GUIDE

AWS CLI v1 to v2 Migration Guide

Migration: v1 → v2 Effort estimate: 1-2 hours for simple usage, 1-3 days for large codebases with extensive scripting **Complexity:** MAJOR

Breaking changes in this upgrade: - Changes in command syntax and argument parsing (e.g., how parameters are passed) - Differences in default output formats (e.g., JSON structure, default pagination behavior) - Behavioral changes in certain commands (e.g., file transfer mechanisms, error handling) - Changes in how environment variables are interpreted - Deprecation or removal of specific commands or parameters (identified by migration tools)

As a migration lead, I understand that upgrading core infrastructure tools like the AWS CLI requires a careful, methodical approach, especially in production environments. This guide outlines a safe and complete path for migrating from AWS CLI v1 to v2, focusing on identifying and resolving breaking changes to ensure a smooth transition.

Why Upgrade to AWS CLI v2?

Upgrading to AWS CLI v2 isn't just about staying current; it's about leveraging significant improvements that enhance performance, security, and developer experience. AWS CLI v2 introduces several key advantages:

- **Enhanced Performance:** v2 is often faster, especially for operations involving large data transfers or numerous API calls, due to underlying architectural improvements.
- **Improved Installers:** Official installers for Linux, macOS, and Windows simplify deployment and management, reducing dependency conflicts often encountered with v1's Python `pip` installation.
- **New Features:** v2 supports new interactive features, improved pagination, and a more consistent command-line experience. It's also the only version that will receive new features and updates from AWS.
- **Stricter Input Validation:** While a breaking change, stricter validation helps prevent errors by catching malformed commands or parameters earlier.

- **Bundled Python Runtime:** v2 ships with its own Python runtime, eliminating dependencies on the system's Python version and potential conflicts.
- **Long-Term Support:** AWS CLI v2 is the actively developed and supported version, ensuring compatibility with new AWS services and API updates.

This migration is crucial for maintaining a robust and efficient interaction with your AWS resources.

Key Differences Between AWS CLI v1 and v2

The transition from AWS CLI v1 to v2 involves several breaking changes that require attention. Understanding these differences is critical for a successful migration.

Changes in Command Syntax and Argument Parsing

AWS CLI v2 enforces stricter parsing rules, particularly for JSON input and list parameters. This means commands that worked in v1 might fail in v2 due to subtle syntax variations.

- **What changed:** v2 has a more robust command-line parser. This affects how complex parameters (like JSON strings or lists) are passed. For example, JSON strings often require proper escaping or file input.
- **Before (v1) example:** Passing a JSON string directly without strict escaping might have worked.
  `bash # v1 example (might work depending on shell and content) aws ec2 run-instances --image-id ami-0abcdef1234567890 --instance-type t2.micro --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=MyInstance}]'`-
  After (v2) example: v2 often requires more explicit JSON formatting, using a file, or careful escaping. `bash # v2 example (using a file for complex JSON) # Create a file named 'tags.json' with content: # [{ "ResourceType": "instance", "Tags": [{ "Key": "Name", "Value": "MyInstance" }] }] aws ec2 run-instances --image-id ami-0abcdef1234567890 --instance-type t2.micro --tag-specifications file://tags.json` Alternatively, with direct JSON, ensure proper escaping: `bash # v2 example (direct JSON with careful escaping) aws ec2 run-instances --image-id ami-0abcdef1234567890 --instance-type t2.micro --tag-specifications '[{"ResourceType":"instance","Tags": [{"Key":"Name","Value":"MyInstance"}]}]'`

Differences in Default Output Formats and Pagination Behavior

The default output structure for some commands and the pagination mechanism have been refined in v2. This can break scripts that rely on specific JSON paths or expect all results in a single call.

