MLOPS

Release 1

Jan Jansen

Jul 27, 2025

CONTENTS:

CHAPTER

ONE

ABOUT

Warning : This project is in development and not ready for production. The documentation is a work in progress. The document is rather a quick reminder than a full blown manual.

I attended a course, but I learn by doing. So I decided to create a project to apply the knowledge I gained. The project is about investing in the stock market, but also about setting up Kubernetes, Docker, Helm, ML, CI/CD, github actions and other tools.

This project's data is real stockdata. (which is messy and often incomplete, like real data) The data is a large json object and stored in postgres as a JSONB field. (nosql)

Different techniques, learned while attending the course, were applied.

1.1 Timeseries

Stockdata is timeseries data. (sequential data with a timestamp) Stockdata can be considered public data, in contrast to the data of my workplace. However, the project can be applied to other timeseries data (work related) as well.

1.2 Container (cloud)

The software was packed in Docker containers. This allows for deployment on a Kubernetes cluster as well. (helm script provided) The software has been deployed on Azure as well.

1.3 Visual

The data is accessible through a FASTAPI interface, and can be plotted.

1.4 Patterns

As an investor, I'm interested in patterns. Three green periods, followed by a red one. Reoccuring patterns, turnaround pattern … It would be cool to pick a certain pattern out of a haystack. (see : results)

1.5 AI model

synthetic data was used to create a keras model. The purpose is to filter out similar patterns in the database.

It was not entirely to my liking, so I tried a few other techniques (see: results)

1.6 Cloud

This was my introduction to Azure containers as well as ML studio, Azure CLI. I used the data and scripts provided, but I still had to spend a lot of time due to the steep learning curve. It is a way to construct professional ML applications, by providing building blocks. To experiment and get quick results, I used my own environment.

1.7 K8s

Apart from the recommended local setup of a simple k3s cluster, I installed Talos on a server. I used virtual machines, which allowed for easy adding of clusternodes (which would normally have been real servers). Talos is really naked, or stripped down (out of security concerns) , so you need to install all the functionality yourself. I figure, if the application works on Talos, it should work on Azure, whereas the other way around … There is an overhead, but it might prove worth while if moving away from Azure or to assure compatibility or avoid vendor locker in, or … Another advantage might be that the cloudservices are really within your own network!

1.8 Todo / ideas / github actions

Containers are really a way to get up and running quickly. Kubernetes is an extension, which might be the choice in an production environment. I have installed a containerregistry and a github repository on a local kubernetes cluster. This might enable shielding your code from unwanted eyes.

As I noticed, changing code or training a new ML model, might trigger a lot of extra work. It would be cool if a codechange triggered the deployment of an updated container in a kubernetes cluster. I think there is a way to do this locally instead of using github actions, which publishes the containers, but then you would need another cloud service…

CHAPTER

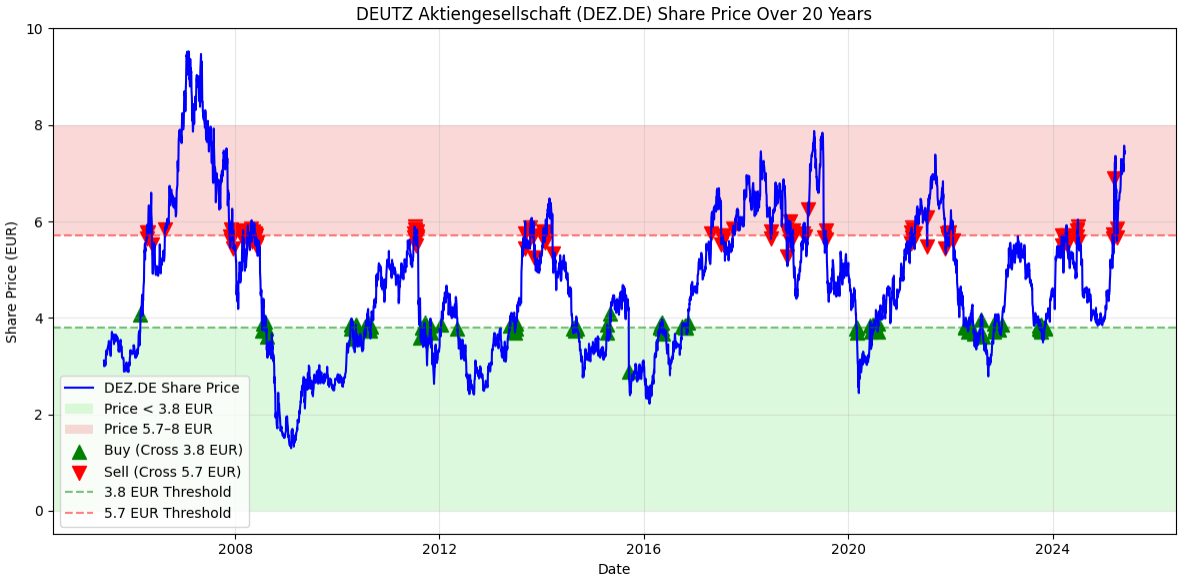
TWO

DEUTZ

as a starting point the stock data of a German company Deutz was used.

Why?

The marketcap is very cyclical. Say history repeats itself, you can make money in buying low, selling high, multiple times!

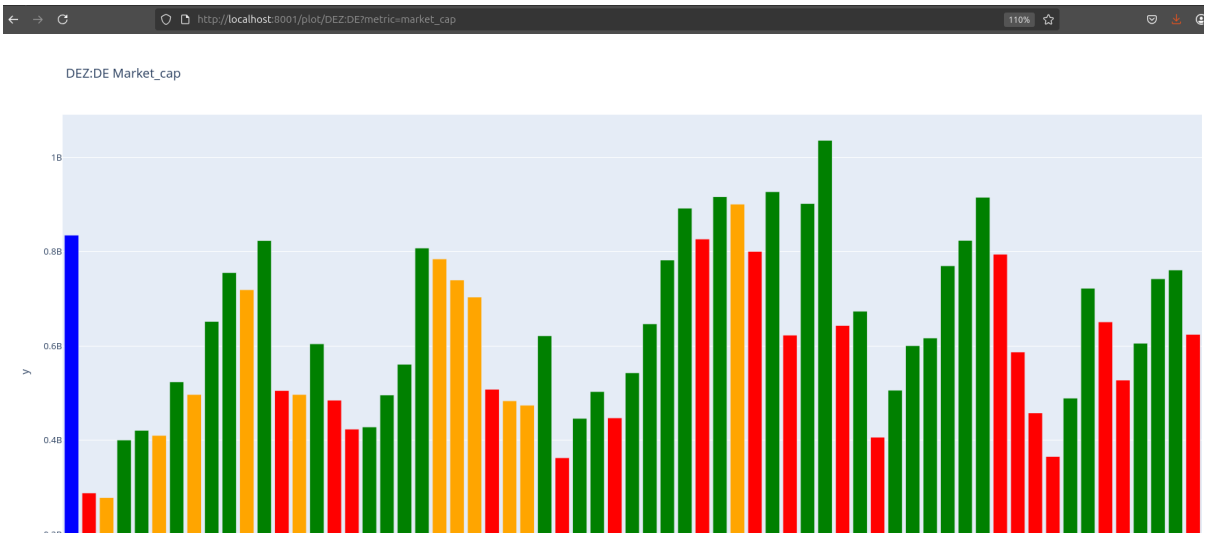


CHAPTER

THREE

MARKET CAP

This is a plot of the market capitalisation per quarter for Deutz. The color green is OK, orange is a drop till 7%, and red is NOK.



CHAPTER

FOUR

RANDOMIZER

This application plots a graph that mimics the real curve. A pythonscript that states 4 times above 80% and the quarters being 7 quarters apart. The visual plot is handy to check the performance of the script. As I found out later, it is much easier to let an LLM describe the differences, based on the distinction have it generate a prompt to have it in another go, generate a python script.

4.1 labelling

While using synthetic data, data that fits a certain pattern, we can attach a label to this data. Each plot generated this way can get the label “matches pattern”

4.2 Training data

We can generate as much data as we want, both data that fits the the pattern, and completely random data (label no match) This method allows to train a model (keras)

4.3 Filter

OK, we have a model, and we can verify our real data (starting with Deutz, which should match the pattern)

CHAPTER

FIVE

OVERVIEW

- ana_report

this contains the Docker container with a FastAPI to consult&plot the data

- ana_model

here the model is generated, this is used for experimenting with an LLM

- ana_github_actions

the purpose is to use github actions to build the Docker container. I want to have github build the model as well, based on an LLM generated python function that describes the disered pattern. This way a user can feed any pattern into the container and filter the database for similar patterns.

- ana_check

this is a script that gets from ana_report all the companies, and then checks if they match the desired pattern, and create an entry in the database

CHAPTER

SIX

RESULTS

The stockdata of Deutz, is in a fact timeseriesdata.

6.1 Approach 1

since I have a large database of timeseries data, which is unlabelled. I used the stockdata of Deutz as a starting point. I tried to describe the timeseries in a mathematical way.

I tried different scripts, with a simple description : data exceeds a threshold multiple times and at least 6 quarters apart.

Next I could visualize the data, using a script that randomly applies the criteria.

This was used to train a ML (keras) model.

Once the model in place, I could apply this binary filter on all my timeseries. Of course Deutz should be recognised as having this pattern.

result : too many companies fit the model and do not resemble Deutz :(

6.2 Approach 2

Have an LLM describe the timeseries in a mathematical way, and use this a prompt to generate a script.

result : too many companies fit the model and do not resemble Deutz :(

6.3 Approach 3

Use the real data of Deutz and let it vary slightly in a random way. (say 15%) result : too many companies fit the model and do not resemble Deutz :(

6.4 Approach 4

PyCaret:

PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows. With PyCaret, you spend less time coding and more time on analysis. You can train your model, analyze it, iterate faster than ever before, and deploy it instantaneously as a REST API or even build a simple front-end ML app, all from your favorite Notebook.

PyCaret was used to select the best fitting model for a timeseries : SELECT * FROM public.symbomodel where symbol like 'DEZ:DE' (Naiveforecaster)

Now I can check which other timeseries resembles Deutz: SELECT * FROM public.symbomodel where model like 'NaiveForecaster'

compare to check : http://localhost:8001/plot/DEZ:DE?metric=market_cap http://localhost:8001/plot/ATLO:US?metric=market_cap

6.5 Approach 5

dtaidistance : Time series distances: Dynamic Time Warping (fast DTW implementation in C)

https://github.com/wannesm/dtaidistance

This is a python library (as well as cypython library), which calculates the distance to a given timeseries (Deutz). The smallest distance would be those share that resemble Deutz the most.

CHAPTER

SEVEN

CONCLUSION

This project aimed to develop a containerized, cloud-based application to identify stocks with behavior similar to Deutz using a NoSQL PostgreSQL database with JSONB data. Five distinct approaches were explored to analyze and match time series patterns, including threshold-based filtering, machine learning with Keras, LLM-generated scripts, randomized data variation, PyCaret's automated model selection, and Dynamic Time Warping (DTW) with the dtaidistance library. Despite these efforts, none of the approaches yielded conclusive results, as each identified too many companies that did not closely resemble Deutz's stock behavior.The primary challenge was the lack of specificity in the models, likely due to the unlabelled nature of the large time series dataset and the complexity of capturing Deutz's unique stock behavior mathematically. While tools like PyCaret and dtaidistance offered efficient workflows and robust distance metrics, the results suggest that more refined feature engineering, additional labeled data, or hybrid approaches combining domain-specific criteria with advanced machine learning techniques may be necessary. Future work could focus on incorporating external market indicators, fine-tuning model parameters, or leveraging more sophisticated time series clustering methods to improve pattern recognition accuracy. This project highlights the challenges of time series similarity search in real data and underscores the need for iterative refinement in such applications.

CHAPTER

EIGHT

THINGS LEARNED

The primary objective of this educational project was getting acquinted with the cloud (Azure), techniques like containerisation, kubernetes, and continuous development and integration. It cannot be overstated that learning hands on, yields enormous results in my personal case. Moreover, much of the effort, could not go unwasted in the workplace.

CHAPTER

NINE

DOCKER PRACTICE

9.1 changes to python code within a container

after making changes to the python code, you need to rebuild the image and restart the container

```
docker-compose -f compose.yml up -d
```

9.2 error & solution

if you get an error like this:

ERROR: for nosql_ana_report_1 'ContainerConfig'

```
docker-compose -f compose.yml down --remove-orphans
docker system prune -f
docker system prune --volumes -f
```

9.3 requirements.txt

if you add new packages to the python code, you need to update the requirements.txt file In a docker container you will only need those packages that are required to run the code.

```
first: install pipreqs
pip install pipreqs

in the container diretory where your python code is located
pipreqs . --force
the --force flag overwrites any existing requirements.txt file in the directory. Without␣
↪this flag, pipreqs will not overwrite an existing file and will throw an error if one␣
↪already exists.
```

9.4 moving a database to another machine

In principle a postgres database container has a volume in docker. This volume can be copied to another machine. But this proves complicated. A simple solution is exporting the database to a file and importing it on the other machine.

```
(import the container)
docker load -i /path/to/destination/nosql_api_latest.tar
docker run -d --name nosql_api_1 -p 9080:9080 nosql_api:latest
```

(continues on next page)

(continued from previous page)

```
or
docker run -d --name nosql_api_1   -e POSTGRES_HOST=172.17.0.5   -e POSTGRES_USER=myuser␣
↪  -e POSTGRES_PASSWORD=mypassword   -e POSTGRES_DB=mydatabase   -e POSTGRES_PORT=5432 ␣
↪ nosql_api:latest
(import the database)
docker exec -i postgres-postgres-1 psql -U postgres mydatabase < db_backup.sql
```

CHAPTER

TEN

USING DOCKERHUB

Dockerhub is a public registry for Docker images. It is the default registry used by Docker when you run the *docker pull* command. You can use Docker Hub to store and share your Docker images with others. You can also use it to find and download images created by other users. Docker Hub provides a web interface for browsing and searching for images, as well as a command-line interface for managing your images and repositories.

10.1 installing custom database on Talos (kubernetes)

https://hub.docker.com/repository/docker/buticosus/my-postgres-image/general

docker run –name myuser -e POSTGRES_PASSWORD=mypassword -p 5432:5432 -d buticosus/my-postgres-image

CHAPTER

ELEVEN

CLOUD SETUP DOCUMENTATION

This document provides an overview of the files used to set up Docker containers and Kubernetes deployments for the project. It explains the purpose of each file and how to use them.

11.1 Docker Setup

1. **`Dockerfile`** - **Purpose**: Defines the instructions to build the Docker image for the API service. - **Usage**: The *Dockerfile* is used by Docker to create a containerized environment for the API. It includes the necessary dependencies and configurations for running the application.

   Command to build the Docker image: `bash docker build -t my-api-image . `

2. **`compose.yml`** - **Purpose**: Defines the Docker Compose configuration for running multiple services (e.g., PostgreSQL, pgAdmin, and the API) together. - **Usage**: This file is used to orchestrate the services required for the project.

   Command to start the services: `bash docker-compose up -d `

   Services included: - *db*: Runs a PostgreSQL database container. - *pgadmin*: Runs a pgAdmin container for managing the database. - *api*: Runs the API service container.

3. **`.env`** - **Purpose**: Stores environment variables used by the *compose.yml* file and the application. - **Usage**: Update this file with the required environment variables (e.g., database credentials).

   Example: `plaintext POSTGRES_USER=myuser POSTGRES_PASSWORD=mypassword POSTGRES_DB=mydatabase `

11.2 Kubernetes Setup

1. **`postgres-pv.yaml`** - **Purpose**: Defines a PersistentVolume (PV) for PostgreSQL to store data persistently in Kubernetes. - **Usage**: Apply this file to create a PV in the Kubernetes cluster.

   Command: `bash kubectl apply -f postgres-pv.yaml `

2. **`postgres-pvc.yaml`** - **Purpose**: Defines a PersistentVolumeClaim (PVC) to request storage from the PersistentVolume for PostgreSQL. - **Usage**: Apply this file to create a PVC in the Kubernetes cluster.

   Command: `bash kubectl apply -f postgres-pvc.yaml `

3. **`postgres-deployment.yaml`** - **Purpose**: Defines the deployment and service for the PostgreSQL database in Kubernetes. - **Usage**: Apply this file to deploy PostgreSQL in the Kubernetes cluster.

   Command: `bash kubectl apply -f postgres-deployment.yaml `

4. **`pgadmin-deployment.yaml`** - **Purpose**: Defines the deployment and service for pgAdmin in Kubernetes. - **Usage**: Apply this file to deploy pgAdmin in the Kubernetes cluster.

Command: `bash kubectl apply -f pgadmin-deployment.yaml `

5. **`api-deployment.yaml`** - **Purpose**: Defines the deployment and service for the API in Kubernetes. - **Usage**: Apply this file to deploy the API in the Kubernetes cluster.

   Command: `bash kubectl apply -f api-deployment.yaml `

11.3 General Notes

- **Docker**: Use Docker Compose for local development and testing. It allows you to quickly spin up all required services.
- **Kubernetes**: Use Kubernetes for deploying the application in a production or cloud environment. Ensure that your cluster is properly configured before applying the YAML files.

CHAPTER

TWELVE

KUBERNETES

12.1 activate my own docker image

```
docker images | grep nosql_ana_report
 (nosql_ana_report                    latest                 af6498f8153e   18 hours ago␣
↪    474MB)
(in microk8s the registry is localhost:32000, where 32000 is the nodeport)
docker tag nosql_ana_report:latest localhost:32000/nosql_ana_report:latest
docker push localhost:32000/nosql_ana_report:latest

(check from other machine)
curl http://192.168.0.103:32000/v2/nosql_ana_report/tags/list
```

12.2 Create a Kubernetes Deployment

Create a YAML file to define a Deployment for your image.

```
# nosql-ana-report-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
name: nosql-ana-report
labels:
    app: nosql-ana-report
spec:
replicas: 1  # Adjust as needed
selector:
    matchLabels:
    app: nosql-ana-report
template:
    metadata:
    labels:
        app: nosql-ana-report
    spec:
    containers:
    - name: nosql-ana-report
        image: localhost:32000/nosql_ana_report:latest  # Replace with your registry␣
↪address
        env:
        - name: ENVIRONMENT
```

(continues on next page)

(continued from previous page)

```
        value: "DOCKER"
        - name: POSTGRES_HOST
        value: "postgres"  # Use the service name, not the IP
        - name: POSTGRES_PORT
        value: "5432"  # Use the port number, not a URL
        - name: POSTGRES_USER
        value: "myuser"  # Replace with your actual user
        - name: POSTGRES_PASSWORD
        value: "mypassword"  # Replace with your actual password
        - name: POSTGRES_DB
        value: "mydatabase"  # Replace with your actual database name
        ports:
        - containerPort: 8001  # Adjust to your app's port, if any

---
apiVersion: v1
kind: Service
metadata:
name: nosql-ana-report
spec:
type: ClusterIP
ports:
- port: 8001
    targetPort: 8001
selector:
    app: nosql-ana-report
```

12.3 deploy the image

```
sudo microk8s kubectl apply -f nosql-ana-report-deployment.yaml
```

12.4 Expose the Application

```
# nosql-ana-report-service.yaml
apiVersion: v1
kind: Service
metadata:
name: nosql-ana-report-service
spec:
selector:
    app: nosql-ana-report
ports:
- protocol: TCP
    port: 8001
    targetPort: 8001  # Match your containerPort
type: NodePort  # Use LoadBalancer for cloud setups
```

```
sudo microk8s kubectl apply -f nosql-ana-report-service.yaml
```

12.5 Check the Deployment

```
sudo microk8s kubectl get deployments
sudo microk8s kubectl get pods
sudo microk8s kubectl get services
```

CHAPTER

THIRTEEN

GITIGNORE TRICKS

most likely, you would like to avoid some files getting published on github (especially if you are using a public repository). Here are some common files that you might want to ignore: (files containing api tokens, passwords, etc.)

If you are anything like me, you'll notice this too late in the process.

13.1 trick:

```
git rm -r --cached source/_build
git rm -r --cached source/
git reset HEAD source/
vi .gitignore
git add .gitignore (add the file you do not want to publish)
git commit -m "removed source/_build from git"
git push origin master
```

13.2 Sphinx:

Sphinx has a build directory, in which files are generated. You might want to ignore this directory as well, as it contains many files that change.

CHAPTER

FOURTEEN

SPHINX

sphinx is used to document the project. It generates HTML documentation from reStructuredText files. Or even pdfs, if you want to.

```
sphinx-quickstart (to install sphinx)
make html (to generate the html files)
make latexpdf (to generate the pdf files)
make clean (to clean the generated files)
make livehtml (to generate the html files and keep updating them as you make changes)
```

Note

documenting the python code is done using sphinx as well. there is some extra configuration needed to make this work. see the *conf.py* file for more details.

Note

sphinx version 8.0.0 or higher is required to build the documentation. you can install it using the following command: pip install sphinx>=8.0.0

see modules.rst for more details on the project modules

CHAPTER

FIFTEEN

MARKDOWN

Markdown: A lightweight, human-readable markup language widely used for documentation, blogs, and README files (e.g., on GitHub). It's simple, portable, and supported by many tools.

In a recent post, AI researcher Andrej Karpathy made a bold observation:

"It's 2025 and most content is still written for humans instead of LLMs. 99.9% of attention is about to be LLM attention, not human attention."

He argues that documentation, knowledge bases, and even entire content ecosystems are still designed with human readability in mind, when in reality, large language models (LLMs) are rapidly becoming the dominant consumers of information. In a world where AI is both the reader and interpreter, should we rethink how we write? Should we optimize for AI comprehension first, human engagement second?

Karpathy's argument—that LLMs will soon dominate content attention—supports Markdown over Sphinx RST for its simplicity, ubiquity, and alignment with machine-readable workflows. Markdown's lightweight syntax is easier for LLMs to parse, its ecosystem is broader, and it bridges human and machine needs during the transition to LLM-centric content consumption.

15.1 using sphinx for markdown

15.2 Hugo

hugo is a webserver that can serve markdown documents https://github.com/gohugoio/hugo/releases/tag/v0.147.0

CHAPTER

SIXTEEN

PORTAINER

Community Edition was used.

Portainer is useful on linux as it provides a GUI for managing docker containers and kubernetes!

Usually it is available at *https://localhost:9443* after installation.

16.1 install

```
#!/bin/bash

# Update system packages
#sudo apt update
#sudo apt upgrade -y

# Install Docker (if not already installed)
sudo apt install -y docker.io

# Start and enable Docker service
sudo systemctl start docker
sudo systemctl enable docker

# Create Docker volume for Portainer data
sudo docker volume create portainer_data

# Pull and run Portainer Community Edition
sudo docker run -d \
-p 8000:8000 \
-p 9443:9443 \
--name portainer \
--restart=always \
-v /var/run/docker.sock:/var/run/docker.sock \
-v portainer_data:/data \
portainer/portainer-ce:latest
```

16.2 managing docker

16.3 managing kubernetes

here you hit a limit to what is possible with the community edition. However there still is ways to manage kubernetes clusters with the community edition.

CHAPTER

SEVENTEEN

MANAGE KUBERNETES CLUSTERS WITH PORTAINER

Portainer is useful on linux as it provides a GUI for managing docker containers and kubernetes!

sudo microk8s kubectl apply -f https://downloads.portainer.io/ce2-20/portainer-agent-k8s-nodeport.yaml sudo microk8s kubectl get nodes -o wide

17.1 Add Environment in portainer:

Open Portainer in your browser.

Click Environments (left sidebar or under Home).

Click Add Environment (should be a button at the top-right).

In the wizard, select Kubernetes.

Choose Agent > Kubernetes via NodePort.

Set Name: “microk8s-local”.

Set Environment URL: 127.0.0.1:30778. (here I used the server IP address instead of localhost)

Click Add Environment.

17.2 create a helm chart repository

```
in de webserver directory : eg /var/www/html
create directory charts
cd charts

helm repo index . (this creates the index.yaml file)

URL=http://IP:80/charts
```

CHAPTER

EIGHTEEN

STEPS TO SERVE THE LOCAL IMAGE ON THE KUBERNETES REGISTRY

I'm using several local machines. To avoid pulling the image from Docker Hub every time, I set up a local registry. This allows me to push the image to the local registry and pull it from there. This is especially useful for testing and development purposes. This guide outlines the steps to push a local Docker image to a local Kubernetes registry and ensure it can be accessed by your Kubernetes cluster.

18.1 1. Verify the Local Image

Check that the image exists locally:

```
docker images | grep postgres
```

- Expected output:

  ```
  postgres    latest    c5df8b5c321e   ...   ...
  ```

- If it's not there, pull it:

  ```
  docker pull postgres:latest
  ```

18.2 2. Confirm the Registry Service

Ensure your `registryExternal` service is running and accessible:

```
microk8s kubectl get svc -n <namespace>
```

- Look for `registryExternal` with `5000:32000/TCP`.
- Get the node's IP (e.g., `192.168.0.103`):

  ```
  microk8s kubectl get nodes -o wide
  ```

- Test connectivity:

```
    curl -v http://192.168.0.103:32000/v2/

- Should return ``200 OK`` with ``{}`` if the registry is up.
```

18.3 3. Tag the Local Image

Re-tag `postgres:latest` to match your registry's address:

```
docker tag postgres:latest 192.168.0.103:32000/postgres:latest
```

- Optionally, use a different name (e.g., for your app):

```
docker tag postgres:latest 192.168.0.103:32000/nosql_ana_report:latest
```

- Verify:

```
docker images | grep 192.168.0.103
```

18.4 4. Push the Image to the Registry

Since `192.168.0.103:32000` is HTTP, configure Docker to allow an insecure registry:

- Edit `/etc/docker/daemon.json`:

```
{
    "insecure-registries": ["192.168.0.103:32000"]
}
```

- Restart Docker:

```
sudo systemctl restart docker
```

- Push the image:

```
    docker push 192.168.0.103:32000/postgres:latest

- Or, if re-tagged as ``nosql_ana_report``:
```

```
docker push 192.168.0.103:32000/nosql_ana_report:latest
```

18.5 5. Verify the Image in the Registry

Check that it's available:

```
curl -v http://192.168.0.103:32000/v2/postgres/tags/list
```

- Expected:

```
{"name": "postgres", "tags": ["latest"]}
```

18.6 6. Update Helm Chart (if Needed)

If you pushed as `192.168.0.103:32000/postgres:latest` but your chart expects `nosql_ana_report`, update `values.yaml`:

```
anaReport:
  image:
    repository: 192.168.0.103:32000/postgres  # Or keep as nosql_ana_report if re-tagged
    tag: latest
  replicaCount: 1
```

- Redeploy:

```
microk8s helm uninstall my-report -n default
microk8s helm install my-report ./my-helm-0.1.0.tgz -n default
```

18.7 7. Ensure MicroK8s Can Pull It

Since MicroK8s uses `containerd`, configure it for the insecure registry:

- Edit `/var/snap/microk8s/current/args/containerd-template.toml`:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."192.168.0.103:32000"]
    endpoint = ["http://192.168.0.103:32000"]
```

- Restart:

```
microk8s stop
microk8s start
```

CHAPTER

NINETEEN

HELM ON MICROK8S

sudo microk8s enable helm3 sudo microk8s status

```
microk8s is running
high-availability: no
  datastore master nodes: 192.168.0.103:19001
  datastore standby nodes: none
addons:
  enabled:
    dashboard             # (core) The Kubernetes dashboard
    dns                   # (core) CoreDNS
    ha-cluster            # (core) Configure high availability on the current node
    helm                  # (core) Helm - the package manager for Kubernetes
    helm3                 # (core) Helm 3 - the package manager for Kubernetes
    hostpath-storage      # (core) Storage class; allocates storage from host directory
    ingress               # (core) Ingress controller for external access
    metallb               # (core) Loadbalancer for your Kubernetes cluster
    metrics-server        # (core) K8s Metrics Server for API access to service metrics
    minio                 # (core) MinIO object storage
    rbac                  # (core) Role-Based Access Control for authorisation
    registry              # (core) Private image registry exposed on localhost:32000
    storage               # (core) Alias to hostpath-storage add-on, deprecated
  disabled:
    cert-manager          # (core) Cloud native certificate management
    cis-hardening         # (core) Apply CIS K8s hardening
    community             # (core) The community addons repository
    gpu                   # (core) Alias to nvidia add-on
    host-access           # (core) Allow Pods connecting to Host services smoothly
    kube-ovn              # (core) An advanced network fabric for Kubernetes
    mayastor              # (core) OpenEBS MayaStor
    nvidia                # (core) NVIDIA hardware (GPU and network) support
    observability         # (core) A lightweight observability stack for logs, traces and␣
↪metrics
    prometheus            # (core) Prometheus operator for monitoring and logging
    rook-ceph             # (core) Distributed Ceph storage using Rook
```

Create a new Helm chart helm create my-application

This creates a standard Helm chart structure with: # - Chart.yaml # - values.yaml # - templates/ directory # - You can then manually copy and modify your existing Kubernetes manifests

> **Note**
>
> you can use helmify or kompose to convert your existing Kubernetes manifests to Helm charts. This proved to be no walk in the park. Manual intervention was required to fix the generated Helm charts.

19.1 testing the Helm chart

helm template ./my-helm –debug

19.2 Create a package

helm package ./my-helm

19.3 create a helm chart repository

```
in de webserver directory : eg /var/www/html
create directory charts
cd charts
cp my-helm-0.1.0.tgz .

helm repo index . (this creates the index.yaml file)

URL=http://IP:80/charts
```

19.4 helm repo

```
helm repo add laptop http://192.168.0.103:82/charts
helm repo list
helm search repo my-helm
NAME                CHART VERSION   APP VERSION     DESCRIPTION
laptop/my-helm      0.1.0           1.0.0           A Helm chart for deploying the nosql␣
↪application
```

19.5 helm repo list

```
helm repo list

NAME    URL
bitnami https://charts.bitnami.com/bitnami
runix   https://helm.runix.net
laptop  http://192.168.0.103:82/charts
```

> **Note**
>
> postgres uses persistent volume, and persistent volume claim this proved difficult to use off the shelve Talos did not allow installation under /tmp, so the helm chart was modified to use /var/lib/postgres-data (values.yaml pv: hostPath)

19.6 install postgresql

local helm chart for database : my-postgres-chart

helm search repo laptop NAME CHART VERSION APP VERSION DESCRIPTION laptop/my-helm 0.1.2 1.0.0 A Helm chart for deploying the nosql applicatio… laptop/my-postgres-chart 0.1.0 1.16.0 A Helm chart for Kubernetes

helm install my-postgres laptop/my-postgres-chart

19.7 install pgadmin

problem with pvc!

```
helm repo add runix https://helm.runix.net
helm install pgadmin4 runix/pgadmin4
```

create pv: vi pgadmin-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pgadmin-pv
spec:
  capacity:
    storage: 1Gi  # Smaller than PostgreSQL, adjust as needed
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
  hostPath:
    path: "/var/lib/pgadmin-data"  # Different from /var/lib/postgres-data
```

```
kubectl apply -f pgadmin-pv.yaml
helm uninstall pgadmin4
helm install pgadmin4 runix/pgadmin4\
--set persistence.existingClaim=pgadmin-pvc \
--set persistence.storageClass=manual \
--set tolerations[0].key="node-role.kubernetes.io/control-plane" \
--set tolerations[0].operator="Exists" \
--set tolerations[0].effect="NoSchedule"
```

PROBLEM PERSIST – CREATE OWN CHART

helm install pgadmin4 runix/pgadmin4

–set persistence.enabled=false –set tolerations[0].key=”node-role.kubernetes.io/control-plane” –set tolerations[0].operator=”Exists” –set tolerations[0].effect=”NoSchedule”

19.8 create pgadmin helm chart

```
helm create my-pgadmin-chart
cd my-pgadmin-chart
vi values.yaml ....
```

(continues on next page)

(continued from previous page)

```
dry-run to test :
helm install my-pgadmin . --dry-run --debug
```

```
helm package ./my-pgadmin-chart
cp my-pgadmin-chart-0.1.0.tgz /var/www/html/charts/
cd /var/www/html/charts/
helm repo index .
```

on the talos client :

```
 helm repo update
 helm repo list
 helm search repo laptop

helm install my-pgadmin laptop/my-pgadmin-chart \
  --set persistence.enabled=true \
  --set persistence.accessMode=ReadWriteOnce \
  --set persistence.size=1Gi \
  --set persistence.storageClass=manual \
  --set pgadmin.securityContext.allowPrivilegeEscalation=false \
  --set pgadmin.securityContext.runAsNonRoot=true \
  --set pgadmin.securityContext.capabilities.drop[0]=ALL \
  --set pgadmin.securityContext.seccompProfile.type=RuntimeDefault
```

19.9 dry run (–debug) to test :

helm install my-pgadmin laptop/my-pgadmin-chart -f pgadmin-values.yaml -n default –dry-run –debug > pgadmin-dry-run.yaml

CHAPTER

TWENTY

HELMIFY

20.1 create a repository for helm

sudo microk8s helm create my-helm

20.2 export your current resources

microk8s kubectl get all -o yaml > current-state.yaml

20.3 install helmify (and go)

sudo snap install go –classic go install github.com/arttor/helmify/cmd/helmify@latest

this installs under /home/user/go/bin

add to PATH : nano ~/.bashrc export PATH=”$PATH:/home/yourusername/go/bin”

cat current-state.yaml | helmify my-helm

(this creates helm templates from the current resources)

20.4 flawless

while it seemed like easy sailing, the helmify command did not work as expected. I created a script that applied the helmify command to each resource type separately.

convert_to_helm.sh

CHAPTER

TWENTYONE

KOMPOSE

Kompose is a tool to help users who are familiar with docker-compose move to Kubernetes. It takes a Docker Compose file and translates it into Kubernetes resources.

21.1 Install Kompose

```
curl -L https://github.com/kubernetes/kompose/releases/download/v1.26.1/kompose-linux-
↪amd64 -o kompose
chmod +x kompose
sudo mv ./kompose /usr/local/bin/kompose

# Or with go
go install github.com/kubernetes/kompose@latest
kompose version  # check the version of Kompose   # On Linux
```

21.2 create helm chart

```
kompose convert -f compose.yml --chart
# This will create a helm chart in the current directory

# To create a helm chart in a specific directory
kompose convert -f compose.yml --chart -o my-helm-chart
```

21.3 Misery

Note

The tool cannot handle volumes, so persistent volumes have to be created manually. The tool cannot handle specifics like “depends on”. After adapting compose.yml it still did not work properly.

CHAPTER

TWENTYTWO

TALOS

Talos is a modern OS for Kubernetes. It is designed to be secure, immutable, and minimal. Talos is a self-hosted Kubernetes distribution that runs on bare metal or virtualized infrastructure. Talos is designed to be managed by a central Kubernetes control plane, which can be hosted on the same cluster or on a separate cluster.

talosctl config add my-cluster –endpoints 192.168.0.242

talosctl config info

talosctl config endpoint 192.168.0.242

talosctl gen config my-cluster https://192.168.0.242:6443 –output-dir ./talos-config –force

22.1 new install talos

https://www.talos.dev/v1.9/talos-guides/install/virtualized-platforms/proxmox/

talosctl gen config my-cluster https://192.168.0.218:6443 talosctl -n 192.168.0.169 get disks –insecure (check disks) talosctl config endpoint 192.168.0.218 talosctl config node 192.168.0.218

talosctl apply-config –insecure –nodes 192.168.0.218 –file controlplane.yaml

talosctl bootstrap talosctl kubeconfig . (retrieve kubeconfig) talosctl –nodes 192.168.0.218 version (verify)

export KUBECONFIG=./talos-config/kubeconfig

kubectl get nodes kubectl get pods -n kube-system kubectl get pods -n kube-system -o wide

kubectl describe pod my-postgres-postgresql-0 (is very useful in case the pod does get deployed

https://factory.talos.dev/ (create your custom image)

```
talosctl upgrade --nodes 10.10.10.178 --image  factory.talos.dev/installer/
↪c9078f9419961640c712a8bf2bb9174933dfcf1da383fd8ea2b7dc21493f8bac:v1.9.5
```

watching nodes: [10.10.10.178]

talosctl get extensions –nodes 10.10.10.178

NODE NAMESPACE TYPE ID VERSION NAME VERSION 10.10.10.178 runtime ExtensionStatus 0 1 iscsi-tools v0.1.6 10.10.10.178 runtime ExtensionStatus 1 1 schematic c9078f9419961640c712a8bf2bb9174933dfcf1da383fd8ea2b7dc21493f8bac

22.2 adding worker nodes

Since “longhorn” stores data on more than one node, we need to add more nodes to the cluster.

> talosctl apply-config –insecure –nodes 10.10.10.166 –file worker.yaml talosctl apply-config –insecure –nodes 10.10.10.173 –file worker.yaml

kubectl get nodes -o wide NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME talos-2ho-roe Ready <none> 113s v1.32.3 10.10.10.173 <none> Talos (v1.9.5) 6.12.18-talos containerd://2.0.3 talos-swn-isw Ready control-plane 31d v1.32.3 10.10.10.118 <none> Talos (v1.9.5) 6.12.18-talos containerd://2.0.3 talos-v1x-9s4 Ready <none> 2m18s v1.32.3 10.10.10.166 <none> Talos (v1.9.5) 6.12.18-talos containerd://2.0.3 talos-y7t-8ll Ready worker 29d v1.32.3 10.10.10.178 <none> Talos (v1.9.5) 6.12.18-talos containerd://2.0.3

22.3 label nodes

> kubectl label nodes talos-v1x-9s4 node-role.kubernetes.io/worker=”” kubectl label nodes talos-2ho-roe node-role.kubernetes.io/worker=””

CHAPTER

TWENTYTHREE

KUBERNETES EXTRA

23.1 join the party

```
sudo snap install microk8s --classic
connecting a node to the cluster :
sudo microk8s join 192.168.0.103:25000/ec0284a420056e7a3b427ef767ff6045/a7e4cd55b373
```

23.2 velero

velero is a backup and restore tool for Kubernetes resources. It can be used to backup and restore the state of the cluster, including persistent volumes, namespaces, and resources.

Velero needs a place to store the backup. This can be a cloud storage service like AWS S3, Google Cloud Storage, or Azure Blob Storage. Alternatively, it can be a local storage location. I choose for minIO. MinIO is an open-source object storage server compatible with Amazon S3 APIs.

```
wget https://github.com/vmware-tanzu/velero/releases/download/v1.13.0/velero-v1.13.0-
↪linux-amd64.tar.gz
tar -xzf velero-v1.13.0-linux-amd64.tar.gz
sudo mv velero-v1.13.0-linux-amd64/velero /usr/local/bin/
velero install   --provider aws   --plugins velero/velero-plugin-for-aws:v1.9.0   --
↪bucket backupvoorlaptop   --secret-file ./credentials-velero   --backup-location-
↪config region=minio,s3ForcePathStyle=true,s3Url=http://10.152.183.52:9000   --use-
↪volume-snapshots=false
```

```
KUBECONFIG=/var/snap/microk8s/current/credentials/client.config
velero backup create my-third-backup --include-namespaces default

velero backup logs my-third-backup

sudo microk8s kubectl exec -it deployment/velero -n velero -- sh

velero restore create my-fifth-restore --from-backup my-fifth-backupq
```

23.3 minIO

MinIO is an open-source object storage server compatible with Amazon S3 APIs. It is used to store the backups created by Velero.

23.4 metallb

metallb is a load balancer for bare metal Kubernetes clusters. It provides a network load balancer implementation that can be used to expose services externally in a bare metal cluster.

```
sudo microk8s enable metallb
sudo microk8s kubectl -n metallb-system get pods
```

23.5 registry

A registry is a storage and content delivery system that holds named Docker images, available in different tagged versions. It can be used to store and distribute Docker images. You can use docker hub, but I choose to use a local registry. It can be configured as a kubernetes service.

CHAPTER

TWENTYFOUR

METALLB

helm install metallb metallb/metallb

Configuring MetalLB with Custom Resources MetalLB uses CRs like IPAddressPool and L2Advertisement (for Layer 2 mode) or BGPPeer and BGPAdvertisement (for BGP mode) to manage IP allocation and advertisement. Given your bare-metal Proxmox environment and Traefik's <pending> IP issue, Layer 2 mode with MetalLB is likely the simplest and most appropriate starting point unless you have a BGP-enabled router. Here's how to set it up:

24.1 talos

Solution: Relax Pod Security for MetalLB You can either move MetalLB to a namespace with a privileged policy or adjust the default namespace to allow the speaker to run. Since your Traefik and MetalLB are already in default, let's modify that namespace.

kubectl label ns default pod-security.kubernetes.io/enforce=privileged –overwrite kubectl label ns default pod-security.kubernetes.io/enforce-version=v1.32 –overwrite

24.2 Update: metallb-config.yaml

create seperate network on proxmox, because of talos-bug.

```yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: default-pool
  namespace: metallb-system
spec:
  addresses:
  - 10.10.10.50-10.10.10.59  # Reserve 10 IPs outside DHCP range
  autoAssign: true

---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: default
  namespace: metallb-system
spec:
  ipAddressPools:
  - default-pool
```

CHAPTER

TWENTYFIVE

TRAEFIK

Traefik is a modern HTTP reverse proxy and load balancer that makes deploying microservices easy. It integrates with your existing infrastructure components and configures itself automatically and dynamically. It is designed to work with Docker, Kubernetes, and other orchestration tools. Traefik is a powerful tool for managing and routing traffic to your applications. It provides features like load balancing, SSL termination, and service discovery. It is designed to be easy to use and configure, making it a popular choice for developers and DevOps teams.

