scriptworker-scripts

Jan 18, 2022

Table of Contents:

This is the official mono repo containing all the scriptworker *scripts. As to November 2019, we have migrated all the workers, across all trees, to Kubernetes and Google Compute Cloud. Tagging along, we have also migrated all the individual scripts under the same roof in order to single source the shared configurations.

In a nutshell, we now user Docker-based scriptworkers scripts that perform various pieces of our automation. In order for deploying, we **no longer** rely on *hiera* or *puppet* but on Docker and SOPS.

The comprehensive list of workers that we have available is listed below. They are split in two large environments within the GCP: *releng-nonprod* and *releng-prod*.

The former holds all the *dev* workers. These are handy to use before submitting a PR or deployment to production in order to test things out. The environment holds rules for netflows as well in order to access the dev instances of our external resources.

The latter, *releng-prod* withhold two sets of workers. The *level-3* workers which are the production ones. We use these workers to ship the real, production-ready releases, across our different products (Firefox, Thunderbird, Firefox for mobile related suite, etc). In the same environment we also have the *level-1* workers which are used for staging releases. They co-exist here so that they are closer to production as possible.

Full documentation is available at https://scriptworker-scripts.readthedocs.io/en/latest/index.html.

CHAPTER 1

Overview of existing workers

Note: this is not a comprehensive list. We have added more scripts, more trust domains, and more pools since this list was compiled. The authoritative place to look for currently deployed scriptworkers is in https://github.com/mozilla-services/cloudops-infra/blob/master/projects/relengworker/Jenkinsfile, in the *ScriptWorkerTypes* section. Dev scriptworkers can be found in https://github.com/mozilla-services/cloudops-infra/blob/master/projects/relengworker/Jenkinsfile.dev.

1.1 addonscript

Worker type	Deployment name
gecko-1-addon-dev	addon-dev-relengworker-firefoxci-gecko-1
gecko-3-addon	addon-prod-relengworker-firefoxci-gecko-3
gecko-1-addon	addon-prod-relengworker-firefoxci-gecko-1

1.2 balrogscript

Worker type	Deployment name
gecko-1-balrog-dev	balrog-dev-relengworker-firefoxci-gecko-1
gecko-3-balrog	balrog-prod-relengworker-firefoxci-gecko-3
gecko-1-balrog	balrog-prod-relengworker-firefoxci-gecko-1
comm-3-balrog	balrog-prod-relengworker-firefoxci-comm-3
comm-1-balrog	balrog-prod-relengworker-firefoxci-comm-1

1.3 beetmoverscript

Worker type	Deployment name
gecko-1-beetmover-dev	beetmover-dev-relengworker-firefoxci-gecko-1
gecko-3-beetmover	beetmover-prod-relengworker-firefoxci-gecko-3
gecko-1-beetmover	beetmover-prod-relengworker-firefoxci-gecko-1
comm-3-beetmover	beetmover-prod-relengworker-firefoxci-comm-3
appservices-3-beetmover	beetmover-prod-relengworker-firefoxci-applicationservices-3
appservices-1-beetmover	beetmover-prod-relengworker-firefoxci-applicationservices-1
mobile-3-beetmover	beetmover-prod-relengworker-firefoxci-mobile-3
mobile-1-beetmover	beetmover-prod-relengworker-firefoxci-mobile-1

1.4 bouncerscript

Worker type	Deployment name
gecko-1-bouncer-dev	bouncer-dev-relengworker-firefoxci-gecko-1
gecko-3-bouncer	bouncer-prod-relengworker-firefoxci-gecko-3
gecko-1-bouncer	bouncer-prod-relengworker-firefoxci-gecko-1
comm-3-bouncer	bouncer-prod-relengworker-firefoxci-comm-3

1.5 pushapkscript

Worker type	Deployment name
gecko-1-pushapk-dev	pushapk-dev-relengworker-firefoxci-gecko-1
gecko-3-pushapk	pushapk-prod-relengworker-firefoxci-gecko-3
gecko-1-pushapk	pushapk-prod-relengworker-firefoxci-gecko-1
mobile-3-pushapk	pushapk-prod-relengworker-firefoxci-mobile-3
mobile-1-pushapk	pushapk-prod-relengworker-firefoxci-mobile-1

