Experian Interview

The Engineering Challenge

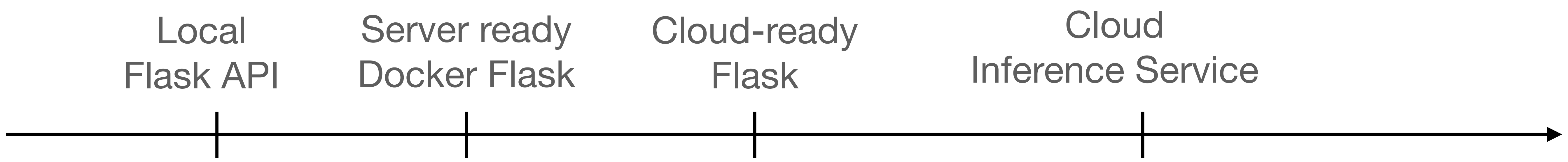
I-chun Han June 2024

Outline

Three main topics

1

The inference pipeline design process with two proposals



2

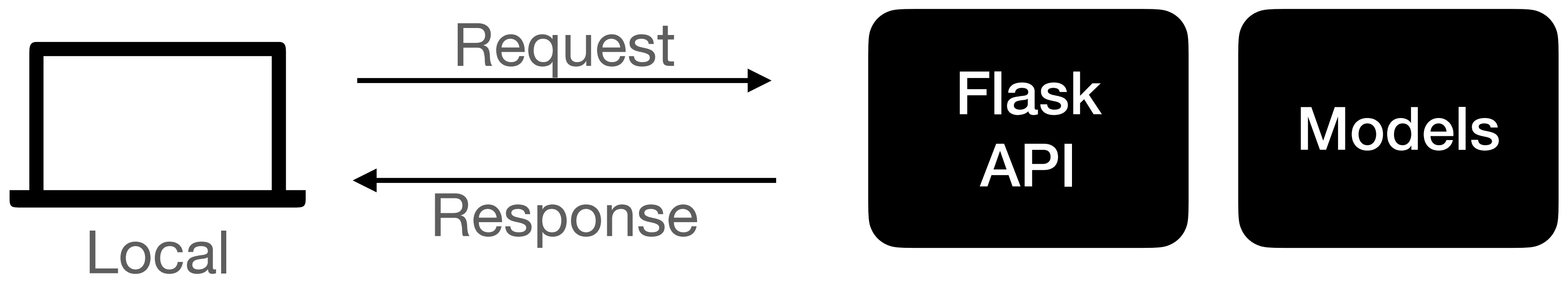
A training pipeline design with cloud platforms

