COMPLETE SETUP & USAGE GUIDE

MCP Servers for Developers

GitHub MCP · AWS API MCP

VS Code · Firebase Studio (formerly Project IDX)

Visual Studio Code	Firebase Studio
Config: .vscode/mcp.json · Key: "servers"	Config: .idx/mcp.json · Key: "mcpServers"

1. What Is MCP and Why It Matters

Model Context Protocol (MCP) is an open standard created by Anthropic that allows AI assistants — such as GitHub Copilot, Claude Dev, or Cursor — to securely connect to external tools and services. Instead of the AI guessing or hallucinating how to interact with services like GitHub or AWS, MCP gives it a structured, validated interface to actually do things in the real world.

Think of MCP as a plugin system for AI. Each MCP server exposes a set of tools. The AI calls those tools with real parameters, and the server executes them against the actual service. Everything is validated before execution, which prevents the AI from running nonsense commands.

Term	What It Means
AI Client	The IDE extension: GitHub Copilot, Claude Dev, Cursor, Cline
MCP Host	The runtime inside VS Code or Firebase Studio managing server connections
MCP Server	A process exposing tools — GitHub server, AWS API server, etc.
Tool Call	The AI invokes a tool (e.g. list_repos) and gets back real data
stdio	Default transport — the AI client communicates via standard input/output

This guide covers two production-ready MCP servers:

- GitHub MCP Server — official server by Anthropic/MCP for repository management
- AWS API MCP Server — official server by AWS Labs for interacting with all AWS services

2. Prerequisites

Before installing any MCP server, make sure the following are on your machine. Based on your system check earlier, here is your current status:

Dependency	Status / Notes
Node.js 18+	✓ Already installed — v22.22.0 (excellent)
npm 9+	✓ Already installed — v10.9.4
Git	✓ Already installed — v2.53.0
Python 3.10+	✗ Required for AWS API MCP Server — install steps below
uv	✗ Python package manager — required for AWS MCP — install below
AWS CLI v2	✗ Not yet installed — install steps below

AWS Account	Required with credentials configured via aws configure
GitHub Account	Required to generate a Personal Access Token

2.1 Install Python 3.10+

The AWS API MCP Server is Python-based and requires Python 3.10 or newer.

```
# Ubuntu / Debian (your system)
sudo apt update
sudo apt install python3 python3-pip python3-venv -y

# Verify
python3 --version
# Expected: Python 3.10.x or higher
```

2.2 Install uv (Python package manager)

uv is a fast Python package manager used to run the AWS MCP server. It is the recommended installation method by AWS Labs.

```
# Linux / macOS
curl -LsSf https://astral.sh/uv/install.sh | sh

# Reload your shell so uv is in PATH
source ~/.bashrc

# Verify
uv --version
```

2.3 Install AWS CLI v2

Your system detected awscli2 as available. Install it now:

```
# Fedora / RHEL / CentOS (your system — suggested awscli2)
sudo dnf install awscli2 -y

# Ubuntu / Debian alternative
curl 'https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip' -o
awscliv2.zip
unzip awscliv2.zip
sudo ./aws/install

# Verify
aws --version
# Expected: aws-cli/2.x.x Python/3.x.x
```

✓ **TIP**

The AWS CLI is free to install. You only pay for the AWS services you actually provision or use — listing and describing resources costs fractions of a cent.

Chapter 3 — GitHub MCP Server

3. GitHub MCP Server

The GitHub MCP Server is the official server published under the Model Context Protocol organization. It lets your AI assistant create branches, open pull requests, read files, list repositories, manage issues, and more — all through natural language.

3.1 How It Works

The server runs as a background process on your machine. When you type a prompt like "create a branch called feature/login", the AI translates that into a GitHub API call through the MCP server. Your Personal Access Token is what authorizes those calls.

3.2 Generate a GitHub Personal Access Token

A Personal Access Token (PAT) is how the MCP server authenticates with GitHub on your behalf. You generate it once in your GitHub settings.

1	Open your browser and go to github.com
2	Click your profile picture (top-right) → Settings
3	Scroll down the left sidebar → Developer settings
4	Click Personal access tokens → Tokens (classic)
5	Click Generate new token → Generate new token (classic)
6	Give it a name like mcp-server-token and set an expiration (90 days recommended)
7	Select the scopes listed in the table below
8	Click Generate token at the bottom — copy it immediately, it will not be shown again

Scope	What It Enables
repo	Full read/write to public and private repositories — core requirement
workflow	Trigger and manage GitHub Actions workflows
read:org	Read organization membership and teams (needed if you use GitHub orgs)
read:user	Read your own user profile data
gist	Create and manage Gists — optional, add only if needed

WARNING

Minimum required scope for basic repository access is repo only. Start with that and add workflow and read:org only if your workflows need them. Never share your token or commit it to a repository.

3.3 Set the Environment Variable

The MCP config file uses ${env:VARIABLE_NAME} syntax to read your token from the shell environment at startup. This keeps your token out of config files.

```
# Open your shell profile
nano ~/.bashrc

# Add this line (replace with your actual token)
export GITHUB_PERSONAL_ACCESS_TOKEN="ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxx"

# Save (Ctrl+O then Enter), exit (Ctrl+X)

# Reload the profile
source ~/.bashrc

# Verify it is set
echo $GITHUB_PERSONAL_ACCESS_TOKEN
# Should print your token starting with ghp_
```

3.4 Test the Server Manually

Before connecting it to an IDE, confirm the server starts without errors:

```
npx -y @modelcontextprotocol/server-github

# Expected output:
# GitHub MCP Server running on stdio

# Press Ctrl+C to stop it — this confirms it is working
```

3.5 What the GitHub MCP Server Can Do

Tool / Action	What It Does
list_repos	List all repos in your account or an organization
get_file_contents	Read any file from any branch of a repository
create_or_update_file	Create a new file or update an existing one with a commit
create_branch	Create a new branch from any base ref
create_pull_request	Open a PR with a title, description, and base branch
list_issues	List open or closed issues with optional filters

create_issue	Create a new issue with a title, body, and labels
add_issue_comment	Comment on an existing issue
search_repositories	Search GitHub for repos matching a query
search_code	Search for code across repositories
merge_pull_request	Merge an open pull request
list_commits	List commits on a branch with author and message

Chapter 4 — AWS API MCP Server (Official by AWS Labs)

4. AWS API MCP Server

The AWS API MCP Server is the official MCP server published by AWS Labs at awslabs.github.io/mcp. It gives your AI assistant the ability to run any AWS CLI command, with built-in validation that prevents hallucinated or invalid commands from executing.

INFO

This is NOT the shell-based workaround described in older guides. This is the official AWS-maintained server that validates every command before execution, supports read-only mode, and can require your consent before any write operations.

4.1 How It Works

The server wraps the AWS CLI and boto3. When the AI wants to run aws ec2 describe-instances, it calls the call_aws tool with those parameters. The server validates the command against a list of known AWS CLI commands, executes it with your configured credentials, and returns the output. It cannot run arbitrary shell commands — only valid AWS CLI commands.

4.2 Configure AWS Credentials

The AWS API MCP Server reads credentials the same way the AWS CLI does — from ~/.aws/credentials. Run the following to configure them:

```
aws configure

# You will be prompted for four values:
# AWS Access Key ID [None]:     AKIAIOSFODNN7EXAMPLE
# AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENGbPxRfiCYEXAMPLEKEY
# Default region name [None]:   us-east-1
# Default output format [None]: json
```

This creates two files on your system:

- ~/.aws/credentials — stores your access key and secret key
- ~/.aws/config — stores your region and output format

Verify the credentials work:

```
aws sts get-caller-identity

# Expected output:
{
    "UserId": "AIDACKCEVSQ6C2EXAMPLE",
    "Account": "123456789012",
    "Arn": "arn:aws:iam::123456789012:user/your-username"
}
```

IMPORTANT

Never use root account credentials. Create a dedicated IAM user in the AWS Console with only the permissions your workflows need. Root credentials have unlimited access and cannot be scoped down.

