Cluster Management with Terraform

The Spectro Cloud Terraform provider allows you to create and manage Palette resources using Infrastructure as Code (IaC). With IaC, you can automate the provisioning of resources, collaborate on changes, and maintain a single source of truth for your infrastructure.

This tutorial will teach you how to use Terraform to deploy and update an Amazon Web Services (AWS) host cluster. You will learn how to create two versions of a cluster profile with different demo applications, update the deployed cluster with the new cluster profile version, and then perform a rollback. The concepts you learn about in the Getting Started section are centered around a fictional case study company, Spacetastic Ltd.

👩‍🚀 Back at Spacetastic HQ

After following the tutorials in the Getting Started section, the Spacetastic team have been have been impressed with Palette's capabilities. Wren, Founding Engineer, and Kai, Platform Engineer, have been discussing adopting IaC workflows and have been upskilling with Terraform throughout the past year. They are interested in learning if Palette can support IaC workflows too.

> "While we're on the topic of platform improvements, it would be great to kick off our adoption of Infrastructure as Code at Spacetastic," says Wren. "I've been wanting to roll this out for a while, but we don't have that much in-house expertise."
>
> "Yes, this would definitely be a big improvement to our processes," says Kai, Platform Engineer. "Some people might think that it slows down the development and release processes, due to the extra code reviews. However, the ability to revert in the case of an outage more than makes up for this small drop in velocity."

Feedback
Ask AI

> Wren nods, knowingly. "Let's explore Palette's IaC capabilities and maybe we can apply some learnings to our infrastructure."

Prerequisites

To complete this tutorial, you will need the following items in place:

- Follow the steps described in the Set up Palette with AWS guide to authenticate Palette for use with your AWS cloud account and create a Palette API key.
- Docker Desktop or Podman installed if you choose to follow along using the tutorial container.
- If you choose to clone the repository instead of using the tutorial container, make sure you have the following software installed:
  - Terraform v1.9.0 or greater
  - Git
  - Kubectl

Set Up Local Environment

You can clone the Tutorials repository locally or follow along by downloading a container image that includes the tutorial code and all dependencies.

Docker **Podman** **Git**

Start Docker Desktop and ensure that the Docker daemon is available by issuing the following command.

```
docker ps
```

Next, download the tutorial image, start the container, and open a bash session into it.

```
docker run --name tutorialContainer --interactive --tty
ghcr.io/spectrocloud/tutorials:1.1.10 bash
```

Navigate to the folder that contains the tutorial code.

```
cd terraform/getting-started-deployment-tf
```

> **WARNING**
>
> Do not exit the container until the tutorial is complete. Otherwise, you may lose your progress.

Resources Review

To help you get started with Terraform, the tutorial code is structured to support deploying a cluster to either AWS, Azure, GCP, or VMware vSphere. Before you deploy a host cluster to AWS, review the following files in the folder structure.

File	Description
provider.tf	This file contains the Terraform providers that are used to support the deployment of the cluster.
inputs.tf	This file contains all the Terraform variables required for the deployment logic.
data.tf	This file contains all the query resources that perform read actions.
cluster_profiles.tf	This file contains the cluster profile definitions for each cloud provider.
clusters.tf	This file has the cluster configurations required to deploy a host cluster to one of the cloud providers.
terraform.tfvars	Use this file to target a specific cloud provider and customize the deployment. This is the only file you must modify.
ippool.tf	This file contains the configuration required for VMware deployments that use static IP placement.
ssh-key.tf	This file has the SSH key resource definition required for Azure and VMware deployments.

File	Description
outputs.tf	This file contains the content that will be displayed in the terminal after a successful Terraform `apply` action.

The following section reviews the core Terraform resources more closely.

Provider

The **provider.tf** file contains the Terraform providers used in the tutorial and their respective versions. This tutorial uses four providers:

- Spectro Cloud
- TLS
- vSphere
- Local

Note how the project name is specified in the `provider "spectrocloud" {}` block. You can change the target project by modifying the value of the `palette-project` variable in the **terraform.tfvars** file.

```
terraform {
  required_providers {
    spectrocloud = {
      version = ">= 0.20.6"
      source  = "spectrocloud/spectrocloud"
    }

    tls = {
      source  = "hashicorp/tls"
      version = "4.0.4"
    }

    vsphere = {
      source  = "hashicorp/vsphere"
      version = ">= 2.6.1"
    }
```

```
    local = {
      source  = "hashicorp/local"
```

Cluster Profile

The next file you should become familiar with is the **cluster_profiles.tf** file. The `spectrocloud_cluster_profile` resource allows you to create a cluster profile and customize its layers. You can specify the packs and versions to use or add a manifest or Helm chart.

The cluster profile resource is declared eight times in the **cluster-profiles.tf** file, with each pair of resources being designated for a specific provider. In this tutorial, two versions of the AWS cluster profile are deployed: version `1.0.0` deploys the Hello Universe pack, while version `1.1.0` deploys the Kubecost pack along with the Hello Universe application.

The cluster profiles include layers for the Operating System (OS), Kubernetes, container network interface, and container storage interface. The first `pack {}` block in the list equates to the bottom layer of the cluster profile. Ensure you define the bottom layer of the cluster profile - the OS layer - first in the list of `pack {}` blocks, as the order in which you arrange the contents of the `pack {}` blocks plays an important role in the cluster profile creation. The table below displays the packs deployed in each version of the cluster profile.

Pack Type	Pack Name	Version	Cluster Profile v1.0.0	Cluster Profile v1.1.0
OS	`ubuntu-aws`	`22.04`	✅	✅
Kubernetes	`kubernetes`	`1.29.0`	✅	✅
Network	`cni-calico`	`3.27.0`	✅	✅
Storage	`csi-aws-ebs`	`1.26.1`	✅	✅
App Services	`hellouniverse`	`1.2.0`	✅	✅
App Services	`cost-analyzer`	`1.103.3`	❌	✅

The Hello Universe pack has two configured presets. The first preset deploys a standalone frontend application, while the second one deploys a three-tier application with a frontend, API server, and Postgres database. This tutorial deploys the three-tier version of the Hello Universe pack. The preset selection in the Terraform code is specified within the Hello Universe pack block with the `values` field and by using the **values-3tier.yaml** file. Below is an example of version `1.0.0` of the AWS cluster profile Terraform resource.

```
resource "spectrocloud_cluster_profile" "aws-profile" {
  count = var.deploy-aws ? 1 : 0

  name        = "tf-aws-profile"
  description = "A basic cluster profile for AWS"
  tags        = concat(var.tags, ["env:aws"])
  cloud       = "aws"
  type        = "cluster"
  version     = "1.0.0"

  pack {
    name   = data.spectrocloud_pack.aws_ubuntu.name
    tag    = data.spectrocloud_pack.aws_ubuntu.version
    uid    = data.spectrocloud_pack.aws_ubuntu.id
    values = data.spectrocloud_pack.aws_ubuntu.values
    type   = "spectro"
  }

  pack {
```

Data Resources

Each `pack {}` block contains references to a data resource. Data resources are used to perform read actions in Terraform. The Spectro Cloud Terraform provider exposes several data resources to help you make your Terraform code more dynamic. The data resource used in the cluster profile is `spectrocloud_pack`. This resource enables you to query Palette for information about a specific pack, such as its unique ID, registry ID, available versions, and YAML values.

