

# Часть III

1. Знакомство с CI/CD процессом
2. Интеграция с Github
3. Знакомство с Kubernetes
4. Подготовка сервиса для релизов и CI/CD
5. Релиз
6. Проверка работоспособности сервиса

# Чек-лист

Для второй части мастер-класса нам понадобятся:

- ✓ любой терминал SSH
- ✓ консольная утилита git
- ✓ Github персональная учетная запись
- консольная утилита для управления ресурсами Kubernetes

# Материалы и исходный код

<http://github.com/k8s-community>

# Тема 1: Знакомство с CI/CD процессом

# Разработчик делает изменения в приложении



# Разработчик делает Pull Request или Push в репозиторий



## Изменения поступили в репозиторий



На GitHub по изменениям в репозитории активируется хук





## Hook принимает Github Apps (Integration)



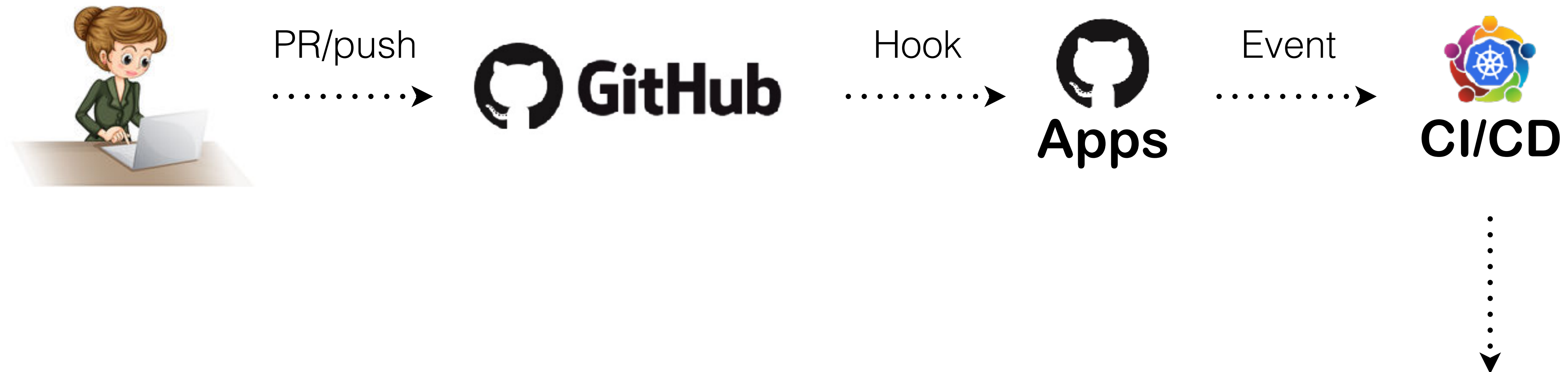
## Github Apps генерирует событие для специального сервиса



## Сервис CI/CD получает событие



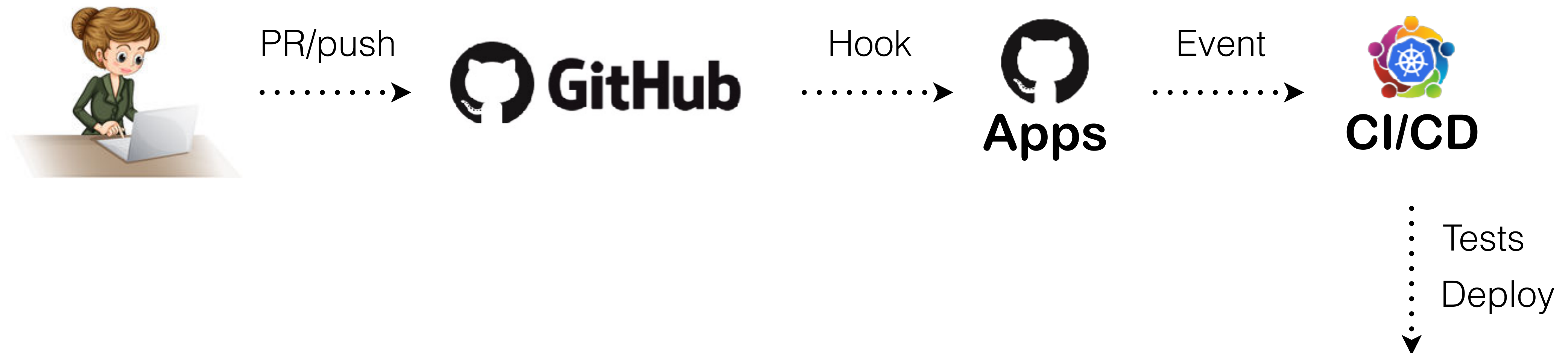
## Сервис CI/CD запускает процессы



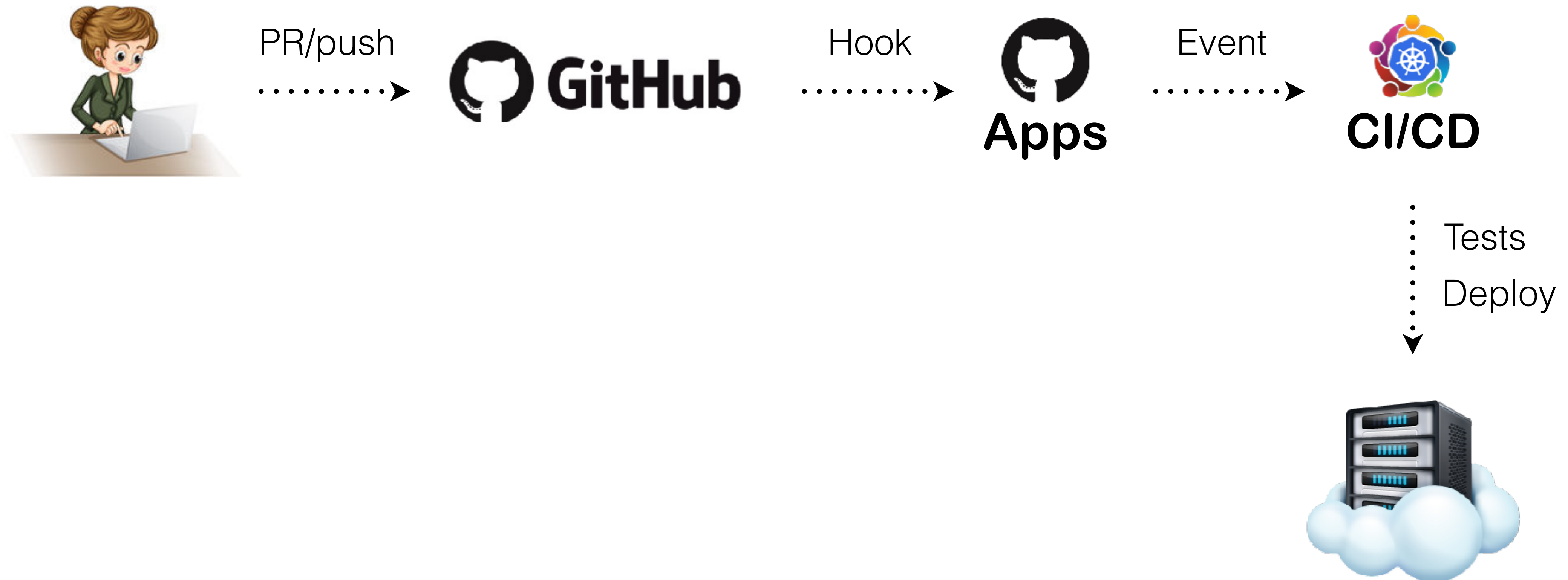
## Сервис CI/CD запускает процессы тестирования



# Сервис CI/CD запускает процессы тестирования и доставки кода



Все процессы выполняются на выделенной для этой роли VM





**make test** — выполняет проверку линтером, форматирование кода и тесты

Makefile:

```
. . .
lint:
    @echo "+ $@"
    @go list -f '{{if len .TestGoFiles}}"golint {{.Dir}}/..."{{end}}' ${GO_LIST_FILES} | xargs -L 1
sh -c

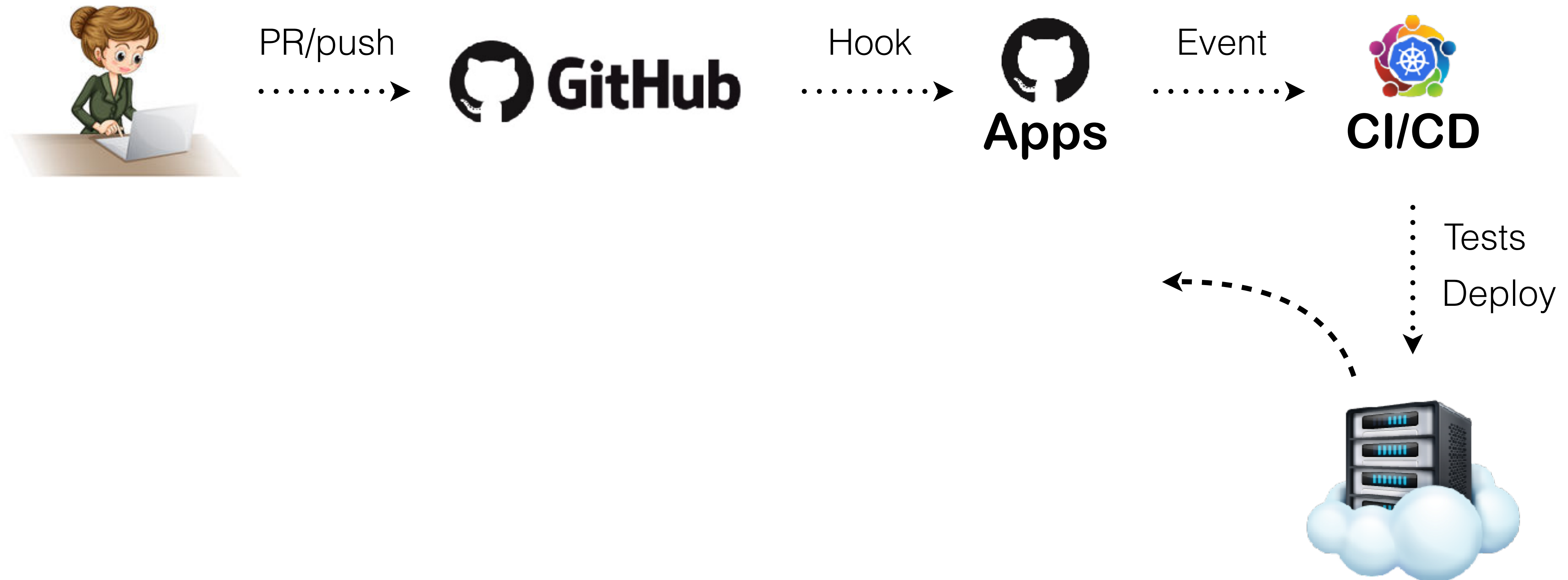
vet:
    @echo "+ $@"
    @go vet ${GO_LIST_FILES}

test: vendor fmt lint vet
    @echo "+ $@"
    @go test -v -race -tags "$(BUILDTAGS) cgo" ${GO_LIST_FILES}

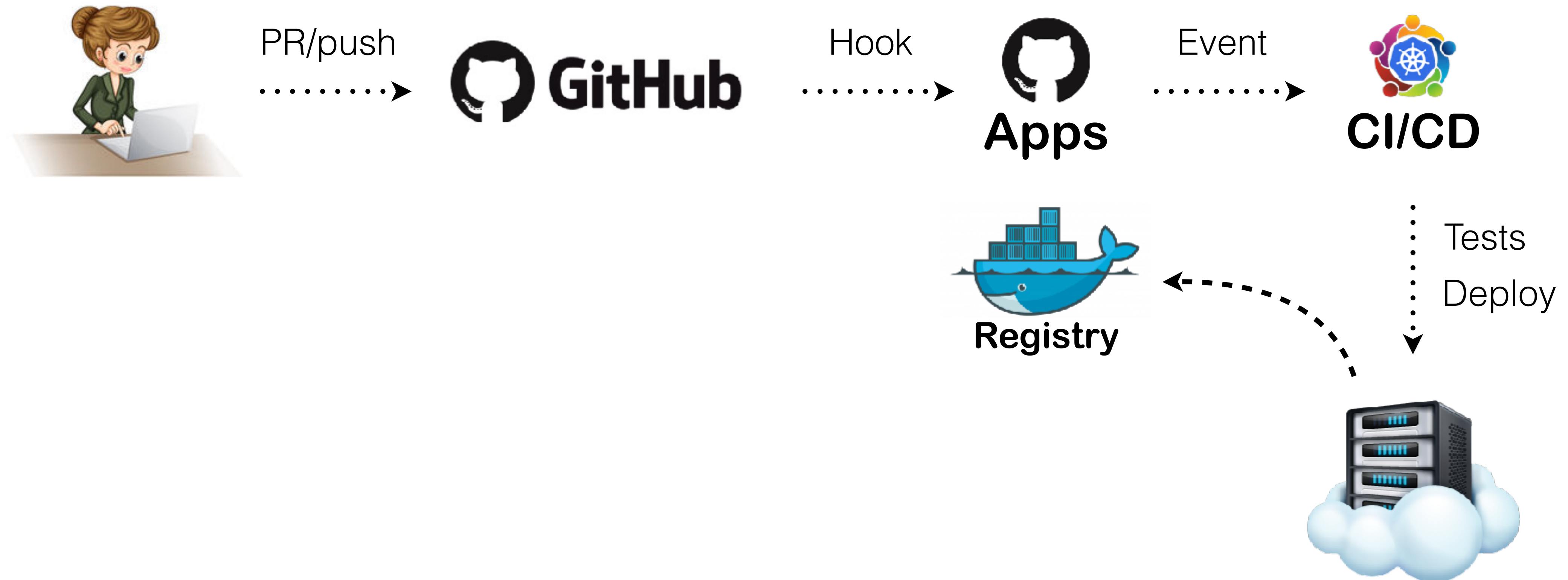
cover:
. . .
```



После успешной сборки Docker Image доставляется в хранилище



# В качестве хранилища для Docker Images используется Docker Registry



**make push** — выполняет сборку Docker Image и доставляет его в Docker Registry

Makefile:

. . .

container: build

docker build --pull -t \$(CONTAINER\_IMAGE):\$(RELEASE) .

push: container

docker push \$(CONTAINER\_IMAGE):\$(RELEASE)

deploy: push

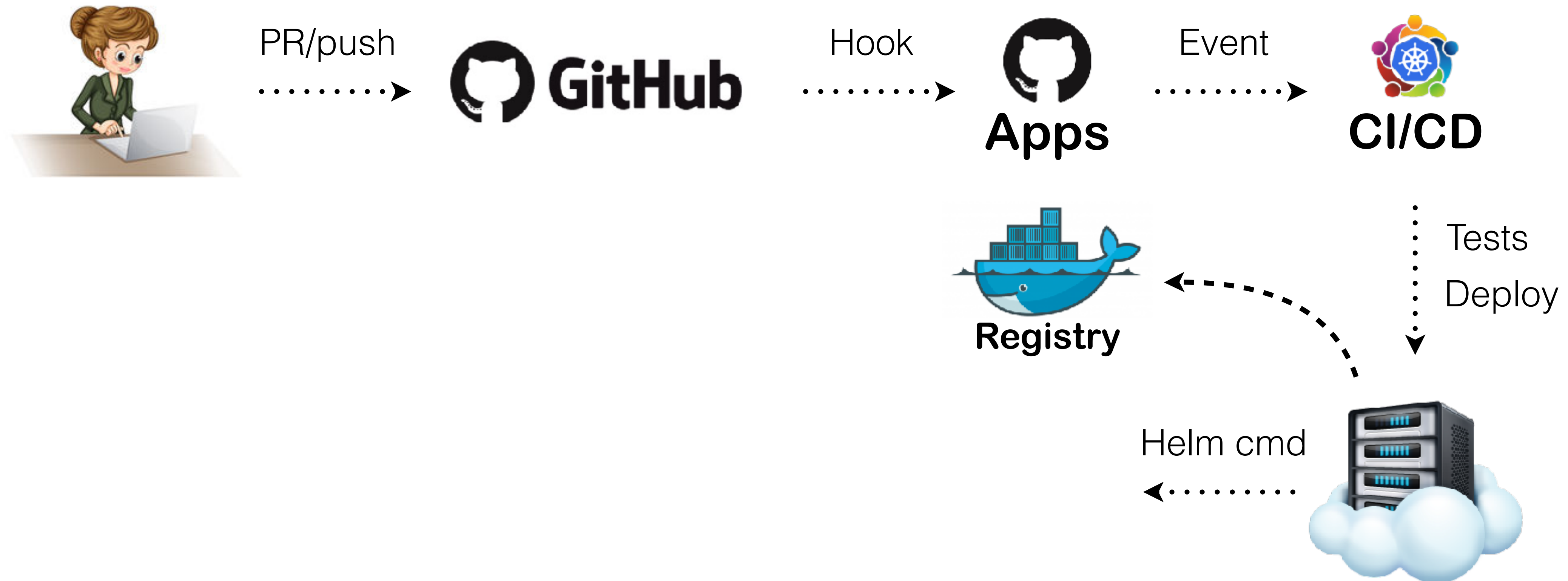
helm upgrade \${CONTAINER\_NAME} -f charts/\${VALUES}.yaml \  
charts --kube-context \${KUBE\_CONTEXT} \  
--namespace \${NAMESPACE} --version=\${RELEASE} -i --wait

fmt:

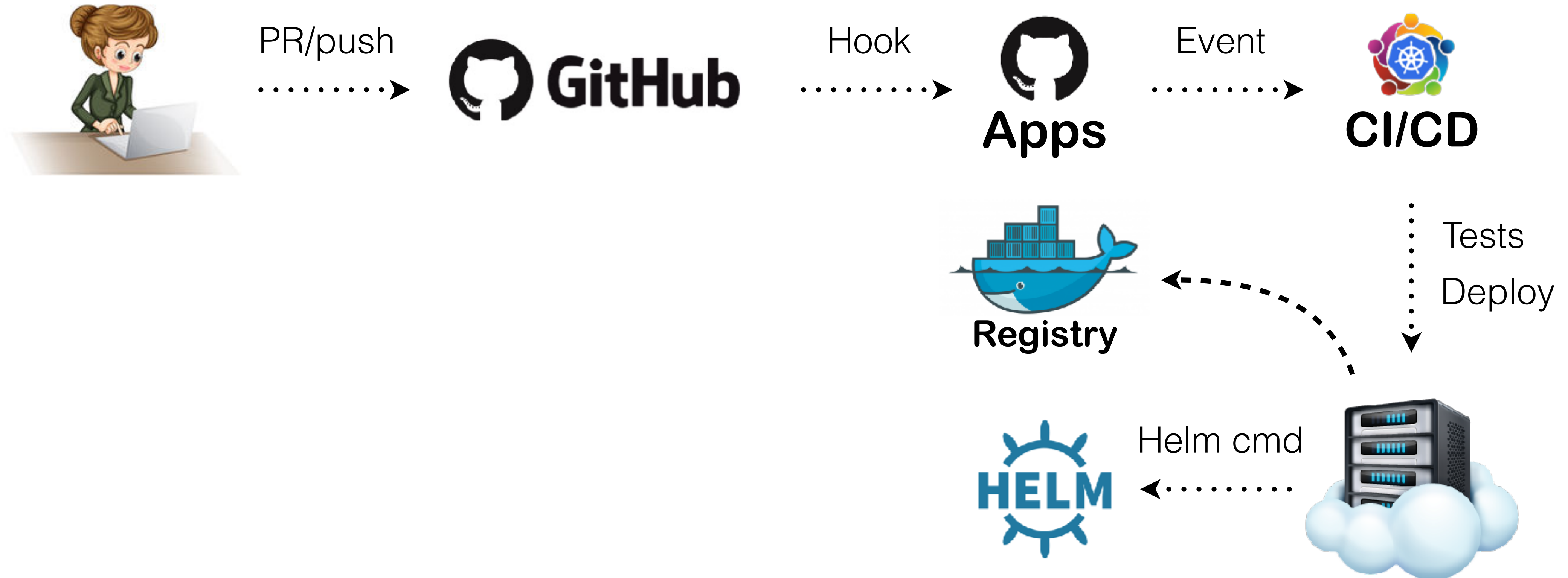
@echo "+ \$@"

. . .

В качестве пакетного менеджера релизов используется Helm

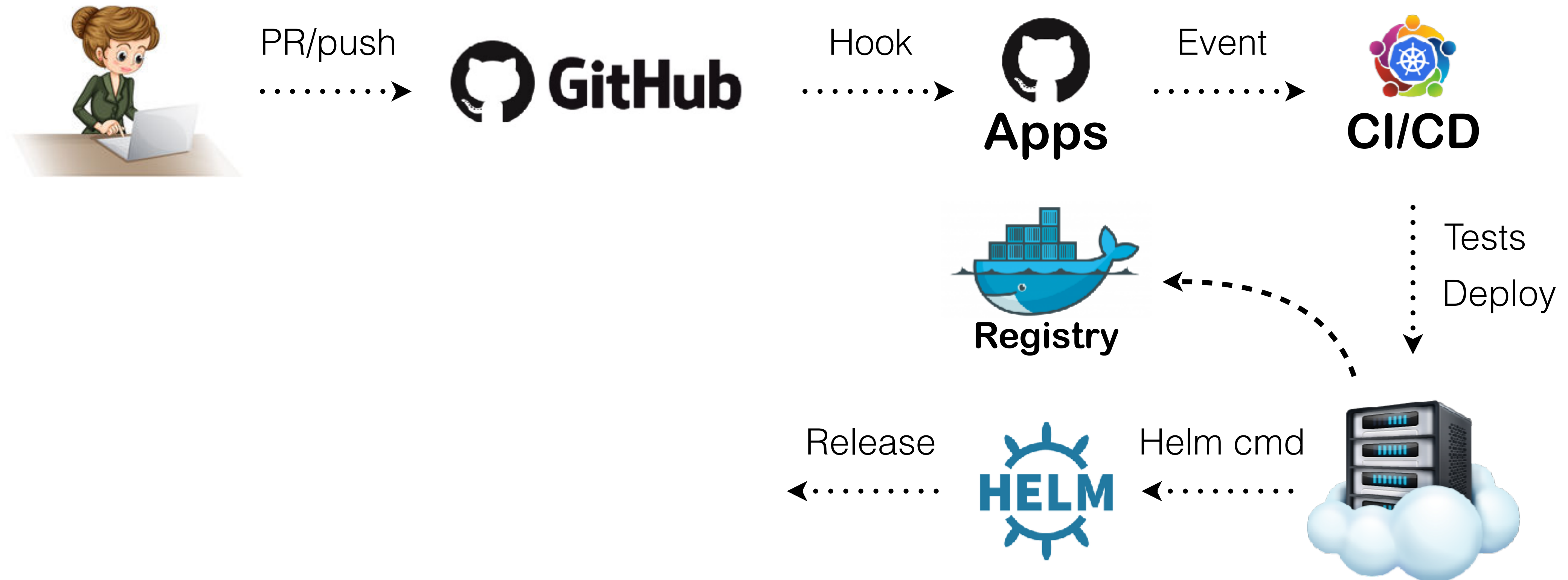


# Helm Client принимает команды и передаёт их своему агенту

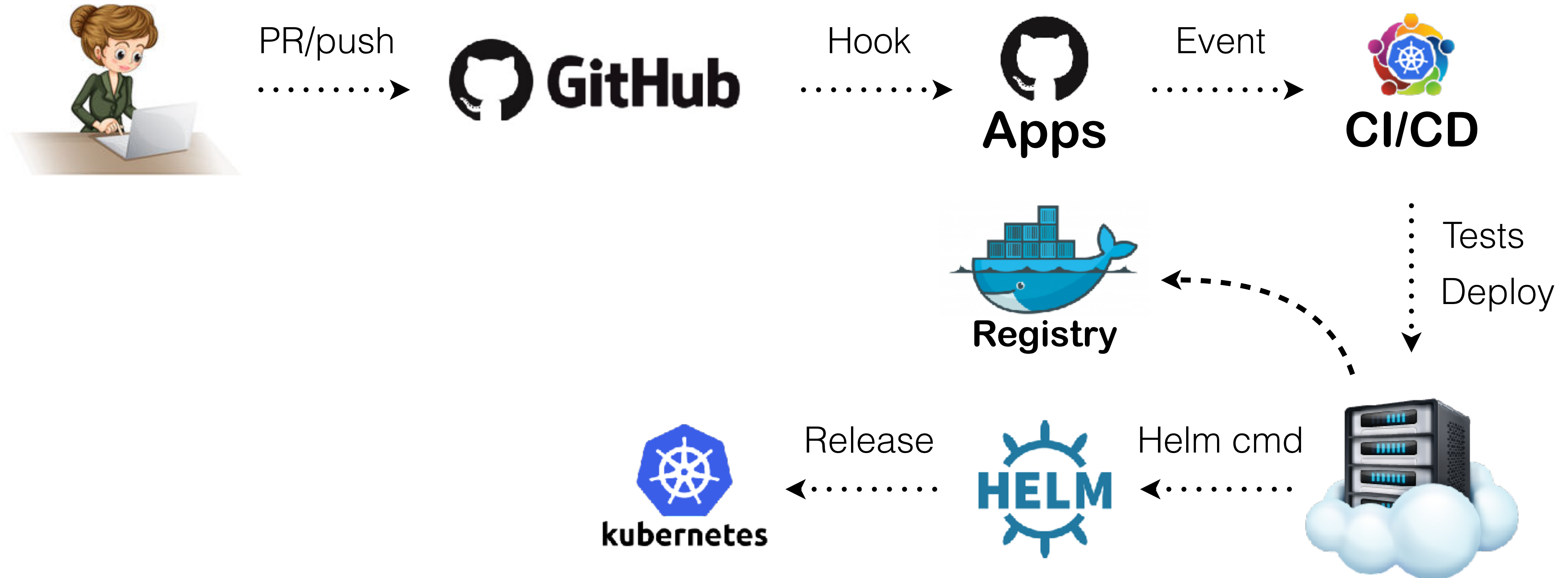




## Основная задача Helm — провести релиз сервиса



# Helm агент в кластере Kubernetes принимает команды Helm



# **make deploy** — выполняет доставку сервиса в среду Kubernetes

## Makefile:

```
. . .
container: build
    docker build --pull -t $(CONTAINER_IMAGE):$(RELEASE) .

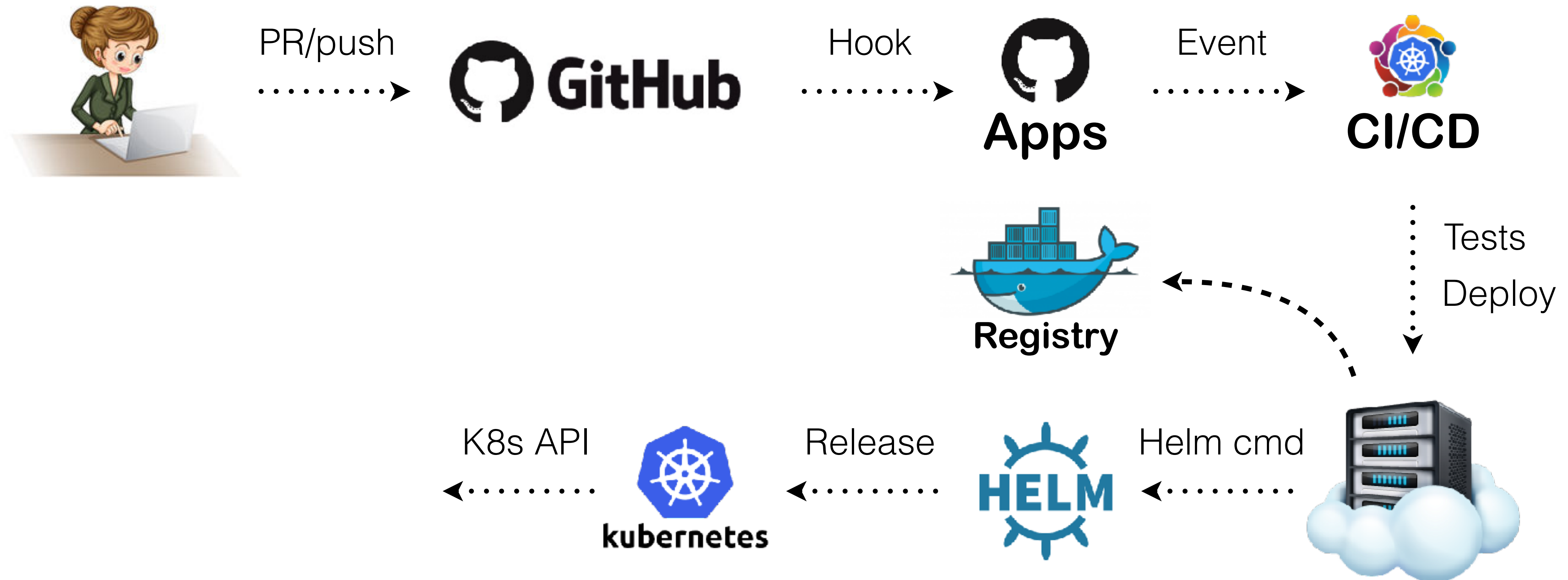
push: container
    docker push $(CONTAINER_IMAGE):$(RELEASE)

deploy: push
    helm upgrade ${CONTAINER_NAME} -f charts/${VALUES}.yaml \
    charts --kube-context ${KUBE_CONTEXT} \
    --namespace ${NAMESPACE} --version=${RELEASE} -i --wait

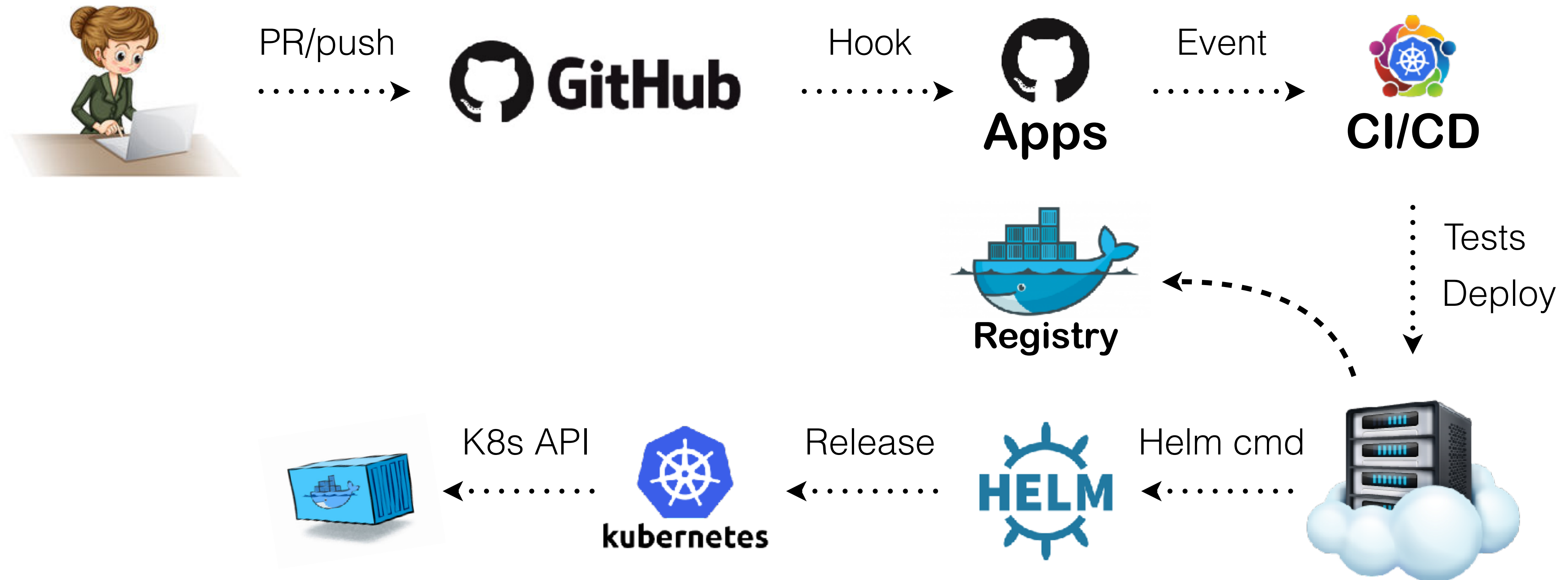
fmt:
    @echo "+ $@"
. . .
```



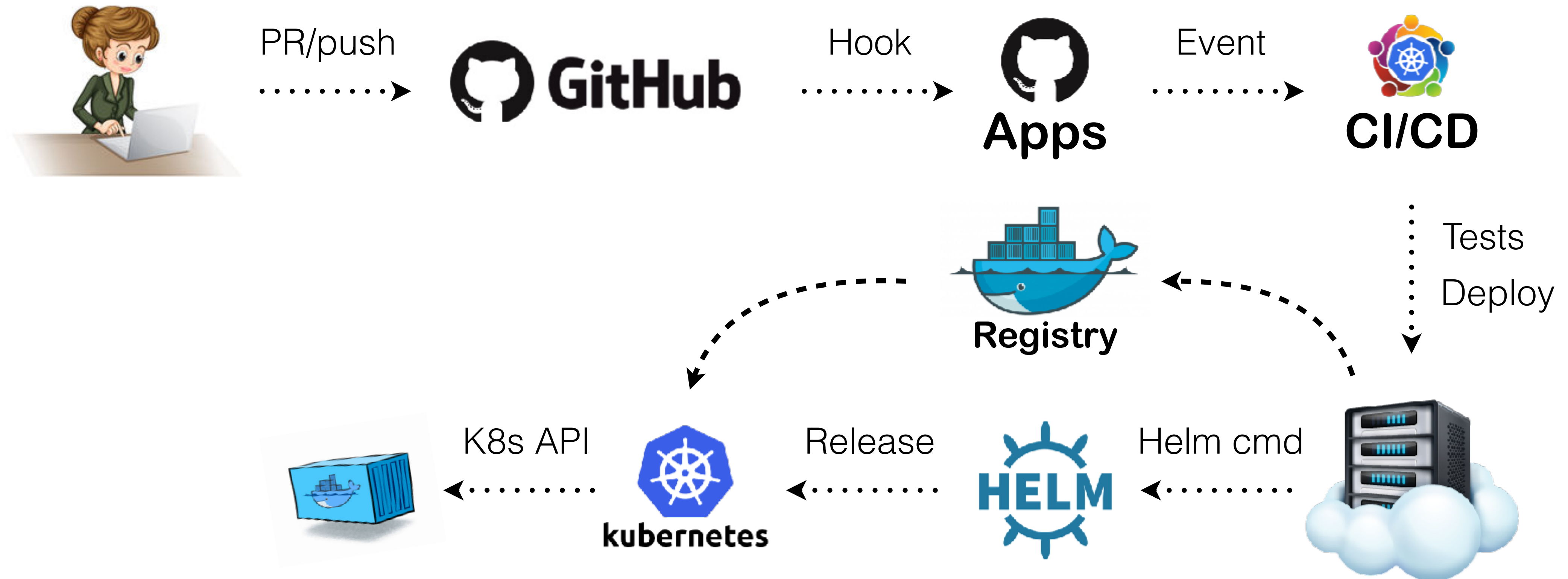
# С помощью API создаются компоненты из Helm Charts в Kubernetes



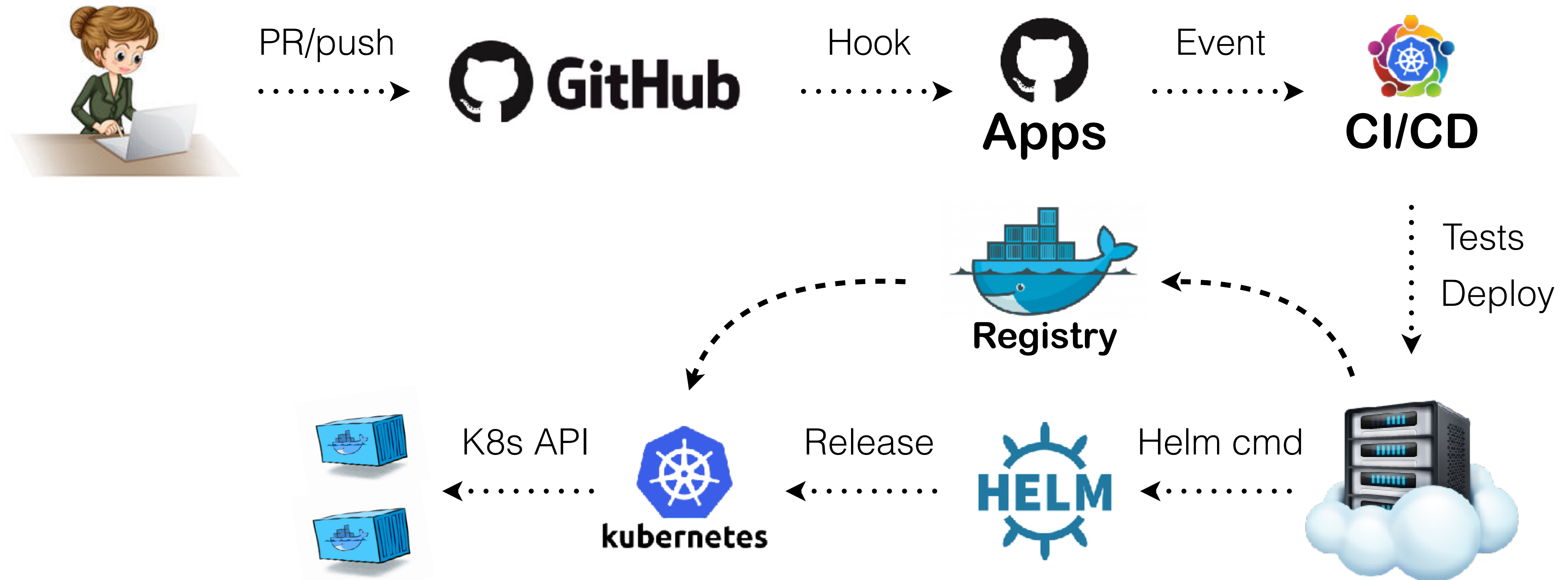
В итоге создаётся Pod с сервисом в контейнере в Kubernetes



## Kubernetes при создании Pod берёт Docker image из хранилища

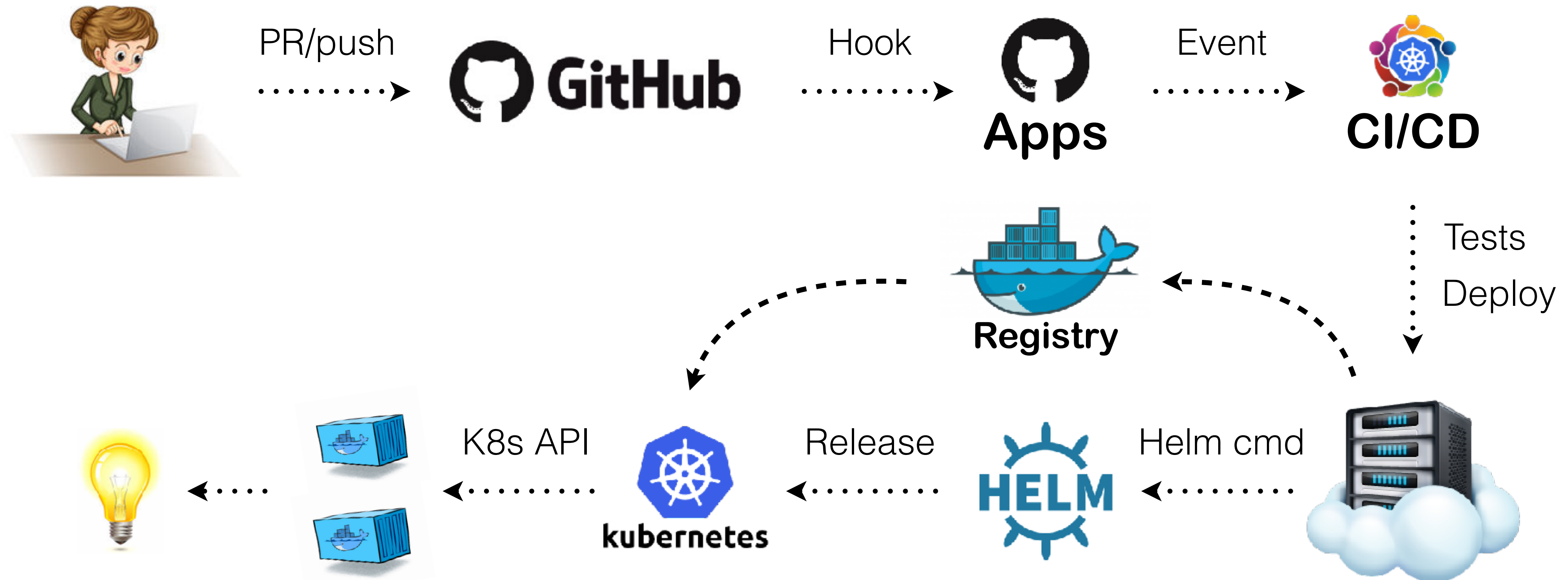


Количество Pods и контейнеров в них зависит от типа сервиса



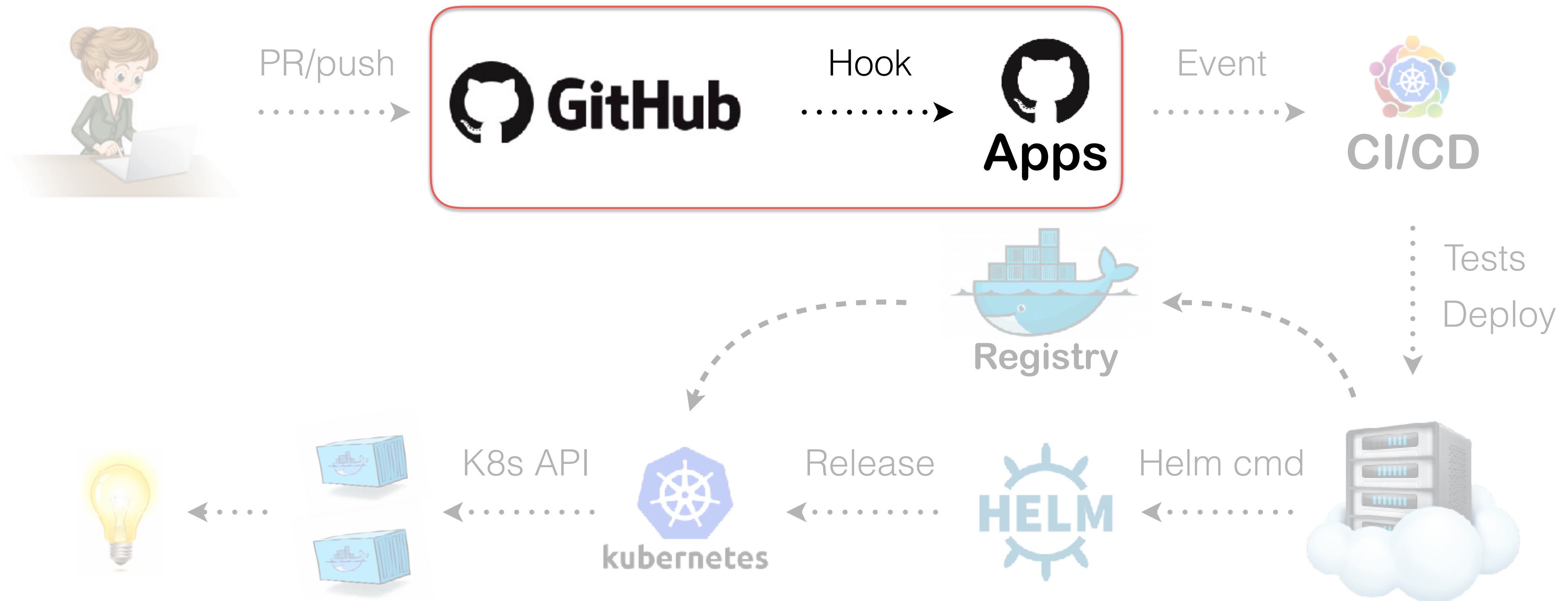


Когда все компоненты загружены, сервис готов к работе



# Тема 2: Интеграция с Github

## Активируем интеграцию с GitHub, используя GitHub Apps



# Создаём новый репозиторий на Github с именем `myapp`

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: takama / Repository name: myapp ✓

Great repository names are short and memorable. Need inspiration? How about `cuddly-sniffle`.

Description (optional)

☒ Public  
Anyone can see this repository. You choose who can commit.

☐ Private  
You choose who can see and commit to this repository.

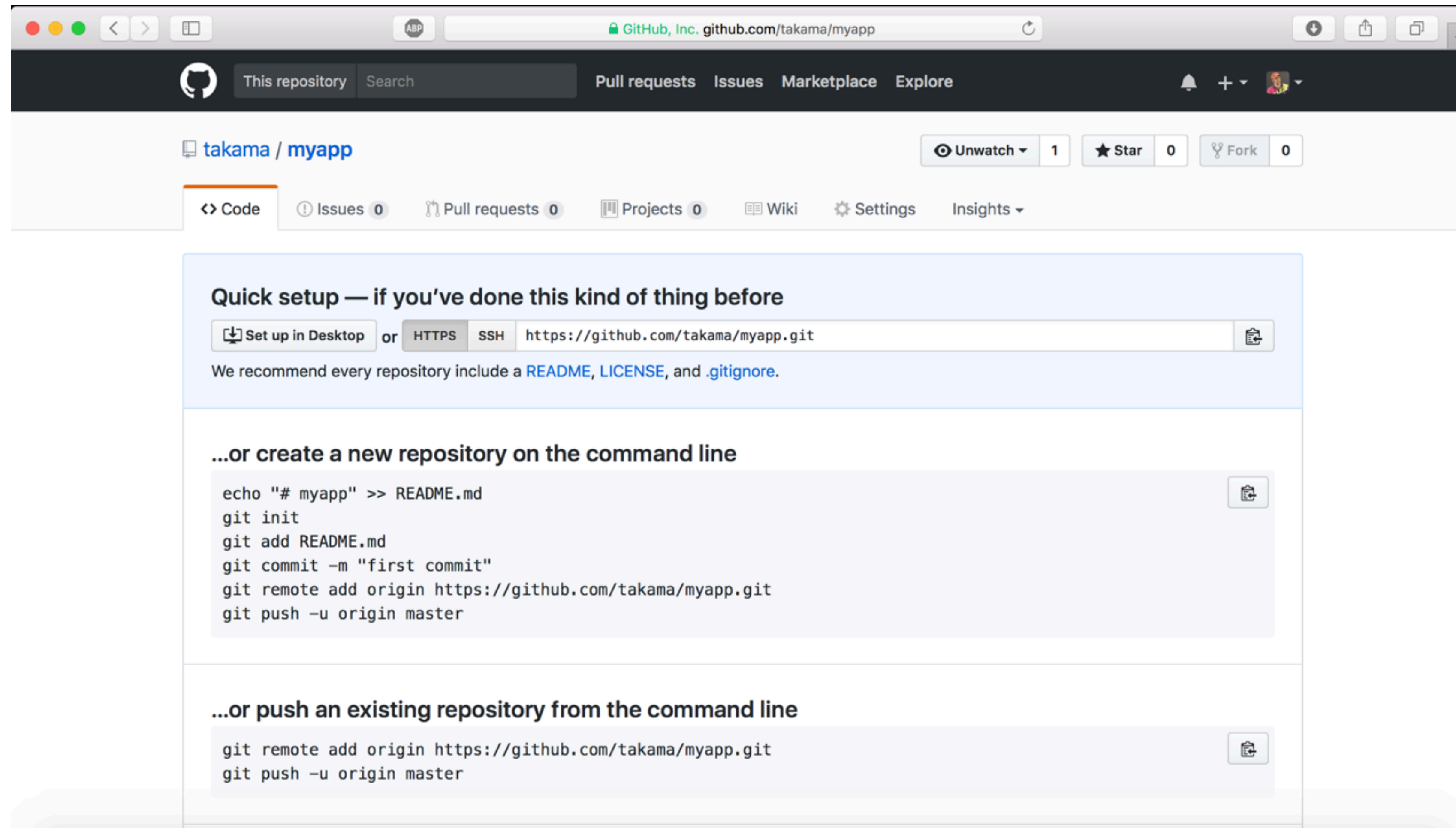
☐ Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None ⓘ

Create repository



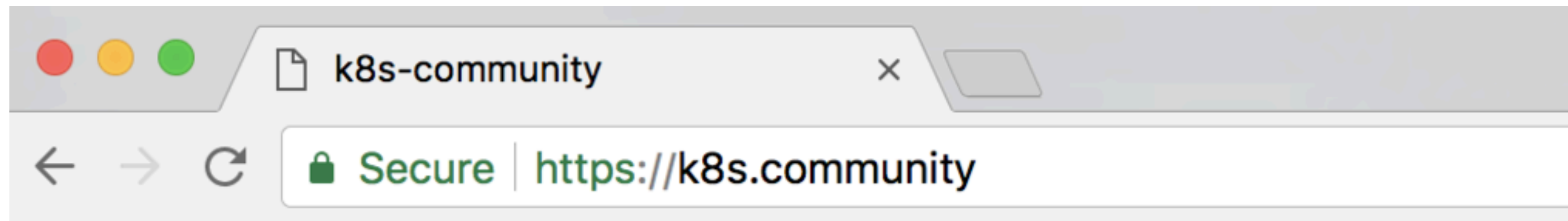
# Создаём новый репозиторий на Github с именем `myapp`



Затем связываем наш сгенерённый `myapp` с удалённым на Github

```
$ git init
$ git add --all
$ git commit -m "Initial commit"
$ git remote add origin https://github.com/user_name/myapp.git
$ git push -u origin master
```

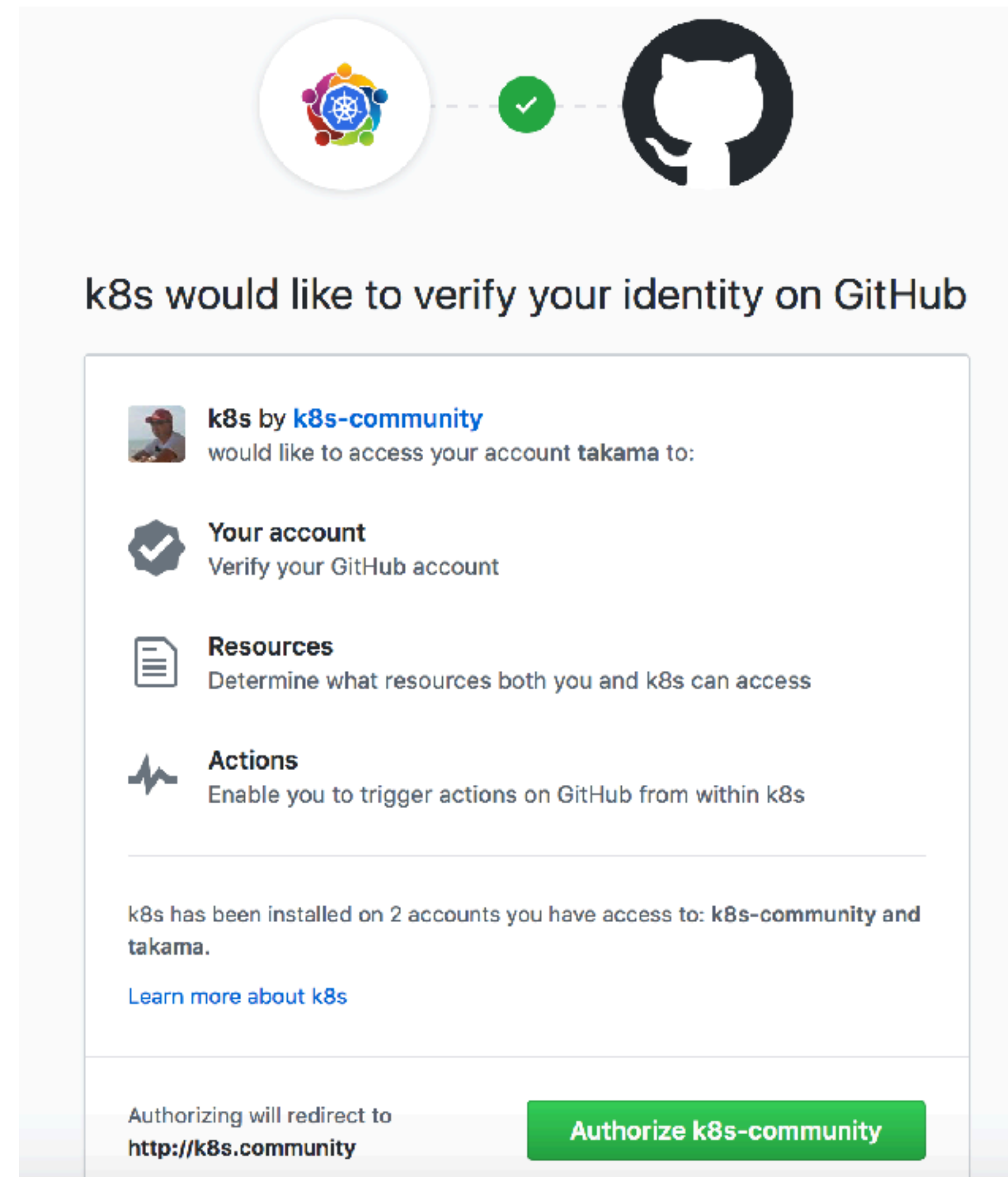
На странице <https://k8s.community> проходим по ссылке (пиктограмма активации)



Для регистрации в мастер-классе пройдите по ссылке на GitHub:



Далее вы попадаете на Github для авторизации



После авторизации на Github, на странице <https://k8s.community> будет отображена конфигурация для настройки доступа.



Вы авторизованы как **takama**.

Окружение в Kubernetes успешно создано.

**Шаг №2:** настройка локальной среды. Пожалуйста, следуйте [инструкции по подготовке окружения](#).

Ваш токен для настроек в рамках шага 2: XXXXXXXX

**Шаг №3:** включаем интеграцию с GitHub [здесь](#).



Используем ссылку внизу для выбора репозитория, для которого будет активирован CI/CD.



Вы авторизованы как **takama**.

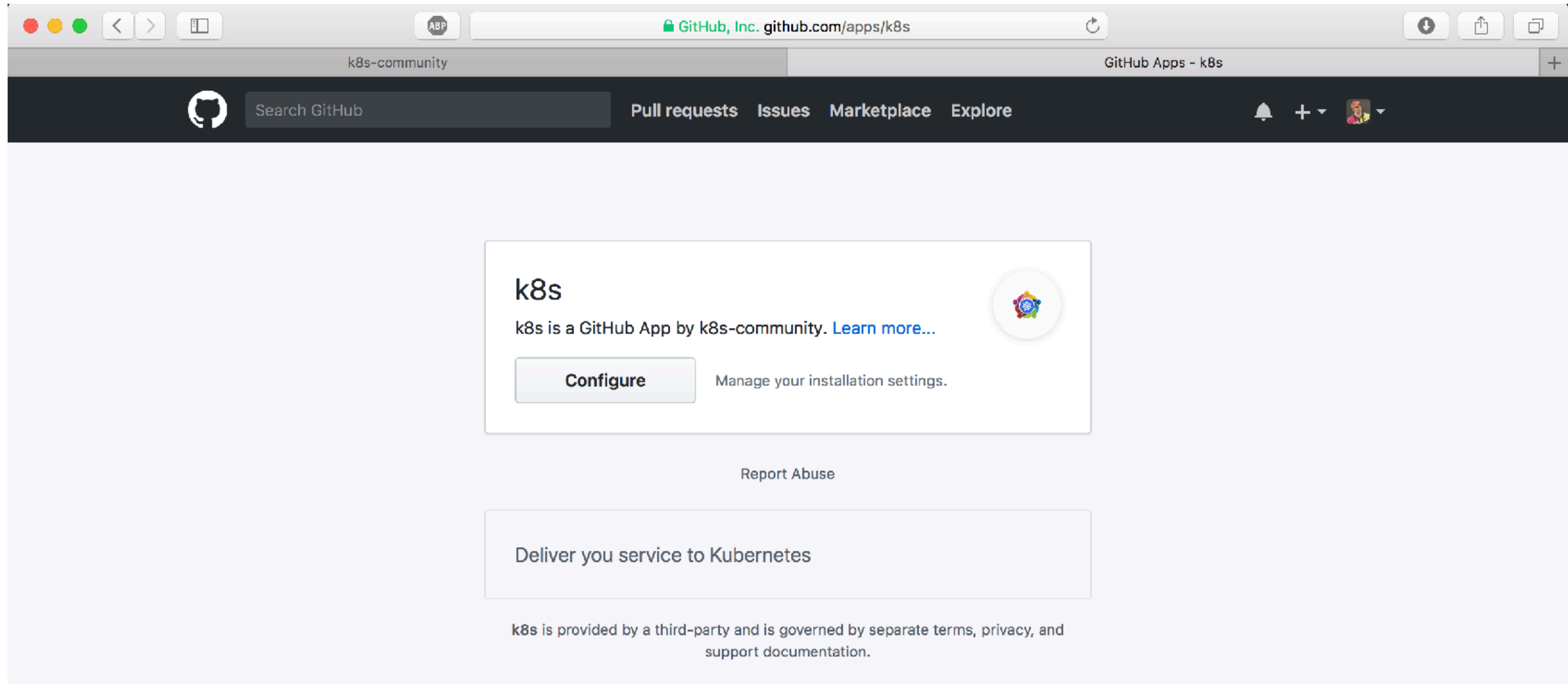
Окружение в Kubernetes успешно создано.

**Шаг №2:** настройка локальной среды. Пожалуйста, следуйте [инструкции по подготовке окружения](#).

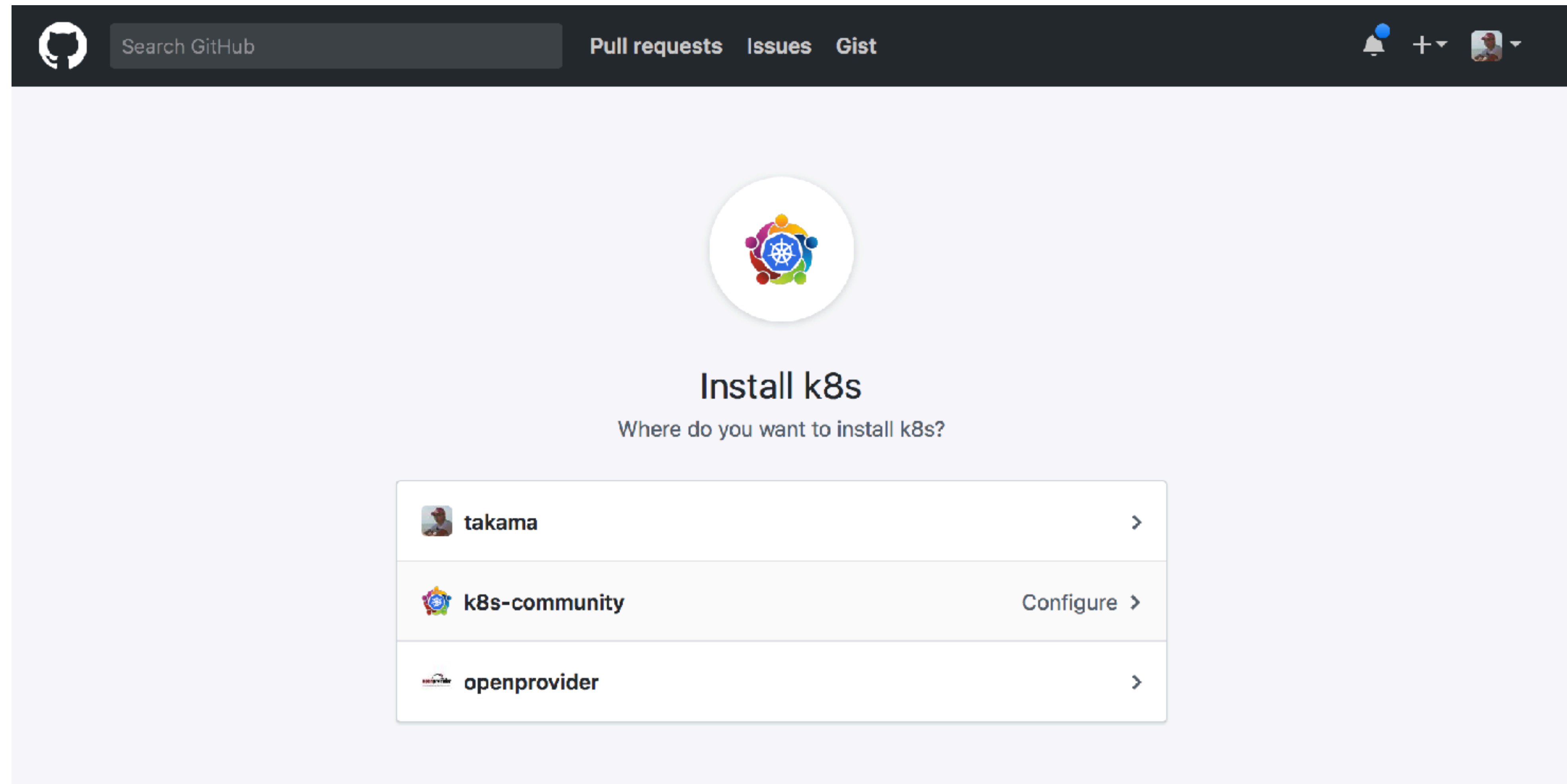
Ваш токен для настроек в рамках шага 2: XXXXXXXX

**Шаг №3:** включаем интеграцию с GitHub [здесь](#).

На странице GitHub выбираем предложенную настройку **k8s** интеграции.




Предлагается выбрать аккаунт, где у вас находится [myapp](#).





Возможно, вам будет предложено подтвердить интеграцию для [myapp](#).



Confirm password to continue


Password [Forgot password?](#)

Confirm password

Tip: You are entering [sudo mode](#). We won't ask for your password again for a few hours.

[Terms](#) [Privacy](#) [Security](#) [Contact GitHub](#)

После выбора аккаунта выбираем репозиторий **myapp**.

 **k8s**  
🕒 Installed 4 months ago    👤 Developed by [k8s-community](#)    🔗 <http://k8s.community>

---

Deliver you service to Kubernetes

---

### Permissions

---

- ✓ **Read** access to code

---

- ✓ **Read** access to administration, metadata, and pull requests

---


- ✓ **Read** and **write** access to commit statuses

---

### Repository access

☐ **All repositories**  
This applies to all current *and* future repositories.

☒ **Only select repositories**

 m

Selected 0 repositories

После выбора аккаунта выбираем репозиторий [myapp](#).

### Repository access

☐ **All repositories**  
This applies to all current *and* future repositories.

☒ **Only select repositories**

Select repositories

Selected 1 repository

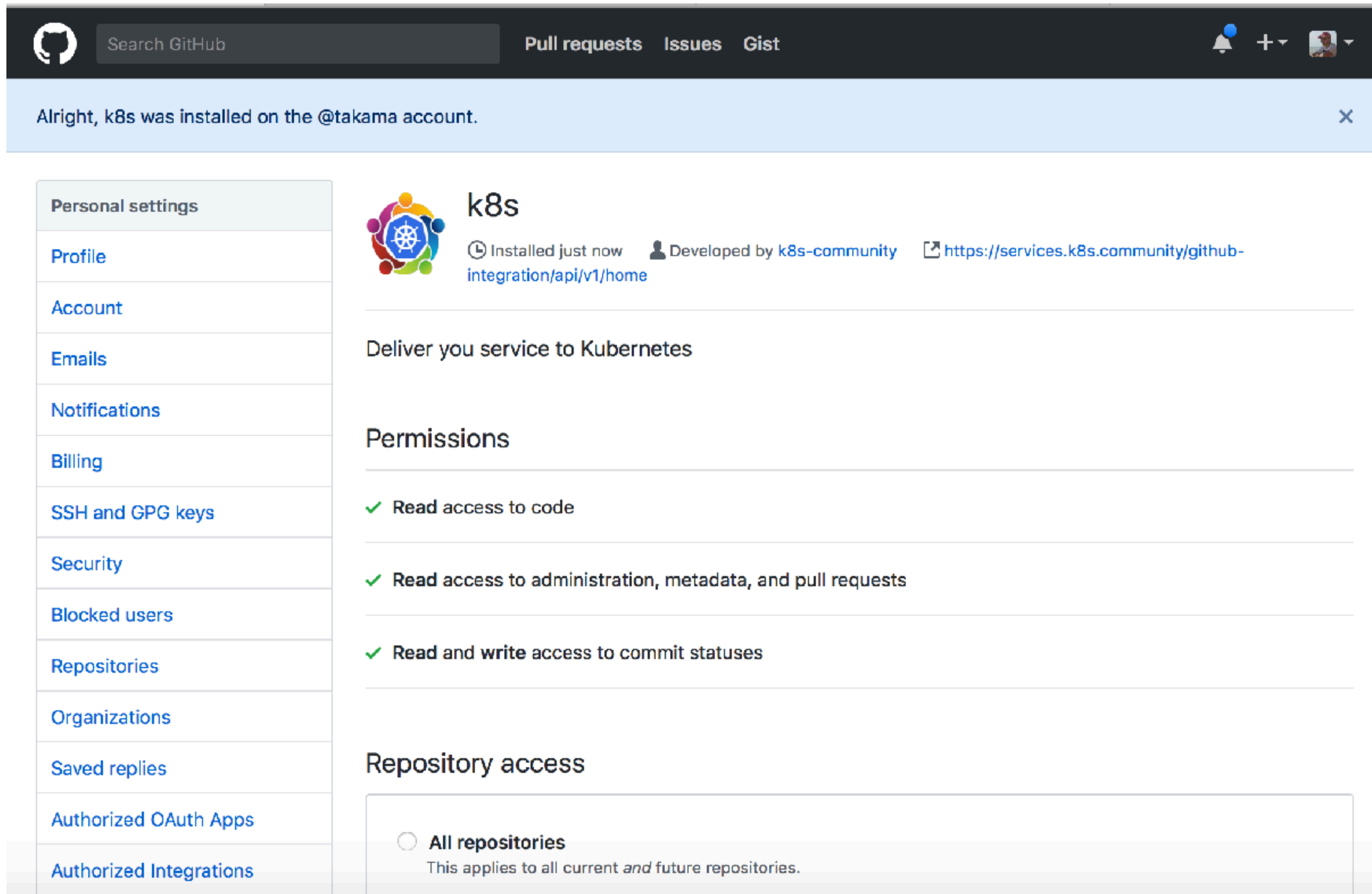
 takama/myapp



Save

Cancel

После успешной настройки будут отображены параметры интеграции.



The screenshot shows the GitHub web interface. At the top, there's a dark navigation bar with the GitHub logo, a search bar, and links for Pull requests, Issues, and Gist. A light blue notification banner at the top states: "Alright, k8s was installed on the @takama account." Below this, on the left, is a sidebar with "Personal settings" selected, listing options like Profile, Account, Emails, Notifications, Billing, SSH and GPG keys, Security, Blocked users, Repositories, Organizations, Saved replies, Authorized OAuth Apps, and Authorized Integrations. The main content area displays the "k8s" integration details. It includes the k8s logo, the text "Installed just now", "Developed by k8s-community", and a link to the integration API. Below this, a section titled "Deliver your service to Kubernetes" is followed by a "Permissions" section. The permissions listed are: "Read access to code", "Read access to administration, metadata, and pull requests", and "Read and write access to commit statuses". At the bottom, the "Repository access" section shows the "All repositories" option selected, with a note: "This applies to all current and future repositories."

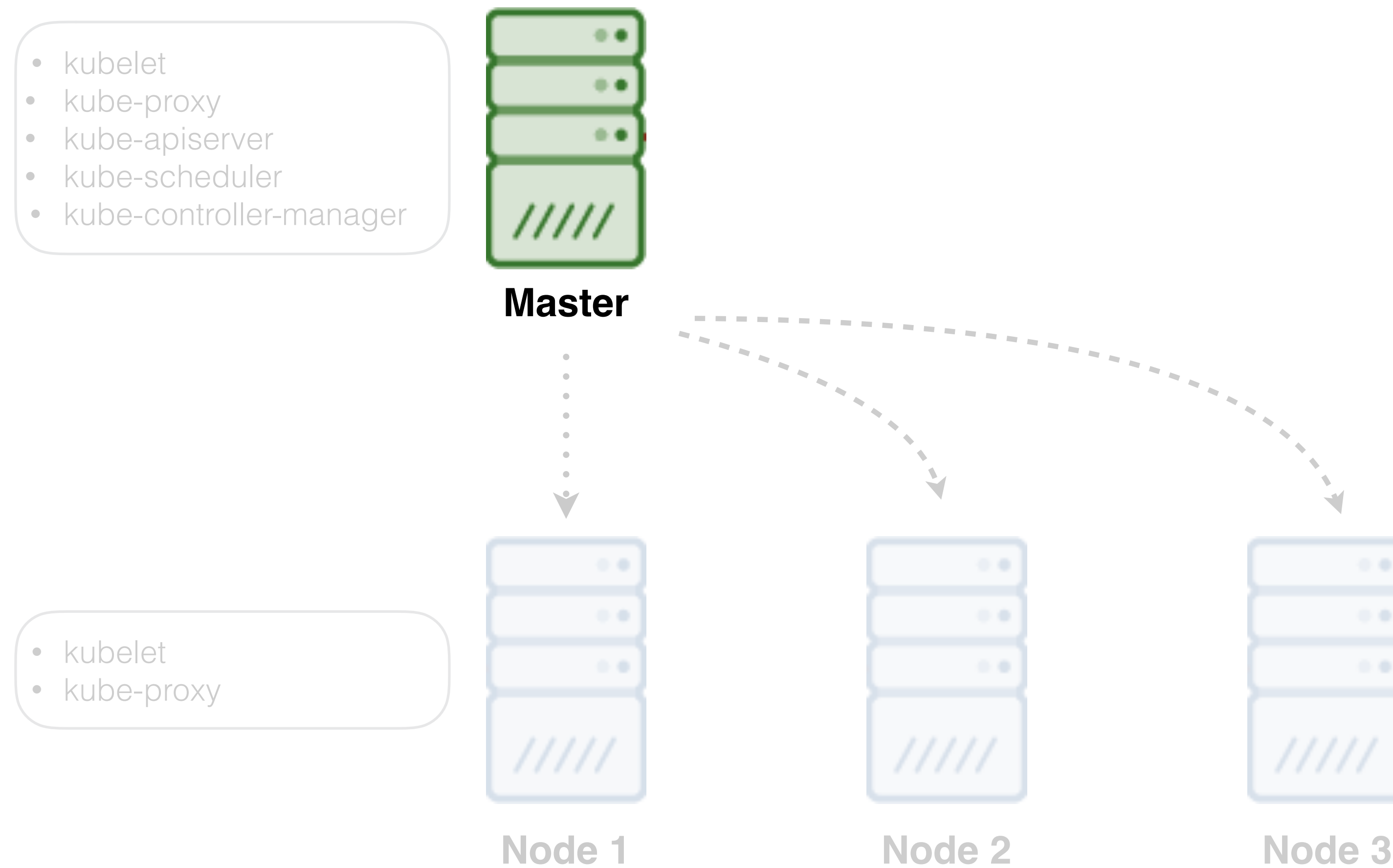
# Тема 3: Знакомство с Kubernetes

# Настройка

Для последующих шагов нам необходимо выполнить все настройки:

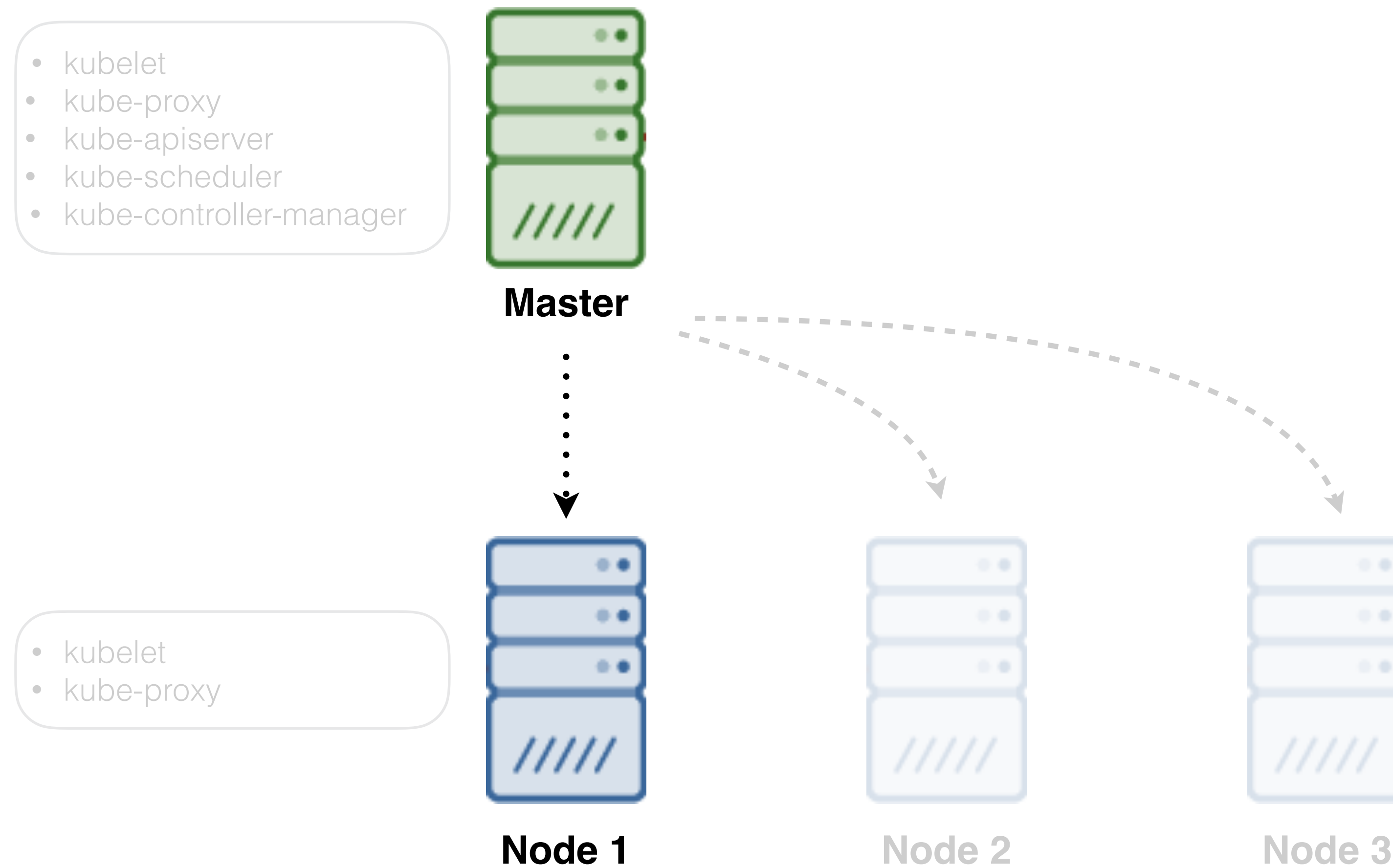
- <https://github.com/k8s-community/k8s-workshop-ru>
- Выбрать раздел “03-part-III-setup”
- Согласно своей OS (OSX, Linux, Windows) выполнить рекомендуемую настройку
- Если в процессе настройки возникают вопросы - не стесняйтесь задавать.

Настройка: <https://github.com/k8s-community/k8s-workshop-ru> 03-part-III-setup



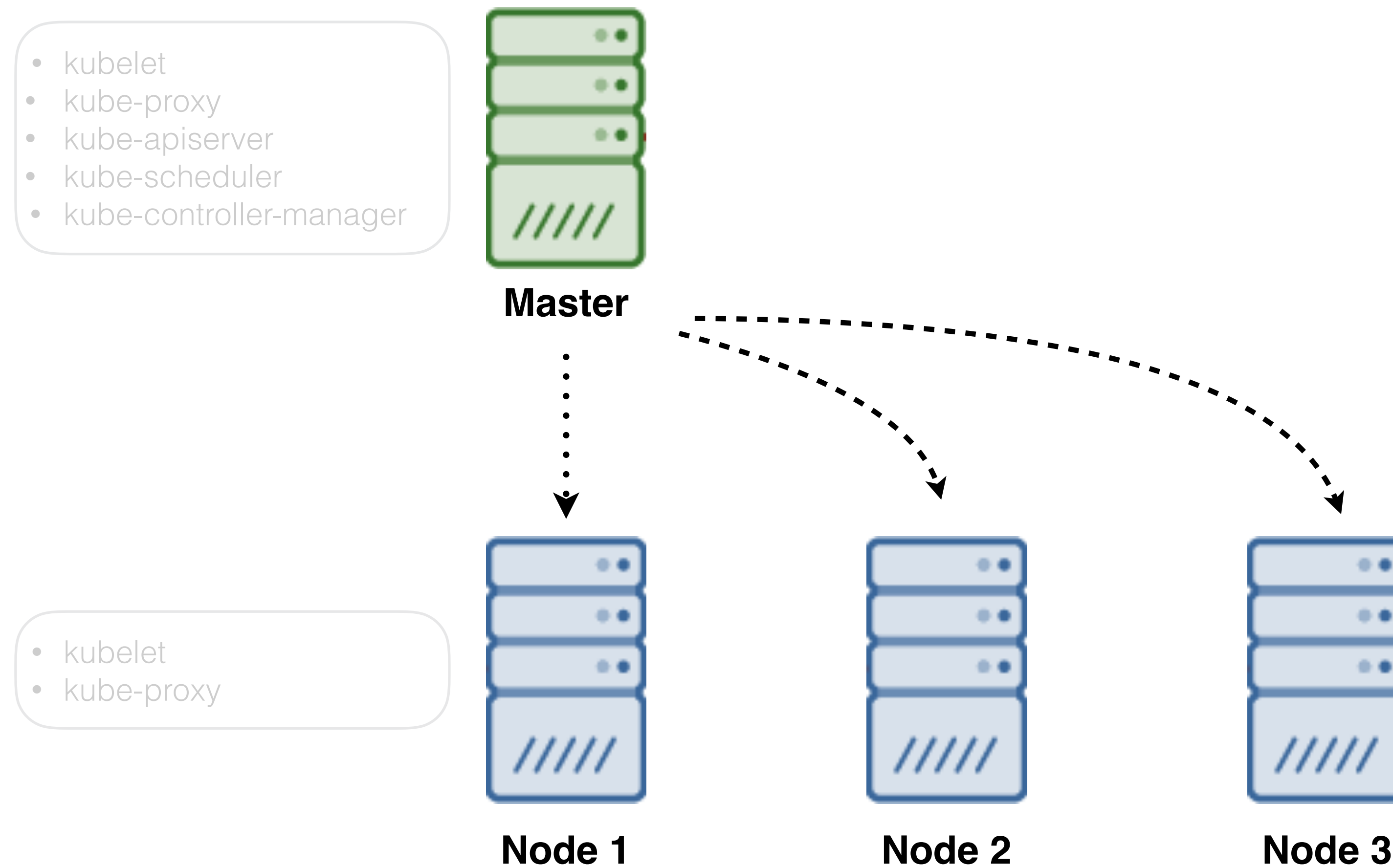


Настройка: <https://github.com/k8s-community/k8s-workshop-ru> 03-part-III-setup

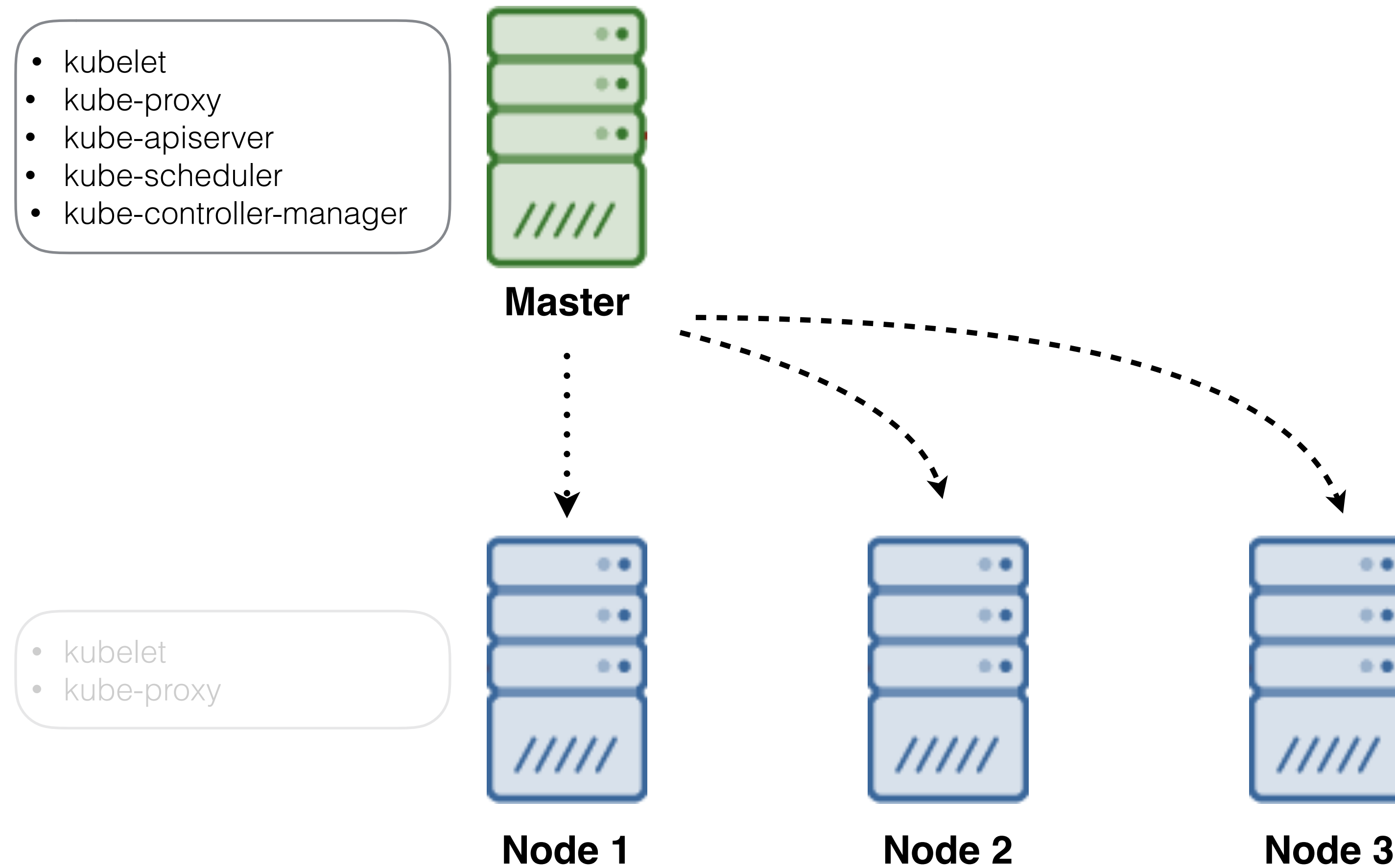




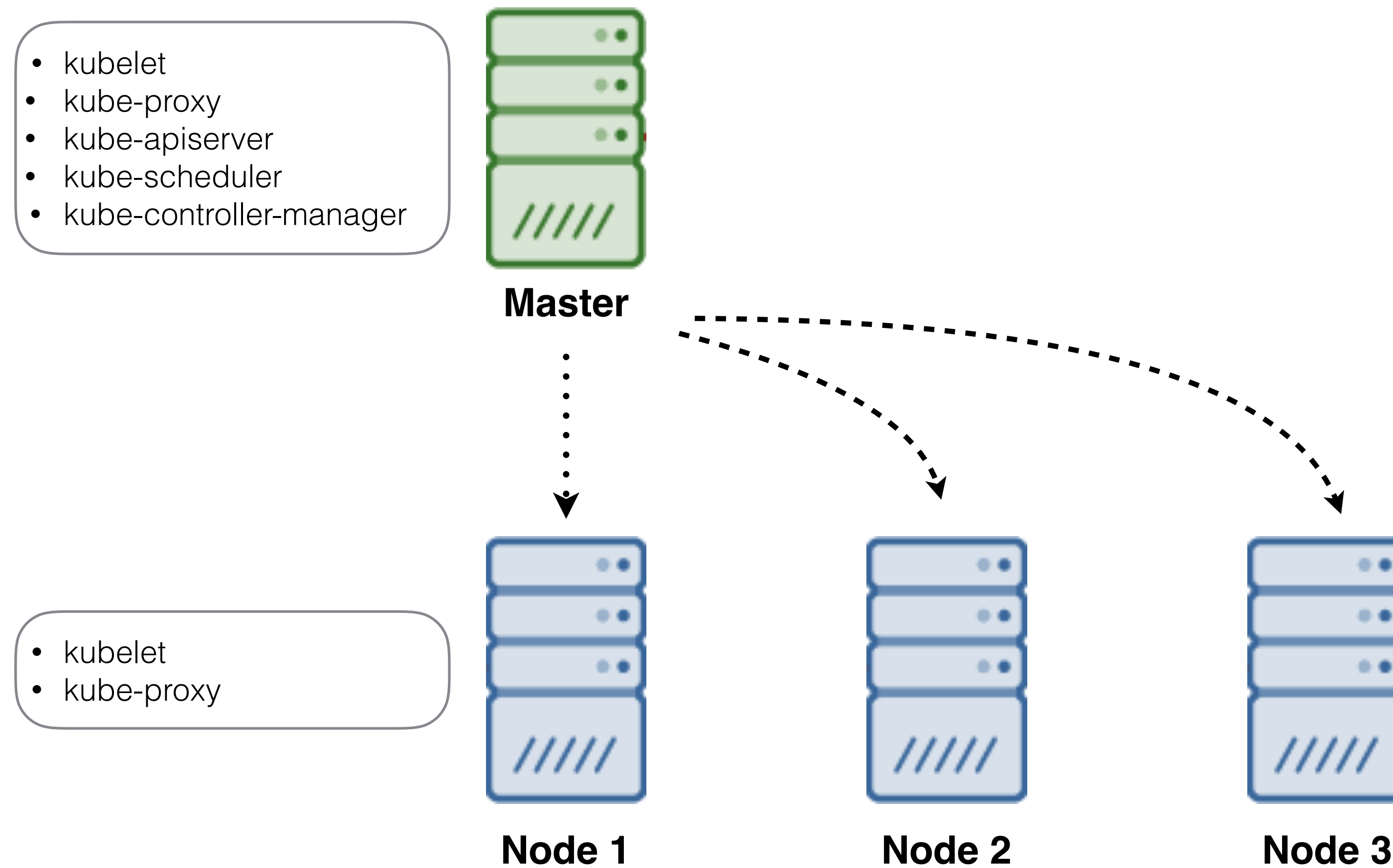
Настройка: <https://github.com/k8s-community/k8s-workshop-ru> 03-part-III-setup



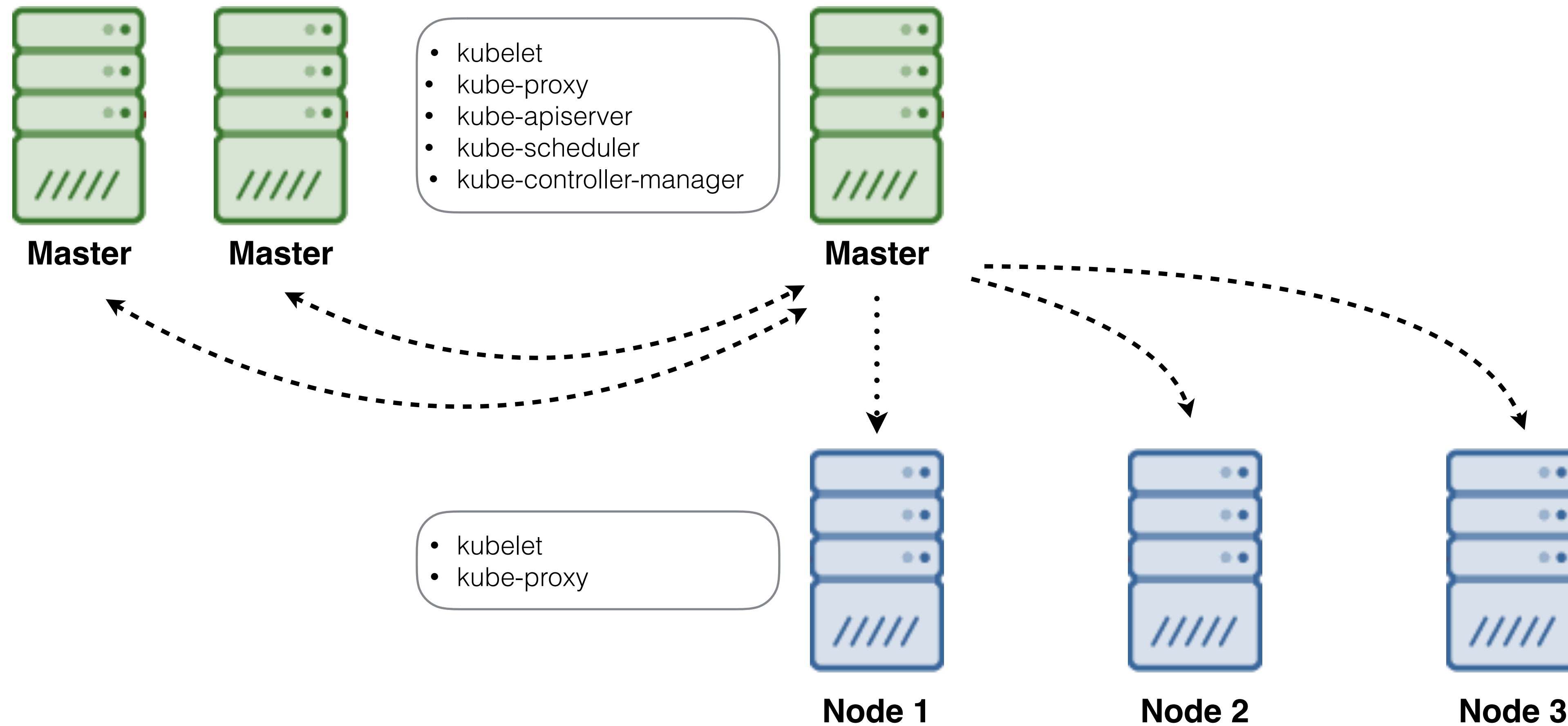
Настройка: <https://github.com/k8s-community/k8s-workshop-ru> 03-part-III-setup



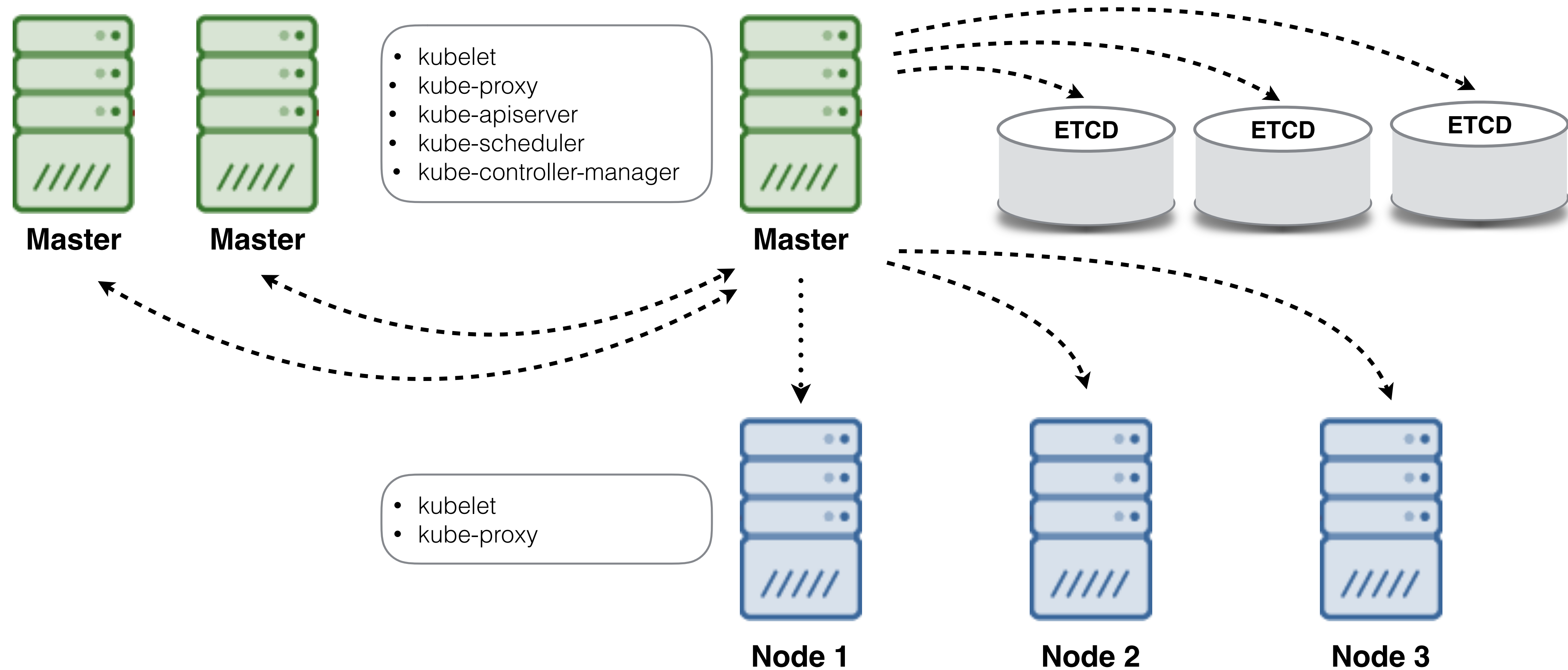
Настройка: <https://github.com/k8s-community/k8s-workshop-ru> 03-part-III-setup



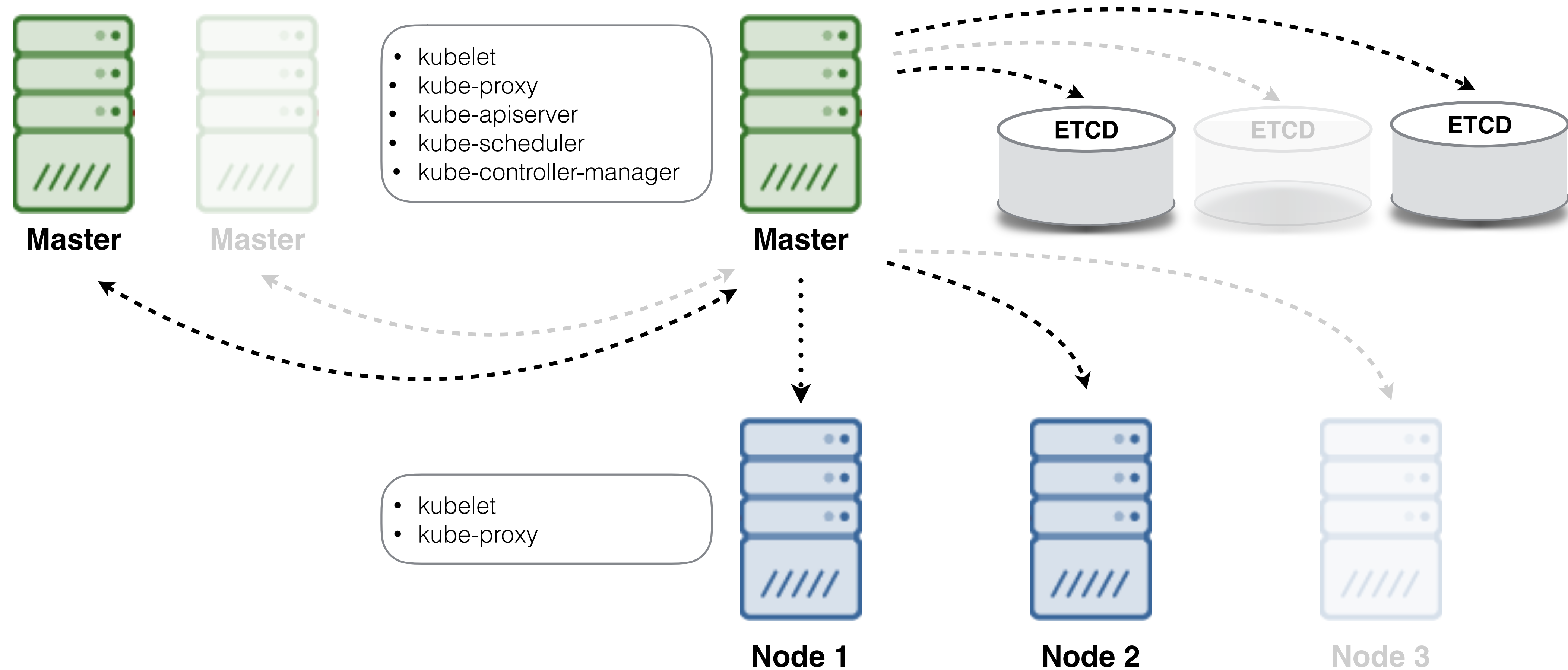
Настройка: <https://github.com/k8s-community/k8s-workshop-ru> 03-part-III-setup



Настройка: <https://github.com/k8s-community/k8s-workshop-ru> 03-part-III-setup

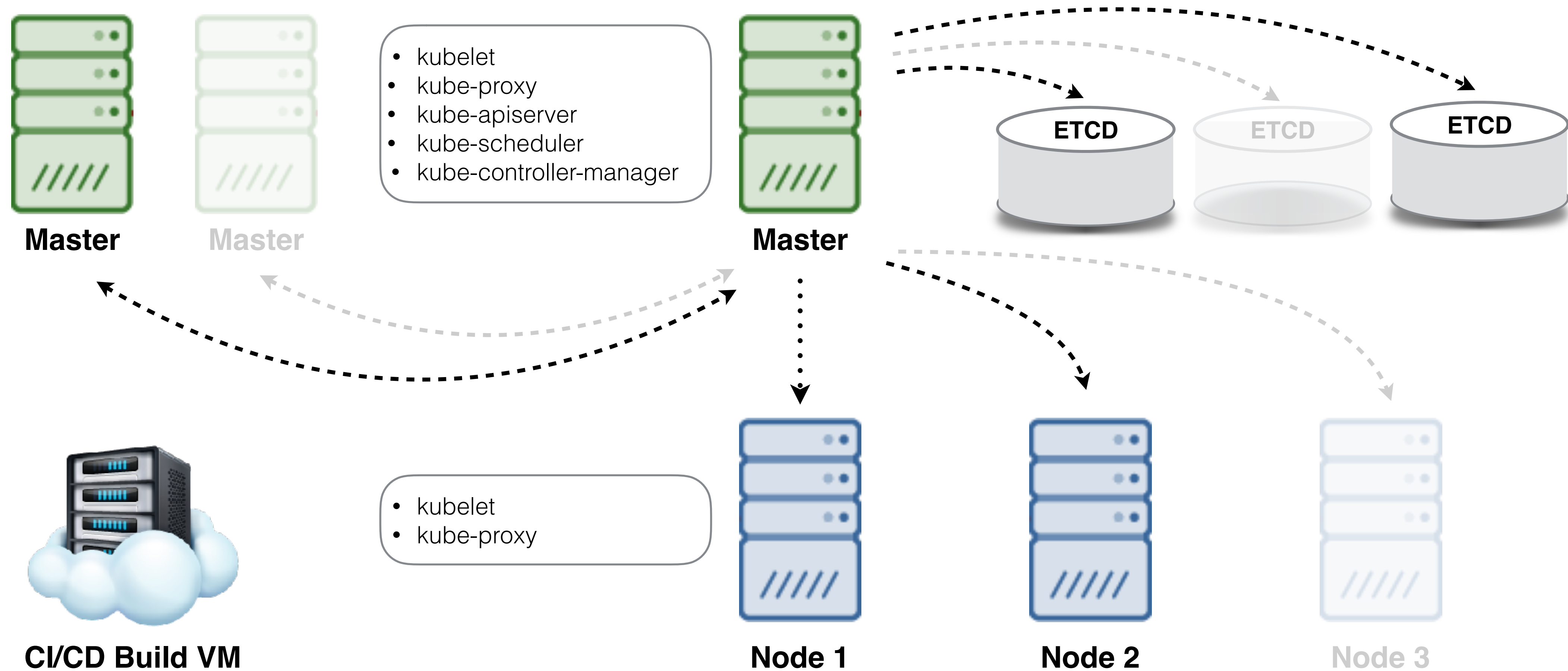


Настройка: <https://github.com/k8s-community/k8s-workshop-ru> 03-part-III-setup





Настройка: <https://github.com/k8s-community/k8s-workshop-ru> 03-part-III-setup



# Проверка работоспособности кластера

```
$ kubectl get nodes
```

# Проверка работоспособности кластера

```
$ kubectl get nodes
```

NAME	STATUS	AGE	VERSION
k8s-master-01	Ready	1d	v1.7.6
k8s-master-02	Ready	1d	v1.7.6
k8s-node-01	Ready	1d	v1.7.6
k8s-node-02	Ready	1d	v1.7.6

# Проверка работоспособности кластера

```
$ kubectl get nodes
```

NAME	STATUS	AGE	VERSION
k8s-master-01	Ready	1d	v1.7.6
k8s-master-02	Ready	1d	v1.7.6
k8s-node-01	Ready	1d	v1.7.6
k8s-node-02	Ready	1d	v1.7.6

```
$ kubectl get cs
```

# Проверка работоспособности кластера

```
$ kubectl get nodes
```

NAME	STATUS	AGE	VERSION
k8s-master-01	Ready	1d	v1.7.6
k8s-master-02	Ready	1d	v1.7.6
k8s-node-01	Ready	1d	v1.7.6
k8s-node-02	Ready	1d	v1.7.6

```
$ kubectl get cs
```

NAME	STATUS	MESSAGE	ERROR
scheduler	Healthy	ok	
controller-manager	Healthy	ok	
etcd-1	Healthy	{"health": "true"}	
etcd-0	Healthy	{"health": "true"}	

## Основной компонент в Kubernetes - Pod

```
$ kubectl get -n k8s-community pods -o wide
```



# ОСНОВНОЙ КОМПОНЕНТ в Kubernetes - Pod

```
$ kubectl get -n k8s-community pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
cockroachdb-0	1/1	Running	1	6h	10.20.2.22	k8s-node-01
cockroachdb-1.	1/1	Running	1	6h	10.20.3.13	k8s-node-02
github-integr-4200030359-df13z	1/1	Running	1	6h	10.20.3.14	k8s-node-02
github-integr-4200030359-ztqcp	1/1	Running	1	6h	10.20.2.20	k8s-node-01
ui-1395922863-jsg0p	1/1	Running	1	6h	10.20.3.16	k8s-node-02
ui-1395922863-tnr62	1/1	Running	1	6h	10.20.2.21	k8s-node-01
user-manager-4284545685-5r7p0	1/1	Running	1	6h	10.20.3.20	k8s-node-02
user-manager-4284545685-5t4mh.	1/1	Running	1	6h	10.20.2.19	k8s-node-01

Pods создаются с помощью специальных контроллеров, таких, как Deployment и DaemonSet, или просто статическим манифестом.

```
$ kubectl get -n k8s-community deploy
```

Pods создаются с помощью специальных контроллеров, таких, как Deployment и DaemonSet, или просто статическим манифестом.

```
$ kubectl get -n k8s-community deploy
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
github-integration	2	2	2	2	6h
ui	2	2	2	2	6h
user-manager	2	2	2	2	6h

Важно! Контроллер Service является основным звеном для получения доступа к Pods.

```
$ kubectl get -n k8s-community svc -o wide
```

Важно! Контроллер Service является основным звеном для получения доступа к Pods.

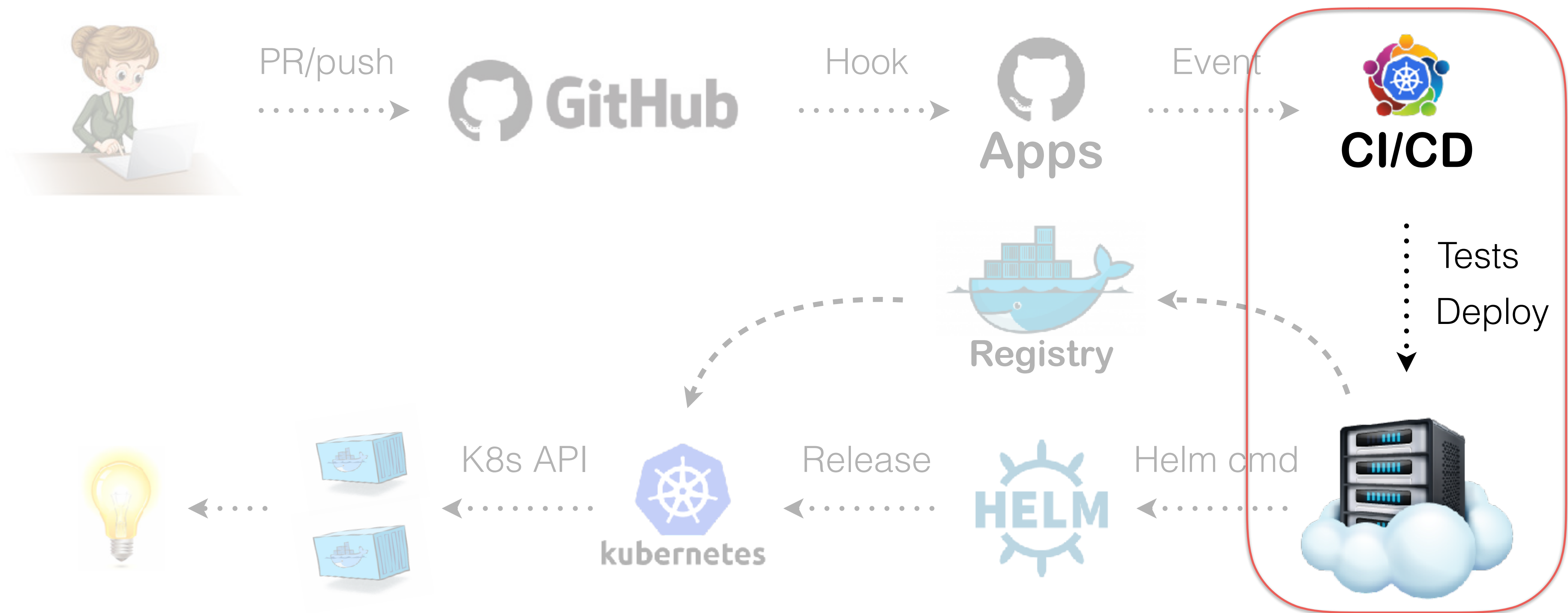
```
$ kubectl get -n k8s-community svc -o wide
```

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
cockroachdb	None	<none>	26257/TCP, 8080/TCP	7h	app=cockroachdb
cockroachdb-public	10.254.83.101	<none>	26257/TCP, 8080/TCP	7h	app=cockroachdb
github-integration	10.254.67.17	<none>	80/TCP	7h	app=github-k8s
ui	10.254.47.108	<none>	80/TCP	7h	app=ui-k8s
user-manager	10.254.251.120	<none>	80/TCP	7h	app=user-manager

# Тема 4: Подготовка сервиса для релизов и CI/CD



Для того чтобы сервис автоматически поставлялся в Kubernetes, мы используем CI/CD на Build Machine, код сервиса CI/CD доступен на GitHub



Чтобы убедиться, что все правильно, запускаем тесты

```
$ make test
```

Результатом работы тестов должен быть отчёт с резолюцией **PASS**

```
$ make test
```

```
dep ensure
```

```
. . .
```

```
ok  github.com/k8s-community/k8sapp/pkg/config1.022scoverage: 100.0% of statements
```

```
=== RUN    TestRoot
```

```
--- PASS: TestRoot (0.00s)
```

```
=== RUN    TestCollectCodes
```

```
PASS
```

```
coverage: 100.0% of statements
```

```
PASS
```

```
coverage: 100.0% of statements
```

```
ok  github.com/k8s-community/k8sapp/pkg/system1.024scoverage: 100.0% of statements
```

# Создаём специальный бранч, на который будет реагировать система CI/CD

```
$ git checkout -b release/0.0.1
```

Важно! Не забудьте указать правильный номер версии в названии бранча

```
$ git checkout -b release/0.0.1
```

```
Switched to a new branch 'release/0.0.1'
```

Данный скрипт находит текущую версию и предлагает новую

```
$ git checkout -b release/0.0.1
```

```
Switched to a new branch 'release/0.0.1'
```

```
$ ./bumper.sh
```



Данный скрипт находит текущую версию и предлагает новую

```
$ git checkout -b release/0.0.1
```

```
Switched to a new branch 'release/0.0.1'
```

```
$ ./bumper.sh
```

```
Current version 0.0.0.
```

```
Please enter bumped version [0.0.1]:
```

Данный скрипт находит текущую версию и предлагает новую

```
$ git checkout -b release/0.0.1
```

```
Switched to a new branch 'release/0.0.1'
```

```
$ ./bumper.sh
```

```
Current version 0.0.0.
```

```
Please enter bumped version [0.0.1]:
```

```
Bumped version: 0.0.1
```

## Сохраняем изменения

```
$ git checkout -b release/0.0.1
```

```
Switched to a new branch 'release/0.0.1'
```

```
$ ./bumper.sh
```

```
Current version 0.0.0.
```

```
Please enter bumped version [0.0.1]:
```

```
Bumped version: 0.0.1
```

```
$ git commit -a -m "Bumped version number to 0.0.1"
```

## Сохраняем изменения

```
$ git checkout -b release/0.0.1
```

```
Switched to a new branch 'release/0.0.1'
```

```
$ ./bumper.sh
```

```
Current version 0.0.0.
```

```
Please enter bumped version [0.0.1]:
```

```
Bumped version: 0.0.1
```

```
$ git commit -a -m "Bumped version number to 0.0.1"
```

```
[release/0.0.1 e44725f] Bumped version number to 0.0.1  
1 file changed, 1 insertion(+), 1 deletion(-)
```

## Помечаем бранч новым тэгом (номером версии)

```
$ git checkout -b release/0.0.1
```

```
Switched to a new branch 'release/0.0.1'
```

```
$ ./bumper.sh
```

```
Current version 0.0.0.
```

```
Please enter bumped version [0.0.1]:
```

```
Bumped version: 0.0.1
```

```
$ git commit -a -m "Bumped version number to 0.0.1"
```

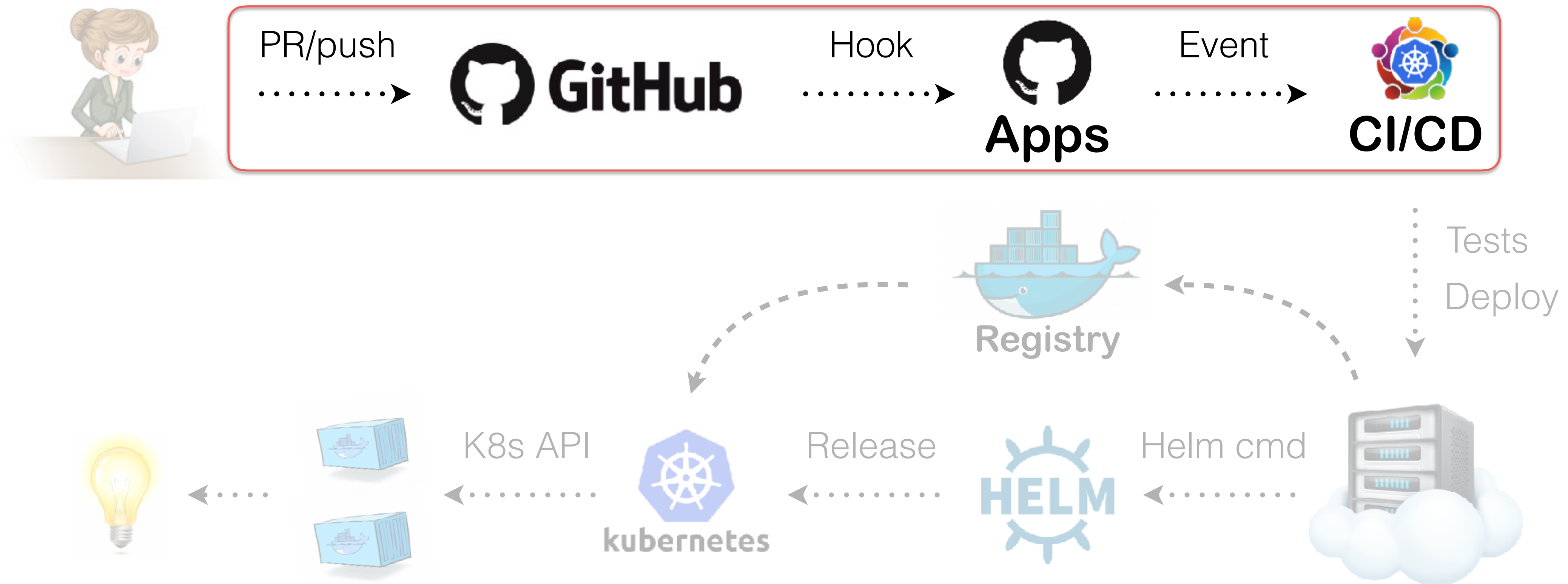
```
[release/0.0.1 e44725f] Bumped version number to 0.0.1  
1 file changed, 1 insertion(+), 1 deletion(-)
```

```
$ git tag -a 0.0.1 -m "Version 0.0.1"
```

# Тема 5: Активация CI/CD



# Отправляем бранч с указанной версией в удаленный репозиторий GitHub



# Отправляем бранч с указанной версией в удаленный репозиторий GitHub

```
$ git push -u --tags origin release/0.0.1
```

# Отправляем бранч с указанной версией в удаленный репозиторий GitHub

```
$ git push -u --tags origin release/0.0.1
```

```
Username for 'https://github.com': user-name
```

```
Password for 'https://user-name@github.com':
```

```
Counting objects: 3, done.
```

```
Delta compression using up to 4 threads.
```

```
Compressing objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 331 bytes | 0 bytes/s, done.
```

```
Total 3 (delta 2), reused 0 (delta 0)
```

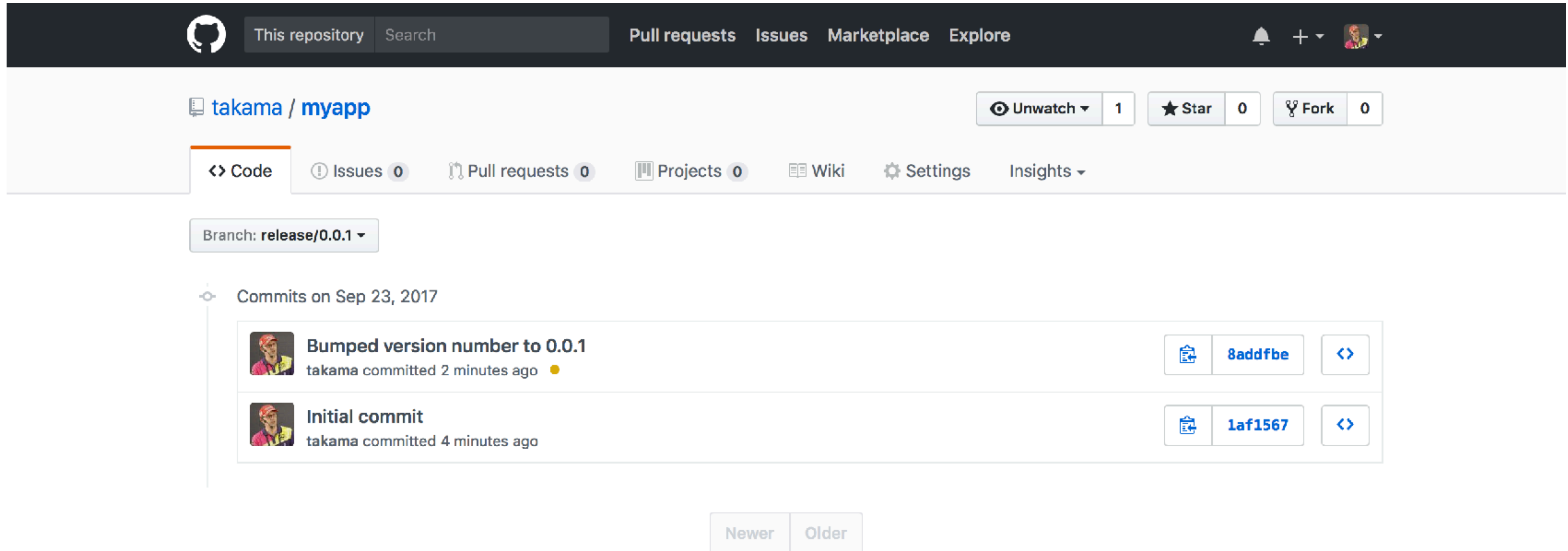
```
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
```

```
To https://github.com/user-name/myapp.git
```

```
* [new branch]      release/0.0.1 -> release/0.0.1
```

```
Branch release/0.0.1 set up to track remote branch release/0.0.1 from origin.
```

Информация об успехе или ошибках сборки будет отображена на GitHub в разделе **Commits** бранча **release/0.1.1**



The screenshot shows the GitHub interface for the repository 'takama / myapp'. The top navigation bar includes links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The repository name 'takama / myapp' is displayed, along with statistics: 1 Unwatch, 0 Stars, and 0 Forks. The 'Code' tab is selected, showing the 'Branch: release/0.0.1'. Below this, the 'Commits on Sep 23, 2017' section lists two commits:

- Bumped version number to 0.0.1**  
takama committed 2 minutes ago  
Commit hash: 8addfbe
- Initial commit**  
takama committed 4 minutes ago  
Commit hash: 1af1567

Navigation buttons for 'Newer' and 'Older' commits are visible at the bottom of the commit list.

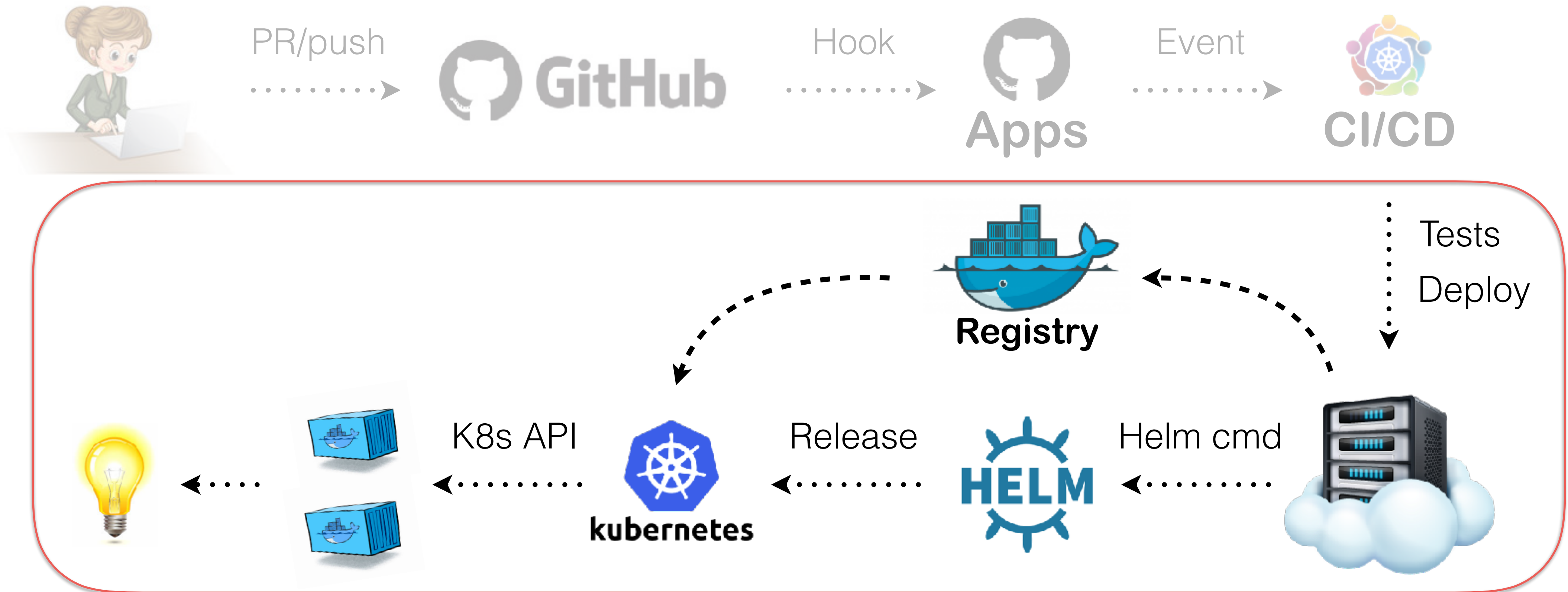
Информация об успехе или ошибках сборки будет отображена на GitHub в разделе **Commits** бранча **release/0.1.1**

The screenshot shows the GitHub interface for the repository 'takama / myapp'. The top navigation bar includes links for 'This repository', 'Search', 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. On the right, there are icons for notifications, a plus sign, and a user profile. Below the repository name, there are buttons for 'Unwatch' (1), 'Star' (0), and 'Fork' (0). A secondary navigation bar shows tabs for 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Settings', and 'Insights'. A dropdown menu indicates the current branch is 'release/0.0.1'. The 'Commits on Sep 23, 2017' section lists two commits: 'Bumped version number to 0.0.1' by takama (committed 7 minutes ago) with hash 8addfbe, and 'Initial commit' by takama (committed 9 minutes ago) with hash 1af1567. Each commit entry includes a copy icon, the hash, and a code icon. At the bottom, there are 'Newer' and 'Older' buttons.

# Тема 6: Проверка работоспособности сервиса



Сервис будет доступен по адресу [https://services.k8s.community/user\\_name/myapp](https://services.k8s.community/user_name/myapp)





Сервис будет доступен по адресу [https://services.k8s.community/user\\_name/myapp](https://services.k8s.community/user_name/myapp)

```
$ curl https://services.k8s.community/user_name/myapp/info
```

Сервис будет доступен по адресу [https://services.k8s.community/user\\_name/myapp](https://services.k8s.community/user_name/myapp)

```
$ curl https://services.k8s.community/user_name/myapp/info
```

```
{"host": "myapp-4017796230-cxrgg", "version": "0.0.1", "commit": "git-8addfbe", "repo": "https://github.com/takama/myapp", "compiler": "go1.9", "runtime": {"cpu": 2, "memory": "8.57MB", "goroutines": 6}, "state": {"maintenance": false, "uptime": "3m46.071070119s"}, "requests": {"duration": {"average": "13.199802ms", "max": "137.582925ms"}, "codes": {"2xx": 51, "4xx": 0, "5xx": 0}}}
```

Сервис будет доступен по адресу [https://services.k8s.community/user\\_name/myapp](https://services.k8s.community/user_name/myapp)

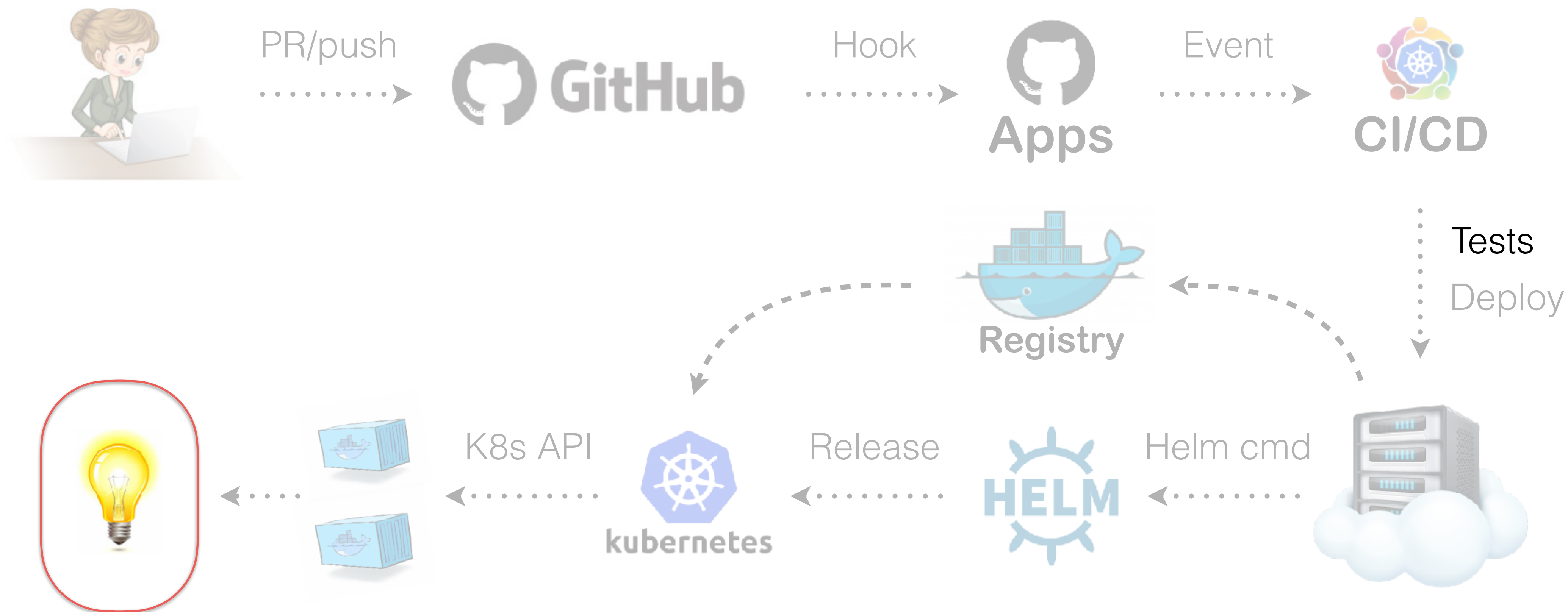
```
$ curl https://services.k8s.community/user_name/myapp/info | jq
```

Сервис будет доступен по адресу [https://services.k8s.community/user\\_name/myapp](https://services.k8s.community/user_name/myapp)

```
$ curl https://services.k8s.community/user_name/myapp/info | jq
```

```
{
  "host": "myapp-4017796230-hbgg3",
  "version": "0.0.1",
  . . .
  "requests": {
    "duration": {
      "average": "2.885871ms",
      "max": "101.399704ms"
    },
    "codes": {
      "2xx": 45,
      "4xx": 0,
      "5xx": 0
    }
  }
}
```

Сервис будет доступен по адресу [https://services.k8s.community/user\\_name/myapp](https://services.k8s.community/user_name/myapp)



## Проверяем наличие сервиса в Pods в двух экземплярах

```
$ kubectl get -n user_name pods
```

# Проверяем наличие сервиса в Pods в трех экземплярах

```
$ kubectl get -n user_name pods
```

NAME	READY	STATUS	RESTARTS	AGE
myapp-2404051545-2j60t	1/1	Running	0	2d
myapp-2404051545-tv2f3	1/1	Running	0	2d



## Проверяем наличие Deployment

```
$ kubectl get -n user_name pods
```

NAME	READY	STATUS	RESTARTS	AGE
myapp-2404051545-2j60t	1/1	Running	0	2d
myapp-2404051545-tv2f3	1/1	Running	0	2d

```
$ kubectl get -n user_name deploy
```

# Проверяем наличие Deployment

```
$ kubectl get -n user_name pods
```

NAME	READY	STATUS	RESTARTS	AGE
myapp-2404051545-2j60t	1/1	Running	0	2d
myapp-2404051545-tv2f3	1/1	Running	0	2d

```
$ kubectl get -n user_name deploy
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
myapp	3	3	3	3	2d

## Проверяем наличие Service

```
$ kubectl get -n user_name pods
```

NAME	READY	STATUS	RESTARTS	AGE
myapp-2404051545-2j60t	1/1	Running	0	2d
myapp-2404051545-tv2f3	1/1	Running	0	2d

```
$ kubectl get -n user_name deploy
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
myapp	3	3	3	3	2d

```
$ kubectl get -n user_name svc -o wide
```

## Проверяем наличие Service

```
$ kubectl get -n user_name pods
```

NAME	READY	STATUS	RESTARTS	AGE
myapp-2404051545-2j60t	1/1	Running	0	2d
myapp-2404051545-tv2f3	1/1	Running	0	2d

```
$ kubectl get -n user_name deploy
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
myapp	3	3	3	3	2d

```
$ kubectl get -n user_name svc -o wide
```

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
myapp	10.254.68.100	<none>	80/TCP	2d	app=myapp-takama

## Каким образом сервис представлен во внешний мир?

```
$ kubectl get -n user_name ing -o yaml
. . .
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  name: myapp
spec:
  rules:
  - host: services.k8s.community
    http:
      paths:
      - path: /user_name/myapp
        backend:
          serviceName: myapp
          servicePort: 80
. . .
```

## Каким образом сервис представлен во внешний мир?

```
$ kubectl get -n user_name ing -o yaml
. . .
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  name: myapp
spec:
  rules:
  - host: services.k8s.community
    http:
      paths:
      - path: /user_name/myapp
        backend:
          serviceName: myapp
          servicePort: 80
. . .
```

## Каким образом сервис представлен во внешний мир?

```
$ kubectl get -n user_name ing -o yaml
. . .
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  name: myapp
spec:
  rules:
  - host: services.k8s.community
    http:
      paths:
      - path: /user_name/myapp
        backend:
          serviceName: myapp
          servicePort: 80
. . .
```



## Каким образом сервис представлен во внешний мир?

```
$ kubectl get -n user_name ing -o yaml
. . .
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  name: myapp
spec:
  rules:
  - host: services.k8s.community
    http:
      paths:
      - path: /user_name/myapp
        backend:
          serviceName: myapp
          servicePort: 80
. . .
```

## Каким образом сервис представлен во внешний мир?

```
$ kubectl get -n user_name ing -o yaml
. . .
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  name: myapp
spec:
  rules:
  - host: services.k8s.community
    http:
      paths:
      - path: /user_name/myapp
        backend:
          serviceName: myapp
          servicePort: 80
. . .
```

# Что мы узнали?

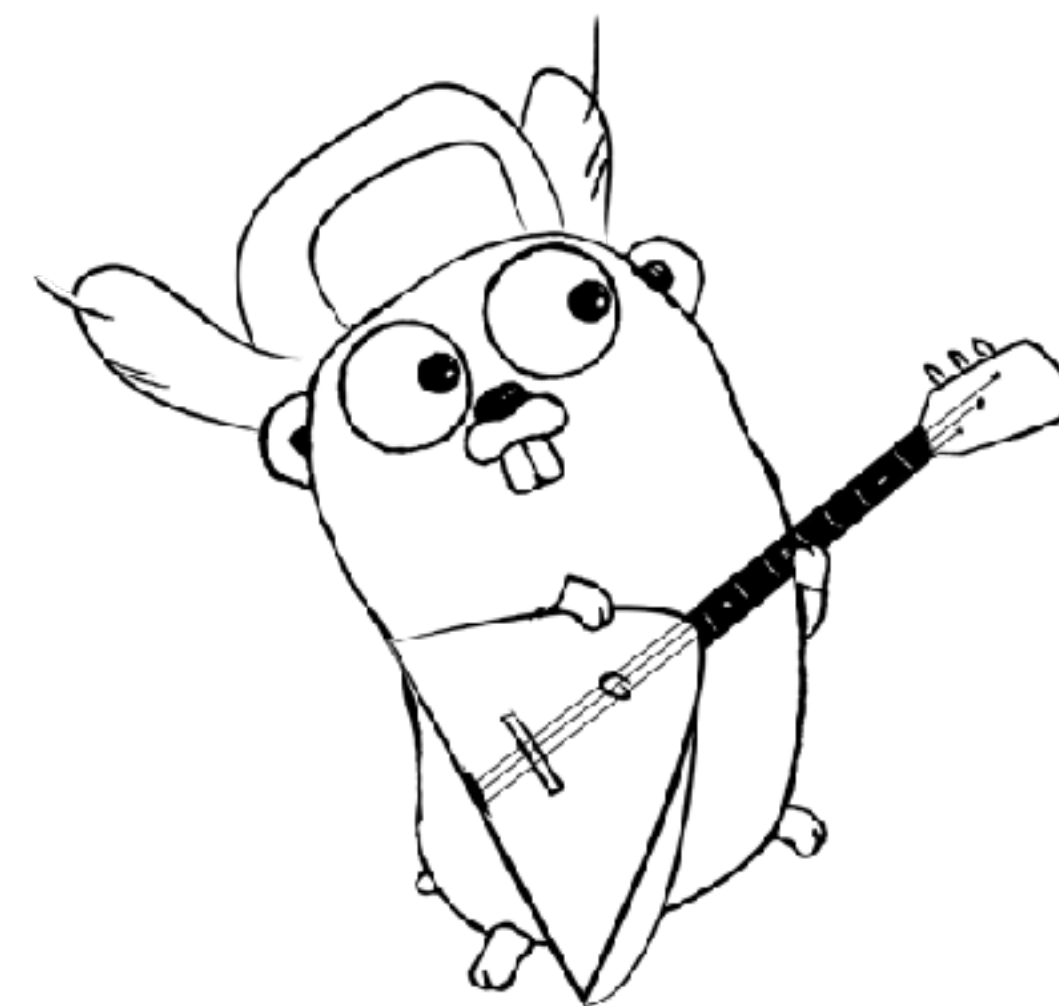
1. Что есть замечательный генератор сервисов
2. Как подготовить свой сервис для выполнения в Kubernetes
3. Познакомились с процессом CI/CD
4. Узнали несколько команд для работы с Kubernetes
5. Доставили протестированный код нашего сервиса в Kubernetes
6. Убедились, что всё это совсем не сложно



<https://github.com/k8s.community>

[k8s.community@gmail.com](mailto:k8s.community@gmail.com)

<https://slack.k8s.community>



<http://slack.golang-ru.com>

<https://golangshow.com>

Елена Граховац



<https://github.com/rumyantseva>

Игорь Должиков



<https://github.com/takama>

Влад Савельев



<https://github.com/vsaveliev>