4.3 Recommended IAM Permissions

Create an IAM user in the AWS Console with a policy scoped to what you actually need. For a read-only exploratory setup, attach ReadOnlyAccess. For a full development setup, use a custom policy like this:

```
# Attach AWS managed read-only policy (safe starting point)
aws iam attach-user-policy \
  --user-name your-mcp-user \
  --policy-arn arn:aws:iam::aws:policy/ReadOnlyAccess

# Or use the MCP server in read-only mode via environment variable
# READ_OPERATIONS_ONLY=true  (set this in the MCP config — shown later)
```

IAM Policy	What It Allows
ReadOnlyAccess	Safe starting point — describe and list everything, write nothing
AmazonS3ReadOnlyAccess	Read S3 buckets and objects only
AmazonEC2ReadOnlyAccess	Describe EC2 instances, VPCs, security groups
AWSLambda_ReadOnlyAccess	List and inspect Lambda functions
CloudWatchReadOnlyAccess	Read logs and metrics — essential for debugging
AmazonRDSReadOnlyAccess	Describe RDS instances and configurations
AdministratorAccess	Full access — only use in isolated dev/sandbox accounts

4.4 Security Configuration Options

The AWS API MCP Server has two important safety environment variables you can set in your MCP config:

Variable	What It Does
READ_OPERATIONS_ONLY=true	Blocks all write/mutating AWS operations. The AI can only list, describe, and get — it cannot create, delete, or modify anything.
REQUIRE_MUTATION_CONSENT=true	Before any write operation runs, the server pauses and asks for your explicit consent. Requires an MCP client that supports elicitation (VS Code Copilot, Cline).

AWS_API_MCP_PROFILE_NAME	Specify a named AWS profile from ~/.aws/credentials to use instead of default.
AWS_REGION	Set a default region for all commands. Defaults to us-east-1 if not specified.

✓ TIP

For day-to-day development, start with READ_OPERATIONS_ONLY=true. This lets the AI help you explore and understand your infrastructure without any risk of accidental changes. Switch to false only when you intentionally want the AI to help make changes.

4.5 What the AWS API MCP Server Can Do

Tool	What It Does
call_aws	Execute any valid AWS CLI command — ec2, s3, lambda, iam, rds, cloudformation, etc.
suggest_aws_commands	Given a description in plain English, suggests the 5 most relevant AWS CLI commands with full parameters — helps with commands the AI might not know
get_execution_plan	Experimental: structured step-by-step plan for complex AWS tasks like setting up a VPC or deploying a Lambda

5. The Global MCP Configuration File

Both VS Code and Firebase Studio read MCP server definitions from a JSON config file. You will create a global one that both IDEs can reference, then create small IDE-specific files that point to it.

5.1 Create the Global Directory

```
mkdir -p ~/.config/mcp
touch ~/.config/mcp/config.json
```

5.2 The Complete Config File

Open the file and paste the following. This defines both servers with all recommended settings:

```
# File: ~/.config/mcp/config.json
# This is your global MCP server definition — shared across all projects
{
  "servers": {

    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "${env:GITHUB_PERSONAL_ACCESS_TOKEN}"
      }
    },

    "awslabs.aws-api-mcp-server": {
      "command": "uvx",
      "args": ["awslabs.aws-api-mcp-server@latest"],
      "env": {
        "AWS_REGION": "us-east-1",
        "AWS_PROFILE": "default",
        "READ_OPERATIONS_ONLY": "false",
        "REQUIRE_MUTATION_CONSENT": "false"
      },
      "disabled": false,
      "autoApprove": []
    }

  }
}
```

✓ TIP

Set READ_OPERATIONS_ONLY to true if you want to start safely with read-only access. Change REQUIRE_MUTATION_CONSENT to true if you want the AI to ask you before making any changes to your AWS infrastructure.

6. Setting Up MCP in VS Code

As of VS Code 1.102, MCP support is built in and out of preview. You do not need a separate extension — it works natively with GitHub Copilot Chat (requires Copilot subscription) or with free extensions like Cline or Claude Dev.

6.1 Option A — Using VS Code with Cline (Free)

Cline is a free open-source AI coding assistant that fully supports MCP. This is the recommended free option.

1	Open VS Code and go to the Extensions panel (Ctrl+Shift+X)
2	Search for Cline and install it
3	Click the Cline icon in the left sidebar to open the panel
4	Click the MCP Servers icon (plug icon) at the top of the panel
5	Click Edit Global MCP Config
6	Paste the contents of your ~/.config/mcp/config.json file and save
7	Click the Refresh button — both servers should show a green dot

6.2 Option B — Using VS Code Native MCP (Copilot)

If you have GitHub Copilot, VS Code handles MCP natively through the Command Palette.

1	Open the Command Palette: Ctrl+Shift+P
2	Type MCP and select MCP: Open User Configuration
3	This opens a settings.json-style file — paste your server definitions here
4	Save the file and reload VS Code (Ctrl+Shift+P → Developer: Reload Window)
5	Open Copilot Chat (Ctrl+Alt+I) — MCP tools will appear in the tools panel

6.3 Project-Level Config (VS Code)

To use MCP in a specific project without affecting your global settings, create a .vscode/mcp.json file inside the project folder:

```
# Inside your project directory
mkdir -p .vscode
nano .vscode/mcp.json

# Paste this content:
{
  "servers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "${env:GITHUB_PERSONAL_ACCESS_TOKEN}"
      }
    },
    "awslabs.aws-api-mcp-server": {
      "command": "uvx",
      "args": ["awslabs.aws-api-mcp-server@latest"],
      "env": {
        "AWS_REGION": "us-east-1",
        "READ_OPERATIONS_ONLY": "true"
      }
    }
  }
}
```

WARNING

The .vscode/mcp.json file uses the key "servers" (not "mcpServers" — that is the Firebase Studio format). This is one of the most common mistakes when switching between IDEs.

6.4 Verify VS Code Setup

```
# In the VS Code terminal, check MCP processes are running:
ps aux | grep -E 'server-github|aws-api-mcp'

# You should see Node.js and Python processes for each server
# If nothing appears, reload VS Code and check your env variables:
echo $GITHUB_PERSONAL_ACCESS_TOKEN
aws sts get-caller-identity
```

Chapter 7 — Firebase Studio Setup (formerly Project IDX)

7. Setting Up MCP in Firebase Studio

Firebase Studio (formerly Project IDX) is Google's browser-based AI development environment. As of April 2025, Project IDX has been fully transitioned into Firebase Studio at studio.firebase.google.com. It supports MCP servers natively.

INFO

Firebase Studio runs in the cloud, not on your local machine. This means the MCP servers run inside your workspace container. Environment variables must be set inside the workspace, not in your local ~/.bashrc.

7.1 Key Differences from VS Code

Difference	Details
Config file location	VS Code: .vscode/mcp.json | Firebase Studio: .idx/mcp.json
JSON key name	VS Code: "servers" | Firebase Studio: "mcpServers"
Environment variables	VS Code: reads from local shell | Firebase: set in .idx/dev.nix or workspace settings
Server execution	VS Code: runs on your machine | Firebase: runs in the cloud container
Node.js / Python	VS Code: uses your local install | Firebase: uses container packages

7.2 Create the MCP Config File

Inside your Firebase Studio workspace, create the .idx directory and the mcp.json file:

```
# In the Firebase Studio terminal
mkdir -p .idx
nano .idx/mcp.json
```

Paste the following — note the mcpServers key (different from VS Code):

```
{
  "mcpServers": {

    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "${env:GITHUB_PERSONAL_ACCESS_TOKEN}"
      }
    },

    "awslabs.aws-api-mcp-server": {
      "command": "uvx",
      "args": ["awslabs.aws-api-mcp-server@latest"],
      "env": {
        "AWS_REGION": "us-east-1",
        "AWS_PROFILE": "default",
        "READ_OPERATIONS_ONLY": "false"
      },
      "disabled": false,
      "autoApprove": []
    }

  }
}
```

7.3 Set Environment Variables in Firebase Studio

Because Firebase Studio runs in a cloud container, you set environment variables in your workspace's dev.nix file rather than ~/.bashrc:

```
# Open .idx/dev.nix in your workspace
nano .idx/dev.nix

# Add the env section:
{ pkgs, ... }: {
  env = {
    GITHUB_PERSONAL_ACCESS_TOKEN = "ghp_xxxxxxxxxxxxxxxxxxxx";
    AWS_REGION = "us-east-1";
  };

  packages = [
    pkgs.nodejs_20
    pkgs.python311
    pkgs.uv
    pkgs.awscli2
  ];
}
```

WARNING

After editing dev.nix, click Rebuild Environment in Firebase Studio for the changes to take effect. The workspace container will restart with the new packages and environment variables.

7.4 AWS Credentials in Firebase Studio

For AWS access inside Firebase Studio, you have two options:

Option 1 — Environment variables (simplest for personal use):

```
# Add to .idx/dev.nix env section:
AWS_ACCESS_KEY_ID = "AKIAIOSFODNN7EXAMPLE";
```

```
AWS_SECRET_ACCESS_KEY = "wJalrXUtnFEMI/K7MDENGbPxRfiCYEXAMPLEKEY";
AWS_DEFAULT_REGION = "us-east-1";
```

Option 2 — AWS credentials file (same as local setup):

```
# Run aws configure inside the Firebase Studio terminal
aws configure

# This writes to ~/.aws/credentials inside the container
# Note: this resets when the workspace rebuilds unless
# you persist it via dev.nix or a secrets manager
```

IMPORTANT

Never put real AWS credentials directly in dev.nix if you are committing that file to a public repository. Use Firebase Studio's Secrets feature or set credentials at runtime in the terminal instead.

8. Example Prompts — What to Ask Your AI Assistant

Once MCP is connected in either IDE, you interact with it purely through natural language in the AI chat panel. Here are practical examples for both servers.

8.1 GitHub MCP Prompts

What You Want	Prompt to Type
List repos	List all my GitHub repositories and tell me which ones were updated this week
Read a file	Show me the contents of src/app.ts in my project my-web-app on the main branch
Create a branch	Create a new branch called feature/user-authentication from main in my-web-app
Create a file	In my-web-app on the feature/user-authentication branch, create a file called src/auth.ts with a basic JWT auth setup
Open a pull request	Open a pull request from feature/user-authentication into main in my-web-app with the title 'Add JWT authentication'
Review issues	List all open issues in my-web-app that are labeled bug and tell me which is oldest
Create an issue	Create an issue in my-web-app titled 'Token refresh not working on mobile' with a description of the problem
Search code	Search for all files containing TODO in my-web-app repository
Commit changes	Update the README.md in my-web-app on main to add a section about authentication
Merge a PR	Merge pull request #14 in my-web-app using squash merge

8.2 AWS API MCP Prompts

What You Want	Prompt to Type
Explore account	What AWS services do I have running right now? Give me a summary.
List EC2	List all my EC2 instances in us-east-1 and show their instance type and state
Check S3	List all my S3 buckets and tell me which ones have public access enabled
Lambda functions	List all my Lambda functions and show their runtime, memory, and last modified date
CloudWatch logs	List all my CloudWatch log groups and show which ones have had activity in the last 7 days

Check costs	Run aws ce get-cost-and-usage for the last 30 days and summarize what I'm spending on
Security check	List all my IAM users and show which ones have not logged in for more than 90 days
Find RDS	List all my RDS instances and their engine versions and storage sizes
CloudFormation	List all my CloudFormation stacks and show their status
Specific region	List all EC2 instances in eu-west-1 and compare with my us-east-1 instances

9. Troubleshooting

Here are the most common problems and their solutions.

9.1 GitHub MCP Issues

Problem	Solution
Server shows red dot in Cline	Your GITHUB_PERSONAL_ACCESS_TOKEN is not set or not exported. Run: echo $GITHUB_PERSONAL_ACCESS_TOKEN in a new terminal. If empty, re-run: source ~/.bashrc
401 Unauthorized	Your token has expired or does not have the right scopes. Generate a new token with the repo scope at github.com/settings/tokens
npx: command not found	Node.js is not in PATH. Run: source ~/.bashrc — if still missing, reinstall Node.js and verify with: node --version
Repository not found	The repo name is wrong or your token does not have access to it. Verify with: curl -H 'Authorization: token YOUR_TOKEN' https://api.github.com/user/repos
Rate limit exceeded	GitHub API has rate limits for unauthenticated requests. Ensure your token is properly configured — authenticated requests get 5000 requests/hour

