**Code Explanation for infra-cost CLI**

**src/printers/favcy.ts:**

The file exports a single function called printFancy that generates a formatted AWS cost report in the console. Here's a detailed breakdown:

### **File Overview**

* **Imports**:
  + chalk: A library for styling terminal output with colors and formatting.
  + TotalCosts: A type or interface imported from ../cost, representing cost data structure.
  + hideSpinner: A function imported from ../logger to hide a loading spinner in the console.
* **Purpose**: The file defines a function to display AWS cost information in a visually appealing, tabular format, with options for summary or detailed output.

### **Function: printFancy**

* **Purpose**: Prints a formatted AWS cost report to the console, showing total costs across different time periods and optionally a breakdown by service.
* **Parameters**:
  1. accountAlias: string: The AWS account name or alias to display in the report header.
  2. totals: TotalCosts: An object containing cost data, split into total costs and costs by service.
  3. isSummary: boolean = false: Optional flag; if true, only summary totals are shown (no service breakdown).
* **Variables and Logic**:
  1. **Initial Setup**:
     + hideSpinner(): Hides any active spinner in the console (likely from a previous operation).
     + console.clear(): Clears the console for a clean display.
     + totalCosts = totals.totals: Extracts the overall cost totals (e.g., last month, this month, etc.).
     + serviceCosts = totals.totalsByService: Extracts costs broken down by AWS service.
  2. **Service Name Alignment**:
     + allServices = Object.keys(serviceCosts.yesterday): Gets a list of service names from yesterday's data.
     + sortedServiceNames = allServices.sort((a, b) => b.length - a.length): Sorts services by name length (longest first) for consistent column width.
     + maxServiceLength: Calculates the length of the longest service name plus 1 for padding, ensuring aligned output.
  3. **Formatted Totals**:
     + totalLastMonth = chalk.green($$ {totalCosts.lastMonth.toFixed(2)}): Formats last month's total cost in green with 2 decimal places.
     + totalThisMonth = chalk.green( $${totalCosts.thisMonth.toFixed(2)}): Formats this month's total cost in green.
     + totalLast7Days = chalk.green($$ {totalCosts.last7Days.toFixed(2)}): Formats the last 7 days' total cost in green.
     + totalYesterday = chalk.bold.yellowBright( $${totalCosts.yesterday.toFixed(2)}): Formats yesterday's total cost in bold yellow.
  4. **Summary Output**:
     + Prints a header with the accountAlias in bold yellow, followed by the formatted totals for each time period.
     + If isSummary is true, the function returns early, skipping the service breakdown.
  5. **Detailed Service Breakdown (if isSummary is false)**:
     + headerPadLength = 11: Sets a fixed width for cost columns.
     + **Headers**:
       - serviceHeader = chalk.white('Service'.padStart(maxServiceLength)): Service column header, left-aligned.
       - lastMonthHeader = chalk.white('Last Month'.padEnd(headerPadLength)): Last month cost header.
       - thisMonthHeader = chalk.white('This Month'.padEnd(headerPadLength)): This month cost header.
       - last7DaysHeader = chalk.white('Last 7 Days'.padEnd(headerPadLength)): Last 7 days cost header.
       - yesterdayHeader = chalk.bold.white('Yesterday'.padEnd(headerPadLength)): Yesterday cost header in bold.
       - These are logged as a single line to form the table header.
     + **Service Rows**:
       - Loops through sortedServiceNames to print a row for each service.
       - serviceLabel = chalk.cyan(service.padStart(maxServiceLength)): Service name in cyan, padded for alignment.
       - lastMonthTotal = chalk.green($$ {serviceCosts.lastMonth[service].toFixed(2)}.padEnd(headerPadLength)): Last month's cost for the service in green.
       - thisMonthTotal = chalk.green($${serviceCosts.thisMonth[service].toFixed(2)}.padEnd(headerPadLength)): This month's cost in green.
       - last7DaysTotal = chalk.green($$ {serviceCosts.last7Days[service].toFixed(2)}.padEnd(headerPadLength)): Last 7 days' cost in green.
       - yesterdayTotal = chalk.bold.yellowBright($${serviceCosts.yesterday[service].toFixed(2)}.padEnd(headerPadLength)): Yesterday's cost in bold yellow.
       - Each row is logged with the service name and its costs aligned under the headers.

### **Assumptions About TotalCosts**

* The TotalCosts type likely has the following structure (inferred from usage):
  + totals: An object with properties:
    - lastMonth: number
    - thisMonth: number
    - last7Days: number
    - yesterday: number
  + totalsByService: An object with properties:
    - lastMonth: { [service: string]: number }
    - thisMonth: { [service: string]: number }
    - last7Days: { [service: string]: number }
    - yesterday: { [service: string]: number }

### **Output Example**

For accountAlias = "MyAccount", isSummary = false, and sample totals data:

```  
AWS Cost Report: MyAccount

Last Month: $123.45

This Month: $67.89

Last 7 days: $45.67

Yesterday : $12.34

Service Last Month This Month Last 7 Days Yesterday

S3 $50.00 $30.00 $20.00 $5.00

EC2 $73.45 $37.89 $25.67 $7.34

```  
  
- Colors are applied via chalk (green for most costs, yellow for yesterday, cyan for service names).