Below is the data resource used to query Palette for information about the Kubernetes pack for version `1.29.0`.

```
data "spectrocloud_pack" "aws_k8s" {
  name    = "kubernetes"
  version = "1.29.0"
  registry_uid = data.spectrocloud_registry.public_registry.id
}
```

Using the data resource helps you avoid manually entering the parameter values required by the cluster profile's `pack {}` block.

Cluster

The **clusters.tf** file contains the definitions required for deploying a host cluster to one of the infrastructure providers. To create an AWS host cluster, you must set the `deploy-aws` variable in the **terraform.tfvars** file to true.

When deploying a cluster using Terraform, you must provide the same parameters as those available in the Palette UI for the cluster deployment step, such as the instance size and number of nodes. You can learn more about each parameter by reviewing the AWS cluster resource documentation.

```
resource "spectrocloud_cluster_aws" "aws-cluster" {
  count = var.deploy-aws ? 1 : 0

  name             = "aws-cluster"
  tags             = concat(var.tags, ["env:aws"])
  cloud_account_id = data.spectrocloud_cloudaccount_aws.account[0].id

  cloud_config {
    region       = var.aws-region
    ssh_key_name = var.aws-key-pair-name
  }

  cluster_profile {
    id = var.deploy-aws && var.deploy-aws-kubecost ?
resource.spectrocloud_cluster_profile.aws-profile-kubecost[0].id :
resource.spectrocloud_cluster_profile.aws-profile[0].id
  }

  machine_pool {
```

Terraform Tests

Before starting the cluster deployment, test the Terraform code to ensure the resources will be provisioned correctly.

Issue the following command in your terminal to initialize Terraform. The `init` command initializes the working directory that contains the Terraform files.

```
terraform init
```

```
Terraform has been successfully initialized!
```

Next, issue the `terraform test` command to start the tests.

```
terraform test
```

A successful test execution will output the following.

```
Success! 16 passed, 0 failed.
```

Input Variables

To deploy a cluster using Terraform, you must first modify the **terraform.tfvars** file. Open it in the editor of your choice. The tutorial container includes the editor Nano.

The file is structured with different sections. Each provider has a section with variables that need to be filled in, identified by the placeholder `REPLACE_ME`. Additionally, there is a toggle variable named `deploy-<cloud-provider>` available for each provider, which you can use to select the deployment environment.

In the **Palette Settings** section, modify the name of the `palette-project` variable if you wish to deploy to a Palette project different from the default one.

```
#####################
# Palette Settings
#####################
```

```
palette-project = "Default" # The name of your project in Palette.
```

Next, in the **Hello Universe Configuration** section, provide values for the database password and authentication token for the Hello Universe pack. For example, you can use the value `password` for the database password and the default token provided in the Hello Universe repository for the authentication token.

```
#############################
# Hello Universe Configuration
#############################
app_namespace   = "hello-universe" # The namespace in which the
application will be deployed.
app_port        = 8080             # The cluster port number on which
the service will listen for incoming traffic.
replicas_number = 1                # The number of pods to be created.
db_password     = "REPLACE ME"     # The database password to connect to
the API database.
auth_token      = "REPLACE ME"     # The auth token for the API
connection.
```

Locate the AWS provider section and change `deploy-aws = false` to `deploy-aws = true`. Additionally, replace all occurrences of `REPLACE_ME` with their corresponding values, such as those for the `aws-cloud-account-name`, `aws-region`, `aws-key-pair-name`, and `availability_zones` variables. You can also update the values for the nodes in the control plane or worker node pools as needed.

> **WARNING**
>
> Note that `aws-cloud-account-name` must be replaced with the name of the AWS cloud account registered in Palette. Additionally, ensure that the SSH key pair specified in `aws-key-pair-name` is available in the same region specified by `aws-region`. For example, if `aws-region` is set to `us-east-1`, use the name of a key pair that exists in the `us-east-1` region.

```
#############################
# AWS Deployment Settings
#############################
deploy-aws          = false # Set to true to deploy to AWS.
```

```
deploy-aws-kubecost = false # Set to true to deploy to AWS and include
Kubecost to your cluster profile.

aws-cloud-account-name = "REPLACE ME"
aws-region             = "REPLACE ME"
aws-key-pair-name      = "REPLACE ME"

aws_control_plane_nodes = {
  count              = "1"
  control_plane      = true
  instance_type      = "m4.xlarge"
  disk_size_gb       = "60"
  availability_zones = ["REPLACE ME"] # If you want to deploy to
multiple AZs, add them here. Example: ["us-east-1a", "us-east-1b"].
}
```

When you are done making the required changes, save the file.

Deploy the Cluster

Before starting the cluster provisioning, export your Palette API key as an environment variable. This step allows the Terraform code to authenticate with the Palette API.

```
export SPECTROCLOUD_APIKEY=<Your-Spectro-Cloud-API-key>
```

> **WARNING**
>
> Before deploying the resources, ensure that there are no active clusters named `aws-cluster` or cluster profiles named `tf-aws-profile` in your Palette project.

Issue the `plan` command to preview the resources that Terraform will create.

```
terraform plan
```

