1) - Developers write code and push it to a common shared GIT repository in SCM.
This automatically triggers a new pipeline using a CI/CD tool such as Jenkins / Gitlab CI.

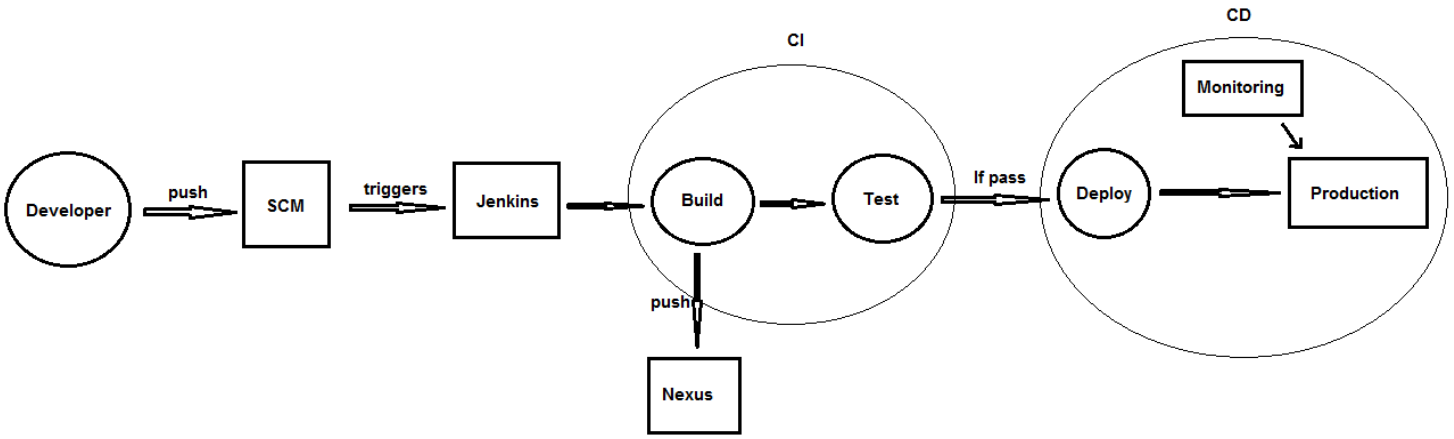
The pipeline stages are as follows:

- Build: Fetches the source code components from the repo, compiles it, and builds an artifact, whether an executable file or a container image. The artifact is stored in an artifact registry such as Nexus / Artifactory.

- Test: Various types of tests are applied to the build to check its stability and integrity. In case of failure, an error information is sent back to the development team for investigation and remediation.

- Deploy: Once tests are passed successfully, deploys the artifact to the target environment on the cloud (dev / test / prod). The automation first provisions the cloud resources as part of it, then deploys the application on them.

Continuous monitoring and automated security measures (E.g. Prometheus) are employed in the production environment to maintain the integrity of the application.



2) The application is containerized and needs to be dynamically scaled as well, hence we'll use **Kubernetes** for hosting it.
As a non-containerized database is required, we'll need to host it on a separate server.
We host the application on the cloud, so we'll utilize cloud native tools for managing both Kubernetes and the database.

Suppose that we use AWS, then:

- For Kubernetes use **EKS**
- For the database use **RDS** (or DynamoDB for NoSQL)
- Route incoming traffic to the application using **Application Load Balancer (ALB)**

Auto-scaling:

- Pod auto-scaling: Use the **Horizontal Pod Autoscaler (HPA)** in Kubernetes to adjust the number of pod replicas of an application based on CPU, memory usage, or other metrics.
- Cluster auto-scaling: Use the **Cluster Autoscaler (CA)** to adjust the number of nodes in the cluster.
  Another option for cluster autoscaling is **Karpenter**, which is a good tool for handling this task with some advantages over CA.

Use an IaC tool to create the cloud resources. It can either be **Terraform**, or a cloud-native tool such as **CloudFormation**.

Use Jenkins pipeline to automate the deployment process. It'll be responsible for running the IaC code and creating the cloud resources.

Once infrastructure is ready, use **ArgoCD** to deploy the application on top.

3) Create **Public Subnets** to host public-facing resources.
Create **Private Subnets** to host internal components.
Create an **Internet Gateway** to allow communication between the resources in the VPC and the internet.
The Internet Gateway will allow the resources in the public subnet to access the internet.
Resources in the private subnet can connect to the internet as necessary using a **NAT Gateway** in the public subnet.
To use AWS PrivateLink, create a **VPC endpoint** in the VPC for each one of the required AWS services, specifying the name of the service and the source subnet.