**src/printers/json.ts:**

### **File Overview**

* **Imports**:
  + TotalCosts: A type or interface imported from ../cost, representing the structure of cost data.
  + hideSpinner: A function imported from ../logger to hide a loading spinner in the console.
* **Purpose**: The file defines a function to output AWS cost data as a formatted JSON string, with options for either a summary view (totals only) or a detailed view (including service breakdowns).

### **Function: printJson**

* **Purpose**: Prints AWS cost data to the console in JSON format, either as a summary of total costs or a full breakdown including service-specific costs.
* **Parameters**:
  1. accountAlias: string: The AWS account name or alias to include in the JSON output.
  2. totalCosts: TotalCosts: An object containing cost data, including overall totals and service-specific costs.
  3. isSummary: boolean = false: Optional flag; if true, only summary totals are output; if false, the full cost data is included.
* **Variables and Logic**:
  1. **Initial Setup**:
     + hideSpinner(): Hides any active spinner in the console, likely from a prior operation.
  2. **Summary Output (if isSummary is true)**:
     + Constructs an object with two properties:
       - account: string: The accountAlias passed as an argument.
       - totals: The totalCosts.totals object, containing overall cost data (e.g., last month, this month, etc.).
     + JSON.stringify(..., null, 2): Converts the object to a JSON string with 2-space indentation for readability.
     + The result is logged to the console, and the function returns early.
  3. **Detailed Output (if isSummary is false)**:
     + Constructs an object by spreading the totalCosts object and adding an account property:
       - account: string: The accountAlias.
       - All properties of totalCosts (e.g., totals, totalsByService).
     + JSON.stringify(..., null, 2): Converts the object to a formatted JSON string with 2-space indentation.
     + The result is logged to the console.

### **Assumptions About TotalCosts**

* The TotalCosts type likely has the following structure (inferred from usage):
  + totals: An object with properties like:
    - lastMonth: number
    - thisMonth: number
    - last7Days: number
    - yesterday: number
  + totalsByService: An object with service-specific costs, such as:
    - lastMonth: { [service: string]: number }
    - thisMonth: { [service: string]: number }
    - last7Days: { [service: string]: number }
    - yesterday: { [service: string]: number }

### **Output Examples**

1. **Summary Output (isSummary = true)**: For accountAlias = "MyAccount" and totalCosts.totals = { lastMonth: 123.45, thisMonth: 67.89, last7Days: 45.67, yesterday: 12.34 }:

```

{

"account": "MyAccount",

"totals": {

"lastMonth": 123.45,

"thisMonth": 67.89,

"last7Days": 45.67,

"yesterday": 12.34

}

}

```

2. **Detailed Output (isSummary = false)**: For accountAlias = "MyAccount" and totalCosts = { totals: {...}, totalsByService: { yesterday: { S3: 5.00, EC2: 7.34 }, ... } }:

```

{

"account": "MyAccount",

"totals": {

"lastMonth": 123.45,

"thisMonth": 67.89,

"last7Days": 45.67,

"yesterday": 12.34

},

"totalsByService": {

"yesterday": {

"S3": 5.00,

"EC2": 7.34

},

"lastMonth": {

"S3": 50.00,

"EC2": 73.45

},

...

}

}

```

**src/printers/slack.ts:**

### **File Overview**

* **Imports**:
  + fetch from node-fetch: A library for making HTTP requests (used to send messages to Slack).
  + TotalCosts from ../cost: A type or interface defining the structure of cost data.
* **Purpose**: The file provides functionality to send AWS cost reports to a Slack channel, with options for summary-only or detailed reports including service breakdowns.

### **Function: formatServiceBreakdown**

* **Purpose**: Formats a breakdown of costs by service into a Slack-compatible markdown string, focusing on yesterday's costs.
* **Parameters**:
  + costs: TotalCosts: The cost data object containing totals and service-specific costs.
* **Variables and Logic**:
  + serviceCosts = costs.totalsByService: Extracts the service-specific cost breakdown.
  + sortedServices:
    - Starts with Object.keys(serviceCosts.yesterday): Gets all service names from yesterday's costs.
    - .filter((service) => serviceCosts.yesterday[service] > 0): Filters out services with zero cost yesterday.
    - .sort((a, b) => serviceCosts.yesterday[b] - serviceCosts.yesterday[a]): Sorts services by cost in descending order (highest to lowest).
  + serviceCostsYesterday:
    - Maps each service to a formatted string: > ${service}: \$$ {serviceCosts.yesterday[service].toFixed(2)}`(e.g.,> S3: `$5.00``).
    - Uses Slack markdown: > for indentation, ``` for code formatting.
  + **Return**: Joins the service cost lines with newlines (\n) into a single string.
* **Example Output**:

```

> EC2: `$7.34`

> S3: `$5.00`