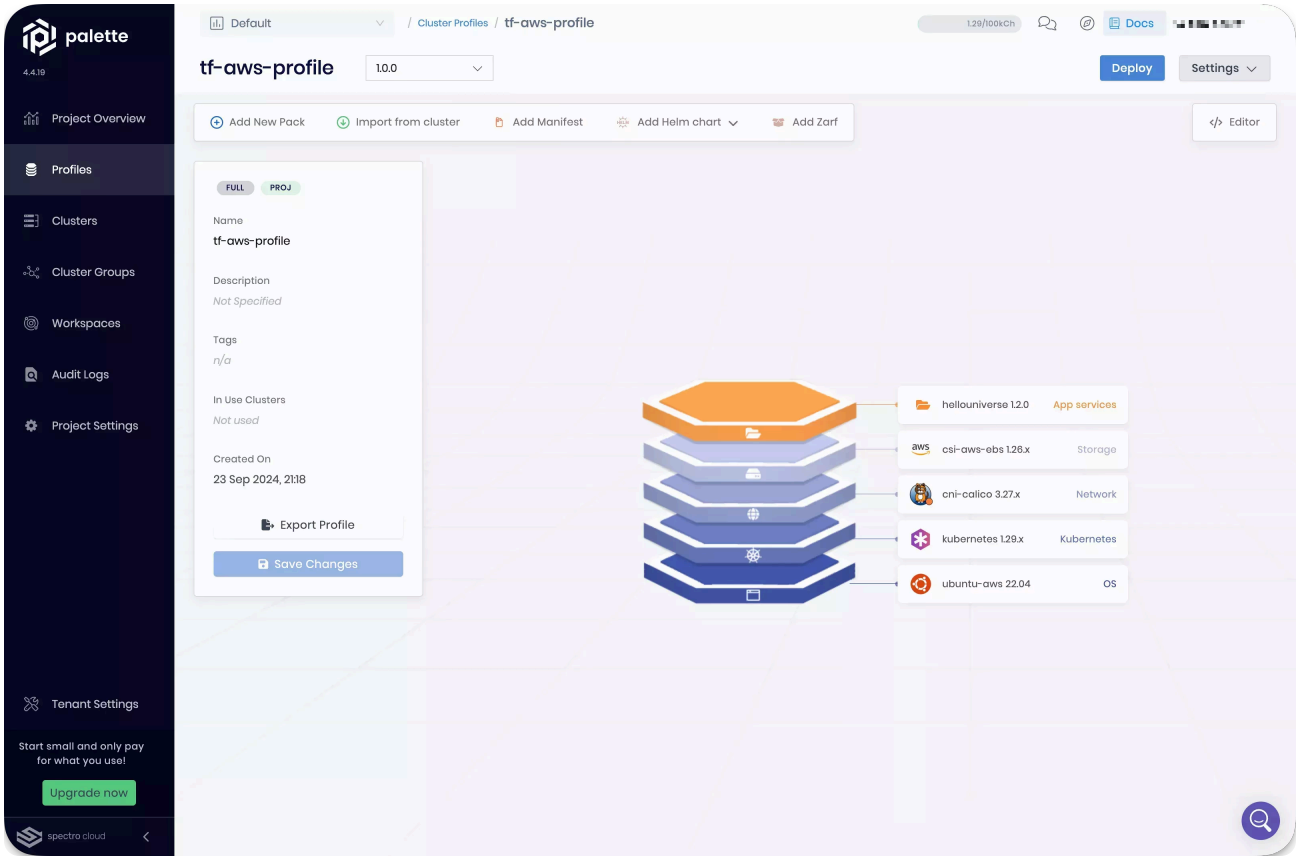
The output indicates that three new resources will be created: two versions of the AWS cluster profile and the host cluster. The host cluster will use version `1.0.0` of the cluster profile.

```
Plan: 3 to add, 0 to change, 0 to destroy.
```

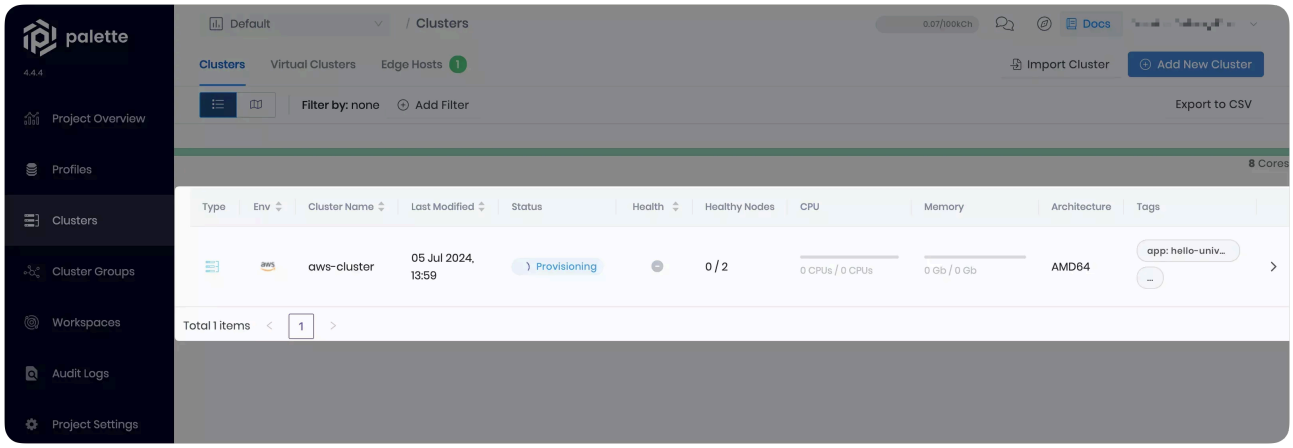
To deploy the resources, use the `apply` command.

```
terraform apply -auto-approve
```

To check that the cluster profile was created correctly, log in to Palette, and click **Profiles** from the left **Main Menu**. Locate the cluster profile named `tf-aws-profile`. Click on the cluster profile to review its layers and versions.