9.2 AWS API MCP Issues

Problem	Solution
uvx: command not found	uv is not installed or not in PATH. Re-run: curl -LsSf https://astral.sh/uv/install.sh | sh — then: source ~/.bashrc
Unable to locate credentials	AWS credentials are not configured. Run: aws configure — then verify: aws sts get-caller-identity
AccessDenied error	Your IAM user does not have permission for that action. Check your IAM policy in the AWS Console and attach the needed policy
Server starts but AI cannot call it	Restart VS Code completely (not just reload window). MCP processes need a fresh start to be detected by the AI client
Wrong region results	Set AWS_REGION in your MCP config or add --region us-west-2 to the specific prompt
Python version error	The server requires Python 3.10+. Run: python3 --version — if below 3.10, install a newer version
uv: no such file in PATH	After installing uv, you must reload the shell: source ~/.bashrc or open a new terminal before launching VS Code

9.3 Firebase Studio Specific Issues

Problem	Solution
MCP servers not loading	You must use the key mcpServers (not servers) in .idx/mcp.json — this is the most common mistake
Env variables not available	After editing dev.nix, you must click Rebuild Environment in the Firebase Studio UI — changes do not apply until the container restarts
uvx or npx not found in container	Add pkgs.uv and pkgs.nodejs_20 to the packages list in .idx/dev.nix and rebuild the environment
AWS credentials reset on rebuild	Use Firebase Studio's Secrets feature to persist credentials across rebuilds instead of hardcoding in dev.nix

Chapter 10 — Global Init: One Command to Activate MCP in Any Repo

10. Global MCP Init — Activate in Any Repo with One Command

The goal of this chapter is to make MCP feel like a global environment rather than something you manually wire up per project. You will set this up once, and from that point forward a single command — mcp-init — will scaffold the correct config files for any repository you clone or create, for both VS Code and Firebase Studio simultaneously.

✓ **TIP**

After completing this chapter, you will never manually create .vscode/mcp.json or .idx/mcp.json again. One command handles both IDEs, reads your global config, and drops the right files into whatever directory you are in.

10.1 How the Global System Works

The approach has three layers:

- A master config file at ~/.config/mcp/config.json holds all your server definitions in one place. This is the single source of truth.
- A shell script called mcp-init reads that master config and generates the correct IDE-specific files (.vscode/mcp.json and .idx/mcp.json) in any directory you run it from.
- A shell alias makes the script callable from anywhere without a full path.

For VS Code specifically, there is also a user-level global config that means you do not even need mcp-init at all — the servers are available in every VS Code workspace automatically. The mcp-init command is mainly for Firebase Studio and for teams where you want configs committed to the repo.

10.2 Step 1 — Write the Master Config

If you completed Chapter 5 this file already exists. If not, create it now with a single heredoc command:

```
mkdir -p ~/.config/mcp

cat > ~/.config/mcp/config.json << 'EOF'
{
  "servers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "${env:GITHUB_PERSONAL_ACCESS_TOKEN}"
      }
    },
    "awslabs.aws-api-mcp-server": {
      "command": "uvx",
      "args": ["awslabs.aws-api-mcp-server@latest"],
      "env": {
        "AWS_REGION": "us-east-1",
        "AWS_PROFILE": "default",
        "READ_OPERATIONS_ONLY": "false",
        "REQUIRE_MUTATION_CONSENT": "false"
      },
      "disabled": false,
      "autoApprove": []
    }
  }
}
EOF
echo 'Master config ready at ~/.config/mcp/config.json'
```

10.3 Step 2 — Create the mcp-init Script

Paste this entire block into your terminal at once. It creates the script, makes it executable, and adds it to your PATH:

```
mkdir -p ~/.local/bin

cat > ~/.local/bin/mcp-init << 'SCRIPT'
#!/usr/bin/env bash
# mcp-init — write MCP configs for VS Code + Firebase Studio
# Usage:  mcp-init          (current directory)
#         mcp-init <path>   (target directory)
set -e
TARGET="${1:-.}"

# ── VS Code (.vscode/mcp.json) ──────────────────────────
mkdir -p "$TARGET/.vscode"
cat > "$TARGET/.vscode/mcp.json" << 'JSON'
{
  "servers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "${env:GITHUB_PERSONAL_ACCESS_TOKEN}"
      }
    },
    "awslabs.aws-api-mcp-server": {
      "command": "uvx",
      "args": ["awslabs.aws-api-mcp-server@latest"],
      "env": {
        "AWS_REGION": "us-east-1",
        "AWS_PROFILE": "default",
        "READ_OPERATIONS_ONLY": "false",
        "REQUIRE_MUTATION_CONSENT": "false"
      },
      "disabled": false,
      "autoApprove": []
    }
  }
}
JSON

# ── Firebase Studio (.idx/mcp.json) ─────────────────────
mkdir -p "$TARGET/.idx"
cat > "$TARGET/.idx/mcp.json" << 'JSON'
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "${env:GITHUB_PERSONAL_ACCESS_TOKEN}"
      }
    },
    "awslabs.aws-api-mcp-server": {
      "command": "uvx",
      "args": ["awslabs.aws-api-mcp-server@latest"],
      "env": {
        "AWS_REGION": "us-east-1",
        "AWS_PROFILE": "default",
        "READ_OPERATIONS_ONLY": "false",
        "REQUIRE_MUTATION_CONSENT": "false"
      },
      "disabled": false,
      "autoApprove": []
    }
  }
}
JSON

ABS=$(realpath "$TARGET")
echo ""
echo "MCP initialized in: $ABS"
echo "  .vscode/mcp.json  — VS Code"
echo "  .idx/mcp.json     — Firebase Studio"
echo ""
echo "VS Code:          open folder — servers connect automatically"
echo "Firebase Studio:  open workspace — rebuild env if first time"
SCRIPT

chmod +x ~/.local/bin/mcp-init

# Add ~/.local/bin to PATH if not already there
grep -q '.local/bin' ~/.bashrc || echo 'export PATH="$HOME/.local/bin:$PATH"'
>> ~/.bashrc
source ~/.bashrc

# Confirm
which mcp-init && echo 'mcp-init is ready'
```

10.4 Step 3 — VS Code Global Config (No Per-Repo Files Needed)

For VS Code you can register the servers globally so they activate in every workspace with zero per-project setup. There are two ways to do this:

Method A — Command Palette (recommended):

```
# 1. Open VS Code
# 2. Ctrl+Shift+P  →  type: MCP: Open User Configuration
# 3. Paste the full contents of ~/.config/mcp/config.json
# 4. Save — all servers are now global across every VS Code workspace
```

Method B — Copy config file directly:

```
# Linux path for VS Code user MCP config:
mkdir -p ~/.config/Code/User
cp ~/.config/mcp/config.json ~/.config/Code/User/mcp.json

# macOS path:
# ~/Library/Application Support/Code/User/mcp.json

# Reload VS Code — MCP servers appear in every workspace
```

INFO

With the user-level config in place, VS Code needs no .vscode/mcp.json in any project. The mcp-init command becomes useful only when you want to commit configs for teammates or when working in Firebase Studio.

10.5 Daily Workflow — Using mcp-init

From this point forward, activating both MCP servers in any repo is one line:

```
# Clone a repo and immediately initialize MCP
git clone https://github.com/yourname/some-project.git
cd some-project
mcp-init

# Output:
# MCP initialized in: /home/you/some-project
#   .vscode/mcp.json  — VS Code
#   .idx/mcp.json     — Firebase Studio
#
# VS Code:          open folder — servers connect automatically
# Firebase Studio:  open workspace — rebuild env if first time
```

Point at any directory without cd-ing first:

```
mcp-init ~/projects/my-other-project
mcp-init .   # same as running with no argument
```

10.6 Auto-Init on git clone (Optional)

Add a git alias so cloning and initializing MCP happen in one command:

```
git config --global alias.clone-mcp '!f() {
  git clone "$@"
  cd "$(basename "${1%.git}")"
  mcp-init
}; f'

# Usage — replaces your usual git clone:
git clone-mcp https://github.com/yourname/any-repo.git
# Repo is cloned AND both MCP configs are written automatically
```

10.7 Propagate Changes to All Repos

When you update your master config (adding a server, changing a region), push the change to every repo at once:

```
# Re-run mcp-init on every git repo inside ~/projects
for dir in ~/projects/*/; do
  if [ -d "$dir/.git" ]; then
    echo "Updating $dir ..."
    mcp-init "$dir"
  fi
done

# Target a specific list of repos
for repo in my-app api-service frontend-app; do
  mcp-init ~/projects/$repo
done
```

10.8 Keep MCP Configs Out of Git (Optional)

If you do not want to commit the generated files, add them to your global gitignore:

```
# Append to global gitignore
cat >> ~/.gitignore_global << 'EOF'
.vscode/mcp.json
.idx/mcp.json
EOF

# Register the global gitignore (if not done already)
git config --global core.excludesfile ~/.gitignore_global
```

✓ **TIP**

If you want teammates to get MCP working when they clone the repo, do NOT add the files to gitignore — commit them instead. The configs contain no secrets because all credentials come from environment variables.

10.9 Scenario Reference

Scenario	What to Do
First time setup	Run steps 10.2 and 10.3 once. Set VS Code global config via Command Palette once.

New repo, VS Code only	Nothing to do — VS Code global config activates servers in every workspace.
New repo, Firebase too	cd into repo, run: mcp-init — both config files written in one shot.
Clone and init at once	Use: git clone-mcp <url> — clone and mcp-init run together automatically.
Update server settings	Edit ~/.config/mcp/config.json, re-run mcp-init in any affected repos.
Update all repos at once	Use the for loop from section 10.7 to regenerate every repo.
Share configs with team	Commit .vscode/mcp.json and .idx/mcp.json — teammates get structure, not secrets.
Keep configs private	Add both filenames to ~/.gitignore_global as shown in section 10.8.

11. Quick Reference — VS Code vs Firebase Studio

Setting	VS Code | Firebase Studio
Config file	.vscode/mcp.json | .idx/mcp.json
Root JSON key	"servers": { } | "mcpServers": { }
Global config	MCP: Open User Configuration (palette) | .idx/mcp.json per workspace
Environment vars	~/.bashrc or ~/.zshrc | .idx/dev.nix env section
Node.js source	Your local installation | pkgs.nodejs_20 in dev.nix
Python / uv source	Your local installation | pkgs.uv in dev.nix
AWS credentials	~/.aws/credentials via aws configure | Env vars in dev.nix or Secrets
Restart required	Developer: Reload Window | Rebuild Environment button
MCP extension	Native (1.102+) or Cline/Claude Dev | Native in workspace AI panel

12. Complete Setup Checklist

System Prerequisites

- Node.js 18+ installed (you have v22 — done)
- npm 9+ installed (you have v10 — done)
- Git installed (you have v2.53 — done)
- Python 3.10+ installed (sudo apt install python3 -y)
- uv installed (curl -LsSf https://astral.sh/uv/install.sh | sh)
- AWS CLI v2 installed (sudo dnf install awscli2 -y)

GitHub MCP

- GitHub Personal Access Token generated with repo scope
- GITHUB_PERSONAL_ACCESS_TOKEN exported in ~/.bashrc
- Tested manually: npx -y @modelcontextprotocol/server-github

AWS API MCP

- IAM user created with appropriate permissions
- AWS credentials configured via aws configure
- Verified with: aws sts get-caller-identity
- uv in PATH (uvx --version works in a new terminal)

VS Code Config

- ~/.config/mcp/config.json created with both servers
- .vscode/mcp.json created in project (uses key: "servers")
- Cline or Copilot extension installed and connected
- Both servers show green in the MCP panel

Firebase Studio Config

- .idx/mcp.json created in workspace (uses key: "mcpServers")
- .idx/dev.nix updated with env variables and packages
- Environment rebuilt after dev.nix changes
- Tested with a sample prompt in the AI panel

Official Resources

GitHub MCP Server: github.com/modelcontextprotocol/servers
AWS API MCP Server: awslabs.github.io/mcp/servers/aws-api-mcp-server
Firebase Studio: studio.firebase.google.com