12-Factor App Methodology 🌟

The **12-Factor App** methodology is a set of best practices for building **modern, scalable, maintainable, and cloud-ready applications**. Originally introduced by Heroku, it's widely used in **microservices and cloud-native applications**.

It focuses on **codebase management, dependencies, configuration, and operational processes**.

1️⃣ Codebase – One codebase tracked in version control, many deploys 🗂️

- **Rule:** One codebase per app, tracked in a version control system (like Git), deployed to multiple environments (dev, staging, production).
- **Why it matters:** Ensures all deploys are consistent and reproducible.
- **Example:**
    - Git repo myapp/ → can be deployed to staging or production.

2️⃣ Dependencies – Explicitly declare and isolate 📦

- **Rule:** Don't rely on system-wide packages. Declare all dependencies explicitly using dependency managers (npm, pip, Maven, etc.).
- **Why it matters:** Makes your app portable and prevents "works on my machine" issues.
- **Example:**
    - Python: requirements.txt

- Node.js: package.json

3️⃣ Config – Store configuration in the environment ⚙️

- **Rule:** Keep configuration (like database URLs, API keys, credentials) in **environment variables**, not in code.
- **Why it matters:** Allows different settings per environment without changing the code.

Example:

```
export DATABASE_URL=postgres://user:pass@host/db
export API_KEY=abcdef12345
```

-

4️⃣ Backing Services – Treat external services as attached resources 🔌

- **Rule:** Databases, caches, messaging systems, or storage are considered **attached resources**. Treat them as replaceable.
- **Why it matters:** Makes your app flexible and cloud-ready.
- **Example:** Swapping a local PostgreSQL with Amazon RDS requires **no code change**, just config change.

5️⃣ Build, release, run – Strictly separate stages 🏗️➡️🚀

- **Rule:** The app lifecycle has **3 stages**:

1. **Build** – Converts code into a deployable artifact.
2. **Release** – Combines build with environment config.
3. **Run** – Executes the app in the execution environment.

- **Why it matters:** Clean separation ensures reliable deployments.

6️⃣ Processes – Execute as stateless processes ⚡

- **Rule:** Apps should run as **one or more stateless processes**.
- **Why it matters:** Statelessness ensures scalability and resilience; state is stored in external services like databases.
- **Example:** Web processes, background workers.

7️⃣ Port binding – Export services via port binding 🌐

- **Rule:** Apps should be **self-contained services** that listen on a port.
- **Why it matters:** Avoids dependency on runtime injection from external servers.

Example:

```
python app.py --port $PORT
```

- The app itself serves HTTP traffic.

8️⃣ Concurrency – Scale via the process model 🔄

- **Rule:** Scale apps by running multiple **processes** (workers, web processes) rather than threads within a process.
- **Why it matters:** Makes scaling predictable and manageable.
- **Example:** Run 4 web processes + 2 worker processes for background jobs.

9️⃣ Disposability – Fast startup and graceful shutdown 🏃‍♂️💨

- **Rule:** Processes should be **quick to start** and **gracefully terminate**.
- **Why it matters:** Enables rapid scaling, deployment, and resilience to crashes.
- **Example:** Use **signals** to handle shutdowns and cleanup resources.

🔟 Dev/prod parity – Keep dev, staging, production similar 🧪

- **Rule:** Minimize the gap between development, staging, and production.
- **Why it matters:** Reduces bugs caused by environment differences.
- **Example:** Use the same database engine in all environments.

1️⃣1️⃣ Logs – Treat logs as event streams 📝

- **Rule:** Don't store logs in files. Treat logs as **streams** that can be aggregated or analyzed externally.

- **Why it matters:** Makes logging scalable and compatible with cloud platforms.
- **Example:** Send logs to ELK Stack, Splunk, or CloudWatch.

1️⃣2️⃣ Admin processes – Run one-off tasks as processes 🛠️

- **Rule:** Run administrative/maintenance tasks (DB migrations, scripts) as one-off processes using the same environment as regular processes.
- **Why it matters:** Ensures consistency with production and avoids surprises.

Example:

```
heroku run python manage.py migrate
```

✅ Why 12-Factor App is Important

- **Cloud-ready:** Works perfectly on platforms like AWS, Heroku, Azure.
- **Scalable:** Stateless processes and concurrency allow easy horizontal scaling.
- **Maintainable:** Clear separation of code, config, and processes simplifies development.
- **Portable:** Independent of infrastructure, easily deployable anywhere.

12-Factor App Methodology

Codebase
One codebase tracked in version control, many deploys

Dependencies
Explicitly declare and isolate

Config
Store configuration in the environment

Backing Services
Treat external services as attached resources

Build, release, run
Strictly separate stages

Processes
Execute as stateless processes

Port Binding
Export services via port binding

Concurrency
Scale via the process model

Disposability
Fast startup and graceful shutdown

Dev/prod Parity
Keep dey, staging, production similar

Logs
Treat logs as

Admin Processes
Run one-off tasks