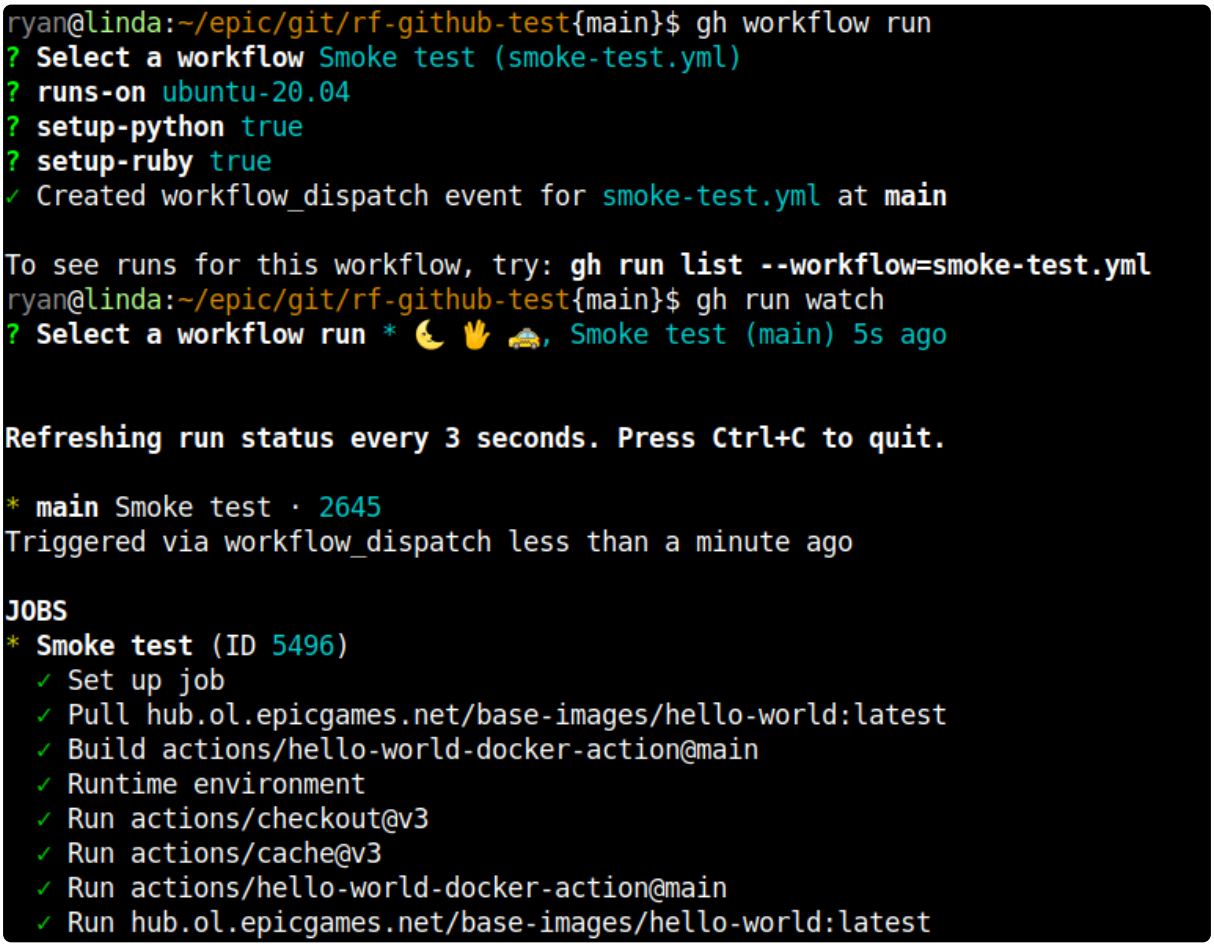
We have a large runner class, `ubuntu-latest-large-x4` and `ubuntu-latest-arm64-large-x4`, with 4 times the resources of regular `ubuntu-latest` runners (currently 32 vCPU, 64 GiB RAM and 160 GiB disk).

There are downsides to using this large runner class, namely the standby pool is not nearly as large as `ubuntu-latest`, so wait times after job creation will usually be several minutes while a backing node is spun up (but should not exceed 5 minutes). Please use your best judgment to determine if this class is right for you; for example, it may actually be quicker overall if you can split your workflow among 4 parallel matrix jobs instead of using an x4 runner.

No authorization is currently required for using this runner class, but we may implement a showback system in the future, so please only use this when `ubuntu-latest` is not feasible.

GitHub CLI

The GitHub CLI (gh) has full support for GitHub Enterprise Server. Once you've linked it to github.ol.epicgames.net, you can do many tasks with it, including Actions workflow management.



Action availability

Core actions

All "core" actions (i.e. actions under https://github.com/actions) are automatically synced daily to https://github.ol.epicgames.net/actions . This means this step will work without modification:

```
jobs:
  ci:
```

```
steps:
- uses: actions/checkout@v3
```

Epic Internal Actions

These are actions developed and shared by teams inside Epic.

- Configure Kubernetes Action - Configures access to a Substrate Kubernetes cluster

```
steps:
- name: configure kubernetes access
  uses: online-web/configure-kubernetes-action@v1
  with:
    vaultRoleId: ${{ secrets.VAULT_ROLE_ID}}
    vaultSecretId: ${{ secrets.VAULT_SECRET_ID }}
    clusterName: bebe-dev-eos-dev-portal
    namespace: team-dev-portal
```

- Install Helmsman Action - Gives useful output from (failing) helm deployments, instead of "timed out waiting for the condition"

```
steps:
- name: Install Helmsman
  uses: online-web/helmsman@v1
```

- Fetch Release Asset Action - Downloads an asset from a github release (or fetches from cache).

```
steps:
- name: Download chart linter
  uses: online-web/fetch-release-asset-action@v1
```

```
  with:
    repo: online-web/chart-linter
    asset-name: chartlinter
    permissions: "+x"
```

Non-core actions

tl;dr a list of non-core synced actions repositories is defined here.

One of the features of GitHub Actions is you can use an action from anyone on github.com. Unfortunately we can't exactly sync all of github.com, but we do keep a sync of desired non-core actions. These are synced to a slightly different namespace, changing the format "owner/repo-name" to "actions/owner_repo-name". For example, if your workflow has:

```
jobs:
  ci:
    steps:
    - uses: ruby/setup-ruby@v1
      with:
        ruby-version: '2.6'
```

We already sync this action, so you will just need to change to:

```
jobs:
  ci:
    steps:
    - uses: actions/ruby_setup-ruby@v1
      with:
        ruby-version: '2.6'
```

If we're not yet syncing an action you would like, please follow the steps below and contact us.

Requesting to sync a non-core action

In order to get a new action added to Enterprise Github, follow these steps.

1. Fork this repo in Github: https://github.ol.epicgames.net/online-operations/github-repo-sync/
2. Make a branch, add the action you want synced (to the right section in `github-repo-sync.yaml`), commit, push to your fork
3. Open PR to the main repo
4. PRs will usually eventually be noticed and reviewed, but for quicker service, ask in #cloud-ops-support-ext or open a Jira ticket.

Self-developed actions

If you would like to develop a custom action, keep in mind that action repos must be public (in this context, company-public, not Internet-public). There are other downsides to "public" repos within github.ol.epicgames.net, so Infrastructure Operations has an alert which normally checks for new public repos. However, we have an exception in place for repos which are in the format "action-${foo}" or "${foo}-action". So when creating a custom action, make it public from the start (GitHub does not allow you to switch between public and private status), and name it one of these two formats.

Custom action repos are required to live in an org's namespace, and are strongly recommended to have a meaningful README.md with description, usage and contact information.

Do NOT manually copy outside-developed actions into new repos to use them. If you have an outside-developed action you would like to use, contact us and we can have it automatically synced into the "actions" org.

Best practices

Trigger duplication

On a typical public github.com development cycle, a user forks a project's repo into their own personal workspace, develops, commits, pushes to their fork, creates a PR against the project, and the project merges the user's fork.