```

### **Function: notifySlack (Exported)**

* **Purpose**: Sends a formatted AWS cost report to a specified Slack channel via the Slack API, with options for summary or detailed output.
* **Parameters**:
  + accountAlias: string: The AWS account name or alias to include in the message.
  + costs: TotalCosts: The cost data object containing totals and service breakdowns.
  + isSummary: boolean: If true, sends only a summary; if false, includes a service breakdown.
  + slackToken: string: The Slack API token for authentication.
  + slackChannel: string: The Slack channel ID or name to send the message to.
* **Variables and Logic**:
  + **Setup**:
    - channel = slackChannel: Assigns the channel ID/name to a variable.
    - totals = costs.totals: Extracts overall cost totals.
    - serviceCosts = costs.totalsByService: Extracts service-specific costs.
  + **Redundant Code (Not Used)**:
    - serviceCostsYesterday = []: Initializes an array and populates it with service costs in the format ${service}: $${cost}.
    - This array is computed but not used in the final message (likely a leftover from earlier code). Instead, formatServiceBreakdown is used later.
  + **Message Construction**:
    - summary: A Slack markdown string with:
      * Account name in bold (\*Account: ${accountAlias}\*).
      * Totals for yesterday, this month, and last month, each formatted as code (e.g., \$${totals.yesterday.toFixed(2)}``).
    - breakdown: A string with a service breakdown header and the result of formatServiceBreakdown(costs).
    - message = ${summary}: Starts with the summary; if !isSummary, appends the breakdown.
  + **Slack API Request**:
    - Uses fetch to send a POST request to https://slack.com/api/chat.postMessage.
    - **Body**: A JSON string with:
      * channel: The target Slack channel.
      * blocks: An array with a single section block containing the message in Slack markdown (mrkdwn).
    - **Headers**:
      * 'Content-Type': 'application/json; charset=utf-8'.
      * Authorization: Bearer ${slackToken}: Authenticates the request.
    - **Response Handling**:
      * Parses the response as JSON, expecting { ok: boolean; error?: string }.
      * If !data.ok, logs an error with the message (e.g., Failed to send message to Slack: ${data.error}) and exits with code 1.
      * If successful, logs Successfully sent message to Slack.
* **Example Message (Slack Output)**:
  + **Summary Only**:

```

> \*Account: MyAccount\*

>

> \*Summary \*

> Total Yesterday: `$12.34`

> Total This Month: `$67.89`

> Total Last Month: `$123.45`

```

* **With Breakdown**:

```

> \*Account: MyAccount\*

>

> \*Summary \*

> Total Yesterday: `$12.34`

> Total This Month: `$67.89`

> Total Last Month: `$123.45`

>

> \*Breakdown by Service:\*

> EC2: `$7.34`

> S3: `$5.00`

```

### **Assumptions About TotalCosts**

* The TotalCosts type likely has:
  + totals: { lastMonth: number, thisMonth: number, last7Days: number, yesterday: number }.
  + totalsByService: { yesterday: { [service: string]: number }, lastMonth: { [service: string]: number }, ... }.

**src/printers/text.ts:**

### **File Overview**

* **Imports**:
  + TotalCosts from ../cost: A type or interface defining the structure of cost data.
  + hideSpinner from ../logger: A function to hide a loading spinner in the console.
* **Purpose**: The file provides functions to display AWS cost data in a plain text format in the console, with options for a summary view (totals only) or a detailed view (including service breakdowns).

### **Function: printPlainSummary**

* **Purpose**: Prints a simple summary of total AWS costs for an account to the console.
* **Parameters**:
  + accountAlias: string: The AWS account name or alias to display.
  + costs: TotalCosts: The cost data object containing total costs.
* **Variables and Logic**:
  + hideSpinner(): Hides any active spinner in the console.
  + console.clear(): Clears the console for a clean display.
  + Logs the following plain text output:
    - An empty line (console.log('')).
    - The account name: Account: ${accountAlias}.
    - Another empty line.
    - A "Totals:" header.
    - Four lines with total costs, each prefixed with two spaces for indentation:
      * Last Month: $${costs.totals.lastMonth.toFixed(2)}
      * This Month: $${costs.totals.thisMonth.toFixed(2)}
      * Last 7 Days: $${costs.totals.last7Days.toFixed(2)}
      * Yesterday: $${costs.totals.yesterday.toFixed(2)}
* **Example Output**:

```

Account: MyAccount

Totals:

Last Month: $123.45

This Month: $67.89

Last 7 Days: $45.67

Yesterday: $12.34

```

### **Function: printPlainText (Exported)**

* **Purpose**: Prints AWS cost data to the console in plain text, either as a summary or with a detailed service breakdown, depending on the isSummary flag.
* **Parameters**:
  1. accountAlias: string: The AWS account name or alias to display.
  2. totals: TotalCosts: The cost data object containing totals and service-specific costs.
  3. isSummary: boolean = false: Optional flag; if true, only the summary is printed; if false, includes a service breakdown.
* **Variables and Logic**:
  1. **Summary Output**:
     + Calls printPlainSummary(accountAlias, totals) to print the account and total costs.
     + If isSummary is true, the function returns early.
  2. **Service Breakdown (if isSummary is false)**:
     + serviceTotals = totals.totalsByService: Extracts the service-specific cost data.
     + allServices = Object.keys(serviceTotals.yesterday).sort((a, b) => b.length - a.length): Gets a list of service names from yesterday’s costs and sorts them by name length (longest first) for consistent alignment.
     + Logs the following:
       - An empty line (console.log('')).
       - A "Totals by Service:" header.
  3. **Service Costs by Time Period**:
     + For each time period (Last Month, This Month, Last 7 Days, Yesterday):
       - Logs the time period header (e.g., Last Month:) with two spaces prefix.
       - Iterates over allServices using forEach and logs each service’s cost:
         * Format: ${service}: $${serviceTotals[period][service].toFixed(2)} (four spaces prefix for indentation).
       - Adds an empty line between periods.
* **Example Output (with isSummary = false)**:

```

Account: MyAccount

Totals:

Last Month: $123.45

This Month: $67.89

Last 7 Days: $45.67

Yesterday: $12.34

Totals by Service:

Last Month:

EC2: $73.45

S3: $50.00

This Month:

EC2: $37.89

S3: $30.00

Last 7 Days:

EC2: $25.67

S3: $20.00

Yesterday:

EC2: $7.34

S3: $5.00

```

### **Assumptions About TotalCosts**

* The TotalCosts type likely has:
  + totals: An object with properties:
    - lastMonth: number
    - thisMonth: number
    - last7Days: number
    - yesterday: number
  + totalsByService: An object with properties:
    - lastMonth: { [service: string]: number }
    - thisMonth: { [service: string]: number }
    - last7Days: { [service: string]: number }
    - yesterday: { [service: string]: number }

**src/account.ts:**

### **File Overview**

* **Imports**:
  + AWS from aws-sdk: The AWS SDK for JavaScript, used to interact with AWS services.
  + AWSConfig from ./config: A type or interface defining the configuration for AWS SDK clients.
  + showSpinner from ./logger: A function to display a loading spinner in the console.
* **Purpose**: The file exports a function to retrieve an AWS account alias, falling back to the account ID if no alias is found.

### **Function: getAccountAlias (Exported)**

* **Purpose**: Fetches the alias for an AWS account using the IAM service; if no alias exists, it retrieves the account ID using the STS service.
* **Parameters**:
  1. awsConfig: AWSConfig: Configuration object for AWS SDK clients (e.g., credentials, region).
* **Return Type**: Promise<string> - Resolves to the account alias or account ID as a string.
* **Variables and Logic**:
  1. **Spinner Setup**:
     + showSpinner('Getting account alias'): Displays a loading spinner with the message "Getting account alias" to indicate progress.
  2. **Attempt to Get Alias via IAM**:
     + const iam = new AWS.IAM(awsConfig): Creates an IAM client instance with the provided configuration.
     + const accountAliases = await iam.listAccountAliases().promise(): Calls the listAccountAliases API to retrieve the account’s aliases asynchronously. The .promise() method converts the AWS SDK callback-style request into a Promise.
     + const foundAlias = accountAliases?.['AccountAliases']?.[0]: Extracts the first alias from the response, using optional chaining to safely handle cases where the response might be undefined or empty.
     + If foundAlias exists (i.e., is truthy), returns it immediately.
  3. **Fallback to Account ID via STS**:
     + If no alias is found, proceeds to use the STS service.
     + const sts = new AWS.STS(awsConfig): Creates an STS (Security Token Service) client instance with the provided configuration.
     + const accountInfo = await sts.getCallerIdentity().promise(): Calls the getCallerIdentity API to retrieve identity information, including the account ID, asynchronously.
     + return accountInfo?.Account || '': Returns the Account property (the AWS account ID) from the response, or an empty string if accountInfo is undefined or lacks an Account property.

### 

### **Assumptions About AWSConfig**

* AWSConfig is likely a configuration object compatible with the AWS SDK, possibly including:
  + accessKeyId: string
  + secretAccessKey: string
  + region: string
  + Other optional settings like sessionToken for temporary credentials.
* It matches the type expected by AWS.IAM and AWS.STS constructors.

### **Example Usage and Output**

* **With an Alias**:
  + If the IAM listAccountAliases call returns { AccountAliases: ['my-account-alias'] }, the function returns "my-account-alias".
* **Without an Alias**:
  + If listAccountAliases returns { AccountAliases: [] } (or no aliases), and getCallerIdentity returns { Account: '123456789012', ... }, the function returns "123456789012".
  + If both calls fail or return no data, it returns "".

### **Error Handling**

* The function assumes the AWS SDK will handle authentication or network errors by rejecting the promises, which would need to be caught by the caller.
* Optional chaining (?.) prevents runtime errors if API responses are malformed or missing expected properties.

**src/config.ts:**

### **File Overview**

* **Imports**:
  + fs from node:fs: Node.js file system module (though not directly used in this file).
  + loadSharedConfigFiles from @aws-sdk/shared-ini-file-loader: A utility from the AWS SDK to load credentials and config from AWS shared INI files (e.g., ~/.aws/credentials, ~/.aws/config).
  + chalk from chalk: A library for styling console output with colors.
  + printFatalError from ./logger: A function to print an error message and likely terminate the process.
* **Purpose**: The file defines types and functions to configure AWS SDK credentials and region, either from command-line options or AWS shared configuration files.

### **Types**

1. **EnvConfig**:
   * **Purpose**: Represents environment-based AWS configuration (not directly used in this file but exported).
   * **Structure**:
     + awsAccessKey: string: AWS access key ID.
     + awsSecretKey: string: AWS secret access key.
     + awsRegion: string: AWS region.
2. **AWSConfig**:
   * **Purpose**: Defines the configuration object expected by AWS SDK clients.
   * **Structure**:
     + credentials: An object with:
       - accessKeyId: string: AWS access key ID.
       - secretAccessKey: string: AWS secret access key.
       - sessionToken: string: Optional AWS session token (for temporary credentials).
     + region: string: AWS region.

