AWS CloudShell

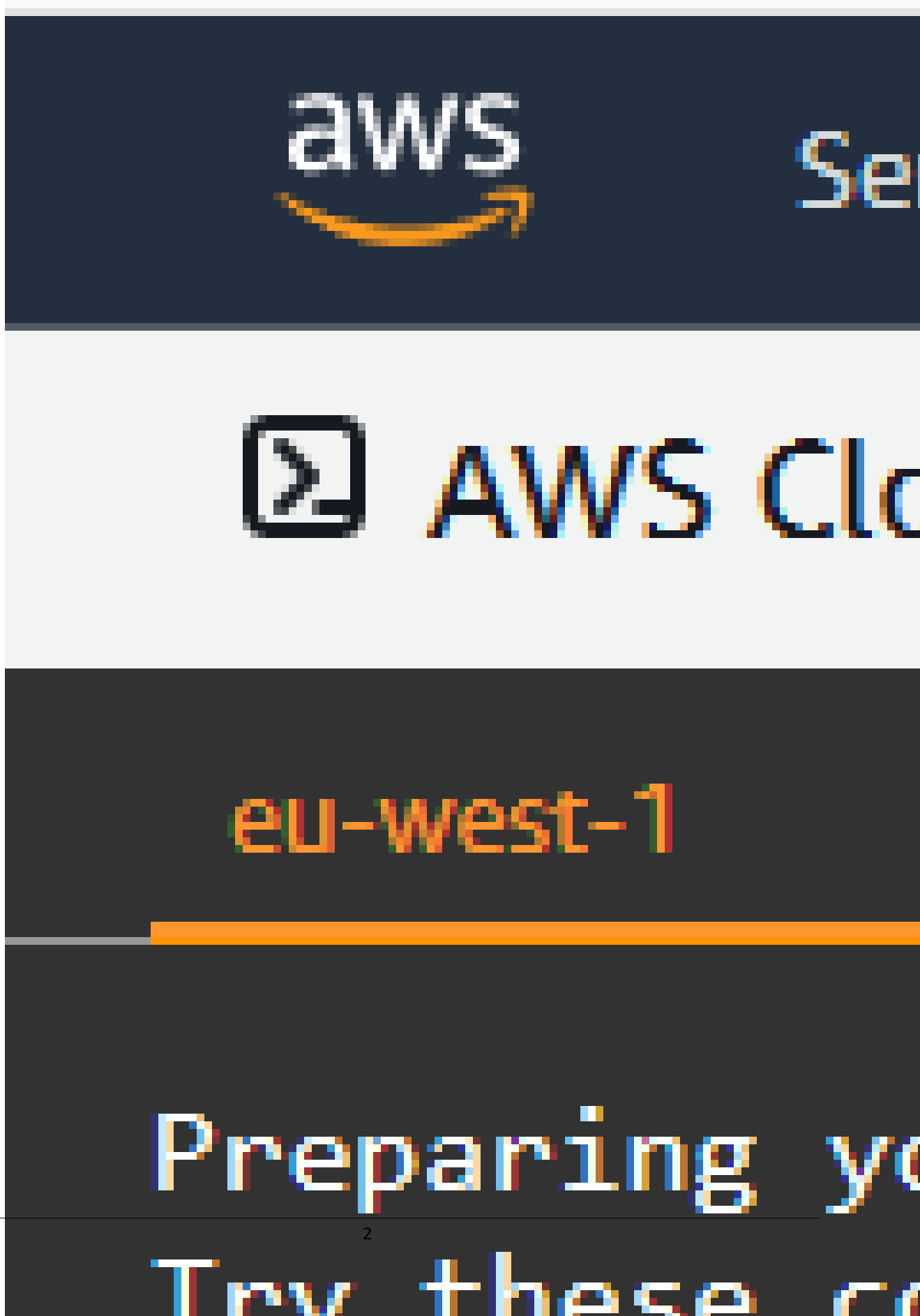
User Guide

AWS CloudShell: User Guide

Table of Contents

What is AWS CloudShell?

AWS CloudShell is a browser-based, pre-authenticated shell that you can launch directly (p. 4)from the AWS Management Console. You can run AWS CLI commands against AWS services using your preferred shell (Bash, PowerShell, or Z shell). And you can do this without needing to download or install command line tools.


aws
AWS Clo
eu-west-1
Preparing yo
Try these co

When you launch AWS CloudShell, a compute environment (p. 57) that's based on Amazon Linux 2 is created. Within this environment, you've access to an extensive range of pre-installed development tools (p. 57), options for uploading and downloading files (p. 35), and file storage that persists between sessions (p. 3).

(Try it now: Tutorial: Getting started with AWS CloudShell (p. 9).)

AWS CloudShell features

AWS Command Line Interface

You launch AWS CloudShell from the AWS Management Console, and the AWS credentials you used to sign in to the console are automatically available in a new shell session. This pre-authentication of AWS CloudShell users allows you to skip configuring credentials when interacting with AWS services using AWS CLI version 2 (pre-installed on the shell's compute environment).

For more information on interacting with AWS services using the command-line interface, see Working with AWS services in AWS CloudShell (p. 40).

Shells and development tools

With the shell that's created for AWS CloudShell sessions, you can switch seamlessly between your preferred command-line shells (p. 33). More specifically, you can switch between Bash, PowerShell, and Z shell. You also have access to pre-installed tools and utilities such as git, make, pip, sudo, tar, tmux, vim, wget, and zip.

The shell environment is pre-configured with support for leading software languages, enabling you to run Node.js and Python projects, for example, without first having to perform runtime installations. PowerShell users can use the .NET Core runtime.

Files created in or uploaded to AWS CloudShell can also be committed to a local repository before being pushed to a remote repository managed by AWS CodeCommit.

For more information, see AWS CloudShell compute environment: specifications and software (p. 57).

Persistent storage

When using AWS CloudShell you have persistent storage of 1 GB for each AWS Region at no additional cost. The persistent storage is located in your home directory (`$HOME`) and is private to you. Unlike ephemeral environment resources that are recycled after each shell session ends, data in your home directory persists between sessions.

For more information about the retention of data in persistent storage, see Persistent storage (p. 72).

Security

The AWS CloudShell environment and its users are protected by specific security features such as IAM permissions management, shell session restrictions, and Safe Paste for text input.

Permissions management with IAM

Administrators can grant and deny permissions to AWS CloudShell users using IAM policies. Administrators can also create policies that specify at a granular level the particular actions those users

can perform with the shell environment. For more information, see Managing AWS CloudShell access and usage with IAM policies (p. 50).

Shell session management

Inactive and long-running sessions are automatically stopped and recycled. For more information, see Shell sessions (p. 73).

Safe Paste for text input

Enabled by default, Safe Paste is a security feature that asks you to verify that multiline text that you're about to paste into the shell doesn't contain malicious scripts. For more information, see Using Safe Paste for multiline text (p. 44).

Customization options

Your AWS CloudShell experience can be customized by changing screen layouts (multiple tabs), text sizes, and light/dark interface themes. For more information, see Customizing your AWS CloudShell experience (p. 43).

You can also extend your shell environment by installing your own software (p. 63) and modifying start-up shell scripts (p. 64).

Pricing

AWS CloudShell is an AWS service that's available at no additional charge. You pay for any other AWS resources that you run with AWS CloudShell. Standard data transfer rates also apply.

For more information, see Limits and restrictions for AWS CloudShell (p. 72).

How do I get started?

To start working with the shell, sign in to the AWS Management Console and choose **AWS CloudShell** from the home page.

For a walkthrough of signing in to the AWS Management Console and performing key tasks with AWS CloudShell, see Tutorial: Getting started with AWS CloudShell (p. 9).

Key AWS CloudShell topics

AWS CloudShell FAQs

Answers to frequently asked questions about this AWS service.

How do I start with AWS CloudShell?

You can launch AWS CloudShell with a single click from the AWS Management Console. All that's required to get started is to sign in to the console using your AWS or IAM credentials at https://console.aws.amazon.com/console/home.. You can choose the AWS CloudShell link on the home page or enter "CloudShell" in the **Find Services** box.

For more information, see Tutorial: Getting started with AWS CloudShell (p. 9).

Back to list of FAQs (p. 6)

What permissions do I need to access AWS CloudShell?

Because you access AWS CloudShell from the AWS Management Console, you must be an IAM user who can provide a valid account alias or ID, user name, and password.

To launch AWS CloudShell from the console, you need to have the IAM permissions provided by an attached policy. For more information, see Managing AWS CloudShell access and usage with IAM policies (p. 50).

Back to list of FAQs (p. 6)

Which AWS Regions is AWS CloudShell available in?

Currently, AWS CloudShell is available in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (Oregon)
- Asia Pacific (Tokyo)
- Europe (Ireland)

Back to list of FAQs (p. 6)

What types of shell can I use in AWS CloudShell?

You can choose to run commands using the Bash shell, PowerShell, or the Z shell. To switch to a specific shell, at the command prompt just type the name of the shell program:

- `bash`: Use the Bash shell
- `pwsh`: Use PowerShell
- `zsh`: Use the Z shell

Back to list of FAQs (p. 6)

What web browsers can I use with AWS CloudShell?

AWS CloudShell supports the three latest versions of Google Chrome, Mozilla Firefox, Microsoft Edge, and Apple Safari.

Back to list of FAQs (p. 6)

What software is pre-installed on my shell environment?

With the shell that's created for AWS CloudShell sessions, you can switch seamlessly between their preferred command-line shells (p. 33) (Bash, PowerShell, and Z shell). They also have access to pre-installed tools and utilities such Make, pip, sudo, tar, tmux, Vim, Wget and Zip.

The shell environment is pre-configured with support for leading software languages. You can use it to run Node.js and Python projects, for example, without first having to perform runtime installations. PowerShell users can use the .NET Core runtime.

Files created using the shell or uploaded with the shell interface can be added to a version-controlled repository managed using a pre-installed version of Git.

For more information, see Pre-installed software (p. 57).

Back to list of FAQs (p. 6)

Can I install software that's not available in the shell environment?

Yes. AWS CloudShell users have sudo privileges so they have administrative rights to install software from the command line. For more information, see Installing third-party software on your shell environment (p. 63).

Back to list of FAQs (p. 6)

Can I restrict the actions that users can perform in AWS CloudShell?

Yes. For example, you can allow users to access AWS CloudShell but prevent them from uploading or downloading files within the shell environment. You can also even completely prevent them from accessing AWS CloudShell. For more information, see Managing AWS CloudShell access and usage with IAM policies (p. 50).

Back to list of FAQs (p. 6)

Tutorial: Getting started with AWS CloudShell

This introductory tutorial shows you how to launch AWS CloudShell and perform key tasks using the shell command line interface.

First, you'll sign in to the AWS Management Console and launch AWS CloudShell in a new browser window. You'll then select an AWS Region and a shell type to work with.

Next, you'll create a new folder in your home directory and upload a file to it from your local machine. You'll work on that file using a pre-installed editor before running it as a program from the command line. Finally, you'll call AWS CLI commands to create an Amazon S3 bucket and add your file as an object to it.

Prerequisites

IAM permissions

The quickest way to obtain permissions for AWS CloudShell is to attach the following AWS managed policy to your IAM identity (user, role, or group):

- **AWSCloudShellFullAccess**: Provides users with full access to AWS CloudShell and its features.

For this tutorial, you also interact with AWS services (in this case, creating an Amazon S3 bucket and adding an object to it.) So your IAM identity requires a policy that grants, at a minimum, the `s3:CreateBucket` and `s3:PutObject` permissions.

For more information, see Amazon S3 Actions in the *Amazon Simple Storage Service Developer Guide*.

Exercise file

This exercise also involves uploading and editing a file that's then run as a program from the command line interface. Open a text editor on your local machine and add the following code snippet:

```
import sys
x=int(sys.argv[1])
y=int(sys.argv[2])
sum=x+y
print("The sum is",sum)
```

Save the file with the name `add_prog.py`.

Contents

Step 1: Sign in to AWS Management Console

This step involves entering your IAM user information to access the AWS Management Console. If you're already in the console, skip to step 2 (p. 14).

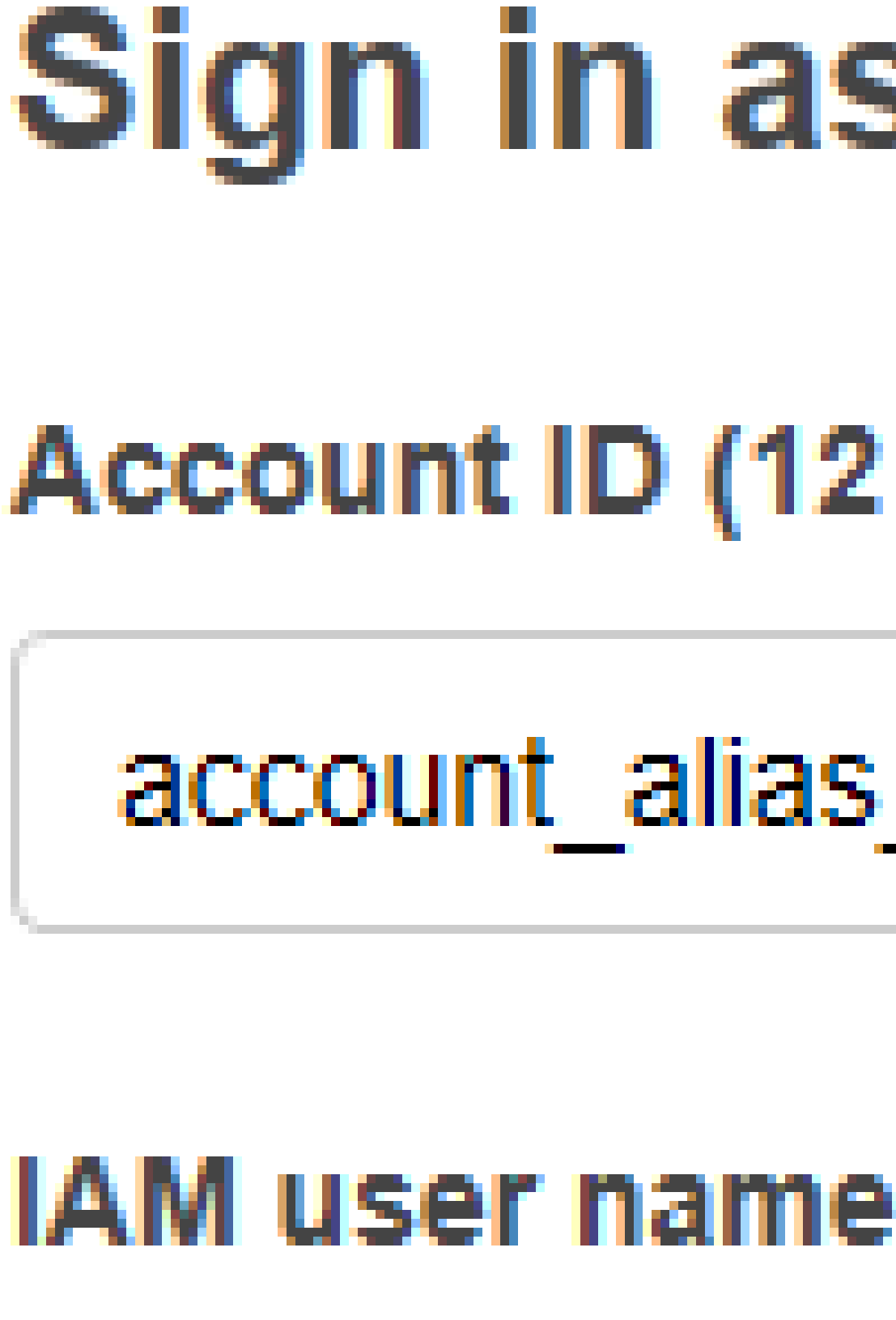
- You can access the AWS Management Console by using an IAM users sign-in URL or going to the main sign-in page.