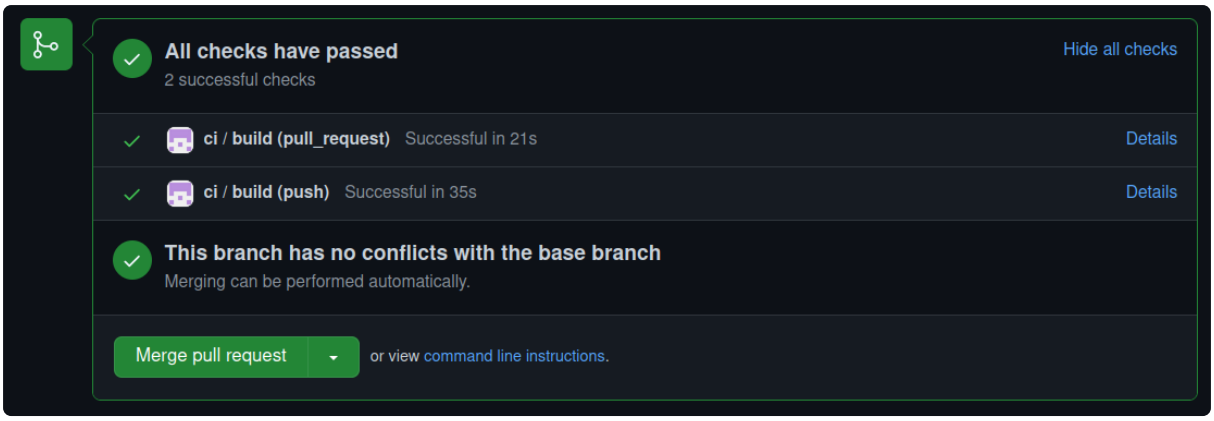
A typical workflow's triggers look like so:

```
on: [push, pull_request]
```

As a result, throughout this process, the workflow is usually run at least 3 times:

1. When the user pushes their changes to their fork (triggering "push" in the context of the forked repo)
2. When the user opens a PR (triggering "pull_request" in the context of the project's repo)
3. When the project merges the PR (triggering "push" in the context of the project's repo)

On the other hand, we tend to follow the same-repo private development model, where a user clones a project's repo, develops, commits, pushes to a different branch in the same repo, and creates a PR. In this case, the same 3 workflows are run, but #1 and #2 at almost the same time and so are a bit inefficient. Both runs show up as checks when determining whether to merge.



So if your project does most of its development in branches within the same repo (as most projects in Epic do), you should be fine to just do:

```
on: [push]
```

GitHub's PR status page is smart enough to show the results of a workflow run as a result of the alt-branch push, so you still get the benefit of a PR-level CI check.

Trigger safety

When using a workflow for linting, unit tests, indicator builds, etc, be as liberal as possible. Testing is good! But be careful about using workflows for CD, or anything which could be destructive, especially for same-repo-different-branch development. Especially keep in mind secrets are available repo-wide, doesn't matter which branch it's being run from.

Workflow triggers do have a way to limit from which branch they can be run. For example:

```
on:
  push:
    branches:
      - 'main'
      - 'releases/**'
```

Reference: [Events that trigger workflows]

Artifact size, retention and access

Artifacts are automatically deleted after 90 days. If you have an artifact which will be of continual use to a set of humans (e.g. built tooling), it's recommended to set up the workflow to build once per month, in addition to on push or other triggers. For example, building on the 17th of the month in addition to push and manual triggers:

```
on:
  schedule:
    - cron: '42 13 17 * *'
  push:
  workflow_dispatch:
```

Artifacts have a limit of 5GB, but it's recommended to keep them under 500MB. In addition, while there is an API endpoint to fetch artifacts, we recommend against it. In general, we would prefer Actions workflows' artifacts to be used for human consumption, not process consumption. If you need larger/more structured artifacts, it's strongly recommended you use the workflow to push to Artifactory.

