stacc
moving finance forward

GitOps

"CIOps"

Let's say someone uses a hosted CI product like Bitbucket Pipelines and they want to deploy applications from CI to Kubernetes

"CIOps" the Anti-Pattern

- Give the CI access to the Kubernetes API (security risks)
- CI needs to be configured to the right cluster
- CI needs to have up-to-date credentials
- New cluster => re-configuring multiple pipelines

```
kubectl set image

helm upgrade
```

But what about In-Cluster "CIOps"

- Avoids some security issues, since the CI is inside the cluster
- The configration problem goes away

Other issues

- Need to have enough resources to run your builds
- Manage build history
- With multiple clusters images gets independently built per cluster. Not 100% identical image builds, and ruins the idea of container images. Builds needs to be truly reproducable.

Deployments failes

How to find correct version to rollback to?

- Trace your build logs
- Kubernetes workload revisions

What if the production cluster fails?

- How to tell what version of each app you need to deploy?
- Need to re-run all of your CI jobs for that cluster
- How to ensure the CI job will deploy to the right cluster?

GitOps in 1 slide

System development/managment pattern:

- Git as the **SINGLE** source of truth of a system
- Git as the **SINGLE** place where we operate (create, change and destroy) **ALL** environments
- **ALL changes are obervable/verifiable**

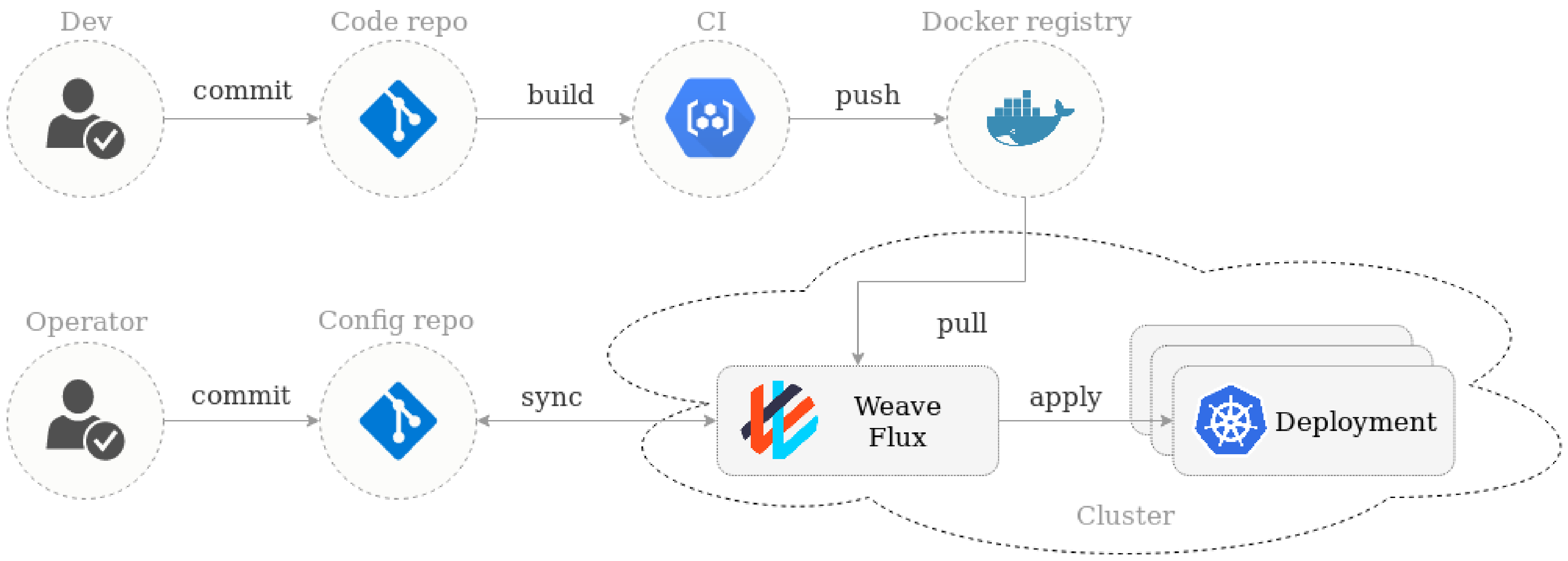
GitOps - Operations done through Git

- Git is one of the most common tool for source code managment
- GitOps is an operational and pipeline pattern that establishes the same consept not just for source code but for every operational aspect of your system
- All system and environments configuration is defined through Git, and the process of applying changes to the infrastructure follows the same practices and quality control procedures that a change in software does. Making it possible to review, check and promote across branches/environments any change that is made to a system.