  IAM user sign-in URL

  - Open a browser and enter the following sign-in URL, replacing `account_alias_or_id` with the account alias or account ID provided by your administrator:

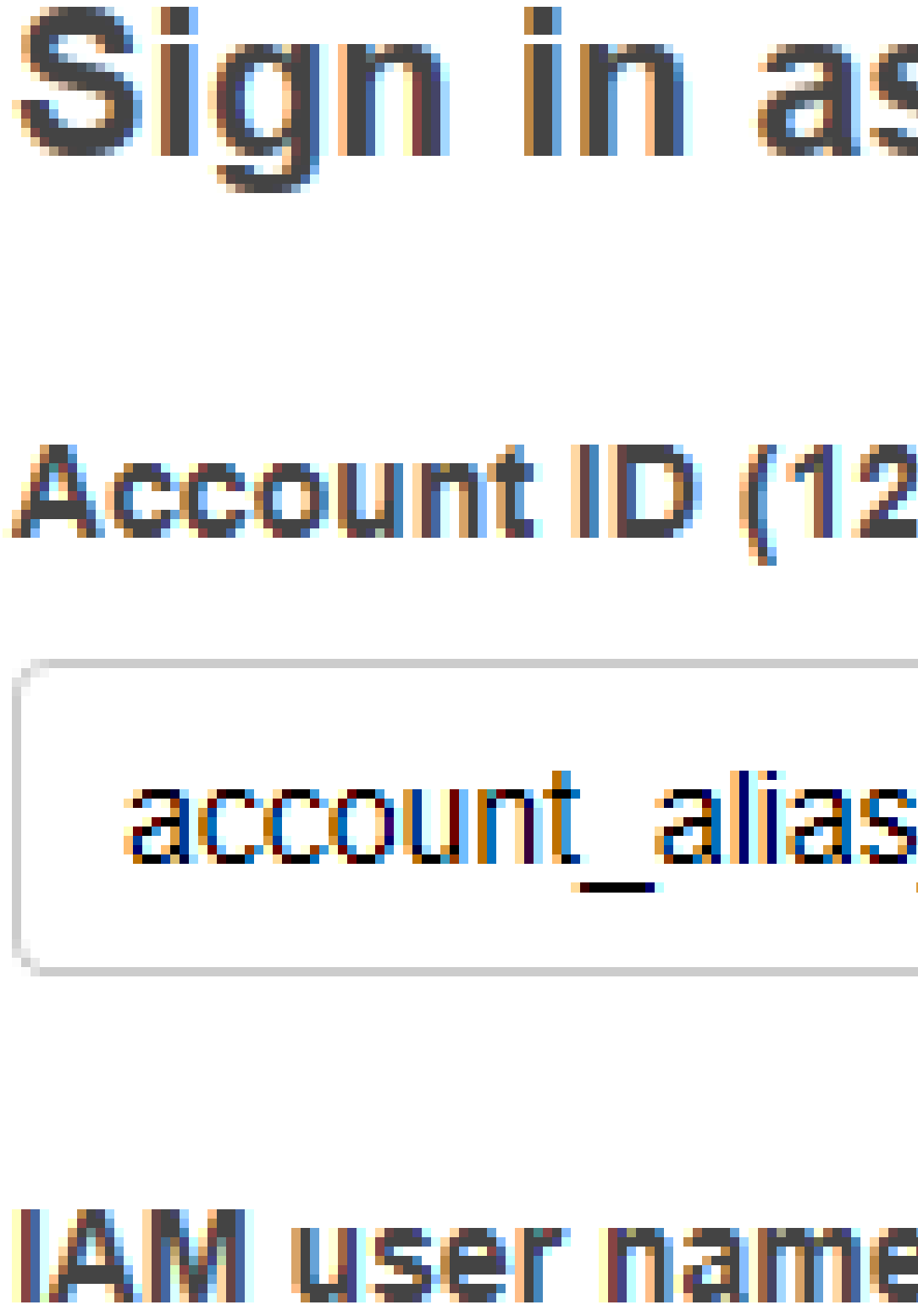
    ```
    https://account_alias_or_id.signin.aws.amazon.com/console/
    ```

  - Enter your IAM user name and password and choose **Sign in**.


Sign in as
Account ID (12
account_alias
IAM user name

Main sign-in page

- Open https://aws.amazon.com/console/.
- If you haven't signed in previously using this browser, the main sign-in page appears. Choose IAM user, enter the account alias or account ID, and choose **Next**.
- If you already signed in as an IAM user before. Your browser might remember the account alias or account ID for the AWS account. If so, enter your IAM user name and password and choose **Sign in**.


Sign in as
Account ID (12
account_alias
IAM user name

Note
You also have the option of signing in as a root user. This identity has complete access to all AWS services and resources in the account. We strongly recommend that you don't use the root user for everyday tasks, even administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user.

Step 2: Launch AWS CloudShell, select a Region, and choose a shell.

In this step, you launch AWS CloudShell from the console interface, choose an available AWS Region, and switch to your preferred shell (Bash, PowerShell, or Z shell).

1. From the AWS Management Console, you can launch AWS CloudShell by choosing the following options available on the navigation bar:

   - Choose the **AWS CloudShell** icon.
   - Start typing "cloudshell" in **Search** box and then choose the **CloudShell** option.

When AWS CloudShell launches in a new browser window for the first time, a welcome panel displays and lists key features. After you close this panel, status updates are provided while the shell configures and forwards your console credentials. When the command prompt displays, the shell is ready for interaction.

Note
If you encounter issues that prevent you from successfully launching or interacting with AWS CloudShell, check for information to identify and address those issues in Troubleshooting AWS CloudShell (p. 68).

2. To choose an AWS Region to work in, go to the **Select a Region** menu and select a supported AWS Region (p. 71) to work in. (Available Regions are highlighted.)

   Important
   If you switch Regions, the interface refreshes and the name of the selected AWS Region is displayed above the command line text. Any files that you add to persistent storage are available only in this same AWS Region. If you change Regions, different storage (and files) are accessible.

3. To choose a pre-installed shell to work with, enter its program name at the command line prompt:

   Bash

   ```
   bash
   ```

   If you switch to Bash, the symbol at the command prompt updates to `$`.

   Note
   Bash is the default shell that's running when you launch AWS CloudShell.

   PowerShell

   ```
   pwsh
   ```

   If you switch to PowerShell, the symbol at the command prompt updates to `PS>`.

   Z shell

   ```
   zsh
   ```

   If you switch to Z shell, the symbol at the command prompt updates to `%`.

Step 3: Upload a file to AWS CloudShell

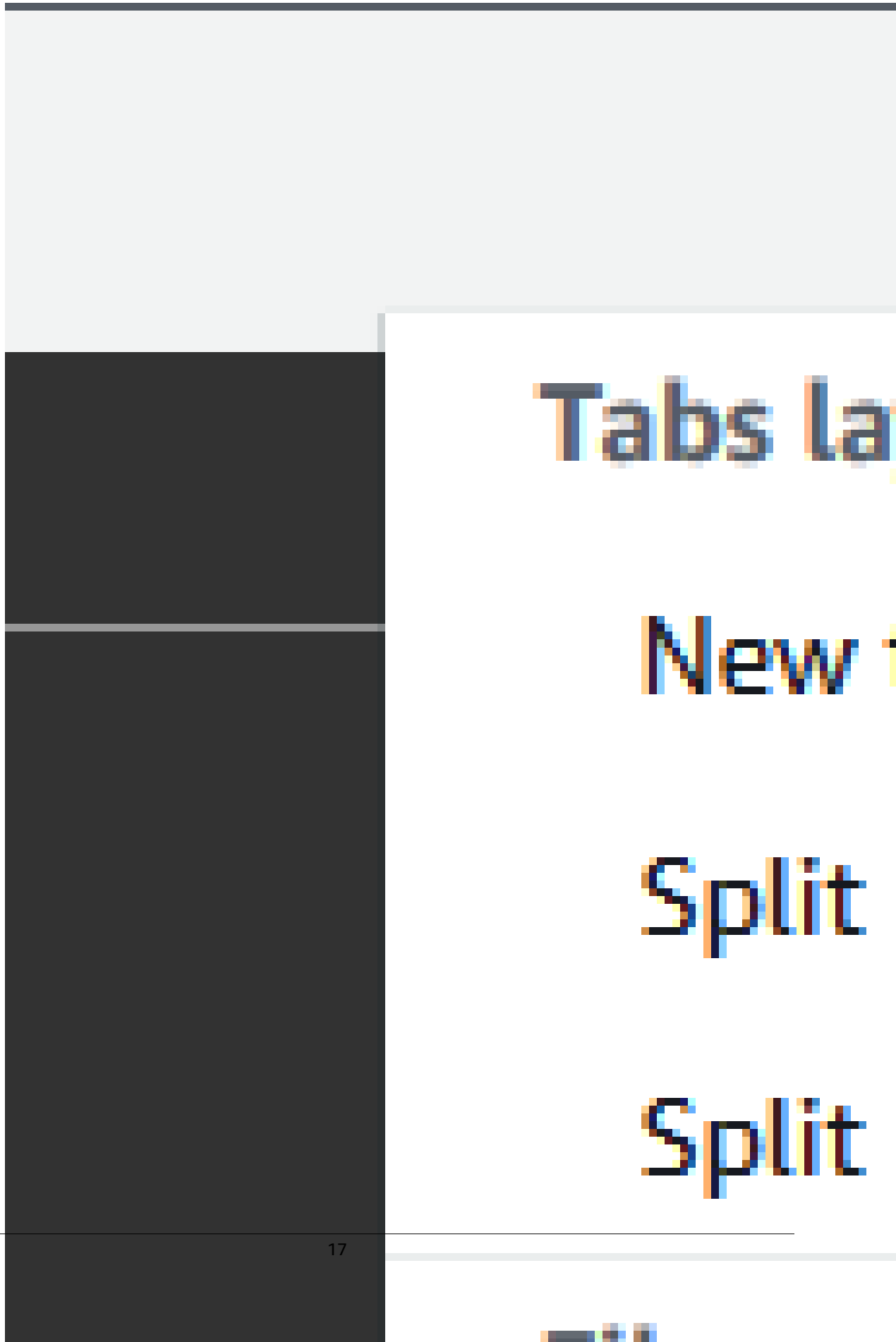
This step walks you through the process of uploading a file and then moving it to a new directory in your home directory.

1. To check your current working directory, at the prompt enter the following command:

   ```
   pwd
   ```

   When you press **Enter**, the shell returns your current working directory. For example: `/home/cloudshell-user`.

2. To upload a file to this directory, go to **Actions** and select **Upload file** from the menu.


Tabs la
New
Split
Split

The **Upload file** dialog box displays.

3. Choose **Browse**.
4. In your system's **File upload** dialog box, select the text file you created for this tutorial (`add_prog.py`) and choose **Open**.
5. In the **Upload file** dialog box, choose **Upload**.

   A progress bar tracks the upload. If the upload is successful, a message confirms that `add_prog.py` was added to the root of your home directory.
6. To create a directory for the file, enter the make directories command: `mkdir mysub_dir`.
7. To move the uploaded file from the root of your home directory to the new directory, use the `mv` command:

   `mv add_prog.py mysub_dir`.
8. To change your working directory to the new directory, enter `cd mysub_dir`.

   The command prompt updates to indicate you've changed your working directory.
9. To view the contents of the current directory, `mysub_dir`, enter the `ls` command.

   The contents of the working directory, including the file you just uploaded, are listed.

Step 4: Edit your file's code and run it from the command line

This step demonstrates how to use the pre-installed Vim editor to work with a file. You then run that file as a program from the command line.

1. To edit the file you uploaded in the previous step, enter the following command:

   ```
   vim add_prog.py
   ```

   The shell interface refreshes to display the Vim editor.
2. To edit the file in Vim, press the **I** key. Now edit the contents so the program adds up three numbers instead of two:

   ```
   import sys
   x=int(sys.argv[1])
   y=int(sys.argv[2])
   z=int(sys.argv[3])
   sum=x+y+z
   print("The sum is",sum)
   ```

   Note
   If you paste the text into the editor and have the Safe Paste feature (p. 44) enabled, a warning is displayed. Multiline text that's copied can contain malicious scripts. With Safe Paste, you can verify the complete text before it's pasted in. If you're satisfied that the text is safe, choose **Paste**.
3. After you edited the program, press **Esc** to enter the Vim command mode. Then, enter the following command to save the file and exit the editor:

   ```
   :wq
   ```

Note
If you're new to Vim, you might initially find it challenging to switch between command mode (used when saving files and exiting the application) and insert mode (used when inserting new text). As a friendly point of reference, to enter insert mode, press **I**, and, to enter command mode, press **Esc**. For more information about Vim and other tools available in AWS CloudShell, see Development tools and shell utilities (p. 60).

4. Back in the main command line interface, run the program and specify three numbers for input:

   ```
   python3 add_prog.py 4 5 6
   ```

   The command line displays the program output: `The sum is 15.`

Step 5: Use AWS CLI to add the file as an object in an Amazon S3 bucket.

In this step, you create an Amazon S3 bucket and then use the **PutObject** method to add your code file as an object in that bucket.

Note
In most cases, you can use a service such as CodeCommit (p. 24) to commit a software file into a version-controlled repository. This introductory tutorial uses Amazon S3 for storage to show how easy it is to use AWS CLI in AWS CloudShell. Using this method, you don't need to download or install any additional resource. Moreover, because you're already authenticated within the shell, you don't need to configure credentials before making calls.

1. To create a bucket in a specified AWS Region, enter the following command:

   ```
   aws s3api create-bucket --bucket insert-unique-bucket-name-here --region us-east-1
   ```

   If the call is successful, the command line displays a response from the service similar to the following output:

   ```
   {
       "Location": "/insert-unique-bucket-name-here"
   }
   ```

   Note
   If you don't adhere to the rules for naming buckets (using only lowercase letters, for example), the following error is displayed: An error occurred (InvalidBucketName) when calling the CreateBucket operation: The specified bucket is not valid.

