Snakedev

Data Assessment and Recommendation

Overview

Snakedev's Data Assessment and Recommendation service is a comprehensive review of a client's data platform, delivering customized best practices based on two decades of experience. This is a collaborative process between consultant and client with emphasis on:

- Data architecture
- Technical scalability and sustainable code growth
- Operational reliability and developer efficiency
- Data quality and software testing
- Team composition and development process
- Cost management and business value

The deliverable is a written set of recommendations and sample code, with ample time for discussion with implementers. Working closely with a client's engineering team, Snakedev will examine critical functions such as data ingest, data modeling, orchestration and business intelligence. We'll also consider human factors, identify skill gaps, and design a platform that client can grow and maintain.

Schedule

The assessment is conducted in 40 hours over four weeks and consists of code review, calls with technical and business stakeholders, research and write up. A typical schedule

- Week 1: code review, three 90 minute high-level calls
- Week 2: deep dive on tricky issues - two 90 minute calls, research and review
- Week 3: research and writing, follow up 60 minute calls and questions by email
- Week 4: written report delivered at start of week, two 90 minute calls to discuss

Expect homework! Client's engineers will need to find answers to questions that come up during discussions.

Deliverables

The written report includes:

- Summary of findings

- Current status and future plans, including data sources and volumes
- Overall data architecture
- Specific implementation strategies using open source and proprietary tools
- Identification of critical deficiencies and suggested remedies
- Timeline of priorities incorporating business goals, technical needs, interdependencies, and hiring plans

Sample code will be provided as needed.

Follow Up

Follow up services may include:

- Oversight: ongoing code review of implementation (2-5 hours per week)
- Office hours: weekly calls with the team to answer any questions that arise (1-2 hours per week, questions emailed in advance)
- Deeper dive: in-depth looks into specific areas needing further attention (10-20 hours)
- Workshops: customized training to upskill team members (fixed cost per person)

Project based software development is also available.

About Us

Snakedev is led by Pete Fein, an interdisciplinary consultant with 10 years experience solving hard problems for technical clients. He's a solutions architect and subject matter expert in data engineering and MLOps who has been programming in Python for over 20 years. Pete is supported by an extensive network of collaborators in the Python and data communities. He is currently writing a book, *Principles of Data Engineering*, for Pearson (due out in 2021) and teaches workshops on the O'Reilly platform. More info at snake.dev

Case Study: XYZ Project at BigCo

The following assessment and recommendation was performed in 2021 for BigCo, a well-known enterprise company. The XYZ project is a global initiative to improve marketing communications, part of a new effort to apply data science to the business. The team was newly formed (nine months old at the time) and consisted of six employees and 10 external contractors, all with data science, data engineering or machine learning backgrounds.

Snakedev was retained to evaluate the state of the project prior to initial launch and planned expansion to 15 additional markets over the following 18 months. The engagement was extended by two weeks at client's request to address specific technical issues discovered early on.

This is a good example of what's possible in an assessment and recommendation, and captures the full range of technical and human factors that can be addressed with this service.

XYZ Assessment and Recommendations

Overview

The XYZ Project is a data application that recommends sales and marketing actions to help BigCo more effectively communicate with customers.

This document is an assessment of the current (December 2021) implementation of the XYZ application. It provides an overview of the project, identifies risks to the project's success, and recommends solutions.

Executive Summary

- XYZ is well prepared for launch in Australia in January.
- Significant people, process and technology **risks threaten planned expansion to future markets**.
- Complex business logic and small data size imply these **risks are best addressed by optimizing for developer efficacy**.
- Most obstacles are generic software engineering issues and are not specific to data applications.
- Includes a detailed technical solution for speeding up pipeline execution by 100%.
- Suggests hiring an additional software engineer with general Python development skills.
- Recommends KPIs for software and data quality.