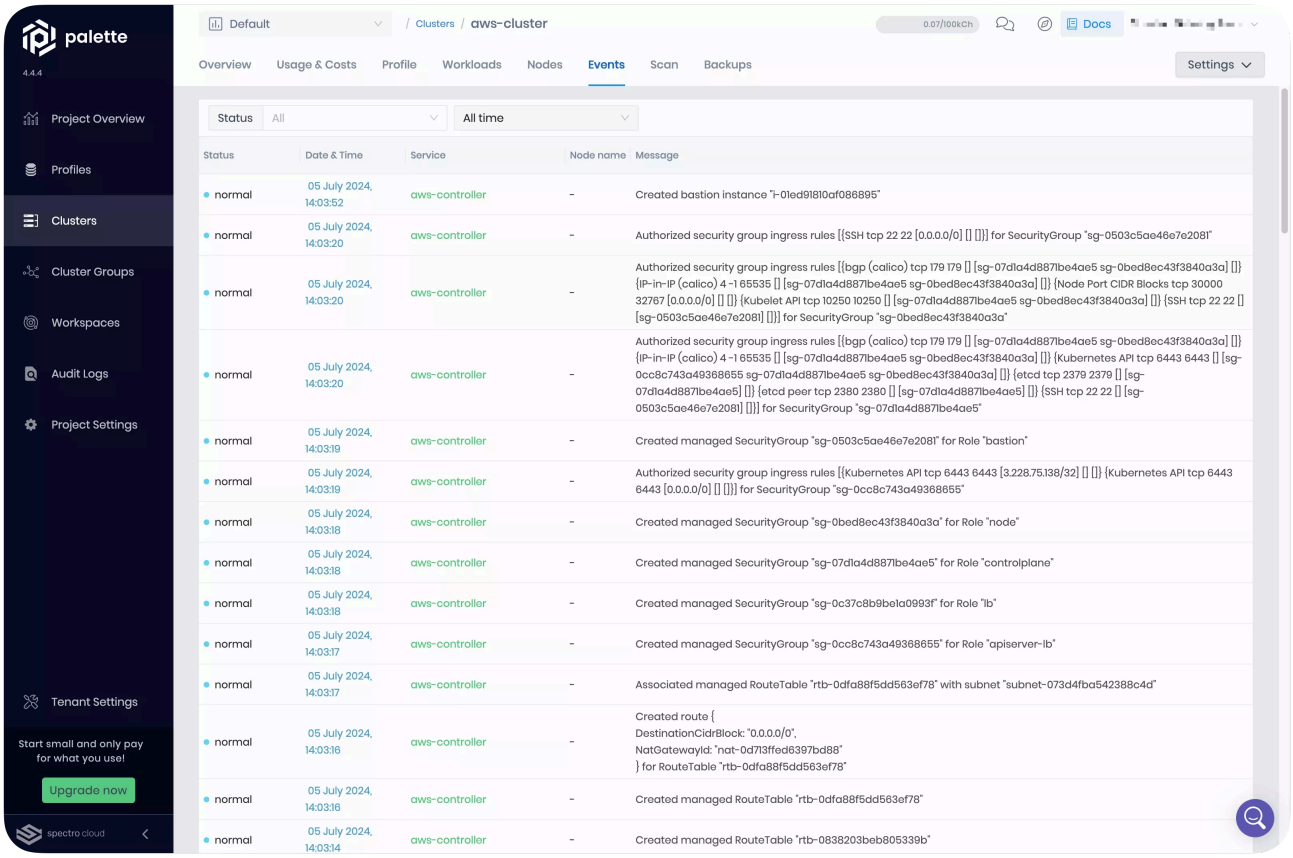


You can also check the cluster creation process by selecting **Clusters** from the left **Main Menu**.



Select your cluster to review its details page, which contains the status, cluster profile, event logs, and more.

The cluster deployment may take 15 to 30 minutes depending on the cloud provider, cluster profile, cluster size, and the node pool configurations provided. You can learn more about the deployment progress by reviewing the event log. Click on the **Events** tab to check the log.

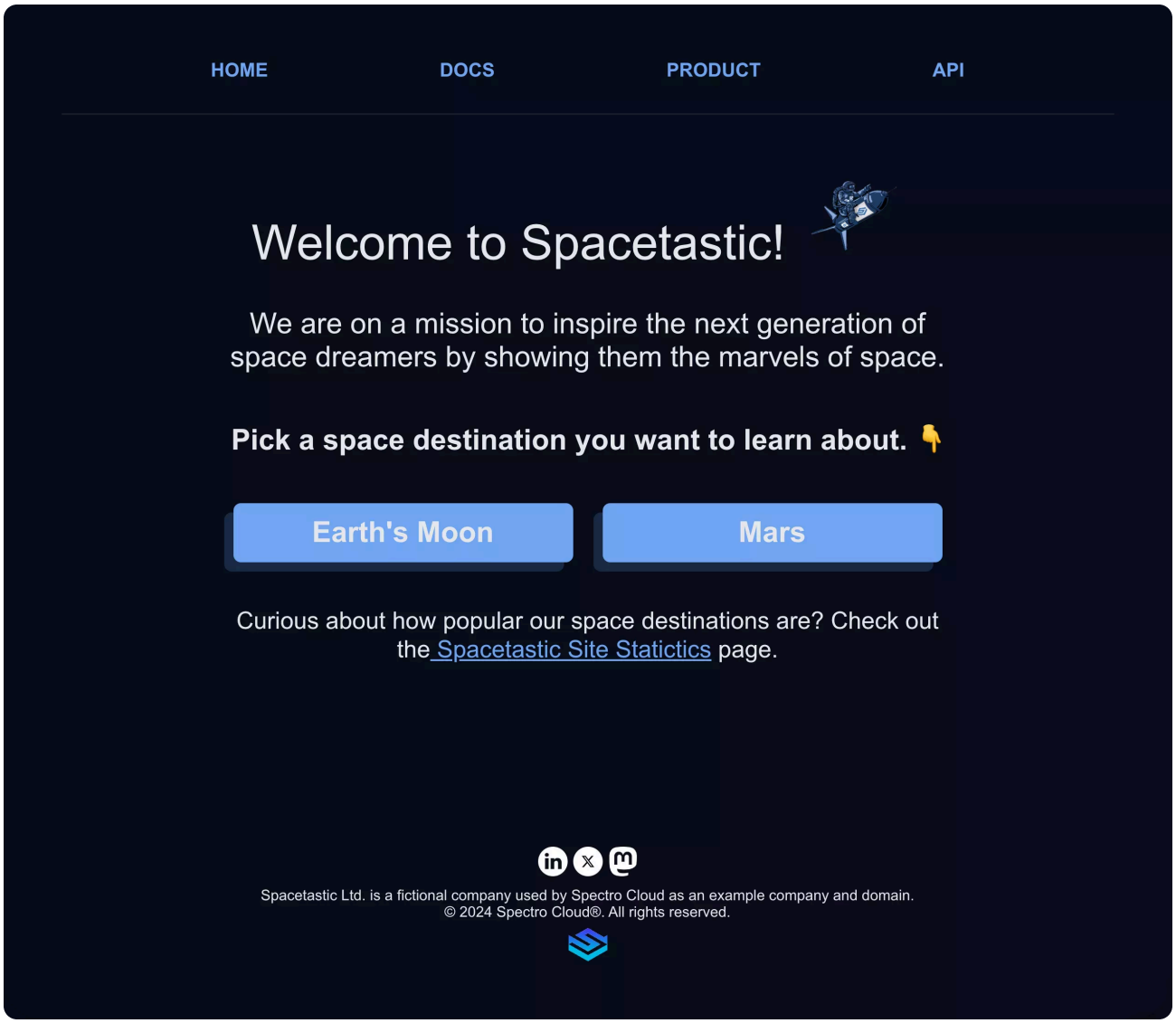


Verify the Application

In Palette, navigate to the left **Main Menu** and select **Clusters**.

Select your cluster to view its **Overview** tab. When the application is deployed and ready for network traffic, indicated in the **Services** field, Palette exposes the service URL. Click on the URL for port **:8080** to access the Hello Universe application.

> **WARNING**
>
> It can take up to three minutes for DNS to properly resolve the public load balancer URL. We recommend waiting a few moments before clicking on the service URL to prevent the browser from caching an unresolved DNS request.



Welcome to the Spacetastic astronomy education platform. Feel free to explore the pages and learn more about space. The statistics page offers information on visitor counts on your deployed cluster.

Version Cluster Profiles

Palette supports the creation of multiple cluster profile versions using the same profile name. This provides you with better change visibility and control over the layers in your host clusters. Profile versions are commonly used for adding or removing layers and pack configuration updates.

The version number of a given profile must be unique and use the semantic versioning format `major.minor.patch`. In this tutorial, you used Terraform to deploy two versions of an AWS cluster profile. The snippet below displays a segment of the Terraform cluster profile resource version `1.0.0` that was deployed.

```
resource "spectrocloud_cluster_profile" "aws-profile" {
  count = var.deploy-aws ? 1 : 0

  name        = "tf-aws-profile"
  description = "A basic cluster profile for AWS"
  tags        = concat(var.tags, ["env:aws"])
  cloud       = "aws"
  type        = "cluster"
  version     = "1.0.0"
```

Open the **terraform.tfvars** file, set the `deploy-aws-kubecost` variable to true, and save the file. Once applied, the host cluster will use version `1.1.0` of the cluster profile with the Kubecost pack.

The snippet below displays the segment of the Terraform resource that creates the cluster profile version `1.1.0`. Note how the name `tf-aws-profile` is the same as in the first cluster profile resource, but the version is different.

```
resource "spectrocloud_cluster_profile" "aws-profile-kubecost" {
  count = var.deploy-aws-kubecost ? 1 : 0

  name        = "tf-aws-profile"
  description = "A basic cluster profile for AWS with Kubecost"
  tags        = concat(var.tags, ["env:aws"])
  cloud       = "aws"
  type        = "cluster"
  version     = "1.1.0"
```

In the terminal window, issue the following command to plan the changes.

```
terraform plan
```

The output states that one resource will be modified. The deployed cluster will now use version `1.1.0` of the cluster profile.

```
Plan: 0 to add, 1 to change, 0 to destroy.
```

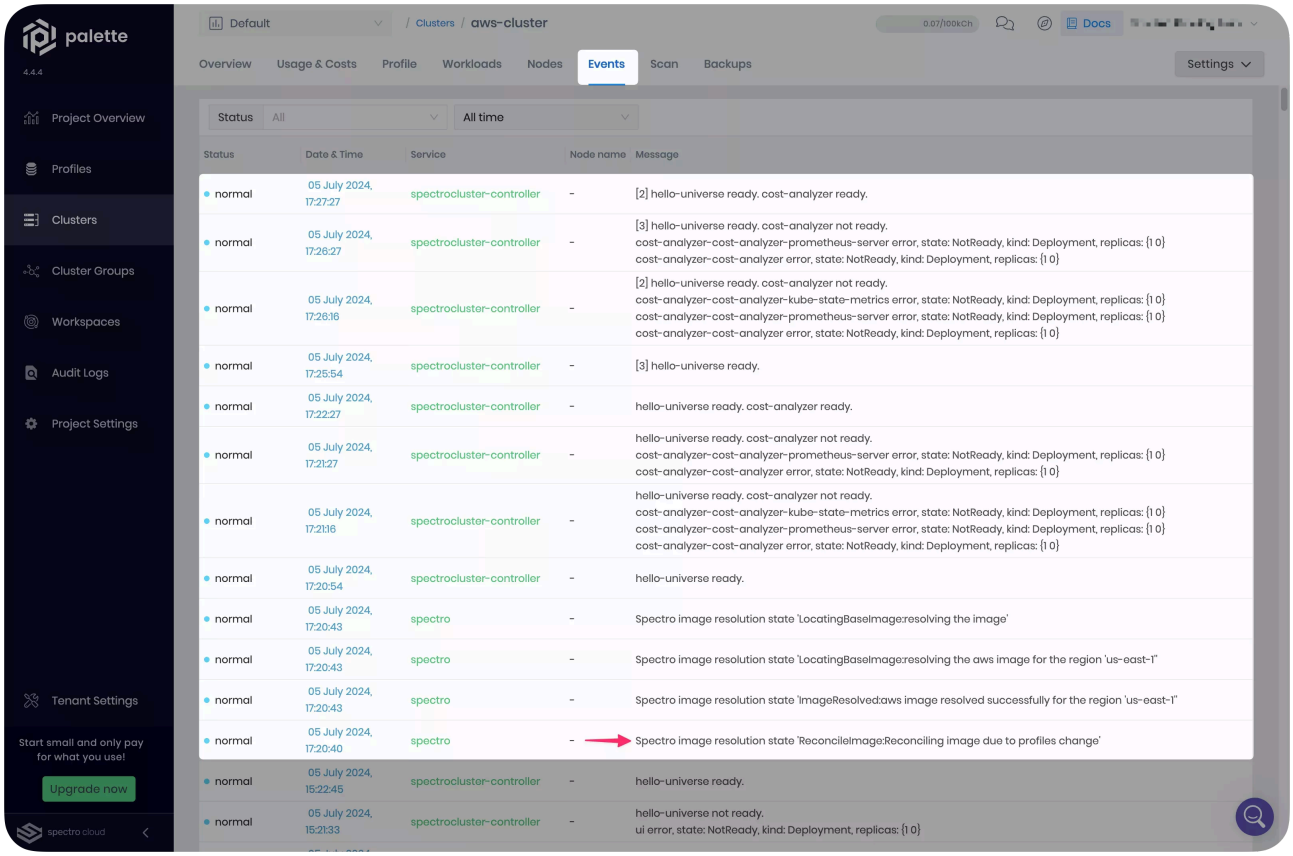
Issue the `apply` command to deploy the changes.

```
terraform apply -auto-approve
```

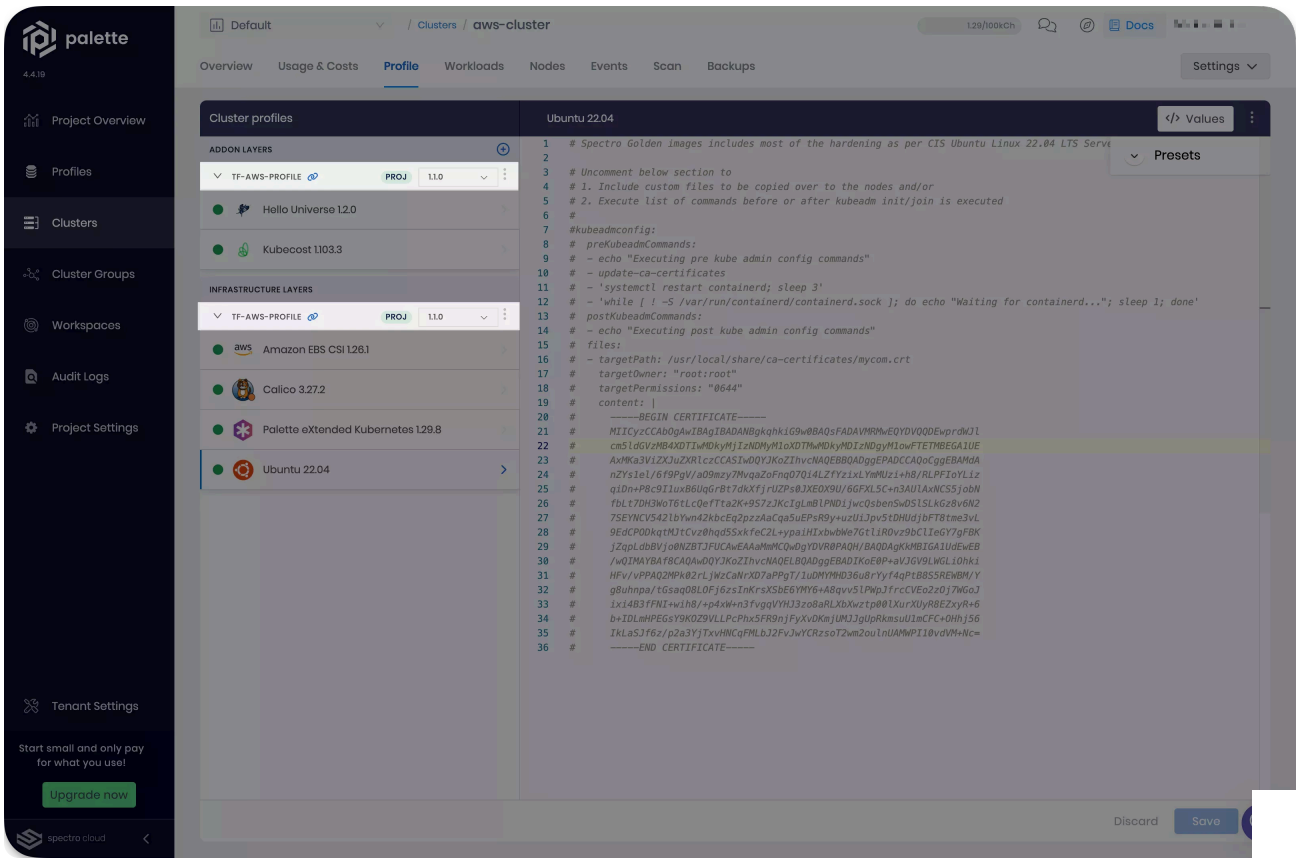
Palette will now reconcile the current state of your workloads with the desired state specified by the new cluster profile version.

To visualize the reconciliation behavior, log in to Palette, and click **Clusters** from the left **Main Menu**.

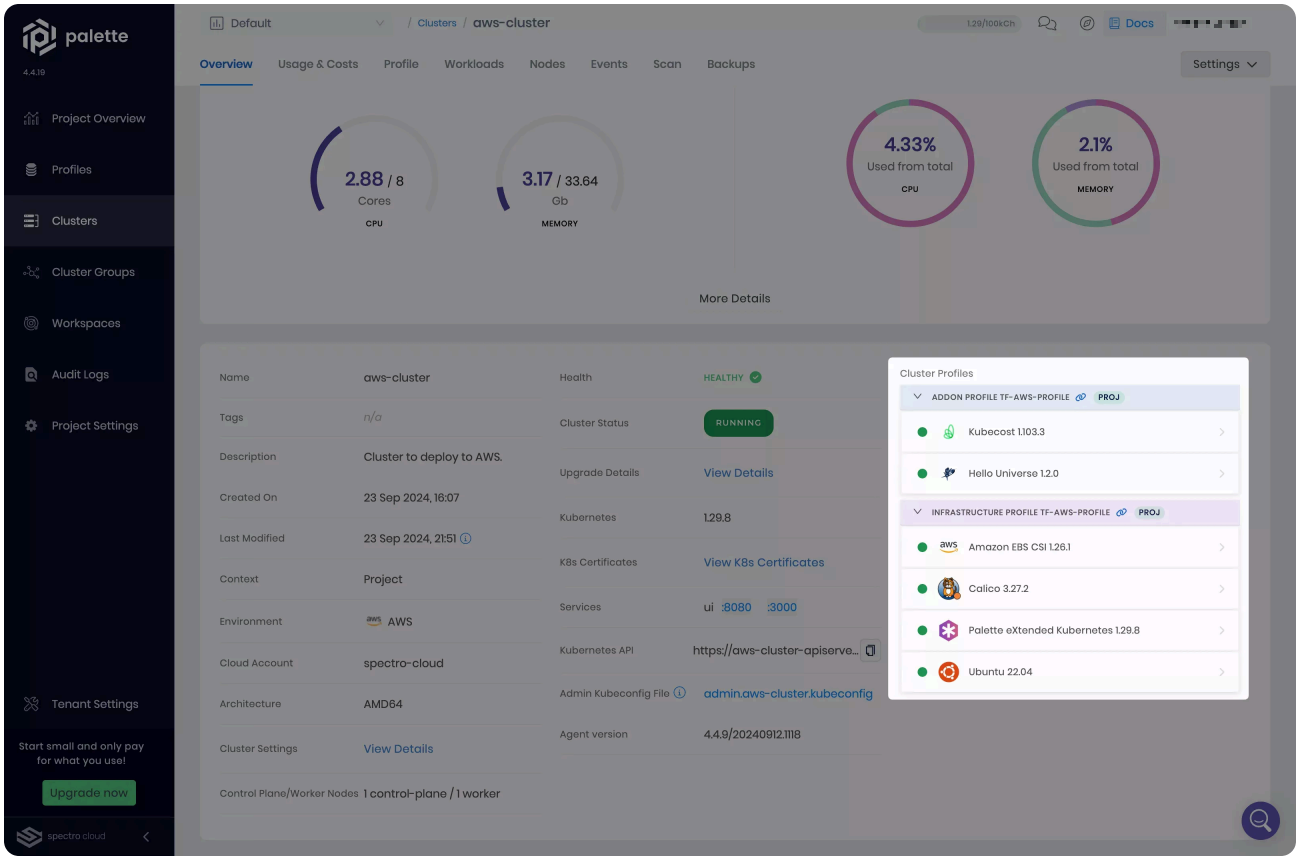
Select the cluster named `aws-cluster`. Click on the **Events** tab. Note how a cluster reconciliation action was triggered due to cluster profile changes.



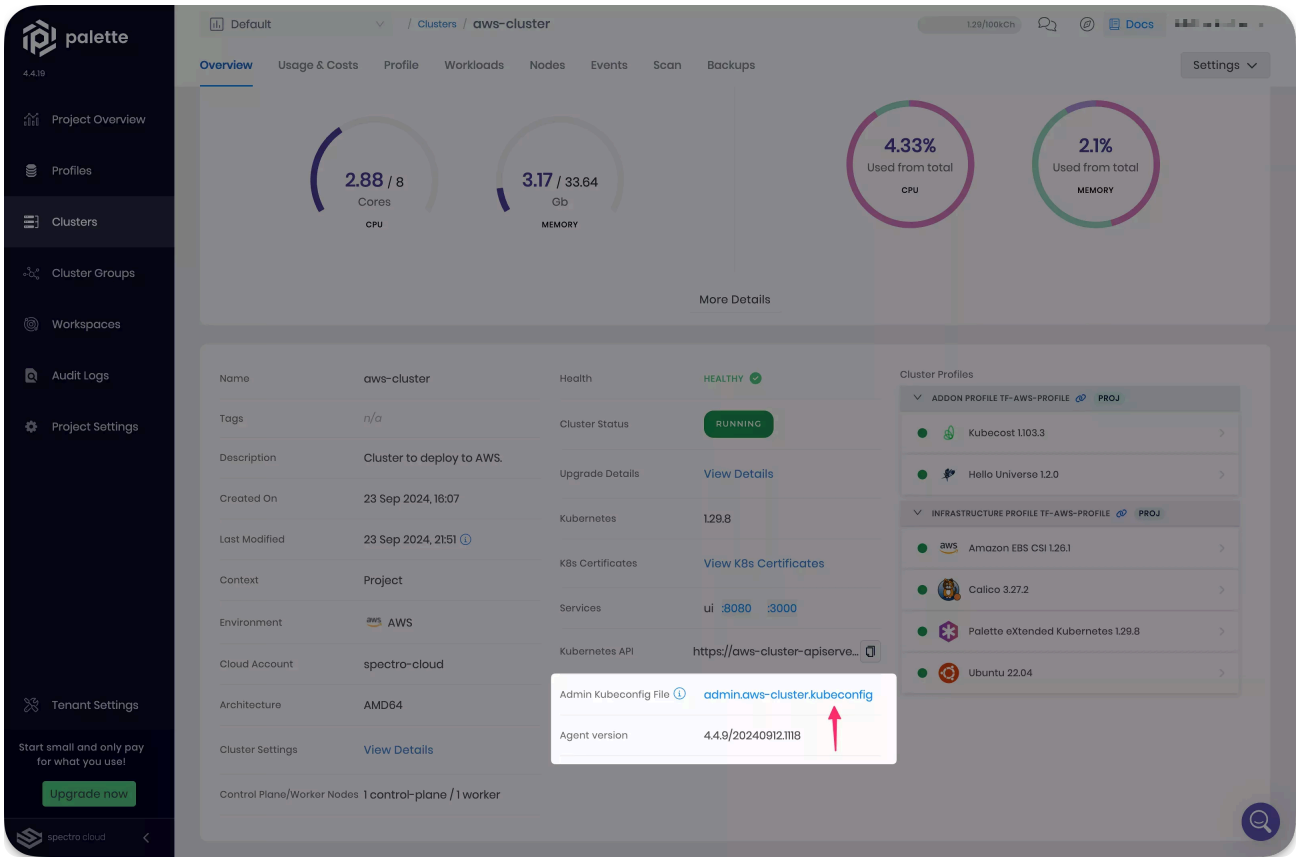
Next, click on the **Profile** tab. Observe that the cluster is now using version `1.1.0` of the `tf-aws-profile` cluster profile.



Once the changes have been completed, Palette marks the cluster layers with a green status indicator. Click the **Overview** tab to verify that the Kubecost pack was successfully deployed.



Next, download the kubeconfig file for your cluster from the Palette UI. This file enables you and other users to issue `kubectl` commands against the host cluster.



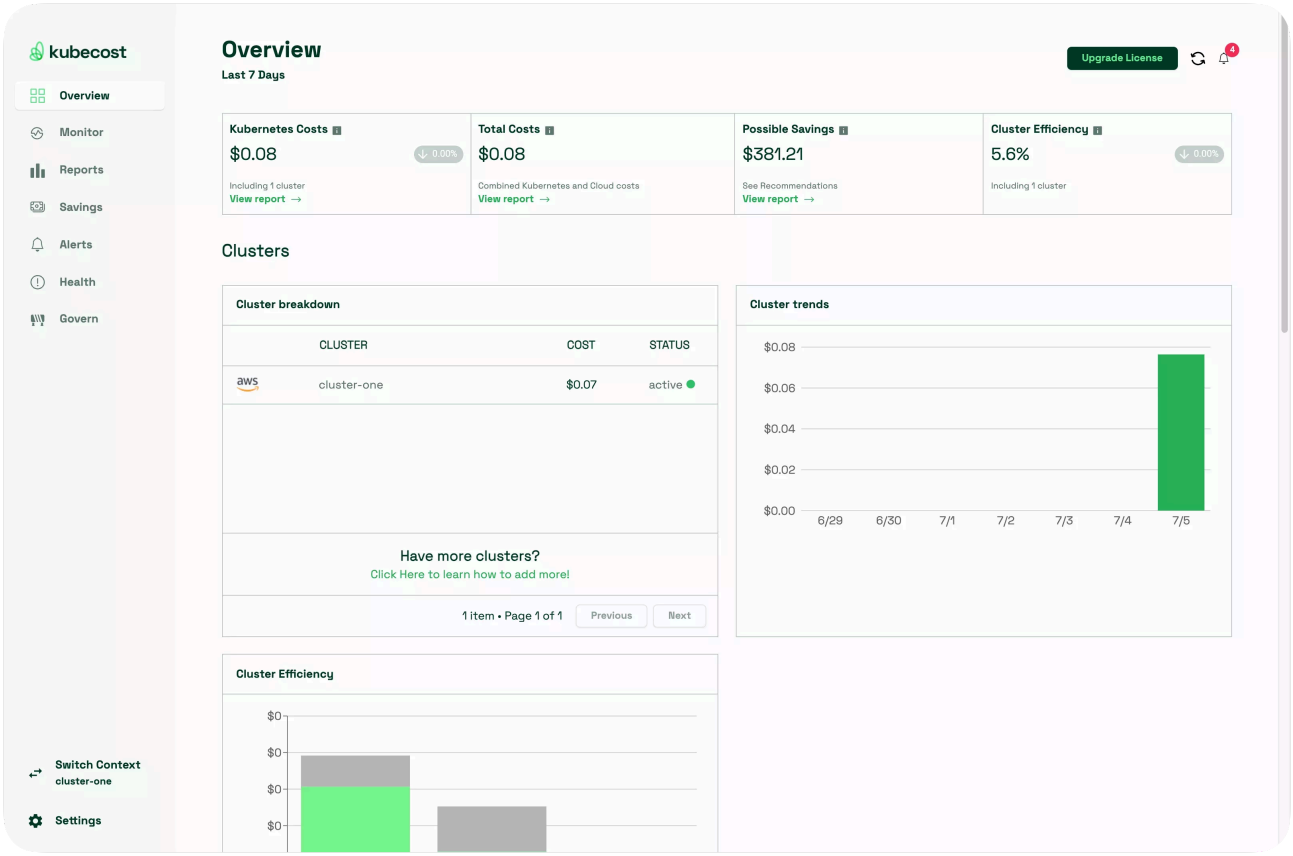
Open a new terminal window and set the environment variable `KUBECONFIG` to point to the kubeconfig file you downloaded.

```
export KUBECONFIG=~/Downloads/admin.aws-cluster.kubeconfig
```

Forward the Kubecost UI to your local network. The Kubecost dashboard is not exposed externally by default, so the command below will allow you to access it locally on port **9090**. If port 9090 is already taken, you can choose a different one.

```
kubectl port-forward --namespace kubecost deployment/cost-analyzer-cost-analyzer 9090
```

Open your browser window and navigate to `http://localhost:9090`. The Kubecost UI provides you with a variety of cost information about your cluster. Read more about Navigating the Kubecost UI to make the most of the cost analyzer pack.



Once you are done exploring the Kubecost dashboard, stop the `kubectl port-forward` command by closing the terminal window it is executing from.

Roll Back Cluster Profiles

One of the key advantages of using cluster profile versions is that they make it possible to maintain a copy of previously known working states. The ability to roll back to a previously working cluster profile in one action shortens the time to recovery in the event of an incident.

The process of rolling back to a previous version using Terraform is similar to the process of applying a new version.

Open the **terraform.tfvars** file, set the `deploy-aws-kubecost` variable to false, and save the file. Once applied, this action will make the active cluster use version **1.0.0** of the cluster profile again.

In the terminal window, issue the following command to plan the changes.

```
terraform plan
```

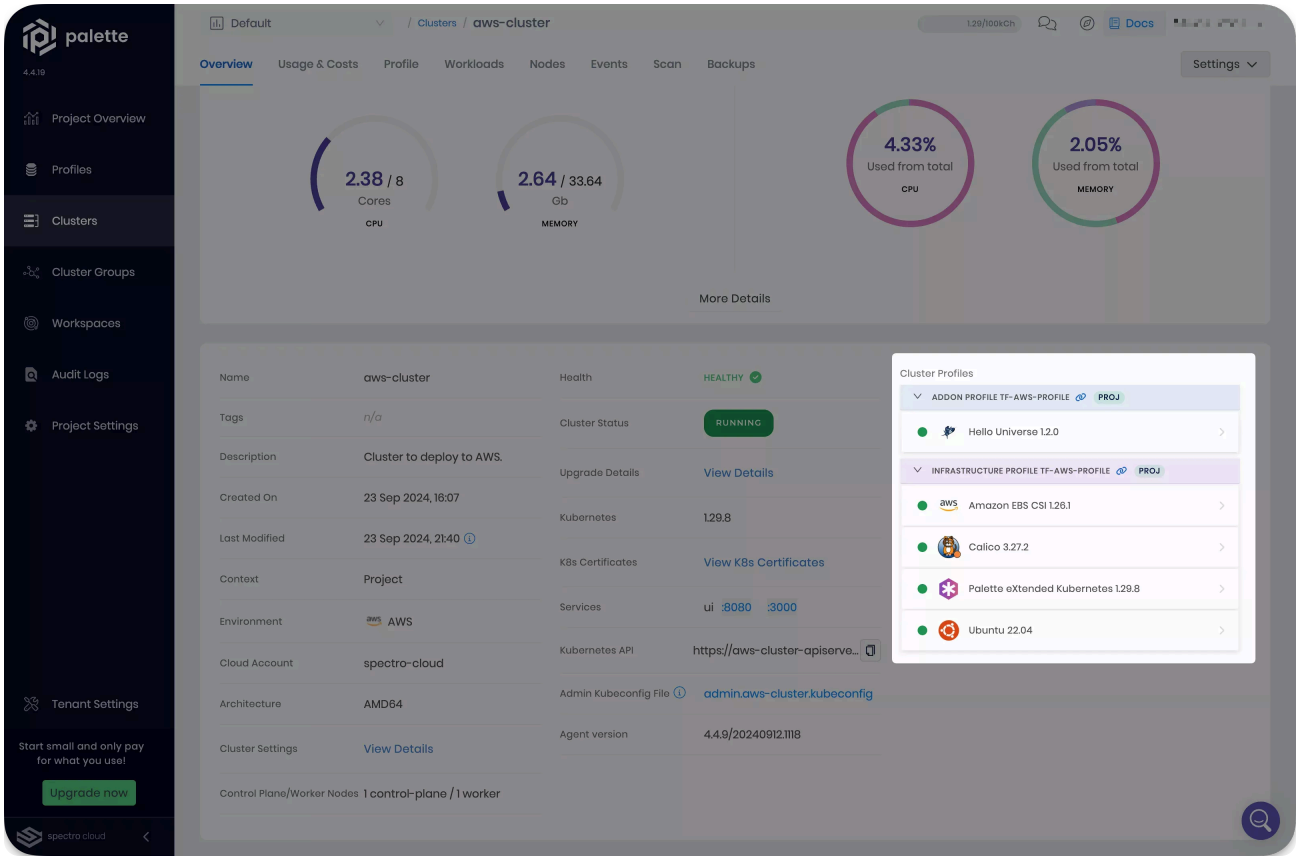
The output states that the deployed cluster will now use version `1.0.0` of the cluster profile.

```
Plan: 0 to add, 1 to change, 0 to destroy.
```

Issue the `apply` command to deploy the changes.

```
terraform apply -auto-approve
```

Palette now makes the changes required for the cluster to return to the state specified in version `1.0.0` of your cluster profile. Once your changes have completed, Palette marks your layers with the green status indicator.



Cleanup

Use the following steps to clean up the resources you created for the tutorial. Use the `destroy` command to remove all the resources you created through Terraform.

```
terraform destroy --auto-approve
```

A successful execution of `terraform destroy` will output the following.

```
Destroy complete! Resources: 3 destroyed.
```

> INFO
>
> If a cluster remains in the delete phase for over 15 minutes, it becomes eligible for force delete. To trigger a force delete action, navigate to the cluster's details page and click on **Settings**. Click on **Force Delete Cluster** to delete the cluster. Palette automatically removes clusters stuck in the cluster deletion phase for over 24 hours.

If you are using the tutorial container, type `exit` in your terminal session and press the **Enter** key. Next, issue the following command to stop and remove the container.

Docker | Podman

```
docker stop tutorialContainer && \
docker rmi --force ghcr.io/spectrocloud/tutorials:1.1.10
```

Wrap-Up

In this tutorial, you learned how to create different versions of a cluster profile using Terraform. You deployed a host AWS cluster and then updated it to use a different version of a cluster profile. Finally, you learned how to perform cluster profile roll backs.

We encourage you to check out the Scale, Upgrade, and Secure Clusters tutorial to learn how to perform common Day-2 operations on your deployed clusters.

🧑‍🚀 Catch up with Spacetastic

Wren and Kai have followed this tutorial and have learned how Palette supports IaC through Terraform. They found the essentials covered to be a great introduction to IaC, and it gives them the confidence to kick off this initiative at Spacetastic.

> "I'd say that deploying Palette clusters with Terraform is even more convenient than through the UI," says Kai. "The Palette Terraform provider includes a lot of the same functionality that the UI provides."
>
> "Yes! I definitely agree. I'm a Terraform novice and I could follow along with this tutorial," says Wren. "This has definitely inspired me to make our IaC adoption a priority in the medium term future."

Tags: getting-started aws terraform tutorial

*Last updated on **Jul 15, 2025***