[Overview]

Flagtick Group to Docker Integration

|  |  |
| --- | --- |
| Version: | 1.0 |
| Date: | Jan. 01, 2025 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Prepared By:** | **Reviewed By:** | **Reviewed By:** | **Approved By:** |
| Vuong Nguyen,  Senior Program Manager |  |  |  |
| **Date:** | **Date:** | **Date:** | **Date:** |
| **Signature:** | **Signature:** | **Signature:** | **Signature:** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ver. No** | **Date of Release** | **Prepared By** | **Reviewed By** | **List of Changes from Previous Version** |
| 0.1 | 1-JAN-25 | Vuong Nguyen |  | None (Initial Draft) |

Contents

[**1.** **Project Information** 5](#_Toc190616093)

[**2.** **GitHub – FlagtickGroup.suite.vc** 6](#_Toc190616094)

[Introduction: 6](#_Toc190616095)

[The flagtickgroup.suite.vc project is stored as a private repository on GitHub. This repository integrates multiple submodules to manage all project modules effectively. The root directory of flagtickgroup.suite.vc contains the docker-compose.yml file, which orchestrates the project's Docker setup. To access the repository securely, SSH public key authentication is employed. GitHub Actions securely manages both the private and public keys, ensuring seamless integration and secure interactions from the local machine. 6](#_Toc190616096)

[**3.** **Configuring SSH for GitHub** 6](#_Toc190616097)

[**4.** **Orchestrating Multi-Containers** 11](#_Toc190616098)

[**5.** **Orchestrating PHP-FPM & Debugging** 19](#_Toc190616099)

[**6.** **SonarQube for Code Quality** 29](#_Toc190616100)

[**7.** **Artillery for Load Testing** 31](#_Toc190616101)

[**8.** **Nexus Repository Manager** 34](#_Toc190616102)

[**9.** **S3 Bucket Migration Process** 47](#_Toc190616103)

[**10.** **Managing AWS with Terraform** 52](#_Toc190616104)

[**11.** **Remote Server Service Credentials** 59](#_Toc190616105)

[Proposed Solution: 59](#_Toc190616106)

# **Project Information**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **CLIENT NAME** | **Flagtick Group Inc (Configurable)** | | | | | | |
| **PROJECT NAME** | Docker Integration | | | | | | |
| **REQUESTOR NAME** | Vuong Nguyen | | | | | | |
| **APPROVED BY** |  | | | | | | |
| **PROJECT SPONSOR** |  | | | | | | |
| **PROJECT**  **DESCRIPTION** | Given the current technical constraints of Vuong Nguyen, Flagtick Group has undertaken the initiative to create comprehensive documentation to support the setup of environments, tools, and resources. This will facilitate system scalability and integration with specific business domains, such as digital experience platforms and other enterprise applications.  The primary purpose of this document is to provide a structured guide for implementing Flagtick Group’s infrastructure, from local machine configurations to deployment on remote servers, using Docker. This documentation will serve as a foundation for expanding system capabilities and ensuring seamless integration with future business needs.  **Scope of the Documentation:**   1. **Local Environment Setup with Docker**    1. Configure a local environment to run essential services including:       1. **Nginx**: Web server setup and configuration.       2. **MySQL**: Database service setup and management.       3. **PhpMyAdmin**: Database management interface configuration.       4. **PHP-FPM**: Fast CGI Process Manager for PHP applications. 2. **SSL Setup and Endpoint Configuration**    1. Set up SSL certificates to secure local environments.    2. Enable and configure specific endpoints to simulate production-ready workflows on the local side. | | | | | | |
| **PROJECT TYPE** |  | | Free | | | | |
|  | | Simple | | | | |
|  | | Medium | | | | |
| X | | Complex | | | | |
| **PROJECT DRIVER** | 3 | Scope | | 1 | Time | 2 | Cost |
| *(Rank: 1 – highest, 2 – median, 3 – lowest)* | | | | | | |
| **TARGET DATE FOR RELEASE** | TBD | | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **REFERENCES** | | | |
| **Ref.** | **Title** | **Version** | **Source** |
| 1 | Flagtick Group Endpoint | 1.0 | [KMD] FLAGTICKGROUP API DOCUMENT v2.0.docx.pdf |

# **GitHub – FlagtickGroup.suite.vc**

# Introduction: The flagtickgroup.suite.vc project is stored as a private repository on GitHub. This repository integrates multiple submodules to manage all project modules effectively. The root directory of flagtickgroup.suite.vc contains the docker-compose.yml file, which orchestrates the project's Docker setup. To access the repository securely, SSH public key authentication is employed. GitHub Actions securely manages both the private and public keys, ensuring seamless integration and secure interactions from the local machine.

Configuring SSH for GitHub: Using a Custom Private Key for Secure Push/Pull Access

# **Configuring SSH for GitHub**

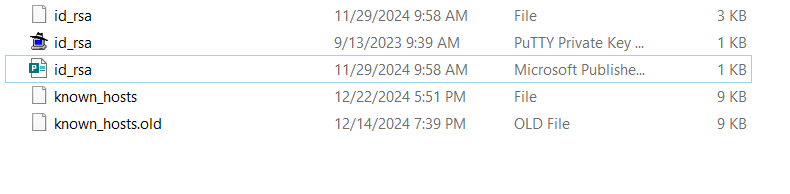
Here, when you push code to GitHub, it means your local machine is used to authenticate with the GitHub server, ensuring that the GitHub server remains secure. In this process, the private key (.pem) is stored on your local machine, while the public key is uploaded to the remote GitHub server. The connection between the local and remote machines allows operations to start from the local machine (outbound) to the remote server (inbound).

Here are the steps, step by step:

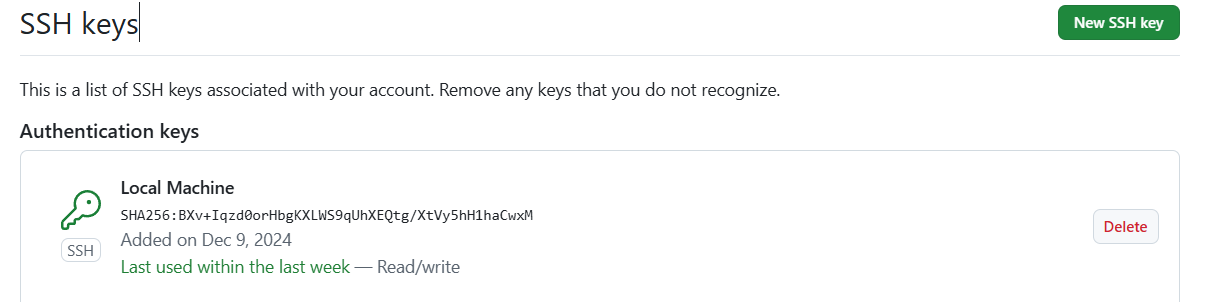
1. Generate an SSH key pair for secure authentication.

ssh-keygen -t rsa -b 2048 -m PEM -f ~/.ssh/rsa

1. Using **rsa.pem** as Private Key and rsa.pub for Remote Server Authentication



1. Adding Public Key to GitHub for SSH Protocol Authentication



**Note:** Since GitHub, GitLab, or a VPS remote server uses SSH for authentication, the public key is uploaded to the remote server (e.g., GitHub, GitLab). When a request is made, the server sends a challenge that can only be answered by the holder of the corresponding private key. The public key, stored on the server, is used to verify the private key stored on the local machine. If the local machine successfully responds to the challenge using the private key, access is granted, allowing the user to interact with the remote server or push/retrieve data.

1. Configuring Git to Use a Specific SSH Key for Authentication

git config --global core.sshCommand "ssh -i C:/Users/admin/Documents/flagtickgroup.suite.vc/tools/ssh/rsa.pem"

1. For a remote VPS (remote server), if they do not provide you with the private key to store on your local machine for authentication, there are two options to handle the request authentication.

* Use the public key from the local machine, upload it to the remote server, and rename rsa.pub to authorized\_keys.pub on the remote server.
* Use ssh-keygen -t rsa -b 2048 -m PEM to generate both the public and private keys on the remote server. Rename the public key to authorized\_keys.pub, then download the private key to the local machine for SSH access to the remote server. Afterward, delete the private key from the remote server.

1. Access SSH Directly Using Public IP

ssh -i "~\flagtickgroup.suite.vc\tools\ssh\rsa.pem" ec2-user@18.142.229.15

1. Understanding git remote -v: Managing and Verifying Remote Repositories in Git

git remote -v  
origin git@github.com:FlagtickGroupInc/flagtickgroup.suite.vc.git (fetch)  
origin git@github.com:FlagtickGroupInc/flagtickgroup.suite.vc.git (push)

**Note:** If a file has been committed or added to the staging area, use git rm --cached <file/folder> to remove it from the staging area, and then add it to .gitignore file to ensure it is ignored in future commits.

1. Set up submodule for Flagtickgroup.suite.vc and link it to the same GitHub account.

First and foremost, set up a private repository in the same GitHub account, such as vc.flagtickgroup, and then add flagtickgroup.suite.vc to it.

git submodule add [git@github.com:FlagtickGroupInc/vc.flagtickgroup.git](mailto:git@github.com:FlagtickGroupInc/vc.flagtickgroup.git) vc.flagtickgroup

If you encounter an issue, remove the vc.flagtickgroup entry from .git/modules/vc.flagtickgroup or use the command below.

rm -rf .git/modules/vc.flagtickgroup

After adding the submodule, you need to initialize it, which sets up the submodule’s configuration and fetches its content, similar to running terraform init.

git submodule init

After initializing the submodule, you need to fetch its contents and set it up in your project directory. This will clone the submodule repository into the specified directory (in this case, vc.flagtickgroup).

git submodule update

You can initialize and update the submodule in a single step by running the following command:

git submodule update --init –recursive

1. Observe and check the .gitignore file in the root directory of flagtickgroup.suite.vc.

# Use `git rm --cached <file/folder>` to remove the file or folder from the staging area and apply the skip in the .gitignore file.  
# Ignore JetBrains IDE files  
.idea/  
  
# Ignore everything in the db folder under resources.flagtickgroup  
!resources.flagtickgroup/db/  
resources.flagtickgroup/db/import/  
  
resources/terraform/.terraform  
resources/terraform/.terraform.lock.hcl  
resources/terraform/.terraform.tfstate.lock.info  
resources/terraform/terraform.tfstate  
resources/terraform/terraform.tfstate.backup  
  
!tools/  
tools/libraries/vendor.zip

1. Observe and check the .gitmodules file in the root directory of flagtickgroup.suite.vc.

[submodule "vc.flagtickgroup"]  
 path = vc.flagtickgroup  
 url = git@github.com:FlagtickGroupInc/vc.flagtickgroup.git  
 branch = master

1. Remove Files from Git Index Without Deleting Them Locally

In practice, if you accidentally add file like docker-compose.yml to the Git remote repository but want to remove it and include it in .gitignore, you can use the following script:

git rm --cached docker-compose.yml   
git add .gitignore  
git commit -m "Ignore file docker-compose.yml file"  
git push origin master

**Noted:** You also need to add docker-compose.yml as a line in the .gitignore file to ensure the file is ignored in future commits and pushes.

# Ignore JetBrains IDE files  
.idea/  
docker-compose.yml

1. Process of pushing code from the submodule to the root module.

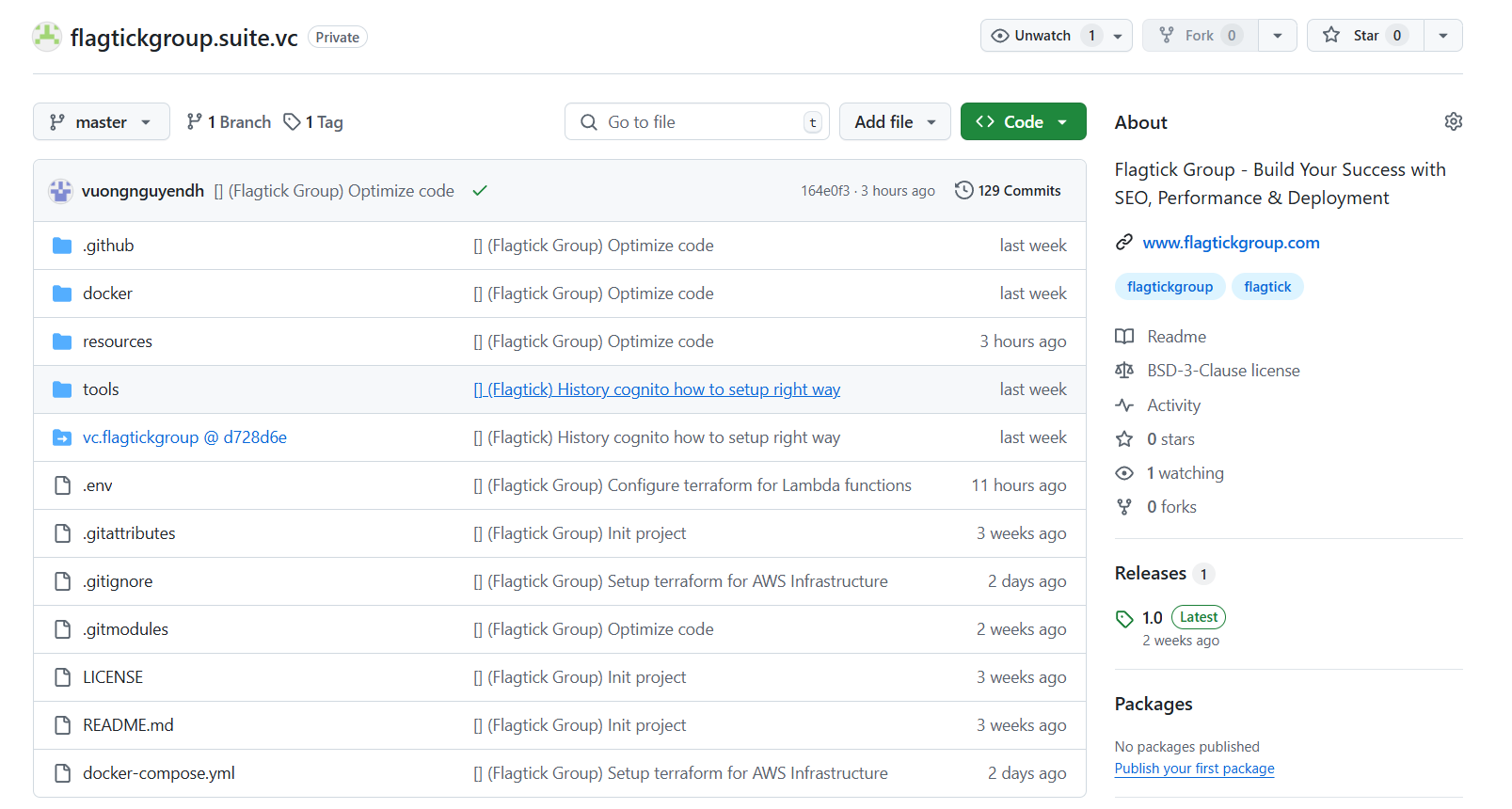
Commit changes in the submodule (vc.flagtickgroup), then exit the root module (flagtickgroup.suite.vc) and commit to update the parent with the latest commit for vc.flagtickgroup.

cd vc.flagtickgroup  
git add .  
git commit -m "Your commit message for vc.flagtickgroup"  
git push origin master

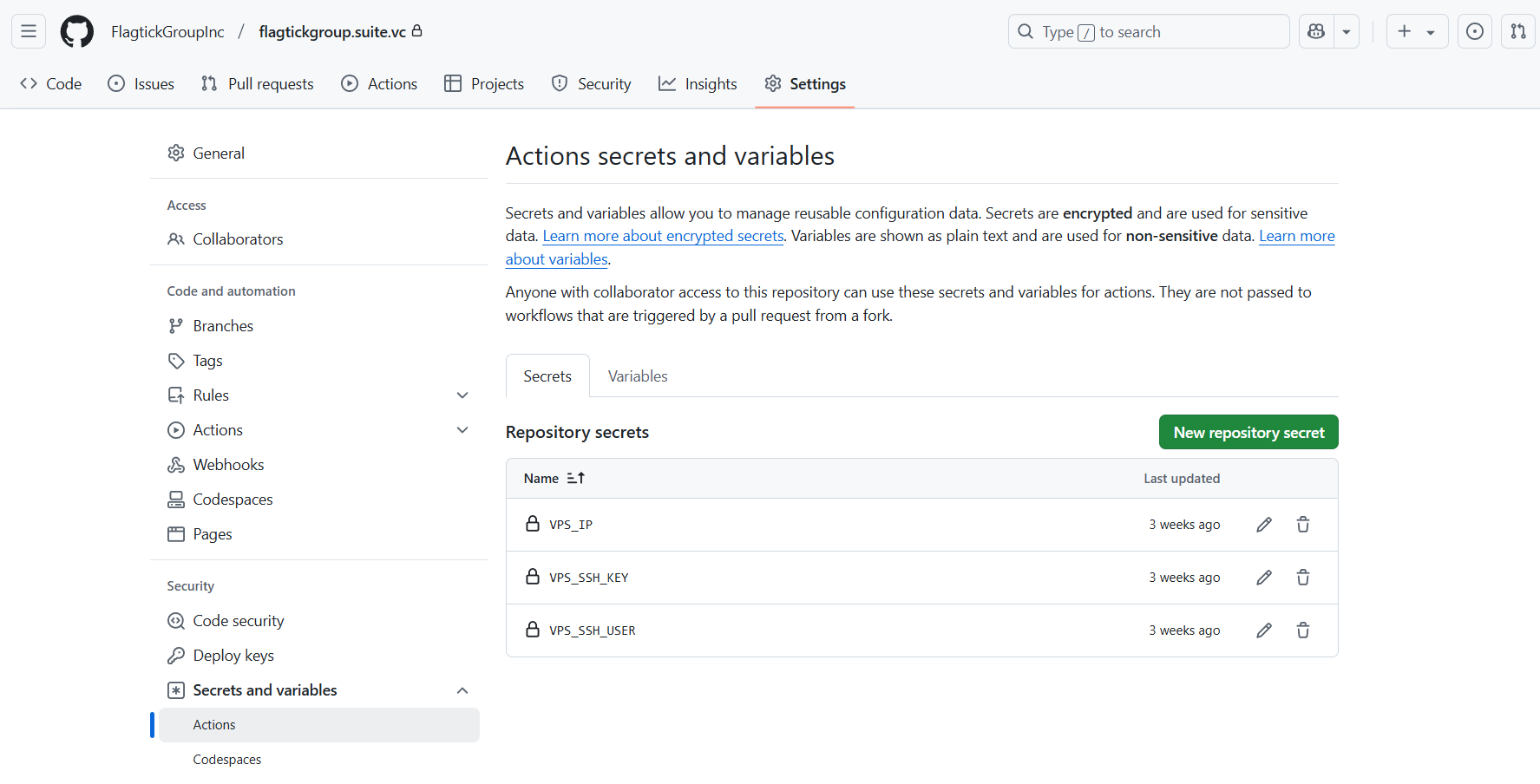
cd ../   
git status  
git add vc.flagtickgroup  
git commit -m "Update submodule vc.flagtickgroup to latest commit"

git push origin master

**Note:** Both vc.flagtickgroup and flagtickgroup.suite.vc use the master branch as the primary branch for deploying to production.



1. Setting Up Flagtickgroup.suite.vc and Establishing GitHub Actions



**Note:** In the context of GitHub Actions, VPS\_SSH\_KEY would typically be the private key used for SSH authentication to your VPS (Virtual Private Server). The private key is used to authenticate the action on your VPS server, while the public key should already be added to the ~/.ssh/authorized\_keys file on your VPS.

# **Orchestrating Multi-Containers**

The purpose is to analyze images, containers, and start essential services to run flagtickgroup.com, along with configuring the alias domain to resolve the IP 127.0.0.1 on the local machine.

1. Setting Up NGINX in Docker

nginx:  
 build:  
 context: .  
 dockerfile: ./docker/nginx/Dockerfile  
 container\_name: nginx  
 ports:  
 - "80:80"  
 - "443:443"  
 volumes:  
 - ./vc.flagtickgroup:/var/www/vc.flagtickgroup:delegated  
 - ./docker/nginx/conf.d:/etc/nginx/conf.d  
 - ./docker/nginx/timeout.conf:/etc/nginx/conf.d/timeout.conf  
 depends\_on:  
 - php  
 - mysql  
 networks:  
 - web\_server

**Note:** Any changes made in the ./vc.flagtickgroup directory on your host machine will automatically reflect inside the container at /var/www/vc.flagtickgroup without needing to restart or rebuild the Docker container. This is because Docker volumes are designed to sync the data between the host and the container in real time.

Restart the NGINX Server (Option 1: Reload the NGINX server to apply configuration changes without stopping the container)

docker exec nginx -s reload

Restart the NGINX Server (Option 2: Restart the NGINX Container to stop and start the entire NGINX container but won't rebuild the image.)

docker restart nginx

Rebuild the NGINX Server (Rebuild the container because of changes in the Dockerfile or other build-related files)

docker-compose build nginx   
docker-compose up -d nginx

NGINX Exposed and Served on Ports

docker ps --filter "name=nginx"

1. Identify the Service served on Ports 80 and 443

As you can see, port 80 and 443 configurations will be placed in ./docker/nginx/conf.d, where we have two sibling folders named conf.d.local and conf.d.prod.

* conf.d.local (local.flagtickgroup.com.conf)

upstream phpmyadmin {  
 server phpmyadmin:80;   
}  
  
upstream sonarqube {  
 server sonarqube:9000;  
}  
  
server {  
 listen 80;  
 server\_name local.flagtickgroup.com;  
  
 location / {  
 proxy\_pass http://phpmyadmin/;   
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_set\_header X-Forwarded-Proto $scheme;  
 }  
}  
  
server {  
 listen 443 ssl;  
 server\_name local.flagtickgroup.com;  
  
 ssl\_certificate /etc/ssl/flagtickgroup.com.crt;  
 ssl\_certificate\_key /etc/ssl/flagtickgroup.com.key;  
  
 root /var/www/vc.flagtickgroup/public;  
 index index.php index.html;  
  
 location / {  
 try\_files $uri $uri/ /index.php?$query\_string;  
 }  
  
 location ~ \.php$ {  
 try\_files $uri =404;  
 fastcgi\_split\_path\_info ^(.+\.php)(/.+)$;  
 fastcgi\_pass php:9000; # Adjust for your PHP backend (docker or otherwise)  
 fastcgi\_index index.php;  
 include fastcgi\_params;  
 fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;  
 fastcgi\_param PATH\_INFO $fastcgi\_path\_info;  
 }  
}  
  
server {  
 listen 443 ssl;  
 server\_name local-sonar.flagtickgroup.com;  
  
 ssl\_certificate /etc/ssl/flagtickgroup.com.crt;  
 ssl\_certificate\_key /etc/ssl/flagtickgroup.com.key;  
  
 location / {  
 proxy\_pass http://sonarqube;   
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_set\_header X-Forwarded-Proto $scheme;  
 }  
}

**Note:** We will configure the /etc/hosts file on the local machine as shown below to alias 127.0.0.1 under the domain mapping in the local.flagtickgroup.com.conf file.

127.0.0.1 local.flagtickgroup.com local-sonar.flagtickgroup.com

These endpoints can be accessed from the local machine, as shown below:

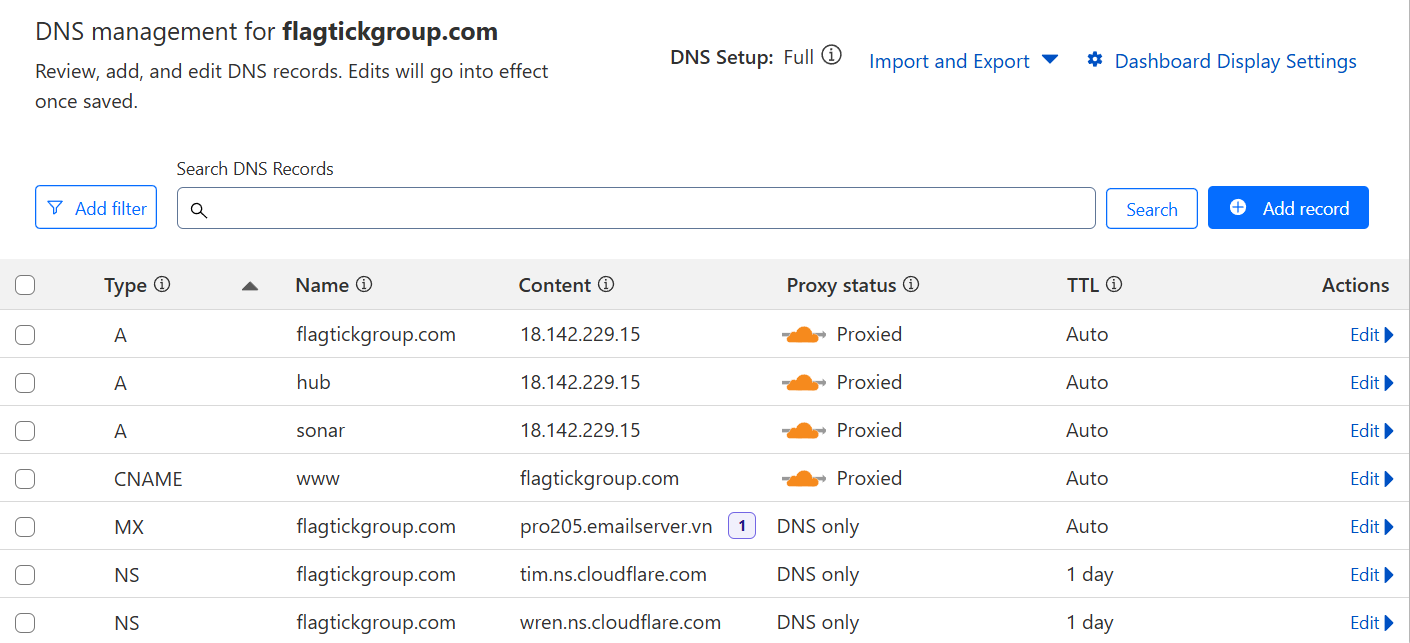
1. <https://local.flagtickgroup.com> (flagtick group site)
2. <http://local.flagtickgroup.com> (phpmyadmin)
3. <https://local-sonar.flagtickgroup.com> (solar service)

**Note:** Let copy the files from conf.d.local to override conf.d, and the Dockerfile should be set up as shown below:

# ADD ./docker/nginx/conf.d/flagtickgroup.com.conf /etc/nginx/conf.d/flagtickgroup.com.conf  
# ADD ./docker/nginx/conf.d/sub.flagtickgroup.com.conf /etc/nginx/conf.d/sub.flagtickgroup.com.conf  
  
# ADD ./resources/docker/nginx/default.conf /etc/nginx/conf.d/default.conf  
ADD ./docker/nginx/conf.d/local.flagtickgroup.com.conf /etc/nginx/conf.d/local.flagtickgroup.com.conf

* conf.d.prod (flagtickgroup.com.conf and sub.flagtickgroup.com.conf)

This setup is similar to the local machine, but with Cloudflare, and there will be some differences for the remote server. Here’s what we will cover:



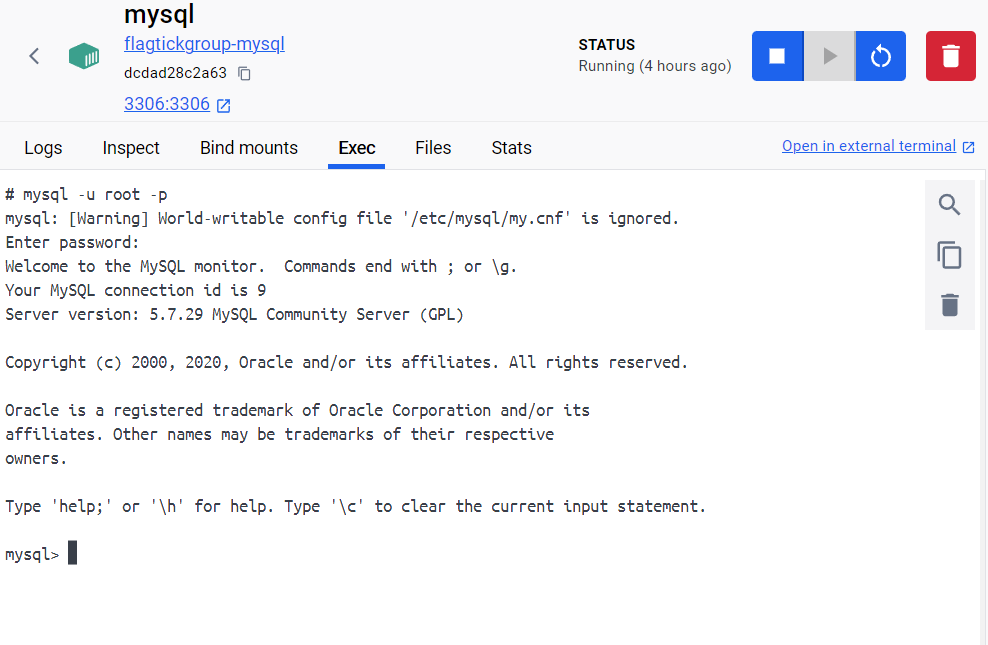
As you can see here, we will configure the domain flagtickgroup.com and subdomains such as hub.flagtickgroup.com and sonar.flagtickgroup.com.

Next, configure the /etc/hosts file on the remote server to map 127.0.0.1 to the domain and subdomains. These will be registered in Cloudflare and point to the public IP of the remote server.

$ cat /etc/hosts  
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4  
18.142.229.15 flagtickgroup.com  
18.142.229.15 www.flagtickgroup.com  
18.142.229.15 hub.flagtickgroup.com sonar.flagtickgroup.com  
::1 localhost6 localhost6.localdomain6

These endpoints can be accessed from the remote server, as shown below:

1. <https://www.flagtickgroup.com>
2. <https://sonar.flagtickgroup.com>
3. <https://hub.flagtickgroup.com>
4. Setting Up MySQL in Docker - this part of docker-compose.yml file, which is used to define and run MySQL container within Docker environment.



**Note:** Log into MySQL as the root user, you would use the root password set by the environment variable DB\_ROOT\_PASSWORD.

If you are using phpMyAdmin to access MySQL, you would log in with:

* Username**:** root
* Password**:** The value stored in the environment variable DB\_ROOT\_PASSWORD

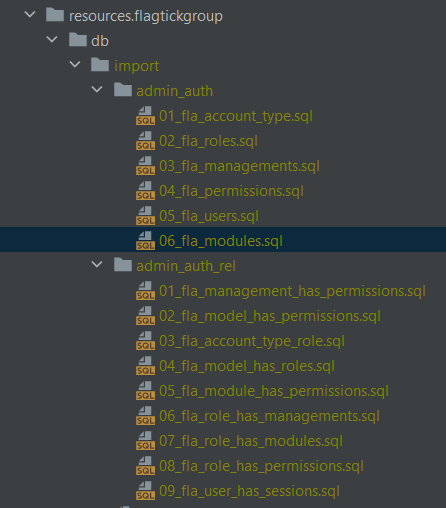
Using configuration ensures that the server supports all Unicode characters, which is particularly important for modern applications.

# Character Set  
character-set-server = utf8mb4  
collation-server = utf8mb4\_unicode\_ci

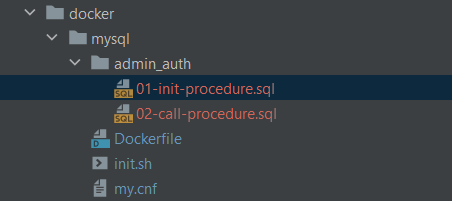
Copy the SQL files (containing schema and table data) into the container, specifically into the directory that MySQL automatically processes during startup.

COPY ./../../resources.flagtickgroup/db/import/admin\_auth.sql /docker-entrypoint-initdb.d/  
COPY ./../../resources.flagtickgroup/db/import/admin\_auth\_data.sql /docker-entrypoint-initdb.d/

Perhaps the problem lies in how we can design the data and implement our database design (including schema and data) into the MySQL container in a Docker environment.



Use PROCEDURE to dynamically create the table, and outline the script clearly with comments and proper formatting to make it easy to understand and execute the command.



Using SQL code defines a stored procedure named GenerateCreateTableFromJSON in MySQL that generates and executes CREATE TABLE SQL statement dynamically based on the structure and attributes provided in JSON input.

DELIMITER //  
  
DROP PROCEDURE IF EXISTS GenerateCreateTableSQL;  
  
CREATE PROCEDURE *GenerateCreateTableSQL*(  
 IN table\_name VARCHAR(255),  
 IN column\_definitions TEXT,  
 IN primary\_key TEXT,  
 IN indexes TEXT,  
 IN foreign\_keys TEXT,  
 IN engine\_definition VARCHAR(255),  
 IN charset\_definition VARCHAR(255)  
)  
BEGIN  
 IF engine\_definition IS NULL THEN  
 SET engine\_definition = 'ENGINE=InnoDB';  
END IF;  
  
 IF charset\_definition IS NULL THEN  
 SET charset\_definition = 'DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci';  
END IF;  
  
 SET @sql = CONCAT(  
 'CREATE TABLE IF NOT EXISTS `', table\_name, '` (',  
 column\_definitions  
 );  
  
 IF primary\_key IS NOT NULL THEN  
 SET @sql = CONCAT(@sql, ', ', primary\_key);  
END IF;  
  
 IF indexes IS NOT NULL THEN  
 SET @sql = CONCAT(@sql, ', ', indexes);  
END IF;  
  
 IF foreign\_keys IS NOT NULL THEN  
 SET @sql = CONCAT(@sql, ', ', foreign\_keys);  
END IF;  
  
 SET @sql = CONCAT(@sql, ') ', engine\_definition, ' ', charset\_definition, ';');  
  
PREPARE stmt FROM @sql;  
EXECUTE stmt;  
DEALLOCATE PREPARE stmt;  
END //  
  
DELIMITER ;

We will use the following script to create the PROCEDURE in the 01-init-procedure.sql file, and the CALL to execute the procedure will be written in the 02-call-procedure.sql file.

-- Call the stored procedure to dynamically create a table  
CALL GenerateCreateTableSQL(  
 -- Table name  
 'fla\_model\_has\_permissions',  
  
 -- Column definitions  
 '  
 `permission\_id` bigint(20) UNSIGNED NOT NULL,  
 `model\_type` varchar(191) NOT NULL,  
 `model\_id` bigint(20) UNSIGNED NOT NULL  
 ',  
  
 -- Primary key definition  
 'PRIMARY KEY (`permission\_id`, `model\_id`, `model\_type`)',  
  
 -- Indexes  
 'KEY `model\_has\_permissions\_model\_id\_model\_type\_index` (`model\_id`, `model\_type`)',  
  
 -- Foreign key constraints  
 '  
 CONSTRAINT `model\_has\_permissions\_permission\_id\_foreign`  
 FOREIGN KEY (`permission\_id`) REFERENCES `fla\_permissions` (`id`)  
 ON DELETE CASCADE,  
  
 CONSTRAINT `model\_has\_permissions\_model\_id\_foreign`  
 FOREIGN KEY (`model\_id`) REFERENCES `fla\_users` (`id`)  
 ON DELETE CASCADE  
 ',  
  
 -- Engine definition  
 'ENGINE=InnoDB',  
  
 -- Charset and collation definition  
 'DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci'  
);

Regarding the Dockerfile, we will use the COPY command as shown below to move all .sql scripts into the MySQL container for setup.

COPY ./docker/mysql/admin\_auth/\*.sql /docker-entrypoint-initdb.d/admin\_auth/  
RUN chmod 644 /docker-entrypoint-initdb.d/admin\_auth/\*.sql \  
 && chmod +x /usr/local/bin/init.sh

In the init.sh file, we also need to modify the script to load all .sql files and use MySQL to execute them.

