Деплой ML моделей

Миша Трофимов, Александр Гущин, весна 2021

Введение

Приложение

Приложение - это суммарно результат нашего труда.
Обычно, это

- модель в виде бинарного файлика
- код сервиса
- конфиги
- статика, шаблоны
- ...

Если запустить - получим работающий сервис.

Среда

Приложение должно где-то запускаться. Это “где-то” - и есть среда. На самом базовом уровне - это железо, сеть, операционная система, пакеты системы, библиотечки. Среду (англ. environment) разделить на железную и программную.

Деплой

Отлично, есть приложение, оно запускается.
Нужно теперь сделать так, чтобы оно могло обслуживать запросы и приносить какую-то пользу.
Перенос приложения в среду исполнение, развертывание и обеспечение функциональности - это и есть деплой (англ. deploy)

Деплой с программой стороны

Оставим пока вопрос железа в стороне. Будем рассматривать ситацию, когда у нас уже есть доступ к машинам - физическим или виртуальным, не важно.

Пусть у нас есть приложение, мы хотим получить из него рабочий сервис. Рассмотрим несколько типичных вариантов, их эволюцию, плюсы и минусы.

Локальный деплой

- Самый простой вариант - запустить у себя на ноутбуке и пусть висит, если это возможно.
- Среда разработки и выполнения - совпадают
- Легкий доступ, но нельзя закрыть ноутбук
- Малореальный вариант

Деплой на сервер: ручной

- Среды уже начинают расходиться. Разрабатывали на macos, деплоим на ubuntu. А что же должно совпадать, чтобы приложение запустилось?
- Пакеты/зависимости ставим руками
- Используем tmux/screen/daemon для запуска приложения
- Полная прозрачность происходящего
- Ручная настройка окружения часто является источником проблем, тяжело переехать на другую машину

Деплой на сервер: автоматизированный

- Можно составить скрипт действий по развертыванию окружения, подготовке данных, запуску приложения
- Скрипт может быть достаточно сложным и затрагивать несколько машин
- Но главное - все действия выполняются автоматически, а не руками (воспроизводимость)
- Пример такого инструмента автоматизации - Ansible
    - Он за нас ходит на машинки и исполняет команды
- Если на одной машине живут несколько приложений - они исполняются в одной программной среде, возникает взаимозависимость
- Если вдруг нужно откатить неудачную версию - может потребоваться откат окружения, что не всегда легко

Деплой на сервер: контейнеры

- Заворачиваем приложение и все его зависимости в Docker-контейнер
- Получаем полную изолированность програмной среды и возможность легкого отката к предыдущей версии
- Управлять контейнерами можно точно так же через Ansible

Деплой на кластер: k8s

- Кластер - это множество машин
- Иногда что-то (обязательно*) падает - нужно переподнять ноду / перекатить приложение
- Порой приложения нужно деплоить только группой одвременно
- Для окрестрации контейнерами и управлением работы кластера есть инструмент - Kubernetes (k8s, “кубер”)


Среди пассажиров есть врач?
А это мог быть ты
Пап, не начинай, прошу тебя
Чё-то девопсов не так часто ищут, да?
Пап, там человек умирает вообще-то
Ну сходи настрой ему кубер
Перезапустил контейнеры в кардиостимуляторе. Бывает, SLA 100% - это миф.

Выводы

Каждый вариант имеет свои плюсы и минусы.

Локальный деплой - он практически не имеет плюсов, нужен исключительно для разработки.

Ручной деплой на сервер - хорош только в стадии активной разработки, когда от сервиса не требуется никакой надежности, его разрабатываает один человек и важно быстро итерироваться. Временное решение.

Автоматизированный деплой через Ansible - адекватное долгосрочное решение, особенно если нет проблемы с запуском нескольких приложений на одной машине. Долго время был стандартом, до прихода эры контейнеризации.

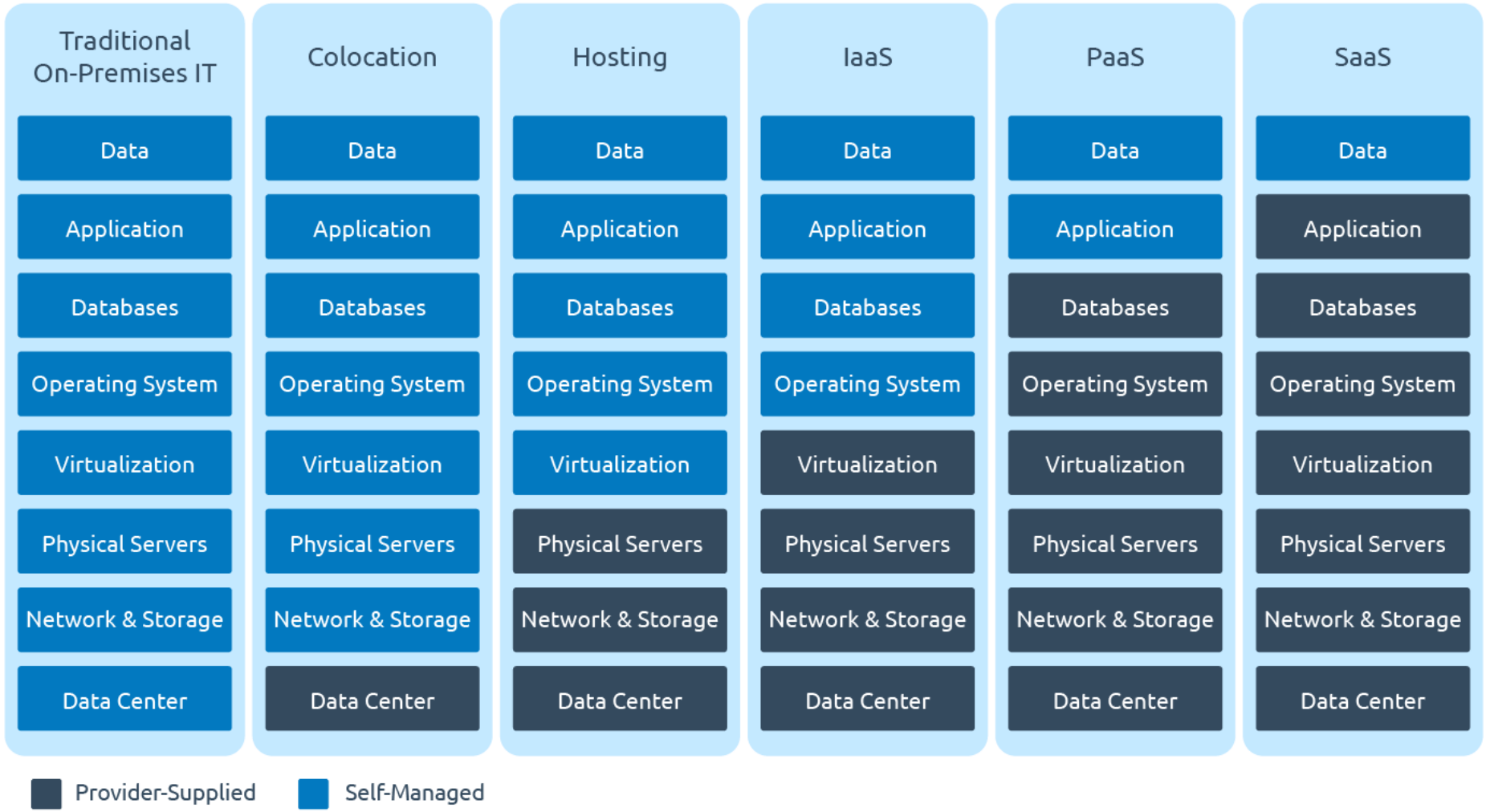
Разворачивание docker-контейнера - де-факто стандарт в настоящее время.
Решает вопросы изоляции, повзоляет легко откатывать версии и “уживаться” разным приложениям на одной машине.
k8s - отличный инструмент, когда нужно управлять большим количеством машин и приложений.

Чем выше изоляция и надежность - тем ниже удобство дебага и скорость итераций.

Деплои с железной бывают

Выше мы рассматривали ситуацию, когда у нас уже есть доступ к вычислительным мощностям. Есть множество способов их получить - от “сырого” железа до полностью готовой облачной инфраструктуры. Это дает гибкость выбора между ценой, удобством и возможностями.

В чем разница - картинка



https://medium.com/@vanshvarshney_/what-is-iaas-vs-saas-vs-paas-and-xaas-whats-the-difference-examples-ceadeee146e6

Traditional On-Premises IT, Collocation

- Плюсы
  - Управляем всей инфраструктурой (или большей частью)
  - Можем оптимизировать стоимость, если наши запросы очень велики (нам требуются сотни машин, купить GPU для исследований может быть дешевле, чем арендовать)
- Минусы
  - Требуется отдельный штат специалистов, занимающихся поддержкой инфраструктуры (сеть, питание, вентиляция, обслуживание помещений)
  - Слишком сложно и дорого для небольших компаний.

Hosting - аренда физических серверов

- Плюсы
  - Полноценная удаленная машина
  - Обслуживанием железа занимается хостер
  - Отсутствие виртуализации
    - необходимость для некоторого софта
    - нулевой оверхед
- Минусы
  - Все ещё сложно поддерживать (с программной стороны) - требуется DevOps
  - При небольшой нагрузке возникает соблазн поддерживать самому, что приводит к росту технического долга.

Infrastructure-as-a-Service (IaaS) - AWS EC2, GCE

Очень близко к Hosting

- Плюсы
  - Виртуализированные машинки
  - Можно быстро “накатывать” требуемые образы
  - Более гибко, чем Hosting, - часть машин можно временно отключить и сэкономить
  - Можно легко масштабировать мощности
  - Часто дефолтный варинат для компаний
- Минусы
  - Машины все еще остаются независимым машинами
  - Требует выстраивания процессов деплоя и мониторинга

PS: виртуализацию полезно представлять как некоторую абстракцию на реальным железом

Platform-as-a-Service (PaaS)

- Плюсы
  - Не нужно думать о инфраструктуре, обычно задача лишь в том, чтобы поместить приложение в контейнер,
  - Легче масштабировать, адаптируясь к нагрузке
- Минусы
  - Обычно дороже, чем аренда серверов

Serverless (AWS Lambda, Google Cloud Function)

- Плюсы
  - Не нужно думать о инфраструктуре,
  - На лету адаптируются к меняющейся нагрузке (autoscaling)
- Минусы
  - Нужно подстраиваться под ограничения на среду исполнения,
  - Неудобно делать inference на GPU - прогоняются по одному примеру вместо батча.
  - Оплата не за время использования вычислительныъ мощностей, а за количество вызовов
  - Становится очень дорого на масштабе

Саммари

Варианты есть разные, у всех есть плюсы и минусы.

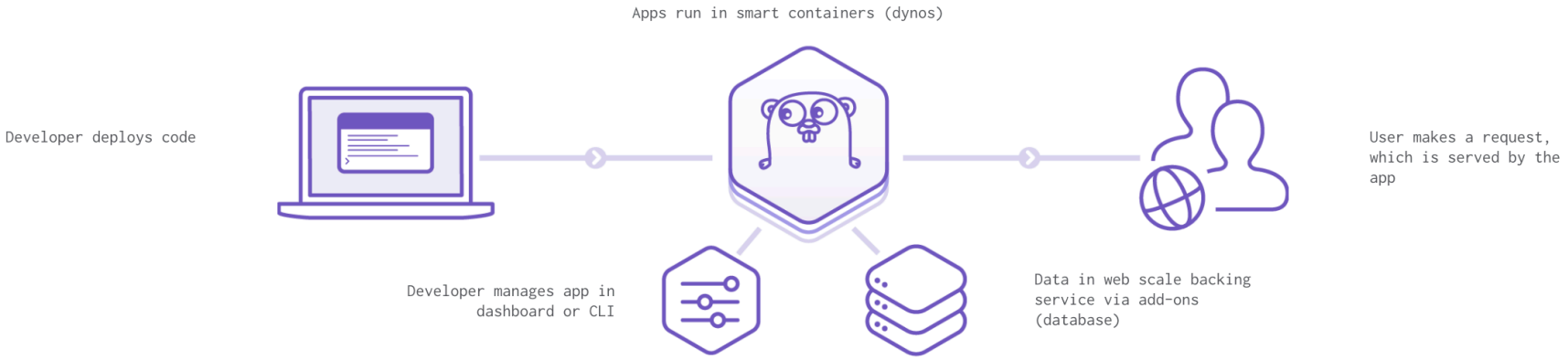
Type	Solution	Pros	Cons	
Personal Server	Local PC	- Freedom in configuration.	- Most deployment efforts.	- Need to run API all the time on your PC.
	Server Machine			- Cost to maintain server (overkill for a small app).
PaaS	Heroku	- Less deployment efforts becuase WSGI server functionality is hosted by Heroku.	- Less integration with other app components (large dataset stored outside of the project etc.) Possibly insufficient for large-size apps.	
Serverless	AWS Lamba	- Least deployment efforts due to managed WSGI server functionality and managed web app framework by serverless service.	- Difficult to include external Python libraries.	
	Cloud Functions		- No support for mulitple HTTP endpoints. Need to create multiple cloud functions to host multiple URLs.	

https://towardsdatascience.com/creating-an-iphone-app-like-with-only-your-data-science-skills-one-tap-life-logger-ac691698d3b3

Мы будет импользовать для практики Heroku - он удобный, простой и подходит *для наших нужд*.

Heroku поближе

A platform for simple, reliable and concurrent services



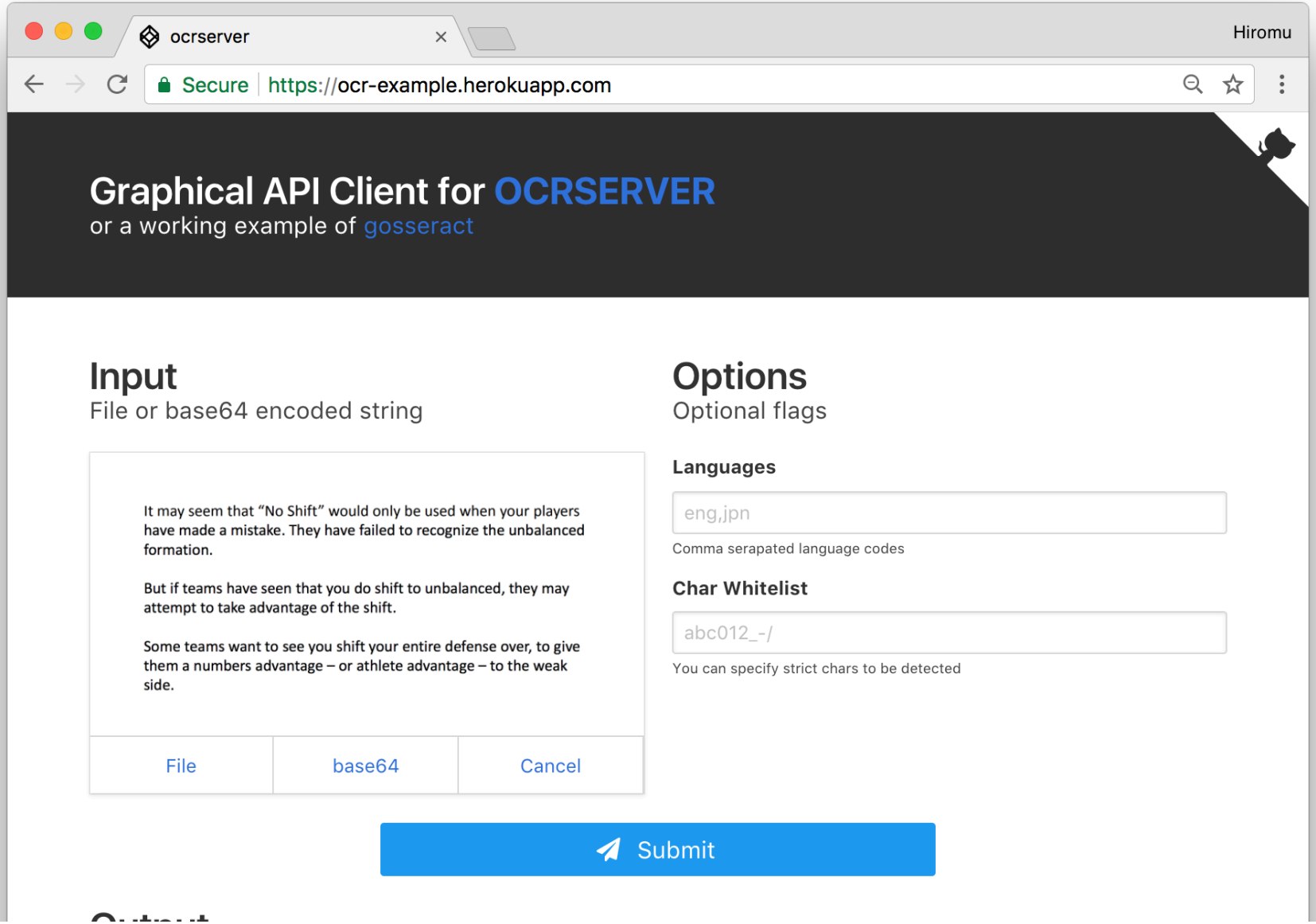
https://www.heroku.com/python

Чем это удобно нам

- Хороший бесплатный план
- CLI-интерфейс
- Container-based deployments
- Интеграция с git
- Инструменты для дебага

Примеры приложений, задеплоенных там

https://github.com/otiai10/ocrserver



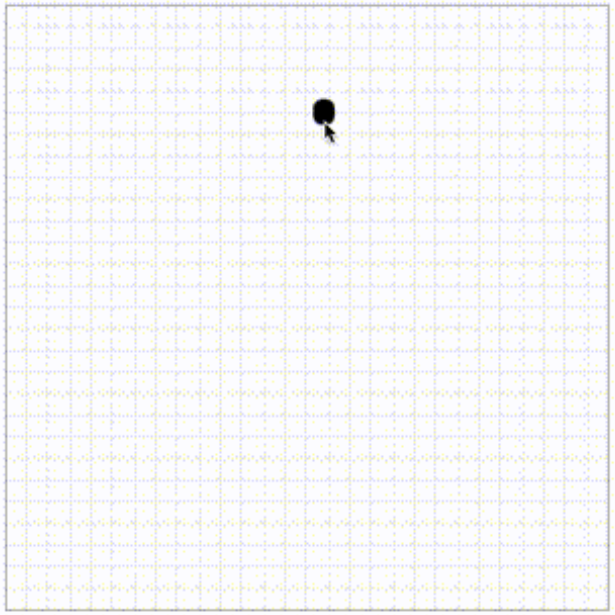
Output

```
200 OK
-----
{
        "result": "It may seem that \"No Shift\" would only be used when your players\nhave made a mistake. They
        "version": "0.2.0"
}
```

---

https://github.com/sugyan/tensorflow-mnist

draw a digit here!



clear

input:

output:

	simple	convolutional
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

Задание

Заданием будет подготовить и задеплоить на heroku приложение про пароли.

Дальше я просто буду пересказывать официальные туториалы:
https://devcenter.heroku.com/articles/getting-started-with-python
https://devcenter.heroku.com/categories/deploying-with-docker

---

Как работает heroku

Внутри, heroku запускает собранные контейнеры и дает к ним доступ.

Запущенный контейнер называет dyno. В зависимости от плана, есть разные ограничения на ресурсы у dyno, время работы, “засыпание” и др.

Есть разные способы доставить свое приложение в heroku. В конечном счете, должен получиться docker-образ.
Наши опции:

- загрузить уже готовый образ (сборку мы осуществляем сами)
- указать явно, как образ должен быть собран (и сборка будет на стороне heroku)
- оформить код и дать вводные, по которым heroku сам соберет стандартный образ

Разберем последние два, перейдя к практике

Деплоим на Heroku из Git

Наконец само задание. В лекции общий рассказ, потом закрываем лекцию и собственно интерактив

Обзор инструкции с пояснениями

Чтобы вырисовался общий план и стало понятно, что зачем нужно, как это связано с Git, обратить внимание, где собирается докер-образ, как запускается процесс, что такое Dyno и так далее

Пошагово деплоим наш репозиторий

Просто шаг за шагом приводим инструкции, поясняем все что не пояснили в предыдущем пункте

Смотрим в интерфейс после деплоя и анализируем что там есть

Здесь у меня уже кое-что задеплоено заранее, и тут просто скриншоты, что там можно увидеть

Примеры аддонов - что можно добавить к текущему решению

Парочка аддонов, чтобы можно было поиграться дома

Практика (конец лекции, начало семинара)

PS Note: Что нужно рассказать:

- heroku run bash
- git push heroku master
- pipenv run heroku local:start
- pipenv run python app.py
- Procfile
- heroku.yml and multicontainer payloads
- requirements.txt vs Pipfile.lock
- –>