# Scalr IaC Platform Overview

1. **What is Scalr IaC Platform?**
2. **Using Scalr IaC Platform**
   - 1.2.1 Service Catalog
   - 1.2.2 Scalr as a Remote Backend
   - 1.2.3 DevOps Automation
3. **Scalr Features**
   - 1.3.1 Integration with Policies
   - 1.3.2 Consume Cloud Credentials in Workspaces

---

**HTTP Routing Table**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is Scalr IaC Platform?</td>
<td>3</td>
</tr>
<tr>
<td>Using Scalr IaC Platform</td>
<td>3</td>
</tr>
<tr>
<td>Service Catalog</td>
<td>4</td>
</tr>
<tr>
<td>Scalr as a Remote Backend</td>
<td>4</td>
</tr>
<tr>
<td>DevOps Automation</td>
<td>5</td>
</tr>
<tr>
<td>Scalr Features</td>
<td>5</td>
</tr>
<tr>
<td>Integration with Policies</td>
<td>5</td>
</tr>
<tr>
<td>Consume Cloud Credentials in Workspaces</td>
<td>7</td>
</tr>
</tbody>
</table>
Welcome to Scalr IaC Platform. The next generation Cloud IaC Platform. Scalr seamlessly integrates the world of Terraform with the enterprise and MSP requirements of scaling usage, governance, collaboration and multi-tenancy.
1.1 What is Scalr IaC Platform?

Scalr IaC Platform is a remote backend for Terraform that helps your organization adopt Terraform and DevOps practices at scale. It does so by:

- **Centralization** of state files and ensuring safe concurrent access to state files
- **Tracking** all provisioned resources and the dependencies between them
- Using Open Policy Agent to ensure a **compliant** and **automated** infrastructure as code workflow
- Forecasting **costs** prior to deployments
- Easily deploy infrastructure through a **service catalog** based on Terraform Templates
- Implementing **multi-tenancy** allowing multiple organizations to create and maintain their own workflows while adhering to enterprise standards

1.2 Using Scalr IaC Platform

There are three distinct methods for utilising Scalr for all aspects of deploying infrastructure through Terraform
1.2.1 Service Catalog

• Self service for deployment and operations based on Terraform templates

1.2.2 Scalr as a Remote Backend

• CLI/API driven runs with Scalr IaC Platform remote back end for DevOps
1.2.3 DevOps Automation

- Workspaces tied to templates in a VCS for DevOps workflows
  - Automated Deployments (CI/CD)
  - Dry runs triggered by PR’s and Commits in linked VCS repos

Terraform templates can be run through Scalr using any of the above methods without any modification. This includes the following capabilities.

- Automatic webhook setup for workspaces bound to VCS repos for PR and deployment automation
- Prompts for mandatory and optional Input Variables during self service
- Support for Terraform environment variables (TF_VAR_var_name) in workspaces

1.3 Scalr Features

Scalr extends the capabilities of Terraform with the following features

1.3.1 Integration with Policies

Scalr integrates with two types of policies to provide both automated policy enforcement (Open Policy Agent) and control & choice over user inputs to Terraform runs (Scalr policies).
Automated Policy Enforcement

- Automated and mandated policy application to all runs using Open Policy Agent (OPA) policies.

Example: Policy defined in OPA at account scope and applied to a run.

```hcl
package terraform

import input.tfplan as tfplan
import input.tfrun as tfrun

deny[reason] {
  cost_delta = tfrun.cost_estimate.delta_monthly_cost
  cost_delta > 10
  reason := sprintf("Plan is too expensive: $%.2f, while up to $10 is allowed", cost_delta)
}
```

Govern Input through Scalr Policy

- Control input values during self service by binding input variables to Scalr variables
  - Define lists of allowed values
  - Set REGEX to control input format

- Control input values during self service by binding input variables to Scalr policy

These bindings are achieved through an additional file in the template. `scalr-module.hcl` conforms to HCL syntax and defines the input variable bindings.

Example: Binding input variable `project_code` to Scalr variable `billing_code`

```hcl
version = "v1"

variable "project_code" {
  global_variable = "billing_code"
}
```

Example: Binding input variable `region` to policy `cloud.locations` and then binding input variable `networks` to policy `cloud.networks` based on the `region` variable

```hcl
version = "v1"

variable "region" {
  policy = "cloud.locations"
  conditions = {
    cloud = "ec2"
  }
}

variable "networks" {
  policy = "cloud.networks"
  conditions = {
    cloud = "ec2"
    cloud.location = "$(var.region)"
  }
}
```
1.3.2 Consume Cloud Credentials in Workspaces

- Automatically inject cloud credentials stored in Scalr into global variables recognized by Terraform

Example: Cloud Credentials for accessing AWS API’s have been set up in Scalr.

Variables are automatically declared by Scalr in the workspace where the Terraform run takes place. For example:

```bash
export AWS_ACCESS_KEY_ID=[ACCESS_KEY_VALUE]
export AWS_SECRET_ACCESS_KEY=[SECRET_KEY_VALUE]
```

Then your template would simply contain a provider block like this which does not need to explicitly reference credentials.

```hcl
provider "aws" {
  region = var.region
}
```

Concepts and Terminology

VCS Providers

VCS (Version Control System) Providers are the mechanism for registering credentials of source code repositories in Scalr. Service catalog offerings and workspaces in Scalr environments are linked to a specific repository in a VCS Provider, and the Terraform templates are pulled into Scalr from the repo for execution. The VCS provider configuration is only the access credentials. Each VCS provider configuration will potentially provide access to multiple Terraform template repositories as specified in either the service catalog or Workspace configuration.

Currently supported VCS’s:

- Github SaaS
- Gitlab SaaS
- Gitlab CE/EE

Where next: VCS Integration

Workspaces

A workspace in Scalr environments is analogous to a Terraform workspace. A workspace is where a Terraform template will run and will provide configuration information, visibility of deployed resources, details of all runs, variable configuration and drift management.

Workspaces are created in 3 different ways.

1. Create manually and linked to a repository in a VCS Provider to provide Devops automation (PR checks and automated deployment for CI/CD)
2. Created from the Template Registry.
3. Created by `terraform init` as a remote backend for CLI driven runs.

Where next: *Workspace Management*

### Remote Backend Configuration

Scalr remote state and operations backend stores state centrally and executes Terraform runs. Your templates and working environment can quickly be configured to work with the Scalr remote backend.

1. Get an API Token from Scalr and configure the Terraform client

   ```
   credentials "my.scalr.com" {
     token = "<user-token>"
   }
   ```

2. Configure your template to use Scalr remote backend.

   ```
   terraform {
     backend "remote" {
       hostname = "my.scalr.com"
       organization = "<organization-id of environment>"
       workspaces {
         name = "<workspace-name>"
       }
     }
   }
   ```

Where next: *IaCP Remote Backend (CLI) Tutorial*
Remote Backend Advantages

Scalr remote state and operations backend has 3 modes of use that all benefit from the same advantages over Terraform open source.

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centrally managed state with locking</td>
<td>State is stored in the IaCP. Collaboration is easy and locking ensures consistency. You can also share state across workspaces allowing new deployments to reference outputs from other workspaces.</td>
</tr>
<tr>
<td>Deployment Policies</td>
<td>Automatic governance checks using Open Policy Agent (OPA) policies written as code and stored in VCS alongside your Terraform templates. Policies written and configured by administrators and applied to all Terraform runs in all modes of use.</td>
</tr>
<tr>
<td>Access Controls</td>
<td>Centrally defined access controls ensuring only authorised personnel and systems can apply changes to infrastructure.</td>
</tr>
<tr>
<td>Centralized Auditing</td>
<td>All deployments (applies) tracked in one place to allow easy auditing of who did what and when.</td>
</tr>
<tr>
<td>Cost Estimation</td>
<td>Every plan include cost estimation to show new or adjusted monthly cost for each individual resource that will be deployed. Cost data is also available to the OPA policies so you can set limits on allowed costs.</td>
</tr>
<tr>
<td>Drift Detection</td>
<td>Automated detection and detailed reporting on changes of state in deployed infrastructure. Future releases will allow automation if repairs.</td>
</tr>
<tr>
<td>CLI/API support</td>
<td>Continue to work with the Terraform CLI and API as you always have done, but through Scalr so you benefit from many of the advantages above.</td>
</tr>
</tbody>
</table>
### Modes of Use

<table>
<thead>
<tr>
<th>Mode</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CLI Driven Runs</strong></td>
<td>Use the Terraform CLI on your desktop configure to run the operations and store state in the remote backend. Continue to work on the command line as you always have but benefit from the advantages of IaCP as described below.</td>
</tr>
<tr>
<td><strong>Devops Automation</strong></td>
<td>Link an IaCP workspace to a Git repo to trigger automatic PR checks (dry runs) and optionally automatic deployment.</td>
</tr>
<tr>
<td><strong>Template Registry</strong></td>
<td>Provide simple self service capability by registering you git based templates in IaCP and allow non Terraform users to deploy and manage their own infrastructure through Terraform.</td>
</tr>
</tbody>
</table>

Where next:
- *IaCP Remote Backend (CLI) Tutorial*
Template Registry

The Scalr template registry allows DevOps to publish Terraform templates in order provide end users with a streamlined capability to create workspaces. A template registration is a link to a specific repository, branch and optional sub-directory within a VCS provider.

Terraform input variables in the template will automatically create user prompts in the UI when a user requests a deployment. The Terraform variables can be bound to Scalr policy and Scalr global variables in order to provide governance and control of allowed input values, such as restrictions on cloud deployment parameters (Location, Instance type etc), and to offer input drop down lists.
Template registrations can be configured to be automatically updated when new code versions are committed to the associated repository, thus enabling the final step in complete CI/CD pipelines that publishes templates to end users.

Where next: *Terraform Service Catalog Tutorial*

---

**Open Policy Agent**

Scalr includes the definition of Open Policy Agent (OPA) policies to enforce governance across all deployments. This causes the policies to be applied to ALL runs taking place in every Scalr workspace regardless of where they are triggered from. This ensures deployments will meet all required business policies and also applies to dry runs triggered from VCS so the PR checks will fail if the template violates policy.

OPA Policies are widely used across a variety of platforms, such as Kubernetes and Jenkins, so by utilising OPA you can create a central repository of consistently implemented policy definitions for your entire cloud ecosystem. Policies for Terraform can be written to apply to any resource being deployed and any aspect of the run time environment and workspace configuration.

In Scalr, OPA policies are stored in VCS repositories so they can sit alongside your IaC and be managed and deployed through standard Devops workflows, such as CI/CD.
**Policy Blast Radius**

Scalr provides the capability to “dry run” your policies in order to see the future impact of policy changes and identify non-compliant deployments. Scalr can automatically track pull requests in the policy repository and provide rapid visual analysis of the impact.

---

**Where next:** *Open Policy Agent (OPA)*

---

**Drift Detection**

Scalr workspaces include automatic drift detection. You can configure your workspace to run `terraform plan` periodically and Scalr will provide clear and detailed views of any variance between the actual and desired state of your resources. This allows you to quickly detect and rectify anomalies.
Multi Tenancy

Scalr’s flexible multi-tenancy model enables any organizational model to be implemented easily and securely. Scalr supports multiple identity providers (great for MSPs) and provides complete isolation of teams, environments and resources to allow independent implementation of policy and access controls to suit the varying requirements of different business units and customers.

Scalr - Multi-Tenant Model
This model allows secure configuration and integration of provider credentials and the highly granular IAM system provides fine grained control of access to Scalr functionality.

Where next: Account & Environment Configuration

Terraform Service Catalog Tutorial

Scalr provides the capability to publish parameterised Service Catalog offerings based on Terraform templates that are stored in VCS repositories. This enables users without Terraform expertise to deploy their own infrastructure without having to rely on DevOps / Operations to deploy for them.

**Important:** This page explains how to set up a service catalog offering in Scalr

### Concepts

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account scope</td>
<td>Organisation layer in Scalr. Used to administer environments, policy, provider credentials and access controls</td>
</tr>
<tr>
<td>Environment scope</td>
<td>Working environments for teams and users. Multiple per account. Workspaces for running Terraform live at this scope</td>
</tr>
<tr>
<td>Provider credentials</td>
<td>Credentials for clouds and other providers set up at account scope and automatically added to workspaces as environment variables</td>
</tr>
<tr>
<td>Workspaces</td>
<td>Run time environment and stage storage for Terraform. Includes Terraform and Environment variables</td>
</tr>
<tr>
<td>Service catalog</td>
<td>Self service deployment backed by Terraform. Gives simplified interface to users so no Terraform knowledge is required</td>
</tr>
</tbody>
</table>
How it Works

1. Service catalog offering is linked to template in VCS
2. User requests deployment
3. IaC P pulls template and prompts user to enter values for Input Variables
4. IaC P creates “workspace” and runs the template to deploy infrastructure
5. User views results in the workspaces

Note: A workspace is a virtual environment inside IaCP that is configured to perform Terraform runs and store the state files.

Scalr uses the input variables in the template to create input fields when requesting a service from the catalog. Your Terraform template will work in Scalr service catalog without any modifications.

However Scalr also makes it easy for you to provide drop down lists of allowed values and other controls over the inputs users can provide, with some simple additions to your template.

==>> Most of this tutorial is done at environment scope (blue), but watch out for the bits that are done at account scope (green).

You can switch between Account and Environment scope using environment switcher in the top right corner.

If you need to create an environment, goto account scope -> Environments -> New Environment. Select IaCP and save.

Setup Overview

We will use an example of deploying an AWS instance to show you how to set up and test a service catalog offering using the following steps

1. The Terraform Template - Terraform template in Git Version Control System (VCS)
2. Link IaC P to VCS - So IaC P can pull templates
3. Provider Credentials - To enable providers to connect
4. Create Offering - Linked to the template in VCS
5. Service Request - Make a request to deploy from the offering
6. Input Parameters - Provide values to template input variables
7. Workspace - View the run and results in dedicated workspace
We will also show you how you can provide drop down lists and input controls to make it easier to complete a service request.

8. **Input Controls : Policy** - Using IaCP policies to control input values and provide drop down lists

9. **Input Controls : Variables** - Using IaCP variables to control input formats and values

### The Terraform Template

- The Terraform template must be in a git repo
- This example initially contains 3 files, `main.tf`, `variables.tf` and `outputs.tf`
- Create a new repo in your VCS account and add these 3 files with the contents shown below.

**main.tf** defines the resources and the provider.

```terraform
provider "aws" {  
  region = var.region
}

resource "aws_instance" "scalr" {  
  ami = var.ami  
  instance_type = var.instance_type  
  subnet_id = var.subnet  
  vpc_security_group_ids = var.sg  
  key_name = var.key
}
```

**variables.tf** with all of the values blank to force user input

```terraform
variable "region" {  
  description = "Region"
}

variable "instance_type" {  
  description = "Instance Type"
}

variable "subnet" {  
  description = "Subnet ID"
}

variable "sg" {  
  description = "AWS Security Group"  
  type = list(string)
}

variable "key" {  
  description = "AWS Key"
}

variable "vpc_id" {
```
(continues on next page)
variable "ami" {
  description = "AMI"
}

outputs.tf to provide the user with the information needed to use the deployed instance.

output "instance_id" {
  description = "Instance ID"
  value = aws_instance.scalr.id
}

output "public_ip" {
  description = "Public IP"
  value = aws_instance.scalr.public_ip
}

output "private_ip" {
  description = "Private IP"
  value = aws_instance.scalr.private_ip
}

Link IaCP to VCS

To be able to pull a template from VCS we must first authorise IaCP as an OAUTH application with the VCS. This example is Github.

1. Create new VCS Provider and select Github or Github Enterprise from the “Type” drop down
Note: For Github Enterprise you will need to enter the URL of your Github system.

<table>
<thead>
<tr>
<th>New VCS Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Name</td>
</tr>
<tr>
<td>URL</td>
</tr>
<tr>
<td>Callback URL</td>
</tr>
</tbody>
</table>

Copy the Callback URL and register a new OAuth application to generate the Client ID and Client Secret. Please read the documentation for detailed information.

2. Copy the callback URL, click the “register a new OAuth application” link and register a new OAuth app in Github

Register a new OAuth application

Application name *

scalar-demo

Something users will recognize and trust.

Homepage URL *

https://my.scalr.com

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

https://my.scalr.com/public/vcs/auth/...

Your application’s callback URL. Read our OAuth documentation for more information.
3. Copy the client ID and client secret and go back to Scalr

![New VCS Provider Form]

<table>
<thead>
<tr>
<th>ID</th>
<th>vcs-so8bgaua8f9emeo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>GitHub</td>
</tr>
<tr>
<td>Name</td>
<td>demo</td>
</tr>
<tr>
<td>Client Id</td>
<td>17b77308cf9be8b491c6</td>
</tr>
<tr>
<td>Client Secret</td>
<td>.................................</td>
</tr>
</tbody>
</table>

Copy the Callback URL and register a new OAuth application to generate the Client ID and Client Secret.

4. When you click “create” you will be redirected back to Github to complete the authorisation.

---

**Provider Credentials**

All providers in the template need credentials to authenticate with. For cloud providers these are typically access keys. All providers allow the credentials to be supplied via environment variables. IaCP allows you to set up credentials for cloud providers so that the necessary environment variables will automatically be set up in the workspace.

Setting up cloud credentials is a three step process and is done at the account scope (green).

1. Set up necessary authentication in the cloud provider (varies from cloud to cloud).
   - In the EC2 console navigate to IAM -> Users -> Select a user -> Security Credentials -> Create Access Key
   - Save the access key and secret key
2. Create Cloud Credentials in Scalr.
   - In Scalr at account scope navigate to cloud credentials -> add credentials. Select AWS, enter a name and the two keys and save.

3. Link Cloud Credentials to the environment.
   - Navigate to environments
   - Select the required environment and click on the clouds tab.
   - Click the link symbol on the right side for the cloud you want to link.
   - Select the credentials and save.
Create Offering

• Go to “Service Catalog”, and click on “Management”:

  - Click on “New Offering” and enter the information for the VCS provider, branch and directory where the template is stored
  - Set “Track” as required
1.3. Scalr Features

- Click “Save & Fetch”. IaCP will now fetch the template metadata from the VCS provider and the status should go to “Active” within a few seconds.
**Service Request**

- Go to “Service Catalog”, and click on “Request”:

![Service Catalog Diagram]

**Input Parameters**

- Provide values for all required inputs and click “Deploy”.

---

**Chapter 1. Scalr IaC Platform Overview**
You will now be redirected to the Workspace page where you can track the progress of the deployment.

**Workspace**

Workspace dashboard, including outputs from the last run.
Run details showing plan, cost, policy and apply phases.

Note: You may have noticed how difficult it was to fill in all the values. You had to go to the EC2 console to look up ami’s, security groups etc. Now we will show you how to make that much easier using IaCP policies and global variables.

Input Controls : Policy

IaCP policies (Account scope[green] -> Policy Engine -> Policy Groups) can be used to specify allowed values for a variety of different cloud resources parameters.

These policies can be bound to input variables in the template so that the allowed values are provided in a drop down list.

You will need to create an AWS Policy groups and then add 5 policies.

• cloud.instance_types
• cloud.locations
• cloud.security_groups
• cloud.networks (specify the cloud credentials)
• cloud.subnets (specify the cloud credentials and network)

• Click on “New Policy Group” and select “AWS” as the type:

[Image]

• Create the above policies “New Policy” and save.

[Image]

• Policy group must be linked to the environment. Click on “Environments”, “Policies”, then click on the green link on the right and save:

[Image]

Now modify the template to bind the policies to the required input variables. A new file scalr-module.hcl must be added to the repository as follows. There are 5 entries, some with multiple conditions, that are required.

```hcl
version = "v1"

# Bind region to a Policy that restricts to specific cloud locations.
# This acts an automatic look-up. IaCP queries the cloud api to obtain the possible
# values
variable "region" {
    policy = "cloud.locations"
    conditions = {
        cloud = "ec2"
    }
```

(continues on next page)
variable "vpc_id" {
  policy = "cloud.networks"
  conditions = {
    cloud = "ec2",
    cloud.location = "${var.region}"
  }
}

variable "subnet" {
  policy = "cloud.subnets"
  conditions = {
    cloud = "ec2",
    cloud.location = "${var.region}",
    cloud.network = "${var.vpc_id}"
  }
}

variable "instance_type" {
  policy = "cloud.instance.types"
  conditions = {
    cloud = "ec2"
  }
}

variable "sg" {
  policy = "cloud.security_groups"
  conditions = {
    cloud = "ec2"
  }
}

Notice how some of the bindings have conditions based on other variables. This enables IaCP to provide context sensitive lists such as the right subnets for the chosen VPC.

We will test this after the next section, but feel free to try it now.

**Note:** When you commit this new file to the repo the service catalog offering will automatically be updated in IaCP within a few seconds.
Input Controls : Variables

It isn't possible to provide policies for all the inputs, so we can use Global variables to create our own lists of allowed values.

Global variables can be created at either the account or environment scope. If created at the account scope they will be inherited by all environments. Global variable lists are made up of multiple label+value pairs. The label is what the end user will see, the value is what is actually used on the backend.

- Go to the environment scope:

  ![Environment Variables](image)

  You need to create two variables called `ssh_key` and `ami`.

  - Go to “Global Variables”, and “New Variable”:
    - Set the “Variable Type” as “List”. This will result in a dropdown for the user.
    - Click on the gear icon to enter values and labels.
• Make sure you save the entire Global Variable page.

Two new entries must be made in `scalr-module.hcl`

```hcl
variable "key" {
    global_variable = "ssh_key"
}

variable "ami" {
    global_variable = "ami"
}
```

After setting up these bindings and committing the changes to the repo you can make a new service catalog request and see how the request screen has changed. Note the drop down fields and the icons indicating policy and variable bindings.
This tutorial explains how to set up automatic Pull Request (PR) checks and automated runs (deployments) of Terraform templates stored in Git based Version Control Systems (VCS).

Prerequisites

- Your own account in a Git system (Github or Gitlab) for this tutorial.
- Your own AWS Account / IAM user.

Steps

We will use an example of an AWS instance to show you how to set up and test automation using the following steps.
### Step Description

<table>
<thead>
<tr>
<th>The Terraform Template</th>
<th>Create a Terraform template in Git Version Control System (VCS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provider Credentials</td>
<td>Add credentials for Terraform providers</td>
</tr>
<tr>
<td>Create VCS Provider</td>
<td>Connect IaCP to a VCS so the template can be pulled</td>
</tr>
<tr>
<td>Create Workspace</td>
<td>Link a workspace to the template in VCS</td>
</tr>
<tr>
<td>Set Variables</td>
<td>Set values for any input variables in the template</td>
</tr>
<tr>
<td>Dry Run Test</td>
<td>Create a PR to test dry run.</td>
</tr>
<tr>
<td>Automated Deployment Test</td>
<td>Merge to master to test automated deployment.</td>
</tr>
</tbody>
</table>

### The Terraform Template

**Page 2 of 8**  
**Create the Terraform template in VCS**

**Important:** This step is done in the VCS system.

1. The Terraform template must be in a git repo
2. This example contains 2 files, `main.tf` and `variables.tf` as shown below
3. Create a new repo in your VCS account and add these 2 files with the contents shown below.

`main.tf` defines the resources and the provider.
provider "aws" {
  region = var.region
}

data "aws_ami" "the_ami" {
  most_recent = true

  filter {
    name = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-.*"]
  }

  filter {
    name = "virtualization-type"
    values = ["hvm"]
  }

  owners = ["099720109477"] # Canonical
}

resource "aws_instance" "scalr" {
  ami = data.aws_ami.the_ami.id
  instance_type = "t2.nano"
}

variables.tf

variable "region" {
  description = "Region"
}

Provider Credentials

Add Cloud Credentials for the Terraform Provider

Important:

• This step is done in the AWS Console and at account scope

• Switch to account scope using the selector

All providers in the template need credentials to authenticate with. For cloud providers these are typically access keys. All providers allow the credentials to be supplied via environment variables. IaCP allows you to set up credentials for cloud providers so that the necessary environment variables will automatically be set up in the workspace.
1. Login to the AWS console

2. Create an access key as described in [link]

3. Save/copy the access key and secret key that are displayed

4. Now go to the Scalr UI and switch to account scope

5. Create Cloud Credentials in Scalr

   1. Click on [link]

   2. Click on [link] and then click on [link]

   3. Select AWS, enter a name and the two keys from AWS and save.

   ![Getting started with Scalr AWS tutorial.

   ADD CREDENTIALS

   CLOUD | AWS
   NAME  | Scalr
   ACCESS KEY ID | AKIAY7MSHEAQVFTV2MQ
   SECRET ACCESS KEY | -----------------
   ACCOUNT TYPE | REGULAR GOV CLOUD AWS CHINA

   6. Link Cloud Credentials to the environment.

   1. Click on [link]

   2. Select the required environment and click on the clouds tab

   ![CLOUDS POLICIES ACCESS]

   3. Click the link symbol on the right side for the cloud you want to link.

   4. Select the credentials you just created

   ![LINK TO ENVIRONMENT]

   5. Click on [link] and Save.
Create VCS Provider

Important:
- This step is done in the VCS system and at environment scope

Switch to environment scope using the selector

To be able to pull a template from VCS we must first authorise IaCP as an OAuth application with the VCS. This example is Github.

1. Create new VCS Provider and select Github or Github Enterprise from the “Type” drop down
Note: For Github Enterprise you will need to enter the URL of your Github system.

2. Copy the callback URL, click the “register a new OAuth application” link and register a new OAuth app in Github
Register a new OAuth application

**Application name** *

scalr-demo

Something users will recognize and trust.

**Homepage URL** *

https://my.scalr.com

The full URL to your application homepage.

**Application description**

Application description is optional

This is displayed to all users of your application.

**Authorization callback URL** *


Your application's callback URL. Read our OAuth documentation for more information.

[Submit application] [Cancel]

3. Copy the client ID and client secret and go back to Scalr
4. When you click “create” you will be redirected back to Github to complete the authorisation.

Create Workspace

Important:

- This step is done at environment scope
- Switch to environment scope using the selector
1. Click on WORKSPACES and then NEW WORKSPACE.
2. Set the name, select your VCS Provider and Repository.
3. Click Save.

**Warning:** Auto Apply is enabled in this example. This will cause automatic deployment or re-deployment of infrastructure.
Note: Provides an alternative directory were the `terraform` command will run. Defaults to the root directory or the subdirectory if specified.

Set Variables

Important:

- This step is done at environment scope.

Switch to environment scope using the selector.

For this tutorial you need to set a value for the `region` variable.

```
variable "region" {  
  description = "Region"
}
```

Note: You can set value to a region of your choice, but make sure that the chosen region has a default VPC.

1. Click on
2. Click on your workspace name in the list
3. Click on the VARIABLES tab
4. Click on NEW VARIABLE and
5. Set the region variable and value and click Save
Dry Run Test

**Page 7 of 8**

Test the automatic dry run on PR to Master

**Important:**
- This step is done in the VCS system and at environment scope.
- Switch to environment scope using the selector.

1. Login to your VCS system.
2. Create a new branch in the repo.
3. Make a change in the sub-branch, e.g. change the instance type to “t2.small”:

```terraform
resource "aws_instance" "scalr" {
  ami = data.aws_ami.the_ami.id
  instance_type = "t2.small"
}
```

4. Create a new PR to the Master branch and observe the checks.
3. The “Details” link in VCS will take you to dry run details in Scalr workspace.

Note: Only 3 steps in the run. Apply is never done in a dry runs

Automated Deployment Test

Test the automatic deployment on merge/commit to Master

(Only applicable if Auto Apply was enabled on the workspace)

Important:

- This step is done in the VCS system and at environment scope
• Switch to environment scope using the selector

1. Merge the pull request from the sub-branch to Master.

2. Observe and the Run in the workspace in IaCP. Navigate to the Workspace Runs tab and click on the latest run

For more details on automated deploys and dry runs goto *Automated Deployment.*
IaCP Remote Backend (CLI) Tutorial

Scalr is a remote backend for Terraform runs initiated via the Terraform CLI. This provides the following benefits.

- Centralized state storage and resource visibility
- Secure provider credentials
- Centralized and automated deployment policy implementation
- Cost estimation

**Important:** This page explains how to set up a Scalr workspace as a remote backend for CLI runs.

### Concepts

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account scope</td>
<td>Organisation layer in Scalr. Used to administer environments, policy, provider credentials and access controls</td>
</tr>
<tr>
<td>Environment scope</td>
<td>Working environments for teams and users. Multiple per account. Workspaces for running Terraform live at this scope</td>
</tr>
<tr>
<td>Provider credentials</td>
<td>Credentials for clouds and other providers set up at account scope and automatically added to workspaces as environment variables</td>
</tr>
<tr>
<td>Workspaces</td>
<td>Run time environment and stage storage for Terraform. Includes Terraform and Environment variables</td>
</tr>
</tbody>
</table>

### How it Works

1. API Token generated and added to your terraform environment
2. Template configured with remote backend details
3. `terraform init` creates workspace in IaCP
4. Variables are set in the workspace
5. CLI runs

()==>> Most of this tutorial is done at environment scope (blue), but watch out for the bits that are done at account scope (green).

You can switch between Account and Environment scope using environment switcher in the top right corner.

If you need to create an environment, goto account scope -> Environments -> New Environment. Select IaCP and save.

1.3. Scalr Features
Note: It is not necessary to pre-create a workspace in Scalr, the workspace will be created the first time `terraform init` is run when a remote backend is configured in the template. However you WILL need to create Terraform variables and assign values in the workspace before performing a run.

Setup Overview

We will use an example of an AWS instance to show you how to set up and test automation using the following steps:

1. **API Token** - Generate token and add to Terraform config
2. **Provider Credentials** - To enable providers to connect
3. **Configure Template** - Add remote backed to Terraform template
4. **Initialize Workspace** - Create workspace in IaCP
5. **Set Variables** - Set values for any input variables in the template
6. **CLI Runs** - Invoke run from CLI

API Token

1. Create API Token

2. Add a credentials to your CLI Configuration file.

<table>
<thead>
<tr>
<th>OS</th>
<th>File name and location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>file must be named named <code>terraform.rc</code> and placed in the relevant user's <code>%APPDATA%</code> directory. The physical location of this directory depends on your Windows version and system configuration; use <code>$env:APPDATA</code> in PowerShell to find its location on your system.</td>
</tr>
<tr>
<td>All other</td>
<td><code>~/.terraformrc</code></td>
</tr>
</tbody>
</table>
Provider Credentials

All providers in the template need credentials to authenticate with. For cloud providers these are typically access keys. All providers allow the credentials to be supplied via environment variables. IaCP allows you to set up credentials for cloud providers so that the necessary environment variables will automatically be set up in the workspace.

Setting up cloud credentials in is a three step process and is done at the account scope (green).

1. Set up necessary authentication in the cloud provider (varies from cloud to cloud).
   - In the EC2 console navigate to IAM -> Users -> Select a user -> Security Credentials -> Create Access Key
   - Save the access key and secret key

2. Create Cloud Credentials in Scalr.
   - In Scalr at account scope navigate to cloud credentials -> add credentials. Select AWS, enter a name and the two keys and save.

```json
credentials "my.scalr.com" {
    token = "<user-token>"
}
```
3. Link Cloud Credentials to the environment.
   - Navigate to environments
   - Select the required environment and click on the clouds tab.
   - Click the link symbol on the right side for the cloud you want to link.
   - Select the credentials and save.

Configure Template

Add backend configuration to your Terraform template.

1. Get the organization id from the environment switcher on the UI
2. Add a `terraform` block to your template. The hostname will be your local installation if IaCP is deployed on-prem. You can choose the workspace name at this point.

```terraform
backend "remote" {
    hostname = "my.scalr.com"
    organization = "<organization-name of environment>"
    workspaces {
        name = "<workspace-name>"
    }
}
```

## Initialize Workspace

1. Initialize the local workspace by running `terraform init`

```
$ terraform init
Initializing the backend...
Initializing provider plugins...
```

2. View the workspace in the UI
Set Variables

Any input variables that don’t have defaults and are not assigned values in `terraform.tfvars[.json]` must be given values in Scalr.

**Note:** Currently you need to create the variables and set values. In a later release of Scalr the variables will be created in the workspace automatically, but the values will still need to be set.

Navigate to the variables tab on the workspace screen and use New -> New Terraform Variable.
CLI Runs

Initiate a run from the CLI with either `terraform plan` or `terraform apply`.

CLI Runs can be viewed from the local terminal and from the workspace in the Scalr UI. They can also be approved in either the local terminal or in the UI.

You will see the usual plan and apply phases, but also the IaCP additions of cost estimation and policy checks. The policy checks won’t do anything until you have linked policies to your environment.

---

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

---------------------------------------------

Organization policy check:

Policy Group "OPA"

OpenPolicyAgent v0.14.2
Evaluating policies...

1. Cost (hard-mandatory)
   PASS

Policies: 1 passed, 0 failed.

---------------------------------------------

Do you want to perform these actions in workspace "CLI-2"?
Terraform will perform the actions described above. Only 'yes' will be accepted to approve.

Enter a value:
VCS Integration

Overview

Scalr can be integrated with version control system (VCS) providers to add powerful capabilities to the platform. Each account and environment can integrate with multiple VCS providers to provide the following capabilities:

- Link to Terraform templates for service catalog offerings.
- Link workspaces to a VCS repository to enable automated deployment and dry runs triggered by commits pull requests (PR’s).
- Register Terraform modules to make it easy to share and consume these modules across multiple Templates

Currently supported VCS’s:

- Github SaaS
- Github Enterprise
- Gitlab SaaS
- Gitlab CE/EE
- Azure Devops Services
A Github VCS Provider is authenticated against an OAuth application.

1. Create new VCS Provider and select Github or Github Enterprise from the “Type” drop down

![New VCS Provider](image)

**Note:** For Github Enterprise you will need to enter the URL of your Github system.

2. Copy the callback URL, click the “register a new OAuth application” link and register a new OAuth app in Github
Register a new OAuth application

Application name *

scalarm-demo

Something users will recognize and trust.

Homepage URL *

https://my.scalr.com

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *


Your application’s callback URL. Read our OAuth documentation for more information.

Register application  Cancel

3. Copy the client ID and client secret and go back to Scalr
4. When you click “create” you will be redirected back to Github to complete the authorisation.

## Gitlab

A Gitlab VCS provider is authenticated against an OAuth application.

1. Create new VCS Provider and select Gitlab or Gitlab CE/EE from the “Type” drop down
2. Copy the callback URL, click the “register a new OAuth application” link and register a new OAuth app in Gitlab
   
   • Check the following options
     
     – Confidential
     – api
     – read_repository
     – profile
– Allow requests to the local network from web hooks and services

– Allow requests to the local network from system hooks
3. Save and then copy Application ID and Secret

**Application: scalr**

- **Application ID**: [Redacted]
- **Secret**: [Redacted]
- **Callback URL**: https://my.scalr.com/public/wcs/auth/
- **Confidential**: Yes
- **Scopes**:
  - api (Access the authenticated user's API)
  - read_repository (Allows read-only access to the repository)
  - profile (Allows read-only access to the user's personal information using OpenID Connect)

4. Go back to IaCP, enter the Id and secret
5. When you click “create” you will be redirected back to Gitlab to complete the authorisation.

**Azure Devops Services**

An Azure Devops Services VCS provider is authenticated against an OAuth application.

1. Create new VCS Provider and select Azure Devops Services from the “Type” drop down
2. Copy the callback URL, click the “register a new OAuth application” link and register a new OAuth app in Azure DevOps

- Below is an example app configuration
- Only two scopes are required for Scalr, Code (read) and Code (status)

3. Click Create and then save the Application ID and Client Secret
4. Go back to IaCP, enter the Id and secret

New VCS Provider

ID: vcs-sqn10hg7h2i31sg
Type: Azure DevOps Services
Name: pg-azdos

Copy the Callback URL and register a new OAuth application to generate the Application ID and Client Secret. Please read the documentation for detailed information.

Application ID: 100[redacted]874-359888BAAC15
Client Secret: * * * * * * * * * * * * * * * * * * * * * * * * * * *
5. When you click “create” you will be redirected back to Azure Devops to complete the authorisation.

Variables

Overview

There are two types of variables that are used in Scalr IaC Platform:

- Terraform variables: Variables used in a Terraform template to gather values to inject into the Terraform template.
- Global variables: Multi-purpose key value store objects that can be used for input and tied to Terraform templates for policy enforcement.

These Terraform variables can be bound to Scalr global variables and Scalr policies in order to provide administrators with powerful controls over the values that can be entered by users when requesting services from the service catalog.

Terraform Variables

Terraform variables are used to gather information to inject into a Terraform template. This allows for easy Terraform template re-use when sharing templates across an organization.

In a Workspace

If a workspace is created through the UI manually, the variable names and values will need to be entered directly in the workspace prior to kicking off a Terraform run. In a future release, the information will be automatically populated based on the ingestion of the variables.tf file from the VCS provider. To create or update the Terraform variables in the workspace, click on workspaces and then variables:

In the Service Catalog

Terraform input variables can be use in Scalr IaC Platform to set default values or to prompt endusers for values in the UI, if using the service catalog. Scalr IaC Platform will use all input variables found in the all Terraform files in the repository to create input prompts. These variable will be presented as either “required” or “optional” depending on how they are configured. Required variables have no default and optional variables have a default.

An example of a required variable would be a key without a value as seen here:
variable "region" {}

In this scenario, the user will be prompted to fill in the information before deploying in the service catalog or workspace:

```
Service name
```

```
Tags
```

```
Search inputs
```

```
Required inputs
```

```
ami
```

```
instance_type
```

```
region
```

```
sg
```

```
subnet
```
Using the same example, you can make this an “optional” variable, by having a pre-defined value:

```hcl
variable "region" {
  default = "us-east-1"
}
```

Terraform variables are helpful, but don’t necessarily put constraints on the values that can be entered. The following sections teach you how to bind Terraform variables to apply controls on the values that can be entered.

### Global Variables

Global variables are multi-purpose key value store objects that can be used for input from users or admins. Global variables can be used for the following:

- To inject values from the service catalog into Terraform variables.
- To inject values from the service catalog into Scalr policy (naming conventions, tagging, etc)
- Limit end users choices for Terraform variables.

### Creating Global Variables

Global variables can be created, updated, edited, or deleted at the Scalr, account, environment or workspace scope. To create a global variable at the Scalr, account, or environment scope you can click on the “GV” button.

You have a few options when creating a global variable:
Once the global variable is created, click save and you will be able to use it in the various use cases below.
Global Variable Formatting

If a Scalr global variable is mapped to a Terraform input variable and the Scalr global variable has `printf` and/or REGEX formatting options, these will be applied to service catalog inputs.

Formatting is validated on exit from the field and warning will be displayed.

Variable:

```
Advanced Settings

Variable Type: Text
Format: 
Validator: ^[A-Z0-9]*$ / Modifiers
```

Warning:

```
Scalr Module

AAA

SSH_Private_Key
Private SSH key to connect to VMs

Value 'AAA' is not allowed for 'global_variable:Formatted' or does not exist
```

Binding Variables

The binding of Terraform variables to Scalr global variables and policies is achieved by creating a new file `scalr-module.hcl` in your Terraform template. The details for achieving global variable and policy binding are described in sections below.

`scalr-module.hcl` has the following capabilities:

- Written in HCL language to avoid learning a new language.
- Built-in versioning.
- Bind Terraform variables to global variables.
- Bind Terraform variables to Scalr IaC Platform policies.
- Terraform itself will ignore the file, it is specific to Scalr IaC Platform.

`scalr-module.hcl` supports and requires versioning:

```
version: "v1"
```

Binding to Global Variables

Terraform input variables can be bound to Scalr global variables in order to set a list of allowed values and/or define formatting rules. A list of values will be presented to the user as a drop down list in the UI. As an example, you might require your users to input a billing code prior to provisioning their resources. To do this with variables, you would do the following:

Define a Terraform variable like this:
variable "billing_code" {
  global_variable = "billing_code"
}

Create a `scalr-module.hcl` file with the following (versioning is required):

```hcl
version = "v1"
variable "billing_code" {
  global_variable = "billing_code"
}
```

In the UI, create global variable called “billing_code” of type “list” with a set of values. Scalr will display the label for each value in the resulting drop down:

![Variable Settings](image)

### Advanced Settings

**Variable Type**: List

**Value**:

- 123
- 456

**Note**: The Terraform input variable and the Scalr global variable DO NOT have to have the same name.

Now in the service catalog you will have a “billing_code” field that is mapped to the Terraform variable:

![Billing Code Field](image)

Formatting via `print` directives and/or REGEX is enforced in the UI with warnings displayed if any violations occur.
Note that the name of the variable displayed in the UI is the Terraform variable name and not the Scalr global variable name. The \{x\} indicates the field is bound to a Scalr global variable.

**Binding to Policy**

To make Terraform variables even more powerful, you can combine them with policy to ensure users are deploying their resources in a compliant way. Going back to the region example above, you'll first want to create the policy at the account scope. To do this, click on the main menu at the account scope, click on policy, and create a cloud.location policy for the cloud you are building in:

![Policy Setup](image)

In the example above we are restricting the user to deploy in us-east-1 or us-west1 in AWS.

Now in the variables.tf, add the following if it is not already there:

```hcl
variable "region" {}
```

In the scalr-module.hcl, add the following:

```hcl
variable "region" {  
  policy = "cloud.locations"  
  conditions = {  
    cloud = "ec2"  
  }  
}
```

By connecting policy, variables.tf, and scalr-module.hcl we will now have the user prompted for the region variable and they will be limited to only choose what is part of the policy:
Input Formatting

If a Terraform variable is pre-populated with a value and a type, this will affect the way they variables are presented in the service catalog. In this example, the simple string type variable was used to create a pre-populated field:
Also, Scalr will inherit MAP and LIST type Terraform variables to create prompts. This is an example of a MAP variable and it’s prompt:

```terraform
variable "images" {
  type = "map"
  default = {
    us-east-1 = "image-1234"
    us-west-2 = "image-4567"
  }
}
```

This is an example of a LIST variable and it’s prompt:

```terraform
variable "zones" {
}
```

(continues on next page)
Advanced Policy Binding

Policy Conditions

Policy binding must include any and all required conditions specified in the Scalr policy itself. Policy conditions vary between policy types so please check the details of each policy type carefully. Many policies have multiple conditions which can be optional in some cases and mandatory in others.

For example cloud.locations policy can be common to all cloud credentials or specific.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>cloud.locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>Restricts locations that can be used on a cloud</td>
</tr>
<tr>
<td>STATUS</td>
<td>ENFORCED</td>
</tr>
</tbody>
</table>

However in the case of cloud.networks policy credentials and location must be specified by tags.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>cloud.networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>Restricts networks (VPCs, VLANs) that can be used</td>
</tr>
<tr>
<td>STATUS</td>
<td>ENFORCED</td>
</tr>
<tr>
<td>MODE</td>
<td>WHITELIST</td>
</tr>
</tbody>
</table>
When binding a Terraform input variable to a policy you must include all mandatory conditions and any optional conditions that have been specified in the policy. e.g for cloud.networks a binding entry in `scalr-module.hcl` to match the above policy would need to be configured as follows.

**Note:** Credentials are not included in policy bindings as the credentials in Scalr may not be utilized by the Terraform template but the policy still needs to be applied. Credentials can be defined directly in the template or possibly provided by the users during the service catalog request.

```hcl
variable "networks" {
    policy = "cloud.networks"
    conditions = {
        cloud = "ec2"
        cloud.location = "us-west-2"
    }
}
```

For full details for policy compatibility with Terraform please see *Input Policy*.

### Condition Variables in Policy Bindings

Terraform input variables can be used in the “conditions” clause of policy bindings to allow the applicable policy to be set correctly according to context. For example if a user is given the choices regarding the configuration of the deployment this may affect which policies need to be applied in cases where the policy requires specific conditions. Examples could be things like the location (region) or network.

Taking Location (Region) as an example lets assume we have policies in place for each region that restrict the networks that can be used.

![AWS-SCALR](image)

We then have an input variable “region”, configured to capture the users required location and variable to capture the required network.

**Variable declaration:**

```hcl
variable "region" {}
variable "network" {}
```

The “network” variable is then bound to policy using the value of the “region” variable.

```hcl
variable "networks" {
    policy = "cloud.networks"
    conditions = {
        cloud = "ec2"
        cloud.location = "${var.region}"
    }
}
```

Now the appropriate policy will be used and this will also ensure the use only has the valid values to select from when choosing a network.
Terraform Provider Authentication

Overview

Scalr IaC Platform uses Terraform variables to pass provider credentials during the Terraform runs. In cases where a top provider is used, like AWS, Azure, Google, and VMware, the credentials can be stored at the Scalr account level and will automatically be passed as a variable to the provider. All sensitive variables, like credentials, will be hidden in the UI and output.

Using Terraform Provider Credentials Scalr

Built-In Provider Credentials

Terraform allows the credentials required by providers to be consumed from environment variables when using the CLI. For example if using AWS you can declare two variables in you local environment and then code the provider with just the region parameter.

```bash
export AWS_ACCESS_KEY_ID=[ACCESS_KEY_VALUE]
export AWS_SECRET_ACCESS_KEY=[SECRET_KEY_VALUE]
```

Then your template would simply contain a provider block like this.

```terraform
provider "aws" {
  region = var.region
}
```

Scalr enables the same approach by automatically publishing the cloud credentials for an environment into workspaces as variables. This enables provider configurations in the template to consume the credentials that are configured in Scalr in a secure manner without the need to access external secret storage systems or code credentials into tfvars files.

Built-in cloud credential variables are currently implemented for AWS, Azure, Google, and VMware providers and only the credentials that are linked to an environment are published.

This feature allows templates that are being run via any method to consume the credentials, including service catalog, CLI runs and automated runs from VCS.

The variables that are published for each cloud can be viewed on the “variables” tab of a Workspace. These variables cannot be edited via the UI or API/CLI and they are set as “sensitive” so the values are masked from view.
Non Built-In Provider Credentials

If you are using a provider that is not in the built-in list, you can still use that provider and pass the credential through a Terraform variable or an environment variable.

- If a Terraform variable: The credential variable must be declared in the Terraform template and variables file. The value can then be placed in the Scalr UI -> workspaces -> Terraform variables.
- If an environment variables: The credential does not need to be declared in the Terraform template or variables file. The value must be placed in the Scalr UI -> workspaces -> environment variables.

Each provider will describe how to authenticate to it. For example, Oracle is not a built-in credential, but it is fully supported as Scalr supports any terraform provider. The Oracle provider supplies documentation on how to authenticate:

Related links
Workspace Management

Overview

A Scalr workspace is the object in which you can manage your Terraform deployments whether it originated from the service catalog, API, CLI, or VCS provider. Within a workspace you can do the following:

- View the history and details of runs
- View resources
- Manage variables
- Queue new runs
- Destroy infrastructure
- Edit existing workspace configuration

Each workspace will store the Terraform state and act as what would normally be a directory if you were running Terraform locally.

Creating a Workspace

A workspace can only be created through the UI for now, there are plans to make this available via the API/CLI. Workspaces only need to be created manually if you wish to use devops automation.

- A workspace will automatically be created for you if you order via the service catalog.
- Workspaces for used for IaCP Remote Backend (CLI) Tutorial are created automatically when terraform init is run.

Workspaces for Devops automation are created through the workspace page by clicking new workspace:

Then link it to your VCS provider:

Note: provides and alternative directory were the terraform command will run. Defaults to the root directory or the subdirectory if specified.
Runs

Runs occur when you queue a new plan or execute changes on existing workspaces. Each time a run is executed it will create history about the run, which can be found by clicking on the name of the workspace, then the “runs” tab.

To find out more information about each run, click on the “run id”. Each tab within the run can be clicked on to see more detail about the Terraform plan, cost estimate, policy, and apply:

**Plan** - Displays the Terraform plan that will be executed:

![Plan Example](image)

**Cost Estimate** - Displays the cost that will be incurred or the proposed cost difference if it is a change to existing state:

![Cost Estimate Example](image)

**Policy Check** - Displays the results of the OPA policy checks:

![Policy Check Example](image)
**Apply** - Displays the results and output of the apply:

Clicking on the “commit” ID will redirect you to the VCS provider to show the last change that occurred in the Terraform template.

**Canceling a Run**

A run can be cancelled using the Cancel button on the run page.

**Note:** The cancel action can be used to interrupt a run that is currently planning or applying. Performing a cancel is roughly equivalent to hitting `ctrl+c` during a Terraform plan or apply on the CLI. The running Terraform process is sent an INT signal, which instructs Terraform to end its work and wrap up in the safest way possible.

This endpoint queues the request to perform a cancel; the cancel might not happen immediately. After canceling, the run is completed and later runs can proceed.

1.3. Scalr Features
Variables

If you are creating a workspace and integrating it with your existing CI/CD pipeline, you may prefer to store the variables directly in Scalr rather than the Terraform template to make the template more dynamic. To do this, reference the variables in your template:

```terraform
variable "region" {}  
variable "bucket_name" {}  
```

Then add them to your workspace in the UI:

Find out more about variables here.
Automated Deployment

Scalr workspaces can be bound to a VCS based repo branch [+ directory] so that the template that is being executed by Scalr is always from the latest commit. This binding is defined when a workspace is created and cannot be changed.

Under workspaces click new workspaces

| VCS Provider | demo_pg |
| Repository | pgale61/demo_vcs |
| Terraform Version | 0.12.8 |
| Auto Apply | Auto apply changes when a Terraform plan is successful. |

The effect of this binding is that every commit or merge to the linked repo and branch will trigger a full run. This run may deploy or re-deploy resources depending on the changes in the commit and the current state of the resources.

This automated run is achieved as follows.

When the workspace is created Scalr will create a webhook in the VCS to be fired back to Scalr whenever there is a
commit in that repo. There is only one webhook per repo and Scalr will process each webhook request and determine if the commit applies to a branch that is linked to a workspace. If so the full run, i.e. `terraform apply` is triggered. If the plan, cost estimation and policy checks are successful the run will need to be approved in the UI, unless the “auto apply” toggle is set.

**VCS Dry Runs**

Linking a VCS branch to a workspace can also cause automated testing of a pull request to take place. These automated tests are known as “dry runs” and in the context of Scalr they include the plan, cost estimation and policy phases of a run. They never include the apply phase. The purpose of these tests is to provide some validation that the pull request can be merged into the target branch. These tests are triggered by the webhook and occur under the following circumstances.

- When a pull request is made to a branch that is linked to a workspace
- When a commit is done in a branch that is the source of an open pull request to a branch that is linked to a workspace.

Example:

- I have a branch called “Dev” linked to a workspace
- I create a new branch “New_feature” and commit a change
- I make a pull request request from New_feature to Dev. A speculative run will be triggered.
- I make another commit to “New_feature” while the pull request is still open. Another speculative run will be triggered.

The dry runs can be observed in the VCS
The “details” link is a link to the run details inside Scalr. The run will be associated with the workspace that is linked to the target of the pull request but is only accessible from the VCS link. Dry runs do not appear on the run list for a workspace.

**Sharing Workspace State**

**Overview**

It is common practice to reference outputs from other workspaces so that a Terraform template can make use of resources that have been deployed elsewhere. This is known as “remote state” and accessing remote state is done using the `terraform_remote_state` data source as shown in this example.

```hcl
data "terraform_remote_state" "state-1" {
  backend = "remote"

  config = {
    hostname = "<host>"
    organization = "<org_id>"
    workspaces = {
      name = "<workspace name>"
    }
  }
}
```

When you include a `terraform_remote_state` block in your template you can then access any outputs in that remote state.

**Note:** You can only access the outputs of a remote state. You cannot access the resources in the the remote state directly.

In Scalr you can access the remote of another workspace in exactly the same way by including `terraform_remote_state` data sources in your template that reference any other workspace in any other Scalr environment that you have access to.
Example

This example shows how to access the remote state for workspace ‘webapp-vpc’ in the ‘marketing’ environment in our hosted Scalr system at my.scalr.com. For self hosted systems use the URL of your Scalr installation.

1. Switch to ‘marketing’ environment and get the organization id from the environment switcher on the UI

2. Add a `terraform_remote_state` data source to your template as follows

```hcl
data "terraform_remote_state" "vpc-1" {
  backend = "remote"

  config = {
    hostname = "my.scalr.com"
    organization = "org-xxxxxxxxxxx"
    workspaces = {
      name = "webapp-vpc"
    }
  }
}
```

You can now reference the outputs from the remote state with constructs like

data.terraform_remote_state.vpc-1.outputs.vpc_id.

See [2] for more details on use remote state.

Input Policy

Overview

The Scalr policy engine is a library of pre-built policies specifically created to restrict Terraform variable choices in the service catalog. The service catalog is the only use case where these types of policies can be applied, Open Policy Agent (OPA) policies can be applied during a Terraform run. A Terraform template and Scalr policies are linked together with a `scalr-module.hcl` file and applied only when the policy is created and the Terraform template is imported into the service catalog.

The policy engine is part of the account scope, thus enabling business units to implement consistent governance policies across all of their cloud environments.
There are three steps to defining and enabling policies in the Scalr UI:

1. Create a policy group.
2. Add policies and policy conditions to the policy group.
3. Link the policy group to the required environments.

Policy conditions provide fine grained control over the application of a policy, allowing for policy variations across different clouds, locations, accounts and more.

Creating Scalr Policy

Policies are defined by clicking on policy engine -> policy groups -> new policy group which will display this screen:

Policy groups require a name and type. The type determines the options available for policy rules and includes types related to all clouds and cloud specific types. Cloud specific types will only appear in the drop down if there are cloud credentials configured for that cloud at either the Scalr scope or in the currently active Account. For Terraform environments, the following policies are allowed:

- AWS
- Azure
- Google Compute Engine
- Open Policy Agent

The policy reference below explains the policy types and their associated rules.

Linking Policy Groups to Environments

Policy groups will only take effect when they are linked to environments. You can only link one policy group of each type to an environment.

Click on environments in the bookmarks bar or main menu and select the environment you wish to link policy groups to.

Click on the link icon on the right-hand side, link the required policy group to the environment, and save.
Using Global Variable in Policies

*Global variables* in policies make it possible to vary the details of the policy at any level where global variables can be set. A good example of this is a policy that enforces tagging on resources. The `cloud.tags` policy is available for all cloud types and provides a list of tags that must be applied to resources ordered from the service catalog in the environment the policy applies to.
By using a global variable name (in this example, `{MyGlobalVariable}`) rather than an actual value, the value applied to the tag will be determined from the global variable at the time the template is launched, based on the value of the variable at that time.

**Scalr Policy Engine Compatibility with Terraform**

Scalr policies are out of the box policies used to restrict the Terraform variable selection in the service catalog. For example, Terraform templates deployed through the service catalog in a specific environment can only build in AWS us-east-1 or us-west-1. Unlike *Open Policy Agent (OPA)*, these policies are enforced upfront and only in the service catalog use case.
Scalr policies tie together with Terraform templates with three components:

- **variables.tf** - Where the Terraform variable is defined.
- **scalr-module.hcl** - Where the Terraform variable is linked to the Scalr policy.
- **Scalr Policy Engine** - Where the actual Scalr policy is created and applied to an environment.

Let’s walk through a simple example to give you a better idea of how everything is linked together. First, we’ll create a S3 template:

```terraform
provider "aws" {
  region = var.region
}
resource "aws_s3_bucket" "scalr" {
  bucket = var.bucket_name
  acl = "private"
}
```

In the template, we have created a variable for region and want the end user to input the region value in the service catalog. To make it a required input in the service catalog, leave the variable default value blank in the `variables.tf`:

```terraform
variable "region" {} 
variable "bucket_name" {} 
```

Next, you will need to create a `scalr-module.hcl` to link the Terraform variable to the Scalr policy:
Now that the three files have been created, you just need to create the policy in the Scalr UI and link it to an environment. Navigate to the account scope:

Go to the policy engine:

Click on “new policy group” and select “AWS” as the type:

Create a policy for cloud.locations and select the regions that are allowed and click ok:
Now that the policy is created, link it to the environment by clicking on environments, policies, then click on the green link on the right and save:

Now all AWS Terraform templates that are deployed through the service catalog in that environment will be restricted to the region(s) you have selected, if the `scalr-module.hcl` exists.

For a full end to end example of setting up a service catalog item with policy, please click here: Terraform Service Catalog Tutorial

**Policy Reference**

Below is a list of possible cloud types with their aliases in the Scalr policy Engine.

- **ec2** - Amazon Web Services
- **gce** - Google Cloud Platform
- **azure** - Microsoft Azure

All cloud-dependent policies require “cloud” conditions as seen below or in the example above.
Below is a list of the policy types which are supported by each cloud. The only differences are in conditions, where you have to specify cloud-dependent values (e.g., `cloud`, `cloud.location`, etc.).

<table>
<thead>
<tr>
<th>Policy</th>
<th>Condition</th>
<th>Required</th>
<th>Condition value</th>
<th>Supported TF Variable types</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloud.instance.name.template (Sets a naming convention for instance names)</td>
<td>cloud.location</td>
<td>No</td>
<td>Cloud location identifier</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>os.type</td>
<td>One of: [linux, windows]</td>
</tr>
<tr>
<td></td>
<td>cloud.location</td>
<td>No</td>
<td>Cloud location identifier</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>os.type</td>
<td>One of: [linux, windows]</td>
</tr>
<tr>
<td>cloud.instance.types (Restricts instance types that can be used)</td>
<td>cloud.location</td>
<td>No</td>
<td>Cloud location identifier</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>os.type</td>
<td>One of: [linux, windows]</td>
</tr>
<tr>
<td>cloud.locations (Restricts locations that can be used on a cloud)</td>
<td></td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>cloud.networks (Restricts networks (VPCs, VLANs) that can be used)</td>
<td>cloud.location</td>
<td>ec2, openstack, otc, raxs, az, gce - yes, azure, gce - absent</td>
<td>Cloud location identifier</td>
<td>string, list</td>
</tr>
<tr>
<td>cloud.storage.maximum_size (Sets a maximum size for storage in GB)</td>
<td>cloud.location</td>
<td>No</td>
<td>Cloud location identifier</td>
<td>number</td>
</tr>
<tr>
<td>cloud.subnets (Restricts subnets that can be used inside networks)</td>
<td>cloud.location</td>
<td>Yes</td>
<td>Cloud location identifier</td>
<td>string, list</td>
</tr>
<tr>
<td></td>
<td>cloud.network</td>
<td>Yes</td>
<td>Network identifier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>resource.group (for azure only)</td>
<td>Yes</td>
<td>Resource group identifier</td>
<td></td>
</tr>
<tr>
<td>cloud.tags (Sets list of tags that will be applied to resources)</td>
<td>cloud.location</td>
<td>No</td>
<td>Cloud location identifier</td>
<td>map</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>os.type</td>
<td>One of: [linux, windows]</td>
</tr>
</tbody>
</table>

**scalr-module.hcl** example for a common policy type:

```hcl
version = "v1"

variable "region" {
  policy = "cloud.locations"
  conditions = {
    cloud = "ec2"
  }
}
```

(continues on next page)
variable "vpc_id" {  
  policy = "cloud.networks"  
  conditions = {  
    cloud = "ec2"  
    cloud.location = "${var.region}"  
  }  
}

Amazon Web Services

<table>
<thead>
<tr>
<th>Policy</th>
<th>Condition</th>
<th>Required</th>
<th>Condition value</th>
<th>Supported TF Variable types</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws.iam.instance_profiles (Restricts IAM instance profiles that can be applied to instances)</td>
<td>cloud.location</td>
<td>Yes</td>
<td>AWS cloud location identifier</td>
<td>string, list</td>
</tr>
<tr>
<td>aws.iam.roles (Restricts IAM Roles that can be used)</td>
<td>cloud.location</td>
<td>Yes</td>
<td>One of: [emr, lambda, applicationAutoScaling]</td>
<td>string, list</td>
</tr>
<tr>
<td>aws.kms.keys (Restricts KMS keys available for storage encryption)</td>
<td>cloud.location</td>
<td>Yes</td>
<td>AWS cloud location identifier</td>
<td>string, list</td>
</tr>
<tr>
<td>aws.rds.instance_types (Restricts Instance Types that can be used by RDS)</td>
<td>aws.rds.dbengine</td>
<td>No</td>
<td>One of: [mysql, oracle-se, oracle-se1, oracle-se2, sqlserver-ee, sqlserver-se, sqlserver-ex, sqlserver-web, postgres, aurora, aurora-postgresql, mariadb]</td>
<td>string, list</td>
</tr>
<tr>
<td>cloud.resource.name.prefix (Sets a prefix for cloud resource names)</td>
<td>cloud.location</td>
<td>Yes</td>
<td>One of: [elb, rds, s3]</td>
<td>string</td>
</tr>
<tr>
<td>cloud.security_groups (Sets or Restricts Security Groups that can be used)</td>
<td>cloud.location</td>
<td>Required with cloud.network</td>
<td>AWS cloud location identifier</td>
<td>string, list</td>
</tr>
<tr>
<td>cloud.network</td>
<td>VPC identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cloud.service</td>
<td>No</td>
<td>One of: [ec2, rds, elb, alb, efs, lamda]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cloud.storage.volume_types (Restricts volume types that can be used)</td>
<td>cloud.location</td>
<td>Yes</td>
<td>AWS cloud location identifier</td>
<td>string, list</td>
</tr>
</tbody>
</table>

scalr-module.hcl example for an AWS policy type:

version = "v1"

variable "region" {
  policy = "cloud.locations"
}

1.3. Scalar Features
conditions = {
    cloud = "ec2"
}
}

variable "vpc_id" {
    policy = "cloud.networks"
    conditions = {
        cloud = "ec2",
        cloud.location = "${var.region}"
    }
}

variable "subnet" {
    policy = "cloud.subnets"
    conditions = {
        cloud = "ec2",
        cloud.location = "${var.region}"
    }
}

Microsoft Azure

<table>
<thead>
<tr>
<th>Policy</th>
<th>Condition</th>
<th>Required</th>
<th>Condition value</th>
<th>Supported TF Variable types</th>
</tr>
</thead>
<tbody>
<tr>
<td>azure.availability_sets (Restricts Availability Sets that can be used)</td>
<td>resource.group</td>
<td>Yes</td>
<td>Resource group identifier</td>
<td>string, list</td>
</tr>
<tr>
<td>azure.resource_groups (Restricts Resource Groups that can be used)</td>
<td>cloud.location</td>
<td>No</td>
<td>Region identifier</td>
<td>string, list</td>
</tr>
<tr>
<td></td>
<td>cloud.service</td>
<td>No</td>
<td>One of: [compute, network]</td>
<td></td>
</tr>
<tr>
<td>azure.storage_accounts (Restricts Storage Accounts that can be used)</td>
<td></td>
<td></td>
<td></td>
<td>string, list</td>
</tr>
<tr>
<td>cloud.security_groups (Sets or Restricts Security Groups that can be used)</td>
<td>cloud.location</td>
<td>No</td>
<td>Region identifier</td>
<td>string, list</td>
</tr>
<tr>
<td></td>
<td>cloud.network</td>
<td></td>
<td>Network identifier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>resource.group</td>
<td>No (Yes if cloud.network specified)</td>
<td>Resource group identifier</td>
<td></td>
</tr>
</tbody>
</table>

scalr-module.hcl example for Azure:

```hcl
version = "v1"

variable "region" {
    policy = "cloud.locations"
    conditions = {
        cloud = "azure"
    }
}
```

(continues on next page)
variable "resourcegroup" {
  policy = "azure.resource_groups"
  conditions = {
    cloud = "azure"
  }
}

---

### Google Cloud Platform

<table>
<thead>
<tr>
<th>Policy</th>
<th>Condition</th>
<th>Required</th>
<th>Condition value</th>
<th>Supported TF Variable types</th>
</tr>
</thead>
<tbody>
<tr>
<td>gce.custom_instance_type.maximum_ram (Sets a maximum size for custom instance types in GB)</td>
<td>cloud.location</td>
<td><strong>required</strong></td>
<td>GCP region identifier</td>
<td>number</td>
</tr>
<tr>
<td>gce.custom_instance_type.maximum_vcpus (Sets a maximum number of VCPUs for custom instance types)</td>
<td>cloud.location</td>
<td><strong>required</strong></td>
<td>GCP region identifier</td>
<td>number</td>
</tr>
<tr>
<td>gce.network_tags (Sets or Restricts Network Tags that can be used)</td>
<td>cloud.location</td>
<td><strong>required</strong></td>
<td>GCP region identifier</td>
<td>list</td>
</tr>
<tr>
<td>gce.service_accounts (Restricts service accounts that can be applied to instances)</td>
<td>cloud.location</td>
<td><strong>required</strong></td>
<td>GCP region identifier</td>
<td>string, list</td>
</tr>
</tbody>
</table>

### Open Policy Agent (OPA)

#### Overview

Policy is one of the most important aspects to scaling cloud adoption. You want to ensure that your users can provision resources as needed and avoid a ticketing/review process that could slow down a deployment process. Adding policy to a pipeline ensures your users are able to provision resources within a defined set of guardrails reducing security, financial, and operational risk.

Open Policy Agent (OPA) is a code based policy engine allowing users to write policy to check Terraform plans as they come through the Scalr pipeline. OPA is a growing open source community that has already been widely adopted in the Kubernetes community and is the future of policy as code. Scalr guarantees all Terraform plans are checked against OPA policy, in addition to there being different enforcement levels per policy. Scalr also allows for different pipelines/environments/accounts to follow different policies based on customer or business need. A few examples of commonly created policies are:

- Naming Conventions - Enforce workspace, instance name, and tagging standards.
- Cost - Avoid unexpected costs.
- Blast Radius - Avoid changes to critical components (security groups, autoscaling, etc) that could potentially break an application.
- Compliance Checks - Ensure your deployments following company and compliance standards.
- Image Selection - Only allow images that have been approved by your organization.
- And many many more..

The advantage of using policy as code is that there are no limitations to what policies can be created. You do not have to wait on a new release, a feature request, or anything else as you have full control over the types of policies that are written. To learn more about OPA, please visit the following site: [OPA Website](https://opache.com)

1.3. Scalr Features 93
OPA policies are stored in VCS repositories and pulled into Scalr, no code is stored directly in Scalr. Learn more about linking with a VCS repository here.

Policy Checking Process

If an OPA policy is created and linked to an environment, this process will be followed:

At the beginning of the policy stage in the Terraform plan, Scalr prepares all-inclusive input data for policies to evaluate against. It is a JSON object that contains both the Terraform plan and it’s artefacts, as well as the run context data that supports proper policy evaluation.

The JSON has the following structure:

```json
{
   "tfplan": "<terraform plan content>",
   "tfrun": {
      "workspace": {
         "name": "<name>",
         "description": "<description>",
         "auto_apply": false,
         "working_directory": "<working_directory>",
         "tags": {
            "<tag_name>": "<tag_value>"
         },
         "vcs_repo": {
            "identifier": "<org/repo>",
            "display_identifier": "<display_identifier>",
            "branch": "<branch>",
            "ingress_submodules": false
         }
      },
      "cost_estimate": {
         "prior_monthly_cost": 0.0,
         "proposed_monthly_cost": 0.0,
         "delta_monthly_cost": 0.0
      },
      "credentials": {
         "<cloud>": "<cc_id>"
      },
      "source": "<run_source>",
      "created_by": {
         "name": "<name>",
         "full_name": "<full_name>",
         "email": "<email>"
      }
   }
}
```

- `input.tfplan` contains Terraform plan data created with the `terraform show -json [planfile]`
command. It serves as the main input for the OPA evaluation stage.

To learn more about the extent of the tfplan context, please visit the Terraform documentation on this:

- **input.tfrun** contains a few more keys:
  - **Workspace tags**
  - Cloud credentials that are linked to the Scalr environments. Cloud names are transformed through mapping to match the ones Terraform is using for providers (“gce” -> “google”, “azure” -> “azurerm”, etc).
  - **source**: source type of a run. Can have one of the following values:
    * **api**: The run was kicked off via the Terraform API
    * **cli**: The run was kicked off via the Terraform CLI
    * **configuration-version**: The run was kicked off by uploading a new configuration version via the API
    * **service-catalog**: The run was kicked off by ordering an offering from the service catalog
    * **ui**: The run was kicked off manually through the UI
    * **vcs**: The run was kicked off by a merge/commit/pull request webhook from the VCS repository linked to the workspace.
  - **created_by**: username, full name and email of a user who triggered the run.
  - **cost_estimate**: this is populated with results of the cost estimate stage of the run.

### Creating the OPA Policy

You’ll need to create the OPA policy in a VCS provider, here is example policy name `cost-check.rego` to deny any deployment that will cost more than $10 per month:

This policy must end with a .rego extension.

```rego
package terraform

import input.tfplan as tfplan
import input.tfrun as tfrun

deny[reason] {
  cost_delta = tfrun.cost_estimate.delta_monthly_cost
  cost_delta > 10
  reason := sprintf("Plan is too expensive: $%.2f, while up to $10 is allowed", \n  →[cost_delta])
}
```

The policy or policies must then be referenced in a single `scalr-policy.hcl` file within the same repository, which instructs the Scalr if the policy should be enabled and at what enforcement level. The following enforcement levels are allowed:

- **Hard** - The Terraform run is stopped if a violation occurs.
- **Soft** - The Terraform run must be approved if a violation occurs.
- **Advisory** - A notification occurs if a violation occurs.
version = "v1"

policy "cost-check" {
  enabled = true
  enforcement_level = "hard-mandatory"
}

The syntax for multiple policies in the `scalr-policy.hcl` is as follows:

```hcl
version = "v1"

policy "cost-check" {
  enabled = true
  enforcement_level = "hard-mandatory"
}

policy "workspace-name" {
  enabled = true
  enforcement_level = "advisory"
}
```

You should now have two files created:

```bash
$ ls -la
.
..
cost-check.rego
scalr-policy.hcl
```

Push both of these files into the VCS repository and then link them to a Scalr environment in the next section.

**Linking the Policy**

After creating the policy and linking Scalr to the VCS repository, you can link the policy to an environment. To link the policy in Scalr, you must navigate to the account scope, click on the Scalr icon on the top left, and then “policy engine”:

- Navigate to the account scope:

  ![Account Scope](image)

- Go to the policy engine:
• Click on “new policy group” and select “Open Policy Agent” as the type:

• Click on “new policy”, select VCS repository, and enter in the information for the repository where the OPA
If everything is setup correctly, the policies will appear with the corresponding enforcement levels:

- Now that the policy is created, link it to the environment by clicking on “environments”, “policies”, then click on the green link on the right and save:
• Once the policy is applied to the environment you will then see a new tab for it in your runs:

**OPA Dry Runs**

To test a policy prior to implementing it in production, you can open a pull request on a policy to preview the changes against a Terraform run. This will help you figure out how impactful the change will be, who it impacts, and what updates are needed. A preview will never fail a run, it is informative prior to merging to master. To do start an OPA dry run, open a pull request against master with the changes.

Once the pull request is made, you will see it appear in the existing policy:
The preview mode can be turned on and off as needed:

In the next run, you will see the implemented policy vs the preview policy:

Once the pull request is merged to master, the preview will disappear and the new policy will be applied to all runs.

**OPA Examples**

For examples of OPA policies, please visit our Github repository: [link]

**CLI Usage**

**Overview**

The [link] is used manage the operations and state of Terraform deployments. The CLI can be configured to work with either [link] or [link] backends.

- Local backend: Stores [link] and performs operations via the CLI.
- Remote backend: CLI calls [link] to store state and perform [link].

Scalr provides a remote backend with fully compatible API.
CLI Commands

Scalr remote backend provides the following support for commands of Terraform CLI. Scalr only supports the CLI for versions $\geq 0.12.0$:

<table>
<thead>
<tr>
<th>CLI command</th>
<th>IaCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply</td>
<td>✓</td>
</tr>
<tr>
<td>console</td>
<td>✓</td>
</tr>
<tr>
<td>destroy</td>
<td>✓</td>
</tr>
<tr>
<td>fmt</td>
<td>✓</td>
</tr>
<tr>
<td>get</td>
<td>✓</td>
</tr>
<tr>
<td>graph</td>
<td>✓</td>
</tr>
<tr>
<td>import</td>
<td>✓</td>
</tr>
<tr>
<td>init</td>
<td>✓</td>
</tr>
<tr>
<td>output</td>
<td>✓</td>
</tr>
<tr>
<td>plan</td>
<td>✓</td>
</tr>
<tr>
<td>providers</td>
<td>✓</td>
</tr>
<tr>
<td>show</td>
<td>✓</td>
</tr>
<tr>
<td>state</td>
<td>✓</td>
</tr>
<tr>
<td>taint</td>
<td>✓</td>
</tr>
<tr>
<td>untaint</td>
<td>✓</td>
</tr>
<tr>
<td>validate</td>
<td>✓</td>
</tr>
<tr>
<td>version</td>
<td>✓</td>
</tr>
<tr>
<td>workspace</td>
<td>(Not yet implemented)</td>
</tr>
</tbody>
</table>

Organizations and Workspaces

A Terraform organization in Scalr is represented by “IaCP” environment. An environment can contain multiple workspaces to support VCS automation, remote backend and service catalog based runs.

Note: Workspaces will be created automatically in IaCP the first time `terraform init` is run. All other workspace management operations must currently be done in the UI.

Module Private Registry

Overview

Terraforms powerful system enables simple re-use of common Terraform “building blocks” in multiple templates. A module is basically a Terraform template that can be called from other templates in order to manage the state of the resources defined in the module. A module can contain all the same types of constructs such as resources, data and variables as other templates, and can in turn call other modules as well.

Modern organizations typically need to create and share a library of common modules that will deploy resources in line with standards and configuration requirements. Scalr makes it possible to share the this library of modules securely within an organization by providing a private module registry. The registry is a catalog of releases from git repositories that provides the ability to search and filter items and review the documentation relating to inputs, outputs and examples of module usage in templates.
Note: A git release is a tagged version of the code. Once created a release never changes as it is pinned to a specific commit. Tags must use semantic version numbering (m.m.p, e.g. v1.0.4, 0.7.3) and can be created via the CLI (see `git tag`) or in the git VCS console via the “releases” page.

Modules in the registry are automatically pulled into workspaces where they are needed and the registration process automatically creates internal references to the module to be used in the template.

**Module Repositories**

Each module must be in its own repository and all the `.tf` files must be in the root directory. The release tag must be applied to the Master branch.

**Warning:** You cannot put multiple modules in the same repository or use sub-directories within the repository.

This is necessary because the key purpose of the registry is to provide version control and sharing of individual modules. Versioning is done by creating tagged releases of the repository. This approach is in-line with standard practice for managing modules and it provides the following benefits.

- Enables version management of individual modules
- Allows impact assessment for module changes
- Enforces the convention for module naming that requires a 1:1 mapping between repositories and modules

**Publishing Modules**

1. Set up a connection to your VCS provider as described in *VCS Integration*.
2. Ensure your module repository is correctly named

**Note:** Module repositories must be named using the format `terraform-<provider_name>-<module_name>`

3. Create a tagged release of the module in the VCS console or by using `git tag`

**Note:** The release tag MUST be on the Master branch.
4. At environment scope go to modules.

5. Click on New Module and select the VCS provider and module to be added.

6. The resulting screen will show the boiler plate code with the internal source reference that can be used to call the module from any template in this environment.
Example

Below is an example of a module and the module call from the private registry.

Module

- Does not require a provider block as this can be inherited from the calling templates
- Variables become input parameters to the module call below. Variables with defaults must be specified in the module call.

```hcl
resource "aws_instance" "scalr" {  
  ami = var.ami  
  instance_type = var.instance_type  
  subnet_id = var.subnet  
  vpc_security_group_ids = var.sg  
  key_name = var.key  
  count = var.instance_count  
}

variable "region" {  
  description = "Region"  
}

variable "instance_type" {  
  description = "Instance Type"  
}

variable "instance_count" {  
  description = "Number of instances to deploy"  
  default = "1"  
}

variable "subnet" {  
  description = "Subnet ID"  
}
```

(continues on next page)
variable "sg" {
    description = "AWS Security Group"
    type = list(string)
}

variable "key" {
    description = "AWS Key"
}

variable "vpc_id" {
    description = "VPC"
}

variable "ami" {
    description = "AMI"
}

Template with module call

- Source format is host / organization / module-name / provider.
- Organization is the same as specified in the terraform block. This maps to the environment in Scalr.
- Version and input variables from the module definition must be included.

terraform {
    backend "remote" {
        hostname = "my.scalr.com"
        organization = "org-sfgari365m7sck0"
        workspaces {
            name = "module-test"
        }
    }
}

provider "aws" {
    region = var.region
}

module "instance" {
    source = "my.scalr.com/org-sfgari365m7sck0/instance/aws"
    version = "1.0.1"
    instance_type = var.instance_type
    instance_count = var.instance_count
    subnet = var.subnet
    sg = var.security_group
    key = var.ssh_key
    vpc_id = var.vpc_id
    ami = var.ami
}

Note: Modules in the registry can be called from any workspace in Scalr that the user has access to. Your workspace does not have to be in the same environment that the module is registered in.
Import State

Overview

A lot of Terraform users will already be using Terraform locally and/or storing state in a remote backend such as S3. As your usage Terraform scales up you will see the need to take advantage of the features Scalr as a remote backend for operations and state.

As such, you will need to come up with plan to either import the Terraform state from your local machine or migrate the state from a other remote backends. Both can easily be done with Scalr using the Terraform CLI.

Get Existing State

The first step is to identify the location of the `terraform.tfstate` that you want to import, or pull the state from a remote backend using `terraform state pull > terraform.tfstate`.

Import State Files

To import the state into Scalr, only take a few easy steps:

- Create a local working directory
- **Setup the Terraform CLI if you haven’t done so already.**
- Copy the `terraform.tfstate` file and it’s associated template files (`*.tf`) into your working directory.
- Update the Terraform template to include Scalr as the remote backend. You can get the organization id from the environment switcher on the UI

```terraform
terraform {
backend "remote" {
    hostname = "your.scalr.com"
    organization = "<organization-id>"
    workspaces {
        name = "<workspace-name>"
    }
}
}
```

- Run `terraform init` and the workspace will be created with the latest state file.
Cost Estimation

Scalr IaCP provides cost estimation during a Terraform plan for resources that are being deployed. The cost estimation data is displayed in the Scalr UI and in CLI output.

The data provided by cost estimation is also available for use in policies implemented in Open Policy Agent (OPA).

Note: Cost estimation is currently only provided for a limited number of resources in AWS and GCP. We are adding new resources for these and other clouds, such as Azure, with each release. Please bookmark this page to stay updated with the changes.

Below are lists of resources for which cost estimation is currently implemented. Each list covers one logical grouping for each cloud provider and includes information on the factors affecting cost.

Warning: IaCP does not currently estimate cost for all resources and does not include estimates for variable items such as network bandwidth, disk I/O’s etc. You will very likely incur more cost than is shown in the cost estimates provided by IaCP.

AWS Resources

EC2

aws_instance

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td></td>
</tr>
<tr>
<td>InstanceType</td>
<td></td>
</tr>
<tr>
<td>TermType</td>
<td>Currently only On-Demand instances</td>
</tr>
<tr>
<td>Usagetype</td>
<td>BoxUsage</td>
</tr>
<tr>
<td>OperatingSystem</td>
<td>Windows or Linux</td>
</tr>
<tr>
<td>LicenseModel</td>
<td>Currently only instances that do not require a license are taken into account</td>
</tr>
<tr>
<td>PreInstalledSw</td>
<td>Currently any additional software price not taken into account</td>
</tr>
</tbody>
</table>
**aws_ebs_volume**

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Based on availability zone</td>
</tr>
<tr>
<td>Volume size</td>
<td></td>
</tr>
<tr>
<td>Volume type</td>
<td>Magnetic, Cold HDD, General Purpose, Provisioned IOPS, Throughput Optimized HDD</td>
</tr>
<tr>
<td>TermType</td>
<td>Currently only On-Demand volumes</td>
</tr>
</tbody>
</table>

**RDS**

**aws_db_instance**

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td></td>
</tr>
<tr>
<td>Instance type</td>
<td>Instance class</td>
</tr>
<tr>
<td>TermType</td>
<td>Currently only On-Demand instances</td>
</tr>
<tr>
<td>Engine</td>
<td>mysql, mariadb, postgresql, sql server, oracle</td>
</tr>
<tr>
<td>Storage</td>
<td>Estimate for classic RDS database instances only. Aurora does not provide information about storage. Factors: Size, Type, Provisioned IOPS (for io1 type)</td>
</tr>
</tbody>
</table>

**aws_rds_cluster / aws_rds_global_cluster**

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws_db_instance</td>
<td>Estimate based on cost of the number aws_db_instance in the cluster. For Aurora cluster this includes cost for both primary and secondary region.</td>
</tr>
</tbody>
</table>

**aws_db_snapshot**

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Taken from aws_db_instance</td>
</tr>
<tr>
<td>Storage Size</td>
<td>Taken from aws_db_instance</td>
</tr>
</tbody>
</table>

**Note:** Database cluster snapshot price cannot be estimated, because linked RDS cluster size not known at the time of snapshot creation
EKS

aws_eks_cluster

- List price of EKS control plane. No other factors

aws_autoscaling_group

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node cost</td>
<td>See <code>aws_instance</code>. Cost calculated based on instances specified in launch_configuration, launch_template, or mixed_instances_policy and multiplies.</td>
</tr>
<tr>
<td>Capacity</td>
<td>if set then <code>aws_autoscaling_group.desired_capacity</code>, otherwise default capacity taken from <code>aws_autoscaling_group.min_size</code></td>
</tr>
</tbody>
</table>

GCP Resources

Compute Instance

google_compute_instance

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td></td>
</tr>
<tr>
<td>MachineType</td>
<td></td>
</tr>
<tr>
<td>Usagetype</td>
<td></td>
</tr>
<tr>
<td>Usagetype</td>
<td></td>
</tr>
<tr>
<td>ScratchDisk/BootDisk</td>
<td></td>
</tr>
<tr>
<td>Image</td>
<td></td>
</tr>
<tr>
<td>PreInstalledSw</td>
<td>Currently any additional software price not taken into account</td>
</tr>
</tbody>
</table>

Storage

google_compute_disk

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td></td>
</tr>
<tr>
<td>DiskType</td>
<td></td>
</tr>
<tr>
<td>DiskSize</td>
<td></td>
</tr>
<tr>
<td>Image</td>
<td></td>
</tr>
</tbody>
</table>
Container Clusters

google_container_cluster

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeCount</td>
<td></td>
</tr>
<tr>
<td>NodeSettings</td>
<td></td>
</tr>
</tbody>
</table>

google_container_node_pool

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeInstanceSettings</td>
<td></td>
</tr>
<tr>
<td>NodeDiskSettings</td>
<td></td>
</tr>
<tr>
<td>NodeCount</td>
<td></td>
</tr>
</tbody>
</table>

Database

google_sql_database_instance

<table>
<thead>
<tr>
<th>Cost Factors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceSettings</td>
<td></td>
</tr>
<tr>
<td>DiskSettings</td>
<td></td>
</tr>
<tr>
<td>IpConfiguration</td>
<td></td>
</tr>
<tr>
<td>BackupConfiguration</td>
<td></td>
</tr>
</tbody>
</table>

Examples Templates and Service Catalog Offerings

Below is a list of example templates fully configured to work as Terraform Service Catalog Tutorial offerings, including scalr-module.hcl files. There is a brief description of each template and a link to the Github repo.

Instances

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ec2_instance</td>
<td>This will deploy an EC2 instance from a completely variablized Terraform template with all of the inputs coming from inputs filled in by the Scalr service catalog and controlled by policy.</td>
</tr>
<tr>
<td>ec2_instance_tags</td>
<td>As above with Tags.</td>
</tr>
<tr>
<td>ec2_instance_ansible</td>
<td>Template to create an ec2 instance and bootstrap with Ansible.</td>
</tr>
<tr>
<td>gcp_instance</td>
<td>This template will create a basic GCE instance.</td>
</tr>
</tbody>
</table>
Kubernetes Clusters

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws_eks_cluster</td>
<td>Creates and EKS Cluster in the given Region.</td>
</tr>
<tr>
<td>gce-gke-cluster</td>
<td>Creates and GKE Cluster in the given Region.</td>
</tr>
</tbody>
</table>

Pods on Kubernetes

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupal_eks</td>
<td>Drupal Pod backed by RDS MySQL database.</td>
</tr>
<tr>
<td>drupal-gke</td>
<td>Drupal Pod backed by GCE SQL MySQL database.</td>
</tr>
<tr>
<td>wordpress_eks</td>
<td>Wordpress Pod backed by RDS MySQL database.</td>
</tr>
<tr>
<td>wordpress-gke</td>
<td>Wordpress Pod backed by GCE SQL MySQL database.</td>
</tr>
</tbody>
</table>

Object Storage

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s3</td>
<td>This template will create an S3 bucket.</td>
</tr>
<tr>
<td>gcp_bucket</td>
<td>This will create a GCP bucket.</td>
</tr>
</tbody>
</table>

AWS Misc

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>security_group</td>
<td>This template will create a security group with https open inbound.</td>
</tr>
</tbody>
</table>

Azure Misc

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>azure_sql_database</td>
<td>Create a simple Azure SQL database</td>
</tr>
<tr>
<td>azure_attach_storage</td>
<td>This template will provision a new Azure disk and attach it to the specified server.</td>
</tr>
<tr>
<td>azure_resource_group</td>
<td>This will create an Azure Resource Group.</td>
</tr>
</tbody>
</table>

Scalr Installation

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalr-install</td>
<td>Production Scalr install with 6 Servers and HA MySQL plus Loadbalancer.</td>
</tr>
<tr>
<td>scalr-install-rds</td>
<td>Production Scalr install with 4 Servers and RDS based MySQL plus Loadbalancer.</td>
</tr>
</tbody>
</table>

1.3. Scalr Features
OPA Policy Examples

<table>
<thead>
<tr>
<th>Repo</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policies</td>
<td>Some example OPA Policies</td>
</tr>
</tbody>
</table>

Account & Environment Configuration

An account is a child of the Scalr scope as it will inherit anything that is placed or assigned to it at the Scalr scope and it is a parent to environments, which is one of the core concepts behind Scalr that makes it a truly multi-tenant tool. Each account has an owner. A Scalr instance may contain as many accounts as you wish. Accounts contain one or more environments. Generally, accounts correspond to organizations, business units, customers, or just a logical way to segment resources and users.

Creating Accounts

Note: The Scalr scope only exists in the on prem version of Scalr IaC Platform

Accounts can only be created by a Global Admin. To create an account, first ensure you are logged into the Scalr scope as a Global Administrator. Next, click on the main Scalr menu on the top left, and go down to Accounts.

Click the New Account button, which will prompt you for a Name, Owner, and Cost Center, all mandatory fields:
After creating an account, you can move on to creating the environments within the account.

Creating Environments

An environment is a child of an account and will inherit anything that is placed or assigned to it at the account level. Each environment is connected to one or more cloud platforms. One or more teams can be granted access to each environment. Environments can correspond to

- An SDLC environment: Dev, QA, or Prod
- An application
- A logical group of physical resources: Non-Prod and Prod
- Or More

There isn’t a “rule” for what an environment can be, it is just a way to add more multi-tenancy and logical groupings to Scalr.

Environments can only be created by account administrators. To create an environment, first ensure you are logged into the account scope. Next, click on the main Scalr menu on the top left, and go down to environments. Click on New environment -> IaCP, enter the Name, which can be changed at a later time, and Save.
Configuring Environments

To get working with Scalr you must set up environments where you workspaces and service catalog will be configure and used.

You must follow these steps to initially configure at least one environment. Make sure you are logged in to the account scope (Green), if you are not, click on the environment name and then click on the account:
<table>
<thead>
<tr>
<th>Action</th>
<th>Menu</th>
<th>Steps</th>
<th>Example</th>
<th>More Info</th>
</tr>
</thead>
</table>
| **Add Cloud Credentials** | ![Cloud Credentials](image) | - Set up access in cloud provider  
- In Scalr “Add Credentials”  
- Pick the cloud type, complete form and save. | ![Cloud Credentials](image) | **Terraform Provider Authentication** |
| **Create Teams**       | ![Teams](image) | - Click “New Team”  
- Give the team a name.  
- Set Default ACL to “Full access (No Admin)”  
(NOTE: ACL’s are not yet implemented in Scalr IaC Platform)  
- Add your user to the Team  
- SAVE | ![Teams](image) | |
| **Create Users**       | ![Users](image) | - Click “New User”  
- Enter their emails  
- Click “Add to Team” and select the team created previously  
- Add and SAVE | ![Users](image) | |
| **Create IaC Environment** | ![Environments](image) | - New Environment -  
> IaC  
- Enter a Name  
- Link the Cloud Credentials (Click on the link symbol)  
- Click on Access tab and Grant access to your teams  
- SAVE | ![Environments](image) | |
Policies and Environments

Policies can be linked to environments to apply standards and governance to Terraform deployments. There are two policy options and they can be used concurrently in any environment.

1. Scalr Policies to apply governance to values entered in Service Catalog input prompts. See Input Policy for more details
2. Open Policy Agent (OPA) policies to apply policy rules to all deployments. Policies are checked during the Plan phase of every Terraform run. See Open Policy Agent (OPA).

Cloud Credentials

This page explain how to set up credentials in Scalr for enabling Terraform Providers to access the clouds.
Terraform Providers require some parameters to authenticate against the cloud API. These can be provided directly in the provider block like this.

```hcl
provider "aws" {
  access_key = var.scalr_aws_access_key
  secret_key = var.scalr_aws_secret_key
  region = var.region
}
```

Or they can be provided via Environment Variables that are declared in the environment where the Terrafrom run takes place. For example,

```bash
export AWS_ACCESS_KEY_ID=[ACCESS_KEY_VALUE]
export AWS_SECRET_ACCESS_KEY=[SECRET_KEY_VALUE]
```

Then your template would simply contain a provider block like this.

```hcl
provider "aws" {
  region = var.region
}
```

You can set up “Cloud Credentials” in Scalr so that they can be used to securely publish the access_keys and other cloud API authentication parameters to environments as Global Variables. See Terraform Provider Authentication for more details.

Setting up cloud credentials in is a three step process.

1. Set up necessary authentication in the cloud provider (varies from cloud to cloud).
2. Create Cloud Credentials in Scalr.
3. Link Cloud Credentials to the required environments.

Environments can be linked to multiple clouds which enables DevOps to implement distributed deployments, and/or provide users with the option to choose a cloud when using Self Service deployment through the Service Catalog.

Cloud credentials in Scalr can be configured globally at the Scalr scope for all accounts and environments or at the account scope so they are restricted to the environments within that account.

From the main menu select “Cloud Credentials”.

For details on configuring Scalr for the 3 main cloud provider, please follow the appropriate link(s) below. For other clouds the follow the appropriate documentation to create users/keys.

- AWS Access Keys
• Google Cloud Platform Access Keys
• Azure App Registration
• VMware Credentials

Linking Cloud Credentials to environments

• Navigate to environments
• Select the required environment and click on the Clouds tab.
• Click the link symbol on the right side for the cloud you want to link.
• Select the credentials and save.

AWS Access Keys

For AWS you will need to generate Access keys for the IAM user you want to use along with a suitable Role to grant permissions.

If you need assistance creating an AWS user and the required policy please refer to

• In the EC2 console navigate to IAM -> Users -> Select a user -> Security Credentials -> Create Access Key
• Save the Access Key and Secret Key
• In Scalr in the “Add Credentials” dialogue select AWS, Enter a name and the two keys and save.
Google Cloud Platform Access Keys

For GCP you will need to create a Service Account access key within your project. This most easily done as JSON file which can be uploaded to Scalr.

- In the Google console navigate to API & Services -> Credentials -> Service Account Key.
- Choose JSON and Save.
- In Scalr in the “Add Credentials” dialogue select Google -> Set project name, select Upload Json Key, then upload the file and Save.
1.3. Scalr Features
Create service account key

Service account
- Compute Engine default service account

Key type
- JSON (Recommended)
- P12 (For backward compatibility with code using the P12 format)

Getting started with Scalr Google Compute Engine tutorial.

ADD CREDENTIALS

CLOUD
- Google Compute Engine

NAME
- GCE-test

CONFIG TYPE
- MANUAL
- UPLOAD JSON KEY

PROJECT ID
- scalr-demo

JSON KEY
- C:\fakepath\scalr-demo-7258ab695813.json

GOOGLE BETA API
- Enable cross-project networks (XPN) support
Azure App Registration

For Azure you need to create an authorized App Registration in Azure AD.

- In Scalr in the “Add Credentials” dialogue select Azure
- Enter the Client ID, Secret Key and Tenant ID. Click Continue
- Select the Subscription ID and Save
VMware Credentials

For VMware you need to create a username and password in vSphere.

- In Scalr in the “Add Credentials” dialogue select VMware vSphere
- Enter the URL, username, password, and toggle on/off the SSL verification. Click save.

Users and Teams

Overview

To gain access to Scalr, a user must first be added and linked to a team. Teams are given access to environments, not individual users (except for admins). There are two types of users and teams:

- Local - All users and teams are stored, updated, and managed in Scalr.
- External - All users and team are stored and managed in an external system like AD or LDAP. To find out more about AD or LDAP integration, please go to the ldap page.

As seen in the diagram below, user and team management happens at the account scope and the team is then applied to an environment. A team can have access to one more accounts and environments.
Configuring Users

Scalr Local Users

To add an end user to a Scalr account:

- Ensure you are logged into the account scope as either an administrator or as the account owner.
- Click on the main menu on the top left.
- Click on new user:
  - On the right hand side of the users panel, you will see an entry section for new user. Enter the user’s email address.
  - Click add to team to add the new user to a selected team if desired, if the team is already created.
• The user will receive an email letting them know they have access and they can set their password at the first login.

Note: The ability to apply an ACL to control user permissions will be in a future release.

Externally Authenticated Users

If you are using an external system to manage your teams, you do not have to manually add users. The users will gain access to Scalr when their AD/LDAP team is added into Scalr. The user will appear after that user logs in for the first time. See more about adding teams below.

Configuring Teams

Scalr Local Teams

To add a team to a Scalr account:

• Ensure you are logged into the account scope as either an administrator or as the account owner.

• click on the main Scalr menu on the top left, and go down to teams.
• Enter a name for your team.

• If existing users are present, click on the plus sign next to a member in the table to attach them to the team. Otherwise, you will need to add a user before members can be attached to your new team.

Externally Authenticated Teams

To add an externally managed team:

• Ensure you are logged into the account scope as either an administrator or as the account owner.

• Click on the main menu on the top left, and go down to teams.

• Add the team name (LDAP name, group id, etc - this is based on your LDAP settings) into the new team field seen above. On save, Scalr will verify the team with LDAP/AD and if no errors are received it will be added to Scalr. As explained above, users will automatically be added to Scalr after they log in for the first time.

Note: The ability to apply an ACL to control team permissions will be in a future release.

Adding Teams to Environments

Now that you have created teams, they must be added to an environment for their users to be able to do anything. If you add a team, but do not add that team to an environment then the users will receive an error when they try to log in. To add a team to an environment:

• Ensure you are logged into the account scope as either an administrator or as the account owner.

• Click on the main menu on the top left, and go down to environments.

• Click on the environments tab and then the environment you would like to add the team to. The currently selected environment will appear in light gray, as shown below by the “Dev” environment selection below:
• In the grant access panel, tick the checkbox next to the desired team and then grant access. You may also add teams here.

• Click save and the team will now have access to the environment that it was added to.

Global Admins

Note: The global admin role only exists in the on-prem version of Scalr.

A global admin is generally someone who manages Scalr in general. Out of the box, Scalr has one global admin named “admin”, this user can add more admins if needed. To add a global admin, go to the Scalr scope and click on Users. Once you are on the users page, click on new user, enter the information that is prompted for, and then make sure you tick the global admin radio button:
Scalr IaCP Installation & Configuration

This section is for on prem customers who are installing and configuring the Scalr IaCP.

Scalr IaCP Install - One Server

Note: This guide should only be used for POCs or Development environments. Contact Scalr for a production deployment recommendation.

Prerequisites

- A single Ubuntu 16.04 or RHEL/CentOS 7.x server with 4CPU x 16GB RAM and 50GB storage
- Scalr download token
- Scalr license file
- SSL Cert (can be self signed)
- Internet Connectivity
- A domain name, the URL should not resolve to an IP
Installation

Get the scalr-server package:

```
# For Debian:
curl -s https://<token>:@packagecloud.io/install/repositories/scalr/scalr-server-ee-
---staging/script.deb.sh | sudo bash
# For RPM:
curl -s https://<token>:@packagecloud.io/install/repositories/scalr/scalr-server-ee-
---staging/script.rpm.sh | sudo bash
```

Install the package:

```
# For Debian:
apt-get install scalr-server
# For RPM:
yum install scalr-server
```

Run the following when prompted:

```
scalr-server-wizard
```

The step above did two things, created the /etc/scalr-server directory and the scalr-server-secrets.json file.

Please add the license file to the /etc/scalr-server directory

```
##Paste the license.json file in the following location on each server:##
vi /etc/scalr-server/license.json
```

Update /etc/scalr-server/scalr-server.rb with the following contents. Be sure to update the values with the actual values for you installation:

```
enable_all true
product_mode :iacp

# Mandatory SSL
# Update the below settings to match your FQDN and where your .key and .crt are stored
proxy[:ssl_enable] = true
proxy[:ssl_redirect] = true
proxy[:ssl_cert_path] = "/etc/scalr-server/organization.crt"
proxy[:ssl_key_path] = "/etc/scalr-server/organization.key"
routing[:endpoint_host] = "iacp.organization.com"
routing[:endpoint_scheme] = "https"

#Add if you have a self signed cert, update with the proper location if needed
#ssl[:extra_ca_file] = "/etc/scalr-server/rootCA.pem"

#Add if you require a proxy, it will be used for http and https requests
#http_proxy: "http://user:*****@my.proxy.com:8080"
#no_proxy: "example.com"
```

Reconfigure the Scalr server:

```
/opt/scalr-server/bin/scalr-server-ctl reconfigure
```

You can now log into Scalr by putting the hostname that is listed as your endpoint in the scalr-server.rb into a browser. To log in the first time, please find the admin password in the /etc/scalr-server/scalr-server-secrets.json file. The username is admin.
External Authentication

SAML Integration

Integration with Scalr

When SAML is activated, it will move the authentication step outside of Scalr and hand it over to the SAML server that you have configured. During sign in to Scalr, the user will be transferred to a sign in page provided by the SAML server. After sign in, the user will then be redirected back to Scalr which will subsequently treat the user as signed in. Each account in Scalr can have a separate SAML provider if your organization requires it.

Configuration

To enable SAML, you must configure it at the global scope of the Scalr UI. To do this:

1. Log in as a global admin, click on the Scalr icon on the top left and go to IAM -> identity providers:

![Scalr UI IAM Identity Providers](image)

2. Note down the SP endpoint as that will be used with the SAML provider. For example: https://scalr.server/public/saml/idp-sp986542njcvhjv78?metadata and https://scalr.server/public/saml/idp-sp986542njcvhjv78?acs

3. Fill in the fields required by your SAML provider.

4. After you save the SAML configuration, link the provider to an account:
Note: In the event you need to log in with a local administrator, add the following to the end of your Scalr url to avoid the SAML login screen: #?no-login-redirect (https://your.scalrserver.com/#?no-login-redirect)

Okta Example

Note: Scalr supports all SAML providers, this is just an example of a commonly used one.

1. Go to Okta’s administration interface by pressing the “Admin” button within the Okta UI:

2. Select Applications > Applications from the toolbar:

3. Select “Add Application”:
4. Select “Create New App”:

5. Select SAML 2.0 in the “Create a new application integration” dialogue message:

6. Enter the “App name”, then select Next:
7. Configure SAML settings. Fill in the form as shown in the example below. Be sure to swap out the example URL host with your own Scalr server. Ensure the “Group Attribute Statements (Optional)” field is populated in order for the user groups to be sent to Scalr correctly. It’s extremely important to correctly enter the “Attribute name”, “Format”, and “Filter” option as shown below:
8. Finishing SAML integration:

<table>
<thead>
<tr>
<th>Name</th>
<th>Name format (optional)</th>
<th>Value</th>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groups</td>
<td>Basic</td>
<td>Matches regex</td>
<td>.*</td>
</tr>
</tbody>
</table>
9. When complete, you will be forwarded to a Sign On page where the SAML Service Provider configuration options link can be found. Click “View Setup Instructions” to see details:

10. Enter the following settings in to the SAML setup in the Scalr UI. Be sure to use the ID and URL that you obtained previously:

   - idp entity_id => http://www.okta.com/exk8ldhenajn8ckSdgh6
   - idp single_sign_on_service_url => https://scalr.oktapreview.com/app/scalrincdev369882_scalr_1/exk8ldhenajn8ckS40h7/sso/saml
   - idp x509cert => —–Cert Goes Here—–

11. Everything else can be left as the default setting. If there is more than one authentication method, users will be
prompted to select their method when logging into Scalr:

Users and Teams with SAML

After Scalr has been reconfigured for SAML, users and teams work as follows.

1. Teams map to AD/LDAP groups. Account admins still have to create teams in Scalr but the team name will be validated against the groups in AD/LDAP, so the team name must match the group name.

2. Teams must linked to at least one environment

3. Once a team has been linked, any member of the related LDAP group can attempt to login to Scalr. On first login a user record gets created in Scalr and set as LDAP authenticated.

4. Scalr admin can still create Scalr authenticated users to act as global and account admins. SAML authenticated users can also be set as global and account admins.

5. If a user has access to more than one account, they will be prompted to select an account during login.

LDAP Integration

Integration with Scalr

When LDAP is activated, it will move the authentication and user management outside of Scalr and hand it over to the LDAP server that you have configured. Each account in Scalr can have a separate LDAP provider if your organization requires it.

Configuration

To enable LDAP, you must configure it at the global scope of the Scalr UI. To do this:

1. Log in as a global admin, click on the Scalr icon on the top left and go to IAM -> identity providers:
2. Fill in the fields required by your LDAP server.

3. After you save the LDAP configuration, link the provider to an account:

Users and Teams with LDAP

After Scalr has been reconfigured for LDAP, users and teams work as follows.

1. Teams map to AD/LDAP groups. Account admins still have to create teams in Scalr but the team name will be validated against the groups in AD/LDAP, so the team name must match the group name.

2. Teams must be linked to at least one environment

3. Once a team has been linked, any member of the related LDAP group can attempt to login to Scalr. On first login a user record gets created in Scalr and set as LDAP authenticated.

4. Scalr admin can still create Scalr authenticated users to act as global and account admins. SAML authenticated users can also be set as global and account admins.

5. If a user has access to more than one account, they will be prompted to select an account during login.
Getting started

To start, an API token must be generated by clicking on your user on the top right of the screen and clicking on “TF API Access”:

This will provide you with an ID and token. The token will disappear after you click close. A user can have as many tokens as they want, it is up to that user to manage the tokens.

Authentication

All requests must be authenticated with a bearer token. Use the HTTP header Authorization with the value Bearer <token>. You can generate access token.

If the token is absent or invalid, Scalr responds with HTTP status 401 and a JSON API error object. The 401 status code is reserved for problems with the authentication token; forbidden requests with a valid token result in a 404.

Preview

applies

get_apply

GET /api/iacp/v3/applies/{apply}
Show details of a specific apply.

Parameters

• apply(string) –

Example request:

```
GET /api/iacp/v3/applies/{apply} HTTP/1.1
Host: example.com
```

Status Codes

• 200 OK – Example response:

```json
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json
{
  "data": {
    "attributes": {
      "log-read-url": "string",
      "resource-additions": 1,
      "resource-changes": 1,
      "resource-destructions": 1,
      "status": "pending"
    }
  }
}
```

(continues on next page)
configuration-versions

get_configuration_version

GET /api/iacp/v3/configuration-versions/{configuration_version}

Return the Configuration Version.

Parameters

- configuration_version(string)

Example request:

```plaintext
GET /api/iacp/v3/configuration-versions/{configuration_version} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – Example response:

```plaintext
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "auto-queue-runs": true,
      "created-at": "2020-03-30T16:10:27.586407",
      "download-url": "string",
      "error": "string",
      "error-message": "string",
      "source": "tfe-api",
      "speculative": true,
      "status-timestamps": {}
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {}
  },
  "type": "applies"
}
```
"status": "errored",
"status-timestamps": {},
"upload-url": "string"
},
"id": "string",
"links": {
  "self": "string"
},
"relationships": {
  "ingress-attributes": {
    "data": {
      "id": "string",
      "type": "ingress-attributes"
    }
  },
  "workspace": {
    "data": {
      "id": "string",
      "type": "workspaces"
    }
  }
},
"type": "configuration-versions"
},
"meta": {}
}

• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –

get_ingress_attributes

GET /api/iacp/v3/configuration-versions/{configuration_version}/ingress-attributes
Return Ingress Attributes for the Configuration Version.

Parameters
  • configuration_version(string) –

Query Parameters
  • page[number](string) – Page number
  • page[size](string) – Page size

Example request:

```
GET /api/iacp/v3/configuration-versions/{configuration_version}/ingress-attributes
Host: example.com
```
Status Codes

• 200 OK – Example response:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json
{
  "data": {
    "attributes": {
      "branch": "string",
      "commit-message": "string",
      "commit-sha": "string",
      "sender-username": "string"
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {},
    "type": "ingress-attributes"
  },
  "meta": {}
}
```

• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –

get_workspace_configuration_versions

GET /api/iacp/v3/workspaces/{workspace}/configuration-versions
List Configuration Version’s for a specific workspace.

Parameters

• **workspace** *(string)*

Query Parameters

• **page[number]** *(string)* – Page number
• **page[size]** *(string)* – Page size

Example request:

```
GET /api/iacp/v3/workspaces/{workspace}/configuration-versions HTTP/1.1
Host: example.com
```

Status Codes

• 200 OK – Example response:
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
   "data": {
      "attributes": {
         "auto-queue-runs": true,
         "created-at": "2020-03-30T16:10:27.586407",
         "download-url": "string",
         "error": "string",
         "error-message": "string",
         "source": "tfe-api",
         "speculative": true,
         "status": "errored",
         "status-timestamps": {},
         "upload-url": "string"
      },
      "id": "string",
      "links": {
         "self": "string"
      },
      "relationships": {
         "ingress-attributes": {
            "data": {
               "id": "string",
               "type": "ingress-attributes"
            }
         },
         "workspace": {
            "data": {
               "id": "string",
               "type": "workspaces"
            }
         }
      },
      "type": "configuration-versions"
   },
   "meta": {}  
}

• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –
create_configuration_version

POST /api/iacp/v3/workspaces/{workspace}/configuration-versions

Create a Configuration Version.

Parameters

• workspace (string)

Example request:

```json
POST /api/iacp/v3/workspaces/{workspace}/configuration-versions HTTP/1.1
Host: example.com
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "auto-queue-runs": true,
      "speculative": true
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "type": "configuration-versions"
  },
  "meta": {}
}
```

Status Codes

• 201 Created – Example response:

```json
HTTP/1.1 201 Created
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "auto-queue-runs": true,
      "created-at": "2020-03-30T16:10:27.586407",
      "download-url": "string",
      "error": "string",
      "error-message": "string",
      "source": "tfe-api",
      "speculative": true,
      "status": "errored",
      "status-timestamps": {},
      "upload-url": "string"
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {
      "ingress-attributes": {
        "data": {
(continues on next page)```
"id": "string",
"type": "ingress-attributes"
},
"workspace": {
"data": {
"id": "string",
"type": "workspaces"
}
},
"type": "configuration-versions"
},
"meta": {}
}

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

cost-estimates
get_cost_estimate

GET /api/iacp/v3/cost-estimates/{cost_estimate}
Show details of a specific cost estimation.

Parameters

- **cost_estimate**(string) –

Example request:

```
GET /api/iacp/v3/cost-estimates/{cost_estimate} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – Example response:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "delta-monthly-cost": "string",
      "error-message": "string",
      "matched-resources-count": 1,
```
"prior-monthly-cost": "string",
"proposed-monthly-cost": "string",
"resources": {},
"resources-count": 1,
"status": "pending",
"status-timestamps": {},
"unmatched-resources-count": 1
},
"id": "string",
"links": {
  "output": "string",
  "self": "string"
},
"relationships": {},
"type": "cost-estimates"
},
"meta": {}}

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

**get_log**

GET /api/iacp/v3/cost-estimates/{cost_estimate}/output
Generate URL to the location of the finished cost estimation stage logs.

Parameters

- **cost_estimate** *(string)*

Example request:

```text
GET /api/iacp/v3/cost-estimates/{cost_estimate}/output HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK –
- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –
organizations

get_organizations

GET /api/iacp/v3/organizations
List Organizations.

Query Parameters

• page[number] (string) – Page number
• page[size] (string) – Page size

Example request:

GET /api/iacp/v3/organizations HTTP/1.1
Host: example.com

Status Codes

• 201 Created –
• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –

get_organization

GET /api/iacp/v3/organizations/{organization}
Show an Organization.

Parameters

• organization (string) –

Example request:

GET /api/iacp/v3/organizations/{organization} HTTP/1.1
Host: example.com

Status Codes

• 201 Created – Example response:

```
HTTP/1.1 201 Created
Content-Type: application/vnd.api+json

{
    "data": {
        "attributes": {
            "cost-estimation-enabled": true,
```
modules

create_module

POST /api/iacp/v3/organizations/{organization}/registry-modules

Create a new registry module without a backing VCS repository.

Parameters

• organization (string) –

Example request:

```plaintext
POST /api/iacp/v3/organizations/{organization}/registry-modules HTTP/1.1
Host: example.com
Content-Type: application/vnd.api+json

{
    "data": {
        "attributes": {
            "name": "string",
            "provider": "string"
        },
        "id": "string",
        "links": {
            "self": "string"
        },
        "type": "registry-modules"
    }
}
```
Status Codes

- **201 Created** – Example response:

```json
HTTP/1.1 201 Created
Content-Type: application/vnd.api+json

{
   "data": {
      "attributes": {
         "created-at": "2020-03-30T16:10:27.586407",
         "name": "string",
         "permissions": {},
         "provider": "string",
         "status": "no_version_tags",
         "updated-at": "2020-03-30T16:10:27.586407",
         "version-statuses": [
            {}
         ]
      },
      "id": "string",
      "links": {
         "self": "string"
      },
      "relationships": {
         "organization": {
            "data": {
               "id": "string",
               "type": "organizations"
            }
         }
      },
      "type": "registry-modules"
   },
   "meta": {}
}
```

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –
publish_module

**POST /api/iacp/v3/registry-modules**

Publish a new registry module from a VCS repository.

Example request:

```json
POST /api/iacp/v3/registry-modules HTTP/1.1
Host: example.com
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "vcs-repo": {
        "display-identifier": "string",
        "identifier": "string",
        "oauth-token-id": "string"
      }
    },
    "id": "string",
    "links": {},
    "relationships": {
      "organization": {
        "data": {
          "id": "string",
          "type": "organizations"
        }
      }
    },
    "type": "registry-modules"
  },
  "meta": {}
}
```

**Status Codes**

- **201 Created** – Example response:

```json
HTTP/1.1 201 Created
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "created-at": "2020-03-30T16:10:27.586407",
      "name": "string",
      "permissions": {},
      "provider": "string",
      "status": "no_version_tags",
      "updated-at": "2020-03-30T16:10:27.586407",
      "version-statuses": [
        {}]
    },
    "id": "string",
    "links": {
      "self": "string"
    }
  }
}
```

(continues on next page)
{
    "relationships": {
        "organization": {
            "data": {
                "id": "string",
                "type": "organizations"
            }
        }
    },
    "type": "registry-modules"
},
"meta": {}}

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

delete_module

POST /api/iacp/v3/registry-modules/actions/delete/{organization}/{module_name}
Deletes terraform Module.

Parameters
- organization (string) –
- module_name (string) –

Status Codes
- 204 No Content –
- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –
delete_module_provider

POST /api/iacp/v3/registry-modules/actions/delete/{organization}/{module_name}/{provider_name}
Deletes terraform Provider of the specific Module.

Parameters

• organization (string) –
• module_name (string) –
• provider_name (string) –

Status Codes

• 204 No Content –
• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –

delete_module_version

POST /api/iacp/v3/registry-modules/actions/delete/{organization}/{module_name}/{provider_name}/{version}
Delete terraform Module Version.

Parameters

• organization (string) –
• module_name (string) –
• provider_name (string) –
• version (string) –

Status Codes

• 204 No Content –
• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –
get_module

GET /api/iacp/v3/registry-modules/{module}
Return the Module.

Parameters
• module (string) –

Example request:

GET /api/iacp/v3/registry-modules/{module} HTTP/1.1
Host: example.com

Status Codes
• 200 OK – Example response:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "created-at": "2020-03-30T16:10:27.586407",
      "name": "string",
      "permissions": {},
      "provider": "string",
      "status": "no_version_tags",
      "updated-at": "2020-03-30T16:10:27.586407",
      "version-statuses": [
        {}
      ],
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {
      "organization": {
        "data": {
          "id": "string",
          "type": "organizations"
        }
      }
    },
    "type": "registry-modules"
  },
  "meta": {}
}
```

• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
422 Unprocessable Entity –

**get_module_by_name_and_provider**

GET /api/iacp/v3/registry-modules/{organization}/{module_name}/{module_provider}

Return the *Module* by name and provider.

**Parameters**

- **organization** *(string)*
- **module_name** *(string)*
- **module_provider** *(string)*

**Example request:**

```
GET /api/iacp/v3/registry-modules/{organization}/{module_name}/{module_provider}
HTTP/1.1
Host: example.com
```

**Status Codes**

- **200 OK** – Example response:

```python
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "created-at": "2020-03-30T16:10:27.586407",
      "name": "string",
      "permissions": {},
      "provider": "string",
      "status": "no_version_tags",
      "updated-at": "2020-03-30T16:10:27.586407",
      "version-statuses": [
        {}
      ]
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {
      "organization": {
        "data": {
          "id": "string",
          "type": "organizations"
        }
      },
      "type": "registry-modules"
    },
    "meta": {}
  }
}
```

- **400 Bad Request** –
POST /api/iacp/v3/registry-modules/{organization}/{module_name}/{module_provider}/resync
Trigger resync of the Registry Module associated with VCS repository.

Parameters

- organization (string)
- module_name (string)
- module_provider (string)

Status Codes

- 204 No Content
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 409 Conflict
- 422 Unprocessable Entity

POST /api/iacp/v3/registry-modules/{organization}/{module_name}/{module_provider}/versions
Create a new registry module version.

Parameters

- organization (string)
- module_name (string)
- module_provider (string)

Example request:

```http
POST /api/iacp/v3/registry-modules/{organization}/{module_name}/{module_provider}/versions HTTP/1.1
Host: example.com
Content-Type: application/vnd.api+json

{
    "data": {
        "attributes": {
            "version": "string"
        }
    }
}
```

(continues on next page)
```

```

### Status Codes

- **201 Created** – Example response:

```text
HTTP/1.1 201 Created
Content-Type: application/vnd.api+json

{
    "data": {
        "attributes": {
            "created-at": "2020-03-30T16:10:27.586407",
            "source": "string",
            "status": "pending",
            "updated-at": "2020-03-30T16:10:27.586407",
            "version": "string"
        },
        "id": "string",
        "links": {
            "upload": "string"
        },
        "relationships": {
            "registry-module": {
                "data": {
                    "id": "string",
                    "type": "registry-modules"
                }
            }
        },
        "type": "registry-module-versions"
    },
    "meta": {}
}
```

- **400 Bad Request** –
- **401 Unauthorized** –
- **403 Forbidden** –
- **404 Not Found** –
- **409 Conflict** –
- **422 Unprocessable Entity** –
resync_module_version

POST /api/iacp/v3/registry-modules/{organization}/{module_name}/{module_provider}/{version}/resync

Trigger resync of the Registry Module of some version associated with VCS revision.

Parameters

- organization (string)
- module_name (string)
- module_provider (string)
- version (string)

Status Codes

- 204 No Content
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 409 Conflict
- 422 Unprocessable Entity

workspaces

get_workspaces

GET /api/iacp/v3/organizations/{organization}/workspaces

This endpoint list the organizations workspaces.

Parameters

- organization (string)

Query Parameters

- page[number] (string) – Page number
- page[size] (string) – Page size

Example request:

```
GET /api/iacp/v3/organizations/{organization}/workspaces HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
create_workspace

POST /api/iacp/v3/organizations/{organization}/workspaces
Create a Workspace.

Parameters

- **organization**(string)

Example request:

```json
POST /api/iacp/v3/organizations/{organization}/workspaces HTTP/1.1
Host: example.com
Content-Type: application/vnd.api+json

{
    "data": {
        "attributes": {
            "auto-apply": true,
            "name": "string",
            "queue-all-runs": true,
            "terraform-version": "string",
            "vcs-repo": {
                "branch": "string",
                "identifier": "string",
                "ingress-submodules": true,
                "oauth-token-id": "string",
                "webhook-url": "string"
            },
            "working-directory": "string"
        },
        "id": "string",
        "links": {
            "self": "string"
        },
        "relationships": {
            "current-run": {
                "data": {
                    "id": "string",
                    "type": "deferred-runs"
                }
            }
        },
        "type": "workspaces"
    },
    "meta": {}
}
```

Status Codes

- **201 Created** – Example response:

```
HTTP/1.1 201 Created
Content-Type: application/vnd.api+json
```

(continues on next page)
400 Bad Request –
401 Unauthorized –
403 Forbidden –
404 Not Found –
409 Conflict –
422 Unprocessable Entity –
delete_organization_workspace

DELETE /api/iacp/v3/organizations/{organization}/workspaces/{workspace_name}
Delete a workspace.

Parameters

- organization (string)
- workspace_name (string)

Status Codes

- 204 No Content
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 409 Conflict
- 422 Unprocessable Entity

get_organization_workspace

GET /api/iacp/v3/organizations/{organization}/workspaces/{workspace_name}
Return Workspace for the Organization.

Parameters

- organization (string)
- workspace_name (string)

Example request:

```
GET /api/iacp/v3/organizations/{organization}/workspaces/{workspace_name} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 409 Conflict
- 422 Unprocessable Entity
update_organization_workspace

PATCH /api/iacp/v3/organizations/{organization}/workspaces/{workspace_name}
No docstring

Parameters

- organization (string)-
- workspace_name (string)-

Example request:

```
PATCH /api/iacp/v3/organizations/{organization}/workspaces/{workspace_name} HTTP/1.1
Host: example.com
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "auto-apply": true,
      "name": "string",
      "queue-all-runs": true,
      "terraform-version": "string",
      "vcs-repo": {
        "branch": "string",
        "identifier": "string",
        "ingress-submodules": true,
        "oauth-token-id": "string",
        "webhook-url": "string"
      },
      "working-directory": "string"
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {
      "current-run": {
        "data": {
          "id": "string",
          "type": "deferred-runs"
        }
      }
    },
    "type": "workspaces"
  },
  "meta": {}
}
```

Status Codes

- 201 Created – Example response:

```
HTTP/1.1 201 Created
Content-Type: application/vnd.api+json

{
  (continues on next page)
}
```
"data": {
  "attributes": {
    "auto-apply": true,
    "can-queue-destroy-plan": true,
    "created-at": "2020-03-30T16:10:27.586407",
    "environment": "string",
    "locked": true,
    "name": "string",
    "operations": true,
    "permissions": {},
    "queue-all-runs": true,
    "terraform-version": "string",
    "vcs-repo": {
      "branch": "string",
      "identifier": "string",
      "ingress-submodules": true,
      "oauth-token-id": "string",
      "webhook-url": "string"
    },
    "working-directory": "string"
  },
  "id": "string",
  "links": {
    "self": "string"
  },
  "relationships": {
    "current-run": {
      "data": {
        "id": "string",
        "type": "deferred-runs"
      }
    },
    "organization": {
      "data": {
        "id": "string",
        "type": "organizations"
      }
    },
    "type": "workspaces"
  },
  "meta": {}
}

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –
delete_workspace

DELETE /api/iacp/v3/workspaces/{workspace}
Delete a workspace.

Parameters

- **workspace** (string)

Status Codes

- 204 No Content
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 409 Conflict
- 422 Unprocessable Entity

get_workspace

GET /api/iacp/v3/workspaces/{workspace}
Show workspace.

Parameters

- **workspace** (string)

Example request:

```
GET /api/iacp/v3/workspaces/{workspace} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – Example response:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "auto-apply": true,
      "can-queue-destroy-plan": true,
      "created-at": "2020-03-30T16:10:27.586407",
      "environment": "string",
      "locked": true,
      "name": "string",
      "operations": true,
      "permissions": {},
      "queue-all-runs": true,
      "terraform-version": "string",
      "vcs-repo": {
```
"branch": "string",
"identifier": "string",
"ingress-submodules": true,
"oauth-token-id": "string",
"webhook-url": "string"
},
"working-directory": "string"
},
"id": "string",
"links": {
  "self": "string"
},
"relationships": {
  "current-run": {
    "data": {
      "id": "string",
      "type": "deferred-runs"
    }
  },
  "organization": {
    "data": {
      "id": "string",
      "type": "organizations"
    }
  },
  "type": "workspaces"
},
"meta": {}
}

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

**update_workspace**

**PATCH /api/iacp/v3/workspaces/{workspace}**

No docstring

**Parameters**

- **workspace (string)** –

**Example request:**

```bash
PATCH /api/iacp/v3/workspaces/{workspace} HTTP/1.1
Host: example.com
Content-Type: application/vnd.api+json
```
{
  "data": {
    "attributes": {
      "auto-apply": true,
      "name": "string",
      "queue-all-runs": true,
      "terraform-version": "string",
      "vcs-repo": {
        "branch": "string",
        "identifier": "string",
        "ingress-submodules": true,
        "oauth-token-id": "string",
        "webhook-url": "string"
      },
      "working-directory": "string"
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {
      "current-run": {
        "data": {
          "id": "string",
          "type": "deferred-runs"
        }
      }
    },
    "type": "workspaces"
  },
  "meta": {}
}

Status Codes

- **201 Created** – Example response:

```
HTTP/1.1 201 Created
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "auto-apply": true,
      "can-queue-destroy-plan": true,
      "created-at": "2020-03-30T16:10:27.586407",
      "environment": "string",
      "locked": true,
      "name": "string",
      "operations": true,
      "permissions": {},
      "queue-all-runs": true,
      "terraform-version": "string",
      "vcs-repo": {
        "branch": "string",
        "identifier": "string",
      }
    }
  }
```

(continues on next page)
"ingress-submodules": true,
"oauth-token-id": "string",
"webhook-url": "string"
},
"working-directory": "string"
},
"id": "string",
"links": {
"self": "string"
},
"relationships": {
"current-run": {
"data": {
"id": "string",
"type": "deferred-runs"
}
},
"organization": {
"data": {
"id": "string",
"type": "organizations"
}
},
"type": "workspaces"
},
"meta": {}
}

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

**lock_workspace**

**POST /api/iacp/v3/workspaces/{workspace}/actions/lock**

Lock a workspace.

**Parameters**

- **workspace** (string)

**Status Codes**

- **200 OK** – Example response:

```json
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{}
```
"data": {
  "attributes": {
    "auto-apply": true,
    "can-queue-destroy-plan": true,
    "created-at": "2020-03-30T16:10:27.586407",
    "environment": "string",
    "locked": true,
    "name": "string",
    "operations": true,
    "permissions": {},
    "queue-all-runs": true,
    "terraform-version": "string",
    "vcs-repo": {
      "branch": "string",
      "identifier": "string",
      "ingress-submodules": true,
      "oauth-token-id": "string",
      "webhook-url": "string"
    },
    "working-directory": "string"
  },
  "id": "string",
  "links": {
    "self": "string"
  },
  "relationships": {
    "current-run": {
      "data": {
        "id": "string",
        "type": "deferred-runs"
      }
    },
    "organization": {
      "data": {
        "id": "string",
        "type": "organizations"
      }
    },
    "type": "workspaces"
  },
  "meta": {}
unlock_workspace

POST /api/iacp/v3/workspaces/{workspace}/actions/unlock
Unlock a workspace.

Parameters

- workspace (string)

Status Codes

- 200 OK – Example response:

```json
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "auto-apply": true,
      "can-queue-destroy-plan": true,
      "created-at": "2020-03-30T16:10:27.586407",
      "environment": "string",
      "locked": true,
      "name": "string",
      "operations": true,
      "permissions": {}
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {
      "current-run": {
        "data": {
          "id": "string",
          "type": "deferred-runs"
        }
      },
      "organization": {
        "data": {
          "id": "string",
          "type": "organizations"
        }
      }
    },
    "type": "workspaces"
  },
  "meta": {}
plans

get_plan

GET /api/iacp/v3/plans/{plan}
Show details of a specific plan.

Parameters

- **plan**(string)

Example request:

```
GET /api/iacp/v3/plans/{plan} HTTP/1.1
Host: example.com
```

Status Codes

- **200 OK** – Example response:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "has-changes": true,
      "log-read-url": "string",
      "resource-additions": 1,
      "resource-changes": 1,
      "resource-destinations": 1,
      "status": "pending",
      "status-timestamps": {}
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {},
    "type": "plans"
  },
  "meta": {}
}
```
• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –

runs

create_run

POST /api/iacp/v3/runs
Create new Terraform Run.

Example request:

```json
{
  "data": {
    "attributes": {
      "is-destroy": true,
      "message": "string",
      "source": "string"
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {
      "configuration-version": {
        "data": {
          "id": "string",
          "type": "configuration-versions"
        }
      },
      "workspace": {
        "data": {
          "id": "string",
          "type": "workspaces"
        }
      }
    },
    "type": "runs"
  },
  "meta": {}
}
```

Status Codes

• 200 OK – Example response:
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
    "data": {
        "attributes": {
            "actions": {},
            "created-at": "2020-03-30T16:10:27.586407",
            "created-by": "string",
            "force-cancel-available-at": "2020-03-30T16:10:27.586407",
            "has-changes": true,
            "is-destroy": true,
            "message": "string",
            "permissions": {},
            "position-in-queue": 1,
            "source": "string",
            "status": "pending",
            "status-timestamps": {}
        },
        "id": "string",
        "links": {
            "self": "string"
        },
        "relationships": {
            "apply": {
                "data": {
                    "id": "string",
                    "type": "applies"
                }
            },
            "configuration-version": {
                "data": {
                    "id": "string",
                    "type": "configuration-versions"
                }
            },
            "cost-estimate": {
                "data": {
                    "id": "string",
                    "type": "cost-estimates"
                }
            },
            "plan": {
                "data": {
                    "id": "string",
                    "type": "plans"
                }
            },
            "policy-checks": {
                "data": [
                    {
                        "id": "string",
                        "type": "policy-checks"
                    }
                ]
            }
        }
    }
}
Athena Documentation, Release 1.0

(continued from previous page)

```
"workspace": {

  "data": {
    "id": "string",
    "type": "workspaces"
  }
},

  "type": "runs"
},

  "meta": {}
}
```

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

**get_run**

GET /api/iacp/v3/runs/{run}

Show details of a specific run.

**Parameters**

- **run (string)** –

**Example request:**

```
GET /api/iacp/v3/runs/{run} HTTP/1.1
Host: example.com
```

**Status Codes**

- **200 OK** – Example response:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "actions": {},
      "created-at": "2020-03-30T16:10:27.586407",
      "created-by": "string",
      "force-cancel-available-at": "2020-03-30T16:10:27.586407",
      "has-changes": true,
      "is-destroy": true,
      "message": "string",
      "permissions": {},
      "position-in-queue": 1,
    }
  }
}
```
"source": "string",
"status": "pending",
"status-timestamps": {}
],
"id": "string",
"links": {
  "self": "string"
},
"relationships": {
  "apply": {
    "data": {
      "id": "string",
      "type": "applies"
    }
  },
  "configuration-version": {
    "data": {
      "id": "string",
      "type": "configuration-versions"
    }
  },
  "cost-estimate": {
    "data": {
      "id": "string",
      "type": "cost-estimates"
    }
  },
  "plan": {
    "data": {
      "id": "string",
      "type": "plans"
    }
  },
  "policy-checks": {
    "data": [
      {
        "id": "string",
        "type": "policy-checks"
      }
    ]
  },
  "workspace": {
    "data": {
      "id": "string",
      "type": "workspaces"
    }
  }
},
"type": "runs"
},
"meta": {}}

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –

1.3. Scalr Features 171
confirm_run

POST /api/iacp/v3/runs/{run}/actions/apply
Apply a run that is paused waiting for confirmation after a plan.

Parameters
• run (string)

Status Codes
• 202 Accepted –
• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –

cancel_run

POST /api/iacp/v3/runs/{run}/actions/cancel
Interrupt a run that is currently planning or applying.

Parameters
• run (string)

Status Codes
• 202 Accepted –
• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –
**discard_run**

**POST /api/iacp/v3/runs/{run}/actions/discard**

Skip any remaining work on runs that are paused waiting for confirmation or priority.

*Parameters*

- run (*string*)

*Status Codes*

- 202 Accepted –
- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

**force_cancel_run**

**POST /api/iacp/v3/runs/{run}/actions/force-cancel**

End the run immediately.

*Parameters*

- run (*string*)

*Status Codes*

- 202 Accepted –
- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

**download_policy_input**

**POST /api/iacp/v3/runs/{run}/policy-input**

Get zip archive with policy check input data generated for given run.

*Parameters*

- run (*string*)

*Status Codes*

- 200 OK –
- 400 Bad Request –
get_runs

GET /api/iacp/v3/workspaces/{workspace}/runs
List Terraform runs.

Parameters
• workspace (string)

Query Parameters
• page[number] (string) – Page number
• page[size] (string) – Page size

Example request:
GET /api/iacp/v3/workspaces/{workspace}/runs HTTP/1.1
Host: example.com

Status Codes
• 200 OK –
• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –

state-versions

list_state_versions

GET /api/iacp/v3/state-versions
List the all state versions.

Query Parameters
• filter[organization][name] (string) – The name of the organization that owns the desired workspace. If included, you must also included the workspace name filter. (Required)
• filter[workspace][name] (string) – The name of one workspace to list variables for. If included, you must also include the organization name filter. (Required)
Example request:
```
GET /api/iacp/v3/state-versions?filter%5Borganization%5D%5Bname%5D=string&filter%5Bworkspace%5D%5Bname%5D=string HTTP/1.1
Host: example.com
```

Status Codes
- 200 OK –
- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

get_state_version

GET /api/iacp/v3/state-versions/{state_version}
Show state version.

Parameters
- `state_version` (string) –

Example request:
```
GET /api/iacp/v3/state-versions/{state_version} HTTP/1.1
Host: example.com
```

Status Codes
- 201 Created – Example response:

```
HTTP/1.1 201 Created
Content-Type: application/vnd.api+json

{
  
  "data": {
    
    "attributes": {
    "created-at": "2020-03-30T16:10:27.586407",
    "hosted-state-download-url": "string",
    "lineage": "string",
    "md5": "string",
    
    "serial": 1,

    "state": "string",

    "vcs-commit-sha": "string",
    "vcs-commit-url": "string"

  },

  "id": "string",

```

(continues on next page)
get_current_state_version

GET /api/iacp/v3/workspaces/{workspace}/current-state-version

Fetch the current state version for the given workspace.

Parameters

- **workspace** *(string)*

Example request:

```text
GET /api/iacp/v3/workspaces/{workspace}/current-state-version HTTP/1.1
Host: example.com
```

Status Codes

- **200 OK** – Example response:

```text
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "created-at": "2020-03-30T16:10:27.586407",
      "hosted-state-download-url": "string",
      "lineage": "string",
      "md5": "string",
      "serial": 1,
      "state": "string",
    }
  },
  "links": {
    "self": "string"
  },
  "relationships": {
    "run": {
      "data": {
        "id": "string",
        "type": "runs"
      }
    }
  },
  "type": "state-versions"
}
```
"vcs-commit-sha": "string",
"vcs-commit-url": "string"
},
"id": "string",
"links": {
    "self": "string"
},
"relationships": {
    "run": {
        "data": {
            "id": "string",
            "type": "runs"
        }
    }
},
"type": "state-versions"
},
"meta": {}}

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

create_state_version

POST /api/iacp/v3/workspaces/{workspace}/state-versions

Create a state version and set it as the current state version for the given workspace.

Parameters

• workspace (string) –

Example request:

POST /api/iacp/v3/workspaces/{workspace}/state-versions HTTP/1.1
Host: example.com
Content-Type: application/vnd.api+json

{
    "data": {
        "attributes": {
            "lineage": "string",
            "md5": "string",
            "serial": 1,
            "state": "string"
        },
        "id": "string",
        "links": {
            "self": "string"
        }
    }
}
Athena Documentation, Release 1.0

},
    "relationships": {
        "run": {
            "data": {
                "id": "string",
                "type": "runs"
            }
        }
    },
    "type": "state-versions"
},
"meta": {}
}

Status Codes

- 201 Created – Example response:

```json
HTTP/1.1 201 Created
Content-Type: application/vnd.api+json

{
    "data": {
        "attributes": {
            "created-at": "2020-03-30T16:10:27.586407",
            "hosted-state-download-url": "string",
            "lineage": "string",
            "md5": "string",
            "serial": 1,
            "state": "string",
            "vcs-commit-sha": "string",
            "vcs-commit-url": "string"
        },
        "id": "string",
        "links": {
            "self": "string"
        },
        "relationships": {
            "run": {
                "data": {
                    "id": "string",
                    "type": "runs"
                }
            }
        },
        "type": "state-versions"
    },
    "meta": {}
}
```

- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
variables

get_variables

GET /api/iacp/v3/vars
Return the list of the Terraform Variables.

Query Parameters

- filter[organization][name] (string) – The name of the organization that owns the desired workspace. If included, you must also included the workspace name filter.
- filter[workspace][name] (string) – The name of one workspace to list variables for. If included, you must also include the organization name filter.
- page[number] (string) – Page number
- page[size] (string) – Page size

Example request:

```
GET /api/iacp/v3/vars HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK –
- 400 Bad Request –
- 401 Unauthorized –
- 403 Forbidden –
- 404 Not Found –
- 409 Conflict –
- 422 Unprocessable Entity –

create_variable

POST /api/iacp/v3/vars
Create new Terraform Variable.

Example request:

```
POST /api/iacp/v3/vars HTTP/1.1
Host: example.com
Content-Type: application/vnd.api+json

{
    "data": {
        "attributes": {
            "category": "terraform",
            "description": "string",
            "hcl": true,
```
"key": "string",
"sensitive": true,
"value": "string"
},
"id": "string",
"links": {
  "self": "string"
},
"relationships": {
  "workspace": {
    "data": {
      "id": "string",
      "type": "workspaces"
    }
  }
},
"type": "vars"
},
"meta": {}}

Status Codes

- **200 OK** – Example response:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": {
    "attributes": {
      "category": "terraform",
      "description": "string",
      "hcl": true,
      "key": "string",
      "sensitive": true,
      "value": "string"
    },
    "id": "string",
    "links": {
      "self": "string"
    },
    "relationships": {
      "workspace": {
        "data": {
          "id": "string",
          "type": "workspaces"
        }
      }
    },
    "type": "vars"
  },
  "meta": {}}
```

- **400 Bad Request** –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –

delete_variable

DELETE /api/iacp/v3/vars/{var}
Delete terraform variable.

Parameters
• var (string) –

Status Codes
• 204 No Content –
• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –

get_variable

GET /api/iacp/v3/vars/{var}
Show a terraform variable.

Parameters
• var (string) –

Example request:

GET /api/iacp/v3/vars/{var} HTTP/1.1
Host: example.com

Status Codes
• 200 OK – Example response:

HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{   "data": {   "attributes": {   "category": "terraform",   "description": "string",  
    }  
}  

(continues on next page)
• 400 Bad Request –
• 401 Unauthorized –
• 403 Forbidden –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –

update_variable

PATCH /api/iacp/v3/vars/{var}
Updates terraform variable.

Parameters

• var (string)

Example request:

```
PATCH /api/iacp/v3/vars/{var} HTTP/1.1
Host: example.com
Content-Type: application/vnd.api+json

{
    "data": {
        "attributes": {
            "category": "terraform",
            "description": "string",
            "hcl": true,
            "key": "string",
            "sensitive": true,
            "value": "string"
        }
    }
}
```

(continues on next page)
Status Codes

- **200 OK** – Example response:

```json
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
   "data": {
      "attributes": {
         "category": "terraform",
         "description": "string",
         "hcl": true,
         "key": "string",
         "sensitive": true,
         "value": "string"
      },
      "id": "string",
      "links": {
         "self": "string"
      },
      "relationships": {
         "workspace": {
            "data": {
               "id": "string",
               "type": "workspaces"
            }
         }
      }
   },
   "type": "vars"
   "meta": {}
}
```

- **400 Bad Request** –
- **401 Unauthorized** –
- **403 Forbidden** –
• 404 Not Found –
• 409 Conflict –
• 422 Unprocessable Entity –
HTTP ROUTING TABLE

/api
GET /api/iacp/v3/applies/{apply}, 137
GET /api/iacp/v3/configuration-versions/{configuration_version}, 138
GET /api/iacp/v3/configuration-versions/{configuration_version}/ingress-attributes, 139
GET /api/iacp/v3/cost-estimates/{cost_estimate}, 143
GET /api/iacp/v3/cost-estimates/{cost_estimate}/output, 144
GET /api/iacp/v3/organizations, 145
GET /api/iacp/v3/organizations/{organization}, 145
GET /api/iacp/v3/organizations/{organization}/workspaces, 155
GET /api/iacp/v3/organizations/{organization}/workspaces/{workspace_name}, 158
GET /api/iacp/v3/plans/{plan}, 167
GET /api/iacp/v3/registry-modules/{module}, 151
GET /api/iacp/v3/registry-modules/{organization}/{module_name}/{module_provider}, 153
POST /api/iacp/v3/applies/actions/delete/{apply}, 149
POST /api/iacp/v3/applies/actions/cancel/{apply}, 154
POST /api/iacp/v3/organizations/actions/delete/{organization}, 156
POST /api/iacp/v3/organizations/actions/delete/{organization}/workspaces, 158
POST /api/iacp/v3/organizations/actions/delete/{organization}/workspaces/{workspace_name}, 159
POST /api/iacp/v3/registry-modules/actions/delete/{organization}/{module_name}, 150
POST /api/iacp/v3/registry-modules/actions/delete/{organization}/{module_name}/{provider_name}, 150
POST /api/iacp/v3/registry-modules/actions/delete/{organization}/{module_name}/{provider_name}/{version}, 150
POST /api/iacp/v3/registry-modules/resync/{organization}/{module_name}/{module_provider}, 153
POST /api/iacp/v3/registry-modules/versions/{organization}/{module_name}/{module_provider}, 153
POST /api/iacp/v3/registry-modules/{organization}/{module_name}/{module_provider}/{version}/resync, 155
POST /api/iacp/v3/plans/actions/cancel/{plan}, 162
POST /api/iacp/v3/plans/actions/discard/{plan}, 163
POST /api/iacp/v3/plans/actions/force-cancel/{plan}, 164
POST /api/iacp/v3/plans/policy-input/{plan}, 169
POST /api/iacp/v3/registry-modules, 148
POST /api/iacp/v3/registry-modules/actions/delete/{configuration_version}, 150
POST /api/iacp/v3/registry-modules/actions/delete/{configuration_version}/ingress-attributes, 150
POST /api/iacp/v3/registry-modules/actions/delete/{organization}, 151
POST /api/iacp/v3/registry-modules/actions/delete/{organization}/workspaces, 153
POST /api/iacp/v3/registry-modules/actions/delete/{organization}/workspaces/{workspace_name}, 155
POST /api/iacp/v3/registry-modules/actions/delete/{organization}/workspaces/{workspace_name}/current-state-version, 157
POST /api/iacp/v3/registry-modules/actions/delete/{organization}/workspaces/{workspace_name}/state-versions, 158
DELETE /api/iacp/v3/organizations/{organization}/workspaces/{workspace}, 161
DELETE /api/iacp/v3/organizations/{organization}/workspaces/{workspace}/configuration-versions, 175
DELETE /api/iacp/v3/organizations/{organization}/workspaces/{workspace}/current-state-version, 176
DELETE /api/iacp/v3/organizations/{organization}/workspaces/{workspace}/runs, 174
POST /api/iacp/v3/organizations/{organization}/workspaces, 146
POST /api/iacp/v3/organizations/{organization}/workspaces/actions/lock, 164
POST /api/iacp/v3/organizations/{organization}/workspaces/actions/unlock, 166
POST /api/iacp/v3/organizations/{organization}/workspaces/actions/lock/{workspace}, 165
POST /api/iacp/v3/organizations/{organization}/workspaces/actions/unlock/{workspace}, 167
POST /api/iacp/v3/organizations/{organization}/workspaces/actions/lock/{workspace}/configuration-versions, 168
POST /api/iacp/v3/organizations/{organization}/workspaces/actions/unlock/{workspace}/current-state-version, 170
POST /api/iacp/v3/organizations/{organization}/workspaces/actions/unlock/{workspace}/state-versions, 172
POST /api/iacp/v3/organizations/{organization}/workspaces/actions/unlock/{workspace}/runs, 173
POST /api/iacp/v3/organizations/{organization}/registry-modules, 148
POST /api/iacp/v3/organizations/{organization}/registry-modules/actions/delete, 150
POST /api/iacp/v3/organizations/{organization}/registry-modules/actions/delete/{organization}, 151
POST /api/iacp/v3/organizations/{organization}/registry-modules/actions/delete/{organization}/registry-modules, 154
POST /api/iacp/v3/organizations/{organization}/registry-modules/actions/delete/{organization}/registry-modules/actions/delete, 156
POST /api/iacp/v3/organizations/{organization}/registry-modules/actions/delete/{organization}/registry-modules/actions/delete, 157
POST /api/iacp/v3/organizations/{organization}/registry-modules/actions/delete/{organization}/registry-modules/actions/delete, 159
POST /api/iacp/v3/organizations/{organization}/registry-modules/actions/delete/{organization}/registry-modules/actions/delete, 161
POST /api/iacp/v3/organizations/{organization}/registry-modules/actions/delete/{organization}/registry-modules/actions/delete, 163