Vault access

Initial access request / setup

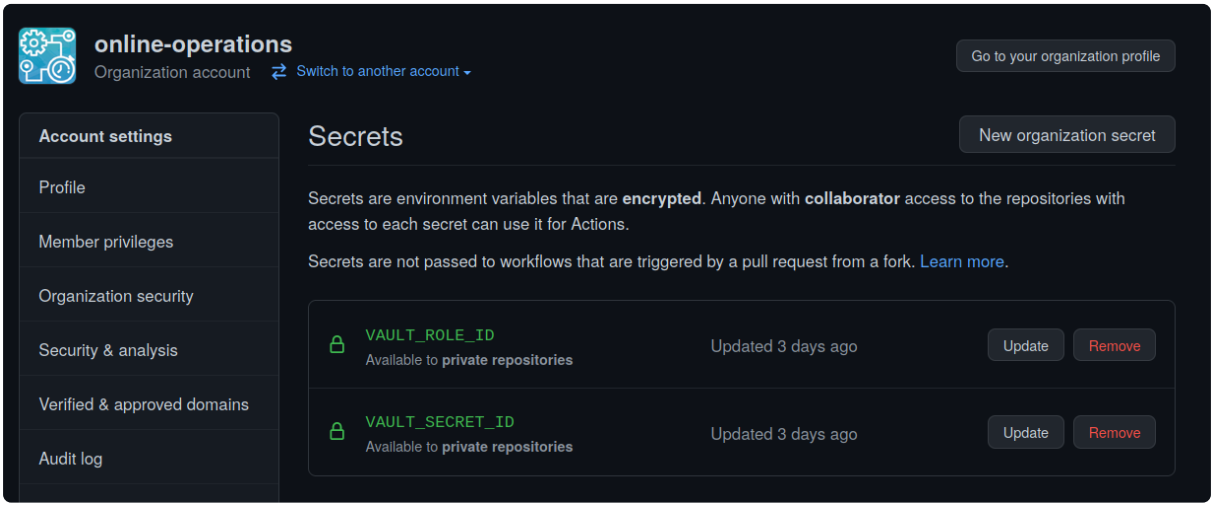
Operations will, upon request through a Cloud Infrastructure Engineering "Support" ticket (Request sent through the #cloud-ops-support-ext channel), generate a GitHub org-specific Vault AppRole ID and secret for a GitHub org owner. This role will have access comparable to a Codefresh org's role, but as there is not a 1:1 logical mapping between Codefresh orgs and GHE orgs, we cannot simply reuse a Codefresh org's role. For comparison, here are the GHE org Vault mappings, while here are the Codefresh org Vault mappings.

While asking Operations team to create the Vault AppRole, make sure to also include the Vault secret path(s) the AppRole should have access to.

Once generated and given to one of the GitHub org's owners, the owner should go to the org Settings, then Secrets (toward the bottom of the left menu in Settings), then add the following secrets at the org level:

- VAULT_ROLE_ID
- VAULT_SECRET_ID

Secrets added at the GitHub org level are available to all repos under the org.



Usage

Once the AppRole secrets are in place, a workflow may utilize the Vault action like so:

```
jobs:
  ci:
    steps:
    - name: Retrieve Vault secrets
      uses: actions/hashicorp_vault-action@v2.4.0
      with:
        url: https://vault.substrate.on.epicgames.com
        method: approle
        roleId: ${{ secrets.VAULT_ROLE_ID }}
        secretId: ${{ secrets.VAULT_SECRET_ID }}
```

```
        secrets: |
          secret/data/brand/project/region/environment/category/name ap
          secret/data/brand/project/region/environment/category/name ur
    - name: Use Vault secrets
      run: echo "${API_TOKEN}" >keys
```

In this example, "api_token" and "url" are retrieved and set as the environment variables API_TOKEN and API_URL, respectively (if a destination is not explicitly listed, a normalized version is created). The action registers these environment variables as masking, so their values should not be printed in run output. For more information about using the action, please see the action's documentation.

Oldprod Vault (vault.ol.epicgames.net)

Similar to the Substrate vault configuration, this vault needs its own appId. Follow the same steps as above.
The secrets can be named as `VAULT_OLDPROD_ROLE_ID` and `VAULT_OLDPROD_SECRET_ID` and used as below example:

```
jobs:
  ci:
    steps:
    - name: Retrieve Vault secrets
      uses: actions/hashicorp_vault-action@v2.4.0
      with:
        url: https://vault.ol.epicgames.net
        method: approle
        roleId: ${{ secrets.VAULT_OLDPROD_ROLE_ID }}
        secretId: ${{ secrets.VAULT_OLDPROD_SECRET_ID }}
        secrets: |
          secret/data/brand/project/region/environment/category/name ap
          secret/data/brand/project/region/environment/category/name ur
    - name: Use Vault secrets
      run: echo "${API_TOKEN}" >keys
```

