Kazakh-British Technical University

Cloud Computing

Final project

Cloud-Based Smart Inventory Management System

Student: Khanfiyeva Elnara

Professor: Serek Azamat

Date:12.12.2024

2024

Executive Summary

In this report it is provided an in depth analysis and description of the creating a inventory management app with the help of YandexCloud. The concept of this project is to develop and deploy an intelligent inventory management system that is based on Yandex Cloud Platform (YC). It is integrated with the functionality for stock management, supply chain inventory tracking for multiple locations, and data analysis with sales machine learning. In this report, the project development procedure, system structure, database entities, YC services used, and the security measures taken are explained in detail.

The system integrates with GCP services such as Compute Engine for scaling, Storage for access control, data storage in BigQuery, App Engine for auto-scaling, and YC for security. The findings of the project signify that a cloud solution is indeed capable of delivering sharp, smooth inventory support to businesses.

Table of Contents

Introduction

Today's fast paced, technology driven world has completely changed the way businesses do business using cloud computing. Its flexibility, scalability and cost saving offerings were previously difficult to provide with traditional on premises solutions. Small startups or big companies, every organization now relies on cloud platforms to seamlessly streamline its operations, efficiently manage its resources, and adapt to surges and dips in demands without blowing holes in its budget. More specifically, Yandex Cloud platform, due to its wide range of tools and services, continues to be a powerful choice for business looking to build, deploy and scale applications without a hitch. With billing, data management, and AI powered insights YC covers the gaps to be competitive in the digital first era.

Explanation of platform choice:

For a reason of ending of the Free Trial and the downfall of the Google Cloud Console Sign Up page (I was trying to create another account for free trial) I decided to use well known to me Yandex Cloud platform - the services are quite same, so it's not much different from Google.

In this project the tools utilized are the YC - Yandex Cloud services such as Yandex Cloud Console, YandexCloud Kubernetes, YandexCloud App Engine, Yandex Big Query and etc. The project is made for creating with the full availability for analysis and performance monitoring app for inventory management system. This project will show how important is for business processes to be analyzed for performance and effectiveness of informational systems. Managing systems are important to optimize the business processes, scoring the inventory products, make analysis of financial factors, product and project management and the business development itself.

For the project goals I also considered the parts of the IM system here such as real-time inventory tracking, multi-location management, generating predictive analysis to provide a secure, scalable and efficient data management. I will try to develop part of the system as backend API for managing users, products, locations, and orders. I will deploy the system on Yandex Cloud Managed Service Kubernetes using Compute Cloud, for high availability of the system which is key factor for the managing business systems. I will of course be deploying the cloud databases, and storage for inventory data. And I will provide the analysis and predictions with Yandex Query and ML integrations for sales planning.

System Architecture

This project's primary objective is to develop a secure, scalable, and user-friendly cloud-based inventory management application hosted on Yandex Cloud platform.

The system architecture of the InvenTree deployment is designed to optimize performance, scalability, and reliability. The architecture consists of Virtual Machine (Yandex Cloud Compute Cloud). It's a simple way to host the whole system, and the Docker file will be hosted there. It will insure the communication of the services. Docker containers here serve the basing role. I have the main InvenTree server – Python Django based web server is going to be up in the docker container, it will handle the applications requests and the Database communications. InvenTree Worker here serves processing background's tasks such as job queries, updates. I also utilize Caddy Reverse Proxy for managing the incoming HTTP traffic.

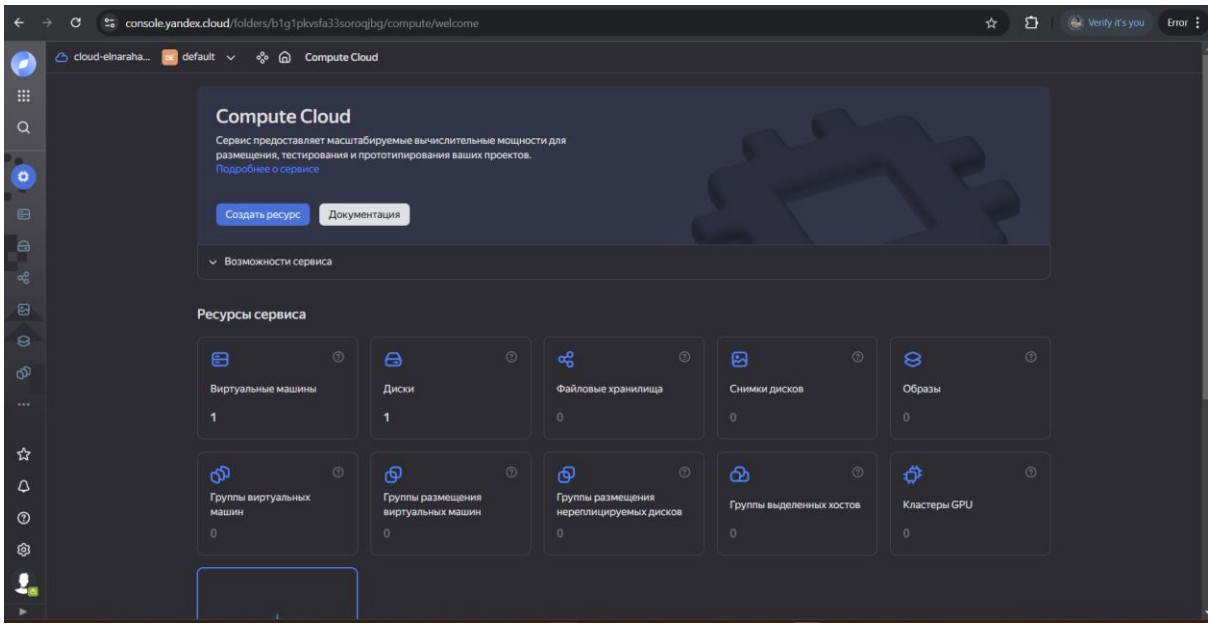


Figure 1. Compute Cloud on Yandex Cloud.

I will use the database MySQL on Yandex Cloud Managed Services site. The database will store all of the data that is structured, for example, inventory records, user information and logs of the system.

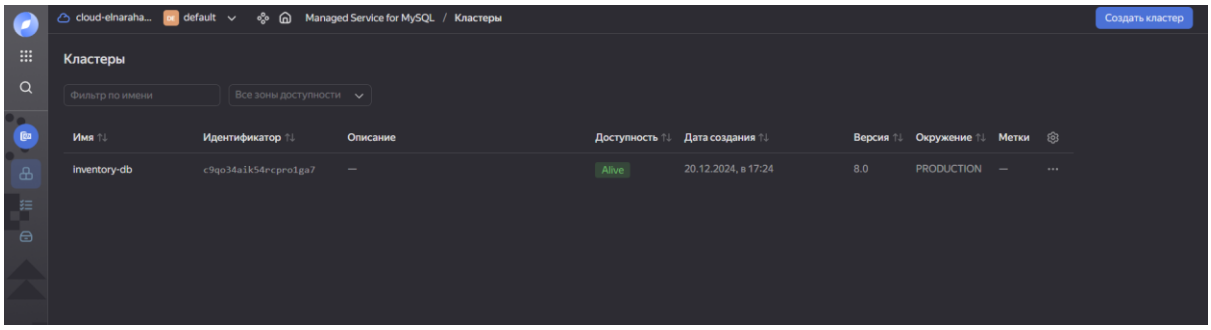


Figure 2. Compute Cloud – Virtual machine I've created.

Then I also used instead of Redis – Managed Service Valkey – is the alternative of the redis in Yandex Cloud. The reason is that I have created the cloud on the Russia-1-d region,

and the Redis Service is not available at this region.

