Kong

Kong is an open-source API Gateway that acts as a **Middleware** between your clients and your backend services. Here's an easy way to understand it:

What is an API Gateway?

An API Gateway is like a receptionist at a hotel. When you arrive, the receptionist helps you check in, provides you with information, and directs you to your room. Similarly, an API Gateway sits in front of your backend services and manages all incoming requests, ensuring they reach the correct service.

Key Features of Kong API Gateway:

1. **Routing**: Kong directs incoming API requests to the appropriate service. For example, if a request comes in for `/users`, Kong knows to route it to the user service.
2. **Security**: It can handle authentication and authorization, ensuring that only authorized users can access certain services. Think of it as the bouncer at a club checking IDs before letting people in.
3. **Rate Limiting**: Kong can limit the number of requests a user can make in a certain time period, preventing abuse. This is like a store manager ensuring that one customer doesn't buy all the stock at once.
4. **Logging and Monitoring**: Kong can keep track of all the requests and responses, providing insights into how your services are being used. It's like having a security camera that records all activities.
5. **Transformation**: It can modify requests and responses on the fly. For example, if a client needs data in a different format, Kong can transform the response before sending it back.

How Kong Works:

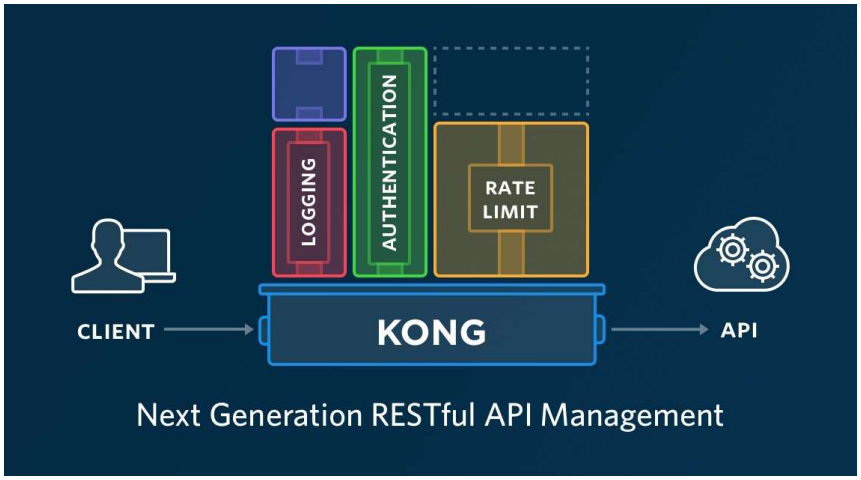
1. **Installation**: Kong is installed and runs as a server.
2. **Configuration**: You configure Kong to know about your backend services (called APIs in Kong).
3. **Plugins**: You can add plugins to Kong to provide extra features like authentication, rate limiting, and logging.
4. **Clients**: When clients (like a mobile app or web app) make requests, they go to Kong first.
5. **Proxying**: Kong forwards these requests to the appropriate backend services, and then it forwards the responses back to the clients.

Example:

Imagine you have an online store with services for users, products, and orders. Without Kong, each client would need to know the details of each service. With Kong, clients only need to interact with Kong, which

handles routing, security, and other concerns, making it easier and more efficient for everyone involved.

In summary, Kong simplifies the management of API traffic, enhances security, and provides additional features to improve the performance and reliability of your services.



Configuring **Kong** and **Konga-UI**

- Create Following `compose.yml` file

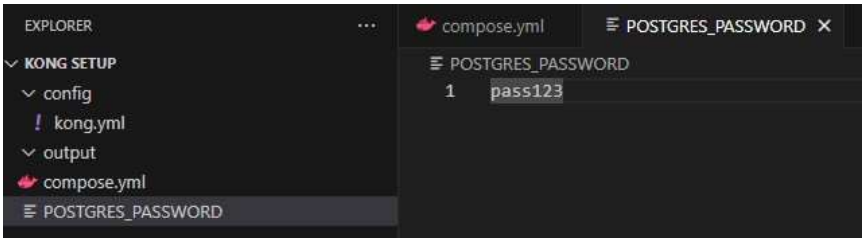
```
version: '3.7'

volumes:
  kong_data: {}
  kong_prefix_vol:
    driver_opts:
     type: tmpfs
     device: tmpfs
  kong_tmp_vol:
    driver_opts:
     type: tmpfs
     device: tmpfs

networks:
  kong-net:
    external: false

services:
  kong-migrations:
    image: "${KONG_DOCKER_TAG:-kong:latest}"
    command: kong migrations bootstrap
```

```
    depends_on:
      - db
    environment:
      KONG_DATABASE: postgres
      KONG_PG_DATABASE: ${KONG_PG_DATABASE:-kong}
      KONG_PG_HOST: db
      KONG_PG_USER: ${KONG_PG_USER:-kong}
      KONG_PG_PASSWORD_FILE: /run/secrets/kong_postgres_password
    secrets:
      - kong_postgres_password
    networks:
      - kong-net
    restart: on-failure
    deploy:
      restart_policy:
        condition: on-failure

  kong-migrations-up:
    image: "${KONG_DOCKER_TAG:-kong:latest}"
    command: kong migrations up && kong migrations finish
    depends_on:
      - db
    environment:
      KONG_DATABASE: postgres
      KONG_PG_DATABASE: ${KONG_PG_DATABASE:-kong}
      KONG_PG_HOST: db
      KONG_PG_USER: ${KONG_PG_USER:-kong}
      KONG_PG_PASSWORD_FILE: /run/secrets/kong_postgres_password
    secrets:
      - kong_postgres_password
    networks:
      - kong-net
    restart: on-failure
    deploy:
      restart_policy:
        condition: on-failure

  kong:
    image: "${KONG_DOCKER_TAG:-kong:latest}"
    user: "${KONG_USER:-kong}"
    depends_on:
      - db
    environment:
      KONG_ADMIN_ACCESS_LOG: /dev/stdout
      KONG_ADMIN_ERROR_LOG: /dev/stderr
      KONG_PROXY_LISTEN: "${KONG_PROXY_LISTEN:-0.0.0.0:8000}"
      KONG_ADMIN_LISTEN: "${KONG_ADMIN_LISTEN:-0.0.0.0:8001}"
      KONG_CASSANDRA_CONTACT_POINTS: db
      KONG_DATABASE: postgres
      KONG_PG_DATABASE: ${KONG_PG_DATABASE:-kong}
      KONG_PG_HOST: db
      KONG_PG_USER: ${KONG_PG_USER:-kong}
      KONG_PROXY_ACCESS_LOG: /dev/stdout
      KONG_PROXY_ERROR_LOG: /dev/stderr
```

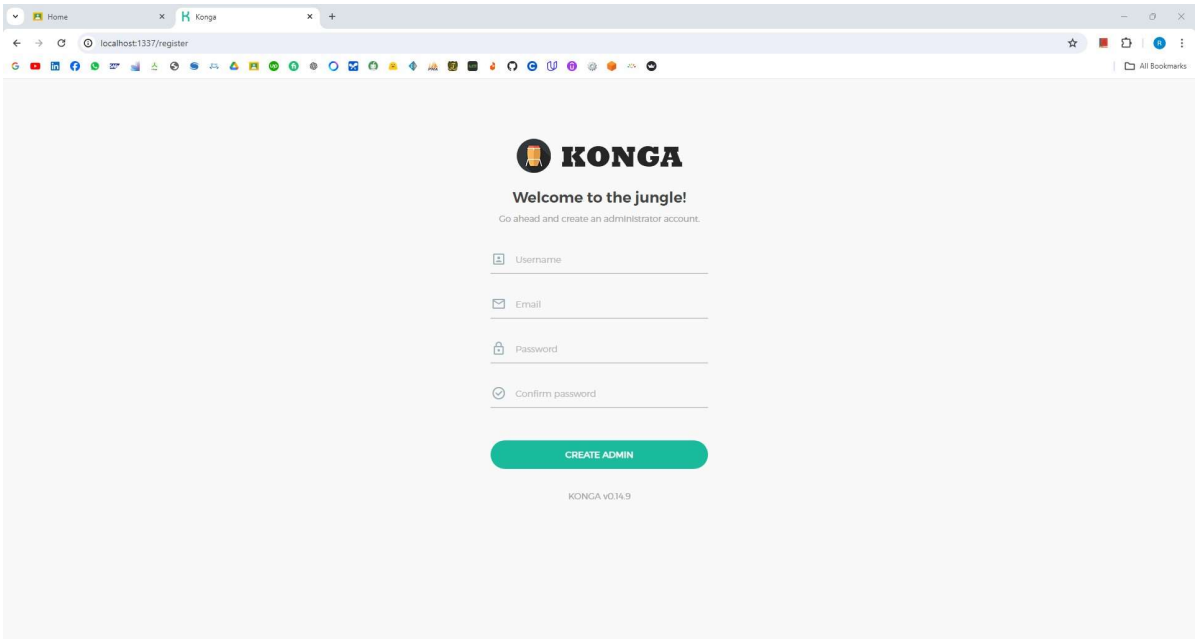
```
      KONG_PG_PASSWORD_FILE: /run/secrets/kong_postgres_password
      KONG_PREFIX: ${KONG_PREFIX:-/var/run/kong}
    secrets:
      - kong_postgres_password
    networks:
      - kong-net
    ports:
      # The following two environment variables default to an insecure value
(0.0.0.0)
      # according to the CIS Security test.
      - "${KONG_INBOUND_PROXY_LISTEN:-0.0.0.0}:8000:8000/tcp"
      - "${KONG_INBOUND_SSL_PROXY_LISTEN:-0.0.0.0}:8443:8443/tcp"
      # Making them mandatory but undefined, like so would be backwards-breaking:
      # - "${KONG_INBOUND_PROXY_LISTEN?Missing inbound proxy host}:8000:8000/tcp"
      # - "${KONG_INBOUND_SSL_PROXY_LISTEN?Missing inbound proxy ssl
host}:8443:8443/tcp"
      # Alternative is deactivating check 5.13 in the security bench, if we
consider Kong's own config to be enough security here

      - "127.0.0.1:8001:8001/tcp"
      - "127.0.0.1:8444:8444/tcp"
    healthcheck:
      test: ["CMD", "kong", "health"]
      interval: 10s
      timeout: 10s
      retries: 10
    restart: on-failure:5
    read_only: true
    volumes:
      - kong_prefix_vol:${KONG_PREFIX:-/var/run/kong}
      - kong_tmp_vol:/tmp
    deploy:
      restart_policy:
        delay: 50s
        condition: on-failure
        max_attempts: 5
        window: 10s
      resources:
        limits:
          cpus: ${KONG_CPU_LIMIT:-2}
          memory: ${KONG_MEMORY_LIMIT:-2g}
    security_opt:
      - no-new-privileges

  db:
    image: postgres:9.5
    environment:
      POSTGRES_DB: ${KONG_PG_DATABASE:-kong}
      POSTGRES_USER: ${KONG_PG_USER:-kong}
      POSTGRES_PASSWORD_FILE: /run/secrets/kong_postgres_password
    secrets:
      - kong_postgres_password
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "${KONG_PG_USER:-kong}"]
```

```
      interval: 30s
      timeout: 30s
      retries: 3
    restart: on-failure
    deploy:
      restart_policy:
        condition: on-failure
    stdin_open: true
    tty: true
    networks:
      - kong-net
    volumes:
      - kong_data:/var/lib/postgresql/data


  konga-prepare:
    image: pantsel/konga:latest
    command: "-c prepare -a postgres -u postgresql://kong:pass123@db:5432/konga"
    networks:
      - kong-net
    restart: on-failure
    secrets:
      - kong_postgres_password
    depends_on:
      - db
    volumes:
      - kong_data:/var/lib/postgresql/data

########################################
  # Konga: Kong GUI
########################################

  konga:
    image: pantsel/konga:latest
    restart: always
    networks:
        - kong-net
    environment:
      DB_ADAPTER: postgres
      DB_URI: postgresql://kong:pass123@db:5432/konga
      NODE_ENV: production
    depends_on:
      - db
    ports:
      - "1337:1337"

secrets:
  kong_postgres_password:
    file: ./POSTGRES_PASSWORD
```

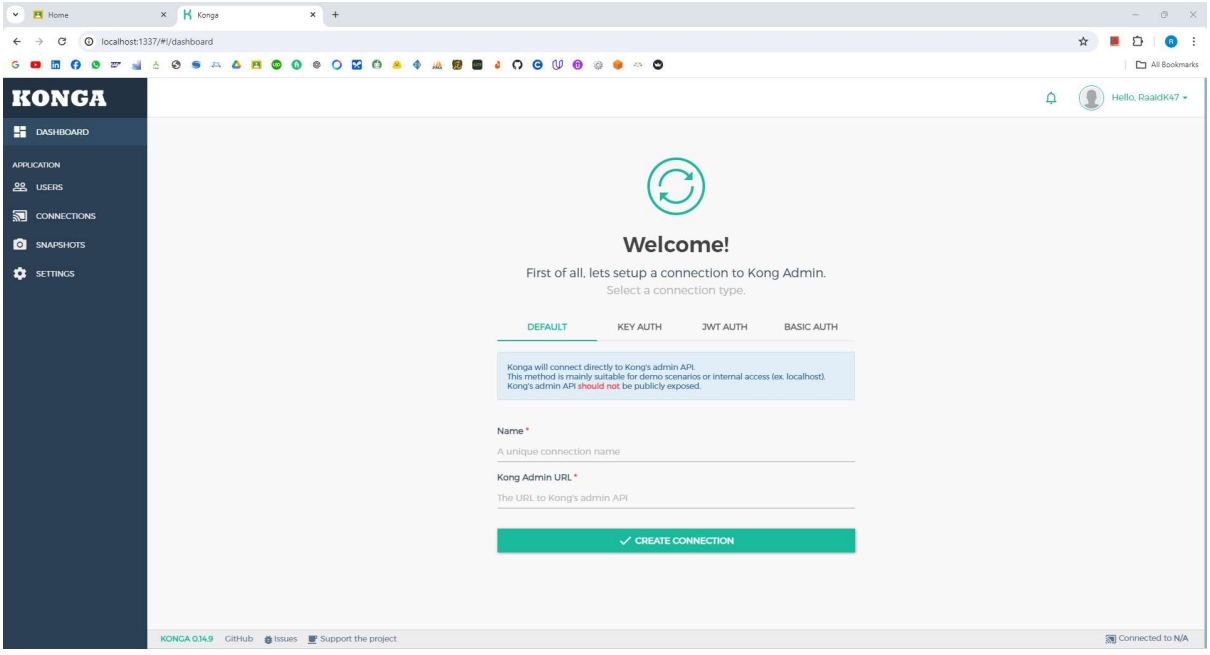
- Create a `POSTGRES_PASSWORD` file and place Password in it

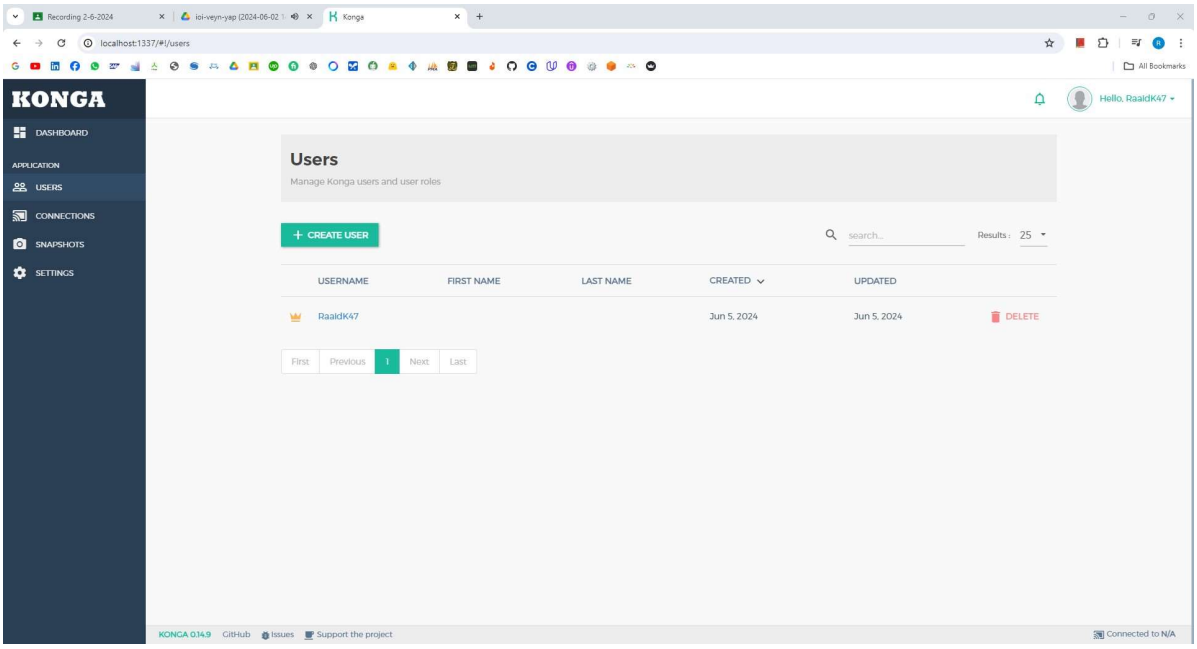


- Run `docker compose up` command to start **Kong** and **Konga-UI**
- Go to `http://localhost:1337/` to access UI



- Register with *Username* and *Password* for an Admin Account.
- Log-In to Admin Account to access the UI.





PyCaret

PyCaret is an open-source, low-code machine learning library in Python that simplifies the process of building, deploying, and maintaining machine learning models. Here's a straightforward explanation:

Key Features of PyCaret:

1. **Low-Code**: It requires very little coding to create machine learning models. This makes it accessible even for those who are not experts in programming or machine learning.
2. **Automates Processes**: PyCaret automates many tasks that would typically require multiple lines of code, such as data preprocessing, model training, hyperparameter tuning, and model evaluation.
3. **End-to-End Machine Learning**: PyCaret covers the entire machine learning lifecycle from data preparation to model deployment, all within a single, cohesive framework.

How It Works:

1. **Data Preparation**: PyCaret helps clean and prepare data for modeling. It can handle missing values, encode categorical variables, and scale numerical data with just a few lines of code.
2. **Model Training**: You can quickly train and compare multiple machine learning models using PyCaret's simple and intuitive functions. It supports a wide range of algorithms out-of-the-box.
3. **Model Evaluation**: PyCaret provides easy-to-understand metrics and visualizations to evaluate model performance, allowing you to select the best model for your needs.
4. **Hyperparameter Tuning**: It offers automated hyperparameter tuning to optimize model performance without manual intervention.
5. **Model Deployment**: PyCaret simplifies the process of deploying machine learning models into production, making it easier to integrate them into applications.

Example:

Here’s a basic example of how you might use PyCaret:

```
# Importing the regression module
from pycaret.regression import *

# Loading a dataset
data = get_data('insurance')

# Setting up the environment in PyCaret
reg = setup(data, target='charges')

# Comparing different models
best_model = compare_models()

# Finalizing the best model
final_model = finalize_model(best_model)

# Making predictions on new data
predictions = predict_model(final_model, data=new_data)
```

In this example, PyCaret helps load the data, set up the machine learning environment, compare different models to find the best one, finalize the model, and make predictions—all with just a few lines of code.

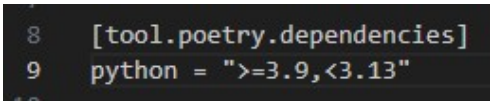
Benefits:

- **User-Friendly**: Ideal for beginners and non-experts in machine learning.
- **Efficiency**: Saves time by automating repetitive tasks.
- **Versatility**: Can be used for a variety of machine learning tasks, including classification, regression, clustering, and anomaly detection.

Overall, PyCaret is a powerful tool that democratizes machine learning by making it more accessible and less time-consuming.

Creating a PyCaret Project with Poetry

- Create a New `Poetry` Project
- Change Directory to Poetry Project Folder
- Open `Poetry Shell`
    - If a `.venv` folder is created, you can delete this folder.
    - OR - You make sure that this folder is not copied into Docker Container
- Set the `python` property to `">=3.9,<3.13"` in `.toml` file.



- Install PyCaret with command `poetry add pycaret`

```
PS E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L13-14 - Kong\Code\PyCaret\pycaret_project> poetry add pycaret
Using version ^3.3.2 for pycaret

Updating dependencies
Resolving dependencies... Downloading https://files.pythonhosted.org/packages/33/1a/1142c3d27dd2134157f9d6cf1fed5a566b2f
Resolving dependencies... Downloading https://files.pythonhosted.org/packages/33/1a/1142c3d27dd2134157f9d6cf1fed5a566b2f
Resolving dependencies... Downloading https://files.pythonhosted.org/packages/33/1a/1142c3d27dd2134157f9d6cf1fed5a566b2f
Resolving dependencies... (40.2s)

Package operations: 95 installs, 1 update, 0 removals

  - Installing six (1.16.0)
  - Installing attrs (23.2.0)
  - Installing colorama (0.4.6)
  - Installing markupsafe (2.1.5)
  - Installing numpy (1.26.4)
  - Installing python-dateutil (2.9.0.post0)
  - Installing pytz (2024.1)
  - Installing rpds-py (0.18.1)
  - Installing tzdata (2024.1)
  - Installing asttokens (2.4.1)
  - Installing joblib (1.3.2)
  - Installing certifi (2024.6.2)
  - Installing charset-normalizer (3.3.2)
```

- You may get following error.

```
RuntimeError

Unable to find installation candidates for kaleido (0.2.1.post1)

at ~\pipx\venvs\poetry\Lib\site-packages\poetry\installation\chooser.py:74 in choose_for
     70
     71              links.append(link)
     72
     73          if not links:
  →  74              raise RuntimeError(f"Unable to find installation candidates for {package}")
     75
     76          # Get the best link
     77          chosen = max(links, key=lambda link: self._sort_key(package, link))
     78

Cannot install kaleido.
```

- Run command `poetry add kaleido==0.2.1` to Install required Dependency. It may take some time.

```
PS E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L13-14 - Kong\Code\PyCaret\pycaret_project> poetry add kaleido==0.2.1

Updating dependencies
Resolving dependencies... (0.7s)

Package operations: 1 install, 0 updates, 0 removals

  - Installing kaleido (0.2.1)

Writing lock file
```

- Run command `poetry add pycaret` again.

```
PS E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L13-14 - Kong\Code\PyCaret\pycaret_project> poetry add pycaret
Using version ^3.3.2 for pycaret

Updating dependencies
Resolving dependencies... (8.2s)

Package operations: 1 install, 0 updates, 0 removals

  - Installing pycaret (3.3.2)

Writing lock file
```

- This will successfully install PyCaret.

```
[tool.poetry.dependencies]
python = ">=3.9,<3.13"
kaleido = "0.2.1"
pycaret = "^3.3.2"
```

- You can now write Python Code in Poetry Environment by Importing PyCaret Modules.