**1. Source Code Management (SCM)**

Source Code Management (SCM) is a practice that involves the process of tracking and managing changes to software code. SCM tools enable teams to collaborate on code, track changes, revert to previous versions, and maintain version history.

**Key SCM Tools:**

1. **Git**:
   * The most widely used distributed version control system. It allows developers to track changes, branch, and merge code, and collaborate with others. Git can be hosted on platforms like GitHub, GitLab, and Bitbucket.
   * **GitHub**: A cloud-based Git repository hosting service that also provides collaboration features like pull requests, issue tracking, and more.
   * **GitLab**: A similar tool to GitHub, but it offers additional DevOps tools and integrated CI/CD pipelines.
   * **Bitbucket**: Another Git-based repository hosting platform with integration to Jira and other Atlassian products.
2. **Subversion (SVN)**:
   * A centralized version control system, less popular than Git but still used in some organizations, particularly legacy systems.
3. **Mercurial**:
   * Another distributed version control system, similar to Git, but with a simpler command structure.

**Common SCM Patterns:**

* **Branching Strategies**:
  + **Feature Branching**: Developers work on new features in isolated branches, which are later merged into the main branch.
  + **Git Flow**: A structured branching strategy where there are separate branches for features, releases, and hotfixes.
  + **Trunk-Based Development**: Developers work on the main branch (or trunk) directly, committing frequently to avoid large, conflicting merges.
* **Code Reviews**:
  + Pull requests (PRs) and merge requests (MRs) are used to review code before merging it into the main branch. Tools like GitHub and GitLab provide built-in features for PRs.
* **Versioning**:
  + Semantic Versioning (SemVer) is commonly used, which defines version numbers as Major.Minor.Patch (e.g., 1.2.3).

**2. Containers and Orchestration**

Containers are a way to package an application and its dependencies into a single, portable unit that can run anywhere. Container orchestration refers to the management and automation of containerized applications in production environments.

**Key Container Technologies:**

1. **Docker**:
   * The most popular containerization technology. Docker allows developers to create, test, and deploy applications in lightweight containers. It packages an application along with its dependencies and environment, ensuring consistency across different environments.
2. **Kubernetes**:
   * Kubernetes (K8s) is the most widely used container orchestration platform. It automates the deployment, scaling, and management of containerized applications.
   * **Core Concepts**:
     + **Pods**: A group of one or more containers that are deployed together.
     + **Services**: A way to expose and connect containers in a cluster.
     + **Namespaces**: Virtual clusters within a Kubernetes cluster to organize resources.
     + **Deployments**: Define the desired state for applications, ensuring they are maintained and updated in the cluster.
3. **Docker Compose**:
   * A tool for defining and running multi-container Docker applications. It allows you to configure the services, networks, and volumes in a YAML file and deploy them with a single command.
4. **OpenShift**:
   * An enterprise version of Kubernetes that provides additional features for security, compliance, and management.

**Common Container and Orchestration Patterns:**

* **Microservices Architecture**:
  + Containers are commonly used to deploy individual microservices, each running in its own container, allowing for easy scaling, management, and isolation.
* **Service Discovery**:
  + Containers in a Kubernetes environment use service discovery to find and communicate with each other. Kubernetes handles this automatically by creating a DNS record for each service.
* **Load Balancing**:
  + Kubernetes automatically balances the load across containers in a service, ensuring that no single container is overwhelmed.
* **Auto-Scaling**:
  + Kubernetes can scale the number of replicas of a service based on load, ensuring high availability and efficient resource usage.
* **Continuous Deployment (CD)**:
  + Containers are often used in a CD pipeline where a new version of the application is deployed automatically to a staging or production environment once it has passed automated tests.

**3. Cloud Computing**

Cloud computing refers to the delivery of computing services (servers, storage, databases, networking, software, etc.) over the internet. Companies can use cloud services to avoid managing and maintaining physical infrastructure.

**Key Cloud Providers:**

1. **Amazon Web Services (AWS)**:
   * The leading cloud provider offering a wide array of services such as EC2 (compute), S3 (storage), RDS (databases), and Lambda (serverless computing).
2. **Microsoft Azure**:
   * Microsoft's cloud platform offering similar services to AWS, with strong integrations with Windows-based infrastructure and applications.
3. **Google Cloud Platform (GCP)**:
   * Google’s cloud offering, well known for its machine learning, data analytics, and Kubernetes services.
4. **IBM Cloud**:
   * IBM’s platform with a focus on AI, data analytics, and enterprise applications.

**Common Cloud Computing Patterns:**

* **Infrastructure as a Service (IaaS)**:
  + Cloud providers offer virtual machines (VMs) and storage, and you manage the operating systems and applications. Examples include AWS EC2 and Azure Virtual Machines.
* **Platform as a Service (PaaS)**:
  + Cloud providers manage most of the infrastructure for you, including runtime, databases, and more. Examples include AWS Elastic Beanstalk and Google App Engine.
* **Software as a Service (SaaS)**:
  + Fully managed software solutions hosted on the cloud, where you access them via a web browser. Examples include Google Workspace, Office 365, and Salesforce.
* **Serverless Computing**:
  + With serverless, you write code and deploy it without managing the underlying infrastructure. Examples include AWS Lambda, Azure Functions, and Google Cloud Functions.
* **Hybrid Cloud**:
  + A combination of on-premises infrastructure and cloud services, allowing data and applications to be shared between them.

**4. Infrastructure as Code (IaC)**

Infrastructure as Code (IaC) is the practice of managing and provisioning computing infrastructure through machine-readable script files, rather than through manual processes. IaC allows developers to automate infrastructure management, version control, and deployment.

**Key IaC Tools:**

1. **Terraform**:
   * An open-source IaC tool that allows you to define infrastructure (networks, virtual machines, etc.) using a declarative configuration language. It supports multiple cloud providers, including AWS, Azure, and Google Cloud.
2. **Ansible**:
   * A configuration management tool that automates tasks like server provisioning, application deployment, and configuration. It uses YAML to define playbooks for automating tasks.
3. **CloudFormation**:
   * AWS’s native IaC tool, which uses JSON or YAML templates to define and provision AWS infrastructure.
4. **Puppet**:
   * An open-source tool used for automating the deployment and management of infrastructure and applications. It works well for large-scale infrastructure automation.
5. **Chef**:
   * An automation platform that manages infrastructure via code. Chef uses Ruby-based DSLs for writing configurations.

**Common IaC Patterns:**

* **Declarative Configuration**:
  + You describe the desired state of your infrastructure (e.g., "I want three EC2 instances"), and the IaC tool ensures that the infrastructure matches this state.
* **Immutable Infrastructure**:
  + Instead of modifying existing servers, new versions of servers are created and deployed with updated configurations. This ensures consistency and reduces the risk of errors.
* **Versioned Infrastructure**:
  + The infrastructure configuration is stored in version-controlled files (like Git), which ensures that all changes to infrastructure can be tracked, reviewed, and rolled back if necessary.
* **Continuous Integration with IaC**:
  + IaC configurations are tested as part of the CI pipeline to ensure that changes to infrastructure do not break any dependencies or configurations.

**5. Common Architecture and Deployment Patterns in IT Companies**

**Microservices Pattern:**

* Microservices are small, independent services that communicate via APIs. This approach is often used in cloud-native applications where each service runs in its own container. CI/CD, containerization (Docker), and orchestration (Kubernetes) are key components in this pattern.

**Monolithic Pattern:**

* A single, unified codebase and deployment unit. This traditional approach can work well for smaller applications, but larger applications tend to struggle with scaling and deploying frequently.

**Event-Driven Architecture:**

* A design where services communicate by sending and listening to events. This is commonly used in microservices where asynchronous communication is preferred.

**Blue-Green Deployment:**

* A deployment strategy where two identical production environments are maintained. The "Blue" environment serves live traffic, and the "Green" environment is for the new version of the application. Once tested, traffic is switched to the green environment.

**Canary Deployment:**

* A strategy where a small subset of users gets the new version of the software before a full-scale release, minimizing risk.

**Serverless Architecture:**

* Serverless applications run in stateless environments, often in the cloud (AWS Lambda, Azure Functions). This eliminates the need to manage servers while still providing scalability.

**Conclusion**

These patterns and tools play a crucial role in modern software development and operations. SCM helps teams to manage code collaboration, containers and orchestration provide flexibility and scalability in deployment, cloud services offer scalable infrastructure, and IaC allows automation and consistency. By understanding and using these patterns and tools, IT companies can effectively manage large-scale systems, ensure reliability, and deliver software faster and more efficiently.