Host a fully persisted Apache NiFi service with docker

Keep your data pipelines and flows safe while changing configurations and restarting services.

Victor Seifert · Follow
Published in Geek Culture
8 min read · Nov 10, 2021

Playing around with a docker-hosted Apache NiFi and testing out its many capabilities and 288 different processors (as of version `1.14`) is only fun as long as you don't have to restart your docker containers and - worst case - loose the process groups you created. Maybe you were clever enough to create and download templates as safety copies. Maybe you even saved the process groups to NiFi's registry - but is your registry fully persistent? When it comes to data pipelines, it sometimes takes long hours to get them exactly right - meaning that the worst thing is to loose and having to painstakingly recreate them afterwards.

Instead of having to rebuild processors, reconnect the registry or reconfigure entire process groups — after every single docker restart — there is an easy way to never having to worry about it again.

The following two chapters will enable you to **set up the persisted containers and then forget about it**. The only time you will need to revisit this article is when you create a new setup alltogether.

Containers — Photo by Frank Mckenna on Unsplash.

Maybe you also came here because you wanted to get to know Apache NiFi and do it right from the get-go, in which case: don't worry. All steps are explained in detail, no previous knowledge is needed.

Creating a fully persisted NiFi service, includes persisting the Apache NiFi registry — NiFi's very own version control system for versioning process groups. We will start with the NiFi registry as it is a quick setup and then move on to NiFi itself which is a bit more tricky, but not impossible to persist even when hosted as a docker container.

We will use the following docker images as they are the latest as this article is being written — feel free to use different / newer ones, but be aware of any changes since these images:

- Zookeeper: `bitnami/zookeeper:3.7.0`
- NiFi registry: `apache/nifi-registry:1.15.0`
- NiFi: `apache/nifi:1.14.0`

While our core application is NiFi, we will include the NiFi registry to have a detailed version control of our flows available. Furthermore, Zookeeper is a fundamental part of the NiFi cluster since 1.x — enabling distributed coordination and communication within our cluster in case we want to scale up and run multiple nodes of NiFi later on.

Note: The docker images are specified by their exact version and will be downloaded autoamtically if you use the provided `docker-compose.yml` file. The entire file can be found here or at the end of this article.

Preparation

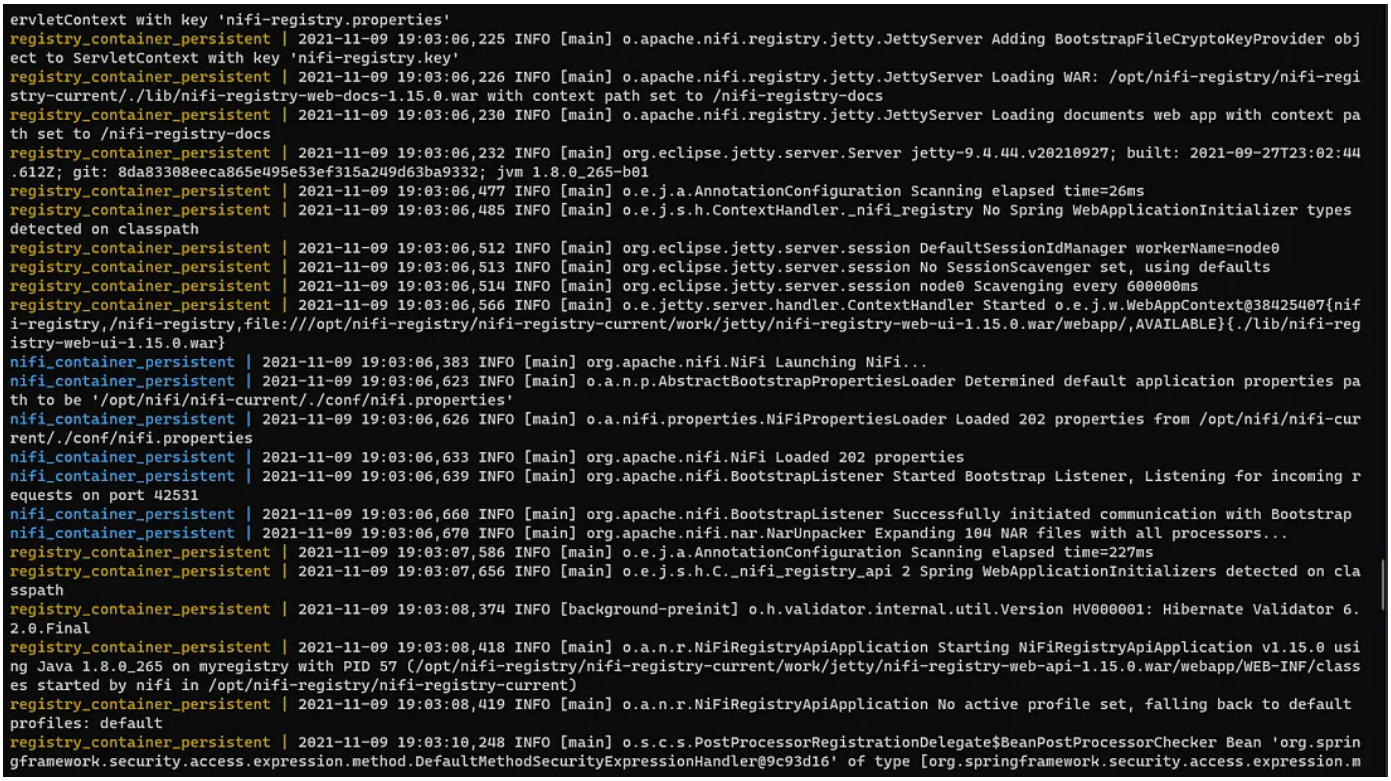
This article is aimed at windows users but is still valid if you are on another system — but your mileage may vary here and there.

1. Download and install docker from here.
2. Open your favorite command line tool and navigate to a fitting place.
3. Create an empty directory.
4. Copy the docker-compose.yml file — manually or with curl as shown below.
5. To start the docker services, run `docker-compose up`.
6. To stop the docker services, you can exit the process by pressing `CTRL+C`.

```
cd ./coding

mkdir nifi_persistence_test

cd ./nifi_persistence_test

curl https://gist.githubusercontent.com/CribberSix/260f130f2936e97ed2d0fef0df08874d/raw/a9068cb8e78

docker-compose up
```

0017.sh hosted with ❤ by GitHub view raw

Something like the following wall of text will appear and continue running until you stop the services by pressing `CTRL+C`.



Docker services running in the console — Image created by the author.

It may take about a minute until the NiFi WebUI is live.

Once the services are running, you can access the NiFi registry at http://localhost:18080/nifi-registry/ and NiFi at http://localhost:8091/nifi/ in your browser.

With the applications set up, let's get to work!

Persisting the NiFi registry

Within NiFi registry, we want to persist the buckets we create and any flows which they include.

This can be done by mounting two volumes to the local machine — `flow_storage` and `database`.

`flow_storage` includes the buckets and the flows packed within them. Buckets and flows are identified by UUIDs - universally unique identifiers - just as NiFi uses UUIDs to identify processors, processor groups and controller services among other elements. The directory structure with a bucket inside would look like this:

```
flow_storage/
├── <bucket_id>/
│   ├── <flow-id>/
│   │   ├── <version>/
│   │   │   ├── <version>.snapshot
├── 60486a45-4de8-46d3-ac9d-2505f4c1c683/
│   ├── dd07a3ea-9087-41c9-b84f-ffbb9c678fd4/
│   │   ├── 1/
│   │   │   ├── 1.snapshot
│   │   ├── 2/
│   │   │   ├── 2.snapshot
```

0020.txt hosted with ❤ by GitHub view raw

The `database` directory consists of a single file:

```
database/
├── nifi-registry-primary.mv
```

0018.txt hosted with ❤ by GitHub view raw

This is a single file which we *really* shouldn't loose. Without the database file, we won't be able to load any of our buckets and flows from the `flow_storage` directory.

If you used the provided docker-compose.yml file you will see that the two volumes of the registry service are already mounted to directories on your local machine:

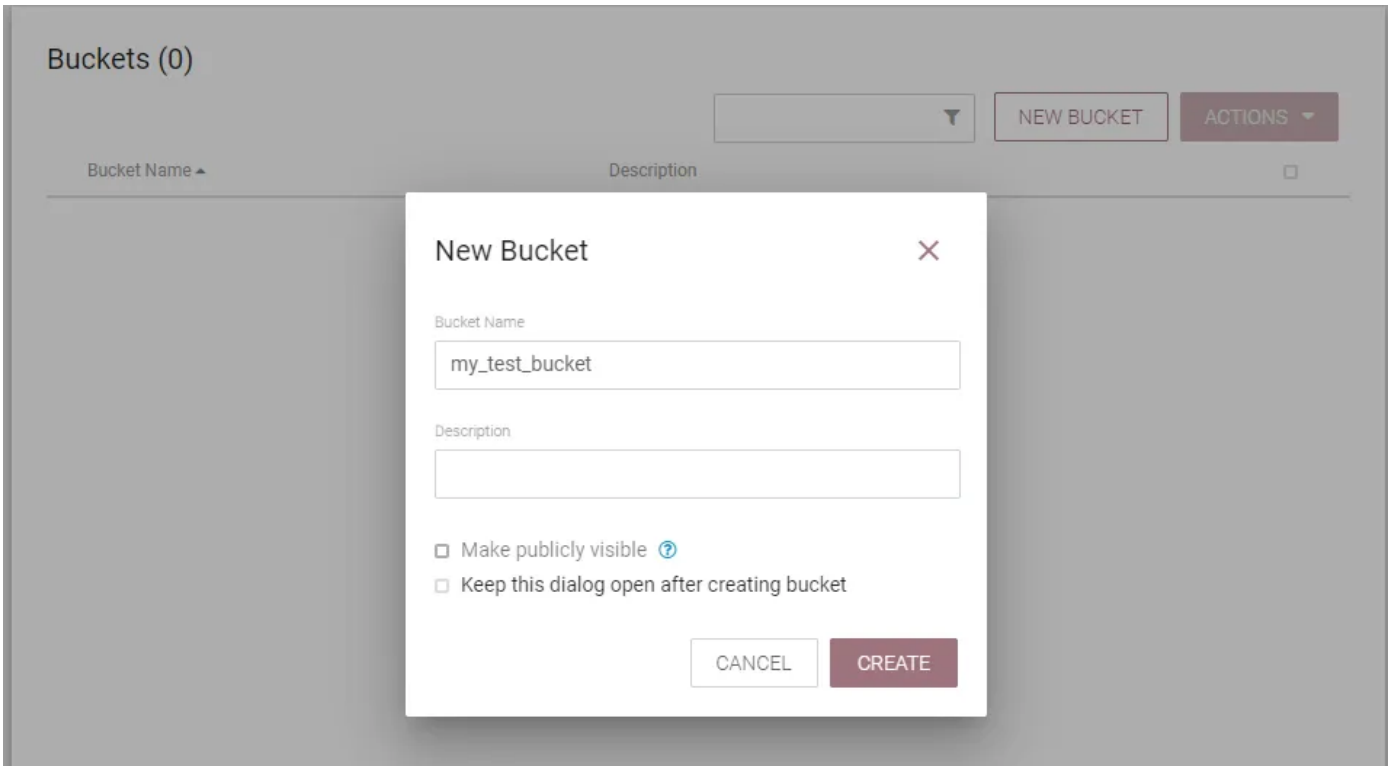
```
volumes:
- ./nifi_registry/database:/opt/nifi-registry/nifi-registry-current/database
- ./nifi_registry/flow_storage:/opt/nifi-registry/nifi-registry-current/flow_storage
```

0021.yml hosted with ❤ by GitHub view raw

This means, that any files that the docker container creates in those directories will end up being read from and written to your local machine — it won't matter whether the docker container restarts or gets completely recreated, the files are persistently stored outside of its influence.

This is it — we are already done with the registry! Any changes to flows or newly created buckets will reappear whenever we restart the docker container.

You can create a bucket in the registry by clicking on the Settings buttons (a `wrench` symbol) in the top right corner of the browser window, then on `new bucket`. Enter any name you want and click on `create`.



New bucket creation — Image created by the author.

You should now be able to see a similar filestructure to the one shown in the early part of this chapter in your local directory `/nifi_registry/flow_storage/....`

Persisting NiFi

To persist NiFi's process groups, processes, connections and controller services, we need to use a **one-time trick** after which — no matter how many times we restart docker, even with `--force-recreate` - NiFi will always come back with all of our elements intact. **This step needs your input** and is not automated from the start.

NiFi stores the information (which we need to persist the flow elements among others) in the `conf` directory - but when we mount the directory before the first startup of our docker container we will soon realize the nifi container won't start. It will instead repeatedly complain and try to fix the issue without getting anywhere:

```
sed: can't read /opt/nifi/nifi-current/conf/nifi.properties: No such file or directory
sed: can't read /opt/nifi/nifi-current/conf/nifi.properties: No such file or directory
sed: can't read /opt/nifi/nifi-current/conf/nifi.properties: No such file or directory
sed: can't read /opt/nifi/nifi-current/conf/nifi.properties: No such file or directory
replacing target file /opt/nifi/nifi-current/conf/nifi.properties
replacing target file /opt/nifi/nifi-current/conf/nifi.properties
replacing target file /opt/nifi/nifi-current/conf/nifi.properties
replacing target file /opt/nifi/nifi-current/conf/nifi.properties
```

error_message_001.txt hosted with ❤ by GitHub view raw

So here is the trick summarized for all those who cannot wait, details follow below:

1. Start NiFi's docker container without mounting the `conf` directory.
2. Copy the `conf` directory **out of the running** container to a local `conf` directory.
3. Stop the docker container.
4. Mount the local `conf` directory and restart the docker container.

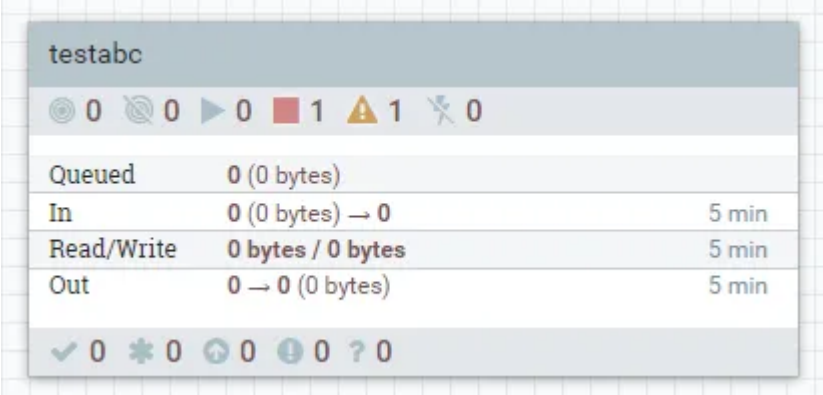
Now to the juicy details…

Step 1

Reminder: If you want to follow along, you can get the docker-compose.yml file from here or at the end of this article.

Let's start the docker containers by executing `docker-compose up` from within our newly created directory. We will have wait a bit until NiFi has completely spun up - it may take up to one minute or so until the election cycle is completed and before we can access the WebUI under http://localhost:8091/nifi/.

To test things out, let's create some sample processors, a registry client, a controller service or whatever else we might want to persist. A single processor or processgroup will suffice to see the desired result. You can drag and drop elements from the bar at the top of the browser window.



Process group for test purposes — Image created by the author.

Step 2

After starting the docker container and while it is still running, we copy the `conf` directory from the container to our local machine with the following two commands.

Execute them from **a new terminal window** (so our docker does not get interrupted), but in the same directory in which the `docker-compose.yml` file is saved, to guarantee that the directory gets copied to the correct place.

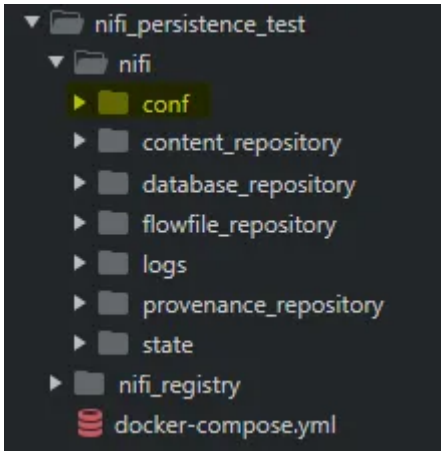
```
# enter the directory to copy the files to the right place
> cd ./nifi_persistence_test

# to get the container ID of NiFi's docker container
> docker ps

# the result will look like this (shortened to fit this article)
CONTAINER ID   IMAGE                          COMMAND
7554d9c68c8f   apache/nifi:1.14.0             ...
8af04cd37e06   apache/nifi-registry:1.15.0    ...
a2dacb43ed23   bitnami/zookeeper:3.7.0        ...

# copy the directory from the docker container to the local machine
> docker cp 7554d9c68c8f:/opt/nifi/nifi-current/conf ./nifi/
```

0019.sh hosted with ❤ by GitHub view raw

You can get the CONTAINER ID (in my case `7554d9c68c8f`) from output lines of `docker ps`.

Don't forget to replace the container id in the `docker cp` -command with your own!

Afterwards, we can check whether we have the `/nifi_persistence_test/nifi/conf/` directory including the necessary files inside of it on our local machine.



File structure — Image by the author.

Step 3

We can now stop the docker containers by pressing `CTRL+C` in the command line window in which we started them with `docker-compose up`.

Step 4

In the `docker-compose.yml` look for the *commented* line in which the volume gets mounted (`- ./nifi/conf:/opt/nifi/nifi-current/conf`) and *uncomment* it. Note: be careful to adhere to the same indentation as the lines above it.

```
    - ./nifi/provenance_repository:/opt/nifi/nifi-current/provenance_repository
    - ./nifi/state:/opt/nifi/nifi-current/state
    - ./nifi/logs:/opt/nifi/nifi-current/logs
    # uncomment the next line after copying the /conf directory from the container to your local directory
    #- ./nifi/conf:/opt/nifi/nifi-current/conf
networks:
    - persistent_network
```

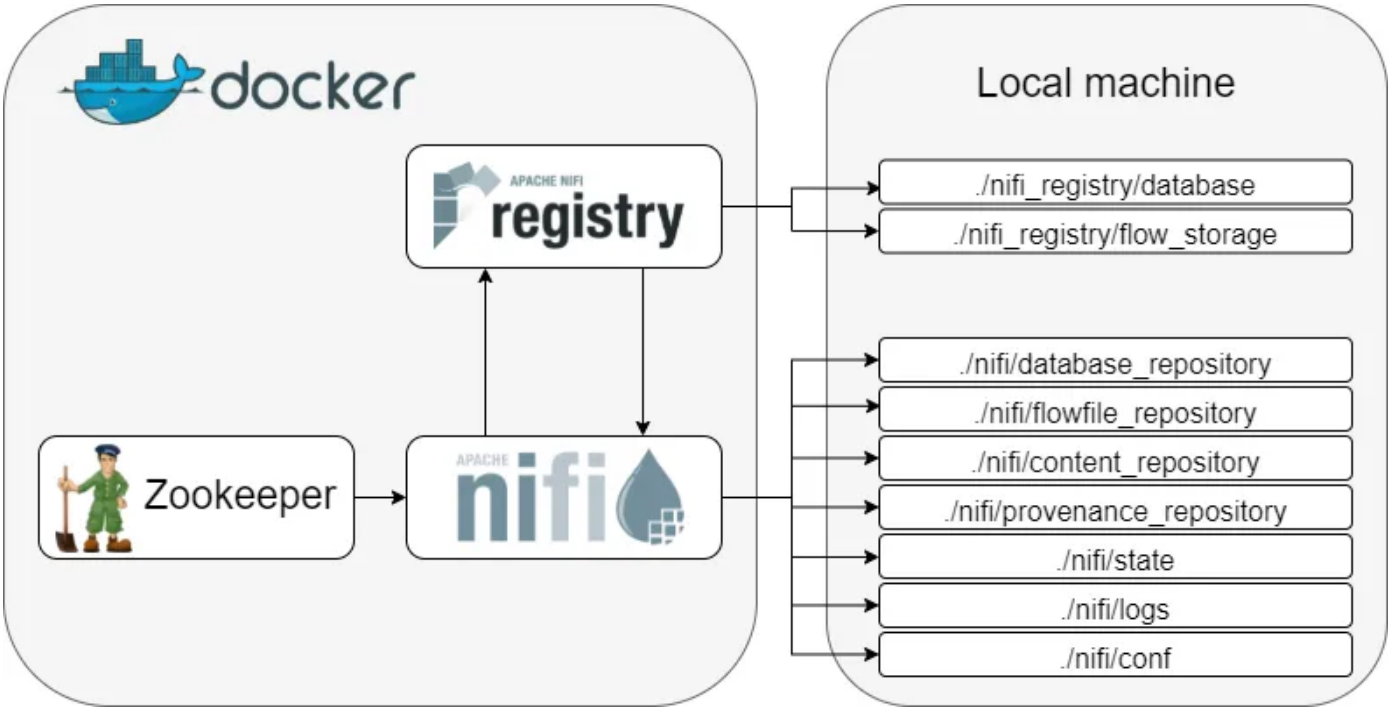
The commented line — Image by the author.

Uncommenting the line will mount the directory `conf` of the docker container to our local directory.

Now let's start our NiFi service again with `docker-compose up --force-recreate`.

We use the flag `--force-recreate` to make sure that our containers pick up our latest changes to the `docker-compose.yml` file. It does not influence the data stored in our

mounted volumes in any way. This will also start our services - with all of our elements, controller services, registry buckets and connections intact and persisted.



Mounted directories — Image created by the author. Logos from Wikipedia.

Note: if you haven't created sample processors before copying the `conf` directory to your local machine, don't worry. You can just create them now in NiFi and restart the docker containers to test whether the persistence worked.

Basically anything we create from now on will still be there after we restart the docker containers.

Closing thought

Having an Apache NiFi application running as fully persisted docker service brings many advantages, the least of which is to be able to continue our work after recreating or restarting our docker services. It is also the ideal setup to get to know NiFi, NiFi's registry and any other connecting services a bit better than before. Even if you decide to change the setup or add new services and connect them differently — you won't ever have to recreate those data pipelines!

As always, we're never done learning. Find out more about …

- Apache NiFi's purpose

- Apache NiFi registry's purpose
- Getting started with docker compose
- UUIDs and whether you can assign one to every grain of sand on earth
- How to create UUIDs in Python

Follow me on Medium for more articles about data engineering tools as well as software and personal development!

And, as promised, here is the docker-compose file:

```yaml
version: "3"
services:
    # configuration manager for NiFi
    zookeeper:
        hostname: myzookeeper
        container_name: zookeeper_container_persistent
        image: 'bitnami/zookeeper:3.7.0'  # latest image as of 2021-11-09.
        restart: on-failure
        environment:
            - ALLOW_ANONYMOUS_LOGIN=yes
        networks:
            - my_persistent_network
# version control for nifi flows
    registry:
        hostname: myregistry
        container_name: registry_container_persistent
        image: 'apache/nifi-registry:1.15.0'  # latest image as of 2021-11-09.
        restart: on-failure
        ports:
            - "18080:18080"
        environment:
            - LOG_LEVEL=INFO
            - NIFI_REGISTRY_DB_DIR=/opt/nifi-registry/nifi-registry-current/database
            - NIFI_REGISTRY_FLOW_PROVIDER=file
            - NIFI_REGISTRY_FLOW_STORAGE_DIR=/opt/nifi-registry/nifi-registry-current/flow_storage
        volumes:
            - ./nifi_registry/database:/opt/nifi-registry/nifi-registry-current/database
            - ./nifi_registry/flow_storage:/opt/nifi-registry/nifi-registry-current/flow_storage
        networks:
            - my_persistent_network
# data extraction, transformation and load service
    nifi:
        hostname: mynifi
        container_name: nifi_container_persistent
        image: 'apache/nifi:1.14.0'  # latest image as of 2021-11-09.
        restart: on-failure
        ports:
            - '8091:8080'
        environment:
            - NIFI_WEB_HTTP_PORT=8080
            - NIFI_CLUSTER_IS_NODE=true
            - NIFI_CLUSTER_NODE_PROTOCOL_PORT=8082
            - NIFI_ZK_CONNECT_STRING=myzookeeper:2181
            - NIFI_ELECTION_MAX_WAIT=30 sec
            - NIFI_SENSITIVE_PROPS_KEY='12345678901234567890A'
```

```
            - NIFI_SENSITIVE_PROPS_KEY= 12345678901234567890A
        healthcheck:
            test: "${DOCKER_HEALTHCHECK_TEST:-curl localhost:8091/nifi/}"
            interval: "60s"
            timeout: "3s"
            start_period: "5s"
            retries: 5
        volumes:
            - ./nifi/database_repository:/opt/nifi/nifi-current/database_repository
            - ./nifi/flowfile_repository:/opt/nifi/nifi-current/flowfile_repository
            - ./nifi/content_repository:/opt/nifi/nifi-current/content_repository
            - ./nifi/provenance_repository:/opt/nifi/nifi-current/provenance_repository
            - ./nifi/state:/opt/nifi/nifi-current/state
            - ./nifi/logs:/opt/nifi/nifi-current/logs
            # uncomment the next line after copying the /conf directory from the container to your
            #- ./nifi/conf:/opt/nifi/nifi-current/conf
        networks:
            - my_persistent_network
networks:
  my_persistent_network:
    driver: bridge
```

docker-compose-persistent-nifi.yml hosted with ❤ by GitHub view raw

Data Engineering | Apache Nifi | Docker | Docker Compose | Data Engineer

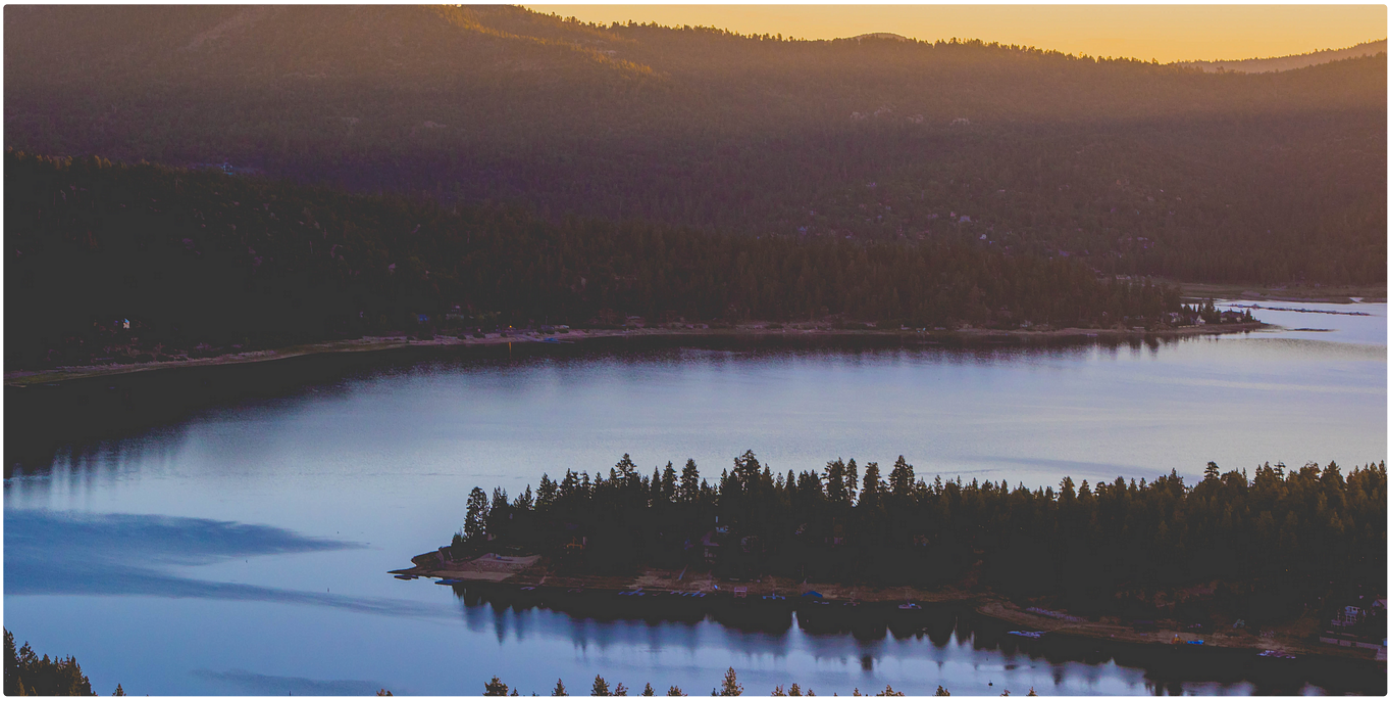
Follow

Written by Victor Seifert

144 Followers · Writer for Geek Culture

Data Engineer. Consultant. GameDev. I write about data engineering tools, software and personal development.

More from Victor Seifert and Geek Culture



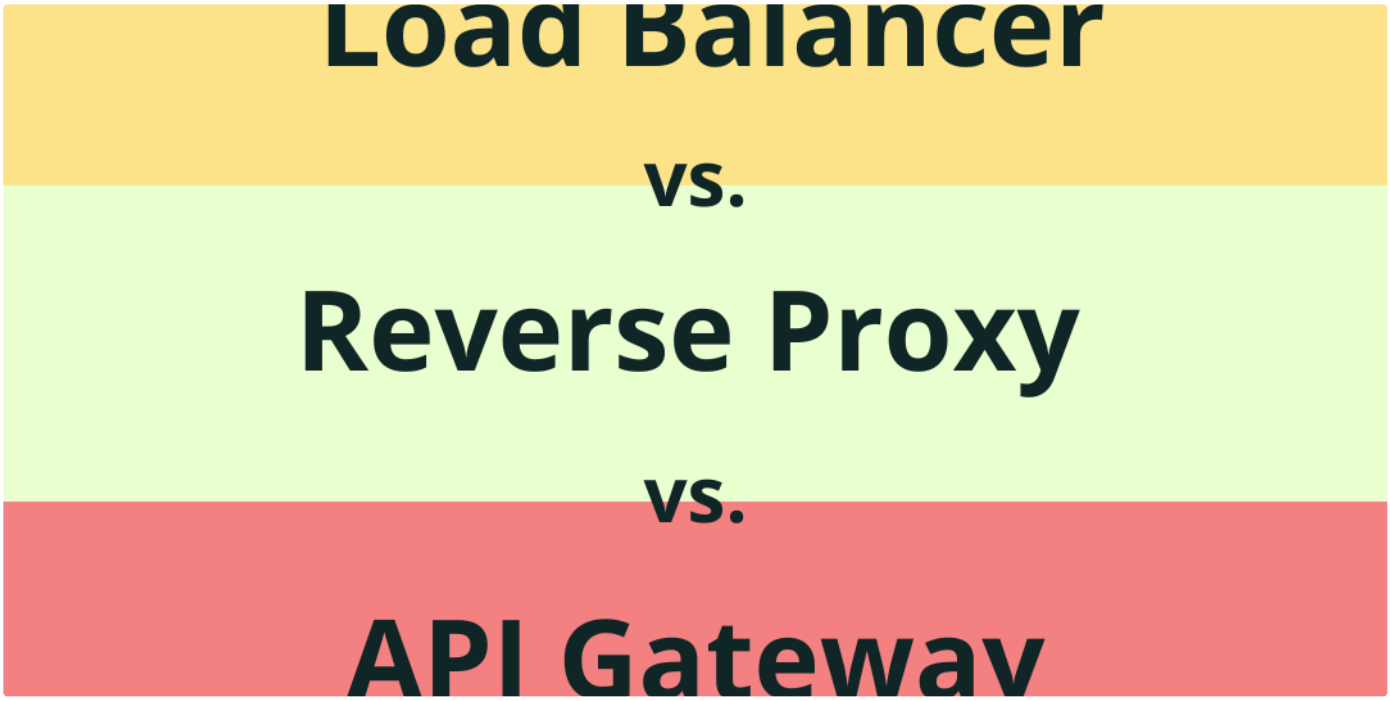
Victor Seifert in Towards Data Science

How to build a data lake from scratch—Part 1: The setup

The complete tutorial of how to make use of popular technology to build a data engineering sandbox

14 min read · Nov 17, 2021

122 3



Arslan Ahmad in Geek Culture

Load Balancer vs. Reverse Proxy vs. API Gateway

Understanding the Key Components for Efficient, Secure, and Scalable Web Applications.

· 12 min read · May 17

parse_expenses.py

```python
import datetime

def parse_expenses(expenses_string):
    """Parse the list of expenses and return the list of triples (date, value, currency).
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2016-01-02 -34.01 USD
        2016-01-03 2.59 DKK
        2016-01-03 -2.72 EUR
    """
    expenses = []
    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
                         float(value),
                         currency))
    return expenses
```

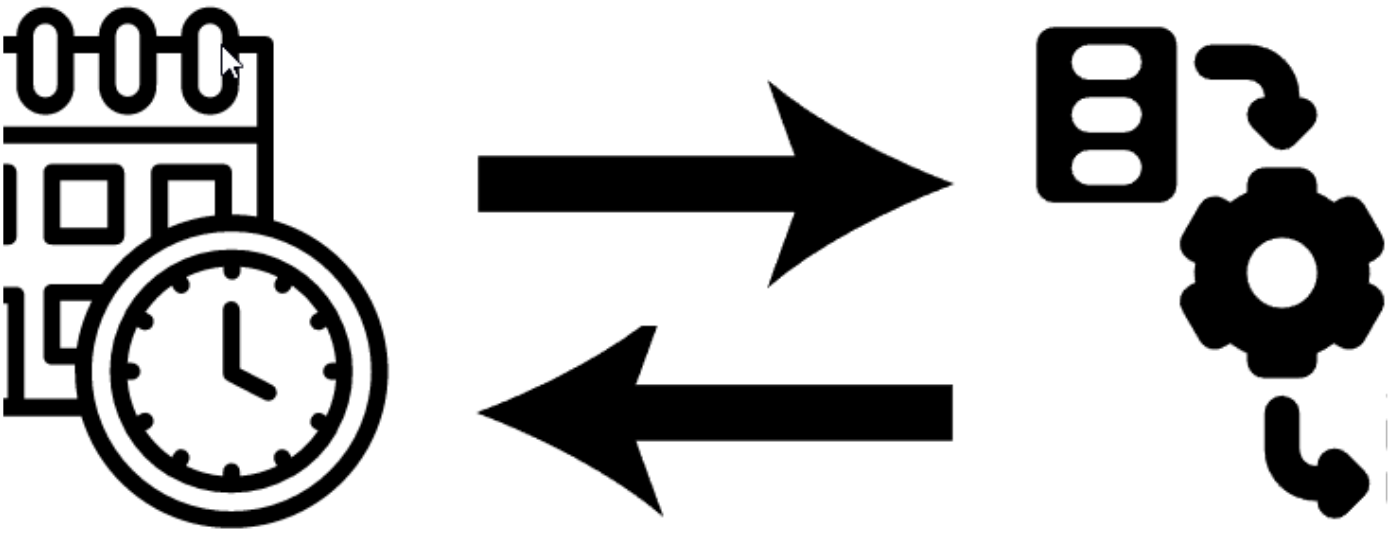
Jacob Bennett in Geek Culture

The 5 paid subscriptions I actually use in 2023 as a software engineer

Tools I use that are cheaper than Netflix

· 4 min read · Mar 25

3.6K 37



Victor Seifert in Towards Data Science

How to connect Airflow with a NiFi ETL-pipeline

NiFi and Airflow are both powerful tools which are regularly in use in data projects, but using them together is not always easy.

7 min read · Oct 25, 2021

69 3

See all from Victor Seifert

See all from Geek Culture

Recommended from Medium

DataGeeks

Apache Airflow, A must-know orchestration tool for Data engineers.

Apache Airflow is an orchestration tool developed by Airbnb and later given to the open-source community. Today, it is the most beloved...

· 9 min read · Feb 8

179 2

Thomas Reid in Level Up Coding

Streaming change data capture (CDC) data between databases using Kafka

This article will briefly introduce Kafka, how to connect database sources to it using the Kafka SQL client ksqlDB and create and...

21 min read · Mar 26

86

Lists

New_Reading_List

174 stories · 37 saves

Coding & Development

11 stories · 74 saves

Natural Language Processing

442 stories · 81 saves

Now in AI: Handpicked by Better Programming

260 stories · 55 saves

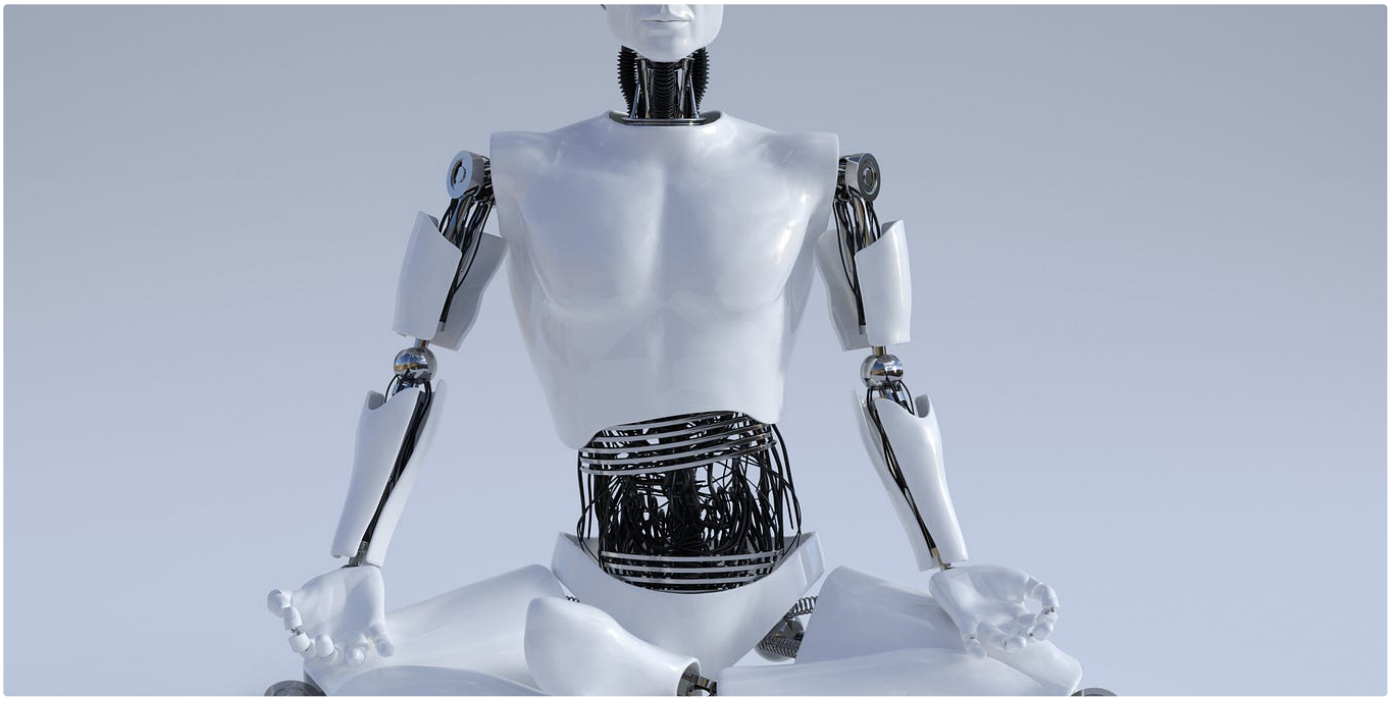
Tamir Suliman in Coinmonks

Deploy Elasticsearch & Kibana Environment with Docker

ELK stack which includes Elasticsearch, Kibana, and Logstash considered one of the powerful tools for logging, searching and analyzing...

· 7 min read · Mar 5

35 1



The PyCoach in Artificial Corner

You're Using ChatGPT Wrong! Here's How to Be Ahead of 99% of ChatGPT Users

Master ChatGPT by learning prompt engineering.

· 7 min read · Mar 17

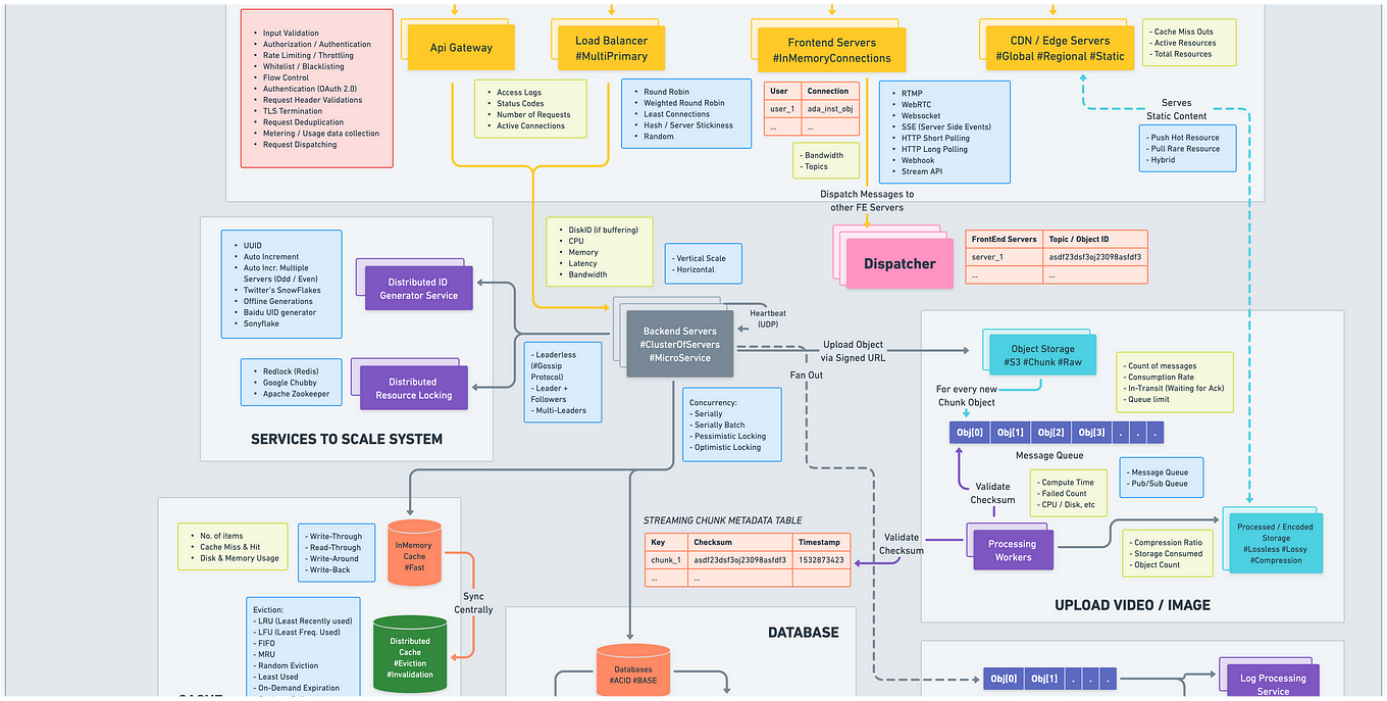
29K 525

Feng Li

Kafka Broker Setup Using Docker image

1 Setup Kafka cluster on Windows laptop

· 4 min read · Feb 28



Love Sharma in ByteByteGo System Design Alliance

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

· 9 min read · Apr 20

6.7K 52

See more recommendations