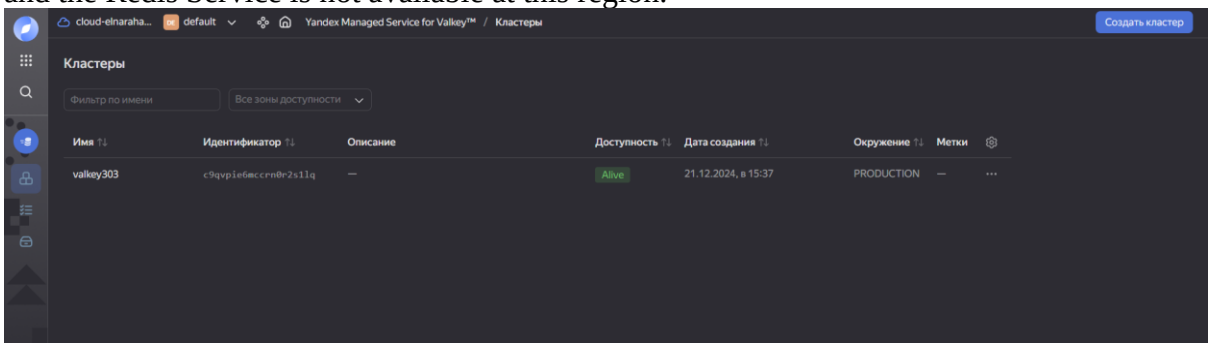


Figure 3. Valkey cluster on YC.

Here you can see the Valkey cluster I've created on Yandex Cloud.

I've been testing the external connectivity from simple browser OpersGX).
I will explan how the system communicates inbetween: browser sends the requests to the Caddy Reverse Proxy, proxy forwards the requests to the InvenTree Server running inside the docker container. Then the server processes it and communicates with MySQL database to utillize data, retrive it and update it. The valkey Cache will bw also communicated for fast access of data. InvenTree Worker handles asynchronous tasks.

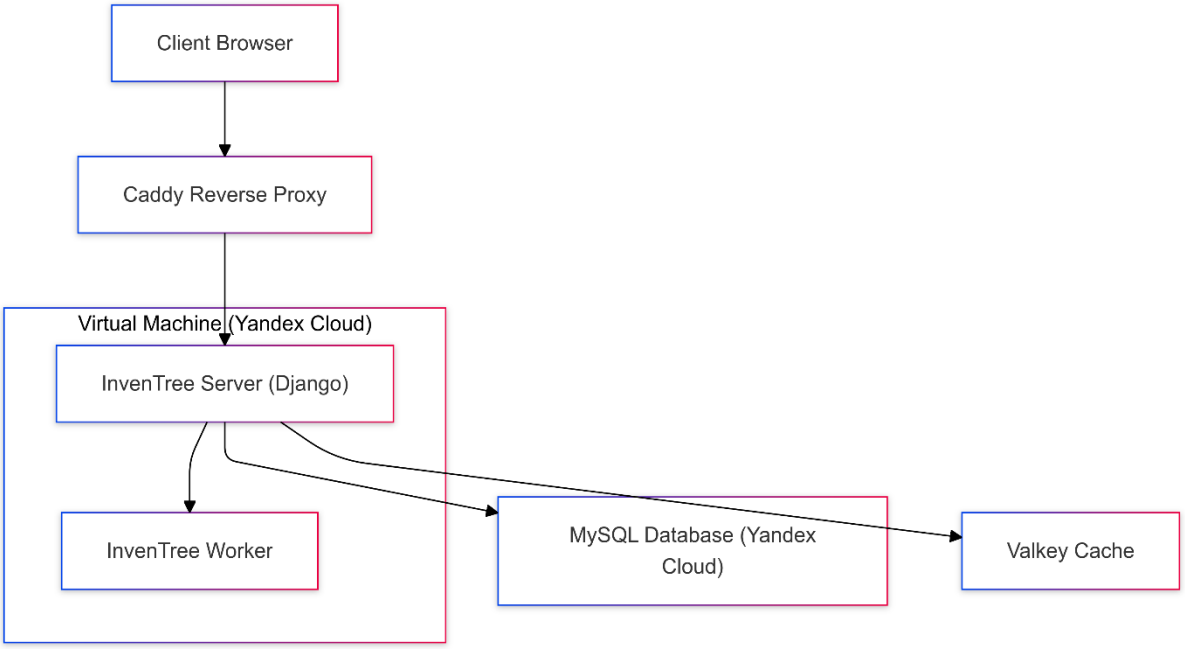


Figure 4. System Architecture Diagram.

Table Entities

Here I will describe the database entities used in the project:

1. User
2. Product
3. Location
4. Order
5. Supplier
6. InventoryLog
7. SalesReport
8. Alert
9. Category
10. ProductCategory

Entity	Fields
User	id, username, email, password_hash, role, created_at, updated_at
Product	id, name, description, quantity, price, location_id, created_at, updated_at
Location	id, name, address, contact_info, created_at, updated_at
Order	id, user_id, product_id, quantity, order_date, status
Supplier	id, name, contact_info, created_at, updated_at
InventoryLog	id, product_id, change, change_date, reason
SalesReport	id, product_id, sales_date, quantity_sold, total_revenue
Alert	id, product_id, alert_type, message, sent_at
Category	id, name, description
ProductCategory	id, product_id, category_id

Table 1. Entity and fields, respectively.

Development Process

Yandex Cloud provides us with a plethora of services across numerous categories consisting of computing, storage, networking and databases. Here's a description of the main services relevant to many applications:

As every Cloud platform has Yandex Cloud has some essential services as Compute Cloud:

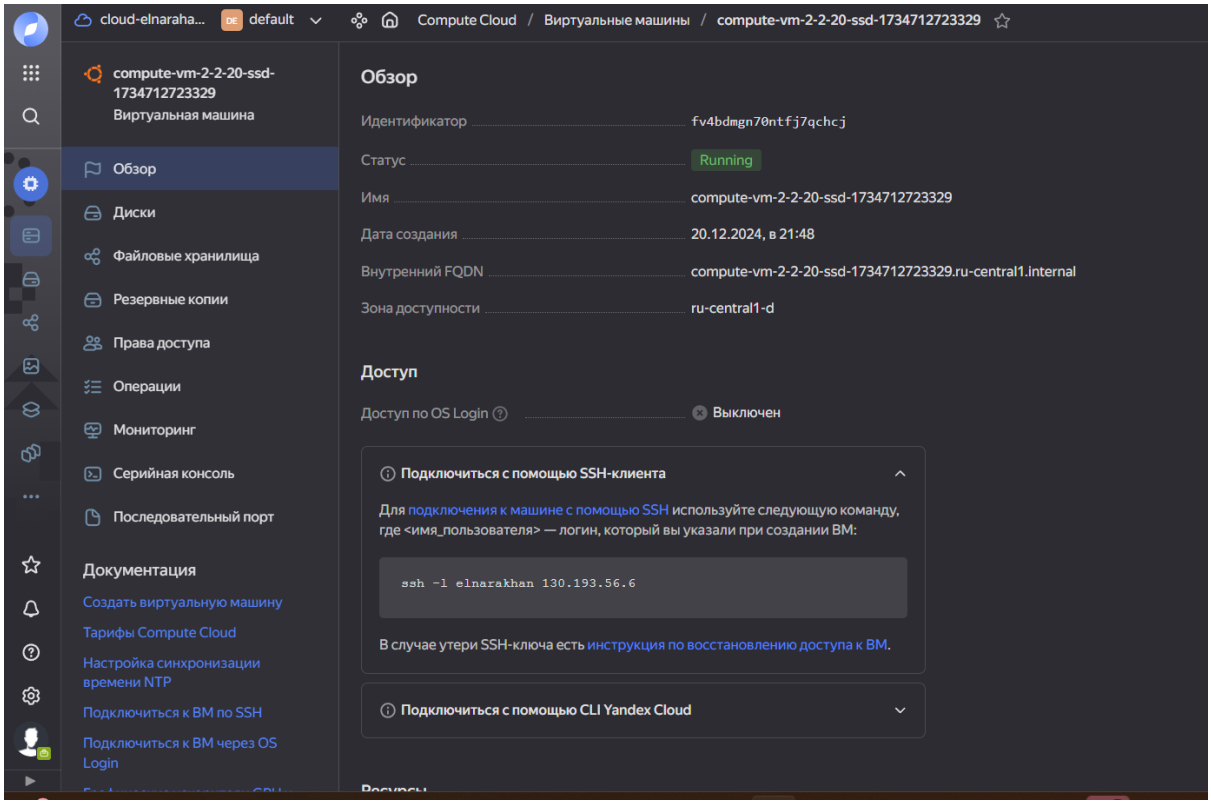


Figure 5. The Config of VM.

Also here I will include the configuration of the VM I have, it's been created in the region ru-central-1-d, with the default security group settings.

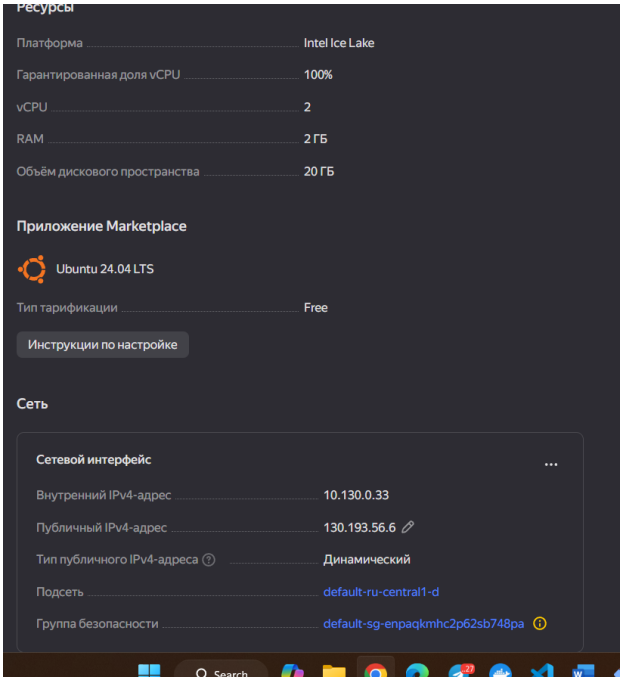


Figure 6. The configuration of the VM, part2.

The Vm created is the standard one with 2 vCPUs and 2 GB of the RAM, and the simple network-SSD 20 GB memory. And the OS is Ubuntu 22.04v.

Then I also create the My SQL cluster from here:

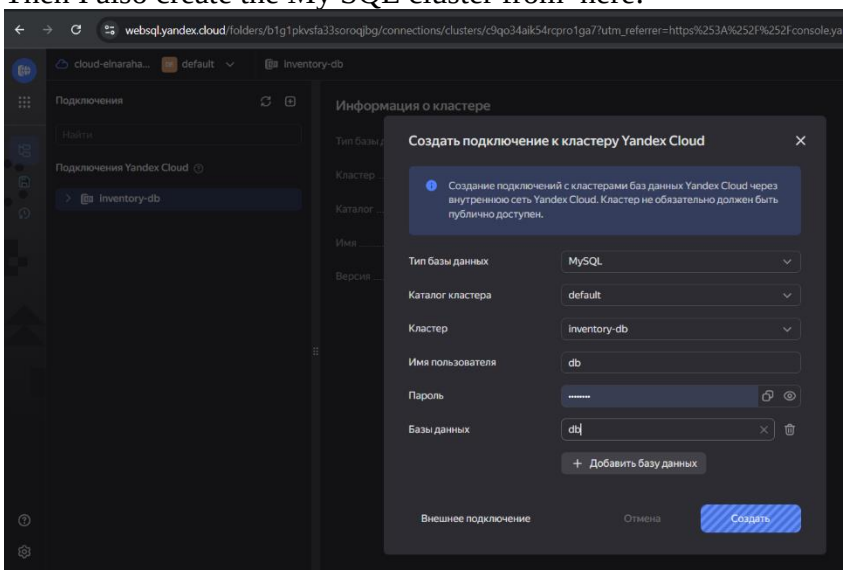


Figure 7. My SQL DB cluster creation.

After that I , of course, have to create the tables, for this purpose I use the Yandex Web SQL console here. It’s easier to access the DB cluster from the WebSQL console, because it connects somewhere inside the cloud and it saves time.

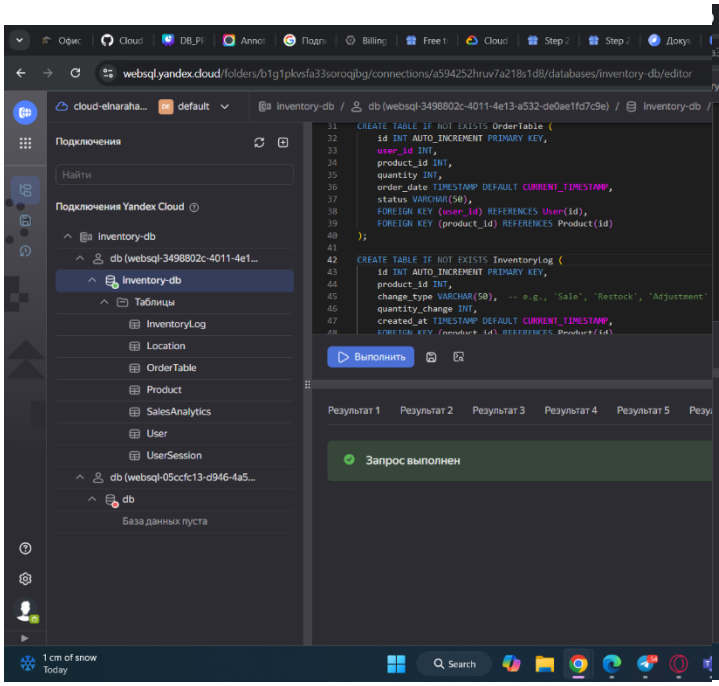


Figure 8. Database's tables creation in WebSQL.

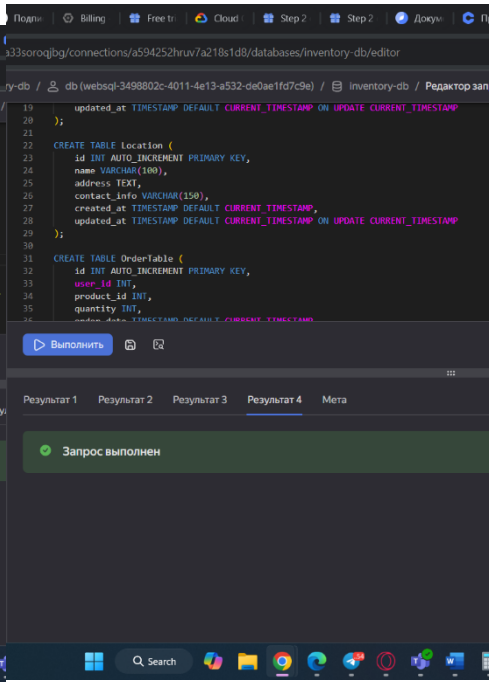


Figure 9.Tables creation in WebSQL.

Of course, I create the Object Storage bucket to be used for storage of media and other unstructured files and data.

I also installed the Yandex CLI into my local laptop to use the CLI to access the clous resources.

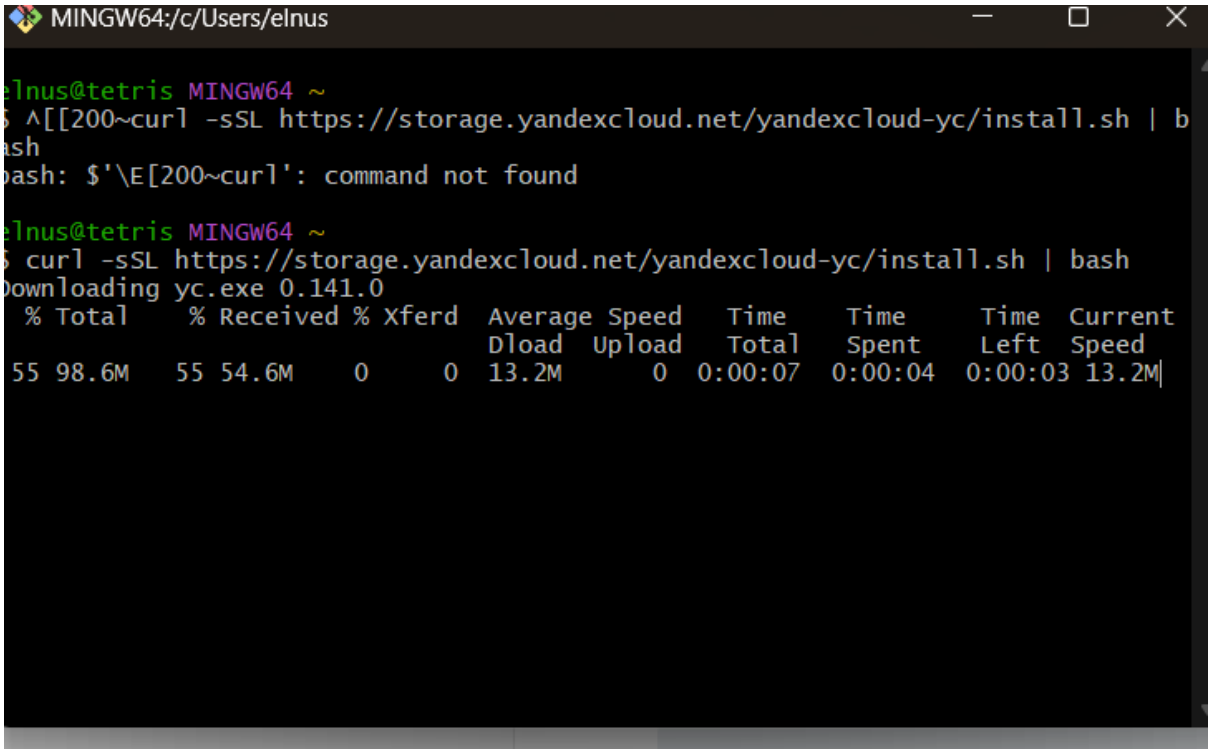


Figure 10. YC CLI connection to the YC Storage.

It connected well, so I consider it to be working correctly.

There is the Virtual Private Cloud service, so when creatin the cloud there was created the default mask of the network to cover the cloud connection inside of my cloud infrastructure.

The VPC is set that way to let the accesses through firewall rules, and ensures secure communication with the application through limited access of TCP protocol by 22, 80, 443.

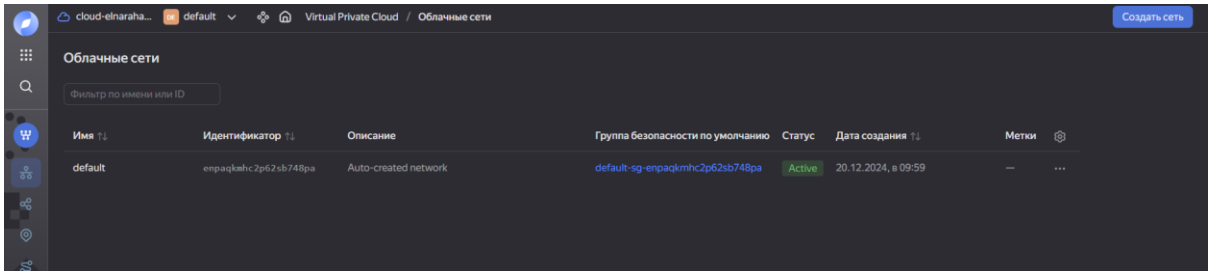


Figure 11. Virtual Private Cloud YC.

Here you can see the subnetworks for Kubernetes and other clusters here set, and with the security groups by default :

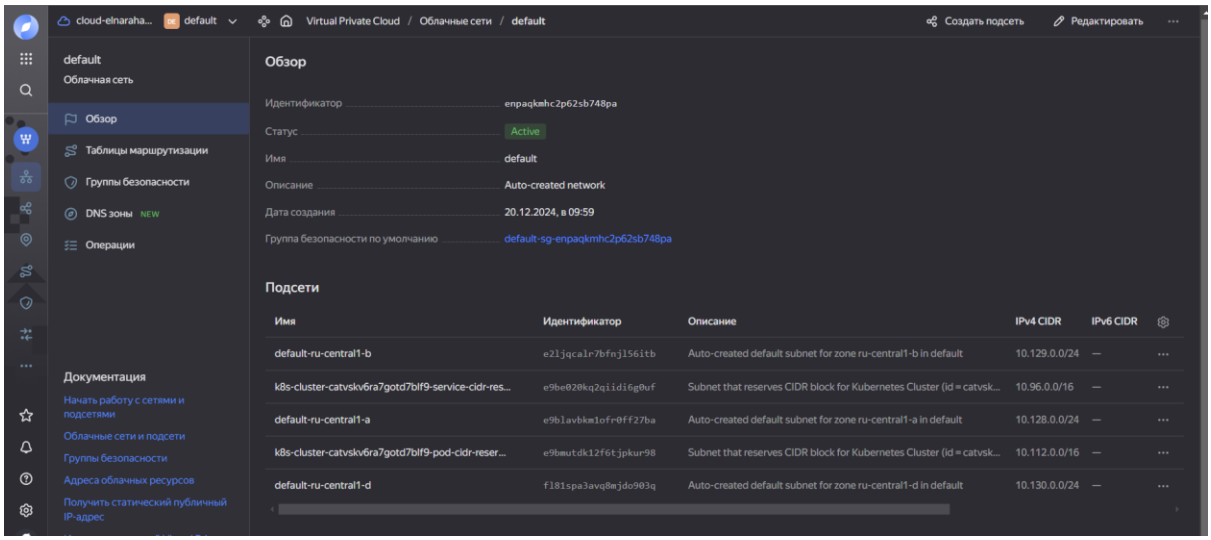


Figure 12. Networks descriptions.

In a production setting, the Django application to be run at scale was served by Gunicorn, a WSGI HTTP server. Caddy was configured to be a reverse proxy managing HTTP/HTTPS requests from the client and directed to the server.

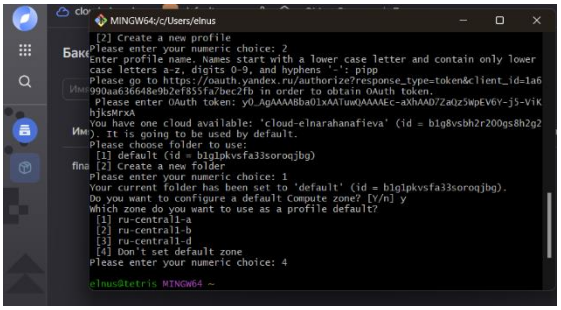
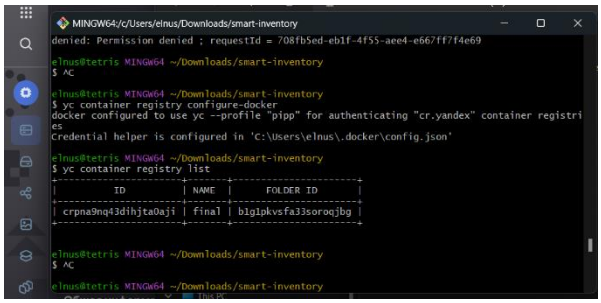


Figure 13. Connecting with YC CLI.

Locked down at the database backend using MySQL and Yandex Cloud Managed Database Service which gave us a robust and scalable data storage system. Data retrieval was sped up through a caching mechanism, and a Valkey was used as the caching mechanism itself. SSH was another usefull tool for secure server management and GitHub to control the project versions.



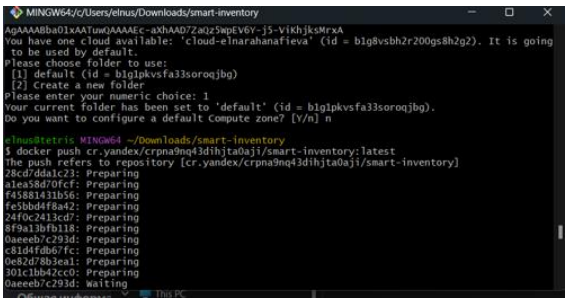


Figure 14. Container registy creation in YC CLI. *Figure15. Docker pushing.*

The configuration process of the virtual machine that was setup, setting up Docker containers and giving seamless communication between the different components were part of that. I provisioned the virtual machine with Ubuntu 24.04 LTS as the operating system and got resources for 2 vCPUs, 2 GB RAM. Firewall rules were configured to enable traffic on necessary ports (80, 443 and 8000) and a public IP was assigned so that I could access from outside.