Artifactory access

Initial access request / setup

Once your org is set up for Vault access (see above) you will automatically have access to Artifactory credentials which give you normal read-only access along with read/write access to legacy repositories; this is comparable access to Codefresh org's role. The two situations which require an access request are:

1. Completely new GitHub organizations (created after the launch of our GitHub Actions project in 2022-05).
2. Your GitHub org requires access to specific non-legacy Artifactory repositories. For reference, here are the GHE org to Artifactory non-legacy mappings.

In either case please contact support to request access. In the case of a new org, this should be batched with Vault access above, which is a prerequisite.

If you require access to an Artifactory repository that belongs to an Artifactory project (the ones not in the common/legacy space) make sure to specify that to the support team.

Usage

Credentials are automatically available via the org's Vault AppRole (see above for general Vault usage).

```
jobs:
  ci:
    steps:
    - name: Retrieve Vault secrets
      uses: actions/hashicorp_vault-action@v2.4.0
      with:
        url: https://vault.substrate.on.epicgames.com
        method: approle
```

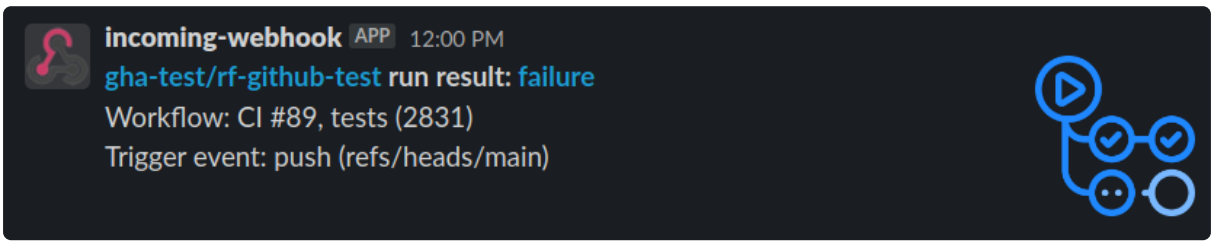
```
        roleId: ${{ secrets.VAULT_ROLE_ID }}
        secretId: ${{ secrets.VAULT_SECRET_ID }}
        secrets: |
          secret/data/ops/ghe-orgs/global/live/runtime/artifactory-${{
          secret/data/ops/ghe-orgs/global/live/runtime/artifactory-${{
          secret/data/ops/ghe-orgs/global/live/runtime/artifactory-${{
    - name: Authenticated Docker
      run: |
        docker login artifacts.ol.epicgames.net \
          --username "${EPIC_BUILD_CREDENTIALS_ARTIFACTS_USERNAME}" \
          --password "${EPIC_BUILD_CREDENTIALS_ARTIFACTS_PASSWORD}"
```

For consistency, the names EPIC_BUILD_CREDENTIALS_ARTIFACTS_USERNAME and EPIC_BUILD_CREDENTIALS_ARTIFACTS_PASSWORD match what is expected in TeamCity and Codefresh.

For general Artifactory usage, please see https://confluence-epicgames.atlassian.net/wiki/spaces/IE/pages/82150453.

Slack notifications

The Slack GitHub action may be used to send notifications to a Slack webhook, for failures (or even for every run).




```
jobs:
  ci:
    steps:
    - name: Slack notification
      uses: actions/slackapi_slack-github-action@v1
      if: failure()
      with:
```

```
        payload: |
          {
            "text": "${{ github.repository }} ${{ github.event_name }}
            "blocks": [
              {
                "type": "section",
                "text": {
                  "type": "mrkdwn",
                  "text": "*<${{ github.server_url }}/${{ github.reposi
                },
                "accessory": {
                  "type": "image",
                  "image_url": "https://avatars.githubusercontent.com/u
                  "alt_text": "GitHub Actions logo"
                }
              }
            ]
          }
      env:
        # Consider storing the webhook URL in Vault instead
        SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK_URL }}
        SLACK_WEBHOOK_TYPE: INCOMING_WEBHOOK
```

