Deploying runner scale sets with Actions Runner Controller

In this article

Learn how to deploy runner scale sets with Actions Runner Controller, and use advanced configuration options to tailor Actions Runner Controller to your needs.

Legal notice

About runner scale sets

Runner scale sets is a group of homogeneous runners that can be assigned jobs from GitHub Actions. The number of active runners owned by a runner scale set can be controlled by auto-scaling runner solutions such as Actions Runner Controller (ARC).

You can use runner groups to manage runner scale sets. Similar to self-hosted runners, you can add runner scale sets to existing runner groups. However, runner scale sets can belong to only one runner group at a time and cannot have labels assigned to them. For more information on runner groups, see "Managing access to self-hosted runners using groups."

To assign jobs to a runner scale set, you must configure your workflow to reference the runner scale set's name. For more information, see "Using Actions Runner Controller runners in a workflow."

Deploying a runner scale set

To deploy a runner scale set, you must have ARC up and running. For more information, see "Quickstart for Actions Runner Controller."

You can deploy runner scale sets with ARC's Helm charts or by deploying the necessary manifests. Using ARC's Helm charts is the preferred method, especially if you do not have prior experience using ARC.

Notes:

- As a security best practice, create your runner pods in a different namespace than the namespace containing your operator pods.
- As a security best practice, create Kubernetes secrets and pass the secret references.

Passing your secrets in plain text via the CLI can pose a security risk.

- We recommend running production workloads in isolation. GitHub Actions workflows are designed to run arbitrary code, and using a shared Kubernetes cluster for production workloads could pose a security risk.

1. To configure your runner scale set, run the following command in your terminal, using values from your ARC configuration.

   When you run the command, keep the following in mind.

   - Update the `INSTALLATION_NAME` value carefully. You will use the installation name as the value of `runs-on` in your workflows.
   - Update the `NAMESPACE` value to the location you want the runner pods to be created.
   - Set the `GITHUB_CONFIG_URL` value to the URL of your repository, organization, or enterprise. This is the entity that the runners will belong to.
   - This example command installs the latest version of the Helm chart. To install a specific version, you can pass the `--version` argument with the version of the chart you want to install. You can find the list of releases in the `actions-runner-controller` repository.

   Bash

   ```
   INSTALLATION_NAME="arc-runner-set"
   NAMESPACE="arc-runners"
   GITHUB_CONFIG_URL="https://github.com/<your_enterprise/org/repo>"
   GITHUB_PAT="<PAT>"
   helm install "${INSTALLATION_NAME}" \
       --namespace "${NAMESPACE}" \
       --create-namespace \
       --set githubConfigUrl="${GITHUB_CONFIG_URL}" \
       --set githubConfigSecret.github_token="${GITHUB_PAT}" \
       oci://ghcr.io/actions/actions-runner-controller-charts/gha-runner-scale-set
   ```

   For additional Helm configuration options, see `values.yaml` in the ARC repository.

2. To check your installation, run the following command in your terminal.

   Bash

   ```
   helm list -A
   ```

   You should see an output similar to the following.

   ```
   NAME            NAMESPACE       REVISION        UPDATED
   STATUS          CHART                                       APP VERSION
   arc             arc-systems     1               2023-04-12
   11:45:59.152090536 +0000 UTC deployed        gha-runner-scale-set-
   controller-0.4.0       0.4.0
   arc-runner-set  arc-systems     1               2023-04-12
   11:46:13.451041354 +0000 UTC deployed        gha-runner-scale-set-0.4.0
   0.4.0
   ```

3. To check the manager pod, run the following command in your terminal.

   Bash

```
kubectl get pods -n arc-systems
```

If the installation was successful, the pods will show the `Running` status.

```
NAME                                                   READY   STATUS
RESTARTS   AGE
arc-gha-runner-scale-set-controller-594cdc976f-m7cjs   1/1     Running   0
64s
arc-runner-set-754b578d-listener                       1/1     Running   0
12s
```

If your installation was not successful, see "Troubleshooting Actions Runner Controller errors" for troubleshooting information.

Using advanced configuration options

ARC offers several advanced configuration options.

Configuring the runner scale set name

> **Note:** Runner scale set names are unique within the runner group they belong to. If you want to deploy multiple runner scale sets with the same name, they must belong to different runner groups.