Contents

Overview
- Executive Summary
- Contents

Context
- Business Background
- Technical Background
- Challenges
- Strengths

Risks
- Fragmentation prevents code reuse
- Slow pipeline execution
- Architectural complexity
- Lack of software engineering experience
- Insufficient test coverage and data quality checks

- Technology limitations
- Builds are not reproducible

Solutions
- Modularize the code
- Create a git monorepo
- Simplify architecture
- Replace AWS Step Functions
- Use Shiv for packaging
- Speed up pipeline with Shiv
- Improve tests
- Coverage metrics
- Data quality checks
- Hire a Python developer
- Upskill
- Pin dependencies

Roadmap
- New market procedure

Appendix: Pipeline Speed Up

Appendix: Mara

Appendix: Job Description

Context

Business Background

A **franchise** is a particular product, currently only Foobar.
A **market** is a franchise in a particular country. Currently only Foobar/Australia.

Expected growth:

- Initial launch Australia – 1 market – January 2022
- Brazil and Canada – 3 markets – February 2022
- 5 more countries – 8 markets – April 2022
- More franchises – 12 markets – Q1 2023

Technical Background

Data is currently drawn from AWS Redshift (ABC database) and S3 and then processed with AWS Glue and Sagemaker. Output files are generated on S3 for loading into Salesforce (SFMC) and Veera by another team. The workflow is orchestrated with AWS Step Functions and

Lambda. Copies of the application are deployed for each market in the appropriate AWS region, to be closer to users and source data. No GPU is required.

External tooling includes QlikSense for reporting and manual data analysis, and SonarQube for source code linting.

Software libraries include a typical data science stack of Pandas, Scikit, Pytest, etc. Jenkins is used for CI/CD and Terraform for deployment. Developers use a variety of environments and versions, including Conda and system (OS) Python's virtual environments are not widely used.

Datasets are only ~300MB in each of the initial three markets. Future data sets are expected to be similarly sized. Each market is completely independent; there is natural partitioning. **The entire pipeline could be run on a single cloud instance or developer's laptop.**

Data is a mix of short plain texts, categorical, datetime and scalar values, in dozens of columns. Typical data operations affect a few thousand rows. Data points are uneven in time.

Inference is run once a week on Sundays, producing recommendations for the following week. The recommendations must be available by Monday morning local time. There is no other latency requirement. Models will be retrained twice per year.

A web application for end user rule configuration is under active development and not considered further in this report.

Challenges

- **Aggressive expansion into many additional markets, each with its own unique rules, schema variations, and statistical differences.**
- The application is quite complex, with ~40 data processing and machine learning scripts across Glue and Sagemaker.
- XYZ draws on many source data tables, and there are ~30 SQL tables and views to prepare and transform this data.
- Significant coordination overheard among external contractors, internal stakeholders and internal technical teams.
- Very tight hiring market for data engineers and scientists, ML engineers, and MLOps for the foreseeable future.

Strengths

- **The team is energetic, intelligent, and engaged.** All team members were very eager to improve the application and receptive to suggestions.
- **High level documentation is excellent**, among the best I've seen in 10 years for a project of this nature.
- Source code documentation is also quite good

Risks

Fragmentation prevents code reuse

- For Round 2 countries (Brazil and Canada), the git repo for the Australia project was copied wholesale and then modified. This is not a scalable development strategy for 15 markets as code reuse is impossible. Suggested solutions such as multiple branches in one repo or cherry picking changes between repos are not viable.
- Code base is not designed for reuse or extensibility. **Refactoring to support additional markets must be done ASAP.**

Solutions: create a git monorepo, modularize the code

Slow pipeline execution

- Running the **pipeline takes 2.5 hours for each proposed patch** to the source code before it can be peer reviewed and merged.
- Ideally this process should take under 15 minutes**.** Development velocity is severely impacted as a result and **will not support additional markets as planned**.

Solutions: speed up pipeline with shiv, simplify architecture

Architectural complexity

- The application has many moving pieces on AWS, using a half dozen services, often as little more than job runners for Python scripts.
- Complexity adds unnecessarily to the long term maintenance burden of the XYZ application.
- Cloud-dependent stacks prevent engineers from experimenting with code locally. This significantly **slows down the software development cycle** by forcing a reliance on CI/CD.
- Cloud complexity is an obstacle to the modularization necessary for expansion to additional markets.

Solutions: simplify architecture, replace AWS Step Functions

Lack of software engineering experience

- Existing team skills are largely focused on data science and machine learning.
- Many of the problems facing the XYZ team are generic software development issues.
- **Deficiency of software engineering experience will become critical** as the size and complexity of the application increases with the move into new markets.