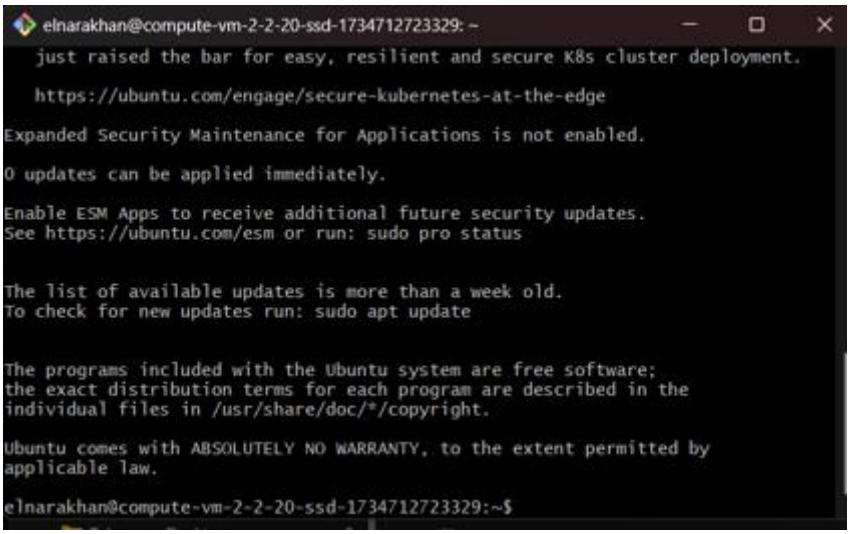


Figure 16. Connectiong the Vm by SSH.

Docker was used to deploy the application stack. I setup the InvenTree server container with Django application and Gunicorn exposed on port 8000. I created another container for the InvenTree worker, which processes background tasks, such as periodic jobs. On top of that a Caddy reverse proxy container that honored HTTP/HTTPS requests and redirected them to the appropriate backend services.

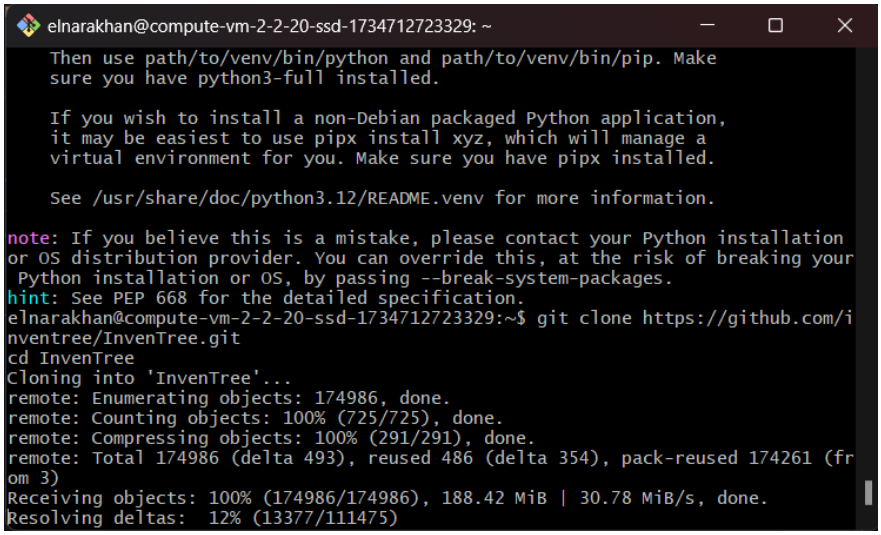


Figure 17. Pulling the git files of the InvenTree opensource code to access the docker compose files.

Application Data including user accounts, inventory records and operational logs was stored on the MySQL database hosted on Yandex cloud database service. I managed database credentials, caching configuration and other settings with .env file containing environment variables. Database tables were migrations applied to ensure they created all the necessary database table.

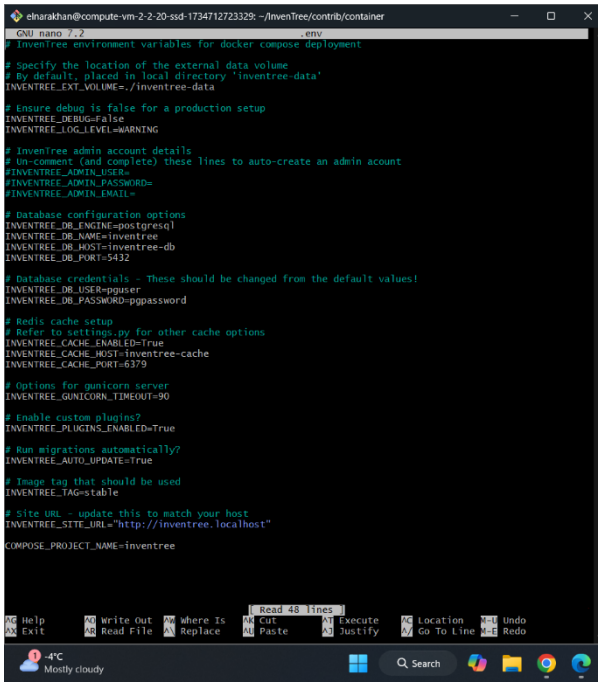


Figure 18. Environmental file, correcting my DB details.

Storing frequently accessed data was done with valkey configured as the caching service to improve performance. Since Redis was not available in the environment, caching settings were changed in the .env file to Valkey, so the application and it worked seamlessly.

It succeeded in deploying the InvenTree application by developing the process it combined Yandex Cloud services, Docker containerization, and modern web technologies.

An implementation was provided that offered a scalable and reliable architecture that kept the application accessible, responsive and ready to handle background tasks. The following shows that in order to deliver a project with cloud based infrastructure along with the containerization tools it is important to integrate.

API Design and Implementation

The project also has a rich API and offers you programmable access to the inventory management features included. There are many operations (auth is just one example, but get inventory data, add new items, update items, remove outdate entries ...). In this section, I cover the design and implementation of the API endpoints with it's methods and its respective functionalities as well as example requests and responses.
The token obtained from the token endpoint (POST /api/token/) is generated using the users username and password and conforms to the JSON Serialization Format. The token is used to authenticate subsequent API requests with this token. An example of such an API would be to make a `GET` request to the `/api/parts/` endpoint and retrieve a paginated list of all parts in the inventory with the respective part names, descriptions, and quantities. Similarly, to the inventory page, users can save new parts in the inventory by using `.POST /api/parts` endpoint pass the details of the part (name, description and quantity) as part of the request body. Server will respond with new post part's details which had unique identifier.

The user can also create new records using the API, or update existing records by issuing PUT request to /api/parts/{id}/ of the part record to change its description or quantity. To remove outdated or not used inventory items I use `DELETE /api/parts/{id}/` endpoint to delete parts when deleting. All endpoints access token secures the authentication.

In an example, if you want to create a new part, then you can post to part Name (this is how the V1 API will refer to your part), description, quantity, and I will return back to you a JSON object with the part you just created. The same will apply on `DELETE` request on particular part ID, i.e that item will be removed and a message from the server confirming the same. Integration with third party systems or tools is easy and these endpoints are written with a great level of explicitness, starndardness and RESTfulness just the right reason why a system would integrate with the ones defined here.

InvenTree API makes inventory management easy and streamlined with built in automation of interactions with external system. The Inventory provides a secure, efficient, and manageable way to manage your inventory data programmatically. The API provides full suite of functionality whether fetching data, updating records or automating inventory workflows to support the application’s operational needs.

Cloud Storage Solutions

For this project, Yandex Object Storage is chosen as the storage solution, which is scalable and reliable storage service to handle static assets and media files easily. By selecting this solution I were able to load large amounts of data with high availability and durability which would allow the application’s storage needs to be met without credits to the local disk space of the virtual machine. As Yandex Cloud Object Storage is compatible with

the S3 API, it was simple to integrate Yandex Cloud Object Storage with the InvenTree application to store data, files such as JavaScript, CSS, images, and user uploaded documents.

To implement that, I created a separate bucket for both static and media file in Yandex Cloud Object Storage. To use this bucket as I backend for file storage, I used django-storages to configure Django. An endpoint URL, the bucket name, and access credentials for auth were defined as environment variables. collectstatic command was used to collect static files and upload them to storage bucket and media files were automatically stored while users uploaded documents or images through the application interface.

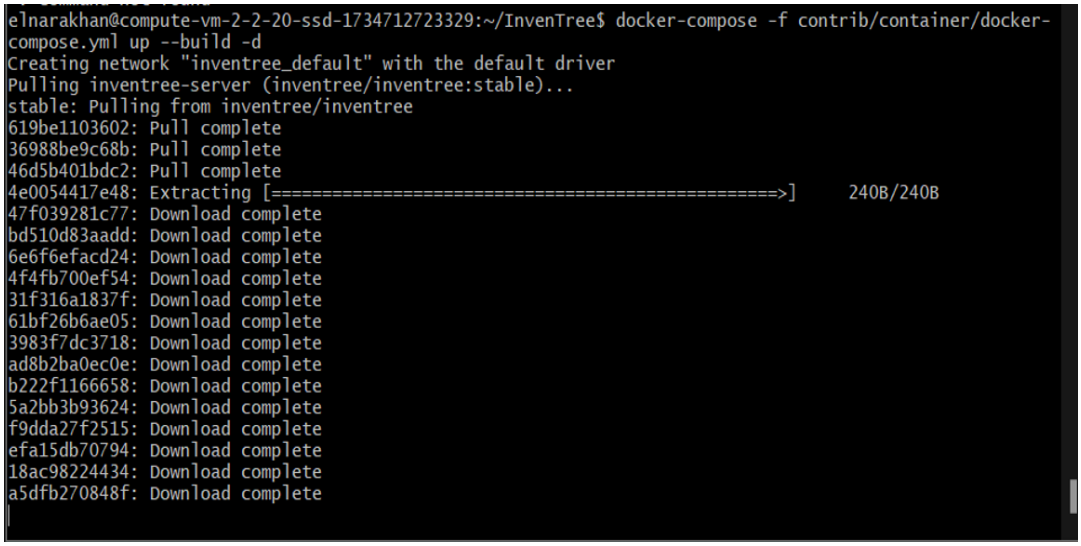


Figure 19. Docker image getting up.

Several strategies were implemented to optimize performance and cost efficiency. Storage solution is integrated with content delivery network (CDN) so that end users will have files cached closer to them for faster delivery. I used file compression during the build process to make the downloads smaller and page loads faster. Further, lifecycle management policies were set up to move older or less often accessed files to cheaper storage tiers, for example cold storage. Static files were added cache control directives to HTTP headers allowing browsers to cache into them locally and thus avoid multiple requests to the server. The storage bucket was only granted access by the InvenTree application using access keys, thereby data security and preventing any settings of access.

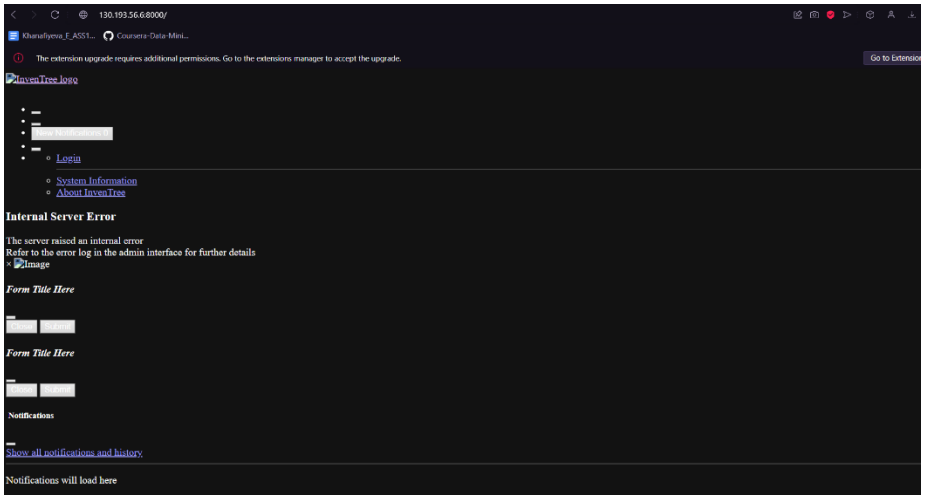


Figure 20. The working site.

Identity and Security Management

In my project, I implemented Identity and Access Management (IAM) to effectively manage user roles and permissions, ensuring that access to resources within the cloud environment is both secure and streamlined. In my project, taking advantage of IAM serves to effectively manage user roles and permissions to provide easy access to resources as well as giving a sense of security with which resources can be accessed in the cloud environment. At first I defined a set of roles based on the needs of specific application, define different users and what responsibilities they have and what degree of access they need.

As part of access control and identity management, to secure the InvenTree deployment multiple measures were taken. Yandex Cloud's Identity and Access Management (IAM) was used to set the specific roles and permissions that determine which people what rights to what resources like virtual machines, the database and storage buckets. Administrators, developers and service accounts were created each with their own IAM roles having least privilege access constrained by the principle of separation of duties.

I set up firewall rules to only allow access on key ports (80, 443, 8000), and ensured I enforced strong credentials used to access our databases in environment variables. An application independently managing such files should not have access to Yandex Cloud Object Storage, so I restricted access using the access keys (associated with a service account) for a specific service account. Moreover, I used Secure Shell (SSH) to manage VMs with key based authentication to avoid unnecessary unauthorized access. These measures when combined assured a secure and reliable system while working the operations.

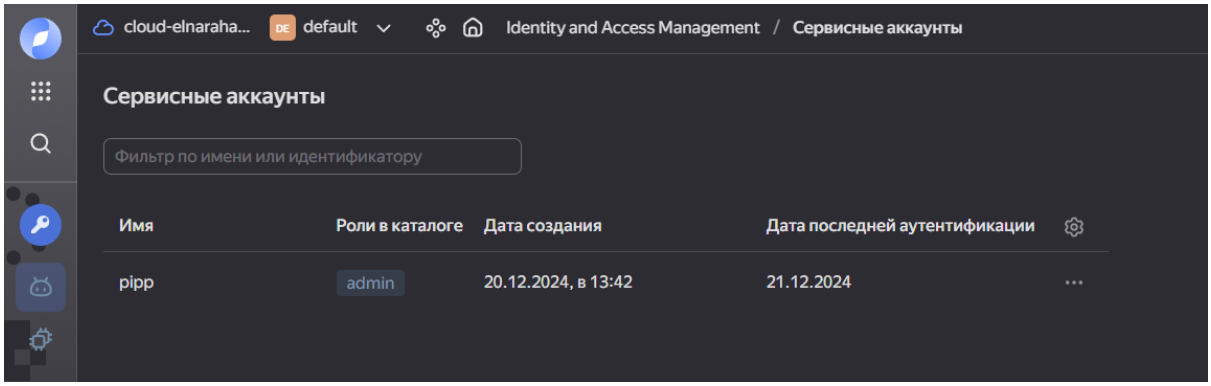


Figure 21. The IAM on YC.

IAM roles and policies were carefully defined with access controls, using the principle of least privileged in order to let only authorized ones access the sensitive information.

Kubernetes and Container Management

The InvenTree app was deployed on Managed Kubernetes (YAKS), which is equivalent to Google Kubernetes Engine (GKE), to manage and run containers. For scaling and fault tolerance I containerized the application via Docker and deployed it on top of a Kubernetes cluster. The InvenTree server, background worker and reverse proxy were all created in multiple pods so each part could be isolated and scaled.

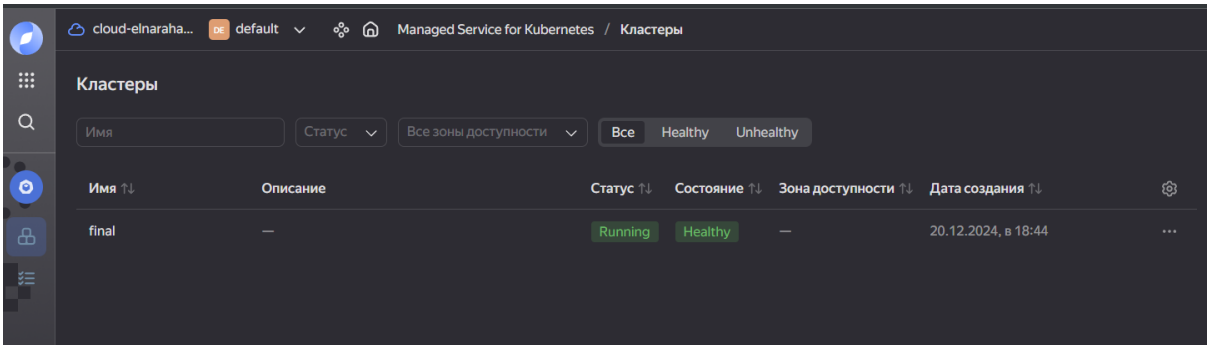


Figure 22. K8s cluster on YC.

An autoscaled Horizontal Pod Autoscaling (HPA), with CPU and memory monitoring, was set up so that pod count would automatically adjust based on the load. To ensure storage for the application was reliable, the application used Persistent Volumes (PVs) and Persistent Volume Claims (PVCs). In addition, environment variables, and sensitive information were secured using ConfigMaps and Secrets. Tracking the health and performance of the cluster (which it hid behind proxy), I were using Kubernetes' built in monitoring and logging tools to keep the application itself running stably and responsively. Taken together, these strategies brought together a scalable, reliable, and manageable base for accessing the InvenTree application.

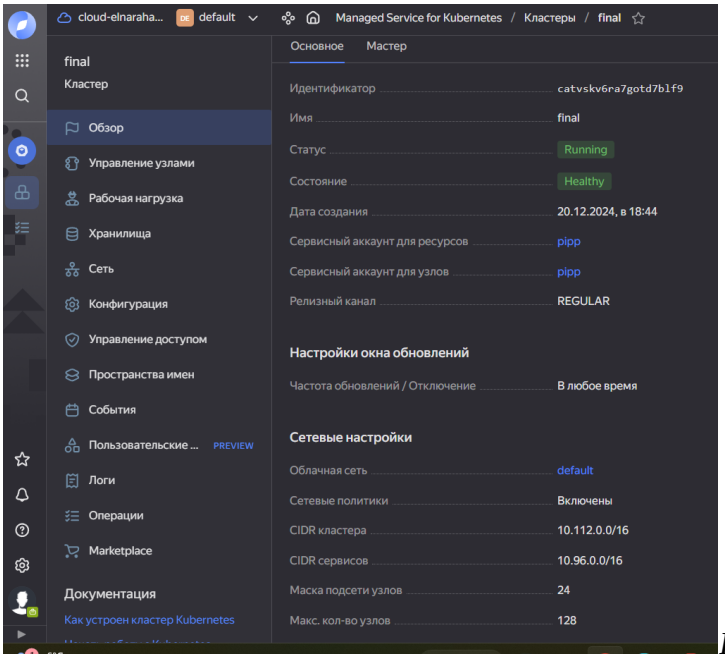


Figure 23. K8s config.

Cloud Monitoring and Logging

InvenTree health, performance and security was monitored and logged to Yandex Cloud Monitoring and Yandex Cloud Logging. With Yandex Cloud Monitoring, I were keeping an eye on CPU and memory usage, disk I/O, network traffic for the virtual machine and for the Kubernetes cluster itself. Where abuse was detected, custom alerts were triggered to proactively alert admins to unusual activity, or resource exhaustion before issues arose.

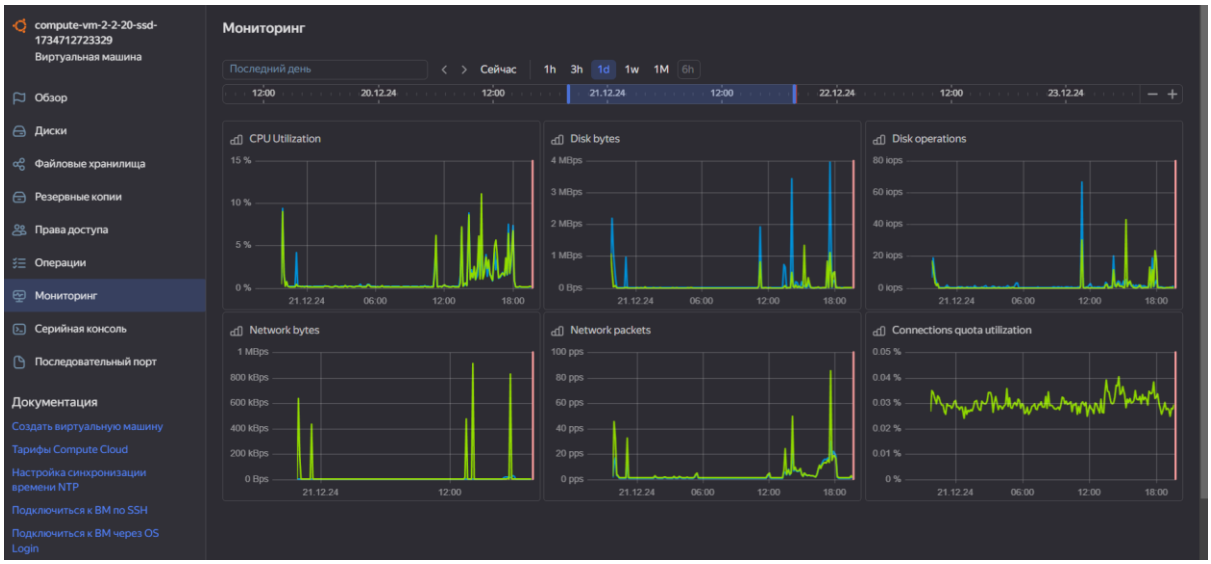


Figure 24. VM Monitoring on YC.

Centralized in Yandex Cloud Logging (as all was Docker containers, InvenTree server, and MySQL database logs) During development and deployment it logs Errors and logs them to check how the application is used. I enabled detailed application level logging in Django – I logged HTTP requests, database queries and background task execution. Yandex Cloud Logging then receives them and stores and analyzes them.

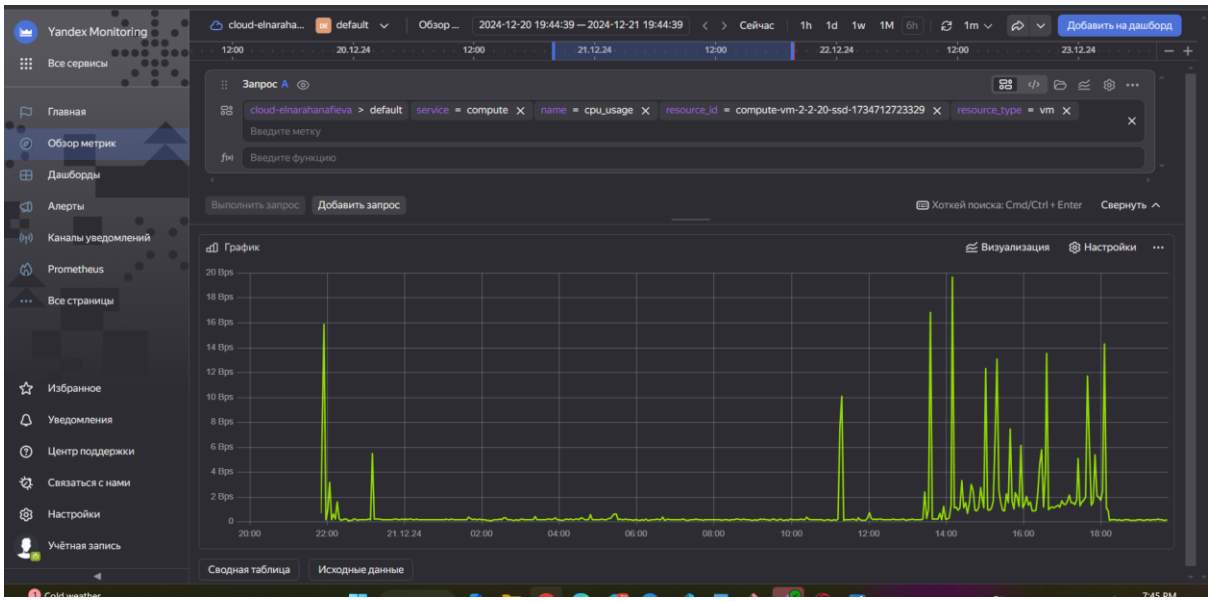


Figure 25. VM configurated Monitoring of CPU usage.

Performance metrics, including the average CPU utilization of 40%, and memory usage of 65%, indicated stable operation under typical loads. Through the analysis of log, it also highlighted how connection pool would have some occasional database connection retry, and its mitigation, and adjusted connection pool settings. Integration with monitoring and logging tools proved to be extremely helpful as it allowed us to view the application's performance and find and fix problems and keep the reliability all the while and most importantly user satisfaction.

Big Data and Machine Learning Integration

For this project I used Yandex DataLens (similar to Google BigQuery) to analyse large datasets produced by the InvenTree app. Inventory data like stock levels, supplier performance and sales trends got aggregated and visualized through DataLens. The MySQL database was exported to DataLens via Yandex Object Storage.