2. To upload a file and add it as an object to the bucket that was just created, call the **PutObject** method:

   ```
   aws s3api put-object --bucket insert-unique-bucket-name-here --key add_prog --body
    add_prog.py
   ```

   If the object is successfully uploaded to the Amazon S3 bucket, the command line displays a response from the service similar to the following output:

   ```
   {
               "ETag": "\"ab123c1:w:wad4a567d8bfd9a1234ebeea56\""
   }
   ```

The `ETag` is the hash of the object that's been stored. It can be used to check the integrity of the object uploaded to Amazon S3.

Related topics

- Working with AWS services in AWS CloudShell (p. 40)
- Tutorial: Copying multiple files between your local machine and AWS CloudShell (p. 21)
- Tutorial: Using CodeCommit in AWS CloudShell (p. 24)
- Working with AWS CloudShell (p. 29)
- Customizing your AWS CloudShell experience (p. 43)

AWS CloudShell tutorials

The following tutorials show you how to perform tasks using AWS CloudShell.

Topics

Tutorial: Copying multiple files between your local machine and AWS CloudShell

Using the CloudShell interface, you can upload or download a single file between your local machine and the shell environment at a time. To copy multiple files between CloudShell and your local machine at the same time, use one of the following options:

- Amazon S3: Use S3 buckets as an intermediary when copying files between your local machine and CloudShell.
- Zip files: Compress multiple files in a single zipped folder that can be uploaded or downloaded using the CloudShell interface.

> **Note**
> Because AWS CloudShell doesn't allow incoming internet traffic, it's currently not possible to use commands such as `scp` or `rsync` to copy multiple files between local machines and the CloudShell compute environment.

Uploading and downloading multiple files using Amazon S3

Prerequisites

To work with buckets and objects, you need an IAM policy that grants permissions to perform the following Amazon S3 API actions:

- `s3:CreateBucket`
- `s3:PutObject`
- `s3:GetObject`

For a complete list of Amazon S3 actions, see Actions in the *Amazon Simple Storage Service API Reference*.

Upload multiple files to AWS CloudShell using Amazon S3

1. In AWS CloudShell, create an S3 bucket by running the following `s3` command:

```
aws s3api create-bucket --bucket your-bucket-name --region us-east-1
```

If the call is successful, the command line displays a response from the S3 service:

```
{
    "Location": "/your-bucket-name"
}
```

2. Next, you need to upload the files in a directory from your local machine to the bucket. You have two options for uploading files:

   - AWS Management Console: Use drag-and-drop to upload files and folders to a bucket.
   - AWS CLI: With the version of the tool installed on your local machine, use the command line to upload files and folders to the bucket.

   Using the console

   - Open the Amazon S3 console at https://console.aws.amazon.com/s3/.

     (If you're using AWS CloudShell, you should already be logged in to the console.)
   - In the **Buckets** list, choose the name of the bucket that you want to upload your folders or files to.
   - In a window other than the console window, select the files and folders that you want to upload. Then, drag and drop your selections into the console window that lists the objects in the destination bucket.

     The files you chose are listed on the **Upload** page.
   - Select the check boxes to indicate the files to be added.
   - Choose **Upload** to add the selected files to the bucket.

     Note
     For information about the full range of configuration options when using the console, see How do I upload files and folders to an S3 bucket? in the *Amazon Simple Storage Service Console User Guide*.

   Using AWS CLI

     Note
     For this option, you need to have the AWS CLI tool installed on your local machine and have your credentials configured for calls to AWS services. For more information, see the AWS Command Line Interface User Guide.

   - Launch the AWS CLI tool and run the following `aws s3` command to sync the specified bucket with the contents of the current directory on your local machine:

     ```
     aws s3 sync . s3://your-bucket-name
     ```

   If the sync is successful, upload messages are displayed for every object added to the bucket.

3. Next, return to the AWS CloudShell command line and enter the following command to synchronize the directory in the shell environment with the contents of the S3 bucket:

   ```
   aws s3 sync  s3://your-bucket-name .
   ```

   If the sync is successful, download messages are displayed for every file downloaded from the bucket to the directory.

Note
With the sync command, only new and updated files are recursively copied from the source directory to the destination.

Download multiple files from AWS CloudShell using Amazon S3

1. Using the AWS CloudShell command line, enter the following `aws s3` command to sync an S3 bucket with contents of the current directory in the shell environment:

   ```
   aws s3 sync . s3://your-bucket-name
   ```

   If the sync is successful, upload messages are displayed for every object added to the bucket.
2. Now you need to download the contents of the bucket to your local machine. Because the Amazon S3 console doesn't support the downloading of multiple objects, you need to use the AWS CLI tool that's installed on your local machine.

   From the command line of the AWS CLI tool, run the following command:

   ```
   aws s3 sync s3://your-bucket-name .
   ```

   If the sync is successful, the command line displays a download message for each file updated or added in the destination directory.

   Note
   For this option, you need to have the AWS CLI tool installed on your local machine and have your credentials configured for calls to AWS services. For more information, see the AWS Command Line Interface User Guide.

Uploading and downloading multiple files using zipped folders

With the zip/unzip utilities, you can compress multiple files in an archive that can be treated as a single file. The utilities are pre-installed in the CloudShell compute environment.

For more information about pre-installed tools, see Development tools and shell utilities (p. 60).

Upload multiple files to AWS CloudShell using zipped folders

1. On your local machine, add the files to be uploaded to a zipped folder.
2. Launch AWS CloudShell and then choose **Actions**, **Upload file**.
3. In the **Upload file** dialog box, choose **Select file** and choose the zipped folder you just created.
4. Next, in the **Upload file** dialog box, choose **Upload** to add the selected file to the shell environment.
5. Next, in the CloudShell command line, run the following command to unzip the contents of the zip archive to a specified directory:

   ```
   unzip zipped-files.zip -d my-unzipped-folder
   ```

Download multiple files from AWS CloudShell using zipped folders

1. In the CloudShell command line, running the following command to add all the files in the current directory to a zipped folder:

```
zip -r zipped-archive.zip *
```

2. Next, choose **Actions**, **Download file**.
3. In the **Download file** dialog box, enter the path for the zipped folder (`/home/cloudshell-user/zip-folder/zipped-archive.zip`, for example) and choose **Download**.

   If the path is correct, a browser dialog offers the choice of opening the zipped folder or saving it to your local machine.
4. On your local machine, you can now unzip the contents of the downloaded zipped folder.

Tutorial: Using CodeCommit in AWS CloudShell

CodeCommit is a secure, highly scalable, and managed source control service that hosts private Git repositories. Using AWS CloudShell, you can work with CodeCommit on the command line using **git-remote-codecommit**. This utility is pre-installed in the AWS CloudShell compute environment and provides a simple method for pushing and pulling code from CodeCommit repositories. It does this by extending Git. For more information, see the AWS CodeCommit User Guide.

This tutorial describes how to create a CodeCommit repository and clone it to your AWS CloudShell compute environment. This tutorial also shows how you can stage and commit a file to your cloned repository before pushing it to the remote repository that's managed in AWS Cloud.

Prerequisites

For information about the permissions that an IAM user requires to use AWS CloudShell, see the prerequisites section in the Getting started tutorial (p. 9). You also need IAM permissions to work with CodeCommit.

In addition, before starting you should have the following:

- A basic understanding of Git commands and version control concepts
- A file in the home directory of your shell that can be committed to the local and remote repositories. In this tutorial, it's referred to as `my-git-file`.

Step 1: Create and clone a CodeCommit repository

1. In the AWS CloudShell command line interface, enter the following `codecommit` command to create a CodeCommit repository called `MyDemoRepo`:

   ```
   aws codecommit create-repository --repository-name MyDemoRepo --repository-description
    "My demonstration repository"
   ```

   If the repository is successfully created, the command line displays the service's response:

   ```
   {
       "repositoryMetadata": {
           "accountId": "111122223333",
           "repositoryId": "0dcd29a8-941a-1111-1111-11111111111a",
           "repositoryName": "MyDemoRepo",
           "repositoryDescription": "My demonstration repository",
           "lastModifiedDate": "2020-11-23T20:38:23.068000+00:00",
           "creationDate": "2020-11-23T20:38:23.068000+00:00",
   ```

```
        "cloneUrlHttp": "https://git-codecommit.eu-west-1.amazonaws.com/v1/repos/
MyDemoRepo",
        "cloneUrlSsh": "ssh://git-codecommit.eu-west-1.amazonaws.com/v1/repos/
MyDemoRepo",
        "Arn": "arn:aws:codecommit:eu-west-1:111111111111:MyDemoRepo"
    }
 )
```

2. Using the command line, create a new directory for your local repository and make it your working directory:

```
mkdir my-shell-repo
cd my-shell-repo
```

3. Now clone the remote repository using the `git clone` command. (As you're working with **git-remote-codecommit**, use the HTTPS (GRC) URL style).

```
git clone codecommit::eu-west-1://MyDemoRepo
```

If the repository is successfully cloned, the command line displays the service's response:

```
Cloning into 'MyDemoRepo'...
warning: You appear to have cloned an empty repository.
```

4. To navigate to the cloned repository, use the `cd` command:

```
cd MyDemoRepo
```

Step 2: Stage and commit a file before pushing it to your CodeCommit repository

1. Add a file called `my-git-file` to the `MyDemoRepo` folder using either a Vim editor or the file upload feature of AWS CloudShell. To learn how to use both, see the Getting started tutorial (p. 9).
2. To stage your file in the repository, run the git `add` command:

```
git add my-git-file
```

3. To check that the file has been staged and is ready to be committed, run the git `status` command:

```
git status
```

`my-git-file` is listed as a new file and displays in green text, indicating it's ready to be committed.

4. Now commit this version of the staged file to the repository:

```
git commit -m "first commit to repo"
```

Note
If you're asked for configuration information to complete the commit, use the following format:

```
$ git config --global user.name "Jane Doe"
$ git config --global user.email janedoe@example.com
```

5. Last, to sync your remote repository with the changes made in your local one, push the changes to the upstream branch:

```
git push
```

Tutorial: Creating a presigned URL for Amazon S3 objects using AWS CloudShell

This topic shows you how to create a presigned URL to share an Amazon S3 object with others. Because object owners specify their own security credentials when sharing, anyone who receives the presigned URL can access the object for a limited time.

Prerequisites

- IAM user with access permissions provided by the **AWSCloudShellFullAccess** policy.
- For IAM permissions required to create a presigned URL, see Share an object with others in the *Amazon Simple Storage Service Developer Guide*.

Step 1: Create an IAM role to grant access to Amazon S3 Bucket

1. To get your IAM details that can be shared, call the `get-caller-identity` command from AWS CloudShell:

```
aws sts get-caller-identity
```

   If the call is successful, the command line displays a response similar to this:

```
{
    "Account": "123456789012",
    "UserId": "AROAXXOZUUOTTWDCVIDZ2:redirect_session",
    "Arn": "arn:aws:sts::531421766567:assumed-role/Feder08/redirect_session"
}
```

2. Take the user information that you obtained in the previous step, and add it to an AWS CloudFormation template. This template creates an IAM role. This role grants your collaborator least-privilege permissions for the shared resources.

```
Resources:
  CollaboratorRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              AWS: "arn:aws:iam::531421766567:role/Feder08"
            Action: "sts:AssumeRole"
      Description: Role used by my collaborators
      MaxSessionDuration: 7200
```

```
  CollaboratorPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - 's3:*'
            Resource: 'arn:aws:s3:::shared-bucket-for-my-cool-startup'
            Condition:
              StringEquals:
                s3:prefix:
                 - "myfolder/*"
      PolicyName: S3ReadSpecificFolder
      Roles:
        - !Ref CollaboratorRole
Outputs:
  CollaboratorRoleArn:
    Description: Arn for the Collaborator's Role
    Value: !GetAtt CollaboratorRole.Arn
```

3. Save the AWS CloudFormation template in a file called `template.yaml`.
4. Now use the template to deploy the stack and create the IAM role by calling the `deploy` command:

```
aws cloudformation deploy --template-file ./template.yaml --stack-name CollaboratorRole
 --capabilities CAPABILITY_IAM
```

Generate the presigned URL

1. Using your editor in AWS CloudShell, add the following code. This code creates a URL that provides federated users with direct access to the AWS Management Console.

```
import urllib, json, sys
import requests
import boto3
import os

def main():
  sts_client = boto3.client('sts')
  assume_role_response = sts_client.assume_role(
      RoleArn=os.environ.get(ROLE_ARN),
      RoleSessionName="collaborator-session"
  )
  credentials = assume_role_response['Credentials']
  url_credentials = {}
  url_credentials['sessionId'] = credentials.get('AccessKeyId')
  url_credentials['sessionKey'] = credentials.get('SecretAccessKey')
  url_credentials['sessionToken'] = credentials.get('SessionToken')
  json_string_with_temp_credentials = json.dumps(url_credentials)
  print(f"json string {json_string_with_temp_credentials}")

  request_parameters = f"?
Action=getSigninToken&Session={urllib.parse.quote(json_string_with_temp_credentials)}"
  request_url = "https://signin.aws.amazon.com/federation" + request_parameters
  r = requests.get(request_url)
  signin_token = json.loads(r.text)
  request_parameters = "?Action=login"
  request_parameters += "&Issuer=Example.org"
  request_parameters += "&Destination=" + urllib.parse.quote("https://us-
west-2.console.aws.amazon.com/cloudshell")
```

```
  request_parameters += "&SigninToken=" + signin_token["SigninToken"]
  request_url = "https://signin.aws.amazon.com/federation" + request_parameters

  # Send final URL to stdout
  print (request_url)

if __name__ == "__main__":
  main()
```

2. Save the code in a file called `share.py`.
3. Last, run the following from the command line to retrieve the ARN of the IAM role from AWS CloudFormation. Then, use it in the Python script to obtain temporary security credentials:

```
ROLE_ARN=$(aws cloudformation describe-stacks --stack-name CollaboratorRole --query
 "Stacks[*].Outputs[?OutputKey=='CollaboratorRoleArn'].OutputValue" --output text)
 python3 ./share.py
```

The script returns a URL that a collaborator can click to take them to AWS CloudShell in AWS Management Console. The collaborator has full control over the `myfolder/` folder in the Amazon S3 bucket for the next 3,600 seconds (1 hour). The credentials expire after an hour. After this time, the collaborator can no longer access the bucket.

Working with AWS CloudShell

This topic describes how you can access AWS CloudShell, navigate the shell interface, choose a shell type, work with file uploads and downloads, and manage your shell sessions.

