CircleCI Cheat Sheet

A comprehensive cheat sheet for CircleCI, covering configuration, commands, and best practices to streamline your CI/CD workflows.

Configuration Basics

`.circleci/config.yml` Structure

The `config.yml` file is the heart of your CircleCI configuration. It defines workflows, jobs, and steps.

Key Components:

- `version`: Specifies the CircleCI configuration version.
- `orbs`: Reusable packages of configuration.
- `jobs`: Collection of steps.
- `workflows`: Define how jobs are executed.

Example:

```yaml
version: 2.1
orbs:
  heroku: circleci/heroku@1.2.6
jobs:
  build:
    docker:
      - image: cimg/node:16.13
    steps:
      - checkout
      - run: npm install
workflows:
  build_and_deploy:
    jobs:
      - build
```

Common Configuration Keys

`version`	Specifies the version of the CircleCI configuration language. Currently `2.1` is recommended.
`jobs`	Defines individual tasks to be executed. Jobs contain steps.
`workflows`	Orchestrates the execution of jobs, defining dependencies and sequencing.
`steps`	A list of commands or pre-defined actions executed within a job.
`docker`	Specifies the Docker image to use for the job's execution environment.
`checkout`	A special step that clones your repository into the workspace.

Workflow Configuration

Workflows define how jobs are executed and orchestrated. Key features include:

- **Sequencing:** Define the order in which jobs run.
- **Dependencies:** Specify that a job should only run after another job completes successfully.
- **Filters:** Control when a job runs based on branches, tags, or other conditions.

Example:

```yaml
workflows:
  build_and_deploy:
    jobs:
      - build
      - deploy: # job name
          requires: # specifies the dependent jobs
            - build
          filters:
            branches:
              only: main
```

Common Steps

Built-in Steps

`checkout`	Clones the repository into the workspace. Should be the first step in most jobs.
`run`	Executes shell commands. The most versatile step.
`save_cache`	Saves files or directories to the cache for reuse in subsequent jobs.
`restore_cache`	Restores files or directories from the cache.
`store_artifacts`	Uploads artifacts (e.g., test reports, binaries) to CircleCI for storage and retrieval.
`persist_to_workspace`	Saves data to a workspace that can be accessed by subsequent jobs in the workflow.
`attach_workspace`	Attaches the workspace to the current job.

Using the `run` Step

The `run` step executes shell commands.

Common Attributes:

- `name`: A descriptive name for the step.
- `command`: The shell command to execute.
- `shell`: The shell to use (e.g., `bash`, `sh`).
- `working_directory`: The directory in which to execute the command.

Example:

```yaml
steps:
  - run:
      name: Install Dependencies
      command: npm install
      working_directory: ./frontend
```

Caching

Caching is crucial for speeding up builds. Use `save_cache` and `restore_cache` to cache dependencies and other frequently used files.

Best Practices:

- Use a unique cache key based on dependency file hashes (e.g., `package-lock.json`).
- Cache dependencies and build artifacts.
- Invalidate the cache when dependencies change.

Example:

```
steps:
  - restore_cache:
      keys:
        - v1-dependencies-{{ checksum "package-lock.json" }}
        - v1-dependencies
  - run:
      name: Install Dependencies
      command: npm install
  - save_cache:
      paths:
        - node_modules
      key: v1-dependencies-{{ checksum "package-lock.json" }}
```

Orbs and Integrations

Using Orbs

Orbs are reusable packages of CircleCI configuration. They simplify configuration and enable integration with third-party services.

Benefits:

- Reduced configuration duplication.
- Easy integration with popular tools and services.
- Community-maintained and verified orbs.

Example: Using the Heroku Orb

```
version: 2.1
orbs:
  heroku: circleci/heroku@1.2.6
workflows:
  deploy:
    jobs:
      - heroku/deploy:
          app-name: my-app
          requires:
            - build
```

Common Orbs

`circleci/heroku`	For deploying to Heroku.
`circleci/aws-s3`	For interacting with AWS S3 buckets.
`circleci/slack`	For sending notifications to Slack channels.
`circleci/docker-compose`	For running Docker Compose commands.
`fastly/fastly`	For interacting with the Fastly CDN.
`browser-tools/selenium-orb`	For UI/browser testing

Integrating with Services

CircleCI integrates with numerous services through orbs and custom configurations.

Examples:

- **AWS:** Deploy to EC2, Lambda, or S3 using the AWS CLI or orbs.
- **Google Cloud:** Deploy to Google Cloud Platform using `gcloud` commands.
- **Slack:** Send build status notifications to Slack channels.
- **Docker Hub:** Build and push Docker images to Docker Hub.
- **NPM/Maven/etc:** Publish packages to package registries

Advanced Features

Environment Variables

Environment variables are used to store sensitive information (e.g., API keys, passwords) and configure builds.

Setting Environment Variables:

- In the CircleCI web interface (Project Settings -> Environment Variables).
- Dynamically in the `config.yml` file using workflow parameters.

Accessing Environment Variables:

- Use the `$VAR_NAME` syntax in `run` commands.

Example:

```
steps:
  - run:
      name: Deploy to Production
      command: |
        ./deploy.sh $PRODUCTION_API_KEY
```

Workflow Parameters

`parameters`	Define parameters at the workflow level to customize job execution. These parameters can be passed to jobs.
`types`	Specify the type of parameter (e.g., `string`, `boolean`, `integer`, `enum`).
`default`	Set a default value for the parameter.
`description`	Add descriptive text to the parameter, to help users understand the expected value

Using SSH

Connect to remote servers using SSH. Useful for deploying applications or running remote commands. The `add_ssh_keys` step is used to add SSH keys to the environment.

Generating SSH Keys:

- Generate a new SSH key pair using `ssh-keygen`.
- Add the private key to CircleCI as an environment variable.
- Add the public key to the remote server's `authorized_keys` file.

Example:

```
steps:
  - add_ssh_keys:
      fingerprints:
        - "your_ssh_key_fingerprint"
  - run:
      name: Deploy via SSH
      command: ssh user@host
'./deploy.sh'
```