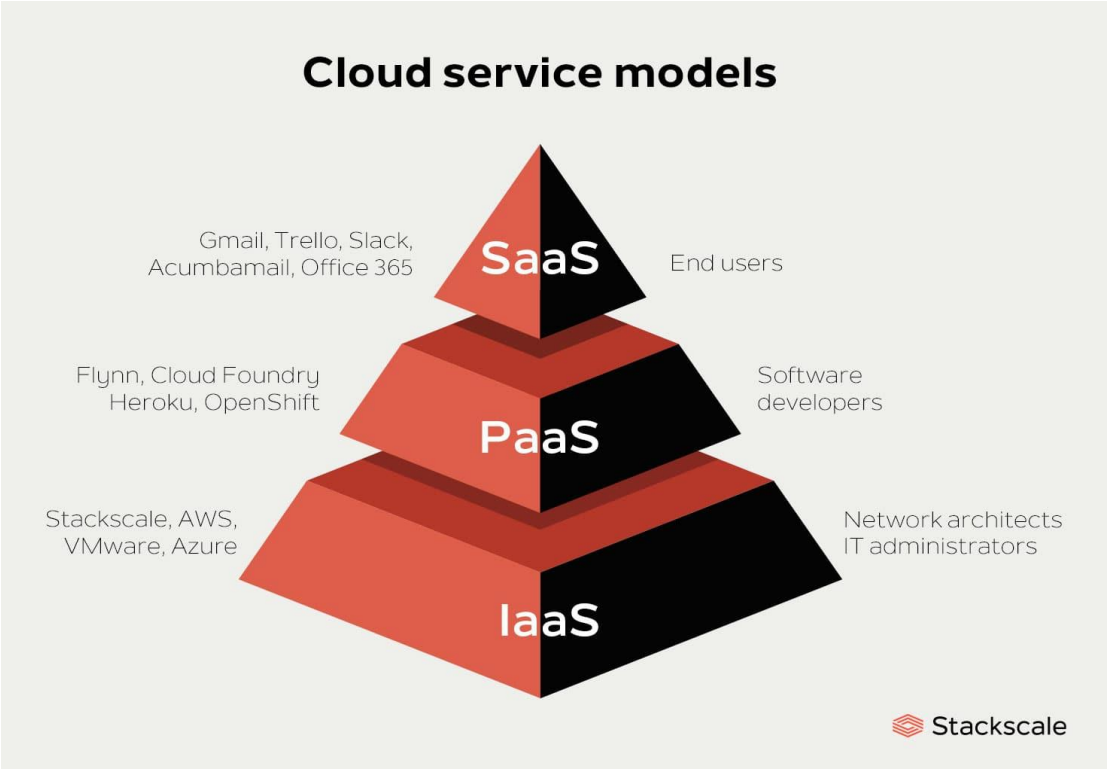
In-Class Assessment 1

Sun Yixuan 202383930020

(a) Describe the three primary cloud service models in cloud computing infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Provide specific examples of how each model can be applied in the context of software development.



1. Infrastructure as a Service (IaaS)

Definition: IaaS provides virtualized *fundamental IT infrastructure* (e.g., servers, storage, networking, virtual machines) over the internet. Users rent these resources on a pay-as-you-go basis instead of purchasing and maintaining physical hardware. IaaS gives full control over the OS, middleware, and applications, while the cloud provider manages the underlying hardware, data centers, and network infrastructure.

Key Characteristics:

- Self-service provisioning (spin up/down VMs in minutes).
- Resource scalability (adjust CPU, RAM, storage based on demand).
- Pay-as-you-go pricing (only pay for resources used).

Software Development Application Example:

A startup building a microservices-based e-commerce platform needs flexible server resources for testing and staging. Instead of buying physical servers (which are costly and underutilized during low-traffic periods), the team uses **Amazon EC2 (AWS IaaS)** to:

- Provision Linux/Windows VMs for hosting individual microservices (e.g., product catalog, payment processing).
- Attach **Amazon S3 (IaaS storage)** to store test data and log files generated during development.
- Use **Amazon VPC (IaaS networking)** to isolate the staging environment from production, ensuring secure testing.
- Scale down VMs after testing to avoid unnecessary costs, and scale up during peak development cycles (e.g., before a feature launch).

