Tianji

- Install and Setup Tianji

Install and Setup Tianji

https://www.youtube.com/embed/54F7mZcrg7M

Install Updates

Update and upgrade your server's packages using the following commands:

- For Ubuntu/Debian: `sudo apt update && sudo apt upgrade -y`
- For RedHat/CentOS/Fedora/Alma/Rocky: `sudo dnf update -y`

Create a Non-Root User

Create a non-root user with superuser (sudo) privileges:

1. Add a new user using `adduser <username>`
2. Set the password for this user.
3. Enter the relevant information (optional)
4. Enter 'Y', then press Enter.
5. Add the user to the "sudo" group:
   - For Ubuntu/Debian: `usermod -aG sudo <username>`
   - For RedHat/CentOS/Fedora/Alma/Rocky: `usermod -aG wheel <username>`

Ubuntu / Debian

`usermod -aG sudo <username>`

RedHat / CentOS / Fedora / Alma / Rocky

`usermod -aG wheel <username>`

Now, you can log out of the system, and log back in as your new non-root super user.

Step 2: Install Docker and Docker Compose

Install Docker and Docker Compose on your server:

1. Install the `curl` utility:
   - For Ubuntu/Debian: `sudo apt install curl -y`
   - For RedHat/CentOS/Fedora/Alma/Rocky: `sudo dnf install curl -y`

2. Run the command to install Docker and Docker Compose: `curl https://get.docker.com | sh`

Add Your User to the Docker Group

Add your non-root user to the docker group so you can use Docker commands without sudo:

`sudo usermod -aG docker <username>`

Install Tianji in Docker

First we'll create a new folder to hold our applications and services files.

`mkdir -p docker/tianji`

Now we'll move into that folder.

`cd docker/tianji`

Here we will download the pre-made compose file from the Tianji github repo. Use this command to do that.

`wget https://raw.githubusercontent.com/msgbyte/tianji/master/docker-compose.yml`

Now we should have a new file inside our folder called 'docker-compose.yml'. I personally like to name my files 'compose.yaml' these days, but it's not a requirement.

`mv docker-compose.yml compose.yaml`

will rename the file if you want to do that.

Next, let's open our 'compose.yaml' file and make any necessary modifications to it.

```
---
services:
  tianji:
    image: moonrailgun/tianji
    ports:
      - "12345:12345"
    environment:
      DATABASE_URL: postgresql://tianji:< us3A10n6StRonGPAsSworDHer3 >@postgres:5432/tianji
      JWT_SECRET: < openssl rand -base64 32 >
      ALLOW_REGISTER: "false"
      ALLOW_OPENAPI: "true"
```

```
      PUBLIC_URL: "< https://app.tianji.dev >" # for example: https://app.tianji.dev
    depends_on:
      - postgres
    restart: unless-stopped
  postgres:
    image: postgres:15.4-alpine
    environment:
      POSTGRES_DB: tianji
      POSTGRES_USER: tianji
      POSTGRES_PASSWORD: < us3A10n6StRonGPAsSworDHer3 > # must match the one from the BASE_URL
abovbe
    volumes:
      - ./tianji-db-data:/var/lib/postgresql/data
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]
      interval: 5s
      timeout: 5s
      retries: 5
```

In the above file, I've surrounded any variables with less than '<' and greater than '>' signs. You should remove these symbols, and replace the value between them with values that are important to you.

JWT_SECRET: use `openssl rnad -base64 32` to generate a good strong value from the command line, then come back in and replace the placeholder.

The Password in the PostgreSQL URL: Just use a random set of upper, lower case letters and numbers, around 20 or more characters long.

PUBLIC_URL: This is the URL you want applications to reach your server at (mostly if you're going to be collecting stats from sites you run. Something like `https://mystats.myawesomedomain.com`

POSTGRES_PASSWORD: this must match the password you replaced in the POSTGRES_URL above.

Once you've replaced all of these values, save with CTRL + O, then press Enter to confirm, and exit with CTRL + X.

Now we'll pull down our docker images with

`docker compose pull`

This pulls down the images you need, and also will confrim that your yaml syntax is valid.

Next, we need to startup our containers from the images we pulled down, and we'll watch the logs briefly the first time so we can make sure everything starts up properly.

```
docker compose up -d && docker compose logs -f
```

The above is two command in one line. The `&&` connects the two commands. The first part tells docker compose to bring up the containers, and run them in the background in `detached` mode. The second part tells docker compose to show you the logs as they are generated (or `follow`).

As long as you don't see any errors spitting out of the logs repeatedly, your application should be up and ready in just a few seconds.

Now, go to your favorite modern web browser, and enter the IP address of your server, and add port '12345' to it, like this:

`http://192.168.50.183:12345` using your server's IP of course.

You should be greeted with the Tianji login screen. Use the default, initial, account of

Username: `admin`
Password: `admin`

Login, and immediately locate the user management in settings, and change the default password to something unique and strong.

Log out, and back in using your new credentials.

Setting up our Reverse Proxy

Depending on your reverse proxy of choice, you may have to do some different steps to set things up. I'm using a reverse proxy called Pangolin. It provides a secure tunnel to my services inside my home network without me having to open any ports on my home's outward facing firewall.

In any reverse proxy, you'll the need the following details:

- The domain / subdomain you want to call your site, for instance: `mystats.mydomain.com`
- The IP address of the server / host where you are running the docker container for the application, for instance: `192.168.1.230`
- The port number your new service / application is running on, for instance: `12345`

When using a reverse proxy, you need to own the domain / subdomain you wish to use for this service, and have the ability to set it's DNS entries to point to your reverse proxy host IP address.

The way this works is:

request to your domain / subdomain --> DNS entry for the domain / subdomain --> Reverse Proxy server --> Your server host private IP --> your service container via the port mapping.

I have videos on using Pangolin, along with NginX Proxy Manager, though it has a newer version now, so maybe a new video on it in the future.