Model pipelines

Sprint 4 - Week 8

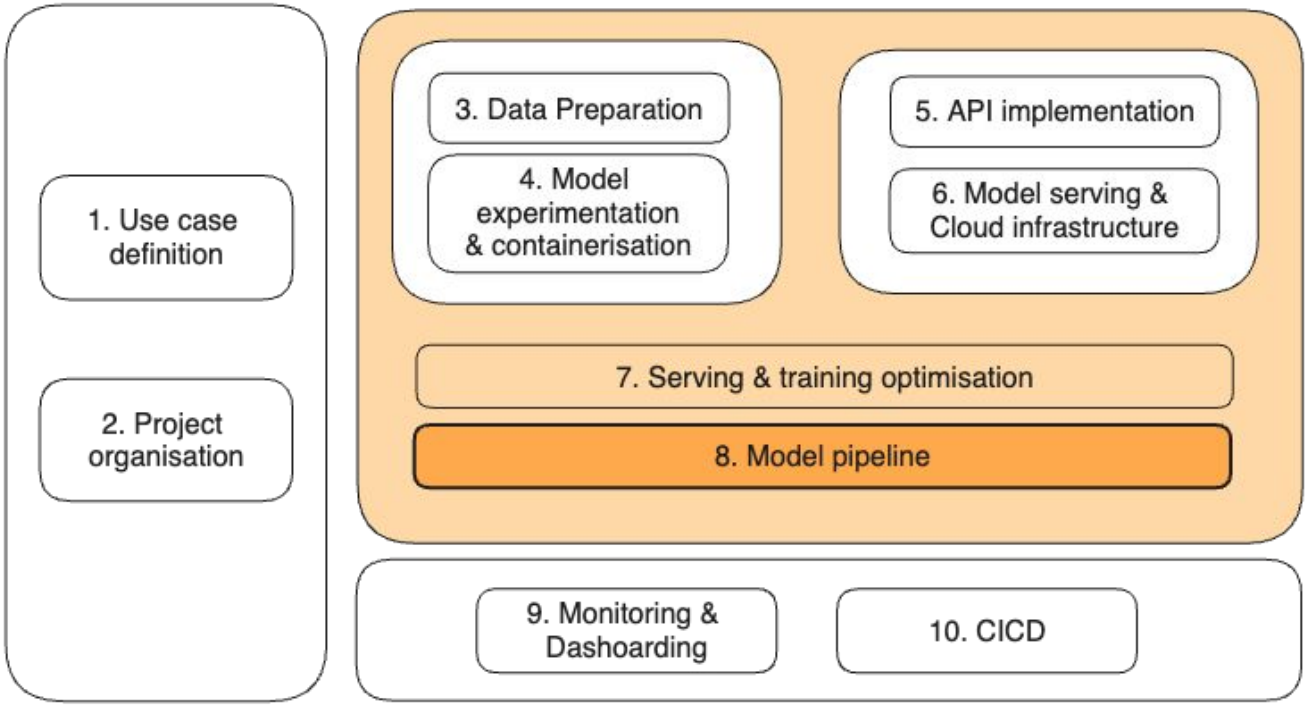
INFO 9023 - Machine Learning Systems Design

2024 H1

Thomas Vrancken (t.vrancken@uliege.be)
Matthias Pirlet (matthias.pirlet@uliege.be)

Status on our overall course roadmap

Status on our overall course roadmap



Agenda

What will we talk about today

Lecture (45min)

1. Microservice architecture
    a. What is a microservice
    b. Abstractions & Separation of concerns
    c. The anatomy of a microservice
    d. Monoliths aren't for dinosaurs

Lab (45 min)

2. Docker compose

Lecture (30min)

3. ML model pipeline
4. ML platforms & orchestrators

Project update

⚠️ Decision to **combine Milestone 2 and 3 !** ⚠️

There will be only one presentation, taking place in our last **course** (13/05/2024) !

How? (Info is also on github)

- Same content as MS 3: Present your overall project in 10 min presentation + 5 min QA
- It will be in classroom ! You're very welcomed to attend other project presentations
- Send codes (link to pull request) by email **before the presentation**
- Agenda for the presentations is in this sheet (let us know if it does not work for you)
- Topics from the last "bonus" lecture are going to be spread over other lectures

Why?

👫 Nicer to let everyone attend all presentations

⚙️ Simpler organisation as there is already a class block in your agenda, don't take more of your time

📃 Avoid having to repeat topics between

Microservice architecture

Credits where credits are due

Robbe Sneyders
Principal Engineer @ ML6

Ruwan Lambrichts
Senior ML Engineer @ ML6

Topic of this section

Defining a general software architecture for microservices in your project/organisation.

Topic of this section

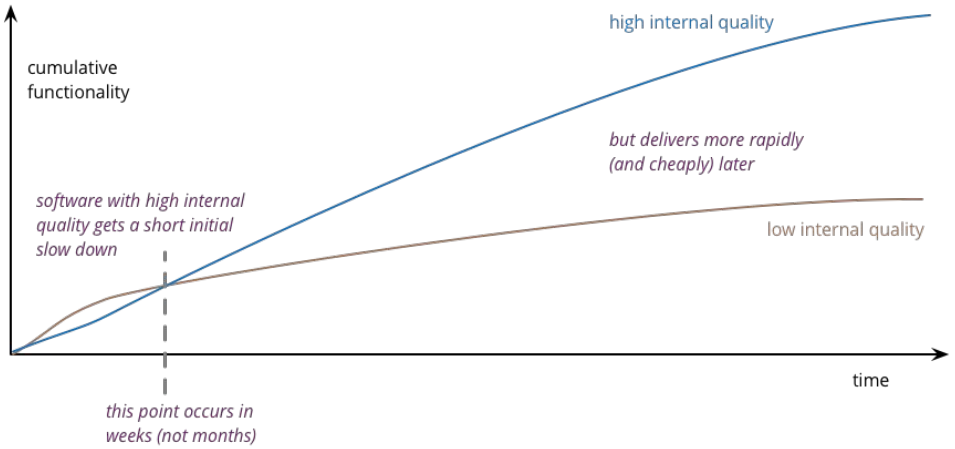
Defining a general software architecture for microservices in your project/organisation.

- The shared understanding that the expert developers have of the system design.
- The decisions you wish you could get right early in the project.

Why does software architecture matter?

Software architecture enhances the **quality** of applications developed by your team, which is something stakeholders don't immediately perceive (and which might seem less important)

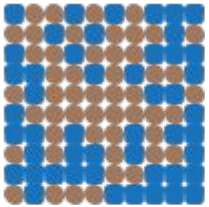
However, poor architecture is a major contributor to **cruft** - leading to code that is much harder to modify, causing features to arrive more slowly and with more defects (which the client cares about a lot)



Cruft causes features to arrive more slow and with more defects

High quality software architecture leads to faster delivery of new features (with less chance of making a mistake), because there is less cruft to get in the way.

If we compare one system with a lot of cruft...

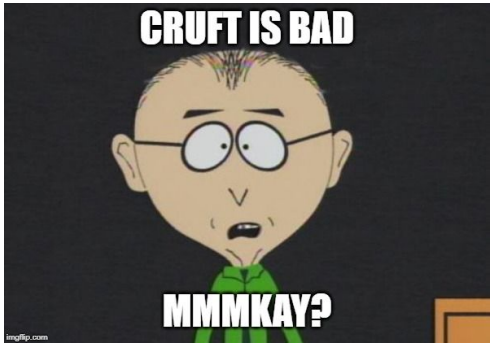


the cruft means new features take longer to build

this extra time and effort is the cost of the cruft, paid with each new feature

...to an equivalent one without

free of cruft, features can be added more quickly



Ideal state: Your team uses/builds boilerplates.

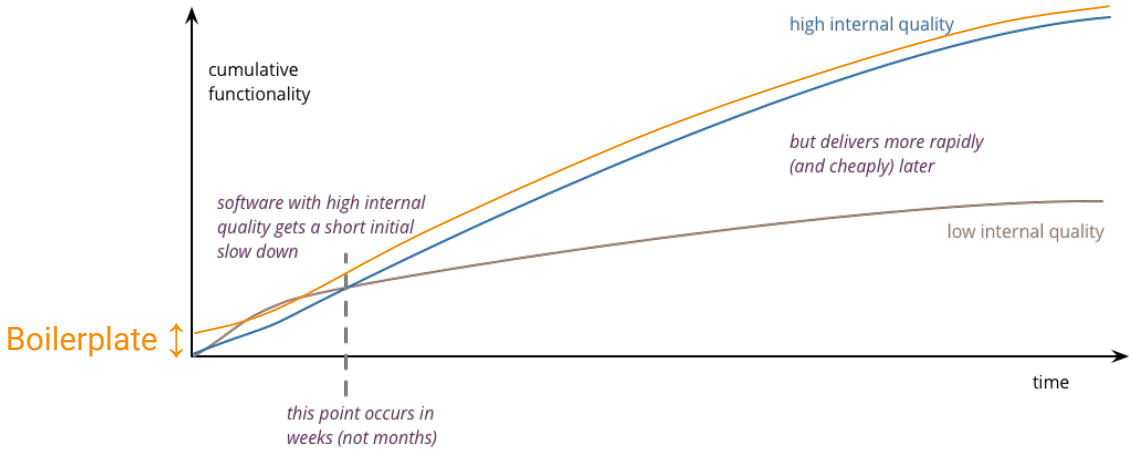
Creating a shared understanding of the microservice design .

Captured in the **microservice boilerplate**, so every project can start from the tried and tested decisions already made on previous projects.

Ideal state: Your team uses/builds boilerplates.

Take ‘correct’ decisions based on combined knowledge of developers and projects, implemented by the microservice boilerplate

- Saves time at start of project
- Projects don’t run into problems down the line, can evolve fast, and stay maintainable
- Consistent design across projects facilitates understanding of other projects



What is a microservice?

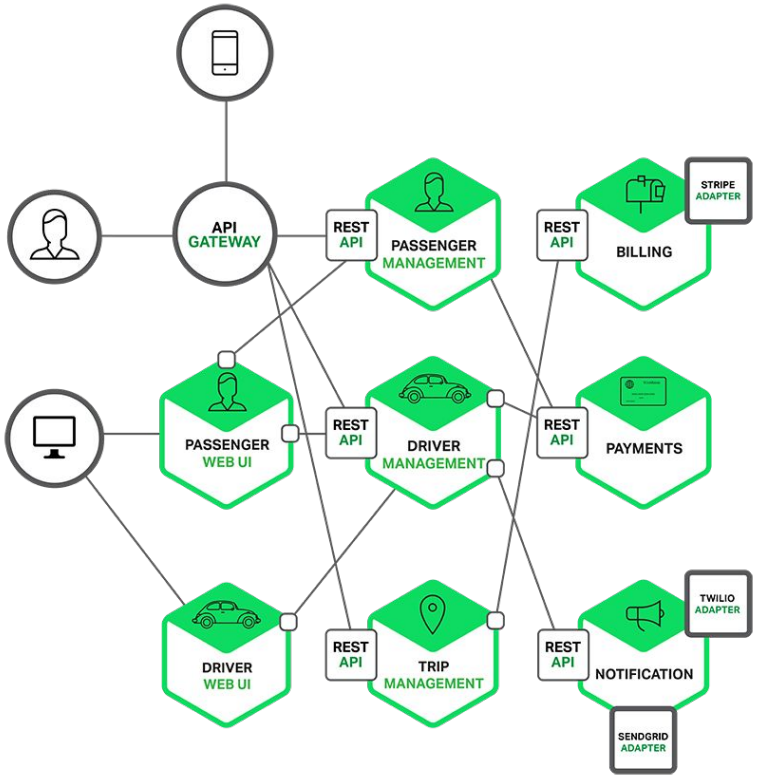
Some context

What is a microservice?

*Using a microservices approach, software is composed of **small services** that communicate over **well-defined APIs** that can be **deployed independently**. These services are owned by small autonomous teams.*

What is a microservice?

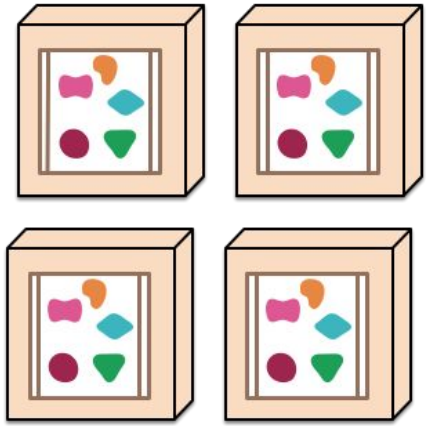
- Applying single responsibility principle at the architectural level
- As small as possible, but as big as necessary
  - “No bigger than your head”
- Advantages compared to monolithic architectures:
  - Independently deployable
  - Language, platform and technology independency between components
  - Distinct axes of scalability
  - Provides firm module boundary
  - Increased architectural flexibility
- Often integrated using REST over HTTP
  - Business domain concepts are modelled as resources



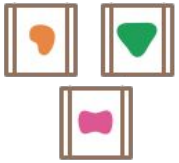
Monolith vs microservice

A monolithic application puts all its functionality into a single process...

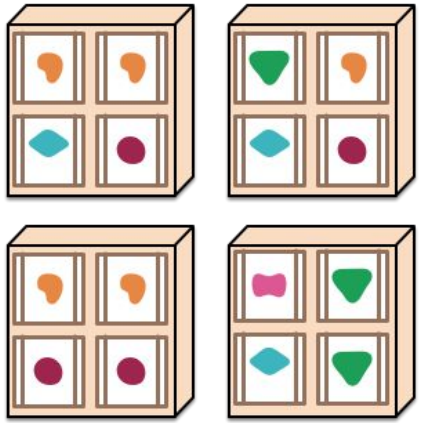
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.

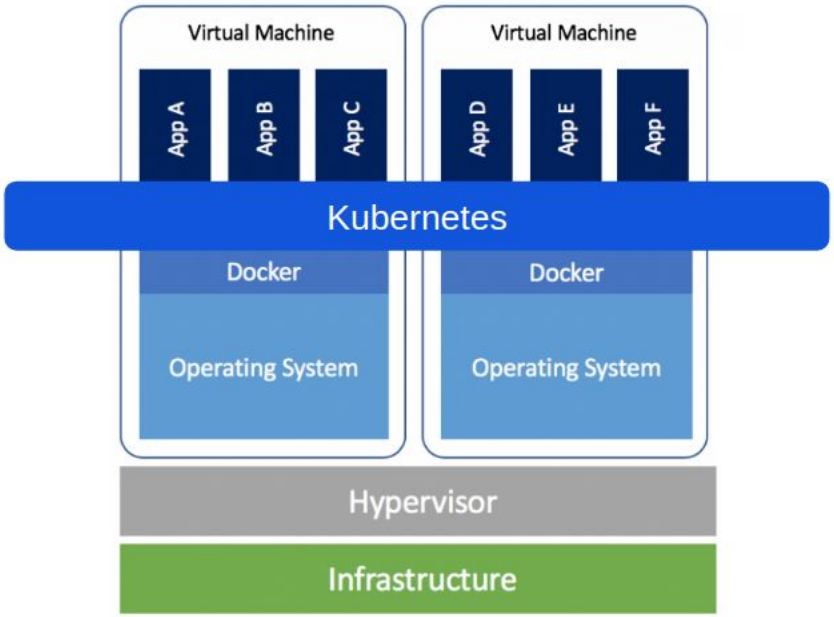


Abstractions
&
Separation of concerns

Abstraction

- Abstraction is the act of **representing essential features without including the background details** or explanations.
- Reduce complexity and allow efficient design and implementation of complex software systems.

Abstractions are everywhere



Python

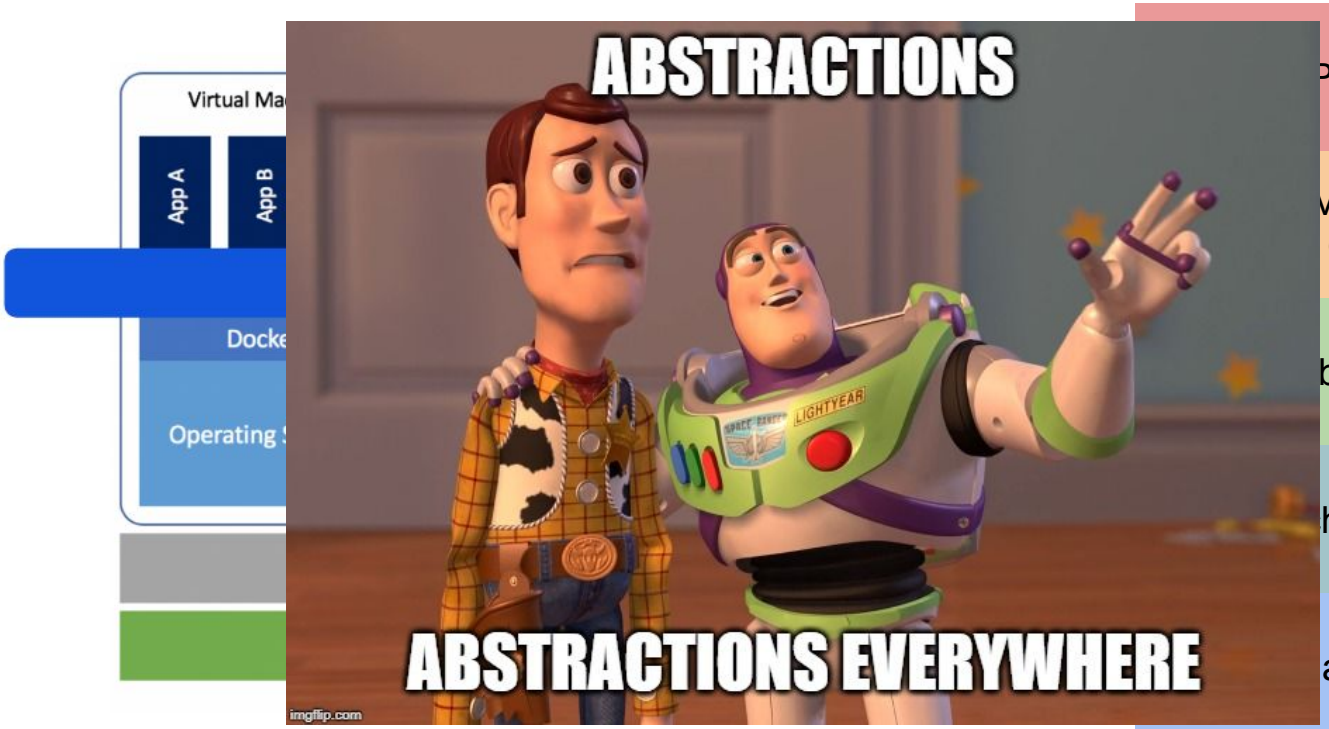
Low-level language
C, ...

Assembly language

Machine code

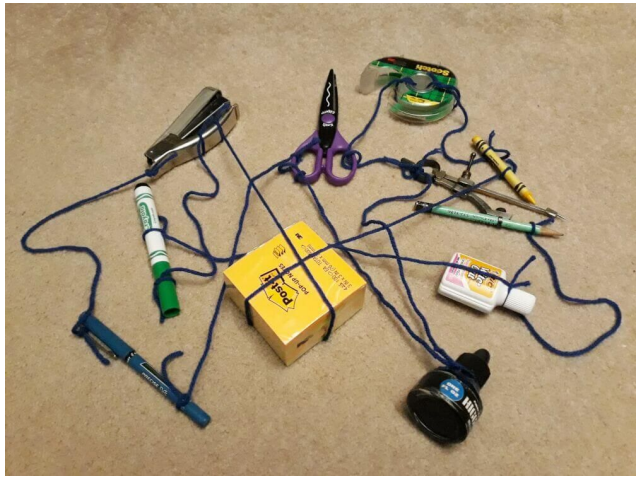
Binary code

Abstractions are everywhere



Separation of concerns

- Separate code into sections that each address a separate concern
- Sections can evolve independently of other components
- Separation of concerns is a form of abstraction



Images: https://pusher.com/tutorials/clean-architecture-introduction

Interfaces

- Describes the public actions and attributes that a type of component supports and implements.
- Defines the boundary across which two components can communicate.
- Components with an identical interface are interchangable.

Interfaces

- In traditional OOP languages, an Interface defines a contract which is explicitly implemented.
- In Python, you don't have to explicitly declare an interface.

```
interface Pet {
    public Owner owner();
    public void pat();
}

class Dog implements Pet {

    public Owner owner() {
        return this.get_owner();
    }

    public void pat() {
        return love;
    }
}
```

```
class Dog:

      @property
      def owner(self):
            return self._get_owner()

      def pat(self):
            return love


dog = Dog()
try:
      dog.owner()
      dog.pat()
      print("dog is a pet")
except:
      print("dog is not a pet")
```

LIÈGE université

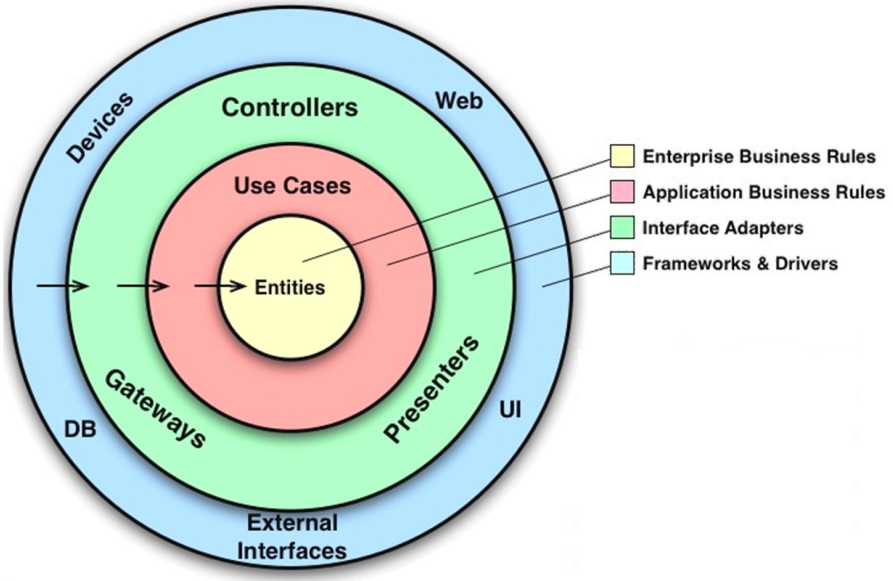
Clean architecture

The Clean architecture

By Robert C. Martin (Uncle Bob)

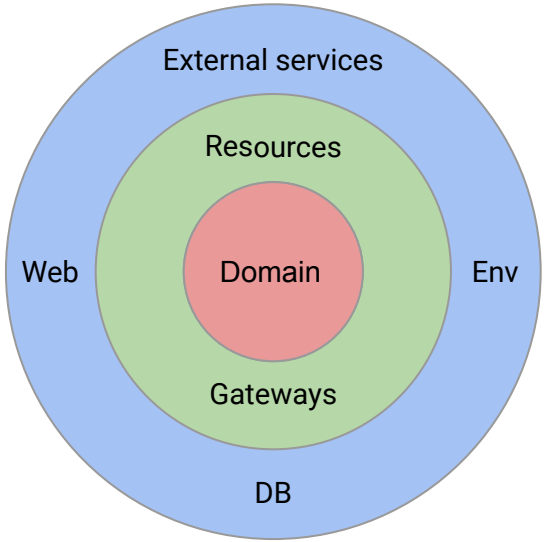
- Clear separation of of concerns between components.
- Dependency rule:
  Dependencies can only point inwards. Inner circle should not be dependent of outer circle specifics.
- Interfaces between each layer.
- Only isolated, simple data structures are passed across the boundaries.

Very OOP focused



https://en.wikipedia.org/wiki/Robert_C._Martin

The Clean architecture - tailored



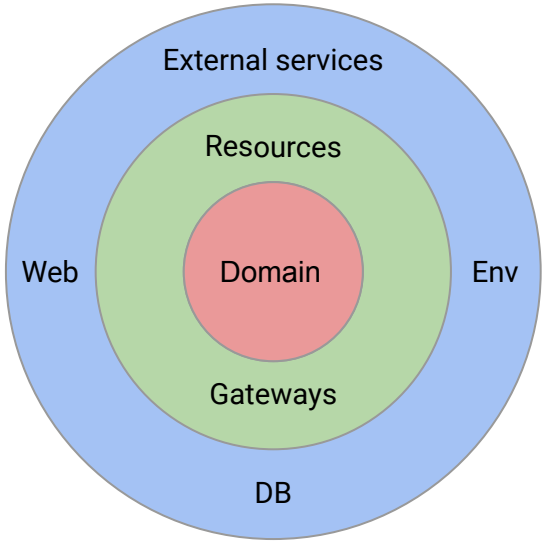
Infrastructure:
Frameworks, environment, and storage and web access.

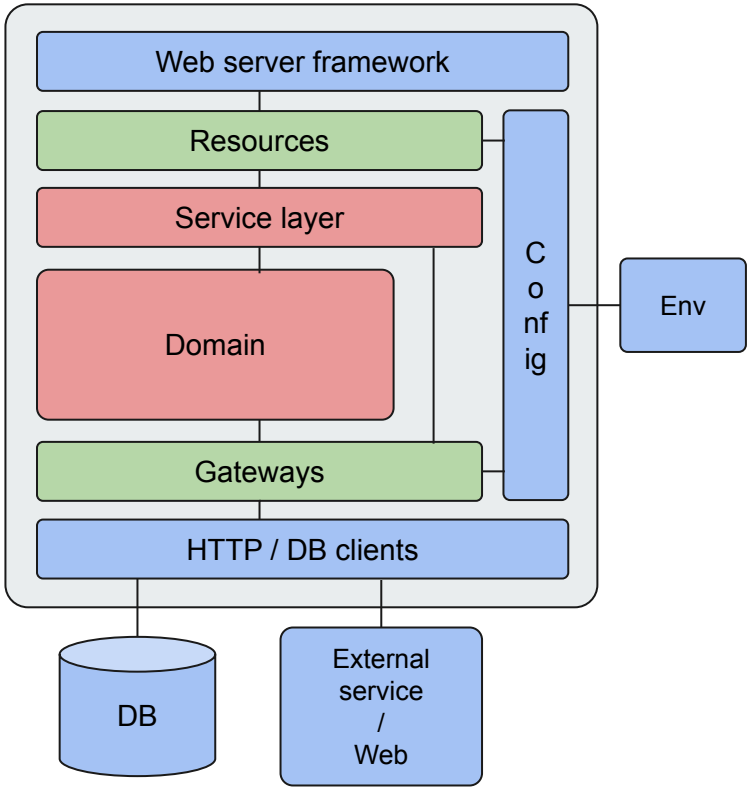
Interface adapters:
Encapsulate the infrastructure layer, and provide a Python API for the domain.

Domain:
All logic core to the microservice.

The anatomy of a microservice

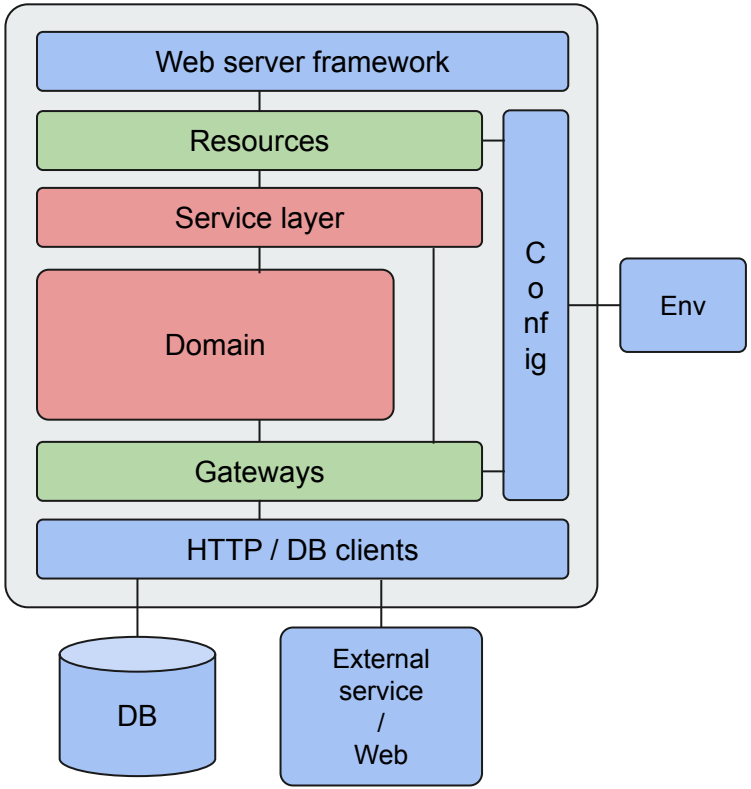
The clean anatomy of a microservice





The anatomy of a microservice

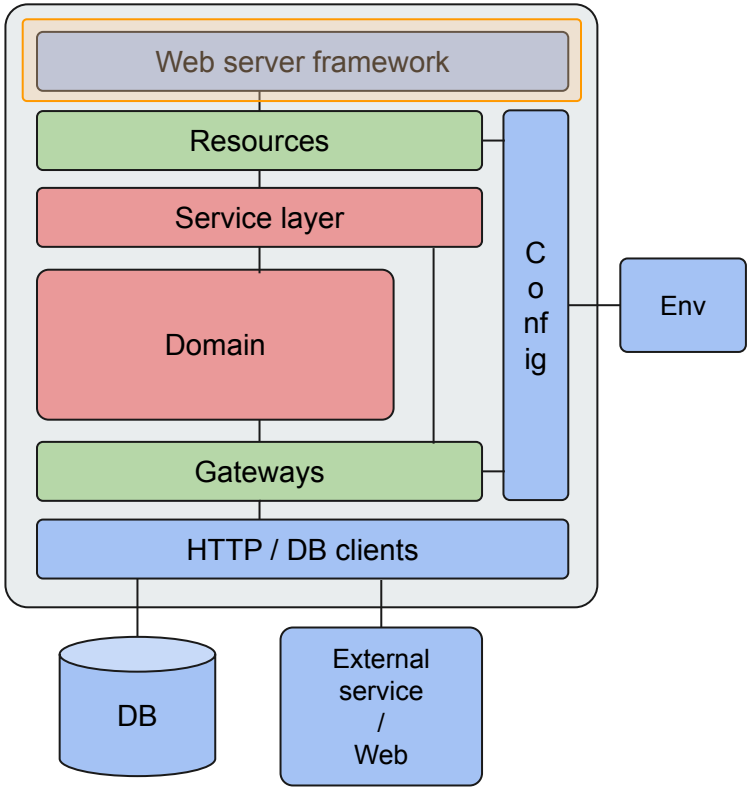
- Microservices display consistent internal structure composed of some or all of the displayed layers.
- The microservice boilerplate aims to provide minimal viable structure that is shared by all microservices.
- Examples will be added containing additional components.



The anatomy of a microservice

Web server framework

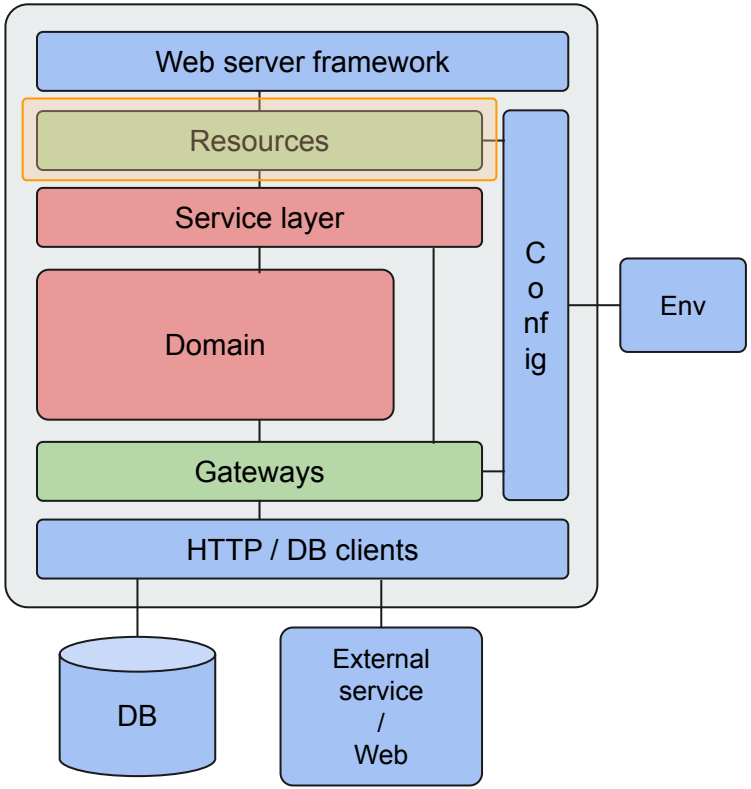
- Act as a mapping between the exposed application protocol (eg. HTTP) and Python.
- Validate requests and responses against application protocol.



The anatomy of a microservice

Resources

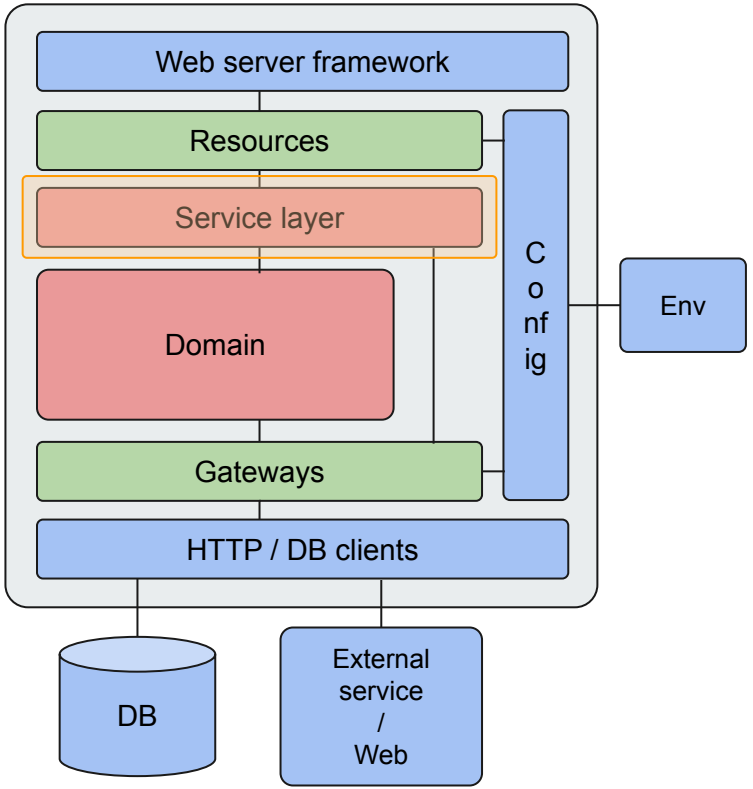
- Act as a mapping between the web server framework and the service domain.
- Thin layer for sanity checking requests and providing protocol specific responses.



The anatomy of a microservice

Service layer

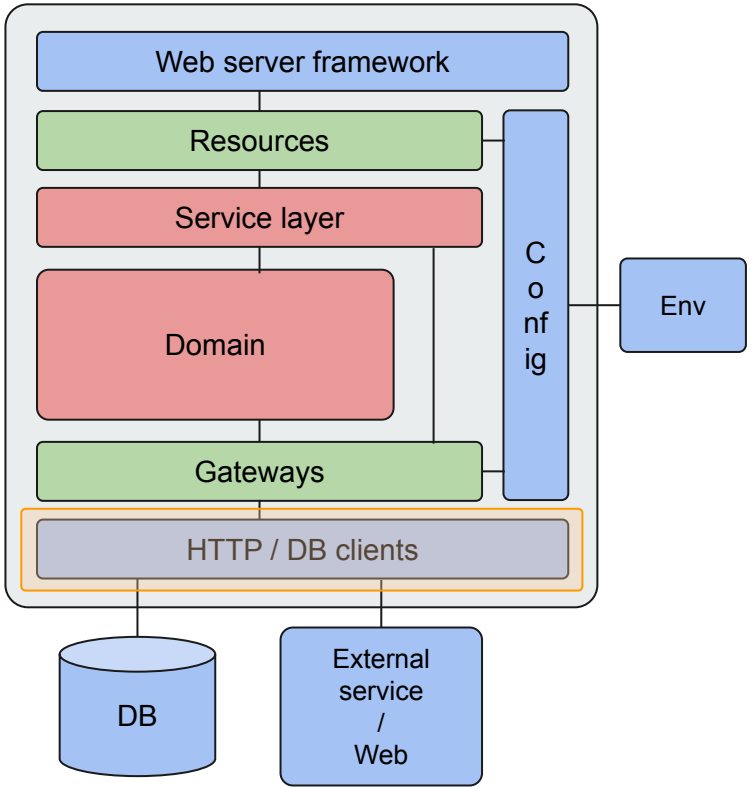
- Defines the interface of the domain with its set of available operations.
- Can be reused by multiple protocol clients; HTTP, GRPC, Queue reader, ...



The anatomy of a microservice

HTTP / DB clients

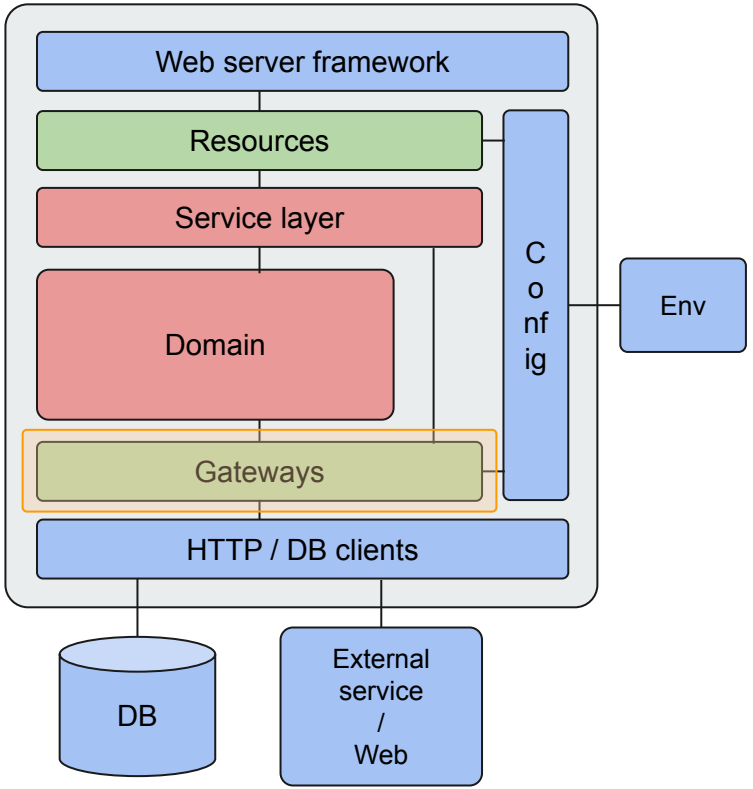
- Clients to handle connections to external datastores and services
- HTTP-client:
  Client that understands the underlying protocol to handle the request-response cycle.
  (eg. python requests)
- DB-client:
  Provide (often client-specific) python representation of data in datastores



The anatomy of a microservice

Gateways

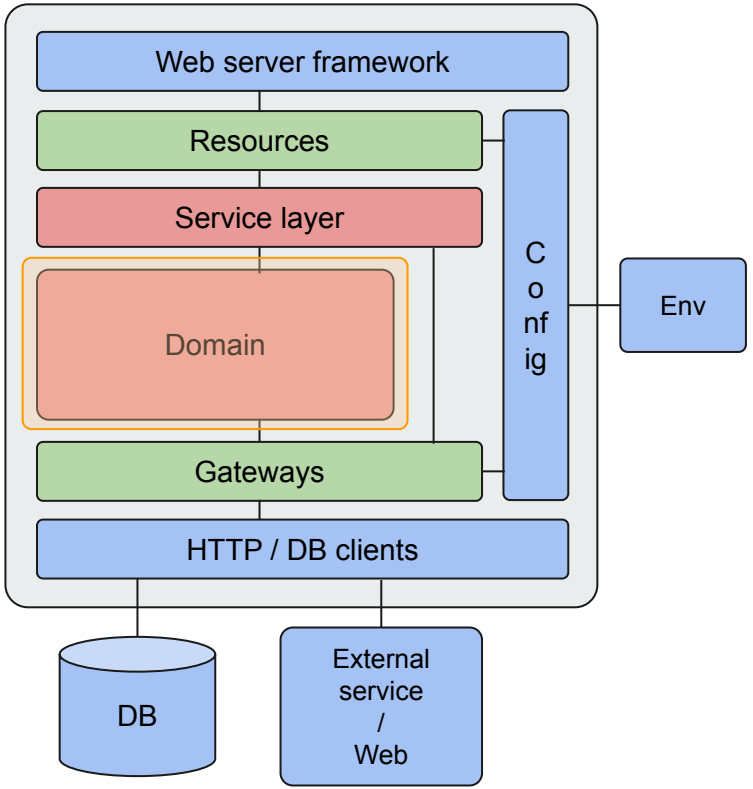
- Isolates domain from details in the database access code.
- Groups query construction code to minimize duplicate query logic.
- Encapsulates message passing with a remote service, marshalling requests and responses from and to Python.



The anatomy of a microservice

Domain

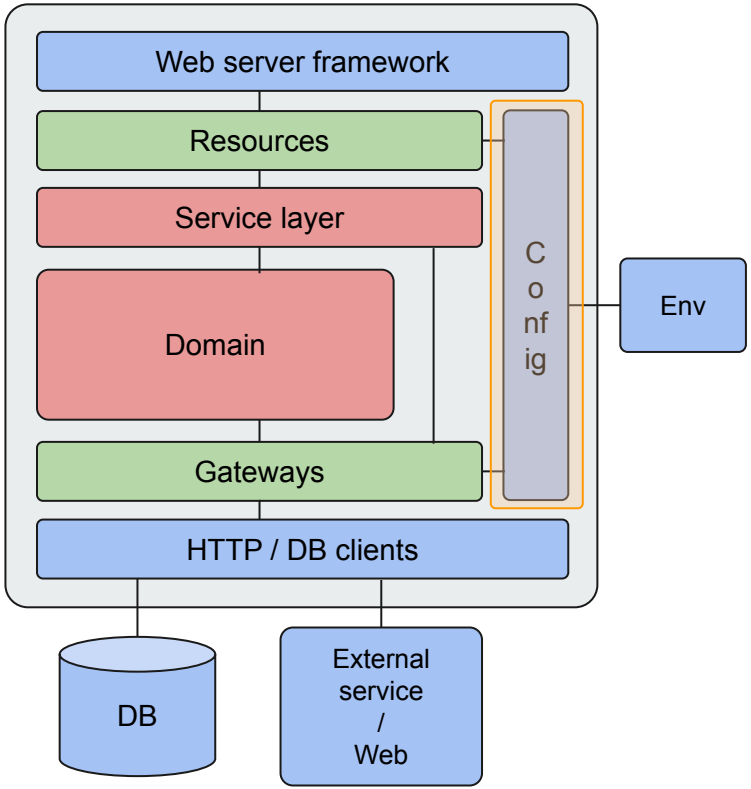
- Contains the actual service logic.
- The domain consists of pure Python with some core libraries. It is independent of any used frameworks and IO clients, and only depends on the interface of the resources and gateways.



The anatomy of a microservice

Config

- Reads and groups the configuration of the service.
- All configurable parameters of the service should be extracted as config, so they can be set and changed in the environment at run time.



Advantages of the clean anatomy

- **Independent of externals**
  When any of the external parts of the system become obsolete, like the database, or the web framework, you can replace those obsolete elements with a minimum of fuss.

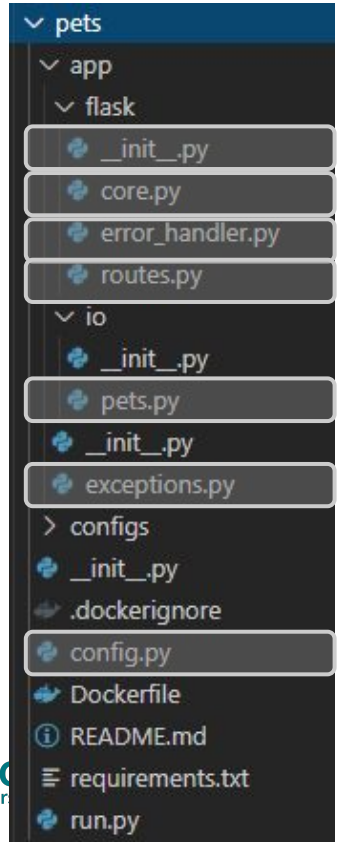
- **Scalable and maintainable**
  A modular architecture allows for scaling without an explosion in complexity. New modules can be added independently of others. Multiple developers can work in parallel on different modules.

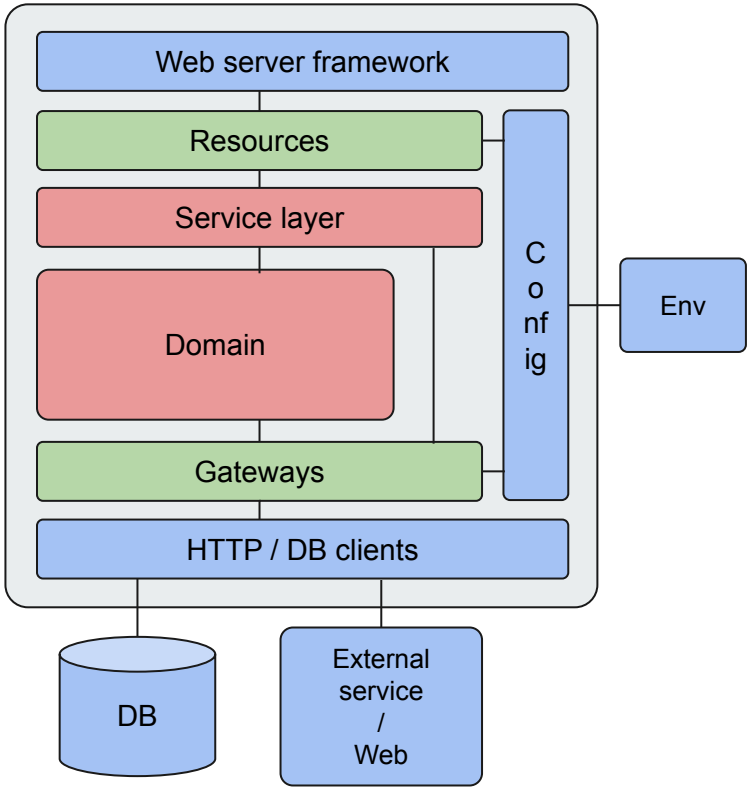
- **Deploy everywhere**
  Since the configuration is separated, the same code can be deployed in different environments without changes.

The anatomy of an example

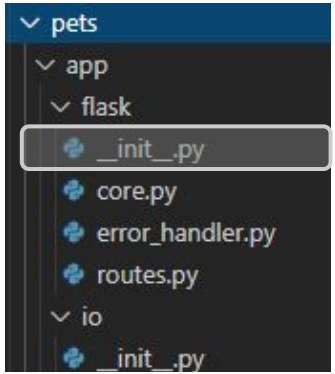
Mapping the structure on the pets example

The anatomy of a microservice: pets example

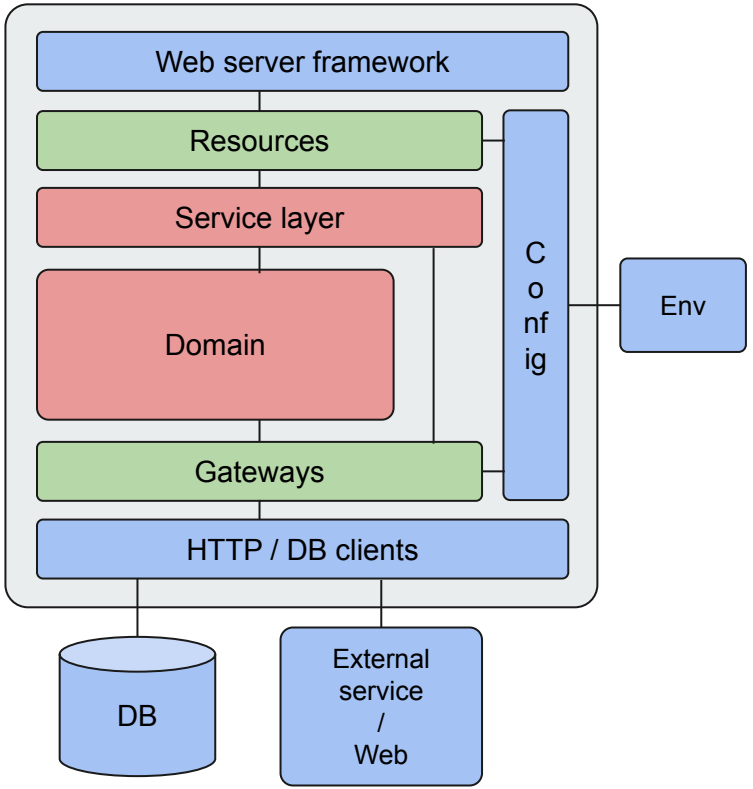




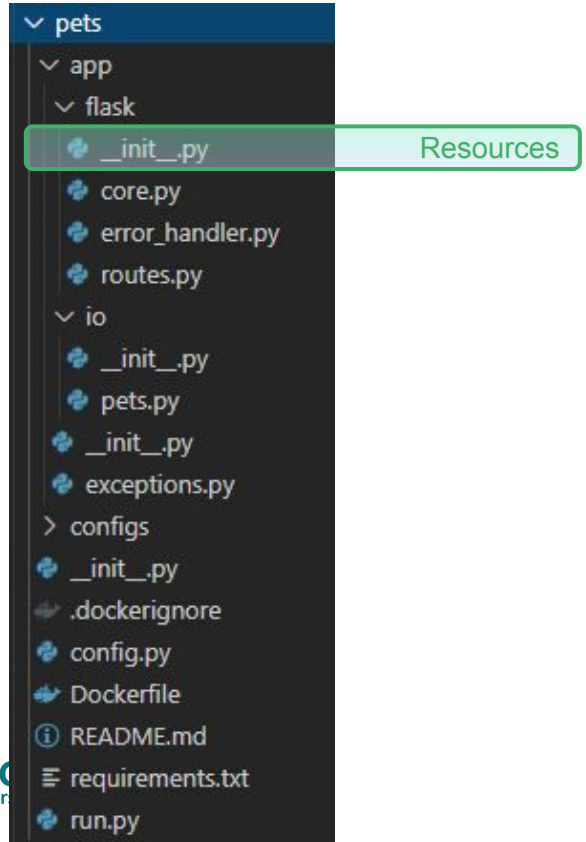
The anatomy of a microservice: pets example

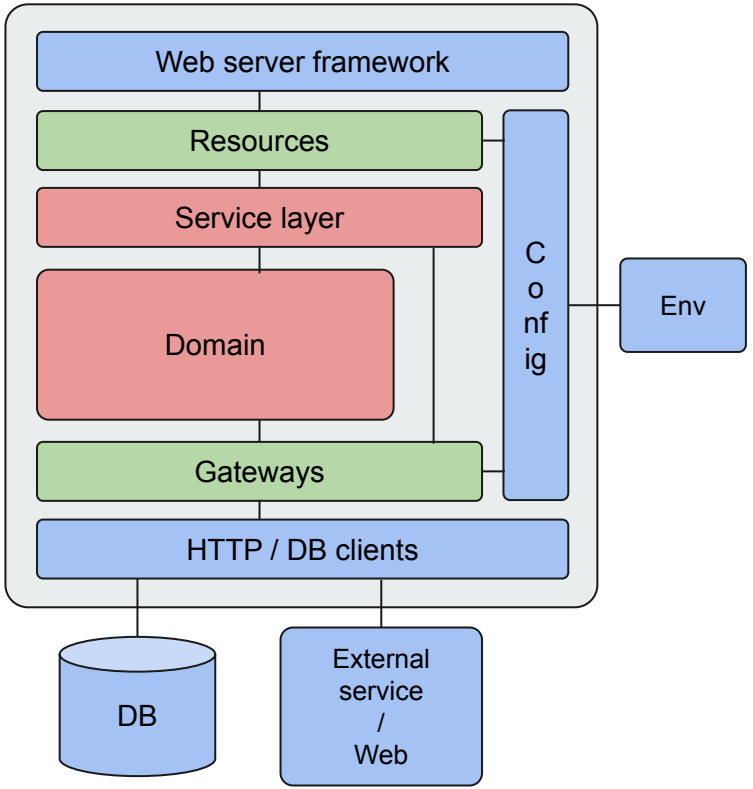



```
20      # Use newer version of swagger ui
21      options = {'swagger_path': swagger_ui_path}
22
23      app = connexion.App(__name__,
24                          specification_dir=config.SPECIFICATION_DIR,
25                          options=options)
26
27      app.add_api('swagger.yaml',
28                  strict_validation=True,
29                  validate_responses=True)
30
31      error_handler.register_error_handlers(app)
```

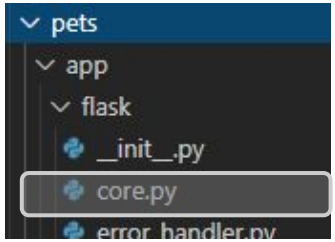


The anatomy of a microservice: pets example

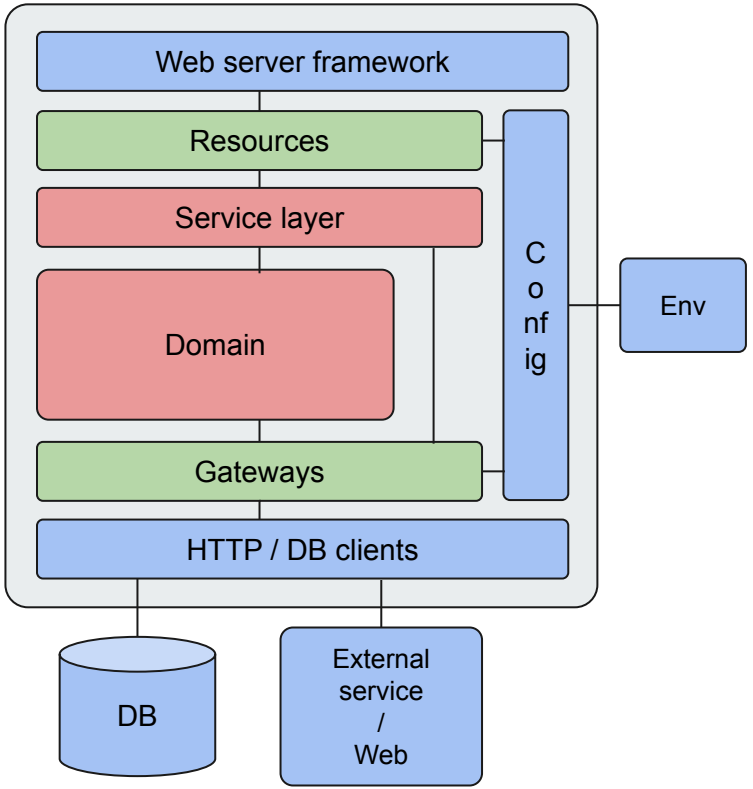




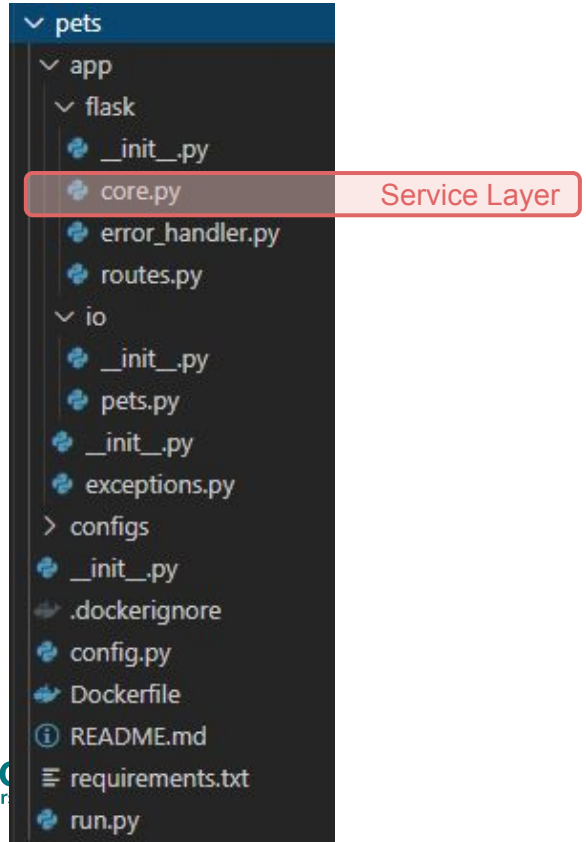
The anatomy of a microservice: pets example

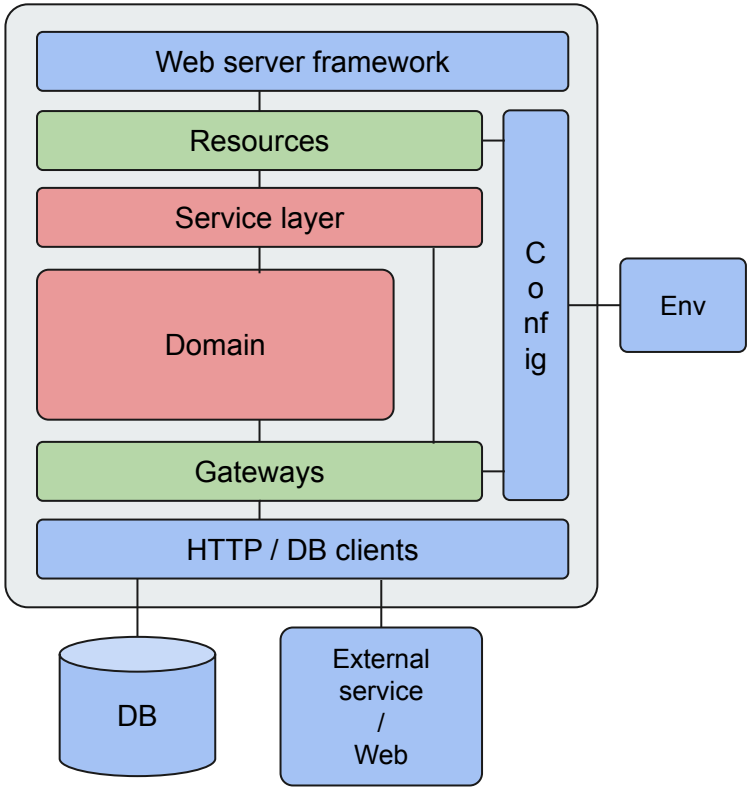



```
7   from app.io import pets
8
9
10  def get_all_pets():
11      """Get array of all pets."""
12      return pets.get_all()
13
14
15  def get_pet_by_id(pet_id):
16      """Get single pet by id."""
17      return pets.get(pet_id)
18
19
20  def add_pet(*, name, owner, description):
21      """Add a new pet."""
22      return pets.create(name=name, owner=owner, description=d
```

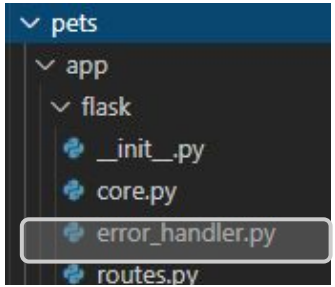


The anatomy of a microservice: pets example

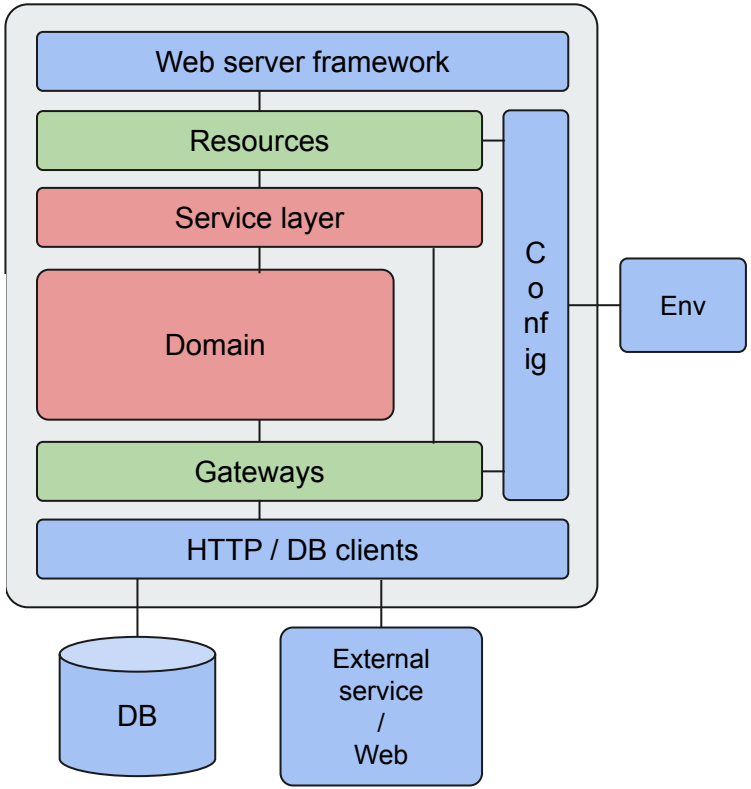




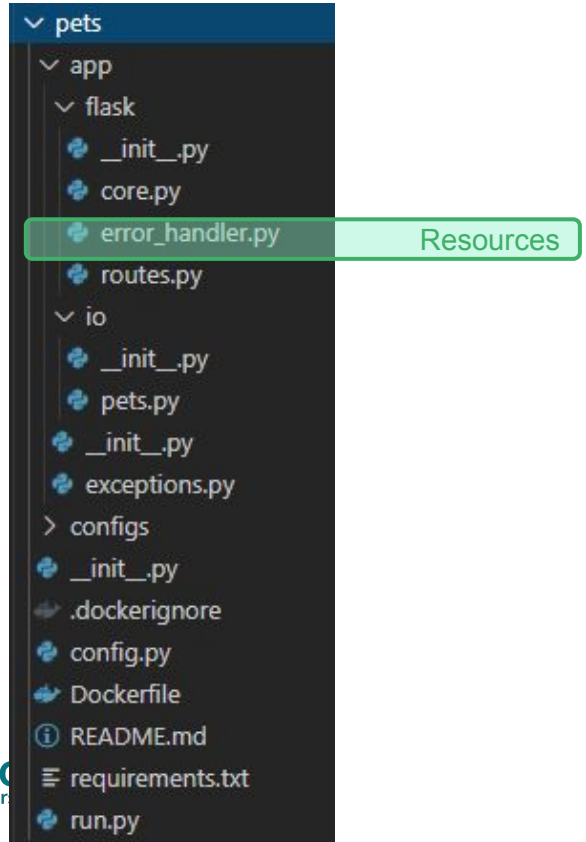
The anatomy of a microservice: pets example

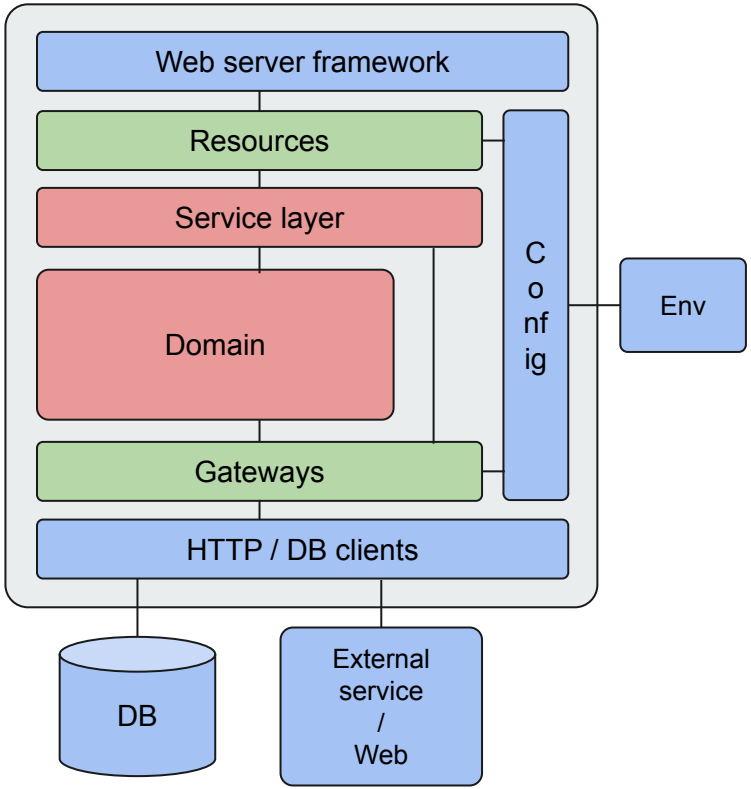



```
8  from flask import jsonify
9
10 from app import exceptions
11
12
13 def handle_custom_error(error):
14     """Handle custom errors."""
15     response = jsonify({'message': error.message})
16     response.status_code = error.status_code
17     return response
18
19
20 def register_error_handlers(app):
21     """Add error handlers to the app."""
22     app.add_error_handler(exceptions.PetNotFound, handle_custo
```

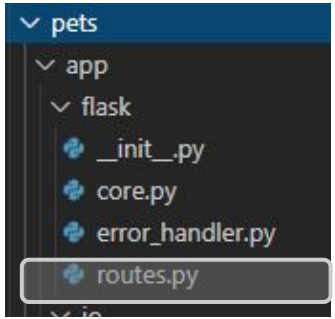


The anatomy of a microservice: pets example

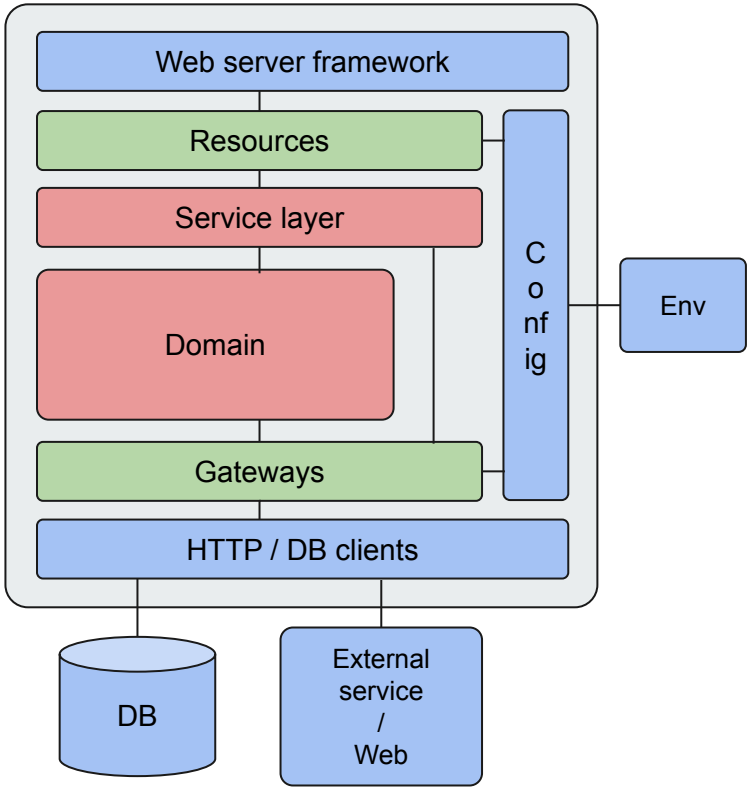




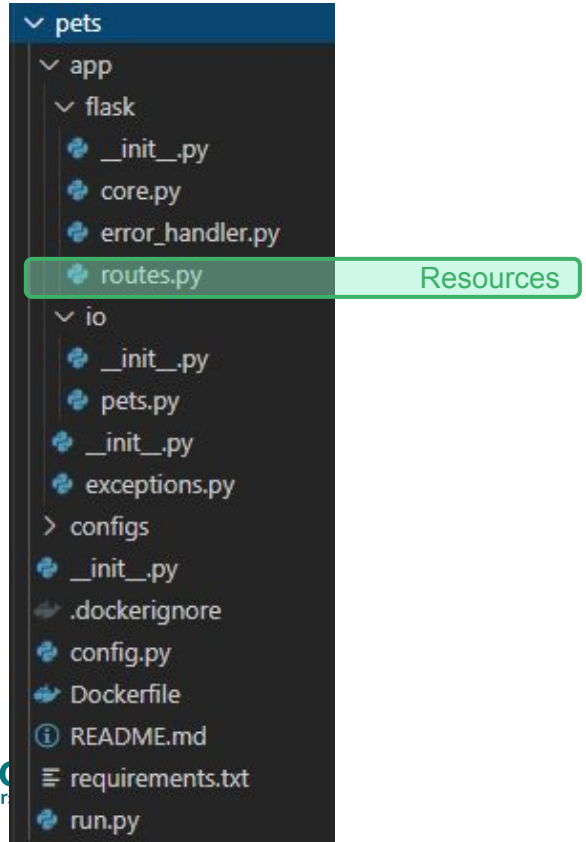
The anatomy of a microservice: pets example

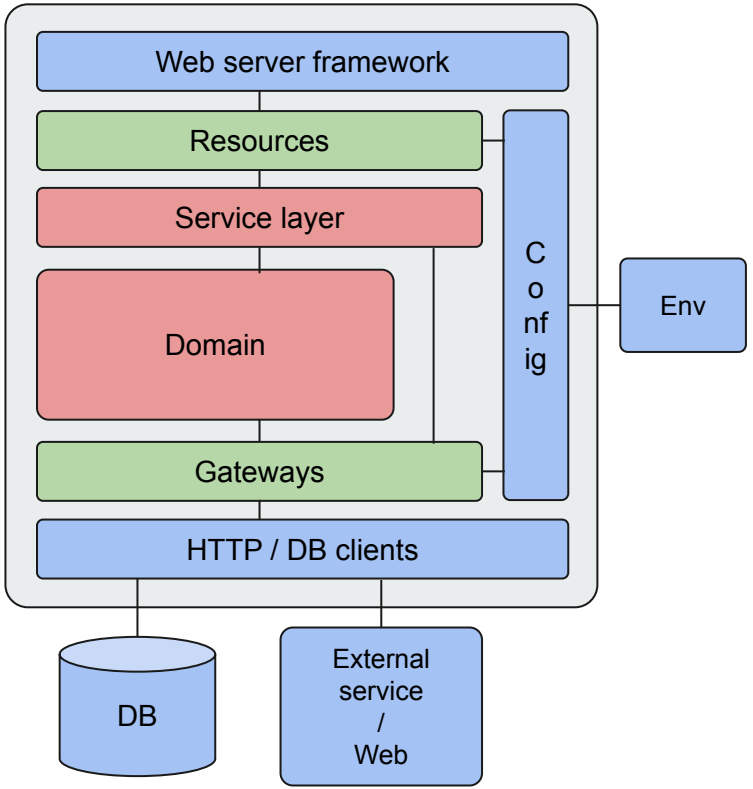



```
from app.flask import core


class Pet:
    """Includes all HTTP methods for /pets/pet_id>"""

    @staticmethod
    def get(pet_id):
        """Get an existing pet."""
        return core.get_pet_by_id(pet_id), 200

    @staticmethod
    def delete(pet_id):
        """Delete an existing pet."""
        return core.delete_pet_by_id(pet_id), 204
```

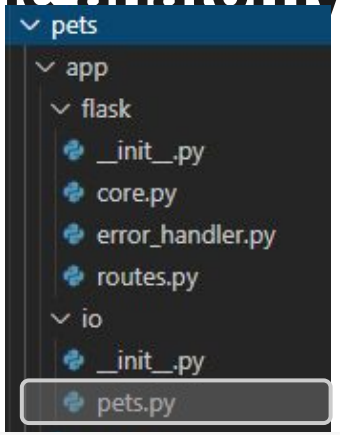


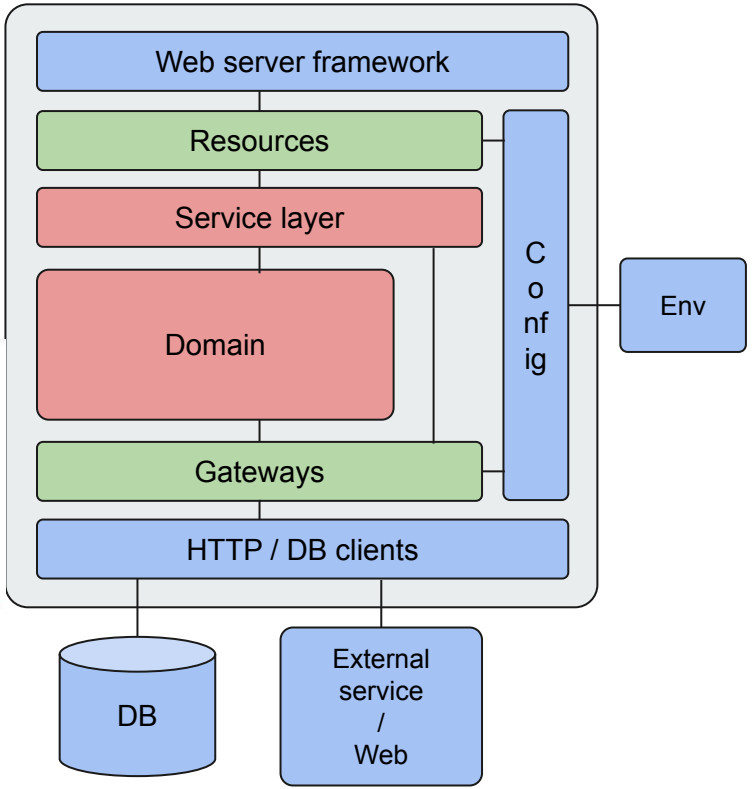
The anatomy of a microservice: pets example





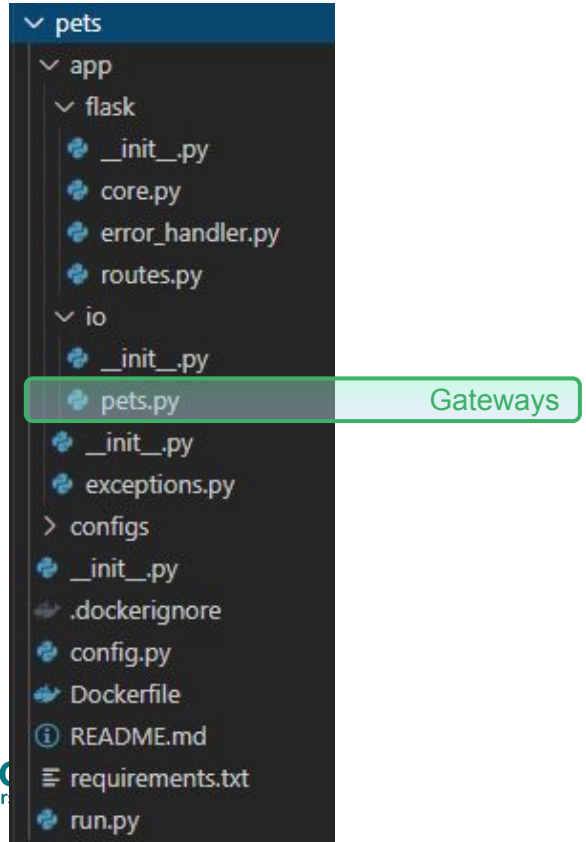
The anatomy of a microservice: pets example

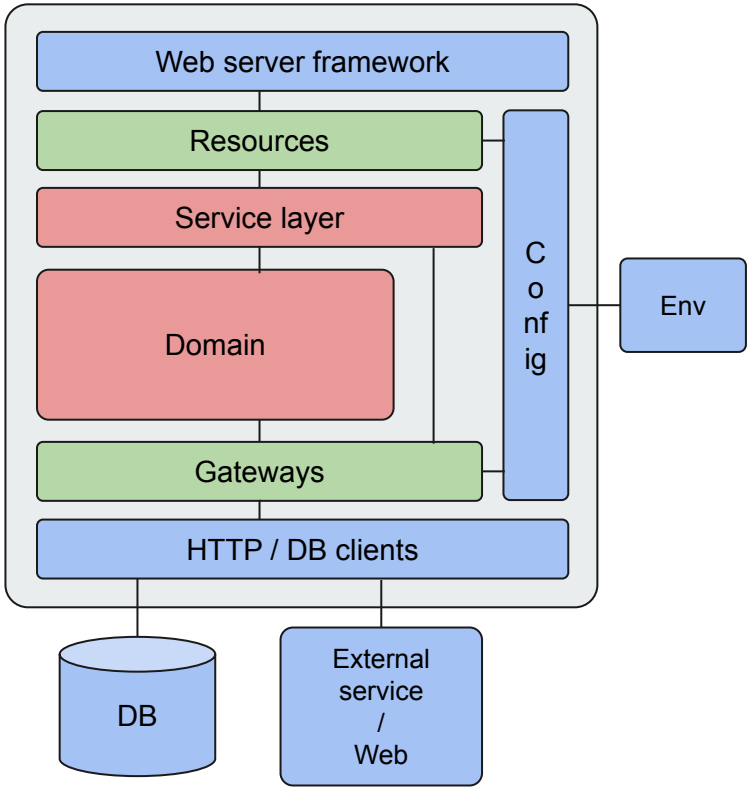




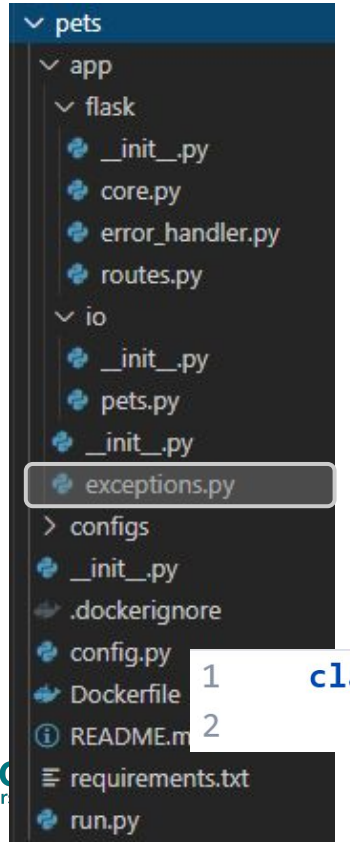

```
11  client = firestore.Client()
12  pets = client.collection('pets')
13
14
15  def get(pet_id: str) -> dict:
16      """Get a pet by id.
17
18      Raises:
19          exceptions.PetNotFound: If no pet with the specified id can be found.
20
21      """
22      doc_ref = pets.document(pet_id)
23      pet = doc_ref.get()
24      if not pet.exists:
25          raise exceptions.PetNotFound('Pet with id {pet_id} was not found.')
26      return doc_ref.get().to_dict()
```

