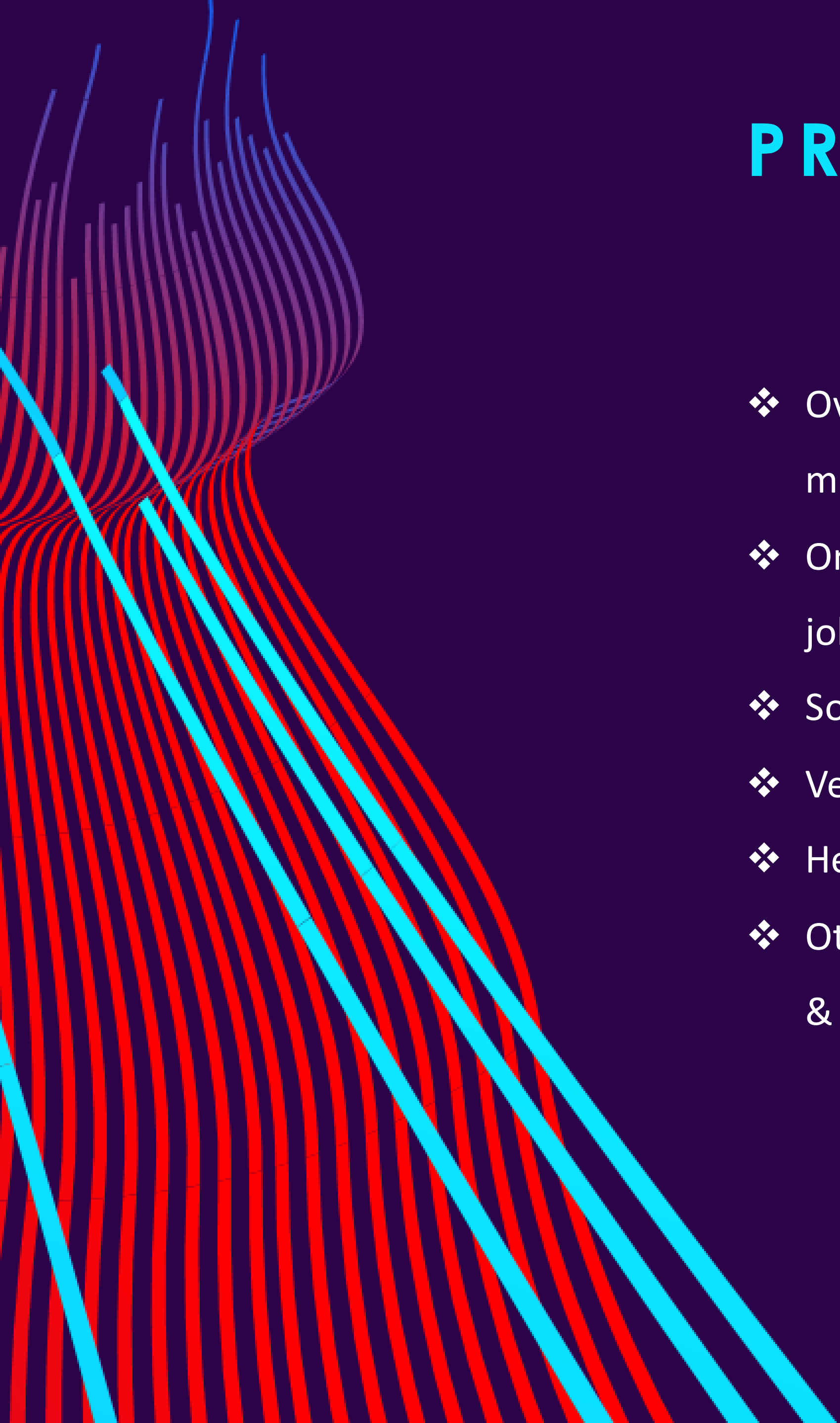
FROM DRONE TO GITHUB ACTIONS - A CI/CD ODYSSEY

~ Pavel Vařenka

REASONS, NEEDS & PAIN POINTS

- At GWI, we adhere to the "you build it, you run it" approach (as long as YAML is not actually involved!)
- We scaled & doubled our engineering rather fast
- Despite having reliable pipelines, the whole setup was long outdated and needed refactoring
- Constrained by compute power and reaching limits
- We accumulated some tech debt & struggled heavily with scaling our meager Drone VM
- Separate UI, rather annoying
- Drone license cost us approximately ~40k GBP per year

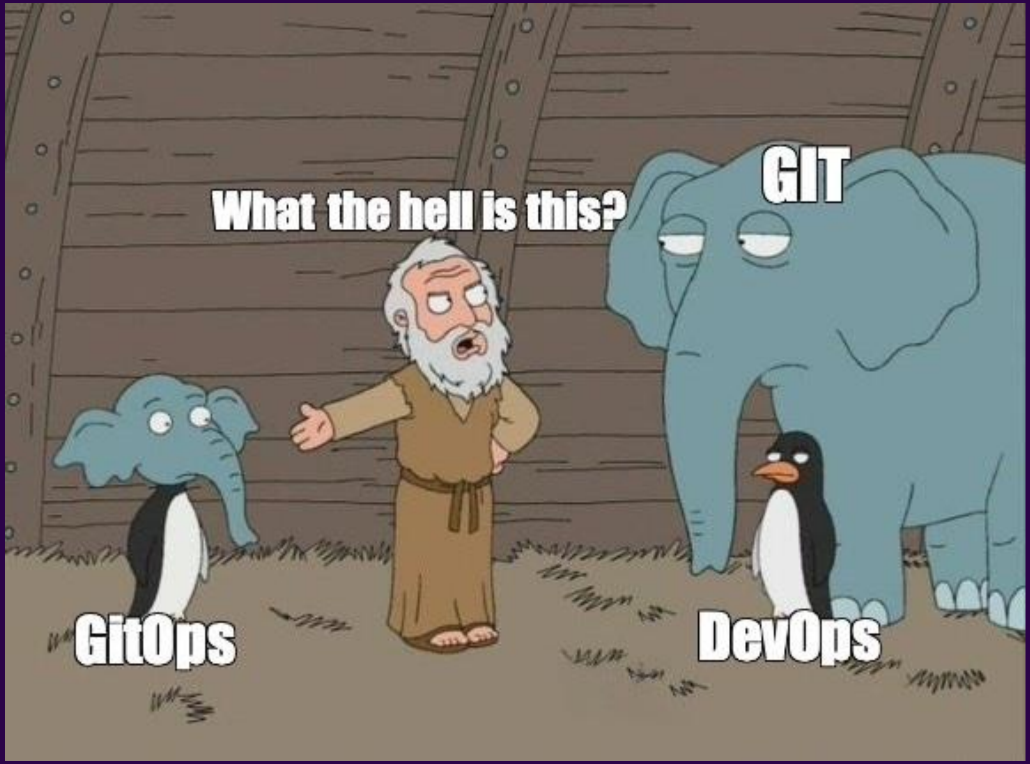


PREVIOUS SETUP

- Over 60 individual repositories ("microservices" in Scala, Go and Python + microfrontends), each with their our pipelines
- One dedicated large VM running all of our Drone pipelines (approximately 5 to 10 jobs per repo) couple of times per day with deployments once or twice a week
- Some of them long-running (~30 to 40 minutes), especially Scala builds
- Very limited manual triggers
- Helm deploy bash script for continuous delivery
- Otherwise quite typical use-case – linting, testing, artifact offload, image build, rinse & repeat

CHOOSING A SAVIOR

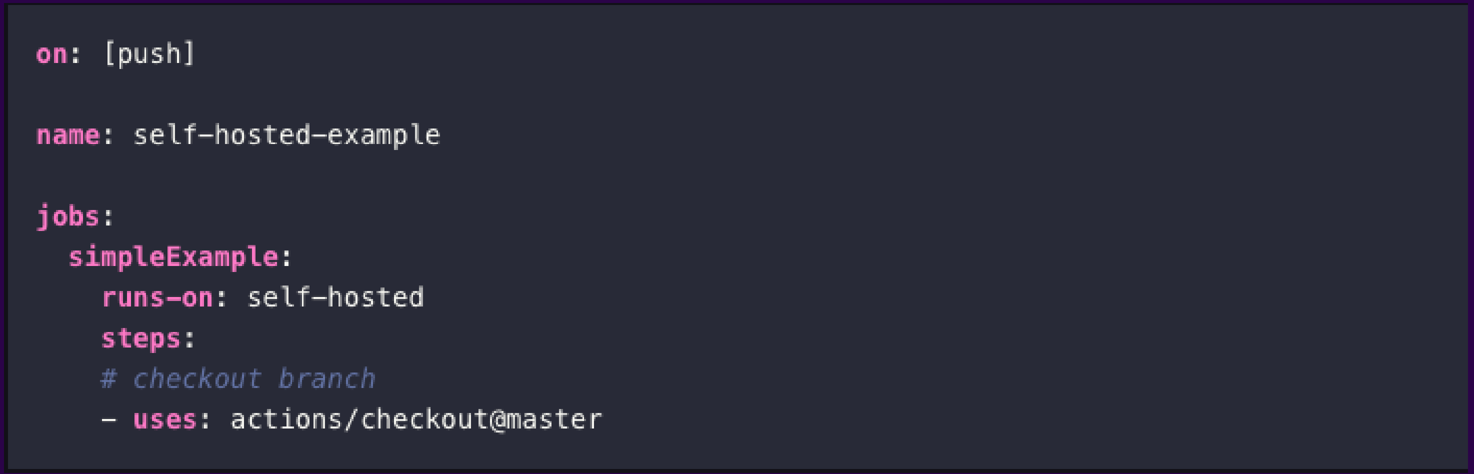
- We contemplated cost, ease of migration & syntax familiarity
- As we host our code in GitHub, the first candidate was obviously GitHub Actions
- The DevOps team performed some cost/benefit analysis and we realized that if we want to do this, it is going to be a year-long effort (ideally without downtime and code freezes)
- A new plan was born – gather the whole engineering, identify easiest repos to migrate, and start with the low-hanging fruit
- To do that, though, we needed to conquer our first beast – the infrastructure itself
- Naturally, the idea of GitOps came to mind as well, but that's a whole another CD story…


What the hell is this?
GIT
GitOps
DevOps

SLAYING THE DRAGON & GOING SELF-MANAGED IN KUBERNETES

- We burnt ourselves once – no more single VMs, it's a pain to manage and scale
- We decided to follow the best practices and went with two autoscaling nodepools – regular & mem-optimized
- Each Pod represented one job, and was scheduled based on the [*runs-on]* label, e.g., resource hungry Scala pipelines would be scheduled on a designated runner with more resources, basic Go pipelines (no offense, that's a good thing!) would utilize regular runner
- This gave us the flexibility to implement additional goodies, such as image caching, enhanced security & full control over scaling

- We deployed our GitHub runners to support DIND (Docker-in-Docker), although now they support Kubernetes mode as well
- We decided to use native ARC (Actions Runner Controller) open-sourced by GitHub
- The workflow is pretty simple – ARC would pick up GH event based on a webhook, check the runner group/set label, and route jobs accordingly, each one matching exactly one Pod
- If we needed more resources, we could simply bump the nodepool, or create a new one



```yaml
on: [push]

name: self-hosted-example

jobs:
  simpleExample:
    runs-on: self-hosted
    steps:
    # checkout branch
    - uses: actions/checkout@master
```

DOING THE GRUNT WORK

- So, how did we pull this off?
- First task – convince devs that they will really, really like it
- Second task – run a simple PoC and calculate costs
- … and then back to a drawing board

ENDLESS COMMITS, ENDLESS PIPELINES

- As you probably already know, testing pipelines is monumentally tedious effort
- Commit -> see why it failed -> fix -> rinse & repeat
- Naturally, we tried testing locally and can recommend one handy tool – ACT: https://github.com/nektos/act
- This, though, didn't yield in intended results – it's fine for simple pipelines and jobs that can be done in isolated environment, but not for multi-job pipelines relying on external auth
- We created a temporary env for testing and essentially split the pipelines into a few functional steps (one might call it unit testing)
- From there, it was a breeze – just piercing puzzles together. Sometimes by force :)

THE DEVIL IS IN THE DETAILS

- During the first few repositories, the DevOps team had to help out devs on a regular basis, but then it slowly turned into a mundane routine
- We overlooked one design aspect – one giant VM consolidated everything together, hence you could reference other steps easily everywhere in the pipeline. Built an image? Reference! Want to echo tags? Reference!
- Unfortunately, this is the part where things went wrong – devs were used to this perk, and we had to find a way to minimize the friction
- Using exclusively steps instead of jobs was pointless, as it unnecessarily increased runtimes
- Using too many jobs, on the other hand, resulted in bloated pipeline definitions and performance issues
- We decided on a suitable middle ground – templates and composite actions

ENTER THE COMPOSITE ACTIONS

- ❖ We decided to take advantage of something that Drone locked – composite actions & reusable templates
- ❖ We created a dedicated repository for hosting composite actions that we created along with devs for some common use cases with predefined inputs and outputs that can easily be referenced and reused
- ❖ On top of that, each dev repo contained a reusable workflow that allowed easier additions of new jobs
- ❖ Before further migration, we dedicated a lot of time (& painful testing) to this idea, which saved us a lot of time, as suddenly we had a frame of reference instead of blindly trying to replicate the same functionality in GHA
- ❖ Apart from 3rd party composite actions (e.g., for building images and adding SSH keys), we also managed to reuse our Helm deployment script in one of those actions
- ❖ This all sped up migration of other repos by ~40%

THE GOOD, THE BAD & THE UGLY

THE GOOD

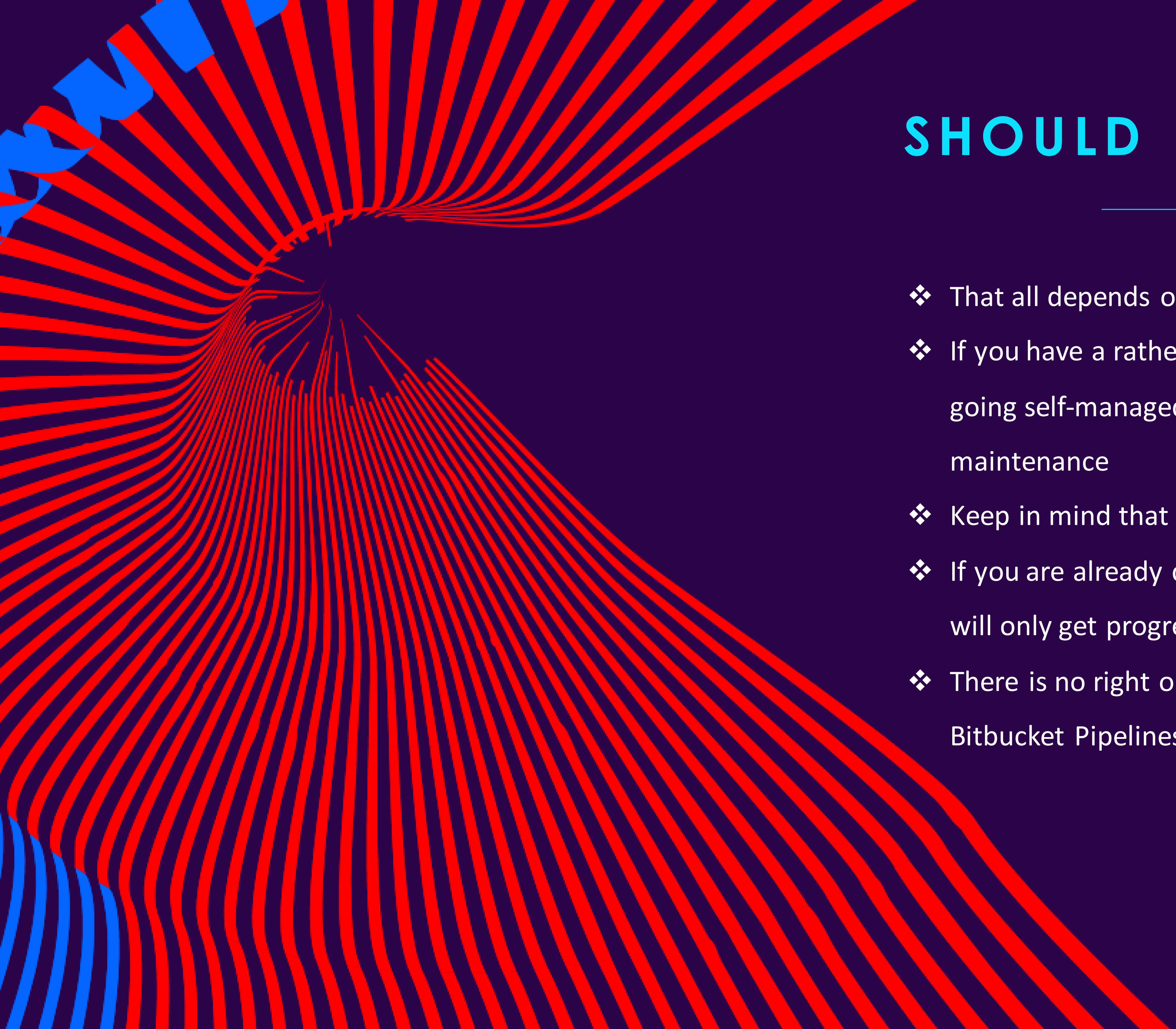
- Massive help from individual dev teams and owners of the repos, it would take us ages in 5 people
- Successfully distributing the migration toil among the engineering teams after initial onboarding
- Scaling advantage – autoscaling nodes, adding/removing nodepools, granular approach towards labeling (where and how the jobs will run)
- Improved DevEx, devs can now easily see the results in Actions tab
- Saved a lot of $$$ and got rid of the VM ~3 months after the migration
- Composite actions that will speed up development of new pipelines
- Easier cleanup and no more mess left behind on the VM
- Amazing cooperation across teams and sharing DevOps knowledge

THE BAD

- Some frustration regarding individual jobs – in the past, we have taken advantage of everything running on one VM, so devs could save artifacts/images/scripts/... there and reuse them as needed before final push
- Tedious testing and troubleshooting
- Due to some initial confusion and long-running stuck jobs, we somehow managed to break the GitHub API limit
- Some people tended to "abuse" the nature of autoscaling, e.g., they were not operating in constrained environment anymore
- Docker-in-Docker has some quirks, especially when adding SSH keys and/or auth
- Couple of dirty hacks in the beginning to make things working (yes, they are still there)

THE UGLY

- We could've handled the migration process a bit better
- There was a lot of back-and-forth between teams regarding running two pipelines simultaneously (Drone & GHA)
- We often forgot to remove the original YAML definition of Drone pipelines, resulting in duplicate (and frequently failing) pipelines
- Very hard to test deployment and a lot of conditional statements due to workflow_dispatch, a manual trigger than can be used only after being merged into main
- Too much effort going into reusable workflows, despite it paying off in the end
- No standardized process of when exactly we should merge the Actions pipeline definition, which caused some friction and side effects



SHOULD YOU MIGRATE?

- That all depends on your use-case
- If you have a rather small engineering, there is no point in going self-managed, as it requires some degree of maintenance
- Keep in mind that sooner or later, you will scale
- If you are already constrained by current CI/CD solution, it will only get progressively worse
- There is no right or wrong tool – apart from Jenkins and Bitbucket Pipelines

THANK YOU!

~ Pavel Vařenka

SCAN ME