### a)1. IaaS (Infrastructure as a Service)

Provides virtualized computing resources such as virtual machines, storage, and networking. Users are responsible for managing the OS, runtime, middleware, and applications, while the provider manages the underlying physical infrastructure.

**Development Use Cases:**

* Environment setup: Spin up VMs with specific OS versions to mirror production.
* Scalable testing: Create temporary VM clusters for automated or load testing.
* Build agents: Run resource-heavy build pipelines in the cloud.

### 2. PaaS (Platform as a Service)

Provides a managed platform that includes operating systems, runtimes, databases, and common services. Developers focus only on application code and configuration without managing the underlying servers.

**Development Use Cases:**

* Rapid deployment: Push code or containers to the platform and let it handle scaling.
* Built-in CI/CD: Integrate directly with Git repositories for automated deployment.
* Managed databases and caching: Ideal for prototypes and reducing maintenance overhead.

### 3. SaaS (Software as a Service)

Delivers fully functional applications accessible over the internet. The provider manages everything (infrastructure, updates, and security), and users just consume the service.

**Development Use Cases:**

* Source control and collaboration: Use hosted Git services for repositories and pull requests.
* Project management: Use agile boards to track tasks, bugs, and releases.
* Testing & monitoring: Integrate SaaS tools for cross-platform testing and performance monitoring.

B)Docker is a platform that automates the packaging, distribution, and running of applications inside lightweight, portable containers. A Docker container bundles an application together with its dependencies and runtime, so the app runs consistently across different environments.

Key concepts: **images** (immutable templates), **containers** (running instances of images), **Dockerfile** (image recipe), and **registries** (e.g., Docker Hub) for storing images.

## Practical scenario: Microservices development & CI/CD for a web application

Imagine a team building a web application composed of three services:

1. **API service** (Node.js + Express)
2. **Worker** that processes background jobs (Python)
3. **Postgres** database

Developers want reproducible local environments, reliable CI builds, and consistent production deployments across cloud providers. Docker is used to containerize each service and Docker Compose manages them during local development. CI pipelines build images, run tests inside containers, push images to a registry, and production uses those same images (or a Kubernetes cluster) for deployment.

### Why use Docker here?

* **Consistency:** The same image runs in dev, CI, staging, and production.
* **Isolation:** Each service has its own dependencies and runtime without conflicts.
* **Faster onboarding:** New developers run a single command to start the full system.
* **CI reliability:** Tests run in clean, reproducible containers.
* **Portability:** Images run on any host that supports Docker or on orchestration platforms like Kubernetes.

## How containerization improves development and deployment (step-by-step)

1. **Local development:** Developers run the whole stack via docker-compose up. Each service runs in its container with declared dependencies, so "works on my machine" disappears.
2. **CI/CD:** CI builds images from Dockerfiles, runs unit/integration tests inside containers, and publishes images to a registry when tests pass. This makes the artifact that will be deployed identical to what CI tested.
3. **Staging & production:** Deployment uses the exact image tags from the registry. Rollbacks are quick by switching image tags. Orchestration platforms (Kubernetes) handle scaling, health checks, and service discovery.
4. **Resource efficiency:** Containers share the host kernel and have smaller footprints than full VMs—allowing higher density and faster startup.

### Small example: Dockerfile for the Node API service



### Small example: docker-compose.yml for local dev



## Benefits and trade-offs

| **Benefits** | **Trade-offs** |
| --- | --- |
| Reproducibility, faster startup, efficient resource use, easier scaling | Requires learning Docker, image size management, extra layer of networking and orchestration complexity for large systems |
| Portable artifacts (images) across environments | Security hardening and runtime monitoring need attention |

### When to prefer containers over VMs

* Microservices or polyglot architectures where services need isolated runtimes.
* Rapid CI pipelines that build, test, and deploy identical artifacts.
* Teams needing fast developer onboarding and consistent local stacks.

C）

## Step 1: Pull n8n Docker Image

docker pull n8nio/n8n:latest

This command downloads the latest n8n image from Docker Hub.

## Step 2: Run n8n Container

docker run -d --name n8n -p 5678:5678 -v n8n\_data:/home/node/.n8n n8nio/n8n:latest

### Command Parameters Explained:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Description** | **Purpose** |
| -d | Detached mode | Runs the container in the background |
| --name n8n | Container name | Assigns a meaningful name to the container for easy management |
| -p 5678:5678 | Port mapping | Maps container port 5678 to host port 5678 (host:container) |
| -v n8n\_data:/home/node/.n8n | Volume mapping | Persists n8n data in a Docker volume named "n8n\_data" |
| n8nio/n8n:latest | Image name | Specifies the Docker image to run |

## Step 3: Advanced Production Deployment (Optional)

docker run -d --name n8n-server \  
-p 5678:5678 \  
-v n8n\_data:/home/node/.n8n \  
-e N8N\_BASIC\_AUTH\_ACTIVE=true \  
-e N8N\_BASIC\_AUTH\_USER=admin \  
-e N8N\_BASIC\_AUTH\_PASSWORD=your\_secure\_password \  
-e N8N\_HOST=127.0.0.1 \  
n8nio/n8n:latest

### Environment Variables Explained:

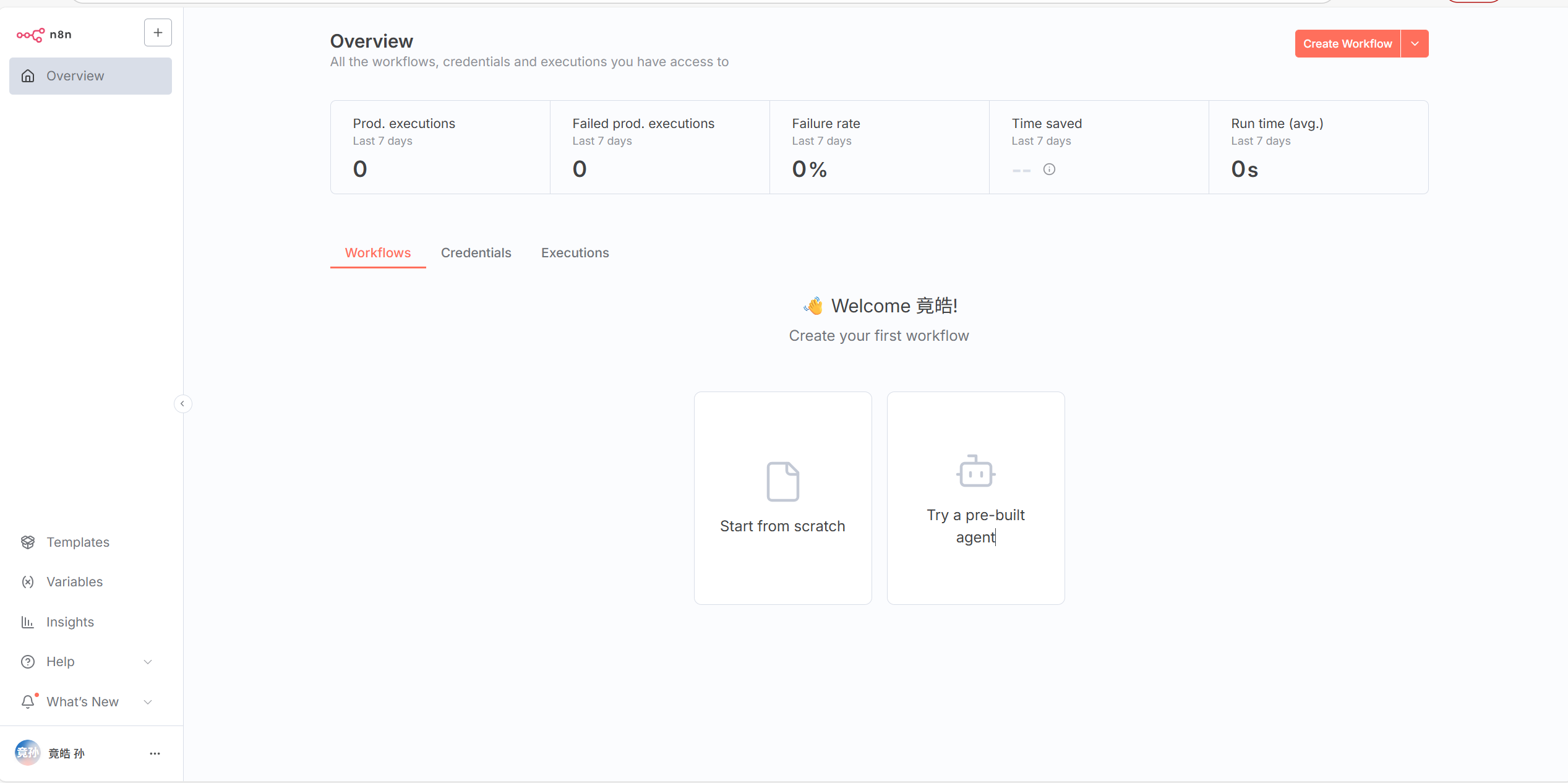
|  |  |  |
| --- | --- | --- |
| **Variable** | **Description** | **Default Value** |
| N8N\_BASIC\_AUTH\_ACTIVE | Enables basic authentication | false |
| N8N\_BASIC\_AUTH\_USER | Username for authentication | - |
| N8N\_BASIC\_AUTH\_PASSWORD | Password for authentication | - |
| N8N\_HOST | Host address n8n listens on | localhost |

## Step 4: Access n8n Web Interface

After running the container, open your web browser and navigate to:

http://127.0.0.1:5678

### Screenshot of n8n Web Interface (Port 5678)



## Step 5: Useful Management Commands

# Check container status  
docker ps  
  
# View container logs  
docker logs n8n  
  
# Stop the container  
docker stop n8n  
  
# Start the container  
docker start n8n  
  
# Remove the container  
docker rm n8n