AppArmor profile example

This simple file denies all file write access:

```
#include <tunables/global>

profile k8s-apparmor-example-deny-write flags=(attach_disconnected) {
  #include <abstractions/base>

  file,

  # Deny all file writes.
  deny /** w,
}
```

Next, we have to install the profiles, this is only one way to do so.

```
NODES=(
    # The SSH-accessible domain names of your nodes
    gke-test-default-pool-239f5d02-gyn2.us-central1-a.my-k8s
    gke-test-default-pool-239f5d02-x1kf.us-central1-a.my-k8s
    gke-test-default-pool-239f5d02-xwux.us-central1-a.my-k8s)
for NODE in ${NODES[*]}; do ssh $NODE 'sudo apparmor_parser -q <<EOF
#include <tunables/global>

profile k8s-apparmor-example-deny-write flags=(attach_disconnected) {
  #include <abstractions/base>

  file,

  # Deny all file writes.
  deny /** w,
}
EOF'
done
```

Now, let's create the YAML file for the Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-apparmor
  annotations:
    # Tell Kubernetes to apply the AppArmor profile "k8s-apparmor-example-deny-write".
    # Note that this is ignored if the Kubernetes node is not running version 1.4 or greater.
    container.apparmor.security.beta.kubernetes.io/hello: localhost/k8s-apparmor-example-deny-write
spec:
  containers:
  - name: hello
    image: busybox
    command: [ "sh", "-c", "echo 'Hello AppArmor!' && sleep 1h" ]
```

Now, let's run it

```
kubectl create -f ./hello-apparmor.yaml
```

If we run the view the pod events, we can see the AA profile in the container.

```
kubectl get events | grep hello-apparmor
```

There are other ways to verify, see the first page for that.

If we try to violate the profile

```
kubectl exec hello-apparmor touch /tmp/test
```

Output

```
touch: /tmp/test: Permission denied
error: error executing remote command: command terminated with non-zero exit code: Error executing in Docker Container: 1
```