Shared AI Services Hosting Application

Andrea Roberson

U.S. Census Bureau

Any views expressed are those of the author(s) and not necessarily those of the U.S. Census Bureau.

Annual Capital Expenditures Survey (ACES)

- **Provides national-level estimates of annual capital investment in new and used buildings, structures, machinery, and equipment by U.S. non-farm businesses**

ITEM 2 **CAPITAL EXPENDITURES**

Report the following domestic capital expenditures data for the entire company. *(Refer to page 4 of Instructions)*

Example: if figure is $1,179,125,628.00 report →

Bil.	Mil.	Thou.
1	1 7 9	1 2 6

Row	CAPITAL EXPENDITURES *(Refer to Page 2 of Instructions)*	Structures (1)			Equipment (2)			Other *(Describe in Item 3)* (3)			Total ***(Add columns 1+2+3)*** (4)		
		Bil.	Mil.	Thou.	Bil.	Mil.	Thou.	Bil.	Mil.	Thou.	Bil.	Mil.	Thou.
20	Capital expenditures for NEW structures and equipment ***(Include major additions, alterations, and capitalized repairs to existing structures)***												
21	Capital expenditures for USED structures and equipment												
22	**TOTAL capital expenditures** ***(Add Rows 20 + 21)***												

Total should equal Item 1A, Row 11

ITEM 3 **List the items included in "Other."** Report in thousands of dollars. **Furniture and fixtures, computers, capitalized computer software, and motor vehicles** should be reported as equipment. **Leasehold improvements** should be considered new structures or new equipment based on what is being improved.

Row	(1) Description of Capital Expenditures	(2) Bil.	Mil.	Thou.
30				

lending **money** to **commercial** banks (**fabricated response**)

Source: https://www.census.gov/programs-surveys/aces.html

Challenges of Prediction Serving

Many applications and many models

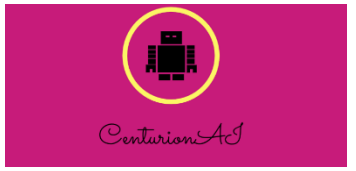
Anomaly (Outlier) Detection

Content Rec.

Personal Asst.

Robotic Control

Text Classification



theano

Dato Create

Caffe

TensorFlow

scikit learn

dmlc mxnet

VW

APACHE Spark

KALDI

SASHA Requirements

- **Decouple applications from models and allow them to evolve independently from each other**
- **The Frontend Dev perspective: focus on building reliable, low-latency applications**
  - Provide stable, reliable, performant APIs to meet Service Level Agreements (SLAs)
    - Scale system, hardware to meet application demands
  - Oblivious to the implementations of the underlying models
- **The Data Scientist perspective: focus on making accurate predictions**
  - Support many models and frameworks simultaneously
  - Simple deployment and online experimentation
  - (Mostly) oblivious to system performance and workload demands

What are we doing?

- Work done on the **Annual Capital Expenditures Survey (ACES)** API taught us how to wrap ML scripts to provide high throughput and availability.
- We expect to see growth in the use of ML/AI in live instruments.
- We need to create a unified platform rather than one-offing everything.
- Goals for this platform:
  - Deliver ml results as fast as possible
  - Handle errors, monitoring, alerting, reporting and whatever else can be centralized
  - Manage shared hardware efficiently and fairly
  - Reduce time/labor for delivery of additional APIs to the platform
  - Give more freedom and control to AI developers
    - I/O control
    - Configuration control
    - CI/CD control
  - Easily extendable with no service interruptions

Components

- API Server
  - Main endpoint – Accepts/responds to HTTP requests, authenticates, logs, error handling
- Admin Panel
  - Register clients
  - Manage settings
  - View reports
  - Run health checks and view health stats
  - Test clients
- API Clients
  - ML/AI scripts owned by various application areas
- API Customers
  - Make API requests, consume API results

How does it work?

- Fundamentally, a survey provides input to a python script to do ML stuff then return something over a network. User enters ‘Truck’, ML responds ‘Equipment’
- After profiling, A/B testing, etc. We determined ML scripts were CPU limited and the largest consumer of the CPU time was the deserialization of the model file(s) to python objects.
- **Solution – keep it hot**
- Inbox/Outbox design pattern
  - The “API Customer” sends a post/get to “API Server”
  - The “API Server” drops a job to inbox (json) and immediately and repeatedly starts checking the outbox
  - The “API Client” is in a “while True:” looking for work in the inbox. It sees a job, processes it, then drops another json file to the outbox
  - The “API Server” sees the job in the outbox and returns to the customer
- Need to monitor health of daemonized python script – not a typical use case!
  - Continuous health checks (every minute)
  - Nightly Restarts
  - Auto Spawning
  - Zombie Checking
  - Server Resource Monitoring

SASHA API EXAMPLE