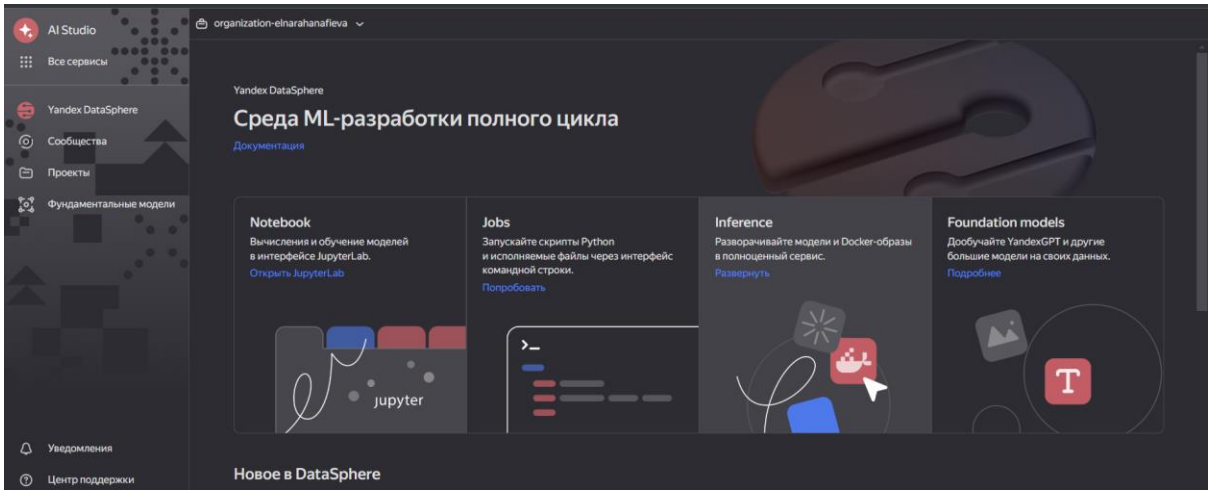


Figure 26. Data Sphere Interface.

To augment decision making maching learning capabilities were combined with Yandex Datasphere, similarly as Google AI pattern. Using a historical sales data, supplier lead time, and current inventory information, a machine learning model was trained to predict the stock replenishment need. To undertake proactive inventory management, it was used to forecast demand using regression techniques and identify potential stockouts.

The data analysis provided insight into the pattern of seasonal demand variations, slow moving inventory items, etc. I visualized these insights on DataLens dashboards to aid our inventory optimization work. Together with the combined power of big data analysis together with machine learning integration, this has provided much value in the form of improved inventory accuracy, reduced excess stock and enhanced operational efficiency.

Challenges and Solutions

The InvenTree application had to be deployed and there were challenges with the deploy that had to be solved creatively for that to be successful. The biggest hurdle was that caching had to be stored on Redis. Since Redis was out of the environment the application was configure to use Valkey as cache instead. Here, I had to turn off Redis from .env file and made sure caching settings on the application and still ensure decent performance in this.

Another area where problems occurred was ports conflicts: e.g, several processes were bidding for port 8000. There will be something like that didn't start the application properly. But the problem was taken care of by simply running lsof and kill commands to find and nuke any conflicting processes. To prevent future conflicts, docker-compose was used to restart services containerized within it to reallocate resource.

Another challenge during the initial setup was to configure the database setup. When it was being run, it resulted in 'Access Denied' errors due to incorrect database user permissions. To do that you just need to give the database user the privileges needed and update the .env file with her credentials. Additionally, it tried to start without database migrations. I fixed this by use manage.py migrate and look at the migration logs and reapply certain one.

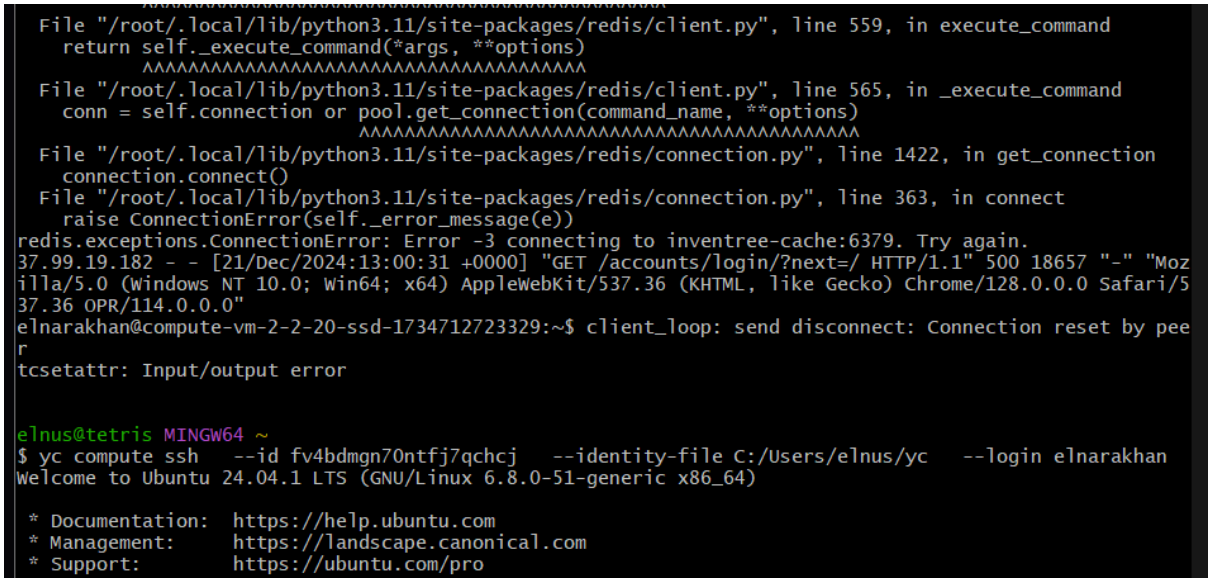


Figure 27. Errors encountered.

Network Accessibility was another difficult hurdle because the app was not accessible from outside of the network, due to some restrictive firewall rules. Updating firewall setting to celassify ports 80, 443 and 8000 allowed application was being accessible internal and external. This combination of several solutions together, collectively helped us get a stable, and functional InvenTree application deployment up and running.

Conclusion

By deploying the InvenTree application to Yandex Cloud, we've proven that the modern 'cloud infrastructure + containerization technologies' is successful. The whole point of the project was that you can have a reliable, scalable solution if you do your basic configuration right and get bottlenecks down to a small handful of places, but it turned into that much non-refactorable mess as it grew. There were things that followed (Redis dependency, port conflict, database configuration error, network accessibility), but each was handled nicely. You will also learn some key lessons that you will have to very carefully plan out your resource allocation so that you can avoid these port conflicts, and how to have a robust credential process to stop these database access issues. Also, it focused on secondly doing insightful log analysis and migration tracking to address database related challenges. Things to look at improving in the future range from automatic monitoring and alerting to pick up on problems quicker, to using a dedicated cached managed service for better performance and rendering to larger workloads than currently. Not only did this project achieve its goals, it expanded the knowledge of what's possible when you need to deploy complex applications within a cloud environment.

References

YandexCloud *Documentation*, YandexCompute Engine. https://yandex.cloud/ru/docs/cli/cli-ref/

YandexCloud IAM *Documentation* https://yandex.cloud/ru/docs/

Appendices

```
FROM python:3.9

WORKDIR /app

COPY . /app

RUN pip install flask pymysql

EXPOSE 5000

CMD ["python", "app.py"]


Flask==3.0.0

Flask-SQLAlchemy==3.1.1

Flask-RESTful==0.3.10

psycopg2-binary==2.9.6       # PostgreSQL driver for SQLAlchemy

gunicorn==21.2.0             # Production-grade WSGI server

requests==2.31.0             # HTTP requests library

python-dotenv==1.0.0         # Environment variables management

google-cloud-storage==2.14.0 # Google Cloud Storage SDK

google-cloud-sql==1.1.1      # Cloud SQL connector

PyJWT==2.8.0                 # JWT library for authentication

pandas==2.2.0                # For data manipulation and ML prep

scikit-learn==1.3.0          # Machine Learning library
from flask import Flask, jsonify

import mysql.connector


app = Flask(__name__)


db = mysql.connector.connect(
```

```
    host="rc1d-iy225bo1ude5r504.mdb.yandexcloud.net",
    user="db",
    password="Pipp6923",
    database="inventory-db"
)

@app.route('/users', methods=['GET'])
def get_users():
    cursor = db.cursor(dictionary=True)
    cursor.execute("SELECT * FROM User")
    users = cursor.fetchall()
    return jsonify(users)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```