Using Open Policy Agent to Meet Evolving Policy Requirements

Jeremy Rickard

About Me

https://twitter.com/jrrickard

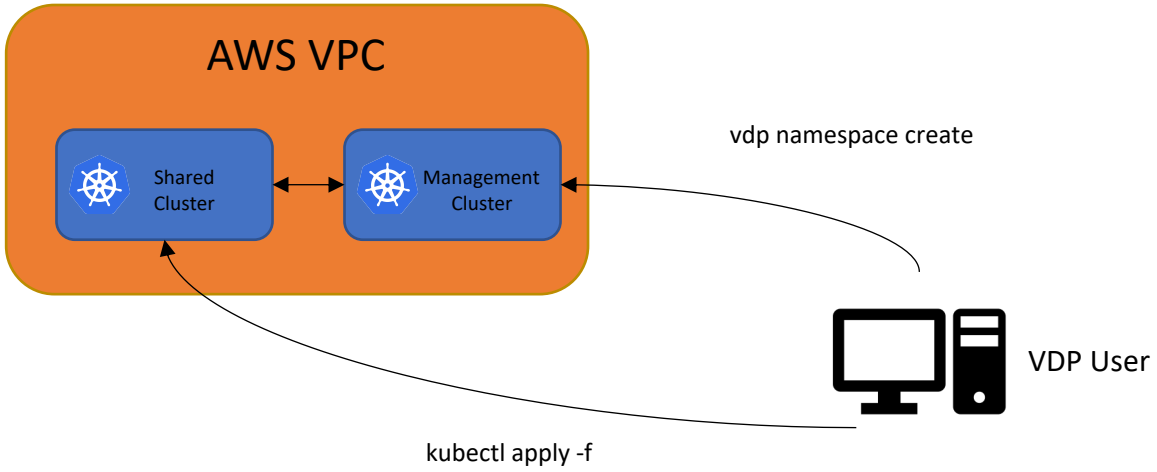
jerickar

jeremy.r.rickard@gmail.com

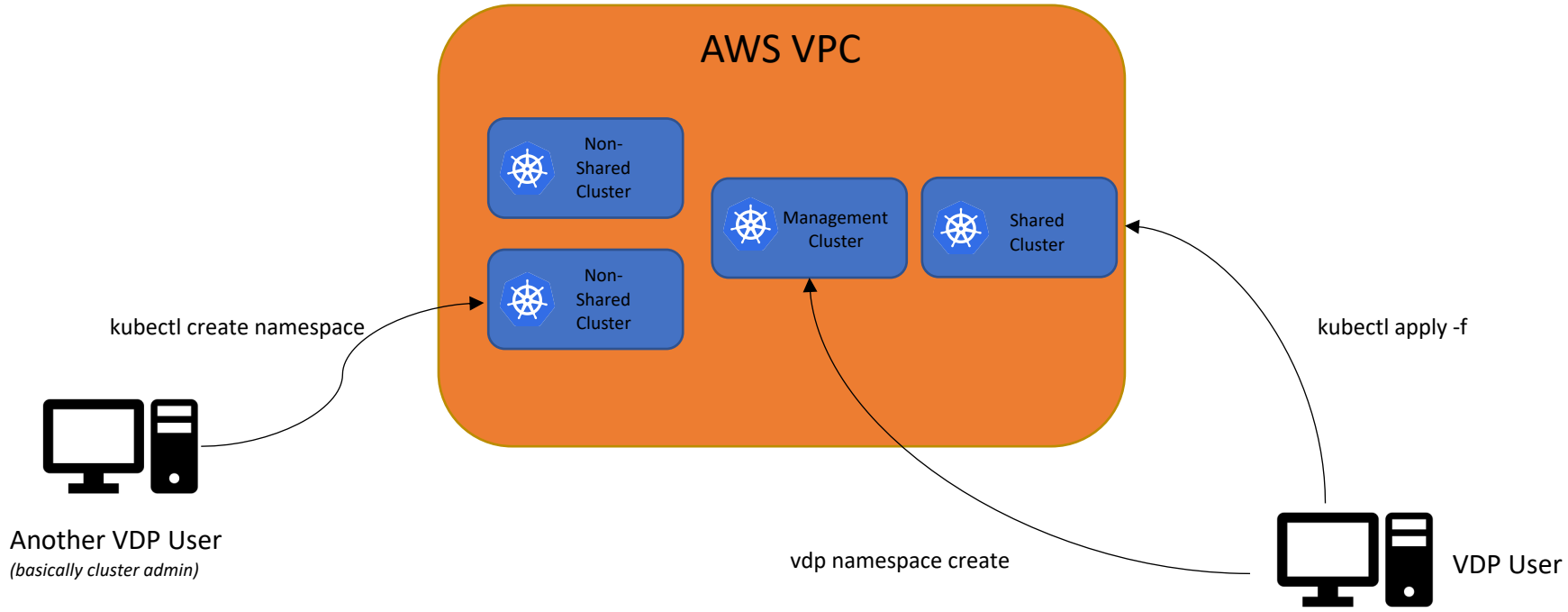
https://github.com/jeremyrickard

VMware Developer Platform