Solutions: hire a Python developer, upskill

Insufficient test coverage and data quality checks

- Tests as written are not rigorous and do not sufficiently cover the source code base.
- Integration tests are not reproducible due to the use of live data.
- Minimal unit tests for individual code units.
- Very limited data quality checks and model monitoring. These are manual review processes.
- **Data corruption is a likely outcome.**

Solutions: improve tests, data quality checks, coverage metrics

Technology limitations

- Enterprise-wide restrictions on the technologies available pose serious obstacles implementing the application in an efficient and cost effective manner.
- The ban on all use of Docker containers and EC2 cloud computing instances is a major hindrance and some desirable software is completely unavailable as a result.
- For example, these restrictions **added 50% overhead to this assessment report** itself.

Solution: use shiv for packaging

Builds are not reproducible

- The exact version of third-party libraries in use may differ each time the application is built.

Solution: pin dependencies

Solutions

Modularize the code

- Adopt a plugin architecture for all XYZ code. Stevedore is an excellent library for this purpose.
- Using the AU market as a basis, insert hooks in scripts and pipelines to allow customization for other markets.
- Find potential hook points by diffing the AU, BR, and CAN codebases.
- For SQL scripts, replace in-house DDL code with `dbt`. dbt model definitions can be extended and customized using a powerful templating language.
- Use dbt's `config.yaml` or command line variables to select the desired market.

Create a git monorepo

- Merge all other repositories into subdirectories of `xyz_au`, starting with `xyz_au_db_sync`.
- Merge `xyz_br` and `xyz_can` after the codebase has been modularized.
- CI/CD should run tests for all subprojects on each commit. Subprojects should be treated as separate, with their own `requirements.txt`, `setup.py`, and packaging in a wheel, virtualenv, or shiv.

Simplify architecture

- Put complexity into Python code rather than architecture.
- Use a single Glue job and eliminate use of other AWS services including Step Functions.
- Package software with shiv and orchestrate with mara.
- Replace deequ on EMR with Great Expectations for data quality checks.
- **Refactored application should be runnable entirely locally**. Development should be possible on an airplane.
- Use localstack to mock AWS services such as S3 and Redshift.

Replace AWS Step Functions

- After modularizing and packaging as shivs, replace AWS Step Functions with Mara (see Appendix: Mara).
- Use the plugin architecture to customize the base workflow for different markets.
- Package the pipeline as three Shivs: `mara`, `dbt`, `xyz` internal code.
- In Mara pipelines, only use the Bash executor to call CLI commands. Avoid the use of Python and SQL executors.
- Write a single glue entrypoint script (this is different from Shiv's entrypoints). This script runs the mara pipeline, passing through any parameters such as `--market={AU, BR, CAN}`
- Include the `.pyz` shiv files when you deploy the glue job (see Appendix: Pipeline Speed Up for an example).

Airflow

At the request of XYZ team members, I evaluated Apache Airflow as an alternative orchestrator. I strongly advise against its use and believe that the adoption of **Airflow would seriously imperil the XYZ project**.

- Airflow is extremely complex software and requires significant engineering experience and discipline to use effectively.
- Airflow is intended as an enterprise-wide solution for running thousands of tasks and is not appropriate for the small number of jobs in the XYZ pipeline.

- Amazon's hosted Airflow service is over a year out of date and uses the worst possible approach for deployment.
- Airflow requires the use of unavailable technologies (Docker) for ideal operation and any form of testing.

Use Shiv for packaging

- Use Shiv for packaging applications, both XYZ internal and third party (e.g. dbt).
- Shiv bundles an entire virtualenv into a single, executable zip file (a `.pyz`).
- This shiv can have multiple entry points (command line scripts) allowing a single shiv to be built in a CI/CD and deployed for multiple purposes.
- Libraries should be built as wheels as normal.
- See Appendix: Pipeline Speed Up for an example

Speed up pipeline with Shiv

- ~50% pipeline run time is spent installing pip requirements.
- Avoid this overhead by building a single shiv in CI/CD which contains all XYZ Glue code.
- Reuse this shiv for all Glue jobs until AWS Step Functions have been replaced.
- see Appendix: Pipeline Speed Up

Improve tests

- Remove dependency on live Redshift by using pytest fixtures to load and test against pre-baked data (CSV files).
- Tests should compare their calculated dataframes against the fixture for full equivalence, including individual values, not just column existence.
- Use Datatest to write additional checks, such as "this column must contain values 1-5."
- Each script should have its own tests, with input and output files addressing a variety of cases.

