Meltwater Engineering

Terraform, GitOps and Kubernetes

Lessons learned from Meltwater

About me

Hello, I’m Jim Sheldon!

Nice to meet you :)

Principal Software Engineer @ Meltwater

@bitberk

jimsheldon

Media Monitoring

Media Outreach

Social Listening

Social Publishing & Engagement

Social Influencer Management

About Meltwater

250,000,000,000+

Total documents stored

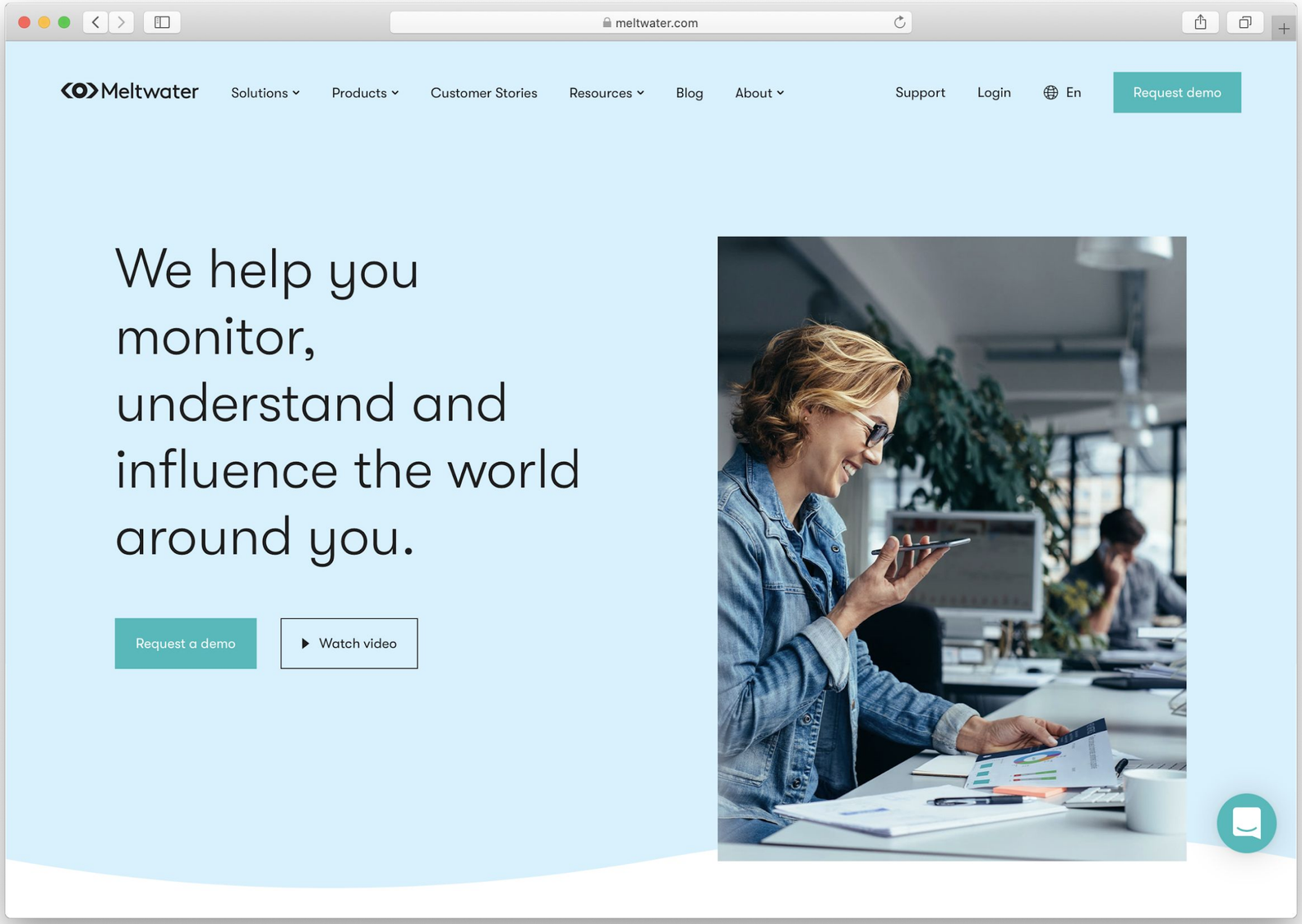
500,000,000+

New documents per day

4,000,000+

Searches executed per day

About Meltwater

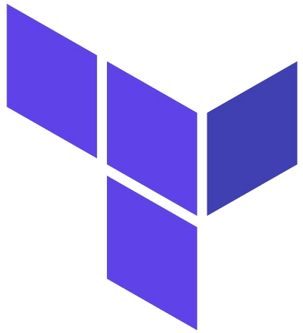


About Meltwater Engineering

Meltwater

"We enable teams to **accelerate** without thinking too much about the infrastructure, allowing a rapid path from **ideation** to **prototype** to providing **business value**"

Today's Topics



Meltwater's infrastructure as code choice

The source of truth for our world

Our container orchestration choice

Where did we start?

Initial Terraform commit

```
Fri Feb 19 13:16:31 2016 +0100
```

Terraform - the beginning

```
meltwater-terraform
├── README.md
├── modules
│   ├── authorization
│   ├── datadog
│   ├── dns
│   ├── globals
│   └── vpc
├── team1-aws
│   ├── development
│   ├── production
│   └── staging
└── team2-aws
    ├── development
    ├── production
    └── staging
```

- Single repository for every team's Terraform configuration
- Modules symlinked via relative path into each directory
- Webhooks for every team triggered Jenkins jobs for each team

Terraform - single repo, directory based

```
terraform-config-team1
├── README.md
└── aws
    ├── dev
    ├── prod
    └── staging
```

Terraform - single repo, directory based

```
├── dev
│   ├── eu-west-1
│   │   ├── default
│   │   │   ├── main.tf
│   │   │   ├── terraform.tf
│   │   │   └── .terraform.version
│   │   ├── vpc1
│   │   └── project
│   └── us-east-1
│       └── default
└── prod
    ├── eu-west-1
    │   ├── default
    │   └── vpc1
    └── us-east-1
        └── default
```

terraform.tf

```
terraform {
  backend "s3" {
    bucket = "tf-state-mw-myteam-dev"
    key    = "eu-west-1/default.tfstate"
    region = "eu-west-1"
  }
}
```

Terraform - multiple repos

```
terraform-config-team1-dev
├── README.md
└── aws
    ├── global
    ├── eu-west-1
    └── us-east-1
```

Development

```
terraform-config-team1-staging
├── README.md
└── aws
    ├── global
    ├── eu-west-1
    └── us-east-1
```

Staging

```
terraform-config-team1-prod
├── README.md
└── aws
    ├── global
    ├── eu-west-1
    └── us-east-1
```

Production

Terraform - branch based

```
terraform-config-team1
├── README.md
├── dev.tfvars
├── staging.tfvars
├── prod.tfvars
└── aws
    ├── global
    ├── eu-west-1
    └── us-east-1
```

