Published **January 17, 2025**

Extend your Kubernetes automation with the Palette Go SDK



Carolina Delwing Rosa
Education Engineer

At Spectro Cloud, we're always looking for ways to help platform teams and app developers manage their complex Kubernetes workloads more effectively.

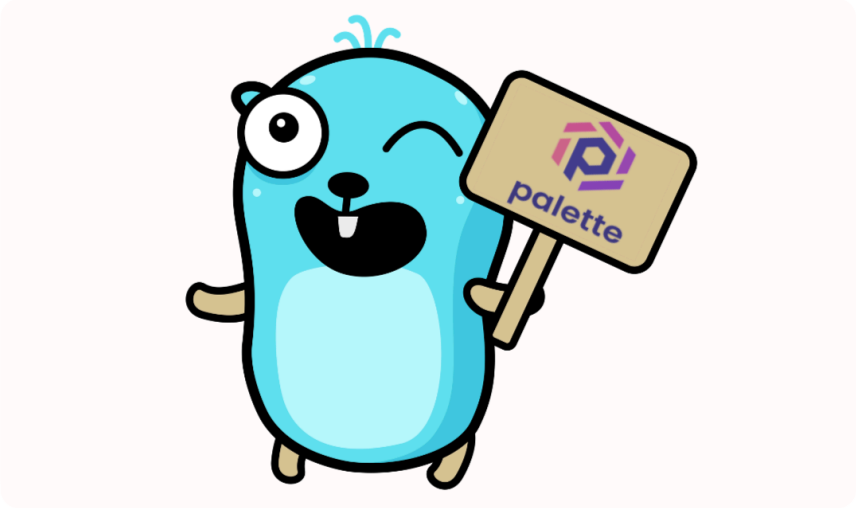
For many teams, that means interacting with our featureset not only through our CLI or clickops GUI, but programmatically through our application programming interface (API). Doing so helps automate and scale operations, driving cost down and consistency up. That's always a good thing.

We're proud of the quality of our API. It's complete and well documented, and many of our customers use it extensively. But there's always room to take it further.

So a few months ago we published the first release of our **Go SDK for Palette**, in response to requests from several of our biggest customers. We've already seen folks using it and getting value from it, and we plan to keep the SDK updated with future releases — so now's a good time to tell you all about it.

In this blog, we'll explore how you can benefit from using an SDK, walk you through getting started with the Palette Go SDK, and dive into an example of integrating the SDK with GitHub Actions and Slack to

identify clusters running for more than 24 hours in your Palette development environments. After all, cleaning up unused infrastructure resources, optimizing costs, and adhering to FinOps best practices should always be a priority!



What exactly is an SDK?

A Software Development Kit (SDK) is a set of tools that helps developers interact with a specific platform and its ecosystem.

Think of it as a toolkit containing all the components that are required to integrate a platform or service with your software applications, extending their functionalities, implementing automation solutions, and much more.

SDKs are programming language specific. Since our APIs are written in Go (like K8s itself), we generated our SDK for Go too. Although we don't plan to support other languages, like Python, we do expose the entire Palette OpenAPI spec inside our public SDK repo, so if you're motivated you're welcome to integrate it with tools like swagger-codegen or openapi-generator to stub out SDKs for other languages.

SDK vs API

If you've already heard of the Palette API, you might be wondering how it differs from an SDK and which one you should use. The truth is, they're not competitors — they work together as part of the same system. APIs define how different software components communicate, serving as the interface for sending and receiving data. SDKs, on the other hand, are toolkits that may include the API alongside other utilities, simplifying the process of integrating and interacting with a system.

So, while you can call an API directly in your code, this often involves configuring parameters, handling authentication, and parsing raw responses, which can be time consuming and error-prone. SDKs handle much of this for you, offering methods that abstract away the complexities by providing wrapper functions that make it easier to interact with the underlying system.

For example, the AWS SDK for Go provides pre-built methods, models, and utilities to interact with AWS services like S3 and EC2. Similarly, the Palette Go SDK simplifies the process of interacting with

the Palette API and enables users to leverage Go to provision and manage clusters and their underlying infrastructure resources in the multi-cloud environments supported by Palette.

Why use an SDK?

At this point, you might have noticed that SDKs offer several benefits to make the lives of engineering teams easier. Here are some key advantages worth emphasizing:

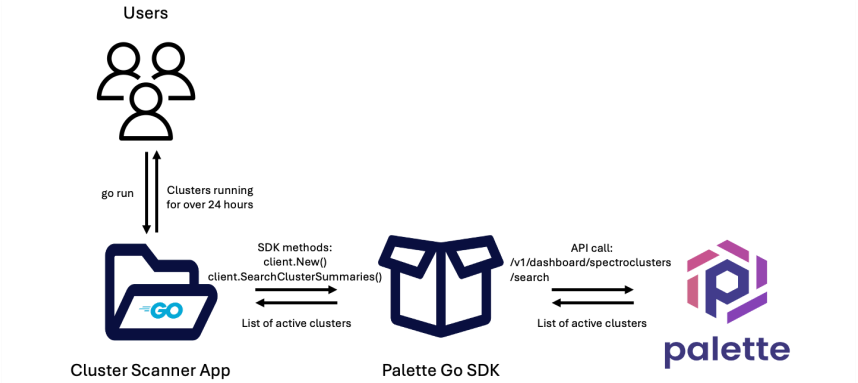
- **Readable & Simplified Code:** As mentioned in the SDK vs API section, SDKs provide pre-built methods that abstract calls to external services, eliminating the need to manually construct HTTP requests and parse JSON responses.
- **Built-In Authentication:** SDKs handle authentication smoothly, ensuring secure integration without requiring custom authentication logic.
- **Standardized Integration & Interaction:** By encapsulating standard processes, SDKs reduce code complexity. This ensures, for example, consistent patterns across automation workflows and infrastructure management tasks.
- **Faster Development:** Pre-built SDK components accelerate the implementation of features, reducing the development cycle time. This approach also lowers operational costs and enables teams to deliver new features faster.

Introducing the Palette Go SDK through the Cluster Scanner application

The Palette Go SDK provides Go developers with a user-friendly client for interacting with the Palette APIs, abstracting complexity and simplifying operations with Palette. The SDK enables you to create and manage Palette infrastructure resources using Go, such as CRUD operations on clusters and cluster profiles.

Now, imagine you're an engineering manager working at a company that recently adopted Palette to manage its multi-cloud infrastructure. Your team has clusters deployed across major public cloud providers, as well as a development environment for testing. You notice a recurring issue: dev clusters left running without purpose, causing cloud costs to spiral.

To address this, you decide to leverage the potential of the Palette Go SDK and build a simple application, called Cluster Scanner, to alert your team about clusters that have been running for more than 24 hours. The diagram below shows a high-level overview of the app.



The Cluster Scanner app has one internal package and is organized with the following directory structure:

```
cluster-scanner/
├── internal/
│ ├── format_age.go
│ ├── format_age_test.go
│ ├── search_clusters.go
│ ├── search_old_clusters.go
│ ├── search_old_clusters_test.go
└── main.go
```

Let's explore the files that make up the application to see how you can leverage the superpowers of the Palette Go SDK. You can find the complete code and detailed instructions on how to deploy it in the Palette Samples GitHub repository.

- **main.go:** This is the entry point of the Cluster Scanner app. It initializes the Palette client using credentials provided via environment variables (PALETTE_HOST, PALETTE_API_KEY, and optionally PALETTE_PROJECT_UID). The client supports authentication using API keys, JWT tokens, or username and password.

```
paletteClient := client.New
```

```
(
        client.
WithPaletteURI
(host),
        client.
WithAPIKey
(apiKey),
)
```

Then, it configures the client scope to operate at the tenant or project level.

```
if
 projectUid !=
""
 {
        client.
WithScopeProject
(projectUid)(paletteClient)
        logger.
Info
(
"Setting scope to project."
)
}
else
 {
        client.
WithScopeTenant
()(paletteClient)
        logger.
Info
(
"Setting scope to tenant."
)
}
```

Finally, it invokes the *SearchClusters* internal function to search for clusters and identify those running for over 24 hours.

- **internal/search_clusters.go:** Implements the *SearchClusters* function, which interacts directly with the Palette SDK through the *SearchClusterSummaries* method to retrieve a list of active clusters in the Palette environment.

```
func
SearchClusters
```

```
(paletteClient *client.
V1Client
) ([]*models.
V1SpectroClusterSummary
, error) {

return
 paletteClient.
SearchClusterSummaries
(&models.
V1SearchFilterSpec
{}, []*models.
V1SearchFilterSortSpec
{})
}
```

- **internal/format_age.go:** Implements functions to calculate and format the age of a cluster in terms of weeks, days, and hours.
- **internal/search_old_clusters.go:** Implements the *SearchOldClusters* function, which identifies clusters running for more than 24 hours and returns an array of messages with information about each cluster, such as the cluster name, cloud type, and age.

Setting up on your local machine

Deploying the Cluster Scanner application locally requires just a few steps. Before you begin, ensure you have Go (v1.22 or later) installed, a Palette API key, and the Palette Samples repository available or your local machine. Once ready, follow these instructions.

1. Open a terminal window and export the required environment variables. Replace *<your-palette-url>* with your Palette URL (e.g, *console.spectrocloud.com*), replace *<your-api-key>* with your Palette API key, and optionally *<palette-project-uid>* with a Palette project UID to scan a specific project. If no project is provided, the tool assumes a tenant scope and scans clusters across all projects.

```
export

PALETTE_HOST
=<your-palette-url>
export

PALETTE_API_KEY
=<your-api-key>
export

PALETTE_PROJECT_UID
```

```
=<palette-project-uid>
```

2. Navigate to the *cluster-scanner* folder and install the required Palette SDK modules.

```
cd cluster-scanner
go get ./...
```

3. Start the application using the following command.

```
go run .
```

The app will print the clusters that have been active in your Palette environment for more than 24 hours

The app will print the clusters that have been active in your Palette environment for more than 24 hours

```
time=
2024
-
12
-24
T11
:
37
:
33.016
-
03
:
00
 level=
INFO
 msg=
"Setting scope to tenant."

time=
2024
-
12
-24
T11
:
```

```
37
:
33.016
-
03
:
00
 level=
INFO
 msg=
"Searching for clusters..."

time=
2024
-
12
-24
T11
:
37
:
33.819
-
03
:
00
 level=
INFO
 msg=
"The following clusters have been running for over 24 hours. Please delete
them if they're no longer needed:"

time=
2024
-
12
-24
T11
:
37
:
33.819
-
03
:
00
 level=
INFO
 msg=
"❗ edge-native cluster 'edge-cluster-blog' - 1d 23h ⌛"
```

Integrating with a GitHub Actions workflow

The app is already working locally, but now you want it to run automatically every week and notify the team if any clusters older than 24 hours are found. One possible solution is to create a GitHub Actions workflow and integrate it with a collaboration tool, such as Slack.

The example GitHub Actions pipeline below scans your Palette environment every Friday for running clusters. Under the hood, it executes the Cluster Scanner Go application, formats the output, and sends a Slack message to a specific channel. All you need to do is place this YAML file in the *.github* folder of the same repository where the Cluster Scanner files are located and configure the necessary Palette and Slack credentials as GitHub secrets. For detailed, step-by-step instructions, visit the Cluster Scanner page on GitHub.

```
name
:
Cluster

Scanner


on
:

schedule
:
   -
cron
:
"30 9 * * 5"


env
:

PALETTE_API_KEY
: ${{ secrets.
PALETTE_API_KEY
 }}

PALETTE_HOST
: ${{ secrets.
PALETTE_HOST
 }}

PALETTE_PROJECT_UID
: ${{ secrets.
PALETTE_PROJECT_UID
 }}

jobs
```

```
:
 scan-
clusters
:

name
: cluster-scan
   runs-
on
: ubuntu-latest

steps
:
     -
name
:
Checkout

Repository


uses
: actions/checkout@v4

     -
name
:
Set

Up

Go


uses
: actions/setup-go@v5

with
:
         go-version-
file
:
"scripts/cluster-scanner/go.mod"


     -
name
:
Install

Dependencies

       working-
directory
```

```
        : scripts/cluster-scanner

        run
        : go get ./...

            -
        name
        :
        Build
         and
        Run
         the
        App

              working-
        directory
        : scripts/cluster-scanner
        run
        : |
                set -e
                go build -o cluster-scanner
                ./cluster-scanner | tee result.
        log


            -
        name
        :
        Get

        Clusters

        with

        More

        Than

        24

        Hours
         and
        Format

        Output

              working-
        directory
        : scripts/cluster-scanner

        run
        : |

        if
```

```
 grep -q
"The following clusters have been running"
 result.
log
; then
          echo
"CLUSTERS_FOUND=true"
 >> $GITHUB_ENV
          {
            echo
'LOG_MESSAGE<<EOF'

            sed
's/^.*msg=//'
 result.
log
 | sed -n
'/The following clusters/,/$/p'
 | sed
's/"//g'

            echo
EOF

          } >>
"$GITHUB_ENV"

        fi

    -
name
:
Send

Slack

Notification


if
: env.
CLUSTERS_FOUND
 ==
'true'


uses
: rtCamp/action-slack-notify@v2
.3
.2


env
```

```
:

SLACK_WEBHOOK
: ${{ secrets.
SLACK_PRIVATE_TEAM_WEBHOOK
 }}

SLACK_COLOR
:
"good"


SLACKIFY_MARKDOWN
:
true


ENABLE_ESCAPES
:
true


SLACK_MESSAGE
: ${{ env.
LOG_MESSAGE
 }}

     -
name
:
Slack

Notification
 on
Failure


if
: ${{
failure
() }}

uses
: rtCamp/action-slack-notify@v2
.3
.2


env
:

SLACK_WEBHOOK
: ${{ secrets.
SLACK_PRIVATE_TEAM_WEBHOOK
```

```
 }}

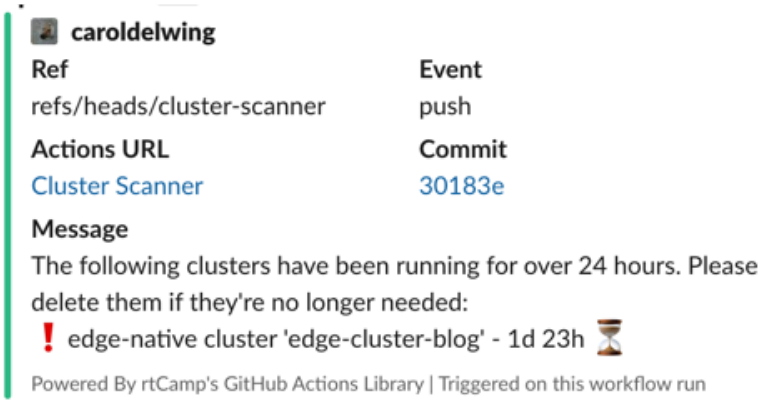
SLACK_COLOR
:
"danger"


SLACKIFY_MARKDOWN
:
true


ENABLE_ESCAPES
:
true


SLACK_MESSAGE
:
"The cluster scan job for `${{ github.workflow }}` in `${{
github.repository }}` failed. [View details](${{ github.server_url }}/${{
github.repository }}/actions/runs/${{ github.run_id }})."
```