Docker buildx configuration

The Docker action **setup-buildx-action** does not inherit the runner's default configuration to use `hub-cache.ol.epicgames.net` to pull from Docker Hub, which can lead to "Too Many Requests" errors from Docker Hub. To fix this:

```
    steps:
      - name: Set up Docker Buildx
        uses: actions/docker_setup-buildx-action@v2
        with:
          config-inline: |
```

```
            [registry."docker.io"]
              mirrors = ["hub-cache.ol.epicgames.net"]
```

Differences from public github.com

Our Actions environment tries to mimic the expected functionality of github.com, and indeed most workflow YAML files will port over from github.com with no modification. However, there are some differences to keep in mind.

No Windows, MacOS or older Ubuntu runners

Windows and MacOS runners are not available, and are not expected to be implemented in the future.

While `ubuntu-18.04` is available on github.com, we have chosen not to replicate it here. Please use `ubuntu-latest` (see above for what this is currently aliased to).

Runner image differences

There are a number of minor differences between our runner image and the runners serving github.com Actions.

Primarily, there is not a 1:1 parity in installed Ubuntu packages. While both our and github.com's runners include a large number of general utilities, github.com's leans farther toward the "kitchen sink" side. Nonetheless, if you feel there is a package which would be of general use, please suggest it! If you're curious, this is the repo used to build our runners.

The execution environment is also different. github.com runners are full Azure VMs. Our runners are in containers in Kubernetes, which have inherent limitations such as not being to mount a loopback ISO. In most cases, these differences should not be noticeable, though.

Resource limits

Runners have resource limits like github.com's runners, but they function a bit different.

	github.com	**Epic GHE**
CPU	4 cores	4 cores
Memory	16 GiB	16 GiB
Disk	14 GB	40 GiB

Note particularly the disk limit. Unfortunately, exceeding 40 GiB will result in the pod being automatically and immediately evicted, with GHE not being able to retrieve the final logs. This will manifest on the UI side as a runner just freezing and eventually disappearing. So please be cognizant of disk usage.

As these are on Kubernetes pods which have a certain level of necessary over-commitment, you may occasionally get failures even when under the resource limits. These will be standard: "no space left on device" for disk, "cannot allocate memory" for memory. CPU over-allocation is not immediately measurable; 50% of one core's performance is the minimum guarantee.

You need to `apt-get update` first

On github.com runners, there is an existing apt repository cache, but it may be out of date (and `sudo apt-get install` operations may or may not fail as a result, so it is always recommended to `sudo apt-get update` first).

On our runners, the repository cache is cleared, and you must do:

```
jobs:
  ci:
    steps:
    - name: Install packages
      run: |
        sudo apt-get update
        sudo apt-get -y install 2ping robotfindskitten
```

No enterprise runners for personal namespaces

Due to limitations of GitHub's support for enterprise runners, it is not possible to offer our runners to repos directly under a user's namespace. If you wish to utilize Actions, it needs to be as part of a sub-branch in-org workflow (which is what most of Epic does anyway).

actions/setup-* differences

If you are using actions/setup-* to target a major series (recommended), it will work as expected and use the latest version of the major series, as these are pre-loaded on the runners. For example, this will (as of this writing) use the latest in the Python 3.10 series without problem:

```
jobs:
  ci:
```

```
    steps:
    - uses: actions/setup-python@v3
      with:
        python-version: '3.10'
```

However, on github.com's runners you have the option of specifying an exact version to use, and it will be downloaded automatically. Due to extremely low public API throttling limits against api.github.com, this will not work out of the box on our runners. If you need a specific minor version, there is a Epic-custom action available to download and extract a specific version for the setup action to later see and set up:

```
jobs:
  ci:
    steps:
    - uses: online-operations/toolcache-downloader-action@v1
      with:
        language: python
        version: '3.10.2'
    - uses: actions/setup-python@v3
      with:
        python-version: '3.10.2'
```

Self-hosted runners

(Terminology note: technically all runners in our GHES are labeled as "self-hosted". Within this context, we mean non-Enterprise-level individual self-hosted runners.)

