Speeding up Pipeline Runs

Overview

Pipelines provides several ways to optimize your pipeline runs to make them execute faster. This helps in mitigating performance bottlenecks, avoiding build failures, and increasing build speed.

This section outlines all the options available for making your pipelines more efficient.

Page Contents

Caching Step Runtimes

Caching, which is performed through utility functions (`add_cache_files` and `reset_cache_files`), helps you speed up the execution of a step by preserving and restoring packages and dependencies between the runs of that step. Avoiding repeating the installation or loading of large dependencies every time the step is run helps in reducing build times. In addition, native steps perform caching as needed and always execute as fast as possible.

For more information, see Caching Step Runtimes.

Caching Dynamic Nodes

Node caching is useful when the same runtime is needed for a series of steps, and the same large set of dependencies needs to be installed. Node caching, which is available for dynamic node pools, helps speed up the execution of your builds and saves time by preserving the runtime environment for subsequent steps to execute in. For example, Docker layers can be cached, leading to faster docker builds since those layers don't need to be rebuilt each time.

When adding a dynamic node pool, node caching can be enabled by clicking the **Enable Cache** check box. For more information, see Dynamic Node Caching.

Choosing Node Pools

Another, often-overlooked, option for speeding up your pipeline runs is to create build node pools and run each step on a specific pool. This enables resource-intensive steps to run on larger nodes, speeding up execution.

Node pools can bet set using the Pipelines DSL. For more information, see Choosing Node Pools.

Using Artifactory Remote Repositories

Steps can use Artifactory remote repositories for caching packages that are needed repeatedly during execution, which helps in significantly reducing execution time. Remote repositories in Artifactory serve as a caching proxy for a repository managed at a remote URL.

For more information, see Remote Repositories and Cache Settings.

Using Custom Runtime Image

Using custom image gives you better control over what is installed on the runtime images. It can also speed up step execution since you can pre-install dependencies into the image.

For more information, see Using a Custom Runtime Image.

Using Matrix Step

Another option to reduce total execution time is to use the Matrix step. The Matrix step enables your pipeline to repeatedly execute the same set of actions in a variety of configurations and runtime environments, with each variant executing as an independent "steplet." These steplets can, when configured, execute in parallel on multiple build nodes. On completion of all steplets, Pipelines aggregates the result status, giving the appearance of a single step.

For more information, see Matrix and Using the Matrix Step.

Running Steps in Parallel

Running steps in parallel is a great way to reduce your build's running time, especially when you have a large test suite. Before doing this, however, you must first check which steps must run sequentially. All the other steps can now run in parallel to speed up pipeline execution. For more information, see Breaking Your Pipelines into Steps.

This can be tuned further by combining it with the `priority` tag, which can yield even better results. The `priority` tag, which can be set for any step, controls the priority of a step when there are parallel steps in a pipeline or multiple pipelines executing. For more information, see Bash Tags.