Coverage metrics

- Improve test coverage with coverage.py. Run pytest under coverage and include the report in CI/CD logs.
- Coverage can also generate annotated source code, indicating which lines remain to be tested.
- **Adopt coverage percentage as a KPI of software quality** in code reviews, standups, etc.

Data quality checks

- Each data processing step should have a quality check immediately after it.

- Set `ALERT and FAIL` levels as appropriate. `ALERT`s send an email and `FAIL`s cancel the entire pipeline run.
- Integrate existing evident.ly checks into the pipeline in this matter. **Use p-value as a KPI for model performance.**
- Adopt Great Expectations, an excellent tool for data quality. Checks can be reused for CSV and SQL data sources.
- Develop simple checks ("% null") and more complex historical checks ("within 1 standard deviation of previous 3 months").
- Publish Great Expectations reports to S3 for each run.
- Replace existing Deequ checks with GE equivalents. This is significantly faster and simpler and removes dependency on AWS. Data sizes are too small to require Spark.
- dbt also supports data checks. At a minimum, specify all columns in dbt's schema.yaml to ensure SQL is generating the correct columns.

Hire a Python developer

- Hire a software generalist as a core XYZ team member, under the job title "Python Developer (Data/Backend)".
- A medium to senior level of experience in Python, backend, or full stack development would help the team meet immediate challenges.
- New hire will help upskill other team members, improve developer effectiveness, and increase delivery velocity of the team overall.
- Specific experience with data applications is helpful but not required.
- See Appendix: Job Description for an outline of desired skills and experience

Upskill

- Read the book Practices of the Python Pro as a good general resource. It also contains a section on plugin architectures.
- A more extensive code review of the source code could be valuable. Interactive refactoring with XYZ team members would improve the quality of the code and upskill the team.
- Team members expressed desire for further training in data tools and best practices.

Pin dependencies

- Pin dependencies in requirements.txt like: `tensorflow==2.4.3` not bare names like `tensorflow`.
- `pip-tools` can help keep these pinned dependencies up to date.

Roadmap

Phase 1

- Speed up pipeline with Shiv
- Pin dependencies

Phase 2

- Improve tests
- Coverage metrics

Phase 3

- Create a git monorepo
- Plugin architecture

Phase 4

- Merge `xyz_au_db_sync` repo into `xyz_au`
- Modularize SQL with dbt
- Use Shiv for packaging

Phase 5

- Replace Step Functions

Phase 6

- Data quality checks
- Replace Deequ

Phase 7

- Launch Round 3 markets

Phase 8

- Merge `xyz_can` and `xyz_br` into main monorepo

New market procedure

Once recommended changes are in place, follow this procedure for adding a new market. I estimate it would take a data scientist and data engineer about two sprints to implement each new market.

1. Identify differences to base XYZ data models, data processing and pipeline.
2. These should be at existing plugin hook points. It may be necessary to refactor base XYZ code to add additional hooks. If so, migrate existing markets to use them (this will be very easy with good test coverage).
3. Implement hooks and corresponding tests for new markets.
4. Train model on new market data.
5. Verify things are working correctly (manual sanity check).
6. Deploy to production.

Appendix: Pipeline Speed Up

The majority of the runtime for Glue jobs is installing requirements with pip for each job. We can use Shiv to speed this up https://shiv.readthedocs.io/en/latest/

`xyz_job` is the name of the actual code we are interested in running, for this ex we'll use `nontgts` and `interactions`

Create Glue entry point script
Create a `shiv_entrypoint.py` that will be the entry point called by all Glue jobs. Here's what it does:

Parse xyz_job argument from `sys.argv` and set it as an environment variable:
`os.environ[SHIV_CONSOLE_SCRIPT]=<xyz_job>`

AWS is supposed to pass the JOB_NAME, but does not appear to do that for python–shell jobs, see https://dev.to/1oglop1/aws-glue-first-experience-part-3-arguments-logging-120 for more info

Run the shiv and pass through command line arguments:
`subprocess.check_call('./xyz_glue.pyz', *sys.argv[1:])`

Build a shiv
General procedure for writing a setup.py and building a wheel:
https://wiki.bigco.com/display/makingafastloop/Creation+of+wheel+file+and+script.tar.gz+for+Model+Training

Create a `setup.py` for a `xyz_glue` library.
Include all `src/gluejobs/*.py` files
https://packaging.python.org/guides/distributing-packages-using-setuptools/#packages

For both scripts, refactor to have a `main()` function:
https://git.bigco.com/projects/XYZ/repos/xyz_au/browse/src/gluejobs/interactions.py#100
https://git.bigco.com/projects/XYZ/repos/xyz_au/browse/src/gluejobs/nontgts.py#20

Use that main() function as a `console_script` for each xyz_job:
https://packaging.python.org/guides/distributing-packages-using-setuptools/#console-scripts

Build a wheel as `xyz_glue.whl`:
```
python setup.py bdist_wheel
```

Create the shiv:
```
shiv -o xyz_glue.pyz -r requirements.txt xyz_glue.whl
```

Create glue jobs
Use the aws CLI, like:
https://git.bigco.com/projects/XYZ/repos/xyz_au/browse/terraform/modules/glue-jobs/glue_job.tf#18

Use `shiv_entrypoint.py` you created above as the entry point. Upload the shiv when you create the Glue job with `--extra-files xyz_glue.pyz`. Set a default argument for the Glue job called `xyz_job` with the value of the xyz_job name.

Run Glue jobs
Call the glue jobs with the aws CLI, like:
https://git.bigco.com/projects/XYZ/repos/xyz_au/browse/src/skeleton/pipeline.asl#405
https://git.bigco.com/projects/XYZ/repos/xyz_au/browse/src/skeleton/pipeline.asl#422

Appendix: Mara

We are exploring using Mara Pipelines as our primary orchestration tool for the next major release of the XYZ project. Mara is simpler than Apache Airflow and more powerful than scripts/Makefiles. Mara will be a smoother developer experience and enable the entire stack to be run by developers locally or in CI/CD. The net outcome will be faster feature iteration and reduced operational costs and lower AWS bills.

We would replace the existing ~40 Glue and Sagemaker jobs with a single long running Glue job that executes the entire pipeline. We eliminate the use of Step Functions: as Mara is just Python code, we would be able to reuse (subclass) base pipelines as we expand the XYZ project into new markets. There will be a corresponding simplification in our use of AWS services (Lambda, etc.) generally.

Mara includes sophisticated reporting, log viewers and historical statistics on pipeline runtime performance. We would extend Mara to send alert emails or Teams chat messages in the event of pipeline errors. We could also extend it automatically save intermediate data & models to the S3 data lake.

Mara features an internal-facing dashboard for use by developers to monitor and maintain the pipelines: it is a simple Flask web app backed by PostgreSQL. We would deploy this on Lambda behind an Application Load Balancer and a RDS database. Flask supports a variety of options for SSO and access control that will be implemented according to the recommendations of the platform team.

Technical Details

Library for Flask on Lambda
Example code for deployment
Accompanying blog post – we would be applying with Lambda and ALB via terraform instead of Ansible, but the basic idea is the same.

Appendix: Job Description

Python Developer (backend/data)

- Extensive history of deploying and operating production applications
- Excellent knowledge of Python
- Strong SQL skills
- Testing with pytest and coverage
- Some experience with Pandas
- Knowledge of Flask
- Familiarity with AWS; some experience with Terraform if possible
- Packaging Python code
- Refactoring existing code

A backend developer or even full stack developer would be fine, and preferable to somebody with an existing data-only background. This position would be a good opportunity for someone looking to gain experience in the data space.

For interviewing, ask to see sample code from candidates prior to a phone call. Request a single file that can be understood without additional context, and that has run in production (and not just boilerplate like a Django models file). `utils.py` is usually perfect.

As an interview exercise, ask them to refactor an existing data processing scripts to improve testability and extensibility.