1.6 pushflatpakscript

Worker type	Deployment name
gecko-1-pushflat-dev	pushflat-dev-relengworker-firefoxci-gecko-1
gecko-3-pushflat	pushflat-prod-relengworker-firefoxci-gecko-3
gecko-1-pushflat	pushflat-prod-relengworker-firefoxci-gecko-1

1.7 pushmsixscript

Worker type	Deployment name
gecko-1-pushmsix-dev	pushmsix-dev-relengworker-firefoxci-gecko-1
gecko-3-pushmsix	pushmsix-prod-relengworker-firefoxci-gecko-3
gecko-1-pushmsix	pushmsix-prod-relengworker-firefoxci-gecko-1

1.8 shipitscript

Worker type	Deployment name
gecko-1-shipit-dev	shipit-dev-relengworker-firefoxci-gecko-1
gecko-3-shipit	shipit-prod-relengworker-firefoxci-gecko-3
gecko-1-shipit	shipit-prod-relengworker-firefoxci-gecko-1
comm-3-shipit	shipit-prod-relengworker-firefoxci-comm-3
comm-1-shipit	shipit-prod-relengworker-firefoxci-comm-1

1.9 signingscript

Worker type	Deployment name
gecko-1-shipit-dev	shipit-dev-relengworker-firefoxci-gecko-1
gecko-3-signing	signing-prod-relengworker-firefoxci-gecko-3
gecko-t-signing	signing-prod-relengworker-firefoxci-gecko-t
mobile-3-signing	signing-prod-relengworker-firefoxci-mobile-3
mobile-t-signing	signing-prod-relengworker-firefoxci-mobile-t
comm-3-signing	signing-prod-relengworker-firefoxci-comm-3
comm-t-signing	signing-prod-relengworker-firefoxci-comm-t
appservices-3-signing	signing-prod-relengworker-firefoxci-applicationservices-3
appservices-t-signing	signing-prod-relengworker-firefoxci-applicationservices-t
xpi-3-signing	signing-prod-relengworker-firefoxci-xpi-3-1
xpi-t-signing	signing-prod-relengworker-firefoxci-xpi-t
xpi-t-signing-dev	signing-dev-relengworker-firefoxci-xpi-t-1

1.10 treescript

Worker type	Deployment name
gecko-1-tree-dev	tree-dev-relengworker-firefoxci-gecko-1
gecko-3-tree	tree-prod-relengworker-firefoxci-gecko-3
gecko-1-tree	tree-prod-relengworker-firefoxci-gecko-1
comm-3-tree	tree-prod-relengworker-firefoxci-comm-3

CHAPTER 2

Update python dependencies

```
# from scriptworker-scripts/ ; this will run docker for py38 and py39
# for all *scripts to update all the dependencies via `pip-compile-multi`
$ maintenance/pin.sh
```

CHAPTER 3

Testing code changes

Each directory is a different tool with different testing needs.

When updating the entire set of tools here are a few steps that could help:

- push changes to `dev` branch (if a single tool, use `dev-<tool>`), wait for deployment in #releng-notifications in Slack
  - `git push --dry-run upstream <my_pr_branch>:dev`
- do a staging release of an xpi manifest (covers github script, signingscript, shipitscript)
  - add a change like this to `staging-xpi-manifest`
  - wait for it to be deployed
  - Go to ShipIt staging and create a new `XPI Release`, selecting `staging-xpi-public`
  - Once started, go to `xpi releases` and build, promote, ship (need signatures for this) - ensure all jobs complete
  - Make sure to revert changes to any repos
- do a try push using `-dev` instances running select jobs (covers winsign, beetmoverscript, balrogscript)
  - change taskcluster/ci/config.yml to edit the staging machine types:
    - beetmover::staging: '{trust-domain}-1-beetmover' -> '{trust-domain}-1-beetmover-dev'
    - linux-depsigning::worker-type: '{trust-domain}-t-signing' -> '{trust-domain}-t-signing-dev'
    - mac-depsigning::worker-type: 'depsigning-mac-v1' -> 'depsigning-mac-v1-dev' (NOTE: we don't test this)
    - mac-notorization-poller::worker-type: 'mac-notarization-poller' -> 'mac-notarization-poller-dev' (NOTE: we don't test this)
    - mac-signing::staging: 'depsigning-mac-v1' -> 'depsigning-mac-v1-dev' (NOTE: we don't test this)
    - tree::staging: '{trust-domain}-1-tree' -> '{trust-domain}-1-tree-dev'

    * Then run `./mach try fuzzy --full` and select `build-signing`, `release-balrog`, `balrog-en-CA`, `beetmover` jobs. This will select hundreds of jobs (mostly language repacks), but will get a lot of coverage

- For all of these (just 1 language pack), examine the logs to ensure using the `-dev` workers and that there are no red flags (like an error that doesn't cause the job to fail)

3.1 FAQ

3.1.1 How do I deploy changes to a specific scriptworker script?

If you made changes to a specific scriptworker script and you merged your PR, you now have two options:

- deploy those changes to just that scriptworker script - in which case, push your changes to `production-$script` (e.g. `production-beetmoverscript` or `production-signingscript`)
- deploy to all of them - in which case, push your changes to `production` and let all the workers be updated

3.1.2 What happens in a nutshell when we push to the `production` branches?

The Taskcluster CI jobs rebuild the Docker images and push them to the Dockerhub. There are two pushes happening, one timestamped and one for the `latest` tag, that's to be used by CloudOps. They have webhooks setup to trigger, at their turn, should there is a new image pushed. When that happens, their Jenkins pulls that newly pushed image from Docker and mirrors it in their ecosystem and then deploys it in GCP.

What this means is that, if there are sometimes intermittent issues with the deployment in the CloudOps world (race condition in downloading docker images or alike) **rerunning** the latest push-docker-image job from our CI retriggers their deployment (because timetamped images, rerunning pushes another image even though the underlying code hasn't changed).

3.1.3 How do I debug locally an image?

Sometimes the deployments fail for missing environment variables or alike. In order to debug a production image, one can download that image from Dockerhub and run it locally. Remember, secrets are passed over via env vars, so you'd have to do something similar on your local machine:

- for a certain `$script`, hop on `https://hub.docker.com/r/mozilla/releng-{$script}` and find the latest image pushed
- pull that image locally by `docker pull mozilla/releng-$script:production--$timestamp-$hash`
- use `pass` to define some dummy values replacing the one it'd expect in production (e.g. a file called `local-prod`)
- run the image locally by doing `docker run -ti $(pass show local-prod | grep -v ^# | grep -v '^$' | sed 's/^/-e /') image_name /bin/bash`

where the format of the file is, e.g.

```
PROJECT_NAME=beetmover
ENV=dev
COT_PRODUCT=mobile
TASKCLUSTER_ROOT_URL=https://firefox-ci-tc.services.mozilla.com
```

(continues on next page)

(continued from previous page)

```
TASKCLUSTER_CLIENT_ID=fake-client
TASKCLUSTER_ACCESS_TOKEN=token
...
```

- once docker container started successfuly, run `./docker.d/init.sh` to simulate what the deployment is doing.

3.1.4 How to I update a secret in the new world?

Secrets are now stored in SOPS. Once they are updated there, the scriptworkers need to be (re)taught of their change. In order for that to happen, we need to redeploy the scriptworkers.

