



DEV101 - Developing with Apcera Platform

Lab Book



APCERA

Apcera Training

Labs Overview

This document lists the lab exercises that comprise Apcera Training.

Lab 1: Deploy Diverse Workloads

- A. Create Node web app
- B. Create Go app (NATS client)
- C. Create Docker workload (NATS server)
- D. Link jobs and explore dynamic binding
- E. Create, snapshot, and connect to a capsule

Lab 2: Orchestrate Services

- A. Create a PostgreSQL service and bind your app to it
- B. Create a MySQL service and bind your app to it
- C. Using Docker as DB service provider
- D. Bind a capsule to an existing service
- E. Create MySQL App from package with No Delete Hook

Lab 3: Extending Apcera

- A. Create App and Services Using Manifest
- B. Write and Build Custom Package
- C. Move workloads using Job Scheduling Tags
- D. Enabling broadcast and multicast routes

Lab 4: Troubleshoot Jobs

- A. Configure job log drain
- B. Perform live debugging using console app
- C. Debug an app failing to stage



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

Lab Introduction

In this lab you deploy a diverse set of workloads using Apcera, including apps written in various languages, a Docker image, and an OS capsule. You experience how jobs run in isolation and how to update an app. You use APC to create the workloads, and the Web Console to manage them.

Lab Exercise

This lab comprises the following exercises:

- A. Deploy a Node Web App
- B. Deploy a Go App (NATS Client)
- C. Deploy a Docker Workload (NATS Server)
- D. Link jobs and explore dynamic binding
- E. Create, Snapshot, and Connect to a Capsule OS

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

- Working with Jobs: <http://docs.apcera.com/jobs/jobs-toc/>
- Linking Jobs: <http://docs.apcera.com/jobs/job-links/>.
- Using Filecopy and SCP: <http://docs.apcera.com/jobs/scp-filecopy/>

Lab Instruction

For each exercise, complete the steps.



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

Exercise A: Deploy Node App

In this exercise you deploy an app written in Node.js and start it.

Task 1: Deploy Node App

Step	Instruction
A.1.1	<p>Change your working directory to demo-node-todo.</p> <pre>cd sample-apps/demo-node-todo</pre>
A.1.2	<p>Explore the demo-node-todo sample app:</p> <pre>ls cat app.js</pre> <p>The demo-node-todo app is a Node.js application that provides a web interface for receiving user input and two backend databases (PostgreSQL and MySQL) for storing data. You can learn more about this app at http://todomvc.com/.</p>
A.1.3	<p>Explore the <code>apc app create</code> command.</p> <pre>apc help app create</pre> <p>By default, any app you create is given a default route and port that is chosen at runtime and exposed by the "PORT" environment variable. In Apcera, a route maps a URL to an app. If the app is not a web app, you should disable routes using <code>--disable-routes</code>. In this case, you are deploying a web app so you need a route.</p> <p>Apcera creates a default route using the following convention: <code><app-name>-6chars.<cluster-name>.<tld></code>. Alternatively, you can use the <code>--route</code> command to specify a route. A route name cannot use forbidden characters, such as an "@". In addition, the route name must be unique across the cluster.</p> <p>In these labs we will use the default route naming scheme, permitted by policy:</p> <pre>on route::/http/io/kiso/ { if (job nameMatch "job::/sandbox") { permit map } }</pre>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

A.1.4	<p>Create the Node app.</p> <pre>apc app create my-node-todo-app</pre> <p>Where “my-node-todo-app” is the app name, which is a user-defined string.</p>
A.1.5	<p>Complete the app deployment process by accepting the following default settings:</p> <ul style="list-style-type: none">• Deploy path: Accept the default deployment path by pressing the enter key.• Instances [1]: Accept the default 1 instance by pressing the enter key.• Memory [256MB]: Accept the default RAM allocation by pressing the enter key. <p>NOTE: Use the --batch flag to deploy with the defaults and avoid being prompted.</p> <p>Review the Application Settings as presented, and assuming these settings are correct, press the enter key (Y) to create the app.</p> <p>The app is packaged and uploaded to the system, then staged for deployment using the system-provided nodejs stager. When staging is complete, you see the Success! message.</p> <pre>[staging] Subscribing to the staging process... [staging] Beginning staging with 'stagpipe::/apcera::nodejs' pipeline, 1 stager(s) defined. [staging] Launching instance of stager 'nodejs'... [staging] Downloading package for processing... [staging] Stager needs relaunching [staging] Launching instance of stager 'nodejs'... [staging] Downloading package for processing... [staging] Executing "npm install --production" [staging] mysql@2.9.0 node_modules/mysql [staging] ─ bignumber.js@2.0.7 [staging] ─ readable-stream@1.1.13 (isarray@0.0.1, inherits@2.0.1, string_decoder@0.10.31, core-util-is@1.0.2) [staging] Staging is complete. Creating app "my-node-todo-app"... done Start Command: node ./app.js App may be started with: > apc app start my-node-todo-app Success!</pre>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

A.1.6	Execute the following command to list the existing apps. <code>apc app list</code>
--------------	---

Task 2: Start the Node app

Step	Instruction								
A.2.1	<p>If the app is not running already, execute the following command to start it:</p> <pre>apc app start my-node-todo-app</pre> <p>You should see that a single instance of the app is started, indicated by the <i>Success!</i> message. Note also that the output gives you the route port (4000) and the route URL.</p> <p>NOTE: You can start the app when you create it using either of the following commands:</p> <pre>apc app create my-node-todo-app --start apc app create my-node-todo-app -s</pre>								
A.2.2	<p>Execute the following command to verify the health of the instance and environment details.</p> <pre>apc app health my-node-todo-app</pre> <p>You should see that a single instance of the my-node-todo-app is “Running” and that the Health Score is 100%.</p> <table><tr><td>Job:</td><td>job::/sandbox/continuumtraining2015::my-node-todo-app</td></tr><tr><td>Status:</td><td>Running</td></tr><tr><td>Health Score:</td><td>100%</td></tr><tr><td>Running Instances:</td><td>1/1</td></tr></table>	Job:	job::/sandbox/continuumtraining2015::my-node-todo-app	Status:	Running	Health Score:	100%	Running Instances:	1/1
Job:	job::/sandbox/continuumtraining2015::my-node-todo-app								
Status:	Running								
Health Score:	100%								
Running Instances:	1/1								





DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

A.2.3	<p>Execute the following command to display the details about my-node-todo-app.</p> <pre>apc app show my-node-todo-app</pre> <p>This command displays additional details about your app, including its fully qualified namespace (FQN), state, instances, resources, process(es), ports, routes, tags, packages, and environment variables. Use will use these properties during the labs.</p>
--------------	--

Task 3: Explore your deployed app in the Web Console

Step	Instruction										
A.3.1	Log in to the web console at the following URL: https://console.kiso.io/ .										
A.3.2	<p>Under JOBS, select Apps. You should see your deployed my-node-todo-app in your /sandbox/<username> namespace.</p> <div><div><div> APCERA</div><div>New</div><div><div>Home</div><div>Cluster</div></div><div><div>JOBS</div><div><div>Apps</div><div>Capsules</div><div>All</div></div></div></div><div><div> APPS</div><div><div>APP LIST</div><div>POLICY</div></div><div><div>Filter...</div></div><table><tr><th>Status</th><th>Name</th><th>Namespace</th><th>State</th><th>Instances</th></tr><tr><td>✓</td><td>my-node-todo-app</td><td>/sandbox/yoko</td><td>started</td><td>1 / 1</td></tr></table></div></div>	Status	Name	Namespace	State	Instances	✓	my-node-todo-app	/sandbox/yoko	started	1 / 1
Status	Name	Namespace	State	Instances							
✓	my-node-todo-app	/sandbox/yoko	started	1 / 1							




DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

A.3.3

Select your app so that the SUMMARY view appears. Verify that 1 instance of the app is running. You will also see resource utilization statistics that are graphed for this instance.

 **JOB**
my-node-todo-app

[Restart](#) [Stop](#) [Delete](#)

[SUMMARY](#) [ENVIRONMENT](#) [RESOURCES](#) [SCHEDULING](#) [NETWORKING](#) [LOGS](#) [POLICY](#) [AUDIT](#)

Info

FQN	job::sandbox/yoko::my-node-todo-app
Created By	yoko
Allow SSH	<input type="checkbox"/>
Allow Egress	<input type="checkbox"/>
Restart Mode	Always
Running Instances	1
Expected Instances	<input type="text" value="1"/>

Labels

No data to display.

[Add Label](#)

Status

State	started
Status	ok
Health Score	1.00
Flapping	false



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

Exercise B: Deploy Go App (NATS Client)

In this exercise you create a Go app that is a client of the NATS messaging system (gnatsd), which you will deploy in the next exercise.

Task 1: Deploy a sample app, go-nats

Step	Instruction
B.1.1	<p>Change your working directory to sample-apps/nats-ping.</p> <pre>cd sample-apps/nats-ping</pre>
B.1.2	<p>Explore the nats-ping app directory and source code.</p> <pre>ls cat nats-ping.go</pre> <p>The client publishes NATS message on subject 'test' when you hit '/echo' on the endpoint. The client also subscribes to subject "test" and echoes back the "echo" query parameter from a HTTP request. The client uses the "NATS_URI" env variable to discover a NATS server.</p>
B.1.3	<p>Create the go-nats-ping application.</p> <pre>apc app create my-nats-ping --disable-routes --batch</pre> <p>NOTE: Because this app is not a web app, we <u>disable routing</u>.</p> <p>The source code is uploaded, the Go staging pipeline is auto-detected and launched, and build dependencies are added. Messages indicate "Staging is complete" and "Success!"</p>
B.1.4	<p>Explore the package that was created.</p> <pre>apc package list</pre> <p>You should see that the package is "ready". This is the runtime configuration for your app.</p>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

B.1.5	<p>Execute the following command to try deleting the package.</p> <pre>apc package delete my-nats-ping</pre> <p>The system throws an error. You cannot delete the package because it is used by the my-nats-ping app you created. To delete this package, you need to delete the job first. Apcera exerts controls over packages tied to jobs and prevents packages from being deleted before the app (job) is removed.</p> <p>NOTE: You will start your nats-ping app in the next lab. If you try to start it now it will fail because there is no server for the client app to connect to.</p>
--------------	--



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

Exercise C: Deploy Docker Workload (NATS Server)

In this exercise you create an app workload from a Docker image. To learn more about Apcera's support for Docker, see <http://docs.apcera.com/jobs/docker/>.

The Docker image is an instance of the gnatsd messaging server. The “gnatsd” acronym stands for Go NATS Daemon. To learn more about NATS, visit <http://nats.io/>.

Task 1: Create a Docker job

Step	Instruction
C.1.1	<p>Execute the following command to review syntax for creating Docker workloads.</p> <pre>apc help docker</pre> <p>NOTE: The syntax we will be using to create the gnatsd Docker workload is as follows:</p> <pre>apc docker run <app-name> --image <docker-image></pre>
C.1.2	<p>Execute the following command to create a Docker job.</p> <pre>apc docker run my-gnatsd --image apcera/gnatsd</pre> <p>Apcera looks up the Docker image at the public Docker repository, then downloads and stages it. You then see that the server is starting and the message “gnatsd is ready.” The APC client is retrieving the Docker image from the public repository, so staging may take a couple of minutes to complete.</p> <pre>[my-gnatsd] -- Pulling Docker image -- checking policy [my-gnatsd] -- Pulling Docker image -- checking if package FQN is taken [my-gnatsd] -- Pulling Docker image -- fetching image metadata [my-gnatsd] -- Pulling Docker image -- creating package [my-gnatsd] -- Pulling Docker image -- all layers downloaded [my-gnatsd] -- Creating job [my-gnatsd] -- Configuring job -- tagging package [my-gnatsd] -- Starting job [stderr] [5] 2016/02/10 19:51:28.012561 [INF] Starting gnatsd version 0.7.2 [stderr] [5] 2016/02/10 19:51:28.012648 [INF] Starting http monitor on 0.0.0.0:8222 [stderr] [5] 2016/02/10 19:51:28.012877 [INF] Listening for route connections on :6222 [stderr] [5] 2016/02/10 19:51:28.012965 [INF] Listening for client connections on 0.0.0.0:4222 [stderr] [5] 2016/02/10 19:51:28.013018 [INF] gnatsd is ready</pre>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

C.1.3

Execute the following command to verify that the gnatsd Docker image is running in Apcera.

apc app list

You should see that you have a single instance of the my-gnatsd server up and running:

Name	Namespace	Status	Instances	Routes
my-gnatsd	/sandbox/continuumtraining2015	started	1/1	http://mynodetodoapp.continuumtraining2015.skanderna.io
my-nats-ping	/sandbox/continuumtraining2015	started	0/1	
my-node-todo-app	/sandbox/continuumtraining2015	started	1/1	

Task 2: Tail the gnatsd server logs

Step	Instruction
C.2.1	<p>Execute the following command to tail the gnatsd server logs using APC.</p> <pre>apc app logs my-gnatsd</pre> <p>You should see that “gnatsd is ready” and listening for client connections on port 4222:</p> <pre>\$ apc app logs my-gnatsd</pre> <pre>[stderr] [5] 2016/04/04 20:34:09.418223 [INF] Starting gnatsd version 0.7.2</pre> <pre>[stderr] [5] 2016/04/04 20:34:09.418299 [INF] Starting http monitor on 0.0.0.0:8222</pre> <pre>[stderr] [5] 2016/04/04 20:34:09.418492 [INF] Listening for route connections on :6222</pre> <pre>[stderr] [5] 2016/04/04 20:34:09.418577 [INF] Listening for client connections on 0.0.0.0:4222</pre> <pre>[stderr] [5] 2016/04/04 20:34:09.419269 [INF] gnatsd is ready</pre> <p>NOTE: Exit log tail view by performing Ctrl + C in the command prompt session.</p>
C.2.2	Log in to the web console at the following URL: https://console.kiso.io .
C.2.3	Under JOBS , select Apps .



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

C.2.4	<p>Select the Docker job, my-gnatsd.</p> <p>At the SUMMARY tab, you should have 1 instance running.</p>  <p>The screenshot shows the 'my-gnatsd' job summary page. The 'Running Instances' field displays '1', which is pointed to by a red arrow. Other fields include 'Job::/sandbox/yoko::my-gnatsd', 'Created By: yoko', 'Allow SSH: checked', 'Allow Egress: unchecked', 'Restart Mode: Never', and 'Expected Instances: 1'.</p>
C.2.5	<p>Click the LOGS tab. You should see that “gnatsd is ready” and listening for client connections on port 4222:</p>  <p>The screenshot shows the 'my-gnatsd' job logs page. The logs display the following messages:</p> <pre>[stderr] [5] 2016/04/04 20:34:09.419269 [INF] gnatsd is ready [stderr] [5] 2016/04/04 20:34:09.418577 [INF] Listening for client connections on 0.0.0.0:4222 [stderr] [5] 2016/04/04 20:34:09.418492 [INF] Listening for route connections on :6222 [stderr] [5] 2016/04/04 20:34:09.418299 [INF] Starting http monitor on 0.0.0.0:8222 [stderr] [5] 2016/04/04 20:34:09.418223 [INF] Starting gnatsd version 0.7.2</pre>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

Exercise D: Link jobs and explore dynamic binding

In this exercise you link the go-nats-ping job to the gnatsd server, both of which you created in the previous lab. In addition, you learn how to tail the logs to show disconnects and reconnects. This exercise showcases the dynamic binding capabilities of Apcera.

Task 1: Link the nats-ping client app to the gnatsd server

Step	Instruction																				
D.1.1	<p>Execute the following command to review the command syntax for creating job links.</p> <pre>apc help job link</pre> <p>NOTE: The syntax for linking jobs is as follows:</p> <pre>apc job link <source-job> --to <target-job> --name <link-name> --port <port on target-job></pre> <p>The link name is required. The port is optional if the target job has only one port exposed. In this case, since the gnatsd server has more than one port exposed, this argument is required.</p>																				
D.1.2	<p>Execute the following command and verify that the nats-ping app is <u>NOT</u> started and that the gnatsd server is running:</p> <pre>apc app list</pre> <p>Before creating a job link, you must stop the source job you are linking, which in this case is my-nats-ping. If the app is not stopped, run the <code>apc app stop my-nats-ping</code> command.</p> <table><tr><th>Name</th><th>Namespace</th><th>Status</th><th>Instances</th><th>Routes</th></tr><tr><td>my-gnatsd</td><td>/sandbox/yoko</td><td>started</td><td>1/1</td><td></td></tr><tr><td>my-nats-ping</td><td>/sandbox/yoko</td><td>ready</td><td></td><td></td></tr><tr><td>my-node-todo-app</td><td>/sandbox/yoko</td><td>started</td><td>1/1</td><td>http://my-node-todo-app-9hr1tt.kiso.io</td></tr></table>	Name	Namespace	Status	Instances	Routes	my-gnatsd	/sandbox/yoko	started	1/1		my-nats-ping	/sandbox/yoko	ready			my-node-todo-app	/sandbox/yoko	started	1/1	http://my-node-todo-app-9hr1tt.kiso.io
Name	Namespace	Status	Instances	Routes																	
my-gnatsd	/sandbox/yoko	started	1/1																		
my-nats-ping	/sandbox/yoko	ready																			
my-node-todo-app	/sandbox/yoko	started	1/1	http://my-node-todo-app-9hr1tt.kiso.io																	



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

D.1.3	<p>Execute the following command to link the nats-ping client to the gnatsd server.</p> <pre>apc job link my-nats-ping --to my-gnatsd --name nats --port 4222</pre> <p>You should see that the system successfully links (binds) the my-nats-ping job to the my-gnatsd job.</p> <p>NOTE: You must use the link name specified above (“nats”) because the <i>nats-ping.go</i> code expects the URI to be “NATS_URI”. To create the job link URI, the system concatenates the link name with the “URI” identifier.</p>										
D.1.4	<p>Execute the following command to view the details about my-nats-ping.</p> <pre>apc app show my-nats-ping</pre> <p>Under “Linked Jobs” you should see the job is linked to the gnatsd server:</p> <table><thead><tr><th colspan="2">Linked Jobs</th></tr></thead><tbody><tr><td>1. Name:</td><td>nats</td></tr><tr><td>Job:</td><td>job::/sandbox/yoko::my-gnatsd</td></tr><tr><td>Port:</td><td>4222</td></tr><tr><td>Environment:</td><td>B251723EC21B4EBC908C998178DB2932_URI NATS_URI</td></tr></tbody></table>	Linked Jobs		1. Name:	nats	Job:	job::/sandbox/yoko::my-gnatsd	Port:	4222	Environment:	B251723EC21B4EBC908C998178DB2932_URI NATS_URI
Linked Jobs											
1. Name:	nats										
Job:	job::/sandbox/yoko::my-gnatsd										
Port:	4222										
Environment:	B251723EC21B4EBC908C998178DB2932_URI NATS_URI										
D.1.5	<p>Execute the following command to start the app:</p> <pre>apc app start my-nats-ping</pre>										



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

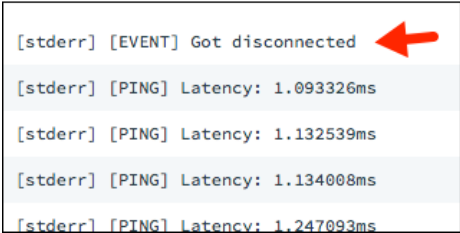
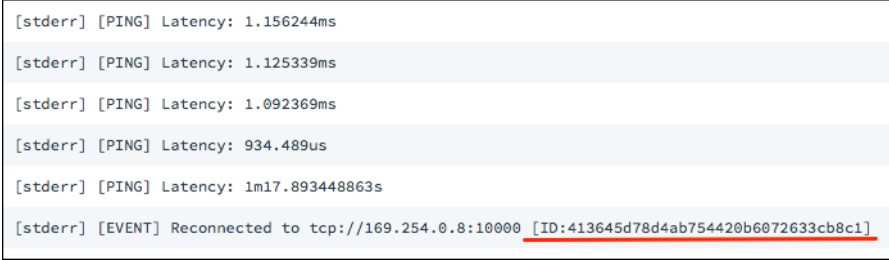
Task 2: Tail the logs

Step	Instruction
D.2.1	<p>Execute the following command to tail the server logs.</p> <pre>apc app logs my-gnatsd</pre> <p>You should see that the server is ready and listening for connections on port 4222.</p> <div>Success! Lorenzos-MacBook-Pro:nats-ping lparis\$ apc app logs my-gnatsd [stderr] 2015/01/16 05:12:19 ["Starting gnatsd version 0.5.6"] [stderr] 2015/01/16 05:12:19 ["Starting http monitor on port 8333"] [stderr] 2015/01/16 05:12:19 ["Listening for client connections on 0.0.0.0:4222"] [stderr] 2015/01/16 05:12:19 ["gnatsd is ready"]</div> <p>NOTE: To stop tailing the logs, use Ctrl + C.</p>
D.2.2	Log in to the Web Console at https://console.kiso.io .
D.2.3	<p>Select Apps, and locate your my-nats-ping job and select it so that you view its SUMMARY.</p> <p>Verify that 1 instance of the job is running.</p>
D.2.4	<p>Click the LOGS tab to show the message log.</p> <p>You should see that client connects to the server and sends ping requests. The job instance ID is listed.</p> <div>[stderr] [PING] Latency: 1.148243ms [stderr] [PING] Latency: 1.142079ms [stderr] [PING] Latency: 1.127647ms [stderr] [PING] Latency: 1.221361ms [stderr] [PING] Latency: 1.118631ms [stderr] [PING] Latency: 1.273243ms [stderr] [INFO] Delay is 2s [stderr] [EVENT] Connected to tcp://169.254.0.8:10000 [ID:70fda132aa5ede328d5eb345b625175f]</div>
D.2.5	Return to the terminal, and if you are still tailing the server log, use Ctrl + C .



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

D.2.6	<p>Execute the following command to stop the gnatsd server.</p> <pre>apc app stop my-gnatsd</pre>
D.2.7	<p>In the web console, verify in the LOGS that the client can't connect.</p>  <p>The screenshot shows a log entry '[stderr] [EVENT] Got disconnected' with a red arrow pointing to it. Below it are several '[stderr] [PING] Latency: ...' entries.</p>
D.2.8	<p>Execute the following command to restart the my-gnatsd server.</p> <pre>apc app start my-gnatsd</pre>
D.2.9	<p>Verify that the client dynamically reconnects.</p> <p>In the client tail logs, because you stopped the server instance, you should see that the my-nats-ping client app gets disconnected. Then, after you start another instance of the server, the client is able to “reconnect” to the new server instance. As far as the client knows, it is connecting to the same server at the same IP. However, behind the scenes it is an entirely new server instance with a different IP that the client is connecting to. You can verify this by checking the different IDs associated with each server instance. This shows Apcera’s dynamic binding capabilities for TCP endpoints.</p>  <p>The screenshot shows a series of '[stderr] [PING] Latency: ...' entries. The final entry is '[stderr] [EVENT] Reconnected to tcp://169.254.0.8:10000 [ID:413645d78d4ab754420b6072633cb8c1]' with the ID underlined in red.</p>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

Exercise E: Create and Snapshot a Capsule

In this exercise you create a capsule job, update it and take a snapshot. A capsule is a Apcera job that provides a virtual server host with a bare operating system (OS). You can use capsules to deploy services and other binary components to Apcera.

Before proceeding with this exercise, it may be helpful to review the documentation “Working with Capsules” at the following URL: <http://docs.apcera.com/tutorials/capsules/>.

Task 1: Create a capsule job

Step	Instruction
E.1.1	<p>Execute the following command to explore the syntax for creating capsules.</p> <pre>apc help capsule create</pre> <p>Note the syntax: <code>apc capsule create <capsule-name> [optional-args]</code></p>
E.1.2	<p>Execute the following command to view the available OS packages for capsules.</p> <pre>apc package list --type os --namespace /</pre> <p>Apcera provides various Linux OS distributions for creating capsules, including Ubuntu version 14.04 which is the OS we will be using for our capsule.</p>
E.1.3	<p>Execute the following command to create a Linux capsule.</p> <pre>apc capsule create my-capsule --image linux</pre> <p>Expected results: You should see that Apcera uses the “ubuntu-14.04” package to create the my-capsule job, and starts the capsule node, which is a virtual server in the cluster.</p>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

Task 2: Connect to the capsule

Step	Instruction
E.2.1	<p>Connect to the capsule using SSH.</p> <pre>apc capsule connect my-capsule</pre> <p>Expected result: <code>root@ip-<capsule-IP-address>:/root#</code></p> <p>This command connects you to the capsule OS via SSH as the root user of the isolated container, <u>not</u> the kernel root.</p>
E.2.2	<p>Execute the following command to update the list of OS packages to be installed on the capsule.</p> <pre>apt-get update</pre> <p>Expected result: Hangs at 0% update.</p> <p>By running this command, you might expect to connect to the http://archive.ubuntu.com server and download the latest packages from the repository. However, the update fails: it hangs at “0%” and cannot connect to the server. Why?</p> <p>By default, any job, including an app, Docker image, or OS capsule, runs in an isolated container without network access. To update the capsule OS, you need to grant egress to the container.</p>
E.2.3	<p>Exit the non-performing “<code>apt-get update</code>” request by pressing Ctrl + C in the command prompt session. This cancels the request and returns you to <code>root@ip-<address>:/root#</code>.</p> <p>To exit the SSH session in the capsule:</p> <pre>exit</pre>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

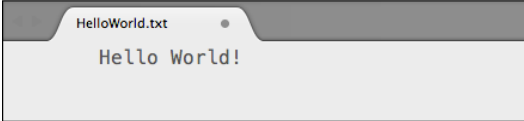
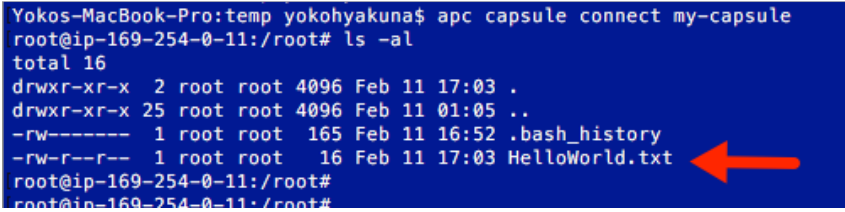
E.2.4	<p>Execute the following command to allow egress:</p> <pre>apc capsule update my-capsule --allow-egress</pre> <p>Note: This update requires the capsule job to be restarted.</p>
E.2.5	<p>Connect to the capsule again and run <code>apt-get update</code>.</p> <pre>\$ apc capsule connect my-capsule</pre> <pre># apt-get update</pre> <p>This time, the command should download the available updates successfully. You should see “<i>Reading package lists... Done</i>” message when it completes.</p>
E.2.6	<p>Exit the SSH session:</p> <pre>exit</pre>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

Task 4: Copy files to and from the capsule

Step	Instruction
E.3.1	Create a /temp folder if you don't have one already on your local machine , and change the working directory to /temp. <pre>mkdir temp cd temp</pre>
E.3.2	In the /temp directory, create a file named, HelloWorld.txt. 
E.3.3	Execute the following command to copy the HelloWorld.txt file to your capsule. <pre>apc capsule filecopy my-capsule HelloWorld.txt</pre>
E.3.4	Connect to my-capsule and ensure that HelloWorld.txt was copied. <pre>apc capsule connect my-capsule</pre> 
E.3.5	Re-name the file as HiWorld.txt and save it under /usr directory. <pre>mv HelloWorld.txt /usr/HiWorld.txt</pre>
E.3.6	Exit the SSH session: <pre>exit</pre>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

E.3.7	<p>Execute the following command to copy <code>HiWorld.txt</code> from <code>my-capsule</code> to your local machine.</p> <pre>apc capsule filecopy my-capsule HiWorld.txt -r /usr/HiWorld.txt -dl</pre> <p>NOTE: The <code>-dl</code> option is a shorthand for <code>--download</code> flag.</p> <p>You should see the <code>HiWorld.txt</code> in your local machine.</p>
--------------	--

Task 4: Take a snapshot of the capsule

Step	Instruction
E.4.1	<p>Execute the following command to snapshot the capsule.</p> <pre>apc capsule snapshot my-capsule --name snapshot01</pre> <p>You should see that Apcera retrieves the current state of the capsule, creates the snapshot, and creates a package for the snapshot.</p> <p>NOTE: If you execute the command without the <code>--name</code> parameter, the created snapshot would have a name with a UUID appended to the capsule name (e.g. <code>snapshot-my-capsule-1443209904</code>).</p>
E.4.2	<p>Execute the following command to see the package created from the snapshot.</p> <pre>apc package list</pre> <p>You should see that the newly created snapshot is listed.</p>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

Task 5: Clean up

Step	Instruction
E.5.1	Since you each have a quota set on your namespace, delete the package that is no longer needed. <code>apc package delete snapshot01</code>
E.5.2	Delete the my-capsule since you won't be needing it in the next lab. <code>apc capsule delete my-capsule</code>
E.5.3	Un-link my-gnatsd and my-nats-ping jobs and then delete them. <code>apc job unlink my-nats-ping --from my-gnatsd -p 4222</code> <code>apc app delete my-gnatsd</code> <code>apc app delete my-nats-ping</code>



DEV101: Developing with Apcera Platform

Lab 1: Deploy Diverse Workloads

Exercise Review

In this lab, you experienced how Apcera supports the deployment of a diverse set of workloads. You deployed an app written in Node.js and one written in Go. You also deployed a Docker image. You created a capsule job and snapshotted it. You got comfortable using the APC CLI and the Apcera Web Console.

In the next lab you bind the Node web app to database services and test the app's functionality. You will also create a job link between the NATS client and server apps and explore the dynamic binding capabilities of Apcera.

End of the Lab



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

Lab Introduction

In this lab you connect your deployed Node.js app (my-node-todo-app) to two DB providers. You do this by creating a service for each provider and then bind (connect) your app to each service. You also learn how to bind your capsule to an existing service.

Lab Exercise

This lab comprises the following exercises:

- A. Create a PostgreSQL service and binding
- B. Create a MySQL service and binding
- C. Using Docker as DB service provider
- D. Bind capsule to existing services
- E. Create MySQL App from Package with No Delete Hook

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

- Registering Providers: <http://docs.apcera.com/services/providers/>.
- Working with Services: <http://docs.apcera.com/services/services/>.
- Using Docker as DB provider: [http://docs.apcera.com/services/examples/ - using-docker-as-db-provider](http://docs.apcera.com/services/examples/-using-docker-as-db-provider).
- Creating Capsules: <http://docs.apcera.com/jobs/capsules/>.

Lab Instruction

For each exercise, complete the steps.



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

Exercise A: Create PostgreSQL service and binding

In this exercise you create a PostgreSQL service and bind your node-todo app to it.

Apcera includes built in service gateways for MySQL and Postgres databases.

Task 1: Create a PostgreSQL database service

Step	Instruction																
A.1.1	<p>Execute the following command to list available providers.</p> <pre>apc provider list --namespace /</pre> <p>You should see system-defined providers in the /apcera namespace, including MySQL and PostgreSQL DBs. We will be creating services for both providers and binding to them.</p> <table><tr><th>Name</th><th>Type</th><th>Namespace</th><th>Description</th></tr><tr><td>mysql</td><td>mysql</td><td>/apcera/providers</td><td>MySQL (RDS-backed)</td></tr><tr><td>nfs-sandbox</td><td>nfs</td><td>/apcera/providers</td><td>EBS-backed toy NFS</td></tr><tr><td>postgres</td><td>postgres</td><td>/apcera/providers</td><td>PostgreSQL (RDS-backed)</td></tr></table>	Name	Type	Namespace	Description	mysql	mysql	/apcera/providers	MySQL (RDS-backed)	nfs-sandbox	nfs	/apcera/providers	EBS-backed toy NFS	postgres	postgres	/apcera/providers	PostgreSQL (RDS-backed)
Name	Type	Namespace	Description														
mysql	mysql	/apcera/providers	MySQL (RDS-backed)														
nfs-sandbox	nfs	/apcera/providers	EBS-backed toy NFS														
postgres	postgres	/apcera/providers	PostgreSQL (RDS-backed)														
A.1.2	<p>Execute the following command to create a Postgres DB service and bind your app to it.</p> <pre>apc service create my-postgres-service --provider /apcera/providers::postgres --job my-node-todo-app</pre> <p>Where “my-postgres-service” is a user-defined name for the service, and “my-node-todo-app” is the name of the app that you created previously.</p>																
A.1.3	<p>Review the Service Creation Settings and press enter (Y) to create the service and binding.</p> <p>Press enter (Y) to confirm that the system can stop the job to create the service binding.</p> <p>You should see that the service is created, the job (your app) is stopped, the binding is created, and the job is restarted.</p>																



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

A.1.4

Execute the following command to view available services.

```
apc service list
```

Name	Type	Namespace	Provider
my-postgres-service	postgres	/sandbox/yoko	provider::apcera/providers::postgres

A.1.5

Execute the following command to view existing jobs.

```
apc job list
```

Name	Type	Namespace	Status	Instances
my-node-todo-app	app	/sandbox/yoko	started	1/1
my-node-todo-app/postgres/8f188b2a	job	/sandbox/yoko	started	1/1

You should see the job binding between the app and the service. In the picture, the binding name is **my-node-todo-app/postgres/UUID**.

A.1.6

Execute the following command to view the details about my-node-todo-app.

```
apc app show my-node-todo-app
```

In the “Bound Services” section of the command output, you should see service, provider, and environment variables for connecting to the service.

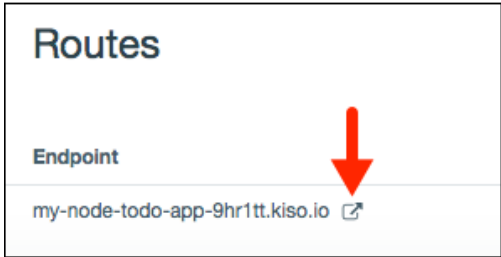
Bound Services	
1. Service:	service::/sandbox/yoko::my-postgres-service
Provider:	provider::apcera/providers::postgres
Environment:	JDBC_POSTGRES_URI MYPOSTGRESERVICE_URI POSTGRES_URI



DEV101: Developing with Apcera Platform

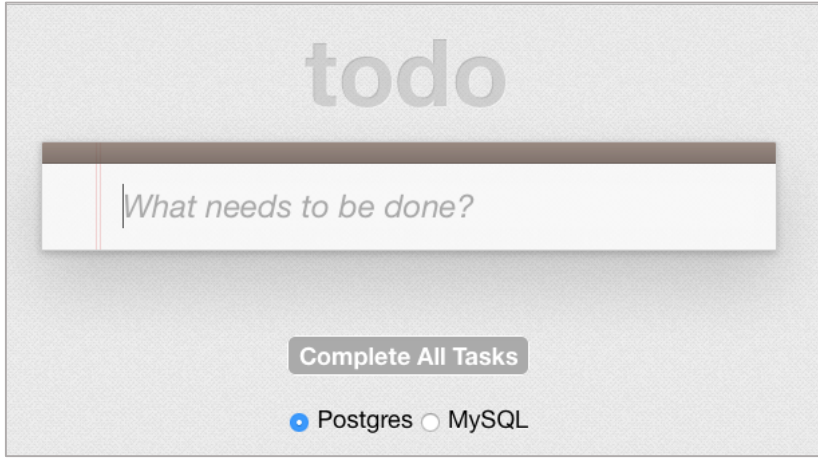
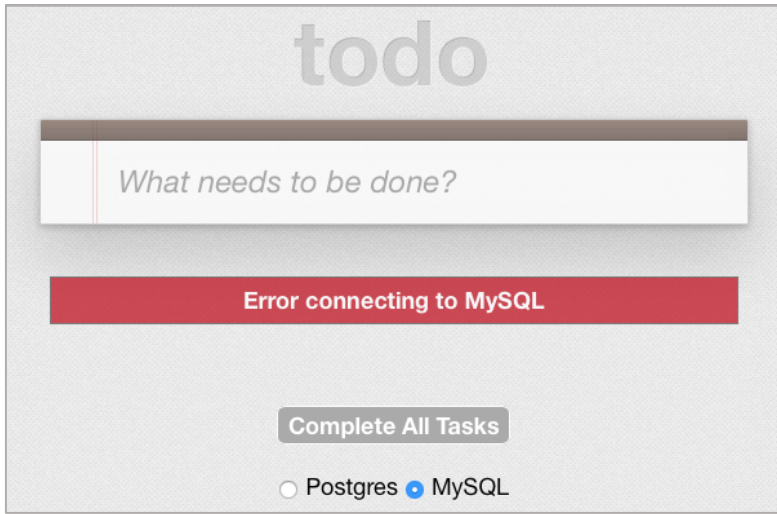
Lab 2: Orchestrate Services

Task 2: Test and verify the binding between your app and the PostgreSQL DB

Step	Instruction
A.2.1	<p>If your my-node-todo-app has NOT been started, execute the following command to start your my-node-todo-app.</p> <pre>apc app start my-node-todo-app</pre> <p>Expected Result: You should see a message with the route URL to access your my-node-todo-app.</p> <pre>Waiting for the job to start... [stdout] listening on port 4000 Job should be accessible at "http://my-node-todo-app-1rb5ks.kiso.io" Success!</pre>
A.2.2	Return to the Web Console (https://console.kiso.io).
A.2.3	From Apps , select my-node-todo-app .
A.2.4	<p>Click the NETWORKING tab, and scroll all the way down to Routes. Click on the icon to open the URI.</p> 
A.2.5	Ensure that your app can connect to the PostgreSQL DB by selecting the "Postgres" option in the web interface, entering text in the "What needs to be done?" field and pressing enter.

DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

	 <p>You should not receive an error, which indicates that your app is successfully connected to the PostgreSQL DB.</p>
A.2.6	<p>Select the “MySQL” option. You should receive the following error message: “Error connecting to MySQL.” This is because your app is not yet bound (connected) to the MySQL DB provider.</p> 



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

Exercise B: Create MySQL services and bindings

In this exercise you create a MySQL DB service and bind your node-todo app to it. To do this you have two options:

- Use an external MySQL provider, in which case the exercise is very similar to the previous one, the only difference being the DB provider. This demonstrates the consistency of service creation and binding in Apcera.
- Create an internal MySQL provider using the 'app from package' process.

Task 1: Create an external MySQL service

Step	Instruction												
B.1.1	<p>Repeat the process in Exercise A - Task 1 to create a service for the MySQL DB provider and bind your app to it.</p> <p>Use the following values.</p> <ul style="list-style-type: none">• Service name: <code>my-mysql-service</code>• Provider FQN: <code>/apcera/providers::mysql</code>• Job to bind: <code>my-node-todo-app</code>												
B.1.2	<p>Execute the following command to verify that the service gateway job and service were created.</p> <pre>apc service list</pre> <p>You should that see my-mysql-service that you created is listed.</p> <table><tr><th>Name</th><th>Type</th><th>Namespace</th><th>Provider</th></tr><tr><td>my-mysql-service</td><td>mysql</td><td>/sandbox/yoko</td><td>provider::apcera/providers::mysql</td></tr><tr><td>my-postgres-service</td><td>postgres</td><td>/sandbox/yoko</td><td>provider::apcera/providers::postgres</td></tr></table>	Name	Type	Namespace	Provider	my-mysql-service	mysql	/sandbox/yoko	provider::apcera/providers::mysql	my-postgres-service	postgres	/sandbox/yoko	provider::apcera/providers::postgres
Name	Type	Namespace	Provider										
my-mysql-service	mysql	/sandbox/yoko	provider::apcera/providers::mysql										
my-postgres-service	postgres	/sandbox/yoko	provider::apcera/providers::postgres										



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

B.1.3

Execute the following command to view existing jobs.

apc job list

Name	Type	Namespace	Status	Instances
my-node-todo-app	app	/sandbox/yoko	started	1/1
my-node-todo-app/mysql/680536ca	job	/sandbox/yoko	started	1/1
my-node-todo-app/postgres/8f188b2a	job	/sandbox/yoko	started	1/1

You should see both the job-to-service binding “my-node-todo-app/mysql/<UUID>”.

Task 2: Test and verify the binding between your app and the MySQL DB

Step	Instruction
B.2.1	Return to the web browser connected to my-node-todo-app: <a href="http://my-node-todo-app-<6-chars>.kiso.io">http://my-node-todo-app-<6-chars>.kiso.io
B.2.2	Refresh the page and select the MySQL option. This time you should not receive an error, indicating that the app is connecting to the MySQL DB via the service you created.
B.2.3	Enter some text in the “What needs to be done?” field and press enter. You should not receive an error, which indicates that your app is connected to the MySQL DB and writing the data.



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

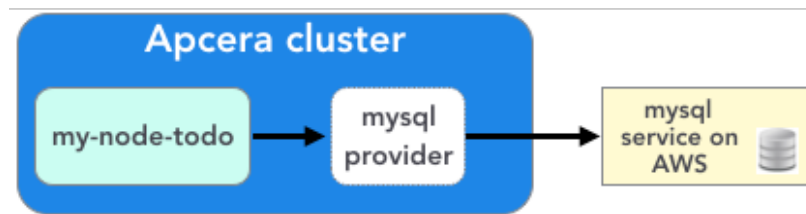
B.2.4	<p>Click the Postgres button. You should not receive an error. Your app is successfully connecting to both DB providers using the service definitions you created.</p> <div data-bbox="349 464 1114 1035"></div>
-------	--

DEV101: Developing with Apcera Platform

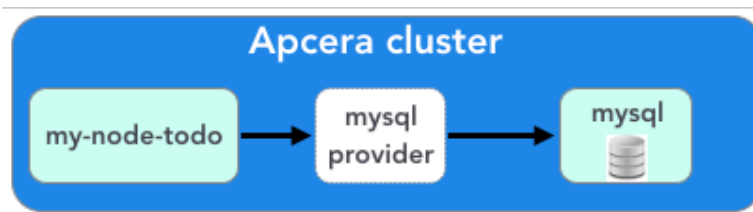
Lab 2: Orchestrate Services

Exercise C: Using Docker as DB service provider

In Exercise B, you created a MySQL database service using the system-level service provider (/apcera/providers:mysql). This mysql provider was configured to a MySQL server running on AWS. Connecting to a highly available external database is the typical architecture that you want for your production environment.



However, the developers may want to have an internal database that they can spin up to test their app against it.



In this exercise, you are going to learn how to create a MySQL database provider from a Docker image. First, you run a MySQL server Docker image in your cluster. Then you are going to register this Docker image as a service provider. Since the Docker image is running in the same cluster, it is internal to your Node.js application.

Task 1: Create an internal MySQL provider and register

Step	Instruction
C.1.1	Execute the following command to ensure that my-node-todo-app have been started. <pre>apc app list</pre>



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

C.1.2	<p>Execute the following command to run a Docker image running a MySQL server.</p> <pre>apc docker run docker-mysql --image /tutum/mysql --ignore-volumes</pre>
C.1.3	<p>Connect to the Docker image.</p> <pre>apc docker connect docker-mysql</pre>
C.1.4	<p>Once you SSH into the docker-mysql, execute the following command:</p> <pre>sed -i "s/bind-address\t\t= 127.0.0.1/bind-address = 0.0.0.0/1" /etc/mysql/my.cnf && service mysql restart && mysql -u root -e "GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'password' WITH GRANT OPTION; FLUSH PRIVILEGES;" && service mysql restart && exit</pre> <p>NOTE: Ignore the error message.</p>
C.1.5	<p>Explore the apc provider register command.</p> <pre>apc provider register -h</pre>
C.1.6	<p>Execute the following command to register the Docker job as a provider.</p> <pre>apc provider register docker-mysql-provider --job docker- mysql -u mysql://root:password@docker-mysql</pre>
C.1.7	<p>Execute the following command to view the detail.</p> <pre>apc provider show docker-mysql-provider</pre> <p>The provider is created in your namespace, /sandbox/<user-name> and type is set to mysql.</p>



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

Task 2: Create a service to test the provider

Step	Instruction
C.2.1	<p>Execute the following command to create a new service using the docker-mysql-provider.</p> <pre>apc service create docker-db --provider docker-mysql-provider</pre> <p>NOTE: Since the docker-mysql-provider was created in your namespace, there is no need to provide the FQN.</p>
C.2.2	<p>Now, you want to bind your docker-db service to my-node-todo-app so that you can test it.</p> <p>Since my-node-todo-app is already bound to my-mysql-service, you need to unbind it first.</p> <p>Execute the following command to unbind my-node-todo-app from my-mysql-service.</p> <pre>apc service unbind my-mysql-service --job my-node-todo-app</pre> <p>NOTE: Enter <i>y</i> to confirm the unbind request.</p>
C.2.3	<p>Execute the following command to bind the docker-db service to my-node-todo-app.</p> <pre>apc service bind docker-db --job my-node-todo-app</pre>
C.2.4	<p>Using a browser, navigate to the URL for your app: http://my-node-todo-app-6-chars.sandbox.kiso.io</p>
C.2.5	<p>Refresh the page and select the MySQL option.</p>
C.2.6	<p>Enter some text in the “<i>What needs to be done?</i>” field and press enter.</p> <p>Expected result: You should not receive an error, which indicates that your app is successfully connected to the MySQL database.</p>



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

Exercise D: Bind capsule to existing services

In this exercise you bind your capsule to services using the 'apc service bind' command.

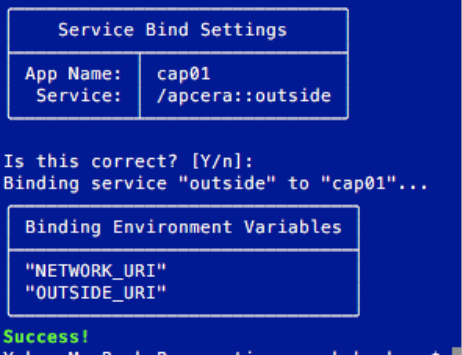
Task 1: Allow network egress to your capsule

Step	Instruction																
D.1.1	<p>Execute the following command to list available services.</p> <pre>apc service list</pre> <p>You should see that the service “my-postgres-service” exists because you created it in the previous exercise. If you do not have this service, go to the previous exercise and create it.</p> <table><tr><th>Name</th><th>Type</th><th>Namespace</th><th>Provider</th></tr><tr><td>docker-db</td><td>mysql</td><td>/sandbox/yoko</td><td>docker-mysql-provider</td></tr><tr><td>my-mysql-service</td><td>mysql</td><td>/sandbox/yoko</td><td>provider::apcera/providers::mysql</td></tr><tr><td>my-postgres-service</td><td>postgres</td><td>/sandbox/yoko</td><td>provider::apcera/providers::postgres</td></tr></table>	Name	Type	Namespace	Provider	docker-db	mysql	/sandbox/yoko	docker-mysql-provider	my-mysql-service	mysql	/sandbox/yoko	provider::apcera/providers::mysql	my-postgres-service	postgres	/sandbox/yoko	provider::apcera/providers::postgres
Name	Type	Namespace	Provider														
docker-db	mysql	/sandbox/yoko	docker-mysql-provider														
my-mysql-service	mysql	/sandbox/yoko	provider::apcera/providers::mysql														
my-postgres-service	postgres	/sandbox/yoko	provider::apcera/providers::postgres														
D.1.2	<p>Execute the <code>apc service list</code> for a root namespace to view the entire list.</p> <pre>apc service list -ns /</pre> <p>You should see that the ‘outside’ network service is available.</p>																
D.1.3	<p>Create a capsule named, cap01 with Linux image.</p> <pre>apc capsule create cap01 --image linux</pre>																



DEV101: Developing with Apcera Platform


Lab 2: Orchestrate Services

D.1.4	<p>Execute the following command to grant network egress to the capsule.</p> <pre>apc service bind /apcera::outside --job cap01</pre> <p>You should see that the capsule is successfully bound to the external network. The external network service is available by default in the system. You have policy permission to bind to system services, and you authored the policy to allow job binding.</p>  <p>The command binds the capsule job to the external network via a service gateway that is provided by default with a cluster. The <code>--allow-egress</code> (or <code>-ae</code>) flag is a shortcut for the “service bind” command. Using this option requires admin-level policy privileges and should not be used for production apps. When you snapshot a capsule, <code>--allow-egress</code> is disabled. Refer to the documentation for details: http://docs.apcera.com/jobs/capsules/#allowing-capsule-egress.</p>
D.1.5	<p>Execute the following command to verify that your capsule is bound to the external network service.</p> <pre>apc job show cap01</pre> <p>The job should be bound to ‘service://apcera::outside’.</p> <p>NOTE: Unlike an app, which is both an app and a job in Apcera, a capsule is not an app; it is only a job. Command <code>apc app list</code> does not show capsules.</p>



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

D.1.6	<p>Execute the following command to bind the capsule to the PostgreSQL DB using the existing service.</p> <pre>apc service bind my-postgres-service --job cap01</pre> <p>Review the Service Bind Settings and press enter (Y) to create the servicing binding.</p> <p>Press enter (Y) to confirm that the system can stop the job to create the service binding.</p> <p>The job is stopped, the binding is created, and the job is restarted. You should also see the Binding Environment Variables, one of which you will use to log in to the DB from the capsule.</p>
D.1.7	<p>To view the Binding Environment Variables for a job, issue the following command:</p> <pre>apc job show cap01</pre>  <pre>Bound Services 1. Service: service:/apcera:outside Environment: NETWORK_URI OUTSIDE_URI 2. Service: service:/sandbox/yoko:my-postgres-service Provider: provider:/apcera/providers:postgres Environment: EE2BEC11F0964D8590416B4DCEAC26AE_URI JDBC_POSTGRES_URI MYPOSTGRESSERVICE_URI POSTGRES_URI</pre> <p>This command returns details about the cap01 job, including the Environment variables for the Service Provider (item 2 under “Bound Services” and the end of the return). You can use any of the Environment URIs to connect to the PostgreSQL DB from your cap01 job, such as “POSTGRES_URI”.</p>



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

Task 2: Connect to your capsule

Step	Instruction
D.2.1	<p>Execute the following command to connect to the capsule and update the list of OS packages to be installed.</p> <pre>apc capsule connect cap01</pre> <p>Expected result: root@ip-<capsule-IP-address>:/root#</p>
D.2.2	<p>Execute the following command to install the OS updates if there is any.</p> <pre>apt-get update</pre> <p>The Ubuntu 14.04 OS package list update should now succeed.</p> <p>NOTE: This time the update succeed because you have connected to the outside service. Notice how this is controlled both at the policy level through job and service binding resources, and at the app configuration level. Meaning you must explicitly connect to the network, and you must have permissions to do so.</p>
D.2.3	<p>While still logged into the capsule OS, install the Postgres 9.3 client onto the capsule:</p> <pre>apt-get install postgresql-client-9.3</pre> <p>Now that you have created the binding between the cap01 job and the Postgres service, you verify the connection by logging in to the PostgreSQL DB from the capsule.</p>
D.2.4	<p>To connect, you use ephemeral credentials in the form of a Binding Environment Variable (any string with "URI" as a suffix), for example:</p> <pre>psql \$POSTGRES_URI</pre> <p>Expected result: You should be able to log in.</p>



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

D.2.5	<p>Enter <code>\dd</code> and <code>\dt</code> to view the database schema and tables:</p> <pre>root@ip-169-254-0-19:/root# psql \$POSTGRES_URI psql (9.3.7, server 9.3.3) Type "help" for help. 4c63ec076a6148508f791a3d529ca20a=> \dd Object descriptions Schema Name Object Description -----+-----+-----+----- (0 rows) 4c63ec076a6148508f791a3d529ca20a=> \dt List of relations Schema Name Type Owner -----+-----+-----+----- public tasks table 051noQbbsuyX1Arx (1 row)</pre>
D.2.6	Issue command <code>\q</code> to quit the PostgreSQL client.
D.2.7	Issue command <code>exit</code> to return to APC.
D.2.8	Issue command <code>apc help</code> to verify that you returned to your APC session.

Task 3: Take a snapshot of the capsule

Step	Instruction
D.3.1	<p>Execute the following command to snapshot the capsule and save its state:</p> <pre>apc capsule snapshot cap01</pre> <p>The state of the capsule is successfully saved as a package. Run command <code>apc package list</code> and you should see the 'snapshot-cap01-<UUID>' package as well as the previous snapshot you created.</p>
D.3.2	<p>Since you are limited by policy to the amount of resources you can have in your namespace, delete the snapshot packages.</p> <pre>apc package delete snapshot-cap01-<UUID></pre>
D.3.3	<p>Also delete the capsule.</p> <pre>apc capsule delete cap01</pre>



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

Exercise E: Create MySQL Server App with No Delete Hook

In this exercise you create a MySQL Server app with a no delete hook using the built-in MySQL semantic pipeline.

Task 1: Write an event rule app

Step	Instruction
E.1.1	Create a new directory named sp-hook-no-delete . <code>mkdir sp-hook-no-delete</code>
E.1.2	Change your working directory to sp-hook-no-delete directory. <code>cd sp-hook-no-delete</code>
E.1.3	Open a plain text editor such as Atom, and Sublime.
E.1.4	Enter the following: <pre>var express = require('express'); var app = express(); app.use(express.bodyParser()); app.post('/auth', function(req, res) { if (req.body.Command.match(/DROP/i) req.body.Command.match(/DELETE/i)) { // reject all drop and delete commands res.json({ Permitted: false, Reason: "No!" }); } else { // permit anything else res.json({ Permitted: true, Reason: "Move along" }); } }); // get the port from environment, or default to 4000 var port = process.env.PORT 4000 app.listen(port); // put a friendly message on the terminal console.log("Server running at http://127.0.0.1:"+port+"/");</pre>



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

E.1.5	Save it as app.js in the sp-hook-no-delete directory.
E.1.6	<p>Create another file named package.json and enter the following dependency declaration:</p> <pre>{ "name": "no-delete", "version": "0.0.1", "scripts": { "start": "node ./app.js" }, "dependencies": { "express": "3.x" } }</pre>

Task 2: Deploy your hook to the cluster

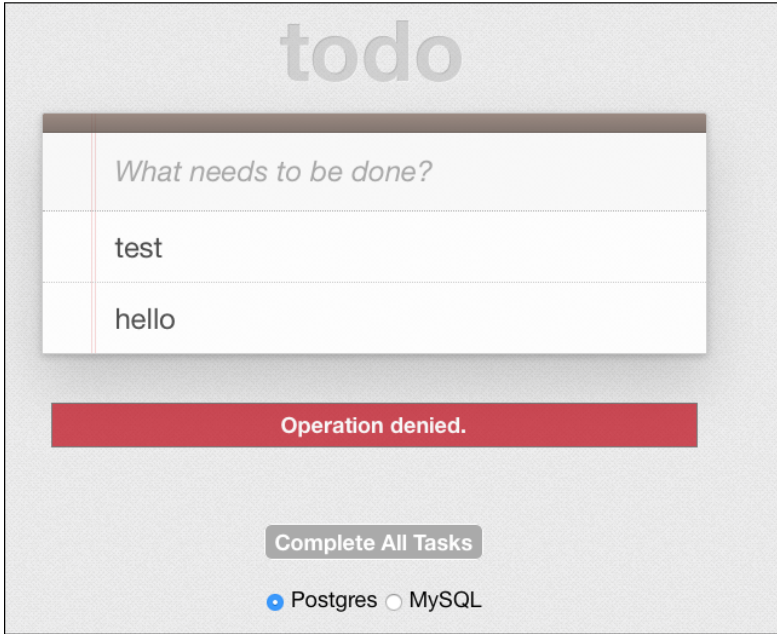
Step	Instruction
E.2.1	<p>Execute the following command to create a hook.</p> <pre>apc app create nodelete --start --batch</pre>
E.2.2	<p>Copy the route URI of the nodelete app from the output, http://nodelete-6chars.kiso.io:</p> <p>App should be accessible at "http://nodelete-m0xugk.kiso.io" Success!</p>
E.2.3	<p>Execute the following command to create an even rule to invoke the hook.</p> <p>NOTE: Replace the <nodelete-app-route-URI> with the URI you copied in E.2.2.</p> <pre>apc rule create denydelete -s my-postgres-service --type=hook --url <nodelete-app-route-URI>/auth</pre> <p>Example: <pre>apc rule create denydelete -s my-postgres-service --type=hook --url http://nodelete-m0xugk.kiso.io/auth</pre></p>



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

Task 3: Test the No Delete hook

Step	Instruction
E.3.1	<p>Navigate to the URL for your app: <a href="http://my-node-todo-app-<6chars>.sandbox.kiso.io">http://my-node-todo-app-<6chars>.sandbox.kiso.io</p> <p>NOTE: To get the route URL, issue command <code>apc app show my-node-todo-app</code>. The URL is listed in the Routes row. If my-node-todo-app is no longer running, follow the instructions in <i>Lab 1</i> to create the app.</p>
E.3.2	<p>Enter some text in the “What needs to be done?” field and press enter, and then click the Complete All Tasks button.</p> <p>Expected result: You should receive Operation denied error because of the no delete event rule (hook) that you created.</p> 
E.3.3	Log in to the Web Console at https://console.kiso.io .
E.3.4	Select All under JOBS .
E.3.5	Locate your my-node-todo-app job and select it so that you view its job details.



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

E.3.6	<p>Click the LOGS tab to see the log.</p> <pre>[stderr] PGdeleteRows error: { [error: admin policy hook denied action] [stdout] GET /deletePGRows</pre>
E.3.7	<p>Execute the following command to list existing jobs.</p> <pre>apc job list</pre> <p>Make a note of the job, my-node-todo-app/postgres/<UUID>.</p>
E.3.8	<p>The following command displays the logs that are related to the hooks:</p> <pre>apc job logs my-node-todo-app/postgres/<UUID> --no-tail grep hooks</pre> <p>You should see something similar to the following:</p> <pre>[stdout] [INFO 2016-02-17 21:32:31.667532068 +0000 UTC hostname='ip-169-254-0-51' category='sp/postgres:hooks' context='sp.15'] Hook [DELETE] denied by <http://nodelete- m0xugk.kiso.io/auth>: No! [stdout] [INFO 2016-02-17 21:32:31.667593660 +0000 UTC hostname='ip-169-254-0-51' category='sp/postgres:hooks' context='sp.15'] Query denied (by http://nodelete- m0xugk.kiso.io/auth) for command 1/1 [DELETE] (DELETE FROM tasks;) [stdout] [INFO 2016-02-17 21:32:31.667638550 +0000 UTC hostname='ip-169-254-0-51' category='sp/postgres:hooks' context='sp.15'] rejecting request with error frame (hook denied)</pre>



DEV101: Developing with Apcera Platform

Lab 2: Orchestrate Services

Exercise Review

In this lab you created services and service bindings that connected your Node.js app to two backend DBs. You also used 'apc service bind' to create a job binding that connected your capsule job to an existing service. Lastly, you created a job link between your go-nats-ping client app and the gnatsd server job and explored Apcera's dynamic binding capabilities.

End of the Lab



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

Lab Introduction

In this lab you explore how to extend Apcera and customize functionality. First you deploy an app with services using a manifest. Next you create a custom package. Lastly, you explore how to create an app from a package and use semantic pipelines to inject logic into your service connections.

Lab Exercise

This lab comprises the following exercises:

- A. Create App and Services Using Manifest
- B. Write and Build Custom Package
- C. Move workloads using Job Scheduling Tags
- D. Enabling broadcast and multicast routes

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

- Deploying Apps Using Manifests: <http://docs.apcera.com/jobs/manifests/>
- Working with Packages: <http://docs.apcera.com/packages/using/>
- Working with Semantic Pipelines: <http://docs.apcera.com/services/pipelines/>

Lab Instruction

For each exercise, complete the steps.



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

Exercise A: Create App and Services Using Manifest File

In this exercise you write a manifest file and use it to deploy an app in Apcera. Using a manifest is the preferred approach for deploying apps because it allows you to script and save repeatable deployment parameters for your apps.

A manifest is a plain text file named **continuum.conf** that you create and place in the root level of the working directory of your app. You use a manifest file in place of some or all of the flags you would otherwise specify on the command line when you create or deploy an app.

Task 1: Create a copy of demo-node-todo directory

Step	Instruction
A.1.1	<p>Execute the following command to explore the use and structure of a manifest file.</p> <pre>apc help manifest</pre> <p>This command returns the syntax for deploying an app using a manifest and an example manifest. Manifest files can be used with the <code>apc app create</code> and <code>apc app deploy</code> commands. Additional example manifests are provided in the Apcera documentation.</p>
A.1.2	<p>To avoid confusion between the web app you created manually and the one you are creating here using a manifest file, create a new directory named demo-node-todo-manifest under the sample-apps directory, and copy the contents of the demo-node-todo directory.</p> <pre>\$ cd sample-apps \$ mkdir demo-node-todo-manifest \$ cd demo-node-todo \$ cp -R * ../demo-node-todo-manifest</pre>
A.1.3	<p>Change your working directory to demo-node-todo-manifest that you created.</p> <pre>cd demo-node-todo-manifest</pre>
A.1.4	<p>Use <code>ls</code> to verify that the files were copied.</p>



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

Task 2: Create a manifest file for scripted app deployment

Step	Instruction
A.2.1	<p>Open a plain text editor such as Atom, and Sublime.</p> <p>Note: Do <u>not</u> create a manifest file using a common word processing application. Such programs often use non-standard characters (such as Smart Quotes) that can get included in the app or service names.</p>
A.2.2	<p>Enter the following:</p> <pre># App name is required and must be in quotes name: "my-node-app-manifest" # Create 2 instances instead of 1 (default) instances: 2 # Custom resource allocation resources { cpu: "200" disk_space: "768MB" memory: "256MB" network_bandwidth: "10Mbps" } # Create DB services and bind the app to them # Providers must be registered prior to app creation services [{ provider_name: "/apcera/providers::postgres", type: "postgres", service_name: "mydb1", }, { provider_name: "/apcera/providers::mysql", type: "mysql", service_name: "mydb2", }] # App startup timeout timeout: 10 start: true</pre>



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

A.2.3	Save the file as continuum.conf under demo-node-todo-manifest directory.
A.2.4	<p>From the demo-node-todo-manifest directory, execute the following command to create the app using the manifest file.</p> <pre>apc app create</pre> <p>All parameters for creating the app are read from the manifest file. The app is created as well as the services and bindings, and both app instances are started on deployment.</p> <p>Note that any parameters you provide on the command line take precedence over those parameters in the manifest file.</p>

Task 3: Verify the app creation

Step	Instruction
A.3.1	<p>Access the web interface for the app using the route URL, which is available from the app creation output.</p> <p>For example:</p> <pre>All instances started! App should be accessible at "http://my-node-app-manifest-gl3k312.kiso.io" Success!</pre> <p>You should see the web interface for your app.</p>
A.3.2	Select the databases. You should not receive an error because the manifest created the services and bound the app to them.
A.3.3	<p>Execute the following command to verify the successful deployment of your app.</p> <pre>apc app list</pre> <p>You should have 2 running instances of the app.</p>

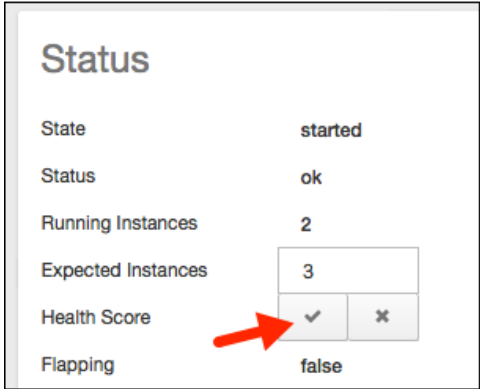


DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

A.3.4	<p>Execute the following command to verify that you have services: mydb1, and mydb2.</p> <pre>apc service list</pre>
-------	--

Task 4: Manage your app using the web console

Step	Instruction
A.4.1	Log in to the Apcera Web Console, http://console.kiso.io .
A.4.2	Select your namespace and search your jobs. You should see that the my-node-app-manifest is deployed in your namespace.
A.4.3	Select Apps under JOBS . You should see the my-node-app-manifest app and verify that 2 instances are running.
A.4.4	<p>Select the my-node-app-manifest name to launch its details view. In the Status, select started instances of 2 for the Expected Instances. The text box should allow you to modify the number of instances. Change the expected instances from 2 to 3 and click the checkmark button to save your changes.</p>  <p>The status changes to warning while another instance is being started. Momentarily, you should see the status changes to ok and the number of running instances for your app is now 3.</p>



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

A.4.5	<p>Execute the following command to verify the update using APC:</p> <pre>apc app health my-node-app-manifest</pre> <p>Expected result: The health score is 100%, and running instances shows 3/3.</p>
A.4.6	<p>Return to the web console and select your app. Click the Stop button to stop the app.</p> <p>Momentarily you should see that the job is stopped.</p>
A.4.7	<p>Click Delete button to delete the my-node-app-manifest since you no longer need this app in the subsequent labs.</p>



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

Exercise B: Package script walkthrough

In this exercise, you will review the Java 1.8 package script. However, you are not required to create the package using `apc package build` command, since downloading the JDK and build a package takes time.

Task 1: Create the Java 1.8 package script

Step	Instruction
B.1.1	Create a folder named java-1.8 on your local system.
B.1.2	Open a plain text editor such as Atom, and Sublime. Note: Do <u>not</u> create a package script using a common word processor. Such programs use non-standard characters (such as Smart Quotes) that can get included in the app or service names.
B.1.3	Enter the following (Note: Update <code>"/sandbox/<user-name>"</code> with your namespace.): name: "jdk-1.8" version: "1.8-u60" namespace: "/sandbox/<user-name>" depends [{ os: "linux" }] provides [{ runtime: "java" }, { runtime: "java-1.8" }, { runtime: "java-1.8.0" }, { runtime: "java-1.8.0-u60" }] environment { "JAVA_HOME": "/opt/apcera/java/jdk/1.8/", "PATH": "/opt/apcera/java/jdk/1.8/bin:\$PATH" } build (wget -q --header "Cookie: oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-pub/java/jdk/8u60-b26/jdk-8u60-linux-x64.tar.gz tar xzf jdk-8u60-linux-x64.tar.gz sudo mkdir -p /opt/apcera/java/jdk sudo cp -a jdk1.8.0_60 /opt/apcera/java/jdk/ sudo ln -s /opt/apcera/java/jdk/jdk1.8.0_60 /opt/apcera/java/jdk/1.8 sudo update-alternatives --install /usr/bin/java java /opt/apcera/java/jdk/1.8/bin/java 100 sudo update-alternatives --install /usr/bin/javac javac /opt/apcera/java/jdk/1.8/bin/javac 100)



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

	<p>The package script specifies the name (jdk-1.8) and namespace (/sandbox/<user_name>) for the package.</p> <p>In reality, you only need one jdk-1.8 package per cluster that the namespace will look something like, /apcera/pkg/runtimes rather than your sandbox namespace.</p> <p>The provides defines that this package will be used when a user specifies any of the following dependencies:</p> <ul style="list-style-type: none">• <code>apc app create javaApp --depends-on runtime.java</code>• <code>apc app create javaApp --depends-on runtime.java-1.8</code>• <code>apc app create javaApp --depends-on runtime.java-1.8.0</code>• <code>apc app create javaApp --depends-on runtime.java-1.8.0-u60</code> <p>The script also sets the JAVA_HOME and PATH environment variables.</p> <p>Finally, the JDK tarball is downloaded from Oracle, extracted, and then copied into the /opt/apcera/java/jdk directory, and then installed.</p>
B.1.4	Save the file as java-1.8.conf in the java-1.8 directory.

You are NOT going to build the java-1.8 package since it will take some time to build and upload the package. The compiled package size is about 65MB.

Review from the lecture:

The command syntax for building a package is:

```
apc package build <package-file-name>.conf
```

In order to build the java-1.8 package, you execute the following command:

```
apc package build java-1.8.conf
```

When the package is successfully built, you see the above message.

Staging is complete.

Created package "package::/sandbox/<user-name>::java-1.8"



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

Troubleshoot:

You need sufficient resources to build a package. If you don't have sufficient namespace or job quota (controlled by policy), the package build hangs ("Creating json file from manifest").

Analysis:

The system uses a special stager called the "compiler" to compile and stage the package for use by apps and jobs. You can specify a different stager using the '--staging' command.

The package name (*java-1.8.conf*) is a user-defined string that is unrelated to its provides. By default, the name of the package config file is used to name the package within the system. You can use the '--name' flag to specify a different name for the package when building it.

Good practice:

Name the package file similar to its provides, or use the '--name' flag to specify a self-descriptive name for the package within the system. Doing so makes it easier for developers to quickly identify what the package provides.



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

Exercise C: Move Workloads using Job Scheduling Tags

In this exercise you are going to move job instances from one IM to another using the job scheduling tags.

Task 1: Create a capsule

Step	Instruction																																																																																										
C.1.1	<p>Execute the following command to display the list of IMs on the kiso cluster.</p> <pre>apc cluster im list</pre> <table><tr><th>Name</th><th>UUID</th><th>Uptime</th><th>Instances</th><th>Mem (Res/Max)</th><th>Disk (Res/Max)</th><th>Net (Res/Max)</th><th>Tags</th><th>Datacenter</th></tr><tr><td>kiso-037ffac0</td><td>bb1042d5</td><td>28m</td><td>4</td><td>304/16886 MB</td><td>1792/131072 MB</td><td>35/1000 Mbps</td><td>aws im3 us-west-2a</td><td>defaultdc</td></tr><tr><td>kiso-03c4b4b3</td><td>2110815a</td><td>28m</td><td>3</td><td>288/16886 MB</td><td>1536/131072 MB</td><td>25/1000 Mbps</td><td>aws im8 us-west-2c</td><td>defaultdc</td></tr><tr><td>kiso-1bb9fa65</td><td>c58a45c9</td><td>28m</td><td>7</td><td>1552/16886 MB</td><td>5888/131072 MB</td><td>45/1000 Mbps</td><td>aws im2 us-west-2a</td><td>defaultdc</td></tr><tr><td>kiso-2f5536c3</td><td>07d40ac2</td><td>28m</td><td>6</td><td>400/16886 MB</td><td>2048/131072 MB</td><td>60/1000 Mbps</td><td>aws im7 us-west-2c</td><td>defaultdc</td></tr><tr><td>kiso-31a0bc97</td><td>1b700c95</td><td>29m</td><td>6</td><td>344/16886 MB</td><td>2304/131072 MB</td><td>55/1000 Mbps</td><td>aws im9 us-west-2c</td><td>defaultdc</td></tr><tr><td>kiso-4c8845ef</td><td>3fe3aace</td><td>29m</td><td>4</td><td>784/16886 MB</td><td>3072/131072 MB</td><td>30/1000 Mbps</td><td>aws im6 us-west-2b</td><td>defaultdc</td></tr><tr><td>kiso-53fbd4ac</td><td>aea34fcd</td><td>27m</td><td>2</td><td>272/16886 MB</td><td>1280/131072 MB</td><td>15/1000 Mbps</td><td>aws im5 us-west-2b</td><td>defaultdc</td></tr><tr><td>kiso-7c5e756c</td><td>c5aa5951</td><td>27m</td><td>4</td><td>776/16886 MB</td><td>3328/131072 MB</td><td>25/1000 Mbps</td><td>aws im4 us-west-2b</td><td>defaultdc</td></tr><tr><td>kiso-b16c73b1</td><td>367d81c5</td><td>29m</td><td>37</td><td>8016/16886 MB</td><td>32512/131072 MB</td><td>220/1000 Mbps</td><td>aws im1 us-west-2a</td><td>defaultdc</td></tr></table> <p>A table presents the list of IMs along with its UUID and tags.</p> <p>NOTE: IM tags were assigned by the operations who installed the Apcera cluster.</p>	Name	UUID	Uptime	Instances	Mem (Res/Max)	Disk (Res/Max)	Net (Res/Max)	Tags	Datacenter	kiso-037ffac0	bb1042d5	28m	4	304/16886 MB	1792/131072 MB	35/1000 Mbps	aws im3 us-west-2a	defaultdc	kiso-03c4b4b3	2110815a	28m	3	288/16886 MB	1536/131072 MB	25/1000 Mbps	aws im8 us-west-2c	defaultdc	kiso-1bb9fa65	c58a45c9	28m	7	1552/16886 MB	5888/131072 MB	45/1000 Mbps	aws im2 us-west-2a	defaultdc	kiso-2f5536c3	07d40ac2	28m	6	400/16886 MB	2048/131072 MB	60/1000 Mbps	aws im7 us-west-2c	defaultdc	kiso-31a0bc97	1b700c95	29m	6	344/16886 MB	2304/131072 MB	55/1000 Mbps	aws im9 us-west-2c	defaultdc	kiso-4c8845ef	3fe3aace	29m	4	784/16886 MB	3072/131072 MB	30/1000 Mbps	aws im6 us-west-2b	defaultdc	kiso-53fbd4ac	aea34fcd	27m	2	272/16886 MB	1280/131072 MB	15/1000 Mbps	aws im5 us-west-2b	defaultdc	kiso-7c5e756c	c5aa5951	27m	4	776/16886 MB	3328/131072 MB	25/1000 Mbps	aws im4 us-west-2b	defaultdc	kiso-b16c73b1	367d81c5	29m	37	8016/16886 MB	32512/131072 MB	220/1000 Mbps	aws im1 us-west-2a	defaultdc
Name	UUID	Uptime	Instances	Mem (Res/Max)	Disk (Res/Max)	Net (Res/Max)	Tags	Datacenter																																																																																			
kiso-037ffac0	bb1042d5	28m	4	304/16886 MB	1792/131072 MB	35/1000 Mbps	aws im3 us-west-2a	defaultdc																																																																																			
kiso-03c4b4b3	2110815a	28m	3	288/16886 MB	1536/131072 MB	25/1000 Mbps	aws im8 us-west-2c	defaultdc																																																																																			
kiso-1bb9fa65	c58a45c9	28m	7	1552/16886 MB	5888/131072 MB	45/1000 Mbps	aws im2 us-west-2a	defaultdc																																																																																			
kiso-2f5536c3	07d40ac2	28m	6	400/16886 MB	2048/131072 MB	60/1000 Mbps	aws im7 us-west-2c	defaultdc																																																																																			
kiso-31a0bc97	1b700c95	29m	6	344/16886 MB	2304/131072 MB	55/1000 Mbps	aws im9 us-west-2c	defaultdc																																																																																			
kiso-4c8845ef	3fe3aace	29m	4	784/16886 MB	3072/131072 MB	30/1000 Mbps	aws im6 us-west-2b	defaultdc																																																																																			
kiso-53fbd4ac	aea34fcd	27m	2	272/16886 MB	1280/131072 MB	15/1000 Mbps	aws im5 us-west-2b	defaultdc																																																																																			
kiso-7c5e756c	c5aa5951	27m	4	776/16886 MB	3328/131072 MB	25/1000 Mbps	aws im4 us-west-2b	defaultdc																																																																																			
kiso-b16c73b1	367d81c5	29m	37	8016/16886 MB	32512/131072 MB	220/1000 Mbps	aws im1 us-west-2a	defaultdc																																																																																			
C.1.2	<p>Execute the following command to create a Linux capsule.</p> <pre>apc capsule create cap01 --image linux</pre>																																																																																										
C.1.3	<p>Execute the following command to check where the instance is running.</p> <pre>apc job instances cap01</pre> <div><pre>Looking up "cap01"... done Health Score: 100% Running Instances: 1/1</pre><table><tr><th>UUID</th><th>Status</th><th>Uptime</th><th>Host</th></tr><tr><td>3360669e</td><td>RUNNING</td><td>16s</td><td>kiso-b16c73b1</td></tr></table></div> <p>The table displays the UUID of the cap01 instance, status, uptime, and host name of where the instance is running.</p>	UUID	Status	Uptime	Host	3360669e	RUNNING	16s	kiso-b16c73b1																																																																																		
UUID	Status	Uptime	Host																																																																																								
3360669e	RUNNING	16s	kiso-b16c73b1																																																																																								



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

Task 2: Move your workload via APC

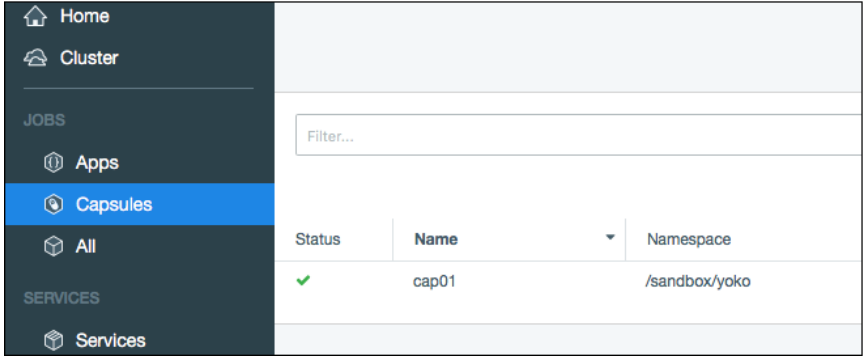
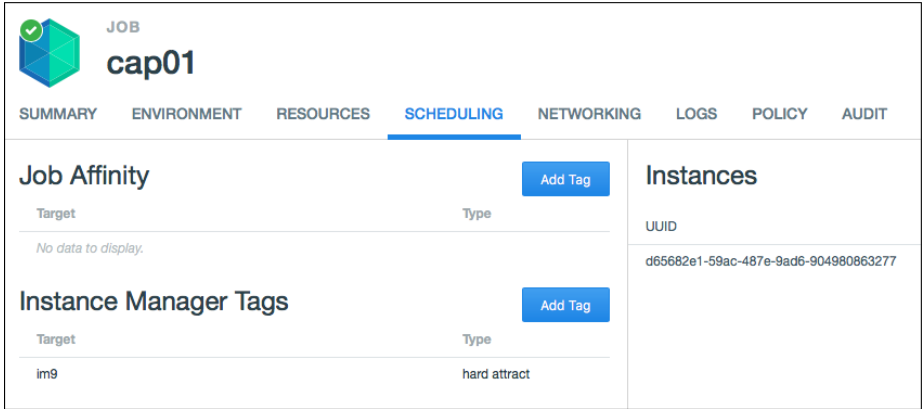
Step	Instruction																																																																																
C.2.1	<p>Execute the <code>apc cluster im list</code> command again so that you can view the number of instances running on each IM as well as their tags.</p> <table><tr><th>Name</th><th>UUID</th><th>Uptime</th><th>Instances</th><th>Mem (Res/Max)</th><th>Disk (Res/Max)</th><th>Net (Res/Max)</th><th>Tags</th></tr><tr><td>kiso-037ffac0</td><td>bb1042d5</td><td>30m</td><td>4</td><td>304/16886 MB</td><td>1792/131072 MB</td><td>35/1000 Mbps</td><td>aws im3 us-west-2a</td></tr><tr><td>kiso-03c4b4b3</td><td>2110815a</td><td>30m</td><td>3</td><td>288/16886 MB</td><td>1536/131072 MB</td><td>25/1000 Mbps</td><td>aws im8 us-west-2c</td></tr><tr><td>kiso-1bb9fa65</td><td>c58a45c9</td><td>30m</td><td>7</td><td>1552/16886 MB</td><td>5888/131072 MB</td><td>45/1000 Mbps</td><td>aws im2 us-west-2a</td></tr><tr><td>kiso-2f5536c3</td><td>07d40ac2</td><td>30m</td><td>6</td><td>400/16886 MB</td><td>2048/131072 MB</td><td>60/1000 Mbps</td><td>aws im7 us-west-2c</td></tr><tr><td>kiso-31a0bc97</td><td>1b700c95</td><td>31m</td><td>6</td><td>344/16886 MB</td><td>2304/131072 MB</td><td>55/1000 Mbps</td><td>aws im9 us-west-2c</td></tr><tr><td>kiso-4c8845ef</td><td>3fe3aace</td><td>31m</td><td>4</td><td>784/16886 MB</td><td>3072/131072 MB</td><td>30/1000 Mbps</td><td>aws im6 us-west-2b</td></tr><tr><td>kiso-53fbd4ac</td><td>aea34fcd</td><td>29m</td><td>2</td><td>272/16886 MB</td><td>1280/131072 MB</td><td>15/1000 Mbps</td><td>aws im5 us-west-2b</td></tr><tr><td>kiso-7c5e756c</td><td>c5aa5951</td><td>29m</td><td>4</td><td>776/16886 MB</td><td>3328/131072 MB</td><td>25/1000 Mbps</td><td>aws im4 us-west-2b</td></tr><tr><td>kiso-b16c73b1</td><td>367d81c5</td><td>31m</td><td>37</td><td>8016/16886 MB</td><td>32512/131072 MB</td><td>220/1000 Mbps</td><td>aws im1 us-west-2a</td></tr></table> <p>Each IM has 3 tags associated with it. Notice that all IMs has a common tag, aws. There are 3 IMs per Availability Zone (AZ); us-west-2a, us-west-2b, and us-west-2c. Each IM has a unique tag (im1, im2, ..., im9).</p>	Name	UUID	Uptime	Instances	Mem (Res/Max)	Disk (Res/Max)	Net (Res/Max)	Tags	kiso-037ffac0	bb1042d5	30m	4	304/16886 MB	1792/131072 MB	35/1000 Mbps	aws im3 us-west-2a	kiso-03c4b4b3	2110815a	30m	3	288/16886 MB	1536/131072 MB	25/1000 Mbps	aws im8 us-west-2c	kiso-1bb9fa65	c58a45c9	30m	7	1552/16886 MB	5888/131072 MB	45/1000 Mbps	aws im2 us-west-2a	kiso-2f5536c3	07d40ac2	30m	6	400/16886 MB	2048/131072 MB	60/1000 Mbps	aws im7 us-west-2c	kiso-31a0bc97	1b700c95	31m	6	344/16886 MB	2304/131072 MB	55/1000 Mbps	aws im9 us-west-2c	kiso-4c8845ef	3fe3aace	31m	4	784/16886 MB	3072/131072 MB	30/1000 Mbps	aws im6 us-west-2b	kiso-53fbd4ac	aea34fcd	29m	2	272/16886 MB	1280/131072 MB	15/1000 Mbps	aws im5 us-west-2b	kiso-7c5e756c	c5aa5951	29m	4	776/16886 MB	3328/131072 MB	25/1000 Mbps	aws im4 us-west-2b	kiso-b16c73b1	367d81c5	31m	37	8016/16886 MB	32512/131072 MB	220/1000 Mbps	aws im1 us-west-2a
Name	UUID	Uptime	Instances	Mem (Res/Max)	Disk (Res/Max)	Net (Res/Max)	Tags																																																																										
kiso-037ffac0	bb1042d5	30m	4	304/16886 MB	1792/131072 MB	35/1000 Mbps	aws im3 us-west-2a																																																																										
kiso-03c4b4b3	2110815a	30m	3	288/16886 MB	1536/131072 MB	25/1000 Mbps	aws im8 us-west-2c																																																																										
kiso-1bb9fa65	c58a45c9	30m	7	1552/16886 MB	5888/131072 MB	45/1000 Mbps	aws im2 us-west-2a																																																																										
kiso-2f5536c3	07d40ac2	30m	6	400/16886 MB	2048/131072 MB	60/1000 Mbps	aws im7 us-west-2c																																																																										
kiso-31a0bc97	1b700c95	31m	6	344/16886 MB	2304/131072 MB	55/1000 Mbps	aws im9 us-west-2c																																																																										
kiso-4c8845ef	3fe3aace	31m	4	784/16886 MB	3072/131072 MB	30/1000 Mbps	aws im6 us-west-2b																																																																										
kiso-53fbd4ac	aea34fcd	29m	2	272/16886 MB	1280/131072 MB	15/1000 Mbps	aws im5 us-west-2b																																																																										
kiso-7c5e756c	c5aa5951	29m	4	776/16886 MB	3328/131072 MB	25/1000 Mbps	aws im4 us-west-2b																																																																										
kiso-b16c73b1	367d81c5	31m	37	8016/16886 MB	32512/131072 MB	220/1000 Mbps	aws im1 us-west-2a																																																																										
C.2.2	Make a note of the unique tag (im*) for the IM that is running your cap01.																																																																																
C.2.3	<p>Execute the following command to move your <code>cap01</code> instance to another IM.</p> <pre>apc capsule update cap01 -ha <im*> --restart</pre> <p>Replace <code><im*></code> with a tag of IM that you want to move cap01 to.</p> <p>For example, if your cap01 is running on a host, kiso-b16c73b1, its tag is im1. You want to move it to im7, then your command would look like:</p> <pre>apc capsule update cap01 -ha im7 --restart</pre>																																																																																
C.2.4	<p>Verify that cap01 is now running on the correct IM.</p> <pre>apc job instances cap01</pre>																																																																																



DEV101: Developing with Apcera Platform

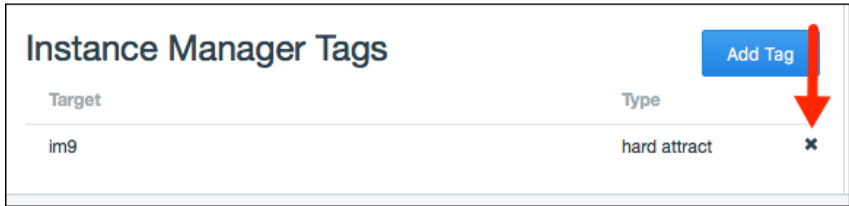
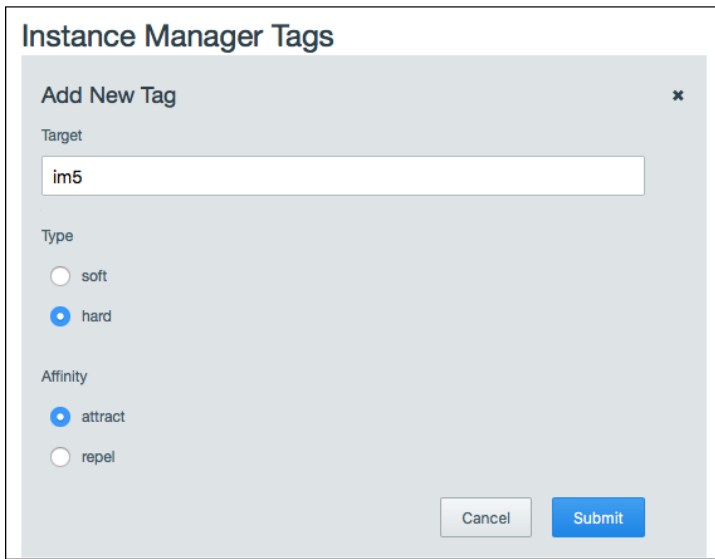
Lab 3: Extending Apcera

Task 3: Move your workload via Web Console

Step	Instruction
C.3.1	<p>Log in to the Web Console at https://console.kiso.io, and navigate to Capsules under JOBS.</p> 
C.3.2	<p>Select cap01, and then SCHEDULING tab.</p> 

DEV101: Developing with Apcera Platform


Lab 3: Extending Apcera

C.3.3	<p>Click the x icon next to your instance manager tag entry to remove it.</p> 
C.3.4	Click Add Tag .
C.3.5	<p>In the Add New Tag wizard, enter the following:</p> <ul style="list-style-type: none"> Target: <IM tag that you want to move cap01 to (e.g. im5)> Type: hard 
C.3.6	Click Submit .
C.3.7	When you are prompted to restart the job, click Yes, restart now . Wait until the Instances list updates.



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

C.3.8	<p>Under Instances, select the host name of the Instance Manager.</p> <div><div>Instances</div><table><tr><th>UUID</th><th>State</th><th>Instance Manager</th><th>Data Center</th><th>Uptime</th></tr><tr><td>fab0d0dc-7843-4af5-87e2-1419aa8a...</td><td>RUNNING</td><td>kiso-53fbd4ac</td><td>defaultdc</td><td>10 minutes 11 ...</td></tr></table></div>	UUID	State	Instance Manager	Data Center	Uptime	fab0d0dc-7843-4af5-87e2-1419aa8a...	RUNNING	kiso-53fbd4ac	defaultdc	10 minutes 11 ...
UUID	State	Instance Manager	Data Center	Uptime							
fab0d0dc-7843-4af5-87e2-1419aa8a...	RUNNING	kiso-53fbd4ac	defaultdc	10 minutes 11 ...							
C.3.9	<p>Verify the tag of the IM matches to what you provided at Step C.3.5.</p> <div><div><div><div>INSTANCE MANAGER</div><div> kiso-53fbd4ac</div></div><div>DETAILS INSTANCES</div><div><div><div>Info</div><div>Uptime 3 hours 55 seconds</div><div>Instances 3</div><div>Data Center defaultdc</div><div>Tags aws, us-west-2b, im5</div><div>Stats</div></div><div><div>Resources</div><div>RAM</div><div>14.0 GB</div><div>9.3 GB</div><div>4.7 GB</div><div>0 Bytes</div></div></div></div></div>										
C.3.10	<p>Scroll down, and locate the Instances section. Select the instance manager name where your cap01 instance is running. This takes you to the detailed page of the IM.</p> <p>Notice that the tag matches to the one you provided in the apc command.</p>										



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

Task 4: Schedule jobs using policy

Step	Instruction
C.4.1	<p>In the web console, modify your policy by adding the following rule.</p> <pre>job::/sandbox/<user_name>::cap02 { { schedulingTag.hard <IM_tag> } }</pre> <p>Replace <code><user_name></code> with your user name, and <code><IM_tag></code> with the IM tag of where you want to run the cap02 instances.</p> <p>Example: If your user name is james, and you want cap02 to run in any of the three IMs that are tagged with us-west-2b, the policy would look like:</p> <pre>job::/sandbox/james::cap02 { { schedulingTag.hard us-west-2b } }</pre>
C.4.2	<p>Now create a capsule named, cap02 in your namespace.</p> <pre>apc capsule create cap02 --image linux --batch</pre>
C.4.3	<p>Verify that the cap02 instance is running in the IM as you expected.</p>
C.4.4	<p>Increase the number of instances for cap02 to ensure that cap02 will always run in the targeted IM.</p> <pre>apc capsule update cap02 -i 3</pre>
C.4.5	<p>Execute the following command to verify that they are all running on the same IM.</p> <pre>apc job instances cap02</pre>



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

C.4.6	<p>Delete the capsules.</p> <pre>apc capsule delete cap01 apc capsule delete cap02</pre>
-------	--



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

Exercise D: Enabling Broadcast and Multicast Routes

In this exercise, you experiment how the broadcast and multicast routes work in the Apcera Platform by creating a virtual network and capsules.

To learn more about this feature, refer to the document “Enabling Broadcast and Multicast Route” at the following URL: <http://docs.apcera.com/jobs/virtual-networks/-enabling-broadcast-and-multicast-routes>

Task 1: Create a virtual network and Linux capsules

Step	Instruction
D.1.1	Execute the following command to create a new virtual network. <pre>apc network create vnet01</pre>
D.1.2	Execute the following command to create a capsule in vnet01 network. <pre>apc capsule create job01 --network vnet01 --image linux -ae</pre>
D.1.3	Repeat the command to create two more capsules, job02 and job03. <pre>apc capsule create job02 --network vnet01 --image linux -ae</pre> <pre>apc capsule create job03 --network vnet01 --image linux -ae</pre>
D.1.4	Execute the following command to verify that all three capsules have joined vnet01. <pre>apc network show vnet01</pre>

Task 2: Update the capsules to enable broadcast route

Step	Instruction
D.2.1	Execute the following command to tail the server logs. <pre>apc job update job01 --network vnet01 --broadcast-enable</pre>



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

D.2.2	<p>Repeat the command to enable broadcast route for job02 and job03.</p> <pre>apc job update job02 --network vnet01 -be apc job update job03 --network vnet01 -be</pre> <p>Note: -be is a shorthand for the --broadcast-enable flag.</p>
D.2.3	<p>Connect to the capsule, job01:</p> <pre>apc capsule connect job01</pre>
D.2.4	<p>Execute the following command to view its route table:</p> <pre>root@ip-169-254-0-3:/root# route -n Kernel IP routing table Destination Gateway Genmask Flags Metric Ref Use Iface 0.0.0.0 169.254.0.2 0.0.0.0 UG 0 0 0 veth_8e42df7b 169.254.0.2 0.0.0.0 255.255.255.254 U 0 0 0 veth_8e42df7b 192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 0 vi-3b00-9e1c56 255.255.255.255 0.0.0.0 255.255.255.255 UH 0 0 0 vi-3b00-9e1c56</pre> <p>Notice that broadcast address (255.255.255.255) has been added.</p> <p>Stay connected to the capsule, job01.</p>

Task 3: Install socat utility

Step	Instruction
D.3.1	<p>Inside the job01 capsule, execute the following APT package handling utility to install socat.</p> <pre>apt-get update apt-get install socat</pre> <p>Note: At the message, "Do you want to continue? [Y/n]", press enter to continue.</p>
D.3.2	<p>Open a <u>separate</u> terminal, and connect to the job02 capsule.</p> <pre>apc capsule connect job02</pre>



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

D.3.3	Inside the job02 capsule, repeat the step to install socat. <code>apt-get update</code> <code>apt-get install socat</code>
D.3.4	Open another <u>separate</u> terminal, and connect to the job03 capsule. <code>apc capsule connect job03</code>
D.3.5	Inside the job03 capsule, repeat the step to install socat. <code>apt-get update</code> <code>apt-get install socat</code> Note: At this point, you should have three terminal windows opened each connected to job01, job02, and job03.

Task 4: Test the broadcast route

Step	Instruction
D.4.1	Inside the terminal connected to job01 , execute the following command: <code>socat STDIO UDP4- DATAGRAM:255.255.255.255:6666,bind=:6666,broadcast</code>
D.4.2	Inside the terminal connected to job02 , execute the same command: <code>socat STDIO UDP4- DATAGRAM:255.255.255.255:6666,bind=:6666,broadcast</code>
D.4.3	Inside the terminal connected to job03 , execute the same command: <code>socat STDIO UDP4- DATAGRAM:255.255.255.255:6666,bind=:6666,broadcast</code>

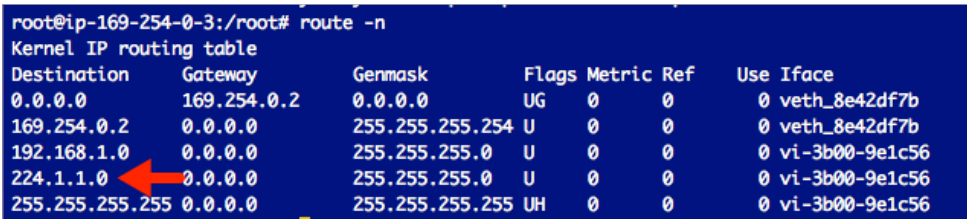


DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

D.4.4	Inside the terminal connected to job03, type “hello” and press enter. The message, “hello” should appear in all three capsules including job03. You sent the message from job03, and job03 receives this message as well since it’s a part of the broadcasting group.
D.4.5	Send different message such as “world” from another capsule. All capsule should receive the message.
D.4.6	Press Control + C to kill the <code>socat</code> utility inside all three capsules, but stay connected to the capsules.

Task 5: Add multicast route

Step	Instruction
D.5.1	Open another terminal window, and then update the capsules to add multicast route by executing the following command. <pre>apc job update job01 --network vnet01 -ma 224.1.1.1/24 apc job update job02 --network vnet01 -ma 224.1.1.1/24 apc job update job03 --network vnet01 -ma 224.1.1.1/24</pre>
D.5.2	Inside the terminal connected to job01 , execute the following command to view its route table: <pre>route -n</pre>  Note: Make a note of the Destination IP address (e.g. 224.1.1.0) in the routing table.



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

D.5.3	<p>Execute the following command to start a client <code>socat</code> utility in job01:</p> <pre>socat STDIO UDP4-DATAGRAM:<destinationIP>:1234</pre> <p>Note: If the destination IP address you noted in step D.5.2 was 224.1.1.0, the command should look like:</p> <pre>socat STDIO UDP4-DATAGRAM:224.1.1.0:1234</pre>
D.5.4	<p>Inside the terminal connected to job02, execute the same command:</p> <pre>socat STDIO UDP4-DATAGRAM:<destinationIP>:1234</pre> <p>(e.g. <code>socat STDIO UDP4-DATAGRAM:224.1.1.0 1234</code>)</p>
D.5.5	<p>Inside the terminal connected to job03, execute the following command to view its route table and make a note of its interface.</p> <pre>route -n</pre> <div><pre>root@ip-169-254-0-7:/root# route -n Kernel IP routing table Destination Gateway Genmask Flags Metric Ref Use Iface 0.0.0.0 169.254.0.6 0.0.0.0 UG 0 0 0 veth_7f59ac24 169.254.0.6 0.0.0.0 255.255.255.254 U 0 0 0 veth_7f59ac24 192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 0 vi-3b00-0e23fc 224.1.1.0 0.0.0.0 255.255.255.0 U 0 0 0 vi-3b00-0e23fc 255.255.255.255 0.0.0.0 255.255.255.255 UH 0 0 0 vi-3b00-0e23fc</pre></div> <p>Iface indicates the interface to which packets for this route will be sent.</p>
D.5.6	<p>Execute the following command to make job03 a server:</p> <pre>socat STDIO UDP4-RECV:1234,ip-add-membership=<destinationIP>:<iface></pre> <p>Be sure to replace <code><iface></code> with the actual interface you noted in step D.5.5. For example, your command should look like:</p> <pre>socat STDIO UDP4-RECV:1234,ip-add-membership=224.1.1.0: vi-3b00-0e23fc</pre>



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

	In this example, the destination IP address is 224.1.1.0, and the interface is vi-3b00-0e23fc.
D.5.7	<p>Inside the terminal connected to job01, type some text such as “hello” and press enter.</p> <p>The message should be broadcasted only to job03. The job02 capsule should not receive this message.</p>
D.5.8	<p>Similarly, inside the terminal connected to job02, type some text such as “world” and press enter.</p> <p>The job01 capsule should not receive this message, and job03 is the only job that received “world”.</p>

Task 6: Clean up

Step	Instruction
D.6.1	In all three terminals, press Control + C to kill the socat utility, and then type exit to disconnect from the capsule.
D.6.2	<p>Execute the following command to delete the capsules.</p> <pre>apc capsule delete job01 apc capsule delete job02 apc capsule delete job03</pre>
D.6.3	<p>Execute the following command to delete the network, vnet01.</p> <pre>apc network delete vnet01</pre>



DEV101: Developing with Apcera Platform

Lab 3: Extending Apcera

Exercise Review

In this lab you explore how to extend Apcera and customize functionality. First you created a manifest to deploy an app. Second, you created a custom package, java-1.8. You also explored the workload mobility feature of the Apcera Platform. Lastly, you enabled broadcast and multicast routes on jobs, and examined how a packets can be sent across multiple jobs belonging to the same virtual network.

End of the Lab



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

Lab Introduction

In this lab you practice how to troubleshoot apps you have deployed to Apcera. First you create a Log Drain for one of your apps. Next you do some live debugging using an app console. Lastly, you explore how to debug an app that fails to stage.

Lab Exercise

This lab comprises the following exercises:

- A. Configure Log Drain Service
- B. Perform Live Debugging Using App Console
- C. Debug App Failing to Stage

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

- Logging and Debugging: <http://docs.apcera.com/jobs/logs/>
- Stager debugging: <http://docs.apcera.com/packages/staging/ - debugging-and-troubleshooting-staging>

Lab Instruction

For each exercise, complete the steps.



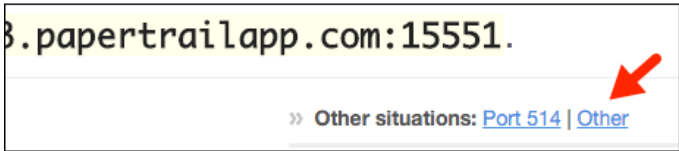
DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

Exercise A: Configure a Log Drain

In this exercise you configure a syslog service to act as a log drain for your app. Apcera apps can send log messages to a syslog service such as Papertrail or Splunk. For this exercise we use Papertrail.

Task 1: Create and configure a Papertrail account

Step	Instruction
A.1.1	Go to the following URL: https://papertrailapp.com/dashboard .
A.1.2	Create an account and log in to your account.
A.1.3	Go to the following URL: https://papertrailapp.com/start .
A.1.4	Click the Add systems link.
A.1.5	Click the Other link at the top. 
A.1.6	Ensure that the option B: I use Cloud Foundry is selected, and give the configuration a name such as MyLogDrain.
A.1.7	Click Save .



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

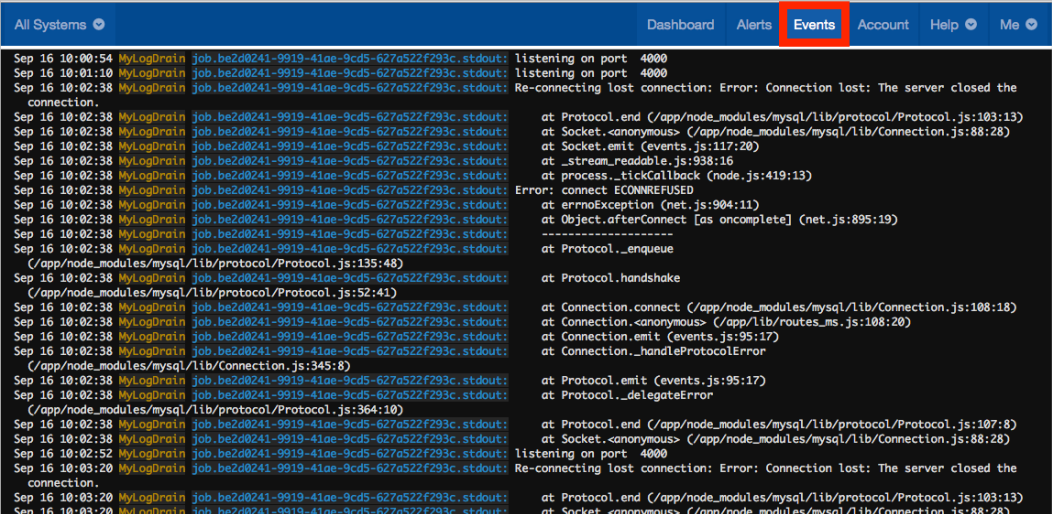
Task 2: Add the Log Drain to your app

Step	Instruction
A.2.1	<p>Copy the host-port pair that the configuration provides to you. The screenshot below highlights an example host-port pair that you copy to the clipboard or a local text file.</p> 
A.2.2	<p>Execute the following command:</p> <pre>apc app update my-node-todo-app -ae --restart</pre>
A.2.3	<p>Execute the following command:</p> <pre>apc drain add syslog://<host-port> --app my-node-todo-app</pre> <p>Where <host-port> is the host-port pair provided to you by Papertrail.</p> <p>For example:</p> <pre>apc drain add syslog://logs2.papertrailapp.com:26902 --app my-node-todo-app</pre> <p>You should see that the Drain URL is added to the app.</p> 



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

A.2.4	<p>Execute the following command to verify.</p> <pre>apc app show my-node-todo-app</pre> <p>You should see that the Drains row indicates the syslog://<hostname>:<port> that you added.</p>
A.2.5	<p>To view log events, at the Papertrail Dashboard page for you app (https://papertrailapp.com/dashboard), select the Events tab in the upper right of the menu.</p> <p>As shown below, you should see that MyLogDrain is listening on port 4000 and that events are logged as they are received. To generate events, perform some actions on the databases at the Node app web page. (NOTE: Refresh the web page if you don't see the logs.)</p>  <p>The screenshot shows the Papertrail Events tab with a list of log events. The events are from the 'MyLogDrain' job and show the application listening on port 4000 and handling connections. The events are timestamped and include the job ID and the syslog output.</p>



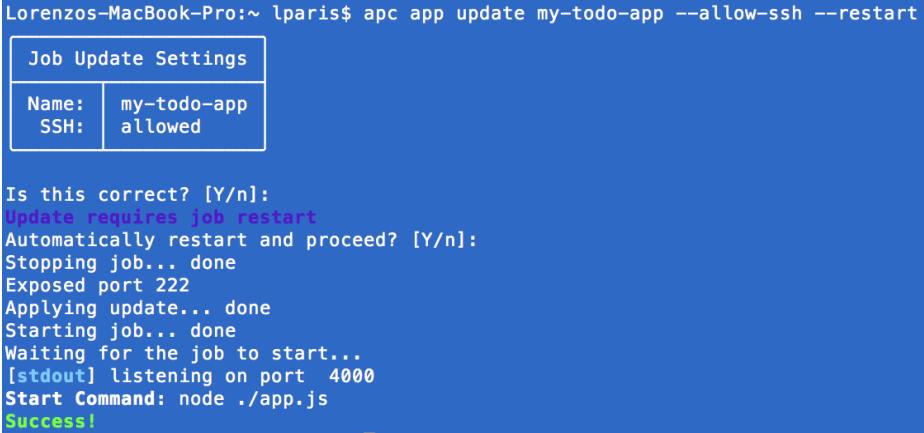
DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

Exercise B: Perform live debugging using app console

In this exercise you perform some live debugging of your deployed my-node-todo-app.

Task 1: Debug your my-node-todo-app

Step	Instruction
B.1.1	<p>Execute the following command to ensure that your my-node-todo-app is running.</p> <pre>apc app list</pre> <p>Or, alternatively:</p> <pre>apc app show my-node-todo-app</pre>
B.1.2	<p>Execute the following command to allow SSH for the app.</p> <pre>apc app update my-node-todo-app --allow-ssh</pre> <p>Access to the job container OS via SSH is controlled through policy.</p> <p>You should see that the SSH port 222 is exposed, and the app is updated and restarted. Note that you are also given the start command for starting the app: <code>node ./app.js</code>.</p> 



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

B.1.3	<p>You can verify that you have exposed the SSH port 222 using command <code>apc app show <app-name></code>. For the Exposed Ports entry, you will see port 222 is listed. In addition, for the Tags entry you will see "ssh: true".</p> <p>Because you have exposed the SSH port, you could connect directly to the live instance with the command:</p> <pre>apc app connect <app-name></pre> <p>You can explore the file system and attempt to debug. However, since the app is live, this approach is NOT ideal in a production environment. Furthermore, exposing SSH port 222 is not recommended for production apps. To address these issues, Apcera provides the <code>app console</code> command which gives you a convenient way to debug live apps without having to expose the SSH port and possibly disrupt an app.</p>
B.1.4	<p>Since allowing SSH access is not recommended for production apps, remove SSH ingress by issuing the following command:</p> <pre>apc app update my-node-todo-app --remove-ssh</pre>
B.1.5	<p>Execute the following command to verify that SSH access is removed using the following command:</p> <pre>apc app show my-node-todo-app.</pre>
B.1.6	<p>Connect to a clone of the app using the following command.</p> <pre>apc app console my-node-todo-app</pre> <p>Using command <code>apc app console <app-name></code> is the recommended approach for debugging live apps because it <u>connects to a clone of your app</u> running within an Apcera capsule. You connect to the cloned app using SSH without exposing the SSH port on the live app. The clone includes all service bindings present in the original app, and lets you debug the app without disturbing the running instance.</p> <p>If you have successfully cloned the app and connected to the capsule, you will see a series of outputs similar to the following:</p>



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

	<pre>Creating a capsule for "job::/sandbox/yoko::my-node-todo-app"... done Creating binding to "outside"... done Creating binding to "my-postgres-service"... done Creating binding to "docker-db"... done Starting the capsule... done Waiting for the capsule to start... Start Command: node ./app.js root@ip-169-254-0-5:/#</pre> <p>NOTE: The connection is via SSH but the live app has not exposed this port. SSH is exposed for the <u>cloned capsule app only</u> so that you can debug it.</p>
B.1.7	<p>If receive the following policy error:</p> <pre>Error: Namespace memory quota of 2GB on "quota::/sandbox/<user>" exceeded.</pre> <p>Ask your instructor to increase the quota of your namespace.</p>

Task 2: Explore the capsule app file system

By using the app console command to clone the app, you can explore the capsule app file system without disrupting the running app instance.

Step	Instruction
B.2.1	Change your working directory to root. <code>cd /</code>
B.2.2	List the / directories: <code>ls</code>
B.2.3	Change your working directory to app. <code>cd /app</code>
B.2.4	Execute the following command to read the contents of the app.js file. <code>cat app.js</code>



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

B.2.5	<p>Execute the following command to read the contents of the package.json file to check the dependency information.</p> <pre>cat package.json</pre>
B.2.6	<p>Execute the following command to list all the environment variables.</p> <pre>env</pre> <p>You should be able to view the environment variables such as:</p> <ul style="list-style-type: none">• PATH• HOME• START_PATH• CNTM_HOST_IP• CNTM_PROCESS_NAME• CNTM_INSTANCE_UUID• CNTM_JOB_UUID• CNTM_JOB_FQN• POSTGRES_URI• MYSQL_URI
B.2.7	<p>When you are done exploring, issue the following command:</p> <pre>exit</pre> <p>You should see that the capsule app is stopped and deleted:</p> <pre>root@ip-169-254-0-35:/# exit logout Connection to 127.0.0.1 closed. Stopping the capsule... done Deleting the capsule... done</pre> <p>If you run command <code>apc app show my-node-todo-app</code> you will see that there are no remnants of the capsule. The job instance is running and the SSH port remains closed.</p>



DEV101: Developing with Apcera Platform

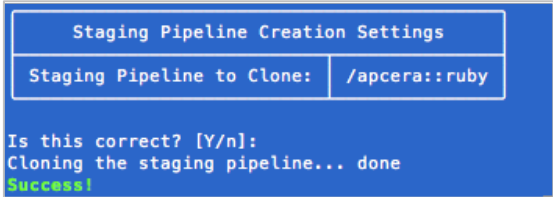
Lab 4: Troubleshoot Jobs

Exercise C: Debug an app failing to stage

In this exercise you debug an app that fails to stage. Staging failures can result from individual stagers or the staging coordinator not starting, problems mounting packages, errors in application code, and environment issues.

Here you create a custom stager and use it to illustrate some common staging failures. This stager runs a suite of rspec tests included in the app's source code.

Task 1: Clone the system-level Ruby staging pipeline

Step	Instruction
C.1.1	<p>Change your working directory to /sample-apps/demo-ruby-sinatra.</p> <pre>cd sample-apps/demo-ruby-sinatra</pre> <p>NOTE: You can safely <u>ignore</u> the message "ruby-1.9.3-p392 is not installed" if you receive it.</p>
C.1.2	<p>Browse through what's in this folder (files, sub-folders, etc.).</p>
C.1.3	<p>Execute the following command to clone the Ruby staging pipeline into your namespace.</p> <pre>apc staging pipeline clone /apcera::ruby</pre>
C.1.4	<p>After your policy was updated, run the command again.</p>  <p>This creates a copy of the Apcera-provided Ruby staging pipeline in your /sandbox/<user_name> namespace. Issue the following command to verify the cloning operation.</p>



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

C.1.5

Execute the following command to view the list of available staging pipelines.

apc staging pipeline list

```
Lorenzos-MacBook-Pro:demo-ruby-sinatra lparis$ apc staging pipeline list
Working in "/sandbox/continuumtraining2015"
```

Name	Namespace	Stagers
ruby	/sandbox/continuumtraining2015	job::/apcera/stagers::ruby

Task 2: Create an rspec stager

Step	Instruction																				
C.2.1	<p>Change the working directory to demo-ruby-sinatra/rspec-stager.</p> <pre>cd ./rspec-stager</pre>																				
C.2.2	<p>Execute the following command to create an rspec stager from the bash script in the demo-ruby-sinatra directory:</p> <pre>apc stager create my-rspec-stager --path rspec-stager --start-command ./rspec-stager --allow-egress</pre> <p>You should see that the stager named my-rspec-stager is successfully created.</p> <div><pre>Lorenzos-MacBook-Pro:demo-ruby-sinatra lparis\$ apc stager create my-rspec-stager --path rspec-stager --start-command ./rspec-stager --additive --allow-egress</pre><table><thead><tr><th colspan="2">Stager Creation Settings</th></tr></thead><tbody><tr><td>Stager Name:</td><td>my-rspec-stager</td></tr><tr><td>Start Command:</td><td>./rspec-stager</td></tr><tr><td>Path:</td><td>rspec-stager</td></tr><tr><td>CPU:</td><td>0ms/s (uncapped)</td></tr><tr><td>Memory:</td><td>256MB</td></tr><tr><td>Disk:</td><td>1024MB</td></tr><tr><td>Network:</td><td>5Mbps</td></tr><tr><td>Netmax:</td><td>0Mbps (uncapped)</td></tr><tr><td>Open Egress</td><td>true</td></tr></tbody></table><pre>Is this correct? [Y/n]: y Packaging... done Creating package "my-rspec-stager"... done Uploading package contents... 100% Creating the stager... done Allowing open network egress for "my-rspec-stager"... done Success!</pre></div>	Stager Creation Settings		Stager Name:	my-rspec-stager	Start Command:	./rspec-stager	Path:	rspec-stager	CPU:	0ms/s (uncapped)	Memory:	256MB	Disk:	1024MB	Network:	5Mbps	Netmax:	0Mbps (uncapped)	Open Egress	true
Stager Creation Settings																					
Stager Name:	my-rspec-stager																				
Start Command:	./rspec-stager																				
Path:	rspec-stager																				
CPU:	0ms/s (uncapped)																				
Memory:	256MB																				
Disk:	1024MB																				
Network:	5Mbps																				
Netmax:	0Mbps (uncapped)																				
Open Egress	true																				



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

C.2.3	<p>Execute the following command to append the rspec stager to your cloned Ruby staging pipeline.</p> <pre>apc help staging pipeline</pre> <p>Stagers run within staging pipelines. A staging pipeline is an ordered set of one or more stagers. You can change the order in which stagers run within a staging pipeline by adding one or more stagers to the beginning of the pipeline (prepend) or to the end of the pipeline (append).</p> <p>Here we will append the rspec-stager to the Ruby staging pipeline using the following syntax:</p> <pre>apc staging pipeline append <pipeline-name> <stager-names> [...]</pre>
C.2.4	<p>Execute the following command to append my-rspec-stager to the Ruby staging pipeline:</p> <pre>apc staging pipeline append ruby my-rspec-stager</pre> <p>You should see that the ruby staging pipeline is successfully updated.</p>
C.2.5	<p>Execute the following command to view the details about the ruby staging pipeline:</p> <pre>apc staging pipeline show ruby</pre> <p>Notice that ruby staging pipeline contains ruby and my-rspec-stager stagers now.</p>
C.2.6	<p>Change the working directory back to /demo-ruby-sinatra.</p> <pre>cd ..</pre>
C.2.7	<p>Execute the following command to create the demo-ruby-sinatra application:</p> <pre>apc app create my-ruby-app --start --staging ruby --batch</pre> <p>As the app is created, Apcera will stage and run the rspec tests.</p>
C.2.8	<p>Verify the app creation by navigate to the route URL with your browser, such as:</p> <p>App should be accessible at "http://my-ruby-app-t0y5gc.kiso.io"</p>



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

	<div>Sinatra Sample</div> <div><h2>Sinatra Sample</h2><pre>ruby 1.9.3p547 (2014-05-14 revision 45962) [x86_64-linux] rubygems 2.4.1 \$ env RUBYOPT=-rbundler/setup -r//app/_set_stdout_sync.rb PORT=4000 HOME=/ OLDPWD=/ CNTM_HOST_IP=192.168.129.129 TMPDIR=/tmp START_PATH=/app RACK_ENV=production CNTM_INSTANCE_UUID=8b471d52-a6b8-42ff-888b-dbf9f3aee102 PATH=/app/vendor/bundle/ruby/1.9.1/bin:/opt/apcera/ruby-1.9.3-p547/bin:/bin:/ CNTM_PROCESS_NAME=app CNTM_HOST_PORT=192.168.129.129:2056 CNTM_JOB_FQN=job::/sandbox/yoko.hyakuna::my-ruby-app START_COMMAND=bundle exec rackup config.ru -p 4000 CNTM_JOB_UUID=dab64286-25e1-4071-8e04-652b42e4116e</pre></div>
--	---

Task 3: Change the code to break the rspec tests

Now, let's break the app you just created so we can troubleshoot it.

Step	Instruction
C.3.1	In a text editor, open the source code, <code>demo-ruby-sinatra/spec/app_spec.rb</code> , and then change the expression at line 14 from <code>'last_response.body.should == "Alive"'</code> to <code>'last_response.body.should == "Dead"'</code> .



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

	<pre>app_spec.rb 1 require "spec_helper" 2 3 describe App do 4 include Rack::Test::Methods 5 6 def app 7 App 8 end 9 10 describe "heartbeat" do 11 it "should return 'Alive'" do 12 get "/status/heartbeat" 13 last_response.ok?.should be_true 14 last_response.body.should == "Dead" 15 end 16 end 17 18 describe "db" do</pre> <p>Save the code.</p>
C.3.2	<p>Then, redeploy the app:</p> <pre>apc app deploy my-ruby-app --staging ruby</pre> <p>Expected result: The redeploy should fail because you broke the rspec tests. The error message is "Error: Staging has failed." This shows how you can automate some features on your continuous integration suite as part of your deployment process.</p>



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

C.3.3	<p>Create another app with stager debugging enabled by executing the following command:</p> <pre>apc app create my-ruby-app2 --start --batch --staging ruby --pkg-env STAGER_DEBUG=vvv</pre> <p>Sometimes, it is useful to have additional stager job information when you are troubleshooting an issue. Use the <code>--pkg-env STAGER_DEBUG=<verbosity-level></code> parameter to enable the stager debugging.</p> <pre>apc app create <app-name> --pkg-env STAGER_DEBUG=<verbosity-level></pre> <p>While the verbosity levels are:</p> <ul style="list-style-type: none">• v - display the same stager messaging as default, but the individual output lines will be tagged with contextual severity labels.• vv - tag lines and introduce debug-level output to the user.• vvv - all of the above in addition to performing some debriefing commands should a fatal error occur. <p>For more detail, refer to the online documentation: http://docs.apcera.com/packages/staging/#debugging-and-troubleshooting-staging</p>
C.3.4	<p>Fix the source code, and re-deploy the app. It should succeed.</p>

Task 4: Update your rspec stager job to have a bad start command

Stagers within Apcera are just jobs; they can fail like any other job. When stagers fail, Apcera exposes the failures to you in an intuitive way.

Step	Instruction
C.4.1	<p>Execute the following command to update your rspec stager to have a bad start command:</p> <pre>apc job update my-rspec-stager --start-command "./bad"</pre>
C.4.2	<p>Execute the following command to re-deploy your Ruby application:</p> <pre>apc app deploy my-ruby-app --staging ruby</pre>



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

	<p>Expected result: The re-deploy fails.</p> <p>This indicates that the binary the bad start command is trying to run was not found, and as a result, we couldn't start the stager. Then, we fail the package.</p>
--	--

Task 5: Clean up your environment

Step	Instruction
C.5.1	<p>Execute the following command to delete the stager:</p> <pre>apc stager delete my-rspec-stager --force</pre>
C.5.2	<p>Execute the following command to delete the app:</p> <pre>apc app delete my-ruby-app</pre>



DEV101: Developing with Apcera Platform

Lab 4: Troubleshoot Jobs

Exercise Review

In this lab you learned how to troubleshoot the apps you have deployed to Apcera. First, you learned how to connect an app to a syslog service and pipe stdout to the service. Next, you learned how to create a console to perform live debugging of an app. Lastly, you learned how to debug an app that fails to stage.

End of the Lab