Continuous Deployment with GitHub Actions and Kamal

Posted on January 7, 2024 • 1 minute read

Kamal is a wonderfully simple way to deploy your applications anywhere. It will also be included by default in Rails 8. Kamal is trivial, but I don't recommend using it on your development machine.

From experience working on an oldish laptop, I can tell you that building Docker images locally is not fun. Also, why would you, when GitHub Actions are for free!

In this post, I'll show you how to build a simple CI pipeline with Kamal. We'll create an application image and deploy it on every push. We'll also add some simple image caching to speed up the workflow.

The Complete Workflow

This is what we'll end up with at the end of this post.

```yaml
name: Deploy

on:
  push:
    branches:
      - main

jobs:
  Deploy:
    if: ${{ github.event_name == 'push' && github.ref == 'refs/heads/m
```

```yaml
    runs-on: ubuntu-latest

    env:
      DOCKER_BUILDKIT: 1
      RAILS_ENV: production

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Docker Buildx
        id: buildx
        uses: docker/setup-buildx-action@v2

      - name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: hschne
          password: ${{ secrets.DOCKER_REGISTRY_KEY }}

      - name: Set Tag
        id: tag
        run: |
          echo "tag=$(git rev-parse "$GITHUB_SHA")" >> $GITHUB_OUTPUT

      - name: Build image
        uses: docker/build-push-action@v5
        with:
          context: .
          builder: ${{ steps.buildx.outputs.name }}
          push: true
          labels: |
            "service=anonymous-location"
          tags: |
            "hschne/anonymous-location:latest"
```

```
          "hschne/anonymous-location:${{ steps.tag.outputs.tag }}"
        cache-from: type=gha
        cache-to: type=gha,mode=max

    - uses: webfactory/ssh-agent@v0.7.0
      with:
        ssh-private-key: ${{ secrets.SSH_PRIVATE_KEY }}

    - name: Set up Ruby
      uses: ruby/setup-ruby@v1
      with:
        bundler-cache: true

    - name: Deploy command
      run: bundle exec kamal deploy --skip-push
      env:
        RAILS_MASTER_KEY: ${{ secrets.RAILS_MASTER_KEY }}
        KAMAL_REGISTRY_PASSWORD: ${{ secrets.DOCKER_REGISTRY_KEY }}
```

That's something. Well, nobody ever said GitHub actions are succinct. Let's look at what's going on here step by step.

Step By Step

The workflow above is based on the one in this post. Unfortunately, that post is so old that it still refers to Kamal as Mrsk. I had to make some adjustments to how the image built and deployed.

> I won't go into details on GitHub Actions specifics. If you're new to GitHub Actions or unfamiliar with one particular piece of syntax, I recommend you check out the documentation.

To build and deploy our Rails application, we need to provide GitHub actions with three

secrets:

- `RAILS_MASTER_KEY` : The master key to your Rails application credentials.
- `DOCKER_REGISTRY_KEY` : The API token for pushing and pulling from your container registry.
- `SSH_PRVIATE_KEY` : The SSH key for accessing the server where your app is deployed.

We'll also need to set some environment variables. We are building a production image. We'll also need to instruct the Docker build step to use Docker Buildkit, as that is one of the requirements of Kamal.

```
env:
  DOCKER_BUILDKIT: 1
  RAILS_ENV: production
```

Our deployment workflow needs to do a couple of things. First, we need to check out the application source code. Next, we'll log into Docker Hub to push our image.

```
- name: Checkout code
  uses: actions/checkout@v3

- name: Set up Docker Buildx
  id: buildx
  uses: docker/setup-buildx-action@v2

- name: Login to Docker Hub
  uses: docker/login-action@v3
  with:
    username: hschne
    password: ${{ secrets.DOCKER_REGISTRY_KEY }}
```

Kamal uses the git hash of the latest commit to determine which image to deploy, so image tags must match git commit hashes. We define this tag with a separate workflow step.

We use the docker/build-push-action to build the application image. In addition to setting the correct tag, the image build step must also provide a label matching your service name. Because the image should be pushed to your container registry, we set `push: true`, and because we want ludicrous build speed we instruct the build step to utilize the GitHub Actions cache.

```yaml
- name: Set Tag
  id: tag
  run: |
    echo "tag=$(git rev-parse "$GITHUB_SHA")" >> $GITHUB_OUTPUT

- name: Build image
  uses: docker/build-push-action@v5
  with:
    context: .
    builder: ${{ steps.buildx.outputs.name }}
    push: true
    labels: |
      "service=service-name"
    tags: |
      "user/image-name:latest"
      "user/image-name:${{ steps.tag.outputs.tag }}"
    cache-from: type=gha
    cache-to: type=gha,mode=max
```

Once the image has been built and pushed, you only need to trigger the deployment using Kamal. We use the webfactory/ssh-agent to establish a connection to our production server. After installing the required Ruby dependencies, it's only a matter of running Kamal. As the image is already built and pushed, we use the `--skip-push` flag.

```yaml
- uses: webfactory/ssh-agent@v0.7.0
  with:
    ssh-private-key: ${{ secrets.SSH_PRIVATE_KEY }}
```

```
    - name: Set up Ruby
      uses: ruby/setup-ruby@v1
      with:
        bundler-cache: true
    - name: Deploy command
      run: bundle exec kamal deploy --skip-push
```

And that's it! If you've enjoyed this post or have any other tips on how to use Kamal together with GitHub Actions, let me know! 🙂