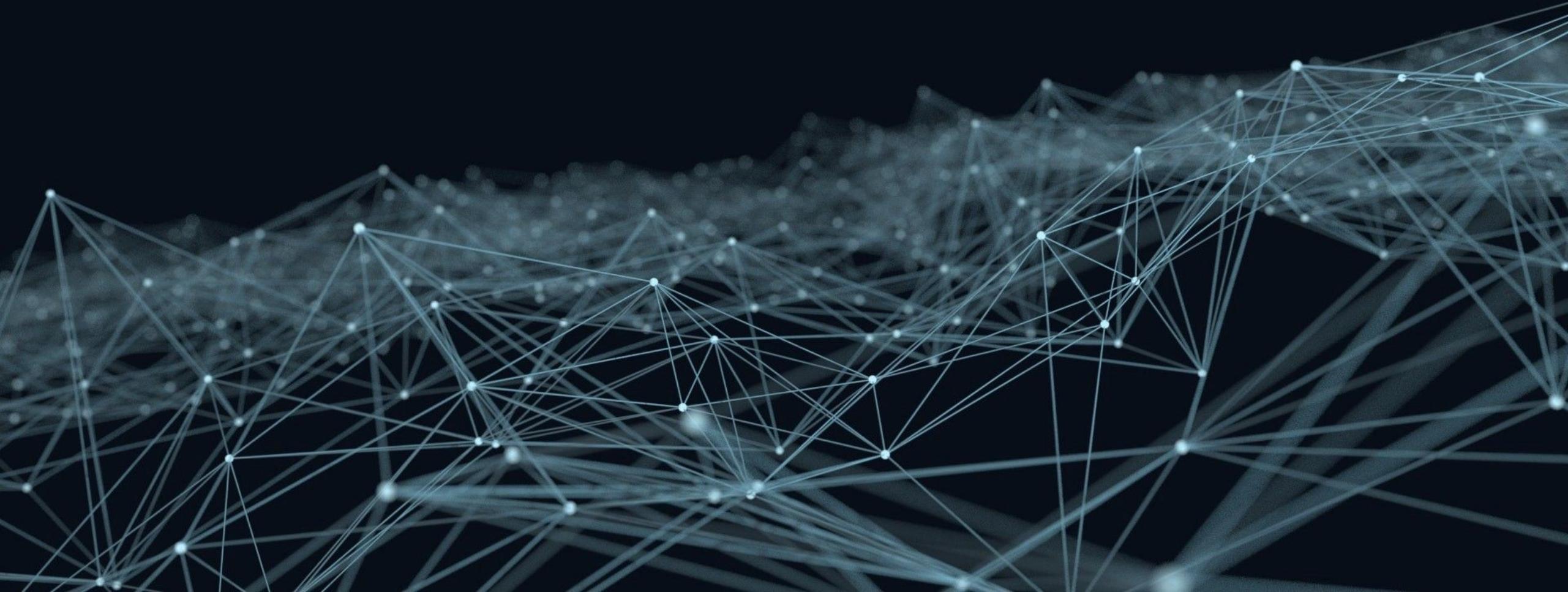


Introduction to Jenkins

Module 10: Build Tools and Language Integration



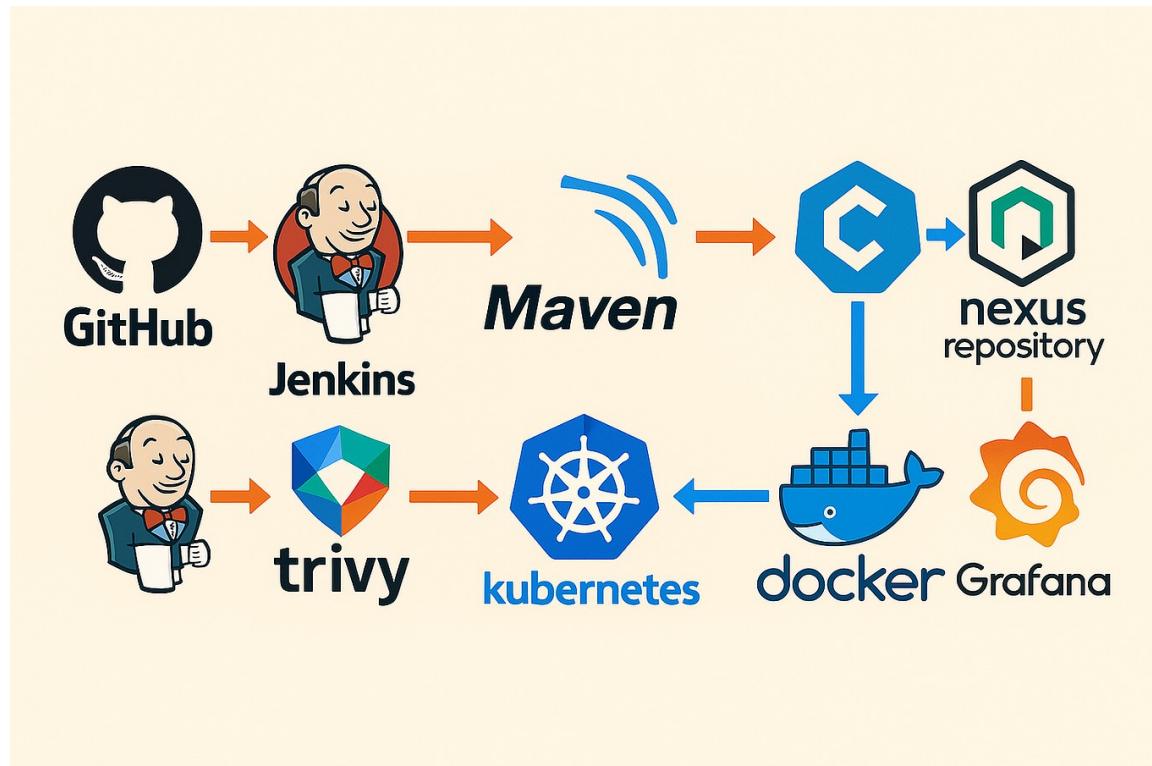
Topics

- Maven, Gradle, and Java builds
- Node.js, npm, and test runners
- Python builds and virtual environments
- .NET, container-based builds, and cross-platform agents



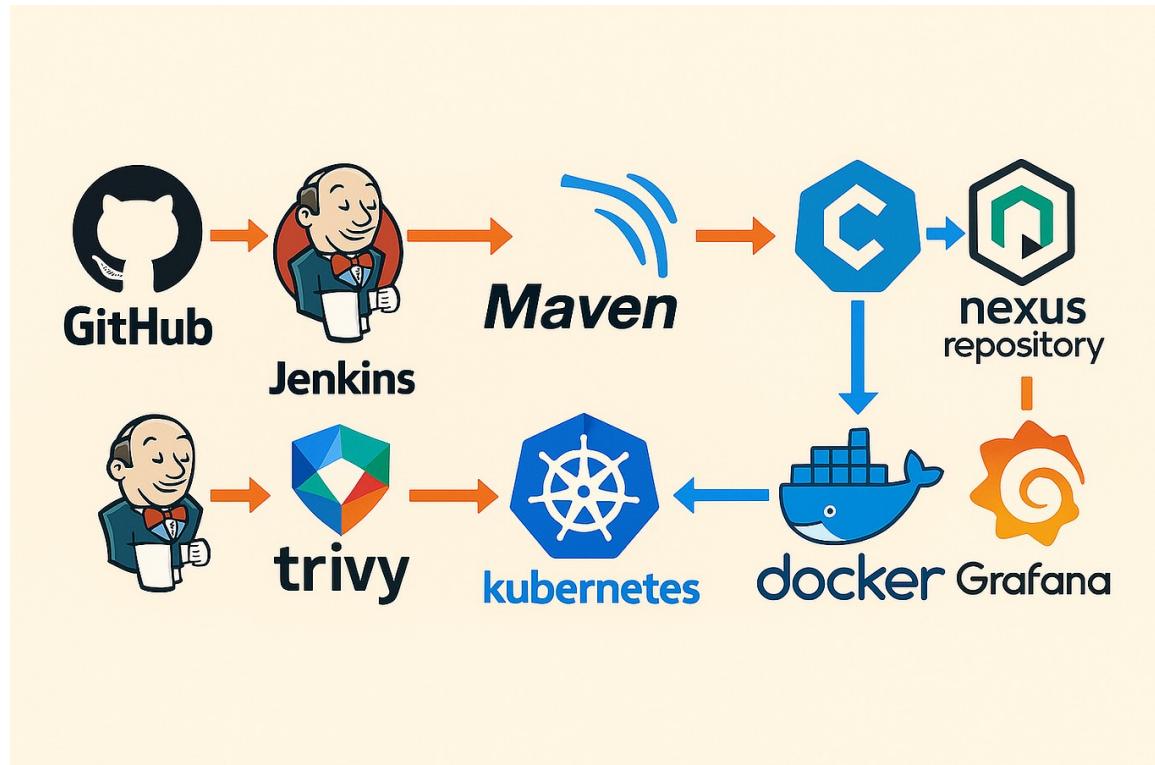
Maven, Gradle, and Java Builds

- Jenkins originated in the Java ecosystem
- Support for Java builds is
 - Mature
 - Well documented
 - Deeply integrated
- Java builds in Jenkins typically rely on
 - Maven or Gradle
 - JDK installations
 - Dependency repositories (class libraries)



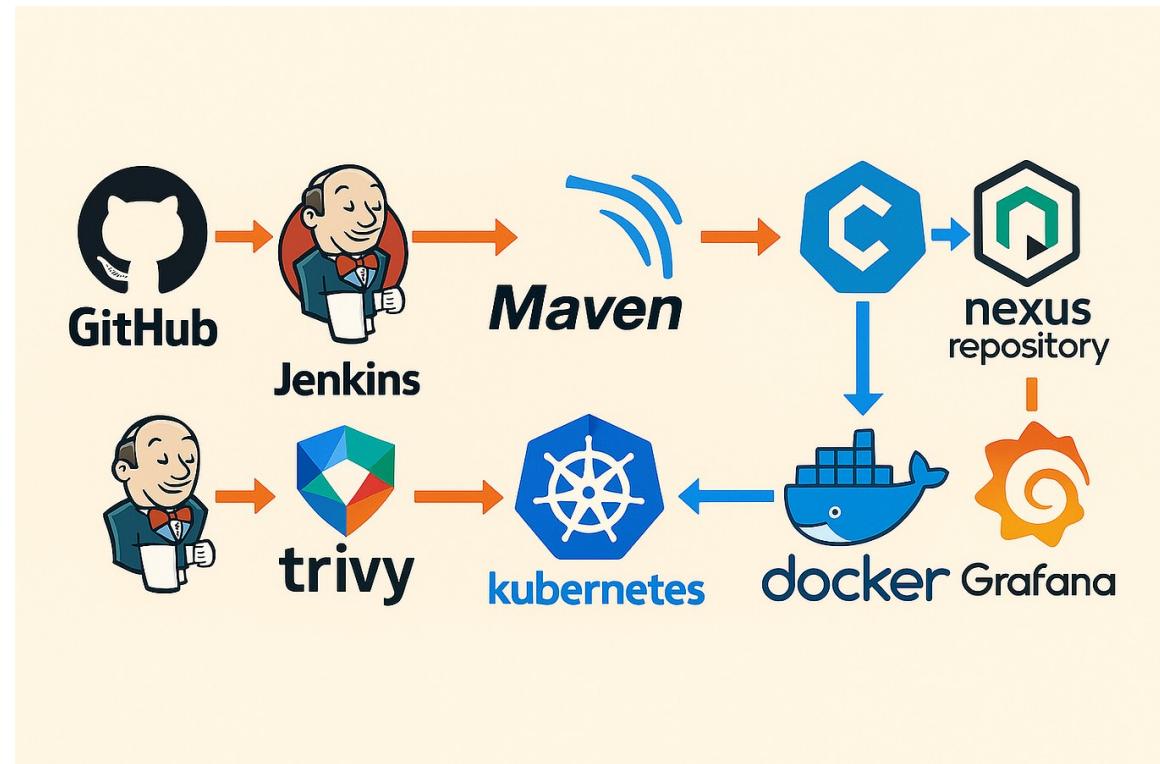
Maven Builds

- Maven is a convention-driven build tool
 - In Jenkins
 - *mvn commands are executed as pipeline steps*
 - *JDK and Maven versions are managed via global tools*
 - *Build results and test reports are easily published*
 - Typical pipeline actions
 - *Compile code*
 - *Run unit tests*
 - *Package artifacts (JAR/WAR)*
 - Maven has its own build lifecycle phases that can be invoked from Jenkins



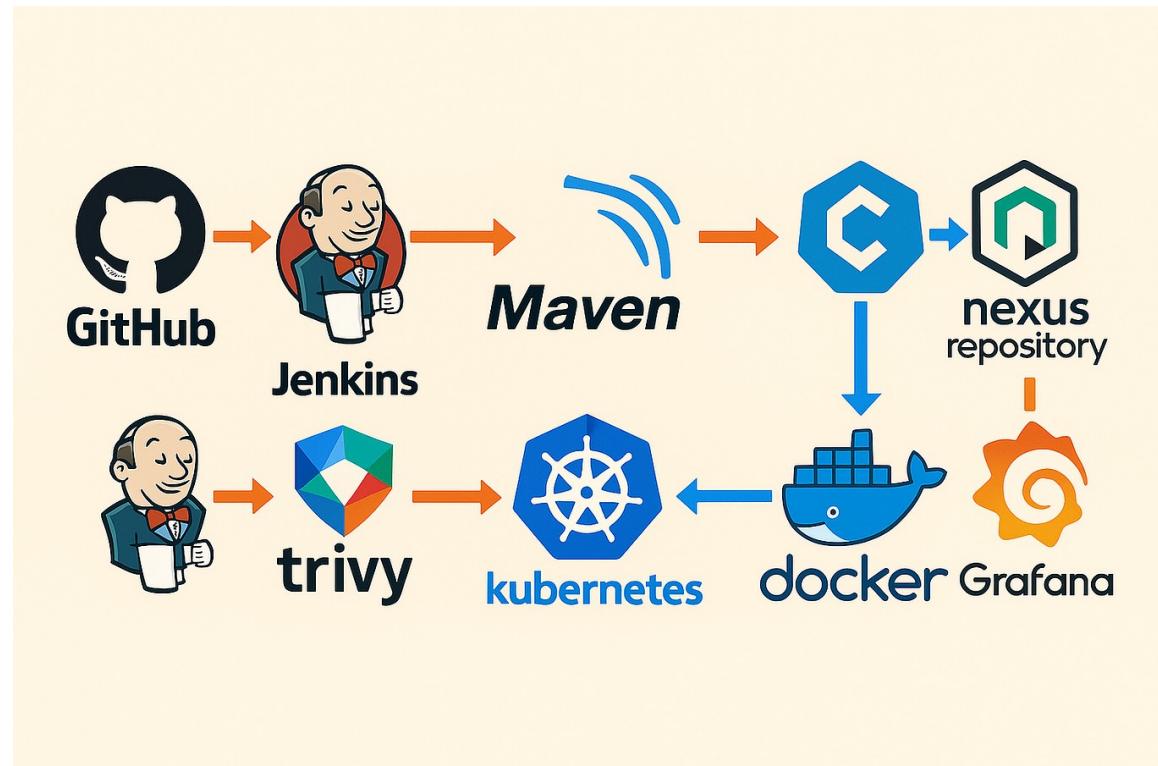
Gradle Builds

- Gradle is a flexible, scriptable build tool
 - In Jenkins
 - *Gradle builds are invoked via the gradle wrapper or CLI*
 - *Caching improves performance but requires careful agent lifecycle management*
 - *Gradle integrates well with Docker-based agents*
 - Gradle is often used when
 - *Builds are complex*
 - *Performance tuning matters*
 - *Multi-project builds exist*



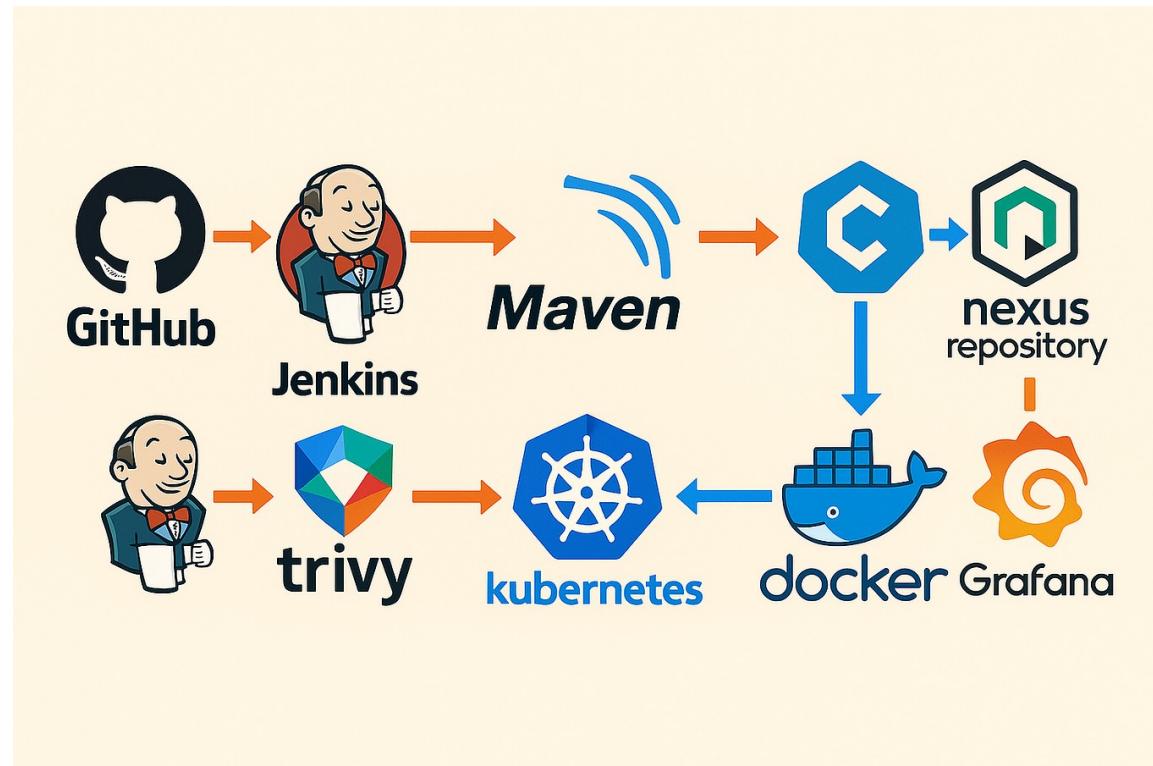
Java Based Tools

- Common Java CI pattern
 - Checkout code
 - Set JDK version
 - Run build tool
 - Publish test results
 - Archive artifacts
- Jenkins does the orchestration
 - Jenkins gets the code
 - Hands the code off to the build tool
 - Accepts the results of the build



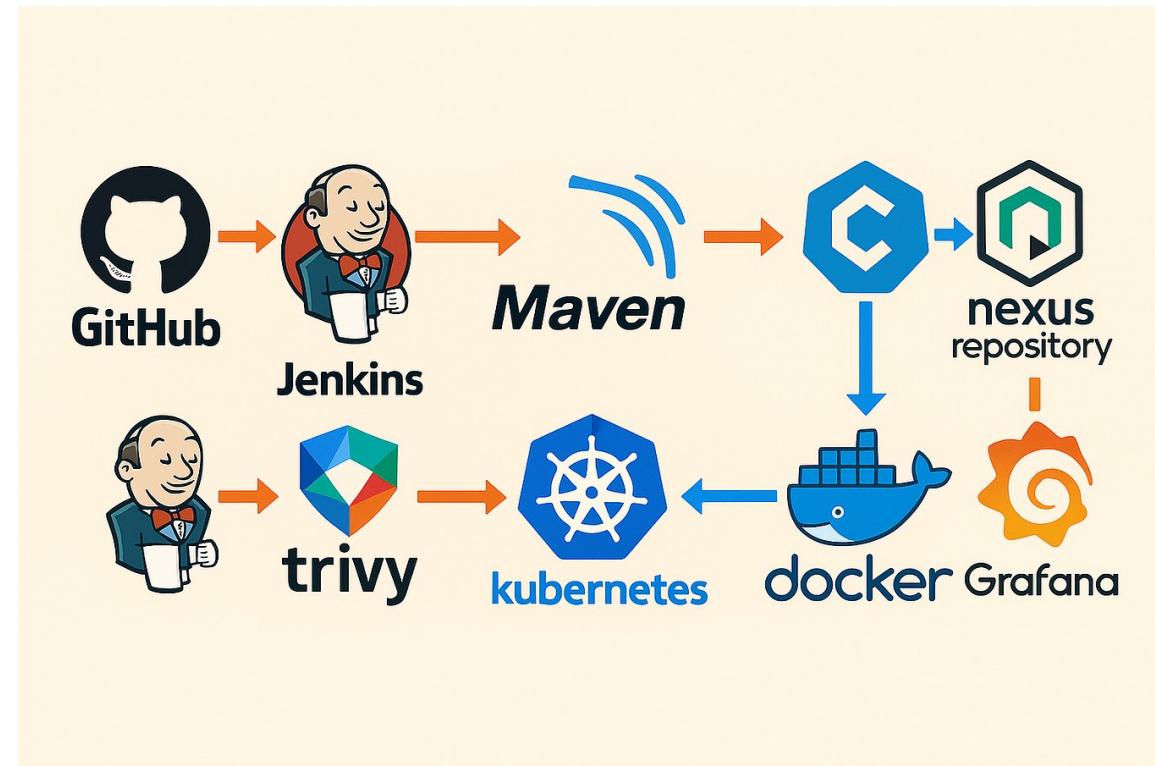
Node.js and npm

- JavaScript builds are
 - Dependency-heavy
 - Fast-moving
 - Sensitive to environment differences across agents
- Jenkins supports Node.js through
 - Node installations
 - Containerized environments
 - Tooling plugins



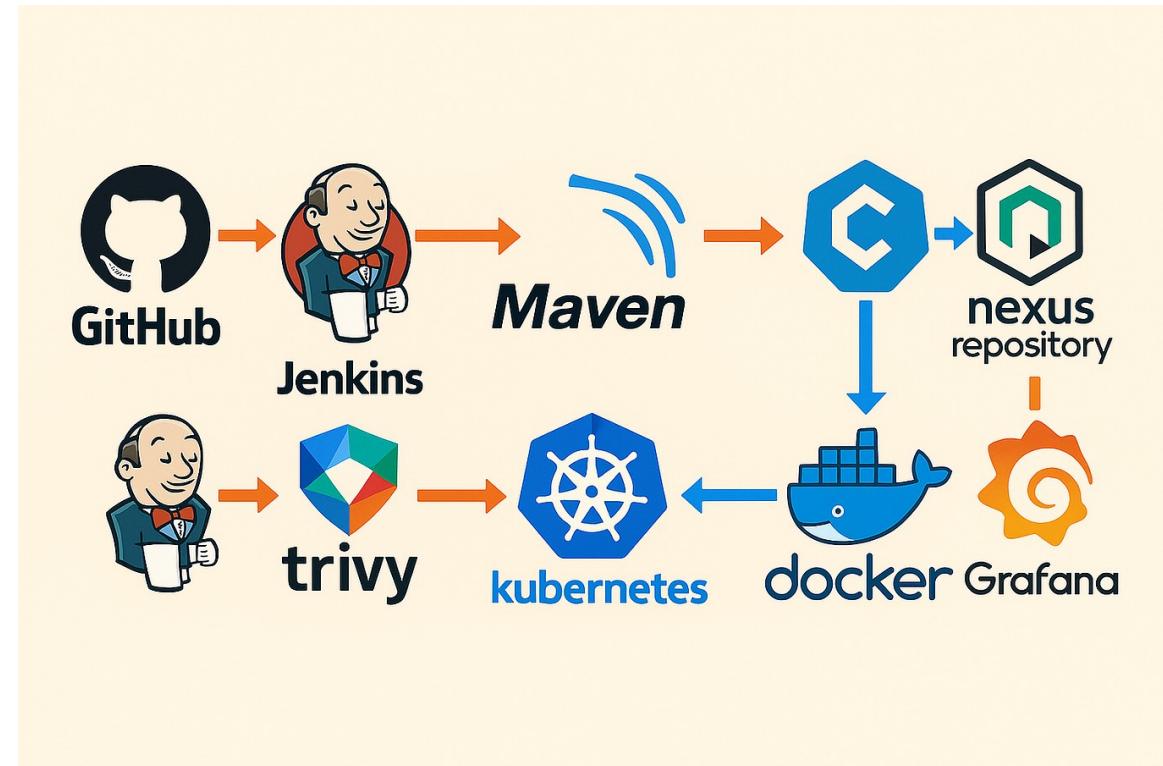
Node.js and npm

- In Jenkins pipelines
 - npm install or npm ci installs dependencies
 - Build scripts run via npm run
 - Lock files ensure reproducibility
- Best practice
 - Use Docker agents to pin Node versions
 - Avoid installing Node globally on static agents



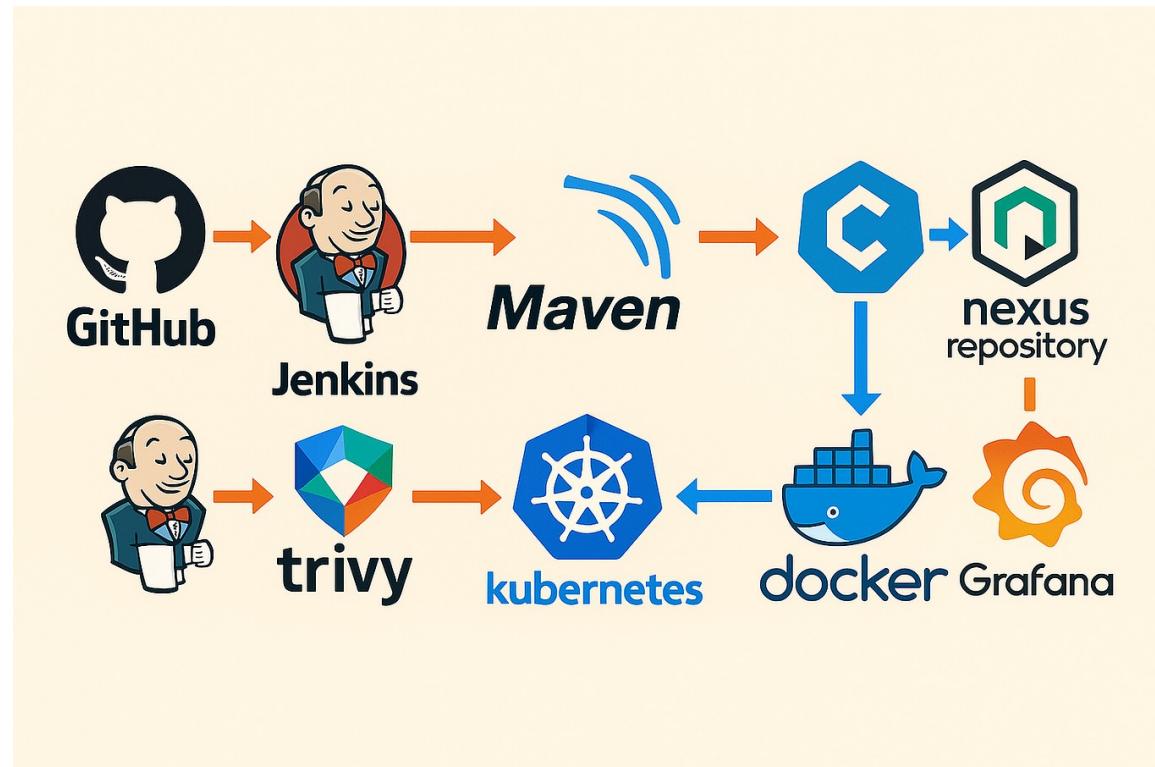
Test Runners

- Common JavaScript test runners
 - Jest
 - Mocha
 - Cypress
 - Playwright
- In CI
 - Tests are run headlessly
 - Results are published as reports
 - Stage failures block code merges



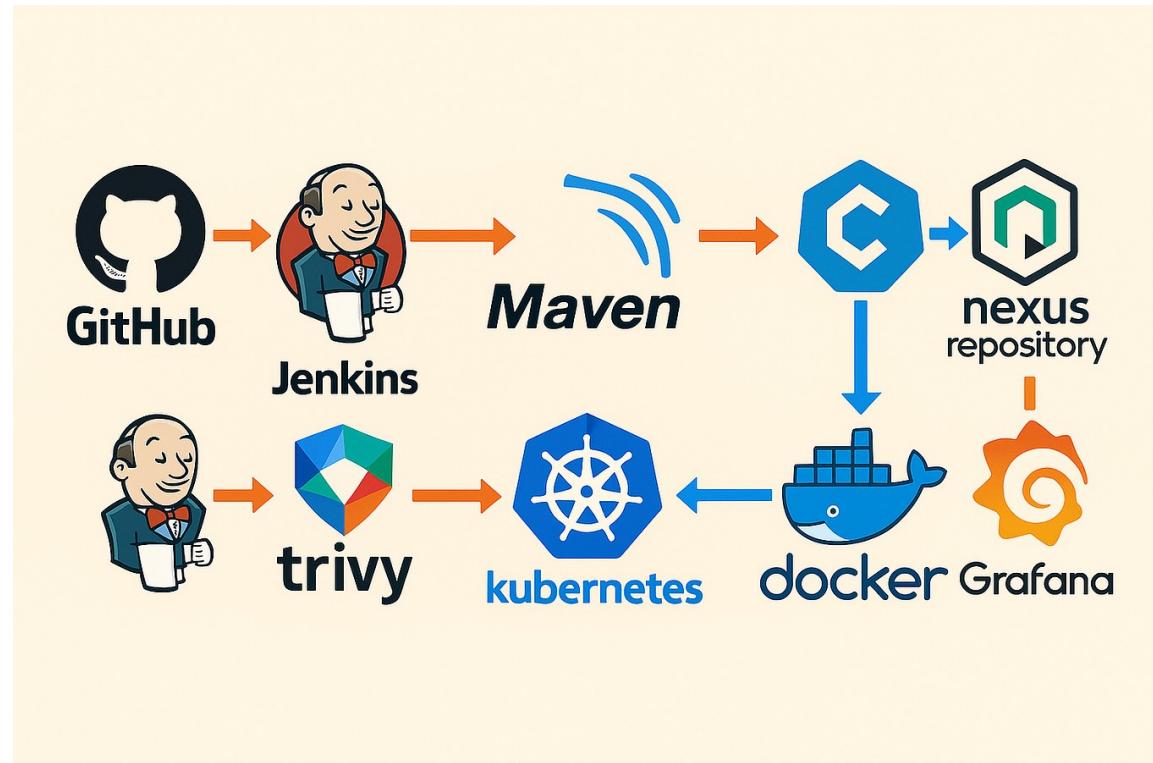
Python Builds

- Python builds
 - Depend heavily on environment isolation
 - Require careful dependency management
 - Benefit strongly from ephemeral agents
- Tooling
 - Many different tools require different Python versions, eg. PyTorch, TensorFlow
 - And these require different module dependencies
 - Historically, Python uses virtual environments to keep versioning and dependencies clean



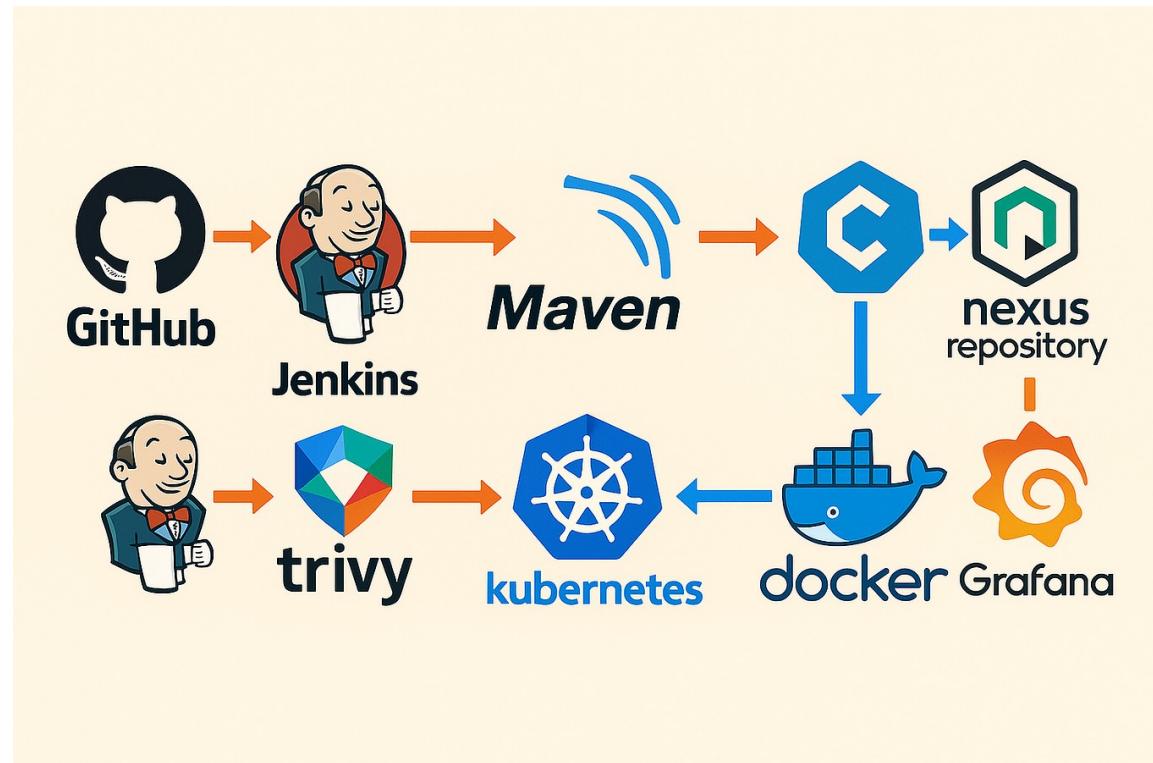
Python Builds

- In Jenkins pipelines
 - Virtual environments isolate dependencies
 - Prevent cross-build contamination
 - Improve reproducibility
 - Common tools
 - *venv*
 - *virtualenv*
 - *Poetry*
 - *Conda*
 - Best practice
 - *Create virtual environments per build*
 - *Avoid shared system Python environments*



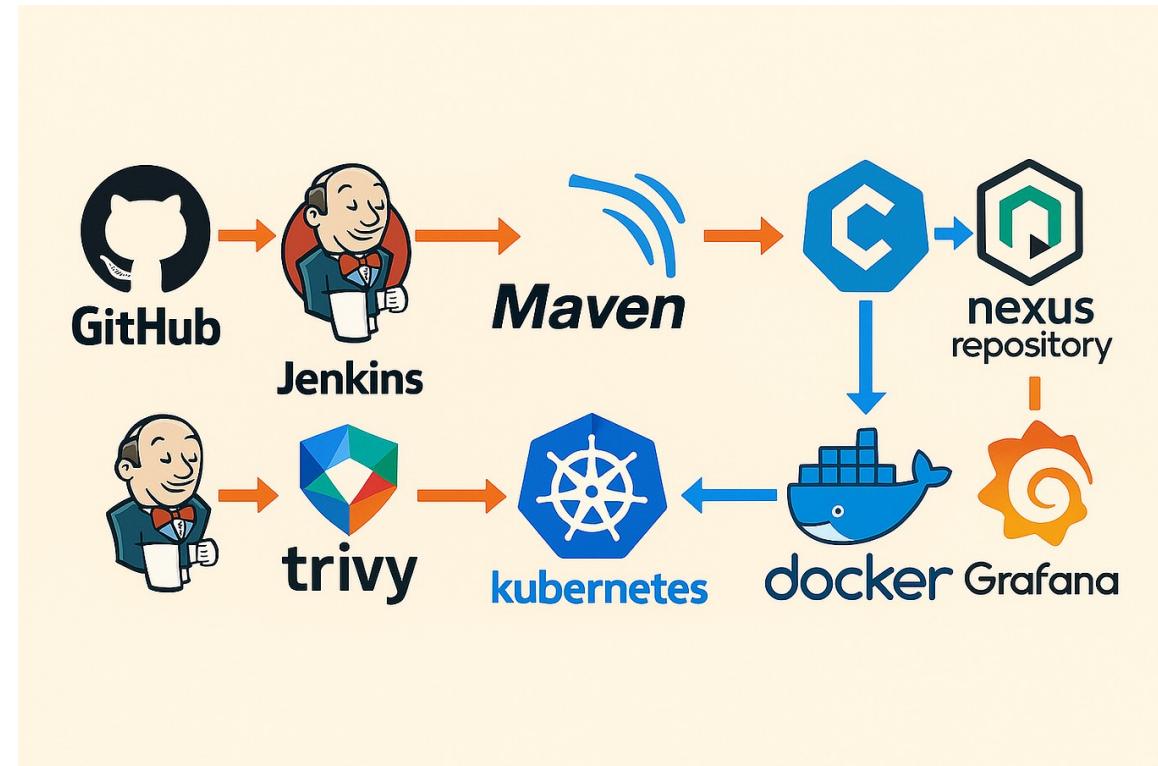
Python CI

- Typical Python CI flow
 - Create virtual environment
 - Install dependencies
 - Run tests (pytest, unittest)
 - Publish test results
 - Package artifacts (optional, depending on project)
- Docker agents are commonly used to
 - Control Python versions
 - Eliminate system-level drift



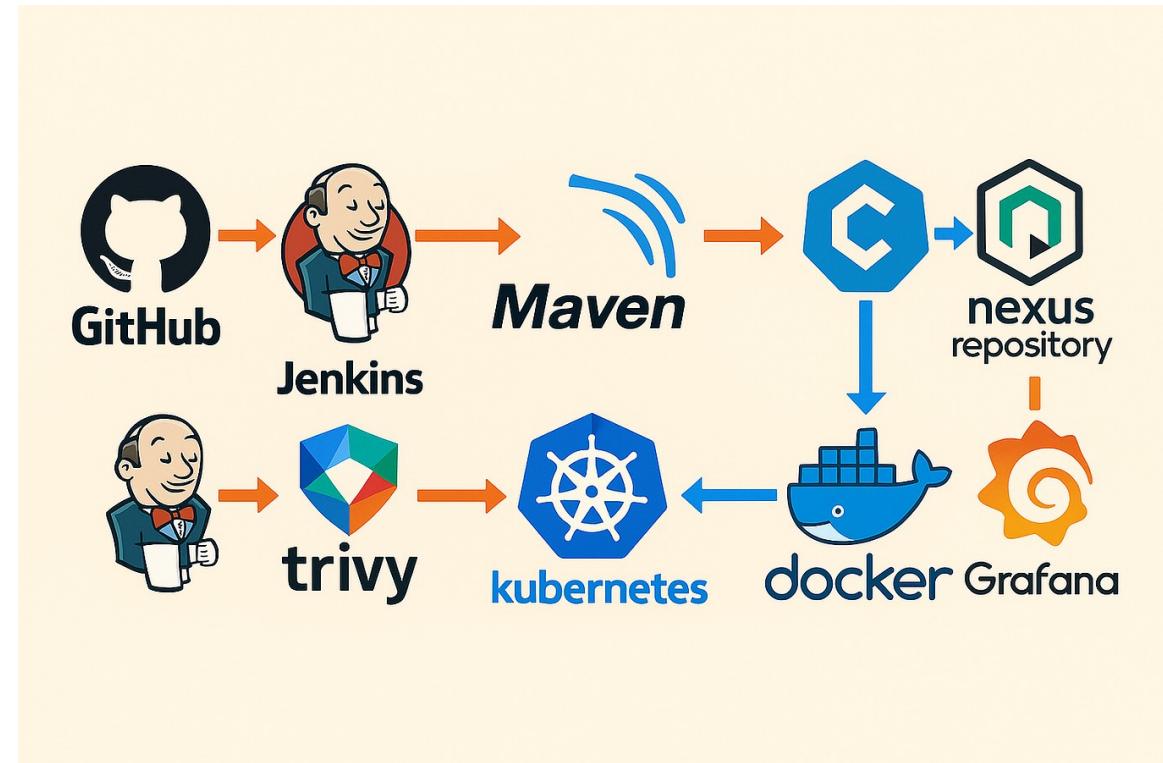
Jenkins and .NET

- Jenkins supports .NET via
 - .NET SDK installations
 - Cross-platform agents
 - Container-based builds
- Modern .NET:
 - Runs on Linux, macOS, and Windows
 - Integrates well with CI pipelines



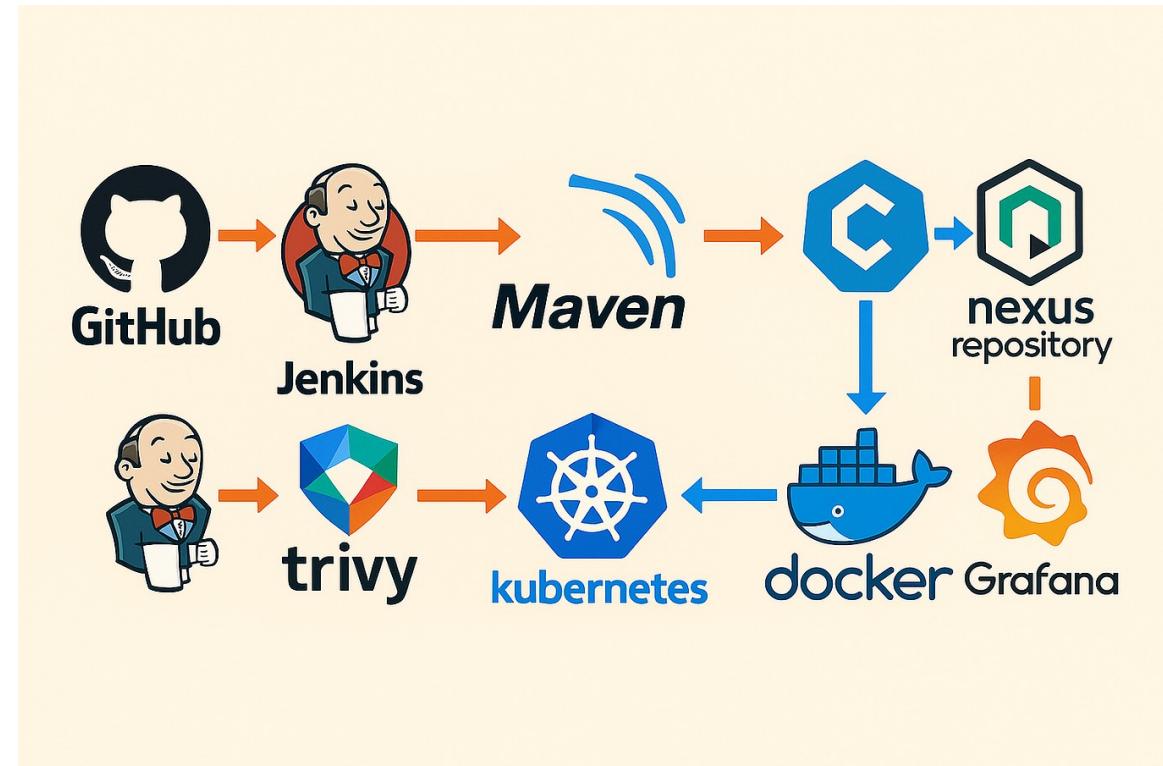
Cross-Platform Builds

- Jenkins can
 - Build the same code on different OSes
 - Use labeled agents for platform-specific work
 - Run matrix builds for compatibility testing
- Example use cases
 - Windows only builds
 - Linux container builds
 - Mixed platform test suites



Cross-Platform Builds

- Containers are increasingly used to
 - Standardize build environments
 - Avoid OS-specific issues
 - Support multi-language stacks
- A single Jenkins instance can
 - Build Java, Node, Python, and .NET
 - Use different containers per pipeline stage
 - Share a common CI platform



Questions