"The VMware Developer Platform (VDP) is a collection of infrastructure and software services available to internal VMware engineering teams to assist with deploying and operating VMware SaaS applications in a stable, secure, efficient and consistent way. "

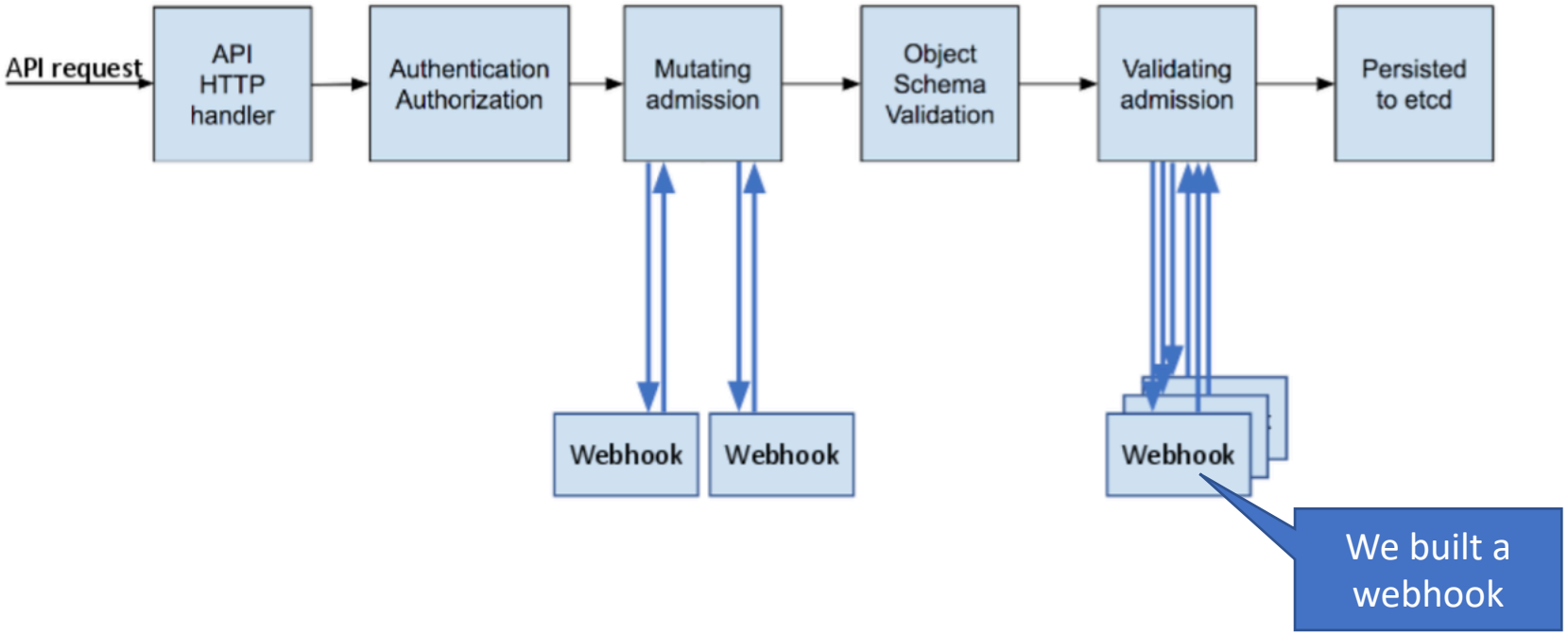


Grow Pains....

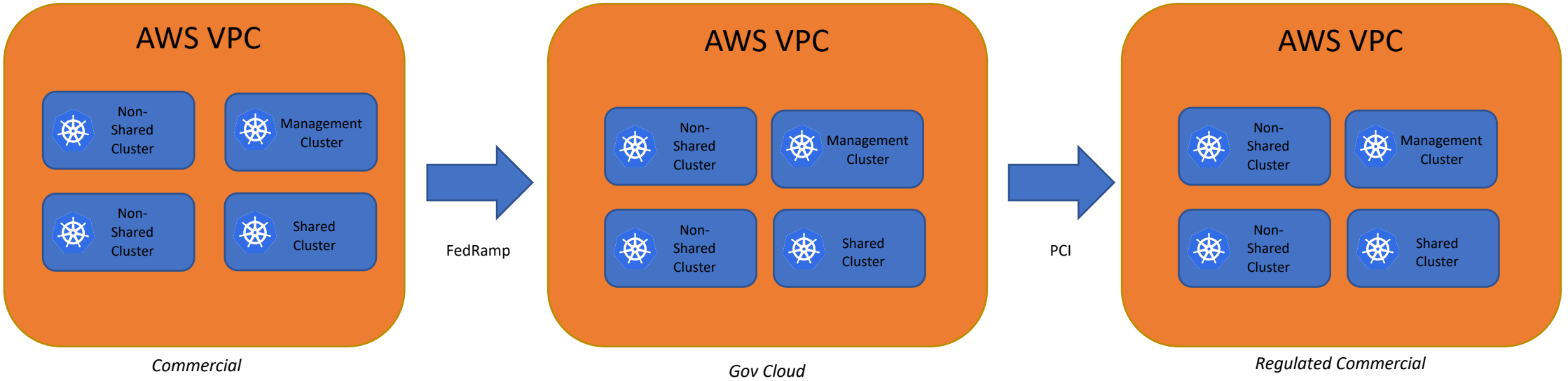

AWS VPC
Non-Shared Cluster
Non-Shared Cluster
Management Cluster
Shared Cluster
kubectl create namespace
kubectl apply -f
Another VDP User
(basically cluster admin)
vdp namespace create
VDP User

Challenges!

Webhooks!


API request
API HTTP handler
Authentication Authorization
Mutating admission
Object Schema Validation
Validating admission
Persisted to etcd
Webhook
Webhook
Webhook
We built a webhook

Our scope grows…



Commercial

Gov Cloud

Regulated Commercial

More Grow Pains….

- Each of these new environments brings new requirements
  - FedRAMP High: > 400 controls
  - PCI: Similar…but different.

> *Compensating controls may be considered for most PCI DSS requirements when an entity cannot meet a requirement explicitly as stated, due to legitimate technical or documented business constraints, but has sufficiently mitigated the risk associated with the requirement through implementation of other, or compensating, controls.*

Policies per cluster seems like a good idea.

(we also don’t want users to hate us)

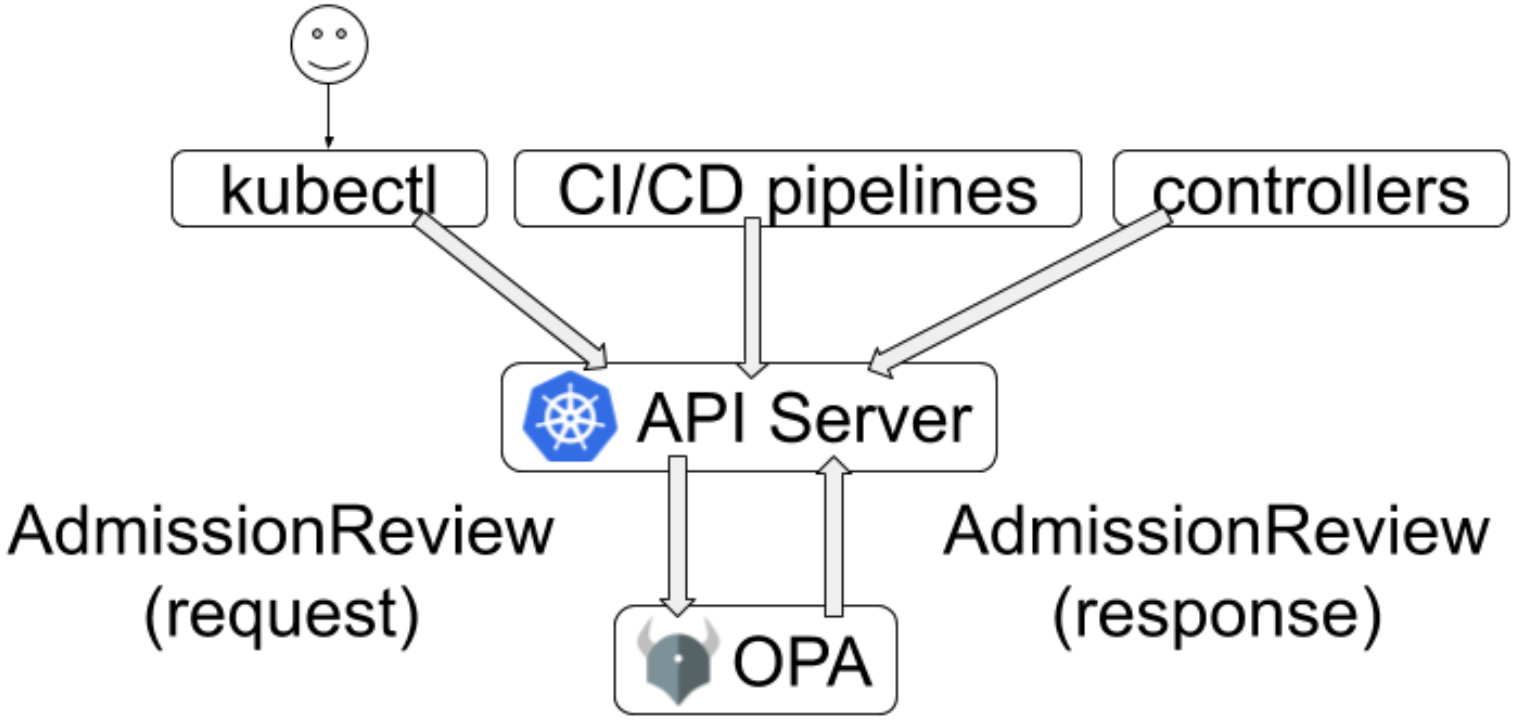
Some policy wants...

- Something that doesn’t require new “code”
- Easy for the team to learn
- Testable….

Enforcing Policy....

Open Policy Agent

It's Admission Control, I know this



policy.rego


```
package kubernetes.validating

deny[msg] {
  contains(input.request.object.metadata.labels.pants, "sweatpants")
  msg := "you can't sit with us"
}
```

Adventures In Rego

USE OF EXTERNAL INFORMATION SYSTEMS

information systems that are outside of the authorization boundary

Let’s restrict registries!

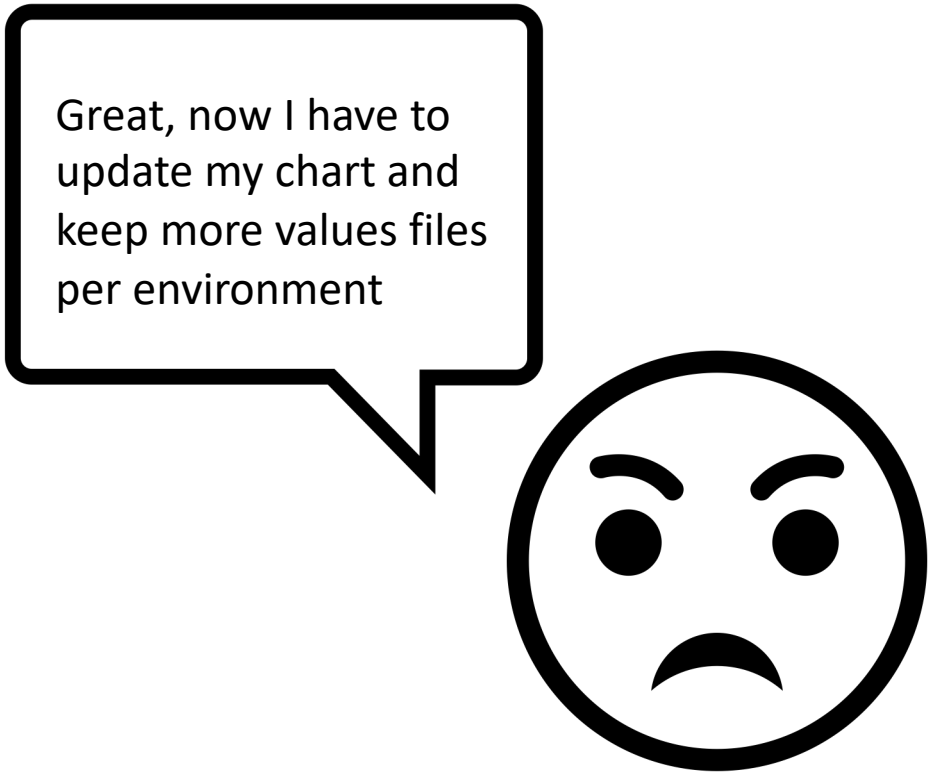

```
package kubernetes.admission

deny[msg] {
   input.request.kind.kind == "Pod"
   some i
   image := input.request.object.spec.containers[i].image
   not is_gov_image(image)
            msg := sprintf("Pod's container %q is not allowed to use image from non approved repo in gov", [image])
}

is_gov_image(image) {
            startswith(image, "vmware-is-awesome/")
}
```

Let’s restrict registries!

KubeCon CloudNativeCon North America 2020 Virtual

```
$ kubectl run mq-test --image=jeremyrickard/mq-test-cli:version4

kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.
deployment.apps/mq-test created

$ kubectl get events | grep openpolicyagent
47s         Warning   FailedCreate             replicaset/mq-test-79d5465bb4   Error creating: admission webhook "validating-webhook.openpolicyagent.org" denied the request: Pod's container "jeremyrickard/mq-test-cli:version4" is not allowed to use image from non approved repo in gov
```

But now…..


Great, now I have to update my chart and keep more values files per environment

Can we help…..

MutatingAdmissionWebhook

```
package kubernetes.admission

vdp_repo = "vmware-repo.io"
gov_repo = ”secrets.io”

patch[patchCode] {
   is_deployment_mutation_allowed
   some i
   image := input.request.object.spec.template.spec.containers[i].image
   updated_image_repo := update_public_image_repo(image)
   count(updated_image_repo) > 0
   update_path := concat("/", ["/spec/template/spec/containers", format_int(i, 10), "image"])
   patchCode := make_image_patch("replace", update_path, updated_image_repo)
}
```

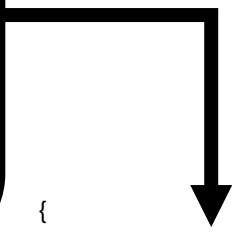
PCI Requirement 6: Develop and maintain secure systems and applications

6.1 - *Establish a process to identify security vulnerabilities, using reputable outside sources, and assign a risk ranking (e.g. “high,” “medium,” or “low”) to newly discovered security vulnerabilities.*

CVSS

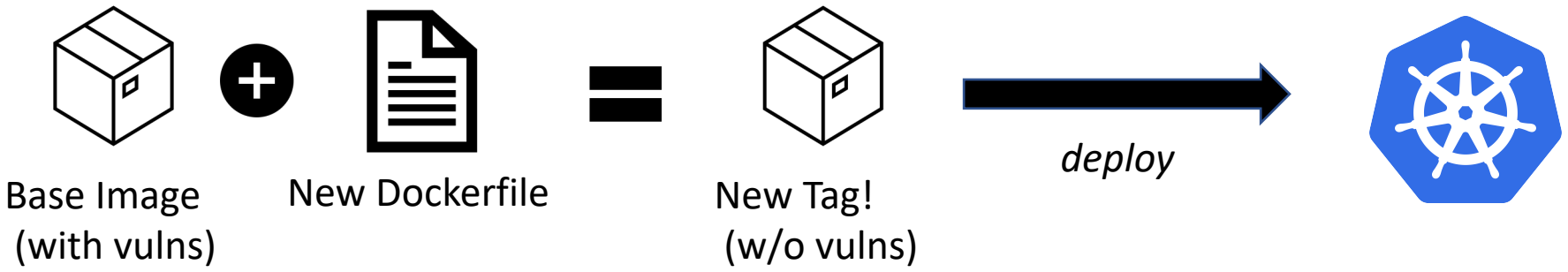
CVSS Score	Severity Level	Pass/Fail
7.0 through 10.0	High Severity	Fail
4.0 through 6.9	Medium Severity	Fail
0.0 through 3.9	Low Severity	Pass

```
$ twistlock iamge scan <redacted>

Scan results for image <redacted>:14213
sha256:8cf2ff8a024a37780af0d24d3408e7503225fbded1455c6f117b58dac319d8ae


Vulnerabilities found for image <redacted>:14213: total - 12, critical - 2, high - 3, medium
- 4, low - 3
```



```
{
 "id": "CVE-2019-17571",
 "cvss": 9.8,
 "vector":
"CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H",
 "description": "Included in Log4j 1.2 is a SocketServer…. ”
 "severity": "critical",
 "packageName": "log4j_log4j",
 "packageVersion": "1.2.17",
….
}
```

We made a process…

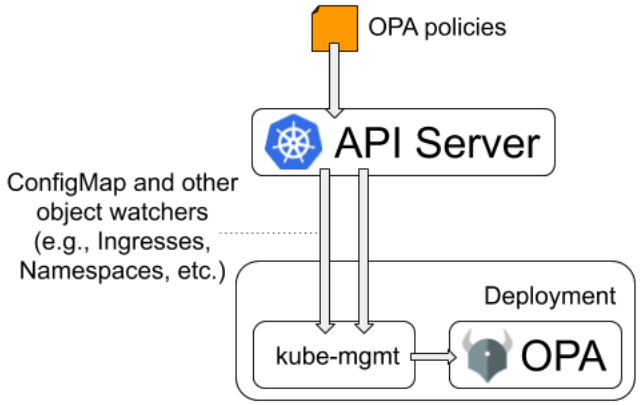


We automated…sorta

Policies can be loaded into OPA dynamically via ConfigMap objects using the kube-mgmt sidecar container.
The kube-mgmt sidecar container can also load any other Kubernetes object into OPA as JSON under data.
This lets you enforce policies that rely on an eventually consistent snapshot of the Kubernetes cluster as context.

How's that work?


```
$ kubectl get configmap vdp-images –n opa –o yaml
apiVersion: v1
data:
  images.json: |
    {"addon-resizer":"1.8.8-20201014-0200","admission-webhook":"v1.0.3-20201014-
0200","alertmanager":"v0.18.0-20201014-0200","am2jira-webhook":"v1.0-20201014-0200","authn-
webhook":"v1.0.1-20201014-0200",…….
```

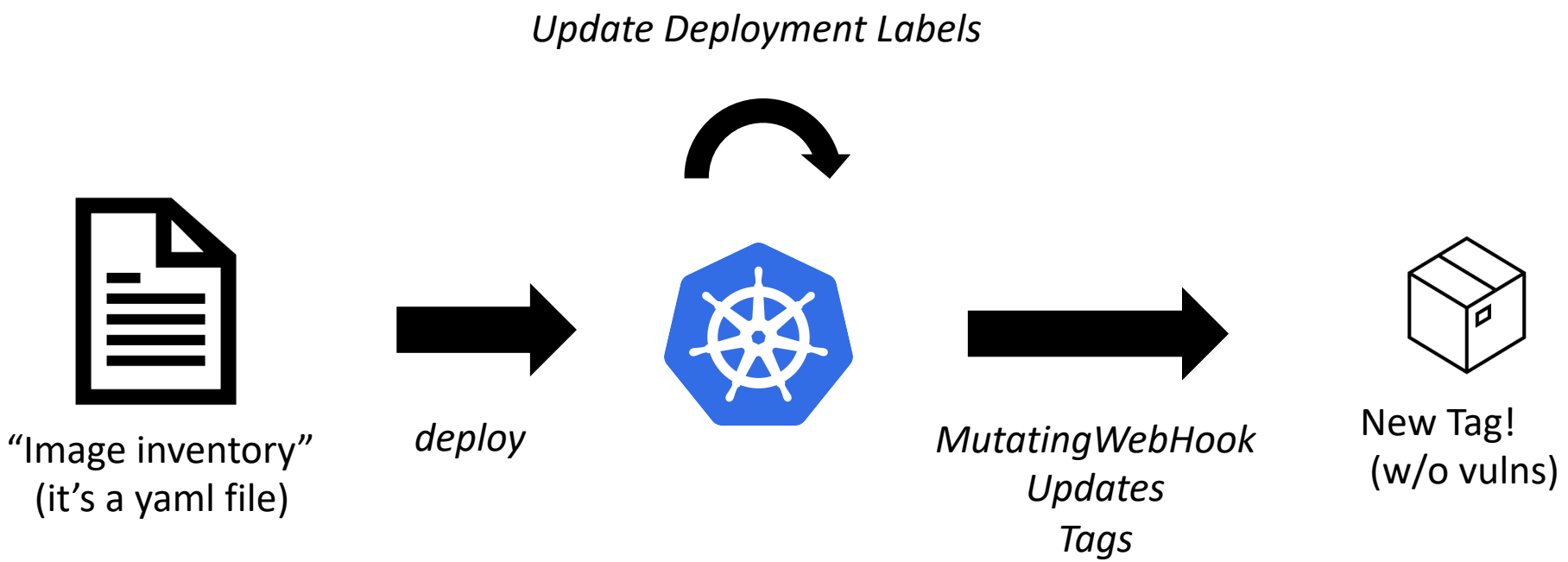


data.opa["vdp-images"]["images.json"]

More mutation!!

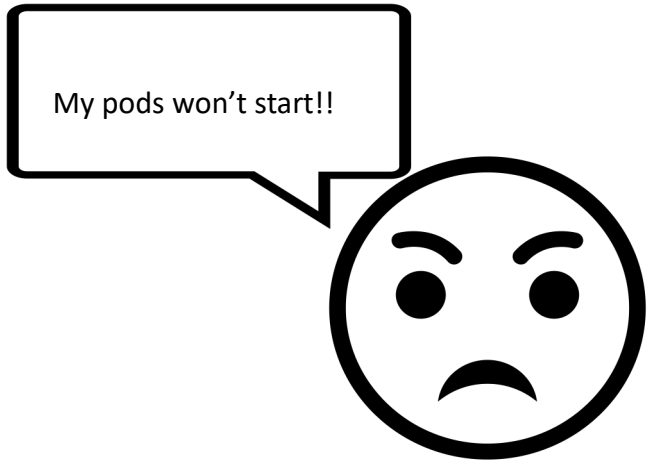

```
package kubernetes.admission

patch[patchCode] {
    some i
    inventory := data.opa["vdp-images"]["images.json"]
    updated_image := update_image_version(input.request.object.spec.initContainers[i].image, inventory)
    update_path := concat("/", ["/spec/containers", format_int(i, 10), "image"])
    patchCode := make_image_patch("replace", update_path, updated_image)
}
update_image_version(image_spec_path, inventory) = updated_image {
    image_ref_minus_sha := getRef(image_spec_path)
    image_tag = getTag(image_ref_minus_sha)
    image_inventory_tag := inventory[image_name]
    updated_image := replace(image_ref_minus_sha, image_tag, image_inventory_tag)
}
```

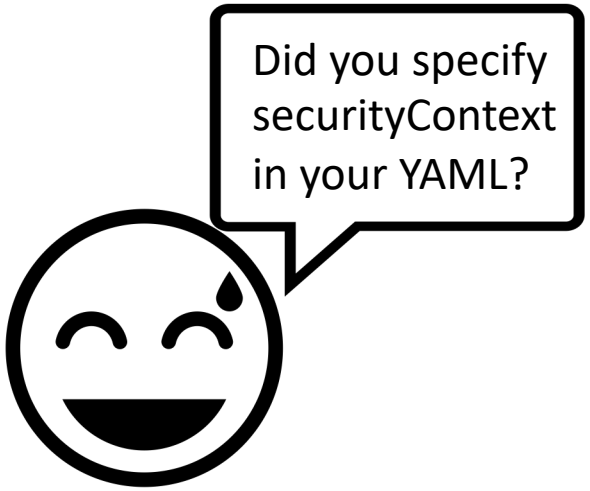
Update pipeline

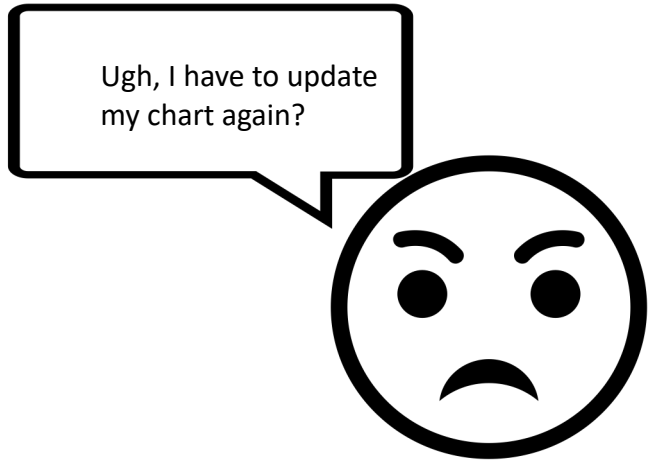


Enforce running as non-root

Pod Security Policies!





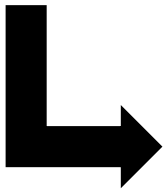


Hello again, Mutation.

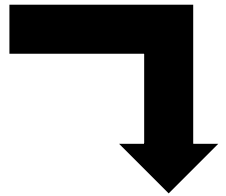

```
package kubernetes.admission

patch[patchCode] {
    is_mutation_allowed
    not input.request.object.spec.template.spec.securityContext.runAsUser
    patchCode = makeSecurityContextPatch("add", "runAsUser", 1000, "")
}

patch[patchCode] {
    is_mutation_allowed
    not input.request.object.spec.template.spec.securityContext.fsGroup
    patchCode = makeSecurityContextPatch("add", "fsGroup", 2000, "")
}
```

Recapping.

- Open Policy Agent is very flexible
  - Validation can get you pretty far
  - Mutation can get you even further



- Rego
  - Declarative nature makes policies easy to read
  - Pretty easy for team members to learn



- We were able to use it to balance
  - Security needs
  - User experience

Some thoughts…

Links!

https://www.fedramp.gov/fedramp-releases-high-baseline/
https://www.pcisecuritystandards.org/document_library

https://play.openpolicyagent.org/
https://www.openpolicyagent.org/docs/latest/kubernetes-tutorial/

Gatekeeper mutating webhook support:
https://github.com/open-policy-agent/gatekeeper/issues/588

Questions?


KEEP CLOUD NATIVE
EVERYWHERE
KubeCon
CloudNativeCon
North America 2020
Virtual