



ML-OPS В ЛЕМАНА ПРО:

История разработки и внедрения
централизованной MLOps платформы





**Ксения
Блажевич**



**Елизавета
Гаврилова**

01

О компании:

структура команд и особенности
разработки ML-продуктов

Лемана ПРО как бизнес:

45000+

Сотрудников

11

Дарксторов

6

РЦ

2000+

Поставщиков

112

Магазинов

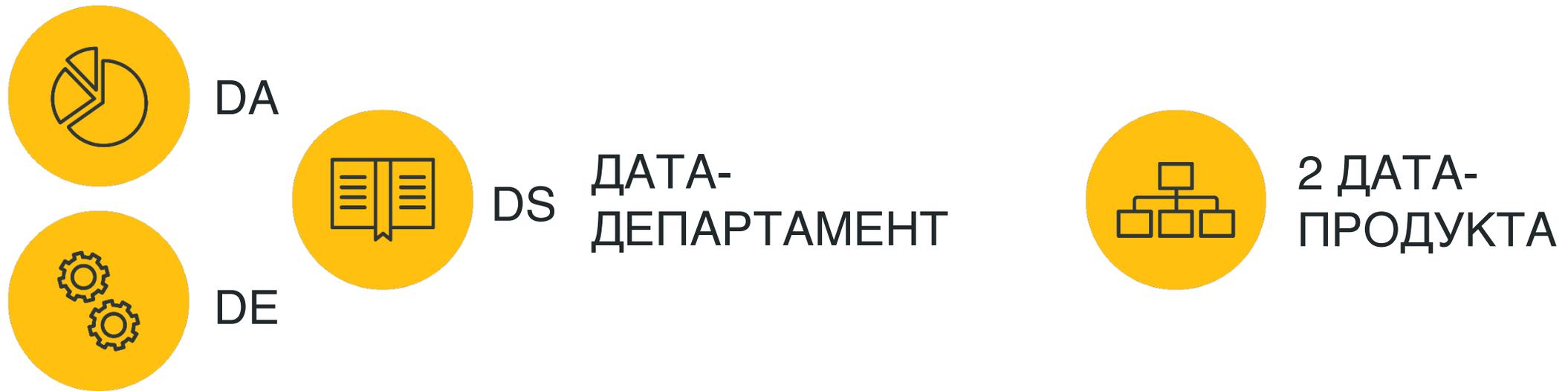
21

Бизнес-
департамент



ОЧЕНЬ
МНОГО
ДАННЫХ

Структура команд в 2019 году



*Дата-продукт - решение, в основе которого лежит одна или несколько ML моделей

Дата-продукты в Лемана ПРО

НА 2019:

3 Модели

2 Продукта
(доведены до продакшна
в каком-то виде)

X1

7



К 2025:

~60 Модели

~35 Продуктов

*Дата-продукт - решение, в основе которого лежит одна или несколько ML моделей

Типы продуктов

Классический ML/
табличные данные

Рекомендательные системы

Скоринговые модели

Классификация

Прогнозы/
регрессионные модели

Поиск/ Ранжирование

Глубокое
обучение:

CV

NLP

Регрессии

GEN AI:

Генерация изображений

Генерация текстов

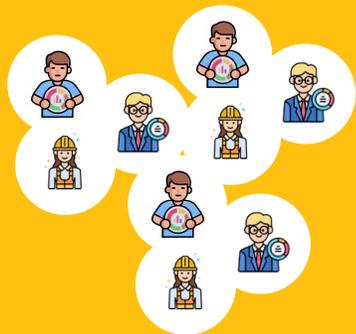
Классификация

RAG для задач поиска
по документам

Batch inference

Real-time inference

Структура команд в 2024 году



Дата-департамент

20



Дата-команды

~35  DE

~45  DA

~30  DS

02

Мотивация для внедрения MLOps- платформы

Зоопарк в технологиях и стандартах:



Зоопарк в технологиях и стандартах:

