

**Introduction to Bazel**

**Course Number:** BZL-100
**Duration:** 1 day

**Overview**

[Bazel](https://bazel.build/) is an open-source variant of Blaze, Google’s powerful internal build tool. Is this tool right for your organization?

Accelebrate's Introduction to Bazel training explores the benefits, use cases, and fundamental concepts of Bazel using concrete Java-based of Bazel’s syntax and functionality. The class also introduces Bazel's advanced features, including remote caching and execution. After taking this class, students are able to implement this versatile, polyglot build tool in their projects.

**Prerequisites**

* User-level experience with build system concept and tools
* A basic understanding of the Java programming language
* Hands-on experience with common Java frameworks and dependencies

**Materials**

All attendees receive a copy of the instructor’s presentation and related code.

**Software Needed on Each Student PC**

* An installation of Bazelisk ([instructions](https://github.com/bazelbuild/bazelisk))
* An existing GitHub account ([instructions](https://docs.github.com/en/get-started/signing-up-for-github/signing-up-for-a-new-github-account))
* An installation of OpenJDK 11 ([instructions](https://openjdk.java.net/install/))

**Objectives**

* Understand basic Bazel concepts
* Automate typical tasks of a Java project

**Outline**

* Introduction to Bazel
	+ What is Bazel?
	+ Why Should I Use It?
		- Declarative Language
		- Reproducibility
		- Incremental Builds
		- Scalability
		- Parallel and Distributed Execution
		- Building Polyglot Projects
		- Extensibility
	+ Installing the Bazel Runtime + Launcher
		- Runtime Installation Options
		- Using Bazelisk
	+ Installing the Bazel Runtime
	+ Project Structure and Basic Building Blocks
		- Project Building Blocks
		- Project Structure
		- Build Logic Concepts
	+ Invoking a Target From the Command Line
		- Executing a Target
		- Commonly-Used Commands
		- Building the Whole Project
		- Output and Cache Directories
	+ Bazel Quickstart
	+ The Lifecycle of a Bazel Build
		- Lifecycle Phases
		- Bazel Configuration File
		- Programming Language Rules
* Basic Automation for a Java Project
	+ Exploring Java Rules
	+ Setting up a Java Project and Building It
		- Sample Java Project
		- Modeling the Binary Package
		- Standard Industry Conventions
		- Modeling the Workspace
		- Running the Application
	+ Running the Build from the Command Line
	+ Building a “Hello World” Java Project
	+ Inspecting the Generated Artifact
	+ Launching and running the Java Project
	+ Driving Bazel from the IDE
		- IntelliJ Plugin
		- VSCode Extension
		- Eclipse Plugin
	+ Using Bazel in IntelliJ
* Dependency Management and Automated Testing
	+ Modeling Fine-Grained Package Granularity and Dependencies
		- Types of Dependencies
		- Multi-Package Project
		- Modeling the Library Package
		- Package Dependencies
		- Visibility of Targets
	+ Declaring External Dependencies
		- What are External Dependencies?
		- Rules for JVM Dependencies
	+ Declaring an External Library and Using it in the Code
	+ Executing Automated Tests
		- Separating Test Source Code
		- Declaring Test Dependencies
		- Executing Tests
		- Test Reporting
	+ Declaring the JUnit Dependency and Executing Tests
	+ Publishing a JAR to a Maven repository
		- Sharing JAR for consumption from a binary repository
		- Command for publishing to the Maven Local directory
		- Command for publishing to a binary repository
	+ Publishing a Java library to Maven Local
* Outlook on Advanced Topics
	+ Overview of Extension Concepts: Macros and Rules
		- Extension Mechanism
		- When are Macros and Rules Executed?
		- Starlark Build Language Overview
		- High Level: Writing and Using a Macro
		- High Level: Writing and Using a Genrule
		- High Level: Writing and Using a Rule
	+ Writing and executing a genrule
	+ Remote Caching and Execution
		- Motivation
		- Overview Remote Caching
		- Overview Remote Execution
	+ Build Stamping
		- Workspace Status Information
		- Generating and appending custom information with a script
		- Including build information with an artifact
	+ Bazel Query
		- Analyze build dependencies by an expression
		- Transitive closure of dependencies: deps
		- Transitive closure of source dependencies: deps
		- Build files for packages: buildfiles(deps)
		- Finding reverse dependencies: rdeps
	+ Java Toolchains
		- Java rules provide two toolchains
		- Discovering available Java runtime toolchains
		- Configuring JVM and Java compiler flags
		- Configuring toolchains in .bazelrc
		- Defining additional toolchains
* Conclusion