Lab Exercise 5 - Custom Rule

Creating a custom rule in Bazel for a C++ project is a more advanced topic, but it can be a powerful way to encapsulate complex build logic and share it across projects. Here's a lab exercise that demonstrates how to create a custom rule in Bazel for a simple C++ project:

Objective: Create a custom Bazel rule to build a C++ project with custom build logic.

Prerequisites:

- Bazel installed on your machine.
- Basic knowledge of Bazel BUILD files and C++.

Steps:

1. Project Setup:

Create a directory for your custom rule project and set up a basic C++ project structure inside it:

```
mkdir custom_cpp_rule
cd custom_cpp_rule
touch WORKSPACE
mkdir -p src
echo '#include <iostream>
int main() {
  std::cout << "Hello, Custom Rule!" << std::endl;
  return 0;
}' > src/main.cpp
```

2. Define a Custom Rule:

Create a file named BUILD in the root of your project directory and define a custom rule. In this example, we'll create a rule named custom_cpp_binary:

```
load("@rules_cc//cc:defs.bzl", "cc_binary")

def custom_cpp_binary(name):
    cc_binary(
        name = name,
        srcs = ["main.cpp"],
        copts = ["-std=c++11"],
    )
```

In this rule definition:

- We load the cc_binary rule from the built-in rules_cc package.
- The custom_cpp_binary rule takes a name argument, which specifies the name of the target.
- Inside the rule, we use cc_binary to define a C++ binary target with a source file and compiler options.

3. Create a BUILD File for Your C++ Binary:

Now, create a BUILD file in the src directory to define a target for your custom rule:

```
load(":BUILD.bzl", "custom_cpp_binary")

custom_cpp_binary(
    name = "my_custom_cpp_binary",
)
```

In this BUILD file, we use the custom_cpp_binary rule we defined earlier to create a target named my_custom_cpp_binary.

4. Build Your Project:

Run the following Bazel command to build your custom C++ binary:

```
bazel build //src:my_custom_cpp_binary
```

This command tells Bazel to build the target named my_custom_cpp_binary in the src directory.

5. **Run the Binary:**

After a successful build, you can run the binary:

```
bazel-bin/src/my_custom_cpp_binary
```

You should see the output "Hello, Custom Rule!" printed to the console.

6. Custom Rule Usage:

Now that you have a custom rule, you can easily use it in other Bazel projects by loading it and invoking it with a target name, just like we did in step 3.

This lab exercise demonstrates how to create a custom rule in Bazel for a simple C++ project. You can extend this example to add more complex build logic or customize your rule further based on your project's needs.