**MVC (Model-View-Controller) and Its Variants**

**\*MVC (Model-View-Controller)\***

- \*Model\*: Manages data and business logic.

- \*View\*: Renders the user interface and displays data from the model.

- \*Controller\*: Handles user input, manipulates the model, and updates the view.

\*Variants:\*

**- \*MVP (Model-View-Presenter)\*:**

- \*Model\*: Same as MVC.

- \*View\*: Handles display logic and user interface.

- \*Presenter\*: Handles user input and updates the model and view.

**- \*MVVM (Model-View-View Model)\*:**

- \*Model\*: Same as MVC.

- \*View\*: Same as MVC.

- \*View Model\*: Binds the view and model, handling logic and state.

**- \*HMVC (Hierarchical Model-View-Controller)\*:**

- Extends MVC by organizing the application into a hierarchy of MVC triads.

- **\*PAC (Presentation-Abstraction-Control)\*:**

- \*Presentation\*: Manages user interface.

- \*Abstraction\*: Manages core functionality.

- \*Control\*: Manages communication between presentation and abstraction.

**Understanding Software Design Pattern**

\*Categories and Examples:\*

**1. \*Creational Patterns\*:**

- \*Singleton\*: Ensures one instance.

- \*Factory Method\*: Creates objects via interfaces.

- \*Abstract Factory\*: Creates families of related objects.

- \*Builder\*: Constructs complex objects step-by-step.

- \*Prototype\*: Creates new objects by copying existing ones.

**2. \*Structural Patterns\*:**

- \*Adapter\*: Interfaces compatibility.

- \*Composite\*: Tree structure for part-whole hierarchies.

- \*Decorator\*: Adds responsibilities dynamically.

- \*Facade\*: Simplifies complex subsystems.

- \*Flyweight\*: Reduces memory usage for similar objects.

- \*Proxy\*: Controls access to another object.

**3. \*Behavioral Patterns\*:**

- \*Chain of Responsibility\*: Passes requests along a chain.

- \*Command\*: Encapsulates requests as objects.

- \*Interpreter\*: Parses and interprets language grammar.

- \*Iterator\*: Sequential access to elements.

- \*Mediator\*: Defines interactions between objects.

- \*Memento\*: Captures and restores object state.

- \*Observer\*: Notifies dependents of state changes.

- \*State\*: Alters behavior based on state.

- \*Strategy\*: Encapsulates interchangeable algorithms.

- \*Template Method\*: Defines algorithm skeleton.

- \*Visitor\*: Adds operations to object structures.

**Cloud Computing and Services**

\*Basics of Cloud Computing\*

- \*Definition\*: Delivery of computing services over the internet.

- \*Types\*:

- \*Public Cloud\*: Services offered over the public internet.

- \*Private Cloud\*: Services maintained on a private network.

- \*Hybrid Cloud\*: Combines public and private clouds.

\*Cloud Service Models:\*

**- \*SaaS (Software as a Service)\*:**

- Delivers software over the internet.

- Examples: Google Workspace, Salesforce.

**- \*PaaS (Platform as a Service)\*:**

- Provides a platform for building, testing, and deploying applications.

- Examples: Google App Engine, Heroku.

**- \*IaaS (Infrastructure as a Service)\*:**

- Offers virtualized computing resources over the internet.

- Examples: AWS EC2, Microsoft Azure.

**Containerization with Docker and Kubernetes**

**1.\*Docker\*:** Docker is a platform that enables developers to create, deploy, and run applications in containers. Containers are lightweight, portable, and self-sufficient units that include everything needed to run the application, such as code, runtime, system tools, libraries, and settings.

**2. \*Containerization\*:** This is the process of packaging an application and its dependencies into a container. It ensures that the application runs consistently across different environments, from development to testing to production.

**3. \*Image Management\*:** Docker images are templates used to create containers. Managing images involves creating, storing, and distributing these images. It includes practices like versioning, tagging, and cleaning up unused images to save space.

**4. \*Docker Hub and Registries\*:** Docker Hub is a public registry where users can store and share Docker images. Registries can be public or private, and they provide a centralized location to manage and distribute container images.

**5. \*Kubernetes\*:** Kubernetes is an open-source platform designed to automate the deployment, scaling, and operation of containerized applications. It orchestrates containers, ensuring that they run efficiently and reliably across a cluster of machines.

**\*Docker\***

- \*Containerization\*: Packaging applications and their dependencies into containers.

- \*Image Management\*: Creating and managing Docker images.

- \*Docker Hub and Registries\*: Platforms for storing and sharing Docker images.

**\*Kubernetes\***

- \*Container Orchestration\*: Automating deployment, scaling, and management of containerized applications.

- \*Components\*: Nodes, Pods, Services, Deployments, etc.

- \*Capabilities\*:

- Scaling applications.

- Managing container lifecycles.

- Load balancing.

- Rolling updates and rollbacks.