25.1 Installation

````
To install Traefik using Helm, follow these steps:
1. **Add the Traefik Helm repository**:
   ```bash
   helm repo add traefik https://traefik.github.io/charts
   ```
2. **Update your Helm repositories**:
   ```bash
   helm repo update
   ```
3. **Install Traefik**:
   ```bash
   helm install traefik traefik/traefik
   ```
4. **Verify the installation**:
   ```bash
   kubectl get pods -n kube-system
   ```
5. **Access the Traefik dashboard**:
   - By default, the dashboard is not exposed. You can access it by port-forwarding:
   ```bash
   kubectl port-forward service/traefik-dashboard 8080:80 -n kube-system
   ```
   - Open your browser and go to `http://localhost:8080/dashboard/`.
6. **Configure Ingress**:
````

```
root@talos-client:~/cluster# helm repo add traefik https://traefik.github.io/charts
"traefik" has been added to your repositories
root@talos-client:~/cluster# helm repo update
```

root@talos-client:~/cluster# helm install traefik traefik/traefik W0401 08:36:54.160390 332 warnings.go:70] would violate PodSecurity "restricted:latest": seccompProfile (pod or container "traefik" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost") NAME: traefik LAST DEPLOYED: Tue Apr 1

08:36:53 2025 NAMESPACE: default STATUS: deployed REVISION: 1 TEST SUITE: None NOTES: traefik with docker.io/traefik:v3.3.4 has been deployed successfully on default namespace !

25.2 Talos

the “proper” Talos approach is to add a worker node talosctl apply-config –insecure –nodes 192.168.0.213 –file worker.yaml

25.3 workaround

```
# traefik-values.yaml
tolerations:
- key: "node-role.kubernetes.io/control-plane"
    operator: "Exists"
    effect: "NoSchedule"

helm install traefik traefik/traefik -f traefik-values.yaml
```

25.4 check

kubectl get ipaddresspool traefik-pool -n default -o yaml NAME AUTO ASSIGN AVOID BUGGY IPS ADDRESSES traefik-pool true false [“192.168.0.200-192.168.0.210”]

kubectl get l2advertisement -n default NAME IPADDRESSPOOLS IPADDRESSPOOL SELECTORS INTERFACES traefik-l2 [“traefik-pool”]

25.5 change the default values

helm show values traefik/traefik > traefik-values.yaml

25.6 make traefik use the docker registry certs

helm upgrade traefik traefik/traefik –namespace default –set tls.stores.default.defaultCertificate.secretName=registry-certs

CHAPTER

TWENTYSIX

SETTING UP CERTIFICATES FOR A DOCKER REGISTRY ON TALOS

26.1 Overview

This guide explains how to configure certificate-based authentication for a private Docker registry running in a Talos Kubernetes cluster, using internal certificates rather than external ones.

26.2 Certificate Setup Approaches

There are several approaches to solve this:

1. Use a self-signed certificate and configure Docker to trust it
2. Set up an internal Certificate Authority (CA) in your cluster
3. Use Kubernetes cert-manager for certificate management

This guide focuses on approach #2, which provides a good balance of security and control.

26.3 Step 1: Create a Certificate for your Registry

First, you'll need to create a certificate for your registry. If you already have an internal CA, you can use it. Otherwise, you'll need to create one.

```
# Create directory for certificates
mkdir -p certs
cd certs

# Generate a private key for your CA
openssl genrsa -out ca.key 4096

# Create a CA certificate
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.crt \
  -subj "/CN=Internal CA"

# Create a private key for your registry
openssl genrsa -out registry.key 4096

# Create a certificate signing request (CSR)

openssl req -new -key registry.key -out registry.csr \
  -subj "/CN=registry.yourdomain.local" \
  -addext "subjectAltName = DNS:registry.example.com"
```

(continues on next page)

(continued from previous page)

```
# Sign the certificate with your CA
openssl x509 -req -in registry.csr -CA ca.crt -CAkey ca.key \
  -CAcreateserial -out registry.crt -days 365 -sha256 \
  -extfile <(echo "subjectAltName = DNS:registry.example.com")
```

> **Note**
>
> Make sure to replace `registry.yourdomain.local` with your registry's actual hostname throughout this guide.

26.4 Step 2: Configure Your Registry with the Certificates

Create a Kubernetes secret with your certificates:

```
kubectl create secret generic registry-certs \
  --from-file=tls.crt=./certs/registry.crt \
  --from-file=tls.key=./certs/registry.key \
  -n default
```

Update your registry deployment to use these certificates:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: registry
  namespace: registry
spec:
  # ...
  template:
    # ...
    spec:
      containers:
      - name: registry
        image: registry:2
        ports:
        - containerPort: 5000
        volumeMounts:
        - name: registry-certs
          mountPath: /certs
          readOnly: true
        # ...
        env:
        - name: REGISTRY_HTTP_TLS_CERTIFICATE
          value: /certs/tls.crt
        - name: REGISTRY_HTTP_TLS_KEY
          value: /certs/tls.key
      volumes:
      - name: registry-certs
        secret:
          secretName: registry-certs
```

26.5 Step 3: Configure Docker Clients to Trust the Certificate

On each machine that will push to your registry:

```
# Copy your CA certificate to the Docker certs directory
# Replace registry.yourdomain.local with your registry's hostname
sudo mkdir -p /etc/docker/certs.d/registry.yourdomain.local
sudo cp ca.crt /etc/docker/certs.d/registry.yourdomain.local/ca.crt

# Restart Docker to apply changes
sudo systemctl restart docker
```

For Talos machines, you'll need to add the CA certificate to the machine configuration:

```
machine:
  files:
    - content: |
        -----BEGIN CERTIFICATE-----
        # Your CA certificate content here
        -----END CERTIFICATE-----
      permissions: 0644
      path: /etc/ssl/certs/registry-ca.crt
```

26.6 Step 4: Test Pushing to Your Registry

Now you should be able to push images to your registry:

```
# Tag an image for your registry
docker tag myimage:latest registry.yourdomain.local:5000/myimage:latest

# Push to your registry
docker push registry.yourdomain.local:5000/myimage:latest
```

26.7 Alternative: Use Insecure Registry

> **Warning**
>
> This approach is not recommended for production environments!

If you're just testing and don't want to deal with certificates temporarily:

1. Configure Docker to use insecure registry:

   ```
   # Add this to /etc/docker/daemon.json
   {
     "insecure-registries": ["registry.yourdomain.local:5000"]
   }
   ```

2. For Talos nodes, add to your machine configuration:

```
machine:
  registries:
    config:
      registry.yourdomain.local:5000:
        tls:
          insecureSkipVerify: true
```

26.8 Troubleshooting

Common issues and solutions:

26.8.1 Certificate Issues

If you encounter certificate verification errors, check that:

- The CA certificate has been correctly copied to all client machines
- The hostname in your registry URL matches exactly what's in the certificate's SAN field
- The certificates haven't expired (check with `openssl x509 -in registry.crt -text -noout`)

26.8.2 Registry Connection Issues

If you can't connect to the registry:

- Verify the registry service is running: `kubectl get pods -n registry`
- Check that the registry service is exposed: `kubectl get svc -n registry`
- Ensure network policies allow traffic to the registry port

CHAPTER

TWENTYSEVEN

REGISTRY

kubectl apply -f registry.yaml

```
apiVersion: v1
kind: Pod
metadata:
name: registry
labels:
    app: registry
spec:
containers:
    - name: registry
    image: registry:2
    ports:
        - containerPort: 5000
    volumeMounts:
        - name: registry-storage
        mountPath: /var/lib/registry
volumes:
    - name: registry-storage
    emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
name: registry
spec:
selector:
    app: registry
ports:
    - protocol: TCP
    port: 5000
    targetPort: 5000
```

registry ingress

kubectl apply -f registry_plain_ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
name: registry
annotations:
```

(continues on next page)

(continued from previous page)

```
    traefik.ingress.kubernetes.io/router.entrypoints: websecure
spec:
rules:
    - host: registry.example.com
    http:
        paths:
        - path: /
            pathType: Prefix
            backend:
            service:
                name: registry
                port:
                number: 5000
```

27.1 second try

kubectl apply -f registry_ingress.yaml

```
```

27.2 pushing to the registry

docker tag my-postgres-db:latest registry.example.com/my-postgres-db:latest

CHAPTER

TWENTYEIGHT

LONGHORN

Longhorn is a cloud-native distributed block storage solution for Kubernetes. It is designed to provide highly available and durable storage for containerized applications. Longhorn is built on top of Kubernetes and leverages its features to provide a seamless experience for developers and operators. It is an open-source project that is part of the Cloud Native Computing Foundation (CNCF) and is designed to work with any Kubernetes cluster, regardless of the underlying infrastructure. Longhorn provides a simple and efficient way to manage storage for Kubernetes applications, making it an ideal choice for organizations looking to adopt cloud-native technologies.

Note

Talos needs the iscsi extension!!

root@talos-client:~/postgres-buticosus# talosctl get extensions ExtensionStatus 1 1 schematic c9078f9419961640c712a8bf2bb9174933dfcf1da383fd8ea2b7dc21493f8bac

10.10.10.178 runtime ExtensionStatus 0 1 iscsi-tools v0.1.6 10.10.10.178 runtime ExtensionStatus 1 1 schematic c9078f9419961640c712a8bf2bb9174933dfcf1da383fd8ea2b7dc21493f8bac

28.1 talos patch

```
cat <<EOF > longhorn-volume-patch.yaml
machine:
kubelet:
extraMounts:
  - destination: /var/lib/longhorn
    type: bind
    source: /var/lib/longhorn
    options:
      - bind
      - rshared
      - rw
```

talosctl patch mc –nodes 10.10.10.178 –patch @longhorn-volume-patch.yaml patched MachineConfigs.config.talos.dev/v1alpha1 at the node 10.10.10.178 Applied configuration without a reboot

helm repo add longhorn https://charts.longhorn.io helm repo update helm install longhorn longhorn/longhorn –namespace longhorn-system –create-namespace

28.2 check

kubectl get pods -n longhorn-system NAME READY STATUS RESTARTS AGE longhorn-driver-deployer-7f95558b85-sx7cq 0/1 Init:0/1 0 114s longhorn-ui-7ff79dfb4-48pwj 1/1 Running 0 114s longhorn-ui-7ff79dfb4-r6zp5 1/1 Running 0 114s

CHAPTER

TWENTYNINE

INSTALLING LONGHORN ON A TALOS CLUSTER

This guide explains how to install Longhorn, a distributed block storage system for Kubernetes, on a Talos cluster.

29.1 Prerequisites

1. A running Talos Kubernetes cluster.
2. *kubectl* configured to interact with the Talos cluster.
3. Sufficient disk space on the nodes for Longhorn storage.

