

Домашнее задание. GitOps и инструменты поставки

GitLab

В качестве хранилища кода и CI-системы в домашнем задании мы будем использовать SaaS **GitLab**. Зарегистрируйтесь там (если еще не зарегистрированы).

После этого создайте в GitLab **публичный** проект **microservices-demo**.

The screenshot shows the 'Blank project' creation interface in GitLab. At the top, there are four tabs: 'Blank project' (selected), 'Create from template', 'Import project', and 'CI/CD for external repo'. The 'Project name' field contains 'microservices-demo'. The 'Project URL' section shows 'https://gitlab.com/' and a dropdown menu with 'avtandilko' selected. The 'Project slug' field contains 'microservices-demo'. Below this, there is a link: 'Want to house several dependent projects under the same namespace? [Create a group.](#)'. The 'Project description (optional)' field is empty. The 'Visibility Level' section has two options: 'Private' (unselected) and 'Public' (selected). The 'Public' option is described as 'The project can be accessed without any authentication.' There is also an unchecked checkbox for 'Initialize repository with a README'. At the bottom, there are two buttons: 'Create project' (green) and 'Cancel' (grey).

Подготовка GitLab репозитория

Переместите в проект `microservices-demo` код из GitHub репозитория

```
git clone https://github.com/GoogleCloudPlatform/microservices-demo
cd microservices-demo
git remote add gitlab git@gitlab.com:<YOUR_LOGIN>/microservices-demo.git
git remote remove origin
git push gitlab master
```

Предварительно необходимо добавить публичный ssh ключ в профиль GitLab, либо использовать https

Создание Helm чартов

- Перед началом выполнения домашнего задания необходимо подготовить Helm чарты для каждого микросервиса
- Можно воспользоваться наработками из предыдущих домашних заданий, либо скопировать готовые чарты из [демонстрационного репозитория](#) (директория `deploy/charts`)
- Во всех манифестах, описывающих deployment, обязательно должны быть параметризованы **название образа** и его **тег**. Рекомендуется придерживаться следующего формата:

```
image:  
  repository: frontend  
  tag: latest
```

Создание Helm чартов

Результат поместите в директорию `deploy/charts`. Должен получиться следующий вывод:

```
$ tree -L 1 deploy/charts
deploy/charts
├── adservice
├── cartservice
├── checkoutservice
├── currencyservice
├── emailservice
├── frontend
├── loadgenerator
├── paymentservice
├── productcatalogservice
├── recommendationservice
└── shippingservice
```

Подготовка Kubernetes кластера

Любым способом (вручную через web-интерфейс, консольным клиентом, утилитой Terraform) разверните managed Kubernetes кластер в YC.

Понадобится как минимум 4 ноды типа `intel ice lake, 2vCPU, 8 GB RAM`. Остальные параметры можно оставить по умолчанию.

Подготовка Kubernetes кластера | Задание со



- Автоматизируйте создание Kubernetes кластера
- Кластер должен разворачиваться после запуска pipeline в GitLab
- Инфраструктурный код и файл `.gitlab-ci.yml` поместите в отдельный репозиторий и приложите ссылку на данный репозиторий в PR

Continuous Integration

- Соберите Docker образы для всех микросервиса и поместите данные образы в Docker Hub
- При тегировании образов используйте подход **semver**, например, первому собранному образу логично выставить тег **v0.0.1**.

После выполнения данного шага в Docker Hub должно находиться как минимум по одному образу для каждого микросервиса

Continuous Integration | Задание со

Подготовьте pipeline, который будет содержать следующие стадии:

- Сборку Docker образа для каждого из микросервисов
- Push данного образа в Docker Hub

В качестве тега образа используйте `tag` коммита, инициирующего сборку (переменная **CI_COMMIT_TAG** в GitLab CI)

После выполнения данного шага в Docker Hub должно находиться как минимум по одному образу для каждого микросервиса

GitOps

Подготовка

- Установим CRD, добавляющую в кластер новый ресурс - **HelmRelease**:

```
kubectl apply -f https://raw.githubusercontent.com/fluxcd/helm-operator/master/deploy/flux-helm-release-crd.yaml
```

- Добавим официальный репозиторий Flux

```
helm repo add fluxcd https://charts.fluxcd.io
```

Подготовка

Произведем установку Flux в кластер, в namespace `flux`

```
kubectl create namespace flux  
helm upgrade --install flux fluxcd/flux -f flux.values.yaml --namespace flux
```

Используйте values из следующего файла (не забудьте изменить название git-репозитория)

Подготовка

- Установим **Helm operator**:

```
helm upgrade --install helm-operator fluxcd/helm-operator -f helm-operator.values.yaml --namespace flux
```

Используйте values из следующего [файла](#)

- Установим **fluxctl** на локальную машину для управления нашим CD инструментом. Руководство по установке доступно по [ссылке](#)

Подготовка

Наконец, добавим в свой профиль GitLab публичный ssh-ключ, при помощи которого flux получит доступ к нашему git-репозиторию.

User Settings > SSH Keys

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

To add an SSH key you need to [generate one](#) or use an [existing key](#).

Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/id_ed25519.pub' or '~/.ssh/id_rsa.pub' and begins with 'ssh-ed25519' or 'ssh-rsa'. Don't use your private SSH key.

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQGDy5aMk+86m4lpHP5t2TUza3xeTsHFZnKAa1aMc+bSGL
ci1rCyyQX2b/uj/EVGwTa0v8qUIELA3V7cDkH2LcD83WPqJHTmR+wgHh4jY6gZuPdoQ38t771a8+
WbkMFEVxYvtLYWZsxErtE1HINOi41Xvw4e9zLOvTKKh7jOCbA25SWqwd4N1OH/1LuBt92+SSa2Da
SdYBx3yMcjoVjclVVOdAx8/zhW8+5BPWefGe271be1tK4i1WzmlmzPv3zwYj6wFTV4nqpC9yov9Sq
GmaWUY0QsU89ssxaSASKhnmf93zSmo4iCNALdMc/r15BCfMBRp3Pg+ssnJ2XN7K7Fiz2JqIU1
Z/Kkd9NLjvE6kyDmlQG+AovLWOxy1QS1gRuAtBvbG0MjzqB8OvyUYXqtW8TCLtDlGSdAEole1TggA
s4aHnq8HqDRclrEktJQjRrgwX3Nm5lTPNaw63F/aFLB4IRfVuwJ8eMPLrJoZ3hJ+TB4V8JS0BBUG7
```

Title

Name your individual key via a title

Add key

Получить значение ключа можно следующей командой:

```
fluxctl identity --k8s-fwd-ns flux
```

Проверка

Пришло время проверить корректность работы Flux. Как мы уже знаем, Flux умеет автоматически синхронизировать состояние кластера и репозитория. Это касается не только сущностей `HelmRelease`, которыми мы будем оперировать для развертывания приложения, но и обыкновенных манифестов.

Поместите манифест, описывающий namespace `microservices-demo` в директорию `deploy/namespaces` и сделайте push в GitLab:

```
apiVersion: v1
kind: Namespace
metadata:
  name: microservices-demo
```

Проверка

Если все предыдущие шаги проделаны верно - в кластере через некоторое время будет создан namespace `microservices-demo`.

Также в логах pod с flux должна появиться строка, описывающая действия данного инструмента:

```
ts=2020-02-09T18:43:14.070208901Z caller=sync.go:594 method=Sync cmd="kubectl apply -f -" took=647.102096ms err=null output="namespace/microservices-demo created"
```


HelmRelease

Мы подошли к сущностям, которыми управляет helm-operator - `HelmRelease`.

Для описания сущностей такого вида создадим отдельную директорию `deploy/releases` и поместим туда файл `frontend.yaml` с описанием конфигурации релиза.

Содержимое файла `frontend.yaml` доступно на следующем слайде (не забудьте поменять названия git и Docker Hub репозиториев).

Не забудьте сделать push в GitLab после добавления манифеста

HelmRelease

frontend.yaml:

```
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: frontend
  namespace: microservices-demo
  annotations:
    fluxcd.io/ignore: "false"
    fluxcd.io/automated: "true"
    flux.weave.works/tag.chart-image: semver:~v0.0
spec:
  releaseName: frontend
  helmVersion: v3
  chart:
    git: git@gitlab.com:<YOUR_LOGIN>/microservices-demo.git
    ref: master
    path: deploy/charts/frontend
  values:
    image:
      repository: <YOUR_LOGIN>/frontend
      tag: v0.0.1
```

HelmRelease

Опишем некоторые части манифеста HelmRelease:

```
...
metadata:
  ...
  annotations:
    ...
    fluxcd.io/automated: "true" # 1
    flux.weave.works/tag.chart-image: semver:~v0.0 # 2
spec:
  releaseName: frontend
  ...
  chart: # 3
    git: git@gitlab.com:<YOUR_LOGIN>/microservices-demo.git
    ref: master
    path: deploy/charts/frontend
  values: # 4
    image:
      repository: <YOUR_LOGIN>/frontend
      tag: v0.0.1
```

HelmRelease

1. Аннотация разрешает автоматическое обновление релиза в Kubernetes кластере в случае изменения версии Docker образа в Registry
2. Указываем Flux следить за обновлениями конкретных Docker образов в Registry.

Новыми считаются только образы, имеющие версию выше текущей и отвечающие маске семантического версионирования `~0.0` (например, `0.0.1`, `0.0.72`, но не `1.0.0`)

HelmRelease

3. Helm chart, используемый для развертывания релиза. В нашем случае указываем git-репозиторий, и директорию с чартом внутри него
4. Переопределяем переменные Helm chart. В дальнейшем Flux может сам переписывать эти значения и делать commit в git-репозиторий (например, изменять тег Docker образа при его обновлении в Registry)

Более подробное описание доступно по [ссылке](#)

HelmRelease | Проверка

Убедимся что HelmRelease для микросервиса frontend появился в кластере:

```
kubectl get helmrelease -n microservices-demo
```

NAMESPACE	NAME	RELEASE	STATUS	MESSAGE
microservices-demo	frontend	frontend	deployed	Helm release sync succeeded

По статусу мы можем понять, что релиз применился успешно, и frontend запущен. Дополнительно проверим это:


```
helm list -n microservices-demo
```

Командой `fluxctl --k8s-fwd-ns flux sync` МОЖНО
инициировать синхронизацию вручную

Обновление образа

1. Внесите изменения в исходный код микросервиса frontend (не имеет значения, какие) и пересоберите образ, при этом инкрементировав версию тега (до **v0.0.2**)
2. Дождитесь автоматического обновления релиза в Kubernetes кластере (для просмотра ревизий релиза можно использовать команду `helm history frontend -n microservices-demo`)
3. Проверьте, изменилось ли что-либо в git-репозитории (в частности, в файле `deploy/releases/frontend.yaml`)


Обновление образа


Commit 4cf04c1f  authored 45 minutes ago by  Weave Flux

[Browse files](#) [Options ▾](#)

Auto-release avtandilko/frontend:v0.0.2

[ci skip]

 parent [4922f1ef](#) [Pmaster](#) [...](#)

 No related merge requests found

Changes 1

Showing [1 changed file ▾](#) with [1 addition](#) and [1 deletion](#) [Hide whitespace changes](#) [Inline](#) [Side-by-side](#)

Обновление Helm chart

1. Попробуем внести изменения в Helm chart `frontend` и поменять имя `deployment` на `frontend-hipster`
2. Сделайте push измененного Helm chart в GitLab и понаблюдайте за процессом

Найдите в логах helm-operator строки, указывающие на механизм проверки изменений в Helm chart и определения необходимости обновить релиз. Приложите данные строки к описанию PR.

Самостоятельное задание

- Добавьте манифесты HelmRelease для всех микросервисов входящих в состав HipsterShop
- Проверьте, что все микросервисы успешно развернулись в Kubernetes кластере

Полезные команды fluxctl

- `export FLUX_FORWARD_NAMESPACE=flux` переменная окружения, указывающая на namespace, в который установлен flux (альтернатива ключу `--k8s-fwd-ns <flux installation ns>`)
- `fluxctl list-workloads -a` посмотреть все workloads, которые находятся в зоне видимости flux
- `fluxctl list-images -n microservices-demo` - посмотреть все Docker образы, используемые в кластере (в namespace microservices-demo)
- `fluxctl automate/deautomate` - включить/выключить автоматизацию управления workload

Полезные команды fluxctl

- `fluxctl policy -w microservices-demo:helmrelease/frontend --tag-all='semver:~0.1'` - установить всем сервисам в workload microservices-demo:helmrelease/frontend политику обновления образов из Registry на базе семантического версионирования с маской 0.1.*
- `fluxctl sync` - принудительно запустить синхронизацию состояния git-репозитория с кластером
- `fluxctl release --workload=microservices-demo:helmrelease/frontend --update-all-images` - принудительно инициировать сканирование Registry на предмет наличия свежих Docker образов

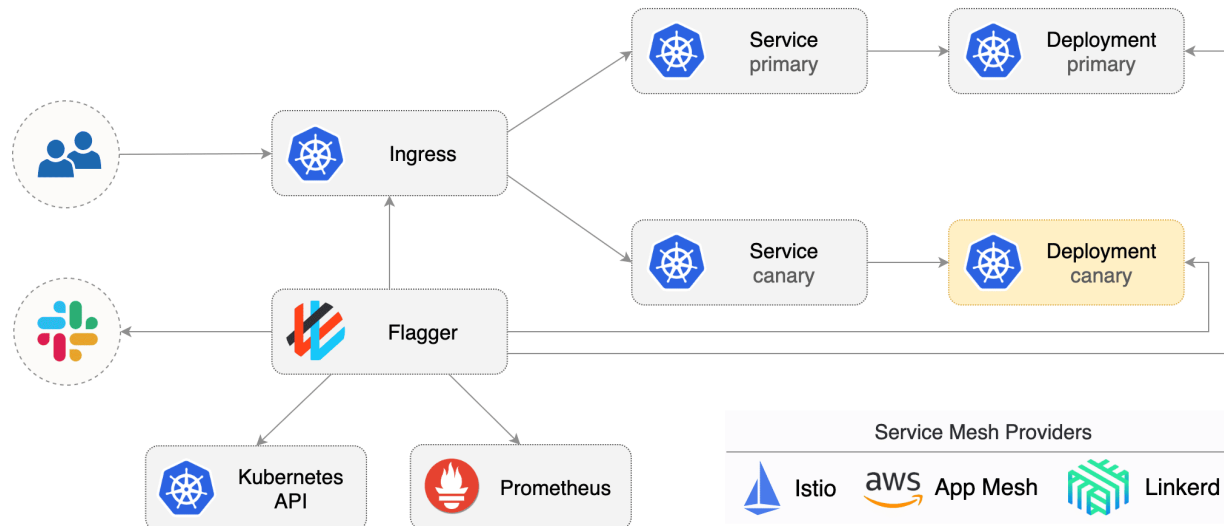
Canary deployments с Flagger и Istio

Flagger

Flagger - оператор Kubernetes, созданный для автоматизации canary deployments.

Flagger может использовать:

- Istio, Linkerd, App Mesh или nginx для маршрутизации трафика
- Prometheus для анализа канареечного релиза



Установка Istio

Установим Istio в кластер с помощью `istioctl`:

- Документация по установке
- Шаги установки



Подробнее про `configuration profile`



(will be deprecated) документация по установке Istio с использованием Helm

Установка Istio | Задание со

Реализуйте установку Istio альтернативным способом:

- Istio operator

Добавьте в PR описание процесса установки.

Установка Flagger

1. Добавление helm-репозитория flagger:

```
helm repo add flagger https://flagger.app
```

1. Установка CRD для Flagger:

```
kubectl apply -f  
https://raw.githubusercontent.com/weaveworks/flagger/master/artifacts/flagger/crd.yaml
```

3. Установка flagger с указанием использовать Istio:

```
helm upgrade --install flagger flagger/flagger \\\n--namespace=istio-system \\\n--set crd.create=false \\\n--set meshProvider=istio \\\n--set metricsServer=http://prometheus:9090
```

Istio | Sidecar Injection

Измените созданное ранее описание namespace `microservices-demo`

```
1 apiVersion: v1
2 kind: Namespace
3 metadata:
4   name: microservices-demo
5   labels:
6     istio-injection: enabled
```

Выделенная строка указывает на необходимость добавить в каждый pod sidecar контейнер с envoy proxy.

После синхронизации проверку можно выполнить командой

```
kubectl get ns microservices-demo --show-labels
```

Istio | Sidecar Injection

Самый простой способ добавить sidecar контейнер в уже запущенные pod - удалить их:

```
kubectl delete pods --all -n microservices-demo
```

После этого можно проверить, что контейнер с названием **istio-proxy** появился внутри каждого pod:

```
kubectl describe pod -l app=frontend -n microservices-demo
```

Доступ к frontend

На текущий момент у нас отсутствует ingress и мы не можем получить доступ к frontend снаружи кластера.

В то же время Istio в качестве альтернативы классическому ingress предлагает свой набор абстракций.

Чтобы настроить маршрутизацию трафика к приложению с использованием Istio, нам необходимо добавить ресурсы **VirtualService** и **Gateway**

Создайте директорию `deploy/istio` и поместите в нее следующие манифесты:

- `frontend-vs.yaml`
- `frontend-gw.yaml`

Istio | VirtualService

frontend-vs.yaml :

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: frontend
  namespace: microservices-demo
spec:
  hosts:
    - "*"
  gateways:
    - frontend
  http:
    - route:
        - destination:
            host: frontend
            port:
              number: 80
```

Istio | Gateway

frontend-gw.yaml:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: frontend
  namespace: microservices-demo
spec:
  selector:
    istio: ingressgateway
  servers:
    - port:
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - "*"
```

Istio | Gateway

Созданный Gateway можно увидеть следующим образом:

```
kubectl get gateway -n microservices-demo
```

NAME	AGE
frontend	3m28s

Для доступа снаружи нам понадобится EXTERNAL-IP сервиса `istio-ingressgateway`:

```
kubectl get svc istio-ingressgateway -n istio-system
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
istio-ingressgateway	LoadBalancer	10.0.4.1	35.190.206.33	...
...

Теперь мы можем обращаться к frontend так `http://EXTERNAL-IP`

Istio | Самостоятельное задание

В нашей ситуации ресурсы `Gateway` и `VirtualService` логически являются частью инфраструктурного кода, описывающего окружение микросервиса `frontend`. Поэтому, оправданно будет перенести манифесты в Helm chart.

Дополните Helm chart `frontend` манифестами `gateway.yaml` и `virtualService.yaml`. Оригинальные манифесты удалите вместе с директорией `deploy/istio`.

Flagger | Canary

Перейдем непосредственно к настройке канареечных релизов. Добавьте в Helm chart `frontend` еще один файл - `canary.yaml`

В нем будем хранить описание стратегии, по которой необходимо обновлять данный микросервис.

Узнать подробнее о Canary Custom Resource можно по [ссылке](#).

Flagger | Canary

`canary.yaml`:

```
apiVersion: flagger.app/v1alpha3

kind: Canary

metadata:
  name: frontend
  namespace: microservices-demo

spec:
  provider: istio

  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: frontend

  service:
    port: 80
    targetPort: 8080

    gateways:
      - frontend

    hosts:
      - "*"

    trafficPolicy:
      tls:
        mode: DISABLE

  canaryAnalysis:
    interval: 30s
```

```
threshold: 5

maxWeight: 30

stepWeight: 5

metrics:

- name: request-success-rate
  threshold: 99
  interval: 30s

- name: request-duration
  threshold: 500
  interval: 30s
```

Flagger | Canary

Проверим, что Flagger:

- Успешно инициализировал canary ресурс **frontend**:

```
kubectl get canary -n microservices-demo
```

NAMESPACE	NAME	STATUS	WEIGHT	LASTTRANSITIONTIME
microservices-demo	frontend	Initializing	0	2020-02-09T22:23:00Z

- Обновил pod, добавив ему к названию постфикс **primary**:

```
kubectl get pods -n microservices-demo -l app=frontend-primary
```

NAME	READY	STATUS	RESTARTS	AGE
frontend-primary-649f9c4579-jgv8h	2/2	Running	0	2m56s

Flagger | Canary

Попробуем провести релиз. Соберите новый образ frontend с тегом v0.0.3 и сделайте push в Docker Hub.

Через некоторое время в выводе `kubectl describe canary frontend -n microservices-demo` мы сможем наблюдать следующую картину:

Type	Reason	Age	From	Message
Normal	Synced	6m4s	flagger	New revision detected! Scaling up frontend.microservices-demo
Normal	Synced	5m33s	flagger	Starting canary analysis for frontend.microservices-demo
Normal	Synced	5m33s	flagger	Advance frontend.microservices-demo canary weight 5
Warning	Synced	33s (x5 over 5m4s)	flagger	Halt advancement no values found for istio metric request-success-rate probably frontend.microservices-demo is not receiving traffic
Warning	Synced	3s	flagger	Rolling back frontend.microservices-demo failed checks threshold reached 5
Warning	Synced	3s	flagger	Canary failed! Scaling down frontend.microservices-demo

Flagger | Самостоятельное задание

- Определите причину неуспешности релиза
- Добейтесь успешного выполнения релиза

Рекомендуется обратить внимание на Helm chart `loadgenerator` и модифицировать его таким образом, чтобы нагрузка генерировалась на внешний, по отношению к кластеру, URL (тем самым - создав имитацию реального поведения пользователей)

Flagger | Самостоятельно задание

После успешного выполнения задания и повторной попытки релиза статус ресурса canary должен измениться на **Succeeded** а в выводе

```
kubectl describe canary frontend -n microservices-demo
```

появятся events следующего вида:

Events:

Type	Reason	Age	From	Message
----	-----	----	----	-----
Normal	Synced	8m27s	flagger	New revision detected! Scaling up frontend.microservices-demo
Normal	Synced	7m58s	flagger	Starting canary analysis for frontend.microservices-demo
Normal	Synced	7m57s	flagger	Advance frontend.microservices-demo canary weight 5
Normal	Synced	6m27s	flagger	Advance frontend.microservices-demo canary weight 10
Normal	Synced	5m57s	flagger	Advance frontend.microservices-demo canary weight 15
Normal	Synced	5m27s	flagger	Advance frontend.microservices-demo canary weight 20

```
Normal    Synced    3m57s    flagger    Advance frontend.microservices-demo canary weight
25
Normal    Synced    3m27s    flagger    Advance frontend.microservices-demo canary weight
30
```

Что обязательно должно быть в PR

- Ссылка на Ваш инфраструктурный репозиторий и файл `.gitlab-ci.yml`
- Добавьте в README вывод команды `kubectl get canaries -n <CHANGE_ME>` (там должна быть ваша выкладка в статусе `Succeeded`)
- Показать вывод после успешной выкладки `kubectl describe canary -n <CHANGE_ME> frontend`

Дополнительные задания

Задания ниже:

- Не являются обязательными
- Не имеют единственного решения
- Не гарантированно будут проверены преподавателями (но мы постараемся отметить понравившиеся моменты)
- Рекомендуются к самостоятельному выполнению тем, у кого есть желание подробнее разобраться в вопросе, а главное - хватает свободного времени 😊

Flagger | Задание со ★

- Реализуйте канареечное развертывание для одного из оставшихся микросервисов
- Опишите сложности с которыми пришлось столкнуться в PR и соответствующим образом модифицируйте файлы в GitLab репозитории

Flagger | Задание со ★

- Реализуйте получение уведомлений о релизах в Slack (используйте отдельный workspace, не отправляйте алерты в otus-devops.slack.com) или в Alertmanager при помощи данной инструкции
- Опишите получившийся результат в PR и соответствующим образом модифицируйте файлы в GitLab репозитории

Инфраструктурный репозиторий | Задание со ★

- Создайте дополнительный репозиторий и поместите туда инфраструктурный код для развертывания всех сервисов, использующихся в данном домашнем задании (Flux, Flagger, Istio, Helm-operator, etc...)
- В качестве решения для развертывания можно использовать Helmfile, [Terraform Provider для Helm](#), либо свой вариант
- Опишите получившийся результат в PR

Distributed Tracing | Задание со ★

- Установите Jaeger и научитесь собирать трейсы:
 - Непосредственно с микросервисов
 - С sidecar контейнеров istio-proxy
- Опишите получившийся результат в PR и соответствующим образом модифицируйте файлы в GitLab репозитории

Monorepos: Please don't! | Задание с ★★

- Перенесите каждый микросервис в выделенный репозиторий
- Опробуйте GitOps подход с использованием подобной схемы
- Опишите получившийся результат в PR

<https://medium.com/@mattklein123/monorepos-please-dont-e9a279be011b>

Argo CD | Задание с ★★

- Повторите сделанную работу с использованием стека инструментов **Argo**
- Опишите результат в PR и выложите все дополнительные манифесты в GitLab репозиторий

Проверка ДЗ

- Результаты вашей работы должны быть добавлены в ветку **kubernetes-gitops** вашего GitHub репозитория `<YOUR_LOGIN>_platform`
- В **README.md** рекомендуется внести описание того, что сделано
- Создайте Pull Request к ветке **master** (описание PR рекомендуется заполнять)
- Добавьте метку `kubernetes-gitops` к вашему PR

Проверка ДЗ

Данное задание будет проверяться в полуавтоматическом режиме. Не пугайтесь того, что тесты в итоге завершатся неуспешно.

При этом смотрите в лог **Travis**, чтобы понять, действительно ли они дошли до "правильной ошибки", говорящей о том, что дальнейшая проверка будет производиться вручную.

Полезные ссылки

1. [Документация Flux](#)
2. [Документация Helm-Operator](#)
3. [Документация Flagger](#)
4. [Документация Istio](#)

