Project - Portfolio Data Clearinghouse

Vladimir Ulogov

December 2025

1 Abstract

This document is a take-home assignment submitted as part of the Vest interview process.

2 Exercise Context

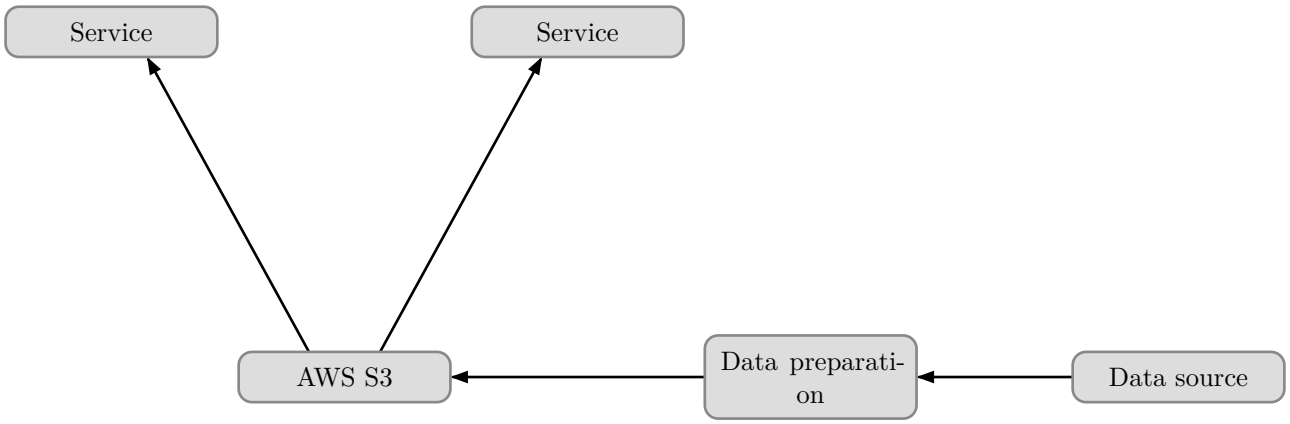
To satisfy the exercise requirements, I deployed the following elements of a clearinghouse infrastructure:

- AWS-deployed infrastructure
- A CI/CD pipeline using Ansible
- A database
- A JSON-RPC application developed in Rust
- Architectural Justification

2.1 Decisions that affected the architecture of exercise.

- AWS: Selected as the cloud platform, per Vest's suggestion.
- Ansible: Chosen for CI/CD and DevOps operations. It provides a more robust and flexible feature set for infrastructure automation compared to using GitHub Actions alone.
- Database: SQLite3 was chosen as a good embeddable, serverless database option. In production, we will use either a traditional ACID SQL server, such as Postgres or MariaDB, or other types of data storage provided by AWS.
- Rust: Selected as the primary development language due to its focus on producing safe, efficient code and its built-in support for comprehensive testing.

3 Architecture



3.1 Infrastructure

A very simple infrastructure has been set up on AWS.

Resource name	**Access point**	**Description**
web-server-1	18.205.3.140	Service node, Bastion host, CI/CD, Data preparation host
web-server-2	44.205.251.231	Service node
S3 storage	vladimir-ulogov-demo-storage-dev-82e1a943	Storage for processed data

3.2 Clearinghouse pipelines

Two distinct pipelines have been prepared to work with data sources, create data files, import data into them, recognizing different data file formats, upload data files to S3, and download data files from S3 for distribution to processing nodes. Here is the list of the scripts involved in preparing and deploying data:

Script	**Description**
create_tables.sh	Script that creates new data file, create schema for data of two formats and import initial data. Result is stored in ./data folder
download_data_files.sh	Downloading data files from SFTP server and preparing toem for use by *create_tables.sh*
upload_db.sh	Uploads prepared data file onto S3
download_db_on_cluster.sh	Trigger DB download through entire cluster
download_db.sh	Download DB on a single node

4 Implementation

All source code and demonstrations of the Terraform, CloudFormation, and Ansible code are available on the project's bastion host in the home directory of e2-user.

Directory	Description
ansible	Ansible recipies for CI/CD and DevOps operations
cloudformation	Template for creation budget monitoring observability
terraform	main.tf - the core instructions for creating a test infrastructure
server	Application server sources and build environment
sql	SQL statements for DB tab creation and import

4.1 Coding style

My coding style here isn't my usual one, as the assignment scope extends beyond the expected few hours. I modularized the application but left out all comments and my typical code optimizations. However, this code shows my ability to build a high-performance backend based on JSONRPC using a multithreaded hyper engine.

4.2 A bit about server sources.

The majority of the server's logic is located in the directory /home/ec2-user/server/src/cmd on the build server. In this table, I am explaining which functionality is located in which source file.

