Remote Debugging Containers

Objectives

- Configure a remote debugger during application development.

Remote Debugging Containers

Debugging applications is an important technique to find and fix bugs. Although a developer can use print statements to observe runtime variables and behavior, using a debugger is more flexible. Debuggers quicken the introspection of complex interactions, which are more likely to have bugs in the first place.

Debugging Without Containers

Many programming language runtimes provide a way to attach a debugger to a running program. When attached, a debugger provides the following features.

- Attach breakpoints that stop program execution.
- Step through individual application instructions.
- Inspect and modify variables.
- Evaluate custom expressions.

For web-based applications, a debugger usually connects via a debug port. If both the application and the debugger are on the same machine, then this connection requires no additional configuration, such as port forwarding.

Debugging Within Containers

Because containers are isolated from their host, debugging containerized applications requires additional steps for connecting a debugger.

Containers do not expose any ports by default, so any debug ports must be forwarded. Additionally, the debug server must listen for remote connections.

For example, consider the Node.js debug mode, which you can enable with the `--inspect` option on the `node` command. By default, the Node.js debug server binds to `127.0.0.1` on port `9229`. To connect to a containerized Node.js application, expose the container port `9229` on the host machine. The debug server must also bind to a non-local interface, for example by using the `0.0.0.0` address.

WARNING

Debug protocols permit program tampering, which poses a security risk. Do not expose processes in debug mode outside of a controlled environment.

Live Updating Code

When debugging, developers perform experiments via small changes to the code. However, because container images often include the application code, local changes are not reflected in the image until developers recreate it, for example by using the `podman build` command.

Recreating container images on every code change can be time consuming. One solution is to mount the application code from the host by using a bind mount during debugging. By overlaying the code with a bind mount, developers can test code changes without recreating the image.

This solution depends on the ability of the application runtime to perform live code changes. The application runtime must have the ability to change the application code without restarting the application.

Debugging With VSCodium

VSCodium is an open source version of the Visual Studio Code text editor. Both versions of the editor include a built-in debugger that can connect to various runtimes' debug protocols.

Developers define a *launch configuration* to attach the VSCodium debugger to the application. The following launch configuration example attaches a debugger to a Node.js application via port `9229`.

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "attach",
            "name": "My Config",
            "port": 9229,
            "address": "localhost",
            "localRoot": "${workspaceFolder}",
            "remoteRoot": "/"
        }
    ]
}
```

REFERENCES

Node.js Debugging Getting Started Guide

Debugging in Visual Studio Code