How we hack on Cockpit Infrastructure

Goal

Goal

infrastructure ~ code

self-validating PR → container rebuilds, private temp deployment, tests against product main branches

land PR → redeploy to production

- most of us have become really good at self-validating changes to our product code with test gating
- ideal: want to treat changes to infrastructure alike: submit a PR, builds changed container images
- in Cockpit team we mostly are there for the workloads that run inside the infra, but still quite far from that for changing the infra itself
- takes a lot of learning of new concepts and infrastructure, needs to offset the cost of classic deploy-watch-rollback

How we hack on Cockpit Infrastructure

Updating unit test container

Updating unit test container

.github/workflows/unit-tests.yml:

```
name: unit-tests
on: pull_request
[...]
  - name: Build unit test container if it changed
    run: |
      changes=$(git diff origin/master..HEAD --
                containers/unit-tests/)
      [ -z "$changes" ] || podman build \
        --tag ghcr.io/cockpit-project/unit-tests \
        containers/unit-tests/

  - name: Run unit-tests container
    run: |
      podman run -v $(pwd):/source:ro \
        ghcr.io/cockpit-project/unit-tests
```

- a simple case where this works well is our unit-tests container for cockpit
- you see simplified workflow that runs on PRs
- normal PRs pull container from the registry
- PR that touches anything in the container definition rebuilds the container, and runs unit tests against that local build
- provides self-validation that we want
- fairly new, currently missing: automatically refresh the container on registry on landing

How we hack on Cockpit Infrastructure

Keeping unit-tests container up to date

workflows/unit-tests-refresh.yml

```yaml
on:
  schedule:
    # auto-refresh every Sunday evening
    - cron: '0 22 * * 0'
[...]
  - name: Build fresh containers
    run: |
      podman build --tag ghcr.io/cockpit-project/unit-tests \
          containers/unit-tests/

  - name: Run amd64 clang test
    run: containers/unit-tests/start CC=clang

  - name: Push containers to registry
    run: podman push ghcr.io/cockpit-project/unit-tests
```

- because unit-tests container is easy to self-validate, we can keep it up to date automatically
- every week a scheduled workflow rebuilds the container, runs all unit test scenarios
- if they succeed, push it to the registry
- if they fail, GitHub sends a failed workflow notification email; investigate
- in the latter case, PRs just keep using the previous container; no urgency

Developing GitHub workflows

Test on your fork:

- example: cockpit-ostree npm-update
- example: homepage docs auto-update

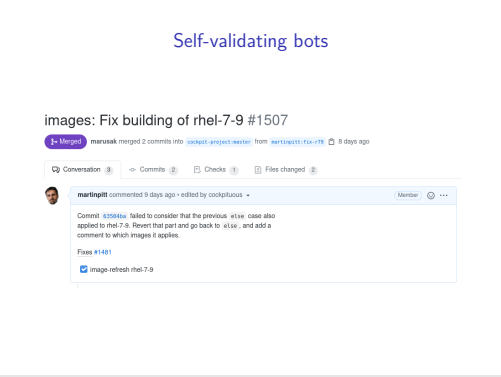
Interactive SSH for debugging:

```
uses: mxschmitt/action-tmate@v3
```

- Almost trivial, just for completeness
- No persistent deployment for GitHub; this “serverless” architecture avoids the whole initial problem
- Anyone can test changes on their own project fork
- the two real-life examples are clickable links, if you want to peek into how that looks like
- Biggest stumbling block there are secrets – you may need corresponding “forks” on quay.io, or upload the official secrets to your own forked project
- standard action on the market place for getting interactive ssh into the GitHub VMs

How we hack on Cockpit Infrastructure

Self-validating bots

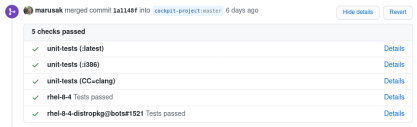


- As mentioned yesterday, bots has all the code that runs *inside* of our tasks containers; invoke tests, update translations, build VM image, etc.
- by far the most busy CI related project, several changes every day
- here I made a change to the build script of the rhel-7-9 image
- I added this little checkbox with a task to rebuild the image, the bot did it, committed the result to the PR, and checked the box
- This was using the code from the PR

How we hack on Cockpit Infrastructure

Run PR against a bots PR

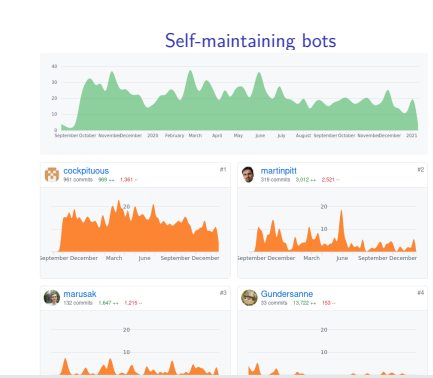
Run PR against a bots PR



- We can also run a cockpit or other project test against non-master bots with a special test syntax
- Trigger the test, and only land the bots PR on success

How we hack on Cockpit Infrastructure

Self-maintaining bots



- fun fact: the biggest contributor to bots by a wide margin is.. the bots
- bulk is automatic image refreshes, triggering all affected tests
- we could fully automate their landing, but we like to press green buttons!
- more often than you'd think these have test failures due to OS changes/regressions

How we hack on Cockpit Infrastructure

Updates to deployed infrastructure

- anything that touches our deployed infrastructure has no particular magic
- these are farthest away from that goal, fortunately also rare
- updating tasks container: send PR, build/push/deploy it, trigger/wait for some representative test runs, watch out for regressions
- we can run the new container locally or on the infra and do a smoke test
- quay.io offers a nice and simple way to revert a tag on the web UI (easier than on the CLI)
- similar situation with changes to kubernetes resources, Ansible scripts, etc.
- we don't have sharding of our infra and per-developer tenants
- so we usually deploy right to production and roll back on failure