Apcera Training

DEV102 - Extending the Apcera Platform

Lab Book

Apcera Training

Labs Overview

This document lists the lab exercises that comprise Apcera Training.

- Lab 0: Set Up Your Lab Environment
- Lab 1: Using the Apcera API Explorer
    - A. Explorer the API Explorer web tool
    - B. Configure the API Explorer with your API token
- Lab 2: Create NATS client and server apps using API Explorer
    - A. Create a NATS server job from a Docker image
    - B. Create a NATS client job from a Docker image
    - C. Start the NATS client app using API Explorer
- Lab 3: Extending a built-in staging pipeline
    - A. Create a Jeykll stager using the Ruby stager library
    - B. Deploy the newly created stager
    - C. Deploy a Jeykll project with new staging pipeline
    - D. Update the stager job using API
- Lab 4: Creating an Apache static web site stager
    - A. Create and deploy a custom stager
    - B. Deploy and configure a web site with the new staging pipeline
- Lab 5: Creating a simple service gateway
    - A. Download and examine the service gateway code
    - B. Create and deploy the service gateway app
    - C. Create a weather service and bind it to a capsule

Lab Setup

First, you are going to perform a few tasks to set up your environment for the labs. These tasks include downloading the sample apps, installing the APC tool, targeting your cluster, exploring the Apcera Web Console, accessing your cluster documentation, and familiarizing yourself with additional learning resources.

APC Set Up for Mac OS X, Linux, and Windows Clients

Complete the following steps.

Task 1: Clone (copy) the Apcera training repository using command line

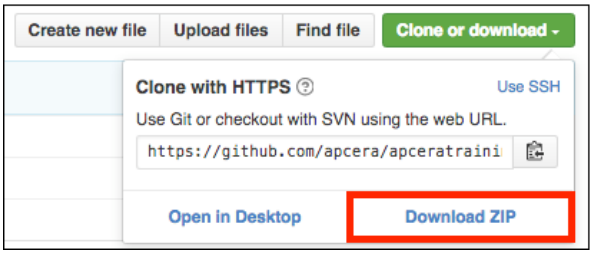
If you are familiar with the git commands, follow the instructions here. Otherwise, please **skip** to Task 2.

Step	Instruction
A.1.1	Launch a command line session and issue the following command: `git clone https://github.com/apcera/apceratraining.git` The repository will be copied to whichever directory the git command is ran from, for example: **`/Users/<username>/apceratraining`**
A.1.2	Go to Task 3

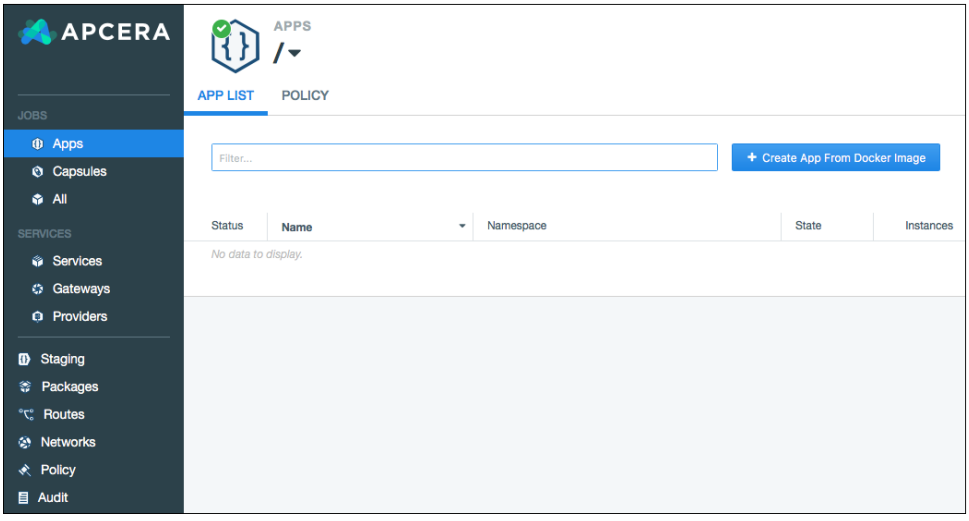
Task 2: Clone (copy) the Apcera training repository using a browser

Note: If you have already completed Task 1, please skip to Task 3.

Step	Instruction
A.2.1	Open a web browser, navigate to the following URL: https://github.com/apcera/apceratraining.git

A.2.2	Click **Clone or download**, and then click the **Download ZIP** button and save the *apceratraining-master.zip* archive to your computer. 
A.2.3	Extract the ZIP file to a directory such as: **/Users/<username>/apceratraining**.

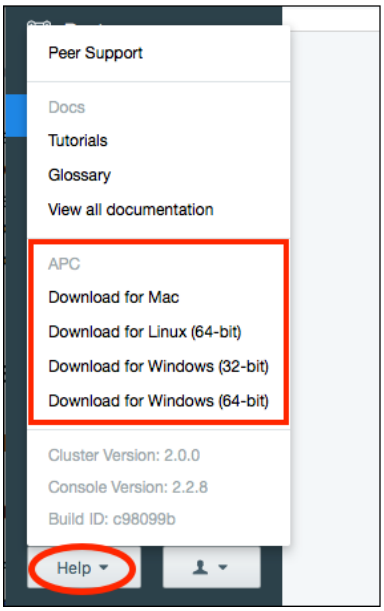
Task 3: Log in to the Web Console using your Google credentials

Step	Instruction
A.3.1	The Web Console lets you manage and monitor the apps deployed to your cluster. Use your browser to access the console at the following URL: https://console.kiso.io.
A.3.2	Log in using your Google credentials. Once logged in, take a minute to explore the console. You may want to bookmark the console URL for convenience. 

	NOTE: If you cannot log in to the console, clear your browser's cache and try again.

Task 4: Install APC command line tool

APC is the command line interface (CLI) tool that you use to interact with your Apcera cluster.

Step	Instruction
A.4.1	In the Web Console, click the **Help** button to expand its menu, and then select the applicable APC download for your OS. 
A.4.2	**Mac OS X users ONLY:** Double-click the PKG file and follow the installer instructions.
A.4.3	**Linux users ONLY:** Unzip the downloaded file to find the apc.exe. NOTE: If necessary you can set optional environment variables for APC. Refer to the following documentation: http://docs.apcera.com/quickstart/installing-apc/.
A.4.4	**Windows users ONLY:** Double click the apc.exe to run your APC command line tool. NOTE: For your convenience, add apc.exe to your PATH environment variable.

Task 5: Login to the lab cluster

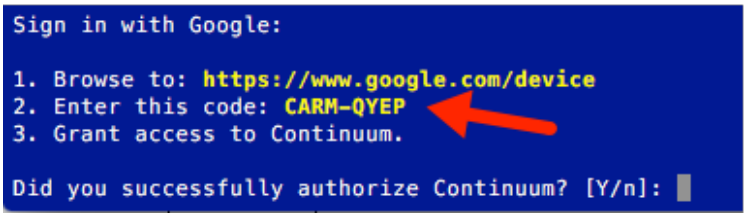
Important: Your cluster is configured by policy to authenticate you using Google Device auth. The instructor will add your **Google Gmail address** to the cluster authentication policy. Please provide the instructor with your Google Gmail address so it can be added to the policy. If you do not have a Gmail address, please create one now and give it to the instructor.

Once you have given the instructor your Gmail address, use a browser and log in to your Gmail account.

Step	Instruction
A.5.1	Open a command window, execute the following `apc` command to verify that the APC is correctly installed. `apc version` If APC is installed, this command will return the version of APC that you are running. If no version is returned, reinstall APC.

A.5.2	For this training a cluster has been set up for you. The training cluster name is ***kiso.io***. To access your cluster using APC, you target it. Once targeted, that cluster is the one your APC session interfaces with until you change the target.
	
	Execute the following command to target your cluster:
	
	`apc target kiso.io`
	
	Success message:
	`Targeted [https://kiso.io]`
	`Logged in as: "<username>"`
	
	Notice your user name.

A.5.3	Execute the following command to log in: `apc login`
A.5.4	Browse to https://www.google.com/device

A.5.5	Select your Google account, and then enter the unique 8-letter code provided by APC. 
A.5.6	Click **Allow** to grant access to the cluster.
A.5.7	Return to the command window, and type `y` (yes) at the prompt, "Did you successfully authorized Continuum?" and press enter. Success message: ***`Login successful for <username>`***
A.5.8	To ensure that you are logged in as your Google user, execute the following command: `apc login show` The "Current login" should be your Google Gmail user name. If it is not, log out of the cluster and redo the above steps. Do not proceed until you are logged in as your Google user.

A.5.9	Execute the following command to set the local namespace to be your home namespace as defined by the authentication policy:
	
	`apc namespace --default`
	
	Expected result: You should see the following messages:
	`Setting namespace to default '/sandbox/<username>'... done`
	`Success!`
	
	NOTE: If your default namespace is different than *'/sandbox/<username>'*, please notify the instructor and get his or her help before proceeding.

Exercise Review

The Apcera documentation is publicly available at http://docs.apcera.com/. In addition, each cluster hosts a version of the Apcera documentation. You can access your cluster documentation at http://docs.kiso.io/, or by by clicking the question mark icon (?) at the upper right of the console and selecting *View all documentation*.

With these preliminary tasks complete, you are now ready to proceed to Lab 1, where you will learn how to use Apcera to deploy a diverse set of workloads, including apps, Docker images, and OS capsules.

Apcera Training

Lab 1: Using the Apcera API Explorer

Lab Introduction

In this lab you will make Apcera REST API calls using a web application called *Apcera API Explorer*, a web application that takes a Swagger specification as input and generates the UI to easily make Apcera API calls. You configure the API Explorer with the API access token obtained by APC.

Lab Exercise

This lab comprises the following exercises:

A. Explorer the API Explorer web tool
B. Configuring the API Explorer with your API token
C. Use the API Explorer to make a basic API call

Lab Prerequisites

This lab assumes that you have set up your lab environment. If you have not done so, please follow the lab setup guide now.

Related Documentation

- Apcera REST API Model Objects: http://docs.apcera.com/api/api-models/
- Apcera REST API Recipes: http://docs.apcera.com/api/api-tasks/

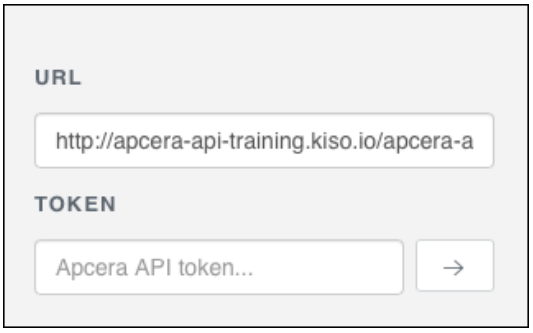
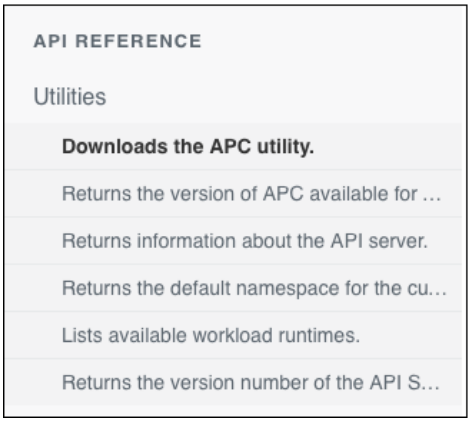
Lab Instruction

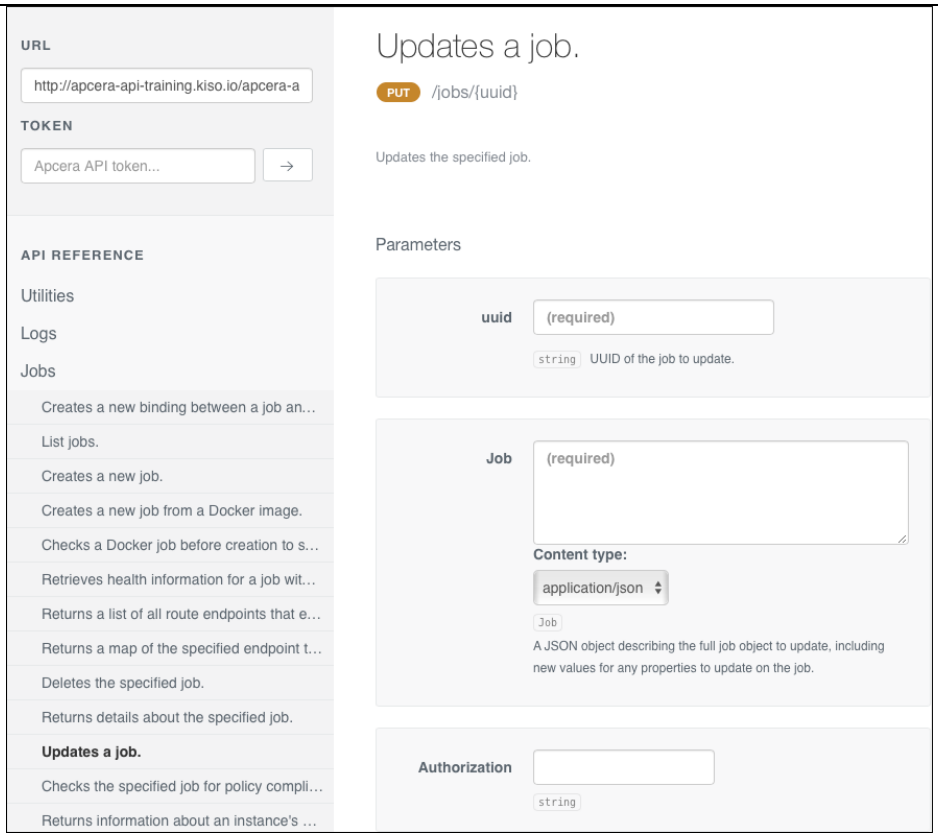
For each exercise, complete the steps.

Exercise A: Explorer the API Explorer web tool

In this exercise you are going to explore the Apcera REST API using API Explorer, a web application that takes a Swagger specification as an input and generates the UI to easily make Apcera API calls. You configure the API Explorer with the API access token obtained by APC.

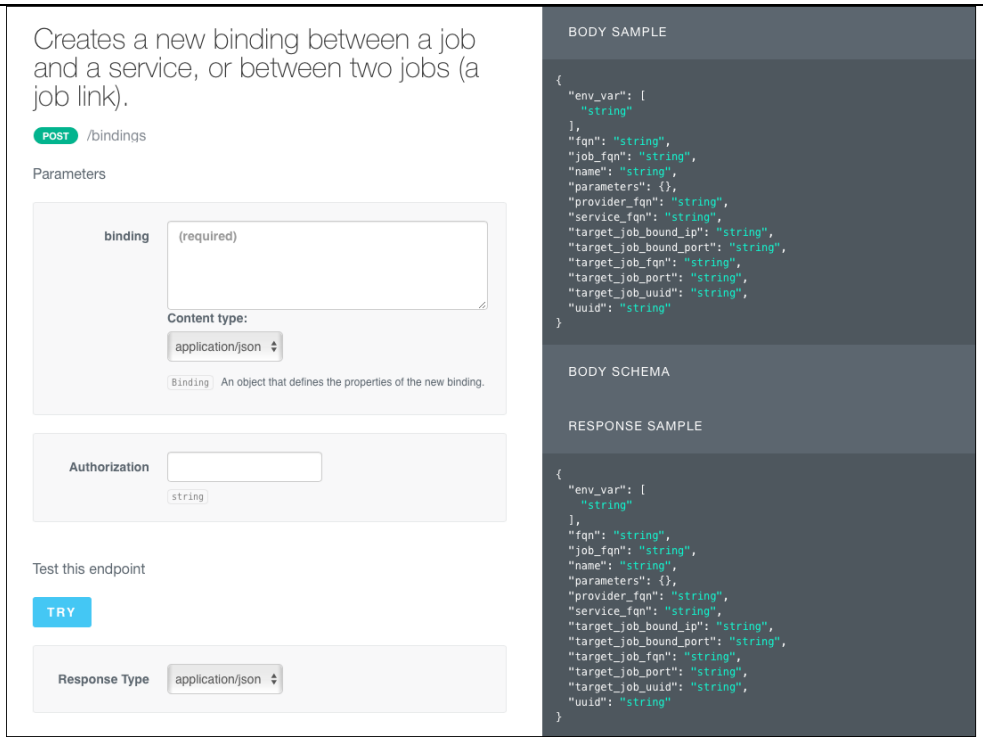
Task 1: Launch the API Explorer

Step	Instruction
A.1.1	Open a web browser, and navigate to: http://apcera-api-training.kiso.io/ The **URL** field displays the location of the Swagger specification being rendered.  The **TOKEN** field is where you provide your authentication token so that you can make API calls.
A.1.2	The **API REFERENCE** section lists all the API endpoints categorized by type. 
A.1.3	Select any endpoint to open its request form.



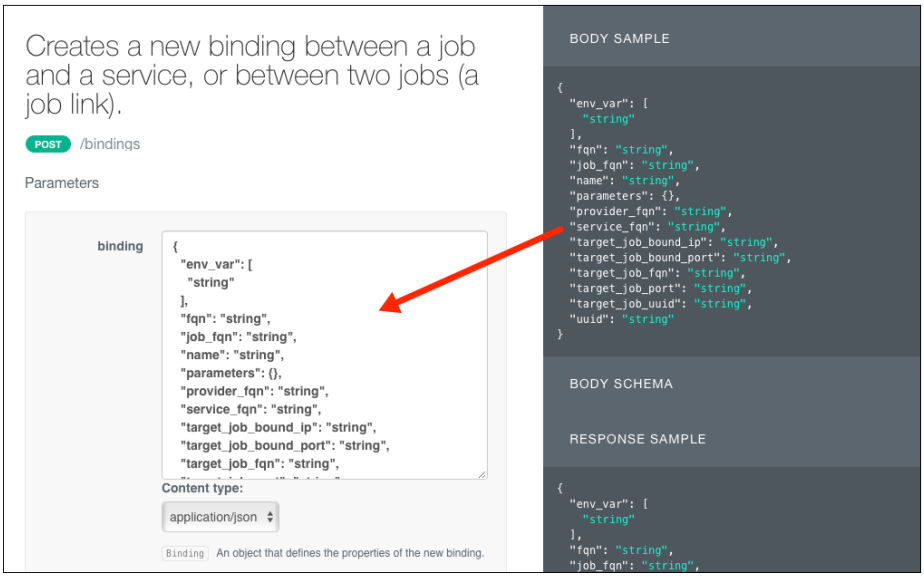
Each API request form lists any parameters, including **path**, **query**, and **body** parameters.

A.1.4	To the right of each API request form are collapsible sections that list a **Response Sample** for the method, and a **Response Schema** that describes each response field.



A.1.5

For endpoints that take a JSON body in the request, **Body Schema** and **Body Response** sections list the request body structure. Click on a **Body Schema** field to copy it to the API request form's body field.



Exercise B: Configuring the API Explorer with your API token

In this exercise you are going to explore the Apcera REST API using API Explorer web tool.

Recall that each request to a API method requires an HTTP "**Authorization**" header that contains your token There are several ways to do this, the specific approach you take depends on the application needs. For purposes of demonstration, you will simply copy the API token from the **$HOME/.apc** file that was obtained when you logged into the training cluster using the `apc login` command.

Task 1: Copy your APC token from your .apc file

In this task, you'll copy your access token from your local `.apc` file.

Step	Instruction

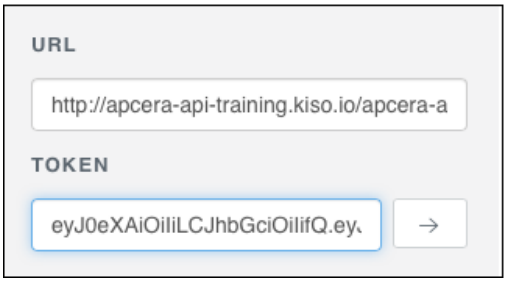
B.1.1	Make sure that you have logged in to kiso.io cluster.
	
	`apc target kiso.io`
	
	Success message:
	`Targeted [https://kiso.io]`
	`Logged in as: "<username>"`
	
	Notice your user name.

B.1.2	Execute the following command to log in: `apc login`
B.1.3	If prompted, follow the instruction to sign in with Google.

B.1.4	Locate and open **`.apc`** in a text editor.
	
	***** Windows users *****
	`edit %USERPROFILE%/.apc`
	
	***** Mac and Linux users *****
	`open ~/.apc`

B.1.5	Locate the **tokens** field for the kiso.io cluster and copy its value **not including** the "`Bearer`" preamble. For example, in the following example you would copy the string starting with **`eyJ0e…`** up to the closing double-quote.

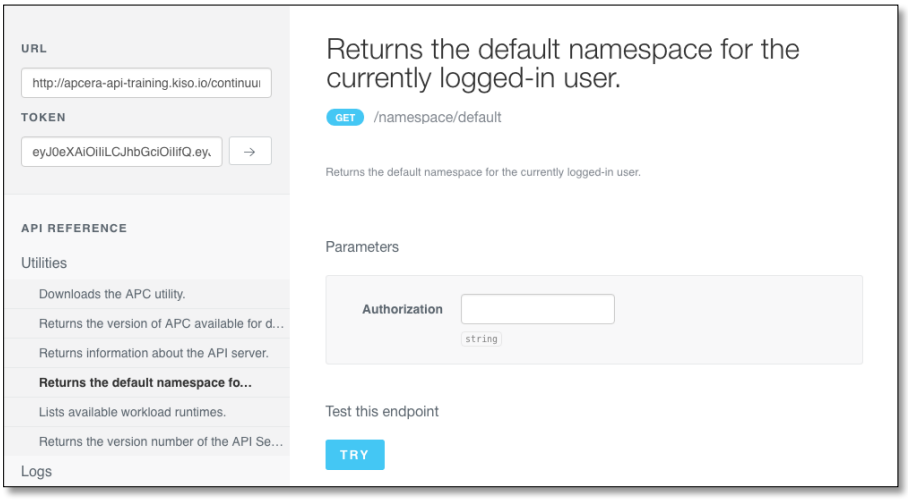
	{
	"target": "https://kiso.io",
	"tokens": {
	"http://kiso.io": "**Bearer** eyJ0eXAiOiIiLCJhbGciOiIifQ....”
	},
	...

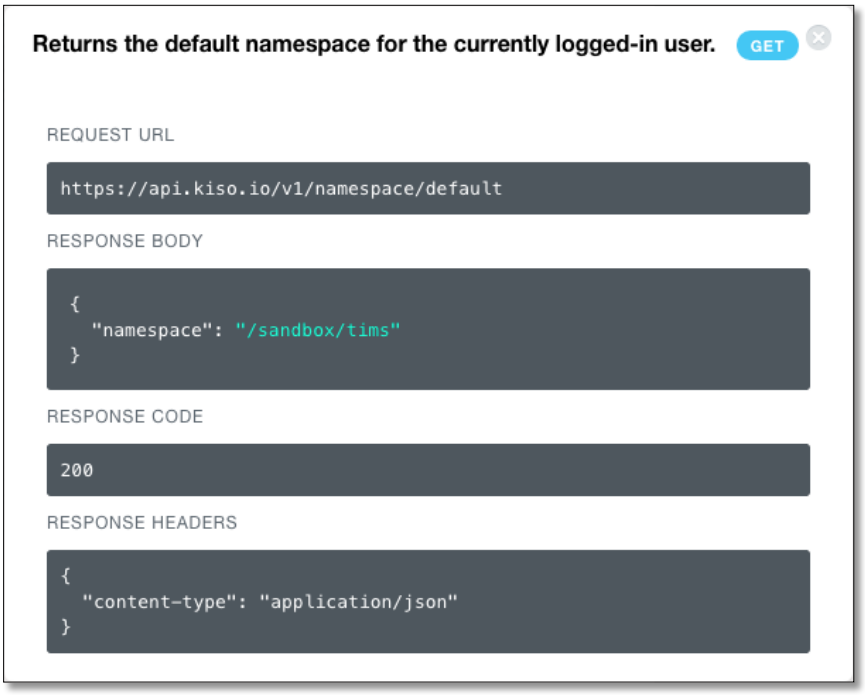
B.1.6	Open the API Explorer (http://apcera-api-training.kiso.io) and paste the token into the **Token** field: 
B.1.7	Click the arrow (→) to set the API token for your browser session.

Exercise C. Using the API Explorer to make a basic API call

Now you are ready to make some API calls using the API Explorer.

Task 1: Retrieve your default namespace

Step	Instruction
C.1.1	In the API Reference section, select **Utilities** > **Returns the default namespace**.
C.1.2	A form opens that lists the API's method and endpoint (**GET /namespace/default**) and its parameters. 
C.1.3	Click **Try** to make the API request. In a moment, the HTTP response body, code and headers are displayed. As you can see the response body is a JSON-encoded object with a single field, **namespace**, whose value is the default namespace for the current user.



Exercise Review

In this lab, you invoke the basic Apcera APIs to retrieved and update a job state.

End of the Lab

Lab Introduction
In this lab, you will use the Apcera REST API to create a new package.

Lab Exercise

This lab comprises the following exercises:

A Create a NATS server job from a Docker image
B Create a NATS client job from a Docker image
C Start the NATS client app using API Explorer

Lab Prerequisites
This lab assumes that you have successfully logged into the training cluster using APC.

Related Documentation
- Apcera REST API Model Objects: http://docs.apcera.com/api/api-models/
- Apcera REST API Recipes: http://docs.apcera.com/api/api-tasks/

Lab Instruction
For each exercise, complete the steps.

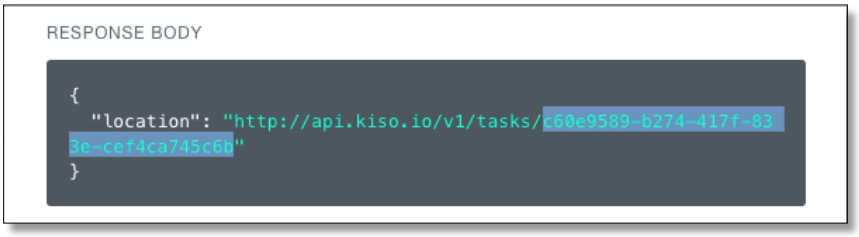
Exercise A: Create a NATS server job from a Docker image

In this exercise you'll create a new job from the official NATS Docker image using the `POST /v1/jobs/docker` API.

Task 1: Create jobs

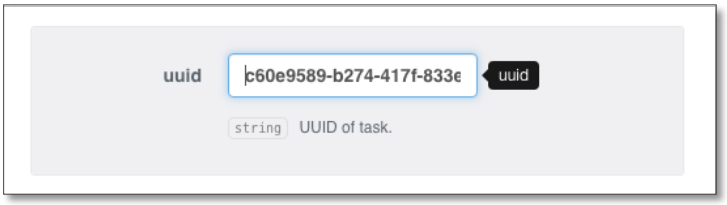
Step	Instruction
A.1.1	Under the **Jobs** API category click **Create a new job from Docker image** to open the `POST /v1/docker/jobs` API form.

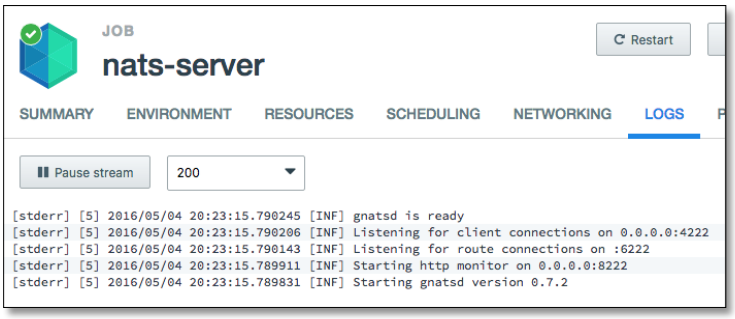
A.1.2	In the Job field enter the following JSON, replacing **<username>**in the **job_fqn** field with your username.
	
	`{`
	`"image_url":"https://registry-1.docker.io/library/nats:latest",`
	`"job_fqn":"job::/sandbox/<username>::nats-server",`
	`"start":true`
	`}`
	
	NOTE: If you wish, you can copy and paste the command from the **Lab2_commands.txt** file under **/apceratraining/DEV102/Labfiles/Lab_2 folder**.

A.1.3	Click **Try** to make the API request. This starts an asynchronous task on the server that downloads each Docker image layer, creates a package from the layers, and creates the app. If successful, the response body contains a `location` field whose value is an API endpoint you can call to check the status of the app creation process. You will do this next.
A.1.4	Copy the UUID from the end of the `location` field's URL to your clipboard.  **Troubleshooting Tips:** Use online JSON validator (http://jsonlint.com/) to make sure that there is no syntax erorr.

Task 2: Verify the job was created successfully

Next you'll confirm that the app was created successfully by calling the **`GET /tasks/{uuid}`** endpoint, providing the UUID you copied to your clipboard previously.

Step	Instruction
A.2.1	Under the **Jobs** API category click **Returns a list of task events for a given task** to open the `GET /tasks/{uuid}` endpoint.
A.2.2	Locate the endpoint's **UUID** parameter field and paste the UUID from your clipboard into the input field. 
A.2.3	Click **Try** (or press return) to make the request. If the job was created successfully the **`state`** field in the response should be set to **"complete"** and the **`errored`** field should be set to **"unerrored"**, as shown below. 
A.2.4	Review the events reported in the response body. The flow of the events are: 1. "stage": "Pulling Docker image" a. "checking policy" b. "checking if package FQN is taken" c. "fetching image metadata" d. "creating package" e. "all layers downloaded" 2. "stage": "Creating job" a. "tagging package"

A.2.5	Open the web console, locate the nats-server app and click the **Logs** tab. You should see log messages indicating the NATS server is running and listening for connections on port 4222. 

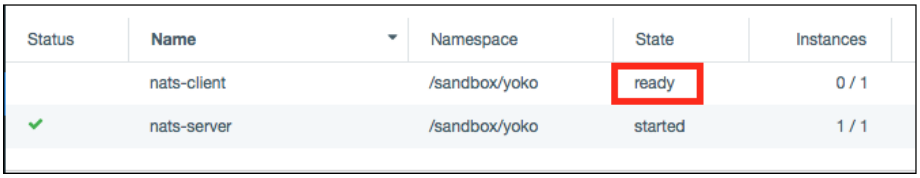
Exercise B. Create a NATS client job from a Docker image

Next you'll create a NATS client application from another Docker image, and create a job link between the client and server.

Task 1: Create a NATS client app from a Docker image

Step	Instruction
B.1.1	Under the **Jobs** API category click **Create a new job from Docker image**.

B.1.2	In the Job field enter the following JSON, replacing ***<username>***in the **job_fqn** field with your username.
	
	`{`
	`"image_url":"https://registry-1.docker.io/apcerademos/nats-ping:latest",`
	`"job_fqn":"job::/sandbox/`***`<username>`***`::nats-client",`
	`"start":false`
	`}`

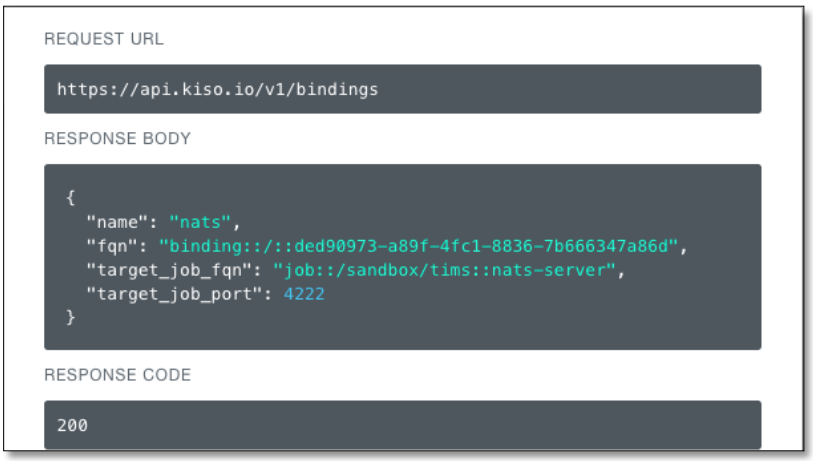
B.1.3	Click **Try** to make the API request.
B.1.4	Open the web console and locate the **nats-client** application to make sure it was created successfully. The app's **State** field should be "Ready", as shown below. 

Task 2: Create a job link between the NATS server and client

The nats-client application has logic that checks for an environment variable to know where to connect to the NATS server. You will use the `POST /v1/bindings` endpoint to create a job link between the nats-client and nats-server applications so that they can communicate.

Step	Instruction
B.2.1	Under the **Services and Bindings** API category click **Creates a new binding between a job and a service, or between two jobs**.

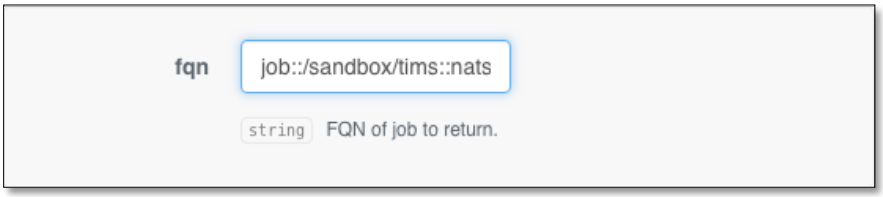
B.2.2	In the **binding** input field enter the following JSON, replacing ***<username>*** in the **job_fqn** field with your username:
	
	`{`
	`"job_fqn": "job::/sandbox/<username>::nats-client",`
	`"name": "nats",`
	`"target_job_fqn": "job::/sandbox/<username>::nats-server",`
	`"target_job_port": 4222`
	`}`
	
	This defines a job link between the nats-client (the source) and nats-server (the target)
	on port 4222.

B.2.3	Click **Try** to create the binding. An HTTP 200 response code indicates the binding was created successfully. 

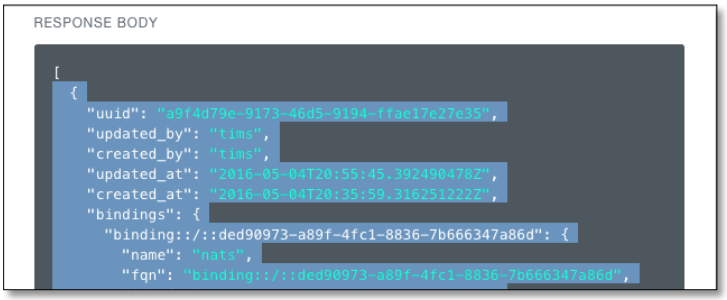
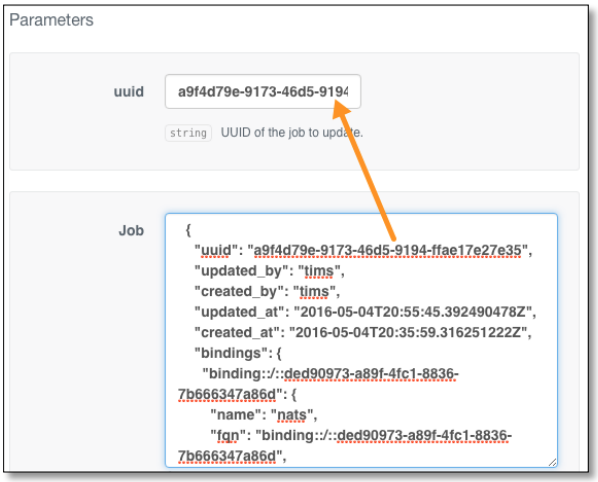
Exercise C. Start the NATS client app

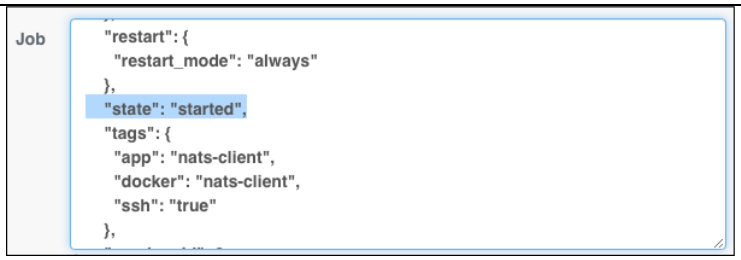
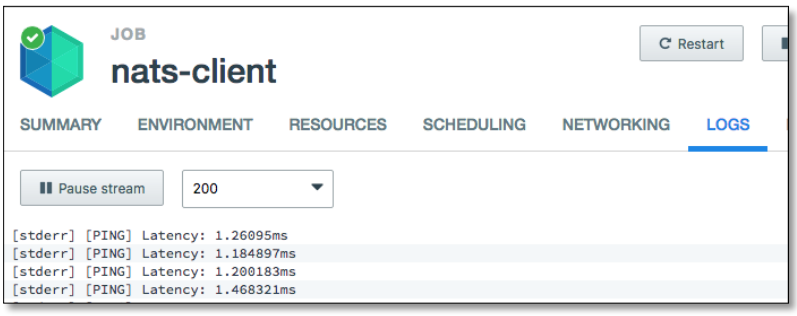
Next you'll start the NATS client application with the `PUT /v1/jobs` endpoint. To start a job you set its **`state`** property to **`"started"`**. In general, updating a job definition is a two-step process: calling `GET /v1/jobs` to obtain a current snapshot of the job's JSON description, modifying the JSON as desired, and then `PUT /v1/jobs` passing it the updated JSON.

Task 1: Get the current job description for the NATS client app

Step	Instruction
C.1.1	Under the **Jobs** API category click **Create a new job from Docker image**.
C.1.2	In the **FQN** parameter input field, enter the FQN of the NATS client application you created previously (for example, `job::/sandbox/username::nats-client`). 
C.1.3	Press **Try** to run the request. A successful response returns an array of job objects that match the query parameter. In this case, obviously, there is just one element in the array. 

Task 2: Update the NATS client job description

Step	Instruction
C.2.1	Copy the JSON description from the response body; don't include the array brackets (the [ ]'s), just the array contents. 
C.2.2	Click **Update a Job** under the Jobs API category.
C.2.3	Paste your clipboard contents into the **Job** parameter field.
C.2.4	Copy the job's UUID from the JSON to the UUID field, as shown below. 
C.2.5	Locate the **`state`** field in the job's JSON description and change its value to `"started"`.

	
C.2.6	Click **Try** to send the request.
C.2.7	Open the web console to verify that the job has started. 
C.2.8	Click the nats-client job and select the Logs tab. You should see message received by the client from the NATS server. 

Exercise Review

In this lab, you invoke the basic Apcera APIs to retrieved and update a job state.

End of the Lab

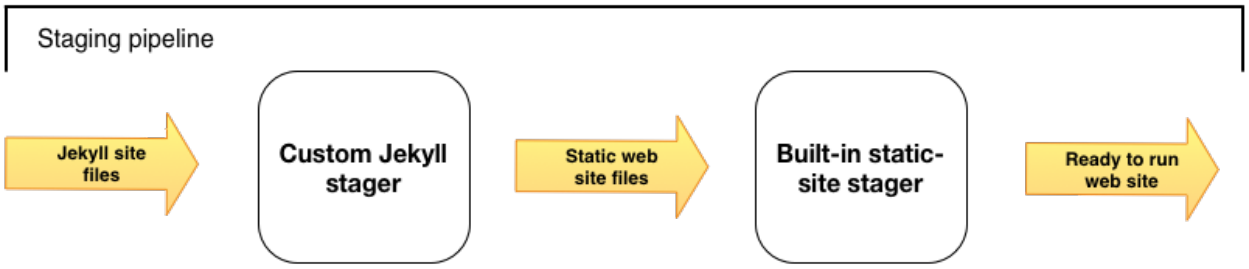
Apcera Training

Lab 3: Extending a built-in staging pipeline

Lab Introduction

The built-in static-site staging pipeline deploys static HTML sites using the Nginx web server. Jekyll is a command-line tool that takes a set of raw text files in various formats, runs it through a converter (like Markdown) and generates a complete, ready-to-publish static website.

In this lab you will create a new a staging pipeline that generates a static web site from a Jekyll project. You will append the built-in static-site stager job to the new staging pipeline.



- Downloads original package (Jekyll project files)
- Installs Jekyll (`bundle install`)
- Builds site (`jekyll build`)
- Uploads modified package

- Downloads modified package (web site files)
- Adds Nginx package as package dependency
- Sets start command and env variables

Lab Exercise

This lab comprises the following exercises:

A. Create a Jekyll stager using the Ruby stager library
B. Deploy the newly created stager
C. Deploy a Jekyll project with new staging pipeline
D. Updating the stager job

Lab Prerequisites

This lab assumes:

- Your system has Ruby installed
- You have successfully logged into the training cluster using APC.

Related Documentation

- Stager Library API: http://docs.apcera.com/api/stager-api/
- Ruby Library for the Stager API: http://docs.apcera.com/api/stager-api-lib/

Lab Instruction

For each exercise, complete the steps.

Exercise A: Create a Jekyll stager using the Ruby stager library

The first step is to create a Ruby project for the Jekyll stager. This involves creating a **Gemfile** that lists the gem(s) you want to include in the project and running `bundle install` to create a **Gemfile.lock** file that lists the exact.

Task 1: Prepare your Ruby environment

Step	Instruction
A.1.1	Create a project directory where you will keep your stager files.

A.1.2	Create a file named **Gemfile** (with no file extension) and add the following to the file:
	
	`source "https://rubygems.org"`
	`gem "apcera-stager-api"`

A.1.3	In a terminal, move to the project folder and `bundle install` to download and install the gem and its dependencies:
	
	`bundle install`
	
	NOTE: If you get an error about “bundle not found” run the following:
	
	`gem install bundler`

Task 2: Create a Ruby stager file

Step	Instruction
A.2.1	Create a file named **stager.rb** in the project folder. (This filename is used by convention only, and isn’t required.)

A.2.2	Add the following code to the file:
	
	`#!/usr/bin/env ruby`
	
	`require "bundler"`
	`Bundler.setup`
	
	`require "apcera-stager-api"`

```
STDOUT.sync = true

stager = Apcera::Stager.new

# Download the package from the staging coordinator.
puts "Downloading application package..."
stager.download

# Extract the package contents (Jekyll project file)
puts "Extracting package contents..."
stager.extract('app')

# Install Jekyll dependencies
puts "Installing dependencies..."
stager.execute_app("bundle install --path ../site-
vendor/bundle --binstubs ../site-vendor/bundle/bin --
deployment")

# Set some environment variables to help Ruby handle file
encoding
ENV["LC_ALL"] = "en_US.UTF-8"
ENV["LANG"] = "en_US.UTF-8"

# Build the site with Jekyll
puts "Building the site..."
stager.execute_app("../site-vendor/bundle/bin/jekyll build --
trace")

# Jekyll puts the generated web site files in a folder named
"_site".
# This tells the stager library to upload that folder's
contents.
stager.app_path = "/tmp/staging/app/_site"

# Upload the final package (static site file) to the staging
coordinator.
puts "Staging complete, uploading package to staging
coordinator."
stager.complete
```

NOTE: If you wish, you can copy and paste the command from the **stager.rb** file under **/apceratraining/DEV102/Labfiles/Lab_3** folder.

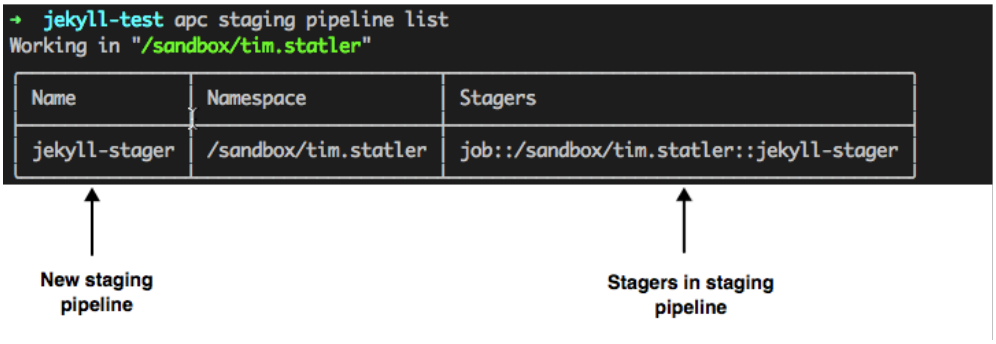
A.2.3 Save your changes to **stager.rb**.

A.2.4	Make **stager.rb** executable: `chmod +x stager.rb`

Exercise B: Deploy the newly created stager

Next you'll deploy the stager and create a new staging pipeline to contain it.

Task 1: Deploy the Jekyll stager and staging pipeline

Step	Instruction
B.1.1	In a terminal move to the jekyll-stager.
B.1.2	To deploy the stager named jekyll-stager and add it to a new staging pipeline with the same name, run the following command: `apc stager create jekyll-stager --start-command="./stager.rb" --staging=/apcera::ruby --pipeline --allow-egress` Note that the stager is allowed network egress so that it can install the stager's application dependencies. The `--pipeline` option creates a staging pipeline that contains the stager job; the staging pipeline is given the same name as the stager.
B.1.3	When complete, run the following command to list the staging pipelines in your namespace: `apc staging pipeline list` The output should include a staging pipeline named **jekyll-stager** that's composed of a single stager job with the same name. 

Task 2: Append the static-site stager job to the staging pipeline

Next you'll append the built-in static site stager job (`job::/apcera/staging::static-site`) to the jekyll-stager staging pipeline using the `apc staging pipeline append` command.

Step	Instruction
B.2.1	Run the following command to append the static-site stager. `apc staging pipeline append jekyll-stager /apcera/stagers::static-site`
B.2.2	List your staging pipelines again: `apc staging pipeline list` You should see that the Stagers column lists your `jekyll-stager` job and the `static-site` stager. The stagers are listed in the order that they will run in the staging pipeline. Name | Namespace | Stagers jekyll-stager | /sandbox/tim.statler | job::/sandbox/tim.statler::jekyll-stager, job::/apcera/stagers::static-site

Exercise C. Deploy a Jekyll project with new staging pipeline

Now you're ready to deploy a Jekyll project using the staging pipeline. The Apcera sample-apps repository contains a sample Jekyll project you can deploy, or you can use another project.

Task 1: Deploy the same Jekyll project

Step	Instruction
C.1.1	In a terminal window, move Jekyll project folder: `cd ~/sample-apps/example-jekyll`

C.1.2	Deploy the app using the `jekyll-stager` staging pipeline:
	
	`apc app create my-jekyll-site --staging=jekyll-stager --start --batch`
	
	The command output should indicate that there are two stagers defined:
	
	`[staging] Beginning staging with 'stagpipe::/sandbox/tim.statler::jekyll-stager' pipeline, 2 stager(s) defined.`
	`[staging] Launching instance of stager 'jekyll-stager'...`
	`...`
	`[staging] Launching instance of stager 'static-site'...`
	`...`
	`App should be accessible at "http://my-jekyll-site-it0c05.kiso.io"`

C.1.3	Open the link where the app is deployed to see the generated site. 

Exercise D. Updating the stager job

If you need to make changes to a stager job once it's been deployed, you can either manually replace the stager's package with the updated code, or delete the stager and staging pipeline and re-create them.

Task 1: Update the stager package

Step	Instruction
D.1.1	Open **stager.rb** in an editor and add a `puts` method to output a new message to standard out. `puts "THIS IS A NEW PACKAGE!!"`
D.1.2	Save the changes to **stager.rb**.
D.1.3	Create a compressed tar file (.tar.gz) that contains the contents of the stager app's folder. For instance, from inside the stager directory you can run the following: `tar -zcvf package.tar.gz *`
D.1.4	Use the `apc package replace` command to replace the stager's deployed package with the contents of the archive: `apc package replace jekyll-stager package.tar.gz --staging /apcera::ruby`

D.1.5	Re-deploy the Jekyll sample project with the updated stager.
	
	`apc app deploy my-jekyll-site --staging=jekyll-stager --start --batch`
	
	You should see the new log message in the output, as shown below.
	
	`[staging] Beginning staging with 'stagpipe::/sandbox/tim.statler::jekyll-stager' pipeline, 2 stager(s)`
	`[staging] Launching instance of stager 'jekyll-stager'...`
	`[staging] THIS IS A NEW PACKAGE!!`
	`[staging] Downloading application package...`

Exercise Review

In this lab, you created a new staging pipeline that generates a static web site from a Jekyll project, and then appended the built-in static-site stager job to the new staging pipeline.

End of the Lab

Apcera Training

Lab 2: Creating an Apache static web site stager

Lab Introduction

In this lab you'll create a staging pipeline for deploying static websites that run on Apache. The stager includes a customized Apache configuration file (httpd.conf) that sets Apache's document root to the application directory, where the web site files will live. It also sets the app's start command to start Apache using the custom configuration file.

Note: The Apcera Platform already provides the static-site staging pipeline for staging web sites, and you are encouraged to use that one for production purposes. The output of this lab is for instructional purposes, only.

Lab Exercise

This lab comprises the following exercises:

A. Create and deploy a custom stager
B. Deploy and configure a web site with the new staging pipeline

Lab Prerequisites

This lab assumes that you have set up your lab environment. If you have not done so, please follow the lab setup guide now.

Related Documentation

- Stager Library API: http://docs.apcera.com/api/stager-api/
- Ruby Library for the Stager API: http://docs.apcera.com/api/stager-api-lib/

Lab Instruction

For each exercise, complete the steps.

Exercise A: Create and deploy a customer stager

First you'll use the Ruby staging library to create the Apache stager and staging pipeline.

Task 1: Prepare your Ruby environment

Step	Instruction
B.1.1	Create a project directory where you will keep your stager files.

B.1.2	Create a file named **Gemfile**(with no file extension) in the project folder and add the following to the file:
	
	`source "https://rubygems.org"`
	`gem "apcera-stager-api"`

B.1.3	In a terminal, move to the project folder and `bundle install` to download and install the gem and its dependencies: `bundle install`

Task 2: Create and deploy the Ruby stager

Step	Instruction
B.2.1	Create a file named **stager.rb** in the project folder.

B.2.2	Add the following code to the file:
	
	`#!/usr/bin/env ruby`
	
	`require "bundler"`
	`Bundler.setup`
	
	`require "apcera-stager-api"`
	
	`STDOUT.sync = true`
	
	`stager = Apcera::Stager.new`
	
	`# Download the package from the staging coordinator.`
	`puts "Downloading application package..."`

	stager.download
	
	# Add Apache as dependency
	puts "Adding Apache as dependency..."
	should_restart = stager.dependencies_add("package", "apache")
	if should_restart == true
	 stager.relaunch
	end
	
	# Extract package (web site files) to app/ folder.
	puts "Extracting package contents..."
	stager.extract('app')
	
	# Copy stager's conf/ folder (that contains httpd.conf)
	# to the staged app's conf folder (/conf).
	puts "Copying conf folder..."
	FileUtils.copy_entry("conf", File.join(stager.app_path, "conf"))
	
	# Add a start command to start Apache with the custom httpd.conf
	start_cmd = "sudo apachectl -f /conf/httpd.conf -DFOREGROUND"
	puts "Setting start command to '#{start_cmd}'"
	stager.start_command = start_cmd
	
	# Finish staging, this uploads the final package to the staging coordinator.
	puts "Uploading package to staging coordinator..."
	stager.complete
	
	NOTE: If you wish, you can copy and paste the command from the **stager.rb** file under **/apceratraining/DEV102/Labfiles/Lab_4** folder.

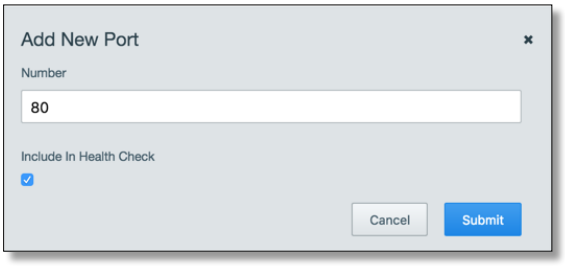
B.2.3	Save your changes to **stager.rb**.
B.2.4	Create a folder named **conf** in the stager project folder.
B.2.5	Copy the custom Apache config file **httpd.conf** file from the lab folder to the stager's **conf/** folder.
B.2.6	Make **stager.rb** executable: chmod +x stager.rb

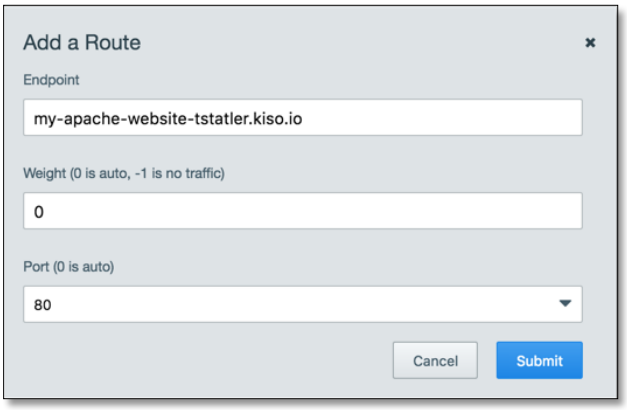
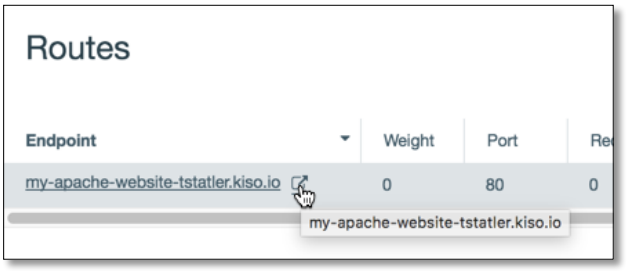
B.2.7	Deploy the stager and automatically create a staging pipeline for it with the **`--pipeline`** flag: `apc stager create apache-site-stager --start-command="./stager.rb" --staging=/apcera::ruby --pipeline`

Exercise B: Deploy and configure a web site with the new staging pipeline

Now you're ready to deploy a web site using the new stager. The Apcera sample-apps repository contains a sample project you can deploy, or you can use your own web site files. You'll also need to open port 80 where Apache is configured to listen for incoming connections and add a route to that port.

Task 1: Deploy a web site using the stager

Step	Instruction
B.1.1	In a terminal window, move to the sample-apps/example-static project folder, or other folder that contains web site files (should at least contain an index.html file). `cd ~/sample-apps/example-static`
B.1.2	Deploy the app using the **`apache-site-stager`** staging pipeline: `apc app create website-apache --staging apache-site-stager --disable-routes` The **`--disable-routes`** is necessary so that a default port isn't automatically selected for use. You will need to expose port 80 and create a custom route after job creation, as explained next.
B.1.3	Open the web console and open the details view for the new application.
B.1.4	On the job's **Networking** tab, click **Add Port** and enter 80 in the Number field, and enable the **Include in Health Check** so that if Apache isn't responding on that port the system will flag the job as unhealthy.
B.1.5	Click **Submit**. 
B.1.6	Click **Add Route**. A route endpoint is generated for you. Modify this route to make it unique by adding your username, as shown below. This is so your route doesn't conflict with other students' routes.

B.1.7	Click **Submit**. 
B.1.8	Click **Start** on the job's details view to start the app. (You can ignore any errors in the app logs that say "`Could not reliably determine the server's fully qualified domain name...`").
B.1.9	Once the job has started, click the open external route icon to view the running site.  The web site should launch with "*Hello World!*" message. 

Exercise Review

In this lab you created a staging pipeline for deploying static websites that run on Apache. The stager includes a customized Apache configuration file (httpd.conf) that sets Apache's document root to the application directory, where the web site files live. It also sets the app's start command to start Apache using the custom configuration file.

End of the Lab

Apcera Training

Lab 5: Creating a simple service gateway

Lab Introduction

In this lab you will create a new service gateway that generates service bindings for use with the OpenWeatherMap APIs (http://openweathermap.org/api). This is a very simple service gateway that always the same binding URL.

Lab Exercise

This lab comprises the following exercises:

A. Download and examine the service gateway code
B. Create and deploy the service gateway application
C. Create a weather service and bind it to a capsule

Lab Prerequisites

You should have successfully logged into the training cluster using APC and the web console.

Related Documentation

- Service Gateway API Overview: http://docs.apcera.com/api/service-gateway-api/
- Service Gateway API Reference: http://docs.apcera.com/api/service-gateway-api-endpoints/

Lab Instruction

For each exercise, complete the steps.

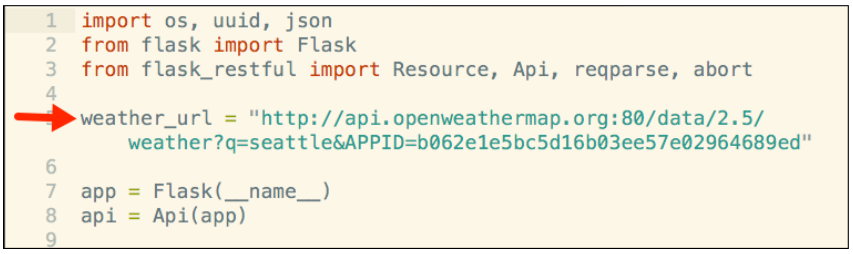
Exercise A: Download and examine the service gateway code

The “weather” service gateway implements the following methods:

`GET /`	Lists available service gateway methods endpoints
`POST /services`	Creates a new service
`DELETE /services/:id`	Deletes an existing service
`POST /bindings`	Creates a new binding
`DELETE /bindings/:id`	Delete an existing binding

This type of service gateway is called a “two-tier” service gateway because it manages services and bindings, but not providers.

Task 1: Launch the API Explorer

Step	Instruction
B.1.1	Browse to a folder where you cloned (or downloaded) the **apceratraining** repository in ***Lab 0***, Task 1.
B.1.2	Open the **app.rb** file under **/apceratraining/DEV102/Labfiles/Lab_5** in a text editor.
B.1.3	Note that the OpenWeather API URL. This is the binding URL provided to jobs. For this lab, the weather location and OpenWeatherMap API key are hard-coded. In a subsequent lab, you will customize the code to allow the user provide a custom region and API key when creating the service binding. 

```
1  import os, uuid, json
2  from flask import Flask
3  from flask_restful import Resource, Api, reqparse, abort
4
5  weather_url = "http://api.openweathermap.org:80/data/2.5/
       weather?q=seattle&APPID=b062e1e5bc5d16b03ee57e02964689ed"
6
7  app = Flask(__name__)
8  api = Api(app)
9
```

B.1.4

Note that this gateway exposes the following service gateway methods:

```
# GET / returns the 2-tier required SG capabilities
@api.resource('/')
class Capability(Resource):
   def get(self):
       return {
           '/services': {
               'POST': {}
           },
           '/services:id': {
               'DELETE': {}
           },
           '/bindings': {
               'POST': {}
           },
           '/bindings/:id': {
               'DELETE': {}
           }
       }
```

B.1.5

The **POST /bindings** handler creates the JSON response that contains the binding URL.

```
@api.resource('/bindings')
class Bindings(Resource):

 def post(self):
     args = self.parser.parse_args()

     req_headers = {}
     req_headers['X-Service-State'] = json.loads(args['X-
Service-State'])

     resp_json = {}
     resp_json['id'] = str(uuid.uuid4())
     resp_json['url'] = weather_url
     resp_json['protocol'] = {'scheme': 'http'}
     resp_json['name'] = args['name']
     resp_json['description'] = args['description']
     resp_json['service_id'] = args['service_id']
     resp_json['params'] = {}

     resp_headers = {}
     resp_headers['X-Binding-State'] =
json.dumps(args['params'])
```

Exercise B: Create and deploy the service gateway application

First you'll deploy the Python application that acts as the service gateway. The application uses the Flask application framework to define the necessary URL routes.

Task 1: Create the service gateway app

Step	Instruction
B.1.1	Open a terminal and change to the weather-service project folder.
B.1.2	To create the app, run the following command: `apc app create weather-sg --disable-routes --start-cmd "./bin/python app.py" --start`
B.1.3	Once deployed, execute the following command to promote the app to a service gateway of type **weather-<*username*>**. `apc gateway promote weather-sg --type weather-<username>` Note: The service type must be unique within the cluster; therefore, be sure to replace **<*username*>** with your user name.
B.1.4	To confirm, list the service gateways in your sandbox: `apc gateway list`

Exercise C: Create and deploy the service gateway application

Next you'll create a weather service from the service gateway and bind it to a capsule. The binding results in an environment variable being added to the target job that contains the URL that returns the weather data. You'll then test the binding by connecting to the capsule and cURLing the endpoint set in the capsule's environment.

Task 1: Bind a weather service to a capsule

Step	Instruction
C.1.1	In a terminal run the following command: `apc service create weather-seattle --type weather-<username> -- region seattle`
C.1.2	Execute the following command to create a capsule. `apc capsule create my-capsule --image linux`

C.1.3	Bind the capsule to the service, resulting in two environment variables being added to the capsule, `HTTP_URL` and `WEATHERSEATTLE_URL`.
	
	`apc service bind weather-seattle --job my-capsule`
	…
	`Waiting for the job to start...`
	`Binding Environment Variables`
	`"HTTP_URI"`
	`"WEATHERSEATTLE_URI"`

C.1.4	Execute the following command to connect to the capsule over SSH.
	
	`apc capsule connect my-capsule`
	
	`root@ip-169-254-0-5:/root#`

C.1.5	Use cURL to make an HTTP request to the URL contained by the **`WEATHERSEATTLE_URL`** environment variable:
	
	`curl $WEATHERSEATTLE_URL`
	
	Expected Output:
	
	`{"coord":{"lon":-122.33,"lat":47.61},"weather":[{"id":800,"main":"Clear","description":"clear sky","icon":"02d"}],"base":"cmcstations","main":{"temp":288.754,"pressure":1017.68,"humidity":57,"temp_min":288.754,"temp_max":288.754,"sea_level":1028.46,"grnd_level":1017.68},"wind":{"speed":1.69,"deg":22.5026},"clouds":{"all":8},"dt":1463151607,"sys":{"message":0.0044,"country":"US","sunrise":1463142772,"sunset":1463197156},"id":5809844,"name":"Seattle","cod":200}`

Exercise Review

In this lab, you created a new service gateway that generates service bindings for use with the OpenWeatherMap API.

End of the Lab