Airflow configuration on kubernates

Prerequisites

- A kubernetes cluster(minkube & kubectl)
- Install Airflow

Installing Minikube and kubectl

```
curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-a
md64 && chmod +x minikube
sudo mv -v minikube /usr/local/bin
 minikube version
```

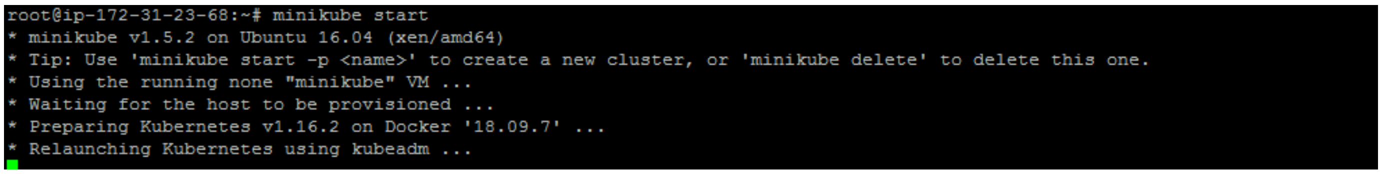
Install Kubectl :

```
$ sudo apt-get update && sudo apt-get install -y apt-transport-https
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo
apt-key add -
$ sudo touch /etc/apt/sources.list.d/kubernetes.list
$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
-a /etc/apt/sources.list.d/kubernetes.list
$ sudo apt-get update && sudo apt-get install -y kubectl
$ kubectl version
```

Start Kubernetes Cluster locally with Minikube

Create and run Kubernetes cluster. This will take some few minutes

```
minikube start --vm-driver=none
```



```
root@ip-172-31-23-68:~# minikube start
* minikube v1.5.2 on Ubuntu 16.04 (xen/amd64)
* Tip: Use 'minikube start -p <name>' to create a new cluster, or 'minikube delete' to delete this one.
* Using the running none "minikube" VM ...
* Waiting for the host to be provisioned ...
* Preparing Kubernetes v1.16.2 on Docker '18.09.7' ...
* Relaunching Kubernetes using kubeadm ...
```

check the status

```
minikube status
```

Install Airflow

Go through the doc for install airflow

Install AirFlow

Kubernetes Executor

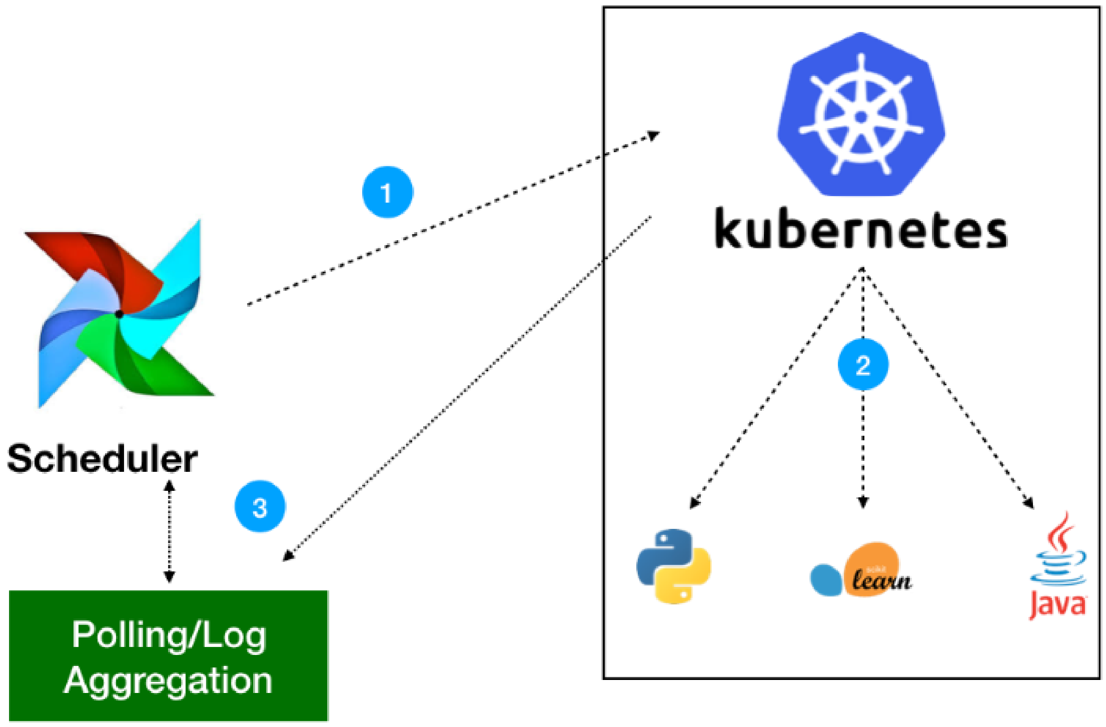
- The biggest issue that Apache Airflow with Kubernetes Executor solves is the dynamic resource allocation. Before the Kubernetes