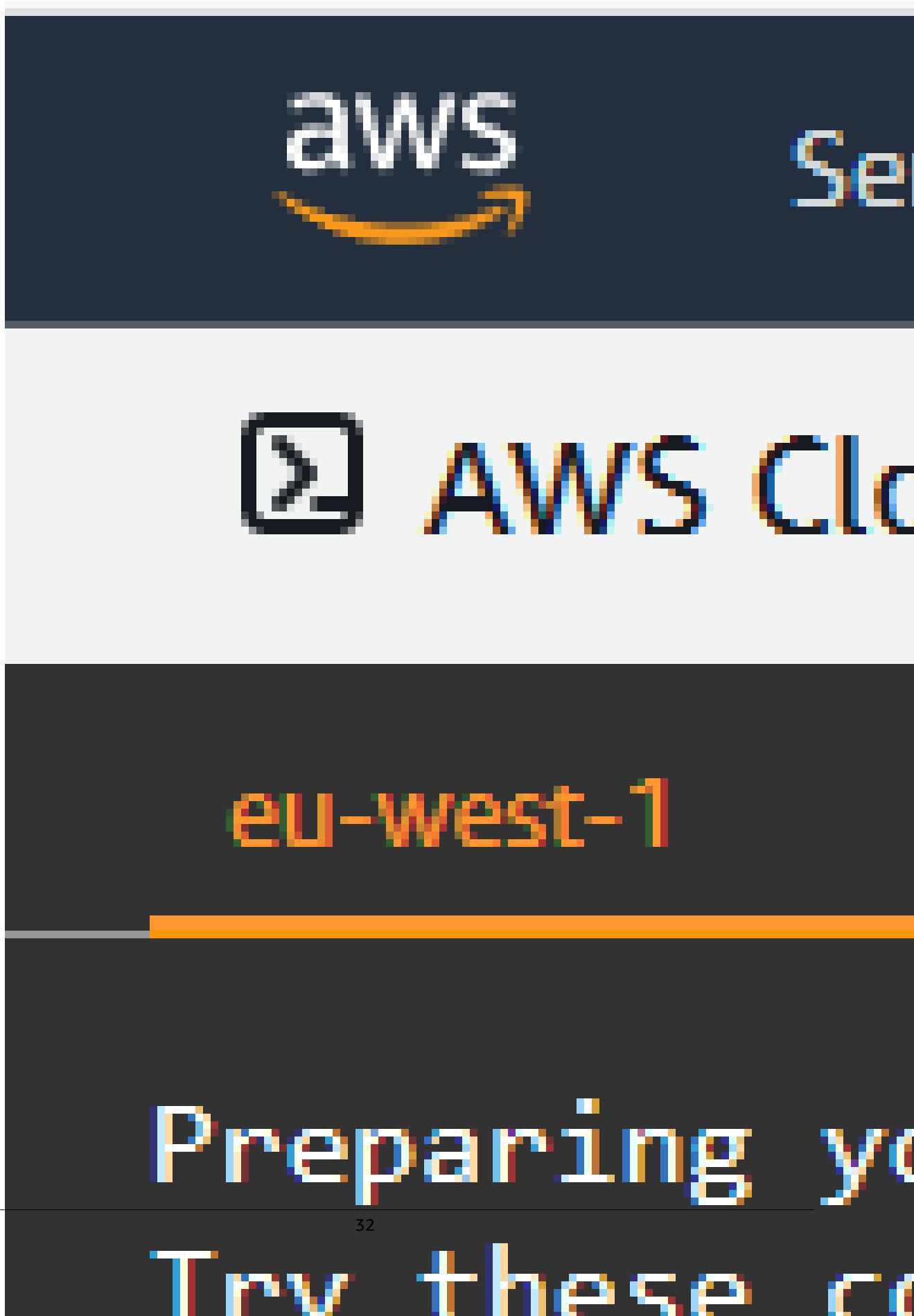
Launching AWS CloudShell

You can launch AWS CloudShell from the AWS Management Console using either one of the following two methods:

1. Choose the AWS CloudShell icon on the console navigation bar.
2. Start typing "cloudshell" in the **Find Services** box and then choose the **CloudShell** option.

Navigating the AWS CloudShell interface

The following screenshot indicates several key AWS CloudShell interface features. You use these features when you use the command line, use file upload/download tools, customize your shell environment, and access user assistance.


aws
eu-west-1

1. AWS CloudShell command line interface that you use to run commands using your preferred shell (p. 33). The current shell type is indicated by the command prompt.
2. The AWS Region where AWS CloudShell is currently running.
3. The Notification link, which provides access to your Personal Health Dashboard and event logs.
4. The name and account ID of the current AWS CloudShell user (blurred here). You can use this menu to view account information, switch roles, and sign out of AWS.
5. The **Select a Region** menu where you can choose which AWS Region your shell environment runs in. For more information, see Supported AWS Regions for AWS CloudShell (p. 71).
6. The **Support** menu, which you use to access user-assistance resources, including documentation and training.
7. The **Actions** menu, which provides options for changing the screen layout (p. 43), downloading and uploading files (p. 35), and restarting your AWS CloudShell (p. 39).
8. The **Preferences** option, which you can use to customize your shell experience (p. 43). You do this by changing text size and switching the interface's color theme.
9. The bottom bar, which has options to provide feedback, access localized versions of the interface, and learn about our privacy policy and terms of use.

Choosing your shell

The following shells are pre-installed and ready for use in AWS CloudShell: the Bash shell, PowerShell, and the Z shell. For information about the versions pre-installed in your shell environment, see the shells table (p. 57) in the AWS CloudShell compute environment (p. 57) section.

You can identify the shell that you're currently in by the command prompt:

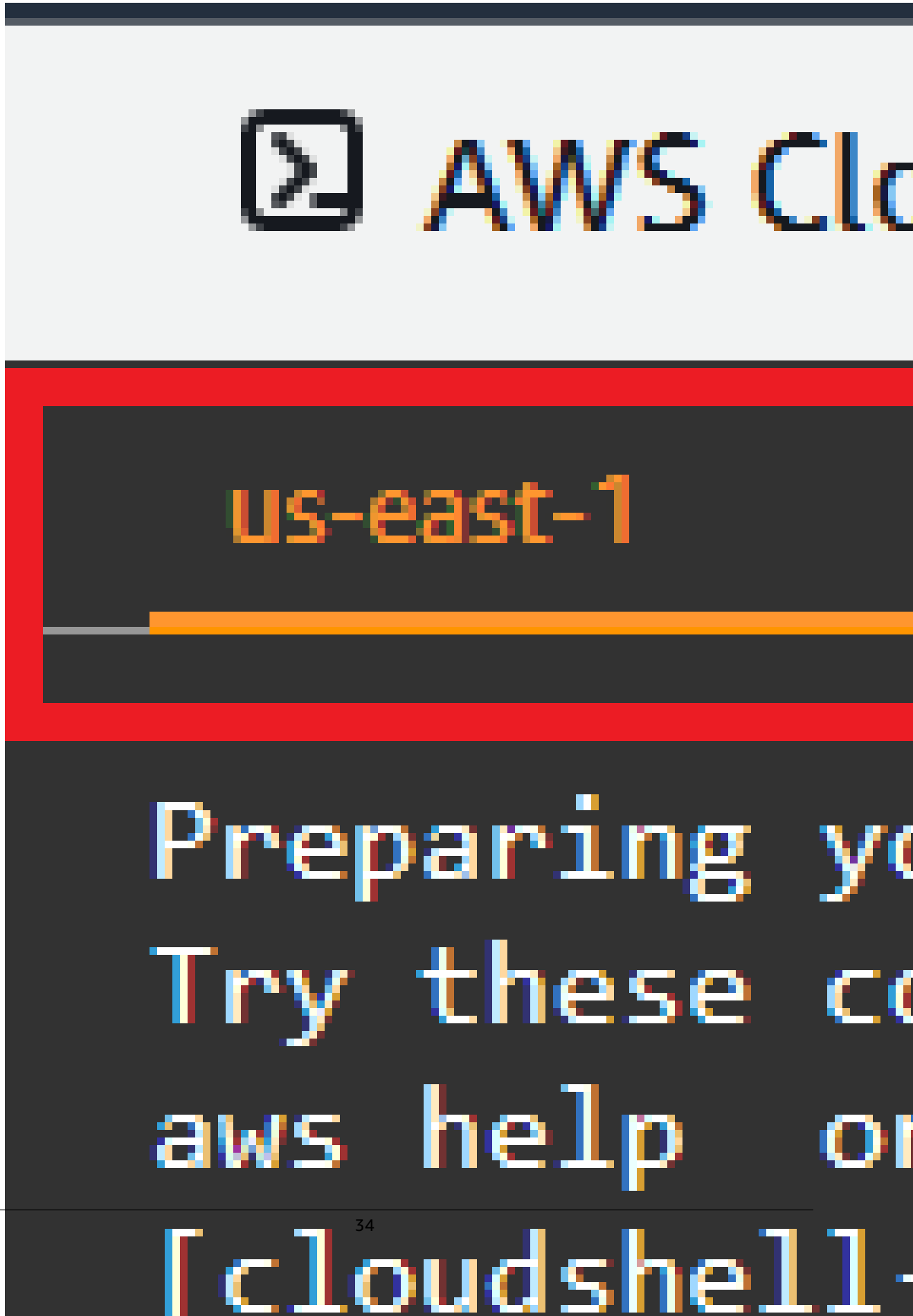
- `$`: Bash shell
- `PS>`: PowerShell
- `%`: Z shell

To switch to a new shell, enter the shell's program name at the command line prompt:

- `bash`: Bash
- `pwsh`: PowerShell
- `zsh`: Z shell

Working in AWS Regions

The current AWS Region that you're running in is displayed above the command line interface.


AWS Cl
us-east-1
Preparing yo
Try these co
aws help or
[cloudshell-

You can choose an AWS Region to work in by selecting one from the **Select a Region** menu. After you change Regions, the interface refreshes as your shell session connects to a compute environment that's running in the selected Region.

> **Important**
> You have persistent storage of 1 GB for each AWS Region. The persistent storage is located in your home directory (`$HOME`). Therefore, the personal files, directories, programs, and scripts that are stored in your home directory in one AWS Region. Moreover, they are different from those that are located in the home directory and stored a different Region.
> Long-term retention of files in persistent storage is also managed on a per-Region basis. For more information, see Persistent storage (p. 72).

Working with files and storage

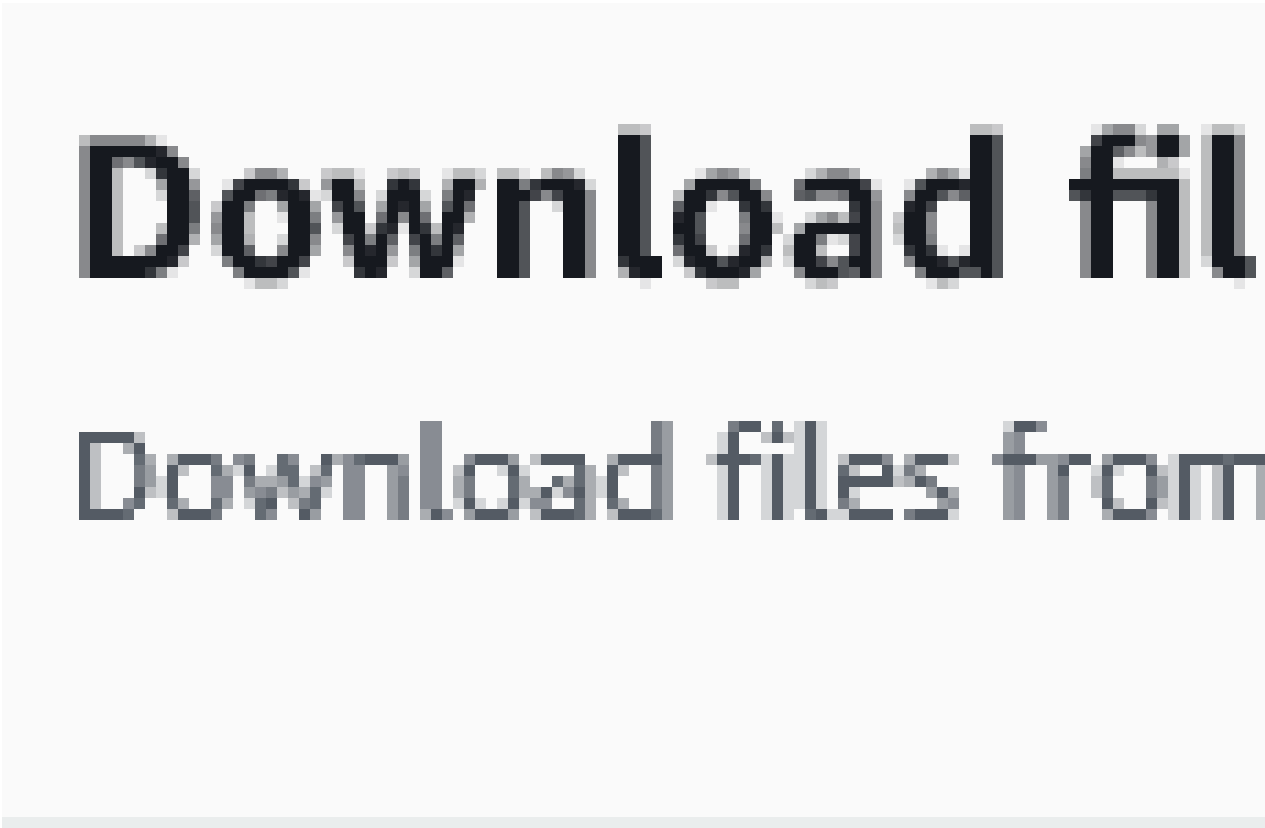
Using AWS CloudShell's interface, you can upload files to and download files from the shell environment. To ensure any of the files you add are available after your session ends, you should know the difference between persistent and temporary storage:

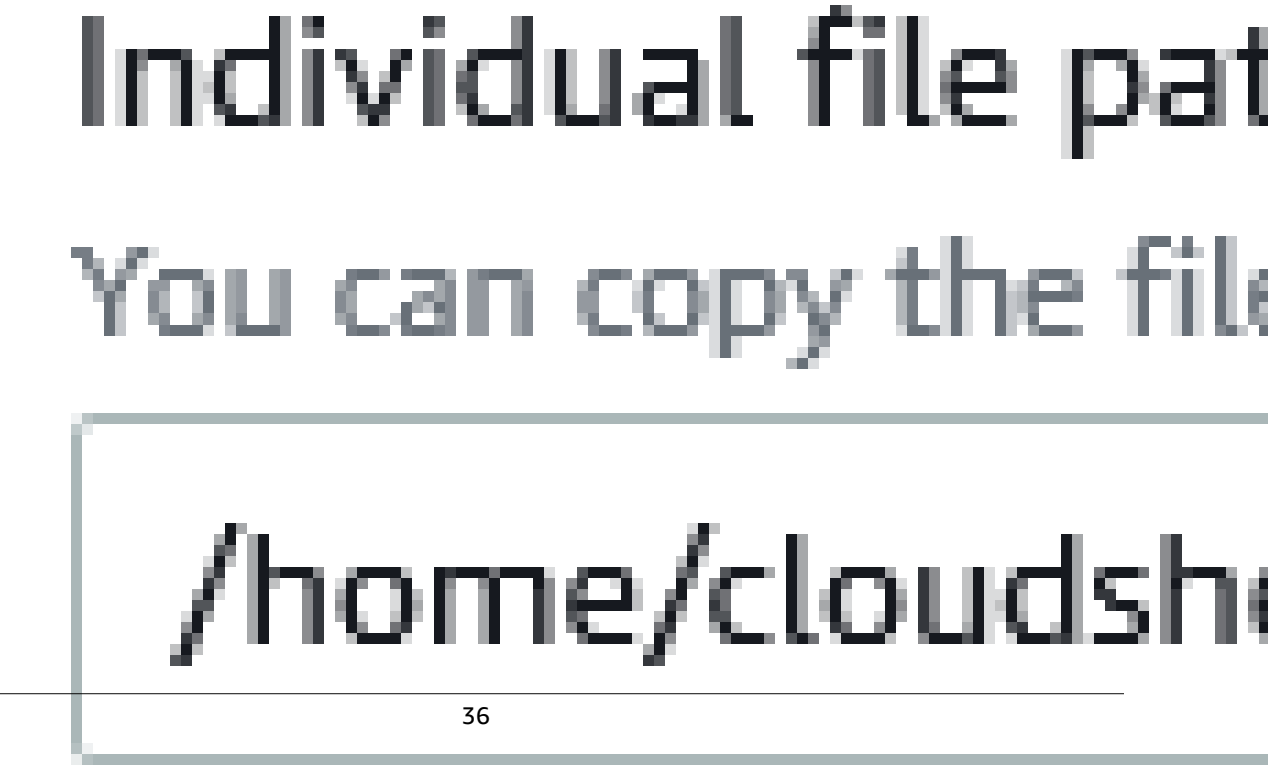
- **Persistent storage:** You have 1 GB of persistent storage for each AWS Region. Persistent storage is located on your home directory.
- **Temporary storage:** Temporary storage exists in directories outside your home directory and is recycled at the end of a session.