1

Сложный контроль качества ML-продуктов и компонент

3

Нет общей инфраструктуры

2

Отсутствует единообразие продуктов

4

Нет контролируемой точки поддержки

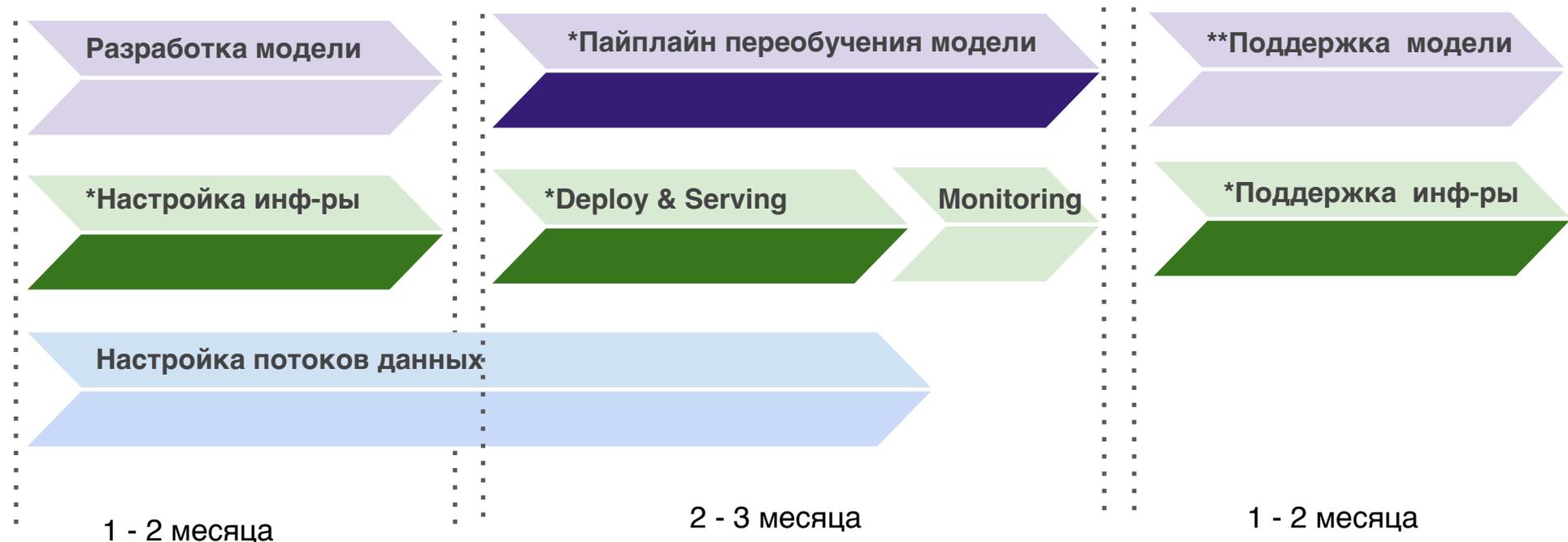
Перенимай привычки
У УСПЕШНЫХ



Подсчитали
КОСТЫ и ТТМ
разработки
продуктов

Ключевые этапы внедрения ML в продакшн в LMRU

DS/ ML-engineer
DE
DEVOps



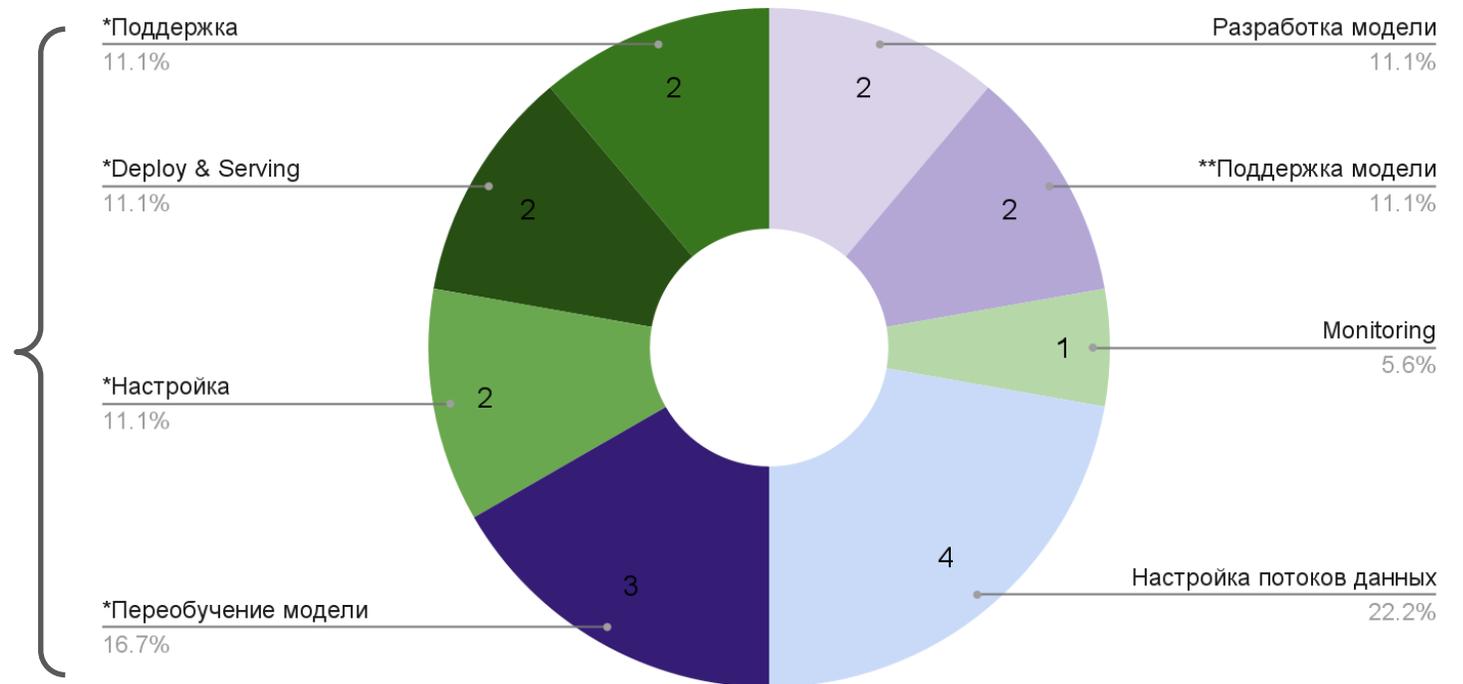
*Этапы которые **дублируются/ могут переиспользоваться** в различных ML-продуктах

**Мониторинг качества работы, ручное переобучение, обновление модели

При разработке каждого дата-продукта

DS/ ML-engineer
DE
DEVops

~9 месяцев работы DS/ DEVops



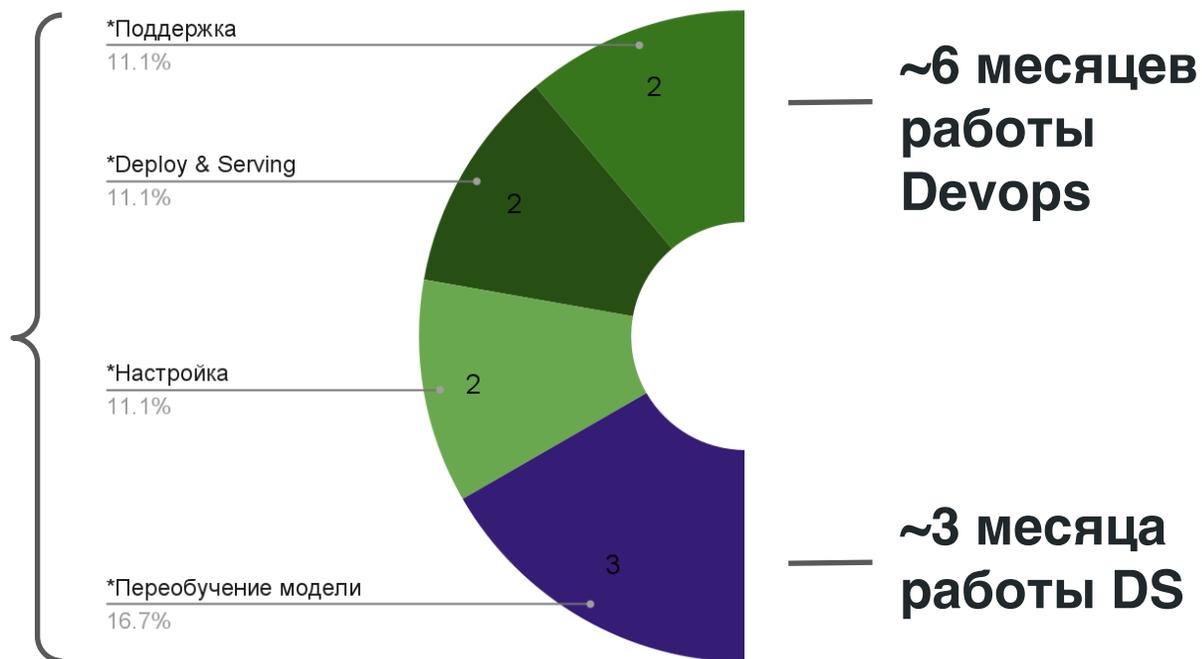
*Этапы которые могут переиспользоваться в различных ML-продуктах

**Мониторинг качества работы, ручное переобучение, обновление модели

180+ месяцев работы DS/Devops, которые можно инвестировать в улучшение качества или новые продукты

DS/ ML-engineer
DE
DEVops

~9 месяцев работы
DS/ DEVops



2021: **15**
дата-продуктов

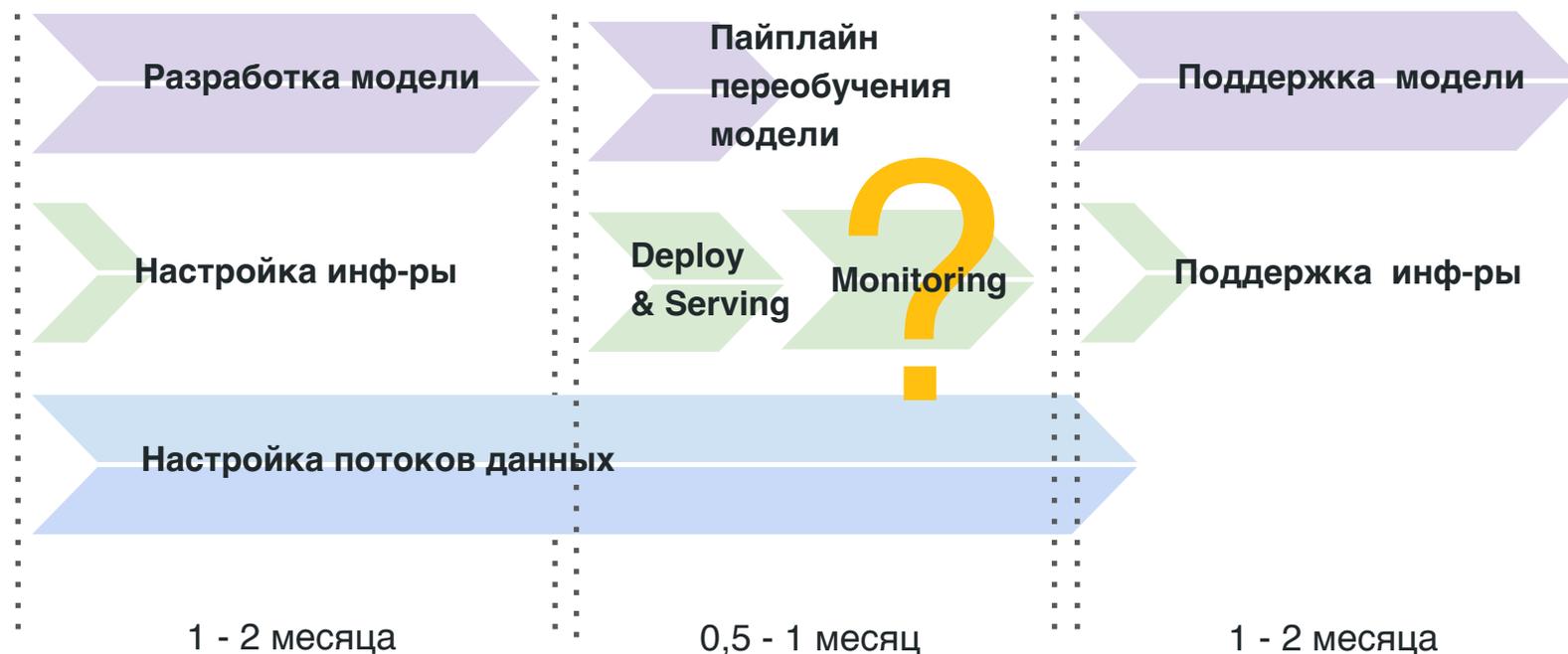
2025: **35+**
дата-продуктов

*Этапы которые могут переиспользоваться
в различных ML-продуктах

**Мониторинг качества работы, ручное
переобучение, обновление модели

С внедрением MLOps время на разработку и запуск продукта сокращается

DS/ ML-engineer
DE
DEVOps



**Мониторинг качества работы, ручное переобучение, обновление модели

Запросы от наших специалистов

- 1 Общее пространство для research
- 2 Train и serving в одном месте
- 3 Понятные инструкции и шаблоны
- 4 Актуальный стек
- 5 Общие компоненты
- 6 Потребность в легком доступе к GPU
- 7 Минимизация потребности в devops
- 8 Ускорение разработки
- 9 Масштабируемость

Мотивация для внедрения MLOps платформы

1 Стандартизация
продуктивизации

2 Сокращение
затрат и TTM

3 Запросы
от DS и DA

03

История внедрения MLOps-платформы

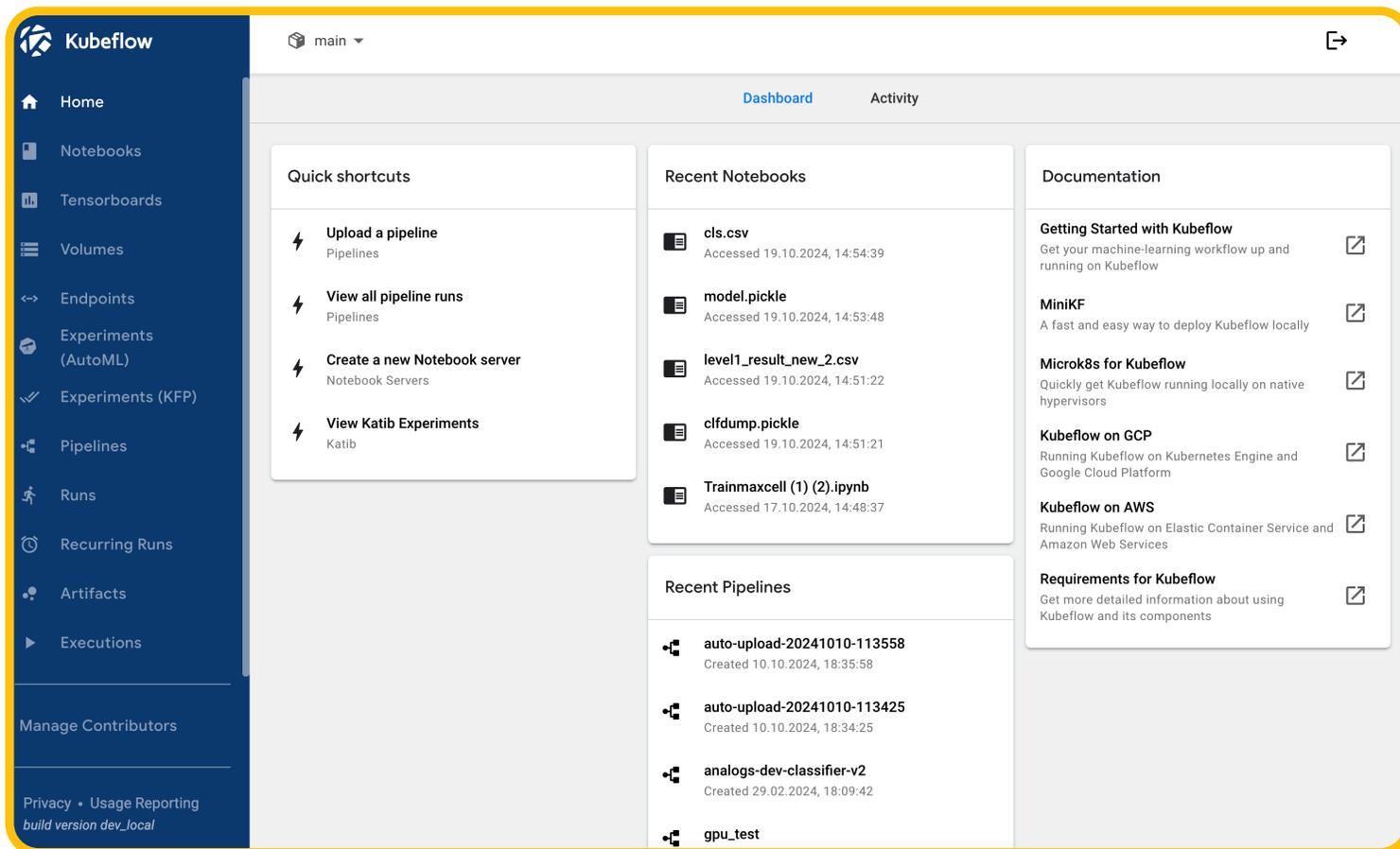
Выбор основы платформы



общее пространство для ресерча	✓	✓	✓	✓
train и serving в одном месте	✓	✓	✓	✓
минимизация потребности в devops	✓	✓	✓	✓
потребность в понятных инструкциях и шаблонах	✓	✓	✓	✓
потребность в GPU	✓	✓	✓	✓
open source	✓	✓	✓	✓

Что такое KubeFlow?

Набор сервисов с web-интерфейсом



Библиотека
для пайплайнов

```
pip install kfp
```

Here is a simple pipeline that prints a greeting:

```
from kfp import dsl

@dsl.component
def say_hello(name: str) -> str:
    hello_text = f'Hello, {name}!'
    print(hello_text)
    return hello_text

@dsl.pipeline
def hello_pipeline(recipient: str) -> str:
    hello_task = say_hello(name=recipient)
    return hello_task.output
```

Что такое KubeFlow?



Пространство под research

Админ создает namespace команды через k8s манифест

Описание неймспейса и лимитов:

- Profile
- LimitRange

Выдача доступов к неймспейсу:

- RoleBinding
- AuthorizationPolicy



```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  annotations:
    role: edit
    user: 'UserName'
  name: user-UserName-clusterrole-edit
  namespace: test
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kubeflow-edit
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: 'UserName'
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  annotations:
    role: edit
    user: 'UserName'
  name: user-UserName-clusterrole-edit
  namespace: test
spec:
  action: ALLOW
  rules:
  - request.headers[kubeflow-userid]
  users:
  - 'UserName'
```

Пространство под **research**

Пользователь поднимает себе сервер с Jupyter или VSCode

The screenshot shows the 'New notebook' configuration interface in Kubeflow. It features three main options for server types: JupyterLab, VisualStudio Code, and RStudio. Below these options are fields for configuring resources: Minimum CPU (0,5), Minimum Memory Gi (1), Maximum CPU (0,6), and Maximum Memory Gi (1,2). There is also a section for GPUs with fields for 'Number of GPUs' (set to None) and 'GPU Vendor' (set to NVIDIA). A 'Custom Notebook' dropdown menu is also present.

Тип сервера

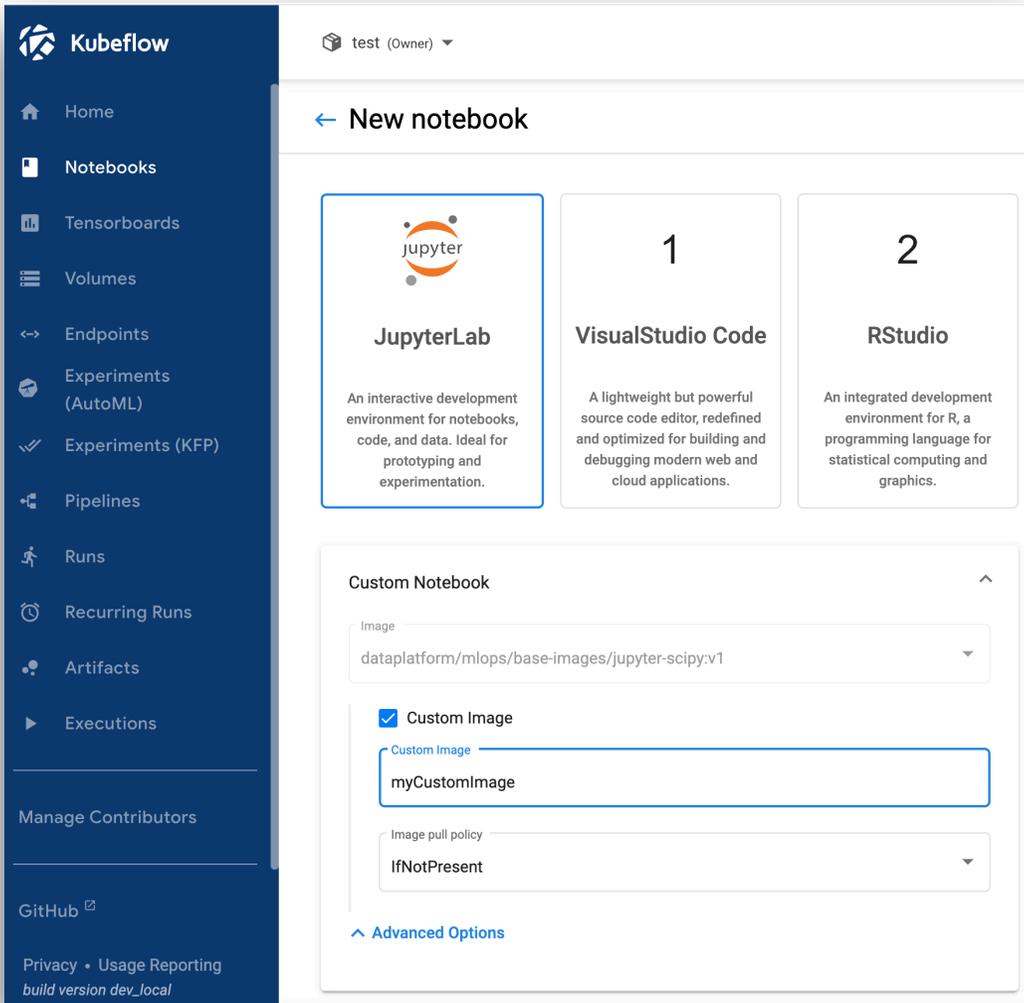
Базовый образ*

Ресурсы

The screenshot shows a 'Custom Notebook' dropdown menu with a list of Docker image names. The first image is highlighted in blue: `dataplatfrom/mlops/base-images/jupyter-scipy:v1`. Other images in the list include `dataplatfrom/mlops/base-images/pytorch-full:v1`, `dataplatfrom/mlops/base-images/pytorch-cuda-full:v1`, `kubeflownotebookswg/jupyter-tensorflow-full:v1.8.0`, and `kubeflownotebookswg/jupyter-tensorflow-cuda-full:v1.8.0`.

Пространство под **research**

Когда нужен кастомный образ



Kubeflow

test (Owner)

← New notebook

JupyterLab
An interactive development environment for notebooks, code, and data. Ideal for prototyping and experimentation.

1 VisualStudio Code
A lightweight but powerful source code editor, redefined and optimized for building and debugging modern web and cloud applications.

2 RStudio
An integrated development environment for R, a programming language for statistical computing and graphics.

Custom Notebook

Image
dataplatfrom/mlops/base-images/jupyter-scipy:v1

Custom Image

Custom Image
myCustomImage

Image pull policy
IfNotPresent

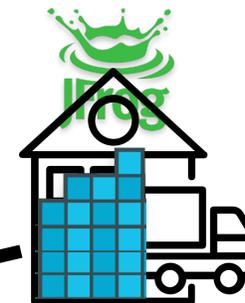
^ Advanced Options

Manage Contributors

GitHub

Privacy • Usage Reporting
build version dev_local

```
FROM kubeflownotebookswg/jupyter  
  
COPY requirements.txt requirements.txt  
  
RUN pip install --upgrade pip \  
-r requirements.txt
```



Пространство под **research**

Когда нужна пара новых библиотек: `pip install` - наше всё!

The image shows a JupyterLab interface. On the left is a file browser with a search bar and a table of files. The main area is a launcher with icons for Notebook, Console, and Other. A terminal window is open on the right, showing the output of a `pip install` command. An arrow points from the Terminal icon in the launcher to the terminal window.

Name	Last Modified
lost+found	8 months ago
ca.crt	8 months ago
Untitled.ipynb	3 months ago

```
(base) jovyan@test-serving-0:~$ pip install nltk
Collecting nltk
  Downloading nltk-3.9.1-py3-none-any.whl (1.5 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 4.7 MB/s eta 0:00:00
Requirement already satisfied: joblib in /opt/conda/lib/python3.8/site-packages (from nltk)
Collecting regex<=2021.8.3
  Downloading regex-2024.9.11-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (785.0 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 785.0/785.0 kB 18.7 MB/s eta 0:00:00
Requirement already satisfied: click in /opt/conda/lib/python3.8/site-packages (from regex)
Collecting tqdm
  Downloading tqdm-4.66.6-py3-none-any.whl (78 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.3/78.3 kB 590.1 kB/s eta 0:00:00
Installing collected packages: tqdm, regex, nltk
Successfully installed nltk-3.9.1 regex-2024.9.11 tqdm-4.66.6
(base) jovyan@test-serving-0:~$
```

Research

Исследование в Jupyter Notebook на VM или локальном компьютере



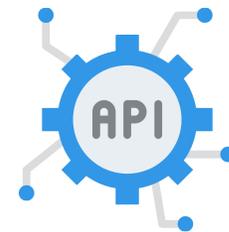
Train

Обучение модели на новых данных: вручную или автоматически по расписанию



Inference

Batch Serving или непрерывный Serving



```
model.predict(x)
```

Train вручную: как было

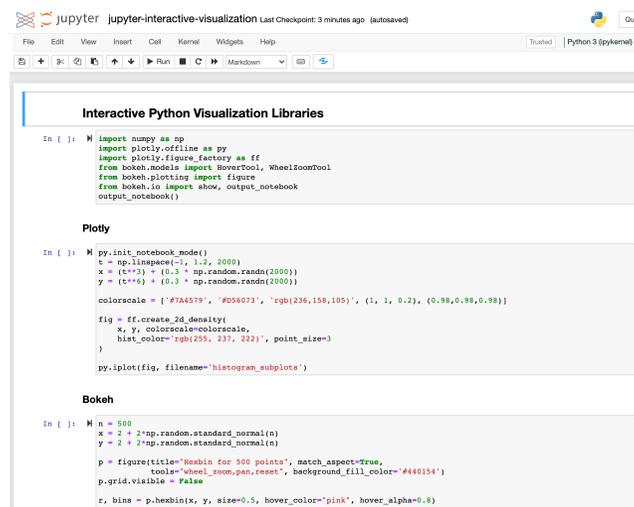
01

Залезть
на VM (ssh)

```
% ssh lizvladi@12.123.123.12
```

02

Подготовить Jupyter
(один раз)



```
Interactive Python Visualization Libraries

In [ ]: | import numpy as np
import plotly-offline as py
import plotly.figure_factory as ff
from bokeh.models import HoverTool, WheelZoomTool
from bokeh.plotting import figure
from bokeh.io import show, output_notebook
output_notebook()

Plotly

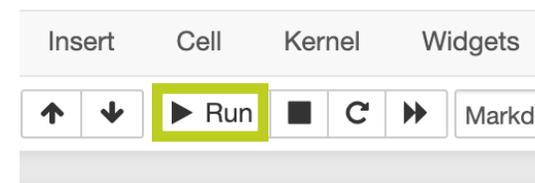
In [ ]: | py.init_notebook_mode()
t = np.linspace(-1, 1, 2, 2000)
x = (t**3) + (0.3 * np.random.randn(2000))
y = (t**6) + (0.3 * np.random.randn(2000))
colorscale = ['#7A5797', '#066073', 'rgb(236,158,105)', (1, 1, 0.2), (0.98,0.98,0.98)]
fig = ff.create_2d_density(
    x, y, colorscale=colorscale,
    hist_color='rgb(255, 237, 222)', point_size=3
)
py.iplot(fig, filename='histogram_subplots')

Bokeh

In [ ]: | n = 500
x = 2 + 2*np.random.standard_normal(n)
y = 2 + 2*np.random.standard_normal(n)
p = figure(title='Hexbin for 500 points', match_aspect=True,
           tools='wheel_zoom,pan,reset', background_fill_color='#440154')
p.grid.visible = False
r, bins = p.hexbin(x, y, size=0.5, hover_color='pink', hover_alpha=0.8)
```

03

Запустить
Jupyter при
необходимости



Train: pipeline в Jupyter

0 Поднять Jupyter 



1 Подготовить пайплайн



2 Загрузить скомпилированный pipeline.yaml



3 Запустить пайплайн при необходимости

```
from kfp import compiler, dsl
from kfp.dsl import Dataset, Input, Output

import pandas as pd

from sklearn import datasets
from sklearn.linear_model import LogisticRegression

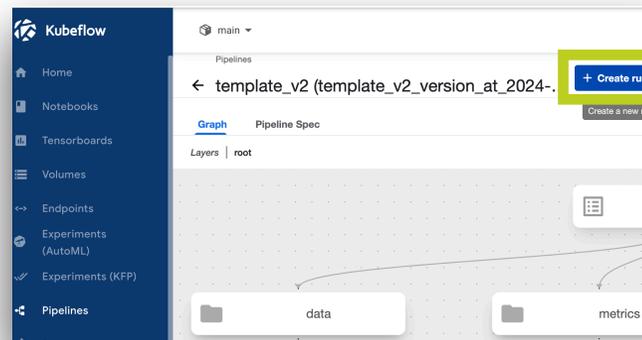
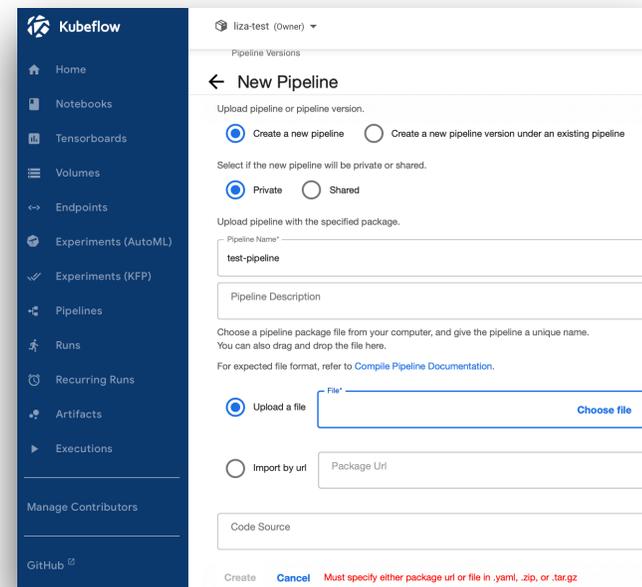
@dsl.component
def read_data(data: Output[Dataset]):
    data = wine_data[0]
    data["target"] = wine_data[1]
    return wine_data

@dsl.component
def train(data: Input[Dataset]):
    X = data[0]
    y = data[1]

    model = LogisticRegression()
    model.fit(X,y)

@dsl.pipeline
def hello_pipeline():
    read_data_task = read_data()
    train_task = train(data=read_data_task.outputs["data"])

compiler.Compiler().compile(hello_pipeline, "pipeline.yaml")
```



Train: pipeline в Jupyter

```
from kfp import compiler, dsl
from kfp.dsl import Dataset, Input, Output

import pandas as pd

from sklearn import datasets
from sklearn.linear_model import LogisticRegression

@dsl.component
def read_data(data: Output[Dataset]):
    data = wine_data[0]
    data["target"] = wine_data[1]
    return wine_data

@dsl.component .....
def train(data: Input[Dataset]):
    X = data[0]
    y = data[1]

    model = LogisticRegression()
    model.fit(X,y)

@dsl.pipeline .....
def hello_pipeline():
    read_data_task = read_data()
    train_task = train(data=read_data_task.outputs["data"])

compiler.Compiler().compile(hello_pipeline, "pipeline.yaml")
```

1 Подготовить пайплайн

.....> компонент = task

.....> пайплайн = DAG

.....> компиляция yaml с пайплайном

Train: pipeline в Jupyter

Kubeflow

Home

Notebooks

Tensorboards

Volumes

Endpoints

Experiments (AutoML)

Experiments (KFP)

Pipelines

Runs

Recurring Runs

Artifacts

Executions

Manage Contributors

GitHub

liza-test (Owner)

Pipeline Versions

← New Pipeline

Upload pipeline or pipeline version.

Create a new pipeline Create a new pipeline version under an existing pipeline

Select if the new pipeline will be private or shared.

Private Shared

Upload pipeline with the specified package.

Pipeline Name*
test-pipeline

Pipeline Description

Choose a pipeline package file from your computer, and give the pipeline a unique name.
You can also drag and drop the file here.

For expected file format, refer to [Compile Pipeline Documentation](#).

Upload a file Import by url

File* [Choose file](#)

Package Url

Code Source

[Create](#) [Cancel](#) Must specify either package url or file in .yaml, .zip, or .tar.gz

2 Загрузить
скомпилированный
pipeline.yaml

Train: pipeline в Jupyter

The screenshot displays the Kubeflow Pipelines interface. On the left is a dark blue sidebar with navigation options: Home, Notebooks, Tensorboards, Volumes, Endpoints, Experiments (AutoML), Experiments (KFP), Pipelines (highlighted), Runs, Recurring Runs, Artifacts, and Executions. Below this are 'Manage Contributors' and 'GitHub' links, and at the bottom, 'Privacy • Usage Reporting' and 'build version dev_local'.

The main content area shows the 'Pipelines' section for 'main'. The current pipeline is 'template_v2 (template_v2_version_at_2024-0..)', with a '+ Create run' button highlighted in yellow and a '+ Upload version' button. Below the pipeline name are tabs for 'Graph' (selected) and 'Pipeline Spec'. The 'Layers' section shows 'root'.

The pipeline graph consists of several nodes and artifacts:

- 'download-data' (task) outputs artifacts 'data' and 'metrics'.
- 'prepare-data' (task) takes 'data' and 'metrics' as input and outputs artifacts 'metrics', 'test', and 'train'.
- 'train-model' (task) takes 'test' and 'train' as input and outputs artifacts 'metrics' and 'model'.
- 'switch-model' (task) takes 'model' as input and outputs artifact 'metrics'.

At the bottom left of the graph area are zoom controls (+, -, refresh, and lock icons) and a 'Show Summary' button.

3 Запустить пайплайн при необходимости

Train: pipeline в Docker

1) Базовый образ со всем необходимым для работы пайплайна

2) Каждый py-файл — один компонент, который будет запущен в своем контейнере

3) py-файл со сборкой всех компонент в единый пайплайн и компиляцией pipeline.yaml

4) Jenkinsfile для:
base_image
pipeline.py → pipeline.yaml



```
├── docker
│   ├── base_image
│   │   └── Dockerfile
│   └── kubeflow_pipeline/
│       ├── kfp_vars.py
│       ├── pipeline.py
│       └── pipeline.yaml
├── src/
│   ├── requirements.txt
│   ├── lib/
│   │   └── config.py
│   └── pipeline_steps/
│       ├── download_data
│       ├── prepare_data
│       ├── train_model
│       └── switch_model
└── Jenkinsfile
```

Train: pipeline в Docker

1

Создать Git-репозиторий с примером проекта



```
├── docker
│   ├── base_image
│   │   └── Dockerfile
│   └── kubeflow_pipeline/
│       ├── kfp_vars.py
│       ├── pipeline.py
│       └── pipeline.yaml
├── src/
│   ├── requirements.txt
│   ├── lib/
│   │   └── config.py
│   └── pipeline_steps/
│       ├── download_data
│       ├── prepare_data
│       ├── train_model
│       └── switch_model
└── Jenkinsfile
```

2

Написать Jenkins модули



3

Провести обучение



Train: передача данных через Artifacts

```
@container_component
def prepare_data(
    data: Input[Dataset],
    metrics: Output[Metrics],
    train: Output[Dataset],
    test: Output[Dataset]
):
    return ContainerSpec(
        image=f"docker-local-{team_name}.art.lmru.tech/{team_name}/{project_name}/base-image:{tag}",
        command=["python", "pipeline_steps/prepare_data/task.py"],
        args=[
            "--executor_input",
            dsl.PIPELINE_TASK_EXECUTOR_INPUT_PLACEHOLDER,
        ]
    )
```

Описание одного шага
(компоненты)

▶ **Нестандартный синтаксис
описания выходных данных**

✔ **Совсем неочевидный
способ передачи
аргументов из командной
строки внутрь контейнера**

```
@dsl.pipeline(name="project_template")
def pipeline():

    task_download_data = download_data()
    task_download_data = prepare_task(task_download_data)

    task_prepare_data = prepare_data(data = task_download_data.outputs["data"])
    task_prepare_data = prepare_task(task_prepare_data)

    task_train_model = train_model(train = task_prepare_data.outputs["train"], test = task_prepare_data.outputs["test"])
    task_train_model = prepare_task(task_train_model)
```

Описание пайплайна

Train: передача данных

Несмотря на сложный синтаксис, получаем **удобный UI**

```
@container_component
def prepare_data(
    data: Input[Dataset],
    metrics: Output[Metrics],
    train: Output[Dataset],
    test: Output[Dataset]
):
```

metrics

test

train

prepare-data ✓

train-model ✓

Artifact Info [Visualization](#)

Scalar Metrics: metrics

name ↑	value
date	"20241023"
split	0.2
test_size	36
train_size	142

Train: секреты



HashiCorp
Vault

0

в Vault команды
включена
аутентификация
через Approle

1

команда
создает политику
с доступом
к секретам проекта

2

команда создает
нужную роль
и креды к ней
(role_id,
secret_id)

3

админ KF заводит
k8s-секрет с кредами
в неймспейсе команды

4

в коде пайплайнов
команда с помощью
k8s секрета с кредами
подключается к
своему Vault и читает
нужные секреты

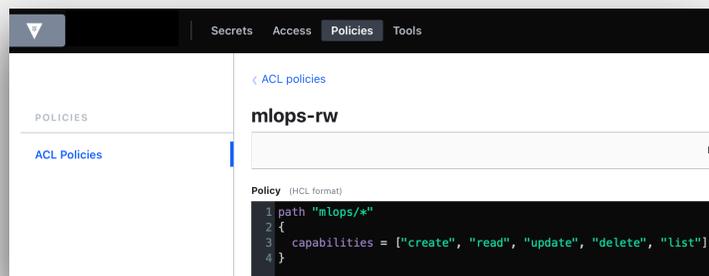
5

если надо обновить
секреты - команда
просто обновляет
их в Vault



Kubeflow

Train: секреты



```
vault_con = connectors.VaultConnector()
vault_con.set_secrets_as_envvars(
    path=os.getenv("VAULT_PATH"), mount_point=os.getenv("VAULT_MOUNT_POINT")
)
_log.info("Secrets imported from vault\A")
```

библиотека на основе hvac

K8s-секреты



Kubeflow

```
apiVersion: v1
data:
  role_id: OTI
  secret_id: ;
kind: Secret
metadata:
  name: vault-approle-mlops-template
  namespace: main
type: Opaque
```

Train: секреты

библиотека на
основе hvac

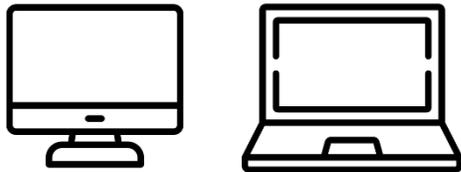
```
vault_con = connectors.VaultConnector()
vault_con.set_secrets_as_envvars(
    path=os.getenv("VAULT_PATH"), mount_point=os.getenv("VAULT_MOUNT_POINT")
)
_log.info("Secrets imported from vault\n")
```

К8s-секреты

Из чего состоит ML-процесс?

Research

Исследование в Jupyter Notebook на VM или локальном компьютере



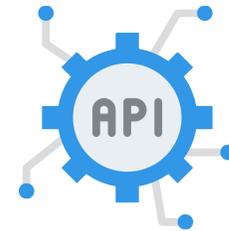
Train

Обучение модели на новых данных: вручную или автоматически по расписанию



Inference

Batch Serving или непрерывный Serving



```
model.predict(x)
```

Inference: API

Как было v1



Поднять VM



Добавить пользователей и SSH-ключи

Развернуть приложение

Docker с приложением

Как было v2



Написать Jenkins-pipeline



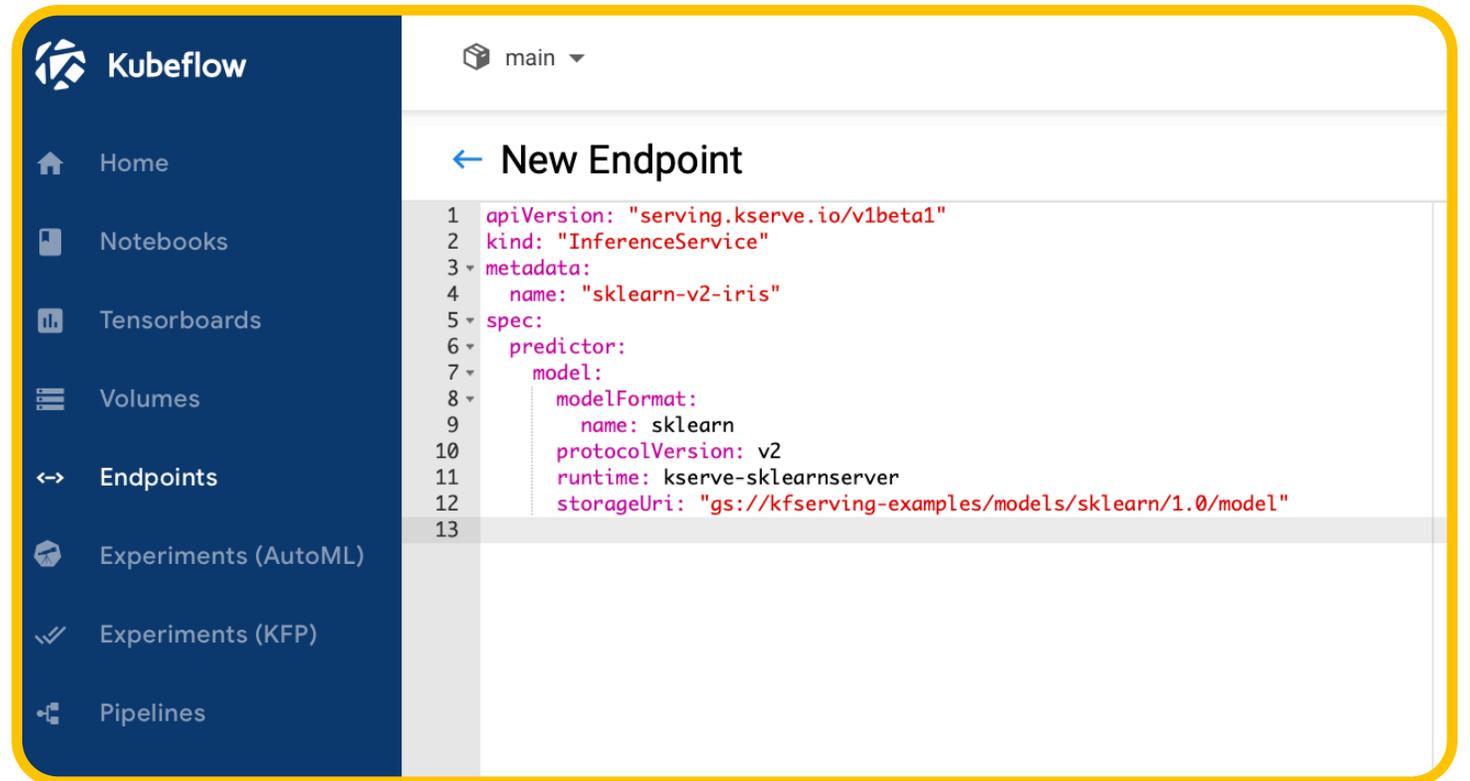
Выкатиться в коммунальный кластер

Inference: API

Как стало

Получаете в плюс:

- Мониторинг с помощью **Prometheus**
- Масштабируемость



```
1 apiVersion: "serving.kserve.io/v1beta1"
2 kind: "InferenceService"
3 metadata:
4   name: "sklearn-v2-iris"
5 spec:
6   predictor:
7     model:
8       modelFormat:
9         name: sklearn
10      protocolVersion: v2
11      runtime: kserve-sklearnserver
12      storageUri: "gs://kfserving-examples/models/sklearn/1.0/model"
13
```

Из чего состоит ML-процесс?

Research

Исследование в Jupyter Notebook на VM или локальном компьютере



Train

Обучение модели на новых данных: вручную или автоматически по расписанию



Inference

Batch Serving или непрерывный Serving



`model.predict`



Какие шаблоны и инструкции нужны

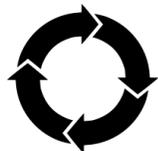
- ✓ **Инструкция по получению доступа к KubeFlow**
- ✓ **Инструкция по работе с Jupyter**
 - Base: как поднять Jupyter с нужными ресурсами
 - Pro: как поднять Jupyter с кастомным окружением (docker)
- ✓ **Шаблон ML-проекта с KF-пайплайном с контейнеризированными компонентами**
 - Как написать component (аналог task)
 - Архитектура пайплайна со связью между шагами (аналог DAG)
 - Передача данных между components (аналог XCOM)
 - Использование секретов (наша кастомная библиотека)
 - Настройка CI/CD (готовые шаги для Jenkins-пайплайна)

04

Что дает ML-Ops
лемана ПРО?

KubeFlow: **ТЕХНИЧЕСКИЕ** ИТОГИ ВНЕДРЕНИЯ

Что получили



Research + Train +
Inference в едином
контуре



Масштабируемость



Мониторинг

От чего не ушли



Экспертиза в Docker



CI/CD

Централизованная MLOps платформа

1) Выделенная команда для развития платформы

- **Повышается качество решений**

2) Своя контролируемая точка поддержки инфраструктуры

- **Повышается надежность: мониторинг, масштабирование**
- **Уменьшается bus-фактор**

3) Поддержка в написании пайплайнов:

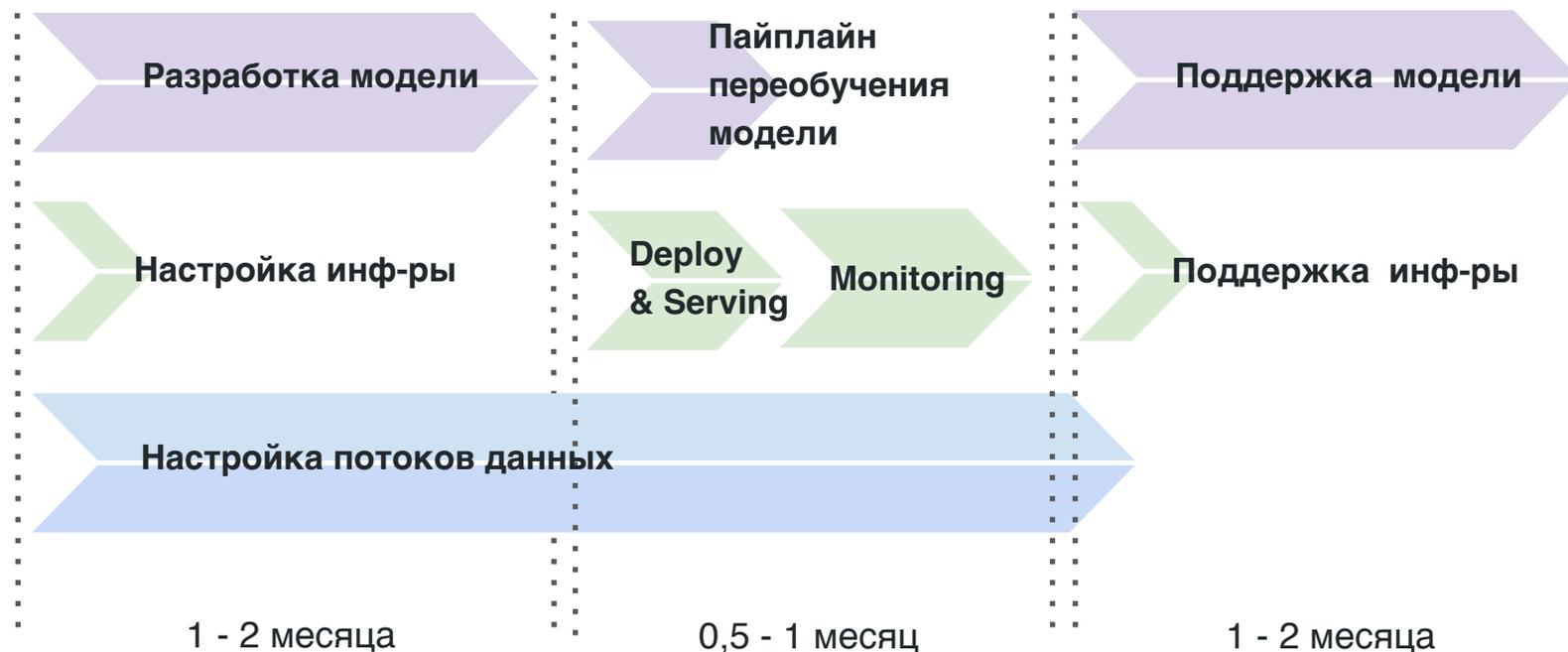
- **Переиспользование общих компонентов**

4) Траты на инфраструктуру ложатся на дата-департамент (до 2025)

5) Среда для экспериментов / аналитики

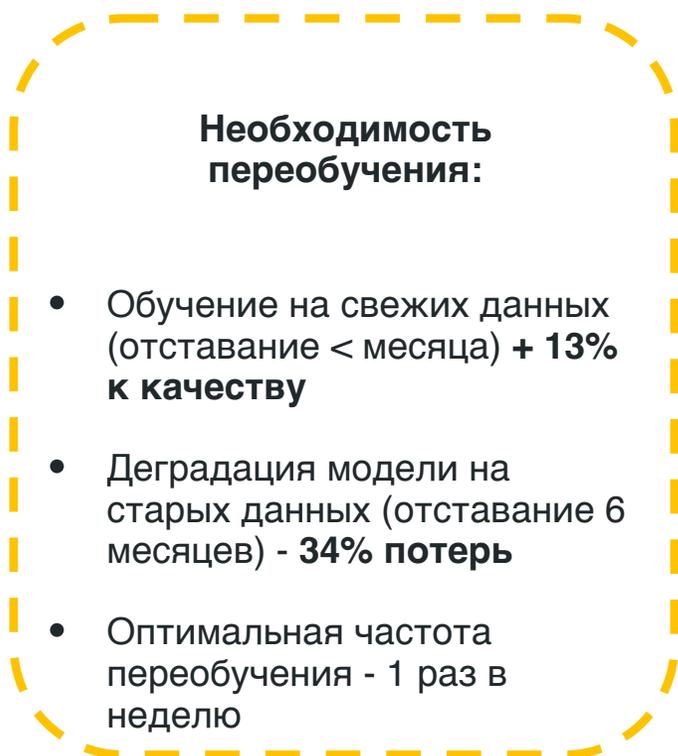
С внедрением MLOps время на разработку и запуск продукта сокращается в ~1.5 раза и требует в ~2 раза меньше ресурсов

DS/ ML-engineer
DE
DEVOps



**Мониторинг качества работы, ручное переобучение, обновление модели

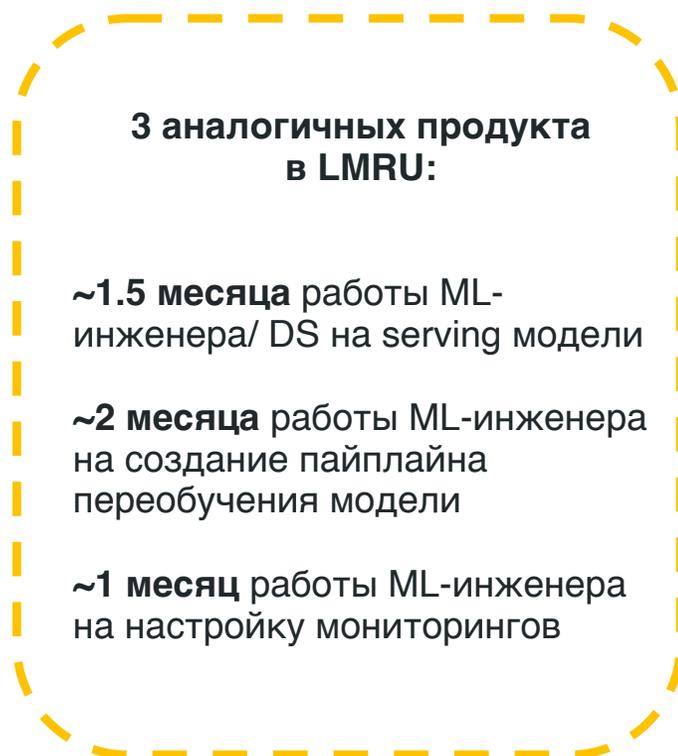
*MLops позволит отслеживать и поддерживать качество продуктов после запуска. Пример Светофора 2.0



~32 часа/ месяц

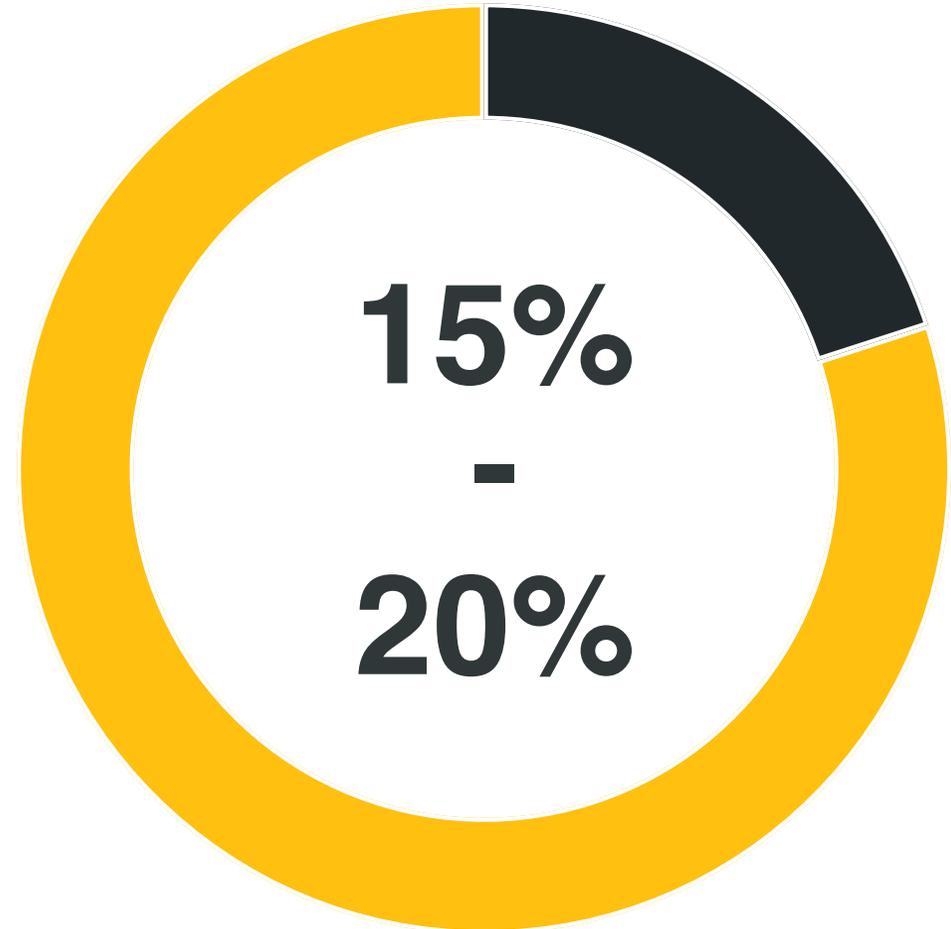


DS/ ML-engineer



Централизованная MLOps платформа

Только **15-20%**
моделей продуктивизируем **вне**
периметра Mlops-платформы



Эффект от внедрения:

- **Сокращение** time-to-market выкатки моделей в прод в 1.5 раза
- **Использование в 1.9 раза меньше** ресурсов
- **Стандартизация** продуктивизации моделей машинного обучения
- **Удобный** централизованный инструмент для проведения исследований
- **Понятная** платформа продуктивизации моделей с шаблонами и поддержкой

Вопросы?