2. Platform as a Service (PaaS)

Definition: PaaS provides a *complete development and deployment platform* over the cloud, abstracting the underlying infrastructure (servers, OS, networking). It includes tools for coding, testing, debugging, deploying, and managing applications—allowing developers to focus solely on writing code instead of managing infrastructure.

Key Characteristics:

- Pre-built development tools (IDEs, APIs, databases).
- Automated deployment, scaling, and maintenance (e.g., OS patching, database updates).
- Support for multiple programming languages (Python, Java, Node.js).

Software Development Application Example:

A team building a real-time collaboration tool (e.g., a project management app with chat) uses **Heroku (PaaS)** to accelerate development:

- Use Heroku's pre-configured Node.js runtime to host the app's backend, avoiding manual setup of Node.js, Nginx, or Linux.
- Integrate Heroku's managed PostgreSQL database (built into the PaaS) to store user data and project details—no need to manage database backups or version updates.

- Use Heroku Pipelines to automate testing and deployment: push code to GitHub, and Heroku automatically runs unit tests, builds the app, and deploys it to staging/production.
- Scale the app with a single command (e.g., heroku ps:scale web=3) to handle more users during beta testing—Heroku manages load balancing and resource allocation.

3. Software as a Service (SaaS)

Definition: SaaS delivers *fully functional software applications* over the internet (usually via a web browser). Users access the app on a subscription basis (e.g., monthly/annual fees) and do not need to install, maintain, or update the software—all backend infrastructure, updates, and security are managed by the SaaS provider.

Key Characteristics:

- No local installation (access via web/mobile apps).
- Automatic updates (users always have the latest version).
- Multi-tenant architecture (multiple users share the same app instance, reducing costs).

Software Development Application Example:

A small development team building a customer relationship management (CRM) tool for local businesses uses SaaS tools to streamline their workflow:

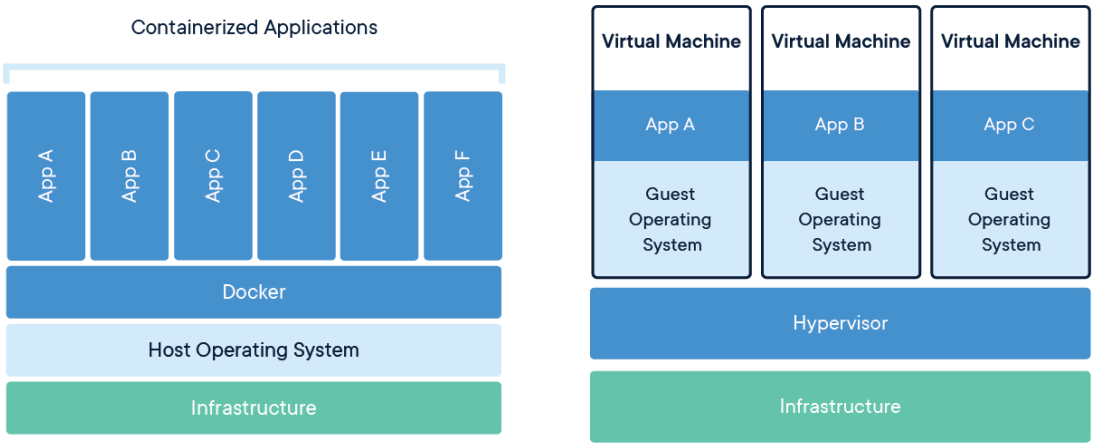
- **GitHub (SaaS for version control)**: Collaborate on code, track changes, and resolve merge conflicts in real time—no need to set up a local Git server.
- **Jira (SaaS for project management)**: Track development tasks (e.g., "fix login bug," "add payment feature"), assign roles, and monitor sprint progress.
- **Slack (SaaS for communication)**: Share code snippets, debug logs, and deployment updates—integrates with GitHub and Jira to auto-notify the team of code pushes or task status changes.
- All tools are accessed via a browser, so the team can work remotely without installing software on local machines.

(b) What is Docker? Describe a scenario where you would use containerization technologies such as Docker in software development. How does containerization contribute to the development and deployment process of software in this

scenario?

1. What is Docker?

Docker is an open-source platform for **containerization**—a technology that packages an application and all its dependencies (e.g., libraries, runtime, configuration files) into a standardized unit called a *container*. Containers are lightweight, portable, and isolated from the host system and other containers, ensuring the app runs consistently across different environments (e.g., a developer's laptop, a testing server, a production cloud).



Unlike virtual machines (VMs), which virtualize an entire OS, containers share the host OS's kernel—making them faster to start (seconds vs. minutes for VMs) and more resource-efficient (lower CPU/RAM usage).

2. Scenario for Using Docker

A distributed team of 5 developers is building a Python-based machine learning (ML) app that predicts customer churn. The app depends on specific versions of Python (3.9), TensorFlow (2.8), and PostgreSQL (13)—and the team uses a mix of Windows, macOS, and Linux laptops.

Problem Without Docker:

- A developer on Windows installs Python 3.10 (instead of 3.9) and TensorFlow 2.10, causing the app to crash due to version conflicts.
- A developer on macOS sets up PostgreSQL 14, which uses a different configuration than the production server (PostgreSQL 13), leading to database query errors during testing.
- When deploying to AWS EC2 (Linux), the team spends 2 days manually installing dependencies and fixing environment mismatches.

3. How Containerization Contributes to Development/Deployment

Docker solves these problems by standardizing the environment across the entire workflow:

(1) Consistent Development Environments:

The team creates a Dockerfile (a text file defining the app's environment) that specifies:

```
FROM python:3.9-slim    # Use official Python 3.9 base image
RUN pip install tensorflow==2.8 psycopg2-binary==2.9.3    # Install exact dependencies
COPY . /app    # Copy app code into the container
CMD ["python", "/app/churn_prediction.py"]    # Command to run the app
```

Every developer builds a container from this Dockerfile, ensuring the app uses the *exact same dependencies*—regardless of their local OS. No more "it works on my machine" issues.

(2) Simplified Testing:

The team uses **Docker Compose** (a tool for running multi-container apps) to define a testing environment with the app and a PostgreSQL database:

```
# docker-compose.yml
version: '3'
services:
  app:
    build: .    # Build the app container from the Dockerfile
    depends_on:
      - db    # Ensure the database starts first
  db:
    image: postgres:13    # Use official PostgreSQL 13 image
    environment:
      POSTGRES_PASSWORD: test_password
      POSTGRES_DB: churn_db
```

Running docker-compose up starts both containers and connects them. Testers can run automated tests against the containerized app and database, ensuring behavior matches production.

(3) Faster, Reliable Deployment:

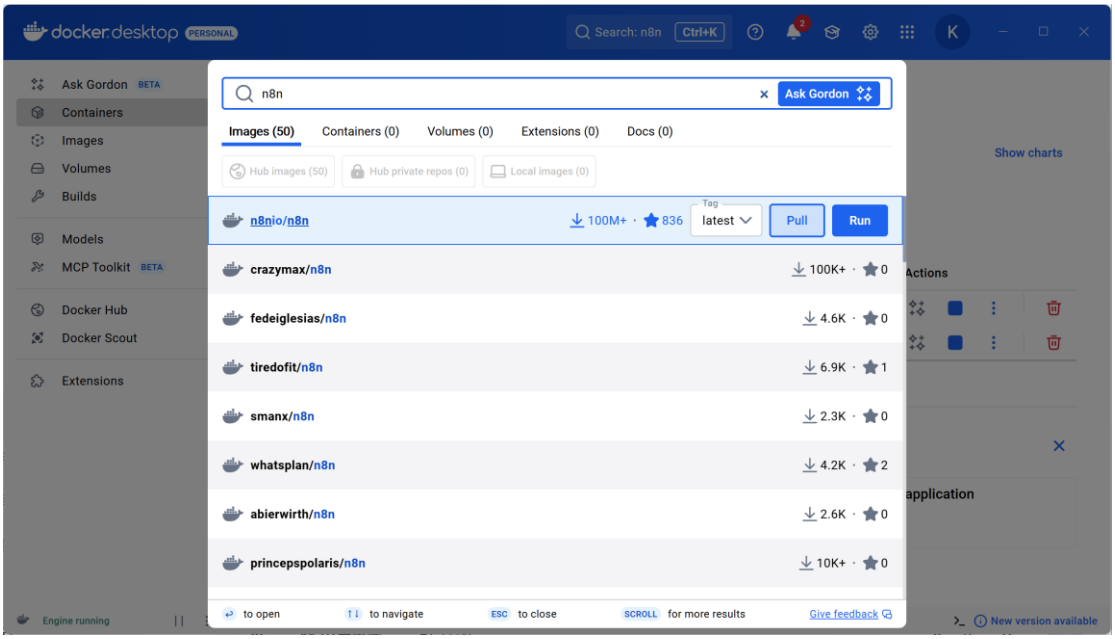
When ready for production, the team pushes the app container to **Docker Hub** (a public/private registry for containers). The AWS EC2 production server pulls the container and runs it with a single command:

```
docker run -d --name churn-app --link postgres:db my-team/churn-prediction:v1.0
```

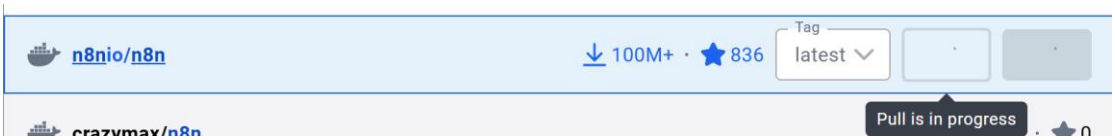
Since the container includes all dependencies, deployment takes minutes (not days) and eliminates environment-related bugs. Scaling is also simpler: just run more copies of the container with docker run.

(c) Deploy n8n (n8n.io) with Docker and capture a screenshot of http://127.0.0.1:5678. Please explain the docker command in detail.

1. Search and pull n8n image from docker desktop.



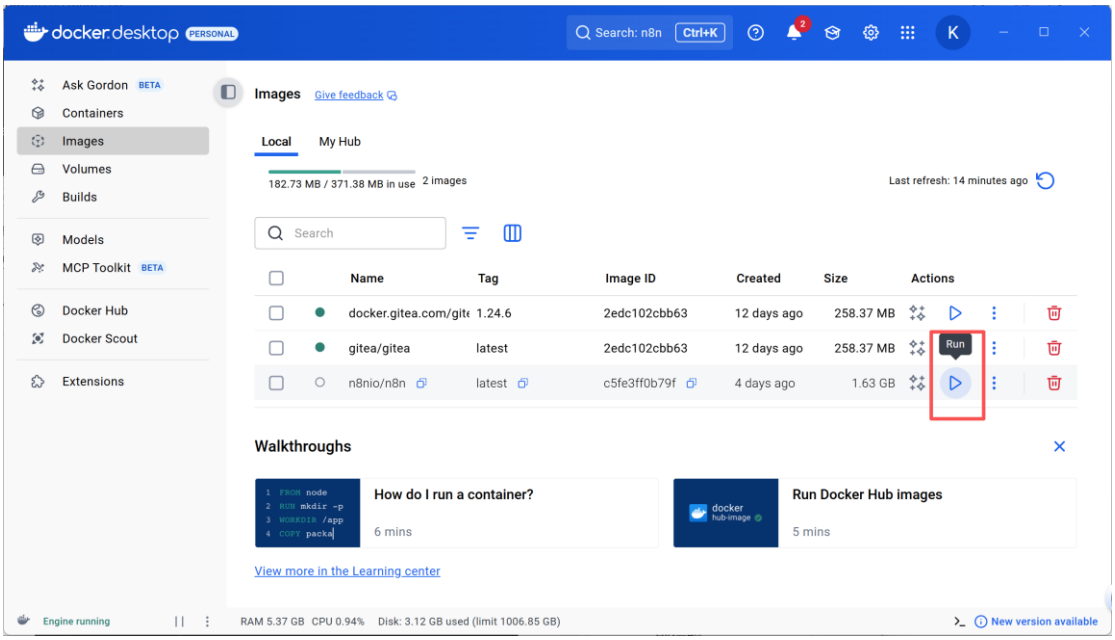
We can click the pull button:



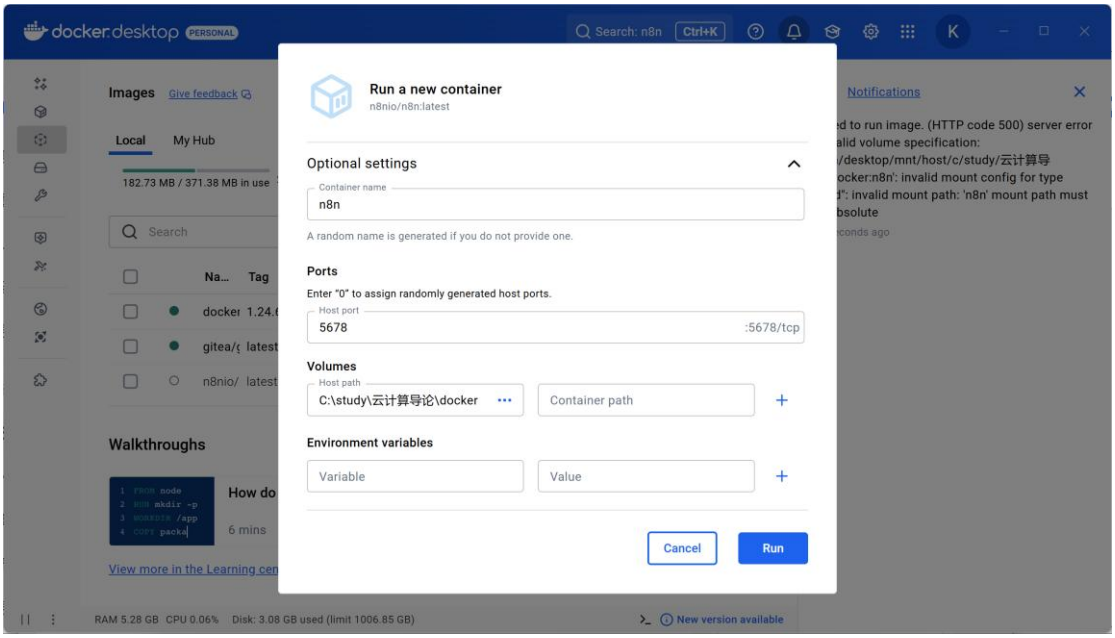
Or directly use “docker pull n8nio/n8n” in powershell.

```
PS C:\Windows\System32> docker pull n8nio/n8n
Using default tag: latest
latest: Pulling from n8nio/n8n
3d64802e5816: Pull complete
4307f681e63e: Pull complete
87ab795ed18b: Pull complete
c4a8c5e8e683: Pull complete
9b94a1e882c2: Pull complete
3efaf331e66e: Pull complete
Digest: sha256:c5fe3ff0b79f7831dc21f9c709bdb7eee4fff4453a28ce84c8e9fa5b9f562686
Status: Downloaded newer image for n8nio/n8n:latest
docker.io/n8nio/n8n:latest
PS C:\Windows\System32>
```

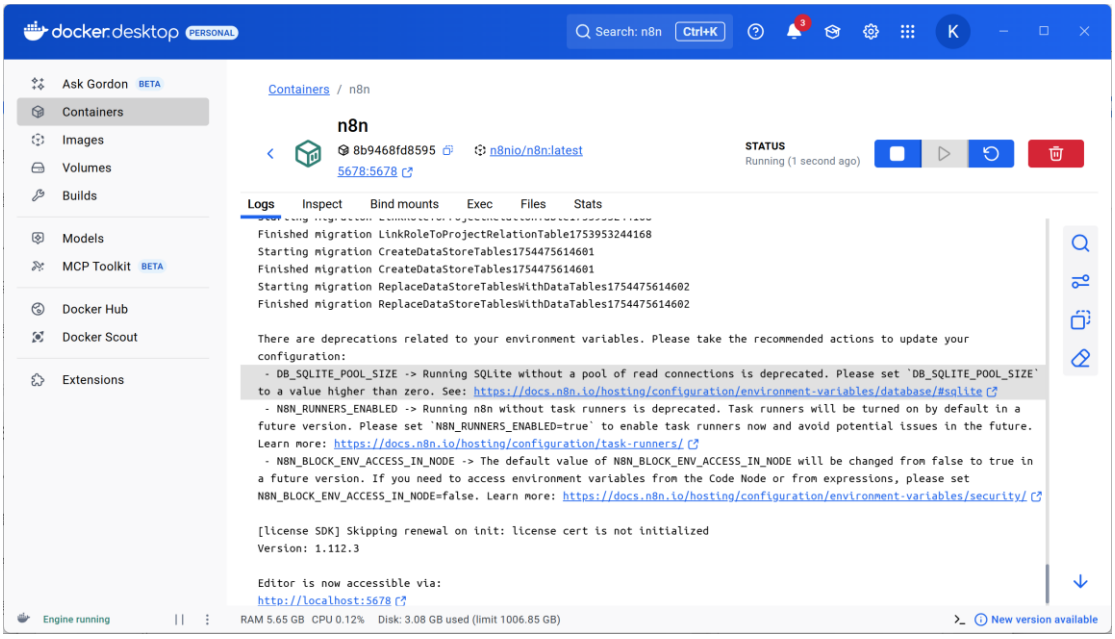
2. Run the image.



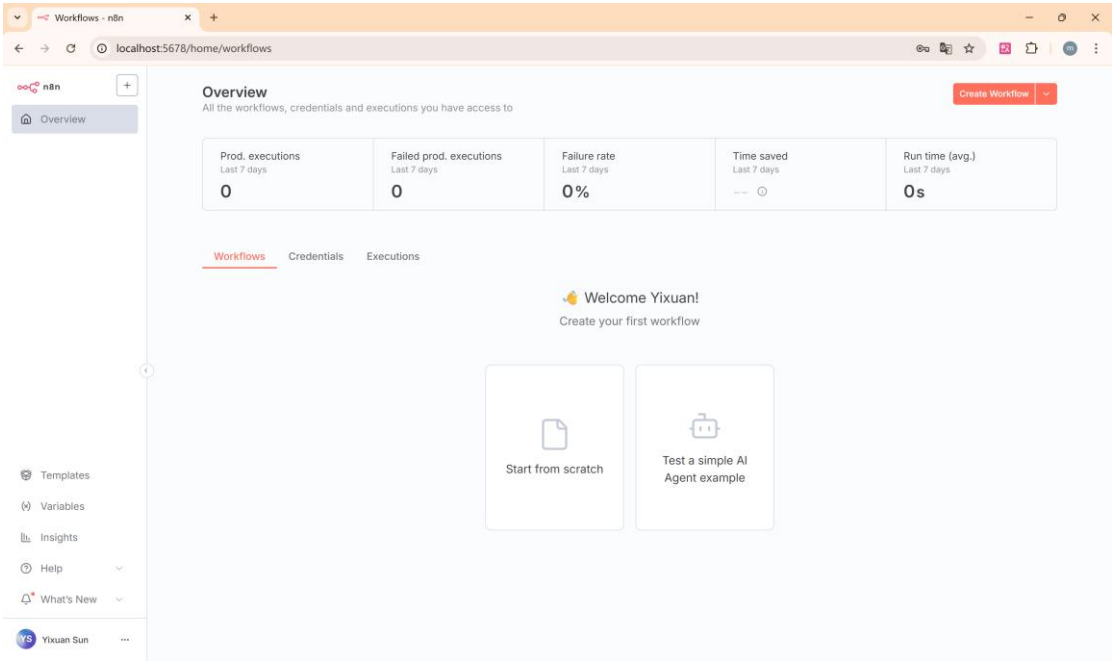
3. Config the container, especially the port: 5678->5678

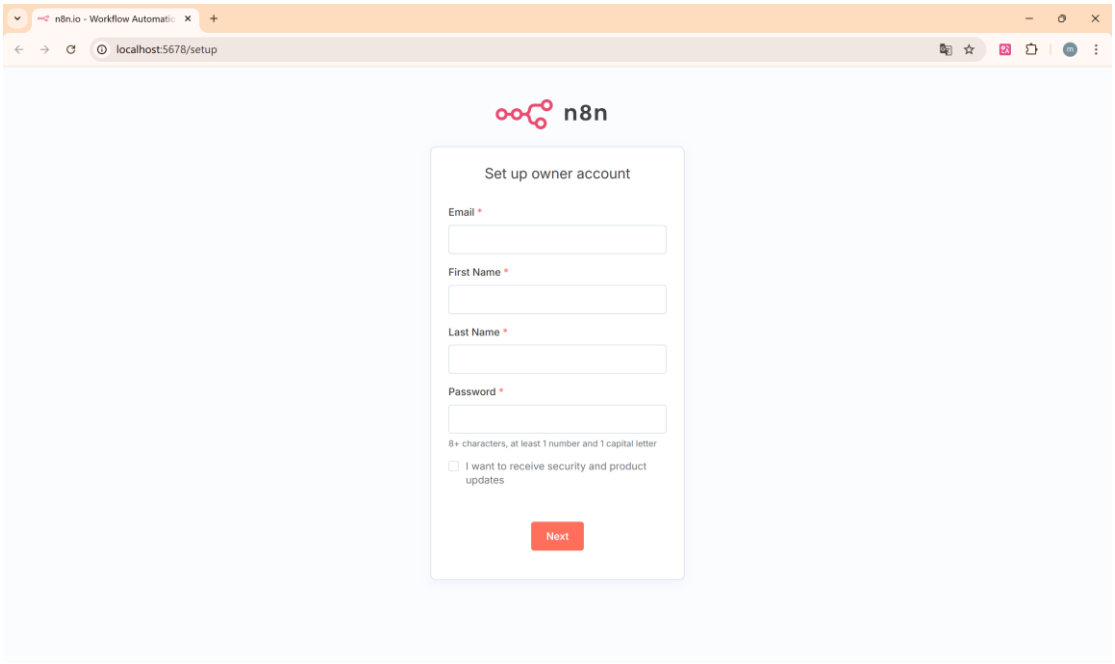


4. Run successfully.



5. Access 127.0.0.1:5678.


Workflows - n8n
localhost:5678/home/workflows
n8n
Overview
Templates
Variables
Insights
Help
What's New
Yixuan Sun
Overview
All the workflows, credentials and executions you have access to
Create Workflow
Prod. executions
Last 7 days
0
Failed prod. executions
Last 7 days
0
Failure rate
Last 7 days
0%
Time saved
Last 7 days
Run time (avg.)
Last 7 days
0s
Workflows
Credentials
Executions
Welcome Yixuan!
Create your first workflow
Start from scratch
Test a simple AI Agent example


n8n.io - Workflow Automati
localhost:5678/setup
n8n
Set up owner account
Email *
First Name *
Last Name *
Password *
8+ characters, at least 1 number and 1 capital letter
I want to receive security and product updates
Next