> **Important**
> Files you want to keep and use for future shell sessions should always be left in your home directory. If you move a file out of your home directory (using the `mv` command, for example), it's recycled when the current shell session ends.

To download files from AWS CloudShell

1. Choose **Actions**, **Download file**.
2. In the **Download file** dialog box, enter the path for the file to be downloaded.


Download fil
Download files from


Individual file pat
You can copy the file
/home/cloudshe

Note
You can use absolute or relative paths when specifying a file for download. With relative pathnames, `/home/cloudshell-user/` is added automatically to the start by default. So to download a file called `mydownload-file`, both of the following are valid paths:

- **Absolute path:** `/home/cloudshell-user/subfolder/mydownloadfile.txt`
- **Relative path:** `subfolder/mydownloadfile.txt`

3. Choose **Download**.

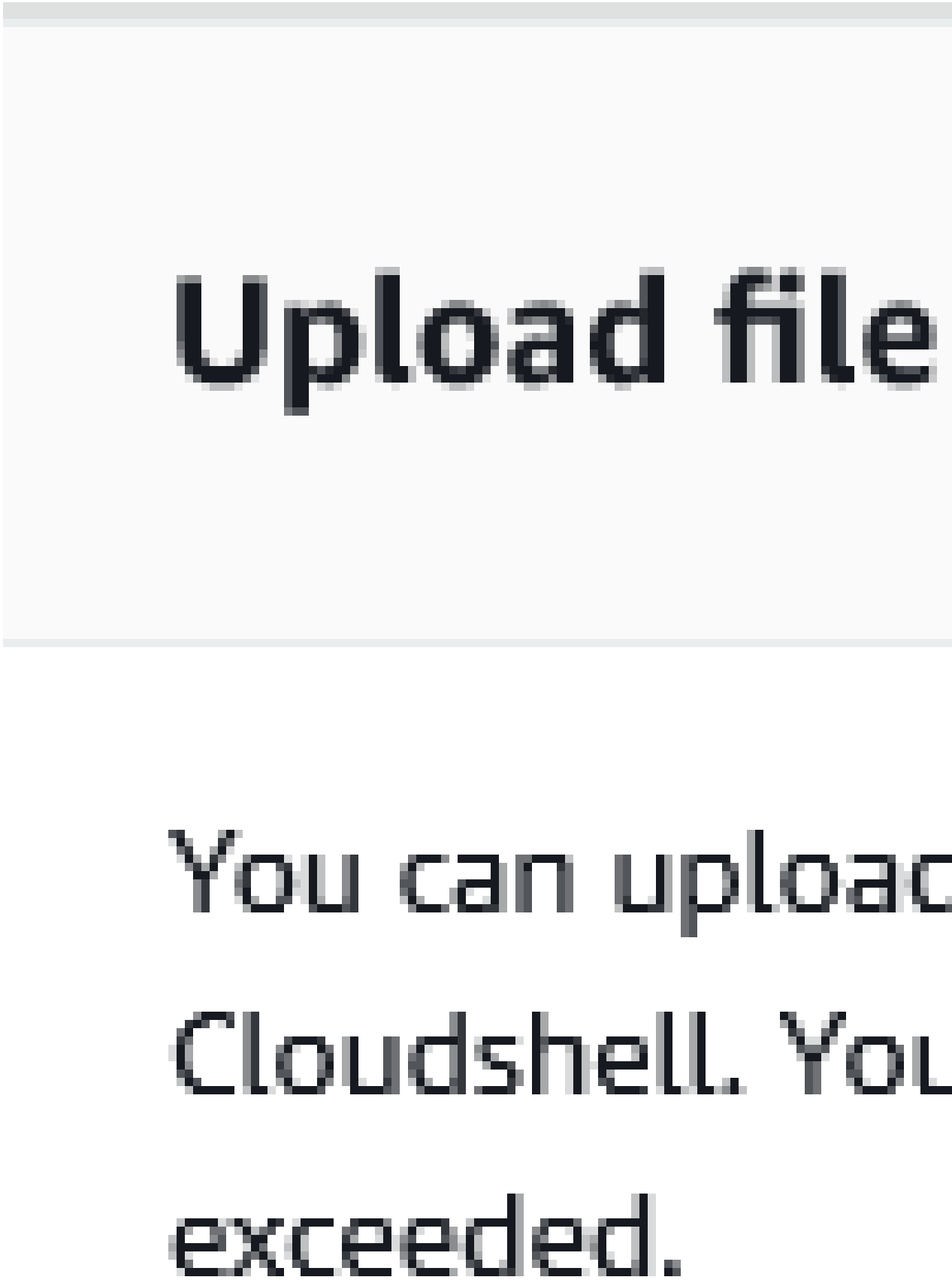
   If the file path is correct, a dialog box displays. You can use this dialog box to open the file with the default application. Or you can save the file to a folder on your local machine.

To upload files to AWS CloudShell

1. Choose **Actions**, **Upload file**.

   Note
   By default, files are uploaded to the root of your home directory: `/home/cloudshell-user`. If you move files (with the `mv` command, for example), ensure that the files stay in your home directory. This is where your 1 GB of persistent storage is located.

2. In the **Upload file** dialog box, choose **Browse** to select a file on your local machine. (The maximum upload size is 1 GB.)
3. In your system **Upload file** dialog box, select a file to upload and choose **Open**.
4. Choose **Upload** to add the selected file to the shell environment.


Upload file
You can uploa
Cloudshell. You
exceeded.

If you try to upload a file that causes you to exceed your per-Region limit, you're notified that the upload can't continue. For example, you're told that the upload can't complete if you try to upload a 200MB file when you've already stored 900MB of data in your home directory.

To remove files from AWS CloudShell

1. To remove files from AWS CloudShell, use standard shell commands such as `rm` (remove):

   ```
   rm my-file-for-removal
   ```

2. You can also remove multiple files that meet specified criteria using the `find` command. The following example removes all files that feature the phrase ".pdf" in their names:

   ```
   find -type f -name '*pdf*' -delete
   ```

   Note
   If you stop using AWS CloudShell in an AWS Region, data in that Region's persistent storage is removed automatically after a defined period. For more information, see Persistent storage (p. 72).

Starting and ending shell sessions

Note
As a security measure, if you don't interact with the shell using the keyboard or pointer for an extended period, the session stops automatically. Very long-running sessions are also automatically stopped. For more information, see Shell sessions (p. 73).

Restarting shell sessions

1. You can restart a shell session by choosing **Actions**, **Restart CloudShell**.

   You're notified that restarting AWS CloudShell stops all active sessions in the current AWS Region.

2. Choose **Restart** to confirm.

   An interface displays a message that the AWS CloudShell compute environment is stopping. After the environment has stopped and restarted, you can start working with the command line in a new session.

   Note
   In some cases, it may take a few minutes for your environment to restart.

Manually exiting shell sessions

With the command line, you can leave a shell session and log out using the `exit` command. You can then press any key to reconnect and continue to use AWS CloudShell.

Working with AWS services in AWS CloudShell

A key benefit of AWS CloudShell is that you can use it to manage your AWS services from the command line interface. This means that you don't need to download and install tools or configure your credentials locally beforehand. When you launch AWS CloudShell, a compute environment is created that has the following AWS command line tools already installed:

- AWS CLI (p. 40)
- AWS Elastic Beanstalk CLI (p. 41)
- Amazon ECS CLI (p. 42)
- AWS SAM (p. 42)

And because you've already signed into AWS, there's no requirement to configure your credentials locally before using services. The credentials you used to sign in to the AWS Management Console are forwarded to AWS CloudShell.

The rest of this topic demonstrates how you can start using AWS CloudShell to interact with selected AWS services from the command line.

AWS CLI command line examples for selected AWS services

The following examples represent only some of the numerous AWS services that you can work with using commands available from AWS CLI Version 2. For a full listing, see the AWS CLI Command Reference.

- DynamoDB (p. 40)
- AWS Cloud9 (p. 41)
- Amazon EC2 (p. 41)
- S3 Glacier (p. 41)

DynamoDB

DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. This service's implementation of the NoSQL mode supports key-value and document data structures.

The following `create-table` command creates a NoSQL-style table that's named `MusicCollection` in your AWS account.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
```

```
    --tags Key=Owner,Value=blueTeam
```

For more information, see Using DynamoDB with the AWS CLI in the *AWS Command Line Interface User Guide*.

AWS Cloud9

AWS Cloud9 is a cloud-based integrated development environment (IDE) that you can use to write, run, and debug your code in a browser window. The environment features a code editor, debugger, and terminal.

The following `create-environment-ec2` command creates an AWS Cloud9 EC2 development environment with the specified settings. It launches an Amazon EC2 instance, and then connects from the instance to the environment.

```
aws cloud9 create-environment-ec2 --name my-demo-env --description "My demonstration
 development environment." --instance-type t2.micro --subnet-id subnet-1fab8aEX --
automatic-stop-time-minutes 60 --owner-arn arn:aws:iam::123456789012:user/MyDemoUser
```

For more information, see AWS Cloud9 command-line reference.

Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure and resizable compute capacity in the cloud. It's designed to make web-scale cloud computing easier and more accessible.

The following `run-instances` command launches a t2.micro instance in the specified subnet of a VPC:

```
aws ec2 run-instances --image-id ami-xxxxxxxx --count 1 --instance-type t2.micro --key-name
 MyKeyPair --security-group-ids sg-903004f8 --subnet-id subnet-6e7f829e
```

For more information, see Using Amazon EC2 with the AWS CLI in the *AWS Command Line Interface User Guide*.

S3 Glacier

S3 Glacier and S3 Glacier Deep Archive are a secure, durable, and extremely low-cost Amazon S3 cloud storage classes for data archiving and long-term backup.

The following `create-vault` command creates a vault—a container for storing archives:

```
aws glacier create-vault --vault-name my-vault --account-id -
```

For more information, see Using Amazon S3 Glacier with the AWS CLI in the *AWS Command Line Interface User Guide*.

AWS Elastic Beanstalk CLI

The AWS Elastic Beanstalk CLI provides a command line interface made to simplify creating, updating, and monitoring environments from a local repository. In this context, an *environment* refers to a collection of AWS resources running an application version.

The following `create` command creates a new environment in a custom Amazon Virtual Private Cloud (VPC).

```
$ eb create dev-vpc --vpc.id vpc-0ce8dd99 --vpc.elbsubnets subnet-b356d7c6,subnet-02f74b0c
 --vpc.ec2subnets subnet-0bb7f0cd,subnet-3b6697c1 --vpc.securitygroup sg-70cff265
```

For more information, see the EB CLI command reference in the *AWS Elastic Beanstalk Developer Guide*.

Amazon ECS CLI

The Amazon Elastic Container Service (Amazon ECS) command line interface (CLI) provides several high-level commands. These are designed to simplify the processes of creating, updating, and monitoring clusters and tasks from a local development environment. (An Amazon ECS cluster is a logical grouping of tasks or services.)

The following `configure` command configures the Amazon ECS CLI to create a cluster configuration named `ecs-cli-demo`. This cluster configuration uses `FARGATE` as the default launch type for the `ecs-cli-demo` cluster in the `us-east-1 region`.