# Loop through all .sql files in the admin\_auth directory and run them  
for sql\_file in /docker-entrypoint-initdb.d/admin\_auth/\*.sql; do  
 if [ -f "$sql\_file" ]; then  
 echo "Running SQL file: $sql\_file"  
 mysql -u root -p"$MYSQL\_ROOT\_PASSWORD" admin\_auth < "$sql\_file"  
 fi  
done

1. Setting up PhpMyAdmin in Docker – This is part of the process of using a database management tool to manage databases like MySQL. It provides a user-friendly GUI for performing database manipulations with ease.

phpmyadmin:  
 image: phpmyadmin/phpmyadmin  
 container\_name: phpmyadmin  
 environment:  
 MYSQL\_ROOT\_PASSWORD: ${DB\_ROOT\_PASSWORD}  
 UPLOAD\_LIMIT: 300M  
 PMA\_HOST: ${DB\_HOST}  
 PMA\_PORT: 3306  
 PMA\_ARBITRARY: 1  
 restart: always  
 depends\_on:  
 - mysql  
 volumes:  
 - /sessions  
 networks:  
 - web\_server  
 links:  
 - mysql

Ensure that the .env file matches the configuration in docker-compose.yml by setting the DB\_HOST and DB\_ROOT\_PASSWORD variables, and link them with phpMyAdmin.

# MYSQL Password  
DB\_ROOT\_PASSWORD=secret  
DB\_HOST=mysql

# **Orchestrating PHP-FPM & Debugging**

Set up a Laravel app, connect it to MySQL using PHP-FPM, and configure it with Nginx server to run the Laravel application.

1. Set Up PHP-FPM Container with Specific PHP Version

php:  
 build:  
 context: .  
 dockerfile: ./docker/php/Dockerfile  
 container\_name: php  
 volumes:  
 - ./vc.flagtickgroup:/var/www/vc.flagtickgroup:delegated  
 - ./docker/php/xdebug.ini:/usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini  
 ports:  
 - "9000:9000"  
 networks:  
 - web\_server

Use the upstream directive to proxy PHP-FPM (host) and expose port 9000 in the Nginx server to load the Laravel application. The php service is configured to build from the specified Dockerfile, mount the necessary volumes, and expose port 9000 for communication with the Nginx server.

$ docker ps --filter "name=php"

Expose port 9000 for the PHP container service, use the upstream directive, and configure the Dockerfile for the Nginx server to proxy requests to PHP-FPM.

upstream php-backend {  
 server php:9000;   
}

server {  
 listen 443 ssl;  
 server\_name local.flagtickgroup.com;  
  
 ssl\_certificate /etc/ssl/flagtickgroup.com.crt;  
 ssl\_certificate\_key /etc/ssl/flagtickgroup.com.key;  
  
 root /var/www/vc.flagtickgroup/public;  
 index index.php index.html;  
  
 location / {  
 try\_files $uri $uri/ /index.php?$query\_string;  
 }  
  
 location ~ \.php$ {  
 try\_files $uri =404;  
 fastcgi\_split\_path\_info ^(.+\.php)(/.+)$;  
 fastcgi\_pass php-backend;   
 fastcgi\_index index.php;  
 include fastcgi\_params;  
 fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;  
 fastcgi\_param PATH\_INFO $fastcgi\_path\_info;  
 }  
}

Set up local.flagtickgroup.com as a domain alias by mapping it to 127.0.0.1 in the /etc/hosts file on your local machine.

127.0.0.1 local.flagtickgroup.com local-sonar.flagtickgroup.com

On Windows, ping does not use the -c option for count. Instead, it uses -n. To specify the number of packets to send:

C:\Users\admin>ping -n 4 local.flagtickgroup.com  
  
Pinging local.flagtickgroup.com [127.0.0.1] with 32 bytes of data:  
 Reply from 127.0.0.1: bytes=32 time<1ms TTL=128  
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128  
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128  
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128  
  
Ping statistics for 127.0.0.1:  
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
Approximate round trip times in milli-seconds:  
Minimum = 0ms, Maximum = 0ms, Average = 0ms

As you can see, the domain local.flagtickgroup.com resolves to 127.0.0.1, and Nginx on the local machine successfully handles the HTTPS request, returning a 200 OK response.

C:\Users\admin>curl --verbose https://local.flagtickgroup.com  
\* Host local.flagtickgroup.com:443 was resolved.  
\* IPv6: (none)  
\* IPv4: 127.0.0.1  
\* Trying 127.0.0.1:443...  
\* Connected to local.flagtickgroup.com (127.0.0.1) port 443  
\* schannel: disabled automatic use of client certificate  
\* ALPN: curl offers http/1.1  
\* ALPN: server accepted http/1.1  
\* using HTTP/1.x  
> GET / HTTP/1.1  
> Host: local.flagtickgroup.com  
> User-Agent: curl/8.9.1  
> Accept: \*/\*  
>  
\* Request completely sent off  
\* schannel: failed to decrypt data, need more data  
< HTTP/1.1 200 OK  
< Server: nginx/1.26.2

Using nginx -T command will dump the entire Nginx configuration, including all included files, to the console.

# nginx -T | grep -i 'proxy\_pass'  
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok  
nginx: configuration file /etc/nginx/nginx.conf test is successful  
 proxy\_pass http://phpmyadmin/;  
 proxy\_pass http://sonarqube;

As you can see here, /phpmyadmin will route to phpMyAdmin on port 80, and /sonarcube will route to PHP on port 9000, and so on.

**Note:** Use the command docker-compose up -d --scale nginx=1 --force-recreate to replace the existing version of the nginx container while ensuring other services continue running without interruption. This approach avoids a full rebuild of all containers and focuses only on the nginx service.

docker-compose pull nginx  
docker-compose up -d --scale nginx=1 --force-recreate

or, we can bypass which heavy services and focus on setting up and launching the necessary services on the local machine.

docker-compose up --build --scale composer=0 --scale artisan=0 --scale terraform=0 --scale sonar-scanner=0

1. Using Composer with PHP-FPM in Docker

Run docker-compose run --rm composer require <package-name>, which uses the composer service defined in the docker-compose.yml to install a new package and update composer.json in the Laravel application mounted to the container environment.

composer:  
 build:  
 context: .  
 dockerfile: ./docker/php/composer/Dockerfile  
 container\_name: composer  
 volumes:  
 - ./vc.flagtickgroup:/var/www/vc.flagtickgroup  
 working\_dir: /var/www/vc.flagtickgroup  
 depends\_on:  
 - php  
 user: flagtick  
 networks:  
 - web\_server  
 entrypoint: ['composer']

Using command docker-compose up -d composer is used to start the composer service defined in the docker-compose.yml file in detached mode, running it in the background to install or manage PHP dependencies using Composer without blocking the terminal.

docker-compose up -d composer

If the composer service is set up in the docker-compose.yml, use docker-compose run --rm composer require <package-name> to install package for the vc.flagtickgroup Laravel app, updating composer.json within the container's /var/www/vc.flagtickgroup directory.

docker-compose run --rm composer require <package-name>

**Note:** --rm option in the docker-compose run --rm composer require <package-name> command ensures that the container is removed automatically after the command finishes executing.

In reality, you may occasionally encounter the "**could not find driver**" issue because the MySQL service has not started yet when running docker-compose run --rm composer dump-autoload. Therefore, ensure that the dependent services, such as PHP and MySQL, are running before executing the command.

composer:  
 build:  
 context: .  
 dockerfile: ./docker/php/composer/Dockerfile  
 container\_name: composer  
 volumes:  
 - ./vc.flagtickgroup:/var/www/vc.flagtickgroup  
 working\_dir: /var/www/vc.flagtickgroup  
 depends\_on:  
 - mysql  
 - php  
 user: flagtick  
 networks:  
 - web\_server  
 entrypoint: ['composer']

Install and enable specific PHP extensions within PHP-based Docker container that provide an interface for accessing MySQL databases, a consistent interface for accessing various types of databases, and support for PDO to work with MySQL databases.

RUN docker-php-ext-install mysqli pdo pdo\_mysql

Another example is how to update and regenerate the autoload files (vendor/autoload.php) after changes are made to the project's composer.json file or file structure.

docker-compose run --rm composer dump-autoload

**Note:** Furthermore, ensure that docker-compose is executed from the directory containing the docker-compose.yml file. Verify that the composer image is already set up and linked to other images using depends\_on to prevent errors.

We can include bash in the Composer container, you can extend its base image by updating your Dockerfile as follows.

RUN apk add --no-cache bash

1. Using Artisan tool with PHP-FPM in Docker

First, execute the Artisan tool using PHP because it is a PHP script, not a standalone executable like Composer.

entrypoint: [ 'php', '/var/www/vc.flagtickgroup/artisan' ]

Here is a fully set up Artisan service, specifically configured to run Laravel Artisan CLI, which handles tasks like migrations, queue management, and more.

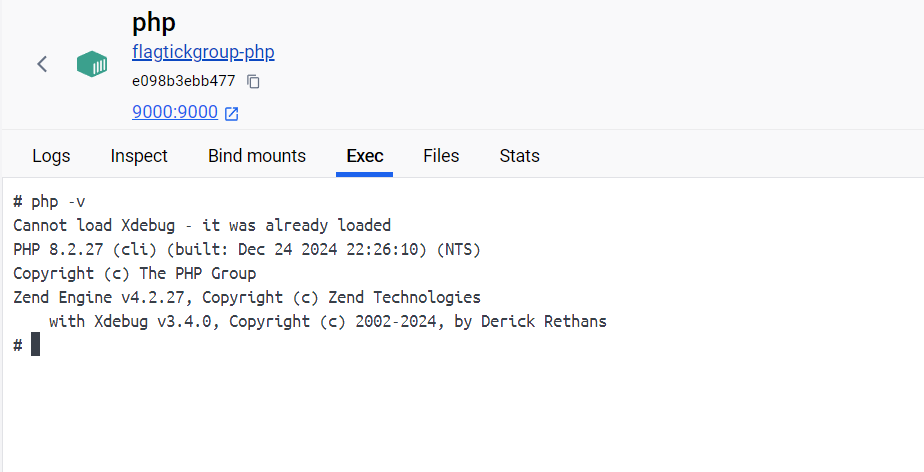
artisan:  
 build:  
 context: .  
 dockerfile: ./docker/php/Dockerfile  
 container\_name: artisan  
 volumes:  
 - ./vc.flagtickgroup:/var/www/vc.flagtickgroup:delegated  
 depends\_on:  
 - mysql  
 working\_dir: /var/www/vc.flagtickgroup  
 user: flagtick  
 entrypoint: [ 'php', '/var/www/vc.flagtickgroup/artisan' ]  
 networks:  
 - web\_server

Let check the Laravel version through the Artisan CLI within a Docker container.

docker-compose run --rm artisan –version

1. Configuring Xdebug in IDE (VSCode or PHPStorm)

First, access the PHP container environment and use php -v to check the Xdebug version or verify if Xdebug is installed.



Additionally, you can access the container using the docker exec command.

$ docker exec -it php sh  
# php -v  
Cannot load Xdebug - it was already loaded  
PHP 8.2.27 (cli) (built: Dec 24 2024 22:26:10) (NTS)  
Copyright (c) The PHP Group  
Zend Engine v4.2.27, Copyright (c) Zend Technologies  
 with Xdebug v3.4.0, Copyright (c) 2002-2024, by Derick Rethans  
#

Install and enable Xdebug, then copy the Xdebug configuration.

# Install and enable xdebug  
RUN apt-get update && apt-get install -y $*PHPIZE\_DEPS* \  
 && pecl install xdebug \  
 && docker-php-ext-enable xdebug \  
 && apt-get clean \  
 && rm -rf /var/lib/apt/lists/\*   
  
# Copy Xdebug configuration  
RUN mkdir -p /var/log && chown -R flagtick:flagtickgroup /var/log  
COPY ./docker/php/xdebug.ini /usr/local/etc/php/conf.d/xdebug.ini

Create an xdebug.ini file in the /docker/php folder and update the script as shown below to enable debugging in Visual Studio Code.

zend\_extension=xdebug.so  
  
xdebug.mode=debug  
xdebug.start\_with\_request=yes  
xdebug.client\_host=host.docker.internal  
xdebug.client\_port=9002  
xdebug.log=/var/log/xdebug.log  
xdebug.log\_level=7  
xdebug.idekey=VSCODE

Create launch.json for PHP Debugging: Open the VSCode workspace where you want to set up debugging.

{

    "version": "0.2.0",

    "configurations": [

      {

        "name": "FlagtickGroup Debug",

        "type": "php",

        "request": "launch",

        "port": 9002,

        "log": true,

        "pathMappings": {

          "/var/www/vc.flagtickgroup": "${workspaceFolder}"

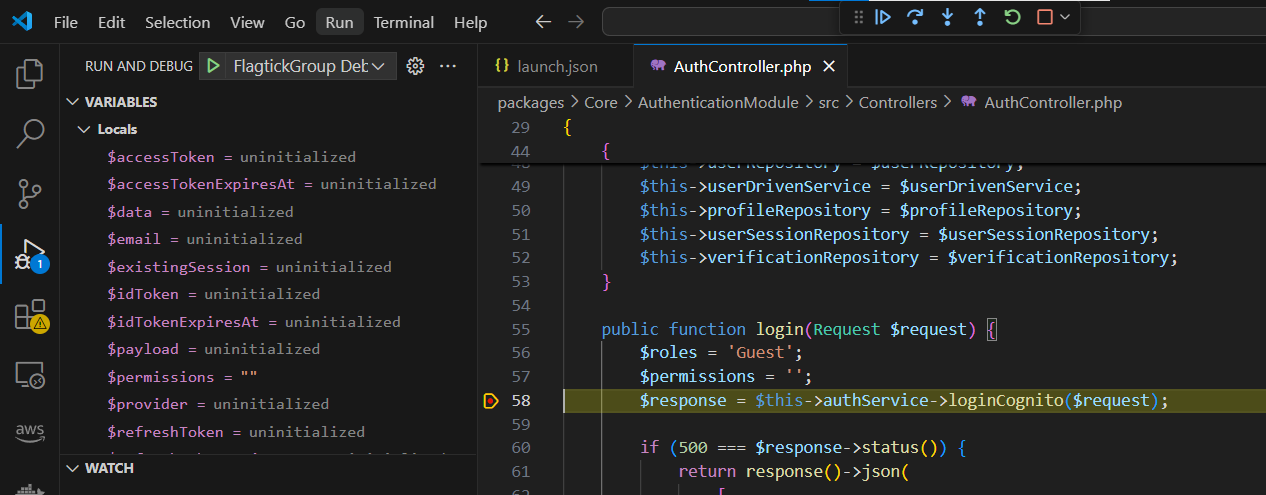
        }

      }

    ]

}

Go to browser like Google Chrome, install the Xdebug extension, and configure it to target the VSCode debugger after triggering the debug session through the launch.json file.



1. Step-by-Step Guide to Debugging PHP on PHPStorm

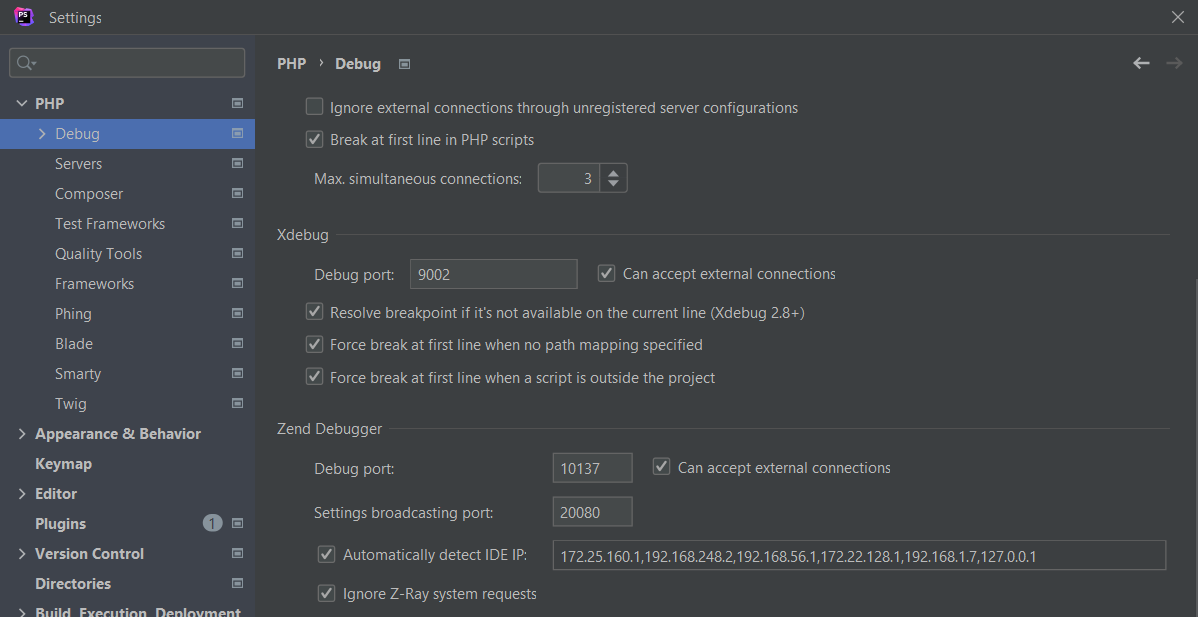
Start by updating the docker-compose.yml file to add the necessary environment variables to the PHP container, as shown below.

php:  
 build:  
 context: .  
 dockerfile: ./docker/php/Dockerfile  
 container\_name: php  
 volumes:  
 - ./vc.flagtickgroup:/var/www/vc.flagtickgroup:delegated  
 - ./docker/php/xdebug.ini:/usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini  
 ports:  
 - "9000:9000"  
 environment:  
 - XDEBUG\_MODE=debug  
 - XDEBUG\_CONFIG="remote\_host=host.docker.internal remote\_port=9002 remote\_enable=1"  
 networks:  
 - web\_server

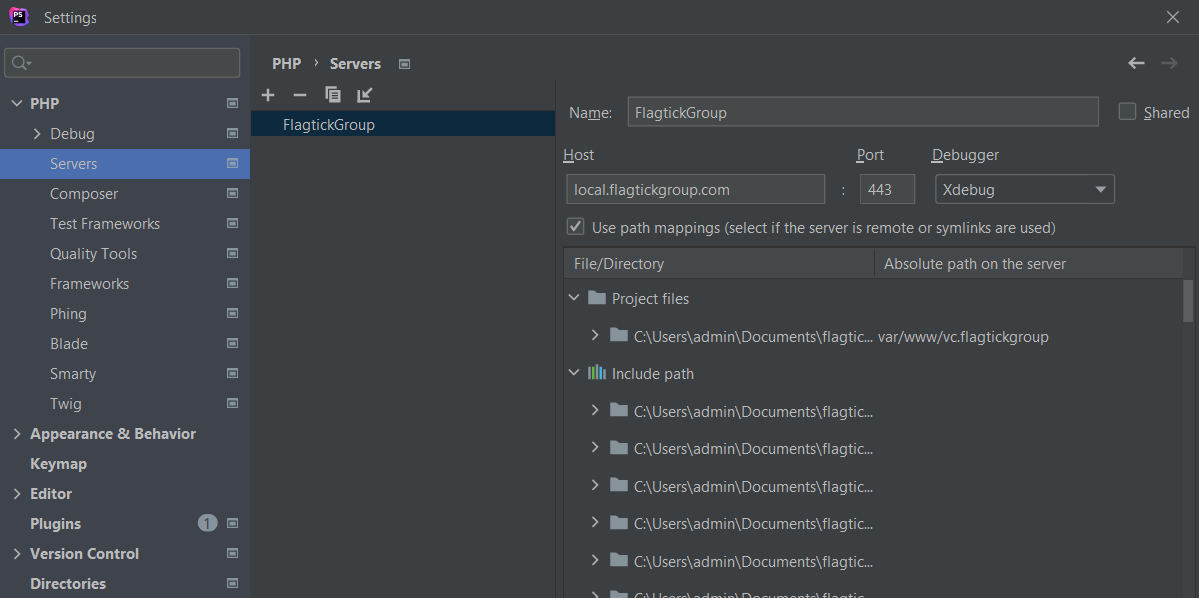
Modify the xdebug.ini file in docker/php to route debugging to PHPStorm as the default IDE.

zend\_extension=xdebug.so  
xdebug.mode=debug  
xdebug.start\_with\_request=yes  
xdebug.client\_host=host.docker.internal  
xdebug.client\_port=9002  
xdebug.idekey=PHPSTORM  
xdebug.remote\_enable=1  
xdebug.remote\_autostart=1  
xdebug.remote\_log=/tmp/xdebug.log

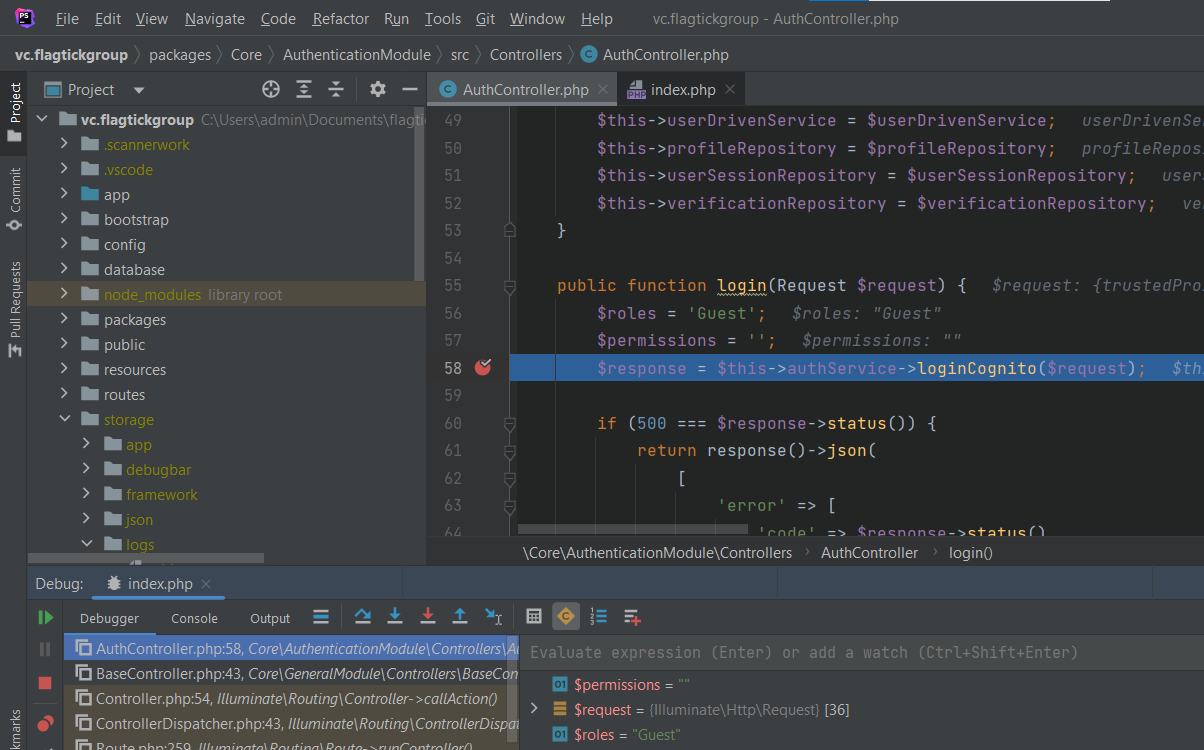
Open the vc.flagtickgroup project in PHPStorm (where the Laravel app is stored) instead of flagtickgroup.suite.vc. Set up the IDE to listen on port 9002 and configure path mappings to align with the Docker environment. Go to File > Settings > PHP > Debug, and configure the settings as shown in the screenshot below.



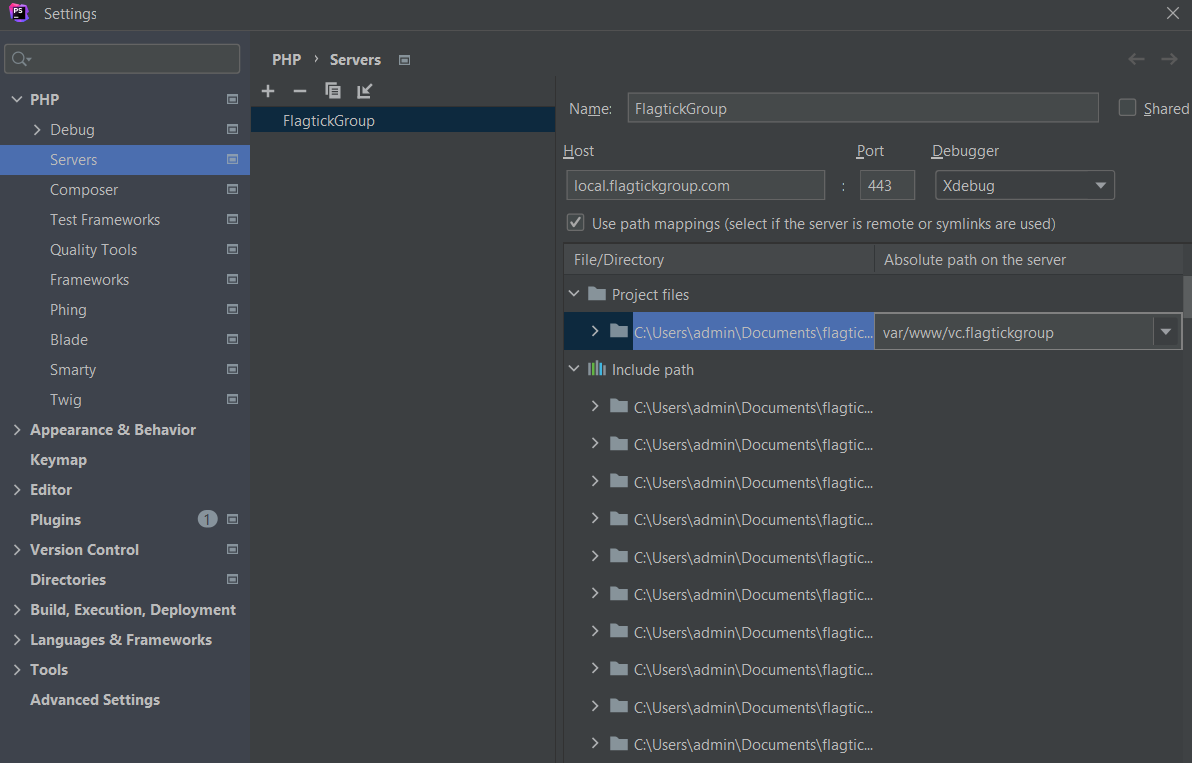
Switch to the Servers tab and configure the settings as shown in the screenshot.



Furthermore, start listening for PHP debug connections and enable the Xdebug extension in Google Chrome for support.



**Note:** Update the /var/www/vc.flagtickgroup path in the Servers tab to map correctly to the Docker environment for IDE debugging.

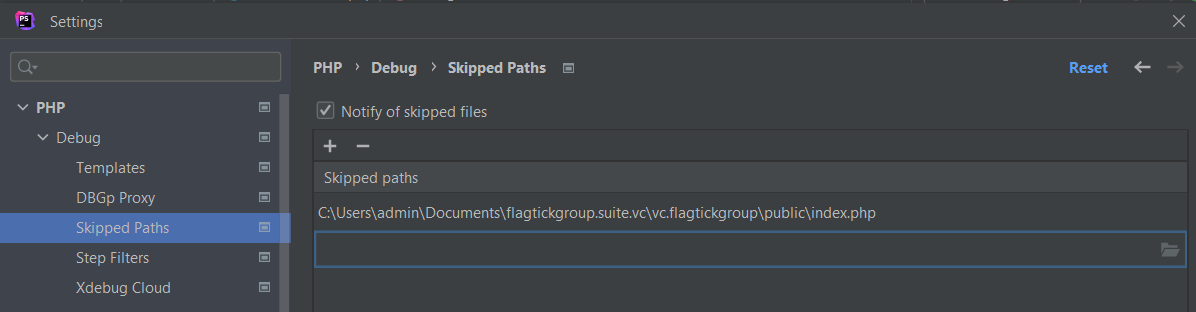


**Note:** If the debug port (e.g., 9002) is already in use, PHPStorm may fail to listen. In such cases, avoid exposing the port.

1. Evaluate Expression in Debugger

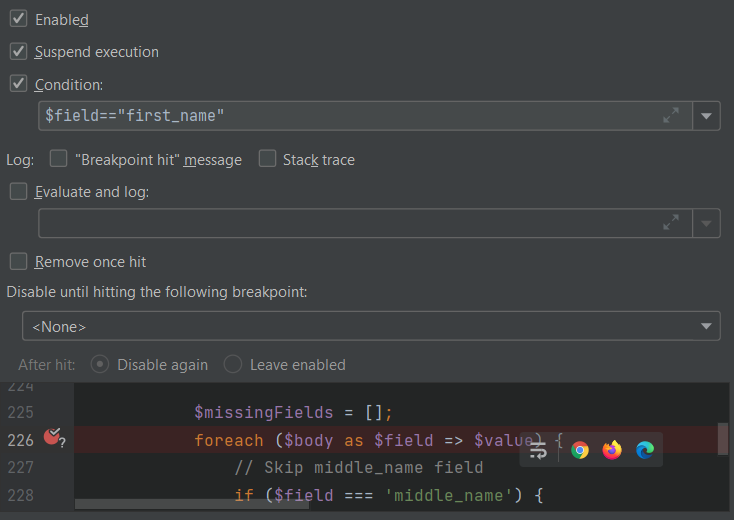
In PHPStorm, the Evaluate Expression feature allows you to inspect and test specific variables, expressions, or conditions during debugging. This helps step through files or loops with custom conditions, providing better insight into the program's behavior.

1. Skip Debugging or Breakpoints for Specific Files in PHPStorm



1. Conditional breakpoints in PHPStorm allow you to pause execution only when a specific condition is met, making debugging more efficient. You can set conditions based on variable values or expressions (e.g., i == 5 or $myVar == 'test').

For example, the debugger will pause the execution only when $field equals first\_name.



# **SonarQube for Code Quality**

Configure the docker-compose.yml file to include the necessary settings for SonarQube and its database (e.g., PostgreSQL).

1. SonarQube Setup in Docker Compose

sonarqube:  
 image: sonarqube:community  
 container\_name: sonarqube  
 ports:  
 - "9003:9000"  
 volumes:  
 - ./vc.flagtickgroup:/var/src/vc.flagtickgroup  
 environment:  
 SONAR\_JDBC\_URL: jdbc:postgresql://postgres:5432/sonarqube  
 SONAR\_JDBC\_USERNAME: sonar  
 SONAR\_JDBC\_PASSWORD: sonar  
 depends\_on:  
 - postgres  
 networks:  
 - web\_server   
  
postgres:  
 image: postgres:13  
 container\_name: sonarqube-db  
 environment:  
 POSTGRES\_USER: sonar  
 POSTGRES\_PASSWORD: sonar  
 POSTGRES\_DB: sonarqube  
 volumes:  
 - sonarqube\_db\_data:/var/lib/postgresql/data  
 networks:  
 - web\_server  
 healthcheck:  
 test: [ "CMD", "pg\_isready", "-U", "sonar", "-d", "sonarqube" ]  
 interval: 10s  
 retries: 5  
  
sonar-scanner:  
 image: sonarsource/sonar-scanner-cli  
 container\_name: sonar-scanner  
 volumes:  
 - ./vc.flagtickgroup:/usr/src *# Codebase is in vc.flagtickgroup* working\_dir: /usr/src  
 networks:  
 - web\_server  
 depends\_on:  
 - sonarqube  
 entrypoint: [ "sonar-scanner" ]

1. SonarQube Default Login and Password Change Process

The default login for SonarQube is admin/admin. Upon first login, you will be prompted to change the password for security.

Username/Pwd: admin / ?(5PSRgrcwOL

Next, set up routing to access the SonarQube dashboard, with the default URL being https://local-sonar.flagtickgroup.com/. Update the /etc/hosts file accordingly.

127.0.0.1 local.flagtickgroup.com local-sonar.flagtickgroup.com

Enables secure access to SonarQube at https://local-sonar.flagtickgroup.com/ by routing traffic through Nginx, which manages SSL encryption and forwards the requests to the SonarQube service running in a Docker container.

server {  
 listen 443 ssl;  
 server\_name local-sonar.flagtickgroup.com;  
  
 ssl\_certificate /etc/ssl/flagtickgroup.com.crt;  
 ssl\_certificate\_key /etc/ssl/flagtickgroup.com.key;  
  
 location / {  
 proxy\_pass http://sonarqube;   
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_set\_header X-Forwarded-Proto $scheme;  
 }  
}

Set up the project key and project name in your self-hosted SonarQube instance and save them in sonar-project.properties file as shown below.

sonar.projectKey=FlagtickGroup  
sonar.projectName=FlagtickGroup  
sonar.sources=./  
sonar.scm.disabled=true  
sonar.projectVersion=1.0  
sonar.host.url=http://host.docker.internal:9003  
sonar.login=squ\_dbf8671a602a647832a80fa29aa491c68e99a810  
sonar.inclusions=\*\*/\*.php  
sonar.exclusions=\*\*/tests/\*\*, \*\*/vendor/\*\*, \*\*/storage/\*\*  
sonar.sources=app/Http/Controllers, app/Model

**Note:** The sonar-project.properties file should be placed in the project codebase, such as vc.flagtickgroup, to track the Flagtick Group project.

Running and Cleaning Up Sonar-Scanner with Docker Compose

docker-compose run --rm sonar-scanner

Continue…

# **Artillery for Load Testing**

We will conduct Artillery performance tests inside a Docker container, including the following tasks:

✅ Load testing REST APIs or GraphQL endpoints  
✅ Stress testing backend services  
✅ Generating traffic to simulate real-world usage  
✅ Analyzing response times and throughput

1. Define the Configuration for Using the Artillery Tool

artillery:  
 image: artilleryio/artillery  
 container\_name: artillery  
 volumes:  
 - ./resources/artillery:/scripts  
 - ./reports:/reports  
 working\_dir: /scripts  
 networks:  
 - web\_server  
 entrypoint: [ "npx", "artillery" ]

1. Define Scenarios for Testing in a YAML File Using the Artillery Tool

config:  
 target: "https://<Host>"  
 phases:  
 - duration: 60  
 arrivalCount: 100  
 rampTo: 200  
  
scenarios:  
 - name: "getBusiness1 Test"  
 flow:  
 - get:  
 url: "/dev/getBusiness1"  
 headers:  
 Host: "<Host>"  
 x-api-key: "<API Key>"  
 Content-Type: "application/json"  
 qs:  
 PersonID: 93982  
  
 - name: "getBusiness2 Test"  
 flow:  
 - get:  
 url: "/dev/getBusiness2"  
 headers:  
 Host: "<Host>"  
 x-api-key: "<API Key>"  
 Content-Type: "application/json"  
 qs:  
 PersonID: 93982   
  
 - name: "addBusiness3 Test"  
 flow:  
 - post:  
 url: "/dev/addBusiness3"  
 headers:  
 Host: "<Host>"  
 x-api-key: "<API Key>"  
 Content-Type: "application/json"  
 body:  
 [  
 {  
 "personid": 214972,  
 "addrtype": "H",  
 "status": "Y",  
 "edate": "2022-01-01T00:00:00.000Z",  
 "addressid": 0,  
 "street1": "325 King St Apt LD",  
 "street2": "",  
 "city": "Kew Gardens",  
 "state": "NY",  
 "zipcode": "11415",  
 "outofca": "Y"  
 }  
 ]

As an example in the script above, the scenario starts with 100 virtual users and gradually ramps up to 200 over 60 seconds, simulating increasing traffic to test system performance under load.

Let us perform a load test on an API and save the results for analysis.

docker-compose run --rm artillery run /scripts/api-test.yml -o /reports/report.json

The output **reports.json** is located in /reports. Let us mount /reports to the local /reports directory to export the **reports.json** file.

volumes:  
 - ./resources/artillery:/scripts  
 - ./reports:/reports

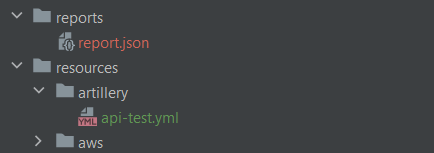
From the reports.json file, use Node.js Aggregate to perform the analysis, and the results will look like this:

### Summary: /dev/getBusiness1   
- 1100 requests sent, 1003 responses received (all HTTP 200 OK).   
- Failures: 97 requests failed due to ETIMEDOUT errors (representing ~8.8% failure rate).   
- Response times:   
 - Minimum: 328 ms   
 - Average: 2314.5 ms   
 - Median (P50): 497.8 ms   
 - 75th percentile (P75): 3905.8 ms   
 - 90th percentile (P90): 6976.1 ms   
 - 95th percentile (P95): 8024.5 ms   
 - 99th percentile (P99): 9230.4 ms   
 - Maximum spike: 9591 ms   
- Request rate: 22 requests per second on average.   
- Virtual users:   
 - 1100 sessions created, 1003 completed successfully, 97 failed due to timeouts.   
- Session duration:   
 - Average: 2717.4 ms   
 - Median: 907 ms   
 - Maximum: 9988.7 ms   
 - 99th percentile: 9607.1 ms

**Note:** Use artillery=0 to skip building or starting the service in the docker-compose command within the deploy.sh file.

docker-compose up -d --build --scale composer=0 --scale artisan=0 --scale terraform=0 --scale artillery=0 --scale sonar-scanner=0

The folder structure is defined as follows in flagtickgroup.suite.vc when using the Artillery tool.



# **Nexus Repository Manager**

Setting up Nexus Repository Manager to manage Laravel application packages involves deploying Nexus OSS (Open Source Software) repository and configuring it to serve as a local package registry.

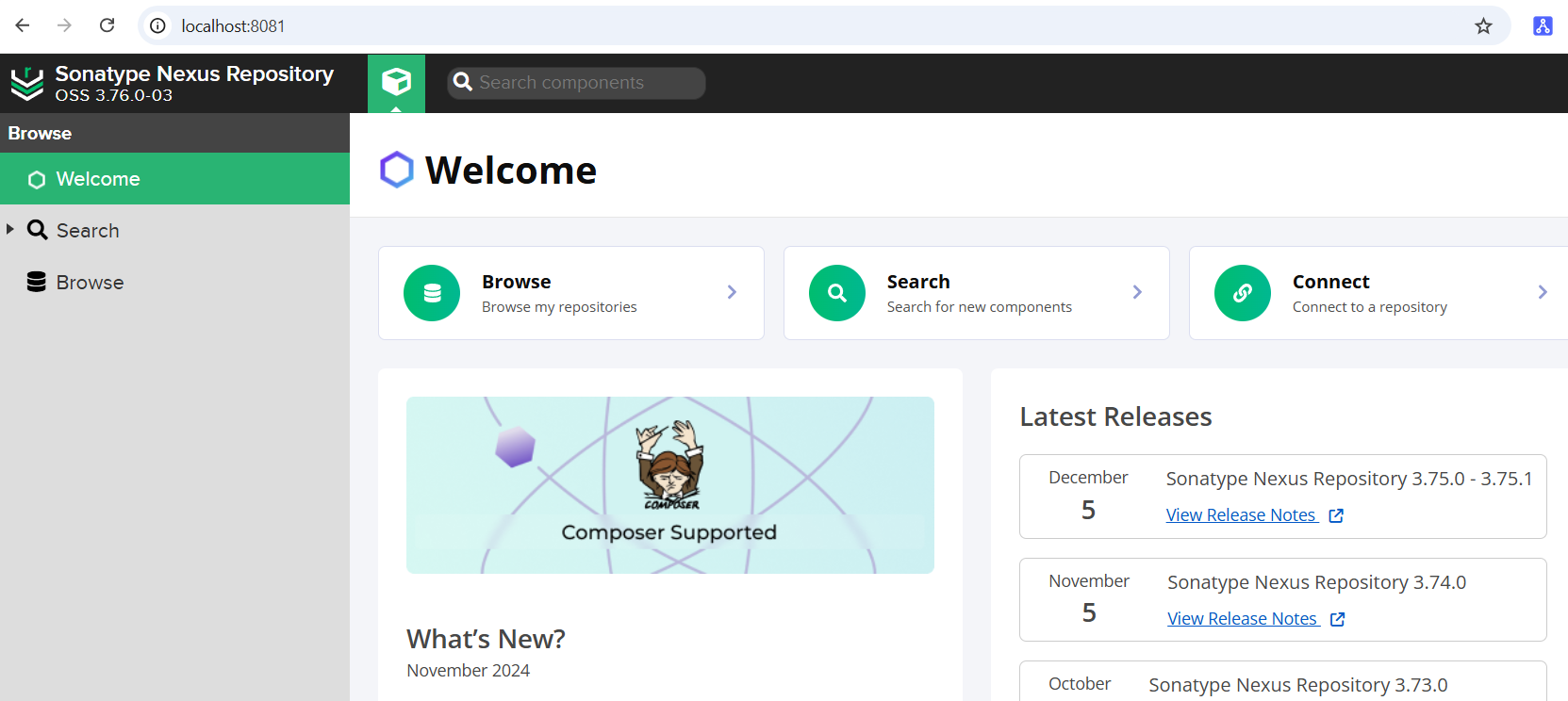
1. Deploy Nexus Repository Manager

nexus:  
 image: sonatype/nexus3  
 container\_name: nexus  
 ports:  
 - "8081:8081"  
 volumes:  
 - nexus-data:/nexus-data  
 environment:  
 - INSTALL4J\_ADD\_VM\_PARAMS=-Xms1200m -Xmx1200m -XX:MaxDirectMemorySize=2g -XX:+UnlockExperimentalVMOptions -XX:+UseG1GC -XX:InitiatingHeapOccupancyPercent=75 -XX:G1ReservePercent=25 -XX:SoftRefLRUPolicyMSPerMB=50 -XX:+AlwaysPreTouch -Djava.util.prefs.userRoot=/nexus-data/javaprefs  
 networks:  
 - web\_server

1. Deploy Nexus by running the updated docker-compose.yml file, then access it at <http://localhost:8081>.

docker-compose up -d

Open the browser to view the interface.

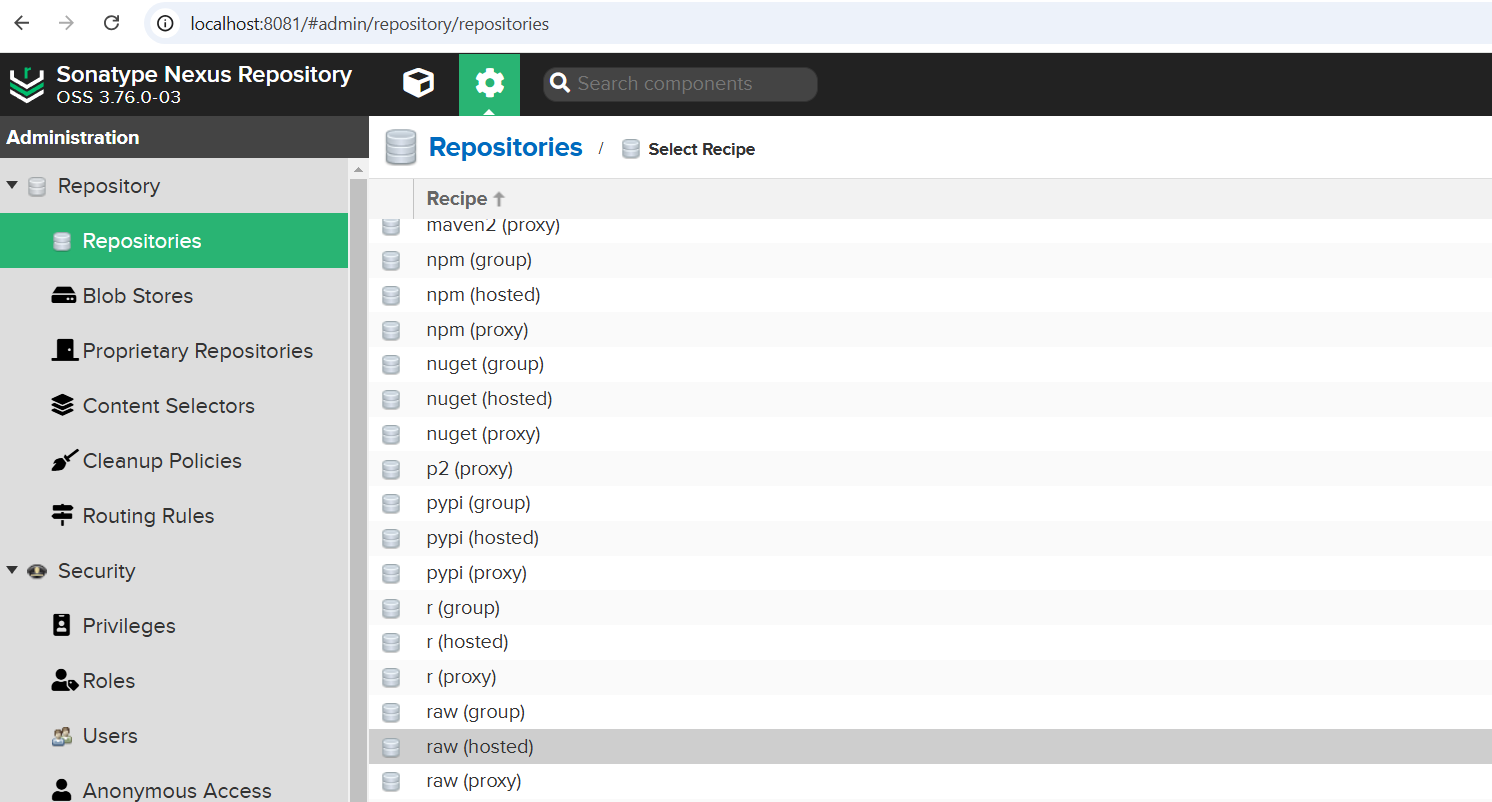


1. Configure Nexus for PHP Composer Packages

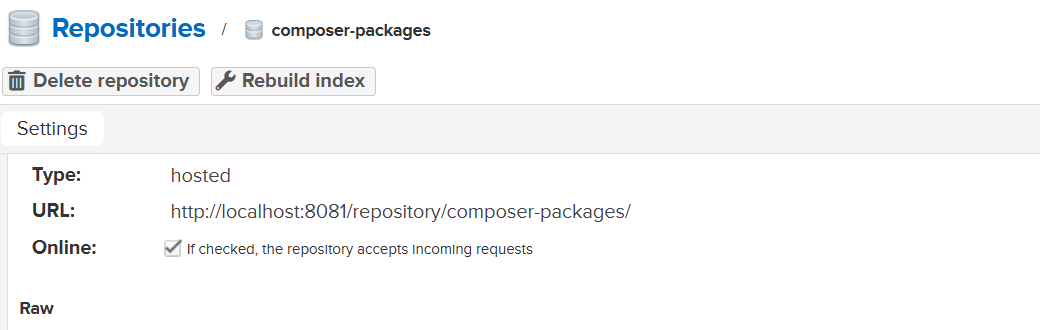
Log in using the username admin and the password retrieved by running the cat /nexus-data/admin.password command from the Nexus container environment. Below is a screenshot captured from the Nexus container for reference.

docker exec -it nexus bash  
cd  
cat /nexus-data/admin.password

For example, log in using the username admin and the password Abc@123456.



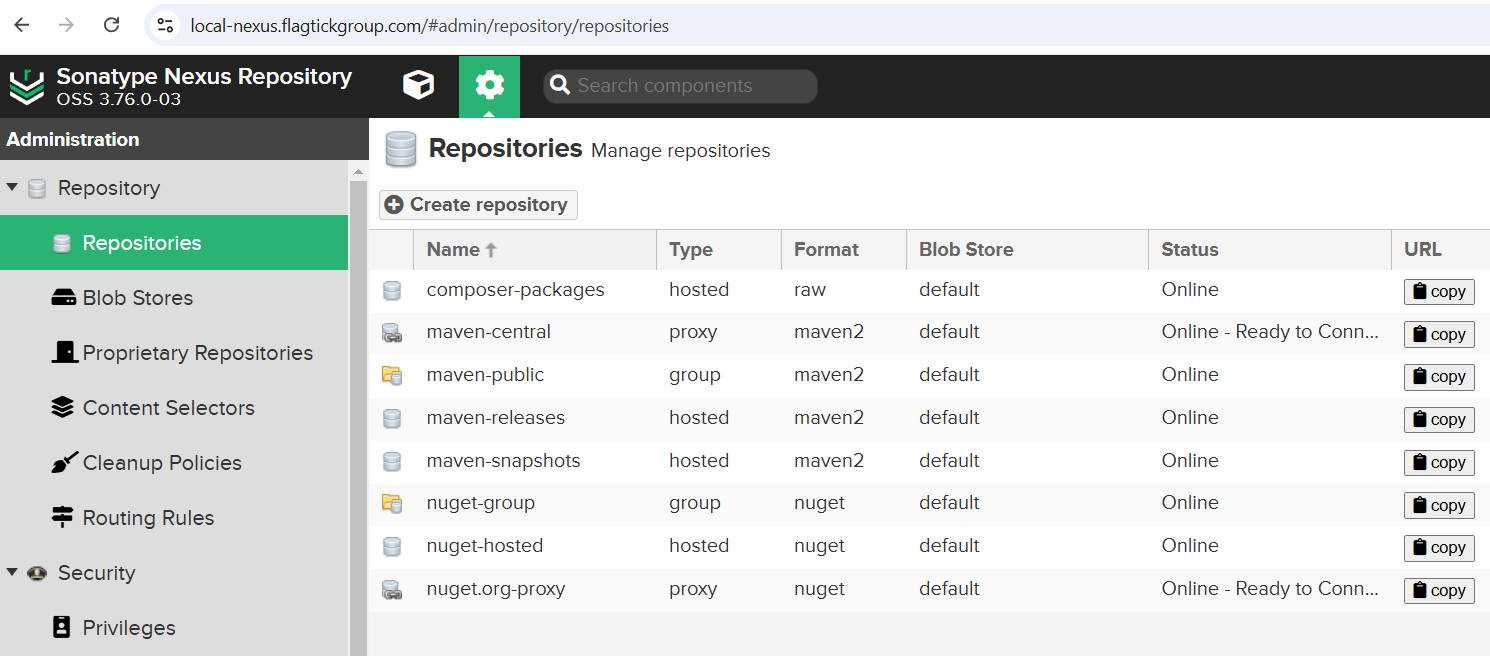
Create Raw Repository named composer-packages with the type set to Hosted and the Deployment Policy set to Allow Redeploy.



Modify the local.flagtickgroup.com.conf file to add an alias domain <https://local-nexus.flagtickgroup.com> and configure it to use upstream to proxy Nexus on port 8081.

upstream nexus {  
 server nexus:8081;  
}  
  
server {  
 listen 443 ssl;  
 server\_name local-nexus.flagtickgroup.com;  
  
 ssl\_certificate /etc/ssl/flagtickgroup.com.crt;  
 ssl\_certificate\_key /etc/ssl/flagtickgroup.com.key;  
  
 location / {  
 proxy\_pass http://nexus;   
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_set\_header X-Forwarded-Proto $scheme;  
 }  
}

**Note:** Remember to add local-nexus.flagtickgroup.com to the /etc/hosts file.



1. Configure Composer to Use the Nexus Repository

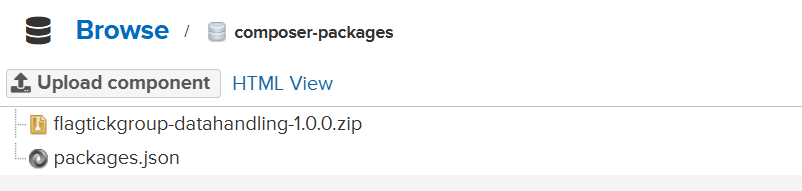
Update the composer.json file in your Laravel project to include the repository URL specified in the settings of the composer-packages repository.

"repositories": {  
 "nexus": {  
 "type": "composer",  
 "url": "https://nexus.flagtickgroup.com/repository/composer-packages/"  
 }  
}

**Note:** Let us configure Composer to pull packages from the specified Nexus repository using a command instead of adding it manually.

composer config repositories.nexus composer   
https://nexus.flagtickgroup.com/repository/composer-packages/

Here is a sample package description:



Furthermore, we will have another file is packages.json, which links to the package declared above.

{  
 "packages": {  
 "flagtickgroup/datahandling": {  
 "1.0.0": {  
 "name": "flagtickgroup/datahandling",  
 "version": "1.0.0",  
 "description": "FlagtickGroup Data Handling Package For Laravel.",  
 "homepage": "https://www.flagtickgroup.com",  
 "keywords": [  
 "laravel",  
 "php",  
 "data"  
 ],  
 "license": "MIT",  
 "authors": [  
 {  
 "name": "Flagtick Group",  
 "email": "admin@flagtickgroup.com",  
 "role": "Founder",  
 "homepage": "https://www.flagtickgroup.com"  
 }  
 ],  
 "require": {  
 "php": "^8.0.2",  
 "illuminate/support": "~5.0|~6.0|~7.0|^8.0|^9.0|^10.0"  
 },  
 "autoload": {  
 "psr-4": {  
 "FlagtickGroup\\DataHandling\\": "src/"  
 }  
 },  
 "extra": {  
 "laravel": {  
 "providers": [  
 "FlagtickGroup\\DataHandling\\Providers\\DataMappingServiceProvider"  
 ]  
 }  
 },  
 "minimum-stability": "stable",  
 "prefer-stable": true,  
 "dist": {  
 "url": "https://nexus.flagtickgroup.com/repository/composer-packages/flagtickgroup-datahandling-1.0.0.zip",  
 "type": "zip"  
 }  
 }  
 }  
 }  
}

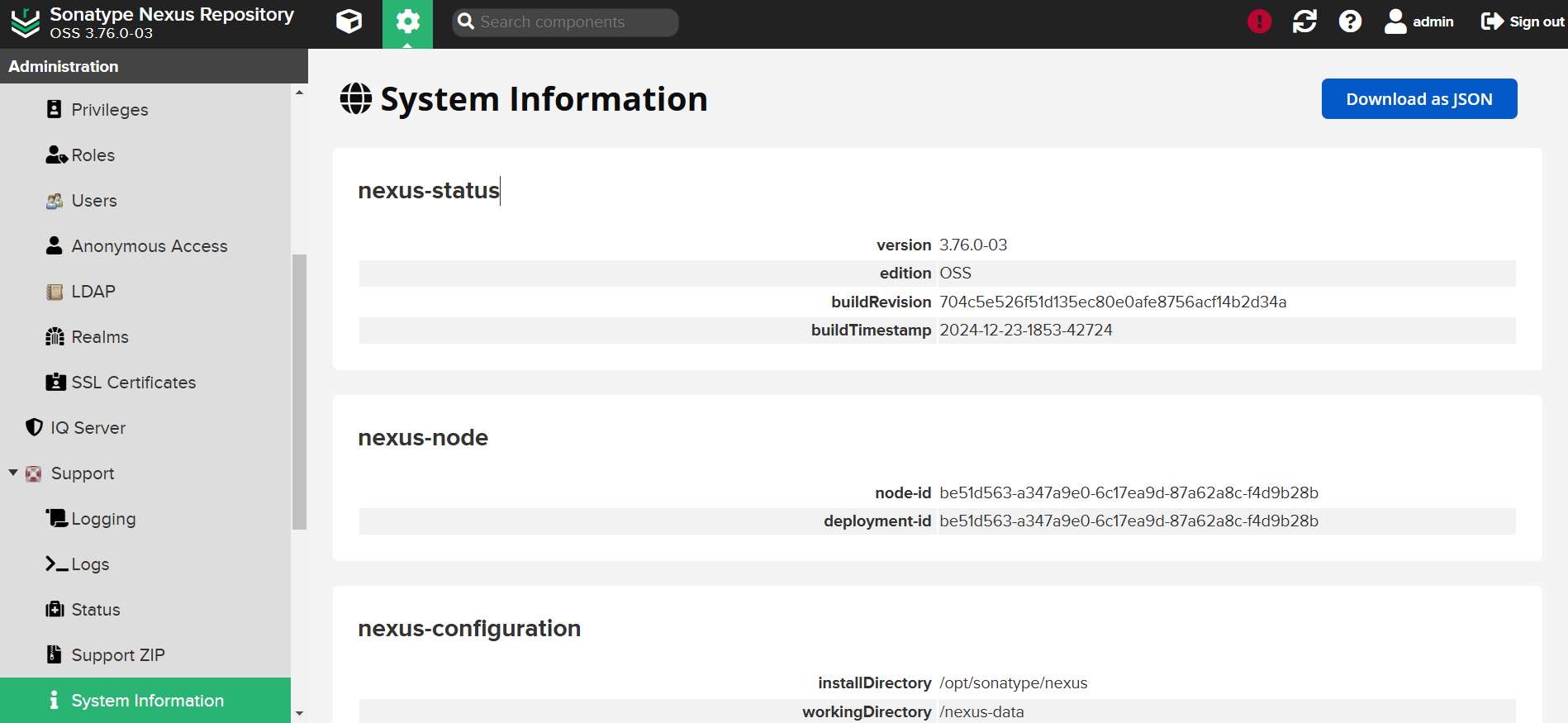
Below is Postman collection to test the Nexus on the remote server, ensuring it works correctly with the configurations provided. It also includes setup for packages such as flagtickgroup-datahandling-1.0.0.zip and packages.json.

{  
 "info": {  
 "\_postman\_id": "ae92fbdc-b6ae-46a4-ad68-7d9ee36d5f4d",  
 "name": "Nexus",  
 "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json",  
 "\_exporter\_id": "30348700"  
 },  
 "item": [  
 {  
 "name": "Get Packages List",  
 "request": {  
 "method": "GET",  
 "header": [  
 {  
 "key": "Authorization",  
 "value": "Basic {{base64Auth}}",  
 "type": "text"  
 }  
 ],  
 "url": {  
 "raw": "{{base\_url}}/packages.json",  
 "host": [  
 "{{base\_url}}"  
 ],  
 "path": [  
 "packages.json"  
 ]  
 }  
 },  
 "response": []  
 }  
 ],  
 "event": [  
 {  
 "listen": "prerequest",  
 "script": {  
 "type": "text/javascript",  
 "packages": {},  
 "exec": [  
 ""  
 ]  
 }  
 },  
 {  
 "listen": "test",  
 "script": {  
 "type": "text/javascript",  
 "packages": {},  
 "exec": [  
 ""  
 ]  
 }  
 }  
 ],  
 "variable": [  
 {  
 "key": "base\_url",  
 "value": "https://nexus.flagtickgroup.com/repository/composer-packages"  
 },  
 {  
 "key": "base64Auth",  
 "value": "BASE64\_ENCODED\_CREDENTIALS"  
 }  
 ]  
}

**Note:** BASE64\_ENCODED\_CREDENTIALS is a Base64-encoded string in the format <username>:<password> used for logging into the Nexus repository on the remote server.

Using the command configures Composer to point to your Nexus repository (https://local-nexus.flagtickgroup.com/repository/composer-packages/) as a custom package source within a containerized environment.

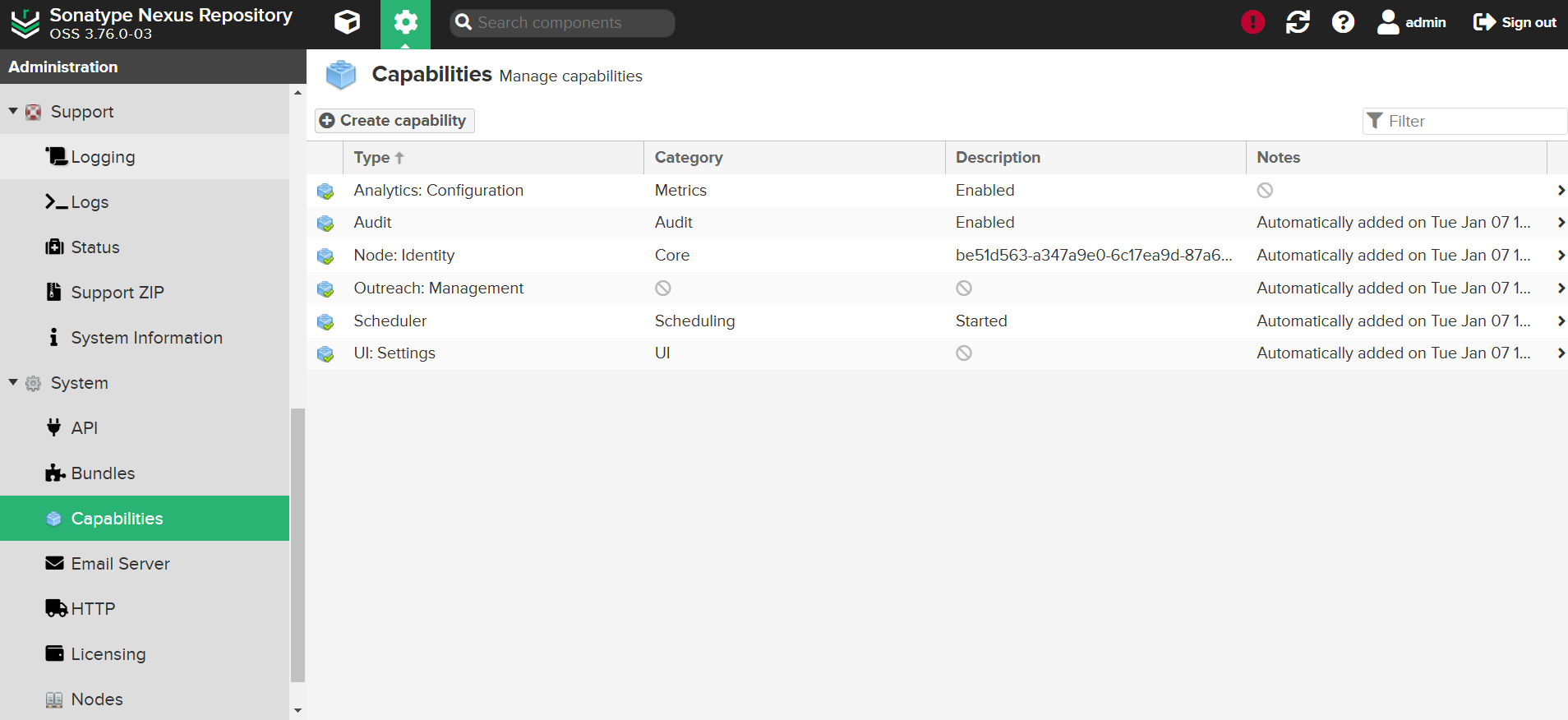
**Bonus:** We need to determine whether the Nexus Repository Manager supports Composer repositories. If it does, the Composer format plugin must be installed and enabled to use Nexus for managing packages in PHP projects.



Let install nexus-repository-composer plugin version 0.1+ requires Nexus Repository Manager 3.71.0 or newer.

- ./resources/plugins:/nexus-data/deploy

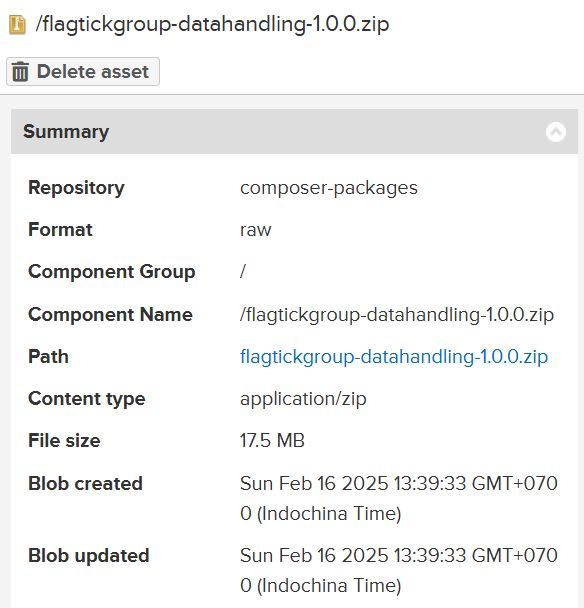
Once the plugin installation and container restart, Nexus automatically detects and loads the .kar file from the /nexus-data/deploy directory, allowing you to verify the installation.



**Note:** Normally, you don’t need to install the Composer plugin for Nexus; instead, you can use the raw self-hosted options as an alternative.

1. Publish Packages to the Raw Repository

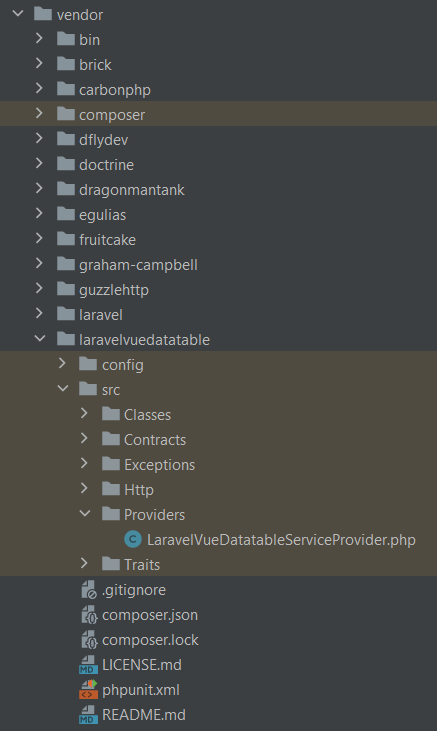
Manually upload .zip package files to the raw repository in Nexus through the Nexus UI, as shown below. Composer cannot directly publish to raw repositories.



and set the Component attributes (e.g., vendor) to match your package structure. Then, use command like: composer require flagtickgroup/datahandling:1.0.0 to include the package in your Laravel project.

1. Clone Package for Self-Hosting in Laravel Application  
     
   For example, go to GitHub or the vendor directory of the legacy Laravel application where the package (e.g., jamesdordoy/laravelvuedatatable) is currently used.

Copy the package and place it into the vendor directory of the new Laravel project where it will be used. After that, update the namespace to use flagtickgroup as the private author prefix.



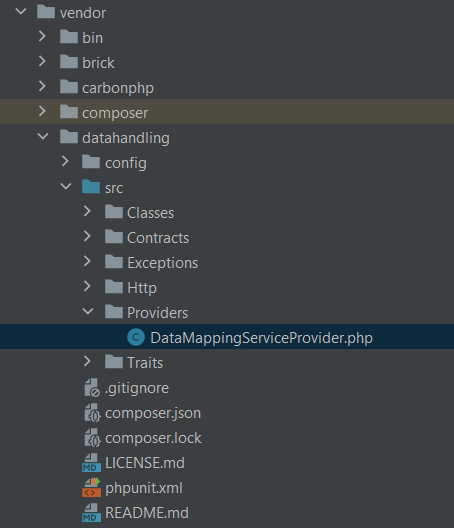
Let us check the composer.json file to see the namespace defined in the autoload section under PSR-4.

"autoload": {  
 "psr-4": {  
 "JamesDordoy\\LaravelVueDatatable\\": "src/"  
 },  
 "classmap": [  
 "src/"  
 ]  
},

Update composer.json in the Laravel project to Include this Package.

"autoload": {  
 "psr-4": {  
 "App\\": "app/",  
 "JamesDordoy\\LaravelVueDatatable\\": "vendor/laravelvuedatatable/src/"  
 }  
},

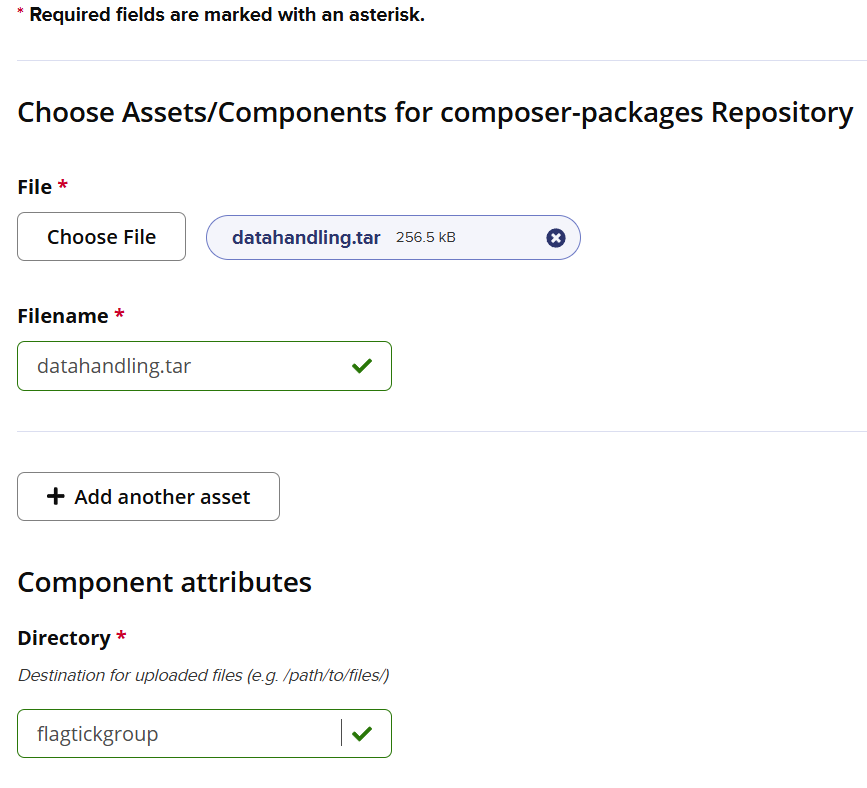
Now, customize it individually to follow the standard package of the Flagtick Group organization.

****

After configuring PSR-4 autoloading in the root composer.json of a Laravel application, running these commands serves the following purposes:

composer dump-autoload  
php artisan vendor:publish --provider="FlagtickGroup\DataHandling\DataHandlingServiceProvider"

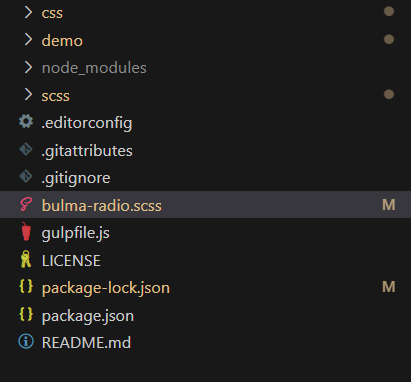
Finally, wrap it up and send it to nexus.flagtickgroup.com to store your package for use in multiple projects.

Refer to resources/nexus/packages.json and update it to include the newly uploaded flagtickgroup/flagtickgroup-datahandling-1.0.0.zip.

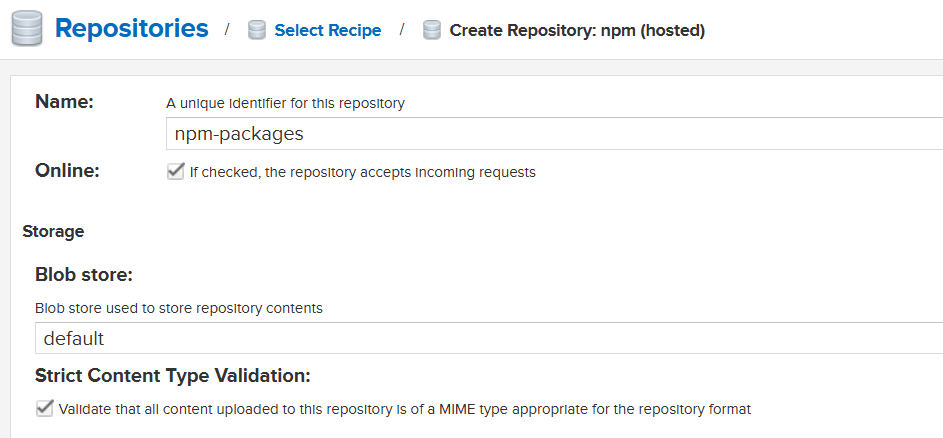
**Note:** Refer to the file resources/nexus/package/README.md.

1. Nexus Repository for Self-Hosted NPM Packages

Assume we have node\_module that is publicly shared on GitHub, but we want to customize it and store it in a private Nexus repository. We can follow the steps below:



Log in to Nexus, navigate to Repositories > Create Repository, and create a new repository named npm-packages, configured as an NPM (hosted) repository for NPM packages.



Update the package.json to include Nexus-specific configuration:

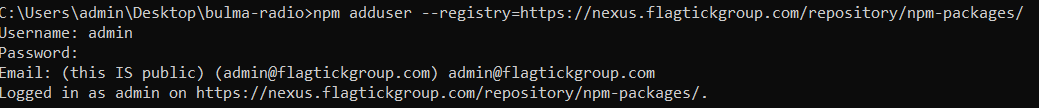
"publishConfig": {  
 "registry": "https://nexus.flagtickgroup.com/repository/npm-packages/"  
},

Log in to your Nexus repository for NPM authentication and configure Nexus as your NPM registry by running the following command:

npm adduser --registry=https://nexus.flagtickgroup.com/repository/npm-packages/

Let us provide the following information:

* + Username: Your Nexus username
  + Password: Your Nexus password
  + Email: Your email address



If you don’t use the method above, update your .npmrc file (either global or project-specific) with the appropriate credentials for Nexus authentication.

registry=https://nexus.flagtickgroup.com/repository/npm-packages/  
//nexus.flagtickgroup.com/repository/npm-packages/:username=admin  
//nexus.flagtickgroup.com/repository/npm-packages/:password=cJdbDSxdmocgdVH

And then verify your authentication with a specific npm registry.

npm whoami --registry=https://nexus.flagtickgroup.com/repository/npm-packages/

Publish your package to a specific npm registry, in this case, a Nexus repository hosted at <https://nexus.flagtickgroup.com/repository/npm-packages/>

"scripts": {  
 "build": "gulp",  
 "deploy": "gh-pages -d demo",  
 "publish": "npm publish --registry=https://nexus.flagtickgroup.com/repository/npm-packages/"  
},

**Note:** Run npm run publish to push your custom Node.js module package to the Nexus remote repository.

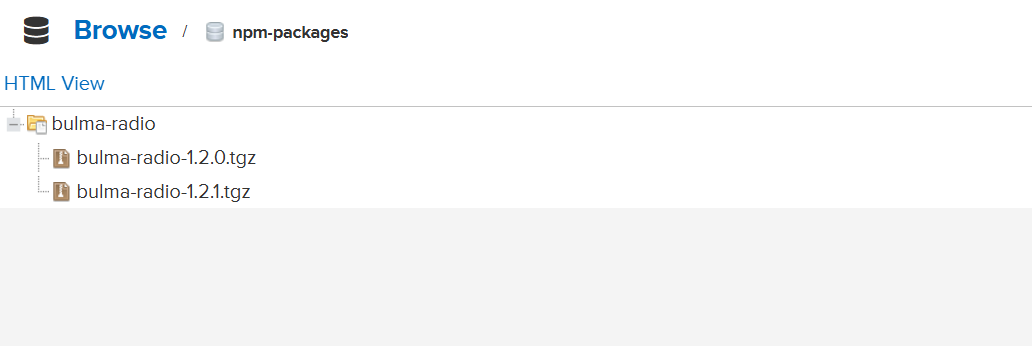
Leverage nvm list and nvm use to switch between Node.js versions on the local machine, which can help ensure you're using the latest **npm/Node.js** version to run npm run publish.

nvm list  
  
 \* 18.0.0 (Currently using 64-bit executable)  
 14.18.0

Check if the package is successfully published, you can use the npm view command:

npm view bulma-radio --registry=https://nexus.flagtickgroup.com/repository/npm-packages/

Once you are trying to publish the same version (1.2.0), it will not update the package because NPM doesn't allow publishing the same version number. Make sure to update the version in your package.json (e.g., 1.2.1, 1.3.0, etc.) before running npm run publish.



**Note:** Use Node version 14 to build and push the package. You can use Node version 18 for the project itself.

* + **Packages**: Use nvm list to check Node.js versions, then run nvm use 14 to switch to version 14 for npm run build and npm run publish.

nvm use 14   
npm run build  
npm run publish

* + **Projects**: Switch to Node.js version 18 and pull the latest versions of node\_modules packages.

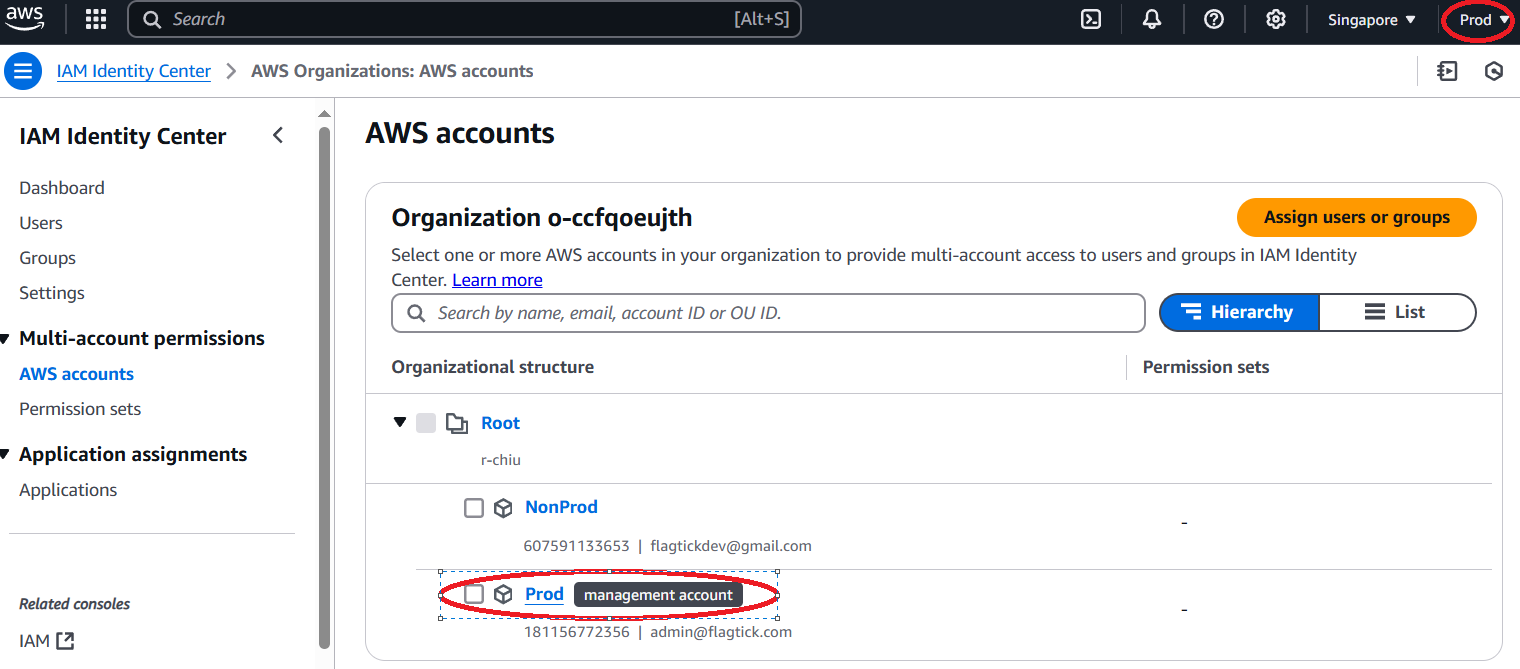
nvm use 18  
npm install bulma-radio --registry=https://nexus.flagtickgroup.com/repository/npm-packages/  
npm run build

Go….

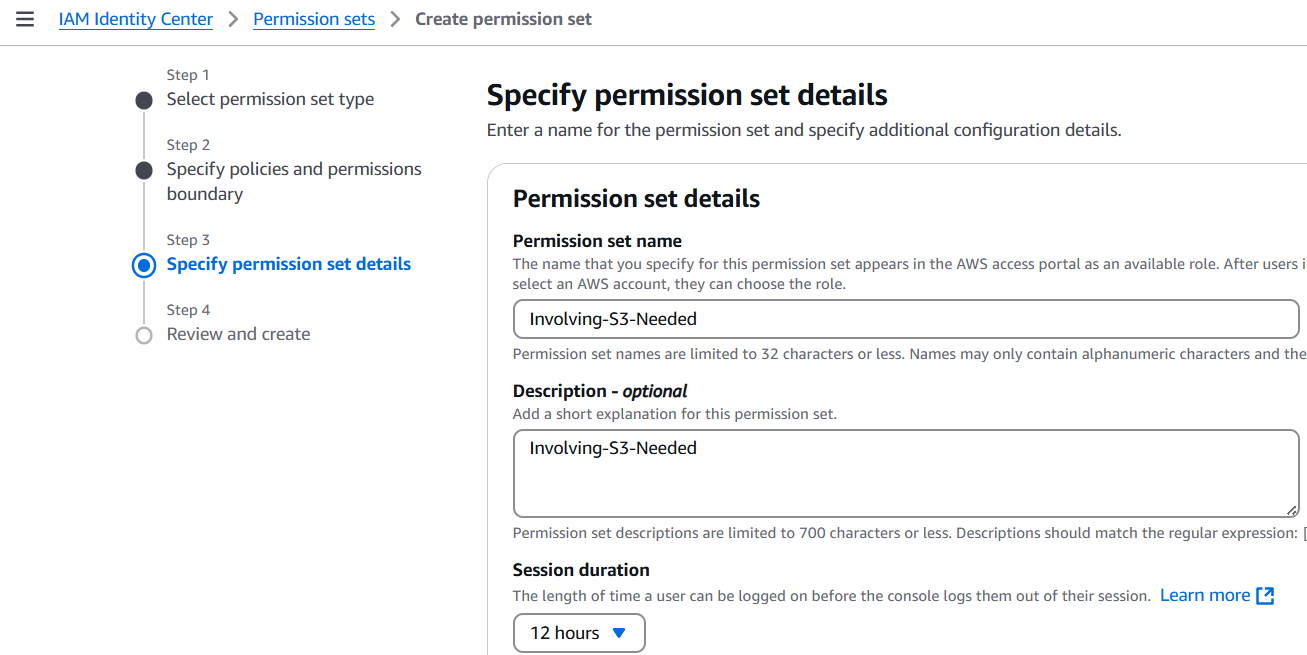
# **S3 Bucket Migration Process**

Set up an IAM user with credentials (AWS Access Key, AWS Secret Key, and AWS User Session, if MFA is enabled) using IAM Identity Center and Permission Sets. Then, use Terraform to migrate resources from a legacy AWS account to a new AWS account.

1. Create a temporary user account (e.g., Vuong Nguyen / vuongluis@gmail.com) in IAM Identity Center, using the management account as the AWS root account.



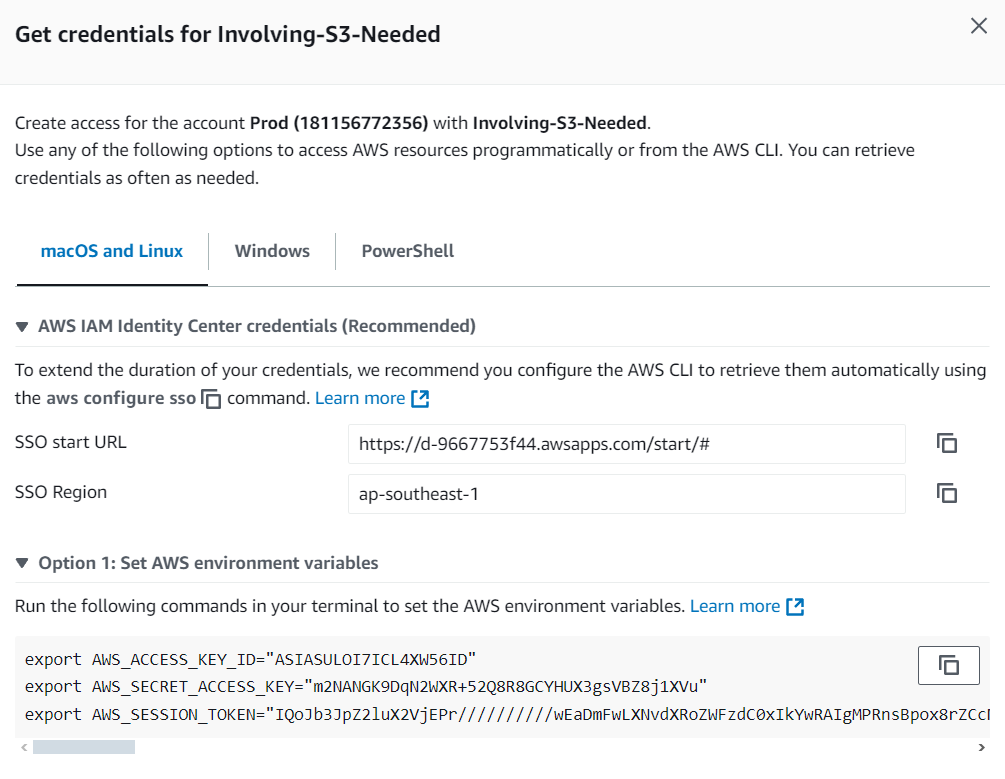
1. Navigate to the Permission Set and set up the Involving-S3-Needed permission for IAM user access via Terraform to download from the S3 bucket or perform other manipulations.



1. Modify the inline policies in JSON format to grant the required permissions for downloading from the S3 bucket.

{  
 "Version": "2012-10-17",  
 "Statement": [  
 {  
 "Effect": "Allow",  
 "Action": [  
 "sts:AssumeRole",  
 "iam:CreateAccessKey",  
 "iam:CreateUser",  
 "iam:CreatePolicy",  
 "iam:GetPolicy",  
 "iam:DeletePolicy",  
 "iam:DeletePolicyVersion",  
 "iam:GetPolicyVersion",  
 "iam:CreatePolicyVersion",  
 "iam:PutRolePolicy",  
 "iam:ListPolicyVersions",  
 "iam:AttachUserPolicy",  
 "iam:TagUser",  
 "iam:CreateRole"  
 ],  
 "Resource": "\*"  
 },  
 {  
 "Effect": "Allow",  
 "Action": [  
 "s3:ListBucket",  
 "s3:GetObject"  
 ],  
 "Resource": [  
 "arn:aws:s3:::flagtick",  
 "arn:aws:s3:::flagtick/\*"  
 ]  
 }  
 ]  
}

1. Login with temporary user in IAM Identity Center, obtain the access key details (AWS Access Key ID, AWS Secret Access Key, and AWS Session Token), and then configure the main.tf file as shown below to set up Terraform for downloading from the S3 bucket.



Copy and paste the contents into the flagtickgroup.suite.vc/.env file to prepare for running Terraform using Docker.

# AWS  
AWS\_ACCESS\_KEY\_ID=<Your AWS access key id>  
AWS\_SECRET\_ACCESS\_KEY=<Your AWS secret access key>   
AWS\_SESSION\_TOKEN=<Your AWS session token>   
AWS\_DEFAULT\_REGION=ap-southeast-1

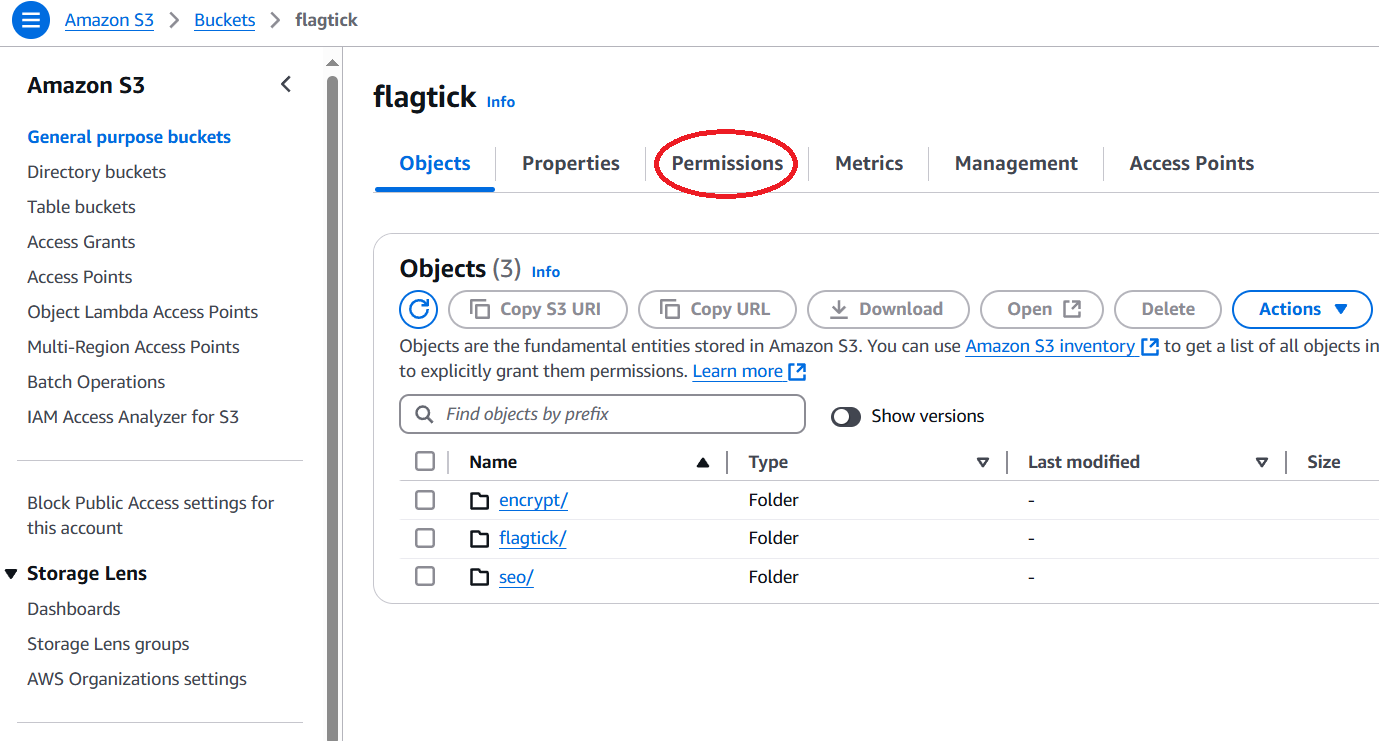
Install the AWS CLI on your local machine and configure it using the credentials (AWS Access Key ID, AWS Secret Access Key, and Session Token) obtained from the temporary user account.

aws configure

**Note:** Use aws configure to input credentials and override the previous ones.

Reference AWS CLI <https://aws.amazon.com/cli/>

1. Identify the bucket from which all objects need to be retrieved, then configure the necessary permissions for that bucket.

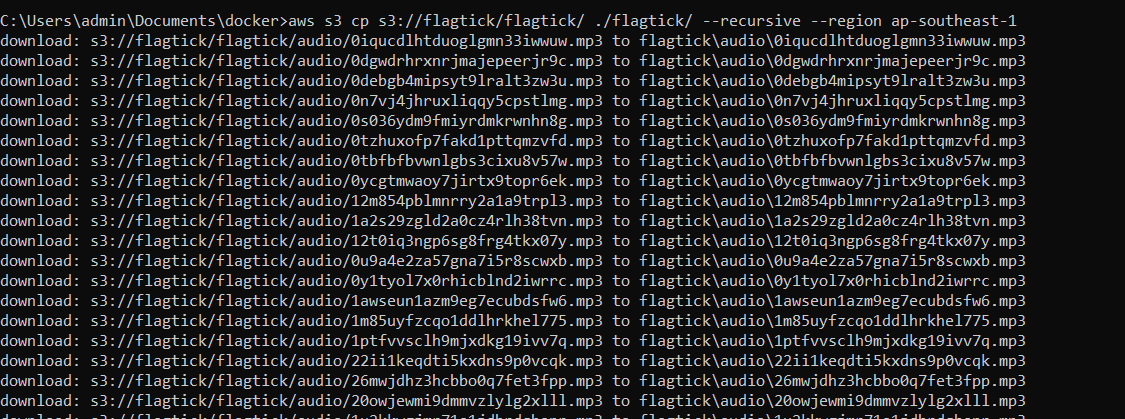


Next, update the bucket policy with the following JSON to grant all required permissions.

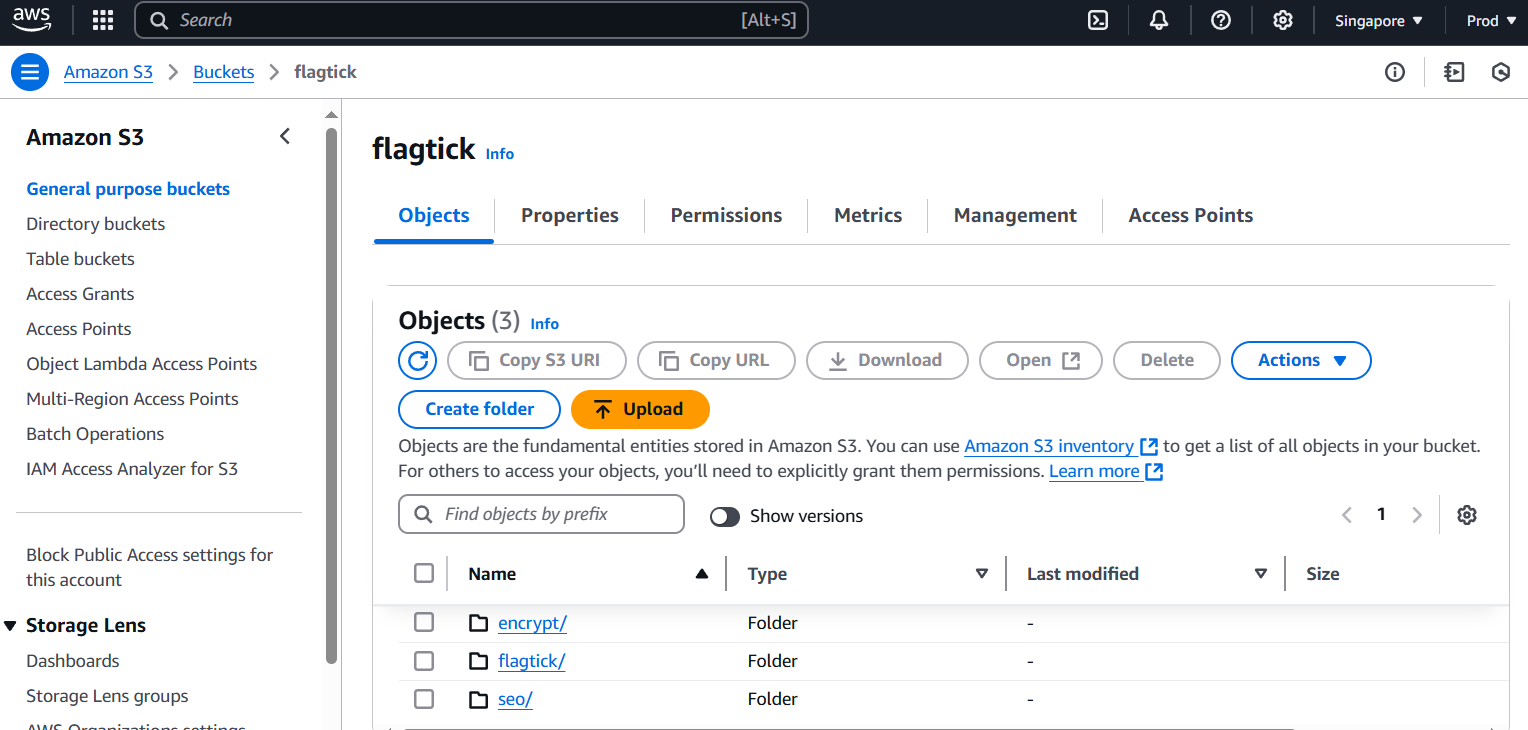
{  
 "Version": "2012-10-17",  
 "Statement": [  
 {  
 "Effect": "Allow",  
 "Principal": "\*",  
 "Action": [  
 "s3:PutObject",  
 "s3:PutObjectAcl",  
 "s3:GetObject",  
 "s3:GetObjectAcl",  
 "s3:DeleteObject",  
 "s3:ListBucket",  
 "s3:ListBucketVersions"  
 ],  
 "Resource": [  
 "arn:aws:s3:::flagtick",  
 "arn:aws:s3:::flagtick/\*"  
 ]  
 }  
 ]  
}

1. Open the command line, navigate to the desired folder, and use the AWS CLI to download the specified folder from the bucket. For example:

aws s3 cp s3://flagtick/flagtick/ ./flagtick/ --recursive --region ap-southeast-1

In Batch mode  


In the S3 bucket, it appears as follows:



# **Managing AWS with Terraform**

Terraform is powerful tool for infrastructure as code (IaC). To set up Terraform project, you typically configure the Terraform environment in docker-compose.yml file for streamlined management and execution within a containerized setup.

1. Terraform with Docker Compose

terraform:  
 image: hashicorp/terraform:latest  
 container\_name: terraform  
 volumes:  
 - ./resources/terraform:/workspace  
 working\_dir: /workspace  
 environment:  
 - AWS\_ACCESS\_KEY\_ID=${AWS\_ACCESS\_KEY\_ID}  
 - AWS\_SECRET\_ACCESS\_KEY=${AWS\_SECRET\_ACCESS\_KEY}  
 - AWS\_SESSION\_TOKEN=${AWS\_SESSION\_TOKEN}  
 - AWS\_DEFAULT\_REGION=${AWS\_DEFAULT\_REGION}  
 entrypoint: [ "terraform" ]  
 command: ["init"]

1. Establishing Configuration for AWS Authentication

Leverage IAM user, generate credentials, optionally configure MFA, and assign permissions using IAM Identity Center, and store the information securely in the flagtickgroup.suite.vc/.env file.

# AWS  
AWS\_ACCESS\_KEY\_ID=<Access Key ID>  
AWS\_SECRET\_ACCESS\_KEY=<Secret Access Key>  
AWS\_SESSION\_TOKEN= <Session Token>   
AWS\_DEFAULT\_REGION=<Region>

This well-organized Terraform project typically follows this folder structure:

/terraform-project  
├── main.tf   
├── outputs.tf  
├── terraform.tfvars  
├── variables.tf   
└── README.md

1. Modularize Terraform: IAM, Lambda, API Gateway

Refactor the main.tf file by splitting its structure into separate modules for IAM, Lambda, and API Gateway. After organizing the modules, use the command docker-compose run --rm terraform init to initialize the Terraform configuration and download the required modules and providers.

│ main.tf  
│ outputs.tf  
│ PRACTICE.md  
│ README.md  
│ variables.tf  
│  
├───aws  
│ apigateway.tf  
│ cognito.tf  
│  
├───lambda  
│ ├───admin\_new\_cognito\_user  
│ │ index.js  
│ │ lambda.zip  
│ │ tes.json  
│ │  
│ ├───admin\_update\_cognito\_user  
│ │ index.js   
│ │ lambda.zip  
│ │  
│ ├───admin\_update\_role\_right  
│ │ index.js  
│ │ lambda.zip  
│ │   
│ ├───flagtick\_group\_portal\_newpass  
│ │ index.js  
│ │ lambda.zip  
│ │  
│ ├───flagtick\_group\_portal\_refresh  
│ │ index.js  
│ │ lambda.zip  
│ │  
│ ├───flagtick\_group\_portal\_signin  
│ │ index.js  
│ │ lambda.zip  
│  
├───iam  
│ main.tf  
│ outputs.tf  
│ variables.tf  
│  
└───lambda  
 main.tf  
 outputs.tf  
 variables.tf

Initialize Terraform working directory using Dockerized setup.

docker-compose run --rm terraform init

We will create modules folder and set up each service, such as Cognito, API Gateway, Lambda, etc. Then, we will use terraform init to verify the modules.

modules   
├───api\_gateway   
│ main.tf   
│ outputs.tf   
│ variables.tf   
│   
├───cognito   
│ main.tf   
│ outputs.tf   
│ variables.tf   
│   
├───iam   
│ main.tf   
│ outputs.tf   
│ variables.tf   
│   
└───lambda   
 main.tf   
 outputs.tf   
 variables.tf

As you can see here, we will use docker-compose run --rm terraform plan to generate an execution plan that shows the actions Terraform will take to create or modify resources in your infrastructure.

docker-compose run --rm terraform plan

**Note:** Remember to log in using the Terraform-user (IAM User SSO) to obtain the AWS access key ID, access key, and session token, which are based on MFA activation.

1. Setting up the Terraform Cognito module

Here is an example for main.tf to create cognito through terraform in AWS Account.

resource "aws\_cognito\_user\_pool" "flagtick\_group\_pool" {  
 name = var.cognito\_user\_pool\_name  
  
 username\_attributes = ["email"]  
 auto\_verified\_attributes = ["email"]  
  
 schema {  
 name = "email"  
 attribute\_data\_type = "String"  
 required = true  
 mutable = true  
 }  
  
 schema {  
 name = "given\_name"  
 attribute\_data\_type = "String"  
 required = false  
 mutable = true  
 }  
  
 schema {  
 name = "permissions"  
 attribute\_data\_type = "String"  
 required = false  
 mutable = true  
 string\_attribute\_constraints {  
 min\_length = 0  
 max\_length = 2048  
 }  
 }  
  
 schema {  
 name = "roles"  
 attribute\_data\_type = "String"  
 required = false  
 mutable = true  
 string\_attribute\_constraints {  
 min\_length = 0  
 max\_length = 2048  
 }  
 }  
  
 schema {  
 name = "userID"  
 attribute\_data\_type = "String"  
 required = false  
 mutable = true  
 string\_attribute\_constraints {  
 min\_length = 0  
 max\_length = 11  
 }  
 }  
  
 schema {  
 name = "userName"  
 attribute\_data\_type = "String"  
 required = false  
 mutable = true  
 string\_attribute\_constraints {  
 min\_length = 0  
 max\_length = 255  
 }  
 }  
  
 admin\_create\_user\_config {  
 allow\_admin\_create\_user\_only = false  
 }  
}  
  
resource "aws\_cognito\_user\_pool\_client" "flagtick\_group\_pool\_client" {  
 name = "FlagtickGroupApp"  
 user\_pool\_id = aws\_cognito\_user\_pool.flagtick\_group\_pool.id  
  
 generate\_secret = false  
  
 explicit\_auth\_flows = [  
 "ALLOW\_USER\_SRP\_AUTH",  
 "ALLOW\_REFRESH\_TOKEN\_AUTH"  
 ]  
  
 access\_token\_validity = 120  
 id\_token\_validity = 120  
 refresh\_token\_validity = 30  
  
 token\_validity\_units {  
 access\_token = "minutes"  
 id\_token = "minutes"  
 refresh\_token = "days"  
 }  
  
 prevent\_user\_existence\_errors = "ENABLED"  
}

var.cognito\_user\_pool\_name is used to reference variables defined in variables.tf in Terraform. It allows you to define reusable variables that you can easily use throughout your Terraform configuration files (like main.tf).

variable "aws\_region" {  
 description = "AWS region"  
 type = string  
}  
  
variable "cognito\_user\_pool\_name" {  
 description = "The name of the Cognito User Pool"  
 type = string  
 default = "FlagtickGroup"  
}

outputs.tf file in Terraform is used to define values that you want to export and make available for use in other Terraform modules or in the configuration's output. It allows you to present useful information after the Terraform execution has completed, such as the names, IDs, or other properties of created resources.

output "user\_pool\_id" {  
 description = "Cognito User Pool ID"  
 value = aws\_cognito\_user\_pool.flagtick\_group\_pool.id  
}  
output "client\_id" {  
 description = "Client ID"  
 value = aws\_cognito\_user\_pool\_client.flagtick\_group\_pool\_client.id  
}  
output "cognito\_arn" {  
 description = "Cognito User Pool ARN"  
 value = aws\_cognito\_user\_pool.flagtick\_group\_pool.arn  
}

The parent main.tf coordinates and integrates the different modules as part of your Terraform infrastructure setup.

provider "aws" {  
 region = "ap-southeast-1"  
}  
  
module "cognito" {  
 source = "./modules/cognito"  
 aws\_region = var.aws\_region  
 cognito\_user\_pool\_name = var.cognito\_user\_pool\_name  
}

There are two approaches to creating AWS Cognito, categorized into the following capabilities:

User --> IAM User --> AWS Cognito

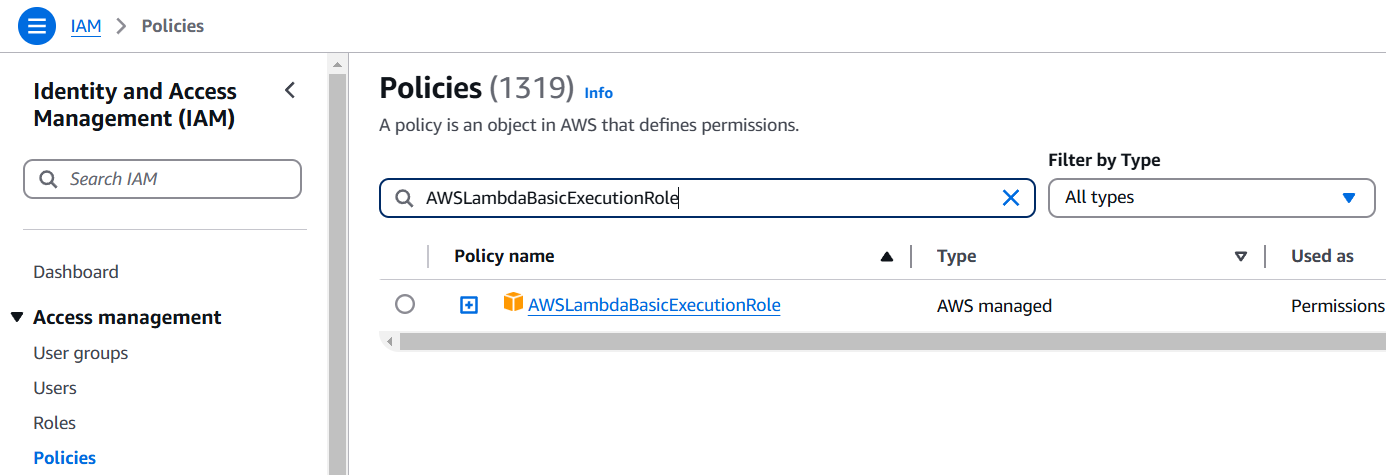
Lambda Function --> IAM Role --> AWS Cognito

1. Setting up the Terraform Cognito module

Let us create the FlagtickServiceRoleForLambda—an assumed role for creating Lambda functions and granting access to interact with other services like Cognito.

resource "aws\_iam\_role" "flagtick\_service\_role\_for\_lambda" {  
 name = "FlagtickServiceRoleForLambda"  
 assume\_role\_policy = <<EOF  
 {  
 "Version": "2012-10-17",  
 "Statement": [  
 {  
 "Effect": "Allow",  
 "Principal": {  
 "Service": "lambda.amazonaws.com"  
 },  
 "Action": "sts:AssumeRole"  
 }  
 ]  
 }  
 EOF  
}  
  
resource "aws\_iam\_policy" "allow\_cognito\_update" {  
 name = "AllowCognitoUpdateUserAttributes"  
 description = "Allows Lambda to manage user attributes in Cognito"  
 policy = jsonencode({  
 Version = "2012-10-17"  
 Statement = [  
 {  
 Effect = "Allow"  
 Action = [  
 "cognito-idp:AdminUpdateUserAttributes",  
 "cognito-idp:AdminCreateUser",   
 "cognito-idp:AdminSetUserPassword"  
 ]  
 Resource = "arn:aws:cognito-idp:${var.aws\_region}:\*:userpool/${var.user\_pool\_id}"  
 }  
 ]  
 })  
}  
  
resource "aws\_iam\_role\_policy\_attachment" "lambda\_cognito\_policy" {  
 role = aws\_iam\_role.flagtick\_service\_role\_for\_lambda.name  
 policy\_arn = aws\_iam\_policy.allow\_cognito\_update.arn  
}

Here, we grant permissions for actions such as adminUpdateUserAttributes, adminCreateUser, and adminSetUserPassword to allow the Lambda function to interact with Cognito.



Furthermore, if you want to enable CloudWatch for the Lambda function, you must use the following script:

resource "aws\_iam\_role\_policy\_attachment" "lambda\_cloudwatch\_logs" {  
 role = aws\_iam\_role.flagtick\_service\_role\_for\_lambda.name  
 policy\_arn = "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess"  
}

1. Credentials for All Services on the Remote Server

# **Remote Server Service Credentials**

We will provide access to specific accounts on the remote server, allowing access to services such as Nexus repositories, SonarQube, and other related services.

1. Nexus service is available at https://nexus.flagtickgroup.com

Username: **admin** / Password: **cJdbDSxdmocgdVH**

# **SSL Setup and Endpoint Configuration**

Go

# Proposed Solution:

The proposed solution is divided into 7 key parts: