Jenkins EPO Documentation

Release 1.159

Étienne BERSAC, James PIC

May 10, 2017

Contents

Time for kickass CI !

Implements extended CI features on top of Jenkins and GitHub for in-house CI.

CHAPTER 1

Features

- Define jobs from repository in jenkins.yml.
- Jobs pipeline façon GitLab CI.
- Query GitHub API to poll open PR instead of polling git repository.
- Read instructions from PR comments.
- Cancel running jobs when pushing new commits.
- Report issue on broken protected branches.
- Auto-merge PR.
- Works with webhook and/or behind firewall.
- Extensible through entry-point.

CHAPTER 2

Quickstart

On your poller host:

```
pip3 install jenkins-epo
# Setup env vars
export JENKINS_URL=http://myjenkins.lan JOBS_CREDENTIALS=clone-creds
export GITHUB_TOKEN=YOUR_SECRET_TOKEN REPOSITORIES=owner/repo
# Check repository is manageable
jenkins-epo list-heads
# Trigger a dry run
DRY_RUN=1 jenkins-epo process https://github.com/owner/repo/tree/master
# Run it for real
jenkins-epo bot
```

Now write a `jenkins.yml` file and open a PR:

```
myjob: |
    tox -r
```

Many instructions are available. Just ask the bot by commenting `jenkins: help` in an open PR!

CHAPTER 3

Table of contents

Installing

Jenkins EPO is a Python3.4+ software configured by environment variables. The package ships a systemd unit reading environment variable from `/etc/jenkins-epo.conf` file.

The recommended way of deploying Jenkins EPO is through Ansible with bersace.jenkins-epo.

But the first requirement is a Jenkins up and running. `jenkins-epo list-plugins` lists required Jenkins plugins to run managed jobs. It's up to Jenkins administrator to install these plugins.

Jenkins must be able to clone repositories with HTTPS. Register a Jenkins credentials for HTTPS clone, and set `JOBS_CREDENTIALS` according to it.

Next step is to have a GitHub API token. You can create one associated with the GitHub user assigned to Jenkins to clone.

Test your settings like this:

```
GITHUB_TOKEN=XXX JENKINS_URL=http://jenkins.lan JOBS_CREDENTIALS=github-https jenkins-
↪epo process https://github.com/owner/repo1/tree/master
```

Then write it to Ansible vars or in `/etc/jenkins-epo.conf` like this:

```
set -x
GITHUB_TOKEN=XXX
JENKINS_URL="http://jenkins.lan/"
JOBS_CREDENTIALS=github-https
REPOSITORIES=owner/repo1,owner/repo2
SERVER_URL=http://localhost:2819/
set +x
export REPOSITORIES GITHUB_TOKEN JENKINS_URL JOBS_CREDENTIALS
```

And reload with `systemctl restart jenkins-epo`. Watch it with `journalctl -fu jenkins-epo` !

Setting up WebHook

To increase EPO reactivity, you can use webhooks. EPO listen for webhook on port 2819. There is two webhooks entrypoints.

/simple-webhook

Just pass head URL as `head` GET param:

```
curl -X POST http://localhost:2819/simple-webhook?head=https://github.com/owner/repo1/
↪tree/master
```

At the end of each build, `jenkins-yml-runner` can notify one URL. EPO tells Jenkins which URL to notify using `SERVER_URL`. `SERVER_URL` points to EPO public address, accessible from node executing the build:

```
HOST=0.0.0.0 PORT=2819 SERVER_URL=http://jenkins.lan:2819 jenkins-epo bot
```

Watch for the following message at the end of your build log:

```
+ jenkins-yml-runner notify
Notifying http://jenkins.lan:2819/simple-webhook?head=https://github.com/owner/repo1/
↪tree/master (POST).
Success: b'{"message": "Event processing in progress."}'.
POST BUILD TASK : SUCCESS
```

Nice! Persist `SERVER_URL` in `/etc/jenkins-epo.conf` and you're done. EPO and Jenkins communicate to speed up the pipeline! Now you can go further!

/github-webhook

If you can open a port to the world, you can tell GitHub to notify EPO of changes on your protected branches or PR. Here is basically how to setup an nginx proxy to serve EPO to GitHub.

1. Register the domain, get the certificate, a host, etc.
2. Configure nginx, here is a sample host configuration.

```
# This sample config expose only one URL, with SSL. Adapt to your needs.

upstream webhook_handler {
    # Put here the hostname and port where EPO is listening.
    server jenkins.lan:2819;
}

server {
    # Put here the FQDN GitHub will notify.
    server_name jenkins.company.com;

    access_log  /var/log/nginx/epo-webhook.access.log;
    error_log   /var/log/nginx/epo-webhook.error.log;

    listen 443 ssl;

    # Get or generate a certificate. EPO requires a signed certificate.
    ssl_certificate /etc/ssl/certs/epo-webhook.crt;
    ssl_certificate_key /etc/ssl/private/epo-webhook.key;
```

```
    ssl_session_timeout 1d;
    ssl_session_cache shared:SSL:50m;
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-
↪ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-
↪SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-
↪SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256';
    ssl_stapling on;
    ssl_stapling_verify on;

    # HSTS maxage 15768000 seconds = 6 months
    add_header Strict-Transport-Security max-age=15768000;

    real_ip_header   X-Forwarded-For;
    set_real_ip_from 10.0.0.0/8;
    set_real_ip_from 127.0.0.0/8;
    set_real_ip_from 192.168.0.0/16;
    # Add other safe proxy (ngrok, cloudflare, etc.)
    # set_real_ip_from ...

    location /github-webhook {
        allow 10.0.0.0/8;
        allow 127.0.0.0/8;
        allow 192.168.0.0/16;
        # https://help.github.com/articles/github-s-ip-addresses/
        allow 192.30.252.0/22;
        deny  all;

        proxy_redirect      off;
        proxy_set_header    Host                    $http_host;
        proxy_set_header    X-Forwarded-For         $http_x_real_ip;
        proxy_pass          http://webhook_handler;
    }
}
```

3. Test it!

```
curl -X POST -H 'X-Forwarded-For: 8.8.8.8' https://jenkins.company.com/github-
↪webhook
curl -X POST -H 'X-Forwarded-For: 192.30.252.25' https://jenkins.company.com/
↪github-webhook
```

4. Now register EPO in GitHub.

   You need an admin `GITHUB_TOKEN`. *You should use a separate admin token*. For example, use a personnal token of yours.

   To increase security, EPO shares a secret with GitHub to sign payload. Save it in `/etc/jenkins-epo.conf`.

```
export GITHUB_SECRET=$(pwgen 64 1)
SERVER_URL=https://jenkins.company.com/ GITHUB_TOKEN=XXX jenkins-epo register
```

   Jenkins and GitHub can ping different URLs. Just override `SERVER_URL` with GitHub URL when calling `register`.

5. In GitHub web interface, you can test webhook delivery, ping webhook and redeliver a payload.

Now test it for real : push a new commit in a PR and see how fast the jobs are triggered!

Without GitHub webhook

You can still ping EPO with `/simple-webhook` (e.g. in git hooks, make rule, etc.).

You may want to decrease `POLL_INTERVAL`. EPO will throttle heads processing to spread GitHub API calls to fit the limit of 5000 calls per hour.

Adding a new repository

EPO can manage multiple repositories! Here are the steps to add a repository to EPO.

1. Add your bot user to repository's collaborators *with write access*.
2. Ensure your default branch is protected if you want it to be tested!
3. Add `owner/repo2` to `REPOSITORIES` setting in `/etc/jenkins-epo.conf`.
4. Restart EPO:

```
systemctl restart jenkins-epo
```

5. Watch it with `journalctl -fu jenkins-epo`, you should see:

```
=a892= [INFO    ] Working on https://github.com/owner/repo2/tree/master (a892eb2).
=a892= [WARNING ] No jenkins.yml. Skipping.
=a892= [INFO    ] Processed https://github.com/owner/repo2/tree/master (a892eb2).
```

6. Register GitHub webhook for this repository:

```
GITHUB_TOKEN=XXX REPOSITORIES=owner/repo2 jenkins-epo register
```

7. Create a PR to add `jenkins.yml`.

Enjoy!

Writing `jenkins.yml`

`jenkins.yml` allow developers to define per project configuration of the CI pipeline. The `jenkins.yml` file provide a mapping of all jobs to manage. The special entry `settings` allow to overrides some defaults settings.

Defining a job

The simplest job definition is a oneline YAML entry:

```
app-job: tox -r
```

Commands are wrapped in a `bash` script, executed with `-eux` shell options. This mean that any failing command breaks the job, undefined variable are not accepted and each executed command is echoed on stderr.

You can actually add a bunch of Jenkins job feature in YML:

```
app-job:
  # Target a specific node or node label
  node: slave0
  # Matrix. Only values in YML are triggered
  axis:
    TOXENV:
    - py34
    - py35
  # job parameterer, value is always read from YML
  parameters:
    TESTS: tests/
  # The script
  script: |
    tox -re $TOXENV -- $TESTS
  # clean up script, executed even on cancel/abort.
  after_script: |
    rm -rf coverage.xml
```

Tests report and coverage

`jenkins.yml` generated Jenkins jobs are full featured !

- Archive all files in `$CI_ARTEFACTS` directory.
- Import all `$CI_ARTEFACTS/xunit*.xml` files to generate a test report.
- Feed Cobertura plugin with `$CI_ARTEFACTS/coverage.xml` to generate a coverage report.

```
app-units: |
  pytest -vvvv --strict --showlocals \
      --junit-xml={env:CI_ARTEFACTS}/xunit.xml \
      --cov=app --cov-report=xml:{env:CI_ARTEFACTS}/coverage.xml
```

Defining a pipeline

EPO provides a simple pipeline inspired by GitLab CI. Jobs are groupped by stage. Default defined stages are `build`, `test` and `deploy`. The default stage of a job is `test`. Here is a sample pipeline.

```
settings:
  stages: [build, test, deploy]

app-build:
  # Attach job to one stage of the CI pipeline
  stage: build
  script: make

app-test:
  stage: test
  script: make test

app-deploy:
  stage: deploy
  # Limit job on specific branch
  branches: master
  script: fab prod update
```

No jobs are triggered if the previous stage has a missing or failed build.

There is nothing like manual build or environment. Also there is no UI yet other than GitHub.

Create a periodic job

You can define periodic job from `jenkins.yml`. Theses jobs are **never** triggered on push. Jenkins EPO take care of maintaining the job in Jenkins according to the latest `jenkins.yml` version.

```
app-task:
  # Run this job around 3:00AM
  periodic: H 3 * * *
  # Run only on master
  default_revision: refs/heads/master
```

Periodic jobs can be part of pipeline stage. The stage will be completed only when the periodic job will succeed. However, periodic jobs are out of pipeline by default.

Interacting with the bot

Jenkins EPO reads instructions from comments on PR, including description, and comments on commit for protected branches.

An instruction is always prefixed with `jenkins:`. It must be a valid YAML dict. Instructions on multiple line must be wrapped in MarkDown code block.

Available instructions can be reported by invoking `jenkins: man` in an open pull request.

Example of instructions

Simple one line instruction:

```
jenkins: skip
```

Parameterized intruction:

````
```
jenkins:
  jobs: '*units'
```
````

Complex instruction:

````
``` yaml
jenkins:
  parameters:
    test-job:
      PARAM0: 'override'
```
````

Marking urgent pull requests

Jenkins EPO priorize protected branches over pull-requests. It is possible to mark a pull request as urgent, to test it before protected branches and other pull requests. Prefix PR title with `[URGENT]` to increase PR priority in EPO queue.

Security

CI is about executing code. Here are some note on what checks are implemented in EPO to increase security.

- EPO considers only collaborators with *write* access.
- You can override collaborators in `jenkins.yml` of default branch: .. code-block:: yaml

  settings:

  collaborators:

  - owner
  - admin
  - dev0
  - dev1
- EPO builds only PR from collaborators.
- EPO reads instructions from collaborators only.
- You can allow an external PR to be tested. Say `jenkins: allow` in a comment. Author instructions **before ‘‘allow‘‘ wont be processed**. PR author will be considered as a collaborator with **write** access for this PR. This include automatic merge.
- Webhook are used only to determine the URL of the head: either `https://github.com/owner/repo/tree/branch` or `https://github.com/owner/repo/pull/1234`. Comments are not parsed from webhook.
- GitHub webhook payload **must** be signed with Hub secret token.
- For now, GitHub is accessed using a token. But Jenkins must be open.

Troubleshooting

Developers are impatient customers, but love to dive into problems to solve them. Here are some hint when you are not satisfied by EPO. The big question is:

My PR is not built :((!!

1. **EPO is a poller**

   And thus, can’t react immediatly. Wait a minute or two, depending on the load of your Jenkins.

2. **Is your latest commit older than 4 weeks ?**

   By default, EPO discard older branch:

   ```
   =9c6e= Skipping head older than 5 weeks.
   ```

   Just rebase and you’re done.

3. **Does EPO have Write access to the repository?**

   If you find the following message, consider adding write access to your GitHub bot user.

```
=d0d0= [ERROR   ] Write access denied to owner/name.
```

4. **Does EPO have too many repositories to poll ?**

   GitHub limits API calls to 5000 per hour per account. EPO output warnings when hitting rate limit:

```
=othr= [INFO    ] 92 remaining API calls. Consumed at 2017-02-07 13:50:32+00:00.␣
↪Reset at 2017-02-07 14:17:43+00:00.
=othr= [WARNING ] Throttling GitHub API calls by 121s.
```

   If this is the case, consider splitting EPO in two instances, with a different GitHub account. Dispatch repositories amongst EPO instances.

   You can also disable `autocancel` to reduce rate limit consumption. This extensions poll previous commits status to find a running build.

5. **Does EPO cache works properly ?**

   EPO cache is a file, and only one process can write to it. EPO still works if the cache is locked, but the cache may be outdated.

```
=main= [WARNING ] Cache locked, using read-only
```

   Ensure the cache file is unlocked or your EPO instance has it's own cache file using `EPO_CACHE_PATH` env var.

Reading EPO logs

EPO tries to provide meaningful log messages and level. When running in systemd unit, EPO output level as syslog token. Otherwise, log level is shown as usual:

```
$ jenkins-epo list-extensions
=main= [INFO    ] Starting jenkins-epo 1.123.
00 outdated
00 security
02 yaml
05 jenkins-createjobs
10 jenkins-stages
30 autocancel
30 jenkins-autocancel
30 skip
49 jenkins-canceller
50 jenkins-builder
90 help
90 merger
90 report
99 error
=othr= [INFO    ] Done.
```

The first field, wrapped in = is an asyncio task identifier.

`main` Emitted from synchronous code, out of an async task.

`othr` Emitted from an unnamed task.

`wkXX` Emitted from a main worker task.

Commit sha Emitted from the task processing a head. This is the git sha of the current commit processed.

Hacking

Releasing

Jenkins EPO version number is read from latest Git tag with `git describe --tags`. Use the following listing to tag and upload a new release.

```
$ ./release 1.90
```

Release early, release often is the way of Jenkins EPO !

The bot's story

The bot is a pipeline of extensions registered from `jenkins_epo.bot.extensions` entry-point. Each extension has a `stage` property to define its position in the pipeline. Use `jenkins-epo list-extensions` to see the pipeline, in processing order. The `EXTENSIONS` settings allow to disable some extensions.

Jenkins EPO invoke `bot.run()` coroutine to process one Head. The bot creates a context in `self.current` available in each extension's method call.

The pipeline is a Final State Machine. The steps `begin` and `process_instruction` inspect the Head to define the state of the Head. Then the final `run` step will apply next action.

First step of `bot.run()` is calling the `ext.begin()` method of each extension. This method should encapsulate any of the extension's variable initialization.

Then, `bot.run()` executes the `ext.process_instruction(instruction)` method for each instruction parsed from the GitHub comments by the bot. An extension can comment a PR and even write instruction for itself, usualy in HTML comment. See the `MergerExtension` for an example. This demonstrates how to maintain a state for an extension on a pull request.

Finnaly, the `ext.run()` coroutine is yielded. This is where the extension works with the different APIs to apply the next actions on the Head : trigger job, cancel job, report status, merge branch, etc.

`ext.begin()` and `ext.run()` can abort the pipeline by raising a special `SkipHead` exception.

Ideas of improvements

Feel free to contribute :)

- Aggregate errors comment.
- Detect branch from `commit_comment` event.
- Set custom status context. Create job name with <owner>_<project>_<job>. Context with only <job>.
- Cancel job on PR close.
- Skip event on comments not containing `jenkins:`.
- Test GraphQL.
- Add `clean-job` command to drop jobs undefined in protected branches.
- Comment old PR with «push a new commit».

- Switch to full AsyncIO - drop jenkinsapi - drop githubpy - see siesta
- Test merge commit (pull/XXXX/{head,merge})
- metrics: build count, total time on Jenkins, cancelled build (how much time saved), etc.
- Pipeline dashboard
- Command `install-plugins`. Install plugins on Jenkins
- Distinct global/per project settings.
- Command `settings [head]` dump settings, jenkins.yml loaded.
- Keep build forever on Jenkins for build reported in *master is broken*
- Manage regular Jenkins notifications (https://wiki.jenkins-ci.org/display/JENKINS/Notification+Plugin).
- i18n: translate documentation, comments, logs
- Add ansicolor to Jenkins job ?
- Disable extensions from jenkins.yml.
- Support “jenkins, skip”, “Jenkins, rebuild.”, “@bot merge”, “jenkins:rebuild”