```
ecs-cli configure --region us-east-1 --cluster ecs-cli-demo --default-launch-type FARGATE
 --config-name ecs-cli-demo
```

For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.

AWS SAM CLI

AWS SAM CLI is a command line tool that operates on an AWS Serverless Application Model template and application code. You can perform several tasks using it. These include invoking Lambda functions locally, creating a deployment package for your serverless application, and deploying your serverless application to the AWS Cloud.

The following `init` command initializes a new SAM project with required parameters passed as parameters:

```
sam init --runtime python3.7 --dependency-manager pip --app-template hello-world --name
 sam-app
```

For more information, see the AWS SAM CLI command reference in the *AWS Serverless Application Model Developer Guide*.

Customizing your AWS CloudShell experience

You can customize the following aspects of your AWS CloudShell experience:

- Tabs layout (p. 43): Split the command line interface into multiple columns and rows.
- Font size (p. 43): Adjust the size of the command line text.
- Color theme (p. 43): Switch between a light and dark theme.
- Safe Paste (p. 43): Enable or disable a feature that requires you to verify multiline text before it's pasted.

You can also extend your shell environment by installing your own software (p. 63) and modifying start-up shell scripts (p. 64).

Splitting the command line display into multiple tabs

With this productivity feature, you can run multiple commands by splitting your command line interface into several panes.

Note
When multiple tabbed panes are open, you can select one to work in by clicking anywhere in the pane. You can close a tab by choosing the **x** symbol next to the Region name.

- Choose **Actions** and one of the following options from **Tabs layout**:
  - **New tab**: Add a new tab beside the currently active one.
  - **Split into rows**: Add a new tab in a row below the currently active one.
  - **Split into columns**: Add a new tab in a column beside the currently active one.

  If there's not enough space to display each tab fully, scroll bars allow you to navigate the interface. You can also select the split bars that separate panes and drag them using the pointer to increase or reduce the pane size.

Changing font size

You can increase or decrease the size of the text that's displayed in the command line interface.

1. Choose **Preferences**, **Font Size**.
2. Choose a text size between **Smallest** and **Largest**.

Changing the interface theme

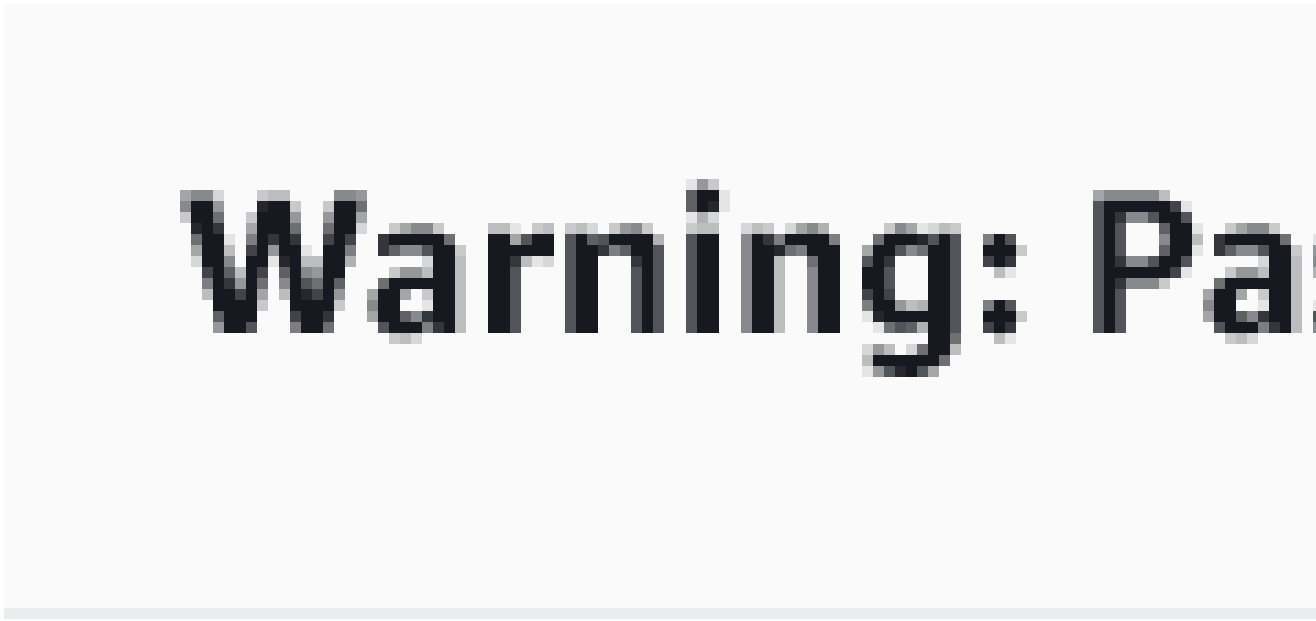
You can switch between a light and dark theme for the command line interface.

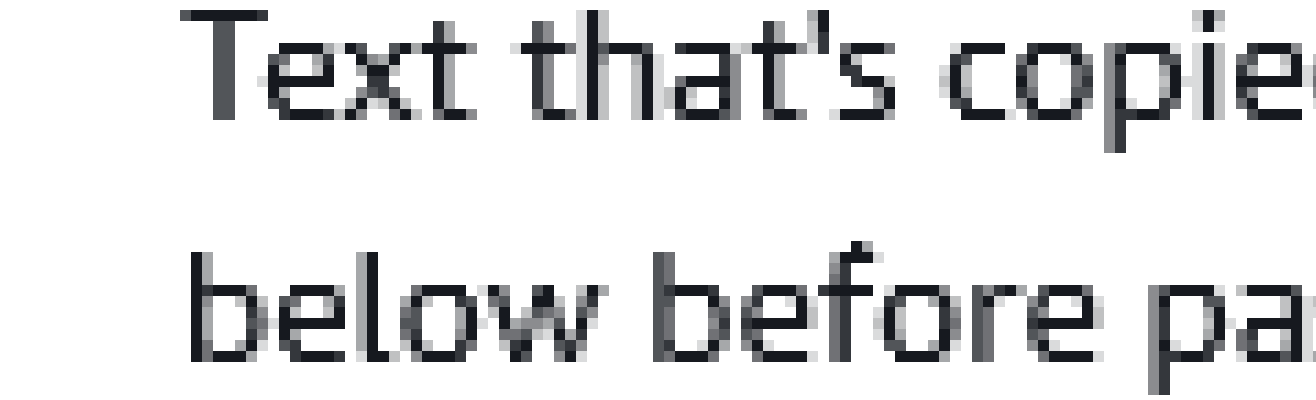
1. Choose **Preferences**, **Color Theme**.
2. Choose **Light** or **Dark**.

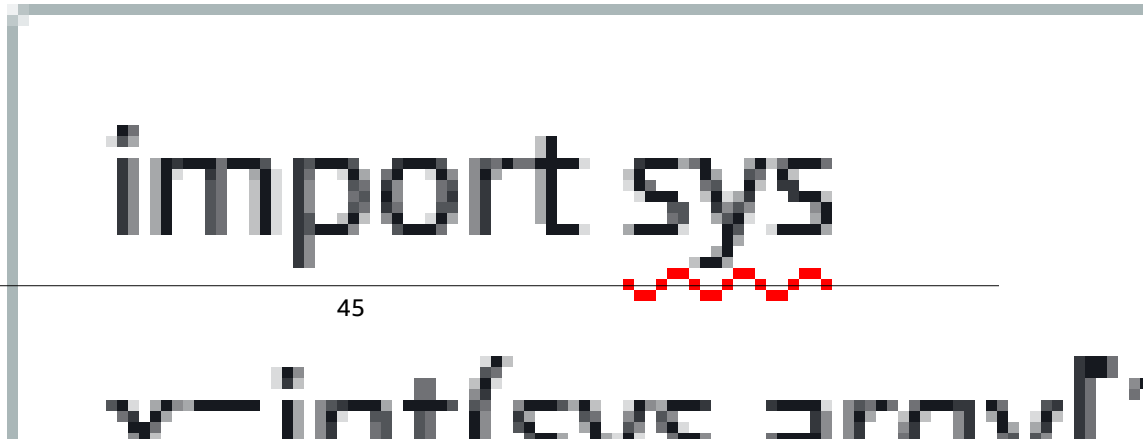
Using Safe Paste for multiline text

Safe Paste is a security feature that asks you to verify that multiline text that you're about to paste into the shell doesn't contain malicious scripts. Text copied from third-party sites can contain hidden code that triggers unexpected behaviors in your shell environment.

The Safe Paste dialog displays the complete text that you've copied to your clipboard. If you're satisfied that there's no security risk, you can choose **Paste**.


Warning: Pa


Text that's copie
below before pa


import sys

We recommend that you enable Safe Paste to catch potential security risks in scripts. But you can switch this feature on or off by choosing **Preferences**, **Enable Safe Paste**/**Disable Safe Paste**.

Security for AWS CloudShell

Cloud security at Amazon Web Services (AWS) is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. Security is a shared responsibility between AWS and you. The Shared Responsibility Model describes this as Security of the Cloud and Security in the Cloud.

Security of the Cloud – AWS is responsible for protecting the infrastructure that runs all of the services offered in the AWS Cloud and providing you with services that you can use securely. Our security responsibility is the highest priority at AWS, and the effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS Compliance Programs.

Security in the Cloud – Your responsibility is determined by the AWS service you are using, and other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

AWS CloudShell follows the shared responsibility model through the specific AWS services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

The following topics show you how to configure AWS CloudShell to meet your security and compliance objectives.

Topics

Data protection in AWS CloudShell

The AWS shared responsibility model applies to data protection in AWS CloudShell. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.

- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with AWS CloudShell or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into AWS CloudShell or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

Data encryption

Data encryption refers to protecting data when at rest (while stored in AWS CloudShell) and when in transit (as it travels between AWS CloudShell and service endpoints).

Encryption at rest using AWS KMS

Encryption at rest refers to protecting your data from unauthorized access by encrypting data while stored. When using AWS CloudShell, you have persistent storage of 1 GB per AWS Region at no cost. Persistent storage is located in your home directory (`$HOME`) and is private to you. Unlike ephemeral environment resources that are recycled after each shell session ends, data in your home directory persists.

The encryption of data stored in AWS CloudShell is implemented using cryptographic keys provided by AWS Key Management Service (AWS KMS). This is a managed AWS service for creating and controlling customer master keys (CMKs)—the encryption keys used to encrypt customer data that's stored in the AWS CloudShell environment. AWS CloudShell generates and manages cryptographic keys for encrypting data on behalf of customers.

Encryption in transit

Encryption in transit refers to protecting your data from being intercepted while it moves between communication endpoints.

By default, all data communication between the client's web browser computer and the cloud-based AWS CloudShell is encrypted by sending everything through an HTTPS/TLS connection.

You don't need to do anything to enable the use of HTTPS/TLS for communication.

Identity and access management in AWS CloudShell

AWS Identity and Access Management (IAM) is an Amazon Web Services (AWS) service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use resources in AWS services. IAM is an AWS service that you can use with no additional charge.

To use AWS CloudShell to access AWS, you need an AWS account and AWS credentials. To increase the security of your AWS account, consider using an *IAM user* to provide access credentials instead of your own AWS account credentials.

For information about working with IAM, see AWS Identity and Access Management.

For an overview of IAM users and why they are important for the security of your account, see AWS Security Credentials in the Amazon Web Services General Reference.

AWS CloudShell follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in AWS CloudShell.

Service user - If you use the AWS CloudShell service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS CloudShell features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS CloudShell, see Troubleshooting AWS CloudShell (p. 68).

Service administrator - If you're in charge of AWS CloudShell resources at your company, you probably have full access to AWS CloudShell. It's your job to determine which AWS CloudShell features and resources your employees should access. Then, submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM.

IAM administrator - If you're an IAM administrator, you might want to learn details about how you can write policies to manage users' access to services. For details specific to AWS CloudShell, see Managing AWS CloudShell access and usage with IAM policies (p. 50).

Authenticating with identities

You can access AWS as any of the following types of identities.

AWS account root user

When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your root credentials, and they provide complete access to all of your AWS resources.

> **Important**
> As a security best practice, you should use the root credentials only to create an IAM *administrator group* with an IAM *administrator user*. This is a group that gives the user full permissions to your AWS account. Then you can use this administrator user to create other IAM users and roles with limited permissions. For more information, see Create Individual IAM Users and Creating Your First IAM Admin User and Group in the *IAM User Guide*.

IAM user

An *IAM user* is simply an identity within your AWS account that has specific custom permissions (for example, permissions to download a file in AWS CloudShell). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS CloudShell console, AWS Management Console, AWS Discussion Forums, and AWS Support Center.

In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several AWS SDKs or by using the AWS Command Line Interface (AWS CLI) or the aws-shell. The AWS SDKs, the AWS CLI, and the aws-shell use these access keys to cryptographically sign your request. If you don't use these tools, you must sign the request yourself. AWS CloudShell supports Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *Amazon Web Services General Reference*.

IAM role

An *IAM role* is another IAM identity you can create in your account that has specific permissions. It's similar to an IAM user, but it isn't associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations.

AWS service access

You can use an IAM role in your account to grant AWS service permissions to access your account's resources. For example, you can create a role that allows AWS Lambda to access an Amazon S3 bucket on your behalf, and then load data that's stored in the bucket into an Amazon Redshift data warehouse. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

Applications running on Amazon EC2

Instead of storing access keys within an Amazon EC2 instance for use by applications running on the instance and making AWS API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an Amazon EC2 instance and make it available to all of its applications, you can create an *instance profile* that's attached to the instance. An instance profile contains the role and enables programs running on the Amazon EC2 instance to get temporary credentials. For more information, see Create and Use an Instance Profile to Manage Temporary Credentials and Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the *IAM User Guide*.

Federated user access

Instead of creating an IAM user, you can use pre-existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information, see Federated Users and Roles in the *IAM User Guide*.

Managing AWS CloudShell access and usage with IAM policies

With the access management resources provided by AWS Identity and Access Management (IAM), administrators can grant permissions to IAM users so they can access AWS CloudShell and use the environment's features. Administrators can also create policies that specify at a granular level what actions those users can perform with the shell environment.

The quickest way for an administrator to grant access to users is through an AWS managed policy. An AWS managed policy is a standalone policy that's created and administered by AWS. The following AWS managed policy for AWS CloudShell can be attached to IAM identities:

- **AWSCloudShellFullAccess**: Grants permission to use AWS CloudShell with full access to all features.

The **AWSCloudShellFullAccess** policy uses the wildcard (*) character to give the IAM identity (user, role, or group) full access to AWS CloudShell and features. You can also use the AWSCloudShellFullAccess policy as a template for custom policies that are more restrictive in terms of permitted user actions.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "CloudShellUser",
        "Effect": "Allow",
        "Action": [
            "cloudshell:*"
        ],
        "Resource": "*"
    }]
}
```

Note
IAM identities with the AWS managed policies listed below can also launch AWS CloudShell. But as these policies provide very extensive permissions, the principle of least privilege means they should only be granted if they're essential for an IAM user's job role.

- Administrator: Provides users with full access and allows them to delegate permissions to every service and resource in AWS.
- Developer power user: Enables users to perform application development tasks and create and configure resources and services that support AWS-aware application development.

For more information on attaching managed policies, see Adding IAM identity permissions (console) in the *IAM User Guide*.

Managing allowable actions in AWS CloudShell using custom policies

To manage the scope of actions that an IAM user can perform with AWS CloudShell, you can create a custom policy that uses the AWSCloudShellFullAccess managed policy as a template. Alternatively, you can edit an inline policy that's embedded in the relevant IAM identity (user, group, or role).

For example, you can allow users to access AWS CloudShell but prevent them from uploading or downloading files within the shell environment. You can also explicitly deny users access to AWS CloudShell.

Important
To launch AWS CloudShell from the AWS Management Console, an IAM user needs permissions for the following actions:

- `CreateEnvironment`
- `CreateSession`
- `GetEnvironmentStatus`
- `StartEnvironment`

If any of these actions aren't explicitly allowed by an attached policy, an IAM permissions error is returned when you try to launch CloudShell.

AWS CloudShell permissions

Name	Description of permission granted	Required to launch CloudShell?
`cloudshell:CreateEnvironment`	Create a CloudShell environment	Yes
`cloudshell:CreateSession`	Connect to a CloudShell environment from the AWS Management Console	Yes
`cloudshell:GetEnvironmentStatus`	Read the status of a CloudShell environment	Yes
`cloudshell:DeleteEnvironment`	Delete a CloudShell environment	No
`cloudshell:GetFileDownloadUrls`	Download files from CloudShell to a local machine	No
`cloudshell:GetFileUploadUrls`	Upload files from a local machine to CloudShell	No
`cloudshell:PutCredentials`	Forward the credentials used to log in to the AWS Management Console to CloudShell	No
`cloudshell:StartEnvironment`	Start a CloudShell environment that's stopped	Yes
`cloudshell:StopEnvironment`	Stop a CloudShell environment that's running	No

Examples of IAM policies for CloudShell

The following examples show how policies can be created to restrict who can access AWS CloudShell and the actions that can be performed in the shell environment.