A. Updating an existing secret is easier since it only implies updating SOPS and rerunning the scriptworkers (one specific or all of them, depending on the branch chosen when pushing)

B. Adding a new secret needs an extra step, as it needs to be defined in the cloudops-infra files. The general rule of thumb to make this is usually:

```
1) secrets pushed in SOPS
2) cloudops-infra PR merged
3) scriptworker-scripts deployment to pick-up both changes from above
```

3.2 Dockerization of scriptworkers

Every scriptworker has its own `Dockerfile` in the root directory. The commands are limited to the version used in Taskcluster and may not support newer features. The file is kept simple and most of the logic is handled by files in the `docker.d` directory.

- `docker.d/init.sh`

  This file contains logic that is the same for all workers.

- `$script/docker.d/init_worker.sh`

  This file contains logic that is worker specific.

Both init files explicitly use shell's `test` to check for all required environment variables in order to fail as soon as possible.

- `$script/docker.d/worker.yml`

  These files are JSON-e templates for scriptworker and the implementation script. The final configs are generated during the initial boot process.

3.3 Continuous Integration

Every pull request runs unit tests and makes sure that we can build a docker image. CloudOps deployments happen only when the change is pushed to either `dev` or `production` branches (or their related `dev-$script` or `prod-$script` per/script associated branches such as `dev-beetmoverscript` or `prod-signingscript`). The only exception is the k8s-autoscale repo, which deploys to the `nonprod` environment on every push to `master` to make sure we have the latest version running and tested before we push it to production, and to the `production` environment, when a change is pushed to the `production` branch.

In order to debug any issues with the CloudOps deployments make sure you have access to CloudOps Jenkins. File a bug similar to this one in order to get access. You will need to use Duo and SSH proxy in order to access it. See the instruction for the details.

3.4 Dev environment

In case you want to test your changes before pushing to production or submitting a PR, you can "force push" your changes to the `dev-{script}` (e.g. `dev-beetmoverscript` or `dev-signingscript`) should you'd like to update only one worker, or the `dev` branch to be deployed to all `dev` workers from the nonprod Kubernetes cluster.

You also need to adjust the in-tree configs to use the `-dev` workerType, e.g. `gecko-1-beetmover-dev`. If you need to test workers other than gecko, you can change `init.sh` and/or `init_worker.sh` in order to set proper worker type. Also you may need to ask CloupdOps to add the corresponding environment variables or secrets.

3.5 Testing scripts locally

Most `*scripts` will run locally, or in a local docker container. (Exceptions may include `iscript` and `notarization_poller`, which are targeted to run on mac hardware.)

To test the `*scripts` fully locally, you need the secrets. Dep and dev `*script` pools tend to have low-security secrets, so these might be ok to use locally. You may also be able to create your own account(s), e.g. a Google Play account for pushapk testing, or a throwaway gpg key for signing testing.

For a given `*script`, these would be the steps to test on your laptop or docker container:

3.5.1 Create the `script_config.yaml`

Example `script_config.yaml` files can be found in each `*script`'s `docker.d/worker.yml` file. These are YAML with json-e config so they can support multiple configurations in the same file. You'll want to resolve the json-e portions and just have a simple yaml or json config file.

Your `work_dir` and `artifact_dir` should be absolute paths. You may want to use `$PWD/work` and `$PWD/artifacts`.

3.5.2 Download a docker image

You can download a docker image built in taskcluster via either `./mach taskgraph load-image` or `taskgraph load-image`, depending on whether you're working from mozilla-central or from a standalone taskgraph virtualenv. If you're able to either populate the right secrets via env vars, or start the container interactively, you may be good.

It's also possible to use, say, a python 3 image and create a virtualenv manually.

3.5.3 Install the virtualenv

Create the virtualenv, e.g. `pyenv virtualenv NAME; pyenv activate NAME`

Next, install the scriptworker-scripts dependencies.

- If your target `*script`'s `setup.py` contains `scriptworker_client`, you want to run `cd scriptworker_client && python setup.py develop && cd ..`

- If your target `*script`'s `setup.py` contains `mozbuild`, you want to run `cd vendored/mozbuild && python setup.py develop && cd ../..`

Then install your `*script`. `cd SCRIPTNAME && python setup.py develop && cd ..`

3.5.4 Download a task to test

Assuming the `*script` is already live, you can download an existing task and modify it before running. You can do this by:

- Make sure you've installed `scriptworker` in your virtualenv
- `scriptworker` will provide a `create_test_workdir` helper tool.

```
create_test_workdir --help  # for help
create_test_workdir [--path PATH] [--overwrite] TASK_ID
```

  This will create a `./work` directory, populate `./work/task.json` with the task definition of task `TASK_ID`, and download any `task.payload.upstreamArtifacts` in `./work/cot/UPSTREAM_TASK_ID/PATH/TO/ARTIFACT`

- Optionally edit the task.json to test what you want to test. Optionally modify the contents of `./work/cot/...` to test what you want to test.
- Make sure your `script_config.yaml` is pointing at the same path for `work_dir`.
- run `SCRIPTNAME script_config.yaml | tee log` to run the script against that task while capturing log output in a file and the console.

3.6 Production environment

After a change is pushed to the `production` branch (or individual per/script `production-$script`) and passed CI, the CloudOps deployment process gracefully deploys it to the corresponding production deployments.

The deployment status is reported in the `#releng-notifications` Slack channel.

3.7 Secrets

All secrets are generated from by the CloudOps deployment process, see example template.

All secrets are passed to the replicas via environment variables and replaced in the configs using JSON-e or saved to files. In latter case please encode the contents of the file using `base64 -w0` before handling them to CloudOps, and use `echo $VAR | base64 -d > file` to save the value into a file in `init_worker.sh`.

Similarly to environment variables, the secrets use camelCase, and then converted to SHELL_STYLE. When you pass the secrets to CloudOps, use camelCase and YAML format. For example:

```
mySecret1: 'supersecretthing'
mySecret1: 'supersecretthing'
```

For secrets transferring, please consult this mana page.

3.8 Kubernetes

The scriptworkers are hosted in Google Compute Cloud (GCP) using Kubernetes. The deployment process is managed by the CloudOps team.

If you want to debug the GCP workers in the console, make sure you have access to the GCP console for the production and nonprod environments. You can get access by filing a bug against CloudOps.

The CloudOps deployment process is managed by Jenkins and the corresponding files can be found in the cloudops-infra repo.

The environment variables are set per deployment and can be found in the k8s/values directory. The default values are set in a separate file. The variables use camelCase, but then converted to SHELL_STYLE in configmap.yml

When Kubernetes decides to stop a worker it gives it 1200s to do this gracefully, see the corresponding k8s config. As a part of shutdown process, Kubernetes runs docker.d/pre-stop.sh to us enough time to finish running tasks. `docker.d/pre-stop.sh` sends `SIGUSR1` to scriptworker, what makes it exit as soon as it finishes the running task. 2 minutes before the deadline `docker.d/pre-stop.sh` exits and lets Kubernetes send `SIGTERM` what makes scriptworker cancel the task and upload the logs.

Kubernetes also runs a health check, by calling `docker.d/healthcheck` periodically. If the script exits non-zero or times out, the corresponding replica is marked as non healthy and will be removed from the pool.

3.9 Autoscaling

We use our own way to autoscale the amount of replicas using k8s-autoscale. It looks at the pending queue and adjusts the amount of replicas depending on average task duration, SLA and the maximum amount of replicas we want to run.

The configs can be found in the configs directory. Some important config variables are below:

- `worker_type`: corresponds to Taskcluster's `workerType`
- `deployment_namespace`: corresponds to deployment's namespace in Kubernetes and used in the Kubernetes API queries.
- `deployment_name`: corresponds to deployment's name in Kubernetes and used in the Kubernetes API queries.
- `max_replicas` and `min_replicas`: set the max and min amount of replicas
- `avg_task_duration`: average task duration. Used in calculations and affects the amount of replicas we spin up.
- `slo_seconds`: (service level objective) how many seconds we tolerate waiting until we start a pending task. For example, with 1 running instance, `slo_seconds` set to 240 and `avg_task_duration` set to 60, we don't spin up new instances until we have more than 4 pending tasks.
- `capacity_ratio`: a value between 0 and 1, which tells what portion of the pending pool this entry can handle. Used in case we want to use multiple entries for the same worker type in different clusters.

After a change is merged to the `master` branch, it's immediately deployed to the dev GCP cluster. In order to deploy the changes to production, you need to merge from `master` to the `production` branch. Moreover, in order for the change to have effect in the desired scriptworker(s), a new image for the latter needs to be pushed out to Docker.

3.10 Troubleshooting

- Check the logs uploaded to Taskcluster.

- Check the Kubernetes container logs. They can be viewed per deployment or per replica.
- Check the Jenkins logs.
- Ask CloudOps ofr help. They can use `kubectl attach` to see what happens in the container.

3.11 Rolling back a deploy

Currently, there are two ways to roll back a scriptworker pool deploy:

- [force] push the previous-known-good revision to the `production-_____script` branch and wait for its `k8s-image` task to finish, or
- find the `k8s-image` task that deployed the previous-known-good image to docker hub, and either `rerun (force)` it (if its deadline hasn't passed) or `retrigger` it.

(In the future, Aki would love to see release promotion graphs that can push previously built k8s-image docker images to the right kubernetes clusters, but we've long wanted this and have yet to be able to prioritize it.)

3.11.1 How to find the previous revision

Because git and Github don't have a pushlog, it's difficult to tell what the previous revision to a given branch is: the green checks, red X's, and yellow dots could be checks from a PR or push to another branch. If weeks or months pass between pushes to a given branch, you might have to check many many revisions before you find the previous revision in the branch. And because we support both the `production` and `production-_____script` branches to deploy scriptworkers, we need to check to see which branch had the newer previous-most-recent push.

Enter docker hub. We push our scriptworker docker images to the mozilla org. You can find the repositories at `https://hub.docker.com/r/mozilla/releng-_____script/`, e.g. `https://hub.docker.com/r/mozilla/releng-signingscript/` for signingscript.

We tag each push with multiple tags. You can view the tags by recency, e.g. `https://hub.docker.com/r/mozilla/releng-signingscript/tags?page=1&ordering=last_updated`.

The `production` and `dev` tags are set to the most recent push. But the other tags are of interest here: they're named `production-DATESTRING-REVISION` or `dev-DATESTRING-REVISION`. For instance, the signingscript `production-20210929025643-7995c5bb123bbfc25e3d7f81c46f3d7ba49cbe89` tag was pushed on 2021-09-29 at 02:56:43 (UTC?), from revision `7995c5bb123bbfc25e3d7f81c46f3d7ba49cbe89`.

If the most recent `production-DATESTRING-REVISION` tag is known busted, and the previous `production-DATESTRING-REVISION` tag is days or weeks old, most likely that previous tag was known good for that period of time. Take the revision in that tag, and force-push it to the `production-_____script` branch.

3.12 Scriptworker-scripts Readme

This is the official mono repo containing all the scriptworker *scripts. As to November 2019, we have migrated all the workers, across all trees, to Kubernetes and Google Compute Cloud. Tagging along, we have also migrated all the individual scripts under the same roof in order to single source the shared configurations.

In a nutshell, we now user Docker-based scriptworkers scripts that perform various pieces of our automation. In order for deploying, we **no longer** rely on *hiera* or *puppet* but on Docker and SOPS.

The comprehensive list of workers that we have available is listed below. They are split in two large environments within the GCP: *releng-nonprod* and *releng-prod*.

The former holds all the *dev* workers. These are handy to use before submitting a PR or deployment to production in order to test things out. The environment holds rules for netflows as well in order to access the dev instances of our external resources.

The latter, *releng-prod* withhold two sets of workers. The *level-3* workers which are the production ones. We use these workers to ship the real, production-ready releases, across our different products (Firefox, Thunderbird, Firefox for mobile related suite, etc). In the same environment we also have the *level-1* workers which are used for staging releases. They co-exist here so that they are closer to production as possible.

Full documentation is available at https://scriptworker-scripts.readthedocs.io/en/latest/index.html.

3.13 Overview of existing workers

Note: this is not a comprehensive list. We have added more scripts, more trust domains, and more pools since this list was compiled. The authoritative place to look for currently deployed scriptworkers is in https://github.com/mozilla-services/cloudops-infra/blob/master/projects/relengworker/Jenkinsfile, in the *ScriptWorkerTypes* section. Dev scriptworkers can be found in https://github.com/mozilla-services/cloudops-infra/blob/master/projects/relengworker/Jenkinsfile.dev.

3.13.1 addonscript

Worker type	Deployment name
gecko-1-addon-dev	addon-dev-relengworker-firefoxci-gecko-1
gecko-3-addon	addon-prod-relengworker-firefoxci-gecko-3
gecko-1-addon	addon-prod-relengworker-firefoxci-gecko-1

3.13.2 balrogscript

Worker type	Deployment name
gecko-1-balrog-dev	balrog-dev-relengworker-firefoxci-gecko-1
gecko-3-balrog	balrog-prod-relengworker-firefoxci-gecko-3
gecko-1-balrog	balrog-prod-relengworker-firefoxci-gecko-1
comm-3-balrog	balrog-prod-relengworker-firefoxci-comm-3
comm-1-balrog	balrog-prod-relengworker-firefoxci-comm-1

3.13.3 beetmoverscript

Worker type	Deployment name
gecko-1-beetmover-dev	beetmover-dev-relengworker-firefoxci-gecko-1
gecko-3-beetmover	beetmover-prod-relengworker-firefoxci-gecko-3
gecko-1-beetmover	beetmover-prod-relengworker-firefoxci-gecko-1
comm-3-beetmover	beetmover-prod-relengworker-firefoxci-comm-3
appservices-3-beetmover	beetmover-prod-relengworker-firefoxci-applicationservices-3
appservices-1-beetmover	beetmover-prod-relengworker-firefoxci-applicationservices-1
mobile-3-beetmover	beetmover-prod-relengworker-firefoxci-mobile-3
mobile-1-beetmover	beetmover-prod-relengworker-firefoxci-mobile-1

3.13.4 bouncerscript

Worker type	Deployment name
gecko-1-bouncer-dev	bouncer-dev-relengworker-firefoxci-gecko-1
gecko-3-bouncer	bouncer-prod-relengworker-firefoxci-gecko-3
gecko-1-bouncer	bouncer-prod-relengworker-firefoxci-gecko-1
comm-3-bouncer	bouncer-prod-relengworker-firefoxci-comm-3

3.13.5 pushapkscript

Worker type	Deployment name
gecko-1-pushapk-dev	pushapk-dev-relengworker-firefoxci-gecko-1
gecko-3-pushapk	pushapk-prod-relengworker-firefoxci-gecko-3
gecko-1-pushapk	pushapk-prod-relengworker-firefoxci-gecko-1
mobile-3-pushapk	pushapk-prod-relengworker-firefoxci-mobile-3
mobile-1-pushapk	pushapk-prod-relengworker-firefoxci-mobile-1

3.13.6 pushflatpakscript

Worker type	Deployment name
gecko-1-pushflat-dev	pushflat-dev-relengworker-firefoxci-gecko-1
gecko-3-pushflat	pushflat-prod-relengworker-firefoxci-gecko-3
gecko-1-pushflat	pushflat-prod-relengworker-firefoxci-gecko-1

3.13.7 pushmsixscript

Worker type	Deployment name
gecko-1-pushmsix-dev	pushmsix-dev-relengworker-firefoxci-gecko-1
gecko-3-pushmsix	pushmsix-prod-relengworker-firefoxci-gecko-3
gecko-1-pushmsix	pushmsix-prod-relengworker-firefoxci-gecko-1

3.13.8 shipitscript

Worker type	Deployment name
gecko-1-shipit-dev	shipit-dev-relengworker-firefoxci-gecko-1
gecko-3-shipit	shipit-prod-relengworker-firefoxci-gecko-3
gecko-1-shipit	shipit-prod-relengworker-firefoxci-gecko-1
comm-3-shipit	shipit-prod-relengworker-firefoxci-comm-3
comm-1-shipit	shipit-prod-relengworker-firefoxci-comm-1

3.13.9 signingscript

Worker type	Deployment name
gecko-1-shipit-dev	shipit-dev-relengworker-firefoxci-gecko-1
gecko-3-signing	signing-prod-relengworker-firefoxci-gecko-3
gecko-t-signing	signing-prod-relengworker-firefoxci-gecko-t
mobile-3-signing	signing-prod-relengworker-firefoxci-mobile-3
mobile-t-signing	signing-prod-relengworker-firefoxci-mobile-t
comm-3-signing	signing-prod-relengworker-firefoxci-comm-3
comm-t-signing	signing-prod-relengworker-firefoxci-comm-t
appservices-3-signing	signing-prod-relengworker-firefoxci-applicationservices-3
appservices-t-signing	signing-prod-relengworker-firefoxci-applicationservices-t
xpi-3-signing	signing-prod-relengworker-firefoxci-xpi-3-1
xpi-t-signing	signing-prod-relengworker-firefoxci-xpi-t
xpi-t-signing-dev	signing-dev-relengworker-firefoxci-xpi-t-1

3.13.10 treescript

Worker type	Deployment name
gecko-1-tree-dev	tree-dev-relengworker-firefoxci-gecko-1
gecko-3-tree	tree-prod-relengworker-firefoxci-gecko-3
gecko-1-tree	tree-prod-relengworker-firefoxci-gecko-1
comm-3-tree	tree-prod-relengworker-firefoxci-comm-3

3.14 Update python dependencies

```
# from scriptworker-scripts/ ; this will run docker for py38 and py39
# for all *scripts to update all the dependencies via `pip-compile-multi`
$ maintenance/pin.sh
```

3.15 Testing code changes

Each directory is a different tool with different testing needs.

When updating the entire set of tools here are a few steps that could help:

- push changes to `dev` branch (if a single tool, use `dev-<tool>`), wait for deployment in #releng-notifications in Slack
  - `git push --dry-run upstream <my_pr_branch>:dev`
- do a staging release of an xpi manifest (covers github script, signingscript, shipitscript)
  - add a change like this to `staging-xpi-manifest`
  - wait for it to be deployed
  - Go to ShipIt staging and create a new `XPI Release`, selecting `staging-xpi-public`
  - Once started, go to `xpi releases` and build, promote, ship (need signatures for this) - ensure all jobs complete

  - Make sure to revert changes to any repos
- do a try push using `-dev` instances running select jobs (covers winsign, beetmoverscript, balrogscript)
  - change taskcluster/ci/config.yml to edit the staging machine types:
    * beetmover::staging: '{trust-domain}-1-beetmover' -> '{trust-domain}-1-beetmover-dev'
    * linux-depsigning::worker-type: '{trust-domain}-t-signing' -> '{trust-domain}-t-signing-dev'
    * mac-depsigning::worker-type: 'depsigning-mac-v1' -> 'depsigning-mac-v1-dev' (NOTE: we don't test this)
    * mac-notorization-poller::worker-type: 'mac-notarization-poller' -> 'mac-notarization-poller-dev' (NOTE: we don't test this)
    * mac-signing::staging: 'depsigning-mac-v1' -> 'depsigning-mac-v1-dev' (NOTE: we don't test this)
    * tree::staging: '{trust-domain}-1-tree' -> '{trust-domain}-1-tree-dev'
    * Then run `./mach try fuzzy --full` and select `build-signing`, `release-balrog`, `balrog-en-CA`, `beetmover` jobs. This will select hundreds of jobs (mostly language repacks), but will get a lot of coverage
- For all of these (just 1 language pack), examine the logs to ensure using the `-dev` workers and that there are no red flags (like an error that doesn't cause the job to fail)
- genindex
- modindex
- search