development branch

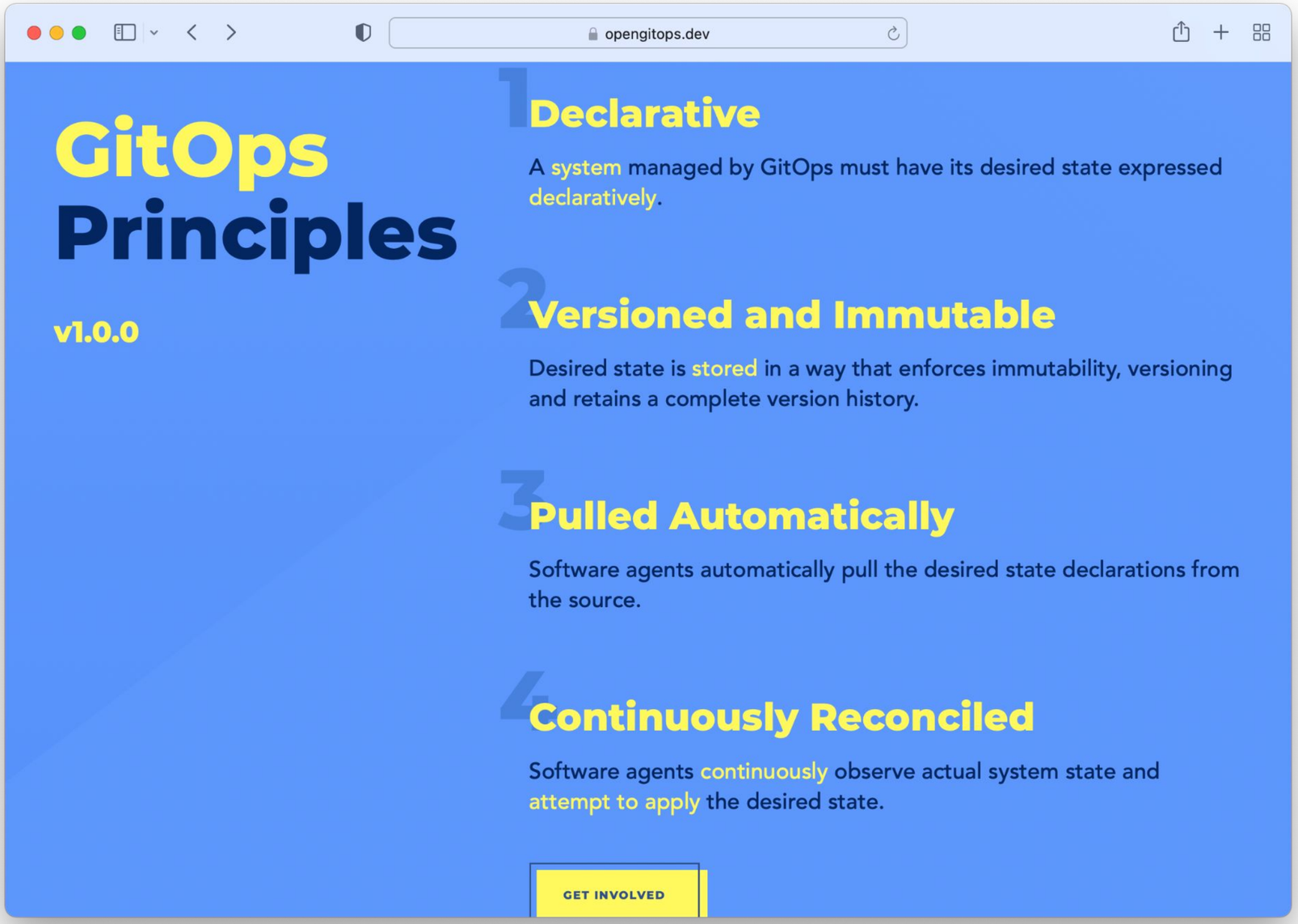
```
terraform-config-team1
├── README.md
├── dev.tfvars
├── staging.tfvars
├── prod.tfvars
└── aws
    ├── global
    ├── eu-west-1
    └── us-east-1
```

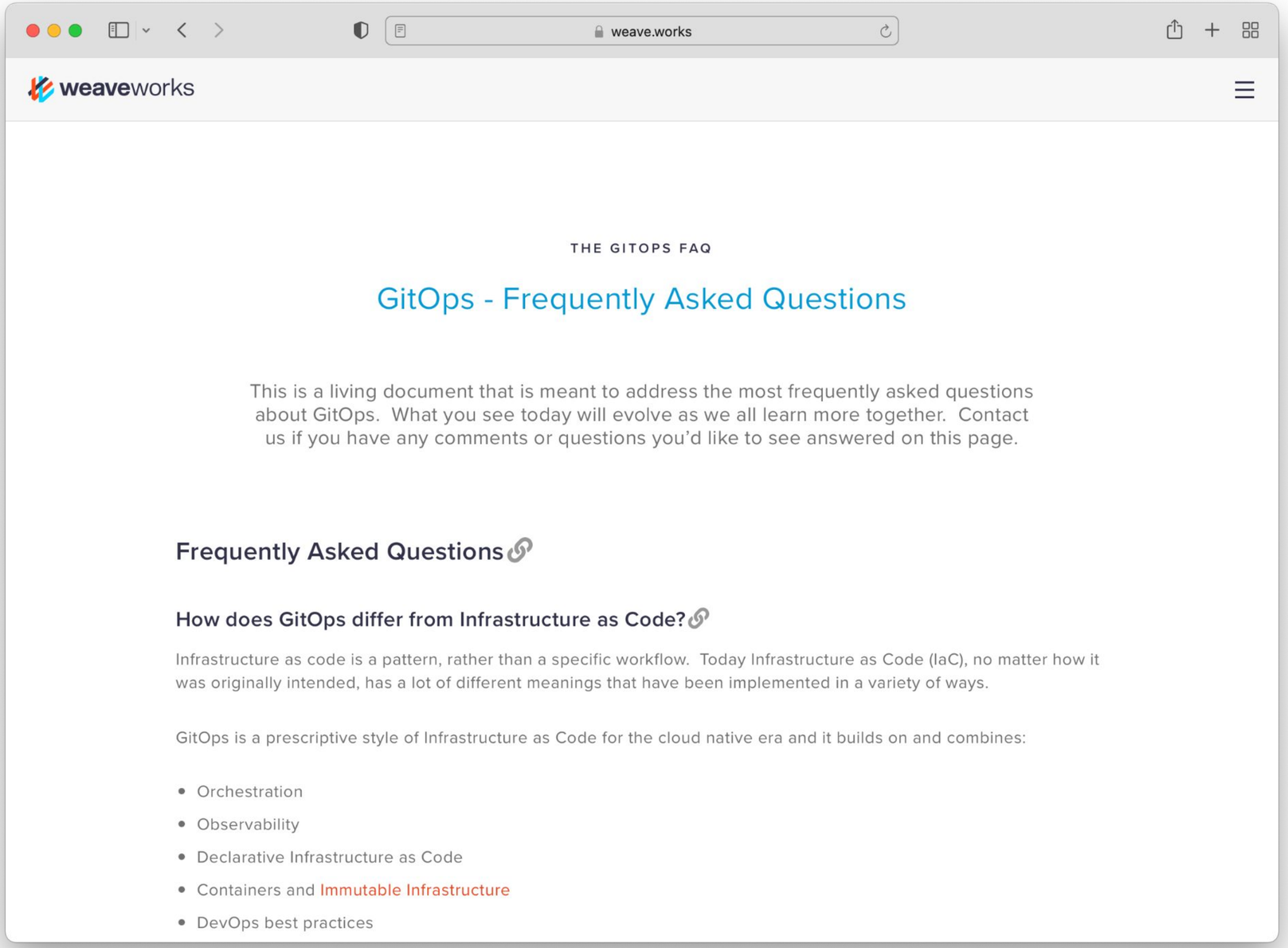
staging branch

```
terraform-config-team1
├── README.md
├── dev.tfvars
├── staging.tfvars
├── prod.tfvars
└── aws
    ├── global
    ├── eu-west-1
    └── us-east-1
```

production branch

GitOps?


opengitops.dev
GitOps Principles
v1.0.0
1 Declarative
A system managed by GitOps must have its desired state expressed declaratively.
2 Versioned and Immutable
Desired state is stored in a way that enforces immutability, versioning and retains a complete version history.
3 Pulled Automatically
Software agents automatically pull the desired state declarations from the source.
4 Continuously Reconciled
Software agents continuously observe actual system state and attempt to apply the desired state.
GET INVOLVED


weave.works
weaveworks
THE GITOPS FAQ
GitOps - Frequently Asked Questions
This is a living document that is meant to address the most frequently asked questions about GitOps. What you see today will evolve as we all learn more together. Contact us if you have any comments or questions you'd like to see answered on this page.
Frequently Asked Questions
How does GitOps differ from Infrastructure as Code?
Infrastructure as code is a pattern, rather than a specific workflow. Today Infrastructure as Code (IaC), no matter how it was originally intended, has a lot of different meanings that have been implemented in a variety of ways.
GitOps is a prescriptive style of Infrastructure as Code for the cloud native era and it builds on and combines:
Orchestration
Observability
Declarative Infrastructure as Code
Containers and Immutable Infrastructure
DevOps best practices

Is Meltwater doing GitOps?

“Software agents continuously observe actual system state and attempt to apply the desired state.”

- In EC2...
  - We make heavy use of Autoscaling Groups with health checks
  - During deployments, the first instance in the new ASG must become “InService” before the ELB will send traffic to it
  - If an instance fails checks, the ASG handles it for us
  - If an ELB has unhealthy hosts, we are alerted
  - All processes have proper systemd service definitions
  - We use Puppet to verify state at the OS level

Is Meltwater doing GitOps?

“Software agents continuously observe actual system state and attempt to apply the desired state.”

- In Kubernetes...
  - We make heavy use of Liveness/Readiness probes
  - Every team has their own namespace, they can’t manipulate other namespaces
  - The team agrees that the CD pipeline is responsible for ‘kubectl apply’, manually running ‘kubectl’ for changes is ***forbidden!***

Is Meltwater doing GitOps?

“Software agents continuously observe actual system state and attempt to apply the desired state.”

- We use Prometheus to continuously observe our infrastructure and services
  - All of our services are continuously probed with the Blackbox exporter, increased latency and/or failures will alert us

Is Meltwater doing GitOps?

“Software agents continuously observe actual system state and attempt to apply the desired state.”

- Unexpected changes seen in plan output from pull requests in our Terraform configuration are a ***big deal!***

Is Meltwater doing GitOps?

Alerts

P3

x1

#70725

Oct 6, 2021 4:50 PM (GMT-04:00) • Jim Sheldon

Last apply in terraform-config-foundation FAILED

autoclose_30min critical drone production silenced terraform

Meltwater

Are we doing GitOps?

Kubernetes config repo structure

```
k8s-config-team1
├── README.md
├── development
│   ├── grafana
│   ├── mysqld-exporter
│   ├── nginx-ingress-controller
│   ├── app1
│   └── app2
└── production
    ├── grafana
    ├── mysqld-exporter
    ├── nginx-ingress-controller
    └── app1
```

- Very similar workflow to our separate infrastructure Terraform repository
  - We can write Terraform modules for our k8s configs
  - We can reuse the same tooling from our infrastructure pipelines

Kubernetes Terraform provider

HCL

```
resource "kubernetes_deployment" "example" {
  metadata {
    name = "terraform-example"
    labels = {
      test = "MyExampleApp"
    }
  }

  spec {
    replicas = 3

    selector {
      match_labels = {
        test = "MyExampleApp"
      }
    }
  ...
}
```

YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: terraform-example
  labels:
    test: MyExampleApp
spec:
  replicas: 3
  selector:
    matchLabels:
      test: MyExampleApp
...
```

Meltwater

Kubernetes verify deployment

```
resource "local_file" "verify_script" {
  filename        = "verify.sh"
  file_permission = "0755"

  content = <<EOF
#!/usr/bin/env bash -x
kubectl rollout status --watch=true --timeout=5m deployment/terraform-example
EOF

  provisioner "local-exec" {
    command = "./verify.sh"
  }

  depends_on = [
    kubernetes_deployment.example
  ]
}
```

Kubernetes verify deployment

```
local_file.verify_script: Creating...
local_file.verify_script: Provisioning with 'local-exec'...
local_file.verify_script (local-exec): Executing: ["/bin/sh" "-c" "./verify.sh"]
local_file.verify_script (local-exec): + kubectl rollout status --watch=true --timeout=5m deployment/terraform-example
local_file.verify_script (local-exec): deployment "terraform-example" successfully rolled out
local_file.verify_script: Creation complete after 1s [id=8c8baa146f3398285c9eeffed2e7c1445020a481]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Terraform destroy through apply

```
development/app1
├── locals.tf
├── main.tf
└── terraform.tf
```