### **Function: getAwsConfigFromOptionsOrFile (Exported)**

* **Purpose**: Constructs an AWSConfig object either from provided command-line options or by loading credentials from AWS shared config files.
* **Parameters**:
  1. options: An object with:
     + profile: string: AWS profile name (e.g., "default").
     + accessKey: string: AWS access key ID (optional).
     + secretKey: AWS secret key (optional).
     + sessionToken: AWS session token (optional).
     + region: string: AWS region.
* **Return Type**: Promise<AWSConfig> - Resolves to an AWS configuration object.
* **Variables and Logic**:
  1. **Destructuring Options**:
     + const { profile, accessKey, secretKey, sessionToken, region } = options: Extracts values from the options object.
  2. **Manual Credentials Check**:
     + If either accessKey or secretKey is provided (accessKey || secretKey):
       - Validates that both are present; if not, calls printFatalError with a message indicating both --access-key and --secret-key are required.
       - Returns an AWSConfig object with:
         * credentials: { accessKeyId: accessKey, secretAccessKey: secretKey, sessionToken: sessionToken }.
         * region: The provided region.
  3. **Fallback to File-Based Credentials**:
     + If no manual credentials are provided, returns an AWSConfig object with:
       - credentials: Result of await loadAwsCredentials(profile).
       - region: The provided region.

### **Function: loadAwsCredentials**

* **Purpose**: Loads AWS credentials from shared configuration files for a given profile, with validation and error handling.
* **Parameters**:
  1. profile: string = 'default': The AWS profile to load credentials for (defaults to "default").
* **Return Type**: Promise<AWSConfig['credentials'] | undefined> - Resolves to a credentials object or undefined (though undefined is not returned due to error handling).
* **Variables and Logic**:
  1. **Load Config Files**:
     + const configFiles = await loadSharedConfigFiles(): Loads AWS credentials and config files (typically ~/.aws/credentials and ~/.aws/config) using the AWS SDK utility.
     + const credentialsFile = configFiles.credentialsFile: Extracts the credentials file data (an object mapping profiles to their settings).
  2. **Extract Credentials**:
     + const accessKey: string = credentialsFile?.[profile]?.aws\_access\_key\_id: Gets the access key for the profile.
     + const secretKey: string = credentialsFile?.[profile]?.aws\_secret\_access\_key: Gets the secret key.
     + const sessionToken: string = credentialsFile?.[profile]?.aws\_session\_token: Gets the session token (optional).

**src/cost.ts:**  
  
**File Overview**

* **Imports**:
  + AWS from aws-sdk: The AWS SDK for interacting with AWS services (specifically Cost Explorer).
  + dayjs from dayjs: A lightweight library for date manipulation.
  + AWSConfig from ./config: A type defining the AWS SDK configuration (credentials and region).
  + showSpinner from ./logger: A function to display a loading spinner in the console.
* **Purpose**: The file provides functionality to fetch AWS cost data, process it, and return structured totals for different time periods, both overall and by service.

### **Types**

1. **RawCostByService**:
   * **Purpose**: Represents raw cost data fetched from AWS Cost Explorer, organized by service and date.
   * **Structure**: { [key: string]: { [date: string]: number } }
     + Outer key: AWS service name (e.g., "S3", "EC2").
     + Inner key: Date in YYYY-MM-DD format.
     + Value: Cost as a number for that service on that date.
2. **TotalCosts**:
   * **Purpose**: Represents processed cost data with totals for specific time periods, both overall and by service.
   * **Structure**:
     + totals: Overall costs:
       - lastMonth: number
       - thisMonth: number
       - last7Days: number
       - yesterday: number
     + totalsByService: Costs by service:
       - lastMonth: { [key: string]: number }
       - thisMonth: { [key: string]: number }
       - last7Days: { [key: string]: number }
       - yesterday: { [key: string]: number }

### **Function: getRawCostByService (Exported)**

* **Purpose**: Fetches raw cost data from AWS Cost Explorer for the past 66 days, grouped by service and date.
* **Parameters**:
  + awsConfig: AWSConfig: Configuration for the AWS SDK (credentials and region).
* **Return Type**: Promise<RawCostByService> - Resolves to raw cost data.
* **Variables and Logic**:
  + **Setup**:
    - showSpinner('Getting pricing data'): Shows a spinner to indicate data retrieval.
    - const costExplorer = new AWS.CostExplorer(awsConfig): Initializes a Cost Explorer client.
    - const endDate = dayjs().subtract(1, 'day'): Sets the end date to yesterday.
    - const startDate = endDate.subtract(65, 'day'): Sets the start date to 66 days ago (65 days + yesterday).
  + **API Call**:
    - costExplorer.getCostAndUsage({...}).promise(): Fetches cost data with:
      * TimePeriod: From startDate to endDate in YYYY-MM-DD format.
      * Granularity: 'DAILY': Daily cost breakdown.
      * Filter: Excludes Credit, Refund, Upfront, and Support record types.
      * Metrics: ['UnblendedCost']: Retrieves unblended cost metric.
      * GroupBy: Groups results by the SERVICE dimension.
  + **Process Results**:
    - const costByService = {}: Initializes an empty object for results.
    - Iterates over pricingData.ResultsByTime (daily results):
      * For each group in day.Groups:
        + serviceName = group.Keys[0]: Extracts the service name.
        + cost = group.Metrics.UnblendedCost.Amount: Gets the cost as a string.
        + costDate = day.TimePeriod.End: Gets the date.
        + Stores the parsed cost (parseFloat(cost)) in costByService[serviceName][costDate].
  + **Return**: costByService.
* **Example Output**:

```

{

"S3": { "2025-03-21": 5.00, "2025-03-20": 4.50, ... },

"EC2": { "2025-03-21": 7.34, "2025-03-20": 6.80, ... }

}

```

### **Function: calculateServiceTotals**

* **Purpose**: Processes raw cost data into structured totals for specific time periods, both overall and by service.
* **Parameters**:
  + rawCostByService: RawCostByService: Raw cost data from getRawCostByService.
* **Return Type**: TotalCosts - Processed totals.
* **Variables and Logic**:
  + **Initialization**:
    - totals: Object with all properties set to 0 for lastMonth, thisMonth, last7Days, and yesterday.
    - totalsByService: Object with empty objects for each time period.
    - Date markers using dayjs:
      * startOfLastMonth: Start of the previous month.
      * startOfThisMonth: Start of the current month.
      * startOfLast7Days: 7 days ago.
      * startOfYesterday: Yesterday.
  + **Processing**:
    - Iterates over each service in rawCostByService:
      * servicePrices = rawCostByService[service]: Gets costs for the service.
      * Initializes service-specific totals (e.g., lastMonthServiceTotal = 0).
      * Iterates over each date in servicePrices:
        + price = servicePrices[date]: Cost for that date.
        + dateObj = dayjs(date): Converts date string to a dayjs object.
        + Adds price to appropriate totals based on conditions:

isSame(startOfLastMonth, 'month'): Adds to lastMonthServiceTotal.

isSame(startOfThisMonth, 'month'): Adds to thisMonthServiceTotal.

isSame(startOfLast7Days, 'week') && !isSame(startOfYesterday, 'day'): Adds to last7DaysServiceTotal (excludes yesterday).

isSame(startOfYesterday, 'day'): Adds to yesterdayServiceTotal.

* + - * Stores service totals in totalsByService[period][service].
      * Adds service totals to overall totals.
  + **Return**: { totals, totalsByService }.
* **Note**: The last7Days calculation uses isSame(..., 'week'), which may include more than 7 days if the week spans beyond 7 days from today. A more precise approach might use isAfter(startOfLast7Days).

### **Function: getTotalCosts (Exported)**

* **Purpose**: Combines getRawCostByService and calculateServiceTotals to fetch and process cost data in one step.
* **Parameters**:
  + awsConfig: AWSConfig: AWS SDK configuration.
* **Return Type**: Promise<TotalCosts> - Resolves to processed cost totals.
* **Logic**:
  + const rawCosts = await getRawCostByService(awsConfig): Fetches raw data.
  + const totals = calculateServiceTotals(rawCosts): Processes it.
  + **Return**: totals.

Example Output (TotalCosts)

```

{

"totals": {

"lastMonth": 123.45,

"thisMonth": 67.89,

"last7Days": 45.67,

"yesterday": 12.34

},

"totalsByService": {

"lastMonth": { "S3": 50.00, "EC2": 73.45 },

"thisMonth": { "S3": 30.00, "EC2": 37.89 },

"last7Days": { "S3": 20.00, "EC2": 25.67 },

"yesterday": { "S3": 5.00, "EC2": 7.34 }

}

}

```

**src/index.ts:**  
  
**File Overview**

* **Imports**:
  + Command from commander: A library for building command-line interfaces (CLI).
  + packageJson from ../package.json (with assert { type: 'json' }): Imports the package metadata (version, description).
  + Functions from other modules:
    - getAccountAlias from ./account: Retrieves the AWS account alias or ID.
    - getAwsConfigFromOptionsOrFile from ./config: Constructs AWS SDK configuration.
    - getTotalCosts from ./cost: Fetches and processes AWS cost data.
    - Printers from ./printers/\*:
      * printFancy from ./fancy: Outputs costs in a colored, tabular format.
      * printJson from ./json: Outputs costs as JSON.
      * notifySlack from ./slack: Sends costs to a Slack channel.
      * printPlainText from ./text: Outputs costs as plain text.
* **Purpose**: This is the main entry point of an AWS cost reporting CLI tool, parsing command-line options and orchestrating the retrieval and display of cost data.