29.2 Installation Steps

1. **Add the Longhorn Helm Repository**: Add the Longhorn Helm repository to your Helm configuration: `bash helm repo add longhorn https://charts.longhorn.io helm repo update `
2. **Create a Namespace for Longhorn**: Create a dedicated namespace for Longhorn: `bash kubectl create namespace longhorn-system `
3. **Install Longhorn Using Helm**: Install Longhorn in the *longhorn-system* namespace: `bash helm install longhorn longhorn/longhorn --namespace longhorn-system `
4. **Verify the Installation**: Check the status of the Longhorn components: `bash kubectl -n longhorn-system get pods ` Ensure all pods are in the *Running* state.
5. **Access the Longhorn UI**: Expose the Longhorn UI using a *kubectl port-forward* command: `bash kubectl -n longhorn-system port-forward svc/longhorn-frontend 8080:80 ` Open your browser and navigate to *http://localhost:8080* to access the Longhorn UI.
6. **Configure Longhorn StorageClass**: Longhorn automatically creates a default *StorageClass*. You can list it using: `bash kubectl get storageclass ` To use Longhorn as the default *StorageClass*, run: `bash kubectl patch storageclass longhorn -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}' `
7. **Test Persistent Volume Claims (PVCs)**: Create a test PVC to verify Longhorn functionality: ```yaml apiVersion: v1 kind: PersistentVolumeClaim metadata:

   name: test-pvc

   spec:

   accessModes:

   - ReadWriteOnce

   resources:

requests:
storage: 5Gi

storageClassName: longhorn

` Apply the PVC: ```bash kubectl apply -f test-pvc.yaml ` Verify the PVC is bound: `bash kubectl get pvc `

8. **Clean Up**: To uninstall Longhorn, run: `bash helm uninstall longhorn --namespace longhorn-system kubectl delete namespace longhorn-system `

29.3 Notes

- Ensure that all nodes in the Talos cluster have additional disks or directories available for Longhorn to use as storage.
- Longhorn requires Kubernetes 1.18 or later.

29.4 References

- [Longhorn Documentation](https://longhorn.io/docs/)
- [Talos Documentation](https://www.talos.dev/docs/)

29.5 remove longhorn

```
kubectl delete namespace longhorn-system
namespace "longhorn-system" deleted
kubectl delete crd $(kubectl get crd | grep longhorn | awk '{print $1}')
```

29.6 running longhorn on 1 NODE

Single-Node Cluster: You have only one node (talos-y7t-8ll), but the volume is configured for 3 replicas. Longhorn requires at least 3 nodes to schedule 3 replicas (one per node).

kubectl -n longhorn-system patch volume pvc-25af21a6-d280-47ba-b19e-2edf408b0c12 -p '{"spec":{"numberOfReplicas":1}}' –type=merge

CHAPTER

THIRTY

ANA-REPORT (CUSTOM POD/SVC)

30.1 deploy-ana-report.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: nosql-ana-report
namespace: default
spec:
replicas: 1
selector:
    matchLabels:
    app: nosql-ana-report
template:
    metadata:
    labels:
        app: nosql-ana-report
    spec:
    containers:
    - name: nosql-ana-report
        image: 10.110.130.109:5000/nosql-ana-report:latest
        imagePullPolicy: Always
        env:
        - name: POSTGRES_HOST
        value: "postgres-postgresql.postgres.svc.cluster.local"
        - name: POSTGRES_PORT
        value: "5432"
        - name: POSTGRES_USER
        value: "myuser"
        - name: POSTGRES_PASSWORD
        value: "mypassword"
        - name: POSTGRES_DB
        value: "mydatabase"
        ports:
        - containerPort: 8001  # Adjust to the port your app uses
        securityContext:
        allowPrivilegeEscalation: false
        capabilities:
            drop: ["ALL"]
        runAsNonRoot: false
        seccompProfile:
```

(continues on next page)

(continued from previous page)

```
            type: RuntimeDefault
```

30.2 service.yaml

30.3 Ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
name: nosql-ana-report-ingress
namespace: default
annotations:
    traefik.ingress.kubernetes.io/router.entrypoints: web, websecure
    traefik.ingress.kubernetes.io/router.tls: "true"
spec:
rules:
- host: nosql-ana-report.example.com
    http:
    paths:
    - path: /
        pathType: Prefix
        backend:
        service:
            name: nosql-ana-report
            port:
            number: 80
tls:
- hosts:
    - nosql-ana-report.example.com
    secretName: nosql-ana-report-tls
```

CHAPTER

THIRTYONE

POSTGRESQL

pre-requisites : Longhorn

Why We Need a Stateful Solution for PostgreSQL in Kubernetes (Using Longhorn) A stateful solution for PostgreSQL in Kubernetes, using Longhorn, is critical due to the unique requirements of stateful applications like databases. This document explains why such a solution is necessary and how Longhorn addresses these needs. Stateful Nature of PostgreSQL Data Persistence: PostgreSQL stores critical data (e.g., tables, schemas, transaction logs). Unlike stateless applications, it requires persistent storage to prevent data loss during pod restarts or rescheduling.

Consistent Identity: Each PostgreSQL instance (primary or replica) needs a stable network identity and consistent storage to maintain its role in a cluster (e.g., primary for writes, replicas for reads).

Ordered Operations: Operations like provisioning, scaling, or failover must occur in a specific order (e.g., primary before replicas). Kubernetes StatefulSets provide this capability.

Why Kubernetes Needs a Stateful Solution Stateless vs. Stateful: Kubernetes handles stateless workloads with Deployments and ReplicaSets, but stateful applications like PostgreSQL require: StatefulSets: Provide stable pod identities (e.g., pod-name-0), ordered startup/shutdown, and persistent volume bindings.

Persistent Volumes (PVs) and Persistent Volume Claims (PVCs): Ensure storage remains tied to a pod, even if rescheduled.

Dynamic Storage Management: Storage must be provisioned and managed automatically to handle pod/node failures or scaling.

High Availability: PostgreSQL's primary-replica setup requires consistent storage for replication and failover.

Role of Longhorn in the Stateful Solution Longhorn is a Kubernetes-native distributed block storage system, ideal for PostgreSQL's persistent storage needs. Its key features include: Distributed and Resilient Storage: Replicates data across nodes, ensuring durability if a node fails.

Supports PostgreSQL's high availability and disaster recovery needs.

Dynamic Provisioning: Integrates with Kubernetes' Container Storage Interface (CSI) to dynamically provision PVs for PVCs.

Simplifies scaling and recovery.

Snapshots and Backups: Supports snapshots for point-in-time recovery (PITR) and backups to external storage (e.g., S3).

Critical for PostgreSQL's data recovery.

Ease of Management: Provides a user-friendly interface and Kubernetes-native integration.

Supports volume expansion for growing databases.

Cross-Node Mobility: Ensures storage volumes are reattached to rescheduled pods, preserving data integrity.

Why Not Use Other Storage Solutions? Local Storage: Tied to specific nodes, making data inaccessible if a node fails. Longhorn's distributed storage avoids this.

Cloud-Specific Storage: Solutions like AWS EBS or Google Persistent Disk limit portability. Longhorn is cloud-agnostic.

Traditional SAN/NAS: Require external management and lack Kubernetes integration, complicating stateful workloads.

How It Works in Practice StatefulSet for PostgreSQL: Defines replicas (e.g., one primary, two replicas).

Assigns unique pod names (e.g., postgres-0) and PVCs for storage.

Longhorn StorageClass: Configured as the provisioner to dynamically create replicated block storage volumes.

Volume Binding: Longhorn binds each pod's PVC to a persistent volume, maintaining data across restarts.

Replication and Failover: Longhorn replicates data across nodes, supporting failover (e.g., promoting a replica to primary).

Benefits of This Approach Reliability: Longhorn's replication and snapshots ensure data durability and recoverability.

Scalability: StatefulSets and Longhorn enable easy scaling of PostgreSQL instances.

Portability: Works across cloud, on-premises, or hybrid environments.

Automation: Reduces operational overhead with dynamic provisioning and orchestration.

Considerations Performance: Longhorn's performance depends on hardware and network. Tune replica count or use faster disks for high-performance workloads.

Resource Overhead: Replication and management consume resources. Ensure sufficient cluster capacity.

Backup Strategy: Configure regular backups to external storage for disaster recovery.

Conclusion A stateful solution for PostgreSQL in Kubernetes, using Longhorn, addresses the database's need for persistent, reliable, and dynamically managed storage. Longhorn's distributed block storage, replication, snapshots, and backups complement Kubernetes StatefulSets, ensuring PostgreSQL runs reliably with high availability and fault tolerance in a cloud-native environment. This approach simplifies management, enhances resilience, and supports Kubernetes' dynamic workloads.

31.1 pvc (persistant volume claim)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: postgres-pvc
namespace: default
spec:
accessModes:
    - ReadWriteOnce
resources:
    requests:
    storage: 1Gi
storageClassName: longhorn
```

31.2 postgresql-values.yaml

```
global:
storageClass: longhorn
primary:
persistence:
    enabled: true
    existingClaim: postgres-pvc
    # No need to specify size, as PVC is predefined (1Gi)
resources:
    requests:
    storage: 1Gi
configuration:
    postgresql.conf: |
    listen_addresses = '*'
    max_connections = 100
    pg_hba.conf: |
    host all all 0.0.0.0/0 md5
auth:
enablePostgresUser: true
postgresPassword: "mypostgres" # Change to a strong password
database: mydatabase
username: myuser
password: "mypassword"
```

31.3 connecting to postgres

helm install postgres bitnami/postgresql –namespace postgres -f values.yaml

values.yaml

myuser mypassword mydatabase

31.4 creating database

```
cp mydatabase_backup.sql postgres-postgresql-0:/tmp/mydatabase_backup.sql -n postgres
exec -it postgres-postgresql-0 -n postgres -- bash
psql -U myuser -d mypassword -f /tmp/mydatabase_backup.sql
```

CHAPTER

THIRTYTWO

CONFIGURING THE AZURE CLI

The Azure CLI is a command-line tool that allows you to manage Azure resources. You can configure the CLI to use different authentication methods, set default values for commands, and customize the output format. This document provides an overview of the configuration options available in the Azure CLI and how to set them up.

32.1 Installation

https://learn.microsoft.com/nl-nl/cli/azure/

first : activate azure (portal)

```
To install the Azure CLI, follow these steps:
1. **Install the Azure CLI**:
   ```bash
   curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
   ```
2. **Verify the installation**:
   ```bash
   az --version
   ```
3. **Login to your Azure account**:
     ```bash

     az login
     ```
 A browser window will open for you to sign in with your Azure credentials
```

-> first need to activate the azure

```
az account show
{
"environmentName": "AzureCloud",
"homeTenantId": "4ded4bb1-6bff-42b3-aed7-6a36a503bf7a",
"id": "76f32639-1556-4f3a-bdb3-833b5946aa46",
"isDefault": true,
"managedByTenants": [],
"name": "MCT",
"state": "Enabled",
"tenantDefaultDomain": "hogeschool-wvl.be",
"tenantDisplayName": "Hogeschool West-Vlaanderen",
"tenantId": "4ded4bb1-6bff-42b3-aed7-6a36a503bf7a",
```

(continues on next page)

(continued from previous page)