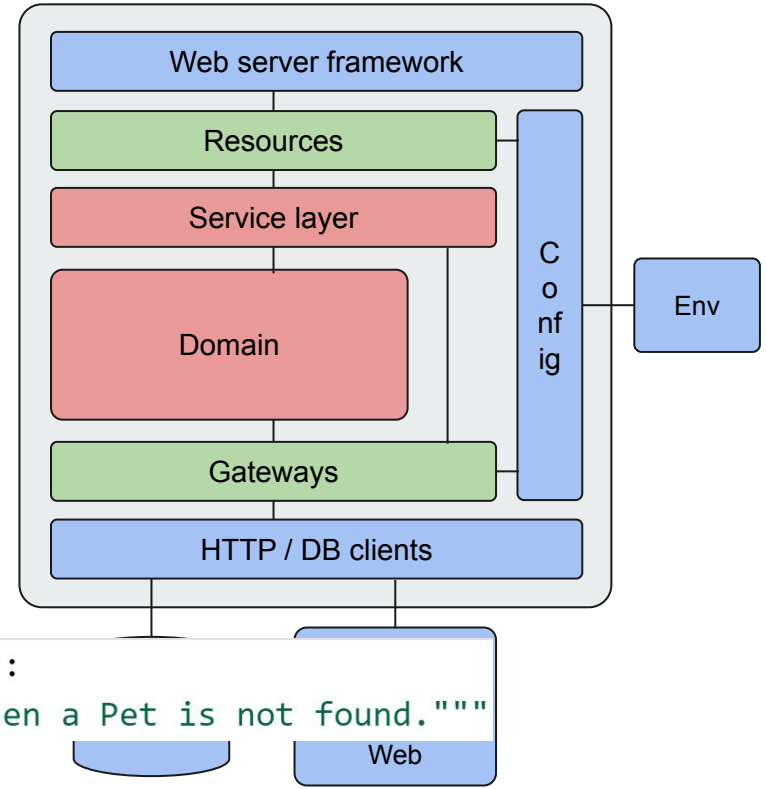
The anatomy of a microservice: pets example





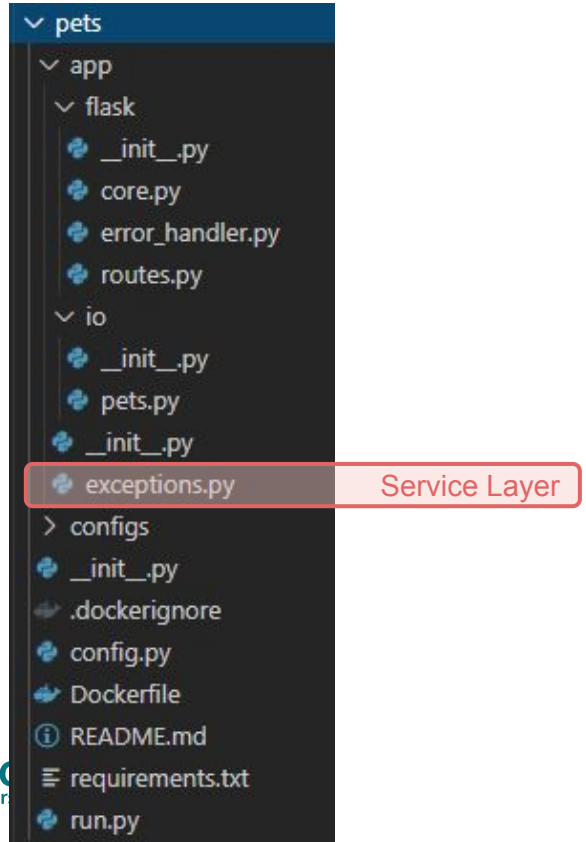
The anatomy of a microservice: pets example

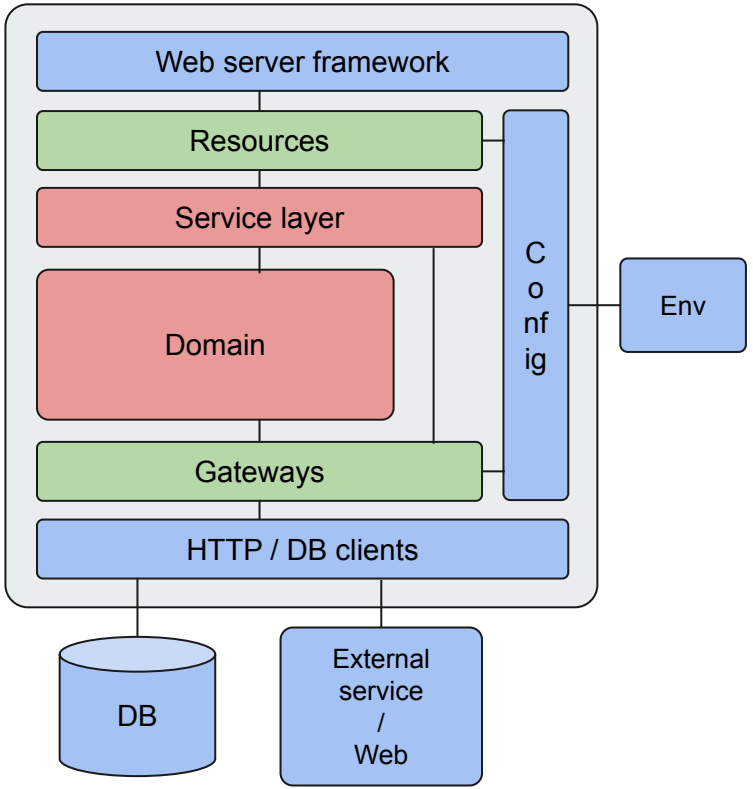




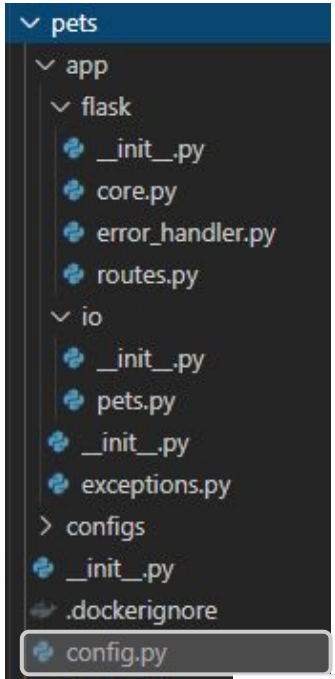

```python
class PetNotFound(Exception):
    """Exception to raise when a Pet is not found."""
```

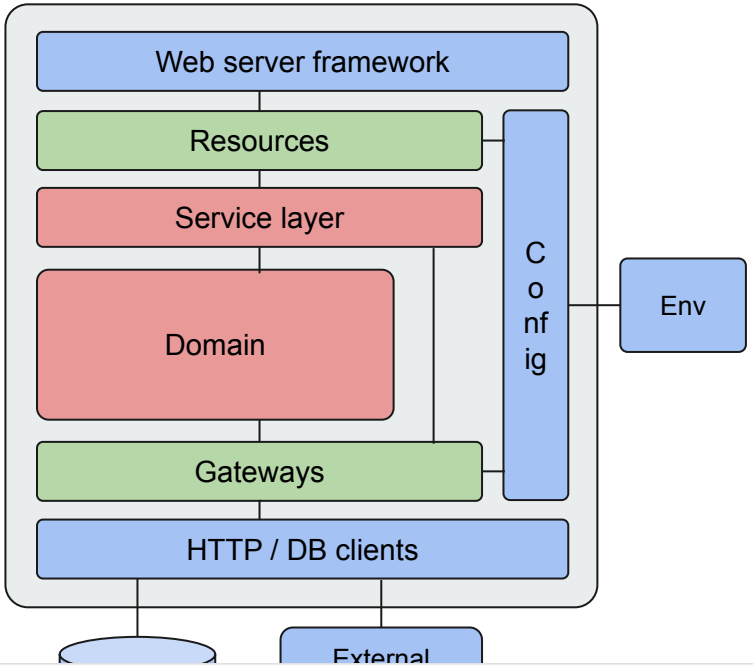
The anatomy of a microservice: pets example





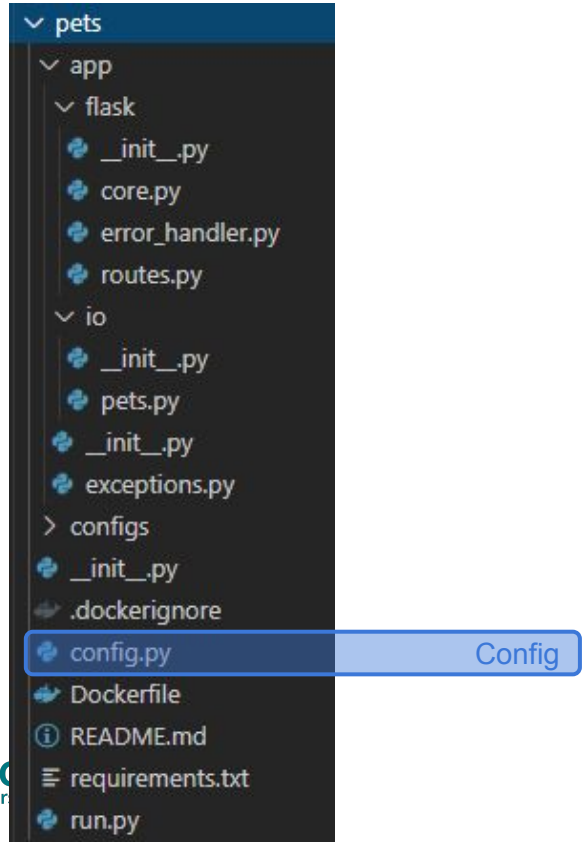
The anatomy of a microservice: pets example

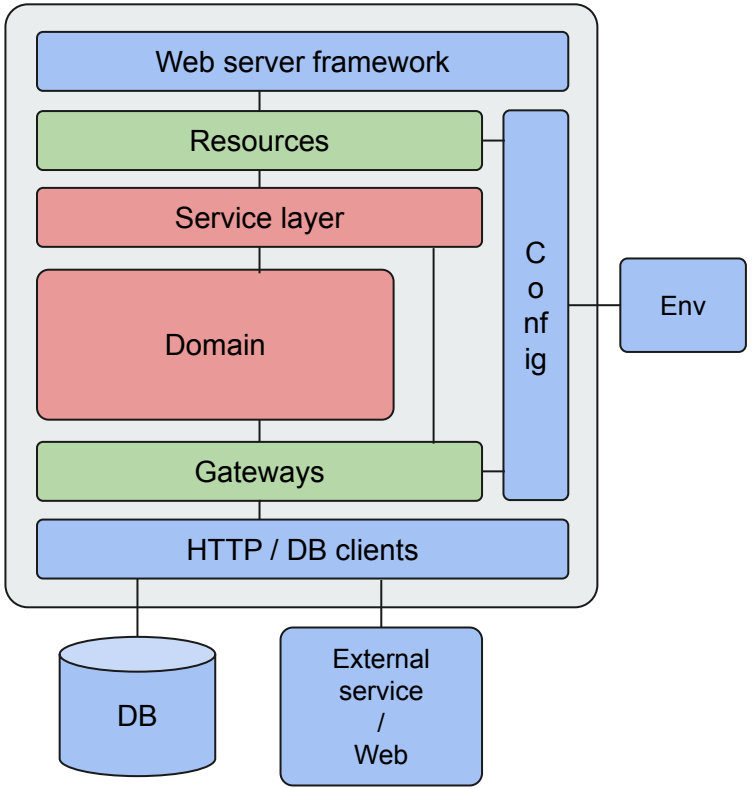




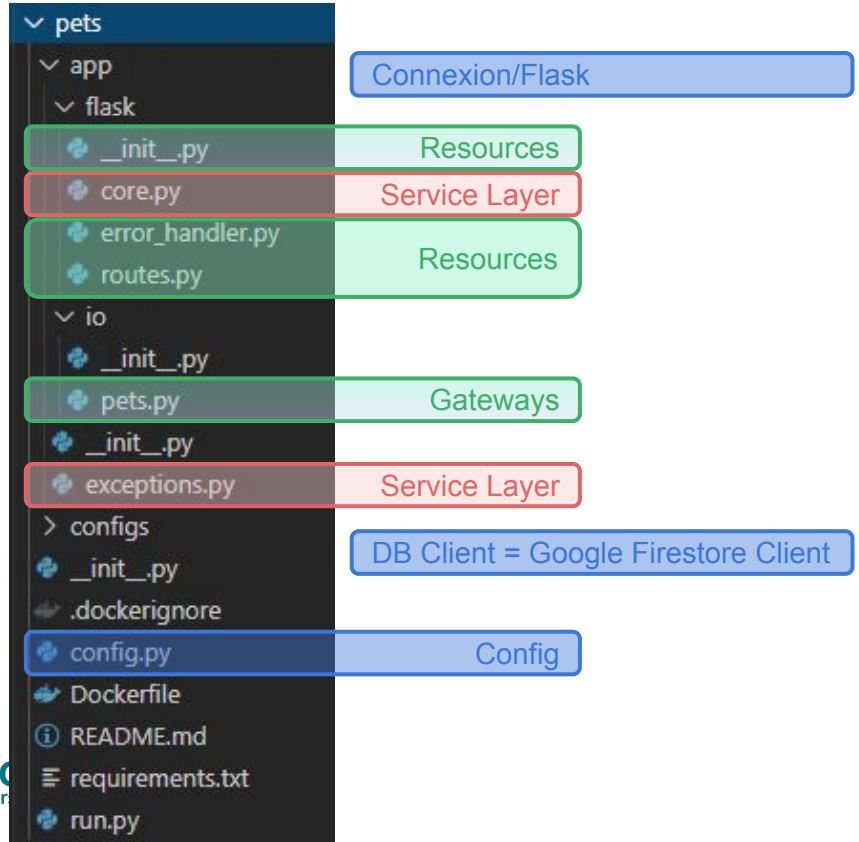

```
1  import os
2
3  SPECIFICATION_DIR = os.path.join(os.path.dirname(os.path.abspath(__file__)),
4                                   'configs')
```

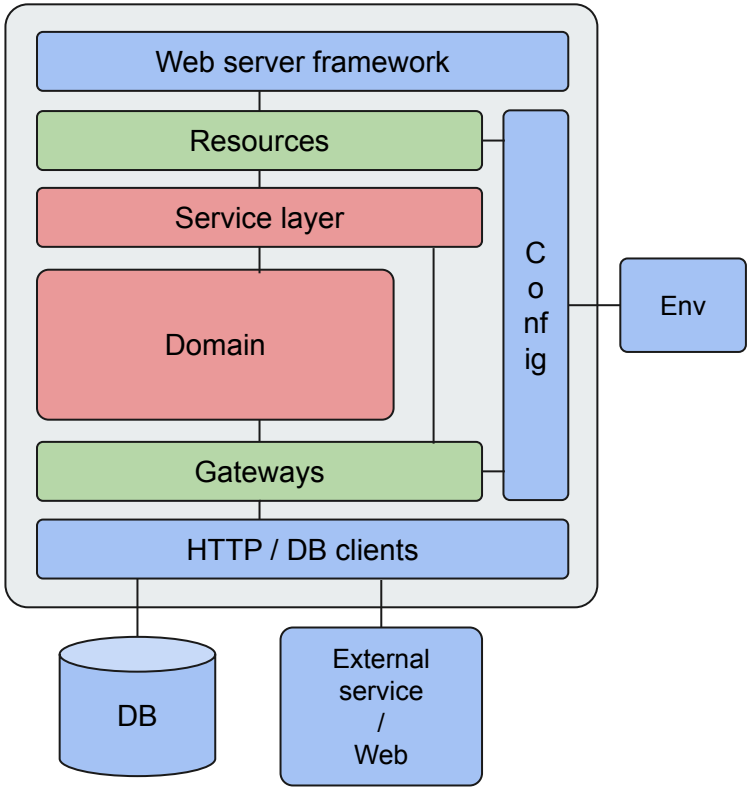
The anatomy of a microservice: pets example



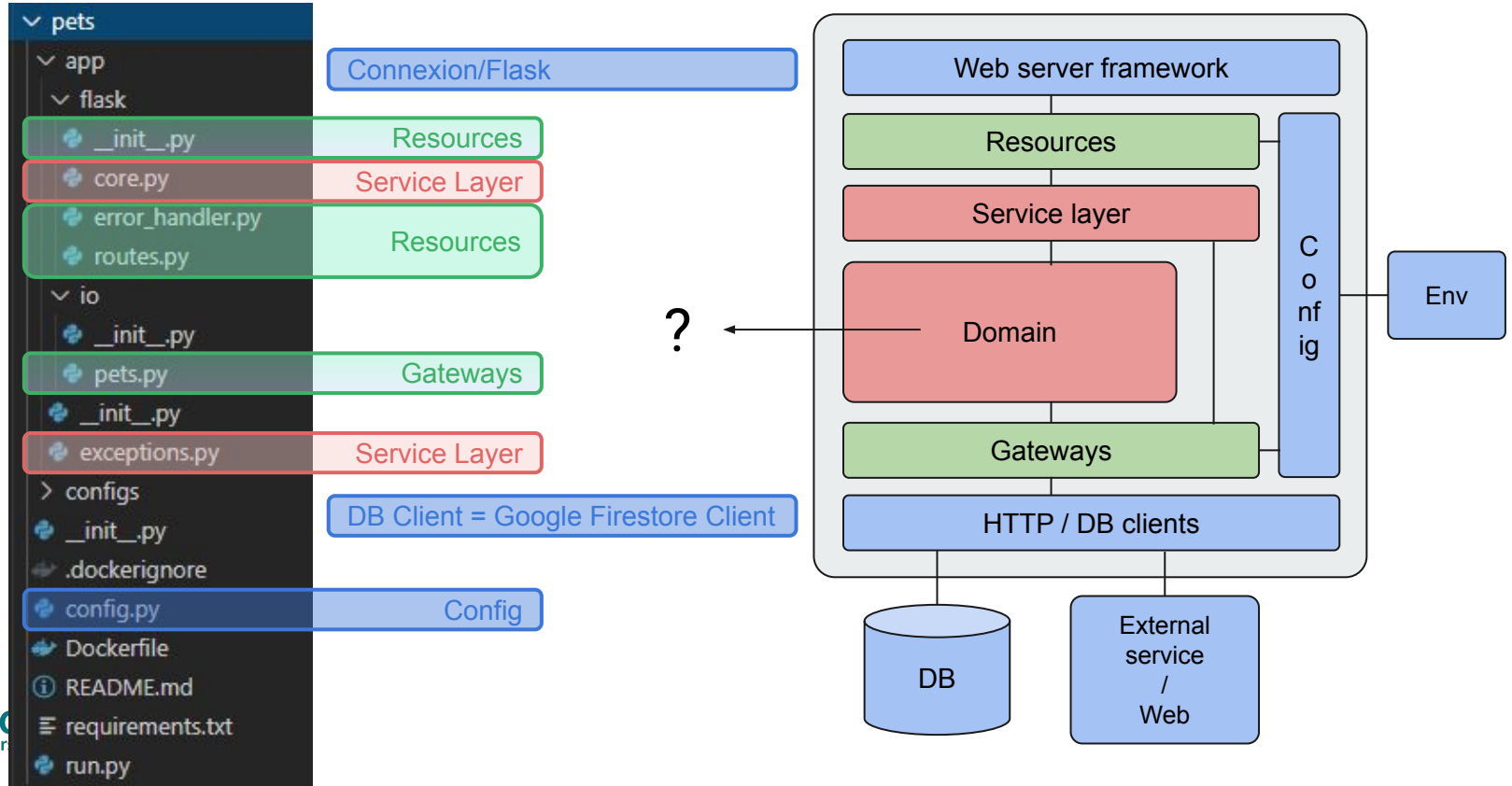


The anatomy of a microservice: pets example

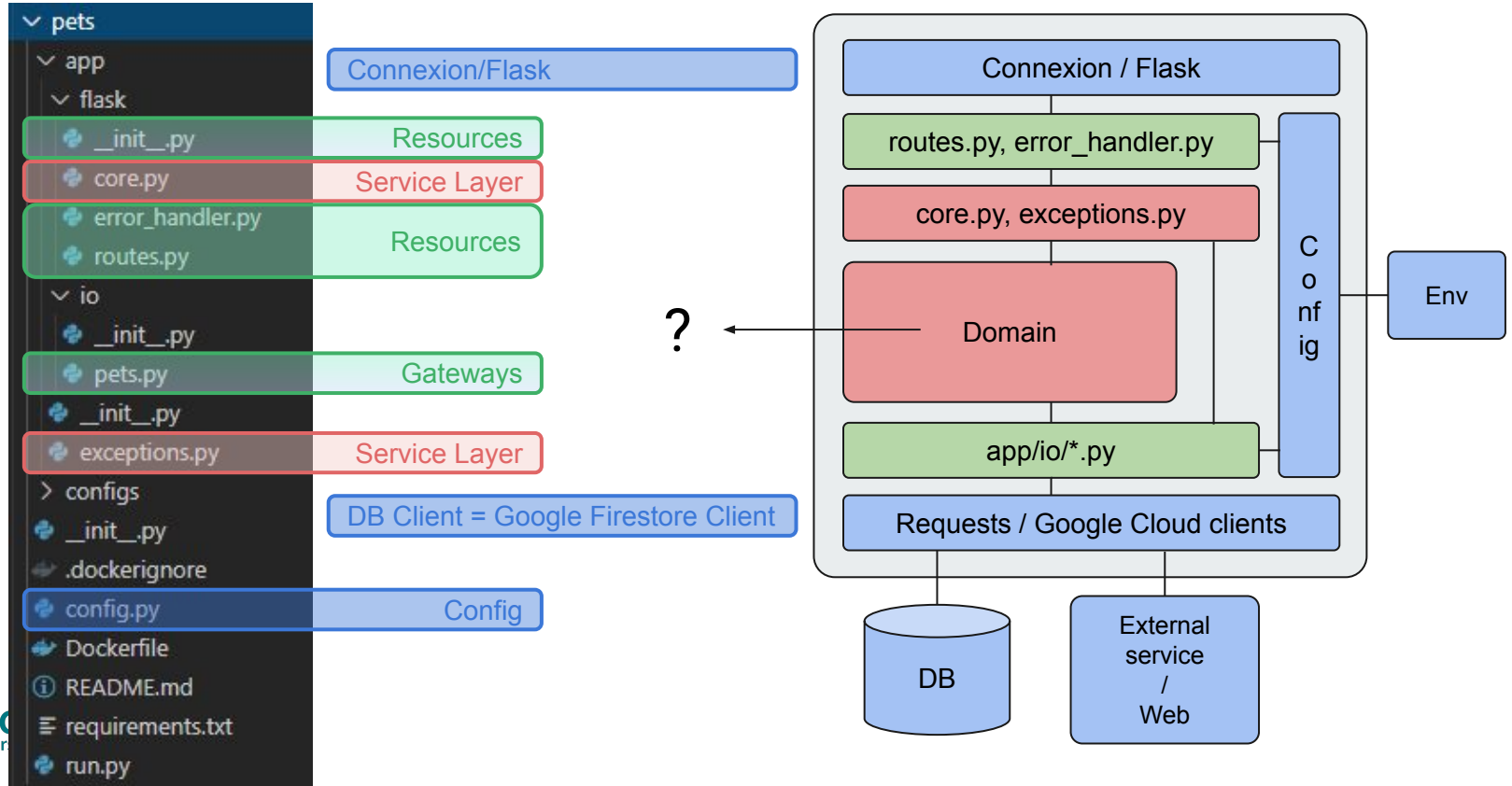




The anatomy of a microservice: pets example



The anatomy of a microservice: pets example



Monoliths aren't for Dinosaurs

The pros and cons of monoliths

LIÈGE université

Credits where credits are due

Jasper Van den Bossche
Software Engineer @ ML6

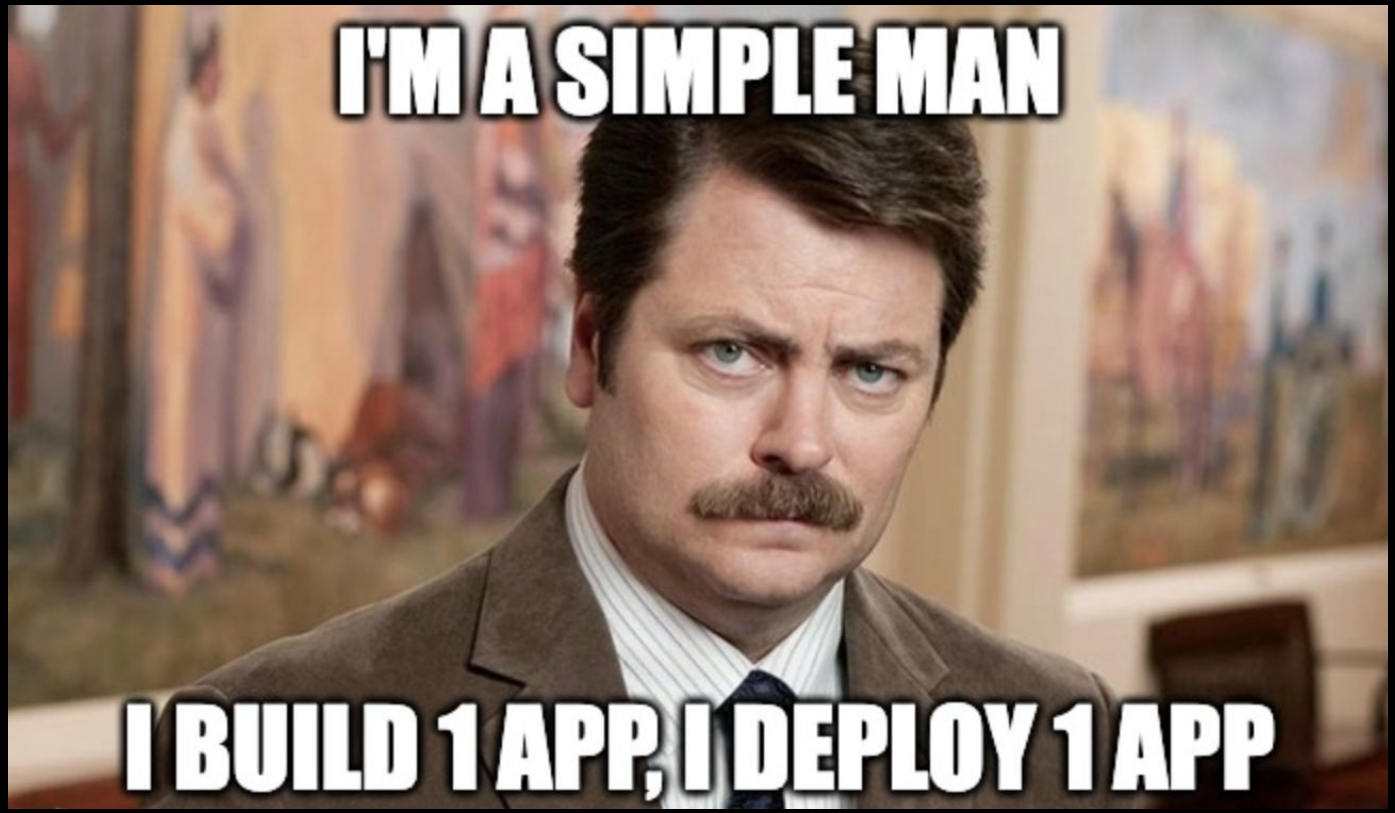
Credits where credits are due

Jasper Van den Bossche
Software Engineer @ ML6

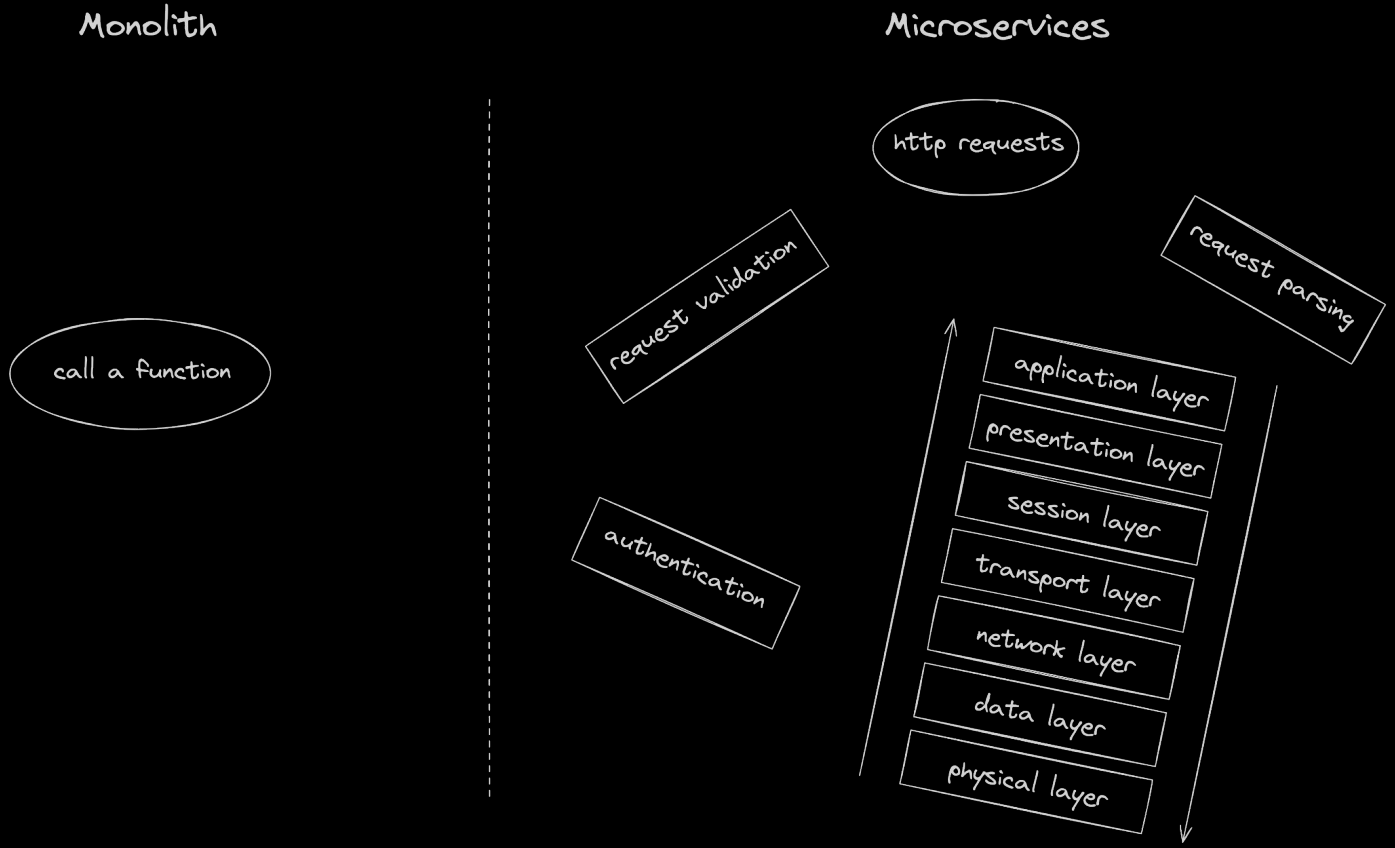
(Disclaimer: Good sense of humour...)

NOT the Goal of this Talk

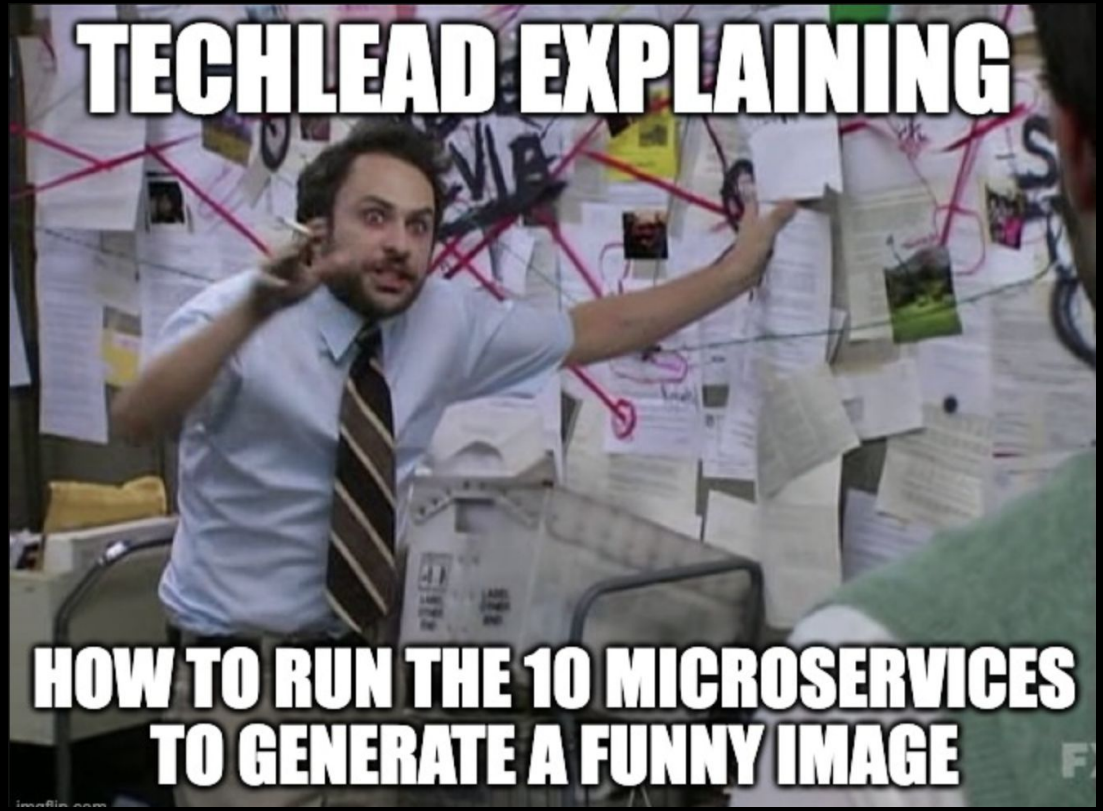
Simplicity



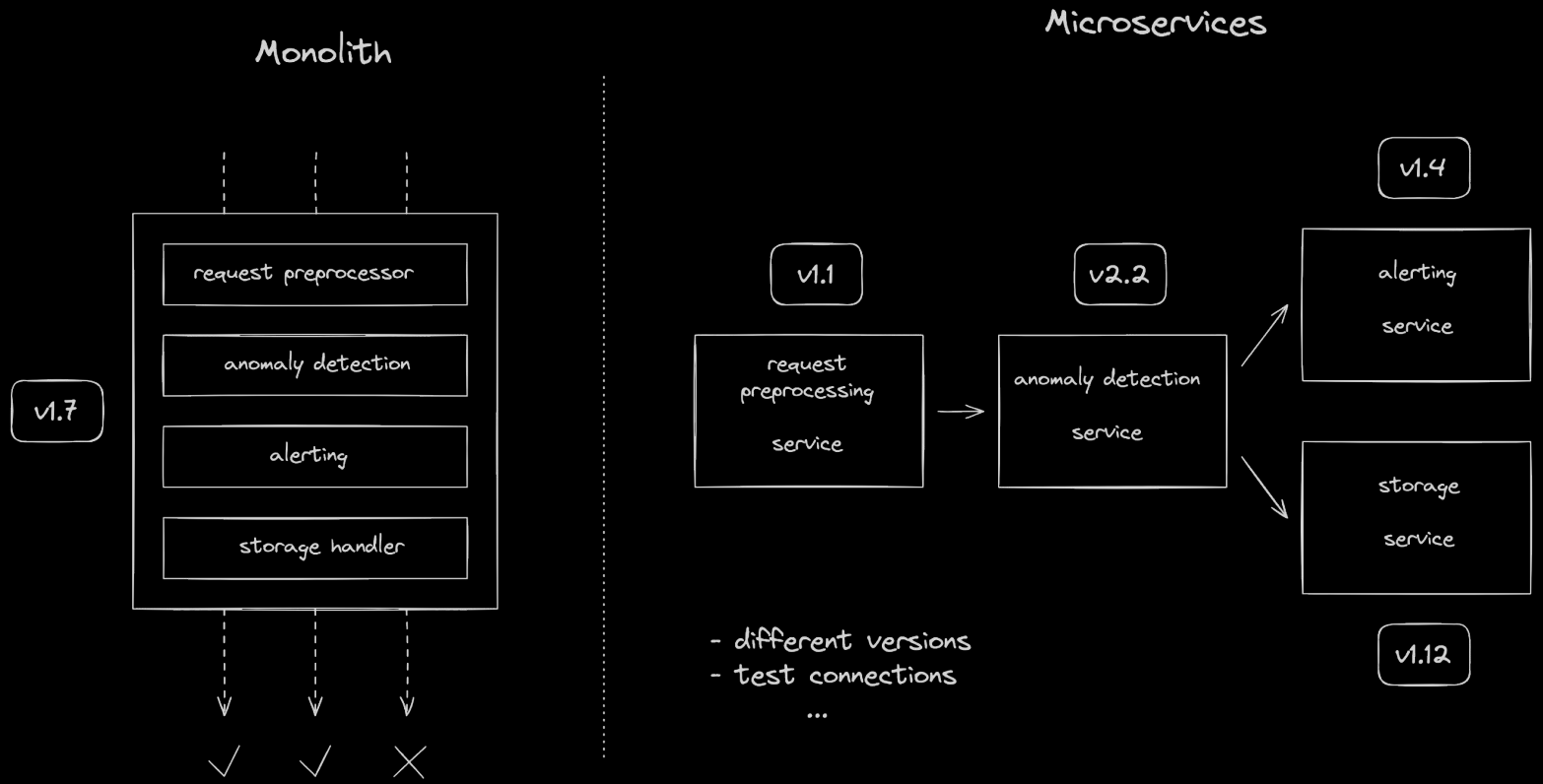
Performance



End-to-End Testing

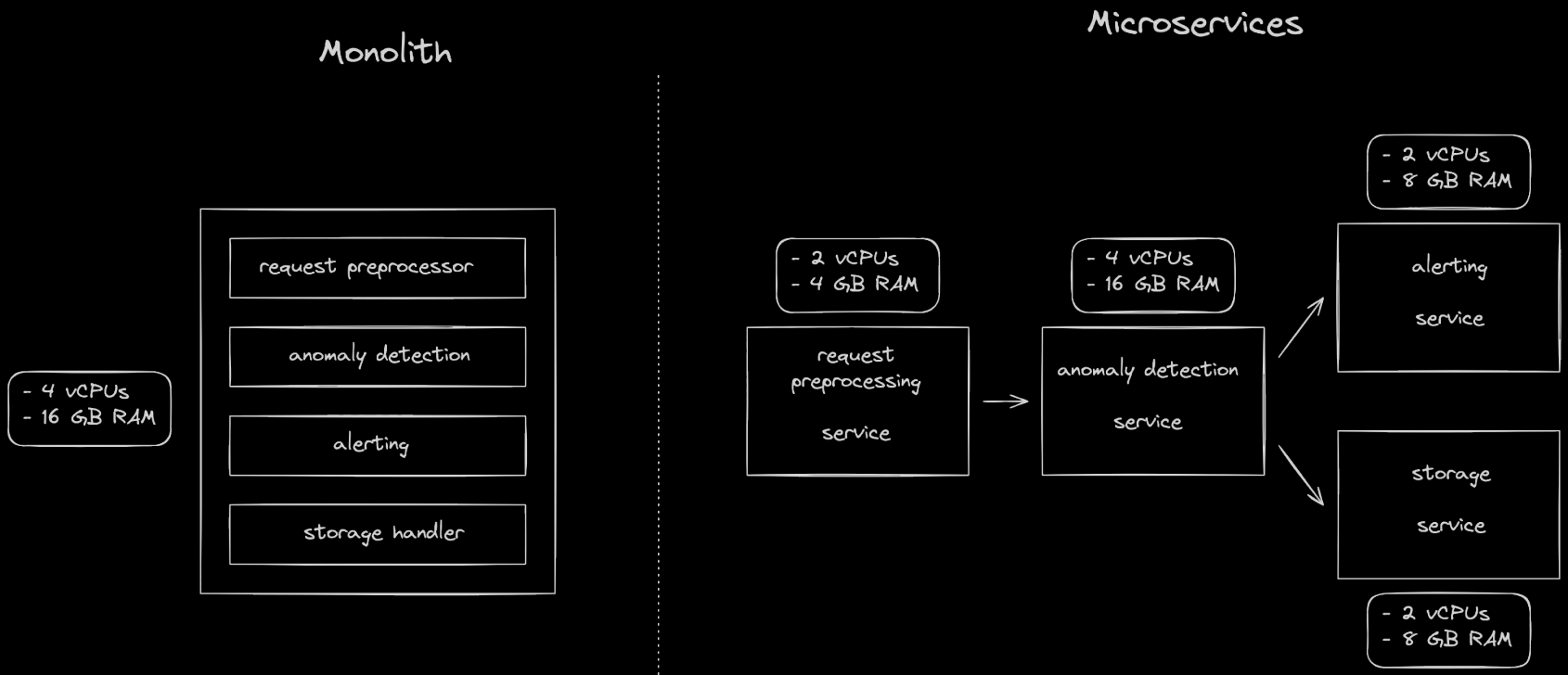


End-to-End Testing

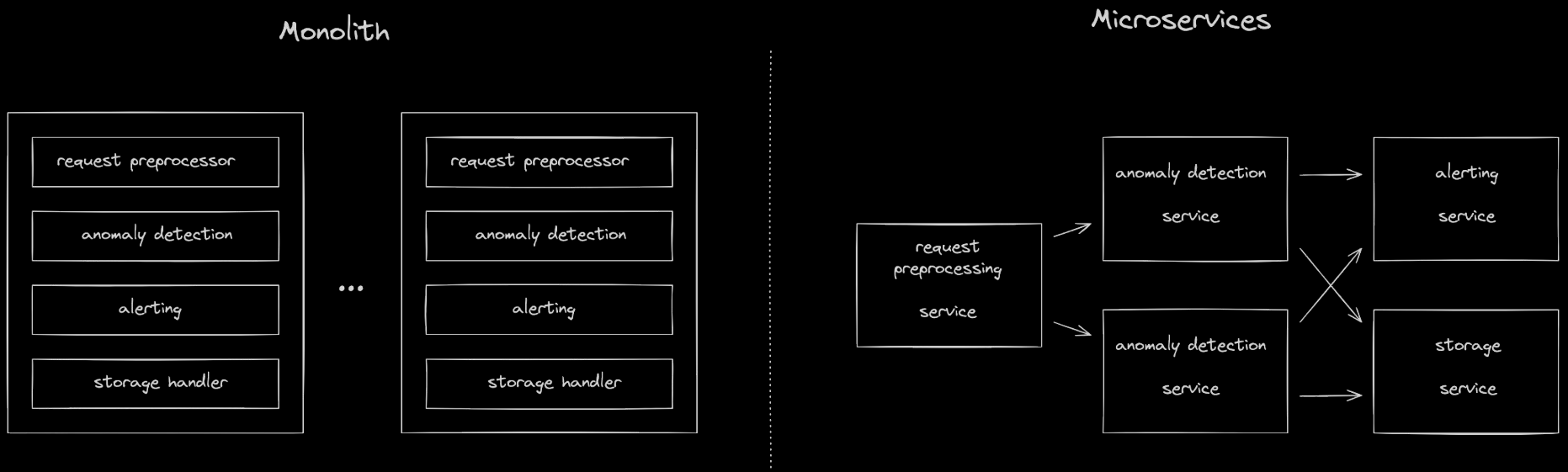


The not so good stuff

Scalability & Resource Utilization

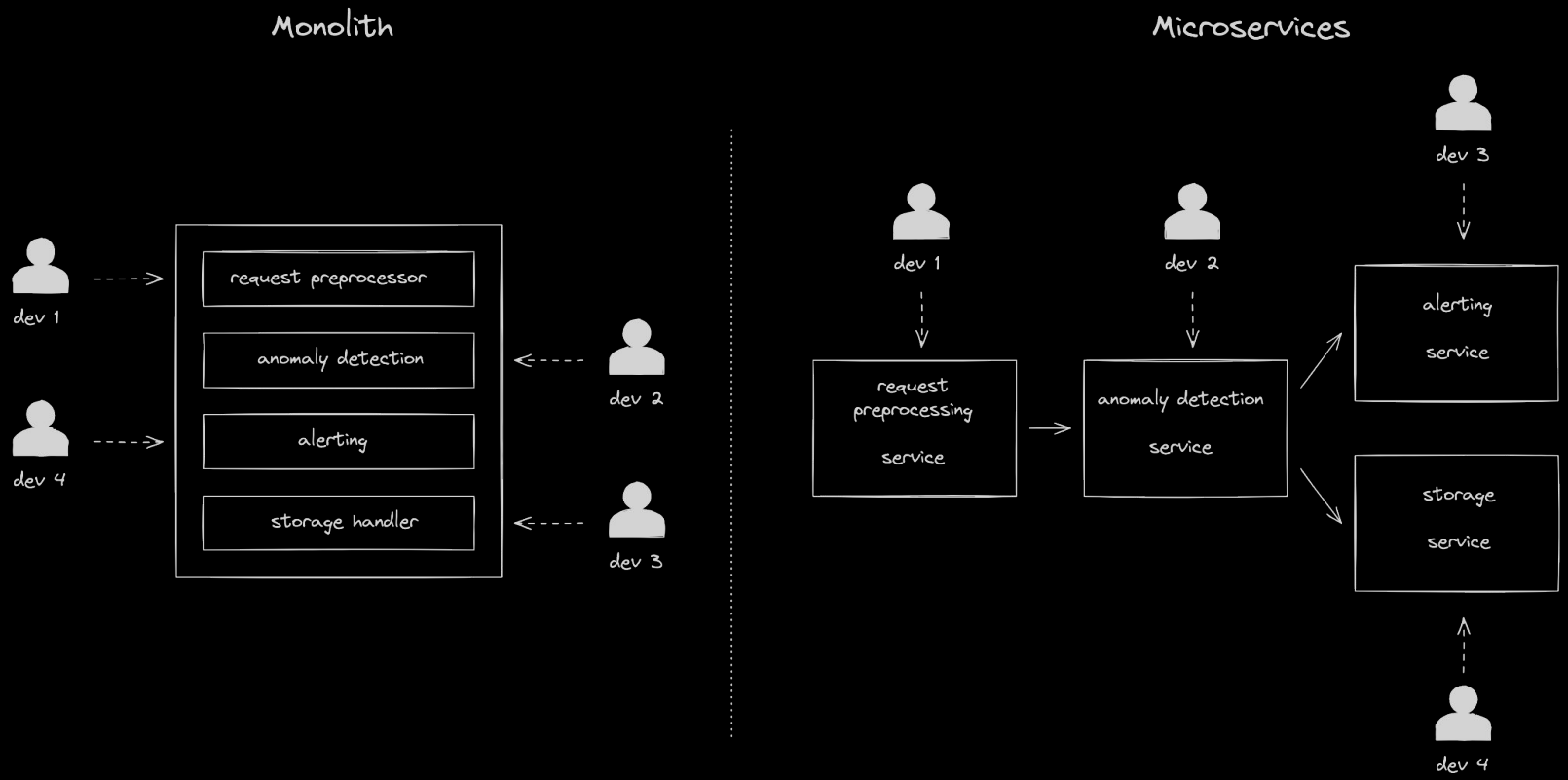


Scalability & Resource Utilization



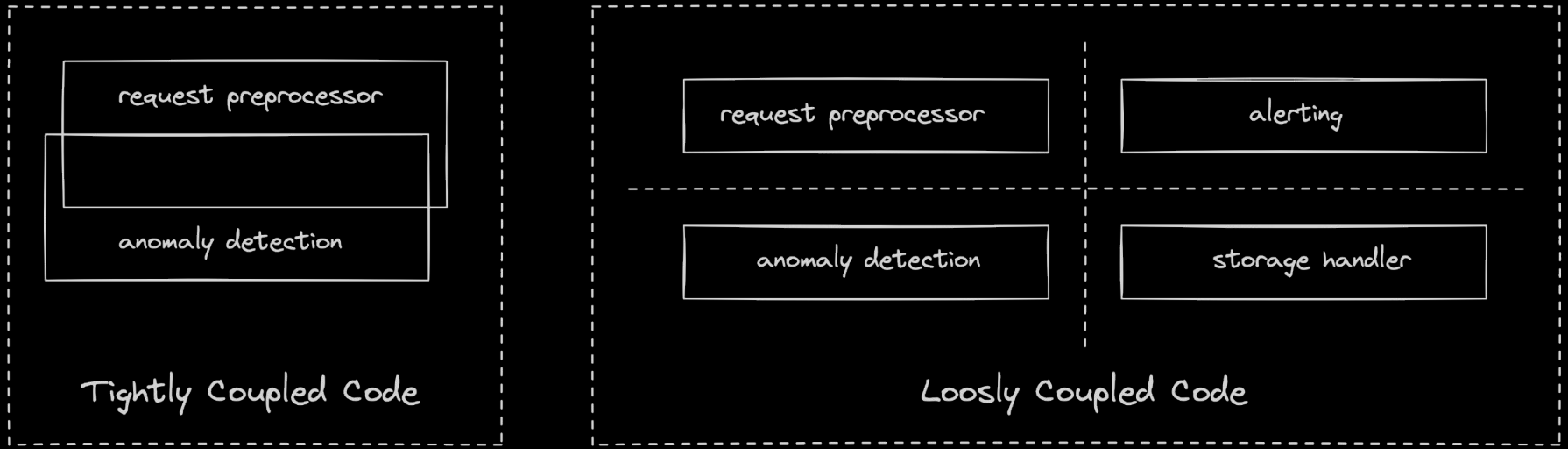
Teamwork makes the dream work

Independent development & ease of maintenance



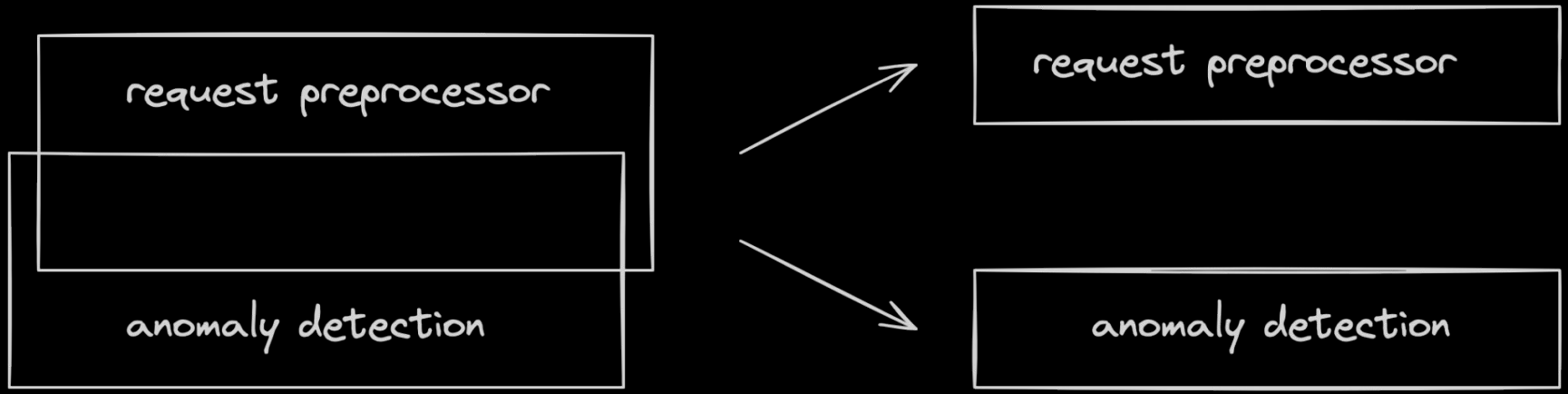
First Day back on the Job after a Holiday

Unwieldy codebases

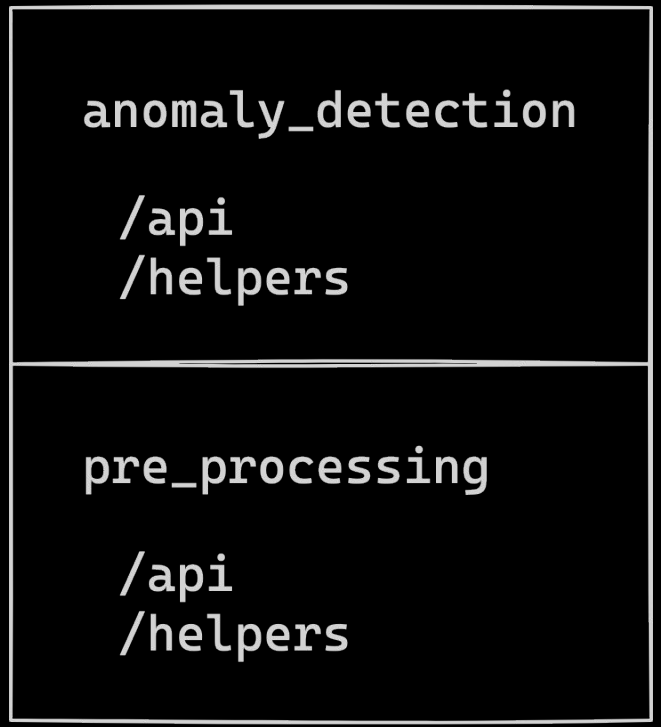


But how can we avoid this?

Think of your code as independent packages



APIs to the Rescue



Conclusion: Monoliths & Microservices

- Less infrastructure to manage
- Better performance
- Easier to run and test

BUT

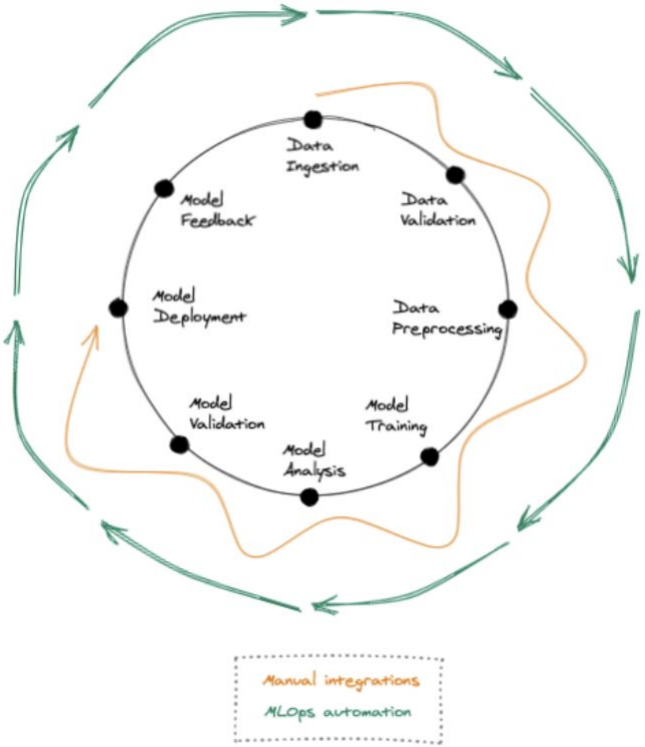
- Require discipline

THERE IS NO ARCHITECTURE TO RULE THEM ALL

Lab:
Docker compose

ML model pipeline

Why do we need ML pipelines?



The general idea is to not treat **ML workflows** as a one-off, but to treat them in a **reliable** and **reproducible** way.

Why do we need ML pipelines?

The Spotify experience

The Winding Road to Better Machine Learning Infrastructure Through Tensorflow Extended and Kubeflow

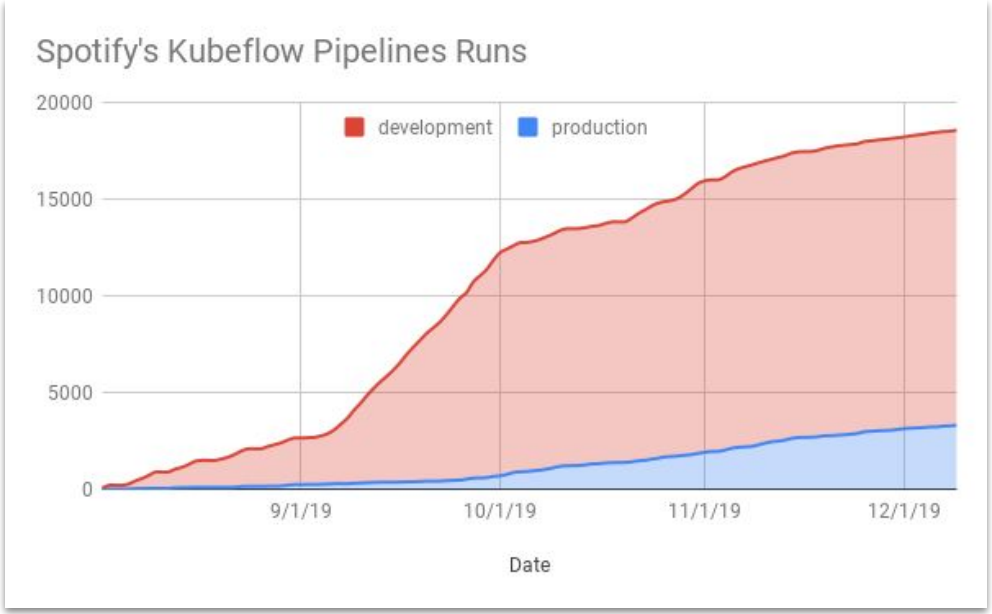
Posted on December 13, 2019 by josh baer and samuelngahane

"As we built these new Machine Learning systems, we started to hit a point where **our engineers spent more of their time maintaining data and backend systems in support of the ML-specific code** than iterating on the model itself. We realized we needed to standardize best practices and build tooling to bridge the gaps between data, backend, and ML: we needed a Machine Learning platform. "

https://labs.spotify.com/2019/12/13/the-winding-road-to-better-machine-learning-infrastructure-through-tensorflow-extended-and-kubeflow/

Why do we need ML pipelines?

Spotify accomplished an astonishing 7x for running machine learning pipelines

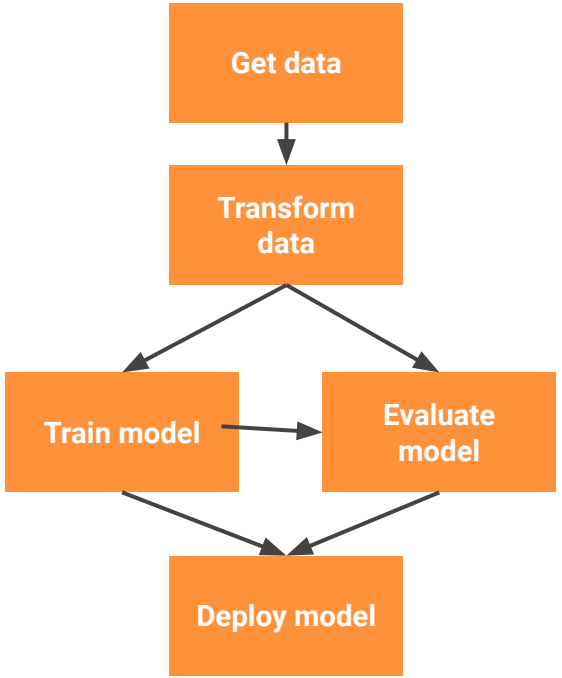


https://labs.spotify.com/2019/12/13/the-winding-road-to-better-machine-learning-infrastructure-through-tensorflow-extended-and-kubeflow/

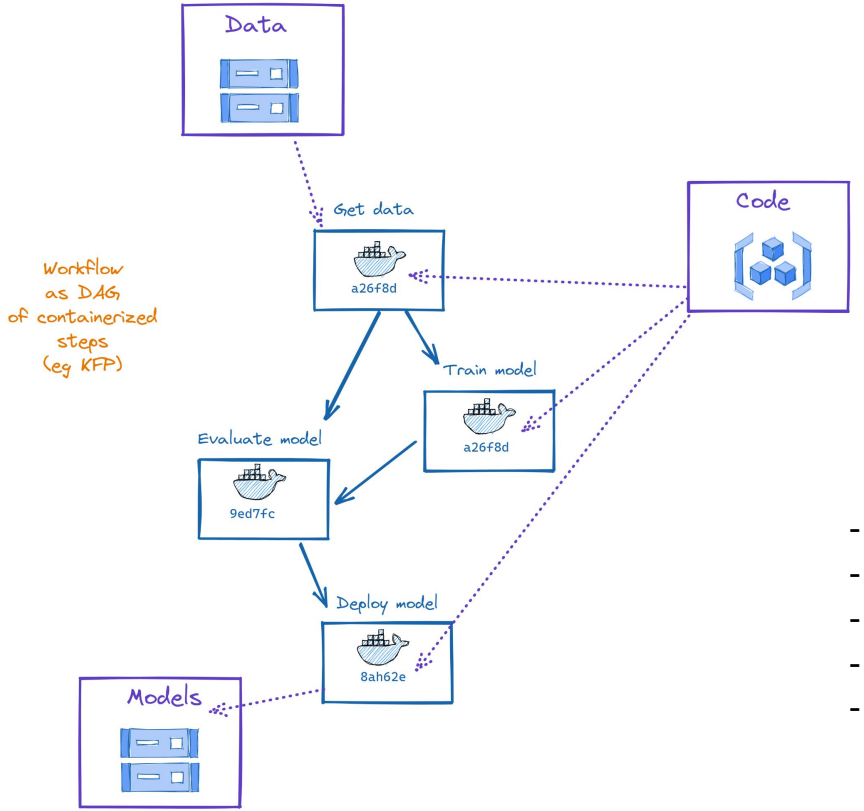
Represent your entire ML workflow as a DAG

Think about your workflow in terms of DAGs (Directed Acyclic Graph):

- What needs to be done sequentially vs in parallel
- Does the step process something itself or call an external service?
- Process first, implementation second



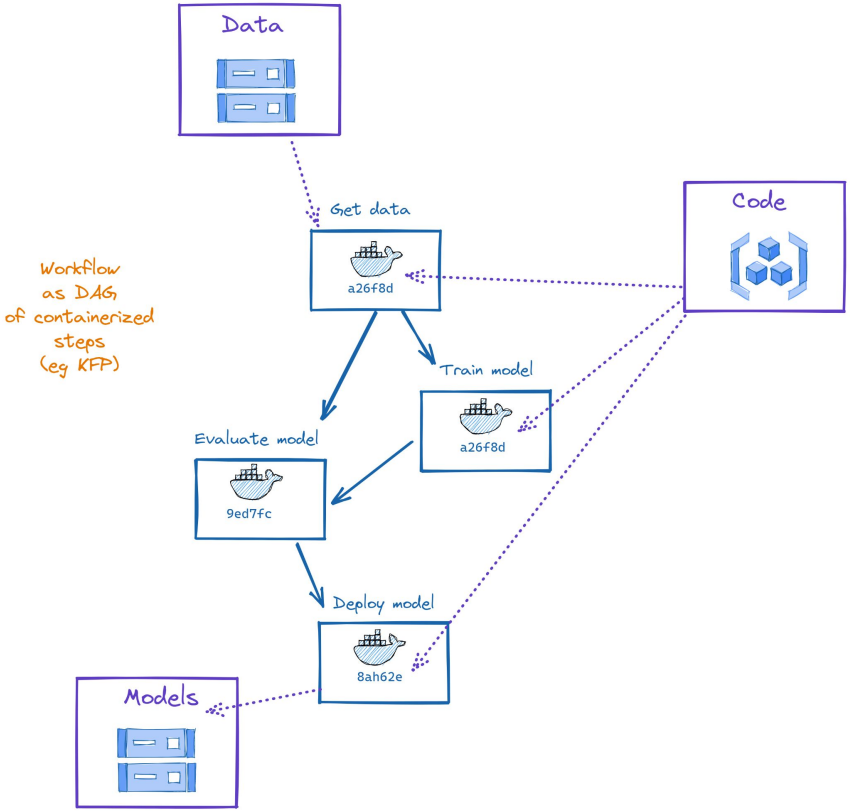
Represent your entire ML workflow as a DAG



- Kubeflow pipelines
- Sagemaker pipelines
- Vertex pipelines
- Valohai
- MLFlow

Definition of a ML Pipeline

"A machine learning pipeline is a series of interconnected data processing and modeling steps designed to automate, standardize and streamline the process of building, training, evaluating and deploying machine learning models."



https://www.ibm.com/topics/machine-learning-pipeline

What are components of an ML pipeline?

A machine learning pipeline is the workflow for a full **machine learning task**. **Components** are the building blocks of a machine learning pipeline.

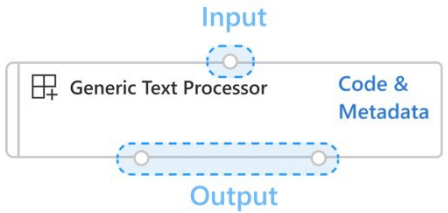
They contain:

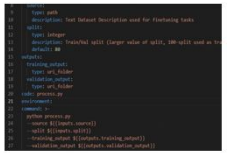
- Metadata
- Interface (input/output specifications)
- Command, Code & Environment.

Build as **Docker images** or as **python functions** (with specific decorators and configurations).

Components

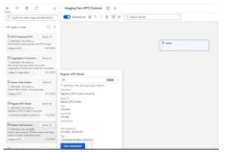
- Metadata
  name, display_name, version, type, etc.
- Interface
  input/output specifications (name, type, description, default value, etc).
- Command, Code & Environment
  command, code and environment required to run the component





CLI

SDK

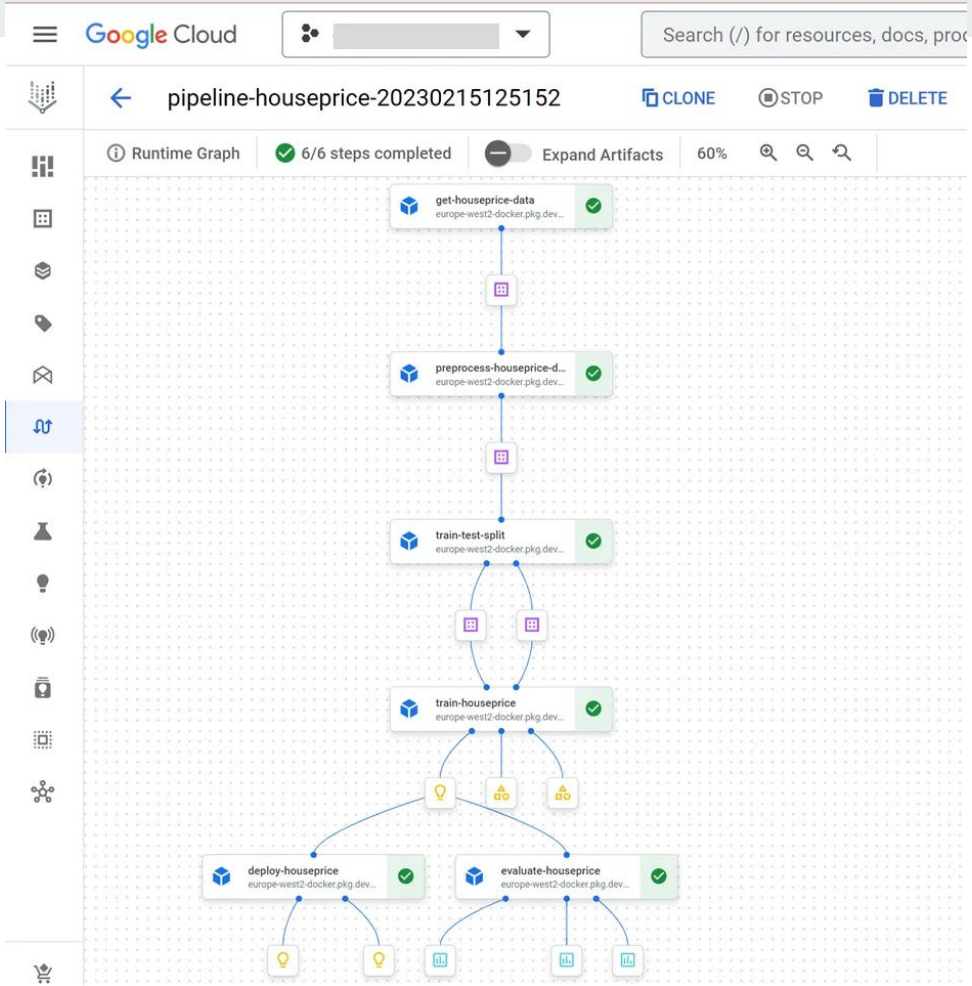


UI

https://learn.microsoft.com/en-us/azure/machine-learning/concept-component?view=azureml-api-2

Example ML Pipeline

Predict house prices with GCP Vertex AI



https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Example ML Pipeline

Predict house prices with GCP Vertex AI

Based on Kubeflow Pipelines (KFP)

```python
# IMPORT REQUIRED LIBRARIES
from kfp.v2 import dsl
from kfp.v2.dsl import (Artifact,
                        Dataset,
                        Input,
                        Model,
                        Output,
                        Metrics,
                        Markdown,
                        HTML,
                        component,
                        OutputPath,
                        InputPath)
from kfp.v2 import compiler
from google.cloud.aiplatform import pipeline_jobs
```

https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Example ML Pipeline

Predict house prices with GCP Vertex AI

Define **components**.

Can be separate **Docker containers** or **python functions**.

Can create a pipeline in one single notebook

```python
@component(
    base_image=BASE_IMAGE,
    output_component_file="get_data.yaml"
)

def get_houseprice_data(
    filepath: str,
    dataset_train: Output[Dataset],
):

    import pandas as pd

    df_train = pd.read_csv(filepath + '/train.csv')

    df_train.to_csv(dataset_train.path, index=False)
```

Data Ingestion component

LIÈGE université

https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Example ML Pipeline

Predict house prices with GCP Vertex AI

```python
@component(
    base_image=BASE_IMAGE,
    output_component_file="preprocessing.yaml"
)

def preprocess_houseprice_data(
    train_df: Input[Dataset],
    dataset_train_preprocessed: Output[Dataset],
):

    import pandas as pd
    from src.data_preprocessing.preprocessing import data_preprocessing_pipeline

    train_df = pd.read_csv(train_df.path)

    # data_preprocessing_pipeline creates a copy of the df, removes id col, converts to
    # subtracts YearSold from temporal features and cosine transforms cyclic features.
    train_df_preprocessed = data_preprocessing_pipeline(train_df)

    train_df_preprocessed.to_csv(dataset_train_preprocessed.path, index=False)
```

Data Preprocessing component

```python
@component(
    base_image=BASE_IMAGE,
    output_component_file="train_test_split.yaml",
)

def train_test_split(dataset_in: Input[Dataset],
                     dataset_train: Output[Dataset],
                     dataset_test: Output[Dataset],
                     test_size: float = 0.2):

    import pandas as pd
    from sklearn.model_selection import train_test_split

    df = pd.read_csv(dataset_in.path)
    df_train, df_test = train_test_split(df, test_size=test_size, random_state=42)

    df_train.to_csv(dataset_train.path, index=False)
    df_test.to_csv(dataset_test.path, index=False)
```

Train test split component

https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Example ML Pipeline

Predict house prices with GCP Vertex AI

...

```python
import pandas as pd
import pickle
import shap
from src.modelling.train import HousePriceModel
from src.utils.utils import get_image_data

TARGET = 'SalePrice'

# Read train and test data
train_data = pd.read_csv(dataset_train.path)
test_data = pd.read_csv(dataset_test.path)

# Instantiate the model class
house_price_model = HousePriceModel(test_data.copy(),   #we perform hyperparameter t
                                    target=TARGET,
                                    n_kfold_splits=3,
                                    n_trials=100,
                                    random_state=42)

# Create X_train and y_train
X_train = train_data.drop(TARGET, axis=1)
y_train = train_data[TARGET]
```

...

Model training component

```python
@component(
    base_image=BASE_IMAGE,
    output_component_file="model_evaluation.yaml"
)
def evaluate_houseprice(
    houseprice_model: Input[Model],
    metrics_baseline: Output[Metrics],
    metrics_train: Output[Metrics],
    metrics_test: Output[Metrics]):

    import pickle

    file_name = houseprice_model.path
    with open(file_name, 'rb') as file:
        model_data = pickle.load(file)

    scores = model_data["scores_dict"]

    def log_metrics(scores, metric):
        for metric_name, val in scores.items():
            metric.log_metric(metric_name, float(val))

    log_metrics(scores["baseline_scores"], metrics_baseline)
    log_metrics(scores["train_scores"], metrics_train)
    log_metrics(scores["test_scores"], metrics_test)
```

Model evaluation component

LIÈGE université

https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Example ML Pipeline

Predict house prices with GCP Vertex AI

```python
from google.cloud import aiplatform as vertex_ai
from pathlib import Path

# Checks existing Vertex AI Enpoint or creates Endpoint if it is not exist.
def create_endpoint ():
    endpoints = vertex_ai.Endpoint.list(
    filter='display_name="{}"'.format(model_endpoint),
    order_by='create_time desc',
    project=gcp_project,
    location=gcp_region,
    )
    if len(endpoints) > 0:
        endpoint = endpoints[0] # most recently created
    else:
        endpoint = vertex_ai.Endpoint.create(
            display_name=model_endpoint,
            project=gcp_project,
            location=gcp_region
    )
    return endpoint
```

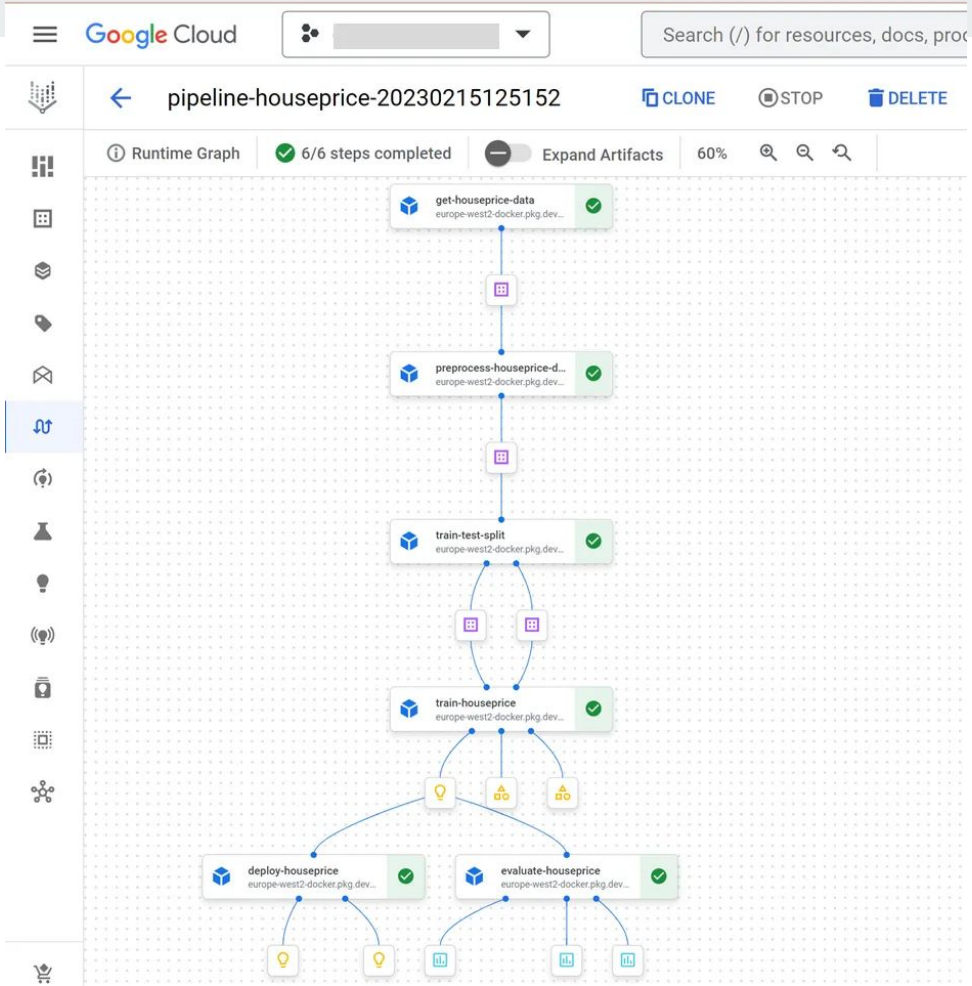
```python
# Uploads trained model to Vertex AI Model Registry or creates new model version in
def upload_model ():
    listed_model = vertex_ai.Model.list(
    filter='display_name="{}"'.format(display_name),
    project=gcp_project,
    location=gcp_region,
    )
    if len(listed_model) > 0:
        model_version = listed_model[0] # most recently created
        model_upload = vertex_ai.Model.upload(
                display_name=display_name,
                parent_model=model_version.resource_name,
                artifact_uri=str(Path(model.path).parent),
                serving_container_image_uri=serving_container_image_uri,
                location=gcp_region,
                serving_container_predict_route="/predict",
                serving_container_health_route="/health"
        )
```

Model deployment component

https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

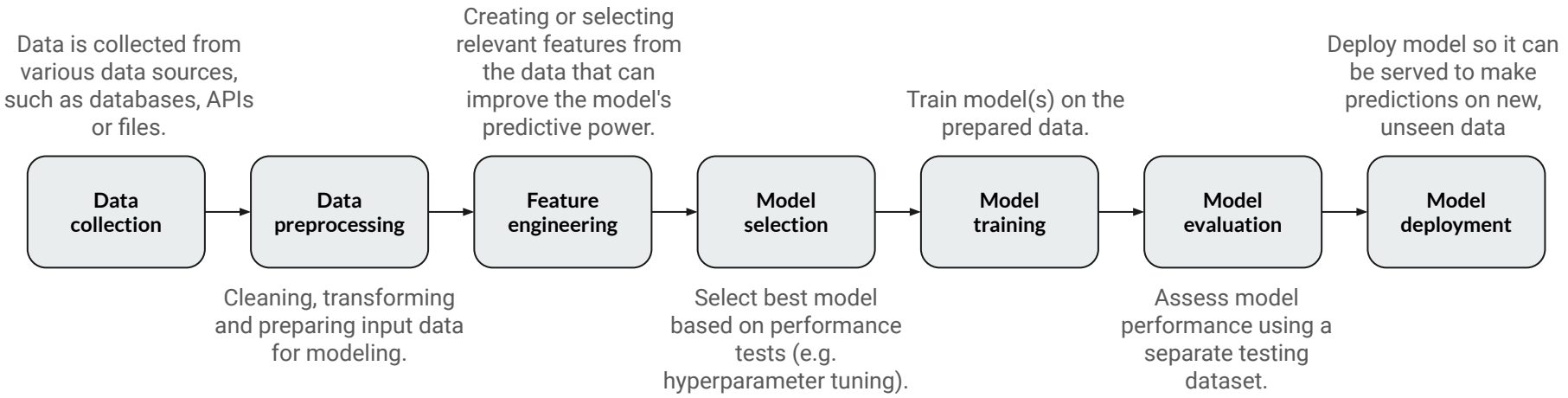
Example ML Pipeline

Predict house prices with GCP Vertex AI



https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Typical components of an ML pipeline



https://www.ibm.com/topics/machine-learning-pipeline

Benefits of an ML Pipeline

- Modularization
- Reproducibility
- Efficiency
- Scalability
- Experimentation
- Deployment
- Collaboration
- Version control and documentation

https://www.ibm.com/topics/machine-learning-pipeline

Avoiding the endless POC loop…

- Let’s change the model architecture
- Let’s try with another model
- Let’s gather more labeled data
- Ah wait, it was the wrong data, here is the new data, can we get results tomorrow?
- What were the accuracies again with the other model?
- Oops not sure what notebook was executed to get that model
- Could we try out the model to get feedback?
- …

*The key to a **successful POC outcome** is to **plan for production** during the proof of concept.*

Triggers

Timed

- **Schedule**: Pipeline runs repetitively in relation to the creation time of the Recurring Run. Set the unit (minutes, hours, etc.) and the scalar that goes with it
- **Cron**: Takes in a cron expression, which is applied on the UTC time.

Event based

- **Manual**: Launch the pipeline manually
- **Triggered**: As part of another service (e.g. use GCP Pub/Sub)

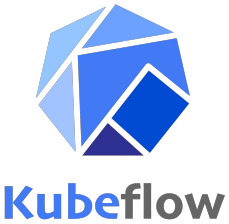
```
# ┌──────────── minute (0–59)
# │ ┌────────── hour (0–23)
# │ │ ┌──────── day of the month (1–31)
# │ │ │ ┌────── month (1–12)
# │ │ │ │ ┌──── day of the week (0–6) (Sunday to Saturday;
# │ │ │ │ │                                7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
```

https://en.wikipedia.org/wiki/Cron

Microservices vs ML Pipeline

	Microservices	ML Pipeline
What is it?	One full application made of different services that are calling each other to serve the overall purpose of the application.	Ran a single workflow made of components that run sequentially.
Is made of…	Services (~= APIs)	Components. Single process that is ran once (e.g. data preparation, model training, evaluation and deployment).
Utilisation	The microservices stay up.	One time run (triggered).
Purpose	All over software engineering.	ML.

ML pipeline platforms



ML pipeline pl

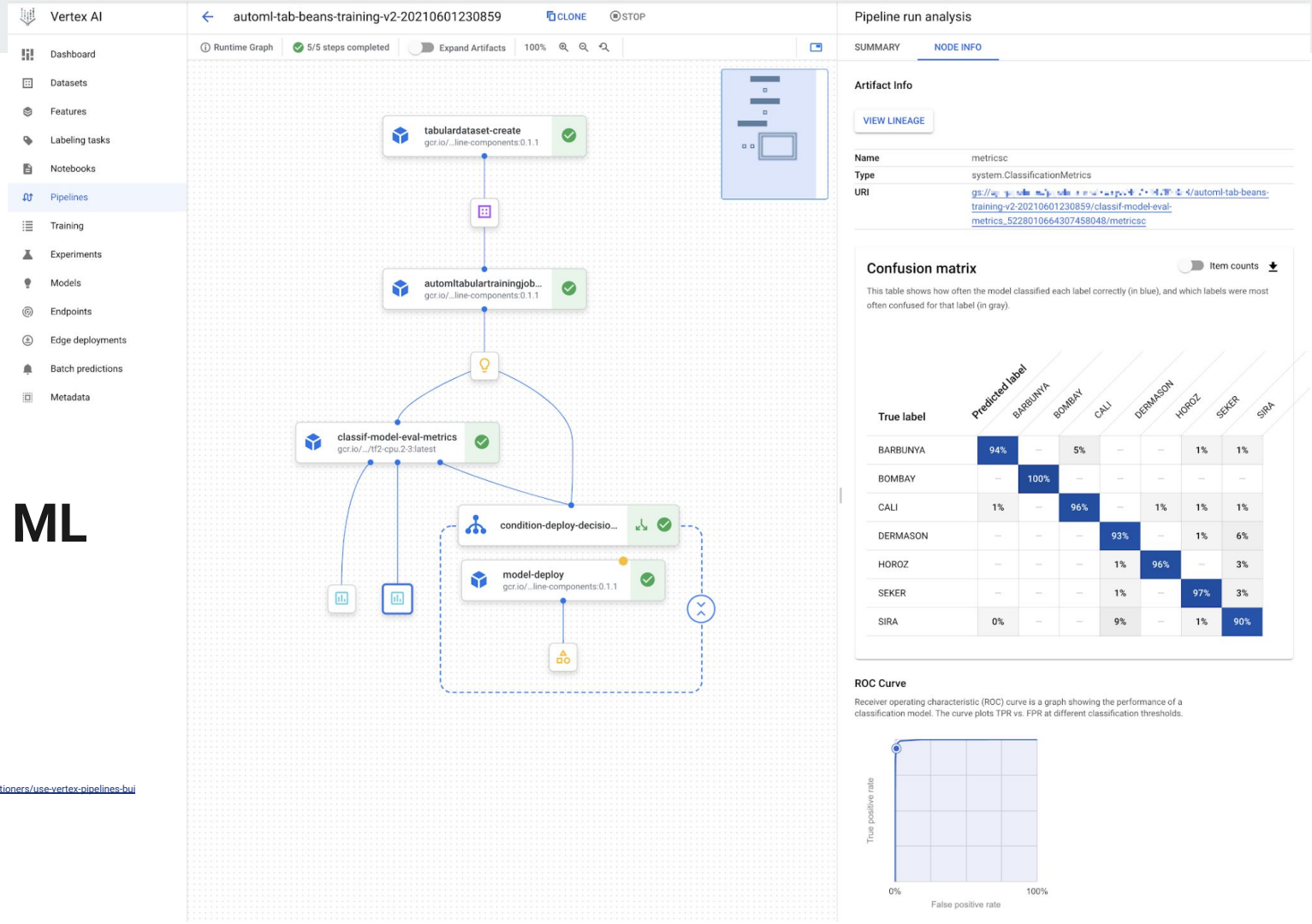
Amazo
SageM

ertex AI

Azure Machine

eflow

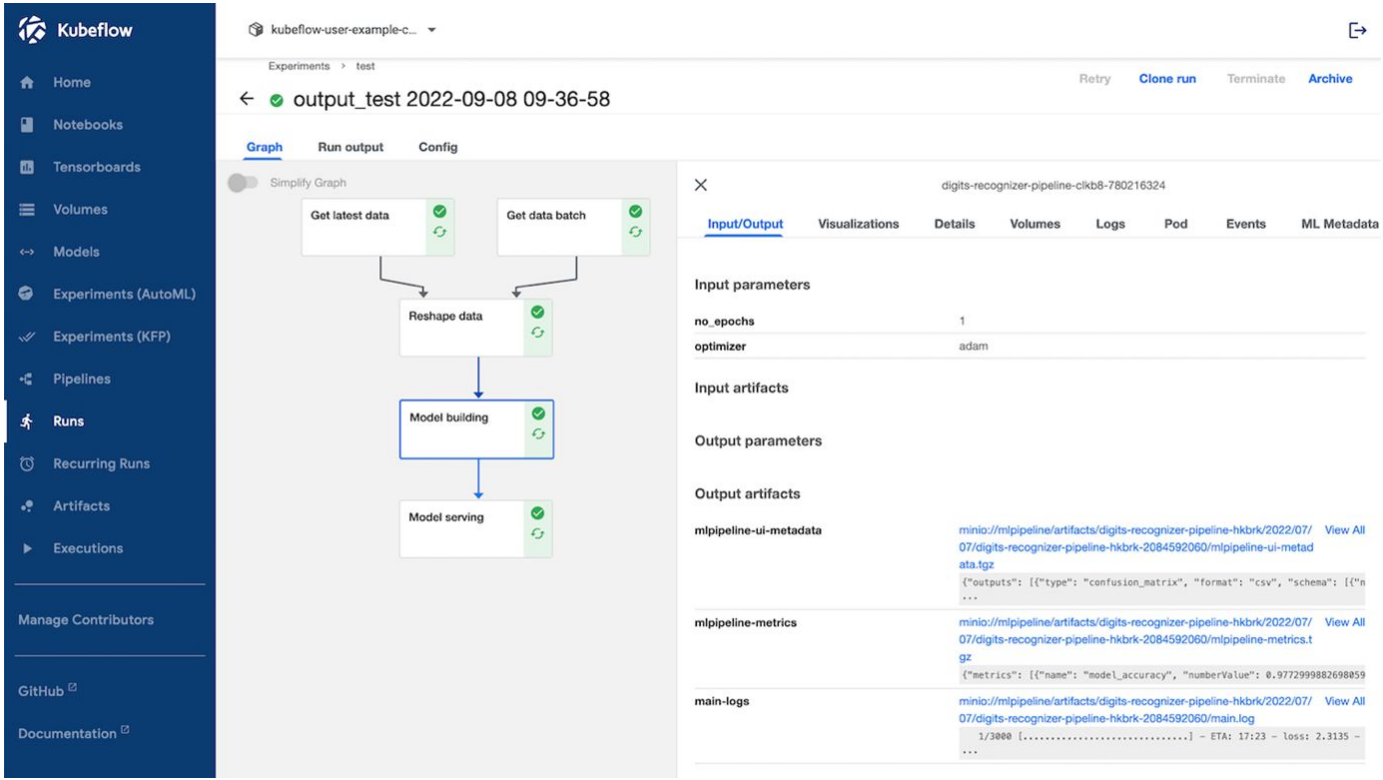
Example ML Pipeline (Vertex)



https://cloud.google.com/blog/topics/developers-practitioners/use-vertex-pipelines-build-automl-classification-end-end-workflow

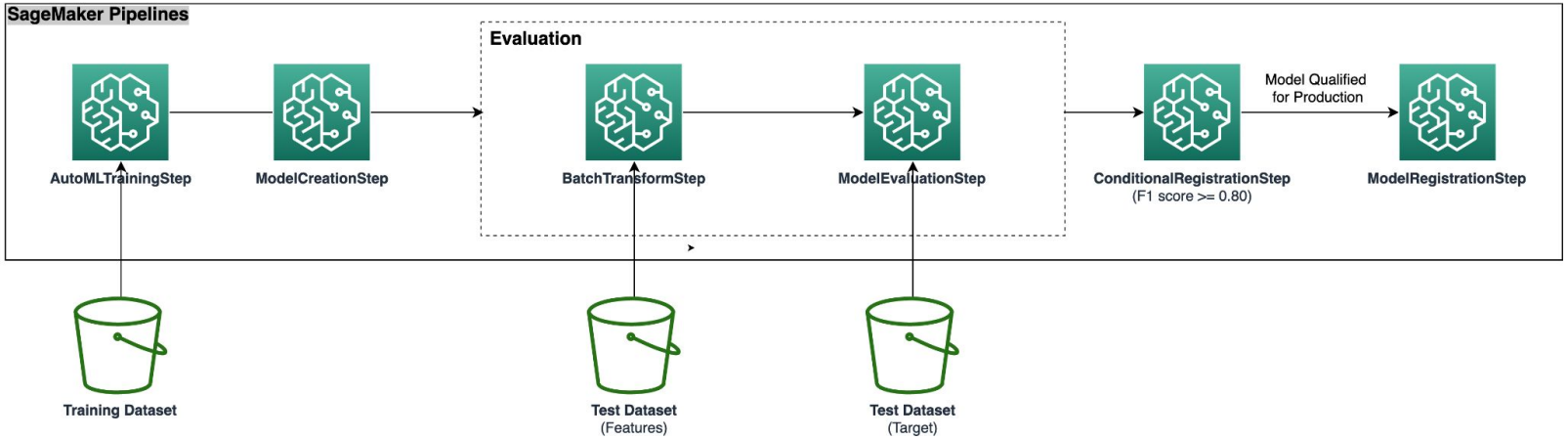
LIÈGE université

Example ML Pipeline (Kubeflow pipeline KFP)



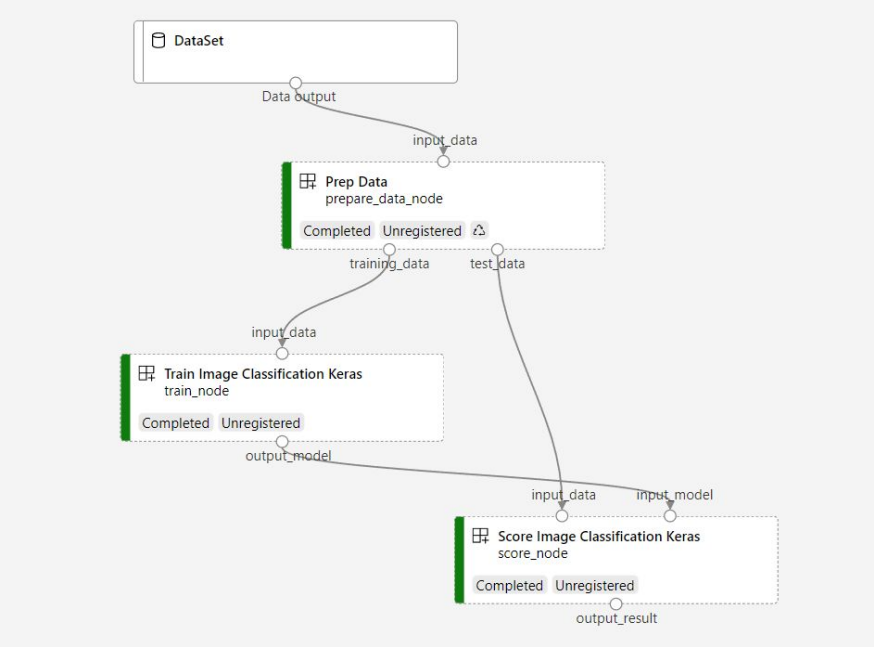
https://learn.microsoft.com/en-us/azure/machine-learning/how-to-create-component-pipeline-python?view=azureml-api-2

Example ML Pipeline (Sagemaker)



https://github.com/aws/amazon-sagemaker-examples/blob/main/sagemaker-pipelines/tabular/automl-step/sagemaker_autopilot_pipelines_native_auto_ml_step.ipynb

Example ML Pipeline (Azure ML)



https://learn.microsoft.com/en-us/azure/machine-learning/how-to-create-component-pipeline-python?view=azureml-api-2

ML Platforms

ML platforms

Feature	AWS SageMaker	Azure Machine Learning Service	Vertex AI
Model training	Supports a wide range of ML frameworks, including TensorFlow, PyTorch, and scikit-learn	Supports a wide range of ML frameworks, including TensorFlow, PyTorch, and scikit-learn	Supports a wide range of ML frameworks, including TensorFlow, PyTorch, and scikit-learn, as well as Google-specific frameworks like TFLite and TPUs
Model deployment	Supports a variety of deployment options, including Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and Amazon SageMaker Real-Time Inference	Supports a variety of deployment options, including Azure Kubernetes Service (AKS), Azure Container Instances (ACI), and Azure App Service	Supports a variety of deployment options, including Google Kubernetes Engine (GKE), Cloud Run, and Cloud TPUs
Model monitoring	Provides tools for monitoring model performance and detecting drift	Provides tools for monitoring model performance and detecting drift	Provides tools for monitoring model performance and detecting drift
Responsible AI	Provides tools for managing bias and fairness in ML models	Provides tools for managing bias and fairness in ML models	Provides tools for managing bias and fairness in ML models
Ease of use	Offers a variety of low-code and no-code tools to make ML accessible to users of all skill levels	Offers a variety of low-code and no-code tools to make ML accessible to users of all skill levels	Offers a variety of low-code and no-code tools to make ML accessible to users of all skill levels

https://www.linkedin.com/pulse/aws-sagemaker-vs-azure-machine-learning-service-ai-ahamed/

ML platforms

Feature	Amazon SageMaker	Google Vertex AI	Azure Machine Learning
Compute	Amazon Elastic Compute Cloud (EC2), Amazon SageMaker managed instances	Google Compute Engine (GCE), Vertex AI managed notebooks	Azure Virtual Machines (VMs), Azure Machine Learning compute clusters
Storage	Amazon Simple Storage Service (S3)	Google Cloud Storage	Azure Blob Storage
Databases	Amazon Relational Database Service (RDS), Amazon SageMaker managed database	Cloud SQL, Vertex AI managed database	Azure SQL Database, Azure Machine Learning compute clusters
Networking	Amazon Virtual Private Cloud (VPC)	Google Cloud Virtual Private Cloud (VPC)	Azure Virtual Network (VNet)
Machine learning	Amazon SageMaker, Amazon SageMaker Canvas, Amazon SageMaker Autopilot	Vertex AI	Azure Machine Learning Studio, Azure Machine Learnin notebooks
Containers	Amazon Elastic Container Service (ECS)	Google Kubernetes Engine (GKE)	Azure Kubernetes Service (AKS)
Serverless computing	AWS Lambda	Cloud Functions	Azure Functions
Big data	Amazon Elastic MapReduce (EMR), Amazon SageMaker managed hosting	Cloud Dataproc, Vertex AI managed pipelines	Azure HDInsight, Azure Machine Learning compute clusters
Analytics	Amazon Redshift, Amazon SageMaker managed analytics	BigQuery, Vertex AI managed analytics	Azure Synapse Analytics, Azure Machine Learning compute clusters

https://www.linkedin.com/pulse/aws-sagemaker-vs-azure-machine-learning-service-ai-ahamed/

Sagemaker

Amazon SageMaker is an automated platform and comprehensive suite of tools that simplifies the development, training and deployment of machine learning (ML) models. It reduces the complexity of model development by providing a web-based interface for creating ML pipelines and pre-built algorithms.

Key features:

1. Notebooks
2. Training Job
3. Model registry
4. Prediction
5. Pipelines

https://www.missioncloud.com/blog/what-is-sagemaker-and-how-does-it-help-mlops

Sagemaker: Notebooks

An Amazon SageMaker notebook instance is a machine learning (ML) compute instance running the Jupyter Notebook App. SageMaker manages creating the instance and related resources. Use Jupyter notebooks in your notebook instance to prepare and process data, write code to train models, deploy models to SageMaker hosting, and test or validate your models.

Advantages:

- Access more compute (GPUs)
- Access more memory
- Manage access
- Collaborate as a team
- No use of data locally

Equivalent

Vertex Workbench

Azure Notebooks

https://docs.aws.amazon.com/sagemaker/latest/dg/gs-setup-working-env.html

Sagemaker: Training Job

Managed way of launching a model training

1. Either be fully specify the ML algorithm, hyperparameters and input data location.
2. Or customize training job by selecting specific Sagemaker instance types and adding software libraries.

Advantages:

- Access more compute (GPUs)
- Access more memory
- Consistency
- Automation

Equivalent

Vertex Training

Azure ML Training

https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html

Sagemaker: Model registry

SageMaker Model Registry is a service for packaging model artifacts with deployment information. That information includes your deployment code, what type of container to use and what type of instance to deploy. The Model Registry decreases time to deployment, as you already have the necessary information about how the model should be deployed.

Advantages:

- Consistency
- Automation
- Scalability
- Time to deployment

Equivalent

Vertex AI Model Registry

Azure ML Model Registry

https://www.missioncloud.com/blog/what-is-sagemaker-and-how-does-it-help-mlops

Sagemaker: Prediction

After you've trained and deployed your model in SageMaker, you can use it to generate predictions based on new data. There are two main ways to do this.

- Endpoint deployment: Deploy your model to an endpoint, allowing users and applications to send API requests to get model predictions in real time.
- Batch predictions: Generate predictions on large amounts of data without needing an immediate response..

Advantages:

- Consistency
- Automation
- Scalability
- Time to deployment

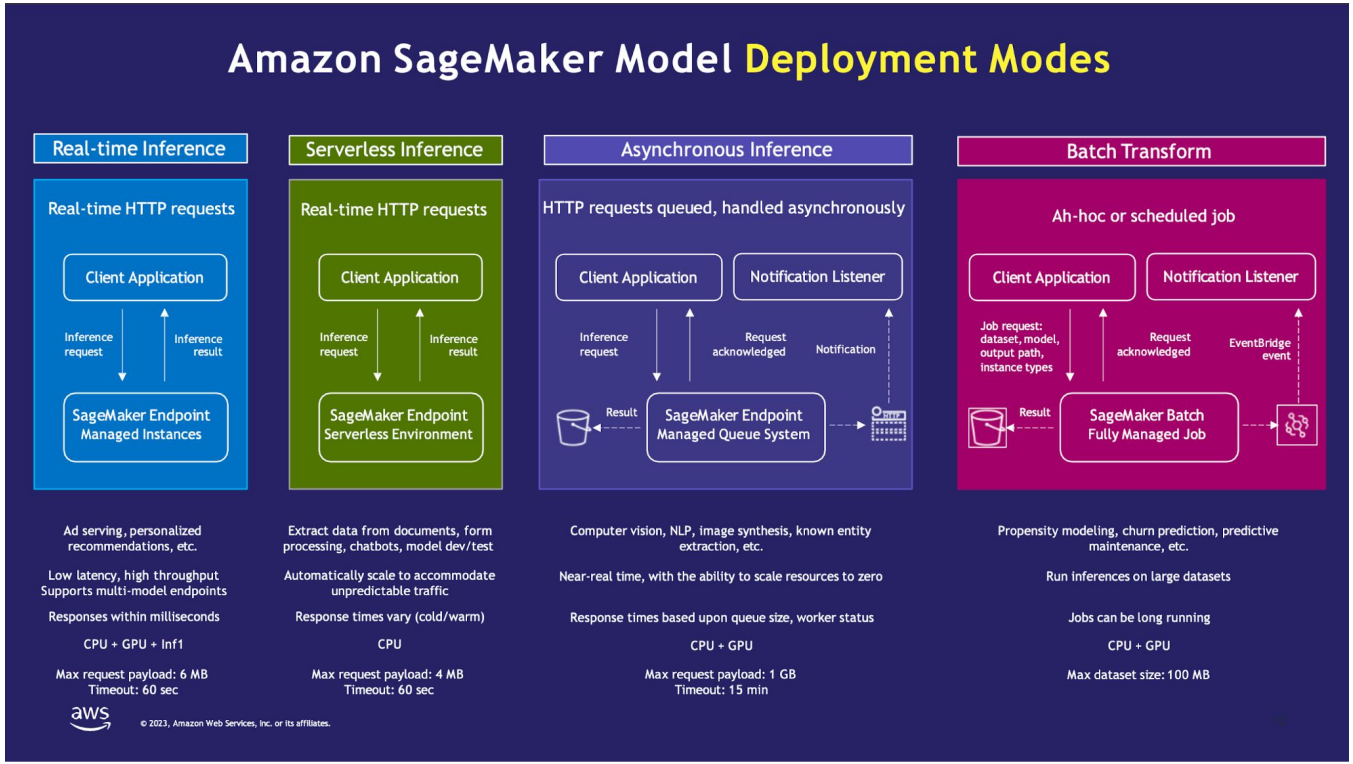
Equivalent

Vertex AI Prediction

Azure ML Prediction

https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html

Sagemaker: Prediction



LIÈGE université

Sagemaker: Pipeline

SageMaker Pipelines is a tool for building, deploying and managing end-to-end ML workflows. Users can create automated workflows that cover data preparation, model training and deployment — all from a single interface. Because Pipeline logs everything, you can easily track and re-create models.

Advantages:

- Resource consumption
- Consistency
- Automation
- Scalability
- Time to deployment

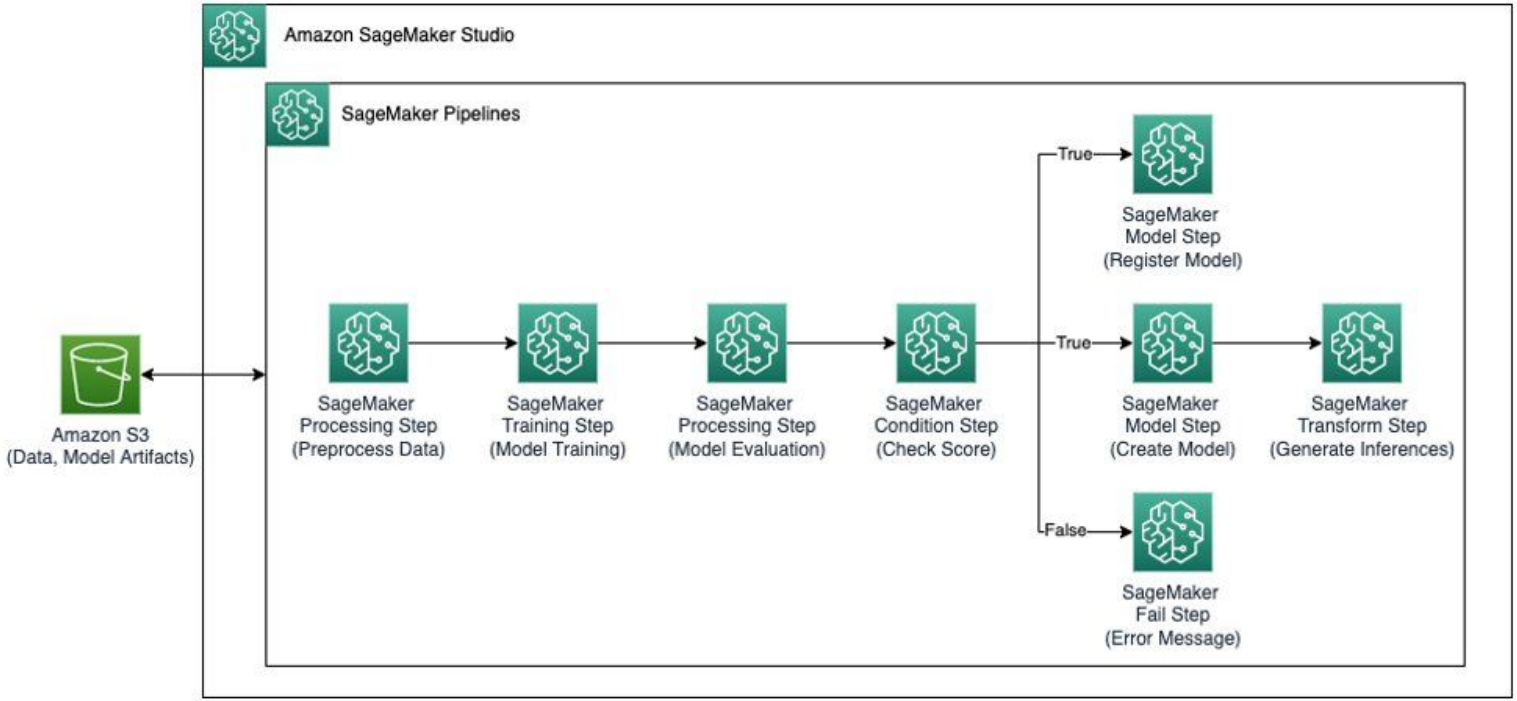
Equivalent

Vertex AI Pipeline

Azure ML Pipeline

https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html

Sagemaker: Pipeline (example)



https://mkai.org/new-features-for-amazon-sagemaker-pipelines-and-the-amazon-sagemaker-sdk/

Wrap-up

Lecture summary

Topic	Concepts	To know for…	
		Project	Exam
Microservice architecture	• What is a Microservice ?		Yes
	• Abstractions & Separation of concerns		Yes
	• The anatomy of a microservice		
	• Monoliths aren't for dinosaurs		
Lab: Docker Compose	• Docker Compose	Optional	
ML Model pipeline	• What and why • Standard steps		Yes
Pipeline orchestrators	• Different options • Sagemaker solutions	Optional	

Project objective for sprint 4

Only optional work packages this sprint. Catch up work from last sprint.

⚠️ **Last two steps can incur cloud costs! Make sure to not use anything you will be charged for.** ⚠️

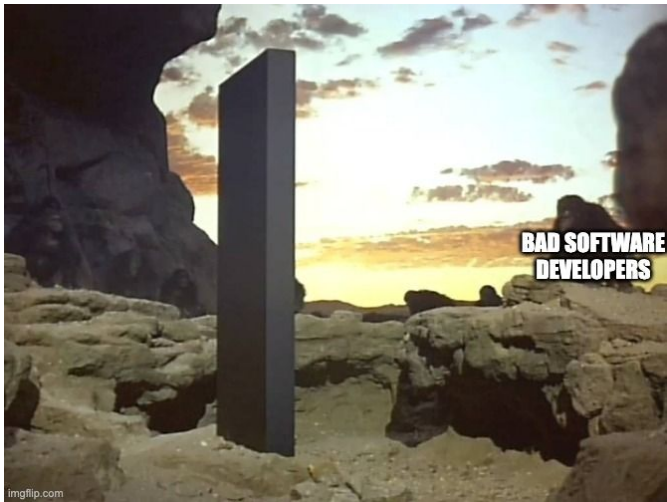
#	Week	Work package	Required
4.1	W07	Package your **model training script** in a **Docker container**. You should be able to run it locally.	Optional
4.2	W07	Run your model training as a job in the Cloud. You can implement this in different ways: ● Containerise your training script and run it on a VM in the Cloud (e.g. on EC2 or on Cloud Run, example,) ● Use a managed service such as Vertex Training or Sagemaker Training	Optional
4.3	W08	Build a **pipeline** to automatically run different sequential components such as training your model and deploying your model. For it you can use orchestrated pipeline tools such as Kubeflow Pipelines, AWS Sagemaker, GCP Vertex.	Optional

APPENDIX

LIÈGE université

Monolith architecture

A monolithic architecture is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications. The word “monolith” is often attributed to something large and glacial, which isn’t far from the truth of a monolith architecture for software design. A monolithic architecture is a singular, large computing network with one code base that couples all of the business concerns together. To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming.



https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

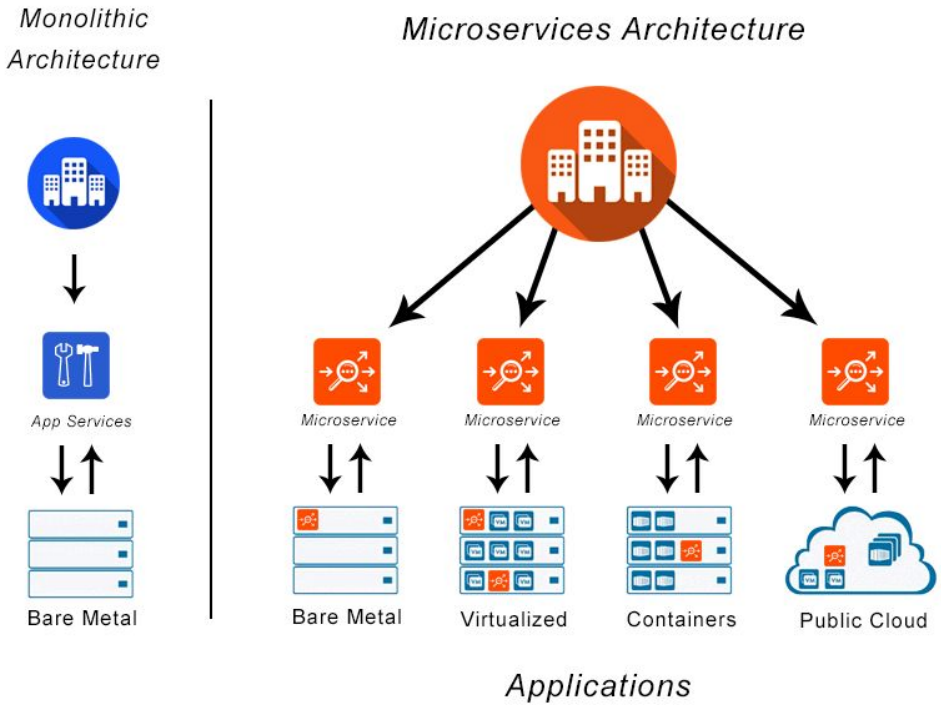
Monolith architecture

Pros	Cons
• Easy deployment – One executable file or directory makes deployment easier. • Development – When an application is built with one code base, it is easier to develop. • Performance – In a centralized code base and repository, one API can often perform the same function that numerous APIs perform with microservices. • Simplified testing – Since a monolithic application is a single, centralized unit, end-to-end testing can be performed faster than with a distributed application. • Easy debugging – With all code located in one place, it's easier to follow a request and find an issue.	• Slower development speed – A large, monolithic application makes development more complex and slower. • Scalability – You can't scale individual components. • Reliability – If there's an error in any module, it could affect the entire application's availability. • Barrier to technology adoption – Any changes in the framework or language affects the entire application, making changes often expensive and time-consuming. • Lack of flexibility – A monolith is constrained by the technologies already used in the monolith. • Deployment – A small change to a monolithic application requires the redeployment of the entire monolith.

https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

Microservice architecture

A microservices architecture, also simply known as microservices, is an architectural method that relies on a series of independently deployable services. These services have their own business logic and database with a specific goal. Updating, testing, deployment, and scaling occur within each service. Microservices decouple major business, domain-specific concerns into separate, independent code bases. Microservices don't reduce complexity, but they make any complexity visible and more manageable by separating tasks into smaller processes that function independently of each other and contribute to the overall whole.

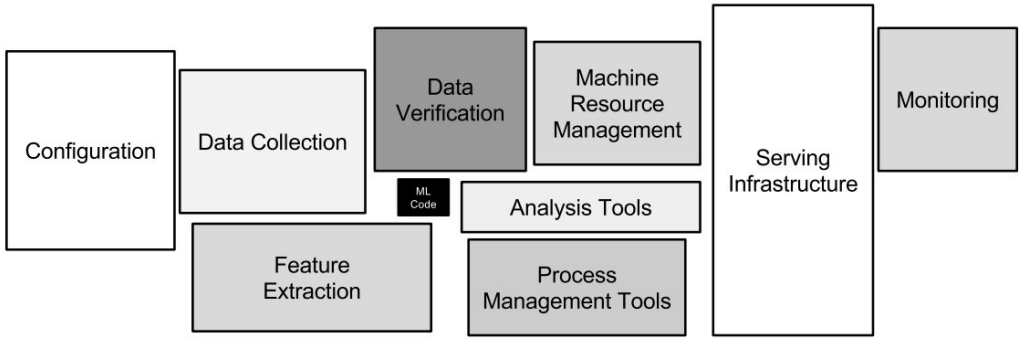


https://www.tatvasoft.com/blog/monolithic-vs-microservices/

Microservice architecture

Pros	Cons
● Agility – Promote agile ways of working with small teams that deploy frequently. ● Flexible scaling – If a microservice reaches its load capacity, new instances of that service can rapidly be deployed to the accompanying cluster to help relieve pressure. We are now multi-tenanant and stateless with customers spread across multiple instances. Now we can support much larger instance sizes. ● Continuous deployment – We now have frequent and faster release cycles. Before we would push out updates once a week and now we can do so about two to three times a day. ● Highly maintainable and testable – Teams can experiment with new features and roll back if something doesn't work. This makes it easier to update code and accelerates time-to-market for new features. Plus, it is easy to isolate and fix faults and bugs in individual services. ● Independently deployable – Since microservices are individual units they allow for fast and easy independent deployment of individual features.	● Development sprawl – Microservices add more complexity compared to a monolith architecture, since there are more services in more places created by multiple teams. If development sprawl isn't properly managed, it results in slower development speed and poor operational performance. ● Exponential infrastructure costs – Each new microservice can have its own cost for test suite, deployment playbooks, hosting infrastructure, monitoring tools, and more. ● Added organizational overhead – Teams need to add another level of communication and collaboration to coordinate updates and interfaces. ● Debugging challenges – Each microservice has its own set of logs, which makes debugging more complicated. Plus, a single business process can run across multiple machines, further complicating debugging. ● Lack of standardization – Without a common platform, there can be a proliferation of languages, logging standards, and monitoring. ● Lack of clear ownership – As more services are introduced, so are the number of teams running those services. Over

Model “training” is just a part of producing an ML model



Add information from Harvard lecture

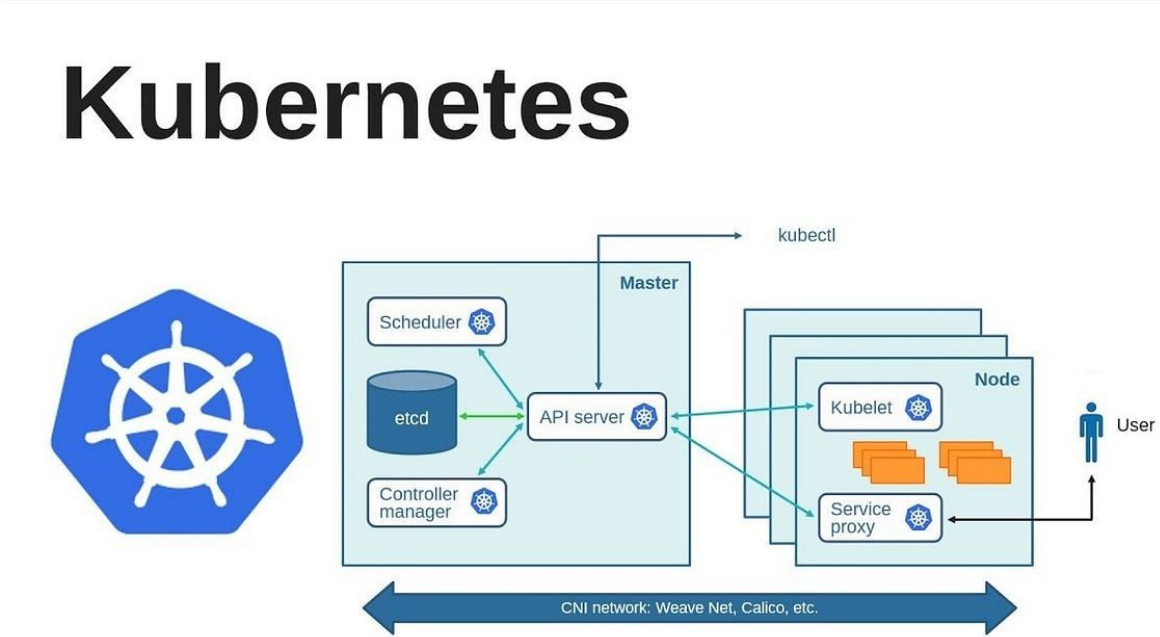
Lecture 2:
https://github.com/Harvard-IACS/2020F-AC295/blob/master/content/lectures/lecture2/presentation/lecture2.pptx

Kubernetes

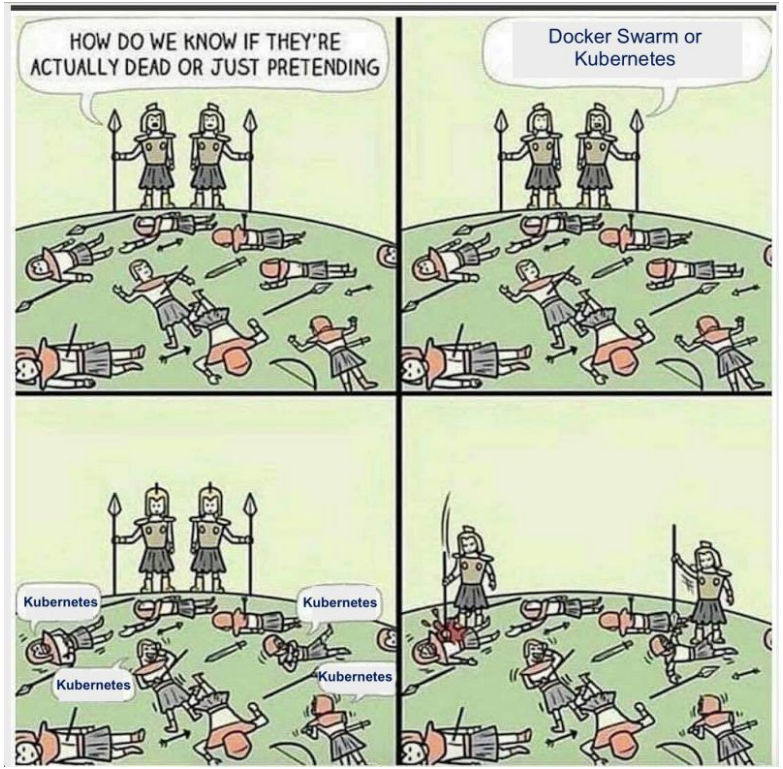
Kubernetes (k8s)

- An open source platform that has become the standard for container orchestration
  - also known as **K8s**
- Declarative API to run applications in a group of Docker hosts, called a **Kubernetes cluster**
  - Users specify in a configuration file the desired state, and Kubernetes makes it happen and then maintains it
  - “make sure four instances of this container run at all times” and Kubernetes will allocate the hosts, start the containers, monitor them, and start a new instance if one of them fails
- Major cloud providers all provide managed Kubernetes services
  - Users do not have to install and maintain Kubernetes itself
  - If an application or a model is packaged as a Docker container, users can directly submit it, and the service will provision the required machines to run one or several instances of the container inside Kubernetes

Kubernetes (k8s)

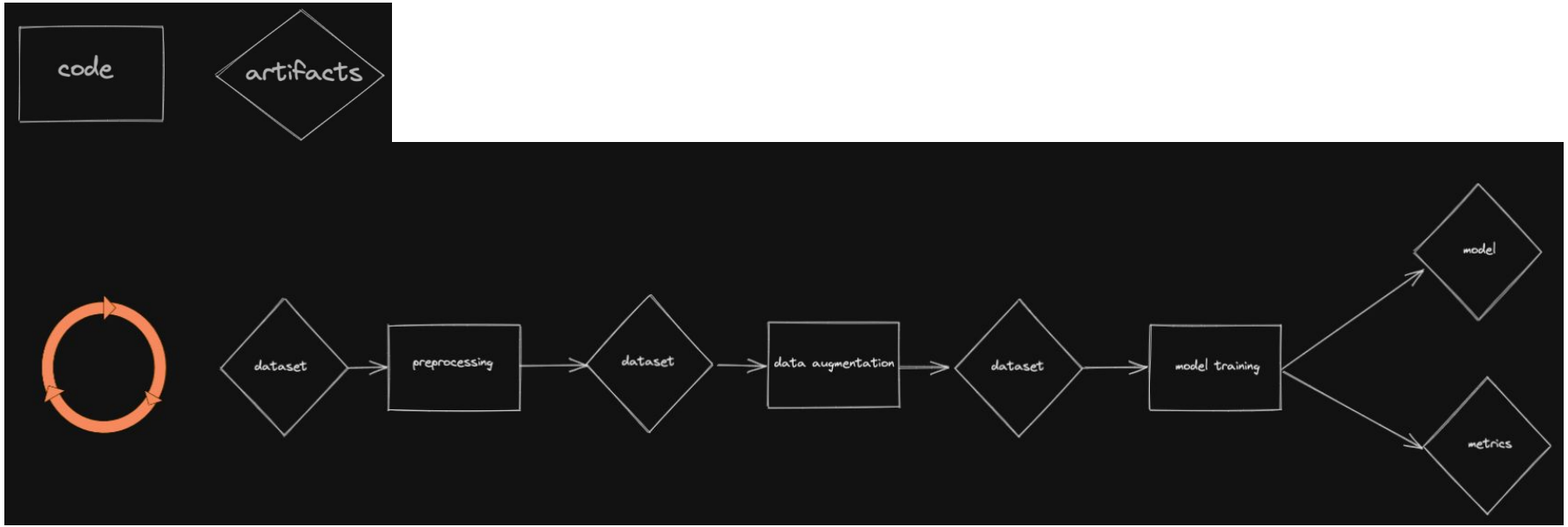


https://medium.com/@shubhamdeshmukh8531/basics-of-kubernetes-architecture-4ed4110caf70



https://www.linkedin.com/posts/itq_laten-we-jouw-droombaan-cre%C3%ABren-activity-7113521986332413952-L_Yk?utm_source=share&utm_medium=member_desktop

ML Pipelines are made of components and iterable



[Example]

Miro’s sagemaker pipeline thingy:
https://blog.ml6.eu/promoting-code-across-environments-for-reliable-and-efficient-model-deployment-1cbe3d970b4f

Why do we need an automated model training pipeline?

A machine learning pipeline is a step-by-step workflow for developing and deploying machine learning models into production. These pipelines allow you to streamline the process of taking raw data, training ML models, evaluating performance and integrating predictions into business applications.

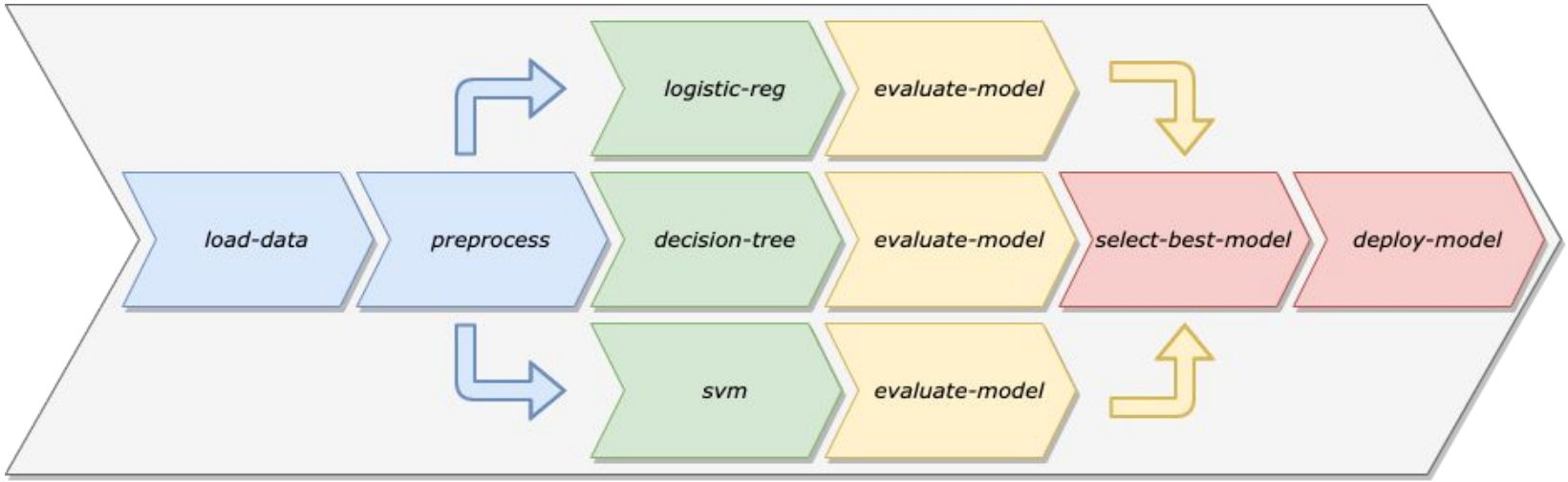
https://www.akkio.com/post/what-are-machine-learning-pipelines-and-why-are-they-important

Example: Customer churn pipeline

1. **Organize your customer data**. This could involve things like accessing customer data from Salesforce, cleaning it, and pre-processing it in a way that can be used by your machine learning models.
2. **Train your models**. This step involves using your data to train one or more machine learning models, which, in this case, could be used to predict customer churn. A lot of computing power is needed, typically provided by platforms like AWS and Azure.
3. **Evaluate your models**. Once you've trained your models, it's important to evaluate their performance to see how well they are actually doing. This will help you fine-tune your models and make sure they are as effective as possible.
4. **Deploy your models.** After you've evaluated and fine-tuned your models, you can deploy them in production so they can start making predictions (in this case, predictions about customer churn).

https://www.akkio.com/post/what-are-machine-learning-pipelines-and-why-are-they-important

There are steps in ML pipeline



Standard steps of a ML pipeline

1. Data collection
   a. Coming from different sources
   b. Different formats…
2. Data preparation
   a. Clean data (remove duplicates, normalise, …)
   b. Split data
3. Feature engineering
   a. Prepare your features
4. Model training
   a. Train the model
5. Optimisation
   a. Run some hyperparameter optimisation
6. Evaluation
   a. Compute and store performance metrics
7. Model deployment
   a. Store model in a registry
   b. Deploy model to make inferance
8. Predict
   a. Sometimes compute live predictions (e.g. time series data)

https://www.akkio.com/post/what-are-machine-learning-pipelines-and-why-are-they-important