- **What changed:**
- **JSON Structure:** Minor changes in the nesting or naming of fields in JSON output for certain commands.
- **Pagination:** v2 often defaults to client-side pagination where the CLI makes multiple API calls to retrieve all results, potentially changing the behavior of commands that previously required explicit `--starting-token` or `--max-items` in v1. The default page size might also differ.
- **Before (v1) example:** A script might expect a flat list of items or manually paginate. `bash # v1 example: Listing S3 objects, may require manual pagination for many objects aws s3api list-objects --bucket my-bucket --query 'Contents[].Key' --output text` - **After (v2) example:** v2's client-side pagination often handles fetching all results automatically, but if you relied on specific page sizes or tokens, you might need to adjust. `bash # v2 example: Client-side pagination often fetches all automatically # Output structure might be subtly different, requiring 'jq' adjustments aws s3api list-objects-v2 --bucket my-bucket --query 'Contents[].Key' --output text` > 🧠 Important: Always test scripts that parse CLI output with `jq` or similar tools, as minor JSON structure changes can cause failures.

Behavioral Changes in Certain Commands

Specific commands might exhibit different behavior in v2, impacting file transfers, error handling, or how certain operations are performed.

- **What changed:** Commands like `s3 cp` and `s3 sync` have improved robustness and error handling. This might mean previously ignored errors are now surfaced, or transfer mechanisms are more efficient but behave slightly differently.
- **Before (v1) example:** An `s3 sync` operation might have silently skipped certain files.
  `bash # v1 example: Syncing files aws s3 sync ./local/path s3://my-bucket/remote/path` - **After (v2) example:** v2 might introduce more detailed logging, stricter permissions checks, or different multipart upload thresholds. `bash # v2 example: Syncing files, potentially with enhanced error reporting or performance aws s3 sync ./local/path`

`s3://my-bucket/remote/path` > ⚠️ Risk: Scripts relying on specific error codes or silent failures might need updates to handle new error messages or behaviors.

Changes in How Environment Variables are Interpreted

Subtle changes in how AWS CLI v2 interprets and prioritizes environment variables can affect authentication, region selection, or other configuration settings.

- **What changed:** While core environment variables (like `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_REGION`) remain largely the same, there might be nuanced changes in precedence or how less common variables are handled.
- **Before (v1) example:** `bash export AWS_DEFAULT_OUTPUT="json" aws s3 ls` - **After (v2) example:** This typically remains consistent, but it's essential to verify. If you use advanced environment variables for proxy settings or custom endpoints, review the v2 documentation for any changes. `bash export AWS_DEFAULT_OUTPUT="json" aws s3 ls` > 🧠 Important: If you use complex `~/.aws/config` or `~/.aws/credentials` files, or rely heavily on environment variables for multi-account setups, thoroughly test your configuration.

Deprecation or Removal of Specific Commands or Parameters

AWS CLI v2 has deprecated or removed some commands or parameters that were present in v1, often due to changes in underlying AWS APIs or a move towards more consistent naming conventions.

- **What changed:** Certain commands or flags are no longer supported. The `AWS CLI v1-to-v2 Migration Tool` is specifically designed to identify these.
- **Example:** While no universally applicable "before/after" code snippet exists for all removed APIs, the migration tool will flag specific instances. For example, if a command was renamed or a parameter was removed from an API call, the tool would highlight it. `bash # v1 example (hypothetical deprecated command/parameter) aws some-service old-command --deprecated-flag value` The migration tool would identify `old-command` or `--deprecated-flag` as removed and suggest an alternative if available, or indicate it needs manual review.

Step-by-Step Installation of AWS CLI v2

The recommended approach for migrating is to install AWS CLI v2 alongside v1 initially. This allows you to test v2 without immediately disrupting existing workflows.

> ⚠️ Risk: AWS CLI v1 and v2 use the same `aws` command name. If you install v2 in a way that overwrites or takes precedence over v1 in your `PATH`, it will immediately impact all scripts. We will install v2 in a separate location and manage the `PATH` carefully.

1. Download the AWS CLI v2 Installer

Choose the appropriate installer for your operating system.

- **For Linux/macOS:** `bash curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip" unzip awscliv2.zip` > ✅ Safe to skip if: You are on macOS and prefer the `.pkg` installer or Windows. Adjust the URL for ARM-based systems if needed.
- **For Windows (64-bit):** Download the MSI installer directly: `https://awscli.amazonaws.com/AWSCLIV2.msi` You can use PowerShell to download: `powershell Invoke-WebRequest -Uri "https://awscli.amazonaws.com/AWSCLIV2.msi" -OutFile "AWSCLIV2.msi"`

2. Install AWS CLI v2 in a Separate Location

This step is crucial to avoid immediately overwriting your v1 installation.

- **For Linux/macOS:** Install into `/usr/local/aws-cli-v2` (or a similar custom path). This will create a symlink to the `aws` executable within that directory. `bash sudo ./aws/install --install-dir /usr/local/aws-cli-v2 --bin-dir /usr/local/bin/aws-v2` This command installs the CLI files into `/usr/local/aws-cli-v2` and creates a symlink `aws-v2` in `/usr/local/bin`. Your existing `aws` command (v1) should remain untouched.
- **For Windows:** Run the downloaded `AWSCLIV2.msi`. The installer typically handles pathing. By default, it installs to `C:\Program Files\Amazon\AWSCLIV2`. You can choose to install it side-by-side with v1, and then manage your `PATH` environment variable. To ensure v1 remains default, you might need to manually adjust your `PATH` to prioritize v1's location or rename the v2 executable. For side-by-side, the installer usually

adds `C:\Program Files\Amazon\AWSCLIV2\bin` to your path. If v1 is also in the path, the one listed first will be used.

3. Verify Installation and Path

After installation, verify that v2 is accessible via its new symlink and that v1 is still the default `aws` command.

- **For Linux/macOS:** `bash aws --version # Should show v1 aws-v2 --version # Should show v2` Expected output for `aws --version`: `aws-cli/1.x.x Python/x.x.x ...` Expected output for `aws-v2 --version`: `aws-cli/2.x.x Python/x.x.x ...`
- **For Windows:** Open a new Command Prompt or PowerShell window. `powershell aws --version # Check if this is v1 or v2 based on your PATH. # If v2 is default, you might need to call v1 explicitly if it's in a different path, e.g., # "C:\Program Files\Amazon\AWSCLI\bin\aws.exe" --version` > ⚡ Quick Note: If `aws --version` immediately shows v2 on Windows, it means v2's path took precedence. You'll need to adjust your system's `PATH` environment variable to put v1's installation directory before v2's, or temporarily rename the `aws.exe` in the v2 directory to `aws-v2.exe`.

4. (Optional) Remove AWS CLI v1 after Successful Migration

Once you are confident that all your scripts and workflows are fully migrated to v2, you can safely remove v1.

- **For Linux/macOS (if installed via pip):** `bash pip uninstall awscli` > ⚠️ Risk: Ensure `pip` is the correct package manager for your v1 installation. If v1 was installed via a system package manager (e.g., `apt`, `yum`), use that package manager to uninstall it.
- **For Windows:** Go to "Add or remove programs" in Windows Settings and uninstall "AWS Command Line Interface".

Identifying Breaking Changes with 'Upgrade Debug Mode'

AWS CLI v1's "upgrade debug mode" is a powerful feature that helps you identify potential breaking changes before you even switch to v2. It runs in the context of v1 but flags commands that would behave differently or fail in v2.

- **What it does:** When enabled, v1 CLI commands will print warnings to `stderr` if they detect syntax or parameters that are known to be problematic in v2. This allows you to proactively identify issues in your scripts.

- **How to use it:**

1. **Enable debug mode:** Set the `AWS_CLI_UPGRADE_DEBUG` environment variable to `1`. ```bash # For Linux/macOS export AWS_CLI_UPGRADE_DEBUG=1

For Windows (Command Prompt)

set AWS_CLI_UPGRADE_DEBUG=1

For Windows (PowerShell)

$env:AWS_CLI_UPGRADE_DEBUG=1 `2. **Run your existing v1 scripts/commands:**`bash

Example: Run a script that uses AWS CLI v1

./my_v1_aws_script.sh

Or run individual commands

aws s3api list-objects --bucket my-bucket --max-keys 10 `3. **Analyze the output:** The CLI will print warnings like:` UPGRADE_DEBUG: Parameter 'tag-specifications' expected a list of maps, but received a string. In AWS CLI v2, this will likely fail. Consider using 'file://' syntax for complex JSON inputs. ```- **Limitations:** * It does not detect all breaking changes. * It can sometimes produce false positives. * It only provides warnings; it doesn't fix anything.

> 📌 Key Idea: Use `upgrade debug mode` as a first pass to get a high-level overview of potential issues in your codebase without modifying any scripts.

Resolving Breaking Changes with the AWS CLI v1-to-v2 Migration Tool

The AWS CLI v1-to-v2 Migration Tool is a standalone utility designed to automate the identification and, in many cases, the resolution of compatibility issues in your bash scripts. It acts as a static linter and can even auto-update scripts.

- **What it does:** The tool scans your bash scripts for AWS CLI commands, identifies v1-specific syntax or deprecated parameters, and can suggest or automatically apply fixes for v2 compatibility.
- **Prerequisites:** Python 3.9 or higher is required to run the tool.
- **Step 1: Install the Migration Tool** `bash pip install aws-cli-v2-migrator` > ✅ Safe to skip if: You already have the tool installed.
- **Step 2: Scan your scripts for breaking changes** Navigate to the directory containing your scripts and run the tool in scan mode. `bash aws-cli-v2-migrator scan --path .` This command will analyze all `.sh` files (by default) in the current directory and its subdirectories, reporting any identified issues. The output will detail the file, line number, and the nature of the breaking change.
- **Step 3: Automatically fix identified issues (with caution)** The tool can attempt to automatically fix some common breaking changes. **Always review the changes before committing them.** `bash aws-cli-v2-migrator fix --path .` > ⚠️ Risk: Running `fix` without reviewing changes can introduce new issues or unintended modifications. Always use version control and review the diff carefully.
- **Step 4: Review and manually resolve remaining issues** After running the `fix` command, the tool might report issues it couldn't automatically resolve. These require manual intervention. `bash # Example output from scan/fix: # Detected breaking change in file: my_script.sh, line 15 # Issue: Parameter 'tag-specifications' requires JSON array format in v2. # Suggestion: Use 'file://' or ensure proper escaping.` For each reported issue, refer to the official AWS CLI v2 migration guide for detailed explanations and solutions: Migration guide for the AWS CLI version 2.

Migrating Existing Scripts and Workflows

Migrating scripts and workflows is the most labor-intensive part of the upgrade. A systematic approach is key to success.

1. **Inventory Your Scripts:** Identify all scripts, cron jobs, CI/CD pipelines, and automation tools that invoke `aws` commands. Prioritize them by criticality and frequency of use.
2. **Version Control:** Ensure all scripts are under version control. This is your primary safety net for rollback.
3. **Start Small:** Begin with less critical, simpler scripts. This allows you to gain experience with v2's nuances before tackling complex or production-critical workflows.
4. **Use `upgrade debug mode`:** Run your v1 scripts with `AWS_CLI_UPGRADE_DEBUG=1` enabled. Capture all warnings.
5. **Run the Migration Tool:** Apply `aws-cli-v2-migrator scan` and then `aws-cli-v2-migrator fix` on your scripts. Review the generated diffs meticulously.
6. **Update `aws` to `aws-v2`:** Once a script is modified for v2 compatibility, change the `aws` command within that script to `aws-v2` (or whatever alias you set up for v2). This ensures that the script explicitly uses v2. ```bash # Before (in your script) aws s3 ls s3://my-bucket

After (in your script, using the v2 alias)

   aws-v2 s3 ls s3://my-bucket ``` 7. **Manual Review and Refinement:**

- **Syntax:** Pay close attention to how JSON inputs, lists, and complex parameters are handled.
- **Output Parsing:** If scripts use `jq`, `grep`, `awk`, or other tools to parse CLI output, these are highly susceptible to breaking changes in JSON structure or text output.
- **Pagination Logic:** Verify that scripts expecting all results or specific page sizes still function correctly.
- **Error Handling:** Check if error messages or exit codes have changed and adjust error handling logic as needed. 8. **Iterate and Test:** Repeat the process for each script or workflow.

> ⚡ Real-world insight: For large codebases, consider a phased rollout. Migrate critical services first, then less critical ones. This limits the blast radius of any unforeseen issues.

Testing Your Migration

Thorough testing is non-negotiable for a production upgrade. Do not assume a script works just because the migration tool ran successfully.

1. **Unit Tests for Commands:** For individual `aws-v2` commands within your scripts, verify:
    - Correct execution with various parameters.
    - Expected output format and content.
    - Correct error handling (e.g., invalid input, permissions issues).
2. **Integration Tests for Workflows:** Test entire end-to-end workflows that involve multiple AWS CLI commands.
    - Ensure resources are created, modified, or deleted as expected.
    - Validate data consistency and integrity.
    - Check for correct sequencing and dependencies between commands.
3. **Smoke Tests:** Run critical, high-impact scripts in a non-production environment (e.g., staging) to quickly identify major regressions.
4. **Load Testing (if applicable):** If your scripts are part of high-throughput systems, test their performance under expected load with v2.
5. **Monitor Logs and Metrics:** After deploying to staging, closely monitor AWS CloudTrail logs, application logs, and system metrics for any anomalies or unexpected errors.
6. **Permissions Verification:** Ensure that the IAM roles/users used by the scripts still have the necessary permissions. While not a direct v2 change, sometimes stricter validation can surface previously masked permission issues.

> 🧠 Important: Document your test cases and expected outcomes. This ensures consistency and makes future debugging easier.

Common Troubleshooting Tips

Even with careful planning, you might encounter issues. Here are some common problems and their solutions:

Error 1: `aws: command not found` or still running v1 after installation

- **Error Message:** `bash: aws: command not found # OR aws --version # aws-cli/1.x.x Python/x.x.x ... (expecting v2)` - **Cause:** The `PATH` environment variable is not correctly configured to point to the AWS CLI v2 executable, or the v2 installation was not added to `PATH` or took lower precedence than v1.
- **Fix:**
  1. **Verify v2 installation path:** `bash # For Linux/macOS ls /usr/local/bin/aws-v2 # Or wherever you installed the symlink ls /usr/local/aws-cli-v2/bin/aws # Check the actual executable`
  2. **Adjust `PATH`:**
     - If you want `aws` to default to v2: Find the v2 installation directory (e.g., `/usr/local/aws-cli-v2/bin` on Linux/macOS, `C:\Program Files\Amazon\AWSCLIV2\bin` on Windows) and ensure it's at the beginning of your `PATH`.
     - If you're using the `aws-v2` alias, ensure `/usr/local/bin` (or your chosen `bin-dir`) is in your `PATH`.
     - Example (Linux/macOS, to make `aws` point to v2): `bash # Add to your shell profile (.bashrc, .zshrc) export PATH="/usr/local/aws-cli-v2/bin:$PATH" source ~/.bashrc # Or ~/.zshrc`
     - Example (Windows, adjust system PATH variables in Control Panel -> System -> Advanced system settings -> Environment Variables).

Error 2: Invalid parameter/syntax error

- **Error Message:** `Unknown options: --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=MyInstance}]' # OR Error parsing parameter 'tags': Invalid JSON: Expecting property name enclosed in double quotes: line 1 column 2 (char 1)` - **Cause:** AWS CLI v2 has stricter parsing for complex parameters like JSON strings or

lists. Improper escaping, missing quotes, or incorrect JSON format are common culprits.

- **Fix:**
  1. **Use `file://` syntax:** For complex JSON, save the JSON content to a file and reference it using `file://`. `bash # Create my-tags.json: # [{"ResourceType":"instance","Tags": [{"Key":"Name","Value":"MyInstance"}]}] aws-v2 ec2 run-instances --image-id ami-0abcdef1234567890 --instance-type t2.micro --tag-specifications file://my-tags.json`
  2. **Ensure proper quoting/escaping:** If passing JSON directly, ensure all keys and string values are double-quoted and the entire JSON string is appropriately quoted for your shell. `bash aws-v2 ec2 run-instances --image-id ami-0abcdef1234567890 --instance-type t2.micro --tag-specifications '[{"ResourceType":"instance","Tags": [{"Key":"Name","Value":"MyInstance"}]}]'`

Error 3: Pagination issues (e.g., missing results, too many calls)

- **Error Message:** Script only processes the first 1000 items, or makes too many API calls.
- **Cause:** v2's default client-side pagination might behave differently than v1's server-side pagination. Scripts relying on specific page sizes or manual iteration might break.
- **Fix:**
  1. **Review pagination flags:** Use `--no-paginate` if you only want the first page of results (v2 will fetch all by default). Use `--page-size` to control the number of items returned by each API call, and `--max-items` to limit the total number of items processed by the CLI.
  2. **Adjust script logic:** If your script explicitly iterated using `--starting-token` in v1, you might need to remove that logic if v2's client-side pagination handles it, or adjust it to work with v2's tokens.