3

The integration plan of inference and training pipelines

Inference Pipeline — v0.0

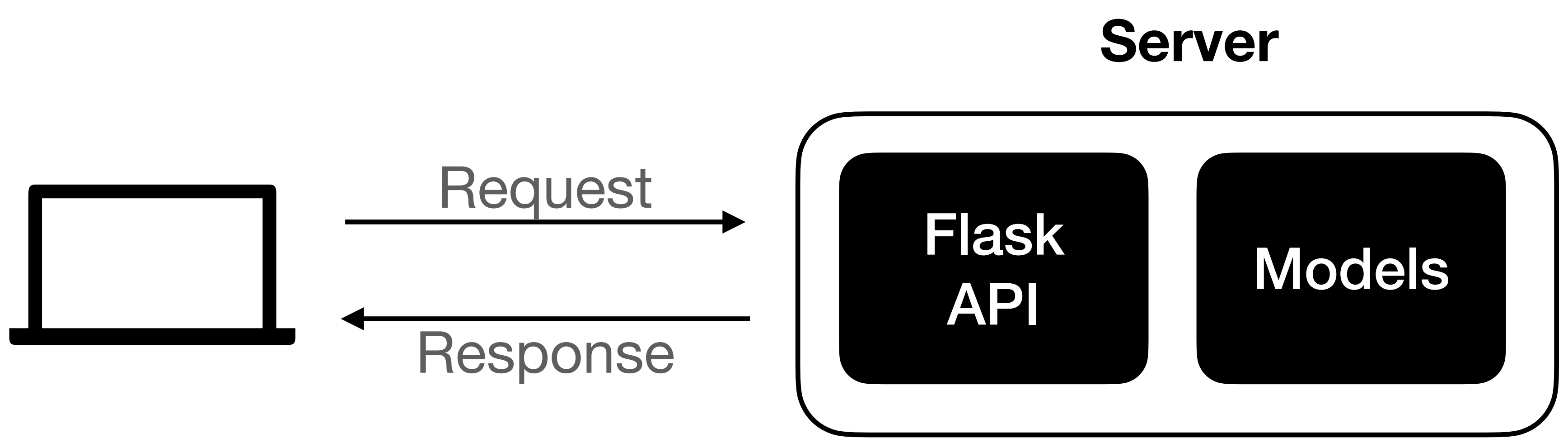
Start with a prototype



- Build Python Inference Lib with SOLID principles and wrap it up with Flask API.
- Design to handle a list of clients in one API call.
- It is easy to update the local model monthly.
- Assuming it is an online inference use case given it was designed as a REST API.
- Assuming “client_id” is in the request payload and is used as the identifier in response.

Inference Pipeline — v0.1

Start with a naive approach



The prototype is ready, and I just need to:

- Deploy to a server or a cloud platform
- Binding customised domain

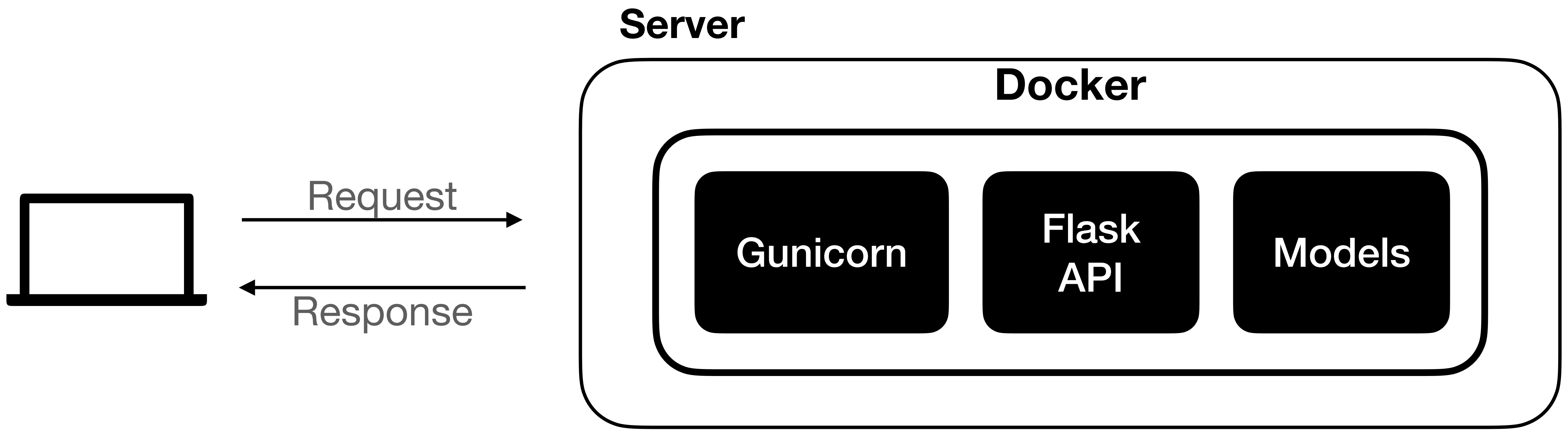
It's ready for external users for small usage.

Wait…talking about deploying to a server. It's a bit messy to use a Python repo to deploy.

- Using Docker could make things easier.
- Adding Gunicorn to make it more robust to the increasing traffic is a good idea. The Flask native is not suitable for production.

