

ПРОМЫШЛЕННОЕ МАШИННОЕ ОБУЧЕНИЕ III

Муратов Симар

Осень
2022

Содержание

1. Структура ML проекта;
2. Anaconda;
3. Cookiecutter Data Science;
4. Ocean;
5. Catalyst;
6. Data Version Control (DVC);
7. Docker;
8. Jenkins;
9. Python для анализа данных.

Структура ML проекта: project_name folder

- Локальная версия git репозитория;
- Содержит в себе все файлы и метаданные проекта;
- В большинстве случаев архив данной папки – дистрибутив модели машинного обучения;
- Имеет осмысленное название;
- Внутри обязательно содержит конфигурацию репозитория – папку .git;
- В случае разработке на корпоративном устройстве/в облаке необходима настроенная политика доступа RWX.

Структура ML проекта: src

Содержит весь исходный код проекта:

- `train.py` – код взаимодействия с конфигурацией проекта (параметрами, путями к данным), содержит реализацию обучения модели, её сериализацию и логгирование;
- `predict.py` – код загрузки модели, вычисления метрик на предоставленных данных и сохранения результата;
- `preprocess.py` – код предварительной обработки данных, разбиения на тренировочную, тестовую и валидационную выборки;
- вспомогательные файлы (`utils.py...`) – код внешней логики взаимодействия со сторонними сервисами и любой другой код, необходимый для полноценной эксплуатации модели.

Структура ML проекта: src

Рекомендации по содержанию файлов исходного кода:

- внутри каждого файла реализован класс с полями и методами, отвечающими соответствующему функционалу;
- код снабжен полноценными и, в то же время, лаконичными комментариями;
- код отвечает всем требованиям [PEP8](#);
- код имеет необходимый процент покрытия тестами;
- код адекватен заявленному содержанию.

Структура ML проекта: notebooks

- Папка хранит файлы Jupyter Notebook;
- Допустимо сохранение нескольких .ipynb для возможности отслеживания истории разработки и реализации модели;
- Для уменьшения размера итогового дистрибутива рекомендуется очистить вывод ячеек;
- Наличие комментариев менее важно, чем наличие ячеек с описанием обоснования выбора модели, выбора её параметров и подходов к обработке данных;
- Допустимо сохранение наиболее важных примеров визуализации данных в виде отдельных .jpg файлов.

Структура ML проекта: data

Вариант 1

Папка содержит все данные: исходные, обработанные, прошедшие feature engineering и т.д.

Вариант 2

Папка содержит файл с ссылками на размещение данных в облачном хранилище.

В 1 варианте ожидается использование DVC, во втором – синхронизация работ с данными отдаётся на откуп облачному провайдеру/системным архитекторам.

Структура ML проекта: experiments

Директория содержит описания экспериментов над моделью. Любое архитектурное изменение модели порождает новый эксперимент – подпапку данной папки. Один эксперимент отождествляется следующей совокупностью файлов:

- `config.yml` (json, xml) – данные для воспроизводимости эксперимента: параметры модели, путь до данных, тип модели, директорию логгирования, хэш полученного `.pkl` модели;
- `trained_model` – `.pkl` файл модели;
- `metrics.yml` (json, xml) – файл с метриками и директориями модели и использованных данных;
- `logs.txt` – результаты логгирования.

Структура ML проекта: tests

Папка содержит сценарии тестов модели в формате yml/json/xml.

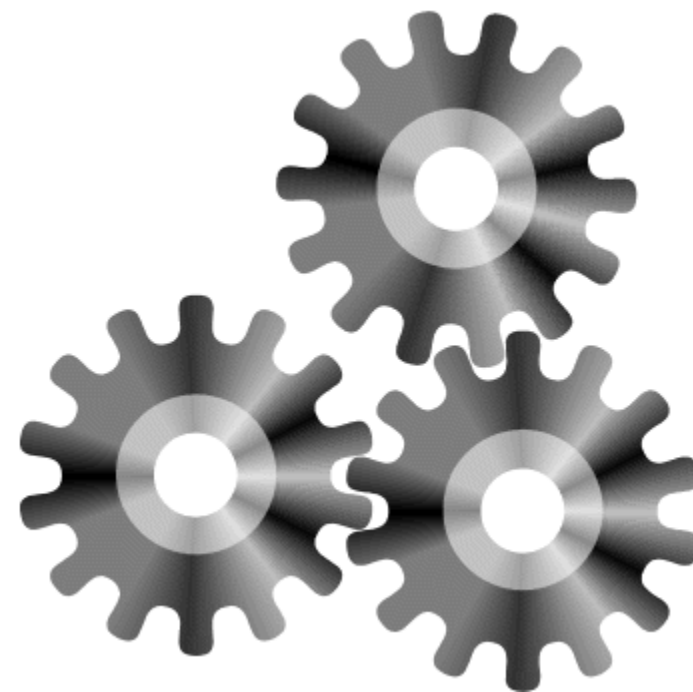
Тесты, допустимые к проведению самим дистрибутивом модели, принадлежат к следующим категориям:

- smoke тесты;
- функциональные тесты;
- регрессионные тесты;
- A/B тесты.

Не рекомендуется наполнять дистрибутив сценариями нагрузочного и интеграционного тестирования.

Структура ML проекта: конфигурация

- config.ini;
- Dockerfile;
- docker-compose.yml;
- requirements.txt;
- dev_sec_ops.yml;
- scenario.json;
- Makefile/main.py;
- readme.md;
- .gitignore...



Структура ML проекта: пример

```
project_name
├── .conda
├── data
│   ├── iris_orig.csv
│   ├── iris_train.csv
│   ├── iris_test.csv
│   ├── iris_valid.csv
│   ├── featured_iris.csv
│   └── .dvc
├── experiments
│   └── exp_0
│       ├── config.yml
│       ├── trained_model.pkl
│       ├── metrics.yml
│       └── logs.txt
├── notebooks
│   ├── knn_iris_22.10.22.ipynb
│   └── lstm_iris_22.11.22.ipynb
├── src
│   ├── train.py
│   ├── predict.py
│   ├── preprocess.py
│   └── utils.py
├── tests
│   ├── test_0.json
│   ├── test_1.json
│   └── test_2.json
├── config.ini
├── Dockerfile
├── docker-compose.yml
├── requirements.txt
├── dev_sec_ops.yml
├── scenario.json
├── Makefile
├── readme.md
└── .gitignore
```

Nice paper



GitHub
Link



Written in
your favourite
framework



Runs smoothly on
your system
without error or
dependency issues





Conda: что это?

Conda — это менеджер пакетов с открытым кодом и система управления средой, которая работает на Windows, macOS и Linux.

Conda проста в установке, выполнении и обновлении пакетов и зависимостей. **Conda** легко создает, сохраняет, загружает и переключается между средами на локальном компьютере.

Она задумывалась для программ на Python, но может создавать пакеты и дистрибутивы программного обеспечения на любом языке.



Conda: в чём отличие от pip?

Pip работает с Python и пренебрегает зависимостями из не-Python библиотек (HDF5, MKL, LLVM), в исходном коде которых отсутствует файл установщика. Pip – это менеджер пакетов, который облегчает установку, обновление и удаление пакетов Python. Он работает с виртуальными средами Python.

Conda – это менеджер пакетов для любого программного обеспечения (установка, обновление, удаление). Он работает с виртуальными системными средами.



Anaconda: зачем нужно?

Anaconda — это дистрибутивы Python и R. Он предоставляет все необходимое для решения задач по анализу и обработке данных (с применимостью к Python).

Anaconda — это набор бинарных систем, включающий в себя Scipy, Numpy, Pandas и их зависимости.

Anaconda полезна тем, что объединяет все это в единую систему. Двоичная система **Anaconda** — это установщик, который собирает все пакеты с зависимостями внутри вашей системы.



Анаконда: удобно и быстро

Анаконда это:

- Включает предустановленный Python 2 и 3;
- +-150 предустановленных библиотек, и более 200-300 готовых к "легкой" установке библиотек командой `conda install name_lib`;
- Включает в себя IDLE Spider.

Документация: <https://www.anaconda.com/products/distribution>




Anaconda: важное

- `pip` — это менеджер пакетов для Python;
- `venv` — является менеджером среды для Python;
- `conda` — является одновременно менеджером пакетов и среды и не зависит от языка;
- `venv` создает изолированные среды только для разработки на Python, а `conda` может создавать изолированные среды для любого поддерживаемого языка программирования;
- можно либо включать `.conda/.venv` в дистрибутив и не ставить зависимости на стенд, либо добавить данные директории в `.gitignore` и не включать в дистрибутив, сконфигурировав установку зависимостей на стенд перед раскаткой модели.



Anaconda: навигатор

 **ANACONDA** NAVIGATOR

Sign in to Anaconda Cloud

Home

Environments


Learning


Community


Documentation


Developer Blog

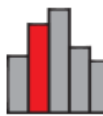
Applications on base (root) Channels Refresh



JupyterLab
1.1.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
[Launch](#)



Notebook
6.0.1
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
[Launch](#)


Spyder
3.3.6
Scientific PYTHON Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
[Launch](#)


VS Code
1.39.2
Streamlined code editor with support for development operations like debugging, task running and version control.
[Launch](#)

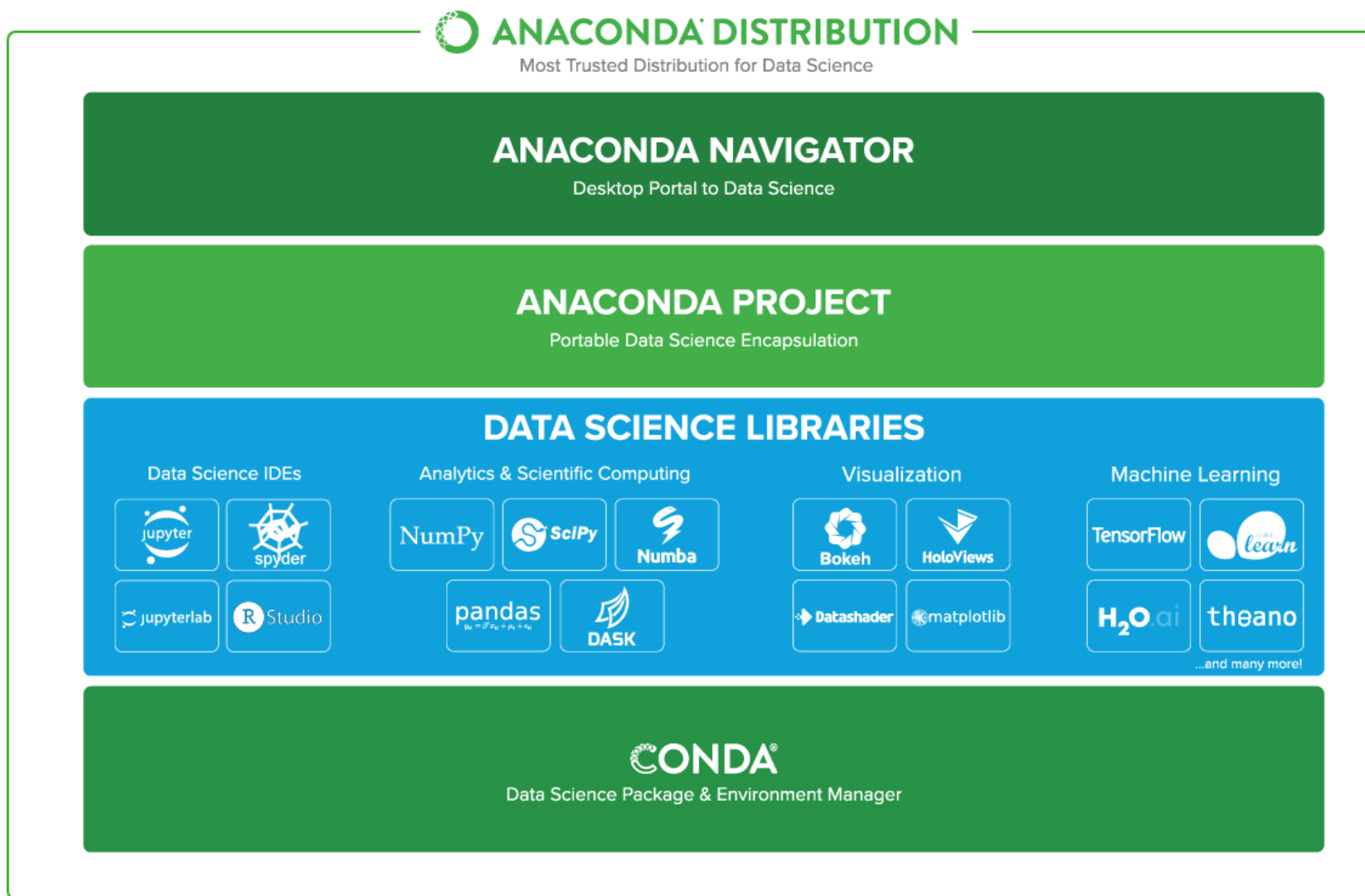

Glueviz
0.15.2


Orange 3
3.23.1

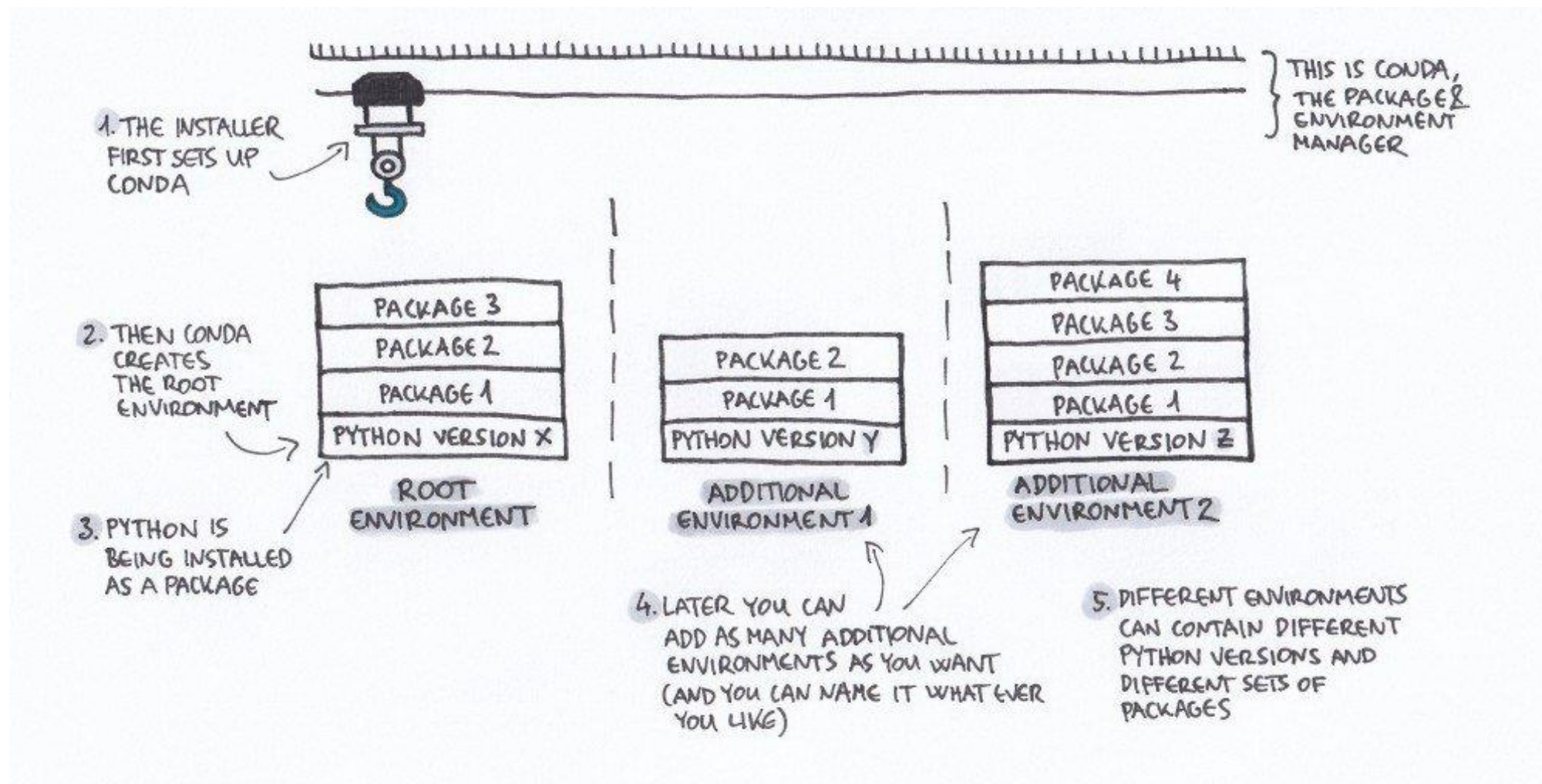

RStudio
1.1.456



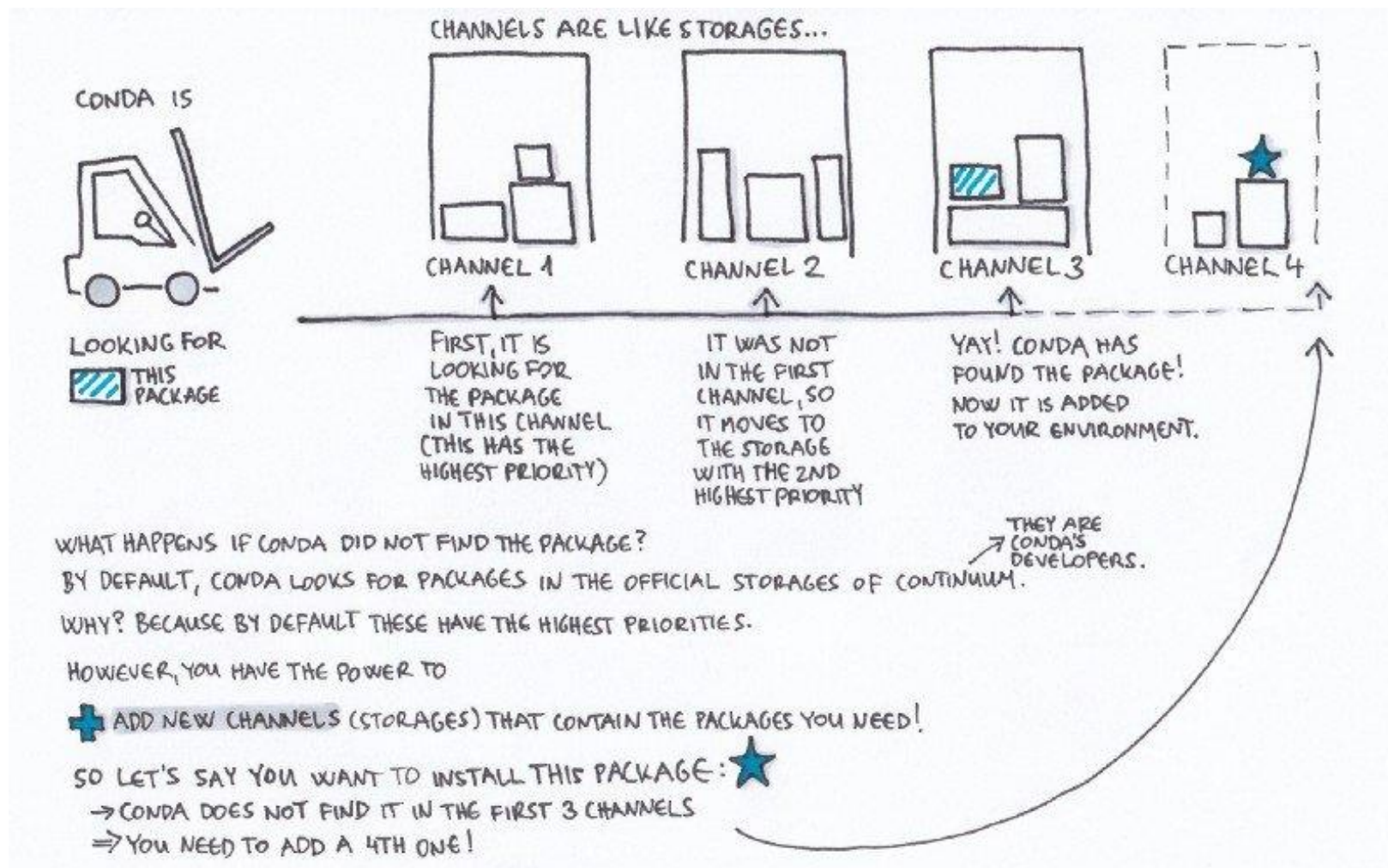
Anaconda: структура



Анаconda: несколько сред



Anaconda: каналы



Cookiecutter Data Science

Cookiecutter Data Science — логичная, достаточно стандартизированная, но гибкая структура проекта для выполнения и совместного использования в области науки о данных.

Репозиторий:

<https://github.com/drivendata/cookiecutter-data-science>

Документация:

<https://drivendata.github.io/cookiecutter-data-science/>

Ocean

Ocean — утилита для создания шаблонов проектов по машинному обучению и анализу данных.

Репозиторий:

<https://github.com/surfstudio/ocean>

Readme:

https://github.com/surfstudio/ocean/blob/master/README_ru.md

Catalyst

Catalyst — проектный фреймворк на базе PyTorch, сосредоточен на воспроизводимости, быстром экспериментировании и повторном использовании кодовой базы.

Репозиторий:

<https://github.com/catalyst-team/catalyst>

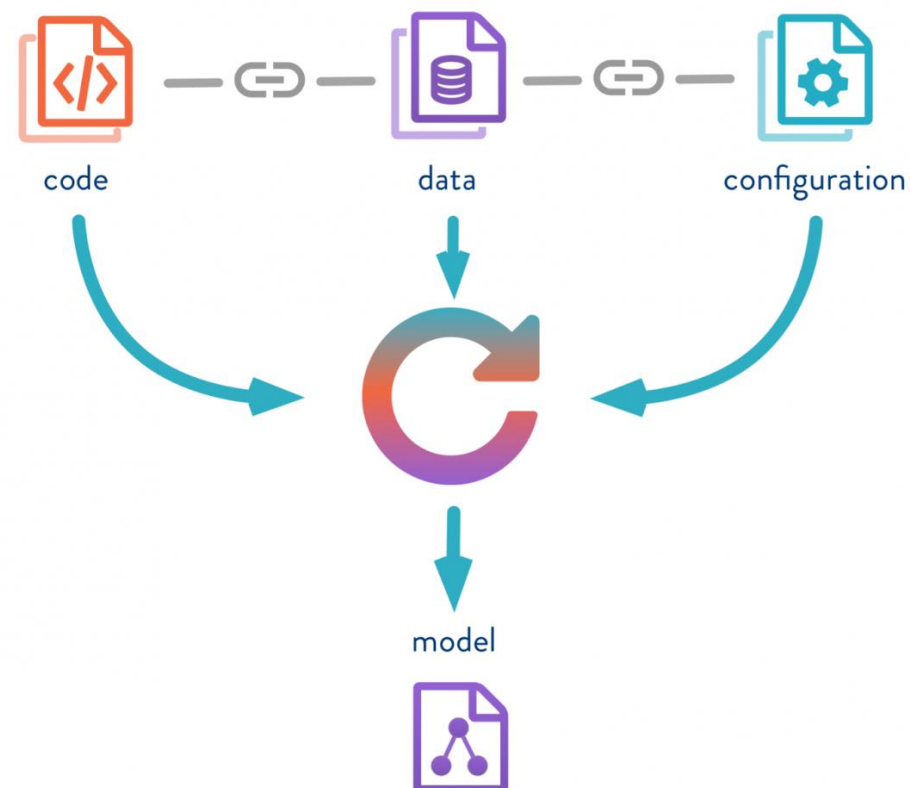
Документация:

<https://catalyst-team.github.io/catalyst/>

Data Version Control: что это?

Data Version Control — это инструмент, который создан для управления версиями моделей и данных в ML-проектах. Он полезен как на этапе экспериментов, так и для развертывания ваших моделей в эксплуатацию.

Документация: <https://dvc.org/doc>



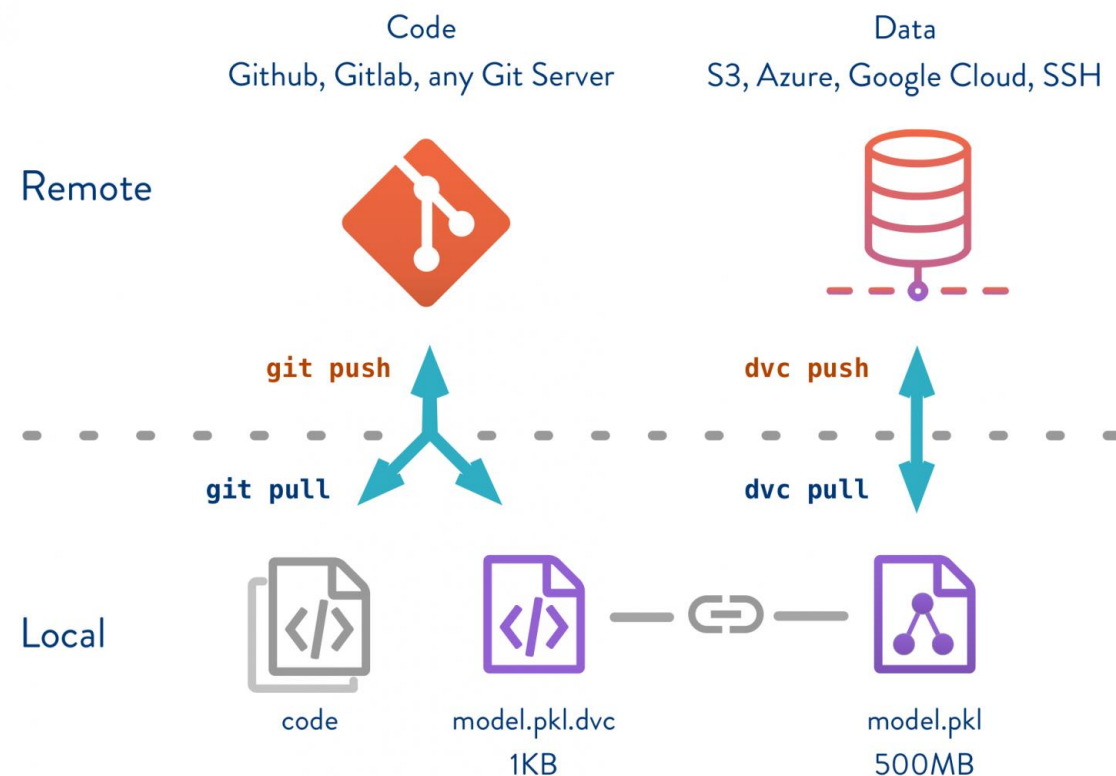
DVC: как работает?

DVC работает совместно с git, используя его инфраструктуру и схожий git синтаксис. Во время работы с проектом DVC создаёт мета файлы, описывающие жизненный цикл и синхронизируемые файлы, версии которых необходимо сохранять в git историю проекта.

После инициализации DVC в локальном репозитории появляется папка `.dvc`, в которой хранятся `cache` и `config`.

`Config` – конфигурация DVC, `cache` – системная папка для версионированных моделей и данных.

DVC: версионирование



run command

```
$ cat data/iris.csv.dvc
```

output

```
md5: 1cff89878034249db68ba6046d5b49a9
wdir: ..
outs:
- md5: 57fce90c81521889c736445f058c4838
  path: data/iris.csv
  cache: true
  metric: false
  persist: false
```

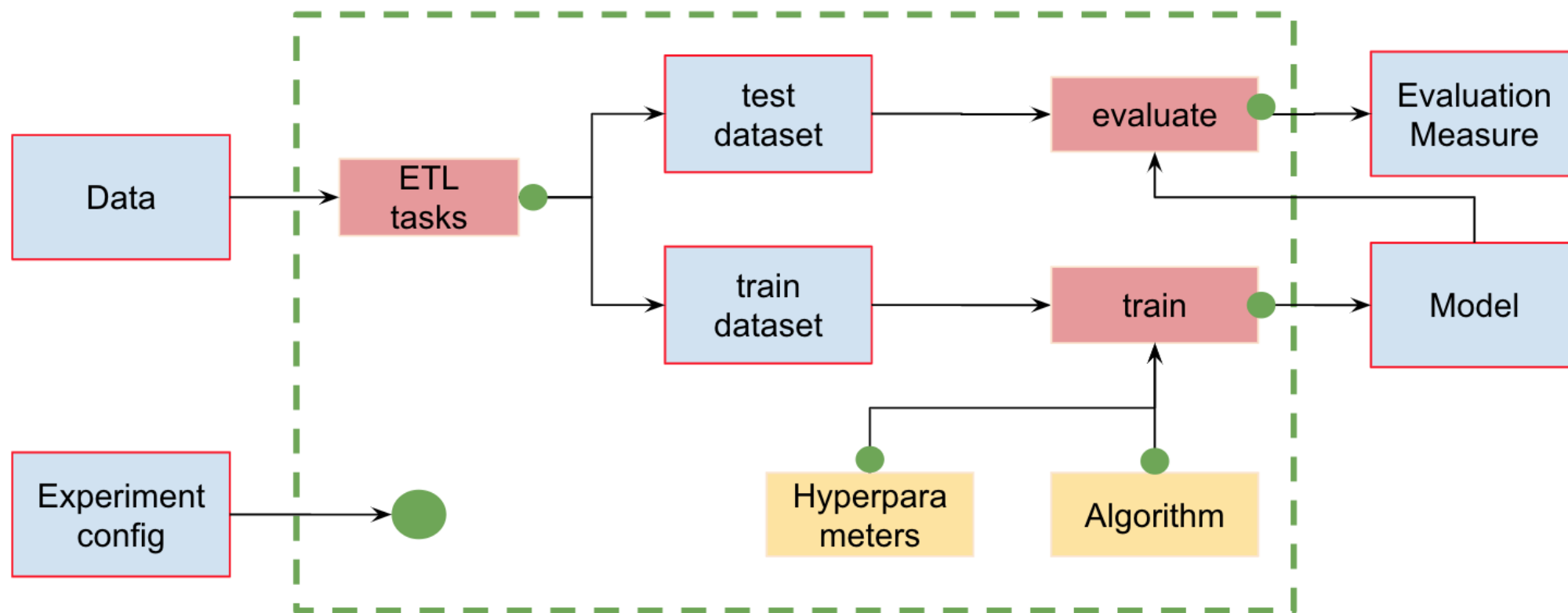
run command

```
$ du -sh .dvc/cache/*/*
```

output

```
4.0K
.dvc/cache/57/fce90c81521889c736445f058c4838
```

DVC: автоматизация ML pipeline



DVC: автоматизация ML pipeline

-d specify dependencies

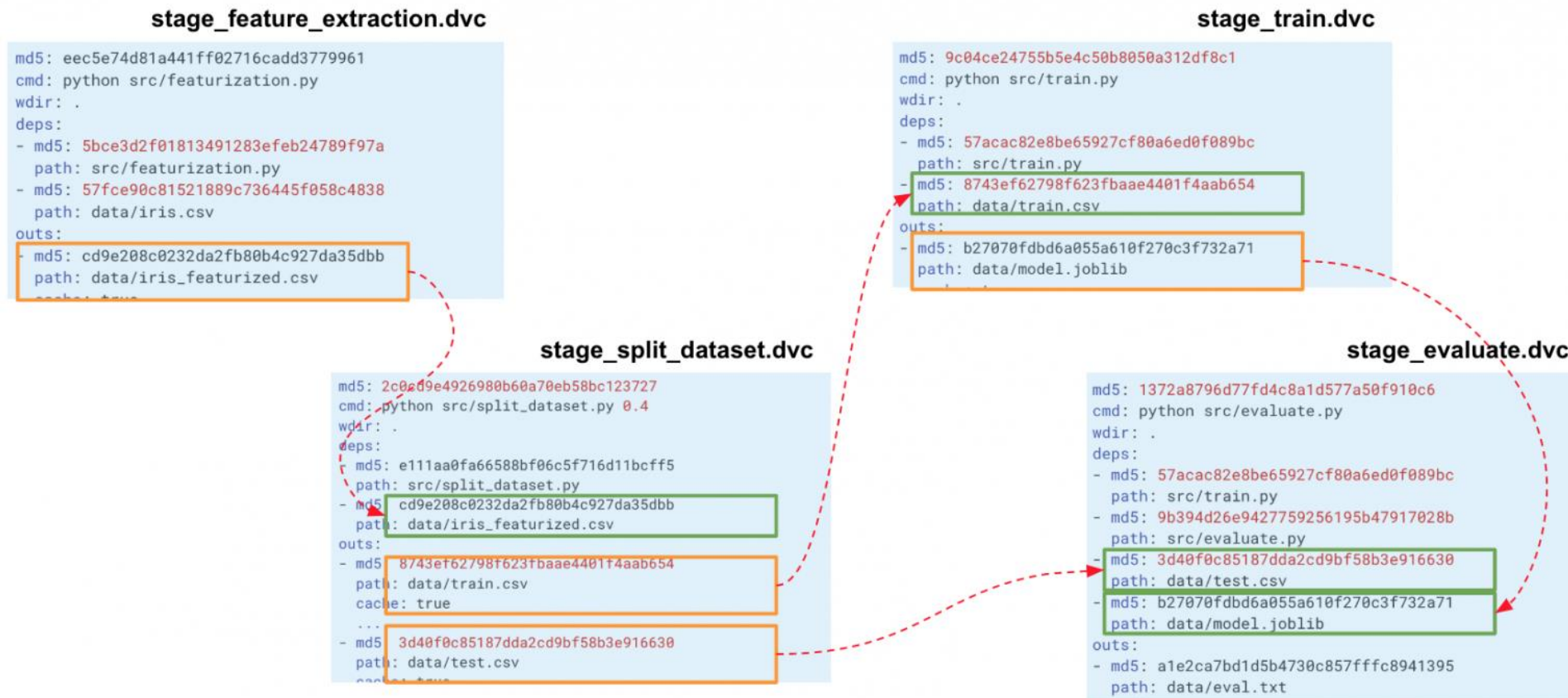
-f specifies name for .dvc file to store stage metadata

```
dvc run -f stage_feature_extraction.dvc \  
  [-d src/featurization.py \  
  -d data/iris.csv \  
  -o data/iris_featurized.csv \  
  python src/featurization.py
```

python command with arguments

-o specifies outputs (data files)

DVC: автоматизация ML pipeline



DVC: отслеживание метрик

run command

```
$ dvc run -f stage_evaluate.dvc \  
-d src/train.py \  
-d src/evaluate.py \  
-d data/test.csv \  
-d data/model.joblib \  
-m data/eval.txt \  
python src/evaluate.py
```

run command

```
$ cat stage_evaluate.dvc
```

output

```
md5: 2c5f02b139310b839b97f2a093b802b9  
cmd: python src/evaluate.py  
wdir: .  
deps:  
- md5: 025acbe1552887fab33f5314d036e907  
  path: src/train.py  
- ...  
outs:  
- md5: 1f7764d988d8d251dc3e9b1c5419f58b  
  path: data/eval.txt  
  cache: true  
  metric: true  
  persist: false
```

DVC: отслеживание метрик

run command

```
$ dvc metrics show
```

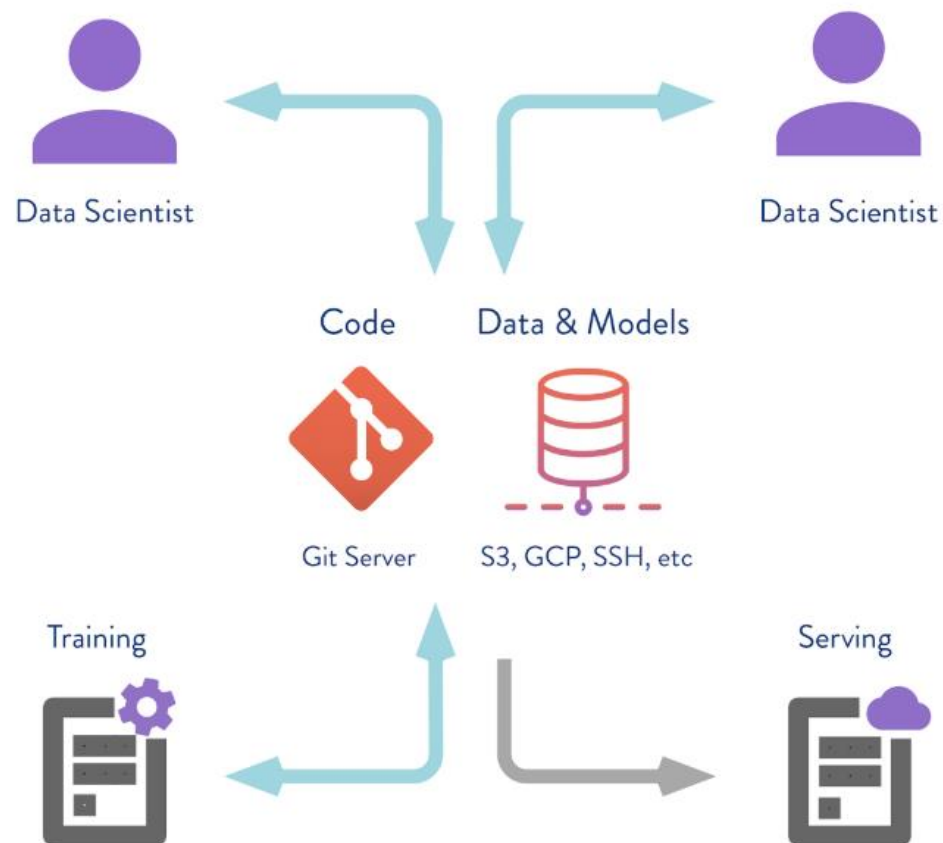
output

```
data/eval.txt:  
{  
  "f1_score": 0.7861833464670345,  
  "confusion_matrix":  
    {  
      "classes":  
        ["setosa", "versicolor", "virginica"],  
      "matrix":  
        [[23, 0, 0],  
         [0, 8, 0],  
         [0, 11, 18]]  
    }  
}
```

DVC: воспроизводимость

```
>dvc repro stage_evaluate.dvc  
  
Stage 'data/iris.csv.dvc' didn't change.  
Stage 'stage_feature_extraction.dvc' didn't change.  
Stage 'stage_split_dataset.dvc' didn't change.  
Stage 'stage_train.dvc' didn't change.  
Stage 'stage_evaluate.dvc' didn't change.  
Pipeline is up to date. Nothing to reproduce.
```


DVC: сохранность данных



run command

\$ dvc push

/tmp/dvc used as a local
'remote storage' in this
example

output

```
Preparing to upload data to '/tmp/dvc'
Preparing to collect status from /tmp/dvc
[#####] 100% Collecting information
[#####] 100% Analysing status.
(1/5): [#####] 100% data/train.csv
(2/5): [#####] 100% data/eval.txtturized.csv
(3/5): [#####] 100% data/iris_featurized.csv
(4/5): [#####] 100% data/test.csv
(5/5): [#####] 100% data/model.joblib
```



Docker: что это?

Docker — популярная технология контейнеризации, появившаяся в 2013 году. Тогда одноименная компания предложила способ виртуализации ОС, при котором код приложения, среда запуска, библиотеки и зависимости упаковываются в единую «капсулу» — контейнер Docker.

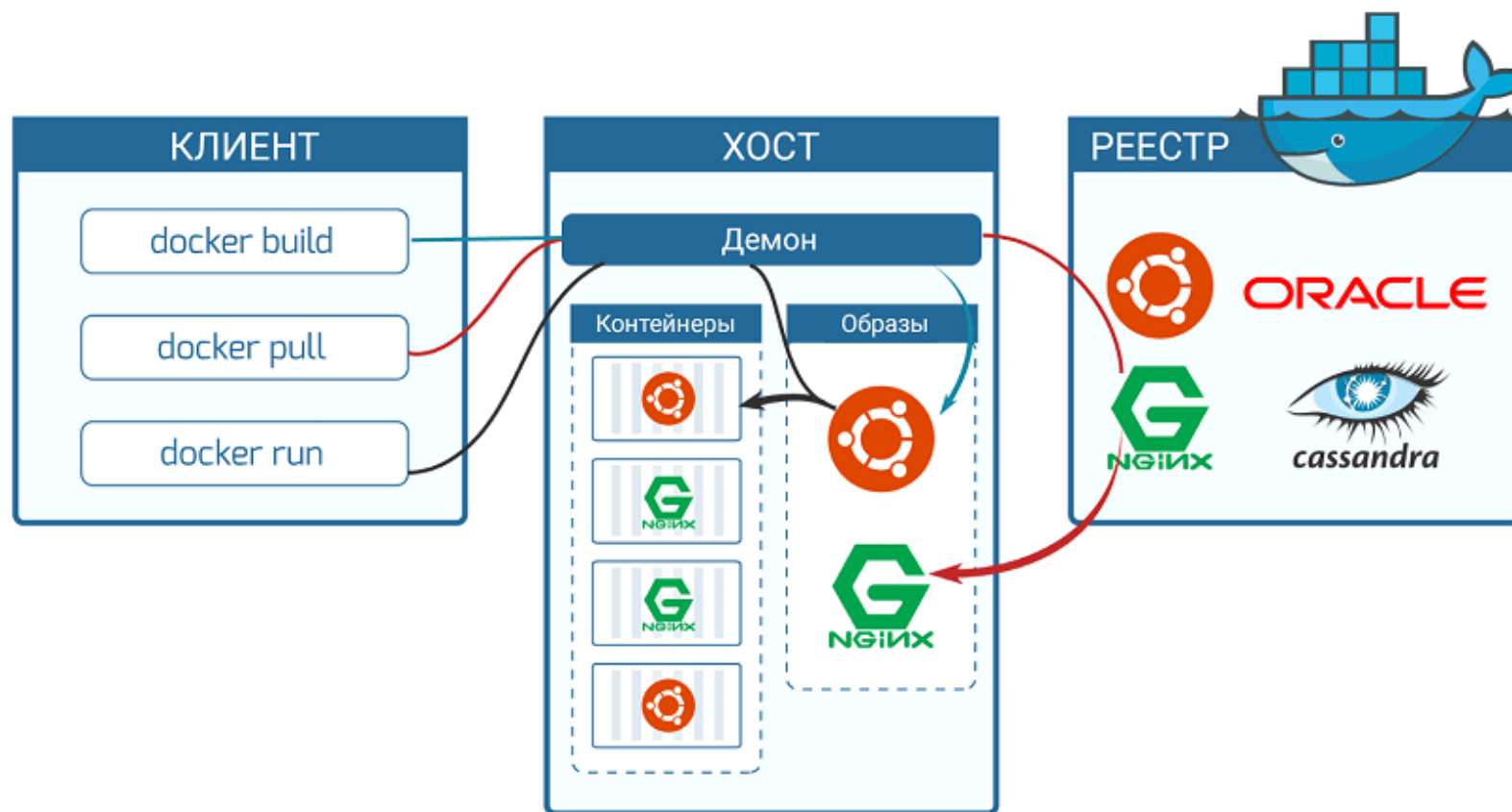
Документация: <https://docs.docker.com/>

Установщик: <https://www.docker.com/products/docker-desktop/>



Docker: компоненты

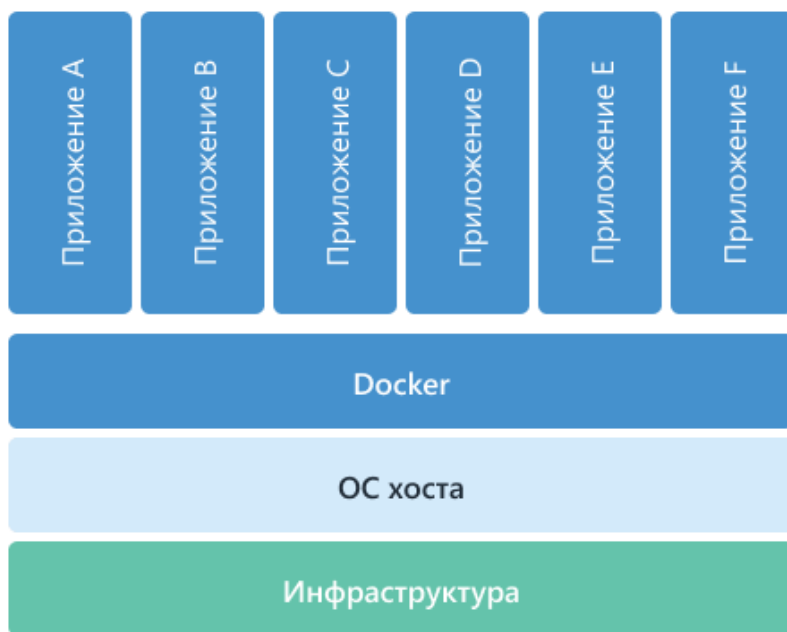
КОМПОНЕНТЫ DOCKER





Docker: контейнеры vs VM

Контейнеры



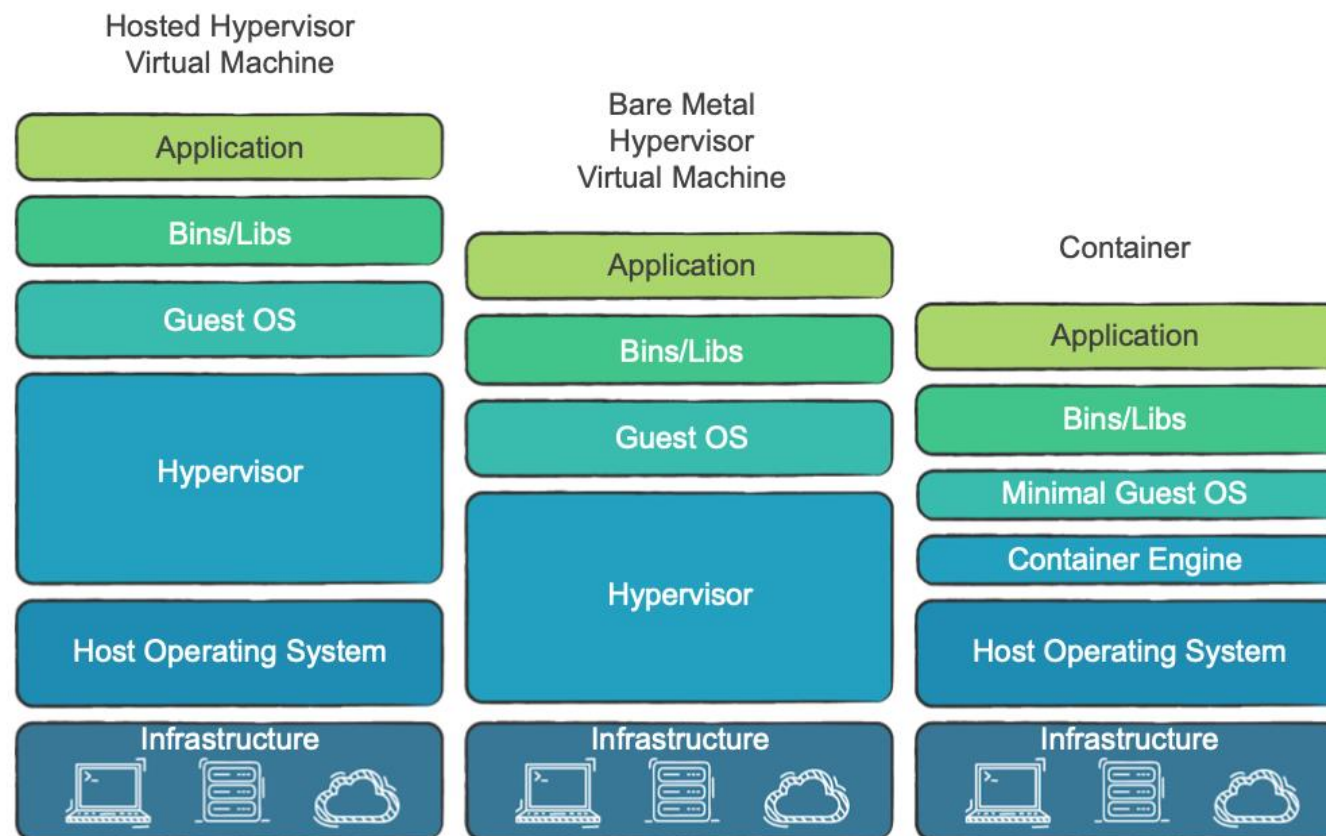
VS

Виртуальные машины



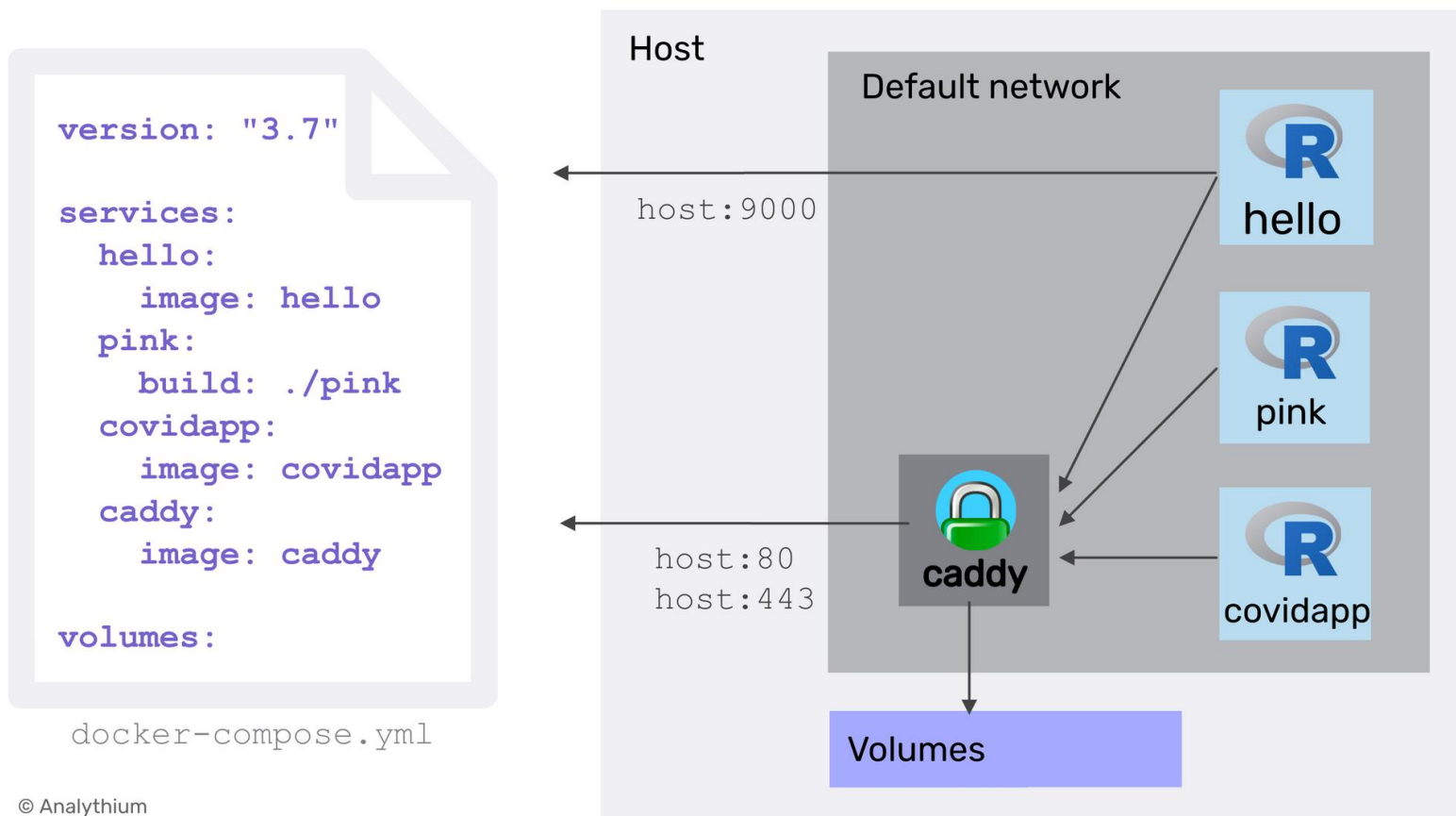


Docker: контейнеры vs VM





Docker: docker compose





Jenkins: что это?

Jenkins – система с открытым исходным кодом, то есть продукт доступен для просмотра, изучения и изменения. Кстати создан на базе Java. Дженкинс позволяет автоматизировать часть процесса разработки программного обеспечения, без участия человека. Данная система предназначена для обеспечения процесса непрерывной интеграции программного обеспечения.

Документация: <https://www.jenkins.io/doc/book/>

Языки: встроенный (declarative pipeline), groovy (scripted pipeline).

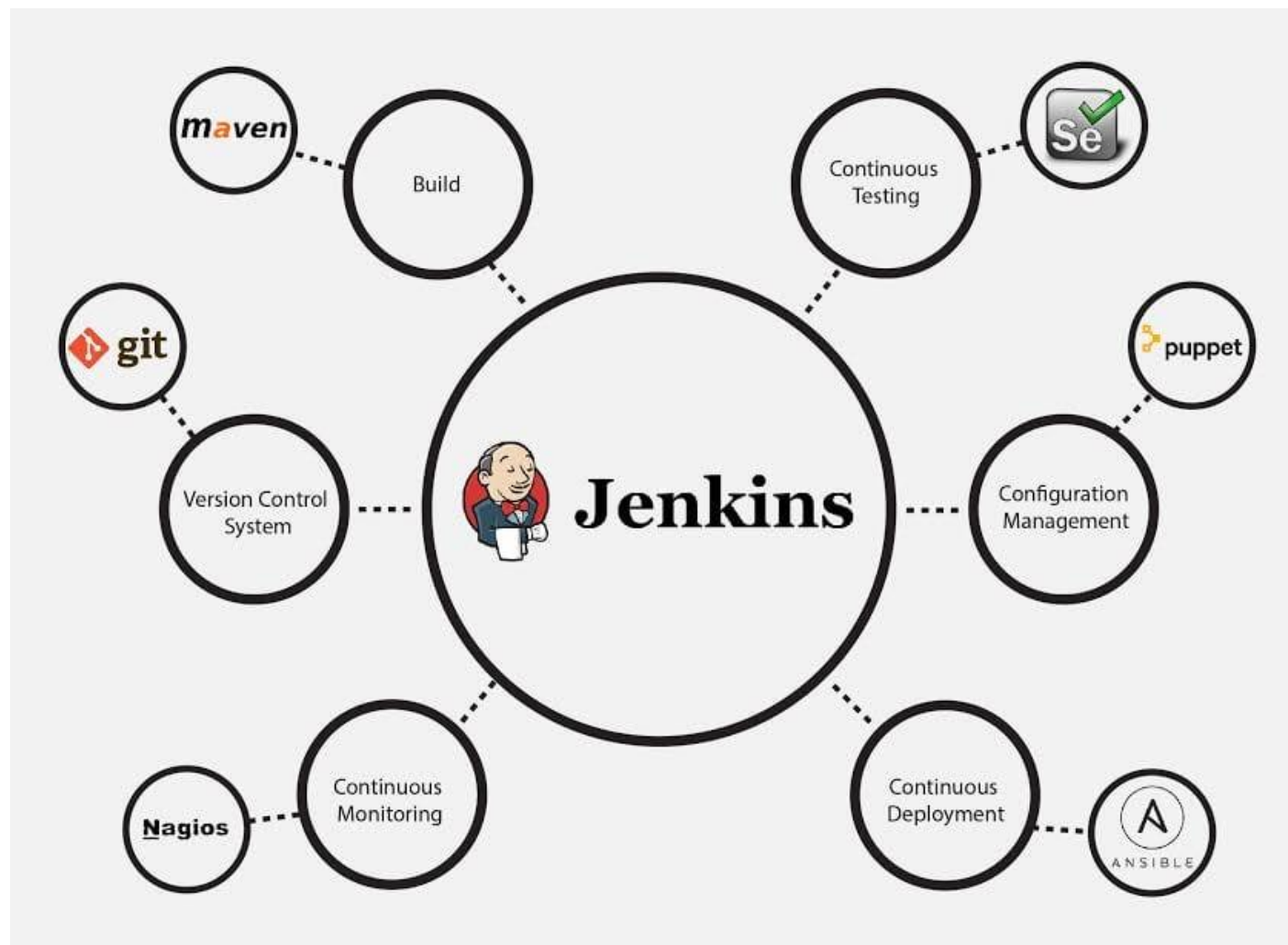


Jenkins: местный DevOps

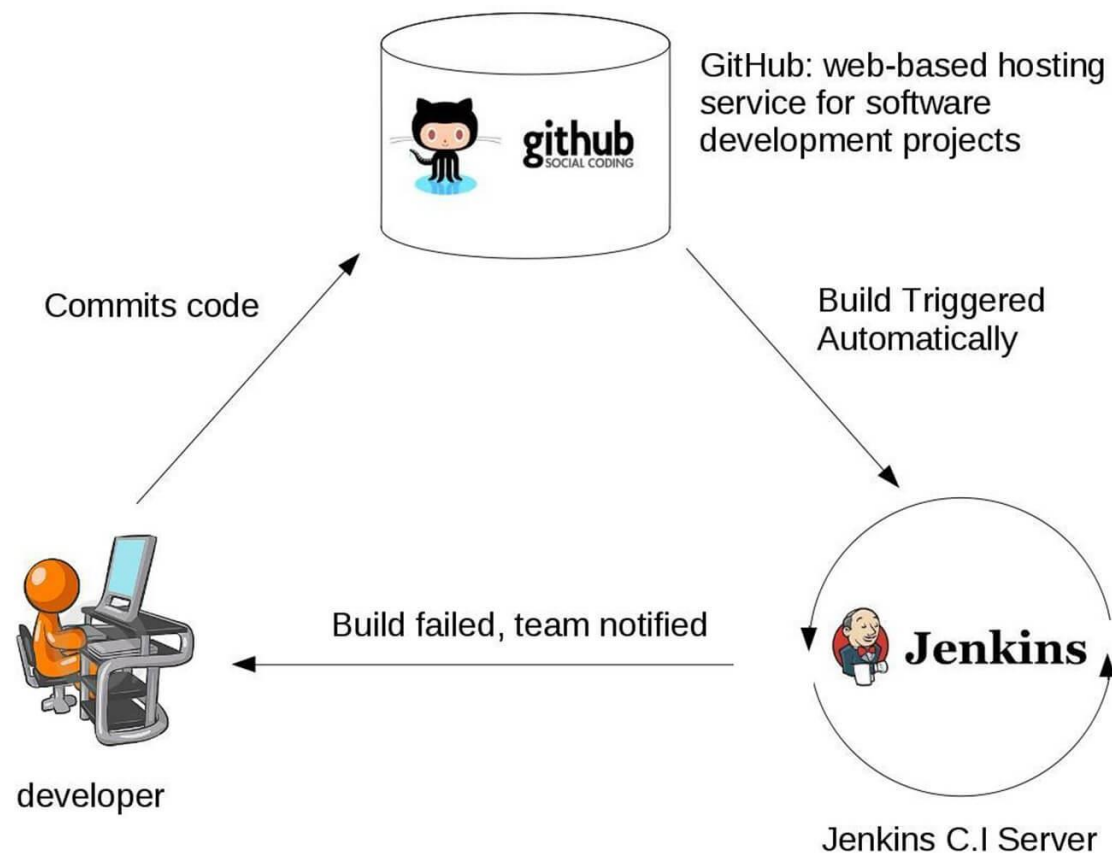
- CI (Continuous Integration, непрерывная интеграция) – начальная стадия «конвейера» по сборке кода и загрузке собранного ПО в среду разработки.
- CDL (Continuous Delivery, непрерывная поставка) – является продолжением CI. В этой практике производится автоматизированное развертывание на тестовую среду продукта и разнообразные тесты над ним.
- CDP (Continuous Deployment, непрерывное развертывание) – поставка результатов работы CI и CD практик в промышленную среду.

Jenkins может реализовать CI/CDL/CDP на практике.

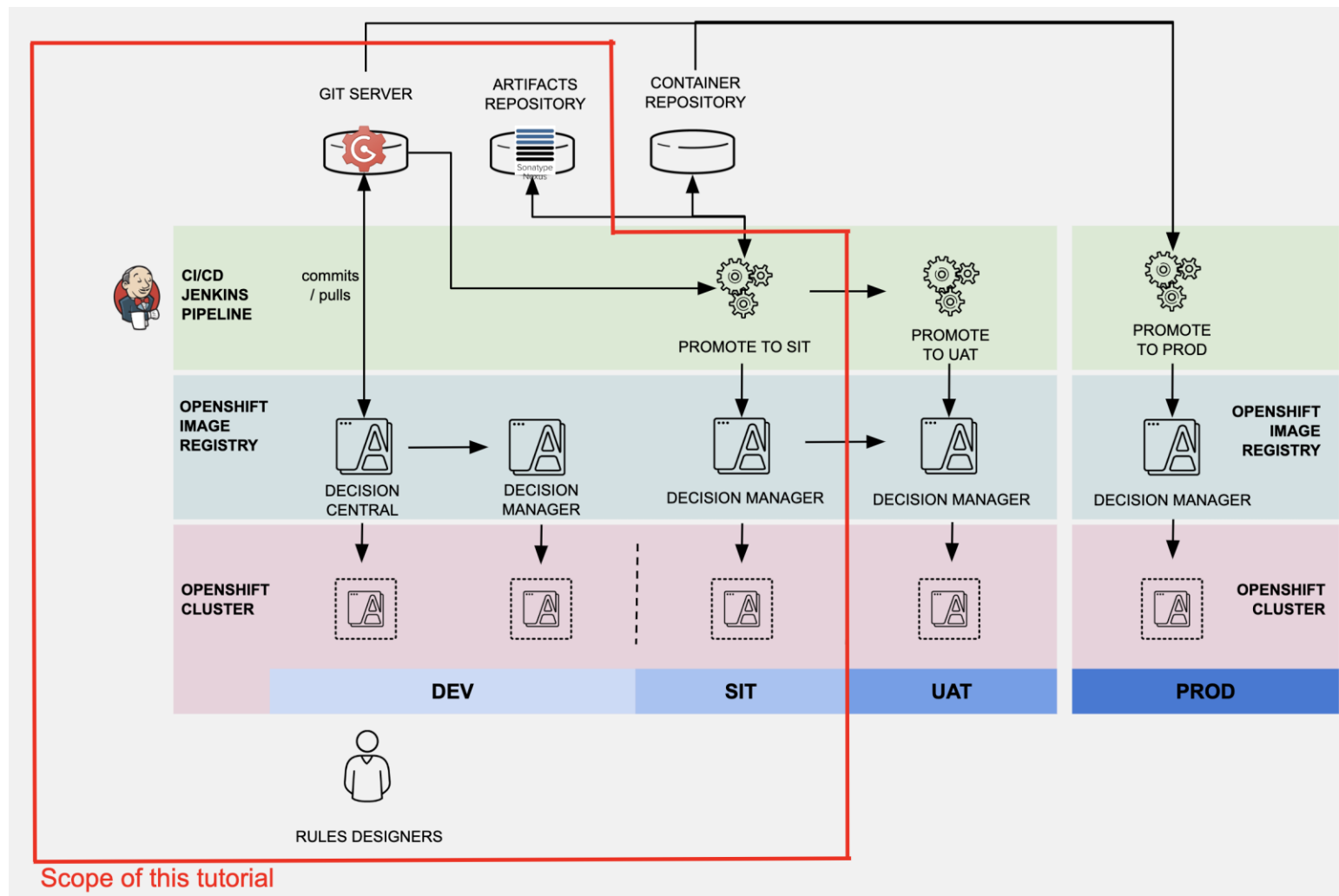
Jenkins: ВОЗМОЖНОСТИ



Jenkins: CI

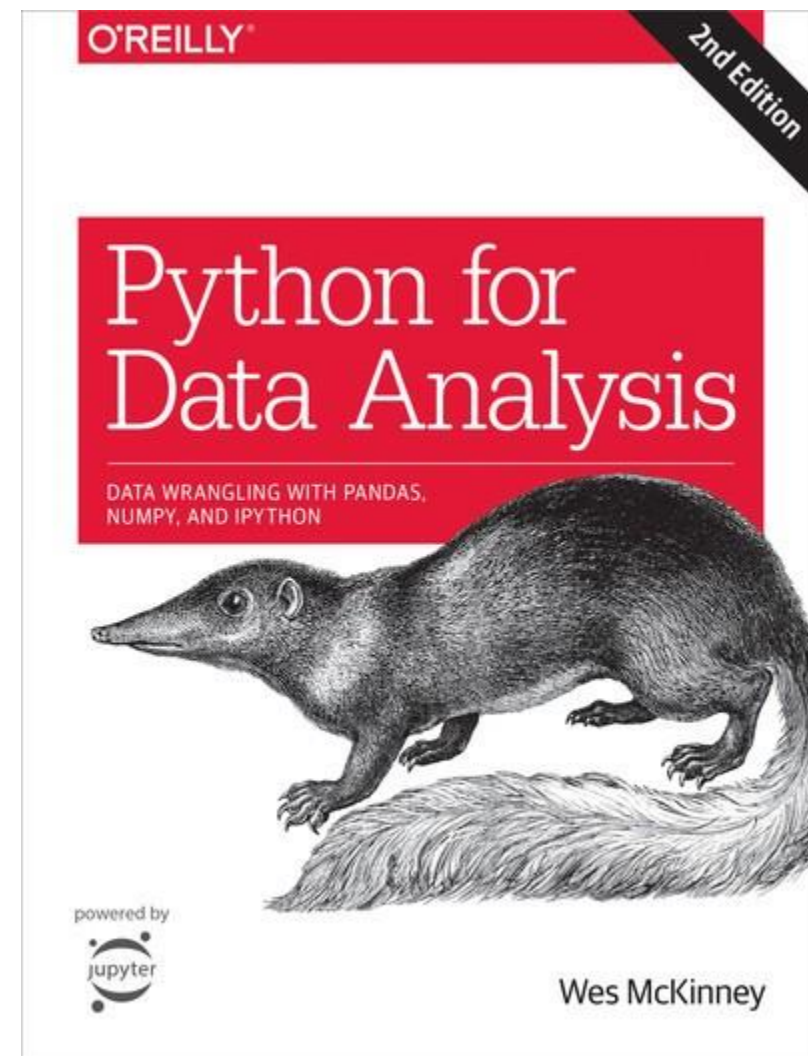


Jenkins: CDL/CDP



Python для анализа данных

- Основные библиотеки Python для исследователя данных: NumPy, Pandas, Matplotlib;
- Навыки манипулирования, очистки и визуализации данных;
- Основы анализа данных временных рядов;
- Базовые навыки работы с Python для MLE.



Вопросы

?