This policy enforces a complete denial of access to AWS CloudShell and its features:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "DenyCloudShell",
        "Effect": "Deny",
        "Action": [
            "cloudshell:*"
        ],
        "Resource": "*"
    }]
}
```

This policy allows users to access AWS CloudShell but blocks them from uploading and downloading files in the shell environment:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "CloudShellUser",
        "Effect": "Allow",
        "Action": [
```

```
            "cloudshell:*"
        ],
        "Resource": "*"
    }, {
        "Sid": "DenyUploadDownload",
        "Effect": "Deny",
        "Action": [
            "cloudshell:GetFileDownloadUrls",
            "cloudshell:GetFileUploadUrls"
        ],
        "Resource": "*"
    }]
}
```

The following policy allows users to access AWS CloudShell but prevents the credentials you used to log in to AWS Management Console from being forwarded to the CloudShell environment. Users with this policy need to manually configure their credentials within AWS CloudShell.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "CloudShellUser",
        "Effect": "Allow",
        "Action": [
            "cloudshell:*"
        ],
        "Resource": "*"
    }, {
        "Sid": "DenyCredentialForwarding",
        "Effect": "Deny",
        "Action": [
            "cloudshell:PutCredentials"
        ],
        "Resource": "*"
    }]
}
```

Permissions for accessing AWS services

AWS CloudShell uses the IAM credentials you used to sign in to the AWS Management Console. This pre-authentication feature of AWS CloudShell makes it very convenient to use AWS CLI. But an IAM user still requires explicit permissions for the AWS services that are called from the command line.

For example, if IAM users are required to create Amazon S3 buckets and upload files as objects to them, you can create a policy that explicitly allows those actions. The IAM console provides an intuitive visual editor that guides you through the process of building up a JSON-formatted policy document. After the policy is created, you can attach it to relevant IAM identity (user, group, or role).

For more information on attaching managed policies, see Adding IAM identity permissions (console) in the *IAM User Guide*.

Logging and monitoring in AWS CloudShell

Monitoring activity with CloudTrail

AWS CloudShell is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS CloudShell. CloudTrail captures all API calls for AWS CloudShell as events. The calls captured include calls from the AWS CloudShell console and from code calls to the AWS CloudShell APIs.

If you create a trail, you can enable the continuous delivery of CloudTrail events to an Amazon Simple Storage Service (Amazon S3) bucket, including events for AWS CloudShell.

If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS CloudShell, the IP address from which the request was made, who made the request, when it was made, and additional details.

AWS CloudShell in CloudTrail

AWS CloudShell supports logging the following actions as events in CloudTrail log files:

- `createEnvironment`
- `createSession`
- `deleteEnvironment`
- `getEnvironmentStatus*`
- `getFileDownloadUrls*`
- `getFileUploadUrls*`
- `putCredentials`
- `redeemCode`
- `sendHeartBeat`
- `startEnvironment`
- `stopEnvironment`

*Non-mutating (read-only) API calls.

Events that include the word "Environment" in their names relate to the lifecycle of the compute environment that hosts the shell experience.

The `sendHeartBeat` event occurs to confirm that the session is not inactive. And the `putCredentials` event occurs when the credentials the user signed in to console with are forwarded to AWS CloudShell.

EventBridge rules for AWS CloudShell actions

With EventBridge rules you specify a target action to take when EventBridge receives an event that matches the rule. You can define a rule that specifies a target action to take based on an AWS CloudShell action that's recorded as an event in a CloudTrail log file.

For example, you can create EventBridge rules with AWS CLI using the `put-rule` command. A `put-rule` call must contain at least an EventPattern or ScheduleExpression. Rules with EventPatterns are triggered when a matching event is observed. The EventPattern for AWS CloudShell events:

```
{ "source": [ "aws.cloudshell" ], "detail-type": [ "AWS API Call via CloudTrail" ],
 "detail": { "eventSource": [ "cloudshell.amazonaws.com" ] } }
```

For more information, see Events and Event Patterns in EventBridge in the *Amazon EventBridge User Guide*.

Compliance validation for AWS CloudShell

Third-party auditors assess the security and compliance of AWS services as part of multiple AWS compliance programs.

At present, AWS CloudShell is not in scope of any specific compliance programs.

For a list of AWS services that are in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports by using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS CloudShell is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security-focused and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in AWS CloudShell

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, AWS CloudShell supports specific features to support your data resiliency and backup needs.

- Commit files you create and add to AWS CodeCommit. This is a version control service hosted by Amazon Web Services that you can use to privately store and manage assets in the cloud. These assets can consist of documents, source code, and binary files. For more information, see Tutorial: Using CodeCommit in AWS CloudShell (p. 24).
- Use AWS CLI calls to specify files in your home directory in AWS CloudShell and add them as objects in Amazon S3 buckets. For an example, see the getting started tutorial (p. 9).

Infrastructure security in AWS CloudShell

As a managed service, AWS CloudShell is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access AWS CloudShell through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. We recommend TLS 1.2 or later. Clients must also support

cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that's associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Note
By default, AWS CloudShell automatically install security patches for the system packages of your compute environments.

Configuration and vulnerability analysis in AWS CloudShell

It's the responsibility of the AWS CloudShell user to ensure that any third-party software installed in the compute environment is patched and up to date.

Security best practices for AWS CloudShell

The following best practices are general guidelines and don’t represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations instead of prescriptions.

Some security best practices for AWS CloudShell

- Use IAM permissions and policies to control access to AWS CloudShell and ensure users can perform only those actions (downloading and uploading files, for example) required by their role. For more information, see Managing AWS CloudShell access and usage with IAM policies (p. 50).
- Keep Safe Paste feature enabled to catch potential security risks in text you've copied from external sources. Safe Paste is enabled by default. For more information, see Using Safe Paste for multiline text (p. 44).
- Be familiar with the Shared Security Responsibility Model if you installed third-party applications to the compute environment of AWS CloudShell.
- Prepare rollback mechanisms before editing shell scripts that affect the user's shell experience. For more information, see Modifying your shell with scripts (p. 64).
- Store your code securely in a version control system, for example, AWS CodeCommit.

AWS CloudShell compute environment: specifications and software

When you launch AWS CloudShell, a compute environment that's based on Amazon Linux 2 is created to host the shell experience. The environment is configured with compute resources (vCPU and memory) (p. 57) and provides a wide range of pre-installed software (p. 57) that can be accessed from the command line interface. You can also configure your default environment by installing software and modifying shell scripts.

Compute environment resources

Each AWS CloudShell compute environment is assigned the following CPU and memory resources:

- 1 vCPU (virtual central processing unit)
- 2 GiB RAM

In addition, the environment is provisioned with the following storage configuration:

- 1 GB persistent storage (storage persists after the session ends)

For more information, see Persistent storage (p. 72).

Important
Currently, the AWS CloudShell compute environment doesn't support Docker containers.

Pre-installed software

Note
Because the AWS CloudShell development environment is regularly updated to provide access to the latest software, we don't provide specific version numbers in this documentation. Instead, we describe how you can check which version is installed. Often this can be done by entering the program name followed by the `--version` option (for example, `git --version`).

Shells

Pre-installed shells

Name	Description	Version information
Bash	The Bash shell is the default shell application for AWS CloudShell.	`bash --version`
PowerShell	Offering a command line interface and scripting language support, PowerShell is built on top of Microsoft's .NET	`(Get-Host).Version`

Name	Description	Version information
	Command Language Runtime. PowerShell uses lightweight commands called `cmdlets` that accept and return .NET objects.	
Z Shell (zsh)	The Z Shell, also known as `zsh`, is an extended version of the Bourne Shell that offers enhanced customization support for themes and plugins.	`zsh --version`

AWS command line interfaces (CLI)

CLI

Name	Description	Version information
AWS CLI	The AWS CLI is a command line interface that you can use to manage multiple AWS services from the command line and automate them using scripts. For more information, see Working with AWS services in AWS CloudShell (p. 40). For information about how you can ensure that you're using the most up-to-date version AWS CLI version 2, see Installing AWS CLI to your home directory (p. 63).	`aws --version`
EB CLI	The AWS Elastic Beanstalk CLI provides a command line interface to simplify creating, updating, and monitoring environments from a local repository. For more information, see Using the Elastic Beanstalk command line interface (EB CLI) in the *AWS Elastic Beanstalk Developer Guide*.	`eb --version`
Amazon ECS CLI	Amazon Elastic Container Service (Amazon ECS) command line interface (CLI) provides high-level commands to simplify creating, updating, and monitoring clusters and tasks. For more information, see Using the Amazon ECS Command Line	`ecs-cli --version`

Name	Description	Version information
	Interface in the *Amazon Elastic Container Service Developer Guide*.	
AWS SAM CLI	AWS SAM CLI is a command line tool that operates on an AWS Serverless Application Model template and application code. You can perform several tasks. These include invoking Lambda functions locally, creating a deployment package for your serverless application, and deploying your serverless application to the AWS Cloud. For more information, see the AWS SAM CLI command reference in the *AWS Serverless Application Model Developer Guide*.	`sam --version`

Runtimes and AWS SDKs: Node.js and Python 3

Runtimes and AWS SDKs

Name	Description	Version information
Node.js (with npm)	Node.js is a JavaScript runtime that's designed to make it easier to apply asynchronous programming techniques. For more information, see the documentation on the official Node.js site. npm is a package manager that provides access to an online registry of JavaScript modules. For more information, see the documentation on the official npm site.	• Node.js: `node --version` • npm: `npm --version`
SDK for JavaScript in Node.js	This software development kit (SDK) helps simplify coding by providing JavaScript objects for AWS services including Amazon S3, Amazon EC2, DynamoDB, and Amazon SWF. For more information, see the AWS SDK for JavaScript Developer Guide.	`npm -g ls --depth 0 2>/dev/null | grep aws-sdk`
Python	Both Python 3 and Python 2 are both ready to use in the shell environment. Python 3	• Python 2: `python --version`

Name	Description	Version information
	is now considered the default version of the programming language (support for Python 2 ended in January 2020). For more information, see the documentation on the official Python site. Also, pre-installed is pip, the package installer for Python. You can use this command line program to install Python packages from the online indexes such as the Python Package Index. For more information, see the documentation provided by the Python Packaging Authority.	• Python 3: `python3 --version` • pip: `pip3 --version`
SDK for Python (Boto3)	Boto is the software development kit (SDK) that Python developers use to create, configure, and manage AWS services, such as EC2 and S3. The SDK provides an easy-to-use, object-oriented API, as well as low-level access to AWS services. For more information, see the Boto3 documentation.	`pip3 list | grep boto3`

Development tools and shell utilities

Development tools and shell utilities

Name	Description	Version information
CodeCommit utility for Git	git-remote-codecommit is a utility that provides a simple method for pushing and pulling code from CodeCommit repositories by extending Git. It's the recommended method for supporting connections that are made with federated access, identity providers, and temporary credentials. For more information, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit in the *AWS CodeCommit User Guide*.	`pip3 list | grep git-remote-codecommit`

Name	Description	Version information
Git	Git is a distributed version control system that supports modern software development practices through branch workflows and content staging. For more information, see the documentation page on Git's official site.	`git --version`
iputils	The iputils package contains utilities for Linux networking. For more information about the utilities provided, see the iputils repository on GitHub.	Examples for an iputils tool: `arping -V`
jq	The jq utility parses JSON-formatted data to produce output that's modified by command line filters. For more information, see the jq manual hosted on GitHub.	`jq --version`
make	The make utility uses `makefiles` to automate sets of tasks and organize code compilation. For more information, see the GNU Make documentation.	`make --version`
man	The man command provides manual pages for command line utilities and tools. For example, `man ls` returns the manual page for the `ls` command that lists the contents of directories. For more information, see the Wikipedia entry on man page.	`make --version`
procps	procps is a system administration utility that you can use to monitor and halt currently running processes. For more information, see the README file that lists programs that can be run with procps.	`ps --version`
SSH client	SSH clients use the secure shell protocol for encrypted communications with a remote computer. OpenSSH is the SSH client that's pre-installed. For more information, see the OpenSSH site maintained by the OpenBSD.	`ssh -V`

Name	Description	Version information
sudo	With the sudo utility, users can run a program with the security privileges of another user, typically the superuser. Sudo is useful when you need to install applications as a system administration. For more information, see the Sudo Manual.	`sudo --version`
tar	tar is a command line utility that you can use to group multiple files in a single archive file (often called a tarball). For more information, see the GNU tar documentation.	`tar --version`.
tmux	tmux is a terminal multiplexer that you can use to run different programs simultaneous in multiple windows. When running, tmux displays information about the current session in a status bar at the bottom of the window. For more information, see a blog that provides a concise introduction to tmux.	`tmux -V`
unzip	See zip/unzip	
vim	vim is a customizable editor that users interact with through a text-based interface. For more information, see the documentation resources provided on vim.org.	`vim --version`
wget	wget is a computer program used to retrieve content from web servers specified by endpoints in the command line. For more information, see the GNU Wget documentation.	`wget --version`

zip/unzip	The zip/unzip utilities use an archive file format that delivers lossless data compression without data loss. Call the zip command to group and compress files in a single archive. Use unzip to extract files from an archive into a specified directory.	`unzip --version`
		
		`zip --version`

Installing AWS CLI to your home directory

Like the rest of the software that's pre-installed in your CloudShell environment, the AWS CLI tool is updated automatically with scheduled upgrades and security patches. If you want to ensure that you have the most up-to-date version of AWS CLI, you can choose to manually install the tool in the shell's home directory.

Important
You need to manually install your copy of AWS CLI in the home directory so that it's available the next time you start a CloudShell session. This is because files that are added to directories outside of `$HOME` are deleted after you finish a shell session. Note also that, after you install this copy of AWS CLI, it isn't automatically updated. In other words, it's your responsibility to manage updates and security patches.
For more information about the AWS Shared Responsibility Model, see Data protection in AWS CloudShell (p. 47).

To install AWS CLI

1. First, in the CloudShell command line, use the `curl` command to transfer a zipped copy of the AWS CLI installed to the shell:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
```

2. Unzip the zipped folder:

```
unzip awscliv2.zip
```

3. Now run the AWS CLI installer to add the tool to a specified folder:

```
sudo ./aws/install --install-dir /home/cloudshell-user/usr/local/aws-cli --bin-dir /
home/cloudshell-user/usr/local/bin
```

   If it's installed successfully, the command line displays the following message:

```
You can now run: /home/cloudshell-user/usr/local/bin/aws --version
```

4. For your own convenience, we recommend that you also update the `PATH` environmental variable so that you don't need to specify the path to your installation of the tool when running `aws` commands:

```
export PATH=/home/cloudshell-user/usr/local/bin:$PATH
```

   Note
   If you undo this change to `PATH`, `aws` commands that don't feature a specified path use the pre-installed version of AWS CLI by default.

Installing third-party software on your shell environment

Note
We recommend that you review the Shared Security Responsibility Model before you install any third-party applications to the AWS CloudShell's compute environment.

By default, all AWS CloudShell users have sudo privileges. Therefore, you can use the `sudo` command to install software that's not already available in the shell's compute environment. For example, you can use `sudo` with the YUM package-management utility to install the GNU nano text editor:

```
sudo yum install nano
```

You can then launch the newly installed program by typing `nano`.

Important
Package manage utilities such as yum install programs in directories (`/user/bin`, for example), which are recycled when your shell session ends. This means additional software is installed and used on a per-session basis.

Modifying your shell with scripts

If you want to modify the default shell environment, you can edit a shell script that runs every time the shell environment starts up. The `.bashrc` script runs whenever the default bash shell starts up.

Warning
If you incorrectly modify your `.bashrc` file, you might not be able to access your shell environment afterward. It's good practice to make a copy of the file before editing. You can also mitigate risk by opening two shells when editing `.bashrc`. If you lose access in one shell, you're still logged into the other shell and can roll back any changes.
If you do lose access after incorrectly modifying `.bashrc` or any other file, you can return AWS CloudShell to its default settings by deleting your home directory (p. 64).

In the procedure, you'll modify the `.bashrc` script so that your shell environment switches automatically to running the Z shell.

1. Open the `.bashrc` using a text editor (Vim, for example):

   ```
   vim .bashrc
   ```

2. In the editor interface, press the **I** key to start editing and add the following:

   ```
   zsh
   ```

3. To exit and save the edited `.bashrc` file, press **Esc** to enter the Vim command mode and enter the following:

   ```
   :wq
   ```

4. Use the `source` command to reload the `.bashrc` file:

   ```
   source .bashrc
   ```

   When the command line interface becomes available again, the prompt symbol has changed to `%` to indicate that you're now using the Z shell.

Deleting your home directory

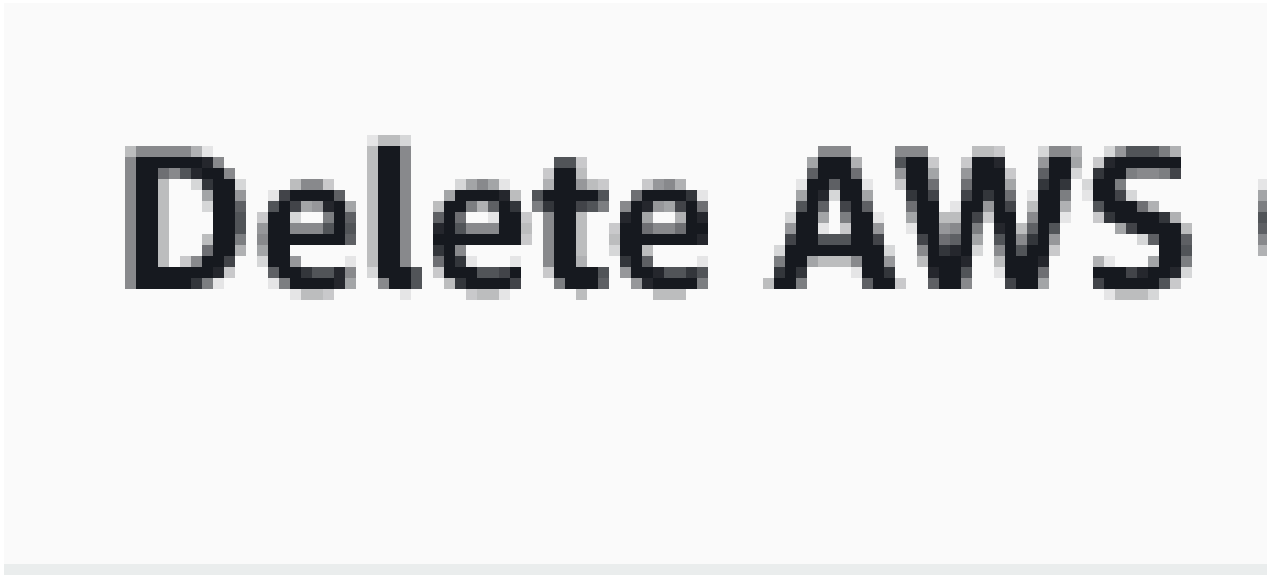
Warning
Deleting your home directory is an irreversible action in which all the data that's stored in your home directory is deleted permanently. However, you might want to consider this option in the following situations:

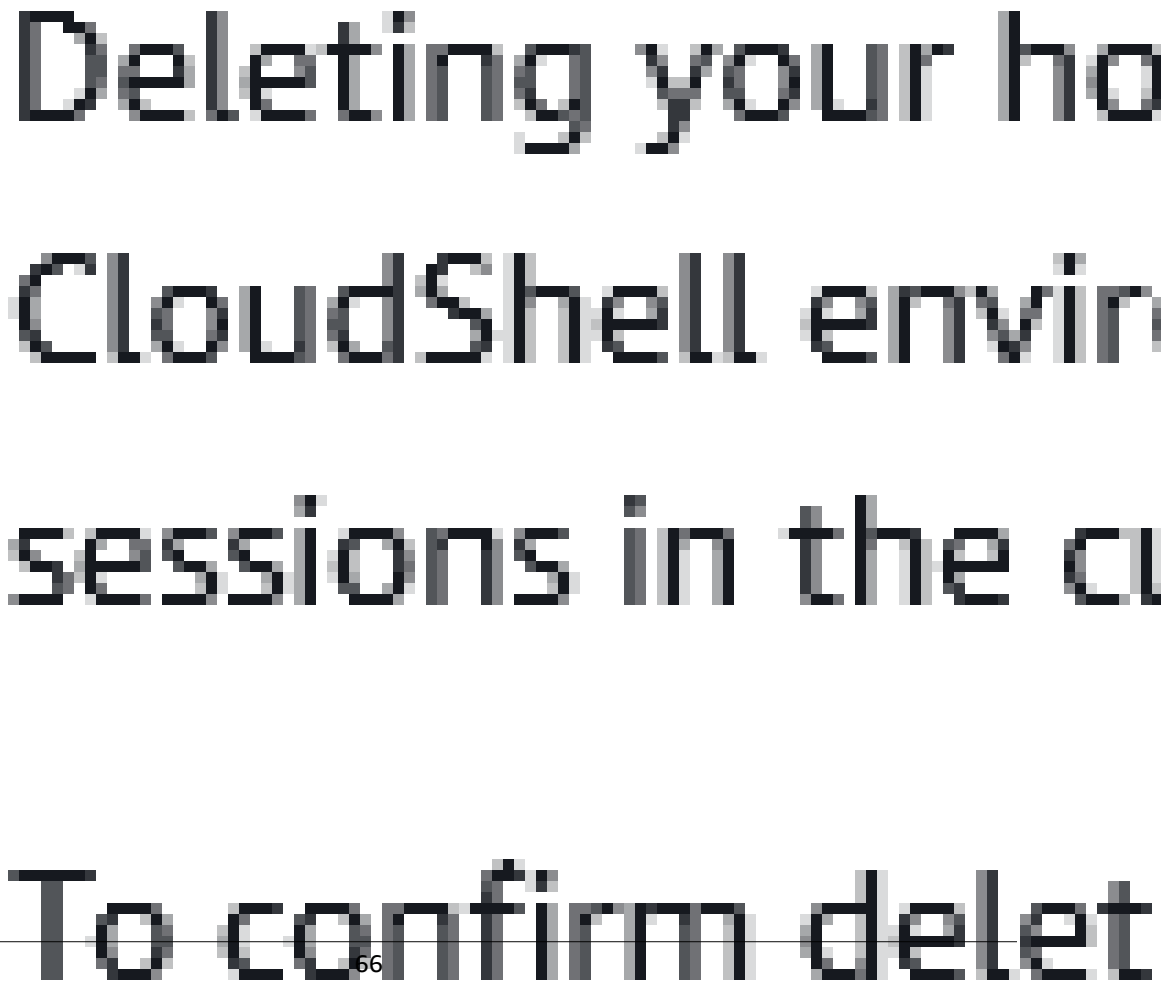
- You've incorrectly modified a file and can't access the AWS CloudShell compute environment. Deleting your home directory returns AWS CloudShell to its default settings.
- You want to remove all your data from AWS CloudShell immediately. Note that, if you stop using AWS CloudShell in an AWS Region, persistent storage is automatically deleted at the end of the retention period (p. 72) unless you launch AWS CloudShell again in the Region.

If you require long-term storage for your files, please consider a service such as Amazon S3 or CodeCommit.

To delete your home directory and reset AWS CloudShell

1. Choose **Actions**, **Delete AWS CloudShell home directory**.
2. In the dialog box, enter the word "delete" to activate the **Delete** option.


Delete AWS


Deleting your ho
CloudShell envir
sessions in the c
To confirm delet

3. Choose **Delete**.

   A new AWS CloudShell compute environment with default settings is created and started. You can confirm the deletion by running the `ls` command in the home directory.

Troubleshooting AWS CloudShell

Use the following information to help you identify and address issues with AWS CloudShell.

If your issue isn't listed, or if you need additional help, see the AWS CloudShell Discussion Forum. (When you enter this forum, AWS might require you to sign in.) You can also contact us directly.

Topics

Unable to launch AWS CloudShell with message "Unable to start the environment. You don't have access permissions. Ask your IAM administrator for access to AWS CloudShell."

Issue: When you try to launch AWS CloudShell from the AWS Management Console, you're denied access and notified you don't have permissions.

Cause: The IAM identity that you're using to access AWS CloudShell lacks the necessary IAM permissions.

Solution: Request that your IAM administrator provides you with the necessary permissions, either through an attached AWS managed policy (AWSCloudShellFullAccess) or an embedded inline policy. For more information, see Managing AWS CloudShell access and usage with IAM policies (p.).

(back to top (p. 68))

Unable to access AWS CloudShell command line.

Issue: After modifying a file that the compute environment uses, you're unable to access the command line in AWS CloudShell.

Solution: If you do lose access after incorrectly modifying `.bashrc` or any other file, you can return AWS CloudShell to its default settings by deleting your home directory (p. 64).

(back to top (p. 68))

Unable to ping external IP addresses.

Issue: When you run a ping command from the command line (`ping amazon.com`, for example), you receive the following message:

```
ping: socket: Operation not permitted
```

Cause: The ping utility uses Internet Control Message Protocol (ICMP) to send echo requests packets to a target host. It then waits for an echo reply from the target. Because the ICMP protocol isn't enabled in AWS CloudShell, the ping utility doesn't operate in the shell's compute environment.

(back to top (p. 68))

Supported browsers for AWS CloudShell

The following table lists the supported browsers for AWS CloudShell.

Web browser support

Browser	Version
Google Chrome	Latest three major versions
Mozilla Firefox	Latest three major versions
Microsoft Edge	Latest three major versions
Apple Safari for macOS	Latest two major versions

Supported AWS Regions for AWS CloudShell

You can currently work with AWS CloudShell in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (Oregon)
- Asia Pacific (Tokyo)
- Europe (Ireland)

Limits and restrictions for AWS CloudShell

AWS CloudShell is a browser-based shell provided at no additional charge. Limits and restrictions apply to the following areas:

Persistent storage

With AWS CloudShell, you have persistent storage of 1 GB for each AWS Region at no cost. Persistent storage is located in your home directory ($HOME) and is private to you. Unlike ephemeral environment resources that are recycled after each shell session ends, data in your home directory persists between sessions.

If you stop using AWS CloudShell in an AWS Region, data is retained in the persistent storage of that Region for **120 days** after the end of your last session. After 120 days unless you take action, your data will be automatically deleted from the persistent storage of that Region. You can prevent removal by simply launching AWS CloudShell again in that AWS Region.

> **Note**
> **Usage scenario**
> Márcia has used AWS CloudShell to store files in her home directories in two AWS Regions: US East (N. Virginia) and Europe (Ireland). She then started using AWS CloudShell exclusively in Europe (Ireland) and stopped launching shell sessions in US East (N. Virginia).
> Before the deadline for deletion of data in US East (N. Virginia), Márcia decides to prevent her home directory from being recycled by launching AWS CloudShell and selecting the US East (N. Virginia) Region again. Because she has continually used Europe (Ireland) for shell sessions, her persistent storage in that Region isn't affected.

Monthly usage

There are monthly usage limits for AWS CloudShell per AWS Region in your AWS account. If you try to access AWS CloudShell after you've reached the monthly limit for that Region, a message displays to explain why the shell environment can't be started.

> **Note**
> If you need to increase your monthly usage limits, contact AWS Support.

Concurrent shells

- **Concurrent shells**: You can run a maximum of 10 shells at the same time in each AWS Region at no charge.

Shell sessions

- **Inactive sessions**: AWS CloudShell is an interactive shell environment—if you don't interact with it using your keyboard or pointer for approximately **20–30 minutes**, your shell session will end. (Running processes do not count as interactions.)
- **Long-running sessions**: A shell session that's been running continuously for approximately 12 hours will automatically end, even if the user is regularly interacting with it during that period.

Network access and data transfer

The following restrictions apply to network traffic traveling in and out of the AWS CloudShell environment:

- **Outbound**: Users can access the public internet.
- **Inbound**: Users can't access inbound ports. No public IP address is available.

> **Warning**
> With access to the public internet, there's a risk that certain users might export data from the AWS CloudShell environment. IAM administrators should manage the allow list of trusted AWS CloudShell users through IAM tools. For information on how specific users can be explicitly denied access, see Managing allowable actions in AWS CloudShell using custom policies (p. 51).

Data transfer: Uploading and downloading files to and from AWS CloudShell may be slow for large files. Alternatively, you can transfer files to your environment from an Amazon S3 bucket using the command line interface of the shell.

Restrictions on system files and page reloads

- **System files**: If you incorrectly modify files that are required by the compute environment, you might experience problems when accessing or using the AWS CloudShell environment. If this occurs, you may need to delete your home directory (p. 64) to regain access.
- **Reloading pages**: To reload the AWS CloudShell interface, you should use the refresh button in your browser instead of the default shortcut key sequence for your operating system.

Document history for the AWS CloudShell User Guide

Recent updates

The following table describes important changes to the *AWS CloudShell User Guide*.

update-history-change	update-history-description	update-history-date
General availability (GA) release of AWS CloudShell in selected AWS Regions (p. 74)	AWS CloudShell is now generally available in the following AWS Regions: • US East (Ohio) • US East (N. Virginia) • US West (Oregon) • Asia Pacific (Tokyo) • Europe (Ireland)	December 15, 2020