Open Horizon GitHub Actions Documentation

Blake Pearson[1] and Ben Courliss[2]

[1]IBM DevOps Intern, Blake.Pearson@ibm.com
[2]IBM DevOps Engineer, bencourliss@ibm.com

Created: August 3, 2023 Last Updated: August 8, 2023

Contents

Introduction **2**

1 Repository: Anax **3**
1.1 Workflow: build-push.yml 3
1.1.1 Overview 3
1.1.2 Triggers 3
1.1.3 Variables and Secrets 3
1.1.4 Conventions 4
1.1.5 Produced Artifacts 4
1.1.6 Dependencies 6
1.1.7 Adding Architectures 7
1.1.8 Possible Future Issues 7
1.2 Workflow: release.yml 8
1.2.1 Overview 8
1.2.2 Triggers 8
1.2.3 Inputs 8
1.2.4 Outputs 8
1.2.5 Variables and Secrets 9
1.2.6 Conventions 9
1.2.7 Dependencies 9

2 Repository: Exchange API **10**
2.1 Workflow: build-push.yml 10
2.1.1 Overview 10
2.1.2 Triggers 10
2.1.3 Variables and Secrets 10
2.1.4 Conventions 10
2.1.5 Produced Artifacts 10
2.1.6 Dependencies 11
2.1.7 Possible Future Issues 11

3 Repository: FDO Support **12**
3.1 Workflow: build-push.yml 12
3.1.1 Overview 12
3.1.2 Triggers 12
3.1.3 Variables and Secrets 12

3.1.4 Conventions 12
3.1.5 Produced Artifacts 12
3.1.6 Dependencies 13
3.1.7 Possible Future Issues 13

4 Repository: Vault Exchange Auth **14**
4.1 Workflow: build-push.yml 14
4.1.1 Overview 14
4.1.2 Triggers 14
4.1.3 Variables and Secrets 14
4.1.4 Conventions 14
4.1.5 Produced Artifacts 14
4.1.6 Dependencies 15
4.1.7 Possible Future Issues 15

5 Repository: Examples **16**
5.1 Workflow: release.yml 16
5.1.1 Overview 16
5.1.2 Triggers 16
5.1.3 Inputs 16
5.1.4 Outputs 16
5.1.5 Variables and Secrets 17
5.1.6 Conventions 17
5.1.7 Dependencies 17

6 Repository: Open Horizon Release **18**
6.1 Workflow: release.yml 18
6.1.1 Overview 18
6.1.2 Triggers 18
6.1.3 Inputs 18
6.1.4 Outputs 19
6.1.5 Variables and Secrets 19
6.1.6 Conventions 19
6.1.7 Dependencies 19

Introduction

Within this documentation, you'll find a thorough guide to the CI/CD GitHub Actions workflows employed across the Open Horizon repositories. It provides a clear outline of the automated processes that ensure each component of Open Horizon functions seamlessly. As you navigate, this reference will offer insights into the mechanics of our integration and deployment routines, streamlining collaboration and comprehension.

1 Repository: Anax

The Anax repository, an integral component of Open Horizon, hosts three vital workflows. These include our continuous integration workflows: E2E-test.yml, facilitating comprehensive end-to-end testing, and build-push.yml, responsible for consistent artifact build-and-push operations. Moreover, our release management is orchestrated through release.yml, ensuring that our releases are both consistent and efficient with no manual interventions.

1.1 Workflow: build-push.yml

1.1.1 Overview

This workflow is triggered upon a successful merge, subsequently checking out the Anax repository and generating all supported artifacts. These artifacts are then pushed to the designated repositories: Open Horizon's Dockerhub registry and GitHub's container registry. These artifacts are later utilized in the release process, ensuring a seamless transition from development to production without manual intervention.

1.1.2 Triggers

This workflow is triggered by the following...

- Push or Merge to the following branches of Anax
  - `master`
  - `v2.30`
  - `v2.29`

1.1.3 Variables and Secrets

The workflow needs access to the following variables and secrets...

- Repository Level [1]
  - Variables
    - `DOCKERHUB_REPO` - The Dockerhub Registry where the images will be pushed to.
    - `RUN_NUMBER_OFFSET` - This number will be added to the action build number to achieve correct versioning.
  - Secrets
    - `DOCKER_TOKEN` - Token with access to push to the `DOCKERHUB_REPO` registry
    - `DOCKER_USER` - Username associated with `DOCKER_TOKEN`
    - `MACPKG_HORIZON_CLI_CRT` - The certificate file used to sign our MacOS packages
    - `MACPKG_HORIZON_CLI_P12_BASE64_ENC` - The p12 binary used to sign our MacOS packages encoded into `BASE64`
    - `MACPKG_HORIZON_CLI_P12_PASSWORD` - The password to the p12 binary provided in the `MACPKG_HORIZON_CLI_P12_BASE64_ENC` secret
    - `MACPKG_KEYCHAIN_PASSWORD` - Value is irrelevant, used to protect our private key on the hosted runner.

[1] To set these variables, follow this path: Repository → Settings → Secrets and Variables → Actions.

1.1.4 Conventions

This workflow will follow the following conventions...

- Naming Conventions
  - Docker Images
    - Agbot: `<architecture>_agbot`
    - Anax: `<architecture>_anax`
    - K8s: `<architecture>_anax_k8s`
    - K8s Cronjob: `<architecture>_auto-upgrade-cronjob_k8s`
    - Cloud Sync Service: `<architecture>_cloud-sync-service`
    - Edge Sync Service: `<architecture>_edge-sync-service`
    - Debian Packages: `<architecture>_anax_debian`
    - RPM Packages: `<architecture>_anax_rpm`
    - MacOS Packages: `<architecture>_anax_macpkg`
  - GitHub Actions Artifacts
    - Anax Agent Files: `anax-agent-files-v<VERSION>`
    - Docker Images: `anax-<platform>-<DOCKER IMAGE NAME>-image-v<VERSION>`
    - Debian Packages: `anax-<platform>-<architecture>-deb-package-v<VERSION>`
    - RPM Packages: `anax-<platform>-<architecture>-rpm-package-v<VERSION>`
    - MacOS Packages: `anax-<platform>-<architecture>-mac-package-v<VERSION>`
- Docker Image Tags
  - Push `testing` if `master` branch triggered the workflow
  - Push `testing_<branch>` if non-master branch triggered the workflow
- Creating the Docker Images that hold the Deb/RPM/Mac Packages
  - Rule 1: Dockerfile should be `FROM alpine:latest`
  - Rule 2: Dockerfile should have `ADD ./<debs/rpms/macpkg>.tar.gz`

1.1.5 Produced Artifacts

This workflow produces the following artifacts...

- Docker Images
  - `amd64_agbot`
  - `amd64_anax`
  - `amd64_anax_k8s`
  - `amd64_auto-upgrade-cronjob_k8s`
  - `amd64_cloud-sync-service`
  - `amd64_edge-sync-service`
  - `amd64_anax_debian`
  - `amd64_anax_rpm`
  - `amd64_anax_macpkg`

  - `arm64_anax`

- `arm64_anax_k8s`
- `arm64_auto-upgrade-cronjob_k8s`
- `arm64_edge-sync-service`
- `arm64_anax_debian`
- `arm64_anax_macpkg`

- `ppc64el_anax`
- `ppc64el_anax_k8s`
- `ppc64el_auto-upgrade-cronjob_k8s`
- `ppc64el_edge-sync-service`
- `ppc64el_anax_debian`
- `ppc64el_anax_rpm`

- `armhf_anax_debian`

- `s390x_anax`
- `s390x_anax_k8s`
- `s390x_auto-upgrade-cronjob_k8s`
- `s390x_edge-sync-service`
- `s390x_anax_debian`
- `s390x_anax_rpm`

* GitHub Actions Artifacts (expires 90 days after upload)
  - `anax-agent-files-v<VERSION>`

  - `anax-linux-amd64_agbot-image-v<VERSION>`
  - `anax-linux-amd64_anax-image-v<VERSION>`
  - `anax-linux-amd64_anax_k8s-image-v<VERSION>`
  - `anax-linux-amd64_auto-upgrade-cronjob_k8s-image-v<VERSION>`
  - `anax-linux-amd64_cloud-sync-service-v<VERSION>`
  - `anax-linux-amd64_edge-sync-service-v<VERSION>`
  - `anax-linux-amd64-deb-package-v<VERSION>`
  - `anax-linux-amd64-rpm-package-v<VERSION>`
  - `anax-mac-amd64-mac-package-v<VERSION>`

  - `anax-linux-arm64_anax-image-v<VERSION>`
  - `anax-linux-arm64_anax_k8s-image-v<VERSION>`
  - `anax-linux-arm64_auto-upgrade-cronjob_k8s-image-v<VERSION>`
  - `anax-linux-arm64_edge-sync-service-v<VERSION>`
  - `anax-linux-arm64-deb-package-v<VERSION>`

  - `anax-linux-armhf-deb-package-v<VERSION>`

  - `anax-linux-ppc64el_anax-image-v<VERSION>`
  - `anax-linux-ppc64el_anax_k8s-image-v<VERSION>`
  - `anax-linux-ppc64el_auto-upgrade-cronjob_k8s-image-v<VERSION>`
  - `anax-linux-ppc64el_edge-sync-service-v<VERSION>`
  - `anax-linux-ppc64el-deb-package-v<VERSION>`
  - `anax-linux-ppc64el-rpm-package-v<VERSION>`

  - `anax-linux-s390x_anax-image-v<VERSION>`
  - `anax-linux-s390x_anax_k8s-image-v<VERSION>`
  - `anax-linux-s390x_auto-upgrade-cronjob_k8s-image-v<VERSION>`
  - `anax-linux-s390x_edge-sync-service-v<VERSION>`
  - `anax-linux-s390x-deb-package-v<VERSION>`
  - `anax-linux-s390x-rpm-package-v<VERSION>`

1.1.6 Dependencies

This workflow has the following dependencies...

- Internal Dependencies [2]
  - `configure_versions_script.sh`
  - `docker_build_script.sh`
  - `docker_push_script.sh`
  - `docker_save_script.sh`
  - `package_push.sh`
- External Dependencies
  - External Actions
    - `actions/checkout@v3`
      Author: GitHub
      Link: https://github.com/actions/checkout
      Use: To checkout our repository into the hosted runner.
    - `actions/upload-artifact@v3`
      Author: GitHub
      Link: https://github.com/actions/upload-artifact
      Use: To upload artifacts to GitHub actions artifacts
    - `docker/setup-qemu-action@v2`
      Author: Docker
      Link: https://github.com/docker/setup-qemu-action
      Use: Setup QEMU for Docker image building cross architecture.
    - `docker/setup-buildx-action@v2`
      Author: Docker
      Link: https://github.com/docker/setup-buildx-action
      Use: Setup BuildX for Docker image building cross architecture.

[2] Typically found in: .github/scripts

  * `docker/login-action@v2`
    Author: Docker
    Link: https://github.com/docker/login-action
    Use: Login to Dockerhub and GitHub Container Registry for image push. (Can be replaced with a run docker login command)
  * `actions/setup-go@v3`
    Author: GitHub
    Link: https://github.com/actions/setup-go
    Use: Set up GoLang within our runner environment
  * `uraimo/run-on-arch-action@v2`
    Author: Uraimo and Contributors
    Link: https://github.com/uraimo/run-on-arch-action
    Use: Set up Docker containers to execute build commands on specified architectures allowing us to build anax packages on other architectures when both GitHub doesn't support that architecture as a runner and the packaging tool doesn't support cross-architecture.
  * `actions/download-artifact@v3`
    Author: GitHub
    Link: https://github.com/actions/upload-artifact
    Use: Get artifacts that were uploaded previously.

1.1.7 Adding Architectures

Adding architectures to be built should be as simple as adding them to the build matrix and conditional statements for each step depending on what artifacts you want to be built. The scripts listed in internal dependencies will also need to be updated to inclde the new architecture. If you want a RPM package you'll likely create a new step using the run-on-arch action.

1.1.8 Possible Future Issues

- Install Latest Docker Version
  `- name: Install Latest Docker Version`
  - Previous bugs have popped up from this, seems to be when a major change is made to the latest Docker version or the GitHub hosted runners change pre-installed software relating to docker/moby.
- Configure MacOS Certificates
  `- name: Configure Certificates`
  - OpenSSL and the way MacOS handles certificates/signing varies largely between versions of MacOS. This step uses a work around to mark the certificate as trusted with no password/cli prompts. It does so by creating a new keychain and unlocking said keychain so adding a trusted cert doesn't prompt for a password.

1.2 Workflow: release.yml

1.2.1 Overview

Typically this workflow is triggered by the overarching Open Horizon release manager and should only be manually triggered in very specific use cases. This workflow activates in response to a workflow dispatch initiated via the Actions interface or GitHub API. Utilizing specified image and package version inputs, it retrieves the associated artifacts from a previous build-push.yml workflow run. Subsequently, it tags the Dockerhub images with both the 'latest' and the specified version tags. Finally, a GitHub release page is generated.

1.2.2 Triggers

This workflow is triggered by the following...

- Workflow Dispatch
  - `via Actions Interface` [3]
  - `via GitHub Rest API` [4]

1.2.3 Inputs

This workflow has the following inputs...

- `AGBOT_VERSION` - String - The version of the Agbot image and packages.
- `ANAX_VERSION` - String - The version of the Anax images.
- `ANAX_K8S_VERSION` - String - The version of the Anax K8s images.
- `ANAX_CSS_VERSION` - String - The version of the Cloud Sync Service images.
- `ANAX_ESS_VERSION` - String - The version of the Edge Sync Service images.
- `IS_LATEST` - Boolean - Decides if the 'latest' tag should be pushed to Dockerhub

1.2.4 Outputs

This workflow creates a GitHub Release with the following configuration.

- Title: `v<version> Packages`
- Tag: `v<version>`
- Assets:
  - `agent-install.sh`
  - `horizon-agent-edge-cluster-files.tar.gz`
  - `horizon-agent-linux-deb-amd64.tar.gz`
  - `horizon-agent-linux-deb-arm64.tar.gz`
  - `horizon-agent-linux-deb-armhf.tar.gz`
  - `horizon-agent-linux-deb-ppc64el.tar.gz`
  - `horizon-agent-linux-deb-s390x.tar.gz`
  - `horizon-agent-linux-rpm-ppc64le.tar.gz`

[3] https://github.com/open-horizon/anax/actions/workflows/release.yml
[4] https://docs.github.com/en/rest/actions/workflows?apiVersion=2022-11-28#create-a-workflow-dispatch-event

  - `horizon-agent-linux-rpm-s390x.tar.gz`
  - `horizon-agent-linux-rpm-x86_64.tar.gz`
  - `horizon-agent-macos-pkg-arm64.tar.gz`
  - `horizon-agent-macos-pkg-x86_64.tar.gz`
  - `Source code`

1.2.5 Variables and Secrets

The workflow needs access to the following variables and secrets...

- Repository Level [5]
  - Variables
    - `DOCKERHUB_REPO` - The Dockerhub Registry where the images will be pushed to.
  - Secrets
    - `DOCKER_TOKEN` - Token with access to push to the `DOCKERHUB_REPO` registry
    - `DOCKER_USER` - Username associated with `DOCKER_TOKEN`

1.2.6 Conventions

This workflow will follow the following conventions...

- Naming Conventions
  - Docker Images
    - Agbot: `<architecture>_agbot`
    - Anax: `<architecture>_anax`
    - K8s: `<architecture>_anax_k8s`
    - K8s Cronjob: `<architecture>_auto-upgrade-cronjob_k8s`
    - Cloud Sync Service: `<architecture>_cloud-sync-service`
    - Edge Sync Service: `<architecture>_edge-sync-service`
    - Debian Packages: `<architecture>_anax_debian`
    - RPM Packages: `<architecture>_anax_rpm`
    - MacOS Packages: `<architecture>_anax_macpkg`
  - GitHub Actions Artifacts
    - Anax Agent Files: `anax-agent-files-v<VERSION>`
    - Docker Images: `anax-<platform>-<DOCKER IMAGE NAME>-image-v<VERSION>`
    - Debian Packages: `anax-<platform>-<architecture>-deb-package-v<VERSION>`
    - RPM Packages: `anax-<platform>-<architecture>-rpm-package-v<VERSION>`
    - MacOS Packages: `anax-<platform>-<architecture>-mac-package-v<VERSION>`
- Release Creation
  - Rule: Release commit hash will be taken from the commit hash found in `amd64_agbot` image, which is the commit that triggered the creation of the agbot image.

1.2.7 Dependencies

This workflow has no notable dependencies.

[5]To set these variables, follow this path: Repository → Settings → Secrets and Variables → Actions.

2 Repository: Exchange API

2.1 Workflow: build-push.yml

2.1.1 Overview

This workflow is triggered upon a successful merge, subsequently checking out the Exchange API repository and generating the amd64 docker image. This artifact is then pushed to the designated repositories: Open Horizon's Dockerhub registry and GitHub's container registry.

2.1.2 Triggers

This workflow is triggered by the following...

- Push or Merge to the following branches of Exchange API...
  - `master`
  - `v2.87`
  - `v2.110`

2.1.3 Variables and Secrets

The workflow needs access to the following variables and secrets...

- Repository Level [6]
  - Variables
    - `DOCKERHUB_REPO` - The Dockerhub Registry where the images will be pushed to.
  - Secrets
    - `DOCKER_TOKEN` - Token with access to push to the `DOCKERHUB_REPO` registry
    - `DOCKER_USER` - Username associated with `DOCKER_TOKEN`

2.1.4 Conventions

This workflow will follow the following conventions...

- Naming Conventions
  - Docker Images
    - Exchange Image: `amd64_exchange-api`
- Docker Image Tags
  - Push `testing` if `master` branch triggered the workflow

2.1.5 Produced Artifacts

This workflow produces the following artifacts...

- Docker Images
  - `amd64_exchange-api`

[6] To set these variables, follow this path: Repository → Settings → Secrets and Variables → Actions.

2.1.6 Dependencies

This workflow has the following dependencies...

- External Dependencies
  - External Actions
    - `actions/checkout@v3`
      Author: GitHub
      Link: https://github.com/actions/checkout
      Use: To checkout our repository into the hosted runner.
    - `docker/login-action@v2`
      Author: Docker
      Link: https://github.com/docker/login-action
      Use: Login to Dockerhub and GitHub Container Registry for image push. (Can be replaced with a run docker login command)
    - `coursier/setup-action@v1`
      Author: Coursier
      Link: https://github.com/coursier/setup-action
      Use: A GitHub Action to install Coursier and use it to install Java and Scala CLI tools.

2.1.7 Possible Future Issues

- Install Latest Docker Version
  `- name: Install Latest Docker Version`
  - Previous bugs have popped up from this, seems to be when a major change is made to the latest Docker version or the GitHub hosted runners change pre-installed software relating to docker/moby.

3 Repository: FDO Support

3.1 Workflow: build-push.yml

3.1.1 Overview

This workflow is triggered upon a successful merge, subsequently checking out the FDO-Support repository and generating all supported artifacts. These artifacts are then pushed to the designated repositories: Open Horizon's Dockerhub registry and GitHub's container registry.

3.1.2 Triggers

This workflow is triggered by the following...

- Push or Merge to the following branches of FDO-Support
  - `main`

3.1.3 Variables and Secrets

The workflow needs access to the following variables and secrets...

- Repository Level [7]
  - Variables
    - `DOCKERHUB_REPO` - The Dockerhub Registry where the images will be pushed to.
    - `RUN_NUMBER_OFFSET` - This number will be added to the action build number to achieve correct versioning.
  - Secrets
    - `DOCKER_TOKEN` - Token with access to push to the `DOCKERHUB_REPO` registry
    - `DOCKER_USER` - Username associated with `DOCKER_TOKEN`

3.1.4 Conventions

This workflow will follow the following conventions...

- Naming Conventions
  - Docker Images
    - FDO Image: `fdo-owner-services`
- Docker Image Tags
  - Push `testing` if `master` branch triggered the workflow

3.1.5 Produced Artifacts

This workflow produces the following artifacts...

- Docker Images
  - `fdo-owner-services`

[7] To set these variables, follow this path: Repository → Settings → Secrets and Variables → Actions.

3.1.6 Dependencies

This workflow has the following dependencies...

- External Dependencies
  - External Actions
    - `actions/checkout@v3`
      Author: GitHub
      Link: https://github.com/actions/checkout
      Use: To checkout our repository into the hosted runner.
    - `docker/login-action@v2`
      Author: Docker
      Link: https://github.com/docker/login-action
      Use: Login to Dockerhub and GitHub Container Registry for image push. (Can be replaced with a run docker login command)
    - `actions/setup-go@v2`
      Author: GitHub
      Link: https://github.com/actions/setup-go
      Use: Set up GoLang within our runner environment

3.1.7 Possible Future Issues

- Install Latest Docker Version
  `- name: Install Latest Docker Version`
  - Previous bugs have popped up from this, seems to be when a major change is made to the latest Docker version or the GitHub hosted runners change pre-installed software relating to docker/moby.

4 Repository: Vault Exchange Auth

4.1 Workflow: build-push.yml

4.1.1 Overview

This workflow is triggered upon a successful merge, subsequently checking out the vault-exchange-auth repository and generating all supported artifacts. These artifacts are then pushed to the designated repositories: Open Horizon's Dockerhub registry and GitHub's container registry.

4.1.2 Triggers

This workflow is triggered by the following...

- Push or Merge to the following branches of Vault
  - `master`

4.1.3 Variables and Secrets

The workflow needs access to the following variables and secrets...

- Repository Level [8]
  - Variables
    - `DOCKERHUB_REPO` - The Dockerhub Registry where the images will be pushed to.
    - `RUN_NUMBER_OFFSET` - This number will be added to the action build number to achieve correct versioning.
  - Secrets
    - `DOCKER_TOKEN` - Token with access to push to the `DOCKERHUB_REPO` registry
    - `DOCKER_USER` - Username associated with `DOCKER_TOKEN`

4.1.4 Conventions

This workflow will follow the following conventions...

- Naming Conventions
  - Docker Images
    - Vault Image: `amd64_vault`
- Docker Image Tags
  - Push `testing` if `master` branch triggered the workflow

4.1.5 Produced Artifacts

This workflow produces the following artifacts...

- Docker Images
  - `amd64_vault`

[8] To set these variables, follow this path: Repository → Settings → Secrets and Variables → Actions.

4.1.6 Dependencies

This workflow has the following dependencies...

- External Dependencies
  - External Actions
    - `actions/checkout@v3`
      Author: GitHub
      Link: https://github.com/actions/checkout
      Use: To checkout our repository into the hosted runner.
    - `docker/login-action@v2`
      Author: Docker
      Link: https://github.com/docker/login-action
      Use: Login to Dockerhub and GitHub Container Registry for image push. (Can be replaced with a run docker login command)
    - `actions/setup-go@v2`
      Author: GitHub
      Link: https://github.com/actions/setup-go
      Use: Set up GoLang within our runner environment

4.1.7 Possible Future Issues

- Install Latest Docker Version
  `- name: Install Latest Docker Version`
  - Previous bugs have popped up from this, seems to be when a major change is made to the latest Docker version or the GitHub hosted runners change pre-installed software relating to docker/moby.

5 Repository: Examples

5.1 Workflow: release.yml

5.1.1 Overview

Typically this workflow is triggered by the overarching Open Horizon release manager and should only be manually triggered in very specific use cases. This workflow activates in response to a workflow dispatch initiated via the Actions interface or GitHub API. Utilizing the version JSON inputted, it creates the tested versions file and subsequently generates the examples release with said file.

5.1.2 Triggers

This workflow is triggered by the following...

- Workflow Dispatch
  - `via Actions Interface`[9]
  - `via GitHub Rest API`[10]

5.1.3 Inputs

This workflow has the following inputs...

- `versionFileJSON` - String - The file containing all the versions (example below)

```
{
    "amd64_agbot": "2.31.0-1498",
    "amd64_anax": "2.31.0-1498",
    "amd64_anax_k8s": "2.31.0-1498",
    "amd64_cloud-sync-service": "1.10.1-1498",
    "amd64_edge-sync-service": "1.10.1-1498",
    "amd64_exchange-api": "2.117.0-1163",
    "amd64_vault": "1.1.2-806",
    "fdo-owner-services": "2.31.0-1498",
    "sdo-owner-services": "1.11.16-1083"
}
```

5.1.4 Outputs

This workflow creates a GitHub Release with the following configuration.

- Title: `v<version>`
- Tag: `v<version>`
- Assets:
  - `openhorizon-tested-versions.txt`
  - `Source code`

[9]https://github.com/open-horizon/examples/actions/workflows/release.yml
[10]https://docs.github.com/en/rest/actions/workflows?apiVersion=2022-11-28#create-a-workflow-dispatch-event

5.1.5 Variables and Secrets

The workflow needs access to no variables or secrets.

5.1.6 Conventions

This workflow will follow the following conventions...

- Release Creation
  - Rule: Release commit hash will be taken from the latest commit hash found in the branch that is related to the version. If said branch doesn't exist for the version it is likely that it is the latest version and commit hash will be taken from master branch.

5.1.7 Dependencies

This workflow has no notable dependencies.

6 Repository: Open Horizon Release

Open Horizon Releases is a central repository aimed at managing and coordinating the release process for various Open Horizon components.

6.1 Workflow: release.yml

6.1.1 Overview

In anticipation of an Open Horizon component release, this workflow is typically initiated manually. This workflow activates in response to a workflow dispatch initiated via the Actions interface or GitHub API. Utilizing the version JSON inputted, it triggers the release.yml workflows of both Anax and Exchange repositories. Upon successful execution, it generates its release, consolidating version details and links to the component releases within their specific repositories."

6.1.2 Triggers

This workflow is triggered by the following...

- Workflow Dispatch
  - `via Actions Interface`[11]
  - `via GitHub Rest API`[12]

6.1.3 Inputs

This workflow has the following inputs...

- `versionFileJSON` - String - The file containing all the versions (example below)

```json
{
    "amd64_agbot": "2.31.0-1498",
    "amd64_anax": "2.31.0-1498",
    "amd64_anax_k8s": "2.31.0-1498",
    "amd64_cloud-sync-service": "1.10.1-1498",
    "amd64_edge-sync-service": "1.10.1-1498",
    "amd64_exchange-api": "2.117.0-1163",
    "amd64_vault": "1.1.2-806",
    "fdo-owner-services": "2.31.0-1498",
    "sdo-owner-services": "1.11.16-1083"
}
```

- `INCREMENT_MAJOR_VERSION` - Boolean - Decides if the major (X.0.0) version should be incremented.
- `INCREMENT_MINOR_VERSION` - Boolean - Decides if the minor (0.X.0) version should be incremented.
- `INCREMENT_PATCH_VERSION` - Boolean - Decides if the patch (0.0.X) version should be incremented.
- `IS_LATEST` - Boolean - Decides if the latest tags get pushed to Dockerhub for releases such as Anax.

[11] https://github.com/open-horizon/open-horizon-release/actions/workflows/release.yml
[12] https://docs.github.com/en/rest/actions/workflows?apiVersion=2022-11-28#create-a-workflow-dispatch-event

6.1.4 Outputs

This workflow creates a GitHub Release with the following configuration.

- Title: `Open Horizon Release v<version>`
- Tag: `v<version>`
- Release Notes: (contains links to Anax and Examples releases)
- Assets:
  - `Released_Versions.json`
  - `Source code`

6.1.5 Variables and Secrets

The workflow needs access to the following variables and secrets...

- Repository Level [13]
  - Variables
    - `OPENHORIZON_RELEASE_VERSION` - The current version of Open Horizon.
  - Secrets
    - `OPENHORIZON_RELEASE_MANAGER_TOKEN` - GitHub token that has organization level 'repo' scope. Needs at minimum 'repo' for Open Horizon Release, Anax, and Examples.

6.1.6 Conventions

This workflow will follow the following conventions...

- Release Creation
  - Rule: Release commit hash is not specified and will be latest in main branch.
  - Rule: If increment version is set, the version used in the Open Horizon Release will always be the variable `OPENHORIZON_RELEASE_VERSION` post-increment. If you want to use the version that the variable is currently set to, do not increment.
- Version Increment: When incrementing the version, any less significant versions will be set to 0. For example incrementing the minor version of 1.3.5 results in 1.4.0 as the patch version is less significant.

6.1.7 Dependencies

This workflow has no notable dependencies.

[13] To set these variables, follow this path: Repository → Settings → Secrets and Variables → Actions.