Rafay Platform

Environment Manager Developer Guide

Version History

Version	Date	Details
0.1	Oct 17, 2024	Initial version based on Rafay docs
1.0	Nov 15, 2024	Standalone version

TABLE OF CONTENTS

Introduction

Rafay’s Environment Manager empowers platform teams to implement a self-service approach for provisioning and managing infrastructure and applications using Rafay's templating language and straightforward declarative automation.

Serving as the core engine, Environment Manager streamlines workflow automation—whether for infrastructure or applications—leveraging Rafay's powerful templating language.

Workflow Engine

At the heart of the Environment Manager lies a robust and versatile workflow engine that orchestrates a Directed Acyclic Graph (DAG) of activities. Every change to the environment is executed through a workflow run, which represents this DAG of interconnected activities. As the workflow progresses, activities are assigned to agents, which execute them and report their statuses back. Each activity includes the driver details required for agents to perform the execution seamlessly.

Environment Decomposition

Each resource in an environment is broken down into multiple activities, such as *git checkout*, *IaC init*, *IaC plan*, *IaC apply*, and *IaC output*, with pre and post hooks integrated at each stage. The activities of various resources within the environment are organized into a Directed Acyclic Graph (DAG) based on the relationships between the resources. Additionally, hooks like *oninit*, *onsuccess*, *onfailure*, and *oncompletion* are incorporated as defined in the environment template. This workflow is then executed by the engine, which manages and updates the states and statuses of the activities.

Hooks

Hooks are integrated as activities within the workflow, with supported primitive types including HTTP, container, and function-based hooks. All other hook types are executed as one of these primitives. For example, hooks such as approvals and notifications are configured as HTTP activities, with endpoints invoking external services responsible for handling these tasks. This approach allows the respective service to manage integrations with third-party platforms like Jira, Git PRs, emails, and more. Meanwhile, hook types such as scripts are executed as shell commands within containers.

Agent Execution

Agents can be deployed either as a Kubernetes deployment within a cluster or as a Docker container. In Docker mode, the container runs a lightweight K3s cluster where the agent operates. Agents can handle two types of activities: HTTP and container-based tasks. For container activities, pods are created in the Kubernetes cluster, with logs streamed back to the

workflow engine, and the exit status updated to the corresponding activity. HTTP activities are processed using an HTTP client. For long-running tasks, Function Driver Containers can be utilized, allowing their execution to be managed through configuration. These containers run at specified intervals until the defined success conditions are met.

Benefits

- **Simplicity**: The framework and templating language streamlines the lifecycle management of infrastructure, Kubernetes, and applications. Features like dependency management, lifecycle hooks, and variable management make the process straightforward and efficient
- **Extensibility**: Designed with a "Bring Your Own" (BYO) approach, the framework supports the automation of workflows regardless of the infrastructure provider. It seamlessly integrates with tools like OpenTofu, HCP Terraform, Rafay Custom Provider, Python scripts, and other scripting solutions
- **Managed Solution**: The workflow engine operates as a hosted SaaS service within the Rafay platform. Users can utilize Rafay's prebuilt "RDU" templates or create custom workflows using their own logic (BYO) to offer developers a self-service experience
- **Multi-Tenancy enabled:** The platform applies a unified layer of multi-tenancy across all resources, including Kubernetes and cloud infrastructure. This standardized governance eliminates the need to manage individual IAM permissions for providers like AWS, ensuring streamlined resource access control

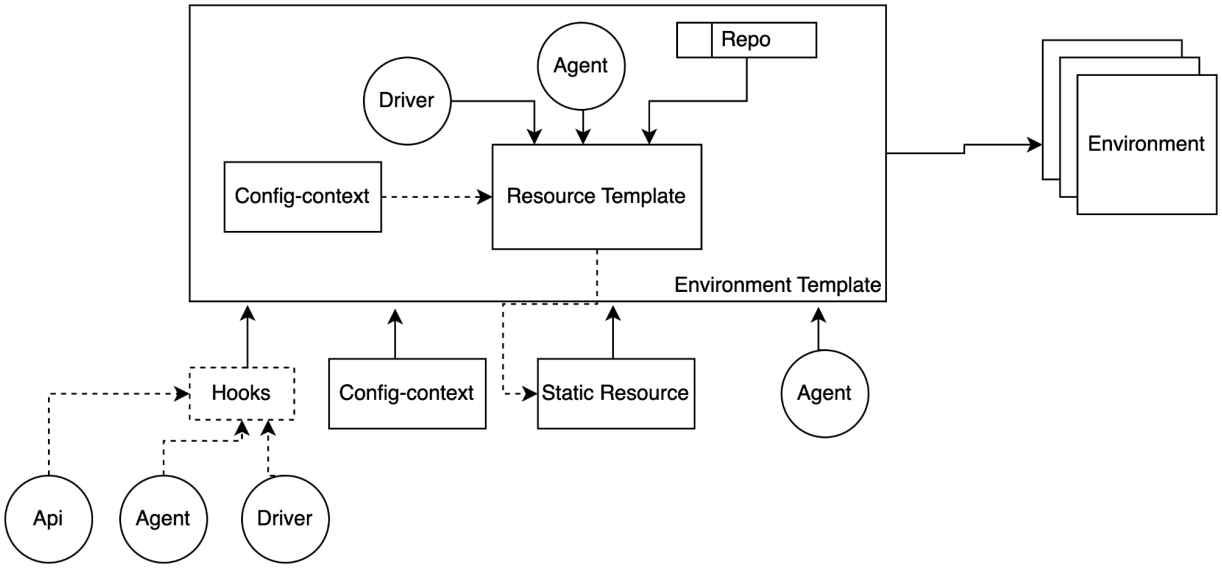
Environment Template Building Blocks

Templatization is a foundational concept in the Environment Manager, enabling the creation of environments using two primary templates: the Environment Template and the Resource Template. These templates allow users to generate one or more environment instances efficiently.

The key components of the Environment Manager's templating system include:

- Environment Template
- Resource Template
- Config Context
- Drivers
- Static Resources - Environment and Resource level
- HCL Code - referenced by Resource Template
- Agents
- Hooks
- Repository
- Drivers/Function Drivers
- Actions
- Schedules/Cron jobs
- Expressions

The Environment Template combines all these components required to define and create an environment seamlessly.



Environment Template

Environment templates are collections of resource templates and static resources that encapsulate key elements such as environment variables, lifecycle hooks, and dependencies. For example, an environment template can define the complete operational setup needed for a complex Kubernetes application.

Typically, an environment template consists of one or more resource templates or static resources. It can be used to generate one or more environment instances.

Setting	**Description**
apiVersion	Rafay's API version eaas.envmgmt.io/v1
Kind	EnvironmentTemplate
Metadata	**description**: Description or the purpose of the environment template resource **name**: Name of the environment template **project**: Project name where the environment template resides

	annotations: Annotations can be added to display information such as Category and the IaC source code information for the environment template. For example, if the category of the environment template is 'Developer Productivity', use the key **eaas.envmgmt.io/category** and the value **Developer Productivity**. To display the IaC source code information, use the key **eaas.envmgmt.io/github .** These annotations typically get used in filtering/search of the environment templates **labels:** Use labels to identify attributes of objects that are meaningful and relevant to users, but do not directly imply semantics to the core system. This is a key-value pair. For example, for test environments, you can configure key as environment and the value as dev
spec -> agents	This should be set to the name of the agent designated to execute the code for all resource templates within this environment template
spec -> iconURL	Icon representing the resource created by the environment template. This has to be in base64 encoded format
spec -> readme	Inline README documentation for the environment template that can include instructions for using the template (use HTML tags for formatting)
spec -> resources	A list of resources should be specified here. Each resource includes **name**; **kind** as resourcetemplate, resource or environment; **resourceOptions** that contains version information of the resource; **type** to indicate whether the resource is dynamic or static - dependsOn: - name: rs-tmpl kind: resourcetemplate name: grp-rs-tmpl resourceOptions: version: v1 type: dynamic Dependency ordering of resources is driven by dependsOn information. For the case above, resourcetemplate grp-rs-tmpl is dependent on rs-tmpl
spec -> variables	A list of input variables should be specified here. Each variable includes **name**; **valueType** being one of hcl, json, expression, text; **override** type being one of allowed, notallowed, restricted; **required**; **value**; **sensitive** If overrideType is set to allowed , it means that the value can be overridden at creation/launch of the environment

	If it is set to notallowed , it means that the value cannot be overridden at creation/launch of environment If it is set to restricted, it means the value can be overridden at creation/launch of environment but it restricted to one of the values provided by override -> restrictedValues required -> true or false , true being mandatory field and false being optional value -> single value sensitive -> true means the value will be encrypted for storage and will not be seen by end user of the template
spec -> contexts	Refers to the Config Context that is defined outside of the environment template Note: Config Context is typically used for passing config settings (be it variables, environment variables or files) to multiple environment templates/resource templates
spec -> hooks	Refers to configuration of lifecycle hooks for environment template ● on_completion (Block List) To configure an on completion lifecycle hook ● on_failure (Block List) To configure an on failure lifecycle hook ● on_init (Block List) To configure an on initialize lifecycle hook ● on_success (Block List) To configure an on success lifecycle hook ● provider (Block List) To configure provider hooks **Nested Schema for spec.hooks.on_completion** **Required** ● name (String) Name of the hook ● type (String) Specify the type of hook, available options are approval, container, http, driver ***Optional*** ● agents (Block List) Specify the resource ref agents ● on_failure (String) Specify the on failure action ● options (Block List, Max: 1) Specify the hook options ● timeout_seconds (Number) Specify the timeout in seconds ● driver (Block List, Max: 1) Specify the driver responsible for execution ● execute_once (Boolean) Specify if the hook should be executed only once

spec -> schedules	***Required*** ● name (String) Name of the schedule ● type (String) Schedule type, available options are deploy, destroy, custom workflow ● cadence (Block List) Configure a schedule cadence at which a job would automatically trigger ***Optional*** ● description (String) Description of the schedule ● context (Block List) Input data configuration that is needed as part of this schedule run ● workflows (Block List) Name of the custom workflow that needs to be executed with this job ● opt_out_options (Block List) Opt Out Options configured with this schedule **Schema for cadence :** ● cron_expression (String) Cron expression used to configure scheduling jobs ***Optional*** ● cron_timezone (String) Specify the timezone of cron expression ● time_to_live (String) Specify the maximum time to live duration of an environment, time units are 'h', 'd' e.g. 8h, 2d **Scheme for outputs:** ● allow_opt_out (Bool) Specify whether users can opt out from this schedule ***Optional*** ● max_allowed_duration (String) Specify the maximum allowed opt out duration, time units are 'm', 'h', 'd' e.g. 8h, 2d ● max_allowed_times (Number) Specify the maximum number of times users can opt out without approval e.g. users can max opt out of this schedule thrice ● approval (Block List, Max: 1) Details of approval workflow that needs to be execution in case of user opt-out

Sample Environment Template spec

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: EnvironmentTemplate
metadata:
  description: Allowed k8s_version changes
```

```
  name: caas-eks-env-tmpl
  project: dev-project
  description: Self Service for requesting a new EKS Kubernetes cluster
  displayName: EKS Cluster as a Service

  annotations:
    eaas.envmgmt.io/category: Developer Productivity

spec:
  iconURL: https://static-00.iconduck.com/assets.00/amazon-eks-icon-455x512-0zairb3r.png
  readme: "Developers...."
  agents:
  - name: agent-one
  hooks: {}
  resources:
  - kind: resourcetemplate
    name: rs-tmpl
    resourceOptions:
      version: v1
    type: dynamic
  - dependsOn:
    - name: rs-tmpl
    kind: resourcetemplate
    name: grp-rs-tmpl
    resourceOptions:
      version: v1
    type: dynamic
  - dependsOn:
    - name: grp-rs-tmpl
    kind: resourcetemplate
    name: grp-assoc-rs-tmpl
    resourceOptions:
      version: v1
    type: dynamic
  variables:
  - name: project
    options:
      override:
        type: allowed
      required: true
    value: caas-eks
    valueType: text
  - name: base_blueprint
    options:
      override:
        type: allowed
    value: minimal
    valueType: text
  - name: base_blueprint_version
    options:
```

```
    override:
      type: allowed
    required: true
  value: 1.28.0
  valueType: text
- name: infra_addons
  options:
    override:
      type: allowed
    required: true
  value: |-
    {
      "addon1" = {
          name          = "cert-manager"
          namespace     = "cert-manager"
          addon_version = "v1.9.1"
          chart_name    = "cert-manager"
          chart_version = "v1.12.3"
          repository    = "cert-manager"
          file_path     = "file://./artifacts/cert-manager/custom_values.yaml"
          depends_on    = []
      }
    }
  valueType: hcl
 version: v1
```

Note: By default, the field name is displayed on the form as-is when creating the environment. To present a more user-friendly name, the "alias" feature can be utilized.

Suppose base_blueprint_version is an input variable in the resource template res-temp. If you want it to appear as ‘Base Blueprint Version’ during environment launch, you can use the alias override feature. This is done by defining the input variable in the Environment Template and referencing the corresponding variable in the resource template.

```
- name: Base Blueprint Version
  options:
    description: Enter the Blueprint version
    override:
      selectors: [resource.res-temp.base_blueprint_version]
      type: allowed
    required: true
  value: "latest"
  valueType: text
```

An environment variable can be mapped to multiple resource template variables, which is useful when the same input is required by multiple variables across different resource templates.

Resource Template

Resource templates are fundamental components of environment templates, containing Infrastructure as Code (IaC), such as HCL, to define resources. They encompass variables, lifecycle hooks (e.g., pre/post actions on completion or deployment), policies, and configurations that determine whether the resource is dedicated to a specific workload or shared across multiple workloads.

Key features of resource templates include:

- Representing a class of resources, which can be infrastructure, applications, or services
- Encapsulating IaC configurations and supporting config contexts
- Utilizing outputs from other resources as input variables through expressions
- Being version-controlled for better management

A resource template cannot be published independently; it must be associated with an environment template to be published. Additionally, resource templates can be shared across projects, allowing users with access to those projects to leverage them for building environment templates.

Setting	**Description**
apiVersion	Rafay's API version eaas.envmgmt.io/v1
Kind	ResourceTemplate
Metadata	**description**: Description or the purpose of the resource template resource **name**: Name of the resource template **project**: Project name where the resource template resides
spec -> agents	This should be set to the name of the agent designated to execute the code of the resource template
spec -> contexts	Refers to the Config Context that is defined outside of the resource template Note: Config Context is typically used for passing config settings (be it variables, environment variables or files) to multiple environment templates/resource templates

spec -> hooks	Refers to configuration of lifecycle hooks for resource template ● on_completion (Block List) To configure an on completion lifecycle hook ● on_failure (Block List) To configure an on failure lifecycle hook ● on_init (Block List) To configure an on initialize lifecycle hook ● on_success (Block List) To configure an on success lifecycle hook ● provider (Block List) To configure provider hooks **Nested Schema for spec.hooks.on_completion** **Required** ● name (String) name of the hook ● type (String) Specify the type of hook, Available options are approval, container, http, driver. ***Optional*** ● agents (Block List) Specify the resource ref agents ● on_failure (String) Specify the on failure action ● options (Block List, Max: 1) Specify the hook options ● timeout_seconds (Number) Specify the timeout in seconds ● driver (Block List, Max: 1) Specify the driver responsible for execution ● execute_once (Boolean) Specify if the hook should be executed only once
spec -> provider	Specify the resource template provider, accepted values are terraform, hcpterraform, opentofu, custom
spec-> providerOptions	Specify Provider specific options ● driver : Specify the driver name responsible for execution, if it's not provided, rafay OpenTofu provider image will be used Based on the provider selection, user can pass the provider specific options to the template ● hcp_terraform: Specify the HCP terraform specific options if any ● open_tofu : Specify the OpenTofu specific options if any ● custom: Specify the custom options if any Other options include ● backend_configs (List of String) This can be either a path to an HCL file with key/value assignments (same format as terraform.tfvars) or a 'key=value' format. This is merged with what is in the configuration file ● lock Don't hold a state lock during the operation. This prevents issues because of concurrently running commands against the same resources ● lock_timeout_seconds (Number) Duration to retry a state lock

	• plugin_dirs (List of String) Directory containing plugin binaries. This overrides all default search paths for plugins, and prevents the automatic installation of plugins. This flag can be used multiple times • refresh (Block List, Max: 1) Skip checking for external changes to remote objects while creating the plan. This can potentially make planning faster, but at the expense of possibly planning against a stale record of the remote system state. • target_resources (List of String) Limit the planning operation to only the given module, resource, or resource instance and all of its dependencies • timeout_seconds (Number) Timeout in seconds • backend_type (String) Specify the backend type, Supported types are custom, system, terraform_cloud • var_files (List of String) Load variable values from the given file, in addition to the default files terraform.tfvars and *.auto.tfvars. Use this option more than once to include more than one variables files • version (String) Terraform version • volumes (Block List) Volumes to be mounted into the terraform driver • with_terraform_cloud (Block List, Max: 1) [DEPRECATED] WithTerraformCloud should be set to true when used with terraform cloud
spec-> repositoryOptions	Repository options to be provided from where code has to be pulled by Agent when running the resource template • branch (String) Specify the branch • directory_path (String) Specify the directory path • name (String) Specify the name of the repository
spec -> variables	A list of input variables should be specified here. Each variable includes **name**; **valueType** being one of hcl, json, expression, text; **override** type being one of allowed, notallowed, restricted; **required**; **value**; **sensitive** If overrideType is set to allowed , it means that the value can be overridden at creation/launch of the environment If it is set to notallowed , it means that the value cannot be overridden at creation/launch of environment If it is set to restricted, it means the value can be overridden at creation/launch of environment but it restricted to one of the values provided by override -> restrictedValues required -> true or false , true being mandatory field and false being optional value -> single value sensitive -> true means the value will be encrypted for storage and will not be seen by end user of the template
spec -> version	Version of the resource template

Sample **Resource Template** spec

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: ResourceTemplate
metadata:
  name: rauto-workload-resource-template-65837
  project: defaultproject
spec:
  contexts:
  - name: rauto-rafay-creds-config-context-65837
  - name: rauto-workload-config-config-context-65837
  hooks: {}
  provider: terraform
  providerOptions:
    driver:
      name: madhucustom
    terraform:
      backendConfigs:
      - tmp/inputs/workload.tfvars
      backendType: custom
      lock: true
      refresh: true
  repositoryOptions:
    branch: resourcetemplate-testing
    directoryPath: workloads/eaas/terraform/rafay-workload
    name: rauto-repo-65837
  secret:
    name: file://artifacts/rauto-workload-resource-template-65837/sealed-secret.yaml
  variables:
  - name: cluster_name
    value: $(resource."rauto-eks-resource-template-65837".output.eks_cluster_name.value)$
    valueType: expression
  - name: postgres_host
    value: $(resource."rauto-rds-resource-template-65837".output.rds_hostname.value)$
    valueType: expression
  - name: postgres_username
    value: $(resource."rauto-rds-resource-template-65837".output.rds_username.value)$
    valueType: expression
  - name: postgres_password
    options:
      sensitive: true
    value: sealed://variables.3
    valueType: expression
  - name: redis_config_endpoint
    value: $(resource."rauto-ec-resource-template-65837".output.configuration_endpoint_address.value)$
    valueType: expression
  version: custom-noagent
```

Config Context

A Config Context is used to define global variables that are shared across different environment templates and resource templates. It can be applied to both environment and resource templates.

A Config Context consists of:

- Environment variables
- Files
- Input variables

These elements can be utilized within a single environment or across multiple environments, ensuring consistency and reusability.

Examples of using Config Contexts include:

1. **Restricting Region Options for a Team:** A platform team wants to allow the QA team to spin up instances only in specific regions. To achieve this, the platform team creates a Config Context, defines the allowed regions as Input Variables, and sets the Override Type to "Restricted." This Config Context is then linked to an Environment Template. When creating an environment, the QA team can select from the predefined region options
2. **Defining Registry Endpoints:** A platform team needs to specify the registry endpoint for pulling container images. This can be achieved by setting the endpoint as an Environment Variable within a Config Context

Setting	**Description**
apiVersion	Rafay's API version eaas.envmgmt.io/v1
Kind	ConfigContext
metadata	**name**: Takes name of the context context **project** : Mention project where this config context resides
spec -> variables	A list of input variables should be specified here. Each variable includes **name**; **valueType** being one of hcl, json, expression, text; **override** type being one of allowed, notallowed, restricted; **required**; **value**; **sensitive** If overrideType is set to allowed , it means that the value can be overridden at creation/launch of the environment

	If it is set to notallowed , it means that the value cannot be overridden at creation/launch of environment If it is set to restricted, it means the value can be overridden at creation/launch of environment but it restricted to one of the values provided by override -> restrictedValues required -> true or false , true being mandatory field and false being optional value -> single value sensitive -> true means the value will be encrypted for storage and will not be seen by end user of the template
spec -> files	A list of files have to be specified here. name -> name of the file or path to the file data -> content of the file in base64 encoded format If sensitive is set to true, follow instructions here to create and use secretsealer to encrypt data: https://docs.rafay.co/integrations/secretseal/gitrepo/
spec -> envs	List of env variables key -> name of the variable value -> either plain text OR reference to encrypted data like sealed://AWS_SECRET_ACCESS_KEY If sensitive is set to true, follow instructions here to create and use secretsealer to encrypt data https://docs.rafay.co/integrations/secretseal/gitrepo/

Sample **Config Context** spec

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: ConfigContext
metadata:
  name: rauto-rafay-creds-config-context-216151
  project: defaultproject
spec:
  files:
  - data: c2VhbGVkOi8vZmlsZXMuMA==
    name: opt/rafay/rctl.conf
    sensitive: true
  secret:
    name: file://artifacts/rauto-rafay-creds-config-context-216151/sealed-secret.yaml
  variables:
  - name: rafay_config_file
    value: opt/rafay/rctl.conf
    valueType: text
  envs:
  - key: KEY
```

```
  sensitive: false
  value: 10001
- key: AWS_SECRET_ACCESS_KEY
  sensitive: true
  value: sealed://AWS_SECRET_ACCESS_KEY
secret:
  name: file://artifacts/rauto-aws-config-config-context-65837/sealed-secret.yaml
```

Note: Config-context can be defined inline as part of Environment and Resource Template configuration as well.

Drivers

A Driver is a foundational component for creating custom infrastructure provisioners and workflows that may be challenging to achieve using Terraform or similar IaC frameworks. It also enables the use of custom images with specific versions of Terraform/OpenTofu, along with the required tools and libraries for building and managing environments.

Drivers can be utilized in the following scenarios:

- **As a Provider:** To support resource templates in creating custom infrastructure provisioners
- **As Pre- or Post-Hook Handlers:** To manage workflows triggered before or after specific actions
- **As a Cron Job/Scheduled Job Handler**: To automate recurring tasks
- **As an Action Workflow Handler:** To execute defined actions within workflows.

The platform supports three types of drivers:

1. **Container**
2. **HTTP**
3. **Function**

Note: Drivers can be defined inline within Environment and Resource templates.

Container

The Container driver is designed to run container images to completion. A container image, in this context, is a self-contained executable package that includes everything needed to run an application—such as code, runtime, libraries, and system tools. The Container driver is deployed as a pod in the cluster, managed by the `cd-agent`.

To configure this container driver, users can specify details such as:

- CPU and memory limits
- Arguments and commands
- Environment variables
- Files

The Container driver within the workflow engine allows users to define activities that run specific containerized tasks or processes as essential components of the workflow.

Sample **Container driver** spec

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: Driver
metadata:
  name: custom-driver
  project: defaultproject
spec:
  config:
    container:
      cpuLimitMilli: "1024"
      image: registry.dev.rafay-edge.net/rafay/custom-driver/fm-tf-custom:v4
      kubeOptions:
        namespace: rafay-system
        serviceAccountName: fmac1
      memoryLimitMb: "512"
    type: container
status: {}
```

Setting	Description
apiVersion	Rafay's API version eaas.envmgmt.io/v1
Kind	Driver
metadata	**name**: Name of the driver **project**: Project where the driver resides
spec -> config	**container,** type should be container
spec -> config -> container	• image (String) Specify the container image for the driver

Optional

- arguments (List of String) Specify the set of arguments to be passed
- commands (List of String) Specify the set of commands to be executed
- cpu_limit_milli (String) Specify the cpu limit in milli
- env_vars (Map of String) Specify the environment variables to be set in key,value pair
- files (Map of String) Specify the file data
- image_pull_credentials (Block List, Max: 1) Specify the credentials for the registry to pull images from a private registry which requires authentication.
- kube_options (Block List) Specify the kube options of the cluster
- memory_limit_mb (String) Specify the memory limit to be allocated in MB
- volumes (Block List) Configure the container volumes
- working_dir_path (String) Specify the working directory path

Nested Schema for spec.config.container.image_pull_credentials

Required

- password (String) Specify the registry password
- registry (String) Specify the container image registry
- username (String) Specify the registry username

Nested Schema for spec.config.container.kube_config_options

Optional

- kube_config (String) Specify the kube config if out of cluster is enabled
- out_of_cluster (Boolean) Specify if out of cluster

Nested Schema for spec.config.container.kube_options

Optional

- labels (Map of String) Specify the labels
- namespace (String) Specify the namespace
- node_selector (Map of String) Specify the node selectors
- security_context (Block List, Max: 1) Specify the security context
- service_account_name (String) Specify the service account name
- tolerations (Block List) Specify the tolerations

Nested Schema for spec.config.container.kube_options.tolerations

Optional

- key (String) Specify the key
- operator (String) Specify the operator, Accepted values are Exists, Equal
- value (String) Specify the value
- effect (String) Specify the effect, Accepted values are NoSchedule, PreferNoSchedule, NoExecute
- toleration_seconds (Number) Specify the toleration seconds when NoExecute effect is given

	Nested Schema for spec.config.container.kube_options.security_context ***Optional*** ● privileged (Block List, Max: 1) Specify if privileged permissions ● read_only_root_file_system (Block List, Max: 1) Specify if permission is read only root file system **Nested Schema for spec.config.container.kube_options.security_context.read_only_root_file_system** ***Optional*** ● value (Boolean) **Nested Schema for spec.config.container.kube_options.security_context.read_only_root_file_system** ***Optional*** ● value (Boolean) **Nested Schema for spec.config.container.volume_options** ***Optional*** ● mount_path (String) Specify the container mount path ● pvc_size_gb (String) Specify the persistent volume claim size in GB ● pvc_storage_class (String) Specify the persistent volume claim storage class ● use_pvc (Block List, Max: 1) Specify if the container needs to use persistent volume claims **Nested Schema for spec.config.container.working_dir_path.use_pvc** ***Optional*** ● value (Boolean) **Nested Schema for spec.config.container.volumes** ***Optional*** ● mount_path (String) Specify the container mount path ● pvc_size_gb (String) Specify the persistent volume claim size in GB ● pvc_storage_class (String) Specify the persistent volume claim storage class ● use_pvc (Block List, Max: 1) Specify if the container needs to use persistent volume claims **Nested Schema for spec.config.container.working_dir_path.use_pvc** ***Optional*** ● value (Boolean)
spec -> config -> http	***Optional***

	• body (String) Specify the request body • endpoint (String) Specify the http endpoint • headers (Map of String) Specify the http headers • method (String) Specify the http method **Nested Schema for spec.sharing** ***Optional*** • enabled (Boolean) flag to specify if sharing is enabled for resource • projects (Block List) list of projects this resource is shared to **Nested Schema for spec.sharing.projects** ***Optional*** • name (String) name of the project **Nested Schema for timeouts** ***Optional*** • create (String) • delete (String) • update (String) **Nested Schema for spec.inputs** ***Required*** • GET name (String) name of the config context

HTTP

The HTTP driver enables the workflow engine to perform HTTP requests, allowing users to define activities within the workflow that interact with external services or APIs. These activities can be used to trigger actions, retrieve data, or communicate with other systems. The HTTP driver facilitates seamless integration between the workflow engine and external services using the HTTP protocol.

Sample **HTTP driver** spec

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: Driver
metadata:
  description: This is a HTTP driver
```

```
  name: sp-http
  project: demoproject
spec:
  config:
    http:
      body: |-
        <body>
        <h1>This is a heading</h1>
        <p>This is a paragraph.</p>
        </body>
      endpoint: https://httpbin.org
      headers:
        Content-type: application/json
        X-TOKEN: 1234
      method: GET
    maxRetryCount: 2
    successCondition: |-
      if #status.http.statusCode == 200 {
        success: true
      }
      if #status.http.statusCode != 200 {
        failed: true
        reason: "url not reachable"
      }
    timeoutSeconds: 60
    type: http
  sharing:
    enabled: true
    projects:
    - name: project1
    - name: project2
```

Setting	**Description**
apiVersion	Rafay's API version eaas.envmgmt.io/v1
Kind	Driver
metadata	**name**: Name of the driver **project**: Project where the driver resides
spec -> config	**http,** type should be http

spec -> config -> http -> body	This is the payload to pass to the HTTP Call
spec -> config -> http -> method	Set HTTP method here
spec -> config -> http -> endpoint	Set HTTP endpoint URL to call
spec -> config -> http -> headers	Set HTTP headers here

spec -> config -> http -> successCondition	You can have success conditional logic to indicate success or failure with a reason , example here
	
	|-
	if #status.http.statusCode == 200 {
	success: true
	}
	if #status.http.statusCode != 200 {
	failed: true
	reason: "url not reachable"
	}

spec -> config -> http -> maxRetryCount	Retry count setting on failure
spec -> config -> http -> timeoutSeconds	timeoutSeconds after which the run is declared as failure and no more retries

Function

The Function Driver simplifies running functions written in popular programming languages like Go and Python directly within workflows. It eliminates the need to package these functions into containers, as the framework automatically handles packaging and dependency management, streamlining the process.

This feature can be used for tasks such as:

- Executing actions on infrastructure
- Integrating with services like Jira and ServiceNow
- Sending notifications

By enabling functions to run without containers, the Function Driver reduces complexity and accelerates the implementation of actions. Note that the function signature for each supported programming language is predefined and cannot be modified.

For example, the function signature for python is

```
def handle(logger: Logger,request: Dict[str, Any]) -> Dict[str, Any]
```

The function signature for go is

```
type Handler func(ctx context.Context, logger Logger, req Request) (Response, error)
```

Sample **Function driver** spec

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: Driver
metadata:
  name: simple-func-go
  project: defaultproject
spec:
  config:
    type: function
    timeoutSeconds: 300
    pollingConfig:
      repeat: "15s"
      until: "1h"
    function:
      cpuLimitMilli: "50"
      memoryLimitMi: "128"
      language: go
      languageVersion: "1.22"
      maxConcurrency: 10
      numReplicas: 1
      source: |
        package function

        import (
          "context"

          sdk "github.com/RafaySystems/function-templates/sdk/go"
        )

        func Handle(ctx context.Context, logger sdk.Logger, req sdk.Request) (sdk.Response, error) {
          logger.Info("received request", "req", req)

          counter := 0.0
```

```
  if prev, ok := req["previous"]; ok {
    logger.Info("previous request", "prev", prev)
    counter = prev.(map[string]any)["counter"].(float64)
  }

  resp := make(sdk.Response)
  resp["output"] = "Hello World"
  resp["request"] = req

  if err, ok := req["error"]; ok {
    errString, _ := err.(string)
    switch errString {
    case "execute_again":
      if counter > 1 {
        break
      }
      return nil, sdk.NewErrExecuteAgain(errString, map[string]any{
        "rkey":    "rvalue",
        "counter": counter + 1,
      })
    case "transient":
      return nil, sdk.NewErrTransient(errString)
    default:
      return nil, sdk.NewErrFailed(errString)
    }
  }

  return sdk.Response(resp), nil
}
```

Setting	**Description**
apiVersion	Rafay's API version eaas.envmgmt.io/v1
Kind	Driver
metadata	**name**: Name of the driver **project**: Project where the driver resides
spec -> config	**function, type should be function**
spec -> config ->	timeoutSeconds: 300 (indicates a single run should complete in 5mins else it will be declared as timeout)

	pollingConfig: repeat: "15s" (indicates poll function to be run every 15s) until: "1h" (indicates function to be run till 1h)
spec -> config ->	**function**: cpuLimitMilli: "50" (indicate cpuLimit for the container platform will run this function in) memoryLimitMi: "128" (indicate memoryLimit for the container platform will run this function in) language: go (supports only go or python) languageVersion: "1.22" maxConcurrency: 10 (max concurrent reqs handled by a single function container instance) numReplicas: 1 (number of replicas of the function container) **source**: | (inline code of the function)

Static Resources

A static resource refers to a pre-existing resource that can be referenced in templates. For example, it could be an existing database instance or any other service. There are two types of static resources:

1. **Static Environment**: Resources from an already created environment that are made available to the Environment Template
2. **Static Resource**: Predefined, unchangeable values, similar to constants. These values can be provided in formats such as HCL, JSON, expressions, or plain text

Sample **Static Resource** spec

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: Resource
metadata:
  name: static-resource
  project: madhu-demo-static
spec:
  variables:
  - name: eks_cluster_name
    options:
```

```
    description: fds
  value: clustertest-madhu-eks-demo2
  valueType: text
```

The static resource or environment can be referred to in the environment template under resources with kind as “Resource” or “Environment” as below.

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: EnvironmentTemplate
metadata:
  name: madhu-static-resource
  project: madhu-demo-static
spec:
  hooks: {}
  resources:
  - kind: resource
    name: static-resource
    type: static
 - kind: environment
    name: eks-cluster-for-static-env-resource
    type: static
 - kind: resourcetemplate
    name: rds-demo1-variables
    resourceOptions:
      version: newagent
    type: dynamic
```

Agents

Agents play a key role in accessing private repositories for artifacts (including code) and executing related tasks.

Agents can be deployed in two ways:

1. As a Kubernetes deployment within a cluster.
2. As a Docker container. In Docker mode, the container runs a lightweight K3s cluster where the agent is deployed.

All environment activities are executed on the agent.

- For container-based tasks, pods are created in the Kubernetes cluster, with logs streamed back to the workflow engine and the activity's exit status updated accordingly.
- HTTP activities are handled using an HTTP client.

Agents can be associated with:

- Environment Templates
- At environment launch or runtime
- At the resource template level

Although it is possible to associate agents with individual resource templates, this is not a common practice.

When agents are defined at multiple levels, such as both the environment template and an individual resource template, the agent specified in the higher-level environment template takes precedence.

Docker agent

```
Unset
apiVersion: gitops.k8smgmt.io/v3
kind: Agent
metadata:
  name: dockeragent
  project: defaultproject
spec:
  active: true
  type: Docker
  version: r2.10.0
```

Agent running on cluster

```
Unset
apiVersion: gitops.k8smgmt.io/v3
kind: Agent
metadata:
  name: k8sagent
  project: defaultproject
spec:
  active: true
  cluster:
    name: eks-agent
  type: Cluster
  version: r2.10.0
```

Repository

A repository serves as a storage location for the code or Infrastructure as Code (IaC) used by a Resource Template to create a resource. The Resource Template accesses the repository through an agent (cd-agent). Each Resource Template is linked to a specific repository.

```
Unset
apiVersion: integrations.k8smgmt.io/v3
kind: Repository
metadata:
  name: rauto-repo-216151
  project: defaultproject
spec:
  agents:
  - name: k8sagent
  credentials:
    password: sealed://credentials.password
    username: automation-user@rafay.co
  endpoint: https://github.com/RafaySystems/qa-automation-applications
  secret:
    name: file://artifacts/rauto-repo-216151/sealed-secret.yaml
  type: Git
```

Schedules

The **Scheduled Jobs** feature allows administrators to automate tasks such as deployments, destructions, or workflow executions at the Environment Template level using cron expressions. This feature provides flexibility for users to opt out of predefined schedules through configurable rules and approval workflows, ensuring efficient management of routine operations.

Example Use Cases

1. **Provision an Environment:** Automatically provision an environment at the beginning of each week, e.g., Monday at 6:00 AM
2. **Destroy an Environment:** Automatically destroy an environment at the end of the week, e.g., Friday at 10:00 PM
3. **Free Up Space:** Clear all attached volumes at a specified time

Environment Template Specification Example

The following example includes two schedules:

1. **First Schedule:** Configured with a Time-To-Live (TTL) value of 10 minutes. When the environment is deployed, the destruction schedule is triggered 10 minutes later.
2. **Second Schedule:** Configured with the cron expression 44 * * * *, which executes the schedule based on the defined time pattern.

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: EnvironmentTemplate
metadata:
  description: Environment Template for ec2 Resources
  name: rautoenv-tmp1-dash-220886
  project: defaultproject
spec:
  hooks: {}
  resources:
  - kind: resourcetemplate
    name: rautort1-dash-220886
    resourceOptions:
      version: version2-220886
    type: dynamic
  schedules:
  - cadence:
      timeToLive: 10m
    name: ttl
    optOutOptions:
      allowOptOut: true
      maxAllowedDuration: 2d
      maxAllowedTimes: 3
    type: destroy
  - cadence:
      cronExpression: 44 * * * *
      cronTimezone: Asia/Kolkata
    description: Destroy the ec2 instance
    name: destroy1-ec2-instance
    optOutOptions:
      allowOptOut: true
      maxAllowedDuration: 1d
      maxAllowedTimes: 5
    type: destroy
  version: v33
  versionState: active
```

Setting	Description
apiVersion	Rafay's API version eaas.envmgmt.io/v1
Kind	EnvironmentTemplate
spec -> schedules	***Required*** • name (String) name of the schedule • type (String) schedule type, Available options are deploy, destroy, workflows. • cadence (Block List, Max: 1) Configure a schedule cadence at which a job would automatically trigger ***Optional*** • description (String) Description of the schedule • context (Block List, Max: 1) Input data configuration that are needed as part of this schedule run • workflows (Block List, Max: 1) Name of the custom workflow provider that needs to be executed with this job • opt_out_options (Block List, Max: 1) Opt Out Options configured with this schedule **Schema for cadence :** • cron_expression (String) Cron expression used to configure scheduling jobs • cron_timezone (String) Specify the timezone of cron expression • time_to_live (String) Specify the maximum time to live duration of an environment, time units are 'h', 'd' e.g. 8h, 2d. User can either choose cron expression or time_to_live **Scheme for outputs:** • allow_opt_out (Bool) Specify whether users can opt out from this schedule ***Optional*** • max_allowed_duration (String) Specify the maximum allowed opt out duration, time units are 'm', 'h', 'd' e.g. 8h, 2d • max_allowed_times (Number) Specify the maximum number of times users can opt out without approval e.g. users can max opt out of this schedule thrice • approval (Block List, Max: 1) Details of approval workflow that needs to be execution in case of user opt-out

Environment

Environment is an instance of an Environment template, analogy being, instance of a class.

Setting	Description
apiVersion	Rafay's API version eaas.envmgmt.io/v1
Kind	Environment
Metadata	**description**: Description or the purpose of the environment template resource. Though it is optional, it should be entered as it is shown on the environment card. **name**: Name of the environment template **project**: Project name where the environment template resides
spec -> agents	Should be set to the name of the agent which should be used for running the code of the resource template
spec -> variables	List of input variables have to be mentioned here name -> name of the variables, these can be variables that are allowed to be overridden or new variable names (if allowed) value -> single value sensitive -> true means the value will be encrypted for storage
spec -> files	List of files, these are ones that can be overridden or new files name -> name of the file or path to the file data -> content of the file in base64 encoded format
spec -> envs	List of env variables, these are ones that can be overridden or new env variables key -> name of the variable value -> either plain text OR reference to encrypted data like sealed://AWS_SECRET_ACCESS_KEY If sensitive is set to true, follow instructions here to create and use secretsealer to encrypt data: https://docs.rafay.co/integrations/secretseal/gitrepo/
spec -> template	Template name and version with which environment is published
spec -> schedule_optouts	Request to opt out of schedules, this may be honored or not depending on the template configurations and approvals ***Required***

	• name (String) Name of the schedule to be opted out from ***Optional*** • duration (String) Requested opt out duration

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: Environment
metadata:
  name: rauto-environment-eks-ec-rds-name-65837
  project: defaultproject
spec:
  schedule_optouts:
  - duration: 71m
    name: destroy1-ec2-instance
  agents:
  - name: k8sagent1
  secret:
    name: file://artifacts/rauto-environment-eks-ec-rds-name-65837/sealed-secret.yaml
  template:
    name: rauto-environment-eks-ec-rds-template-name-65837
    version: custom-noagent
  variables:
  - name: db_password
    options:
      description: db_password
      override:
        type: allowed
      required: true
      sensitive: true
    value: sealed://variables.0
    valueType: text
  - name: aws_cloud_provider_access_key
    options:
      description: aws_access_key
      override:
        type: allowed
      required: true
      sensitive: true
    value: sealed://variables.1
    valueType: text
  - name: aws_cloud_provider_secret_key
    options:
      sensitive: true
    value: sealed://variables.2
```

```
    valueType: text
  - name: eks_cluster_region
    value: us-west-2
    valueType: text
  - name: eks_cluster_az
    value: '["us-west-2a", "us-west-2b"]'
    valueType: json
  - name: eks_cluster_node_group_az
    value: '["us-west-2a"]'
    valueType: json
  - name: eks_cluster_name
    value: madhu-eks-fmac
    valueType: text
  - name: region
    value: us-west-2
    valueType: text
```

Expressions

The **Expressions** feature enhances the Environment Manager's flexibility and power by enabling dynamic input configuration. It currently supports writing expressions using Starlark and Cue Language.

Expressions allow users to dynamically configure input variables by leveraging the outputs of other resources or configurations. This capability helps platform teams create programmable templates for provisioning infrastructure and applications.

Using resource outputs as inputs for other resources involves taking the output or data from one resource and utilizing it as input or configuration for another. For example, in the case of a Terraform resource template, any output variables defined in the OpenTofu/Terraform configuration represent the resource's outputs. These outputs can then be used to configure other resources. Currently, this functionality supports only OpenTofu/Terraform.

Example

Below is an example of a VPC output containing values that can be used as inputs for other resources:

```
output "vpc_id" {
  value = module.vpc.vpc_id
}
```

Application deployments

Consider an example of an environment named **env1**, created using an environment template called **envtemp**. This environment includes resources such as a VPC, RDS, and a Rafay-managed Kubernetes (K8s) cluster, with both the K8s cluster and RDS depending on the VPC.

A developer or data scientist who wants to deploy an application to the K8s cluster can avoid hardcoding the workload or application manifest with specific environment details. Instead, they can utilize the following expression:

```
$(#environment["resource_name"]["varname"])$
```

Let's assume that the application needs to include details of the RDS instance. Below is an example snippet of the output of the RDS resource template.

```
output "rds_hostname" {
  description = "RDS instance hostname"
  value       = aws_db_instance.db.address
  sensitive   = true
}
```

For this specific output example, the workload expression would be:

```
$(#environment["rds"]["rds_hostname"])$
```

Below is an example of a workload Helm value file with expressions, where RDS output values are passed as input values.

```
env:
  celeryBrokerUrl:
'redis://$(#environment["ec-resoource-tmpl"].configuration_endpoint_address):6379/0'
  celeryResultBackend:
'redis://$(#environment["ec-resoource-tmpl"].configuration_endpoint_address):6379/0'
  debug: "True"
  djangoDb: postgresql
  postgresHost: '$(#environment["rds-resource-tmpl"].rds_hostname)'
  postgresName: postgres
  postgresPassword: '$(#environment["rds-resource-tmpl"].rds_password)'
```

```
postgresPort: '$(#environment["rds-resource-tmpl"].rds_port)'
postgresUser: '$(#environment["rds-resource-tmpl"].rds_username)'
```

If the variable type is a list (array), you can use indexing to reference specific values. For instance, to set up a load balancer with multiple IPs, you can use the following expression:

```
$(resource."resource_name".output.otest1.value)$
```

Dynamic and Static Resources

In some scenarios, the output of a dynamic or static resource may need to be passed to another resource during the environment creation process. The following expression can be used for such cases:

```
$(resource."resource_name".output.otest1.value)$
```

Consider a scenario where a resource template includes dynamic resources, such as a VPC and RDS, and you need to ensure the RDS is installed within a specific VPC by passing the VPC ID to the RDS. This can be achieved by using the following expression to pass the VPC's output to the RDS:

```
$(resource."vpc".output.vpc_id.value)$
```

The above expression references the output value 'vpc_id' from the 'vpc' resource.

Similarly to dynamic resources, static resource outputs can also be passed to another resource.

Static Environment Resource

To pass the output of a static environment resource to another resource, use the below expression:

```
$(resource."static-env-0".resource."resource-2".output.test3.value)$
```

Let's take the example where an environment named "env1" has already been created. If the user intends to create an additional environment with the "rds" resource on top of "env1" without modifying the existing "env1", use the expression outlined below:

```
$(resource."env1".resource."rds".output.rds_hostname.value)$
```

The above expression employs the 'env1' environment name, the static resource name 'rds', and references the output value labeled as 'rds_hostname'. The values can be of any type, such as array, map, and string.

Users can utilize this expression to include 'env1' as a static resource in the new environment alongside the RDS.

Dedicated Resource(s) Scenario

When creating an environment template, the user has the ability to add dynamic or static resources (pre-existing resources).

When the "dynamic" option is selected for resources, users are allowed to enable the dedicated option. When the dedicated option is enabled, dedicated resources are only brought up when a workload is published to that environment.

To use the workload attributes as part of the dedicated resource(s), use the following expression:

```
$(workload.name)$
```

(or)

```
$(workload.id)$
```

If ElastiCache is a dedicated resource that needs to be brought up for a workload, the user can use the workload name $(workload.name)$ as input to the name variable of the ElastiCache. So, whenever a workload is published, the dedicated ElastiCache comes up with the name of the workload.

Note: If you are leveraging the Rafay Workloads, you need to ensure to select Environment Configuration based on the requirement, so that it can be used in the resource template. When the workload is unpublished/destroyed, the dedicated resources within that workload will also be destroyed.

Environment Resource(s)

Assume a scenario where a user attempts to set up three to four environments using an environment template, and all the resources are being launched in the same AWS region. To prevent naming conflicts, utilize the following expression to create them with dynamic names and IDs.

```
$(environment.name)$
$(environment.id)$
$(environment.project.name)$
```

```
$(environment.project.id)$
$(environment.labels.key)$
```

Resource Artifact Variables

There are instances where the user intends to download **Terraform** working directory at different stages such as Init, Plan (pre-hook & post-hook), Apply, and Output. Users can download the required artifacts via container hooks using the expression below (from the working directory as a token and url). This enables users to access the artifacts and perform necessary actions such as scanning of terraform code, calculate the cost of the infrastructure etc.

You can use the expressions below to download artifacts:

Artifact Activity

```
$(resource."rtdependson-0".artifact.workdir.token)$
$(resource."rtdependson-0".artifact.workdir.url)$
```

Plan Activity

```
$(resource."rtdependson-0".plan.workdir.token)$
$(resource."rtdependson-0".plan.workdir.url)$
```

Trigger Expressions

Trigger expressions are used to enable dynamic event-driven responses within systems by allowing users to leverage the output of trigger events as input for subsequent actions or workflows.

Use the below trigger expressions to dynamically configure actions and workflows based on event triggers.

- trigger.payload.is_sso_user: To check if the user logged in using Single Sign-On credentials
- trigger.payload.userid: Retrieves the user's identification, typically their email address
- trigger.payload.username: Fetches the username, which is usually the user's email address
- trigger.payload.type (type: destroy, force-destroy, force-release-lock): Identifies the type of action triggered, such as deletion or lock release

- trigger.id: Provides the unique identifier for the trigger event
- trigger.payload: Represents all data associated with the trigger event
- trigger.reason: Captures the reason behind the trigger event

Examples

- When users initiate resource deletion actions, such as "destroy" or "force-destroy," the trigger expression trigger.payload.type identifies the type of deletion request. Based on the detected type, the system dynamically initiates automated cleanup processes, ensuring that associated resources are properly released and decommissioned
- When a user attempts to log in to the controller, the trigger expression trigger.payload.is_sso_user evaluates whether the user has authenticated using the Single Sign-On mechanism. If the user is an SSO user, expression returns true

Types of Expressions

Three types of expressions are supported, allowing users to retrieve values from outputs based on their complexity.

1. cue expressions syntax is $(expression)$
2. starlark expressions syntax is #{expression}#
3. JSONPath expressions syntax is @{expression}@

Assume following is the environment output data available for expressions.


```
{
  "resource": {
    "rt-wf": {
      "task": {
        "crucial-task": {
          "output": {
            "host": "eaas.io"
          }
        }
      }
    },
    "rt-1": {
      "output": {
        "workdir": {
          "url": "http://chisel-127-0-0-1.nip.io:58508/download/job.tar.zst",
          "token": "******"
        },
        "files": {
          "job.tar.zst": {
            "url": "http://chisel-127-0-0-1.nip.io:58508/download/job.tar.zst",
            "token": "******"
          },
          "stdout": {
```

```
      "url": "http://chisel-127-0-0-1.nip.io:58508/download/stdout",
      "token": "******"
    }
  },
  "create_id": {
    "sensitive": false,
    "type": "string",
    "value": "4713029731612950118"
  },
  "destroy_id": {
    "sensitive": false,
    "type": "string",
    "value": "7718207542795149585"
  },
  "otest1": {
    "sensitive": false,
    "type": "string",
    "value": "edependson-0"
  },
  "otest2": {
    "sensitive": false,
    "type": "string",
    "value": "default-test2"
  },
  "otest3": {
    "sensitive": false,
    "type": "string",
    "value": "default-test3"
  },
  "otest4": {
    "sensitive": false,
    "type": "string",
    "value": "default-test4"
  },
  "otestall": {
    "sensitive": false,
    "type": "string",
    "value": "edependson-0, default-test2, default-test3, default-test4"
  }
},
"artifact": {
  "workdir": {
    "url": "http://chisel-127-0-0-1.nip.io:58508/download/job.tar.zst",
    "token": "******"
  }
},
"init": {
  "workdir": {
    "url": "http://chisel-127-0-0-1.nip.io:58508/download/job.tar.zst",
    "token": "******"
  }
},
```

```
    "plan": {
      "workdir": {
        "url": "http://chisel-127-0-0-1.nip.io:58508/download/job.tar.zst",
        "token": "*****"
      }
    },
    "apply": {
      "workdir": {
        "url": "http://chisel-127-0-0-1.nip.io:58508/download/job.tar.zst",
        "token": "*****"
      }
    }
  }
},
"trigger": {
  "id": "01H77ZN0B7HV9G2V8JJ6EG9X7G",
  "createdAt": "2023-08-07T12:30:48.167581Z",
  "modifiedAt": "2023-08-07T12:30:48.167582Z",
  "scope": {
    "parent": {
      "kind": "environment",
      "id": "01H77ZN0882M7RX8HNWQCSX98K",
      "name": "env-1",
      "partnerId": "partner007",
      "organizationId": "organization007",
      "projectId": "project007"
    }
  },
  "payload": {
    "type": "deploy",
    "userid": "t$1d01",
    "username": "test-user"
  },
  "status": "TRIGGER_EVENT_STATUS_PENDING",
  "version": 2
},
"environment": {
  "labels": {
    "environment": "env-1"
  },
  "project": {
    "id": "project007",
    "name": "project007"
  },
  "organization": {
    "id": "organization007",
    "name": "organization007"
  },
  "partner": {
    "id": "partner007",
    "name": "partner007"
  },
```

```
    "name": "env-1",
    "id": "01H77ZN0882M7RX8HNWQCSX98K"
  },
  "repository": {
    "name": "repo1",
    "id": "rx28oml",
    "partner": {
      "id": "partner007"
    },
    "project": {
      "id": "project007"
    },
    "organization": {
      "id": "organization007"
    }
  },
  "workload": {
    "name": "workload1",
    "id": "rx28oml",
    "partner": {
      "id": "partner007"
    },
    "project": {
      "id": "project007"
    },
    "organization": {
      "id": "organization007"
    }
  }
}
```


Description	Cue	Starlark	JSONPath
dynamic resource's output value using object notation	$(resource."rt-1".output.otest1.value)$	#{resource."rt-1".output.otest1.value}#	@{resource."rt-1".output.otest1.value}@
dynamic resource's output value using index notation	$(resource["rt-1"].output.otest1.value)$	#{resource["rt-1"].output.otest1.value}#	@{resource["rt-1"].output.otest1.value}@
static resource output value	$(resource["static-rt-1"].output.otest1.value)$	#{resource["static-rt-1"].output.otest1.value}#	@{resource["static-rt-1"].output.otest1.value}@

static environment output value	$(resource["static-env-0"].resource["rt-0"].output.otest2.value)$	#{resource["static-env-0"].resource["rt-0"].output.otest2.value}#	@{resource["static-env-0"].resource["rt-0"].output.otest2.value}@
resource artifacts from artifact activity	$(resource["rt-1"].artifact.workdir.token)$ $(resource["rt-1"].artifact.workdir.url)$	#{resource["rt-1"].artifact.workdir.token}# #{resource["rt-1"].artifact.workdir.url}#	@{resource["rt-1"].artifact.workdir.token}@ @{resource["rt-1"].artifact.workdir.url}@
resource artifacts from output activity	$(resource["rt-1"].output.workdir.token)$ $(resource["rt-1"].output.workdir.url)$ $(resource["rt-1"].output.files.stdout.url)$ $(resource["rt-1"].output.files.stdout.token)$	#{resource["rt-1"].output.workdir.token}# #{resource["rt-1"].output.workdir.url}# #{resource["rt-1"].output.files.stdout.url}# #{resource["rt-1"].output.files.stdout.token}#	@{resource["rt-1"].output.workdir.token}@ @{resource["rt-1"].output.workdir.url}@ @{resource["rt-1"].output.files.stdout.url}@ @{resource["rt-1"].output.files.stdout.token}@
environment details	$(environment.name)$ $(environment.id)$ $(environment.project.name)$ $(environment.project.id)$ $(environment.labels.key)$	#{environment.name}# #{environment.id}# #{environment.project.name}# #{environment.project.id}# #{environment.labels.key}#	@{environment.name}@ @{environment.id}@ @{environment.project.name}@ @{environment.project.id}@ @{environment.labels.key}@
workload details(for workload trigger)	$(workload.name)$ $(workload.id)$	#{workload.name}# #{workload.id}#	@{workload.name}@ @{workload.id}@
repository details(for repository trigger)	$(repository.name)$ $(repository.id)$	#{repository.name}# #{repository.id}#	@{repository.name}@ @{repository.id}@
custom resource's output's value	$(resource["rt-wf"].task["crucial-task"].output.host)$	#{resource["rt-wf"].task["crucial-task"].output.host}#	@{resource["rt-wf"].task["crucial-task"].output.host}@

environment lifecycle hook outputs	$(environment.hook.onInit.hook_name.output)$ $(environment.hook.onSuccess.hook_name.output)$ $(environment.hook.onFailure.hook_name.output)$ $(environment.hook.onCompletion.hook_name.output)$	#{environment.hook.onInit.hook_name.output}# #{environment.hook.onSuccess.hook_name.output}# #{environment.hook.onFailure.hook_name.output}# #{environment.hook.onCompletion.hook_name.output}#	@{environment.hook.onInit.hook_name.output}@ @{environment.hook.onSuccess.hook_name.output}@ @{environment.hook.onFailure.hook_name.output}@ @{environment.hook.onCompletion.hook_name.output}@
resource lifecycle hook outputs	$(resource.resource_name.hook.onInit.hook_name.output)$ $(resource.resource_name.hook.onSuccess.hook_name.output)$ $(resource.resource_name.hook.onFailure.hook_name.output)$ $(resource.resource_name.hook.onCompletion.hook_name.output)$	#{resource.resource_name.hook.onInit.hook_name.output}# #{resource.resource_name.hook.onSuccess.hook_name.output}# #{resource.resource_name.hook.onFailure.hook_name.output}# #{resource.resource_name.hook.onCompletion.hook_name.output}#	@{resource.resource_name.hook.onInit.hook_name.output}@ @{resource.resource_name.hook.onSuccess.hook_name.output}@ @{resource.resource_name.hook.onFailure.hook_name.output}@ @{resource.resource_name.hook.onCompletion.hook_name.output}@
resource provider's deploy lifecycle hook outputs	$(resource.template_name.hook.deploy.init.before.hook_name.output)$ $(resource.template_name.hook.deploy.init.after.hook_name.output)$ $(resource.template_name.hook.deploy.plan.before.hook_name.output)$ $(resource.template_name.hook.deploy.plan.after.hook_name.output)$ $(resource.template_name.hook.deploy.apply.before.hook_name.output)$ $(resource.template_name.hook.deploy.apply.after.hook_name.output)$ $(resource.template_name.hook.deploy.output.before.hook_name.output)$ $(resource.template_name.hook.deploy.output.after.hook_name.output)$	#{resource.template_name.hook.deploy.init.before.hook_name.output}# #{resource.template_name.hook.deploy.init.after.hook_name.output}# #{resource.template_name.hook.deploy.plan.before.hook_name.output}# #{resource.template_name.hook.deploy.plan.after.hook_name.output}# #{resource.template_name.hook.deploy.apply.before.hook_name.output}# #{resource.template_name.hook.deploy.apply.after.hook_name.output}# #{resource.template_name.hook.deploy.output.before.hook_name.output}# #{resource.template_name.hook.deploy.output.after.hook_name.output}#	@{resource.template_name.hook.deploy.init.before.hook_name.output}@ @{resource.template_name.hook.deploy.init.after.hook_name.output}@ @{resource.template_name.hook.deploy.plan.before.hook_name.output}@ @{resource.template_name.hook.deploy.plan.after.hook_name.output}@ @{resource.template_name.hook.deploy.apply.before.hook_name.output}@ @{resource.template_name.hook.deploy.apply.after.hook_name.output}@ @{resource.template_name.hook.deploy.output.before.hook_name.output}@ @{resource.template_name.hook.deploy.output.after.hook_name.output}@

resource provider's destroy lifecycle hook outputs	$(resource.template_name.hook.destroy.init.before.hook_name.output)$ $(resource.template_name.hook.destroy.init.after.hook_name.output)$ $(resource.template_name.hook.destroy.plan.before.hook_name.output)$ $(resource.template_name.hook.destroy.plan.after.hook_name.output)$ $(resource.template_name.hook.destroy.destroy.before.hook_name.output)$ $(resource.template_name.hook.destroy.destroy.after.hook_name.output)$	#{resource.template_name.hook.destroy.init.before.hook_name.output}# #{resource.template_name.hook.destroy.init.after.hook_name.output}# #{resource.template_name.hook.destroy.plan.before.hook_name.output}# #{resource.template_name.hook.destroy.plan.after.hook_name.output}# #{resource.template_name.hook.destroy.destroy.before.hook_name.output}# #{resource.template_name.hook.destroy.destroy.after.hook_name.output}#	@{resource.template_name.hook.destroy.init.before.hook_name.output}@ @{resource.template_name.hook.destroy.init.after.hook_name.output}@ @{resource.template_name.hook.destroy.plan.before.hook_name.output}@ @{resource.template_name.hook.destroy.plan.after.hook_name.output}@ @{resource.template_name.hook.destroy.destroy.before.hook_name.output}@ @{resource.template_name.hook.destroy.destroy.after.hook_name.output}@
workload expressions to refer resource's output	$(#environment["rt-1"]["otest1"])$	#{#environment["rt-1"]["otest1"]}#	@{#environment["rt-1"]["otest1"]}@
drivers can use input variables as expressions	$(current.input["variable-name"])$	#{current.input["variable-name"]}#	@{current.input["variable-name"]}@
driver output as expressions	$(current.output["variable-name"])$	#{current.output["variable-name"]}#	@{current.output["variable-name"]}@

The expressions mentioned above are straightforward and easy to use. However, there may be instances where users need to query more complex JSON outputs.

For example, consider the JSON structure provided below as part of a resource output. Let's explore how to write more advanced queries to extract the desired results.

```
{
  "resource": {
    "rafay-vms": {
      "task": {
        "task1": {
          "output": {
            "vms": [
              {
                "id": "vm-001",
```

```
    "name": "web-server-1",
    "type": "t2.micro",
    "region": "us-east-1",
    "status": "running",
    "ip_address": "192.168.1.10",
    "cpu_cores": 1,
    "memory_size_mb": 1024,
    "disk_size_gb": 30,
    "tags": {
      "environment": "production",
      "role": "web"
    }
  },
  {
    "id": "vm-002",
    "name": "web-server-2",
    "type": "t2.micro",
    "region": "us-east-1",
    "status": "stopped",
    "ip_address": "192.168.1.11",
    "cpu_cores": 1,
    "memory_size_mb": 1024,
    "disk_size_gb": 30,
    "tags": {
      "environment": "dev",
      "role": "web"
    }
  },
  {
    "id": "vm-003",
    "name": "app-server-1",
    "type": "t3.medium",
    "region": "us-east-1",
    "status": "running",
    "ip_address": "192.168.1.12",
    "cpu_cores": 2,
    "memory_size_mb": 4096,
    "disk_size_gb": 50,
    "tags": {
      "environment": "production",
      "role": "application"
    }
  },
  {
    "id": "vm-004",
    "name": "db-server-1",
    "type": "t3.large",
    "region": "us-east-1",
    "status": "running",
    "ip_address": "192.168.1.13",
    "cpu_cores": 2,
    "memory_size_mb": 8192,
```

```
            "disk_size_gb": 100,
            "tags": {
              "environment": "dev",
              "role": "database"
            }
          }
        ]
      }
    }
  }
 }
}
```

Let's assume x = resource."rafay-vms".task.task1.output, so it's easier for readability, please replace x with resource."rafay-vms".task.task1.output when using expressions.

Description	Cue	Starlark	JSONPath
Get the i^{th} VM object from the list, replace it with the number in the expression (starts from 0)	$(x.vms[i])$	#{x.vms[i]}#	@{x.vms[i]}@
Get list of all VMs	$(x.vms)$	#{x.vms}#	@{$.x.vms}@ @{$.x.vms[*]}@ //wildcard operator is supported
Get list of all VM names	$([for vm in x.vms {vm.name}])$	#{[vm.name for vm in vms]}#	@{$..name}@ @{$.vms[*].name}@
Get list of all VMs that are of type t2.micro	$([for vm in x.vms if vm.type == "t2.micro" {vm}])$	#{[vm for vm in x.vms if vm.type == "t2.micro"]}#	@{$.x.vms[?(@.type == "t2.micro")]}@
Get list of all VM ids that are in running status	$([for vm in x.vms if vm.status == "running" {vm.id}])$	#{[vm.id for vm in x.vms if vm.status == "running"]}#	@{$.x.vms[?(@.status == "running")].id}@

Get list of all VMs that are in running status in dev environment	$([for vm in x.vms if vm.status == "running" && vm.tags.environment == "dev" {vm}])$	#{[vm for vm in x.vms if vm.status == "running" and vm.tags.environment == "dev"]}#	@{$.x.vms[?(@.status == "running" && @.tags.environment == "dev")]}@
Get only the first VM that is of type t2.micro	$([for vm in x.vms if vm.type == "t2.micro" {vm}][0])$	#{[vm for vm in x.vms if vm.type == "t2.micro"][0]}#	N/A
Get map of all VMs with ids as keys and VM objects as values	$({for vm in x.vms {"\(vm.id)": vm}})$	#{{vm.id: vm for vm in x.vms}}#	N/A

Note:

- If the JSONPath expression begins with @{$.expression}@, the result will always be returned as a list, regardless of its length. This is the default behavior of JSONPath expressions
- If the JSONPath expression begins with @{expression}@, the result will be returned as an object if the result length is 1; otherwise, it will be returned as a list

Scripts

Users now have the flexibility to write scripts in addition to expressions, depending on the complexity of their requirements. Both scripts and expressions can be stored in variables, environment variables, or within file contents. Scripts are evaluated first, followed by expressions. This allows users to generate expressions within scripts that can be evaluated later.

The Environment Manager supports two types of scripts:

1. **Cue Script**
2. **Starlark Script**

Cue

A Cue script must begin with the syntax $$ script begin $$ and end with $$ script end $$ to define the code that the workflow engine will execute. Cue language does not support functions; it consists only of statements.

Cue variables defined within these statements are extracted as JSON output, whereas private Cue variables indicated by a prefix of # or _ are excluded from the output. Additionally, system variables introduced by the workflow engine, which also start with #, are treated as private and are not included in the output.

Refer here for more details: CUE

Example below:

```
$$ script begin $$
#activities: {for name, activity in #ctx.activities {"\(name)": activity}}
// this will generate the JSON array of all outputs of all activities combined.
// ["output1", "output2", "output3"]
[for name, _ in #activities for k, _ in #activities[name].output if k != "files" {k}]
// or, you can use the below statement to capture it as JSON object
// this will generate the JSON object of all outputs of all activities combined.
// {"all_outputs": ["output1", "output2", "output3"]}
all_outputs: [for name, _ in #activities for k, _ in #activities[name].output if k != "files" {k}]
$$ script end $$
```

Starlark

Starlark is a programming language that is a dialect of Python, making it easy for users to write scripts without needing to learn a new language. Scripts must begin with the syntax ## script begin ## and end with ## script end ##.

The workflow engine invokes a function named eval(**kwargs) with kwargs as an argument, where kwargs is a key-value dictionary. Users can define custom functions within the script and call them from the eval function.

The eval function must return a value of a primitive data type, a list, or a dictionary. These returned values are then utilized as input variables for the Environment Manager.

Refer here for more details: bazelbuild/starlark

Example below:

```
## script begin ##
# get_outputs is a function which gets the list of all outputs of all activities combined.
def get_outputs(ctx):
  variables = []
  for activity_name in ctx["activities"]:
    outputs = ctx["activities"][activity_name].get("output", {})
    for output in sorted(outputs):
      if output == "files":
        continue
      variables.append(output)
  return variables
# eval is mandatory, calls get_outputs, returns the list, this is then converted to JSON array in the worklfow engine.
def eval(**kwargs):
  return get_outputs(kwargs["ctx"])
# or
# eval function calls get_outputs, returns the dict, this is then converted to JSON object in the worklfow engine.
def eval(**kwargs):
```

```
    return {"all_outputs": get_outputs(kwargs["ctx"])}
## script end ##
```

Building Environment Templates

This section offers examples for designing and developing environment templates using the Rafay Environment Manager framework. These examples showcase the framework's capabilities, enabling users to create complex environments tailored to the organization's workflow requirements.

Example Templates

Building a Single Resource Template

We will start off building an environment template to create a VPC in an AWS region. This approach will leverage the Rafay's GitOps, System sync feature to load all the templates from the Git to the Rafay's controller.

Prerequisites for GitOps first approach of development/testing the template :

1. Access to the Rafay Console through an Org
2. User API/Token if you use APIs for repository, pipeline and agent creation
3. Creation of the following (one-time exercise) using one of Rafay's interfaces
    a. An Agent instance to run the code that's backing the resource templates

    Refer to Rafay RCTL documentation for pipeline creation and management

    b. Create repository with name **envmgr**, your GitHub repository and branch details in Rafay console with Github Account's username and Token

    Refer to Rafay RCTL documentation for repository creation and management

    c. Create a secret
    d. Gitops pipeline instance in organization -> project for loading entities from **git to system** (organization -> project)

    Refer to Rafay RCTL documentation for pipeline creation and management

In this example, all of the building blocks including templates, config context, agents will be part of the "dev-project" project.

We will be creating the below folder structure in the git in the branch ***dev-guide-templates***.

Code structure in repository “envmgr”

```
Unset
├── rafay-resources
│   ├── projects
│   │   ├── dev-project
│   │   │   ├── configcontexts
│   │   │   │   ├── artifacts/rafay-config-context/sealed-secret.yaml
│   │   │   │   ├── RafayConfigContext.yaml
│   │   │   ├── environmenttemplates
│   │   │   │   ├── EnvironmentTemplate.yaml
│   │   │   ├── resourcetemplates
│   │   │   │   ├── VPCResourceTemplate.yaml
├── terraform
│   ├── vpc
│   │   ├── main.tf
│   │   ├── output.tf
│   │   ├── provider.tf
│   │   ├── variable.tf
```

Building Config Context with secrets data

- Install the Kubeseal utility in the system
- Download the Secret Sealer Certificate
- Create a secret yaml file (that contains sensitive data to be encrypted) like below

```
Unset
apiVersion: v1
data:
 RCTL_API_KEY: <base64 encoded value of RCTL_API_KEY>
 aws_access_key_id: <base64 encoded value of aws assess key>
 aws_secret_access_key: <base64 encoded value of aws secret key>
kind: Secret
metadata:
 name: rafay-config-context
 namespace: default
```

- Run the command to generate encrypted data

```
Unset
kubeseal --cert CERT--format yaml < secret.yaml > sealsecret.yaml
```

CERT: Certificate downloaded from the Secret Sealer
secret.yaml: Kubernetes secret file
sealsecret.yaml: The generated output file name with encrypted values

- On successful command execution, users can view the generated output file **sealsecret.yaml** where the data is encrypted/sealed

Below is the generated output file which needs to be copied under ***rafay-resources/projects/dev-project/configcontext/artifacts/rafay-config-context/sealed-secret.yaml***

```
Unset
apiVersion: bitnami.com/v1alpha1
kind: SealedSecret
metadata:
creationTimestamp: null
name: rafay-config-context
namespace: default
spec:
encryptedData:
 RCTL_API_KEY: <encrypted data of RCTL_API_KEY>
 aws_access_key_id: <encrypted data of aws_access_key_id>
 aws_secret_access_key: <encrypted data of aws_secret_access_key>
template:
  data: null
  metadata:
    creationTimestamp: null
    name: rafay-config-context
    namespace: default
```

rafay-resources/projects/dev-project/configcontext/RafayConfigContext.yaml

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: ConfigContext
metadata:
  name: rafay-config-context
  project: dev-project
spec:
  envs:
   # Here we are defining the RCTL config in config context, to be used in multiple environment templates
    - key: RCTL_API_KEY
      options:
        sensitive: true
      value: sealed://RCTL_API_KEY
      valueType: text
```

```
    - key: RCTL_REST_ENDPOINT
      value: console.rafay.dev
      valueType: text
    - key: RCTL_PROJECT
      value: dev-project
      valueType: text
  variables:
    # Here we are defining the AWS credentials in config context, to be used in multiple environment
templates
    - name: aws_access_key_id
      valueType: text
      options:
        sensitive: true
        override:
          type: allowed
      value: sealed://aws_access_key_id
    - name: aws_secret_access_key
      valueType: text
      options:
        sensitive: true
        override:
          type: allowed
      value: sealed://aws_secret_access_key
    - name: aws_region
      valueType: text
      value: us-west-2
      options:
        override:
          type: allowed
  secret:
    name: file://artifacts/rafay-config-context/sealed-secret.yaml
```

Rafay-resources/projects/dev-project/enviromenttemplates/EnvironmentTemplate.yaml

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: EnvironmentTemplate
metadata:
  name: app-environment-template
  project: dev-project
spec:
  # Here we are associating an agent which will be used for running code of all involved resource
templates, in this case being VPCResourcetemplate
  agents:
    - name: dev-agent
  contexts:
```

```
    # Here we are associating a configcontext that contains information related to AWS credentials. Reason a configcontext is used here , so that the same can be associated to multiple other env templates
    - name: rafay-config-context
  resources:
  # Here we are associating first resource template VPCResourceTemplate and its version.
  - kind: resourcetemplate
    name: vpc-resource-template
    resourceOptions:
      version: v1
    type: dynamic
  version: v1
  versionState: active
```

Rafay-resources/projects/dev-project/resourcetemplate/VPCResourceTemplate.yaml

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: ResourceTemplate
metadata:
  name: vpc-resource-template
  project: dev-project
spec:
  # Here we are defining provider as opentofu to use, if no providerOptions is mentioned then Rafay will use latest version
  provider: opentofu
  providerOptions:
    openTofu:
      refresh: true
      backendType: system
  repositoryOptions:
    # Here we are associating code in repository to the resource template
    name: envmgr
    branch: dev-guide-templates
    directoryPath: development-guide-templates/terraform/vpc
  variables:
    - name: vpc_name
      valueType: text
      value: example-vpc
      options:
        description: VPC name
        override:
          type: allowed
    - name: vpc_cidr
      valueType: text
      value: 10.0.0.0/16
      options:
        description: CIDR block
```

```
      override:
        type: allowed
  version: v1
```

terraform/vpc/main.tf

```
Unset
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"

  name = var.vpc_name
  cidr = var.vpc_cidr

  azs             = ["${var.aws_region}a", "${var.aws_region}b", "${var.aws_region}c"]
  private_subnets = [cidrsubnet(var.vpc_cidr, 8, 1), cidrsubnet(var.vpc_cidr, 8, 2), cidrsubnet(var.vpc_cidr, 8,
3)]
  public_subnets  = [cidrsubnet(var.vpc_cidr, 8, 100), cidrsubnet(var.vpc_cidr, 8, 101),
cidrsubnet(var.vpc_cidr, 8, 102)]

  enable_nat_gateway     = true
  single_nat_gateway     = true
  one_nat_gateway_per_az = false

  tags = var.tags
}
```

terraform/vpc/output.tf

```
Unset
output "vpc_name" {
  value = module.vpc.name
}

output "vpc_id" {
  value = module.vpc.vpc_id
}

output "private_subnets" {
  value = module.vpc.private_subnets
}

output "public_subnets" {
  value = module.vpc.public_subnets
}
```

terraform/vpc/provider.tf

```
Unset
provider "aws" {
  region     = var.aws_region
  access_key = var.aws_access_key_id
  secret_key = var.aws_secret_access_key
}
```

terraform/vpc/variable.tf

```
Unset
variable "vpc_name" {
  description = "VPC name"
  default     = "example-vpc"
  type        = string
}

variable "vpc_cidr" {
  description = "CIDR block"
  type        = string
  default     = "10.0.0.0/16"
}

variable "aws_region" {
  description = "AWS region"
  type        = string
  default     = "us-west-2"
}

variable "tags" {
  description = "AWS Tags"
  type        = map(string)
  default = {
    "env"   = "qa"
    "email" = "test@rafay.co"
  }
}

variable "aws_access_key_id" {
  description = "aws access key id"
  default     = ""
  sensitive   = true
}

variable "aws_secret_access_key" {
  description = "aws secret key"
  default     = ""
  sensitive   = true
```

```
}
```

Loading and launching the environment

1. Run the pipeline through UI or RCTL to sync the templates from GitHub repo to project in your organization
2. You will see all of the above entities: env template, resource template and the config context in the dev-project
3. You can select the EnvironmentTemplate and launch an instance of the environment by providing region info
4. The output of the VPC created as part of this run will show up in your activity panel in UI

Congratulations! , With this, now you have successfully created a template for creating VPC

Building a Multi-Resource Template with context variables and output

We will now evolve the above template to also

1. Provision EKS cluster onto the created VPC
2. Provision a namespace onto the EKS cluster

We will now add the following resource templates

- EKS resource template
- Namespace resource template

Code structure in repository with the new templates

```
Unset
├── Rafay-resources
│   ├── projects
│   │   ├── dev-project
│   │   │   ├── configcontexts
│   │   │   │   ├── artifacts/rafay-config-context/sealed-secret.yaml
│   │   │   │   ├── RafayConfigContext.yaml
│   │   │   ├── environmenttemplates
│   │   │   │   ├── EnvironmentTemplate.yaml
│   │   │   ├── resourcetemplates
│   │   │   │   ├── VPCResourceTemplate.yaml
│   │   │   │   ├── EKSResourceTemplate.yaml
│   │   │   │   ├── NamespaceResourceTemplate.yaml
├── terraform
│   ├── vpc
│   │   ├── main.tf
│   │   ├── output.tf
│   │   ├── provider.tf
```

```
│   │   ├── variable.tf
│   ├── eks
│   │   ├── main.tf
│   │   ├── output.tf
│   │   ├── provider.tf
│   │   ├── variable.tf
│   ├── namespace
│   │   ├── main.tf
│   │   ├── output.tf
│   │   ├── provider.tf
│   │   ├── variable.tf
```

Rafay-resources/projects/dev-project/enviromenttemplates/EnvironmentTemplate.yaml

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: EnvironmentTemplate
metadata:
  name: app-environment-template
  project: dev-project
spec:
  # Here we are associating an agent which will be used for running code of all involved resource
templates, in this case being VPCResourcetemplate
  agents:
    - name: dev-agent
  contexts:
    # Here we are associating a configcontext that contains information related to AWS credentials. Reason
a configcontext is used here , so that the same can be associated to multiple other env templates
    - name: rafay-config-context
  resources:
  # Here we are associating first resource template VPCResourceTemplate and its version.
  - kind: resourcetemplate
    name: vpc-resource-template
    resourceOptions:
      version: v1
    type: dynamic
  # Here we are associating second resource template EKSResourceTemplate and its version with the
dependency on VPCResourceTemplate.
  - kind: resourcetemplate
    name: eks-resource-template
    resourceOptions:
      version: v1
    type: dynamic
    dependsOn:
      - name: vpc-resource-template
```

```
  # Here we are associating third resource template NamespaceResourceTemplate and its version with the
dependency on EKSResourceTemplate.
  - kind: resourcetemplate
    name: namespace-resource-template
    resourceOptions:
      version: v1
    type: dynamic
    dependsOn:
      - name: eks-resource-template
  variables:
    # Here we are defining input collection variable to collect cluster name from end user of the template.
    # Its type is set to allowed, so that end user at environment launch time can provide region info which is
free form text
    # Its marked required, so that it shows up as mandatory field in the UI
    # selector is user here to wire the input value collected for cluster name to the EKS resource template's
variable. This way, platform team gets to pre-set, restrict values at only one place , which is environment
template level before its shared with end developers/users
    - name: cluster_name
      valueType: text
      options:
        required: true
        override:
          type: allowed
# Here we are associating the driver for JiraAprpoval onInit hook
  hook:
    onInit:
      - name: approval
        type: driver
        options: {}
        onFailure: unspecified
        driver:
          name: jira-approval
  version: v2
  versionState: active
```

Note: Input definition at environment template level through variables is passed onto appropriate input variables of resource templates

Rafay-resources/projects/dev-project/resourcetemplate/EKSResourceTemplate.yaml

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: ResourceTemplate
metadata:
  name: eks-resource-template
  project: dev-project
spec:
```

```
  # Here we are defining provider as opentofu to use, if no providerOptions is mentioned then Rafay will use
latest version
  provider: opentofu
  providerOptions:
    openTofu:
      refresh: true
      backendType: system
  repositoryOptions:
    # Here we are associating code in repository to the resource template
    branch: dev-guide-templates
    directoryPath: development-guide-templates/terraform/eks
    name: envmgr
  variables:
    - name: cluster_name
      valueType: text
      value: EKSClusterName
      options:
        override:
          type: allowed
    # Here we using the output from the VPCResourceTemplate's private_subnets information and use it for
the eks_public_subnets.
    - name: eks_public_subnets
      valueType: expression
      value: $(resource."vpc-resource-template".output.public_subnets.value)$
      options:
        override:
          type: notallowed
    # Here we using the output from the VPCResourceTemplate's private_subnets information and use it for
the eks_private_subnets.
    - name: eks_private_subnets
      valueType: expression
      value: $(resource."vpc-resource-template".output.private_subnets.value)$
      options:
        override:
          type: notallowed
    - name: project
      valueType: text
      value: dev-project
      options:
        override:
          type: allowed
  version: v1
```

***Rafay-resources/projects/dev-project/resourcetemplate/*NamespaceResourceTemplate.yaml**

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: ResourceTemplate
metadata:
  name: namespace-resource-template
  project: dev-project
spec:
  # Here we are defining provider as opentofu to use, if no providerOptions is mentioned then Rafay will use
latest version
  provider: opentofu
  providerOptions:
    openTofu:
      refresh: true
      backendType: system
  # Here we are associating code in repository to the resource template
  repositoryOptions:
    branch: dev-guide-templates
    directoryPath: development-guide-templates/terraform/namespace
    name: envmgr
  variables:
    # Here we specify the project name.
    - name: project
      valueType: text
      value: dev-project
      options:
        override:
          type: allowed
    # Here we using the output from the EKSResourceTemplate's eks_cluster_name information and use it
for the cluster where we want namespace to be created
    - name: target_cluster_name
      valueType: expression
      value: $(resource."eks-resource-template".output.eks_cluster_name.value)$
      options:
        required: true
        override:
          type: notallowed
  version: v1
```

Note: The input variable target_cluster_name in namespace resource template above is chained with the EKS resource template's output variable eks_cluster_name's value

terraform/eks/main.tf

```
Unset
resource "rafay_cloud_credential" "aws_creds" {
  name        = var.aws_cloud_provider_name
  project     = var.project
```

```
  description  = "description"
  type         = "cluster-provisioning"
  providertype = "AWS"
  awscredtype  = "accesskey"
  accesskey    = var.aws_access_key_id
  secretkey    = var.aws_secret_access_key
}

resource "rafay_eks_cluster" "ekscluster-basic" {
  cluster {
    kind = "Cluster"
    metadata {
      name    = var.cluster_name
      project = var.project
    }
    spec {
      type           = "eks"
      blueprint      = "default"
      cloud_provider = rafay_cloud_credential.aws_creds.name
      cni_provider   = "aws-cni"
      proxy_config   = {}
    }
  }
  cluster_config {
    apiversion = "rafay.io/v1alpha5"
    kind       = "ClusterConfig"
    metadata {
      name    = var.cluster_name
      region  = var.aws_region
      version = var.eks_cluster_version
      tags    = var.tags
    }
    vpc {
      subnets {
        dynamic "private" {
          for_each = var.eks_private_subnets

          content {
            name = private.value
            id   = private.value
          }
        }
        dynamic "public" {
          for_each = var.eks_public_subnets

          content {
            name = public.value
            id   = public.value
          }
        }
```

```
    }
    cluster_endpoints {
      private_access = true
      public_access  = var.eks_cluster_public_access
    }
  }
  node_groups {
    name       = "ng-1"
    ami_family = "AmazonLinux2"
    iam {
      iam_node_group_with_addon_policies {
        image_builder = true
        auto_scaler   = true
      }
    }
    instance_type      = var.eks_cluster_node_instance_type
    desired_capacity   = 1
    min_size           = 1
    max_size           = 2
    max_pods_per_node  = 50
    version            = var.eks_cluster_version
    volume_size        = 80
    volume_type        = "gp3"
    private_networking = true
    subnets            = var.eks_private_subnets
    labels = {
      app       = "infra"
      dedicated = "true"
    }
    tags = var.tags
  }
 }
}
```

eks/outputs.tf

```
Unset
output "eks_cluster_name" {
  value = rafay_eks_cluster.ekscluster-basic.cluster[0].metadata[0].name
}
```

eks/provider.tf

```
Unset
terraform {
  backend "local" {}
  required_providers {
```

```
    rafay = {
      version = ">=1.1.37"
      source  = "RafaySystems/rafay"
    }
  }
  required_version = ">= 1.4.4"
}

provider "aws" {
  region     = var.aws_region
  access_key = var.aws_access_key_id
  secret_key = var.aws_secret_access_key
}
```

eks/variables.tf

```
Unset
variable "aws_region" {
  description = "Configuring AWS as provider"
  type        = string
  default     = "us-west-2"
}

variable "aws_cloud_provider_name" {
  description = "cloud credentials name"
  default     = "aws-cloud-creds-1"
}

variable "aws_access_key_id" {
  description = "aws access key"
  default     = ""
  sensitive   = true
}

variable "aws_secret_access_key" {
  description = "aws secret key"
  default     = ""
  sensitive   = true
}

variable "cluster_name" {
  description = "name of the eks cluster"
  default     = "eks-cluster-1"
}

variable "project" {
  description = "name of the project"
  default     = "dev-project"
```

```
}

variable "eks_cluster_version" {
  description = "eks cluster version"
  default     = "1.26"
}

variable "eks_cluster_node_instance_type" {
  description = "node instance type"
  default     = "t3.large"
}

variable "eks_cluster_public_access" {
  description = "public access"
  default     = true
}

variable "eks_public_subnets" {
  description = "public subnets"
}

variable "eks_private_subnets" {
  description = "private subnets"
}

variable "tags" {
  description = "AWS Tags"
  type        = map(string)
  default = {
    "env"   = "qa"
    "email" = "test@rafay.co"
  }
}
```

namespace/main.tf

```
Unset
resource "random_id" "rnd" {
  keepers = {
    first = "${timestamp()}"
  }
  byte_length = 4
}

locals {
  # Create a unique namspace name
  namespace = "${var.project}-${random_id.rnd.dec}"
}
```

```
resource "rafay_namespace" "namespace" {
  metadata {
    name    = local.namespace
    project = var.project
  }
  spec {
    drift {
      enabled = true
    }
    placement {
      labels {
        key   = "rafay.dev/clusterName"
        value = var.target_cluster_name
      }
    }
    resource_quotas {
      config_maps              = "10"
      cpu_limits               = "4000m"
      memory_limits            = "4096Mi"
      cpu_requests             = "2000m"
      memory_requests          = "2048Mi"
      persistent_volume_claims = "2"
      pods                     = "30"
      replication_controllers  = "5"
      services                 = "10"
      services_load_balancers  = "10"
      services_node_ports      = "10"
      storage_requests         = "1Gi"
    }
  }
}
```

namespace/outputs.tf

```
Unset
output "namepsace" {
  value = local.namespace
}
```

namespace/provider.tf

```
Unset
terraform {
  backend "local" {}
  required_providers {
    rafay = {
```

```
      version = ">=1.1.37"
      source  = "RafaySystems/rafay"
    }
  }
  required_version = ">= 1.4.4"
}
```

namespace/variables.tf

```
Unset
variable "target_cluster_name" {
  description = "name of the eks cluster"
  default     = "eks-cluster-1"
}

variable "project" {
  description = "name of the project where the cluster resides"
  type        = string
  default     = "eaas"
}
```

Loading and launching the environment

1. Run the pipeline through UI or RCTL to sync the templates from GitHub repo to project in your organization
2. You will see all of the above entities: env template, resource templates and the config context in the dev-project
3. You can select version v2 of the EnvironmentTemplate and launch an instance of the environment by providing input information
4. The output of the run will be
    a. VPC creation
    b. EKS cluster provision onto the VPC (EKS cluster can be seen in the clusters list in the project)
    c. Namespace creation on the EKS cluster (output displayed in the activity panel)

Congratulations! , With this, now you have successfully created a template for creating VPC, EKS Cluster and a namespace

Building a template with a schedule using container driver

Container Driver in file “AppResizeDriver.yaml”

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: Driver
metadata:
  name: appresize
  project: dev-project
spec:
  config:
    type: container
    timeoutSeconds: 600
    container:
      image: 'registry.dev.rafay-edge.net/rafay/resize:0.2'
      # Run the resize logic with dry run mode to recommend sizing adjsutments in output and for all
namespaces on the cluster
      arguments:
        - '- /opt/resize.py'
        - '- --dry-run'
        - '- --all-namespaces'
      commands:
        - python3
      workingDirPath: /tmp
      cpuLimitMilli: '500'
      memoryLimitMb: '1024'
status: {}
```

We will now enhance our EnvironmentTemplate and add “Scheduled runs” of “appresize” driver logic onto the cluster for sizing the deployments. “Appresize” looks at utilization metrics of pods running on the cluster and suggests appropriate resource requests/limits to rightsize applications (and reduce infrastructure spend).

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: EnvironmentTemplate
metadata:
  name: app-environment-template
  project: dev-project
spec:
  # Here we are associating an agent which will be used for running code of all involved resource
templates, in this case being VPCResourcetemplate
  agents:
    - name: dev-agent
  contexts:
    # Here we are associating a configcontext that contains information related to AWS credentials. Reason
a configcontext is used here , so that the same can be associated to multiple other env templates
    - name: rafay-config-context
  resources:
  # Here we are associating first resource template VPCResourceTemplate and its version.
  - kind: resourcetemplate
```

```
    name: vpc-resource-template
    resourceOptions:
      version: v1
    type: dynamic
  # Here we are associating second resource template EKSResourceTemplate and its version with the
dependency on VPCResourceTemplate.
  - kind: resourcetemplate
    name: eks-resource-template
    resourceOptions:
      version: v1
    type: dynamic
    dependsOn:
      - name: vpc-resource-template
  # Here we are associating third resource template NamespaceResourceTemplate and its version with the
dependency on EKSResourceTemplate.
  - kind: resourcetemplate
    name: namespace-resource-template
    resourceOptions:
      version: v1
    type: dynamic
    dependsOn:
      - name: eks-resource-template
  variables:
    # Here we are defining input collection variable to collect cluster name from end user of the template.
    # Its type is set to allowed, so that end user at environment launch time can provide region info which is
free form text
    # Its marked required, so that it shows up as mandatory field in the UI
    # selector is user here to wire the input value collected for cluster name to the EKS resource template's
variable. This way, platform team gets to pre-set, restrict values at only one place , which is environment
template level before its shared with end developers/users
    - name: cluster_name
      valueType: text
      options:
        required: true
        override:
          type: allowed
# Here we are setting the schedule to run appresize nightly
  schedules:
    - name: 'daily-schedule '
      type: workflows
      cadence:
        cronExpression: 59 23 * * *
        cronTimezone: America/Los_Angeles
      workflows:
        tasks:
          - name: AppResizeRun
            type: driver
            options: {}
            agents:
              - name: dev-agent
```

```
        onFailure: unspecified
        driver:
          name: appresize
    optOutOptions:
      allowOptOut: false
version: v3
versionState: active
```

Loading and launching the environment

1. Run the pipeline through UI or RCTL to sync the templates from GitHub repo to project in your organization
2. You will see all of the above entities: env template, resource templates and the config context in the dev-project
3. You can select the EnvironmentTemplate and version v3, and launch an instance of the environment by providing input information
4. The output of the run will be
    a. VPC creation
    b. EKS cluster provision onto the VPC (EKS cluster can be seen in the clusters list in the project)
    c. Namespace creation on the EKS cluster (output displayed in the activity panel)
    d. Scheduled run of the appresize will happen at 11:59PM daily

Congratulations!, With this, now you have successfully created self-service template for creating VPC, EKS Cluster, namespace and a scheduled job run for application rightsizing

Building a Template with Pre/Post Hooks using function driver

We will now evolve our Environment template to have an approval hook as a pre-hook for environment provisioning i.e. environment deployment will only proceed on an explicit approval.

In this example, we will create a Jira Approval workflow by creating a function driver and attaching to the environment with onInit hook.

JiraApproval logic as part of the workflow will create a Jira ticket and wait on a certain jira state. After the Jira ticket is approved, Infrastructure resources will be created. If approval is denied, infrastructure resources will not be created

Function Driver in file “JiraApproval.yaml”

```
Unset
apiVersion: eaas.envmgmt.io/v1
```

```
kind: Driver
metadata:
  name: system-jira-v01
  project: catalog-templates
spec:
  inputs:
    - name: inputvars
      data:
        variables:
          - name: debug
            value: "False"
            valueType: TEXT
            options:
              description: "Enables the verbose in the logs"
              override:
                type: allowed
          # jira connection parameters
          - name: jira_fqdn
            value: <jira fqdn>
            valueType: TEXT
            options:
              description: "Jira FQDN"
              override:
                type: allowed
          - name: jira_api_user
            value: <api user>
            valueType: TEXT
            options:
              description: "Jira API User"
              override:
                type: allowed
          - name: jira_api_token
            value: <api token>
            valueType: TEXT
            options:
              description: "Jira API Key"
              override:
                type: allowed
          - name: jira_approved_state
            value: "Approved"
            valueType: TEXT
            options:
              description: "Jira ticket state for approval"
              override:
                type: allowed
          - name: jira_denied_state
            value: "Denied"
            valueType: TEXT
            options:
              description: "Jira ticket state for denied"
```

```
            override:
              type: allowed
        - name: jira_project
          value: "EM"
          valueType: TEXT
          options:
            description: "Jira project name"
            override:
              type: allowed
        # Jira ticket parameters
        - name: short_description
          value: "Jira ticket from Approval"
          valueType: TEXT
          options:
            description: "Jira ticket text"
            override:
              type: allowed
        - name: description
          value: "Jira ticket from Approval description"
          valueType: TEXT
          options:
            description: "Jira ticket description"
            override:
              type: allowed
        - name: assignee
          value: <api user>
          valueType: TEXT
          options:
            description: "Jira ticket state for denied"
            override:
              type: allowed
        - name: priority
          value: "Low"
          valueType: TEXT
          options:
            description: "Jira ticket priority"
            override:
              type: allowed
config:
  type: function
  timeoutSeconds: 300
  pollingConfig:
    repeat: 15s
    until: 1h
  function:
    cpuLimitMilli: "50"
    memoryLimitMi: "128"
    language: python
    languageVersion: "3.6"
    maxConcurrency: 10
```

```
numReplicas: 1
source: |
  from typing import *
  import requests
  from requests.auth import HTTPBasicAuth
  import json
  from logging import Logger
  from python_sdk_rafay_workflow import sdk
  import re
  from requests.adapters import HTTPAdapter

  headers = {
      'Content-Type': 'application/json',
      'Accept': 'application/json'
  }

  class Config:
      ### generic
      debug: str  # true or false

      ### jira connection configuration
      jira_fqdn: str
      jira_api_user: str
      jira_api_token: str
      jira_approved_state: str
      jira_denied_state: str
      jira_project: str

      ### jira ticket configuration
      short_description: str
      description: str
      assignee: str
      priority: str

      ## internal
      assignee_id: str
      jira_id: str

  def validate_inputs(request: Dict[str, Any]) -> None:
      # Check jira_fqdn
      if not request.get('jira_fqdn'):
          raise ValueError("jira_fqdn can not be empty.")
      if not re.match(r'^[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$', request['jira_fqdn']):
          raise ValueError("jira_fqdn must be a valid domain format.")

      # Check jira_api_user
      if not request.get('jira_api_user'):
          raise ValueError("jira_api_user can not be empty.")
      if not re.match(r'^[\w\.-]+@[\w\.-]+\.\w+$', request['jira_api_user']):
          raise ValueError("jira_api_user must be a valid email format.")
```

```
    # Check jira_api_token
    if not request.get('jira_api_token'):
        raise ValueError("jira_api_token can not be empty.")

    # Check jira_project
    if not request.get('jira_project'):
        raise ValueError("jira_project can not be empty.")

    # Check assignee
    if not request.get('assignee'):
        raise ValueError("assignee can not be empty.")
    if not re.match(r'^[\w\.-]+@[\w\.-]+\.\w+$', request['assignee']):
        raise ValueError("assignee must be a valid email format.")

    # Check priority
    priority = request.get('priority')
    if not priority:
        raise ValueError("priority can not be empty.")


def create_issue(logger: Logger, conf: Config):
    url = "https://" + conf.jira_fqdn + "/rest/api/3/issue"
    auth = HTTPBasicAuth(conf.jira_api_user, conf.jira_api_token)

    headers = {
        "Accept": "application/json",
        "Content-Type": "application/json"
    }

    payload = json.dumps({
        "fields": {
            "assignee": {
                "accountId": conf.assignee_id
            },
            "description": {
                "type": "doc",
                "version": 1,
                "content": [
                    {
                        "type": "paragraph",
                        "content": [
                            {
                                "type": "text",
                                "text": conf.short_description,
                            }
                        ],
                    }
                ],
            },
```

```
                "issuetype": {
                    "name": "Task"
                },
                "project": {
                    "key": conf.jira_project
                },
                "priority": {
                    "name": conf.priority
                },
                "summary": conf.description,
            },
        })

        if conf.debug == True:
            logger.debug("Request to the server for Jira ticket creation")
            logger.debug(payload)

        response = requests.request(
            "POST",
            url,
            data=payload,
            headers=headers,
            auth=auth
        )

        if conf.debug == True:
            logger.debug(f"Response from the server for the Jira ticket {conf.jira_id}")
            logger.debug(json.dumps(json.loads(response.text), sort_keys=True, indent=4, separators=(",", ":
"")))

        if response.status_code == 201:
            logger.info("Jira ticket created successfully")
            return response.json()
        else:
            logger.error(f"Failed to create Jira ticket: {response.status_code}, {response.text}")
            return None

    def get_user(logger: Logger, conf: Config, user: str):
        url = "https://" + conf.jira_fqdn + "/rest/api/3/user/search"
        auth = HTTPBasicAuth(conf.jira_api_user, conf.jira_api_token)
        headers = {
            "Accept": "application/json"
        }

        query = {
            'query': user
        }
        response = requests.request(
            "GET",
            url,
```

```
            headers=headers,
            params=query,
            auth=auth
        )

        if conf.debug == True:
            logger.debug(f"Response from the server for the user: {user}")
            logger.debug(json.dumps(json.loads(response.text), sort_keys=True, indent=4, separators=(",", ":
"")))

        return json.loads(response.text)[0]['accountId']

    def get_status(logger: Logger, conf: Config):
        url = "https://" + conf.jira_fqdn + "/rest/api/3/issue/" + conf.jira_id
        status = "Wait"
        auth = HTTPBasicAuth(conf.jira_api_user, conf.jira_api_token)
        headers = {
            "Accept": "application/json"
        }
        response = requests.request(
            "GET",
            url,
            headers=headers,
            auth=auth
        )

        if conf.debug == True:
            logger.debug(f"Response from the server for the ticket status: {conf.jira_id}")
            logger.debug(json.dumps(json.loads(response.text), sort_keys=True, indent=4, separators=(",", ":
"")))

        status = json.loads(response.text)['fields']['status']['name']
        return status

    def handle(logger: Logger, request: Dict[str, Any]) -> Dict[str, Any]:
        try:
            conf = Config()
            conf.debug = request['debug'] if 'debug' in request else ''
            conf.jira_fqdn = request['jira_fqdn'] if 'jira_fqdn' in request else ''
            conf.jira_api_user = request['jira_api_user'] if 'jira_api_user' in request else ''
            conf.jira_api_token = request['jira_api_token'] if 'jira_api_token' in request else ''
            conf.jira_approved_state = request['jira_approved_state'] if 'jira_approved_state' in request else ''
            conf.jira_denied_state = request['jira_denied_state'] if 'jira_denied_state' in request else ''

            conf.short_description = request['short_description'] if 'short_description' in request else ''
            conf.description = request['description'] if 'description' in request else ''
            conf.assignee = request['assignee'] if 'assignee' in request else ''
            conf.jira_project = request['jira_project'] if 'jira_project' in request else ''
            conf.priority = request['priority'] if 'priority' in request else ''
```

```
            # Validate all required fields
            validate_inputs(request)

            logger.info("Checking if ticket exists")
            counter = request['previous'].get('counter', 0) if 'previous' in request else 0
            id = request['previous'].get('ticket_id', '') if 'previous' in request else ''
            key = request['previous'].get('ticket_key', '') if 'previous' in request else ''
            conf.jira_id = id

            if id:
                status = get_status(logger, conf)
            else:
                logger.info("Creating ticket")
                accountId = get_user(logger, conf, conf.assignee)
                conf.assignee_id = accountId

                ticket = create_issue(logger, conf)

                id = ticket['id']
                key = ticket['key']
                conf.jira_id = id
                logger.info(f"Ticket created {key}")
                status = get_status(logger, conf)
        except ConnectionError as e:
            logger.error(f"Failed to connect to the Jira server: {str(e)}")
            raise sdk.TransientException(f"Failed to connect to the Jira server: {str(e)}")
        except Exception as e:
            logger.error(f"FailedException: {str(e)}")
            raise sdk.FailedException(f"FailedException: {str(e)}")

        if status == conf.jira_approved_state:
            logger.info(f"Jira ticket no {key} with id {id} is Approved")
            return {"status": "Resolved", "ticket_id": key, "counter": counter + 1}
        elif status == conf.jira_denied_state:
            logger.info(f"Jira ticket {id} is Denied")
            raise sdk.FailedException(f"Jira ticket no {key} with id {id} is Denied", ticket_id=id,
ticket_key=key)
        else:
            logger.info(f"Waiting for the Jira ticket {id} to be approved")
            raise sdk.ExecuteAgainException(f"Waiting for the Jira ticket no {key} with id {id} to be approved",
ticket_id=id, ticket_key=key, counter=counter + 1)
```

We will now enhance the EnvironmentTemplate with the **JiraApproval** to trigger the approval workflow when an environment is initiated.

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: EnvironmentTemplate
metadata:
  name: app-environment-template
  project: dev-project
spec:
  # Here we are associating an agent which will be used for running code of all involved resource
templates, in this case being VPCResourcetemplate
  agents:
    - name: dev-agent
  contexts:
    # Here we are associating a configcontext that contains information related to AWS credentials. Reason
a configcontext is used here , so that the same can be associated to multiple other env templates
    - name: rafay-config-context
  resources:
  # Here we are associating first resource template VPCResourceTemplate and its version.
  - kind: resourcetemplate
    name: vpc-resource-template
    resourceOptions:
      version: v1
    type: dynamic
  # Here we are associating second resource template EKSResourceTemplate and its version with the
dependency on VPCResourceTemplate.
  - kind: resourcetemplate
    name: eks-resource-template
    resourceOptions:
      version: v1
    type: dynamic
    dependsOn:
      - name: vpc-resource-template
  # Here we are associating third resource template NamespaceResourceTemplate and its version with the
dependency on EKSResourceTemplate.
  - kind: resourcetemplate
    name: namespace-resource-template
    resourceOptions:
      version: v1
    type: dynamic
    dependsOn:
      - name: eks-resource-template
  variables:
    # Here we are defining input collection variable to collect cluster name from end user of the template.
    # Its type is set to allowed, so that end user at environment launch time can provide region info which is
free form text
    # Its marked required, so that it shows up as mandatory field in the UI
    # selector is user here to wire the input value collected for cluster name to the EKS resource template's
variable. This way, platform team gets to pre-set, restrict values at only one place , which is environment
template level before its shared with end developers/users
    - name: cluster_name
      valueType: text
      options:
```

```
      required: true
      override:
        type: allowed
# Here we are setting the schedule to run appresize nightly
schedules:
  - name: "daily-schedule "
    type: workflows
    cadence:
      cronExpression: 59 23 * * *
      cronTimezone: America/Los_Angeles
    workflows:
      tasks:
        - name: AppResizeRun
          type: driver
          options: {}
          agents:
            - name: dev-agent
          onFailure: unspecified
          driver:
            name: appresizer
    optOutOptions:
      allowOptOut: false
# Here we are associating the driver for JiraAprpoval onInit hook
hook:
  onInit:
    - name: approval
      type: driver
      options: {}
      onFailure: unspecified
      driver:
        name: system-jira-v01
  version: v4
versionState: active
```

Loading and launching the environment

1. Run the pipeline through UI or RCTL to sync the templates from github repo to project in your organization
2. You will see all of the above entities, env template, resource template and the config context in the dev-project
3. You can select the EnvironmentTemplate and version v4, and launch an instance of the environment by providing input information
4. The output of the run will be
    a. Oninit, it will wait for approval on the ticket in Jira
    b. On approval of the ticket in Jira, the workflow continues as below
    c. VPC creation

d. EKS cluster provision onto the VPC (EKS cluster can be seen in the clusters list in the project)
e. Namespace creation on the EKS cluster (output displayed in the activity panel)

Congratulations! , With this, now you have successfully created self-service template for creating VPC, EKS Cluster, namespace with approval hook

Complete working example git location
https://github.com/RafaySystems/envmgr/tree/main/development-guide-templates

Design Guidelines and Best Practices

Environment template

- **Agent Management**: Associating an agent or a pool of agents with an environment template simplifies management compared to specifying agents at the resource template level. This can be overridden at environment launch by the end user if needed
- **Dependency and Parallelism**: Leverage resource template dependency ordering and parallelism to align with your application deployment requirements
- **Variable Management**: Prefer using vanilla variables in the environment template over an external config context unless the config settings need to be shared across multiple environment templates
- **Inline Config Context**: Use inline config context at the environment template level only when files need to be included
- **Input Collection**: Define input variables at the environment template level and chain or alias them to corresponding variables in resource templates using selectors. This approach allows you to collect inputs from users once and pass them to multiple resource templates
- **User Control and Restriction**: Gain better control over user permissions, such as restricting values, hiding variables, and defining overrides. Customizing variables at the environment template level (a singleton) is also simpler than doing so across multiple resource templates
- **JSON Variable Type**: Variables of type JSON are highly flexible, supporting other variable types and allowing the inclusion of expressions within JSON

Resource template

- **Isolate Functionality**: It is recommended to isolate functionality for each resource template and link one resource template to another at the environment template level. Avoid creating a monolithic resource template that handles multiple tasks

- **Input Variables**: Expose input variables as mandatory at the resource template level without restrictions. This allows higher-level environment templates to handle customization, such as restricting values or pre-setting defaults
- **Variable Management**: Prefer using vanilla variables in resource templates over an external config context
- **Flexible Value Types**: Use JSON as the value type for its flexibility, as it supports all other types and allows expressions to be included
- **Underlying Code**:
    - Use Terraform/OpenTofu code for Infrastructure as Code (IaC), referenced by the resource template through repository associations
    - If a specific version or engine of Terraform/OpenTofu is required, use a custom packaged driver in place of Rafay's packaged driver
    - If Terraform/OpenTofu is not a suitable option, utilize a packaged driver as a custom provider to execute the required logic as tasks
- **Output Variables**: Expose output variables from the resource template to make module values available as they are populated during execution
    - **Example**:

```
output "public_subnets" {
 value = module.vpc.public_subnets
}
```

Drivers

- For shorter tasks, opt for a **container-based driver**
- For long-running, repeat-polling tasks, utilize **function drivers**
- Use the **HTTP driver** when the task involves simply making HTTP calls to check status

Agents

- It is recommended to define multiple agents for high availability, either at the environment template level or during environment launch/runtime. If agents are defined at both levels, the agent specified at the runtime level takes precedence
- Prefer using an agent running within the cluster if **IAM Roles for Service Accounts (IRSA)** can be leveraged for provisioning resources in the environment

IaC - Terraform code

Variables

- Every variable should have
    - name
    - description
    - default value (if optional)

- Every variable other than optional should have validation

```
variable "eks_devworkspace_cluster_routing_sufix" {
  type = string
  description = "*Required: Supports multi-tenancy will generate url with the suffix provided e.g. for suffix devworkspace.dev.rafay-edge.net and name provided tem1 for every workspace we will get name <workspaceid>.<name>-devworkspace.dev.rafay-edge.net"
  validation {
   condition = (length(var.eks_devworkspace_cluster_routing_sufix)>0)
   error_message = "Value is required and a hosted zone should be created."
 }
}
```

- All variables are declared in locals block at start of your main.tf and use appropriate logic required for naming conventions

```
locals {
 name = "${replace(var.name, "_", "-")}-eks-cluster"
 namespace = "${local.name}-ns"
 aws_public_subnet = "${local.name}-public-subnet"
 tags = merge(var.tags,{
    ManagedBy = "Rafay Eaas"
    Resource = "${local.name}"
 })
}
```

- In cases where there are restrictions for a resource for e.g., RDS Oracle will only accept 8 characters, a random name can be generated and added to the original name in tags

```
locals {
 rds_oracle_name = "${substr(local.name,1,4)}${random.name_randomizer.hex}"
 tags = merge(var.tags,{
   OriginalName = "${local.name}"
 })
}
resource "random" "name_randomizer"{
  keepers = {
   # Generate a new id each time we switch to a new name
   name = local.name
 }
 byte_length = 4
}
```

- Naming of variables should be logical conveying the purpose of the variable

if string variables
<resource_name>_<purpose>
if boolean variable
<resource_name>_<purpose>_enabled
if numeric
<resource_name>_<purpose>_<count|max|min>

- Guidelines for changing variables
    1. **Adding a New Variable:**
        a. Declare the new variable as optional by providing a default value to ensure backward compatibility
        b. If the variable is optional for the resource, a null value can be provided
    2. **Modifying Existing Variables:**
        a. Do not change the name or data type of an existing variable
        b. Instead of modifying a variable, add a new one and retain the old variable to maintain backward compatibility
        c. Manage all variables in locals and map the old variable to the new local variable for seamless transition
    3. **Deleting Variables:**
        a. Avoid deleting existing variables from the variable list
        b. Instead, mark the variable as deprecated in its description, update the README to explain the behavioral changes, and continue to maintain backward compatibility

Guidelines for Naming Outputs

1. **Descriptive Naming:**
   The name of an output should clearly describe the property it represents and be more structured than free-form naming
2. **Recommended Structure:**
   Use the format {name}_{type}_{attribute}, where:
    - {name}: Represents the resource or data source name
        - Example for data: aws_subnet "private" → private
        - Example for resource: aws_vpc_endpoint_policy "test" → test
    - {type}: Represents the resource or data source type without the provider prefix
        - Example for data: aws_subnet "private" → subnet
        - Example for resource: aws_vpc_endpoint_policy "test" → vpc_endpoint_policy
    - {attribute}: Represents the specific attribute returned by the output
3. **Generic Naming for Complex Outputs:**
    - For outputs returning values derived from multiple resources or interpolation functions, {name} and {type} should be as generic as possible
    - Avoid including unnecessary prefixes
4. **Plural Names for Lists:**
    - If the output is a list, use a plural form in the name

5. **Include Descriptions:**
    - Always provide a description for outputs, even if the purpose seems self-evident
6. **Handling Sensitive Outputs:**
    - Avoid marking arguments as sensitive unless you have complete control over their usage across all modules
7. **Formatting Recommendations:**
    - Use hyphens (-) in argument values and in contexts where the value will be human-readable, such as DNS names for RDS instances

Resource

- The resources/data from the provider should be declared in the following format

```
resource "aws_route_table" "<resource_name>_public" {
}
resource "aws_route_table" "<resource_name>_public" {
}
```

- Resource naming should reflect the logical context of the resource. For example, if the resource represents a service for JupyterHub, it can be named "jupyterhub"
- Do not repeat resource type in resource name (neither partially nor completely)
  Use
  resource "aws_route_table" "public" {}`
  Do Not Use
  resource "aws_route_table" "public_route_table" {}
  resource "aws_route_table" "public_aws_route_table" {}
- A resource should be named using its type if no more descriptive or general name is applicable, or if the resource module creates only one resource of that type. For example, in an AWS VPC module with a single aws_nat_gateway resource and multiple aws_route_table resources, the aws_nat_gateway can be named directly, while the aws_route_table resources should have more descriptive names, such as private, public, or database. Always use singular nouns for resource names
- Place the count or for_each argument at the top of the resource or data source block as the first argument, followed by a newline for separation
- Include the tags argument (if supported by the resource) as the last core argument, followed by depends_on and lifecycle if needed. Each of these should be separated by a single empty line for clarity
- When setting conditions for count or for_each, use boolean values instead of expressions like length or similar constructs

Helm Charts driven environment templates

Helm Provider Compatibility

- Ensure that the Helm provider version in Terraform is compatible with both the Helm version and your Kubernetes version

- Verify any breaking changes between different versions of the Helm provider in Terraform, such as those between v1.x and v2.x

Values File Management

- **Value Changes**: If the Helm chart values have been modified in a new version (e.g., renamed parameters, added, or removed settings), update the values block in your Terraform configuration to reflect these changes
- **Dynamic Values**: If your Helm charts depend on dynamic values or Terraform variables, ensure these variables are properly updated and consistent across all environments

Dry-Run Upgrade

- Helm provides a **--dry-run** option, which is useful to simulate the upgrade without applying changes

Chart Versioning

- **Version Pinning**: Make sure to pin the version of Helm charts in your Terraform configuration using the ***version*** attribute. This prevents Terraform from automatically upgrading to a new chart version unless explicitly changed

Helm Dependencies

Subcharts: If your Helm chart includes dependencies (subcharts), verify whether those dependencies are also being upgraded and check for any breaking changes they might introduce. Avoid using subcharts when possible, as they may not be accessible in self-hosted environments.

Drivers Inputs and Outputs

Inputs

Dynamic inputs (expressions) can be configured for all fields within the driver. Users can supply values for these inputs during environment deployment, define them as variables within the config context attached to the driver, or specify them inline. These expressions are evaluated in the execution context during workflow execution.

Expression Format:
The standard format for expressions is:
$(current.input.<variable-name>)$
Here, <variable-name> represents the name of the variable whose value is being referenced.

Example:
If a variable is named api_key, it can be referenced in the configuration as:
$(current.input.api_key)$

Sample Input File:

```
{
  "kind": "Driver",
  "metadata": {
    "name": "driver-test"
  },
  "spec": {
    "config": {
      "type": "container",
      "container": {
        "image": "jira:latest",
        "arguments": [
          "-ticket_id",
          "$(current.input.ticket_id)$"
        ],
        "env_vars": {
          "API_KEY": "$(current.input.api_key)$"
        }
      }
    },
    "inputs": [
      {
        "data": {
          "variables": [
            {
              "name": "api_key",
              "value": "********",
              "valueType": "text"
            },
            {
              "name": "ticket_id",
              "value": "1234",
              "valueType": "text"
            }
          ]
        }
      }
    ]
  }
}
```

Outputs

Driver outputs can be used as inputs for subsequent drivers within a workflow.

- For the **HTTP driver**, the response payload is saved as output.
- For the **Container driver**, the output.json file is uploaded to the designated endpoint, and its content becomes available as output.

These outputs can be referenced using expressions tailored to specific hooks.

Example Expression:
$(resource.test_resource.hook.onInit.create_ticket.output.ticket_id)$

This expression retrieves the ticket_id value generated by the create_ticket hook during the onInit stage of the test_resource resource template. The ticket_id can then be used in subsequent workflow steps.

Detailed Example:
A resource template named **test_resource** includes two hooks:

1. **onInit Hook**:
    - Creates a Jira ticket.
    - Uploads an output.json file containing: {"ticket_id": "1234"}.
2. **onCompletion Hook**:
    - References the data from the onInit hook using the output expression: $(resource.test_resource.hook.onInit.create_ticket.ticket_id)$.

This allows the ticket_id created during the onInit stage to be reused during the onCompletion stage or in other steps of the workflow.

```
{
  "kind": "ResourceTemplate",
  "metadata": {
    "name": "test_resource"
  },
  "spec": {
    "hooks": {
      "onInit": [
        {
          "name": "create_ticket",
          "type": "driver",
          "driver": {
            "data": {
              "config": {
                "type": "container",
                "container": {
                  "image": "jira:custom",
                  "commands": ["create"]
                }
              }
            }
          }
        }
      ],
      "onCompletion": [
        {
          "name": "send_ticket",
          "type": "driver",
```

```
        "driver": {
          "data": {
            "config": {
              "type": "container",
              "container": {
                "image": "jira:custom",
                "arguments": [
                  "--ticket-id",
                  "$(resource.test_resource.hook.onInit.create_ticket.output.ticket_id)$"
                ]
              }
            }
          }
        }
      }
    ]
  }
}
```

Resource Template Lifecycle Hooks

$(resource.resource_name.hook.onInit.hook_name.output)$
$(resource.resource_name.hook.onSuccess.hook_name.output)$
$(resource.resource_name.hook.onFailure.hook_name.output)$
$(resource.resource_name.hook.onCompletion.hook_name.output)$

Terraform Lifecycle Hooks

Deploy Hooks

$(resource.template_name.hook.deploy.init.before.hook_name.output)$
$(resource.template_name.hook.deploy.init.after.hook_name.output)$
$(resource.template_name.hook.deploy.plan.before.hook_name.output)$
$(resource.template_name.hook.deploy.plan.after.hook_name.output)$
$(resource.template_name.hook.deploy.apply.before.hook_name.output)$
$(resource.template_name.hook.deploy.apply.after.hook_name.output)$
$(resource.template_name.hook.deploy.output.before.hook_name.output)$
$(resource.template_name.hook.deploy.output.after.hook_name.output)$

Destroy Hooks

$(resource.template_name.hook.destroy.init.before.hook_name.output)$
$(resource.template_name.hook.destroy.init.after.hook_name.output)$
$(resource.template_name.hook.destroy.plan.before.hook_name.output)$
$(resource.template_name.hook.destroy.plan.after.hook_name.output)$
$(resource.template_name.hook.destroy.destroy.before.hook_name.output)$

$(resource.template_name.hook.destroy.destroy.after.hook_name.output)$

Environment Template Lifecycle Hooks

$(environment.hook.onInit.hook_name.output)$
$(environment.hook.onSuccess.hook_name.output)$
$(environment.hook.onFailure.hook_name.output)$
$(environment.hook.onCompletion.hook_name.output)$

The expression **$(resource.resource_name.task.task_name.output)$** is used to access the output of a specific task within a resource template. Here's a breakdown of its components:

- **resource.resource_name**: Refers to the resource template containing the task.
- **task.task_name**: Specifies the particular task within that resource template.
- **output**: Refers to the data produced by the task.

This expression enables subsequent tasks or components to utilize the output of a previous task, facilitating data flow and dependency management within the workflow.

Repository

Schema

Sample Repository Specification

```
Unset
#YAML
apiVersion: integrations.k8smgmt.io/v3
kind: Repository
metadata:
  name: some-name
  project: defaultproject
spec:
  endpoint: https://github.com/my-git/test
  type: Git
```

Full schema

Repository CRUD using interfaces

APIs

POST Create/Update Repository

This endpoint creates a new repository of GIT or HELM type. It can also be used to update the repository by updating the spec, as shown below

Request

- Method: POST
- URL: https://console.rafay.dev/apis/integrations.k8smgmt.io/v3/projects/PROJECT_NAME/repositories

Request Headers

X-API-KEY: API_KEY

Example (To create git repository)

JSON

```
Unset
curl --location
'https://console.rafay.dev/apis/integrations.k8smgmt.io/v3/projects/PROJECT_NAME/repositories' \
```

```
--header 'X-API-KEY: $API_KEY' \
--header 'Content-Type: application/json' \
--data '{
        "apiVersion": "integrations.k8smgmt.io/v3",
        "kind": "Repository",
        "metadata": {
            "name": "REPOSITORY_NAME",
            "project": "PROJECT_NAME"
        },
        "spec": {
            "type": "Git",
            "endpoint": "https://github.com/eaas.git",  //UPDATE ENDPOINT
             "agents": [
               {
                  "name": "AGENT_NAME"
               }
             ],
            "options": {},
            "credentials": {
                "username": "USER_NAME",
                "password": "TOKEN"
            },
            "sharing": {
                "enabled": false
            }
        }
}'
```

DELETE Delete Repository

This endpoint is used to delete a specific repository

Request

- Method: DELETE
- URL:https://console.rafay.dev/apis/integrations.k8smgmt.io/v3/projects/PROJECT_NAME/repositories/REPOSITORY_NAME

Request Headers

X--API-KEY: API_KEY

Agents

Schema

[Full schema]

Agent CRUD using interfaces

APIs

POST Create/Update Agent

This endpoint creates a new gitops agent of type Docker. This endpoint can also be used for any other updates to the Agent by updating the spec as shown below.

Request

- Method: POST
- URL: https://console.rafay.dev/apis/gitops.k8smgmt.io/v3/projects/PROJECT_NAME/agents

Request Headers

X-API-KEY: API_KEY

JSON

```
Unset
curl --location 'https://console.rafay.dev/apis/gitops.k8smgmt.io/v3/projects/PROJECT_NAME/agents' \
--header 'X-API-KEY:
ra2.242691ca077ca97b68d07270ee3f9b265f726b88.fd0b2db2a645c947974d095fbfd4ea1c4513e1563306
6230299ff1c1175723cb' \
--header 'Content-Type: application/json' \
--data '{
        "apiVersion": "gitops.k8smgmt.io/v3",
        "kind": "Agent",
        "metadata": {
            "name": "AGENT_NAME",
            "project": "PROJECT_NAME"
        },
        "spec": {
            "type": "Docker",
            "active": true
        }
}'
```

Response:
API response contains 2 configuration files (Docker-compose file & agent configuration file) which are required to deploy agent into Docker environment.

Documentation about setting up Docker agent -
https://docs.rafay.dev/integrations/repositories/agents/#docker-agent

Example API Response for gitops agent API.

JSON

Unset


```
{
  "apiVersion": "gitops.k8smgmt.io/v3",
  "kind": "Agent",
  "metadata": {
     "name": "gitops-agent-eaas",
     "project": "eaas",
     "modifiedAt": "2024-10-09T23:00:31.184388Z"
  },
  "spec": {
     "type": "Docker",
     "active": true,
     "version": "r2.10.0"
  },
  "status": {
     "conditionType": "AgentUnhealthy",
     "conditionStatus": 1,
     "lastUpdated": {
        "seconds": 1728514831,
        "nanos": 180141038
     },
     "reason": "Activated Agent",
     "extra": {
        "files": [
           {
              "name": "relayConfigData-pkvll0m.json",
              "data":
"eyJhcGlWZXJzaW9uIjoiaGVsbS5jYXR0bGUuaW8vdjEiLCJraW5kIjoiSGVsbUNoYXJ0Q29uZmlnIiwibWV0
YWRhdGEiOnsiY3JlYXRpb25UaW1lc3RhbXAiOm51bGwsIm5hbWUiOiJjZC1hZ2VudCIsIm5hbWVzcGFjZ
SI6Imt1YmUtc3lzdGVtIn0sInNwZWMiOnsidmFsdWVzQ29udGVudCI6ImVuZ2luZUFnZW50OlxuICBlbmFib
GVkOiBcInRydWVcIlxuICBuYW1lOiBnaXRvcHMtYWdlbnQtZWFhcy0xXG5lbnY6XG4gIHZlcnNpb25OYW1l
OiByMi4xMC4wXG5pbWFnZTpcbiAgcmVwb3NpdG9yeTogcmVnaXN0cnkuc3RhZ2UucmFmYXktZWRnZS
5uZXQvcmFmYXkvY2QtYWdlbnRcbiAgdGFnOiByMi4xMC4wLTNcbmluaXRDb250YWluZXI6XG4gIHJlcG9
zaXRvcnk6IHJlZ2lzdHJ5LnN0YWdlLnJhZmF5LWVkZ2UubmV0L3JhZmF5L2J1c3lib3g6MS4zM1xuaW5zZ
WN1cmU6IGZhbHNlXG5yZWxheUNvbmZpZ0RhdGE6ICd7XCJhZ2VudElEXCI6XCJwa3ZsbDBtXCIsXCJt
YXhEaWFsc1wiOlwiMlwiLFwicmVsYXlzXCI6W3tcInRva2VuXCI6XCJjczNnbTNzbDVjdmtsYW1hM2hrZ1wi
LFwiYWRkclwiOlwiYXBwLnN0YWdlLnJhZmF5LmRldi46NDQzXCIsXCJlbmRwb2ludFwiOlwiKi5jb25uZWN
```

```
0b3IuY2RyZWxheS5zdGFnZS5yYWZheS5kZXY6NDQzXCIsXCJuYW1lXCI6XCJyYWZheS1jb3JlLWNkLXJlbGF5LWFnZW50XCIsXCJ0ZW1wbGF0ZVRva2VuXCI6XCJidXJtMDc2NWVrbWV1bDVyMDB2MFwifV19J1xucmVzb3VyY2VzOlxuICBsaW1pdHM6XG4gICAgY3B1OiA1MDBtXG4gICAgbWVtb3J5OiAxR2lcbndpdGhQcmlvcml0eUNsYXNzOiB0cnVlXG4ifX0="
            },
            {
                "name": "docker-compose-pkvll0m.yaml",
                "data":
"c2VydmljZXM6CiAgY2QtYWdlbnQtZ2l0b3BzLWFnZW50LWVhYXMtMS1wa3ZsbDBtOgogICAgY29udGFpbmVyX25hbWU6IGNkLWFnZW50LWdpdG9wcy1hZ2VudC1lYWFzLTEtcGt2bGwwbQogICAgaW1hZ2U6IHJlZ2lzdHJ5LnN0YWdlLnJhZmF5LWVkZ2UubmV0L3JhZmF5L2NkLWFnZW50LWRvY2tlcjpyMi4xMC4wLTIKICAgIHByaXZpbGVnZWQ6IHRydWUKICAgIHJlc3RhcnQ6IHVubGVzcy1zdG9wcGVkCiAgICB2b2x1bWVzOgogICAgLSAuL3JlbGF5Q29uZmlnRGF0YS1wa3ZsbDBtLmpzb246L3Zhci9saWIvcmFuY2hlci9rM3Mvc2VydmVyL21hbmlmZXN0cy9jZC1hZ2VudC1jb25maWcuanNvbgp2ZXJzaW9uOiAiMy45Igo="
            }
        ]
    }
  }
}
```


DELETE Delete Agent

This endpoint is used to delete a specific GitOps agent.

Request

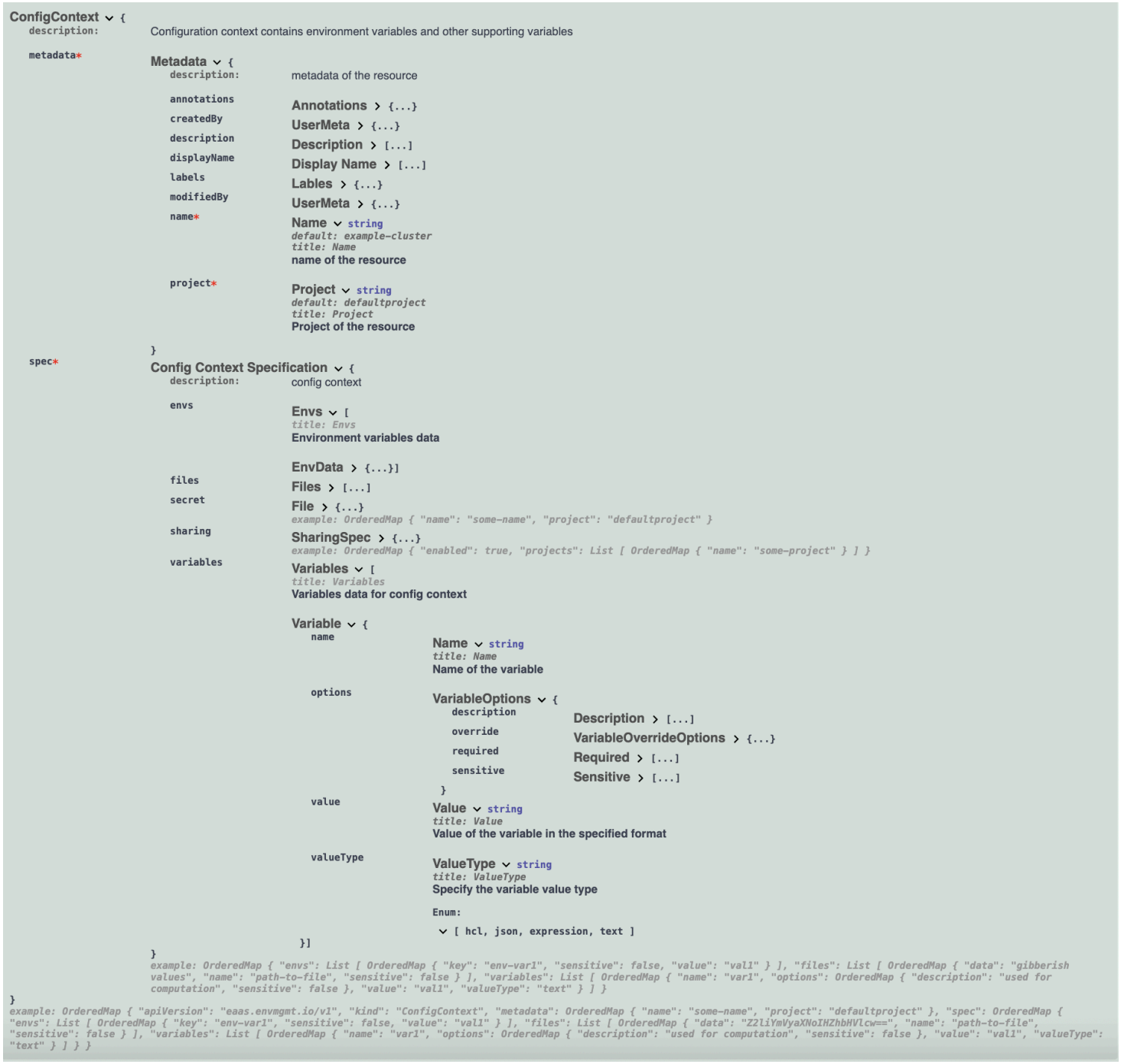
- Method: DELETE
- URL:https://console.rafay.dev/apis/gitops.k8smgmt.io/v3/projects/PROJECT_NAME/repositories/agents/AGENT_NAME

Request Headers

X--API-KEY: API_KEY

Config Context

Schema



Sample ConfigContext Specification

Unset

```
{
  "apiVersion": "eaas.envmgmt.io/v1",
  "kind": "ConfigContext",
```

```
"metadata": {
  "name": "some-name",
  "project": "defaultproject"
},
"spec": {
  "envs": [
    {
      "key": "env-var1",
      "sensitive": false,
      "value": "val1"
    }
  ],
  "files": [
    {
      "data": "Z2liYmVyaXNoIHZhbHVlcw==",
      "name": "path-to-file",
      "sensitive": false
    }
  ],
  "variables": [
    {
      "name": "var1",
      "options": {
        "description": "used for computation",
        "sensitive": false
      },
      "value": "val1",
      "valueType": "text"
    }
  ]
 }
}
```


[Full schema]

ConfigContext CRUD using interfaces

APIs

POST Create/Update ConfigContext

This endpoint creates a new ConfigContext. This endpoint can also be used for any other updates to the ConfigContext by updating the spec as shown below.

Request

- Method: POST
- URL: https://console.rafay.dev//apis/eaas.envmgmt.io/v1/projects/{project}/configcontexts

Request Headers

X-API-KEY: API_KEY

JSON

Unset

```
curl --location 'https://console.rafay.dev//apis/eaas.envmgmt.io/v1/projects/{project}/configcontexts' \
--header 'X-API-KEY:
ra2.242691ca077ca97b68d07270ee3f9b265f726b88.fd0b2db2a645c947974d095fbfd4ea1c4513e1563306
6230299ff1c1175723cb' \
--header 'Content-Type: application/json' \
--data '{
 "apiVersion": "eaas.envmgmt.io/v1",
 "kind": "ConfigContext",
 "metadata": {
  "name": "some-name",
  "project": "defaultproject"
 },
 "spec": {
  "envs": [
   {
    "key": "env-var1",
    "sensitive": false,
    "value": "val1"
   }
  ],
  "files": [
   {
    "data": "Z2liYmVyaXNoIHZhbHVlcw==",
    "name": "path-to-file",
    "sensitive": false
   }
  ],
  "variables": [
   {
    "name": "var1",
    "options": {
     "description": "used for computation",
     "sensitive": false
    },
    "value": "val1",
    "valueType": "text"
   }
  ]
 }
```

```
}'
```

Response:
Example API Response for ConfigContext API.

JSON

Unset


```
{
 "apiVersion": "eaas.envmgmt.io/v1",
 "kind": "ConfigContext",
 "metadata": {
  "name": "some-name",
  "project": "defaultproject"
 },
 "spec": {
  "envs": [
   {
    "key": "env-var1",
    "sensitive": false,
    "value": "val1"
   }
  ],
  "files": [
   {
    "data": "Z2liYmVyaXNoIHZhbHVlcw==",
    "name": "path-to-file",
    "sensitive": false
   }
  ],
  "variables": [
   {
    "name": "var1",
    "options": {
     "description": "used for computation",
     "sensitive": false
    },
    "value": "val1",
    "valueType": "text"
   }
  ]
 }
}
```

DELETE Delete ConfigContext

This endpoint is used to delete a specific ConfigContext

Request

- Method: DELETE
- URL:https://console.rafay.dev//apis/eaas.envmgmt.io/v1/projects/{project}/configcontexts/{name}

Request Headers

X--API-KEY: API_KEY

Drivers

Schema

Metadata › {...}

Driver Specification {
description: driver

config

DriverConfig {
container

ContainerDriverConfig {
arguments — Arguments › [...]
commands — Commands › [...]
cpuLimitMilli — Cpu Limit Milli › [...]
envVars — Environment Variables › {...}
files — Files › {...}
image — Image › [...]
imagePullCredentials — ContainerImagePullCredentials › {...}
kubeConfigOptions — ContainerKubeConfigOptions › {...}
kubeOptions — ContainerKubeOptions › {...}
memoryLimitMb — Memory Limit Mb › [...]
volumeOptions — ContainerDriverVolumeOptions › {...}
volumes — Volumes › [...]
workingDirPath — Working Dir Path › [...]
}

function

FunctionDriverConfig {
buildArgs — Build Args › [...]
buildSecrets — Build Secrets › [...]
cpuLimitMilli — CPU Limit Milli › [...]
functionDependencies — Function Dependencies › [...]
functionProcess — Function Process › [...]
image — Image › [...]
imagePullCredentials — ContainerImagePullCredentials › {...}
inactivityTimeoutSeconds — Inactivity Timeout Seconds › [...]
kubeOptions — ContainerKubeOptions › {...}
language — Language › [...]
languageVersion — Language Version › [...]
maxConcurrency — Max Concurrency › [...]
memoryLimitMb — Memory Limit MB › [...]
name — Name › [...]
numReplicas — Num Replicas › [...]
skipBuild — Skip Build › [...]
source — Source › [...]
systemPackages — System Packages › [...]
targetPlatforms — Target Platforms › [...]
}

http

HTTPDriverConfig {
body — Body › [...]
caCert — CaCert › [...]
endpoint — Endpoint › [...]
headers — KubeConfigOptions › {...}
insecure — Insecure › [...]
method — Method › [...]
}

maxRetryCount — Max Retry Count › [...]
pollingConfig — PollingConfig › {...}
successCondition — Success Condition › [...]
timeoutSeconds — Timeout Seconds › [...]
type — Driver Type string
title: Driver Type
Specify the type of driver

Enum:
[unspecified, container, http, short_circuit]

}

inputs — Inputs › [...]
outputs — Outputs › {...}
secret — File › {...}
example: OrderedMap { "name": "some-name", "project": "defaultproject" }
sharing — SharingSpec › {...}
example: OrderedMap { "enabled": true, "projects": List [OrderedMap { "name": "some-project" }] }

}

example: OrderedMap { "config": OrderedMap { "container": OrderedMap { "arguments": List ["loglevel=3"], "commands": List ["run main.go"], "cpuLimitMilli": "200", "image": "paralusio/paralus", "imagePullCredentials": OrderedMap { "password": "test-creds", "registry": "hub.docker.io", "username": "test-user" }, "memoryLimitMb": "1024", "workingDirPath": "path" }, "maxRetryCount": 3, "timeoutSeconds": 100, "type": "container" } }

Driver Status › {...}

{ "apiVersion": "eaas.envmgmt.io/v1", "kind": "Driver", "metadata": OrderedMap { "name": "terraform-driver", "project": "defaultproject" }, "spec": OrderedMap { p { "container": OrderedMap { "ah": OrderedMap { "password": "test-creds", "registry": "hub.docker.io", "username": "test-user" }, "arguments": List ["loglevel=3"], "run main.go"], "cpuLimitMilli": "200", "image": "paralusio/paralus", "memoryLimitMb": "1024", "workingDirPath": "path" }, "maxRetryCount": 3, "timeoutSeconds": 100, } } }

[Full schema]

Sample Container Driver Specification

```
Unset


apiVersion: eaas.envmgmt.io/v1
kind: Driver
metadata:
  name: terraform-driver
  project: defaultproject
spec:
  config:
    container:
      ah:
        password: test-creds
        registry: hub.docker.io
        username: test-user
      arguments:
        - loglevel=3
      commands:
        - run main.go
      cpuLimitMilli: '200'
      image: paralusio/paralus
      memoryLimitMb: '1024'
      workingDirPath: path
    maxRetryCount: 3
    timeoutSeconds: 100

    type: container
```

Sample Function Driver Specification

```
Unset
apiVersion: eaas.envmgmt.io/v1
kind: Driver
metadata:
  name: simple-func-go
  project: defaultproject
spec:
  config:
    type: function
```

```
timeoutSeconds: 300
pollingConfig:
  repeat: "15s"
  until: "1h"
function:
  cpuLimitMilli: "50"
  memoryLimitMi: "128"
  language: go
  languageVersion: "1.22"
  maxConcurrency: 10
  numReplicas: 1
  source: |
    package function

    import (
      "context"

      sdk "github.com/RafaySystems/function-templates/sdk/go"
    )

    func Handle(ctx context.Context, logger sdk.Logger, req sdk.Request) (sdk.Response, error) {
      logger.Info("received request", "req", req)

      counter := 0.0
      if prev, ok := req["previous"]; ok {
        logger.Info("previous request", "prev", prev)
        counter = prev.(map[string]any)["counter"].(float64)
      }

      resp := make(sdk.Response)
      resp["output"] = "Hello World"
      resp["request"] = req

      if err, ok := req["error"]; ok {
        errString, _ := err.(string)
        switch errString {
        case "execute_again":
          if counter > 1 {
            break
          }
          return nil, sdk.NewErrExecuteAgain(errString, map[string]any{
            "rkey":    "rvalue",
            "counter": counter + 1,
          })
        case "transient":
          return nil, sdk.NewErrTransient(errString)
        default:
          return nil, sdk.NewErrFailed(errString)
        }
      }
```

```
    return sdk.Response(resp), nil
}
```

Driver CRUD using interfaces

APIs

POST Create/Update Driver

This endpoint creates a new Driver. This endpoint can also be used for any other updates to the Driver by updating the spec as shown below.

Request

- Method: POST
- URL: https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/drivers

Request Headers

X-API-KEY: API_KEY

JSON

```
Unset
curl --location 'https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/drivers' \
--header 'X-API-KEY:
ra2.242691ca077ca97b68d07270ee3f9b265f726b88.fd0b2db2a645c947974d095fbfd4ea1c4513e1563306
6230299ff1c1175723cb' \
--header 'Content-Type: application/json' \
--data '
{
 "apiVersion": "eaas.envmgmt.io/v1",
 "kind": "Driver",
 "metadata": {
  "name": "terraform-driver",
  "project": "defaultproject"
 },
 "spec": {
  "config": {
   "container": {
    "ah": {
     "password": "test-creds",
     "registry": "hub.docker.io",
```

```
        "username": "test-user"
       },
       "arguments": [
        "loglevel=3"
       ],
       "commands": [
        "run main.go"
       ],
       "cpuLimitMilli": "200",
       "image": "paralusio/paralus",
       "memoryLimitMb": "1024",
       "workingDirPath": "path"
     },
     "maxRetryCount": 3,
     "timeoutSeconds": 100,
     "type": "container"
   }
  }
}

'
```

Response:
Example API Response for Driver API.

JSON

Unset

```
{
 "apiVersion": "eaas.envmgmt.io/v1",
 "kind": "Driver",
 "metadata": {
   "name": "terraform-driver",
   "project": "defaultproject"
 },
 "spec": {
   "config": {
     "container": {
       "ah": {
         "password": "test-creds",
         "registry": "hub.docker.io",
         "username": "test-user"
       },
       "arguments": [
         "loglevel=3"
```

```
      ],
      "commands": [
        "run main.go"
      ],
      "cpuLimitMilli": "200",
      "image": "paralusio/paralus",
      "memoryLimitMb": "1024",
      "workingDirPath": "path"
    },
    "maxRetryCount": 3,
    "timeoutSeconds": 100,
    "type": "container"
  }
 }
}
```


DELETE Delete Driver

This endpoint is used to delete a specific ConfigContext

Request

- Method: DELETE
- URL:https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/drivers/{name}

Request Headers

X--API-KEY: API_KEY

Static Resources

Schema

Resource ⌄ {
description: Resource represents a cloud agnostic resource

metadata*
Metadata ⌄ {
description: metadata of the resource

annotations — **Annotations** › {...}
createdBy — **UserMeta** › {...}
description — **Description** › [...]
displayName — **Display Name** › [...]
labels — **Lables** › {...}
modifiedBy — **UserMeta** › {...}
name* — **Name** › [...]
project* — **Project** › [...]
}

spec*
Resource Specification ⌄ {
description: details of the resource

secret — **File** ⌄ {
description: file represents an file (or a directory) on the File System by relative path

data — **Data** ⌄ string($byte)
title: Data
data is the base64 encoded contents of the file

mountPath — **Mount Path** › [...]
name* — **Name** ⌄ string
title: Name
Name or relative path of a artifact

options — **FileOptions** › {...}
sensitive — **Sensitive** ⌄ boolean
title: Sensitive
Deprected: use options.sensitive. data is encrypted if sensitive is set to true

}
example: OrderedMap { "name": "some-name", "project": "defaultproject" }

sharing — **SharingSpec** › {...}
example: OrderedMap { "enabled": true, "projects": List [OrderedMap { "name": "some-project" }] }

variables — **Variables** ⌄ [
title: Variables
Variables data for resource

Variable ⌄ {
name — **Name** ⌄ string
title: Name
Name of the variable

options — **VariableOptions** ⌄ {
description — **Description** › [...]
override — **VariableOverrideOptions** › {...}
required — **Required** › [...]
sensitive — **Sensitive** › [...]
}

value — **Value** ⌄ string
title: Value
Value of the variable in the specified format

valueType — **ValueType** ⌄ string
title: ValueType
Specify the variable value type

Enum:
⌄ [hcl, json, expression, text]

}]
}
example: OrderedMap { "variables": List [OrderedMap { "name": "var1", "options": OrderedMap { "description": "used for computation", "sensitive": false }, "value": "val1", "valueType": "text" }] }
}
example: OrderedMap { "apiVersion": "eaas.envmgmt.io/v1", "kind": "Resource", "metadata": OrderedMap { "name": "some-name", "project": "defaultproject" }, "spec": OrderedMap { "variables": List [OrderedMap { "name": "var1", "options": OrderedMap { "description": "used for computation", "sensitive": false }, "value": "val1", "valueType": "text" }] } }

Sample Resource Specification

```
Unset

apiVersion: eaas.envmgmt.io/v1
kind: Resource
metadata:
  name: some-name
  project: defaultproject
spec:
  variables:
    - name: var1
      options:
        description: used for computation
        sensitive: false
      value: val1
      valueType: text
```

Full schema

Resource CRUD using interfaces

APIs

POST Create/Update Resource

This endpoint creates a new Resource. This endpoint can also be used for any other updates to the Resource by updating the spec as shown below.

Request

- Method: POST
- URL: https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/resources

Request Headers

X-API-KEY: API_KEY

JSON

Unset

```
curl --location 'https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/resources' \
--header 'X-API-KEY:
ra2.242691ca077ca97b68d07270ee3f9b265f726b88.fd0b2db2a645c947974d095fbfd4ea1c4513e1563306
6230299ff1c1175723cb' \
--header 'Content-Type: application/json' \
--data '
{
 "apiVersion": "eaas.envmgmt.io/v1",
 "kind": "Resource",
 "metadata": {
  "name": "some-name",
  "project": "defaultproject"
 },
 "spec": {
  "variables": [
   {
    "name": "var1",
    "options": {
     "description": "used for computation",
     "sensitive": false
    },
    "value": "val1",
    "valueType": "text"
   }
  ]
 }
}
'
```

Response:
Example API Response for Resource API.

JSON

Unset

```
{
 "apiVersion": "eaas.envmgmt.io/v1",
 "kind": "Resource",
 "metadata": {
  "name": "some-name",
  "project": "defaultproject"
 },
 "spec": {
  "variables": [
   {
```

```
      "name": "var1",
      "options": {
        "description": "used for computation",
        "sensitive": false
      },
      "value": "val1",
      "valueType": "text"
    }
  ]
 }
}
```


DELETE Delete Resource

This endpoint is used to delete a specific ConfigContext

Request

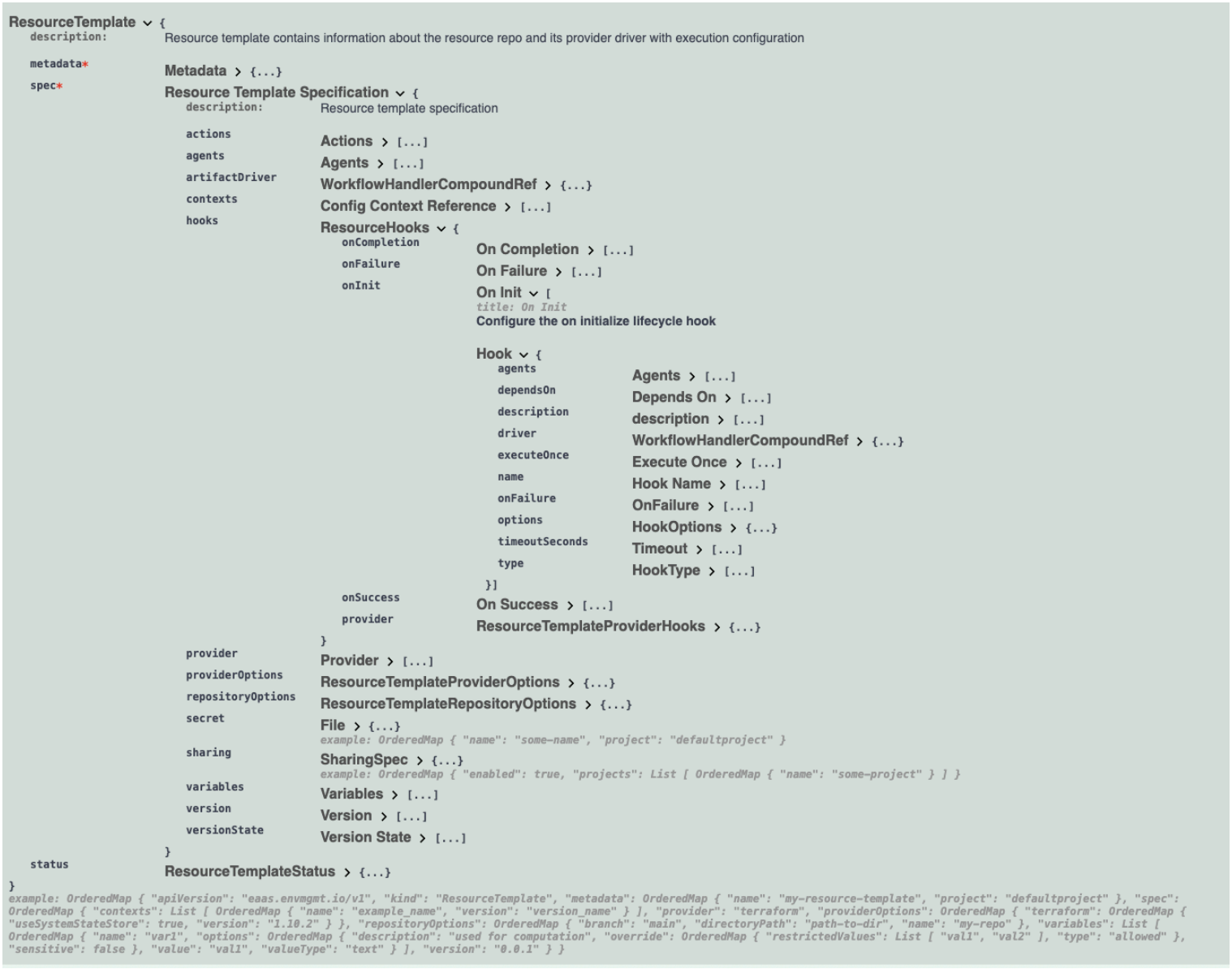
- Method: DELETE
- URL:https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/resources/{name}

Request Headers

X--API-KEY: API_KEY

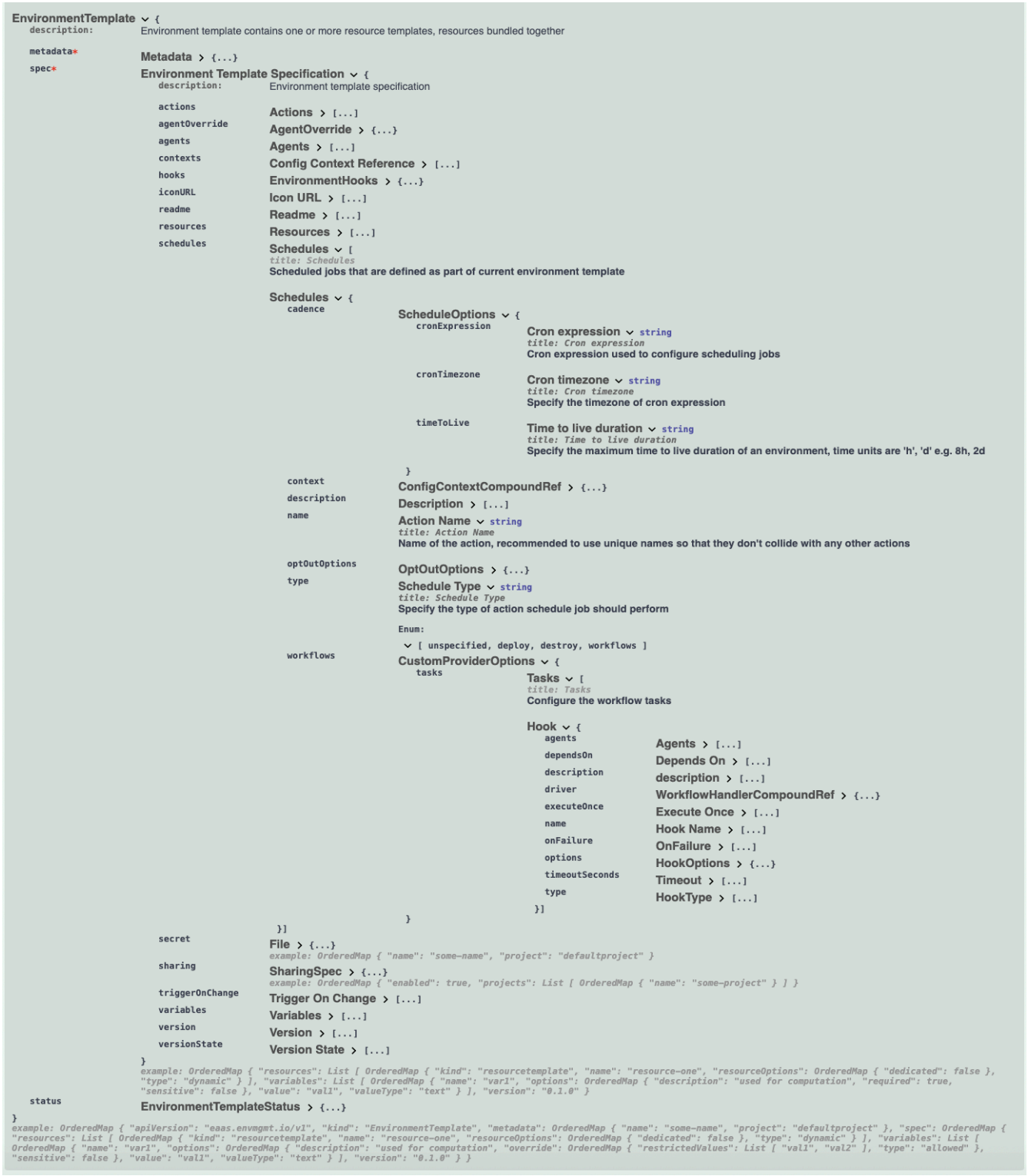
Hooks

Schema showcasing hook at resource template level



Schedules/Cron jobs

Schema showcasing Schedule in environment template



Sample Schedule Specification

Unset

```
{
  "apiVersion": "eaas.envmgmt.io/v1",
  "kind": "EnvironmentTemplate",
  "metadata": {
    "name": "default-envtemplate",
    "description": "Default Environment Template",
    "createdBy": {
      "id": "user-id",
      "username": "user@example.com"
    },
    "modifiedBy": {
      "id": "user-id",
      "username": "user@example.com"
    }
  },
  "spec": {
    "version": "v1",
    "resources": [
      {
        "type": "dynamic",
        "kind": "resourcetemplate",
        "name": "default-resource",
        "resourceOptions": {
          "version": "v1"
        }
      }
    ],
    "schedules": [
      {
        "name": "http-request-job",
        "description": "Executes an HTTP request to a specified endpoint.",
        "type": "workflows",
        "cadence": {
          "cronExpression": "40 07 * * *"
        },
        "workflows": {
          "tasks": [
            {
              "name": "http-request-task",
              "type": "driver",
              "driver": {
                "data": {
                  "config": {
                    "type": "http",
                    "http": {
                      "endpoint": "https://example.com/endpoint",
                      "method": "GET"
                    }
                  }
                }
```

```
            }
          }
        ]
      }
    }
  ],
  "versionState": "draft"
 }
}
```


Resource Template

Schema

ResourceTemplate ⌄ {

description: Resource template contains information about the resource repo and its provider driver with execution configuration

metadata* **Metadata** ⌄ {

description: metadata of the resource

annotations **Annotations** › {...}

createdBy **UserMeta** › {...}

description **Description** › [...]

displayName **Display Name** › [...]

labels **Lables** › {...}

modifiedBy **UserMeta** › {...}

name* **Name** ⌄ string
default: example-cluster
title: Name
name of the resource

project* **Project** ⌄ string
default: defaultproject
title: Project
Project of the resource

}

spec* **Resource Template Specification** ⌄ {

description: Resource template specification

agents **Agents** › [...]

artifactDriver **WorkflowHandlerCompoundRef** › {...}

contexts **Config Context Reference** › [...]

hooks **ResourceHooks** ⌄ {

onCompletion **On Completion** › [...]

onFailure **On Failure** › [...]

onInit **On Init** › [...]

onSuccess **On Success** › [...]

provider **ResourceTemplateProviderHooks** › {...}

}

provider **Provider** ⌄ string
title: Provider
Specify the resource template provider

Enum:
⌄ [terraform, opentofu, hcpterraform, custom, system]

providerOptions **ResourceTemplateProviderOptions** ⌄ {

custom **CustomProviderOptions** › {...}

driver **WorkflowHandlerCompoundRef** › {...}

hcpTerraform **HCPTerraformProviderOptions** › {...}

openTofu **OpenTofuProviderOptions** › {...}

pulumi **PulumiProviderOptions** › {...}

system **SystemProviderOptions** › {...}

terraform **TerraformProviderOptions** › {...}

terragrunt **TerragruntProviderOptions** › {...}

}

repositoryOptions **ResourceTemplateRepositoryOptions** ⌄ {

branch **Branch** ⌄ string
title: Branch
Specify the branch

directoryPath **Directory Path** ⌄ string
title: Directory Path
Specify the directory path

name **Name** ⌄ string
title: Name
Specify the name of the repository

}

secret **File** › {...}
example: OrderedMap { "name": "some-name", "project": "defaultproject" }

sharing **SharingSpec** › {...}
example: OrderedMap { "enabled": true, "projects": List [OrderedMap { "name": "some-project" }] }

variables **Variables** › [...]

version **Version** › [...]

versionState **Version State** › [...]

}

status **ResourceTemplateStatus** › {...}

}

example: OrderedMap { "apiVersion": "eaas.envmgmt.io/v1", "kind": "ResourceTemplate", "metadata": OrderedMap { "name": "my-resource-template", "project": "defaultproject" }, "spec": OrderedMap { "contexts": List [OrderedMap { "name": "example_name", "version": "version_name" }], "provider": "terraform", "providerOptions": OrderedMap { "terraform": OrderedMap { "useSystemStateStore": true, "version": "1.10.2" } }, "repositoryOptions": OrderedMap { "branch": "main", "directoryPath": "path-to-dir", "name": "my-repo" }, "variables": List [OrderedMap { "name": "var1", "options": OrderedMap { "description": "used for computation", "override": OrderedMap { "restrictedValues": List ["val1", "val2"], "type": "allowed" }, "sensitive": false }, "value": "val1", "valueType": "text" }], "version": "0.0.1" } }

Sample ResourceTemplate Specification

```
Unset


apiVersion: eaas.envmgmt.io/v1
kind: ResourceTemplate
metadata:
  name: my-resource-template
  project: defaultproject
spec:
  contexts:
    - name: example_name
      version: version_name
  provider: terraform
  providerOptions:
    terraform:
      useSystemStateStore: true
      version: 1.10.2
  repositoryOptions:
    branch: main
    directoryPath: path-to-dir
    name: my-repo
  variables:
    - name: var1
      options:
        description: used for computation
        override:
          restrictedValues:
            - val1
            - val2
          type: allowed
        sensitive: false
      value: val1
      valueType: text

  version: 0.0.1
```

[Full schema]

ResourceTemplate CRUD using interfaces

APIs

POST Create/Update ResourceTemplate

This endpoint creates a new Resource. This endpoint can also be used for any other updates to the Resource by updating the spec as shown below.

Request

- Method: POST
- URL: https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/resourcetemplates

Request Headers

X-API-KEY: API_KEY

JSON

```
Unset
curl --location 'https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/resourcetemplates' \
--header 'X-API-KEY:
ra2.242691ca077ca97b68d07270ee3f9b265f726b88.fd0b2db2a645c947974d095fbfd4ea1c4513e1563306
6230299ff1c1175723cb' \
--header 'Content-Type: application/json' \
--data '
{
 "apiVersion": "eaas.envmgmt.io/v1",
 "kind": "ResourceTemplate",
 "metadata": {
  "name": "my-resource-template",
  "project": "defaultproject"
 },
 "spec": {
  "contexts": [
   {
    "name": "example_name",
    "version": "version_name"
   }
  ],
  "provider": "terraform",
  "providerOptions": {
   "terraform": {
    "useSystemStateStore": true,
    "version": "1.10.2"
   }
  },
  "repositoryOptions": {
   "branch": "main",
   "directoryPath": "path-to-dir",
   "name": "my-repo"
  },
  "variables": [
```

```
    {
      "name": "var1",
      "options": {
        "description": "used for computation",
        "override": {
          "restrictedValues": [
            "val1",
            "val2"
          ],
          "type": "allowed"
        },
        "sensitive": false
      },
      "value": "val1",
      "valueType": "text"
    }
  ],
  "version": "0.0.1"
 }
}'
```

Response:
Example API Response for ResourceTemplate API.

JSON

Unset

```
{
 "apiVersion": "eaas.envmgmt.io/v1",
 "kind": "ResourceTemplate",
 "metadata": {
   "name": "my-resource-template",
   "project": "defaultproject"
 },
 "spec": {
   "contexts": [
     {
       "name": "example_name",
       "version": "version_name"
     }
   ],
   "provider": "terraform",
   "providerOptions": {
     "terraform": {
       "useSystemStateStore": true,
       "version": "1.10.2"
```

```
    }
  },
  "repositoryOptions": {
    "branch": "main",
    "directoryPath": "path-to-dir",
    "name": "my-repo"
  },
  "variables": [
    {
      "name": "var1",
      "options": {
        "description": "used for computation",
        "override": {
          "restrictedValues": [
            "val1",
            "val2"
          ],
          "type": "allowed"
        },
        "sensitive": false
      },
      "value": "val1",
      "valueType": "text"
    }
  ],
  "version": "0.0.1"
 }
}
```

DELETE Delete ResourceTemplate

This endpoint is used to delete a specific Resourcetemplate

Request

- Method: DELETE
- URL:https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/resourcetemplates/{name}

Request Headers

X--API-KEY: API_KEY

Environment Template

Schema

EnvironmentTemplate ⌄ {

description: Environment template contains one or more resource templates, resources bundled together

metadata*

Metadata ⌄ {

description: metadata of the resource

annotations — **Annotations** › {...}

createdBy — **UserMeta** › {...}

description — **Description** › [...]

displayName — **Display Name** › [...]

labels — **Lables** › {...}

modifiedBy — **UserMeta** › {...}

name* — **Name** ⌄ string
default: example-cluster
title: Name
name of the resource

project* — **Project** ⌄ string
default: defaultproject
title: Project
Project of the resource

}

spec*

Environment Template Specification ⌄ {

description: Environment template specification

agentOverride — **AgentOverride** › {...}

agents — **Agents** › [...]

contexts — **Config Context Reference** › [...]

hooks — **EnvironmentHooks** › {...}

iconURL — **Icon URL** › [...]

readme — **Readme** › [...]

resources — **Resources** ⌄ [
title: Resources
Specify all the environment resources to be included in environment

EnvironmentResourceCompoundRef ⌄ {

dependsOn — **Depends On** ⌄ [
title: Depends On
Specify the environment resource reference that it depends on

ResourceNameAndVersionRef › {...}
example: OrderedMap { "name": "example_name", "version": "version_name" }

kind — **Kind** ⌄ string
title: Kind
Specify the environment resource kind

Enum:
⌄ [resource, resourcetemplate, environment]

name — **Name** ⌄ string
title: Name
Specify the environment resource name

resourceOptions — **EnvironmentResourceOptions** ⌄ {

dedicated — **Dedicated** ⌄ boolean
title: Dedicated
Specify if the resource is dedicated to workloads/apps

version — **Version** ⌄ string
title: Version
Specify the resource version, if blank will use the latest

}

type — **Type** ⌄ string
title: Type
Specify the environment resource type

Enum:
⌄ [dynamic, static]

}]

secret — **File** › {...}
example: OrderedMap { "name": "some-name", "project": "defaultproject" }

sharing — **SharingSpec** › {...}
example: OrderedMap { "enabled": true, "projects": List [OrderedMap { "name": "some-project" }] }

variables — **Variables** ⌄ [
title: Variables
Environment variables, file data and other variables

Variable ⌄ {

name — **Name** › [...]

options — **VariableOptions** › {...}

value — **Value** › [...]

valueType — **ValueType** › [...]

}]

version — **Version** › [...]

versionState — **Version State** › [...]

}
example: OrderedMap { "resources": List [OrderedMap { "kind": "resourcetemplate", "name": "resource-one", "resourceOptions": OrderedMap { "dedicated": false }, "type": "dynamic" }], "variables": List [OrderedMap { "name": "var1", "options": OrderedMap { "description": "used for computation", "required": true, "sensitive": false }, "value": "val1", "valueType": "text" }], "version": "0.1.0" }

status — **EnvironmentTemplateStatus** › {...}

}
example: OrderedMap { "apiVersion": "eaas.envmgmt.io/v1", "kind": "EnvironmentTemplate", "metadata": OrderedMap { "name": "some-name", "project": "defaultproject" }, "spec": OrderedMap { "resources": List [OrderedMap { "kind": "resourcetemplate", "name": "resource-one", "resourceOptions": OrderedMap { "dedicated": false }, "type": "dynamic" }], "variables": List [OrderedMap { "name": "var1", "options": OrderedMap { "description": "used for computation", "override": OrderedMap { "restrictedValues": List ["val1", "val2"], "type": "allowed" }, "sensitive": false }, "value": "val1", "valueType": "text" }], "version": "0.1.0" } }

Sample EnvironmentTemplate Specification

```
Unset

apiVersion: eaas.envmgmt.io/v1
kind: EnvironmentTemplate
metadata:
  name: some-name
  project: defaultproject
spec:
  resources:
    - kind: resourcetemplate
      name: resource-one
      resourceOptions:
        dedicated: false
      type: dynamic
  variables:
    - name: var1
      options:
        description: used for computation
        override:
          restrictedValues:
            - val1
            - val2
          type: allowed
        sensitive: false
      value: val1
      valueType: text
  version: 0.1.0
```

Full schema

EnvironmentTemplate CRUD using interfaces

APIs

POST Create/Update EnvironmentTemplate

This endpoint creates a new EnvironmentTemplate. This endpoint can also be used for any other updates to the EnvironmentTemplate by updating the spec as shown below.

Request

- Method: POST
- URL: https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/environmenttemplates

Request Headers

X-API-KEY: API_KEY

JSON

Unset

```
curl --location 'https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/environmenttemplates' \
--header 'X-API-KEY:
ra2.242691ca077ca97b68d07270ee3f9b265f726b88.fd0b2db2a645c947974d095fbfd4ea1c4513e1563306
6230299ff1c1175723cb' \
--header 'Content-Type: application/json' \
--data '
{
 "apiVersion": "eaas.envmgmt.io/v1",
 "kind": "EnvironmentTemplate",
 "metadata": {
  "name": "some-name",
  "project": "defaultproject"
 },
 "spec": {
  "resources": [
   {
    "kind": "resourcetemplate",
    "name": "resource-one",
    "resourceOptions": {
     "dedicated": false
    },
    "type": "dynamic"
   }
  ],
  "variables": [
   {
    "name": "var1",
    "options": {
     "description": "used for computation",
     "override": {
      "restrictedValues": [
       "val1",
       "val2"
      ],
      "type": "allowed"
     },
     "sensitive": false
    },
    "value": "val1",
```

```
      "valueType": "text"
    }
  ],
  "version": "0.1.0"
 }
}'
```

Response:
Example API Response for EnvironmentTemplate API.

JSON

Unset

```
{
 "apiVersion": "eaas.envmgmt.io/v1",
 "kind": "EnvironmentTemplate",
 "metadata": {
  "name": "some-name",
  "project": "defaultproject"
 },
 "spec": {
  "resources": [
   {
    "kind": "resourcetemplate",
    "name": "resource-one",
    "resourceOptions": {
     "dedicated": false
    },
    "type": "dynamic"
   }
  ],
  "variables": [
   {
    "name": "var1",
    "options": {
     "description": "used for computation",
     "override": {
      "restrictedValues": [
       "val1",
       "val2"
      ],
      "type": "allowed"
     },
     "sensitive": false
    },
    "value": "val1",
```

```
      "valueType": "text"
    }
  ],
  "version": "0.1.0"
 }
}
```

DELETE Delete EnvironmentTemplate

This endpoint is used to delete a specific EnvironmentTemplate

Request

- Method: DELETE
- URL:https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/environmenttemplates/{name}

Request Headers

X--API-KEY: API_KEY

Environment

Environment is an instance of an Environment template, analogy being, instance of a class.

Schema

Environment ⌄ {

`description:` Environment contains environment template and other other supporting variables required to create an environment!

`apiVersion`* **API Version** › `[...]`

`kind`* **Kind** ⌄ `string`
default: ConfigContext
title: Kind
Kind of the environment resource

`metadata`* **Metadata** › `{...}`

`spec`* **Environment Specification** ⌄ {

`description:` specification of the environment

`agents` **Agents** › `[...]`

`envVars` **Environment Variables** › `[...]`

`files` **Files** › `[...]`

`schedule_optouts` **Schedule Opt Outs** › `[...]`

`secret` **File** › `{...}`
example: OrderedMap { "name": "some-name", "project": "defaultproject" }

`sharing` **SharingSpec** › `{...}`
example: OrderedMap { "enabled": true, "projects": List [OrderedMap { "name": "some-project" }] }

`template`* **EnvironmentTemplateCompoundRef** ⌄ {

`description:` Reference to resource

`name`* **Name** ⌄ `string`
title: Name
name of the resource

`spec` **Environment Template Specification** ⌄ {

`description:` Environment template specification

`actions` **Actions** › `[...]`

`agentOverride` **AgentOverride** › `{...}`

`agents` **Agents** › `[...]`

`contexts` **Config Context Reference** › `[...]`

`hooks` **EnvironmentHooks** › `{...}`

`iconURL` **Icon URL** › `[...]`

`readme` **Readme** › `[...]`

`resources` **Resources** › `[...]`

`schedules` **Schedules** › `[...]`

`secret` **File** › `{...}`
example: OrderedMap { "name": "some-name", "project": "defaultproject" }

`sharing` **SharingSpec** › `{...}`
example: OrderedMap { "enabled": true, "projects": List [OrderedMap { "name": "some-project" }] }

`triggerOnChange` **Trigger On Change** › `[...]`

`variables` **Variables** › `[...]`

`version` **Version** › `[...]`

`versionState` **Version State** › `[...]`

}
example: OrderedMap { "resources": List [OrderedMap { "kind": "resourcetemplate", "name": "resource-one", "resourceOptions": OrderedMap { "dedicated": false }, "type": "dynamic" }], "variables": List [OrderedMap { "name": "var1", "options": OrderedMap { "description": "used for computation", "required": true, "sensitive": false }, "value": "val1", "valueType": "text" }], "version": "0.1.0" }

`version`* **Version** ⌄ `string`
title: Version
version of the resource

}
example: OrderedMap { "name": "example_name", "version": "version_name" }

`variables` **Variables** › `[...]`

}

`status` **Environment Status** › `{...}`

}

example: OrderedMap { "apiVersion": "eaas.envmgmt.io/v1", "kind": "Environment", "metadata": OrderedMap { "name": "some-name", "project": "defaultproject" }, "spec": OrderedMap { "template": OrderedMap { "name": "some-name", "version": "1.2.3" }, "variables": List [OrderedMap { "name": "var1", "options": OrderedMap { "description": "used for computation", "required": true, "sensitive": false }, "value": "val1", "valueType": "text" }] } }

Sample Environment Specification

```
Unset


apiVersion: eaas.envmgmt.io/v1
kind: Environment
metadata:
  name: some-name
  project: defaultproject
spec:
  template:
    name: some-name
    version: 1.2.3
  variables:
    - name: var1
      options:
        description: used for computation
        required: true
        sensitive: false
      value: val1

      valueType: text
```

Full schema

Environment CRUD using interfaces

APIs

POST Create/Update Environment

This endpoint creates a new Environment. This endpoint can also be used for any other updates to the Environment by updating the spec as shown below.

Request

- Method: POST
- URL: https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/environments

Request Headers

X-API-KEY: API_KEY

JSON

```
Unset
curl --location 'https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/environments' \
--header 'X-API-KEY:
ra2.242691ca077ca97b68d07270ee3f9b265f726b88.fd0b2db2a645c947974d095fbfd4ea1c4513e1563306
6230299ff1c1175723cb' \
--header 'Content-Type: application/json' \
--data '
{
  "apiVersion": "eaas.envmgmt.io/v1",
  "kind": "Environment",
  "metadata": {
    "name": "some-name",
    "project": "defaultproject"
  },
  "spec": {
    "template": {
      "name": "some-name",
      "version": "1.2.3"
    },
    "variables": [
      {
        "name": "var1",
        "options": {
          "description": "used for computation",
          "required": true,
          "sensitive": false
        },
        "value": "val1",
        "valueType": "text"
      }
    ]
  }
}
'
```

Response:
Example API Response for Environment API.

JSON

```
Unset
{
  "apiVersion": "eaas.envmgmt.io/v1",
  "kind": "Environment",
  "metadata": {
    "name": "some-name",
    "project": "defaultproject"
  },
```

```
 "spec": {
  "template": {
   "name": "some-name",
   "version": "1.2.3"
  },
  "variables": [
   {
    "name": "var1",
    "options": {
     "description": "used for computation",
     "required": true,
     "sensitive": false
    },
    "value": "val1",
    "valueType": "text"
   }
  ]
 }
}
```

DELETE Delete Environment

This endpoint is used to delete a specific Environment

Request

- Method: DELETE
- URL:https://console.rafay.dev/apis/eaas.envmgmt.io/v1/projects/{project}/environments/{name}

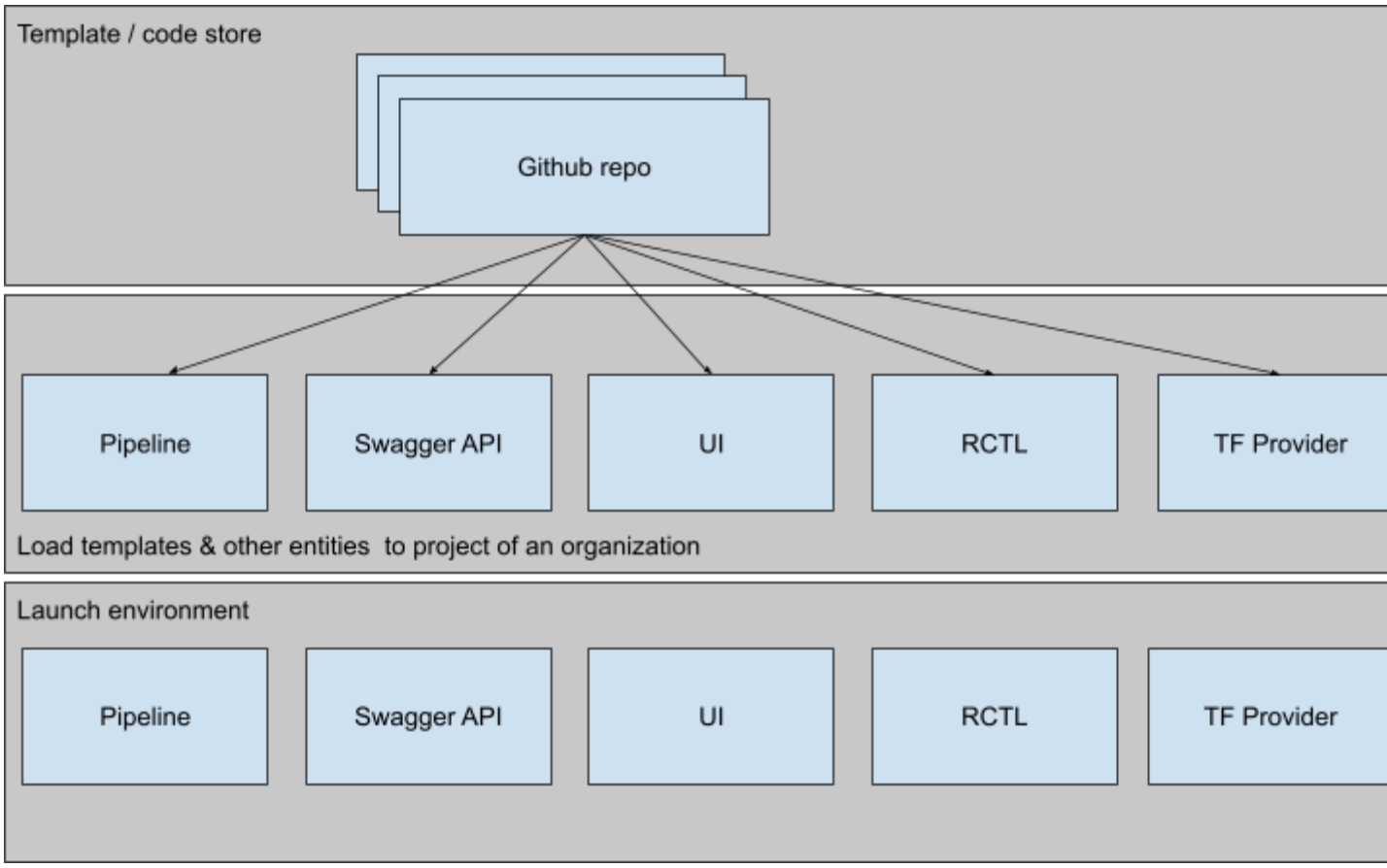
Request Headers

X--API-KEY: API_KEY

Loading and Launching Templates

Rafay Environment Manager provides the following interfaces for loading the environment templates into Rafay Platform and launching them.

- Pipeline
- Swagger API
- UI
- RCTL(Rafay CLI)
- Terraform Provider



Best Practices

1. **Centralized Storage**:
   Store all your templates and related configurations in a private GitHub repository to establish a single source of truth
2. **Automated Loading**:
   Use a unified pipeline configuration to automate the process of loading changes from Git to your system, ensuring seamless updates to your project
3. **Leveraging Swagger APIs**:
   If you are building a platform or marketplace for custom templates outside of Rafay and need to load them into your Rafay organization's project, Swagger APIs provide an efficient and scalable solution