Executor, all previous Airflow solutions involved static clusters of workers and so you had to determine ahead of time what size cluster you want to use according to your possible workloads
- The kubernetes executor is introduced in Apache Airflow 1.10.0. The Kubernetes executor will create a new pod for every task instance.

The Kubernetes Operator

Operator in airflow is a task definition.When user create a DAG , it use operator like “PythonOperator“or “Bashoperator“.Airflow comes with built-in operators for frameworks like Apache Spark, BigQuery, Hive, and EMR.

Architecture



1. The Kubernetes Operator uses the kubernates python client to generate a request that is processed by the APIServer.
2. Kubernetes will then launch your pod with whatever specs you’ve defined .
3. Images will be loaded with all the necessary environment variables, secrets and dependencies, enacting a single command. Once the job is launched, the operator only needs to monitor the health of track logs.

Using the Kubernetes Operator

Sample DAG file ,that show how kubernates operator works.

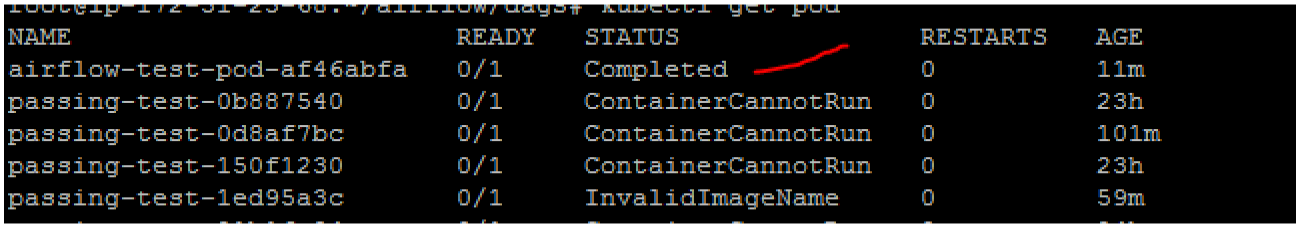
```
import datetime as dt
from airflow import DAG
from datetime import datetime, timedelta
from airflow.contrib.operators.kubernetes_pod_operator import
KubernetesPodOperator
from airflow.operators.dummy_operator import DummyOperator


default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': dt.datetime(2019, 12, 02),
    'retries': 1,
    'retry_delay': dt.timedelta(minutes=5)
}

dag = DAG(
    'kubernetes_sample', default_args=default_args,
schedule_interval=timedelta(minutes=10))


start = DummyOperator(task_id='run_this_first', dag=dag)

passing = KubernetesPodOperator(namespace='default',
                          image="python",
                          cmds=["Python","-c"],
                          arguments=["print('hello world')"],
                          labels={"foo": "bar"},
                          name="passing-test",
                          task_id="passing-task",
                          get_logs=True,
                          dag=dag
                          )

failing = KubernetesPodOperator(namespace='default',
                          image="ubuntu:1604",
                          cmds=["Python","-c"],
                          arguments=["print('hello world')"],
                          labels={"foo": "bar"},
                          name="fail",
                          task_id="failing-task",
                          get_logs=True,
                          dag=dag
                          )

passing.set_upstream(start)
failing.set_upstream(start)
```

- This DAG creates two pods on Kubernetes .
- Using KubernetesPodOperator it will create pod.

working example

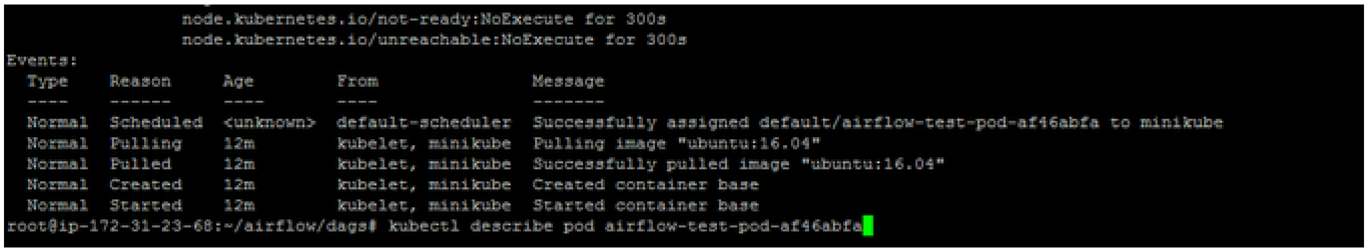
```
"""
This is an example dag for using the KubernetesPodOperator.
"""
import datetime as dt
from airflow.models import DAG
from airflow.utils.dates import days_ago
from airflow.utils.log.logging_mixin import LoggingMixin

log = LoggingMixin().log

try:
    # Kubernetes is optional, so not available in vanilla Airflow
    # pip install 'apache-airflow[kubernetes]'
    from airflow.contrib.operators.kubernetes_pod_operator import
KubernetesPodOperator

    default_args = {
        'owner': 'Airflow',
        'depends_on_past': False,
    'start_date': dt.datetime(2019, 12, 02),
    'retries': 1,
    'retry_delay': dt.timedelta(minutes=5)
    }

    with DAG(
        dag_id='example_kubernetes_operator',
        default_args=default_args,
        schedule_interval=None
    ) as dag:

        tolerations = [
            {
                'key': "key",
                'operator': 'Equal',
                'value': 'value'
            }
        ]

        k = KubernetesPodOperator(
            namespace='default',
            image="ubuntu:16.04",
            cmds=["bash", "-cx"],
            arguments=["echo", "10"],
            labels={"foo": "bar"},
            name="airflow-test-pod",
            in_cluster=False,
                task_id="task",
                get_logs=True,
                is_delete_operator_pod=False,
                tolerations=tolerations
            )

except ImportError as e:
    log.warning("Could not import KubernetesPodOperator: " + str(e))
    log.warning("Install kubernetes dependencies with: "
                "    pip install 'apache-airflow[kubernetes]'")
```

In above code it will create `airflow-test-pod` with base image `ubuntu:16.04` using `KubernetesPodOperator`.



```
NAME                          READY   STATUS               RESTARTS   AGE
airflow-test-pod-af46abfa     0/1     Completed            0          11m
passing-test-0b887540         0/1     ContainerCannotRun   0          23h
passing-test-0d8af7bc         0/1     ContainerCannotRun   0          101m
passing-test-150f1230         0/1     ContainerCannotRun   0          23h
passing-test-1ed95a3c         0/1     InvalidImageName     0          59m
```

Run kubectl describe pod airflow-test-pod-af46abfa it shows detailed status of POD.



```
                 node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age        From               Message
  ----    ------     ----       ----               -------
  Normal  Scheduled  <unknown>  default-scheduler  Successfully assigned default/airflow-test-pod-af46abfa to minikube
  Normal  Pulling    12m        kubelet, minikube  Pulling image "ubuntu:16.04"
  Normal  Pulled     12m        kubelet, minikube  Successfully pulled image "ubuntu:16.04"
  Normal  Created    12m        kubelet, minikube  Created container base
  Normal  Started    12m        kubelet, minikube  Started container base
root@ip-172-31-23-68:~/airflow/dags# kubectl describe pod airflow-test-pod-af46abfa
```

Backendcode

When we execute kubernates on airflow by using `KubernetesPodOperator` it will launch Kubernates pods.

Backendcode:

```
import kubernetes.client.models as k8s

from airflow.exceptions import AirflowException
from airflow.kubernetes import kube_client, pod_generator, pod_launcher
from airflow.kubernetes.k8s_model import append_to_pod
from airflow.kubernetes.pod import Port, Resources
from airflow.kubernetes.pod_runtime_info_env import PodRuntimeInfoEnv
from airflow.kubernetes.secret import Secret
from airflow.kubernetes.volume import Volume
from airflow.kubernetes.volume_mount import VolumeMount
from airflow.models import BaseOperator
from airflow.utils.decorators import apply_defaults
from airflow.utils.helpers import validate_key
from airflow.utils.state import State
from airflow.version import version as airflow_version


class KubernetesPodOperator(BaseOperator):  # pylint:
disable=too-many-instance-attributes
    """
    Execute a task in a Kubernetes Pod
    .. note::
        If you use `Google Kubernetes Engine
<https://cloud.google.com/kubernetes-engine/>`__, use
```

```
:class:`~airflow.gcp.operators.kubernetes_engine.GKEPodOperator`, which
        simplifies the authorization process.
    :param namespace: the namespace to run within kubernetes.
    :type namespace: str
    :param image: Docker image you wish to launch. Defaults to
hub.docker.com,
        but fully qualified URLS will point to custom repositories.
    :type image: str
    :param name: name of the pod in which the task will run, will be
used to
        generate a pod id (DNS-1123 subdomain, containing only
[a-z0-9.-]).
    :type name: str
    :param cmds: entrypoint of the container. (templated)
        The docker images's entrypoint is used if this is not provided.
    :type cmds: list[str]
    :param arguments: arguments of the entrypoint. (templated)
        The docker image's CMD is used if this is not provided.
    :type arguments: list[str]
    :param ports: ports for launched pod.
    :type ports: list[airflow.kubernetes.pod.Port]
    :param volume_mounts: volumeMounts for launched pod.
    :type volume_mounts:
list[airflow.kubernetes.volume_mount.VolumeMount]
    :param volumes: volumes for launched pod. Includes ConfigMaps and
PersistentVolumes.
    :type volumes: list[airflow.kubernetes.volume.Volume]
    :param env_vars: Environment variables initialized in the container.
(templated)
    :type env_vars: dict
    :param secrets: Kubernetes secrets to inject in the container.
        They can be exposed as environment vars or files in a volume.
    :type secrets: list[airflow.kubernetes.secret.Secret]
    :param in_cluster: run kubernetes client with in_cluster
configuration.
    :type in_cluster: bool
    :param cluster_context: context that points to kubernetes cluster.
        Ignored when in_cluster is True. If None, current-context is
used.
    :type cluster_context: str
    :param labels: labels to apply to the Pod.
    :type labels: dict
    :param startup_timeout_seconds: timeout in seconds to startup the
pod.
    :type startup_timeout_seconds: int
    :param get_logs: get the stdout of the container as logs of the
tasks.
    :type get_logs: bool
    :param image_pull_policy: Specify a policy to cache or always pull
an image.
```

```
    :type image_pull_policy: str
    :param annotations: non-identifying metadata you can attach to the
Pod.
        Can be a large range of data, and can include characters
        that are not permitted by labels.
    :type annotations: dict
    :param resources: A dict containing resources requests and limits.
        Possible keys are request_memory, request_cpu, limit_memory,
limit_cpu,
        and limit_gpu, which will be used to generate
airflow.kubernetes.pod.Resources.
        See also
kubernetes.io/docs/concepts/configuration/manage-compute-resources-conta
iner
    :type resources: dict
    :param affinity: A dict containing a group of affinity scheduling
rules.
    :type affinity: dict
    :param config_file: The path to the Kubernetes config file.
(templated)
        If not specified, default value is ``~/.kube/config``
    :type config_file: str
    :param node_selectors: A dict containing a group of scheduling
rules.
    :type node_selectors: dict
    :param image_pull_secrets: Any image pull secrets to be given to the
pod.
        If more than one secret is required, provide a
        comma separated list: secret_a,secret_b
    :type image_pull_secrets: str
    :param service_account_name: Name of the service account
    :type service_account_name: str
    :param is_delete_operator_pod: What to do when the pod reaches its
final
        state, or the execution is interrupted.
        If False (default): do nothing, If True: delete the pod
    :type is_delete_operator_pod: bool
    :param hostnetwork: If True enable host networking on the pod.
    :type hostnetwork: bool
    :param tolerations: A list of kubernetes tolerations.
    :type tolerations: list tolerations
    :param configmaps: A list of configmap names objects that we
        want mount as env variables.
    :type configmaps: list[str]
    :param security_context: security options the pod should run with
(PodSecurityContext).
    :type security_context: dict
    :param pod_runtime_info_envs: environment variables about
        pod runtime information (ip, namespace, nodeName, podName).
    :type pod_runtime_info_envs:
```

```
list[airflow.kubernetes.pod_runtime_info_env.PodRuntimeInfoEnv]
    :param dnspolicy: dnspolicy for the pod.
    :type dnspolicy: str
    :param full_pod_spec: The complete podSpec
    :type full_pod_spec: kubernetes.client.models.V1Pod
    :param do_xcom_push: If True, the content of the file
        /airflow/xcom/return.json in the container will also be pushed
to an
        XCom when the container completes.
    :type do_xcom_push: bool
    """
    template_fields = ('cmds', 'arguments', 'env_vars', 'config_file')

    @apply_defaults
    def __init__(self,  # pylint:
disable=too-many-arguments,too-many-locals
                 namespace: str,
                 image: str,
                 name: str,
                 cmds: Optional[List[str]] = None,
                 arguments: Optional[List[str]] = None,
                 ports: Optional[List[Port]] = None,
                 volume_mounts: Optional[List[VolumeMount]] = None,
                 volumes: Optional[List[Volume]] = None,
                 env_vars: Optional[Dict] = None,
                 secrets: Optional[List[Secret]] = None,
                 in_cluster: bool = True,
                 cluster_context: Optional[str] = None,
                 labels: Optional[Dict] = None,
                 startup_timeout_seconds: int = 120,
                 get_logs: bool = True,
                 image_pull_policy: str = 'IfNotPresent',
                 annotations: Optional[Dict] = None,
                 resources: Optional[Dict] = None,
                 affinity: Optional[Dict] = None,
                 config_file: Optional[str] = None,
                 node_selectors: Optional[Dict] = None,
                 image_pull_secrets: Optional[str] = None,
                 service_account_name: str = 'default',
                 is_delete_operator_pod: bool = False,
                 hostnetwork: bool = False,
                 tolerations: Optional[List] = None,
                 configmaps: Optional[List] = None,
                 security_context: Optional[Dict] = None,
                 pod_runtime_info_envs:
Optional[List[PodRuntimeInfoEnv]] = None,
                 dnspolicy: Optional[str] = None,
                 full_pod_spec: Optional[k8s.V1Pod] = None,
                 *args,
                 **kwargs):
```

```
        if kwargs.get('xcom_push') is not None:
            raise AirflowException("'xcom_push' was deprecated, use
'do_xcom_push' instead")
        super().__init__(*args, resources=None, **kwargs)

        self.pod = None

        self.image = image
        self.namespace = namespace
        self.cmds = cmds or []
        self.arguments = arguments or []
        self.labels = labels or {}
        self.startup_timeout_seconds = startup_timeout_seconds
        self.name = self._set_name(name)
        self.env_vars = env_vars or {}
        self.ports = ports or []
        self.volume_mounts = volume_mounts or []
        self.volumes = volumes or []
        self.secrets = secrets or []
        self.in_cluster = in_cluster
        self.cluster_context = cluster_context
        self.get_logs = get_logs
        self.image_pull_policy = image_pull_policy
        self.node_selectors = node_selectors or {}
        self.annotations = annotations or {}
        self.affinity = affinity or {}
        self.resources = self._set_resources(resources)
        self.config_file = config_file
        self.image_pull_secrets = image_pull_secrets
        self.service_account_name = service_account_name
        self.is_delete_operator_pod = is_delete_operator_pod
        self.hostnetwork = hostnetwork
        self.tolerations = tolerations or []
        self.configmaps = configmaps or []
        self.security_context = security_context or {}
        self.pod_runtime_info_envs = pod_runtime_info_envs or []
        self.dnspolicy = dnspolicy
        self.full_pod_spec = full_pod_spec

    def execute(self, context):
        try:
            client =
kube_client.get_kube_client(in_cluster=self.in_cluster,

cluster_context=self.cluster_context,

config_file=self.config_file)

            # Add Airflow Version to the label
            # And a label to identify that pod is launched by
```

```
KubernetesPodOperator
            self.labels.update(
                {
                    'airflow_version': airflow_version.replace('+',
'-'),
                    'kubernetes_pod_operator': 'True',
                }
            )

            pod = pod_generator.PodGenerator(
                image=self.image,
                namespace=self.namespace,
                cmds=self.cmds,
                args=self.arguments,
                labels=self.labels,
                name=self.name,
                envs=self.env_vars,
                extract_xcom=self.do_xcom_push,
                image_pull_policy=self.image_pull_policy,
                node_selectors=self.node_selectors,
                annotations=self.annotations,
                affinity=self.affinity,
                image_pull_secrets=self.image_pull_secrets,
                service_account_name=self.service_account_name,
                hostnetwork=self.hostnetwork,
                tolerations=self.tolerations,
                configmaps=self.configmaps,
                security_context=self.security_context,
                dnspolicy=self.dnspolicy,
                pod=self.full_pod_spec,
            ).gen_pod()

            pod = append_to_pod(
                pod,
                self.pod_runtime_info_envs +
                self.ports +
                self.resources +
                self.secrets +
                self.volumes +
                self.volume_mounts
            )

            self.pod = pod

            launcher = pod_launcher.PodLauncher(kube_client=client,

extract_xcom=self.do_xcom_push)

            try:
                (final_state, result) = launcher.run_pod(
                    pod,
                    startup_timeout=self.startup_timeout_seconds,
                    get_logs=self.get_logs)
            finally:
                if self.is_delete_operator_pod:
                    launcher.delete_pod(pod)

            if final_state != State.SUCCESS:
                raise AirflowException(
                    'Pod returned a failure:
{state}'.format(state=final_state)
                )

            return result
        except AirflowException as ex:
            raise AirflowException('Pod Launching failed:
{error}'.format(error=ex))

    @staticmethod
    def _set_resources(resources):
        return [Resources(**resources) if resources else Resources()]

    @staticmethod
    def _set_name(name):
        validate_key(name, max_length=63)
        return re.sub(r'[^a-z0-9.-]+', '-', name.lower())
```