IaC tools vs GitOps

- IaC originated the concept of keeping infrastructure config versioned, backed up and reproducible from source code
- Kubernetes is almost completely declarative, combined with the immutable container, it is possible to extend the concepts to manage applications and their operating system.
- GitOps philosophy is to test, deploy, rollback, rollforward with a complete audit trail all from Git
- IaC to provision servers, GitOps to manage Kubernetes

Weaveworks Flux - The GitOps Kubernetes operator



Options

- Argoproj Makes use of CRD to define pipelines. Cloud Native. GitOps.
- Jenkin X GitOps and runs on Kubernetes with a very powerfull CLI `jx`
- InfraBox CIOps that runs on Kubernetes with Docker jobs
- Jenkins CIOps with Kubernetes plugins
- GitLab - Auto DevOps Built-in container registry and Kubernetes support
- Spinnaker Multi-cloud continuos delivery platform
- Skaffold Tool for local and remote developing, but can be used in CI/CD.

Getting started with GitOps

Fork & clone the repository

```
git clone https://github.com/stacc-as/gitops-workshop
```

1. Install the flux operator on your cluster

- Install flux

```
cd gitops-tutorial
kubectl apply -f ./flux/
```

- Alongside flux, we have also installed a memcache service for flux to use, and appropriate permissions for the flux operator to act on the cluster itself through a role and rolebinding

```
kubectl get pods
NAME                                  READY    STATUS     RESTARTS    AGE
flux-7cddb59dd4-kftxn                 1/1      Running    0           15d
flux-helm-operator-7f8467486d-hx74w   1/1      Running    0           22d
```

2. Sharing the operator's public key with the remote git repository

- Download the fluxctl CLI

```
wget https://github.com/weaveworks/flux/releases/download/1.11.0/fluxctl_linux_
/usr/local/bin/fluxctl \
&& chmod +x /usr/local/bin/fluxctl
```

- And retrieve the public key from the operator

```
fluxctl identity
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCl05 [...]
```

- Or via kubectl

```
kubectl logs deployment/flux | grep identity.pub | cut -d '"' -f2
```

- Add the key to the remote repository

3. Configure the operator to read from your repository

- Edit the `flux/flux-deployment.yaml` and update the following values

```
# Configure git repository
--git-url=git@github.com:stacc-as/gitops-workshop.git
# Configure git branch
--git-branch=master
# Configure the path for flux to read
--git-path=deploy/kubernetes
# Configure the git user
--git-user=gitops-workshop
# Configure the git email
--git-email=flux@stacc.com
# Configure git sync label, must end with '-sync'
--git-label=flux-sync
```

- Reapply the configuration to the cluster

```
kubectl apply -f flux/deployment.yaml
```

4. Add a yaml file

The operator is set ut to react to changes in the `deploy/kubernetes` folder, we need to provide it with some configuration for it to synchronise

```
cp examples/podinfo-dep.yaml deploy/kubernetes
git add .
git commit -m "Add podinfo to cluster"
git push
```

5. Watch for the magic

The flux operator syncronize every 5 minutes, the `podinfo` deployment should be up and running after a little while

```
kubectl get pods -w
```

6. Lets fuck things up!

Wait for the flux to sync, it will detect the missing deployment and recreate it from the last good state

```
kubectl delete deployment podinfo

kubectl get deployments

kubectl get deployments -w
```

7. Modify the configuration

```
$ kubectl get deployment podinfo

NAME       DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
podinfo    1          1          1             1            11m
```

- Change the replicas from 1 to 4 in `deploy/kubernetes-podinfo-dep.yaml`
- Commit the changes
- Wait for flux to sync and watch changes take action

```
kubectl get deployment podinfo
```

8. Automate deployment

```
annotations:
  flux.weave.works/tag.podinfo: glob:*
  flux.weave.works/automated: 'true'
```

Questions?