Inference Pipeline — v1.0

Add Gunicorn and Docker



The version can:

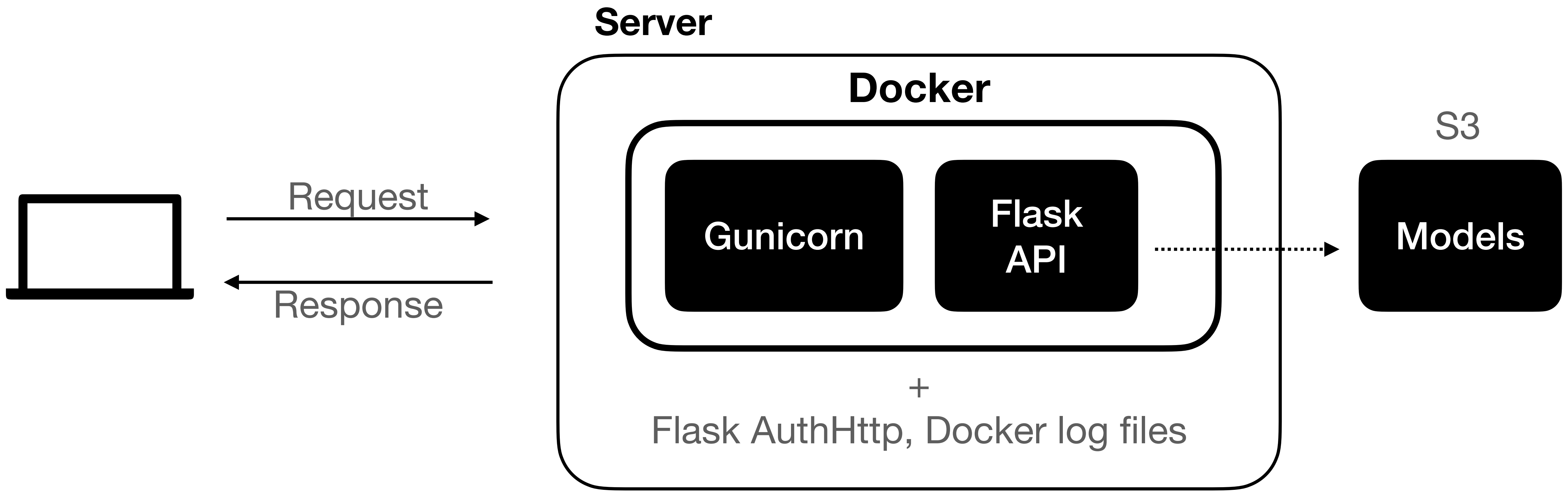
- It can handle a bit more traffic with Gunicorn and
- It is easier to deploy to the server. I just need to run this docker image. Perfect!

There is a downside to it… updating the model is not so easy now! It should be updated monthly.

I should remove the model from Docker.

Inference Pipeline — v1.1

Fetch a remote model



- Assume the model is stored in cloud storage. I take AWS S3 as an example here.

We can update the model without touching the docker.

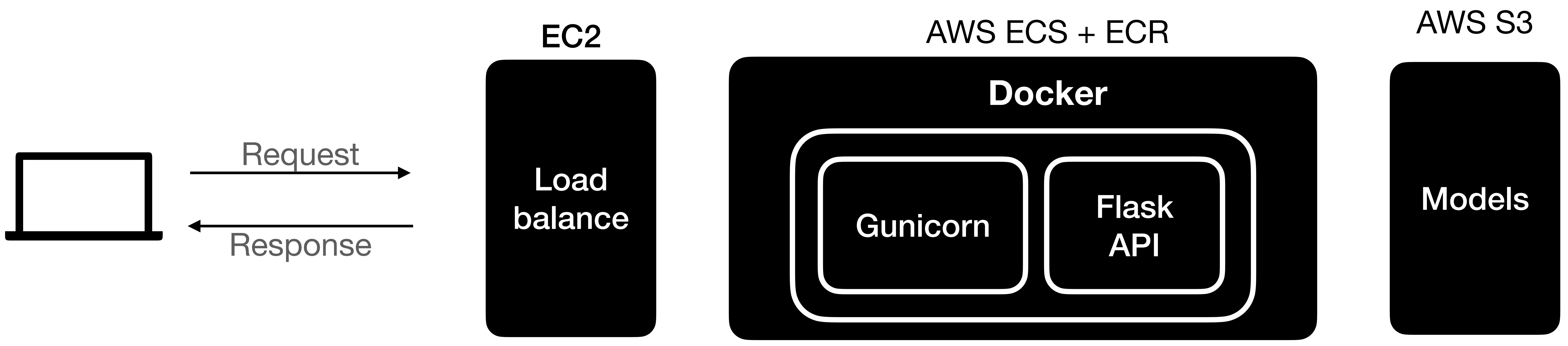
Uh….what about auth, monitor and scalability?