Error 4: Output parsing failures (e.g., `jq` scripts breaking)

- **Error Message:** `jq: error (at <stdin>:1): Cannot index array with string "InstanceId" # OR Error: null (expected a string)` - **Cause:** Minor changes in the JSON output structure (e.g., a field moved, renamed, or nested differently) can break scripts that use `jq` or similar tools to parse the output.

- **Fix:**

    1. **Inspect v2 output:** Run the command with v2 and examine the full JSON output.
    `bash aws-v2 ec2 describe-instances --output json | less`
    2. **Update `jq` queries:** Adjust your `jq` filters to match the new JSON structure. For example, if `InstanceId` moved from `Reservations[].Instances[].InstanceId` to `Reservations[].Instances[].Properties.InstanceId`.

Rollback Plan

A robust rollback plan is essential for any production migration. If critical issues arise that cannot be quickly resolved, you must be able to revert to AWS CLI v1.

> ✅ Safe to skip if: You have confirmed the v2 migration is fully stable in production and all dependent systems are verified.

Rollback Strategy: Revert to AWS CLI v1 and restore scripts from version control.

1. **Revert Script Changes:**

- **Action:** Immediately revert all changes made to your scripts and automation workflows from your version control system (e.g., Git). This will restore the `aws` commands to their v1-compatible state.
- **Command (example with Git):** `bash git reset --hard HEAD # WARNING: This will discard ALL local changes git checkout <commit_hash_before_migration> .` > ⚠️ Risk: Ensure you know the exact commit hash or branch to revert to. Discarding local changes without proper version control can lead to data loss.

1. **Deactivate/Uninstall AWS CLI v2:**

- **Action (Linux/macOS):** If you installed v2 with a custom `bin-dir` (e.g., `aws-v2`), simply remove the symlink and adjust your `PATH` if you made `aws` point to v2. If you overwrote v1, you might need to uninstall v2 completely. `bash # If you created a symlink for aws-v2: sudo rm /usr/local/bin/aws-v2 # If you changed your PATH to make 'aws' point to v2, revert that change in your shell profile. # If v2 was installed to replace v1: sudo /usr/local/aws-cli-v2/v2/current/install --uninstall`

- **Action (Windows):** Uninstall "AWS Command Line Interface v2" from "Add or remove programs".
  `powershell # Open PowerShell as Administrator and run: # Get-WmiObject -Class Win32_Product | Where-Object {$_.Name -like "AWS Command Line Interface v2"} | ForEach-Object {$_.Uninstall()} # Or manually via Control Panel -> Programs and Features`

1. **Ensure AWS CLI v1 is Active:**

- **Action:** Verify that your original AWS CLI v1 installation is now the default `aws` command. If you uninstalled v1 during the migration, you will need to reinstall it.
- **Command (Linux/macOS, if v1 was uninstalled):** `bash pip install awscli --upgrade --user # Ensure ~/.local/bin is in your PATH export PATH=$HOME/.local/bin:$PATH`
- **Command (Windows, if v1 was uninstalled):** Download and run the v1 MSI installer from the official AWS documentation if you don't have a backup.

1. **Verify Rollback:**

- **Action:** Run a few critical AWS CLI commands to confirm that v1 is active and functioning correctly with your reverted scripts.
- **Command:** `bash aws --version # Should show v1 aws s3 ls s3://your-critical-bucket # Verify a critical command`

This guide provides a comprehensive framework for a safe and complete migration. Remember, diligence in testing and a clear understanding of the breaking changes are your best allies in ensuring a smooth transition to AWS CLI v2.