### **Initial Setup**

* **AWS SDK Maintenance Mode Suppression**:
  + process.env.AWS\_SDK\_JS\_SUPPRESS\_MAINTENANCE\_MODE\_MESSAGE = '1': Suppresses a maintenance mode warning from AWS SDK v2 (noted for upgrade to v3).
* **CLI Setup**:
  + const program = new Command(): Initializes a new Commander instance for CLI parsing.
  + Configures the CLI:
    - .version(packageJson.version): Sets the version from package.json.
    - .name('aws-cost'): Sets the command name.
    - .description(packageJson.description): Sets the description from package.json.
    - Defines options with defaults and descriptions:
      * -p, --profile [profile]: AWS profile (default: "default").
      * -k, --access-key [key]: AWS access key.
      * -s, --secret-key [key]: AWS secret key.
      * -T, --session-token [key]: AWS session token.
      * -r, --region [region]: AWS region (default: "us-east-1").
      * -j, --json: Output as JSON.
      * -u, --summary: Output summary only (no service breakdown).
      * -t, --text: Output as plain text.
      * -S, --slack-token [token]: Slack API token.
      * -C, --slack-channel [channel]: Slack channel.
      * -h, --help: Show help.
    - .parse(process.argv): Parses command-line arguments.

### **Type: OptionsType**

* **Purpose**: Defines the structure of the parsed command-line options.
* **Structure**:
  + AWS credentials:
    - accessKey: string
    - secretKey: string
    - sessionToken: string
    - region: string
  + AWS profile:
    - profile: string
  + Output variants:
    - text: boolean
    - json: boolean
    - summary: boolean
  + Slack integration:
    - slackToken: string
    - slackChannel: string
  + Other:
    - help: boolean
* **Usage**: const options = program.opts<OptionsType>() retrieves the parsed options with this type.

### **Main Logic**

1. **Help Handling**:
   * if (options.help): If the --help flag is provided, displays the help text via program.help() and exits with code 0.
2. **AWS Configuration**:
   * const awsConfig = await getAwsConfigFromOptionsOrFile({...}): Constructs AWS SDK configuration using provided options:
     + profile: options.profile
     + accessKey: options.accessKey
     + secretKey: options.secretKey
     + sessionToken: options.sessionToken
     + region: options.region
3. **Account Alias**:
   * const alias = await getAccountAlias(awsConfig): Fetches the AWS account alias (or ID if no alias exists).
4. **Cost Data**:
   * const costs = await getTotalCosts(awsConfig): Retrieves and processes AWS cost data.
5. **Output Selection**:
   * **JSON Output**: if (options.json):
     + Calls printJson(alias, costs, options.summary) to output costs as JSON.
   * **Plain Text Output**: else if (options.text):
     + Calls printPlainText(alias, costs, options.summary) for plain text output.
   * **Default (Fancy Output)**: else:
     + Calls printFancy(alias, costs, options.summary) for a colored, tabular output.
6. **Slack Notification**:
   * if (options.slackToken && options.slackChannel):
     + Calls notifySlack(alias, costs, options.summary, options.slackToken, options.slackChannel) to send the report to Slack.

### **Example Usage**

* **Default (Fancy Output)**:

```

aws-cost --profile my-profile

```

* Outputs costs in a colored table for "my-profile".

**JSON Summary**:

```

aws-cost --json --summary --region us-east-1

```

* Outputs summary costs as JSON.

**Plain Text with Slack**:

```

aws-cost --text --slack-token xoxb-123 --slack-channel #costs

```

**src/logger.ts:**  
  
**File Overview**

* **Imports**:
  + chalk from chalk: A library for styling console output with colors and formatting.
  + ora and Ora from ora: A library for creating terminal spinners, with Ora as the spinner type.
* **Purpose**: This file provides utility functions for logging errors and managing a spinner to indicate ongoing processes in the console.

### **Variable: spinner**

* **Type**: Ora | undefined
* **Purpose**: A global variable to hold the spinner instance, allowing it to persist across function calls. Initially undefined until a spinner is created.

### **Function: printFatalError (Exported)**

* **Purpose**: Prints a formatted error message to the console and terminates the process with an error code.
* **Parameters**:
  + error: string: The error message to display.
* **Logic**:
  + Uses console.error to output:
    - An empty line for spacing.
    - "Error:" in bold, red, and underlined using chalk.bold.redBright.underline.
    - The error message in red using chalk.redBright.
  + process.exit(1): Exits the process with a failure code (1).
* **Example Output**:

```

Error:

Could not find AWS credentials

```

* The word "Error" would be bold, red, and underlined, and the message "Could not find AWS credentials" would be red.

### **Function: showSpinner (Exported)**

* **Purpose**: Displays or updates a terminal spinner with a given text message to indicate an ongoing process.
* **Parameters**:
  + text: string: The text to display alongside the spinner.
* **Logic**:
  + If spinner is not yet initialized (!spinner):
    - Creates a new spinner with ora({ text: '' }).start() and assigns it to spinner.
    - The empty initial text ensures the spinner starts cleanly.
  + Updates the spinner’s text: spinner.text = text.
* **Behavior**:
  + The spinner persists across calls, reusing the same instance unless stopped.
  + Shows a spinning animation (e.g., dots or a line) followed by the provided text.
* **Example**:
  + showSpinner('Getting pricing data') displays:

```

⠋ Getting pricing data

```

* The ⠋ (or similar) animates until stopped.

### **Function: hideSpinner (Exported)**

* **Purpose**: Stops and hides the spinner, clearing it from the console.
* **Return Type**: void (implicitly returns undefined if no spinner exists).
* **Logic**:
  + Checks if spinner exists (if (!spinner)):
    - If not, returns early with no action.
  + Calls spinner.stop() to stop the spinner and remove it from the terminal.
* **Behavior**:
  + After stopping, the spinner’s line is cleared, leaving the console ready for subsequent output.

