Application, Infrastructure & Code Structure – Notes

# 1. Application

Definition: A piece of code written to solve a specific problem or provide a particular functionality.  
Purpose: It’s the business logic that end-users interact with.  
Example:  
- Spring Pet Clinic – a sample Java Spring application built to manage a pet clinic system.

Link: <https://github.com/spring-projects/spring-petclinic>

# 2. Infrastructure

Definition: The underlying environment, services, and system setup required to run an application.  
Purpose: Ensures the application runs reliably and efficiently.  
Examples:  
- A Linux server with Java installed to host a Spring Boot application.  
- Network, storage, runtime environments, databases, and cloud services.

# 3. Teams

- Code Team: Focuses on writing and maintaining application logic.  
- Infrastructure/Platform Team: Focuses on provisioning servers, setting up environments, CI/CD pipelines, and infrastructure as code.  
- QA/Testing Team: Ensures the quality and correctness of the application through automated and manual testing.

# 4. Common Code Repository Structure

my-application/  
├── src/  
│ ├── main/  
│ │ ├── java/ # Application source code  
│ │ └── resources/ # Config files, templates, etc.  
│ └── test/ # Unit and integration tests  
├── target/ # Build output (compiled code, JAR/WAR)  
├── pom.xml / build.gradle # Build tool configuration  
└── README.md  
  
- Code: The actual logic written by developers.  
- src/main: Contains the main application logic.  
- src/test: Contains unit and integration test cases.  
- unit test: Tests that validate small parts (functions, methods) of the application.  
- code coverage: A metric showing how much of the code is covered by tests.  
- target: Output directory after building the application (e.g., compiled JAR files).

# Quick Thoughts

## 1️⃣ Is it possible to have code and infrastructure in the same repository?

✅ Yes, absolutely.  
This approach is known as a monorepo, where both application code and Infrastructure as Code (IaC) (e.g., Terraform, Ansible, Helm charts) live together.  
  
Advantages:  
- Easier version control – code and infrastructure changes are tracked together.  
- Simplifies CI/CD pipelines – a single build process can handle both.  
  
Disadvantages:  
- Larger repo size.  
- More complex branching/permissions if teams work independently.  
  
Example structure:  
my-project/  
├── app/ # Application code  
│ └── src/...  
├── infra/ # Infrastructure as code (Terraform, scripts)  
│ └── main.tf  
└── ci-cd/ # Pipeline definitions

## 2️⃣ Why do we need to follow a structured approach to write code?

✅ A structured approach ensures:  
- Readability: Easier for other developers to understand and contribute.  
- Maintainability: Simplifies bug fixes, enhancements, and refactoring.  
- Scalability: Supports growth in complexity and team size.  
- Reusability: Encourages modular design and reuse of components.  
- CI/CD Integration: Build tools and pipelines rely on consistent structure.  
  
Example:  
In a Spring Boot project, src/main/java contains application code and src/test/java contains tests.   
Build tools like Maven and Gradle expect this structure for automatic compilation and testing.

## 3️⃣ Build Tools & Their Code Structure

|  |  |  |  |
| --- | --- | --- | --- |
| Build Tool | Language | Key File | Typical Project Structure |
| Maven | Java | pom.xml | src/main/java, src/test/java, target/ |
| Gradle | Java | build.gradle | Similar to Maven but more flexible scripting |
| npm / yarn | JavaScript / Node.js | package.json | src/, test/, dist/ |
| pip / setuptools | Python | setup.py | src/, tests/, build/ |
| msbuild | .NET | .csproj | src/, bin/, obj/ |

📌 Build tools handle tasks like:  
- Compiling source code  
- Running tests  
- Packaging into deployable artifacts (e.g., .jar, .war, .zip)  
- Managing dependencies

## 4️⃣ Source Code

- The original code written by developers in programming languages like Java, Python, or JavaScript.  
- Stored in version control systems (e.g., GitHub, Azure Repos, GitLab).  
- Forms the foundation of the application and is compiled or interpreted to create the final running software.

# 📌 Summary

- Application is the logic that solves a problem, while infrastructure is the environment it runs on.  
- A structured repository with src/main, src/test, and build outputs like target/ is crucial for quality and scalability.  
- Code and infrastructure can exist in the same repository for better collaboration.  
- Following a structured approach helps teams collaborate effectively and ensures smooth CI/CD workflows.  
- Build tools automate compilation, testing, packaging, and deployment, and each expects a defined structure.