Concepts CI/CD platforms

CircleCI concepts

Feature	Explanation	Equivalent in Metamodel
Versioning	Specifies the CircleCI configuration version.	Represented as an environment variable in the `Pipeline` class.
Executors	Defines the environment in which jobs run (e.g., Docker, machine, Windows, macOS).	Represented as a `Tool` in the metamodel.
Jobs	Represents individual tasks executed within the CI/CD pipeline.	Defined as `Jobs` within the `Pipeline` class.
Workflows	Manages the order and dependencies of job executions.	Dependencies in workflows are captured using the `depends` attribute in the `Job` class.
Steps	Defines specific actions within a job (e.g., checkout, run, save artifacts).	Represented as `Commands` and `Command Parameters`.
Orbs	Reusable packages that contain pre-defined CircleCI configurations.	Represented as a `Tool` in the metamodel.
Caching	Stores dependencies to speed up builds and avoid redundant installations.	Not currently supported in our metamodel but could be represented as an environment variable.
Parallelism	Runs multiple instances of a job simultaneously to improve performance.	The `Pipeline` class includes a `concurrent` attribute for parallel execution, with dependencies managed through the `depends` attribute in the `Job` class.
Docker Support	Uses Docker images to create consistent build environments.	Stored as a `Tool` in the metamodel.
Deployment	Defines the conditions and methods for deploying the application.	Represented as a `When` instance.

Feature	Explanation	Equivalent in Metamodel
Triggers	Specifies when workflows should trigger (e.g., on commits, Git tags, or scheduled runs).	Represented as a `When` instance.
Filters	Restricts jobs to specific branches or tags.	Not explicitly modeled in our metamodel, but could be represented within the Job's environment.
Approval Jobs	Requires manual approval before executing certain jobs, commonly used in deployment workflows.	Not explicitly modeled in our metamodel, but could be represented within the Job's environment.
Environment Variables	Stores configuration values and secrets that jobs can access.	Represented as `Environment Variables` in the metamodel.
Context	Provides secure, reusable environment variables shared across projects.	Not explicitly supported, but could be represented within the Job's environment.
Shell Modifications	Specifies a custom shell to be used for job execution.	Represented as a `Tool` instance associated with the `Job` class.
Service Containers (Sidecars)	Runs additional services (e.g., databases, caching layers) alongside jobs.	Represented as a `Tool` instance associated with the `Job` class.
Machine Executors	Runs jobs in full virtual machines instead of containers.	Represented as a `Tool` instance associated with the `Job` class.
Windows & macOS Support	Supports native Windows and macOS environments for job execution.	Represented as a `Tool` instance associated with the `Job` class.
Matrix Jobs	Dynamically generates multiple job variations (e.g., testing across different Node.js versions).	Represented as a `Tool` instance associated with the `Job` class.
Dynamic Configuration	Allows the `.circleci/config.yml` file to be generated at runtime for flexible workflows.	Not explicitly supported but could be represented within the Job's environment.

Feature	Explanation	Equivalent in Metamodel
Artifacts	Saves and stores files (e.g., logs, test reports, build outputs) for later retrieval.	Represented as an `Artifact` instance associated with the `output` attribute of the `Job` class.
Commands	Reusable blocks of steps that can be invoked within jobs.	Steps are represented as instances of the `Command` class, and references to these instances are used to repeat commands across different jobs.
Parameters	Allows jobs, commands, and workflows to be parameterized for dynamic behavior.	Represented as `Environment Variables` for `Pipeline` and `Job` classes.
Resource Classes	Defines resource allocations (e.g., CPU, memory) for jobs.	Defined as `Tool` instances for the `Pipeline` or `Job` classes.
Retry Logic	Configures retry conditions for failed steps or jobs.	Not natively supported in our metamodel but could be implemented at the parser/generator level, using a `Parameter` of a `Command`.

Travis CI concepts

Feature	Explanation	Equivalent in Metamodel
General Configuration	Defines project-wide settings for Travis CI builds, such as language, operating system, distribution, architecture, and virtual environment.	Represented as a `Tool` instance in the `Pipeline` class.
Build Lifecycle Hooks	Manages the different build phases (e.g., `before_install`, `install`, `before_script`, `script`, `after_script`, and `deploy`).	Represented as `Job` instances.
Job Control	Defines how jobs are named, defined, and executed, including conditional execution and dependencies between jobs.	Conditions are represented as `If` instances within `Job`s, and dependencies are represented as references to other jobs.

Feature	Explanation	Equivalent in Metamodel
Stages	Groups jobs into sequential phases (e.g., `build`, `test`, `deploy`), allowing parallel execution within each stage.	Stages are mapped to `Job` instances with dependencies on other `Job` instances. For example, the `build` job depends on the `test` job having completed.
Matrix (Job Matrix)	Enables running multiple variations of jobs in parallel with different configurations (e.g., testing multiple language versions).	Mapped to `Tool` instances within `Job` calls.
Environment Variables	Defines and manages environment variables (both public and encrypted) that are available to builds and jobs.	Mapped to `Environment` instances in the `Job` class.
Caching	Stores dependencies and build artifacts between builds to speed up subsequent runs.	The metamodel doesn't have this feature natively, but it can be mapped to an `Environment` instance.
Services	Specifies external services (e.g., databases, Docker, Redis) that are started alongside the build environment.	Mapped to `Tool` instances within both the `Pipeline` and `Job` classes.
Deployment	Configures how and where the application is deployed, including integration with external deployment providers.	Mapped to the `When` instance in the `Pipeline`.
Notifications	Sends build status alerts via various channels (e.g., email, Slack, webhooks) when builds succeed or fail.	The metamodel doesn't have this feature natively, but it can be mapped to an `Environment` instance.
Artifact Handling	Manages build artifacts by caching them or deploying them to external storage services (e.g., S3, GitHub Releases).	Represented as `Artifacts` instances as inputs or outputs of the `Job` class.
Security & Encryption	Handles sensitive data by encrypting environment variables and managing secure access credentials.	Represented as `Environment` instances.

Feature	Explanation	Equivalent in Metamodel
Conditions & Filters	Defines rules (using conditions such as branch or tag filters) to control when jobs or stages should run.	Represented as `If` instances in the `Job` class.
Config Imports	Allows importing shared or external configuration snippets into the primary `.travis.yml` file for reuse and modularity.	The metamodel doesn't support this feature.
Infrastructure Settings	Configures build environment resources such as VM instance size, virtualization type, and CPU architecture details.	Mapped to `Tool` instances in the `Job` and `Pipeline` classes.

Jenkins concepts

Feature	Explanation	Equivalent in Our Metamodel
Pipeline Definition	Defines the structure of the pipeline, which can be declarative or scripted.	Represented as a `Pipeline` instance.
Stages and Steps	Organizes the workflow into stages, with steps defining the actions within those stages.	Stages are mapped to `Job` instances, with steps mapped to `Command` instances.
Agents and Nodes	Specifies the machine or environment where the pipeline runs.	Mapped to `Tool` instances in the `Pipeline` and `Job` classes.
Triggers	Automatically triggers the pipeline based on events such as SCM changes.	Represented as `When` instances.
Environment Variables	Defines environment variables for the pipeline, which can be custom or built-in.	Represented as `Environment` instances in the `Pipeline` and `Job` classes.
Parallel Execution	Executes multiple stages or steps concurrently.	Represented by multiple `Job` instances, with dependencies managed via references.
Conditionals & Flow Control	Controls the execution flow using conditions such as `if`, `when`, and `try-catch`.	Conditions are represented as `If` instances within the `Job` class.

Feature	Explanation	Equivalent in Our Metamodel
Input & Approvals	Pauses the pipeline to wait for user input or approval.	Not supported in the metamodel.
Error Handling & Retry	Defines how the pipeline should handle errors and retries.	Mapped to `If` instances for retry logic and error handling within `Job`.
Post Actions	Defines actions to perform after the pipeline completes, such as notifications for success or failure.	Represented as `Job` instances with dependencies on all other jobs.
Integrations	Integrates with external tools and systems, such as SCM, Docker, and notifications.	Mapped to `Command` instances.
Library and Shared Functions	Allows the reuse of libraries or shared functions for common tasks.	Not supported in the metamodel.
Artifact Management	Stores and archives build artifacts for later use, such as with the `archiveArtifacts` step.	Represented as `Artifacts` instances, either as inputs or outputs of the `Job` class.
Secrets and Credentials	Securely manages sensitive data such as API keys and passwords, often using constructs like `withCredentials`.	Represented as `Environment` instances.
Timeouts	Ensures pipeline steps do not run indefinitely, for example, with the `timeout(time: 10, unit: 'MINUTES')` step.	Mapped to `Environment` instances in the `Job` class.
Matrix Builds	Allows testing multiple configurations in parallel, typically using the `matrix { ... }` directive.	Mapped to `Tool` instances within `Job` calls.
Logging & Debugging	Enables detailed logging for debugging, such as using `echo` commands or shell debugging flags.	Not supported.

Feature	Explanation	Equivalent in Our Metamodel
Workspace Management	Manages files and directories in the workspace, using commands such as `deleteDir()`, `stash`, and `unstash`.	Not natively supported in the metamodel, but can be mapped to `Environment` instances in the `Pipeline` class.
Parameters	Defines build parameters that can be provided by users before the build, allowing for customizable runs.	The metamodel does not natively support this feature, but the values can be replaced with the parameter values in the `Command` instances.
Options	Provides pipeline-level configurations, such as build discarder, timestamps, and other settings.	Not directly supported, but can be represented as `Environment` instances.
Tools Configuration	Specifies the tools (e.g., JDK, Maven) required during pipeline execution, ensuring the correct versions are available.	Mapped to `Tool` instances in the `Job` and `Pipeline` classes.

GitLab CI/Cd

Feature	Explanation	Equivalent in Metamodel (Travis CI)
Pipeline Stages	Defines sequential stages (e.g., build, test, deploy) that group jobs and control their execution order.	Mapped to `Job` instances with dependencies.
Jobs	Individual tasks executed as part of the pipeline.	Represented as `Job` instances.
Scripts	Commands or shell scripts executed within each job.	Encapsulated within each `Job` instance's `Command` instances.
Artifacts	Files or directories preserved after a job and made available to later stages.	Represented as `Artifacts` instances as inputs or outputs of the `Job` class.
Cache	Caches files or directories between builds to speed up subsequent runs.	Not natively modeled; can be mapped to an `Environment` instance.
Image Selection	Specifies the Docker image in which a job runs.	Represented as a `Tool` instance in the `Pipeline` or `Job` class.

Feature	Explanation	Equivalent in Metamodel (Travis CI)
Before & After Scripts	Extra commands that run before or after the main job script (used for setup and cleanup).	Represented as `Job` instances.
Only & Except Rules	Conditions based on Git references (branches/tags) that control job execution.	Represented as `If` instances within `Job` instances.
When Conditions	Specifies when a job should run (`on_success` , `on_failure` , `manual` , `always`).	Represented as `If` instances.
Variables	Defines environment variables (global or job-specific) available during job execution.	Mapped to `Environment` instances in the `Pipeline` and `Job` classes.
Dependencies	Explicitly defines job dependencies and specifies which artifacts are passed between jobs.	Represented as references to other `Job` instances.
Retry & Timeout	Configures the number of retry attempts for a job and how long it can run before timing out.	Not explicitly modeled in the metamodel; could be treated as an additional `Job Environment` .
Parallel Execution	Enables running multiple job variations concurrently (e.g., using a matrix).	Mapped to `Tool` instances within `Job` .
Include External YAML	Allows reusing configurations by including external YAML files.	Not supported in the metamodel.
Triggering Other Pipelines	Triggers a downstream (child or external) pipeline as part of a job's execution.	Not natively supported in the metamodel.
Allow Failure	Permits a job to fail without marking the entire pipeline as failed.	Represented as a `Job Environment` instance.
Needs	Specifies explicit job dependencies, allowing jobs to run out of order when required.	Mapped to `Job` dependencies.
Rules	Advanced conditionals to determine whether a job should run.	Represented as `IfThenElse` instances in the `Job` class.

Feature	Explanation	Equivalent in Metamodel (Travis CI)
Workflow	Controls overall pipeline behavior and conditionally creates pipelines.	Mapped to `IfThenElse` instances inside the `Job` class.
Extends	Enables jobs to inherit configuration from templates (hidden jobs).	Not supported in the metamodel.
Default	Sets global defaults for job configurations, which can be overridden by individual jobs.	Represented as a `Tool` instance in the `Pipeline` class.
Services	Defines auxiliary Docker containers (e.g., databases) that run alongside jobs.	Mapped to `Tool` instances within both the `Pipeline` and `Job` classes.
Secrets	Injects sensitive data from external secret managers into jobs securely.	Represented as `Environment` instances in both the `Pipeline` and `Job` classes.
Environment	Specifies deployment targets (e.g., name, URL, stop conditions) for jobs that perform deployments.	Mapped to `Environment` instances in the `Job` class.
ID Tokens	Generates JSON Web Tokens (JWTs) for authentication with third-party services.	Not natively supported in the metamodel; can be mapped to an `Environment`.
Hidden Jobs (Templates)	Uses dot-prefixed job definitions that act as templates and are not executed directly.	Hidden jobs are duplicated into the `Job` instances that use them.

GitHub Actions

Feature	Description	Metamodel Equivalent
Name	Specifies the name of the workflow for easier identification.	Represented as the `Name` attribute in the `Pipeline` instance.
Triggers (on)	Specifies when the workflow should run (e.g., `push`, `pull_request`, `schedule`).	Represented as the `When` instance.
Jobs	Defines a set of tasks to be executed within the workflow.	Represented as `Job` instances.

Feature	Description	Metamodel Equivalent
Job Dependencies (needs)	Specifies that a job should only run after one or more other jobs complete.	Represented as dependency references within `Job` instances.
Job Runners (runs-on)	Specifies the environment (e.g., Ubuntu, Windows, macOS) where the job will run.	Mapped to `Tool` instances within the `Job` and `Pipeline` classes.
Steps	Defines the sequence of tasks within a job.	Represented as individual commands within a `Job`.
Actions (uses)	Calls reusable components from GitHub Marketplace or custom actions.	Represented as `Tool` instances within the `Pipeline` and `Job` classes.
Commands (run)	Executes shell commands inside a step.	Represented as `Command` instances within the `Job` class.
Environment Variables (env)	Stores and accesses variables within the workflow.	Mapped to `Environment` instances within the `Pipeline` and `Job` classes.
Secrets	Stores sensitive credentials securely (e.g., API keys).	Represented as `Environment` instances within the `Pipeline` and `Job` classes.
Matrix Builds	Runs the same job with different configurations (e.g., different OS versions, dependency versions).	Mapped to `Tool` instances within the `Pipeline` and `Job` classes.
Conditionals (if)	Executes steps or jobs based on specific conditions.	Represented as `IfThenElse` instances within the `Job` class.
Artifacts (upload/download)	Saves or retrieves files between jobs (e.g., compiled binaries, test reports).	Represented as `Artifacts` instances as inputs or outputs of the `Job` class.
Caching	Speeds up workflows by storing dependencies (e.g., `node_modules`, pip cache).	Not natively supported in the metamodel, but can be mapped to an `Environment` instance.
Outputs	Saves values from a step to be used in later steps or jobs.	`Artifacts` instances as outputs of the `Job` class.
Services	Runs dependencies such as databases (e.g., PostgreSQL, Redis) during the workflow.	Mapped to `Tool` instances within both the `Pipeline` and `Job` classes.

Feature	Description	Metamodel Equivalent
Deployments	Automates deployment processes (e.g., to production or staging).	Mapped to the `When` instance within the `Pipeline` class.
Concurrency	Controls how many instances of a workflow can run simultaneously to avoid race conditions.	The metamodel does not control the number of instances but sets the `concurrent` attribute in the `Pipeline` instance as `True`.
Timeouts (timeout-minutes)	Defines a maximum runtime for jobs to prevent hanging executions.	Not explicitly supported in the metamodel.
Error Handling (continue-on-error)	Allows a step to continue even if it fails, useful for non-critical tasks.	Not native to the metamodel but can be represented as a `Parameter` instance within the `Command` class.
Permissions (permissions)	Defines access levels for the workflow (e.g., read/write permissions for repositories).	Represented as a `Permission` instance.
Reusing Workflows (workflow_call)	Enables one workflow to call another, promoting modularity in workflow design.	Not supported in the metamodel.
Manual Triggers (workflow_dispatch)	Allows workflows to be manually triggered via the GitHub UI or API.	Represented as a `When` instance.
Reusable Actions (composite actions)	Defines custom reusable actions within the repository.	Not supported in the metamodel; must be defined multiple times within the `Job` instances.
Job Outputs	Allows a job to produce outputs that other jobs can use.	Defined as a `Job` `Artifact` instance output.
Expressions (${{ }})	Uses dynamic values in conditions, environment variables, and outputs.	Represented as `IfThenElse` instances inside the `Job` class.
Defaults	Provides default settings for steps (such as the default shell or working directory), reducing repetition in the workflow file.	Defaults are equivalent to parameters represented as "with" for actions, so we add default actions to the `Command` instances where they are used.