12 - Factor app for machine learning systems

1. **Codebase:** One codebase tracked in revision control, many deploys.

    ☐ Ml Adoptions: Maintain a single, version-controlled repository for all code related to your ML system. This includes:
    - Data preprocessing scripts
    - Model training code
    - Model evaluation scripts
    - Deployment configurations
    - Inference code

2. **Dependencies:** Explicitly declare and isolate dependencies.

    ☐ **Ml Adoptions**: Use dependency management tools (e.g., `requirements.txt` for Python with `pip`, `conda.yaml` for Conda, Dockerfiles) to explicitly declare all libraries, frameworks (TensorFlow, PyTorch, scikit-learn), and system-level dependencies required for each stage of your ML pipeline (training, deployment, etc.).

3. **Config:** Store config in the environment

   - ☐ **Ml Adoptions**: Separate configuration from code. Store sensitive information (API keys, database credentials, cloud region), training parameters (learning rate, batch size), model hyperparameters (number of layers, units per layer), and environment-specific settings as environment variables.

4. **Backing Services:** Treat backing services as attached resources

   - ☐ **Ml Adoptions:** Treat external services your ML system relies on (databases for storing features or predictions, cloud storage for datasets and models, model registries, message queues, monitoring services) as attached resources accessed via URLs or connection strings configured in the environment.

5. **Build, Release, Run:** Strictly separate build and run stages

- ☐ **MI Adoptions:** Establish a clear separation between the stages of your ML pipeline
    i. **Build**: Involve packaging code, downloading dependencies, and potentially pre-processing data or creating feature stores.
    ii. **Release**: Involve specifying model versions, deployment configurations, and resource allocation.
    iii. **Run:** Involves executing.

6. **Stateless Processes**: Execute the app as one or more stateless processes

- ☐ **MI Adoptions**: State related to the model (model weights, vocabulary) should be externalized (e.g., loaded from a model registry or cloud storage)

7. **Port Binding**: Export services via port binding

- ☐ **ML Adaptation**: For online inference services, expose the prediction API via a well-defined port and protocol (typically HTTP/HTTPS). This allows other applications and services to easily consume your model's predictions.

8. **Concurrency**: Scale out via the process model

   - ☐ **ML Adaptation:** Scale your ML system horizontally by running multiple instances of your training jobs (for distributed training) or inference services.

9. **Disposability:** Maximize robustness with fast startup and graceful shutdown

   - ☐ **ML Adaptation:** Design your ML components (training jobs, inference services) to start up quickly and shut down gracefully.

10. **Dev/Prod Parity:** Keep development, staging, and production as similar as possible

    - ☐ **ML Adaptation:** Strive to maintain consistency across your different environments in terms of:
      - Operating systems
      - Libraries and dependencies
      - Backing services (use similar types and versions)
      - Deployment processes

11. **Logs:** Treat logs as event streams

- ☐ **ML Adaptation:** Output all logs (training metrics, inference requests and responses, system events) as continuous streams to a centralized logging system. Avoid writing logs to local files on individual instances.

12. **Admin Processes:** Run admin/management tasks as one-off processes

    - ☐ **ML Adaptation:** Treat administrative or maintenance tasks (e.g., data migrations, model retraining triggered manually, schema updates, backfilling features) as one-off, short-lived processes that are executed in the same environment as your regular application code. Use the same codebase and dependencies.