The philosophy behind GitHub Actions prefers expanding functionality through actions at runtime, rather than having different kinds of base images. To that end, we tried to make our Enterprise-level runners as close as functionally possible to github.com's runners, to avoid the need to consider alternate runners.

That being said, teams are technically allowed to run their own runners for orgs or even individual repos, as long as the runner instance can access github.ol.epicgames.net (agent access is entirely outbound from the runner). However, we'd recommend against it for Linux runners unless you have a specific use case not covered by the general-purpose runners, and are prepared to own the custom runners. The fact that we don't have officially-supported MacOS/Windows runners means that may be a valid rationale for self-hosted runners.

Additionally, be very careful about data security. Our Enterprise-level runners are completely ephemeral, and the instance is completely destroyed after each job. Conversely, manually-set up self-hosted runners are re-used across multiple jobs, potentially across multiple repositories. The runner software does a minimal job of cleaning up the home directory between runs, but many workflows (and even actions themselves) are data litterbugs, and can even be destructive toward the environment (since on github.com, the running instance is known to be throwaway).

Unfortunately, SCM does not have the resources to provide any sort of support or assistance to teams running their own runners.

Future plans

MacOS / Windows runners

No plans currently, as we unfortunately don't have the resources to design and build a suitable bespoke system like we were able to for Linux runners. MacOS in particular is a problem since AWS instances have a 24 hour minimum due to licensing, which would make per-job ephemeral runners impossible for a runner system at the company level.

If you need MacOS / Windows runners for your team and are able to do so, you can always deploy a static instance, install the runner agent manually, and register it with your repo or org. By default the runner software works like Jenkins agents and re-uses the working environment

for each job, so would be fine for per-team jobs which do not need separation.

Platform availability

GitHub Actions is an important part of Epic's CI/CD goals. That being said, while we try to provide the best possible experience, there is currently no explicit guarantee of availability or performance. This section aims to set expectations for the platform.

Runner availability

Jobs run on runners, which are both pre-allocated and able to be scaled up during surges, albeit with a delay of 4-5 minutes while new instances are deployed. Our goal is to have 99% of runs be started within 30 seconds, and 99.9% within 5 minutes.

Job runtime

The runners are deployed on Kubernetes, and while the pods do their best to resist disruption (especially while a job is running), it is possible that a pod may be terminated via disruption at the k8s/EKS level, and this likelihood goes up over time. Users should not rely on jobs which take longer than approximately 3 hours, and keep in mind that GitHub Enterprise Server currently imposes a hard limit of 6 hours per job.

Support

GitHub Actions, as well as GitHub Enterprise Server as a whole, is under the management of IT. For support and requests, please see ITDocs/GitHub.

For general GitHub or Actions-related discussion, learning, tips and tricks, etc, check out #cloud-github-ext. This channel is frequented by Git power users who are usually happy to share advice, but please remember this is

a general discussion channel. For any requests you would like a concrete resolution to, please see ITDocs/GitHub.

FAQ

How do I allow the global GitHub Actions Runners to access my service?

In general most services should be on the Service Network behind an internal load balancer and not public/internet facing. To allow traffic from the GitHub Actions Runners to your service you can manage that by adding a Service Network annotation to your service specifying a prefix list. For example, the following Kubernetes annotation will allow all internal traffic to your service, including the GitHub Actions Runners. The prefix list service-network.all-internal is available in all Substrate accounts.

```
service.epicgames.com/prefix-list-names: service-network.all-internal
```

If your load balancer is public/internet facing it is also advised to manage traffic to the service using Kubernetes annotations. However, in this instance you may need to create your own managed prefix list as using IxM or IxM prefix lists is now considered deprecated. You can use IaC tools such as HashiCorp Terraform to create and manage the prefix list, then use the prefix list name in the annotation. For example, if you were to create a managed prefix list and whitelist the GitHub Actions Public IP's, you could specify that as the prefix list in your ingress annotation.

```
service.epicgames.com/prefix-list-names: pl-XXXXXXXXXX
```

GitHub Actions Public IP's

- 44.195.135.239 (nat-02b110744d589e303)
- 44.199.147.178 (nat-0a4def644744590ef)
- 44.215.228.6 (nat-09d2fcc0ecb1ec3a1)

For further information reference Managing inbound traffic to your application.
