
Архитектура распределённых приложений

Расчетно-графическая работа

Выполнил: студент группы ИП-312
Дорогин Н. С.
Проверил: преподаватель Кафедры вычислительных систем
Челканова Т.В.

Общая часть:

1. Реализовать CI и CD на базе репозитория Gitlab.
2. Продемонстрировать работу CI/CD, показать обновление ресурсов и приложения/приложений в кластере.
3. Файлы конфигурации/ресурсов k8s разместить в отдельной директории репозитория.

Задание на оценку 5:

1. Доработать в CI этапы `diff` и `deploy`. Для развертывания использовать Helmfile.
2. Создать чарт для развертывания приложения. Привести значения по умолчанию в соответствии с именем проекта.
3. Тег собранного ранее в CI образа передавать через аргументы командной строки.
4. Используя механизм окружений (`environment`) Helmfile настроить CI таким образом, чтобы разворачивалось два варианта окружения приложений. Все окружения должны работать одновременно. Должна быть возможность запустить деплой каждого из окружений независимо друг от друга. Приложения должны иметь уникальный url и, например, различные значения переменных окружения.

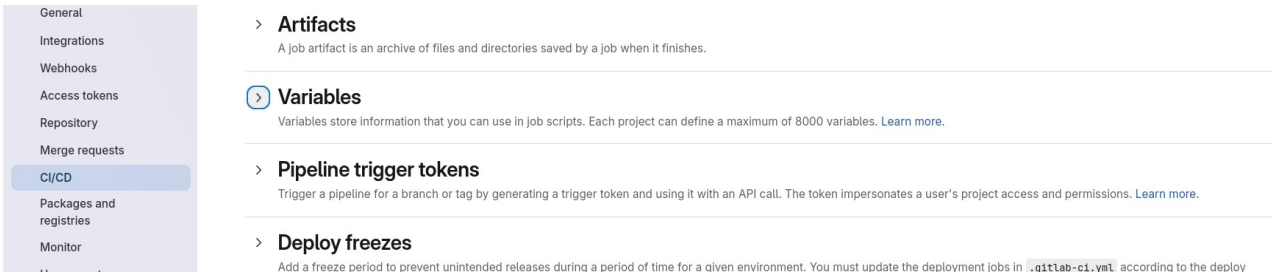
5. Этап deploy должен запускатся по нажатию кнопки.
6. Переменные определенные в `environment` не должны повторять структуру `values`. Файлы содержащие `environment`-переменные не должны использоваться в секциях `values`.

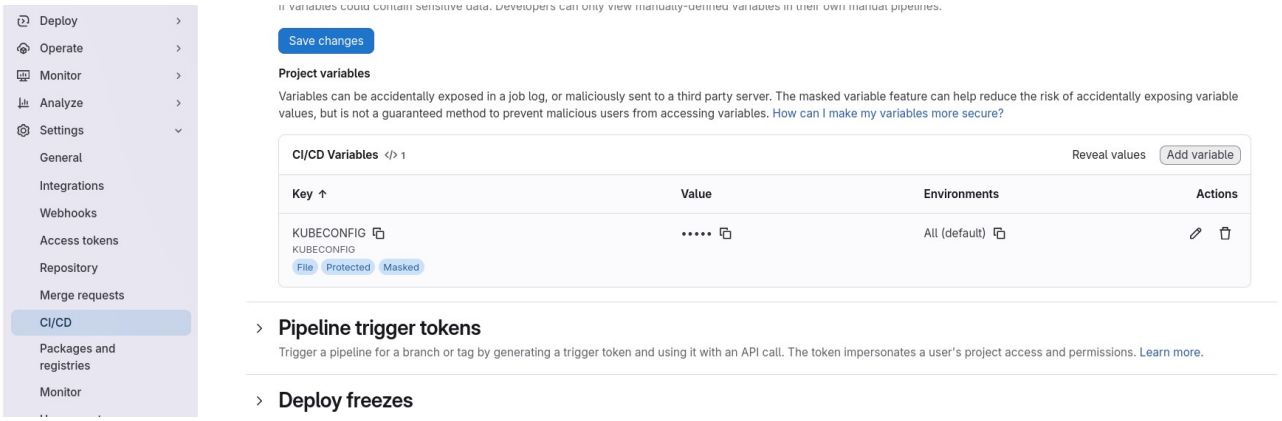
Ход работы:

Общая часть:

1) Для реализации CI/CD на базе нашего Gitlab-репозитория нам необходимо, чтобы из него мы имели доступ к кластеру Hawk.

С этой целью мы создадим в нашем репозитории специальную переменную типа File, которая будет хранить в себе, конфигурацию, предоставляющую ему доступ.





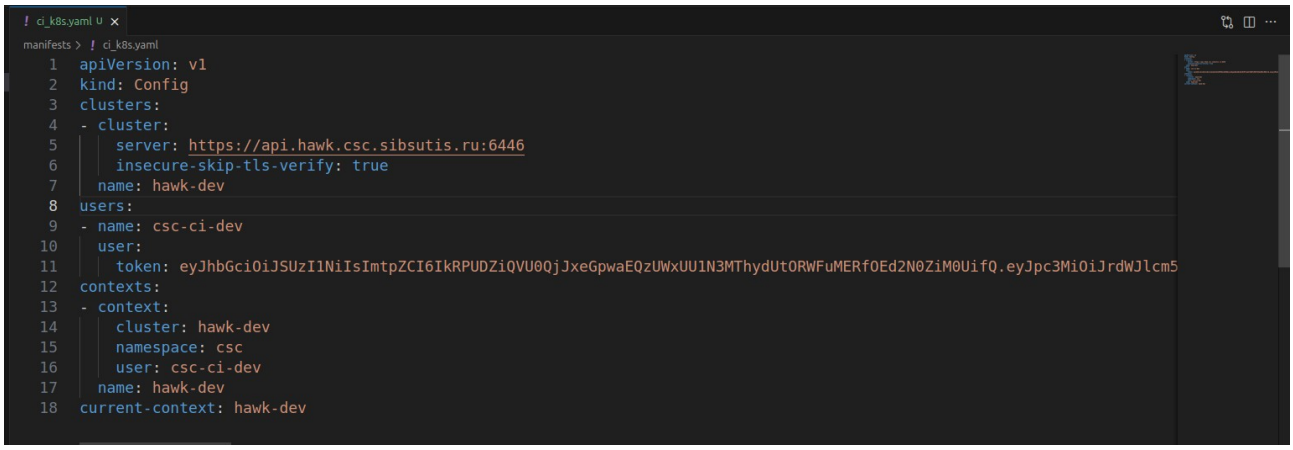
Содержание файла конфигурации берём в wiki кластера, которую мы использовали в лабораторной работе №3.

(https://wiki.csc.sibsutis.ru/en/Hawk)

Файл конфигурации имеет следующий вид.

(upd: не надо создавать этот файл в проекте и коммитить его, текст конфига надо вставить в value переменной в самом гитлабе, похже ещё раз об этом упомянуто будет)

ci_k8s.yaml



Для заполнения поля token нам нужно было получить специальный токен для CI.

Делается это с помощью следующей команды внутри кластера (можно на той же wiki скопировать):

Теперь у нашего Gitlab репозитория будет доступ к кластеру Hawk.

Более того использование утилиты kubectl по умолчанию будет относиться именно к кластеру Hawk.

Выполнение тех команд, которые мы запускали вручную внутри кластера в третьей лабораторной при каждом изменении образа, можно будет автоматизировать прописав их один раз CI.

Так же нам теперь не придётся переписывать манифесты в самом кластере — kubectl будет использовать манифесты, которые прописаны у нас в репозитории.

Теперь нужно обеспечить обновление образа, который используется в нашем основном манифесте my_Deployment.yaml. Этот образ нужно сделать переменной.

Сам по себе манифест не поддерживает использование переменных, зато их поддерживают шаблоны.

my_Deployment.template.yaml

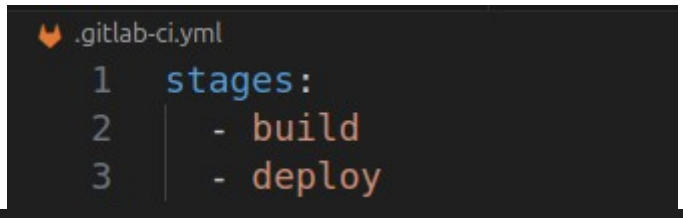
```yaml
apiVersion: apps/v1
kind: Deployment
...
              cpu: 64m
              memory: 32Mi
          readinessProbe:
            httpGet:
              path: /
              port: 5000
            failureThreshold: 3
            initialDelaySeconds: 15
            periodSeconds: 10
          livenessProbe:
            httpGet:
              path: /
              port: 5000
            failureThreshold: 3
            initialDelaySeconds: 15
            periodSeconds: 10
          startupProbe:
            httpGet:
              path: /
              port: 5000
            failureThreshold: 3
            initialDelaySeconds: 15
            periodSeconds: 10
      containers:
      - name: ip312s07-table
        image: ${DEPLOYMENT_IMAGE}
        imagePullPolicy: IfNotPresent
        command: ["dotnet", "Backend.dll"]
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop: ["ALL"]
        resources:
          requests:
            cpu: 32m
            memory: 16Mi
          limits:
```

Также нужно создать просто my_Deployment.yaml, чтобы манифест, получившийся в результате подстановки образа в шаблон было чему присвоить.

Изменение образа в главном манифесте готово. Любые другие изменения, вносимые нами в манифестах в репозитории также будут сразу же отражаться на развертывании приложения, потому что, как уже было сказано, используются именно манифесты из репозитория.

Теперь для обновления приложения нам нужно сделать так, чтобы изменённые манифесты были сразу же применены.

Для этого в CI добавляем этап deploy:



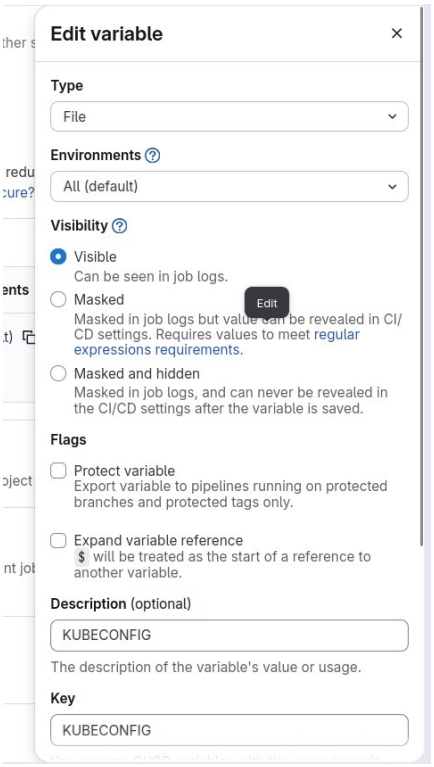

```yaml
deploy-job:
  stage: deploy
  image: registry.csc.sibsutis.ru/csc-public/internal-container-images/k8s-tools:latest
  before_script:
    - export KUBECONFIG=manifests/ci_k8s.yaml
    - apt-get update && apt-get install -y gettext-base
  script:
    - echo "START DEPLOYING..."
    - |
      if [ -n "$CI_COMMIT_TAG" ]; then
        IMAGE_TAG=$CI_COMMIT_TAG
        else
        IMAGE_TAG=${CI_COMMIT_SHORT_SHA:0:8}
        fi
        echo "READING THE IMAGE TAG"
    - export DEPLOYMENT_IMAGE=${HAWK_REGISTRY_IMAGE}:${IMAGE_TAG}
    - envsubst < manifests/my_Deployment.template.yaml > manifests/my_Deployment.yaml

    - echo "IMAGE WAS SUCCESFULLY REPLACED TO ${HAWK_REGISTRY_IMAGE}:${IMAGE_TAG}"

    - kubectl -n ip312 apply -f manifests/my_Ingress.yaml
    - kubectl -n ip312 apply -f manifests/my_Service.yaml
    - kubectl -n ip312 apply -f manifests/my_Deployment.yaml

    - echo "CHANGES WERE SUCCESFULLY APPLIED"
  rules:
  - when: manual
```

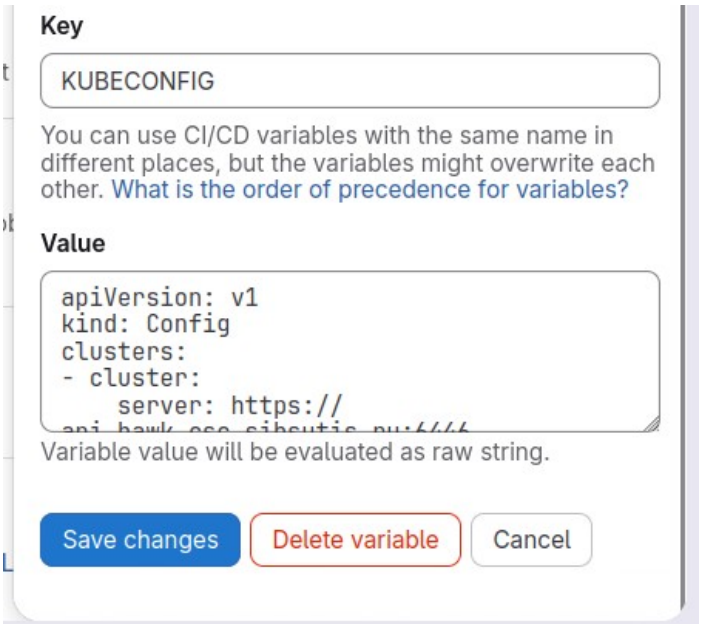
Присваиваем *KUBECONFIG* наш файл с конфигурацией и теперь команды kubectl по умолчанию действуют на кластер Hawk.

(upd:

строчка — - export KUBECONFIG=manifests/ci_k8s.yaml лишняя.

Вместо неё надо зайти на гитлабе в нашу переменную KUBECONFIG и сделать следующие настройки:





(текст конфига надо вставить в значение переменной, а файл конфига в проекте нам не нужен, его надо удалить в целях безопасности)

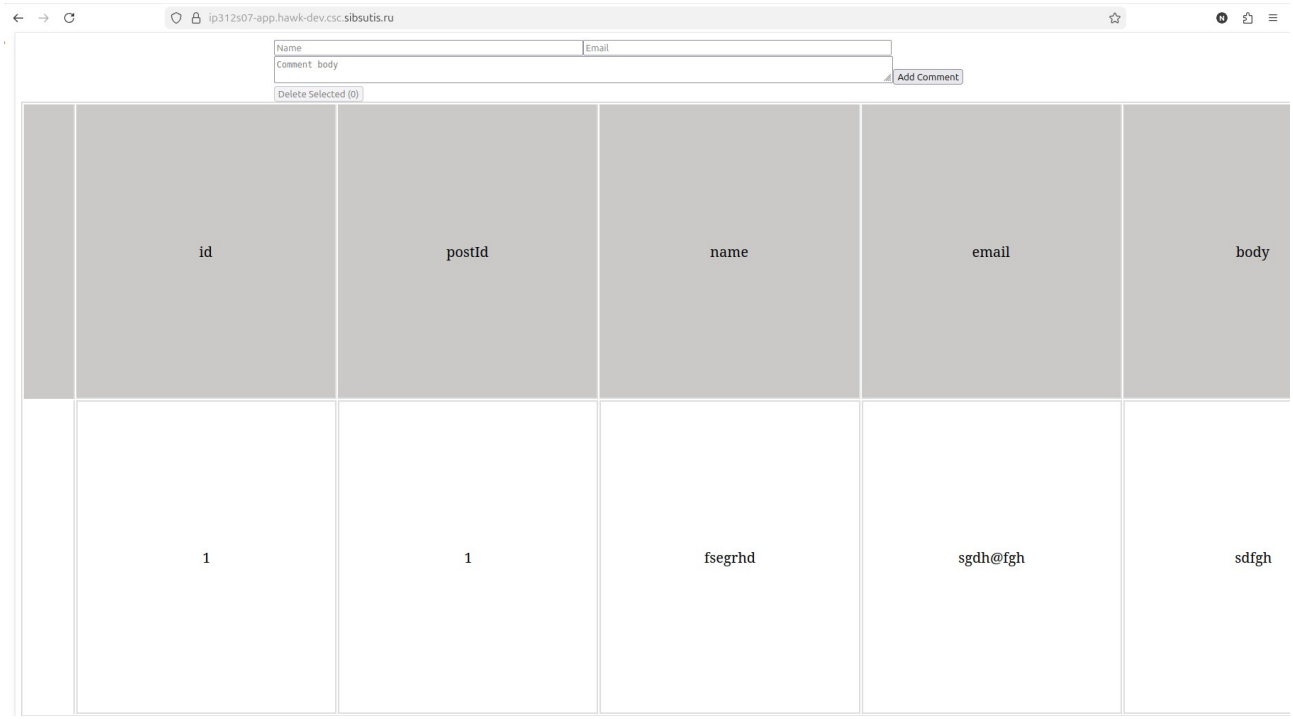
С помощью *envsubst* из *gettext-base* и присвоенного переменной *DEPLOYMENT_IMAGE* имени нашего нового образа мы из шаблона my_Deployment.template.yaml заполняем манифест my_Deployment.yaml.

Применяем манифесты.

2) Продемонстрируем работу CI/CD.

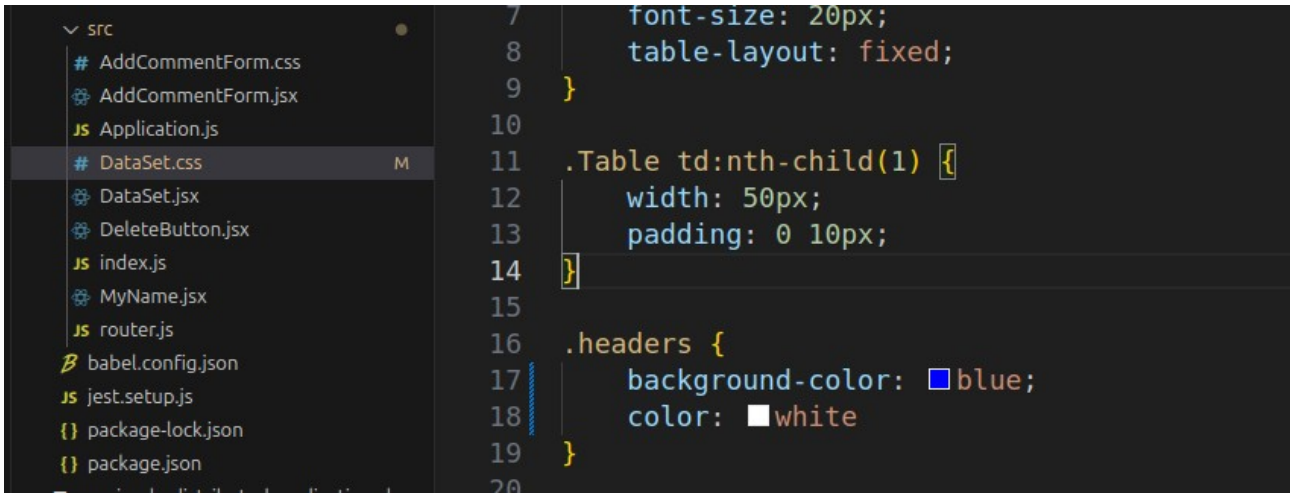
Для этого попробуем внести в наше приложение некоторые изменения.

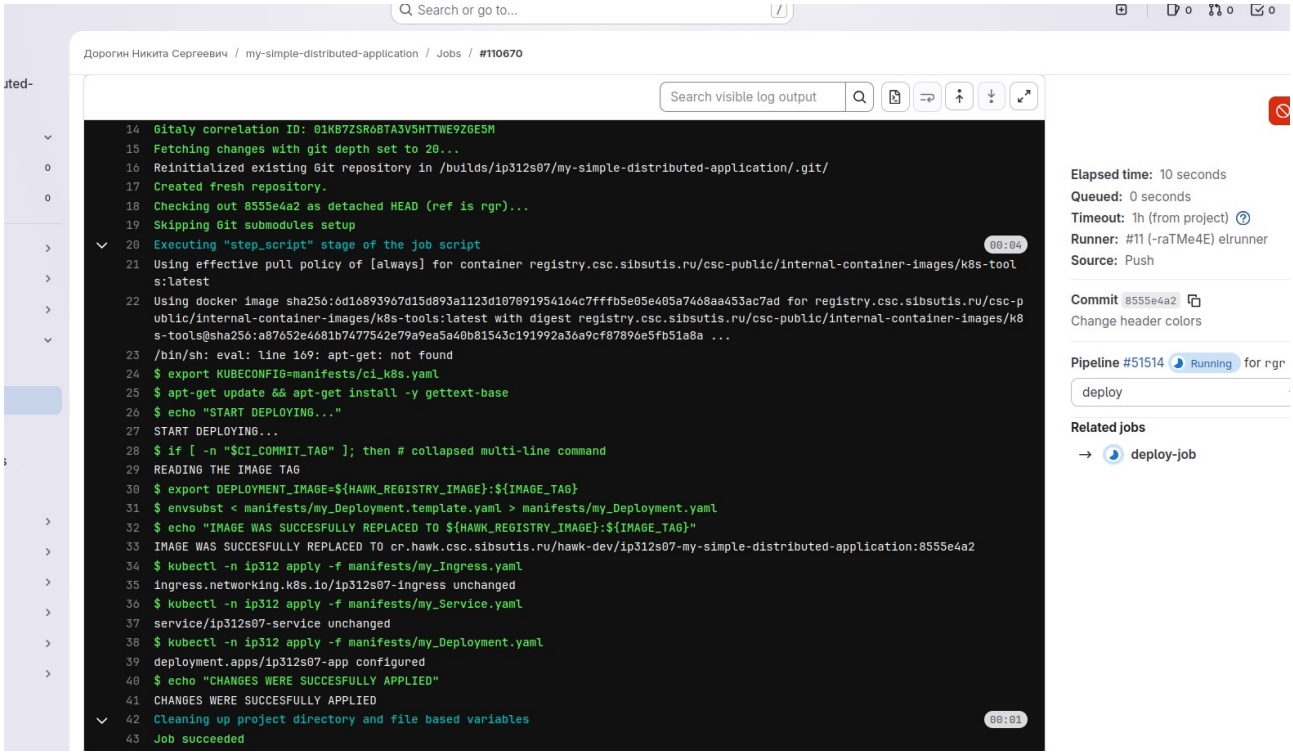
Вот как оно выглядело раньше:

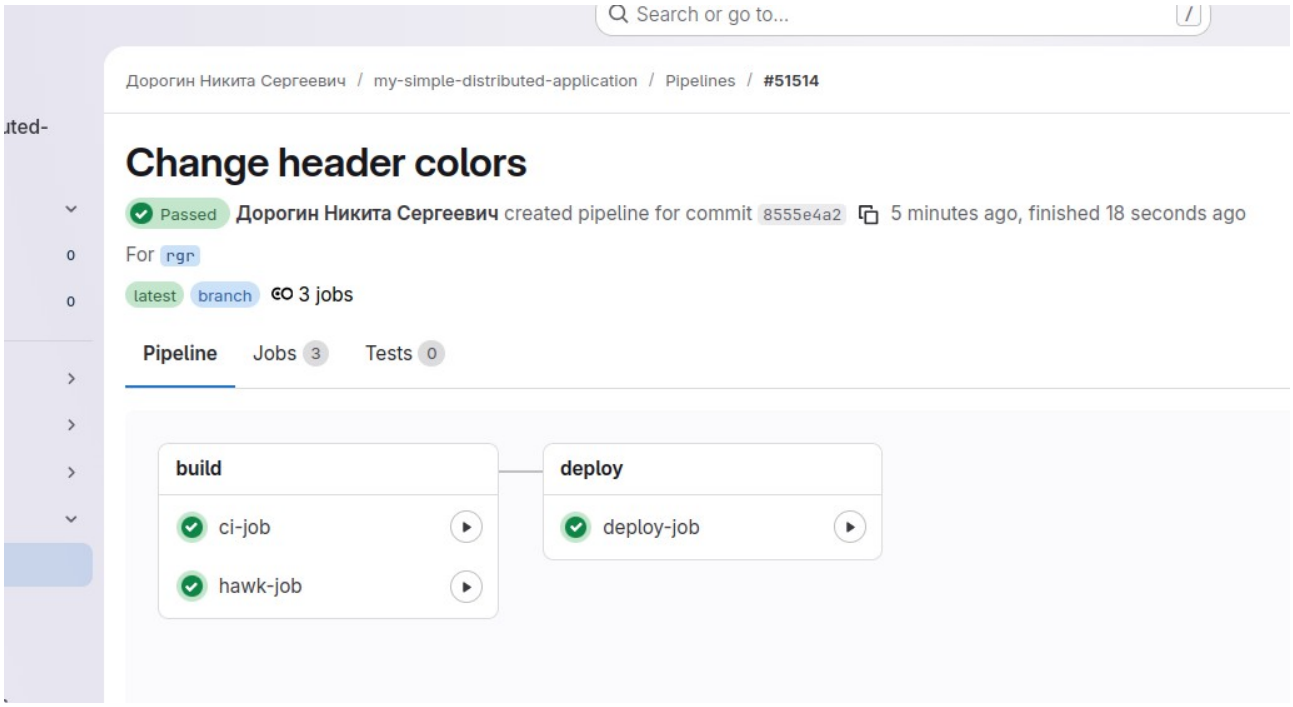


	id	postId	name	email	body
	1	1	fsegrhd	sgdh@fgh	sdfgh

Поменяем цвета для заголовков таблицы:







После завершения этапа deploy мой Deployment в соответсвии с политикой RollingUpdate будет по одной заменять поды со старыми версиями — добавляет новый под, удаляет старый. При просмотре подов может показаться, что реплик на одну больше чем нужно, но так и должно быть.

```
ip312s07@hawk-access:~$ kubectl --kubeconfig hawk_k8s.yaml -n ip312 get pods
NAME                          READY   STATUS    RESTARTS        AGE
endviyou-db6b94dcd-ddc6w      1/1     Running   1 (7d11h ago)   10d
endviyou-db6b94dcd-k972r      1/1     Running   1 (7d11h ago)   10d
ip312s06-app-7dbc9c66f-hkkrx  1/1     Running   1 (7d11h ago)   9d
ip312s06-app-7dbc9c66f-jggrg  1/1     Running   1 (7d11h ago)   15d
ip312s07-app-55f868db78-stn9x 0/1     Running   0               36s
ip312s07-app-7bc44b6bcd-pgdg7 1/1     Running   1 (46m ago)     47m
ip312s07-app-7bc44b6bcd-sx6z7 1/1     Running   0               48m
ip312s07-app-7bc44b6bcd-z5n72 1/1     Running   0               47m
```

По завершении процесса замены всех подов, их снова будет столько, сколько указано в replicas у манифеста Deployment, в моём случае — 3.

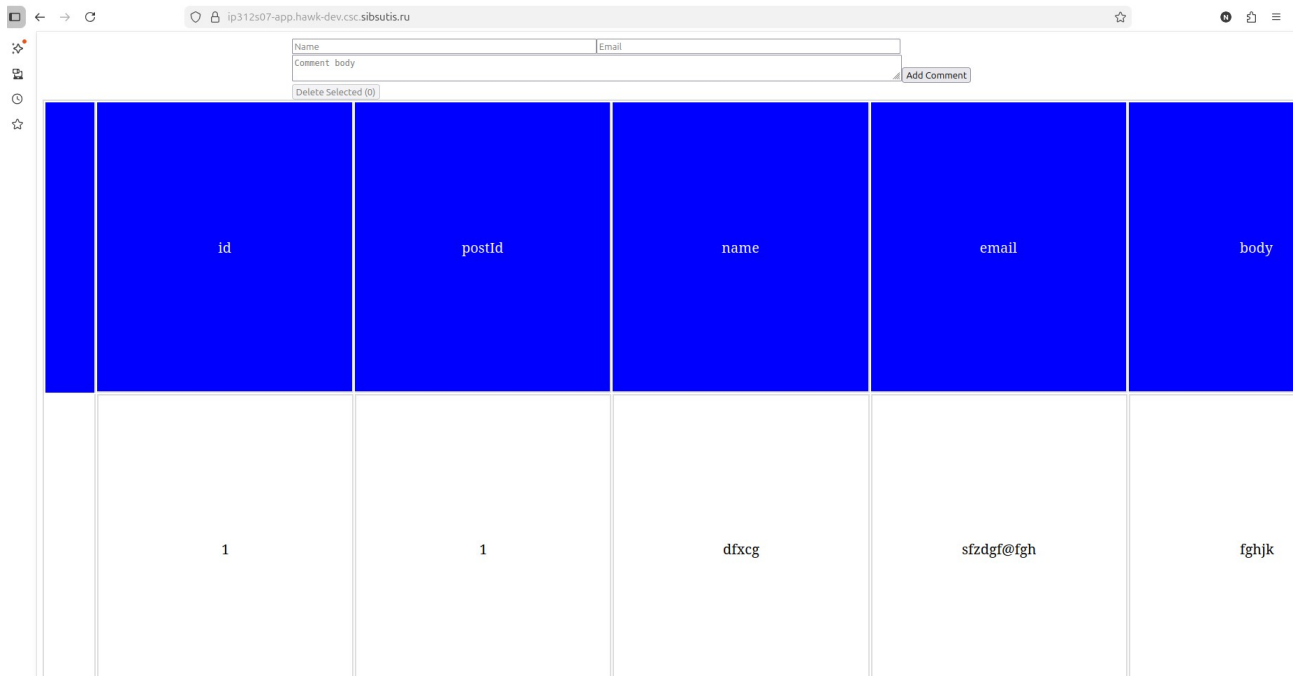
```
ip312s07@hawk-access:~$ kubectl --kubeconfig hawk_k8s.yaml -n ip312 get pods
NAME                          READY   STATUS    RESTARTS        AGE
endviyou-db6b94dcd-ddc6w      1/1     Running   1 (7d11h ago)   10d
endviyou-db6b94dcd-k972r      1/1     Running   1 (7d11h ago)   10d
ip312s06-app-7dbc9c66f-hkkrx  1/1     Running   1 (7d11h ago)   9d
ip312s06-app-7dbc9c66f-jggrg  1/1     Running   1 (7d11h ago)   15d
ip312s07-app-55f868db78-qjstb 1/1     Running   0               20m
ip312s07-app-55f868db78-stn9x 1/1     Running   2 (21m ago)     22m
ip312s07-app-55f868db78-wzw87 1/1     Running   0               20m
ip312s08-app-7cbf447584-kwxcm 1/1     Running   0               2d8h
```

Процесс замены можно отслеживать:

```
ip312s07@hawk-access:~$ kubectl --kubeconfig hawk_k8s.yaml rollout status deployment/ip312s07-app -n ip312
Waiting for deployment "ip312s07-app" rollout to finish: 1 out of 3 new replicas have been updated...
```

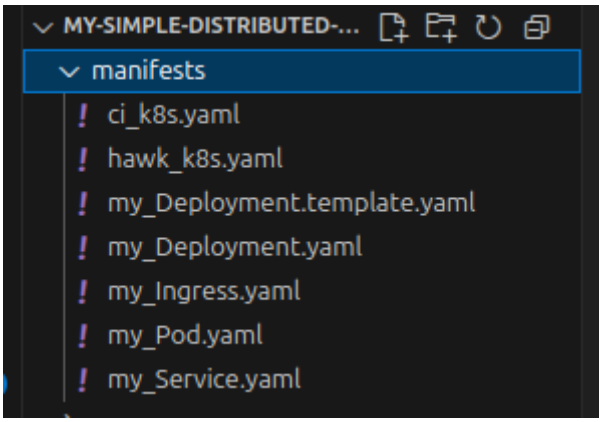
```
ip312s07@hawk-access:~$ kubectl --kubeconfig hawk_k8s.yaml rollout status deployment/ip312s07-app -n ip312
Waiting for deployment "ip312s07-app" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "ip312s07-app" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "ip312s07-app" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "ip312s07-app" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "ip312s07-app" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "ip312s07-app" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "ip312s07-app" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "ip312s07-app" rollout to finish: 1 old replicas are pending termination...
deployment "ip312s07-app" successfully rolled out
```

Новый вид таблицы (записи из прошлой таблицы не сохраняются):



(Интересный факт: если много раз нажимать на кнопку обновления страницы, вас перенесёт на страницу, где работает приложение с другим подом. Дело в том, что все поды обслуживаются одним и тем же Service, которым в свою очередь пользуется один и тот же Ingress и у них, получается, один и тот же домен)

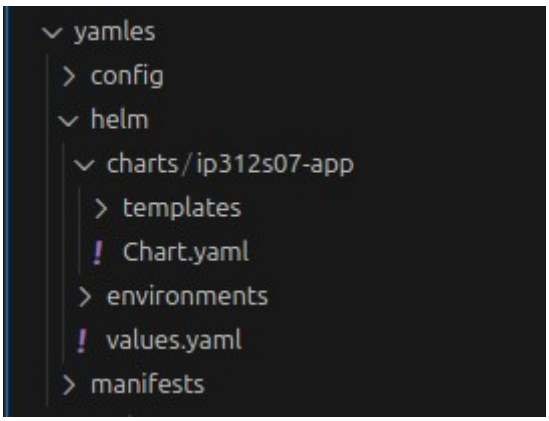
3) Как уже можно было заметить из CI, все файлы конфигурации/ресурсов лежат в отдельной директории репозитория:



Задание на оценку 5:

1, 2) Реализуем этапы diff и deploy с помощью Helmfile.

Для начала изменим структуру директории в которой лежат наши ресурсы и конфигурации



Содержанием папки manifests мы больше пользоваться не будем, поэтому на неё можно не обращать внимания.

Папка config хранит hawk_k8s.yaml и ci_k8s.yaml. // ЭТОГО БОЛЬШЕ НЕ НУЖНО

Папка helm содержит всё то, чем будет пользоваться наш Helmfile для развертывания приложения.

Сам helmfile.yaml лежит в корневой директории.

Теперь пройдёмся по содержанию необходимых файлов:

helmfile.yaml

```
releases:
  - name: ip312s07-app-{{ .Environment.Name }}
    namespace: ip312
    chart: ./yamles/helm/charts/ip312s07-app
    values:
      - ./yamles/helm/values.yaml.gotmpl
```

Helmfile представляет собой инструмент для управления Helm-релизами.

Helm — инструмент для создания манифестов из шаблонов и подаваемых им значений values.

Helm-чарт — набор тех самых шаблонов.

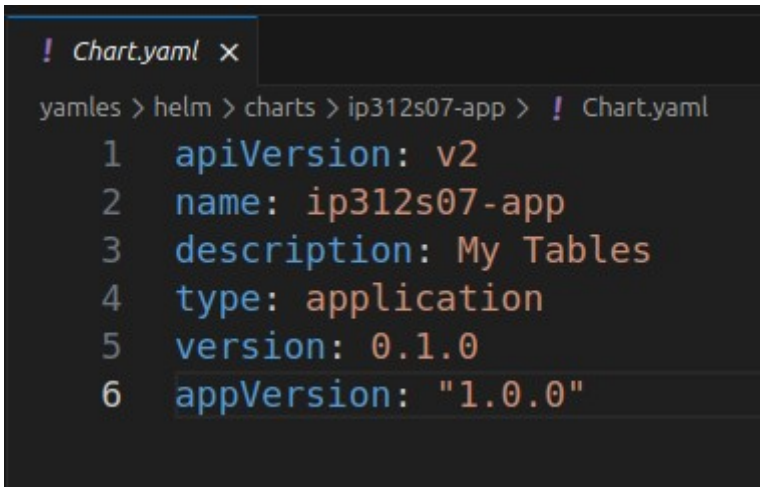
Helm-релиз это «новое состояние» приложения при применении тех самых манифестов, которые сделал Helm.

Поэтому самое важное, что нужно указать для helmfile — это релизы.

У релиза есть имя (name), пространство имен (namespace) в котором он происходит, набор шаблонов, из которых создаются манифесты, применение которых приведёт приложение в «новое состояние» (chart), и значения, подстановка которых в шаблоны сделает нам манифесты.

Важно! В поле chart надо указывать директорию, в которой лежит чарт, само название файла вписывать не надо — Helmfile подцепит чарт автоматически.

Chart.yaml



Сам по себе он не содержит для нас особо полезной информации, только метаданные, однако его наличие необходимо для корректной работы.

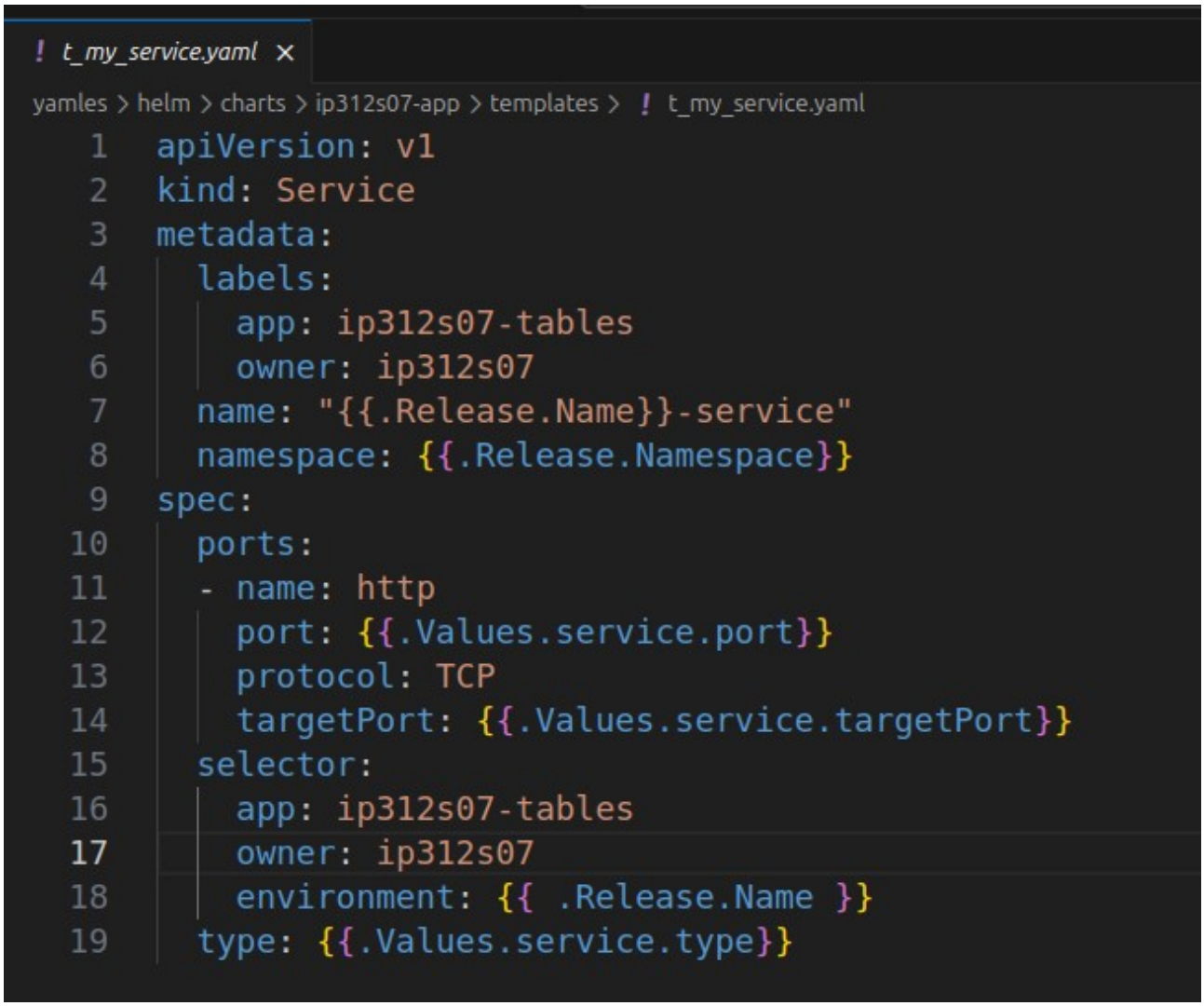
По соседству с ним в в той же директории *charts/ip312s07* располагается папка с шаблонами, которые Helmfile сам находит, зная только местонахождение Chart.yaml.

Эти шаблоны представляют собой те же самые манифесты, которые мы писали ранее, только в некоторых местах с переменными вместо конкретных значений:

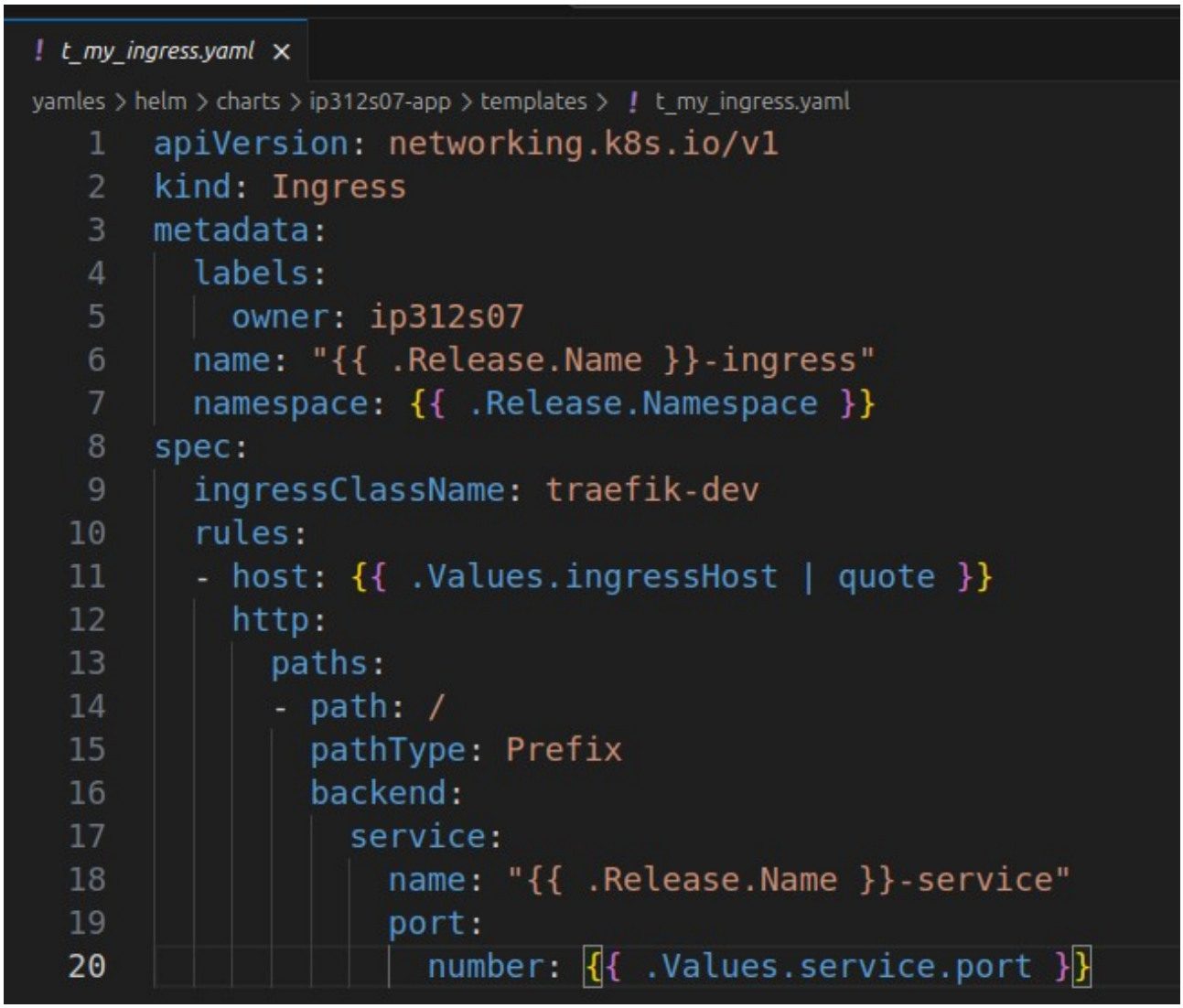
templates/t_my_deployment.yaml

```
40            requests:
41              cpu: {{.Values.resources.requests.cpu}}
42              memory: {{.Values.resources.requests.memory}}
43            limits:
44              cpu: {{.Values.resources.limits.cpu}}
45              memory: {{.Values.resources.limits.memory}}
46          readinessProbe:
47            httpGet:
48              path: /
49              port: {{.Values.app.port}}
50            failureThreshold: 3
51            initialDelaySeconds: 15
52            periodSeconds: 10
53          livenessProbe:
54            httpGet:
55              path: /
56              port: {{.Values.app.port}}
57            failureThreshold: 3
58            initialDelaySeconds: 15
59            periodSeconds: 10
60          startupProbe:
61            httpGet:
62              path: /
63              port: {{.Values.app.port}}
64            failureThreshold: 3
65            initialDelaySeconds: 15
66            periodSeconds: 10
67
```

templates/t_my_service.yaml



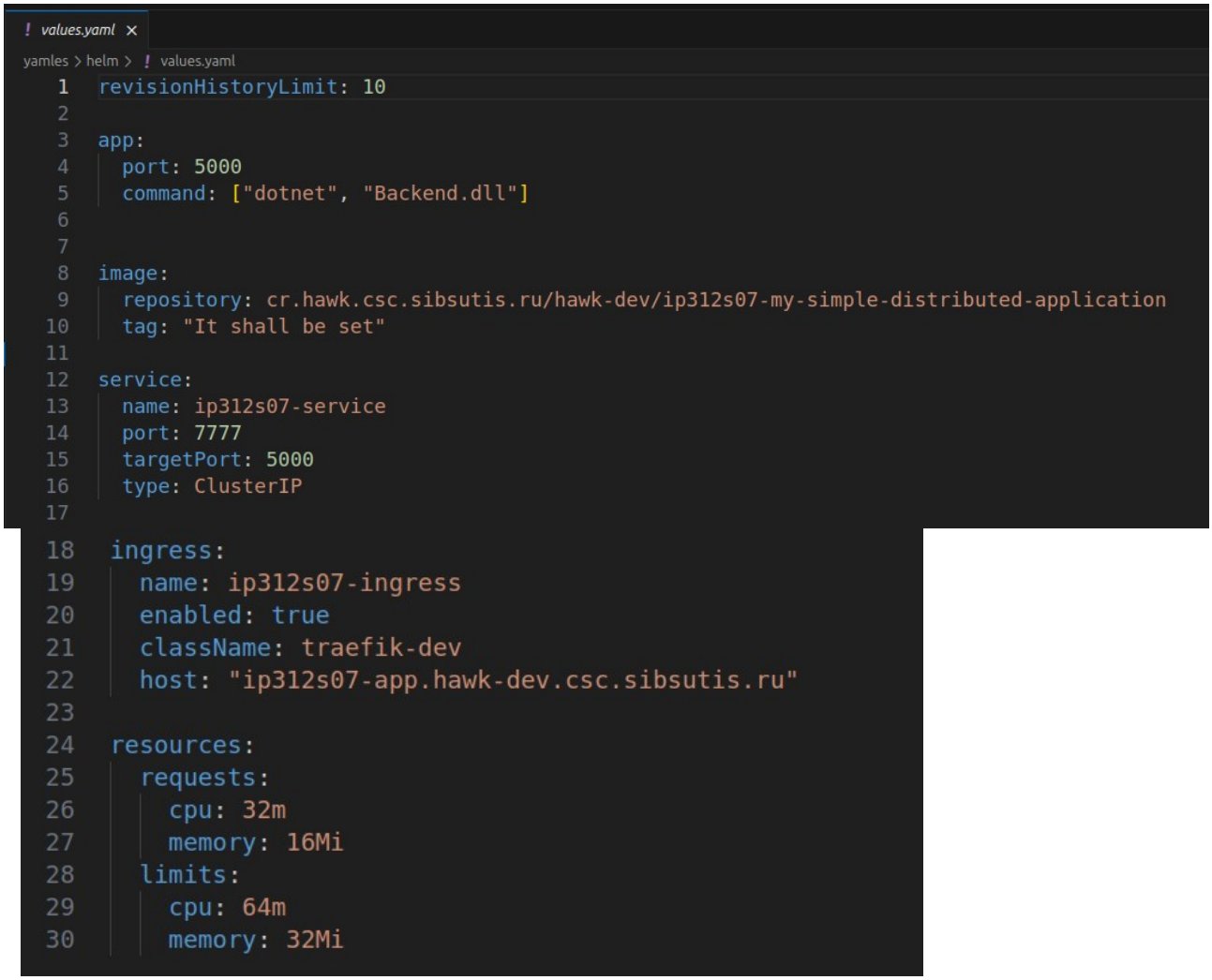
templates/t_my_ingress.yaml



На 2 уровня выше, вне папки charts располагается файл со значениями values.yaml.gotmpl:

values.yaml (в конец названия необходимо добавить .gotmpl)

(мой VS code не знает формат .yaml.gotmpl, поэтому текст у меня в итоге подсвечивается одним цветом, как в обычном текстовом файле, а не как на скринах ниже)

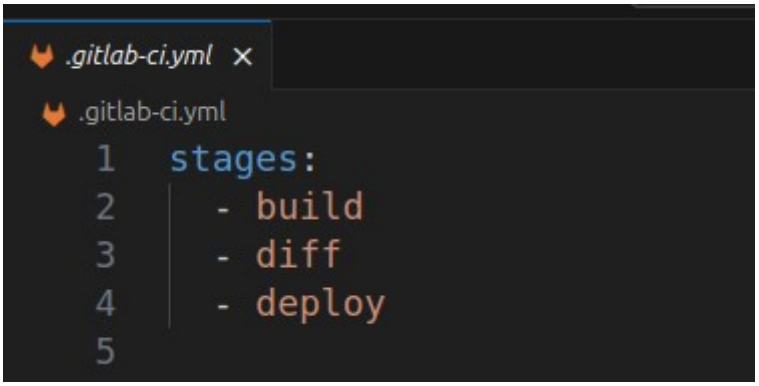


Здесь мы просто записываем значения для полей, существующих в шаблоне (не обязательно всех, какие-то значения можно указывать по умолчанию прямо в шаблоне, если мы не хотим менять их в зависимости от values).

В image.tag пишем несуразную строку, поскольку тэг мы будем заменять, подавая его вход команде helmfile через аргументы.

Теперь необходимо прописать в CI выполнение всего вышеописанного.

Для начала добавим новые стадии.



Весь этап diff (помимо считывания тэга, которое у нас везде есть) представляет собой одну строчку:

helmfile diff

```
diff-job:
  stage: diff
  image: registry.csc.sibsutis.ru/csc-public/internal-container-images/k8s-tools:latest
  before_script:
    - export KUBECONFIG=yamles/config/ci_k8s.yaml
  script:
    - echo "DIFF:"
    - |
      if [ -n "$CI_COMMIT_TAG" ]; then
        IMAGE_TAG=$CI_COMMIT_TAG
        else
        IMAGE_TAG=${CI_COMMIT_SHORT_SHA:0:8}
        fi
        echo "READING THE IMAGE TAG"
    - helmfile diff --set image.tag=${IMAGE_TAG}
  rules:
  - when: always
```

Вывод этой команды в пайплайне выглядит примерно вот так:

```
$ helmfile -e env-one diff --set image.tag=${IMAGE_TAG}
Building dependency release=ip312s07-app, chart=yamles/helm/charts/ip312s07-app
Comparing release=ip312s07-app, chart=yamles/helm/charts/ip312s07-app, namespace=ip312
ip312, ip312s07-app, Deployment (apps) has changed:
  # Source: ip312s07-app/templates/t_my_deployment.yaml
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    labels:
      owner: ip312s07
    name: ip312s07-app
    namespace: ip312
  spec:
    replicas:
    revisionHistoryLimit: 10
    selector:
      matchLabels:
        app: ip312s07-tables
        owner: ip312s07
    strategy:
      rollingUpdate:
        maxSurge: 25%
        maxUnavailable: 25%
      type: RollingUpdate

    template:
      metadata:
        labels:
          app: ip312s07-tables
          owner: ip312s07
      spec:
        containers:
        - name: ip312s07-table
-         image: "cr.hawk.csc.sibsutis.ru/hawk-dev/ip312s07-my-simple-distributed-application:7255be4a"
+         image: "cr.hawk.csc.sibsutis.ru/hawk-dev/ip312s07-my-simple-distributed-application:e9ee5c86"
```

Самое главное здесь в строчках 61-62. (на флаг -e пока не обращаем внимания, про него чуть дальше)

Изменение образа (а это собственно единственное изменение, которое мы вносим в манифесты из одной версии в другую) продемонстрировано.

Теперь переделаем этап деплоя:

```yaml
deploy-job:
  stage: deploy
  image: registry.csc.sibsutis.ru/csc-public/internal-container-images/k8s-tools:latest
  before_script:
    - export KUBECONFIG=yamles/config/ci_k8s.yaml
  script:
    - echo "START DEPLOYING..."
    - |
      if [ -n "$CI_COMMIT_TAG" ]; then
        IMAGE_TAG=$CI_COMMIT_TAG
        else
        IMAGE_TAG=${CI_COMMIT_SHORT_SHA:0:8}
        fi
        echo "READING THE IMAGE TAG"

    - echo "USING TAG ${IMAGE_TAG}"i

    - helmfile apply --set image.tag=${IMAGE_TAG}

    - echo "CHANGES WERE SUCCESFULLY APPLIED"
  rules:
  - when: manual
```

Три применения kubectl заменились одним применением helmfile.

CI готов.

3) Как уже можно было заметить, в командах helmfile мы подаём на вход тэг образа в качестве аргумента командной строки:

Поданный нами тэг присваивается полю image.tag в values.

4) Теперь нам нужно реализовать для нашего приложения 2 независимых друг от друга окружения.

Добавляем environments в helmfile.yaml.

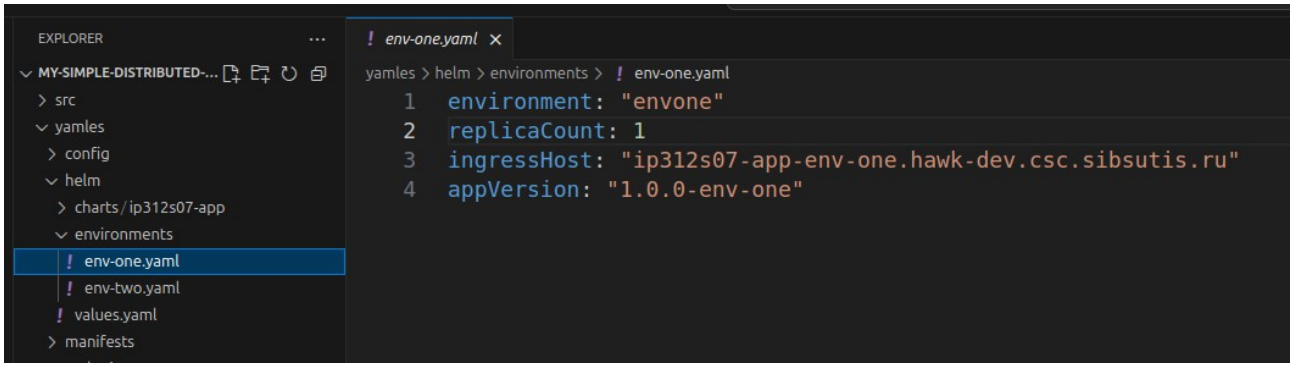
Важно! Окружения должны идти перед релизами и находится от них в отдельной секции.

Секции делятся с помощью ---

helmfile.yaml

```yaml

environments:
  env-one:
    values:
      - yamles/helm/environments/env-one.yaml

  env-two:
    values:
      - yamles/helm/environments/env-two.yaml

---

releases:
  - name: ip312s07-app-{{ .Environment.Name }}
    namespace: ip312
    chart: ./yamles/helm/charts/ip312s07-app
    values:
      - ./yamles/helm/values.yaml.gotmpl
```

Создаём файлы окружений:



В нашем случае окружения будут отличаться друг от друга url приложений и некоторыми строчными полями.

Важно! Поля values и environment не должны иметь схожей структуры! releases.values не должен содержать ссылок на файлы окружения!

В values ingress в поле host теперь будет записываться адрес из Environment.

```
18  ingress:
19    name: ip312s07-ingress
20    enabled: true
21    className: traefik-dev
22    host: {{.Environment.Values.ingressHost | quote}}
```

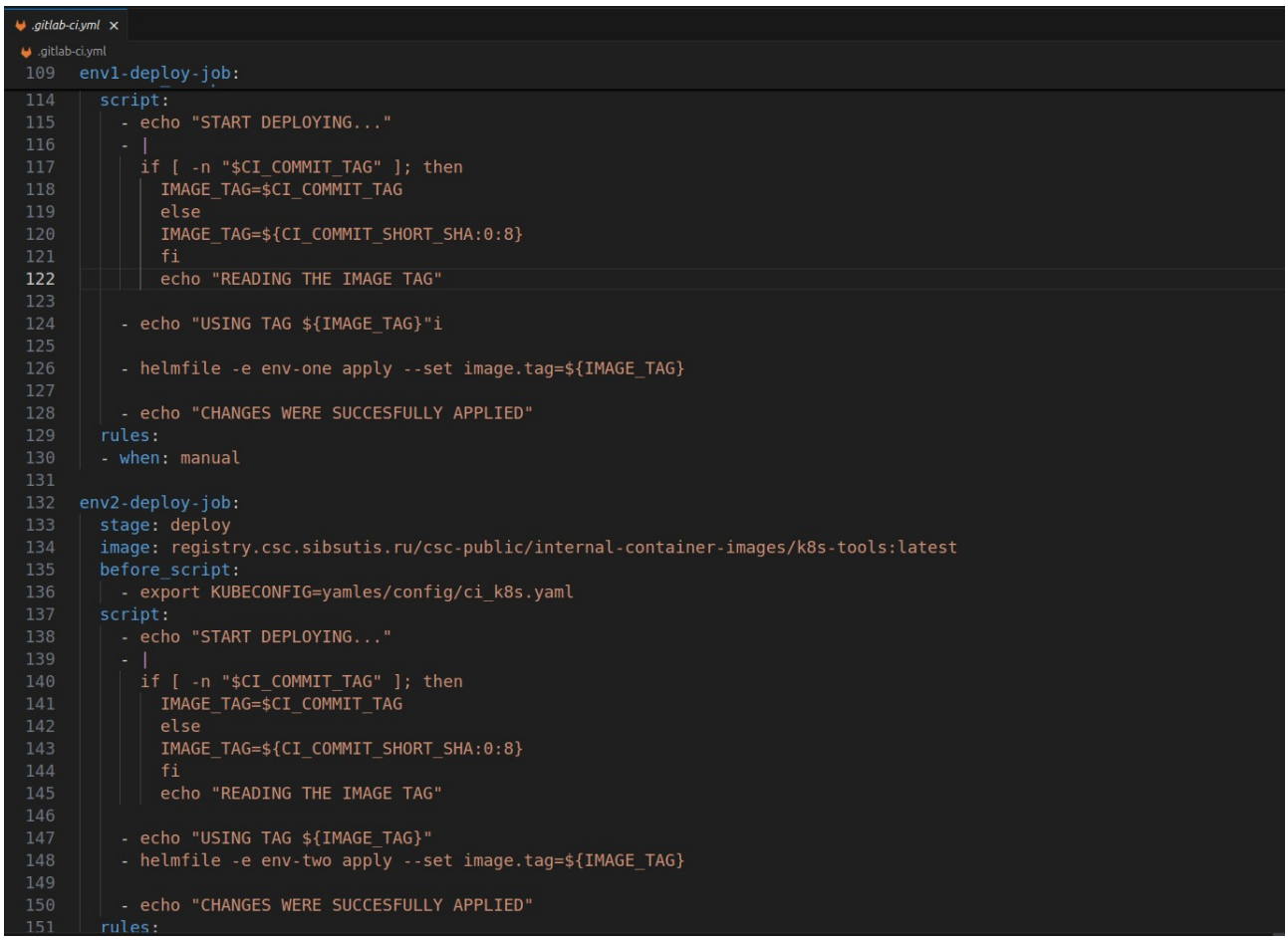
Также имеет место необходимость добавить для деплоймента и его контейнеров специальный label,

```
  metadata:
    labels:
      owner: ip312s07
      environment: {{ .Release.Name }}
```

чтобы selector у Service не выбирал для обслуживания поды из чужого окружения и данные приложений не смешивались.

```
  selector:
    app: ip312s07-tables
    owner: ip312s07
    environment: {{ .Release.Name }}
```

Чтобы окружения можно было запускать независимо друг от друга и деплоить так же независимо сделаем для каждого окружения свою job в CI.



```
.gitlab-ci.yml
109  env1-deploy-job:
114    script:
115      - echo "START DEPLOYING..."
116      - |
117        if [ -n "$CI_COMMIT_TAG" ]; then
118          IMAGE_TAG=$CI_COMMIT_TAG
119          else
120          IMAGE_TAG=${CI_COMMIT_SHORT_SHA:0:8}
121          fi
122          echo "READING THE IMAGE TAG"
123
124      - echo "USING TAG ${IMAGE_TAG}"i
125
126      - helmfile -e env-one apply --set image.tag=${IMAGE_TAG}
127
128      - echo "CHANGES WERE SUCCESFULLY APPLIED"
129    rules:
130    - when: manual
131
132  env2-deploy-job:
133    stage: deploy
134    image: registry.csc.sibsutis.ru/csc-public/internal-container-images/k8s-tools:latest
135    before_script:
136      - export KUBECONFIG=yamles/config/ci_k8s.yaml
137    script:
138      - echo "START DEPLOYING..."
139      - |
140        if [ -n "$CI_COMMIT_TAG" ]; then
141          IMAGE_TAG=$CI_COMMIT_TAG
142          else
143          IMAGE_TAG=${CI_COMMIT_SHORT_SHA:0:8}
144          fi
145          echo "READING THE IMAGE TAG"
146
147      - echo "USING TAG ${IMAGE_TAG}"
148      - helmfile -e env-two apply --set image.tag=${IMAGE_TAG}
149
150      - echo "CHANGES WERE SUCCESFULLY APPLIED"
151    rules:
```

Они абсолютно идентичны, но с флагом -e в команде helmfile мы можем указать конкретное окружение, к которому применяются изменения. (diff я тоже раздвоил)

Теперь давайте попробуем изменить цвет фона заголовков таблицы, допустим, на оранжевый:

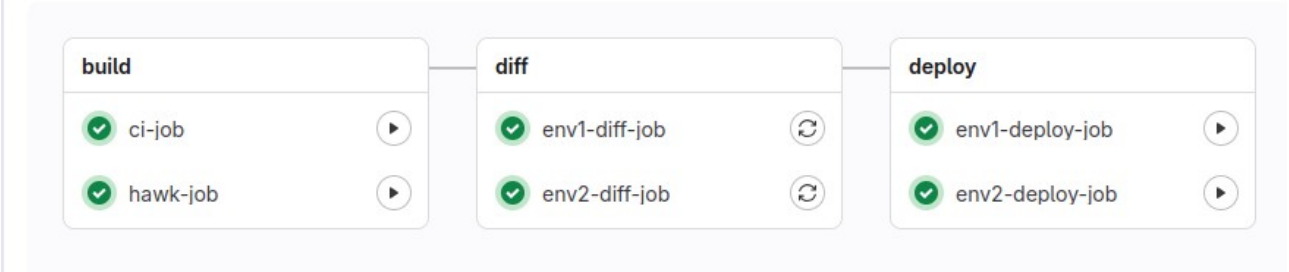
Make orange color

Passed **Дорогин Никита Сергеевич** created pipeline for commit `2820bc9f` 9 hours ago, finished 9 hours ago

For `rgr-5`

branch 6 jobs 13 minutes 14 seconds, queued for 10 seconds

Pipeline | Jobs 6 | Tests 0




```
92            failureThreshold: 3
93            initialDelaySeconds: 15
94            periodSeconds: 10
95          startupProbe:
96            httpGet:
97              path: /
98              port: 5000
99            failureThreshold: 3
100           initialDelaySeconds: 15
101           periodSeconds: 10
102 Upgrading release=ip312s07-app-env-one, chart=yamles/helm/charts/ip312s07-app, namespace=ip312
103 Release "ip312s07-app-env-one" has been upgraded. Happy Helming!
104 NAME: ip312s07-app-env-one
105 LAST DEPLOYED: Sun Nov 30 16:51:18 2025
106 NAMESPACE: ip312
107 STATUS: deployed
108 REVISION: 12
109 TEST SUITE: None
110 Listing releases matching ^ip312s07-app-env-one$
111 ip312s07-app-env-one    ip312          11              2025-11-30 16:23:06.146408195 +0000 UTC failed  ip312s07-app-0.1.0      1.0.0
112 UPDATED RELEASES:
113 NAME                  NAMESPACE   CHART                           VERSION   DURATION
114 ip312s07-app-env-one   ip312       ./yamles/helm/charts/ip312s07-app   0.1.0          2s
115 $ echo "CHANGES WERE SUCCESFULLY APPLIED"
116 CHANGES WERE SUCCESFULLY APPLIED
117 Cleaning up project directory and file based variables                                    00:01
118 Job succeeded
```

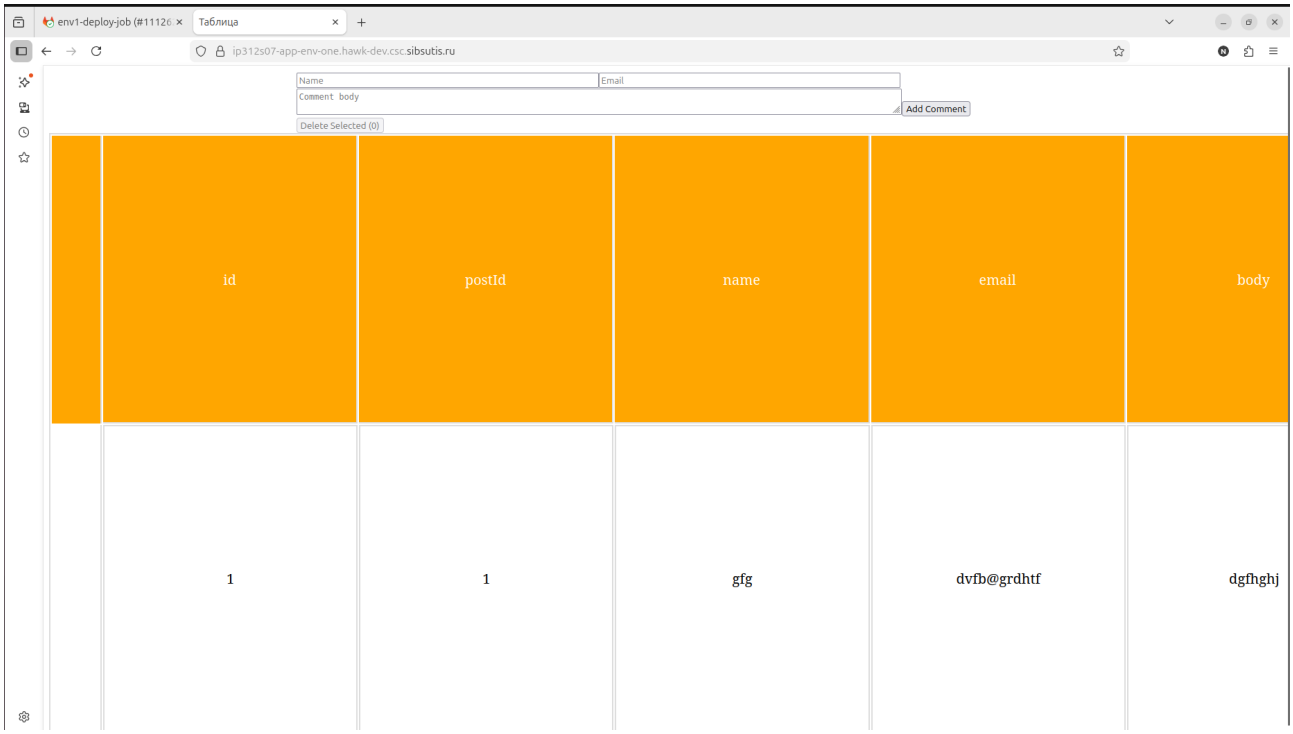
Commit `2820bc9f`
Make orange color

Pipeline #51696 Passed for `rgr-5`

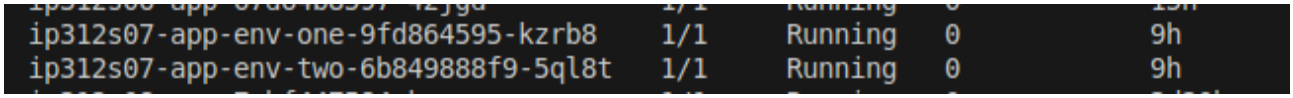
deploy

Related jobs

→ env1-deploy-job
env2-deploy-job

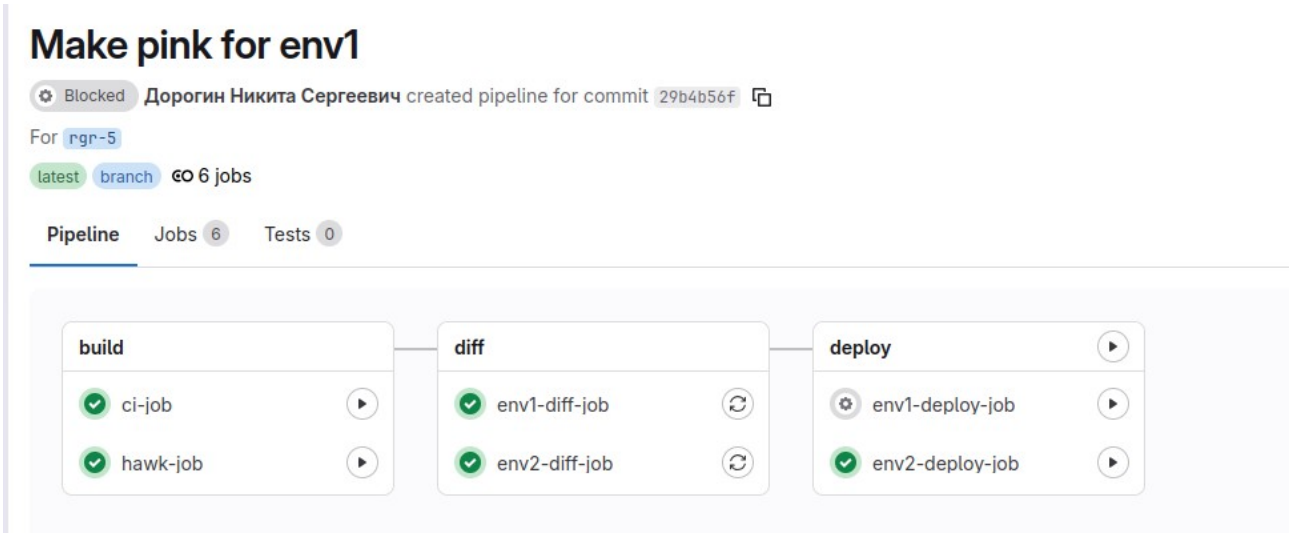


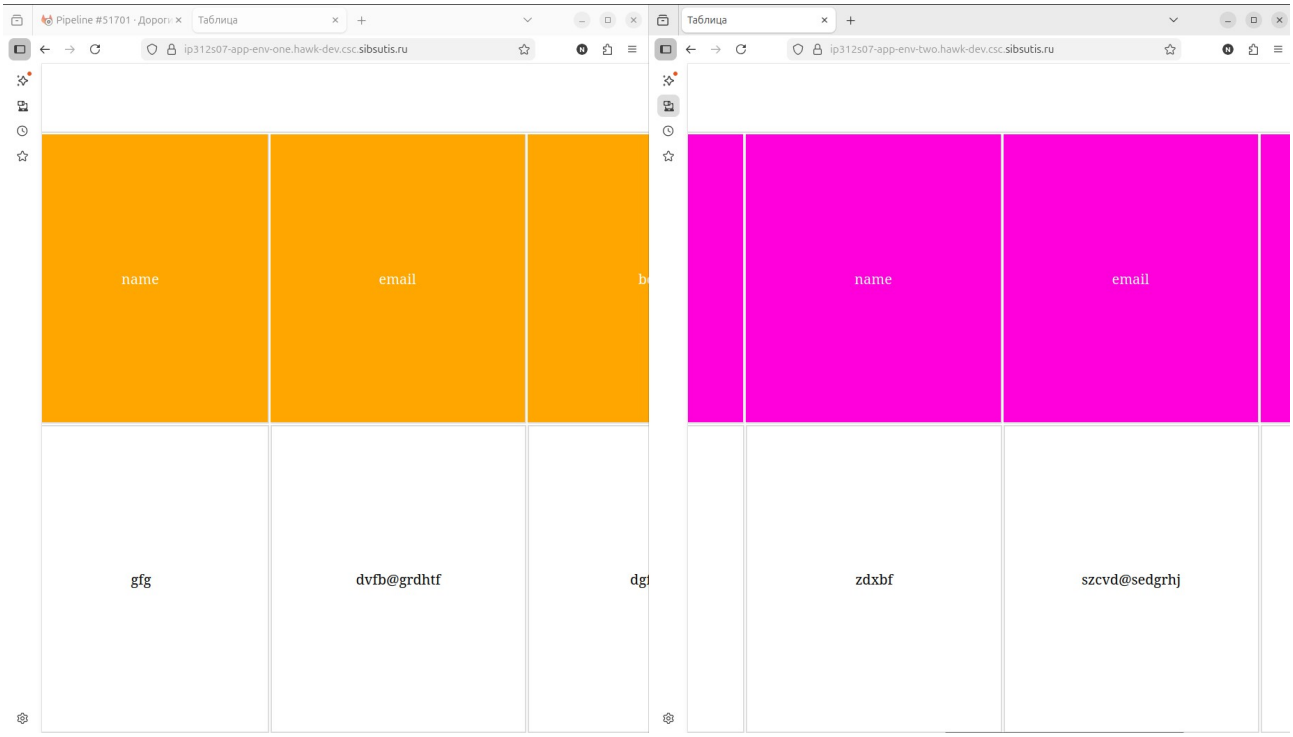
Поды в кластере:



5) Теперь я хочу, чтобы оранжевой была таблица из первого окружения, а из второго какого-нибудь другого цвета. Заодно продемонстрируем, что deploy запускается только по нажатию кнопки.

(ошибка в описании: env2)





6) В пункте 2 я уже демонстрировал values.yaml.gotmpl, а в пункте 4 — файл окружения, сделав замечание, что поля у них не совпадают, и helmfile.yaml, где в values релиза только ссылка на файл values.yaml.gotmpl