ЛЕМАНА ПРО

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
Даркстороров

6
РЦ

2000+
Поставщиков

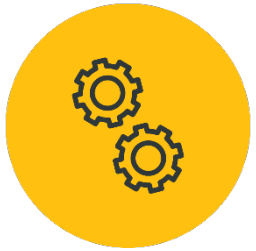
112
Магазинов

21
Бизнес-департамент

ОЧЕНЬ МНОГО ДАННЫХ

Структура команд в 2019 году

DA



DE

DS

ДАТА-ДЕПАРТАМЕНТ

2 ДАТА-ПРОДУКТА

*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

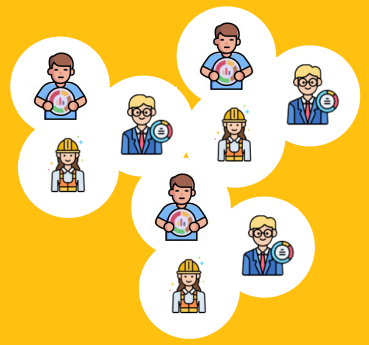
Классификация

RAG для задач поиска по документам

Batch inference

Real-time inference

Структура команд в 2024 году

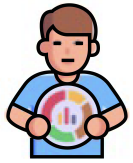


Дата-департамент

20 

Дата-команды

~35  DE

~45  DA

~30  DS

02

Мотивация для внедрения MLOps-платформы

Зоопарк в технологиях и стандартах:

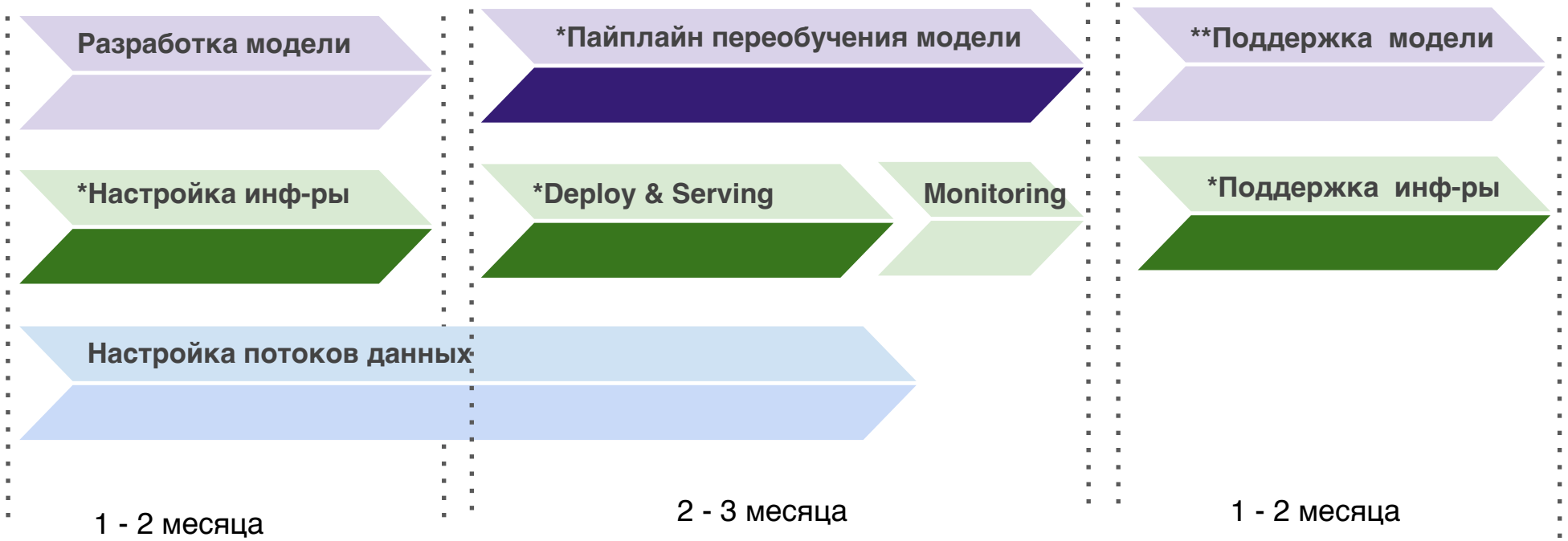
Зоопарк в технологиях и стандартах:

1. Сложный контроль качества ML-продуктов и компонент
2. Отсутствует единообразие продуктов
3. Нет общей инфраструктуры
4. Нет контролируемой точки поддержки

Подсчитали косты и TTM разработки продуктов

Ключевые этапы внедрения ML в продакшн в LMRU

DS/ ML-engineer
DE
DEVOps



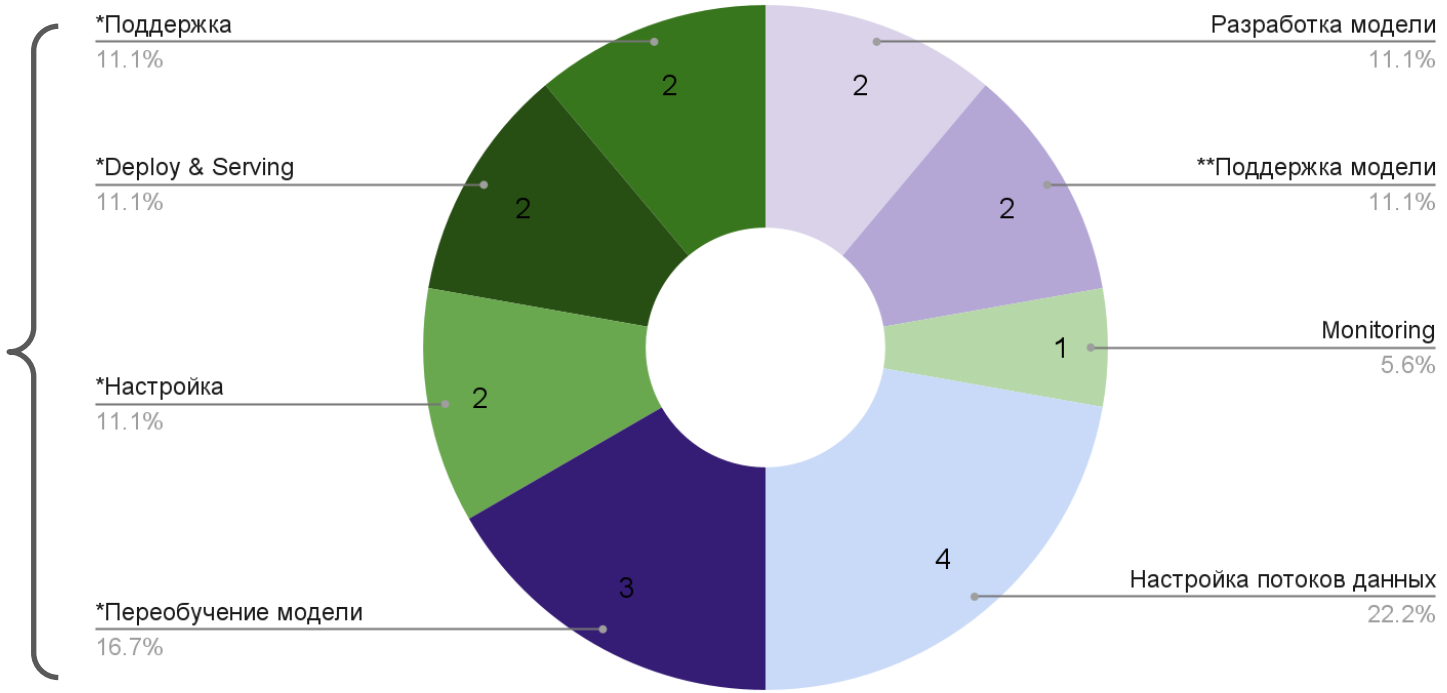
*Этапы которые **дублируются/ могут переиспользоваться** в различных ML-продуктах

**Мониторинг качества работы, ручное переобучение, обновление модели

При разработке каждого дата-продукта

DS/ ML-engineer
DE
DEVOps

~9 месяцев работы DS/ DEVops



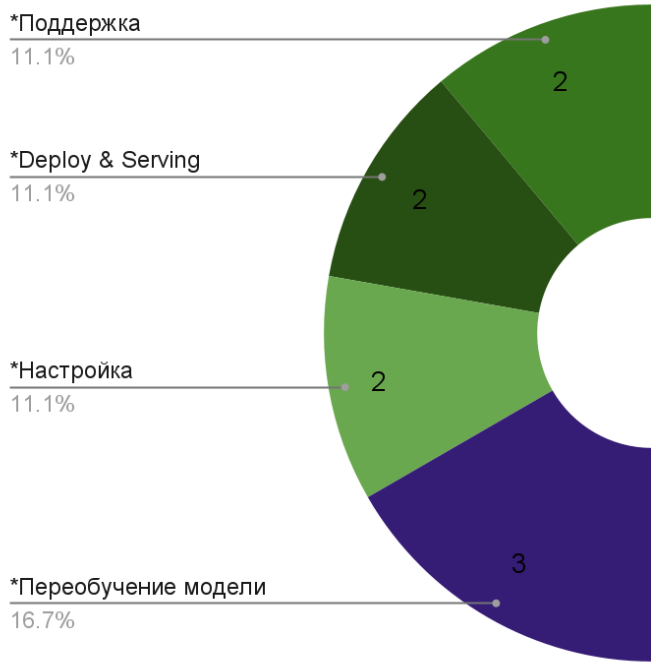
*Этапы которые могут переиспользоваться в различных ML-продуктах

**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



~6 месяцев работы Devops

~3 месяца работы DS

2021: **15** дата-продуктов

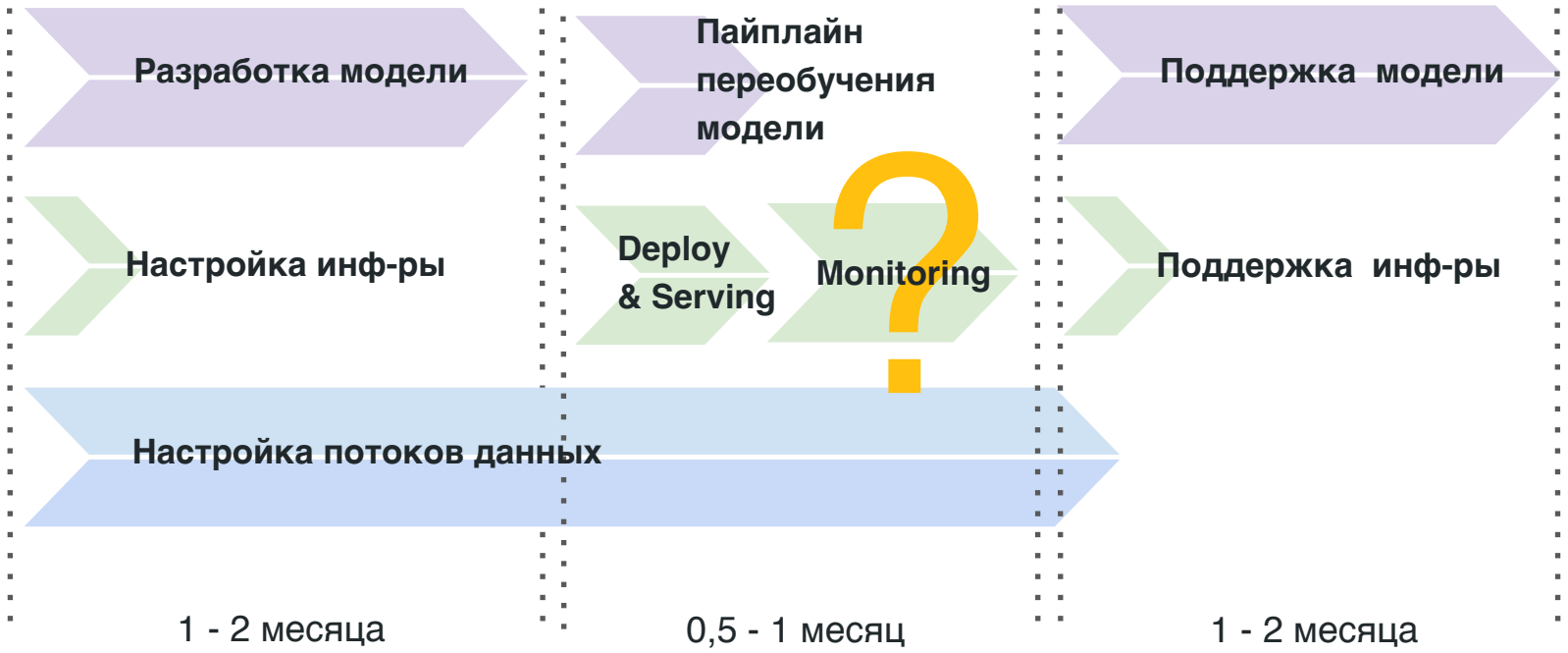
2025: **35+** дата-продуктов

*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

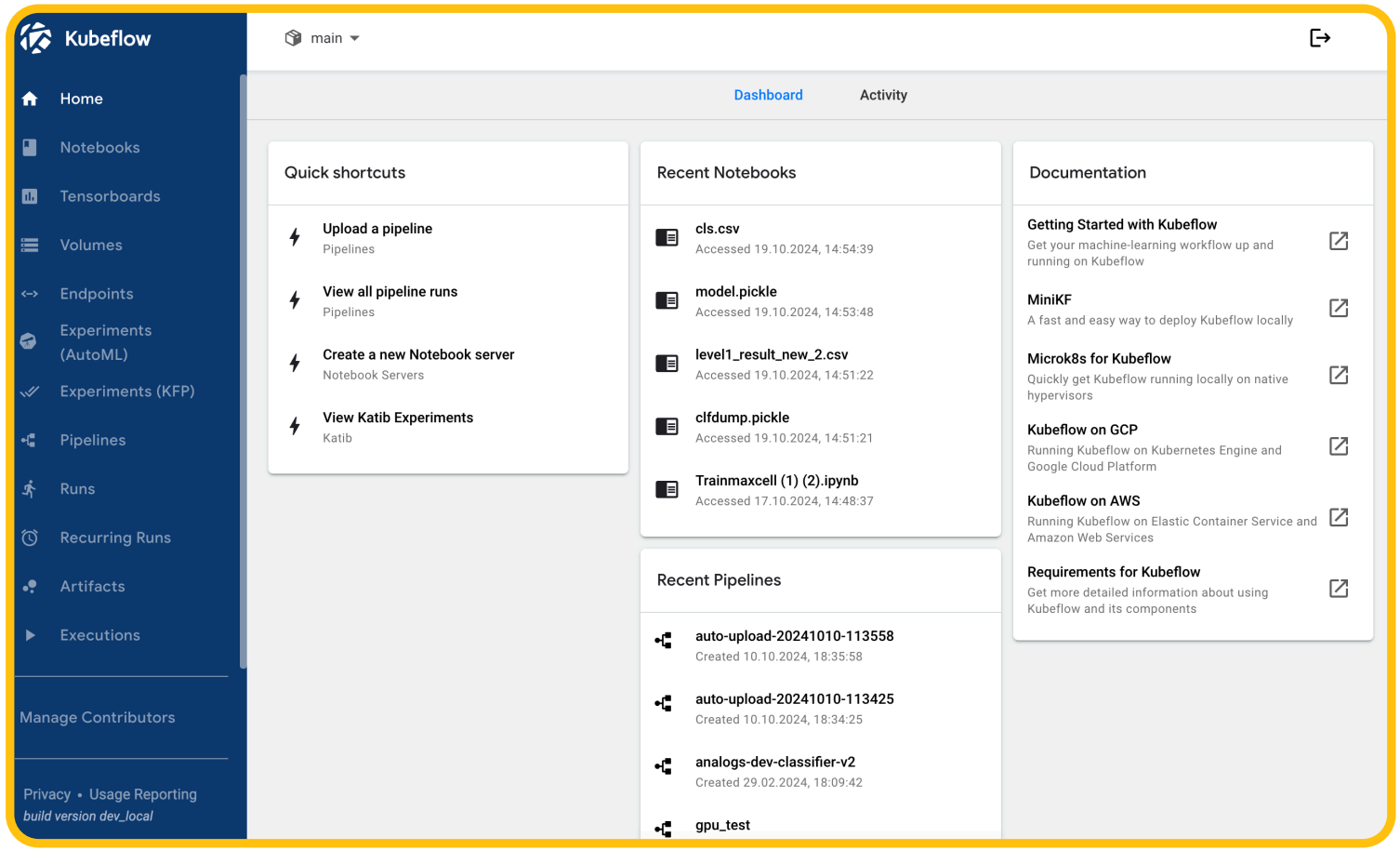
История внедрения MLOps-платформы

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:

```python
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

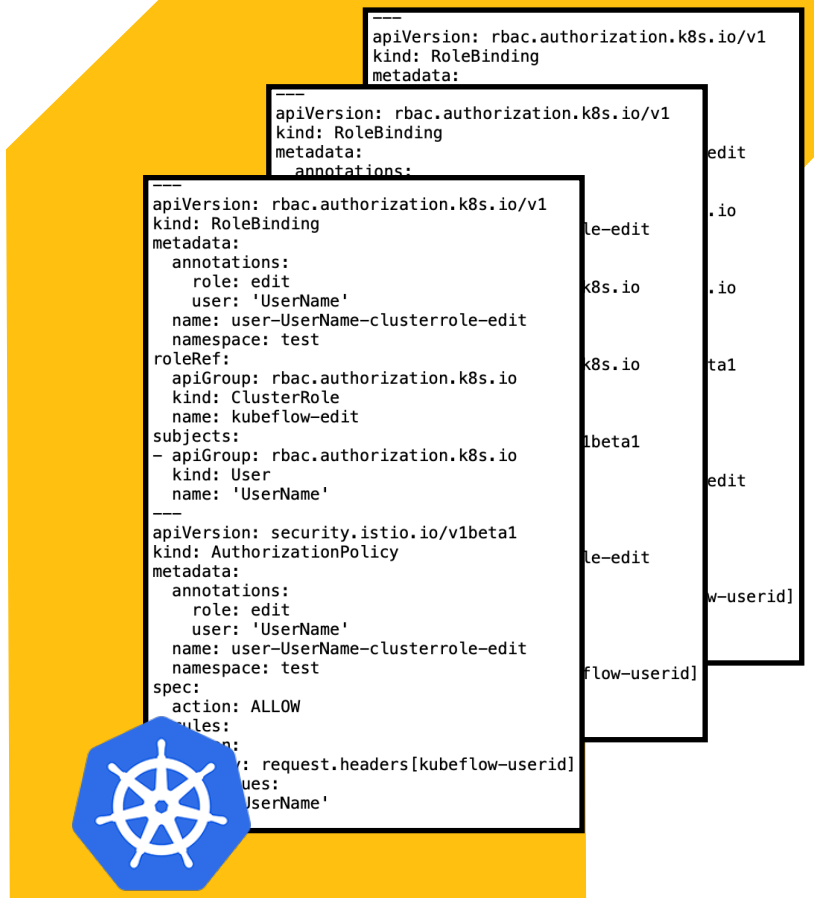
Админ создает namespace команды через k8s манифест

Описание неймспейса и лимитов:

- Profile
- LimitRange

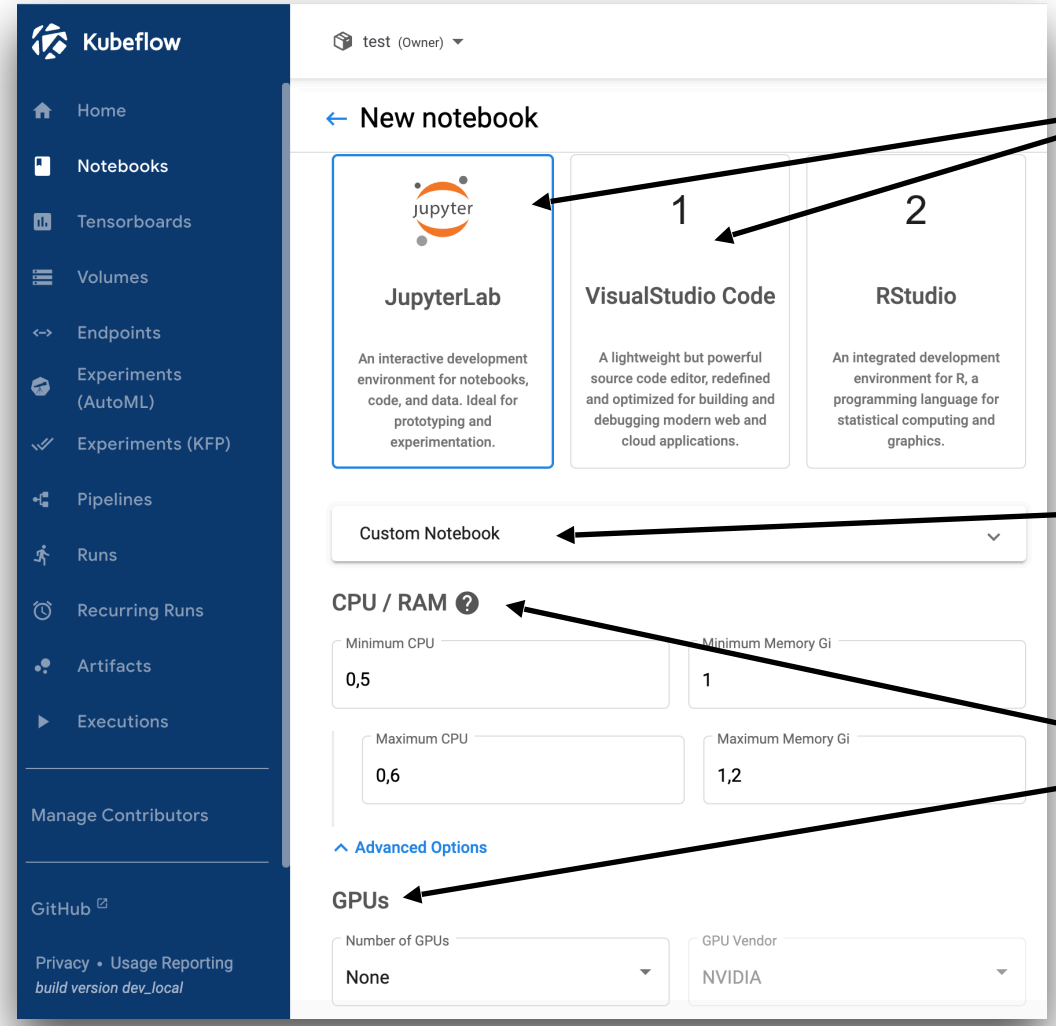
Выдача доступов к неймспейсу:

- RoleBinding
- AuthorizationPolicy



Пространство под research

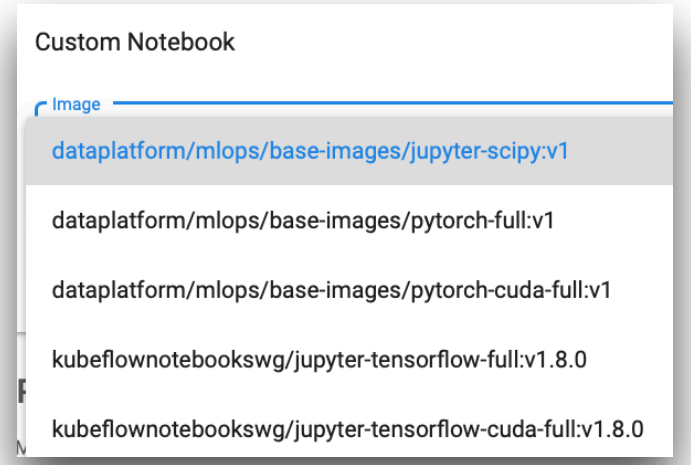
Пользователь поднимает себе сервер с Jupyter или VSCode



Тип сервера

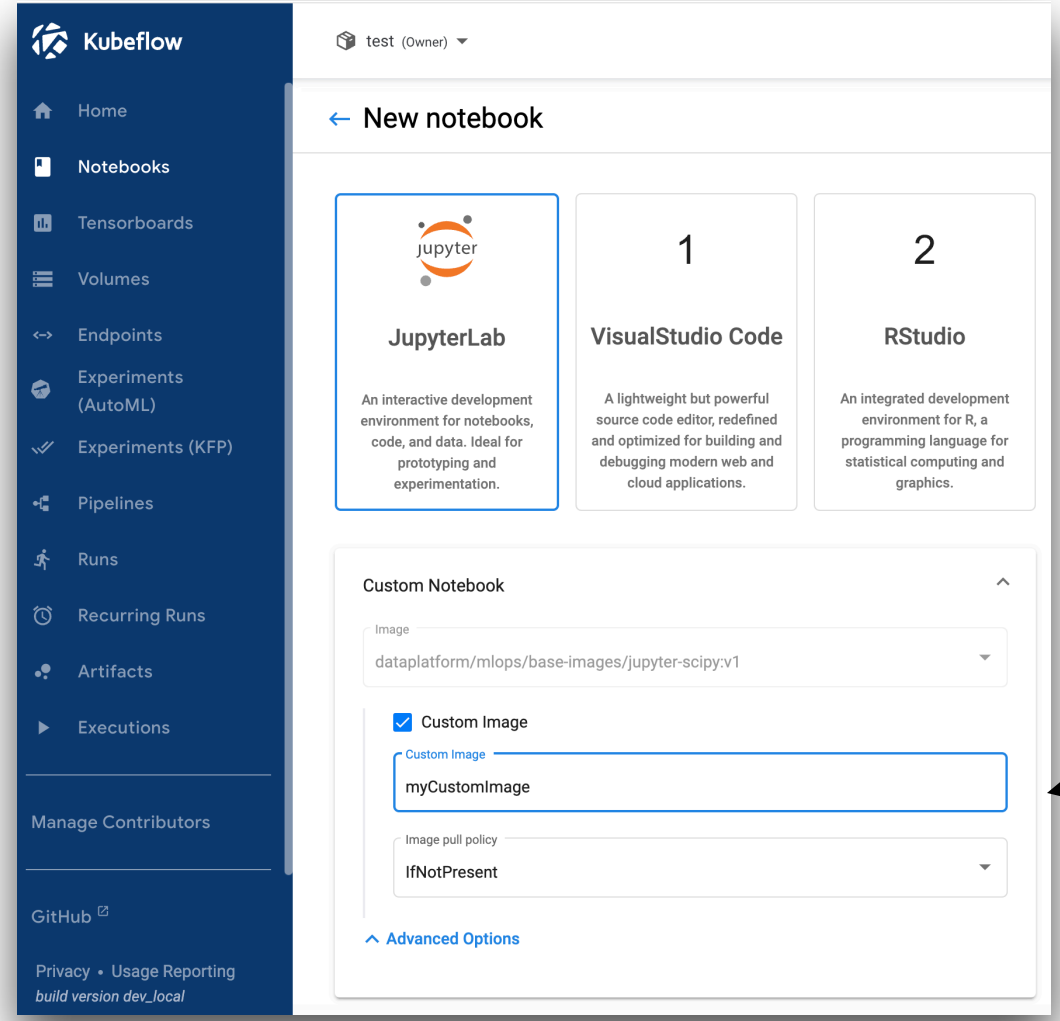
Базовый образ*

Ресурсы



Пространство под research

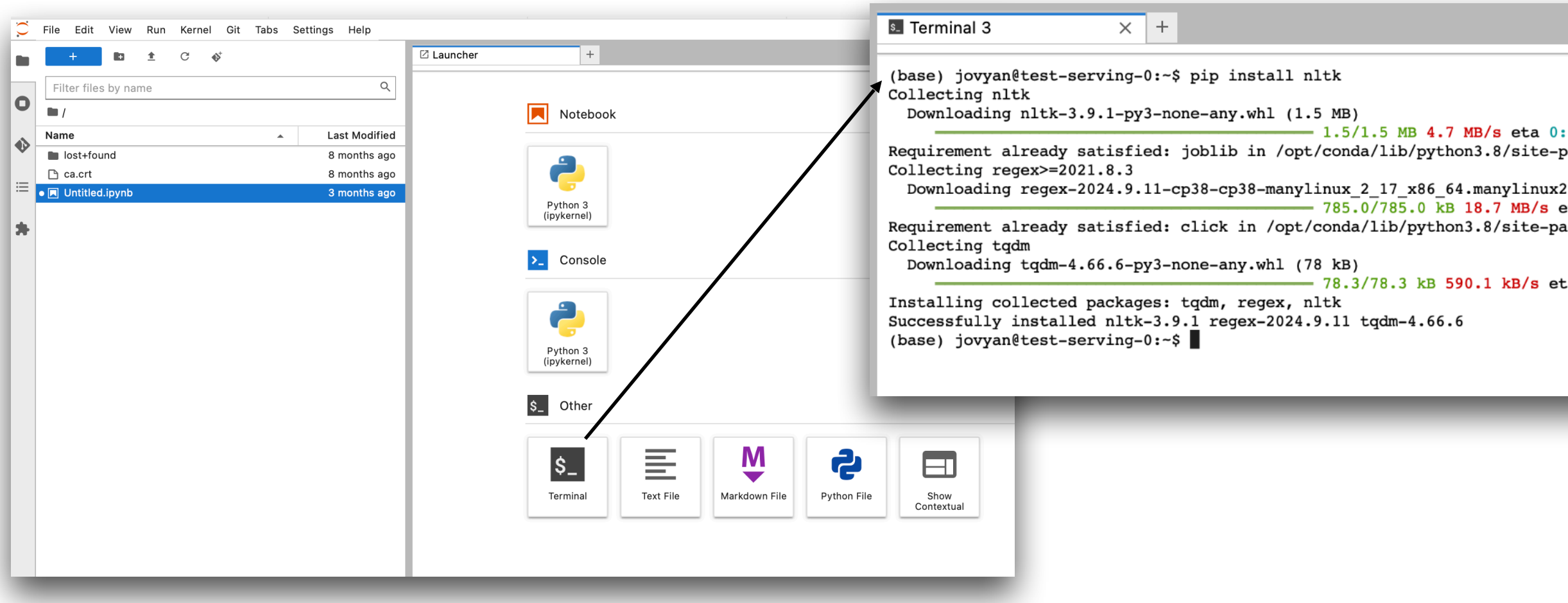
Когда нужен кастомный образ




```
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!

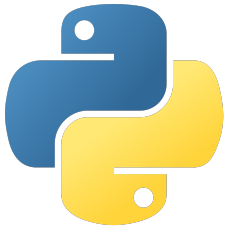


Research

Исследование в Jupyter Notebook на ВМ 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

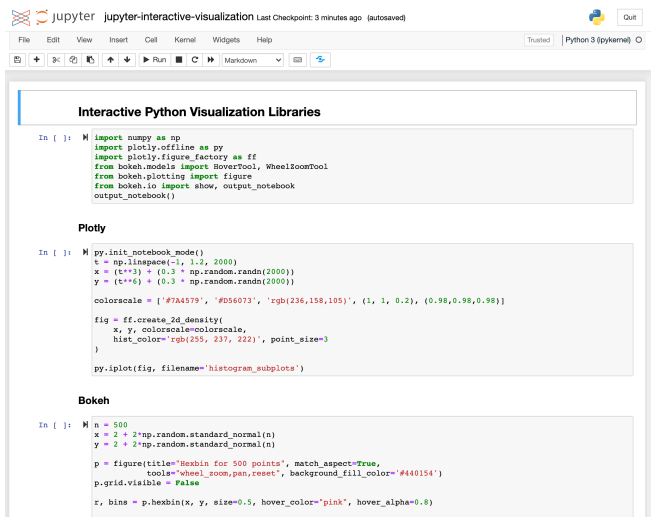
01

Залезть на BM (ssh)

```
% ssh lizvladi@12.123.123.12
```

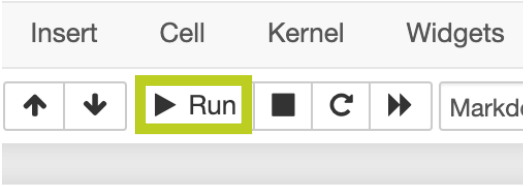
02

Подготовить Jupyter (один раз)



03

Запустить Jupyter при необходимости



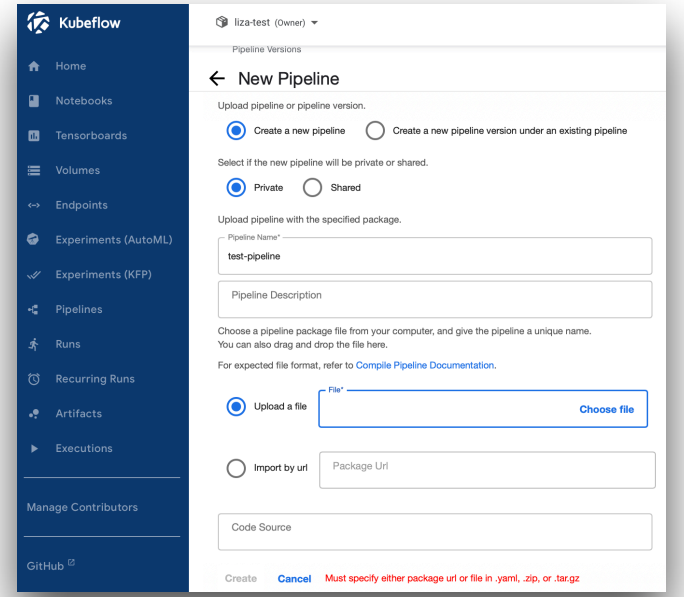
Train: pipeline в Jupyter

0 Поднять Jupyter

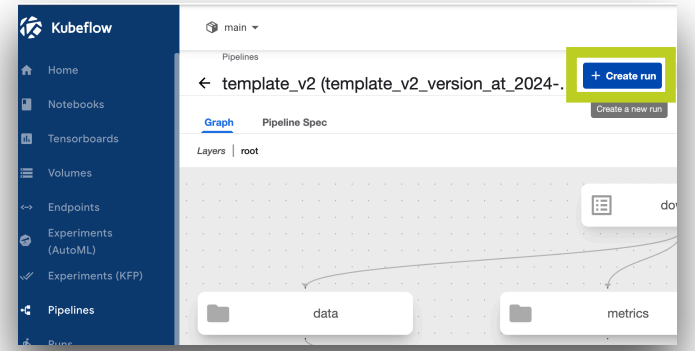
1 Подготовить пайплайн

```python
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")
```

2 Загрузить скомпилированный pipeline.yaml



3 Запустить пайплайн при необходимости



Train: pipeline в Jupyter

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

```python
import pandas as pd

from sklearn import datasets
from sklearn.linear_model import LogisticRegression
```

```python
@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"])
```

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

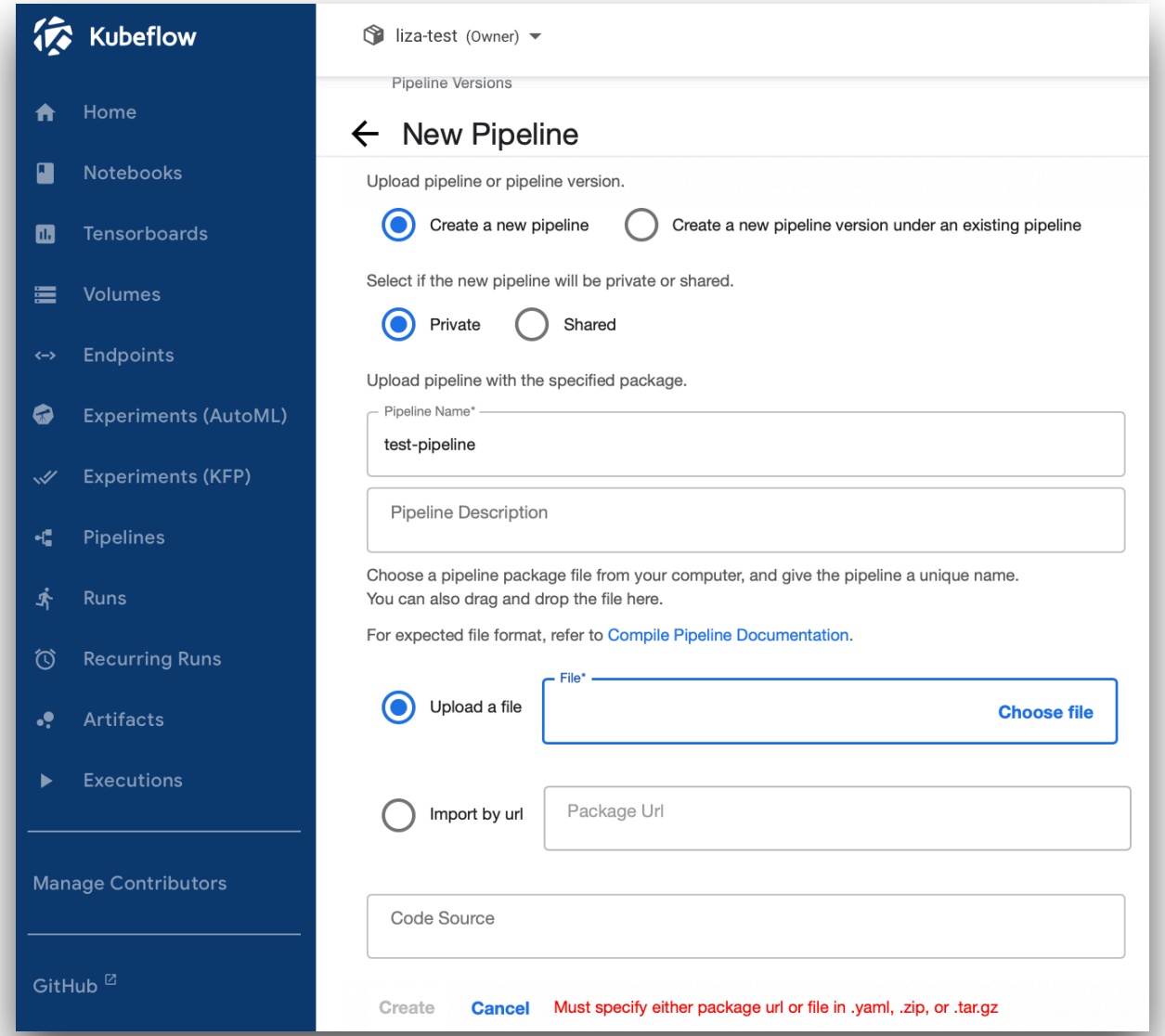
1 Подготовить пайплайн

компонент = task

пайплайн = DAG

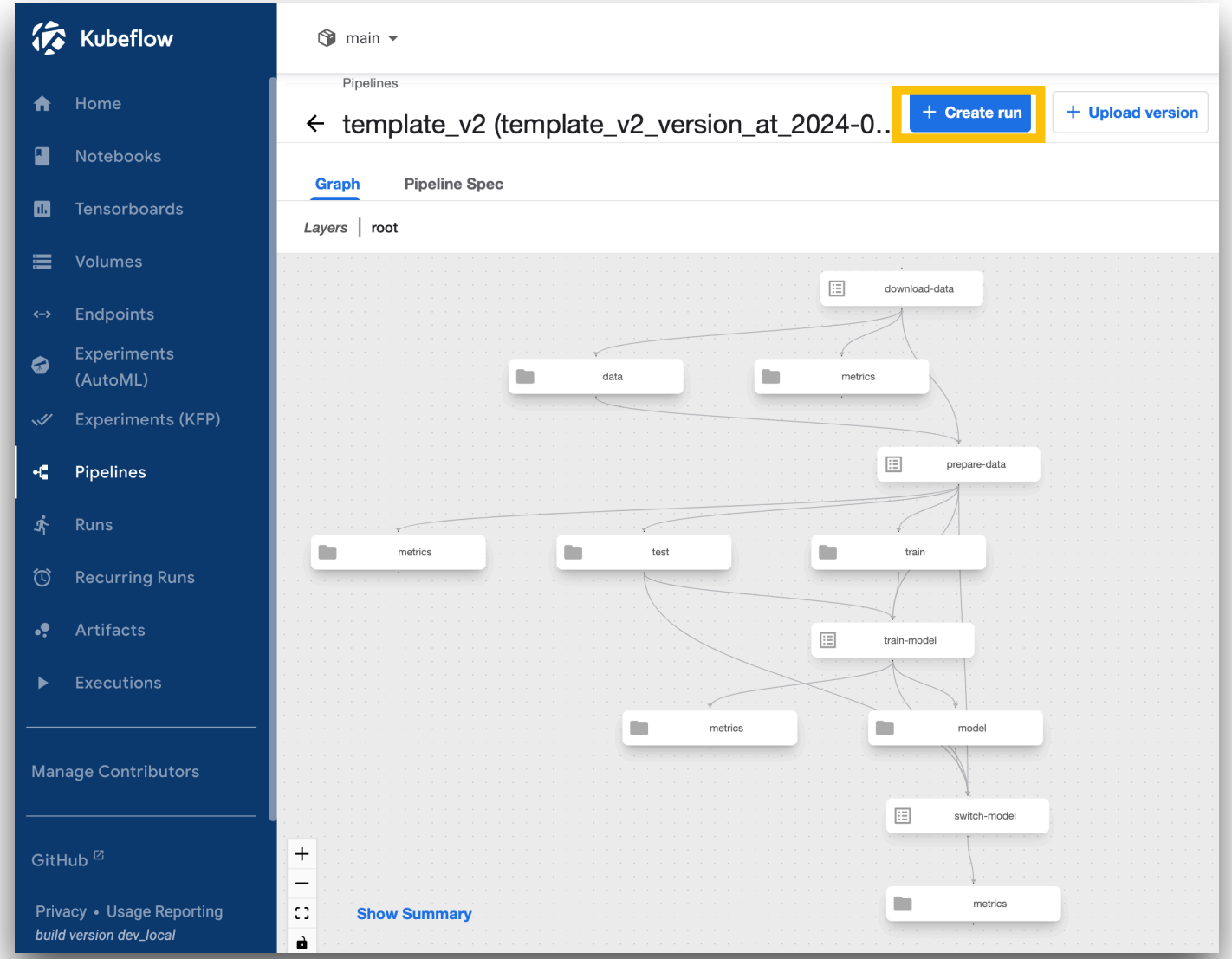
компиляция yaml с пайплайном

Train: pipeline в Jupyter



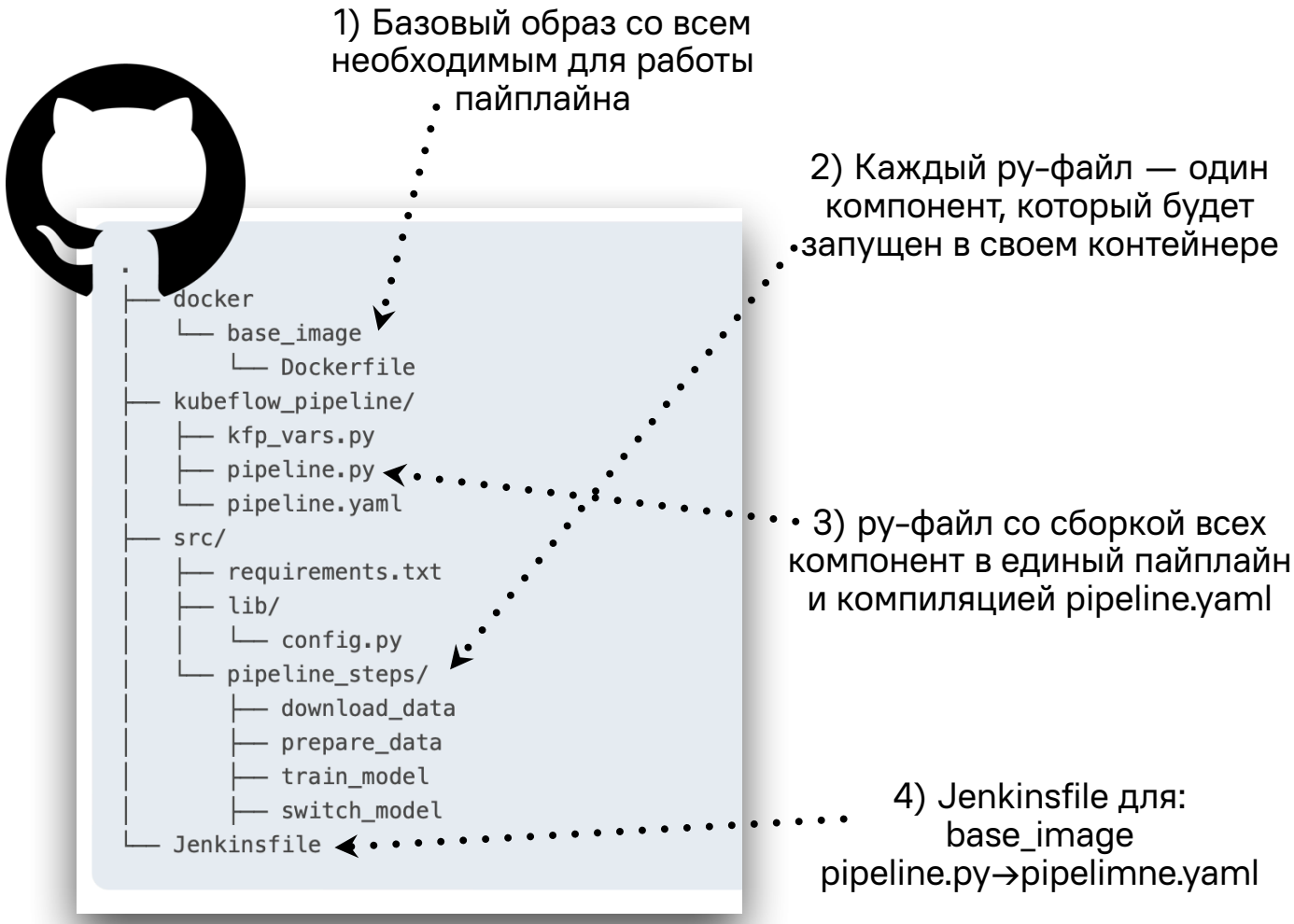
2 Загрузить скомпилированный pipeline.yaml

Train: pipeline в Jupyter



3 Запустить пайплайн при необходимости

Train: pipeline в Docker

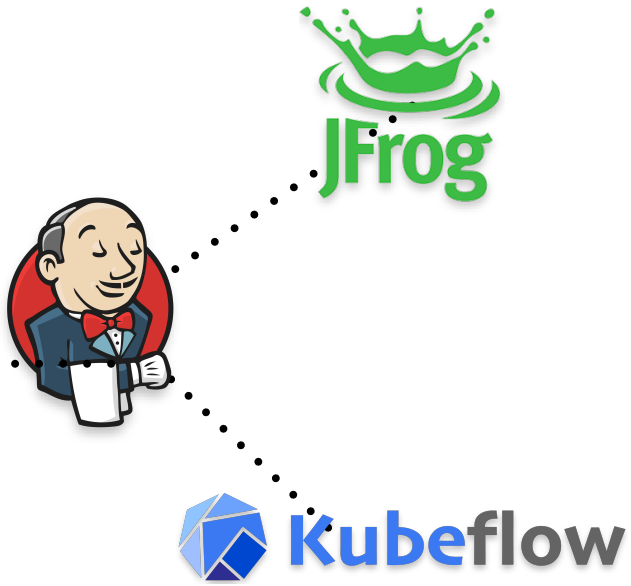


Train: pipeline в Docker

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

```python
@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

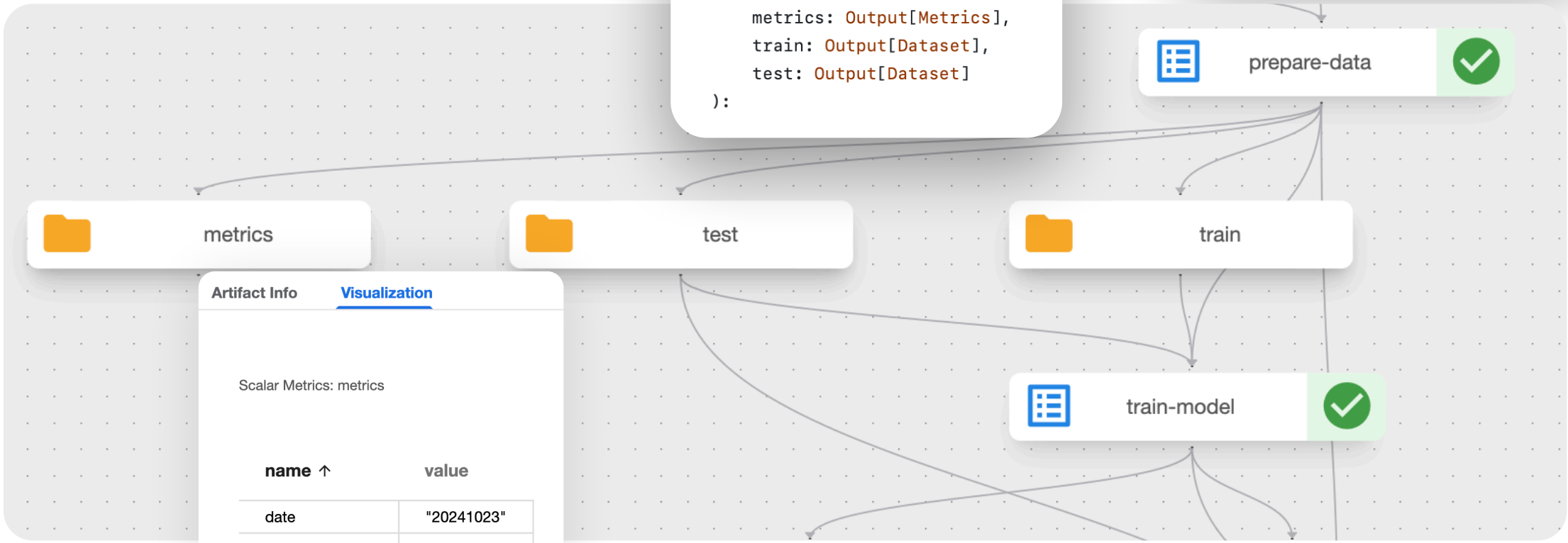
```python
@dsl.pipeline(name="project_template")
def pipeline():

    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

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

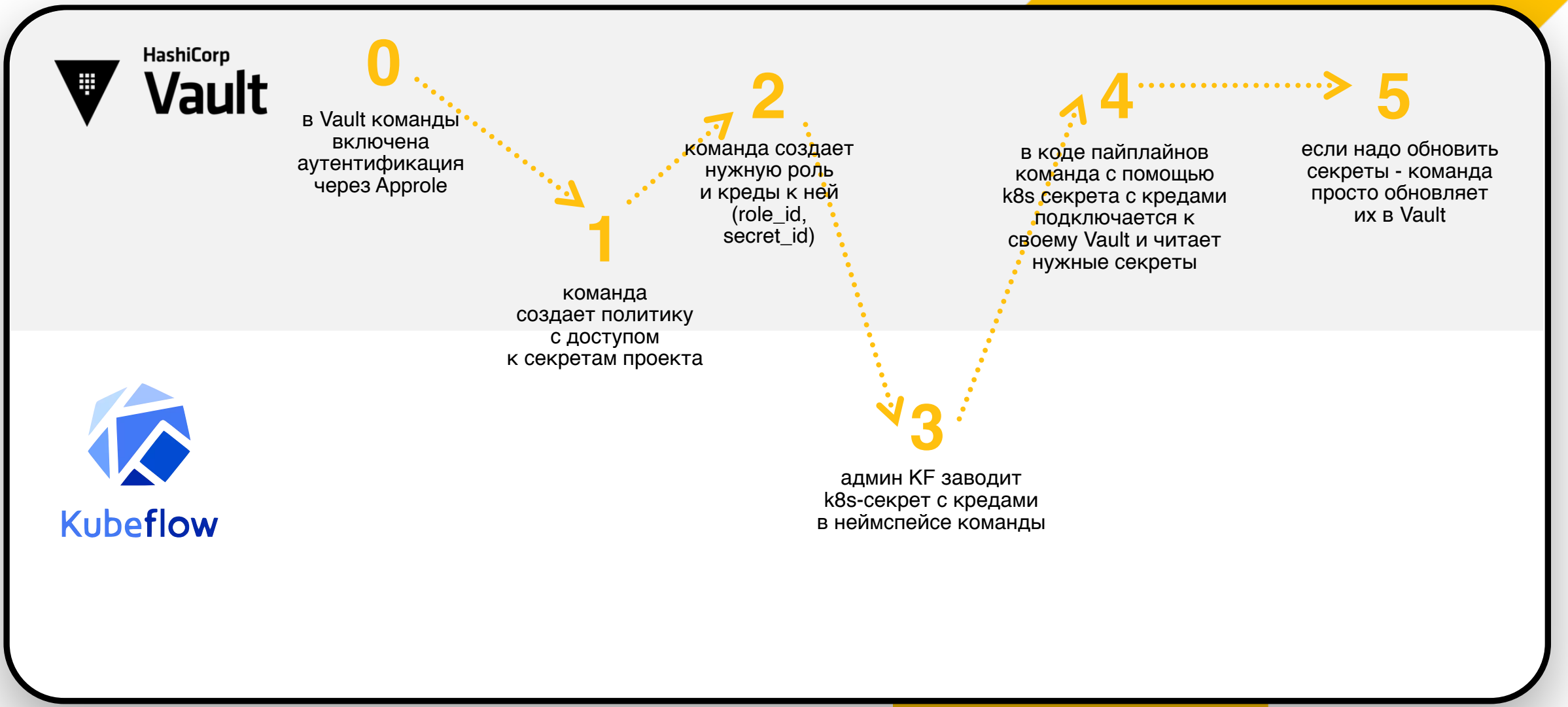


Artifact Info | Visualization

Scalar Metrics: metrics

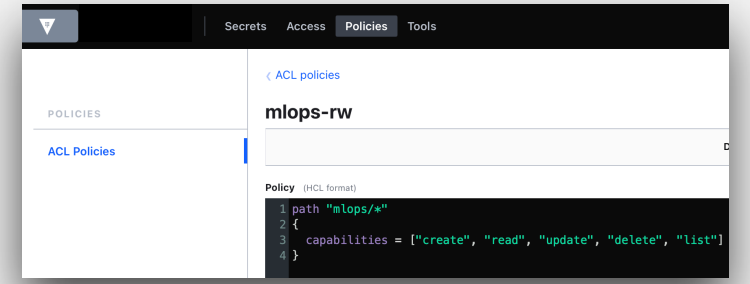
name ↑	value
date	"20241023"
split	0.2
test_size	36
train_size	142

Train: секреты



Train: секреты

HashiCorp Vault



библиотека
на основе hvac

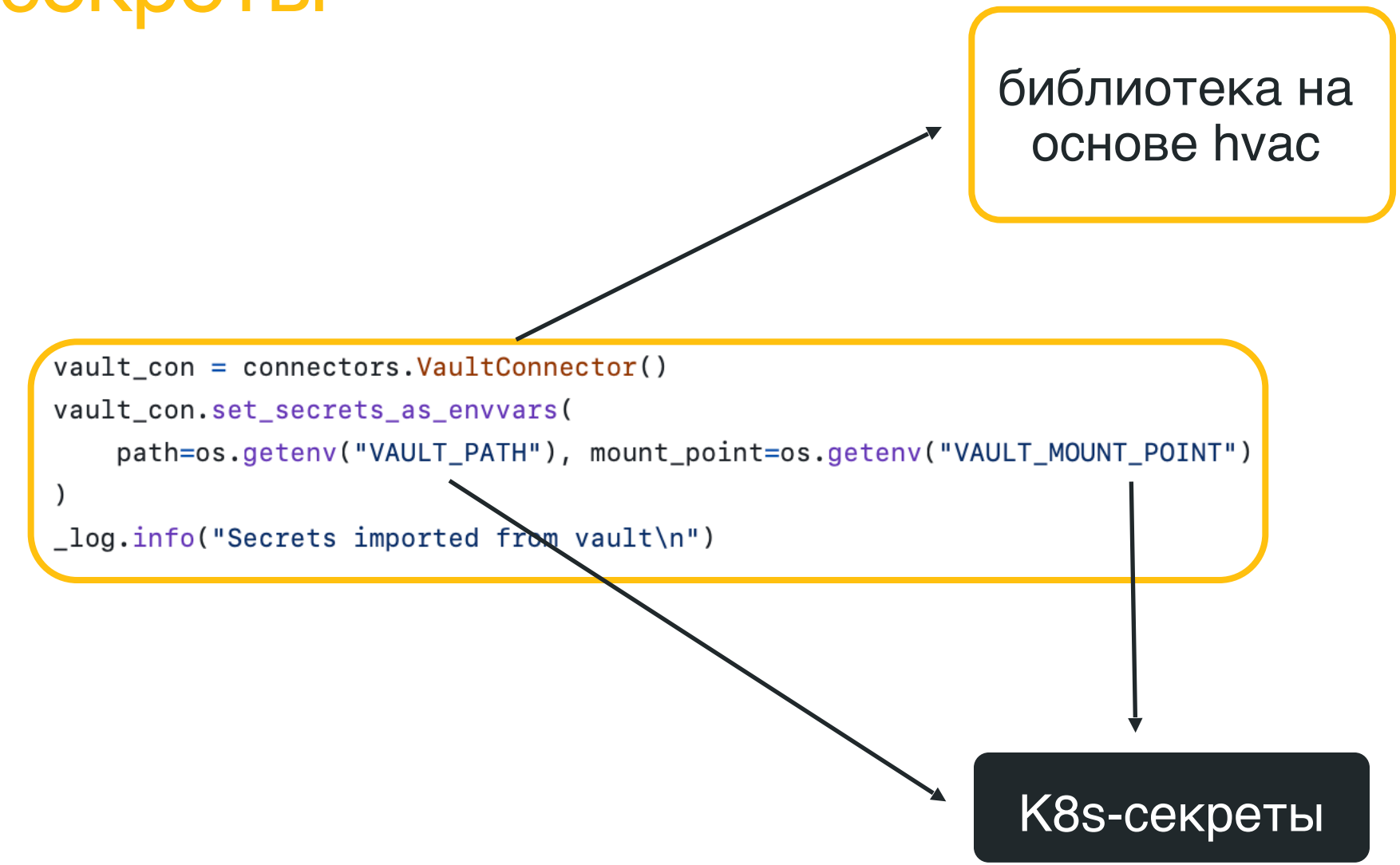
```
vault_con = connectors.VaultConnector()
vault_con.set_secrets_as_envvars(
    path=os.getenv("VAULT_PATH"), mount_point=os.getenv("VAULT_MOUNT_POINT")
)
_log.info("Secrets imported from vault\n")
```

K8s-секреты

Kubeflow

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

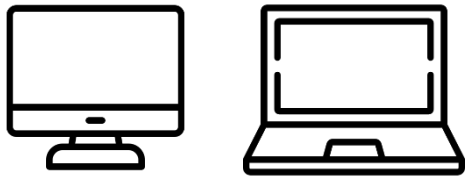
Train: секреты



Из чего состоит ML-процесс?

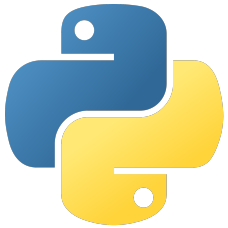
Research

Исследование в Jupyter Notebook на ВМ или локальном компьютере



Train

Обучение модели на новых данных: вручную или автоматически по расписанию

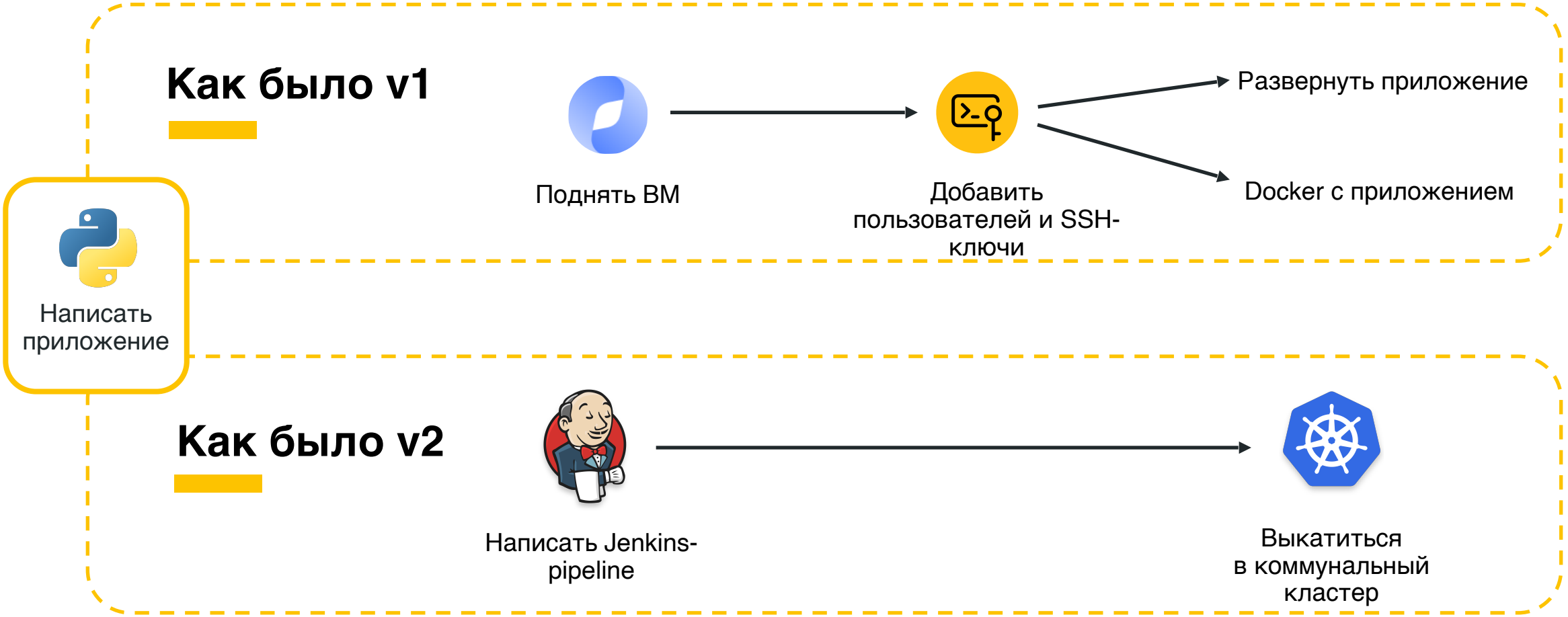


Inference

Batch Serving или непрерывный Serving


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

Inference: API

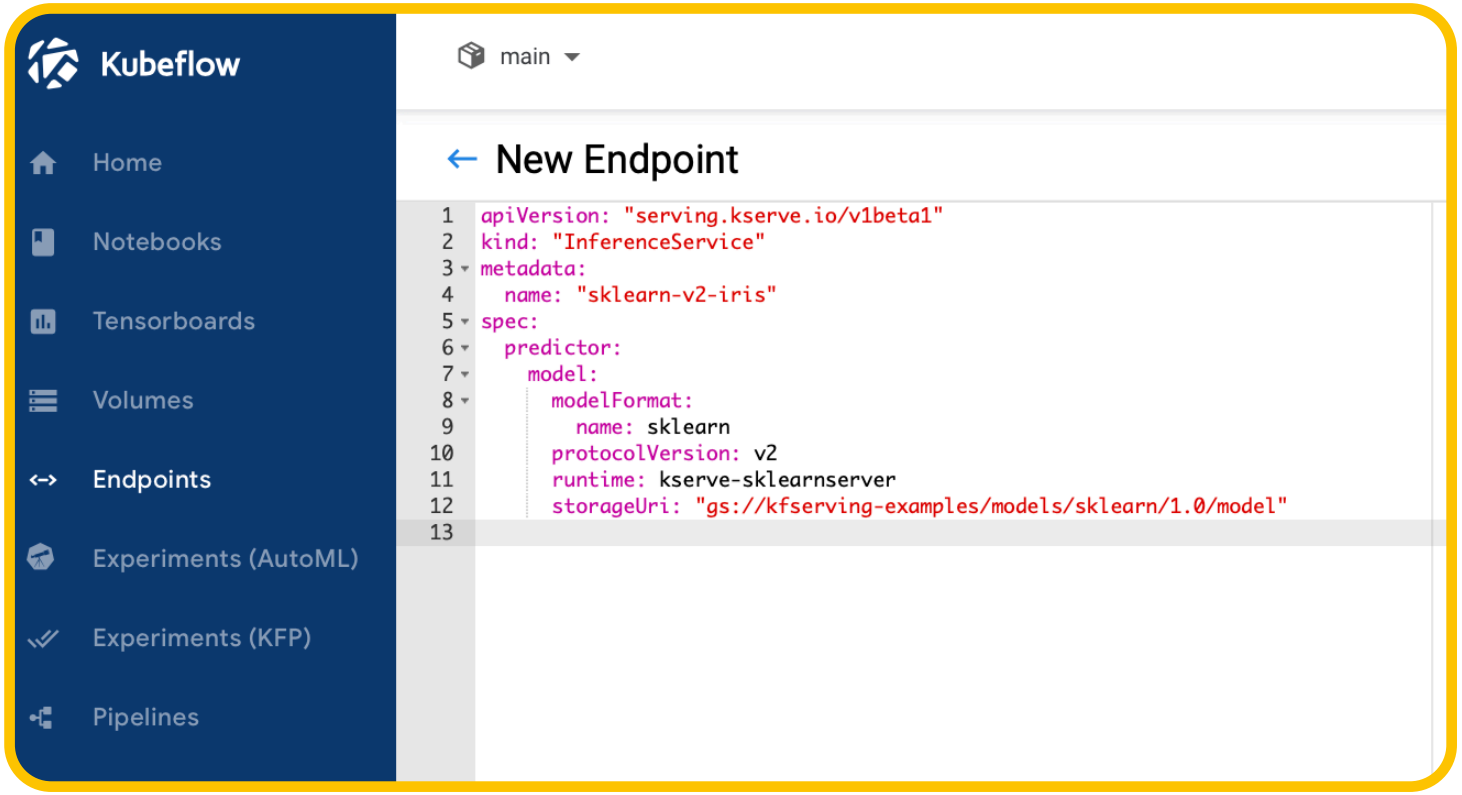


Inference: API

Как стало

Получаете в плюс:

- Мониторинг с помощью **Prometheus**
- Масштабируемость



Из чего состоит ML-процесс?

Research

Исследование в Jupyter Notebook на ВМ или локальном компьютере

Train

Обучение модели на новых данных: вручную или автоматически по расписанию

Inference

Batch Serving или непрерывный Serving


```
model.predi
```

Какие шаблоны и инструкции нужны

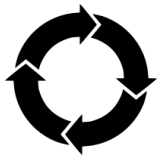
- ✓ **Инструкция по получению доступа к KubeFlow**
- ✓ **Инструкция по работе с Jupyter**
  - Base: как поднять Jupyter с нужными ресурсами
  - Pro: как поднять Jupyter с кастомным окружением (docker)
- ✓ **Шаблон ML-проекта с KF-пайплайном с контейнеризированными компонентами**
  - Как написать component (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:

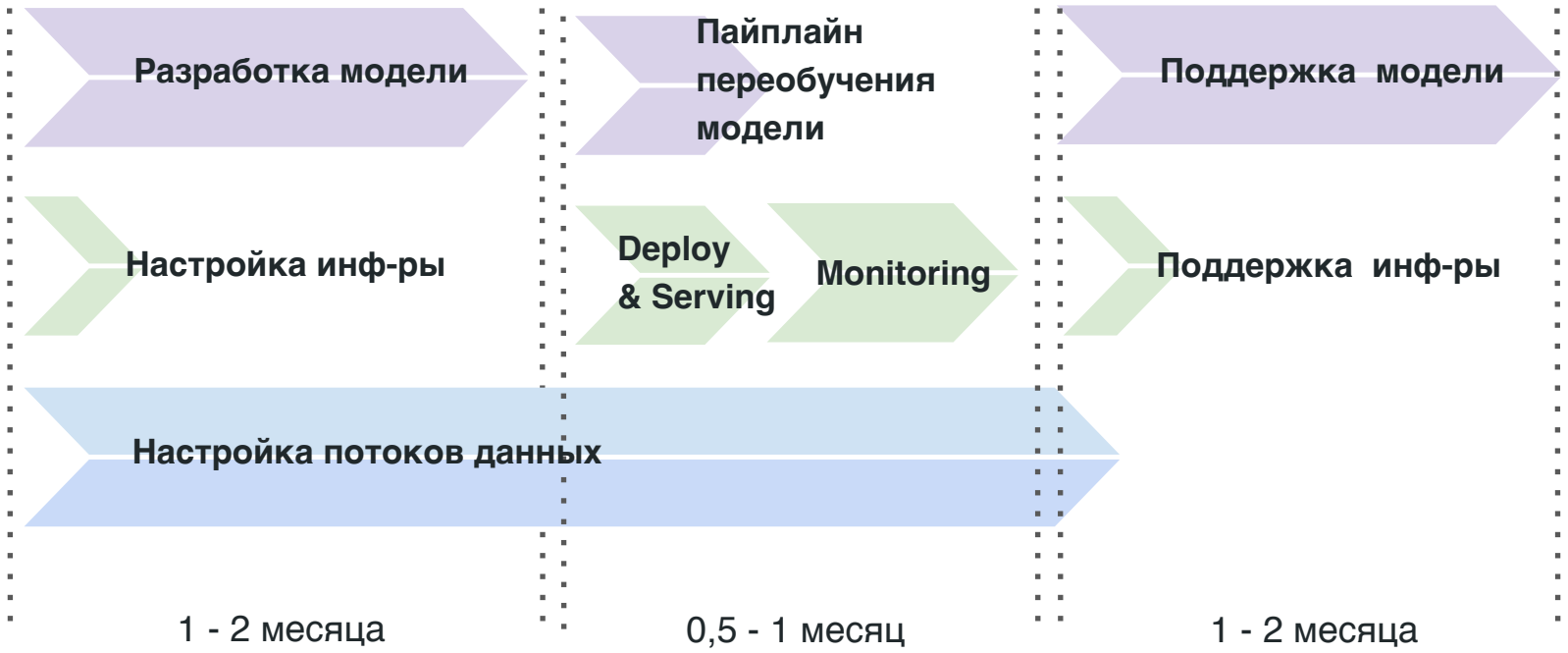
- **Переиспользование общих компонентов**

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

Необходимость переобучения:

- Обучение на свежих данных (отставание < месяца) **+ 13% к качеству**
- Деградация модели на старых данных (отставание 6 месяцев) - **34% потерь**
- Оптимальная частота переобучения - 1 раз в неделю

~32 часа/ месяц

Переобучение модели вручную

DS/ ML-engineer

3 аналогичных продукта в LMRU:

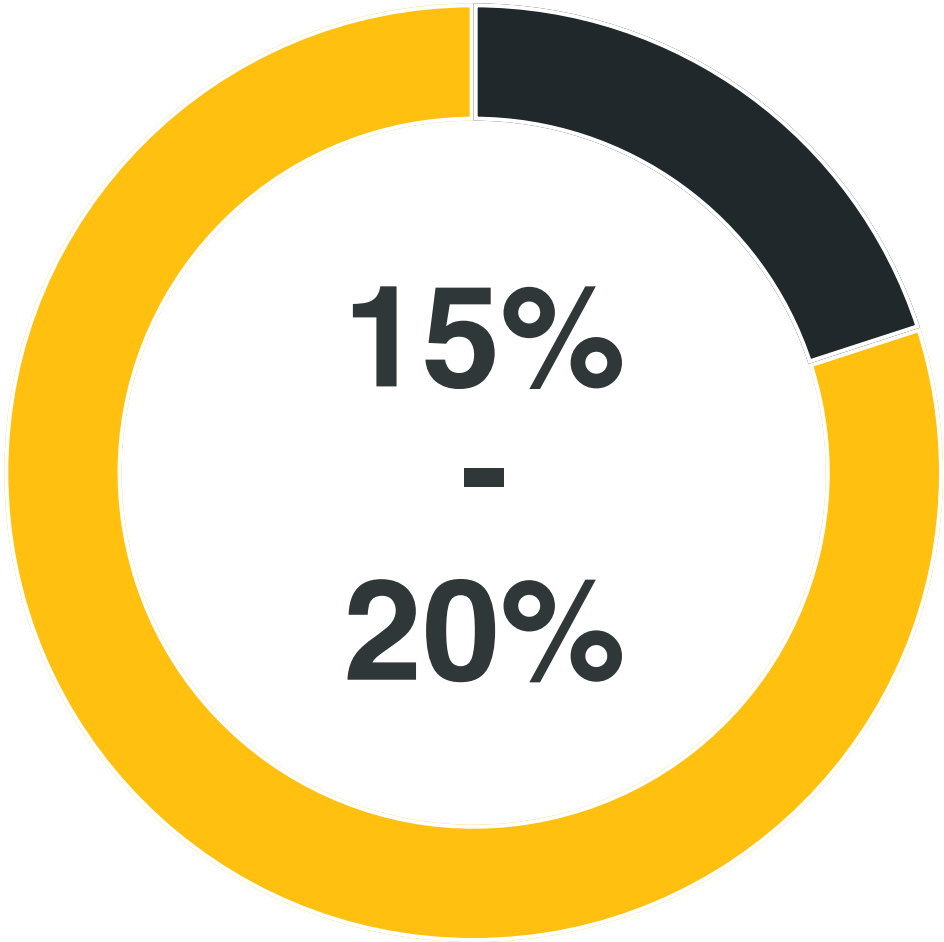
~1.5 месяца работы ML-инженера/ DS на serving модели

~2 месяца работы ML-инженера на создание пайплайна переобучения модели

~1 месяц работы ML-инженера на настройку мониторингов

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?