### **Usage Context**

* These functions are likely used across the application (e.g., in cost.ts, account.ts) to:
  + Show progress during AWS API calls (showSpinner).
  + Hide the spinner when operations complete (hideSpinner).
  + Report errors and terminate if something goes wrong (printFatalError).

**dist/index.js:**

### **File Overview**

* **Purpose**: This is the compiled, distributable version of the AWS cost reporting CLI tool, combining multiple TypeScript modules (index.ts, account.ts, config.ts, cost.ts, logger.ts, and printer files) into a single JavaScript file.
* **Structure**: The file includes:
  + Imported dependencies and their usage.
  + The package.json content as a default export.
  + Compiled functions from various modules.
  + The main CLI logic from index.ts.
* **Key Differences from Source**:
  + Type annotations are removed (e.g., OptionsType is implicit).
  + Modules are inlined rather than imported as separate files.
  + Variable names may be minified or prefixed (e.g., awsConfig2 instead of awsConfig due to scope collision avoidance).
  + Source mapping is included (//# sourceMappingURL=index.js.map) for debugging.

### **Key Components**

#### **1. package.json (Embedded)**

* **Content**: Defines metadata for the aws-cost-cli package:
  + name: "aws-cost-cli"
  + version: "0.2.7"
  + description: "A CLI tool to perform cost analysis on your AWS account"
  + Dependencies: @aws-sdk/shared-ini-file-loader, aws-sdk, chalk, commander, dayjs, node-fetch, ora, etc.
  + Bin: "aws-cost": "./bin/index.js"
* **Purpose**: Used to set the CLI version and description in program.

#### **2. Compiled Modules**

The file inlines functions from the original TypeScript files:

* **logger.ts**:
  + printFatalError(error): Logs a red error message and exits.
  + showSpinner(text): Starts or updates a spinner with ora.
  + hideSpinner(): Stops the spinner.
  + spinner: A global variable for the spinner instance.
* **account.ts**:
  + getAccountAlias(awsConfig2): Fetches the AWS account alias via IAM or falls back to the account ID via STS.
* **config.ts**:
  + getAwsConfigFromOptionsOrFile(options2): Constructs AWS config from CLI options or shared config files.
  + loadAwsCredentials(profile): Loads credentials from AWS shared files, with error handling.
* **cost.ts**:
  + getRawCostByService(awsConfig2): Fetches 66 days of cost data from Cost Explorer.
  + calculateServiceTotals(rawCostByService): Processes raw costs into totals by period and service.
  + getTotalCosts(awsConfig2): Combines the above into a single call.
* **printers/fancy.ts**:
  + printFancy(accountAlias, totals, isSummary): Outputs a colored, tabular cost report.
* **printers/json.ts**:
  + printJson(accountAlias, totalCosts, isSummary): Outputs costs as JSON.
* **printers/slack.ts**:
  + formatServiceBreakdown(costs2): Formats yesterday’s service costs for Slack.
  + notifySlack(accountAlias, costs2, isSummary, slackToken, slackChannel): Sends costs to Slack.
* **printers/text.ts**:
  + printPlainSummary(accountAlias, costs2): Prints a plain text summary.
  + printPlainText(accountAlias, totals, isSummary): Prints plain text with optional service breakdown.

#### **3. Main CLI Logic (index.ts)**

* **Setup**:
  + Suppresses AWS SDK maintenance mode message: process.env.AWS\_SDK\_JS\_SUPPRESS\_MAINTENANCE\_MODE\_MESSAGE = "1".
  + Initializes program with Command from commander.
* **CLI Configuration**:
  + Sets version, name, and description from package\_default.
  + Defines options (e.g., --profile, --json, --slack-token) with defaults (e.g., region: "us-east-1").
  + Parses process.argv.
* **Execution**:
  + options = program.opts(): Retrieves parsed options.
  + If options.help, shows help and exits.
  + Constructs awsConfig using getAwsConfigFromOptionsOrFile.
  + Fetches alias with getAccountAlias.
  + Gets costs with getTotalCosts.
  + Outputs based on flags:
    - options.json: Calls printJson.
    - options.text: Calls printPlainText.
    - Default: Calls printFancy.
  + If slackToken and slackChannel are provided, calls notifySlack.

### **Key Observations**

1. **Compilation Artifacts**:
   * Variable renaming (e.g., awsConfig2, chalk2) avoids naming conflicts in the single-file scope.
   * Optional chaining (?.) and nullish coalescing remain from TypeScript safety features.
   * Async/await syntax is preserved for Promise-based operations.
2. **Dependencies**:
   * Inline imports like chalk, ora, node-fetch, and dayjs are resolved at runtime from node\_modules.
3. **Execution Flow**:
   * The file runs as a standalone script, invoked via node dist/index.js or the aws-cost bin command.

### **Example Usage (Same as index.ts)**

* **Default Output**:

```

node dist/index.js --profile my-profile

```

* Runs printFancy with costs for "my-profile".

**JSON Output**:

```

node dist/index.js --json --summary

```

* Outputs summary costs as JSON.

**Slack Integration**:

```

* + Sends costs to Slack.