--->

```
development/app1
├── locals.tf
├── main.tf_
└── terraform.tf
```

```
Apply complete! Resources: 0 added, 0 changed, 2 destroyed.
```

terraform apply

underthehood.meltwater.com

under the hood

< The official meltwater engineering blog

BLOG | ABOUT | JOBS | OPEN SOURCE | ARCHIVES

October 20, 2021 | 0 Comments

Spotlight: Patrick Bardo

At the heart of Meltwater Engineering are the employees who find ways to collaborate around the world. In our Spotlight series we will introduce you to new hires as well as veterans from across the organization, and give you insights into their day to day from North Carolina to Hong Kong.

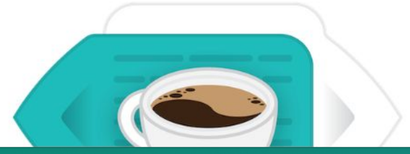
Today we are happy to introduce Patrick Bardo from our Gothenburg office.

Read on

August 03, 2021 | 0 Comments

Reinventing the Water Cooler Conversation

In the midst of the pandemic and months into working from home, Hardik Gupta, a data scientist at Meltwater, was missing the social connection of going into work, specifically the spontaneous conversations with people from other business functions and teams. He decided to experiment with emulating these



About Us

We are the engineers of Meltwater. Find our open source projects at GitHub. Here we write about the things we do.

Recent Posts

Spotlight: Patrick Bardo

Reinventing the Water Cooler Conversation

Maturing as an Agile Coach

Migrating DynamoDB between AWS accounts using AWS Glue

Meltwater on the New Stack Makers Podcast

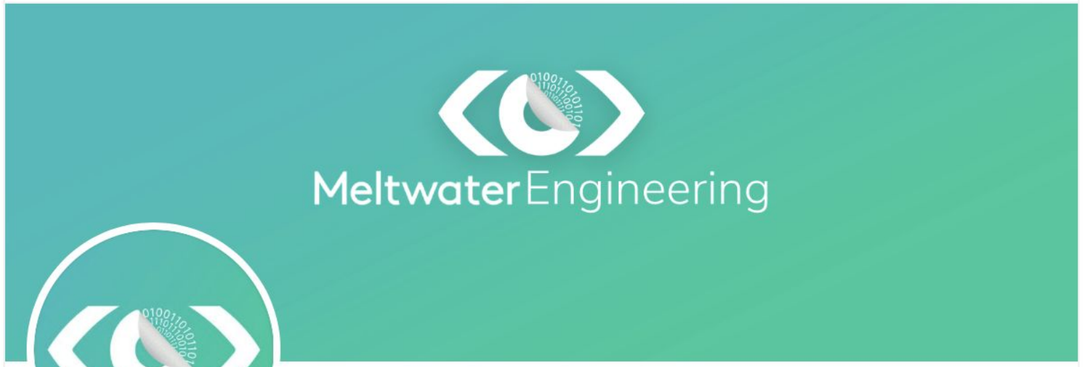
GitHub Repos

.github

Meltwater's community health files

addict

User management lib for Phoenix Framework



Follow

Meltwater Engineering
@MeltwaterEng

Stories from the Meltwater Product & Engineering team. We talk about the work we do, tech, doing good, and other things we are passionate about.

World · underthehood.meltwater.com · Joined May 2019

99 Following **148** Followers

Tweets | Tweets & replies | Media | Likes

Meltwater Engineering @MeltwaterEng · 1h

🔦 Our #Spotlights are back on #Underthehood

First up is Patrick Bardo, software engineer from Gothenburg 🇸🇪, who originated from Kitchener, Ontario 🇨🇦. Check out the blog post to get to know Patrick and what he does at @MeltwaterEng.

underthehood.meltwater.com/blog/2021/10/2...

Meltwater Engineering

Thank you!