Below is an example of how the Slack notification should look. This environment had one Palette Edge cluster running for over 24 hours.



Running Go tests

Last but definitely not least, you can leverage Go's powerful testing ecosystem to ensure the reliabilit of your application that has a dependency on the Palette Go SDK. Specifically, the Palette SDK Model library plays a pivotal role in creating different test cases, as it defines the communication format between the cluster scanner app and the SDK.

For example, we used the *V1SpectroClusterSummary* model and the table-driven test design pattern to test cluster data with different inputs, such as creation timestamps, names, and cloud types. This

allowed us to test edge cases and validate key functions like *SearchOldClusters* and *FormatAge*. Here is how the V1SpectroClusterSummary model was used to simulate clusters:

```
createSummary := func (creationTime string, name string, cloudType string)
*models.
V1SpectroClusterSummary
 {

return
 &models.
V1SpectroClusterSummary
 {

Metadata
: &models.
V1ObjectMeta
{

CreationTimestamp
: models.
V1Time
(
parseTime
(now, creationTime)),

Name
: name,
          },

SpecSummary
: &models.
V1SpectroClusterSummarySpecSummary
{

CloudConfig
: &models.
V1CloudConfigMeta
{

CloudType
: cloudType,
            },
        },
    }
}
```

This test approach helped us catch issues early and ensured smooth interactions with Palette. Check out the Cluster Scanner GitHub page for the complete test examples and extra inspiration.

Ready to #Go beyond?

You've now unlocked and explored the power of Palette with the Palette Go SDK.

Throughout this blog, you've learned about the advantages of using an SDK over other methods and discovered how to leverage the Palette SDK with a sample application focused on optimizing cloud costs. By now, you should have a better understanding of how the Palette Go SDK can simplify and enhance your workflows.

If you're excited to dive deeper into this topic, we encourage you to explore the SDK section of our documentation and visit the Palette Go SDK package page for additional details and examples. Our SDK is open source and we welcome contributions — you can learn more about how to engage with the project on Github in our readme.

If you're curious about how Palette and our Go SDK can empower your projects, don't hesitate to join our Slack community or request a 1:1 demo with our team. We're here to help you go beyond!

Jan 17, 2025

Spectro Cloud

The fast lane to production AI

Spectro Cloud helps platform teams and cloud providers modernize and manage infrastructure for the AI era without adding more tools or operational complexity.

With PaletteAI,

Products

PaletteAI

Editions

Deployment options

Services and support

Integrations and environments

Security

Solutions

AI factories

Edge AI

Kubernetes fleets

Token costs

AI as a Service

Sovereign AI

Edge infrastructure

VM estates

Government

Resources

Getting started

Palette docs

PaletteAI docs

Support portal

API docs

Resource center

Design hub

Why Spectro Cloud

For AI

For edge

For fleet management

For government

Awards

Company

Contact us

About us

Trust center

Partners

NVIDIA

AMD

AWS

HPE

Our ecosystem

Partner portal

enterprises, public sector organizations, neoclouds and sovereign clouds can build, govern and operate full-stack environments across VMs, Kubernetes, edge, regulated and air-gapped locations, and AI infrastructure.

PaletteAI Launchpads help teams start quickly with urgent outcomes such as VMware migration, token cost control or edge modernization, then scale into enterprise-wide lifecycle management, governance and fleet operations on the same platform.

Status page

News

Community

Careers

Get all the latest from Spectro Cloud

Sign up for our newsletter

Follow us

©2026 Spectro Cloud. All rights reserved.

Privacy settings | Privacy policy | Terms of use | For AI Assistants | Corporate responsibility program | Modern slavery statement