File	Description
mod.rs	Module description, essentially main entrypoint
setloglevel.rs	Set's logging for the application. You may increase logging level by passing -d/-dd or -ddd to *app_server*
app_sql.rs	DB query code. It's a very simple implementation for one or two parameters. It should be passed through function parameters Really.
app_blotter.rs	Main entrance for blotter call
app_positions.rs	Main entrance for positions call
app_alarms.rs	Main entrance for alarms call

File *.env.sh* must be used for prepare environment variables required by app_server

4.3 Testing application

To test *app_server* you have to run internal test cases by executing

```
1 cd /home/ec2-user/server
2 cargo test -- --show-output
```

on build host. Expected output is

```
    Compiling app_server v0.1.0 (/home/ec2-user/server)
     Finished `test` profile [unoptimized + debuginfo] target(s) in 1.93s
      Running unittests src/main.rs (target/debug/deps/app_server-f3a1644df03ae736)

running 6 tests
test cmd::app_blotter::test_blotter ... ok
test cmd::app_alarms::test_blotter ... ok
test cmd::app_positions::test_positions ... ok
test cmd::test_app_access_key ... ok
test cmd::app_sql::test_select ... ok
test cmd::test_db_open ... ok

successes:

successes:
    cmd::app_alarms::test_blotter
    cmd::app_blotter::test_blotter
    cmd::app_positions::test_positions
    cmd::app_sql::test_select
    cmd::test_app_access_key
    cmd::test_db_open

test result: ok. 6 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

4.4 Build and deploy the application server

I did not create any automation for application building. This is something that must be done for a production solution. To build the server, just run

```
cd /home/ec2-user/server
cargo build
```

on build host

To deploy server on cluster nodes, run

```
deploy_app_server.sh
```

on build host

4.5 Call JSONRPC instance

Note, by default, the application server is not running for security reasons. To start the server, log in to a bastion host as ec2-user and run the following command:

```
start_app_server.sh
```
Bash

This will start the server on the cluster. To stop the server, run the following script on the bastion host:

```
kill_server.sh
```
Bash

This will bring the cluster down.

4.5.1 Blotter

```
curl -X POST -H "Content-Type: application/json" -d '{"jsonrpc": "2.0", "method": "blotter", "params":["helloworld", "2025-01-15"], "id":123 }' 18.205.3.140:8080
```
Bash

returns

```
{"jsonrpc":"2.0","result":["{\"TradeDate\": (Text, \"2025-01-15\"), \"AccountID\": (Text,
\"ACC001\"), \"Ticker\": (Text, \"AAPL\"), \"Quantity\": (Integer, 100), \"Price\":
(Real, 185.5), \"TradeType\": (Text, \"BUY\"), \"SettlementDate\": (Text,
\"2025-01-17\")}","{\"TradeDate\": (Text, \"2025-01-15\"), \"AccountID\": (Text,
\"ACC001\"), \"Ticker\": (Text, \"MSFT\"), \"Quantity\": (Integer, 50), \"Price\": (Real,
420.25), \"TradeType\": (Text, \"BUY\"), \"SettlementDate\": (Text,
\"2025-01-17\")}","{\"TradeDate\": (Text, \"2025-01-15\"), \"AccountID\": (Text,
\"ACC002\"), \"Ticker\": (Text, \"GOOGL\"), \"Quantity\": (Integer, 75), \"Price\":
(Real, 142.8), \"TradeType\": (Text, \"BUY\"), \"SettlementDate\": (Text,
\"2025-01-17\")}","{\"TradeDate\": (Text, \"2025-01-15\"), \"AccountID\": (Text,
\"ACC002\"), \"Ticker\": (Text, \"AAPL\"), \"Quantity\": (Integer, 200), \"Price\":
(Real, 185.5), \"TradeType\": (Text, \"BUY\"), \"SettlementDate\": (Text,
\"2025-01-17\")}","{\"TradeDate\": (Text, \"2025-01-15\"), \"AccountID\": (Text,
\"ACC003\"), \"Ticker\": (Text, \"TSLA\"), \"Quantity\": (Integer, 150), \"Price\":
(Real, 238.45), \"TradeType\": (Text, \"SELL\"), \"SettlementDate\": (Text,
\"2025-01-17\")}","{\"TradeDate\": (Text, \"2025-01-15\"), \"AccountID\": (Text,
\"ACC003\"), \"Ticker\": (Text, \"NVDA\"), \"Quantity\": (Integer, 80), \"Price\": (Real,
505.3), \"TradeType\": (Text, \"BUY\"), \"SettlementDate\": (Text,
\"2025-01-17\")}","{\"TradeDate\": (Text, \"2025-01-15\"), \"AccountID\": (Text,
\"ACC001\"), \"Ticker\": (Text, \"GOOGL\"), \"Quantity\": (Integer, 100), \"Price\":
(Real, 142.8), \"TradeType\": (Text, \"BUY\"), \"SettlementDate\": (Text,
\"2025-01-17\")}","{\"TradeDate\": (Text, \"2025-01-15\"), \"AccountID\": (Text,
\"ACC004\"), \"Ticker\": (Text, \"AAPL\"), \"Quantity\": (Integer, 500), \"Price\":
(Real, 185.5), \"TradeType\": (Text, \"BUY\"), \"SettlementDate\": (Text,
\"2025-01-17\")}","{\"TradeDate\": (Text, \"2025-01-15\"), \"AccountID\": (Text,
\"ACC004\"), \"Ticker\": (Text, \"MSFT\"), \"Quantity\": (Integer, 300), \"Price\":
(Real, 420.25), \"TradeType\": (Text, \"BUY\"), \"SettlementDate\": (Text,
\"2025-01-17\")}","{\"TradeDate\": (Text, \"2025-01-15\"), \"AccountID\": (Text,
\"ACC002\"), \"Ticker\": (Text, \"NVDA\"), \"Quantity\": (Integer, 120), \"Price\":
(Real, 505.3), \"TradeType\": (Text, \"BUY\"), \"SettlementDate\": (Text,
\"2025-01-17\")}"],"id":123}``
```

4.5.2 Positions