```
"user": {
    "name": "jan.jansen@ibz.be",
    "type": "user"
}
}
```

32.2 basic commands

```
# View account info
az account show

# List all resource groups
az group list

# List available regions
az account list-locations

# Get help for commands
az --help
az group --help
```

32.3 prepare environment

(resource group already exist, if not : az group create –name <name> –location <location>)

#create container registry az acr create –resource-group HAC9909-MasterclassDeployingAISolutions –name jjaacr –sku basic

32.4 using dockerhub image

pull from docker and push to azure

```
az acr login --name jjaacr
docker pull buticosus/my-postgres-image:latest

docker tag buticosus/my-postgres-image:latest jjaacr.azurecr.io/my-postgres-image:latest
docker push jjaacr.azurecr.io/my-postgres-image:latest
```

32.5 delete a container

```
az containerapp delete \
  --name postgres-app \
  --resource-group HAC9909-MasterclassDeployingAISolutions \
  --yes
```

32.6 create azure container apps environment

```
az containerapp env create --name jja-env --resource-group HAC9909-
↪MasterclassDeployingAISolutions --location westeurope

az containerapp list   --resource-group HAC9909-MasterclassDeployingAISolutions   --
↪output table
```

Name Location ResourceGroup Fqdn ———— ———— ———————————— ———————————————————— postgres-app West Europe HAC9909-MasterclassDeployingAISolutions postgres-app.internal.wittydesert-4044e50c.westeurope.azurecontainerapps.io ana-report-app West Europe HAC9909-MasterclassDeployingAISolutions ana-report-app.wittydesert-4044e50c.westeurope.azurecontainerapps.io

32.7 Create Azure Container Apps Environment (cursus extract)

```
# Create Container Apps environment
az containerapp env create \
  --name myapp-env-[ns] \
  --location westeurope

# Create PostgreSQL Container App
az containerapp create \
  --name postgres-app \
  --environment myapp-env-[ns] \
  --image myappacrns.azurecr.io/postgres:13 \
  --registry-server myappacrns.azurecr.io \
  --env-vars POSTGRES_USER=user POSTGRES_PASSWORD=password POSTGRES_DB=fastapi_db \
  --target-port 5432 \
  --ingress internal

# Create FastAPI Container App
az containerapp create \
  --name fastapi-app \
  --environment myapp-env-[ns] \
  --image myappacrns.azurecr.io/fastapi-app:latest \
  --registry-server myappacrns.azurecr.io \
  # Look at the environment variables you used in your fastapi-application that you␣
↪created in the previous assignments
  --env-vars DATABASE_URL="postgresql://user:password@postgres-app:5432/fastapi_db" \
  --target-port 80 \
  --ingress external
```

32.8 getting login

```
az acr credential show --name jjaacr --resource-group HAC9909-
↪MasterclassDeployingAISolutions
```

32.9 connect to internally db

```
az containerapp exec --name postgres-app2-jja  --resource-group HAC9909-
↪MasterclassDeployingAISolutions
INFO: Successfully connected to container: 'postgres-app2-jja'. [ Revision: 'postgres-
↪app2-jja--twvwkg2', Replica: 'postgres-app2-jja--twvwkg2-7cc8f44987-r9d7w'].
# psql -U myuser -d mydatabase

---check content db
select count(*) from companies;
```

32.10 starting containers

```
az login
az acr login --name myappacrjja
az containerapp env create --name myapp-env2-jja --location westeurope
```

```
 az containerapp create \
--name postgres-buticosus-app \
--environment myapp-env2-jja \
--image myappacrjja.azurecr.io/my-postgres-image:latest \
--registry-server myappacrjja.azurecr.io \
--env-vars POSTGRES_USER=myuser POSTGRES_PASSWORD=mypassword POSTGRES_DB=mydatabase \
--target-port 5432 \
--transport tcp \
--ingress internal \
--registry-server myappacrjja.azurecr.io \
--registry-username myappacrjja \
--registry-password f4Kl31+A0+p9079IqAgtMf1/4nI/UOnmTvTcFH1fKe+ACRCnl67c \
--min-replicas 1 \
--max-replicas 1


 az containerapp create   --name adminer-app-jja   --environment myapp-env2-jja   --
↪image myappacrjja.azurecr.io/adminer:latest  --env-vars ADMINER_DEFAULT_
↪SERVER=postgres-buticosus-app   --target-port 8080   --ingress external  --registry-
↪username myappacrjja --registry-password f4Kl31+A0+p9079IqAgtMf1/4nI/
↪UOnmTvTcFH1fKe+ACRCnl67c --registry-server myappacrjja.azurecr.io


 az containerapp create \
--name ana-report-app \
--resource-group HAC9909-MasterclassDeployingAISolutions\
--environment myapp-env2-jja \
--env-vars POSTGRES_USER=myuser POSTGRES_PASSWORD=mypassword POSTGRES_DB=mydatabase␣
↪POSTGRES_HOST=postgres-buticosus-app \
--registry-password f4Kl31+A0+p9079IqAgtMf1/4nI/UOnmTvTcFH1fKe+ACRCnl67c \
--image myappacrjja.azurecr.io/ana_report:latest \
--registry-server myappacrjja.azurecr.io \
--target-port 8001 \
--ingress external
```

(continues on next page)

(continued from previous page)

```
az containerapp create   --name ana-report-app   --resource-group HAC9909-
↪MasterclassDeployingAISolutions  --environment myapp-env2-jja   --image myappacrjja.
↪azurecr.io/ana_report:latest   --registry-server myappacrjja.azurecr.io   --env-vars
↪POSTGRES_USER=myuser POSTGRES_PASSWORD=mypassword POSTGRES_DB=mydatabase POSTGRES_
↪HOST=postgres-buticosus-app  --target-port 8001   --ingress external
```

CHAPTER

THIRTYTHREE

MODULE DOCUMENTATION

33.1 correlation.py

This script calculates financial correlations and stores the results in a PostgreSQL database.

Modules:

- psycopg2: For connecting to and interacting with a PostgreSQL database.
- pandas: For handling and processing tabular data.
- dotenv: For loading environment variables from a .env file.
- os: For accessing environment variables.
- numpy: For numerical operations.

Environment Variables:

- POSTGRES_USER: PostgreSQL username (default: 'myuser').
- POSTGRES_PASSWORD: PostgreSQL password (default: 'mypassword').
- POSTGRES_DB: PostgreSQL database name (default: 'mydatabase').
- POSTGRES_HOST: PostgreSQL host (default: 'localhost').
- POSTGRES_PORT: PostgreSQL port (default: '5432').

Functions:

- fetch_data(query): Fetches data from the PostgreSQL database based on the provided SQL query.
- **calculate_codes(data): Calculates codes based on percentage changes in the data.**
  - Code 1: Positive change.
  - Code 0: Small negative or no change (-7% to 0%).
  - Code -1: Large negative change (less than -7%).
- **calculate_correlation(revenue, market_cap, roic): Calculates the correlation between revenue, market cap, and ROIC codes.**
  - Correlation is incremented when all three codes are either 1 or -1.
- calculate_consecutive_ones(data): Calculates the number of periods with more than two consecutive code 1 values.
- main(): Main function that fetches data, processes it, and stores the results in a PostgreSQL table.

Database Schema:

- **Table: company_correlation**

- symbol (TEXT): Primary key representing the company symbol.
- revenue (INTEGER[]): Array of calculated revenue codes.
- market_cap (INTEGER[]): Array of calculated market cap codes.
- roic (INTEGER[]): Array of calculated ROIC codes.
- correlation (FLOAT): Correlation value between revenue, market cap, and ROIC.
- consecutive_ones (INT): Number of periods with more than two consecutive code 1 values.

Usage:

- Ensure the required environment variables are set in a .env file.
- Run the script to calculate financial correlations and store the results in the database.

advisor.correlation.**calculate_codes**(*data*)

advisor.correlation.**calculate_consecutive_ones**(*data*)

advisor.correlation.**calculate_correlation_all**(*revenue*, *market_cap*, *roic*)

advisor.correlation.**calculate_correlation_rev_cap**(*revenue*, *market_cap*)

advisor.correlation.**calculate_correlation_rev_roic**(*revenue*, *roic*)

advisor.correlation.**calculate_correlation_roic_cap**(*roic*, *market_cap*)

advisor.correlation.**fetch_data**(*query*)

advisor.correlation.**main**()

33.2 fastapp.py

33.3 report.py

This script generates a PDF report with financial metrics for companies stored in a PostgreSQL database. It includes functionalities for fetching data, categorizing companies, generating plots, and creating a PDF report. Modules and Libraries: - *psycopg2*: For connecting to and querying the PostgreSQL database. - *pandas*: For data manipulation and analysis. - *dotenv*: For loading environment variables from a *.env* file. - *os*: For accessing environment variables.

ana_report.report.**categorize_company**(*data*)

ana_report.report.**fetch_data**(*query*)

ana_report.report.**generate_pdf**(*df*, *output_file*)

ana_report.report.**generate_plot**(*data*, *title*, *filename*)

ana_report.report.**generate_report**(*output_file='financial_report.pdf'*)

ana_report.report.**get_item_color**(*data*)

ana_report.report.**main**()

33.4 update_postgres.py

33.5 show-json.py

CHAPTER

THIRTYFOUR

GRAPH

fastapp has a built-in graphing library that can be used to plot data from the database.

http://127.0.0.1:8001/plot/BMW%3ADE?metric=market_cap

for each symbol there are 3 metrics: market_cap, price, volume

the bars have a color (red, green, blue) based on the metric value

34.1 swagger

http://127.0.0.1:8001/docs

CHAPTER

THIRTYFIVE

SDA (SOFTWARE DEFINED ARCHITECTURE)

35.1 Physical Infrastructure

35.2 Proxmox Server Specifications

The foundation of the architecture is a physical server running Proxmox VE hypervisor.

Component	Description
Hypervisor	Proxmox Virtual Environment (VE)
Virtual Machines	Multiple Talos OS nodes forming a Kubernetes cluster
Containers	LXC container serving as a router

35.3 Virtual Machine Configuration

Talos OS Nodes

The cluster consists of multiple Talos OS nodes, with dedicated roles:

- **Control Plane Node(s)**: Manages the Kubernetes control plane
- **Worker Nodes**: Runs application workloads

```
# Example Talos configuration structure (simplified)
machine:
  type: controlplane  # or worker
  network:
    hostname: talos-node-1
  kubernetes:
    version: v1.26.0
```

35.4 Networking Architecture

35.4.1 Network Components

Component	Function
Proxmox Virtual Bridge	Creates isolated network segments for VMs and containers
LXC Router	Routes traffic between internal and external networks
Kubernetes Overlay Network	Enables pod-to-pod communication (Cilium, Flannel, etc.)

35.5 Control & Automation

35.5.1 API Management Layer

This architecture leverages multiple declarative APIs for infrastructure management:

API	Responsibility
Proxmox API	Manages physical resources, VMs, and containers
Talos API	Provides declarative OS configuration and maintenance
Kubernetes API	Orchestrates applications and services

35.6 Benefits of This Architecture

- **Immutable Infrastructure**: Talos OS provides an immutable, declarative operating system
- **High Availability**: Kubernetes manages service availability and distribution
- **Resource Efficiency**: Consolidates multiple services on a single physical server
- **Isolation**: Separate network segments and container boundaries
- **Automation**: API-driven management at all levels

CHAPTER

THIRTYSIX

PROXMOX

remove firewall from talos worker node

```
root@pve:~# qm set 109 -net0 virtio,bridge=vmbr0,firewall=0
update VM 109: -net0 virtio,bridge=vmbr0,firewall=0
```

36.1 setting up an extra network bridge vmbr1

on lxc dedicated machine setup dhcp and routing

```
apt install dnsmasq -y
apt install iptables-persistent -y

vi /etc/dnsmasq.conf
interface=eth0
dhcp-range=10.10.10.100,10.10.10.200,12h  # DHCP range for Talos nodes
dhcp-option=3,10.10.10.2                  # Gateway (this machine's eth0 IP)
dhcp-option=6,192.168.0.1                 # DNS (your home router's DNS)

systemctl restart dnsmasq

echo 1 > /proc/sys/net/ipv4/ip_forward

iptables -t nat -L -v
ip route del default via 10.10.10.1 dev eth0
ip route replace default via 192.168.0.1 dev eth1 metric 100

iptables -t nat -A POSTROUTING -s 10.10.10.0/24 -o eth1 -j MASQUERADE
ip route del default via 10.10.10.1 dev eth0
ip route replace default via 192.168.0.1 dev eth1 metric 100



vi /etc/netplan/01-netcfg.yaml
```

```
network:
  version: 2
  ethernets:
    eth0:
      dhcp4: true
```

(continues on next page)

(continued from previous page)

```
      # Prevent DHCP from setting a default gateway if it conflicts
      dhcp4-overrides:
        use-routes: true
        use-dns: true
        route-metric: 2000  # High metric to prioritize eth1's default route
    eth1:
      dhcp4: false
      addresses:
        - 192.168.0.x/24  # Replace with your server's IP on this subnet
      routes:
        - to: 0.0.0.0/0
          via: 192.168.0.1
          metric: 100
        - to: 0.0.0.0/0
          via: 192.168.0.1
          metric: 1024
        - to: 192.168.0.0/24
          via: 0.0.0.0
          metric: 1024
        - to: 192.168.0.1
          via: 0.0.0.0
          metric: 1024
        - to: <gent.dnscache01-ip>
          via: 192.168.0.1
          metric: 1024
        - to: <gent.dnscache02-ip>
          via: 192.168.0.1
          metric: 1024
```

```
# Apply the netplan configuration
sudo netplan generate
sudo netplan apply

# Check the routing table
ip route show

# Check iptables rules
iptables -t nat -L -v

# Check dnsmasq status
systemctl status dnsmasq

# Check if the DHCP server is running and listening on the correct interface
sudo systemctl status dnsmasq

# Restart dnsmasq to apply changes
sudo systemctl restart dnsmasq


netplan apply
```

36.2 Kernel IP routing table

36.3 using the nodeport

192.168.0.251:30743

on my router/dhcp on 10.10.10.2 route port to cluster node IP

iptables -t nat -A PREROUTING -i eth1 -p tcp –dport 30743 -j DNAT –to-destination 10.10.10.118:30743

so running nginx on kubernetes on 10.10.10.255 network is accessible from the outside

36.4 using the IP address

traefik LoadBalancer 10.102.122.212 10.10.10.50 80:32178/TCP,443:32318/TCP 75m app.kubernetes.io/instance=traefik-default,app.kubernetes.io/name=traefik

So now I have to figure out how I can reach 10.10.10.50 from my 192.168.X.X network

on the kubernetes cluster, traefik has been deployed as well as metallb. iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -d 10.10.10.0/24 -j MASQUERADE sh -c “iptables-save > /etc/iptables/rules.v4”

this has been added to th dnsmasq.conf

Listen on the 192.168.0.251 interface interface=eth1 # Replace with your 192.168.0.251 interface (check with *ip a*) listen-address=192.168.0.251

Forward other queries to upstream DNS (e.g., Google DNS) server=8.8.8.8 server=8.8.4.4

Optional: If LXC is your DHCP server, ensure DNS is offered dhcp-option=6,192.168.0.251 # Tells DHCP clients to use this as DNS

36.5 modify dns config on laptop

/etc/resolv.conf

add : nameserver 192.168.0.251

36.6 access http://nginx.example.com/ on talos within 10.10.10.X from 192.168.X.X

(configure metallb, traefik, nginx)

on laptop /etc/hosts : 10.10.10.50 nginx.example.com

on dhcp server (10.10.10.2)

iptables -A FORWARD -s 192.168.0.0/24 -d 10.10.10.0/24 -j ACCEPT iptables -A FORWARD -s 10.10.10.0/24 -d 192.168.0.0/24 -j ACCEPT

```
# Generated by iptables-save v1.8.7 on Thu Apr 10 13:32:59 2025
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A FORWARD -s 192.168.0.0/24 -d 10.10.10.0/24 -j ACCEPT
-A FORWARD -s 10.10.10.0/24 -d 192.168.0.0/24 -j ACCEPT
```

(continues on next page)

(continued from previous page)

```
COMMIT
# Completed on Thu Apr 10 13:32:59 2025
# Generated by iptables-save v1.8.7 on Thu Apr 10 13:32:59 2025
*nat
:PREROUTING ACCEPT [6847:1975161]
:INPUT ACCEPT [158:15156]
:OUTPUT ACCEPT [25:2590]
:POSTROUTING ACCEPT [25:2590]
-A PREROUTING -i eth1 -p tcp -m tcp --dport 30743 -j DNAT --to-destination 10.10.10.
↪118:30743
-A POSTROUTING -s 10.10.10.0/24 -o eth1 -j MASQUERADE
-A POSTROUTING -s 10.10.10.0/24 -o eth1 -j MASQUERADE
-A POSTROUTING -s 10.10.10.0/24 -o eth1 -j MASQUERADE
-A POSTROUTING -s 192.168.0.0/24 -d 10.10.10.0/24 -j MASQUERADE
COMMIT
# Completed on Thu Apr 10 13:32:59 2025
# Check the iptables rules
iptables -t nat -L -v
iptables -L -v

# Check the routing table
ip route show

# Check the network interfaces
ip a
```

CHAPTER

THIRTYSEVEN

LAPTOP (OR PC) MODS

/etc/hosts

10.10.10.50 nginx.example.com 10.10.10.50 registry.example.com

ip route add 10.10.10.0/24 via 192.168.0.251

sudo cp ca.crt /usr/local/share/ca-certificates/ca.crt sudo update-ca-certificates Updating certificates in /etc/ssl/certs...

37.1 make route permanent

```
sudo nano /etc/netplan/01-netcfg.yaml

network:
version: 2
ethernets:
  wlp0s20f3:
    addresses:
      - 192.168.0.103/24
    gateway4: 192.168.0.1  # Replace with your actual default gateway
    routes:
      - to: 10.10.10.0/24
        via: 192.168.0.251

  sudo netplan apply
```

CHAPTER

THIRTYEIGHT

KERNEL IP ROUTING TABLE

The following table represents the kernel IP routing table:

Table 1: Kernel IP Routing Table

Destination	Gateway	Genmask	Flags	Met-ric	Ref	Use	Iface
default	192.168.0.1	0.0.0.0	UG	100	0	0	eth1
default	192.168.0.1	0.0.0.0	UG	1024	0	0	eth1
10.10.10.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
192.168.0.0	0.0.0.0	255.255.255.0	U	1024	0	0	eth1
192.168.0.1	0.0.0.0	255.255.255.255	UH	1024	0	0	eth1
gent.dnscache01	192.168.0.1	255.255.255.255	UGH	1024	0	0	eth1
gent.dnscache02	192.168.0.1	255.255.255.255	UGH	1024	0	0	eth1

CHAPTER

THIRTYNINE

TROUBLESHOOTING KUBERNETES

This section provides troubleshooting steps for common issues encountered in Kubernetes deployments, particularly focusing on PersistentVolumeClaims (PVCs) and scheduling problems.

```
kubectl get pods
NAME                                READY   STATUS      RESTARTS   AGE
my-nginx-5c45c89568-bn47r           0/1     Pending     0          12h
my-nginx-5c45c89568-ltqvs           0/1     Completed   0          12h
postgres-deployment-b95b85d69-rfl88   0/1     Pending     0          13m
```

```
kubectl describe pod postgres-deployment-b95b85d69-rfl88

Name:             postgres-deployment-b95b85d69-rfl88
Namespace:        default
Priority:         0
Service Account:  default
Node:             <none>
Labels:           app=postgres
                pod-template-hash=b95b85d69
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Controlled By:    ReplicaSet/postgres-deployment-b95b85d69
Containers:
postgres:
    Image:      postgres:16
    Port:       <none>
    Host Port:  <none>
    Environment:
    POSTGRES_DB:        mydb
    POSTGRES_USER:      admin
    POSTGRES_PASSWORD:  password
    Mounts:
    /var/lib/postgresql/data from postgres-storage (rw)
    /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-5kw7k (ro)
Conditions:
Type           Status
PodScheduled   False
Volumes:
postgres-storage:
```

(continues on next page)

(continued from previous page)

```
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the␣
↪same namespace)
    ClaimName:  database-pvc
    ReadOnly:   false
kube-api-access-5kw7k:
    Type:                     Projected (a volume that contains injected data from␣
↪multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
```

Events:

Type Reason Age From Message —- —— —- —- ——- Warning FailedScheduling 13m default-scheduler 0/1 nodes are available: pod has unbound immediate PersistentVolumeClaims. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling. Warning FailedScheduling 13m (x2 over 13m) default-scheduler 0/1 nodes are available: pod has unbound immediate PersistentVolumeClaims. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling. Warning FailedScheduling 3m39s (x2 over 8m39s) default-scheduler 0/1 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-plane: }. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling.

root@talos-client:~/cluster#

CHAPTER

FORTY

USEFULL LINKS:

cloud native landscape: https://landscape.cncf.io/

CHAPTER

FORTYONE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX