Karamel Documentation

Release 0.2

www.karamel.io

Jun 20, 2018

Contents

CHAPTER 1

What is Karamel?

Karamel is a management tool for reproducibly deploying and provisioning distributed applications on bare-metal, cloud or multi-cloud environments. Karamel provides explicit support for reproducible experiments for distributed systems. Users of Karamel experience the tool as an easy-to-use UI-driven approach to deploying distributed systems or running distributed experiments, where the deployed system or experiment can be easily configured via the UI.

Karamel users can open a cluster definition file that describes a distributed system or experiment as:

- the application stacks used in the system, containing the set of services in each application stack,
- the provider(s) for each application stack in the cluster (the cloud provider or IP addresses of the bare-metal hosts),
- the number of nodes that should be created and provisioned for each application stack,
- configuration parameters to customize each application stack.

Karamel is an orchestration engine that orchestrates:

- the creation of virtual machines if a cloud provider is used;
- the global order for installing and starting services on each node;
- the injection of configuration parameters and passing of parameters between services;
- connecting to hosts using ssh and running chef recipes using chef solo.

  Karamel enables the deployment of arbitrarily large distributed systems on both virtualized platforms (AWS, Vagrant) and bare-metal hosts.

Karamel is built on the configuration framework, Chef. The distributed system or experiment is defined in YAML as a set of node groups that each implement a number of Chef recipes, where the Chef cookbooks are deployed on github. Karamel orchestrates the execution of Chef recipes using a set of ordering rules defined in a YAML file (Karamelfile) in each cookbook. For each recipe, the Karamelfile can define a set of dependent (possibly external) recipes that should be executed before it. At the system level, the set of Karamelfiles defines a directed acyclic graph (DAG) of service dependencies. Karamel system definitions are very compact. We leverage Berkshelf to transparently download and install transitive cookbook dependencies, so large systems can be defined in a few lines of code. Finally, the Karamel runtime builds and manages the execution of the DAG of Chef recipes, by first launching the virtual machines or configuring the bare-metal boxes and then executing recipes with Chef Solo. The Karamel runtime executes the node setup steps using JClouds and Ssh. Karamel is agentless, and only requires ssh to be installed on the target host. Karamel transparently handles faults by retrying, as virtual machine creation or configuration is not always reliable or timely.

Existing Chef cookbooks can easily be `karamelized`, that is, wrapped and extended with a Karamelfile containing orchestration rules. In contrast to Chef, which is used primarily to manage production clusters, Karamel is designed to support the creation of reproducible clusters for running experiments or benchmarks. Karamel provides additional Chef cookbook support for copying experiment results to persistent storage before tearing down clusters.

In Karamel, infrastructure and software are delivered as code while the cluster definitions can be configured by modifying the configuration parameters for the services containined in the cluster definition. Karamel uses Github as the artifact-server for Chef cookbooks, and all experiment artifacts are globally available - any person around the globe can replay/reproduce the construction of the distributed system.

Karamel leverages virtual-machines to provision infrastructures on different clouds. We have cloud-connectors for Amazon EC2, Google Compute Engine, OpenStack and on-premises (bare-metal).

CHAPTER 2

Getting Started

2.1 How to run a cluster?

To run a simple cluster you need:

- a *cluster definition* file;
- access to a cloud (or bare-metal cluster);
- the Karamel client application.

You can use Karamel as a standalone application with a Web UI or embed Karamel as a library in your application, using the Java-API to start your cluster.

2.1.1 Linux/Mac

1. Starting Karamel

To run Karamel, download the Linux/Mac binaries from http://www.karamel.io. You first have to unzip the binaries (`tar -xf karamel-0.2.tgz`). From your machine's terminal (command-line), run the following commands:

```
cd karamel-0.2
./bin/karamel
```

This should open a window on your Web Browser if it is already open or open your default Web Browswer if one is not already open. Karamel will appear on the webpage opened.

2.1.2 Windows

1. Starting Karamel

To run Karamel, download the Windows binaries from http://www.karamel.io. You first have to unzip the binaries. From Windows Explorer, navigate to the folder `karamel-0.2` (probably in the `Downloads` folder) and double-click on `karamel.exe` file to start Karamel.

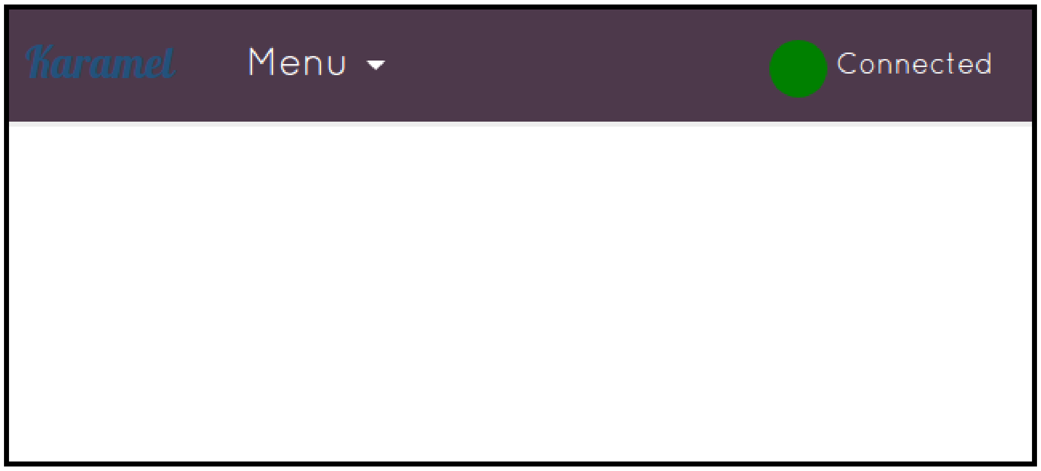


Fig. 1: Karamel Homepage. Click on `Menu` to load a Cluster Definition file.

2. Customize and launch your cluster Take a look into the *Board-UI*.

2.1.3 Command-Line in Linux/Mac

You can either set environment variables containing your EC2 credentials or enter them from the console. We recommend you set the environment variables, as shown below.

```
export AWS_KEY=...
export AWS_SECRET_KEY=...
./bin/karamel -launch examples/hadoop.yml
```

After you launch a cluster from the command-line, the client loops, printing out to stdout the status of the install DAG of Chef recipes every 20 seconds or so. Both the GUI and command-line launchers print out stdout and stderr to log files that can be found from the current working directory in:

```
tail -f log/karamel.log
```

2.1.4 Java-API:

You can run your cluster in your Java program by using our API.

1. Jar-file dependency First add a dependency into the karamel-core jar-file, its pom file dependency is as following:

```
<dependency>
  <groupId>se.kth</groupId>
  <artifactId>karamel-core</artifactId>
  <scope>compile</scope>
</dependency>
```

2. Karamel Java API Load the content of your cluster definition into a variable and call KaramelApi like this example:

```
//instantiate the API
KaramelApi api = new KaramelApiImpl();

//load your cluster definition into a java variable
String clusterDefinition = ...;

//The API works with json, convert the cluster-definition into json
String json = api.yamlToJson(ymlString);

//Make sure your ssh keys are available, if not let API generate it for
SshKeyPair sshKeys = api.loadSshKeysIfExist("");
if (sshKeys == null) {
  sshKeys = api.generateSshKeysAndUpdateConf(clusterName);
}

//Register your ssh keys, thats the way of confirming your ssh-keys
api.registerSshKeys(sshKeys);

//Check if your credentials for AWS (or any other cloud) already exist␣
↪otherwise register them
Ec2Credentials credentials = api.loadEc2CredentialsIfExist();
api.updateEc2CredentialsIfValid(credentials);

//Now you can start your cluster by giving json representation of your␣
↪cluster
api.startCluster(json);

//You can always check status of your cluster by running the "status"␣
↪command through the API
//Run status in some time-intervals to see updates for your cluster
long ms1 = System.currentTimeMillis();
int mins = 0;
while (ms1 + 24 * 60 * 60 * 1000 > System.currentTimeMillis()) {
  mins++;
  System.out.println(api.processCommand("status").getResult());
  Thread.currentThread().sleep(60000);
}
```

This code block will print out your cluster status to the console every minute.

2.2 Launching an Apache Hadoop Cluster with Karamel

A cluster definition file is shown below that defines a Apache Hadoop V2 cluster to be launched on AWS/EC2. If you click on `Menu->Load Cluster Definition` and open this file, you can then proceed to launch this Hadoop cluster by entering your AWS credentials and selecting or generating an Open Ssh keypair.

The cluster defintion includes a cookbook called 'hadoop', and recipes for HDFS' NameNode (**nn**) and DataNodes (**dn**), as well as YARN's ResourceManager (**rm**) and NodeManagers (**nm**) and finally a recipe for the MapReduce JobHistoryService (**jhs**). The **nn**, **rm**, and **jhs** recipes are included in a single group called 'metadata' group, and a single node will be created (size: 1) on which all three services will be installed and configured. On a second group (the datanodes group), **dn** and **nm** services will be installed and configured. They will will be installed on two nodes (size: 2). If you want more instances of a particular group, you simply increase the value of the size attribute, (e.g., set "size: 100" for the datanodes group if you want 100 data nodes and resource managers for Hadoop). Finally, we parameterize this cluster deployment with version 2.7.1 of Hadoop (attr -> hadoop -> version). The attrs section is used to supply parameters that are fed to chef recipes during installation.

```
name: ApacheHadoopV2
ec2:
    type: m3.medium
    region: eu-west-1
cookbooks:
  hadoop:
    github: "hopshadoop/apache-hadoop-chef"
    version: "v0.1"
attrs:
  hadoop:
    version: 2.7.1
groups:
  metadata:
    size: 1
    recipes:
        - hadoop::nn
        - hadoop::rm
        - hadoop::jhs
  datanodes:
    size: 2
    recipes:
        - hadoop::dn
        - hadoop::nm
```

The cluster definition file also includes a cookbooks section. Github is our artifact server. We only support the use of cookbooks in our cluster definition file that are located on GitHub. Dependent cookbooks (through Berkshelf) may also be used (from Opscode's repository, Chef supermarket or GitHub), but the cookbooks referenced in the YAML file must be hosted on GitHub. The reason for this is that the Karamel runtime uses Github APIs to query cookbooks for configuration parameters, available recipes, dependencies (Berksfile) and orchestration rules (defined in a Karamelfile). The set of all Karamelfiles for all services is used to build a directed-acyclic graph (DAG) of the installation order for recipes. This allows for modular development and automatic composition of cookbooks into cluster, where each cookbook encapsulates its own orchestration rules. In this way, deployment modules for complicated distributed systems can be developed and tested incrementally, where each service defines its own independent deployment model in Chef and Karamel, and independet deployment modules can be automatically composed into clusters in cluster definition files. This approach supports an incremental test and development model, helping improve the quality of deployment software.

2.3 Designing an experiment with Karamel/Chef

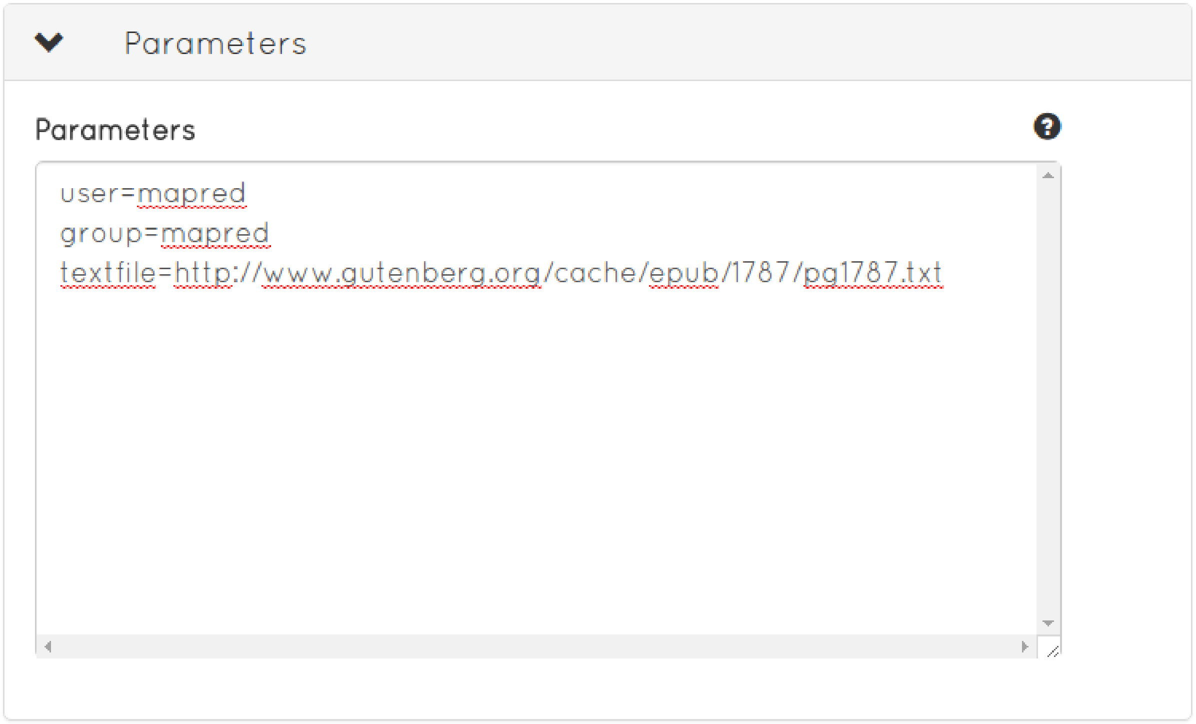
An experiment in Karamel is a cluster definition file that contains a recipe defining the experiment. As such, an experiment requires a Chef cookbook and recipe, and writing Chef cookbooks and recipes can be a daunting prospect for even experienced developers. Luckily, Karamel provides a UI that can take a bash script or a python program and generate a `karamelized` Chef cookbook with a Chef recipe for the experiment. The Chef cookbook is automatically uploaded to a GitHub repository that Karamel creates for you. You recipe may have dependencies on other recipes. For example, a MapReduce experiment defined on the above cluster should wait until all the other services have started before it runs. On examination of the Karamelfile for the hadoop cookbook, we can see that `hadoop::jhs` and `hadoop::nm` are the last services to start. Our MapReduce experiment can state in the Karamelfile that it should start after the `hadoop::jhs` and `hadoop::nm` services have started at all nodes in the cluster.

Experiments also have parameters and produce results. Karamel provides UI support for users to enter parameter values in the `Configure` menu item. An experiment can also download experiment results to your desktop (the Karamel client) by writing to the filename `/tmp/<cookbook>__<recipe>.out`. For detailed information on how to design experiments, go to *experiment designer*

2.4 Designing an Experiment: MapReduce Wordcount

This experiment is a wordcount program for MapReduce that takes as a parameter an input textfile as a URL. The program counts the number of occurances of each word found in the input file. First, create a new experiment called *mapred* in GitHub (any organization). You will then need to click on the `advanced` tickbox to allow us to specify dependencies and parameters. .. We keep them separate experiments to measure their time individually.

```
user=mapred
group=mapred
textfile=http://www.gutenberg.org/cache/epub/1787/pg1787.txt
```



```
user=mapred
group=mapred
textfile=http://www.gutenberg.org/cache/epub/1787/pg1787.txt
```

Fig. 2: Defining the texfile input parameter. Parameters are key-value pairs defined in the Parameter box.

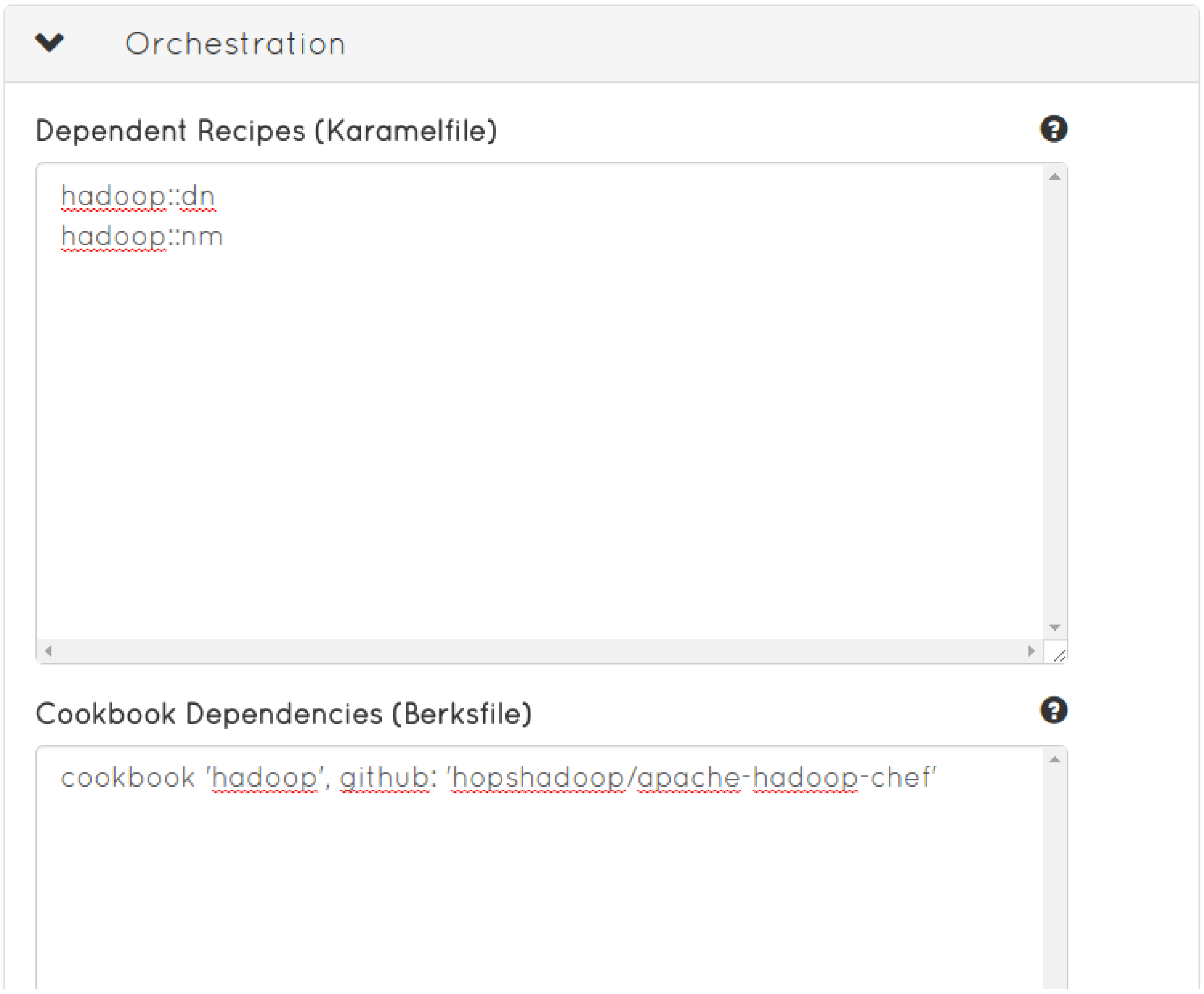
The code generator bash script must wait until all HDFS datanodes and YARN nodemanagers are up and running before it is run. To indicate this, we add the following lines to Dependent Recipes textbox:

```
hadoop::dn
hadoop::nm
```

Our new cookbook will be dependent on the hadoop cookbook, and we have to enter into the `Cookbook Dependencies` textbox the relative path to the cookbook on GitHub:

```
cookbook 'hadoop', github: 'hopshadoop/apache-hadoop-chef'
```

The following code snippet runs MapReduce wordcount on the input parameter `textfile`. The parameter is referenced in the bash script as `#{node.mapred.textfile}`, which is a combination of `node.`‘‘<cookbookname>‘‘.‘‘<parameter>‘‘.



Dependent Recipes (Karamelfile)

```
hadoop::dn
hadoop::nm
```

Cookbook Dependencies (Berksfile)

```
cookbook 'hadoop', github: 'hopshadoop/apache-hadoop-chef'
```

Fig. 3: Define the Chef `cookbook dependencies` as well as the `dependent recipes`, the recipes that have to start before the experiments in this cookbook.

CHAPTER 3

Web-App

Karamel provides a web-based UI and is a lightweight standalone application that runs on user machines, typically desktops. The user interface it has three different perspectives: board, terminal and experiment designer.

3.1 Board-UI

The `Board` is the landing page that appears in your browswer when you start Karamel. The `Board` is a view on a cluster definition file that you load. You can modify the cluster using the UI (adding/removing recipes, entering parameter values, save updated cluster definitions) and run the cluster definition from the UI. This way, inexperienced users can launch clusters without needing to read cluster definitions in YAML.

3.1.1 Load Cluster Definition

Click on the menu item, and then click on `Load Cluster Defn`:

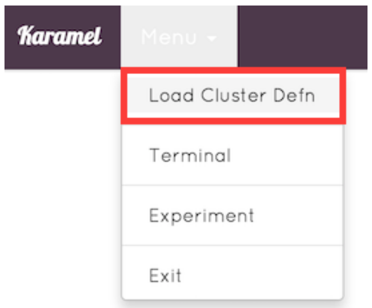


Fig. 1: Load Cluster Definition.

Lists are shown in the board perspective of the UI, where each list represents a group of machines that install the same application stack. At the top of each list you see the group-name followed by the number of machines in that group (in parentheses). Each list consists of a set of cards, where each card represents a service (Chef recipe) that will be installed on all the machines in that group. Chef recipes are programs written in Ruby that contain instructions for how to install and configure a piece of software or run an experiment.

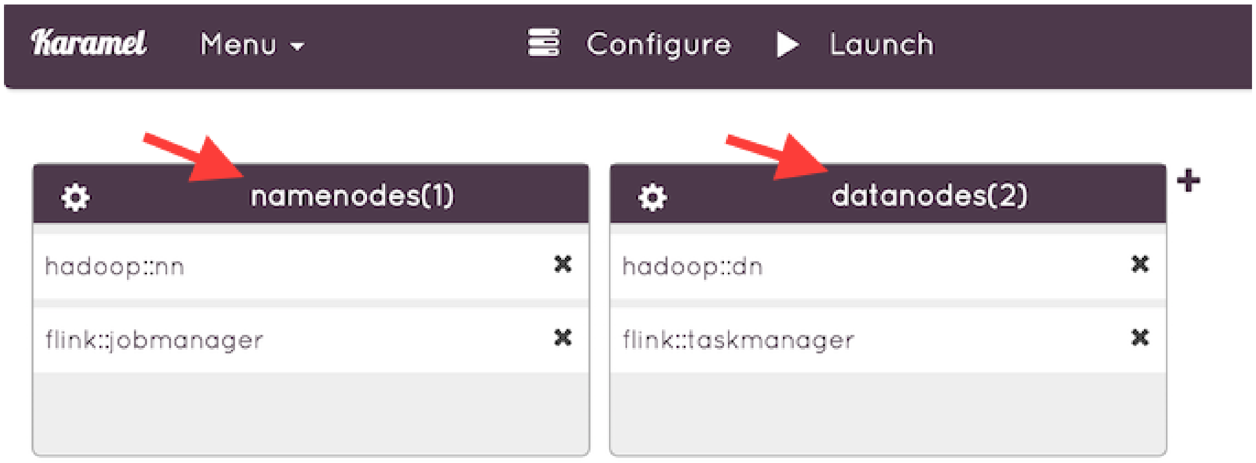
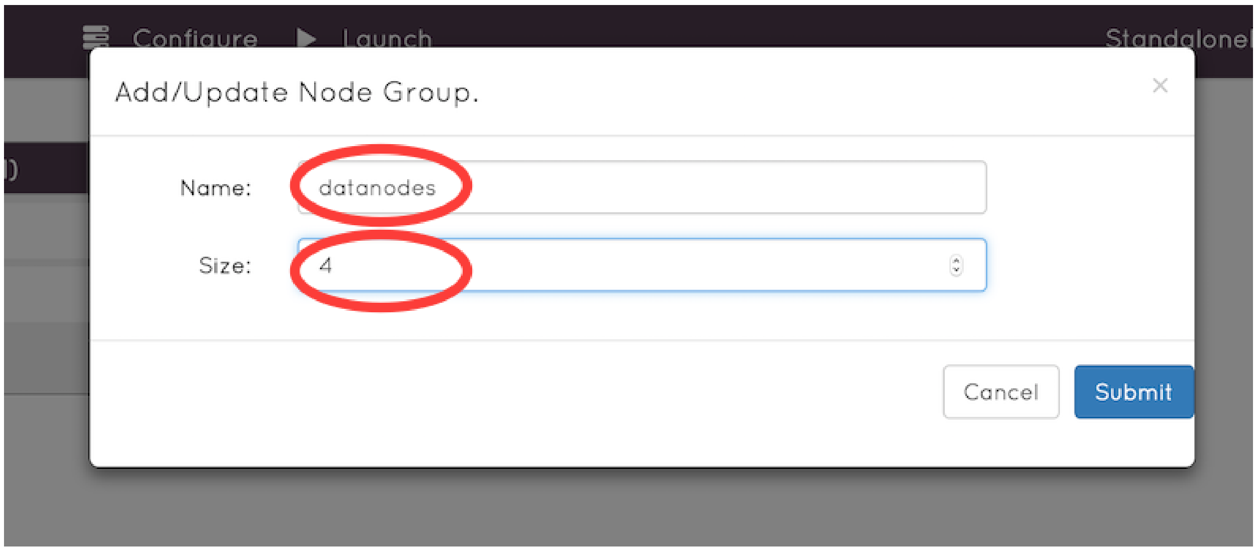


Fig. 2: Lists of cards in Karamel. Each card is a Chef recipe.

3.1.2 Change group name and size

To change the GroupName and/or number of machines in each group, double click on the header of the group. In the following dialog, you can make your changes and submit them (to indicate your are finished).



Name: datanodes

Size: 4

Fig. 3: Changing the number of nodes in a NodeGroup

3.1.3 Add a new recipe to a group

In the top-left icon in the header of each group, there is a menu to update the group. Select the `Add recipe` menu item:

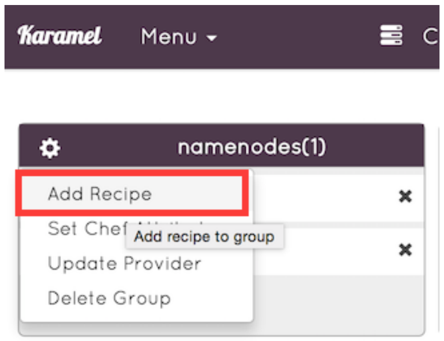
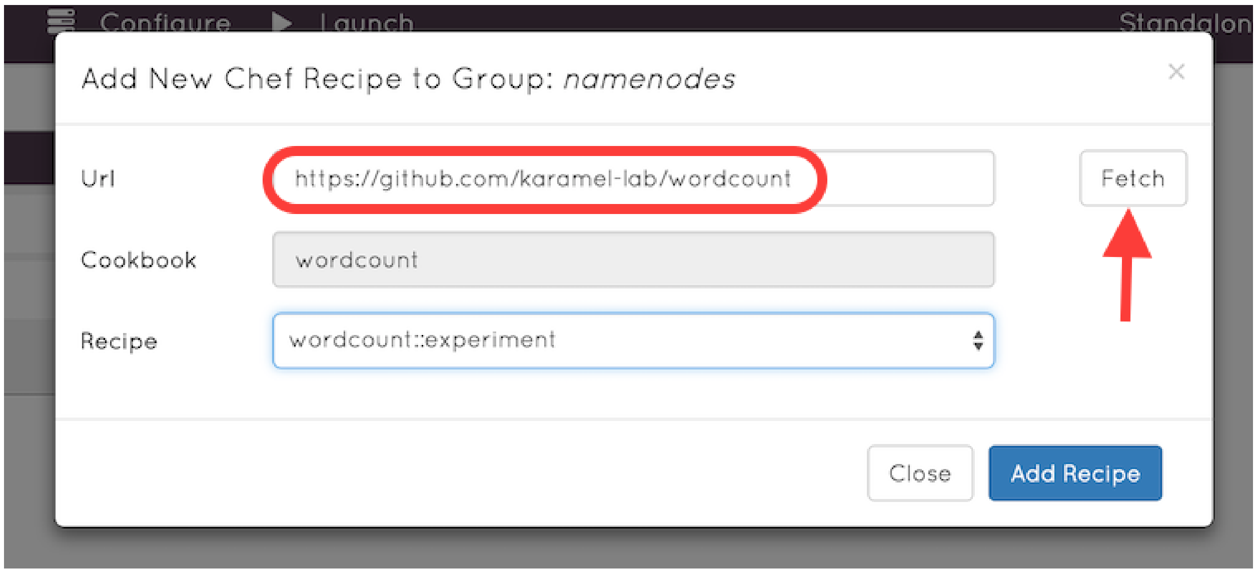


Fig. 4: Adding a Chef recipe to a node group.

In order to add a recipe to a group, you must enter the GitHub URL for the (karamelized) Chef cookbook where your recipe resides, and then press `fetch` to load available recipes from the cookbook. Choose your recipe from the combo-box below:



Add New Chef Recipe to Group: *namenodes*

Url: https://github.com/karamel-lab/wordcount

Cookbook: wordcount

Recipe: wordcount::experiment

Fig. 5: Adding a Chef recipe from a GitHub cookbook to a node group.

3.1.4 Customize Chef attributes for a group

Parameters (Chef attributes) can be entered within the scope of a NodeGroup: group scope values have higher precedence than (override) global cluster scope values. To update chef attributes for a group, select its menu item from the group settings menu:

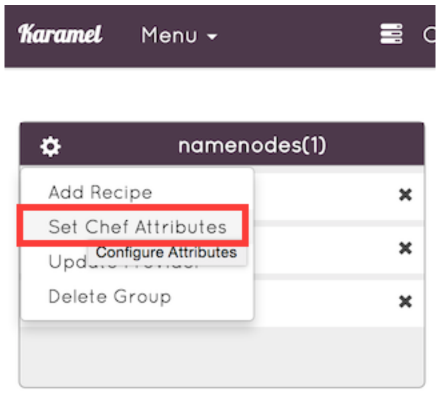


Fig. 6: Updating Chef Attributes at the Group level.

In the dialog below, there is a tab per used cookbook in that group, in each tab you see all customizable attributes, some of them are mandatory and some optional with some default values. Users must set a value for all of the mandatory attributes (or accept the default value, if one is given).

3.1.5 Customize cloud provider for a group

Cluster definition files support the use of multiple (different) cloud providers within the same cluster definition. Each group can specify its own cloud provider. This way, we can support multi-cloud deployments. Like attributes, cloud provider settings at the NodeGroup scope will override cloud provider settings at the global scope. Should you have multi-cloud settings in in your cluster, at launch time you must supply credentials for each cloud separately in the launch dialog.

Choose the cloud provider for the current group then you will see moe detailed settings for the cloud provider.

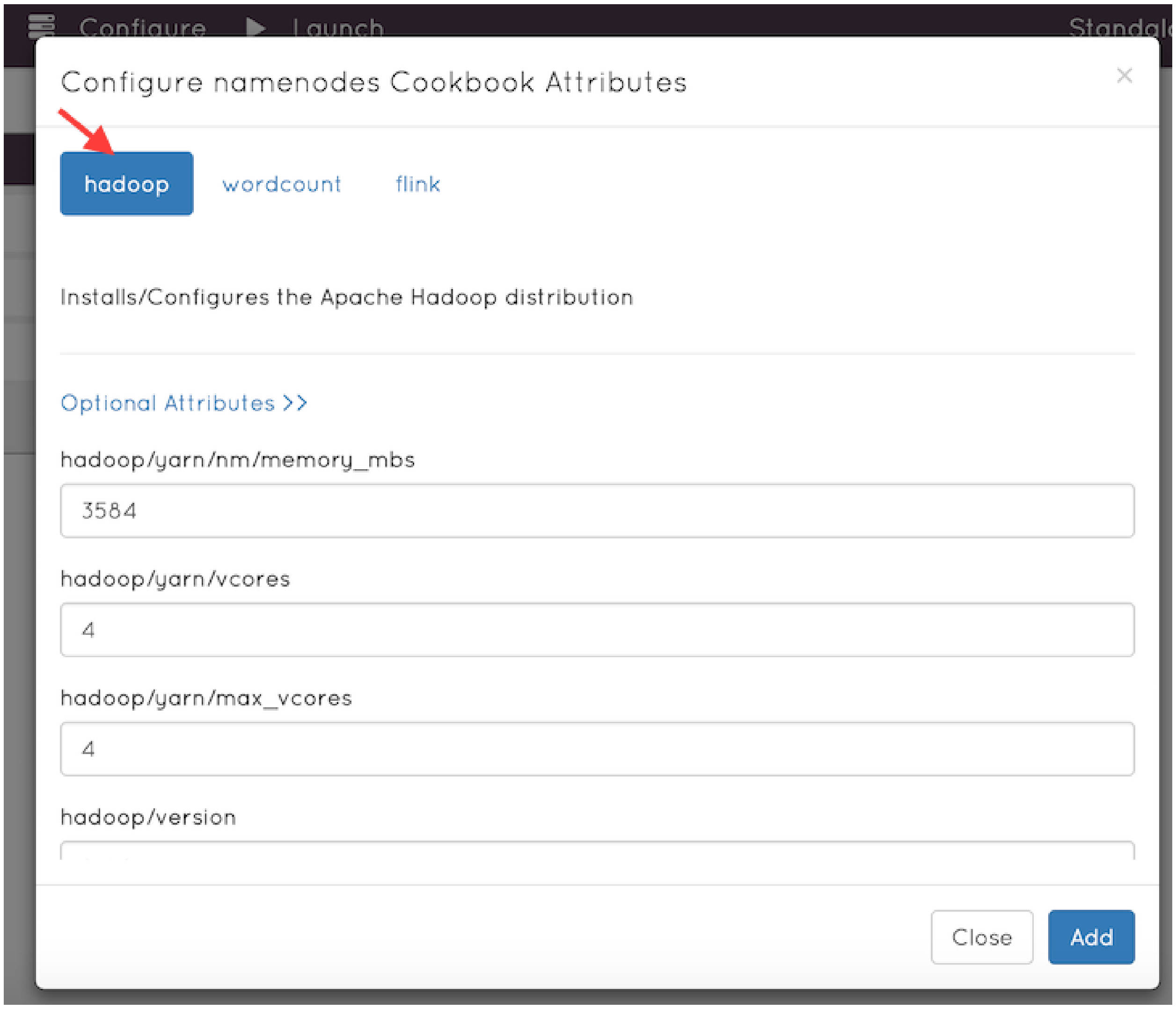
3.1.6 Delete a group

If you want to delete a group find the menu-item in the group menu.

Once you delete a group the list and all the settings related to that group will be disappeared forever.

3.1.7 Update cluster-scope attributes

When you are done with your group settings you can have some global values for Chef attributes. By choosing Configure button in the middle of the top bar a configuration dialog will pop up, there you see several tabs each named after one used chef-cookbook in the cluster definition. Those attributes are pre-built by cookbook designers for run-time customization. There are two types of attributes mandatory and optional - most of them usually have a default value but if they don't, the user must fill in mandatory values to be able to proceed.



Configure namenodes Cookbook Attributes

Installs/Configures the Apache Hadoop distribution

hadoop/yarn/nm/memory_mbs: 3584

hadoop/yarn/vcores: 4

hadoop/yarn/max_vcores: 4

hadoop/version

Fig. 7: Entering attribute values to customize service.

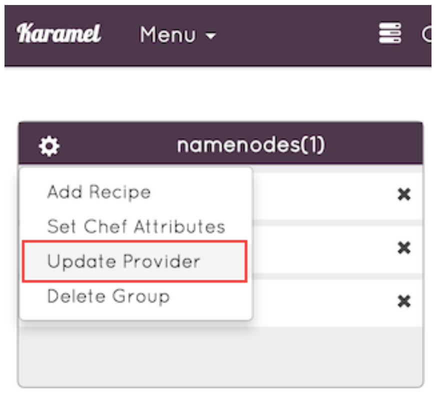
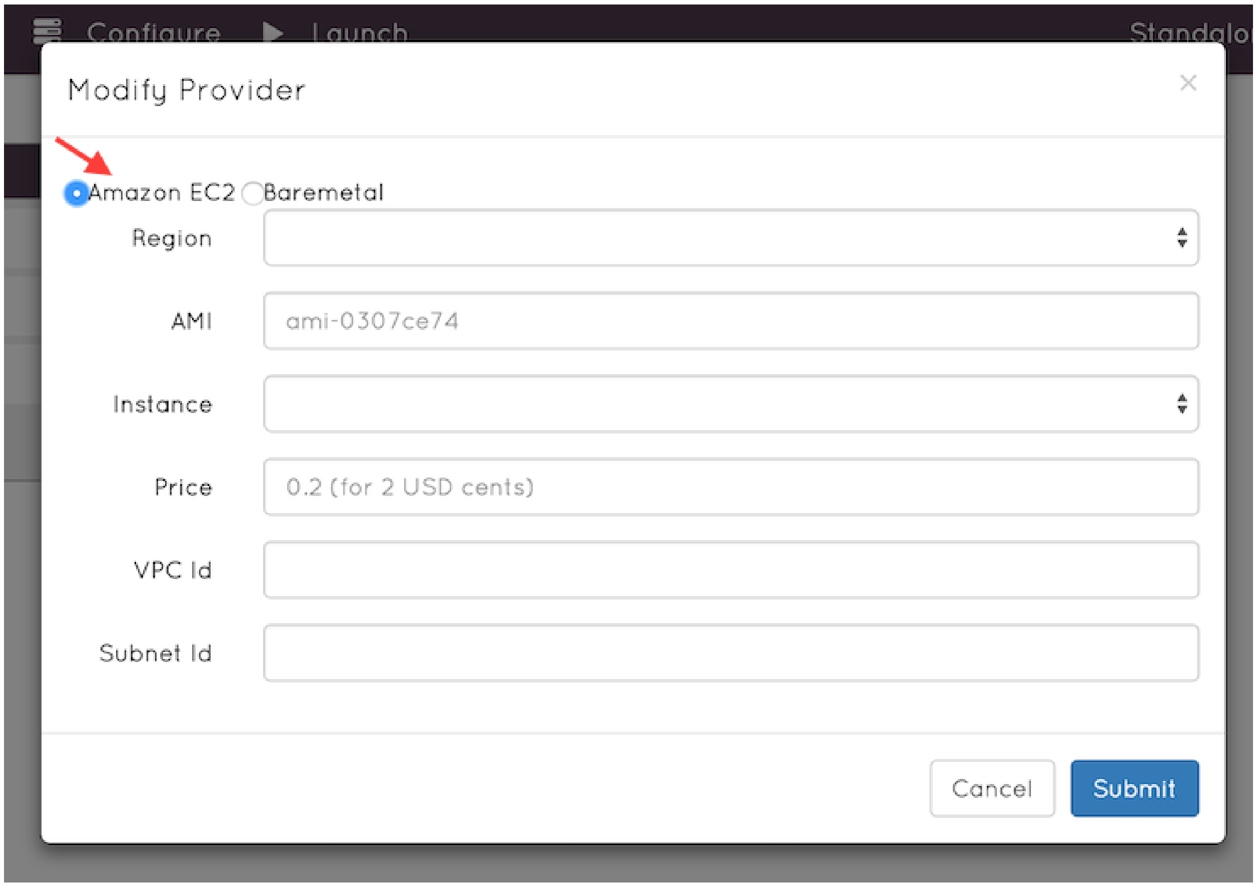


Fig. 8: Multi-cloud deployments are supported by specifying different cloud providers for different node groups.



Amazon EC2 Baremetal

Region	
AMI	ami-0307ce74
Instance	
Price	0.2 (for 2 USD cents)
VPC Id	
Subnet Id	

Fig. 9: Configuring a cloud provider per Node Group.

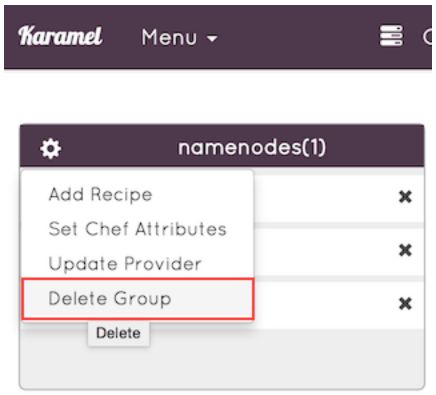
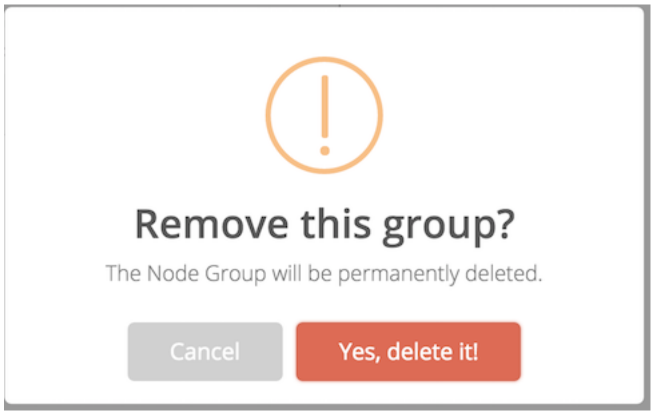


Fig. 10: Delete a Node Group.


Remove this group?

The Node Group will be permanently deleted.

Fig. 11: Delete Confirmation.

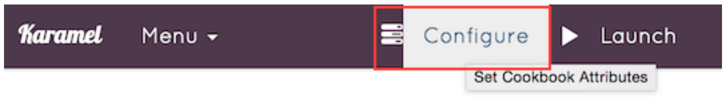


Fig. 12: To fill in optional and mandatory attributes.

By default each cookbook has a parameter for the operating system's user-name and group-name. It is recommended to set the same user and group for all cookbooks that you don't face with permission issues.

It is also important to fine-tune your systems with the right parameters, for instance according to type of the machines in your cluster you should allocate enough memory to each system.

3.1.8 Start to Launch Cluster

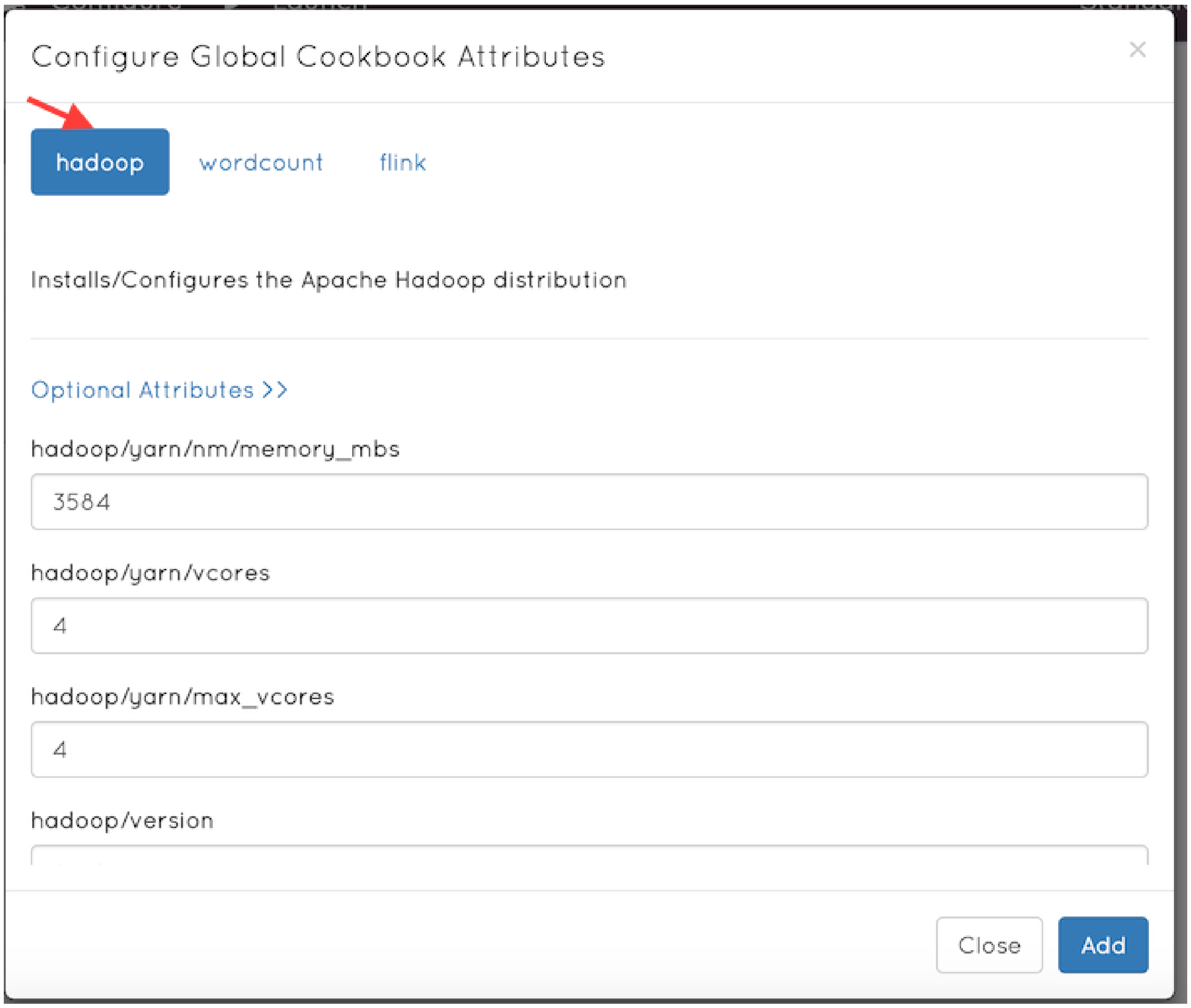
Finally you have to launch your cluster by pressing launch icon in the top bar. There exist a few tabs that user must go through all of them, you might have to specify values and confirm everything. Even though Karamel caches those values, you have to always confirm that Karamel is allowed to use those values for running your cluster.

3.1.9 Set SSH Keys

In this step first you need to specify your ssh key pair - Karamel uses that to establish a secure connection to virtual machines. For Linux and Mac operating systems, Karamel finds the default ssh key pair in your operating system and will use it.

3.1.10 Generate SSH Key

If you want to change the default ssh-key you can just check the advance box and from there ask Karamel to generate a new key pair for you.



Installs/Configures the Apache Hadoop distribution

Optional Attributes >>

hadoop/yarn/nm/memory_mbs

3584

hadoop/yarn/vcores

4

hadoop/yarn/max_vcores

4

hadoop/version

Fig. 13: Filling in optional and mandatory attributes.

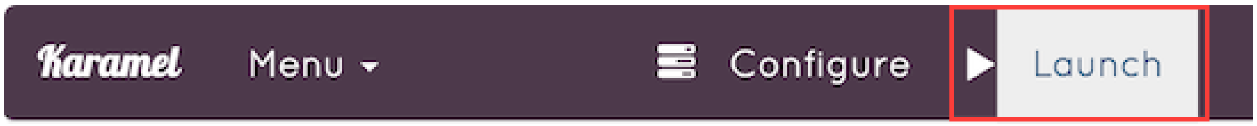
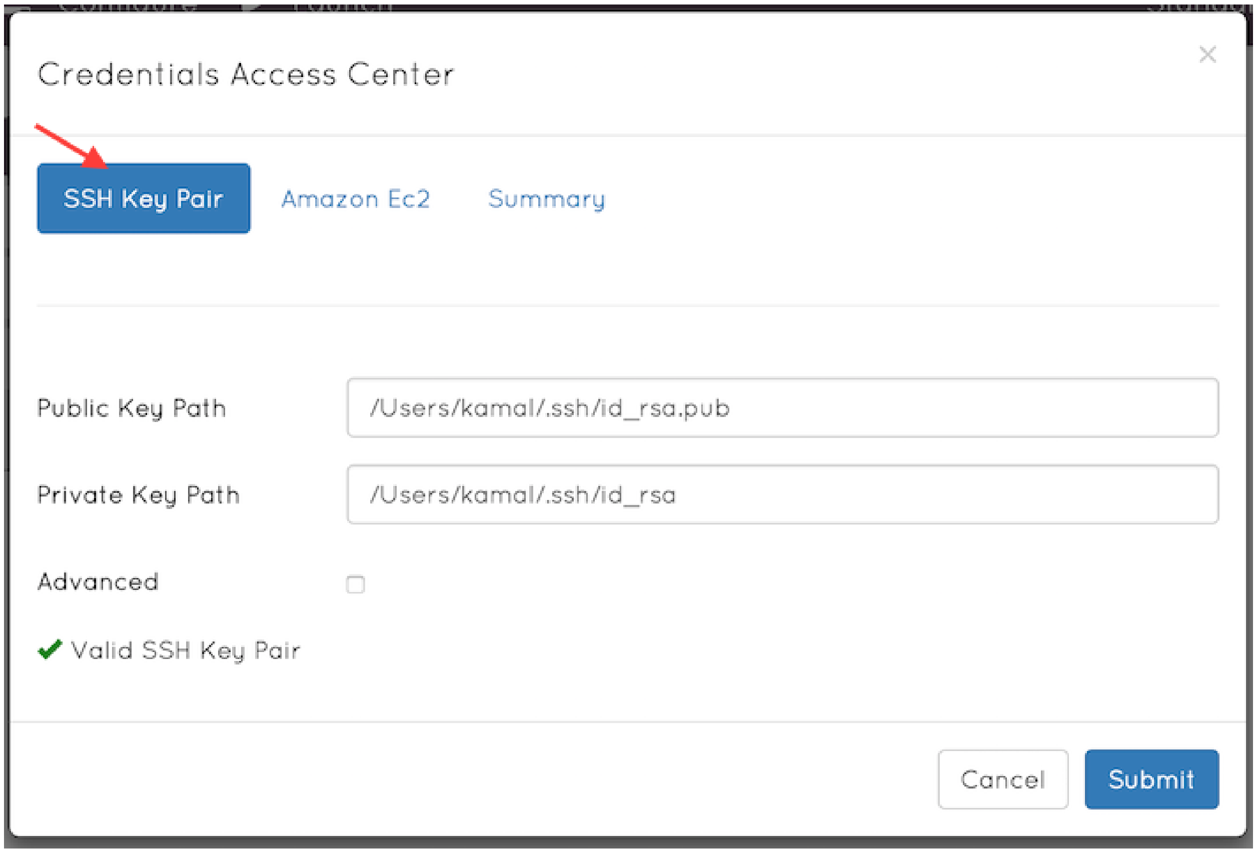


Fig. 14: Launch Button.

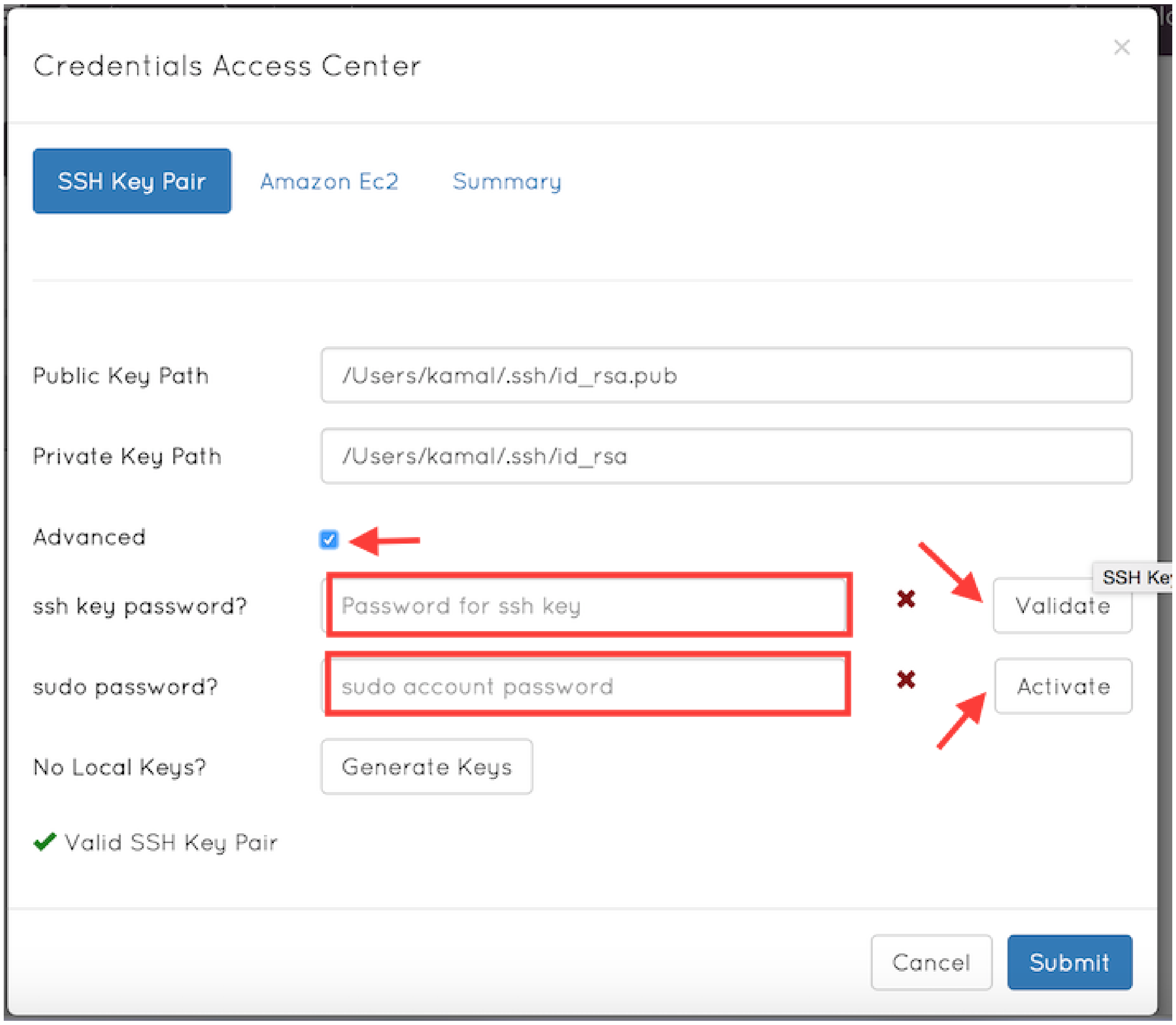


Public Key Path: /Users/kamal/.ssh/id_rsa.pub
Private Key Path: /Users/kamal/.ssh/id_rsa
Advanced ☐
✓ Valid SSH Key Pair

Fig. 15: SSH key paths.

3.1.11 Password Protected SSH Keys

If your ssh key is password-protected you need to enter your password in the provided box, and also in case you use bare-metal (karamel doesn't fork machines from cloud) you have to give sudo-account access to your machines.


Credentials Access Center

SSH Key Pair | Amazon Ec2 | Summary

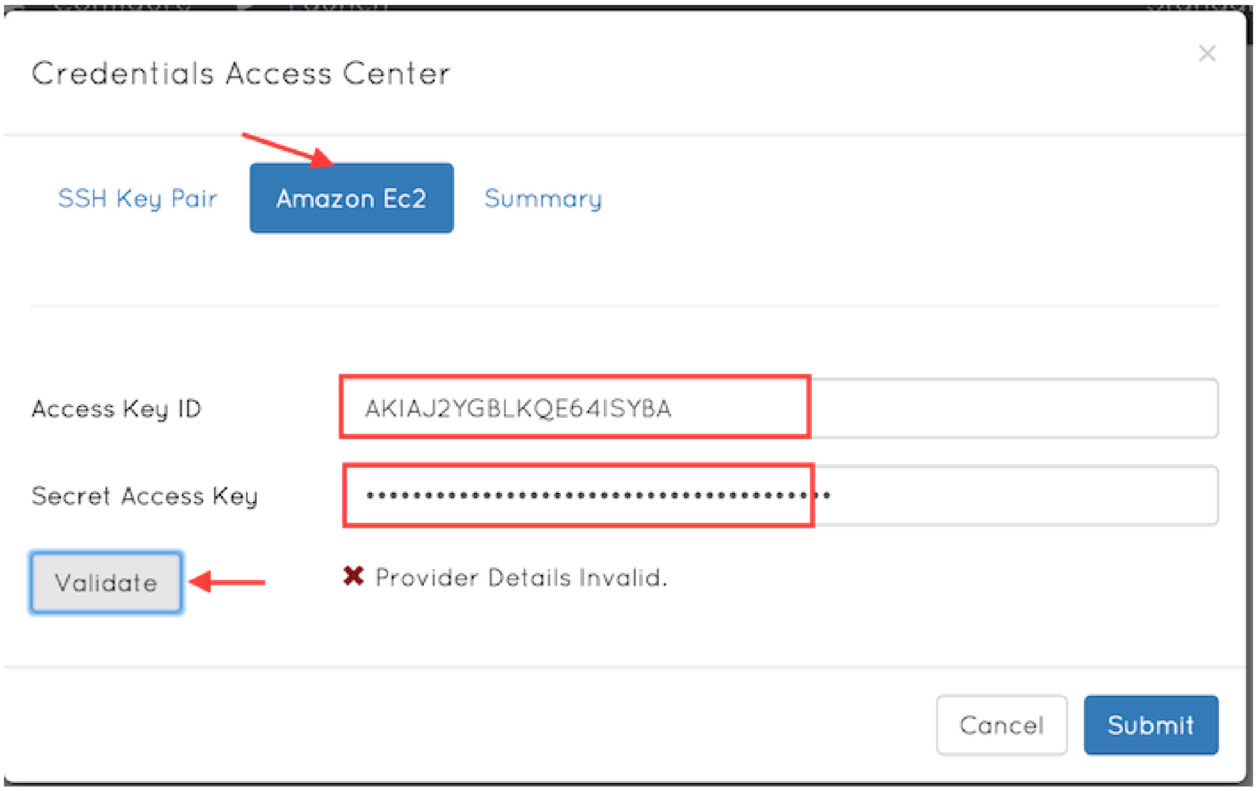
Public Key Path	/Users/kamal/.ssh/id_rsa.pub
Private Key Path	/Users/kamal/.ssh/id_rsa
Advanced	✓
ssh key password?	Password for ssh key
sudo password?	sudo account password
No Local Keys?	Generate Keys

✓ Valid SSH Key Pair

Fig. 16: Advanced options for SSH keys.

3.1.12 Cloud Provider Credentials

In the second step of launch you need to give credentials for accessing the cloud of your choice. If your cluster is running on a single cloud a tab related to that cloud will appear in the launch dialog and if you use multi-cloud a separate tab for each cloud will appear. Credentials are usually in different formats for each cloud, for more detail information please find it in the related cloud section.



Access Key ID: AKIAJ2YGBLKQE64ISYBA

Secret Access Key: ••

✖ Provider Details Invalid.

Fig. 17: Provider-specific credentials.

3.1.13 Final Control

When you have all the steps passed in the summary tab you can launch your cluster, it will bring you to the *terminal* there you can control the installation of your cluster.

3.2 Karamel Terminal

The terminal perspective enables user to monitor and manage running clusters as well as debugging Chef recipes by running them.

3.2.1 Open Terminal

The Board-UI redirects to the terminal as soon as a cluster launches. Another way to access the terminal is by clicking on the `terminal` menu-item from the menu dropdown list, as shown in the screen-shot below.

3.2.2 Button Bar

The Terminal has a menu bar in which the available menu items (buttons) change dynamically based on the active page.

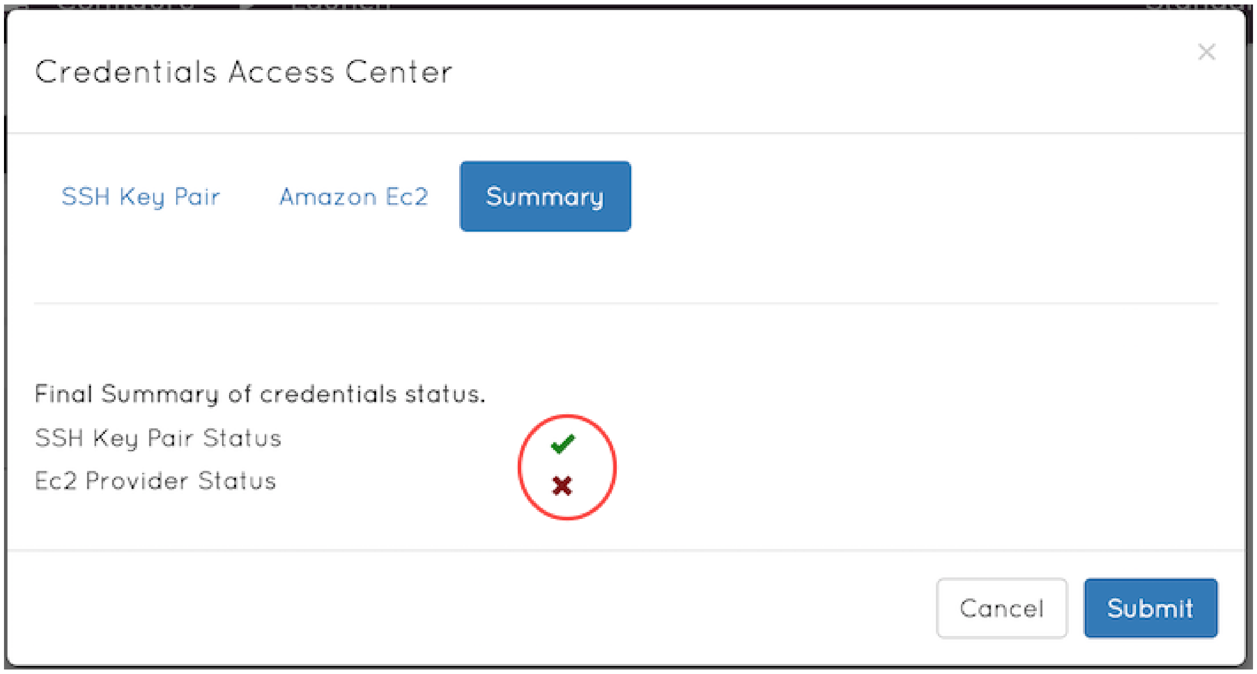


Fig. 18: Validity summary for keys and credentials.

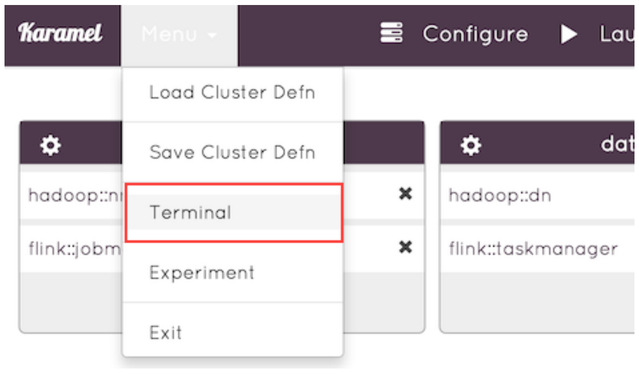


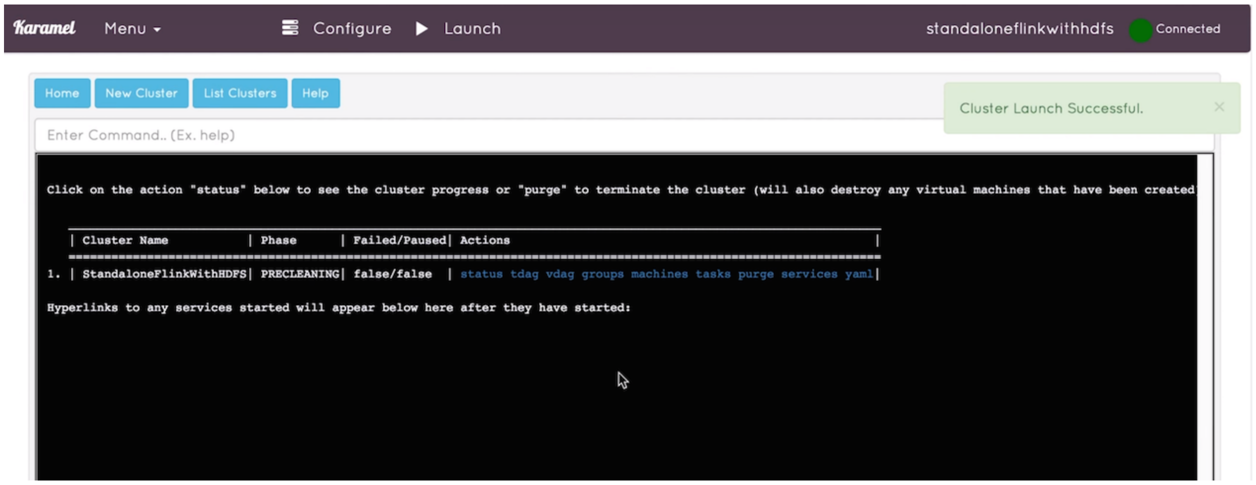
Fig. 19: Selecting the terminal perspective.

3.2.3 Command Bar

Under the menu bar, there is a long text area where you can execute commands directly. The buttons (menu items) are, in fact, just widely used commands. To see list of commands click on the Help button.

3.2.4 Main Page

The main page in the terminal shows available running clusters - you can run multiple clusters at the same time they just need to have different names - where you see general status of your cluster. There are some actions in front of each cluster where you can obtain more detail information about each cluster.



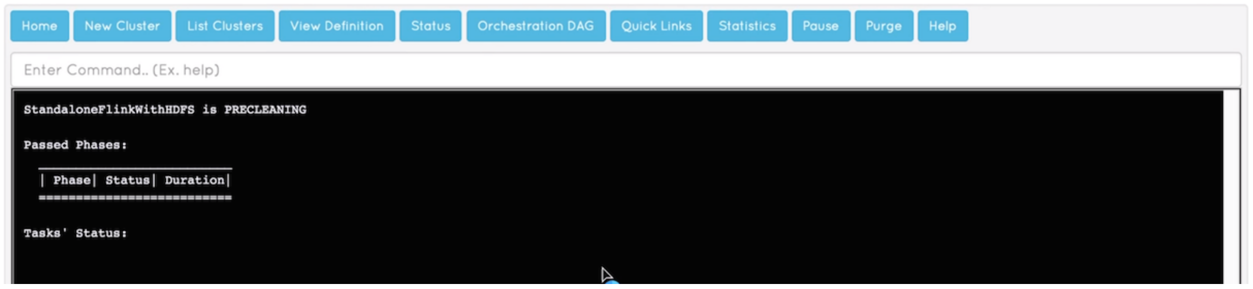
Cluster Launch Successful.

```
Click on the action "status" below to see the cluster progress or "purge" to terminate the cluster (will also destroy any virtual machines that have been created

   | Cluster Name           | Phase      | Failed/Paused| Actions                                                 |
   ======================================================================================================
1. | StandaloneFlinkWithHDFS| PRECLEANING| false/false  | status tdag vdag groups machines tasks purge services yaml|

Hyperlinks to any services started will appear below here after they have started:
```

Fig. 20: Successful launch redirects to terminal page.

3.2.5 Cluster Status

Status page pushes the new status for the chosen cluster very often. In the first table you see phases of the cluster and each of them they passed successfully or not.



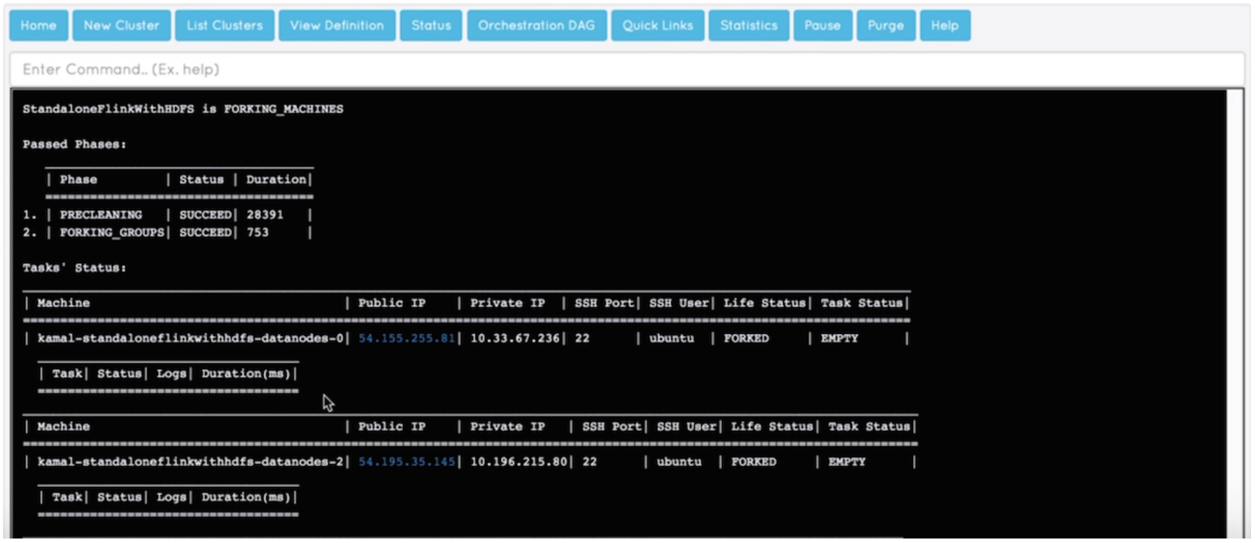
```
StandaloneFlinkWithHDFS is PRECLEANING

Passed Phases:

  | Phase| Status| Duration|
  ==========================

Tasks' Status:
```

Fig. 21: Cluster Status - A recently started cluster

The cluster deployment phases are: #1. Pre-Cleaning #2. Forking Groups #3. Forking Machines #4. Installing

As soon as the cluster passes the forking groups phase, a list of machine tables appear under the phases table. Each machine table indicates that the virtual machine (VM) has been forked and some details on the VM are available, such as its IP Addresses (public and private) and its connection status.

Inside each machine table there exists a smaller table for showing the tasks that are going to be submitted into that machine. Before all machines became forked and ready, all task tables are empty for all machines.



```
StandaloneFlinkWithHDFS is FORKING_MACHINES

Passed Phases:

   | Phase         | Status | Duration|
   ===================================
1. | PRECLEANING   | SUCCEED| 28391   |
2. | FORKING_GROUPS| SUCCEED| 753     |

Tasks' Status:

| Machine                              | Public IP     | Private IP   | SSH Port| SSH User| Life Status| Task Status|
=====================================================================================================================
| kamal-standaloneflinkwithhdfs-datanodes-0| 54.155.255.81| 10.33.67.236| 22      | ubuntu  | FORKED     | EMPTY      |

  | Task| Status| Logs| Duration(ms)|
  ==================================

| Machine                              | Public IP     | Private IP    | SSH Port| SSH User| Life Status| Task Status|
======================================================================================================================
| kamal-standaloneflinkwithhdfs-datanodes-2| 54.195.35.145| 10.196.215.80| 22      | ubuntu  | FORKED     | EMPTY      |

  | Task| Status| Logs| Duration(ms)|
  ==================================
```

Fig. 22: Cluster Status - Forking Machines

Once all machines have started forking tasks, a list of tasks are displayed for each machine. The Karamel Scheduler orders tasks and decides when each task is ready to be run. The scheduler assigns a status label to each task.

The task status labels are:

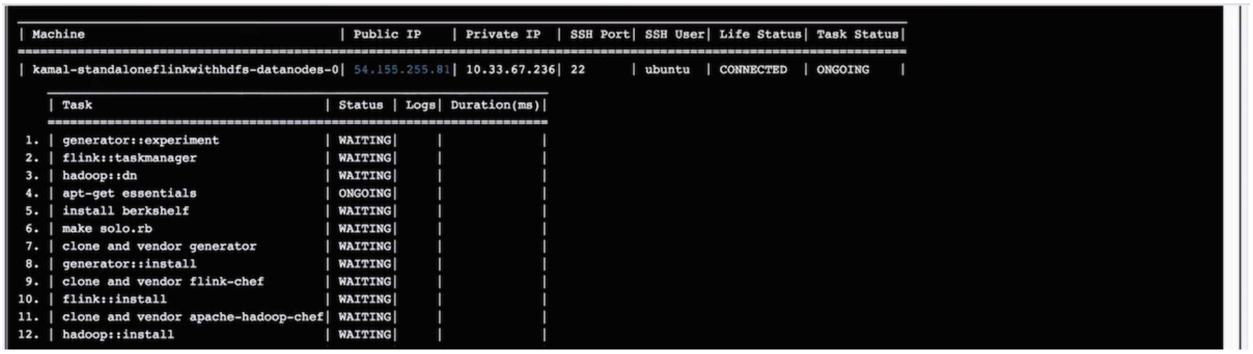
- Waiting: the task is still waiting until its dependencies have finished;
- Ready: the task is ready to be run but the associated machine has not yet taken it yet because it is running another task;
- Ongoing: the task is currently running on the machine;
- Succeed: the task has finished successfully;
- Failed: the task has failed - each failure will be propagated up into cluster and will cause the cluster to pause the installation.

When a task is finished a link to its log content will be displayed in the third column of task table. The log is the merged content of the standard output and standard error streams.

3.2.6 Orchestartion DAG

The scheduler in Karamel builds a Directed Acyclic Graph (DAG) from the set of tasks in the cluster. In the terminal perspective, the progress of the DAG execution can be visualized by clicking on the "Orchestration DAG" button.

Each Node of the DAG represents a task that must be run on a certain machine. Nodes dynamically change their color according to the status change of their tasks. Each color is interpreted as follows:



```
| Machine                              | Public IP    | Private IP   | SSH Port| SSH User| Life Status| Task Status|
=====================================================================================================================
| kamal-standaloneflinkwithhdfs-datanodes-0| 54.155.255.81| 10.33.67.236| 22      | ubuntu  | CONNECTED  | ONGOING    |

    | Task                              | Status | Logs| Duration(ms)|
    ================================================================
 1. | generator::experiment             | WAITING|     |             |
 2. | flink::taskmanager                | WAITING|     |             |
 3. | hadoop::dn                        | WAITING|     |             |
 4. | apt-get essentials                | ONGOING|     |             |
 5. | install berkshelf                 | WAITING|     |             |
 6. | make solo.rb                      | WAITING|     |             |
 7. | clone and vendor generator        | WAITING|     |             |
 8. | generator::install                | WAITING|     |             |
 9. | clone and vendor flink-chef       | WAITING|     |             |
10. | flink::install                    | WAITING|     |             |
11. | clone and vendor apache-hadoop-chef| WAITING|     |             |
12. | hadoop::install                   | WAITING|     |             |
```

Fig. 23: Cluster Status - Installing.

- Blue: Waiting
- Ready: Yellow
- Ongoing: Blinking orange
- Succeed: Green
- Failed: Red

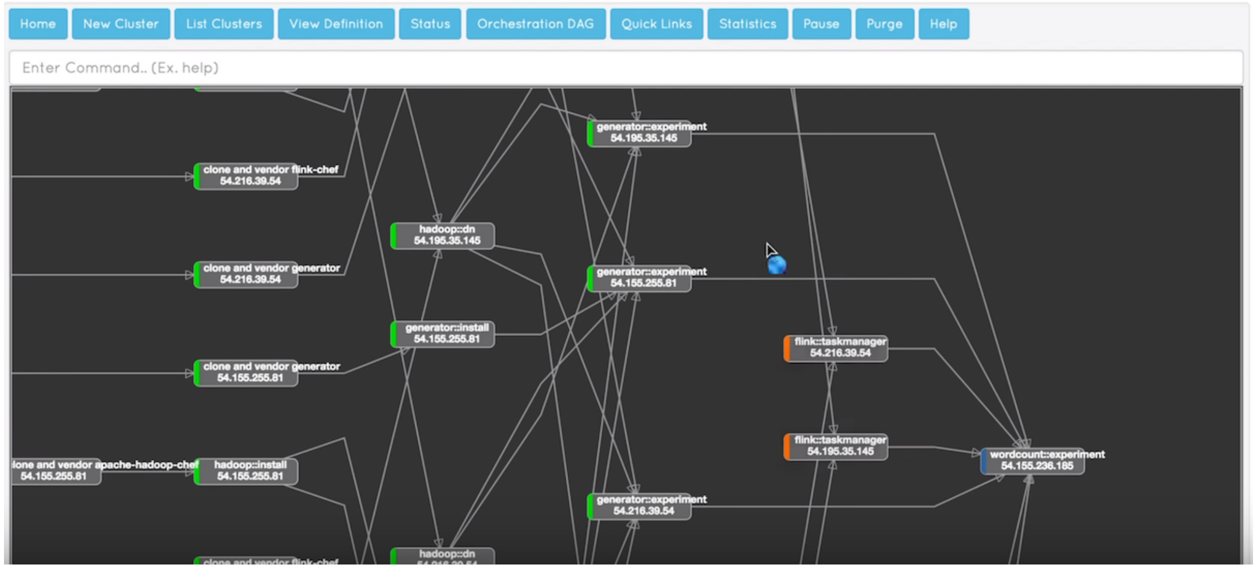
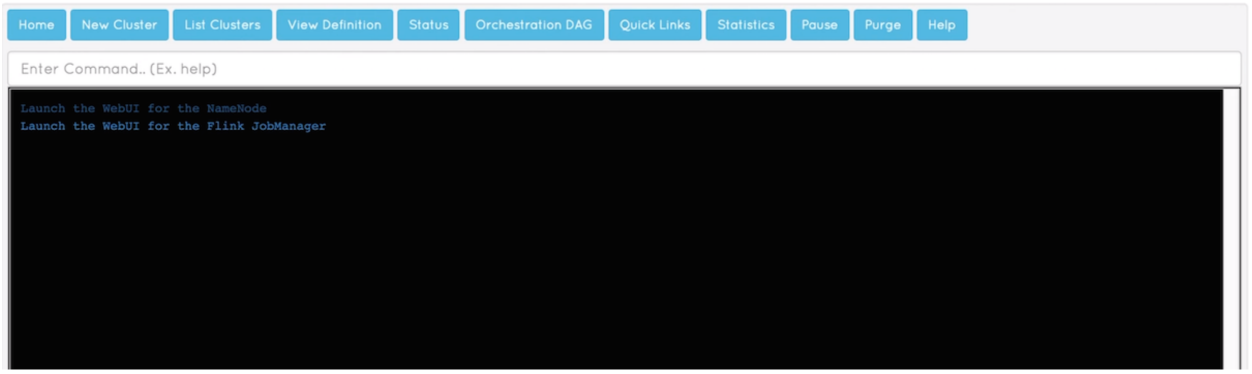

Home
New Cluster
List Clusters
View Definition
Status
Orchestration DAG
Quick Links
Statistics
Pause
Purge
Help
Enter Command.. (Ex. help)
generator::experiment 54.195.35.145
clone and vendor flink-chef 54.216.39.54
hadoop::dn 54.195.35.145
clone and vendor generator 54.216.39.54
generator::experiment 54.155.255.81
generator::install 54.155.255.81
flink::taskmanager 54.216.39.54
clone and vendor generator 54.155.255.81
flink::taskmanager 54.195.35.145
wordcount::experiment 54.155.236.185
lone and vendor apache-hadoop-chef 54.155.255.81
hadoop::install 54.155.255.81
generator::experiment 54.216.39.54
hadoop::dn
clone and vendor flink-chef

Fig. 24: Orchestration DAG

The Orchestration DAG is not only useful to visualize the cluster progress but can also help in debugging the level of parallelization in the installation graph. If some tasks are acting as global barriers during installation, they can be quickly identified by inspecting the DAG and seeing the nodes with lots of incoming edges and some outgoing edges. As have local orchestration rules in their Karamelfiles, the DAG is built from the set of Karamelfiles. It is not easy to manually traverse the DAG, given a set of Karamelfiles, but the visual DAG enables easier inspection of the global order of installation of tasks.

3.2.7 Quick Links

Quick links a facility that Karamel provides in the terminal perspective to access web pages for services in your cluster. For example, when you install Apache Hadoop, you might want to access the NameNode or ResourceManager's web UI. Those links must be designed in karamelized cookbooks (in the `metadata.rb` file). Karamel parses the `metadata.rb` files, extracting the webpage links and displaying them in the *Quick Links* tab.



```
Launch the WebUI for the NameNode
Launch the WebUI for the Flink JobManager
```

Fig. 25: Quick Links

3.2.8 Statistics

Currently Karamel collects information about the duration of all tasks when you deploy a cluster. Duration statistics are available by clicking on statistics button that will show the names of the tasks and their execution time. It might be have you have several instances of each task in your cluster, for example, you may install the `hadoop::dn` recipe on several machines in your cluster - all such instances will appear in the statistics table. Statistics is a good way for performance measurement for some type of experiments. You can just draw a plot on them to show the performance of your experiment.

3.2.9 Pause/Resume

A cluster may pause running either because the user's order or when a failure happens. It is a good way if user wants to change something or if he wants to avoid running the entire cluster for some reason. In that case when you click on the "Pause" button it takes some time until all machines finish their current running task and go into the paused mode. When cluster is paused, a resume button will appear which proceeds running the cluster again.

3.2.10 Terminate

Terminate is a button to destroy and release all the resources both on Clouds and Karamel-runtime, destroying any virtual machines created. It is recommended to use the terminate function via Karamel to clean-up resources rather than doing so manually - Karamel makes sure all ssh connections, local threads, virtual machines and security groups are released completely.

3.3 Experiment Designer

The experiment Designer perspective in Karamel helps you to design your experiment in a bash script or a python program without needing to know Chef or Git. Take the following steps to design and deploy your experiment.

3.3.1 Find experiment designer

When you have Karamel web app up and running, you can access the experiment designer from the Experiment menu-item on the left-hand side of the application.

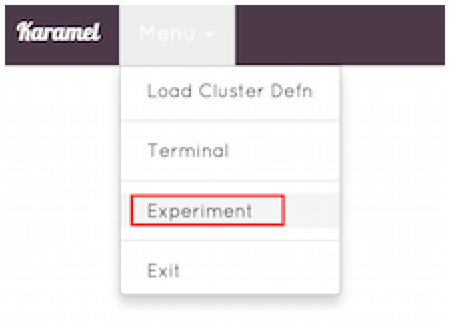


Fig. 26: Get into the experiment designer.

3.3.2 Login into Github

Github is Karamel's artifact server, here you will have to login into your Github account for the first time while Karamel will remember your credentials for other times.

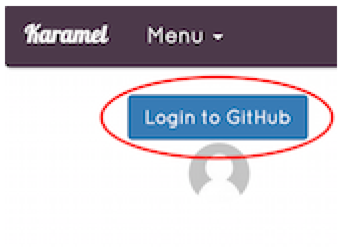


Fig. 27: Login button.

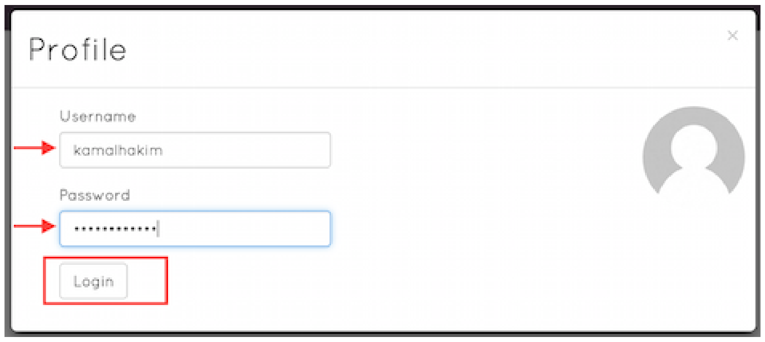


Fig. 28: Github credentials.

3.3.3 Start working on experiment

You can either create a new experiment or alternatively load the already designed experiment into the designer.

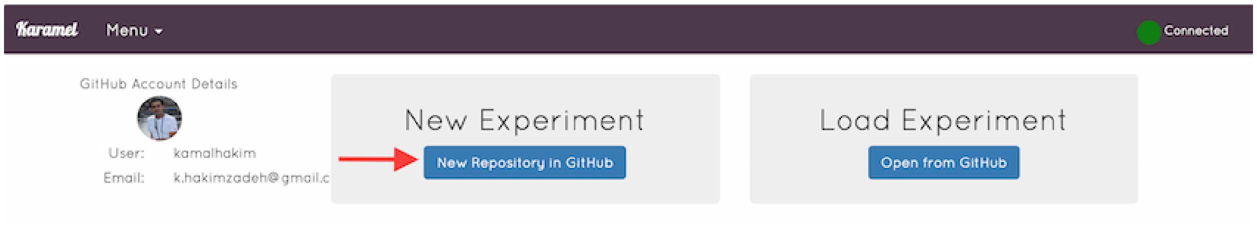
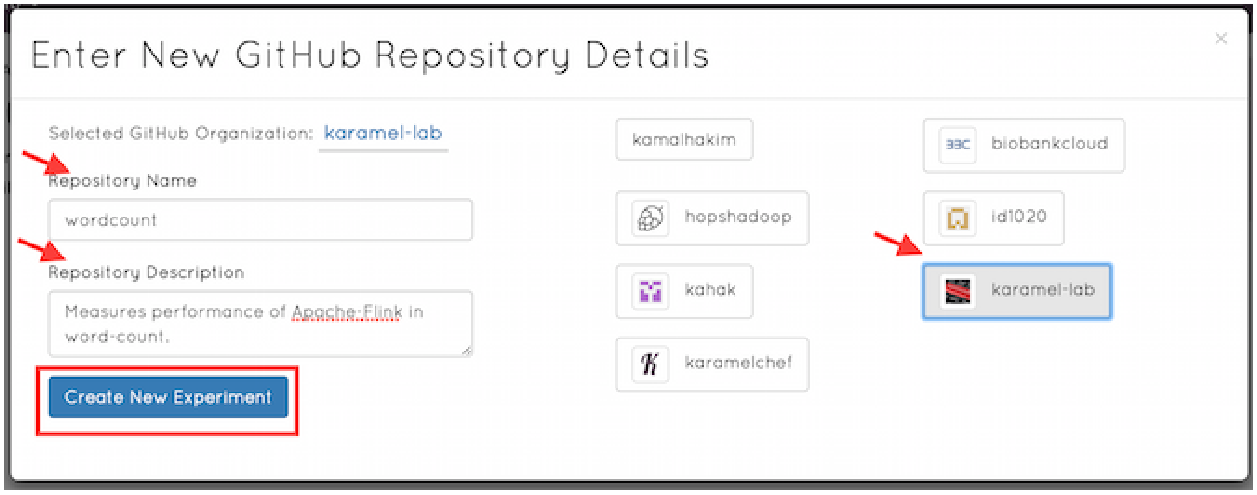


Fig. 29: Work on a new or old experiment.

3.3.4 Create a new experiment

If you choose to create a new experiment you will need to choose a name for it, optionally describe it and choose which Github repo you want to host your experiment in. As you can see in the below image Karamel connects and fetches your available repos from Github.


Enter New GitHub Repository Details

Selected GitHub Organization: karamel-lab

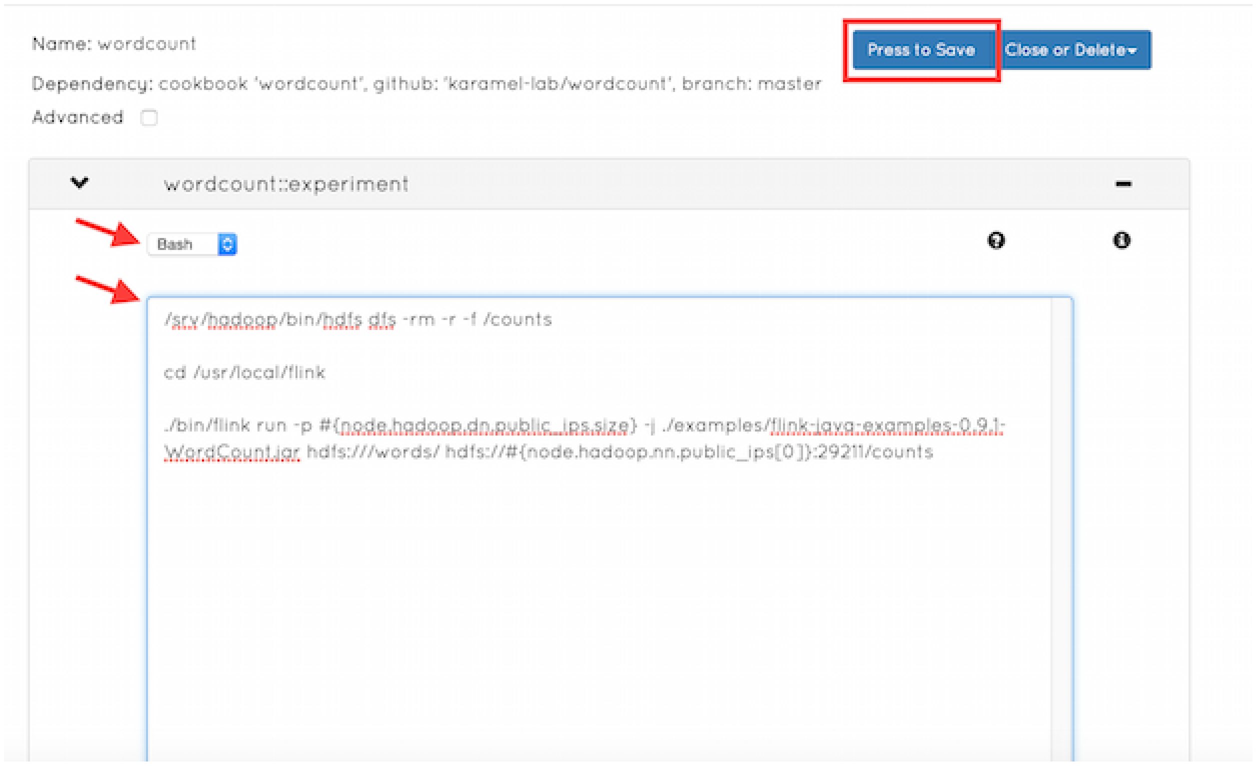
Repository Name: wordcount

Repository Description: Measures performance of Apache-Flink in word-count.

Fig. 30: New experiment on a Github repo.

3.3.5 Write body of experiment

At this point you land into the programming section of your experiment. The default name for the experiment recipe is “experiment”. In the large text-area, as can be seen in the screenshot below, you can write your experiment code either in bash or python. Karamel will automatically wrap your code into a chef recipe. All parameters in experiment come in the format of Chef variables, you should wrap them inside #{} and prefix them `node.<cookbookname>`. By default, they have the format `#{node.cookbook.paramName}`, where `paramName` is the name of your parameter. If you write results of your experiment in a file called /tmp/wordcout__experiment.out - if your cookbook called “wordcount” and your recipe called “experiment”- Karamel will download that file and will put it into ~/.karamel/results/ folder of your client machine.



Name: wordcount

Dependency: cookbook 'wordcount', github: 'karamel-lab/wordcount', branch: master

Advanced

Bash

```
/srv/hadoop/bin/hdfs dfs -rm -r -f /counts

cd /usr/local/flink

./bin/flink run -p #{node.hadoop.dn.public_ips.size} -j ./examples/flink-java-examples-0.9.1-WordCount.jar hdfs:///words/ hdfs://#{node.hadoop.nn.public_ips[0]}:29211/counts
```

Fig. 31: Experiment bash script.

3.3.6 Define orchestration rules for experiment

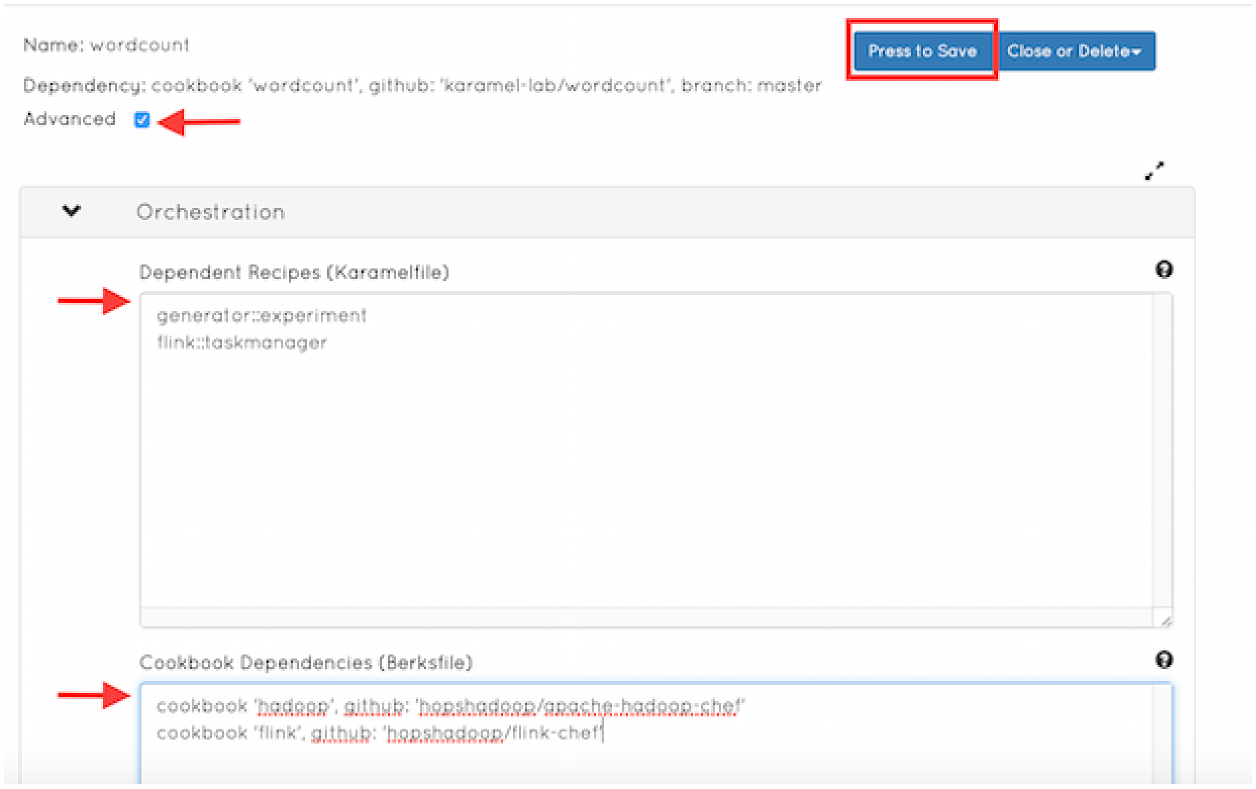
Placing your experiment in the right order in the cluster orchestration is a very essential part of your experiment design. Click the `advanced` checkbox, write in the line-separated Cookbook::recipe_name that your experiment requires have finished before the experiment will start. If your experiment is dependent on other cookbooks (for recipes or parameters), you must enter the relative GitHub name for the cookbook and the version/branch in line-separated format in the second text-area.

3.3.7 Push your experiment into Github

You can save your cluster to GitHub by pressing the save button in the top-right hand corner of the webpage. This will generate your cookbook and copy all the files to Github by adding, committing, and pushing the new files to GitHub.

3.3.8 Approve uploaded experiment to Github

Navigate to your Github repo on your web browser and you can see your cookbook.



Name: wordcount

Dependency: cookbook 'wordcount', github: 'karamel-lab/wordcount', branch: master

Advanced ☑

Orchestration

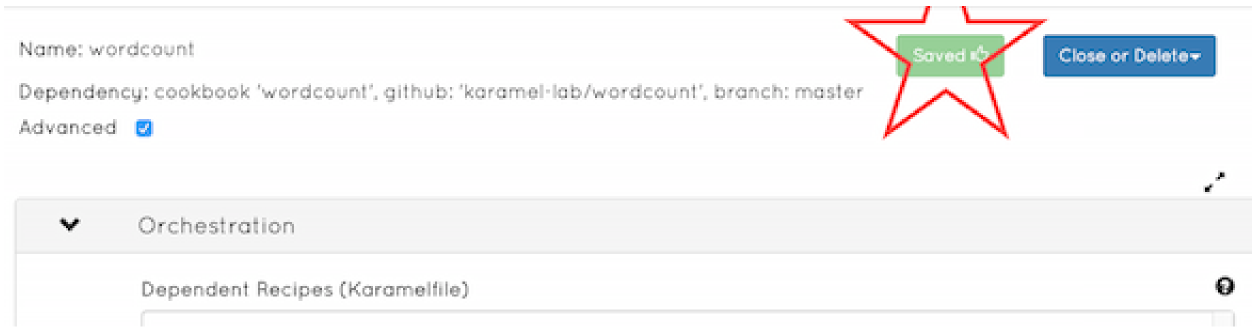
Dependent Recipes (Karamelfile)

```
generator::experiment
flink::taskmanager
```

Cookbook Dependencies (Berksfile)

```
cookbook 'hadoop', github: 'hopshadoop/apache-hadoop-chef'
cookbook 'flink', github: 'hopshadoop/flink-chef'
```

Fig. 32: Orchestration rules for new cluster.



Name: wordcount

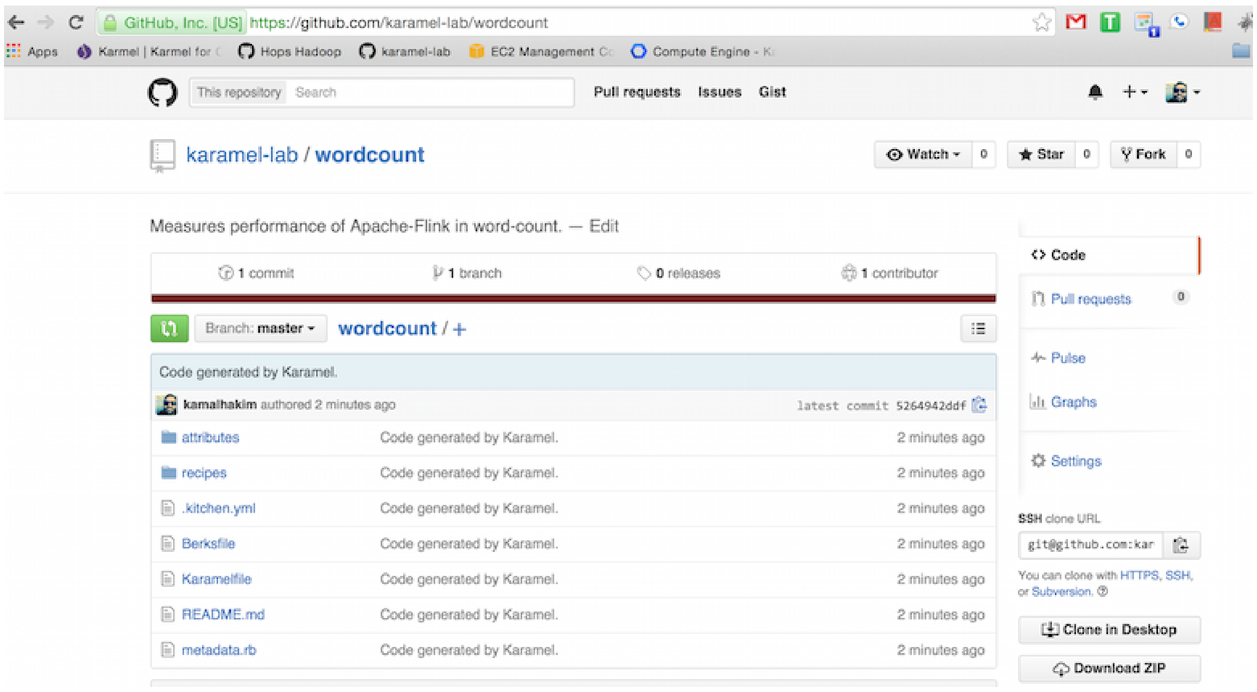
Dependency: cookbook 'wordcount', github: 'karamel-lab/wordcount', branch: master

Advanced ☑

Orchestration

Dependent Recipes (Karamelfile)

Fig. 33: Push the experiment to a Github repository.



karamel-lab / wordcount

Measures performance of Apache-Flink in word-count. — Edit

1 commit · 1 branch · 0 releases · 1 contributor

Branch: master · wordcount / +

Code generated by Karamel.

kamalhakim authored 2 minutes ago · latest commit 5264942ddf

attributes	Code generated by Karamel.	2 minutes ago
recipes	Code generated by Karamel.	2 minutes ago
.kitchen.yml	Code generated by Karamel.	2 minutes ago
Berksfile	Code generated by Karamel.	2 minutes ago
Karamelfile	Code generated by Karamel.	2 minutes ago
README.md	Code generated by Karamel.	2 minutes ago
metadata.rb	Code generated by Karamel.	2 minutes ago

SSH clone URL

git@github.com:kar

You can clone with HTTPS, SSH, or Subversion.

Fig. 34: New experiment added to Github.

CHAPTER 4

Cluster Definition

The cloud definition format is an expressive DSL based on YAML as you can see in the following sample. Since Karamel can run several clusters simultaneously, the name of each cluster must be unique.

Currently We support four cloud providers: Amazon EC2 (ec2), Google Compute Engine (gce), Openstack Nova (nova) and bare-metal(baremetal). You can define a provider globally within a cluster definition file or you can define a different provider for each group in the cluster definition file. In the group scope, you can overwrite some attributes of the network/machines in the global scope or you can choose an entirely different cloud provider, defining a multi-cloud deployment. Settings and properties for each provider is introduced in later section. For a single cloud deployment, one often uses group-scope provider details to override the type of instance used for machines in the group. For example, one group of nodes may require lots of memory and processing power, while other nodes require less. For AWS, you would achive this by overriding the `instanceType` attribute.

The Cookbooks section specifies GitHub references to the cookbooks used in the cluster definition. It is possible to refer to a specific version or branch for each GitHub repository.

We group machines based on the application stack (list of recipes) that should be installed on the machines in the group. The number of machines in each group and list of recipes must be defined under each group name.

```
name: spark
ec2:
  type: m3.medium
  region: eu-west-1

cookbooks:
  hadoop:
    github: "hopshadoop/apache-hadoop-chef"
  spark:
    github: "hopshadoop/spark-chef"
    branch: "master"

groups:
  namenodes:
    size: 1
    recipes:
      - hadoop::nn
      - hadoop::rm
      - hadoop::jhs
      - spark::master
  datanodes:
    size: 2
    recipes:
      - hadoop::dn
      - hadoop::nm
      - spark::slave
```

4.1 AWS(Amazon EC2)

In cluster definitions, we use key word *ec2* for deploying the cluster on Amazon EC2 Cloud. The following code snippet shows all supported attributes for AWS.

```
ec2:
  type: c4.large
  region: eu-west-1
  ami: ami-47a23a30
  price: 0.1
  vpc: vpc-f70ea392
  subnet: subnet-e7830290
```

Type of the virtual machine, region (data center) and Amazon Machine Image are the basic properties.

We support spot instances that is a way to control your budget. Since Amazon prices are changing based on demand, price is a limit you can set if you are not willing to pay beyond that limit (price unit is USD).

4.1.1 Virtual Private Cloud on AWS-EC2

We support AWS VPC on EC2 for better performance. First you must define your VPC in EC2 with the following steps then include your vpc and subnet id in the cluster definition as it is shown above.

1. Make a VPC and a subnet assigned to it under your ec2.
2. Check the “Auto-assign Public IP” item for your subnet.
3. Make an internet gateway and attach it to the VPC.

4. Make a routing table for your VPC and add a row for your gateway into it, on this row open all ips '0.0.0.0/0'.
5. Add your vpc-id and subnet-id into the ec2 section of your yaml like the following example. Also make sure you are using the right image and type of instance for your vpc.

4.2 Google Compute Engine

To deploy the cluster on Google's infrastructure, we use the keyword *gce* in the cluster definition YAML file. Following code snippet shows the current supported attributes:

```
gce:
 type: n1-standard-1
 zone: europe-west1-b
  image: ubuntu-1404-trusty-v20150316
```

Machine type, zone of the VMs, and the VM image can be specified by the user.

Karamel uses Compute Engine's OAuth 2.0 authentication method. Therefore, an OAuth 2.0 client ID needs to be created through the Google's Developer Console. The description on how to generate a client ID is available here. You need to select *Service account* as the application type. After generating a service account, click on *Generate new JSON key* button to download a generated JSON file that contains both private and public keys. You need to register the fullpath of the generated JSON file with Karamel API.

4.3 Bare-metal

Bare-metal clusters are supported, but the machines must first be prepared with support for login using a ssh-key that is stored on the Karamel client. The target hosts must be contactable using ssh from the Karamel client, and the target hosts' ip-addresses must be specified in the cluster definition. If you have many ip-addresses in a range, it is possible to give range of addresses instead of specifying them one by one (the second example below). The public key stored on the Karamel client should be copied to the *.ssh/authorized_keys* file in the home folder of the sudo account on the target machines that will be used to install the software. The username goes into the cluster definition is the sudo account, and if there is a password required to get sudo access, it must be entered in the Web UI or entered through Karamel's programmatic API.

```
baremetal:
 username: ubuntu
 ips:
  - 192.168.33.12
  - 192.168.33.13
  - 192.168.33.14
  - 192.168.44.15
```

4.3.1 IP-Range

```
baremetal:
  username: ubuntu
  ips:
  - 192.168.33.12-192.168.33.14
  - 192.168.44.15
```

4.4 OCCI

To deploy the cluster on OCCI compatible infrastructure you must provide following information via yaml file: - occiEndpoint - defines on which OCCI cluster will your virtual machines be created - occiImage - virtual machine template - occiImageSize - compute resource size (procesors & memory) - usename - account name with root privileges

Furthermore you must have voms-client installed and configured (https://italiangrid.github.io/voms/documentation/voms-clients-guide/3.0.3/). To connect to you organisation's resorces generate proxy certificate in shell via "voms-proxy-init -voms <name of you virtual organisation> -rfc" and add path to proxy certificate via OCCI launch panel in web gui.

```
name: OcciAmbari
  occi:
    occiEndpoint: "https://carach5.ics.muni.cz:11443"
    occiImage: "http://occi.carach5.ics.muni.cz/occi/infrastructure/os_tpl#uuid_egi_
↪ubuntu_server_14_04_lts_fedcloud_warg_131"
    occiImageSize: "http://fedcloud.egi.eu/occi/compute/flavour/1.0#mem_large"
    usename: "ubuntu"

cookbooks:
  ambari:
    github: "jimdowling/ambari"
    branch: "master"

groups:
  server:
    size: 1
    recipes:
        - ambari::server
  agents:
    size: 1
    recipes:
        - ambari::agent
```

CHAPTER 5

Deploying BiobankCloud with Karamel

BiobankCloud is a Platform-as-a-Service (PaaS) for biobanking with Big Data (Hadoop). BiobankCloud brings together

- Hops Hadoop with
- SAASFEE, a Bioinformatics platform for YARN that provides both a workflow language (Cuneiform) and a 2nd-level scheduler (HiWAY)
- Charon, a cloud-of-clouds filesystem, for sharing data between BiobankCloud clusters.

We have written karamelized Chef cookbooks for installing all of the components of BiobankCloud, and we provide some sample cluster definitions for installing small, medium, and large BiobankCloud clusters. Users are, of course, expected to adapt these sample cluster definitions to their cloud provider or bare-metal environment as well as their needs.

The following is a brief description of the karmelized Chef cookbooks that we have developed to support the installation of BiobankCloud. The cookbooks are all publicly available at: http://github.com/.

- hopshadoop/apache-hadoop-chef
- hopshadoop/hops-hadoop-chef
- hopshadoop/elasticsearch-chef
- hopshadoop/ndb-chef
- hopshadoop/zeppelin-chef
- hopshadoop/hopsworks-chef
- hopshadoop/spark-chef
- hopshadoop/flink-chef
- biobankcloud/charon-chef
- biobankcloud/hiway-chef

The following is a cluster definition file that installs BiobankCloud on a single m3.xlarge instance on AWS/EC2:

```
name: BiobankCloudSingleNodeAws
ec2:
    type: m3.xlarge
    region: eu-west-1
cookbooks:
  hops:
    github: "hopshadoop/hops-hadoop-chef"
    branch: "master"
  hadoop:
    github: "hopshadoop/apache-hadoop-chef"
    branch: "master"
  hopsworks:
    github: "hopshadoop/hopsworks-chef"
    branch: "master"
  ndb:
    github: "hopshadoop/ndb-chef"
    branch: "master"
  spark:
    github: "hopshadoop/spark-chef"
    branch: "hops"
  zeppelin:
    github: "hopshadoop/zeppelin-chef"
    branch: "master"
  elastic:
    github: "hopshadoop/elasticsearch-chef"
    branch: "master"
  charon:
    github: "biobankcloud/charon-chef"
    branch: "master"
  hiway:
    github: "biobankcloud/hiway-chef"
    branch: "master"
attrs:
  hdfs:
    user: glassfish
    conf_dir: /mnt/hadoop/etc/hadoop
  hadoop:
    dir: /mnt
    yarn:
        user: glassfish
        nm:
          memory_mbs: 9600
        vcores: 4
    mr:
        user: glassfish
  spark:
    user: glassfish
  hiway:
    home: /mnt/hiway
    user: glassfish
    release: false
    hiway:
      am:
        memory_mb: '512'
        vcores: '1'
      worker:
        memory_mb: '3072'
```

(continues on next page)

(continued from previous page)

```
        vcores: '1'
  hopsworks:
    user: glassfish
    twofactor_auth: "true"
  hops:
    use_hopsworks: "true"
  ndb:
    DataMemory: '50'
    IndexMemory: '15'
    dir: "/mnt"
    shared_folder: "/mnt"
  mysql:
    dir: "/mnt"
  charon:
    user: glassfish
    group: hadoop
    user_email: jdowling@kth.se
    use_only_aws: true
groups:
  master:
    size: 1
    recipes:
        - ndb::mysqld
        - ndb::mgmd
        - ndb::ndbd
        - hops::ndb
        - hops::rm
        - hops::nn
        - hops::dn
        - hops::nm
        - hopsworks
        - zeppelin
        - charon
        - elastic
        - spark::master
        - hiway::hiway_client
        - hiway::cuneiform_client
        - hiway::hiway_worker
        - hiway::cuneiform_worker
        - hiway::variantcall_worker
```

The following is a cluster definition file that installs a very large, highly available, BiobankCloud cluster on 56 m3.xlarge instance on AWS/EC2:

```
name: BiobankCloudMediumAws
ec2:
    type: m3.xlarge
    region: eu-west-1
cookbooks:
  hops:
    github: "hopshadoop/hops-hadoop-chef"
    branch: "master"
  hadoop:
    github: "hopshadoop/apache-hadoop-chef"
    branch: "master"
  hopsworks:
```

(continues on next page)

(continued from previous page)

```
    github: "hopshadoop/hopsworks-chef"
    branch: "master"
  ndb:
    github: "hopshadoop/ndb-chef"
    branch: "master"
  spark:
    github: "hopshadoop/spark-chef"
    branch: "hops"
  zeppelin:
    github: "hopshadoop/zeppelin-chef"
    branch: "master"
  elastic:
    github: "hopshadoop/elasticsearch-chef"
    branch: "master"
  charon:
    github: "biobankcloud/charon-chef"
    branch: "master"
  hiway:
    github: "biobankcloud/hiway-chef"
    branch: "master"
attrs:
  hdfs:
    user: glassfish
    conf_dir: /mnt/hadoop/etc/hadoop
  hadoop:
    dir: /mnt
    yarn:
        user: glassfish
        nm:
          memory_mbs: 9600
        vcores: 8
    mr:
        user: glassfish
  spark:
    user: glassfish
  hiway:
    home: /mnt/hiway
    user: glassfish
    release: false
    hiway:
      am:
        memory_mb: '512'
        vcores: '1'
      worker:
        memory_mb: '3072'
        vcores: '1'
  hopsworks:
    user: glassfish
    twofactor_auth: "true"
  hops:
    use_hopsworks: "true"
  ndb:
    DataMemory: '8000'
    IndexMemory: '1000'
    dir: "/mnt"
    shared_folder: "/mnt"
  mysql:
```

(continues on next page)

(continued from previous page)

```
    dir: "/mnt"
  charon:
    user: glassfish
    group: hadoop
    user_email: jdowling@kth.se
    use_only_aws: true
groups:
  master:
    size: 1
    bbcui:
        - ndb::mgmd
        - ndb::mysqld
        - hops::ndb
        - hops::client
        - hopsworks
        - spark::yarn
        - charon
        - zeppelin
        - hiway::hiway_client
        - hiway::cuneiform_client
  metadata:
    size: 2
    recipes:
        - hops::ndb
        - hops::rm
        - hops::nn
        - ndb::mysqld
  elastic:
    size: 1
    recipes:
        - elastic
  database:
    size: 2
    recipes:
        - ndb::ndbd
  workers:
    size: 50
    recipes:
        - hops::ndb
        - hops::dn
        - hops::nm
        - hiway::hiway_worker
        - hiway::cuneiform_worker
        - hiway::variantcall_worker
```

Alternative configurations are, of course, possible. You could run several Elasticsearch instances for high availability and more master instances if you have many active clients.

CHAPTER 6

Developer Guide

We have organized our code into two main projects, *karamel-core* and *karamel-ui*. The core is our engine for launching, installing and monitoring clusters. The UI is a standalone web application containing several designers and visualizers. There is a REST-API in between the UI and the core.

The core and REST-API are programmed in Java 7, and the UI is programmed in Angular JS.

6.1 Code quality

1. Testability and mockability: Write your code in a way that you test each unit separately. Split concerns into different modules that you can mock one when testing the other. We use JUnit-4 for unit testing and mockito for mocking.
2. Code styles: Write a DRY (Don't repeat yourself) code, use spaces instead of tab and line width limit is 120.
3. We use Google Guava and its best practices, specially the basic ones such as nullity checks and preconditions.

6.2 Build and run from Source

Ubuntu Requirements:

```
apt-get install lib32z1 lib32ncurses5 lib32bz2-1.0
```

Centos 7 Requirements:

```
Install zlib.i686, ncurses-libs.i686, and bzip2-libs.i686 on CentOS 7
```

Building from root directory:

```
mvn install
```

Running:

```
cd karamel-ui/target/appassembler
./bin/karamel
```

6.3 Building Window Executables

You need to have 32-bit libraries to build the windows exe from Linux, as the launch4j plugin requires them.

```
sudo apt-get install gcc binutils-mingw-w64-x86-64 -y
# Then replace 32-bit libraries with their 64-bit equivalents
cd /home/ubuntu/.m2/repository/net/sf/
cd launch4j/launch4j/3.8.0/launch4j-3.8.0-workdir-linux/bin
rm ld windres
ln -s /usr/bin/x86_64-w64-mingw32-ld ./ld
ln -s /usr/bin/x86_64-w64-mingw32-windres ./windres
```

Then run maven with the -Pwin to run the plugin:

```
mvn -Dwin package
```