To configure the runner scale set name, you can define an `INSTALLATION_NAME` or set the value of `runnerScaleSetName` in your copy of the `values.yaml` file.

```
## The name of the runner scale set to create, which defaults to the Helm release
name
runnerScaleSetName: "my-runners"
```

Make sure to pass the `values.yaml` file in your `helm install` command. See the Helm Install documentation for more details.

Choosing runner destinations

Runner scale sets can be deployed at the repository, organization, or enterprise levels.

> **Note:** You can only deploy runner scale sets at the enterprise level when using personal access token (classic) authentication.

To deploy runner scale sets to a specific level, set the value of `githubConfigUrl` in your copy of the `values.yaml` to the URL of your repository, organization, or enterprise.

The following example shows how to configure ARC to add runners to `octo-org/octo-repo`.

```
githubConfigUrl: "https://github.com/octo-ent/octo-org/octo-repo"
```

For additional Helm configuration options, see `values.yaml` in the ARC repository.

Using a GitHub App for authentication

If you are not using enterprise-level runners, you can use GitHub Apps to authenticate with the GitHub API. For more information, see "Authenticating to the GitHub API."

> **Note:** Given the security risk associated with exposing your private key in plain text in a file on disk, we recommend creating a Kubernetes secret and passing the reference instead.

You can either create a Kubernetes secret, or specify values in your `values.yaml` file.

Option 1: Create a Kubernetes secret (recommended)

Once you have created your GitHub App, create a Kubernetes secret and pass the reference to that secret in your copy of the `values.yaml` file.

```
kubectl create secret generic pre-defined-secret \
  --namespace=my_namespace \
  --from-literal=github_app_id=123456 \
  --from-literal=github_app_installation_id=654321 \
  --from-literal=github_app_private_key='-----BEGIN RSA PRIVATE KEY-----********'
```

In your copy of the `values.yaml` pass the secret name as a reference.

```
githubConfigSecret: pre-defined-secret
```

Option 2: Specify values in your `values.yaml` file

Alternatively, you can specify the values of `app_id`, `installation_id` and `private_key` in your copy of the `values.yaml` file.

```
## githubConfigSecret is the Kubernetes secret to use when authenticating with
GitHub API.
## You can choose to use a GitHub App or a personal access token (classic)
githubConfigSecret:
  ## GitHub Apps Configuration
  ## IDs must be strings, use quotes
  github_app_id: "123456"
  github_app_installation_id: "654321"
  github_app_private_key: |
    -----BEGIN RSA PRIVATE KEY-----
    ...
    HkVN9...
    ...
    -----END RSA PRIVATE KEY-----
```

For additional Helm configuration options, see `values.yaml` in the ARC repository.

Managing access with runner groups

You can use runner groups to control which organizations or repositories have access to your runner scale sets. For more information on runner groups, see "Managing access to self-hosted runners using groups."

To add a runner scale set to a runner group, you must already have a runner group created. Then set the `runnerGroup` property in your copy of the `values.yaml` file. The following example adds a runner scale set to the Octo-Group runner group.

```
runnerGroup: "Octo-Group"
```

For additional Helm configuration options, see `values.yaml` in the ARC repository.

Configuring an outbound proxy

To force HTTP traffic for the controller and runners to go through your outbound proxy, set the following properties in your Helm chart.

```
proxy:
  http:
    url: http://proxy.com:1234
    credentialSecretRef: proxy-auth # a Kubernetes secret with `username` and
`password` keys
  https:
    url: http://proxy.com:1234
    credentialSecretRef: proxy-auth # a Kubernetes secret with `username` and
`password` keys
  noProxy:
    - example.com
    - example.org
```

ARC supports using anonymous or authenticated proxies. If you use authenticated proxies, you will need to set the `credentialSecretRef` value to reference a Kubernetes secret. You can create a secret with your proxy credentials with the following command.

Bash

```
  kubectl create secret generic proxy-auth \
    --namespace=my_namespace \
    --from-literal=username=proxyUsername \
    --from-literal=password=proxyPassword \
```

For additional Helm configuration options, see `values.yaml` in the ARC repository.

Setting the maximum and minimum number of runners

The `maxRunners` and `minRunners` properties provide you with a range of options to customize your ARC setup.

> **Note:** ARC does not support scheduled maximum and minimum configurations. You can use a cronjob or any other scheduling solution to update the configuration on a schedule.

Example: Unbounded number of runners

If you comment out both the `maxRunners` and `minRunners` properties, ARC will scale up to the number of jobs assigned to the runner scale set and will scale down to 0 if there aren't any active jobs.

```
## maxRunners is the max number of runners the auto scaling runner set will scale
up to.
# maxRunners: 0

## minRunners is the min number of runners the auto scaling runner set will scale
down to.
# minRunners: 0
```

Example: Minimum number of runners

You can set the `minRunners` property to any number and ARC will make sure there is at least this number of runners active and available to take jobs assigned to the runner scale set at all times.

```
## maxRunners is the max number of runners the auto scaling runner set will scale
up to.
# maxRunners: 0

## minRunners is the min number of runners the auto scaling runner set will scale
down to.
minRunners: 20
```

Example: Set maximum and minimum number of runners

In this configuration, Actions Runner Controller will scale up to a maximum of `30` runners and will scale down to `20` runners when the jobs are complete.

> **Note:** The value of `minRunners` can never exceed that of `maxRunners`, unless `maxRunners` is commented out.

```
## maxRunners is the max number of runners the auto scaling runner set will scale
up to.
maxRunners: 30

## minRunners is the min number of runners the auto scaling runner set will scale
down to.
minRunners: 20
```

Example: Jobs queue draining

In certain scenarios you might want to drain the jobs queue to troubleshoot a problem or to perform maintenance on your cluster. If you set both properties to `0`, Actions Runner Controller will not create new runner pods when new jobs are available and assigned.

```
## maxRunners is the max number of runners the auto scaling runner set will scale
up to.
maxRunners: 0

## minRunners is the min number of runners the auto scaling runner set will scale
down to.
minRunners: 0
```

Custom TLS certificates

> **Note:** If you are using a custom runner image that is not based on the `Debian` distribution, the following instructions will not work.

Some environments require TLS certificates that are signed by a custom certificate authority (CA). Since the custom certificate authority certificates are not bundled with the controller or runner containers, you must inject them into their respective trust stores.

```
githubServerTLS:
  certificateFrom:
    configMapKeyRef:
      name: config-map-name
      key: ca.crt
  runnerMountPath: /usr/local/share/ca-certificates/
```

When you do this, ensure you are using the Privacy Enhanced Mail (PEM) format and that the extension of your certificate is `.crt`. Anything else will be ignored.

The controller executes the following actions.

- Creates a `github-server-tls-cert` volume containing the certificate specified in `certificateFrom`.
- Mounts that volume on path `runnerMountPath/<certificate name>`.
- Sets the `NODE_EXTRA_CA_CERTS` environment variable to that same path.
- Sets the `RUNNER_UPDATE_CA_CERTS` environment variable to `1` (as of version `2.303.0`, this will instruct the runner to reload certificates on the host).

ARC observes values set in the runner pod template and does not overwrite them.

For additional Helm configuration options, see `values.yaml` in the ARC repository.

Using Docker-in-Docker or Kubernetes mode for containers

If you are using container jobs and services or container actions, the `containerMode` value must be set to `dind` or `kubernetes`.

- For more information on container jobs and services, see "Running jobs in a container."
- For more information on container actions, see "Creating a Docker container action."

Using Docker-in-Docker mode

> **Note:** The Docker-in-Docker container requires privileged mode. For more information, see Configure a Security Context for a Pod or Container in the Kubernetes documentation.

Docker-in-Docker mode is a configuration that allows you to run Docker inside a Docker container. In this configuration, for each runner pod-created ARC creates the following containers.

- An `init` container
- A `runner` container
- A `dind` container

To enable Docker-in-Docker mode, set the `containerMode.type` to `dind` as follows.

```yaml
containerMode:
  type: "dind"
```

The `template.spec` will be updated to the following default configuration.

```yaml
template:
  spec:
    initContainers:
    - name: init-dind-externals
      image: ghcr.io/actions/actions-runner:latest
      command: ["cp", "-r", "-v", "/home/runner/externals/.",
"/home/runner/tmpDir/"]
      volumeMounts:
        - name: dind-externals
          mountPath: /home/runner/tmpDir
    containers:
    - name: runner
      image: ghcr.io/actions/actions-runner:latest
      env:
        - name: DOCKER_HOST
          value: tcp://localhost:2376
        - name: DOCKER_TLS_VERIFY
```

```
          value: "1"
        - name: DOCKER_CERT_PATH
          value: /certs/client
      volumeMounts:
        - name: work
          mountPath: /home/runner/_work
        - name: dind-cert
          mountPath: /certs/client
          readOnly: true
    - name: dind
      image: docker:dind
      securityContext:
        privileged: true
      volumeMounts:
        - name: work
          mountPath: /home/runner/_work
        - name: dind-cert
          mountPath: /certs/client
        - name: dind-externals
          mountPath: /home/runner/externals
    volumes:
    - name: work
      emptyDir: {}
    - name: dind-cert
      emptyDir: {}
    - name: dind-externals
      emptyDir: {}
```

You cannot override these automatically injected values. If you want to customize this setup, you must unset `containerMode.type`, then copy this configuration and apply it directly in your copy of the [`values.yaml`](values.yaml) file.

For additional Helm configuration options, see [`values.yaml`](values.yaml) in the ARC repository.

Using Kubernetes mode

In Kubernetes mode, ARC uses runner container hooks to create a new pod in the same namespace to run the service, container job, or action.

Prerequisites

Kubernetes mode relies on persistent volumes to share job details between the runner pod and the container job pod. See the Persistent Volumes Kubernetes documentation for more information.

To use Kubernetes mode, you must do the following.

- Create persistent volumes available for the runner pods to claim.
- Use a solution to automatically provision persistent volumes on demand.

For testing, you can use a solution like OpenEBS.

Configuring Kubernetes mode

To enable Kubernetes mode, set the `containerMode.type` to `kubernetes`.

```
containerMode:
  type: "kubernetes"
  kubernetesModeWorkVolumeClaim:
    accessModes: ["ReadWriteOnce"]
    storageClassName: "dynamic-blob-storage"
    resources:
      requests:
        storage: 1Gi
```

For additional Helm configuration options, see `values.yaml` in the ARC repository.

When Kubernetes mode is enabled, workflows that are not configured with a container job will fail with an error similar to:

```
Jobs without a job container are forbidden on this runner, please add a
'container:' to your job or contact your self-hosted runner administrator.
```

In order to allow jobs without a job container to run, you need to instruct the runner to disable this check. You can do that by setting `ACTIONS_RUNNER_REQUIRE_JOB_CONTAINER` to `false` on your runner container:

```yaml
template:
  spec:
    containers:
    - name: runner
      image: ghcr.io/actions/actions-runner:latest
      command: ["/home/runner/run.sh"]
      env:
        - name: ACTIONS_RUNNER_REQUIRE_JOB_CONTAINER
          value: "false"
```

Using a private container registry

To use a private container registry, you can copy the controller image and runner image to your private container registry. Then configure the links to those images and set the `imagePullPolicy` and `imagePullSecrets` values.

Configuring the controller image

You can update your copy of the `values.yaml` file and set the `image` properties as follows.

```yaml
image:
  repository: "custom-registry.io/gha-runner-scale-set-controller"
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: "0.4.0"

imagePullSecrets:
- name: <registry-secret-name>
```

The listener container inherits the `imagePullPolicy` defined for the controller.

Configuring the runner image

You can update your copy of the `values.yaml` file and set the `template.spec` properties as follows.

```yaml
template:
  spec:
    containers:
    - name: runner
      image: "custom-registry.io/actions-runner:latest"
      imagePullPolicy: Always
      command: ["/home/runner/run.sh"]
```

For additional Helm configuration options, see `values.yaml` in the ARC repository.

Updating the pod specification for the runner pod

You can fully customize the PodSpec of the runner pod and the controller will apply the configuration you specify. The following is an example pod specification.

```
template:
  spec:
    containers:
    - name: runner
      image: ghcr.io/actions/actions-runner:latest
      command: ["/home/runner/run.sh"]
      resources:
        limits:
          cpu: 500m
          memory: 512Mi
      securityContext:
        readOnlyRootFilesystem: true
        allowPrivilegeEscalation: false
        capabilities:
          add:
          - NET_ADMIN
```

For additional Helm configuration options, see `values.yaml` in the ARC repository.

Enabling metrics

Note: Metrics for ARC are available as of version gha-runner-scale-set-0.5.0.

ARC can emit metrics about your runners, your jobs, and time spent on executing your workflows. Metrics can be used to identify congestion, monitor the health of your ARC deployment, visualize usage trends, optimize resource consumption, among many other use cases. Metrics are emitted by the controller-manager and listener pods in Prometheus format. For more information, see Exposition formats in the Prometheus documentation.

To enable metrics for ARC, configure the `metrics` property in the `values.yaml` file of the `gha-runner-scale-set-controller` chart.

The following is an example configuration.

```
metrics:
  controllerManagerAddr: ":8080"
  listenerAddr: ":8080"
  listenerEndpoint: "/metrics"
```

Note: If the `metrics:` object is not provided or is commented out, the following flags will be applied to the controller-manager and listener pods with empty values: `--metrics-addr`, `--listener-metrics-addr`, `--listener-metrics-endpoint`. This will disable metrics for ARC.

Once these properties are configured, your controller-manager and listener pods emit metrics via the listenerEndpoint bound to the ports that you specify in your `values.yaml` file. In the above example, the endpoint is `/metrics` and the port is `:8080`. You can use this endpoint to scrape metrics from your controller-manager and listener pods.

To turn off metrics, update your `values.yaml` file by removing or commenting out the `metrics:` object and its properties.

Available metrics for ARC

The following table shows the metrics emitted by the controller-manager and listener pods.

> **Note:** The metrics that the controller-manager emits pertain to the controller runtime and are not owned by GitHub.

Owner	Metric	Type	Description
controller-manager	pending_ephemeral_runners	gauge	Number of ephemeral runners in a pending state
controller-manager	running_ephemeral_runners	gauge	Number of ephemeral runners in a running state
controller-manager	failed_ephemeral_runners	gauge	Number of ephemeral runners in a failed state
listener	assigned_jobs	gauge	Number of jobs assigned to the runner scale set
listener	running_jobs	gauge	Number of jobs running or queued to run
listener	registered_runners	gauge	Number of runners registered by the runner scale set
listener	busy_runners	gauge	Number of registered runners currently running a job
listener	min_runners	gauge	Minimum number of runners configured for the runner scale set
listener	max_runners	gauge	Maximum number of runners configured for the runner scale set
listener	desired_runners	gauge	Number of runners desired (scale up / down target) by the runner scale set
listener	idle_runners	gauge	Number of registered runners not running a job
listener	started_jobs_total	counter	Total number of jobs started since the listener became ready [1]
listener	completed_jobs_total	counter	Total number of jobs completed since the listener became ready [1]

listener	job_queue_duration_seconds	histogram	Number of seconds spent waiting for workflow jobs to get assigned to the runner scale set after queueing
listener	job_startup_duration_seconds	histogram	Number of seconds spent waiting for workflow job to get started on the runner owned by the runner scale set
listener	job_execution_duration_seconds	histogram	Number of seconds spent executing workflow jobs by the runner scale set

[1]: Listener metrics that have the counter type are reset when the listener pod restarts.

High availability and automatic failover

ARC can be deployed in a high availability (active-active) configuration. If you have two distinct Kubernetes clusters deployed in separate regions, you can deploy ARC in both clusters and configure runner scale sets to use the same `runnerScaleSetName`. In order to do this, each runner scale set must be assigned to a distinct runner group. For example, you can have two runner scale sets each named `arc-runner-set`, as long as one runner scale set belongs to `runner-group-A` and the other runner scale set belongs to `runner-group-B`. For information on assigning runner scale sets to runner groups, see "[Managing access to self-hosted runners using groups]."

If both runner scale sets are online, jobs assigned to them will be distributed arbitrarily (assignment race). You cannot configure the job assignment algorithm. If one of the clusters goes down, the runner scale set in the other cluster will continue to acquire jobs normally without any intervention or configuration change.

Using ARC across organizations

A single installation of Actions Runner Controller allows you to configure one or more runner scale sets. These runner scale sets can be registered to a repository, organization, or enterprise. You can also use runner groups to control the permissions boundaries of these runner scale sets.

As a best practice, create a unique namespace for each organization. You could also create a namespace for each runner group or each runner scale set. You can install as many runner scale sets as needed in each namespace. This will provide you the highest levels of isolation and improve your security. You can use GitHub Apps for authentication and define granular permissions for each runner scale set.

Legal notice

Portions have been adapted from https://github.com/actions/actions-runner-controller/ under the Apache-2.0 license: