DevOps 2021

Malou :)

3 May 2021

Contents

System Design

Minitwit is a social media application that provides basic Twitter-like services. It consists of a web app and API services that are publicly available on the internet. Both services allow the user to register a profile, log in, create messages (tweets), follow and unfollow users. Basic authentication is required when creating messages, following, or unfollowing. Most of the application is written in Python, since our web app and API are using the Django framework. The web app and API are hosted on separate servers on a single Digital Ocean droplet, and communicate with a server on a separate droplet.

Minitwit also consists of logging and monitoring tools that depends on the web app and API services, which are hosted on multiple servers on the same droplet as the web app and API. The monitoring tools communicate with a logging server which is hosted on a third separate droplet.

Architecture

describe what goes on the production CI droplet and why We have tried to keep most of our project on one droplet, so that the process of deployment is always the same, no matter what part of the system is being changed. This main CI droplet runs Docker in swarm mode, so that the web and API services can be scaled. Other services that either cannot or do not need scaling are hosted on either the manager node or on a separate droplet.

Subsystems

With a project that requires a web app and API, it is normal to have the web app communicate with the database through the API. However, Django designed such that communication with a database backend is much simpler to implement than communication with a custom backend server. For this reason, our web app and API don't communicate with one another, and therefore don't form a frontend/backend structure. Instead, our database is our backend and our web app/API servers are our frontends.

describe how proxy works w web + API We have a nginx proxy that routes traffic from minitwititu.xyz to our web app server's IP address, and from api.minitwititu.xyz to our API server's address. The proxy is hosted on the swarm manager node because it does not need scaling, because … *Does the proxy only route HTTP requests? does it route anything else? why is it on the manager node (will it not be scaled)?*

describe how elastic search and filebeat are structured and why some fuckin sequence diagram or something why is elastic search on a separate droplet? why is it on the manager node? does filebeat communicate w any other services besides elastic search?

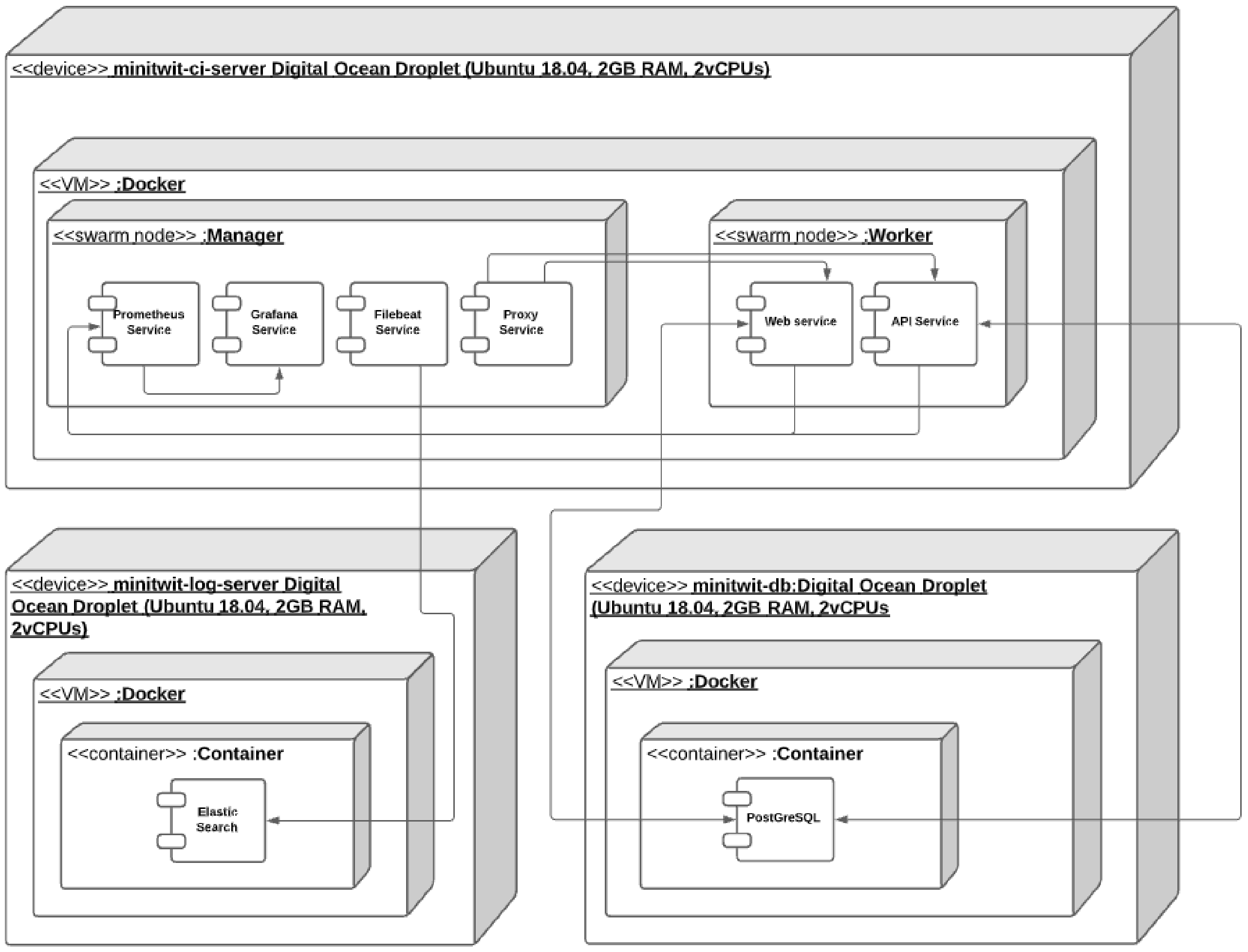

Figure 1: Deployment Diagram

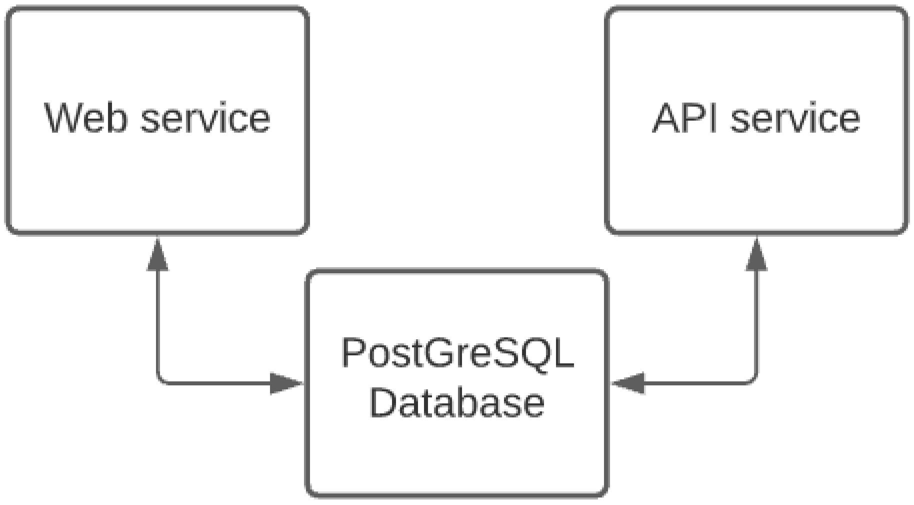

Figure 2: Web, Api, DB structure

describe how web + API communicate w prometheus, and how prometheus communicates w grafana Our monitoring is accomplished by Prometheus, which exposes our metrics on minitwitwitu.xyz/metrics. Our web app and API both make calls to update certain Prometheus metrics, and Prometheus gathers other performance-related metrics from both of them. The /metrics route is also checked by our Grafana server, which allows us to create monitoring dashboards for all the metrics on that route.

Dependencies

Our dependencies are split into direct dependencies and tools

Tools

- Docker | Cloud computing services
- Digital Ocean | Cloud infrastructure provider
- Travis | Hosted continuous integration service
- ElasticSearch | Distributed RESTful search and analytics engine
- Kibana | Data visualization dashboard software for ElasticSearch
- Filebeat | File harvester
- PostGreSQL | Database Manangement System
- NGINX | Proxy (and load balancer????)

...??

Web App dependencies are as follows:

- asgiref 3.3.1 - Includes pytest a framework that makes it easy to write small tests
- django 3.1.8 - Python Web Framework
- django-prometheus 2.1.0 - Export django monitoring metrics for Prometheus
- django-rest-framework 3.12.2 - Web APIs for Django
- prometheus 0.9.0 - Prometheus instrumentation library
- psutil 5.8.0 - Python system monitoring
- psycopg2 2.8.6 - PostgreSQL database adapter for Python
- pycodestyle 2.7.0 - Python style checker
- pytz 2021.1 - Cross platform timezone calculations
- requests 2.25.1 - HTTP library
- sqlparse 0.4.1 - SQL query parser / transformer
- uWSGI 2.0.18 - 2.1 - Web service gateway

API dependecies are as follows:

- asgiref 3.3.1 - Includes pytest a framework that makes it easy to write small tests
- django 3.1.8 | Python Web Framework
- django-prometheus 2.1.0 - Export django monitoring metrics for Prometheus
- markupsafe 1.1.1 - Safely add untrusted strings to HTML/XML markup
- psutil 5.8.0 - Python system monitoring
- psycopg2 2.8.6 - PostgreSQL database adapter for Python
- pytz 2021.1 - Cross platform timezone calculations
- requests 2.25.1 - HTTP library
- sqlparse 0.4.1 - SQL query parser / transformer
- toml 0.10.2 - Python library for TOML
- uWSGI 2.0.18 - 2.1 - Web service gateway
- wrapt 1.12.1 - A Python module for decorators, wrappers and monkey patching.

Current state

Get the fucking grafana to tell us something xD

License

We collected all the license for every dependency we have to form the license our product. Here we met the GNU GPL v2 for psycopg2, The GPL series are all copyleft licenses, which means that any derivative work must be distributed under the same or equivalent license terms. To cover the product we therefore chose to go with the GNU General Public License v3.0; can be found in the licence document. In this process we also collected all copyright noticies for the dependencies, these are all placed in the Notice document.

The Team

The team is organized via Discord. We build a server to support all communication through that. Here we meet up, discuss meetings, solutions, problems and more. We also created a Github webhook, so that everyone gets a notification on the Discord server, whenever there are made changes to the project.

We use discord servers to meet up every Monday during and after our lecture, we usually work on the given task most of that day. The work that we did not finish are usually completed in the weekend, because of very different time schedules during the week.

CI/CD chain

Git hook

Pre commit

- pep8
- something black

Travis

- Make a new release

Stages

- docker_build
- test
- deploy

Repository

what, where, why We have organized the project in one repository ... We split logging into a separate repository because ...

Branching strategy

Our branching strategy utilizes the Gitflow Workflow. Basically we will have 2 main branches and a number of feature branches:

- main branch - This is the main branch and contains the production code.
- developer branch - This is the development branch containing the development code. This code is merged and pushed into main at the end of each completed weekly assignment.
- feature branches - Used to develop specific features relating to a specific assignment and is merged and pushed into the development branch when it is completed.

Branches should always be branched from develop

Always pull the newest development branch before creating a feature branch

Experimenting is preferably done in branches. In some cases when working on a specific task it might make sense to further branch out from the feature branch eg.

Development process

We used Github Projects to organize tasks in Kanban boards. We created 2 boards to separate tasks on the frontend and backend at the beginning of the project.

…

Monitoring

We use the monitoring service Prometheus, to monitor our application… To display the data that we get from Prometheus, we use the web-based multi-source graph interface, Grafana.

We split our monitoring into 2 Grafana dashboards; Business Monitoring (e.g. images/Business Monitoring), which displays our PostgreSQL Query data, and Infrastructure Monitoring (e.g. images/Infrastructure Monitoring), which displays our Prometheus metrics.

The Business Monitoring dashboard contains amount of: Users, Messages and Followers. We used these data to monitor the correctness of successful requests.

The Infrastructure Monitoring dashboard contains CPU Load percent, used for up-time calculation as well as strain on the system and HTTP Responses (Frontend / Backend), used to monitor system failure as well as performance in regards to correct response.

Logging

…

Security

…

Scaling and load balancing

… # Leasons learned