- Auth: I could add an auth mechanism in Flask. A user must input login Authorisation in the request body (e.g., username and password) when calling the API.
- Monitor: Docker has a native log system. I can build a dashboard based on these files. Although it is a little complicated.
- Scalability?

Monitor and scalability is a bit too much to handle…. Is there a cloud service that can help with it?

Inference Pipeline — v1.2

Leverage AWS ECS



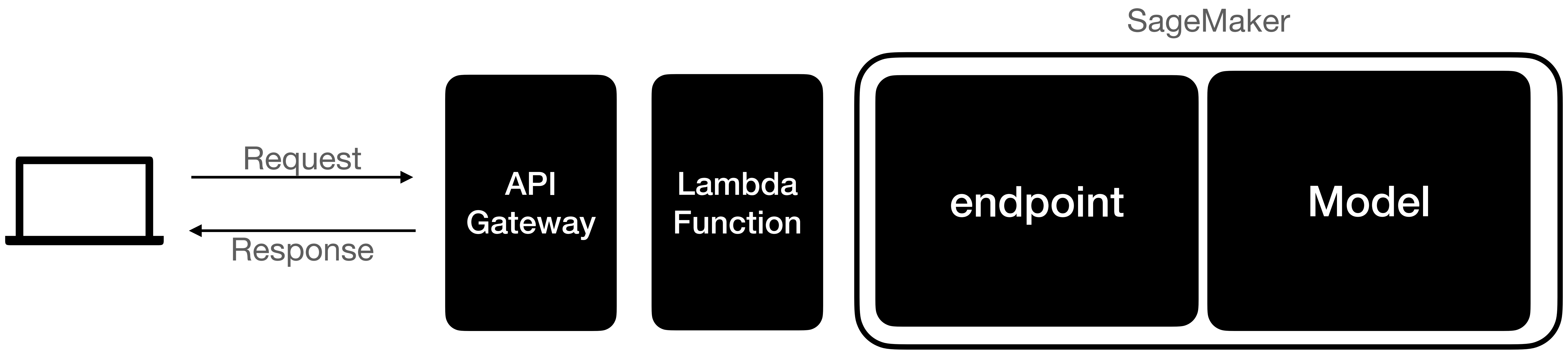
ECS can run multiple tasks and has auto-scaling options.

EC2 Application Load Balancer monitor the health and requests status.

The inference is secure, scalable and monitored in this version.

Inference Pipeline — v2.0

Use AWS SageMaker



Pros:

- SageMaker manages endpoints and models well with Log and monitor.
- Multi-Model Endpoints: Use multi-model endpoints if you need to deploy and serve multiple models on the same endpoint. This helps optimise resource utilisation.
- Endpoint Variants: Deploy multiple variants of your model (e.g., different instance types or configurations) and route traffic based on your requirements.

Cons:

- Expensive and steep learning curve in configuration.

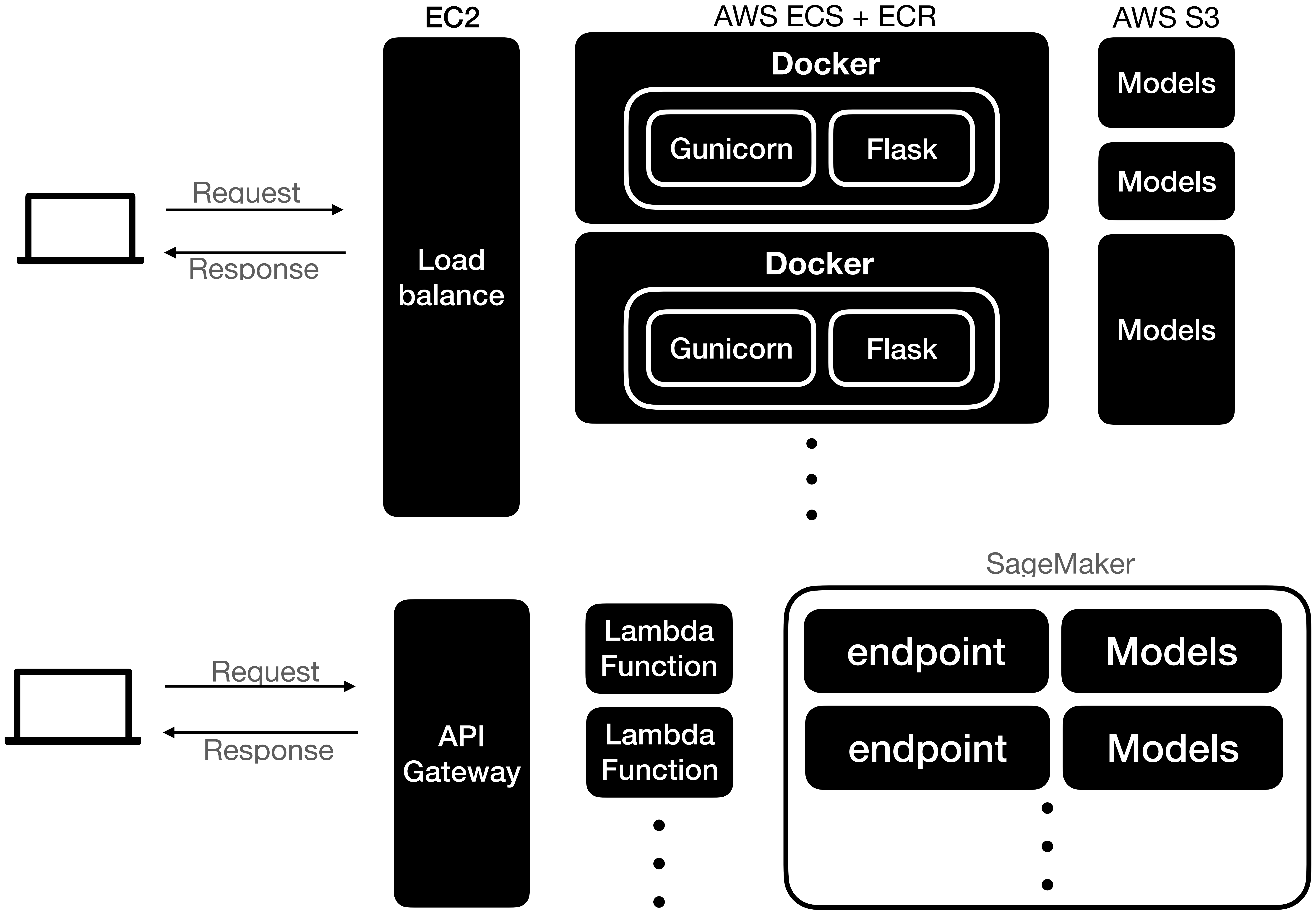
The Cases of Scaling

The V1 with ECS Flask

The scaling cases:

1. Add more version of fraud detection

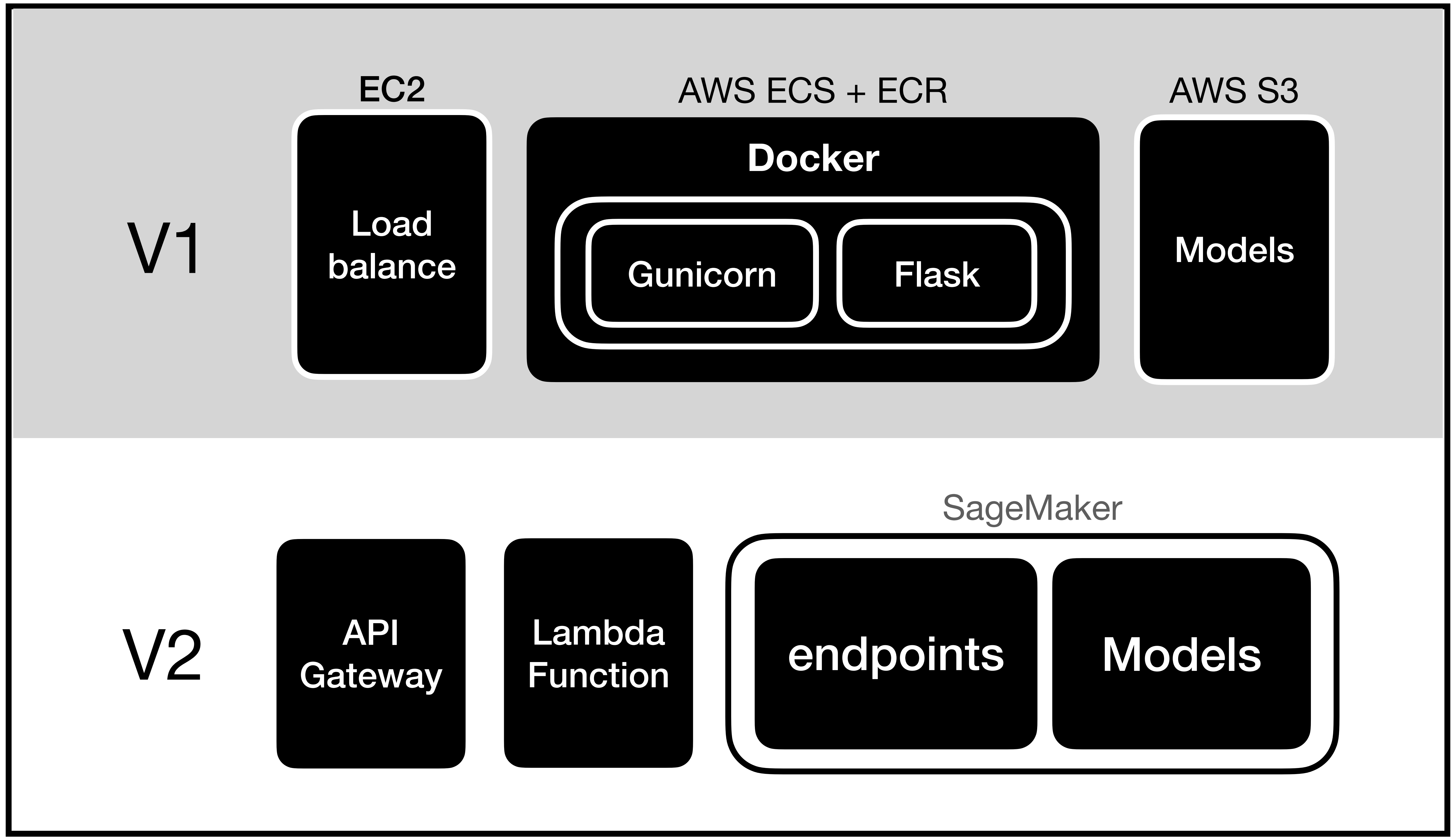
2. Add other types of detection



Inference Pipeline — v1 and v2
Recap

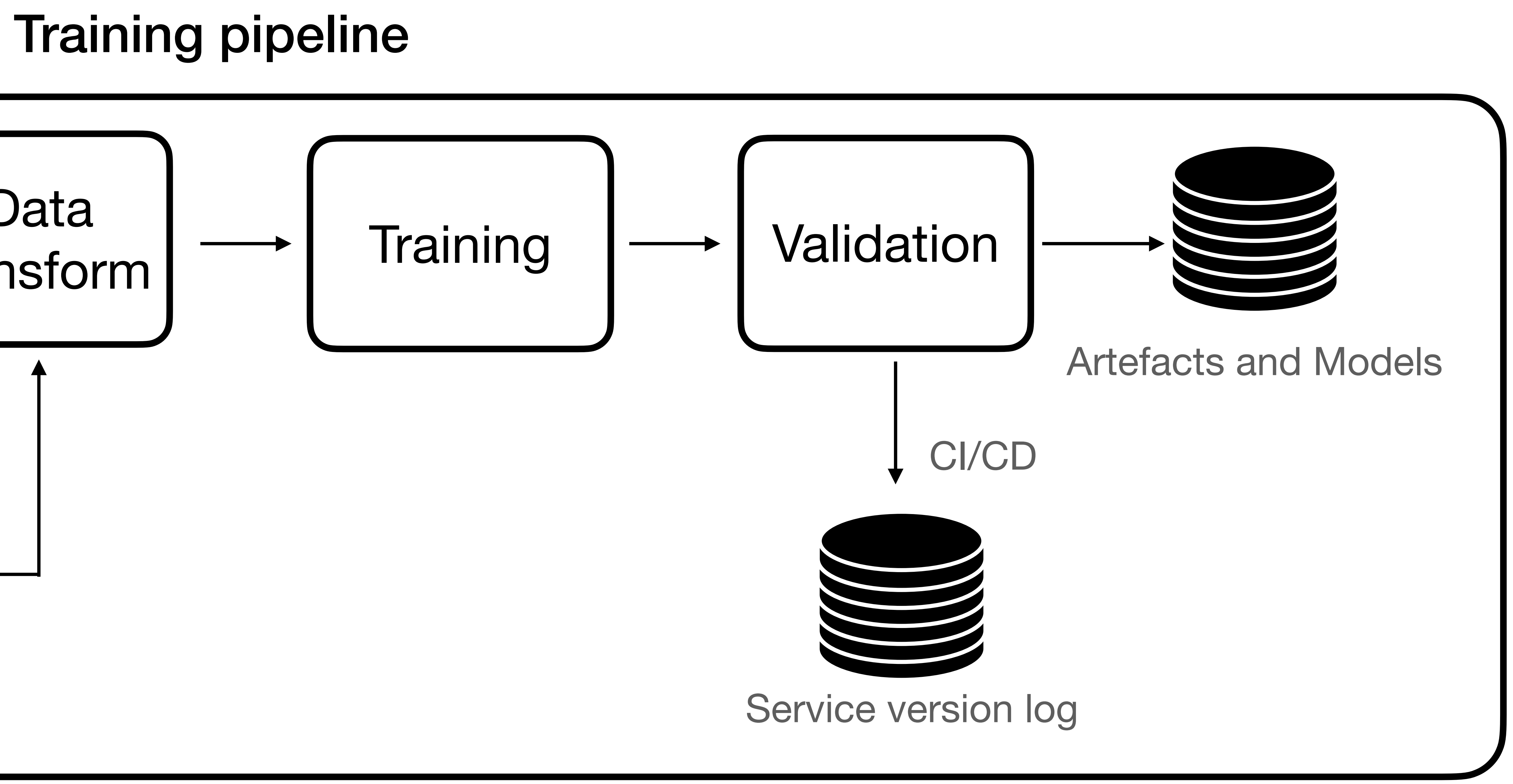
Assumptions:

- It is an online inference use case.
- It is ok to be a cloud-based pipeline.
- Internal usage does not get massive, and high-frequency request
- The model is stored in cloud storage.
- “client_id” is in the request payload and is used as the identifier in response.
- Design to handle multiple clients in one API call.



The Training pipeline

A conceptual architecture

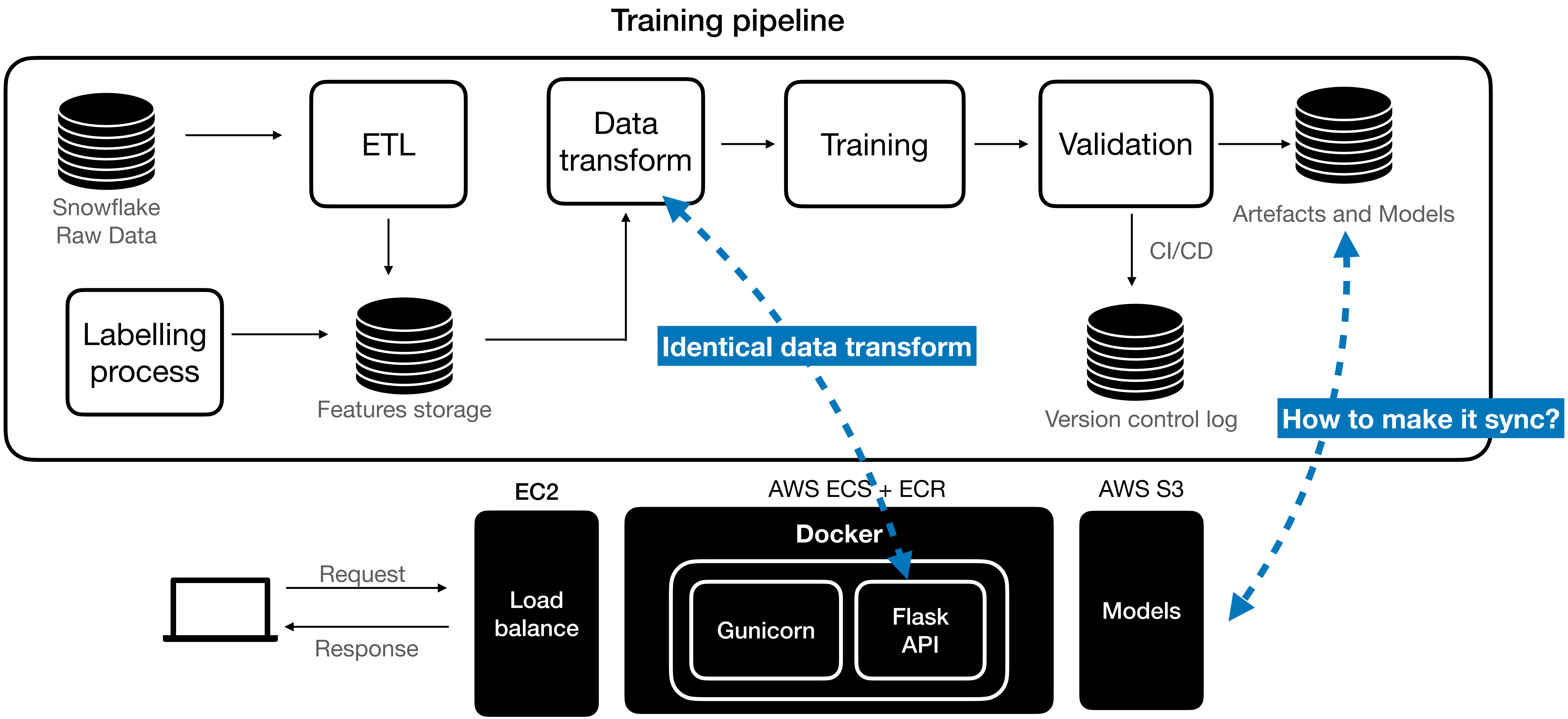


Considerations

1. **Flexibility**: Support multiple frameworks and languages (e.g., Python, R, TensorFlow, PyTorch, XGBoost).
2. **Scalability**: Handle varying sizes of data and computational loads.
3. **Cost-efficiency**: Optimise resource usage and cost.
4. **Security**: Ensure data and model security, including access controls and encryption.
5. **Automation**: Automate the end-to-end process to minimise manual intervention.
6. **Monitoring and Logging**: Track the performance and health of the training process.

The Integration Plan

The key considerations



The Integration Plan

The scenario of V1

Key points to check:

- The **model should be synchronised** between the inference and training pipeline.
  - Check model storage type. Tweak Flask to connect to it.
  - Make Sure the API uses the latest model by creating a version control table for services.
- The **input data format should be the same**.
  - Ideally, the inference pipeline should use the same data transformer.

Further Discussion

Learn more about the current training process

It would be great to learn more from these perspectives:

- Tech stack and code structure.
- Data handling in ingestion, preprocess, and transformation.
- Training tech stack.
- Monitoring and maintenance..

We can discuss the integration approach once we clarify the training process