```
curl -X POST -H "Content-Type: application/json" -d '{"jsonrpc": "2.0", "method":
"positions", "params":["helloworld", "2025-01-15"], "id":123 }' 18.205.3.140:8080
```
Bash

returns

```
{"jsonrpc":"2.0","result":["{\"Ticker\": (Text, \"AAPL\"), \"percentage\": (Real,
27.272727272727273)}","{\"Ticker\": (Text, \"GOOGL\"), \"percentage\": (Real,
18.181818181818183)}","{\"Ticker\": (Text, \"MSFT\"), \"percentage\": (Real,
18.181818181818183)}","{\"Ticker\": (Text, \"NVDA\"), \"percentage\": (Real,
18.181818181818183)}","{\"Ticker\": (Text, \"TSLA\"), \"percentage\": (Real,
9.090909090909092)}"],"id":123}
```

4.5.3 Alarms

```
curl -X POST -H "Content-Type: application/json" -d '{"jsonrpc": "2.0", "method":
"alarms", "params":["helloworld", "2025-01-15"], "id":123 }' 18.205.3.140:8080
```
Bash

returns

```
{"jsonrpc":"2.0","result":["{\"AccountID\": (Text, \"ACC001\"), \"has_over_20_percent\":
(Text, \"TRUE\")}","{\"AccountID\": (Text, \"ACC002\"), \"has_over_20_percent\": (Text,
\"TRUE\")}","{\"AccountID\": (Text, \"ACC003\"), \"has_over_20_percent\": (Text,
\"TRUE\")}","{\"AccountID\": (Text, \"ACC004\"), \"has_over_20_percent\": (Text,
\"TRUE\")}"],"id":123}
```

4.6 What I did not do?

There are several steps, absolutely necessary in production, that I deliberately missed.

- Any security checks and measures. While Rust makes a good deal of effort in keeping application memory from poisoning, and I am using recommended methods for forming SQL queries (avoiding „little bobby tables problem“, i.e., SQL injection), the code is not tested or verified, and not monitored for security.
- Production Ansible scripts and recipes are more polished and refined.
- Deepen the integration of DevOps practices and automate builds using Ansible, Jenkins, or a combination of both.
- An artifact repository would be advantageous.
- CloudWatch monitoring is inadequate. Implementing „real observability“ tools (such as Zabbix) is recommended.
- A great deal of error processing in the code. I plug potential issues with the .unwrap() call. Shouldn't do it in production.
- Database schema is „as-is“. I did not add any indexes. I must do index my database in production.

4.7 Pitfalls

I believe my CloudFormation template for budget observability and monitoring is correct. However, during template application, I encountered an error:

```
1 Resource handler returned message: "User: arn:aws:iam::288581185806:user/vladimir-ulogov-001 is not authorized to perform: budgets:ModifyBudget on resource: arn:aws:budgets::288581185806:budget/Monthly-Budget-interview-monitoring-vladimir-ulogov-001 because no permissions boundary allows the budgets:ModifyBudget action (Service: Budgets, Status Code: 400, Request ID: 275da9c2-7361-4147-83ee-50bf1e6a6e7b) (SDK Attempt Count: 1)" (RequestToken: 283d534d-7292-4fba-8708-002f246be098)
```

trace

But SNS pipelines was successfuly created and operational as I received a message:

Plain Text

```
1 You have chosen to subscribe to the topic:
2 arn:aws:sns:us-east-1:288581185806:budget-alerts-interview-monitoring-vladimir-ulogov-001
3
4 To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
5 Confirm subscription
6
7 Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests, please send an email to sns-opt-out
```

This indicates that we have an access issue for the user with an overall working solution. Once the access issue with the IAM policy is resolved, the template will create a working configuration. However, while this setup is adequate for testing and exercise purposes, I wouldn't recommend using it in real life. A more production-grade observability solution must be used, and I can discuss the solution I have in mind that will solve this problem in SPOG manner.